11. Registers & Counters

- Objectives
  This section deals with some simple and useful sequential circuits. Its objectives are to:
  - Introduce registers as multi-bit storage devices.
  - Introduce counters by adding logic to registers implementing the functional capability to increment and/or decrement their contents.
  - Define shift registers and show how they can be used to implement counters that use the one-hot code.

- Reading Assignment
  - Counters: Chapter 7, Sections 7.8 through 7.11 in Brown & Vranesic.
  - Verilog description of counters: Chapter 7, Sections 7.13 in Brown & Vranesic.

---

11.1. Registers

- A register is a memory device that can be used to store more than one bit of information.
- A register is usually realized as several flip-flops with common control signals that control the movement of data to and from the register.
  - Common refers to the property that the control signals apply to all flip-flops in the same way
  - A register is a generalization of a flip-flop. Where a flip-flop stores one bit, a register stores several bits
  - The main operations on a register are the same as for any storage devices, namely
    - Load or Store: Put new data into the register
    - Read: Retrieve the data stored in the register (usually without changing the stored data)

---

11.2. Control Signals

- When they are asserted, they initiate an action in the register
- Asynchronous Control Signals cause the action to take place immediately
- Synchronous Control Signals must be asserted during a clock assertion to have an effect

- Examples
  - On the following three registers, which control signals are asynchronous and which are synchronous? How are the control signals asserted?

---

11.3. Examples

```verilog
module reg1 (STO, CLR, D, Q);
(parameter n = 16;
input STO, CLR;
input [n-1:0] D;
output [n-1:0] Q;
reg [n-1:0] Q;
always @ (posedge STO or negedge CLR)
if (CLR == 0) Q <= 0;
else Q <= D;
endmodule
```

---

11.4. Examples

![Diagram of a register](image-url)
Verilog description of previous two registers

```verilog
module reg2 (CLK, CLR, LD, OE, D, Q);
parameter n = 4;
input CLK, CLR, LD, OE;
input [n-1:0] D;
output [n-1:0] Q;
reg [n-1:0] IQ, Q;
integer k;
always @(posedge CLK)
  if (CLR) IQ <= 0;
  else if (LD) IQ <= D;
always @(OE)
  if (OE) Q = IQ;
  else Q = 'bz;
endmodule
```

11.2. Counters

A counter is a register capable of incrementing and/or decrementing its contents

\[
Q \leftarrow Q \text{ plus } n \\
Q \leftarrow Q \text{ minus } n
\]

- The definition of "plus" and "minus" depend on the way the register contents encode the integers
- Binary Counters: Encode the integers with the binary number code
Example: 3-bit binary counter:

0 0 0  
0 0 1  
0 1 0  
0 1 1  
+  
1 0 0 
1 0 1 
1 1 0  
1 1 1  
-  
0 0 0  
0 0 1  
0 1 0  
0 1 1  
1 0 0  
1 0 1  
1 1 0  
1 1 1  

Count Sequence

Transitions Table
State Table

What does the counter count?  
The output signals are just the state variables

Example: 3-bit binary up/down counter

Example: Binary mod 6 counter

Design of a Binary Up Counter

Q2 Q1 Q0
0 0 0  
0 0 1  
0 1 0  
0 1 1  
1 0 0  
1 0 1  
1 1 0  
1 1 1  
Q0 Toggles every clock cycle  
Q1 toggles on those clock cycles where Q0=1  
Q2 toggles on those clock cycles where Q0=Q1=1  

Count Sequence

Qi toggles on every clock cycle where Qi = 1, for i > j ≥ 0
Design of a Binary Down Counter

- \( Q_0 \) toggles every clock cycle
- \( Q_1 \) toggles on those clock cycles where \( Q_0 = 0 \)
- \( Q_2 \) toggles on those clock cycles where \( Q_0 = Q_1 = 0 \)

Count Sequence
- \( Q_i \) toggles on every clock cycle where \( Q_j = 0 \), for \( i > j \geq 0 \)

Synchronous, Series-Carry Binary Counter

\[ T_W \geq t_{PFF} + (n-2)t_{PG} + t_{su} \text{ (for } n \geq 2) \]

Synchronous, Parallel-Carry Binary Counter

\[ T_W \geq t_{PFF} + t_{PG} + t_{su} \text{ (for } n \geq 3) \]
Asynchronous Counters

Typical MSI counter chip

LD and CLR are synchronous
LD asserted during the rising edge of the clock loads the register from ABCD.
CLR asserted during the rising edge of the clock clears the counter
CLR overrides LD
LD overrides EN

R_CO = Q_A * Q_b * Q_c * Q_d * ENT, used for cascading chips

Verilog description of the 74x163

module V74x163 (CLK, CLR_L, LD_L, ENP, ENT, D, Q, RCO);
input CLK, CLR_L, LD_L, ENP, ENT;
input [3:0] D;
output RCO;
output [3:0] Q;
reg [3:0] Q;
reg RCO;
always @(posedge CLK)
  if (CLR_L == 0) Q <= 4'b0000;
  else if (LD_L == 0) Q <= D;
  else if (ENT & ENP) Q <= Q +1;
always @(Q or ENT)
  if (Q == 15) RCO = 1;
  else RCO = 0;
endmodule

Verilog description of an up/down counter

module updowncount (R, Clock, L, E, up_down, Q);
parameter n = 8;
input [n-1:0] R;
input Clock, L, E, up_down;
output [n-1:0] Q;
reg [n-1:0] Q;
integer direction;
always @(posedge Clock)
  begin
    if (up_down) direction = 1;
    else direction = -1;
    if (L) Q <= R;
    else if (E) Q <= Q + direction;
  end
endmodule

Verilog description of mod-n counters

module upmodn (Ck, Q);
  parameter n = 6;
  input Ck;
  output [3:0] Q;
  reg [3:0] Q;
  always @(posedge Ck)
    if (Q == n) Q <= 0;
    else Q <= Q + 1;
endmodule

module dwnmodn (Ck, Q);
  parameter n = 5;
  input Ck;
  output [3:0] Q;
  reg [3:0] Q;
  always @(posedge Ck)
    if (Q == 0) Q <= n;
    else Q <= Q - 1;
endmodule
Design of Mod n Counters
- Mod 6 Up Counter
- Mod 5 Down Counter

Decoding Binary Counter States
- The decoding spikes are hazards that cannot be designed out.
- The following circuit will mask the decoding spikes, at the cost of delaying the outputs one clock cycle.

11.3. Shift Registers
- How would you add a control signal to control when the shift register shifted?
- How would you add parallel input capability and why would you want to?
- What kind of control signals are needed?
- Is the shift register drawn above a left shifter or a right shifter?
- How would you make a shift register that could shift either left or right and what control signals would you need?

Example: 74LS194
- Shift left is from A to D
- Shift right is from D to A
- CLR is asynchronous
Verilog Description Of A Shift Register

```verilog
module shift4 (D, LD, LI, Ck, Q);
  input [3:0] D;
  input LD, LI, Ck;
  output [3:0] Q;
  reg [3:0] Q;
  always @(posedge Ck)
    if (LD)
      Q <= D;
    else
      begin
        Q[0] <= Q[1];
        Q[1] <= Q[2];
        Q[2] <= Q[3];
        Q[3] <= LI;
      end
endmodule
```

Ring Counters

Self-Correcting Ring Counter

Johnson counter, switch-tail counter, moebius counter
11.29

Self-Correcting Johnson Counter

Odd Length Johnson Counter

This counter is also self-correcting

11.30

11.4. Design with MSI Components (Counters)

Approach

- The state diagram of a counter is essentially a cycle.
- Whenever a state diagram is close to a cycle, with only a few other edges, it can be realized using a counter by adding logic to take care of the transitions that are not in the cycle.
- The following control inputs on counters are useful:
  - ENABLE: Asserted to cause the counter to count in its natural count sequence. Deasserted to cause the counter to hold its current state.
  - CLEAR: Used to realize a count sequence that is not a power of two and for initialization.
  - LOAD: Used to modify the natural count sequence by forcing the counter to a given state.

Example:

Assert ENABLE when
- The counter is in state 0 and X = 1,
- The counter is in state 1,
- The counter is in state 2 and Z = 1,
- The counter is in state 3,
- The counter is in state 4, or
- The counter is in state 5 and Z = 1.

Assert CLEAR when
- The circuit is to be initialized, or
- The counter is in state 6 and Y = 1.

Assert LOAD when
- The counter is in state 5 and Y,Z = 1,0 (Load 100)
- The counter is in state 6 and Y = 0 (Load 011)
Example:

- Comment: A state table for this diagram would have 7 rows and 256 columns.

- CLEAR = (Q=1)•X1•X2 + INIT
- LOAD = (Q=5)•X6•X7 + (Q=6)
- A = (Q=6)
- B = (Q=5)
- C = 0
- ENABLE = (Q=0)•X0 + (Q=1)•X1 + (Q=2)•X3 + (Q=3)•X4 + (Q=4)•X5 + (Q=5)•X7

11.5. Tips & Tricks

- Don't use asynchronous counters, especially if the clock period is close to the flip-flop propagation delay.
- Don't build mod n counters from binary counters with asynchronous clears.

11.6. Pitfalls

- Detecting the wrong state for resetting a mod n counter.

11.7. Review

- Register control signals and assertions.
- Binary counters and their operations.
  - Reset, Load, Output Enable.
  - Counter timing; maximum clock frequency.
- Mod-n counters
  - Synchronous vs. asynchronous load and reset signals.
- Shift registers and shift register counters.
  - Ring counters, Johnson counters, etc
  - Self-correcting counters
- Counter realization of sequential circuits