ECEN 468 Advanced Logic Design
Department of Electrical and Computer Engineering
Texas A&M University

(Lab exercise created by Jaeyeon Won and Jiang Hu)

Lab 2

Design of UART Transmitter

Purpose:

In this lab, we will design a transmitter of Universal Asynchronous receiver/transmitter (UART) with SystemC, and will use Vista as a tool for simulation and verification. This module will be attached to our entire system later, so can transmit data to other device or processors.

Preparation:

The UART is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as EIA RS-232, RS-422 or RS-485. The universal designation indicates that the data format and transmission speeds are configurable. Figure 1 shows general description of communication between processors through a serial channel. Those processors communicate internally with parallel data to speed up and use a serial channel to communication with other processors to reduce the number of wires, so decrease the cost of hardware.

Figure 1 Processor/modem communication over a serial channel

For this lab, a UART transmits 8-bits data without augmenting by a parity bit that can be used for error detection. For transmission, the modem wraps this 8-bit word with a start-bit in the least significant bit

(LSB), and a stop-bit in the most significant bit (MSB), resulting in the 10-bit word format shown in Figure 2. The first 9 data bits of the word are transmitted in sequence, beginning with the start-bit, with each bit being asserted at the serial line for one cycle (bit-time) of the modem clock. The stop-bit may assert for more than one clock.

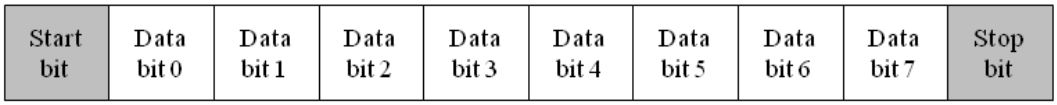

Figure 2 Data format for ASCII text transmitted by a UART

The simplified architecture of a UART presented in Figure 3. It shows the signals used by a host processor to control the UART and to move data to and from a data bus in the host machine.

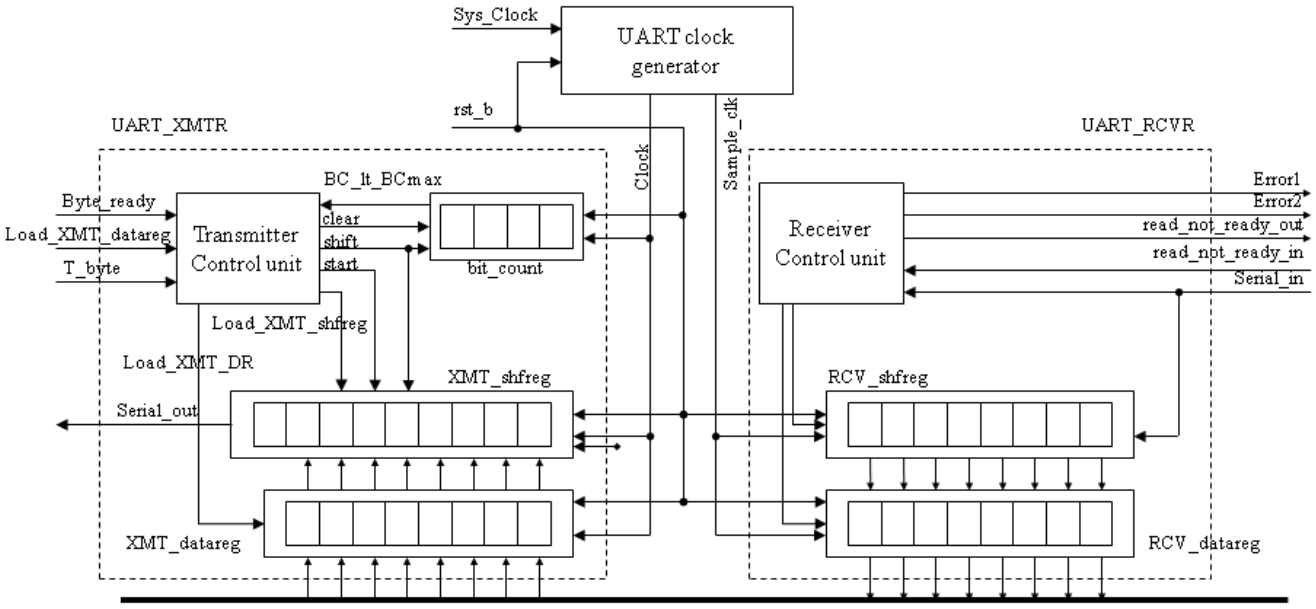

Figure 3 Block diagram of a UART

The input signals are provided by the host processor, and the output signal is the serial data stream. The architecture of the transmitter consists of a control unit, a data register (XMT_datareg), a data shift register (XMT_shfreg), and a status register (bit_count), which counts the bits that are transmitted.

The controller has the inputs (primary/external and status (from the datapath)) listed below. We note that the signal *Load_XMT_datareg* could be passed directly to the datapath unit; instead, we pass *Load_XMT_datareg* to the control unit and assert *Load_XMT_DR* conditionally when the state is *idle* and the external signal *Load_XMT_datareg* is asserted. The status signal, *BC_lt_BCmax*, is asserted while bits are being sent, i.e., if *Bit_count* < *word_size* +1.

a. *Load_XMT_datareg*: assertion is state idle asserts *Load_XMT_DR*, which loads the content of the *Data_Bus* into *XMT_data_reg*

b. *Byte_ready*: assertion causes *Load_XMT_shfreg* to assert, which loads the contents of *XMT_datareg* into *XMT_shfreg*

c. *T_byte* : assertion initiates transmission of a byte of data, including the stop, start, and parity bits

d. *BC_lt_BCmax*: indicates status of the bit counter in the datapath unit

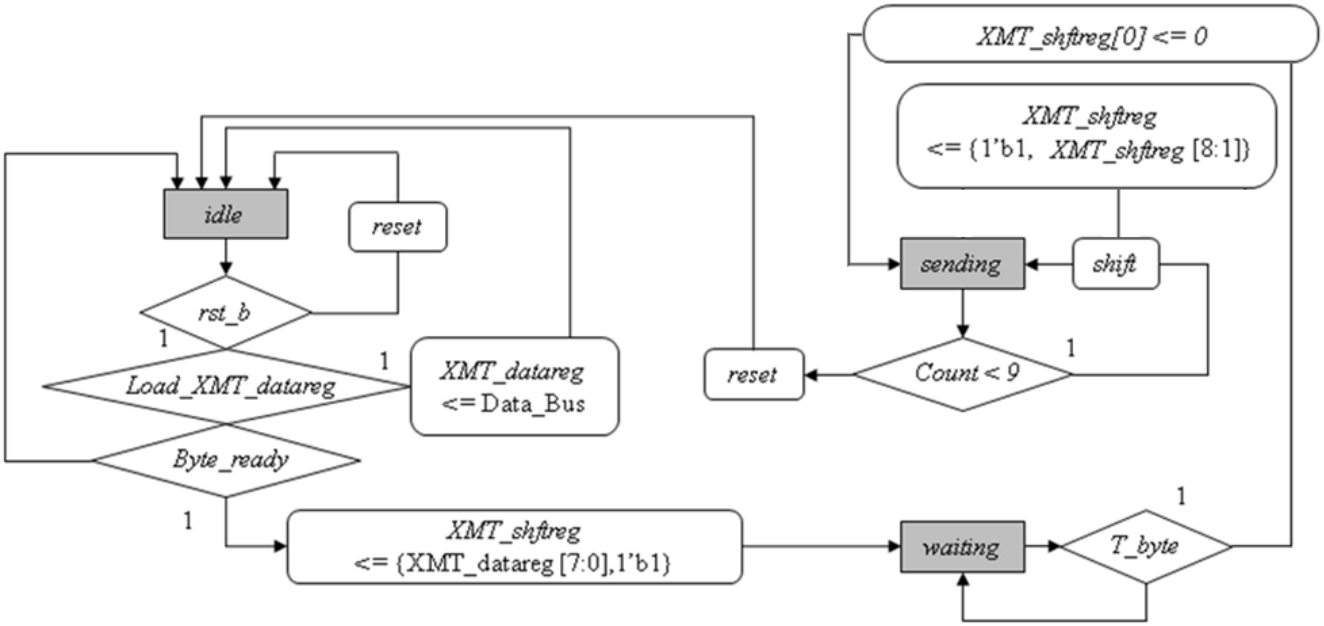

Figure 4 Algorithmic State Machine and Datapath Chart(ASMD) for the UART transmitter

The ASMD chart of the state machine controlling the transmitter is shown in Figure 4. The machine has three states: *idle*, *waiting*, and *sending*. When the active-low, synchronous reset signal *rst_b* is asserted, the machine enters *idle*, *bit_count* is flushed, and *XMT_shftreg* is loaded with 1s. In *idle*, if an active edge of *Clock* occurs while *Load_XMT_data_reg* is asserted by the external host, it will load *XMT_data_reg* with the contents of *Data_Bus*. The machine remains in *idle* until start is asserted to drop *XMT_shftreg*[0].

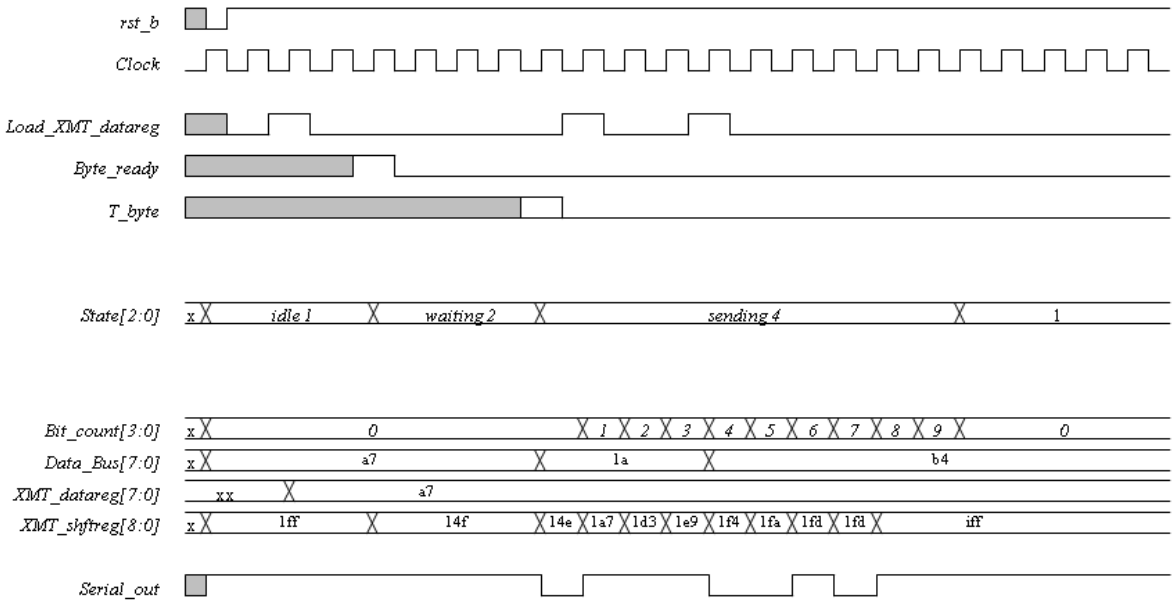

Figure 5 simulation results for the 8-bit UART transmitter

Figure 5 shows an example of timing of transmission. You can design your testbench referring this timing graph. Also, you can check your result based on this timing graph.

3. Functional Simulation

Once the design is completed it must be tested. The functionality of the design module can be tested by applying a testbench and checking the results. The testbench module can instantiate the design module and directly drive the signals in the design module. The testbench can be complied along with the design module. At the end of compilation the simulation results will be displayed.

Download 'Lab2_code.tar.gz' file from the lecture website. The file contains UART_XMTR.cpp, UART_XMTR.h, test.h, test.cpp and main.cpp. You will design modules in these files after you decompress the file.

a. Make working folder for this lab.

HOME]$ mkdir ECEN468/Lab2/SRC (make work folder)
HOME]$ cd ECEN468/Lab2/SRC (move to work folder)
Move the file 'Lab2_code.tar.gz' into the working folder
ECEN468/Lab2/SRC]$ tar xvfz Lab2_code.tar.gz (decompress the file)

Please check the files decompressed. It should contain the files below.
UART_XMTR.cpp, UART_XMTR.h, test.h, test.cpp and main.cpp

b. Insert your code into module files (UART_XMTR.cpp, UART_XMTR.h).
c. Insert your code into the top module (main.cpp).

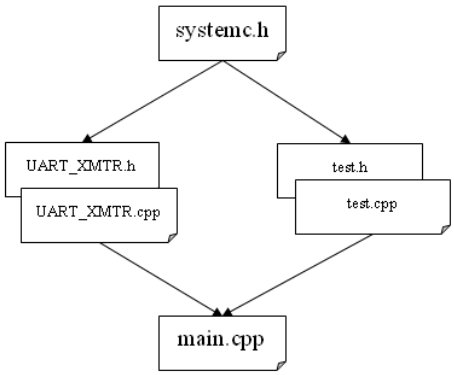

Figure 6 Hierarchy of the files for the system which will be design in this lab

Please refer to Lab1 for about how to use simulator.

d. When you complete your simulation, please show your result to TA within your regular lab hour to get credit for lab evaluation.

Requirements:

1. General requirements.
   a. It should control UART correctly.
      - Protocol : Start bit(‘0’)+Data bits(8bits)+Stop bit(‘1’)
      - Input Ports : Load_XMT_datareg, Byte_ready, T_byte, rst_b, Data_Bus, clk
      - Output Port : Serial_out
      - All behaviors should be synchronized by positive edge of clock signal and negative edge of reset signal
   b. Detailed comments.
   c. Please do not change the names of the signals given and the other functions not mentioned in the report. (You will lose some points if you change any names of the input or output signals.)
   d. You may modify the test-bench given to get the results only if it is resaonable. Please write detailed comments if you have modified.
   e. Late penalty : 20% of total score will be deducted on each subsequent weekday after due date.

2. Submit hardcopy of your report to TA. The report should include contents below.
   a. UART_XMTR.cpp, UART_XMTR.h (5 points)
   b. main.cpp (5 points)
      - Please use named connection to connect UART and positional connection to connect testbench.
   c. sim.out (test message output) with analysis. (10 points)
      - The message should show the bit data transferred using ‘cout’.
   d. Captured waveform with analysis. (10 points)
   e. Q1 : When UART transmitter sends data to UART receiver, which information does UART receiver need to receive data correctly? (5 points)

Note : Please send your source codes, testbench, results(sim.out and *.vcd) to TA’s email address. (5 points)