Lecture 6

Control Logic

Objectives

- Understand how digital systems may be divided into a data path and control logic
- Appreciate the different ways of implementing control logic
- Understand how shift registers and counters can be used to generate arbitrary pulse sequences
- Understand the circumstances that give rise to output glitches

Control Logic

Most digital systems can be divided into

- Data Path: adders, registers etc
- Control Logic: generates timing signals to ensure things happen at the right time and in the right order

Control logic can be implemented with:

- Microprocessor/Microcontroller
  - + Cheap, very flexible, design easy (software)
  - – Slow: most actions require >20 instructions = 2 μs @ clock speed of 10 MHz.

  Use for slow applications.
- Synchronous State Machine
  - + Fast (20 ns/action), Cheap using programmable logic.
  - – Hard to design complex systems. Limited data storage.

  Use for fast, moderately complex systems.
- Counters/Shift Registers
  - + Fast, Cheap, Very easy design.
  - – Simple systems only.

  A special case of synchronous state machines.

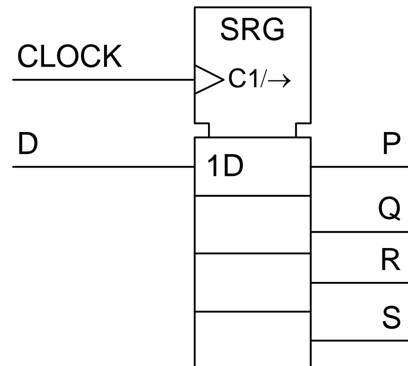
  Use for very simple systems (fast or slow).

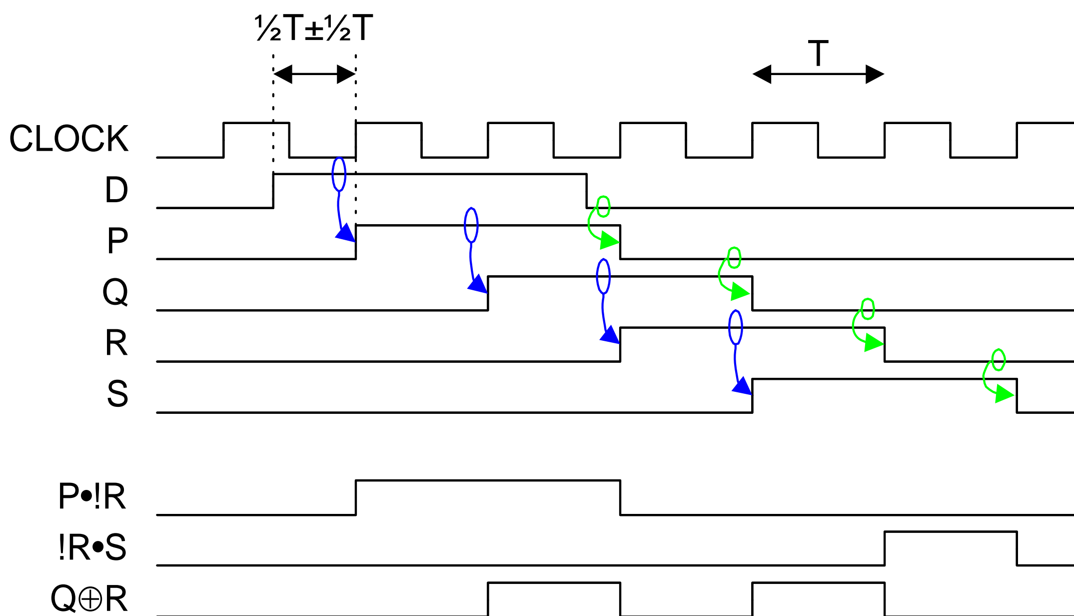
Shift Registers

Easy way to make a sequence of events happen in response to a trigger:

– P, Q, R and S are delayed versions of D but with all transitions on the CLOCK ↑

– Delay from D to P is between 0 and 1 clock cycle.



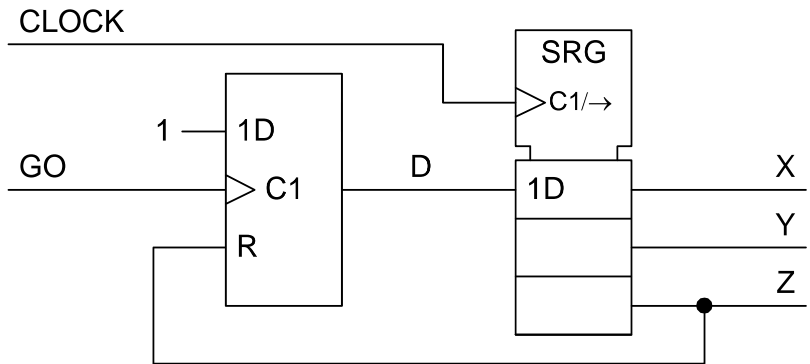


– P•!R gives pulse of length $2T$ approx $½T$ after D↑.

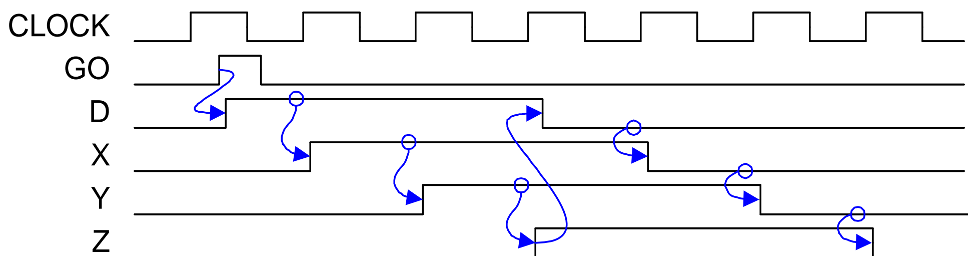
– !R•S gives pulse of length T approx $2½T$ after D↓.

– Q⊕R gives pulses of length T approx $1½T$ after D↑ & ↓

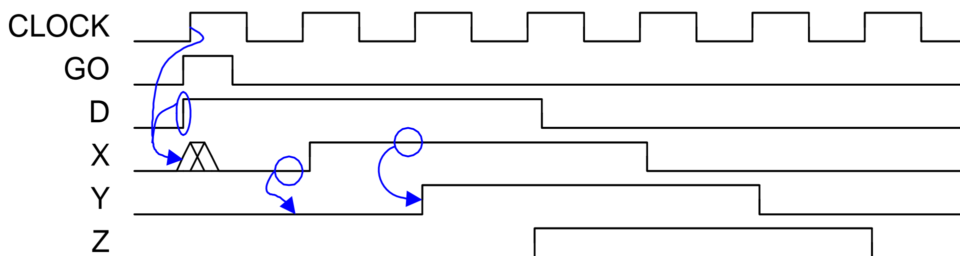
Shift Registers with Short Input Pulses



- D might be ignored if it lasts < 1 CLOCK period
- GO input is sent to a edge-triggered input
- Works like a toaster: Z causes D to turn off halfway through the whole cycle.



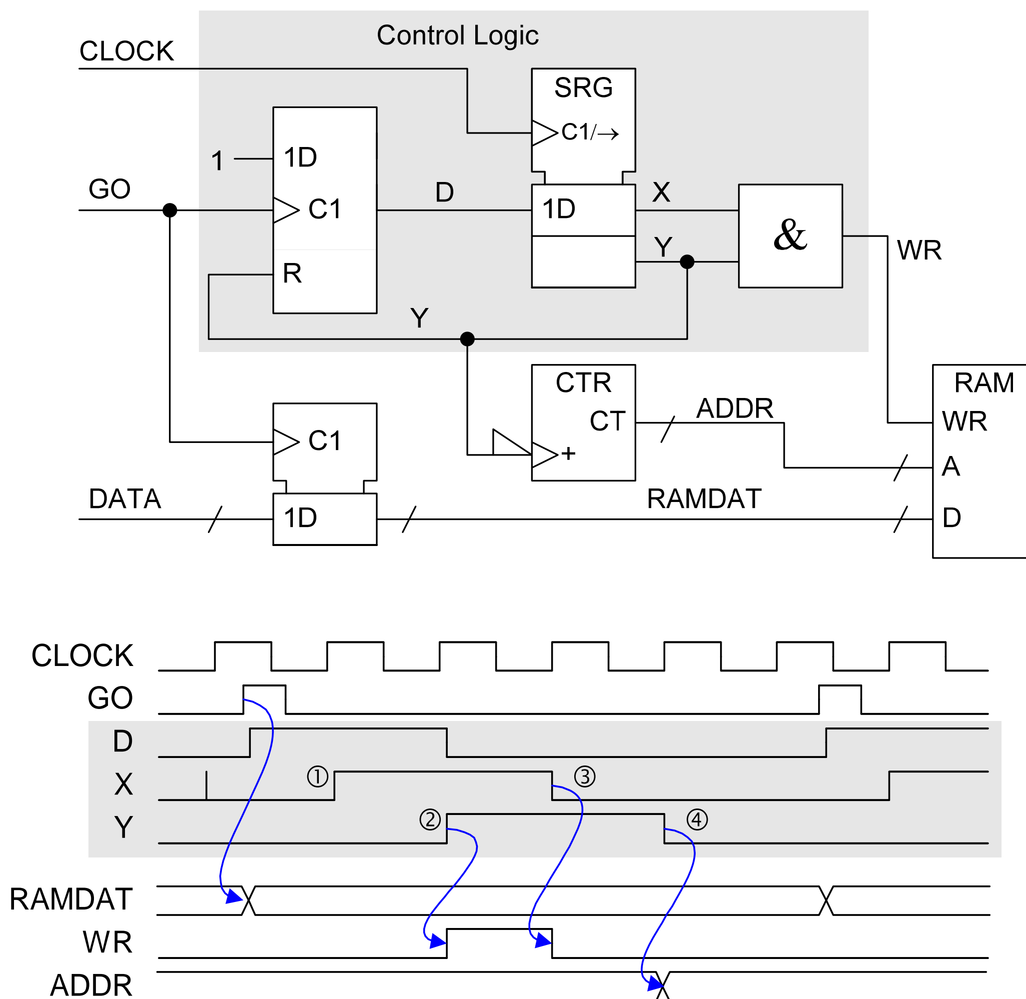
- Use the X output with care: it may oscillate for tens of ns if D changes within setup/hold window:



- X is OK by next clock ↑ so Y and Z are safe to use.

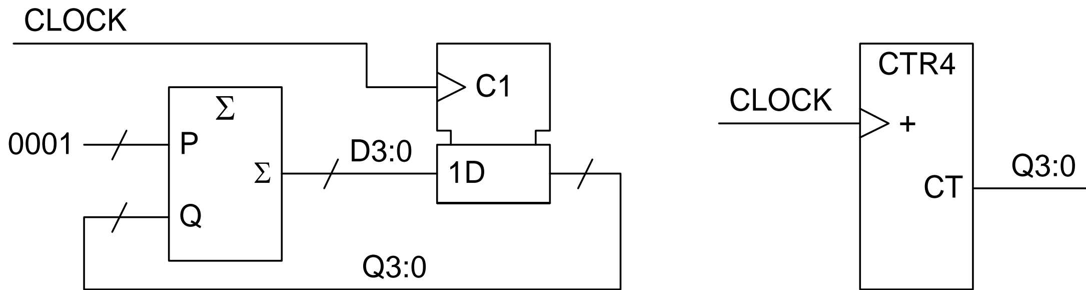
Shift Register Example: Logic Analyser

On every GO rising edge we must sample DATA and store it in the RAM.

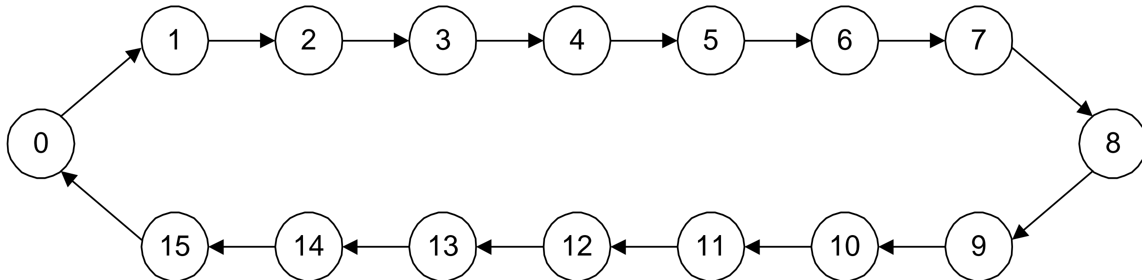


- RAM control signals are easily generated from the shift register. Four time instants available: ① to ④.
- We don't use ① so it doesn't matter if X has a glitch on the previous cycle since it is ANDed with Y (which is low at the time).

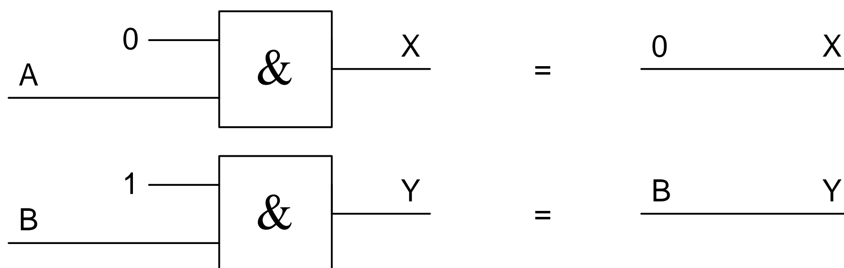
Synchronous Counters



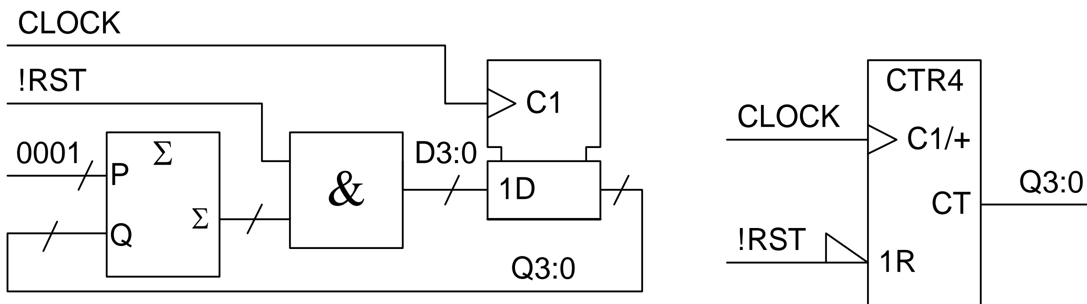
- An N bit binary counter has a cycle length of 2^N states. We can draw a state diagram in which one transition is made for each clock ↑ :



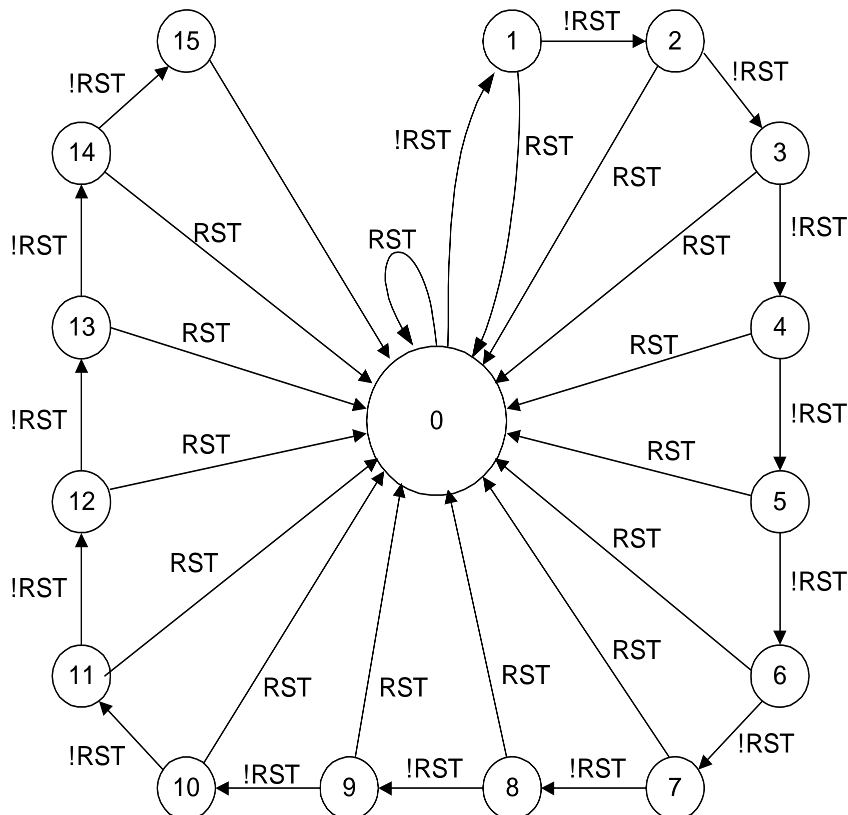
- Adder can be simplified: one set of inputs is fixed so many gates can be eliminated:



Synchronous RESET



- This is a *synchronous* reset input: taking !RST low has no effect until the next clock ↑
- In a synchronous counter everything is done by manipulating the D inputs of the register.



Detecting Counter Output Values

Notation: CT = Contents
0 = least significant bit (LSB)
Bit k has a binary weight of 2^k
1R means reset on next C1 (CLOCK ↑ edge)

- Z is high whenever Q3:0 = 1??1 1001 = 9
  1011 = 11
  lowest value is when 1101 = 13
  all the ? bits are zero 1111 − 15
- Counter resets after 9 giving a cycle length of 10 states:

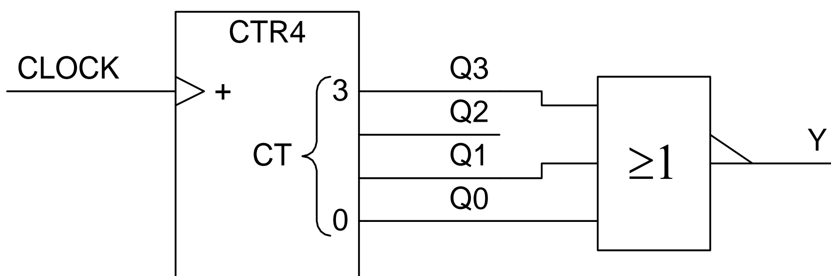
Output Glitches

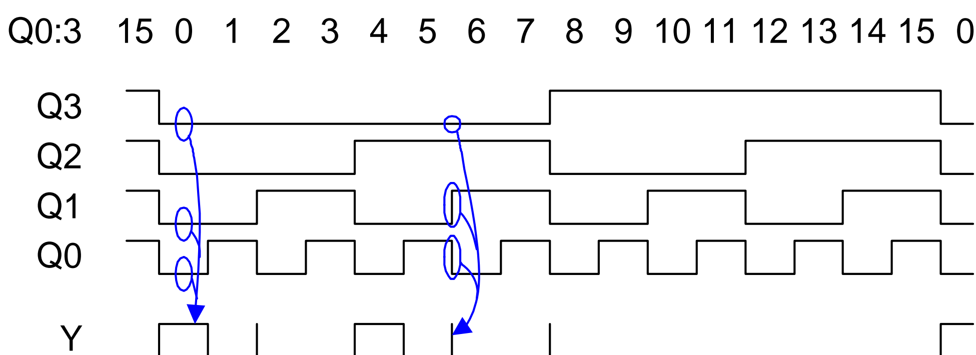
If k counter bits change "simultaneously", other logic circuits using them may briefly see any of 2^k possible values.

Glitches are possible at the logic circuit output if both:

1. These 2^k values include any that would cause the logic circuit output to change.

and 2. The logic circuit output is meant to remain at a constant value.



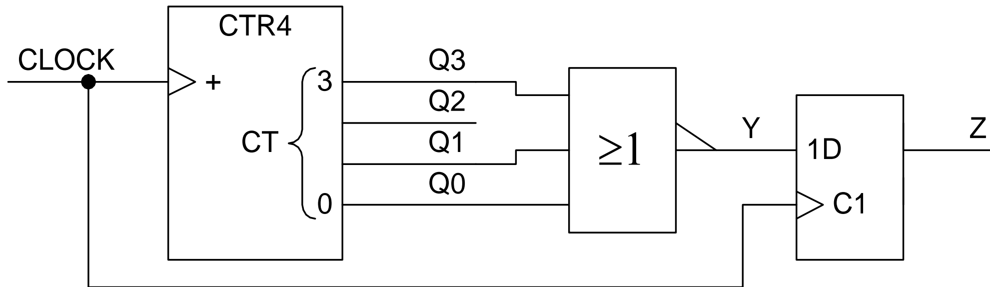


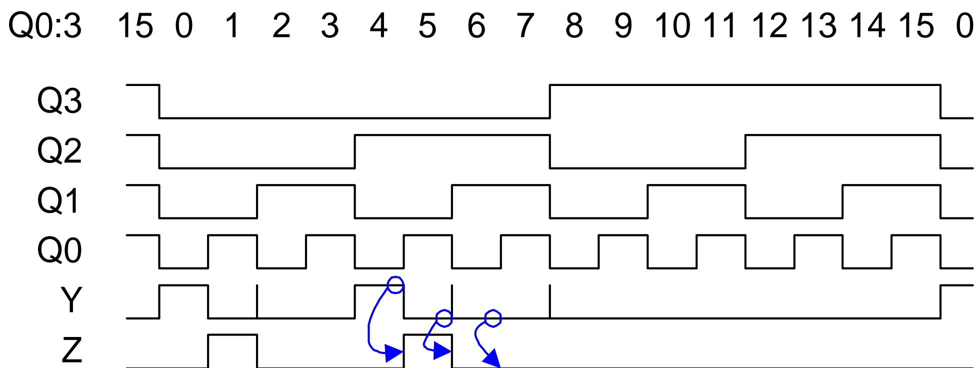
•Y is high when Q=0000 or 0100

•Transition 1 → 2: Q=00?? which includes 0000
•Transition 5 → 6: Q=01?? which includes 0100
•Transition 7 → 8: Q=???? which includes both

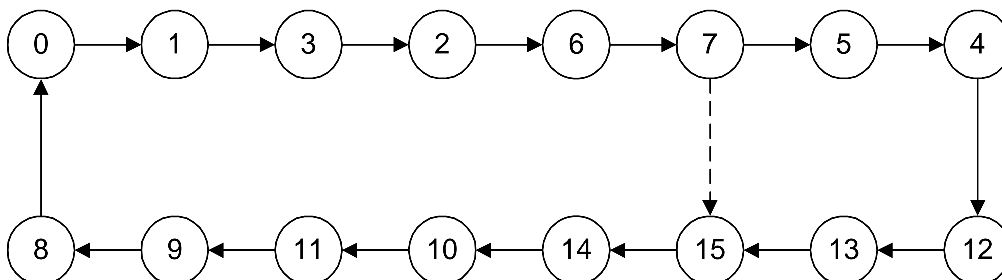
Eliminating Output Glitches

We can eliminate output glitches by delaying Y with a flipflop:





Alternatively use a count sequence where only one bit changes at a time (e.g. Gray code):



Top and bottom rows differ only in the MSB ⇒ any even count length can be made by branching to the bottom row after half the counts. Dashed line gives a ÷12 counter.

Quiz Questions

1. If the CLOCK period is T, what is the range of possible time delays between a change in the DATA input of a shift register and the resultant change in the output of the first stage?
2. How do you combine the outputs of a shift register to generate a pulse for both the rising and the falling edges of its input signal?
3. In order to guarantee that a shift register will notice a pulse on its DATA input, how long must a pulse last?
4. If an AND gate is used to combine 2 of the outputs from a 4-bit counter, how many different count values will make the AND gate output go high?
5. Why do output glitches not occur when a counter counts from 6 to 7?
6. Name two ways in which output glitches may be avoided.

Answers are all in the notes.

Lecture 7

Data Decoding with a Counter

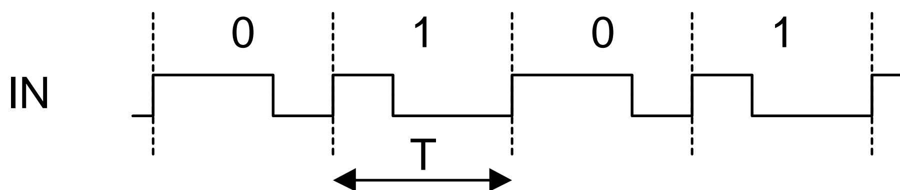
This design example illustrates

- Using a counter to measure time intervals
- The logic symbol notation for a bidirectional counter
- Why it is necessary to use a flipflop to synchronise an asynchronous input signal
- Detailed timing analysis for asynchronous signals
- Assembling a larger design bit by bit

Data Decoding

Task: Decode a data stream where a 0 or 1 is transmitted as a pulse lasting $^2/_3 T$ or $^1/_3 T$ respectively.

Problem: you don't know the value of T.



Method: (a) Wait for a rising edge

(b) Time how long until the next falling edge

(c) Time how long until the next rising edge

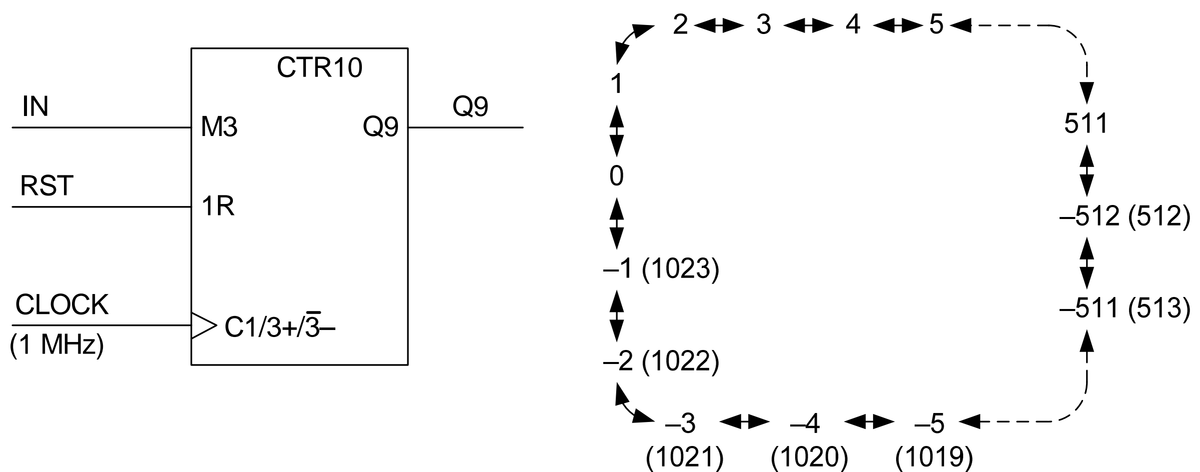
(d) Output a 0 or 1 according to which is longer and then go back to (b).

How do you measure time intervals ?

With a counter.

- Reset the counter at the rising edge
- Count upwards while IN=1
- Count downwards while IN=0
- See if it is +ve or –ve just before you reset it at the next rising edge.

Counter Symbol



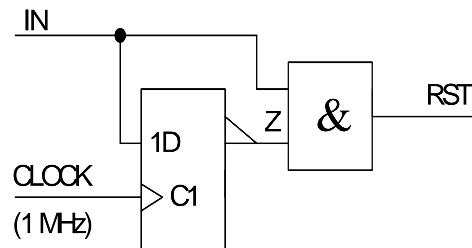
Notation:

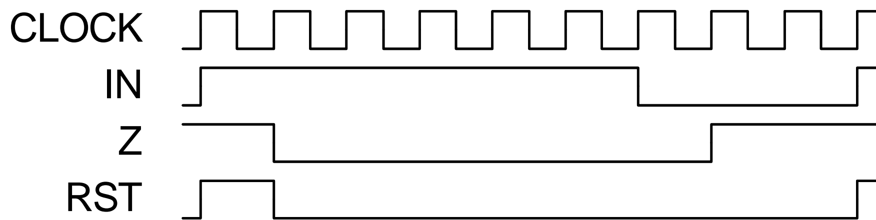
- M3 is a "mode" input which controls the counting direction. We connect this to IN.
- The CLOCK input is
  - +ve edge triggered – indicated by the ">" symbol
  - Has three separate functions divided by "/"
    - C1 means it is a clock for some other feature of the circuit
    - 3+ means that the counter increments on each CLOCK rising edge if the M3 input is high
    - $\overline{3}-$ means that the counter decrements on each CLOCK rising edge if the M3 input is low
- 1R: The "1" means that this input only has any effect when C1 is active (i.e. the rising edge of CLOCK). R means the RST input sets the counter to zero when it is high.
- CTR10 means it is a 10 bit counter: 0 to 1023. It will wrap around from 1023 to 0 when counting up and from 0 to 1023 when counting down so 1023 is equivalent to –1. Q9, the MSB, tells you when it is negative.

Resetting the Counter

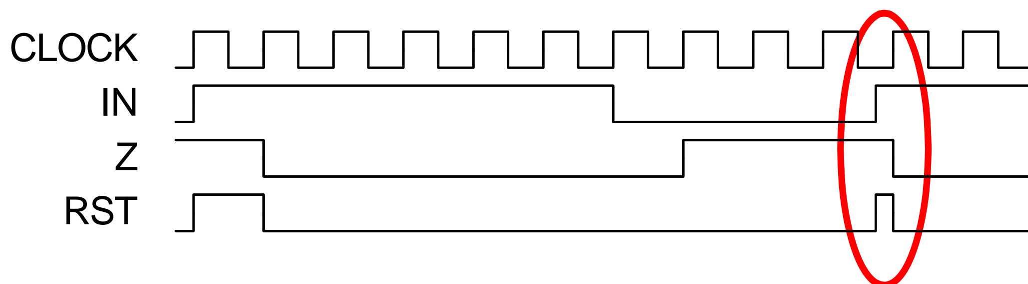
Task: We want to reset the counter on every rising edge of IN.

Method: Use a 1-bit shift register to generate a reset pulse.



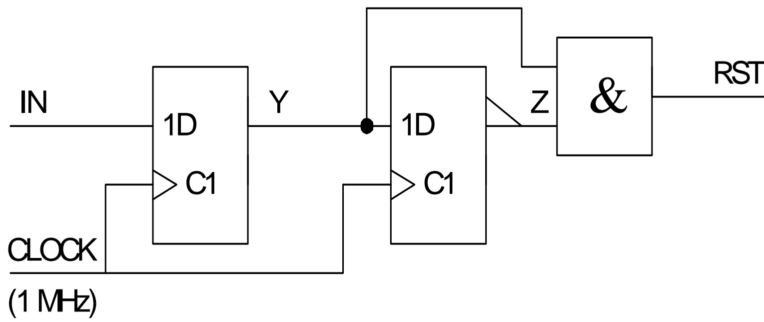


- Z is an inverted version of IN but is 1 clock cycle later.
- RST goes high for one clock cycle every time IN goes high
- **Problem 1**: If IN is unsynchronised (can change at any part of the CLOCK cycle), we might get very short RST pulses.



Getting Rid of Glitches

Solution 1: synchronise IN. Y is always synchronised below.

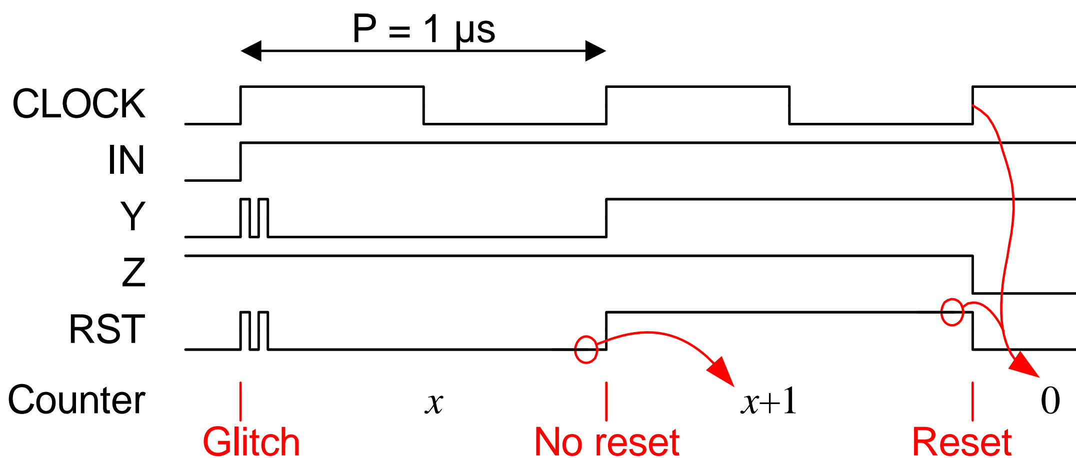


All changes of Y occur just after the clock rising edge.

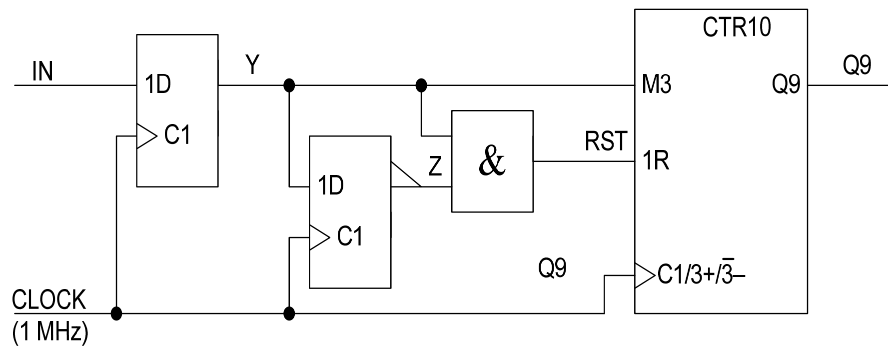
Potential Problem 2:

If IN changes just on the clock edge, Y (and RST) could oscillate.

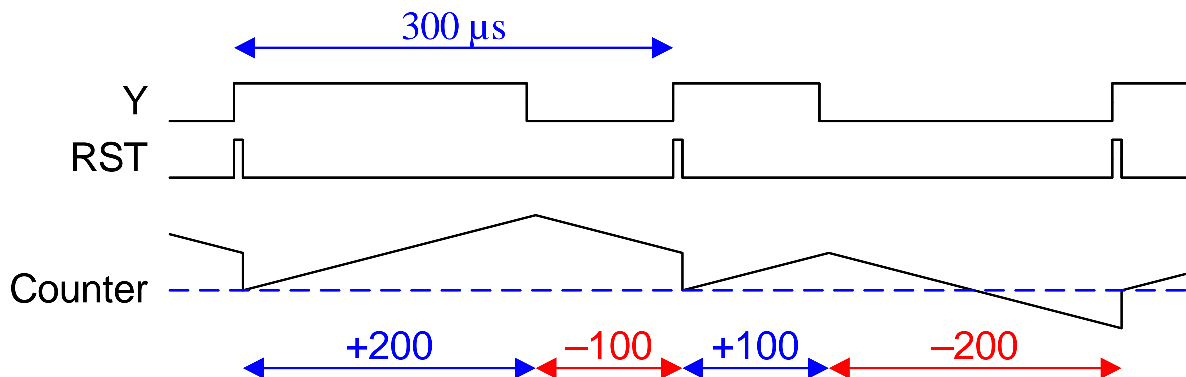
Doesn't matter because the counter only looks at RST on the next clock rising edge and the oscillation will be gone by then.



Timing the input pulses



- Count **up** when Y is high and down when it is **low**
- Each bitcell lasts 300 μs ⇒ 300 clock cycles



For a logic **zero**

- Count up by 200 then down by 100 ⇒ +100 at end of cell

For a logic **one**

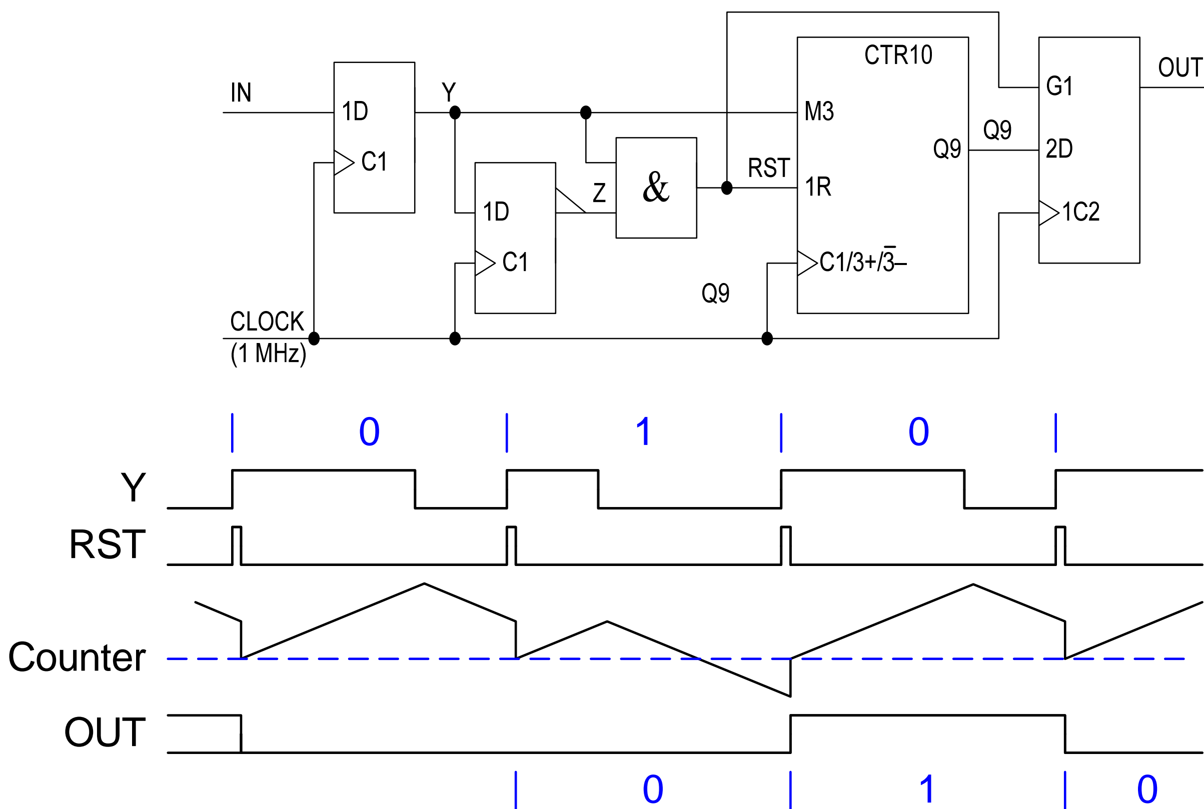
- Count up by 100 then down by 200 ⇒ –100 at end of cell

Counter MSB, Q9, is 0 for positive numbers and 1 for negative

Saving the Answer

We need to remember the value of Q9 just before the counter is reset.

- Use the RST pulse to enable the clock of a flipflop
  - G1 is a "gating" input: it enables something when it is high
  - 1C2 is a clock input but only when G1 is true



OUT gives the decoded data stream but one bitcell late.

Slowest and Fastest Data Rate

Clock = $1/P$ Hz, Bitcell = T seconds, Counter = n bits

Slowest Data Rate

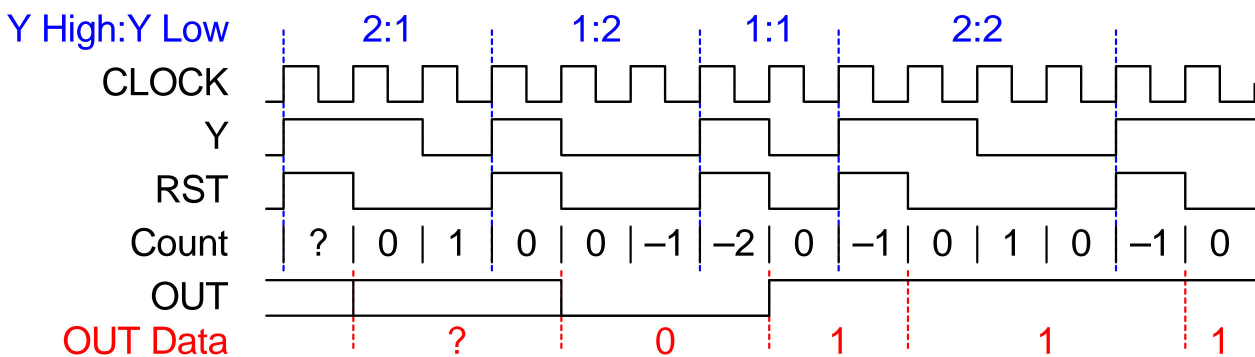
At the end of the bitcell, counter reaches $\pm T/3P$. To ensure that Q9 is correct, this must not exceed half the counter range. Hence

$$T/3P < 0.5 \times 2^n \quad \Rightarrow \quad T < 1.5 \times 2^n \times P = 1536\mu s$$

It doesn't matter if the counter exceeds this range in the middle of a cell: only the final value matters.

Fastest Data Rate

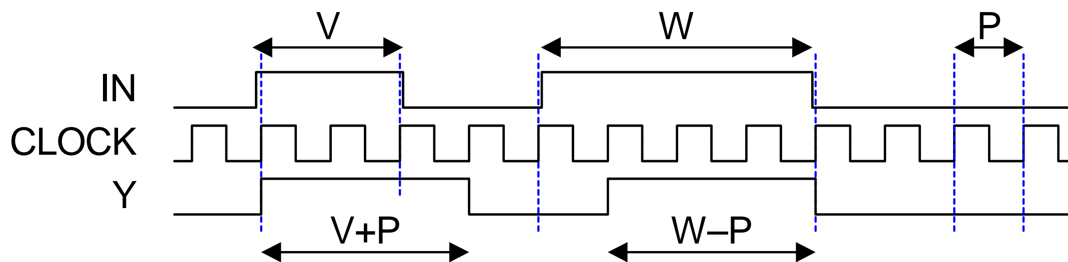
OUT only goes low if Y goes high for **more** cycles than low.



We have to make sure that when IN high:low = $^2/_3 T : {}^1/_3 T$ this results in Y being high for more clock cycles than it is low.

Fastest Data Rate

If a pulse on IN has length W then the length of the corresponding synchronised pulse on Y is W±P.



If IN high:low = ${}^{2}/_{3}T : {}^{1}/_{3}T$

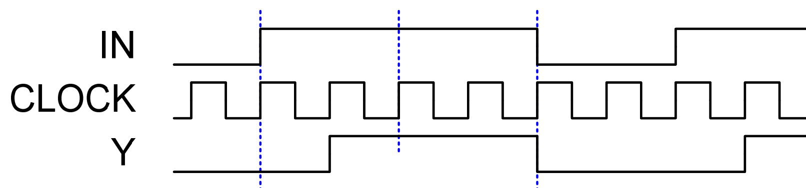
then Y high:low = ${}^{2}/_{3}T \pm P : {}^{1}/_{3}T \pm P$

it follows that we need

$${}^{2}/_{3}T \pm P > {}^{1}/_{3}T \pm P \Rightarrow {}^{2}/_{3}T - P > {}^{1}/_{3}T + P$$

$$\Rightarrow 2T - 3P > T + 3P \Rightarrow T > 6P = 6\ \mu\text{s}$$

Example of failure when $T = 6\ \mu$s



If **rising** edges of IN are just too late to be sensed by the clock but **falling** edge is just early enough then Y is high for 3 cycles and low for three cycles $\Rightarrow$

Quiz Questions

1. If a flipflop input is labelled “2C1” what is its function ?
2. If a counter input is labelled “$C1/3+/\overline{3}-$ ” what is its function?
3. What is the difference between a synchronous and an asynchronous reset input to a counter ?
4. Why doesn’t it matter if the input to an asynchronous reset input has glitches just after the clock rising edge?
5. If a 10-bit counter initially contains 1020 and is then incremented 10 times, what value will it then contain?
6. What is the minimum and maximum number of clock rising edges included in an asynchronous pulse that lasts x clock cycles?
7. What is the smallest values of x to guarantee that
    1. ceil(x) <= floor(2x)
    2. ceil(x) < floor(2x)

Answers are all in the notes.

Lecture 8

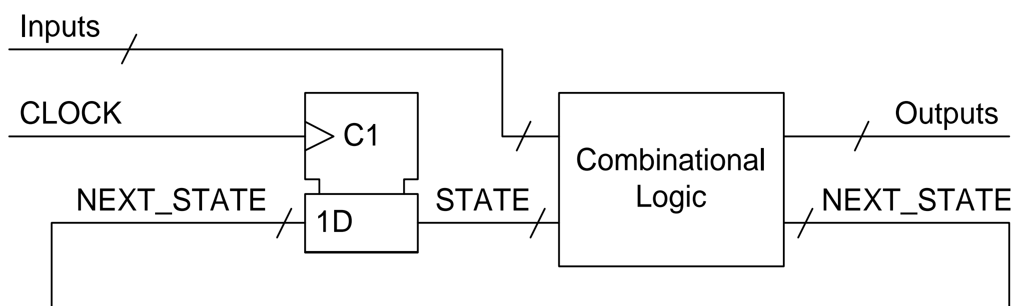
Synchronous State Machine Analysis

Objectives

- Review the definition of a synchronous state machine
- Learn how to construct the state table and state diagram of a state machine from its circuit diagram
- Appreciate the alternative ways of drawing the state diagram
- Learn how to draw the output waveforms of a state machine given its initial state and input waveforms
- Understand the causes of glitches in state machine outputs

Synchronous State Machines

Synchronous State Machine = Register + Logic

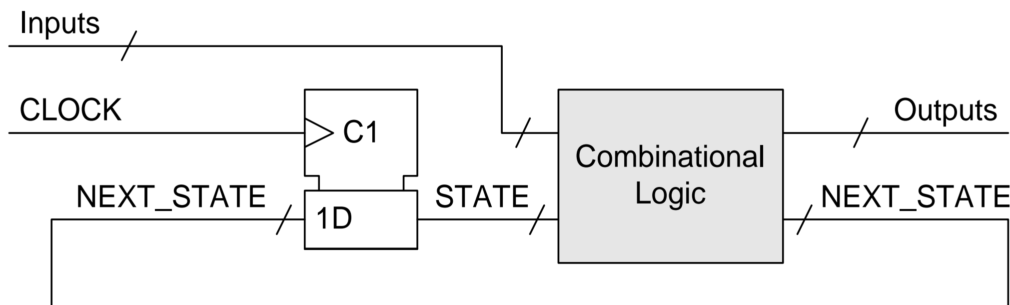


– The *state* is defined by the register contents
– Register has n flipflops $\Rightarrow 2^n$ states
– The state only ever changes on CLOCK↑
  - We stay in a state for an exact number of CLOCK cycles
– The state is the only memory of the past

Rules:

– Never mess around with the clock signal
– Never use *asynchronous* SET/RESET inputs to register (*asynchronous* = independent of CLOCK)

Combinational Logic Block



– The combinational logic outputs specify two things:

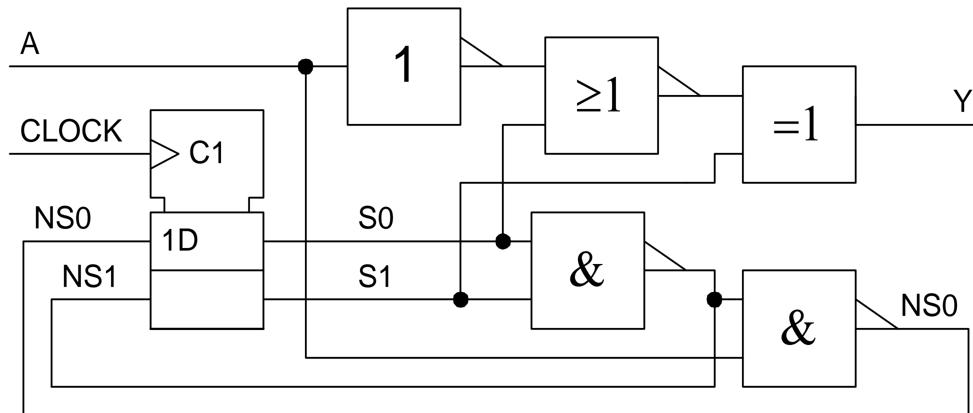
✶ **The output signals during the current state**
These may change during the state if the inputs change

✸ **Which state to go to at the next CLOCK ↑**
This too may change during a state but the only thing that matters is its value just before CLOCK ↑

– *combinational* logic has no internal feedback loops
⇒ no memory

- combinational logic outputs are entirely determined by the current STATE and the current Inputs

Analysing a State Machine



State Table:

Truth table for the combinational logic:

– One row per state: n flipflops $\Rightarrow 2^n$ rows

– One column per input combination:
m input signals $\Rightarrow 2^m$ columns

– Each cell specifies the *next state* and the *output signals during the current state*

- for clarity, we separate the two using a /

NS1,NS0/Y

S1,S0	A=0	A=1
00	11/0	10/1
01	11/0	10/0
10	11/1	10/0
11	01/1	01/1

Drawing the State Diagram

Split state table into two parts:

NS1,NS0/Y

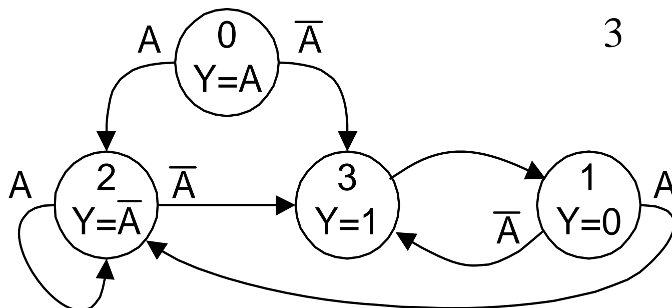
S1,S0	A=0	A=1
00	11/0	10/1
01	11/0	10/0
10	11/1	10/0
11	01/1	01/1

Next State: NS1:0

S1:0	A=0	A=1
0	3	2
1	3	2
2	3	2
3	1	1

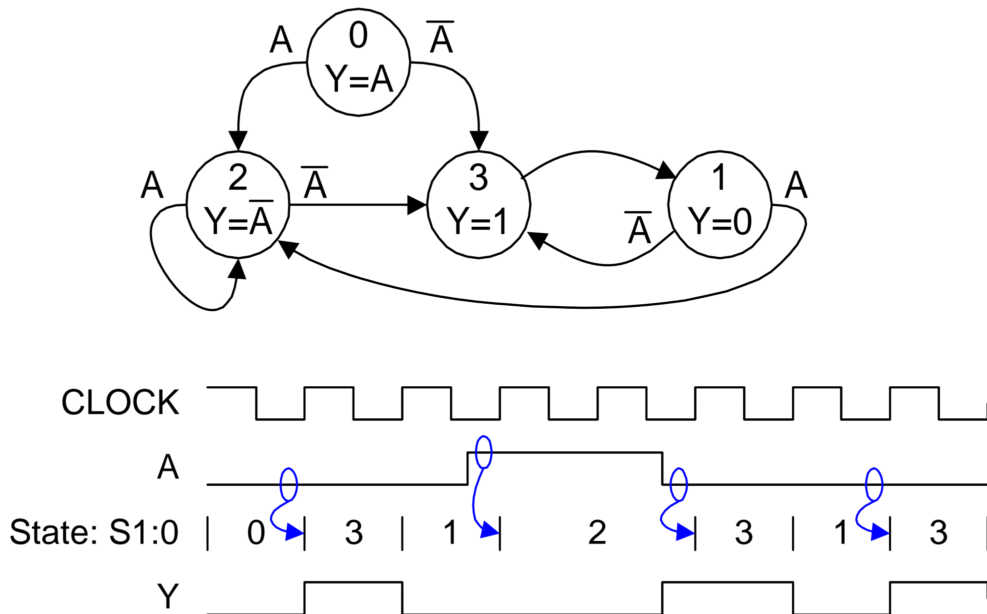
Output Signal: /Y

S1:0	A=0	A=1	
0	/0	/1	Y=A
1	/0	/0	Y=0
2	/1	/0	Y=!A
3	/1	/1	Y=1



- Transition arrows are marked with Boolean expressions saying when they occur
  - Every input combination has exactly one destination.
  - Unlabelled arrows denote unconditional transitions
- Output Signals: Boolean expressions within each state.

Timing Diagram



State machine behaviour is entirely determined by:

- The initial state
- The input signal waveforms

State Sequence:

Determine this first. Next state depends on input values just before CLOCK ↑.

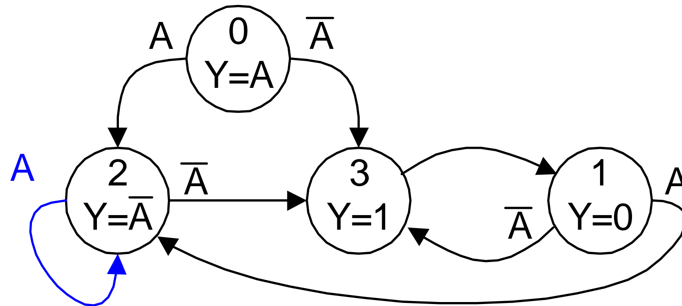
Output Signals:

Defined by Boolean expressions within each state.

If all the expressions are constant 0 or 1 then outputs only ever change on clock ↑. (*Moore machine*)

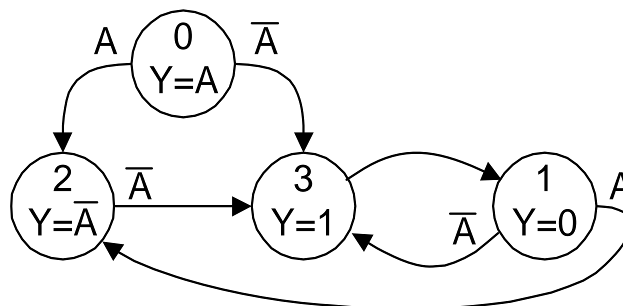
If any expressions involve the inputs (e.g. Y=A) then it is possible for the outputs to change in the middle of a state. (*Mealy machine*)

Self-Transitions

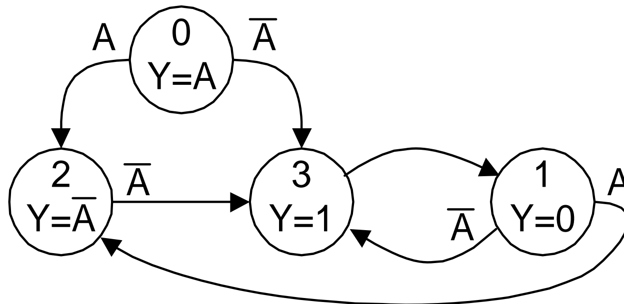


- We can omit transitions from a state to itself.
  - Aim: to save clutter on the diagram.

- The state machine remains in its current state if none of the transition-arrow conditions are satisfied.
  - From state 2, we go to state 3 if !A occurs, otherwise we remain in state 2.

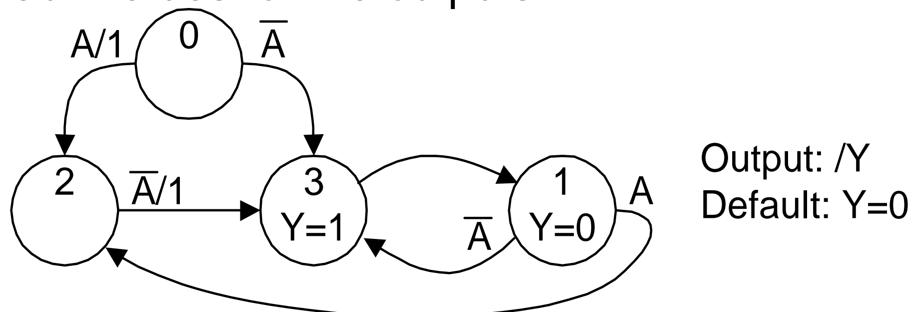


Output Expressions on Arrows



It may make the diagram clearer to put output expressions on the arrows instead of within the state circles:

- Useful if the same Boolean expression determines both the *next state* and the *output signals*.
- For each state, the output specification must be *either* inside the circle *or else* on *every* emitted arrow
- If self transitions are omitted, we must declare default values for the outputs

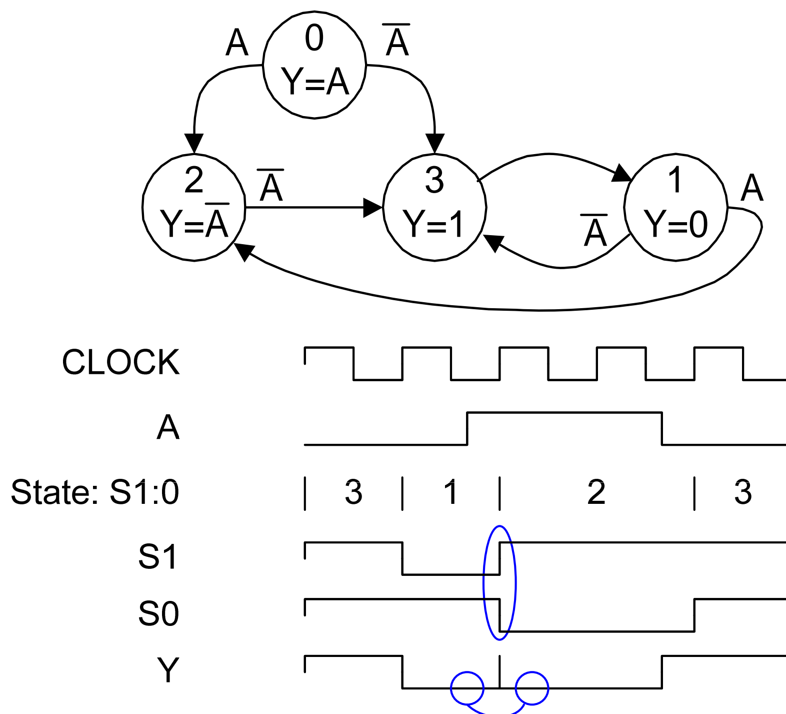


- Outputs written on an arrow apply to the state *emitting* the arrow.
- Outputs still apply for the entire time spent in a state
- This does not affect the Moore/Mealy distinction
- This is a notation change only

Output Glitches

When making a transition from one state to another, the logic is likely to generate a glitch on an output if:

- two or more state bits change
- the output has the same value in both states
- some combination of the changing state bits would cause the output to change

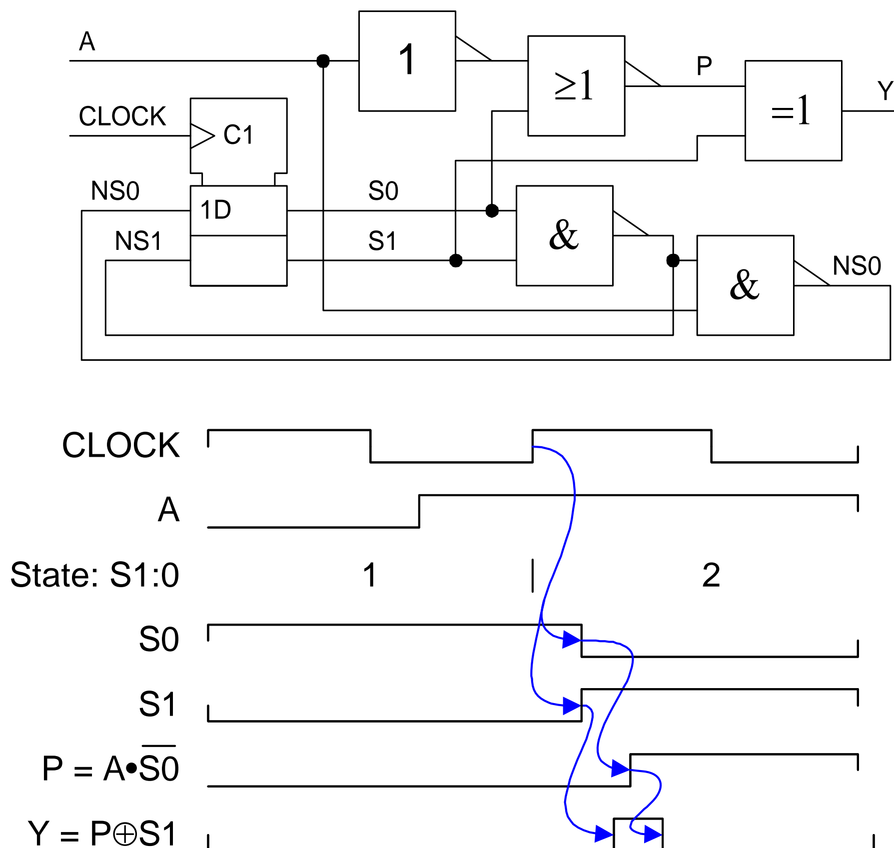


In changing from state 1 to state 2:

- the two states differ in both S0 and S1
- the output is low in both states
- if S0 and S1 both went high then the output would change.

Cause of Output Glitches

Look in detail at the logic when going from state 1 to 2:



The two inputs to the XOR gate (P and S1) are meant to change simultaneously.

In fact S1 changes first because of the delay through the NOR gate.

The XOR gate “sees” the effect of S1 changing before it “sees” the effect of S0 changing. It is as if we went briefly into state 3.

Quiz Questions

1. What is the definition of a Moore machine?
2. What does it mean if an arrow in a state diagram has no Boolean expression attached to it?
3. To which state does an output value refer when it is marked on an arrow in a state diagram? Is it the state the arrow points *towards* or the state the arrow points *away from*?
4. Is the next state determined by the value that the input signals have just *before* or just *after* the CLOCK↑?
5. If transitions from a state to itself have been omitted from a state diagram, how can you tell when such a transition occurs?
6. What are the three conditions that give rise to output glitches?

Answers are all in the notes.

Lecture 9

Synchronous State Machine Design

Objectives

- To learn how to design a state machine to meet specific objectives
- To understand when two or more states are equivalent and can be merged into a single state.
- To understand the principles of assigning state numbers
- To appreciate when it is necessary to synchronise a state machine’s inputs with the CLOCK
- To understand how a state machine is implemented using programmable logic

Designing a Synchronous State Machine

The state is the only way the circuit can remember what happened in the past.

The number of states required equals the number of past histories that the circuit needs to distinguish.

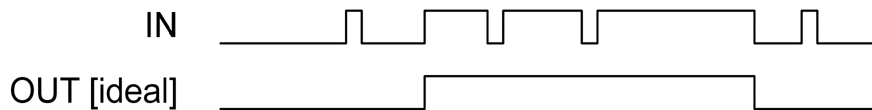
General Design Procedure

- Construct a sequence of input waveforms that includes all relevant situations.
- Go through the sequence from the beginning. Each time an input changes, you must decide:
  - branch back to a previous state if the current situation is materially identical to a previous one
  - create a new state otherwise
- For each state you must ensure that you have specified:
  - which state to branch to for every possible input pattern
  - what signals to output for every possible input pattern

Designing a Noise Pulse Eliminator

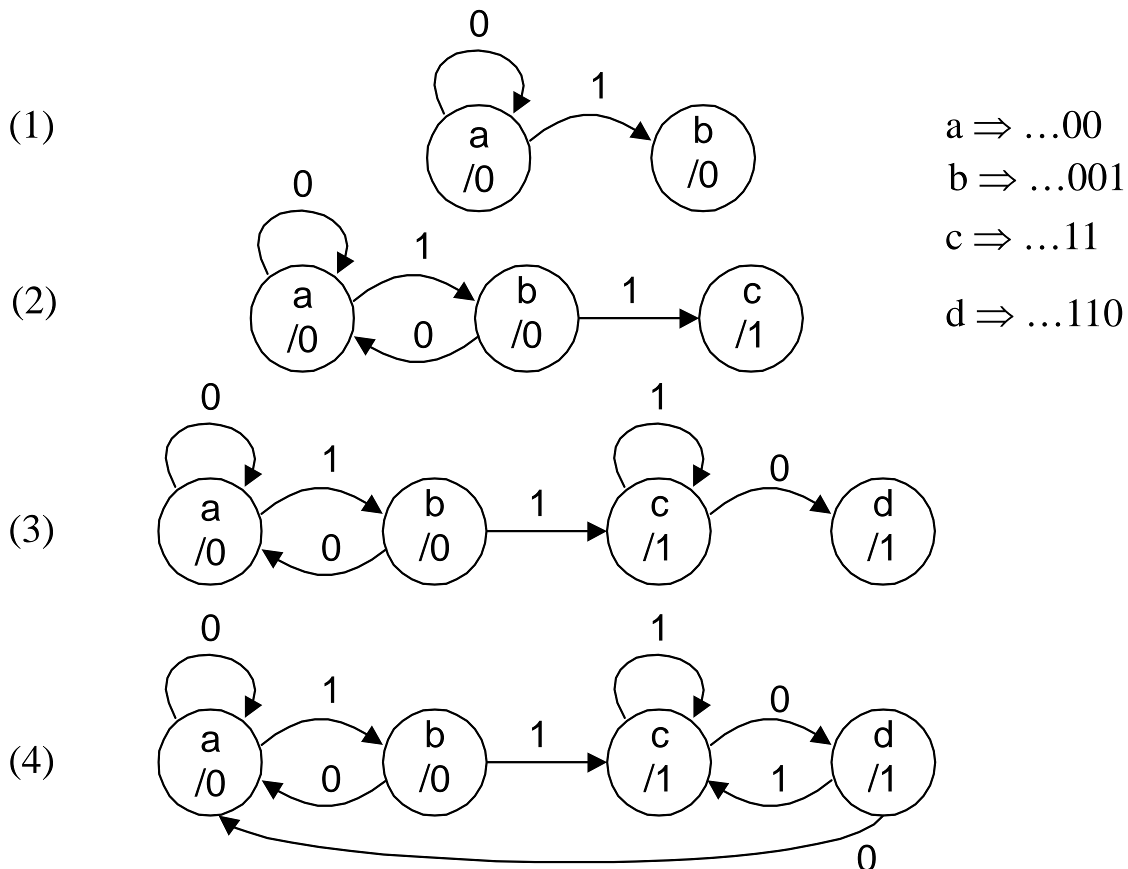
Design Problem: Noise elimination circuit

– We want to remove pulses that last only one clock cycle



- Use letters a,b,… to label states; we choose numbers later.
- Decide what action to take in each state for each of the possible input conditions.
- Use a Moore machine (i.e. output is constant in each state). Easier to design but needs more states & adds output delay.

Assume initially in state "a" and IN has been low for ages



Explanatory Notes

(1) If IN goes high for two (or more) clock cycles then OUT must go high, whereas if it goes high for only one clock cycle then OUT stays low. It follows that the two histories "IN low for ages" and "IN low for ages then high for one clock" are different because if IN is high for the next clock we need different outputs. Hence we need to introduce state b.

(2) If IN goes high for one clock and then goes low again, we can forget it ever changed at all. This glitch on IN will not affect any of our future actions and so we can just return to state a.
If on the other hand we are in state b and IN stays high for a second clock cycle, then the output must change. It follows that we need a new state, c.

(3) The need for state d is exactly the same as for state b earlier. We reach state d at the end of an output pulse when IN has returned low for one clock cycle. We don't change OUT yet because it might be a false alarm.

(4) If we are in state d and IN remains low for a second clock cycle, then it really is the end of the pulse and OUT must go low. We can forget the pulse ever existed and just return to state a.

Each state represents a particular history that we need to distinguish from the others:

(a) IN=0 for >1 clock (b) IN=1 for 1 clock

(c) IN=1 for >1 clock (d) IN=0 for 1 clock

Equivalent States

An initial design often creates more states than are necessary.

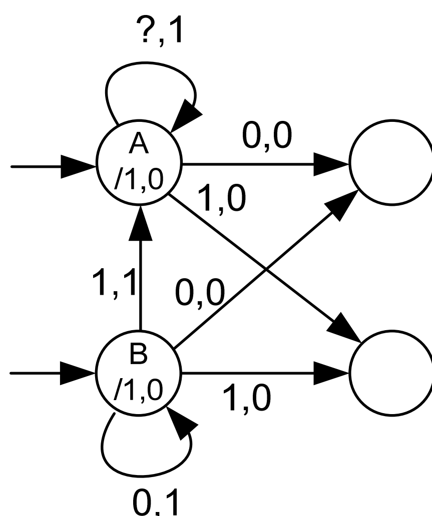
States A and B are said to be equivalent if, for any possible input sequence, you get identical output waveforms regardless of whether the initial state is A or B.

You can simplify a state machine by merging equivalent states into a single state.

Two states are definitely equivalent if:

– They have the same outputs for every possible input combination.
– They have the same next state for every possible input combination (assuming they themselves are equivalent).

This rule won't always find all possible equivalent states and so won't necessarily make the state machine as simple as possible (you will learn a complete rule next year).



States A and B are equivalent

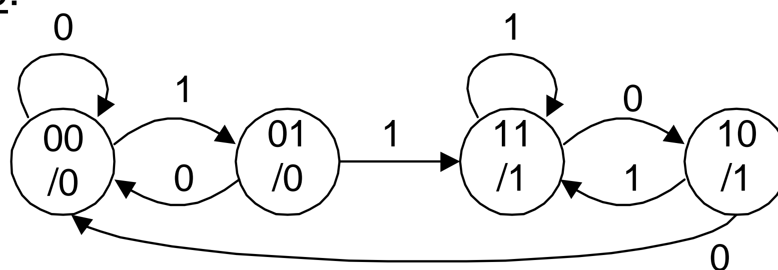
Implementing a State Machine

Assign each state a unique binary number. Your choice affects circuit complexity but the circuit will work correctly whatever choice you make.

State Assignment Guidelines:

- Any outputs that depend only on the state should if possible be used as some of the state bits.
- Assign similar (=most bits the same) numbers to states (a) that are linked by arrows, (b) that share a common destination or source, (c) that have the same outputs.
- If two subsets of the state diagram have identical transitions with identical input conditions, they should be numbered so that corresponding states have similar numbers.

Example:



State Numbers: S1,S0
Inputs/Outputs: IN/OUT

- S1 is the same as OUT (from the first guideline)
- All states linked by arrows differ in only one bit (from the second guideline)

Implementing a State Machine (contd)

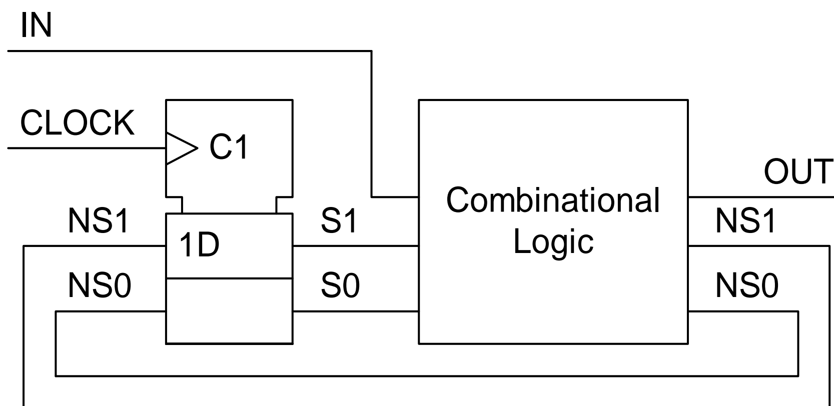
Now we can draw a Karnaugh map (really three K-maps in one) giving NS1, NS0 and OUT in terms of S1, S0 and IN:

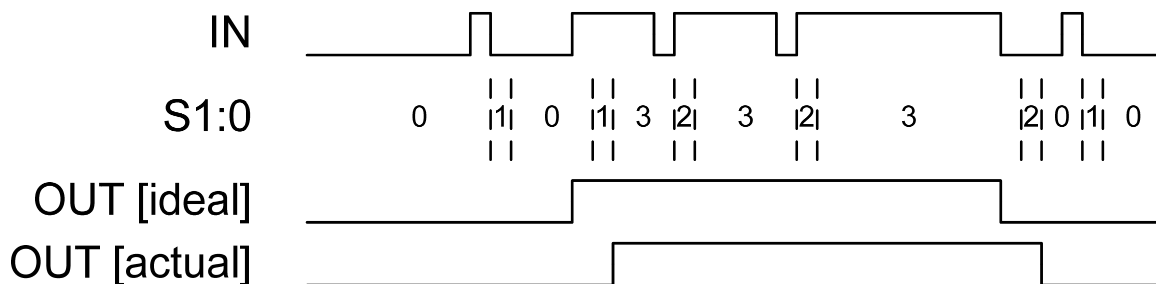
NS1,NS0/OUT

S1,S0	IN=0	IN=1
00	00/0	01/0
01	00/0	11/0
11	10/1	11/1
10	00/1	11/1

From this we can derive Boolean expressions for the combinational logic block:

$$NS1 = IN \cdot (S1 + S0) + S1 \cdot S0 \qquad NS0 = IN \qquad OUT = S1$$

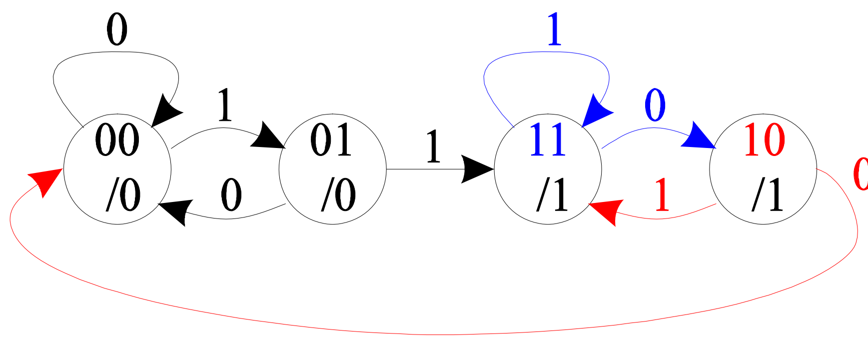


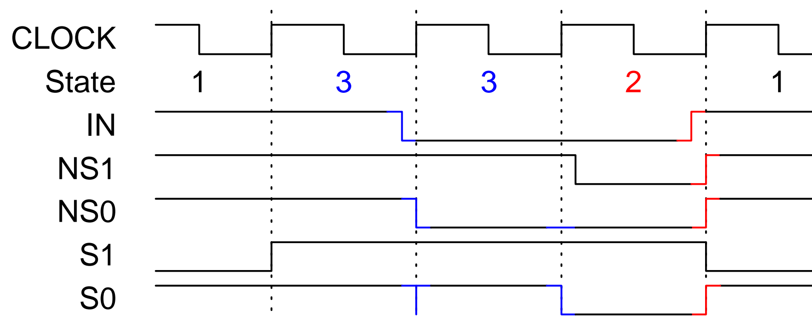


Unsynchronised Inputs

An input transition just before CLOCK ↑ can cause the NS bits to change within the setup/hold window of the register.

If k of the NS bits change we might go to any of 2^k states:





State 3:

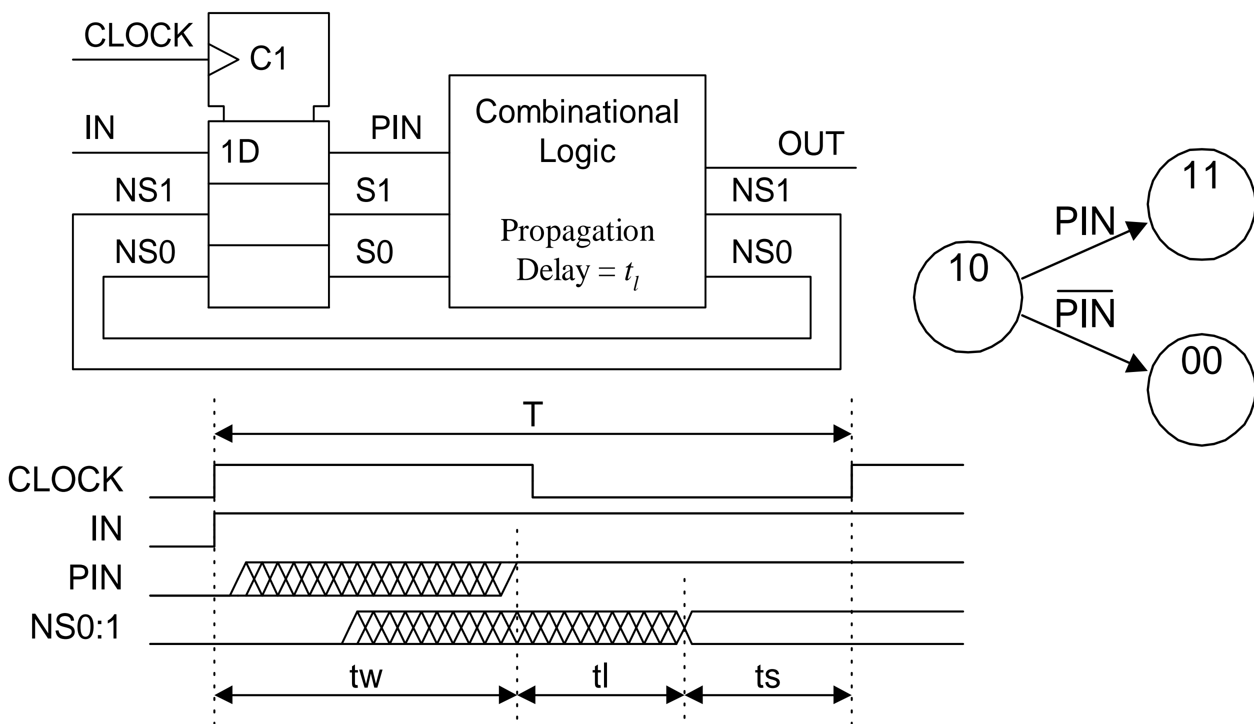
IN ↓ causes NS0:1 to change from 11 to 10 ⇒ $k=1$.
NS0 ↓ too late for S0 but causes glitch on S0
S0 goes low on next CLOCK ↑. Everything is OK.

State 2:

IN ↑ causes NS0:1 to change from 00 to 11 ⇒ $k=2$.
NS0↑ changes in time so S0 → 1.
NS1↑ changes too late so S1 → 0.
Next state is 01 which is an ILLEGAL destination.

Input Synchronization

- An asynchronous input <u>must</u> be synchronized if in any state it affects more than one of the next state bits.
- Inputs can be synchronized by passing them through a register before they go to the combinational logic:



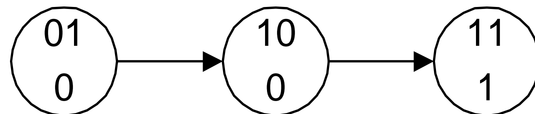
  - Here IN must be synchronized because destinations 11 and 00 differ in more than 1 bit position
  - IN might change within setup-hold window
  - PIN (Previous IN) will be stable t_w after CLOCK ↑ Typical t_w is 25ns for MTBF of 1000 years
  - NS1:0 will be stable t_w+t_l after CLOCK ↑
  - CLOCK period (T) must be greater than $t_w+t_l+t_s$ for reliable operation
  - To get a huge MTBF, send PIN through a 2nd register

Input Sync versus Output Glitches

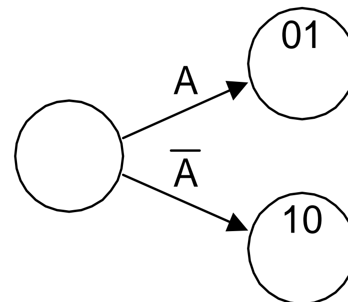
Do not confuse two different problems:

Output glitches are likely if three conditions are true:

- *two consecutive states* differ in more than one bit position
- output is the same in both states
- changing only some of the state bits would cause an output change



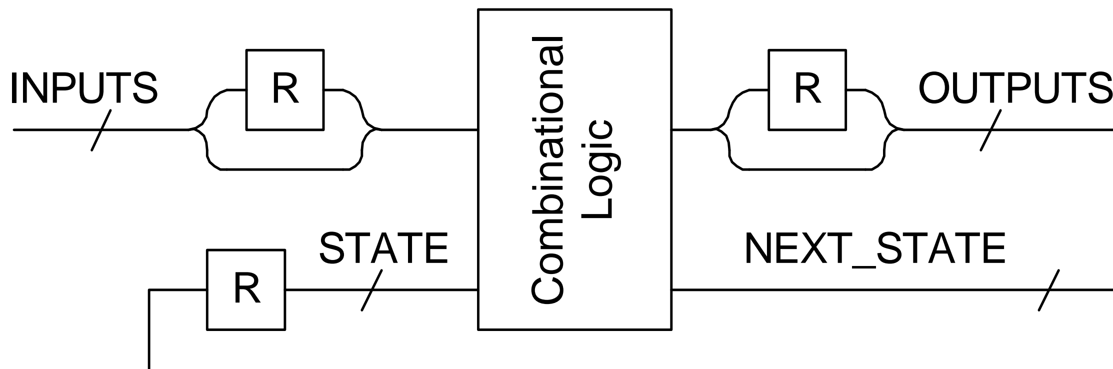
Input synchronisation is needed when *two alternative destinations* differ in more than one bit position.



This is a far more serious problem as it results in the wrong state sequence.

In both cases the solution is to send the offending input or output signal through a register/flipflop.
(This adds a 1-cycle delay).

Universal State Machine Circuit Diagram



- "R" denotes register bits: **all with the same CLOCK**
- **Inputs** can go directly into logic block if they are already synchronized with CLOCK. Others must be passed through a register unless (i) they only affect one bit of the Next_State and (ii) the logic block is hazard-free.
- Glitch-prone **outputs** must be deglitched if they go to a clock or to an asynchronous set/reset/load input.
  - For some state diagrams it is possible to eliminate output glitches by clever state numbering.
- Input synchronization and output deglitching add circuitry and increase input-to-output delays. Avoid if unnecessary.

Quiz Questions

1. What problem can arise if two alternative next states differ in more than one bit position?
2. What problem can arise if two consecutive states differ in more than one bit position?
3. What determines the minimum number of states needed by a state machine to solve a particular problem?
4. What aspects of a state machine’s operation are affected by the assignment of state numbers?
5. Under what conditions can a group of states be merged into a single state?

Answers are all in the notes.