```
//=============================================
// Function : UART Transmitter
//=============================================
#include "UART_XMTR.h"

  // ----- Code Starts Here -----

  void UART_XMTR::Send_bit() {
        switch(IntState)
        {
                case STATE_IDLE:
                        if(Load_XMT_datareg.read())
                            // Get data register ready
                            bit_count = 0;
                            XMT_datareg = Data_Bus.read();
                            NextIntState = STATE_IDLE;
                        if(Byte_ready.read()){
                            // Set shift register equal to data
                            XMT_shftreg = XMT_datareg;
                            // Shift right and set low bit to 1
                            XMT_shftreg = XMT_shftreg << 1;
                            XMT_shftreg[0] = 1;
                            NextIntState = STATE_WAITING;
                        } else {
                            // We need to wait until we have our data byte ready
                            NextIntState = STATE_IDLE;
                        }
                        break;
                case STATE_WAITING:
                        if(T_byte.read()){
                            // We’re ready to send so set lower bit to 0 for start
                            XMT_shftreg[0] = 0;
                            NextIntState = STATE_SENDING;
                        }
                        else {
                            // Loop until T_byte is good
                            NextIntState = STATE_WAITING;
                        }
                        break;
                case STATE_SENDING:
                        if(bit_count < WORD_SIZE + 1){
                            // Continue to shift right and shift 1 in
                            // Since 1 is our stop bit
                            cout << "Sending " << XMT_shftreg[0] << endl;
                            Serial_out.write(XMT_shftreg[0]);
                            XMT_shftreg = XMT_shftreg >> 1;
                            XMT_shftreg[WORD_SIZE - 1] = 1;
                            bit_count++;
                        } else {
                            // Reset back to idle when done
                            NextIntState = STATE_IDLE;
                            XMT_shftreg = 0x1ff;
                        }
                        break;
        default:{
                    NextIntState = STATE_IDLE;
                }
        }
        Serial_out.write(XMT_shftreg[0]);
  }

  void UART_XMTR::Initialize() {
        if(!rst_b.read()){
                IntState = STATE_IDLE;

                XMT_shftreg = 0x1ff;
                bit_count = 0;
        }
        else {
                IntState = NextIntState;
        Send_bit();
        }
  }
```

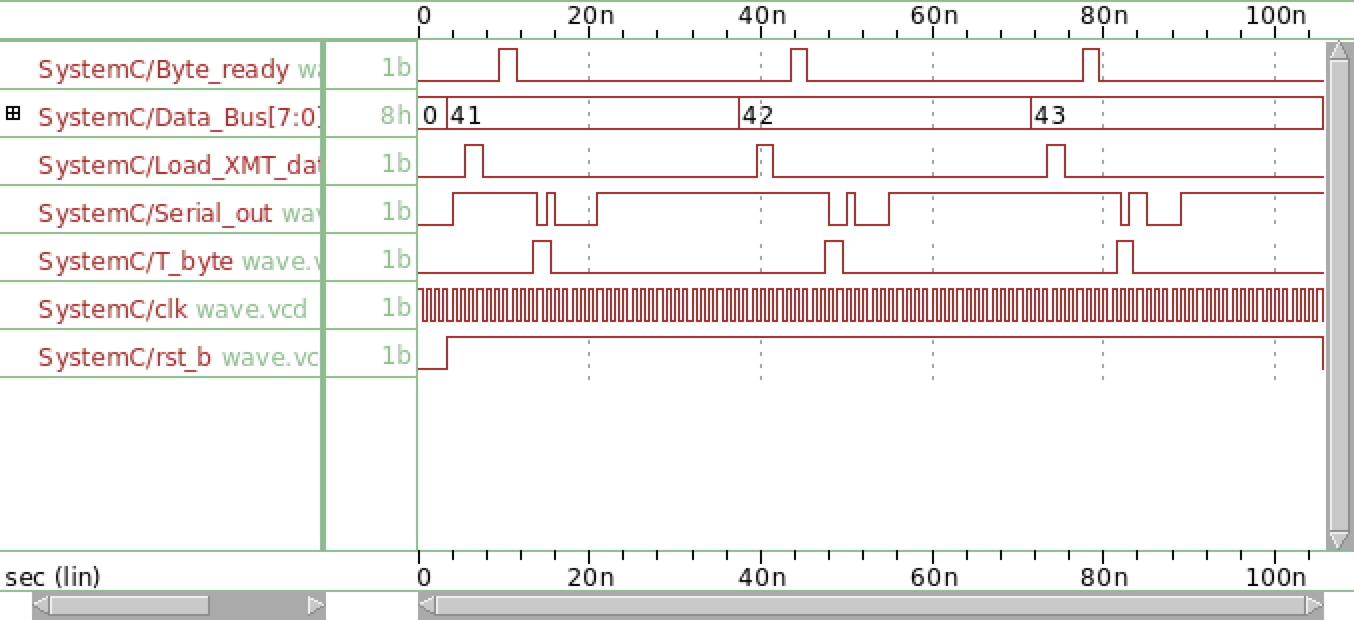
```
//=============================================
// Function : UART Transmitter
//=============================================
#include "systemc.h"

// Define Variables
#define SIZE_BIT_COUNTER        3
#define WORD_SIZE               8

#define STATE_IDLE              0
#define STATE_WAITING           1
#define STATE_SENDING           2

//#define _DEBUG_

// Module Definition
SC_MODULE (UART_XMTR){
        // Input/Output Signals
        sc_in < bool >                  Load_XMT_datareg;
        sc_in < bool >                  Byte_ready;
        sc_in < bool >                  T_byte;
        sc_in < bool >                  rst_b;
        sc_in <sc_uint<WORD_SIZE> > Data_Bus;

        sc_out < bool >                 Serial_out;

        sc_in  < bool >                 clk;

        // Internal Variables
        sc_uint < SIZE_BIT_COUNTER >    IntState, NextIntState;
        sc_uint < WORD_SIZE >                   XMT_datareg;
        sc_uint < WORD_SIZE+1 >                 XMT_shftreg;
        sc_uint < SIZE_BIT_COUNTER+1 >  bit_count;

        // Functions Declaration
        void Send_bit();
        void Initialize();

        // Constructor for the SC_MODULE
        // sensitivity list
        SC_CTOR(UART_XMTR) {
            SC_METHOD(Send_bit);
                //sensitive << Load_XMT_datareg;
            SC_METHOD(Initialize);
                // syncronize on positive clk and negative reset
                sensitive << clk.pos() << rst_b.neg();
        }
};
```

```
#include "UART_XMTR.h"
#include "test.h"

int sc_main (int argc, char* argv[]) {

        // Input/Output Signal
        sc_signal < sc_uint<WORD_SIZE> >        Data_Bus;
        sc_signal < bool >                      Load_XMT_datareg;
        sc_signal < bool >                      Byte_ready;
        sc_signal < bool >                      T_byte;
        sc_signal < bool >                      rst_b;

        sc_signal < bool >                      Serial_out;

        // Clock Generation
        sc_clock clk("clk", 1, SC_NS);

        // Connect the DUT
        // Method 1. Named Connection
        UART_XMTR UART_XMTR_01("SIMULATION UART");
            UART_XMTR_01.Data_Bus(Data_Bus);
            UART_XMTR_01.Load_XMT_datareg(Load_XMT_datareg);
            UART_XMTR_01.Byte_ready(Byte_ready);
            UART_XMTR_01.T_byte(T_byte);
            UART_XMTR_01.rst_b(rst_b);
            UART_XMTR_01.Serial_out(Serial_out);
            UART_XMTR_01.clk(clk);

        // Method 2. Positional connection
        test TEST_01("TEST UART");
        TEST_01(Load_XMT_datareg,
                Byte_ready, T_byte,
                rst_b, Data_Bus, Serial_out, clk);
        // Open VCD file
        sc_trace_file *wf = sc_create_vcd_trace_file("wave");

        // Dump the desired signals
        sc_trace(wf, Load_XMT_datareg, "Load_XMT_datareg");
    sc_trace(wf, Byte_ready, "Byte_ready");
        sc_trace(wf, T_byte, "T_byte");
        sc_trace(wf, rst_b, "rst_b");
        sc_trace(wf, Data_Bus, "Data_Bus");
        sc_trace(wf, Serial_out, "Serial_out");
        sc_trace(wf, clk, "clk");

        // Time to simulate
        // Simulate until it meets sc_stop() if sc_start(-1) or sc_start()
        //sc_start(1000, SC_NS);        // Simulate for 1000ns
        sc_start();

        // Close the dump file
        sc_close_vcd_trace_file(wf);

        return 0;       // Terminate simulation
}
```

```
Built September 15, 2011. License version 2011.9.
Copyright (c) 2005-2011, Mentor Graphics Corporation.

             SystemC 2.2.0 --- Sep 15 2011 00:24:25
        Copyright (c) 1996-2006 by all Contributors
                    ALL RIGHTS RESERVED
(vista) spawn /opt/coe/mentorgraphics/vista312/linux64/tools.64bit/bin/gdb -quiet -interp
=opengdb -runtcl="/opt/coe/mentorgraphics/vista312/generic/tcl/v2/gdb/gdb.tcl" --nx

(vista) BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
Vista SystemC 2.2 Runtime Kernel.
Built September 15, 2011. License version 2011.9.
Copyright (c) 2005-2011, Mentor Graphics Corporation.

             SystemC 2.2.0 --- Sep 15 2011 00:24:25
        Copyright (c) 1996-2006 by all Contributors
                    ALL RIGHTS RESERVED

Warning: (W506) illegal characters: SIMULATION UART substituted by SIMULATION_UART
In file: /vista/Vista32x/64bit/v2/Vista312release/src/tlm2.0.1/../CPP/systemc22/sysc/impl
/kernel/sc_object.cpp:267

Warning: (W506) illegal characters: TEST UART substituted by TEST_UART
In file: /vista/Vista32x/64bit/v2/Vista312release/src/tlm2.0.1/../CPP/systemc22/sysc/impl
/kernel/sc_object.cpp:267
Merging /home/ugrads/s/seth.barberee/ECEN468/Lab2/UART/build/D_PRJDIR_/main.exe ...Done.
Saving types data file /home/ugrads/s/seth.barberee/ECEN468/Lab2/UART/sim/main.db...Done.
WARNING: Default time step is used for VCD tracing.

@3500 ps:: >> START SENDING: 0x41
Sending 0
Sending 1
Sending 0
Sending 0
Sending 0
Sending 0
Sending 0
Sending 1
Sending 1

@37500 ps:: >> START SENDING: 0x42
Sending 0
Sending 0
Sending 1
Sending 0
Sending 0
Sending 0
Sending 0
Sending 1
Sending 1

@71500 ps:: >> START SENDING: 0x43
Sending 0
Sending 1
Sending 1
Sending 0
Sending 0
Sending 0
Sending 0
Sending 1
Sending 1
SystemC: simulation stopped by user.
Program is about to exit.
#0  0x00000000004e9690 in summit_sc::Runtime::atExitCallback ()
(vista)
```

1) When UART transmitter sends data to UART receiver, which information does UART receiver need to receive data correctly?

The receiver needs the start and stop bit to receive the data correctly.


0
20n
40n
60n
80n
100n
SystemC/Byte_ready wa
1b
SystemC/Data_Bus[7:0]
8h
0
41
42
43
SystemC/Load_XMT_dat
1b
SystemC/Serial_out wav
1b
SystemC/T_byte wave.v
1b
SystemC/clk wave.vcd
1b
SystemC/rst_b wave.vc
1b
sec (lin)
0
20n
40n
60n
80n
100n