

## ESE381 Embedded Microprocessor Systems Design II

Spring 2024 - K. Short      revised February 17, 2024 11:20 am

### **Laboratory 04: Software Implementation of a UART Transmitter and Receiver and Use of a Saleae Logic Analyzer and Tera Term Terminal Emulator**

To be performed the week starting February 18th.

#### **Prerequisite Reading**

- 1.
2. Maxim Integrated, *Serial Digital Data Networks* (on Brightspace).
3. Maxim Integrated, *Determining Clock Accuracy Requirements for UART Communications* (on Brightspace).
4. Dallas Semiconductor, *Application Note 83 Fundamentals of RS-232 Serial Communications* (on Brightspace).
5. *Saleae User's Guide 11-12-19*, (on Brightspace).
6. Download Saleae Software using this link: <https://www.saleae.com/downloads/>

#### **Overview**

Microcontrollers have universal synchronous/asynchronous receiver/transmitter (USART) modules that provide, among other things, all the functionality of a universal asynchronous receiver and transmitter (UART) implemented in hardware. These hardware modules provide an option for a hardware/software trade-off in implementing asynchronous serial communications. Their major advantage is that they can perform the operations of serializing and deserializing data using hardware with the microcontroller CPU free to concurrently perform other tasks. Using hardware is typically the preferred approach to implementing asynchronous serial communications and you will use that approach in Laboratory 05.

However, nothing clarifies the basic concepts of asynchronous serial transfer like writing your own C functions to implement a serial transmitter and receiver. In doing so, you make use of C's bit manipulation instructions and library delay functions.

With any serial transfer protocol, debugging a design is greatly simplified if you have the appropriate test equipment. A key piece of equipment for debugging serial communications is a protocol aware logic analyzer. In this laboratory you will learn to use Saleae's 16-channel logic analyzer's protocol aware capabilities. This protocol aware analyzer can analyze a large number of different serial protocols, including asynchronous serial. You will be using this tool extensively this semester as you study some of the most used serial protocols in embedded system design.

When using the asynchronous serial protocol, another key tool is a terminal emulator software application that emulates an ASCII terminal on your PC. The emulated ASCII terminal allows you to send and receive ASCII information in the asynchronous protocol and see this information

displayed on the PC's monitor. There are many different terminal emulators available. For this laboratory, you will be using the Tera Term terminal emulator.

## Design Tasks Summary

*Design Task 1: Software UART Transmitter Function*

*Design Task 2: Software UART Receiver Function*

*Design Task 3: Software UART Receiver Interrupt Service Routine*

*Design Task 4: Interrupt Echo Program*

*Design Task 5: Message Relay Program*

*Design Task 6: Distance Measuring Program*

## Design Tasks

*Design Task 1: Software UART Transmitter Function*

You must write a function named `UART_sw_write`.

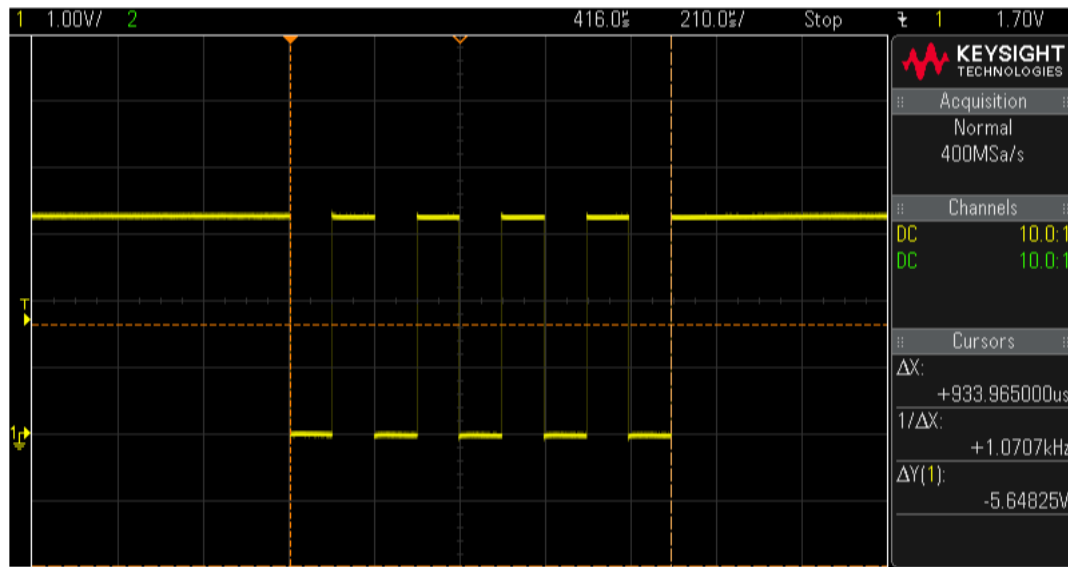
```
void UART_sw_write(char c);
```

This function has a `char` parameter and has the return type `void`. The `char` passed to the function will be an ASCII character that is to be transmitted according to the asynchronous serial protocol. The frame format for the asynchronous transfer is to be fixed at 8N1. The baud rate macro `BAUD_RATE` must be defined using a `#define` directive at the beginning of your program, so that the baud rate can easily be changed.

Write a verification program named `asynch_sw_send` that allows you to send ASCII characters at any of the following three baud rates: 4800, 9600, and 19200 baud using the `UART_sw_write` function. This program must use PB0 as the TX pin for your software transmitter. In the laboratory, this program will be used to verify your `UART_sw_write` function. The characters transmitted will be observed using the oscilloscope, Saleae logic analyzer, and Tera Term. You must use a loop in your program to continuously send the same character with a 1 ms delay between the characters sent.

Your program's duration of a single bit (bit time) needs to be as accurate as reasonably possible for each of the baud rates.

Since we are only going to send a single character at a time, we want to start with a character that provides us the most information. The first thing that we want to know is whether the bit time is



Determine what data value must be transmitted at 9600 baud and 8N1 to create the above trace using the asynchronous serial protocol. Note that, in a program, you can specify the data value to be transmitted in any of several different formats, including hex, ASCII character, and others.

**Submit your program C source file and program verification strategy as part of your prelab.**

### ***Design Task 2: Software UART Receiver Function***

You must write a function named `UART_sw_read`.

```
uint8_t UART_sw_read();
```

This function has the return type `uint8_t`. The `uint8_t` returned by the function must be the right justified ASCII character received. The frame format for the asynchronous transfer must be fixed at 8N1. The baud rate `BAUD_RATE` must be defined by a `#define` macro directive at the beginning of your program, so that the baud rate can easily be changed.

Function `UART_sw_read` must use PB1 as the RX pin for this software receiver. This function must poll PB1 to detect the start of an asynchronous frame. It must then perform a false start check at the middle of the start bit. The function does not return until it has received a full character. This approach would not be a good one in an actual program, because the function would block the continued execution of the entire program until a character is received. If a character is never received the program would be blocked from doing anything else. So a practical program would need to add a time-out feature to the receiver function so that receiver function would only wait for a fixed amount of time and if no character is received in that time, the function would automatically return.

Write a program named `asynch_sw_read` that allows you to read ASCII characters at any of the following three baud rates: 4800, 9600, and 19200 baud. The received character returned by the function must be placed in a global variable. In the laboratory, this program will be used to verify your `UART_sw_read` function. The characters read are visible via the global variable used. The characters read must also be observed as they are being transmitted by using the Saleae logic analyzer. In the main loop of your program simply place a call to the `UART_sw_read` function and place a breakpoint after this call. When the breakpoint is reached you can view the global variable in Studio's Watch window to see the value received.

Your program's duration for a single bit (bit time) needs to be as accurate as reasonably possible.

**Submit your program C source file and program verification strategy as part of your prelab.**

### ***Design Task 3: Software UART Receiver Interrupt Service Routine***

A drawback of the software receiver in Task 2 is that it polls to detect and receive an asynchronous frame. The function does not return until it has detected and received a frame. If the function is used in a larger program that must carry out other tasks in real time, the program blocks waiting for an asynchronous frame to be received.

A better approach would be to use an interrupt to detect the frame's start bit at PB1. You had some experience with pin change interrupts in ESE280.

Modify the function `UART_sw_read` from Task 2 to be an interrupt service routine that services a pin change interrupt request generated by a start bit at PB1.

Write a program named `asynch_sw_read_interrupt` that allows you to read ASCII characters at any of the following three baud rates: 4800, 9600, and 19200 baud. In the laboratory, this program will be used to verify your interrupt service routine's operation. The main loop in your program does not need to do anything. However, after your configuration and initialization code is executed you want execution to stay in the main loop until an interrupt occurs and return to the main loop after the ISR completes. You can write your main loop as follows to keep it from being optimized out of your program.

```
while (1)
{
    //a nop so while loop is not optimized away
    asm volatile ("nop");
}
```

You can observe the character received by your program by having the ISR store it in a global variable and then observing this variable in Studio's Watch window. When a character is typed in Tera Term you must observe the character sent to the Curiosity board using the Saleae logic analyzer. You must then pause program execution to read the global variable's value in the Watch window.

#### ***Design Task 4: Interrupt Echo Program***

Write a program named `interrupt_echo` that combines the code from Task 1 and Task 3 to create a program that uses an interrupt to receive an alphabetic character from Tera Term and echoes the character back to the Tera Term with its case changed. So, if you type a lower case 'a' into Tera Term your program on the AVR128DB48 sends back an upper case 'A'. The characters typed into Tera Term are assumed to be limited to alphabetic characters.

**Submit your program C source file and program verification strategy as part of your prelab.**

#### ***Design Task 5: Message Relay Program***

Modify your program from Design Task 4 to create a program named `interrupt_echo_line` that uses interrupts to receive a line of ASCII characters from the Tera Term. The characters received must be buffered in an 80 character array until a carriage return 'CR' control character (0x0D) is received. Once a carriage return character is received, the program must send (echo or relay) the entire line back to the Tera Term. After a line is echoed back, the Tera Term's cursor must be moved to the beginning of the next line.

**Submit your program's C source file and program verification strategy as part of your prelab.**

#### ***Design Task 6: Distance Measuring Program***

The A02UUUW is a waterproof ultrasonic sensor module with a 4.5m effective ranging distance.



The device's operation is simple. It produces ultrasonic pulses and uses them to measure the round trip time-of-flight of a pulse. It then uses the time-of-flight and the speed of sound to calculate the distance of an object from the sensor. The calculated distance is transmitted from the sensor as four frames of data at 9600 baud.

These frames are transmitted with an 8N1 format. The output from the sensor for a single measurement is in the following form:

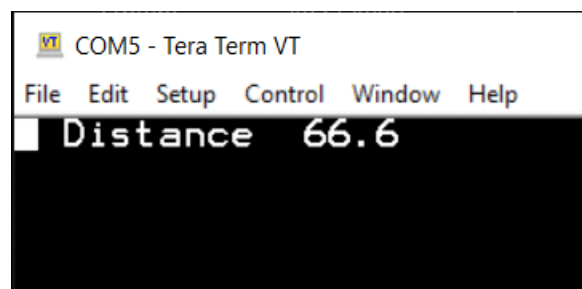
Frame Data	Description	Byte
Header	0xFF	1 byte
DATA_H	Distance Data High 8-bits	1 byte
DATA_L	Distance Data Low 8-bits	1 byte
SUM	Checksum	1 byte

The distance measured, in units of mm, is  $256 * \text{DATA\_H} + \text{DATA\_L}$ .

If the sensor's RX input is allowed to float or is tied high, it will make a measurement every 100 ms and transmit the four bytes that comprise the measurement on its TX output.

In this task, the TX signal from the sensor is connected to the RX input of your software UART. However you will have to change your software receiver to use PB4 instead of PB1. This is necessary to avoid a conflict with the Curiosity Board's Virtual Serial Port. Your software UART must transmit a calculated distance to the Tera Term from its TX output. Draw a block diagram of the Interconnection of the sensor to the AVR128DB48 and the AVR128DB48 to your PC.

Modify your program from Design Task 5 to create a program named `distance` that uses interrupts to receive the characters from the sensor. The characters received from a single measurement must be buffered in a 4 byte memory array. After the four bytes of a measurement are received, your program must compute the distance measured in inches, to a tenth of an inch, and send this value to the Tera Term emulator for display on your PC. The displayed value must appear on the top line in Tera Term monitor window. When a new distance value is sent to the Tera Term it must write over the previous value, so that the result always appears in the terminal window in the same position. and updates every 100 ms.



**Submit your block diagram, program's C source file, and program verification strategy as part of your prelab.**

## Laboratory Activities

### *Laboratory Task 1: Software UART Transmitter Function*

Create a project named `asynch_sw_send`. Connect the Oscilloscope and Saleae Logic Analyzer to your TX signal at PB0 (Curiosity Nano pin 16). **The Curiosity connects to the Tera Term via the Curiosity's CDC TX pin and its USB connection, You do not have to do any wiring for the USB connection.**

Load, debug, and run your program. Measure the bit times for the bits in your transmitted ASCII characters using the Oscilloscope and Saleae Logic Analyzer. Use the Snipping Tool to get a screen capture of a transmitted character as shown on the Saleae Logic Analyzer with the bit times measured and added to the waveform. Compute the percent error in your generated bit width compared to the specified time.

The waveform on the Saleae logic analyzer will indicate to you with white dots where it expected the center of each bit time to be. Those white dots indicate where the Saleae sampled the incoming data signal.

#### *(a) Verification with Oscilloscope*

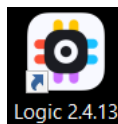
Use the oscilloscope with Single selected as its Run Control to capture your program's output. This output must be similar to that in the previous figure. Use the oscilloscope's trace and its measurement capability to verify the Baud rate.

**When your program is working correctly and generates the required trace, have a TA verify that your program performs as required. Get the TA's signature.**

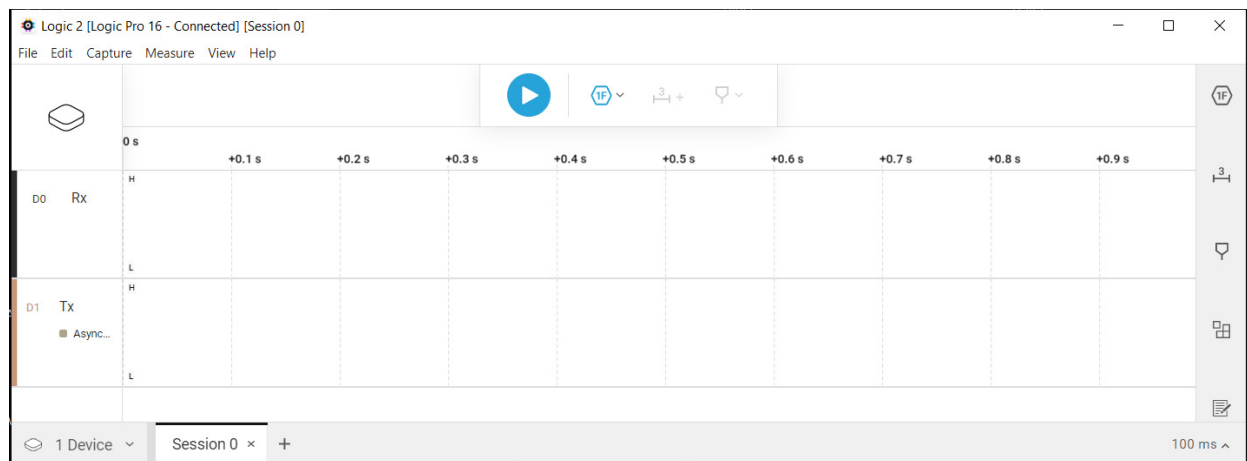
#### *(b) Verification with Saleae Logic Analyzer*

Connect the Saleae logic analyzer's USB connector to a super speed USB connector on the PC. Connect the Saleae logic analyzer's lead 0 and associated ground to the Curiosity Nano board's Tx signal and ground, respectively.

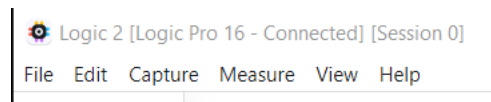
Click on the logic analyzer icon to start its program running.



The logic analyzer's window will open.

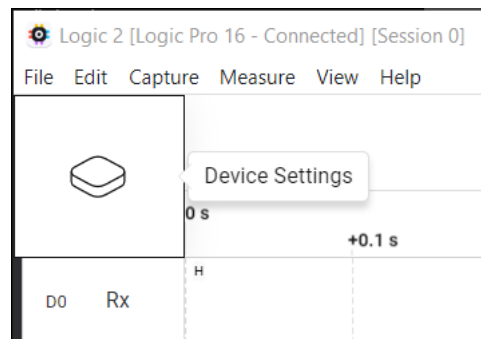


In a few seconds, the text at the top left of the title bar of the application's window, should say Connected.



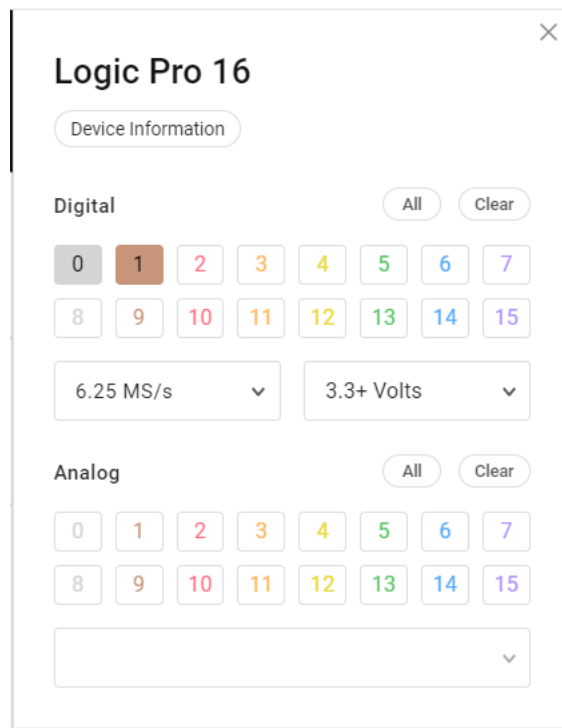
If it says Disconnected, you may actually be in simulation mode and not be getting your displayed data from the AVR128DB48. This can be very misleading because the Saleae demo waveforms may look like the signals you are expecting. If it says Disconnected, you probably have a connection problem with the USB cable.

To make measurements you first need to make any device settings necessary. Click on the box near the top left that has an outline of a Saleae Logic Analyzer in the center of it.





You can now select the number of channels of digital or analog signals to display and the voltage for the logic signals to be observed.



The screenshot shows the 'Logic Pro 16' configuration window. At the top, there is a 'Device Information' tab. Below it, the 'Digital' section is active, showing a grid of 16 channel buttons (0-15). Channel 1 is highlighted in orange, and channel 2 is highlighted in red. To the right of the grid are 'All' and 'Clear' buttons. Below the grid, there are two dropdown menus: the first is set to '6.25 MS/s' and the second is set to '3.3+ Volts'. The 'Analog' section is also visible, showing a grid of 16 channel buttons (0-15) and a dropdown menu below it. The 'All' and 'Clear' buttons are also present for the analog section.

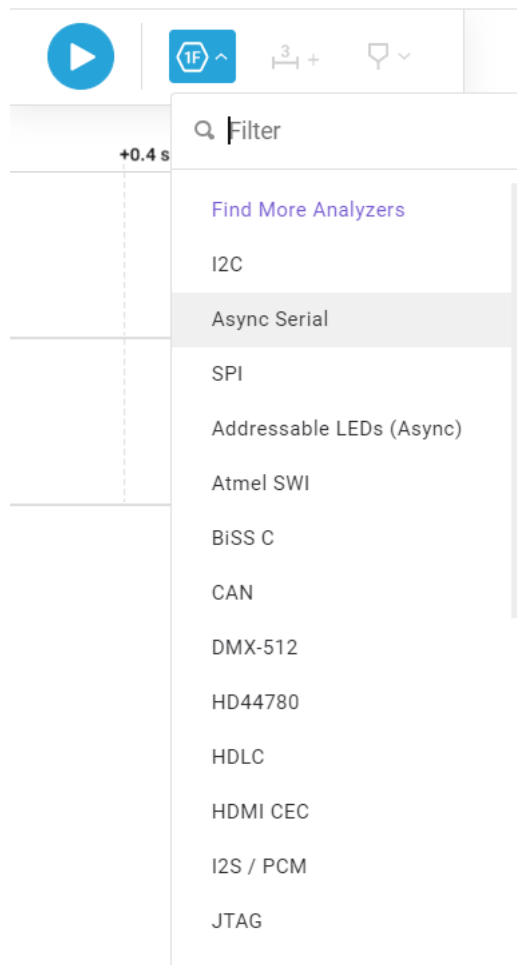
For this task you are only going to look at the Tx signal. So, we only need a single digital channel. However, we have set up two digital channels so that we can look at both transmitted and received signals. We can also show signals in their original analog forms by enabling the corresponding analog channels. Note that the voltage is set for 3.3+ Volts. This is the appropriate selection for any logic operated at a voltage of 3.3V or above. When the 3.3V choice is selected, the logic analyzer uses a 1.65V threshold for determining if a signal is 0 or 1.

The bottom half of this window allows you to set the trigger that starts a capture.

The screenshot shows a configuration window with three tabs: 'Looping', 'Timer', and 'Trigger'. The 'Trigger' tab is selected and highlighted in dark grey. Below the tabs, there is a 'Pattern and Channel' section with a 'reset' button. The pattern is set to a falling edge symbol (a vertical line with a horizontal bar at the top) and the channel is '01. Tx'. Below this is a section for 'Other channel conditions' which is currently collapsed. The 'Capture duration after trigger' is set to '1 s'. There are two toggle switches: 'Trim pre-trigger data' and 'Glitch filter', both of which are currently turned off. At the bottom, the 'Memory buffer size' is set to '3 GB' with minus and plus buttons for adjustment.

In the example above, we have chosen to trigger on the falling edge of channel 01, the Tx signal. This would correspond to the start bit of a character that was being transmitted.

The Saleae logic analyzer can decode many different serial protocols. Of course, we will be using Async Serial for this laboratory, so you need to select Async Serial. Click on the icon in the top center of the window that is a hexagon with 1F in its center.



Select Async Serial.

You need to edit the protocol settings to conform to the protocol that you want. Click the settings icon next to Async Serial and set the parameters of the protocol.

[Async Serial] Async Serial [1] ?

Input Channel \*

01. Tx

Bit Rate (Bits/s)

9600

Bits per Frame

8 Bits per Transfer (Standard)

Stop Bits

1 Stop Bit (Standard)

Parity Bit

No Parity Bit (Standard)

Significant Bit

Least Significant Bit Sent First (Standard)

Signal inversion

Non Inverted (Standard)

Mode

Normal

☒ Show in protocol results table

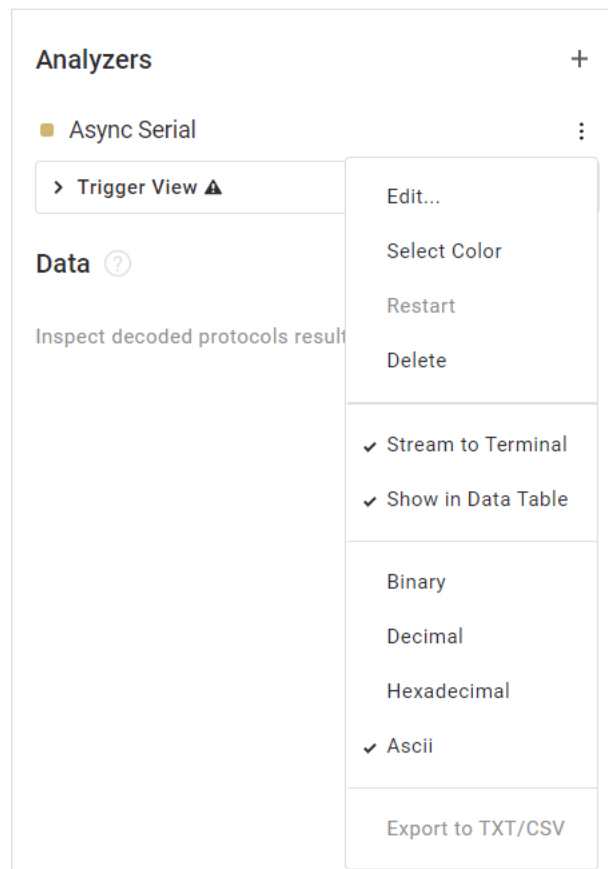
☒ Stream to terminal

Reset

Cancel

Save

Select Stream to Terminal, Show in Data Table and Ascii.



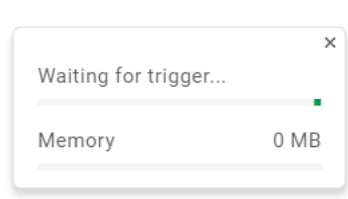
You can click on the text of each signal name in the waveform window to change the channel name to a name appropriate for your application.

The rising or falling edge of one signal must be chosen as the trigger event to start the capture of a signal. Select the falling edge of the Tx signal to trigger the logic analyzer.

Start the logic analyzer running by pressing the blue Start button.



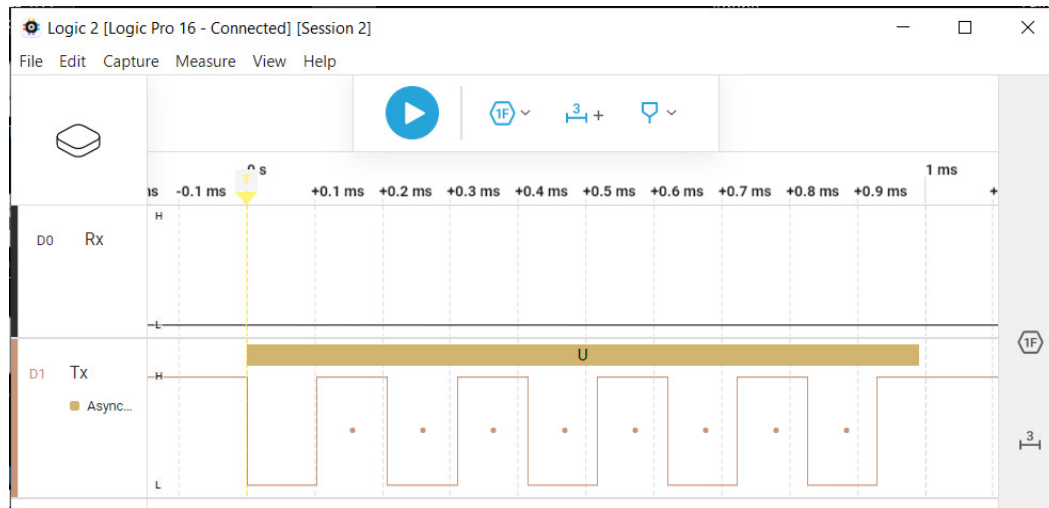
A window will open showing that the logic analyzer is waiting for the trigger event to occur (the negative edge of Tx, corresponding to a start bit).



When you start your program running the trigger event is generated and the analyzer captures the

waveforms.

The Saleae unit displays the waveform and decoded frame value in hex and ascii. The decoded value appears in place of the little white rectangle in the figure that follows. I cut the value out because I want you to determine the data value required to be sent to get the desired waveform.



Once you have captured a waveform, you can right click in the waveform window and select the type of measurement you want associated with the cursor as you move it over the waveform.



In the figure above the cursor has been positioned to measure the width of a single bit. The value returned is 103.04 us. This corresponds well with the desired value of 104.1666666 us computed from  $1/9600$ . Make this measurement for your program and compute the percent error in a bit time. How far from the center, as a percent, will the sample of the stop bit be when a frame is transmitted?

Note that you can click on a signal with the mouse and use the mouse's control wheel to zoom in

or out on the signal.

**Have a TA verify that your program works properly and that your 9600 baud waveform's timing is correct. Get that TA's signature.**

### ***Laboratory Task 2: Software UART Receiver Function***

Create a project named `UART_sw_read`. Connect the Oscilloscope and Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17).

Load, debug, and run your `UART_sw_read` program. Measure the bit times for the bits in your received ASCII characters using the Oscilloscope and Saleae Logic Analyzer. Use the Snipping Tool to get a screen capture of the received characters on the Saleae Logic Analyzer with the white dots indicating when the Saleae Logic Analyzer expects the receiver to sample the data.

**Have a TA verify that your program performs properly and decodes the character read to the correct value. Get that TA's signature.**

### ***Laboratory Task 3: Echo Program***

Create a project named `asynch_sw_read_interrupt`. Connect the Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17) and TX signal at PB0 (Curiosity Nano pin 16).

Load, debug, and run your `asynch_sw_read_interrupt` program. After you type an alphabetic character into the Tera Term you should see the character displayed on the Tera Term monitor, the Saleae Logic Analyzer, and the oscilloscope. Measure the bit times for the bits in your received ASCII characters using the Oscilloscope and Saleae Logic Analyzer. Use the Snipping Tool to get a screen capture of the received characters on the Saleae Logic Analyzer with the white dots indicating when the Saleae Logic Analyzer expects the receiver to sample the data.

**Have a TA verify that your program performs properly by generating an interrupt at the beginning of the start bit and decodes the character read to the correct value. Get that TA's signature.**

### ***Laboratory Task 4: Interrupt Echo Program***

Create a project named `interrupt_echo`. Connect the Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17) and TX signal at PB0 (Curiosity Nano pin 16).

Load, debug, and run your `interrupt_echo` program. After you type an alphabetic character into the Tera Term you should see the character displayed on the Tera Term monitor with the case changed. You can place a breakpoint in your ISR after it has received the character and stored it in a global variable. This will verify that the interrupt occurs and you can verify the received character by viewing its global variable in the Watch window.

**Have a TA verify that your program echoes back each character entered on the Tera Term**

with its case changed.

#### ***Laboratory Task 5: Message Relay Program***

Create a project named `interrupt_echo_line`. Connect the Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17) and TX signal at PB0 (Curiosity Nano pin 16).

Load, debug, and run your `interrupt_echo_line` program. After you type a character into the Tera Term, you can pause execution of your program and verify the received character using a global variable in the Watch window. You can also place your array in the Watch window and observe it fill with the characters you have typed into the Tera Term. As you type characters into the Tera Term, they will not display individually on the Tera Term's monitor. When you type the Enter key on your keyboard a carriage return character will be sent by the Tera Term. When your program receives the CR, it should send back the entire line of characters that you typed including the CR and then it should send a LF so that the Tera Terms cursor is on a new line.

**Have a TA verify that your program buffers and relays back each line of characters entered on the Tera Term.**

#### ***Laboratory Task 6: Distance Measuring Program***

Create a project named `distance`. Connect the Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17) and TX signal at PB0 (Curiosity Nano pin 16).

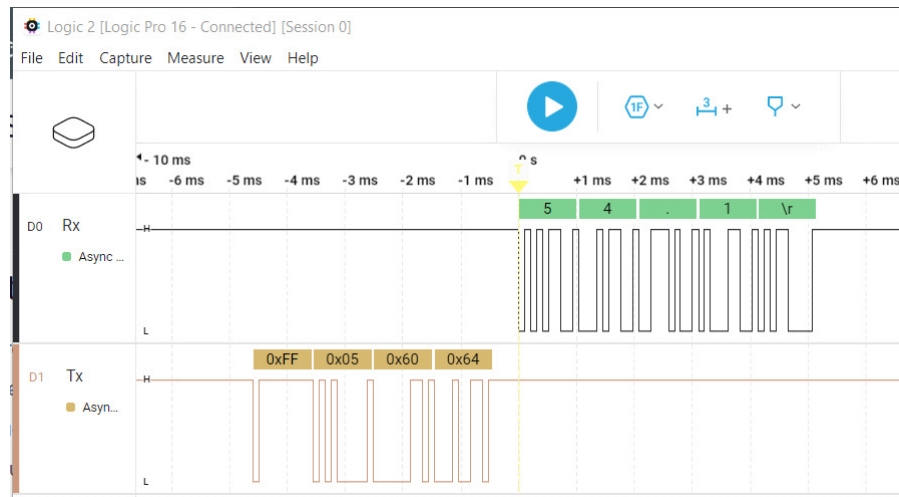
Wire the ultrasonic sensor to your breadboard.

Load, debug, and run your `distance` program. Use a breakpoint so that you can verify that your array in the Watch window is properly filled with the raw data bytes from the sensor. When your program has received a complete measurement it should compute the distance and send it to the Tera Term along with a CR but no LF, so that the value displayed on the Tera Terms remains in the same position on the first line.

Note that the data sent from the sensor is in 8-bit binary, Not ASCII.



Example of output from sensor:



Here Tx is the data from the sensor and Rx is the computed value sent to and received by the Tera Term.

**Have a TA verify that your program displays the correct distance on the Tera Term.**

### Questions

1. What would be the preferred hexadecimal value to send as an initial test of your `UART_sw_write` function from Task 1 and why? Preferred in the sense that it would be the easiest to use to verify the bit times using an oscilloscope or logic analyzer.
2. From the article *Determining Clock Accuracy Requirements for UART Communications* what would you use as the maximum allowable percent error in your bit times for a software transmitter or receiver for the “nasty link” case and the “normal link” case respectively? Assume that the device you are communicating with has no bit time error.  
?
3. If we were to go to very high baud rates, how might that effect the `avr-libc` delay function(s) that you use in your program?
4. If you were implementing an interrupt driven software receiver, as in Task 4, for fairly low baud rates, how could you improve its performance by using one of the counters on the AVR128DB48? Provide a one paragraph explanation of how you would use the counter in the overall implementation of the receiver.