University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS 150 Fall 2000

Check Point 2
UART Design

1 Objective

In this lab you will design, simulate, implement, an *universal asynchronous receiver / transmitter* circuit (UART).

2 Functional Specification

UARTs are used for communication between two devices. For instance, UARTs are used to connect terminals to computers. They provide a means to send data with a minimum of wires, in our case this is the shared bus. The data is sent bit-serially (one bit at a time), and no clock signal is sent along with it. The primary function of a UART is parallel-to-serial conversion when transmitting, and serial-to-parallel conversion when receiving. The fact that a clock is not transmitted with the data complicates the design of a UART. The two systems (sender and receiver) have separate, unsynchronized, clock signals. Although the two clocks will have roughly the same frequency, they will not have the same phase. Part of a UART's function, and the tricky part, is to "sample" the serial input at just the right time to reliably capture the bit stream. This is done by using a high-speed clock to sample the bit stream multiple times per data bit.

In our application the bit transfer rate, or *baud rate*, is 250kHz. Your FPGA board is equipped with a 16MHz crystal oscillator. A 250kHz clock signal could be generated by dividing the crystal oscillator frequency. We do not actually need that frequency for this lab. The frequency we need is 8 times the baud rate: $8 \times (250\text{kHz}) = 2\text{MHz}$.

Figure 1 shows a functional block diagram of the UART receiver that you will design. Bit-serial data is received on the `SERIAL-IN.H` input. When one byte of data has been received, it is output to the `D` output bus, and the output control signal `DRDY.H` is asserted for one clock period. The block is clocked with a frequency 8 times the baud rate, in this case 250kHz.

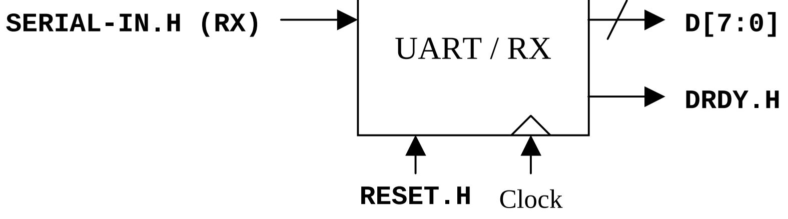


Figure 1: UART Receiver symbol.

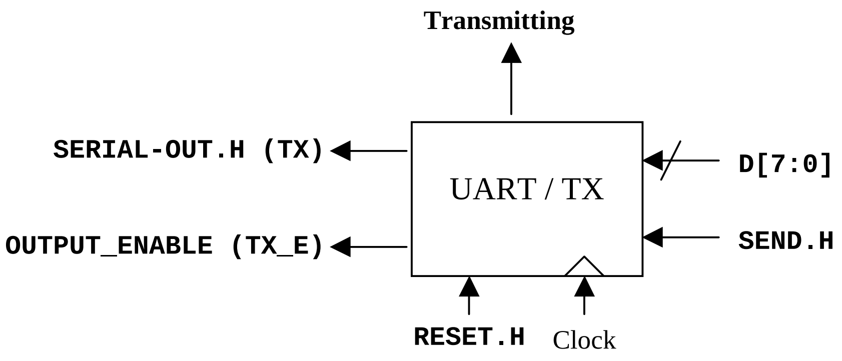


Figure 2: UART Transmitter symbol.

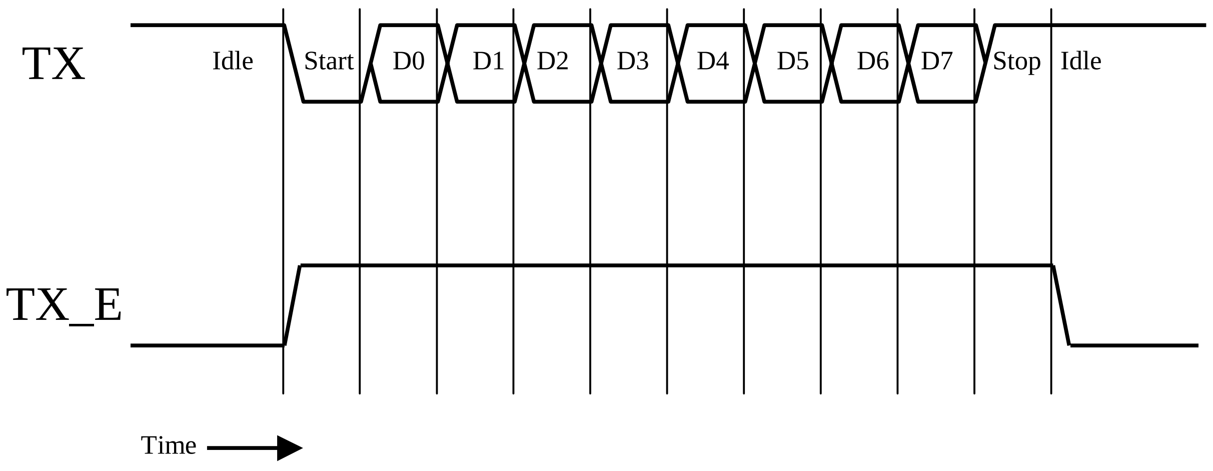


Figure 3: A UART transmission.

Data is transferred one byte at a time to the receiver using the format shown in Figure 3. The transmission character is composed of an 8-bit data byte, sent LSB first, preceded by a start bit (`LOW`) and followed by a stop bit (`HIGH`). When no character is being transmitted, the line is idle (`HIGH`). The line need not go idle between characters, as it is possible for the start bit of a transmission to immediately follow the stop bit of the previous transmission. We also need a TX enable line that enable the line-driver in the transceiver chip when we are transmitting something.

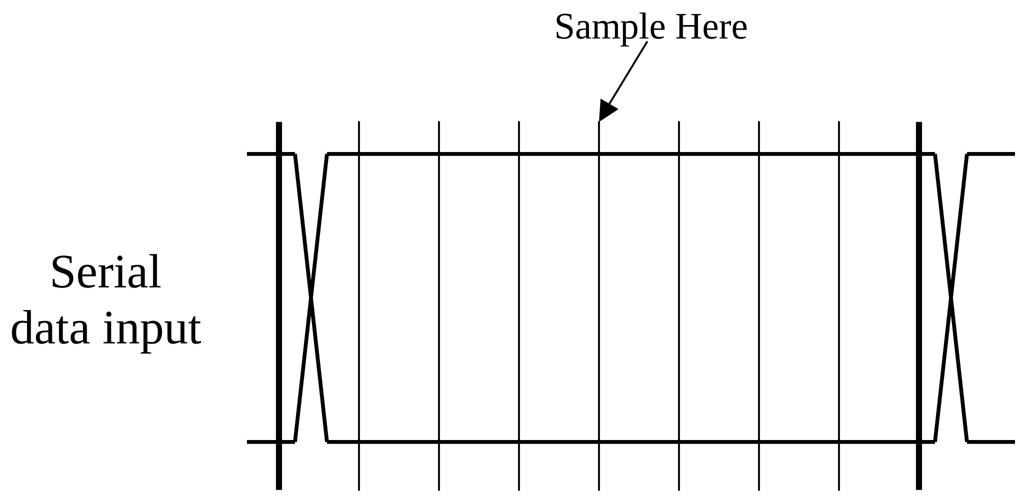
3 Theory of Operation

This section describes the internal operation of the UART receiver that you will design. There are many possible detailed designs – we will not present them all here, but we will give you some ideas.

The transmitter is easier to implement so I will discuss it first. You will probably want to use a loadable shift register to do the parallel to serial conversion. Also since you will be receiving a 8x baud-rate clock you will need to make sure you do the parallel to serial conversion at the right speed (1x baud-rate). This can be achieved by dividing the incoming clock by 8 and using that slower clock for the shift register. Another approach is to generate a shift enable once every 8 cycles (I used this one although it is not clear to me if either method is better than the other). You will also need to generate a transmitting signal to tell the FSM that you are busy and not to send anything else till you are done. This can be done by a counter to keep track of the bits that you have sent or if you are clever just logic on the content of the shift register (if you constructed the shift register the right way). Also when the transmitter is idle the output line should stay 1, and the TX_E line should be 0.

The receiver is a bit more complicated since it is responsible for catching transmission on the asynchronous bus transmitted by another transmitter that probably has a clock with different phase (meaning the other clock does not have to have an edge exactly the same time) and potentially even tiny bit different in period. To counter these difficulties we make the receiver super-sample the data input line. We will use 8x super-sample, this means that even though data is coming at a rate of 250Kbits/s we will be looking at the input 2M times/s.

We will use the super super-sampling approach to find the exact begging of a UART transmission. We can do that by looking for a falling edge on the serial input line (yes I say edge but this has NOTHING to do with a clock input on anything, so don't even think about feeding anything other than a clock into a clock input). An edge detector looks somewhat like debouncer. Once you find the falling edge of that first start bit then you offset your shift register's sampling time so that it will sample at exactly the middle of data bits, right when each bit is "halfway" transmitted. See picture below.



Again there are different ways of doing this one possible way will be covered in the Lab lecture so pay attention to Jeff when he's talking.

In addition to sampling your receiver will also need to keep track of when the line is idle, when it is receiving transmission and when it finished receiving a transmission, and if the transmission was valid (i.e. start bit = 0, stop bit = 1). This is accomplished with an FSM, and probably a counter to make sure you've shifted in the appropriate # of bits. And since you are shifting data in you'd probably want to use a shift register.Your receiver will also need to provide a data present signal to tell the rest of your system that a transmission has been received and present the rest of the system with the byte of data the it received.

Pit fall:

Your UART designs will be clocked at 2MHz as opposed to the rest of your system (8 or 16MHz). This presents an interesting problem when you need it to talk to the rest of the system. See if you can figure this one out yourself. This might not be a big problem for this checkpoint as you can just as well clock the rest of your system at 2MHz but that will not be possible (or at least undesirable) for the rest of the project.

Testing:

To test your design on your own you will need to loop your transmitter's output to your receiver's input. You can do this in your schematic when you are simulating your circuit, however when you are physically testing your circuit you should wire the connection physically with jumpers or wire-wraps outside of the chip.

Design your UART so that the transmitter takes the dipswitches as input (I will have a block for you to take care of the DIPSWITCH input), and transmits one byte every time you push the spare button. Your receiver should display values received by the UART on the Xilinx board's 7-segment display (a pair of LEDs for numeric characters). The circuit should use a data register to record whatever new data is emitted by the UART receiver and output it to the display. As always I will provide an appropriate library for this lab in U:\cs150\… In your schematic, use the CS150 library's `NUMLED1` and `NUMLED2` to display the eight bit output signal (each takes a 4-bit input representing a hex digit 0-F).

Learning to do Pinouts yourself:

Up to now we have given you IO Blocks to handle for signals that travel in and out of the Xilinx chip. For this lab you will draw the schematics for IO yourself. There are only a few signals (TX, TX_E, RX, clk_16MHz, spare, and reset) so it is a relatively painless task. To specify that a signal is to be routed into a pin, a pad symbol is used. Depending on weather the signal is an input, output, or both, a IPAD, OPAD, or IOPAD could be used. To specify to the software the exact pin location of the PAD you will need to and a parameter. You will need to enter "LOC" for the name of the parameter and "P##" for the value of the parameter where ## is the pin number for the PAD.

In order to properly interface with off chip components IBUF and OBUF s are used. For every PAD that you use a corresponding IBUF or OBUF must be used. Take a look at the NUMLED macros to get a better idea of how it is done.

The pin assignments for the 5 signals are:

RX - P68, TX – P37, TX_E – P69, RESET – P56, SPARE – P41, CLK_16MHz – P13.
Reset and Spare are active low so they should be inverted coming in.

4 Pre-lab

Before coming to lab, design an 8-bit UART receiver and transmitter.

Design a clock divider circuit to produce a 2MHz signal from the Xilinx 16MHz clock. It suffices to use a counter from the Xilnx library, with an appropriate choice of bit(s) for output. This clock signal should be distributed to your other circuits using a global clock buffer (BUFG in the library).

To help test your UART in lab, design a circuit to
For the prelab, do the following:

1. Read the data sheets for the following Xilinx library components: (available in the *Xilinx Library Guide*, linked from the course web page)
   CC16CE, CC16RE, CB4CLE, CB4RLE, SR8RE, SR8RLE, BUFG, IBUF, OBUF, IPAD, OPAD, IOPAD.

2. Answer pre-lab questions on the check-off sheet

3. Designs the 3 circuits described above (UART transmitter, receiver, clock-divider).

4. Write a simulation script to simulate the UART receiving a single byte in both of the following cases:
   (a) The line is idle after reset, then a character `0x3E` is transmitted.
   (b) The line is idle, then a character `0xA5` is transmitted.
   Your timing diagram should show clock, `SERIAL-IN.H`, any other control signals, and `DRDY.H`.

5 Lab Assignment

You will physically test your design by having a pair of Xilinx boards communicate over jumper wires. One board will be programmed using a TA-provided bitfile while the other board will use your own design.

Physically connect the RX of one board to the TX of the other board and make another connection the other way around. Also connect the grounds of the two boards together. **Before powering on the boards**, have your TA check-off the connections.

Try sending and receiving different values from both board.

Have your TA check-off your working UART. Your TA may select arbitrary values to send down the jumper cable.

6 Extra Credit

Use the Logic Analyzer to show that you are actually sampling in the middle of each bit and that `DRDY` is being asserted.

7 Acknowledgments

Original lab by J. Wawrzynek (Fall 1994). Modifications by N. Weaver, E. Caspi, N. Zhou.

Name________________________________ Name________________________________

Lab Section (Check one)

M: ☒ AM	T: ☒ AM	W: ☒AM	Th:
☒ PM	☒ PM	☒PM	☒PM

8 Check-offs

Prelab Questions

- Read the data sheets for the following Xilinx library components: (available in the *Xilinx Library Guide*)
  CC16CE, CB4CLE, SR8RE, BUFG CC16CE, CC16RE, CB4CLE, CB4RLE, SR8RE, SR8RLE, BUFG, IBUF, OBUF, IPAD, OPAD, IOPAD.
- Using the library's counters, how would you design a circuit to count a given number of cycles *N*, then assert a 1-bit "done" output? Draw a block diagram.

TA: ________________ (5%)

- Why should the 2MHz clock be distributed to the circuit using a BUFG?

TA: ________________ (5%)

Prelab Design

• UART receiver circuit	TA: ________________	(15%)
• UART transmitter circuit	TA: ________________	(15%)
• Clock divider circuit	TA: ________________	(10%)
• UART simulation script	TA: ________________	(10%)

Schematics and Simulation

• Working UART simulation	TA: ________________	(10%)

Board-to-Board Communication

• Working serial communication	TA: ________________	(30%)
• Verification using logic analyzer	TA: ________________	(5%, extra credit)

Turned in on time	TA: ________________	(×100%)
Turned in 1 week late	TA: ________________	(×50%)