Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

ServoPAL (#28824): *Servo Pulser and Timer*

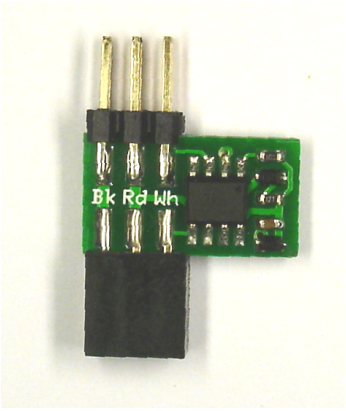
General Description

The ServoPAL is a tiny module that plugs in between your BASIC Stamp and two servo motors to pulse the motors so your PBASIC program doesn't have to. In addition, it provides an "alarm clock" function to perform timing in the background while the BASIC Stamp is busy with other tasks.

Features

- Plugs in between servo headers and servos: no wiring necessary.
- Simplifies PBASIC programming for both standard and continuous-rotation servos.
- Pulses two servos continuously based on single pulses received from the BASIC Stamp.
- Provides an alarm output (200mS to 30 min delay), which can be set by a single pulse.
- All interfacing is done by pulsing: no serial protocols to learn.
- Runs from the servo's power (up to 6.5VDC): no additional power source needed.
- Compact size: stackable side-to-side with additional units on 0.1" servo headers.

What's Included



ServoPAL module

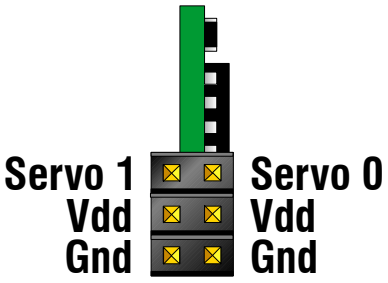
What You Need to Provide

- BASIC Stamp and carrier board (such as the BOE).
- One or two servo motors.

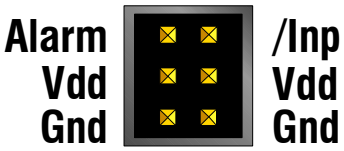
Installation

Installation of the ServoPAL is a simple:

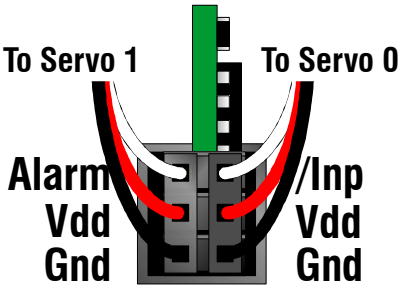
1. Unplug the servo connectors from the servo headers.
2. Plug the ServoPAL into the servo headers. (See diagram below.)
3. Plug the servo connectors into the ServoPAL.



Top View

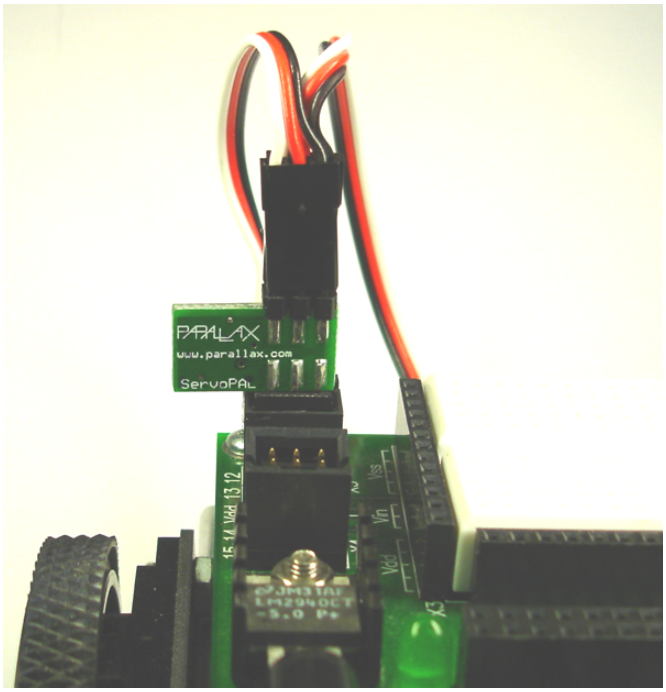


Servo Header



Everything Connected

Below is a photo of the ServoPAL installed on a BOE-Bot's P12/P13 servo header. The wheel servo motor cables are plugged into the ServoPAL. In this case, the **/Inp** input pin is **P12**, the **Alarm** output pin is **P13**, the right servo is **Servo 0**, and the left servo is **Servo 1**.



Hardware Interface and Initialization

Interface to the ServoPAL is realized through its **/Inp** input and **Alarm** "output". When the ServoPAL powers up, both lines are configured as normally-high inputs, pulled up to a nominal +5V through internal 20K to 50K resistances. Commands to the ServoPAL are sent as negative pulses to the **/Inp** pin. If the alarm feature is not used, the BASIC Stamp can use the **Alarm** pin for any other purpose. If this is done, however, you must be careful not to trigger an alarm inadvertently, or else a bus conflict may occur.

The ServoPAL runs autonomously on power obtained from the servo headers. Therefore, it does not reset when the BASIC Stamp resets, and continues to send out pulses during and after reset if it was sending them out before. But it *can* be reset, nonetheless, by sending a pulse of 100mS duration to the **/Inp** pin. Here's the recommended initialization sequence for the ServoPAL in PBASIC:

```
' {$STAMP BS2}
' {$PBASIC 2.5}

nInp  PIN 12              'Define the input pin.
Alarm PIN 13              'Define the alarm pin.

Restart:

INPUT nInp                'Make sure nInp isn't being driven.

DO UNTIL nInp             'Wait for ServoPAL to power up.
LOOP

LOW nInp                  'Set pin to an output and hold it low
PAUSE 100                 '  for 100mS.
HIGH nInp                 'Raise the pin.
```

This sequence accomplishes a couple things:

1. In systems, like the BOE-Bot, where the servo power is switched on after the BASIC Stamp powers up, it will wait for the ServoPAL to be turned on. It does this by setting the **nInp** line to an input and waiting for that line to go high, signifying not only that that the ServoPAL has powered up, but that it has come out of reset and engaged its internal pull-ups.
2. In case the ServoPAL was already running, it then sends a long reset pulse to terminate any servo pulses, to abort any pending alarm, and to ready the ServoPAL for further commands.

Programming Servo Pulses

Servo pulses are programmed in much the same way you'd send pulses directly to a servo motor, using the PBASIC **PULSOUT** command. The only differences are that the pulses are negative-going instead of positive and that both servos are programmed from the same pin. Here's an example that will cause the **Servo 0** output to begin pulsing with 1.5mS pulses. These pulses are automatically repeated every 10-15mS without further intervention from the BASIC Stamp.

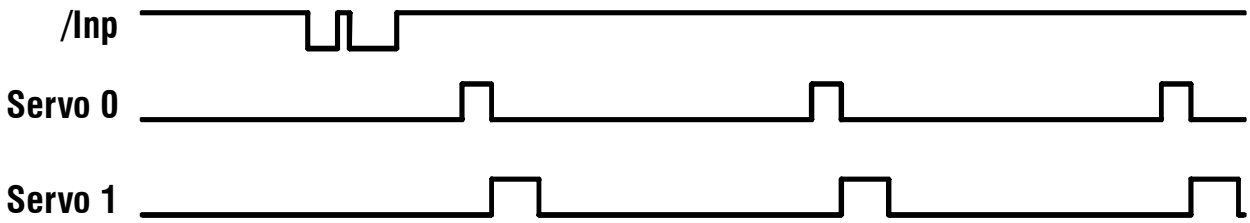
```
PULSOUT nInp, 750
```

Notice that this is identical to the statement that *would* have been used, had the servo been connected directly to the BASIC Stamp. Once the initialization code is executed, your program is primed for negative-going pulses, and you don't need to do anything special to send them. Here's what the waveforms look like that result from the above example:

But what about **Servo 1**? There's only one pin, so how does it get programmed? That's easy: just send another pulse within 1mS of the first one, and that one will be assigned to **Servo 1**:

```
PULSOUT nInp, 750
PULSOUT nInp, 1000
```

The preceding code will program Servo 0 with a pulse width of 1.5mS and Servo 1 with a pulse width of 2.0mS. Here's what the waveforms will look like:

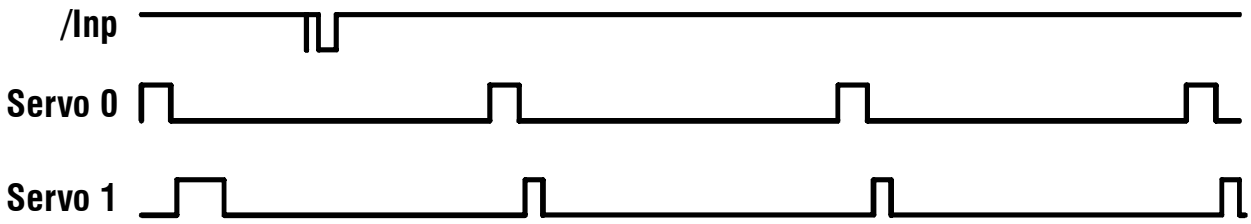


To terminate a sequence of pulses on **/Inp**, just wait at least 2mS before sending another pulse. This signals the ServoPAL that the next pulse is intended for **Servo 0**.

Now, what if you want to program *only* **Servo 1** and leave **Servo 0** alone? That's easy, as well. Just send a very short pulse (4 – 100µS) for **Servo 0** first, then a pulse for **Servo 1**. The ServoPAL will interpret the short pulse as indicating that you don't want to change **Servo 0**'s pulse stream, if one has already been programmed. Here's an example:

```
PULSOUT nInp, 2
PULSOUT nInp, 500
```

Here's what the resulting waveforms look like:

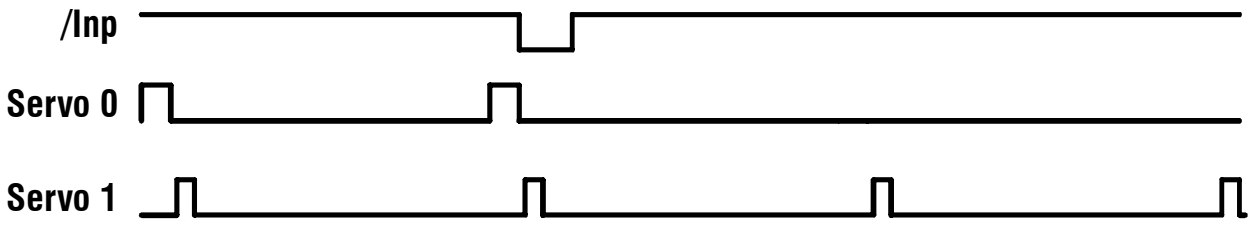


Note: Do not send the ServoPAL any pulses less than 4µS in duration. It could easily become confused, as it might miss them entirely.

And finally, what if you want to terminate a servo's pulse stream? The ServoPAL will only send pulses to a servo that are between approximately 0.5 and 2.5mS. If it encounters a request for pulses well outside this range, it will cease sending pulses to the affected servo. Here's how you can use this to stop sending pulses to a servo:

```
PULSOUT nInp, 2000          'Send a 4mS pulse to Servo 0.
```

Here's the result:



To summarize, here's a table showing the various pulse durations and what they mean:

4 - 100µS	Skip this servo. (Don't change its output.)
0.5 - 2.5mS	Program this pulse width into the affected servo.

4 – 30mS	Kill the servo's output.
100mS or more	Reset the ServoPAL.

Pulse widths outside the individual ranges shown may have unpredictable effects.

Programming the Alarm Function

In addition to servo control, the ServoPAL has an alarm clock (timer) function. You can set an alarm to occur anywhere from 200mS to 30 minutes after it's set (triggered). For programming purposes, the timer is treated by the ServoPAL as a third servo: i.e. after sending pulses for **Servo 0** and **Servo 1**, you send a pulse for the timer. As soon as such a pulse is received, the **Alarm** output is pulled low. It will remain low until the programmed time elapses, whereupon it will again float and be pulled high by means of the internal pullup resistor.

The relation between the pulse width sent by the BASIC Stamp and the actual time programmed into the timer is

Programmed time = Pulse width * 50000 (approximately)

So a pulse width of 4µS yields a timeout of 200mS, and a pulse width of 36000µS yields a timeout of 30 minutes. In terms of **PULSOUT** units, here's the timing for various BASIC Stamps:

	BS1	**BS2**	**BS2e**	**BS2sx**	**BS2p**	**BS2pe**	**BS2px**
Time per **PULSOUT** unit	500mS	100mS	100mS	40mS	40mS	100mS	40mS
Maximum **PULSOUT** units	3600	18000	18000	45000	45000	18000	45000

Here's a BS2 program fragment that sets the timeout period to 5 minutes:

```
PULSOUT nInp, 4           'Skip Servo 0.
PULSOUT nInp, 4           'Skip Servo 1.
PULSOUT nInp, 3000        'Set timeout to 300 seconds.
```

And here's the output waveform:

One additional thing to note about the timer is that it's retriggerable. What this means is that if you set it before a previous setting has timed out, it will start over with the new timeout value. This feature makes the ServoPAL useful as a watchdog timer for the BS2p, BS2pe, and BS2px, which support automatic polling. As long as you keep resetting the watchdog timer, the **Alarm** output will stay low. But if your program hangs up somewhere, the timer will eventually time out and the rising edge on **Alarm** can be used to (re)start the program specified by **POLLRUN**.

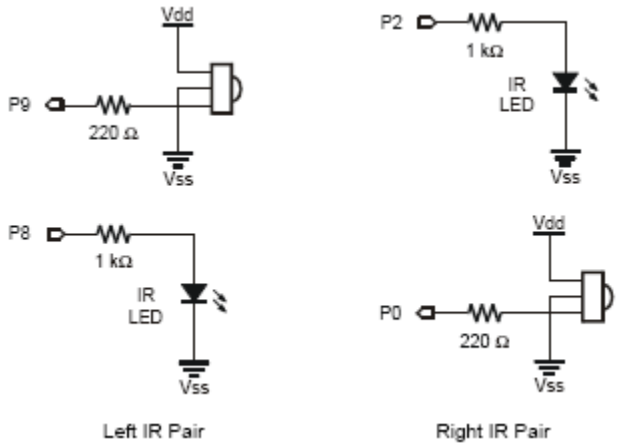
Simple Example Program

Here's a simple-minded BS2 program that moves a BOE-Bot forward for 5 seconds, then in reverse for five seconds, using the Alarm output as a timer:

```
' =============================================================================
'
'   File..... ServoPAL_Simple_Demo.bs2
'   Purpose... Demonstrate ServoPAL capabilites on a BOE-Bot
'   Author.... Parallax, Inc.
'   E-mail.... support@parallax.com
'   Started... 2007.05.01
'   Updated... 2007.10.29
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================

' -----[ Program Description ]---------------------------------------------

' This simple demo of the ServoPAL, when used with a BOE-Bot, will move the
' robot forward for five seconds, then in reverse for five seconds.

' -----[ I/O Definitions ]-------------------------------------------------

nInp  PIN 12              'Define the input pin.
Alarm PIN 13              'Define the alarm pin.

' -----[ Initialization ]--------------------------------------------------

INPUT nInp                'Make sure nInp isn't being driven.
DO : LOOP UNTIL nInp      'Wait for ServoPAL to power up.

LOW nInp                  'Set pin to an output and hold it low
PAUSE 100                 '  for 100mS to reset ServoPAL.
HIGH nInp                 'Raise the pin.
PAUSE 100

' -----[ Program Code ]----------------------------------------------------

PULSOUT nInp, 500        'Program right servo for full forward.
PULSOUT nInp, 1000       'Program left servo for full forward.
PULSOUT nInp, 50         'Program alarm for 5 seconds.

DO : LOOP UNTIL Alarm    'Wait for Alarm.

PULSOUT nInp, 1000       'Program right servo for full reverse.
PULSOUT nInp, 500        'Program left servo fro full reverse.
PULSOUT nInp, 50         'Program alarm for 5 seconds.

DO : LOOP UNTIL Alarm    'Wait for Alarm.

PULSOUT nInp, 2000       'Turn right servo off.
PULSOUT nInp, 2000       'Turn left servo off.
DO:LOOP
```

IR Roaming Program

The next example is an IR roaming program for the BOE-Bot. To use this program, you should connect the IR emitters and sensors as shown in Parallax's *Robotics with the Boe-Bot*, figure 7-4 (copied below).



The program works like this:

1. Start the BOE-Bot moving forward.
2. Keep checking for obstacles until one is found.
3. If obstacle is on left or right, begin turning in opposite direction.
4. If obstacle is in front, back up for one second, then begin turning right for at least ½ second.
5. Continue evasive action until clear of obstacles AND any *timed* motion has completed.
6. Go to step 1.

Here's the program:

```
' =============================================================================
'
'   File..... ServoPAL IR Roaming.bs2
'   Purpose... BOE-Bot IR Roaming Program using ServoPAL
'   Author.... Parallax, Inc.
'   E-mail.... support@parallax.com
'   Started... 2007.10.29
'   Updated...
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================

' -----[ Program Description ]---------------------------------------------

' This program uses the ServoPAL in conjuction with IR emitters and
' detectors to steer the BOE-Bot past obstacles in encounters in its
' path.

' -----[ I/O Definitions ]-------------------------------------------------

IRDetR          PIN     0               'Righthand IR detector.
IREmtR          PIN     2               'Righthand IR emitter.
IREmtL          PIN     8               'Lefthand IR Emitter.
IRDetL          PIN     9               'Lefthand IR detector.

nInp            PIN     12              'ServoPAL input pin.
Alarm           PIN     13              'ServoPAL alarm pin.

' -----[ Constants ]------------------------------------------------------

NONE            CON     0
RIGHT           CON     1
LEFT            CON     2
```

```
BOTH            CON     3

BKWD            CON     0
FWD             CON     3

' -----[ Variables ]----------------------------------------------------

Obstacle        VAR     Nib             'Reading from IR detectors.
Move            VAR     Nib             'Motor command.
Time            VAR     Word

' -----[ Initialization ]------------------------------------------------

INPUT nInp                              'Make sure nInp isn't being driven.
DO : LOOP UNTIL nInp                    'Wait for ServoPAL to power up.

LOW nInp                                'Set pin to an output and hold it low
PAUSE 100                               '  for 100mS to reset ServoPAL.
HIGH nInp                               'Raise the pin.
PAUSE 100

' -----[ Program Code ]--------------------------------------------------

DO
  Move = FWD                          'Start BOE-Bot moving forward,
  GOSUB DoMotor                       '  using the DoMotor routine.

  DO
    GOSUB ReadIR                      '  Read obstacle detectors,
  LOOP UNTIL Obstacle                 '    until an obstacle is found.

  IF (Obstacle = LEFT) THEN           '    Obstacle on the left?

    Move = RIGHT                      '      Yes: Start rotating right.
    GOSUB DoMotor

  ELSEIF (Obstacle = RIGHT) THEN      '    On the right?

    Move = LEFT                       '      Yes: Start rotating left.
    GOSUB DoMotor

  ELSEIF (Obstacle = BOTH) THEN       '    In front?

    Move = BKWD                       '      Yes: Start backing up,
    Time = 10                         '             and set alarm for 1 sec.
    GOSUB DoMotor

    DO UNTIL Alarm : LOOP             '           Wait for alarm to time out.

    Move = RIGHT                      '           Then start rotating right,
    Time = 5                          '             setting alarm for 1/2 sec.
    GOSUB DoMotor

  ENDIF

  DO                                  '         Evasive action is being taken.
    GOSUB ReadIR                      '         Read the obstacle detectors.
  LOOP UNTIL Obstacle = NONE AND Alarm '       Continue (timed) evasive maneuver,
                                      '           until clear AND time is up.
LOOP

' -----[ Subroutines ]--------------------------------------------------

'------[ ReadIR ]-----------------------------------------------------------

' Read both IR sensors. Put inverted values into position in Obstacle.

ReadIR:

  FREQOUT IREmtR, 1, 38500            'Activate the right IR emitter.
  Obstacle.BIT0 = ~IRDetR             'Read the right IR sensor (and invert).
  FREQOUT IREmtL, 1, 38500            'Activate the left IR emitter.
```

```
  Obstacle.BIT1 = ~IRdetL              'Read the left IR sensor (and invert).
  RETURN

'------[ DoMotor ]-----------------------------------------------------------

' Start motors turning to move BOE-Bot in direction indicated. If Time > 0,
' then set alarm and clear Time to zero.

DoMotor:

  IF (Move = BKWD) THEN
    PULSOUT nInp, 900
    PULSOUT nInp, 600
  ELSEIF (Move = LEFT) THEN
    PULSOUT nInp, 600
    PULSOUT nInp, 600
  ELSEIF (Move = RIGHT) THEN
    PULSOUT nInp, 900
    PULSOUT nInp, 900
  ELSE
    PULSOUT nInp, 600
    PULSOUT nInp, 900
  ENDIF
  IF (Time) THEN
    PULSOUT nInp, Time
    Time = 0
  ENDIF
  RETURN
```