Asynchronous Serial Interface Receiver

Introduction

An Asynchronous Serial Interface is a low-speed serial communication interface mostly used by embedded devices. The simplest version uses a single data line in each direction (Transmit Data and Receive Data) and a ground pin. The RS-232 standard specifies levels of at least ±5 volts but embedded systems often use logic-level signals.

In this lab you will design and implement the receiver of a UART (Universal Asynchronous Receiver-Transmitter) that receives an 8-bit value.

The receiver module will be connected to the LED display and will show the received values as two two-digit hexadecimal values.

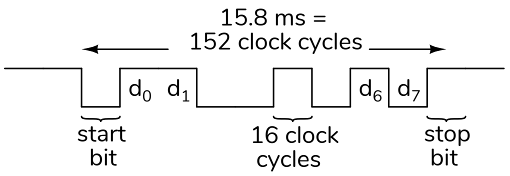
You will test your UART module by transmitting it a string ending with the last two characters of your BCIT ID. The values transmitted will be the ASCII [1] character codes for these characters.

You will use the Analog Discovery 2 (AD2) "UART" protocol analyzer function to generate these test waveforms.

The supplied **lab6b.qar** Quartus project archive contains the required code except for the code in the **receiver** module in the **receiver.sv** file, which you must write.

Specifications

The diagram below shows the asynchronous serial interface waveform output by the AD2 for the value **8'b0101_0011** (8'h53) at 9600 bits per second:

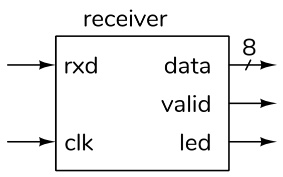


[1]An encoding is a mapping of characters ("glyphs") to numbers. Unicode is the most widely used. ASCII, American Code for Information Interchange, is a subset of Unicode that covers only English-language characters and numbers.

Eight bits per character must be received in order from least-significant bit to most-significant bit. A high logic level should be received as a "1" bit and a low logic level as a "0". An extra "0" ("start bit") is transmitted before the data bits and an extra "1" ("stop bit") is transmitted after the data bits. The signal is high between characters.

Note that this signal is inverted from the conventions used by RS-232 in which a "1" is transmitted as a negative voltage and a "0" as a positive one.

The inputs and outputs of your **receiver** module are shown below:



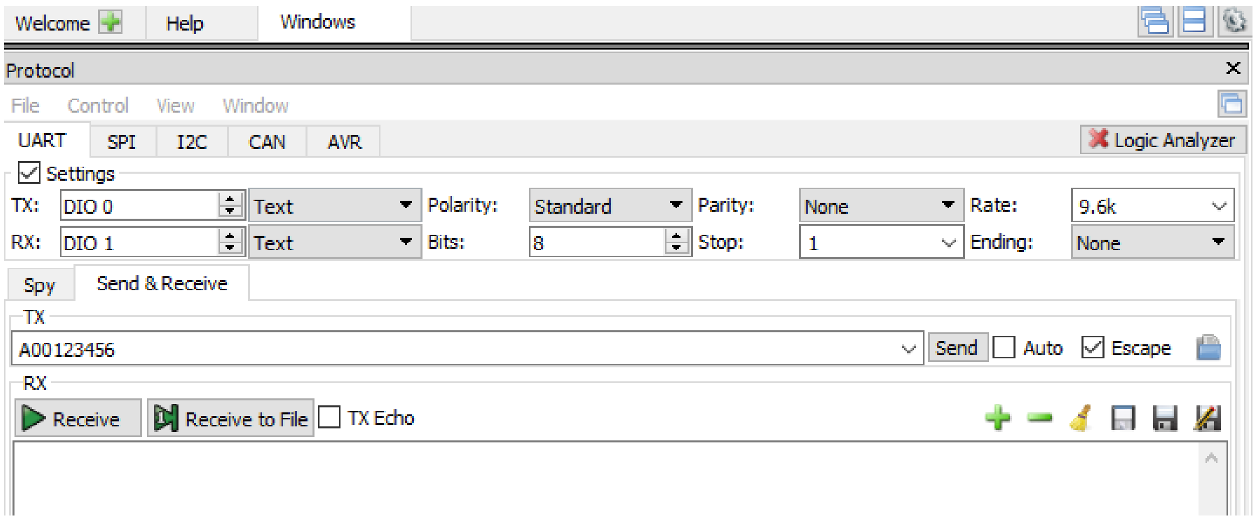
valid should be asserted for one clock period when the value of the **data** output is the eight bits of the most recently received character. The **led** is for troubleshooting and can be set to anything. **rxd** is the serial data and **clk** is a clock at approximately $16 \times 9600 = 153.6$ kHz.

Design Suggestion

As described in the lecture notes, a shift register is a common implementation for the datapath of a serial interface. A state machine controls this shift register.

The suggested controller for an asynchronous serial interface uses a counter. The count value is initialized at the start of the start bit and counts down until the middle of the stop bit (at which point the receiver would wait for the next start bit). This receiver is active for the duration of the start bit, the data bits and the first half of the stop bit. This requires that the counter count down for $m(n+1.5)$ clock cycles where n is the number of data bits and m is the number of clock cycles per bit. The extra 1.5 bit periods include the start bit and half of the stop bit durations.

The received data is shifted into a **data** shift register in the middle of the each bit, including the



Settings: [✓]

TX: DIO 0, Text | Polarity: Standard | Parity: None | Rate: 9.6k

RX: DIO 1, Text | Bits: 8 | Stop: 1 | Ending: None

TX: A00123456 | Auto [ ] | Escape [✓]

RX: TX Echo [ ]

Figure 1: AD2 Protocol Analyzer Window.

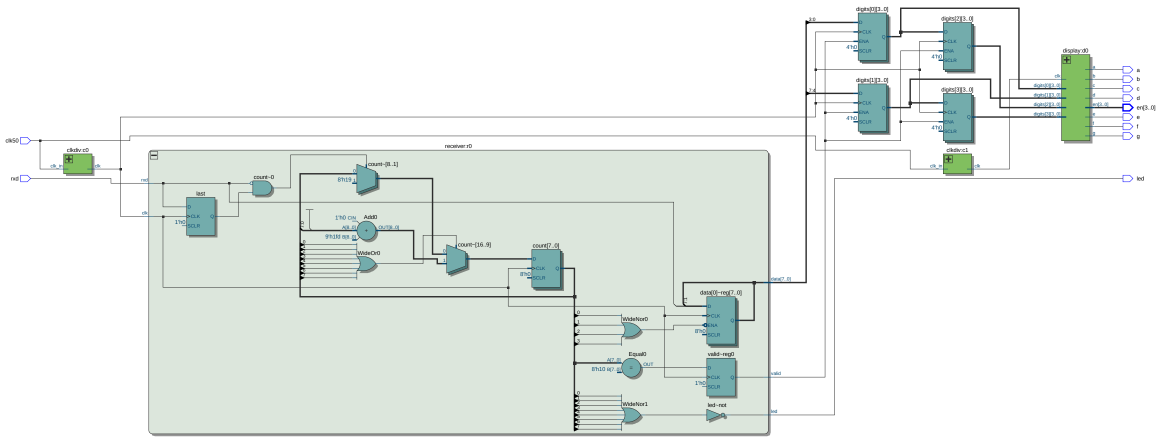


Figure 2: "RTL Netlist" Schematic.

start bit. This requires waiting $m/2$ clock cycles until the middle of the start bit and then waiting m clock bits for subsequent bits. As an example, for $n = 8$ data bits and a clock frequency of $m = 16$ times the bit rate the counter would count down from $16(8+1.5) = 152$ =`8'h98` to zero. Bits are shifted into the shift register when the counter reaches $152 - 8 = 144$ =`8'h90`, $144 - 16 = 128$ =`8'h80`, etc.

Note that the least-significant four bits of all of the above values are `4'h0`. The controller can load the counter with the initial value when a falling edge is detected and decrement it by 1 on each clock cycle while it's not zero. The shift register shifts in a new bit whenever the least-significant four bits are zero.

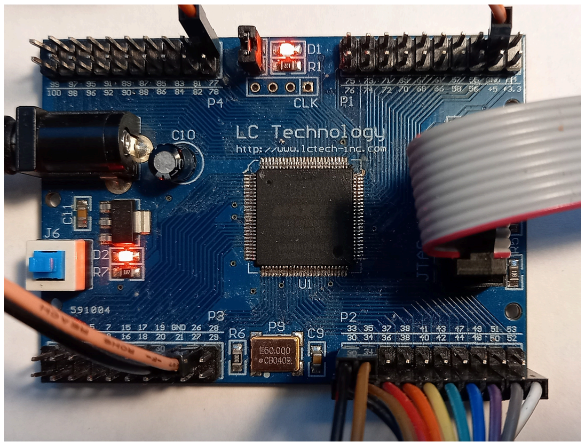
The 'valid' output is true for one clock cycle *after* the final data bit has been shifted in (which happens when the count reaches `8'h10`).

Your design will also need to detect the falling edge at the start of the start bit. The counter value should be set to the starting value if the count value is zero and a falling edge is detected.

The implementation of the datapath is an $n = 8$-bit register. The value on the data input is shifted into the most significant bit at the appropriate `count` values. This results in the first bit ending up in the least-significant bit position (the start bit is shifted out).

CPLD I/O

The default pin assignments for this lab use CPLD pin 26 for the data signal from the AD2 (**rxd**). The pin to the left of pin 26 is a ground and should be connected to one of the AD2's black ground leads. You can connect the 'scope to pin 77 (**led**) for troubleshooting.



The four-digit LED display is connected as in previous labs.

Use of AD2

You will use the AD2 protocol analyzer window to generate the test waveform. You can also use the 'scope window to troubleshoot the operation of your design by monitoring the **led** signal.

Figure 1 shows the AD2 protocol analyzer configured to transmit the characters for BCIT ID A00123456 with no line ending character, at 9600 bits per second, with one stop bit, 8 bits per character, with the output on lead DIO0.

> **Note:** There may be a conflict between Quartus Programmer and the AD2 Waveforms software. You may have to use Waveforms > Settings > Device Manager and temporarily switch to one of the Demo modes to allow the Quartus Programmer to access the USB Blaster.

Procedure

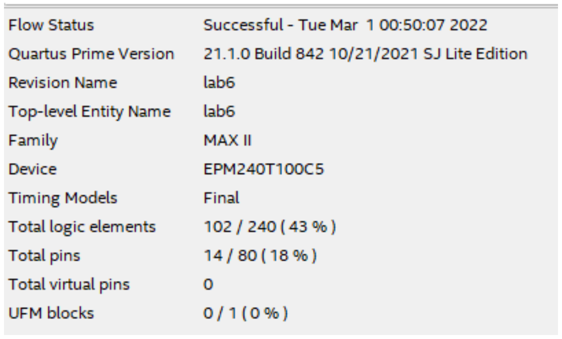
Download the **lab6.qar** Quartus archive and open it with Quartus. Restore it to a convenient directory. Edit the included **receiver.sv** to implement a design that meets the specifications above. Connect the CPLD to the 4-digit LED display as in previous labs. Program the CPLD. Connect the CPLD to the AD2 as shown above. Run the Waveforms software and configure the "Protocol" window as shown above but using your BCIT ID. Press the Send button to sent the data to the CPLD. You should see the ASCII values of the last two characters displayed in hexadecimal. For example, if you BCIT ID ends with 56, the ASCII character codes for the digits '5' and '6' are **8'h35** and **8'h36** and so the display would show 3536.

You can use the **.pof** file on the course web site to check your hardware and AD2 configuration. However, note that this configuration displays the hex values of the least-significant four bits of the most recent four characters received.

Submission

To get credit for completing this lab, submit a PDF document containing the following to the Assignment folder for this lab on the course website:

1. A listing of your **receiver.sv** System Verilog file.
2. The RTL schematic (Tools > RTL Netlist) with the receiver module expanded. This will be similar to Figure 2.
3. A screen capture of your compilation report similar to:



Flow Status	Successful - Tue Mar 1 00:50:07 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	lab6
Top-level Entity Name	lab6
Family	MAX II
Device	EPM240T100C5
Timing Models	Final
Total logic elements	102 / 240 (43 %)
Total pins	14 / 80 (18 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

.

If you do not demonstrate your lab in person, also submit a video showing the initial change to the hex values when you send the last two digits of your BCIT ID.