

Sequential Logic

- 1) Synchronous as an implementation of Sequential
- 2) Synchronous Timing Analysis
- 3) Single synchronous clock design
- 4) Finite State Machines
- 5) Metastability

Comp 120 - Spring 2005 2/1/04 LOG - Synchronous Logic 1

Road Traveled So Far...

Fets & voltages

Voltage-based encoding

- ♦ $V_{OL}, V_{IL}, V_{IH}, V_{OH}$

Combinational contract:

- ♦ discrete-valued inputs
- ♦ complete in/out spec.
- ♦ static discipline
- ♦ t_{CD} and t_{PD}

Logic gates

Acyclic connections

Composable blocks

Design:

- ♦ truth tables
- ♦ sum-of-products
- ♦ simplification
- ♦ muxes, ROMs, PLAs

Combinational logic circuits

Storage & state

Dynamic discipline

Finite-state machines

Metastability

Throughput & latency

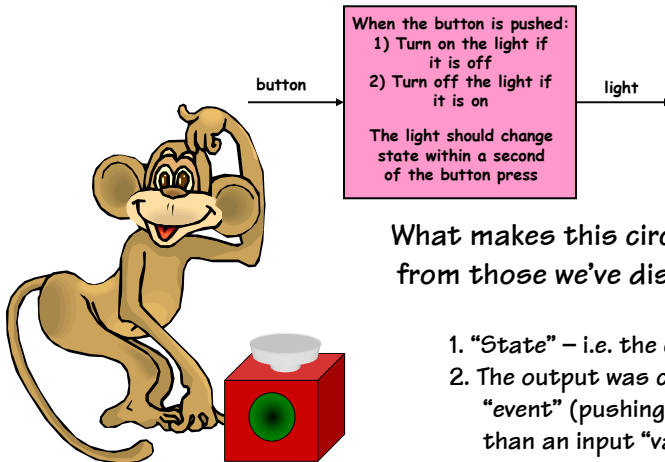
Pipelining

Our motto: Sweat the details once,
and then put a box around it!

Comp 120 - Spring 2005 2/1/04 LOG - Synchronous Logic 2

Something We Can't Build (Yet)

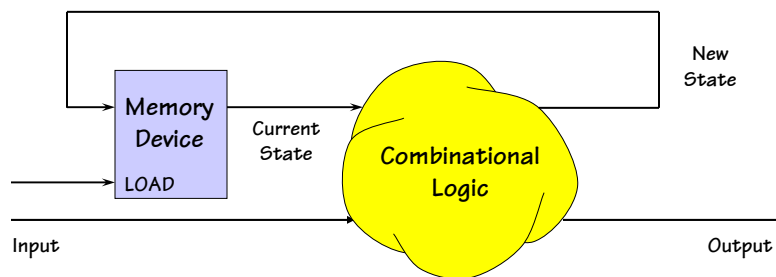
What if you were given the following design specification:



What makes this circuit so different from those we've discussed before?

1. "State" - i.e. the circuit has memory
2. The output was changed by an input "event" (pushing a button) rather than an input "value"

Sequential = Stateful



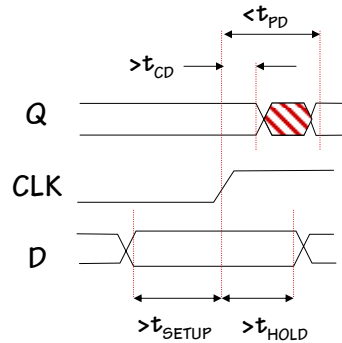
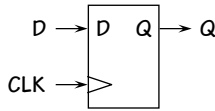
Plan: Build a Sequential Circuit with stored digital STATE -

- Memory stores CURRENT state, produced at output
- Combinational Logic computes
 - NEXT state (from input, current state)
 - OUTPUT bit (from input, current state)
- State changes on LOAD control input

Didn't we develop some memory devices last time?



Review of Flip Flop Timing



t_{PD} : maximum propagation delay, CLK \rightarrow Q

t_{CD} : minimum contamination delay, CLK \rightarrow Q

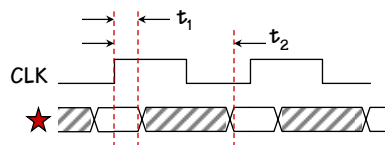
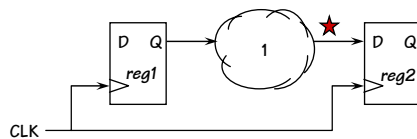
t_{SETUP} : setup time

guarantee that D has propagated through feedback path before master closes

t_{HOLD} : hold time

guarantee master is closed and data is stable before allowing D to change

Synchronous Timing Analysis



$$t_1 = t_{CD,reg1} + t_{CD,1} > t_{HOLD,reg2}$$

$$t_2 = t_{PD,reg1} + t_{PD,1} < t_{CLK} - t_{SETUP,reg2}$$

$$\text{Minimum Clock Period: } t_{CLK} > t_{PD,reg1} + t_{PD,1} + t_{SETUP,reg2}$$

Questions for register-based designs:

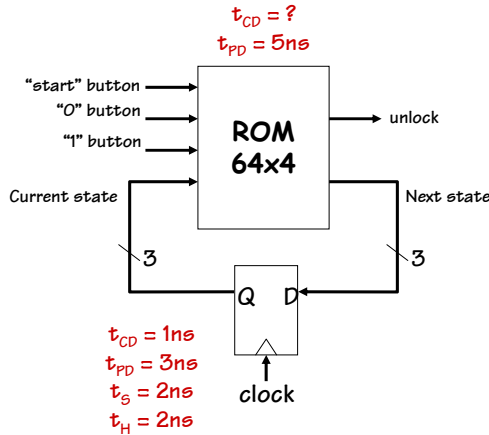
- ♦ How much time for useful work (i.e. for combinational logic delay)?

- ♦ Does it help to guarantee a minimum t_{CD} ? How 'bout designing registers so that

$$t_{CD,reg} > t_{HOLD,reg}?$$

- ♦ What happens if CLK signal doesn't arrive at the two registers at exactly the same time (a phenomenon known as "clock skew")?

Example: Flip Flop Timing



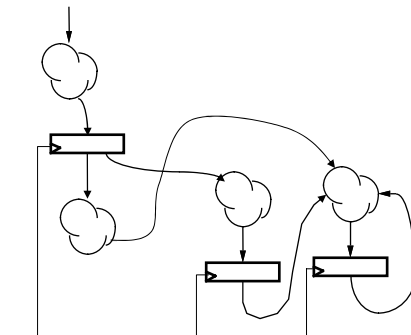
Questions:

1. t_{CD} for the ROM?
2. Min. clock period?
3. Constraints on inputs?

Single Synchronous Clock

Sequential \neq Synchronous

However, Synchronous = A recipe for robust sequential circuits:



- No combinational cycles
- Only care about value of combinational circuits just before rising edge of clock
- Period greater than every combinational delay
- Change saved state after noise-inducing logic transitions have stopped!

Designing Sequential Logic

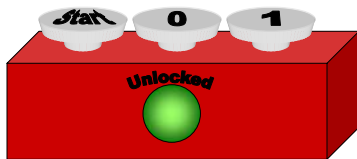
We'll use sequential logic when we need to organize the solution to some design problem as a sequence of steps:

How to open digital combination lock w/ 3 buttons ("start", "0" and "1"):

- Step 1: press "start" button
- Step 2: press "0" button
- Step 3: press "1" button
- Step 4: press "1" button
- Step 5: press "0" button



Information remembered between steps is called **state**. Might be just what step we're on, or might include results from earlier steps we'll need to complete a later step.

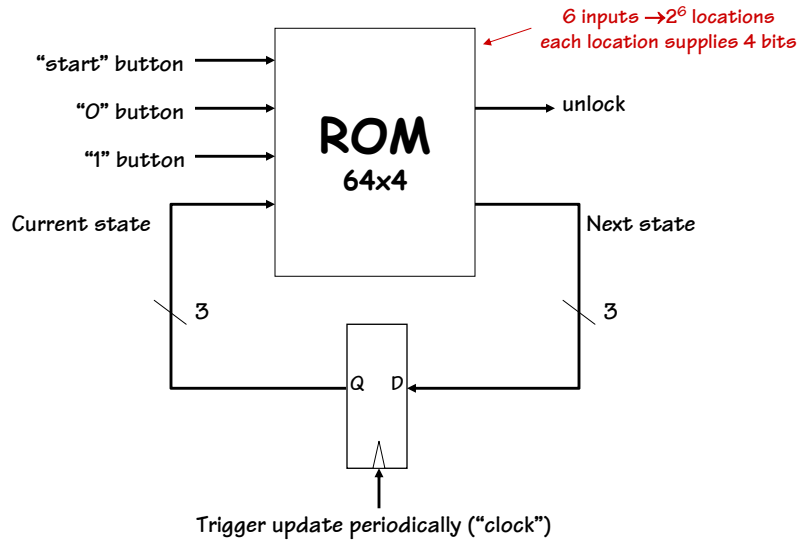


Implementing a "State Machine"

Current state	"start"	"1"	"0"	Next state	unlock
---	1	---	---	start 000	0
start 000	0	0	1	digit1 001	0
start 000	0	1	0	error 101	0
start 000	0	0	0	start 000	0
digit1 001	0	1	0	digit2 010	0
digit1 001	0	0	1	error 101	0
digit1 001	0	0	0	digit1 001	0
digit2 010	0	1	0	digit3 011	0
...					
digit3 011	0	0	1	unlock 100	0
...					
unlock 100	0	1	0	error 101	1
unlock 100	0	0	1	error 101	1
unlock 100	0	0	0	unlock 100	1
error 101	0	---	---	error 101	0

6 different states → encode using 3 bits

Now Do It With Hardware!

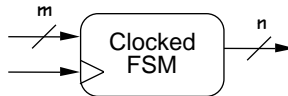


Comp 120 - Spring 2005

2/1/04

LO6 - Synchronous Logic 11

Abstraction du jour: Finite State Machines



A FINITE STATE MACHINE has

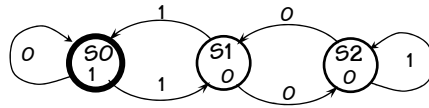
- k STATES $S_1 \dots S_k$ (one is "initial" state)
- m INPUTS $I_1 \dots I_m$
- n OUTPUTS $O_1 \dots O_n$
- Transition Rules $S'(s,i)$
for each state s and input i
- Output Rules $Out(s)$ for each state s

Comp 120 - Spring 2005

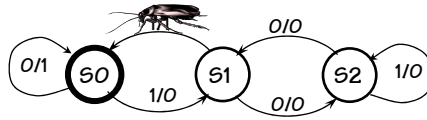
2/1/04

LO6 - Synchronous Logic 12

Valid State Diagrams



MOORE Machine:
Outputs on States



MEALY Machine:
Outputs on Transitions

Arcs leaving a state must be:

- (1) **mutually exclusive**

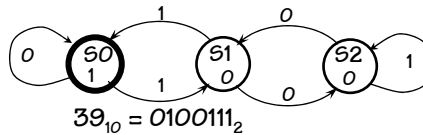
can't have two choices for a given input value

- (2) **collectively exhaustive**

every state must specify what happens for each possible input combination. "Nothing happens" means arc back to itself.

Let's Play State Machine

Let's emulate the behavior specified by the state machine shown below when processing the following string from LSB to MSB.



	State Input		Next Output	
T=0	S0	1	S1	0
T=1	S1	1	S0	1
T=2	S0	1	S1	0
T=3	S1	0	S2	0
T=4	S2	0	S1	0
T=5	S1	1	S0	1
T=6	S0	0	S0	1



It looks to me like this machine outputs a 1 whenever the bit sequence that it has seen thus far is a multiple of 3.

Busted Stuff

CONVENIENT NOTATION:
When a transition is made on the next input regardless of its value the arc can be labeled with an X or -

AMBIGUOUS TRANSITIONS (Mutual Exclusive property violated):
For each input there can only be one arc leaving a state

UNSPECIFIED TRANSITIONS (Collectively Exhaustive property violated):
There must be an arc leaving a state for all valid inputs (It can, however, loop back to the same state)

Comp 120 - Spring 2005 2/1/04 LO6 - Synchronous Logic 17

FSM Party Games

1. What can you say about the number of states?

2. Same question:

3. Here's an FSM. Can you discover its rules?

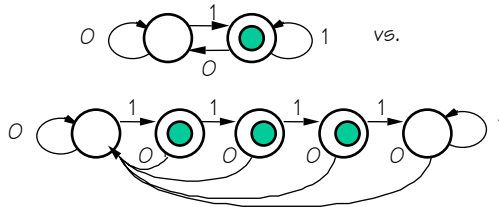
Comp 120 - Spring 2005 2/1/04 LO6 - Synchronous Logic 18

What's My Transition Diagram?



0=OFF,
1=ON?

"1111" =
Surprise!

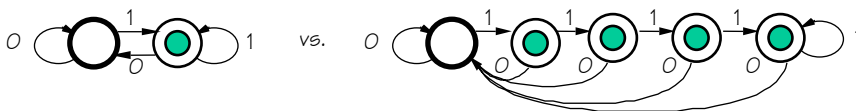


- If you know NOTHING about the FSM, you're never sure!
- If you have a BOUND on the number of states, you can discover its behavior:

K-state FSM: Every (reachable) state can be reached in < k steps.

BUT ... states may be **equivalent!**

FSM Equivalence



ARE THEY DIFFERENT?

NOT in any practical sense! They are EXTERNALLY INDISTINGUISHABLE, hence interchangeable.

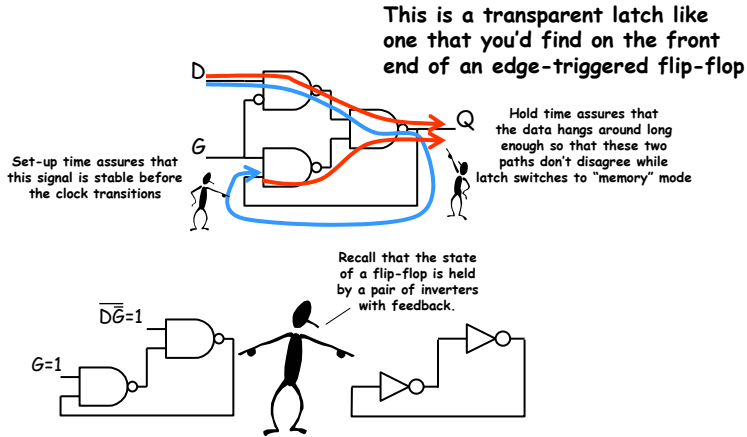
FSMs EQUIVALENT iff every input sequence yields identical output sequences.

ENGINEERING GOAL:

- HAVE an FSM which works...
- WANT simplest (ergo cheapest) equivalent FSM.

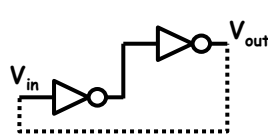
The world doesn't run on our clock!

So, what happens if we miss an occasional set-up or hold time?



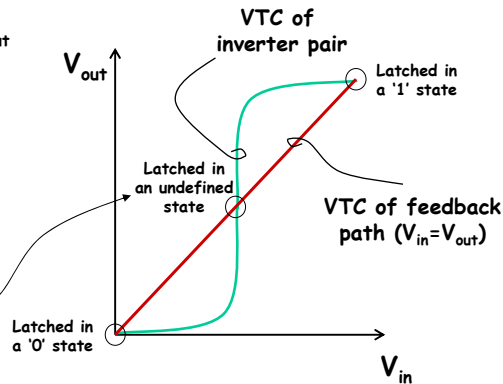
Persistent invalid states

One of the jobs of a digital gate is to restore questionable input signals to a valid output levels.



A question that arises sometimes: which valid level to restore to?

The only way this happens is if we allow an invalid level into our feedback loop. That only happens if we violate our set-up or hold times.

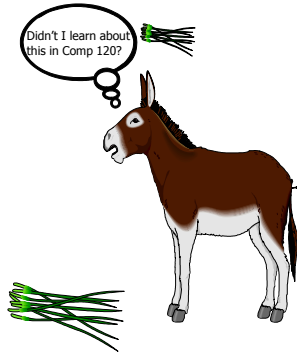


Metastability

Metastability is the occurrence of a persistent invalid output. An unstable equilibria.

The idea of Metastability is not new:

The Paradox of Buridan's Ass

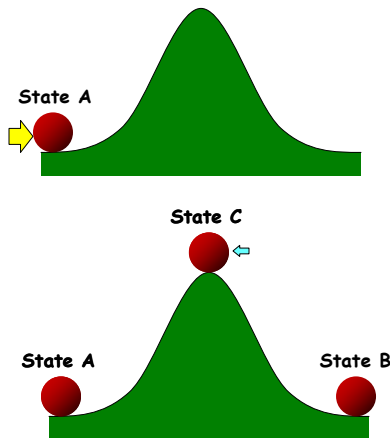


Buridan, Jean (1300-58), French Scholastic philosopher, who held a theory of determinism, contending that the will must choose the greater good. Born in Bethune, he was educated at the University of Paris, where he studied with the English Scholastic philosopher William of Ockham (whom you might recall from his razor business). After his studies were completed, he was appointed professor of philosophy, and later rector, at the same university. Buridan is traditionally, but probably incorrectly, associated with a philosophical dilemma of moral choice called "Buridan's ass."

In the problem an ass starves to death between two alluring bundles of hay because it does not have the will to decide which one to eat.

Real-World Metastability

If we launch a ball up a hill we expect one of 3 possible outcomes:

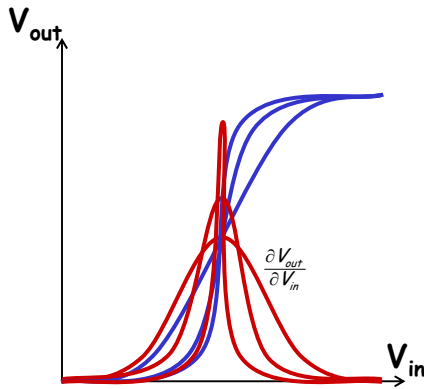


- a) Goes over
- b) Rolls back
- c) Stalls at the apex

That last outcome is not very stable.

- a gust of wind
- Brownian motion
- it doesn't take much

How do balls relate to digital logic?



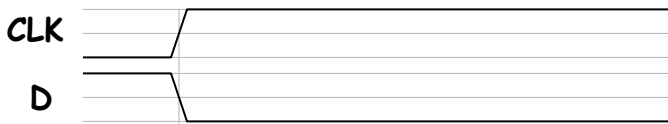
Our hill is simply the derivative of the VTC.

Notice that the higher the gain thru the transition region, the steeper the peak of the hill. Thus, making it harder to get into a metastable state.

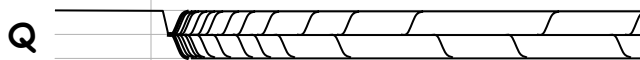
Nonetheless, Metastability will happen!

Observed Behavior: typical metastable symptoms

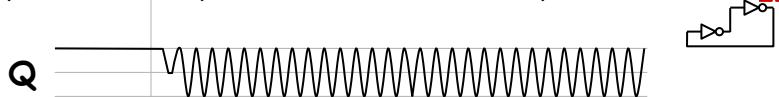
Following a clock edge on an asynchronous input:



We may see exponentially-distributed metastable intervals:



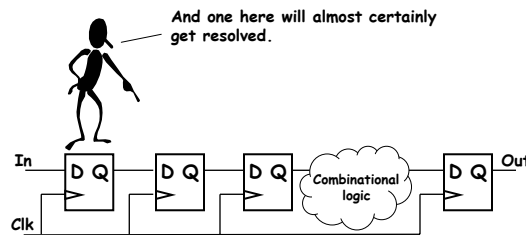
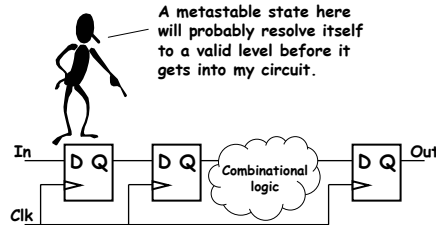
Or periods of high-frequency oscillation (if the feedback path is long):



Avoiding Metastability

Synchronizers, extra flip flops between the asynchronous input and your logic, are the best insurance against metastable states.

The higher the clock rate, the more synchronizers should be considered.



Comp 120 – Spring 2005

2/1/04

L06 – Synchronous Logic 29

Summary

- Dynamic discipline (setup and hold times)
- Sequential logic = combinational logic + memory
- New abstraction: Finite State Machines (FSMs)
 - transition diagrams showing states, outputs, and
 - transition arcs: mutually exclusive, collectively exhaustive
- Metastability
 - Synchronous logic admits the possibility of a persistent invalid state
 - This state will eventually resolve to a valid state, but it could take an arbitrarily long time
 - By using synchronizer Flip Flops on all potentially asynchronous inputs we can reduce probability of admitting an invalid value into our system to a frequency longer than the product's lifespan.

Comp 120 – Spring 2005

2/1/04

L06 – Synchronous Logic 30