ECE 310
Lab 4 Report: 8-Bit Universal Shifter

Ohm Patel

October 5, 2025

Abstract

This lab focused on designing and implementing an 8-bit universal shifter in Verilog using dataflow modeling. The objective was to create a single module capable of performing left and right shifts, rotations, and a hold operation under control of a 3-bit opcode. The only behavioral modeling permitted was for the D flip-flop (DFF) used in the register to store the output. The design was verified using a comprehensive testbench that evaluated each operation under a variety of representative and edge-case input patterns. Simulation results confirmed that the shifter functioned correctly across all cases.

1 Introduction, Background, and Theory

Shift registers are fundamental components in digital systems used for data storage, transfer, and manipulation. A universal shifter extends the capability of a basic shift register by allowing variable shifting and rotation operations in both directions. In this lab, an 8-bit shifter was implemented that supports the following operations:

Opcode	Operation Description
000	Shift left by 1 bit
001	Shift left by 2 bits
010	Logical shift right by 1 bit
011	Logical shift right by 2 bits
100	Rotate left by 1 bit
101	Rotate right by 1 bit
110	Hold (no change)
111	Load zeros (default/invalid case)

The output from the selected operation is stored in an 8-bit register constructed using D flip-flops. The design uses dataflow modeling for all combinational logic, while sequential storage (registering) is handled using behavioral DFFs triggered on the clock's positive edge. An additional control signal, `capture`, allows the circuit to hold its previous output when set high (capture is active-low).

2 Design Implementation

This section explains how the design maps to hardware and includes a block diagram showing the dataflow.

2.1 Block diagram

- An 8-bit input `d_in` feeding a combinational block that implements the 7 operations (shift/rotate/hold/zero).
- An 8-bit multiplexer (8-to-1) selected by the 3-bit opcode `op`.
- Capture logic that controls whether `regInput` receives the multiplexer output or remains the previous `d_out`.
- Eight D flip-flops (one per bit) clocked by `clock`, producing the registered output `d_out`.

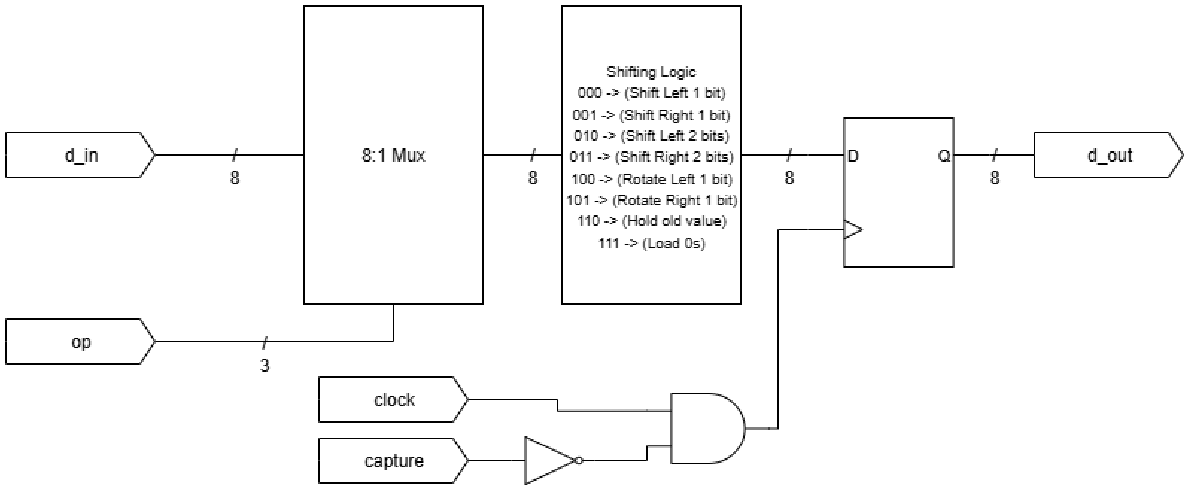
d_in
8
8:1 Mux
8
Shifting Logic
000 -> (Shift Left 1 bit)
001 -> (Shift Right 1 bit)
010 -> (Shift Left 2 bits)
011 -> (Shift Right 2 bits)
100 -> (Rotate Left 1 bit)
101 -> (Rotate Right 1 bit)
110 -> (Hold old value)
111 -> (Load 0s)
8
D
Q
8
d_out
op
3
clock
capture

Figure 1: Block diagram of the 8-bit universal shifter.

2.2 Hardware mapping and justification

- **Opcode width:** A 3-bit opcode (`op[2:0]`) is used to represent 8 possible choices (0–7). The lab requires 7 defined operations (0–6) and a default/invalid case; therefore $n = 3$ is the minimum width that covers all values.
- **Multiplexer sizing:** An 8-to-1 multiplexer (width = 8 bits) selects the resulting 8-bit value from the chosen operation. Each operation produces an 8-bit vector; thus the MUX is 8-bit wide.
- **Capture logic:** The capture signal is active-low. When `capture=0` the register should load the new value from the mux; when `capture=1` the register holds its current value. This is implemented by computing:

  $$\texttt{regInput} = \sim \texttt{capture} \; ? \; \texttt{muxOut} : \texttt{d\_out}$$

  which meets the lab requirement.
- **Sequential elements:** Eight DFFs are used for storage; they are the only allowed behavioral elements. Each DFF samples its input on the positive edge of `clock`.

3 Implementation (Verilog)

The code below shows the D flip-flop module (behavioral, nonblocking assignment), the shifter module (dataflow), and a compact listing of the test vectors used in the testbench.

3.1 D Flip-Flop

Listing 1: D Flip-Flop used in the 8-bit register.

```
module dff(output reg Q, input D, clock);
always @(posedge clock)
    Q <= D;
endmodule
```

3.2 Shifter (dataflow)

Listing 2: 8-bit universal shifter (dataflow model)

```
module shifter_8bit(output [7:0] d_out, input [7:0] d_in, input [2:0] op, input
    capture, clock);

wire [7:0] muxOut =
    (op == 3'b000) ? {d_in[6:0], 1'b0} :     // Shift left 1
    (op == 3'b001) ? {d_in[5:0], 2'b00} :    // Shift left 2
    (op == 3'b010) ? {1'b0, d_in[7:1]} :     // Shift right 1
    (op == 3'b011) ? {2'b00, d_in[7:2]} :    // Shift right 2
    (op == 3'b100) ? {d_in[6:0], d_in[7]} : // Rotate left 1
    (op == 3'b101) ? {d_in[0], d_in[7:1]} : // Rotate right 1
    (op == 3'b110) ? d_out : 8'b00000000;    // Hold or zero
wire [7:0] regInput = (~capture) ? muxOut : d_out;

dff bit0 (d_out[0], regInput[0], clock);
dff bit1 (d_out[1], regInput[1], clock);
dff bit2 (d_out[2], regInput[2], clock);
dff bit3 (d_out[3], regInput[3], clock);
dff bit4 (d_out[4], regInput[4], clock);
dff bit5 (d_out[5], regInput[5], clock);
dff bit6 (d_out[6], regInput[6], clock);
dff bit7 (d_out[7], regInput[7], clock);
endmodule
```

3.3 Testbench (compact listing of the test vectors)

Below is the compact version of the testbench input sequence used in simulation. Each line is the vector applied; the testbench contains clock generation and timing control (not shown here) so each vector is sampled on a clock boundary in simulation.

Listing 3: Representative test cases for the 8-bit universal shifter (compact).

```
initial begin
    d_in = 8'b00000000; op = 3'b000; capture = 1'b0; // Zero input
    d_in = 8'b10110010; op = 3'b000; capture = 1'b0; // Shift left 1
    d_in = 8'b11001100; op = 3'b001; capture = 1'b0; // Shift left 2
    d_in = 8'b01111001; op = 3'b010; capture = 1'b0; // Shift right 1
    d_in = 8'b10000001; op = 3'b011; capture = 1'b0; // Shift right 2
    d_in = 8'b10101010; op = 3'b100; capture = 1'b0; // Rotate left 1
    d_in = 8'b11100001; op = 3'b101; capture = 1'b0; // Rotate right 1
    d_in = 8'b01010101; op = 3'b110; capture = 1'b0; // Hold (capture=0)
    d_in = 8'b01010101; op = 3'b000; capture = 1'b1; // Hold (capture=1)
    d_in = 8'b11111111; op = 3'b111; capture = 1'b0; // Invalid, load 0s
    d_in = 8'b00000001; op = 3'b000; capture = 1'b0; // Edge bit left
    d_in = 8'b10000000; op = 3'b011; capture = 1'b0; // Edge bit right
    d_in = 8'b01010101; op = 3'b100; capture = 1'b0; // Alternating rotate left
    d_in = 8'b10101010; op = 3'b101; capture = 1'b0; // Alternating rotate right
    d_in = 8'b11111111; op = 3'b010; capture = 1'b0; // All ones shift right 1
end
```

4 Test Case Explanation and Justification

This section explains why each test vector was chosen and what is expected. The table below lists the 15 test vectors, the combinational result (what the MUX should produce for the given input and opcode), and the rationale for why that vector is valuable for verification.

TC	d_in	op	Expected d_out (after load)	Reason / Coverage
1	00000000	000	00000000	Baseline zero input — ensures no spurious bits.
2	10110010	000	01100100	Generic pattern, shift-left 1 — tests bit movement and zero-fill.
3	11001100	001	00110000	Repeating pattern, shift-left 2 — multi-bit left shift check.
4	01111001	010	00111100	Random pattern, shift-right 1 — tests logical zero-fill on MSB.
5	10000001	011	00100000	Edge 1s, shift-right 2 — checks correct truncation and zero-fill.
6	10101010	100	01010101	Alternating bits, rotate-left 1 — checks wrap-around MSB→LSB.
7	11100001	101	11110000	Pattern with LSB=1, rotate-right 1 — checks wrap-around LSB→MSB.
8	01010101	110	(hold)	Hold operation (op=110) — should preserve previous register value.
9	capture=1	000	(hold)	Capture asserted (active-high) — hold regardless of op.
10	11111111	111	00000000	Invalid/default opcode → load zeros (explicit default).
11	00000001	000	00000010	Single LSB set — verifies carry into next bit on left shift.
12	10000000	011	00100000	Single MSB set — verifies correct logical right shift by 2.
13	01010101	100	10101010	Alternating pattern rotate-left — tests bit-order reversal.
14	10101010	101	01010101	Alternating pattern rotate-right — tests circular correctness.
15	11111111	010	01111111	All ones shift-right 1 — checks zeros are inserted in MSB.

Table 1: Test vectors, expected outputs, and justification. Expected values are the combinational MUX result and should be observed in the registered output after a clock edge when capture allows loading.

5 Results and Analysis

All test vectors were simulated in Vivado. The observed waveform matched the expected combinational results listed in Table 1. Representative screenshots of the waveform containing the transitions for shifts, rotates, and hold behavior are included below.

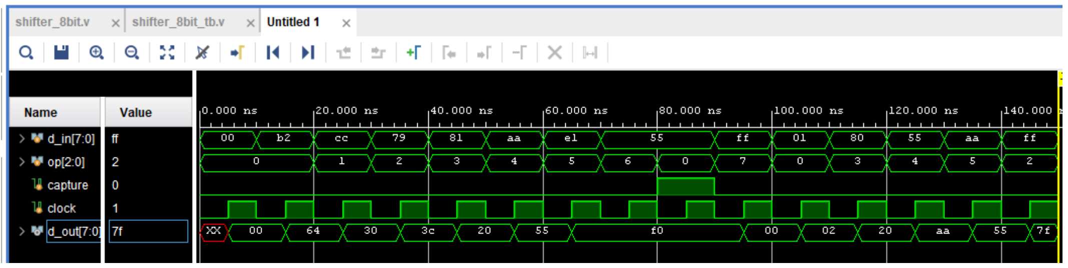

Figure 2: Simulation waveform of the 8-bit universal shifter showing representative test cases.

6 Conclusion

The 8-bit universal shifter was implemented using dataflow modeling for combinational logic and a behavioral D flip-flop for sequential storage, as required by the lab. The design supports left and right logical shifts (by 1 and 2), 1-bit rotates (left and right), a hold operation, and a default load-zero behavior for invalid opcodes. The provided test vectors exercise bit movement, wrap-around, zero-padding behavior, and the capture/hold functionality. Simulation results matched expected outputs for all cases.

This report was compiled using LaTeX.