Topic 9

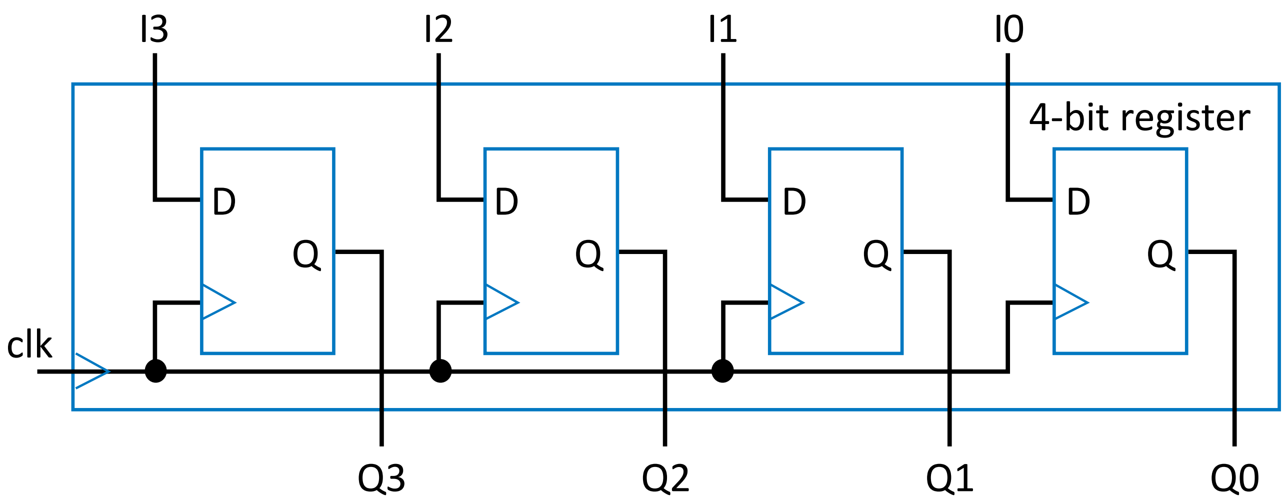
Register & Shifter

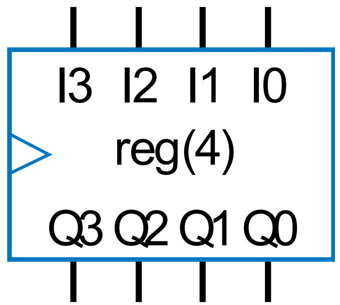
Introduction

- Two major subsystems in typical digital system
  - Datapath
    - Datapath routes data to particular destination device according to control signals generated by controller
  - Controller
    - Controller generates signals to control datapath based on environment event or state of a circuit
- This chapter introduces an important datapath component and simple datapaths

Registers

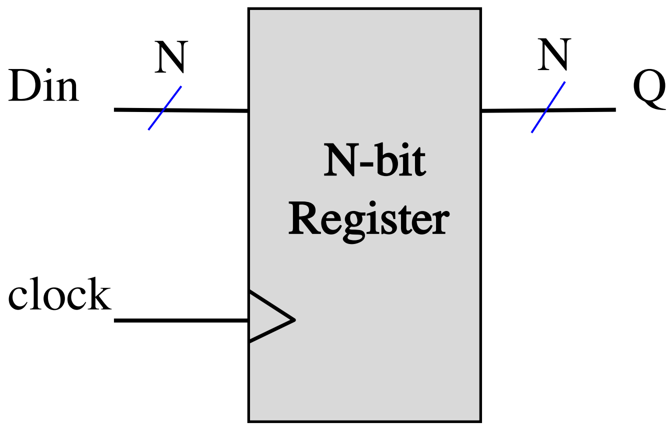
- Can store data, very common in datapaths
- Basic register: Loaded every cycle





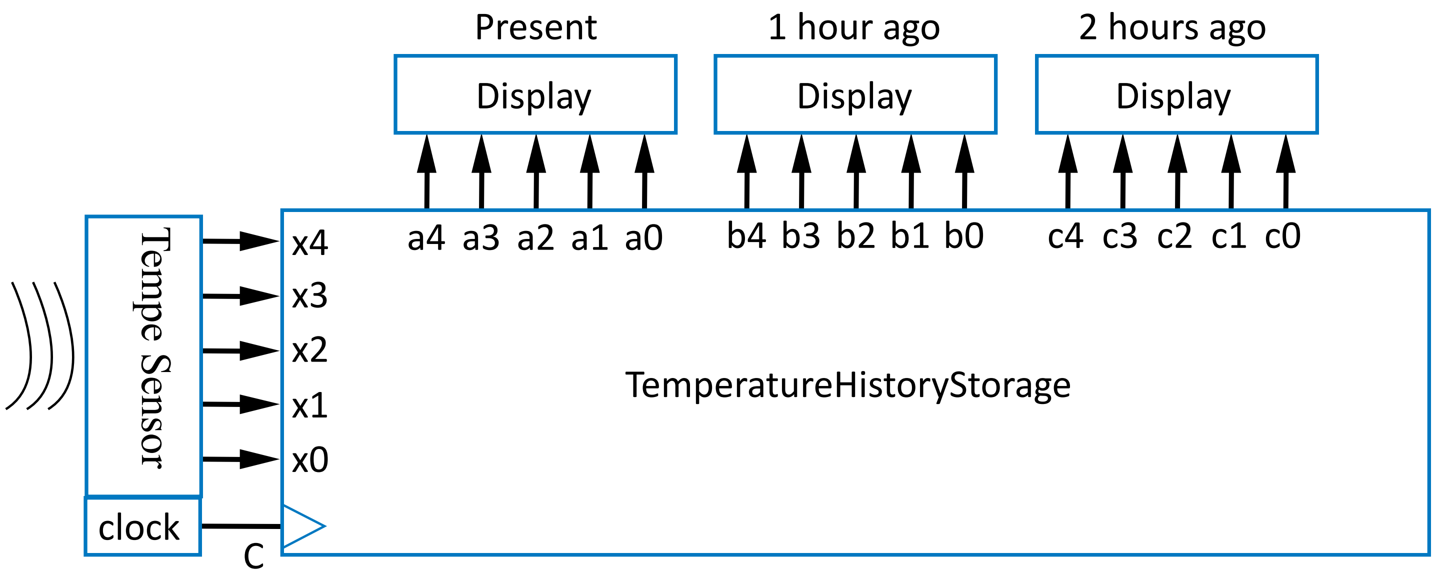
Verilog Modeling of Registers

```
module Reg_N_bits (Q, Din, clock);
  parameter size = 4;
  input clock;
  input [size-1:0] Din;
  output [size-1:0] Q;

  reg Q;

  always @ (posedge clock)
  begin
    Q <= Din;
  end
endmodule
```



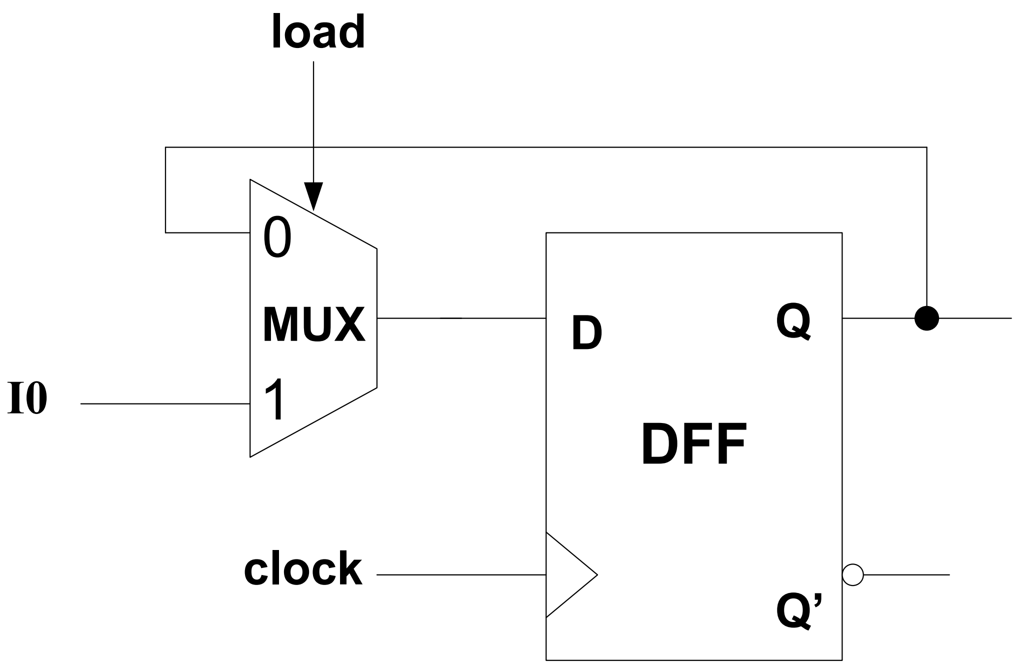
Example Using Registers: Temperature Display

- Temperature history display
  - Sensor outputs temperature as 5-bit binary number
  - Timer pulses C every hour
  - Record temperature on each pulse, display last three recorded values

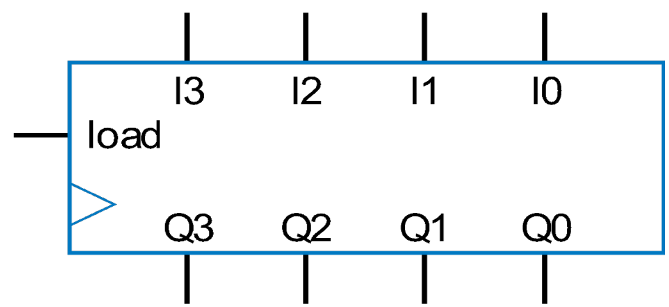


Register with Synchronous Parallel Load

- Add 2x1 mux to each flip-flop
- Register's load input selects mux input to pass
  - Either existing flip-flop value, or new value to load

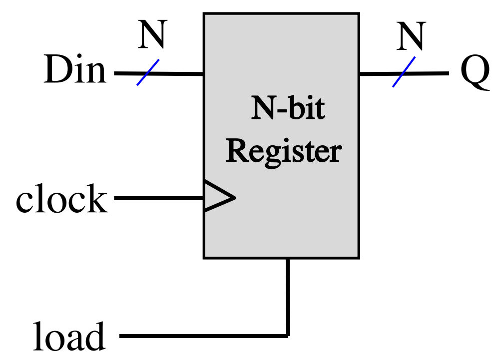


Synchronous active high Load



Verilog Modeling of Registers

```
module Reg_N_bits (Q, Din, clock, load);
  parameter size = 4;
  input clock, load;
  input [size-1:0] Din;
  output [size-1:0] Q;

  reg Q;

  always @ (posedge clock)
  begin
    if (load) Q <= Din;
  end
endmodule
```

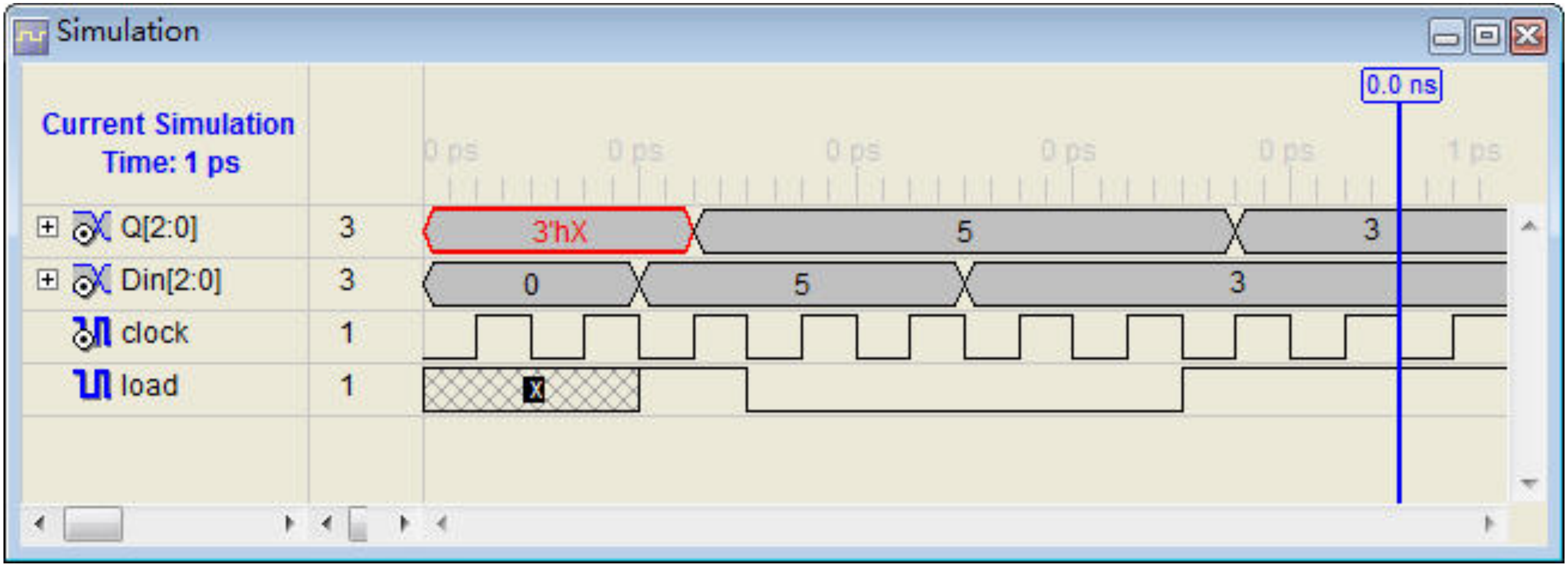


Incomplete **`if`** statement, but OK, why?

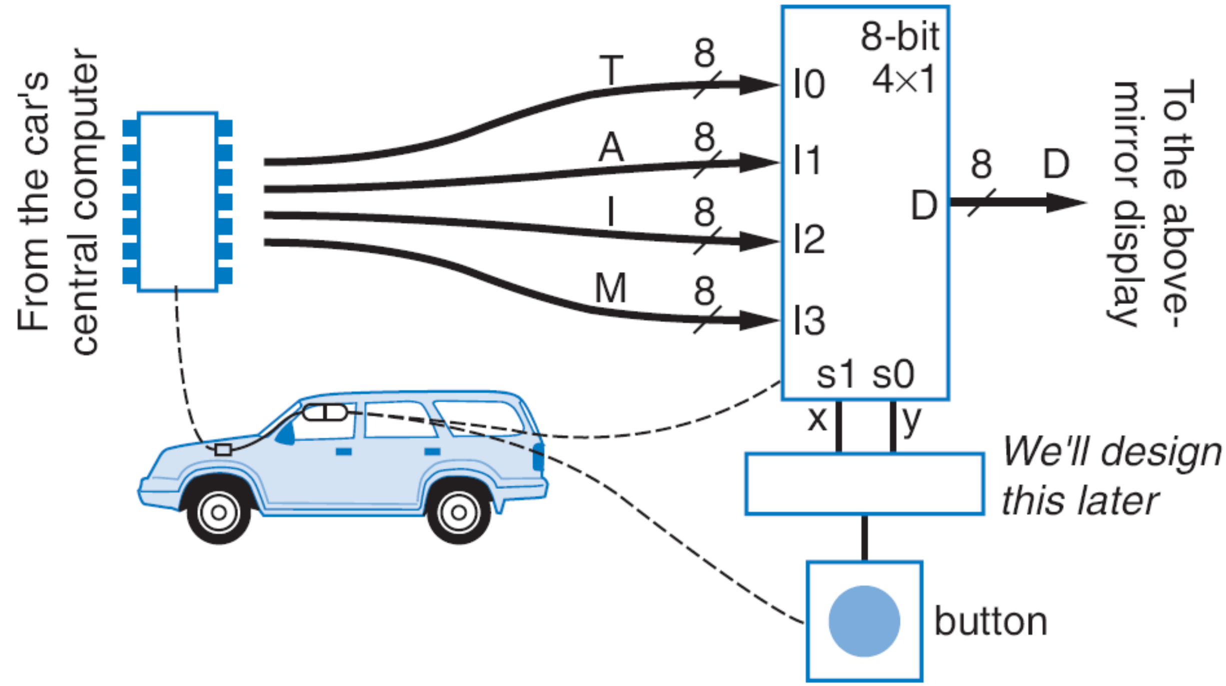
Verilog Testbench for Register

```
module Test_Banch;
  parameter half_period = 50;
  parameter reg_size = 3;

  wire [reg_size-1:0] Q;
  reg  [reg_size-1:0] Din;
  reg                 clock, load;

  Reg_N_bits #(reg_size) UUT (Q, Din, clock, load);

  initial begin
    #0   clock = 0; Din = 0;
    #200 Din = 5; load = 1;
    #100 load = 0;
    #200 Din = 3;
    #200 load = 1;
  end

  always #half_period clock = ~clock;

  initial #1000 $stop;
endmodule
```

Timing Diagram from Simulation

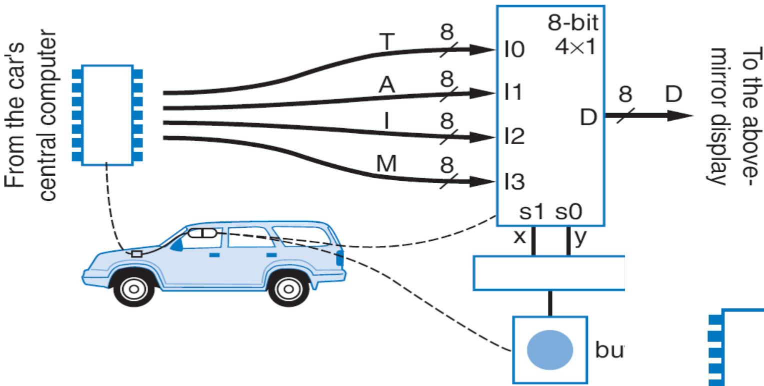


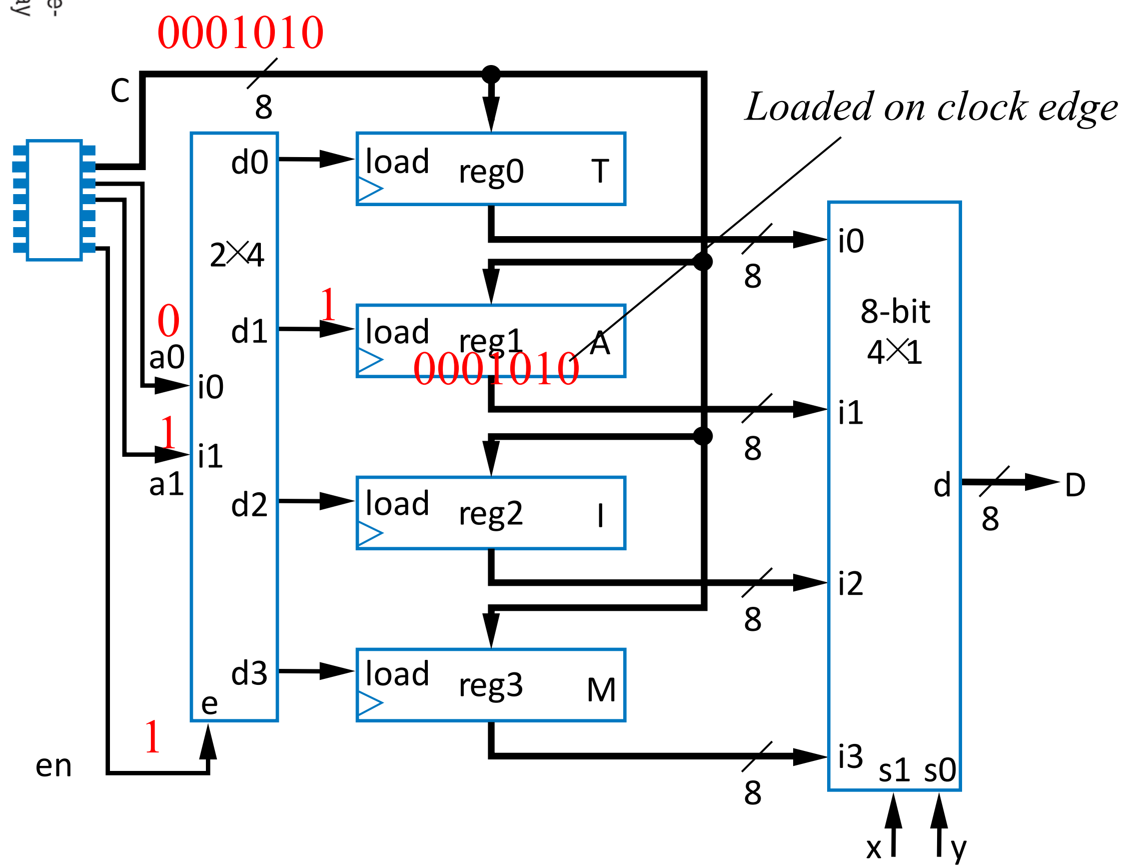
N-bit Mux Example



- Four possible display items
  - Temperature (T), Average miles-per-gallon (A), Instantaneous mpg (I), and Miles remaining (M) -- each is 8-bits wide
  - Choose which to display using two inputs x and y
  - Use 8-bit 4x1 mux

Register Example: Above-Mirror Display

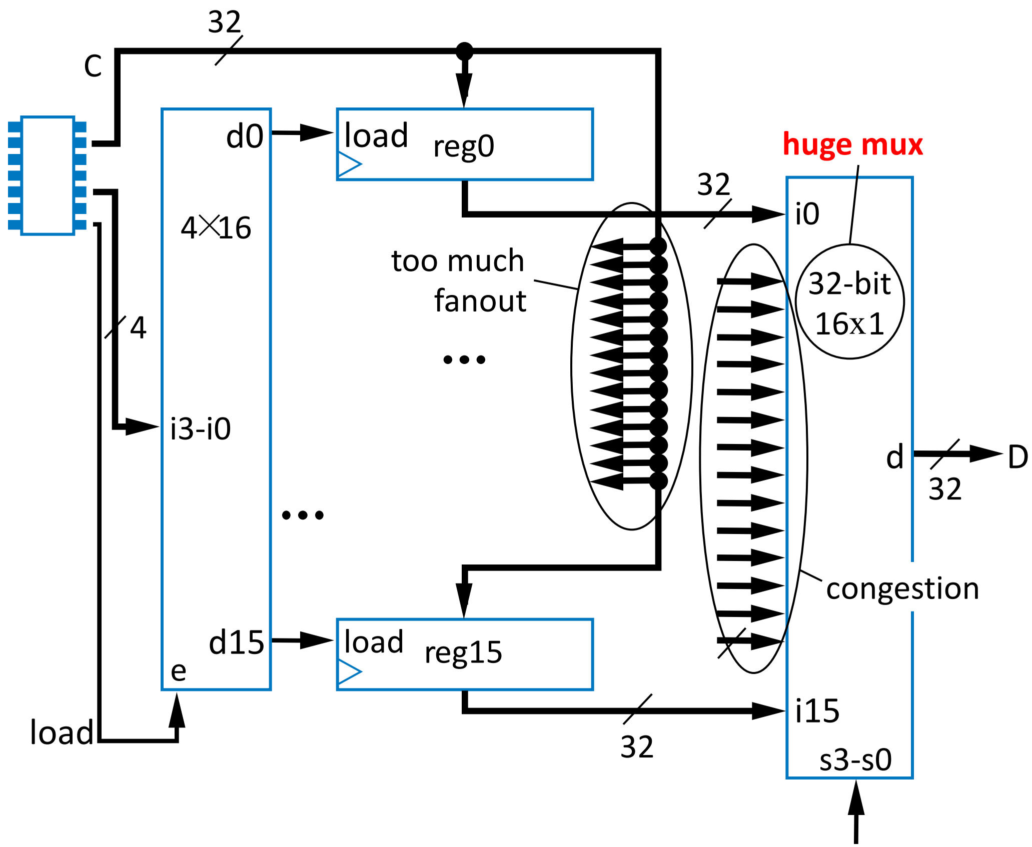




- Four simultaneous values from car's computer: Temperature, Average mpg, Instantaneous mpg, Millage remaining
- To reduce wires: Computer writes only 1 value at a time, loads into one of four registers

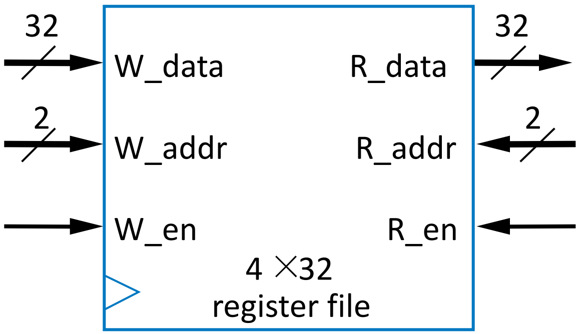
Register Files

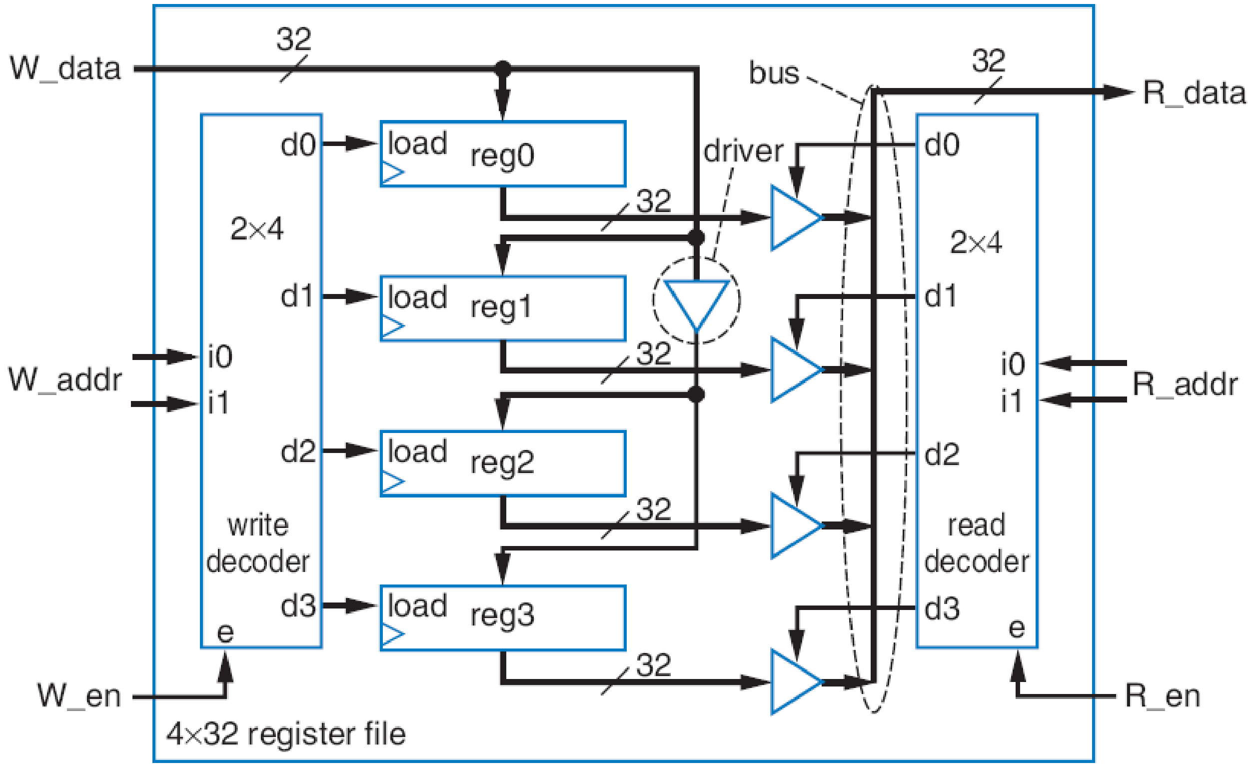
- ***MxN register file*** provides efficient access to M N-bit-wide registers
  - If we have many registers but only need access one or two at a time, a register file is more efficient
  - Ex: Above mirror display (earlier example), but this time having 16 32-bit registers
    - Too many wires, and big mux is too slow



Register File

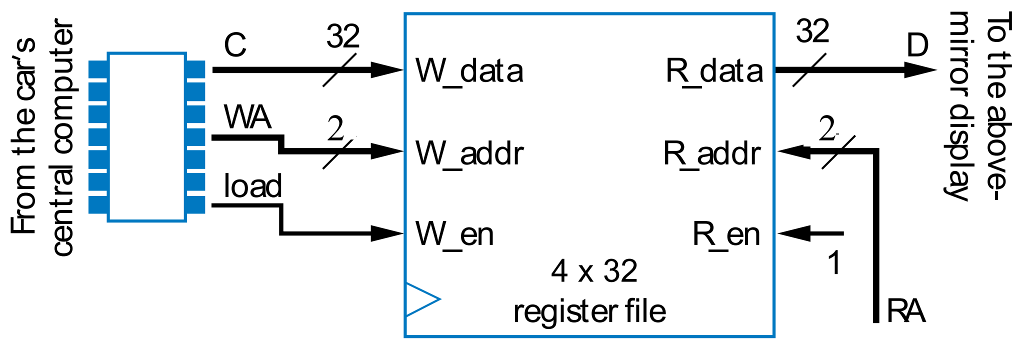
- Packing the registers together





Register-File Example: Above-Mirror Display

- Four 32-bit registers that can be written by car's computer, and displayed
  - Use 4x32 register file
  - Simple, elegant design
- Register file hides complexity internally
  - And because only one register needs to be written and/or read at a time, internal design is simple

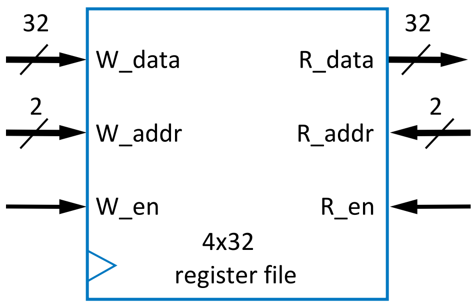


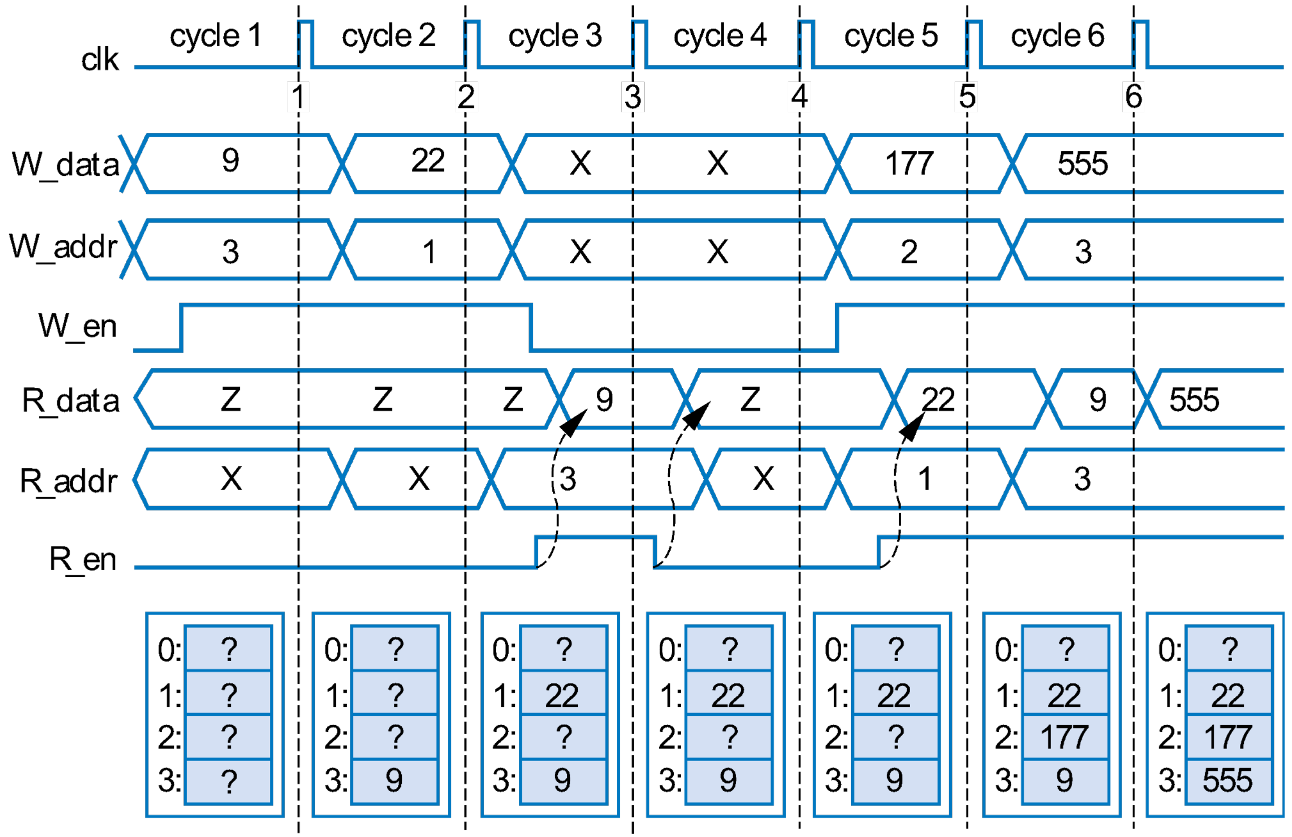
Verilog Modeling of Clocked Register File

```
module memory (R_data, W_data, W_addr, R_addr, W_en, R_en, clock);
  parameter width = 32;
  parameter addr_width = 2;
  parameter number = 2**addr_width;

  output [width-1:0]      R_data;
  input  [width-1:0]      W_data;
  input  [addr_width-1:0] W_addr, R_addr;
  input                   W_en, R_en, clock;

  reg [width-1:0] R_data;
  reg [width-1:0] memory [number-1:0];

  always @(posedge clock) begin
    R_data = 'bz;
    if (W_en)      memory[W_addr] = W_data;
    else if (R_en) R_data = memory[R_addr]; //R_data will be register
  end
endmodule
```

Register File Timing Diagram

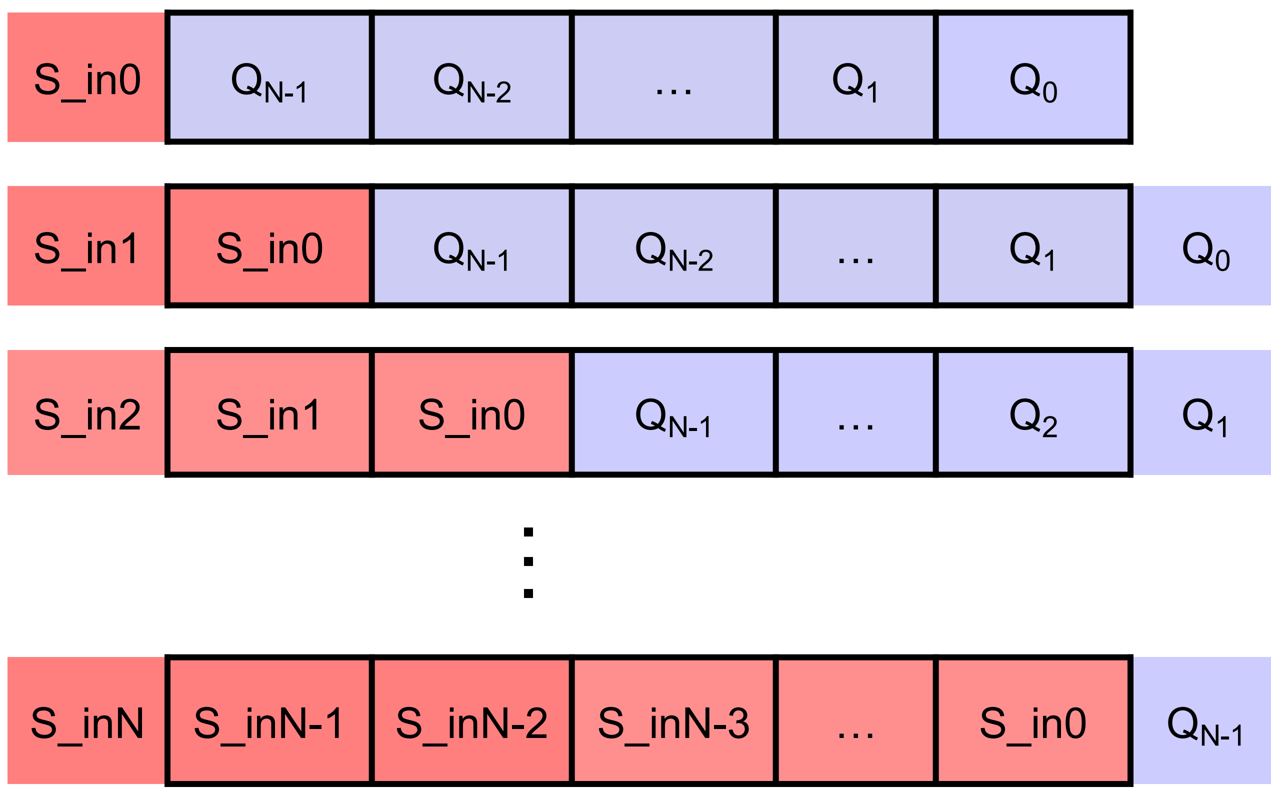
- May be designed to write one register and read one register in one clock cycle
  - May be same register





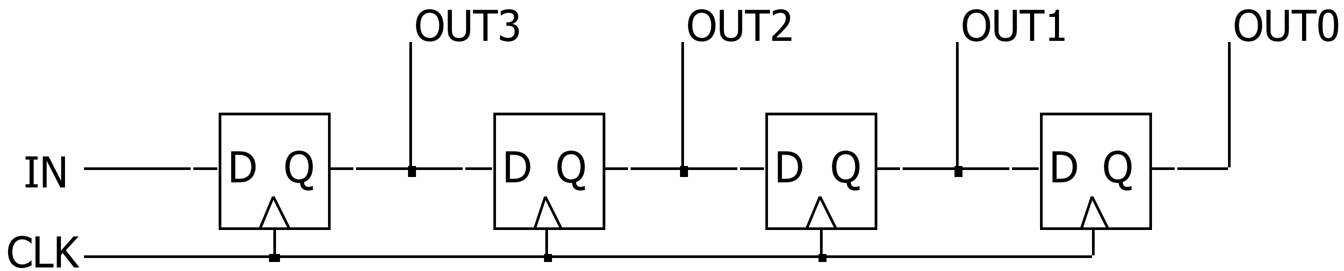
Shift Register

- One type of register
  - Stores binary data
  - Stored data can be shifted right (MSB >> LSM) or left (MSB << LSB)
  - Example: Shift right per clock edge



Shift Register

- Implementation:
  - Connect Q output of one flip flop to the D input of the next flip flop
  - 4-bit shift register



	IN	OUT(3:0)
Initial value:	0	0110
rising edge:	0	0011
rising edge:	0	0001
rising edge:	0	0000
rising edge:	1	1000
rising edge:	0	0100

Verilog Modeling of Shift Registers

```
module Shift_Reg (Q, Dout, Din, clock);
  input clock, Din;
  output Dout;
  output [3:0] Q;

  reg [3:0] Q;

  always @ (posedge clock)
  begin
    Q[2:0] <= Q[3:1];
    Q[3]   <= Din;
  end

  assign Dout = Q[0];
endmodule
```

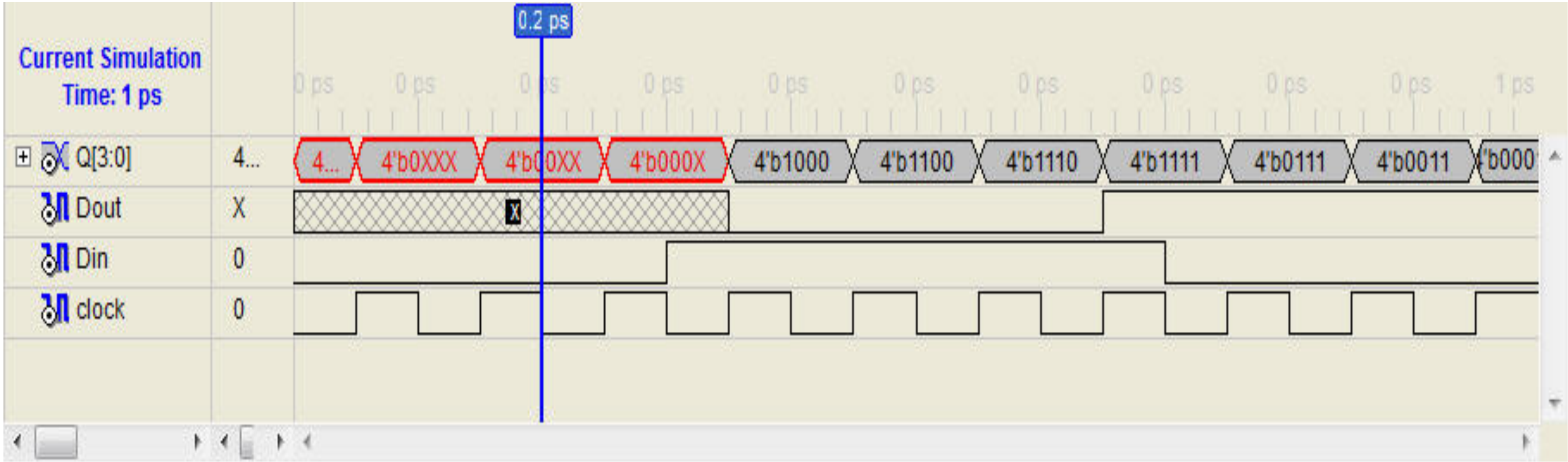
Verilog Testbench for Shift Registers

```
module Test_Banch;
  parameter half_period = 50;
  wire [3:0] Q;  wire Dout;  reg  Din, clock;

  Shift_Reg UUT (Q, Dout, Din, clock);

  initial begin
    $display("************************************************");
    $display("The textual simulation results:");
    $display("************************************************");
    $monitor($time,,"clock = %d  Din = %b  Q = %b  Dout = %d",
             clock, Din, Q, Dout);
  end

  initial begin
    #0   clock = 0; Din = 0;
    #300 Din = 1;
    #400 Din = 0;
  end

  always #half_period clock = ~clock;

  initial #1000 $stop;
endmodule
```

Textural Simulation Result

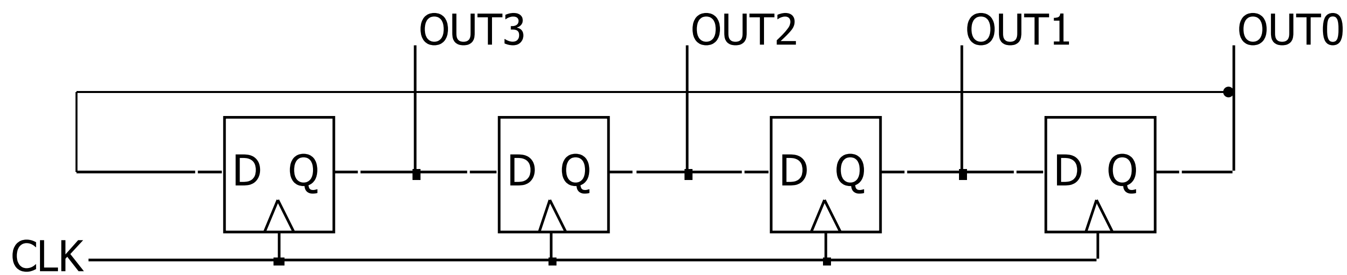
```
*********************************************************
The textual simulation results:
*********************************************************
          0 clock = 0  Din = 0  Q = xxxx  Dout = x
         50 clock = 1  Din = 0  Q = 0xxx  Dout = x
        100 clock = 0  Din = 0  Q = 0xxx  Dout = x
        150 clock = 1  Din = 0  Q = 00xx  Dout = x
        200 clock = 0  Din = 0  Q = 00xx  Dout = x
        250 clock = 1  Din = 0  Q = 000x  Dout = x
        300 clock = 0  Din = 1  Q = 000x  Dout = x
        350 clock = 1  Din = 1  Q = 1000  Dout = 0
        400 clock = 0  Din = 1  Q = 1000  Dout = 0
        450 clock = 1  Din = 1  Q = 1100  Dout = 0
        500 clock = 0  Din = 1  Q = 1100  Dout = 0
        550 clock = 1  Din = 1  Q = 1110  Dout = 0
        600 clock = 0  Din = 1  Q = 1110  Dout = 0
        650 clock = 1  Din = 1  Q = 1111  Dout = 1
        700 clock = 0  Din = 0  Q = 1111  Dout = 1
        750 clock = 1  Din = 0  Q = 0111  Dout = 1
        800 clock = 0  Din = 0  Q = 0111  Dout = 1
        850 clock = 1  Din = 0  Q = 0011  Dout = 1
        900 clock = 0  Din = 0  Q = 0011  Dout = 1
        950 clock = 1  Din = 0  Q = 0001  Dout = 1
Stopped at time : 1.000 ps : File "C:/Users/gzheng/test/tb.v" Line 29
```

Graphical Simulation Results



Rotate Register

- Implementation:
  - Connect Q output of one flip flop to the D input of the next flip flop
  - Connect Q output of the last flip flop to the D input of the first flip flop
  - 4-bit rotate register



	OUT(3:0)
Initial value:	0110
rising edge:	0011
rising edge:	1001
rising edge:	1100
rising edge:	0110
rising edge:	0011

Verilog Modeling of Rotate Registers

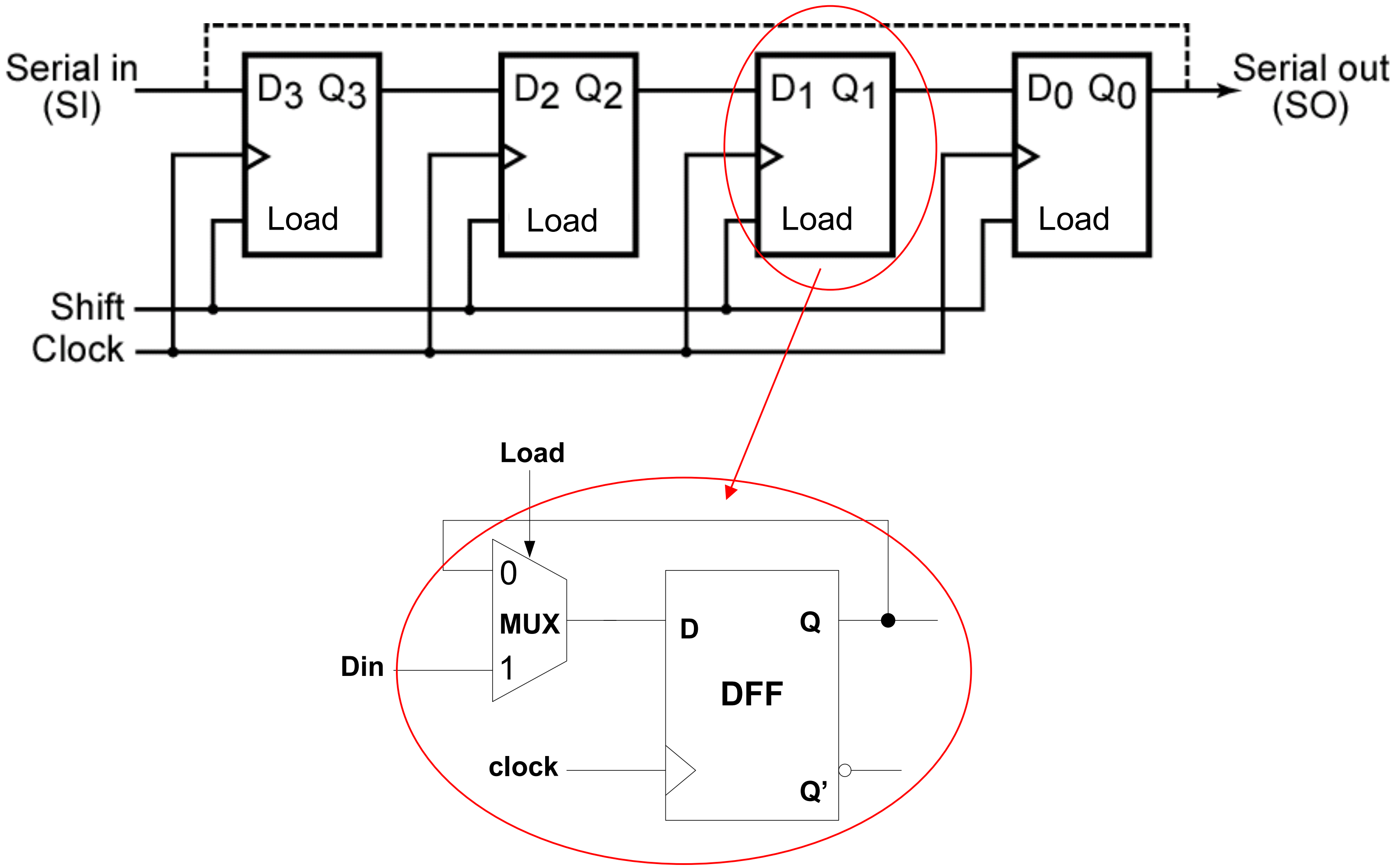
```
module Barral_Shift_Reg (Q, clock);
  input clock;
  output [3:0] Q;

  reg [3:0] Q;

  always @ (posedge clock)
  begin
    Q[2:0] <= Q[3:1];
    Q[3] <= Q[0];
  end

endmodule
```

Modified Rotate Registers

```
module Barral_Shift_Reg (Q, Din, clock, ld);
  input clock, ld, Din;
  output [3:0] Q;

  reg [3:0] Q;

  always @ (posedge clock)
  begin
    Q[2:0] <= Q[3:1];
    if (ld) Q[3] <= Din;
    else    Q[3] <= Q[0];
  end

endmodule
```

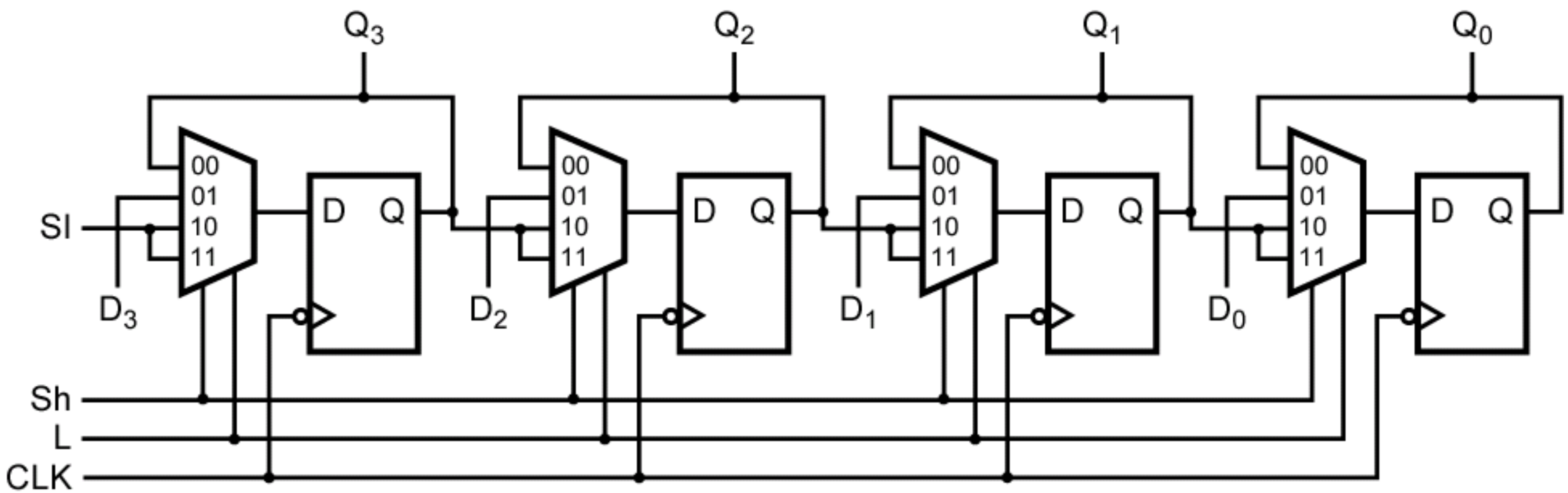
Shift/Rotate Register with Control Input

- Parallel Load control input may be used to hold the shifting



Universal Shift Register

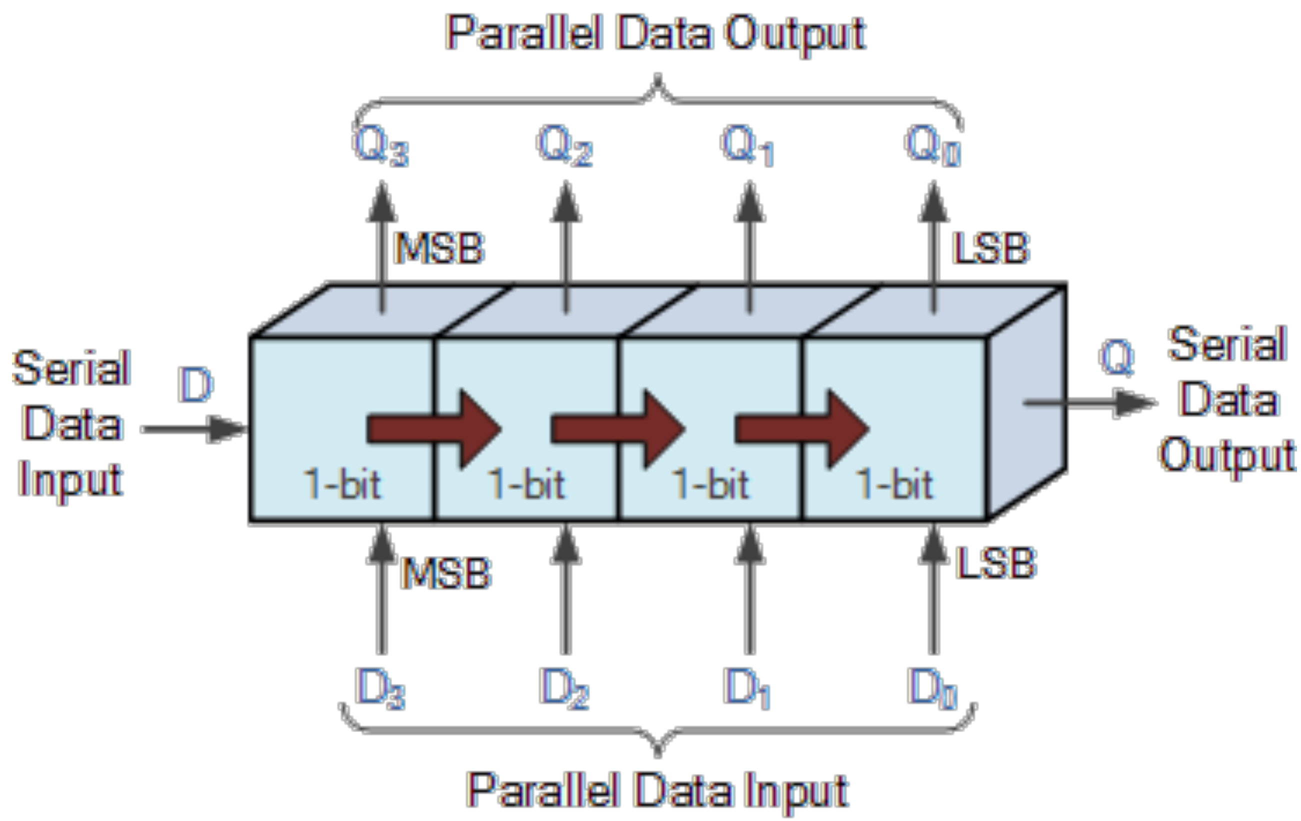
- Multi-functional shift register



- Functions
  - the 4^{th} input of MUX can be used for something else

Inputs		Action
Sh (Shift)	**L (Load)**	
0	**0**	**no change**
0	**1**	**load**
1	**X**	**Shift Right**

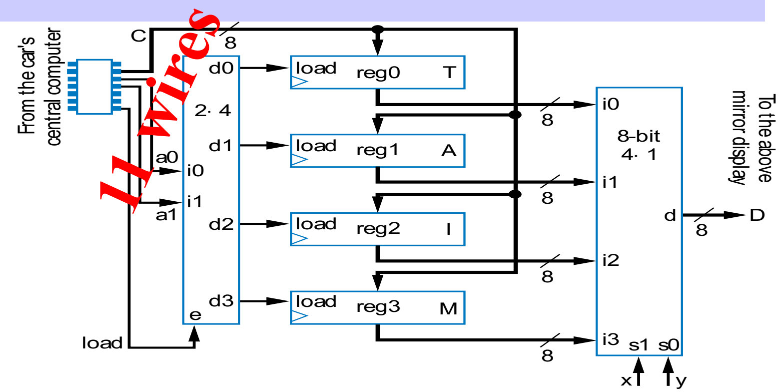
Universal Shift Register



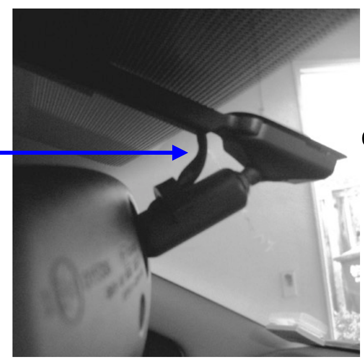
Source: www.electronics-tutorials.ws

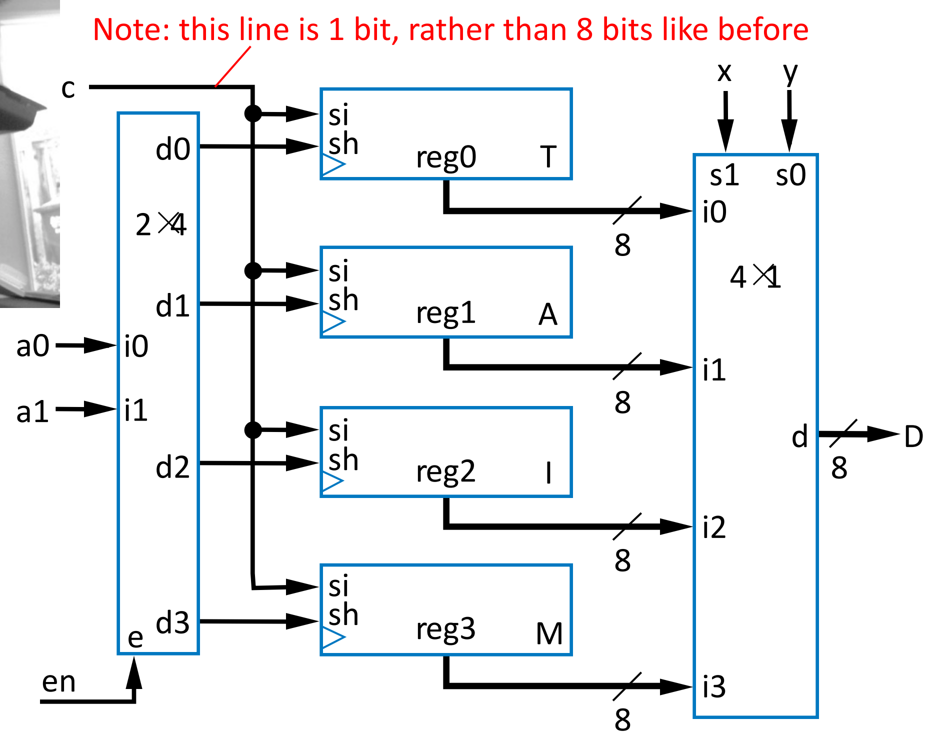
Shift Register Example: Above-Mirror Display

- Earlier example: 8 +2+1 = 11wires from computer to registers



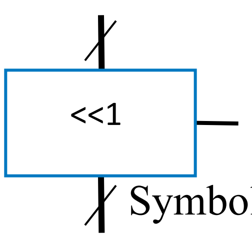
- Use shift registers
  - Wires: 1+2+1=4
  - Computer sends one value at a time, one bit per clock cycle



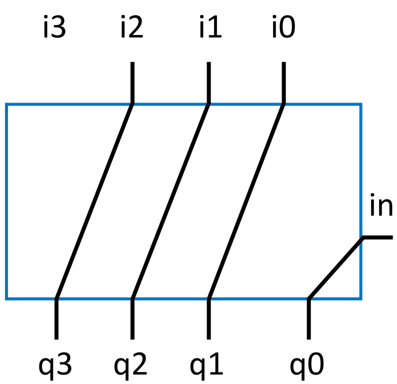


Shifters (Not Shift Register)

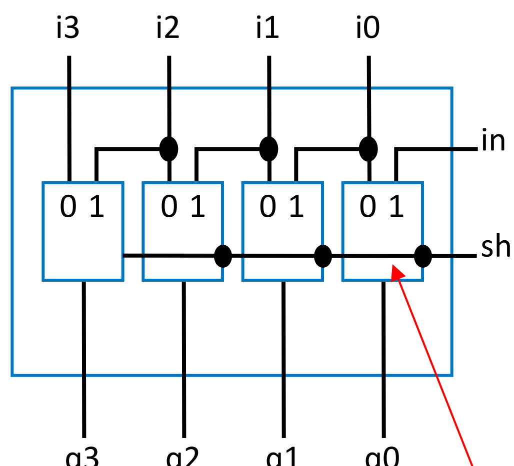
- Combinational Datapath component
- Shifting (e.g., left shifting 0011 yields 0110) useful for:
  - Manipulating bits
  - Shift left once is same as multiplying by 2 (0011 (3) becomes 0110 (6))
  - Shift right once same as dividing by 2



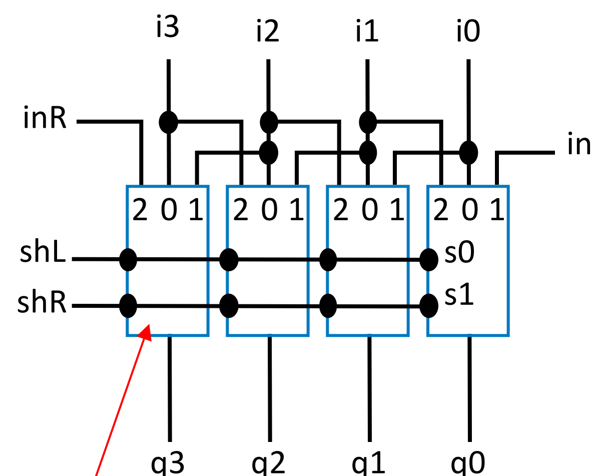
Symbol



Left shifter



Shifter with left shift or no shift



Shifter with left shift, right shift, and no shift

MUX

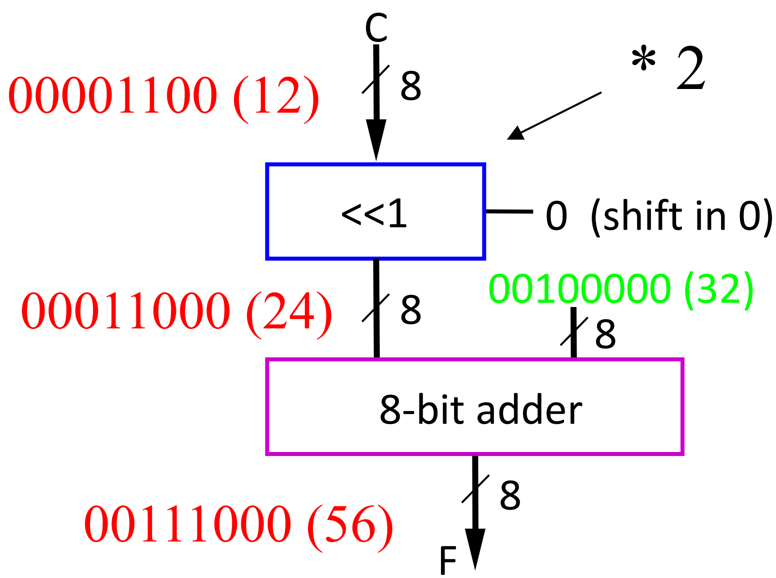
(

Verilog Modeling of Shifter

```
module Shifter_0_or_1 (Q, I, in, sh);
  parameter size = 4;
  input in, sh;
  input [size-1:0] I;
  output [size-1:0] Q;

  reg [size-1:0] Q;

  always @ (sh, in, I)
  begin
    if (sh) begin Q[size-1:1] = I[size-2:0];
                  Q[0]        = in;
             end
    else Q = I;
  end

endmodule
```

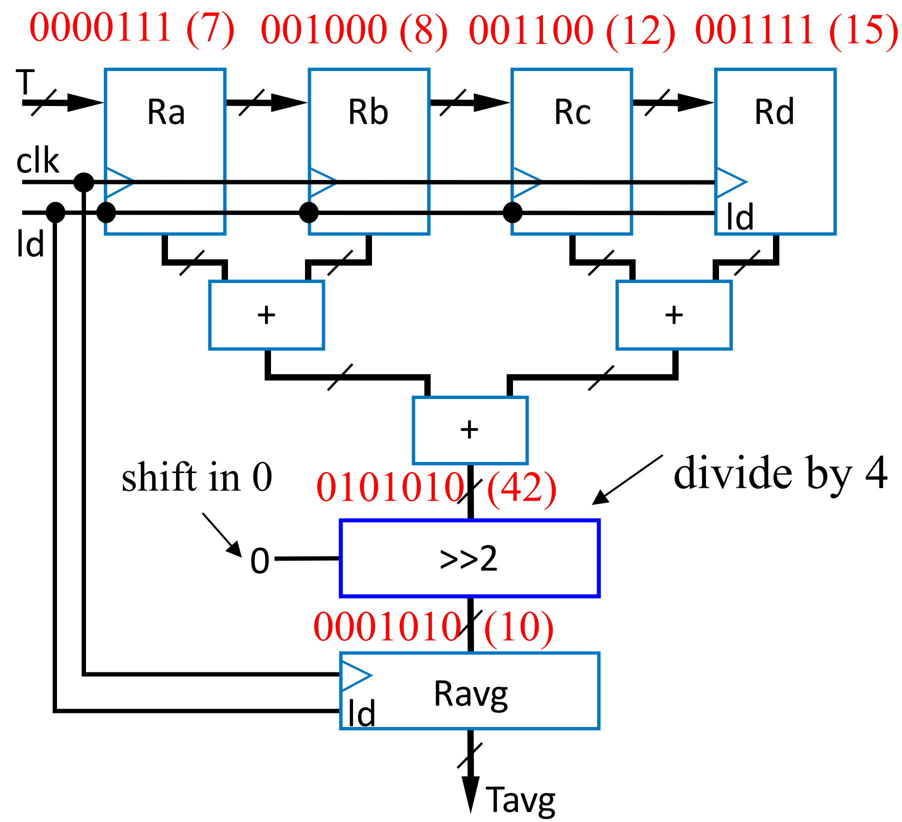
Shifter Example: Approximate Celsius to Fahrenheit Converter

- Convert 8-bit Celsius input to 8-bit Fahrenheit output
  - F = C * 9/5 + 32
  - Approximate: F = C*2 + 32
  - Use left shift: F = left_shift(C) + 32



Shifter Example: Temperature Averager

- Four registers storing a history of temperatures
- Want to output the average of those temperatures
- Add, then divide by four
  - Same as shift right by 2
  - Use three adders, and right shift by two



Bigger Shifter

- A shifter that can shift by any amount
  - But shifting too many bits in one shifter could require design with too many wires
- More elegant design
  - Chain shifters
  - E.g.: Can achieve any shift of 0..7 bits by enabling the correct combination of 4, 2, 1 bit shifters