UART TRANSMITTER

This lab has you implement a UART Transmitter module.

BACKGROUND

Review the notes on UART transmission.

TASK DESCRIPTION

You are to implement a UART transmitter that has the following features:

- Uses a hardware FIFO
- Supports multiple baud rates
- Data format is 8 data + 1 start + 1 stop bit

The module interface is given below:

```
module uart_tx(clk, reset, din, dout, wren, rden, txout, addr)
input clk, reset, wren, rden;
input [7:0] din;
output [7:0] dout;
output txout;       //serial data out
input [2:0] addr;

//This is the initial value for the period register for the baud rate
//generator. This timeout value is one-half of the 16x baud rate clock
//period for shifting out bits
parameter PERIOD = 8'h1A;
```

The input/output definition is:

- *clk* – clock input
- *reset* – high true reset
- *rden* – read enable for dout bus
- *din* – data input bus
- *dout* – data output bus, reflects the contents addressed by the *addr* input when *rden* is high, else *dout* is 0.
- *addr* – address bus for internal registers
- *wren* – write line for a register. The register selected by *addr* is written on the rising clock edge when *wren* is high.
- *txout* – serial data out

Register addressing and reset behavior:

Register	Address	Comments	Value at reset

Period Register for baud rate timer	0b000	This is an 8-bit register, can be read and written.	Set to parameter `PERIOD`
TX Reg	0b001	Write port for TX FIFO. This is a write only port, cannot read this port.	Undefined
Undefined (dout = 0 for this)	0b010	unimplemented	undefined
Status/Control	0b011	This is an 8-bit register, can be read and written.	See comments below

Status/Control register definition:

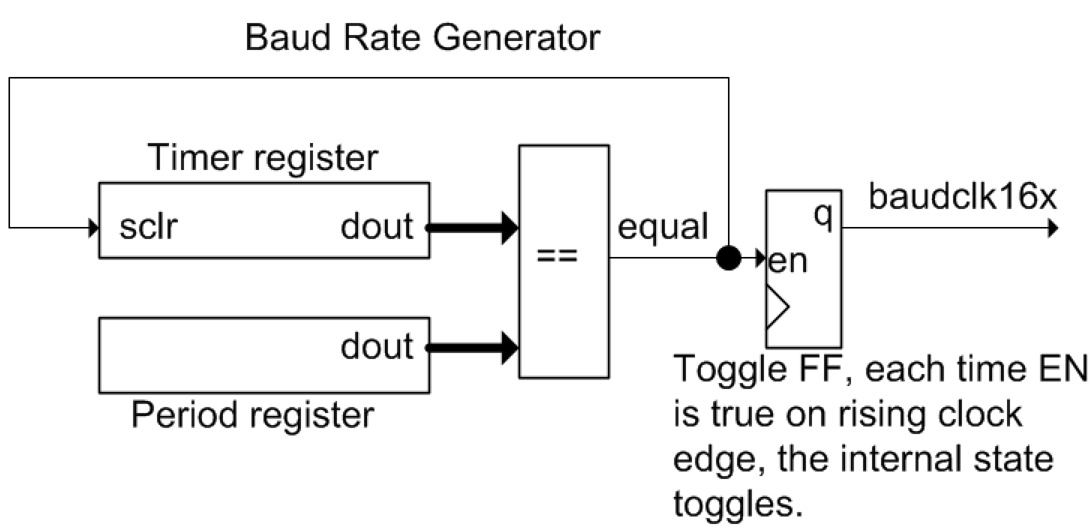
- Bit 0: TXFULL – this is set to a '1' when the transmit FIFO is full. This is a read-only bit, it cannot be cleared by writing to the status register. It is only cleared when the control logic reads the TX FIFO to send data to the internal TX Shift register. To implement this bit, simply pass the FIFO FULL signal out as this bit. The value of this bit at reset is '0' which is configurable when the FIFO is generated.
- Bit 1: TXDONE – this bit is set to a '1' when a byte transmission is finished (including the stop bit!). It can only be cleared by a write to the status/control register
- Bits 2-7: These are currently unimplemented, and should read as 0.

Implementation: TX FIFO

To implement the UART TX block, must use the FIFO that you created in the previous lab. The depth of 8 locations and data width of 8 bits is what you need for this lab. You can tie the synchronous reset input of the FIFO to ‘0’ as it is not needed in this implementation. The ‘reset’ input needs to be tied to the global reset.

Implementation: Baud Rate Generator

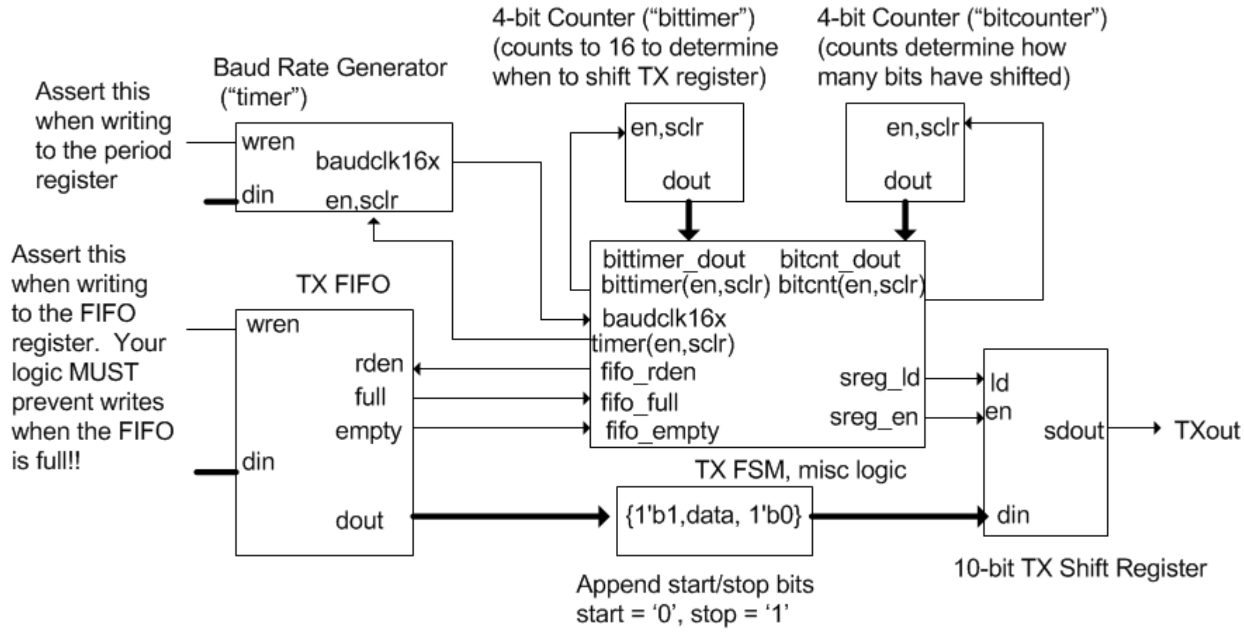
For baud rate control, implement some logic that generates a clock with a period that is 1/16 of a bit time (16 cycles of this clock gives one bit time). A timer with a period register (like what you have previously done) is the approach that must be used for this as shown below.

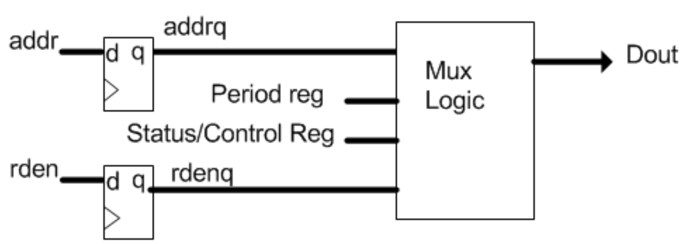


The 16X baud rate clock should be driven from the toggle FF that is toggled each time the timer and period register match. This means that the timeout period will be one-half of the 16X baud rate clock period as two timer matches will generate one complete clock cycle of the 16X baud rate clock.

Implementation: Overall Design

The block diagram below is a diagram of most of the needed hardware. This does not show the baud rate generator above or the status register bits. Note that the DOUT mux logic steering uses registered versions of *addr* and *rden*; this is important.



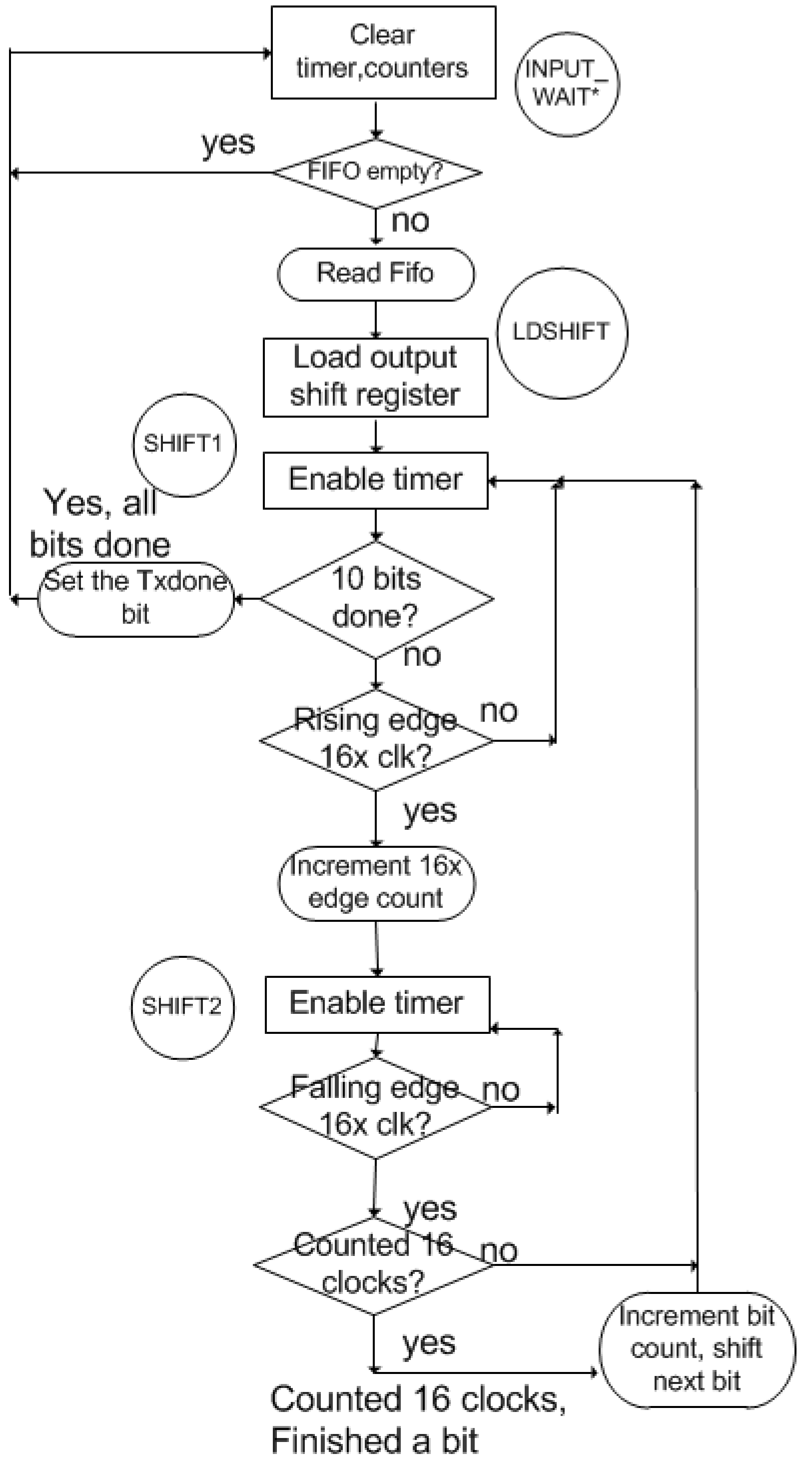


You will need a FSM to control reading data from the TX FIFO, writing it to the TX shift register, and then shifting data out. You only need four states in your FSM. The following is a textual description of the states, a rough ASM chart follows this. You will need to transform this into an implementation.

- INPUT_WAIT_STATE: Stay in this state while the FIFO EMPTY signal is '1'. When FIFO EMPTY becomes '0', assert the FIFO rden (read enable) signal to read the FIFO and go to the LDSHIFT_STATE. The three counters (timer, bittimer, bitcount) should all be disabled and cleared unconditionally in this state.
- LDSHIFT_STATE: This state unconditionally asserts the TX shift register *ld* (load) signal to transfer data from the TX FIFO to the Shift register, and then goes to the SHIFT1_STATE.

- SHIFT1_STATE: The timer should be unconditionally enabled in this state so that it is ticking. If the "bitcounter" count value is 10, then you have shifted all 10 bits and transition back to the INPUT_WAIT_STATE; else if the baudclk16x signal is HIGH, then assert the enable input to the “bittimer” counter so that it increments and transition to the SHIFT2_STATE (this means that you have detected a rising edge on the baudclk16x clock).
- SHIFT2_STATE: Keep the timer unconditionally enabled in this state so that it is ticking. Wait until the baudclk16x signal becomes low (this detects the falling edge of the baudclk16x clock and means the entire bit time has passed). When this occurs, if the "bittimer" counter is 0, this means it has counted up to 15 and wrapped around, so 16 baudclk16x periods have passed meaning that one entire bit time has passed, so increment the "bitccounter" counter by one, and shift the shift register by one position. Regardless of the value of the “bittimer” counter, transition back to the SHIFT1_STATE once baudclk16x becomes low to wait for the next rising edge of the baudclk16x.

This FSM bounces between states SHIFT1_STATE and SHIFT2_STATE while it is serially transmitting a byte, and goes back to INPUT_WAIT_STATE after a byte has been transmitted to determine if there are more bytes in the FIFO.



Some other implementation tips:

1. The shift register should be reset to all "1s" on power up reset so that *txout* starts high.
2. The timer that generates the *baudclk16x* clock is only ticking when you are sending data, so it should be disabled and cleared while you are waiting for data to send. Start it ticking when you are ready to send data.

3. The *txout* line is connected to the LSb of the shift register, and the shift operation is a right shift (from MSb to LSb). The new bit shifted into the MSb should be a '1'.
4. The FSM description above assumes the baudclk16x clock should always start out as '0'. The synchronous clear that clears the 'timer' register in the INPUT_WAIT_STATE should also clear the toggle DFF that is generating the baudclk16x.
5. The *baudclk16x* clock is only being generated when you are sending data – this means you enable and disable the timer appropriately (the timer should be disabled while in INPUT_WAIT_STATE).
6. The *baudclk16x* signal is used as a control input to the FSM (it tests whether it is high or low). The *baudclk16x* signal is NOT a clock signal for any component in this system.
7. The address interface only allows writes to the PERIOD register; writes to the TIMER register are not needed (the timer register should be cleared while waiting for the FIFO to become non-empty).
8. You may be surprised by how many system clock cycles it will take to send just one byte of data. The system clock is 50 MHz, a period of 20 ns. Each transmission is 10 bit times; so a baud rate of 115,200 takes 10 bit times x 1/115,200 = 86,805 ns. The number of system clocks is then 86,805 ns/20 ns = 4341 clocks! Asynchronous serial transmission is SLOW!

Associated files

The ZIP archive associated with this lab contains the following files:

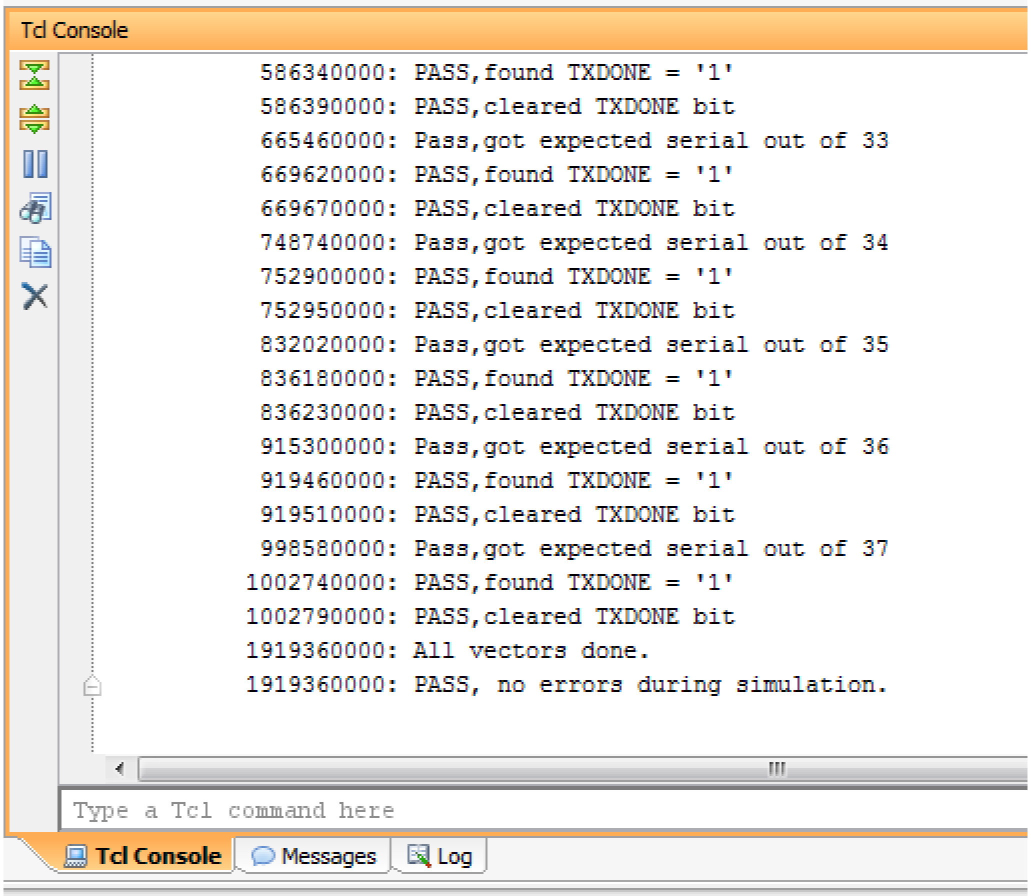
- *uart_tx.v* -- complete this module
- *tb_uart_tx.v* – test bench for the FIFO
- *Basys3_Master.xdc* – constraints files, needs to be added to your project
- *report.doc* – report file that needs to be filled out

uart_tx.v

Use the *empty_template.xpr* project file from lab1, and rename it to lab9_uart_tx.xpr and finish implementing the *uart_tx.v* module. Verify both behavioral simulation and Post-implementation timing simulation using the *tb_uart_tx.v* testbench. The testbench contains a parameter that sets the baud rate of the serial link as shown below, test the design with the 115200 baudrate.

```
//parameter UUT_PERIOD=8'h1A;    //57600 baudrate
parameter UUT_PERIOD=8'h0C;    //115200 baudrate
```

The testbench should be run for 2000 us in order to test all of the vectors.
You will know that all vectors passed when you get the following message:



The full console output for a correct design is here:

```
# run 1000ns
Block Memory Generator module
tb_uart_tx.uut.u0.u0.inst.native_mem_module.blk_mem_gen_v8_2_inst is using a
behavioral model for simulation which will not precisely model memory
collision behavior.
                  260000: PASS, Period register read/write
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_uart_tx_behav'
loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:07 . Memory
(MB): peak = 816.316 ; gain = 0.000
run 2 ms
            79390000: Pass,got expected serial out of 55
            83600000: PASS,found TXDONE = '1'
            83650000: PASS,cleared TXDONE bit
           164190000: Pass,got expected serial out of 1a
           168400000: PASS,found TXDONE = '1'
           168450000: PASS,cleared TXDONE bit
           247520000: Pass,got expected serial out of e2
           251680000: PASS,found TXDONE = '1'
           251730000: PASS,cleared TXDONE bit
           330800000: Pass,got expected serial out of 39
           334960000: PASS,found TXDONE = '1'
           335010000: PASS,cleared TXDONE bit
Checking if FIFO FULL bit is 0
           336450000: PASS, TXFULL bit is expected value
```

```
Checking if FIFO FULL bit is 1
           336990000: PASS, TXFULL bit is expected value
           415570000: Pass,got expected serial out of 30
           419780000: PASS,found TXDONE = '1'
           419830000: PASS,cleared TXDONE bit
           498900000: Pass,got expected serial out of 31
           503060000: PASS,found TXDONE = '1'
           503110000: PASS,cleared TXDONE bit
           582180000: Pass,got expected serial out of 32
           586340000: PASS,found TXDONE = '1'
           586390000: PASS,cleared TXDONE bit
           665460000: Pass,got expected serial out of 33
           669620000: PASS,found TXDONE = '1'
           669670000: PASS,cleared TXDONE bit
           748740000: Pass,got expected serial out of 34
           752900000: PASS,found TXDONE = '1'
           752950000: PASS,cleared TXDONE bit
           832020000: Pass,got expected serial out of 35
           836180000: PASS,found TXDONE = '1'
           836230000: PASS,cleared TXDONE bit
           915300000: Pass,got expected serial out of 36
           919460000: PASS,found TXDONE = '1'
           919510000: PASS,cleared TXDONE bit
           998580000: Pass,got expected serial out of 37
          1002740000: PASS,found TXDONE = '1'
          1002790000: PASS,cleared TXDONE bit
          1919360000: All vectors done.
          1919360000: PASS, no errors during simulation.
```

The testbench tests several character transmissions, and you may be surprised at how many clock cycles it will take to simulate all vectors. For just one character, the amount of time needed is:

1 character times = 1/baudrate x 10 bits x 1 characters.

For a baud rate of 115,200 this time would be:

1 character times = 1/115200 x 10 bits x 1 characters = 86.8 us.

The number of system clocks @50 MHz this requires is 86.8/ 20 ns = 3000 us/20 ns = 3,000,000 ns / 20 ns = 4340 clocks!

Fill out the requested information in the *report.doc* file.

Submission

For submission, create a directory named 'lab9_*netid*', i.e., (lab9_rbr5).

Copy your *lab9_uart_tx* project directory to this directory.
Copy your completed *report.doc* to this directory.

Create a ZIP archive of this directory and submit it.