Data Communication Module Designs

Alex Yang
Engineering School, NPU

UART Design

- Introduction to UART

- Transmitter Design

- Receiver Design

- Clock Generator

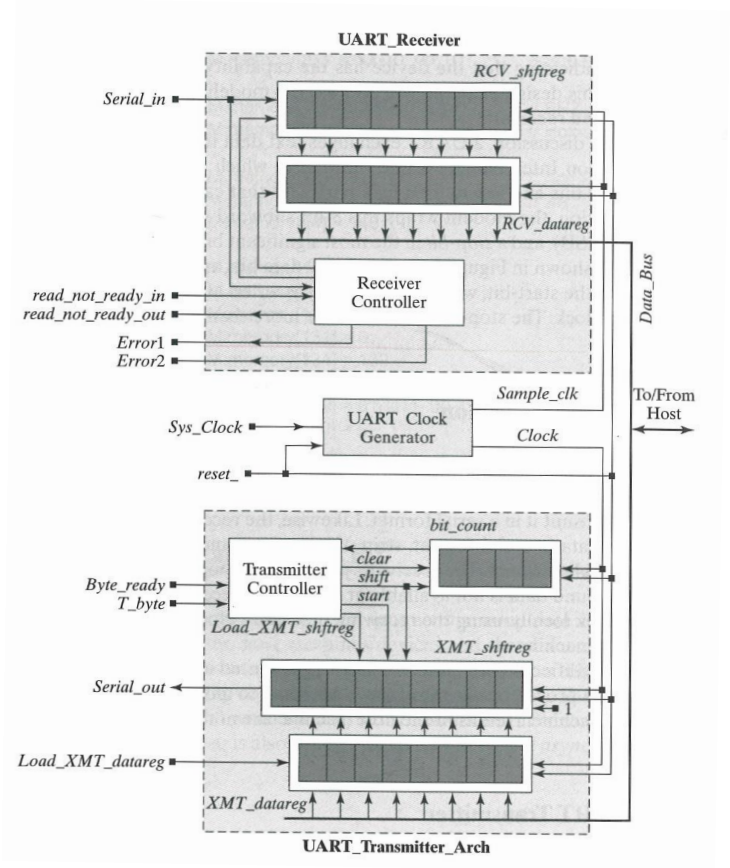
- Top Level Design

I. Introduction:

UART: Universal asynchronous receiver/transmitter. UARTs are commonly used in conjunction with communication standards such as EIA_RS-232, RS-422 or RS-485. Bit0-bit7 are ASCII code and "Parity" is for error check. UART will add "start" and "stop" bits for communication with outside device.

Stop Bit	Parity Bit	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0	Start Bit

UART Architecture:



II. UART Transmitter Design

- UART Transmitter Signals:

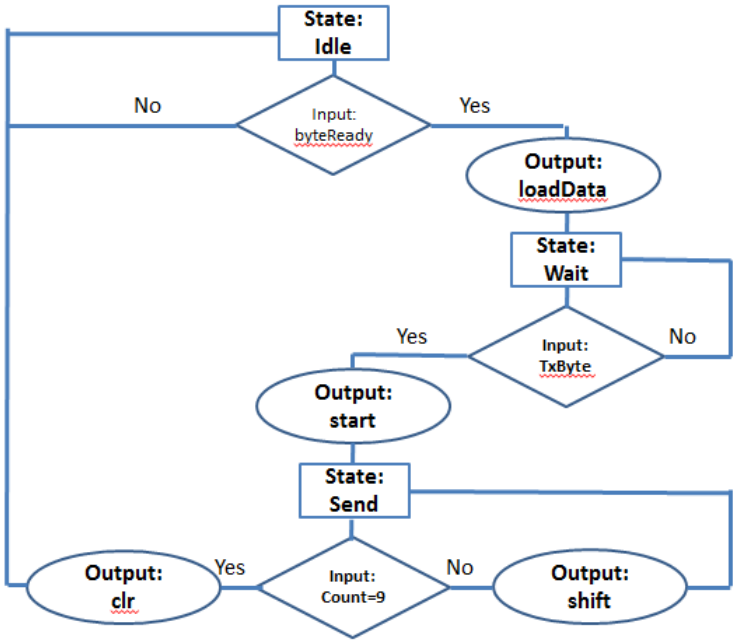
- Input signals:

1) Byte_ready from host computer is to indicate that the data in the bus is valid
2) Load_XMT_datareg from host computer is to tell UART to load data in the bus
3) T_byte from host computer is to ask UART to send data out.
4) Data in the Bus
5) Reset in UART
6) Clock in UART

- Output signals:

1) SeriesOut

- FSM's Algorithm diagram in Transmitter



- Transmitter Verilog Code:

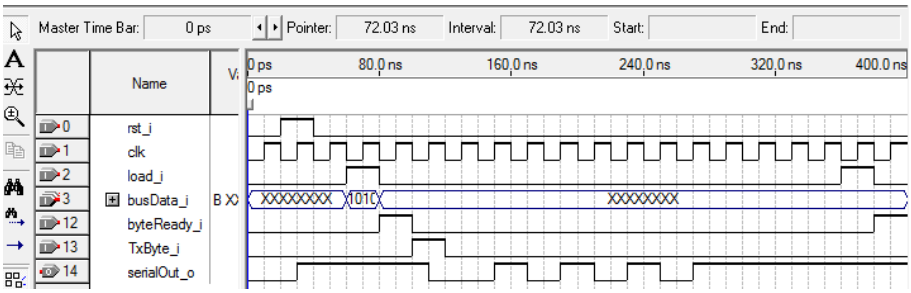
```
module UARTTx(
                clk,
                rst_i,
                byteReady_i,
                load_i,
                TxByte_i,
                busData_i,
                serialOut_o
);
input   clk;
input   rst_i;
input   byteReady_i;
```

```
input   load_i;
input   TxByte_i;
input[7:0] busData_i;
output  serialOut_o;

parameter   pIdle=2'b00;
parameter   pWait=2'b01;
parameter   pSend=2'b10;

reg[1:0]    curSt_r;
reg[1:0]    nxtSt_r;

reg     loadData_r;
reg     start_r;
reg     clr_r;
reg     shift_r;
reg[3:0]    cnt_r;

reg[7:0]    busReg_r;
reg[8:0]    dataReg_r;

//Sequential  Logic
always@(posedge clk)begin
   if(rst_i)begin
      curSt_r <= pIdle;
   end
   else begin
      curSt_r <= nxtSt_r ;
   end
end
// Combo Logic
always@(*)begin
   loadData_r=1'b0;
   start_r=1'b0;
   clr_r=1'b0;
   shift_r=1'b0;
   case(curSt_r)
      pIdle: begin
            if(byteReady_i)begin
               loadData_r=1'b1;
               nxtSt_r=pWait;
            end
            else nxtSt_r=pIdle;
      end
      pWait: begin
         if(TxByte_i)begin
            start_r = 1'b1;
            nxtSt_r= pSend;
         end
         else nxtSt_r=pWait;
      end
      pSend:  begin
         if(cnt_r ==9)begin
            clr_r = 1'b1;
```

```
                nxtSt_r= pIdle;
            end
            else begin
                shift_r = 1'b1;
                nxtSt_r=pSend;
            end
        end
        default: begin
            nxtSt_r= 2'bxx;
        end
    endcase
end
//Implementation block- bit counter
always@(posedge clk)begin
    if(rst_i)begin
        cnt_r<=4'b0000;
    end
    else if(clr_r)begin
        cnt_r<=4'b0000;
    end
    else if(shift_r)begin
        cnt_r<= cnt_r + 1'b1;
    end
end

always@(posedge clk)begin
    if(rst_i)begin
        busReg_r <=8'b0000_0000;
        dataReg_r<=9'b1_1111_1111;
    end
    else if(load_i)begin
        busReg_r <= busData_i;
    end
    else if(loadData_r)begin
        dataReg_r <= {busReg_r, 1'b1};
    end
    else if(start_r)begin
        dataReg_r[0]<=1'b0; //output startbit ==0;
    end
    else if(shift_r)begin
        dataReg_r<={1'b1, dataReg_r[8:1] };
    end
end
            assign  serialOut_o = dataReg_r[0];
endmodule
```

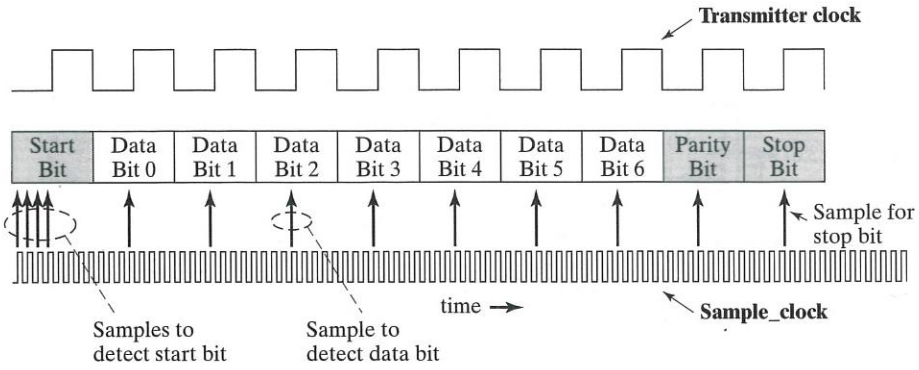
- Verification waveforms:

Simulation: busData_i = 8'b1010_1010
serialOut_o = 0 1 0 1 0 1 0 1



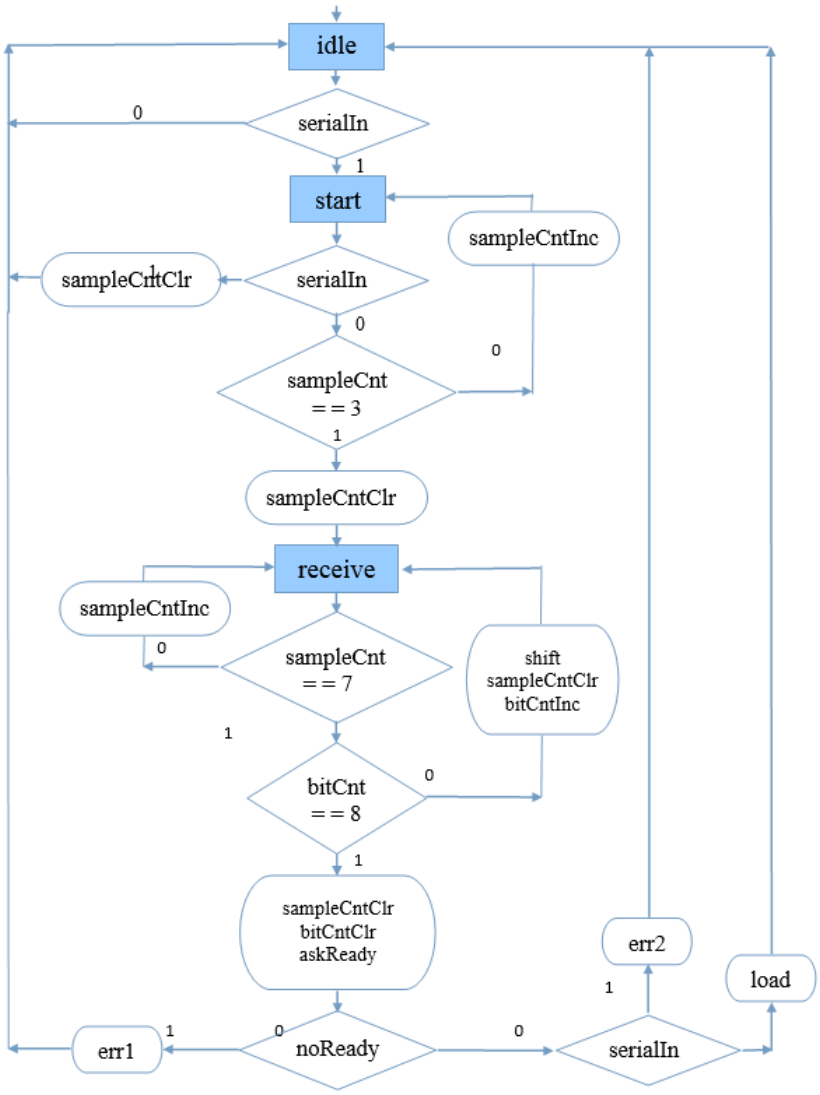
III. UART Receiver Design

Data received by UART don't include clock signal. UART will use higher frequency clock to sample each bit for input data. Usually, sampling frequency in receiver is 8 times frequency of input data.



- Input signals
  1) notReady from host computer is to indicate that host is not ready to receive.
  2) serialIn from outside device
  3) sampleClk is 8 times frequency of transmitter clk
  4) rst
- Output signals
  1) dataOut is to send to host computer
  2) askReady is to ask if host computer is ready to receive or not
  3) err1 is to indicate the host is not ready to receive
  4) err2 is to send to outside device to indicate the missing stop bit

- FSM's algorithm diagram



- Receiver Verilog Code

```
module UARTRx(
                    rst_i,
                    sampleClk,
                    serialIn_i,
                    noReady_i,
                    dataOut_o,
                    askReady_o,
                    err1_o,
                    err2_o
);
input           rst_i;
input           sampleClk;
input           serialIn_i;
input           noReady_i;
output[7:0]     dataOut_o;
output          askReady_o;
```

```
output          err1_o;
output          err2_o;

reg             askReady_o;

parameter       pIdle=2'b00;
parameter       pStart=2'b01;
parameter       pRec=2'b10;

reg[1:0]        curSt_r;
reg[1:0]        nxtSt_r;

reg[2:0]        sampleCnt_r;
reg             sampleCntClr_r;
reg             sampleCntInc_r;
reg[3:0]        bitCnt_r;
reg             bitCntInc_r;
reg             bitCntClr_r;
reg             err1_o;
reg             err2_o;
reg             load_r;
reg             shift_r;
reg[7:0]        shiftReg;
reg[7:0]        dataOut_o;

//FSM-Sequential logic block
always@(posedge sampleClk)begin
        if(rst_i)begin
                curSt_r<=pIdle;
        end
        else begin
                curSt_r<=nxtSt_r;
        end
end

//FSM-Combinational logic block

always@(*)begin
        sampleCntClr_r=1'b0;
        shift_r=1'b0;
        sampleCntInc_r=1'b0;
        bitCntInc_r=1'b0;
        bitCntClr_r=1'b0;
        askReady_o=1'b0;
        err1_o=1'b0;
        load_r=1'b0;
        err2_o=1'b0;
        nxtSt_r=curSt_r;

        case(curSt_r)
                pIdle:begin
                        if(serialIn_i==0)begin
                                nxtSt_r=pStart;
                        end
```

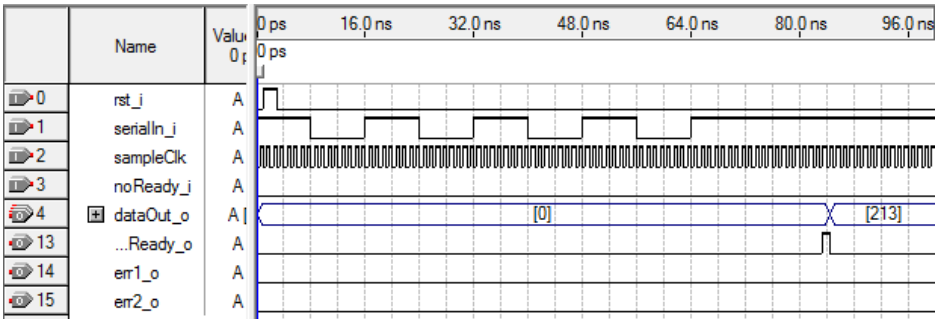
```
                else  nxtSt_r=pIdle;
        end
        pStart: begin
                if(serialIn_i!=0)begin
                        sampleCntClr_r=1'b1;
                        nxtSt_r=pIdle;
                end
                else if(sampleCnt_r==3'b011)begin
                        sampleCntClr_r=1'b1;
                        nxtSt_r=pRec;
                end
                else begin
                        sampleCntInc_r=1'b1;
                        nxtSt_r=pStart;
                end
        end
        pRec:begin
                if(sampleCnt_r==3'b111)begin
                        if(bitCnt_r!=4'b1000)begin
                                shift_r=1'b1;
                                sampleCntClr_r=1'b1;
                                bitCntInc_r=1'b1;
                                nxtSt_r=pRec;
                        end
                        else begin
                                sampleCntClr_r=1'b1;
                                bitCntClr_r=1'b1;
                                askReady_o=1'b1;
                                if(noReady_i==1'b1)begin
                                        err1_o=1'b1;
                                        nxtSt_r=pIdle;
                                end
                                else begin
                                        if(serialIn_i!=0)begin
                                                load_r=1'b1;
                                                nxtSt_r=pIdle;
                                        end
                                        else begin
                                                err2_o=1'b1;
                                                nxtSt_r=pIdle;
                                        end
                                end
                        end
                end
                else begin
                        sampleCntInc_r=1'b1;
                        nxtSt_r=pRec;
                end

        end
        default:begin
                nxtSt_r=2'bxx;
        end
endcase
```

```
    end

    //Implementation
    always@(posedge sampleClk)begin
        if(rst_i)begin
            sampleCnt_r<=3'b000;
        end
        else if(sampleCntClr_r)begin
            sampleCnt_r<=3'b000;
        end
        else if(sampleCntInc_r)begin
            sampleCnt_r<=sampleCnt_r+1'b1;
        end
    //  else begin
    //      sampleCnt_r<=sampleCnt_r;     //Redundant code
    //  end
    end
    always@(posedge sampleClk)begin
        if(rst_i)begin
            bitCnt_r<=4'b0000;
        end
        else if(bitCntClr_r)begin
            bitCnt_r<=4'b0000;
        end
        else if(bitCntInc_r)begin
            bitCnt_r<=bitCnt_r+1'b1;
        end
    //  else begin
    //      bitCnt_r<=bitCnt_r;           //Redundant code
    //  end
    end
    always@(posedge sampleClk)begin
        if(rst_i)begin
            shiftReg<=8'b0000_0000;
            dataOut_o<=8'b0000_0000;
        end
        else if(shift_r)begin
            shiftReg[7:0]={serialIn_i,shiftReg[7:1]};
        end
        else if(load_r)begin
            dataOut_o[7:0]<=shiftReg[7:0];
        end
    end
endmodule
```

- Verification waveforms:

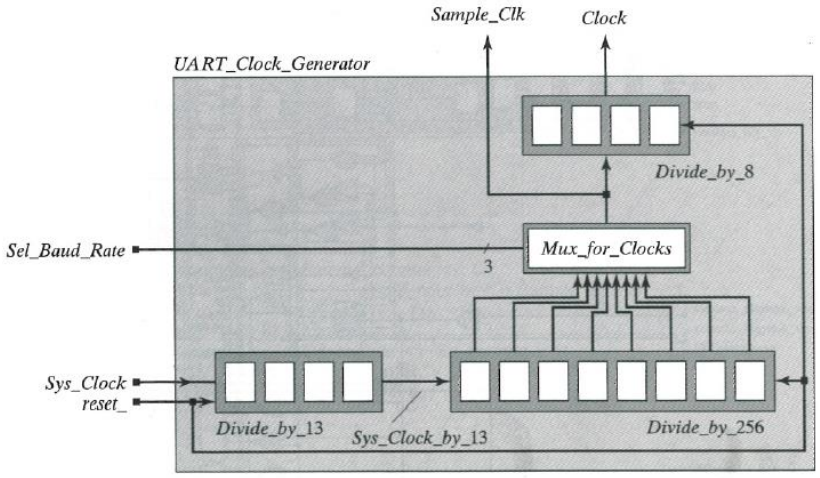
Simulation: serialIn_i = 0(start) 1 0 1 0 _1 0 1 1(parity) 1 (stop)

dataOut_o = 8'b1101_0101 (Decimal # : 213)



IV. Clock Generator

In UART architecture diagram, there is clock generator block to output two clock signals sampleClk and Clk with the different frequencies, namely that sampleClk is 8 times of Clk. Assuming systemClk is 8MHz, design this block based on baud rate in the following table.



brSel	sampleClk	cslk
000	307,696	38462
001	153,838	19231
010	76,920	9615
011	38,464	4808
100	18,232	2404

101	9,616	1202
110	4,808	601
111	2,404	300.5

```
module Div13(
                sysClk,
                rst_i,
                sysClkDiv13_o
);
        input   sysClk;
        input   rst_i;
        output  sysClkDiv13_o;

        reg[3:0] cnt_r;

        assign  sysClkDiv13_o = cnt_r[3];

        // wire Clk1 = (cnt_r == 12);
        // wire Clk2 = (cnt_r > 6);

        always@(posedge sysClk)begin
                if(rst_i)begin
                        cnt_r <=4'b0000;
                end
                else if(cnt_r == 4'd12)begin
                        cnt_r <=4'd0;
                end
                else begin
                        cnt_r <= cnt_r + 1;
                end
        end
 endmodule

`include        “Div13.v”
module          clkGen(
                                sysClk,
                                rst_i,
                                brSel,          //Baud Rate select

                                sampleClk_o,
                                clk_o
);
        input   sysClk;
        input   rst_i;
        input   brSel;          //Baud Rate select

        output  sampleClk_o;
        output  clk_o;
```

```
        wire        sysClkDiv13_w;
        reg[7:0]    Div256_r;
        reg[2:0]    Div8_r;

        Div13       uDiv13(
                                .sysClk         (sysClk),
                                .rst_i          (rst_i),
                                .sysClkDiv13_o  (sysClkDiv13_w)
        );

        always@(posedge sysClkDiv13_w)begin
                if(rst_i)       Div256_r <= 0;
                else            Div256_r <= Div256_r + 1;
        end
        always@(*)begin
                case(brSel)
                        3'b000:     sampleClk_o = Div256_r[0];
                        3'b001:     sampleClk_o = Div256_r[1];
                        3'b010:     sampleClk_o = Div256_r[2];
                        3'b011:     sampleClk_o = Div256_r[3];
                        3'b100:     sampleClk_o = Div256_r[4];
                        3'b101:     sampleClk_o = Div256_r[5];
                        3'b110:     sampleClk_o = Div256_r[6];
                        3'b111:     sampleClk_o = Div256_r[7];
                        default:    sampleClk_o = 0;
                endcase
        end
        always@(posedge sampleClk_o)begin
                if(rst_i)       Div8_r <= 0;
                else            Div8_r <= Div8_r + 1;
        end
        assign      clk_o = Div8_r[2];
endmodule
```

V. Top Level Design

```
`include    "UARTTx.v"
`include    "UARTRx.v"

module Top(
                TxClk,              // From other block in the same chip
                rst_i,              // From other block in the same chip
                TxByteReady_i,      // From other block in the same chip
                TxLoad_i,           // From other block in the same chip
                TxByte_i,           // From other block in the same chip
                TxData_i,           // From other block in the same chip
                TxDout_o,           // To outside of the chip

                        RxClk,          // From other block in the same chip
                        RxDin_i,        // From outside of the chip
                        RxNoReady_i,    // From other block in the same chip
                        RxDout_o,       // To other block in the same chip
                        RxAskReady_o,   // To other block in the same chip
```

```
                        RxErr1_o,               // To other block in the same chip
                        RxErr2_o                // To other block in the same chip
);

        input           Txclk;
input           rst_i;
input           TxByteReady_i;
input           TxLoad_i;
input           TxByte_i;
input           TxData_i;
output[7:0]     TxDout_o;

        input           RxClk;
        input           RxDin_i;
        input           RxNoReady_i;
        output[7:0]     RxDout_o;
        output          RxAskReady_o;
        output          RxErr1_o;
        output          RxErr2_o;

        UARTTx          uUARTTx(
                                .clk            (Txclk),
                        .rst_i          (rst_i),
                        .byteReady_i    (TxByteReady_i),
                        .load_i         (TxLoad_i),
                        .TxByte_i       (TxByte_i),
                        .busData_i      (TxData_i),
                        .serialOut_o    (TxDout_o)
        );
        UARTRx          uUARTRx(
                                .rst_i          (rst_i),
                                .sampleClk      (RxClk),
                                .serialIn_i     (RxDin_i),
                                .noReady_i      (RxNoReady_i),
                                .dataOut_o      (RxDout_o),
                                .askReady_o     (RxAskReady_o),
                                .err1_o         (RxErr1_o),
                                .err2_o         (RxErr2_o)
        );
endmodule
```