National Semiconductor

January 2002

# CR16HCS5/CR16HCS9/CR16MAR5/CR16MAS5 CR16MAS9/CR16MBR5/CR16MCS5/CR16MCS9 Family of 16-bit CAN-enabled CompactRISC Microcontrollers



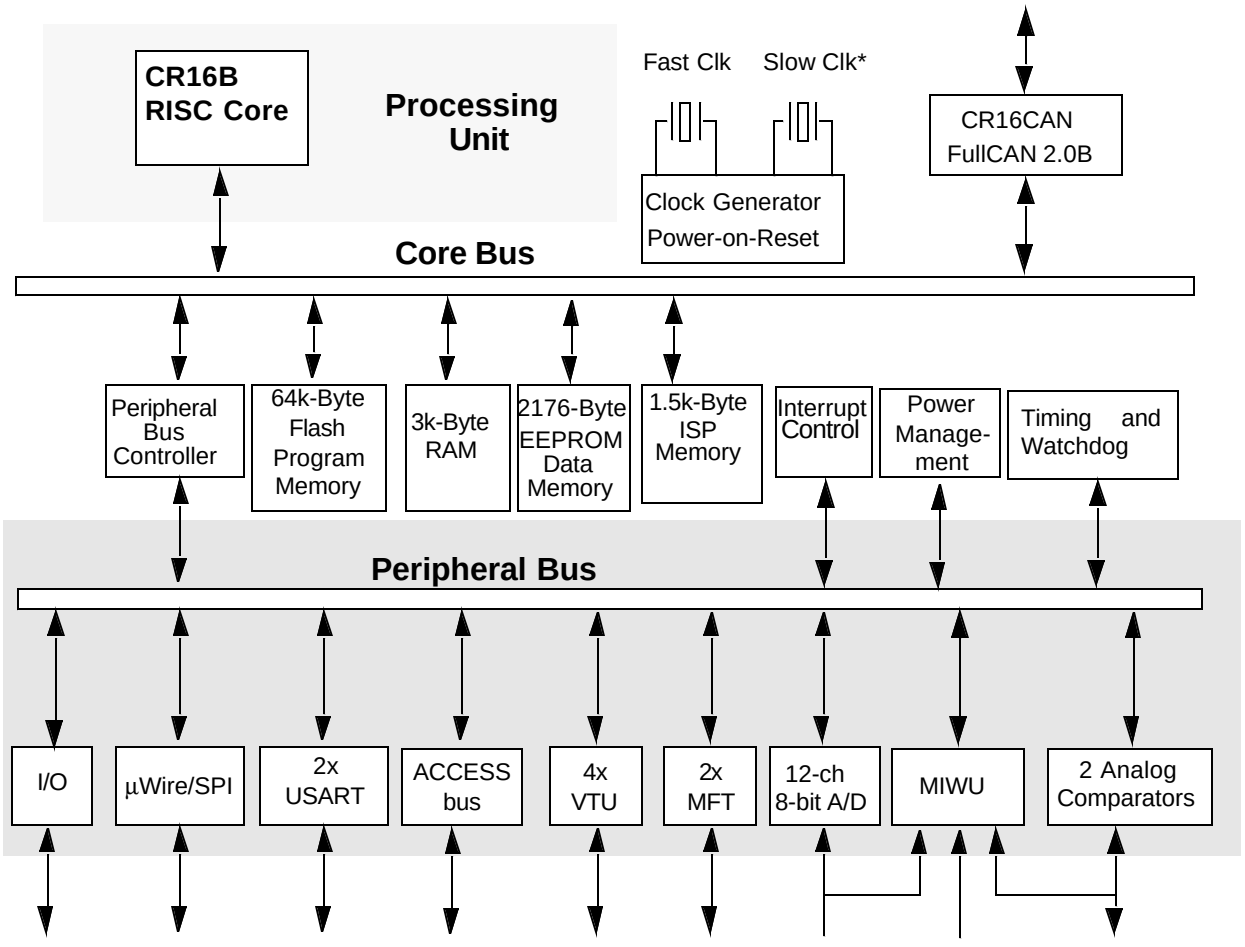
## 1.0 General Description

The family of 16-bit CompactRISC™ microcontroller is based on a Reduced Instruction Set Computer (RISC) architecture. The device operates as a complete microcomputer with all system timing, interrupt logic, flash program memory or ROM memory, RAM, EEPROM data memory, and I/O ports included on-chip. It is ideally suited to a wide range of embedded controller applications because of its high performance, on-chip integrated features and low power consumption resulting in decreased system cost.

The device offers the high performance of a RISC architecture while retaining the advantages of a traditional Complex Instruction Set Computer (CISC): compact code, on-chip memory and I/O, and reduced cost. The CPU uses a three-stage instruction pipeline that allows execution of up to one instruction per clock cycle, or up to 25 million instructions per second (MIPS) at a clock rate of 24 MHz.

The device contains a FullCAN class, CAN serial interface for low/high speed applications with 15 orthogonal message buffers, each supporting standard as well as extended message identifiers.

## Block Diagram




Please note that not all family members contain same peripheral modules and features.

# Table of Contents

## 1.0 General Description (Continued)

The device has up to 64K bytes of reprogrammable flash EEPROM program memory or ROM memory, 1.5K bytes of flash EEPROM In-System-Programming memory, 3K bytes of static RAM, 2K bytes of non-volatile EEPROM data memory and 128 bytes with high endurance, two USARTs, two 16-bit multi-function timers, one SPI/MICROWIRE-PLUS™ serial interface, a 12-channel A/D converter, two analog comparators, WATCHDOG™ protection mechanism, and up to 56 general-purpose I/O pins.

The device operates with a high-frequency crystal as the main clock source and either the prescaled main clock source or with a low frequency (32.768 kHz) oscillator in Power Save mode. The device supports several Power Save modes which are combined with multi-source interrupt and wake-up capabilities.

This device also has a Versatile Timer Unit (VTU) with four timer sub-systems, a CAN interface, and ACCESS.bus synchronous serial bus interface.

Powerful cross-development tools are available from National Semiconductor and third party suppliers to support the development and debugging of application software for the device. These tools let you program the application software in C and are designed to take full advantage of the CompactRISC architecture.

In the following text, device is always referred to the family of 16-bit CAN-enabled CompactRISC Microtroller.

## 2.0 Features

- CPU Features
  - — Fully static core, capable of operating at any rate from 0 to 24 MHz (4 MHz minimum in active mode)
  - — 50 ns instruction cycle time with a 20 MHz external clock frequency
  - — Multi-source vectored interrupts (internal, external, and on-chip peripheral)
  - — Dual clock and reset
- On-chip power-on reset
- On-Chip Memory
  - — Up to 64K bytes flash EEPROM program memory; can be programmed, erased, and reprogrammed by software (100K cycles)
  - — 3K bytes of static RAM data memory
  - — For flash program memory devices, 1.5k bytes flash EEPROM memory is available to store boot loader code (100K cycles)
  - — 2K bytes of non-volatile EEPROM data memory with low endurance (25K cycles) and 128 bytes with high endurance (100K cycles)
- On-Chip Peripherals
  - — Two Universal Synchronous/Asynchronous Receiver/Transmitter (USART) devices
  - — Programmable Idle Timer and real-time clock (T0)
  - — Two dual 16-bit multi-function timers (MFT1 and MFT2)
  - — 8/16-bit SPI/MICROWIRE-PLUS serial interface
  - — 12-channel, 8-bit Analog-to-Digital (A/D) converter with external voltage reference, programmable sample-and-hold delay, and programmable conversion frequency
  - — ACCESS.bus synchronous serial bus
  - — FullCAN interface with 15 message buffers complaint to CAN specification 2.0B active
  - — Versatile Timer Unit with four subsystems (VTU)
  - — Two analog comparators
  - — Integrated WATCHDOG logic
- I/O Features
  - — Up to 56 general-purpose I/O pins (shared with on-chip peripheral I/O pins)
  - — Programmable I/O pin characteristics: TRI-STATE output, push-pull output, weak pull-up input, high-impedance input
  - — Schmitt triggers on general purpose inputs
- Power Supply
  - — 4.5V to 5.5V single-supply operation
- Temperature Range
  - — –40°C to +85°C
  - — –40°C to +125°C
- Development Support
  - — Real-time emulation and full program debug capabilities available
  - — CompactRISC tools provide C programming and debugging support

**CR16 CompactRISC Microcontroller with CAN Interface Family Selection Guide**

**Programmable devices**

| NSID | Speed (MHz) | Flash/ (kByte) | EEPROM Data Memory (Bytes) | SRAM (kBytes) | USART | Timer | I/Os | Temp. Range | Peripherals | Package Type |
|---|---|---|---|---|---|---|---|---|---|---|
| CR16MCS9VJEx | 16 | 64 | 2176 | 3 | 2 | 2MFT, VTU | 56 | E, I | ADC, CAN, Comparators | 80 PQFP |
| CR16MAS9VJEx | 24 | 64 | | 3 | 2 | 2MFT, VTU | 56 | E, I | ADC, CAN, Comparators | 80 PQFP |

**Factory Programmed devices**

| NSID | Speed (MHz) | Flash/ (KByte) | EEPROM Data Memory (Bytes) | SRAM (kBytes) | USART | Timer | I/Os | Temp. Range | Peripherals | Package Type |
|---|---|---|---|---|---|---|---|---|---|---|
| CR16MCS9VJExy | 16 | 64 | 2176 | 3 | 2 | 2MFT, VTU | 56 | E, I | ADC, CAN, Comparators | 80 PQFP |
| CR16MCS9VJExy | 24 | 64 | 2176 | 3 | 2 | 2MFT, VTU | 56 | E, I | ADC, CAN, Comparators | 80 PQFP |

**ROM devices**

| NSID | Speed (MHz) | Flash/ ROM (KByte) | EEPROM Data Memory (Bytes) | SRAM (kBytes) | USART | Timer | I/Os | Temp. Range | Peripherals | Package Type |
|---|---|---|---|---|---|---|---|---|---|---|
| CR16HCS9VJEx | 24 | 64 | 2176 | 3 | 2 | 2MFT, VTU | 56 | E, I | ADC, CAN, Comparators | 80 PQFP |
| CR16MCS5VJEx | 24 | 64 | 2176 | 3 | 2 | 2MFT, VTU | 56 | E, I | ADC, CAN, Comparators | 80 PQFP |
| CR16MBR5VJEx | 24 | 32 | 2176 | 3 | 2 | 2MFT, VTU | 56 | E, I | ADC, CAN, Comparators | 80 PQFP |
| CR16MAR5VJEx | 24 | 32 | | 3 | 2 | 2MFT, VTU | 56 | E, I | CAN, | 80 PQFP |
| CR16MAS5VJEx | 24 | 64 | | 3 | 2 | 2MFT, VTU | 56 | E, I | CAN, | 80 PQFP |

**Note:**

- Suffix x in the NSID is defined below:

Temperature Ranges:

I = Industrial -40°C to +85°C is represented when x is 8
E = Extended -40°C to +125°C is represented when x is 7

- Suffix y in the NSID defines the ROM code.

**Note:** All devices contains Access.bus (ACB), Clock and Reset, MICROWIRE/API, Multi-Input Wake-Up (MIWU), Power Management (PMM), and the Real-Time Timer and Watchdog (TWM) modules. Access.bus is compatible with I2C bus offered by Philips Semiconductor.

**CR16 CompactRISC Microcontroller with CAN Interface Family Devices**

National Semiconductor currently offers a variety of the CR16 CompactRISC Microcontrollers with CAN interface. The CR16MCS offer complete functionality in an 80-pin PQFP package.

# 3.0 Device Overview

The devices are complete microcomputers with all system timing, interrupt logic, program memory, data memory, and I/O ports included on-chip, making it well-suited to a wide range of embedded controller applications.

## 3.1 CR16B CPU CORE

The device uses a CR16B CPU core module. This is the same core used in other CompactRISC family member designs, like DECT or GSM chipsets.

The high performance of the CPU core results from the implementation of a pipelined architecture with a two-bytes-per-cycle pipelined system bus. As a result, the CPU can support a peak execution rate of one instruction per clock cycle.

Compared with conventional RISC processors, the device differs in the following ways:

— The CPU core can use on-chip rather than external memory. This eliminates the need for large and complex bus interface units.
— Most instructions are 16 bits, so all basic instructions are just two bytes long. Additional bytes are sometimes required for immediate values, so instructions can be two or four bytes long.
— Non-aligned word access is allowed. Each instruction can operate on 8-bit or 16-bit data.
— The device is designed to operate with a clock rate in the 10 to 24 MHz range rather than 100 MHz or more. Most embedded systems face EMI and noise constraints that limit clock speed to these lower ranges. A lower clock speed means a simpler, less costly silicon implementation.
— The instruction pipeline uses three stages. A smaller pipeline eliminates the need for costly branch prediction mechanisms and bypass registers, while maintaining adequate performance for typical embedded controller applications.

For more information, please refer to the CR16B Programmer's Reference Manual, Literature #: 633150.

## 3.2 MEMORY

The CompactRISC architecture supports a uniform linear address space of 2 megabytes. The device implementation of this architecture uses only the lowest 128K bytes of address space. Four types of on-chip memory occupy specific intervals within this address space:

- 64K bytes of flash EEPROM program memory (100K cycles)
- 48K bytes ROM programm memory version available also (100K cycles)
- 3K bytes of static RAM
- 2K bytes of EEPROM data memory with low endurance (25K cycles)
- 128 bytes with high endurance (100K cycles)
- 1.5K bytes flash EEPROM memory for ISP code

The 3K bytes of static RAM are used for temporary storage of data and for the program stack and interrupt stack. Read and write operations can be byte-wide or word-wide, depending on the instruction executed by the CPU. Each memory access requires one clock cycle; no wait cycles or hold cycles are required.

There are two types of flash EEPROM data memory storage. The 2K bytes of EEPROM data memory with low endurance (25K cycles) and 128 bytes of flash EEPROM data memory with high endurance (100K cycles) are used for non-volatile storage of data, such as configuration settings entered by the end-user.

The 64K bytes of flash EEPROM program memory are used to store the application program. It has security features to prevent unintentional programming and to prevent unauthorized access to the program code. This memory can be programmed with a device external programming unit or with the device installed in the application system (in-system programming).

There is a factory programmed boot memory used to store In-System-Programming (ISP) code. (This code allows programming of the program memory via one of the USART interfaces in the final application.)

For flash EEPROM program and data memory, the device internally generates the necessary voltages for programming. No additional power supply is required.

## 3.3 INPUT/OUTPUT PORTS

The device has 56 software-configurable I/O pins, organized into seven 8-pin ports called Port B, Port C, Port F, Port G, Port H, Port I, and Port L. Each pin can be configured to operate as a general-purpose input or general-purpose output. In addition, many I/O pins can be configured to operate as a designated input or output for an on-chip peripheral module such as the USART, timer, A/D converter, or MICROWIRE/SPI interface.

The I/O pin characteristics are fully programmable. Each pin can be configured to operate as a TRI-STATE output, push-pull output, weak pull-up input, or high-impedance input.

## 3.4 BUS INTERFACE UNIT

The Bus Interface Unit (BIU) controls the interface between the on-chip modules to the internal core bus. It determines the configured parameters for bus access (such as the number of wait states for memory access) and issues the appropriate bus signals for each requested access.

The BIU uses a set of control registers to determine how many wait states and hold states are to be used when accessing flash EEPROM program memory, ISP memory and the I/O area (Port B and Port C). Upon start-up the configuration registers are set for slowest possible memory access. To achieve fastest possible program execution, appropriate values should be programmed. These settings vary with the clock frequency and the type of on-chip device being accessed.

### 3.5 INTERRUPTS

The Interrupt Control Unit (ICU31L) receives interrupt requests from internal and external sources and generates interrupts to the CPU. An interrupt is an event that temporarily stops the normal flow of program execution and causes a separate interrupt service routine to be executed. After the interrupt is serviced, CPU execution continues with the next instruction in the program following the point of interruption.

Interrupts from the timers, USARTs, MICROWIRE/SPI interface, multi-input wake-up, and A/D converter are all maskable interrupts; they can be enabled or disabled by the software. There are 32 of these maskable interrupts, organized into 32 predetermined levels of priority.

The highest-priority interrupt is the Non-Maskable Interrupt ($\overline{\text{NMI}}$), which is generated by a signal received on the $\overline{\text{NMI}}$ input pin.

### 3.6 MULTI-INPUT WAKE-UP

The Multi-Input Wake-Up (MIWU16) module can be used for either of two purposes: to provide inputs for waking up (exiting) from the HALT, IDLE, or Power Save mode; or to provide general-purpose edge-triggered maskable interrupts from external sources. This 16-channel module generates four programmable interrupts to the CPU based on the signals received on its 16 input channels. Channels can be individually enabled or disabled, and programmed to respond to positive or negative edges.

### 3.7 DUAL CLOCK AND RESET

The Dual Clock and Reset (CLK2RES) module generates a high-speed main system clock from an external crystal network. It also provides the main system reset signal and a power-on reset function.

This module also generates a slow system clock (32.768 kHz) from another external crystal network. The slow clock is used for operating the device in power-save mode. Without a 32.768kHz external crystal network, the low speed system clock can be derived from the high speed clock by a prescaler.

Also, two independent clocks divided down from the high speed clock are available on output pins.

### 3.8 POWER MANAGEMENT

The Power Management Module (PMM) improves the efficiency of the device by changing the operating mode and therefore the power consumption according to the required level of activity.

The device can operate in any of four power modes:

- — Active: The device operates at full speed using the high-frequency clock. All device functions are fully operational.
- — Power Save: The device operates at reduced speed using the slow clock. The CPU and some modules can continue to operate at this low speed.
- — IDLE: The device is inactive except for the Power Management Module and Timing and Watchdog Module, which continue to operate using the slow clock.
- — HALT: The device is inactive but still retains its internal state (RAM and register contents).

### 3.9 MULTI-FUNCTION TIMER

The Multi-Function Timer (MFT16) module contains two independent timer/counter units called MFT1 and MFT2, each containing a pair of 16-bit timer/counter registers. Each timer/counter unit can be configured to operate in any of the following modes:

- — Processor-Independent Pulse Width Modulation (PWM) mode, which generates pulses of a specified width and duty cycle, and which also provides a general-purpose timer/counter.
- — Dual Input Capture mode, which measures the elapsed time between occurrences of external events, and which also provides a general-purpose timer/counter.
- — Dual Independent Timer mode, which generates system timing signals or counts occurrences of external events.
- — Single Input Capture and Single Timer mode, which provides one external event counter and one system timer.

### 3.10 VERSATILE TIMER UNIT

The Versatile Timer Unit (VTU) module contains four independent timer subsystems, each operating in either dual 8-bit PWM configuration, as a single 16-bit PWM timer, or a 16-bit counter with two input capture channels. Each of the four timer subsystems offer an 8-bit clock prescaler to accommodate a wide range of frequencies.

### 3.11 REAL-TIME TIMER AND WATCHDOG

The Timing and Watchdog Module (TWM) generates the clocks and interrupts used for timing periodic functions in the system. It also provides Watchdog protection against software errors. The module operates on the slow system clock.

The real-time timer can generate a periodic interrupt to the CPU at a software-programmed interval. This can be used for real-time functions such as a time-of-day clock. The real-time timer can trigger a wake-up condition from power-save mode via the Multi-Input Wake-Up module.

The Watchdog is designed to detect program execution errors such as an infinite loop or a “runaway” program. Once Watchdog operation is initiated, the application program must periodically write a specific value to a Watchdog register, within specific time intervals. If the software fails to do so, a Watchdog error is triggered, which resets the device.

### 3.12 USART

The USART supports a wide range of programmable baud rates and data formats, and handles parity generation and several error detection schemes. The baud rate is generated on-chip, under software control.

There are two independent USARTs in the device and they offer a wake-up condition from the power-save mode via the Multi-Input Wake-Up module.

### 3.13 MICROWIRE/SPI

The MICROWIRE/SPI (MWSPI) interface module supports synchronous serial communications with other devices that conform to MICROWIRE or Serial Peripheral Interface (SPI) specifications. It supports 8-bit and 16-bit data transfers.

The MICROWIRE interface allows several devices to communicate over a single system consisting of four wires: serial in, serial out, shift clock, and slave enable. At any given time, the MICROWIRE interface operates as the master or a slave. The support supports the full set of slave select for multi-slave implementation.

In master mode, the shift clock is generated on chip under software control. In slave mode, a wake-up out of power-save mode is triggered via the Multi-Input Wake-Up module.

### 3.14 CR16CAN

The CR16CAN device contains a FullCAN class, CAN serial bus interface for applications that require a high speed (up to 1MBits per second) or a low speed interface with CAN bus master capability. The data transfer between CAN and the CPU is established by 15 memory mapped message buffers, which can be individually configured as receive or transmit buffers. An incoming message is filtered by two masks, one for the first 14 message buffers and another one for the 15th message buffer to provide a basic CAN path. A priority decoder allows any buffer to have the highest or lowest transmit priority. Remote transmission requests can be processed automatically by automatic reconfiguration to a receiver after transmission or by automated transmit scheduling upon reception. In addition, a time stamp counter (16-bits wide) is provided to support real time applications.

The CR16CAN device is a fast core bus peripheral, which allows single cycle byte or word read/write access. A set of diagnostic features (such as loopback, listen only, and error identification) support the development with the CR16CAN module and provide a sophisticated error management tool.

The CR16CAN receiver can trigger a wake-up condition out of the power-save modes via the Multi-Input Wake-Up module.

### 3.15 ACCESS.BUS INTERFACE

The ACCESS.bus interface module (ACB) is a two-wire serial interface with the ACCESS.bus physical layer. It is also compatible with Intel's System Management Bus (SMBus) and Philips' $I^2C$ bus. The ACB module can be configured as a bus master or slave, and can maintain bi-directional communications with both multiple master and slave devices.

The ACCESS.bus receiver can trigger a wake-up condition out of the power-save modes via the Multi-Input Wake-Up module.

### 3.16 A/D CONVERTER

The A/D Converter (ADC) module is a 12-channel multiplexed-input analog-to-digital converter. The A/D Converter receives an analog voltage signal on an input pin and converts the analog signal into an 8-bit digital value using successive approximation. The CPU can then read the result from a memory-mapped register. The module supports four automated operating modes, providing single-channel or 4-channel operation in single or continuous mode.

The device has a separate pin, Vref, for the A/D reference voltage.

### 3.17 ANALOG COMPARATORS

The Dual Analog Comparator (ACMP2) module contains two independent analog comparators with all necessary control logic. Each comparator unit compares the analog input voltages applied to two input pins and determines which voltage is higher. The CPU uses a memory-mapped register to control the comparator and to obtain the comparison results. The comparison result can also be applied to comparator output pins.

### 3.18 DEVELOPMENT SUPPORT

A powerful cross-development tool set is available from National Semiconductor and third parties to support the development and debugging of application software for the CR16MCS9. The tool set lets you program the application software in C and is designed to take full advantage of the CompactRISC architecture.

There are In-System Emulation (ISE) devices available for the device from iSYSTEM™, as well as lower-cost evaluation boards. See your National Semiconductor sales representative for current information on availability and features of emulation equipment and evaluation boards.

## 4.0 Device Pinouts

**Table 1 Package Pin Assignments**

| Pin Name | Alternate Function(s) | Pin Number | Type |
|---|---|---|---|
| PH4 | $\overline{\text{MWCS}}$ | 1 | I/O |
| PH5 | MD1D0 | 2 | I/O |
| PH6 | MD0D1 | 3 | I/O |
| PH7 | MSK | 4 | I/O |
| PB0 | D0 | 5 | I/O |
| PB1 | D1 | 6 | I/O |
| PB2 | D2 | 7 | I/O |
| PB3 | D3 | 8 | I/O |
| PB4 | D4 | 9 | I/O |
| PB5 | D5 | 10 | I/O |
| PB6 | D6 | 11 | I/O |
| PB7 | D7 | 12 | I/O |
| $\overline{\text{ENV0}}$/CLKOUT1 | | 13 | I/O |
| SDA | | 14 | I/O |
| SCL | | 15 | I/O |
| GND | | 16 | PWR |
| Vcc | | 17 | PWR |
| GND | | 18 | PWR |
| CANTx | | 19 | O |
| CANRx | | 20 | I |
| PC0 | D8 | 21 | I/O |
| PC1 | D9 | 22 | I/O |
| PC2 | D10 | 23 | I/O |
| PC3 | D11 | 24 | I/O |
| PC4 | D12 | 25 | I/O |
| PC5 | D13 | 26 | I/O |
| PC6 | D14 | 27 | I/O |
| PC7 | D15 | 28 | I/O |
| PG7 | CKX1 | 29 | I/O |
| PG6 | TDX1 | 30 | I/O |
| PG5 | RDX1 | 31 | I/O |
| PG4 | TIO6 | 32 | I/O |
| PG3 | TIO5 | 33 | I/O |
| PG2 | CKX2 | 34 | I/O |
| PG1 | TDX2 | 35 | I/O |
| PG0 | RDX2 | 36 | I/O |
| CLKOUT2 | | 37 | O |
| $\overline{\text{ENV1}}$/CLK[1] | | 37 | I/O |
| PF7 | TIO4 | 38 | I/O |
| PF6 | TIO3 | 39 | I/O |
| PF5 | T2B | 40 | I/O |
| PF4 | T2A | 41 | I/O |
| PF3 | TIO2 | 42 | I/O |
| PF2 | TIO1 | 43 | I/O |
| PF1 | TIB | 44 | I/O |

**Table 1 Package Pin Assignments**

| Pin Name | Alternate Function(s) | Pin Number | Type |
|---|---|---|---|
| PF0 | TIA | 45 | I/O |
| NMI | | 46 | I |
| X1CKO | | 47 | O |
| X1CKI | | 48 | I |
| GND | | 49 | PWR |
| Vcc | | 50 | PWR |
| GND | | 51 | PWR |
| X2CKO | | 52 | O |
| X2CKI | | 53 | I |
| $\overline{\text{RESET}}$[2] | | 54 | I |
| PI0 | ACH0[3] | 55 | I/O |
| PI1 | ACH1[3] | 56 | I/O |
| PI2 | ACN2[3] | 57 | I/O |
| PI3 | ACH3[3] | 58 | I/O |
| PI4 | ACH4[3] | 59 | I/O |
| PI5 | ACH5[3] | 60 | I/O |
| PI6 | ACH6[3] | 61 | I/O |
| PI7 | ACH7[3] | 62 | I/O |
| Vref | | 63 | PWR |
| AGND | | 64 | PWR |
| AVcc | | 65 | PWR |
| PH0 | ACH8[3], WUI4 | 66 | I/O |
| PH1 | ACH9[3], WUI5 | 67 | I/O |
| PH2 | ACH10[3], WUI6 | 68 | I/O |
| PH3 | ACH11[3], WUI7 | 69 | I/O |
| GND | | 70 | PWR |
| Vcc | | 71 | PWR |
| GND | | 72 | PWR |
| PL0 | COMP1N[3], WUI0 | 73 | I/O |
| PL1 | COMP1P[3], WUI1 | 74 | I/O |
| PL2 | COMP1O, WUI2 | 75 | I/O |
| PL3 | COMP2O, WUI3 | 76 | I/O |
| PL4 | COMP2P[3] | 77 | I/O |
| PL5 | COMP2N[3] | 78 | I/O |
| PL6 | TIO7 | 79 | I/O |
| PL7 | TIO8 | 80 | I/O |

**Note 1:** The $\overline{\text{ENV0}}$ and $\overline{\text{ENV1}}$ pins each have a weak pull-up to keep the input from floating.
**Note 2:** The $\overline{\text{RESET}}$ input has a weak pulldown.
**Note 3:** These functions are always enabled, due to the direct low-impedance path to these pins.

## 4.1 PIN DESCRIPTION

The following is a brief description of all device pins.

Some pins have alternate functions which may be enabled. These pins can be individually configured as general purpose pins, even when the module they belong to is enabled.

**Table 2 Input Pins**

| Signal | Type | Active | Pin (* for a shared pin) | Function |
|---|---|---|---|---|
| X1CKI | OSC | High | | Main oscillator clock input. |
| X2CKI | OSC | High | | 32kHz oscillator clock input. |
| $\overline{\text{RESET}}$ | CMOS | Low | | Chip general reset pin. Schmitt trigger input, asynchronous. |
| $\overline{\text{ISE}}$ | CMOS | Low | | Interrupt input for development system. |
| T1B | CMOS | Prog. | * | Timer 1 input B. Shares pin with I/O port pin PF1. |
| T2B | CMOS | Prog. | * | Timer 2 input B. Shares pin with I/O port pin PF5. |
| RDX1 | CMOS | High | * | USART 1 receive data input. Shares pin with I/O port pin PG5. |
| RDX2 | CMOS | High | * | USART 2 receive data input. Shares pin with I/O port pin PG0. |
| ACH0 | Analog | | * | A2D converter channel 0. Shares pin with I/O port pin PI0 |
| ACH1 | Analog | | * | A2D converter channel 1. Shares pin with I/O port pin PI1 |
| ACH2 | Analog | | * | A2D converter channel 2. Shares pin with I/O port pin PI2 |
| ACH3 | Analog | | * | A2D converter channel 3. Shares pin with I/O port pin PI3 |
| ACH4 | Analog | | * | A2D converter channel 4. Shares pin with I/O port pin PI4 |
| ACH5 | Analog | | * | A2D converter channel 5. Shares pin with I/O port pin PI5 |
| ACH6 | Analog | | * | A2D converter channel 6. Shares pin with I/O port pin PI6 |
| ACH7 | Analog | | * | A2D converter channel 7. Shares pin with I/O port pin PI7 |
| ACH8 | Analog | | * | A2D converter channel 8. Shares pin with I/O port pin PH0 |
| ACH9 | Analog | | * | A2D converter channel 9. Shares pin with I/O port pin PH1 |
| ACH10 | Analog | | * | A2D converter channel 10. Shares pin with I/O port pin PH2 |
| ACH11 | Analog | | * | A2D converter channel 11. Shares pin with I/O port pin PH3 |
| $\overline{\text{MWCS}}$ | CMOS | Low | * | SPI/MICROWIRE slave select. Shares pin with I/O port pin PH4. |
| $\overline{\text{NMI}}$ | CMOS | Low | | External non-maskable interrupt. |
| $\overline{\text{ENV0}}$ | CMOS | Low | * | Strap to select operating environment. |
| $\overline{\text{ENV1}}$ | CMOS | Low | * | Strap pin to select operating environment. |
| $\overline{\text{ENV2}}$ | CMOS | Low | | Strap pin to select operating environment. |
| $\overline{\text{CANRx}}$ | CMOS | High | | CAN receive data input. |

**Table 3 Output Pins**

| Signal | Type | Active | Pin (* for a shared pin) | Function |
|---|---|---|---|---|
| X1CKO | OSC | High | | Main oscillator clock output. |
| X2CKO | OSC | High | | 32kHz oscillator clock output. |
| CLK | CMOS | High | * | External reference clock for development environment (shared with $\overline{\text{ENV1}}$). |
| CLKOUT 1 | CMOS | High | * | Clock output generated through prescaler (shared with $\overline{\text{ENV0}}$). |
| CLKOUT 2 | CMOS | High | * | Clock output generated through prescaler (shared with $\overline{\text{ENV1}}$). |

**Table 3 Output Pins**

| Signal | Type | Active | Pin (* for a shared pin) | Function |
|---|---|---|---|---|
| TDX1 | CMOS | High | * | USART 1 transmit data output (shared with PG6). |
| TDX2 | CMOS | High | * | USART 2 transmit data output (shared with PG1). |
| CANTx | CMOS | High | | CAN output. |

**Table 4 Input/Output Pins**

| Signal | Type | Active | Pin (* for a shared pin) | Function |
|---|---|---|---|---|
| PF[0:7] | CMOS | High | * | Generic I/O port. Shared with T1A, T1B, TIO1, TIO2, T2A, T2B, TIO3, TIO4. |
| PG[0:7] | CMOS | High | * | Generic I/O port. Shared with RDX2, TDX2, CKX2, TIO5, TIO6, RDX1, TDX1, CKX1. |
| PB[0:7] | CMOS | High | * | Generic I/O port. |
| PC[0:7] | CMOS | High | * | Generic I/O port. |
| PL[0:7] | CMOS | High | * | Generic I/O port. Shared with 6 comparator pins, MIWU16 on PL0:3. |
| PH[0:7] | CMOS | High | * | Generic I/O port. Shared with ADC input channels 8-11, $\overline{\text{MWCS}}$, MDIDO, MDODI, MSK; MIWU16 on PH4:7. |
| PI[0:7] | CMOS | High | * | Generic I/O port. Shared with ADC input channels 0-7. |
| T1A | CMOS | Prog | * | Timer 1 input A. Shared with I/O port pin PF0. |
| T2A | CMOS | Prog | * | Timer 2 input A. Shared with I/O port pin PF4. |
| TIO[0:7] | CMOS | Prog | * | Versatile timer unit I/Os. Shared with PF2:3, PF6:7, PG3:4, PL6:7. |
| MDIDO | CMOS | High | * | Master In/Slave Out port: SPI/Microwire. Shared with I/O pin PH5, |
| MDODI | CMOS | High | * | Master Out/Slave In port: SPI/Microwire. Shared with I/O pin PH6. |
| MSK | CMOS | Prog | * | SPI/Microwire clock. Shared with I/O pin PH7. |
| CKX1 | CMOS | High | * | USART 1 clock. Shared with I/O pin PG7. |
| CKX2 | CMOS | High | * | USART 2 clock. Shared with I/O pin PG2 |
| SCL | CMOS | High | | ACCESS.bus clock I/O. |
| SDA | CMOS | High | | ACCESS.bus data I/O. |

**Table 5 Power Supply**

| Signal | Function |
|---|---|
| Vcc | Main digital power supply (4 total). |
| Vref | Voltage reference supply for analog to digital converter. |
| AVcc | Analog power supply for analog/digital converter. |
| AGND | Analog reference ground supply. |
| GND | Main digital reference ground (8 total). |

# 5.0 System Configuration

The device has two input pins, $\overline{ENV0}$ and $\overline{ENV1}$, which are used to specify the operating environment of the device upon reset. There are also two system configuration registers, called the Module Configuration (MCFG) register and the Module Status (MSTAT) register.

## 5.1 $\overline{ENV0}$ AND $\overline{ENV1}$ PINS

Upon reset, the operating mode of the device is determined by the state of the $\overline{ENV0}$ and $\overline{ENV1}$ input pins, as indicated in Table6.

**Table 6 Operating Environment Selection**

| $\overline{ENV1}$ | $\overline{ENV0}$ | Operating Environment |
|---|---|---|
| 0 | 0 | Test Mode Flash Memory |
| 0 | 1 | Test Mode |
| 1 | 0 | In-System-Programming mode (ISP) |
| 1 | 1 | Internal ROM enabled Mode (IRE), if program memory is not empty; or ISP-Mode, if program memory is empty |

In the case where the $\overline{ENV1}$ and $\overline{ENV0}$ pins are both high, the reset algorithm looks at the FLCTRL2.EMPTY bit to determine whether the program memory is empty, and sets the operating mode accordingly.

The $\overline{ENV0}$ and $\overline{ENV1}$ pins have on-chip pull-up devices that are enabled during reset while the pins are being sampled. Therefore, if they are left unconnected, the inputs are considered high and the normal operating mode (IRE-Mode) is selected and the CPU starts to execute code at address 0. To enter any other operating mode, the external hardware must drive the appropriate input low.

In the case where the ISP-Mode is selected, the chip starts executing the ISP code residing in the on-chip ISP-Memory area.

The test modes are Reserved for factory testing and for external programming of the flash EEPROM program memory. They should not be invoked otherwise.

## 5.2 MODULE CONFIGURATION (MCFG) REGISTER

The MCFG register is a byte-wide, read/write register that sets the clock output features of the device.

Upon reset, the non-reserved bits of this register are cleared to zero. The start-up software must write a specific value to this register in order to configure the CLK output pin function.

When the software writes to this register, it must write a zero to each reserved bit for the device to operate properly. The register should be written in active mode only, not in power save, HALT, or IDLE mode. However, the register contents are preserved during all power modes.

The MCFG register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | CLK2OE | Reserved | | CLK1OE | CLKOE | Reserved |

CLKOE CPU Clock Output Enable. When this bit is cleared (0), the CLK pin (ENV1) remains in the high-impedance state. When this bit is set (1) in normal operating mode, the CLK pin operates as a CPU clock output.

CLK1OE Generated Clock Output 1 Enable. When cleared (0), the CLKOUT1 pin (ENV0) stays in high impedance state. When set (1), the pin outputs the clock from the prescaler controlled by PRSSC1.SCDIV1.

CLK2OE Generated Clock Output 2 Enable. When this bit is set (1) and CLKOE is cleared, the CLKOUT2 pin (ENV1) outputs the clock from the prescaler controlled by PRSSC1.SCDIV2. Otherwise, the CLKOUT2 pin is in high impedance state.

## 5.3 MODULE STATUS (MSTAT) REGISTER

The MSTAT register is a byte-wide, read-only register that indicates the general status of the device.

The MCFG register format is shown below.

| 7 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| Reserved | PGMBUSY | Reserved | OENV1 | OENV0 |

OENV(1:0) Operating Environment. These two bits contain the values applied to the $\overline{ENV1}$ and $\overline{ENV0}$ pins upon reset. These bit values are controlled by the external hardware upon reset and are held constant in the register until the next reset.

PGMBUSY Flash EEPROM Programming Busy. This bit is automatically set to 1 when either the program memory or the data memory is busy being programmed or erased. It is cleared to 0 when neither of the two flash EEPROM memories is busy being programmed or erased. When this bit is set, the software should not attempt any write access to either of these two memories.

# 6.0 Input/Output Ports

Each device has up to 56 software-configurable I/O pins, organized into seven ports of up to eight pins per port. The ports are named Port B, Port C, Port F, Port G, Port H, Port I, and Port L.

Each pin can be configured to operate as a general-purpose input or general-purpose output. In addition, many I/O pins can be configured to operate as a designated input or output for an on-chip peripheral module such as the USART or the Multi-Input Wakeup. This is called the pin's “alternate function.” The alternate functions of all I/O pins are shown in the pinout diagrams in Table1.

The I/O pin characteristics are fully programmable. Each pin can be configured to operate as a TRI-STATE output, push-pull output, weak pull-up input, or high-impedance input. Different pins within the same port can be individually configured to operate in different modes.

Figure1 is a diagram showing the functional features of an I/O port pin. The register bits, multiplexers, and buffers allow the port pin to be configured into the various operating modes.The output buffer is a TRI-STATE buffer with weak pull-up capability. The weak pull-up, if used, prevents the port pin from going to an undefined state when it operates as an input.

The input buffer is disabled when it is not needed to prevent leakage current caused by an input signal’s level between $V_{cc}$-0.2 and $V_{ss}$+0.2 [Volts]. When enabled, it buffers the input signal and sends the pin's logic level to the appropriate on-chip module where it is latched. A Schmitt-Trigger minimizes the effects of electrical noise.

The electrical characteristics and drive capabilities of the input and output buffers are described in Section25.0.

For some pins, a direct low-impedance path is provided between the pin and an internal analog function. These are the input pins to the A/D converter and the analog comparators.

## 6.1 PORT REGISTERS

Each port has an associated set of memory-mapped registers used for controlling the port and for holding the port data. In general, there are five such registers:

— PxALT: Port alternate function register
— PxDIR: Port direction register
— PxDIN: Port data input register
— PxDOUT: Port data output register
— PxWKPU: Port weak pull-up register

In the descriptions of the ports and port registers, the lowercase letter “x” represents the port designation, either B, C, F, G, H, I, or L. For example, “PxDIR register” means any one of the port direction registers: PBDIR, PCDIR, PFDIR, and so on.

All of the port registers are byte-wide read/write registers, except for the port data input registers, which are read-only registers. Each register bit controls the function of the corresponding port pin. For example, PFDIR.2 (bit 2 of the PFDIR register) controls the operation of port pin PF2.

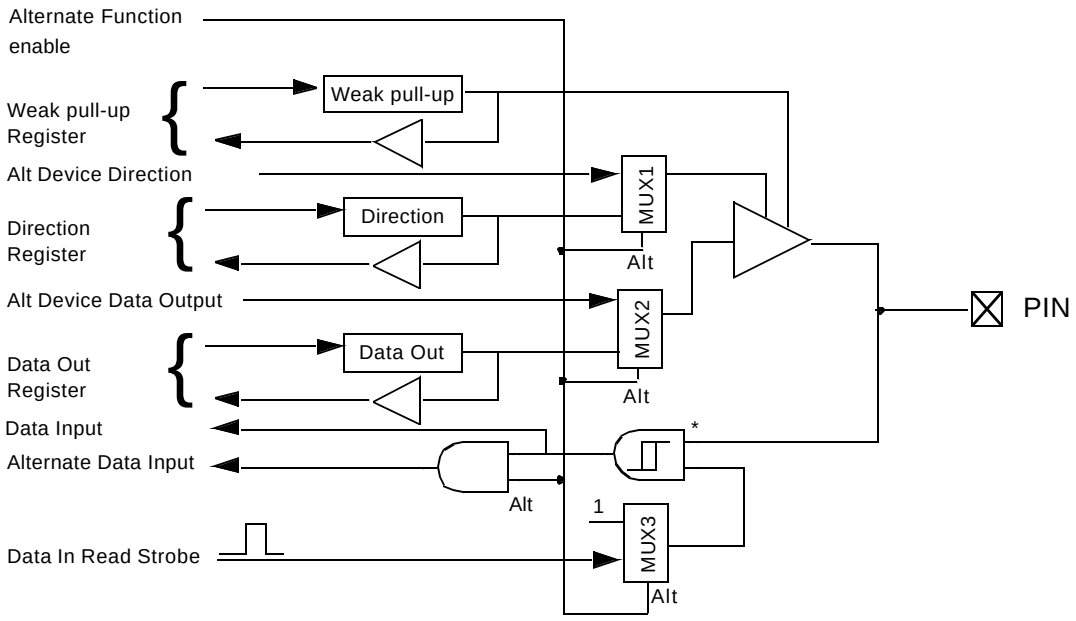



**Figure 1. I/O Pin Functional Diagram**

### 6.1.1 Port Alternate Function Register

Each port that supports an alternate function (any port other than Port B or Port C) has an alternate function register (PxALT). This register determines whether the port pins are used for general-purpose I/O or for the predetermined alternate function. Each port pin can be controlled independently.

A bit cleared to 0 in the alternate function register causes the corresponding pin to be used for general-purpose I/O. In this configuration, the output buffer is controlled by the direction register and the data output register. The input buffer is routed to the data input register. The input buffer is blocked except when the buffer is actually being read.

A bit set to 1 in the alternate function register causes the corresponding pin to be used for its predetermined peripheral I/O function. The output buffer data and TRI-STATE configuration are controlled by signals coming from the on-chip peripheral device. The input buffer is enabled continuously in this case. To minimize power consumption, the input signal should be held within 0.2 volts of the VCC or GND voltage.

A reset operation clears the port alternate function registers to 0, which programs the pins to operate as general-purpose I/O ports. This register must be enabled before the corresponding alternate function is enabled.

### 6.1.2 Port Direction Register

The port direction register (PxDIR) determines whether each port pin is used for input or for output. A bit cleared to 0 causes the pin to operate as an input, which puts the output buffer in the high-impedance state. A bit set to 1 causes the pin to operate as an output, which enables the output buffer.

A reset operation clears the port direction registers to 0, which programs the pins to operate as inputs.

### 6.1.3 Port Data Input Register

The data input register (PxDIN) is a read-only register that returns the current state of each port pin. The CPU can read this register at any time even when the pin is configured as an output.

### 6.1.4 Port Data Output Register

The data output register (PxDOUT) holds the data to be driven onto each port pin configured to operate as a general-purpose output. In this configuration, writing to the register changes the output value. Reading the register returns the last value written to the register.

A reset operation leaves the register contents unchanged. Upon power-up, the registers contain unknown values.

### 6.1.5 Port Weak Pull-Up Register

The weak pull-up register (PxWKPU) determines whether each port pin uses a weak pull-up on the output buffer. A bit set to 1 causes the weak pull-up to be used, while a bit cleared to 0 causes the causes the weak pull-up not to be used.

The pull-up device, if enabled by the register bit, operates in the general-purpose I/O mode whenever the port output buffer is in the TRI-STATE mode. In the alternate function mode, the pull-ups are always disabled.

A reset operation clears the port weak pull-up registers to 0, which disables all pull-ups.

## 6.2 OPEN-DRAIN OPERATION

A port pin can be configured to operate as an inverting open-drain output buffer. To do this, the CPU should clear the bit in the data output register (PxDOUT) and then use the port direction register (PxDIR) to set the value of the port pin. With the direction register bit set to 1 (direction=out), the value zero is forced on the pin. With the direction register bit cleared to 0 (direction=in), the pin is placed in the TRI-STATE mode. If desired, the internal weak pull-up can be enabled to pull the signal high when the output buffer is in the TRI-STATE mode.

# 7.0 CPU and Core Registers

The device uses the same CR16B CPU core as other CompactRISC family members. The core's Reduced Instruction Set Computer (RISC) architecture allows a processing rate of up to one instruction per clock cycle.

The CPU core uses a set of internal registers:

— General-purpose registers (R0-R13, RA, and SP)
— Dedicated address registers (PC, ISP, and INTBASE)
— Processor Status Register (PSR)
— Configuration Register (CFG)

All of these registers are 16 bits wide except for the three address registers, which are 21 bits wide.

Some register bits are designated as “reserved.” The CPU must write a zero to each of these bit locations when it writes to the register. Read operations from reserved bit locations return undefined values.

## 7.1 GENERAL-PURPOSE REGISTERS

There are 16 general-purpose registers, designated R0 through R13, RA, and SP. Registers R0 through R13 can be used for any purpose such as holding variables, addresses, or index values. The RA register is usually used to store the return address upon entry into a subroutine. The SP register is usually used as the pointer to the program run-time stack.

If a general-purpose register is used for a byte-wide operation, only the low-order byte is referenced or modified. The high-order byte is not used or affected by a byte-wide operation.

## 7.2 DEDICATED ADDRESS REGISTERS

There are three dedicated address registers: the Program Counter (PC), the Interrupt Stack Pointer (ISP), and the Interrupt Base Register (INTBASE). Each of these registers is 21 bits wide.

### 7.2.1 Program Counter

The PC register contains the address of the least significant word currently being fetched. It is automatically incremented or changed by the appropriate amount each time an instruction is executed.

The least significant bit of the PC is always zero, thus instructions must always be aligned to an even address in the range of 0000 to 1FFFE hex.

Upon reset, the PC register is initialized to zero and program execution starts at that address (if in IRE-Mode). When a reset signal is received, bits 1 through 16 of the PC register (prior to initialization) are stored in register R0. This allows the software to determine the point in the program at which the reset occurred.

### 7.2.2 Interrupt Stack Pointer

The ISP register points to the lowest address of the last item stored on the interrupt stack. This stack is used by the hardware when an interrupt or trap service procedure is invoked.

### 7.2.3 Interrupt Base Register

The INTBASE register holds the address of the Dispatch Table for interrupts and traps. The least significant bit of the register is always zero. Thus, the Dispatch Table starts at an even address in the range of 0000 to FFFE.

## 7.3 PROCESSOR STATUS REGISTER

The Processor Status Register (PSR) holds status information and selects the operating modes for the CPU core. The format of the register is shown below.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | I | P | E | 0 | N | Z | F | 0 | 0 | L | T | C |

C bit — The Carry (C) bit indicates whether a carry or borrow occurred after addition or subtraction. It is set to 1 if a carry or borrow occurred, or cleared to 0 otherwise.

T bit — The Trace (T) bit, when set, causes a Trace (TRC) trap to be executed after every instruction. This bit is automatically cleared to 0 when a trap or interrupt occurs.

L bit — The Low (L) bit is set by comparison operations. In a comparison of unsigned integers, the bit is set to 1 if the second operand (Rdest) is less than the first operand (Rsrc). Otherwise, it is cleared to 0.

F bit — The Flag (F) bit is a general condition flag that is set by various instructions. It may be used to signal exception conditions or to distinguish the results of an instruction. For example, integer arithmetic instructions use this bit to indicate an overflow condition after an addition or subtraction operation.

Z bit — The Zero (Z) bit is set by comparison operations. In a comparison of integers, the bit is set to 1 if the two operands are equal. Otherwise, it is cleared to 0.

N bit — The Negative (N) bit is set by comparison operations. In a comparison of signed integers, the bit is set to 1 if the second operand (Rdest) is less than the first operand (Rsrc). Otherwise, it is cleared to 0.

E bit — The Local Maskable Interrupt Enable (E) bit is used to enable or disable maskable interrupts. If this bit and the Global Maskable Interrupt Enable (I) bit are both set to 1, all maskable interrupts are accepted. Otherwise, only the nonmaskable interrupt is accepted. The E bit is set to 1 by the Enable Interrupts (EI) instruction and cleared to 0 by the Disable Interrupts (DI) instruction.

P bit — The Trace Trap Pending (P) bit is used together with the Trace (T) bit to prevent a Trace (TRC) trap from occurring more than once for any instruction. The P bit may be cleared to 0 (no TRC trap pending) or set to 1 (TRC trap pending).

I bit — The Global Maskable Interrupt Enable (I) bit is used to enable or disable maskable interrupts. If this bit and the Local Maskable Interrupt Enable (E) bit are both set to 1, all maskable interrupts are accepted. Otherwise, only the non-maskable interrupt is accepted. This bit is automatically cleared to 0 when an interrupt occurs and automatically set to 1 upon completion of an interrupt service routine.

Upon reset, all non-reserved bits of the register are cleared to 0 except for the E bit (bit 9), which is set to 1. When a device reset occurs, the PSR contents prior to the reset are stored into register R1, allowing the initialization software to determine the state of the device prior to the reset operation.

## 7.4 CONFIGURATION REGISTER

The Configuration (CFG) register is a 16-bit core register that determines the size of the INTBASE register. For the device, the CFG register should always be left in its default state (cleared to zero), resulting in a 16-bit INTBASE register.

## 7.5 ADDRESSING MODES

Each instruction operates on one or more operands. An operand can be a register or a memory location.

Most instructions use one, two, or three device registers as operands. The instruction opcode specifies the registers to be operated on. Some instructions may use an immediate value (a value provided in the instruction itself) instead of a register.

Memory locations are accessed only by the Load and Store commands. The memory location to use for a particular instruction can be specified as an absolute, relative, or far-relative address.

The instruction set supports the following addressing modes:

**Register Mode** — The operand is a general-purpose register: R0 through R13, RA, or SP. For example:

ADDB R1, R2

**Immediate Mode** — A constant operand value is specified within the instruction. In a branch instruction, the immediate operand is a displacement from the program counter (PC). In the assembly language syntax, a dollar sign indicates an immediate value. For example:

MULW $4, R4

**Relative Mode** — The operand is located in memory. Its address is obtained by adding the contents of a general purpose register to the constant value encoded into the displacement field of the instruction. For example:

LOADW 12(R5), R6

**Far-Relative Mode** — The operand is located in memory. Its address is obtained by concatenating a pair of adjacent general-purpose registers to form a 21-bit value, and adding this value to the constant value encoded into the displacement field of the instruction.

**Absolute Mode** — The operand is located in memory. Its address is specified within the instruction. For example:

LOADB 4000, R6

For additional information on the instruction set and instruction encoding, see the CompactRISC CR16B Programmer's Reference manual.

## 7.6 STACKS

A stack is a one-dimensional data buffer in which values are entered and removed one at a time. The last valued entered is the first one removed. A register called the stack pointer contains the current address of the last item entered on the stack. In the device, when an item is entered or “pushed” onto the stack, the stack expands downward in memory (the stack pointer is decremented). When an item is removed or “popped” from the stack, the stack shrinks upward in memory (the stack pointer is incremented).

The device uses two type of stacks: the program stack and the interrupt stack.

The program stack is used by the software to save and restore register values upon entry into and exit from a subroutine. The software can also use the program stack to store local and temporary variables. The stack pointer for this stack is the SP register.

The interrupt stack is used to save and restore the program state when an exception occurs (an interrupt or software trap). The on-chip hardware automatically pushes the program state information onto the stack before the exception service procedure is executed. Upon exit from the exception service procedure, the hardware pops this information from the stack and restores the program state. The stack pointer for this stack is the ISP register.

## 7.7 INSTRUCTION SET

Table7 is a summary list of all instructions in the device instruction set. For each instruction, the table shows the mnemonic and a brief description of the operation performed.

In the Mnemonic column, the lower-case letter “i” is used to indicate the type of integer that the instruction operates on, either “B” for byte or “W” for word. For example, the notation ADDi for the “add” instruction means that there are two forms of this instruction, ADDB and ADDW, which operate on bytes and words, respectively.

Similarly, the lower-case string “cond” is used to indicate the type of condition tested by the instruction. For example, the notation Jcond represents a class of conditional jump instructions: JEQ for Jump on Equal, JNE for Jump on Not Equal, and so on.

For detailed information on all instructions, see the CompactRISC CR16B Programmer's Reference manual.

**Table 7 Device Instruction Set Summary**

| Mnemonic | Description |
|---|---|
| ADDi | Add Integer |
| ADDUi | Add Unsigned Integer |
| ADDCi | Add Integer with Carry |
| ANDi | Bitwise Logical AND |
| ASHUi | Arithmetic Shift Unsigned |
| Bcond | Conditional Branch |
| Bcond0i | Compare Register to 0 and Branch |
| Bcond1i | Compare Register to 1and Branch |
| BAL | Branch and Link |
| BR | Unconditional Branch |
| CBITi | Clear Bit in Integer |
| CMPi | Compare Integer |
| DI | Disable Maskable Interrupts |
| EI | Enable Maskable Interrupts |
| EIWAIT | Enable Interrupts and Wait for Interrupt |
| EXCP | Exception |
| Jcond | Conditional Jump |
| JAL | Jump and Link |
| JUMP | Jump |
| LOADi | Load Integer |
| LOADM | Load Multiple Registers |
| LPR | Load Processor Register |
| LSHi | Logical Shift Integer |
| MOVi | Move Integer |
| MOVXB | Move with Sign-Extension |
| MOVZB | Move with Zero-Extension |
| MULi | Multiply Integer |
| MULSi | Multiply Signed |
| MULUW | Multiply Unsigned |
| NOP | No Operation |
| ORi | Bitwise Logical OR |
| POP | Pop Registers from Stack |
| POPRET | Pop and jump RA |
| PUSH | Push Registers on Stack |
| RETX | Return from Exception |
| Scond | Save Condition as Boolean |
| MULi | Multiply Integer |
| SBITi | Set Bit in Integer |
| STORi | Store Integer |
| STORM | Store Registers to Memory |
| SUBi | Subtract Integer |

**Table 7 Device Instruction Set Summary**

| Mnemonic | Description |
|---|---|
| SUBCi | Subtract Integer with Carry |
| TBIT | Test Bit |
| WAIT | Wait for Interrupt |
| XORi | Bitwise Logical Exclusive OR |

# 8.0 Bus Interface Unit

The Bus Interface Unit (BIU) controls the interface between the internal core bus and those on-chip modules which are mapped into BIU zones. These on-chip modules are the flash EEPROM program memory, the ISP-memory and the I/O-zone. It determines the configured parameters for bus access (such as the number of wait states for memory access) and issues the appropriate bus signals for the requested access.

**Note:** The device is manufactured in a 224-pin version which is used in emulation equipment. In the 224-pin device, the BIU controls access to both on-chip and off-chip memory and peripherals. Operation of the 224-pin device and the use of chip-external memory is beyond the scope of this data sheet.

## 8.1 BUS CYCLES

There are four types of data transfer bus cycles:

— Normal read
— Fast read
— Early write
— Late write

The type of data cycle used in a particular transaction depends on the type of CPU operation (a write or a read), the type of memory or I/O being accessed, and the access type programmed into the BIU control registers (early/late write or normal/fast read).

For read operations, a basic normal read takes two clock cycles, whereas a fast read bus cycle takes one clock cycle. Upon reset of the device, normal read bus cycles are enabled by default.

For write operations, a basic late write bus cycle takes two clock cycles, whereas a basic early write bus cycle takes three clock cycles. Upon reset of the device, early write bus cycles are enabled by default. However, late write bus cycles are needed for ordinary write operations, so this configuration should be changed by the application software (see Section8.2.1).

In certain cases, one or more additional clock cycles are added to a bus access cycle. There are two types of additional clock cycles for ordinary memory accesses, called internal wait cycles (TIW) and hold ($T_{hold}$) cycles.

A wait cycle is inserted in a bus cycle just after the memory address has been placed on the address bus. This gives the accessed memory more time to respond to the transaction request. A hold cycle is inserted at the end of a bus cycle. This holds the data on the data bus for an extended number of clock cycles.

## 8.2 BIU CONTROL REGISTERS

The BIU has a set of control registers that determine how many wait cycles and hold cycles are to be used for accessing memory. Upon start-up of the device, these registers should be programmed with appropriate values so that the minimum allowable number of cycles is used. This number varies with the clock frequency used.

There are four applicable BIU registers: the BIU Configuration (BCFG) register, the I/O Configuration (IOCFG) register, the Static Zone 0 Configuration (SZCFG0) register and the Static Zone 1Configuration (SZCFG1) register. These registers control the bus cycle configuration used for accessing the various on-chip memory types.

**Note:** A system configuration register called the Module Configuration (MCFG) register controls the number of wait cycles used for accessing the EEPROM data memory. This register is described in Section5.1.

### 8.2.1 BIU Configuration (BCFG) Register

The BIU Configuration (BCFG) Register is a byte-wide, read/write register that selects either early write or late write bus cycles. The register address is F900 hex. Upon reset, the register is initialized to 07 hex. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | Note 1 | Note 1 | EWR |

EWR Early Write. This bit is cleared to 0 for late write operation (two clock cycles to write) or set to 1 for early write operation.

**Note 1:** These bits (bit 1 or bit 2) control the configuration of the 224-pin device used in emulation equipment. The CPU should set this bit to 1 when it writes to the register.

Upon reset, the BCFG register is initialized to 07 hex, which selects early write operation. However, late write operation is required for normal device operation, so the software should change the register value to 06 hex.

### 8.2.2 I/O Zone Configuration (IOCFG) Register

The I/O Zone Configuration (IOCFG) register is a word-wide, read/write register that sets the timing and bus characteristics of I/O Zone memory accesses. In the device implementation, the registers associated to Port B and Port C reside in the I/O memory array. (These ports are used as a 16-bit data port, if the device operates in development mode.)

The IOCFG register address is F902 hex. Upon reset, the register is initialized to 069F hex. The register format is shown below.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | IPST | Reserved |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BW | Reserved | | HOLD | | WAIT | | |

WAIT Memory Wait cycles
This field specifies the number of TIW (internal wait state) clock cycles added for each memory access, ranging from 000 binary for no additional TIW wait cycles to 111 binary for seven additional TIW wait cycles.

HOLD Memory Hold cycles
This field specifies the number of $T_{hold}$ clock cycles used for each memory access, ranging from 00 binary for no $T_{hold}$ cycles to 11 binary for three $T_{hold}$ clock cycles.

BW Bus Width.
This bit defines the bus width of the zone.
If cleared to 0, a bus width of 8-bit is used.
if set to 1, a bus width of 16-bit is used.
For the device, a bus width of 16-bit needs to be set.

IPST Post Idle.
An idle cycle follows the current bus cycle, when the next bus cycle accesses a different zone.
If cleared to 0, no idle cycle is inserted.
If set to 1, one idle cycle is inserted.
The IPST bit can be cleared to 0, as no idle cycles are required for on-chip accesses.

**Note:** Reserved bits must be cleared to 0 when the CPU writes to the register.

### 8.2.3 Static Zone 0 Configuration (SZCFG0) Register

The Static Zone 0 Configuration (SZCFG0) register is a word-wide, read/write register that sets the timing and bus characteristics of Zone 0 memory accesses. In the device implementation of the CompactRISC architecture, Zone 0 is occupied by the flash EEPROM program memory.

The SCCFG0 register address is F904 hex. Upon reset, the register is initialized to 069F hex. The register format is shown below.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | FRE | IPRE | IPST | Reserved |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BW | Reserved | | HOLD | | WAIT | | |

WAIT Memory Wait cycles
This field specifies the number of TIW (internal wait state) clock cycles added for each memory access, ranging from 000 binary for no additional TIW wait cycles to 111 binary for seven additional TIW wait cycles. These bits are ignored if the SZCFG0.FRE bit is set to 1.

HOLD Memory Hold cycles
This field specifies the number of $T_{hold}$ clock cycles used for each memory access, ranging from 00 binary for no $T_{hold}$ cycles to 11 binary for three $T_{hold}$ clock cycles. These bits are ignored if the SZCFG0.FRE bit is set to 1.

BW Bus Width.
This bit defines the bus width of the zone.
If cleared to 0, a bus width of 8-bit is used.
if set to 1, a bus width of 16-bit is used.
For the devicedevice a bus width of 16-bit needs to be set.

FRE Fast Read Enable
This bit enables (1) or disables (0) fast read bus cycles. A fast read operation takes one clock cycle. A normal read operation takes at least two clock cycles.

IPST Post Idle.
An idle cycle follows the current bus cycle, when the next bus cycle accesses a different zone.
If cleared to 0, no idle cycle is inserted.
If set to 1, one idle cycle is inserted.
The IPST bit can be cleared to 0, as no idle cycles are required for on-chip accesses.

IPRE Preliminary Idle.
An idle cycle is inserted prior to the current bus cycle, when the new bus cycle accesses a different zone.
If cleared to 0, no idle cycle is inserted.
If set to 1, one idle cycle is inserted.
The IPRE bit can be cleared to 0, as no idle cycles are required for on-chip accesses.

**Note:** Reserved bits must be cleared to 0 when the CPU writes to the register.

### 8.2.4 Static Zone 1 Configuration (SZCFG1) Register

The Static Zone 1 Configuration (SZCFG1) register is a word-wide, read/write register that sets the timing and bus characteristics of Zone 1 memory accesses. In the device implementation of the CompactRISC architecture, Zone 1 is occupied by the boot ROM memory (ISP-Memory).

The SCCFG1 register address is F906 hex. Upon reset, the register is initialized to 069F hex. The register format is shown below.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | FRE | IPRE | IPST | Reserved |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BW | Reserved | | HOLD | | WAIT | | |

WAIT Memory Wait cycles
This field specifies the number of TIW (internal wait state) clock cycles added for each memory access, ranging from 000 binary for no additional TIW wait cycles to 111 binary for seven additional TIW wait cycles. These bits are ignored if the SZCFG0.FRE bit is set to 1.

HOLD Memory Hold cycles
This field specifies the number of $T_{hold}$ clock

cycles used for each memory access, ranging from 00 binary for no $T_{hold}$ cycles to 11 binary for three $T_{hold}$ clock cycles. These bits are ignored if the SZCFG0.FRE bit is set to 1.

BW — Bus Width.
This bit defines the bus width of the zone.
If cleared to 0, a bus width of 8-bit is used.
if set to 1, a bus width of 16-bit is used.
For the device a bus width of 16-bit needs to be set.

FRE — Fast Read Enable
This bit enables (1) or disables (0) fast read bus cycles. A fast read operation takes one clock cycle. A normal read operation takes at least two clock cycles.

IPST — Post Idle.
An idle cycle follows the current bus cycle, when the next bus cycle accesses a different zone.
If cleared to 0, no idle cycle is inserted.
If set to 1, one idle cycle is inserted.
The IPST bit can be cleared to 0, as no idle cycles are required for on-chip accesses.

IPRE — Preliminary Idle.
An idle cycle is inserted prior to the current bus cycle, when the new bus cycle accesses a different zone.
If cleared to 0, no idle cycle is inserted.
If set to 1, one idle cycle is inserted.
The IPRE bit can be cleared to 0, as no idle cycles are required for on-chip accesses.

**Note:** Reserved bits must be cleared to 0 when the CPU writes to the register.

## 8.3 WAIT AND HOLD STATES USED

The number of wait cycles and hold cycles inserted into a bus cycle depends on whether it is a read or write operation, the type of memory or I/O being accessed, and the control register settings.

### 8.3.1 Flash EEPROM Program Memory

When the CPU accesses the flash EEPROM program memory (address ranges 0000-BFFF and 1C000-1FFFF), the number of added wait and hold cycles depends on the type of access and the BIU register settings.

In fast read mode (SZCFG0.FRE=1), a read operation is a single cycle access. This limits the maximum CPU operating frequency to either 10 MHz or 20 MHz (see Section9.1.5).

For a read operation in normal read mode (SZCFG0.FRE=0), the number of inserted wait cycles is one plus the value written to the SZCFG0.WAIT field. The number in this field can range from zero to seven, so the total number of wait cycles can range from one to eight. The number of inserted hold cycles is equal to the value written to the SCCFG0.HOLD field, which can range from zero to three.

For a write operation in fast read mode (SZCFG0.FRE=1), the number of inserted wait cycles is one. No hold cycles are used.

For a write operation normal read mode (SZCFG0.FRE=0), the number of wait cycles is equal to the value written to the SZCFG0. WAIT field plus one (in the late write mode) or two (in the early write mode). The number of inserted hold cycles is equal to the value written to the SCCFG0.HOLD field, which can range from zero to three.

Writing to the flash EEPROM program memory is a Flash programming operation that requires some additional steps, as explained in Section9.3.

### 8.3.2 RAM Memory

Read and write accesses to on-chip RAM is performed within a single cycle, regardless of the BIU settings.

### 8.3.3 EEPROM Data Memory

There is either no wait state or one wait state used when the CPU accesses the EEPROM data memory (address F000-F27F hex). The number of required wait states (zero or one) depends on the CPU clock frequency and operating mode, and is controlled by programming of the DMCSR.ZEROWS bit in the MCFG register, as explained in Section9.3. No hold cycles are used.

### 8.3.4 Accesses to Peripheral

When the CPU accesses on-chip peripherals in the range of F800-FAFF hex and FC00-FFFF hex, one wait cycle and one preliminary idle cycle is used. No hold cycles are used.

The IOCFG register determines the access timing for the address range FB00-FB16 hex (Ports B and Port C).

### 8.3.5 Access Timing Summary Table

Table8 is a summary showing the number of access cycles used for various address ranges.

**Table 8 Access Timing Table**

| Address Range (hex) | Memory or I/O Type | Access Cycles | |
|---|---|---|---|
| | | read | write |
| 0000-BFFF | Flash EEPROM Program Memory | SZCFG0.FRE=1:<br>1 cycle | SZCFG0.FRE=1:<br>1 cycle<br>+ BCFG.EWR<br>(+ programming time) |
| | | SZCFG0.FRE=0:<br>2 cycles<br>+ SZCFG0.WAIT<br>+ SZCFG0.HOLD | SZCFG0.FRE=0:<br>2 cycles<br>+ BCFG.EWR<br>+ SZCFG0.WAIT<br>+ SZCFG0.HOLD<br>(+ programming time) |
| C000-CBFF | Static RAM Memory | 1 cycle | 1 cycle |
| F000-F27F | EEPROM Data Memory | MCFG.ZEROWS=1:<br>1 cycle | MCFG.ZEROWS=1:<br>1 cycle<br>(+ programming time) |
| | | MCFG.ZEROWS=0:<br>2 cycles | MCFG.ZEROWS=0:<br>2 cycles<br>(+ programming time) |
| F900-FFFF<br>F800-F9FF<br>FC00-FFFF | On-Chip Peripherals | 2 cycles | 2 cycles |
| FB00-FBFF | Ports B and C | 3 cycle<br>+ IOCFG.WAIT<br>+ IOCFG.HOLD | 3 cycle<br>+ BCFG.EW<br>+ IOCFG.WAIT<br>+ IOCFG.HOLD |

### 8.3.6 Recommended Register Settings

Table9 shows the recommended register settings for various clock rates. Different clock rates require different register settings because the flash EEPROM program memories have specific setup and hold requirements that can be met only by using enough wait cycles and hold cycles.

Between clock rates of 10 MHz and 20MHz, the number of wait states required for memory access (either none or one) depends on the desired power mode of the program memory.

**Table 9 Recommended Register Settings**

| Clock Rate | SZCFG0 | SZCFG1 | IOCFG |
|---|---|---|---|
| < 10 MHz,<br>0 wait state | 0880 hex | 0880 hex | 0080 hex |
| 10 to 20MHz,<br>0 wait state | 0880 hex | 0880 hex | 0080 hex |
| 10 to 20MHz,<br>1 wait state | 0080 hex | 0080 hex | 0080 hex |
| > 20 MHz,<br>1 wait state | 0080 hex | 0080 hex | 0080 hex |

# 9.0 Memory

The CompactRISC architecture supports a uniform linear address space of 2 megabytes, addressed by 21 bits. The device implementation of this architecture uses only the lowest 128K bytes of address space. Each memory location contains a byte consisting of eight bits.

Various types of on-chip memory occupy specific intervals within the address space: 64K bytes of flash EEPROM program memory, 3K bytes of static RAM, 2K bytes of low endurance EEPROM data memory, 128 bytes of high endurance EEPROM data memory, and 1.5K bytes of ISP memory. All of these memories are 16 bits wide, and their contents can be accessed either as bytes (eight bits wide) or words (16 bits wide except for the program memory which only supports word access).

The CPU core uses the Load and Store instructions to access memory. These instructions can operate on bytes or words. For a byte access, the CPU operates on a single byte occupying a specified memory address. For a word access, the CPU operates on two consecutive bytes. In that case, the specified address refers to the least significant byte of the data value; the most significant byte is located at the next higher address. Thus, the ordering of bytes in memory is from least to most significant byte, known as "little-endian" ordering. For more efficient data access operations, 16-bit variables should be stored starting at word boundaries (at even address).

## 9.1 FLASH EEPROM PROGRAM MEMORY

The flash EEPROM program memory is used to store the application program. The 64K bytes of this memory reside in the address range of 0000-BFFF hex and 1C000-1FFFF in Zone 0 of the CR16B address space. A normal CPU write operation to this memory has no effect.

The flash EEPROM Program Memory module has the following features:

- — 64K bytes arranged as 32K by 16 bits
- — Page size of 64 words
- — 30 µs programming pulse per word
- — Page mode erase with a 1 ms pulse, mass erase with 4ms pulse
- — All erased flash EEPROM program memory bits read 1
- — Fast single cycle read access
- — Flexible software controlled In-System-Programming (ISP) capability
- — Pipelined programming cycles through double-buffered data register, with write access disabled when the register is full
- — Programming high voltage and timing generated on-chip
- — Memory disabled when address is out of range
- — Requires valid key for program and erase to proceed
- — Provide busy status during programming and erase
- — Read accesses disabled during programming and erase
- — Security features to limit read/write access

### 9.1.1 Reading

Program memory read accesses can operate without wait cycles with a CPU clock rate of up to 20MHz in the normal mode. At higher clock rates, memory read accesses can operate with one wait state.

The programmed number of wait cycles used (either zero or one) is controlled by the BIU Configuration (BCFG) register and the Static Zone 0 Configuration (SZCFG0) register. These registers are described in Section8.0.

### 9.1.2 Conventional Programming Modes

The flash EEPROM program memory can be programmed either with the device plugged into a flash EEPROM programmer unit (External Programming) or with the device already installed in the application system (In-System-Programming).

If the device is programmed using a flash EEPROM programmer, the device is set into an external programming mode. In this mode the device operates as if it were a pure flash memory device. The flash memory is programmed without involving any CPU activity.

If the device is to be programmed within the user application, it can either be done by an user written boot loader or by utilizing a pre-programmed in-system-programming code (ISP-Code) residing in the boot ROM array of the device.

The device executes the pre-programmed in-system-programming code if it operates in the In-System-Programming Mode (ISP-Mode). To enter the ISP-Mode the device must be reset (or powered-up) with the ENV0-pin set to low level and the ENV1-pin set to high level (or left open). Also if the flash program memory is not programmed yet (FLCTRL2.EMPTY bit is still set) the device automatically enters the ISP-Mode after reset, even though both pins ENV0 and ENV1are at high level (or left open). If the device enters the ISP-Mode it starts execution at address E000 hex.

In ISP-Mode the program code can be downloaded into the device using one of the on-chip USARTs and written into the flash program memory. For more detailed information on the In-System-Programming features of the pre-programmed ISP-Code please refer to the ISP-Monitor manual.

### 9.1.3 User-Coded Programming Routines

Instead of using a flash EEPROM programmer unit or the conventional in-system programming mode, you can write your own processor code to program and erase the flash EEPROM program memory. User-written code is more flexible than using the other programming methods. Like the conventional in-system programming mode, the device is programmed while it is installed in the system. It is not necessary to reset the device or use the $\overline{\text{ENV0}}$/$\overline{\text{ENV1}}$ pins to configure the device.

User-written flash programming code must reside outside of the flash program memory. This is because the entire program memory becomes unavailable while programming or erasing any part of this memory.

### 9.1.4 Flash EEPROM Programming and Verify

The flash EEPROM program memory programming and erase can be performed using different methods. It can be done through user code that is stored in system RAM, or through In-System-Programming mode, but should not be programmed through the flash EEPROM program memory it-

self as no instruction or data can be fetched from it while it is being programmed. All program and erase operations must be preceded immediately by writing the proper key to the program memory key register PGMKEY.

The flash EEPROM program memory is divided into 256 pages, each page containing 64 words (each 16 bits wide). Each page is further divided into two adjacent rows. A page erase will erase one page. Programming is done by writing to all the words within a row, one word following another sequentially within one single high voltage pulse. This is supported through a double-buffered write-data buffer scheme. Byte programming is not supported. Programming should be done on erased rows.

A mass erase requires the following code sequence (assuming that this sequence will not be interrupted to do another flash erase or programming):

1. Check for MSTAT.PGMBUSY not set.
2. Set up flash timing reload registers for mass erase operation.
3. Set FLCSR.MERASE = 1.
4. If interrupt was enabled, disable interrupt.
5. Write proper key value to PGMKEY.
6. Write to any valid location within the flash EEPROM program memory.
7. If interrupt was disabled in step 4, re-enable interrupt.
8. Wait for MSTAT.PGMBUSY to clear.
9. Set FLCSR.MERASE = 0.
10. Restore flash timing reload registers for normal operation.

A page erase requires the following code sequence (assuming that this sequence will not be interrupted to do another flash erase or programming):

1. Check for MSTAT.PGMBUSY not set.
2. Set FLCSR.ERASE = 1.
3. If interrupt was enabled, disable interrupt.
4. Write proper key value to PGMKEY.
5. Write to any valid location within the page to be erased.
6. If interrupt was disabled in step 3, re-enable interrupt.
7. Set FLCSR.ERASE = 0.

When programming, the data to be written into the flash EEPROM program memory is first written into a double-buffered write-data buffer. When a piece of data is written to the page while the flash EEPROM program memory is idle, the write cycle will start. Due to the double-buffered nature of the write-data buffer, a second word can be written to the flash EEPROM program memory. This will then set FLCSR.PMLFULL flag indicating the buffer is now full. When the first write is done, the memory address would be incremented, and the second word would be written to that address while keeping the high voltage pulse active; the FLCSR.PMLFULL flag is cleared. Another word can then be written to the buffer, and this programming will repeat until there are no more words to be programmed. This allows pipelined writes to different words on the same row within the same high voltage pulse. If the programming sequence exceeds a row, the flash programming interface will automatically initiate a programming pulse for the next row. The FLCSR.PMLFULL bit is also cleared when programming of the last word of the current row is completed, e.g. programming of the entire row is completed and MSTAT.PGMBUSY is cleared. This means, the separation of the program memory into rows is transparent to the user, as the transition is handled by the flash program memory interface. Figure 3 shows a flowchart for a programming sequence.

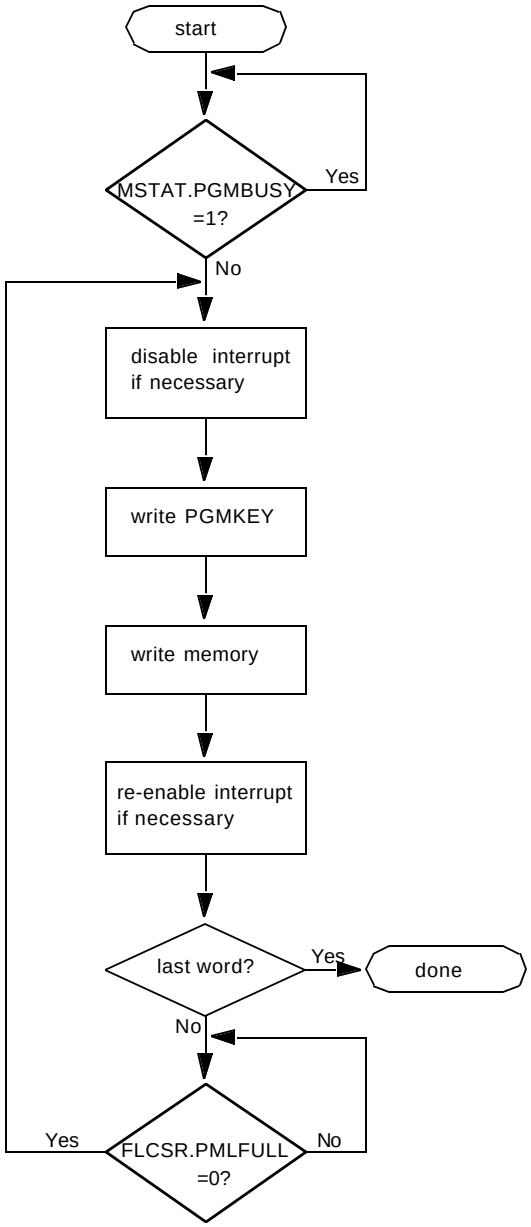



**Figure 2. Programming Sequence for the Program Memory**

### 9.1.5 Erase and Programming Timing

The internal hardware of the device handles the timing of erase and programming operations. To drive the timing control circuits, the device divides the system clock by a programmable prescaler factor. You should select a prescaler value to produce a program/erase clock of 200 kHz (or as close as possible to 200 kHz without exceeding 200 kHz). For the timing control circuit to operate correctly, you must

program the prescaler value in advance and leave it unchanged while a program or erase operation is in progress. A similar (but separate) prescaler factor is applied to the EEPROM data memory. See Section9.1.7 and Section9.3.4 for details.

### 9.1.6 Flash EEPROM Program Memory Control and Status Register (FLCSR)

The Flash EEPROM Program Memory Control and Status (FLCSR) register is a byte-wide, read/write register that contains several status and control bits related to the program memory. All reserved bits must be written with 0 for the memory to operate properly when writing to this register. Upon reset, this register is cleared to zero when the flash memory on the chip is in the idle state.

The register format is shown below.

| 7 | 6 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| MERASE | Reserved | PMLFULL | PMBUSY | PMER | Reserved |

PMER Flash EEPROM Program Memory page erase. When set (1) with MERASE bit cleared, a valid write to the flash EEPROM program memory erases the entire flash EEPROM program memory page pointed to by the write address rather than performing a write to the addressed memory location.

PMBUSY Program Memory Busy. This bit is automatically set to 1 when the flash EEPROM program memory is busy being programmed, and cleared to 0 at all other times. (The MSTAT.PGMBUSY is also set to 1 whenever the PMBUSY bit is set to 1.)

PMLFULL Program Memory Write-Latch Buffer Full. When set (1), the double-buffered data register for program memory write operations is full. When cleared (0), the double-buffered data register is not full.

MERASE Mass Erase Flash EEPROM Program Memory Array. When set (1) in ISP or test mode, a valid write to the flash EEPROM program memory performs an erase to the whole flash EEPROM program memory rather than perform a write to the addressed memory location. However, it is necessary to enter new values into the FLERASE and FLEND registers to adjust the mass erase timing before starting the mass erase.

### 9.1.7 Program Memory Timing Prescaler Register (FLPSLR)

The FLPSLR register is a byte-wide, read/write register that selects the prescaler divider ratio for the flash EEPROM program memory programming clock. Before you program or erase the program memory for the first time, you should program the FLPSLR register with the proper prescaler value, an 8-bit value called FTDIV. The device divides the system clock by (FTDIV+1) to produce the program memory programming clock.

You should choose a value of FTDIV to produce a clock of the highest possible frequency that is equal to or just less than 200 kHz. For example, if the system clock frequency is 12.5 MHz, use the value 3E hex (62 decimal) for FTDIV, because 12.5 MHz / (62+1) = 198.4 kHz. Do not modify this register while a flash EEPROM program or erase operation is in progress.

Upon reset, this register is programmed by default with the value 63 hex (99 decimal), which is an appropriate setting for a 20 MHz system clock.

### 9.1.8 Program Memory Start Time Reload (FLSTART)

The FLSTART register is a byte-wide read/write register that controls the program and erase start delay time. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you program or erase the program memory for the first time, program the FLSTART register with the proper prescaler value, FTSTART. The flash timing counter generates a delay of (FTSTART+1) prescaler output clocks. The default value provides a delay time of 10µs when the prescaler output clock is 200kHz. Do not modify this register while a program or erase operation is in progress.

Upon reset, this register resets to $01_{16}$ when the flash memory on the chip is in an idle state.

### 9.1.9 Program Memory Transition Time Reload Register (FLTRAN)

The FLTRAN register is a byte-wide read/write register that controls some program/erase transition times. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you program or erase the program memory for the first time, you should program the FLTRAM register with the proper prescaler value, FTTRAN. The flash timing counter generates a delay of (FTTRAN + 1) prescaler output clocks. The default value provides a delay time of 5µs when the prescaler output clock is 200kHz. Do not modify this register while a program or erase operation is in progress.

Upon reset, this register resets to $00_{16}$ when the flash memory on the chip is in an idle state.

### 9.1.10 Program Memory Programming Time Reload Register (FLPROG)

The FLPROG register is a byte-wide read/write register that controls the programming pulse width. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you program or erase the program memory for the first time, program the FLPROG register with the proper prescaler value, FTPROG. The flash timing counter generates a programming pulse width of (FTPROG + 1) prescaler output clocks. The default value provides a delay time of 30µs when the prescaler output clock is 200kHz.

Do not modify this register while program/erase operation is in progress.

Upon reset, this register resets to $05_{16}$ when the flash memory on the chip is in idle state.

#### 9.1.11 Program Memory Erase Time Reload Register (FLERASE)

The FLERASE register is a byte-wide read/write register that controls the erase pulse width. This value is loaded into the upper 8 bits of the flash timing counter, and at the same time, $11_2$ is loaded into the lower 2 bits. Before you program or erase the program memory for the first time, program the FLERASE register with the proper prescaler value, FTER. The flash timing counter generates a erase pulse width of 4×(FTER + 1) prescaler output clocks. The default value provides a delay time of 1ms when the prescaler output clock is 200kHz. Do not modify this register while a program or erase operation is in progress.

Upon reset, this register resets to $31_{16}$ when the flash memory on the chip is in idle state.

For mass erase, this value should be changed to $C7_{16}$ to generate a pulse width that is four times as long as the page erase.

#### 9.1.12 Program Memory End Time Reload Register (FLEND)

The FLEND register is a byte-wide read/write register that controls the delay time after a program/erase operation. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you program or erase the program memory for the first time, program the FLEND register with the proper prescaler value, FTEND. The flash timing counter generates a delay of (FTEND + 1) prescaler output clocks. The default value provides a delay time of 5µs when the prescaler output clock is 200kHz. Do not modify this register while program/erase operation is in progress.

Upon reset, this register resets to $00_{16}$ when the flash memory on the chip is in idle state.

For mass erase, this value should be changed to $13_{16}$ to provide for a delay time twenty times that of the standard delay.

#### 9.1.13 Program Memory Prescaler Count Register (FLPCNT)

The FLPCNT register is a byte-wide read-only register that returns the value of the program memory prescaler counter. FPCNT contains the flash timing prescaler present count value.

#### 9.1.14 Program Memory Timer Count Register 1 (FLCNT1)

The FLCNT1 register is a byte-wide read-only register that returns the lower 8 bits of the program memory timing counter value. FLCNTL is the lower 8 bits of the flash timer present count value.

#### 9.1.15 Program Memory Timer Count Register 2 (FLCNT2)

The FLCNT2 register is a byte-wide read-only register that returns the upper 2 bits of the program memory timing counter value and also the state of the key flash memory interface timing signals. The interface timing signals are only used in special test modes. Their function is beyond the scope of this document.

#### 9.1.16 Program Memory Write Key Register (PGMKEY)

The PGMKEY register is a byte-wide, write-only register that must be written with a key value ($A3_{16}$) immediately prior to each write to the flash EEPROM program memory. Otherwise, the write operation to the program memory will fail. This feature is intended to prevent unintentional programming of the program memory.

Reading this register always returns FF hex.

Upon reset, the write enable status that is generated as a result of writing to this key register is cleared.

### 9.2 RAM MEMORY

The static RAM memory is used for temporary storage of data and for the program and interrupt stacks. The 3K bytes of this memory reside in the address range of C000-CBFF hex. Each memory access requires one clock cycle, for a byte or word access. No wait cycles or hold cycles are required. For non-aligned word access, each memory access requires multiple clock cycles.

### 9.3 FLASH EEPROM DATA MEMORY

The flash EEPROM data memory is used for non-volatile storage of data. The 2K bytes of low endurance memory reside in the address range of E800-EFFF hex and the 128 bytes of high endurance memory reside in the address range of F000-F07F hex. The CPU reads or writes this memory by using ordinary byte-wide or word-wide memory access commands. This memory shares the same array as the ISP flash program memory.

This memory also support flash memory test mode and there is no read protection or permanent write protection for this memory.

#### 9.3.1 Reading

The flash EEPROM data memory read accesses can operate without wait cycles with a CPU clock rate of up to 20MHz in the normal mode. At higher clock rates, read accesses can operate with one wait state.

The programmed number of wait cycles used (either zero or one) is controlled by a bit in the Data Memory Control Status register (DMCSR.ZEROWS). This register is described in Section9.3.3.

#### 9.3.2 Programming

Before you begin programming the flash EEPROM data memory, you should set the value in the EEPROM Data Memory Prescaler register. This register sets the prescaler used to generate the data memory programming clock from the system clock, as described in Section9.3.4.

A code fetch from ISP flash EEPROM program memory is not possible while flash EEPROM data memory is being programmed because they share the same memory array.

After the CPU performs a write to the flash EEPROM data memory, the on-chip hardware completes the EEPROM programming in the background. When programming begins, the on-chip hardware sets the DMCSR.DMBUSY bit to 1, and also sets the MSTAT.PGMBUSY bit to 1. When programming is completed, it resets these status bits back to 0. Once the software writes to the flash EEPROM data memory, it should not attempt to access the EEPROM data memory

again until programming is completed and the status bit is reset to 0.

The device hardware internally generates the voltages and timing signals necessary for programming. No additional power supply is required, nor any software required except to check the status bit for completion of programming. The minimum time required to erase and reprogram a byte or word is 1.1 ms. The programmed values can be verified by using normal memory read operations. The prescaler output drives a 10-bit counter to generate timing pulses and there are five reload registers to produce various pulse widths.

If a reset occurs during a programming or erase operation, the operation is terminated. The reset is extended until the flash memory returns to the idle state. Therefore, the timing logic and program or erase state machine is not cleared on reset; they are cleared on power-up with the clear signal active until the bus signals are in a known state.

The flash EEPROM data memory does not have permanent read-protection or write-protection features like those available for the EEPROM program memory. However, the Data Memory Write Key Register provides a way to "lock" the data written to the data memory.

#### 9.3.3 Data Memory Control and Status Register (DMCSR)

The DMCSR register is a byte-wide, read/write register used with the flash EEPROM data memory or ISP flash EEPROM program memory. When writing to this register, all reserved bits must be written with 0 for the memory to operate properly. There are two status/control bits, as shown in the register format below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | ERASE | DMBUSY | ZEROWS | Reserved |

ZEROWS Zero Wait-State Access. When cleared (0), the flash EEPROM data memory will be read in two cycles. When set (1), the flash EEPROM data memory will be read in one cycle.

DMBUSY Data Memory Busy. This bit is automatically set to 1 when the flash EEPROM data memory or the ISP flash EEPROM program memory is busy being programmed, and cleared to 0 at all other times. (The MSTAT.PGMBUSY is also set to 1 whenever the DMBUSY bit is set to 1.)

ERASE Erase ISP Flash Program Memory Page. When set (1) a valid write to the ISP flash EEPROM program memory will erase the entire ISP flash EEPROM program memory page pointed to by the write address rather than performing a write to the addressed memory location. This bit should be cleared to 0 and remain cleared after the write operation.

Upon reset, the DMCSR register is cleared to zero when the flash memory on the chip is in the idle state.

#### 9.3.4 Data Memory Prescaler Register (DMPSLR)

The DMPSLR register is a byte-wide, read/write register that selects the prescaler divider ratio for the EEPROM data memory programming clock. Before you write to the data memory for the first time, you should program the DMPSLR register with the proper prescaler value, an 8-bit value called FTDIV. The device divides the system clock by (FTDIV+1) to produce the data memory programming clock.

You should choose a value of FTDIV to produce a clock of the highest possible frequency that is equal to or just less than 200 kHz. Upon reset, this register is programmed by default with the value 63 hex (99 decimal), which is an appropriate setting for a 20 MHz system clock.

#### 9.3.5 Data Memory Start Time Reload Register (DMSTART)

The DMSTART register is a byte-wide read/write register that controls the program/erase start delay time. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you write to the data memory for the first time, you should program the DMSTART register with the proper prescaler value, an 8-bit value called FTSTART. The flash timing counter generates a delay of (FTSTART + 1) prescaler output clocks. The default value provides a delay time of 10µs when the prescaler output clock is 200kHz. Do not modify this register while program/erase operation is in progress.

Upon reset, this register resets to $01_{16}$ when the flash memory on the chip is in idle state.

#### 9.3.6 Data Memory Transition Time Reload Register (DMTRAN)

The DMTRAN register is a byte-wide read/write register that controls some program/erase transition times. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you write to the data memory for the first time, you should program the DMTRAN register with the proper prescaler value, an 8-bit value called FTTRAN. The flash timing counter generates a delay of (FTTRAN + 1) prescaler output clocks. The default value provides a delay time of 5 µs when the prescaler output clock is 200kHz. Do not modify this register while program/erase operation is in progress.

Upon reset, this register resets to $00_{16}$ when the flash memory on the chip is in idle state.

#### 9.3.7 Data Memory Programming Time Reload Register (DMPROG)

The DMPROG register is a byte-wide read/write register that controls the programming pulse width. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you write to the data memory for the first time, you should program the DMPROG register with the proper prescaler value, an 8-bit value called FTPROG. The flash timing counter generates a programming pulse width of (FTPROG + 1) prescaler output clocks. The default value provides a delay time of 30µs when the prescaler output clock is 200kHz. Do not modify this register while program/erase operation is in progress.

Upon reset, this register resets to $05_{16}$ when the flash memory on the chip is in idle state.

#### 9.3.8 Data Memory Erase Time Reload Register (DMERASE)

The DMERASE register is a byte-wide read/write register that controls the erase pulse width. This value is loaded into the upper 8 bits of the flash timing counter, and at the same time, $11_2$ is loaded into the lower 2 bits. Before you write to the data memory for the first time, you should program the DMERASE register with the proper prescaler value, an 8-bit value called FTER. The flash timing counter generates a erase pulse width of 4×(FTER + 1) prescaler output clocks. The default value provides a delay time of 1ms when the prescaler output clock is 200kHz. Do not modify this register while program/erase operation is in progress.

Upon reset, this register resets to $31_{16}$ when the flash memory on the chip is in idle state.

For mass erase, this value should be changed to $C7_{16}$ when the flash EEPROM data memory goes to idle mode.

#### 9.3.9 Data Memory End Time Reload Register (DMEND)

The DMEND register is a byte-wide read/write register that controls the delay time after a program/erase operation. This value is loaded into the lower 8 bits of the flash timing counter, and at the same time, $00_2$ is loaded into the upper 2 bits. Before you write to the data memory for the first time, you should program the DMEND register with the proper prescaler value, an 8-bit value called FTEND. The flash timing counter generates a delay of (FTEND + 1) prescaler output clocks. The default value provides a delay time of 5µs when the prescaler output clock is 200kHz. Do not modify this register while program/erase operation is in progress.

Upon reset, this register resets to $00_{16}$ when the flash memory on the chip is in idle state.

For mass erase, this value should be changed to $13_{16}$.

#### 9.3.10 Data Memory Prescaler Count Register (DMPCNT)

The DMPCNT register is a byte-wide read-only register that returns the value of the data memory prescaler counter.

FPCNT is the flash timing prescaler present count value.

#### 9.3.11 Data Memory Timer Count Register (DMCNT)

The DMCNT register is a word-wide read-only register that returns the data memory timing counter value. The reserved bits return $000000_2$.

FTCNT[0:9] is the flash timer present count value.

#### 9.3.12 Data Memory Write Key Register (DMKEY)

The DMKEY register is a byte-wide, read/write register that provides a way to "lock" the data contained in the EEPROM data memory. Upon reset, the register is automatically set to C9 hex, which is the key value. Writing to the EEPROM data memory is allowed as long as the DMKEY register contains this value. When the register contains any value other than C9 hex, writing the EEPROM data memory is disallowed.

To "lock" the current data stored in the data memory, write another value (such as 00 hex) to the DMKEY register. To "unlock" the data memory, write the value C9 hex to the DMKEY register.

**Note:** Operation of this register is different in from the PGMKEY register used with the program memory. It is not necessary to write the key value to DMKEY every time you write to the data memory.

### 9.4 ISP MEMORY

The In-System Program memory is part of the flash memory array that contains the flash EEPROM data memory. It is not possible to access the ISP memory while programming the flash EEPROM data memory or access the flash EEPROM data memory while programming the ISP memory. The 1.5K bytes of ISP memory resides in the address range of E000-E5FF and is used for storing the boot ROM. The ROM contains the code that performs in-system programming, and is programmed at the factory. In ISP mode, code execution starts at address E000.

The ISP program memory and flash EEPROM data memory share the same memory array, which makes it impossible to access one type of memory while the other is being programmed.

The ISP memory has the following features:

- 1.5K bytes flash EEPROM program memory
- Page size of 4 words, divided into two rows of 2 words each
- Odd and even bytes within a page can be erased separately
- 30µs programming pulse width per word
- Page mode erase with 1ms pulse, mass erase with 4ms pulse
- All erased memory bits read 1
- Fast read access time
- Requires valid key for program and erase to proceed
- Provide memory protection and security features for flash EEPROM program memory
- Security features may limit accesses to ISP memory
- Disable memory when address is out of range to prevent accessing data memory
- Mass erase only allowed in test modes
- Provide busy status during programming and erase
- Read/write accesses disabled during programming/erase
- Programming high voltage and timing generated on-chip

#### 9.4.1 Reading

The ISP flash EEPROM program memory read accesses can operate without wait cycles with a CPU clock rate of up to 20MHz in the normal mode. At higher clock rates, read accesses can operate with one wait state.

The programmed number of wait cycles used (either zero or one) is controlled by BIU Configuration (BCFG) register and the Static Zone 1 Configuration (SZCFG1) register. These registers are described in Section8.0.

#### 9.4.2 User-Coded Programming Routines

All program and erase operations must be preceded by writing the proper key to the program memory key register ISPKEY. The programming code can be in-system RAM, but cannot be from ISP flash EEPROM program memory or flash EEPROM data memory as accesses within these ranges are

not permitted while ISP flash EEPROM program memory is being programmed.

The ISP flash memory is divided into 192 pages, each page containing 4 words (each 16 bits wide). Each page is further divided into two rows. Erase is carried out one page at a time, whereas programming is carried out one row (or one partial row) at a time.

Once an erase or programming operation is started, the PGMBUSY bit in the MSTAT register is automatically set, and then cleared when the operation is complete. All high-voltage pulses and timing needed for programming and erasing are provided internally. The program memory cannot be accessed while the PGMBUSY bit is set.

**Erase Procedure**

Erasing a page requires the following code sequence:

1. Verify that the MSTAT.PGMBUSY bit is cleared.
2. Set the DMCSR.ERASE bit to 1.
3. Locally disable interrupts.
4. Write proper key value to the ISPKEY register.
5. Write to any valid page to be erased.
6. Re-enable interrupts disabled in Step 3.
7. Set the DMCSR.ERASE bit to 0.

### 9.4.3 Programming Procedure

Programming is done by writing one byte or word at a time and should be done on already erased memory.

Programming the ISP flash EEPROM program memory requires the following code sequence:

1. Verify that the MSTAT.PGMBUSY bit is cleared.
2. Locally disable interrupts.
3. Write proper key value to the ISPKEY register.
4. Write a byte or word to the addressed location.
5. Re-enable interrupts disabled in Step 2.

Programmed values can be verified through normal read operations.

If a reset occurs in the middle of an erase or programming operation, the operation is terminated. The reset is extended until the flash EEPROM memory returns to the idle state.

### 9.4.4 Erase and Programming Timing

The program and erase timing are controlled by the flash EEPROM data memory logic.

### 9.4.5 Memory Control and Protection Features

The last 8 bytes of the ISP memory are reserved for special functions and some of these bytes provide memory protection and security for the flash EEPROM program memory. Read and various types of write protection are provided.

During the reset stretch period, bytes located at E5FE and E5FF are read out to the FLCTRL2 and FLSEC registers respectively. Upon reset and before an instruction fetch, bytes located at E5FC and E5FD are read out to the FLCTRL2 and FLCTRL1 registers respectively. Parts of FLCTRL2 register are loaded at different times.

**E5FE Byte**

Upon reset of the chip, the byte located at E5FE is read into the FLCTRL2 register. It can be written in the ISP or test environments. It can also be written in the IRE environment through a byte write instruction when the write instruction is anywhere within the user boot ROM area (defined above) except for the last two words. When the user boot ROM area has been disabled, this word cannot be programmed in the IRE environment. Note that when this word is erased for reprogramming, the other words in the same page must first be saved, and then re-programmed.

| 7 5 | 4 2 | 1 0 |
|---|---|---|
| EMPTY | Reserved | CODEAREA[9:8] |

CODEAREA[9:8]
: The 2 least significant bits in address E5FE contains the two most significant bits of the 10-bit CODEAREA field. The description of CODEAREA is shown in the E5FC section.

EMPTY
: The EMPTY status indicates if the flash EEPROM program memory array is empty or not. It is located in the 3 most significant bits in address E5FE. When two or more bits in the EMPTY field are set, the flash EEPROM program memory is empty. Upon reset of the device and the environment select pins are all high, the device operates in ISP environment rather than IRE environment. After the program memory has been filled with user code, this field should be cleared to $000_2$.

000, 001, 010, 100: Program memory contains user code

011, 101, 11x: Program memory is empty, do not start up in IRE

**E5FF Byte**

Upon reset, the byte located in the E5FF address is read into the FLSEC register. This byte cannot be written to in the IRE environment. The format of the E5FF byte is shown below:

| 7 4 | 3 0 |
|---|---|
| FROMWR | FROMRD |

The FROMRD and FROMWR fields in address location E5FF respectively provide read and write security to the flash EEPROM program memory array while executing instructions in all environments except IRE. The user should always write $0000_2$ to enable security feature.

0000, 0001, 0010, 0100, 1000: Security feature enabled

0011, 0101, 011x, 1001, 101x, 11xx: Security feature disabled

FROMRD
: Upon reset of the chip, read security is enabled and 0000 is returned in all environments except IRE. The internal program code can only be executed in the IRE environment when read security is activated.

FROMWR
: Upon reset of the chip, write security is enabled and program and erase operations to the flash EEPROM program memory in either programming modes are prevented.

Once read/write security is enabled, the odd numbered bytes from address E5F9 to E5FF cannot be erased. Once a security feature has been enabled, it cannot be undone. To prevent the security status from being erased, the ISP and data memory array cannot be mass erased.

**Note:** In flash memory test mode, this condition also prevents the odd numbered bytes of the high endurance flash EEPROM data memory (F001 to F07F) from being erased;

however, the even numbered bytes of the high endurance flash EEPROM data memory (F000 to F07E) and the ISP flash EEPROM program memory (E000 to E5FE) can be erased.

Read/write is overridden through PADX.

**E5FC Byte**

Upon reset of the chip, E5FC is read into the FLCTRL2 register. The byte at E5FC is written in the ISP or test environments, or in the IRE environment through a byte-write instruction when the write instruction is anywhere within the user boot ROM area except for the last two words. When the user boot ROM area has been disabled by having a value of $7F_{16}$ in BOOTAREA, this word cannot be programmed in the IRE environment. Note that when this word is erased for re-programming, the other words in the same page must first be saved, and then re-programmed also. The E5FC register format is shown below:

| 7 | 0 |
|---|---|
| CODEAREA[7:0] | |

This byte contains the lowest 8 bits of the CODEAREA field. When appended to the left with the lowest 2 bits in the address E5FE, it forms the complete CODEAREA field, which provides write protection to all or part of the program memory, see Figure3. When write security is not enabled and CODEAREA does not contain the value $3FF_{16}$, the program memory range from (CODEAREA×128) to 1FFFF is considered as protected user code area and cannot be written. The minimum protected memory range is therefore 256 bytes when CODEAREA contains the value 3FE. Note that the C000-FFFF memory range is not considered as program memory and is not protected by CODEAREA.

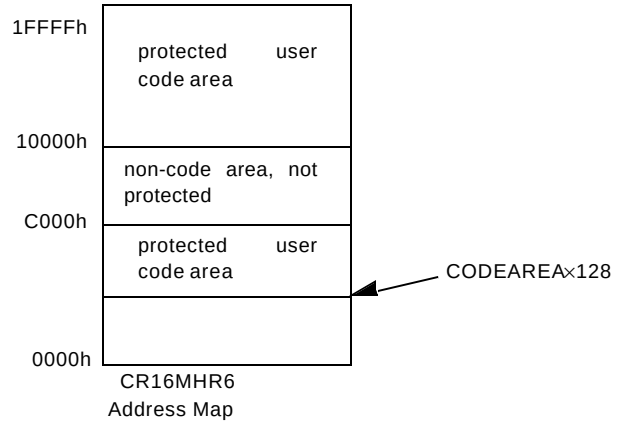



**Figure 3. Memory Protection through CODEAREA**

When CODEAREA contains the value $3FF_{16}$, write protection is disabled. When the user code area overlaps into the user boot ROM area, the overlap area is governed by a more restrictive write protection feature, which is the user boot ROM area. When write security has been enabled, the entire program memory area is already write protected in all environments.

Note that when a new value is written into CODEAREA, write protection controlled by CODEAREA is updated after the next device reset.

**E5FD Byte**

Upon the reset of the chip, the byte located at the E5FD address is read into the FLCTRL1 register. This byte can only be written in the ISP or test environments but not in the IRE environment. If this byte is erased for re-programming, the user must first save the other bytes in the same page, and then re-program those bytes. The format of the E5FD byte is shown below:

| 7 | 6 0 |
|---|---|
| Reserved | BOOTAREA |

BOOTAREA provides write protection to part of the program memory, see Figure4. When the write security feature is not enabled and BOOTAREA does not contain the value $7F_{16}$, then the program memory range from 0 to (BOOTAREA*128)+127 is considered as user boot ROM area and cannot be written to. The maximum protected memory range is therefore 16K-127 bytes when BOOTAREA contains the value $7E_{16}$.

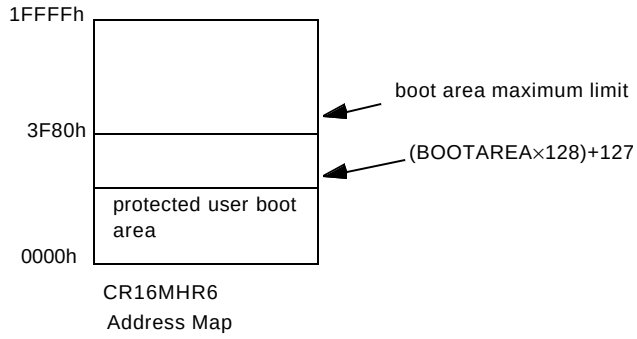



**Figure 4. Memory Protection through BOOTAREA**

When BOOTAREA contains the value $7F_{16}$, write protection is disabled. When write security has been enabled, the entire program memory area is already write protected in all environments.

Note that when a new value is written into BOOTAREA, write protection controlled by BOOTAREA is updated after the next device reset.

### 9.4.6 Test Mode

The ISP flash EEPROM program memory test mode allows direct access to the flash memory from the device pins, and bypasses the CR16B core. This test mode also accesses the flash memory cells that are not used in data memory (three out of four bytes in each page).

### 9.4.7 Flash Program Memory Control Register 1 (FLCTRL1)

The FLCTRL1 register is a read-only byte-wide register. The value of this register is loaded from memory address $E5FD_{16}$ when the chip comes out of reset. The BOOTAREA field defines a user boot ROM area to be write protected. The Flash EEPROM Program Memory Control Register 1 format is shown below:

| 7 | 6 0 |
|---|---|
| Reserved | BOOTAREA |

When BOOTAREA has any value other than $7F_{16}$, then the memory at 0 to (BOOTAREA×128)+15 is considered as user boot ROM area and is write protected. When it has a value of $7F_{16}$, then there is no user boot ROM area to be write protected

### 9.4.8 Flash Program Memory Control Register 2 (FLCTRL2)

The FLCTRL2 register is a read-only word-wide register. The value of this register is loaded from memory addresses $E5FC_{16}$ and $E5FE_{16}$ when the chip comes out of reset. When the device starts execution, the EMPTY bit indicates whether the flash EEPROM program memory is empty of not, and selects the chip to be in IRE or ISP environment if the external environment pins are all high. The CODEAREA field defines a user code area to be write protected. The Flash EEPROM Program Memory Control Register 2 format is shown below:

| 15 13 | 12 10 | 9 0 |
|---|---|---|
| EMPTY | Reserved | CODEAREA |

EMPTY When the bits are either $011_2$, $101_2$, $110_2$, or $111_2$, and if the device's environment select pins are all high, the device will come out of reset in ISP environment instead of IRE environment.

CODEAREA When it has any value other than $3FF_{16}$, then the memory (CODEAREA×128) to $1FFFF_{16}$ is considered as user code area and is write protected. When it has a value of $3FF_{16}$, then there is no code protection area to be write protected.

### 9.4.9 Flash Program Memory Security Register (FLSEC)

The FLSEC register is a read-only byte-wide register. When the chip comes out of reset, the value of this register is loaded from memory address $E5FF_{16}$. The FROMRD and FROMWR field control the read and write security of the flash EEPROM program memory respectively. The Flash EEPROM Program Memory Security register format is shown below:

| 7 4 | 3 0 |
|---|---|
| FROMWR | FROMRD |

0000, 0001, 0010, 0100, 1000: Security feature enabled

0011, 0101, 011x, 1001, 101x, 11xx: Security feature disabled

FROMRD When read security feature is enabled, the flash EEPROM program memory can only be read in IRE environment, but will return $0000_{16}$ in other environments; also, erase to odd numbered bytes from address $E5F9_{16}$ to $E5FF_{16}$ and mass erase to ISP and flash EEPROM data memory array are ignored unless PADX is activated (see security override below).

FROMWR Unless PADX is activated (see override below), when write security feature is enabled, all further writes and erases to flash EEPROM program memory, erase to odd numbered bytes from address $E5F9_{16}$ to $E5FF_{16}$, and mass erase to ISP and flash EEPROM data memory array are ignored.

### 9.4.10 ISP Memory Write Key Register (ISPKEY)

The In-System-Programming Memory Write Key (ISPKEY) register is a byte-wide, write-only register. It contains the enable key to enable writes to ISP flash EEPROM program memory. A value of $6A_{16}$ must be written to this register immediately preceding every write to the ISP flash EEPROM program memory for the flash write operation to proceed, otherwise any other write operation will clear the key (the only exception is that the subsequent write is another write to this key register with the proper key, in which case the key is still set). A read always returns $FF_{16}$. Engineering note: on reset, the write enable status that is generated as a result of a write to this key register is cleared. The ISP Memory Write Key register format is shown below:

| 7 0 |
|---|
| ISPKYVAL |

ISPKYVAL is the ISP Flash Program Memory Write Enable Key Value.

# 10.0 Interrupts

The Interrupt Control Unit (ICU31L) receives interrupt requests from internal and external sources and generates interrupts to the CPU. Interrupts from the timers, USARTs, MICROWIRE/SPI interface, Multi-Input Wake-Up, and A/D converter are all maskable interrupts. The highest-priority interrupt is the Non-Maskable Interrupt (NMI), which is triggered by a falling edge received on the NMI input pin. The NMI pin is not available on the 44-pin packages.

## 10.1 INTERRUPT OPERATION

An *exception* is an event that temporarily stops the normal flow of program execution and causes execution of a separate service routine. Upon completion of the service routine, execution of the interrupted program continues from the point at which it was stopped.

There are two kinds of exceptions, called *traps* and *interrupts*. A trap is the result of some action or condition in the program itself, such as execution of an Exception (EXCP) instruction. An interrupt is a CPU-external event, such as a signal received on a Multi-Input Wake-Up input or a request from an on-chip peripheral module for service.

The operation of traps is beyond the scope of this data sheet. For information on traps, and for additional detailed information on interrupts not provided in this data sheet, please refer to the CompactRISC CR16B Programmer's Reference Manual.

### 10.1.1 Interrupt Operation Summary

When an interrupt occurs, the on-chip hardware performs the following steps:

1. Decrements the Interrupt Stack Point (ISP) by four.
2. Saves the contents of the Program Counter (PC) and Processor Status Register (PSR) on the interrupt stack.
3. Clears the I, P, and T bits in the Processor Status Register (PSR). These are the Global Maskable Interrupt Enable bit, Trace Trap Pending bit, and Trace bit, respectively.
4. Reads the interrupt vector from the Interrupt Vector Register (IVCT).
5. Combines the interrupt vector with the value in the Interrupt Base (INTBASE) register to obtain an address in the Interrupt Dispatch Table, and loads the dispatch table entry into the Program Counter (PC).

From this point onward, the CPU executes the interrupt service routine. The service routine ends with a Return from Exception (RETX) instruction. This returns the CPU to the interrupted program. The CPU restores the contents of the PC and PSR registers from the stack and increments the Interrupt Stack Pointer by four.

### 10.1.2 Service Routine Addresses

When an interrupt or trap occurs, the CPU executes a service routine. There are different service routines for different interrupts and traps. Each service routine may reside anywhere in program memory. The starting addresses of the service routines are contained in a table called the Dispatch Table. Entries in the table are organized in the order shown in Table10.

**Table 10 Dispatch Table Entries**

| |
|---|
| 0: Reserved |
| 1: NMI |
| 2: Reserved |
| 3: Reserved |
| 4: Reserved |
| 5: SVC (Supervisor Call Trap) |
| 6: DVC (Divided by Zero Trap) |
| 7: FLG (Flag Trap) |
| 8: BPT (Breakpoint Trap) |
| 9: TRC (Trace Trap) |
| 10: UND (Undefined Instruction Trap) |
| 11: Reserved |
| 12: Reserved |
| 13: Reserved |
| 14: Reserved |
| 15: Reserved |
| 16: INT0 (Reserved) |
| 17: INT1 (Flash EEPROM Program Memory) |
| 18: INT2 (Reserved) |
| 19: INT3 (Reserved) |
| 20: INT4 (Reserved) |
| 21: INT5 (ADC) |
| 22: INT6 (MIWU Interrupt 3) |
| 23: INT7 (MIWU Interrupt 2) |
| 24: INT8 (MIWU Interrupt 1) |
| 25: INT9 (MIWU Interrupt 0) |
| 26: INT10 (USART 2 Tx) |
| 27: INT11 (USART 1Tx) |
| 28: INT12 (Reserved) |
| 29: INT13 (MICROWIRE/SPI Rx/TX) |
| 30: INT14 (ACCESS.bus) |
| 31: INT15 (USART 2 Rx) |
| 32: INT16 (USART 1 Rx) |
| 33: INT17 (Reserved) |
| 34: INT18 (CAN) |
| 35: INT19 (Reserved) |
| 36: INT20 (Reserved) |
| 37: INT21 (Reserved) |

**Table 10 Dispatch Table Entries**

| |
|---|
| 38: INT22 (Reserved) |
| 39: INT23 (VTUD Interrupt Request 4) |
| 40: INT24 (VTUD Interrupt Request 3) |
| 41: INT25 (VTUD Interrupt Request 3) |
| 42: INT26 (VTUD Interrupt Request 1) |
| 43: INT27 (T2B Timer 2 Interrupt B) |
| 44: INT28 (T2A Timer 2 Interrupt A) |
| 45: INT29 (T1B Timer 1Interrupt B) |
| 46: INT30 (T1A Timer 1Interrupt A) |
| 47: INT31 (RTI Timer 0) |

Each entry in the Dispatch Table consists of two bytes that provide bits 1 through 16 of the starting address of the corresponding service routine. The full 21-bit address of a service routine is reconstructed by adding a leading 0 and a trailing 0 to the 16-bit table entry.

The INTBASE register is a pointer to the Dispatch Table. Upon reset, the initialization software must write the starting address of the Dispatch Table to the INTBASE register, a 21-bit register with the five most significant bits and the least significant bit always equal to 0. It is typically kept in the flash EEPROM program memory. The Dispatch Table is 48 words long.

Each interrupt or trap source has an associated vector number ranging from 0 to 31, as indicated in Table10. When an interrupt occurs, the hardware multiplies the vector by 2, adds the result to the contents of the INTBASE register, and uses the resulting address to obtain the service routine starting address from the corresponding entry in the Dispatch Table. This address is placed in the Program Counter so that the CPU begins executing the interrupt service routine.

Figure5 summarizes the method used by the device to generate the starting address of a service routine.

#### 10.1.3 Stack Usage

When an interrupt occurs, the CPU automatically preserves the contents of the Program Counter (PC) and Processor Status Register (PSR) by pushing them on the interrupt stack and decrementing the Interrupt Stack Pointer by four. The service routine ends with a Return from Exception (RETX) instruction, which returns control to the interrupted program by restoring the PC and PSR values and incrementing the Interrupt Stack Pointer (ISP) by four.

Prior to using any interrupts, the Interrupt Stack Pointer (ISP) must be initialized so that it points to a space in RAM where the interrupt stack will be kept. The stack grows downward in memory (toward address zero) when an interrupt occurs and items are pushed onto the stack. The stack shrinks upward in memory when an interrupt service routine ends and items are popped from the stack.

Many routines need to use the general-purpose registers R0 through R13. To preserve the existing register contents, a routine can save register contents on the program stack upon start of the routine and restore the register contents prior to completion of the routine. The software can also use the program stack to transfer data parameters from one routine to another when the parameters are too large to easily fit into the registers. A high-level language typically allocates the local (non-static) variables on the stack.

The pointer to the program stack is the SP register, which must be initialized prior to any register save/restore operations or data transfer operations. Using the program stack, an interrupt routine needs to initially save the contests of all registers that it uses, and restore those register contents before returning to the interrupted program.

### 10.2 NON-MASKABLE INTERRUPT

A non-maskable interrupt is triggered by a falling edge on the $\overline{\text{NMI}}$ input pin, which generates a software trap. The $\overline{\text{NMI}}$ pin is an asynchronous input with Schmitt trigger characteristics and an internal synchronization circuit. Therefore, no external synchronizing is needed.

Upon reset, the non-maskable interrupt is disabled and should remain disabled until the software initializes the interrupt table, interrupt base, and interrupt stack pointer. It can be enabled by setting either of two control bits in the External NMI Control/Status (EXNMI) register. The two bits are called the EN (Enable) bit and the ENLCK (Enable and Lock) bit.

The EN bit enables the NMI trap until an NMI trap event or a reset occurs. An NMI trap automatically resets the EN bit. Using this bit to enable the NMI trap is intended for applications where the $\overline{\text{NMI}}$ pin is toggled frequently but nested NMI traps are not needed. The trap service routine should re-enable the NMI trap by setting the EN bit before returning to the main program.

The ENLCK bit enables the NMI trap and locks it in the enabled state. In other words, it leaves the NMI trap enabled even after the trap occurs. It can be cleared only by a reset operation. After the bit is set, an NMI trap is triggered by each falling edge on the $\overline{\text{NMI}}$ pin, allowing nested NMI traps.

To use the EN bit, the ENLCK must remain cleared to 0. Otherwise, the EN bit is ignored.

### 10.3 MASKABLE INTERRUPTS

Maskable interrupts can be enabled or disabled under software control. There are 31 level-triggered maskable interrupt sources (including some reserved for future expansion), organized into levels of priority. If more than one interrupt event occurs at any given time, the interrupt source with the highest priority is serviced first. The others must wait until the highest-priority interrupt is serviced and is no longer pending.

Figure11 lists the maskable interrupt sources of the device in order of priority, from the highest-priority interrupt (IRQ31) to the lowest (IRQ0).

To enable a maskable interrupt, the enable bit must be set in the applicable peripheral module and also in the appropriate Interrupt and Enable Mask register, IENAM0 or IENAM1. In addition, both the Global Maskable Interrupt Enable bit (I) and the Local Maskable Interrupt Enable bit (E) must be set to 1 in the PSR register. If either one of these bits is 0, then all maskable interrupts are disabled. The CR16B core supports IRQ0, but ICU31L reserves IRQ0 so that it is not connected to any interrupt source.

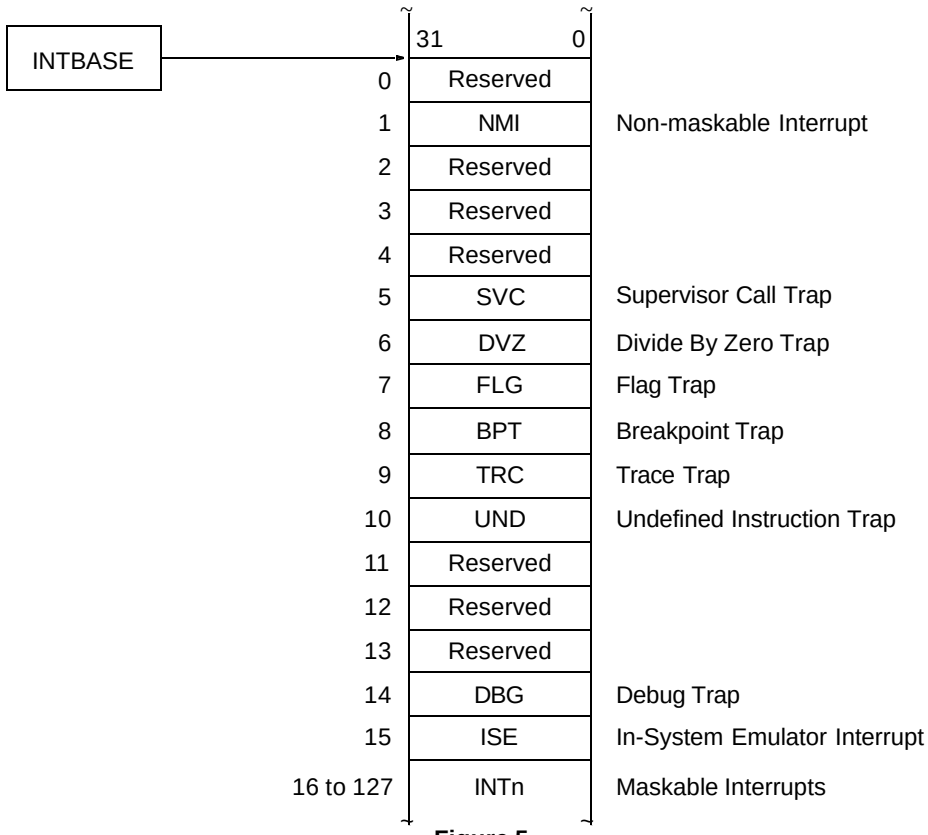



**Figure 5.**

**Table 11 Maskable Interrupt Priority List**

| Interrupt Request | Source |
|---|---|
| IRQ31 | RTI (Timer 0), highest priority |
| IRQ30 | T1A (Timer 1 input A) |
| IRQ29 | T1B (Timer 1 input B) |
| IRQ28 | T2A (Timer 2 input A) |
| IRQ27 | T2B (Timer 2 input B) |
| IRQ26 | VTUA (VTU Interrupt Request 1) |
| IRQ25 | VTUB (VTU Interrupt Request 2) |
| IRQ24 | VTUC (VTU Interrupt Request 3) |
| IRQ23 | VTUD (VTU Interrupt Request 4) |
| IRQ22-IRQ19 | Reserved |
| IRQ18 | CAN |
| IRQ17 | Reserved |
| IRQ16 | USART1 Rx |
| IRQ15 | USART2 Rx |
| IRQ14 | ACCESS.bus |
| IRQ13 | MICROWIRE/SPI Rx/Tx |
| IRQ12 | Reserved |
| IRQ11 | USART1 Tx |
| IRQ10 | USART2 Tx |
| IRQ9 | MIWU16 Interrupt 0 |

**Table 11 Maskable Interrupt Priority List**

| Interrupt Request | Source |
|---|---|
| IRQ8 | MIWU16 Interrupt 1 |
| IRQ7 | MIWU16 Interrupt 2 |
| IRQ6 | MIWU16 Interrupt 3 |
| IRQ5 | ADC |
| IRQ4-IRQ2 | Reserved |
| IRQ1 | Flash Program Memory |
| IRQ0 | Reserved, lowest priority |

Both the E bit and I bit can be controlled with the Load Processor Register (LPR) instruction. In addition, the E bit is easily changed by executing the Enable Interrupts (EI) or Disable Interrupts (DI) instruction. Using the EI and DI instructions avoids the possibility of an interrupt occurring within a read-modify-write operation on the PSR register.

## 10.4 INTERRUPT REGISTERS

The Interrupt Control Unit uses the following interrupt control and status registers:

— Non-Maskable Interrupt Status Register (NMISTAT)
— Non-Maskable Interrupt Status Monitor Reg. (NMIMNTR)
— External NMI Control/Status Register (EXNMI)
— Interrupt Enable and Mask Register 0 (IENAM0)
— Interrupt Enable and Mask Register 1 (IENAM1)
— Interrupt Vector Register (IVCT)

— Interrupt Status Register 0 (ISTAT0)
— Interrupt Status Register 1 (ISTAT1)
— Interrupt Debug Register (IDBG)

The following CPU core registers are also used in processing interrupts:

— Interrupt Stack Pointer (ISP)
— Interrupt Base Register (INTBASE)

### 10.4.1 Non-Maskable Interrupt Status Register (NMISTAT)

The NMISTAT register is a byte-wide, read-only register that holds the current pending status of the Non-Maskable Interrupt (NMI). This register is cleared upon reset. It is also cleared each time it is read. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | EXT |

EXT — External Non-Maskable Interrupt Request. When set to 1 by the hardware, it indicates an external Non-Maskable Interrupt request has occurred. See the description of the EXNMI register below for more information.

### 10.4.2 External NMI Control/Status Register (EXNMI)

The EXNMI register is a byte-wide, read/write register that shows the current state of the $\overline{\text{NMI}}$ pin and also allows the NMI trap to be enabled by setting either the EN bit or the ENLCK bit. Both of these bits are cleared upon reset. When the software writes to this register, it must write 0 to all reserved bit positions for the device to function properly. EN, ENLCK, and TST are cleared upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | ENLCK | PIN | EN |

EN — Enable NMI Trap. When set to 1, $\overline{\text{NMI}}$ traps are enabled and falling edge on the $\overline{\text{NMI}}$ pin generates a NMI trap. Each occurrence of an NMI trap automatically clears the EN bit. The trap service routine should set the EN bit to 1 before returning control to the interrupted program. When EN is cleared to 0, NMI traps are disabled unless they are enabled with the ENLCK bit. When the ENLCK bit is set to 1, the EN bit is ignored.

PIN — $\overline{\text{NMI}}$ Pin. This bit shows the current state of the $\overline{\text{NMI}}$ input pin (without logical inversion). A 1 indicates a high level and a 0 indicates a low level on the pin. This is a read-only bit. In a write operation, the value written to this bit position is ignored.

ENLCK — Enable and Lock NMI Trap. When set to 1, NMI traps are enabled and locked in the enabled state. Each falling edge on the $\overline{\text{NMI}}$ pin generates a NMI trap, even if a previous NMI trap has occurred and is still being processed. When ENLCK is cleared to 0, NMI traps are disabled unless they are enabled with the EN bit.

### 10.4.3 Interrupt Vector Register (IVCT)

The IVCT register is a byte-wide, read-only register that contains the encoded value of the enabled and pending maskable interrupt with the highest priority. The on-chip hardware automatically updates this field whenever there is a change in the highest-priority enabled and pending maskable interrupt. The CPU reads this register during an interrupt acknowledge core bus cycle to determine where to begin executing the interrupt service routine. The register contents are guaranteed to be valid at that time. The register is not guaranteed to contain valid data during a hardware update operation. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | INTVECT | | | | | |

INTVECT — Interrupt Vector. This 6-bit field contains the encoded value of the enabled and pending maskable interrupt with the highest priority. For example, if interrupts IRQ1 and IRQ6 are both enabled and pending, the higher-priority interrupt is IRQ6. As a result the 6 bit interrupt vector is 010110.

### 10.4.4 Interrupt Enable and Mask Register 0 (IENAM0)

The IENAM0 register is a word-wide, read/write register that enables or disables the individual interrupts IRQ0 through IRQ15. The register format is shown below.

| 15 | 0 |
|---|---|
| IENA(15:0) | |

A bit set to 1 enables the corresponding interrupt. A bit cleared to 0 disables the corresponding interrupt. Upon reset, this register is initialized to FFFF hex.

### 10.4.5 Interrupt Enable and Mask Register 1 (IENAM1)

The IENAM0 register is a word-wide, read/write register that enables or disables the individual interrupts IRQ16 through IRQ31. The register format is shown below.

| 15 | 0 |
|---|---|
| IENA(31:16) | |

A bit set to 1 enables the corresponding interrupt. A bit cleared to 0 disables the corresponding interrupt. Upon reset, this register is initialized to FFFF hex.

### 10.4.6 Interrupt Status Register 0 (ISTAT0)

The ISTAT0 register is a word-wide, read-only register that indicates which maskable interrupt inputs to the ICU31L (IRQ0 through IRQ15) are currently active. The register format is shown below.

| 15 | 0 |
|---|---|
| IST(15:0) | |

IST(15:0) — Interrupt Status bits. Each bit indicates the current status of an interrupt input to the ICU31L, corresponding to interrupts IRQ0 through IRQ15. A bit set to 1 indicates an active interrupt input, even when the interrupt is masked out by the IENAM0 register. A bit cleared to 0 indicates an inactive interrupt input.

### 10.4.7 Interrupt Status Register 1 (ISTAT1)

The ISTAT1 register is a word-wide, read-only register that indicates which maskable interrupt inputs to the ICU31L (IRQ16 through IRQ31) are currently active. The register format is shown below.

| 15 | 0 |
|---|---|
| IST(31:16) | |

IST(31:16) Interrupt Status bits. Each bit indicates the current status of an interrupt input to the ICU31L, corresponding to interrupts IRQ16 through IRQ31. A bit set to 1 indicates an active interrupt input, even when the interrupt is masked out by the IENAM0 register. A bit cleared to 0 indicates an inactive interrupt input.

### 10.4.8 Interrupt Debug Register

The IDBG register is a word-wide read-only register, which contains various status information of the ICU31L. The lowest 6 bits contain the INTVECT value during the last read from address FE00. The next 6 bits contain the INTVECT value when a maskable interrupt request is sent to the CR16B core. Upon reset, this register is set to 0000 hex.

## 10.5 INTERRUPT PROGRAMMING PROCEDURES

The following subsections provide information on initializing the device for interrupts, clearing interrupts, and nesting interrupts.

### 10.5.1 Initialization

Upon reset, all interrupts are disabled. To program the device for interrupt operation and to enable interrupts, use the following procedure in the application software:

1. Set the Interrupt Stack Pointer (ISP)
2. Load the INTBASE register so that it points to the base of the Interrupt Dispatch Table.
3. Perform any required preparation steps for the interrupt service routines.
4. Initialize the peripheral devices that can generate interrupts and set their respective interrupt enable bits.
5. Set the relevant bits in the interrupt mask registers (IENAM0 and IENAM1)
   Note: The MIWU16 interrupts have no local interrupt enable bits, which means you can only disable the MIWU16 interrupts if you clear the specific bits in the IENAM register.
6. Use the Load Processor Register (LPR) instruction to set I bit in the PSR register.
7. When the device is ready to execute interrupts, set the E bit in the PSR register by executing the Enable Interrupts (EI) instruction.

Once maskable interrupts are enabled by setting the E and I bits, you can disable and re-enable all maskable interrupts locally by using the Enable Interrupts (EI) and Disable Interrupts (DI) instructions, which set and clear the E bit.

### 10.5.2 Clearing Interrupts

Clearing an interrupt request before it is serviced may cause a spurious interrupt because the CPU may detect an interrupt not reflected in the Interrupt Vector (IVCT) register. To ensure reliable operation, clear interrupt requests only while interrupts are disabled.

Changing the polarity of an interrupt input (for example, in the Multi-Input Wake-Up module) can cause a spurious interrupt, and therefore should be done only while interrupts are disabled.

For the same reason, clearing an enable bit in a peripheral module should be carried out only while the interrupt is disabled.

### 10.5.3 Nesting Interrupts

Interrupts may be nested, or in other words, an interrupt service routine can itself be interrupted by a different interrupt source. There is no hardware limitation on the number of interrupt nesting levels. However, the interrupt stack must not be allowed to overflow its allocated memory space.

Unless specifically enabled by the software, nested interrupts will not occur. When the CPU acknowledges an interrupt, the I bit in the PSR register is automatically cleared to 0 for the duration of the service routine, disabling any further maskable interrupts.

To allow nested interrupts, an interrupt service routine should first set or clear the respective interrupt enable bits to specify which peripherals will be allowed to interrupt the current service routine. The present interrupt routine should be disabled (or interrupt pending bit cleared). The service routine should then set the PSR.I bit to 1, thus enabling maskable interrupts. This bit can be controlled with the Store Processor Register (SPR) and Load Processor Register (LPR) instructions.

**Note:**
Clearing the pending bit of the current interrupt should not be immediately followed by enabling further interrupts by setting the I bit in the PSR register. Wait states must be inserted into the software after clearing the interrupt pending bit and before another interrupt. Placing a NOP instruction will perform this instruction. This is because the instruction which resets the pending bit may not yet be finished when the interrupts are already enabled again by setting the I bit in the PSR register. To avoid this situation the user has to make sure that prior to enabling the interrupt an additional instruction is inserted. This could look like the example below:

```
SBITi   $0, T1ICRL # clear pending bit
NOP                # NOP instruction
MOVW $0x0a00, r0 # enable further interrupts
LPR     r0, psr
```

A CBITi or SBITi instruction may be used to clear the interrupt pending bit. In such cases, a spurious interrupt may occur.

# 11.0 Power Management

The Power Management Module (PMM) improves the efficiency of the device by changing the operating mode (and therefore the power consumption) according to the required level of device activity.

The device can operate in any of four power modes:

— Active
— Power Save
— Idle
— Halt

Table12 summarizes the main properties of the four operating modes: the state of the high-frequency oscillator (on or off), the type of clock used by most modules, and the clock used by the Timing and Watchdog Module (TWM).

**Table 12 Power Mode Operating Summary**

| Mode | High-Frequency Oscillator | Clock Used | TWM Clock |
|---|---|---|---|
| Active | On | Main Clock | Slow Clock |
| Power Save | On or Off | Slow Clock | Slow Clock |
| Idle | On or Off | None | Slow Clock |
| Halt | Off | None | None |

The low-frequency oscillator continues to operate in all four modes and power must be provided continuously to the device power supply pins. In the Halt mode, however, the internal SLCLK does not toggle, and as a result, the TWM timer and Watchdog Module do not operate. For the Power Save and Idle modes, the high-frequency oscillator can be turned on or off under software control, as long as the low-frequency oscillator is used.

## 11.1 ACTIVE MODE

In the Active mode, all device modules are fully operational. This is the operating mode upon reset. Most device modules use the clock generated by the high-frequency clock oscillator. The clock rate is determined by the external crystal network.

Power consumption in the Active mode can be reduced by selectively disabling unused modules and/or by executing the WAIT instruction. When WAIT is executed, the core stops executing new instructions and waits for an interrupt.

## 11.2 POWER SAVE MODE

In the Power Save mode, all device modules operate off the low-frequency clock. If the low-frequency clock is generated from an external crystal network, the high-frequency clock oscillator can be turned off to further reduce power consumption.

All on-chip modules continue to operate in the Power Save mode, with the SLCLK acting as their system clock. If this mode is entered by using the WAIT command, the CPU is inactive and waits for an interrupt to wake up. Otherwise, CPU continues to function normally at the lower frequency of the slow clock.

The low frequency of the clock in Power Save mode limits the operation of modules such as the USARTs, MICROWIRE interface, A/D Converter, and timers because they are driven by the slow clock rather than the normal high-speed clock. In order to work properly in Power Save mode, modules that perform real-time operations (such as a USART baud rate generator) must be reprogrammed to use the slower clock.

To reduce power consumption as much as possible, the program should execute a WAIT instruction during periods of CPU inactivity.

## 11.3 IDLE MODE

In the Idle mode, the clock is stopped for most of the device. Only the Power Management Module and Timing and Watchdog Module continue to operate. Both of these modules use the slow clock in this mode.

## 11.4 HALT MODE

In the Halt mode, all device clocks are disabled and the high-frequency oscillator is shut off. In this mode, the device consumes the least possible power while maintaining the device memory and register contents. The low-frequency oscillator continues to operate in this mode, but with very low power consumption due to its power-optimized design.

## 11.5 CLOCK INPUTS AND RESET CONFIGURATION

The system uses a high frequency clock Active mode. The source of this clock in the device is a high frequency crystal oscillator. The Oscillating High Frequency Clock (OHFC) input indicates to the Power Management Module (PMM) when this clock is stable and therefore usable. The clock can be used when OHFC is set to 1. The PMM does not use the high frequency clock when OHFC is set to 0. OHFC can be the output of a clock monitor or a strapped input signal to this module.

The low frequency clock is used in Power Save mode as the system clock source. In Idle mode, it is used as the clock source for the PMM and the TWM, both of which remain clocked. The clock source may be a low frequency clock oscillator or the prescaler from the high frequency clock.

The Oscillating Low Frequency Clock (OLFC) input indicates to the PMM when the clock is stable and therefore usable. When OLFC is set to 1, it indicates that the clock can be used. When OLFC is set to 0, the PMM does not use the low frequency clock. OLFC is generated by the “slow clock good” output of the Dual Clock and Reset module (CLK2RES).

While in reset (i.e., the reset signal is active), the PMM outputs the clock as long as the clock selected for use upon reset is stable (OHFC or OLFC are 1). If the clock selected is not stable, the PMM clock output remains low.

## 11.6 SWITCHING BETWEEN POWER MODES

Switching from a higher to a lower power consumption mode is accomplished by writing an appropriate value to the Power Management Control/Status Register (PMCSR). Switching from a lower power consumption mode to the Active mode is usually triggered by a hardware interrupt. Figure6 shows the four power consumption modes and the events that trigger a transition from one mode to another.

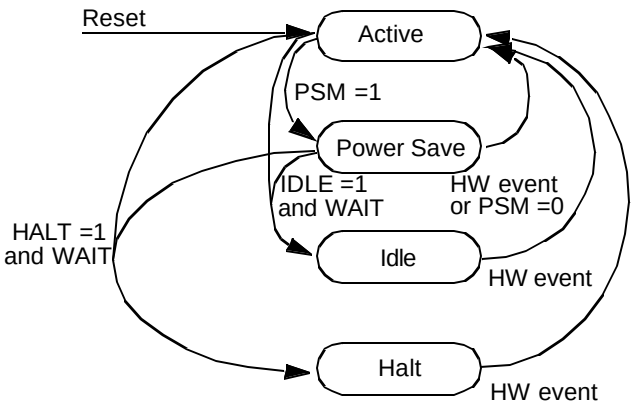



**Figure 6. Power Modes and Transitions**

Some of the power-up transitions are based on the occurrence of a wake-up event. An event of this type can be either a maskable interrupt or a non-maskable interrupt (NMI). All of the maskable hardware wake-up events are gathered and processed by the Multi-Input Wake-Up Module, which is active in all modes. Once a wake-up event is detected, it is latched until an interrupt acknowledge cycle occurs or a reset is applied.

A wake-up event causes a transition to the Active mode and restores normal clock operation, but does not start execution of the program. It is the interrupt service routine associated with the wake-up source (MIWU16 or NMI) that causes actual program execution to resume.

### 11.6.1 Power Management Control/Status Register (PMCSR)

The Power Management Control/Status Register (PMCSR) is a byte-wide, read/write register that controls the operating power mode (Active, Power Save, Idle, or Halt) and enables or disables the high-frequency oscillator in the Power Save and Idle modes. The two most significant bits, OLFC and OHFC, are read-only status bits controlled by the hardware. Upon reset, the non-reserved bits of this register are cleared. The format of the register is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OLFC | OHFC | WBPSM | Reserved | HALT | IDLE | DHF | PSM |

PSM — Power Save Mode. When this bit is 0, the device operates in the Active mode. Writing a 1 to this bit position puts the device into the Power Save mode, either immediately or upon execution of the next WAIT instruction, depending on the WBPSM bit.
The PSM bit can be set and cleared by the software. It is also cleared by the hardware when a hardware wake-up event is detected.

DHF — Disable High-Frequency Oscillator. This bit enables (0) or disables (1) the high-frequency oscillator in the Power Save or Idle mode. (The high-frequency oscillator is always enabled in Active mode and always disabled in Halt mode, regardless of this bit settings.) The DHF bit is cleared automatically when a hardware wake-up event is detected.

IDLE — Idle Mode. When this bit is set and the device is in Power Save mode, the device enters the Idle mode upon execution of a WAIT instruction. In order to enter the Idle mode directly from the Active mode, the WBPSM bit must be set before the WAIT instruction is executed.
The IDLE bit can be set and cleared by the software. When a hardware wake-up event is detected, this bit is cleared automatically and the device returns to the Active mode.

HALT — Halt Mode. When this bit is set and the device is in Idle mode, the device enters the Halt mode upon execution of a WAIT instruction. In order to enter the Halt mode directly from the Active mode, the WBPSM bit must be set before the WAIT instruction is executed.
The Halt bit can be set and cleared by the software. When a hardware wake-up event is detected, this bit is cleared automatically and the device returns to the Active mode.

WBPSM — Wait Before Entering Power Save Mode. When the CPU writes a 1 to the PSM bit, the WBPSM determines when the transition from Active to Power Save mode is done. If the WBPSM bit is 0, the switch to Power Save mode is initiated immediately; the PSM bit in the register is set to 1 upon completion of the switch to Power Save mode. If the WBPSM bit is 1, the device continues to operate in Active mode until the next WAIT instruction, and then enters the Power Save mode. In this case, the PSM bit is set to 1 immediately, even if a WAIT instruction has not yet been executed.
In the Active mode, the WBPSM bit must be set in order to enter the Idle or Halt mode.

OHFC — Oscillating High-Frequency Clock. This read-only bit indicates the status of the high-frequency clock. If this bit is 1, the high-frequency clock is available and stable. If this bit is 0, the high-frequency clock is either disabled, not available to the Power Management Module, or operating but not yet stable. The device can switch to the Active mode only when this bit is 1.

OLFC — Oscillating Low-Frequency Clock. This read-only bit indicates the status of the low-frequency (slow) clock. If this bit is 1, it indicates that the slow clock is running and stable. The slow clock can be either the prescaled fast clock (the default) or the external oscillator (if selected). The Dual Clock module will not allow a transition to the slow crystal mode unless the slow crystal is operating, so this bit should be 1 under normal circumstances.
The device can switch from the Active mode to the Power Save or Idle mode only if the OLFC bit is 1. There is no such restriction on switching to the Halt mode.

### 11.6.2 Active to Power Save Mode

A transition from the Active mode to the Power Save mode is accomplished by writing a 1 to the PMCSR.PSM bit. The transition to Power Save mode is either initiated immediately or upon execution of the next WAIT instruction, depending on the PMCSR.WBPSM bit.

For an immediate transition to Power Save mode (PMCSR.WBPSM=0), the CPU continues to operate using the low-frequency clock. The PMCSR.PSM bit is set to 1 when the transition to the Power Save mode is completed.

For a transition upon the next WAIT instruction (PMCSR.WBPSM=1), the CPU continues to operate in the Active mode until it executes a WAIT instruction. Upon execution of the WAIT instruction, the device enters the Power Save mode and the CPU waits for the next interrupt event. In this case, the PMCSR.PSM bit is set to 1 when it is written, even before the WAIT instruction is executed.

### 11.6.3 Entering the Idle Mode

Entry into the Idle mode is accomplished by writing a 1 to the PMCSR.IDLE bit and then executing a WAIT instruction.

The Idle mode can be entered only from the Active or Power Save mode. For entry from the Active mode, the PMCSR.WBPSM bit must be set before the WAIT instruction is executed.

### 11.6.4 Disabling the High-Frequency Clock

In systems where the low-frequency crystal is available and is used to generate the Slow Clock (SLCLK), power consumption can be reduced further in the Power Save or Idle mode by disabling the high-frequency clock. This is accomplished by writing a 1 to the PMCSR.DHF bit before executing the WAIT instruction that puts the device in the Power Save or Idle mode. The high-frequency clock is turned off only after the device enters the Power Save or Idle mode.

The CPU operates on the low-frequency clock in Power Save mode. It can turn off the high-frequency clock at any time by writing a 1 to the PMCSR.DHF bit.

The high-frequency oscillator is always enabled in Active mode and always disabled in Halt mode, regardless of the PMCSR.DHF bit setting.

Immediately following power-up and entry into the Active mode, the software must wait for the low-frequency clock to become stable before it can put the device in the Power Save mode. It should monitor the PMCSR.OLFC bit for this purpose. Once this bit is set to 1, the slow clock is stable and the Power Save mode can be entered.

### 11.6.5 Entering the Halt Mode

Entry into the Halt mode is accomplished by writing a 1 to the PMCSR.HALT bit and then executing a WAIT instruction.

The Halt mode can be entered only from the Active or Power Save mode. For entry from the Active mode, the PMCSR.WBPSM bit must be set before the WAIT instruction is executed.

### 11.6.6 Software-Controlled Transition to Active Mode

A transition from the Power Save mode to the Active mode can be accomplished by either a software command or a hardware wake-up event. The software method is to write a 0 to the PMCSR.PSM bit. The value of the register bit changes only after the transition to the Active mode is completed.

If the high-frequency oscillator is disabled for Power Save operation, the oscillator must be enabled and allowed to stabilize before the transition to Active mode. To enable the high-frequency oscillator, the software writes a 0 to the PMCSR.DHF bit. Before writing a 0 to the PMCSR.PSM bit, the software should first monitor the PMCSR.OHFC bit to determine whether the oscillator has stabilized.

### 11.6.7 Wake-Up Transition to Active Mode

A hardware wake-up event switches the device directly from Power Save, Idle, or Halt mode to the Active mode.

Hardware wake-up events are:

- a Non-Maskable Interrupt (NMI)
- a valid wake-up event on a Multi-Input Wake-Up channel

When a wake-up event occurs, the on-chip hardware performs the following steps:

1. Clears the PMCSR.DHF bit, thus enabling the high-frequency clock (if it was disabled).
2. Waits for the PMCSR.OHFC bit to be set, which indicates that the high-frequency clock is operating and is stable.
3. Switches the device into the Active mode.

### 11.6.8 Power Mode Switching Protection

The Power Management Module has several mechanisms to protect the device from malfunctions caused by missing or unstable clock signals.

The PMCSR.OHFC and PMCSR.OLFC bits indicate the current status of the high-frequency and low-frequency clock oscillators, respectively. The software can check the appropriate bit before it changes to an operating mode that requires the clock. A status bit set to 1 indicates an operating, stable clock. A status bit cleared to 0 indicates a clock that is disabled, not available, or not yet stable.

During a power mode transition, if there is a request to switch to a mode that uses clock with its status bit cleared to 0, the switch is delayed until that bit is set to 1 by the hardware.

When the system is built without an external crystal network for the low-frequency clock, the high-frequency clock is divided by a prescaler factor to produce the low-frequency clock. In this situation, the high-frequency clock is disabled only in the Halt mode, and cannot be disabled for the Power Save or Idle mode, regardless of the software command issued.

Without an external crystal network for the low-frequency clock, the device comes out of the Halt or Idle mode and enters the Active mode with the high-speed oscillator used as the clock. The device can still enter the Power Save from the Active mode by using the high-frequency-clock divider to generate the slow clock (PMCSR.DHF=0).

**Note:** For correct operation in the absence of a low-frequency crystal, the X2CKI pin must be tied low (not left floating) so that the hardware can detect the absence of the crystal.

## 12.0 Dual Clock and Reset

The Dual Clock and Reset module (CLK2RES) generates a high-speed main system clock from an external crystal network and a slow clock (32.768 kHz or other rate) for operating the device in Power Save mode. It also provides the main system reset signal, a power-on reset function, a main clock prescaler to generate two additional low speed clocks, and an 32kHz oscillator start-up delay.

Figure7 is block diagram of the Dual Clock and Reset module.

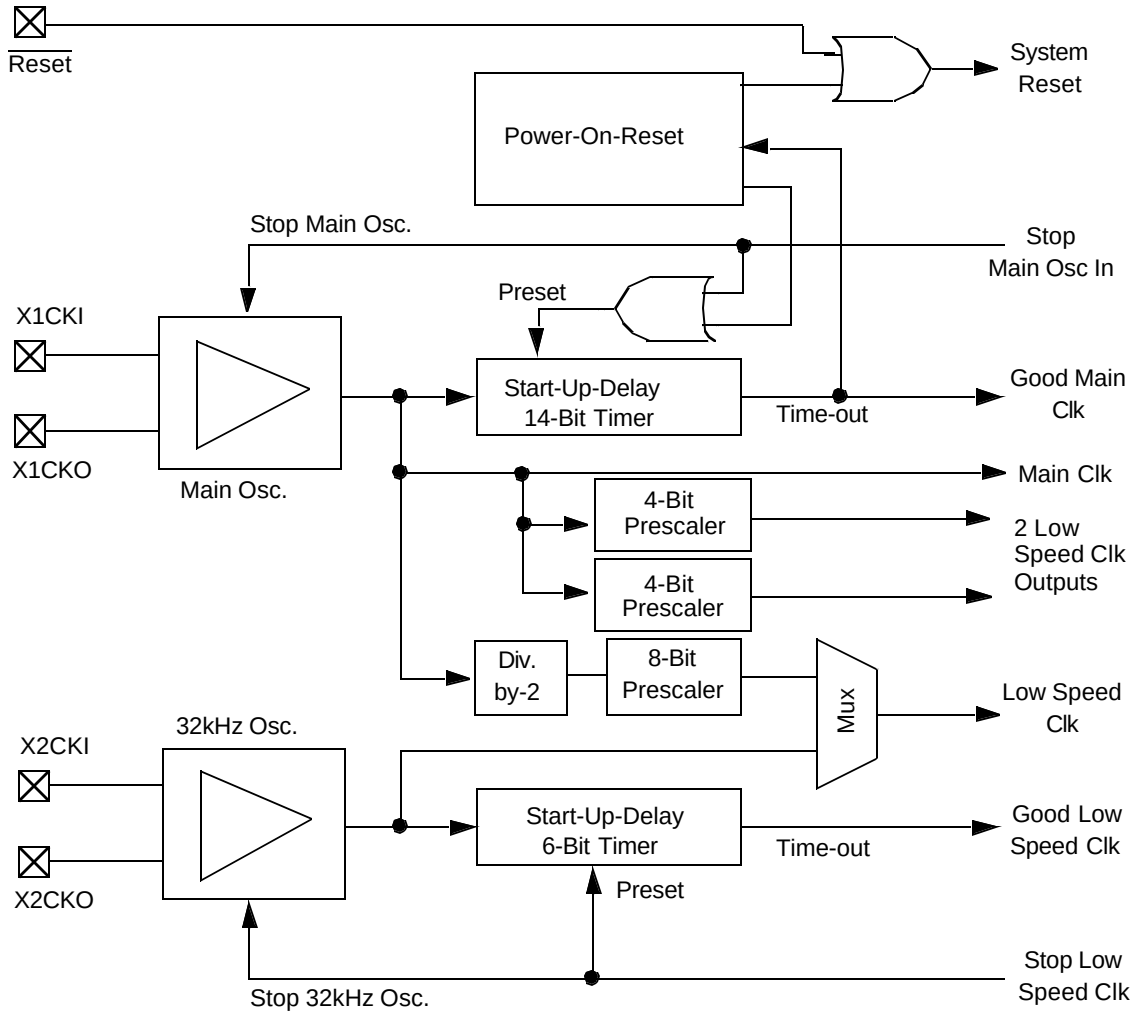



**Figure 7. Dual Clock and Reset Module Block Diagram**

### 12.1 EXTERNAL CRYSTAL NETWORK

An external crystal network is required at pins X1CKI and X1CKO for the main clock. A similar external crystal network may be used at pins X2CKI and X2CKO for the slow clock in packages that have these pins. If an external crystal network is not used for the slow clock, the clock is generated by dividing the fast main clock.

The crystal oscillator you choose may require external components different from the ones specified above. In that case, consult with National's engineer for the component specifications

The crystals and other oscillator components should be placed close to the X1CKI/X1CLO and X2CKI/X2CLO device input pins to keep the printed trace lengths to an absolute minimum.

Figure8 shows the required crystal network at X1CKI/X1CKO and optional crystal network at X2CKI/X2CKO. Table13 shows the component specifications for the main crystal network and Table14 shows the component specifications for the 32.768 kHz crystal network.

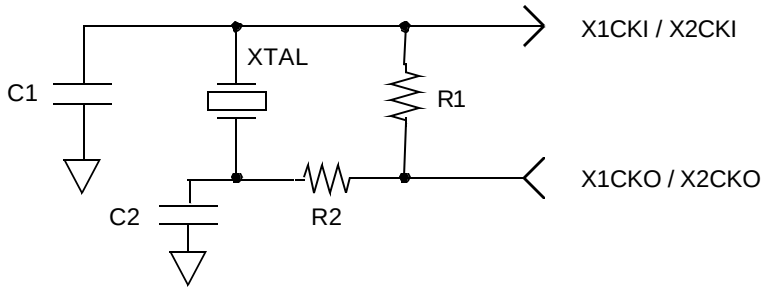



**Figure 8. External Crystal Network**

**Table 13 Component Values of the High Frequency Crystal Circuit**

| Component | Parameters | Values | Values | Values | Values | Values | Tolerance |
|---|---|---|---|---|---|---|---|
| Oscillator | Resonance Frequency<br>Type<br>Max. Serial Resistance<br>Max. Shunt Capacitance<br>Load Capacitance | 4 MHz<br>AT-Cut<br>75 Ω<br>4 pF<br>12 pF | 12 MHz<br>AT-Cut<br>35 Ω<br>4 pF<br>15 pF | 16 MHz<br>AT-Cut<br>35 Ω<br>4 pF<br>15 pF | 20 MHz<br>AT-Cut<br>35 Ω<br>4 pF<br>20 pF | 24 MHz<br>AT-Cut<br>35 Ω<br>4 pF<br>20 pF | N/A |
| Crystal | Resistor R1 | 1 MΩ | 1 MΩ | 1 MΩ | 1 MΩ | 1 MΩ | 5% |
| | Resistor R2 | 0 Ω | 0 Ω | 0 Ω | 0 Ω | 0 Ω | 5% |
| | Capacitor C1, C2 | 22 pF | 20 pF | 20 pF | 20 pF | 20 pF | 20% |

**Table 14 Component Values of the Low Frequency Crystal Circuit**

| Component | Parameters | Values | Tolerance |
|---|---|---|---|
| Oscillator | Resonance Frequency<br>Type<br>Maximum Serial Resistance<br>Maximum Shunt Capacitance<br>Load Capacitance | 32.768kHz<br>Parallel<br>N-Cut or XY-bar<br>40 kΩ<br>2 pF<br>9-13 pF | N/A |
| Crystal | Resistor R1 | 10-20 MΩ | 5% |
| | Resistor R2 | 4.7 kΩ | 5% |
| | Capacitor C1, C2 | 20 pF | 20% |

Choose capacitor component values in the tables obtain the specified load capacitance for the crystal when combined with the parasitic capacitance of the trace, socket, and package (which can vary from 0 to 8 pF). As a guideline, the load capacitance is:

$C_L = (C_1 * C_2)/(C_1+C_2) + C_{parasitic}$

$C_2 > C_1$

$C_1$ can be trimmed to obtain the desired load capacitance.

The start-up time of the 32.768 kHz oscillator can vary from one to six seconds. The long start-up time is due to the high "Q" value and high serial resistance of the crystal necessary to minimize power consumption in Power Save mode.

## 12.2 MAIN SYSTEM CLOCK

The main system clock is generated by the main oscillator. It can be stopped by the Power Management Module to reduce power consumption during periods of reduced activity. When the main clock is restarted, a 14-bit timer generates a "Good Main Clk" signal after a start-up delay of 32,768 clock cycles. This signal is an indicator that the main clock oscillator is stable.

The "Stop Main Osc" signal from the Power Management Module stops and starts the main oscillator. When this signal is asserted, it presets the 14-bit timer to 3FFF hex and stops the main oscillator. When the signal goes inactive, the main oscillator starts and the 14-bit timer counts down from its preset value. When the timer reaches zero, it stops counting and asserts the "Good Main Clk" signal.

## 12.3 SLOW SYSTEM CLOCK

The slow (32.768 kHz) clock is necessary for operating the device in Power Save modes and to provide a clock source for modules such as the Timing and Watchdog Module.

The slow clock operates in a manner similar to the main clock. The "Stop Slow Osc" signal from the Power Management Module stops and starts the slow oscillator. When this signal is asserted, it presets a 6-bit timer to 3F hex and disables the slow oscillator. When the signal goes inactive, the slow oscillator starts and the 6-bit timer counts down from its preset value. When the timer reaches zero, it stops counting

and asserts the “Good Low Speed Clk” signal, thus indicating that the slow clock is stable.

For systems that do not require a reduced power consumption mode, the external crystal network may be omitted for the slow clock. In that case, the slow clock can be created by dividing the main clock by a prescaler factor. The prescaler circuit consists of a fixed divide-by-2 counter and a programmable 8-bit prescaler register. This allows a choice of clock divisors ranging from 2 to 512. The resulting slow clock frequency must not exceed 100 kHz.

A software-programmable multiplexer selects either the prescaled main clock or the 32.768 kHz oscillator as the slow clock. Upon reset, the prescaled main clock is selected, ensuring that the slow clock is always present initially. Selection of the 32.768 kHz oscillator as the slow clock disables the clock prescaler, which allows the CLK1 oscillator to be turned off during power-save operation, thus reducing power consumption and radiated emissions. This can be done only if the module detects a togging low-speed oscillator. If the low-speed oscillator is not operating, the prescaler remains available as the slow clock source.

## 12.4 POWER-ON RESET

The Power-On Reset circuit generates a system reset signal upon power-up and holds the signal active for a period of time to allow the crystal oscillator to stabilize. The circuit detects a power turn-on condition, which presets the 14-bit timer to 3FFF hex. Once oscillation starts and the clock becomes active, the timer starts counting down. When the count reaches zero, the 14-bit timer stops counting and the internal reset signal is deactivated (unless the $\overline{\text{RESET}}$ pin is held low).

The circuit sets a power-on reset flag bit upon detection of a power-on condition. The CPU can read this flag to determine whether a reset was caused by a power-up or by the $\overline{\text{RESET}}$ input.

**Note:** Power-On Reset circuit cannot be used to detect a drop in the supply voltage.

## 12.5 EXTERNAL RESET

An active-low reset input pin called $\overline{\text{RESET}}$ allows the device to be reset at any time. When the signal goes low, it generates an internal system reset signal that remains active until the $\overline{\text{RESET}}$ signal goes high again.

## 12.6 DUAL CLOCK AND RESET REGISTERS

The Dual Clock and Reset module (CLK2RES) contains two registers: the Clock and Reset Control register (CRCTRL) and the Slow Clock Prescaler register (PRSSC).

### 12.6.1 Clock and Reset Control Register (CRCTRL)

Clock and Reset Control Register (CRCTRL) is a byte-wide read/write register that contains the power-on reset flag and selects the type of slow clock. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | POR | SCLK |

SCLK Slow Clock Select. When this bit is set to 1, the 32.728 kHz oscillator is used for the slow clock. When this bit is cleared to 0, the prescaled main clock is used for the slow clock. Upon reset, this bit is cleared to 0.

POR Power-On Reset. This bit is set to 1 by the hardware when a power-on condition is detected, allowing the CPU to determine whether a power-up has occurred. The CPU can clear this bit to 0 but cannot set it to 1. Any attempt by the CPU to set this bit is ignored.

## 12.7 SLOW CLOCK PRESCALER REGISTER (PRSSC)

The Slow Clock Prescaler (PRSSC) register is a byte-wide read/write register that holds the clock divisor used to generate the slow clock from the main clock. The format of the register is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SCDIV | | | | | | | |

SCDIV Slow Clock Divisor. If the clock divider is enabled (CRCTRL.SCLK=0), the main clock is divided by (SCDIV+1)*2 to produce the slow system clock. Upon reset, PRSSC register is set to FF hex.

## 12.8 SLOW CLOCK PRESCALER 1 REGISTER (PRSSC1)

The Slow Clock Prescaler 1 (PRSSC1) register is a byte-wide read/write register that holds the clock divisor used to generate the two additional slow clocks from the high-speed clock. Upon reset, the register is set to 00. The format of the register is shown below.

| 7 4 | 3 0 |
|---|---|
| SCDIV2 | SCDIV1 |

SCDIV1 Slow Clock Divisor 1. The main clock is divided by (SCDIV1+1) to obtain the first slow system clock.

SCDIV1 Slow Clock Divisor 2. The main clock is divided by (SCDIV2+1) to obtain the second slow system clock.

# 13.0 Multi-Input Wake-Up

The Multi-Input Wake-Up (MIWU16) module monitors its 16 input channels for a software-selectable trigger condition. Upon detection of a trigger condition, the module generates an interrupt request and if enabled, a wake-up request. A wake-up request can be used by the power management unit to exit the Halt, Idle, or Power Save mode and return to the active mode. An interrupt request generates an interrupt to the CPU (interrupt IRQ2), allowing interrupt processing in response to external events.

The wake-up event only activates the clocks and CPU, but does not by itself initiate execution of any code. It is the interrupt request associated with the MIWU16 that gets the CPU to start executing code, by jumping to the proper interrupt routine. Therefore, setting up the MIWU16 interrupt handler is essential for any wake-up operation.

There are four interrupt requests that can be routed to the ICU as shown in Figure9. Each of the 16 MIWU channels can be programmed to activate one of these four interrupt requests.

The input pins for the Multi-Input Wake-Up channels are named WUI0 through WUI15.

| | |
|---|---|
| WUI0 | PL0 |
| WUI1 | PL1 |
| WUI2 | PL2 |
| WUI3 | PL3 |
| WUI4 | PH0 |
| WUI5 | PH1 |
| WUI6 | PH2 |
| WUI7 | PH3 |
| WUI8 | TWM-T0OUT |
| WUI9 | ACCESS.bus |
| WUI10 | Canards |
| WUI11 | MWCS |
| WUI12 | RDX1 |
| WUI13 | RDX2 |
| WUI14 | Comparator 1 |
| WUI15 | Comparator 2 |

Each input can be configured to trigger on rising or falling edges, as determined by the setting in the WKEDG register. Each trigger event is latched into the WKPND register. If a trigger event is enabled by its respective bit in the WKENA register, an active wake-up/interrupt signal is generated. The software can determine which channel has generated the active signal by reading the WKPND register.

The Multi-Input Wake-Up module is active at all times, including the Halt mode. All device clocks are stopped in this mode. Therefore, detecting an external trigger condition and the subsequent setting of the pending flag are not synchronous to the system clock.

## 13.1 WAKE-UP EDGE DETECTION REGISTER (WKEDG)

The Wake-Up Edge Detection (WKEDG) register is a word-wide read/write register that controls the edge sensitivity of the Multi-Input Wake-Up pins. Register bits 0 through 15 control input pins WUI0 through WUI15, respectively. A bit cleared to 0 configures the corresponding input to trigger on a rising edge (a low-to-high transition). A bit set to 1 configures the corresponding input to trigger on a falling edge (a high-to-low transition).

This register is cleared upon reset, which configures all 16 inputs to be triggered on rising edges.

The register format is shown below.

| 15 | 0 |
|---|---|
| WKED15-WKED0 | |

## 13.2 WAKE-UP ENABLE REGISTER (WKENA)

The Wake-Up Enable (WKENA) register is a word-wide read/write register that enables or disables each of the Multi-Input Wake-Up channels. Register bits 0 through 15 control channels WUI0 through WUI15, respectively. A bit cleared to 0 disables the wake-up function and a bit set to 1 enables the function.

This register is cleared upon reset, which disables all eight wake-up/interrupt channels.

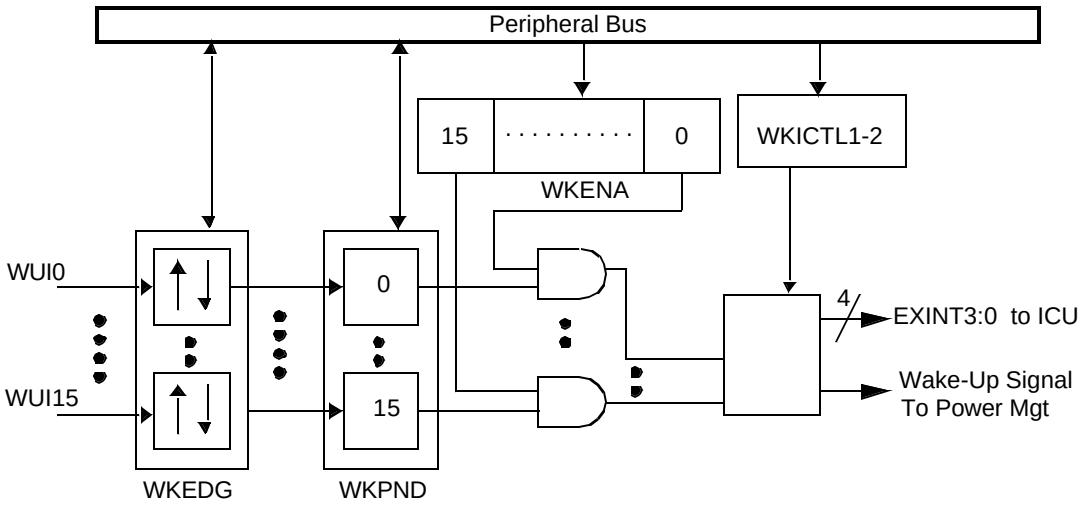



**Figure 9. Multi-Input Wake-Up Module Block Diagram**

The register format is shown below.

| 15 | 0 |
|---|---|
| WKEN15-WKEN0 | |

## 13.3 WAKE-UP INTERRUPT CONTROL REGISTER 1 (WKCTL1)

The Wake-Up Interrupt Control Register 1 (WKICTL1) register is a word-wide read/write register that selects the interrupt request signal for the associated channels WUI0 to WUI7. Upon reset, WKICTL1 is set to 0, which selects MIWU Interrupt Request 0 for all eight channels. The register format is shown below.

| 15 14 | 13 12 | 11 10 | 9 8 | 7 6 | 5 4 | 3 2 | 1 0 |
|---|---|---|---|---|---|---|---|
| WKINTR 7 | WKINTR 6 | WKINTR 5 | WKINTR 4 | WKINTR 3 | WKINTR 2 | WKINTR 1 | WKINTR 0 |

WKINTR0:7 Wake-Up Interrupt Request Select. Each field selects which of the following four interrupt requests outputs to the ICU31L are to be activated for the corresponding channel.

**00** enables MIWU Interrupt Request 0

**01** enables MIWU Interrupt Request 1

**10** enables MIWU Interrupt Request 2

**11** enables MIWU Interrupt Request 3

## 13.4 WAKE-UP INTERRUPT CONTROL REGISTER 1 (WKCTL2)

The Wake-Up Interrupt Control Register 2 (WKICTL2) register is a word-wide read/write register that selects the interrupt request signal for the associated channels WUI8 to WUI15. Upon reset, WKICTL2 is set to 0, which selects MIWU Interrupt Request 0 for all eight channels. The register format is shown below.

| 15 14 | 13 12 | 11 10 | 9 8 | 7 6 | 5 4 | 3 2 | 1 0 |
|---|---|---|---|---|---|---|---|
| WKINTR 15 | WKINTR 14 | WKINTR 13 | WKINTR 12 | WKINTR 11 | WKINTR 10 | WKINTR 9 | WKINTR 8 |

WKINTR8:5 Wake-Up Interrupt Request Select. Each field selects which of the following four interrupt requests outputs to the ICU31L are to be activated for the corresponding channel.

**00** enables MIWU Interrupt Request 0

**01** enables MIWU Interrupt Request 1

**10** enables MIWU Interrupt Request 2

**11** enables MIWU Interrupt Request 3

## 13.5 WAKE-UP PENDING REGISTER (WKPND)

The Wake-Up Pending (WKPND) register is a word-wide read/write register in which the Multi-Input Wake-Up module latches any detected trigger conditions. Register bits 0 through 15 serve as latches for channels WUI0 through WUI15, respectively. A bit cleared to 0 indicates that no trigger condition has occurred. A bit set to 1 indicates that a trigger condition has occurred and is pending on the corresponding channel. This register is cleared upon reset.

The CPU can only write a 1 to any bit position in this register. If the CPU attempts to write a 0, it has no effect on that bit. To clear a bit in this register, the CPU must use the WKPCL register (described below). This implementation prevents a potential hardware-software conflict during a read-modify-write operation on the WKPND register.

The register format is shown below.

| 15 | 0 |
|---|---|
| WKPD15-WKPD0 | |

## 13.6 WAKE-UP PENDING CLEAR REGISTER (WKPCL)

The Wake-Up Pending Clear (WKPCL) register is a word-wide write-only register that lets the CPU clear bits in the WKPND register. Writing a 1 to a bit position in the WKPCL register clears the corresponding bit in the WKPND register. Writing a 0 leaves the corresponding bit in the WKPND register unchanged.

Reading this register location returns unknown data. Therefore, do not use a read-modify-write sequence to set the individual bits. In other words, do not attempt to read the

register and do a logical OR with the register value. Instead, just write the mask directly to the register address.

The register format is shown below.

| 15 | 0 |
|---|---|
| WKCL15-WKCL0 | |

## 13.7 PROGRAMMING PROCEDURES

To set up and use the Multi-Input Wake-Up function, use the following procedure. Performing the steps in the order shown will prevent false triggering of a wake-up condition. This same procedure should be used following a reset because the wake-up inputs are left floating, resulting in unknown data on the input pins.

1. Clear the WKENA register to disable the wake-up channels.
2. If the input originates from an I/O port (the usual case), set the corresponding bit in the port direction register to configure the I/O pin to operate as an input.
3. Write the WKEDG register to select the desired type of edge sensitivity (clear to 0 for rising edge, set to 1 for falling edge).
4. Set all bits in the WKPCL register to clear any pending bits in the WKPND register.
5. Set up the WKICTL1 and WKICTL2 registers to define the interrupt request signal used for each channel.
6. Set the bits in the WKENA register corresponding to the wake-up channels to be activated.

To change the edge sensitivity of a wake-up channel, use the following procedure. Performing the steps in the order shown will prevent false triggering of a wake-up/interrupt condition.

1. Clear the WKENA bit associated with the input to be reprogrammed.
2. Write the new value to the corresponding bit position in the WKEDG register to reprogram the edge sensitivity of the input.
3. Set the corresponding bit in the WKPCL register to clear the pending bit in the WKPND register.
4. Set the same WKENA bit to re-enable the wake-up function.

# 14.0 Real-Time Timer and WATCHDOG

The Timing and WATCHDOG Module (TWM) generates the clocks and interrupts used for timing periodic functions in the system, and also provides Watchdog protection against software errors. The module operates off the slow clock either generated by the external 32kHz oscillator or from the prescaled high speed system clock. The maximum operating clock frequency is 100kHz.

The WATCHDOG is designed to detect program execution errors. Once WATCHDOG operation is initiated, the software must periodically write a specific value to a WATCHDOG register. If the software fails to do so, a WATCHDOG error is triggered, which resets the device.

The TWM is flexible in allowing selection of a variety of clock ratios and clock sources for the WATCHDOG circuit. Once the software configures the TWM, it can lock the configuration for a higher level of protection against erroneous software action. Once locked, the TWM can be released only by a device reset.

## 14.1 TWM STRUCTURE

Figure10 is a block diagram showing the internal structure of the Timing and WATCHDOG module. There are two main sections: the Real-Time Timer (T0) section at the top and the WATCHDOG section on the bottom.

All counting activities of the module are based on the slow clock (SLCLK). A prescaler counter divides this clock to make a slower clock. The prescaler factor is defined by a 3-bit field in the Timer and WATCHDOG Prescaler register, which selects either 1, 2, 4, 8, 16, or 32 and the divide-by factor. Thus, the prescaled clock period can be set to 1, 2, 4, 8, 16, or 32 times the slow clock period. The prescaled clock signal is called T0IN.

## 14.2 TIMER T0 OPERATION

Timer T0 is a programmable 16-bit down counter that can be used as the time base for real-time operations such as a periodic audible tick. It can also be used to drive the WATCHDOG circuit.

The timer starts counting from the value loaded into the TWMT0 register and counts down on each rising edge of T0IN. When the timer reaches zero, it is automatically reloaded from the TWMT0 register and continues counting down from that value. Thus, the frequency of the timer is:

$f_{SLCLK}$ / [(TWMT0+1) * prescaler]

When an external crystal oscillator is used as the SLCLK source or when the fast clock is divided accordingly, $f_{SLCLK}$ is 32.768 kHz.

The value stored in TWMT0 can range from 0001 hex to FFFF hex.

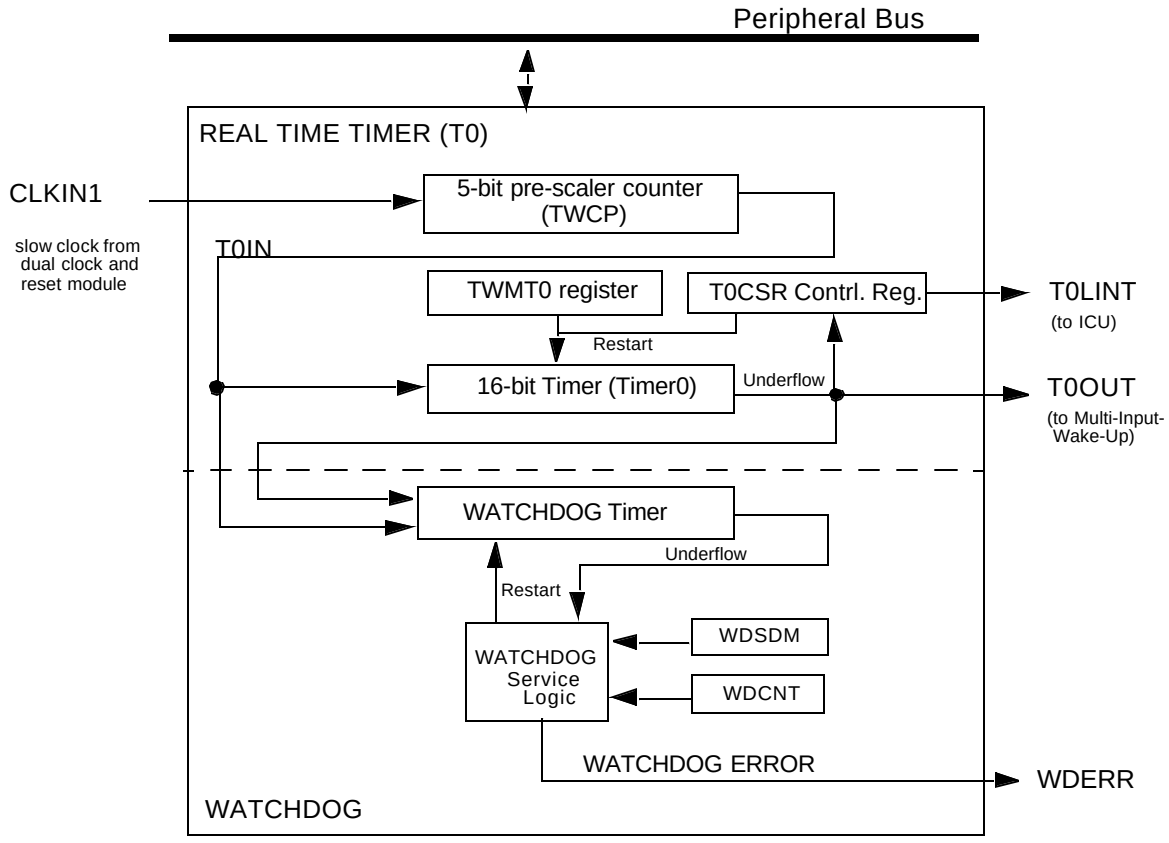



**Figure 10. Timing and WATCHDOG Module Block Diagram**

When the counter reaches zero, an internal timer signal called T0OUT is set to 1 for one T0IN clock cycle. This signal sets the TC bit in the TWMT0 Control and Status Register (T0CSR). It also generates an interrupt called RTI (IRQ14) if the interrupt is enabled by the T0CSR.T0INTE bit.

If the software loads TWMT0 with a new value, the timer uses that value the next time that it reloads the 16-bit timer register (in other words, after reaching zero). The software can restart the timer at any time (on the very next edge of the T0IN clock) by setting the Restart (RST) bit in the T0CSR register. The T0CSR.RST bit is cleared automatically upon restart of the 16-bit timer.

**Note:** If the user wishes to switch to power save or idle mode after setting T0CSR.RST, the user must wait for reset operation to complete before doing the switch.

## 14.3 WATCHDOG OPERATION

The WATCHDOG is an 8-bit down counter that operates on the rising edge of a specified clock source. Upon reset, the WATCHDOG is disabled; it does not count and no WATCHDOG signal is generated. A write to either the WATCHDOG Count (WDCNT) register or the WATCHDOG Service Data Match (WDSDM) register starts the counter. The WATCHDOG counter counts down from the value programmed in to the WDCNT register. Once started, only a reset can stop the WATCHDOG from operating.

The WATCHDOG can be programmed to use either T0OUT or T0IN as its clock source (the output and input of Timer T0, respectively). The TWCFG.WDCT0I bit controls this clock selection.

The software must periodically “service” the WATCHDOG. There are two ways to service the WATCHDOG, the choice depending on the programmed value of the WDSDME bit in the Timer and WATCHDOG Configuration (TWCFG) register.

If TWCFG.WDSDME bit is cleared to 0, the WATCHDOG is serviced by writing a value to the WDCNT register. The value written to the register is reloaded into the WATCHDOG counter. The counter then continues counting down from that value.

If TWCFG.WDSDME bit is set to 1, the WATCHDOG is serviced by writing the value 5C hex to the WATCHDOG Service Data Match (WDSDM) register. This reloads the WATCHDOG counter with the value previously programmed into the WDCNT register. The counter then continues counting down from that value.

A WATCHDOG error signal is generated by any of the following events:

- The WATCHDOG serviced too late.
- The WATCHDOG serviced too often.
- The WDSDM register is written with a value other than 5C hex when WDSDM type servicing is enabled (TWCFG.WDSDME=1).

A WATCHDOG error condition resets the device.

### 14.3.1 Register Locking

The Timer and WATCHDOG Configuration (TWCFG) register is used to set the WATCHDOG configuration. It controls the WATCHDOG clock source (T0IN or T0OUT), the type of WATCHDOG servicing (using WDCNT or WDSDM), and the locking state of the TWCFG, TWCPR, TIMER0, T0CSR, and WDCNT registers. A register that is locked cannot be read or written. A write operation is ignored and a read operation returns unpredictable results.

If the TWCFG register is itself locked, it remains locked until the device is reset. Any other locked registers also remain locked until the device is reset. This feature prevents a runaway program from tampering with the programmed WATCHDOG function.

### 14.3.2 Power Save Mode Operation

The Timer and WATCHDOG Module is active in both the Power Save and Idle modes. The clocks and counters continue to operate normally in these modes. The WDSDM register is accessible in the Power Save and Idle modes, but the other TWM registers are accessible only in the Active mode. Therefore, WATCHDOG servicing must be carried out using the WDSDM register in the Power Save or Idle mode.

In the Halt mode, the entire device is frozen, including the Timer and WATCHDOG Module. Upon return to the Active mode, operation of the module resumes at the point at which it was stopped.

**Note:** After a restart or WATCHDOG service through WDCNT, do not enter Power Save mode for a period equivalent to 5 slow clock cycles.

## 14.4 TWM REGISTERS

The TWM registers controls the operation of the Timing and WATCHDOG Module. There are six such registers:

- Timer and WATCHDOG Configuration Register (TWCFG)
- Timer and WATCHDOG Clock Prescaler Register (TWCP)
- TWM Timer 0 Register (TWMT0)
- TWMT0 Control and Status Register (T0CSR)
- WATCHDOG Count Register (WDCNT)
- WATCHDOG Service Data Match Register (WDSDM)

The WDSDM register is accessible in both Active and Power Save mode. The other TWM registers are accessible only in Active mode.

### 14.4.1 Timer and WATCHDOG Configuration Register (TWCFG)

The TWCFG register is a byte-wide, read/write register that selects the WATCHDOG clock input and service method, and also allows the WATCHDOG registers to be selectively locked. Once a bit is set, that bit cannot be cleared until the device resets. Upon reset, the non-reserved bits of the register are all cleared to 0. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | WDSDME | WDCT0I | LWDCNT | LTWMT0 | LTWCP | LTWCFG |

LTWCFG — Lock TWCFG Register. When cleared to 0, access to the TWCFG register is allowed. When set to 1, the TWCFG register is locked. A locked register cannot be read or written; a read operation returns unpredictable values and a write operation is ignored. Locking the TWCFG register remains in effect until the device is reset.

LTWCP Lock TWCP Register. When cleared to 0, access to the TWCP register is allowed. When set to 1, the TWCP register is locked.

LTWMT0 Lock TWMT0 Register. When cleared to 0, access to the TWMT0 and T0CSR registers are allowed. When set to 1, the TWMT0 and T0CSR registers are locked.

LWDCNT Lock LDWCNT Register. When cleared to 0, access to the LDWCNT register is allowed. When set to 1, the LDWCNT register is locked.

WDCT0I WATCHDOG Clock from T0IN. When cleared to 0, the T0OUT signal (the output of Timer T0) is used as the WATCHDOG clock. When set to 1, the T0IN signal (the prescaled slow clock) is used as the WATCHDOG clock.

WDSDME WATCHDOG Service Data Match Enable. When cleared to 0, WATCHDOG servicing is accomplished by writing a count value to the WDCNT register; write operations to the WATCHDOG Service Data Match (WDSDM) register are ignored. When set to 1, WATCHDOG servicing is accomplished by writing the value 5C hex to the WDSDM register.

#### 14.4.2 Timer and WATCHDOG Clock Prescaler Register (TWCP)

The TWCP register is a byte-wide, read/write register that defines the prescaler value used for dividing the low frequency clock to generate the T0IN clock. Upon reset, the non-reserved bits of the register are cleared to 0. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | MDIV | | |

MDIV Main Clock Divide. This 3-bit field defines the prescaler factor used for dividing the low speed device clock to create the T0IN clock. The allowed 3-bit values and the corresponding clock divisors and clock rates are listed below.

| MDIV | Clock Divisor | TOIN Frequency ($f_{SCLK}$=32.768 kHz) |
|---|---|---|
| 000 | 1 | 32.768 kHz |
| 001 | 2 | 16.384 kHz |
| 010 | 4 | 8.192 kHz |
| 011 | 8 | 4.096 kHz |
| 100 | 16 | 2.056 kHz |
| 101 | 32 | 1.024 kHz |
| other | Reserved | N/A |

#### 14.4.3 TWM Timer 0 Register (TWMT0)

The TWMT0 register is a word-wide, read/write register that defines the T0OUT interrupt rate. Upon reset, TWMT0 register is initialized to FFFF hex. The register format is shown below.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PRESET | | | | | | | | | | | | | | | |

PRESET Timer T0 Preset. Timer T0 is reloaded with this value on each underflow. Thus, the frequency of the Timer T0 interrupt is the frequency of T0IN divided by (PRESET+1). The allowed values of PRESET are 0001 hex through FFFF hex.

#### 14.4.4 TWMT0 Control and Status Register (T0CSR)

The T0CSR register is a byte-wide, read/write register that controls Timer T0 and shows its current status. Upon reset, the non-reserved bits of the register are cleared to 0. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | T0INTE | TC | RST |

RST Restart. When this bit is set to 1, it forces the timer to reload the value in the TWMT0 register on the next rising edge of the selected input clock. The RST bit is reset automatically by the hardware on the same rising edge of the selected input clock. Writing a 0 to this bit position has no effect. Upon reset, the non-reserved bits of the register are cleared to 0.

TC Terminal Count. This bit is set to 1 by the hardware when the Timer T0 count reaches zero and is cleared to 0 when the software reads the T0CSR register. It is a read-only bit. Any data written to this bit position is ignored.

T0INTE Timer T0 Interrupt Enable. When this bit is set to 1, it enables an interrupt to the CPU each time the Timer T0 count reaches zero. When this bit is cleared to 0, Timer T0 interrupts are disabled.

#### 14.4.5 WATCHDOG Count Register (WDCNT)

The WDCNT register is a byte-wide, write-only register that holds the value that is loaded into the WATCHDOG counter each time the WATCHDOG is serviced. The WATCHDOG is started by the first write to this register. Each successive write to this register restarts the WATCHDOG count with the written value. Upon reset, this register is initialized to 0F hex.

#### 14.4.6 WATCHDOG Service Data Match Register (WDSDM)

The WSDSM register is a byte-wide, write-only register used for servicing the WATCHDOG. When this type of servicing is enabled (TWCFG.WDSDME=1), the WATCHDOG is serviced by writing the value 5C hex to the WSDSM register. Each such servicing reloads the WATCHDOG counter with the value previously written to the WDCNT register. Writing any data other than 5C hex triggers a WATCHDOG error. Writing to the register more than once in one WATCHDOG clock cycle also triggers a WATCHDOG error signal. If this type of servicing is disabled (TWCFG.WDSDME=0), any write to the WSDSM register is ignored.

### 14.5 WATCHDOG PROGRAMMING PROCEDURE

The highest level of protection against software errors is achieved by programming and then locking the WATCHDOG registers and using the WDSDM register for servicing. This is the procedure:

1. Write the desired values into the TWM Clock Prescaler register (TWCP) and the TWM Timer 0 register (TWMT0) to control the T0IN and T0OUT clock rates. The frequency of T0IN can be programmed to any of six frequencies ranging from $1/32*f_{SLCLK}$ to $f_{SLCLK}$ . The frequency of T0OUT is equal to the frequency of T0IN divided by (1+PRESET), where PRESET is the value written to the TWMT0 register.
2. Configure the WATCHDOG clock to use either T0IN or T0OUT by setting or clearing the TWCFG.WDCT0I bit.
3. Write the initial value into the WDCNT register. This starts operation of the WATCHDOG and specifies the maximum allowed number of WATCHDOG clock cycles between service operations.
4. Lock the WATCHDOG registers and enable the WATCHDOG Service Data Match Enable function by setting bits 0, 1, 2, 3, and 5 in the TWCFG register.
5. Service the WATCHDOG by periodically writing the value 5C hex to the WDSDM register at an appropriate rate. Servicing must occur at least once per period programmed into the WDCNT register, but no more than once in a single WATCHDOG input clock cycle.

# 15.0 Multi-Function Timer

The Multi-Function Timer (MFT16) module contains two independent timer/counter units called MFT1 and MFT2, each containing a pair of 16-bit timer/counters. Each timer/counter unit offers a choice of clock sources for operation and can be configured to operate in any of the following modes:

- Processor-Independent Pulse Width Modulation (PWM) mode, which generates pulses of a specified width and duty cycle, and which also provides a general-purpose timer/counter
- Dual Input Capture mode, which measures the elapsed time between occurrences of external events, and which also provides a general-purpose timer/counter
- Dual Independent Timer mode, which generates system timing signals or counts occurrences of external events
- Single Input Capture and Single Timer mode, which provides one external event counter and one system timer

The two timer units, MFT1 and MFT2, are identical in operation and separately programmable. Each timer unit uses two I/O pins, called T1A and T1B (for Timer MFT1) or T2A and T2B (for Timer MFT2). The timer I/O pins are alternate functions of the Port F I/O pins.

In the description of the timers, the lower-case letter “n” represents the timer number, either 1 or 2. For example, “TnA” means I/O pin T1A or T2A.

## 15.1 TIMER STRUCTURE

Figure11 is a block diagram showing the internal structure of each timer. There are two main functional blocks: a Timer/Counter and Action block and a Clock Source block. The Timer/Counter and Action block contains two separate timer/counter units, called Timer/Counter I and Timer/Counter II (a total of four timer/counter unit in both MFT1 and MFT2).

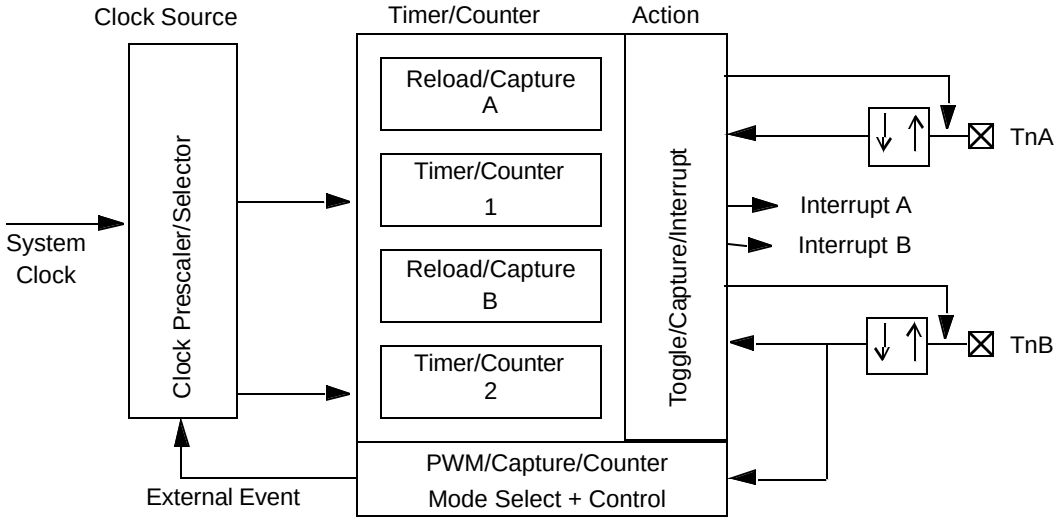



**Figure 11. Multi-Function Timer Block Diagram**

### 15.1.1 Timer/Counter Block

The Timer/Counter block contains the following functional blocks:

— two 16-bit counters, Timer/Counter I (TnCNT1) and Timer/Counter II (TnCNT2)
— two 16-bit reload/capture registers, TnCRA and TnCRB
— control logic necessary to configure the timer to operate in any of the four operating modes
— interrupt control and I/O control logic

In a power-saving mode that uses the low-frequency (32.768 kHz) clock as the system clock, the synchronization circuit requires that the slow clock operate at no more than one-fourth the speed of the 32.768 kHz system clock.

### 15.1.2 Clock Source Block

The Clock Source block generates the signals used to clock the two timer/counter registers. The internal structure of the Clock Source block is shown in Figure12.

**Counter Clock Source Select**

There are two clock source selectors that allow the software to independently select the clock source for each of the two 16-bit counters from any one of the following sources:

— no clock (which stops the counter)
— prescaled system clock
— external event count based on TnB
— pulse accumulate mode based on TnB
— slow clock (derived from the low-frequency oscillator or divided from the high-speed oscillator)

**Prescaler**

The 5-bit clock prescaler allows the software to run the timer with a prescaled clock signal. The prescaler consists of a 5-bit read/write prescaler register (TnPRSC) and a 5-bit down counter. The system clock is divided by the value contained in the prescaler register plus 1. Thus, the timer clock period can be set to any value from 1 to 32 divisions of the system clock period. The prescaler register and down counter are both cleared upon reset.

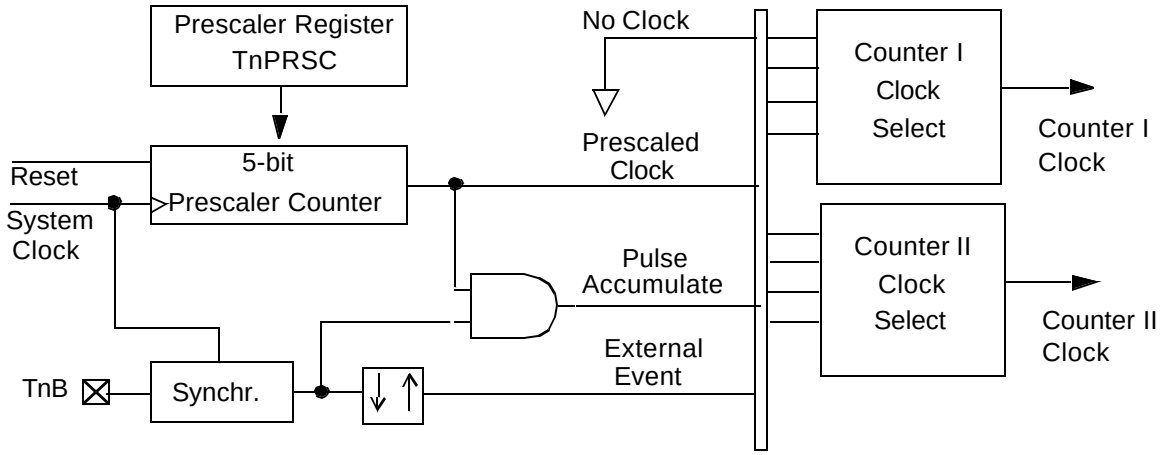



**Figure 12. Clock Source Block Diagram**

**External Event Clock**

The TnB I/O pin can be configured to operate as an external event input clock for either of the two 16-bit counters. This input can be programmed to detect either rising or falling edges. The minimum pulse width of the external signal is one system clock cycle. This means that the maximum frequency at which the counter can run in this mode is one-half of the system clock frequency. This clock source is not available in the capture modes (modes 2 and 4) because the TnB pin is used as one of the two capture inputs.

**Pulse Accumulate Mode**

The counter can also be configured to count prescaler output clock pulses when the TnB is high and not count when TnB is low, as illustrated in Figure13. The resulting count is an indicator of the cumulative time that TnB is high. This is called the “pulse accumulate” mode. In this mode, an AND gate generates a clock signal for the counter whenever a prescaler clock pulse is generated and TnB input is high. (The polarity of the TnB signal is programmable, so the counter can count when TnB is low rather than high.) The pulse accumulate mode is not available in the capture modes (modes 2 and 4) because the TnB pin is used as one of the two capture inputs.

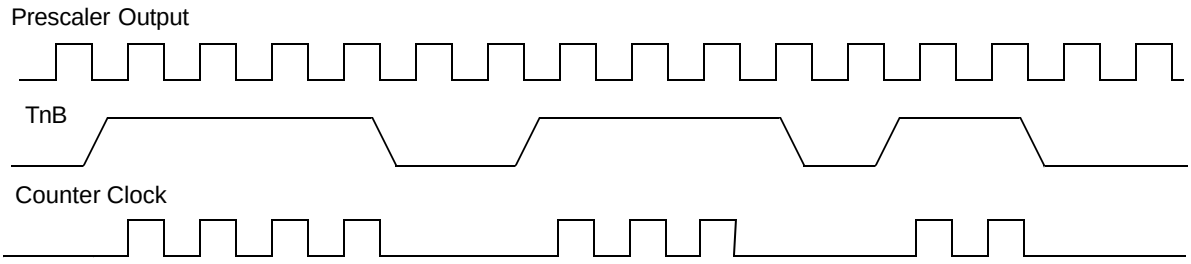



**Figure 13. Pulse Accumulate Mode Operation**

**Slow Clock**

The slow clock is generated by the Dual Clock and Reset (CLK2RES) module. The clock source is either the divided fast clock or the external 32.768 kHz clock crystal (if available and selected). The slow clock can be used as the clock source for the two 16-bit counters. Because the slow clock can be asynchronous to the system clock, a circuit is provided to synchronize the clock signal to the high-frequency system clock before it is used for clocking the counters. The synchronization circuit requires that the slow clock operate at no more than one-fourth the speed of the system clock.

**Limitations in Low-Power Modes**

The Power Save mode uses the low-frequency clock as the system clock. In this mode, the slow clock cannot be used as a clock source for the timers because both CLK and SLCLK are driven then at the same frequency, and the 2:1 system-clock to input clock ratio needed for the synchronization cannot be maintained. However, the External Event Clock and Pulse Accumulate Mode will still work, as long as the external event pulses are at least the size of the whole slow-clock period. Using the prescaled system clock will also work, but at a much slower rate than the original system clock.

Some Power Save modes stops the system clock (the high-frequency and/or low-frequency clock) completely. If the system clock is stopped, the timer stops counting until the system clock resumes operation.

In the Idle or Halt mode, the system clock stops completely, which stops the operation of the timers. In that case, the timers stop counting until the system clock resumes operation.

## 15.2 TIMER OPERATING MODES

Each timer/counter unit can be configured to operate in any of the following modes:

— Processor-Independent Pulse Width Modulation (PWM) mode
— Dual Input Capture mode
— Dual Independent Timer mode
— Single Input Capture and Single Timer mode

Upon reset, the timers are disabled. To configure and start the timers, the software must write a set of values to the registers that control the timers. The registers are described in Section15.5.

### 15.2.1 Mode 1: Processor-Independent PWM

Mode 1 is the Processor-Independent Pulse Width Modulation (PWM) mode, which generates pulses of a specified width and duty cycle, and which also provides a separate general-purpose timer/counter.

Figure14 is a block diagram of the Multi-Function Timer configured to operate in Mode 1. Timer/Counter I (TnCNT1) functions as the time base for the PWM timer. It counts down at the clock rate selected for the counter. When an underflow occurs, the timer register is reloaded alternately from the TnCRA and TnCRB register, and counting proceeds downward from the loaded value.

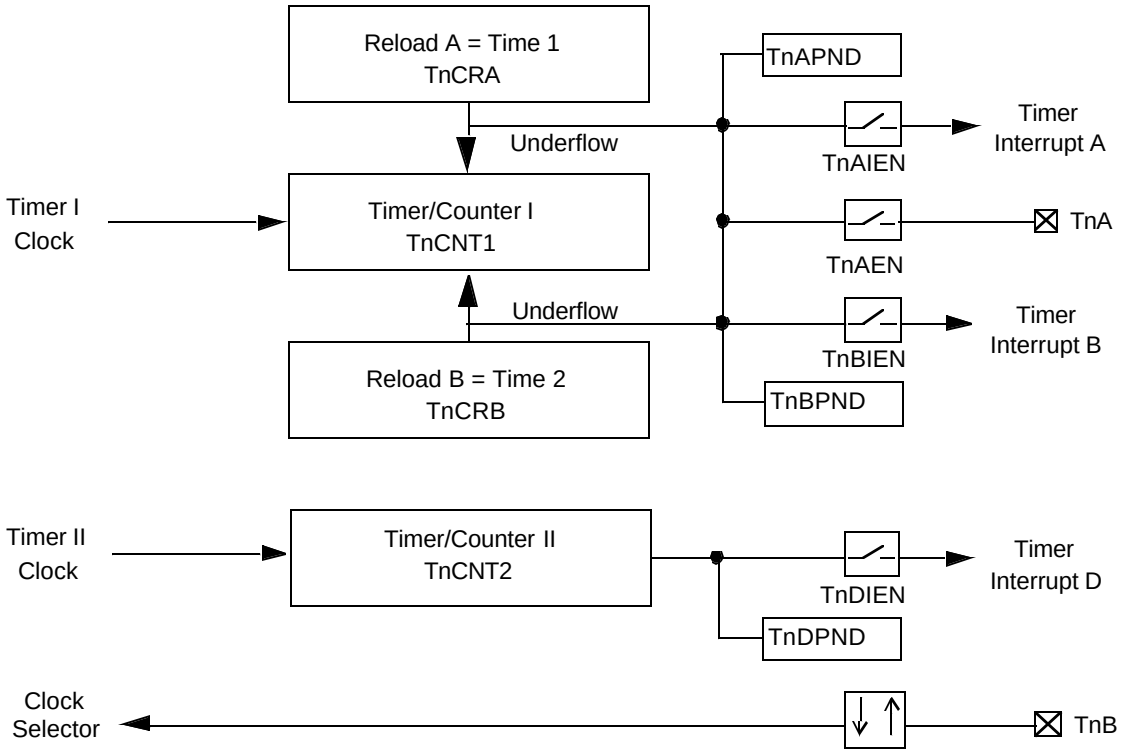



**Figure 14. Mode 1: Processor-Independent PWM Block Diagram**

On the first underflow, the timer is loaded from TnCRA, then from TnCRB on the next underflow, then from TnCRA again on the next underflow, and so on. Every time the counter is stopped and restarted, it always obtains its first reload value from TnCRA. This is true whether the timer is restarted upon reset, after entering Mode 1 from another mode, or after stopping and restarting the clock with the Timer/Counter I clock selector.

The timer can be configured to toggle the TnA output bit upon each underflow. This generates a clock signal on TnA with the width and duty cycle determined by the values stored in the TnCRA and TnCRB registers. This is a “processor-independent” PWM clock because once the timer is set up, no more action is required from the CPU to generate a continuous PWM signal.

The timer can be configured to generate separate interrupts upon reload from TnCRA and TnCRB. The interrupts can be enabled or disabled under software control. The CPU can determine the cause of each interrupt by looking at the TnAPND and TnBPND flags, which are set by the hardware upon each occurrence of a timer reload.

In Mode 1, Timer/Counter II (TnCNT2) can be used either as a simple system timer, an external event counter, or a pulse accumulate counter. The clock counts down using the clock selected with the Timer/Counter II clock selector. It generates an interrupt upon each underflow if the interrupt is enabled with the TnDIEN bit.

### 15.2.2 Mode 2: Dual Input Capture

Mode 2 is the Dual Input Capture mode, which measures the elapsed time between occurrences of external events, and which also provides a separate general-purpose timer/counter.

Figure15 is a block diagram of the Multi-Function Timer configured to operate in Mode 2. The time base of the capture timer depends on Timer/Counter I, which counts down using the clock selected with the Timer/Counter I clock selector.

The TnA and TnB pins function as capture inputs. A transition received on the TnA pin transfers the timer contents to the TnCRA register. Similarly, a transition received on the TnB pin transfers the timer contents to the TnCRB register. Each input pin can be configured to sense either rising or falling edges.

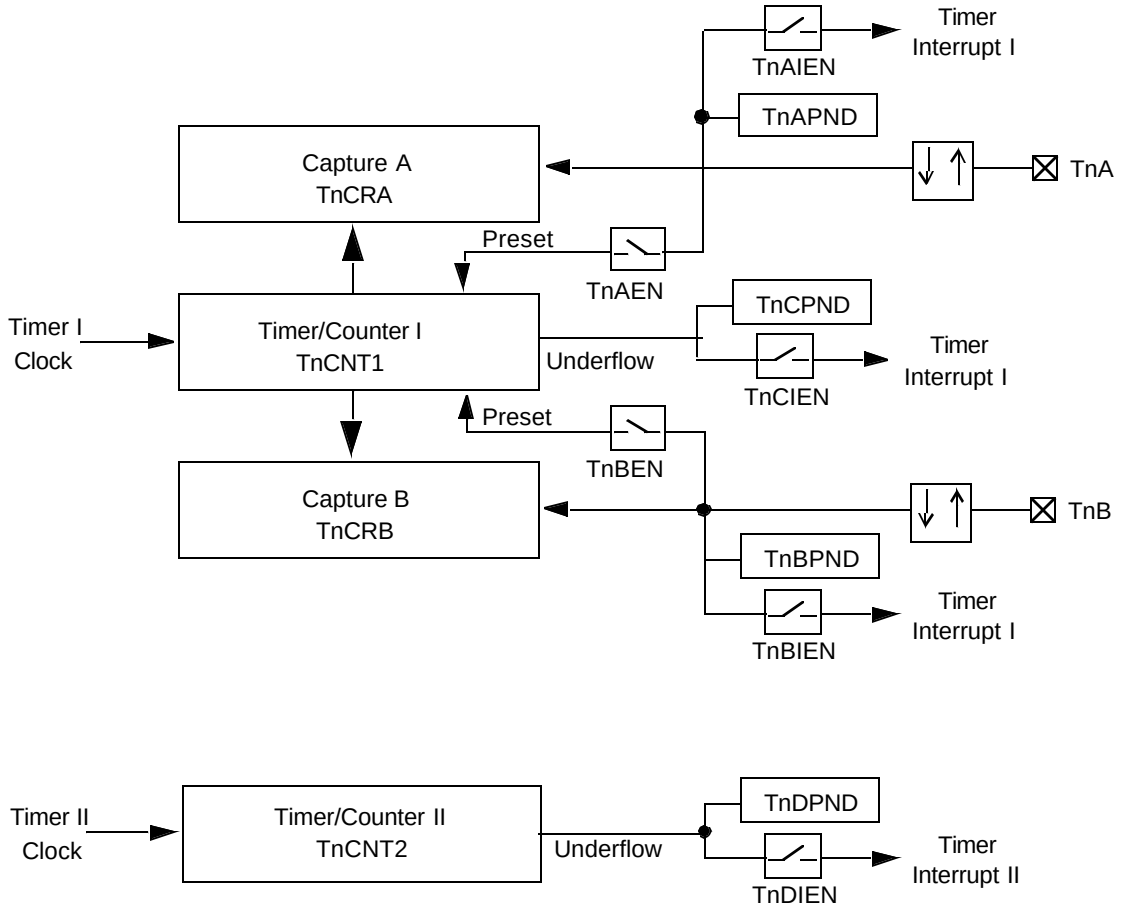



**Figure 15. Mode 2: Dual Input Capture Block Diagram**

The TnA and TnB inputs can be configured to preset the counter to FFFF hex upon reception of a valid capture event. In this case, the current value of the counter is transferred to the corresponding capture register and then the counter is preset to FFFF hex. Using this approach allows the software to determine the on-time and off-time and period of an external signal with a minimum of CPU overhead.

The values captured in the TnCRA register at different times reflect the elapsed time between transitions on the TnA pin. The same is true for the TnCRB register and the TnB pin. The input signal on TnA or TnB must have a pulse width equal to or greater than one system clock cycle.

There are three separate interrupts associated with the capture timer, each with its own enable bit and pending flag. The three interrupt events are reception of a transition on TnA, reception of a transition on TnB, and underflow of the TnCNT1 counter. The enable bits for these events are TnAIEN, TnBIEN, and TnCIEN, respectively.

In Mode 2, Timer/Counter II (TnCNT2) can be used as a simple system timer. The clock counts down using the clock selected with the Timer/Counter II clock selector. It generates an interrupt upon each underflow if the interrupt is enabled with the TnDIEN bit.

Neither Timer/Counter I (TnCNT1) nor Timer/Counter II (TnCNT2) can be configured to operate as an external event counter or to operate in the pulse accumulate mode because the TnB input is used as a capture input. Attempting to select one of these configurations will cause one or both counters to stop.

### 15.2.3 Mode 3: Dual Independent Timer/Counter

Mode 3 is the Dual Independent Timer mode, which generates system timing signals or counts occurrences of external events.

Figure16 is a block diagram of the Multi-Function Timer configured to operate in Mode 3. The timer is configured to operate as a dual independent system timer or dual external event counter. In addition, Timer/Counter I can generate a 50% duty cycle PWM signal on the TnA pin. The TnB pin can be used as an external event input or pulse accumulate input and can be used as the clock source for either Timer/Counter I or Timer/Counter II. Both counters can also be clocked by the prescaled system clock.

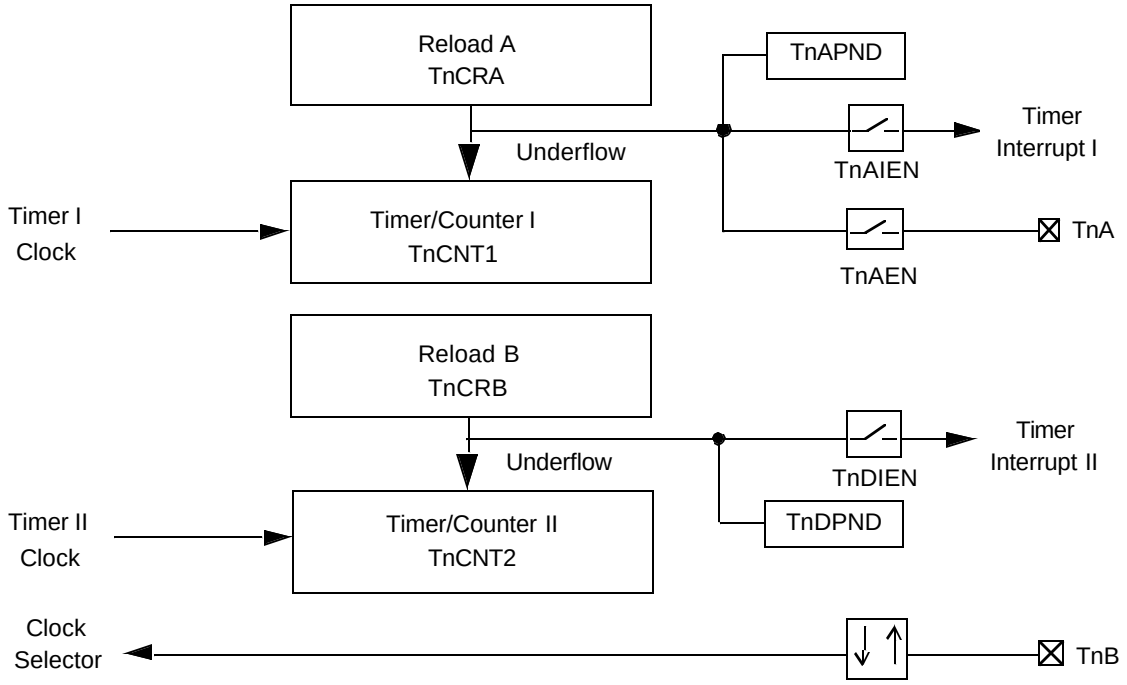



**Figure 16. Mode 3: Dual Independent Timer/Counter Block Diagram**

Timer/Counter I (TnCNT1) counts down at the rate of the selected clock. Upon underflow, it is reloaded from the TnCRA register and counting proceeds down from the reloaded value. In addition, the TnA pin is toggled on each underflow if this function is enabled by the TnAEN bit. The initial state of the TnA pin is software-programmable. When the TnA pin is toggled from low to high, it sets the TnCPND interrupt pending flag and also generates an interrupt if the interrupt is enabled by the TnAIEN bit.

Because TnA toggles on every underflow, a 50% duty cycle PWM signal can be generated on TnA without any further action from the CPU once the pulse train is initiated.

Timer/Counter II (TnCNT2) counts down at the rate of the selected clock. Upon underflow, it is reloaded from the TnCRB register and counting proceeds down from the reloaded value. In addition, each underflow sets the TnDPND interrupt pending flag and generates an interrupt if the interrupt is enabled by the TnDIEN bit.

#### 15.2.4 Mode 4: Input Capture Plus Timer

Mode 4 is the Single Input Capture and Single Timer mode, which provides one external event counter and one system timer.

Figure17 is a block diagram of the Multi-Function Timer configured to operate in Mode 4. This mode offers a combination of Mode 3 and Mode 2 functions. Timer/Counter I is used as a system timer as in Mode 3 and Timer/Counter II is used as a capture timer as in Mode 2, but with a single input rather than two inputs.

Timer/Counter I (TnCNT1) operates the same as in Mode 3. It counts down at the rate of the selected clock. Upon underflow, it is reloaded from the TnCRA register and counting proceeds down from the reloaded value. The TnA pin is toggled on each underflow if this function is enabled by the TnAEN bit. When the TnA pin is toggled from low to high, it sets the TnCPND interrupt pending flag and also generates an interrupt if the interrupt is enabled by the TnAIEN bit. A 50% duty cycle PWM signal can be generated on TnA without any further action from the CPU once the pulse train is initiated.

Timer/Counter II (TnCNT1) counts down at the rate of the selected clock. The TnB pin functions as the capture input. A transition received on TnB transfers the timer contents to the TnCRB register. The input pin can be configured to sense either rising or falling edges.

The TnB input can be configured to preset the counter to FFFF hex upon reception of a valid capture event. In this case, the current value of the counter is transferred to the capture register and then the counter is preset to FFFF hex.

The values captured in the TnCRB register at different times reflect the elapsed time between transitions on the TnA pin. The input signal on TnB must have a pulse width equal to or greater than one system clock cycle.

There are two separate interrupts associated with the capture timer, each with its own enable bit and pending flag. The two interrupt events are reception of a transition on TnB and underflow of the TnCNT2 counter. The enable bits for these events are TnBIEN and TnDIEN, respectively.

Neither Timer/Counter I (TnCNT1) nor Timer/Counter II (TnCNT2) can be configured to operate as an external event

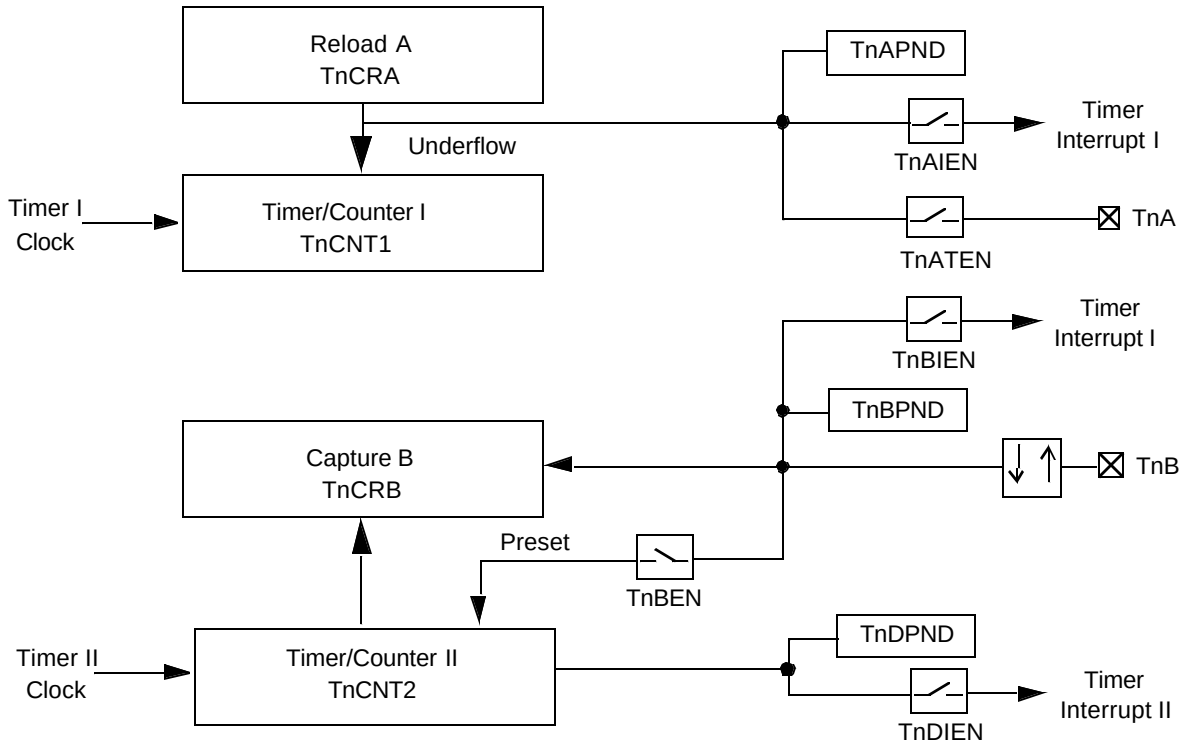



**Figure 17. Mode 4: Input Capture Plus Timer Block Diagram**

counter or to operate in the pulse accumulate mode because the TnB input is used as a capture input. Attempting to select one of these configurations will cause one or both counters to stop. In this mode, Timer/Counter II must be enabled at all times.

## 15.3 TIMER INTERRUPTS

Each Multi-Function Timer unit has four interrupt sources, designated A, B, C, and D. Interrupt sources A, B, and C are mapped into a single system interrupt called Timer Interrupt I, while interrupt source D is mapped into a system interrupt called Timer Interrupt II. Each of the four interrupt sources has its own enable bit and pending flag. The enable flags are named TnAIEN, TnBIEN, TnCIEN, and TnDIEN. The pending flags are named TnAPND, TnBPND, TnCPND, and TnDPND.

For Multi-Function Timer unit MFT1, Timer Interrupts I and II are system interrupts T1A and T1B (IRQ13 and IRQ12), respectively. For Multi-Function Timer unit MFT2, Timer Interrupts I and II are system interrupts T2A and T2B (IRQ11 and IRQ10), respectively.

Table15 shows the events that trigger interrupts A, B, C, and D in each of the four operating modes. Note that some interrupt sources are not used in some operating modes, as indicated by the notation “N/A” (Not Applicable) in the table.

## 15.4 TIMER I/O FUNCTIONS

Each Multi-Function Timer unit uses two I/O pins, called T1A and T1B (for Timer MFT1) or T2A and T2B (for Timer MFT2). The function of each pin depends on the timer operating mode and the TnAEN and TnBEN enable bits. Table16 shows the functions of the pins in each operating mode, and for each combination of enable bit settings.

When pin TnA is configured to operate as a PWM output (TnAEN = 1), the state of the pin is toggled on each underflow of the TnCNT1 counter. In this case, the initial value on the pin is determined by the TnAOUT bit. For example, to start with TnA high, the software should set the TnAOUT bit to 1 prior to enabling the timer clock. This option is available only when the timer is configured to operate in Mode 1, 3, or 4 (in other words, when TnCRA is not used in Capture mode).

**Table 15 Timer Interrupts Overview**

| Sys. Int. | Interrupt pending flag | Mode 1 | Mode 2 | Mode 3 | Mode 4 |
|---|---|---|---|---|---|
| | | PWM + Counter | Dual Input Capture + counter | Dual Counter | Single Capture + counter |
| Timer Int. I (TnA Int.) | TnAPND | TnCNT1 reload from TnCRA | Input capture on TnA transition | TnCNT1 reload from TnCRA | TnCNT1 reload from TnCRA |
| | TnBPND | TnCNT1 reload from TnCRB | Input Capture on TnB transition | N/A | Input Capture on TnB transition |
| | TnCPND | N/A | TnCNT1 underflow | N/A | N/A |
| Timer Int. II (TnB Int.) | TnDPND | TnCNT2 underflow | TnCNT2 underflow | TnCNT2 reload from TnCRB | TnCNT2 underflow |

**Table 16 Timer I/O Functions**

| I/O | TnAEN TnBEN | Mode 1 | Mode 2 | Mode 3 | Mode 4 |
|---|---|---|---|---|---|
| | | PWM + Counter | Dual Input Capture + counter | Dual Counter | Single Capture + counter |
| TnA | TnAEN=0 TnBEN=X | No Output | Capture TnCNT1 into TnCRA | No Output toggle | No Output toggle |
| | TnAEN=1 TnBEN=X | Toggle Output on underflow of TnCNT1 | Capture TnCNT1 into TnCRA and preset TnCNT1 | Toggle Output on underflow of TnCNT1 | Toggle Output on underflow of TnCNT1 |
| TnB | TnAEN=X TnBEN=0 | Ext. Event or Pulse Accumulate Input | Capture TnCNT1 into TnCRB | Ext. Event or Pulse Accumulate Input | Capture TnCNT2 into TnCRB |
| | TnAEN=X TnBEN=1 | Ext. Event or Pulse Accumulate Input | Capture TnCNT1 into TnCRB and preset TnCNT1 | Ext. Event or Pulse Accumulate Input | Capture TnCNT2 into TnCRB and preset TnCNT2 |

## 15.5 TIMER REGISTERS

The following CPU-accessible registers are used to control the Multi-Function Timers:

— Clock Prescaler Register (TnPRSC)
— Clock Unit Control Register (TnCKC)
— Timer/Counter I Register (TnCNT1)
— Timer/Counter II Register (TnCNT2)
— Reload/Capture A Register (TnCRA)
— Reload/Capture B Register (TnCRB)
— Timer Mode Control Register (TnCTRL)
— Timer Interrupt Control Register (TnICTL)
— Timer Interrupt Clear Register (TnICLR)

### 15.5.1 Clock Prescaler Register (TnPRSC)

The Clock Prescaler (TnPRSC) register is a byte-wide, read/write register that holds the current value of the 5-bit clock prescaler (CLKPS). This register is cleared upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | CLKPS | | | | |

CLKPS — Clock Prescaler. When the timer is configured to use the prescaled clock, the system clock is divided by CLKPS+1 to produce the timer clock. Thus, the system clock divide-by factor can range from 1 to 32.

### 15.5.2 Clock Unit Control Register (TnCKC)

The Clock Unit Control (TnCKC) register is a byte-wide, read/write register that selects the clock source for each timer/counter. Selecting the clock source also starts the counter. This register is cleared upon reset, which disables the timer/counters. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | C2CSEL | | | C1CSEL | | |

C1CSEL — Counter I Clock Select. This 3-bit field defines the clock mode for Timer/Counter I as follows:

000 = no clock (timer/counter I stopped)
001 = prescaled system clock
010 = external event on TnB (modes 1 and 3 only)
011 = pulse accumulate mode based on TnB (modes 1 and 3 only)
100 = slow clock *
other values = undefined

C2CSEL — Counter II Clock Select. This 3-bit field defines the clock mode for Timer/Counter II as follows:

000 = no clock (Timer/Counter II stopped modes 1, 2, and 3 only)
001 = prescaled system clock
010 = external event on TnB (modes 1 and 3 only)
011 = pulse accumulate mode based on TnB (modes 1 and 3 only)
100 = slow clock *
other values = undefined

* Operation of the slow clock is determined by the CRC-TRL.SCLK control bit, as described in Section12.6.1.

### 15.5.3 Timer/Counter I Register (TnCNT1)

The Timer/Counter I (TnCNT1) register is a word-wide, read/write register that holds the current count value for Timer/Counter I. The register contents are not affected by a reset and are unknown upon power-up.

### 15.5.4 Timer/Counter II Register (TnCNT2)

The Timer/Counter II (TnCNT2) register is a word-wide, read/write register that holds the current count value for Timer/Counter II. The register contents are not affected by a reset and are unknown upon power-up.

### 15.5.5 Reload/Capture A Register (TnCRA)

The Reload/Capture A (TnCRA) register is a word-wide, read/write register that holds the reload or capture value for Timer/Counter I. The register contents are not affected by a reset and are unknown upon power-up.

### 15.5.6 Reload/Capture B Register (TnCRB)

The Reload/Capture B (TnCRB) register is a word-wide, read/write register that holds the reload or capture value for Timer/Counter II. The register contents are not affected by a reset and are unknown upon power-up.

### 15.5.7 Timer Mode Control Register (TnCTRL)

The Timer Mode Control (TnCTRL) register is a byte-wide, read/write register that sets the operating mode of the timer/counter and the TnA and TnB pins. This register is cleared upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | TnAOUT | TnBEN | TnAEN | TnBEDG | TnAEDG | MDSEL | |

MDSEL — Mode Select. This 2-bit field sets the operating mode of the timer/counter as follows:

00 = Mode 1: PWM plus system timer
01 = Mode 2: Dual Input Capture plus system timer
10 = Mode 3: Dual Timer/Counter
11 = Mode 4: Single Input Capture and Single Timer

TnAEDG — TnA Edge Polarity. When cleared (0), input pin TnA is sensitive to falling edges (high to low transitions). When set (1), input pin TnA is sensitive to rising edges (low to high transitions).

TnBEDG — TnB Edge Polarity. When cleared (0), input pin TnB is sensitive to falling edges (high to low transitions). When set (1), input pin TnB is sensitive to rising edges (low to high transitions). In pulse accumulate mode, when this bit is set (1), the counter is enabled only when TnB is high; when this bit is cleared (0), the counter is enabled only when TnB is low.

TnAEN — TnA Enable. When set (1), the TnA pin is enabled to operate as a preset input or as a PWM output, depending on the timer operating mode. In Mode 2 (Dual Input Capture), a transition on the TnA pin presets the TnCNT1 counter to FFFF hex. In the other modes, TnA functions as a PWM output. When this bit is

cleared (0), operation of the pin for the timer/counter is disabled.

TnBEN — TnB Enable. When set (1), the TnB pin in enabled to operate in Mode 2 (Dual Input Capture) or Mode 4 (Single Input Capture and Single Timer). A transition on the TnB pin presets the corresponding timer/counter to FFFF hex (TnCNT1 in Mode 2 or TnCNT2 in Mode 4). When this bit is cleared (0), operation of the pin for the timer/counter is disabled. This bit setting has no effect in Mode 1 or Mode 3.

TnAOUT — TnA Output Data. This is a status bit that indicates the current state of the TnA pin when the pin is used as a PWM output. When set (1), the TnA pin is high; when cleared (0), the TnA pin is low. The hardware sets and clears this bit, but the software can also read or write this bit at any time and thus control the state of the output pin. In case of conflict, a software write has precedence over a hardware update. This bit setting has no effect when TnA is used as an input.

### 15.5.8 Timer Interrupt Control Register (TnICTL)

The Timer Interrupt Control (TnICTL) register is a byte-wide, read/write register that contains the interrupt enable bits and interrupt pending bits for the four timer interrupt sources, designated A, B, C, and D. The condition that causes each type of interrupt depends on the operating mode, as shown in Table15.

This register is cleared upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TnDIEN | TnCIEN | TnBIEN | TnAIEN | TnDPND | TnCPND | TnBPND | TnAPND |

TnAPND — Timer Interrupt Source A Pending. When this bit is set (1), it indicates that timer interrupt condition “A” has occurred. When this bit is cleared (0), it indicates that the interrupt condition has not occurred. For an explanation of interrupt conditions A, B, C, and D, see Table15
This bit can be set by the hardware or by the software. To clear this bit, the software must use the Timer Interrupt Clear Register (TnICLR). Any attempt by the software to directly write a 0 to this bit is ignored.

TnBPND — Timer Interrupt Source B Pending. See the description of TnAPND.

TnCPND — Timer Interrupt Source C Pending. See the description of TnAPND.

TnDPND — Timer Interrupt Source D Pending. See the description of TnAPND.

TnAIEN — Timer Interrupt A Enable. When set (1), this bit enables an interrupt on each occurrence of interrupt condition “A.” When cleared (0), an occurrence of interrupt condition “A” does not generate an interrupt to the CPU, but still sets the associated pending flag (TnAPND). For an explanation of interrupt conditions A, B, C, and D, see Table15.

TnBIEN — Timer Interrupt B Enable. See the description of TnAIEN.

TnCIEN — Timer Interrupt C Enable. See the description of TnAIEN.

TnDIEN — Timer Interrupt D Enable. See the description of TnAIEN.

### 15.5.9 Timer Interrupt Clear Register (TnICLR)

The Timer Interrupt Clear (TnICLR) register is a byte-wide, write-only register that allows the software to clear the TnAPND, TnBPND, TnCPND, and TnDPND bits in the Timer Interrupt Control (TnICTRL) register. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | TnDCLR | TnCCLR | TnBCLR | TnACLR |

TnACLR — Timer Pending A Clear. When written with a 1, the Timer Interrupt Source A Pending bit (TnAPND) is cleared in the Timer Interrupt Control register (TnICTL). Writing a 0 to the TnACLR bit has no effect.

TnBCLR — Timer Pending B Clear. See the description of TnACLR.

TnCCLR — Timer Pending C Clear. See the description of TnACLR.

TnDCLR — Timer Pending D Clear. See the description of TnACLR.

# 16.0 Versatile-Timer-Unit (VTU)

The Versatile Timer Unit (VTU) contains four fully independent 16-bit timer subsystems. Each timer subsystem can operate either as dual 8-bit PWM timers, as a single 16-bit PWM timer, or as a 16-bit counter with 2 input capture channels. These timer subsystems offers an 8-bit clock prescaler to accommodate a wide range of system frequencies.

The Versatile Timer Unit offers the following features:

- The Versatile Timer Unit (VTU) can be configured to provide:
  - — Eight fully independent 8-bit PWM channels
  - — Four fully independent 16-bit PWM channels
  - — Eight 16-bit input capture channels
- The VTU consists of four timer subsystems, each of which contains:
  - — a 16-bit counter
  - — two 16-bit capture / compare registers
  - — an 8-bit fully programmable clock prescaler
- Each of the four timer subsystems can operate in the following modes:
  - — low power mode, i.e. all clocks are stopped
  - — dual 8-bit PWM mode
  - — 16-bit PWM mode
  - — dual 16-bit input capture mode
- The Versatile-Timer-Unit controls a total of eight I/O pins, each of which can function as either:
  - — PWM output with programmable output polarity
  - — Capture input with programmable event detection and timer reset
- A flexible interrupt scheme with
  - — four separate system level interrupt requests
  - — a total of 16 interrupt sources each with a separate interrupt pending flag and interrupt enable bit

## 16.1 VTU FUNCTIONAL DESCRIPTION

The Versatile-Timer-Unit (VTU) is comprised of four timer subsystems. Each timer subsystem contains an 8-bit clock prescaler, a 16-bit up-counter and two 16-bit registers. Each timer subsystem controls two I/O pins which either function as PWM outputs or capture inputs depending on the mode of operation. There are four system level interrupt requests, one for each timer subsystem. Each system level interrupt request is controlled by four interrupt pending flags with associated enable/disable bits. All four timer subsystems are fully independent and each may operate as a dual 8-bit PWM timer, a 16-bit PWM timer or as a dual 16-bit capture timer. Figure 18 illustrates the main elements of the Versatile-Timer-Unit (VTU).

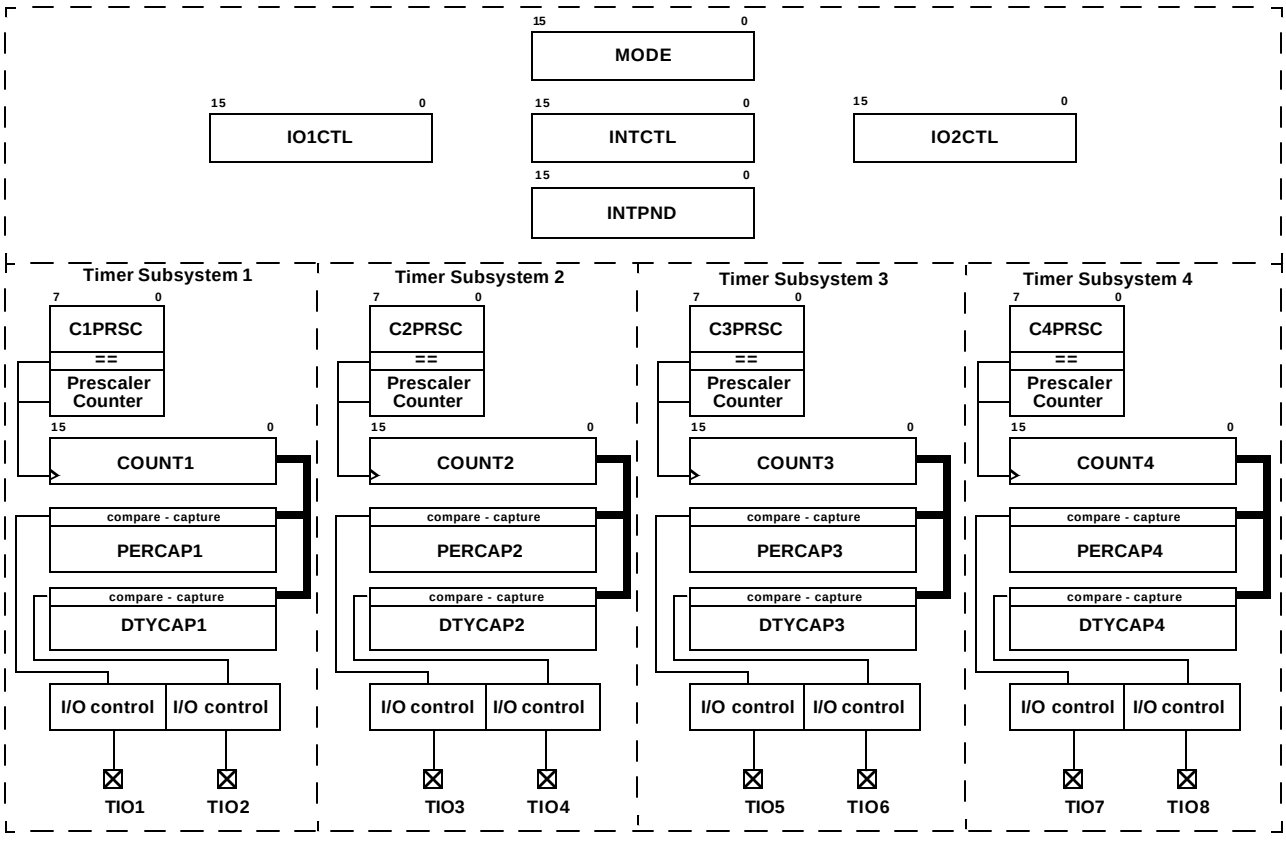



**Figure 18. VTU Block Diagram**

### 16.1.1 Dual 8-bit PWM Mode

Each timer subsystem may be configured to generate two fully independent PWM waveforms on the respective TIOx pins. In this mode, the counter COUNTx is split and operates as two independent 8-bit counters. Each counter increments at the rate determined by the clock prescaler.

Each of the two 8-bit counters may be started and stopped separately via the associated TxRUN bits. Once either of the two 8-bit timers is running the clock prescaler starts counting. Once the clock prescaler counter value matches the value of the associated CxPRSC register field, COUNTx is incremented.

The period of the PWM output waveform is determined by the value of the PERCAPx register. The TIOx output starts at the default value as pro-grammed via the IOxCTL.PxPOL bit. Once the counter value reaches the value of the period register PERCAPx, the counter is reset to $00_{16}$ upon the next counter increment. Upon the following increment from $00_{16}$ to $01_{16}$, the TIOx output will change to the opposite of the default value.

The duty cycle of the PWM output waveform is controlled by the DTYCAPx register value. Once the counter value reaches the value of the duty cycle register DTYCAPx, the PWM output TIOx changes back to its default value upon the next counter increment. Figure19 illustrates this concept.

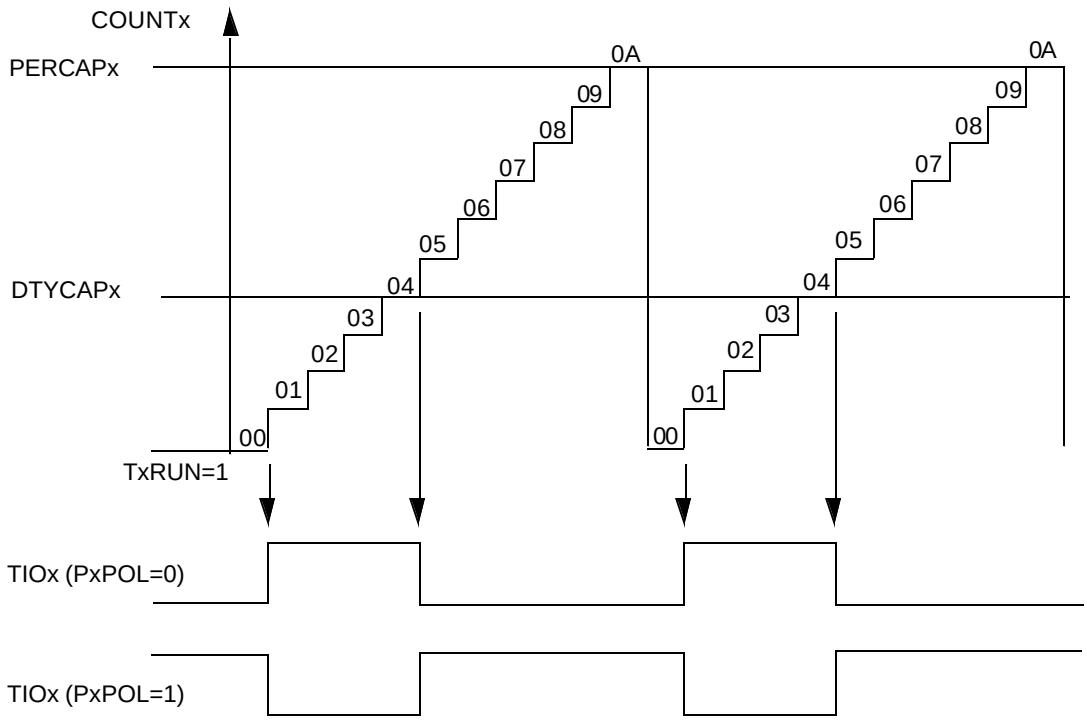



**Figure 19. VTU PWM generation**

The period time is determined by the following formula:

$$PWMperiod = (PERCAPx + 1) * (CxPRSC + 1) * T_{CLK}$$

The duty cycle in percent is calculated as follows:

$$DutyCycle[\%] = (DTYCAPx / (PERCAPx+1)) *100$$

If the duty cycle register (DTYCAPx) holds a value which is greater then the value held in the period register (PERCAPx) the TIOx output will remain at the opposite of its default value which corresponds to a duty cycle of 100%. If the duty cycle register (DTYCAPx) register holds a value of $00_{16}$, the TIOx output will remain at the default value which corresponds to a duty cycle of 0%. In that case the value contained in the PERCAPx register is irrelevant. This scheme allows the duty cycle to be programmed in a range from 0% to 100%.

In order to allow fully synchronized updates of the period and duty cycle compare values, the PERCAPx and DTYCAPx registers are double buffered when operating in PWM mode. Therefore if the user writes to either the period or duty cycle register while either of the two PWM channels is enabled, the new value will not take effect until the counter value matches the previous period value or the timer is stopped.

Reading the PERCAPx or DTYCAPx register will always return the most recent value written to it.

The counter registers can be written if both 8-bit counters are stopped. This allows the user to preset the counters before starting and therefore generate PWM output waveforms with a phase shift relative to one another. If the counter is written with a value other then $00_{16}$ it will start incrementing from that value while TIOx remains at its default value until the first $00_{16}$ to $01_{16}$ transition of the counter value occurs. If the counter is preset to values which are smaller or equal then the value held in the period register (PERCAPx) the counter will count up until a match between the counter value and the PERCAPx register value occurs. The counter will then be reset to $00_{16}$ and continue counting up. Alternatively the counter may be written with a value which is greater then the

value held in the period register. In that case the counter will count up to $FF_{16}$ and then roll over to $00_{16}$. In any case the TIOx pin always changes its state at the $00_{16}$ to $01_{16}$ transition of the counter.

The user software may only write to the COUNTx register if both TxRUN bits of a timer subsystem are cleared. Any writes to the counter register while either timer is running will be ignored.

The two I/O pins associated with a timer subsystem function as independent PWM outputs in the dual 8-bit PWM mode. If a PWM timer is stopped via its associated MODE.TxRUN bit the following actions result:

— The associated TIOx pin will return to its default value as defined by the IOxCTL.PxPOL bit.
— The counter will stop and will retain its last value.
— Any pending updates of the PERCAPx and DTYCAPx register will be completed.
— The prescaler counter will be stopped and reset if both MODE.TxRUN bits are cleared.

Figure20 illustrates the configuration of a timer subsystem while operating in dual 8-bit PWM mode. The numbering in Figure20 refers to timer subsystem 1 but equally applies to the other three timer subsystems.

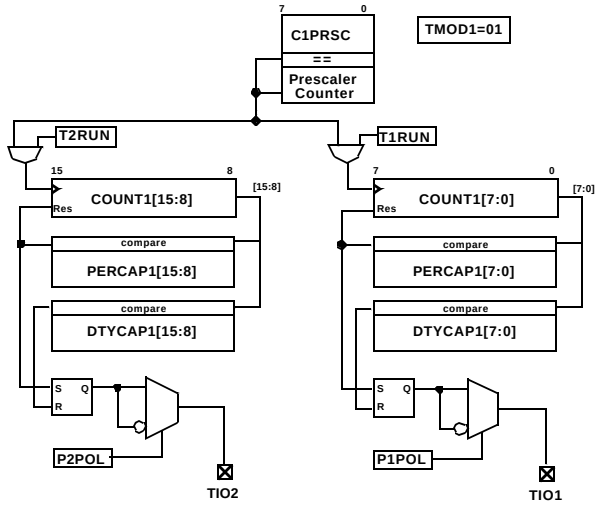



**Figure 20. VTU Dual 8-bit PWM Mode**

### 16.1.2 16-Bit PWM Mode

Each of the four timer subsystems may be independently configured to provide a single 16-bit PWM channel. In this case the lower and upper bytes of the counter are concatenated to form a single 16-bit counter.

Operation in 16-bit PWM mode is conceptually identical to the dual 8-bit PWM operation as outlined under Dual 8-bit PWM Mode on page 59. The 16-bit timer may be started or stopped with the lower MODE.TxRUN bit, i.e. T1RUN for timer subsystem 1.

The two TIOx outputs associated with a timer subsystem can be used to produce either two identical PWM waveforms or two PWM waveforms of opposite polarities. This can be accomplished by setting the two PxPOL bits of the respective timer subsystem to either identical or opposite values.

Figure21 illustrates the configuration of a timer subsystem while operating in 16-bit PWM mode. The numbering in Figure21 refers to timer subsystem 1 but equally applies to the other three timer subsystems.

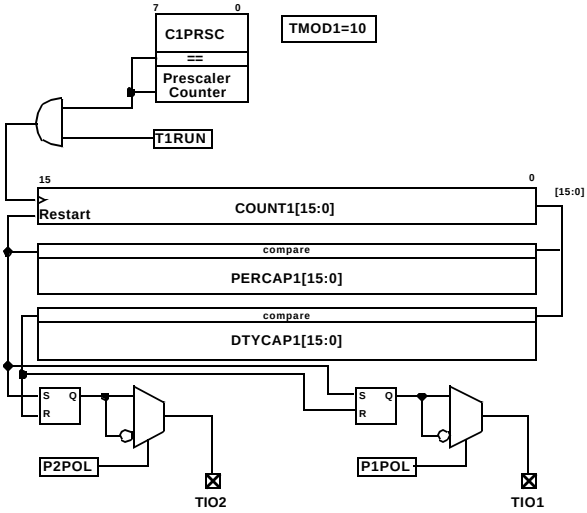



**Figure 21. VTU 16-bit PWM Mode**

### 16.1.3 Dual 16-Bit Capture Mode

In addition to the two PWM modes, each timer subsystem may be configured to operate in an input capture mode which provides two 16-bit capture channels. The input capture mode can be used to precisely measure the period and duty cycle of external signals.

In capture mode the counter COUNTx operates as a 16-bit up-counter while the two TIOx pins associated with a timer subsystem operate as capture inputs. A capture event on the TIOx pins causes the contents of the counter register (COUNTx) to be copied to the PERCAPx or DTYCAPx registers respectively.

Starting the counter is identical to the 16-bit PWM mode, i.e. setting the lower of the two MODE.TxRUN bits will start the counter and the clock prescaler. In addition, the capture event inputs are enabled once the MODE.TxRUN bit is set.

The TIOx capture inputs can be independently configured to detect a capture event on either a positive transition, a negative transition or both a positive and a negative transition. In addition, any capture event may be used to reset the counter COUNTx and the clock prescaler counter. This avoids the need for the user software to keep track of timer overflow conditions and greatly simplifies the direct frequency and duty cycle measurement of an external signal.

Figure22 illustrates the configuration of a timer subsystem while operating in capture mode. The numbering in Figure22

refers to timer subsystem 1 but equally applies to the other three timer subsystems.

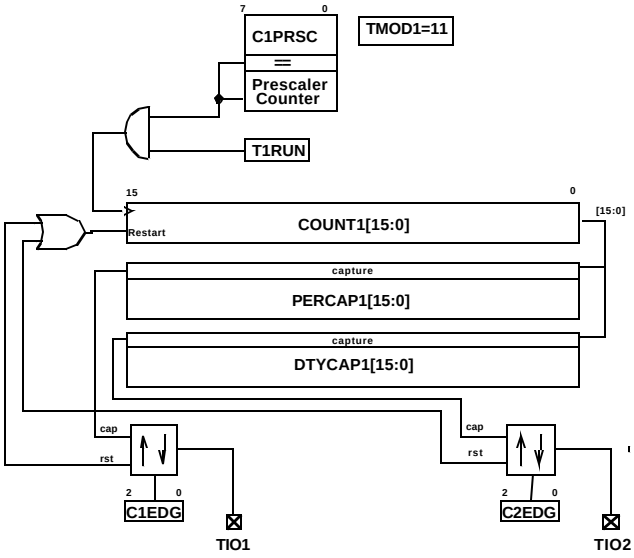



**Figure 22. VTU Dual 16-bit Capture Mode**

### 16.1.4 Low Power Mode

In case a timer subsystem is not used, the user can place it in a low-power-mode. All clocks to a timer subsystem are stopped and the counter and prescaler contents are frozen once low-power-mode is entered. The user may continue to write to the MODE, INTCTL, IOxCTL and CLKxPS registers. Write operations to the INTPND register are allowed; but if a timer subsystem is in low power mode, its associated interrupt pending bits cannot be cleared. The user cannot write to the COUNTx, PERCAPx and DTYCAPx registers of a timer subsystem while it is in low-power-mode. All registers can be read at any time.

### 16.1.5 Interrupts

The Versatile-Timer-Unit (VTU) has a total of 16 interrupt sources, four for each of the four timer subsystems. All interrupt sources have a pending flag and an enable bit associated with them. All interrupt pending flags are denoted IxAPD through IxDPD where “x” relates to the specific timer subsystem. There is one system level interrupt request for each of the four timer subsystems.

Figure23 illustrates the interrupt structure of the versatile timer module.

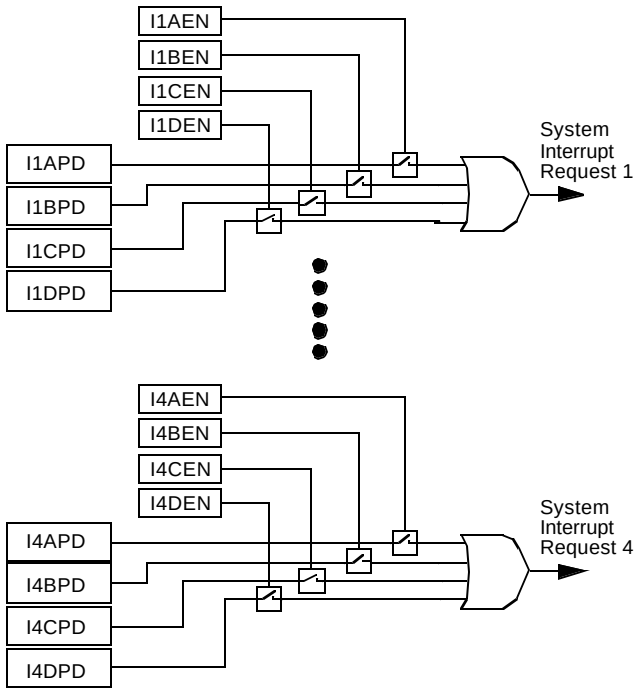



**Figure 23. VTU Interrupt Request Structure**

Each of the timer pending flags - IxAPD through IxDPD - is set by a specific hardware event depending on the mode of operation, i.e., PWM or Capture mode. Table17 outlines the specific hardware events relative to the operation mode which cause an interrupt pending flag to be set.

**Table 17 VTU Interrupt Sources**

| Pending Flag | Dual 8-bit PWM Mode | 16-bit PWM Mode | Capture Mode |
|---|---|---|---|
| IxAPD | Low Byte Duty Cycle match | Duty Cycle match | Capture to DTYCAPx |
| IxBPD | Low Byte Period match | Period match | Capture to PERCAPx |
| IxCPD | High Byte Duty Cycle match | N/A | Counter Overflow |
| IxDPD | High Byte Period match | N/A | N/A |

### 16.1.6 ISE Mode operation

The VTU supports breakpoint operation of the In-System-Emulator (ISE). If FREEZE is asserted, all timer counter clocks will be inhibited and the current value of the timer registers will be frozen; in capture mode, all further capture events are disabled. Once FREEZE becomes inactive, counting will resume from the previous value and the capture input events are re-enabled.

## 16.2 VTU REGISTERS

The Versatile-Timer-Unit contains a total of 19 user accessible registers. All registers are word-wide and are initialized to a known value upon reset. All software accesses to the VTU registers must be word accesses.

### 16.2.1 Mode Control Register (MODE)

The Mode Control (MODE) registries a word-wide read/write register which controls the mode selection of all four timer subsystems. The register is cleared ($0000_{16}$) upon reset.

| 15 14 | 13 | 12 | 11 10 | 9 | 8 |
|---|---|---|---|---|---|
| TMOD4 | T8RUN | T7RUN | TMOD3 | T6RUN | T5RUN |

| 7 6 | 5 | 4 | 3 2 | 1 | 0 |
|---|---|---|---|---|---|
| TMOD2 | T4RUN | T3RUN | TMOD1 | T2RUN | T1RUN |

TxRUN Timer start/stop. If set (1), the associated counter and clock prescaler is started depending on the mode of operation. Once set, the clock to the clock prescaler and the counter are enabled and the counter will increment each time the clock prescaler counter value matches the value defined in the associated clock prescaler field (CxPRSC).

TMODx Timer System Operating Mode. This 2-bit wide field enables or disables the Timer Subsystem and defines it's operating mode.

00: Low-Power-Mode enabled. All clocks to the counter subsystem are stopped. The counter is stopped regardless of the value of the TxRUN bits. Read operations to the Timer Subsystem will return the last value; the user shall not perform any write operations to the Timer Subsystem while it is disabled since those will be ignored.

01: Dual 8-bit PWM mode enabled. Each 8-bit counter may individually be started or stopped via its associated TxRUN bit. The TIOx pins will function as PWM outputs.

10: 16-bit PWM mode enabled. The two 8-bit counters are concatenated to form a single 16-bit counter. The counter may be started or stopped with the lower of the two TxRUN bits, i.e. T1RUN, T3RUN, T5RUN and T7RUN. The TIOx pins will function as PWM outputs.

11: Capture Mode enabled. Both 8-bit counters are concatenated and operate as a single 16-bit counter. The counter may be started or stopped with the lower of the two TxRUN bits, i.e., T1RUN, T3RUN, T5RUN and T7RUN. The TIOx pins will function as capture inputs.

### 16.2.2 I/O Control Register 1 (IO1CTL)

The I/O Control Register 1 (IO1CTL) is a word-wide read/write register. The register controls the functionality of the I/O pins TIO1 through TIO4 depending on the selected mode of operation. The register is cleared ($0000_{16}$) upon reset.

| 15 | 14 12 | 11 | 10 8 | 7 | 6 4 | 3 | 2 0 |
|---|---|---|---|---|---|---|---|
| P4POL | C4EDG | P3POL | C3EDG | P2POL | C2EDG | P1POL | C1EDG |

CxEDG Capture Edge Control. Defines the polarity of a capture event and the reset of the counter. The value of this three bit field has no effect while operating in PWM mode.

| CxEDG | Capture | Counter Reset |
|---|---|---|
| 000 | rising edge | No |
| 001 | falling edge | No |
| 010 | rising edge | Yes |
| 011 | falling edge | Yes |
| 100 | both edges | No |
| 101 | both edges | rising edge |
| 110 | both edges | falling edge |
| 111 | both edges | both edges |

PxPOL PWM Polarity. While operating in PWM mode the bit defines the output polarity of the corresponding PWM output (TIOx).

0 = The PWM output is set (1) upon the $00_{16}$ to $01_{16}$ transition of the counter and will be reset (0) once the counter value matches the duty cycle value.

1 = The PWM output is reset (0) upon the $00_{16}$ to $01_{16}$ transition of the counter and will be set (1) once the counter value matches the duty cycle value.

Once a counter is stopped, the output will assume the value of PxPOL, i.e., its initial value. The PxPOL bit has no effect while operating in capture mode.

### 16.2.3 I/O Control Register 2 (IO2CTL)

The I/O Control Register 2 (IO2CTL) is a word-wide read/write register. The register controls the functionality of the I/O pins TIO5 through TIO8 depending on the selected mode of operation. The register is cleared (0000) upon reset.

| 15 | 14 12 | 11 | 10 8 | 7 | 6 4 | 3 | 2 0 |
|---|---|---|---|---|---|---|---|
| P8POL | C8EDG | P7POL | C7EDG | P6POL | C6EDG | P5POL | C5EDG |

The functionality of the bit fields of the IO2CTL register is identical to the ones described in the IO1CTL register section.

### 16.2.4 Interrupt Control Register (INTCTL)

The Interrupt Control (INTCTL) register is a word-wide read/write register. It contains the interrupt enable bits for all 16 interrupt sources of the Versatile-Timer-Unit. Each interrupt enable bit corresponds to an interrupt pending flag located in the Interrupt Pending Register (INTPND). All INTCTL register bits are solely under software control. The register is cleared ($0000_{16}$) upon reset..

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| I4DEN | I4CEN | I4BEN | I4AEN | I3DEN | I3CEN | I3BEN | I3AEN |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| I2DEN | I2CEN | I2BEN | I2AEN | I1DEN | I1CEN | I1BEN | I1AEN |

IxAEN Timer x interrupt A enable. Enable/Disable an interrupt request based on the corresponding IxAPD flag being set. The associated IxAPD

flag will be updated regardless of the value of the IxAEN bit.

0 Enable system interrupt request for the IxAPD pending flag

1 Disable system interrupt request for the IxAPD pending flag

IxBEN Timer x interrupt B enable. Enable/Disable an interrupt request based on the corresponding IxBPD flag being set. The associated IxBPD flag will be updated regardless of the value of the IxBEN bit.

0 Enable system interrupt request for the IxBPD pending flag

1 Disable system interrupt request for the IxBPD pending flag

IxCEN Timer x interrupt C enable. Enable/Disable an interrupt request based on the corresponding IxCPD flag being set. The associated IxCPD flag will be updated regardless of the value of the IxCEN bit.

0 Enable system interrupt request for the IxCPD pending flag

1 Disable system interrupt request for the IxCPD pending flag

IxDEN Timer x interrupt D enable. Enable/Disable an interrupt request based on the corresponding IxDPD flag being set. The associated IxDPD flag will be updated regardless of the value of the IxDEN bit.

0 Enable system interrupt request for the IxDPD pending flag

1 Disable system interrupt request for the IxDPD pending flag

### 16.2.5 Interrupt Pending Register (INTPND)

The Interrupt Pending (INTPND) register is a word-wide read/write register which contains all 16 interrupt pending flags. There are four interrupt pending flags called IxAPD through IxDPD per timer subsystem. Each interrupt pending flag is set by a hardware event and can be cleared if the user software writes a 1 to the bit position. The value will remain unchanged if a 0 is written to the bit position. All interrupt pending flags are cleared (0) upon reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| I4DPD | I4CPD | I4BPD | I4APD | I3DPD | I3CPD | I3BPD | I3APD |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| I2DPD | I2CPD | I2BPD | I2APD | I1DPD | I1CPD | I1BPD | I1APD |

IxAPD Timer x interrupt A pending. If set (1), indicates that an interrupt condition for the related timer subsystem has occurred. Table 17 on page 61 lists the hardware condition which causes this bit to be set.

IxBPD Timer x interrupt B pending. If set (1), indicates that an interrupt condition for the related timer subsystem has occurred. Table 17 on page 61 lists the hardware condition which causes this bit to be set.

IxCPD Timer x interrupt C pending. If set (1), indicates that an interrupt condition for the related timer subsystem has occurred. Table 17 on page 61 lists the hardware condition which causes this bit to be set.

IxDPD Timer x interrupt D pending. If set (1), indicates that an interrupt condition for the related timer subsystem has occurred. Table 17 on page 61 lists the hardware condition which causes this bit to be set.

### 16.2.6 Clock Prescaler Register 1 (CLK1PS)

CLK1PS is a word-wide read/write register. The register is split into two 8-bit wide field called C1PRSC and C2PRSC. Each field holds the 8-bit clock prescaler compare value for timer subsystems 1 and 2 respectively. The register is cleared upon reset.

| 15 8 | 7 0 |
|---|---|
| C2PRSC | C1PRSC |

C1PRSC Clock Prescaler 1 compare value. Holds the 8-bit prescaler value for timer subsystem 1. The counter of timer subsystem is incremented each time when the clock prescaler compare value matches the value of the clock prescaler counter. The divide-by-ratio is equal to C1PRSC+1 i.e. a value of $00_{16}$ results in a divide by 1 whereas the maximum divide-by ratio is 256 for a C1PRSC value of $FF_{16}$.

C2PRSC Clock Prescaler 2 compare value. Holds the 8-bit prescaler value for timer subsystem 2. The functionality of this field is identical to the one described for C1PRSC in the previous paragraph.

### 16.2.7 Clock Prescaler Register 2 (CLK2PS)

The Clock Prescaler Register 2 (CLK2PS) is a word-wide read/write register. The register is split into two 8-bit wide fields called C3PRSC and C4PRSC. Each field holds the 8-bit clock prescaler compare value for timer subsystems 3 and 4 respectively. The register is cleared upon reset.

| 15 8 | 7 0 |
|---|---|
| C4PRSC | C3PRSC |

C3PRSC Clock Prescaler 3 compare value. Holds the 8-bit prescaler value for timer subsystem 3. The functionality of this field is identical to the one described for C1PRSC on page 63.

C4PRSC Clock Prescaler 4 compare value. Holds the 8-bit prescaler value for timer subsystem 4. The functionality of this field is identical to the one described for C1PRSC on page 63.

### 16.2.8 Counter Registers (COUNTx)

The Counter (COUNTx) registers are word wide read/write registers. There are a total of four registers called COUNT1 through COUNT4, one for each of the four timer subsystems. The user software may read the registers at any time. Read-

ing the register will return the current value of the counter. The register may only be written if the counter is stopped i.e. if both TxRUN bits associated with a timer subsystem are cleared. The registers are cleared upon reset (0000).

| 15 | 0 |
|---|---|
| CNTx | |

### 16.2.9 Period/Capture Registers (PERCAPx)

The Period/Capture (PERCAPx) registers are word-wide read/write registers. There are a total of four registers called PERCAP1 through PERCAP4, one for each timer subsystem. The register hold the period compare value in PWM mode of the counter value at the time the last associated capture event occurred. In PWM mode the register is double buffered. If a new period compare value is written while the counter is running, the write will not take effect until counter value matches the previous period compare value or until the counter is stopped. Reading may take place at any time and will return the most recent value which was written. The PERCAPx registers are reset to 0000 upon reset.

| 15 | 0 |
|---|---|
| PCAPx | |

### 16.2.10 Duty Cycle / Capture Registers (DTYCAPx)

The Duty Cycle/Capture (DTYCAPx) registers are word-wide read/write registers. There are a total of four registers called DTYCAP1 through DTYCAP4, one for each timer subsystem. The registers hold the period compare value in PWM mode or the counter value at the time the last associated capture event occurred. In PWM mode the register is double buffered. If a new duty cycle compare value is written while the counter is running, the write will not take effect until the counter value matches the previous period compare value or until the counter is stopped. In other words, the update takes effect on period boundaries only. Reading may take place at any time and will return the most recent value which was written. The DTYCAPx registers are reset to $0000_{16}$ upon reset.

| 15 | 0 |
|---|---|
| DCAPx | |

# 17.0 MICROWIRE/SPI

MICROWIRE/PLUS is a synchronous serial communications protocol, originally implemented in National Semiconductor's COPS™ and HPC™ families of microcontrollers to minimize the number of connections, and therefore the cost, of communicating with peripherals.

The device has an enhanced MICROWIRE/SPI interface module (MWSPI) that can communicate with all peripherals that conform to MICROWIRE or Serial Peripheral Interface (SPI) specifications. This enhanced MICROWIRE interface is capable of operating as either a master or slave and in 8- or 16-bit mode. Figure24 shows a typical enhanced MICROWIRE interface application.

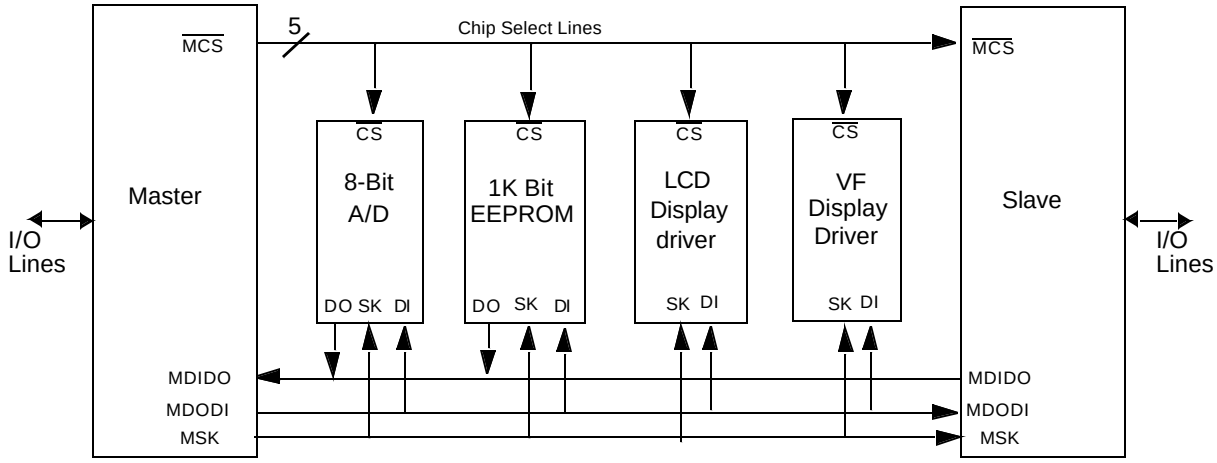



**Figure 24. MICROWIRE Interface**

The enhanced MICROWIRE interface module includes the following features:

- — Programmable operation as a Master or Slave
- — Programmable shift-clock frequency (master only)
- — Programmable 8- or 16-bit mode of operation
- — 8- or 16-bit serial I/O data shift register
- — Two modes of clocking data
- — Serial clock can be low or high when idle
- — 16-bit read buffer
- — Busy flag, Read Buffer Full flag, and Overrun flag for polling and as interrupt sources
- — Supports multiple masters
- — Maximum bit rate of 10M bits/second (master mode) 5M bits/second (slave mode) at 20MHz system clock
- — Supports very low-end slaves with the Slave Ready output
- — Echo back enable/disable (Slave only)

## 17.1 MICROWIRE OPERATION

The MICROWIRE interface allows several devices to be connected on one three-wire system. At any given time, one of these devices operates as the master while all other devices operate as slaves.

The master device supplies the synchronous clock (MSK) for the serial interface and initiates the data transfer. The slave devices respond by sending (or receiving) the requested data. Each slave device uses the master's clock for serially shifting data out (or in), while the master shifts the data in (or out).

The three-wire system includes: the serial data in signal (MDIDO for master mode, MDODI for slave mode), the serial data out signal (MDODI for master mode, MDIDO for slave mode) and the serial clock (MSK).

In slave mode, an optional fourth signal ($\overline{MCS}$) may be used to enable the slave transmit. At any given time, only one slave can respond to the master. Each slave device has its own chip select signal ($\overline{MCS}$) for this purpose.

The MICROWIRE interface allows the device to operate either as a master or slave transferring 8- or 16-bits of data. This is configured via the MMNS bit.

Figure25 shows a block diagram of the enhanced MICROWIRE serial interface in the device.

### 17.1.1 Shifting

The MICROWIRE interface is a full duplex transmitter/receiver. A 16-bit shifter, which can be split into a low and high byte, is used for both transmitting and receiving. In 8-bit mode, only the lower 8-bits are used to transfer data. The transmitted data is shifted out through MDODI pin (master mode) or MDIDO pin (slave mode), starting with the most significant bit. At the same time, the received data is shifted in through MDIDO pin (master mode) or MDODI pin (slave mode), also starting with the most significant bit first.

The shift in and shift out are controlled by the MSK clock. In each clock cycle of MSK, one bit of data is transmitted/received. The 16-bit shifter is accessible via the MWDAT register. Reading the MWDAT register returns the value in the read buffer. Writing to the MWDAT register updates the 16-bit shifter.

### 17.1.2 Reading

The enhanced MICROWIRE interface implements a double buffer on read. As illustrated in Figure25, the double read

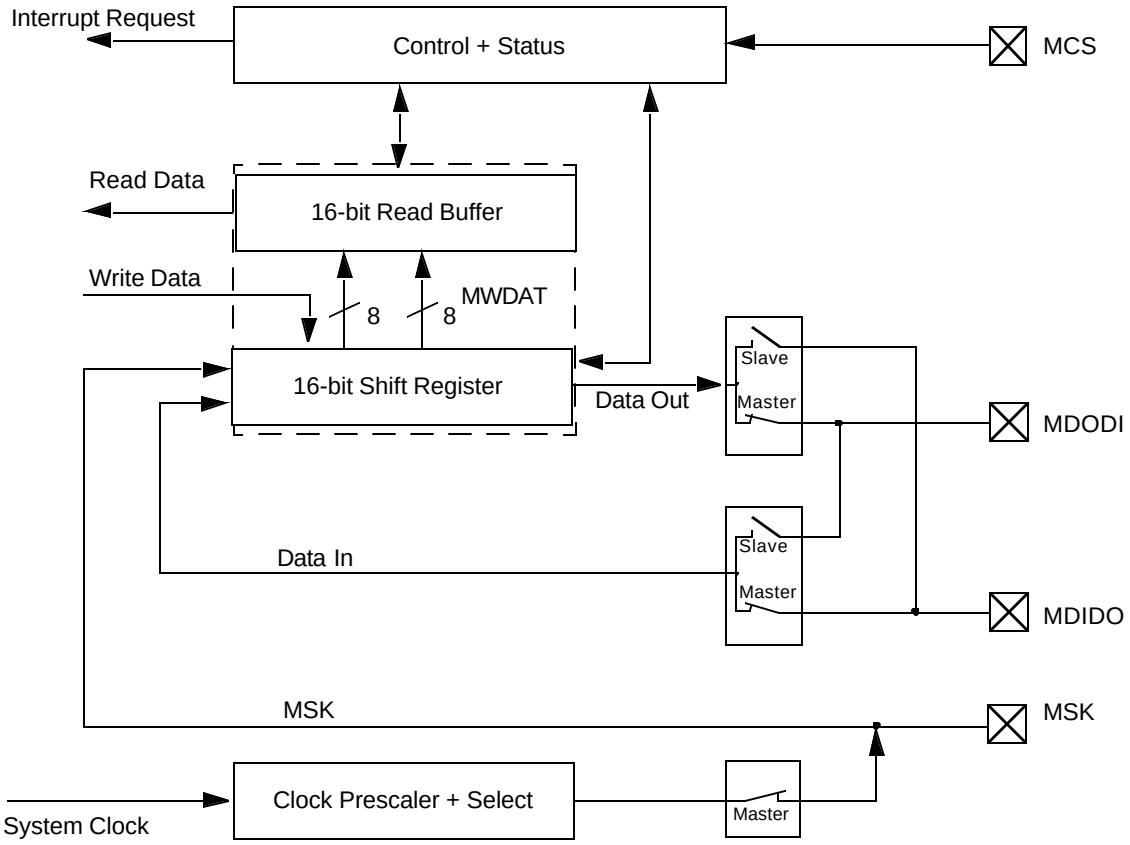



**Figure 25. MICROWIRE Block Diagram**

buffer consists of the 16-bit shifter and a buffer, called the read buffer.

The 16-bit shifter loads the read buffer with new data when the data transfer sequence is completed and previous data in the read buffer has been read. In master mode, an Overrun error occurs when the read buffer is full, the 16-bit shifter is full and a new data transfer sequence starts.

When 8-bit mode is selected, the lower byte of the shift register is loaded into the lower byte of the read buffer and the read buffer's higher byte remains unchanged.

The "Receive Buffer Full" (MRBF) bit indicates if the MWDAT register holds valid data. The MOVR bit indicates that an overrun condition has occurred.

#### 17.1.3 Writing

The "MICROWIRE Busy" (MBSY) bit indicates whether the MWDAT register can be written. All write operations to the MWDAT register update the shifter while the data contained in the read buffer is not affected. Undefined results will occur if the MWDAT register is written to while the MBSY bit is set to 1.

#### 17.1.4 Clocking Modes

Two clocking modes are supported: the normal mode and the alternate mode.

In the normal mode, the output data, which is transmitted on the MDODI pin (master mode) or the MDIDO pin (slave mode), is clocked out on the falling edge of the shift clock MSK. The input data, which is received via the MDIDO pin (master mode) or the MDODI pin (slave mode), is sampled on the rising edge of MSK.

In the alternate mode, the output data is shifted out on the rising edge of MSK on the MDODI pin (master mode) or MDIDO pin (slave mode). The input data, which is received via MDIDO pin (master mode) or MDODI pin (slave mode), is sampled on the falling edge of MSK.

The clocking modes are selected with the MSKM bit. The MIDL bit allows selection of the value of MSK when it is idle (when there is no data being transferred). Various MSK clock frequencies can be programmed via the MCDV bits. Figures 27, 28, 29, and 30 show the data transfer timing for the normal and the alternate modes with the MIDL bit equal to 0 and equal to 1.

Note that when data is shifted out on MDODI (master mode) or MDIDO (slave mode) on the leading edge of the MSK clock, bit 14 (16-bit mode) is shifted out on the second leading edge of the MSK clock. When data are shifted out on MDODI (master mode) or MDIDO (slave mode) on the trailing edge of MSK, bit 14 (16-bit mode) is shifted out on the first trailing edge of MSK.

### 17.2 MASTER MODE

In Master mode, the MSK pin is an output for the shift clock, MSK. When data is written to the (MWnDAT register), eight or sixteen MSK clocks, depending on the mode selected, are generated to shift the eight or sixteen bits of data and then

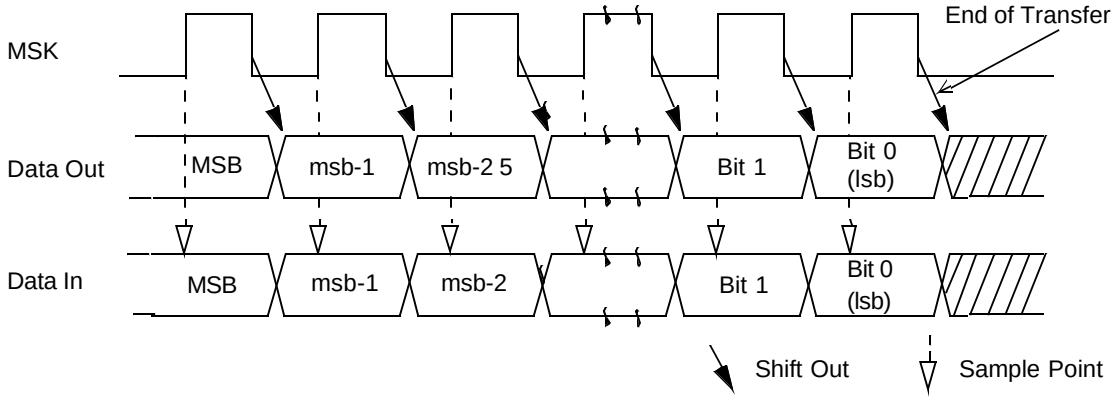



**Figure 26. Normal Mode, MIDL Bit = 0**

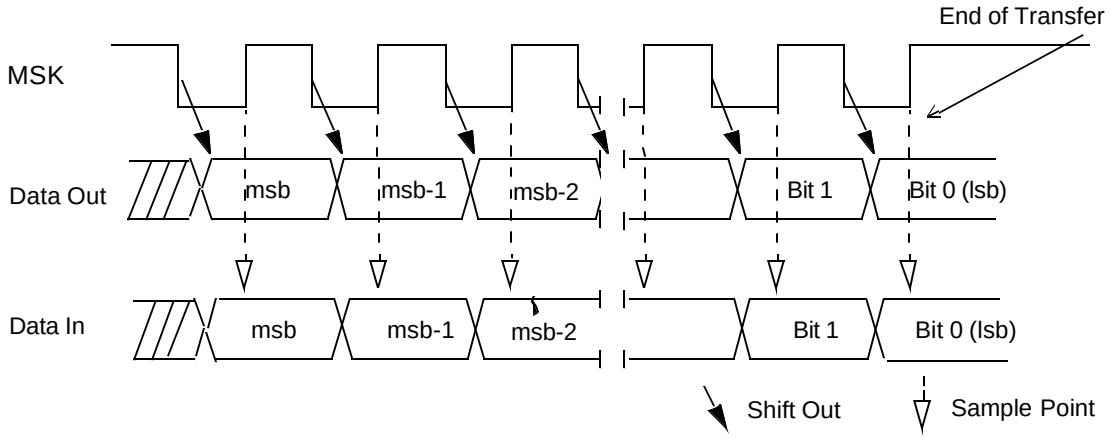



**Figure 27. Normal Mode, MIDL Bit = 1**

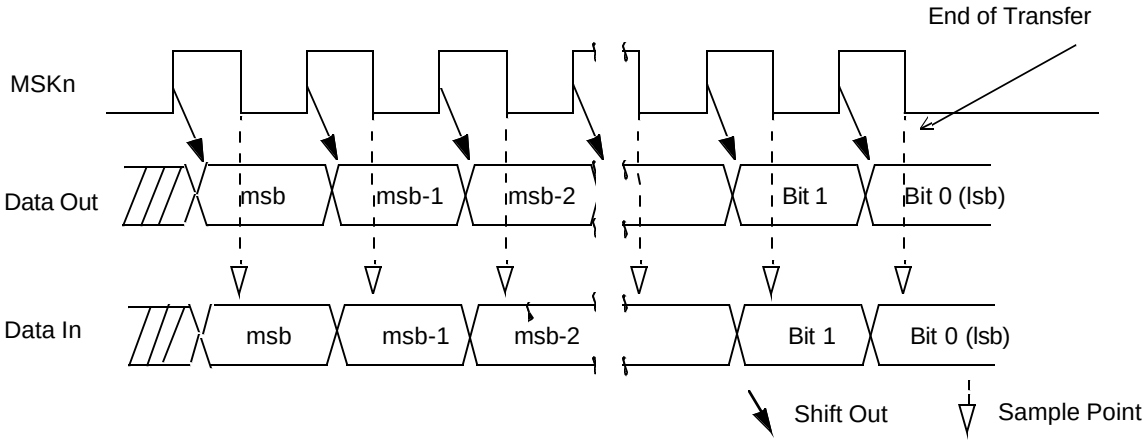



**Figure 28. Alternate Mode, MIDL Bit = 0**

MSK goes idle again. The MSK idle state can be either high or low, depending on the MIDL bit.

### 17.3 SLAVE MODE

In Slave mode, the MSK pin is an input for the shift clock MSK. MDIDO is placed in TRI-STATE mode when $\overline{MCS}$ is inactive. Data transfer is enabled when $\overline{MCS}$ is active.

The slave starts driving MDIDO when $\overline{MCS}$ is activated. The most significant bit (lower byte in 8-bit mode or upper byte in 16-bit mode) is output onto the MDIDO pin first. After eight or sixteen clocks (depending on the selected mode), the data transfer is completed.

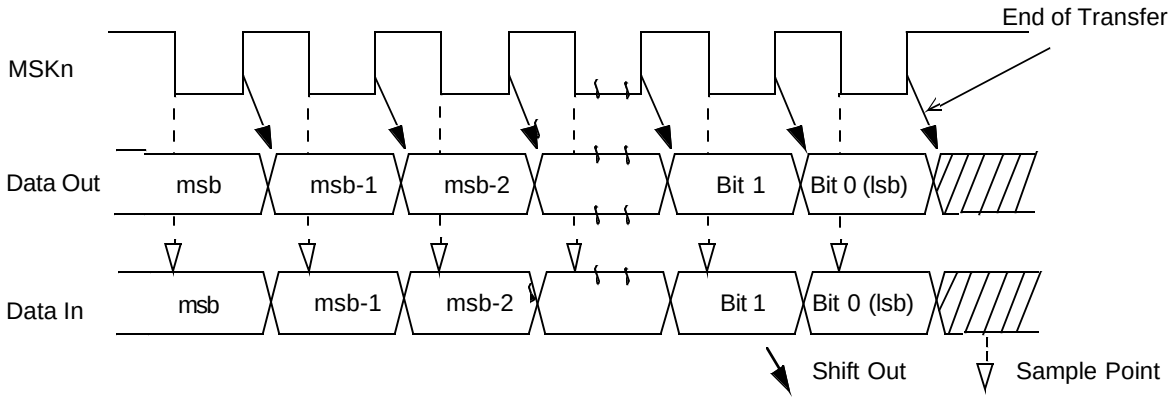



**Figure 29. Alternate Mode, MIDL Bit = 1**

If a new shift process starts before MWDAT was written, i.e., while MWDAT does not contain any valid data, and the "Echo Enable" (MECH) bit is set to 1, the data received from MDODI is transmitted on MDIDO in addition to being shifted to MWDAT. If the MECH bit is cleared to 0, the data transmitted on MDIDO is the data held in the MWDAT register, regardless of its validity. The master may negate the $\overline{MCS}$ signal to synchronize the bit count between the master and the slave. In the case that the slave is the only slave in the system, $\overline{MCS}$ can be tied to $V_{SS}$.

## 17.4 INTERRUPT GENERATION

An interrupt is generated in any of the following cases:

- When the read buffer is full (MRBF=1) and the "Enable Interrupt for Read" bit is set (MEIR=1).
- Whenever the shifter is not busy, i.e. the MBSY bit is cleared (MBSY=0) and the "Enable Interrupt for Write" bit is set (MEIW=1).
- When an overrun condition occurs (MOVR is set to 1) and the "Enable Interrupt on Overrun" bit is set (MEIO=1). This usage is restricted to master mode.

Figure30 illustrates the various interrupt capabilities of this module.

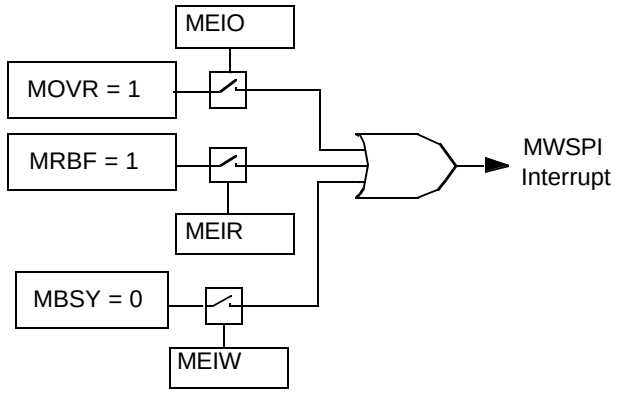



**Figure 30. MWSPI Interrupts**

## 17.5 MICROWIRE INTERFACE REGISTERS

The software interacts with the MICROWIRE interface by accessing the MICROWIRE registers. There are five such registers:

- MICROWIRE Data Register (MWDAT)
- MICROWIRE Control Register (MWCTL)
- MICROWIRE Status Register (MWSTAT)

### 17.5.1 MICROWIRE Data Register (MWDAT)

The MWDAT register is a word-wide, read/write register used to transmit and receive data through the MDODI and MDIDO pins. Figure31 shows the hardware structure of the register.

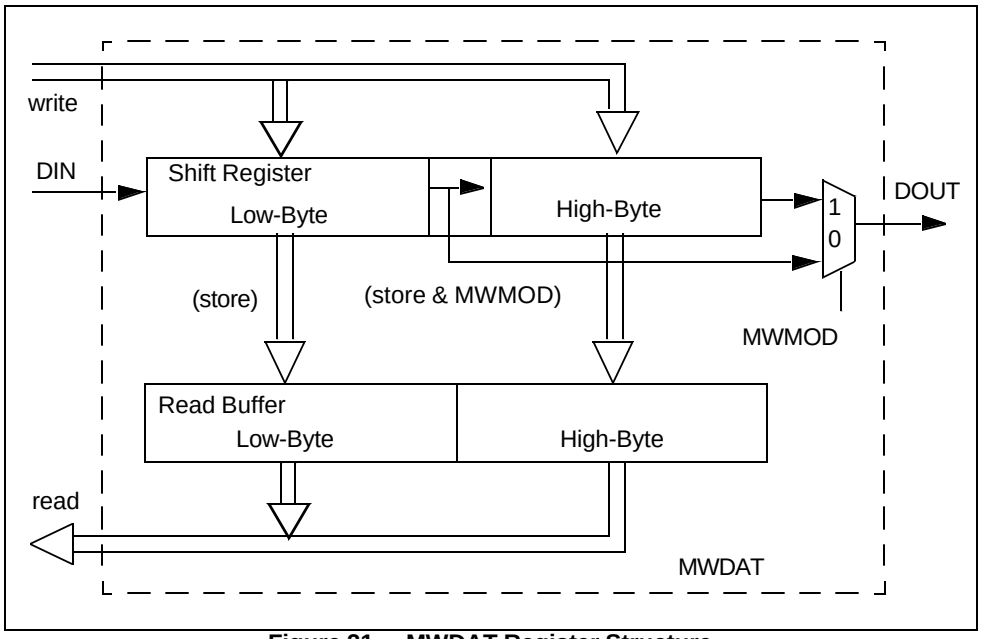



**Figure 31. MWDAT Register Structure**

### 17.5.2 MICROWIRE Control Register (MWCTL)

Upon reset, all non-reserved bits are cleared to 0. The register format is shown below.

| 15 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| MCDV [6:0] | MIDL | MSKM | MEIW | MEIR | MEIO | MECH | MMOD | MMNS | MEN |

MEN MICROWIRE Enable. This bit enables (1) or disables (0) the MICROWIRE interface module. Clearing this bit disables the module, clears the status bits in the MICROWIRE status register (the MBSY, MRBF, and MOVR flags in MWSTAT), and places the MICROWIRE interface pins in the states described in Table18.

**Table 18 Pin Values with MICROWIRE Disabled**

| | |
|---|---|
| MSK | Master: MnIDL Bit<br>Slave: input |
| $\overline{MCS}$ | Input |
| MDIDO | Master: input<br>Slave: TRI-STATE |
| MDODI | Master: known Value<br>Slave: input |

MMNS MICROWIRE Master/Slave Select. When cleared to 0, the device operates as a slave. When set to 1, the device operates as the master.

MMOD MICROWIRE Mode Select (8- or 16-bit). When set to 0, the device operates in 8-bit mode. When set to 1, the device operates in 16-bit mode. This bit should only be changed when the module is disabled or the MICROWIRE interface is idle (MWSTAT.MBSY=0).

MECH MICROWIRE Echo Back. This bit enables (1) or disables (0) the echo back function in slave mode. This bit should be written only when the MICROWIRE interface is idle (MWSTAT.MBSY=0). The MECH bit is ignored in master mode. The MWDAT register is valid from the time the register has been written until the end of the transfer.

In the echo back mode, MDODI is transmitted (echoed back) on MDIDO if MWDAT does not contain any valid data. With the echo back function disabled, the data held in the MWDAT register is transmitted on MDIDO, whether or not the data is valid.

MEIO MICROWIRE Enable Interrupt on Overrun. This bit enables or disables the overrun error interrupt. When set to 1, an interrupt is generated when the Receive Overrun Error flag (MWSTAT.MOVR) is set. Otherwise, no interrupt is generated when an overrun error occurs. This bit should only be enabled in master mode.

MEIR MICROWIRE Enable Interrupt for Read. When set to 1, an interrupt is generated when the Read Buffer Full flag (MWSTAT.MRBF) is set. Otherwise, no interrupt is generated when the read buffer is full.

MEIW MICROWIRE Enable Interrupt for Write. When set to 1, an interrupt is generated when the Busy bit (MWSTAT.MBSY) is cleared, which indicates that a data transfer sequence has been completed and the read buffer is ready to receive the new data. Otherwise, no interrupt is generated when the Busy bit is cleared.

MSKM MICROWIRE Clocking Mode. When cleared to 0, the device uses the normal clocking mode. When set to 1, the device uses the alternate

clocking mode. In the normal mode, the output data is clocked out on the falling edge of MSK and the input data is sampled on the rising edge of MSK. In the alternate mode, the output data is clocked out on the rising edge of MSK and the input data is sampled on the falling edge of MSK.

MIDL MICROWIRE Idle. This bit sets the value of the MSK output when the MICROWIRE interface is idle: 0 for low or 1 for high. This bit should be changed only when the MICROWIRE interface module is disabled (MEN=0) or when no bus transaction is in progress (MWSTAT.MBSY=0).

MCDV MICROWIRE Clock Divider Value. This 7-bit field specifies the divide-by factor used for generating the MSK shift clock from the system clock. The divide-by factor is 2*(MCDV[6:0]+1). This allows selection of a divide-by ratio from 2 to 256. This field is ignored in slave mode (MWCTL1.MMNS=0).

### 17.5.3 MICROWIRE Status Register (MWSTAT)

The MICROWIRE Status Register is a word-wide, read-only register that shows the current status of the MICROWIRE interface module. Upon reset, all non-reserved bits are cleared to 0. The register format is shown below.

| 15 3 | 2 | 1 | 0 |
|---|---|---|---|
| Reserved | MOVR | MRBF | MBSY |

MBSY MICROWIRE Busy. This bit, when set to 1, indicates that the MICROWIRE shifter is busy.

In master mode, MBSY is set to 1 when the MWDAT register is written. In slave mode, this bit is set to 1 on the first leading edge of MSK when $\overline{\text{MCS}}$ is asserted or when the MWDAT register is written, whatever occurs first.

In both master and slave modes, this bit is cleared to 0 when the MICROWIRE data transfer sequence is completed and the read buffer is ready to receive the new data; in other words, when the previous data held in the read buffer has already been read.

If the previous data in the read buffer has not been read and a new data has been received into the shift register, the MBSY will not be cleared, as the transfer could not be completed. This is because the contents of the shift register could not be copied into the read buffer.

MRBF MICROWIRE Read Buffer Full. This bit, when set to 1, indicates that the MICROWIRE read buffer is full and ready to be read by the software. It is set to 1 when the shifter loads the read buffer, which occurs upon completion of a transfer sequence if the read buffer is empty.

The MRBF bit is updated when the MWDAT register is read. At that time, the MRBF bit is cleared to 0 if the shifter does not contain any new data (in other words, the shifter is not receiving data or has not yet received a full byte of data). The MRBF bit remains set to 1 if the shifter already holds new data at the time that MWDAT is read. In that case, MWDAT is immediately reloaded with the new data and is ready to be read by the software.

MOVR MICROWIRE Receive Overrun Error. This bit, when set to 1 in master mode, indicates that a receive overrun error has occurred. This error occurs when the read buffer is full, the 8-bit shifter is full, and a new data transfer sequence starts. This bit is undefined in slave mode.

The MOVR bit, once set, remains set until cleared by the software. The software clears this bit by writing a 1 to its bit position. Writing a 0 to this bit position has no effect. No other bits in the MWSTAT register are affected by a write operation to the register.

# 18.0 USART

The USART module is a full-duplex Universal Synchronous/ Asynchronous Receiver/Transmitter that supports a wide range of software-programmable baud rates and data formats. It handles automatic parity generation and several error detection schemes. There are one or two independent USART modules in each device, depending on the package type.

Each USART module offers the following features:

- — Full-duplex double-buffered receiver/transmitter
- — Synchronous or asynchronous operation
- — programmable baud rate from SYS_CLK/ [2*(1+2^11)*16] up to SYSCLK/2 for USART configured to run in synchronous mode
- — programmable baud rate from SYS_CLK/ [16*(1+2^11)*16] up to SYSCLK/16 for USART configured to run in asynchronous mode
- — Programmable framing formats: seven, eight, or nine data bits; one or two stop bits; and odd, even, mark, space, or no parity
- — Hardware parity generation for data transmission and parity check for data reception
- — Interrupts on “transmit ready” and “receive ready” conditions, separately enabled
- — Software-controlled break transmission and detection
- — Internal diagnostic capability
- — Automatic detection of parity, framing, and overrun errors

## 18.1 FUNCTIONAL OVERVIEW

Figure32 is a block diagram of the USART module showing the basic functional units in the USART:

- — Transmitter
- — Receiver
- — Baud Rate Generator
- — Control and Error Detection

**Note:** In the description of the USART, the lower-case letter “n” represents the USART number. For example, TDXn means TDX1 or TDX2.

The Transmitter block consists of an 8-bit transmit shift register and an 8-bit transmit buffer. Data bytes are loaded in parallel from the buffer into the shift register and then shifted out serially on the TDXn pin.

The Receiver block consists of an 8-bit receive shift register and an 8-bit receive buffer. Data is received serially on the RDXn pin and shifted into the shift register. Once eight bits have been received, the contents of the shift register are transferred in parallel to the receive buffer.

The Transmitter and Receiver blocks both contain extensions for 9-bit data transfers, as required by the 9-bit and loopback operating modes.

The Baud Rate Generator generates the clock for the synchronous and asynchronous operating modes. It consists of two registers and a two-stage counter. The registers are used to specify a prescaler value and a baud rate divisor. The first stage of the counter divides the USART clock based on the value of the programmed prescaler to create a slower clock. The second stage of the counter divides the output of the first stage based on the programmed baud rate divisor to create the baud rate clock.

The Control and Error Detection block contains the USART control registers, control logic, error detection circuit, parity generator/checker, and interrupt generation logic. The control registers and control logic determine the data format, mode of operation, clock source, and type of parity used. The error detection circuit generates parity bits and checks for parity, framing, and overrun errors.

## 18.2 USART OPERATION

The USART has two basic modes of operation: synchronous and asynchronous. In addition, there are two special-purpose synchronous and asynchronous modes, called attention and diagnostic. This section describes the operating modes of the USART.

### 18.2.1 Asynchronous Mode

The asynchronous mode of the USART enables the device to communicate with other devices using just two communication signals: transmit and receive.

In the asynchronous mode, the transmit shift register (TSFT) and the transmit buffer (UnTBUF) double-buffer the data for transmission. To transmit a character, a data byte is loaded in the UnTBUF register. The data is then transferred to the TSFT register. While the TSFT is shifting out the current character (LSB first) on the TDXn pin, the UnTBUF register is loaded by software with the next byte to be transmitted. When TSFT finishes transmission of the last stop bit of the current frame, the contents of UnTBUF are transferred to the TSFT register and the Transmit Buffer Empty flag (UnTBE) is set. The UnTBE flag is automatically reset by the USART when the software loads a new character into the UnTBUF register. During transmission, the UnXMIP bit is set high by the USART. This bit is reset only after the USART has sent the last stop bit of the current character and the UnTBUF register is empty. The UnTBUF register is a read/write register. The TSFT register is not user accessible.

In asynchronous mode, the input frequency to the USART is 16 times the baud rate. In other words, there are 16 clock cycles per bit time. In asynchronous mode the baud rate generator is always the USART clock source.

The receive shift register (RSFT) and the receive buffer (UnRBUF) double buffer the data being received. The USART receiver continuously monitors the signal on the RDXn pin for a low level to detect the beginning of a start bit. Upon sensing this low level, the USART waits for seven input clock cycles and samples again three times. If all three samples still indicate a valid low, then the receiver considers this to be a valid start bit, and the remaining bits in the character frame are each sampled three times, around the mid-bit position. For any bit following the start bit, the logic value is found by majority voting, i.e. the two samples with the same value define the value of the data bit. Figure33 illustrates the process of start bit detection and bit sampling.

Serial data input on the RDXn pin is shifted into the RSFT register. Upon receiving the complete character, the contents of the RSFT register are copied into the UnRBUF register and the Receive Buffer Full flag (UnRBF) is set. The UnRBF

flag is automatically reset when software reads the character from the UnRBUF register. The RSFT register is not user accessible.

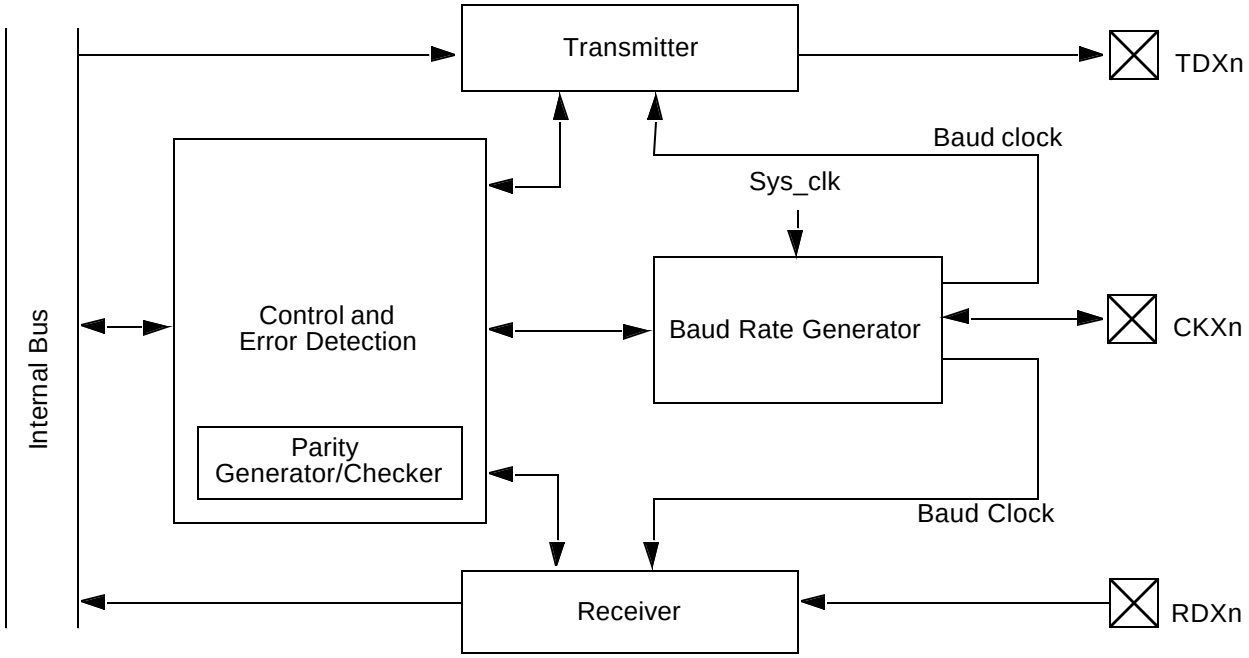



**Figure 32. USART Block Diagram**

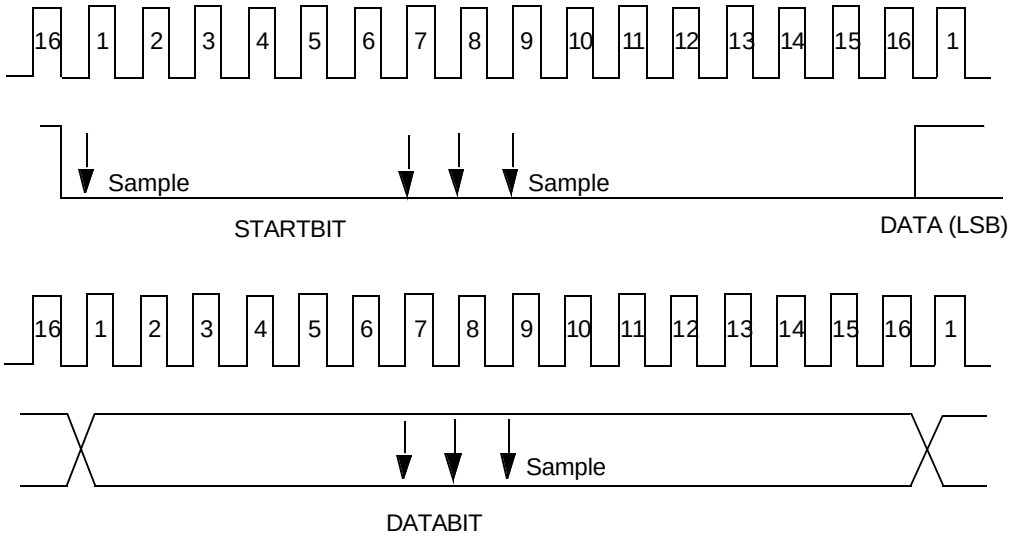



**Figure 33. USART Asynchronous Communication**

### 18.2.2 Synchronous Mode

The synchronous mode of the USART enables the device to communicate with other devices using three communication signals: transmit, receive, and clock. In this mode, data bits are transferred synchronously with the USART clock signal. Data bits are transmitted on the rising edges and received on the falling edges of the clock signal, as shown in Figure34. Data bytes are transmitted and received least significant bit (LSB) first.

In the synchronous mode, the transmit shift register (TSFT) and the transmit buffer (UnTBUF) double-buffer the data for transmission. To transmit a character, a data byte is loaded in the UnTBUF register. The data is then transferred to the TSFT register. The TSFT register shifts out one bit of the current character, LSB first, on each rising edge of the clock.

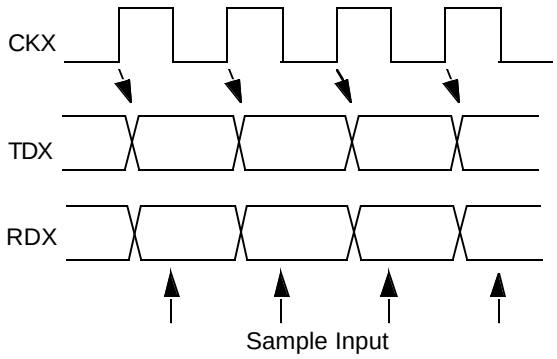



**Figure 34. USART Synchronous Communication**

While the TSFT is shifting out the current character on the TDXn pin, the UnTBUF register may be loaded by the software with the next byte to be transmitted. When the TSFT finishes transmission of the last stop bit within the current frame, the contents of UnTBUF are transferred to the TSFT register and the Transmit Buffer Empty flag (UnTBE) is set. The UnTBE flag is automatically reset by the USART when the software loads a new character into the UnTBUF register. During transmission, the UnXMIP bit is set high by the USART. This bit is reset only after the USART has sent the last frame bit of the current character and the UnTBUF register is empty.

The receive shift register (RSFT) and the receive buffer (UnRBUF) double-buffer the data being received. Serial data received on the RDXn pin is shifted into the RSFT register at the first falling edge of the clock. Each subsequent falling edge of the clock causes an additional bit to be shifted into the RSFT register. The USART assumes a complete character has been received after the correct number of rising edges on CKXn (based on the selected frame format) have been detected. Upon receiving a complete character, the contents of the RSFT register are copied into the UnRBUF register and the Receive Buffer Full flag (UnRBF) is set. The UnRBF flag is automatically reset when the software reads the character from the UnRBUF register.

The transmitter and receiver may be clocked from either an external source provided to the CKXn pin or by the internal baud rate generator. In the latter case, the clock signal is placed on the CKXn pin as an output.

### 18.2.3 Attention Mode

The Attention mode is available for networking this device with other processors. This mode requires the 9-bit data format with no parity. The number of start bits and number of stop bits are programmable. In this mode, two types of 9-bit characters are sent on the network: address characters consisting of 8 address bits and a 1 in the ninth bit position and data characters consisting of 8 data bits and a 0 in the ninth bit position.

While in Attention mode, the USART receiver monitors the communication flow but ignores all characters until an address character is received. Upon the receipt of an address character, the contents of the receive shift register are copied to the receive buffer. The UnRBF flag is set and an interrupt (if enabled) is generated. The UnATN bit is automatically reset to zero, and the USART begins receiving all subsequent characters. The software must examine the contents of the UnRBUF register and respond by accepting the subsequent characters (by leaving the UnATN bit reset) or waiting for the next address character (by setting the UnATN bit again).

The operation of the USART transmitter is not affected by the selection of this mode. The value of the ninth bit to be transmitted is programmed by setting or clearing a bit called UnXB9 in the USART Frame Select register. The value of the ninth bit received is read from UnRB9 in the USART Status Register.

### 18.2.4 Diagnostic Mode

The Diagnostic mode is available for testing of the USART. In this mode, the TDXn and RDXn pins are internally connected together, and data that is shifted out of the transmit shift register is immediately transferred to the receive shift register. This mode supports only the 9-bit data format with no parity. The number of start and stop bits is programmable.

### 18.2.5 Frame Format Selection

The format shown in Figure35 consists of a start bit, seven data bits (excluding parity), and one or two stop bits. If parity bit generation is enabled by setting the UnPEN bit, a parity bit is generated and transmitted following the seven data bits.

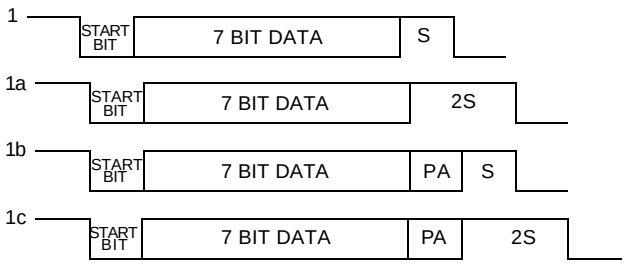



**Figure 35. Seven Data Bit Frame Options**

The format shown in Figure36 consists of one start bit, eight data bits (excluding parity), and one or two stop bits. If parity bit generation is enabled by setting the UnPEN bit, a parity bit is generated and transmitted following the eight data bits.

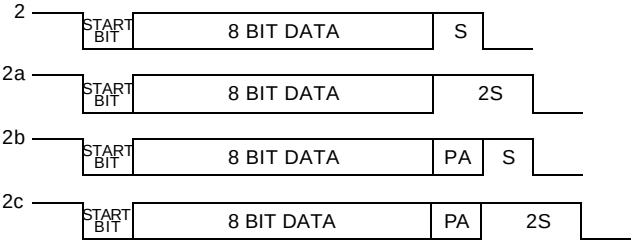



**Figure 36. Eight Data Bit Frame Options**

The format shown in Figure37 consists of one start bit, nine data bits, and one or two stop bits. This format also supports the USART attention feature. When operating in this format, all eight bits of UnTBUF and UnRBUF are used for data. The

ninth data bit is transmitted and received using two bits in the control registers, called UnXB9 and UnRB9. Parity is not generated or verified in this mode.

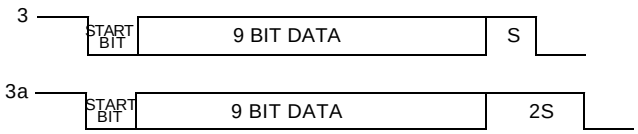



**Figure 37. Nine Data Bit Frame Options**

### 18.2.6 Baud Rate Generator

The Baud Rate Generator creates the basic baud clock from the system clock. The system clock is passed through a two-stage divider chain consisting of a 5-bit baud rate prescaler (UnPSC) and an 11-bit baud rate divisor (UnDIV).

The relationship between the 5-bit prescaler select (UnPSC) setting and the prescaler factors is shown in Table19.

**Table 19 Prescaler Factors**

| Prescaler Select | Prescaler Factor | Prescaler Select | Prescaler Factor |
|---|---|---|---|
| 00000 | 1 | 10000 | 8.5 |
| 00001 | 1 | 10001 | 9 |
| 00010 | 1.5 | 10010 | 9.5 |
| 00011 | 2 | 10011 | 10 |
| 00100 | 2.5 | 10100 | 10.5 |
| 00101 | 3 | 10101 | 11 |
| 00110 | 3.5 | 10110 | 11.5 |
| 00111 | 4 | 10111 | 12 |
| 01000 | 4.5 | 11000 | 12.5 |
| 01001 | 5 | 11001 | 13 |
| 01010 | 5.5 | 11010 | 13.5 |
| 01011 | 6 | 11011 | 14 |

**Table 19 Prescaler Factors**

| Prescaler Select | Prescaler Factor | Prescaler Select | Prescaler Factor |
|---|---|---|---|
| 01100 | 6.5 | 11100 | 14.5 |
| 01101 | 7 | 11101 | 15 |
| 01110 | 7.5 | 11110 | 15.5 |
| 01111 | 8 | 11111 | 16 |

A prescaler factor of zero corresponds to “no clock.” The “no clock” condition is the USART power down mode, in which the USART clock is turned off to reduce power consumption. The application program should select the “no clock” condition before entering a new baud rate. Otherwise, it could cause incorrect data to be received or transmitted. The UnPSR register must contain a value other than zero when an external clock is used at CKXn.

In asynchronous mode, the baud rate is calculated by:

$$BR = \frac{SYS\_CLK}{(16 \times N \times P)}$$

where BR is the baud rate, SYS_CLK is the system clock, N is the value of the baud rate divisor + 1, and P is the prescaler divide factor selected by the value in the UnPSR register.

The divide by 16 is performed because in the asynchronous mode, the input frequency to the USART is 16 times the baud rate. In synchronous mode, the input clock to the USART equals the baud rate.

### 18.2.7 Interrupts

The USART is capable of generating interrupts on:

- Receive Buffer Full
- Receive Error
- Transmit Buffer Empty

Figure38 shows a diagram of the interrupt sources and associated enable bits.

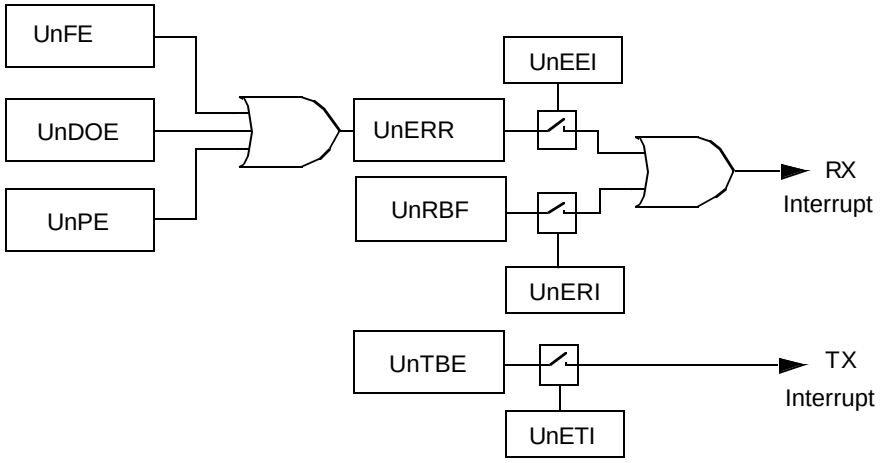



**Figure 38. USART Interrupts**

The interrupts can be individually enabled or disabled using the Enable Transmit Interrupt (UnETI), Enable Receive Interrupt (UnERI) and Enable Receive Error Interrupt (UnEER) bits in the UnICTRL register.

A transmit interrupt is generated when both the UnTBE and UnETI bits are set. To remove this interrupt, software must either disable the interrupt by clearing the UnETI bit or write to the UnTBUF register (thus clearing the UnTBE bit).

A receive interrupt is generated on two conditions:

1. Both the UnRBF and UnERI bits are set. To remove this interrupt, software must either disable the interrupt by clearing the UnERI bit or read from the UnRBUF register (thus clearing the UnRBF bit).
2. Both the UnERR and the UnEEI bits are set. To remove this interrupt the software must either disable it by clearing the UnEEI bit or read the UnSTAT register (thus clearing the UnERR bit).

#### 18.2.8 Break Generation and Detection

A line break is generated when the BRK bit is set in the UnMDSL register. The TDXn line remains low until the program resets the BRK bit.

A line break is detected if RDXn remains low for 10 bit times or longer after a missing stop bit is detected.

#### 18.2.9 Parity Generation and Detection

Parity is only generated or checked with the 7-bit and 8-bit data formats. It is not generated or checked in the diagnostic loopback mode, the attention mode, or in the normal mode with the 9-bit data format. Parity generation and checking are enabled and disabled via the PEN bit in the UnFRS register. The UnPSEL bits in the UnFRS register are used to select odd, even, mark, or space parity.

### 18.3 USART REGISTERS

The software interacts with the USART by accessing the USART registers. There are eight such registers:

— USART Receive Data Buffer (UnRBUF)
— USART Transmit Data Buffer (UnTBUF)
— USART Baud Rate Prescaler Register (UnPSR)
— USART Baud Rate Divisor Register (UnBAUD)
— USART Frame Select Register (UnFRS)
— USART Mode Select Register (UnMDSL)
— USART Status Register (UnSTAT)
— USART Interrupt Control Register (UnICTRL)

#### 18.3.1 USART Receive Data Buffer (UnRBUF)

The USART Receive Data Buffer is a byte-wide, read/write register used to receive each data byte.

#### 18.3.2 USART Transmit Data Buffer (UnTBUF)

The USART Transmit Data Buffer is a byte-wide, read/write register used to transmit each data byte.

#### 18.3.3 USART Baud Rate Prescaler (UnPSR)

The USART Baud Rate Prescaler Register is a byte-wide, read/write register that contains the 5-bit clock prescaler and the upper three bits of the baud rate divisor. This register is cleared upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UnPSC | | | | | UnDIV10 | UnDIV9 | UnDIV8 |

UnPSC Prescaler. This 5-bit field specifies the prescaler value used for dividing the system clock in the first stage of the two-stage divider chain. For the prescaler factors corresponding to each 5-bit value, see Table19.

UnDIV[10:8] Baud Rate Divisor (bits 10-8). This field contains the three highest-order bits (bits 10, 9, and 8) of the USART baud rate divisor used in the second stage of the two-stage divider chain. The remaining bits of the baud rate divisor are contained in the UnBAUD register.

#### 18.3.4 USART Baud Rate Divisor (UnBAUD)

The USART Baud Rate Divisor Register is a byte-wide, read/write register that contains the lower eight bits of the baud rate divisor. This register contents are unknown upon power-up and are left unchanged by a reset operation. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UnDIV7 | UnDIV6 | UnDIV5 | UnDIV4 | UnDIV3 | UnDIV2 | UnDIV1 | UnDIV0 |

UnDIV[7:0] Baud Rate Divisor (bits 7-0). This field contains the eight lowest-order bits of the USART baud rate divisor used in the second stage of the two-stage divider chain. The three highest-order bits are contained in the UnPSR register. The divisor value used is the 11-bit UnDIV value plus 1.

#### 18.3.5 USART Frame Select Register (UnFRS)

The USART Frame Select Register is a byte-wide, read/write register that controls the frame format, including the number of data bits, number of stop bits, and parity type. This register is cleared upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | UnPEN | UnPSEL | | UnXB9 | UnSTP | UnCHAR | |

UnCHAR Character Frame Format. This 2-bit field selects the number of data bits per frame, not including the parity bit, as follows:

00 = eight data bits per frame
01 = seven data bits per frame
10 = nine data bits per frame
11 = loopback mode; nine data bits per frame

UnSTP Number of Stop Bits. This bit sets the number of stop bits transmitted in each frame. If this bit is 0, one stop bit is transmitted. If this bit is 1, two stop bits are transmitted.

UnXB9 Transmit 9th Data Bit. This bit is the value of the ninth data bit, either 0 or 1, transmitted when the USART is configured to transmit nine data bits per frame. It has no effect when the USART is configured to transmit seven or eight data bits per frame.

UnPSEL Parity Select. This 2-bit field selects parity type as follows:

00 = odd parity
01 = even parity
10 = mark (0)
11 = space (1)

When the USART is configured to transmit nine data bits per frame, the parity bit is omitted and the UnPSEL field is ignored.

UnPEN Parity Enable. This bit enables (1) or disables (0) parity bit generation and parity checking. When the USART is configured to transmit nine data bits per frame, there is no parity bit and the UnPEN bit is ignored.

### 18.3.6 USART Mode Select Register (UnMDSL)

The USART Mode Select Register is a byte-wide, read/write register that selects the clock source, synchronization mode, attention mode, and line break generation. This register is cleared upon reset. When the software writes to this register, the reserved bits must be cleared to 0 for proper operation. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | UnCKS | UnBRK | UnATN | UnMOD |

UnMOD Mode of Operation. Set to 0 for asynchronous operation or 1 for synchronous operation.

UnATN Attention Mode. When set to 1, this bit selects the attention mode of operation for the USART. When cleared to 0, the attention mode is disabled. The hardware clears this bit after an address frame is received. An address frame is a 9-bit character with a 1 in the ninth bit position.

UnBRK Force Transmission Break. Setting this bit to 1 causes the TDXn pin to go low. TDXn remains low until the UnBRK bit is cleared to 0 by the software.

UnCKS Synchronous Clock Source. This bit controls the clock source when the USART operates in the synchronous mode (UnMOD=1). If the UnCKS bit is set to 1, the USART operates from an external clock provided on the CKXn pin. If the UnCKS bit is cleared to 0, the USART operates from the baud rate clock produced by the USART on the CKXn pin. This bit is ignored when the USART operates in the asynchronous mode.

### 18.3.7 USART Status Register (UnSTAT)

The USART Status Register is a byte-wide, read-only register that contains the receive and transmit status bits. This register is cleared upon reset. Any attempt by the software to write to this register is ignored. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | UnXMIP | UnRB9 | UnBKD | UnERR | UnDOE | UnFE | UnPE |

UnPE Parity Error. This bit is set to 1 when a parity error is detected within a received character. This bit is automatically cleared to 0 by the hardware when the UnSTAT register is read.

UnFE Framing Error. This bit is set to 1 when the USART fails to receive a valid stop bit at the end of a frame. This bit is automatically cleared to 0 by the hardware when the UnSTAT register is read.

UnDOE Data Overrun Error. This bit is set to 1 when a new character is received and transferred to the UnBUF register before the software has read the previous character from UnBUF. This bit is automatically cleared to 0 by the hardware when the UnSTAT register is read.

UnERR Error Status Flag. This bit is set when a parity, framing, or overrun error occurs (any time that the UnPE, UnFE, or UnDOE bit is set). It is automatically cleared to 0 by the hardware when the UnPE, UnFE, and UnDOE bits are all 0.

UnBKD Break Detect. This bit is set to 1 when a line break condition occurs. This condition is detected if RDXn remains low for at least ten bit times after a missing stop bit has been detected at the end of a frame.
The hardware automatically clears the UnBKD bit upon read of the UnSTAT register, but only if the break condition on RXDn no longer exists. If reading the UnSTAT register does not clear the UnBKD bit because the break is still actively driven on the line, the hardware clears the bit as soon as the break condition no longer exists (when RXDn returns to a high level).

UnRB9 Received 9th Data Bit. With the USART configured to operate in the 9-bit data format, this is equal to the ninth data bit of the last frame received.

UnXMIP Transmit In Progress. The hardware sets this bit to 1 when the USART is transmitting data and clears it to 0 at the end of the last frame bit.

### 18.3.8 USART Interrupt Control Register (UnICTRL)

The USART Interrupt Control Register is a byte-wide register that contains the receive and transmit interrupt status flags (read-only bits) and the interrupt enable bits (read/write bits). The register is set to 01 hex upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UnEEI | UnERI | UnETI | Reserved | | | UnRBF | UnTBE |

UnTBE Transmit Buffer Empty. This read-only bit is set to 1 by the hardware when the USART transfers data from the UnTBUF register to the transmit shift register for transmission. It is automatically cleared to 0 by the hardware on the next write to the UnTBUF register.

UnRBF Receive Buffer Full. This read-only bit is set by the hardware when the USART has received a complete data frame and has transferred the data from the receive shift register to the UnRBUF register. It is automatically cleared to 0 by the hardware when the UnRBUF register is read.

UnETI Enable Transmitter Interrupt. This read/write bit, when set to 1, enables generation of an interrupt when the hardware sets the UnTBE bit.

UnERI Enable Receiver Interrupt. This read/write bit, when set to 1, enables generation of an interrupt when the hardware sets the UnRBF bit.

UnEEI Enable Receive Error Interrupt. This read/write bit, when set to 1, enables generation of an interrupt when the hardware sets the UnERR bit in the UnSTAT register.

## 18.4 BAUD RATE CALCULATIONS

The USART baud rate is determined by the system clock frequency and the values programmed into the UnPSR and UnBAUD registers. Unless the system clock frequency is an exact multiple of the desired baud rate, there will be a small amount of error in the resulting baud rate clock.

The method of baud rate calculation depends on whether the USART is configured to operate in the asynchronous or synchronous mode.

### 18.4.1 Baud Rate in Asynchronous Mode

The equation for calculating the baud rate in asynchronous mode is:

$$BR = \frac{SYS\_CLK}{(16 \times N \times P)}$$

where BR is the baud rate, SYS_CLK is the system clock, N is the value of the baud rate divisor + 1, and P is the prescaler divide factor selected by the value in the UnPSR register.

Assuming a system clock of 5 MHz and a desired baud rate of 9600, the NxP term according to the equation above is:

$$N \times P = \frac{(5 \times 10^6)}{(16 \times 9600)} = 32.552$$

The NxP term is then divided by each Prescaler Factor from Table 19 to obtain a value closest to an integer. The factor for this example is 6.5.

$$N = \frac{32.552}{6.5} = 5.008 \quad (\mathrm{N} = 5)$$

The baud rate register is programmed with a baud rate divisor of 4 (N = baud rate divisor +1). This produces a baud clock of:

$$BR = \frac{(5 \times 10^6)}{(16 \times 5 \times 6.5)} = 9615.385$$

$$\%error = \frac{(9615.385 - 9600)}{9600} = 0.16$$

Note that the percent error is much lower than would be possible without the non-integer prescaler factor. Refer to the table below for more examples.

| System Clock | Desired Baud Rate | N | P | Actual Baud Rate | Percent Error |
|---|---|---|---|---|---|
| 4 MHz | 9600 | 2 | 13 | 9615.385 | 0.16 |
| 5 MHz | 9600 | 5 | 6.5 | 9615.385 | 0.16 |
| 10 MHz | 19200 | 5 | 6.5 | 19230.769 | 0.16 |
| 20 MHz | 19200 | 5 | 13 | 19230.769 | 0.16 |

### 18.4.2 Baud Rate in Synchronous Mode

The equation for calculating the baud rate in synchronous mode is:

$$BR = \frac{SYS\_CLK}{(2 \times N \times P)}$$

where BR is the baud rate, SYS_CLK is the system clock, N is the value of the baud rate divisor + 1, and P is the prescaler divide factor selected by the value in the UnPSR register.

Use the same procedure to determine the values of N and P as in the asynchronous mode. In this case, however, only integer prescaler values are allowed.

# 19.0 ACCESS.bus Interface

The ACCESS.bus interface module (ACB) is a two wire serial interface compatible with the ACCESS.bus physical layer. It permits easy interfacing to a wide range of low-cost memories and I/O devices, including: EEPROMs, SRAMs, timers, A/D converters, D/A converters, clock chips and peripheral drivers. It is also compatible with Intel's SMBus and Philips' $I^2C$ bus. The module can be configured as a bus master or slave, and can maintain bi-directional communications with both multiple master and slave devices.

This section presents an overview of the bus protocol, and its implementation by the module.

- — ACCESS.bus, SMBus and $I^2C$ compliant
- — ACCESS.bus master and slave
- — Supports polling and interrupt controlled operation
- — Generate a wake-up signal on detection of a Start Condition, while in power-down mode
- — Optional internal pull-up on SDA and SCL pins

## 19.1 ACB PROTOCOL OVERVIEW

The ACCESS.bus protocol uses a two-wire interface for bi-directional communications between the ICs connected to the bus. The two interface lines are the Serial Data Line (SDA), and the Serial Clock Line (SCL). These lines should be connected to a positive supply, via a pull-up resistor, and remain HIGH even when the bus is idle.

The ACCESS.bus protocol supports multiple master and slave transmitters and receivers. Each IC has a unique address and can operate as a transmitter or a receiver (though some peripherals are only receivers).

During data transactions, the master device initiates the transaction, generates the clock signal and terminates the transaction. For example, when the ACB initiates a data transaction with an attached ACCESS.bus compliant peripheral, the ACB becomes the master. When the peripheral responds and transmits data to the ACB, their master/slave (data transaction initiator and clock generator) relationship is unchanged, even though their transmitter/receiver functions are reversed.

### 19.1.1 Data Transactions

One data bit is transferred during each clock pulse. Data is sampled during the high state of the serial clock (SCL). Consequently, throughout the clock's high period, the data should remain stable (see Figure 39). Any changes on the SDA line during the high state of the SCL and in the middle of a transaction aborts the current transaction. New data should be sent during the low SCL state. This protocol permits a single data line to transfer both command/control information and data using the synchronous serial clock.

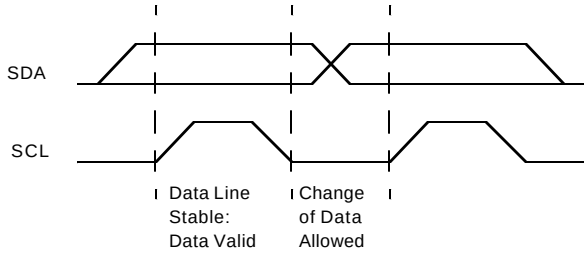



**Figure 39. Bit Transfer**

Each data transaction is composed of a Start Condition, a number of byte transfers (set by the software), and a Stop Condition to terminate the transaction. Each byte is transferred with the most significant bit first, and after each byte (8 bits), an Acknowledge signal must follow.

At each clock cycle, the slave can stall the master while it handles the previous data, or prepares new data. This can be done for each bit transferred or on a byte boundary by the slave holding SCL low to extend the clock-low period. Typically, slaves extend the first clock cycle of a transfer if a byte read has not yet been stored, or if the next byte to be transmitted is not yet ready. Some microcontrollers with limited hardware support for ACESS.bus extend the access after each bit, thus allowing the software time to handle this bit.

#### Start and Stop

The ACCESS.bus master generates Start and Stop Conditions (control codes). After a Start Condition is generated the bus is considered busy and it retains this status until a certain time after a Stop Condition is generated. A high-to-low transition of the data line (SDA) while the clock (SCL) is high indicates a Start Condition. A low-to-high transition of the SDA line while the SCL is high indicates a Stop Condition (Figure40).

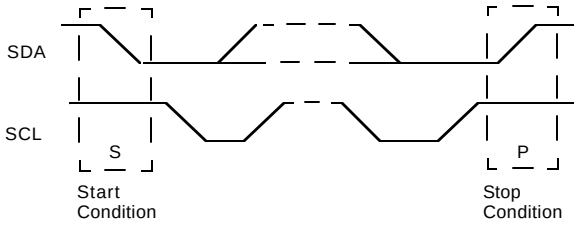



**Figure 40. Start and Stop Conditions**

In addition to the first Start Condition, a repeated Start Condition can be generated in the middle of a transaction. This allows another device to be accessed, or a change in the direction of the data transfer.

#### Acknowledge Cycle

The Acknowledge Cycle consists of two signals: the acknowledge clock pulse the master sends with each byte

transferred, and the acknowledge signal sent by the receiving device (Figure 41).

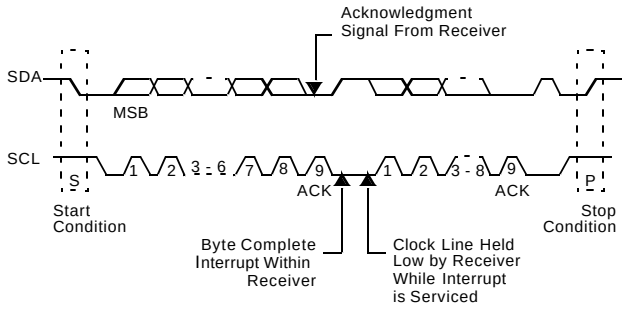



**Figure 41. ACCESS.bus Data Transaction**

The master generates the acknowledge clock pulse on the ninth clock pulse of the byte transfer. The transmitter releases the SDA line (permits it to go high) to allow the receiver to send the acknowledge signal. The receiver must pull down the SDA line during the acknowledge clock pulse, thus signalling the correct reception of the last data byte, and its readiness to receive the next byte. Figure 42 illustrates the acknowledge cycle.

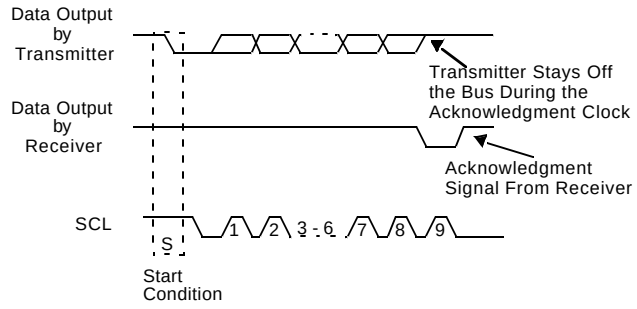



**Figure 42. ACCESS.bus Acknowledge Cycle**

The master generates an acknowledge clock pulse after each byte transfer. The receiver sends an acknowledge signal after every byte received.

There are two exceptions to the “acknowledge after every byte” rule.

1. When the master is the receiver, it must indicate to the transmitter an end of data by not-acknowledging (“negative acknowledge”) the last byte clocked out of the slave. This “negative acknowledge” still includes the acknowledge clock pulse (generated by the master), but the SDA line is not pulled down.
2. When the receiver is full, otherwise occupied, or a problem has occurred, it sends a negative acknowledge to indicate that it can not accept additional data bytes.

**Addressing Transfer Formats**

Each device on the bus has a unique address. Before any data is transmitted, the master transmits the address of the slave being addressed. The slave device should send an acknowledge signal on the SDA line, once it recognizes its address.

The address is the first seven bits after a Start Condition. The direction of the data transfer (R/$\overline{W}$) depends on the bit sent after the address — the eighth bit. A low-to-high transition during a SCL high period indicates the Stop Condition, and ends the transaction (Figure 43).

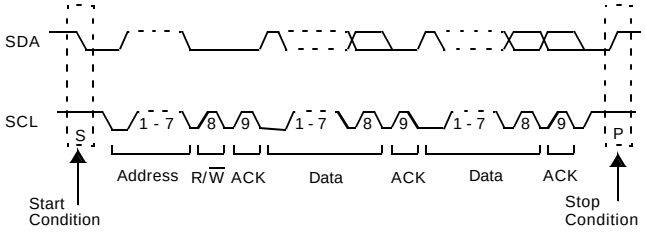



**Figure 43. A Complete ACCESS.bus Data Transaction**

When the address is sent, each device in the system compares this address with its own. If there is a match, the device considers itself addressed and sends an acknowledge signal. Depending upon the state of the R/$\overline{W}$ bit (1:read, 0:write), the device acts as a transmitter or a receiver.

The $I^2C$ bus protocol allows sending a general call address to all slaves connected to the bus. The first byte sent specifies the general call address ($00_{16}$) and the second byte specifies the meaning of the general call (for example, “Write slave address by software only”). Those slaves that require the data acknowledge the call and become slave receivers; the other slaves ignore the call.

**Arbitration on the Bus**

Multiple master devices on the bus, require arbitration between their conflicting bus-access demands. Control of the bus is initially determined according to address bits and clock cycle. If the masters are trying to address the same IC, data comparisons determine the outcome of this arbitration. In master mode, the device immediately aborts a transaction if the value sampled on the SDA lines differs from the value driven by the device. (Exceptions to this rule are SDA while receiving data; in these cases the lines may be driven low by the slave without causing an abort).

The SCL signal is monitored for clock synchronization purpose and allow the slave to stall the bus. The actual clock period will be the one set by the master with the longest clock period or by the slave stall period. The clock high period is determined by the master with the shortest clock high period.

When an abort occurs during the address transmission, the master that identify the conflict, give-up the bus and should switch to slave mode and continue to sample SDA to see if it is being addressed by the winning master on the ACCESS.bus.

## 19.2 ACB FUNCTIONAL DESCRIPTION

The ACB module provides the physical layer for an ACCESS.bus compliant serial interface. The module is configurable as either a master or slave device. As a slave device, the ACB module may issue a request to become the bus master.

### 19.2.1 Master Mode

An ACCESS.bus transaction starts with a master device requesting bus mastership. It sends a Start Condition, followed by the address of the device it wants to access. If this transaction is successfully completed, the software can assume that the device has become the bus master.

For a device to become the bus master, the software should perform the following steps:

1. Set ACBCTL1.START, and configure ACBCTL1.INTEN to the desired operation mode (Polling or Interrupt). This causes the ACB to issue a Start Condition on the ACCESS.bus, as soon as the ACCESS.bus is free (ACBCST.BB=0). It then stalls the bus by holding SCL low.
2. If a bus conflict is detected, (i.e., some other device pulls down the SCL signal before this device does), ACBST.BER is set.
3. If there is no bus conflict, ACBST.MASTER and ACBST.SDAST are set.
4. If ACBCTL1.INTEN is set, and either ACBST.BER or ACBST.SDAST is set, an interrupt is sent to the ICU.

**Sending the Address Byte**

Once this device is the active master of the ACCESS.bus (ACBST.MASTER is set), it can send the address on the bus.

The address sent should not be this device's own address as defined in ACBADDR.ADDR if ACBADDR.SAEN is set, nor should it be the global call address if ACBST.GCMTCH is set.

To send the address byte use the following sequence:

1. Configure the ACBCTL1.INTEN bit according to the desired operation mode. For a receive transaction where the software wants only one byte of data, it should set the ACBCTL1.ACK bit.
   If only an address needs to be sent, set (1) the ACBCTL1.STASTRE bit.
2. Write the address byte (7-bit target device address), and the direction bit, to the ACBSDA register. This causes the module to generate a transaction. At the end of this transaction, the acknowledge bit received is copied to ACBST.NEGACK. During the transaction the SDA and SCL lines are continuously checked for conflict with other devices. If a conflict is detected, the transaction is aborted, ACBST.BER is set, and ACBST.MASTER is cleared.
3. If ACBCTL1.STASTRE is set, and the transaction was successfully completed (i.e., both ACBST.BER and ACBST.NEGACK are cleared), ACBST.STASTR is set. In this case, the ACB stalls any further ACCESS.bus operations (i.e., holds SCL low). If ACBCTL1.INTE is set, it also sends an interrupt to the core.
4. If the requested direction is transmit, and the start transaction was completed successfully (i.e., neither ACBST.NEGACK nor ACBST.BER is set, and no other master has accessed the device), ACBST.SDAST is set to indicate that the module awaits attention.
5. If the requested direction is receive, the start transaction was completed successfully and ACBCTL1.STASTRE is cleared, the module starts receiving the first byte automatically.
6. Check that both ACBST.BER and ACBST.NEGACK are cleared. If the ACBCTL1.INTEN bit is set, an interrupt is generated when either ACBST.BER or ACBST.NEGACK is set.

**Master Transmit**

After becoming the bus master, the device can start transmitting data on the ACCESS.bus.

To transmit a byte, the software should:

1. Check that the BER and NEGACK bits in ACBST are cleared and ACBST.SDAST is set. Also, if ACBCTL1.STASTRE is set, check that ACBST.STASTR is cleared.
2. Write the data byte to be transmitted to the ACBSDA register.

When the slave responds with a negative acknowledge, the ACBST.NEGACK bit is set and the ACBST.SDAST bit remains cleared. In this case, if ACBCTL1.INTEN is set, an interrupt is sent to the core.

**Master Receive**

After becoming the bus master, the device can start receiving data on the ACCESS.bus.

To receive a byte, the software should:

1. Check that ACBST.SDAST is set and ACBST.BER is cleared. Also, if ACBCTL1.STASTRE is set, check that ACBST.STASTR is cleared.
2. Set the ACBCTL1.ACK bit to 1, if the next byte is the last byte that should be read. This causes a negative acknowledge to be sent.
3. Read the data byte from the ACBSDA register.

**Master Stop**

A Stop Condition may be issued only when this device is the active bus master (ACBST.MASTRER=1). To end a transaction, set (1) ACBCTL1.STOP before clearing the current stall flag (i.e., ACBST.SDAST, ACBST.NEGACK or ACBST.STASTR). This causes the module to send a Stop Condition immediately, and clear ACBCTL1.STOP.

**Master Bus Stall**

The ACB module can stall the ACCESS.bus between transfers while waiting for the core's response. The ACCESS.bus is stalled by holding the SCL signal low after the acknowledge cycle. Note that this is interpreted as the beginning of the following bus operation. The user must make sure that the next operation is prepared before the flag that causes the bus stall is cleared.

The flags that can cause a stall in master mode are:

— Negative acknowledge after sending a byte (ACBSTNEGACK=1).
— ACBST.SDAST bit is set.
— If ACBCTL1.STASTRE=1, after a successful start (ACBST.STASTR=1).

**Repeated Start**

A repeated start is performed when this device is already the bus master (ACBST.MASTER is set). In this case the ACCESS.bus is stalled and the ACB is awaiting the core handling due to: negative acknowledge (ACBST.NEGACK=1),

empty buffer (ACBST.SDAST=1) and/or a stop after start (ACBST.STASTR=1).

For a repeated start:

— Set the ACBCTL1.START bit.
— In master receive mode, read the last data item from ACBSDA.
— Follow the address send sequence, as described in “Sending the Address Byte” on page 80.
— If the ACB was awaiting handling due to ACBST.STASTR=1, clear it only after writing the requested address and direction to ACBSDA.

**Master Error Detections**

The ACB detects illegal Start or Stop Conditions (i.e., a Start or Stop Condition within the data transfer, or the acknowledge cycle) and a conflict on the data lines of the ACCESS.bus. If an illegal action is detected, BER is set, and the MASTER mode is exited (MASTER is cleared).

**Bus Idle Error Recovery**

When a request to become the active bus master or a restart operation fails, the ACBST.BER bit is set to indicate the error. In some cases, both this device and the other device may identify the failure and leave the bus idle. In this case, the start sequence may not be completed and the ACCESS.bus may remain deadlocked forever.

To recover from deadlock, use the following sequence:

1. Clear the ACBST.BER bit and ACBCST.BB bit.
2. Wait for a time-out period to check that there is no other active master on the bus (i.e., ACBCST.BB remains cleared).
3. Disable, and re-enable the ACB to put it in the non-addressed slave mode.
4. At this point some of the slaves may not identify the bus error. To recover, the ACB becomes the bus master by issuing a Start Condition and sends an address field; then issue a Stop Condition to synchronize all the slaves.

### 19.2.2 Slave Mode

A slave device waits in Idle mode for a master to initiate a bus transaction. Whenever the ACB is enabled, and it is not acting as a master (i.e., ACBST.MASTER is cleared), it acts as a slave device.

Once a Start Condition on the bus is detected, this device checks whether the address sent by the current master matches either:

— The ACBADDR.ADDR value if ACBADDR.SAEN is set.
— The general call address if ACBCTL1.GCM is set.

This match is checked even when ACBST.MASTER is set. If a bus conflict (on SDA or SCL) is detected, ACBST.BER is set, ACBST.MASTER is cleared and this device continues to search the received message for a match.

If an address match, or a global match, is detected:

— This device asserts its data pin during the acknowledge cycle.
— The ACBCST.MATCH and ACBST.NMATCH bits are set. If ACBST.XMIT is set (i.e., slave transmit mode), ACBST.SDAST is set to indicate that the buffer is empty.
— If ACBCTL1.INTEN is set, an interrupt is generated if both the INTEN and NMINTE bits in ACBCTL1 registers are set.
— The software then reads the ACBST.XMIT bit to identify the direction requested by the master device. It clears the ACBST.NMATCH bit so future byte transfers are identified as data bytes.

**Slave Receive and Transmit**

Slave Receive and Transmit are performed after a match is detected and the data transfer direction is identified. After a byte transfer the ACB extend the acknowledge clock until the software reads or writes the ACBSDA register. The receive and transmit sequence are identical to those used in the master routine.

**Slave Bus Stall**

When operating as a slave, this device stalls the ACCESS.bus by extending the first clock cycle of a transaction in the following cases:

— ACBST.SDAST is set.
— ACBST.NMATCH, and ACBCTL1.NMINTE are set.

**Slave Error Detections**

The ACB detects illegal Start and Stop Conditions on the ACCESS.bus (i.e., a Start or Stop Condition within the data transfer or the acknowledge cycle). When an illegal Start or Stop Condition is detected, the BER bit is set and MATCH and GMATCH are cleared, setting the module to be an unaddressed slave.

**Power Down**

When this device is in Power Save, Idle, or Halt mode, the ACB module is not active but retains its status. If the ACB is enabled (ACBCTL2.ENABLE=1) on detection of a Start Condition, a wake-up signal is issued to the MIWU module. Use this signal to switch this device to Active mode.

The ACB module cannot check the address byte following the start condition that has awaken this device for a match. The ACB responds with a negative acknowledge, and the device should re-send both the Start Condition and the address after this device has had time to wake up.

Check that the ACBCST.BUSY bit is inactive before entering Power Save, Idle or Halt mode. This guarantees that this device does not acknowledge an address sent, and stop responding later.

### 19.2.3 SDA and SCL Pins Configuration

The SDA and SCL are open-drain signals. For more information, see the I/O configuration section.

### 19.2.4 ACB Clock Frequency Configuration

The ACB module permits the user to set the clock frequency used for the ACCESS.bus clock. The clock is set by the ACBCTL2.SCLFRQ field. This field determines the SCL clock period used by this device. This clock low period may be extended by stall periods initiated by the ACB module or by another ACCESS.bus device. In case of a conflict with another bus master, a shorter clock high period may be forced by the other bus master until the conflict is resolved.

## 19.3 ACB REGISTERS

The ACCESS.bus Interface uses the following registers:

— ACB Serial Data Register (ACBSDA)
— ACB Status Register (ACBST)
— ACB Status Control Register (ACBCST)
— ACB Control 1 Register (ACBCTL1)
— ACB Control 2 Register (ACBCTL2)
— ACB Own Address Register (ACBADDR)

### 19.3.1 ACB Serial Data Register (ACBSDA)

The ACB Serial Data Register (ACBSDA) is a byte-wide, read/write shift register used to transmit and receive data. The most significant bit is transmitted (received) first and the least significant bit is transmitted (received) last. Reading or writing to the ACBSDA register is allowed when ACBST.SDAST is set; or for repeated starts after setting the START bit. An attempt to access the register in other cases produces unpredictable results.

| 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|
| DATA | | | | | | | |

### 19.3.2 ACB Status Register (ACBST)

The ACB Status Register (ACBST) is a byte-wide, read-only register that maintains current ACB status. Upon reset, and when the module is disabled, ACBST is cleared ($00_{16}$).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SLVSTP | SDAST | BER | NEGACK | STASTR | NMATCH | MASTER | XMIT |

XMIT — Direction Bit. The XMIT bit is set when the ACB module is currently in master/slave transmit mode. Otherwise it is cleared.

MASTER — MASTER. When set, the MASTER bit indicates that the module is currently in master mode. It is set when a request for bus mastership succeeds. It is cleared upon arbitration loss (BER is set) or the recognition of a Stop Condition.

NMATCH — New match. The NMATCH bit is set when the address byte following a Start Condition, or repeated starts, causes a match or a global-call match. NMATCH is cleared when 1 is written to it. Writing 0 to NMATCH is ignored. If ACBCTL1.INTEN is set, an interrupt is sent when this bit is set.

STASTR — Stall After Start. The STASTR bit is set by the successful completion of an address sending (i.e., a Start Condition sent without a bus error, or negative acknowledge) if ACBCTL1.STASTRE is set. This bit is ignored in slave mode. When STASTR is set, it stalls the ACCESS.bus by pulling down the SCL line, and suspends any other action on the bus (e.g., receives first byte in master receive mode). In addition, if ACBCTL1.INTEN is set, it also sends an interrupt to the core. Writing 1 to STASTR clears it. It is also cleared when the module is disabled. Writing 0 to STASTR has no effect.

NEGACK — Negative acknowledge. This bit is set by hardware when a transmission is not acknowledged on the ninth clock. (In this case SDAST is not set.) Writing 1 to NEGACK clears it. It is also cleared when the module is disabled. Writing 0 to NEGACK is ignored.

BER — Bus Error. BER is set by the hardware when a Start or Stop Condition is detected during data transfer (i.e., Start or Stop Condition during the transfer of bits 2 through 8 and acknowledge cycle), or when an arbitration problem is detected. Writing 1 to BER clears it. It is also cleared when the module is disabled. Writing 0 to BER is ignored.

SDAST — SDA Status. When set, this bit indicates that the SDA data register is waiting for data (transmit - master or slave) or holds data that should be read (receive - master or slave). This bit is cleared when reading from the ACBSDA register during a receive, or when written to during a transmit. When ACBCTL1.START is set, reading ACBSDA register does not clear SDAST. This enables the ACB to send a repeated start in master receive mode.

SLVSTP — Slave Stop. If set, SLVSTP indicates that a Stop Condition was detected after a slave transfer (i.e., after a slave transfer in which MATCH or GCMATCH is set). Writing 1 to SLVSTP clears it. It is also cleared when the module is disabled. Writing 0 to SLVSTP is ignored.

### 19.3.3 ACB Control Status Register (ACBCST)

ACB Control Status Register (ACBCST) is a byte-wide, read/write register that maintains current ACB status. Upon reset and when the module is disabled, the non-reserved bits of ACBCST are cleared (0).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | TGSCL | TSDA | GCMTCH | MATCH | BB | BUSY |

BUSY — BUSY. When BUSY is set, it indicates that the ACB module is:

- Generating a Start Condition
- In Master mode (ACBST.MASTER is set)
- In Slave mode (ACBCST.MATCH or ACBCST.GCMTCH is set)
- In the period between detecting a Start and completing the reception of the address byte. After this, the ACB either becomes not busy or enters slave mode.

The BUSY bit is cleared by the completion of any of the above states, and by disabling the module. BUSY is a read only bit. It should always be written with 0.

BB — Bus Busy When set, BB indicates the bus is busy. It is set when the bus is active (i.e., a low level on either SDA or SCL), or by a Start Condition. It is cleared when the module is disabled, upon detection of a Stop Condition, or when writing 1 to this bit. See “Usage Hints” on page 84 for a description of the use of this bit. This bit should be set when either SDA or SCL are low. This should be done by sampling the SDA and SCL lines continuously and, setting the bit if one of them is low. The bit remains set

until cleared by a STOP condition or a one is written to it.

MATCH — Address Match. In slave mode, MATCH is set when ACBADDR.SAEN is set and the first seven bits of the address byte (the first byte transferred after a Start Condition) matches the 7-bit address in the ACBADDR register. It is cleared by Start Condition, repeated start and Stop Condition (including illegal Start or Stop Condition).

GCMTCH — Global Call Match bit. In slave mode, GCMTCH is set when ACBCTL1.GCMEN is set and the address byte (the first byte transferred after a Start Condition) is $00_{16}$. It is cleared by Start Condition, repeated Start and Stop Condition (including illegal Start or Stop Condition).

TSDA — Test SDA Line. Reads the current value of the SDA line. This bit can be used while recovering from an error condition in which the SDA line is constantly pulled low by a slave that went out of synch. This bit is a read-only bit. Data written to it is ignored.

TGSCL — Toggle SCL Line. This bit enables toggling the SCL line during the process of error recovery. When the SDA line is low, writing 1 to this bit toggles the SCL line for one cycle. Writing 1 to TGSCL when SDA is high is ignored. The bit is cleared when the clock toggle is completed.

### 19.3.4 ACB Control 1 Register (ACBCTL1)

ACB Control 1 Register (ACBCTL1) is a byte-wide, read/write register that configures and controls the ACB module. Upon reset and while the module is disabled (ACBCTL2.ENABLE=0), the ACBCTL1 is cleared ($00_{16}$).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| STASTRE | NMINTE | GCMEN | ACK | Reserved | INTEN | STOP | START |

START — START. This bit is set when a Start Condition needs to be generated on the ACCESS.bus. The START bit is cleared when the Start Condition is sent, or upon detection of a Bus Error (ACBST.BER=1). This bit should be set only when in Master mode, or when requesting Master mode.

If this device is not the active master of the bus (ACBST.MASTER=0), setting START generates a Start Condition as soon as the ACCESS.bus is free (ACBCST.BB=0). An address send sequence should then be performed.

If this device is the active master of the bus (ACBST.MASTER=1), when START is set, a write to the ACBSDA register generates a Start Condition, then the ACBSDA data is transmitted as the slave's address and the requested transfer direction.

This case is a repeated Start Condition. It may be used to switch the direction of the data flow between the master and the slave, or to choose another slave device without using a Stop Condition in between.

STOP — STOP. In master mode, setting this bit generates a Stop Condition that completes or aborts the current message transfer. This bit clears itself after the STOP is issued.

INTEN — Interrupt Enable. When INTEN is cleared ACB interrupt is disabled. When INTEN is set, interrupts are enabled. An interrupt is generated (the interrupt signals to the ICU is high) upon one of the following events:

- An address MATCH is detected (ACBST.NMATCH=1) and NMINTE is set.
- A Bus Error occurs (ACBST.BERR=1).
- Negative acknowledge after sending a byte (ACBST.NEGACK=1).
- An interrupt is generated upon acknowledge of each transaction (same as the hardware set of the ACBST.SDAST bit).
- In master mode if ACBCTL1.STASTRE=1, after a successful start (ACBST.STASTR=1).
- Detection of a Stop Condition while in slave receive mode (ACBST.SLVSTP=1).

ACK — Acknowledge bit. When acting as a receiver (slave or master), this bit holds the value this device sends during the next acknowledge cycle. Setting this bit to 1 instructs the transmitting device to stop sending data, since the receiver either does not need, or cannot receive, any more data. This bit is cleared after the first acknowledge cycle.

This bit is ignored when in transmit mode.

GCMEN — Global Call Match enable. When this bit is set, it enables the match of an incoming address byte to the general call address (Start Condition followed by address byte of $00_{16}$) while the ACB is in slave mode. When cleared, the ACB does not respond to a global call.

NMINTE — New Match Interrupt Enable. Set NMINTE to enable the interrupt on a new match (i.e., when ACBST.NMATCH is set). The interrupt is issued only if ACBCTL1.INTEN is set.

STASTRE — Stall After Start Enable. When set enables the stall after start mechanism. In such a case, the ACB is stalled after the address byte. When STASTRE is cleared, ACBST.STASTR is always cleared.

### 19.3.5 ACB Control 2 Register (ACBCTL2)

The ACB Control 2 register (ACBCTL2) is a byte-wide, read/write register that enables/disables the module and determines ACB clock rate. Upon reset ACBCTL2 is set to $00_{16}$.

| 7 1 | 0 |
|---|---|
| SCLFRQ | ENABLE |

ENABLE — Enable. When this bit is set, the ACB module is enabled. When the Enable bit is cleared, the ACB module is disabled, ACBCTL1, ACBST and ACBCST are cleared, and the clocks are halted.

SCLFRQ — SCL Frequency. This field defines the SCL's period (low time and high time) when this de-

vice serves as a bus master. The clock low time and high time are defined as follows:

$t_{SCLl} = t_{SCLh} = 2*SCLFRQ*t_{CLK}$

Where $t_{CLK}$ is this device's clock cycle when in Active mode.
SCLFRQ may be programmed to values in the range of $0001000_2$ ($8_{10}$) through $1111111_2$ ($127_{10}$). Using any other value has unpredictable results.

### 19.3.6 ACB Own Address Register (ACBADDR)

ACB Own Address Register (ACBADDR) is a byte-wide, read/write register that holds the module's ACCESS.bus address. Reset value is undefined.

| 7 | 6 0 |
|---|---|
| SAEN | ADDR |

ADDR — Own Address. Holds the 7-bit ACCESS.bus address of this device. When in slave mode, the first seven bits received after a Start Condition are compared to this field (first bit received to bit-6, and the last to bit-0). If the address field matches the received data and SAEN is set, a match is declared.

SAEN — Slave Address Enable. When set SAEN indicates that the ADDR field holds a valid address and enables the match of ADDR to an incoming address byte. When cleared, the ACB does not check for an address match.

## 19.4 USAGE HINTS

1. When the ACB is disabled the ACBCST.BB bit is cleared. After enabling the ACB (ACBCTL2.ENABLE is set to 1) in systems with more then one master, the bus may be in the middle of a transaction with another device, which is not reflected by BB.

   There is a need to allow the ACB to synchronize to the bus activity status before issuing a request to become the bus master, to prevent bus errors. Thus, before issuing a request to become the bus master for the first time, the software should check that there is no activity on the bus by checking the BB bit after the bus allowed time-out period.
2. When waking up from power down, before checking ACBCST.MATCH, use ACBCST.BUSY to make sure that the address transaction is over.
3. The BB bit is intended to solve a deadlock in which two, or more, devices detect a usage conflict on the bus and both devices cease being bus masters at the same time. In this situation, the BB bits of both devices are active (because each deduces that there is another master currently performing a transaction, while in fact no device is executing a transaction), and the bus would stay locked until some device sends a ACBCTL1.STOP condition.

   The ACBCST.BB bit allows the software to monitor bus usage, so it can avoid sending a STOP signal in the middle of the transaction of some other device on the bus. This bit detects whether the bus remains unused over a certain period, while the BB bit is set.
4. In some cases the bus may get stuck with the SCL and/or SDA lines active. A possible cause to this is an erroneous Start or Stop Conditions that occur in the middle of a slave receive session.

   When the SCL line is stuck active, there is nothing that can be done, and it is the responsibility of the module that holds the bus to release it.

   In case of SDA line is stuck active, the ACB module enable the release of the bus by using the following sequence. Note that in normal cases SCL may be toggled only by the bus master. This protocol is a recovery scheme which is an exception that should be used only in the case where there is no other master on the bus. The recovery scheme is as follows:

   a. Disable and re-enable the module to set it into the not addressed slave mode.

   b Set the ACBCTL1.START bit to make an attempt to issue a Start Condition.

   c. Check if the SDA line is active (low) by reading ACBCST.TSDA bit. If it is active, issue a single SCL cycle by writing 1 to ACBCST.TGSCL bit. If the SDA line is not active, continue from step 'e'.

   d. Check if ACBST.MASTER is set, which indicates that the Start Condition was sent. If not, repeat step c and d until the SDA is released.

   e. Clear the BB bit. This enables the START bit to be executed. Continue according to "Bus Idle Error Recovery" on page 81.

# 20.0 CR16CAN Module

The CR16CAN device contains a FULL-CAN class, CAN (Controller Area Network) serial bus interface for low/high speed applications. It supports the reception and transmission of extended frames with 29-bit identifier, standard frames with 11-bit identifier, applications that require a high speed (up to 1MBit/s), and a low speed CAN interface with CAN master capability. The data transfer between CAN and the CPU is established by 15 message buffers, which can be individually configured as receive or transmit buffers. Every message buffer includes a status/control register which provides information about its current status and capabilities to configure the buffer. All message buffers are able to generate an interrupt upon the reception of a valid frame or the successful transmission of a frame. In addition, an interrupt on bus errors can be generated.

An incoming message is only accepted if the message identifier passes one of two acceptance filtering masks. The filtering mask can be configured to receive a single message ID per buffer or a group of IDs per receive buffer. One of the buffers uses a separate message filtering procedure. This provides the capability to establish a BASIC-CAN path. Remote transmission requests can be processed automatically by automatic reconfiguration to a receiver after transmission or by automated transmit scheduling upon reception. A priority decoder allows any buffer to have one of 16 transmit priorities including the highest or lowest absolute priority, totaling 240 different transmit priorities.

A decided bit time counter (16-bit wide) is provided to support real time applications. The contents of this counter is captured into the message buffer RAM upon reception or transmission. The counter can be synchronized via the CAN network. This synchronization feature allows a reset of the counter after the reception or transmission of a message in buffer 0.

The CR16CAN is a fast core bus peripheral which allows single cycle byte or word read/write access. The CPU controls the CR16CAN by modifying the various registers in the CR16CAN register block. This includes the initialization of the CAN baud rate, the CAN pin logic level, and the enable/disable of the CR16CAN. A set of diagnostic features, such as loopback, listen only and error identification, support the development with the CR16CAN module and provide a sophisticated error management tool.

The CR16CAN implements the following features:

- CAN specification 2.0B
  - — standard data and remote frames
  - — extended data and remote frames
  - — 0 - 8 bytes data length
  - — programmable bit rate up to 1 Mbit/s
- 15 message buffers, each configurable as receive or transmit buffers
  - — message buffers are 16-bit wide dual-port RAM
  - — one buffer may be used as BASIC-CAN path
- Remote Frame support
  - — automatic transmission after reception of a Remote Transmission Request (RTR)
  - — auto receive after transmission of a RTR
- Acceptance filtering
  - — two filtering capabilities: global acceptance mask & individual buffer identifiers
  - — one of the buffers uses an independent acceptance filtering procedure
- Programmable transmit priority
- Interrupt capability
  - — one interrupt vector for all message buffers (receive/transmit/error)
  - — each interrupt source can be enabled/disabled
- 16-bit counter with time stamp capability on successful reception or transmission of a message
- Power Save capabilities with programmable Wake-Up over the CAN bus (alternate source for the Multi-Input Wake-Up module)
- Push-Pull capability of the input/output pins
- Diagnostic functions
  - — error identification
  - — loopback and listen-only features for test and initialization purposes

## 20.1 FUNCTIONAL DESCRIPTION

As shown in Figure44, the CR16CAN module is separated into three blocks: the CAN core, the interface management and a dual ported RAM containing the message buffers.

There are two dedicated device pins for the CR16CAN interface, CANTX as the transmit output and CANRX as the receive input.

The CAN Core implements the basic CAN protocol features such as bit-stuffing, CRC calculation/checking and error management. It controls the transceiver logic and creates error signals according to the bus rules. In addition, it converts the data stream from the CPU (parallel data) to the serial CAN bus data.

The Interface Management is divided into the register block and the interface management processor. The register block provides the CAN Interface with control information from the CPU and in turn provides the CPU with status information from the CAN module. Additionally it generates the interrupt to the CPU.

The interface management processor is a state machine executing the CPU's transmission and reception commands and controlling the data transfer between several message buffers and RX/TX shift registers.

Fifteen Message Buffers are memory mapped into RAM to transmit/receive data via the CAN bus. Eight 16-bit registers belong to each buffer. One of the registers contains control and status information about the message buffer configuration and the current state of the buffer. The other registers are used for the message identifier, a maximum of up to eight data bytes and the time stamp information. During the receive process the incoming message will be stored at first in a hidden receive buffer until the message is valid. Then the buffer contents will be copied into the first message buffer which accepts the ID of the received message.

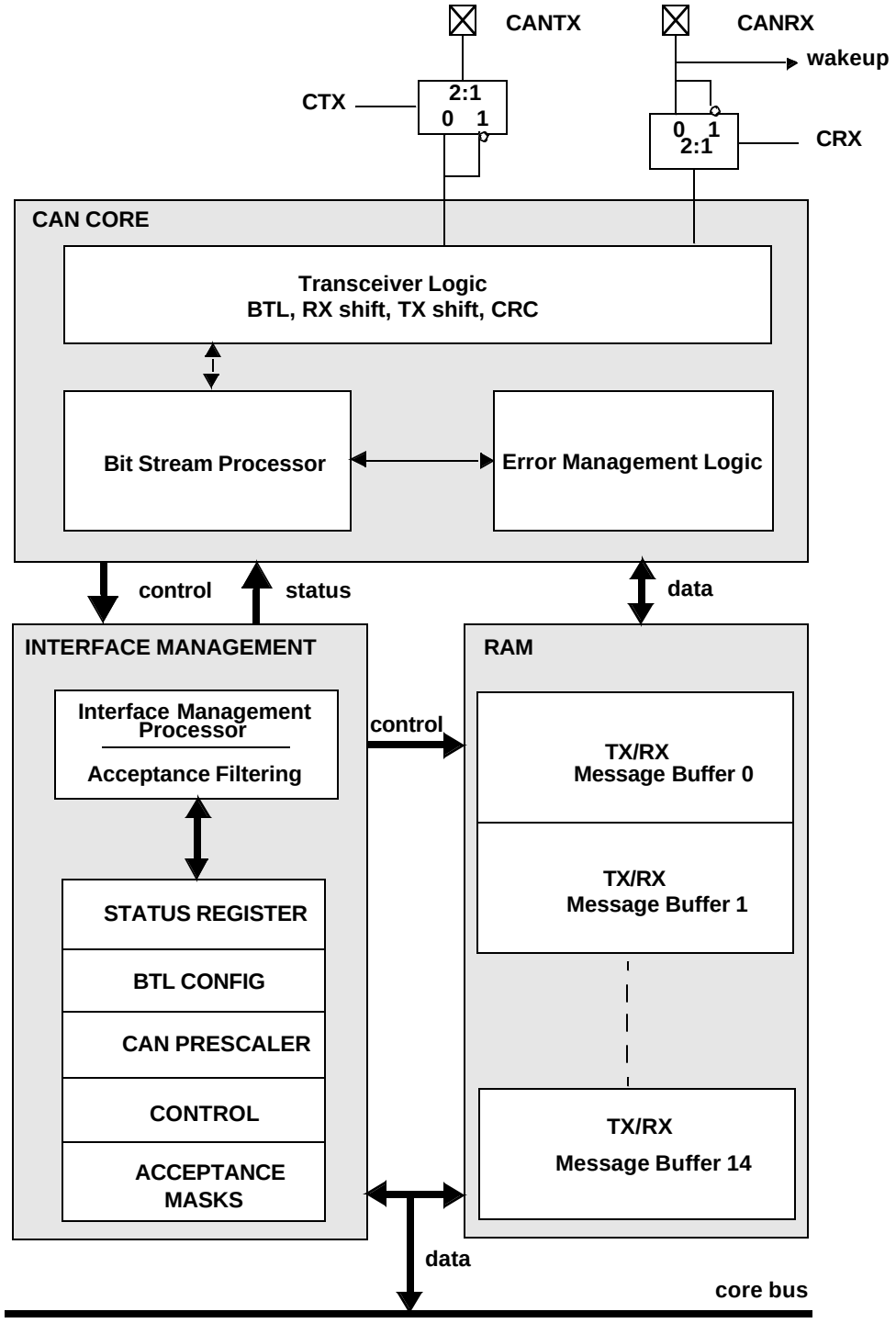



**Figure 44. Block Diagram CR16CAN Interface**

## 20.2 BASIC CAN CONCEPTS

This section provides a generic overview of the basic concepts of the Controller Area Network (CAN).

The CAN protocol is a message based protocol that allows a total of 2032 ( = $2^{11}$-16) different messages in the standard format and 512 million ( = $2^{29}$-16) different messages in the extended frame format.

Every CAN Frame is broadcasted on the common bus. Each module receives every frame and filters out the frames which are not required for the module's task. For example, if a dashboard sends a request to switch on headlights, the CAN module responsible for brake lights must not process this message.

A CAN master module has the ability to set a specific bit called the “remote data request bit” (RTR) in a frame. Such a message is also called “Remote Frame”. It causes another module, either another master or a slave which accepts this remote frame, to transmit a data frame after the remote frame has been completed.

Additional modules can be added to an existing network without a configuration change. These modules can either perform completely new functions requiring new data, or process existing data to perform a new functionality.

As the CAN network is message oriented, a message can be used as a variable which is automatically updated by the controlling processor. If any module cannot process information, it can send an overload frame.

The CAN protocol allows several transmitting modules to start a transmission at the same time as soon as they monitor the bus to be idle. During the start of transmission, every node monitors the bus line to detect whether its message is overwritten by a message with a higher priority. As soon as a transmitting module detects another module with a higher priority accessing the bus, it stops transmitting its own frame and switches to receive mode. For illustration, see Figure45.

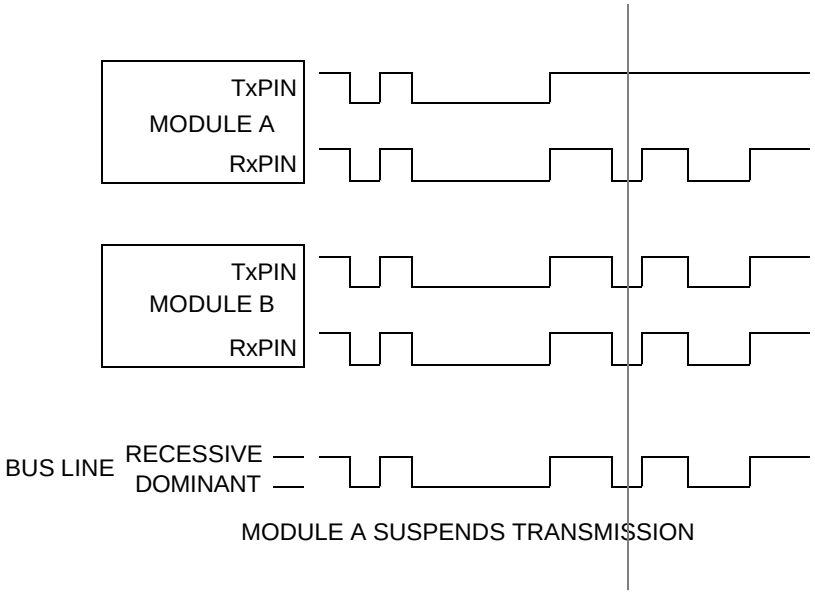



**Figure 45. CAN message arbitration**

If a data or remote frame loses arbitration on the bus due to a higher-prioritized data or remote frame, or if it is destroyed by an error frame, the transmitting module will automatically retransmit it until the transmission was successful or the user has canceled the transmit request.

If a transmitted message loses arbitration, the CR16CAN will restart transmission at the next possible time with the message which has the highest internal transmit priority.

### 20.2.1 CAN Frame Formats

Communication via the CAN bus is basically established by means of four different frame types:

— data frame
— remote frame
— error frame
— overload frame

Data and remote frames can be used in both standard and extended frame format. If no message is being transmitted, i.e., the bus is idle, the bus is kept at the ‘recessive’ level.

Remote and data frames are non-return to zero (NRZ) coded with bit-stuffing in every bit field, which holds computable information for the interface, i.e., start of frame, arbitration field, control field, data field (if present) and CRC field.

Error and overload frames are also NRZ coded but without bit-stuffing.

After five consecutive bits of the same value (including inserted stuff bits so that the stuffed bit stream will not have more than five consecutive bits of the same value), a stuff bit of the inverted value is inserted into the bit stream by the transmitter and deleted by the receiver. The following shows the stuffed and destuffed bit stream for consecutive ones and zeros.

| | | |
|---|---|---|
| original or destuffed bit stream | 10000011111x [a] | 01111100000x |
| stuffed bit stream | 100000**1**1111**0**1x | 011111**0**0000**1**0x |

a. x = {0,1}

**Frame Fields**

Data and remote frames consist of the following different bit fields:

— Start of Frame
— Arbitration Field
— Control Field
— Data Field
— CRC Field
— ACK Field
— EOF Field

The **Start of Frame** indicates the beginning of data and remote frames. It consists of a single 'dominant' bit. A node is only allowed to start transmission when the bus is idle. All nodes have to synchronize to the leading edge (first edge after the bus was idle) caused by SOF of the node which starts transmission first.

The **Arbitration field** consists of the identifier field and the RTR (Remote Transmission Request) bit. For extended frames there is also a SRR (Substitute Remote Request) and a IDE (ID Extension) bit inserted between ID18 and ID17 of the identifier field. The value of the RTR bit is 'dominant' in a data frame and 'recessive' in a remote frame.

The **Control field** consists of six bits. For standard frames it starts with the ID Extension bit (IDE) and a reserved bit (RB0). For extended frames the control field starts with two reserved bits (RB1, RB0). These bits are followed by the 4-bit Data Length Code (DLC).

The CR16CAN receiver accepts all possible combinations of the reserved bits (RB1, RB0). The transmitter must be configured to send only '0' bits.

The DLC indicates the number of bytes in the data field. It consists of four bits. The data field can be of length zero. The admissible number of data bytes for a data frame ranges from 0 to 8.

The **Data field** consists of the data to be transferred within a data frame. It can contain 0 to 8 bytes. A remote frame has no data field.

The **CRC field** consists of the CRC sequence followed by the CRC delimiter. The CRC sequence is derived by the transmitter from the modulo 2 division of the preceding bit fields, starting with the SOF up to the end of the data field, excluding stuff-bits, by the generator polynomial:

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$$

The remainder of this division is the CRC sequence transmitted over the bus. On the receiver side, the module divides all bit fields up to the CRC delimiter excluding stuff-bits, and checks if the result is zero. This will then be interpreted as a valid CRC. After the CRC sequence a single ‘recessive’ bit is transmitted as the CRC delimiter.

The **ACK field** is two bits long and contains the ACK slot and the ACK delimiter. The ACK slot is filled with a ‘recessive’ bit by the transmitter. This bit is overwritten with a ‘dominant’ bit by every receiver that has received a correct CRC sequence. The second bit of the ACK field is a ‘recessive’ bit called the acknowledge delimiter.

The **End of Frame field** closes a data and a remote frame. It consists of seven ‘recessive’ bits.

**Data Frame**

The structure of a standard and extended data frame is shown in Figure46.

A CAN data frame consists of the following fields as previously described:

— Start of Frame (SOF)
— Arbitration field + Extended Arbitration
— Control field
— Data field
— Cyclic Redundancy Check field (CRC)
— Acknowledgment field (ACK)
— End of Frame (EOF)

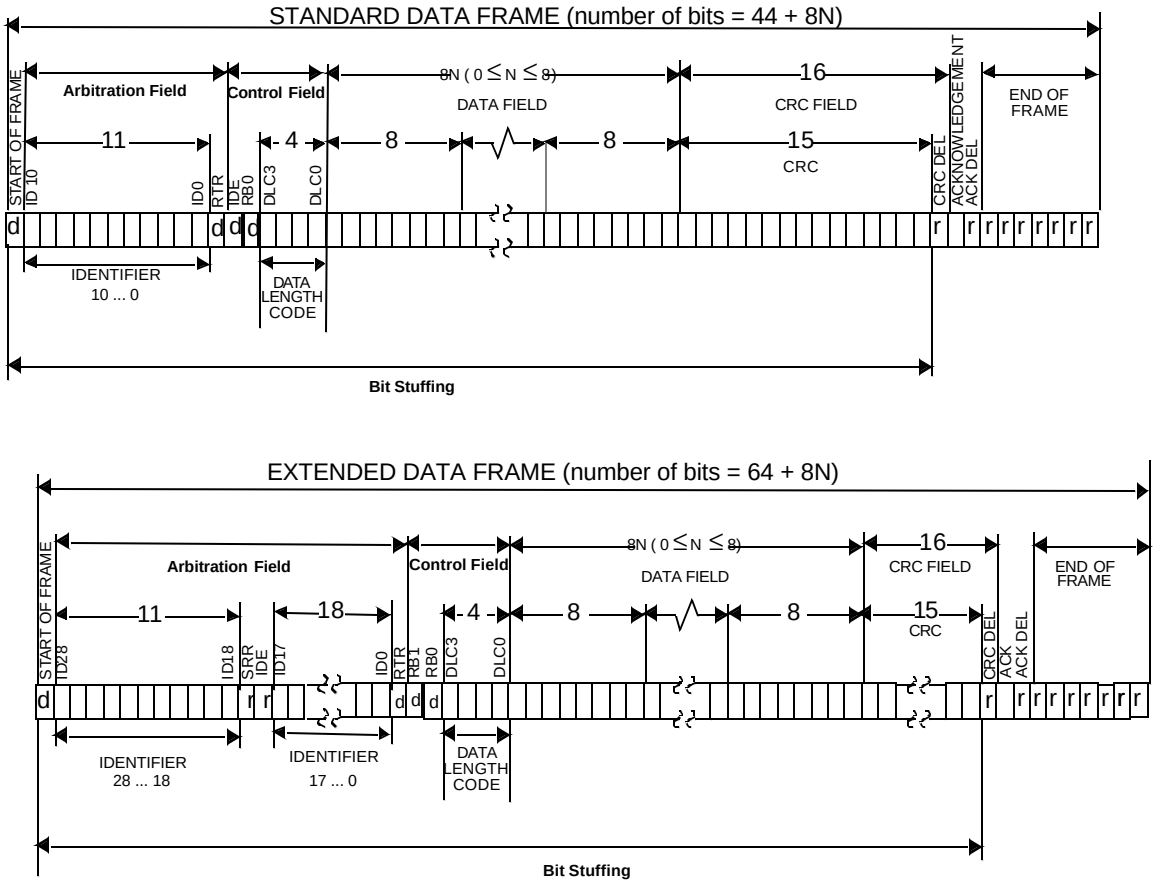
**Remote Frame**

Figure47 shows the structure of a standard and extended remote frame.

A remote frame is comprised of the following fields sections, which is the same as a data frame (see Frame Fields on page 88) except for the data field, which is not present.

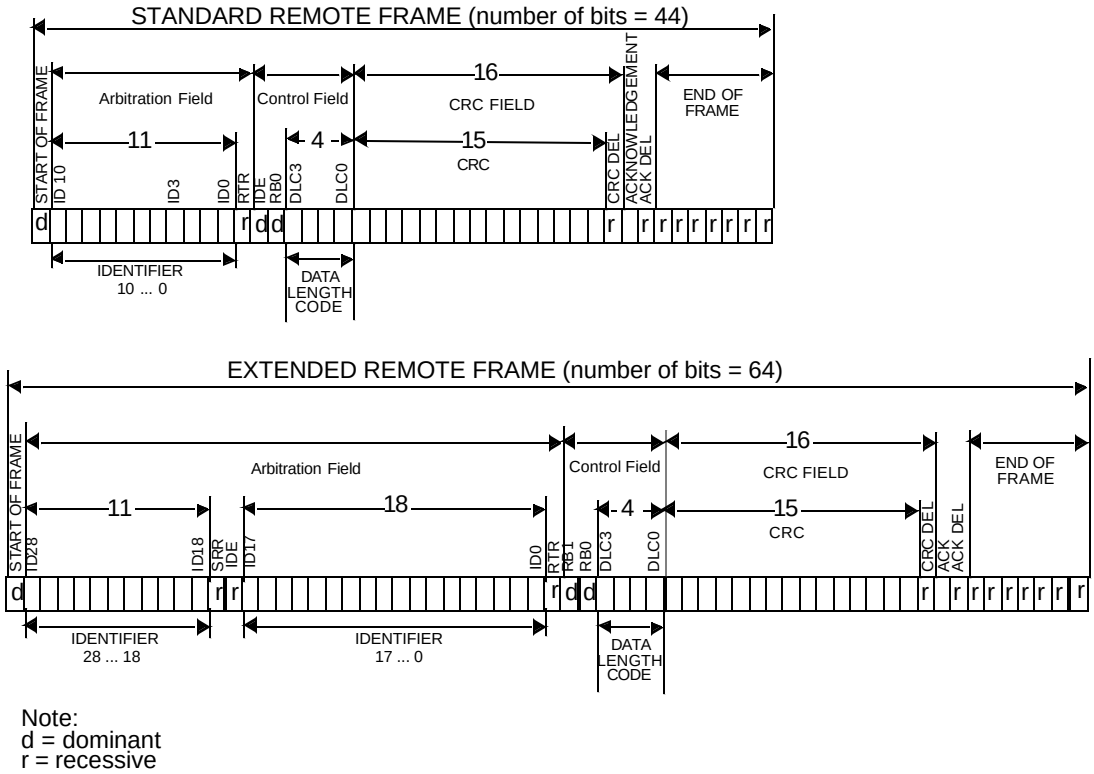
— Start of Frame (SOF)
— Arbitration field + Extended Arbitration
— Control field
— Cyclic Redundancy Check field (CRC)
— Acknowledgment field (ACK)
— End of Frame (EOF)

Note that the DLC must have the same value as the corresponding data frame to prevent contention on the bus. The RTR bit is ‘recessive’.




Note:
d = dominant
r = recessive

**Figure 46. CAN Data Frame (standard and extended)**




Note:
d = dominant
r = recessive

**Figure 47. CAN Remote Frame (standard and extended)**

**Error Frame**

As shown in Figure48, the Error Frame consists of the error flag and the error delimiter bit fields. The error flag field is built up from the various error flags of the different nodes. Therefore, its length may vary from a minimum of six bits up to a maximum of twelve bits depending on when a module has detected the error. Whenever a bit error, stuff error, form error, or acknowledgment error is detected by a node, this node starts transmission of an error flag at the next bit. If a CRC error is detected, transmission of the error flag starts at the bit following the acknowledge delimiter, unless an error flag for a previous error condition has already been started.

If a device is in the error active state, it can send a 'dominant' error flag, whereas a error passive device is only allowed to transmit 'recessive' error flags. This is done to prevent the CAN bus from getting stuck due to a local defect. For the various CAN device states, please refer to Error Detection and Management on page 92.

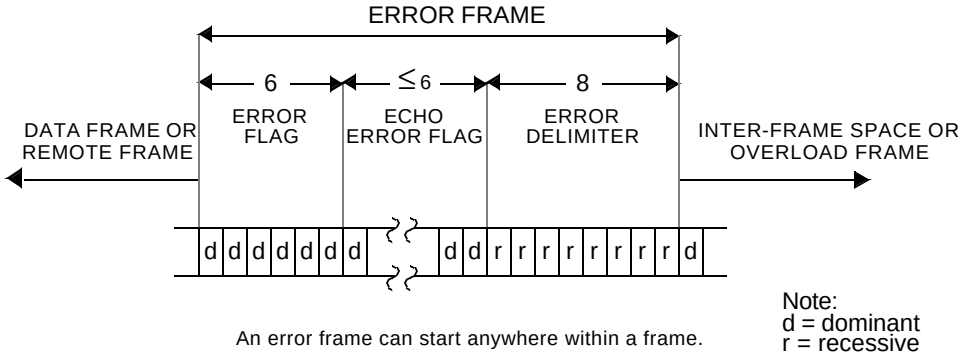



**Figure 48. CAN Error Frame**

**Overload Frame**

As shown in Figure49, an overload frame consists of the overload flag and the overload delimiter bit fields. The bit fields have the same length as the error frame field: six bits for the overload flag and eight bits for the delimiter. The overload frame can only be sent after the end of frame (EOF) field and in this way destroys the fixed form of the intermission field. As a result, all other nodes also detect an overload condition and start the transmission of an overload flag. After an overload flag has been transmitted, the overload frame is closed by the overload delimiter.

**Note:** The CR16CAN never initiates an overload frame due to its inability to process an incoming message. However, it is able to recognize and respond to overload frames initiated by other devices.

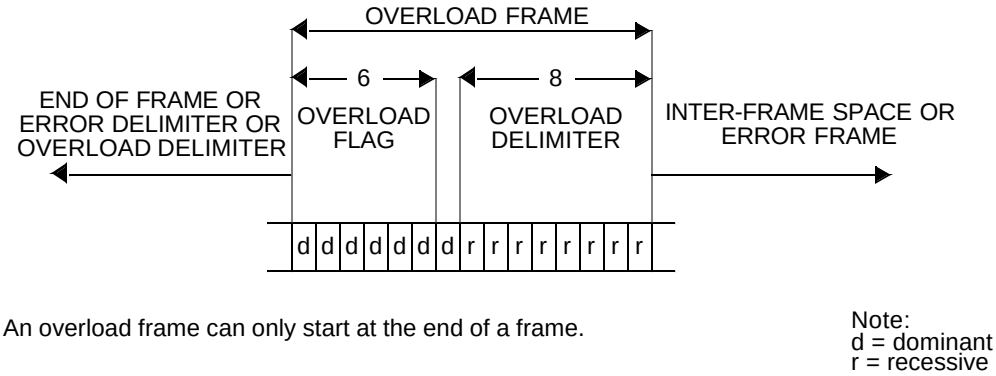



**Figure 49. CAN Overload Frame**

**Interframe Space**

Data and remote frames are separated from every preceding frame (data, remote, error and overload frames) by the interframe space (see Figure50). Error and overload frames are not preceded by an interframe space; they can be transmitted as soon as the condition occurs. The interframe space consists of a minimum of three bit fields depending on the error state of the node.

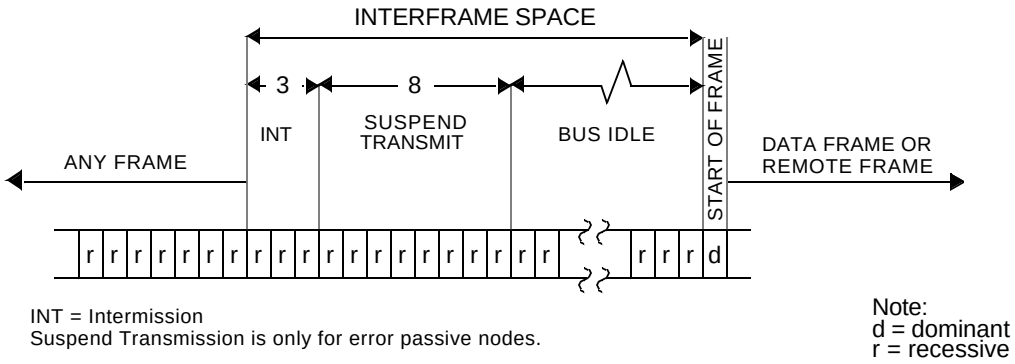



**Figure 50. CAN Interframe Space**

### 20.2.2 Error Detection and Management

There are multiple mechanisms in the CAN protocol to detect errors and inhibit erroneous modules from disabling all bus activities. Each CAN module includes two error counters, a receive and a transmit error counter, for error management.

**Error Types**

The following errors can be detected:

— Bit Error
A CAN device which is currently transmitting also monitors the bus. If the monitored bit value is different from the transmitted bit value, a bit error is detected. However, the reception of a 'dominant' bit instead of a 'recessive' bit during the transmission of a passive error flag, during the stuffed bit stream of the arbitration field or during the acknowledge slot is not interpreted as a bit error.

— Stuff Error
A stuff error is detected if the bit level after 6 consecutive bit times has not changed in a message field that has to be coded according to the bit stuffing method.

— Form Error
A form error is detected, if a fixed frame bit (e.g., CRC delimiter, ACK delimiter) does not have the specified value. For a receiver, a 'dominant' bit during the last bit of End of Frame does not constitute a frame error.

— Bit CRC Error
A CRC error is detected if the remainder of the CRC calculation of a received CRC polynomial is non-zero.

— Acknowledgment Error
An acknowledgment error is detected whenever a transmitting node does not get an acknowledgment from any other node (i.e., when the transmitter does not receive a 'dominant' bit during the ACK frame)

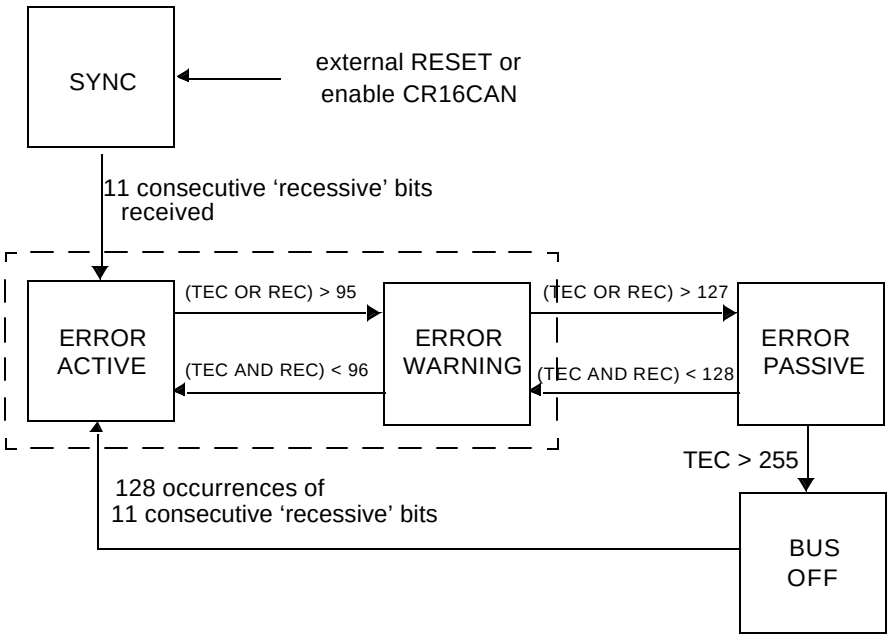



**Figure 51. CR16CAN Bus States**

— Synchronize
Once the CR16CAN is enabled, it goes into a synchronization state to synchronize with the bus by waiting for 11 consecutive recessive bits. After that the CR16CAN becomes error active and can participate in the bus communication. This state must also be entered after waking-up the device via the Multi-Input Wake-Up feature. See System Start-Up and Multi-Input Wake-Up on page 116.
— Error active
An error active unit can participate in bus communication and may send an active ('dominant') error flag.
— Error Warning
The Error Warning state is a sub-state of Error Active to indicate a heavily disturbed bus. The CR16CAN behaves as in Error Active mode. The device is reset into the Error Active mode if the value of both counters is less than 96.
— Error passive
An error passive unit can participate in bus communication. However, if the unit detects an error it is not allowed to send an active error flag. The unit sends only a passive ('recessive') error flag. A device is error passive when the transmit error counter or the receive error counter is greater than 127. A device becoming error passive will send an active error flag. An error passive device becomes error active again when both transmit and receive error counter are less than 128.
— Bus off
A unit that is bus off has the output drivers disabled, i.e., it does not participate in any bus activity. A device is bus off when the transmit error counter is greater than 255. A bus off device will become error active again after monitoring 128*11 'recessive' bits (including bus idle) on the bus. When the device goes from 'bus off' to 'error active', both error counters will have the value '0'.

**Error Counters**

The CR16CAN module contains two error counters to perform the error management. The receive error counter (REC) and the transmit error counter (TEC) are 8-bits wide, located in the 16-bit wide CANEC register. The counters are modified by the CR16CAN according to the rules listed in Table20 "Error Counter Handling".

The Error counters can be read by the users software as described under CAN Error Counter Register (CANEC) on page 115.

**Table 20 Error Counter Handling**

| Condition[a] | Action |
|---|---|
| **Receive Error Counter Conditions [b]** | |
| A receiver detects a Bit Error during sending an active error flag. | increment by 8 |
| A receiver detects a 'dominant' bit as the first bit after sending an error flag | increment by 8 |
| After detecting the 14th consecutive 'dominant' bit following an active error flag or overload flag, or after detecting the 8th consecutive 'dominant' bit following a passive error flag.<br>After each sequence of additional 8 consecutive 'dominant' bits. | increment by 8 |
| Any other error condition (stuff, frame, CRC, ACK) | increment by 1 |
| A valid reception or transmission | decrement by 1 unless counter is already 0 |
| **Transmit Error Counter Conditions** | |
| A transmitter detects a Bit Error during sending an active error flag | increment by 8 |
| After detecting the 14th consecutive 'dominant' bit following an active error flag or overload flag or after detecting the 8th consecutive 'dominant' bit following a passive error flag.<br>After each sequence of additional 8 consecutive 'dominant' bits. | increment by 8 |
| Any other error condition (stuff, frame, CRC, ACK) | increment by 8 |
| A valid reception or transmission | decrement by 1 unless counter is already 0 |

a. This table provides an overview of the CAN error conditions and the behavior of the CR16CAN; for a detailed description of the error management and fault confinement rules, please refer to the CAN Specification 2.0B
b. If the MSB (bit 7) of the REC is set, the node is error passive and the REC will not increment any further.

Special error handling for the TEC counter is performed in the following situations:

— A stuff error occurs during arbitration, when a transmitted 'recessive' stuff bit is received as a 'dominant' bit. This does not lead to an increment of the TEC.
— An ACK-error occurs in an error passive device and no 'dominant' bits are detected while sending the passive error flag. This does not lead to an increment of the TEC.

— If only one device is on the bus and this device transmits a message, it will get no acknowledgment. This will be detected as an error and the message will be repeated. When the device goes 'error passive' and detects an acknowledge error, the TEC counter is not incremented. Therefore the device will not go from 'error passive' to the 'bus off' state due to such a condition.

### 20.2.3 Bit Time Logic

In the Bit Time Logic (BTL), the CAN bus speed and the Synchronization Jump Width can be configured by the user. CR16CAN divides a nominal bit time into three time segments: synchronization segment, time segment 1 (TSEG1) and time segment 2 (TSEG2). Figure52 shows the various elements of a CAN bit time.

**CAN Bit Time**

The number of time quanta in a CAN bit (CAN Bit Time) lies between 4 and 25. The sample point is positioned between TSEG1 and TSEG2 and the transmission point is positioned at the end of TSEG2.

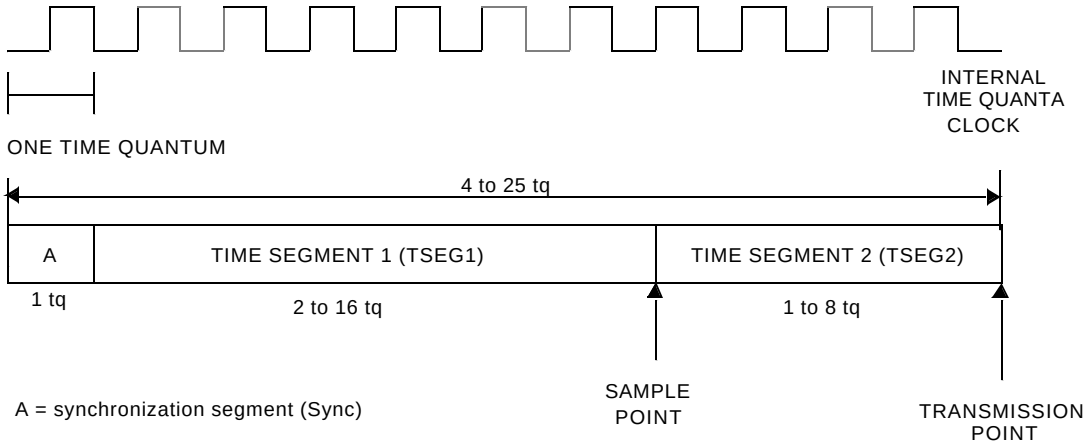



**Figure 52. Bit Timing**

The **time segment 1** includes the propagation segment and the phase segment 1 as specified in the CAN specification 2.0.B. The length of the time segment 1 in time quantas (tq) is defined by the TSEG1[3:0] bits.

The **time segment 2** represents the phase segment 2 as specified in the CAN specification 2.0.B. The length of the time segment 2 in time quantas (tq) is defined by the TSEG2[2:0] bits.

The **Synchronization Jump Width** (SJW) defines the maximum number of time quanta (tq) by which a received CAN bit can be shortened or lengthened in order to achieve resynchronization on 'recessive' to 'dominant' data transitions on the bus. In the CR16CAN implementation the SJW has to be configured less or equal to TSEG1 or TSEG2, whatever is smaller.

**Synchronization**

A CAN device expects the transition of the data signal to be within the synchronization segment of each CAN bit time. This segment has the fixed length of one time quantum.

However, two CAN nodes never operate at exactly the same clock rate and furthermore the bus signal may deviate from the ideal waveform due to the physical conditions of the network (bus length and load). In order to compensate for the various delays within a network, the sample point can be positioned by programming the length of time segments 1 and 2 (see Figure52).

In addition to that, two types of synchronization are supported. The BTL logic compares the incoming edge of a CAN bit with the internal bit timing. The internal bit timing can be adapted by either hard or soft synchronization (re-synchronization).

**Hard synchronization** is done at the beginning of a new frame with the falling edge on the bus while the bus is idle. This is interpreted as the SOF. It restarts the internal logic.

**Soft synchronization** is used during the reception of a bit stream to lengthen or shorten the internal bit time. Depending on the phase error (e), the time segment 1 may be increased or the time segment 2 may be decreased by a specific value, the re-synchronization jump width (SJW).

The phase error is given by the deviation of the edge to the SYNC segment, measured in CAN clocks. The value of the phase error is defined as:

e = 0, if the edge occurs within the SYNC segment.
e > 0, if the edge occurs within TSEG1
e < 0, if the edge occurs within TSEG2 of the previous bit.

Re-synchronization is performed according to the following rules:

- If the magnitude of e is less or equal to the programmed value of SJW, re-synchronization will have the same effect as hard synchronization.
- If e > SJW, the time segment 1 will be lengthened by the value of the SJW (see Figure53).
- If e < -SJW, the time segment 2 will be shortened by the value SJW (see Figure 54).

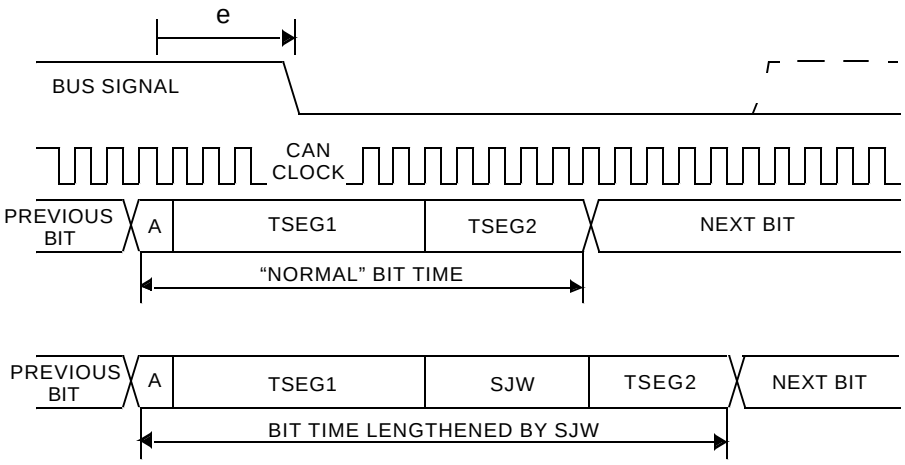



**Figure 53. Re-synchronization (e > SJW)**

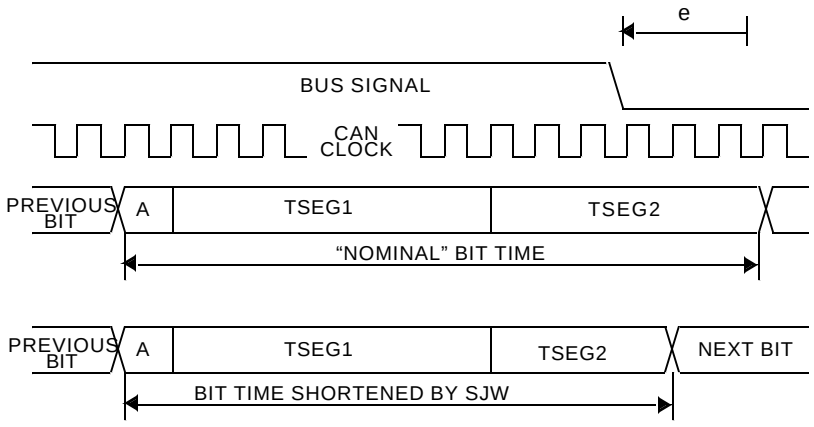



**Figure 54. Re-synchronization (e < -SJW)**

### 20.2.4 Clock Generator

The CAN prescaler (PSC) is shown is Figure55. It divides the CKI input clock by the value defined in the CTIM register. The resulting clock is called time quanta clock and defines the length of one time quanta (tq).

Please refer to CAN Timing Register (CTIM) on page 112 for a detailed description of the CTIM register.

**Note:** PSC is the value of the clock prescaler. TSEG1 and TSEG2 are the length of time segment 1 and 2 in tq.

The resulting bus clock can be calculated by the equation:

$$busclock = \frac{CKI}{(PSC)x(1 + TSEG1 + TSEG2)}$$

The values of PSC and TSEG 1 and 2 are specified by the contents of the registers PSC, TSEG1 and TSEG2 as follows:

PSC = PSC[5:0] + 2
TSEG1 = TSEG1[3:0] + 1
TSEG2 = TSEG2 [2 : 0] + 1

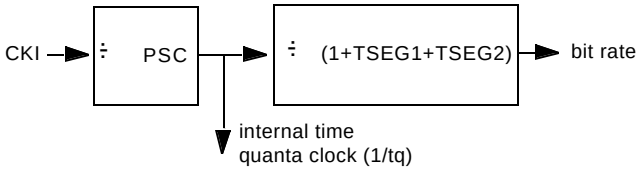



**Figure 55. Bit Rate Generation**

## 20.3 MESSAGE TRANSFER

The CR16CAN has access to 15 independent message buffers, memory mapped in RAM. Each message buffer consists of 8 different 16-bit RAM locations and can be individually configured as a receive message buffer or as a transmit message buffer.

A dedicated acceptance filtering procedure enables the user to configure each buffer to receive only a single message ID or a group of messages. One buffer uses an independent filtering procedure, which provides the possibility to establish a BASIC-CAN path.

For reception of data frame or remote frames, the CR16CAN follows a “receive on first match” rule which means that a given message is only received by one buffer — the first one which matches to the received message ID.

The transmission of a frame can be initiated by the user software writing to the transmit status and priority register. An alternate way to schedule a transmission is the automatic answer to remote frames. In the latter case, the CR16CAN will schedule every buffer for transmission to respond to remote frames with a given identifier if the acceptance mask matches. This implies that a single remote frame is able to poll multiple matching buffers configured to respond to the triggering remote transmission request.

## 20.4 ACCEPTANCE FILTERING

Two 32-bit masks are used to filter unwanted messages from the CAN bus GMASK and BMASK. Figure56 shows the mask and the buffers controlled by the masks.

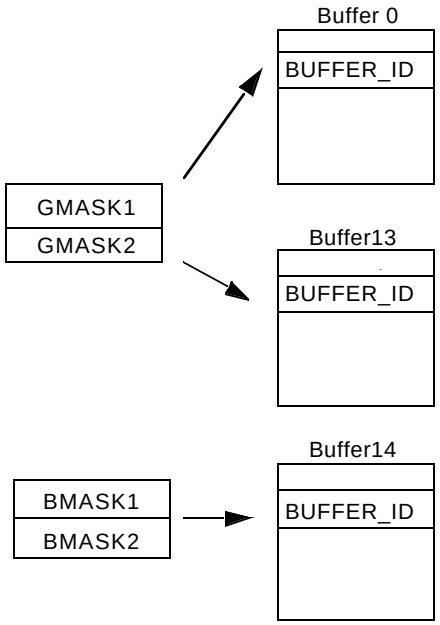



**Figure 56. Acceptance Filtering Structure**

The acceptance filtering of the incoming messages for the buffers 0...13 is done by means of a global filtering mask (GMASK) and by the buffer ID of each buffer.

The acceptance filtering of incoming messages for buffer 14 is done via a separate filtering mask (BMASK) and by the buffer ID of each that buffer.

Once a received message is waiting in the hidden buffer (see Receive Buffer Structure on page 98) to be copied into a buffer, CR16CAN scans all buffer configured as receive buffers for a matching filtering mask. The buffers 0 to 13 are checked in ascending order beginning with buffer 0. The contents of the hidden buffer are copied into the first buffer with matching filtering mask.

Bits holding a “1” in the global filtering mask (GMASK) can be represented as a “don’t care” of the associated bit of each buffer identifier, regardless of whether the buffer identifier bit is “1” or “0”.

This provides the capability to accept only a single ID per buffer or to accept a group of IDs. The following two examples illustrate the difference.

**Example 1: Acceptance of a Single Identifier**

If the global mask is set to $00_{16}$ the acceptance filtering of an incoming message is only determined by the individual buffer ID. This means that only one message ID is accepted per buffer.

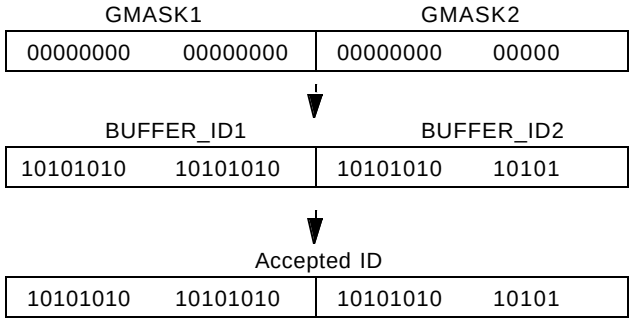



**Figure 57. Acceptance of a Single Identifier**

**Example 2: Reception of an Identifier Group**

Bits in the global mask register set to ‘1’ change the corresponding bit status within the buffer ID to “don’t care” (“X”). Therefore all messages which match the non-“don’t care” bits are accepted.

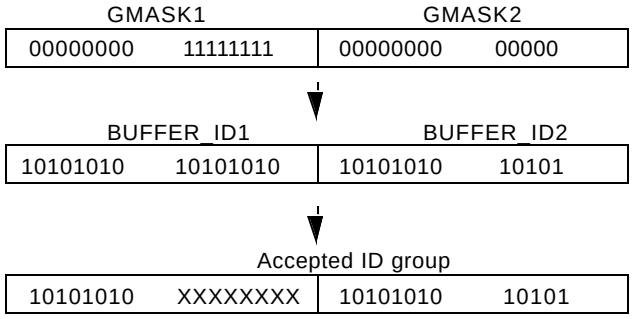



**Figure 58. Acceptance of a Group of Identifiers**

A separate filtering path is used for buffer 14. For this buffer the acceptance filtering is established by the buffer ID in conjunction with the basic filtering mask. This basic mask uses the same method as the global mask. Setting a bit to “1” changes the associated bit in the buffer ID to a “don’t care” bit.

Therefore the basic mask allows a large number of infrequent messages to be received by this buffer.

**Note:** If the BMASK register is equal to the GMASK register, the buffer 14 can be used the same way as the buffers 0 to 13.

The buffers 0 to 13 are scanned prior to buffer 14. Subsequently, the buffer 14 will not be checked for a matching ID when one of the buffers 0 to 13 has already received a message.

By setting the BUFFLOCK bit in the configuration register, the receiving buffer is automatically locked after a reception of one valid frame. The buffer will be unlocked again after the CPU has read the data and has written RX_READY in the buffer status field. With this lock function, the user has the capability to save several messages with the same identifier or same identifier group into more than one buffer. For example, a buffer with the second highest priority will receive a message if the buffer with the highest priority has already received a message and is now locked (provided that both buffers use the same acceptance filtering mask).

As shown in Figure59, several messages with the same ID are received while BUFFLOCK is enabled. The filtering mask of the buffers 0, 1, 13 and 14 is set to accept this message. The first incoming frame will be received by buffer 0. As buffer 0 is now locked the next frame will be received by buffer 1, and so on. If all matching receive buffers are full and locked, a further incoming message will not be received by any buffer.

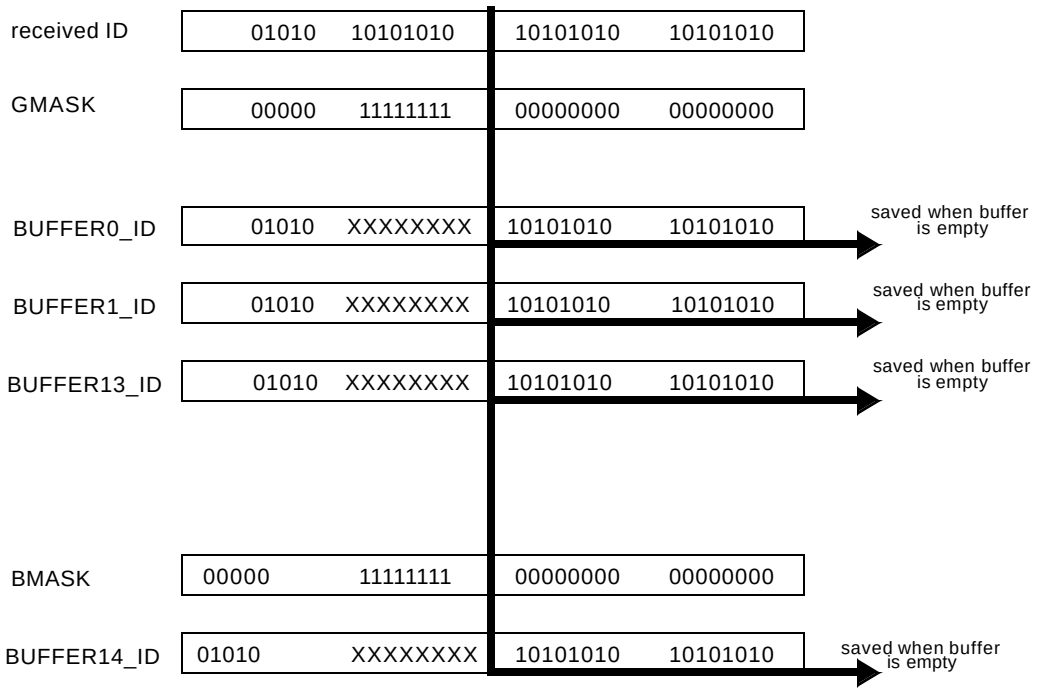



**Figure 59. Message Storage with BUFFLOCK Enabled**

## 20.5 RECEIVE STRUCTURE

All received frames will initially be buffered in a hidden receive buffer until the frame is valid. (The validation point for a received message is the penultimate bit of EOF.) The received identifier is then compared to every buffer ID together with the respective mask and the status. As soon as the validation point is reached, the whole contents of the hidden buffer is copied into the matching message buffer as shown in Figure60.

**Note:** The hidden receive buffer must not be accessed by the CPU.

The following section gives an overview of the reception of the different types of frames.

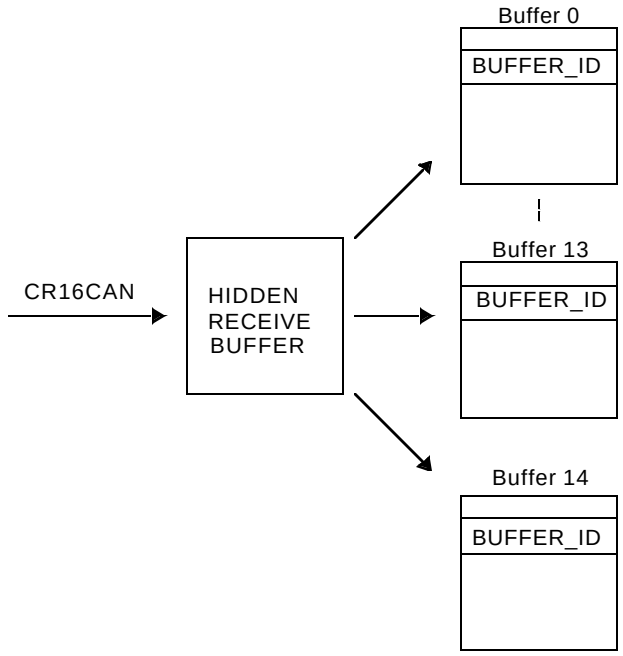



**Figure 60. Receive Buffer Structure**

The **received data frame** will be stored in the first matching receive buffer beginning with buffer 0. For example, if the message is accepted by buffer 5, then at the time the message will be copied, the RX request is cleared and CR16CAN will not try to match the frame to any subsequent buffer.

All contents of the hidden receive buffer are always copied into the respective receive buffer. This includes the received message ID as well as the received Data Length Code (DLC); therefore when some mask bits are set to don't care, the ID field will get the received message ID which could be different from the previous ID. The DLC of the receiving buffer will be updated by the DLC of the received frame. Note that the DLC of the received message is not compared with the DLC already present in the CNSTAT register of the message buffer. This implies that the DLC code of the CNSTAT register indicates how may data bytes actually belong to the latest received message.

The **remote frames** are handled by the CR16CAN interface in two different ways. Firstly, remote frames can be received like data frames by configuring the buffer to be RX_READY and setting the ID bits including the RTR bit. In that case the same procedure applies as described for Data Frames. Secondly, a remote frame can trigger one or more message buffer to transmit a data frame upon reception. This procedure is described under To answer Remote Frames on page 100.

### 20.5.1 Receive Timing

As soon as CR16CAN receives a dominant bit on the CAN bus, the receive process is started. The received ID and data will be stored in the hidden receive buffer if the global or basic acceptance filtering matches. After the reception of the data, CR16CAN tries to match the buffer ID of buffer 0...14. The data will be copied into the buffer after the reception of the 6th EOF bit as a message is valid at this time. The copy process of every frame, regardless of the length, takes at least 17 CKI cycles (see also CPU Access to CR16CAN Registers/Memory on page 105). Figure61 illustrates the receive timing.

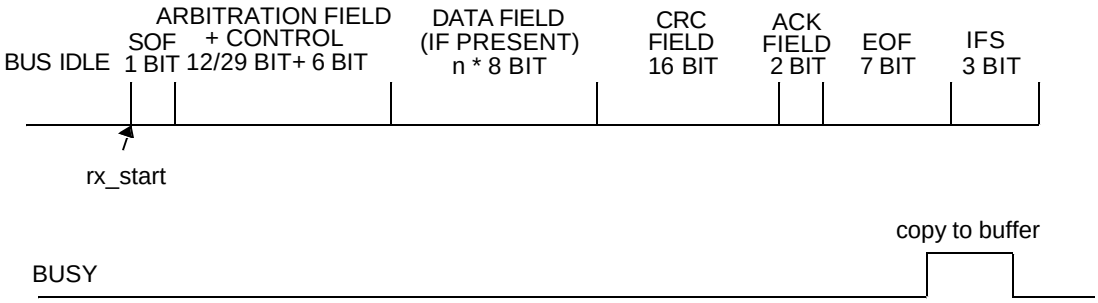



**Figure 61. Receive Timing**

In order to indicate that a frame is waiting in the hidden buffer, the BUSY bit ST[0] of the selected buffer is set during the copy procedure. The BUSY bit will be cleared by CR16CAN right after the data bytes are copied into the buffer. After the copy process is finished, CR16CAN changes the status field to RX_FULL. In turn the CPU should change the status field to RX_READY when the data is processed. When a new message has been received by the same buffer, before the CPU changed the status to RX_READY, the CR16CAN will change the status to RX_OVERRUN to indicate that at least one frame has been overwritten by a new one. Table21 summarizes the current status and the resulting update from the CR16CAN.

**Table 21 Writing to Buffer Status Code During RX_BUSY**

| Current Status | Resulting Status |
|---|---|
| RX_READY | RX_FULL |
| RX_NOT_ACTIVE | RX_NOT_ACTIVE |
| RX_FULL | RX_OVERRUN |

During the assertion of the BUSY bit, all writes to the receiving buffer are disabled with the exception of the status field.

If the status is changed during BUSY being active, the status is updated by the CR16CAN as shown in Table21.

The buffer states are indicated and controlled by the ST[3:0] bits in the CNSTAT register (see Buffer Status/Control Register (CNSTAT) on page 106. The various receive buffer states are explained in RX Buffer States on page 100.

### 20.5.2 Receive Procedure

The user has to execute the following procedure to initialize a message buffer for the reception of a CAN message.

1. Configure the receive masks (GMASK or BMASK, respectively).
2. Configure the buffer ID.
3. Configure the message buffer status as RX_READY.

In order to read the out of a received message, the CPU has to execute the following steps (see Figure62):

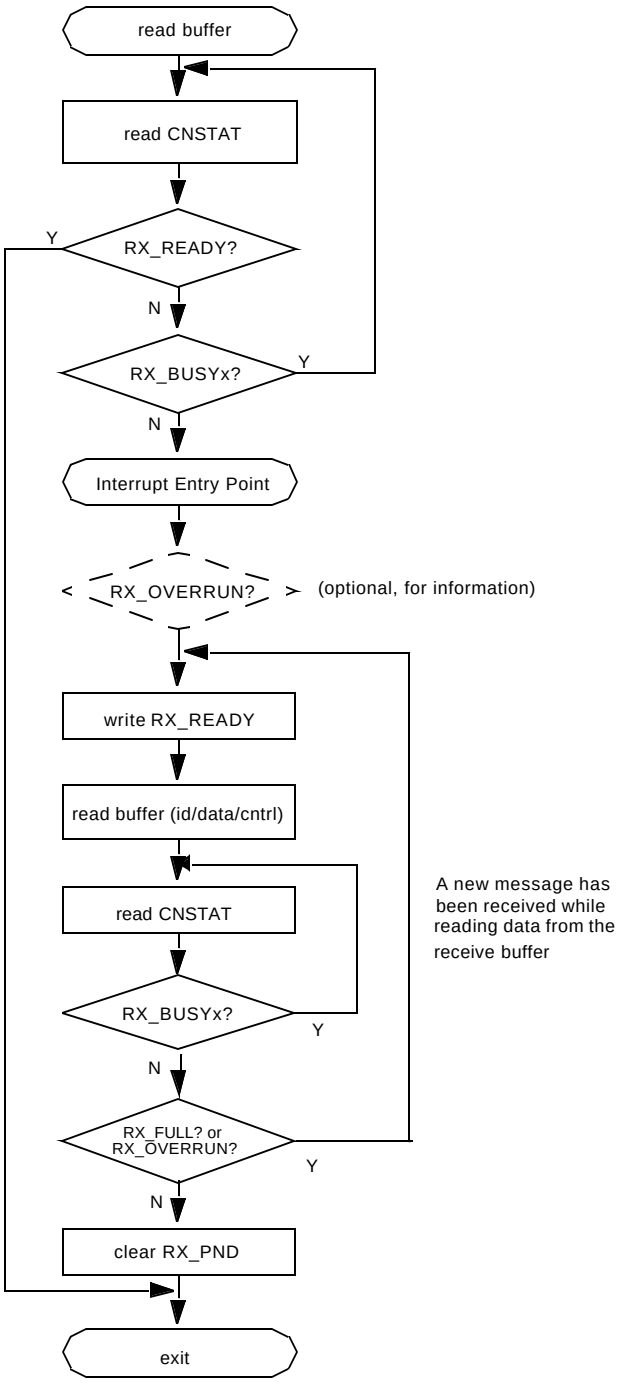



**Figure 62. Buffer Read Routine (BUFFLOCK Disabled)**

The first step is only applicable if polling is used to get the status of the receive buffer. It can be deleted for an interrupt driven receive routine.

1. Read the status (CNSTAT) of the receive buffer. If the status is RX_READY, no was the message received, exit. If the status is RX_BUSY, copy process from hidden receive buffer is not completed yet, read CNSTAT again.

   If a buffer is configured to RX_READY and its interrupt is enabled, it will generate an interrupt as soon as the

buffer has received a message and entered the RX_FULL state (see also Interrupts on page 104). In that case the procedure described below should be followed.

2. Read the status to determine if a new message has overwritten the one originally received which triggered the interrupt.
3. Write RX_READY into CNSTAT.
4. Read the ID/data and message control (DLC/RTR) from the message buffer.
5. Read the buffer status again and check it is not RX_BUSYx. If it is, repeat this step until RX_BUSYx has gone away.
6. If the buffer status is RX_FULL or RX_OVERRUN, one or more messages were copied. In that case, start over with step 2.
7. If status is still RX_READY (as set by the CPU at step 2), clear interrupt pending bit and exit.

When the BUFFLOCK function is enabled (see BUFFLOCK on page 97), it is not necessary to check for new messages received during the read process from the buffer, as this buffer is locked after the reception of the first valid frame. A read from a locked receive buffer can be performed as shown in Figure63.

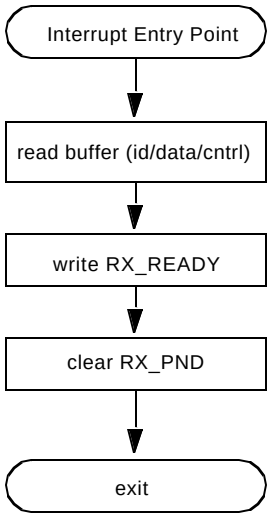



**Figure 63. Buffer Read Routine (BUFFLOCK Enabled)**

For simplicity only the applicable interrupt routine is shown:

1. Read the ID/data and message control (DLC/RTR) from the message buffer.
2. Write RX_READY into CNSTAT.
3. Clear interrupt pending bit and exit.

### 20.5.3 RX Buffer States

As shown in Figure64, a receive procedure starts as soon as the user has set the buffer from the RX_NOT_ACTIVE state into the RX_READY state. The status section of CNSTAT register is set from $0000_2$ to $0010_2$. When a message is received, the buffer will be RX_BUSYx during the copy process from the hidden receive buffer into the message buffer. Afterwards this buffer is RX_FULL. Now the CPU can read the buffer data and either reset the buffer status to RX_READY or receive a new frame before the CPU reads the buffer. In the second case, the buffer state will automatically change to RX_OVERRUN to indicate that at least one message was lost. During the copy process the buffer will again be RX_BUSYx for a short time, but in this case the CNSTAT status section will be $0101_2$, as the buffer was RX_FULL ($0100_2$) before. After finally reading the last received message, the CPU can reset the buffer to RX_READY.

## 20.6 TRANSMIT STRUCTURE

In order to transmit a CAN message, the user has to configure the message buffer by changing the buffer status to TX_NOT_ACTIVE. The buffer is configured for transmission if the ST[3] bit of the buffer status code (CNSTAT) is set to '1'. In TX_NOT_ACTIVE status, the buffer is ready to receive data from the CPU. After receiving all transmission data (ID, data bytes, DLC and PRI), the CPU can start the transmission by writing TX_ONCE into the buffer status register. During the transmission the status of the buffer is TX_BUSYx. After successful transmission CR16CAN will reset the buffer status to TX_NOT_ACTIVE. When the transmission process fails, the buffer condition will remain TX_BUSYx for re-transmission until the frame was successfully transmitted or the CPU has canceled the transmission request.

In order to **Send a Remote Frame** (Remote Transmission Request) to other CAN nodes, the user needs to set the RTR bit of the message identifier to "1" (see Storage of Remote Messages on page 109) and change the status of the message buffer to TX_ONCE. After this remote frame has been transmitted successfully, this message buffer will automatically enter the RX_READY state and is ready to receive the appropriate answer. Note that the mask bits RTR/XRTR need to be set to receive a data frame (RTR = 0) in a buffer which was configured to transmit a remote frame (RTR = 1).

**To answer Remote Frames** if the CPU writes TX_RTR in the buffer status register, the buffer will wait for a remote frame. When a remote frame passes the acceptance filtering mask of one or more buffers, the buffer status will change to TX_ONCE_RTR, the contents of the buffer will be transmitted and afterwards CR16CAN will write TX_RTR in the status code register again.

If the CPU writes TX_ONCE_RTR in the buffer status, the contents of the buffer will be transmitted, and the successful transmission the buffer goes into the "wait for Remote Frame" condition TX_RTR.

### 20.6.1 Transmit Scheduling

After writing TX_ONCE in the buffer status, the transmission process begins and the BUSY-bit is set. As soon as a buffer gets the TX_BUSY status, the buffer is not accessible anymore by the CPU except for the ST[3:1] bits of the CNSTAT register. Starting with the beginning of the CRC field of the current frame, CR16CAN looks for another buffer transmit request and selects the buffer with the highest priority for the next transmission by changing the buffer state from TX_ONCE to TX_BUSY. This transmit request can be canceled by the CPU or can be overwritten by another transmit request of a buffer with a higher priority as long as the transmission of the next frame has not yet started. This means that between the beginning of the CRC field of the current frame and the transmission start of the next frame, two buffers, the current buffer and the buffer scheduled for the next

transmission, are in the BUSY status. In order to cancel the transmit request of the next frame, the CPU has to change the buffer state to TX_NOT_ACTIVE. When the transmit request has been overwritten by another request of a higher priority buffer, CR16CAN changes the buffer state from TX_BUSY to TX_ONCE. Thus, the transmit request remains pending. Figure64 further illustrates the transmit timing.

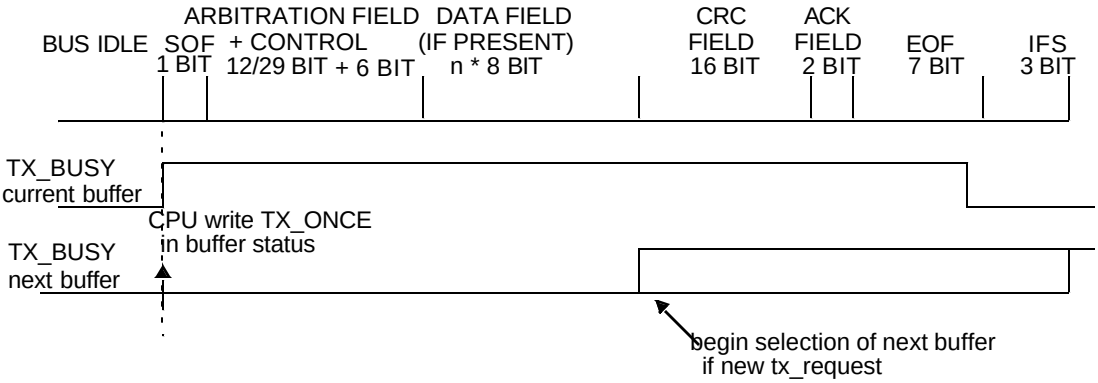



**Figure 64. Data Transmission**

If the transmit process fails or the arbitration is lost, the transmission process will be stopped and will continue after the interrupting reception or the error signaling has finished (see Figure65). In that case a new buffer select follows and the TX process is executed again.

**Note:** The canceled message can be delayed by a TX request of a buffer with a higher priority. During TX_BUSY high, the user cannot change the contents of the message buffer. In all cases writing to the BUSY bit will be ignored.

### 20.6.2 Transmit Priority

CR16CAN is able to generate a stream of scheduled messages without releasing the bus between two messages so that an optimized performance can be achieved. It will arbitrate for the bus right after sending the previous message and will only release the bus due to a lost arbitration.

If more than one buffer is scheduled for transmission, the priority is built by the message buffer number and the priority code in the CNSTAT register. The 8-bit value of the priority is formed by combining the 4-bit TXPRI value and the 4-bit buffer number (0...14) as shown below. The lowest resulting number results in the highest transmit priority.


Table22 shows the transmit priority configuration if the priority is set to TXPRI = 0 for all transmit buffers:

**Table 22 Transmit Priority (TXPRI=0)**

| TXPRI | Buffer Number | PRI | TX Priority |
|---|---|---|---|
| 0 | 0 | 0 | highest |
| 0 | 1 | 1 | |
| : | : | : | : |
| : | : | : | : |
| 0 | 14 | 14 | lowest |

Table23 shows the transmit priority configuration if TXPRI is different from the buffer number:

**Table 23 Transmit Priority (TXPRI not 0)**

| TXPRI | Buffer Number | PRI | TX Priority |
|---|---|---|---|
| 14 | 0 | 224 | lowest |
| 13 | 1 | 209 | |
| 12 | 2 | 194 | |
| 11 | 3 | 179 | |
| 10 | 4 | 164 | |
| 9 | 5 | 149 | |
| 8 | 6 | 134 | |
| 7 | 7 | 119 | |
| 6 | 8 | 104 | |
| 5 | 9 | 89 | |
| 4 | 10 | 74 | |
| 3 | 11 | 59 | |
| 2 | 12 | 44 | |
| 1 | 13 | 29 | |
| 0 | 14 | 14 | highest |

**Note:** If two buffers have the same priority (PRI), the buffer with the lower buffer number will have the higher priority.

### 20.6.3 Transmit Procedure

The transmission of a CAN message has to be executed as follows (see also Figure65)

1. Configure CNSTAT status field as TX_NOT_ACTIVE. If the status is TX_BUSY, a previous transmit request is still pending and the user has no access to the data contents of the buffer. In that case the user may choose to wait until the buffer becomes available again as shown. Other options are to exit from the update routine until the buffer has been transmitted with an interrupt generated, or the transmission is aborted by an error.
2. Load buffer identifier & data registers. (For remote frames the RTR bit of the identifier needs to be set and loading data bytes can be omitted.)
3. Configure CNSTAT status field to the desired value:
   — TX_ONCE to trigger the transmission process of a single frame.
   — TX_ONCE_RTR to trigger the transmission of a single data frame and then wait for a received remote frame to trigger consecutive data frames.
   — TX_RTR waits for a remote frame to trigger the transmission of a data frame.

Writing TX_ONCE or TX_ONCE_RTR in the CNSTAT status field will set the internal transmit request for the CR16CAN.

If a buffer is configured as TX_RTR and a remote frame is received, the data contents of the addressed buffer will be transmitted automatically without further CPU activity.

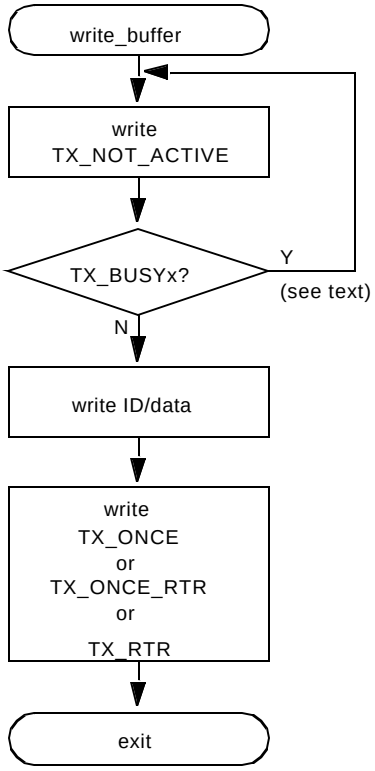



**Figure 65. Buffer Write Routine**

### 20.6.4 TX Buffer States

The transmission process can be started after the user has loaded the buffer registers (data, ID, DLC, PRI) and set the buffer status from TX_NOT_ACTIVE to TX_ONCE, TX_RTR or TX_ONCE_RTR.

When the CPU writes TX_ONCE, the buffer will be TX_BUSY as soon as CR16CAN has scheduled this buffer for the next transmission. After the frame could be successfully transmitted, the buffer status will be automatically reset to TX_NOT_ACTIVE when a data frame was transmitted or to RX_READY when a remote frame was transmitted.

If the CPU configures the message buffer to TX_ONCE_RTR, it will transmit its data contents. During the transmission the buffer state is $1111_2$ as the CPU wrote $1110_2$ into the status section of the CNSTAT register. After the successful transmission the buffer enters the TX_RTR state and waits for a remote frame. When it receives a remote frame, it will go back into the TX_ONCE_RTR state, transmit its data bytes and return to TX_RTR. If the CPU writes $1010_2$ into the buffer status section, it will only enter the TX_RTR state. But it will not send its data bytes before it waits for a remote frame. Figure 66 illustrates the possible transmit buffer states.

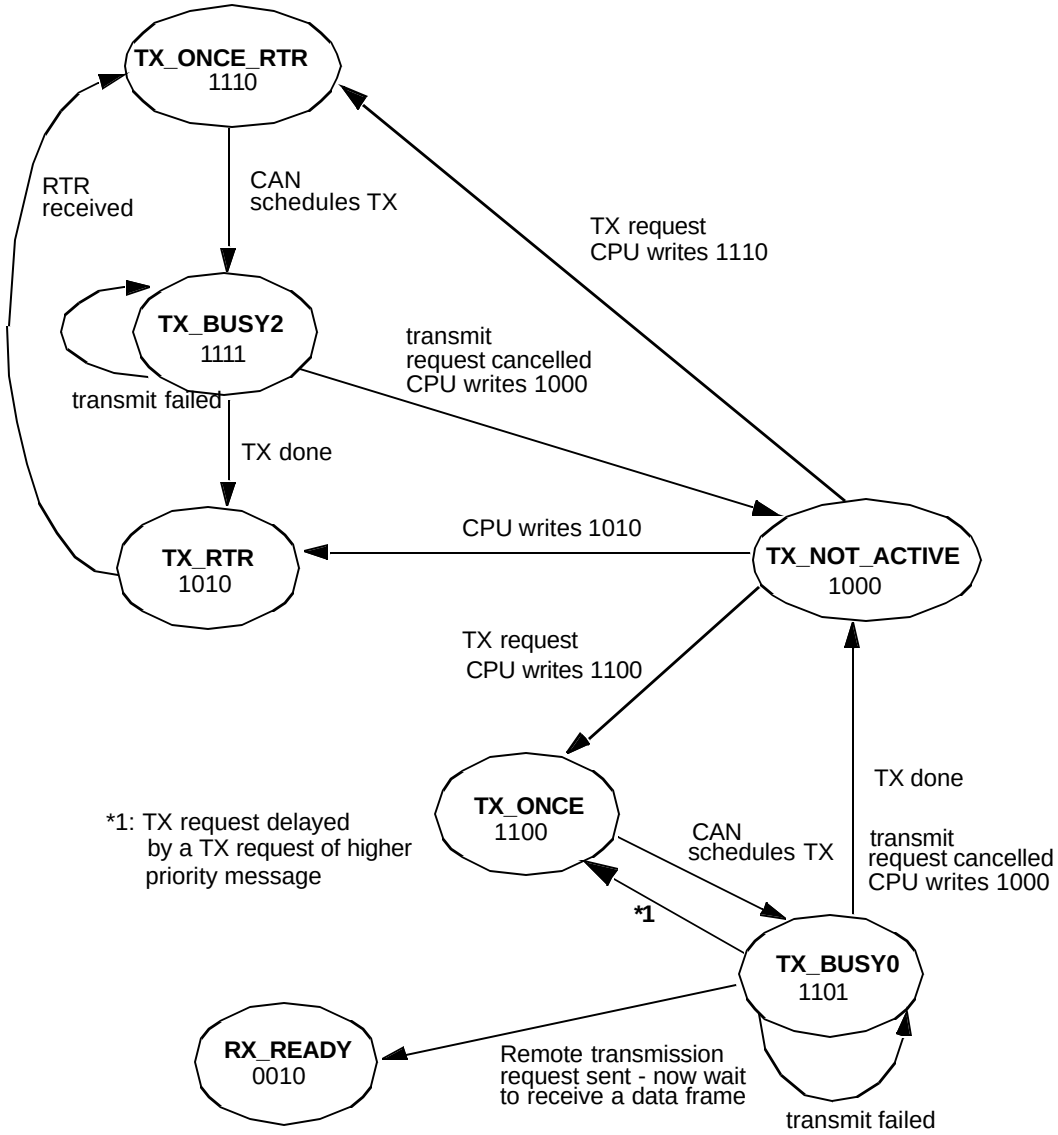



**Figure 66. Transmit Buffer States**

## 20.7 INTERRUPTS

CR16CAN has access to one interrupt vector in the CR16 CPU. The interrupt process can be initiated from the following sources.

- CAN data transfer
  - — Reception of a valid data frame in the buffer. (Buffer state changes from RX_READY to RX_FULL or RX_OVERRUN).
  - — Successful transmission of a data frame. (Buffer state changes from TX_ONCE to TX_NOT_ACTIVE or RX_READY)
  - — Successful response to a remote frame. (Buffer state changes from TX_ONCE_RTR to TX_RTR).
  - — Transmit scheduling. (Buffer state changes from TX_RTR to TX_ONCE_RTR).
- CAN error conditions is the detection of an CAN error. (The CEIPND bit in the CIPND register will be set as well as the corresponding bits in the error diagnostic register CEDIAG).

The receive/transmit interrupt access to every message buffer can be individually enabled/disabled in the CIEN register. The pending flags of the message buffer are located in the CIPND register (read only) and can be cleared by resetting the flags in the CICLR registers.

### 20.7.1 Highest Priority Interrupt Code

In order to reduce decoding time of the CIPND register, the buffer interrupt request with the highest priority is placed as interrupt status code into the IST[3:0] section of the CSTPND register.

Each of the buffer interrupts as well as the error interrupt can be individually enabled or disabled in the CAN Interrupt Enable register (CIEN). As soon as an interrupt condition occurs, every interrupt request is indicated by a flag in the CAN Interrupt Pending register (CIPND). When the interrupt code logic for the present highest priority interrupt request is enabled, this interrupt will be translated into the IST[3:0] bits of the CAN Status Pending register (CSTPND). An interrupt request can be cleared by setting the corresponding bit in the CAN Interrupt Clear register (CICLR) to ‘1’.

Figure67 illustrates the CR16CAN interrupt management.

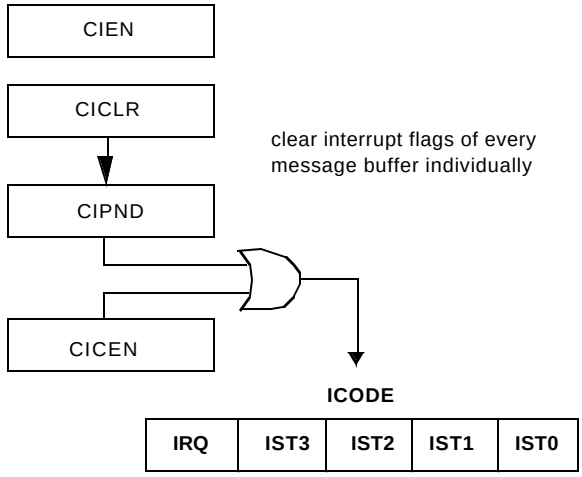



**Figure 67. CR16CAN Interrupt Management**

The highest priority interrupt source is translated into the bits IRQ and IST[3:0] as shown in Table24.

**Table 24 Highest Priority Interrupt Code (ICEN=FFFF)**

| CAN interrupt request | IRQ | IST3 | IST2 | IST1 | IST0 |
|---|---|---|---|---|---|
| no request | 0 | 0 | 0 | 0 | 0 |
| Error interrupt | 1 | 0 | 0 | 0 | 0 |
| Buffer 0 | 1 | 0 | 0 | 0 | 1 |
| Buffer 1 | 1 | 0 | 0 | 1 | 0 |
| Buffer 2 | 1 | 0 | 0 | 1 | 1 |
| Buffer 3 | 1 | 0 | 1 | 0 | 0 |
| Buffer 4 | 1 | 0 | 1 | 0 | 1 |
| Buffer 5 | 1 | 0 | 1 | 1 | 0 |
| Buffer 6 | 1 | 0 | 1 | 1 | 1 |
| Buffer 7 | 1 | 1 | 0 | 0 | 0 |
| Buffer 8 | 1 | 1 | 0 | 0 | 1 |
| Buffer 9 | 1 | 1 | 0 | 1 | 0 |
| Buffer 10 | 1 | 1 | 0 | 1 | 1 |
| Buffer 11 | 1 | 1 | 1 | 0 | 0 |
| Buffer 12 | 1 | 1 | 1 | 0 | 1 |
| Buffer 13 | 1 | 1 | 1 | 1 | 0 |
| Buffer 14 | 1 | 1 | 1 | 1 | 1 |

### 20.7.2 Usage Hints

The interrupt code IST[3:0] can be used within the interrupt handler as a displacement in order to jump to the relevant subroutine.

The CAN Interrupt Code Enable (CICEN) register is used in the CAN interrupt handler if the user wants to service all receive buffer interrupts first followed by all transmit buffer interrupts. In this case, the user can first enable only all receive buffer interrupts to be coded, scan and service all pending interrupt requests in the order of their priority. Then, the user changes the CICEN register to disable all receive buffers, but enable all transmit buffers and service all pending transmit buffer interrupt requests according to their priorities.

## 20.8 TIME STAMP COUNTER

CR16CAN features a free running 16-bit timer (CTMR) incrementing every bit time recognized on the CAN bus. The value of this timer during the ACK slot is captured into the TSTP register of a message buffer after a successful transmission or reception of a message. Figure68 shows a simplified block diagram of the Time Stamp counter.

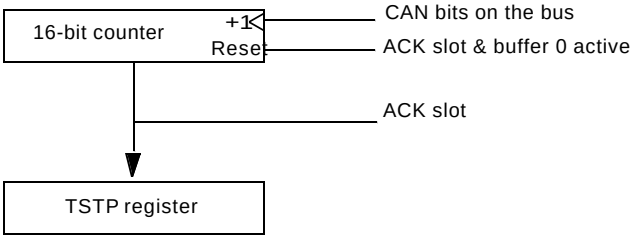



**Figure 68. Time Stamp Counter**

The timer can be synchronized over the CAN network by receiving or transmitting a message to/from buffer 0. In that case the TSTP register of buffer 0 captures the current CTMR value during the ACK slot of a message (as above) and afterwards the CTMR is reset to $0000_2$. Synchronization can be enabled or disabled via the CGCR.TSTPEN bit.

## 20.9 MEMORY ORGANIZATION

CR16CAN occupies 144 words in the memory address space. This space is separated into 15*8 + 8(reserved) words for the message buffers and 14 + 2(reserved) words for control and status.

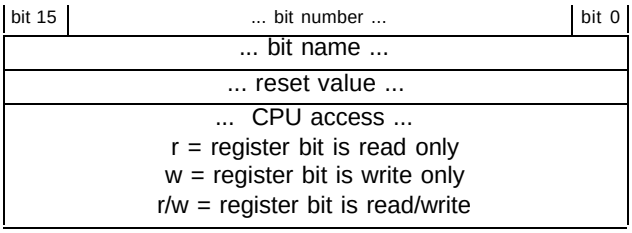
### 20.9.1 CPU Access to CR16CAN Registers/Memory

All memory locations occupied by the message buffers are shared by the CPU and CR16CAN (dual ported RAM). The CR16CAN and the CPU normally have single cycle access to this memory. However, if an access contention occurs, the access to the memory is altered every cycle until the contention is resolved. This internal access arbitration is transparent to the user.

Both word and byte access to the buffer RAM are allowed. If a buffer is busy during the reception of a message (copy process from the hidden receive buffer) or is scheduled for transmission, the CPU has no write access to the data contents of the buffer. Write to the status/control byte and read access to the whole buffer is always enabled.

All configuration and status registers can either be accessed by CR16CAN or the CPU only. These registers provide single cycle word and byte access without any potential wait state.

All register descriptions within the next sections utilize the following layout:




### 20.9.2 Message Buffer Organization

The message buffers are the communication interfaces between CAN and the CPU for the transmission and the reception of CAN frames. There are 15 message buffers located at fixed addresses in the RAM location. As shown in Table25, each buffer consists of two words reserved for the identifiers, 4 words reserved for up to eight CAN data bytes, one word is reserved for time stamp and one word for data length code, transmit priority code and the buffer status code.

**Table 25 Message Buffer Organization**

| ADDR | BUFFER register | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $xxxE_{16}$ | ID1 | XI28 ID10 | XI27 ID9 | XI26 ID8 | XI25 ID7 | XI24 ID6 | XI23 ID5 | XI22 ID4 | XI21 ID3 | XI20 ID2 | XI19 ID1 | XI18 ID0 | SRR RTR | IDE | XI17 | XI16 | XI15 |
| $xxxC_{16}$ | ID0 | XI14 | XI13 | XI12 | XI11 | XI10 | XI9 | XI8 | XI7 | XI6 | XI5 | XI4 | XI3 | XI2 | XI1 | XI0 | RTR |
| $xxxA_{16}$ | DATA0 | Data 1.7 | Data 1.6 | Data 1.5 | Data 1.4 | Data 1.3 | Data 1.2 | Data 1.1 | Data 1.0 | Data 2.7 | Data 2.6 | Data 2.5 | Data 2.4 | Data 2.3 | Data 2.2 | Data 2.1 | Data 2.0 |
| $xxx8_{16}$ | DATA1 | Data 3.7 | Data 3.6 | Data 3.5 | Data 3.4 | Data 3.3 | Data 3.2 | Data 3.1 | Data 3.0 | Data 4.7 | Data 4.6 | Data 4.5 | Data 4.4 | Data 4.3 | Data 4.2 | Data 4.1 | Data 4.0 |
| $xxx6_{16}$ | DATA2 | Data 5.7 | Data 5.6 | Data 5.5 | Data 5.4 | Data 5.3 | Data 5.2 | Data 5.1 | Data 5.0 | Data 6.7 | Data 6.6 | Data 6.5 | Data 6.4 | Data 6.3 | Data 6.2 | Data 6.1 | Data 6.0 |
| $xxx4_{16}$ | DATA3 | Data 7.7 | Data 7.6 | Data 7.5 | Data 7.4 | Data 7.3 | Data 7.2 | Data 7.1 | Data 7.0 | Data 8.7 | Data 8.6 | Data 8.5 | Data 8.4 | Data 8.3 | Data 8.2 | Data 8.1 | Data 8.0 |
| $xxx2_{16}$ | TSTP | TSTP15 | TSTP14 | TSTP13 | TSTP12 | TSTP11 | TSTP10 | TSTP 9 | TSTP 8 | TSTP 7 | TSTP 6 | TSTP 5 | TSTP 4 | TSTP 3 | TSTP 2 | TSTP 1 | TSTP 0 |
| $xxx0_{16}$ | CNTSTAT | DLC3 | DLC2 | DLC1 | DLC0 | Reserved | | | | PRI3 | PRI2 | PRI1 | PRI0 | ST3 | ST2 | ST1 | ST0 |

### 20.9.3 Buffer Status/Control Register (CNSTAT)

The buffer status, the buffer priority and the data length code are controlled by manipulating the contents of the Buffer Status/Control Register (CNSTAT). CPU and CR16CAN have access to this register.

| 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|
| DLC[3:0] | Reserved | PRI[3:0] | ST[3:0] |
| 0 | | | |
| r/w | | | |

ST[3:0] Buffer Status — The CNSTAT register has a status section, which contains the status information of the buffer as shown in Table26. This section can be modified by CR16CAN.
The ST0 bits acts as a buffer busy indication. When the BUSY bit is set, any write access to the buffer is disabled with the exception of the lower byte of the CNTSTAT register. The CR16CAN sets this bit if the buffer data is currently copied from the hidden buffer or if a message is scheduled for transmission or is currently transmitting. The CR16CAN will always reset this bit on a status update.

**Table 26 Buffer Status Section of the CNSTAT Register**

| ST3 (DIR) | ST2 | ST1 | ST0 (BUSY) | Buffer Status |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | RX_NOT_ACTIVE |
| 0 | 0 | 0 | 1 | Reserved for RX_BUSY [a] |
| 0 | 0 | 1 | 0 | RX_READY |
| 0 | 0 | 1 | 1 | RX_BUSY0 [b] |
| 0 | 1 | 0 | 0 | RX_FULL |
| 0 | 1 | 0 | 1 | RX_BUSY1 [b] |
| 0 | 1 | 1 | 0 | RX_OVERRUN |
| 0 | 1 | 1 | 1 | RX_BUSY2 [b] |
| 1 | 0 | 0 | 0 | TX_NOT_ACTIVE |
| 1 | 0 | 0 | 1 | Reserved for TX_BUSY [c] |
| 1 | 1 | 0 | 0 | TX_ONCE |
| 1 | 1 | 0 | 1 | TX_BUSY0 [d] |
| 1 | 0 | 1 | 0 | TX_RTR (automatic response to a remote frame) |
| 1 | 0 | 1 | 1 | Reserved for TX_BUSY1 [e] |
| 1 | 1 | 1 | 0 | TX_ONCE_RTR (changes to TX_RTR after transmission) |
| 1 | 1 | 1 | 1 | TX_BUSY2 [d] |

a. This condition indicates that the user wrote RX_NOT_ACTIVE to a buffer when the data copy process is still active.
b. RX_BUSYx indicates that coping is in progress at three possible times
- data is copied for the first time RX_READY → RX_BUSY0
- data is copied for the second time RX_FULL → RX_BUSY1
- data is copied for the third or more time RX_OVERRUN → RX_BUSY2

c. This state indicates that the user wrote TX_NOT_ACTIVE to a transmit buffer which is scheduled for transmission or is currently transmitting.
d. TX_BUSYx indicates that a buffers is scheduled for transmission or is actively transmitting; it can be due to one of two cases:
- a message is pending for transmission or is currently transmitting
- an automated answer is pending for transmission or is currently transmitting

e. This condition does not occur

PRI[3:0] Transmit Priority Code. The PRI[3:0] bits contain the user defined transmit priority code for the message buffer.

DLC[3:0] Data Length Code. The DLC[3:0] bits determine the number of data bytes within a received/transmitted frame. For transmission, these bits need to be set according to the number of data bytes to be transmitted. For reception, these bits indicate the number of valid received data bytes available in the message buffer. Table27 shows the possible bit combinations for DLC[3:0] for data lengths from 0 to 8 bytes.

**Table 27 Data Length Coding**

| Number of data bytes | DLC3 | DLC2 | DLC1 | DLC0 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |

**Note:** The maximum number of data bytes received/transmitted is 8, even if the data length code is set to a value greater than 8. Thus, if the data length code is greater or equal to eight bytes, the bits DLC2 to DLC0 are ignored.

### 20.9.4 Storage of Standard Messages

During the processing of standard frames, the Extended-Identifier-bit (IDE) is set to “0”. The bits ID1[3:0], ID0[15:0] are “don’t care” bits. A standard frame with eight data bytes is shown in Table28.

IDE Identifier Extension. IDE is set to “0” to indicate that the message is a standard frame using 11 identifier bits. If IDE is set to “1”, the message stored in the buffer is handled as an extended frame.

RTR Remote Transmission Request. RTR is set to “1” to indicate that the message is a remote frame. For a data frame, the RTR bit is set to “0”.

ID[10:0] The ID buffer bits ID10 to ID0 are used for the 11 standard frame identifier bits.

**Table 28 Standard Frame with 8 Data Bytes**

| ADDR | BUFFER register | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $xxxE_{16}$ | ID1 | ID10 | ID9 | ID8 | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 | RTR | IDE | don’t care | | |
| $xxxC_{16}$ | ID0 | don’t care | | | | | | | | | | | | | | | |
| $xxxA_{16}$ | DATA0 | Data 1.7 | Data 1.6 | Data 1.5 | Data 1.4 | Data 1.3 | Data 1.2 | Data 1.1 | Data 1.0 | Data 2.7 | Data 2.6 | Data 2.5 | Data 2.4 | Data 2.3 | Data 2.2 | Data 2.1 | Data 2.0 |
| $xxx8_{16}$ | DATA1 | Data 3.7 | Data 3.6 | Data 3.5 | Data 3.4 | Data 3.3 | Data 3.2 | Data 3.1 | Data 3.0 | Data 4.7 | Data 4.6 | Data 4.5 | Data 4.4 | Data 4.3 | Data 4.2 | Data 4.1 | Data 4.0 |
| $xxx6_{16}$ | DATA2 | Data 5.7 | Data 5.6 | Data 5.5 | Data 5.4 | Data 5.3 | Data 5.2 | Data 5.1 | Data 5.0 | Data 6.7 | Data 6.6 | Data 6.5 | Data 6.4 | Data 6.3 | Data 6.2 | Data 6.1 | Data 6.0 |
| $xxx4_{16}$ | DATA3 | Data 7.7 | Data 7.6 | Data 7.5 | Data 7.4 | Data 7.3 | Data 7.2 | Data 7.1 | Data 7.0 | Data 8.7 | Data 8.6 | Data 8.5 | Data 8.4 | Data 8.3 | Data 8.2 | Data 8.1 | Data 8.0 |
| $xxx2_{16}$ | TSTP | TSTP 15 | TSTP 14 | TSTP 13 | TSTP 12 | TSTP 11 | TSTP 10 | TSTP 9 | TSTP 8 | TSTP 7 | TSTP 6 | TSTP 5 | TSTP 4 | TSTP 3 | TSTP 2 | TSTP 1 | TSTP 0 |
| $xxx0_{16}$ | CNTSTAT | DLC3 | DLC2 | DLC1 | DLC0 | Reserved | | | | PRI3 | PRI2 | PRI1 | PRI0 | ST3 | ST2 | ST1 | ST0 |

### 20.9.5 Storage of Messages with Less Than 8 Data Bytes

The data bytes that are not used for data transfer are “don’t cares”. If the message is transmitted, the data within these bytes will be ignored. If the message is received, the data within these bytes will be overwritten with invalid data.

### 20.9.6 Storage of Extended Messages

If the IDE bit is set to “1”, the buffer handles extended frames. The storage of the extended ID follows the descriptions in Table29. The SRR bit is at the bit position of the RTR bit for standard frame and needs to be transmitted as “1”.

**Table 29 Extended Messages with 8 Data Bytes**

| ADDR | BUFFER register | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $xxxE_{16}$ | ID1 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | SRR | IDE | ID17 | ID16 | ID15 |
| $xxxC_{16}$ | ID0 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 | RTR |
| $xxxA_{16}$ | DATA0 | Data 1.7 | Data 1.6 | Data 1.5 | Data 1.4 | Data 1.3 | Data 1.2 | Data 1.1 | Data 1.0 | Data 2.7 | Data 2.6 | Data 2.5 | Data 2.4 | Data 2.3 | Data 2.2 | Data 2.1 | Data 2.0 |
| $xxx8_{16}$ | DATA1 | Data 3.7 | Data 3.6 | Data 3.5 | Data 3.4 | Data 3.3 | Data 3.2 | Data 3.1 | Data 3.0 | Data 4.7 | Data 4.6 | Data 4.5 | Data 4.4 | Data 4.3 | Data 4.2 | Data 4.1 | Data 4.0 |
| $xxx6_{16}$ | DATA2 | Data 5.7 | Data 5.6 | Data 5.5 | Data 5.4 | Data 5.3 | Data 5.2 | Data 5.1 | Data 5.0 | Data 6.7 | Data 6.6 | Data 6.5 | Data 6.4 | Data 6.3 | Data 6.2 | Data 6.1 | Data 6.0 |
| $xxx4_{16}$ | DATA3 | Data 7.7 | Data 7.6 | Data 7.5 | Data 7.4 | Data 7.3 | Data 7.2 | Data 7.1 | Data 7.0 | Data 8.7 | Data 8.6 | Data 8.5 | Data 8.4 | Data 8.3 | Data 8.2 | Data 8.1 | Data 8.0 |
| $xxx2_{16}$ | TSTP | TSTP 15 | TSTP 14 | TSTP 13 | TSTP 12 | TSTP 11 | TSTP 10 | TSTP 9 | TSTP 8 | TSTP 7 | TSTP 6 | TSTP 5 | TSTP 4 | TSTP 3 | TSTP 2 | TSTP 1 | TSTP 0 |
| $xxx0_{16}$ | CNTSTAT | DLC3 | DLC2 | DLC1 | DLC0 | Reserved | | | | PRI3 | PRI2 | PRI1 | PRI0 | ST3 | ST2 | ST1 | ST0 |

SRR Substitute Remote Request. SRR replaces the RTR bit used in standard frames at this bit position. The SRR bit needs to be set to “1” by the user if the buffer is configured to transmit a message with an extended identifier. It will be received as monitored on the CAN bus.

IDE Identifier Extension. IDE is set to “0” to indicate that the message is a standard frame using 11 identifier bits. If IDE is set to “1”, the message stored in the buffer is handled as an extended frame.

RTR Remote Transmission Request. RTR is set to “1” to indicate that the message is a remote frame. For a data frame, the RTR bit is set to “0”.

ID[28:0] The ID bits 28 to 0 are used to build the 29-bit identifier of an extended frame.

### 20.9.7 Storage of Remote Messages

During remote frame transfer, the buffer registers DATA[3:0] are “don’t cares”. If a remote frame is transmitted, the contents of these registers are ignored. If a remote frame is received, the contents of these registers will be overwritten with invalid data. The structure of a message buffer set up for a remote frame with extended identifier is shown in Table30.

**Table 30 Extended Remote Frame**

| ADDR | BUFFER register | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $xxxE_{16}$ | ID1 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | SRR | IDE | ID17 | ID16 | ID15 |
| $xxxC_{16}$ | ID0 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 | RTR |
| $xxxA_{16}$ | DATA0 | don’t care | | | | | | | | | | | | | | | |
| $xxx8_{16}$ | DATA1 | don’t care | | | | | | | | | | | | | | | |
| $xxx6_{16}$ | DATA2 | don’t care | | | | | | | | | | | | | | | |
| $xxx4_{16}$ | DATA3 | don’t care | | | | | | | | | | | | | | | |
| $xxx2_{16}$ | TSTP | TSTP15 | TSTP14 | TSTP13 | TSTP12 | TSTP11 | TSTP10 | TSTP 9 | TSTP 8 | TSTP 7 | TSTP 6 | TSTP 5 | TSTP 4 | TSTP 3 | TSTP 2 | TSTP 1 | TSTP 0 |
| $xxx0_{16}$ | CNTSTAT | DLC3 | DLC2 | DLC1 | DLC0 | Reserved | | | | PRI3 | PRI2 | PRI1 | PRI0 | ST3 | ST2 | ST1 | ST0 |

SRR Substitute Remote Request. SRR replaces the RTR bit used in standard frames at this bit position. The SRR bit needs to be set to “1” by the user.

IDE Identifier Extension. IDE is set to “0” to indicate that the message is a standard frame using 11 identifier bits. If IDE is set to “1”, the message stored in the buffer is handled as an extended frame.

RTR Remote Transmission Request. RTR is set to “1” to indicate that the message is a remote frame. For a data frame, the RTR bit is set to “0”.

ID[28:0] The ID bits 28 to 0 are used to build the 29-bit identifier of an extended frame. The ID1 buffer bits ID28 to ID18 are used for the 11 standard frame identifier bits.

### 20.9.8 CAN Global Configuration Register (CGCR)

The CAN Global Configuration Register (CGCR) is a 16-bit wide register used to:

- enable/disable the CR16CAN
- configure the BUFFLOCK function for the message buffer 0...14
- enable/disable the time stamp synchronization
- set the logic levels of the CAN Input/Output pins CANRX/CANTX
- choose the data storage direction (DDIR)
- select the error interrupt type (EIT)
- enable/disable diagnostic functions

| 15 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|
| Reserved | EIT | DIAGEN | INTERNAL | LOOPBACK |
| 0 | | | | |
| r/w | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IGNACK | LO | DDIR | TSTPEN | BUFFLOCK | CRX | CTX | CANEN |
| 0 | | | | | | | |
| r/w | | | | | | | |

CANEN CAN Enable. This bit enables/disables the CR16CAN. When the CR16CAN is disabled, all internal states and the TEC and REC counter registers are cleared. In addition the CR16CAN clock is disabled. All CR16CAN control registers and the contents of the message memory are left unchanged.
The user needs to make sure that no message is pending for transmission before the CR16CAN is disabled.
“0” CR16CAN is disabled
“1” CR16CAN is enabled

CTX Control Transmit. This bit configures the logic level of the CAN transmit pin CANTX.
“0” dominate state is “0”; recessive state is “1”

"1" dominate state is "1"; recessive state is "0"

CRX Control Receive. This bit configures the logic level of the CAN receive pin CANRX.

"0" dominate state is "0"; recessive state is "1"

"1" dominate state is "1"; recessive state is "0"

BUFFLOC Buffer Lock. With this bit the user can configure the buffer lock function. If this feature is enabled, a buffer will be locked upon a successful frame reception. The buffer will be unlocked again by writing RX_READY in the buffer status register, i.e., after reading data.

"0" lock function is disabled for all buffers

"1" lock function is enabled for all buffers

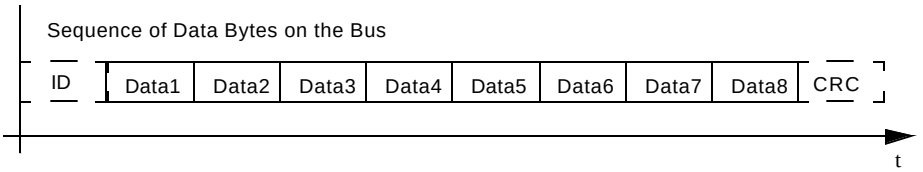
TSTPEN Time Sync Enable. The Time Sync bit enables or disables the time stamp synchronization function of the CR16CAN.

"0" Time synchronization disabled. The Time Stamp counter value is not reset upon reception or transmission of a message to/from buffer 0.

"1" Time synchronization enabled. The Time Stamp counter value is reset upon reception or transmission of a message to/from buffer 0.

DDIR Data Direction. By setting or resetting the DDIR bit, the user can select the direction the data bytes are transmitted and received. The CR16CAN transmits and receives the CAN data byte Data1 first and the data byte Data8 last (Data1, Data2,...,Data7, Data8).

If DDIR is set to "0" the data contents of a received message is stored with the first byte at the highest data address and the last data at the lowest data address (see Figure69). The same applies for transmitted data.

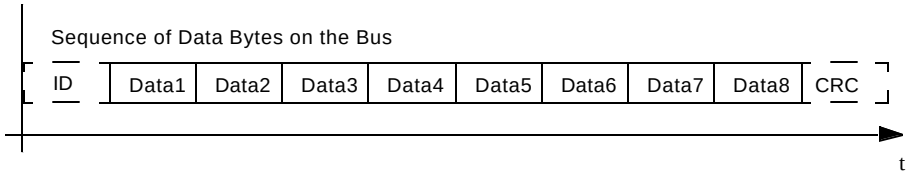



Storage of Data Bytes in the Buffer Memory

| ADDR offset | Data Bytes | |
|---|---|---|
| $0A_{16}$ | Data1 | Data2 |
| $08_{16}$ | Data3 | Data4 |
| $06_{16}$ | Data5 | Data6 |
| $04_{16}$ | Data7 | Data8 |

**Figure 69. Data Direction Bit set to '0'**

Setting the DDIR bit to “1” will cause the direction of the data storage to be reversed — the last byte received is stored at the highest address and the first byte is stored at the lowest address. See Figure70 for illustration.




Storage of Data Bytes in the Buffer Memory

| ADDRoffset | Data Bytes | |
|---|---|---|
| $0A_{16}$ | Data8 | Data7 |
| $08_{16}$ | Data6 | Data5 |
| $06_{16}$ | Data4 | Data3 |
| $04_{16}$ | Data2 | Data1 |

**Figure 70. Data Direction Bit set to ‘1’**

LO — Listen Only — By setting the LO-bit to “1” the CR16CAN interface is configured to behave only as a receiver. This means:

- it cannot transmit any message.
- it cannot send a dominant ACK bit.
- when errors are detected on the bus, the CR16CAN will behave as in the error passive mode.

Using this listen only function, the CR16CAN interface can be adjusted when it gets connected to an operating network with unknown bus speed.

IGNACK — Ignore Acknowledge. If the ignore ACK function is enabled, then by setting the IGNACK bit to “1”, CR16CAN does not expect to receive a dominant ACK bit to indicate the validity of a transmitted message. It will not send an error frame when the transmitted frame in not acknowledged by any other CAN node.

This feature can be used in conjunction with the LOOPBACK option for stand-alone tests outside of a CAN network.

LOOPBACK — Loopback. By setting the LOOPBACK bit, all messages sent by CR16CAN can also be received by a CR16CAN buffer with a matching buffer ID. However, CR16CAN does not acknowledge a message sent by itself. Therefore CR16CAN will send an error frame when no other device connected to the bus has acknowledged the message.

INTERNAL — Internal. If the INTERNAL function is enabled, the TX- and RX-pin of the CR16CAN are internally connected to each other. This feature can be used in conjunction with the LOOPBACK mode. This means that CR16CAN can receive its own sent messages without connecting an external transceiver chip to the RX- and TX-pin; it allows the user to run real stand-alone tests without any peripheral devices.

DIAGEN — Diagnostic Enable. The DIAGEN bit globally enables or disables the special diagnostic features of CR16CAN. This includes the following functions:

- LO (Listen Only)
- IGNACK (Ignore Acknowledge)
- LOOPBACK (Loopback)
- INTERNAL (Internal Loopback)
- write access to hidden receive buffer

EIT — Error Interrupt Type. This bit configures when the Error Interrupt Pending Bit (CIPND.EIPND) is set and an error interrupt is generated if enabled by the Error Interrupt Enable (CIEN.EIEN).

“0” The EIPND bit is set on every error on the CAN bus.

“1” The EIPND bit is set only if the error state (CSTPND.NS) changes as a result of incrementing either the receive or transmit error counter.

### 20.9.9 CAN Timing Register (CTIM)

The Can Timing Register (CTIM) defines the configuration of the Bit Time Logic (BTL).

| 15 9 | 8 7 | 6 3 | 2 0 |
|---|---|---|---|
| PSC[6:0] | SJW[1:0] | TSEG1[3:0] | TSEG2[2:0] |
| 0 | | | |
| r/w | | | |

PSC[6:0] Prescaler Configuration. These bits set the CAN prescaler. The settings are shown in Table31

**Table 31 CAN Prescaler Settings**

| PSC6 | PSC5 | PSC4 | PSC3 | PSC2 | PSC1 | PSC0 | CAN prescaler (PSC) |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 4 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 5 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 6 |
| | : | : | : | : | : | : | : |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 127 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0/1 | 128 |

SJW[1:0] Synchronization Jump Width. These bits set the Synchronization Jump Width which can be programmed between 1 and 4 time quanta (see Table32).

**Table 32 SJW Settings**

| SJW1 | SJW0 | Synchronization Jump Width (SJW) |
|---|---|---|
| 0 | 0 | 1 tq |
| 0 | 1 | 2 tq |
| 1 | 0 | 3 tq |
| 1 | 1 | 4 tq |

**Note:** The settings of SJW has to be configured to be smaller or equal to TSEG1 and TSEG2

TSEG1[3:0] Time Segment 1. These bits configure the length of the Time Segment 1 (TSEG1). It is not recommended to configure the time segment 1 to be smaller than 2tq. (see Table33).

**Table 33 Time Segment 1 Settings**

| TSEG 13 | TSEG 12 | TSEG 11 | TSEG 10 | Length of Time (TSEG1) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | not recommended |
| 0 | 0 | 0 | 1 | 2 tq |
| 0 | 0 | 1 | 0 | 3 tq |
| 0 | 0 | 1 | 1 | 4 tq |
| 0 | 1 | 0 | 0 | 5 tq |
| 0 | 1 | 0 | 1 | 6 tq |
| 0 | 1 | 1 | 0 | 7 tq |
| 0 | 1 | 1 | 1 | 8 tq |
| 1 | 0 | 0 | 0 | 9 tq |
| 1 | 0 | 0 | 1 | 10 tq |
| 1 | 0 | 1 | 0 | 11 tq |
| 1 | 0 | 1 | 1 | 12 tq |
| 1 | 1 | 0 | 0 | 13 tq |
| 1 | 1 | 0 | 1 | 14 tq |
| 1 | 1 | 1 | 0 | 15 tq |
| 1 | 1 | 1 | 1 | 16 tq |

TSEG2[2:0] Time Segment 2. The TSEG2[2:0] bits set the number of time quanta (tq) for phase segment 2 (see Table34).

**Table 34 Time Segment 2 Settings**

| TSEG22 | TSEG21 | TSEG20 | Length of TSEG2 |
|---|---|---|---|
| 0 | 0 | 0 | 1 tq |
| 0 | 0 | 1 | 2 tq |
| 0 | 1 | 0 | 3 tq |
| 0 | 1 | 1 | 4 tq |
| 1 | 0 | 0 | 5 tq |
| 1 | 0 | 1 | 6 tq |
| 1 | 1 | 0 | 7 tq |
| 1 | 1 | 1 | 8 tq |

### 20.9.10 Global Mask Registers (GMSK — GMSKB and GMSKX)

The GMSKB and GMSKX registers allow you to globally mask, or “don’t care” the incoming extended/standard identifier bits, RTR/XRTR and IDE. Throughout this document, the GMSKB and GMSKX 16-bit registers are referenced as a 32-bit register GMSK.

GM[28:15] The following are the bits for the GMSKB register.

| 15 5 | 4 | 3 | 2 0 |
|---|---|---|---|
| GM[28:18] | RTR | IDE | GM[17:15] |
| 0 | | | |
| r/w | | | |

GM[14:0] The following are the bits for the GMSKX register.

| 15 1 | 0 |
|---|---|
| GM[14:0] | XRTR |
| 0 | |
| r/w | |

For all GMSKB and GMSKX register bits, the following applies:

— “0” is the incoming identifier bit must match the corresponding bit in the message buffer identifier register.
— “1” accept “1” or “0” (“don’t care”) of the incoming ID bit independent from the corresponding bit in the message buffer ID registers. The corresponding ID bit in the message buffer will be overwritten by the incoming identifier bits.

When an extended frame is received from the CAN bus, all Global Mask bits GM28 through GM0, IDE, RTR and XRTR are used to mask the incoming message.

During the reception of standard frames only the Global Mask bits GM28 to GM18, RTR and IDE are utilized.

| Global Mask | GM[28:18] | RTR [a] | IDE | GM[17:0] | XRTR |
|---|---|---|---|---|---|
| standard frame | ID[10:0] | RTR | IDE | unused | |
| extended frame | ID[28:18] | SRR | IDE | ID[17:0] | RTR |

a. the RTR bit has a different position in standard and extended frames
— for standard frames the GMSK_RTR bit is used to mask this bit
— for extended frames the GMSK_XRTR bit is used to mask this bit

### 20.9.11 Basic Mask Registers (BMSK — BMSKB and BMSKX)

The two registers BMSKB and BMSKX allow to mask the buffer 14, or “don’t care” the incoming extended/standard identifier bits, RTR/XRTR and IDE. Throughout this document, the two 16-bit registers BMSKB and BMSKX are referenced to as a 32-bit register BMSK.

BM[28:15] The following are the bits for the BMSKB register.

| 15 5 | 4 | 3 | 2 0 |
|---|---|---|---|
| BM[28:18] | RTR | IDE | BM[17:15] |
| 0 | | | |
| r/w | | | |

BM[14:0] The following are the bits for the BMSKX register.

| 15 1 | 0 |
|---|---|
| BM[14:0] | XRTR |
| 0 | |
| r/w | |

For all BMSKB and BMSKX register bits the following applies:

— “0” incoming identifier bit must match the corresponding bit in the message buffer identifier register.
— “1” accept “1” or “0” (“don’t care”) of the incoming ID bit independent from the corresponding bit in the message buffer ID registers. The corresponding ID bit in the message buffer will be overwritten by the incoming identifier bits.

When an extended frame is received from the CAN bus all Basic Mask bits BM28 through BM0, IDE, RTR and XRTR are used to mask the incoming message.

During the reception of standard frames only the Basic Mask bits BM28 to BM18, RTR and IDE are utilized.

| Basic Mask | BM[28:18] | RTR [a] | IDE | BM[17:0] | XRTR |
|---|---|---|---|---|---|
| standard frame | ID[10:0] | RTR | IDE | unused | |
| extended frame | ID[28:18] | SRR | IDE | ID[17:0] | RTR |

a. the RTR bit has a different position in standard and extended frames
— for standard frames the BMSK_RTR bit is used to mask this bit
— for extended frames the BMSK_XRTR bit is used to mask this bit

### 20.9.12 CAN Interrupt Enable Register (CIEN)

The CAN Interrupt Enable (CIEN) register enables the transmit/receive interrupts of the message buffers 0 through 14 as well as the CAN Error Interrupt.

| 15 | 14 0 |
|---|---|
| EIEN | IEN[14:0] |
| 0 | |
| r/w | |

EIEN Error Interrupt Enable. This bit allows the CR16CAN to interrupt the CPU if any kind of CAN receive/transmit errors are detected. This means any error status change in the error counter registers REC/TEC is able to generate an error interrupt if EIEN is enabled.
“0” The error interrupt is disabled and no error interrupt will be generated.
“1” The error interrupt is enabled and a change in REC/TEC will cause an interrupt to be generated.

IEN[14:0] Buffer Interrupt Enable. The IEN[14:0] allow the user to enable/disable interrupt source for each of the message buffers i.e., IEN14 configures buffer14 and IEN0 configures buffer0.
“0” buffer as interrupt source disabled
“1” buffer as interrupt source enabled

### 20.9.13 CAN Interrupt Pending Register (CIPND)

The CIPND register indicates any CAN Receive/Transmit Interrupt Requests caused by the message buffers 0..14 and CAN error occurrences.

| 15 | 14 0 |
|---|---|
| EIPND | IPND[14:0] |
| 0 | |
| r | |

EIPND Error Interrupt Pending — EIPND indicates the status change of TEC/REC and will execute an error interrupt if EIEN is set. The user has the responsibility to reset EIPND by means of the CICLR register.
- "0" CAN status is not changed
- "1" CAN status is changed

IPND[14:0] Buffer Interrupt Pending — IPND[14:0] bits are set by CR16CAN following a successful transmission or reception of a message to or from message buffer 0...14, IPND14 for buffer 14 and IPND0 for buffer 0.
- "0" no interrupt pending for this message buffer
- "1" message buffer has generated an interrupt

### 20.9.14 CAN Interrupt Clear Register (CICLR)

The bits in the CICLR register separately clear all CAN interrupt pending flags caused by the message buffers 0...14 and from the Error Management Logic.

| 15 | 14 0 |
|---|---|
| EICLR | ICLR[14:0] |
| 0 | |
| w | |

EICLR Error Interrupt Clear. The EICLR bit can clear the EIPND bit:
- "0" the contents of the EIPND bit is unchanged
- "1" the contents of the EIPND bit is reset

ICLR[14:0] Buffer Interrupt Clear. The user is able to clear the buffer interrupt pending bits by ICLR[14:0]:
- "0" the contents of the respective IPND bit is unchanged
- "1" the contents of the respective IPND bit is reset

### 20.9.15 CAN Interrupt Code Enable Register (CICEN)

The CAN Interrupt Code Enable Register (CICEN) determines whether the interrupt pending flag in IPND should be translated into the Interrupt Code field of the CSTPND register. All interrupt requests, CAN error and buffer 0...14 interrupts can be enabled/disabled separately for the interrupt code indication field.

| 15 | 14 0 |
|---|---|
| EICEN | ICEN[14:0] |
| 0 | |
| r/w | |

EICEN Error Interrupt Code Enable:
- "0" error interrupt pending is not indicated in the interrupt code
- "1" error interrupt pending is indicated in the interrupt code

ICEN[14:0] Buffer Interrupt Code Enable:
- "0" buffer interrupt pending is not indicated in the interrupt code
- "1" buffer interrupt pending is indicated in the interrupt code

### 20.9.16 CAN Status Pending Register (CSTPND)

The CAN Status Pending Register (CSTPND) contains the status of the CAN Node and the Interrupt Code.

| 15 8 | 7 5 | 4 | 3 0 |
|---|---|---|---|
| Reserved | NS[2:0] | IRQ | IST[3:0] |
| 0 | | | |
| r | | | |

NS[2:0] CAN Node Status. This bits indicate the status of the CAN node as it is described in Table35.

**Table 35 CAN Node Status**

| NS2 | NS1 | NS0 | Node Status |
|---|---|---|---|
| 0 | 0 | 0 | Not Active |
| 0 | 1 | 0 | Error active |
| 0 | 1 | 1 | Error Warning Level |
| 1 | 0 | X | Error passive |
| 1 | 1 | X | Bus off |

IRQ,IST[3:0] Interrupt Code. This section of the Status Pending Register represents the interrupt source of the highest priority interrupt currently pending and enabled in the CICEN register. Table36 shows the several interrupt codes for CICEN=FFFF.

**Table 36 Highest Priority Interrupt Code (CICEN = FFFF)**

| CAN interrupt request | IRQ | IST3 | IST2 | IST1 | IST0 |
|---|---|---|---|---|---|
| no request | 0 | 0 | 0 | 0 | 0 |
| Error interrupt | 1 | 0 | 0 | 0 | 0 |
| Buffer 0 | 1 | 0 | 0 | 0 | 1 |
| Buffer 1 | 1 | 0 | 0 | 1 | 0 |
| Buffer 2 | 1 | 0 | 0 | 1 | 1 |
| Buffer 3 | 1 | 0 | 1 | 0 | 0 |
| Buffer 4 | 1 | 0 | 1 | 0 | 1 |
| Buffer 5 | 1 | 0 | 1 | 1 | 0 |
| Buffer 6 | 1 | 0 | 1 | 1 | 1 |
| Buffer 7 | 1 | 1 | 0 | 0 | 0 |
| Buffer 8 | 1 | 1 | 0 | 0 | 1 |
| Buffer 9 | 1 | 1 | 0 | 1 | 0 |

**Table 36 Highest Priority Interrupt Code (CICEN = FFFF)**

| CAN interrupt request | IRQ | IST3 | IST2 | IST1 | IST0 |
|---|---|---|---|---|---|
| Buffer 10 | 1 | 1 | 0 | 1 | 1 |
| Buffer 11 | 1 | 1 | 1 | 0 | 0 |
| Buffer 12 | 1 | 1 | 1 | 0 | 1 |
| Buffer 13 | 1 | 1 | 1 | 1 | 0 |
| Buffer 14 | 1 | 1 | 1 | 1 | 1 |

### 20.9.17 CAN Error Counter Register (CANEC)

The Can Error Counter Register contains the value of the CAN Receive Error Counter and the CAN Transmit Error Counter.

| 15 8 | 7 0 |
|---|---|
| REC[7:0] | TEC[7:0] |
| 0 | |
| r | |

REC[7:0] CAN Receive Error Counter. The bits REC[7:0] holds the value of the receive error counter.

TEC[7:0] CAN Transmit Error Counter. The bits TEC[7:0] holds the value of the transmit error counter.

### 20.9.18 CAN Error Diagnostic Register (CEDIAG)

The CAN Error Diagnostic (CEDIAG) register provides information about the last detected error. CR16CAN is able to identify the field within the CAN frame format, in which the error occurred, and it identifies the bit number of the erroneous bit within the according frame field. The CPU has read only access and all bits will be cleared upon reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 4 | 3 0 |
|---|---|---|---|---|---|---|---|
| Reserved | DRIVE | MON | CRC | STUFF | TXE | EBID[5:0] | EFID[3:0] |
| 0 | | | | | | | |
| r | | | | | | | |

EFID[3:0] Error Field Identifier. The EDIAG bits 3...0 identify the frame field in which the last error occurred. How the various frame fields are coded into the EFID bits is shown in Table37.

**Table 37 Error Field Identifier**

| Field | EFID3 | EFID2 | EFID1 | EFID0 |
|---|---|---|---|---|
| ERROR | 0 | 0 | 0 | 0 |
| ERROR DEL | 0 | 0 | 0 | 1 |
| ERROR ECHO | 0 | 0 | 1 | 0 |
| BUS IDLE | 0 | 0 | 1 | 1 |
| ACK | 0 | 1 | 0 | 0 |
| EOF | 0 | 1 | 0 | 1 |
| INTERMISSION | 0 | 1 | 1 | 0 |
| SUSPEND TRANSMISSION | 0 | 1 | 1 | 1 |
| SOF | 1 | 0 | 0 | 0 |
| ARBITRATION | 1 | 0 | 0 | 1 |

**Table 37 Error Field Identifier**

| Field | EFID3 | EFID2 | EFID1 | EFID0 |
|---|---|---|---|---|
| IDE | 1 | 0 | 1 | 0 |
| EXTENDED ARBITRATION | 1 | 0 | 1 | 1 |
| R1/R0 | 1 | 1 | 0 | 0 |
| DLC | 1 | 1 | 0 | 1 |
| DATA | 1 | 1 | 1 | 0 |
| CRC | 1 | 1 | 1 | 1 |

EBID[5:0] Error Bit Identifier. The EDIAG[9:4] bits contain the number (position) of the incorrect bit within the erroneous frame field. The bit number starts with the value equal to the respective frame field length minus one at the beginning of each field and is decremented with each CAN bit. Figure71 shows an example on how the EBID is calculated.

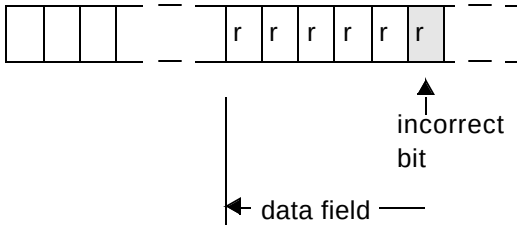



**Figure 71. EBID Example**

Assume the EFID resulted in $1110_2$ and the EBID showed a value of $111001_2$. This means that faulty field was the data field. To calculate the bit position of the error, the DLC of the message needs to be known. For example, for a DLC of 8 data bytes, the bit counter starts with the value: 8 x 8 - 1 = 63; so when EBID[5:0]=$111001_2$ = 57, then the bit number was 63 - 57 = 6.

The following bits provide an information of the error type.

TXE Transmit Error. If set, this bit indicates that the CR16CAN was an active transmitter at the time the error occurred. If reset, the CR16CAN was a receiver.

STUFF Stuff Error. if set, this bit indicates that a the bit stuffing rule was violated at the time the error occurred. Note that certain bit fields do no use bit stuffing and therefore this bit may be ignored in those.

CRC CRC Error. if set, this bit indicates that the CRC is invalid. This bit should only be used if the EFID shows the code of the ACK field.

MON Monitor. This bit shows the bus value on the CANRX pin as seen by the CR16CAN at the time of the error.

DRIVE Drive. This bit shows the output value on the CANTX pin at the time of the error. Note that a

receiver will not drive the bus except during ACK and during an active error flag.

### 20.9.19 CAN Timer Register (CTMR)

The current value of the Time Stamp counter as described in section 20.8 can be monitored via the CAN Timer Register.

| 15 | 0 |
|---|---|
| CTMR[15:0] | |
| 0 | |
| r | |

The CAN Time register is a free running 16-bit counter. It contains the number of CAN bits recognized by CR16CAN since the register has been reset. The counter starts to increment from the value $0000_{16}$ after a hardware reset. If the Timer Stamp enable flag (TSTPEN) in the CAN global configuration register (CGCR) is set, the counter will also be reset upon a message transfer of the message buffer 0.

As described in Time Stamp Counter on page 105, the contents of CTMR are captured into the Time Stamp register of the message buffer after successfully sending or receiving a frame.

## 20.10 SYSTEM START-UP AND MULTI-INPUT WAKE-UP

After system start-up, all CR16CAN related registers are in their reset state. The CR16CAN module can be enabled after all configuration registers are set to their desired value. The following initial setting need to be made:

- configure the CAN Timing register (CTIM) See "Bit Time Logic" on page94.
- configure every buffer to its function as receive/transmit Buffer Status/Control Register (CNSTAT) on page 106.
- set the acceptance filtering masks. See "Acceptance Filtering" on page96.
- enable the CR16CAN interface. See "CAN Global Configuration Register (CGCR)" on page109.

Before disabling the CR16CAN module, the user has to make sure that no transmission is still pending.

**Note:** The device can be awaken from a power saving mode by an activity on the CAN bus by selecting the CAN RX pin as an input to the Multi-Input Wake-Up module. In this case the CR16CAN module must not be disabled before entering the power saving mode. Disabling the CR16CAN module also disables the CAN RX pin.
As an alternative, the CAN RX pin can be connected to any other input pin of the Multi-Input Wake-Up module. This input channel must then be configured to trigger a wake-up event on a falling edge (if a dominant bit is represented by a low level). In this case the CR16CAN module can be disabled before entering a power saving mode. After the device has been waken up, the user has to manually enable the CR16CAN again. All configuration and buffer registers still contain the same data as prior to the power down phase.

### 20.10.1 External Connection

The CR16CAN uses two external pins, CANTX and CANRX to connect to the physical layer of the CAN interface. They provide the functionality as described in Table38.

**Table 38 External CR16CAN Pins**

| Signal Name | Type | Description |
|---|---|---|
| CANTX | Output | Transmit data to the CAN bus |
| CANRX | Input | Receive data from the CAN bus |

The logic levels are configurable by means of two control flags CTX and CRX of the Global Configuration Register CGCR (see "CAN Global Configuration Register (CGCR)" on page 109.

### 20.10.2 Transceiver Connection

An external Transceiver Chip needs to be connected between the CAN block and the bus. It is used to establish a bus connection in differential mode and furthermore provides the driver and protection requirements.

Figure72 shows a possible ISO-High-Speed configuration.

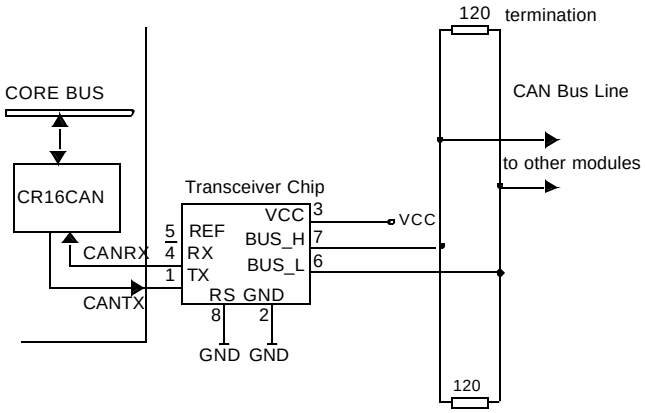



**Figure 72. External Transceiver Connection (ISO-High-Speed)**

### 20.10.3 Timing Requirements

Processing messages and updating message buffers require a certain number of clock cycles by CR16CAN as shown in Table39. These requirements may lead to some restrictions regarding the Bit Time Logic settings and the overall CR16CAN performance which are described below in more detail.

**Table 39 CR16CAN Internal Timing**

| task | # cycles [a] | occurrence/ frame [b] |
|---|---|---|
| copy hidden buffer to receive message buffer | 17 | 0-1 |
| update status from TX_RTR to TX_ONCE_RTR | 3 | 0-15 |
| schedule a message for transmission | 2 | 0-1 |

a. Wait cycles need to be added for CPU access to the message memory as described in CPU Access to CR16CAN Registers/Memory on page 105.

b. Depends on the number of matching identifiers.

The critical path derives from receiving a remote frame which triggers the transmission of one or more data frames. There are a minimum of four bit times in-between two consecutive frames. These bit times start at the validation point of received frame (reception of 6th EOF bit) and end at the earliest possible transmission start of the next frame, which is after the third intermission bit at 100% burst bus load.

These four bit times have to be set in perspective with the timing requirements of the CR16CAN.

The minimum duration of the four CAN bit times is determined by the following Bit Time Logic settings:

PSC = PSCmin = 2

TSEG1 = TSEG1min = 2

TSEG2 = TSEG2min = 1

bit time = Synch + Time Segment 1 + Time Segment 2
= (1 + 2 + 1) tq = 4 tq
= (4 tq x PSC) clock cycles
= (4 tq x 2) clock cycles = 8 clock cycles

For these minimum BTL settings, four CAN bit times take 32 clock cycles.

The following is an example that assumes typical case:

— minimum BTL settings
— reception and copy of a remote frame
— update of one buffer from TX_RTR
— schedule of one buffer from transmit

As outlined in Table39 the copy process, update and scheduling the next transmission gives a total of 17+3+2=22 clock cycles. Therefore under these conditions there is no timing restriction.

The following example assumes the worst case:

— minimum BTL settings
— reception and copy of a remote frame
— update of the 14 remaining buffers from TX_RTR
— schedule of one buffer for transmit

All these actions in total require 17 + 14 x 3 + 2 = 61 clock cycles to be executed by CR16CAN. This leads to the limitation of the Bit Time Logic of 61 / 4 = 15.25 clock cycles per CAN bit as a minimum, resulting in the minimum clock frequencies listed below (the frequency depends on the desired baud rate and assumes the worst case scenario can occur in the application).

Table40 gives examples for the minimum clock frequency in order to ensure proper functionality at various CAN bus speeds.

**Table 40 Min. Clock Frequency Requirements**

| Baud Rate | min. clock frequency |
|---|---|
| 1Mbit/sec | 15.25MHz |
| 500kbit/sec | 7.625MHz |
| 250kbit/sec | 3,81MHz |

### 20.10.4 Bit Time Logic Calculation Examples

The calculation of the CAN bus clocks using CKI = 16MHz is shown in the following examples. The desired baud rate for both examples is 1Mbit/s.

Example 1

PSC = PSC[5:0] + 2 = 0 + 2 = 2

TSEG1 = TSEG1[3:0] + 1 = 3 + 1 = 4

TSEG2 = TSEG2[2:0] + 1 = 2 + 1 = 3

SJW = TSEG2 = 3

— sample point positioned at 62.5% of bit time
— bit time = 125ns x (1 + 4 + 3 ± 3) = (1 ± 0.375)µs
— busclock = 16MHz / (2 x (1 + 4 + 3)) = 1Mbit/s (nominal)

Example 2

PSC = PSC[5:0] + 1 = 2 + 2 = 4

TSEG1 = TSEG1[3:0] + 1 = 1 + 1 = 2

TSEG2 = TSEG2[2:0] + 1 = 0 + 1 = 1

SJW = TSEG2 = 1

- sample point positioned at 75% of bit time
- bit time = 250ns x (1 + 2 + 1 ± 1) = (1 ± 0.25) µs
- busclock = 16MHz / (2 x (1 + 4 + 3)) = 1Mbit/s (nominal)

### 20.10.5 Acceptance Filter Considerations

The CR16CAN provides two acceptance filter masks GMSK and BMSK as described in Acceptance Filtering on page 96, Global Mask Registers (GMSK — GMSKB and GMSKX) on page 112 and Basic Mask Registers (BMSK — BMSKB and BMSKX) on page 113. These masks allow filtering of up to 32 bits of the message, which includes the standard identifier, the extended identifier as well as the frame control bits RTR, SRR and IDE.

### 20.10.6 Remote Frames

Remote frames can be automatically processed by the CR16CAN interface. However, to fully enable that feature, the RTR/XRTR bits (for both standard and extended frames) within the BMSK and/or GMSK register need to be set to “don’t care”. This is because a remote frame with the RTR bit being set to “1” should trigger the transmission of a data frame with the RTR bit set to “0” and therefore the ID bits of the received message need to pass through the acceptance filter. The same applies to transmitting remote frames and switching to receive the corresponding data frames.

# 21.0 Analog Comparators

The Dual Analog Comparator (ACMP2) module contains two independent analog comparators with all necessary control logic. Each comparator unit compares the analog input voltages applied to two input pins and determines which voltage is higher. The comparison results can be placed on two output pins and/or read by the software from a register.

Figure73 is a block diagram of the Dual Analog Comparator module.

The two comparators are designated Comparator 1 (CMP1) and Comparator 2 (CMP2). Each comparator has a positive and a negative input, called CMP1P and CMP1N for Comparator 1 and CMP2P and CMP2N for Comparator 2. An optional output, CMP1O for Comparator 1 or CMP2O for Comparator 2, allows the external hardware to read the comparison results. If the positive input is greater than the negative input, the result is a logic 1. Otherwise, the result is a logic 0. These same results are available to the software by reading the CMPCTRL register. CMP1OP and CMP2OP are the direct outputs of the analog comparator. These signals are connected to the channels of the Multi-Wake-Up module.

## 21.1 ANALOG COMPARATOR CONTROL/ STATUS REGISTER (CMPCTRL)

The CMPCTRL register is a byte-wide, read/write register that controls the comparator module and contains the comparison results. The control bits are read/write bits and the result bits are read-only bits. This register is cleared upon reset. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | CMP2OE | CMP1OE | CMP2EN | CMP1EN | CMP2RD | CMP1RD |

CMP1RD Comparator 1 Read. This read-only bit contains the output of Comparator 1 when the comparator is enabled (CMP1EN=1). CMP1RD is set to 1 when the voltage on CMP1P is greater than the voltage on CMP1N. This bit is always 0 when Comparator 1 is disabled.

CMP2RD Comparator 2 Read. This read-only bit contains the output of Comparator 2 when the comparator is enabled (CMP2EN=1). CMP2RD is set to 1 when the voltage on CMP2P is greater than the voltage on CMP2N. This bit is always 0 when Comparator 2 is disabled.

CMP1EN Comparator 1 Enable. This read/write bit enables (1) or disables (0) Comparator 1.

CMP2EN Comparator 2 Enable. This read/write bit enables (1) or disables (0) Comparator 2.

CMP1OE Comparator 1 Output Enable. This read/write bit, when set to 1, enables the use of the CMP1O pin as the output of Comparator 1 when Comparator 1 is enabled (CMP1EN=1). If Comparator 1 is disabled (CMP1EN=0), setting the CMP1OE bit results in a logic 0 on the CMP1O output pin.

CMP2OE Comparator 2 Output Enable. This read/write bit, when set to 1, enables the use of the CMP2O pin as the output of Comparator 2 when Comparator 2 is enabled (CMP2EN=1). If Comparator 2 is disabled (CMP2EN=0), setting the CMP2OE bit results in a logic 0 on the CMP2O output pin.

## 21.2 ANALOG COMPARATOR USAGE

The comparator I/O pins are alternate functions of the Port L pins. In order for a comparator to operate, its two input pins must be configured to operate as inputs in the alternate function mode.

Using a comparator's output pin is optional. If it is to be used, it must be configured to operate as an output in the alternate function mode. The comparison result bits in the CMPCTRL register are available to the CPU whether or not the output pin is enabled.

The comparators uses DC current whenever they are enabled. Therefore, in order to reduce power consumption, it is recommended that the comparators be disabled when they are not needed, especially before entering any of the Power Save modes.

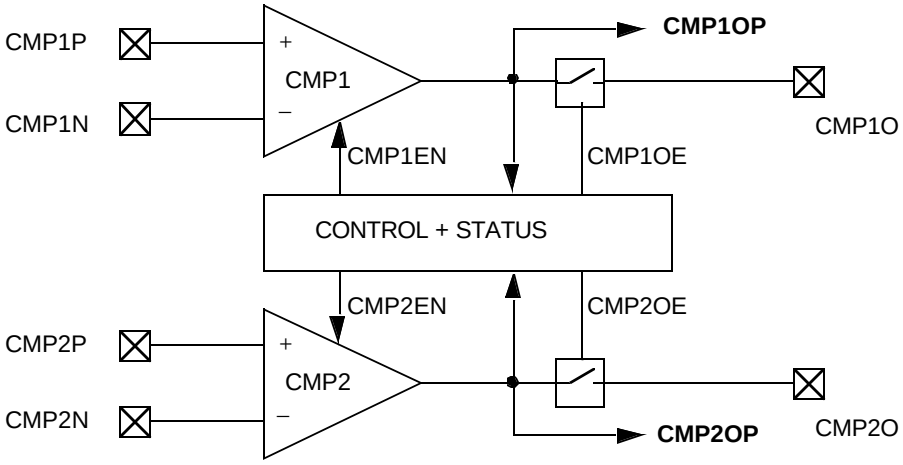



**Figure 73. Dual Analog Comparator Block Diagram**

# 22.0 A/D Converter

The A/D Converter (ADC) module is a 12-channel, multiplexed-input, analog-to-digital converter. The A/D Converter receives an analog voltage on an input pin and converts that voltage into an 8-bit digital value using successive approximation. The CPU can then read the result from a memory-mapped register. The module supports four automated operating modes, providing single-channel or 4-channel scanned operation in single-conversion or continuous mode.

Figure74 is a block diagram of the A/D Converter module.

The analog input signal is selected from the analog inputs using a 12-channel analog multiplexer. The input pins are alternate functions of Port I.

A sample-and-hold circuit samples the analog voltage prior to conversion and holds it stable throughout the conversion process. A programmable initial delay period allows the sampled voltage to stabilize before the conversion process begins.

The input voltage range is from 0V to $V_{REF}$ (the A/D reference voltage). The device has a separate pin, $V_{REF}$, for the reference voltage.

A capacitor should be connected between the $V_{REF}$ and the $A_{VCC}$ pin in order to minimize noise. The recommended value for this capacitor is about 0.47µF.

The internal analog-to-digital converter block is based on a successive approximation algorithm, which compares the sampled voltage against an internally generated sequence of analog voltages. The result is a linear conversion of the analog voltage to an unsigned 8-bit value ranging from 00 hex for 0.0 volts to FF hex for $V_{REF}$.

The clock used by the converter block is generated by a clock divider that scales down the system clock by a programmable factor. The conversion algorithm requires ten A/D Converter clock cycles, or 10 microseconds at the maximum allowed A/D Converter clock rate of 2 MHz.

Conversion can start after the power supply is stable and ADCEN set for 30 µs.

The conversion results are stored in a 4-level data buffer. Depending on the operating mode, the buffer can hold the results of four successive conversions from a single channel or four conversions from adjacent channels scanned in sequence.

## 22.1 OPERATING MODES

The A/D Converter can be configured to operate in any one of four modes:

— Single channel, single conversion
— Single channel, continuous conversion
— 4-channel scan, single conversion
— 4-channel scan, continuous conversion

The configuration is set by the SCAN and CONT fields in the ADC Control 2 Register (ADCCNT2), as indicated in Table41. The A/D converter must be disabled when switching to a different mode.

**Table 41 ADC Operation Modes**

| SCAN | CONT | Mode |
|---|---|---|
| 00 | 0 | Single Channel, Single Conversion |
| 00 | 1 | Single Channel, Continuous Conversion |
| 01 | 0 | 4 Channels Scan, Single Conversion |
| 01 | 1 | 4 Channel Scan, Continuous Conversion |

### 22.1.1 Single Channel, Single Conversion Mode

In the single channel, single conversion mode, the A/D Converter performs a single conversion using a specified channel.

The software starts a conversion by setting the START bit in the ADCCNT2 register. Upon completion of the conversion, the A/D Converter places the result in register ADDATA0, clears the START bit, and sets the EOC (end of conversion) bit in the ADCST register. If the A/D Converter interrupt is enabled, an interrupt to the CPU is generated at this time.

### 22.1.2 Single Channel, Continuous Conversion Mode

In the single channel, continuous conversion mode, the A/D Converter performs conversions repeatedly using the same specified channel.

The software starts a conversion sequence by setting the START bit. The A/D Converter performs four A/D conversions in sequence using the same channel, pausing only for the programmable sampling delay time used in all conversion operations. It loads the four results into the A/D data registers in sequence, starting with ADDATA0 and ending with ADDATA3. After it loads all four registers, it sets the EOC (end of conversion) bit. If the A/D Converter interrupt is enabled, an interrupt to the CPU is generated at this time.

The START bit remains set until cleared by the software. If the software does not clear the START bit, the A/D Converter continues performing conversions using the same input channel, storing the results in ADDATA0 following ADDATA3. To prevent an overrun error, the software must read the results from the data registers before the A/D Converter writes the next result into ADDATA0 following ADDATA3.

When the software clears the START bit, the A/D Converter first completes the conversion currently in progress, then stops and sets the EOC bit. A 2-bit buffer pointer in the ADCST register points to the register containing the final result.

### 22.1.3 4-Channel Scan, Single Conversion Mode

In the 4-channel scan, single conversion mode, the A/D Converter performs four conversions using four adjacent input channels.

The software starts the conversion sequence by setting the START bit. The A/D Converter performs four A/D conversions in sequence using four adjacent channels, starting with the specified channel and pausing only for the programmable sampling delay time. It loads the four results into the A/D data registers in sequence, starting with ADDATA0 and ending

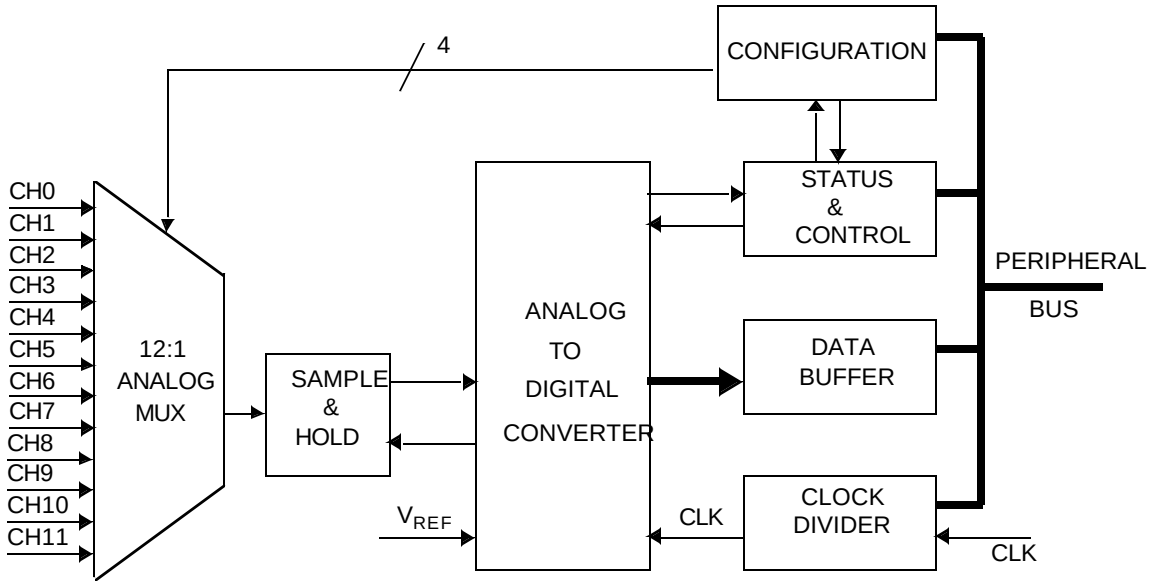



**Figure 74. A/D Converter Block Diagram**

with ADDATA3. After it loads all four registers, it clears the START bit and sets the EOC (end of conversion) bit. If the A/D Converter interrupt is enabled, an interrupt to the CPU is generated at this time.

#### 22.1.4 Channel Scan, Continuous Conversion Mode

In the 4-channel scan, continuous conversion mode, the A/D Converter performs conversions repeatedly using four adjacent input channels.

The software starts conversion operations by setting the START bit. The A/D Converter performs four A/D conversions in sequence using four adjacent channels, starting with the specified channel and pausing only for the programmable sampling delay time. It loads the four results into the A/D data registers in sequence, starting with ADDATA0 and ending with ADDATA3. After it loads all four registers, it sets the EOC (end of conversion) bit. If the A/D Converter interrupt is enabled, an interrupt to the CPU is generated at this time.

The START bit remains set until cleared by the software. If the software does not clear the START bit, the A/D Converter continues performing conversions, repeating the same sequence using the same four input channels and the same sequence of data registers. To prevent an overrun error, the software must read the results from the data registers before the A/D Converter writes the next result into ADDATA0.

When the software clears the START bit, the A/D Converter first completes the 4-channel conversion sequence currently in progress, then stops and sets the EOC bit.

### 22.2 A/D CONVERTER REGISTERS

The software controls the A/D Converter and reads the A/D results by accessing the ADC registers. There are eight such registers:

— ADC Status Register (ADCST)
— ADC Control 1 Register (ADCCNT1)
— ADC Control 2 Register (ADCCNT2)
— ADC Control 3 Register (ADCCNT3)
— ADC Data Registers (ADDATA0 through ADDATA3)

#### 22.2.1 ADC Status Register (ADCST)

The ADCST register is a byte-wide register that indicates the current status of the A/D Converter. One bit in this register, the OVF flag bit, is cleared by writing a 1 to its bit position. The other bits are read-only bits, so the values written to them are ignored. Upon reset, the register is set to 30 hex. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | BUFPTR | | Reserved | OVF | BUSY | EOC |

EOC — End of Conversion. This read-only bit reports the status of the most recent A/D Converter operation. When cleared to 0, it indicates that the conversion is not complete. When set to 1, it indicates that the conversion is complete. The hardware sets this bit when it places the conversion results in the buffer and clears it when any of the data registers are read.

BUSY — ADC Busy. This read-only bit is set to 1 when the A/D Converter is busy converting data and is cleared to 0 when the A/D Converter is idle or disabled.

OVF — Overflow. The hardware sets this bit to 1 when the A/D Converter finishes a conversion and attempts to store the results in one of the data registers (ADDATA0-ADDATA3) while the register is full. When this happens, the A/D Converter overwrites the data in the data register, sets the OVF flag, and continues operating.
The OVF flag remains set until cleared by the software. The software clears the flag by writing a 1 to it. Writing a 0 to this bit has no effect.

BUFPTR — Buffer Pointer. This 2-bit, read-only field identifies the data register that was most recently written with new data:
00 = ADDATA0
01 = ADDATA1
10 = ADDATA2
11 = ADDATA3

This register is initialized to 11 when a new conversion is started (when ADCCNT2.START is changed from 0 to 1) and is automatically incremented every time a result is written to buffers ADDATA0-ADDATA3. The result is a four-entry cyclic FIFO buffer, with BUFPTR pointing to the last entry written by the A/D Converter.

### 22.2.2 ADC Control 1 Register (ADCCNT1)

The ADCCNT1 register is a byte-wide, read/write register used to enable the A/D Converter and its interrupts, and also to control the reference voltage source. When writing to this register, all reserved bits must be written with 0 for the A/D Converter to function properly. Changing any bits other than ADCEN (bit 0) is not allowed while the A/D Converter is active (ADCST.BUSY or ADCCNT2.START set). Upon reset, all non-reserved bits are cleared to 0. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | INTE | Reserved | ADCEN |

ADCEN A/D Converter Enable. Setting this bit enables the A/D Converter and allows a conversion to be started by setting the start bit (ADCCNT2.START). Clearing the ADCEN bit disables the A/D Converter, terminates any conversion in progress, and clears the ADC status flags (ADCST.EOC, ADCST.BUSY, ADCST.OVF, and ADCCNT2.START).

INTE Interrupt Enable. This bit enables (1) or disables (0) A/D Converter interrupts. If enabled, and interrupt occurs at the end of a conversion sequence or when the ADC data buffer is full, depending on the operating mode.

All reserved bits must be written with 0 for ADC to operate properly.

### 22.2.3 ADC Control 2 Register (ADCCNT2)

The ADCCNT2 register is a byte-wide, read/write register used to specify the A/D Converter operating mode and to start conversion operations. All register fields other than the START bit should be changed only while the A/D Converter is inactive (START=0). Data written to the SCAN and CONT fields is ignored if the START bit is already set. Upon reset, the non-reserved bits of this register are cleared to 0. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| START | SCAN | | CONT | CHANNEL | | | |

CHANNEL Channel Select. This 4-bit field selects one of the twelve analog input channels as follows:
0000 = ACH0
0001 = ACH1
0010 = ACH2
0011 = ACH3
0100 = ACH4
0101 = ACH5
0110 = ACH6
0111 = ACH7
1000 = ACH8
1001 = ACH9
1010 = ACH10
1011 = ACH11
11XX = reserved

CONT Continuous Conversion. When cleared to 0, the A/D Converter stops operating upon completion of the programmed conversion cycle (a single conversion or a sequence of four conversions on four channels). When set to 1, the A/D Converter operates continuously by repeating the programmed conversion cycle.

SCAN Scan Mode. This 2-bit field selects the single-conversion mode or 4-channel scan mode as follows:
00 = single-conversion mode
01 = 4-channel scan mode
1X = reserved

START Start Conversion. The software sets this bit to 1 to start a conversion or a 4-channel conversion cycle. In the “continuous” mode, this bit remains set until cleared by the software. In the “single” (non-continuous) mode, the hardware clears this bit upon completion of the programmed conversion cycle. The software should not attempt to set this bit while the A/D Converter is busy (ADCST.BUSY=1).

### 22.2.4 ADC Control 3 Register (ADCCNT3)

The ADCCNT3 register is a byte-wide, read/write register used to specify the analog sampling time delay and the divide-by factor for generating the ADC clock. This register should be written only when the A/D Converter is disabled (ADCCNT1.ADCEN=0). Upon reset, the non-reserved bits of the ADCCNT3 register are cleared to 0. The register format is shown below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | PWREN | DELAY | | | CDIV | | |

CDIV Clock Divide. This 3-bit field sets the divide-by factor for generating the A/D Converter clock from the system clock. The frequency of the A/D Converter clock is equal to the system clock divided by the programmed factor. The resulting A/D Converter clock frequency must be less than or equal to 2 MHz. The divide-by factor is defined as follows:
000 = divide by 1
001 = divide by 2
010 = divide by 4
011 = divide by 8
100 = divide by 16
101 = divide by 32
Other = reserved

DELAY Sampling Time Delay. This 3-bit field defines the number of A/D Converter clock cycles of delay from the time that the input channel is selected until the analog voltage is sampled. The programmed delay should be sufficient, dependent on the source impedance, to allow the sampled signal to reach its final level before the conversion begins. The delay is defined as follows:

000 = 1 A/D Converter clock cycle
001 = 2 A/D Converter clock cycles
010 = 4 A/D Converter clock cycles
011 = 8 A/D Converter clock cycles
100 = 16 A/D Converter clock cycles
101 = 32 A/D Converter clock cycles
110 = 64 A/D Converter clock cycles
111 = reserved

PWREN Power Down Enable. Controls the condition when the ADC is powered down. When PWREN is cleared (0), the ADC powers down upon reset. When PWREN is set (1), the ADC powers down when the ADCEN bit is low.

### 22.2.5 ADC Data Registers (ADDATA0-ADDATA3)

The four ADC Data Registers (ADDATA0 through ADDATA3) are byte-wide, read/write registers that hold the conversion results, which are stored sequentially starting with ADDATA0 and ending with ADDATA3. The results held in these registers are valid only after the ADCST.EOC flag is set. Upon reset, the contents of these registers are undefined.

The value read from a data register is a linear mapping of the analog input voltage to an 8-bit value. The value 00 hex represents 0.0 volts and the value FF hex represents the reference voltage, $V_{REF}$.

## 22.3 A/D CONVERTER PROGRAMMING

The software should set the A/D Converter configuration before it enables the A/D Converter module. The configuration consists of the following settings:

— ADC clock rate: ADCCNT3.CDIV
— Sampling delay: ADCCNT3.DELAY
— Interrupt enable (if required): ADCCNT1.INTE

The ADC clock is created by scaling down the system clock. The fastest allowable clock for the A/D Converter is 2 MHz. Therefore, for the fastest possible operation of the A/D Converter, use the smallest available divide-by factor that results in a clock frequency of 1 MHz or lower. The available divide-by factors are 1, 2, 4, 8, 16, and 32.

For example, if the system clock is 10 MHz, use a divide-by factor of 16. In that case, the A/D Converter clock frequency is 625 kHz, the clock period is 1.6 microseconds, and the A/D conversion time is 16 microseconds (ten clock A/D Converter clock cycles).

The programmable sampling time delay should be made small for faster operation, but large enough to allow the input voltage to settle. The internal resistance and capacitance of the A/D Converter, together with the source resistance of the device that drives the A/D input determine the charge-up time required for the voltage to settle. Figure75 shows a schematic of the charge-up circuit. For the values of $R_{AIN}$ and $C_{AIN}$, see Section25.0.

Interrupts or polling can be used to read the A/D Converter results. For interrupts, the A/D Converter interrupt must be enabled by setting the ADCCNT1.INTE bit. The interrupt is cleared automatically when any one of the data registers (ADDATA0-ADDATA3) is read. For polling, the software reads the ADCST.EOC bit to determine whether the conversion sequence is completed.

Once the A/D Converter configuration has been set up, the software can use the following procedure to perform an A/D conversion sequence:

1. Enable the A/D Converter by setting the ADCCNT1.ADCEN bit and wait 30 µs before performing any conversion.
2. Select the operating mode and channel by writing to the SCAN, CONT, and CHANNEL fields of the ADCCNT2 register. At the same time, start the conversion by setting the START bit in the same register.
3. Wait until the conversion is finished, either by polling or using the A/D Converter interrupt.
4. Read the conversion results from the data registers, ADDATA0 through ADDATA3 (or just ADDATA0 in the single-channel, single-conversion mode).
5. In the continuous conversion modes, repeat Step 3 and Step 4 for as long as samples are needed. Then stop the A/D Converter by clearing either the START bit (ADCCNT2.START) or the A/D Converter enable bit (ADCCNT1.ADCEN).

To minimize power consumption, the A/D Converter should be disabled when it is not needed, especially before entering a Power Save mode.

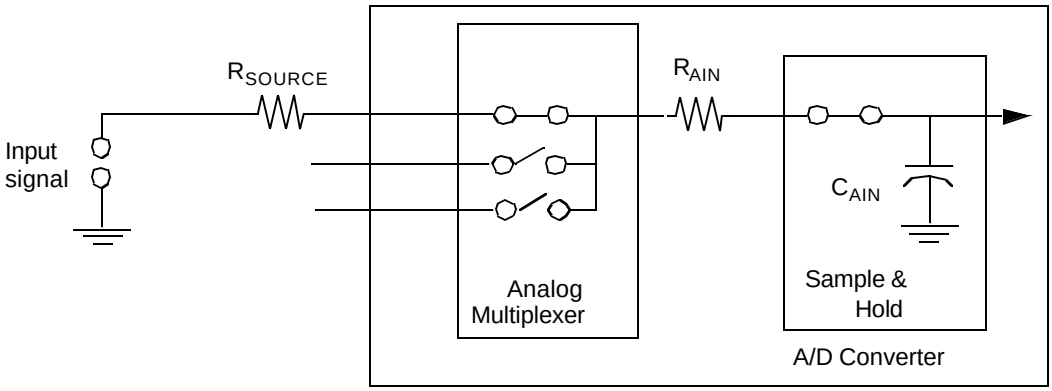



**Figure 75. Sample-and-Hold Charge-Up Schematic**

## 23.0 Memory Map

The CompactRISC architecture supports a uniform linear address space of 2 megabytes. The device implementation of this architecture uses only the lowest 128K bytes of address space, ranging from 0000 to 1FFFF hex. Table42 is a memory map showing the types of memory and peripherals that occupy this memory space. Address ranges not listed in the table are reserved and should not be read or written.

**Table 42 Device Memory Map**

| Address Range (hex) | Description |
|---|---|
| 0000-7FFF | Flash Program Memory[a] |
| 8000-BFFF | Flash Program Memory (48K bytes) |
| C000-CBFF | Static RAM (3K bytes) |
| E000-E5FF | ISP Memory(1.5K bytes) |
| E800-EFFF | Lower Endurance Flash EEPROM Data Memory (2K bytes) |
| F000-F07F | High Endurance Flash EEPROM Data Memory (128 bytes) |
| F400-F7FF | CAN buffers and registers (1K bytes) |
| F800-FAFF | BIU Peripherals (768 bytes) |
| FB00-FB06 | Port B registers |
| FB00-FBFF | I/O Expansion + Ports PB & PC (256bytes) |
| FB10-FB16 | Port C registers |
| FC00-FFFF | Peripherals and other I/O Ports (1K bytes) |
| FC40-FC8A | Clock, Power Management, and Wake-Up registers |
| FCA0-FCA8 | Port G registers |
| FCC0-FCC8 | Port H registers |
| FF00-FF08 | Port L registers |
| FD20-FD28 | Port F registers |
| FE00-FE1E | Interrupt Control Unit registers |
| FE40-FE4E | USART 1 registers |
| FE60-FE66 | MICROWIRE registers |
| FE80-FE8E | USART 2 registers |
| FEC0-FECA | ACCESS.bus registers |
| FEE0-FEE8 | Port I registers |
| FF20-FF2A | Timer and WATCHDOG registers |
| FF40-FF50 | Multi-function Timer1 registers |
| FF60-FF70 | Multi-function Timer2 registers |
| FF80-FFA4 | Versatile Timer Unit registers |
| FFC0-FFD0 | A/D Converter registers |
| FFE0-FFE0 | Analog Comparator register |
| 1C000-1FFFF | Flash Program Memory (16K bytes)[b] |

[a]. 32K ROM or Flash, size depends on device specifications.

Table43 is a detailed memory map showing the specific memory address of the memory, I/O ports, and registers. The table shows the starting address, the size, and a brief description of each memory block and register. For detailed information on using these memory locations, see the applicable sections in the data sheet.

All addresses not listed in the table are reserved and should not be read or written. An attempt to access an unlisted address will have unpredictable results.

Each byte-wide register occupies a single address and can be accessed only in a byte-wide transaction. Each word-wide register occupies two consecutive memory addresses and can be accessed only in a word-wide transaction. Both the byte-wide and word-wide registers reside at word boundaries (even addresses). Thus, each byte-wide register uses only the lowest eight bits of the internal data bus.

Most device registers are read/write registers. However, some registers are read-only or write-only, as indicated in the table. An attempt to read a write-only register or to write a read-only register will have unpredictable results.

When the software writes to a register in which one or more bits are reserved, it must write a zero to each reserved bit unless indicated otherwise in the description of the register. Reading a reserved bit returns an undefined value.

**Table 43 Device Detailed Memory Map**

| Register Name | Size | Register Address (hex) | Access Type | Contents |
|---|---|---|---|---|
| | 32K/48K | 0000 | | Flash EEPROM Program Memory |
| | 3K | C000 | | On-Chip RAM |
| | 2K | E800 | | Low Endurance Flash EEPROM Data Memory |
| | 1.5K | E000 | | ISP Memory |
| | 128 | F000 | | High Endurance Flash EEPROM Data Memory |
| CMB0_CNTSTAT | word | F400 | Read/Write | CAN message buffer 0 Status Register |
| CMB0_TSTP | word | F402 | Read/Write | CAN message buffer 0 Time stamp Register |
| CMB0_DATA3 | word | F404 | Read/Write | CAN message buffer 0 Data 3 Register |
| CMB0_DATA2 | word | F406 | Read/Write | CAN message buffer 0 Data 2 Register |
| CMB0_DATA1 | word | F408 | Read/Write | CAN message buffer 0 Data 1 Register |
| CMB0_DATA0 | word | F40A | Read/Write | CAN message buffer 0 Data 0 Register |
| CMB0_ID0 | word | F40C | Read/Write | CAN message buffer 0 Identifier 0 Register |
| CMB0_ID1 | word | F40E | Read/Write | CAN message buffer 0 Identifier 1 Register |
| CMB1 | 8-word | F410 | Read/Write | CAN message buffer 1 Register |
| CMB2 | 8-word | F420 | Read/Write | CAN message buffer 2 Register |
| CMB3 | 8-word | F430 | Read/Write | CAN message buffer 3 Register |
| CMB4 | 8-word | F440 | Read/Write | CAN message buffer 4 Register |
| CMB5 | 8-word | F450 | Read/Write | CAN message buffer 5 Register |
| CMB6 | 8-word | F460 | Read/Write | CAN message buffer 6 Register |
| CMB7 | 8-word | F470 | Read/Write | CAN message buffer 7 Register |
| CMB8 | 8-word | F480 | Read/Write | CAN message buffer 8 Register |
| CMB9 | 8-word | F490 | Read/Write | CAN message buffer 9 Register |
| CMB10 | 8-word | F4A0 | Read/Write | CAN message buffer 10 Register |
| CMB11 | 8-word | F4B0 | Read/Write | CAN message buffer 11 Register |
| CMB12 | 8-word | F4C0 | Read/Write | CAN message buffer 12 Register |
| CMB13 | 8-word | F4D0 | Read/Write | CAN message buffer 13 Register |
| CMB14 | 8-word | F4E0 | Read/Write | CAN message buffer 14 Register |
| CGCR | word | F500 | Read/Write | CAN Global Configuration Register |
| CTIM | word | F502 | Read/Write | CAN Timing Register |
| GMSKX and GMSK | word | F504 | Read/Write | CAN Global Mask Registers |
| GMSKX and GMSKB | word | F508 | Read/Write | CAN Basic Mask Registers |
| CIEN | word | F50C | Read/Write | CAN Interrupt Enabled Register |
| CIPND | word | F50E | Read/Write | CAN Interrupt Pending Register |
| CICLR | word | F510 | Read/Write | CAN Interrupt Clear Register |
| CICEN | word | F512 | Read/Write | CAN Interrupt Code Enable Register |
| CSTPND | word | F514 | Read/Write | CAN Status Pending Register |
| CANEC | word | F516 | Read/Write | CAN Error Counter Register |
| CEDIAG | word | F518 | Read/Write | CAN Error Diagnostic Register |
| CTMR | word | F51A | Read/Write | CAN Timer Register |
| BCFG | byte | F900 | Read/Write | BIU Configuration Register |
| IOCFG | word | F902 | Read/Write | I/O Zone Configuration Register |
| SZCFG0 | word | F904 | Read/Write | Static Zone 0 Configuration Register |
| SZCFG1 | word | F906 | Read/Write | Static Zone 1 Configuration Register |
| MCFG | byte | F910 | Read/Write | Module Configuration Register |
| MSTAT | byte | F914 | Read Only | Module Status Register |

**Table 43 Device Detailed Memory Map**

| Register Name | Size | Register Address (hex) | Access Type | Contents |
|---|---|---|---|---|
| FLCTRL1 | byte | F930 | Read/Write | Flash EEPROM Program Memory Control Register 1 |
| FLSEC | byte | F932 | Read/Write | Flash EEPROM Program Memory Security Register |
| ISPKEY | byte | F934 | Read/Write | ISP Memory Write Key Register |
| FLCTRL2 | word | F936 | Read/Write | Flash EEPROM Program Memory Control Register 2 |
| DMCSR | byte | F940 | Read/Write | EEPROM Data Memory Control and Status Register |
| DMPSLR | byte | F942 | Read/Write | EEPROM Data Memory Prescaler Register |
| DMSTART | byte | F944 | Read/Write | Data Memory Start Time Reload Register |
| DMTRAN | byte | F946 | Read/Write | Data Memory Transition Time Reload Register |
| DMPROG | byte | F948 | Read/Write | Data Memory Programming Time Reload Register |
| DMERASE | byte | F94A | Read/Write | Data Memory Erase Time Reload Register |
| DMEND | byte | F94C | Read/Write | Data Memory End Time Reload Register |
| DMPCNT | byte | F94E | Read/Write | Data Memory Prescaler Count Register |
| DMCNT | word | F950 | Read/Write | Data Memory Timer Count Register |
| DMKEY | byte | F954 | Read/Write | EEPROM Data Memory Write Key Register |
| FLCSR | byte | F960 | Read/Write | Flash EEPROM Program Memory Control and Status Register |
| FLPSLR | byte | F962 | Read/Write | Flash EEPROM Program Memory Prescaler Register |
| FLSTART | byte | F964 | Read/Write | Program Memory Start Time Reload Register |
| FLTRAN | byte | F966 | Read/Write | Program Memory Transition Time Reload Register |
| FLPROG | byte | F968 | Read/Write | Program Memory Programming Time Reload Register |
| FLERASE | byte | F96A | Read/Write | Program Memory Erase Time Reload Register |
| FLEND | byte | F96C | Read/Write | Program Memory End Time Reload Register |
| FLPCNT | byte | F96E | Read/Write | Program Memory Prescaler Count Reload Register |
| FLCNT1 | byte | F970 | Read/Write | Program Memory Timer Count Register 1 |
| FLCNT2 | byte | F972 | Read/Write | Program Memory Timer Count Register 2 |
| PGMKEY | byte | F974 | Read/Write | Flash EEPROM Program Memory Write Key Register |
| PBDIR | byte | FB00 | Read/Write | Port B Direction Register |
| PBDIN | byte | FB02 | Read Only | Port B Data Input Register |
| PBDOUT | byte | FB04 | Read/Write | Port B Data Output Register |
| PBWKPU | byte | FB06 | Read/Write | Port B Weak Pull-Up Register |
| PCDIR | byte | FB10 | Read/Write | Port C Direction Register |
| PCDIN | byte | FB12 | Read Only | Port C Data Input Register |
| PCDOUT | byte | FB14 | Read/Write | Port C Data Output Register |
| PCWKPU | byte | FB16 | Read/Write | Port C Weak Pull-Up Register |
| CRCTRL | byte | FC40 | Read/Write | Clock and Reset Control Register |
| PRSSC | byte | FC42 | Read/Write | Slow Clock Prescaler Register |
| PRSSC1 | byte | FC44 | Read/Write | Prescaler Slow Clock 1 Register |
| PMCSR | byte | FC60 | Read/Write | Power Management Control/Status Register |
| WKEDG | word | FC80 | Read/Write | Wake-Up Edge Detection Register |
| WKENA | word | FC82 | Read/Write | Wake-Up Enable Register |
| WKICT1 | word | FC84 | Read/Write | Wake-Up Interrupt Control Register 1 |
| WKICTL2 | word | FC86 | Read Set | Wake-Up Interrupt Control Register 2 |
| WKPND | word | FC88 | Write Only | Wake-Up Pending Register |
| WKPCL | word | FC8A | Read/Write | Wake-Up Pending Clear Register |
| PGALT | byte | FCA0 | Read/Write | Port G Alternate Function Register |

**Table 43 Device Detailed Memory Map**

| Register Name | Size | Register Address (hex) | Access Type | Contents |
|---|---|---|---|---|
| PGDIR | byte | FCA2 | Read/Write | Port G Direction Register |
| PGDIN | byte | FCA4 | Read Only | Port G Data Input Register |
| PGDOUT | byte | FCA6 | Read/Write | Port G Data Output Register |
| PGWKPU | byte | FCA8 | Read/Write | Port G Weak Pull-Up Register |
| PHALT | byte | FCC0 | Read/Write | Port H Alternate Function Register |
| PHDIR | byte | FCC2 | Read/Write | Port H Direction Register |
| PHDIN | byte | FCC4 | Read Only | Port H Data Input Register |
| PHDOUT | byte | FCC6 | Read/Write | Port H Data Output Register |
| PHWKUP | byte | FCC8 | Read/Write | Port H Weak Pull-Up Register |
| PFALT | byte | FD20 | Read/Write | Port F Alternate Function Register |
| PFDIR | byte | FD22 | Read/Write | Port F Direction Register |
| PFDIN | byte | FD24 | Read Only | Port F Data Input Register |
| PFDOUT | byte | FD26 | Read/Write | Port F Data Output Register |
| PFWKPU | byte | FD28 | Read/Write | Port F Weak Pull-Up Register |
| IVCT | byte | FE00 | Read Only | Interrupt Vector Register |
| NMISTAT | byte | FE02 | Read Only | NMI Status Register |
| EXNMI | byte | FE04 | Read/Write | External NMI Control/Status Register |
| ISTAT0 | word | FE0A | Read Only | Interrupt Status Register 0 |
| ISTAT1 | word | FE0C | Read Only | Interrupt Status Register 1 |
| IENAM0 | word | FE0E | Read/Write | Interrupt and Enable Mask Register 0 |
| IENAM1 | word | FE10 | Read/Write | Interrupt and Enable Mask Register 1 |
| IDBG | word | FE1A | Read/Write | Interrupt Debug Register |
| U1TBUF | byte | FE40 | Read/Write | USART 1 Transmit Data Buffer |
| U1RBUF | byte | FE42 | Read Only | USART 1 Receive Data Buffer |
| U1ICTRL | byte | FE44 | Read/Write | USART 1 Interrupt Control Register |
| U1STAT | byte | FE46 | Read Only | USART 1 Status Register |
| U1FRS | byte | FE48 | Read/Write | USART 1 Frame Select Register |
| U1MDSL | byte | FE4A | Read/Write | USART 1 Mode Select Register |
| U1BAUD | byte | FE4C | Read/Write | USART 1 Baud Rate Divisor Register |
| U1PSR | byte | FE4E | Read/Write | USART 1 Baud Rate Prescaler |
| MWDAT | byte | FE60 | Read/Write | MICROWIRE Data Register |
| MWCTL | byte | FE62 | Read/Write | MICROWIRE Control Register |
| MWSTAT | word | FE64 | Read/Write | MICROWIRE status Register |
| U2TBUF | byte | FE80 | Read/Write | USART 2 Transmit Data Buffer |
| U2RBUF | byte | FE82 | Read Only | USART 2 Receive Data Buffer |
| U2ICTRL | byte | FE84 | Read/Write | USART 2 Interrupt Control Register |
| U2STAT | byte | FE86 | Read Only | USART 2 Status Register |
| U2FRS | byte | FE88 | Read/Write | USART 2 Frame Select Register |
| U2MDSL | byte | FE8A | Read/Write | USART 2 Mode Select Register |
| ACBSDA | byte | FEC0 | Read/Write | ACB Serial Data Register |
| U2BAUD | byte | FE8C | Read/Write | USART 2 Baud Rate Divisor Register |
| U2PSR | byte | FE8E | Read/Write | USART 2 Baud Rate Prescaler |
| ACBST | byte | FEC2 | Read/Write | ACB Status Register |

**Table 43 Device Detailed Memory Map**

| Register Name | Size | Register Address (hex) | Access Type | Contents |
|---|---|---|---|---|
| ACBCST | byte | FEC4 | Read/Write | ACB Control Status Register |
| ACBCTL1 | byte | FEC6 | Read/Write | ACB Control 1 Register |
| ACBADDR | byte | FEC8 | Read/Write | ACB Own Address Register |
| ACBCTL2 | byte | FECA | Read/Write | ACB Control 2 Register |
| PIALT | byte | FEE0 | Read/Write | Port I Alternate Function Register |
| PIDIR | byte | FEE2 | Read/Write | Port I Direction Register |
| PIDIN | byte | FEE4 | Read Only | Port I Data Input Register |
| PIDOUT | byte | FEE6 | Read/Write | Port I Data Output Register |
| PIWKPU | byte | FEE8 | Read/Write | Port I Weak Pull-Up Register |
| PLALT | byte | FF00 | Read/Write | Port L Alternate Function Register |
| PLDIR | byte | FF02 | Read/Write | Port L Direction Register |
| PLDIN | byte | FF04 | Read Only | Port L Data Input Register |
| PLDOUT | byte | FF06 | Read/Write | Port L Data Output Register |
| PLWKPU | byte | FF08 | Read/Write | Port L Weak Pull-Up Register |
| TWCFG | byte | FF20 | Read/Write | Timer and WATCHDOG Configuration Register |
| TWCP | byte | FF22 | Read/Write | Timer and WATCHDOG Clock Prescaler Register |
| TWMT0 | word | FF24 | Read/Write | TWM Timer 0 Register |
| T0CSR | byte | FF26 | Read/Write | TWMT0 Control and Status Register |
| WDCNT | byte | FF28 | Write Only | WATCHDOG Count Register |
| WDSDM | byte | FF2A | Write Only | WATCHDOG Service Data Match Register |
| T1CNT1 | word | FF40 | Read/Write | T1 Timer/Counter I Register |
| T1CRA | word | FF42 | Read/Write | T1 Reload/Capture A Register |
| T1CRB | word | FF44 | Read/Write | T1 Reload/Capture B Register |
| T1CNT2 | word | FF46 | Read/Write | T1 Timer/Counter II Register |
| T1PRSC | byte | FF48 | Read/Write | T1 Clock Prescaler Register |
| T1CKC | byte | FF4A | Read/Write | T1 Clock Unit Control Register |
| T1CTRL | byte | FF4C | Read/Write | T1 Timer Mode Control Register |
| T1ICTL | byte | FF4E | Read/Write | T1 Timer Interrupt Control Register |
| T1ICLR | byte | FF50 | Read/Write | T1 Timer Interrupt Clear Register |
| T2CNT2 | word | FF60 | Read/Write | T2 Timer/Counter I Register |
| T2CRA | word | FF62 | Read/Write | T2 Reload/Capture A Register |
| T2CRB | word | FF64 | Read/Write | T2 Reload/Capture B Register |
| T2CNT2 | word | FF66 | Read/Write | T2 Timer/Counter II Register |
| T2PRSC | byte | FF68 | Read/Write | T2 Clock Prescaler Register |
| T2CKC | byte | FF6A | Read/Write | T2 Clock Unit Control Register |
| T2CTRL | byte | FF6C | Read/Write | T2 Timer Mode Control Register |
| T2ICTL | byte | FF6E | Read/Write | T2 Timer Interrupt Control Register |
| T2ICLR | byte | FF70 | Read/Write | T2 Timer Interrupt Clear Register |
| MODE | word | FF80 | Read/Write | Mode Control Register |
| IO1CTL | word | FF82 | Read/Write | I/O Control Register 1 |
| IO2CTL | word | FF84 | Read/Write | I/O Control Register 2 |
| INTCTL | word | FF86 | Read/Write | Interrupt Control Register |
| INTPND | word | FF88 | Read/Write | Interrupt Pending Register |
| CLK1PS | word | FF8A | Read/Write | Clock Prescaler Register 1 |
| COUNT1 | word | FF8C | Read/Write | Counter Register 1 |

**Table 43 Device Detailed Memory Map**

| Register Name | Size | Register Address (hex) | Access Type | Contents |
|---|---|---|---|---|
| PERCAP1 | word | FF8E | Read/Write | Period/Capture Register 1 |
| DTYCAP1 | word | FF90 | Read/Write | Duty Cycle/Capture Register 1 |
| COUNT2 | word | FF92 | Read/Write | Count Register 2 |
| PERCAP2 | word | FF94 | Read/Write | Period/Capture Register 2 |
| DTYCAP2 | word | FF96 | Read/Write | Duty Cycle/Capture Register 2 |
| CLK2PS | word | FF98 | Read/Write | Clock Prescaler Register 2 |
| COUNT3 | word | FF9A | Read/Write | Count Register 3 |
| PERCAP3 | word | FF9C | Read/Write | Period/Capture Register 3 |
| DTYCAP3 | word | FF9E | Read/Write | Duty Cycle/Capture Register 3 |
| COUNT4 | word | FFA0 | Read/Write | Count Register 4 |
| PERCAP4 | word | FFA2 | Read/Write | Period/Capture Register 4 |
| DTYCAP4 | word | FFA4 | Read/Write | Duty Cycle/Capture Register 4 |
| ADCST | byte | FFC0 | Read/Write | A/D Converter Status Register |
| ADCCNT1 | byte | FFC2 | Read/Write | A/D Converter Control 1 Register |
| ADCCNT2 | byte | FFC4 | Read/Write | A/D Converter Control 2 Register |
| ADCCNT3 | byte | FFC6 | Read/Write | A/D Converter Control 3 Register |
| ADDATA0 | byte | FFCA | Read Only | A/D Converter Data 0 Register |
| ADDATA1 | byte | FFCC | Read Only | A/D Converter Data 1 Register |
| ADDATA2 | byte | FFCE | Read Only | A/D Converter Data 2 Register |
| ADDATA3 | byte | FFD0 | Read Only | A/D Converter Data 3 Register |
| CMPCTRL | byte | FFE0 | Read/Write | Analog Comparator Control/Status Register |

## 24.0 Register Layouts

The following tables show the functions of the bit fields of the device registers. For more information on using these registers, see the detailed description of the applicable function elsewhere in this data sheet.

### 24.1 REGISTER LAYOUT

| CAN Memory Registers | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CMBn.ID1 | XI28 ID10 | XI27 ID9 | XI26 ID8 | XI25 ID7 | XI24 ID6 | XI23 ID5 | XI22 ID4 | XI21 ID3 | XI20 ID2 | XI19 ID1 | XI18 ID0 | SRR RTR | IDE | XI17 | XI16 | XI15 |
| CMBn.ID0 | XI14 | XI13 | XI12 | XI11 | XI10 | XI9 | XI8 | XI7 | XI6 | XI5 | XI4 | XI3 | XI2 | XI1 | XI0 | RTR |
| CMBn.DATA0 | Data 1.7 | Data 1.6 | Data 1.5 | Data 1.4 | Data 1.3 | Data 1.2 | Data 1.1 | Data 1.0 | Data 2.7 | Data 2.6 | Data 2.5 | Data 2.4 | Data 2.3 | Data 2.2 | Data 2.1 | Data 2.0 |
| CMBn.DATA1 | Data 3.7 | Data 3.6 | Data 3.5 | Data 3.4 | Data 3.3 | Data 3.2 | Data 3.1 | Data 3.0 | Data 4.7 | Data 4.6 | Data 4.5 | Data 4.4 | Data 4.3 | Data 4.2 | Data 4.1 | Data 4.0 |
| CMBn.DATA2 | Data 5.7 | Data 5.6 | Data 5.5 | Data 5.4 | Data 5.3 | Data 5.2 | Data 5.1 | Data 5.0 | Data 6.7 | Data 6.6 | Data 6.5 | Data 6.4 | Data 6.3 | Data 6.2 | Data 6.1 | Data 6.0 |
| CMBn.DATA3 | Data 7.7 | Data 7.6 | Data 7.5 | Data 7.4 | Data 7.3 | Data 7.2 | Data 7.1 | Data 7.0 | Data 8.7 | Data 8.6 | Data 8.5 | Data 8.4 | Data 8.3 | Data 8.2 | Data 8.1 | Data 8.0 |
| CMBn.TSTP | TSTP15 | TSTP14 | TSTP13 | TSTP12 | TSTP11 | TSTP10 | TSTP9 | TSTP8 | TSTP7 | TSTP6 | TSTP5 | TSTP4 | TSTP3 | TSTP2 | TSTP1 | TSTP0 |
| CMBn.CNT-STAT | DLC3 | DLC2 | DLC1 | DLC0 | Reserved | | | | PRI3 | PRI2 | PRI1 | PRI0 | ST3 | ST2 | ST1 | ST0 |

| CAN Control/ Status | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CGCR | Reserved | | | | EIT | DIAGEN | INTERNAL | LOOPBACK | IGNACK | LO | DDIR | TSTPEN | BUFFLOCK | CRX | CTX | CANEN |
| CTIM | PSC[6:0] | | | | | | | SJW[1:0] | | TSEG1[3:0] | | | | TSEG2[2:0] | | |
| GMSKB | GM[28:18] | | | | | | | | | | | RTR | IDE | GM[17:15] | | |
| GMSKX | GM[14:0] | | | | | | | | | | | | | | | XRTR |
| BMSKB | BM[28:18] | | | | | | | | | | | RTR | IDE | BM[17:15] | | |
| BMSKX | BM[14:0] | | | | | | | | | | | | | | | XRTR |
| CIEN | EIEN | IEN[14:0] | | | | | | | | | | | | | | |
| CIPND | EIPND | IPND[14:0] | | | | | | | | | | | | | | |
| CICLR | EICLR | ICLR[14:0] | | | | | | | | | | | | | | |
| CICEN | EICEN | ICEN[14:0] | | | | | | | | | | | | | | |
| CSTPND | Reserved | | | | | | | | NS[2:0] | | | IRQ | IST[3:0] | | | |
| CANEC | REC[7:0] | | | | | | | | TEC[7:0] | | | | | | | |
| CEDIAG | Reserved | DRIVE | MON | CRC | STUFF | TXE | EBID[5:0] | | | | | | EFID[3:0] | | | |
| CTMR | CTMR[15:0] | | | | | | | | | | | | | | | |

| System Configuration Registers | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| MCFG | Reserved | CLK2OE | Reserved | | CLK1OE | CLKOE | Reserved |
| MSTAT | Reserved | | | PGMBUSY | Reserved | OENV1 | OENV0 |

| **BIU Registers** | **15 12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BCFG | Reserved | | | | | | | | | | | | EWR |
| IOCFG | Reserved | | | IPST | Res | BW | Reserved | | HOLD | | WAIT | | |
| SZCFG0 | Reserved | FRE | IPRE | IPST | Res | BW | Reserved | | HOLD | | WAIT | | |
| SZCFG1 | Reserved | FRE | IPRE | IPST | Res | BW | Reserved | | HOLD | | WAIT | | |

| **ISP Registers** | **15 13** | **12 10** | **9 8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FLCTRL1 | Reserved | | | | BOOTAREA | | | | | | |
| FLCTRL2 | EMPTY | Reserved | | CODEAREA | | | | | | | |
| FLSEC | Reserved | | | | FROMWR | | | FROMRD | | | |
| ISPKEY | Reserved | | | | ISPKYVAL | | | | | | |

| **Flash Data Memory Registers** | **15** | **14 10** | **9 8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DMCSR | Reserved | | | | | | | ERASE | DMBUSY | ZEROWS | Reserved |
| DMPSLR | Reserved | | | FTDIV | | | | | | | |
| DMSTART | Reserved | | | FTSTART | | | | | | | |
| DMTRAN | Reserved | | | FTTRAN | | | | | | | |
| DMPROG | Reserved | | | FTPROG | | | | | | | |
| DMERASE | Reserved | | | FTER | | | | | | | |
| DMEND | Reserved | | | FTEND | | | | | | | |
| DMPCNT | Reserved | | | FTPCNT | | | | | | | |
| DMCNT | Reserved | | FTCNT | | | | | | | | |
| DMKEY | Reserved | | | DMKEYVAL | | | | | | | |

| **Flash EEPROM Program Memory Registers** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| FLCSR | MERASE | Reserved | | | PMLFULL | PMBUSY | PMER | Reserved |
| FLPSLR | FTDIV | | | | | | | |
| FLSTART | FTSTART | | | | | | | |
| FLTRAN | FTTRAN | | | | | | | |

| | | |
|---|---|---|
| FLPROG | FTPROG | |
| FLERASE | FTER | |
| FLEND | FTEND | |
| FLPCNT | FTPCNT | |
| FLCNT1 | FTCNTL (0:7) | |
| FLCNT2 | Reserved | FTCNTL (8:9) |
| PGMKEY | PMKEYVAL | |

| **GPIO Registers** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| PxALT | Px Pins Alternate Function Enable | | | | | | | |
| PxDIR | Px Port Direction | | | | | | | |
| PxDIN | Px Port Output Data | | | | | | | |
| PxDOUT | Px Port Input Data | | | | | | | |
| PxWPU | Px Port Weak Pull-up Enable | | | | | | | |

| **ICU31L Registers** | **15 12** | **11 8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|
| IVCT | Reserved | | 0 | 0 | INTVECT | | | | | |
| NMISTAT | Reserved | | | | | | | | | EXT |
| EXNMI | Reserved | | | | | | | ENLCK | PIN | EN |
| ISTAT0 | IST(15:0) | | | | | | | | | |
| ISTAT1 | IST(31:16) | | | | | | | | | |
| IENAM0 | IENA(15:0) | | | | | | | | | |
| IENAM1 | IENA(31:16) | | | | | | | | | |
| IDBG | Reserved | IRQVECT | | | INTVECT | | | | | |

| **MIWU16 Registers** | **15 14** | **13 12** | **11 10** | **9 8** | **7 6** | **5 4** | **3 2** | **1 0** |
|---|---|---|---|---|---|---|---|---|
| WKEDG | WKED | | | | | | | |
| WKENA | WKEN | | | | | | | |
| WKICTL1 | WKINTR7 | WKINTR6 | WKINTR5 | WKINTR4 | WKINTR3 | WKINTR2 | WKINTR1 | WKINTR0 |
| WKICTL2 | WKINTR15 | WKINTR14 | WKINTR13 | WKINTR12 | WKINTR11 | WKINTR10 | WKINTR9 | WKINTR8 |
| WKPND | WKPD | | | | | | | |
| WKPCL | WKCL | | | | | | | |

| Dual Clock + Reset Registers | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| CRCTRL | Reserved | | | | | | POR | SCLK |
| PRSSC | SCDIV | | | | | | | |
| PRSSC1 | SCDIV2 | | | | SCDIV1 | | | |

| Power Management Register | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| PMCSR | OLFC | OHFC | WBPSM | Reserved | HALT | IDLE | DHF | PSM |

| USART Registers | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| UnTBUF | UnTBUF | | | | | | | |
| UnRBUF | UnRBUF | | | | | | | |
| UnICTRL | UnEEI | UnERI | UnETI | Reserved | | | UnRBF | UnTBE |
| UnSTAT | Reserved | UnXMIP | UnRB9 | UnBKD | UnERR | UnDOE | UnFE | UnPE |
| UnFRS | Reserved | UnPEN | UnPSEL | | UnXB9 | UnSTP | UnCHAR | |
| UnMDSL | Reserved | | | | UnCKS | UnBRK | UnATN | UnMOD |
| UnBAUD | UnDIV[7]: UnDIV[0] | | | | | | | |
| UnPSR | UnPSC | | | | | UnDIV[10]: UnDIV[8] | | |

| MWSPI16 Registers | 15 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| MWDAT | MWDAT | | | | | | | | | |
| MWCTL | MCDV | MIDL | MSKM | MEIW | MEIR | MEIO | MECH | MMOD | MMNS | MEN |
| MWSTAT | Reserved | | | | | | | MOVR | MRBF | MBSY |

| ACB Registers | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| ACBSDA | DATA | | | | | | | |
| ACBST | SLVSTP | SDAST | BER | NEGACK | STASTR | NMATCH | MASTER | XMIT |
| ACBCST | Reserved | | TGSCL | TSDA | GMATCH | MATCH | BB | BUSY |
| ACBCTL1 | STASTRE | NMINTE | GCMEN | ACK | Reserved | INTEN | STOP | START |
| ACBADDR | SAEN | ADDR | | | | | | |
| ACBCTL2 | SCLFRQ | | | | | | | ENABLE |

| TIMER Registers | 15 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| TnCNT1 | TnCNT1 | | | | | | | | |
| TnCRA | TnCRA | | | | | | | | |
| TnCRB | TnCRB | | | | | | | | |
| TnCNT2 | TnCNT2 | | | | | | | | |
| TnPRSC | | Reserved | | | CLKPS | | | | |
| TnCKC | | Reserved | | C2CSEL | | | C1CSEL | | |
| TnCTRL | | Reserved | TnAOUT | TnBEN | TnAEN | TnBEDG | TnAEDG | MDSEL | |
| TnICTL | | TnDIEN | TnCIEN | TnBIEN | TnAIEN | TnDPND | TnCPND | TnBPND | TnAPND |
| TnICLR | | Reserved | | | | TnDCLR | TnCCLR | TnBCLR | TnACLR |

| VTU Registers | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODE | TMOD4 | | T8RUN | T7RUN | TMOD3 | | T6RUN | T5RUN | TMOD2 | | T4RUN | T3RUN | TMOD1 | | T2RUN | T1RUN |
| IO1CTL | P4POL | C4EDG | | | P3POL | C3EDG | | | P2POL | C2EDG | | | P1POL | C1EDG | | |
| IO2CTL | P8POL | C8EDG | | | P7POL | C7EDG | | | P6POL | C6EDG | | | P5POL | C5EDG | | |
| INTCTL | I4DEN | I4CEN | I4BEN | I4AEN | I3DEN | I3CEN | I3BEN | I3AEN | I2DEN | I2CEN | I2BEN | I2AEN | I1DEN | I1CEN | I1BEN | I1AEN |
| INTPND | I4DPD | I4CPD | I4BPD | I4APD | I3DPD | I3CPD | I3BPD | I3APD | I2DPD | I2CPD | I2BPD | I2APD | I1DPD | I1CPD | I1BPD | I1APD |
| CLK1PS | C2PRSC | | | | | | | | C1PRSC | | | | | | | |
| COUNT1 | CNT1 | | | | | | | | | | | | | | | |
| PERCAP1 | PCAP1 | | | | | | | | | | | | | | | |
| DTYCAP1 | DCAP1 | | | | | | | | | | | | | | | |
| COUNT2 | CNT2 | | | | | | | | | | | | | | | |
| PERCAP2 | PCAP2 | | | | | | | | | | | | | | | |
| DTYCAP2 | DCAP2 | | | | | | | | | | | | | | | |
| CLK2PS | C4PRSC | | | | | | | | C3PRSC | | | | | | | |
| COUNT3 | CNT3 | | | | | | | | | | | | | | | |
| PERCAP3 | PCAP3 | | | | | | | | | | | | | | | |
| DTYCAP3 | DCAP3 | | | | | | | | | | | | | | | |
| COUNT4 | CNT4 | | | | | | | | | | | | | | | |
| PERCAP4 | PCAP4 | | | | | | | | | | | | | | | |
| DTYCAP4 | DCAP4 | | | | | | | | | | | | | | | |

| TWM Registers | 15 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| TWCFG | | Reserved | | WDSDME | WDCT0I | LWDCNT | LTWMT0 | LTWCP | LTWCFG |
| TWCP | | Reserved | | | | | MDIV | | |
| TIMER0 | PRESET | | | | | | | | |
| T0CSR | | Reserved | | | | | T0INTE | TC | RST |
| WDCNT | | PRESET | | | | | | | |
| WDSDM | | RSTDATA | | | | | | | |

| A/D Registers | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| ADCST | Reserved | | BUFPTR | | Reserved | OVF | BUSY | EOC |
| ADCCNT1 | Reserved | | | | Reserved | INTE | Reserved | ADCEN |
| ADCCNT2 | START | SCAN | | CONT | CHANNEL | | | |
| ADCCNT3 | Reserved | PWREN | DELAY | | CDIV | | | |
| ADDATA0 | RESULT 1 DATA | | | | | | | |
| ADDATA1 | RESULT 2 DATA | | | | | | | |
| ADDATA2 | RESULT 3 DATA | | | | | | | |
| ADDATA3 | RESULT 4 DATA | | | | | | | |

| Analog Comp. Registers | 7 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CMPCTRL | Reserved | CMP2OE | CMP1OE | CMP2EN | CMP1EN | CMP2RD | CMP1RD |

# 25.0 ELECTRICAL AND THERMAL CHARACTERISTICS

## Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

| | |
|---|---|
| Supply Voltage ($V_{CC}$) | 7V |
| Voltage at Any Pin * | –0.6V to $V_{CC}$ +0.6V |
| ESD Protection Level | 2 kV (Human Body Model) |
| Total Current into $V_{CC}$ Pin (Source) | 200 mA |
| Total Current out of GND Pin (Sink) | 200 mA |
| Storage Temperature Range | –65°C to +150°C |

*Note: Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings. * The latch-up tolerance on Access Bus pins 14 and 15 exceeds 150mA.*

## Thermal Characteristics

| Characteristics | Symbol | Value | Unit |
|---|---|---|---|
| Average junction temperature | $T_J$ | $T_A + (P_D \times \Theta_{JA})$ | °C |
| Ambient temperature | $T_A$ | User-determined | °C |
| Package thermal resistance (junction-to-ambient) 80-pin quad flat pack (QFP) | $\Theta_{JA}$ | 49.8 | °C/W |
| Total power dissipation[1] | $P_D$ | $P_{INT} + P_{I/O}$ or $\frac{K}{T_J + 273°C}$ | W |
| Device internal power dissipation | $P_{INT}$ | $I_{DD} \times V_{DD}$ | W |
| I/O pin power dissipation[2] | $P_{I/O}$ | User-determined | W |
| A constant[3] | K | $P_D \times (T_A + 273°C) + \Theta_{JA} \times P_D^2$ | W, °C |

1. This is an approximate value, neglecting $P_{I/O}$.

2. For most applications $P_{I/O} << P_{INT}$ and can be neglected.

3. K is a constant pertaining to the device. Solve for K with a known $T_A$ and a measured $P_D$ (at equilibrium). Use this value of K to solve for $P_D$ and $T_J$ iterntively for any value of $T_A$.

## DC Electrical Characteristics: –40°C ≤ TA ≤ +85°C (also supports -40°C to +125°C)

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| | Operating Voltage | | 4.5 | 5.5 | V |
| | | | | | |
| $V_{IH}$ | Logical 1 CMOS Input Voltage (except ACB & clocks) | | 0.8Vcc | Vcc + 0.5 | V |
| $V_{IL}$ | Logical 0 CMOS Input Voltage (except ACB & clocks) | | -0.5 | 0.2Vcc | V |
| $V_{IHACB}$ | SDA, SCL Logical 1 CMOS Input Voltage | | 0.7Vcc | | V |
| $V_{ILACB}$ | SDA, SCL Logical 0 CMOS Input Voltage | | | 0.3Vcc | V |
| Vxl | Low Level Input Voltage OSC | External X1 clock | 0 | 0.2Vcc | V |
| Vxh | High Level Input Voltage OSC | External X1 clock | 0.5Vcc | Vcc | V |
| Vxl2 | X2CKI Logical 0 Input Voltage | External X2 clock | | 0.3 | V |
| Vxh2 | X2CKI Logical 1 Input Voltage | External X2clock | 1.2 | | V |
| $V_{hys}$ | Hysteresis Loop Width [a] | | 0.1Vcc | | V |
| $I_{OH}$ | Logical 1 CMOS Output Current | $V_{OH}$ = 3.8V, Vcc=4.5V | -1.6 | | mA |
| $I_{OL}$ | Logical 0 CMOS Output Current | $V_{OL}$ = 0.45V, Vcc=4.5V | 1.6 | | mA |

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| $I_{OLACB}$ | SDA, SCL Logical 0 CMOS Output Current | $V_{OL}$ = 0.4V, Vcc=4.5V | 3.0 | | mA |
| $I_{OHW}$ | Weak Pull-up Current | $V_{OH}$ = 3.8V, Vcc=4.5V | -10 | | µA |
| $I_{IL}$ | RESET pin Weak Pull-down Current | $V_{IL}$ = 0.9V, Vcc=4.5V | | 0.4 | µA |
| $I_L$ | High Impedance Input Leakage Current | 0V ≤ Vin ≤ Vcc | - 2.0 | 2.0[j] | µA |
| $I_O$(Off) | Output Leakage Current (I/O pins in input mode) | 0V ≤ Vout ≤ Vcc | - 2.0 | 2.0[j] | µA |
| Icca1 | Digital Supply Current Active Mode [b] | Vcc= 5.5V | | 95 | mA |
| Iccprog | Digital Supply Current Active Mode [c] | Vcc= 5.5V | | 115 | mA |
| Icca2 | Digital Supply Current Active Mode [d] | Vcc = 5.5V | | 58 | mA |
| Iccps | Digital Supply Current Power Save Mode [e] | Vcc= 5.5V | | 9 | mA |
| Iccid | Digital Supply Current Idle Mode [f] | Vcc = 5.5V | | 200 | µA |
| Iccq | Digital Supply Current Halt Mode [f] | Vcc = 5.5V | | 20[k] | µA |
| Iacc | Analog Supply Current Active Mode [g] | Vcc= 5.5V | | 3 | mA |

a. Guaranteed by design
b. Run from internal memory, Iout=0mA, X1CKI=20MHz, not programming flash memory
c. Same conditions as Icca1 but programming or erasing one of the flash memory arrays
d. CPU executing an WAIT instruction, Iout=0mA, X1CKI=20MHz, peripherals not active
e. Running from internal memory, Iout=0mA, X1CKI=20MHz, X2CKI=32.768kHz
f. Iout=0mA, X1CKI=Vcc, X2CKI=32.768kHz
g. ADC and analog comparators enabled
j. $I_L$ adn $I_O$ are 2.0 µA at 85°C and 5.0 µA at 125°C
k. $I_{acq}$ is 20 µA at 85°C and 50 µA at 125°C

## A/D Converter Characteristics

$V_{CC}$ = 5V, TA = 25°C

| Symbol | Parameter | Conditions[a] | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $N_{IL}$ | Integral Error[b] | $V_{REF}$ = $V_{CC}$ | | | ±0.5 | LSB |
| $N_{DL}$ | Differential Error[c] | $V_{REF}$ = $V_{CC}$ | | | ±1.0 | LSB |
| $V_{ABSOLUTE}$ | Absolute Error | $V_{REF}$ = $V_{CC}$ | | | ±1.5 | LSB |
| $V_{IN}$ | Input Voltage Range | $V_{REF}$ < $V_{CC}$ - 0.1 | 0 | | $V_{REF}$ | V |
| $V_{REFEX}$ | External Reference Voltage | | 3.0 | | $V_{DD}$ | V |
| $I_{VREF}$ | $V_{REF}$ input current | $V_{REF}$ = 5V | | | 1.2 | mA |
| $I_{AL}$ | Analog input leakage current | $V_{REF}$ = $V_{CC}$ | | | ±1 | µA |
| $R_{AIN}$ | Analog input resistance[d] | | | | 200 | Ω |
| $C_{AIN}$ | Analog input capacitance[e] | | | | 5 | pF |
| $t_{ADCCLK}$ | Conversion Clock period | | | | 500 | ns |
| $C_{REFEX}$ | External Vref bypass capacitance | | 0.47 | | | µF |
| $t_{ACT}$ | First conversion after Vcc stable | | 30 | | | µs |
| $M_{MONOTONIC}$ | MONOTONICITY[f] | | GUARANTEED | | | |

a. All parameters specified for $f_{OSC}$ =2 MHz, $V_{DD}$ = 5.0V ± 10% unless otherwise noted.
b. Integral (Non-linearity) Error — The maximum difference between the best-fit straight line reference and the actual conversion curves.
c. Differential (Non-linearity) Error — The maximum difference between the best-fit step size of 1 LSB and any actual step size.
d. The resistance between the device input and the internal analog input capacitance.

e. The input signal is measured across the internal capacitance.
f. Conversion result never decreases with an increase in input voltage and has no missing codes.

## Analog Comparator Characteristics

| Symbol | Parameter | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $V_{OS}$ | Input Offset Voltage | Vcc = 5V,<br>$0.4V \le V_{IN} \le V_{CC} - 1.5V$ | | | ±25 | mV |
| $V_{CM}$ | Input Common Mode Voltage Range | | 0.4 | | $V_{CC}$ -1.5 | V |
| $I_{CS}$ | DC Supply Current per Comparator (When Enabled) | $V_{CC}$=5.5V | | | 250 | µA |
| | Response Time | 1V Step / 100mV Overdrive | | | 1 | µs |

## Flash EEPROM Program Memory Programming

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| $t_{PWP}$ | Programming pulse width [a] | | 30 | 40 | µs |
| $t_{EWP}$ | Erase pulse width[b] | | 1 | - | ms |
| $t_{SDP}$ | Charge pump power-up delay[c] | | 10 | - | µs |
| $t_{TTP}$ | Program/erase transition time[d] | | 5 | - | µs |
| $t_{PAH}$ | Programming address hold, new address setup time | | 2 | - | clock cycles |
| $t_{PEP}$ | Charge pump enable hold time | | 1 | - | clock cycles |
| $t_{EDP}$ | Charge pump power hold time[e] | | 5 | | µs |
| $t_{CHVP}$ | Cumulative program high voltage period for each row after erase.[f] | | - | 25 | ms |
| | Data retention | | 100 | - | years |
| | | | - | 100K | cycles |

a. The programming pulse width is determined by the following equation:
$t_{PWP} = T_{clk}$ x (FTDIV+1) x (FTPROG+1), where $T_{clk}$ is the system clock period, FTDIV is the contents of the FLPSLR register and FTPROG is the contents of the FLPROG register.

b. The erase pulse width is determined by the following equation:
$t_{EWP} = T_{clk}$ x (FTDIV+1) x 4 x (FTER+1), where $T_{clk}$ is the system clock period, FTDIV is the contents of the FLPSLR register and FTER is the contents of the FLERASE register.

c. The program/erase start delay time is determined by the following equation:
$t_{SDP} = T_{clk}$ x (FTDIV+1) x (FTSTART+1), where $T_{clk}$ is the system clock period, FTDIV is the contents of the FLPSLR register and FTSTART is the contents of the FLSTART register.

d. The program/erase transition time is determined by the following equation:
$t_{TTP} = T_{clk}$ x (FTDIV+1) x (FTTRAN+1), where $T_{clk}$ is the system clock period, FTDIV is the contents of the FLPSLR register and FTTRAN is the contents of the FLTRAN register.

e. The program/erase end delay time is determined by the following equation:
$t_{EDP} = T_{clk}$ x (FTDIV+1) x (FTEND+1), where $T_{clk}$ is the system clock period, FTDIV is the contents of the FLPSLR register and FTEND is the contents of the FLEND register.

f. Cumulative program high voltage period for each row after erase $t_{CHVP}$ is the accumulated duration a flash cell is exposed to the programming voltage after the last erase cycle. It is the sum of all $t_{HV}$ after the last erase.

## Flash EEPROM Data Programming

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| | re-programming time[a] | | 1.32 | - | ms |
| $t_{PWD}$ | Programming pulse width [b] | | 30 | 40 | µs |
| $t_{EWD}$ | Erase pulse width[c] | | 1 | - | ms |
| $t_{SDD}$ | Charge pump power-up time[d] | | 10 | - | µs |
| $t_{TTD}$ | Program/erase transition time[e] | | 5 | - | µs |
| $t_{PED}$ | Charge pump enable hold time | | 1 | - | clock cycles |
| $t_{EDD}$ | Charge pump power hold time[f] | | 5 | - | µs |
| | Write/erase endurance (high endurance) | | 100,000 | - | cycles |
| | Write/erase endurance (low endurance) | | 25,000 | - | cycles |
| | Data retention | | 100 | - | years |

a. One re-programming cycle involves one erase pulse followed by programming of four bytes.

b. The programming pulse width is determined by the following equation:
$t_{PWD} = T_{clk} \times (FTDIV+1) \times (FTPROG+1)$, where $T_{clk}$ is the system clock period, FTDIV is the contents of the DMPSLR register and FTPROG is the contents of the DMPROG register.

c. The erase pulse width is determined by the following equation:
$t_{EWD} = T_{clk} \times (FTDIV+1) \times 4 \times (FTER+1)$, where $T_{clk}$ is the system clock period, FTDIV is the contents of the DMPSLR register and FTER is the contents of the DMERASE register.

d. The program/erase start delay time is determined by the following equation:
$t_{SDD} = T_{clk} \times (FTDIV+1) \times (FTSTART+1)$, where $T_{clk}$ is the system clock period, FTDIV is the contents of the DMPSLR register and FTSTART is the contents of the DMSTART register.

e. The program/erase transition time is determined by the following equation:
$t_{TTD} = T_{clk} \times (FTDIV+1) \times (FTTRAN+1)$, where $T_{clk}$ is the system clock period, FTDIV is the contents of the DMPSLR register and FTTRAN is the contents of the DMTRAN register.

f. The program/erase end delay time is determined by the following equation:
$t_{EDD} = T_{clk} \times (FTDIV+1) \times (FTEND+1)$, where $T_{clk}$ is the system clock period, FTDIV is the contents of the DMPSLR register and FTEND is the contents of the DMEND register.

## Flash EEPROM ISP-Memory Programming

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| $t_{PWI}$ | Programming pulse with[a] | | 30 | 40 | µs |
| $t_{EWI}$ | Erase pulse width[b] | | 1 | - | ms |
| | Data retention | | 100 | - | years |
| | | | - | 100K | cycles |

a. Programming timing is controlled by the flash EEPROM data memory interface
b. Erase timing is controlled by the flash EEPROM data memory interface

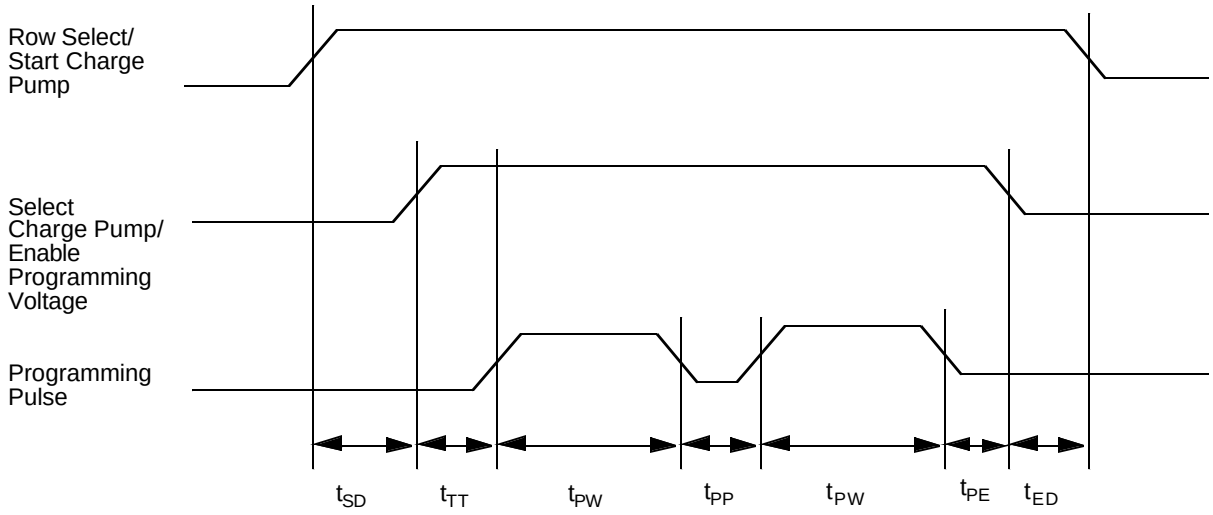



**Figure 76. Flash EEPROM Memory Programming Timing**
**(Sample Sequence for Programming two Words into Flash EEPROM Program Memory**

## Output Signal Levels

All output signals are powered by the digital supply ($V_{CC}$).

Table44 summarizes the states of the output signals during the reset state (when $V_{CC}$ power exists in the reset state) and during the Power Save mode.

The $\overline{\text{RESET}}$ and $\overline{\text{NMI}}$ input pins are active during the Power Save mode. In order to guarantee that the Power Save current not exceed 1mA, these inputs must be driven to a voltage lower than 0.5V or higher than $V_{CC}$-0.5V. An input voltage between 0.5V and ($V_{CC}$-0.5V) may result in power consumption exceeding 1 mA.

**Table 44 Output Pins During Reset and Power-Save**

| Signals on a pin | Reset state (with Vcc) | Power Save mode | Comments |
|---|---|---|---|
| PF[0:7] | TRI-STATE | Previous state | I/O ports will maintain their values when entering power-save mode |
| PG[0:7] | TRI-STATE | Previous state | |
| PI[0:7] | TRI-STATE | Previous state | |
| PL[0:7] | TRI-STATE | Previous state | |
| PB[0:7] | TRI-STATE | Previous state | |
| PC[0:7] | TRI-STATE | Previous state | |

### 25.0.1 Timing Waveforms

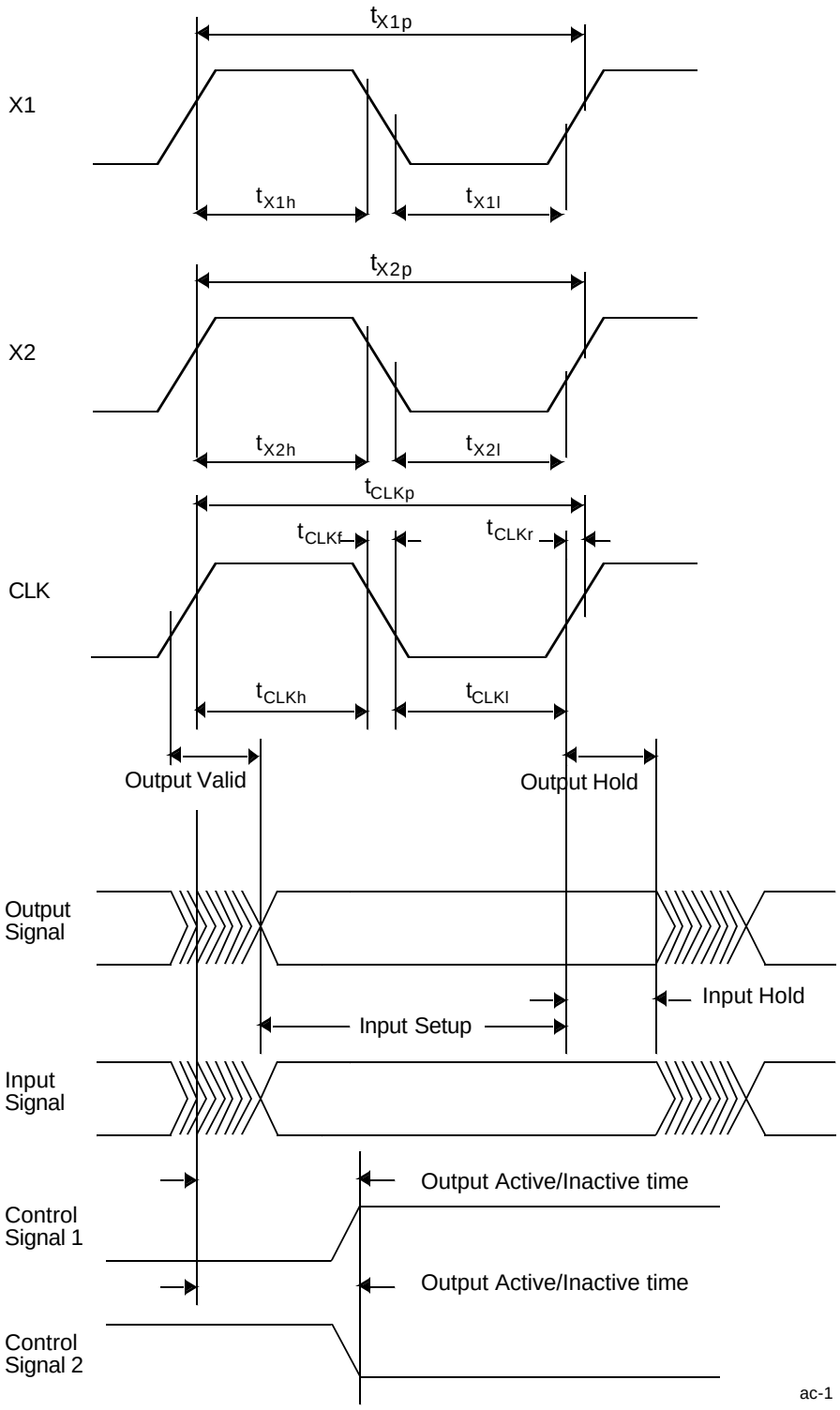



**Figure 77. Clock Waveforms**

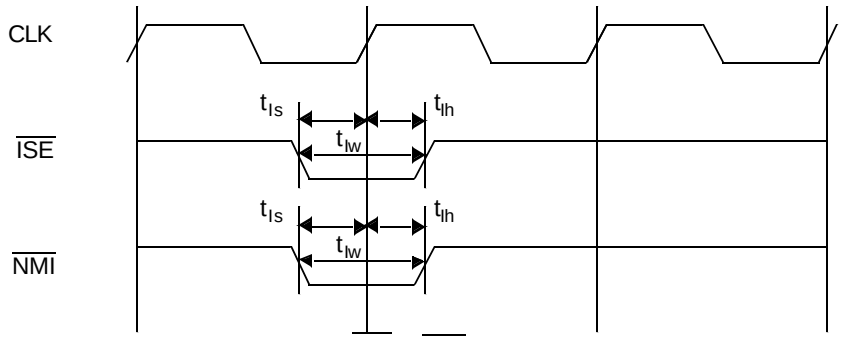



**Figure 78. $\overline{ISE}$ & $\overline{NMI}$ Signal Timing**

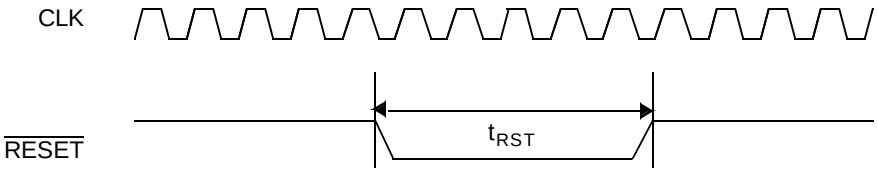



**Figure 79. Non-Power-On Reset**

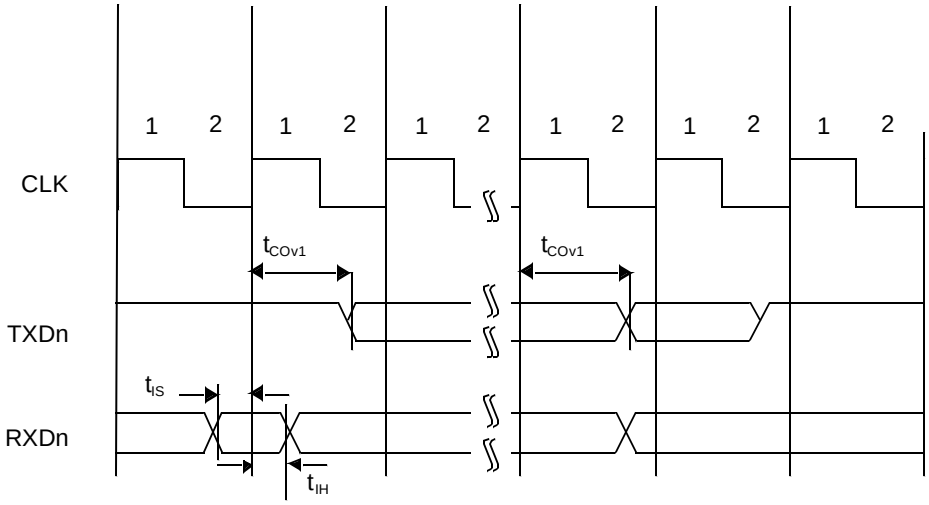



**Figure 80. USART Asynchronous Mode Timing**

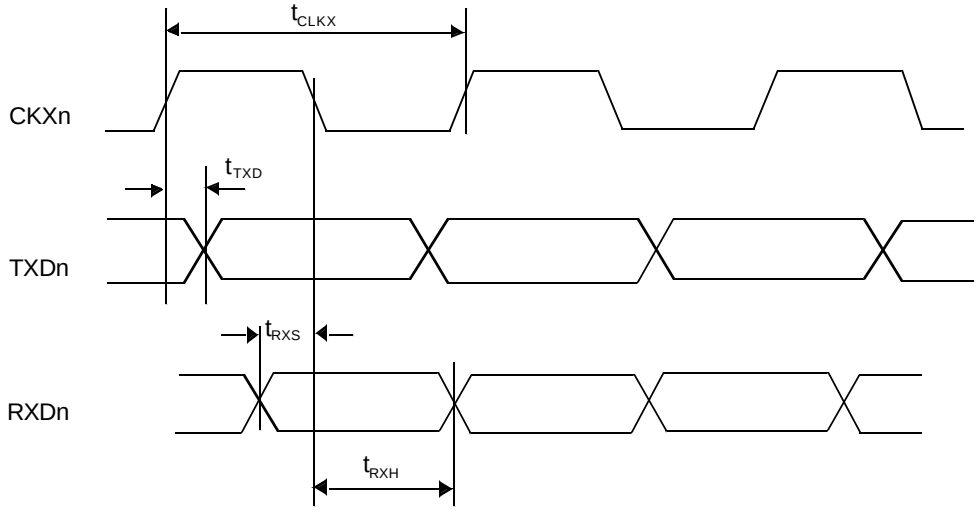



**Figure 81. USART Synchronous Mode Timing**

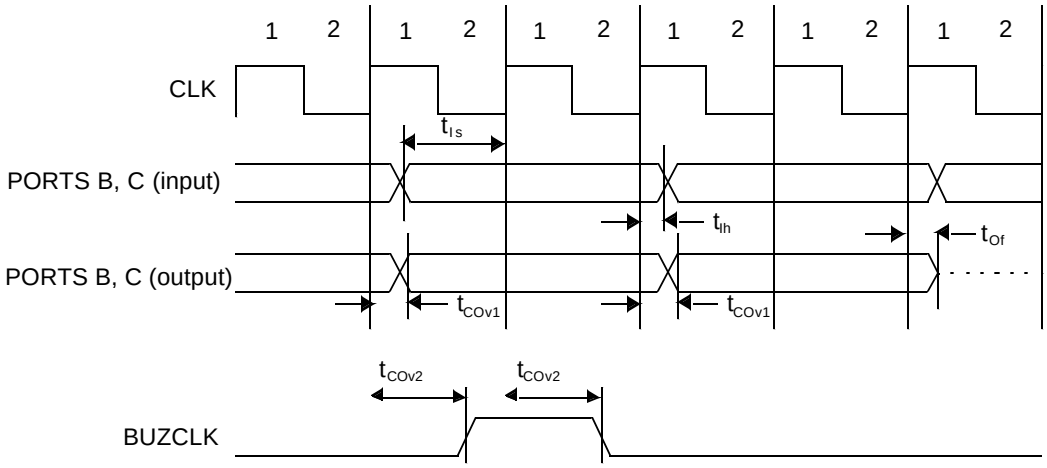



**Figure 82. Port Signals Timing**

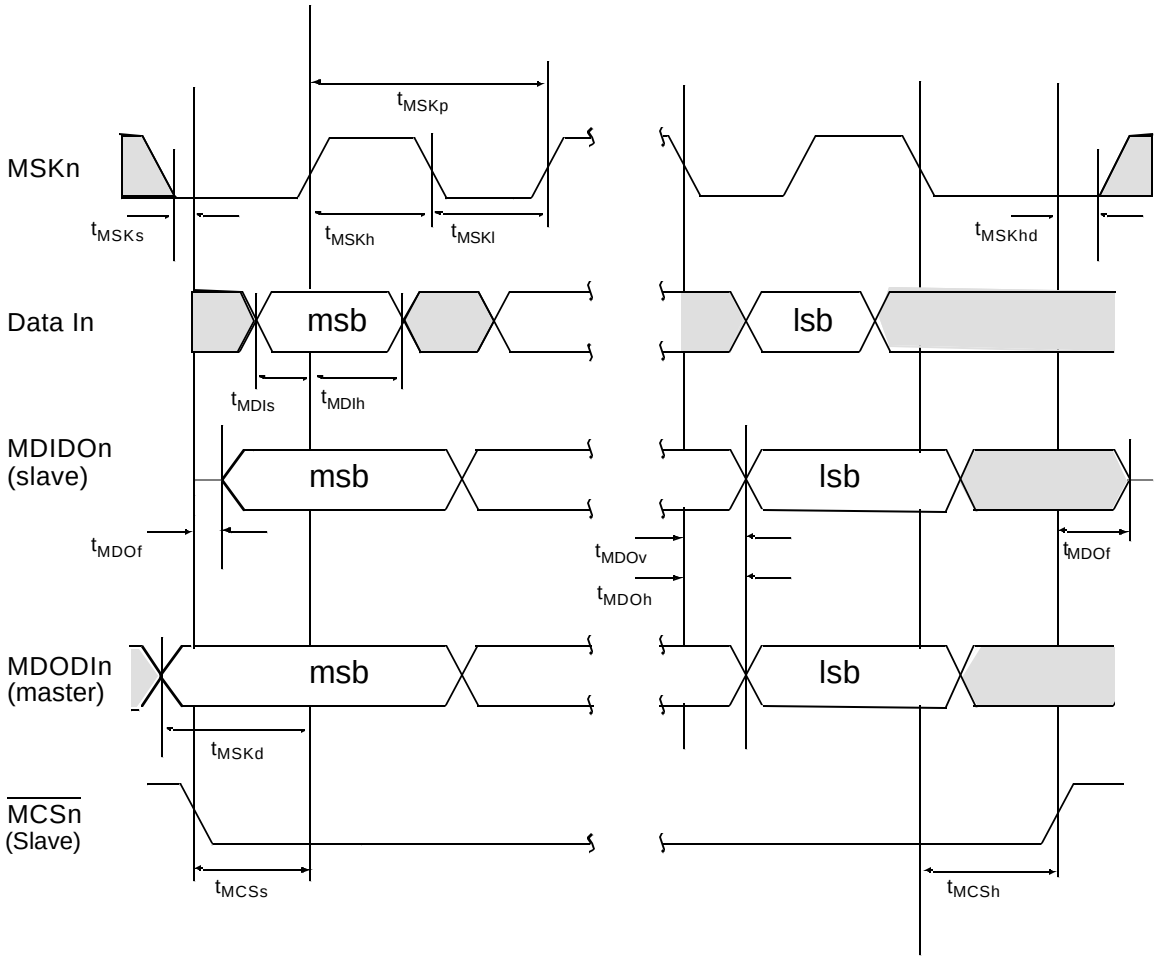



**Figure 83. MICROWIRE Transaction Timing, Normal Mode, MIDL Bit = 0**

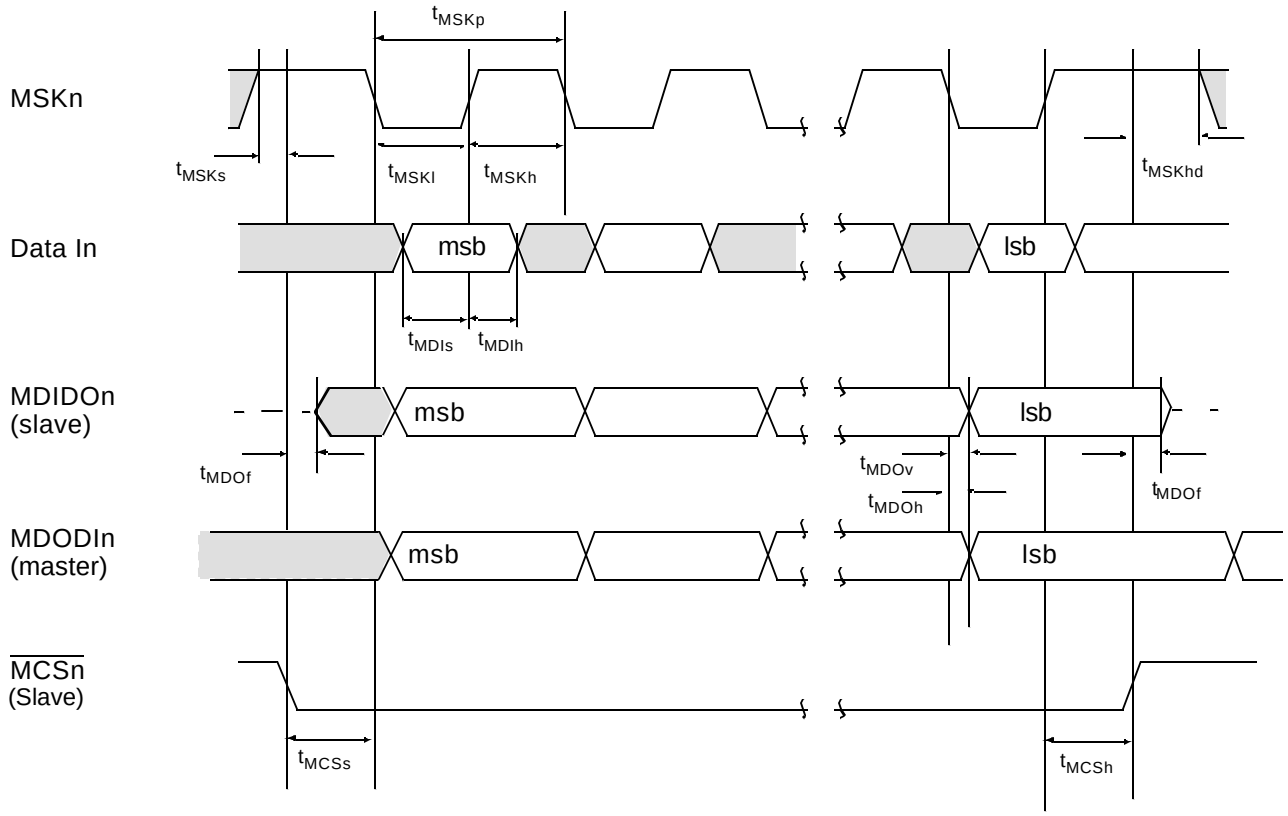



**Figure 84. MICROWIRE Transaction Timing, Normal Mode, MIDL bit = 1**

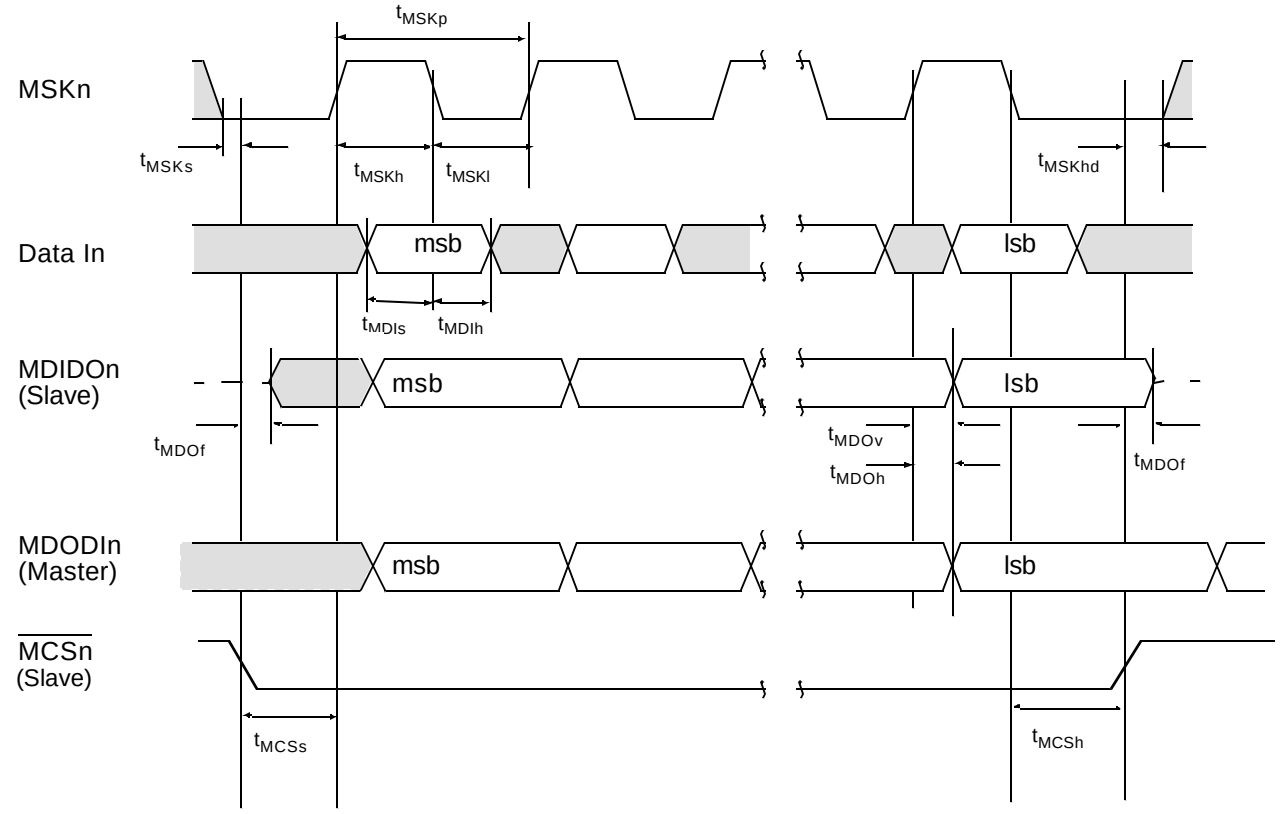



**Figure 85. MICROWIRE Transaction Timing, Alternate Mode, MIDL bit = 0**

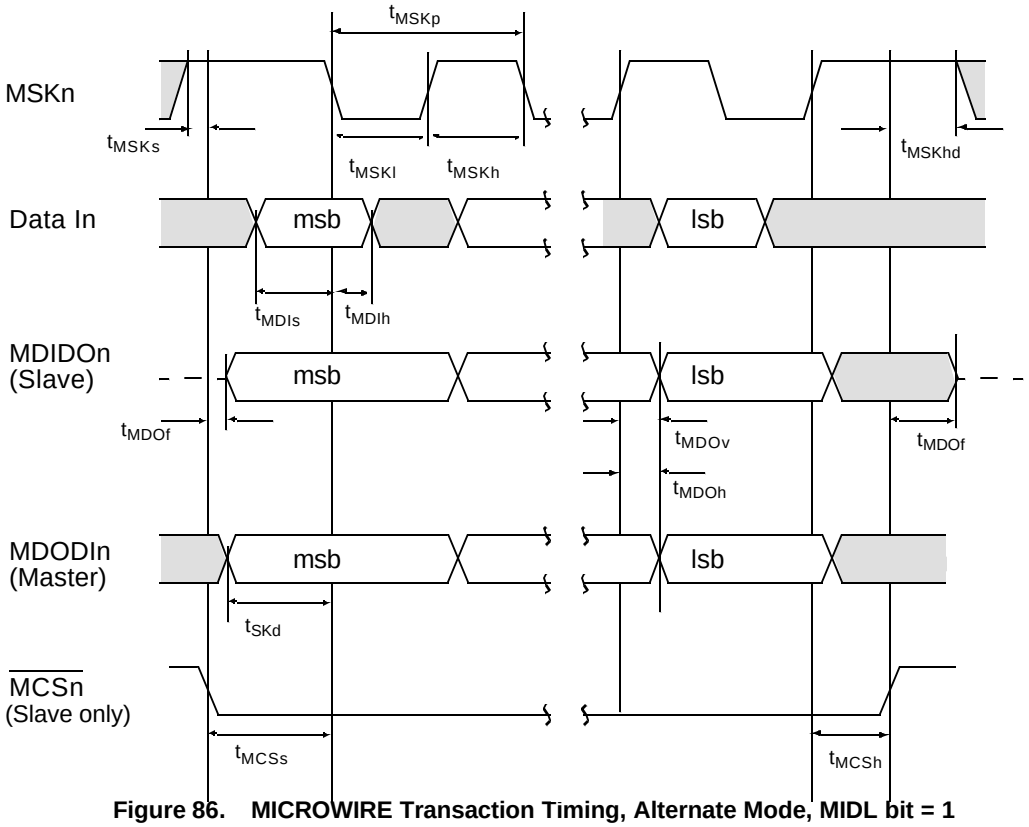



**Figure 86. MICROWIRE Transaction Timing, Alternate Mode, MIDL bit = 1**

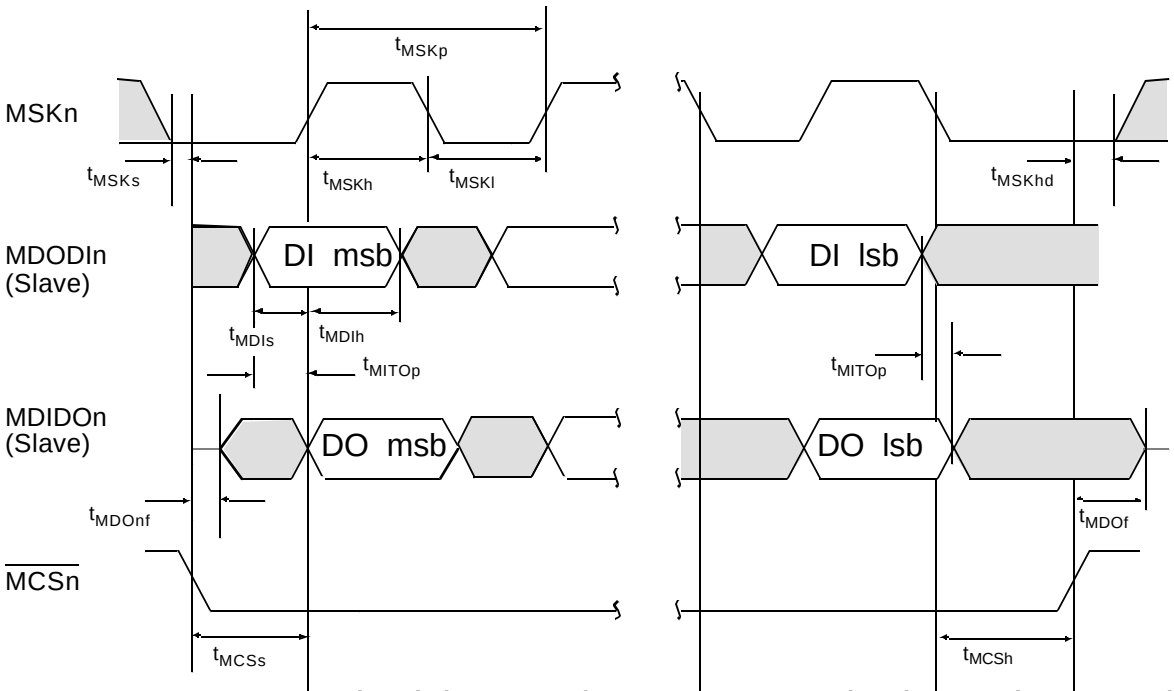
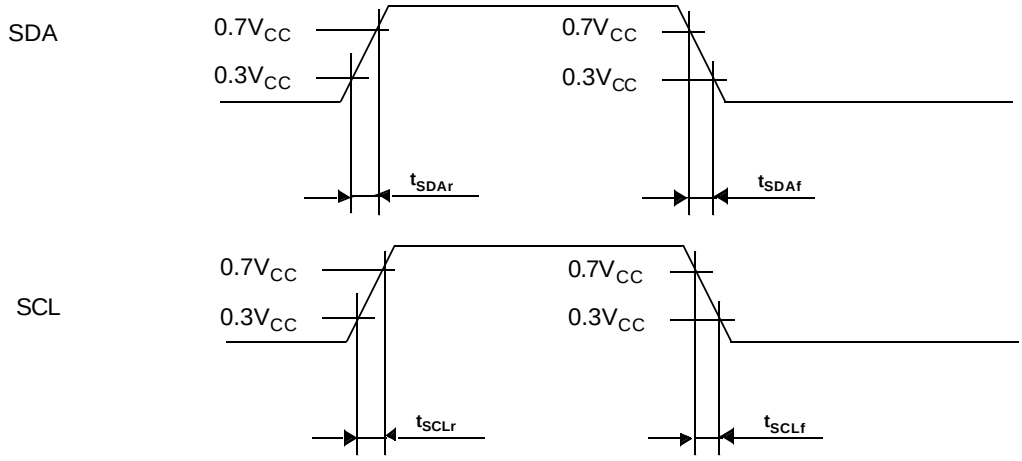
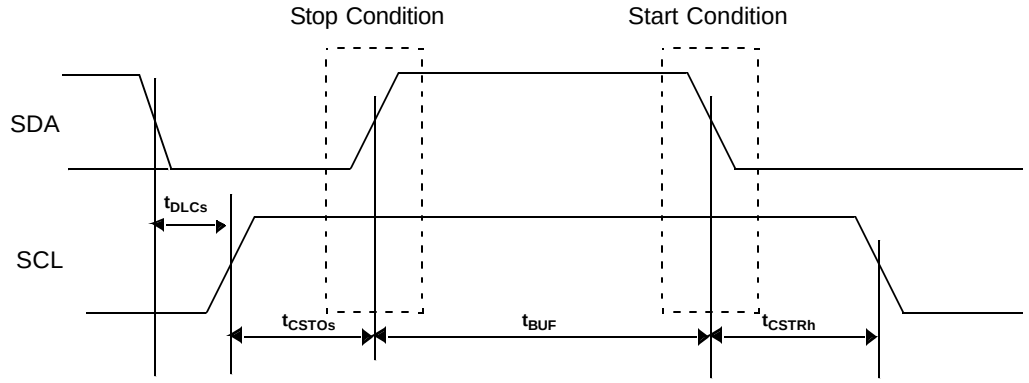



**Figure 87. MICROWIRE Transaction Timing, Data Echoed to Output, Normal Mode, MIDLBit= 0, MECHBit= 1, Slave**



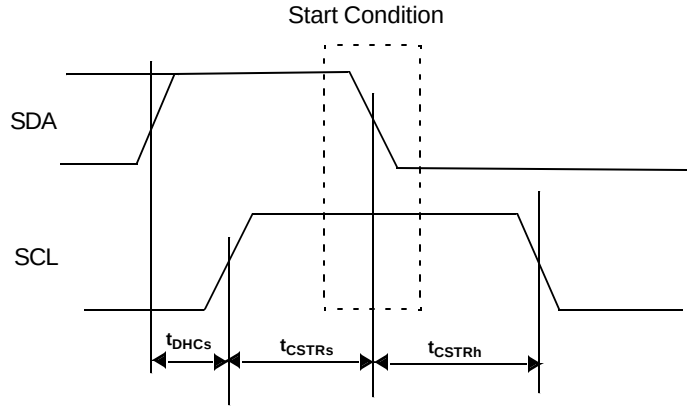

Note: In the timing tables the parameter name is added with an “o” for output signal timing and “i” for input signal timing.

**Figure 88. ACB signals (SDA and SCL) Rise Time and Fall Timing**



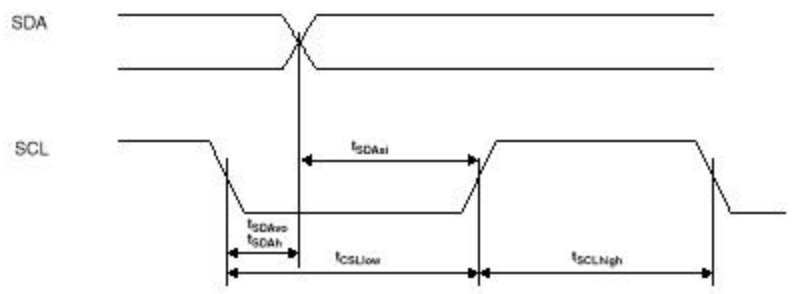

Note: In the timing tables the parameter name is added with an “o” for output signal timing and “i” for input signal timing.

**Figure 89. ACB Start and Stop Condition Timing**




Note: In the timing tables the parameter name is added with an “o” for output signal timing and “i” for input signal timing.

**Figure 90. ACB Start Conditioning Timing**




Note: In the timing tables the parameter name is added a suffix of "o" for output signal timing and "i" for input signal timing, unless the parameter already includes the suffix.

**Figure 91. ACB Data Bits Timing**

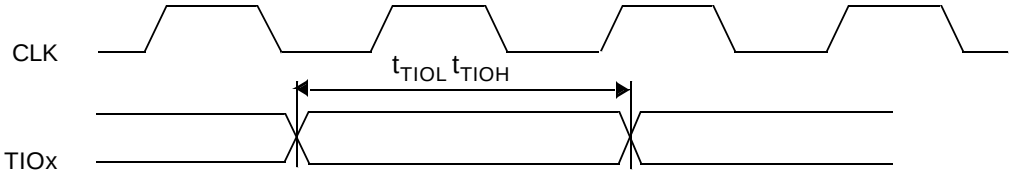



**Figure 92. Versatile-Timer-Unit Input Timing**

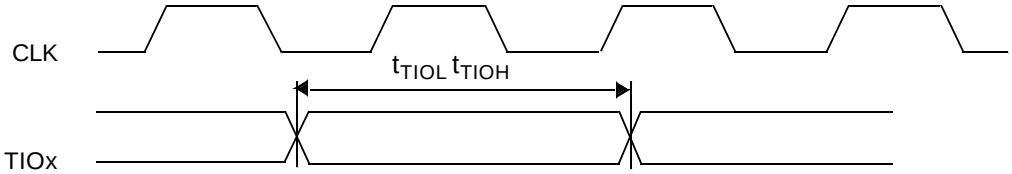



**Figure 93. Versatile-Timer-Unit Input Timing**

### 25.0.2 Timing Tables

**Table 45 Output Signals**

| Symbol | Figure | Description | Reference | Min (ns) | Max (ns) |
|---|---|---|---|---|---|
| Tclk [a] | 77 | CLK clock period | R.E. CLK to next R.E. CLK | 43.4 | 64000[a] |
| $t_{CLKh}$ | 77 | CLK high time | At 2.0V (Both Edges) | 17.3 | |
| $t_{CLKl}$ | 77 | CLK low time | At 0.8V (Both Edges) | 17.3 | |
| $t_{CLKr}$ | 77 | CLK rise time on R.E. CLK | 0.8V to 2.0V | | 3 |
| $t_{CLKf}$ | 77 | CLK fall time on F.E. CLK | 2.0V to 0.8V | | 3 |
| $t_{COv1}$ | | CMOS output valid All signals with prop. delay from CLK R.E. | After R.E. CLK | | 35 |
| **USART Output Signals** | | | | | |
| $t_{TXD}$ | 84 | TXDn output valid | After R.E. CLKXn | | 35 |
| **MICROWIRE / SPI Output Signals** | | | | | |
| $t_{MSKh}$ | 86 | MICROWIRE Clock High | At 2.0V (both edges) | 80 | |
| $t_{MSKl}$ | 86 | MICROWIRE Clock Low | At 0.8V (both edges) | 80 | |
| $t_{MSKp}$ | 86 | MICROWIRE Clock Period | MnIDL bit = 0: R.E. MSK to next R.E. MSKn | 200 | |
| | 87 | | MnIDL bit = 1: F.E. MSK to next F.E. MSKn | | |
| $t_{MSKd}$ | 86 | MSK Leading Edge Delayed (master only) | Data Out Bit #7 Valid | 0.5 $t_{MSK}$ | 1.5 $t_{MSK}$ |
| $t_{MDOf}$ | 86 | MICROWIRE Data Float [b] (slave only) | After R.E. $\overline{MCSn}$ | | 56 |
| $t_{MDOh}$ | 86 | MICROWIRE Data Out Hold | Normal Mode: After F.E. MSK | 0.0 | |
| | | | Alternate Mode: After R.E. MSK | | |
| $t_{MDOnf}$ | 90 | MICROWIRE Data No Float (slave only) | After F.E. $\overline{MWCS}$ | 0 | 56 |
| $t_{MDOv}$ | 86 | MICROWIRE Data Out Valid | Normal Mode: After F.E. MSK | | 56 |
| | | | Alternate Mode: After R.E. MSK | | |
| $t_{MITOp}$ | 90 | MDODI to MDIDO (slave only) | Propagation Time Value is the same in all clocking modes of the MICROWIRE | | 56 |
| **CAN Output Signals** | | | | | |
| $t_{CANTx}$ | | CANTx output valid | After R.E. CLKXn | | 13 |
| **ACCESS.bus Output Signals** | | | | | |
| $t_{BUFo}$ | 89 | Bus free time between Stop and Start Condition | | $t_{SCLhigho}$ | |
| $t_{CSTOso}$ | 89 | SCL setup time | Before Stop Condition | $t_{SCLhigho}$ | |
| $t_{CSTRho}$ | 89 | SCL hold time | After Start Condition | $t_{SCLhigho}$ | |
| $t_{CSTRso}$ | 90 | SCL setup time | Before Start Condition | $t_{SCLhigho}$ | |
| $t_{DHCso}$ | 90 | Data High setup time | Before SCL R.E. | $t_{SCLhigho}$ -$t_{SDAro}$ | |
| $t_{DLCso}$ | 89 | Data Low setup time | Before SCL R.E. | $t_{SCLhigho}$ -$t_{SDAfo}$ | |
| $t_{SCLfo}$ | 88 | SCL signal Fall time | | | 300[c] |

**Table 45 Output Signals**

| Symbol | Figure | Description | Reference | Min (ns) | Max (ns) |
|---|---|---|---|---|---|
| $t_{SCLro}$ | 88 | SCL signal Rise time | | | - [d] |
| $t_{SCLlowo}$ | 91 | SCL low time | After SCL F.E. | K*$t_{CLK}$ - 1[e] | |
| $t_{SCLhigho}$ | 91 | SCL high time | After SCL R.E. | K*$t_{CLK}$ - 1[e] | |
| $t_{SDAfo}$ | 88 | SDA signal Fall time | | | 300 |
| $t_{SDAro}$ | 88 | SDA signal Rise time | | | - |
| $t_{SDAho}$ | 91 | SDA hold time | After SCL F.E. | 7*$t_{CLK}$ - $t_{SCLfo}$ | |
| $t_{SDAvo}$ | 91 | SDA valid time | After SCL F.E. | | 7*tCLK+ tRD |

a. Tclk is the actual clock period of the CPU clock used in the system.
The value of Tclk is system dependent.
The maximum cycle time of 64000ns is for Power Save mode; in active mode, the maximum cycle time is limited to 250ns by the high frequency oscillator.
b. Guaranteed by design, but not fully tested.
c. Assuming signal's capacitance up to 400pF.
d. Depends on the signal's capacitance and the pull-up value. Must be less than 1ms.
e. K is as defined in ACBCTL2.SCLFRQ.

**Table 46 Input Signal Requirements**

| Symbol | Figure | Description | Reference | Min (ns) | Max (ns) |
|---|---|---|---|---|---|
| $t_{X1p}$ | 77 | X1 period | R.E. X1 to next R.E. X1 | 40 | |
| $t_{X1h}$ | 77 | X1 high time, external clock | At 2V level (Both Edges) | 0.5 Tclk - 4 | |
| $t_{X1l}$ | 77 | X1 low time, external clock | At 0.8V level (Both Edges) | 0.5 Tclk - 4 | |
| $t_{X2p}$ | 77 | X2 period [a] | R.E. X2 to next R.E. X2 | 10,000 | |
| $t_{X2h}$ | 77 | X2 high time, external clock | At 2V level (both edges) | 0.5 Tclk - 500 | |
| $t_{X2l}$ | 77 | X2 low time, external clock | At 0.8V level (both edges) | 0.5 Tclk - 500 | |
| $t_{Is}$ | 81 | Input setup time<br>ISE | Before R.E. CLK | 12 | |
| $t_{Ih}$ | 81 | Input hold time<br>$\overline{ISE}$, $\overline{NMI}$, RXD1, RXD2 | After R.E. CLK | 0 | |
| $t_{RST}$ | 82 | Reset time | Reset active to reset end | 4Tclk | |
| **Input Signals** | | | | | |
| | | Input Pulse Width | | 1*Tclk+13 | |
| **USART Input Signals** | | | | | |
| $t_{Is}$ | 80 | Input setup time<br>RXDn (asynchronous mode) | Before R.E. CLK | 12 | |
| $t_{Ih}$ | 80 | Input hold time<br>RXDn (asynchronous mode) | After R.E. CLK | 0 | |
| $t_{CLKX}$ | 81 | CKXn input period<br>(synchronous mode) | | 200 | |
| $t_{RXS}$ | 81 | RDXn setup time<br>(synchronous mode) | Before F.E. CKX in synchronous mode | 4 | |
| $t_{RXH}$ | 81 | RDXn hold time<br>(synchronous mode) | After F.E. CKX in synchronous mode | 2 | |
| **MICROWIRE / SPI Input Signals** | | | | | |
| $t_{MSKh}$ | 83 | MICROWIRE Clock High | At 2.0V (both edges) | 80 | |
| $t_{MSKl}$ | 83 | MICROWIRE Clock Low | At 0.8V (both edges) | 80 | |

**Table 46 Input Signal Requirements**

| Symbol | Figure | Description | Reference | Min (ns) | Max (ns) |
|---|---|---|---|---|---|
| $t_{MSKp}$ | 83 | MICROWIRE Clock Period | MnIDL bit = 0; R.E. MSK to next R.E. MSK | 200 | |
| | 84 | | MIDL bit = 1; F.E. MSK to next F.E. MSK | | |
| $t_{MSKh}$ | 83 | MSK Hold (slave only) | After $\overline{MCS}$ becomes inactive | 40 | |
| $t_{MSKs}$ | 83 | MSK Setup (slave only) | Before $\overline{MCS}$ becomes active | 80 | |
| $t_{MCSh}$ | 83 | $\overline{MCS}$ Hold (slave only) | MIDL bit = 0: After F.E. MSK | 40 | |
| | 84 | | MIDL bit = 1: After R.E. MSK | | |
| $t_{MCSs}$ | 83 | $\overline{MCS}$ Setup (slave only) | MIDL bit = 0: Before R.E. MSK | 80 | |
| | 84 | | MIDL bit = 1: Before F.E. MSK | | |
| $t_{MDIh}$ | 83 | MICROWIRE Data In Hold (master) | Normal Mode: After R.E. MSK | 0 | |
| | 85 | | Alternate Mode: After F.E. MSK | | |
| | 83 | MICROWIRE Data In Hold (slave) | Normal Mode: After R.E. MSK | 40 | |
| | 85 | | Alternate Mode: After F.E. MSK | | |
| $t_{MDIs}$ | 83 | MICROWIRE Data In Setup | Normal Mode: Before R.E. MSK | 80 | |
| | 85 | | Alternate Mode: Before F.E. MSK | | |
| **CAN Input Signals** | | | | | |
| $t_{Is}$ | | CANRx Input setup time) | Before R.E. CLK | 12 | |
| $t_{Ih}$ | | CANRx Input hold time | After R.E. CLK | 0 | |
| **ACCESS.bus Input Signals** | | | | | |
| $t_{BUFi}$ | 89 | Bus free time between Stop and Start Condition | | $t_{SCLhigho}$ | |
| $t_{CSTOsi}$ | 89 | SCL setup time | Before Stop Condition | $8*t_{CLK} - t_{SCLri}$ | |
| $t_{CSTRhi}$ | 89 | SCL hold time | After Start Condition | $8*t_{CLK} - t_{SCLri}$ | |
| $t_{CSTRsi}$ | 90 | SCL setup time | Before Start Condition | $8*t_{CLK} - t_{SCLri}$ | |
| $t_{DHCsi}$ | 90 | Data High setup time | Before SCL R.E. | $2*t_{CLK}$ | |
| $t_{DLCsi}$ | 89 | Data Low setup time | Before SCL R.E. | $2*t_{CLK}$ | |
| $t_{SCLfi}$ | 88 | SCL signal Rise time | | | 300 |
| $t_{SCLri}$ | 88 | SCL signal Fall time | | | 1000 |
| $t_{SCLlowi}$ | 91 | SCL low time | After SCL F.E. | $16*t_{CLK}$ | |
| $t_{SCLhighi}$ | 91 | SCL high time | After SCL R.E. | $16*t_{CLK}$ | |
| $t_{SDAri}$ | 88 | SDA signal Rise time | | | 300 |
| $t_{SDAfi}$ | 88 | SDA signal Fall time | | | 1000 |
| $t_{SDAhi}$ | 91 | SDA hold time | After SCL F.E. | 0 | |
| $t_{SDAsi}$ | 91 | SDA setup time | Before SCL R.E. | $2*t_{CLK}$ | |
| **Multi-Function Timer Input Signals** | | | | | |
| tTAH | 92 | TnA High Time | R.E. CLK | $T_{CLK}+5$ | |
| tTAL | 92 | TnA Low Time | R.E. CLK | $T_{CLK}+5$ | |
| tTBH | 92 | TnB High Time | R.E. CLK | $T_{CLK}+5$ | |
| tTBL | 92 | TnB Low Time | R.E. CLK | $T_{CLK}+5$ | |
| **Versatile Timer Input Signals** | | | | | |
| tTIOH | 96 | TIOx Input High Time | RE CLK | $1.5T_{CLK}+5ns$ | |
| tTIOL | 96 | TIOx Input Low Time | RE CLK | $1.5T_{CLK}+5ns$ | |

a. Only when operating with an external square wave on X2CKI; otherwise a 32kHz crystal network must be used between X2CKI and X2CKO. If the slow clock is internally generated from the fast clock, it may not exceed this given limit.

# 26.0 Appendix

The following document describes problems identified in the CR16 modules.

## 26.1 CR16CAN

### 26.1.1 CR16CAN Problem Descriptions:

Under certain conditions it occurs that the CR16CAN module receives a frame, sent by itself even though the loopback feature is disabled.

This condition consists of two parts, which both must be true to cause this malfunction.

A) The first part is that a transmit buffer and at least one receive buffer are configured with the same identifier. Let's call this identifier ID_RX_TX here. With regard to the receive buffer, this means that the buffer identifier and the corresponding filter masks are setup in a way that the buffer is able to receive frames with the identifier ID_RX_TX.

B) The second part is that the CAN communication must take place in the following sequence:

1. A message with the identifier ID_RX_TX from another CAN node is received into the receive buffer.
2. A message with the identifier ID_RX_TX is sent by the CR16CAN module immediately after the reception took place (Note).

After this communication the frame sent by the CR16CAN module will be copied into the next receive buffer available for the identifier ID_RX_TX.

**Note:** If a frame with an identifier different to ID_RX_TX is sent or received in between the steps 1 and 2, the problem does not occur.

### 26.1.2 CR16CAN Problem Cause

When a frame is received into the hidden receive buffer, the CR16CAN module scans through all CAN message buffers. During this sequence all receive buffers (RX-buffers) capable of receiving this frame are tagged (RX-tag). If the message was received correctly the frame is copied into first tagged buffer (lowest buffer number).

Every CAN node also monitors frames being transmitted in order to switch from transmitter to receiver after a lost arbitration. In order to do this the CR16CAN module also receives transmitted frames into the hidden receive buffer.

The scanning sequence is also applied to transmitted frames. This means, the identifier in the hidden receive buffer is compared with the RX-buffer identifiers. As the identifier is the same as with the last receive scanning sequence, the RX-tags will not be changed.

A CAN buffer receive tag is only updated in the following cases:

- — A new scanning sequence has overwritten the old receive tags due to a different identifier mask under comparison.
- — The CPU has changed the CAN buffer status in the CNSTAT.ST-field to any value which disables the buffer to receive a message (e.g. RX_NOT_ACTIVE or any TX-state)
- — The CPU has changed the CAN buffer identifier.

Applied to the CAN communication sequence described above, this means that the transmitted message, currently present in the hidden receive buffer will be copied into the same receive buffer and the message received from the other CAN node will be overwritten.

**Example**

**Buffer Settings**

Filter Masks:

GMSKB = 0x0000
GMSKX = 0x000F

Buffer configuration:

| CAN Buffer Number | CAN Buffer Status | Buffer Identifier | Identifier Mask |
|---|---|---|---|
| 0 | TX_NOT_ACTIVE | 0x15555550 | 0x1555555X |
| 1 | RX_READY | 0x15555551 | 0x1555555X |
| 2 | RX_READY | 0x15555552 | 0x1555555X |
| 3 | RX_READY | 0x15555003 | 0x1555500X |

X = don't care

**CAN Communication Sequence A:**

(BUFFLOCK disabled)

1. Message sent from another CAN node received into buffer 1.
   Buffer 1 and buffer 2 are tagged for reception of this message.
2. CPU reads out data from CAN buffer 1 and resets the buffer state from RX_FULL to RX_READY (Note 1).
3. CAN buffer 0 sends a frame (status set to TX_ONCE).
4. Status of CAN buffer 1 changes to RX_FULL, because it has received the message sent by buffer 0 (Note 2).

**Note:** 1. Step 2 does not need to be done. In case the buffer 1 status is not updated to RX_READY, the buffer status will change from RX_FULL to RX_OVERRUN in step 4.

**Note:** 2. As BUFFLOCK is disabled, all messages with the identifier ID_RX_TX will be copied into buffer 1. Buffer 2 does not receive any message.

**CAN Communication Sequence B:**

(BUFFLOCK enabled)

1. Message sent from another CAN node received into buffer 1. Buffer 1 is locked now.
   Buffer 1 and buffer 2 are tagged for reception of this message.
2. CAN buffer 0 sends a frame.
3. Status of CAN buffer 2 changes to RX_FULL, because it has received the message sent by buffer 0.

**CAN Communication Sequence C**

(CR16CAN does NOT receive a frame sent by itself.)

1. Message sent from another CAN node received into buffer 1.
   Buffer 1 and buffer 2 are tagged for reception of this message.
2. Message sent from another CAN node received into buffer 3 (ID=0x15555003).
   Only buffer 3 is now tagged for reception.
3. CAN buffer 0 sends a frame (status set to TX_ONCE).
4. Status of CAN buffer 1 and 2 remains RX_READY, because they have not received the message sent by buffer 0.

### 26.1.3 CR16CAN Problem Solutions

**Reset receive buffer tags before transmitting a message**

The receive tag of a CAN receive buffer is reset when the CPU updates the buffer status in the CNSTAT.ST-field to any value which disables the receive buffer. Therefore the user should write the sequence RX_NOT_ACTIVE - RX_READY to all receive buffers which have an identifier filter matching the identifier of the frame to be sent next before the message is sent.

**Modified CAN Communication Sequence:**

(BUFFLOCK disabled)

The same CAN buffer settings as described in also apply to this example.

1. Message sent from another CAN node received into buffer 1.
   Buffer 1 and buffer 2 are tagged for reception of this message.
2. CPU reads out data from CAN buffer 1 and resets the buffer state from RX_FULL to RX_READY.
3. Write RX_NOT_ACTIVE to CNSTAT.ST-field of buffer 1 and buffer 2.
   Buffer 1 and buffer 2 are NOT tagged for reception anymore.
4. Write RX_READY to CNSTAT.ST-field of buffer 1 and buffer 2.
5. CAN buffer 0 sends a frame (status set to TX_ONCE).
6. Status of CAN buffer 1 remains RX_READY, because it has NOT received the message sent by buffer 0.

**Advantage:**

No receive buffer is overwritten by a message sent by the same CR16CAN node.

**Disadvantage:**

The corresponding receive buffers must be disabled for a short period of time. During this time, when the receive buffers are in the RX_NOT_ACTIVE state, correct incoming messages from other CAN nodes will get lost.

This method is more suitable compared to the method described in Section, if the number of transmit buffers with identifier ID_RX_TX is lower than the number of receive buffers set up with the corresponding identifier mask.

**Reset receive buffer tags after reception of a message**

The receive tag of a CAN receive buffer is reset when the CPU updates the buffer status in the CNSTAT.ST-field to any value which disables the receive buffer. Therefore the user should write the sequence RX_NOT_ACTIVE - RX_READY to this receive buffer, which has received the latest message.

**Modified CAN Communication Sequence:**

In the CAN communication example described below, the buffer 14 is set up as basic CAN path, which is able to receive all standard frames. The buffers 1 to 13 cannot receive the frame sent by buffer 0.

Filter Masks:

BMSKB = 0xFFF0
BMSKX = 0x0000

Buffer configuration**:**

| CAN Buffer Number | CAN Buffer Status | Buffer Identifier |
|---|---|---|
| 0 | TX_NOT_ACTIVE | any standard frame |
| 14 | RX_READY | ID1.IDE bit = 1 |

1. Message sent from another CAN node received into buffer 14.
   Buffer 14 is tagged for reception of this message.
2. CPU reads out data from CAN buffer 14.
3. Write RX_NOT_ACTIVE to CNSTAT.ST-field of buffer 14.
   Buffer 14 is NOT tagged for reception anymore.
4. Write RX_READY to CNSTAT.ST-field of buffer 14.
5. CAN buffer 0 sends a frame (status set to TX_ONCE).
6. Status of CAN buffer 14 remains RX_READY, because it has NOT received the message sent by buffer 0.

**Advantage:**

No receive buffer is overwritten by a message sent by the same CR16CAN node.

**Disadvantage:**

The corresponding receive buffers must be disabled for a short period of time. During this time, when the receive buffers are in the RX_NOT_ACTIVE state, correctly incoming messages from other CAN nodes will get lost.

This method is more suitable compared to the method described in Section, if the number of transmit buffers with identifier ID_RX_TX is higher than the number of receive buffers set up with the corresponding identifier mask. This is the case if only the basic CAN path to buffer 14 is configured to receive a range of identifiers, including the identifier ID_RX_TX. All other buffers are configured with unique identifier filters.

**Receive all frames and discard those, which were sent by the same CR16CAN node.**

Another approach to overcome this problem uses the Time Stamp counter of the CR16CAN module to determine, whether a message was sent and received at the same time. This is the case when a transmitted frame is received by the same CAN node.

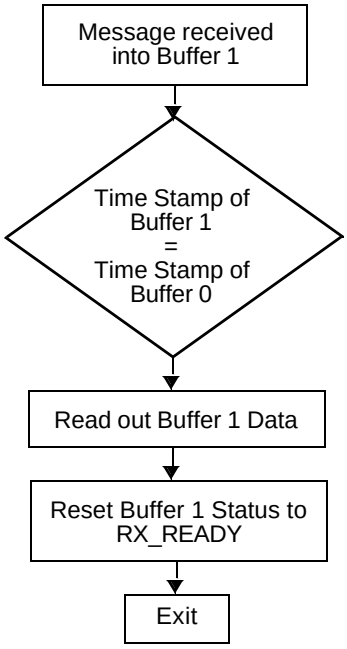
When a frame was successfully sent by CR16CAN the contents of the Time Stamp counter are captured into the Time Stamp register (TSTP) of the transmit buffer during the ACK-slot of the frame currently being sent. Also, when a message is received, the TSTP-register of the receiving buffer is loaded with the Time Stamp counter value during the ACK-slot of the CAN frame currently being received.

This means, in the case where a message is received in one buffer, which was sent from another buffer of the same CR16CAN node, the TSTP-register contents are equal after this transaction.

A comparison of the two TSTP-register values can be inserted into a "read from CAN receive buffer" software routine, to distinguish whether the data received are from another CAN node (valid) or from the same CAN node (invalid).

The flowchart below shows a possible implementation. The same CAN buffer settings as described in Section also apply to this example.

**Modified CAN Receive Sequence:**




**Advantage:**

None of the CAN receive buffers must be disabled at any time.

**Disadvantage:**
The receive buffer contents are overwritten by an invalid message sent from the same CR16CAN node.

## 26.2 8/16-BIT MICROWIRE/SPI (MWSPI16)

### 26.2.1 MWSPI16 Problem Description

According to the specification, the MSKn clock output in master mode should have the value of the MnIDL bit of the MWnCTL register, even when the module is disabled. However, the MSKn pin is enabled and the module is disabled. thus, even if the MnIDL bit is set, the MSKn clock will change to a low level as soon as the module is disabled. If any slave is selected at this time, i will interpret this unwanted transition as a shift clock.

### 26.2.2 MWSPI16 Problem Cause

Even if the module is disabled and the alternate function of the MSKn pin is enabled, the module can still influence the MSKn pin and drives the default value ‘0’.

### 26.2.3 MWSPI16 Problem Solutions

When the MSKn idle level of ‘1’ is to be used, the following procedure should be followed when the module is disabled:

1. Set the MSKn pin to high level in the corresponding port data output register.
2. Configure the MSKn pin to an output in the corresponding port direction register.
3. Disable the alternate function of the MSKn pin in the corresponding port alternate function register.
4. Disable the MWSPI16 module.

## 26.3 TIMING AND WATCHDOG MODULE

### 26.3.1 Timing and WATCHDOG Module Problem Description

The available window for a valid WATCHDOG service varies with the TWM configuration and the operating mode of the R16MCS9. Therefore it is not possible to generally provide the limits for the maximum service window. However, the limits for the minimum service window is guaranteed and should be used.

### 26.3.2 Timing and WATCHDOG Module Problem Cause

The timing and WATCHDOG module uses two different clock signals for its operation, the slow system clock as well as the fast system clock.

The slow system clock can either be generated by an external 32 kHz quartz or it can be derived from the fast system clock by means of a prescaler counter in the CLK2RES modules. The TWM can operate off a maximum slow system clock of 100 kHz. The WATCHDOG counter (down-counter) is either clocked directly by the slow system (T0IN) or it is decremented every time the counter T0 underflows (T0OUT).

The fast system clock is used for accesses to TWM registers, which build the user interface of the TWM. These user interface registers include all memory-mapped registers of the TWM.

Every time the user (CR16B core) writes to a TWM configuration register or to the WATCHDOG Service Data Match register, this “high speed operation” must be synchronized to the internal TWM logic running at the slow clock rate. This synchronization process takes a variable number of low speed clock cycles, depending on the ratio between the low-speed and the high-speed system clock and the phase shift between the two clock signals. The more the two frequencies differ from each other, the longer it takes the synchronization process.

In other words, write operations to the TWM registers take a certain number of low-speed clock cycles to show the desired effects to the TWM logic.

This fact is especially critical for the write operation for the WATCHDOG service, as it affects the allowed window for a valid WATCHDOG service.

If the device runs in active mode, the synchronization process can take up to four WATCHDOG counter clock cycles. This limits the available WATCHDOG service to the window shown in figure 94:

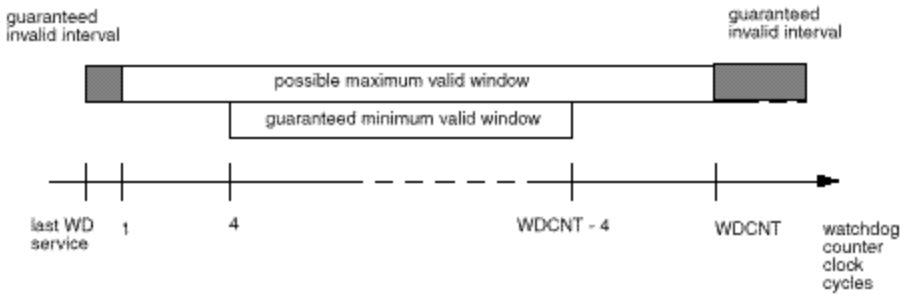



**Figure 94. WATCHDOG Services Windows in Active Mode**

If the device runs in power save mode, the synchronization process can take up to eight WATCHDOG counter clock cycles. This limits the available WATCHDOG service to the window shown in figure 95:

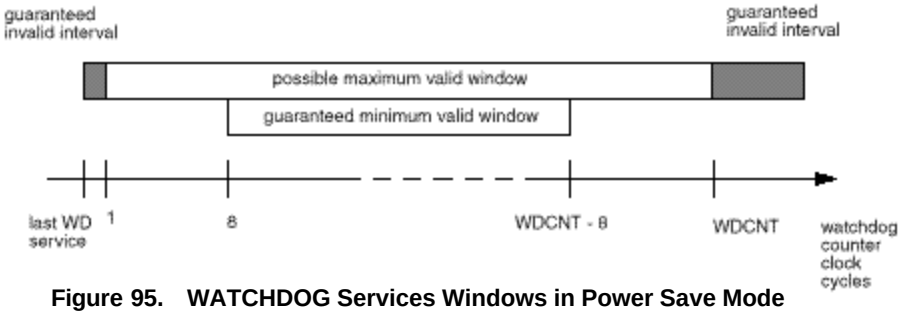



**Figure 95. WATCHDOG Services Windows in Power Save Mode**

### 26.3.3 Timing and WATCHDOG Module Problem Solutions

In order to guarantee a valid WATCHDOG service under all circumstances, the WATCHDOG should only be serviced within the guaranteed minimum valid window, as illustrated in figure 94 and figure 95 in the previous section.

**LIFE SUPPORT POLICY**

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

**National Semiconductor Corporation**
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

**National Semiconductor Europe**
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Francais Tel: +33 (0) 1 41 91 8790

**National Semiconductor Asia Pacific Customer Response Group**
Tel: 65-254-4466
Fax: 65-250-4466
Email: ap.support@nsc.com

**National Semiconductor Japan Ltd.**
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

www.national.com