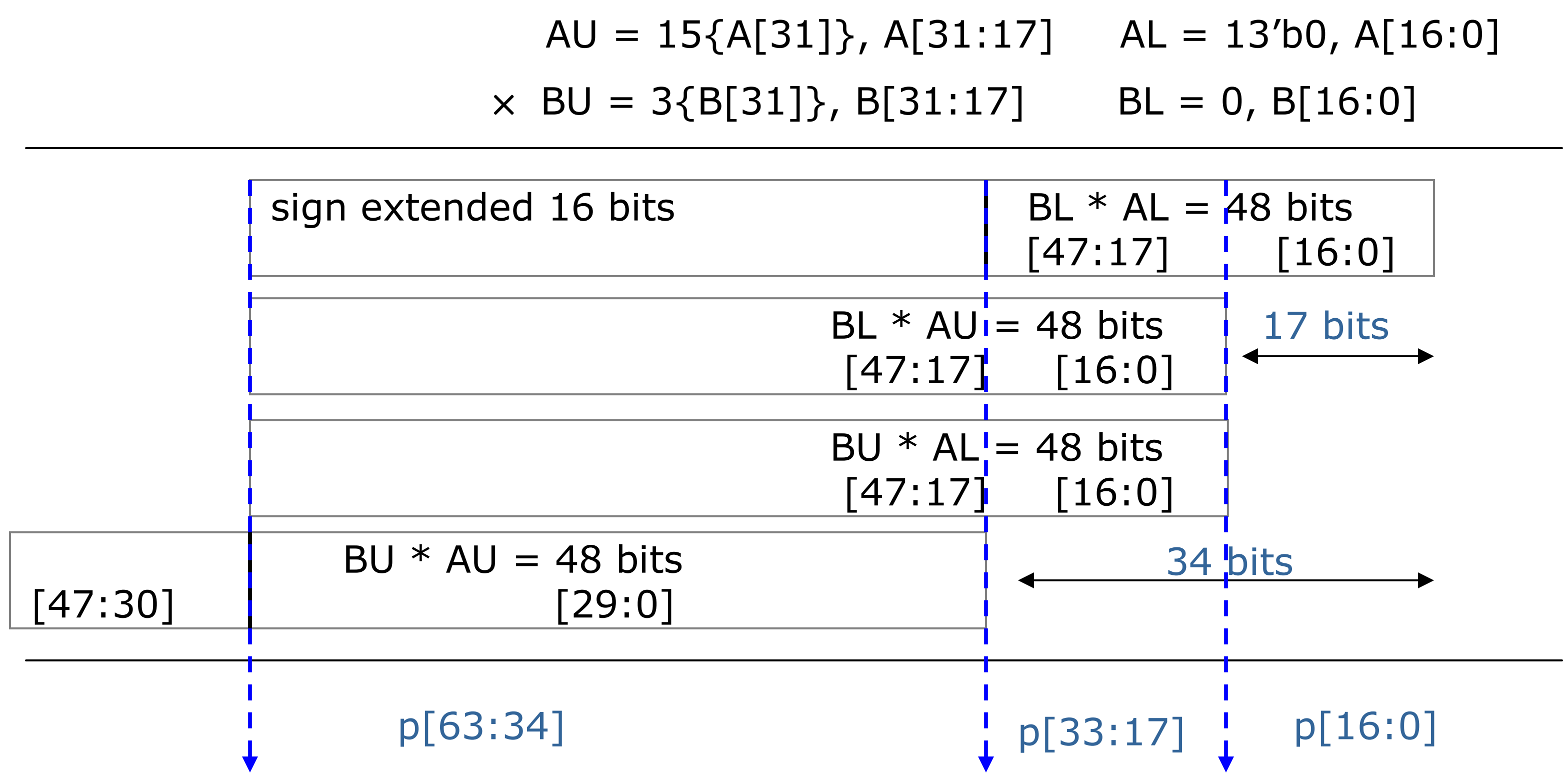
Multiplier, Register File, Memory and UART Implementation

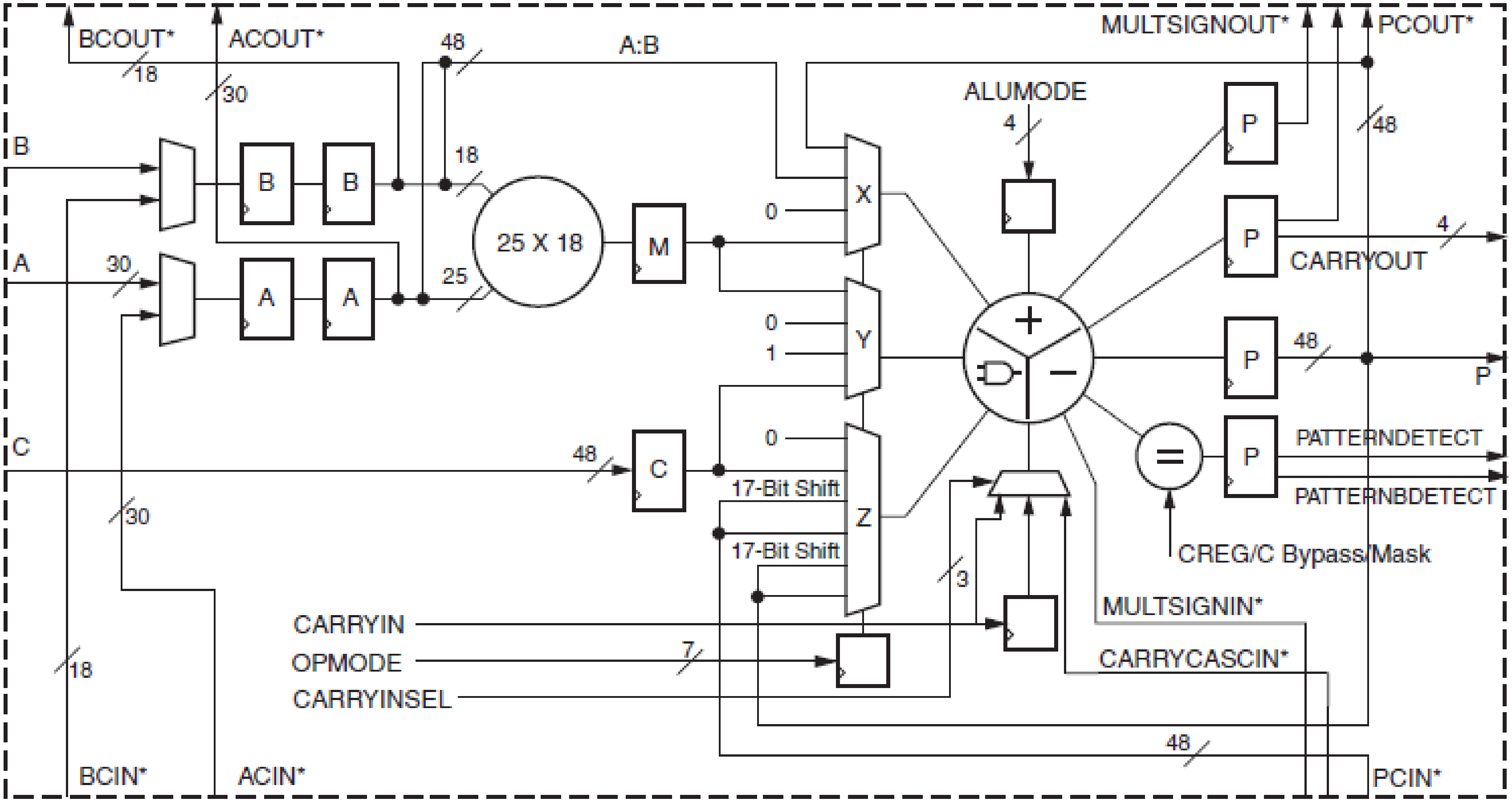
L. Liu
Department of Computer Science, ETH Zürich
Fall semester, 2012

Forming Larger Multipliers with DSP Slices

- Forming a 32×32 two's complement multiplication from a 25×18 DSP slice



The notation "0,B[16:0]" denotes B has a leading zero followed by 17 bits, forming a positive two's complement number.



	cyc0	cyc1	cyc2	cyc3	cyc4
opmode	0+M	P_{rs17}+M	P+M	P_{rs17}+M	--
M	A_L*B_L	A_U*B_L	A_L*B_U	A_U*B_U	--
P	0	A_L*B_L	$(A_L*B_L)_{rs17} + A_U*B_L$	$(A_L*B_L)_{rs17} + A_U*B_L + A_L*B_U$	$((A_L*B_L)_{rs17} + A_U*B_L + A_L*B_U)_{rs17} + A_U*B_U$

Register File

- Implemented with dual-port 16×1 bit distributed RAMs.
- Use 32 RAM16X1D_1 primitives.

Verilog code for TRM register file

```
genvar i;
generate    //dual port register file
        for (i = 0; i < 32; i = i+1)
        begin: rf32
        RAM16X1D # (.INIT(16'h0000))
        rfa(
        .DPO(AA[i]), // data out
        .SPO(B[i]),
        .A0(dst[0]),   // R/W address, controls D and SPO
        .A1(dst[1]),
        .A2(dst[2]),
        .A3(1'b0),
        .D(regmux[i]),  // data in
        .DPRA0(irs[0]), // read-only adr, controls DPO
        .DPRA1(irs[1]),
        .DPRA2(irs[2]),
        .DPRA3(1'b0),
        .WCLK(~clk),
        .WE(regwr));
        end
endgenerate
```

TRM Instruction Memory

- n BRAMs in Virtex-5 FPGA are configured as a nk*36 bit ROM.

Verilog code in IM Module

```
…
genvar i;
generate
  for (i = 0; i< BN; i = i+1)
  begin: ram
    RAMB36 #(...)
    RAMB36_inst (...
     .DOB(rdb[i][31:0]), // 32-bit B port data output
     .DOPB(rdb[i][35:32]), // Port B 4-bit Parity Output
     ...
     .WEA(4'b0), // Port A Write Enable Input
     .WEB(4'b0) // Port B Write Enable Input)
     ... )
  end
endgenerate
```

TRM Data Memory

- n BRAMs in Virtex-5 FPGA are configured as a nk*32 bit RAM.

Verilog code in DM Module

```
…
genvar i;
generate
  for (i = 0; i< BN; i = i+1)
  begin: RAMB_insts //Port B as read port, port A as write port
    RAMB36 #(...)
    RAMB36_inst (...
     .DOPB(), // Port B 4-bit Parity Output
     .DOB(rdb[i]), //32-bit B port data output
     ...
.WEA({wea[i], wea[i], wea[i], wea[i]}), // Port A Write Enable Input
      .WEB(4'b0) // Port B Write Enable Input)
     ... )
  end
endgenerate
```

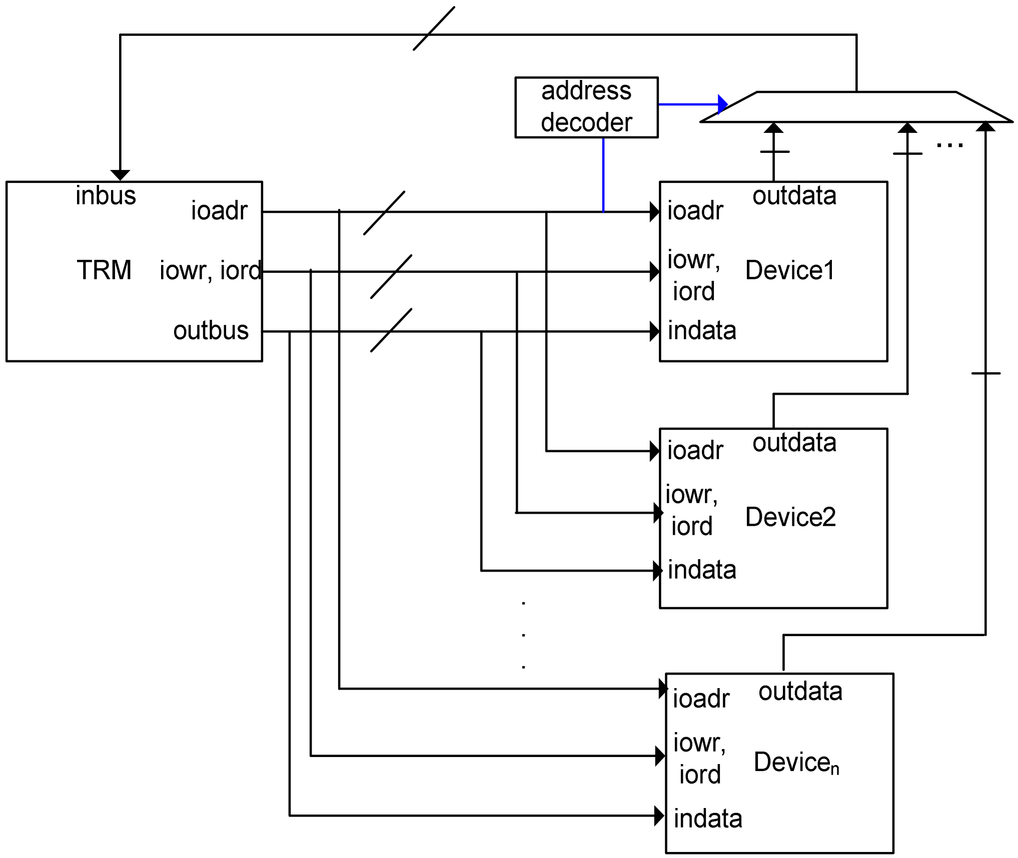
Memory-Mapped Input/Output (I/O)

- Processor accesses I/O devices (like keyboards, monitors, printers) just like it accesses memory
- Each I/O device is assigned one or more address
- When that address is detected, data is read from or written to I/O device instead of memory
- A portion of the address space is dedicated to I/O devices (for example, for a TRM with 2K data memory, address 0xFFFFFFC0 ~ 0xFFFFFFFF are reserved for I/Os.)

Memory-Mapped I/O Hardware

- **Address Decoder:**
  - Looks at address to determine which device/memory communicates with the processor
- **I/O Data Buses:**
  - Hold values read from / written to the I/O devices
- **Multiplexer:**
  - Selects between memory outputs and the data sent to the processor from I/O devices

The I/O Interface of MRC



Verilog code MRC interface

```
module TRM (
   input clk, rst,
   input [31:0] inData,
   output [31:0] outData,
   output [5:0] ioadr,
   output iowr, iord
);
```

I/O Interface Example: Accessing Switches, Leds

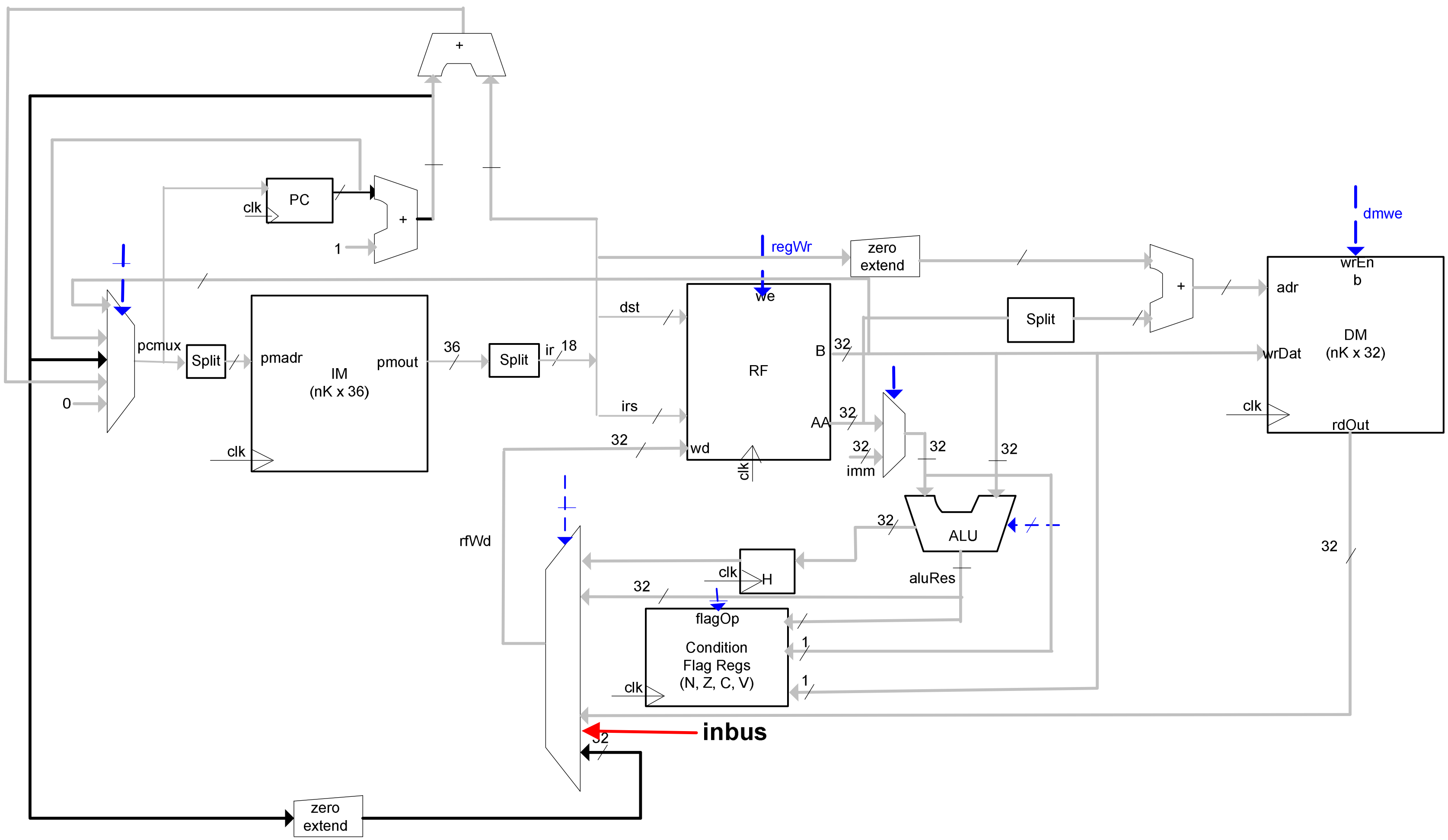
allocate address:

0xFFFFFFC7 to switches and leds

Verilog code in Top Module

```
module Top (
  input CLKBN, CLKBP, rstIn,
  input [7:0] swi,
  output [7:0] leds
);
  ...
TRM trmx(.clk(clk), ... , .ioadr(ioadr), .iowr(iowr), .inbus(inbus),
.outbus(outbus));

LED ledx(.clk(clk), .nReset(rst), .ioadr(ioadr), .iowr(iowr),
.inData(outbus[7:0]), .outData(leds));

Switch switchx(.clk(clk), .nReset(rst), .inData(swi), .outData(swiValue));

assign inbus = ((ioadr == 7) & iord)? {24'b0, swiValue}: 0;
...
end module
```

Selecting Between Memory Outputs and I/O Data



Verilog code in TRM module

```
...
assign regmux =
      (BL | BLR) ? {{{32-PAW}{1'b0}}, nxpc} :
      (LDR & ~ioenb) ? dmout :
      (LDR & ioenb)? inbus: //from IO
      (MUL) ? mulRes[31:0] :
      (ROR) ? s3 :
      (LDH) ? H :
      aluRes;

assign ioenb = &(dmadr[DAW:6]);
assign ioadr = dmadr[5:0];
assign iord = LDR & ~IR[10] & ioenb;
assign iowr = ST & ~IR[10] & ioenb;
assign outbus = B;
...
```

Memory-Mapped I/O Software

- Suppose address 0xFFFFFFC7 is assigned to 8 switches, and 8 Leds.
  - **Read the value from 8 switches and place it on Leds**

Memory-Mapped I/O Code

- **Read the value from switches (0xFFFFFFC7) and place it on leds (0xFFFFFFC7).**

Assembly code:

```
MOV R0, 0
SUB R0, 57
LD R0, [R0]
ST R0, [0]
LD R0, [0]
MOV R1, 0
SUB R1, 57
ST R0, [R1]
```

Oberon code:

….

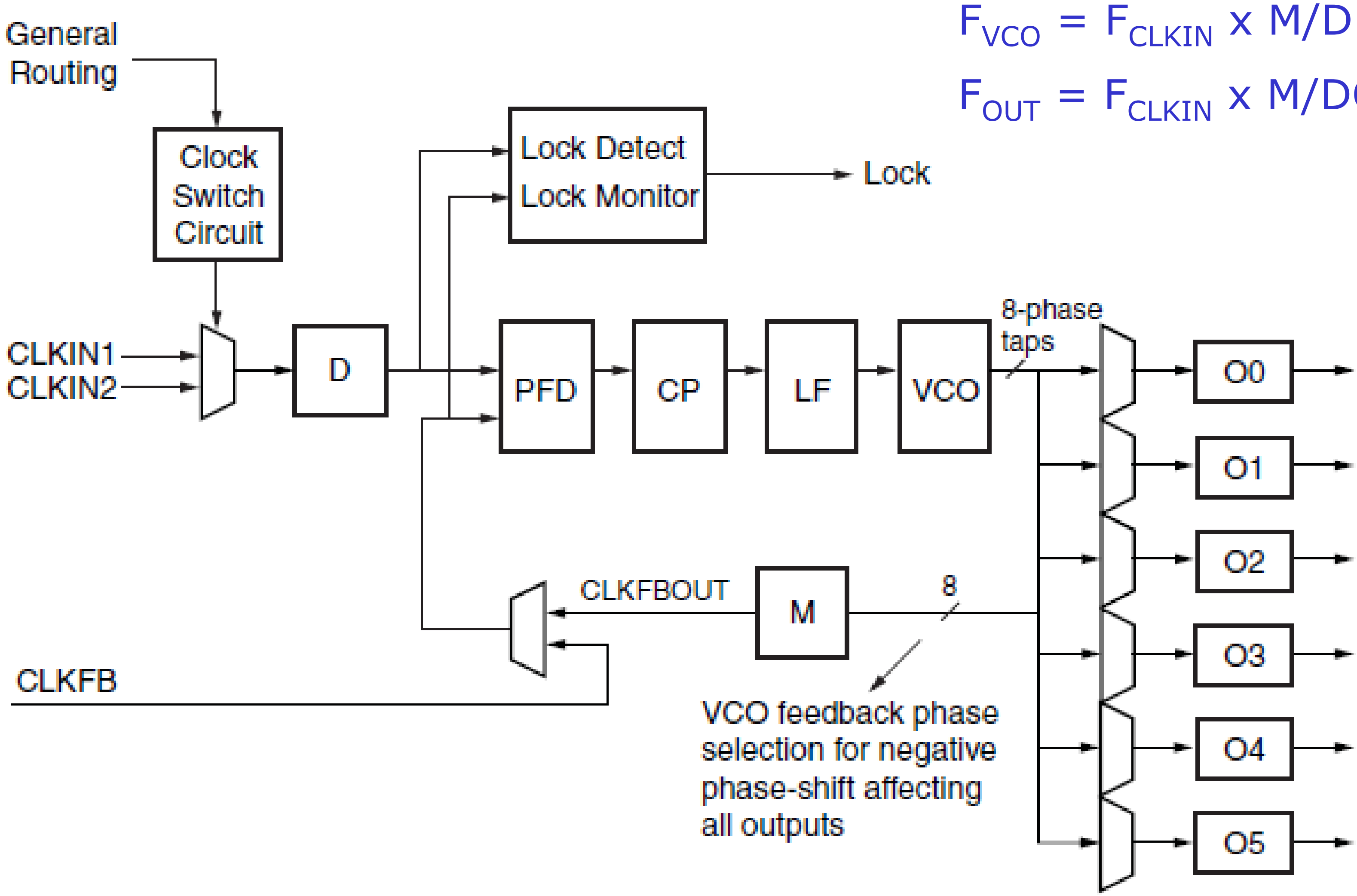
Clock Generation

- Phase Locked Loop Clock Circuit (PLL)
  - Serve as a frequency synthesizer for a wide range of frequencies, and to serve as a jitter filter for either external or internal clocks
  - This component allows the input clock to be:
    - phase shifted
    - multiplied
    - Divided

PLL Block Diagram

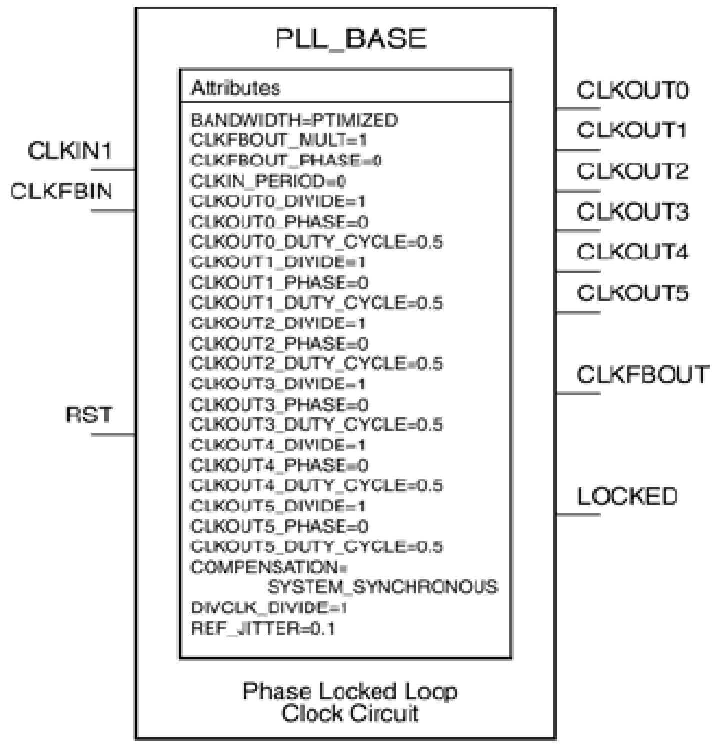
$F_{VCO} = F_{CLKIN} \times M/D$

$F_{OUT} = F_{CLKIN} \times M/DO$



Source:
Xilinx, Virtex-5UserGuide.

PLL Primitives



Verilog code in Top Module

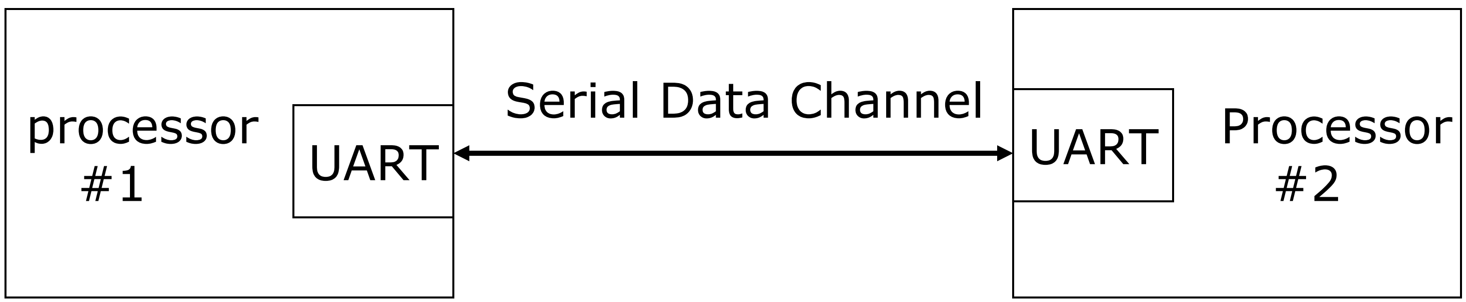
```
PLL_BASE #( ...)
clkBPLL (
...
.CLKOUT2(CLKx),
.LOCKED(pllLock),
.RST(1‘b0));

 BUFG bufc(.I(CLKx), .O(clk));
```

Design and Implementation of UART Controller

Introduction to UART

- UART stands for "Universal Asynchronous Receiver/Transmitter".
  - used to convert serial data to parallel data, and parallel data to serial data.
- Popular serial communication device in computers to interfacing low speed peripheral devices, such as the keyboard, the mouse, modems etc.
- Asynchronous communication, operate on independent clocks.
- Common Speed rates: 9600Bauds, 115200 Bauds.

UART Interface

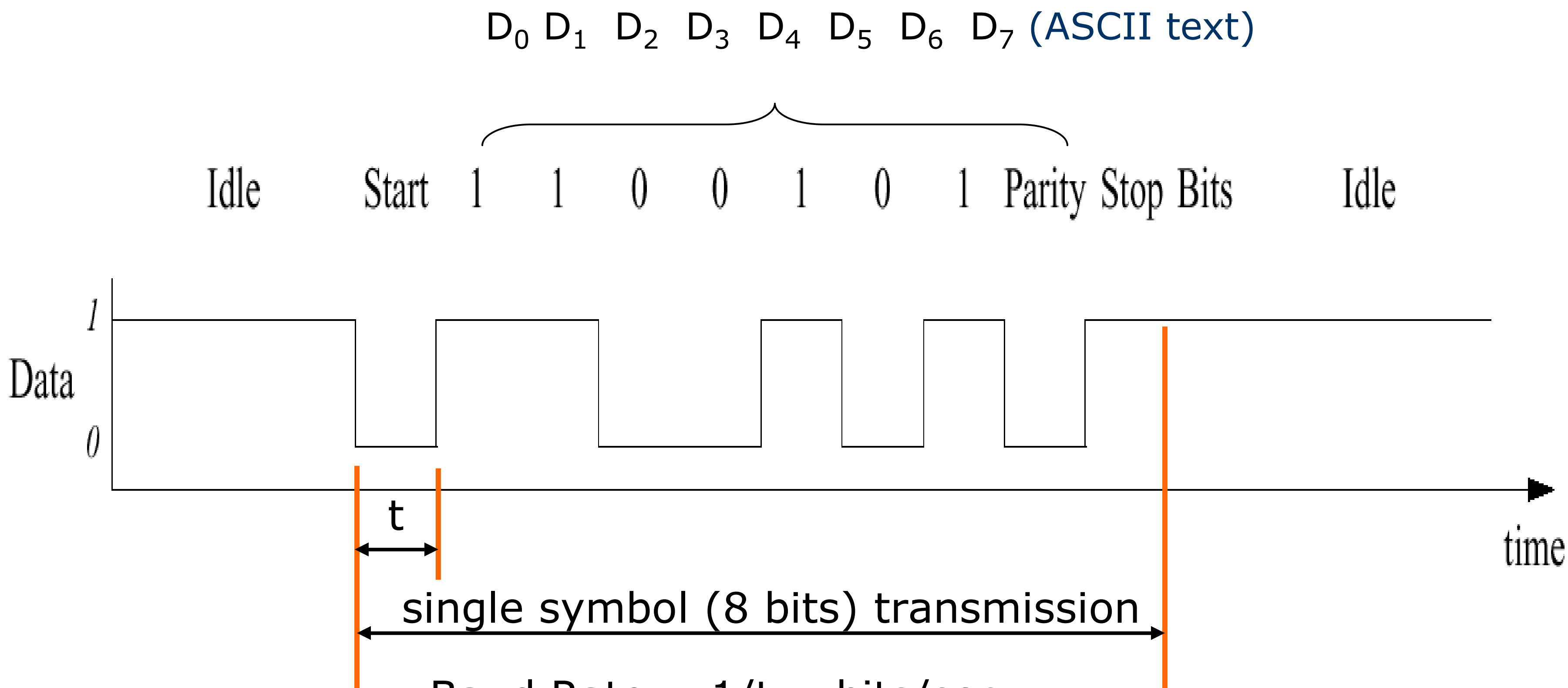


In most computer systems, the UART is connected to circuitry that generates signals that comply with the EIA (Electronics Industries Association) RS232-C specification.

RS232 (Recommended Standard 232)

- Standard for communication of ASCII-coded character data between devices such as data computers and modems
  - Low speed and cheap
- Standard definition:
  - The voltages used to represent 0 and 1 (Electrical)
  - The rate at which data is sent.
  - The format of the data sent.
  - The connectors to be used (physical and mechanical)
  - Extra control signals that may be used.
- Typical data rates ((baud rates) are: 75, 300, 1200, 2400, 9600, 19200 and 115,000 bits/sec

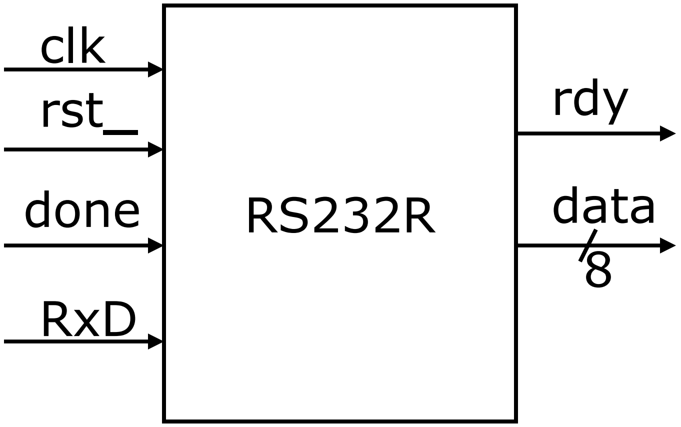
RS232 Sent Data (Bit Stream) Format



Baud Rate = 1/t = bits/sec

RS232 Receiver

- Detect transmission, receive the serial bit stream of data from RxD port, remove the start-bit and transfer the data in a parallel format to the host (TRM in our case) data bus.



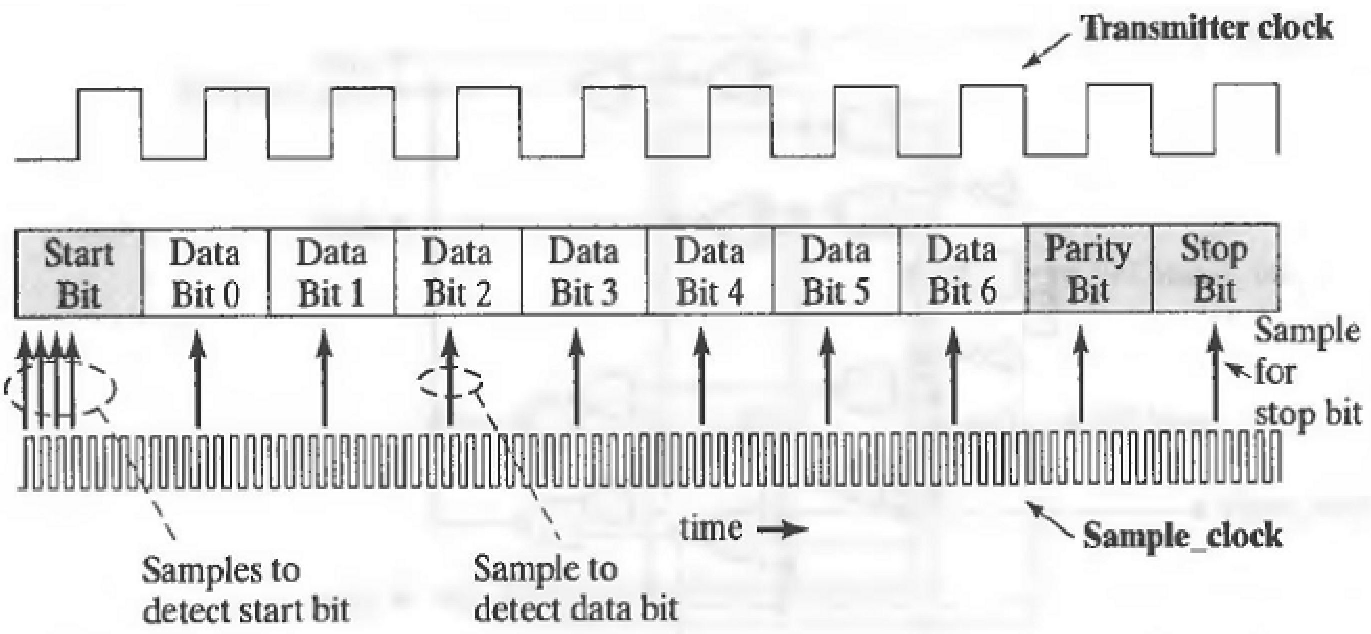
Verilog code in RS232R module

```
module RS232R
#(parameter FreqDiv = 1)
(
input clk, rst,
input done, // "byte has been read"
input RxD,
output rdy,
output [7:0] data);
```

RS232 Receiver Synchronization Issue

- Receiver speed: Host (TRM) runs at (100/FreqDiv) MHz
- Transmitter speed: The baud rate of the RS232 port on Spartan3 board is 115200 bits/sec, which means the clock speed for the transmitter is 115.2KHz.
- Transmitter and receiver each has its own clock running at different frequency.
- How to synchronize the two clocks so that the receiver can sample the coming data in the middle of a bit time.

Sampling Scheme



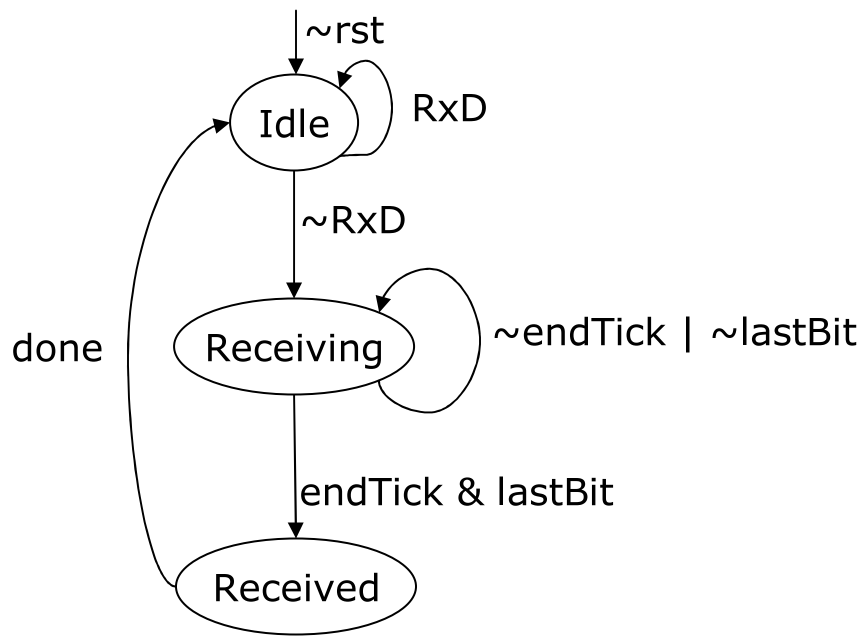
Source

Michael D. Ciletti “Advanced Digital Design with the Verilog HDL”.

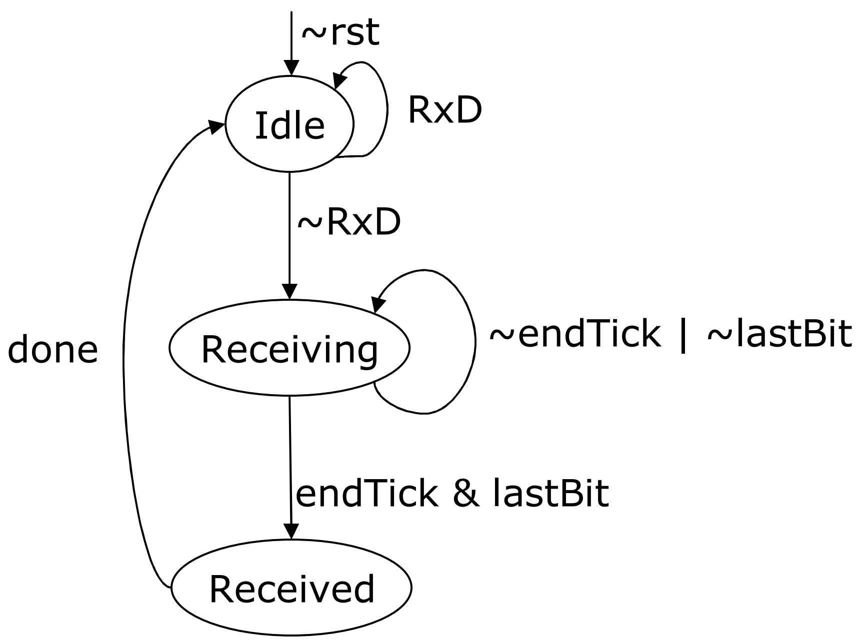
Receiving State Machine

Verilog code in RS232R module

```
always @(posedge clk)
  if (~rst)
    State <= Idle;
  else
       case (State)
         Idle: if (~RxD)
                   State <= Receiving;
     Receiving: if (endTick & lastBit)
            State <= Received;
         Received : if (done)
                State <= Idle;

       endcase
```



Sample Counter - Tick

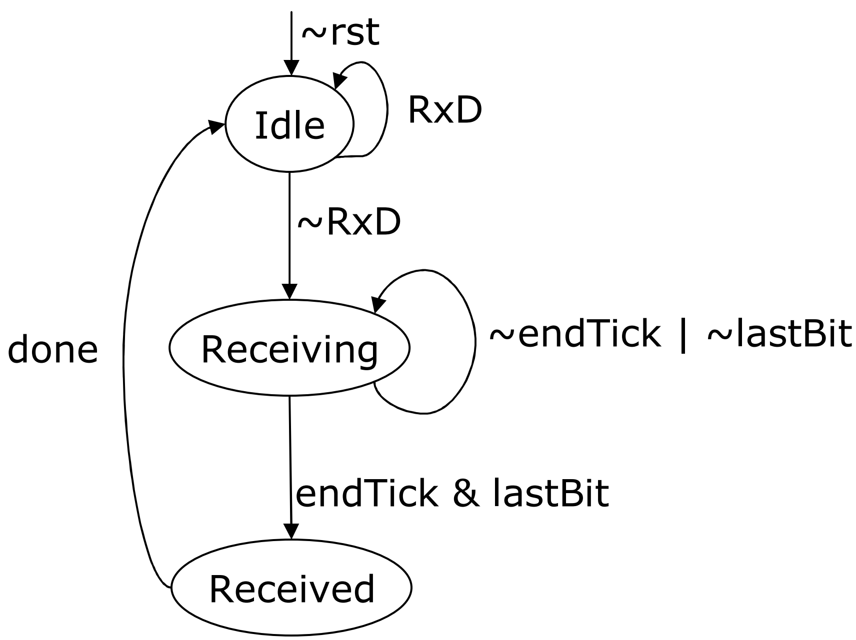


Verilog code in RS232R module

```
always @(posedge clk)
  if (~rst)
    Tick <= 0;
  else if ((State == Receiving)&
~endTick)
    Tick <= Tick + 1;
  else
    Tick <= 0;

assign endTick =
        (Tick == (868/FreqDiv));
assign midTick =
        (Tick == (434/FreqDiv));
```

100000 /115.2 = 868

Bit Counter - BitCnd



Verilog code in RS232R module

```
always @(posedge clk)
  if (~rst)
    BitCnt <= 0;
  else if (lastBit & endTick)
    BitCnt <= 0;
  else if (endTick)
    BitCnt <= BitCnt+1;
```

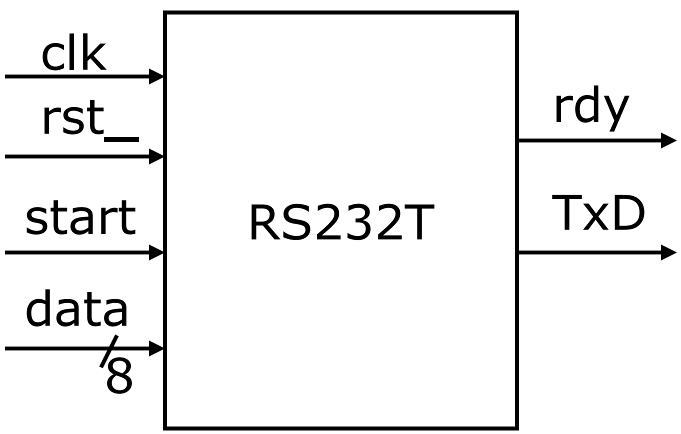
Shift Register - Shreg

Verilog code in RS232R module

```
always @(posedge clk)
  if (~rst)
    Shreg <= 0;
  else if ((State == Receiving) & midTick)
  begin
    Shreg[6:0] <= Shreg[7:1];
Shreg[7] <= RxD;
  end
assign data = Shreg;
assign rdy = (State == Received);
```

RS232 Transmitter

- Accepts a byte of data from the host (TRM) data bus and transmits it as serial data on the TxD port. The baud rate for the transmission is the same as the baud rate for the RS232 receiver.



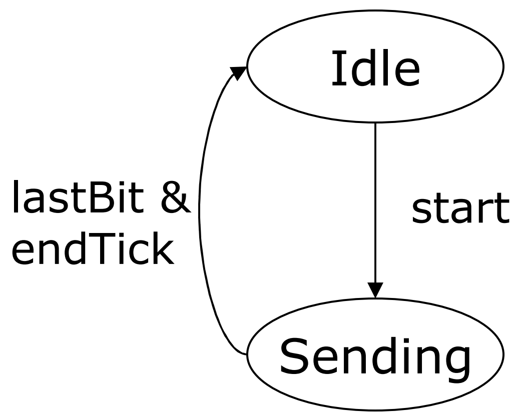
Verilog code in RS232T module

```
module RS232T
#(parameter FreqDiv = 1)
(
input clk, rst,
input start, // request for sending
input [7:0] data,
output rdy,
output TxD);
```

Transmitter Operation

- First the transmitter detects whether the host (TRM) has set the request (“start” input) for transmission.
- If it is the case, it loads the data onto the transmit register.
- Synchronous to the baud clock, the transmitter sets the start bit on the TxD port to initiate the start of a bit stream frame and then bit by bit the symbol data.
- It finally completes the transmission by sending the stop bit.

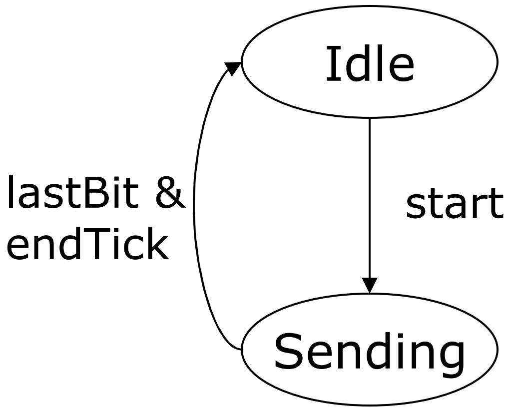
Transmitter State Machine



Verilog code in RS232T module

```
always @(posedge clk)
  if (~rst)
    State <= Idle;
  else
    case (State)
          Idle: if (start)
                  State <= Sending;
       Sending: if (lastBit & endTick)
                  State <= Idle;
    endcase
```

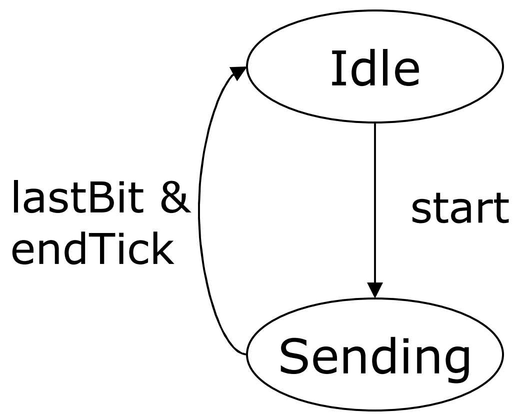
Sample Counter - Tick



Verilog code in RS232T module

```
always @(posedge clk)
  if (~rst)
    Tick <= 0;
  else if ((State == Sending) & ~endTick)
    Tick <= Tick + 1;
  else
    Tick <= 0;

assign endTick =
           (Tick == (868/FreqDiv));
```

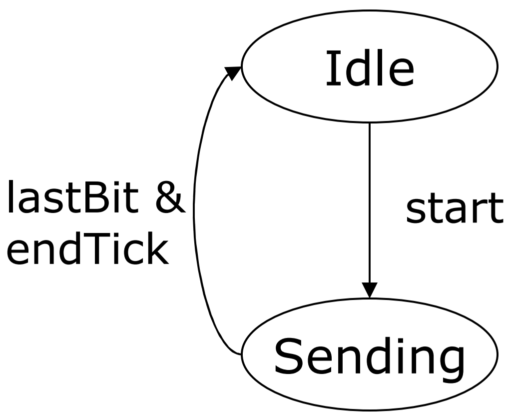
Bit Counter - BitCnt



Verilog code in RS232T module

```
always @(posedge clk)
  if (~rst)
    BitCnt <= 0;
  else if (lastBit & endTick)
    BitCnt <= 0;
  else if (endTick)
    BitCnt <= BitCnt + 1;

assign lastBit = (BitCnt == 9);
```

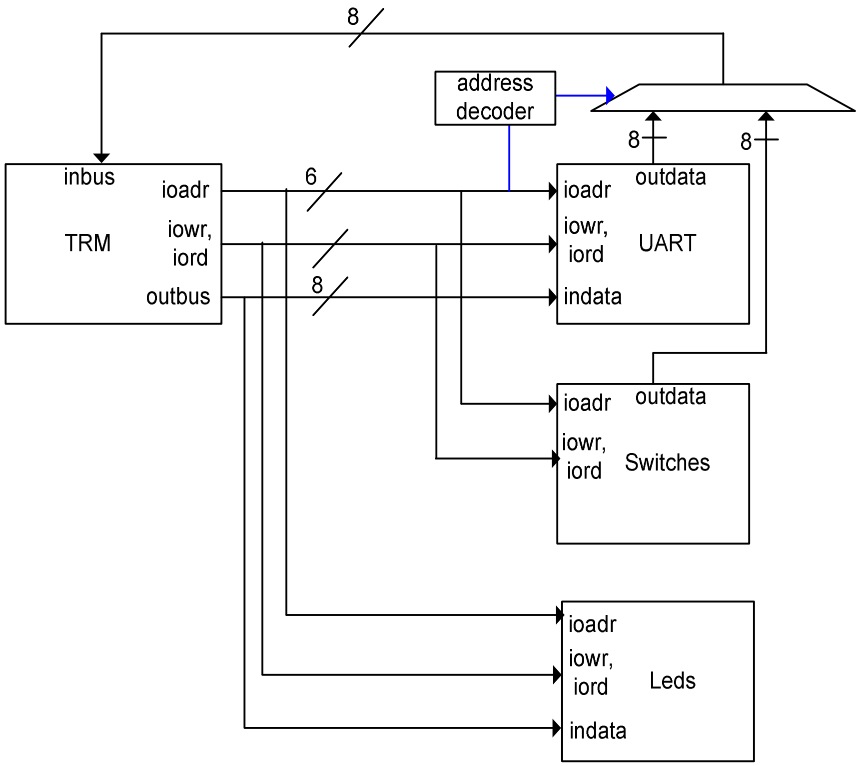
Shift Register - Shreg

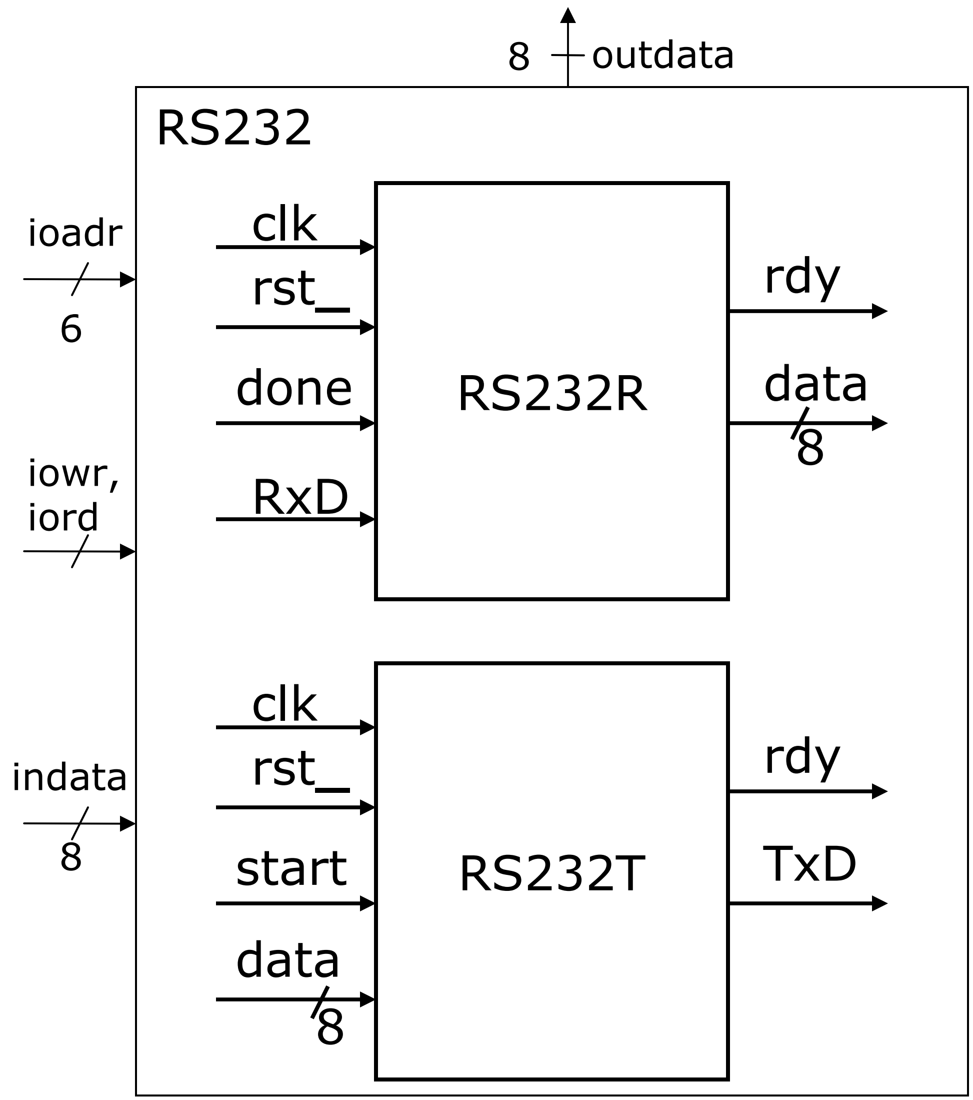


Verilog code in RS232T module

```
always @(posedge clk)
  if (~rst)
    Shreg <= 0;
  else if ((State == Idle) & start)
    begin
      Shreg[0] <= 1'b1;
      Shreg[8:1] <= ~data;
    end
  else if ((State == Sending) & endTick)
    begin
      Shreg[8:0] <= Shreg[9:1];
    end

assign TxD = ~Shreg[0];
```

Interfacing RS232 to TRM



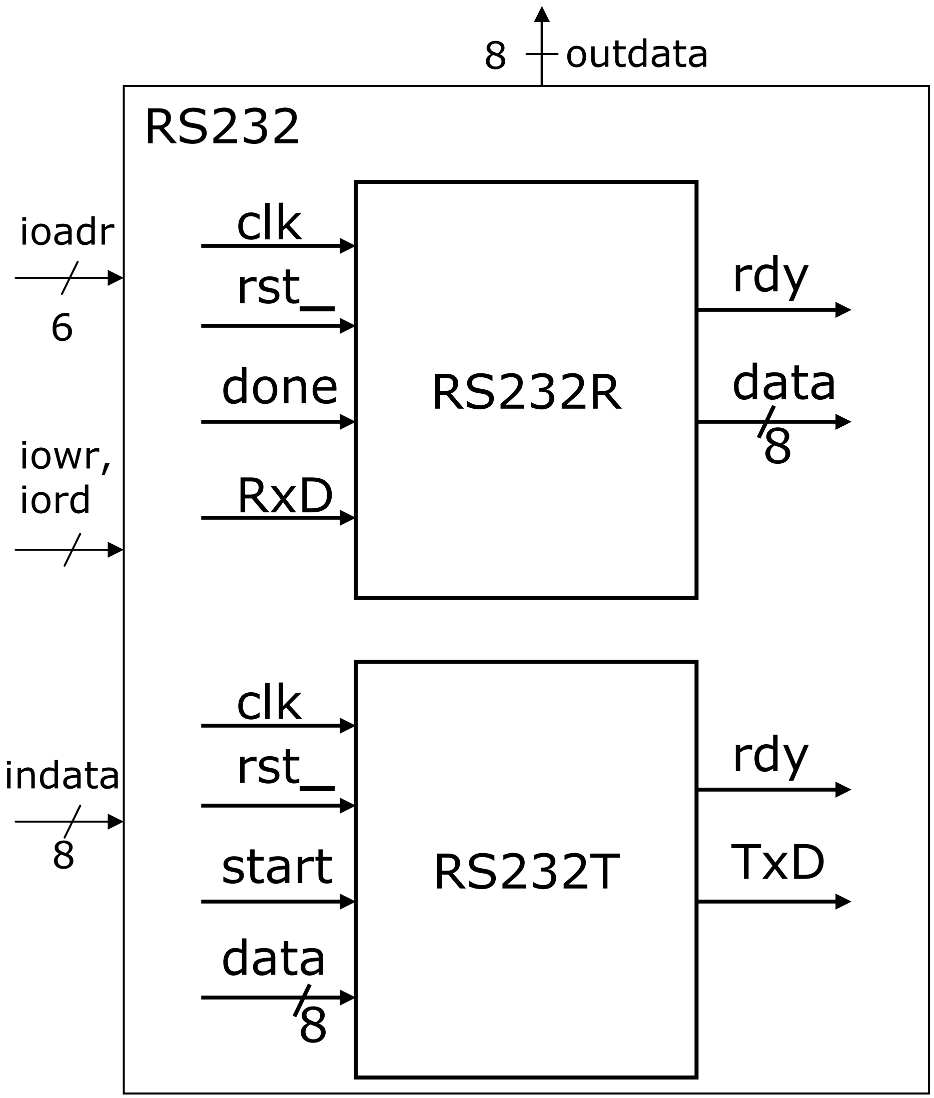


Communication between RS232 and TRM

- TRM must be able to give commands to RS232 to request for sending a byte.
- RS232 must be able to notify TRM when a byte is received.
- Data must be transferred between TRM and RS232.

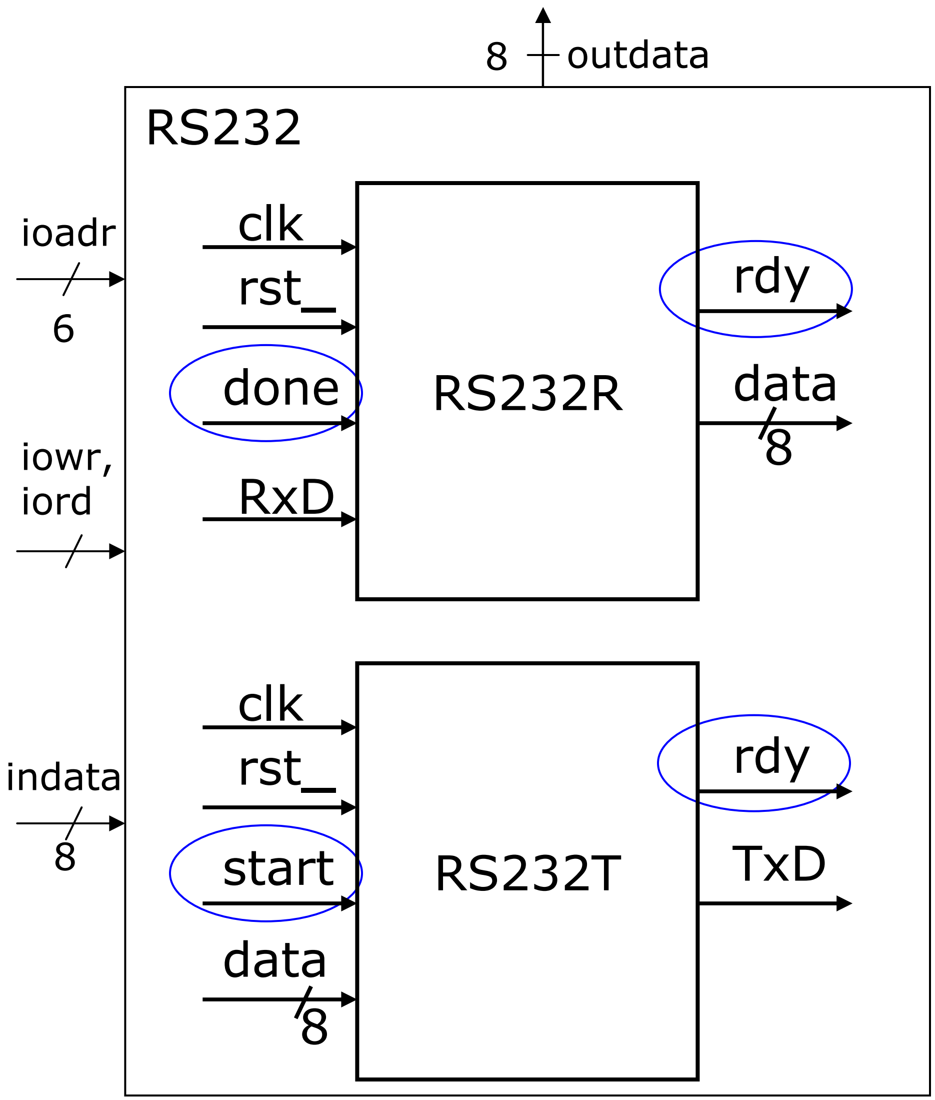
- Assign two I/O addresses to RS232
  - Address 0xFFFFFFC5: used for transferring command and status information.
  - Address 0xFFFFFFC4: used for transferring data.

RS232 Module



Verilog code in RS232 module

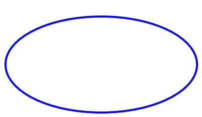
```
module RS232
#(parameter FreqDiv = 1)
(
input clk, rst,
input [3:0] ioadr,
input iowr, iord,
input RxD,
input [7:0] inData,
output [7:0] outData,
output TxD
    );
```



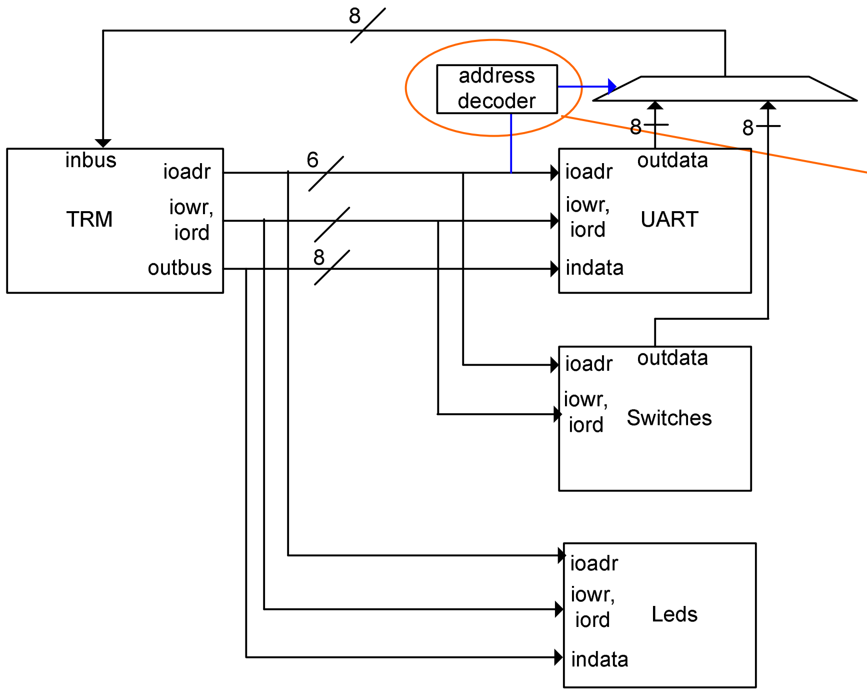
Verilog code in RS232 module

...

```
assign outData =
            (ioadr == 4) ? dataRx :
            {6'b0, rdyTx, rdyRx};
```



command and status signals to / from UART.



Verilog code in Top module

```
assign inData =
((ioadr == 4) | (ioadr == 5)) ?
{24'b0, rs232Data}:
(ioadr == 7)? swiDat: 0;
```

Access I/O in Software

- Polling process: periodically checking bits to see if I/O is ready for the next I/O operations.
- Interrupt-driven I/O: an I/O scheme that employs interrupts to indicate to the processor that an I/O device needs attention.
- If the I/O is ready for the next operation, then starts transferring data.

Example Code

```
CONST RSadr= 0FFFFFC4H;

                                     polling

PROCEDURE Receive*(VAR x: CHAR);
BEGIN
    REPEAT  UNTIL SYSTEM.BIT(RSadr+1, 0);
    SYSTEM.GET(RSadr, x);
END Receive;

PROCEDURE Send*(x: CHAR);
BEGIN
    REPEAT UNTIL SYSTEM.BIT(RSadr+1, 1);
    SYSTEM.PUT(RSadr, x)
END Send;
```

Hardware / Software Interface in Operating System

- The Operating system
  - Provides drivers - software routines that handle low-level device operations, for example, read/write a byte/word from RS232.
  - Guarantees that a user's program accesses only the portions of I/O devices to which the user has rights.
  - Handles the interrupts generated by I/O devices.
  - Manages the accesses to the shared I/O devices.