ECE 385 – Digital Systems Laboratory

Lecture 2 –Lab 3

Zuofu Cheng, Chushan Li

FALL 2021

All information please find in Blackboard System

ECE ILLINOIS

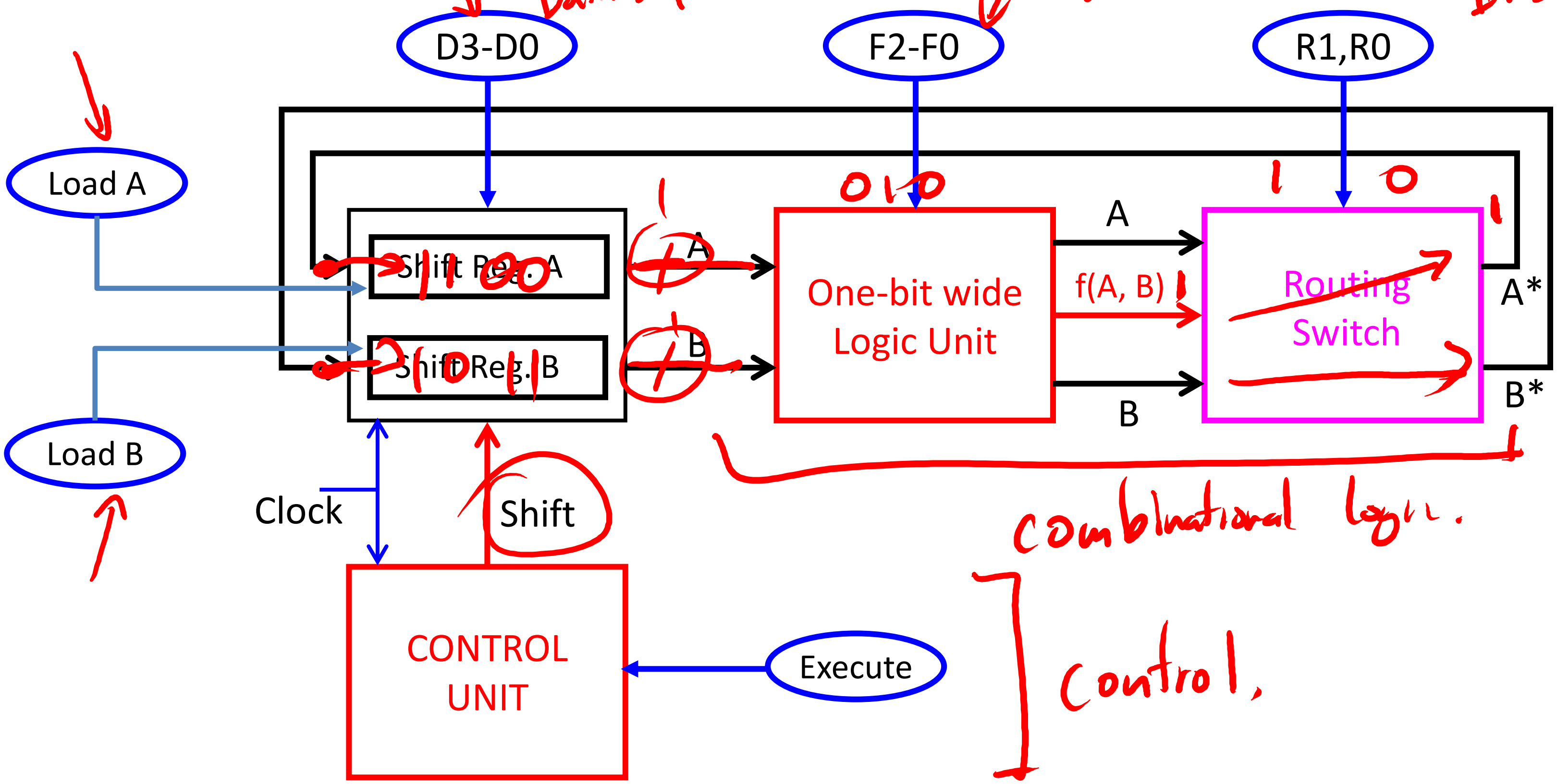
ILLINOIS

Experiment 3 – Introduction & Goals

- A Bit-Serial Logic Processor
  - Two 4-bit registers A, B
  - A or B also serve as a destination register.
    - E.g., AND A, B, A /* A AND B=>A */

D : 1100
Ld A ↑↓
D : 1011
Ld B ↑↓
F 010
R 10
routing select
EXEC ↑

Data Input
function select



datapath.
combinational logic.
control.

example

4bit 4bit,

$A \leftarrow A \oplus B$

1 bit 1

perform operation in serial.

Experiment 3 – Serial Logic Processor

- Design a 4-bit serial logic processor
- Performs 1 of 8 logical operations on 4-bit words (1 bit at a time, serially)
- All logical operations may be performed bit-wise with no additional storage
- Two 4-bit registers (A, B) which can serve as source/destination of each instruction
- Executes **single** instruction per EXECUTE signal

A : 1 1 0 0

B : 1 0 1 1

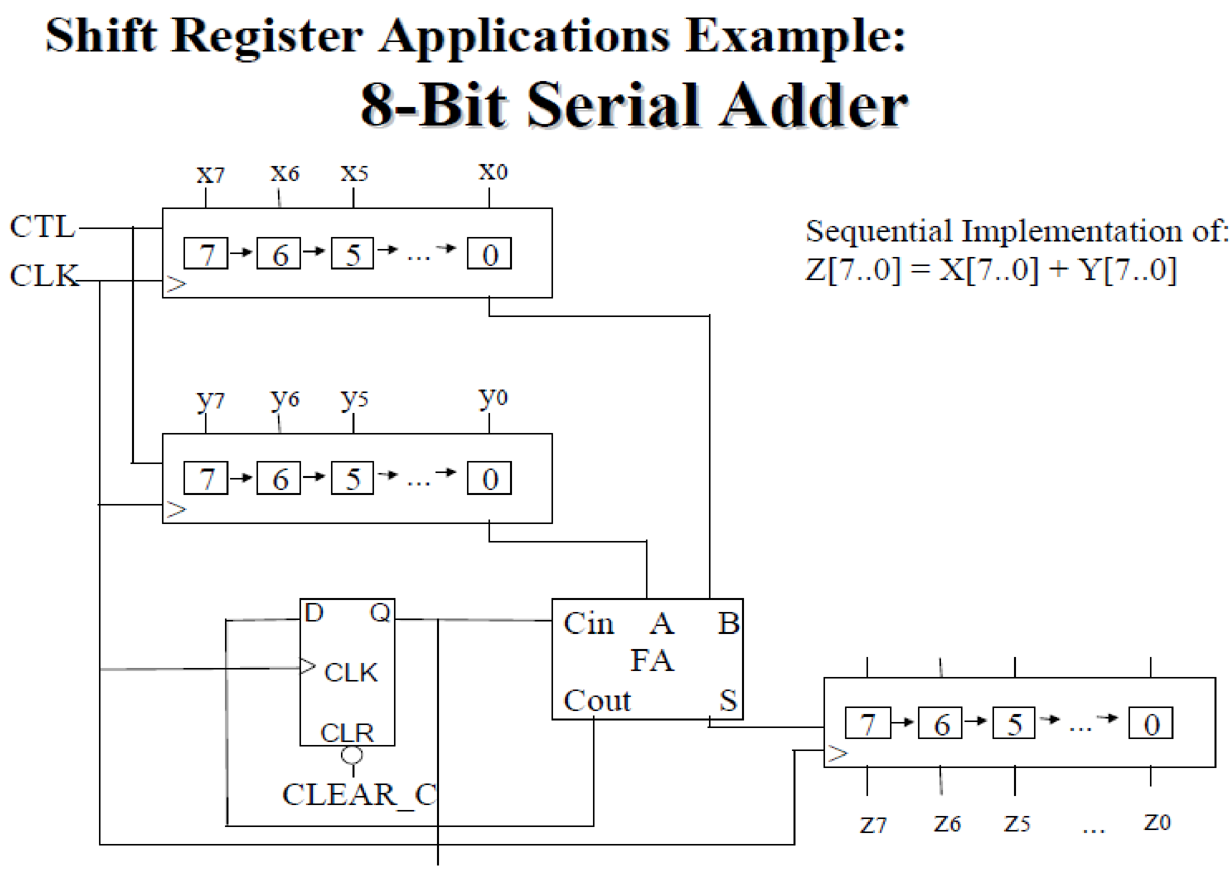
A xor B : 0 1 1 1

Experiment 3 – Demo Points

- Show correct loading of the A and B registers. (1 point)
- Show that the computation cycle is the right length. We will hit execute and hold it high. The shift registers should shift a total of 4 times. No more, no less. (1 point).
- Demonstrate the four routing operations. (1 point)
- Demonstrate the 8 functional operations. (1 point)
- Show that the computation cycle completes even if execute returns to 0 mid-computation. This will be clocked slowly (1 Hz), and execute will be switched high. As soon as it starts shifting, execute will be switched low. The circuit must complete its computation cycle as normal. (1 point)

Experiment 3 – Serial Processing

- All 8 operations (AND, OR, XOR, ‘1’, NAND, NOR, XNOR) can be done bitwise
- Example of 8-bit bitwise addition (we won’t need to implement this because it requires extra flip-flop for carry)

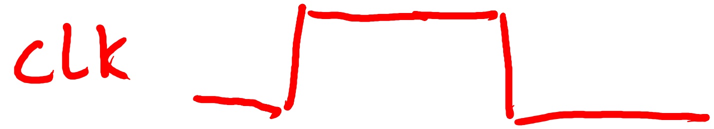


Experiment 3 – Instructions and Design

- 8 Logic functions need to be implemented in the computation unit; function output routed appropriately
- Instructions specified by inputs D[3:0], F[2:0], R[1,0], LoadA, LoadB
- Single instruction executed on EXEC (CPU halts after execution of one instruction until EXEC is high during RE)

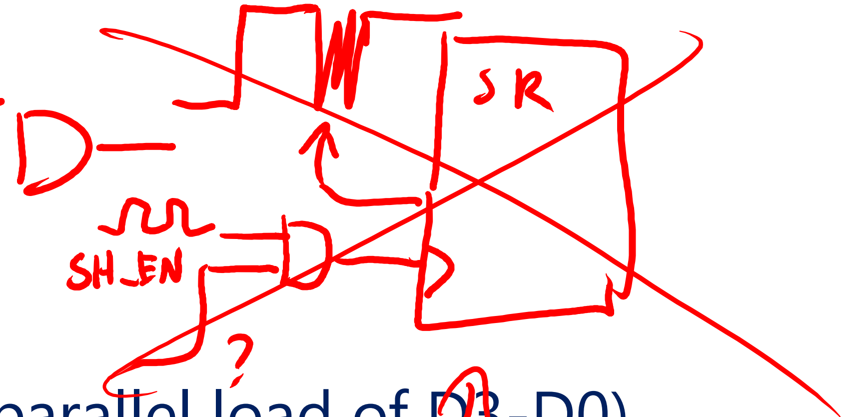
Function Selection Inputs			**Computation Unit Output**
F2	**F1**	**F0**	**f(A, B)**
0	0	0	A AND B
0	0	1	A OR B
0	1	0	A XOR B
0	1	1	1111
1	0	0	A NAND B
1	0	1	A NOR B
1	1	0	A XNOR B
1	1	1	0000

Routing Selection		**Router Output**	
R1	**R0**	**A’**	**B’**
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A



Experiment 3 – Register File

SH_EN

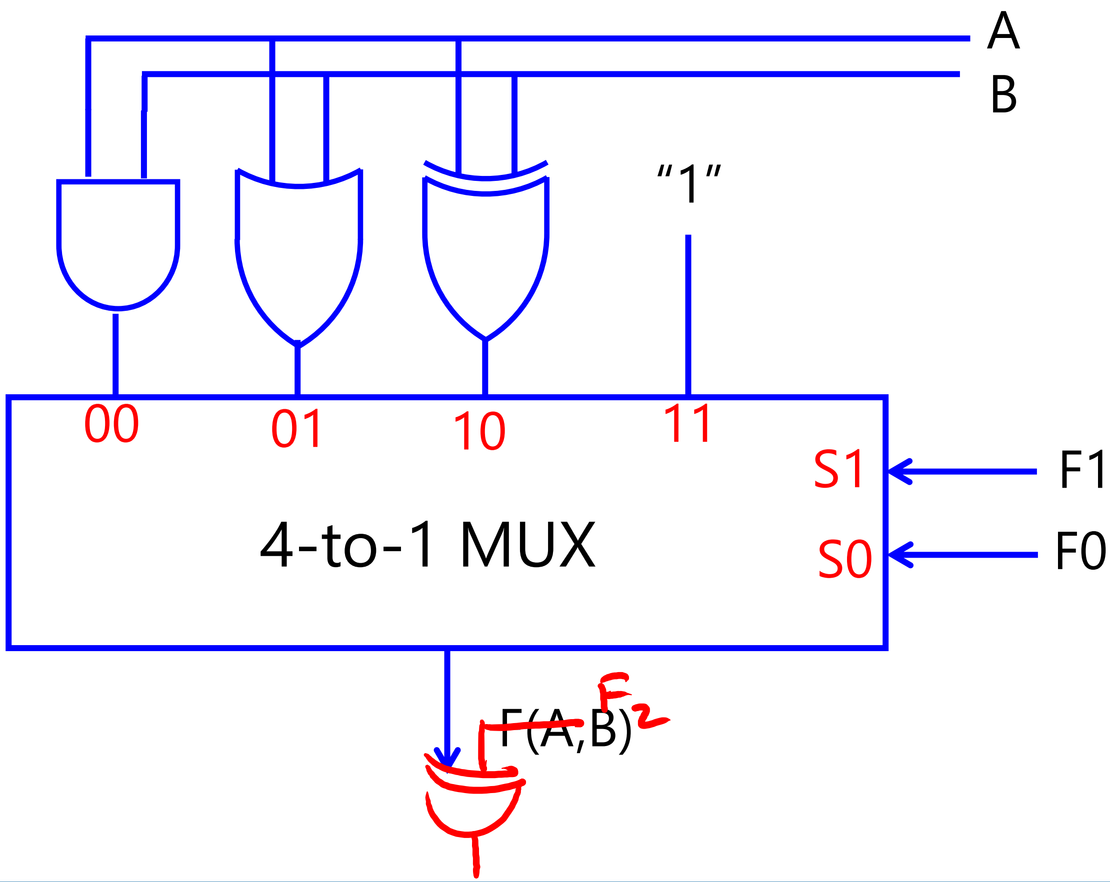


- Use two shift registers
  - Use the parallel in capability (parallel load of D3-D0)
  - Use the parallel out capability (for debugging and demo)
  - Shift out only when EXECUTE is flipped on (and only process a 4-bit word)
  - Shift in from the output of the router
  - Clock input from the FPGA switchbox (or switch for debugging)
    - We will test at 1 kHz
    - Do not gate clock. But do not continuously shift either, will need to have control unit control mode pin.

fully synchronous shift register
shift enable.

Experiment 3 – Logic Unit

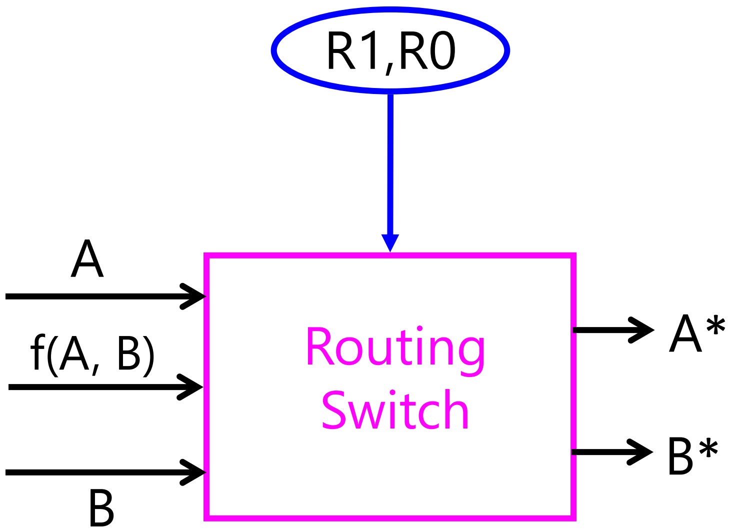
- Will need 8-1 MUX to implement 8 functions / or do you?
- Example (half of what you will need)



F_2	Function Select F1,F0		Function Output F(A,B)
0	0	0	A•B
0	0	1	A + B
0	1	0	A ⊕ B
0	1	1	"1"

Experiment 3 – Routing Unit

- Routes output of logic unit f(A,B) back to register file



Source Select R1,R0	Destination A*	Destination B*
0 0	A	B
0 1	A	f(A,B)
1 0	f(A,B)	B
1 1	B	A

This switch can be implemented using just two 4-to-1 Multiplexers!

Experiment 3 – Control Unit

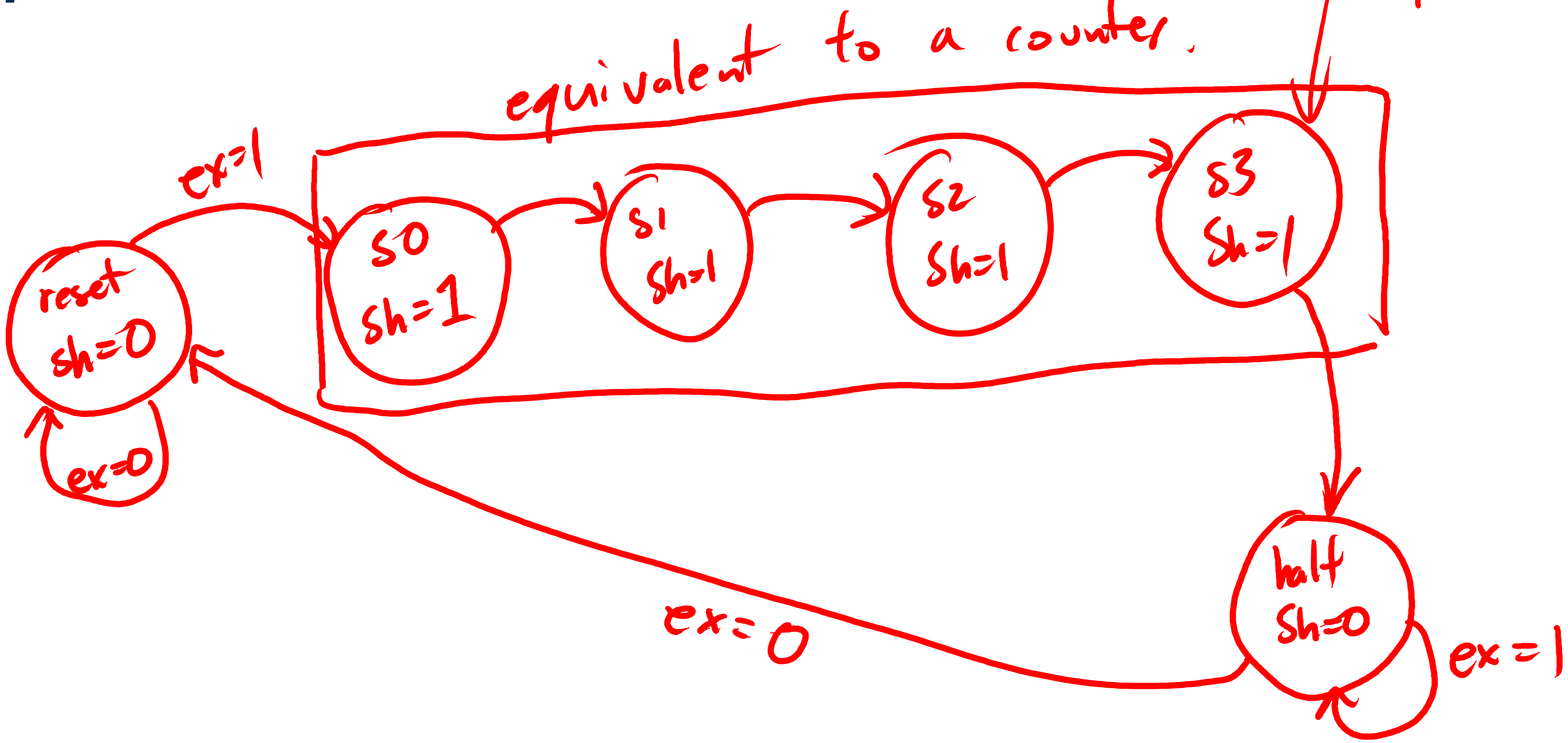
- This is the most challenging part of this design
  - When EXECUTE switch is ON, the controller starts the execute cycle and completes **exactly One Logic Instruction**. It then Halts. The Controller will not start another execution cycle until the EXECUTE switch is OFF and then ON again.
  - The controller also provides the Shift signals to the shift registers for an appropriate number of times to complete one execution cycle (4)
    - Should be designed as a state machine (Mealy or Moore)
    - Controller must execute to completion regardless of subsequent changes to EXECUTE once cycle started
    - Could enumerate each count as separate state, but would require more states

ex = execute
sh = shift

Moore. (aka ECE120 FSM)

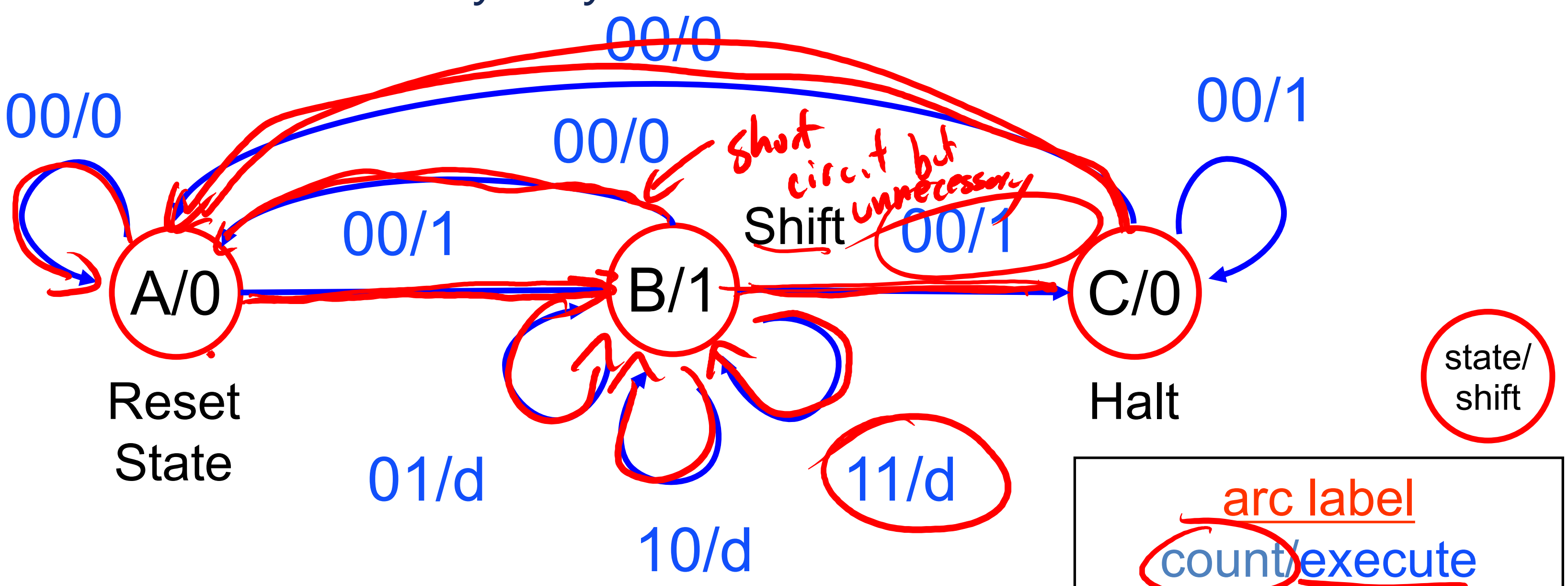
Experiment 3 – State Machine

done. with instruction



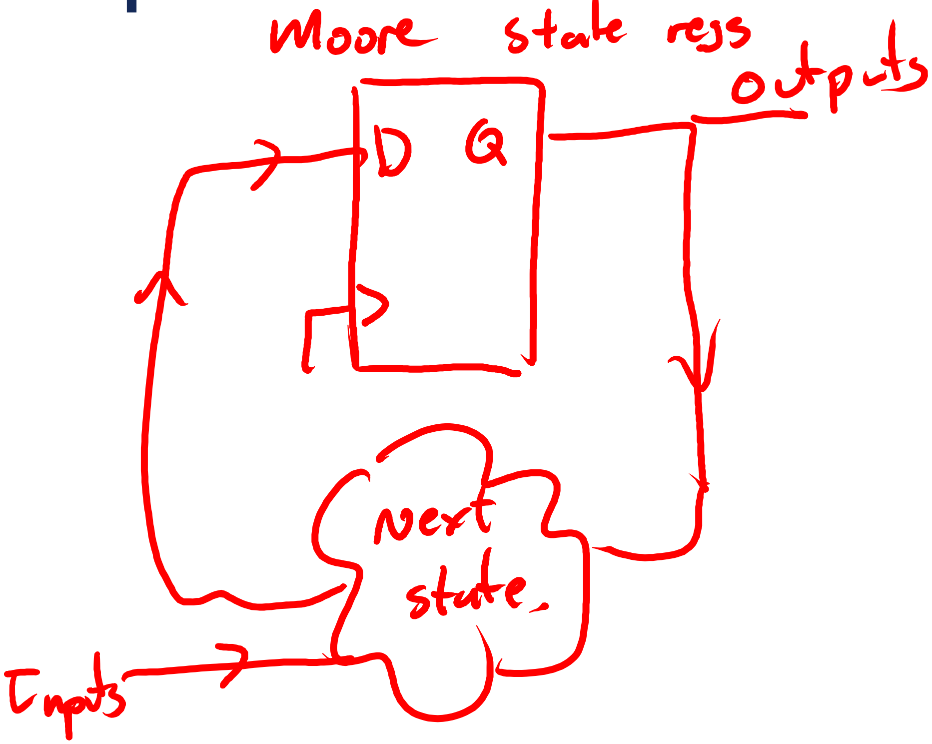
Experiment 3 – State Machine (Moore)

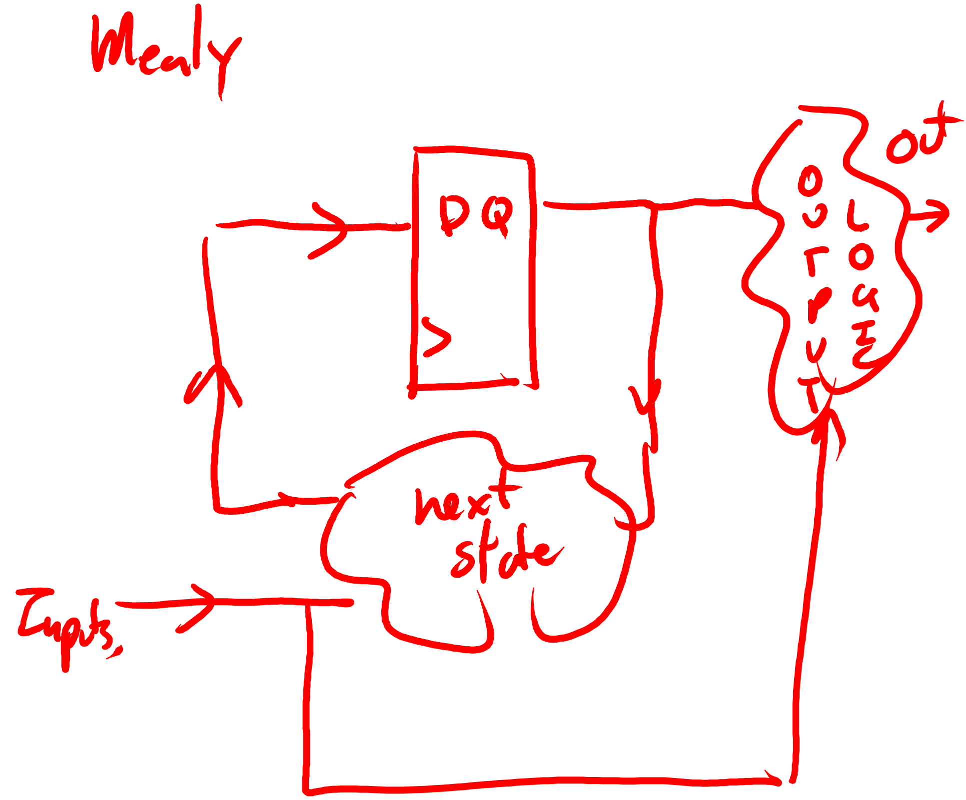
- Arcs are labeled with the counter outputs and the input Switch “EXECUTE” value
- Outputs are determined by only current state



You should include some capability to reset your controller (just involve running through once manually)

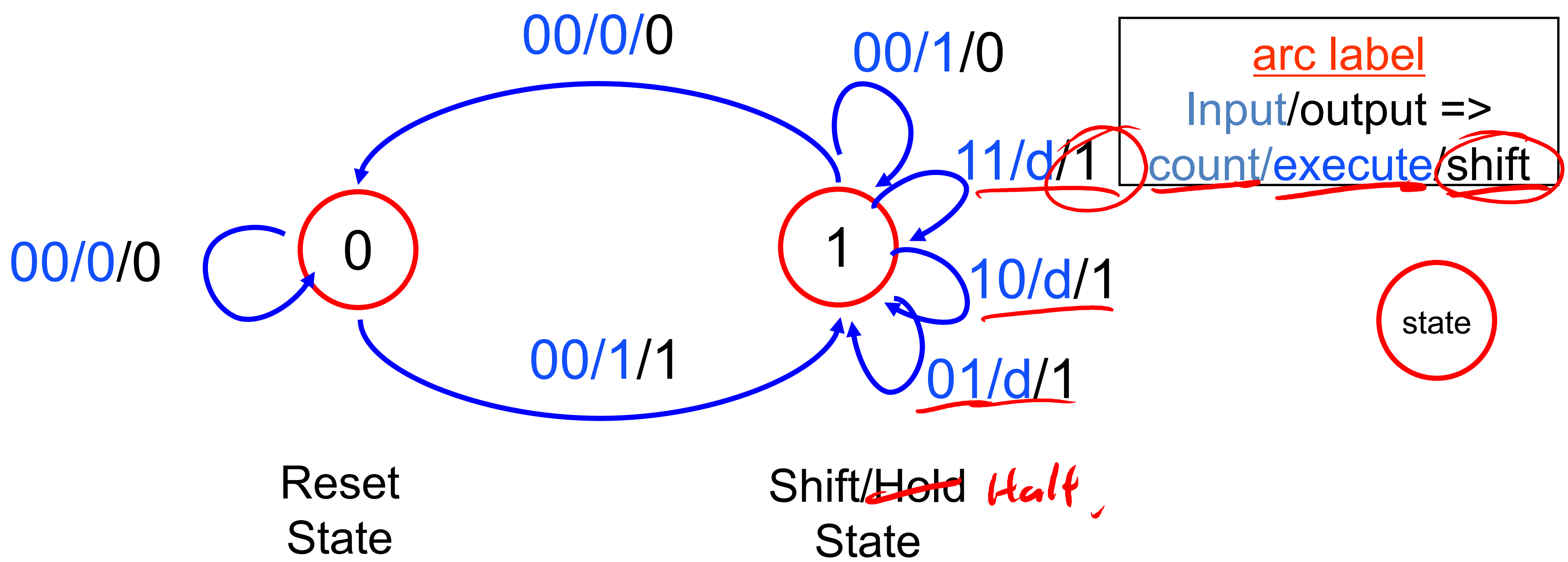
Experiment 3 – State Machine



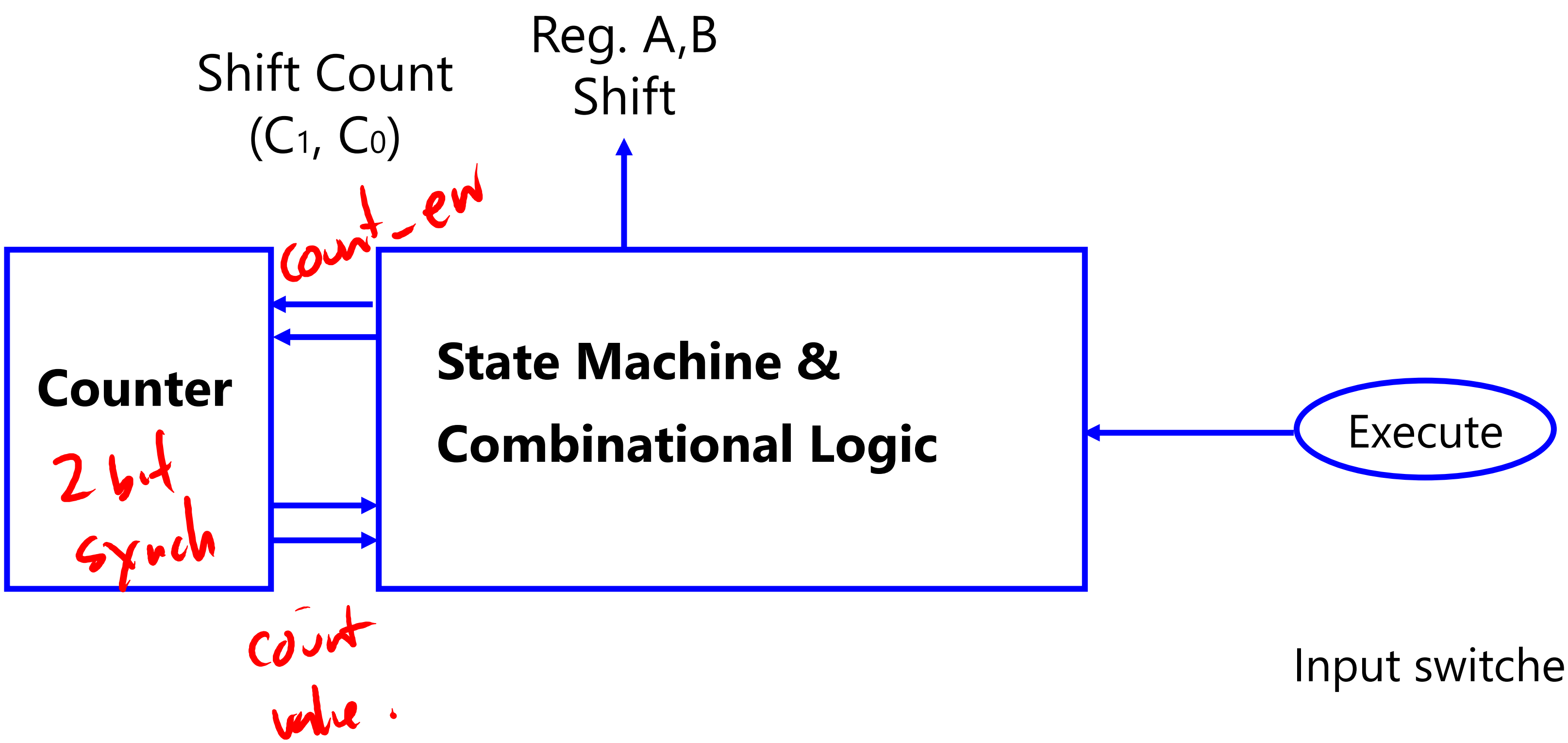


Experiment 3 – State Machine (Mealy)

- Arcs are labeled with both inputs & outputs
- Fewer states than Moore machine (outputs depend on both state and inputs)



Experiment 3 – Control Unit Block Diagram (Moore / Mealy)



Input switches

Experiment 3 –State Machine (Mealy)

SR

Kmaps for

to counter

inputs

Exec. Switch ('E')	Q	C1	C0	Reg. Shift ('S')	Q^+	$C1^+$	$C0^+$
0	0	0	0	0	0	0	0
0	0	0	1	d	d	d	d
0	0	1	0	d	d	d	d
0	0	1	1	d	d	d	d
0	1	0	0	0	0	0	0
0	1	0	1	1	1	1	0
0	1	1	0	1	1	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	0	1	d	d	d	d
1	0	1	0	d	d	d	d
1	0	1	1	d	d	d	d
1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0

State Machine Encoding

- Mealy and Moore machines are only one aspect of state machine design
- State encoding method is also important, 3 typical methods:
  - Binary numerical (easiest to think about, fewest registers)
  - Gray encoding (minimizes bit transitions when switching states)
  - One-hot encoding (minimizes decoding/next state circuitry, can use to efficiently implement lab 3…how?)

Experiment 3 – Hints

- Make design modular and unit test (note how demo points are distributed)
- Do not gate clock (one inverter to shift clock edge is ok)
- Use flip-flops, counters, shift registers, to store state bit(s) in control unit (fully synchronous design)
- Use “don’t care” entries in state transition table to your advantage (the logic should not be very complicated)