TEXAS INSTRUMENTS

THS14xx/5691 EVM for the THS14xx and THS56xx DAC Families

User's Guide

September 2000

AAP Data Conversion

SLAU045A

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Copyright © 2000, Texas Instruments Incorporated

Read This First

About This Manual

The purpose of this user's guide is to serve as a reference manual for the THS14xx 14-bit ADC, analog-to-digital converter module (EVM). This document provides information to assist hardware and software engineers in application development.

How to Use This Manual

- ❑ Chapter 1 – Overview
- ❑ Chapter 2 – Physical Description
- ❑ Chapter 3 – Circuit Description

Information About Cautions and Warnings

This book may contain cautions and warnings.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Contents

Figures

Tables

Chapter 1

Overview

This chapter presents a general overview of the THS14xx/5691EVM evaluation module (EVM), and describes some of the factors that must be considered in using the module.

Topic **Page**

1.1 Purpose

The THS14xx/5691EVM evaluation module (EVM) provides a platform for evaluating the THS14XX analog-to-digital converter (ADC) and the THS56XX digital-to-analog converter (DAC) under various signal, reference, and supply conditions.

Note:

The EVM in its current implementation supports only the THS14XX device family and associated circuitry. Support for the THS56XX family will become available at a later date. As a result of this, only the operation of the THS14XX and its associated circuitry will be described in this user's guide.

1.2 EVM Basic Functions

Analog input to the ADC is provided via two external SMB connectors. This input can be configured onboard to be true differential or single-ended transformer-coupled to the input of the device.

The EVM provides an external SMB connection for ADC clock input. This can be configured to be either ac- or dc-coupled. A site provided on the board for a crystal oscillator to perform this function can be populated if required. Further provision is made to run the ADC from a DSP timer if desired.

Output from the EVM takes place via two 80-pin daughtercard connectors. The digital lines from the ADC are buffered before going to the daughtercard connectors. More information on these connectors can be found in the TMS320C6000 daughtercard specification.

The EVM is powered by 4-mm banana sockets. Separate input connectors are provided for the analog and digital supplies. Provision is made to supply the EVM from the motherboard through the daughtercard connectors.

The EVM has onboard logic that controls the memory mapping of the ADC within the motherboard's peripheral memory space.

1.3 Power Requirements

The EVM can be powered directly through the daughtercard connector's +3.3-V supply. Provision has also been made to allow the EVM to be powered with independent +3.3-V analog and digital supplies where ultimate performance demands.

Voltage Limits

Exceeding the +3.3-V maximum can damage EVM components. Undervoltage may cause improper operation of some or all of the EVM components

WARNING

1.4 THS14xx/5691EVM Operational Procedure

The THS14xx/5691EVM provides a flexible means of evaluating the THS14XX in a number of modes of operation. A basic setup procedure that can be used as a board confidence check follows:

❑ Verify all jumper settings against the schematic jumper table:

Device	Jumper Table (connection)
THS14xx	H3 pins 2–3, H5 pins 2–3, H6 pins 2–3, H7 pins 2–3, H8 pins 1–2, H9 pins 2–3, H12 pins 1–2, H13 pins 1–2

❑ Check that T2 is populated.

❑ Connect supplies to the EVM: +3.3 V on J7 and J8, and GND on J12 and J13.

❑ Switch power supplies on.

❑ Use a function generator with a 50-Ω output to apply a 5-MHz, 1.5-V offset, 3-Vp-p amplitude square-wave signal to J3.

❑ Use a function generator with a 50-Ω output to apply a 100-kHz, 0-V offset, 4-Vp-p amplitude sine-wave signal to J10.

❑ The digital pattern on the output daughtercard connector J1 should now represent a sine wave and can be monitored using a logic analyzer, or the EVM can be plugged into a motherboard and the data can be monitored using suitable software.

Chapter 2

Physical Description

This chapter describes the physical characteristics and PCB layout of the EVM and lists the components used on the module.

Topic **Page**

2.1 PCB Layout

The EVM is constructed on a four-layer, 100-mm (3.94 inch) × 86-mm (3.39 inch), 1.57-mm (0.062 inch) thick PCB using FR-4 material. Figures 2–1 through 2–6 show the individual layers.

Figure 2–1. Silk Top

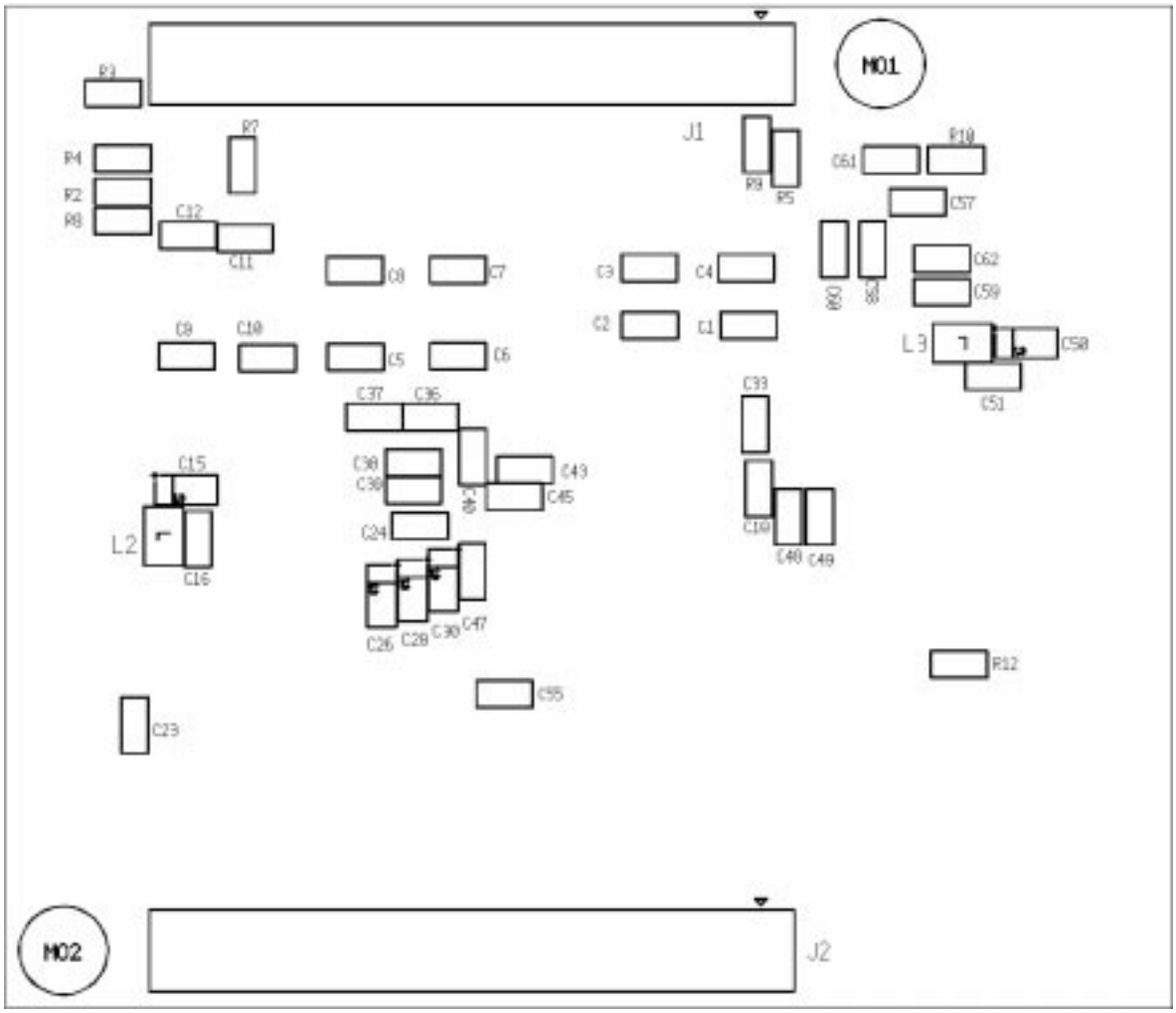


Figure 2–2. Silk Bottom

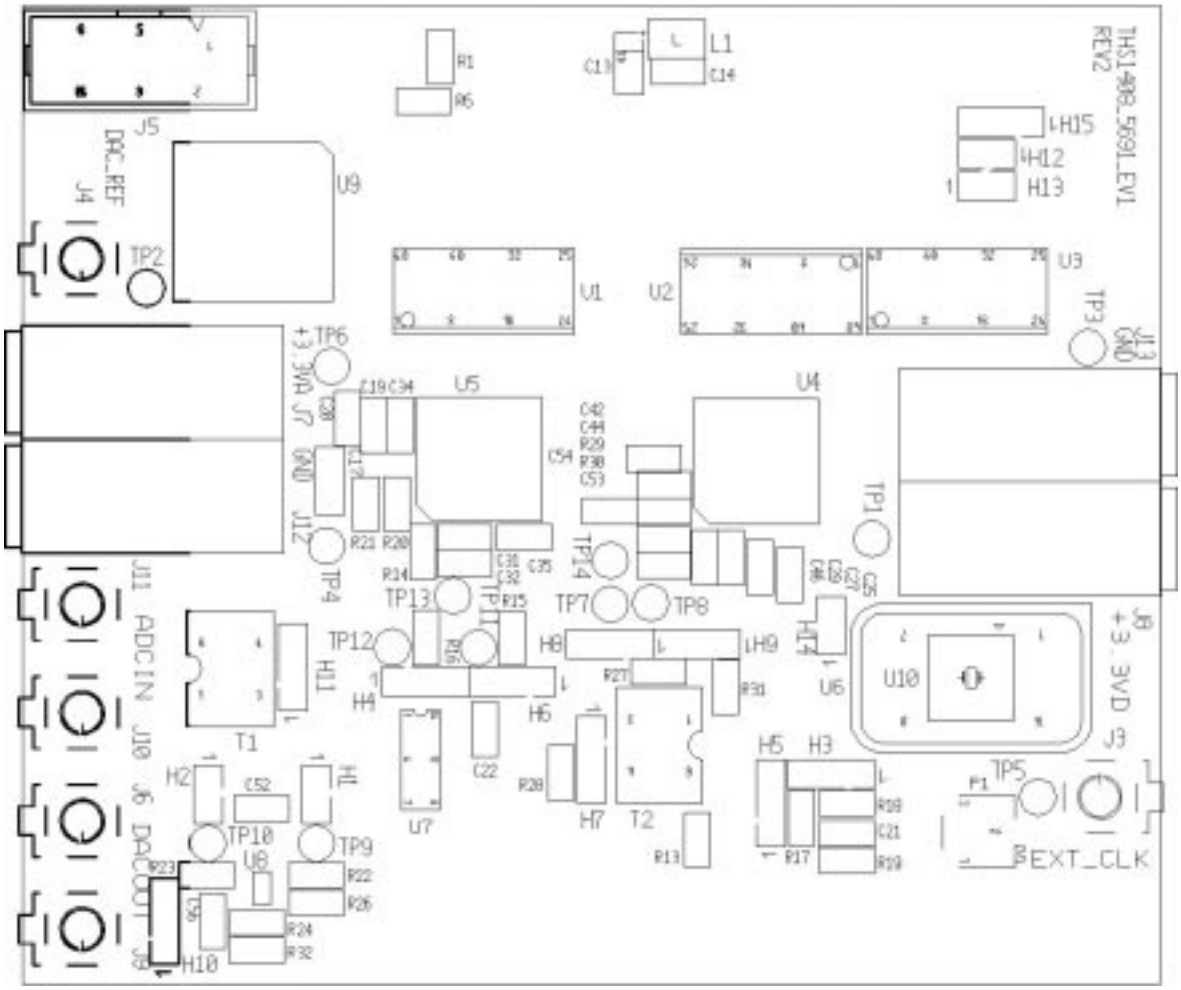


Figure 2–3. Top Layer

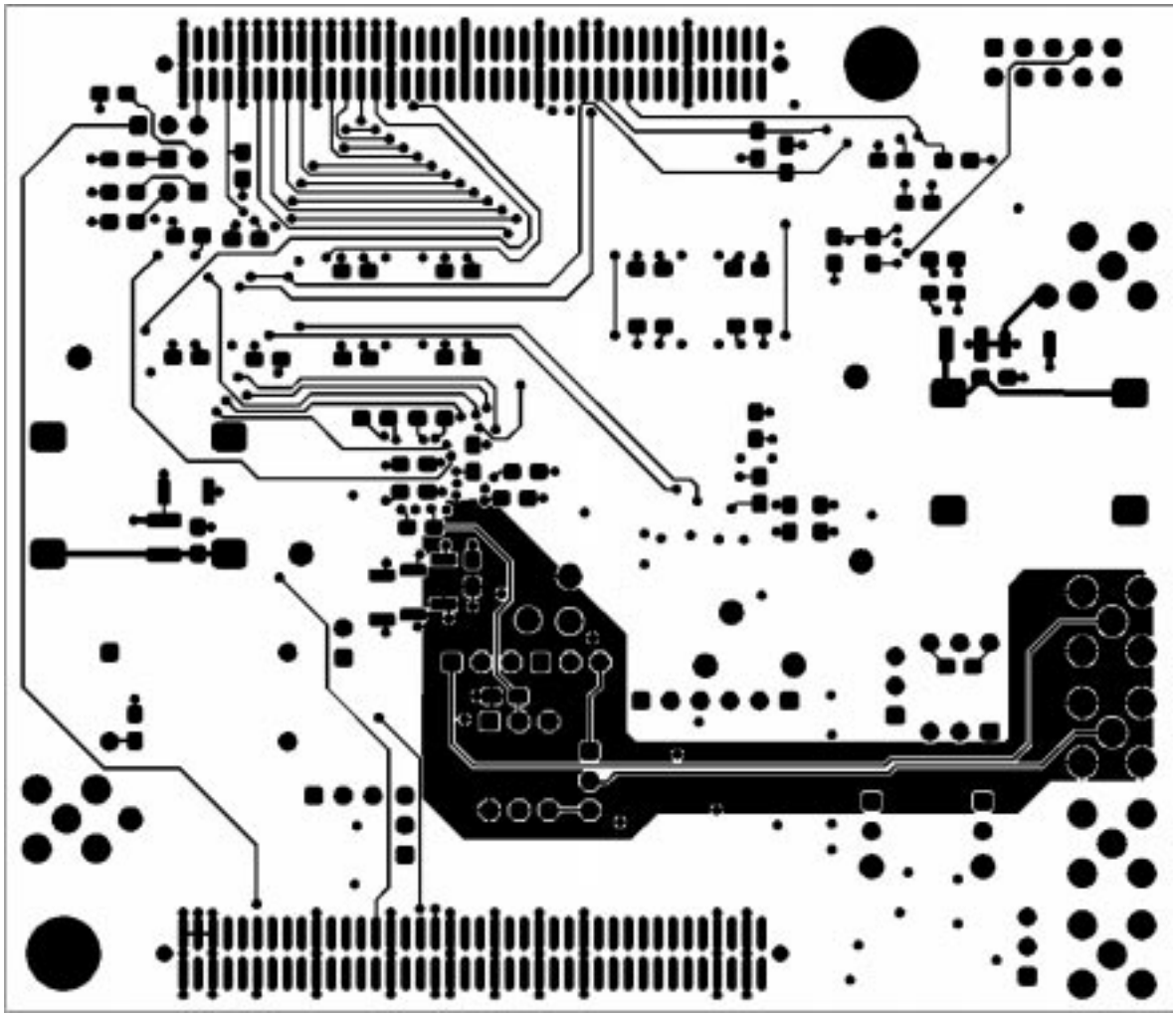


Figure 2–4. Inner Layer 1

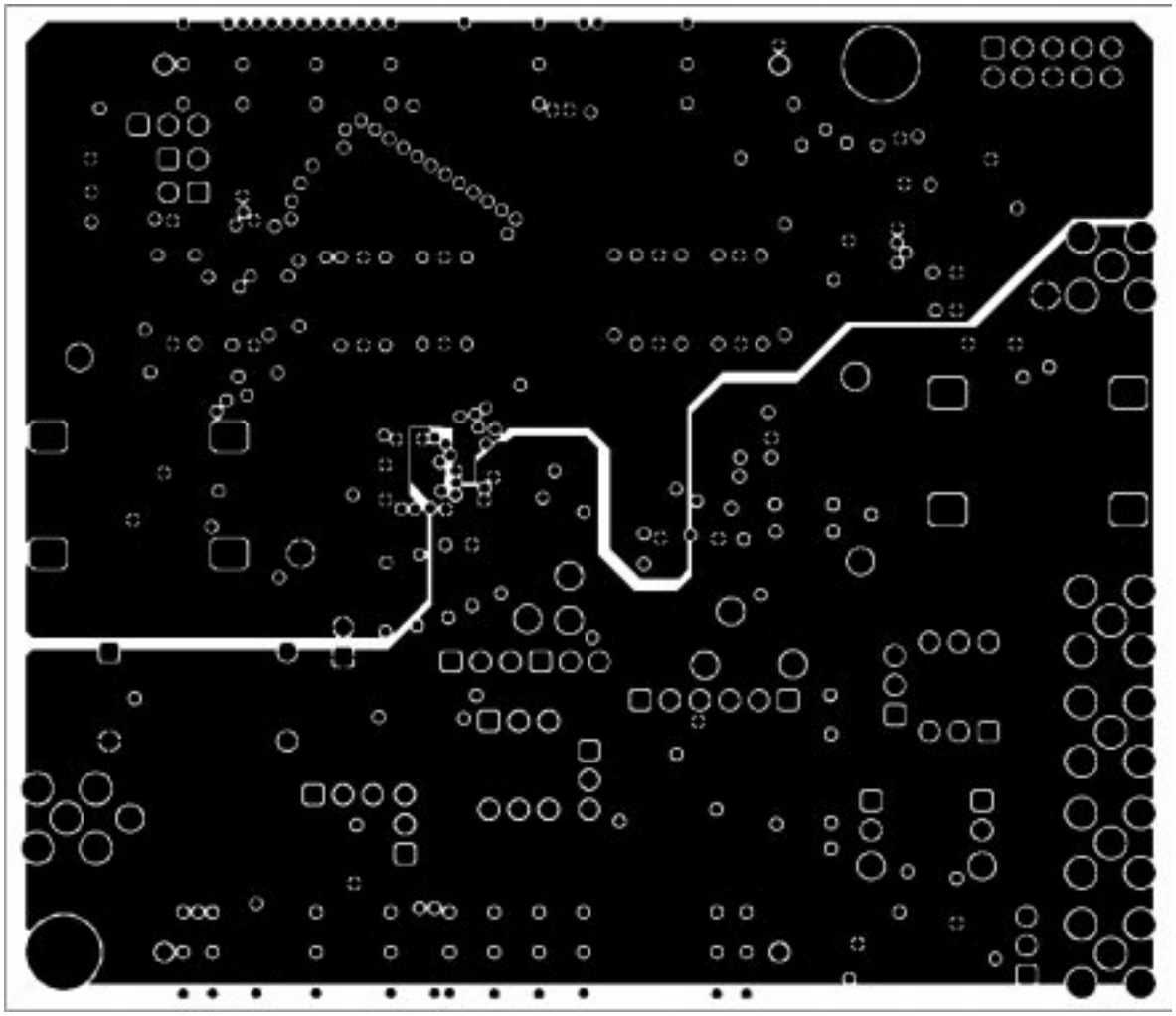


Figure 2–5. Inner Layer 2

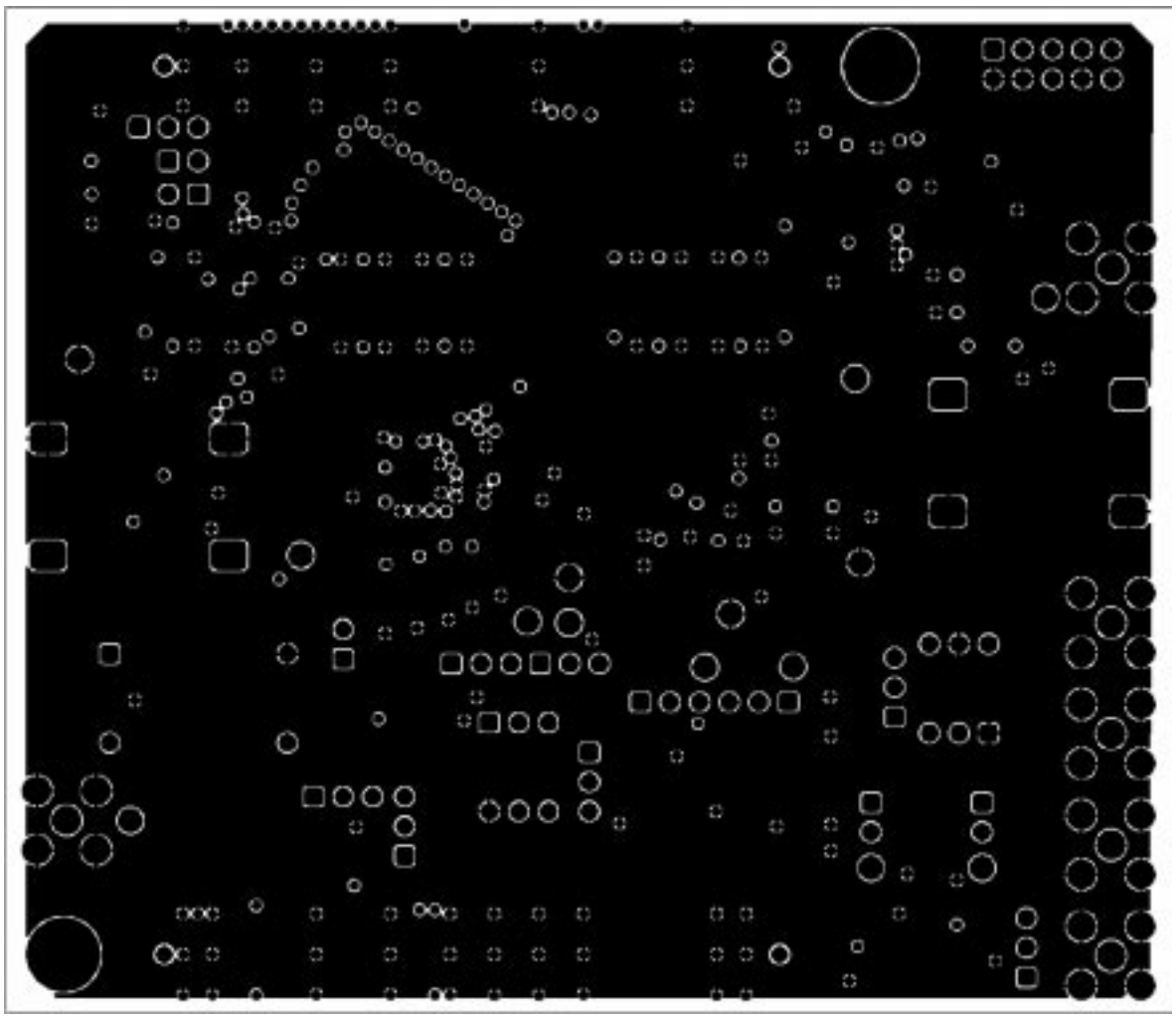
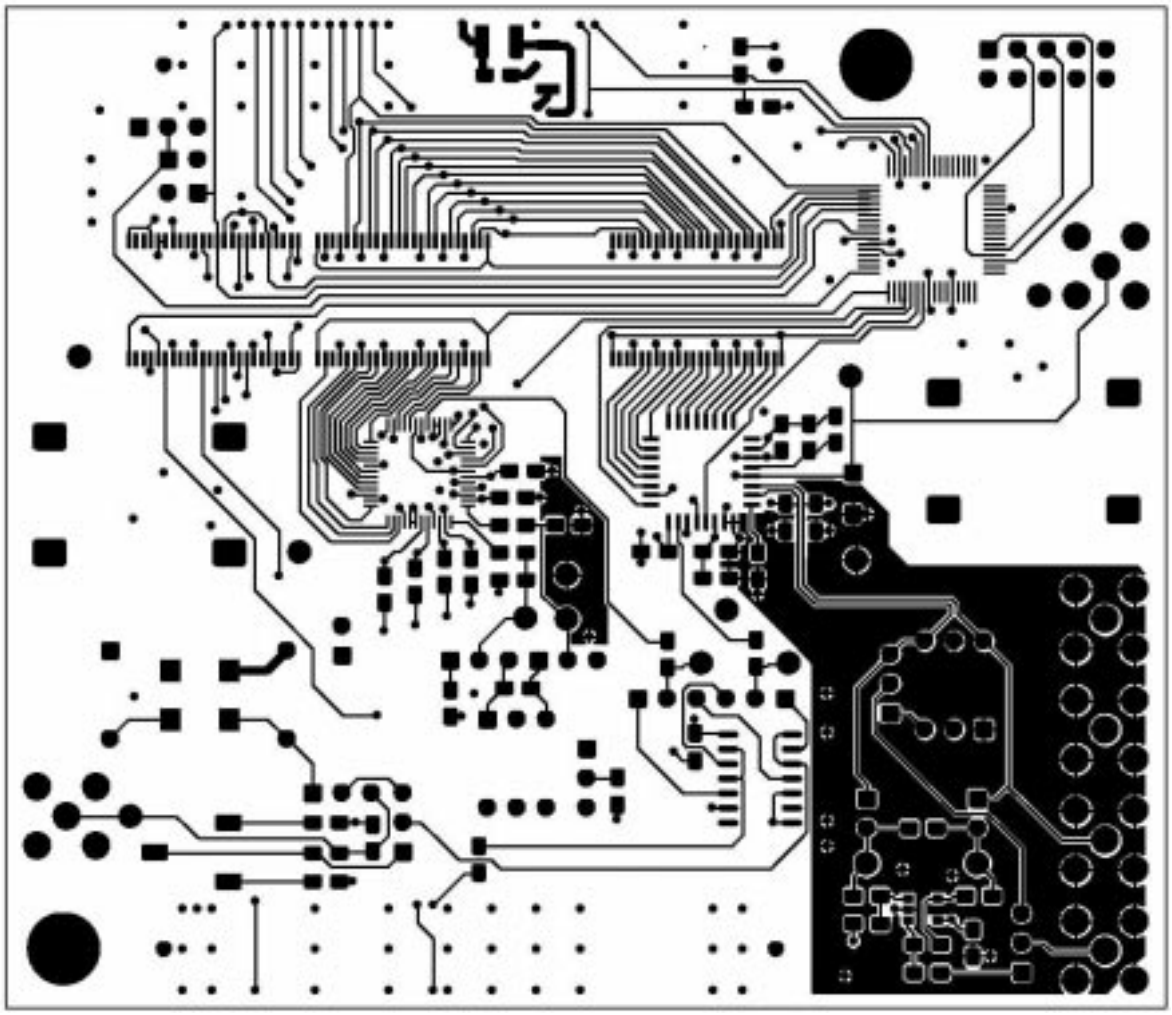


Figure 2–6. Bottom Layer



2.2 Parts List

Table 2–1 lists the parts used in constructing the EVM.

Table 2–1. Parts List

Qty	Reference Description	Description	Manufacturer	Part Number
2	U3, U1	SN74ALB16244DL bus driver	TI	SN74ALB16244DL
1	U2	SN74ALB16245DL bus transceiver	TI	SN74ALB16245DL
6	C13, C15, C26, C28, C50, C30	10-µF 10-V case A SMD tantalum capacitor	AVX	TAJA106K010R
1	C17	1-µF 1206 SMD ceramic capacitor 16-V X7R	AVX	1206YC105K2800J
40	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C14, C16, C18, C19, C21, C22, C23, C24, C25, C27, C29, C32, C34, C35, C36, C37, C38, C39, C40, C43, C45, C47, C51, C55, C56, C57, C58, C59	0.1-µF 0805 SMD ceramic capacitor X7R	TDK	CC0805HX7R104KTR
6	C20, C31, C33, C42, C44, C46	470-pF 0805 SMD ceramic capacitor NPO	AVX	08051A471JAT00J
2	C48, C49	22-pF 0805 SMD ceramic capacitor NPO	AVX	08051A220JAT2A
3	C52, C53, C54	UNPOP		
3	C60, C61, C62	0.01-µF 0805 SMD ceramic capacitor X7R	TDK	CC0805HX7R103KTR
6	J3, J4, J6, J9, J10, J11	SMB connector right angle PCB	MACOM	B65N10G999X99
1	J5	0.1” spacing 2X5 header straight	Elco	008380010000010
1	U7	SN74AHC14D hex inverter	TI	SN74AHC14D
5	H1, H2, H12, H13, H14	0.1” spacing 1X2 header straight	Harwin	M20–9990206
10	H3, H4, H5, H6, H7, H8, H9, H10, H11, H15	0.1” spacing 1X3 header straight	Harwin	M20–9990306
3	L1, L2, L3	0R 1206 thick-film resistor	Multicomp	RMC18W(1206)5%0R0
1	U8	MAX4212EUK 3.3-V operational amplifier	Maxim	MAX4212EUK
1	U6	8-MHz 14-pin DIL 3.3V oscillator 100 ppm	Golledge	GXO–U102F 8 MHz
1	U10	8.0-MHz SMD 3.3V oscillator 100 ppm	Golledge	GXO–U108L 8 MHz
2	J12, J13	4-mm PCB socket black	Deltron	571–0100
2	J7, J8	4-mm PCB socket red	Deltron	571–0500
1	P1	2-kΩ 3269P potentiometer	Bourns	3269P1202

Table 2–1. Parts List (Continued)

Qty	Reference Description	Description	Manufacturer	Part Number
8	R1, R2, R4, R7, R9, R10, R14	10-kΩ 0805 thick-film resistor 1%	Multicomp	RMC110W(O8O5)1%10K
3	R5, R6, R8	510R 0805 thick-film resistor 1%	Multicomp	RMC110W(O8O5)5%510R
2	R15, R16	39R 0805 thick-film resistor 1%	Multicomp	RMC110W(O8O5)1%39R
2	R22, R23	150R 0805 thick-film resistor 1%	Multicomp	RMC110W(O8O5)1%150R
1	R24	750R 0805 thick-film resistor 1%	Rohm	MCR10EZHF7500
3	R13, R29, R30	0R 0805 thick-film resistor 1%	Multicomp	RMC110W(O8O5)5%0R0
1	R3	1K 0805 thick-film resistor 1%	Multicomp	RMC110W(O8O5)1%1K
1	R17, R20, R21, R27, R28, R31, R32	49R9 0805 SMD chip resistor 1%	Rohm	MCR10EZHF49R9
1	R12	UNPOP		
2	R18, R19	5K1 0805 thick-film resistor 1%	Multicomp	RMC110W(O8O5)5%5K1
1	R26	2-kΩ 0805 thick-film resistor 1%	Rohm	MCR10EZHF2001
4	TP3, TP4, TP13, TP14	1.32-mm test pin black	W Hughes	100–103
10	TP1, TP2, TP5, TP6, TP7, TP8, TP9, TP10, TP11, TP12	1.32-mm test pin red	W Hughes	100–107
2	J1, J2	0.05” TFM-series 80-pin TH connector	Samtec	TFM–140–31–S–D–LC
1	U4	THS14XX ADC	TI	THS14XXPFB
1	U5	THS56XX DAC	TI	THS56XXVF
2	T2, T1	TT1–6–KK81 RF transformer	Mini Circuits	TT1–6–KK81
1	U9	XC9536XL–5VQ64C CPLD	Xilinx	XC9536XL–5VQ64C

The following components have been left unpopulated in this implementation: C1, C2, C3, C4, C17, C18, C19, C20, C31, C32, C33, C34, C35, C48, C49, C52, C53, C54, C56, H1, H2, H4, H10, H11, J4, J6, J9, R12, R14, R16, R20, R21, R22, R23, R24, R26, R32, T1, TP6, TP9, TP10, TP12, U1, U5, U8, U10.

Chapter 3

Circuit Description

This chapter contains the EVM schematic diagram and discusses the various functions on the EVM.

Topic **Page**

3.1 Circuit Function

The following paragraphs describe the function of the individual circuits. Refer to the relevant data sheet for device operating characteristics.

3.1.1 Inputs and Outputs

The ADC has differential analog inputs. These are provided via SMB connectors J10 and J11 on the EVM, and can be configured in two ways:

- For true differential input, the positive differential signal is connected to J10 and the negative differential signal is connected to J11. The jumpers on the board are then configured as: H7 (1–2), H8 (2–3), and H9 (1–2). The inputs have 50-Ω terminators.
- For single-ended input, a positive signal is applied to connector J10. The jumpers on the board are then configured as: H7 (2–3), H8 (1–2), and H9 (2–3). Transformer T2 performs the single-ended to differential signal conversion. In this mode, the 50-Ω terminators R27 and R28 perform impedance matching to attain the best distortion performance, but the signal source must be able to drive the 25-Ω load. A 50-Ω source can be used if R27 is removed from the board, with a resulting marginal increase in distortion.

SMB connector J3 can be used to input a clock signal to the board from an external source. If the source is not at the correct dc level required to input to the 74AHC14 hex inverter IC (U7), then it can be ac-coupled through C21, with the dc level trimmed using potentiometer P1 if necessary.

The EVM is designed to comply with the TI TMS320C6000 daughtercard specification. The pinout used is listed in Tables 3–1 and 3–2. In order to make the EVM compatible with both the TMS320C6XXX and TMS320C5XXX motherboard design environments, the EVM chip enable is derived from the following sources:

- $\overline{\text{OUT\_CE}}$ (J1 P78) for the TMS320C6XXX, by inserting a jumper link in position 2–3 of H15.
- OUT_IS (J2 P70) for the TMS320C5XXX, by inserting a jumper link in position 1–2 of H15.

For further explanation of the daughtercard connector interface, refer to the relevant motherboard user's guide or daughtercard specification.

Buffers U1, U2, and U3 are used to avoid bus contention and provide a degree of noise isolation from the DSP motherboard.

Table 3–1. Daughtercard Connector J1

J1 Pin	Name	Function	J1 Pin	Name	Function
1		NC	41	+3.3V	POWER
2		NC	42	+3.3V	POWER
3		NC	43		NC
4		NC	44		NC
5		NC	45		NC
6		NC	46		NC
7		NC	47		NC
8		NC	48		NC
9		NC	49		NC
10		NC	50		NC
11	GND	GND	51	GND	GND
12	GND	GND	52	GND	GND
13		NC	53	BIDIR_ED15	DATA
14		NC	54	BIDIR_ED14	DATA
15		NC	55	BIDIR_ED13	DATA
16		NC	56	BIDIR_ED12	DATA
17		NC	57	BIDIR_ED11	DATA
18		NC	58	BIDIR_ED10	DATA
19		NC	59	BIDIR_ED9	DATA
20	OUT_EA6	Address	60	BIDIR_ED8	DATA
21		NC	61	GND	GND
22		NC	62	GND	GND
23	OUT_EA5	Address	63	BIDIR_ED7	DATA
24	OUT_EA4	Address	64	BIDIR_ED6	DATA
25	OUT_EA3	Address	65	BIDIR_ED5	DATA
26	OUT_EA2	Address	66	BIDIR_ED4	DATA
27		NC	67	BIDIR_ED3	DATA
28		NC	68	BIDIR_ED2	DATA
29		NC	69	BIDIR_ED1	DATA
30		NC	70	BIDIR_ED0	DATA
31	GND	GND	71	GND	GND
32	GND	GND	72	GND	GND
33		NC	73	$\overline{\text{OUT\_ARE}}$	READ STROBE
34		NC	74	$\overline{\text{OUT\_AWE}}$	WRITE STROBE
35		NC	75		NC
36		NC	76		NC
37		NC	77		NC
38		NC	78	$\overline{\text{OUT_CE}}$_6	CHIP ENABLE
39		NC	79	GND	GND
40		NC	80	GND	GND

Table 3–2. Daughtercard Connector J2

J2 Pin	Name	Function	J2 Pin	Name	Function
1		NC	41		NC
2		NC	42		NC
3	GND	GND	43	GND	GND
4	GND	GND	44	GND	GND
5		NC	45	OUT_TOUT	TIMER INPUT
6		NC	46	IN_TINP	TIMER OUTPUT
7	GND	GND	47		NC
8	GND	GND	48		NC
9		NC	49		NC
10		NC	50		NC
11		NC	51	GND	GND
12		NC	52	GND	GND
13		NC	53	IN_EXT_INT	INTERRUPT
14		NC	54		NC
15		NC	55		NC
16		NC	56		NC
17		NC	57		NC
18		NC	58		NC
19		NC	59		NC
20		NC	60		NC
21		NC	61	GND	GND
22		NC	62	GND	GND
23		NC	63		NC
24		NC	64		NC
25	GND	GND	65		NC
26	GND	GND	66		NC
27		NC	67		NC
28		NC	68		NC
29		NC	69		NC
30		NC	70	OUT_IS	CHIP ENABLE
31	GND	GND	71		NC
32	GND	GND	72		NC
33		NC	73		NC
34		NC	74		NC
35		NC	75	GND	GND
36		NC	76	GND	GND
37	GND	GND	77		GND
38	GND	GND	78		NC
39		NC	79	GND	GND
40		NC	80	GND	GND

3.1.2 Clock Options

The EVM provides flexibility for the source of the ADC's conversion clock. This clock can come from an external source as previously described, from a standard DIL14 HCMOS crystal oscillator module populating U6, or from a U108-style surface-mount oscillator populating U10.

Note:

Ensure that the crystal oscillator module selected can operate at the +3.3-V supply voltage.

Supplying the ADC clock using a timer output (OUT_TOUT J2 P45) is an option provided for applications that require synchronization of the ADC conversion to the DSP on the motherboard. This is selected using a jumper link on H6 (1–2).

A summary of the jumper link settings required for the various clock options is shown in Table 3–3.

Table 3–3. Jumper Settings for Clock Options

Clock Options	Onboard Oscillator	External AC Coupled Via J3	External DC Coupled Via J3	DSP Timer Via OUT_TOUT
H 3	1–2	–	2–3	–
H 5	2–3	1–2	2–3	–
H 6	2–3	2–3	2–3	1–2

3.1.3 References

The EVM relies on the THS14xx ADC using its on-chip reference.

3.1.4 Power

Power is supplied to the EVM via 4-mm banana sockets and the daughtercard connectors. This provides flexibility and a trade-off between convenience and performance as follows:

- For best performance: Use a separate low-noise analog power supply connected to J7 (+3.3 V) and J12 (GND). Use the digital supply from the daughtercard connectors.
- For good performance: Use the daughtercard connectors for both analog and digital supplies. This is achieved by inserting a jumper link in H14 on the EVM.
- For stand-alone operation: Use a separate low-noise analog power supply connected to J7 (+3.3 V) and J12 (GND) and a separate low-noise digital power supply connected to J8 (+3.3 V) and J13 (GND).

3.1.5 Interrupts

The ADC produces two signals that can be used to interrupt a DSP:

- ❑ INT: This is routed via buffer U3 to the IN_EXT_INT pin on the daughtercard connector.
- ❑ FOVL: This is routed via buffer U3 to both BIDIR_ED14 and IN_TINP. This gives the DSP code developer flexibility on using the signal in their system.

Note:

This function is only supported on the THS14F01 and THS14F03. Refer to the device datasheet.

3.1.6 Hardware Loopback

The EVM can be configured by insertion of jumper links H12 and H13 to go into a hardware loopback mode. This is primarily intended so that the board can be put into an operational state when it is powered up stand-alone. The control logic for this is explained in section 3.1.8.

3.1.7 ADC Write/Read Cycle

The following daughtercard address and control lines are mapped directly to the ADC when performing a write to or a read from the device:

- ❑ OUT_EA2 is mapped to A0
- ❑ OUT_EA3 is mapped to A1
- ❑ $\overline{\text{OUT\_AWE}}$ is mapped to WRB
- ❑ $\overline{\text{OUT\_ARE}}$ is mapped to OEB

These, combined with the address decode logic described in section 3.1.8, allow direct control of the ADC operation via the normal write/read cycles of a DSP.

3.1.8 Address Decode Logic

The EVM uses a Xilinx XC9536 (U9) CPLD to perform address decode and control. The CPLD decodes the daughtercard address and strobe lines to control the ADC chip select and the buffer direction and output-enable states. The Abel code used to generate the logic is given in Listing 3–1**.**

3.2 Address Decode and Control Logic Listing

Module Address

Address Decode for THS14xx/5691

This file performs the address decode and logic control for the THS14xx_5691_EV1_Rev2 PCB.

The equivalent truth tables are:

Chip Select Decode

OUT_EA6	OUT_EA5	OUT_EA4	OUT_EA3	OUT_EA2	$\overline{\text{OUT\_CE}}$	ADCCSB	DACCSB
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	0	0	1	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
All other combinations						1	1

Buffer Control Decode

Operation	ADCCSB	DACCSB	OUT_ARE	OUT_AWE	DBUFOEB	ABUFDIR	ABUFDIRB	ABUFOEB	DACMODE
Write to DAC	1	0	1	0	0	1	1	1	1
Write to ADC	0	1	1	0	1	0	1	0	1
Read from ADC	0	1	0	1	1	1	0	0	1
Loop back	0	0	0	1	0	1	0	0	0
All other combinations					1	1	1	1	1

Note: Bit 13 of the data bus to the DAC is inverted when in loopback mode to account for the DAC powering up in 2s-complement input mode while the ADC powers up in binary mode.

Inputs

OUT_EA2 pin; Address bit from DSP board

OUT_EA3 pin; Address bit from DSP board

OUT_EA4 pin; Address bit from DSP board

OUT_EA5 pin; Address bit from DSP board

OUT_EA6 pin; Address bit from DSP board

BIDIR_ED13 pin; Data bit 13

$\overline{\text{OUT\_AWE}}$ pin; DSP write enable

OUT_ARE pin; DSP read enable

OUT_CE pin; DSP chip enable

Outputs

DBUFOEB pin istype 'com'; DAC buffers 3-state control

ABUFDIR pin istype 'com'; ADC buffer direction control

ABUFDIRB pin istype 'com'; ADC FOVL flag buffer 3-state control

ABUFOEB pin istype 'com'; ADC data bus buffer 3-state control

DACMODE pin istype 'com'; DAC mode

Mod_ED13 pin istype 'com'; Selectively invert BIDIR_ED13 for loop-back

ADCCSB pin istype 'com'; Chip select for ADC

Equations

```
!ADCCSB = (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE);
!DACCSB = (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE);
!DBUFOEB = (ADCCSB and !DACCSB and OUT_ARE and !OUT_AWE)
# (!ADCCSB and !DACCSB and !OUT_ARE and OUT_AWE); Done in longhand so signal path is only once
  through device
!DBUFOEB = (!((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and ((OUT_EA6
and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and OUT_ARE and
  !OUT_AWE)
# (((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and ((OUT_EA6
  and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
```

```
# (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !OUT_ARE
   and OUT_AWE);
!ABUFDIR = (!ADCCSB and DACCSB and OUT_ARE and !OUT_AWE); Done in longhand so signal path is only
  once through device
!ABUFDIR = (((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !((OUT_EA6
  and !OUT_EA5 and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and OUT_ARE and
  !OUT_AWE);
!ABUFDIRB = (!ADCCSB and DACCSB and !OUT_ARE and OUT_AWE)
# (!ADCCSB and !DACCSB and !OUT_ARE and OUT_AWE); Done in longhand so signal path is only once
  through device
!ABUFDIRB = (((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !((OUT_EA6
and !OUT_EA5 and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !OUT_ARE
   and OUT_AWE);
!ABUFOEB = (!ADCCSB and DACCSB and OUT_ARE and !OUT_AWE)
# (!ADCCSB and DACCSB and !OUT_ARE and OUT_AWE)
# (!ADCCSB and !DACCSB and !OUT_ARE and OUT_AWE); Done in longhand so signal path is only once
  through device
!ABUFOEB = (((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !((OUT_EA6
  and !OUT_EA5 and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and OUT_ARE and
  !OUT_AWE)
# (((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !((OUT_EA6
  and !OUT_EA5 and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !OUT_ARE
   and OUT_AWE)
# (((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
# (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
# (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and ((OUT_EA6
  and !OUT_EA5 and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
```

```
 # (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !OUT_ARE
     and OUT_AWE);
 !DACMODE = (!ADCCSB and !DACCSB and !OUT_ARE and OUT_AWE); Done in longhand so signal path is
    only once through device
 !DACMODE = (((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)# (OUT_EA6 and
   OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and ((OUT_EA6 and !OUT_EA5
   and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !OUT_ARE
   and OUT_AWE);Mod_ED13 = (!BIDIR_ED13 and ((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3
   and !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and ((OUT_EA6
   and !OUT_EA5 and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !OUT_ARE
 and OUT_AWE)
 # (BIDIR_ED13 and !(((OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and
   !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and !OUT_EA4 and OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and ((OUT_EA6
   and !OUT_EA5 and OUT_EA4 and !OUT_EA3 & !OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and !OUT_EA5 and OUT_EA4 and !OUT_EA3 and OUT_EA2 and !OUT_CE)
 # (OUT_EA6 and OUT_EA5 and !OUT_EA4 and !OUT_EA3 and !OUT_EA2 and !OUT_CE)) and !OUT_ARE
   and OUT_AWE));

 END
```

3.3 THS14xx Software Example for C6x0x DSPs

The THS14xx example is available for C6x0x DSPs. It supports a single THS1408, THS1403, THS1401, THS14F03 or THS14F01. It is written in C.

The THS14xx must be connected to the external memory interface of the DSP. The addresses can be specified for all the four registers of the THS14xx by changing the following #define values in the source code:

```
#define    ADC_RES_ADDR      (unsigned int *) (0x01400040)
#define    ADC_PGA_ADDR      (unsigned int *) (0x01400044)
#define    ADC_OFF_ADDR      (unsigned int *) (0x01400048)
#define    ADC_CTL_ADDR      (unsigned int *) (0x0140004C)
```

The example supports both FIFO and FIFO-less operation. It uses one DMA channel (DMA0) and one interrupt (DMA_Channel_0). The interrupt service routine must be included in the interrupt dispatch mechanism by the user. It is recommended to use DSP/BIOS (default).

3.3.1 Constraints

- Only one THS14xx is supported because of system performance limitations and limited DSP resources.
- For FIFO operation, the INT pin of the THS14F0x must be connected to EXT_INT7. This input is used to synchronize the DMA with the interrupt signal from the ADC. The EVM maps the INT signal to the EXT_INT7 input of the DSP, if the C6201 EVM from TI is used.
- For non-FIFO operation the master clock signal of the THS140x must be connected to EXT_INT7. If the DSP timer output is used, the user must modify the DMA_CTRL_VALUE field to synchronize the DMA with the timer.

3.3.2 Device Selection

```
void ths140x_configure(unsigned int pga, unsigned int off-
set, unsigned int ctrl);
```

The example requires that the onboard oscillator of the EVM is used as a sample clock. Therefore, the example is independent on the speed grade used (THS1408, THS1403, THS1401 or THS14F03, THS14F01). FIFO or non-FIFO operation is determined by the control value (FC, bit #4 of the control register). The control value is defined by:

```
#define  ADC_CTRL  (0x08F8)
/*                  8     2's complement output */
/*                   F    OFF = 1, offset correction enabled */
/*                        IP = 1, INT high active */
/*                        FP = 1, FIFO OVL high active */
/*                        FC = 1, FIFO enabled */
/*                    8   F3:0 = 8, 16 word trigger level */
```

In this example, the FIFO is enabled.

3.3.3 Configuration

```
void ths140x_configure(unsigned int pga, unsigned int
offset, unsigned int ctrl);
```

This function initializes the ADC (PGA, offset register , control register).

3.3.4 Read Block Function

```
void ths140x_rblock(void *pData, unsigned long ulCount,
void (*callback) (void *));
```

This function starts a read block transfer from the THS14xx to the DSP. `pData` points to the destination memory. The results will be stored in 32–bit words. The THS14xx has 14-bit outputs. The output is not sign extended. Sign extension must be performed by the signal-processing algorithm.

If the routine is used with the FIFO, care must be taken to avoid a FIFO overrun condition. Such a condition can happen, if the dc_rblock function is called over and over again in a callback function to establish continuous conversion and if the ulCount number is not an integer multiple of the FIFO trigger level. In this case, the FIFO will fill up quickly because (ulCount modulo FIFO trigger level) number of samples will be left in the FIFO after every block transfer.

If a FIFO overrun occurs, the FIFO must be reset by clearing the FIFO reset bit within the THS14F0x control word before another block transfer is initiated. Otherwise the callback function will never be executed.

3.3.5 Interrupt Service Routine

The THS14xx data converter example uses one interrupt service routine for both FIFO and nonFIFO devices. The interrupt service routine is called upon the end of a DMA transfer.

If a FIFO device (THS14F03 or THS14F01) is used, the DMA transfer count is set to the trigger level of the FIFO. Every time the number of samples in the FIFO reaches the trigger level, a short DMA burst is triggered followed by an interrupt. The interrupt service routine checks if there are samples left in the block transfer and sets up a new DMA burst, if necessary.

If a non-FIFO device (THS1408, THS1403, THS1401) is used, the DMA transfer count is set to the number of samples in the block transfer. The DMA must be synchronized with the master clock of the ADC by connecting the ADC master clock to the EXT_INT7 pin of the DSP. After the DMA controller is done with the whole block transfer, one interrupt is generated which terminates the DMA and calls the callback function specified by the ths140x_rblock call.

3.3.6 Example Main Function

The following steps are necessary in order to use the example with DSP/BIOS:

- ❑ Add the interrupt functions to the appropriate interrupt dispatch mechanism. One way to do this is to use the DSP/BIOS:
  - ■ Add a configuration database to the project or open an existing one.
  - ■ Go to the HWI module and insert the following two functions:
    - ■ `_ths140x_risr in HWI_INT8, interrupt source: DMA_Channel_0, Use Dispatcher = false`

If DSP/BIOS is not used, the ISR must be included in the implemented interrupt dispatch mechanism.

The example main function transfers a block of samples from the THS14xx to the DSP memory.

```
/*********************************************************************/
/* Demo program for the THS14xx EVM connected to an C6201 EVM.       */
/* An XDS510 has been used as the emulation interface.               */
/* The following jumper setting should be used:                      */
/* C6201 EVM (SW1–12): ON–ON–OFF–ON–OFF–ON–OFF–ON–ON–ON–ON–ON         */
/* THS14xx EVM:                                                      */
/*    H1  open    /  H2  open  /  H3  1–2  /  H4  open               */
/*    H5  2–3     /  H6  2–3   /  H7  2–3  /  H8  1–2                */
/*    H9  2–3     /  H10 open  /  H11 open /  H12 open               */
/*    H13 open    /  H14 closed/  H15 2–3                            */
/* Supply voltage from DSP, Input ADCIN J10                          */
/* Sample clock from EVM oscillator                                  */
/* AD converter address: 0x014xxxxx                                  */
/* DSP/BIOS                                                          */
/*                                                                   */
/* This simple demo periodically takes 1024 samples.                 */
/* The samples are sign extended. The buffer can be plotted using    */
/* the View–>Graph–>Time/Frequency... menu option of CCS             */
/*********************************************************************/

/* This example is based on DSP/BIOS. DSP/BIOS is used for           */
/* interrupt dispatching. The DSP/BIOS specific parts (include       */
/* files and HWI_xxx) must be changed depending on the implemented   */
/* interrupt mechanism                                               */

/* DSP/BIOS headers */
#include <C6x.h>
#include <std.h>
#include <hwi.h>

/* DSP registers */
#define EMIF 0x01800004
#define DMA_CTRL 0x01840000
#define DMA_CTRL2 0x01840008
#define DMA_SRC 0x01840010
#define DMA_DST 0x01840018
#define DMA_TCT 0x01840020

/* DMA control values */
#define COUNT_VALUE 0x00020000
#define DMA_CTRL_VALUE 0x0701c041
/*                      0       no auto reload */
/*                       7      EMOD = 0, DMA keeps running on emulation stop */
/*                              FS = 1, RSYNC event used to synchronize element */
/*                              TCINT = 1, transfer controller interrupt enabled */
/*                              PRI = 1, DMA priority over CPU */
/*                        01    WSYNC = 00000, no synchronization */
/*                         1c   RSYNC = 00111, ext_int 7 */
/*                              INDEX = 0, use DMA global index register A */
/*                              CNT reload = 0, use DMA global count register A */
/*                           0  SPLIT = 00, disabled */
/*                              ESIZE = 00, 32–bit */
/*                            4 DST DIR = 01, increment by 4 bytes */
/*                              SRC DIR = 00, no modification */
/*                             1START = 00, start without autoinitialization */

/* THS14xxx parameters */
#define    ADC_CTRL         (0x08F8)
/*                             8   2's complement output */
/*                              F  OFF = 1, offset correction enabled */
```

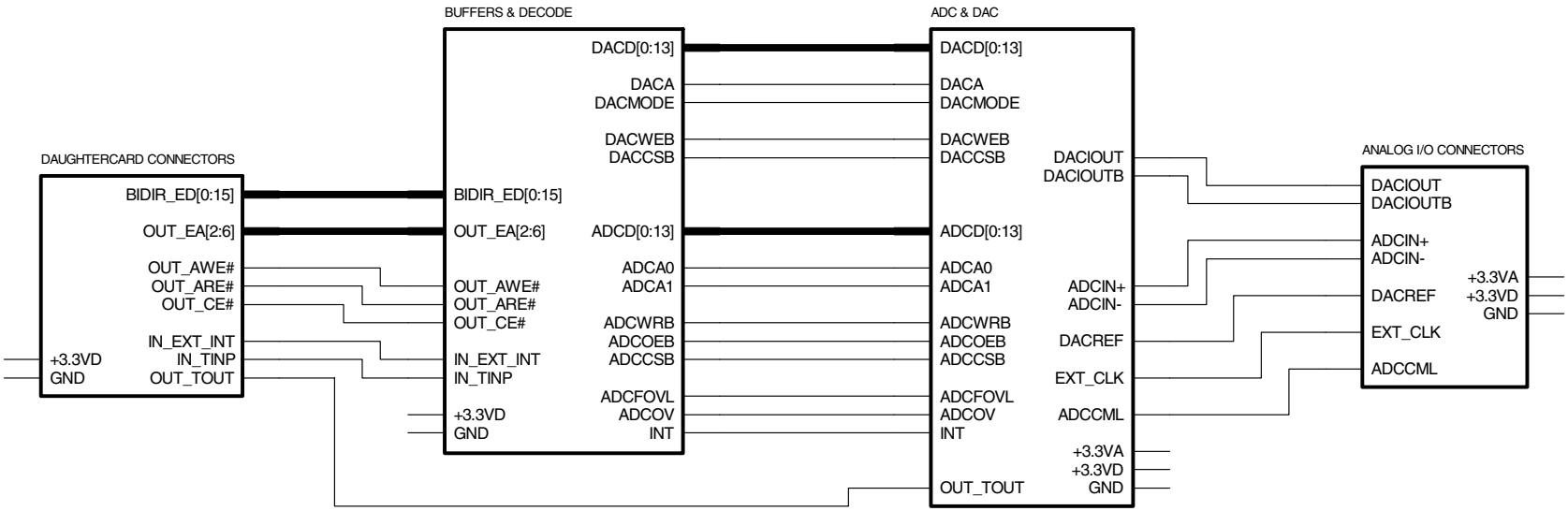
```
/*                              IP = 1, INT high active */
/*                              FP = 1, FIFO OVL high active */
/*                              FC = 1, FIFO enabled */
/*                            8 F3:0 = 8, 16 word trigger level */
#define    ADC_PGA        (0)
#define    ADC_OFFSET     (0)  /* LSBs   */
#define    ADC_RES_ADDR   (unsigned int *) (0x01400040)
#define    ADC_PGA_ADDR   (unsigned int *) (0x01400044)
#define    ADC_OFF_ADDR   (unsigned int *) (0x01400048)
#define    ADC_CTL_ADDR   (unsigned int *) (0x0140004C)

/* receive ISR data */
int *pRData = 0;           /* points to current destination memory */
unsigned int uiRCount = 0; /* samples left in block transfer */
unsigned int uiDCount = 0; /* DMA segment size */
unsigned int uiThreshold = 0; /* FIFO trigger level */
/* pointer to callback function (called at the end of a block transfer */
void (*r_callback) (void *) = 0;

int iDMACtrl = DMA_CTRL_VALUE;
/* receive interrupt service routine (DMA0) */
interrupt void ths140x_risr(void)
{
   pRData += uiDCount;     /* point to next DMA destination segment */
   uiRCount –= uiDCount;   /* subtract last DMA segment size from */
                           /* receive counter */
   if (!uiRCount) /* check, if the whole block has been transfered */
   {
      *(int*) DMA_CTRL &= !0x1;  /* disable DMA */
      if (r_callback)      /* call callback function at the end */
                           /* of the block transfer */
         r_callback(0);
      return;
   }

      /* calculate new DMA segment size and set new DMA values*/
      if (uiRCount < uiThreshold)
         uiDCount = uiRCount;
      else
            uiDCount = uiThreshold;
      *(int*) DMA_TCT = 0x00020000 | uiDCount;
      *(int*) DMA_DST = (int) pRData;
      *(int*) DMA_CTRL2 = 0x00000008;
/*                                  8 FRAME IE = 1, frame interrupt en-
abled */
      *(int*) DMA_CTRL = iDMACtrl;
   IER |= 0x0103;
}

/* write PGA, offset and control value to THS140x */
void ths140x_configure(unsigned int pga, unsigned int offset, unsigned
int ctrl)
{
   *ADC_PGA_ADDR = pga;
   *ADC_OFF_ADDR = offset;
   *ADC_CTL_ADDR = ctrl;
}
```

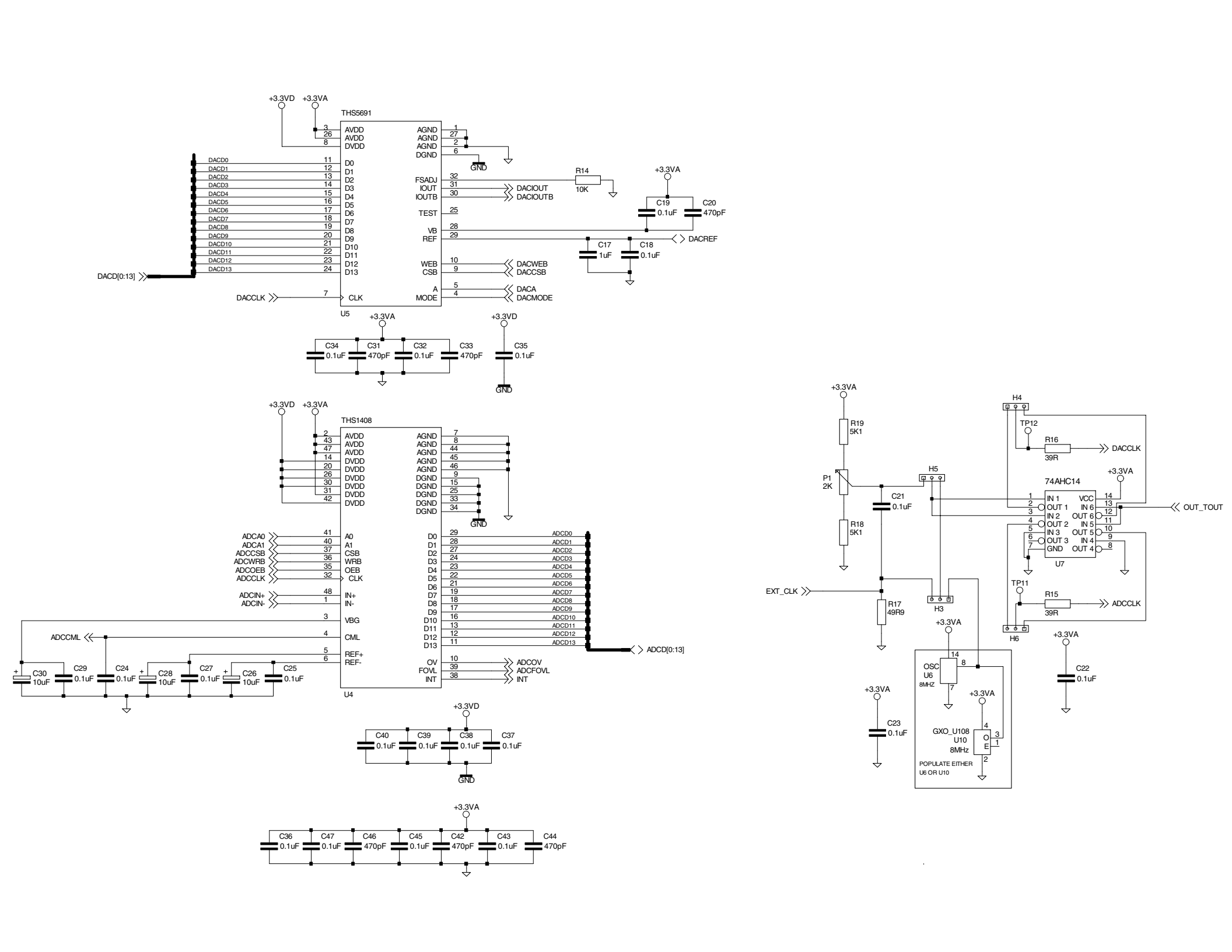
```
/* read block of samples from THS140x and store in pData. */
/* ulCount = length of block */
/* callback = address of callback fctn called upon end of block transfer
*/
void ths140x_rblock(void *pData, unsigned long ulCount, void (*callback)
(void *))
{
   /* store parameters to make them available to isr */
   pRData = pData;
   r_callback = callback;
   uiRCount = ulCount;
   iDMACtrl = DMA_CTRL_VALUE;

   /* if FIFO is used, divide block transfer into segments */
   /* of the size of the FIFO trigger level */
   if (ADC_CTRL & 0x10)
   {
      uiThreshold = (ADC_CTRL & 0xF) * 2;
      iDMACtrl |= 0x04000000; /* whole frame synchronized with RSYNC */
   }
   else
      uiThreshold = ulCount;

   /* calculate DMA segment size */
   if (uiRCount < uiThreshold)
      uiDCount = uiRCount;
   else
      uiDCount = uiThreshold;
   *(int*) DMA_TCT = 0x00020000 | uiDCount;
   *(int*) DMA_SRC = (int) ADC_RES_ADDR;
   *(int*) DMA_DST = (int) pRData;
   *(int*) DMA_CTRL2 = 0x00000008;
   *(int*) DMA_CTRL = iDMACtrl;
   HWI_enable();
   CSR |= 1;
   IER |= 0x0103;
}

/* pData is the sample buffer */
int pData[1024];

/* iDone is the semaphore to synchronize the ISR function with the main
*/
/* thread. The semaphore is set by the callback function */
volatile int iDone;

/* this function is called upon completion of a block transfer. */
/* It is part of the driver interrupt service routine and */
/* therefore time critical */
void callback(void *p)
{
   iDone = 1;
}

/* main routine called after system initialization */
main()
{
```

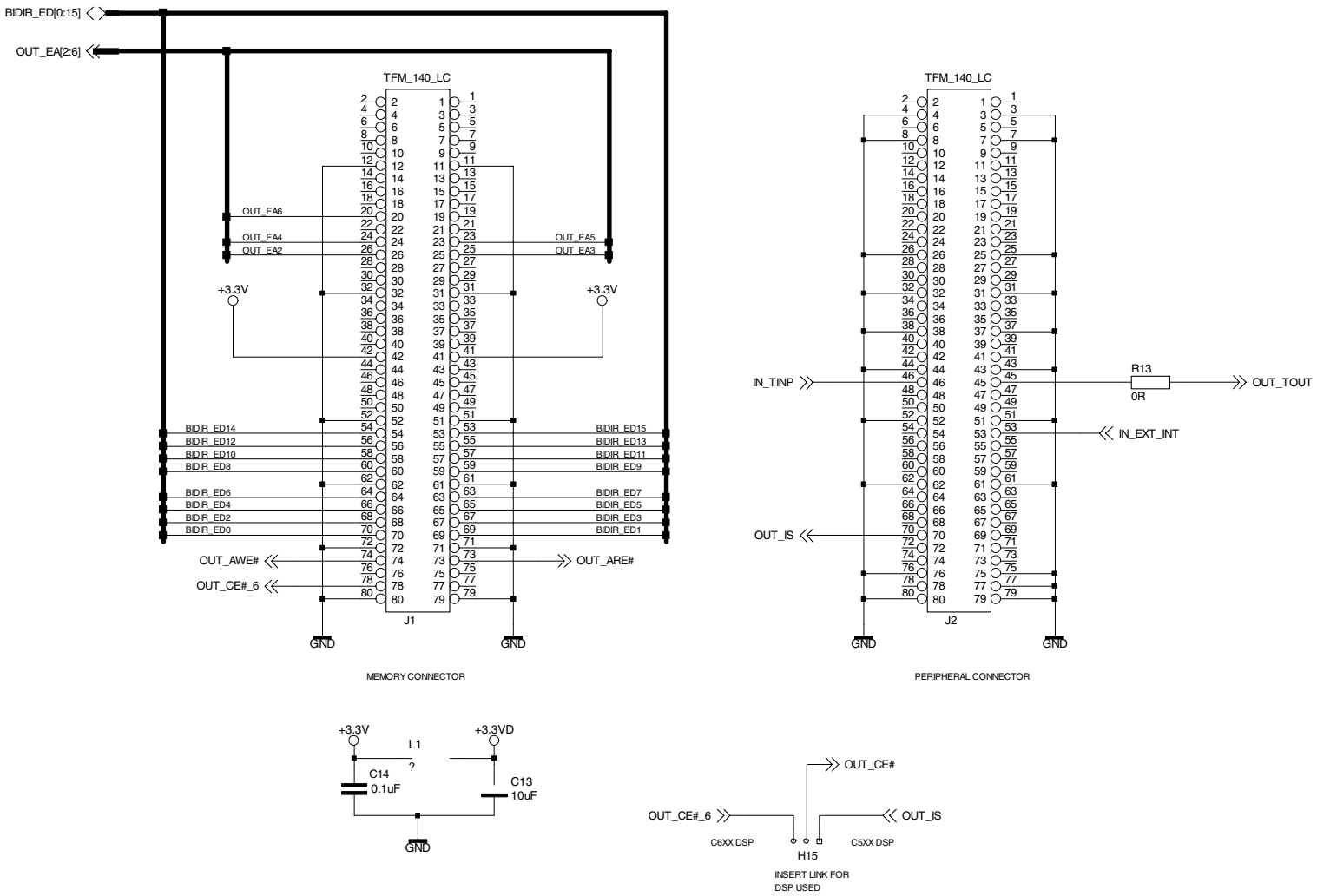
```
   int i;
   int iSample;

   /* clear sample buffer */
   for (i = 0; i < 1024; i++)
      pData[i] = 0;

   /* set EMIF wait states for THS14xxx */
   *(long*) EMIF = 0x21e1c423;

   while (1)
   {
      /* reset THS14F0x FIFO to avoid stall because of eventual */
      /* previous FIFO overflow */
      ths140x_configure(ADC_PGA, ADC_OFFSET, 0);

      /* write original control value */
      ths140x_configure(ADC_PGA, ADC_OFFSET, ADC_CTRL);

      /* reset semaphore iDone and start read block function */
      iDone = 0;
      ths140x_rblock(pData, 1024, callback);

      /* the DSP returns immediately without waiting for the block trans-
fer */
      /* to finish. iDone will be set by the callback function. */
      /* For now, just wait until the DMA is done */
      while (!iDone) {};

      /* sign extend the samples */
         for (i = 0; i < 1024; i++)
         {
            iSample = pData[i] & 0x3FFF;
            if (iSample > 8191)
               iSample –= 16384;
            pData[i] = iSample;
         }
      /* set a break–point to view the sample buffer with the */
      /* CCS View–>Graph–>Time/Frequency... menu option */
   }
}
```

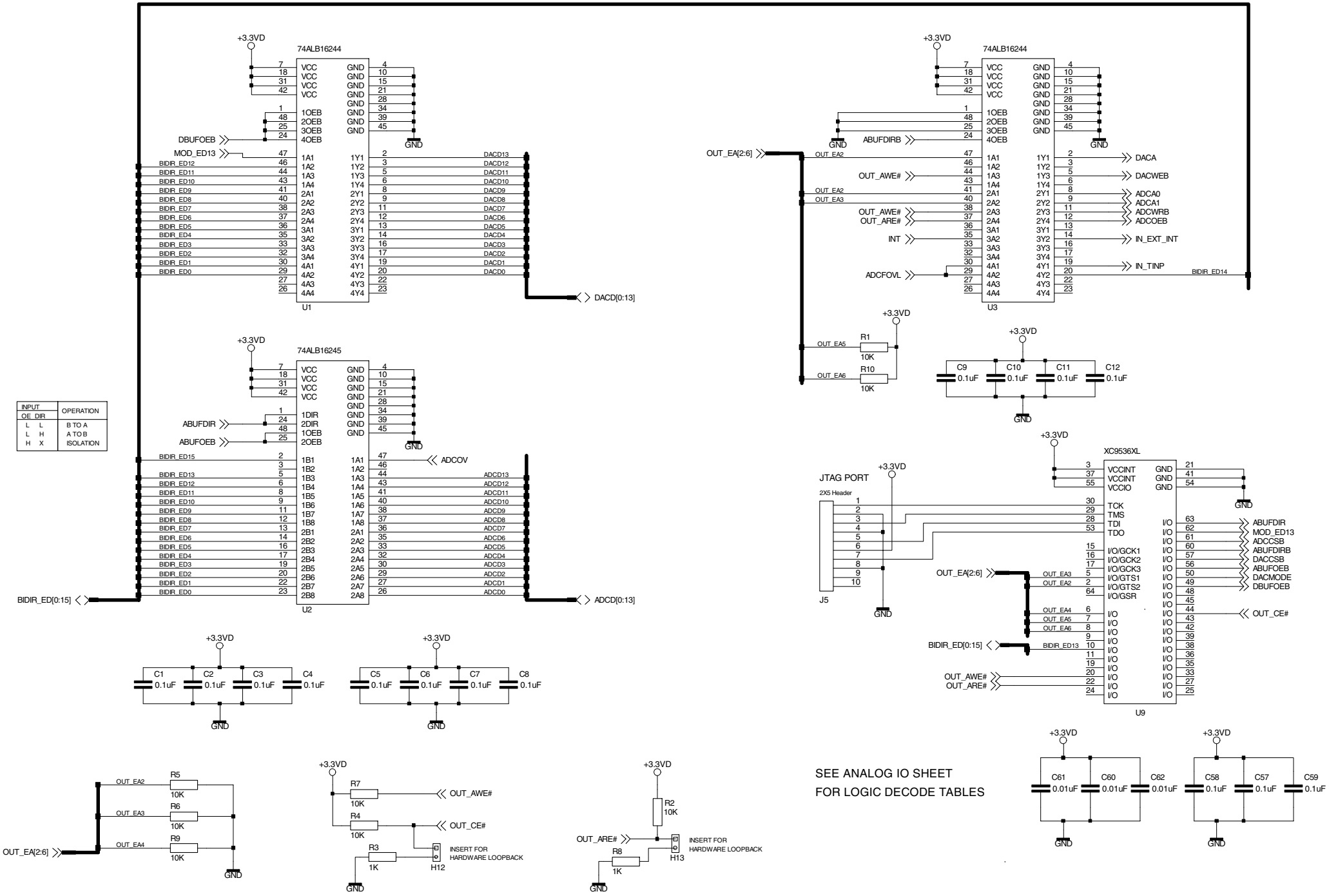
3.4 Schematic Diagrams

This section contains the schematic diagrams for the EVM.

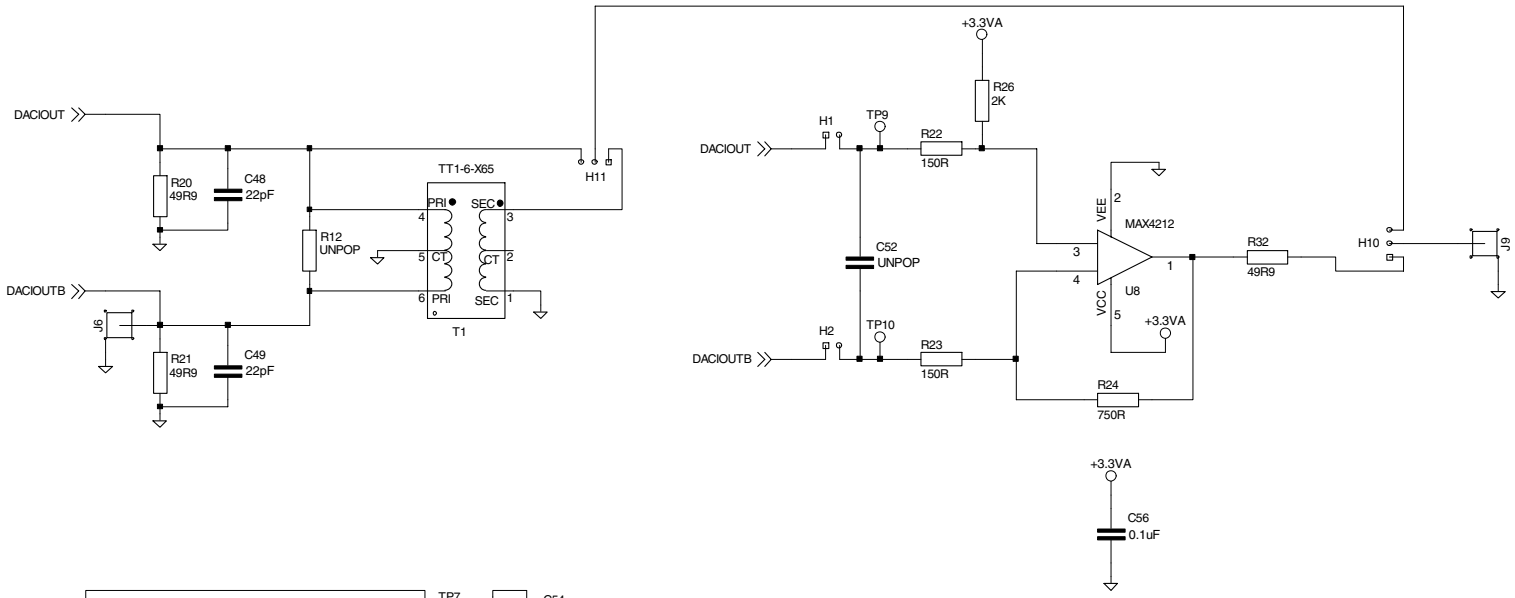

DAUGHTERCARD CONNECTORS
BIDIR_ED[0:15]
OUT_EA[2:6]
OUT_AWE#
OUT_ARE#
OUT_CE#
IN_EXT_INT
IN_TINP
OUT_TOUT
+3.3VD
GND
BUFFERS & DECODE
BIDIR_ED[0:15]
OUT_EA[2:6]
OUT_AWE#
OUT_ARE#
OUT_CE#
IN_EXT_INT
IN_TINP
+3.3VD
GND
DACD[0:13]
DACA
DACMODE
DACWEB
DACCSB
ADCD[0:13]
ADCA0
ADCA1
ADCWRB
ADCOEB
ADCCSB
ADCFOVL
ADCOV
INT
ADC & DAC
DACD[0:13]
DACA
DACMODE
DACWEB
DACCSB
ADCD[0:13]
ADCA0
ADCA1
ADCWRB
ADCOEB
ADCCSB
ADCFOVL
ADCOV
INT
OUT_TOUT
DACIOUT
DACIOUTB
ADCIN+
ADCIN-
DACREF
EXT_CLK
ADCCML
+3.3VA
+3.3VD
GND
ANALOG I/O CONNECTORS
DACIOUT
DACIOUTB
ADCIN+
ADCIN-
DACREF
EXT_CLK
ADCCML
+3.3VA
+3.3VD
GND

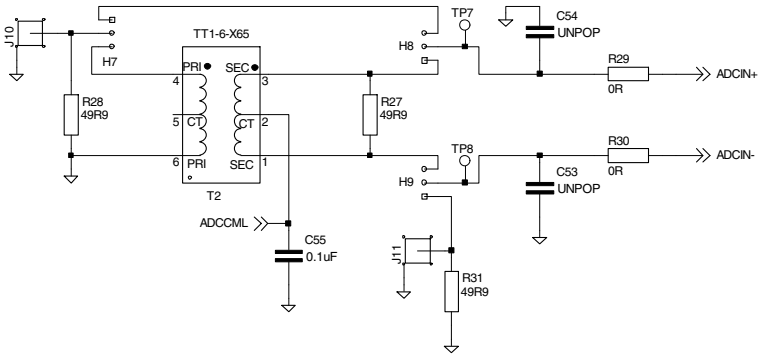

THS5691
+3.3VD
+3.3VA
AVDD
DVDD
AGND
DGND
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
CLK
FSADJ
IOUT
IOUTB
TEST
VB
REF
WEB
CSB
A
MODE
R14
10K
DACIOUT
DACIOUTB
DACREF
DACWEB
DACCSB
DACA
DACMODE
DACD[0:13]
DACCLK
U5
C19
C20
C17
C18
C34
C31
C32
C33
C35
GND
THS1408
ADCA0
ADCA1
ADCCSB
ADCWRB
ADCOEB
ADCCLK
ADCIN+
ADCIN-
A0
A1
CSB
WRB
OEB
CLK
IN+
IN-
VBG
CML
REF+
REF-
OV
FOVL
INT
ADCCML
ADCD0
ADCD1
ADCD2
ADCD3
ADCD4
ADCD5
ADCD6
ADCD7
ADCD8
ADCD9
ADCD10
ADCD11
ADCD12
ADCD13
ADCD[0:13]
ADCOV
ADCFOVL
INT
U4
C30
C29
C24
C28
C27
C26
C25
C40
C39
C38
C37
C36
C47
C46
C45
C42
C43
C44
R19
5K1
P1
2K
R18
5K1
EXT_CLK
R17
49R9
C21
0.1uF
H5
H3
H4
H6
TP12
TP11
R16
R15
39R
DACCLK
ADCCLK
74AHC14
IN 1
OUT 1
IN 2
OUT 2
IN 3
OUT 3
GND
VCC
IN 6
OUT 6
IN 5
OUT 5
IN 4
OUT 4
U7
OUT_TOUT
OSC
U6
8MHZ
GXO_U108
U10
8MHz
O
E
POPULATE EITHER
U6 OR U10
C22
C23


BIDIR_ED[0:15]
OUT_EA[2:6]
TFM_140_LC
OUT_EA6
OUT_EA4
OUT_EA2
OUT_EA5
OUT_EA3
+3.3V
+3.3V
BIDIR_ED14
BIDIR_ED12
BIDIR_ED10
BIDIR_ED8
BIDIR_ED6
BIDIR_ED4
BIDIR_ED2
BIDIR_ED0
BIDIR_ED15
BIDIR_ED13
BIDIR_ED11
BIDIR_ED9
BIDIR_ED7
BIDIR_ED5
BIDIR_ED3
BIDIR_ED1
OUT_AWE#
OUT_CE#_6
OUT_ARE#
J1
GND
GND
MEMORY CONNECTOR
TFM_140_LC
IN_TINP
R13
0R
OUT_TOUT
IN_EXT_INT
OUT_IS
J2
GND
GND
PERIPHERAL CONNECTOR
+3.3V
L1
?
C14
0.1uF
+3.3VD
C13
10uF
GND
OUT_CE#
OUT_CE#_6
OUT_IS
C6XX DSP
H15
C5XX DSP
INSERT LINK FOR
DSP USED



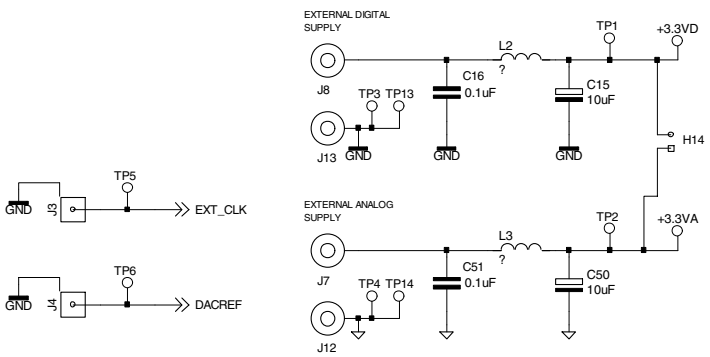
SEE ANALOG IO SHEET
FOR LOGIC DECODE TABLES





BUFFER CONTROL DECODE

OPERATION	ADCCSB	DACCSB	OUT_ARE#	OUT_AWE#	DBUFOEB	ABUFDIR	ABUFDIRB	ABUFOEB	DACMODE
WRITE TO DAC	1	0	1	0	0	1	1	1	1
WRITE TO ADC	0	1	1	0	1	0	1	0	1
READ FROM ADC	0	1	0	1	1	1	0	0	1
LOOPBACK	0	0	0	1	0	1	0	0	0
ALL OTHER COMBINATIONS					1	1	1	1	1



CHIP SELECT DECODE

OUT_EA6	OUT_EA5	OUT_EA4	OUT_EA3	OUT_EA2	OUT_CE#	ADCCSB	DACCSB
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	0	0	1	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
ALL OTHER COMBINATIONS						1	1