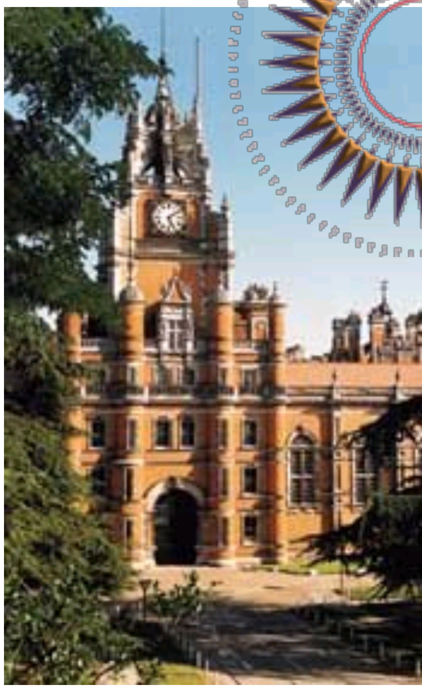


Lecture 2 – State Machines

Karl R. Wilcox
Karl@cs.rhul.ac.uk

Course Overview Reminder

- **Three main strands:**
  - **Digital electronics**
    - **Builds on Computer Engineering I**
  - **Processor design**
    - **Shows how we can build processors using digital logic**
  - **Assembly language compiling**
    - **How to get the best out of the processor**

Digital Logic from Comp. Eng. I

- Basic digital logic was covered in Computer Engineering I
  - Logic gates, truth tables and logic equations
  - Karnaugh maps and circuit minimisation
  - Multiplexors, encoders and decoders
  - Arithmetic elements
  - Flip flops
  - Simple counters

Digital Electronics in Comp. Eng. II

- **We will look first at state machines in general**
- **Look at implementing state machines**
- **Look at programmable logic devices**
- **A different technique for circuit minimisation**
- **Use that technique for state machine reduction**
- **Look at computer aided logic design**

Introduction to State Machines

- State machines appear in many parts of computer science
  - Parsers (compilers)
    - Use state machines to check syntax
  - Artificial Intelligence (and games)
    - Uses state machines to model behaviours
  - System modelling
    - E.g. UML State Diagrams
  - Electronic hardware
    - Washing machines, traffic lights etc.
  - Software development
    - State machines are a useful abstraction for program design

Types of State Machine

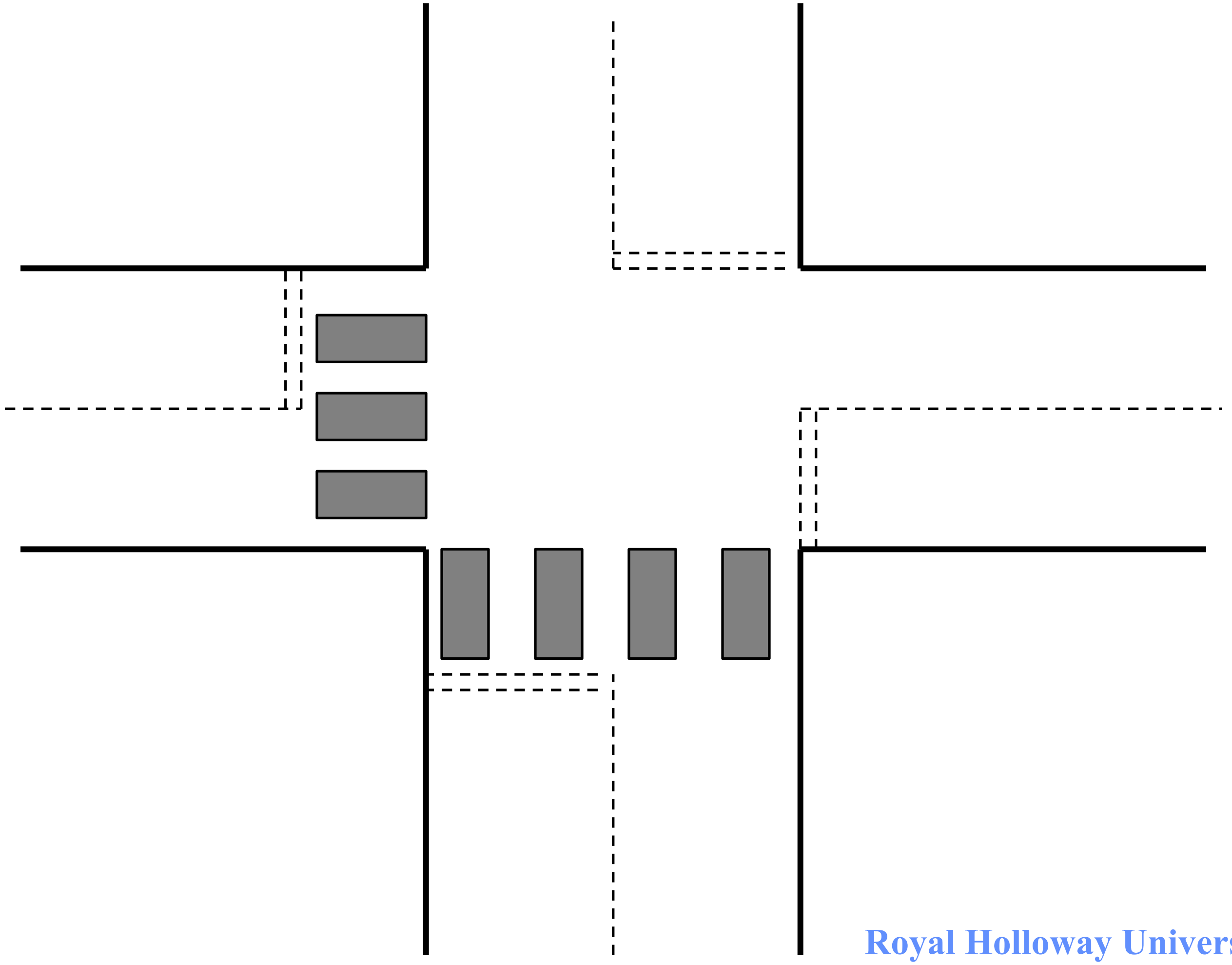
- **Also known as**
  - Finite State Automata
  - State Transition Networks
  - Recursive Transition Networks
  - Augmented Transition Networks

- **For our purposes we are most interested in Synchronous State Machines**
  - Synchronous because the machine is in only one of several, independent states
  - Transitions happen synchronised with clock “ticks”

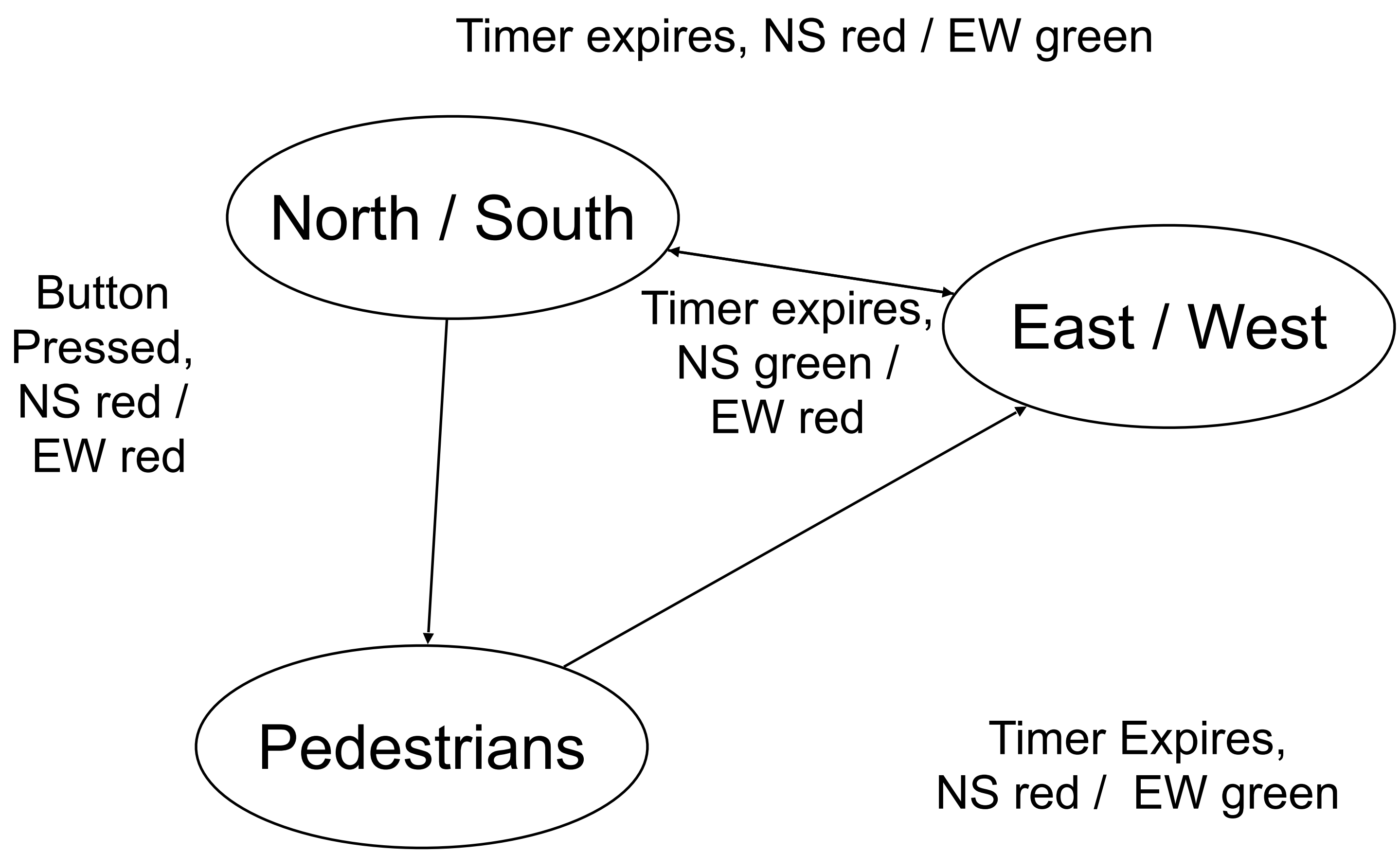
State Machine Diagrams

- States are typically shown as boxes or ovals
  - The name of the state is shown inside
- Possible paths between states are shown by arrows
  - Known as transitions
  - Labelled with a condition and an action
- Transitions do not have to have conditions
  - Unconditional transitions always happen
- Transitions do not have to have actions
  - All outputs retain their previous states

A Simple Traffic Light Junction



State Diagram For Traffic Lights



Possible Improvements

- Have different timers for each mode
- Include traffic sensors to “shortcut” timers
- Allow entry to pedestrian mode from both modes
- Delay between button press and changing to pedestrian mode
- Represent the “internal” traffic light sequence
  - Red -> red / amber -> green -> amber

- All of these can be represented as additional states, transitions or outputs

State Machines As Tables

State	Condition	Next State	Actions
North / South	Timer expires	East / West	NS Red EW Green
North / South	Button press	Pedestrians	NS Red EW Green
East / West	Timer expires	North / South	NS Green EW Red
Pedestrians	Timer expires	East / West	NS Red EW Green

Simulating State Machines

- State Machines are easy to simulate in software
- Can implement through a case / switch statement
  - Case state of….
  - Calls a procedure that sets new state & carries out actions
- Or use an array of structures
  - One entry per state (as pre table on previous page)
  - Include function pointer for actions
  - Current state is an index into the array

Limitations of State Machines

- **Simple state machines cannot model every situation**
- **Example, consider the following machine:**
  - **It has two buttons, A and B**
  - **and an LED output**
  - **The LED should light if the user presses button A any number of times, followed by pressing button B the same number of times**
- **This can be modelled for any *specific* number of button presses**
  - **But NOT for the general case of *n* button presses**

For Completeness RTNs…

- A Recursive Transition Network
  - Can have a state machine “inside” each state
  - E.g. An English sentence consists of a noun-phrase followed by a verb-phrase
    - The fox jumped
  - But noun phrases can include a verb phrase (with a relative pronoun)
    - The fox which was brown jumped
  - We can implement this as “nested” state machines
  - Beware! Recursion can be infinite!
    - He said that she said that he said that she said….

…And ATNs

- An Augmented Transition Network
  - Introduces “memory” into the network
  - Uses “registers” that store values
  - Uses more general tests than a simple “event” to trigger transitions
- Developing ATNs is much more like developing in a procedural language
- We will not cover ATNs further in this course!

Summary

- State Machines are generally useful tools in computer science
  - Both for understanding systems and implementing them
- Synchronous state machines have:
  - Two or more named states
  - Transitions between those states
  - On each clock tick the inputs are examined
  - Transitions happen in response to changed inputs (events)
  - Transitions may cause actions to occur

Next Week

- **We will look at how State Machines can be easily implemented in digital logic**

- **Next Lecture, Monday, C336**