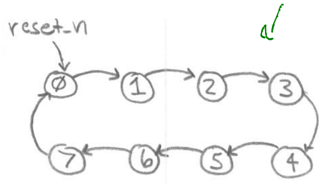
Synchronous Logic Blocks

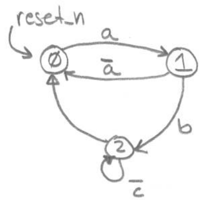
- Some synchronous logic blocks have standard implementations
  - Enabled flip-flops ←
  - Counters; Binary, Johnson ←
  - Shift Registers
  - Accumulator ←

Synchronous Logic Blocks

- These blocks are distinguished from state machines by very regular behavior.
- As such, they can be coded in a simplified manner.
- State machines tend to have more irregular behavior.
- ~~They~~ are coded in a flexible, general purpose way that makes it natural to express complex and non-regular behavior.



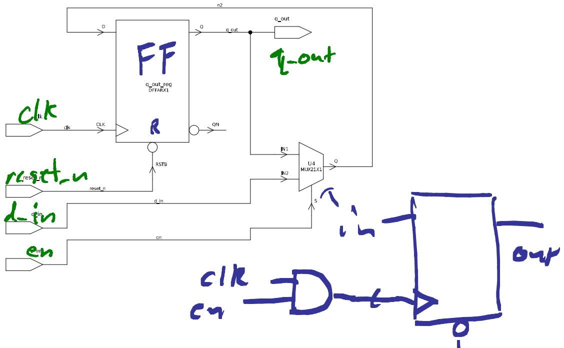
counter is regular



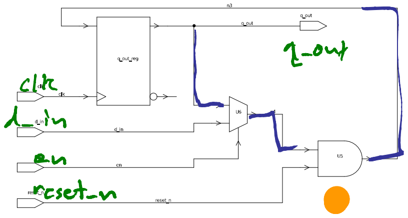
state machine is often irregular

Synchronous Logic Blocks

```
\\enabled flip-flop, asynchronous reset
module en_ff_ar(
  input       clk,
  input       reset_n,
  input       en,
  input       d_in,
  output reg  q_out);
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n) q_out <= '0;
   else if(en)   q_out <= d_in;
endmodule
```

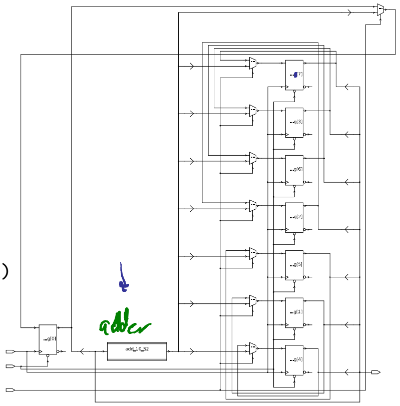


```
\\enabled flip-flop, synchronous reset
module en_ff_sr(
  input       clk,
  input       reset_n,
  input       en,
  input       d_in,
  output reg  q_out);
  always_ff @ (posedge clk)
   if(~reset_n) q_out <= '0;
   else if(en)   q_out <= d_in;
endmodule
```



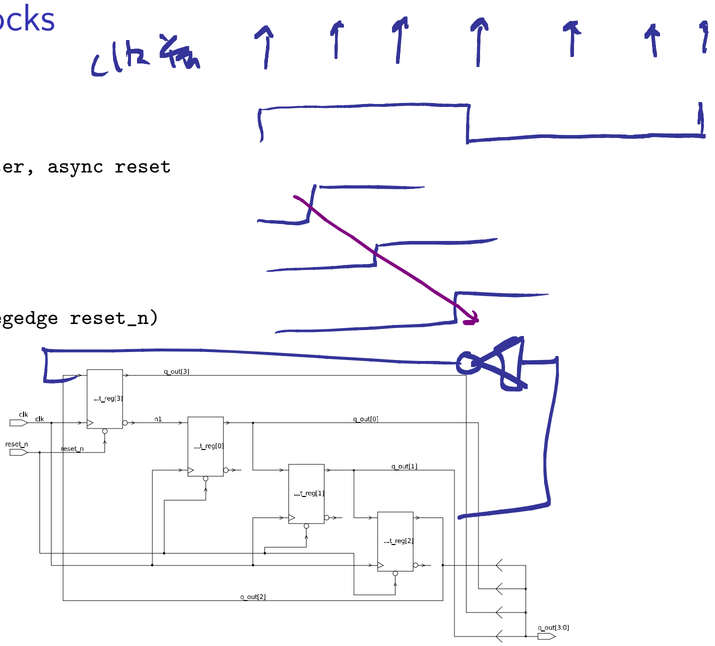
Synchronous Logic Blocks

```
\\binary counter, enabled, async reset
module bin_cntr_en_ar(
  input             clk,
  input             reset_n,
  input             en,
  output reg [7:0] q_out
  );
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n)  q_out <= '0;
   else if(en)   q_out <= q_out + 1;
endmodule
```



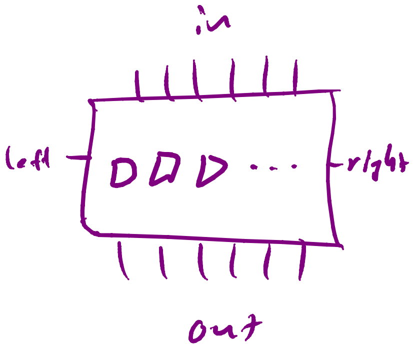
Synchronous Logic Blocks

```
\\Johnson (twisted-ring) counter, async reset
module jhsn_cntr_ar(
  input             clk,
  input             reset_n,
  output reg [3:0]  q_out
  );
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n)
     q_out <= '0;
   else begin
     q_out[0] <= ~q_out[3];
     q_out[1] <=  q_out[0];
     q_out[2] <=  q_out[1];
     q_out[3] <=  q_out[2];
   end //else
endmodule
```



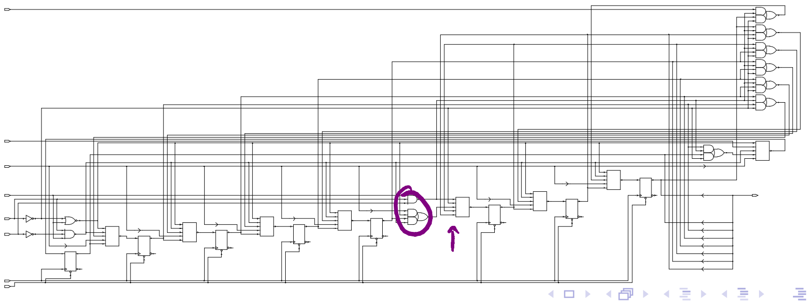
Synchronous Logic Blocks

```
module multi_reg_ar #(parameter WIDTH=8)(
//a multipurpose register
  input                  clk,       //clk
  input                  reset_n,   //async reset
  input                  en,        //enable
  input                  shift,     //shift=1, load=0
  input                  l_or_rt,   //left=1, right=0
  input                  din_left,  //data into MSB
  input                  din_right, //data into LSB
  input      [WIDTH-1:0] d_in,      //parallel inputs
  output reg [WIDTH-1:0] q_out      //parallel outputs
  );
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n)
     q_out <= ’0;               //clear register
   else if(en)
      if(~shift) q_out <= d_in; //broadside load
      else
        if (l_or_rt) // left
          q_out <= {q_out[WIDTH-2:0], din_left };   //shift left
        else
          q_out <= {din_right, q_out[WIDTH-1:1]};   //shift right
endmodule
```



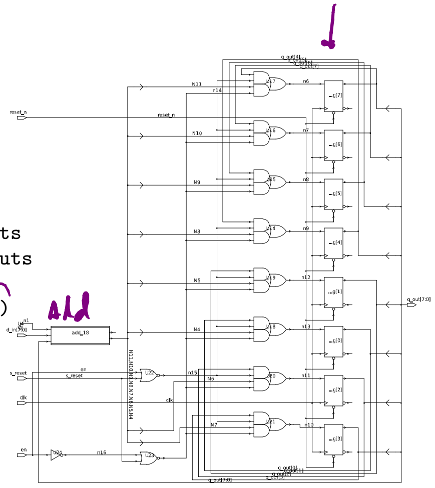
Synchronous Logic Blocks

```
module multi_reg_ar(
......
always_ff @ (posedge clk, negedge reset_n)
 if(~reset_n)
   q_out <= ’0;                //clear register
 else if(en)
    if(~shift) q_out <= d_in; //broadside load
    else
      if (l_or_rt)
        q_out <= {q_out[WIDTH-2:0], din_left };   //shift left
      else
        q_out <= {din_right, q_out[WIDTH-1:1]};   //shift right
```

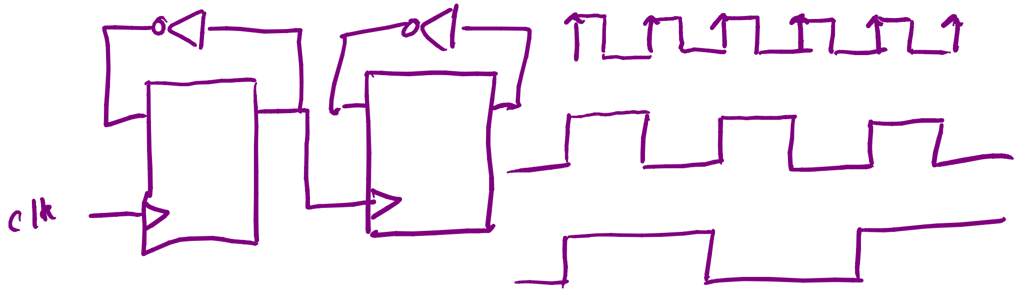


Synchronous Logic Blocks

```
module accumulator (
//a simple accumulator or summer with
//both sync and async resets
//sync reset operates independent of en
  input            clk,      //clk
  input            reset_n, //async reset
  input            s_reset, //sync reset
  input            en,      //enable
  input       [7:0] d_in,    //parallel inputs
  output reg [7:0] q_out     //parallel outputs
  );
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n)
     q_out <= '0;    //async clear
   else if(s_reset)
     q_out <= '0;    //sync clear
   else if(en)
     q_out <= q_out + d_in; //accumulate
endmodule
```



Clock Dividers!



Quiz:

1 ==, ===, >=, !=]

2) for ( )

3) case, if

4) always : sequential logic

5) initial , #5, wait

6) Grad Students: Presentation Topics