Computer Systems

Lecture 5

Synchronous Circuits

Dr José Cano, Dr Lito Michala
School of Computing Science
University of Glasgow
Spring 2020

Outline

- Timing: gate delay, hazards and glitches
- Delay flip flop
- Registers
- Simultaneous update of state

Two issues we must deal with

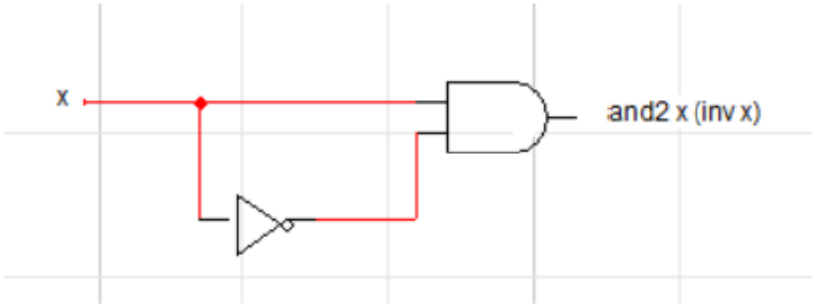
- **Gate delays**: physical devices take time to produce correct outputs
- **State**: circuits need to have internal memory, and logic gates don't provide this
- We will solve these issues using
  - A new component: the delay flip flop (d) provides memory
  - A new way to organise circuits: *synchronous circuits* simplify timing

Time and gate delays

- So far, we have been ignoring time as a circuit runs
- Whenever an input to a logic gate changes, the gate takes a little time to respond and make its output valid
- This is the gate delay,small but not zero
- Generally, circuits contain many logic gates, and the gate delays add up
- We need to allow the logic gates enough time to calculate the correct result
  - Timing is difficult: hazards
  - We can make timing easy: synchronous circuits

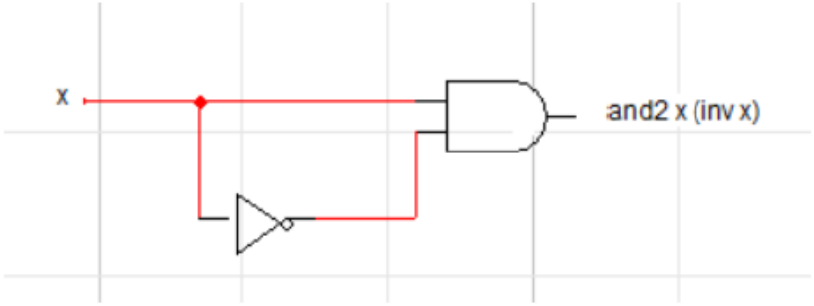
Hazards

- Strange effects can occur when a circuit's inputs become stable at different times (which is common)

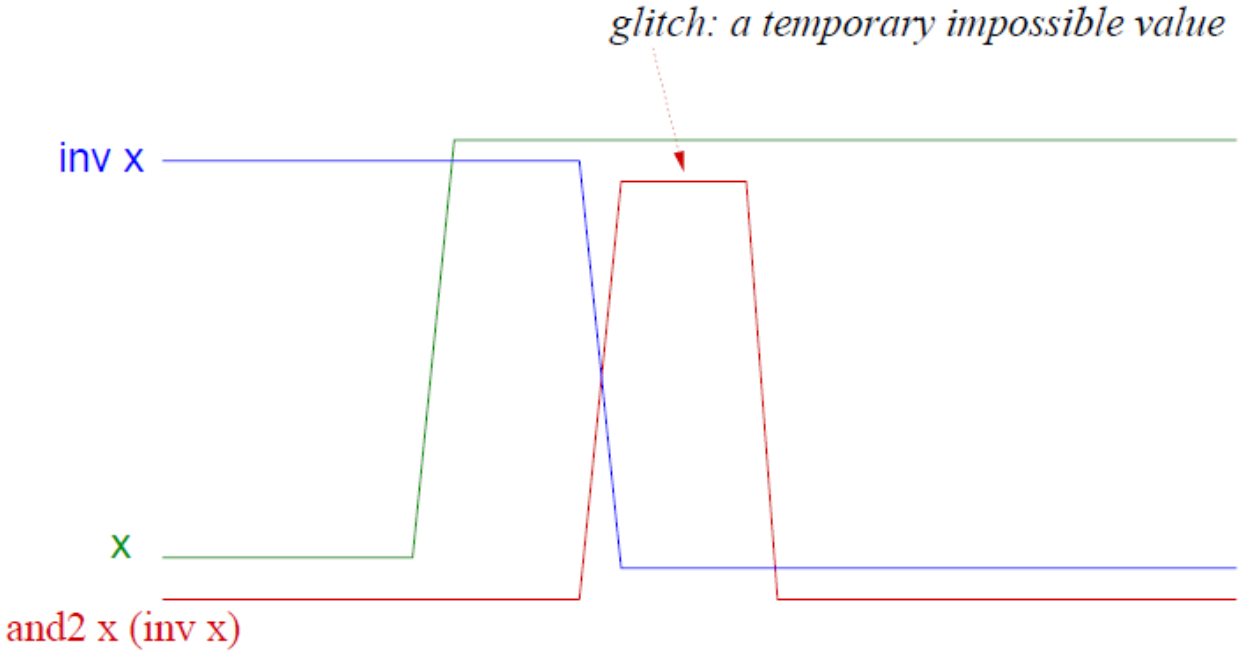


- Based on Boolean algebra, we can transform this mathematically to y = zero
  - However, if we draw a graph showing the signal values as a function of time, we see that when x changes from 0 to 1, the output of inv x takes time to change from 1 to 0
  - As a result, for a little while both of the inputs to the and2 gate are 1, and the output will be 1 momentarily
  - The hardware is not always obeying the laws of Boolean algebra!

A glitch: and2 x (inv x) = 1 momentarily is wrong!



- **Glitch**: for a short time, the output of the circuit is incorrect, but if you wait a short time, it will reach the correct value



Outline

- Timing: gate delay, hazards and glitches
- Delay flip flop
- Registers
- Simultaneous update of state

State

- State means something like “memory”
  - The values of all the memory locations and registers are a point in time
- A circuit consisting only of logic gates is called combinational
- A combinational circuit has no state: its output depends on its input (after gate delays)
- **Problem**: we need to
  - Have a way to hold state: a component that can “remember” a bit
  - Keep the circuit synchronized: the gate delays in different parts of a circuit are likely to be different, and this could lead to confusion
- **Solution**: the dff and the clock

The delay flip flop (dff)

- A major new primitive component!
  - We will introduce state (memory) with an explicit memory device, the delay flip flop dff
  - A dff is a circuit that remembers one bit of data (the “state”)
  - There is a data input x carrying a value to be remembered, and a data ouput y conveying the state value
  - There is also a clock input which “ticks” regularly

The clock

- The dff component copies its input value into its state at specific points in time, determined by a clock signal

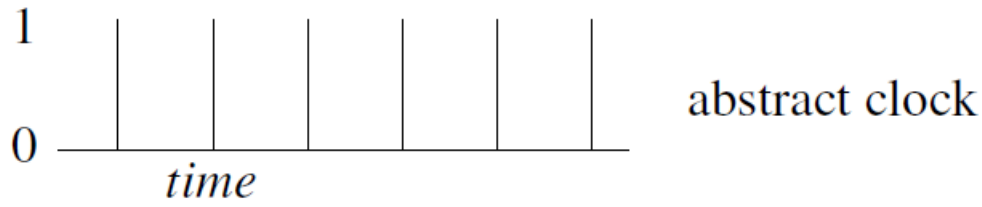
- The behaviour is
  - To output the state continuously
  - To execute state := input whenever a clock tick occurs

- There is just one clock signal, which is sent to all flip flops
  - Every flip flop updates its state at the same time

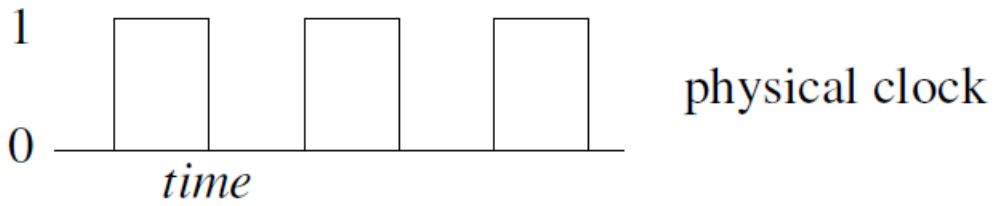
- The clock is generated externally by a circuit that produces a fast and steady sequence of clock ticks

Physical and abstract clock

- Abstractly, the clock is always zero except at a regular sequence of instantaneous points in time, called clock ticks, when it is one



- The abstract clock cannot be built electronically, but the same effect can be obtained with a square wave clock
  - The dff components treat a rising edge as the clock tick



Synchronous circuits

- A circuit is *synchronous* if
  - Every flip flop is connected directly to a unique global clock
  - No logic functions are performed on the clock signal
  - The circuit is designed so that every clock tick reaches each flip flop simultaneously (a little bit of variation is ok, but not much)
  - Every feedback loop in the circuit passes through a flip flop (no feedback in pure combinational logic)
  - The inputs to the circuit are assumed to remain stable throughout an entire clock cycle

Clock Cycles

- Clock ticks are points in time, and clock cycles are intervals of time between two ticks
- At a clock tick, the flip flops get new states which remain stable through the cycle, and the inputs get new values which remain stable
- During the clock cycle, the combinational logic settles down, and eventually all the signals become valid
- Then the clock ticks

The clock must be run slowly enough to ensure that all signals become valid!

Clock speed

- Number of ticks per second is measured in Hz (Hertz)
- A typical computer has about 3 billion ticks per second: 3GHz
- The duration of a clock cycle is the time between ticks: about 1/3 ns (nanosecond)

Outline

- Timing: gate delay, hazards and glitches
- Delay flip flop
- Registers
- Simultaneous update of state

Flip flops and registers

- A flip flop remembers a bit for a very short time, just one clock cycle

- A register remembers a bit until you tell it to load a new value, so its memory can last a long time

The 1-bit register reg1

- Remembers a bit
- Receives a control input *ld* (“load”) and a data input x, and outputs a 1-bit state
- Initial value is zero
- At each clock tick, the register loads the data value x if ld == 1
  - If ld == 0 the register ignores the value of x and just retains its previous state
- Throughout the clock cycle, the register outputs its state value, which will not change (until perhaps the next tick)

The registers in a computer are constructed from many copies of the reg1 circuit

Designing the 1-bit register

- A dff is needed to hold the state

- Since the dff will load a new state value at every clock tick, we need to use combinational logic to determine what that value should be
  - If ld == 1 then the input to the d should be the data input x
  - Otherwise, the input to the d should just be its old state value

- Therefore

  dff_input = (if ld=0 then old_state else x)

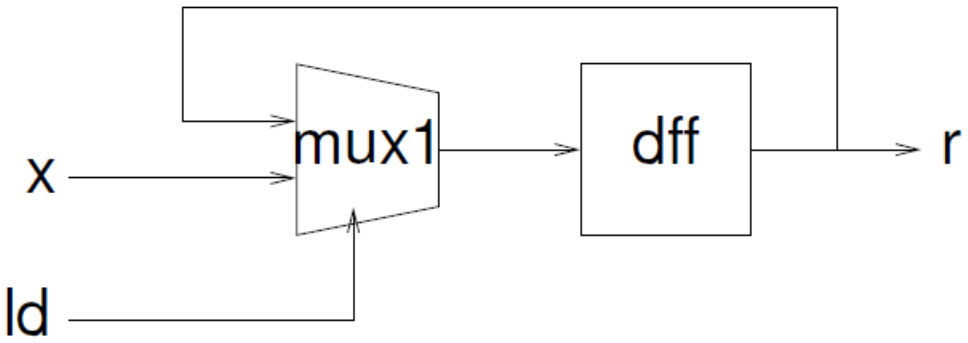
- The conditional is implemented with a multiplexer

  dff_input = mux1 ld old_state x

reg1: the 1-bit register

- Inputs
  - A control input ld (“load”)
  - A data input bit x

- Output
  - The state of the register r



Simulation Tables

- A simulation table provides a systematic way to calculate and display the execution of a synchronous digital circuit
  - It consists of an unbounded sequence of rows, each row corresponds to a clock cycle
  - Every time there is a clock tick, a new row is added to the bottom of the table
  - Every signal of interest has a column, giving its values through time

Simulating the register

- The simulation table will show the inputs, the state, and the internal signals for each clock cycle

- Assume the initial state of the flip flop is “?”
  - The circuit should never use this value!

- At the start of each clock cycle, the values of the input signals are established by the “outside world”, and remain stable for the entire cycle

Cycle 0: receive inputs

- Before the first tick
- Initial state of the flip flop is ?
  - This is also the value of signal r
- Inputs are given: ld=1 and x=1

cycle	inputs		state	internal
	ld	x	r	dff_input
0	1	1	?	

Cycle 0: calculate internal signals

- Calculate

  dff_input

  = mux1 ld r x

  = mux1 1 ? 1

  = 1

- At the clock tick we will use *dff_input* but not r
  - Therefore the undefined initial value of the flip flop will be thrown away, and won't cause problems

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1

Clock tick 1: update flip flops

- Tick ends cycle 0 and begins cycle 1
- The new horizontal line symbolizes the tick
- State of flip flop is replaced by value of its input signal d input

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1			1	

Cycle 1: receive inputs

- Given: ld = 1 and x = 0

- This is a command to load 0 into the state

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	

Cycle 1: calculate internal signals

- Calculate

  dff_input

  = mux1 ld r x

  = mux1 1 1 0

  = 0

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0

Clock tick 2: update flip flops

- Input to flip flop is $dff\_input_1 = 0$

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0
2			0	

Cycle 2: receive inputs

- Given: ld = 0 and x = 1

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0
2	0	1	0	

Cycle 2: calculate internal signals

- Calculate

  dff_input

  = mux1 ld r x

  = mux1 0 0 1

  = 0

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0
2	0	1	0	0

Clock tick 3: update flip flops

- The state r becomes $dff\_input_2 = 0$

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0
2	0	1	0	0
3			0	

Cycle 3: receive inputs

- Given ld = 1 and x = 1

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0
2	0	1	0	0
3	1	1	0	

Cycle 3: calculate internal signals

- Calculate

  dff_input

  = mux1 ld r x

  = mux1 1 0 1

  = 1

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0
2	0	1	0	0
3	1	1	0	1

Clock tick 4: update flip flops

- The state r becomes $dff\_input_3 = 1$

cycle	inputs		state	internal
	ld	*x*	*r*	*dff_input*
0	1	1	?	1
1	1	0	1	0
2	0	1	0	0
3	1	1	0	1
4			1	

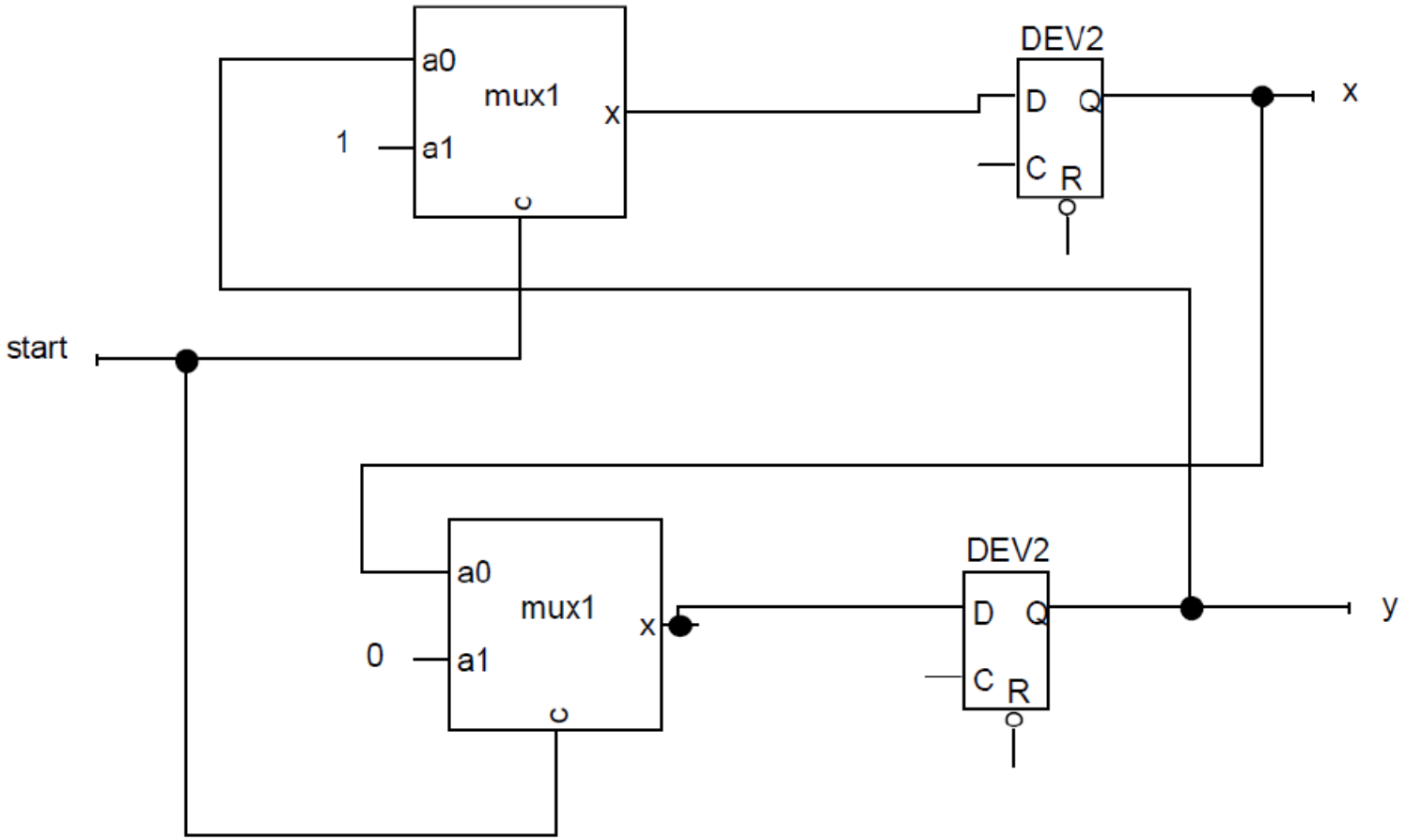
Outline

- Timing: gate delay, hazards and glitches
- Delay flip flop
- Registers
- Simultaneous update of state

Simultaneous update of state

- The state of a circuit consists of the states of all the flip flops
- In a programming language, you often think of executing a sequence of statements, where a statement will modify one variable and leave all the others unchanged
- In a digital circuit, all the flip flops change their state simultaneously at the clock tick
- This is the big difference between hardware and software

A circuit with feedback and two flip flops



- At each clock tick it swaps the values of x and y

Modeling state with parallel assignment

- The values of x and y are updated simultaneously at the clock tick
- You can think of the flip flop states as variables
- But the circuit does not correspond to a sequential program
- It acts like parallel assignments, which are supported in some programming languages
  - All the right hand sides are evaluated in parallel
  - Then all the variables are updated in parallel

```
begin parallel
        { x = dff (mux1 start y one)
          y = dff (mux1 start x zero)
end parallel
```

The effect of parallel assignment

- To run the circuit, set the control input start to 1 for a clock cycle, then leave it 0 while the circuit executes

- If the control signal start is 1, then x initialises to 1 and y initialises to 0

- As long as start is 0, the flip flops keep exchanging their values

Cycle	start	x	y
0	1	?	?
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
...	...	...	...

THE AUTHOR OF THE WINDOWS FILE COPY DIALOG VISITS SOME FRIENDS.

https://xkcd.com/612/