Finite State Machine

Alex Yang
Engineering School, NPU

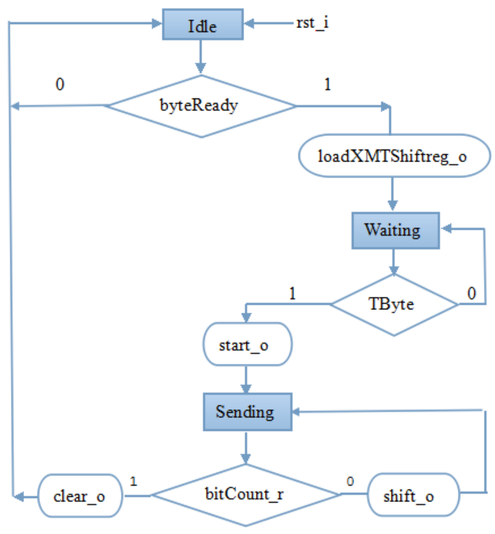
1. Bit Steam “101” Pattern Detector

```
module det101(
                        clk,
                        rst,
                        in_i,
                        out_o
);
    input               clk;
    input               rst;
    input               in_i;
    output              out_o;

    reg                 out_o;

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

    parameter       pIdle   = 2’b00;
    parameter       pSt1    = 2’b01;
    parameter       pSt10   = 2’b10;
    parameter       pSt101  = 2’b11;

    always@(posedge clk)begin
        if(rst)     curSt_r   <= #1 pIdle;
        else        curSt_r   <= #1 nxtSt_r;
    end
    always@(*)begin
        out_o = 1’b0;
        case(curSt_r )
            pIdle: begin
                    if(rst)             nxtSt_r = pIdle;
                    else if(in_i)       nxtSt_r = pSt1;
                    else                nxtSt_r = pIdle;
            end
            pSt1: begin
                    if(in_i)            nxtSt_r = pSt1;
                    else                nxtSt_r = pSt10;
            end
            pSt10: begin
                    if(in_i)begin
                        out_o = 1’b1;
                        nxtSt_r = pSt101;
                    end
                    else nxtSt_r = pIdle;
            end
            pSt101: begin
                    if(in_i)            nxtSt_r = pSt1;
                    else                nxtSt_r = pSt10;
            end
            default:                    nxtSt_r = pIdle;
        endcase
    end
endmodule
```

2. Bit Steam Divisible by 5 Detector

```
module bitDiv5(
                        clk,
                        rst,
                        in_i,
                        flag_o
);
    input               clk;
    input               rst;
    input               in_i;
    output              flag_o;

    reg                 flag_o;

    parameter       pIdle   = 3'b000;
    parameter       pRem0   = 3'b001;
    parameter       pRem1   = 3'b010;
    parameter       pRem2   = 3'b011;
    parameter       pRem3   = 3'b100;
    parameter       pRem4   = 3'b101;

    reg[2:0]        curSt_r;
    reg[2:0]        nxtSt_r;

    always@(posedge clk)begin
        if(rst)     curSt_r <= #1 pIdle;
        else        curSt_r <= #1 nxtSt_r;
    end

    always@(*)begin
        flag_o = 1'b0;
        case(curSt_r)
            pIdle: begin
                    if(rst) nxtSt_r = pIdle;
                    else begin
                        if(in_i)        nxtSt_r = pRem1;
                        else   begin
                                flag_o = 1'b1;
                                nxtSt_r = pRem0;
                        end
                    end
            end
            pRem0: begin
                    if(in_i)                nxtSt_r = pRem1;
                    else   begin
                                            flag_o = 1'b1;
                                            nxtSt_r = pRem0;
                    end
            end
            pRem1: begin
                    if(in_i)                nxtSt_r = pRem3;
                    else                    nxtSt_r = pRem2;
            end
            pRem2: begin
                    if(in_i)begin
                                            flag_o = 1'b1;
                                            nxtSt_r = pRem0;
                    end
                    else                    nxtSt_r = pRem4;
            end
            pRem3: begin
                    if(in_i)                nxtSt_r = pRem2;
                    else                    nxtSt_r = pRem1;
            end
            pRem4: begin
                    if(in_i)                nxtSt_r = pRem4;
                    else                    nxtSt_r = pRem3;
            end
            default:                    nxtSt_r = pIdle;
        endcase
    end
endmodule
```

3. UART Transmitter ASMC



```
module    UartTx(... ...);
... ...;
```

```
                always@(posedge clk_i)begin
                        if (rst_i)          curSt_r <= idle;
                        else                        curSt_r <= nxtSt_r;
                end

                always@(*)begin
                    loadXMTShiftreg_o = 1’b0;
                    clear_o = 1’b0;
                    shift_o = 1’b0;
                    start_o = 1’b0;
                 nxtSt_r= curSt_r;
                    case(curSt_r)
                        Idle: begin
                                if(byteReady) begin
                                                loadXMTShiftreg_o = 1'b1;
                                                nxtSt_r=pWaiting;
                                end
                                else                nxtSt_r=pIdle;
                        end
                        Waiting: begin
                                if(TByte)begin
                                                start_o=1'b1;
                                                nxtSt_r=pSending;
                                end
                                else                nxtSt_r=pIdle;
                        end
                        Sending:begin
                                if(bitCount_r != kWordW+1)begin
                                        shift_o=1'b1;
                                end
                                else begin
                                        clear_o=1'b1;
                                        nxtSt_r=pIdle;
                                end
                        end
                        default: begin
                                nxtSt_r = Idle;
                        end
                  endcase
          end
endmodule
```