Chapter 5
Registers & Counters

Overview of Chapter 5
- Registers:
  - Basic registers
  - Register with load and hold
  - Shift register
  - Register with parallel load and other functions
- Counters:
  - Basics
  - Ripple counters
  - Synchronous counters
  - Counters with parallel load
  - General counters
- HDL Representation:
  - Verilog descriptions

Logic and Computer Design Fundamentals
Kewal K. Saluja and Yu Hen Hu

Chapter 5
Registers
- Register – a collection of binary storage elements
- In theory, a register is sequential logic which can be defined by a state table
- More often think of a register as storing a vector of binary values
- Frequently used to perform simple data storage and data movement and processing operations, but can also be used to store sequential circuit state.

Example: 2-bit Register
- How many states are there?
- How many Inputs? Outputs?
- What is the output function?
- What is the Next State function?
- Moore or Mealy?

State Table:

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 A0</td>
<td>A1 A0</td>
<td>Y1 Y0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>1 1</td>
</tr>
</tbody>
</table>

What happens with a 4-bit register?

Registers: Storage Model
- The sequential logic model is not useful for whole registers
- How can we “selectively” store data in some registers and not change the contents of others?
  - Stop the clock when we don’t want change – clock gating (be careful!)
  - Use a flip flop that can “hold” under some set of logical inputs.
  - SR, T and JK flip-flops all have “hold” states when inputs are “0”.
  - We can use logic and a load control signal to build a register that can be controllably loaded.

Registers with Clock Gating
- Load signal is used to enable the clock signal to pass through if 1 and prevent the clock signal from passing through if 0.
- Example: For Positive Edge Triggering or Neg. Pulse MS
  - Master Clock
  - Gated Clock
  - Load

- What logic is needed for gating? GC = CLK+ LOAD
- What is the problem? Clock Skew of GC
Registers with Load Control

- A more reliable way to selectively load a register is to run the clock continuously and use a load control to change selectively change the register contents.
- Example:
  - 2-bit register with Load Control
- Note: SR FFs would also work here.

Shift Registers

- Shift Registers are a special class of registers that can be used to store and manipulate data.
- In the simplest case, the shift register is simply a set of D flip-flops connected in a row like this:

  ![Shift Register Diagram](image)

  - Data input, "In", is called the "serial input", or "shift right input".
  - Data output, "Out", is often called the "serial output".
  - "A", "B", "C", and "Out" collectively are the "parallel outputs".

Parallel Load Shift Registers

- By adding a mux between each shift register stage, we can "shift", or "load" data.

  ![Parallel Load Shift Register Diagram](image)

  If "SHIFT" line is low, "A" and "B" are replaced by the data on "LOAD_A" and "LOAD_B" lines, else data shifts each clock.

Shift Registers with More Functions

- By placing a 4-input multiplexer in front of each D flip-flop in a shift register, we can implement a device which:
  1. Shifts right
  2. Shifts left
  3. Holds
  4. Parallel loads

- This device becomes a "universal" shift register that can be used for a number of different things.
- In particular, such a structure is good for tying all state elements together so you can "scan" the current state in or out of a system for test purposes.
- The shift-left, shift-right and parallel-load inputs give three ways to load data into the registers.
Serial Data Operations

* By using two shift registers for operands, a full adder, and one more flip flop (for the carry), it is possible to add two numbers serially, starting at the least significant bit.
* Serial addition is not a bad way to add huge numbers of operands, since a “tree” of adders can be made to any depth, and each new level doubles the number of operands.
* Other operations can be performed serially as well, such as parity generation/checking or more complex error-check codes.
* Shifting a binary number left is equivalent to multiplying by 2.
* Shifting a binary number right is equivalent to dividing by 2.

Serial Adder

* The circuit shown uses two shift registers for operands A(3,0) and B(3,0).
* A full adder, and one more flip flop (for the carry) is used to compute the sum.
* The result is stored back in the “B” register along with the final carry.
* This can be extended to trees of adders, adding a large number of operands.

Counters

* Counters are sequential circuits which “count” through a specific state sequence. They can count up, count down, or count through other fixed sequences. Two distinct types are in common usage:
  * Ripple Counters
    * Clocks are connected to flip-flop outputs, thus not truly synchronous
    * Outputs are “delayed” for higher bits.
    * Resurgent because of low power consumption
  * Synchronous Counters
    * Clocks are directly connected to the flip-flops.
    * Logic is used to implement the desired state sequencing.

Counter Basics: Divide by 2

Consider the following circuit:

Draw Waveform A and B

Divide Clock Frequency by 2

The waveforms look like a new clock with twice the period of CP (half the frequency).
The flip-flops are said to “divide-by-2” since the frequency of the output waveform is
1/2 the frequency of clock CP.

Ripple Counter

What happens now? (note clock change on B)

Draw the waveform for B
Here’s what happens:

- Note that B "divides" the frequency in A by 2 ⋅ or 
doubles the period.
- Notice the count (B A) 
  base 10 of:
  \[3\to\!0\to\!1\to\!0\to\!3\to\!2\to\!1\ldots\]
- What kind of counter is this?
  Down Counter!

What type of counter is this?
Up Counter!
Can also use negative edge-
trigerring

Here is the detailed 
waveform behavior:

Starting with A=B=C = "1", equivalent to (C,B,A) = 7 
base 10, the next count will increment the count to 
(A,B,C) = 0 base 10. Here’s what happens in fine 
timing detail:

- The clock to output 
delay causes an 
  increasing delay from 
  clock edge for each 
  stage transition.
- Thus, the count 
  "ripples" from least to 
  most significant bit.
- For n bits, total worst 
  case delay is \( n t_{PHL}. \)
Synchronous Counters

* In order to eliminate the "ripple" effect, we will use a common clock for each flip-flop and a combinational circuit to generate the next state.
* One way to generate a counter is with an Adder/Register circuit:

```
| D3 | Q3 |
| D2 | Q2 |
| D1 | Q1 |
| D0 | Q0 |
```

4-Bit Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>CP</th>
<th>SUM</th>
<th>CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

* Here "CNT" is the count constant:
  0000 is HOLD, 1111 is DOWN and 0001 is UP.
* Note potential logic simplification due to constant applied to B.

Synchronous Counters (Continued)

* A simple 2-bit synchronous counter can be made with two "T" Flip-Flops:

```
<table>
<thead>
<tr>
<th>CP</th>
<th>T</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

* Note that the Q from the first stage enables the second stage to toggle.
* Does it count "up" or "down"?
* How do you extend this to three stages?

Synchronous Counters (Continued)

* The concept can be extended to multiple stages. For binary "UP" counters, the upper stages toggle when ALL of the lower stages are at the value "1" and the clock occurs.
* By "ANDing" in a count enable signal to each "T" input, we can produce a "HOLD" count signal.

Synchronous Counters – Serial Gating

* When a two-input AND gate is used for each stage of the counter with a "ripple-like" carry, this is referred to as serial gating.
* As the size of the counter increases the delay through the combinational logic increases roughly in proportion to \( n \), the number of stages.

Synchronous Counters – Parallel Gating

* When a multiple-input ( >2) AND gates are used for each stage of the counter with logic dedicated to each stage or to a group of stages, this is referred to as parallel gating. It resembles carry lookahead in an adder.
* As the size of the counter increases the delay through the combinational logic increases roughly in proportion to \( n/m \), the number of stages/the group size.

Design: Synchronous BCD

* We can use the sequential logic model to design a synchronous BCD counter with T flip-flops. Below is the State Table.

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>T-FF Excitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>1 0 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 0 1 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>1 1 0 1</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>1 1 1 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 0 0 1</td>
<td>1 0 1 1</td>
</tr>
</tbody>
</table>

Don't care states have been left out here.
Synchronous BCD (Continued)

• Use K-Maps to minimize the next state function:

\[
\begin{align*}
T_8 & = Q_8'Q_1 + Q_4Q_2'Q_1 \\
T_4 & = Q_2Q_1 \\
T_2 & = Q_8'Q_1 \\
T_1 & = 1
\end{align*}
\]

Note: Don’t Cares are included here.

Synchronous BCD (Continued)

What about the Don’t Cares now? All Next States are now specified!!

Current State | Next State | T-FF Excitation
-------------|------------|-----------------|
0 0 0 0 | 0 0 0 1 | 0 0 0 1
0 0 0 1 | 0 0 1 0 | 0 0 1 1
1 0 0 0 | 1 0 1 0 | 0 0 1 1
1 0 0 1 | 0 0 0 0 | 1 0 0 1
1 0 1 0 | ? ? ? ? | 1 0 0 1
1 0 1 1 | ? ? ? ? | 1 0 0 1
1 1 0 0 | ? ? ? ? | 0 0 0 1
1 1 0 1 | ? ? ? ? | 0 0 0 1
1 1 1 0 | ? ? ? ? | 0 0 0 1
1 1 1 1 | ? ? ? ? | 0 0 0 1

Synchronous BCD (Continued)

* Don’t care states have now been specified by the logic.

Current State | Next State | T-FF Excitation
-------------|------------|-----------------|
0 0 0 0 | 0 0 0 1 | 0 0 0 1
0 0 0 1 | 0 0 1 0 | 0 0 1 1
1 0 0 0 | 1 0 1 0 | 0 0 1 1
1 0 0 1 | 0 0 0 0 | 1 0 0 1
1 0 1 0 | 0 1 1 0 | 1 0 0 1
1 0 1 1 | 0 1 1 0 | 1 0 0 1
1 1 0 0 | 1 1 0 0 | 0 0 0 1
1 1 0 1 | 0 1 0 0 | 1 0 0 1
1 1 1 0 | 1 1 1 0 | 1 0 0 1
1 1 1 1 | 1 1 1 0 | 1 0 0 1

Synchronous BCD (Continued)

• What does the complete state diagram look like?

• How does the sequential machine get into some of the states?
Counting Modulo N

• The Load feature can be used to preset the counter synchronously on command.
• The Clear feature can asynchronously reset the counter to zero. (This can lead to counts which are present only a very short time).
• By detecting a "terminal" count of N-1 in a Modulo-N count sequence, we can synchronize load to "zero" to start over.
• By detecting a "terminal" count of N in a Modulo-N count sequence, we can clear the count synchronously to "zero" to start over.
• Alternatively, we can detect the "all-ones" terminal count and use load to preset a count of the maximum count value minus (N-1).

Counting Modulo 7

• A synchronous 4-bit binary counter with a synchronous load and an asynchronous clear is to be used to make a Modulo 7 counter.
• Use the Load feature to detect the count "6" and load in "zero". This gives a count of 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0... etc.

Counting Modulo 7, Asyn.Clear

• A synchronous 4-bit binary counter with a synchronous load and an asynchronous clear is to be used to make a Modulo 7 counter.
• Use the Clear feature to detect the count "7" and clear the count to "zero". This gives a count of 0, 1, 2, 3, 4, 5, 6, 7(short) 0, 1, 2, 3, 4, 5, 6, 7(short) 0, etc.
• DON'T DO THIS! Referred to as a “suicide” counter!

Counting Modulo 7, Preset 9

• A synchronous, 4-bit binary counter with a synchronous load and an asynchronous clear is to be used to make a Modulo 7 counter.
• Use the Load feature to preset the count to "9" when you detect the count "15". This gives a count of 9, 10, 11, 12, 13, 14, 15, 9, 10, 11, 12, 13, 14, 15, 0, ..., etc.
• Sometimes the "Detect 15" is built in to the counter.

Timing Sequences

• For digital systems, useful to generate a multi-phase sequence of timing signals to perform different functions at different intervals. There are many ways to do this.
• Here are a few that start with a clock and use a "counter" to divide the clock into separate phases.
  Counter/Decoder - connect the output of a counter to a decoder.
  Ring Counter - shift a single pulse down a chain of flip-flops connected as a shift register. For "N" phases, use "N" flip-flops. There can be "unwanted" states if not initialized properly.
  Johnson Counter - (Switch-Tail Ring) uses an "N" bit shift register and decoders to produce 2^N phases. There can be "unwanted" states if not initialized properly.

Counter Decoder Example

• Here is one variant of a counter-decoder multi-phase clock generator:
  • Note the "Glitches" due to the ripple count. A synchronous counter would have been better than the ripple counter.
  • Even with a synchronous counter, glitches are possible although reduced in duration.
  • Method OK if "glitches" not a problem, for example, if signals are used as enables in a positive edge-triggered system.
Ring Counter

• A ring counter is just a shift register with feedback. Assuming the initial state ABCD is 1000 we get the timing diagram shown:
• The state must be initialized to operate correctly. Use logic to assure this.
• Same circuit can be used with positive edge-triggering.

Node Counter (Switch-Tail)

• A Johnson counter is also a shift register with feedback. Assuming the initial state ABC is 000 we get the timing diagram shown:
• The state must be initialized to operate correctly. Use logic to assure this.

Verilog for Registers and Counters

• Register – same as flip-flop except multiple bits:
  ```verilog
  reg[3:0] Q;
  input[3:0] D;
  always @(posedge CLK or posedge RESET)
  begin
    if (RESET) Q <= 4'b0000;
    else Q <= D;
  end

  • Shift Register – use concatenate:
    Q <= {Q[2:0], SI};

  • Counter – use increment/decrement:
    count <= count + 1; or count <= count - 1
  ```

Verilog Description of Left Shift Register

```verilog
// 4-bit Shift Register with Reset
// (See Figure 5-3)
module srg_4_r_v (CLK, RESET, SI, Q, SO);
  input CLK, RESET, SI;
  output [3:0] Q;
  output SO;

  reg [3:0] Q;
  assign SO = Q[3];

  always @(posedge CLK or posedge RESET)
  begin
    if (RESET) Q <= 4'b0000;
    else Q <= {Q[2:0], SI};
  end
endmodule
```

Verilog Description of Binary Counter

```verilog
// 4-bit Binary Counter with Reset
// (See Figure 5-10)
module count_4_r_v (CLK, RESET, EN, Q, CO);
  input CLK, RESET, EN;
  output [3:0] Q;
  output CO;

  reg [3:0] count;
  assign Q = count;
  assign CO = (count == 4'b1111 ? 1'b1 : 1'b0);

  always @(posedge CLK or posedge RESET)
  begin
    if (RESET) count <= 4'b0;
    else if (EN) count <= count + 1;
  end
endmodule
```