PARALLAX INC

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

AVR Firmware: SERVO, Version 1
RC Servo Controller (Preliminary)

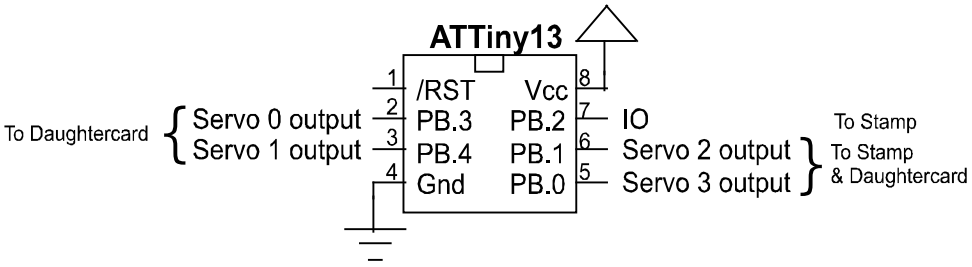
Introduction

This document describes AVR firmware that is used in conjunction with the BS2pe BASIC Stamp motherboard (the MoBoStamp-pe). This firmware can be uploaded to either or both of the motherboard's two AVR coprocessors as file **SERVO1.hex**. Once loaded, the coprocessor is capable of communicating with the BASIC Stamp using PBASIC's **PULSOUT** and **PULSIN** commands. This communication takes place on the AVR's **IO** pin (see illustration below. By utilizing the capabilities of the AVR coprocessor, up to four RC servos can be pulsed continuously without intervention from the BASIC Stamp.

The functions available with this firmware include:

- Programming the pulse widths for any of up to four servos.
- Maintaining output pulses for the programmed servos.
- Keeping unprogrammed I/O pins tri-stated so they can be used by the BASIC Stamp.

The AVR (Atmel ATTiny13) pinout is shown below:



Pin **IO** (BASIC Stamp pin 10 for daughterboard "A"; pin 6, for "B") connects to the Stamp and has a pull-up resistor to Vdd. Communication is bi-directional via a protocol using open-collector pulse signaling. Ports 2 and 3 also connect to the Stamp without external pull-ups, as well as to an attached daughterboard. Ports 0 and 1 connect to an attached daughterboard only. Any or all of ports 0 to 3 can be used to control attached servos.

An appropriate daughterboard for powering and connecting servos is available from Parallax: the PWR-I/O-DB (part #28301). Using the PWR-I/O-DB, servos controlled by this firmware can be plugged into headers A0, A1, A2, and/or A3.

Programming the Servo Pulses

When the AVR comes out of reset, all four servo ports are tri-stated as inputs. Each will remain an input unless or until it's programmed with a valid pulse width. This allows the BASIC Stamp to use the Servo 2 and Servo 3 outputs for other purposes, if there are no servos attached.

The positive servo pulses are programmed by sending negative pulses of the desired width to the AVR via the **IO** pin. Pulses are sent in sequence: first a pulse for servo 0, then a pulse for servo 1, then 2, and finally 3. A sequence of pulses may be terminated early, simply by not sending any more pulses for at least 5 milliseconds. So if you want to program only servos 0 and 1, for example, just send two pulses: one for servo 0 and one for servo 1.

To skip a servo, send a pulse of short duration. Twenty microseconds works fine for this. Servos which are omitted from the end of a pulse sequence, or which are skipped, retain their present state. If a servo output was never programmed, it will remain as an input. If it was programmed since the last reset, it will continue to emit the last programmed pulse width.

Once an output has been programmed, it will remain an output until the AVR is reset. However, it is possible to stop the output of pulses for any servo by sending it a pulse width of more than 3.5 milliseconds – 4 milliseconds for maximum reliability. When this is done, a constant "low" is output until the pin is reprogrammed by a legitimate pulse.

Before sending pulses via the **IO** pin, the BASIC Stamp program must wait until the AVR is ready to receive them. The AVR signals its readiness via a short negative-going pulse on **IO**. The PBASIC program then has 5 milliseconds to begin sending pulses before having to wait for the next opportunity. This "ready" pulse repeats once every 20 milliseconds. For this reason, pin **IO** should be programmed as an input until it's time to send pulses to the AVR. The PBASIC **PULSIN** command can be used to catch the next opportunity. In the following example, servos 0 and 2 are being programmed, skipping servo 1:

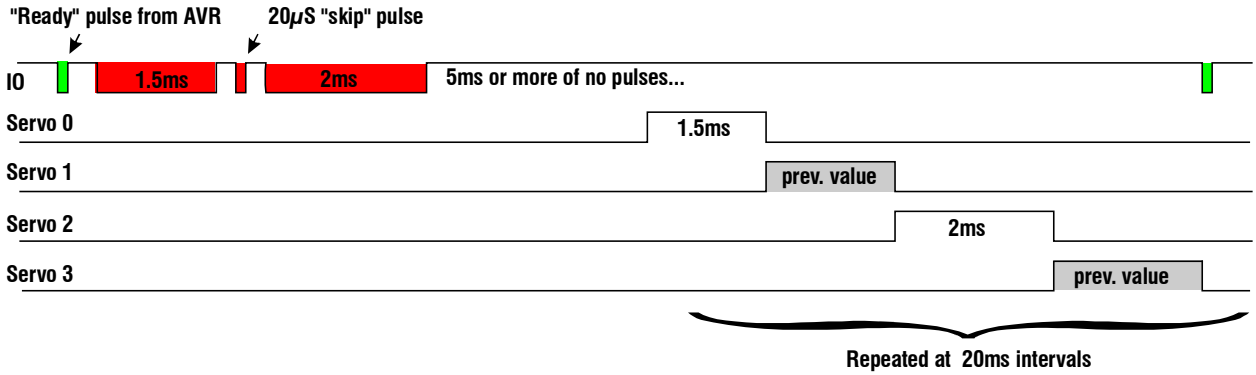
```
IO   PIN 10              'Pin 10 for "A"; pin 6 for "B".
Nil  VAR Bit             'Dummy variable, required for PULSIN

IO = 1                   'Set default output high, but keep as input.

PULSIN IO, 0, Nil        'Wait for "ready" pulse from AVR.
PULSOUT IO, 750          'Set servo 0 to center position (1.5 ms).
PULSOUT IO, 10           'Skip servo 1 (20us).
PULSOUT IO, 1000         'Set servo 1 to far extreme (2 ms).
INPUT IO                 'Return IO to an input state.
```

After execution of this code fragment, servo 0 will receive a continuous stream of 1.5ms pulses, servo 2 a stream of 2ms pulses, and servos 1 and 3 will remain at whatever they'd been before.

The following waveforms illustrate what happens with the above code:



A complete program, which includes a "SetServos" subroutine, follows. You can copy the SetServos routine into your own programs, if it meets your needs.

```
' {$STAMP BS2pe}
' {$PBASIC 2.5}

IO      PIN 10                  'Socket "A". Set to PIN 6 for socket "B".

Nil     VAR Bit                 'Dummy variable for PULSIN.
Servo   VAR Byte(4)             'Array of values for servos.
i       VAR Nib                 'Index variable.

io = 1                          'Set default IO output level to high.
                                'Pin is still an input, though.

DO                              'Repeat forever.
  FOR Servo(0) = 1 TO 254       'Cover full range of motion.
    Servo(1) = Servo(0) + 64  // 255 'Servos will be "90 degrees
    Servo(2) = Servo(0) + 128 // 255 '  out of phase"
    Servo(3) = Servo(0) + 192 // 255 '  with each other.
    GOSUB SetServos             'Set the new servo positions.
    PAUSE 100                   'Wait 100ms.
  NEXT                          'Next positions.
LOOP                            'And so on...

'--------SetServos-----------------------------------
' Subroutine to set each servo pulse stream to the
' width defined by the byte array Servo. Valid widths
' can run from 1 to 254, corresponding to a range of
' 0.484ms to 2.508ms, centering at 128 for 1.5ms.
' Setting a servo value to zero, skips that servo
' output. Setting a servo value to 255 kills that
' servo's output pulse train.
'-----------------------------------------------------

SetServos:
  PULSIN IO, 0, nil             'Wait for AVR's "ready" signal.
  FOR i = 0 TO 3                'For all four servo outputs.
    IF (Servo(i) = 255) THEN    'Kill the output?
      PULSOUT IO, 2000          '  Yes: Send a 4ms pulse.
    ELSEIF (Servo(i)) THEN      '  No:  Normal pulse?
      PULSOUT IO, Servo(i) << 2 + 238  'Yes: Send the proper length pulse.
    ELSE
      PULSOUT IO, 10            '         No:  It's zero, so skip this one.
    ENDIF
  NEXT
  INPUT IO                      'IO is an output from the PULSOUTs.
                                'Make it an input again.
  RETURN
```