

Lab 5

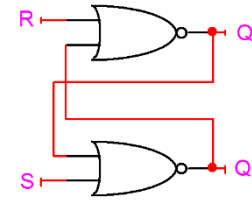
- **Part 1: Flip-Flop Design**
 - Design your very own D, T, SR, and JK flip-flops
- **Part 2: Modified Booth's FSM**
 - Implement a Mealy and a Moore Modified Booth's operation generator FSM
 - Implement a Modified Booth's controller
 - Integrate with Lab 2 datapath

Lab 5

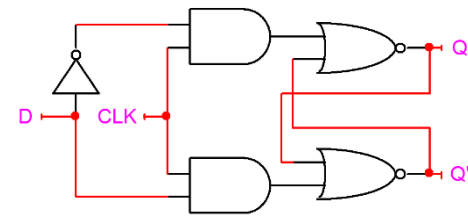
- **Part 3: Sequence Detector FSM**
 - Design three sequence detector FSMs:
 - one that detects “0101 $\rightarrow$ 0011”
 - one that detects “0100 $\rightarrow$ 1001”
 - one that detects both “0101 $\rightarrow$ 0011 or 1001” and “0100 $\rightarrow$ 1001 or 0011”
 - Implement the combined sequence detector
- **Part 4: Partially Specified FSM**
 - Reconstruct FSM from a partially specified automaton and output equation

Designing a D Flip-Flop

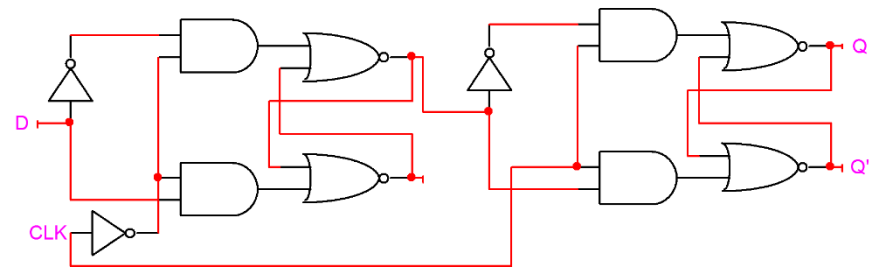
- **Start with SR-latch:**



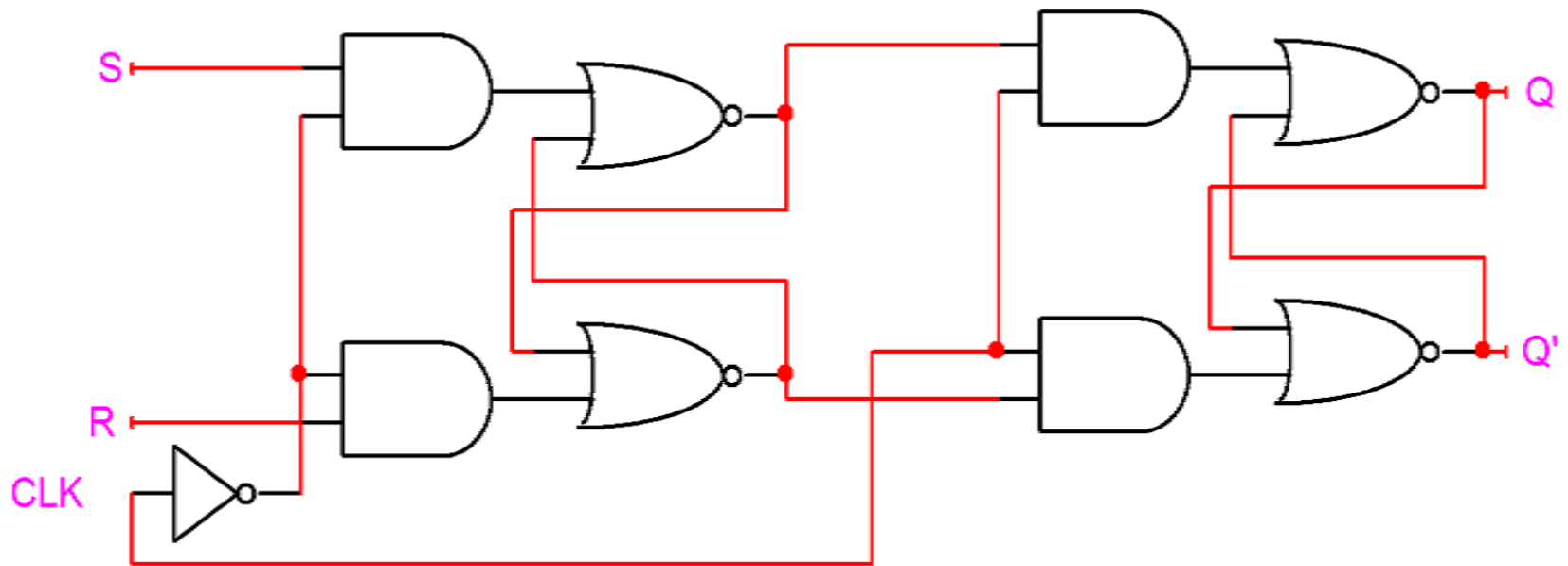
- **Convert to Gated D-latch:**



- **Arrange two Gated D-latches in Master-Slave Configuration:**



Designing an SR Flip-Flop



FSM Design Flow

1. Obtain the *state diagram* of the sequential circuit;
2. Minimize states if possible;
3. Encode states according to the encoding algorithm;
4. Generate *next state & output* tables based on state diagram and flip-flop's excitation table;
5. Derive the logic for next-states and outputs;
6. Implement the logic with flip-flops and combinational gates.

State Minimization

- **Two states are equivalent if:**
 - They have the same outputs on the same inputs
 - They transition to equivalent states on the same inputs
- **Recursive formulation can be traced through an implication table**

State Encoding

- **Minimum-Bit-Change Algorithm**
 - Assign state encodings such that the total number of bit transitions between all states is minimized.
 - Particularly suitable for SR flip-flops.

State Encoding

- **Prioritized Adjacency Algorithm**

- Assign adjacent encodings (codes that are 1 bit different from each other) to pairs with high priority
 - Priority 1: States that transition to the same next-state on the same input
 - Priority 2: States that are transitioned to from the same state on hamming adjacent inputs
 - Priority 3: States that produce the same outputs on the same inputs

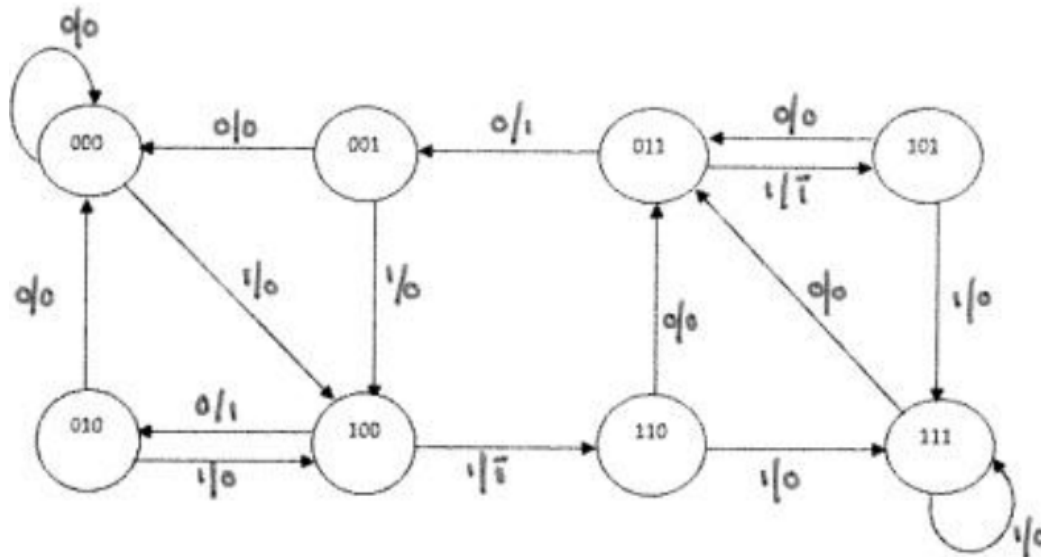
Part 1

- **Implement your own D, T, SR, and JK Flip-Flops**
 - Should use the master-slave latch architecture
 - Must contain asynchronous set and reset inputs

Part 2

- **Mealy Modified Booth's Operation Generator FSM:**

- Apply state minimization
- Implement using SR flip-flops and the minimum-bit-change heuristic



Part 2

- **Moore Modified Booth's Operation Generator FSM:**
 - Derive the FSM
 - Apply state minimization
 - Implement using T flip-flops and the prioritized adjacency heuristic

Part 2

- **Controller FSM:**
 - Minimizes to 4 states, with a counter
 - Skip over the Load cycle if a no-op occurs
 - Implement using JK flip-flops and the encoding heuristic of your choice

Part 2

- **Integration with Lab 2:**
 - Replace original controller with your new controller that skips unnecessary Load states
 - Flip-flop (FF) no longer needed
 - Integrate your Moore Operation Generator
 - Inverted clock signal to shift phase relative to controller
 - Lab 2 addition and subtraction happened asynchronously during Shift cycle; to emulate that the operation generator needs to determine the operation halfway through the Shift cycle

Part 2

- **Your task:**
 - Perform state minimization and implement the Mealy operation generator FSM using SR flip-flops and encode states using minimum-bit-change heuristic
 - Derive, minimize, and implement the Moore operation generator FSM using T flip-flops and encode states using prioritized adjacency heuristic
 - Implement the Controller FSM using JK flip-flops with Load cycle skipping
 - Integrate the Moore operation generator and the controller with your Lab 2 datapath

Part 3

- **Sequence detector for “0101 → 0011”**
 - Draw minimized state diagram
- **Sequence detector for “0100 → 1001”**
 - Draw minimized state diagram
- **Sequence detector for both “0101 → 0011 or 1001” and “0100 → 1001 or 0011”**
 - Minimize the number of states (no more than 8)
 - Implement using T flip-flops
 - Encode with prioritized adjacency heuristic

Part 4

- **Partially specified FSM with output equations**
 - Find what the state encodings are by using the partially specified FSM and output equations
 - Implement the FSM using JK flip-flops
 - Guess what the functionality may be

Output Equations:

$$O1 = Q0' + Q1$$

$$O0 = Q0 + Q1'$$

