

AVR Firmware: ENC, Version 1

Quadrature Encoder Tracking (Preliminary)

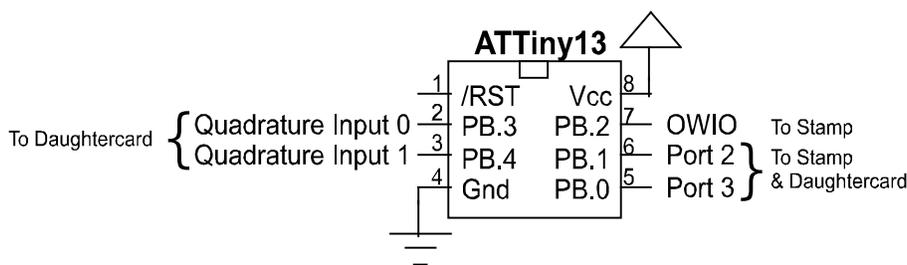
Introduction

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 **ENC1.hex**. Once loaded, the coprocessor is capable of communicating with the BASIC Stamp using PBASIC's **OWOUT** and **OWIN** commands. This communication takes place on the AVR's OWIO pin (see illustration below) to read data from, and write data to, the AVR. By utilizing the capabilities of the AVR coprocessor, the encoder's position can be tracked continuously without intervention from the BASIC Stamp.

The functions available with this firmware include:

- Keeping track of up/down count (16-bits) from quadrature encoder inputs (up to 37.5KHz) on ports 0 and 1.
- Starting and stopping the count.
- Zeroing the count.
- Setting the count to an arbitrary value.

The AVR (Atmel ATTiny13) pinout is shown below:



Pin OWIO connects to the Stamp and has a pull-up resistor to Vdd. Communication is bi-directional via a protocol using open-collector signaling. Ports 2 and 3 also connect to the Stamp without external pull-ups, as well as to an attached daughtercard. Ports 0 and 1 connect to an attached daughtercard only and are the quadrature inputs from the encoder. Ports 2 and 3 are not used by ENC1 and are left floating so the BASIC Stamp can control them directly.

Important: Make sure that the signal levels from the encoder are no higher than the MoBoStamp's Vdd setting. You cannot use 5V encoder signals when Vdd is set to 3.3V!

Command Protocol

Communication with the AVR is via one-character, case-insensitive commands sent using the Stamp's **OWOUT** statement, possibly followed by additional data bytes or by reads using **OWIN**. An example command might be:

```
OWOUT Owio, 0, ["G"]
```

Where **Owio** is the pin (10 for coprocessor "A"; 6, for coprocessor "B") used to communicate with the AVR. The above command ("**G**") will cause ENC1 to start (or resume) counting.

Some commands are used to get data from the AVR. An example would be:

```
Count VAR Word
OWOUT Owio, 0, ["R"]
OWIN Owio, 2, [Count.LOWBYTE, Count.HIGHBYTE]
```

This reads the current value of the encoder "position" into variable **Count**.

Any reset (low pulse lasting longer than 160 microseconds) sent via the OWIO pin will reset the AVR's protocol state machine, interrupting any transaction in progress. It will not affect the value of the count or whether the counting is enabled or stopped. A reset can be incorporated into an **OWOUT** or **OWIN** statement by choosing the second argument appropriately. In the above example, the second argument for the **OWIN** statement is **2**, which means "reset after the input". Doing so will reset the communications protocol, effectively canceling further reads until another read command is issued.

In the following example, a reset is issued *before* the command to begin counting:

```
OWOUT Owio, 1, ["G"]
```

A reset is usually a good idea in the first transaction of a Stamp program. It should also be considered when communication with the AVR is infrequent or done in electrically noisy environments. A too-liberal use of resets, however, can not only slow a program down, but it can also mask program errors associated with AVR communications.

Finally, the AVR comes out of a hardware reset more slowly than the Stamp does. So don't start talking to it right away in your Stamp program. To make sure the AVR is ready for communicating, put a 5ms PAUSE at the beginning of your Stamp program:

```
PAUSE 5
```

This will prevent out-of-the gate misfires.

Firmware Identification

To identify the firmware currently extant in the AVR (assuming it uses the same protocol), send the version command ("**V**"). The AVR will respond with three bytes of data. The first two are characters representing the name of the firmware. The third is a version number. This information can be used by a BASIC Stamp program to make sure the correct AVR firmware is loaded. The following program snippet, where **I**, **J** and **K** are byte variables, prints out this information:

```
OWOUT Owio, 0, ["V"]
OWIN Owio, 0, [I, J, K]
```

```
DEBUG " Device: ", I, J, ", Version: ", DEC K
```

For this firmware, the following line is printed:

```
Device: EN, Version: 1
```

Zero the Counter

To zero the internal 16-bit counter, send the zero ("Z") command:

```
OWOUT Owio, 0, ["Z"]
```

This will zero the counter immediately, regardless of whether counting is enabled or not.

Set the Counter

To set the counter to any arbitrary value, the write ("W") command can be used. Following the command are two arguments: the least significant byte of the new counter value, followed by the most significant byte. For example, the following will set the counter to **\$1234**:

```
OWOUT Owio, 0, ["W", $34, $12]
```

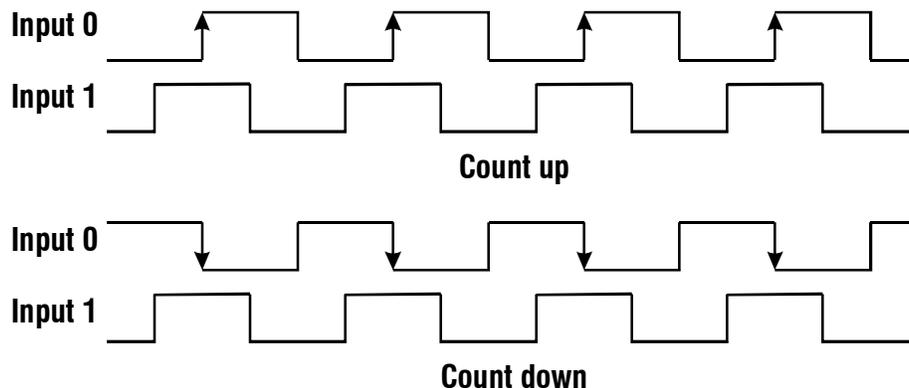
The write will take place immediately after the last byte is received, regardless of whether counting is enabled or not.

Start Counting

When the AVR comes out of hardware reset, the counter is disabled. To enable counting, you must issue the go ("G") command:

```
OWOUT Owio, 0, ["G"]
```

This will cause the counter to begin incrementing or decrementing, based on the quadrature encoder inputs on ports 0 and 1. The count will increase when quadrature input 1 is high, and quadrature input 0 goes from low to high. It will decrease when input 1 is high, and input 0 goes from high to low:



Counting up from \$FFFF (65535) will roll over to 0; counting down from 0 will roll under to \$FFFF. ENC1 has been tested with clean encoder inputs at 37.5KHz without error.

Stop Counting

To stop the counter, issue the stop ("S") command:

```
OWOUT Owio, 0, ["S"]
```

This will halt the counter immediately.

Read Current Count

To enter count read mode, issue the read ("R") command. This will make it possible to get the count from the AVR using subsequent **OWIN** commands. The AVR will stay in read mode until a command protocol reset is issued. Each subsequent read must consist of two bytes: the low byte of the count, followed by the high byte. Here's an example:

```
Cnt VAR Word
I   VAR Word

OWOUT Owio, 0, ["GR"]
FOR I = 1 TO 1000
  OWIN Owio, 0, [Cnt.LOWBYTE, Cnt.HIGHBYTE]
  DEBUG DEC Cnt, CR
NEXT
OWOUT Owio, 1, ["S"]
```

The first **OWOUT** starts the counter ("G"), then enters read mode ("R"). Subsequent to that, 1000 byte pairs are read continuously, without having to reissue the read command each time. To get out of read mode, the last **OWOUT** issues a reset before sending the stop ("S") command.

Read Differential Count

The read differential ("D") command works just like the read current count command, except that it returns the difference between the current count and the count last read. This is handy when all you want to know, say, is how much a rotating shaft has moved since the last time you checked – not what its current position is. Here's a repeat of the above example, but using the read differential command instead:

```
Cnt VAR Word
I   VAR Word

OWOUT Owio, 0, ["GD"]
FOR I = 1 TO 1000
  OWIN Owio, 0, [Cnt.LOWBYTE, Cnt.HIGHBYTE]
  DEBUG DEC Cnt, CR
NEXT
OWOUT Owio, 1, ["S"]
```