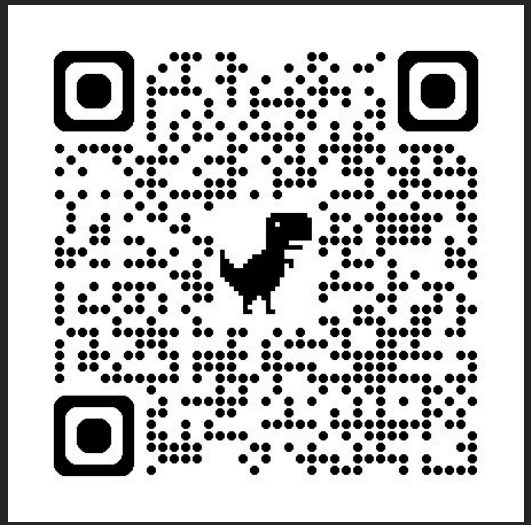
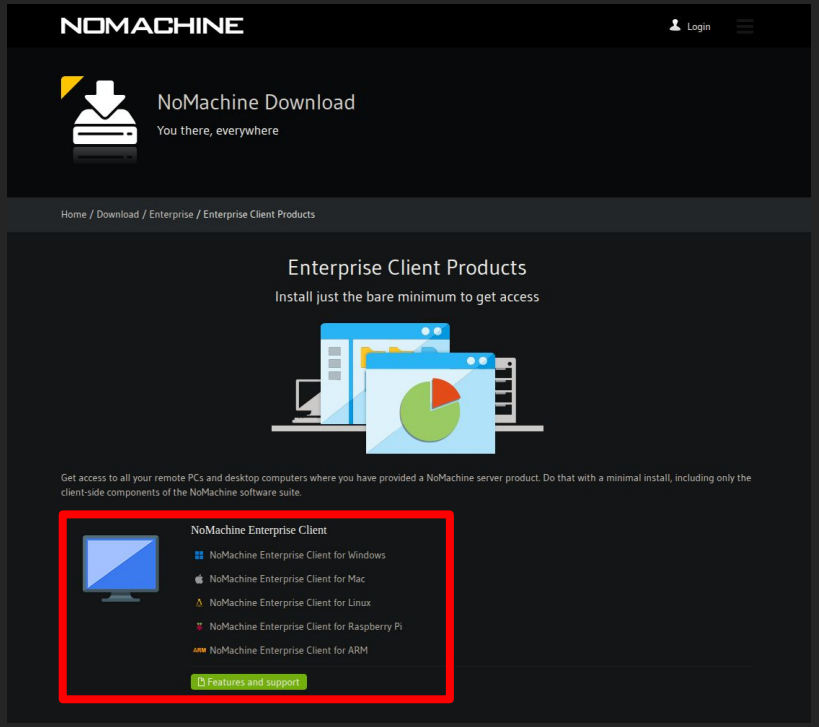
Required Software

Download **NoMachine Enterprise Client** on your laptop





Design Your Own CPU

Session 1 – Intro to SystemVerilog

Session 1 – Intro to SystemVerilog

Session 2 – The single-cycle CPU

Session 3 – Optimisation and bare-metal programming

About Me

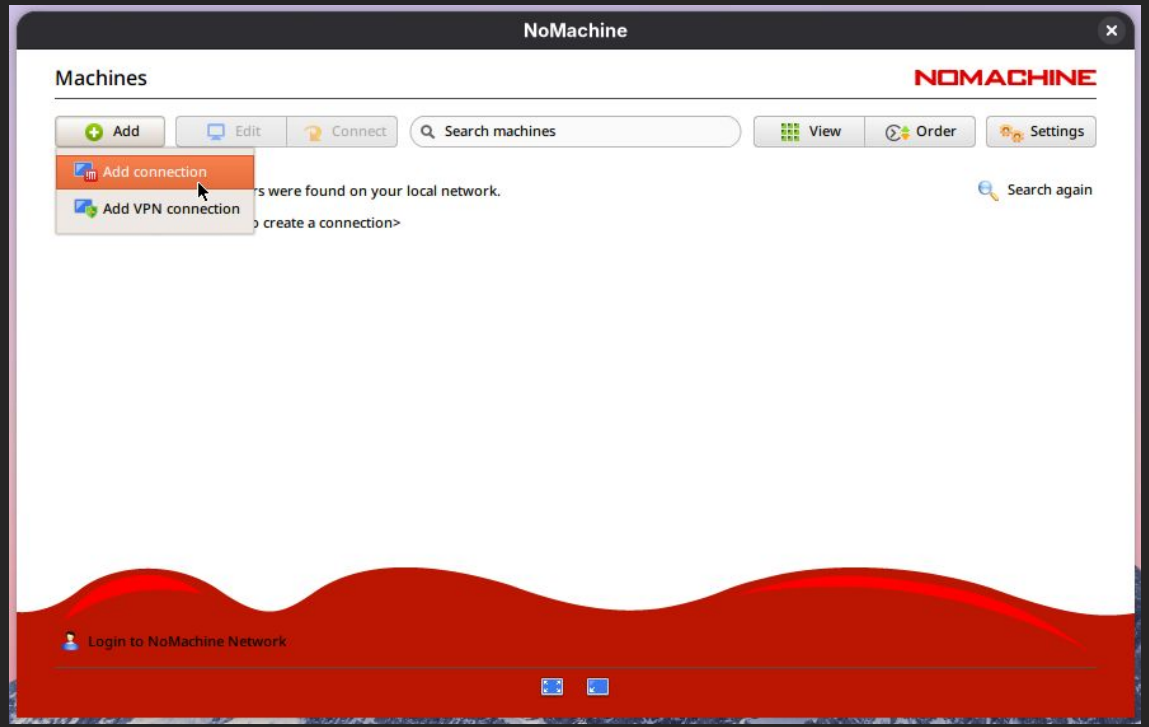
- Part IV Electronic Engineering with Computer Systems
- IP: *A Universal Superscalar Processor Design for MIPS and RISC-V*
- GDP: *Verify a RISC-V Core in 10 Weeks*
- R&D Intern at Codasip (Fabless RISC-V IP firm)

Reasons for Attending These Workshops

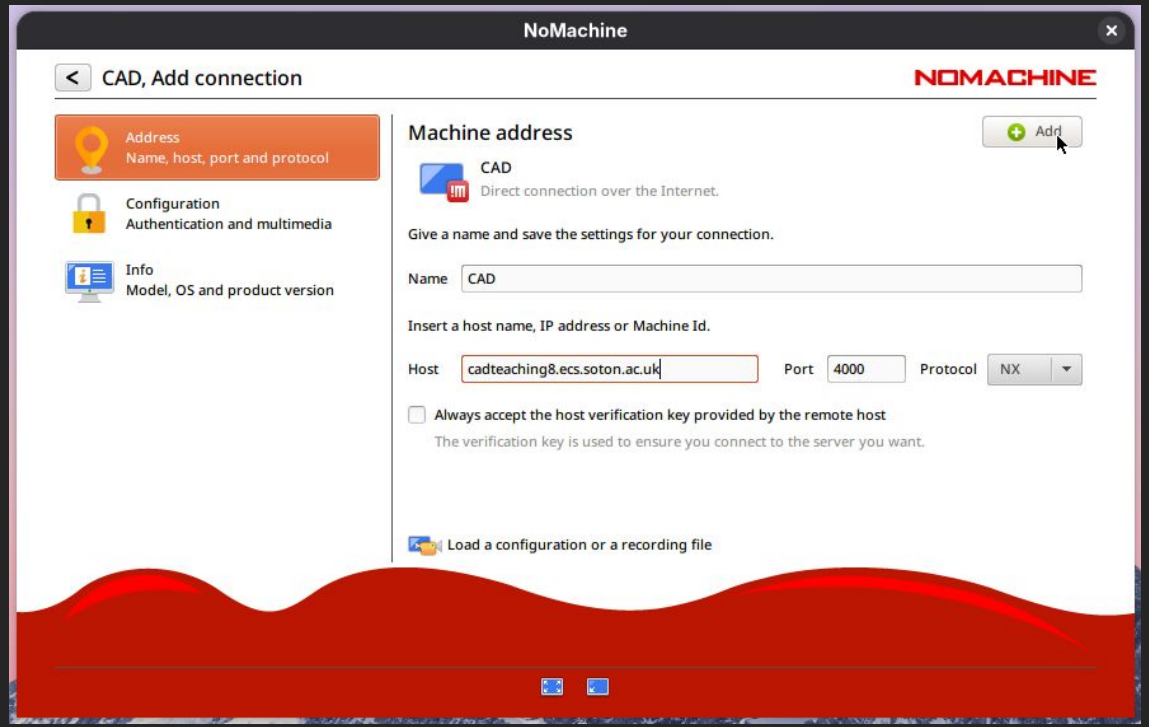
- You're interested in doing a digital design focused IP or GDP
- You want an internship in digital design
- You want to learn or have a refresher on SystemVerilog
- You want to know how digital systems are designed
- For fun

Setup

Setup (1)


NoMachine
Machines
NOMACHINE
Add
Edit
Connect
Search machines
View
Order
Settings
Add connection
Add VPN connection
s were found on your local network.
create a connection>
Search again
Login to NoMachine Network

Setup (2)


NoMachine
CAD, Add connection
NOMACHINE
Address
Name, host, port and protocol
Configuration
Authentication and multimedia
Info
Model, OS and product version
Machine address
Add
CAD
Direct connection over the Internet.
Give a name and save the settings for your connection.
Name
CAD
Insert a host name, IP address or Machine Id.
Host
cadteaching8.ecs.soton.ac.uk
Port
4000
Protocol
NX
Always accept the host verification key provided by the remote host
The verification key is used to ensure you connect to the server you want.
Load a configuration or a recording file

Setup (3)


CAD - NoMachine
CAD
NOMACHINE
You are now connected to the requested machine. You now have to login with a system account. Please type your username and password to login.
Username
th7g22
Password
Save this password in the connection file
Cancel
OK

Setup (4)


CAD - NoMachine
CAD
NOMACHINE
My desktops
New desktop
Create
Find a type
View
Order
Create a new virtual desktop
Create a new custom session
<Connect to an existing desktop>
Save this setting in the connection file
Logged in as user th7g22
TERMINAL SERVER

Setup (5)

- In your terminal, clone this Git repository:

https://github.com/bhart17/design-your-own-cpu

- `git clone https://github.com/bhart17/design-your-own-cpu.git`
- `cd design-your-own-cpu`

Digital Systems

Digital Systems

- Digital systems are systems that operate on *binary signals*
- An electronic digital system typically uses two different voltages to represent these binary signals (low voltage: ‘0’; high voltage; ‘1’)
- Not analogue or mixed-signal, though there is an overlap in methodology and technology

Stateful and Stateless Systems (1)

- Stateless system: the output is a function of the inputs

$$Y = f(A)$$

- Stateful system: the output is a function of the inputs *and the current state*

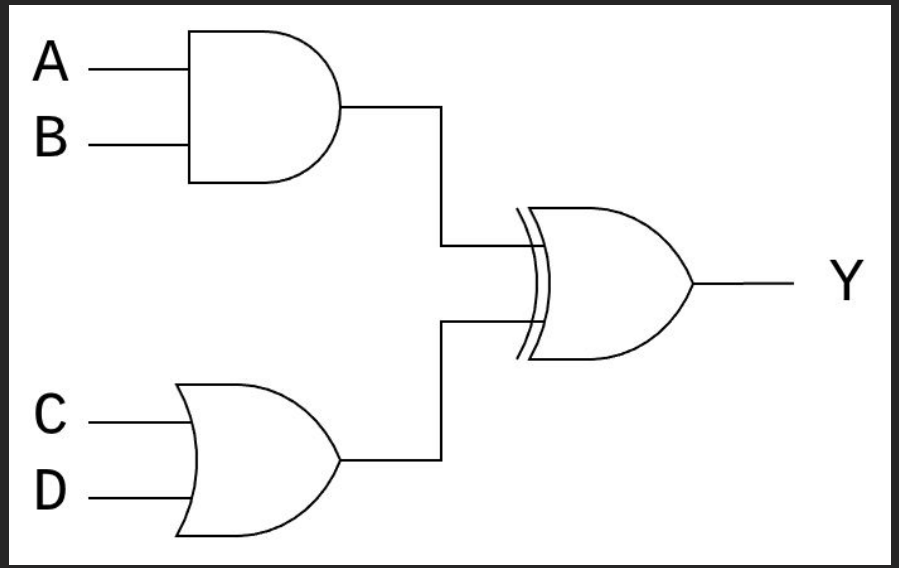
$$Y = f(A, \text{State}_{\text{current}})$$

$$\text{State}_{\text{next}} = g(A, \text{State}_{\text{current}})$$

- All complex digital systems are stateful

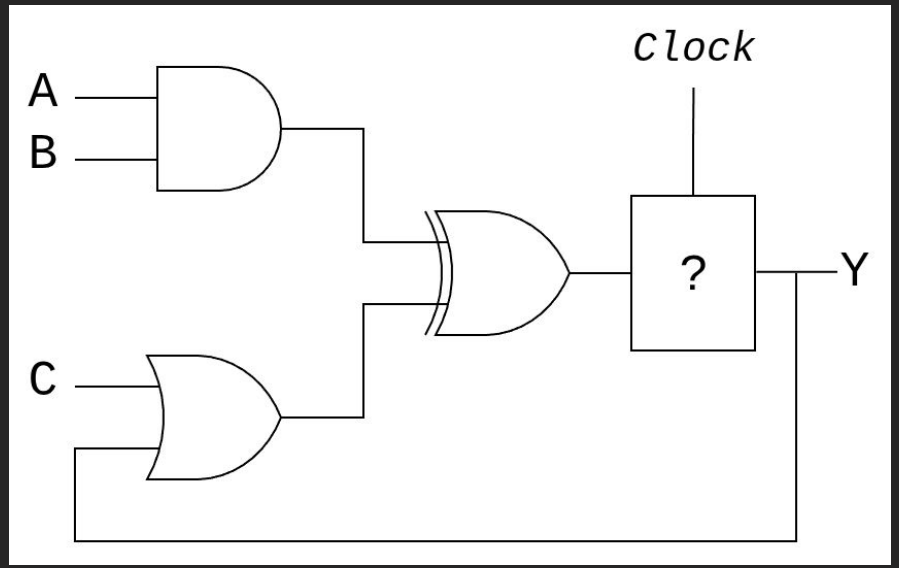
Stateful and Stateless Systems (2)

- A stateless system can be modelled as some set of *combinational* boolean gates



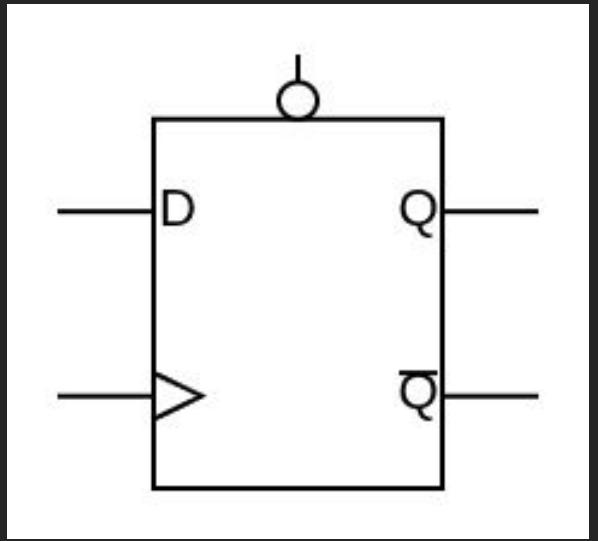
Stateful and Stateless Systems (3)

- A stateful system needs some way of storing state between *cycles*



Flip Flops

- A means of storing digital state
- Has two *stable* states (‘1’ and ‘0’)
- Takes a new state upon a *clock edge*



Overall: Designing digital systems

- We want to model systems that take in a number of binary inputs
- They compute outputs as a function of these inputs **and** the state of the system using boolean logic
- State is updated on a clock edge

Levels of Abstraction

Abstraction

- A system can be described (or *modelled*) in many ways
- Each is a balance between a high-level description versus a logical or physical description
- We want to *abstract* the physical details of a system away when we are designing it

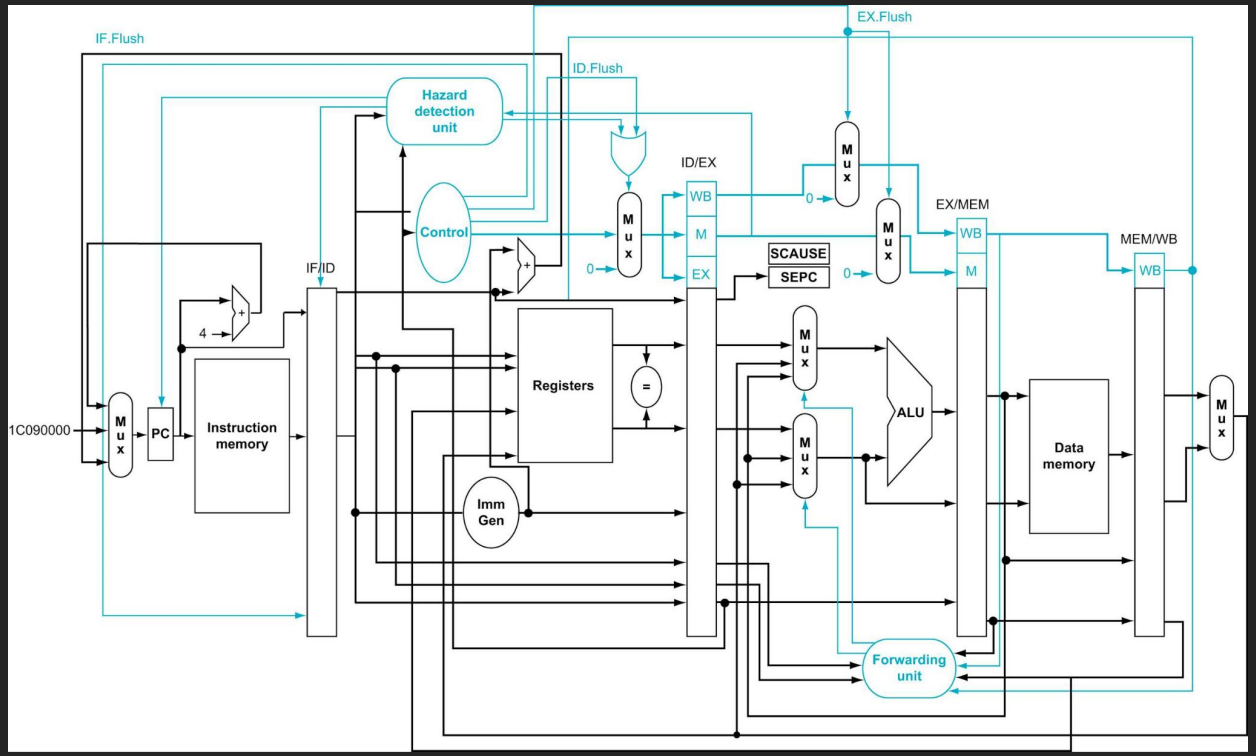
Levels of Abstraction for a Digital IC (1)

1. Specification
2. Block diagram
3. ???
4. Gate-level circuit *(Generic cells)*
5. Transistor-level circuit *(Specific cells)*
    5.1. Transistor circuit diagram
    5.2. Stick diagram
    5.3. Mask diagram

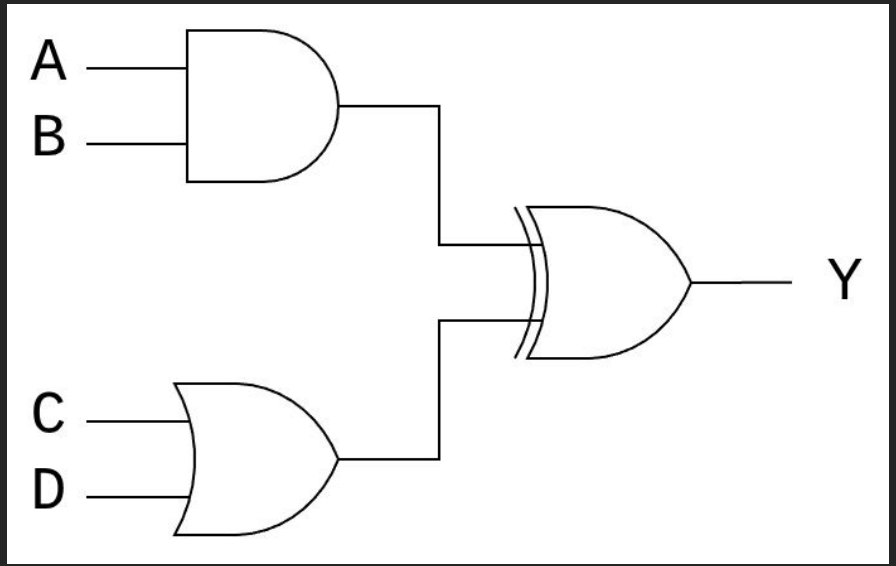
Specification

‘The design should do X and Y but not Z.’

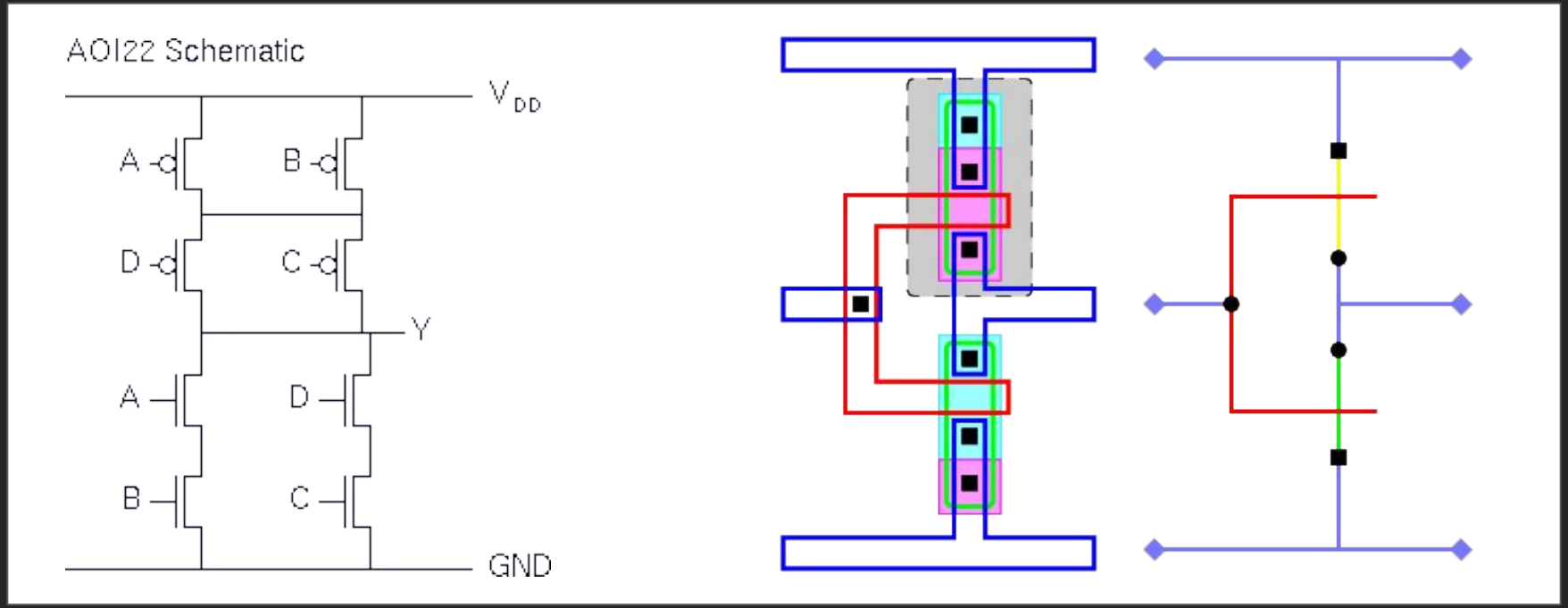
Block Diagram



Gate-level Circuit



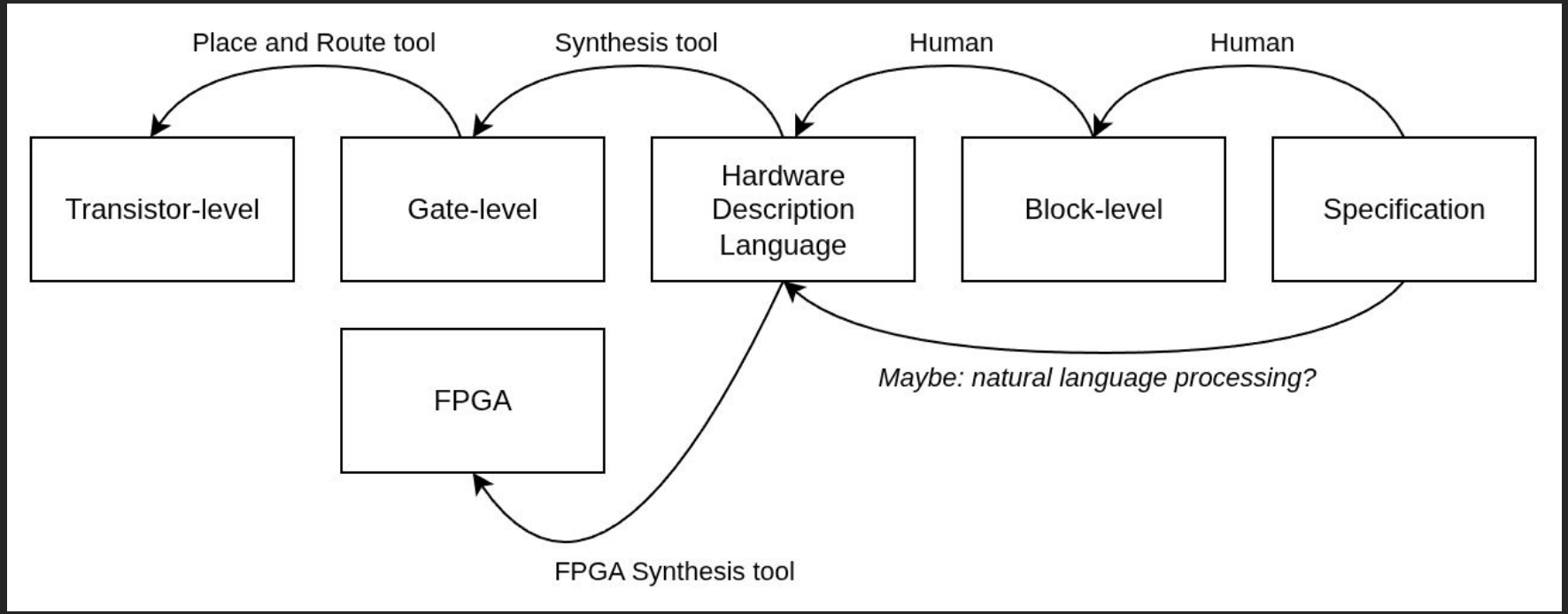
Trasistor-level Circuit



Levels of Abstraction for a Digital IC (2)

1. Specification
2. Block diagram
3. Hardware description language (such as SystemVerilog)
4. Gate-level circuit *(Generic cells)*
5. Transistor-level circuit *(Specific cells)*
    5.1. Transistor circuit diagram
    5.2. Stick diagram
    5.3. Mask diagram

IC Design Flow



SystemVerilog

Hardware Description Languages

- Aiming to describe the logic between an input and output in a formal but high-level way
- Readable by humans but understandable to computers
- Historically called Register Transfer Layer – describing how data flows between *registers*
- Not programming languages!

SystemVerilog

- Defined in IEEE-1800
- Superset of Verilog with the main purpose of adding useful *verification* features
- Most common HDL used today in industry

Purposes of Writing HDL

- When creating an IC: we will want to provide our high-level SystemVerilog to a synthesis tool that will map our design to boolean gates
- However, we want to know if our design works before this stage
- A *simulator* is used to see how it performs under certain inputs
- May seem very close to a programming language – remember, it isn't!

Testbenches

- We write *synthesisable* modules to describe the hardware we want to fabricate
- Then, we can provide synthetic inputs and test patterns to our design in a *testbench*
- A testbench is *unsynthesisable* – it doesn't map to physical gates; only for the simulator

Demonstration of SystemVerilog Syntax

Practical Work

(0.) Hello World

- Run the simulate script and see the output
- Add your own print message in a loop to understand testbench delays
- Remember – this is unsynthesisable!

1. Modelling Basic Gates

- Design the following boolean logic gates:
  - NOT
  - AND
  - OR
  - XOR

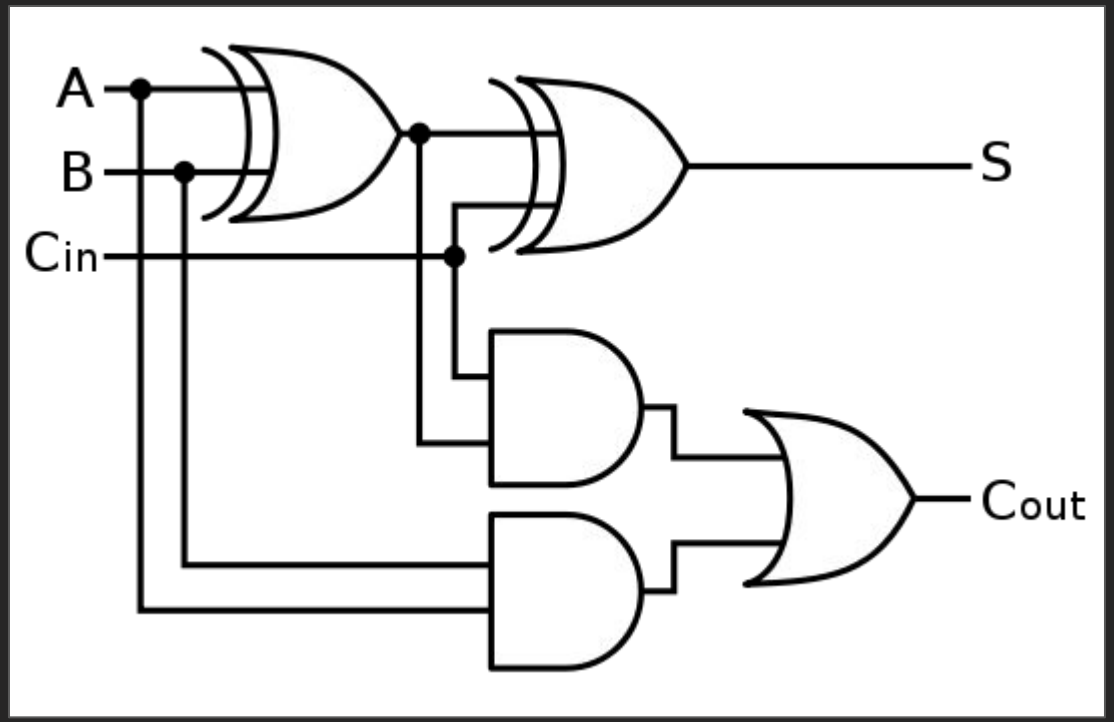
2. Full Adder

- Create a full adder module
- Three 1-bit inputs: `A`, `B`, `Cin`
- Two 1-bit outputs: `Sum`, `Cout`
- We've made some basic gates – should we use them?

Structural versus Behavioural Modelling

- A *structurally* modelled system describes the components and their connections
- A *behaviourally* modelled system describes the high-level behaviour
- You should very rarely write structural HDL – you're just creating a circuit diagram in code
    - High-level meaning is lost: hard to maintain and easy to create bugs
    - The synthesis tool will find a more optimal implementation than you

Structural Full Adder



2. Full Adder

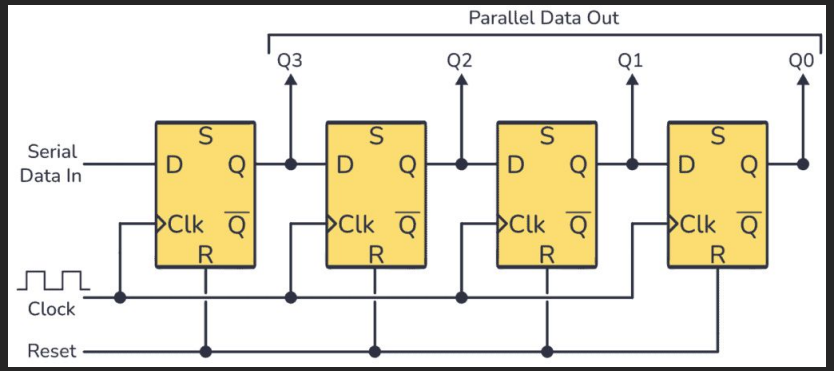
- Create a full adder module
- Three 1-bit inputs: `A`, `B`, `Cin`
- Two 1-bit outputs: `Sum`, `Cout`
- We've made some basic gates – should we use them?

3. Linear-feedback Shift Register

- A crude method for psuedo-random number generation
- First, let's create the shift register
- Then, we can add the linear feedback

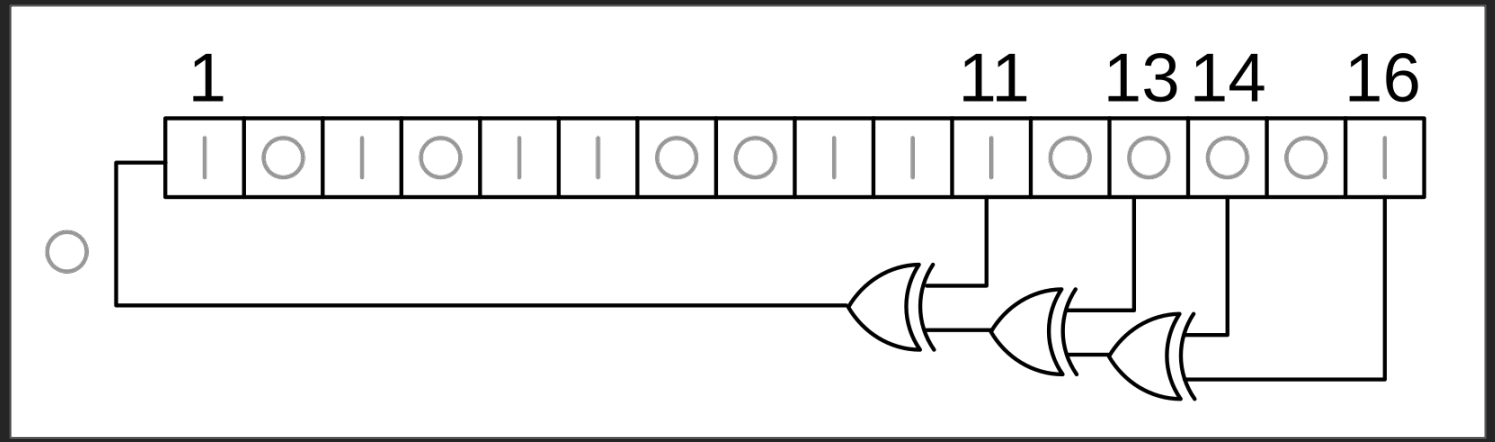
Shift Registers

- Upon a clock cycle, the bits in the register are ‘shifted’ by one place
- The top bit is removed from the register
- A new bit is stored as the bottom bit



Linear-feedback

- The new bottom bit is some function of the current state of the register
- XOR gates are usually used



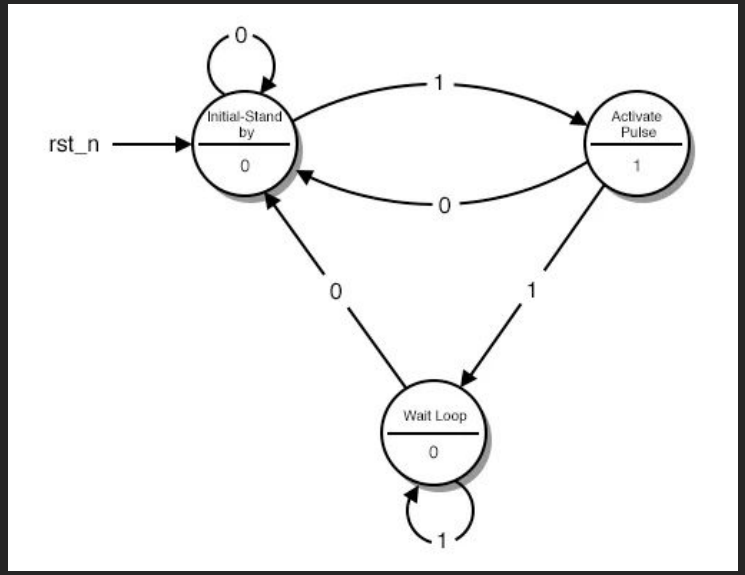
3. Linear-feedback Shift Register

- A crude method for psuedo-random number generation
- First, let’s create the shift register
- Then, we can add the linear feedback
- How “random” are the numbers?

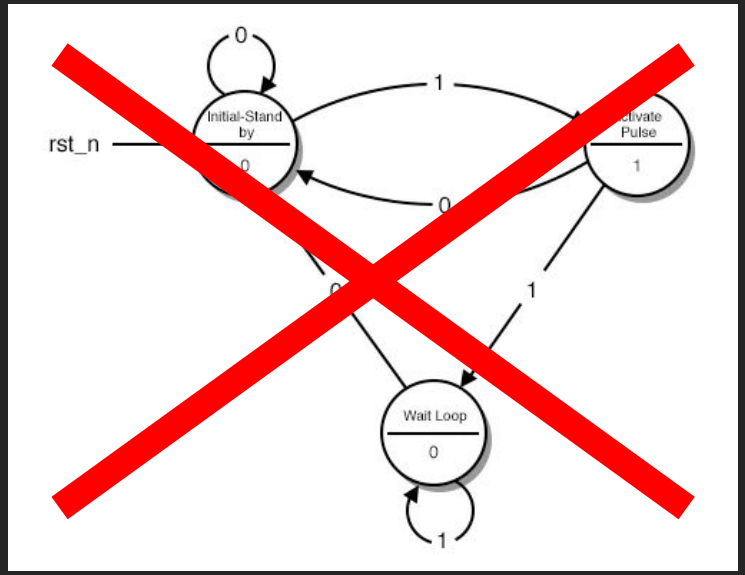
4. Algorithmic State Machine

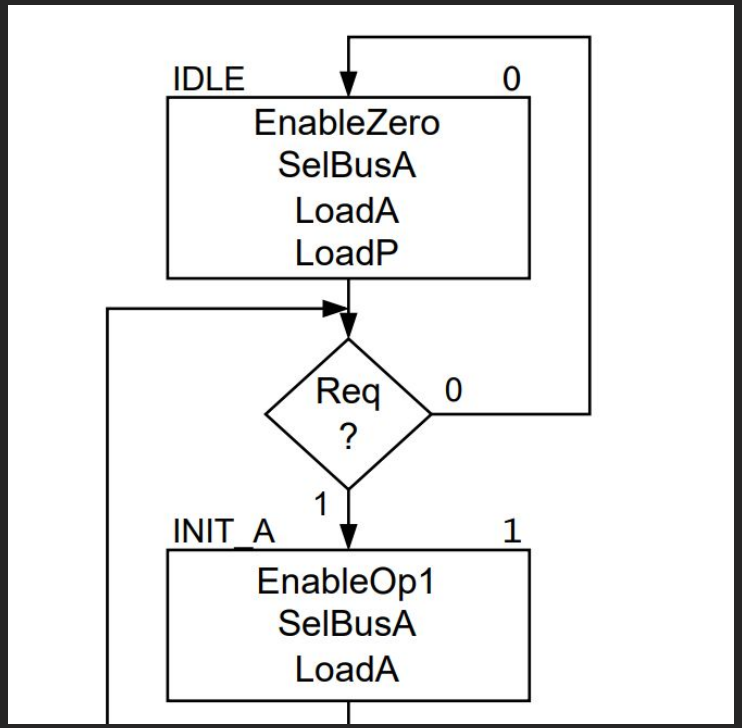
- Create a state machine representing a traffic light controller at a pedestrian crossing
- Then, create a testbench to make sure the design works properly

State Machines (1)

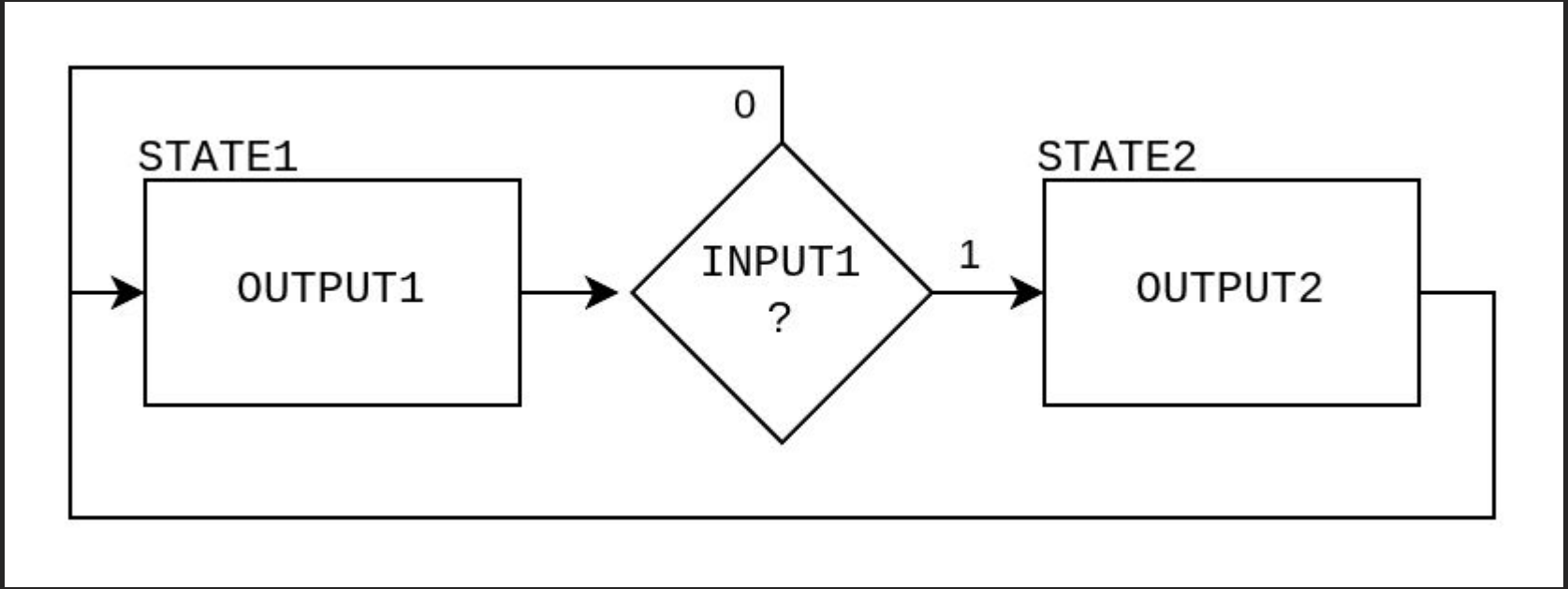


State Machines (2)

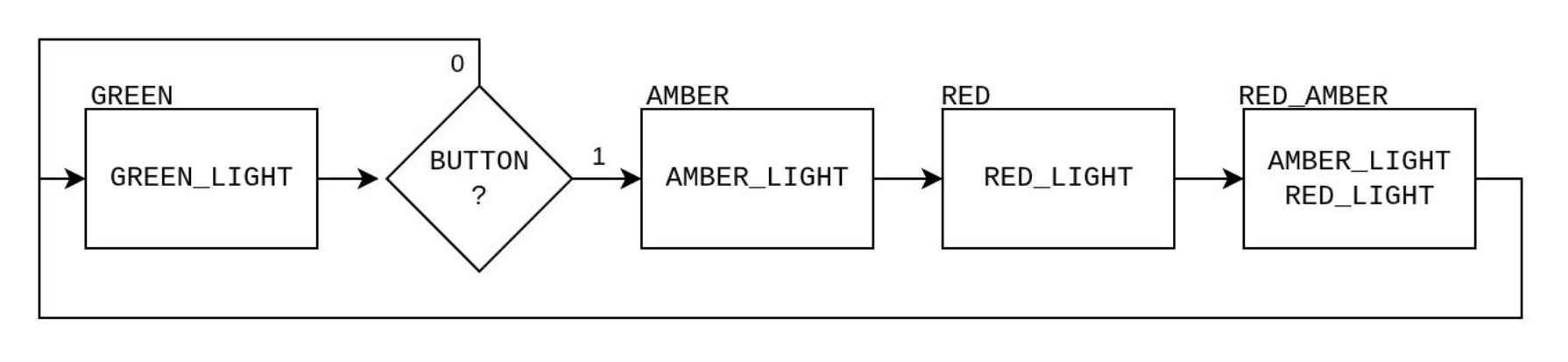




Example State Machine in SystemVerilog



Traffic Light Controller



4. Algorithmic State Machine

- Create a state machine representing a traffic light controller at a pedestrian crossing
- Then, create a testbench to make sure the design works properly

Next Week

- Learn about how CPUs are designed
- Implement some blocks in your own RISC-V CPU
- Run a handwritten machine code program

Some materials and inspiration in this presentation sourced from Iain McNally's notes (https://www.southampton.ac.uk/~bim/)