

Chapter 6: Asynchronous Serial Communication

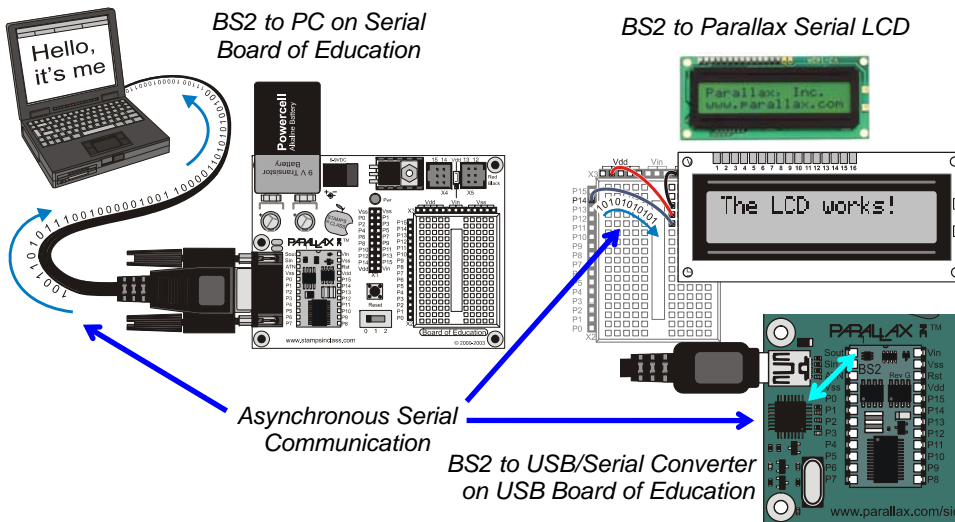
The previous chapter introduced synchronous serial communication, which relies on a separate clock signal to synchronize the exchange of a series of values. In contrast, asynchronous serial communication is the exchange of a series of values without the synchronizing clock signal.

ASYNCHRONOUS SERIAL DEVICES

Figure 6-1 shows some examples of devices that use asynchronous serial communication to exchange information. When a BASIC Stamp executes a DEBUG command, it uses asynchronous serial communication to send information to the PC, either through a serial cable, or to a chip that converts the serial message to the USB protocol. Another example, the BASIC Stamp makes the Parallax Serial Liquid Crystal Display (LCD) display text by sending it asynchronous serial messages. Other useful peripherals that utilize asynchronous serial communication include geographic positioning systems (GPS), radio frequency identification (RFID), and radio frequency communication modules that make it possible for microcontrollers to communicate “wirelessly”. This chapter introduces the signaling these devices use to communicate and examines and decodes certain asynchronous serial messages with the PropScope.

6

Figure 6-1: Asynchronous Serial Device Examples



ACTIVITY #1: ASCII CODES

ASCII stands for American Standard Code for Information Exchange, and it uses numeric codes to represent US alphabet characters. It also includes some special codes called control characters for keys on your keyboard like Esc and Backspace. When the BASIC Stamp sends messages that get displayed as text by a PC or serial LCD, it uses ASCII codes to send the characters that get displayed.

Printable ASCII Chart.bs2 displays characters and their corresponding ASCII codes for values of 32 through 127 in the Debug Terminal.

- ✓ Enter and run Printable ASCII Chart.bs2

```
' Printable ASCII Chart.bs2
' Display 32 through 127 along with the ASCII characters those
' values represent.

' {$STAMP BS2}                ' Target module = BASIC Stamp 2
' {$PBASIC 2.5}              ' Language = PBASIC 2.5

char          VAR          Word          ' Will store ASCII codes

PAUSE 1000                    ' 1 second delay before messages

DEBUG CLS, "    PRINTABLE ASCII CHARACTERS", ' Display chart heading
      CR, "          from 32 to 127"

FOR char = 32 TO 127          ' Character=ASCII value loop
  DEBUG CR$RXY, char-32/24*10, char-32//24+3 ' Position cursor row, column
  DEBUG char, " = ", DEC3 char      ' Display Character=ASCII value
NEXT
```

The Debug Terminal display should resemble Figure 6-2.

- ✓ If the display gets scrambled because there's not enough room in the Debug Terminal, make the window larger and restart the program by pressing and releasing the Reset button on your board.

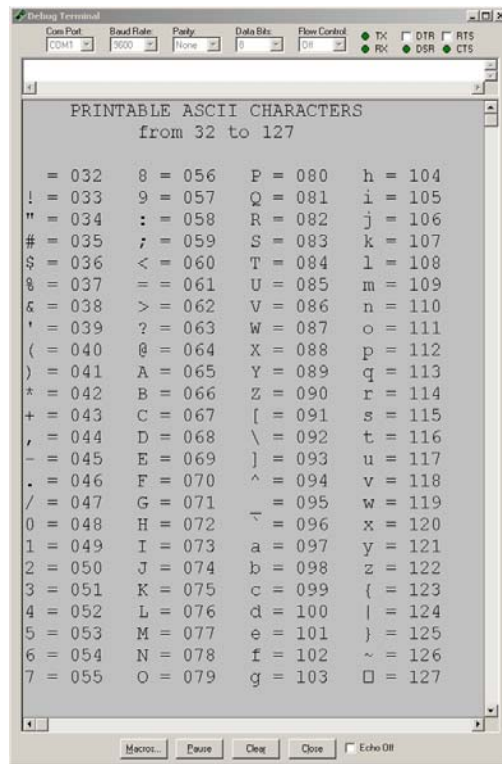


Figure 6-2
Characters and Decimal
Value ASCII Codes
Displayed in Debug
Terminal

6

Figure 6-3 shows ASCII values the Debug Terminal uses as control characters for operations like Clear Screen, Home, Backspace, Tab, Carriage return and others.

- ✓ In the BASIC Stamp Editor, click Edit and select Preferences. Then, click the Debug Function tab.

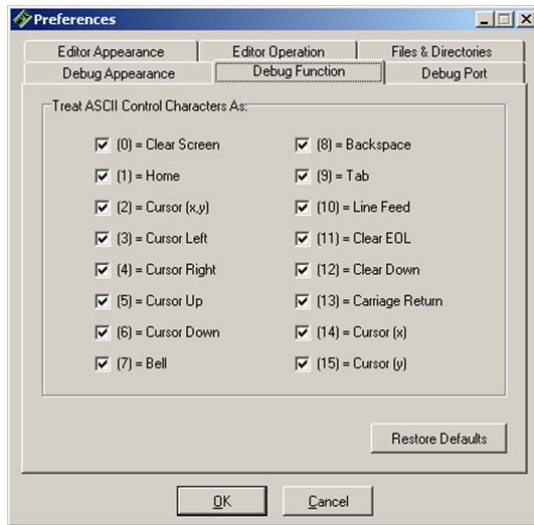


Figure 6-3
Debug Terminal Control
Character Values and
Functions

All of the control characters listed in Figure 6-3 have symbol names in the PBASIC language. For example, instead of `DEBUG 0` for Clear Screen, you can use `DEBUG CLS`. In place of `DEBUG 13` for Carriage Return, you can use `DEBUG CR`. The `DEBUG` command documentation in the BASIC Stamp Editor Help's PBASIC Language Reference has a complete list of PBASIC control character symbol names, descriptions of their functions, and the values they represent.

- ✓ Close the preferences window and open the BASIC Stamp Editor's Help. (Click Help and select BASIC Stamp Editor Help...)
- ✓ In the PBASIC Language Reference, follow the link to the `DEBUG` command documentation.
- ✓ Scroll down to the last table. It lists the names and ASCII values of the functions in Figure 6-3 along with PBASIC Symbol names you can use as `DEBUG` command arguments, like `CLS`, `CR`, `HOME`, etc.

Your Turn: Display “A” to “Z”, “a” to “z”, and 128 to 255

Printable ASCII Chart.bs2 uses the char variable in a FOR...NEXT loop to count from 32 TO 127. Each time through the loop, the DEBUG command transmits the char variable's value to the PC using asynchronous serial communication. The Debug Terminal then displays the ASCII value's corresponding character. The PBASIC language treats characters in quotes as their ASCII value equivalents, so you could actually create a FOR...NEXT loop that counts from "A" to "Z", "a" to "z", or even "!" to "":

- ✓ In Printable ASCII Chart.bs2, try changing 32 TO 127 to "A" TO "Z".
- ✓ Load the modified program into the BASIC Stamp, and observe the results.
- ✓ Repeat for "a" TO "z" and "!" TO "":

The Debug Terminal has more characters in the 128 to 255 range that you can check too. ~~These can be useful for displays in certain languages as well as symbols like degrees °, π, ¼ and other symbols.~~ For example, the a DEBUG command to display 180° would be DEBUG "180", 176. The code 176 makes the Debug Terminal display the ° symbol.

- ✓ Try this modified FOR...NEXT loop in Printable ASCII Chart.bs2.

```
FOR char = 128 TO 255
  DEBUG CRSRXY, char-128/24*10, char-128//24+3
  DEBUG char, " = ", DEC3 char
NEXT
```

- ✓ Make a note of any character codes that might be useful for displays in future BASIC Stamp projects.

ACTIVITY #2: FIRST LOOK AT ASYNCHRONOUS SERIAL BYTES

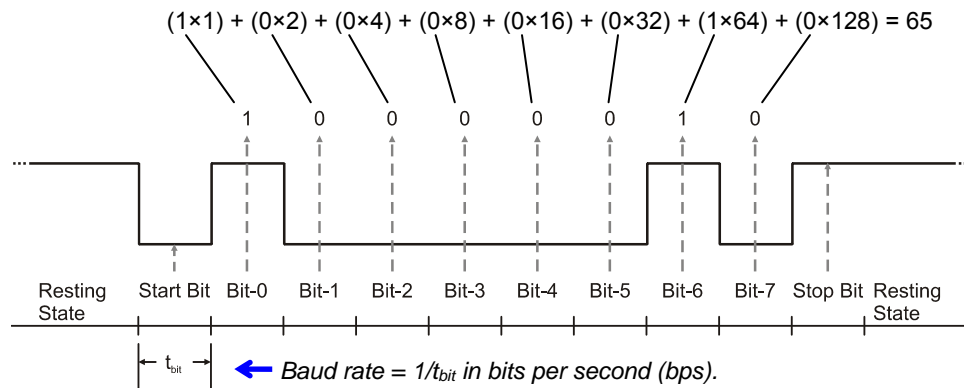
Figure 6-4 shows an example timing diagram for the number 65, which is the letter "A", transmitted with a widely used format of asynchronous serial signaling. The device that transmits this byte starts with a transition from high (Resting State) to low (Start Bit). Both of the devices use that as the starting point. The sending device uses it for updating the bit values it transmits at regular time intervals (t_{bit}), and the receiving device knows to check for new binary values at those intervals. The baud rate determines the time interval, which is $t_{bit} = 1/\text{baud rate}$. For example, if the baud rate is 9600 bits per second (bps), it means that each bit period has to be $1/9600^{\text{th}}$ of a second. In other words, the bit time is $t_{bit} = 1 \div (9600 \text{ bits/second}) \approx 104.17 \mu\text{s/bit}$. So the transmitting device updates its

6

Comment [AL62]: Search replace +/- with this character throughout document.

binary values every 104.17 μ s, and the receiving device checks in the middle of each of those time periods for the next binary value in the byte.

Figure 6-4: Number 65 Transmitted with 8-Bit, True, No Parity Asynchronous Serial Signaling



A device that transmits or receives the asynchronous serial signal in Figure 6-4 is using a format called true signaling, 8 bits, no parity, and one stop bit. The shorthand for 8 bits, no parity, one stop bit, 8N1, and its usually preceded by a baud rate, like this: 9600 bps, 8N1. With True signaling, which is also called non-inverted, a high signal sends a binary 1, and a low signal sends a binary 0. 8 bits means that the signal contains 8 binary values (bits). “No parity” indicates that this signal does not contain a parity bit. Serial signals can also be configured to contain a parity bit, that the transmitter and receiver use to help detect communication errors. Parity bits will be examined in the Projects at the end of the chapter. One stop bit means that that the transmitter has to wait at least one bit period (t_{bit}) before sending another message. With 1 start bit, 8 data bits, and 1 stop bit, the total amount of time it takes to send/receive a single byte is 10 bit periods.

Examining Figure 6-4 from left to right, the resting state of the signal is high. That signal could stay high for an indefinite amount of time if the device transmitting messages doesn’t have anything to send. When it does have something to send, it sends a low Start Bit signal for one bit period. Again, both transmitter and receiver use the negative edge of this signal for timing. The transmitter has to update its output between every bit period, and the receiver has to check for a value in the middle of each bit period. After the start bit, the transmitter sends the least-significant bit (Bit-0 or LSB), followed by Bit-1 during the next bit period, Bit-2 in the bit period after that, and so on, up through Bit-7.

during the 8th bit period after the start bit. The receiver knows to treat bit 0 as the number of 1s, in the value, Bit 1 as the number of 2s, Bit 2 as the number of 4s, and so on up through bit 7, which is the number of 128s in the value. In Figure 6-4, bit 0, which is the number of 1s is high, and so is bit 6, which is the number of 64s. A rest of the bits are low, so the value this signal transmits is $(1 \times 1) + (1 \times 64) = 65$.

In this activity, you will program the BASIC Stamp to use 9600 bps, 8N1 true asynchronous serial signaling to transmit the values in the previous activity's ASCII chart using an I/O pin. A new character will be transmitted once every second, and you will use the PropScope to monitor the sequence of ASCII values.

Asynchronous Serial Test Parts

(1) Resistor – 220 Ω (red-red-brown)
(misc) Jumper wires

Asynchronous Serial Test Circuit

Figure 6-5 shows the test circuit and Figure 6-6 shows a wiring diagram example.

- ✓ Build the circuit in Figure 6-5 using Figure 6-6 as a guide.

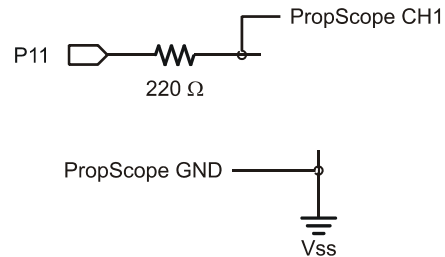


Figure 6-5
Test Circuit for Probing Asynchronous Serial Messages Transmitted by the BASIC Stamp's I/O pin P11.

6

Comment [AL63]: i-box: Resistors are used in this chapter because a function generator signal will be applied to an I/O pin later. They are not needed, except to ensure that there is no way to make a mistake and inadvertently apply function generator voltage to an I/O pin that is set to output.

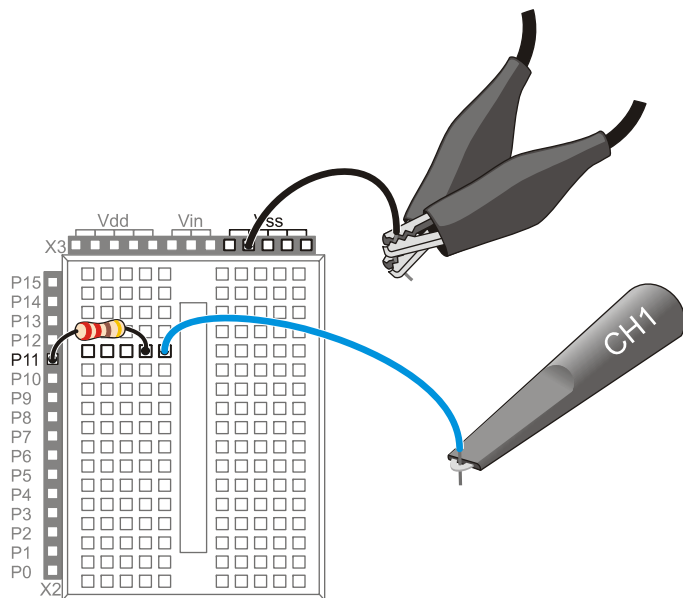


Figure 6-6
Wiring Diagram Example
for Figure 6-5

Asynchronous Serial Test Code

Printable ASCII Chart to IO.bs2 sends a character from the ASCII chart to the DEBUG Terminal once every second. At about the same time, it sends a copy of that character to I/O pin P11 using 9600 8N1 true serial signaling.

- ✓ Open the BASIC Stamp Editor's Help, and look up the SEROUT command in the PBASIC Language Reference.
- ✓ Read the Syntax and function sections.
- ✓ Find the Common Baud Rates and Corresponding Baud mode Values table for the BASIC Stamp 2, and verify that 84 is the Baud mode in SEROUT 11, 84, [char] that will make it send its characters using true signaling at 9600, 8N1.
- ✓ Enter and run Printable ASCII Chart to IO.bs2.

```
' Printable ASCII Chart to IO.bs2
' Display another value in the ASCII character chart once every second.
' Transmit a copy of that value to P11 using 9600 bps 8N1 with true signaling.

' {$STAMP BS2}                ' Target module = BASIC Stamp 2
' {$PBASIC 2.5}              ' Language = PBASIC 2.5

char          VAR          Word          ' For counting and storing ASCII
```



```

PAUSE 1000                                ' 1 second delay before messages
DO                                          ' Main loop
  DEBUG CLS,
    " PRINTABLE ASCII CHARACTERS",      ' Table heading
    CR, " from 32 to 127"
FOR char = 32 TO 127                       ' Count from 32 to 127
  DEBUG CRSRXY, char-32/24*10, char-32//24+3 ' Position cursor
  SEROUT 11, 84, [char]                  ' Send 9600 bps, 8N1 true byte
  DEBUG char, " = ", DEC3 char           ' Display byte in Debug Terminal
  PAUSE 1000                             ' 1 second delay
NEXT                                       ' Repeat FOR...NEXT loop
LOOP                                      ' Repeat main loop

```

6

Asynchronous Serial Test Measurements

Figure 6-7 shows an example of the "A" character, which will display for one second, about 34 seconds into the program. (That's 33 seconds worth of characters and the 1 second PAUSE command at the very beginning.) Keep in mind that the signaling shown by the PropScope will change once every second as the BASIC Stamp transmits a new ASCII value. Also, notice that the Trigger in Figure 6-7 has been set to Fall since the message begins with a negative transition from high resting state to low start bit.

- ✓ Click, hold and drag the CH1 trace downward so that 0 V is only slightly above the time scale. (In Figure 6-7, the 0 V ground line is 1 voltage division above the time scale.)
- ✓ Configure the PropScope's Horizontal, Vertical and Trigger controls according to Figure 6-7.
- ✓ Set the Trigger Voltage control to about 2.5 V, and the Trigger Time control to the second time division.
- ✓ Watch the counting pattern in the Oscilloscope, and use the Debug Terminal as a reference for seeing how the ASCII codes correspond to the signals in the Oscilloscope.
- ✓ Verify that the pattern displayed by the PropScope changes once every second.

Figure 6-7: ASCII 65, one of the values in the 32...127 sequence that will be displayed

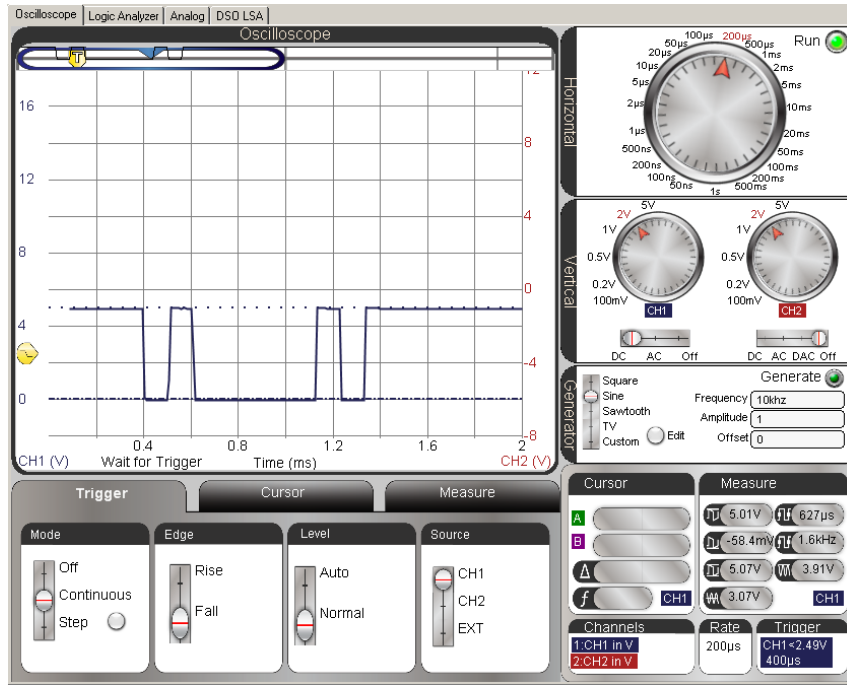


Figure 6-8 shows samples of the binary patterns for the characters "A" through "E", which correspond to ASCII codes 65 through 69. This is a portion of the sequence of asynchronous serial bytes the PropScope should display while Printable ASCII Chart to IO.bs2 is running.

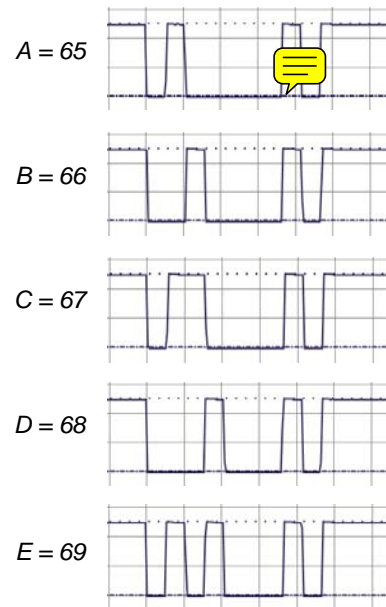


Figure 6-8
Byte Values 65 to 96

These are the ASCII codes for the characters "A" through "E" transmitted at 9600 bps with 8N1 true signaling.

6

Your Turn: DEBUG vs. SEROUT

DEBUG is a special case of the SEROUT command. It's SEROUT 16, 84, [arguments...]. For example, in Printable ASCII Character Chart to IO.bs2, you can replace DEBUG char, " = ", DEC3 char with SEROUT 16, 84, [char, " = ", DEC3 char], and the Debug Terminal will behave exactly the same.

✓ Try it.

ACTIVITY #3: A CLOSER LOOK AT A SERIAL BYTE

In this activity, you will program the BASIC Stamp to send the letter "A" (ASCII 65) using 9600 bps, 8N1, true asynchronous serial signaling and decode its value with the PropScope.

Letter A Test Code

Letter A to P11.bs2 transmits the "A" character via I/O pin P11 once every second, and it also sends it to the Debug Terminal to verify that characters are still getting sent once

every second. The Debug Terminal verification can be useful for situations when you're not sure if the Oscilloscope display should be updating or not.

- ✓ Enter and run Letter A to P11.bs2.

```
' Letter A to P11.bs2
' Transmit "A" to P11 and Debug Terminal once every second.

' {$STAMP BS2}                ' Target module = BASIC Stamp 2
' {$PBASIC 2.5}              ' Language = PBASIC 2.5

PAUSE 1000                    ' 1 second delay before messages

DO                            ' Main loop

  SEROUT 11, 84, ["A"]        ' "A" to P11
  DEBUG "A", " ", DEC3 "A", " ", ' "A" to Debug Terminal
  BIN8 "A", CR
  PAUSE 1000                  ' 1 second delay

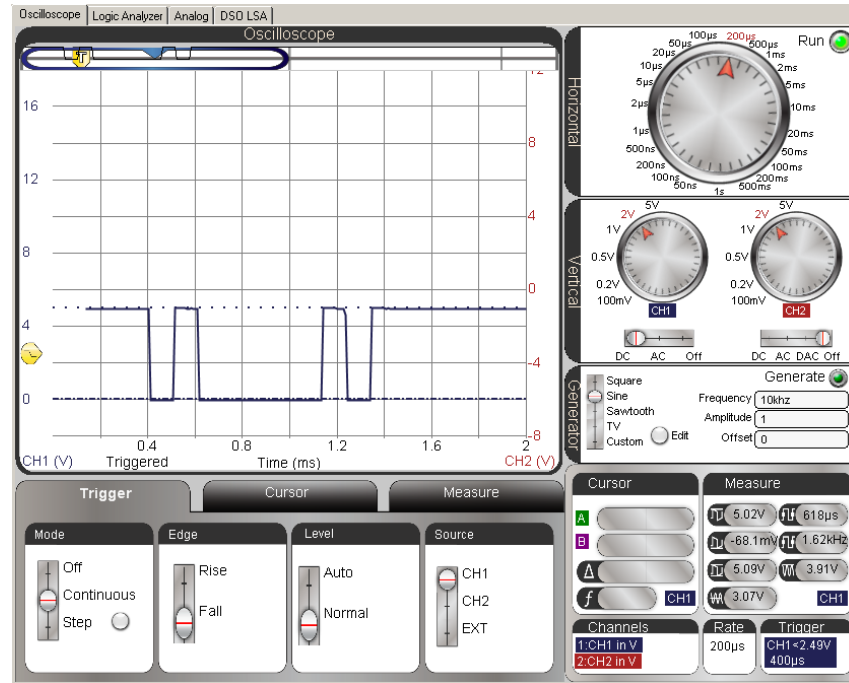
LOOP                          ' Repeat main loop
```

Letter A Test Measurements

Figure 6-9 shows the letter "A" again. It's important to verify that your display shows all the transitions and states in the Oscilloscope screen before continuing.


- ✓ Verify your PropScope's Horizontal, Vertical, and Trigger settings against Figure 6-9.
- ✓ Verify that your Trigger Voltage Control is at about 2.5 V and that the Trigger Time Control is at the 2nd time division (0.4 ms)
- ✓ Make sure you've got a good view of this on your PropScope.

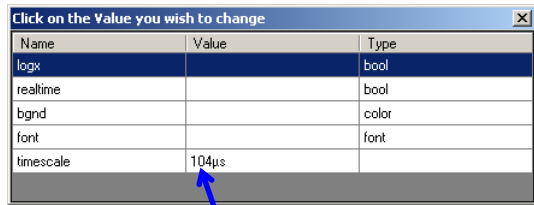
Figure 6-9: The Letter "A" Again



6

The time per division options on the Horizontal dial are not the only options, and this is a case where a custom time per division could come in really handy. Remember from Activity #2 that if the baud rate is 9600 bits per second (bps) Also remember from Activity #1 that the bit time is $t_{bit} = 1 \div (9600 \text{ bits/second}) \approx 104.17 \mu\text{s/bit}$. Wouldn't it be nice if the Oscilloscope had a $104 \mu\text{s}$ units per division setting? Try this:

- ✓ Right click the Oscilloscope screen. The “Click on the Value you wish to change” window in Figure 6-10 should appear.
- ✓ Shade the value 200 in the (timescale, Value) cell and change it to  This will change the time scale value from the Horizontal dial's $200 \mu\text{s/division}$ setting to $104 \mu\text{s/division}$.
- ✓ Press the Enter key to enter your value, and then close the window.



Name	Value	Type
logx		bool
realtime		bool
bgnd		color
font		font
timescale	104us	

Shade the number 200 and change it to 104

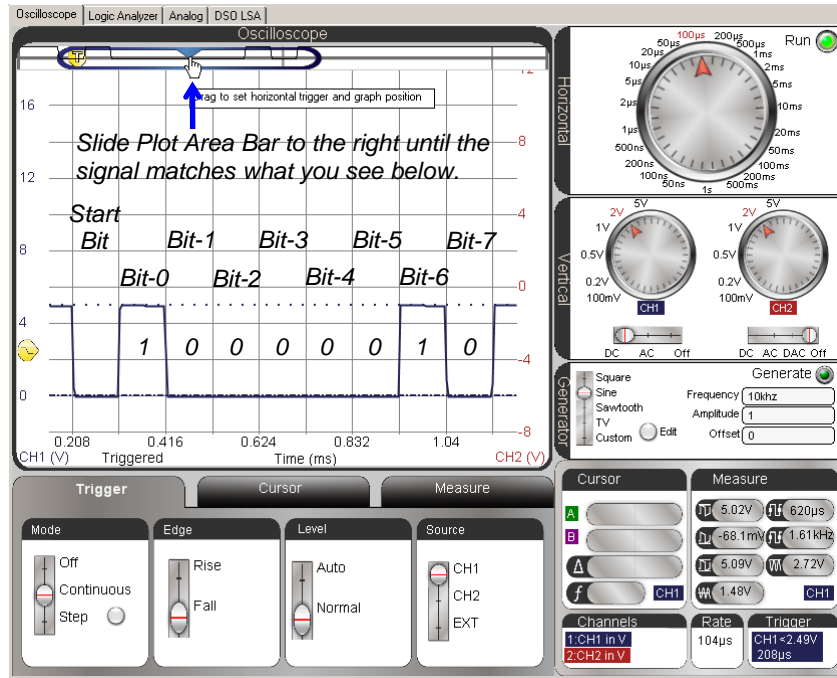
Figure 6-10
Configuring Custom Time/Division

Your Oscilloscope will need a few more adjustments before it resembles Figure 6-11.

- ✓ Adjust the Plot Area Bar slightly to the right so that you can view all the bits of the asynchronous serial 65 byte as shown in Figure 6-11. The falling edge of the start bit should align with the first visible time division line.
- ✓ Compare your results to the timing diagram in Figure 6-4 on page 220.

The important feature here is that each bit in the asynchronous serial byte now occupies a single time division. This makes it much easier to translate an asynchronous serial byte displayed on the oscilloscope screen into the value that's being transmitted. Remember that Bit-0 is the number of 1s in the number, Bit-1 is the number of 2s, Bit-3 is the number of 4s, and so on up through Bit-7, which is the number of 128ths.

Figure 6-11: 9600 bps Byte Value = 65 Viewed with Timescale set to 104 µs/Division



6

Bits in a Byte: The values in Figure 6-11 get stored in a byte so that it looks like this: %01000001. The % sign is a PBASIC formatter that tells the BASIC Stamp Editor that it's a binary number. In this binary number, bit-0 gets stored in the rightmost position, bit-1 in the next position to the left, and so on up through bit-7, which is the leftmost position. It's the opposite of the order from the binary digits get transmitted in an asynchronous serial byte.

Powers of 2 in a Byte: The value stored by Bit-0 determines the number of 1s in the variable, and $2^0 = 1$. The value stored by Bit-1 determines the number of 2s in the variable, and $2^1 = 2$. The value in Bit-3 determines the number of 4s in the variable, and $2^2 = 4$. More generally: Each bit determines whether a binary number has $(1 \text{ or } 0) \times 2^{\text{bit position}}$.

Your Turn: Pick a Byte Value

- ✓ Repeat the measurements in this activity with a character or byte value of your choosing.