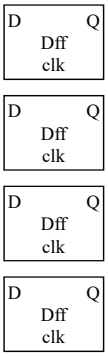
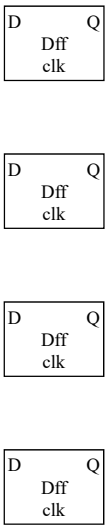
Registers

- 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



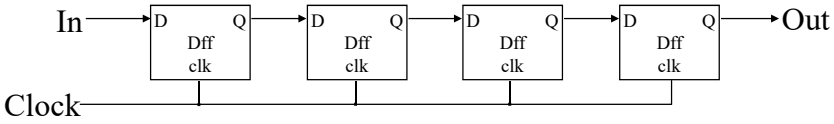
Controlled Register

Reset	Load	Action
0	0	Q = old Q
1	0	Q = 0
0	1	Q = D



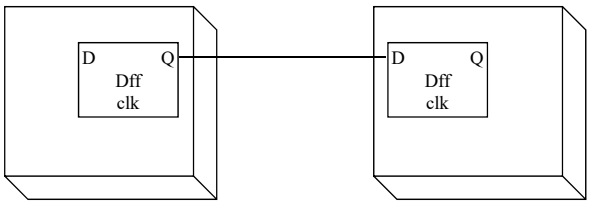
Shift Register

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



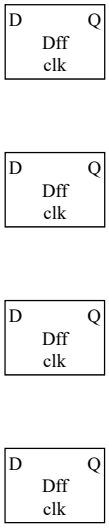
Transfer of Data

- 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

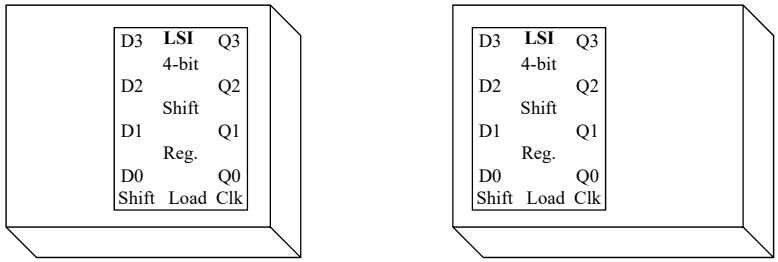


Shift Register w/Parallel Load

Shift	Load	Action
0	0	Q = old Q
1	0	Shift
X	1	Parallel Load



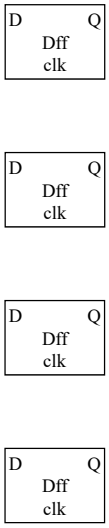
Conversion between Parallel & Serial



Cycle	Op - Sender	Q0	Q1	Q2	Q3		Op - Recvr	Q0	Q1	Q2	Q3
0											
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											

Bidirectional Shift Register w/Parallel Load

Shift	Load	Action
0	0	Q = old Q
0	1	Parallel Load
1	0	Shift Up (V*2)
1	1	Shift Down (V/2)



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

Load	Count	Action
0	0	Q = old Q
0	1	Up Count
1	0	Reset
1	1	Load Parallel

Arbitrary Sequence Counters

- Design a 3-bit count that goes through the sequence 000->010->100->101->111->110->001->011->000->...

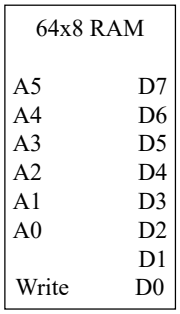
Counters in Verilog

```
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.

Address	Data
000000	00111110
000001	01101011
000010	01011101
000011	01100011
000100	00111110
000101	00000000
000110	11111111
000111	01010101
001000	10101010
001001	00100001
001010	11011010

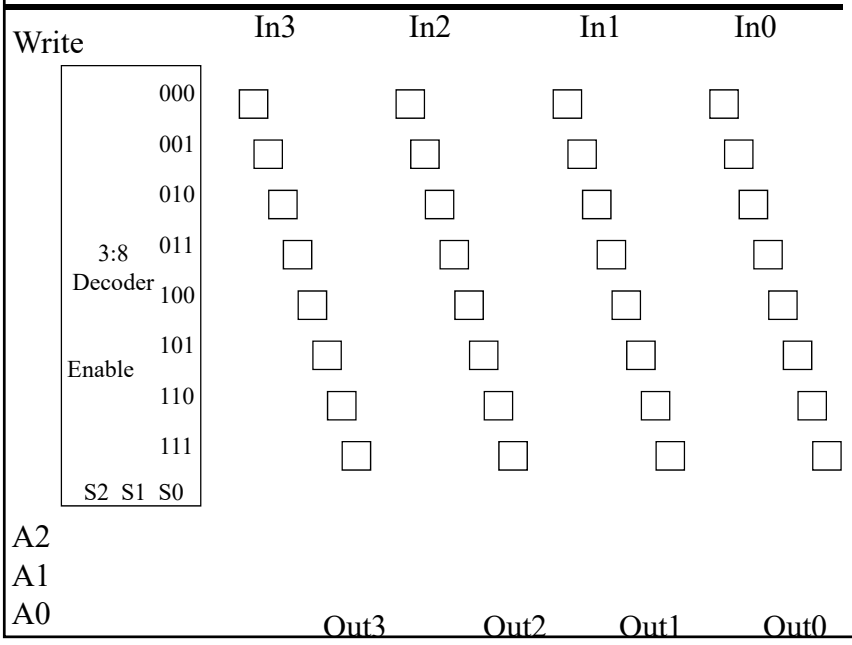


- RAM: Random Access Memory, Read/Write
- ROM: Read-only Memory

RAM Cell

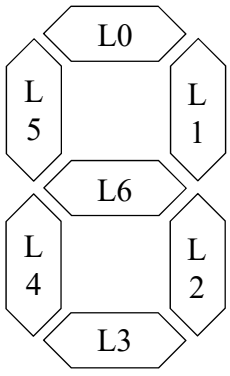
- Requirements:
  - Store one bit of data
  - Change data based on input when row is selected

8x4 RAM



RAM example

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

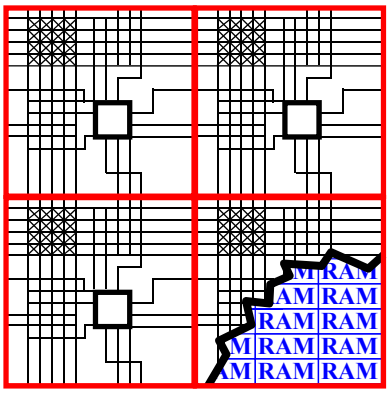


Verilog Memories

```
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



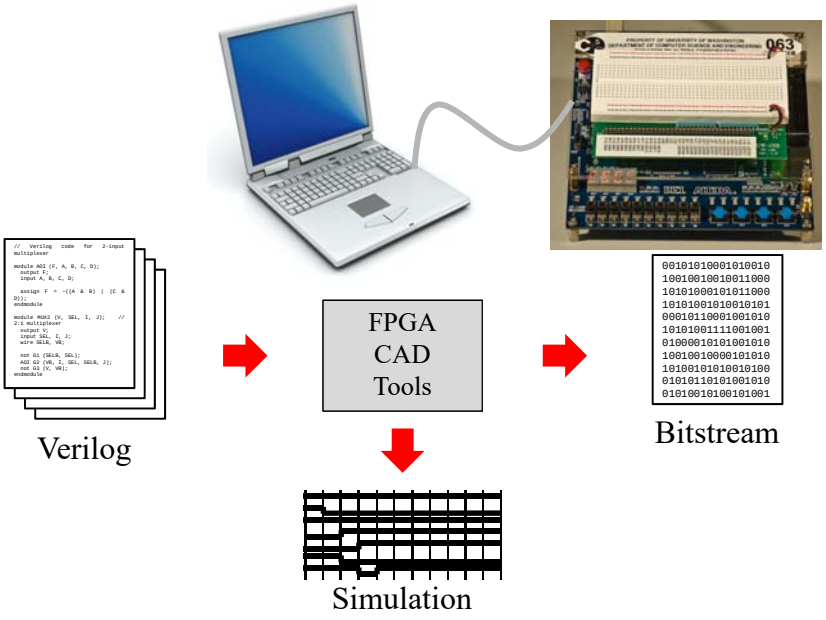
Logic cells imbedded in a general routing structure

Logic cells usually contain:

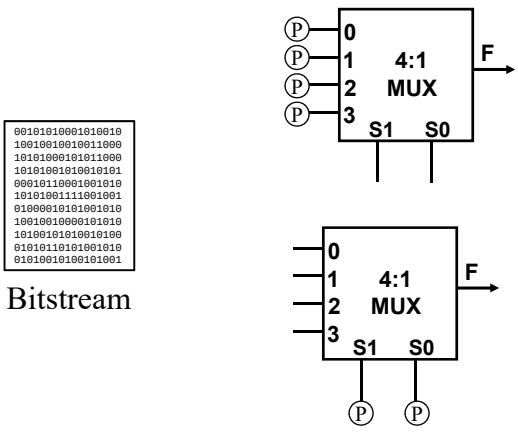
- **6-input Boolean function calculator**
- **Flip-flop (1-bit memory)**

All features electronically (re)programmable

Using an FPGA

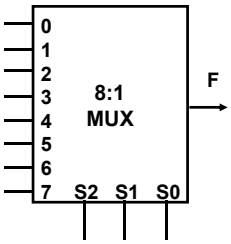


FPGA Programming



FPGA Combinational Logic

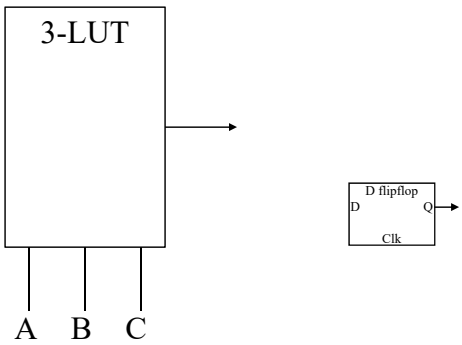
- How can we use Muxes and Programming bits to compute combinational binary function F(A,B,C)?



- Creates a "LUT" or lookup table.

FPGA Sequential Logic

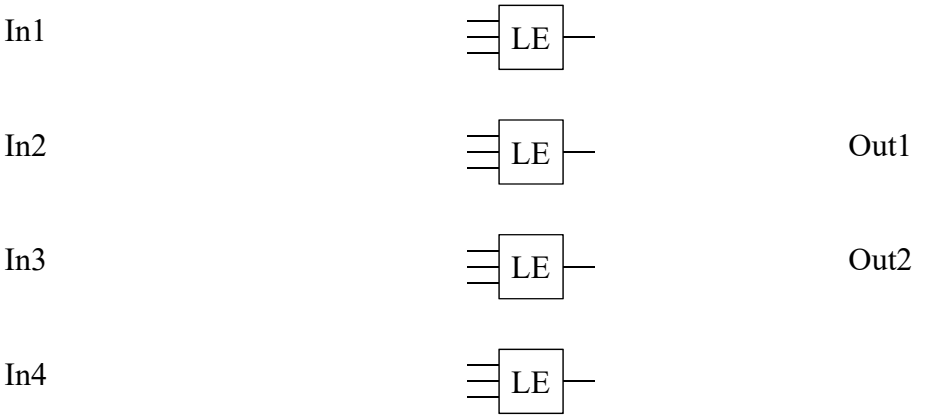
- How do we put DFF's onto LUT outputs only when we need them?



- Creates a "LE" or logic block

FPGA Local Routing

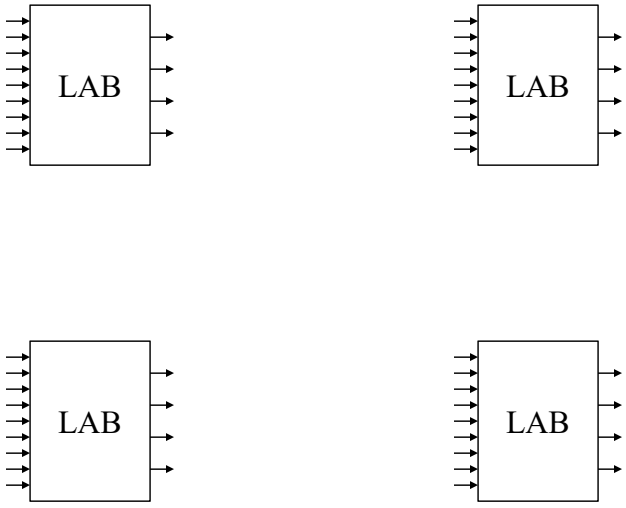
- How do we combine LE's to build larger functions?



- This is an Altera "LAB".

FPGA Global Routing

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



FPGA CAD

- CAD = "Computer-Aided Design"

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