 Registers

- **Readings: 5.8-5.9.3**
- **Storage unit. Can hold an n-bit value**
- **Composed of a group of n flip-flops**
  - Each flip-flop stores 1 bit of information
Controlled Register

<table>
<thead>
<tr>
<th>Reset</th>
<th>Load</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q = old Q</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Q = 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q = D</td>
</tr>
</tbody>
</table>

\[ \text{DFF} \]

\[ \text{DFF} \]

\[ \text{DFF} \]

\[ \text{DFF} \]
Shift Register

- Register that shifts the binary values in one or both directions
Transfer of Data

- 2 modes of communication: Parallel vs. Serial
  - Parallel: all bits transferred at the same time
  - Serial: one bit transferred at a time
- Shift register can be used for serial transfer
## Shift Register w/Parallel Load

<table>
<thead>
<tr>
<th>Shift</th>
<th>Load</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q = old Q</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Shift</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Parallel Load</td>
</tr>
</tbody>
</table>

![Dff with clock]
Conversion between Parallel & Serial

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Op - Sender</th>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Op - Recvr</th>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Bidirectional Shift Register w/Parallel Load

<table>
<thead>
<tr>
<th>Shift</th>
<th>Load</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q = old Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Parallel Load</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Shift Up (V*2)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Shift Down (V/2)</td>
</tr>
</tbody>
</table>

![Diagram of shift register](image)
Counters

- A reg. that goes through a specific state sequence
- *n-bit Binary Counter*: counts from 0 to $2^N - 1$ in binary
- *Up Counter*: Binary value increases by 1
- *Down Counter*: Binary value decreases by 1
- 3-bit binary up counter state diagram:
Binary Up-Counter Imp.
# Complex Binary Counter

<table>
<thead>
<tr>
<th>Load</th>
<th>Count</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q = old Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Up Count</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Load Parallel</td>
</tr>
</tbody>
</table>
Arbitrary Sequence Counters

- Design a 3-bit count that goes through the sequence
  000->010->100->101->111->110->001->011->000->...
module upcounter #(parameter WIDTH=8) (out, incr, reset, clk);
output logic [WIDTH-1:0] out;
input logic incr, reset, clk;
endmodule
Memory

- Need method for storing large amounts of data
  - Computer programs, data, pictures, etc.

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>00111110</td>
</tr>
<tr>
<td>000001</td>
<td>01101011</td>
</tr>
<tr>
<td>000010</td>
<td>01011101</td>
</tr>
<tr>
<td>000011</td>
<td>01100011</td>
</tr>
<tr>
<td>000100</td>
<td>00111110</td>
</tr>
<tr>
<td>000101</td>
<td>00000000</td>
</tr>
<tr>
<td>000110</td>
<td>11111111</td>
</tr>
<tr>
<td>000111</td>
<td>01010101</td>
</tr>
<tr>
<td>001000</td>
<td>10101010</td>
</tr>
<tr>
<td>001001</td>
<td>00100001</td>
</tr>
<tr>
<td>001010</td>
<td>11011010</td>
</tr>
</tbody>
</table>

- RAM: Random Access Memory, Read/Write
- ROM: Read-only Memory
RAM Cell

- Requirements:
  - Store one bit of data
  - Change data based on input when row is selected

Input

Row Select

\[
\begin{array}{cc}
D & Q \\
\text{en} & \\
\end{array}
\]
RAM example

- Use a memory to do a programmable 32-picture animation on a 7-segment display
Verilog Memories

module memory16x6 (data_out, data_in, addr, we, clk);

output logic [5:0] data_out;
input logic [5:0] data_in;
input logic [3:0] addr;
input logic we, clk;

logic [5:0] mem [15:0];

always_ff @(posedge clk) begin
  data_out <= mem[addr];
  if (we)
    mem[addr] <= data_in;
end
endmodule
Field Programmable Gate Arrays (FPGAs)

- Readings: B.6-B.6.5

Logic cells imbedded in a general routing structure

- Logic cells usually contain:
  - 6-input Boolean function calculator
  - Flip-flop (1-bit memory)

All features electronically (re)programmable
Using an FPGA

Verilog

// Verilog code for 2-input multiplexer
module AOI (F, A, B, C, D);
output F;
input A, B, C, D;
assign F = ~((A & B) | (C & D));
endmodule

module MUX2 (V, SEL, I, J);
output V;
input SEL, I, J;
wire SELB, VB;
not G1 (SELB, SEL);
AOI G2 (VB, I, SEL, SELB, J);
not G3 (V, VB);
endmodule

FPGA CAD Tools

Bitstream

Simulation

00101010001010010
10010010010011000
10101000101011000
10101001010010101
00010110001001010
10101001111001001
01000010101001010
10010010000101010
10100101010010100
01010110010101010
010100100101001
FPGA Programming

Bitstream

\( \text{\textcopyright} = 1 \text{ memory cell (stores 1 bit of info)} \)
FPGA Combinational Logic

- How can we use Muxes and Programming bits to compute combinational binary function $F(A,B,C)$?

- Creates a “LUT” or lookup table.
FPGA Sequential Logic

- How do we put DFF’s onto LUT outputs only when we need them?

- Creates a “LE” or logic block
How do we combine LE’s to build larger functions?

- This is an Altera “LAB”.

FPGA Local Routing
FPGA Global Routing

- Can’t do all-to-all/crossbar routing, so what?

```
LAB
```

```
LAB
```
FPGA CAD

- CAD = “Computer-Aided Design”

Verilog

- Tech Mapping: Convert Verilog to LUTs
- Placement: Assign LUTs to specific locations
- Routing: Wire inputs to outputs
- Bitstream Generation: Convert mapping to bits

Bitstream
Modern FPGA: Stratix V

- Logic Blocks
- Multipliers & DSP
- Embedded Memories
- Clocking Logic
- I/O Protocols
DE1-SoC FPGA: Cyclone V 5CSEMA5F31C6N

- ALMs (2x6-LUT): 32k
- DFFs: 128k
- RAMs (10Kb): 3.9k
- 18x18 Hard Multipliers: 174
- Clock generators (PLLs): 6
- General-purpose I/Os: 288
- ARM Cortex A9 cores: 2