Lecture 3 – State Machine Implementation Using ROMs

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

Administration

- Next lecture Monday

- Laboratory Sessions Venue
  - These are held in the Tolansky Laboratory

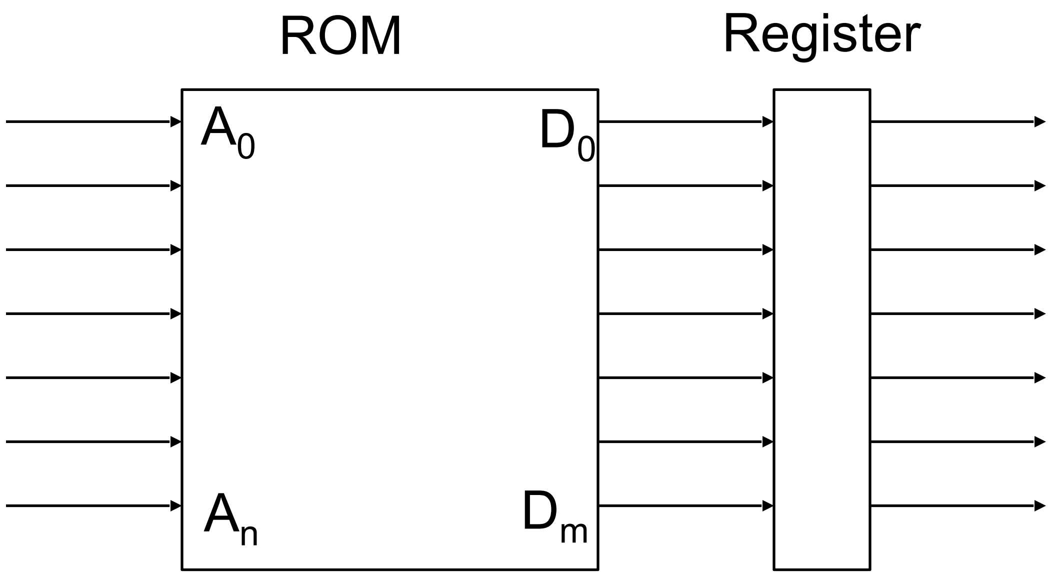
- Due to a software problem, there will be NO lab session this week
  - But there is preparation to be done!

Read Only Memories

- Typically thought of as storage for programs and data in embedded systems
  - "Firmware"
- Addresses are contiguous locations
- Data is bytes of program code or data
- But can also be regarded as general transforms of an input bit pattern (the address) to an output bit pattern (the data)

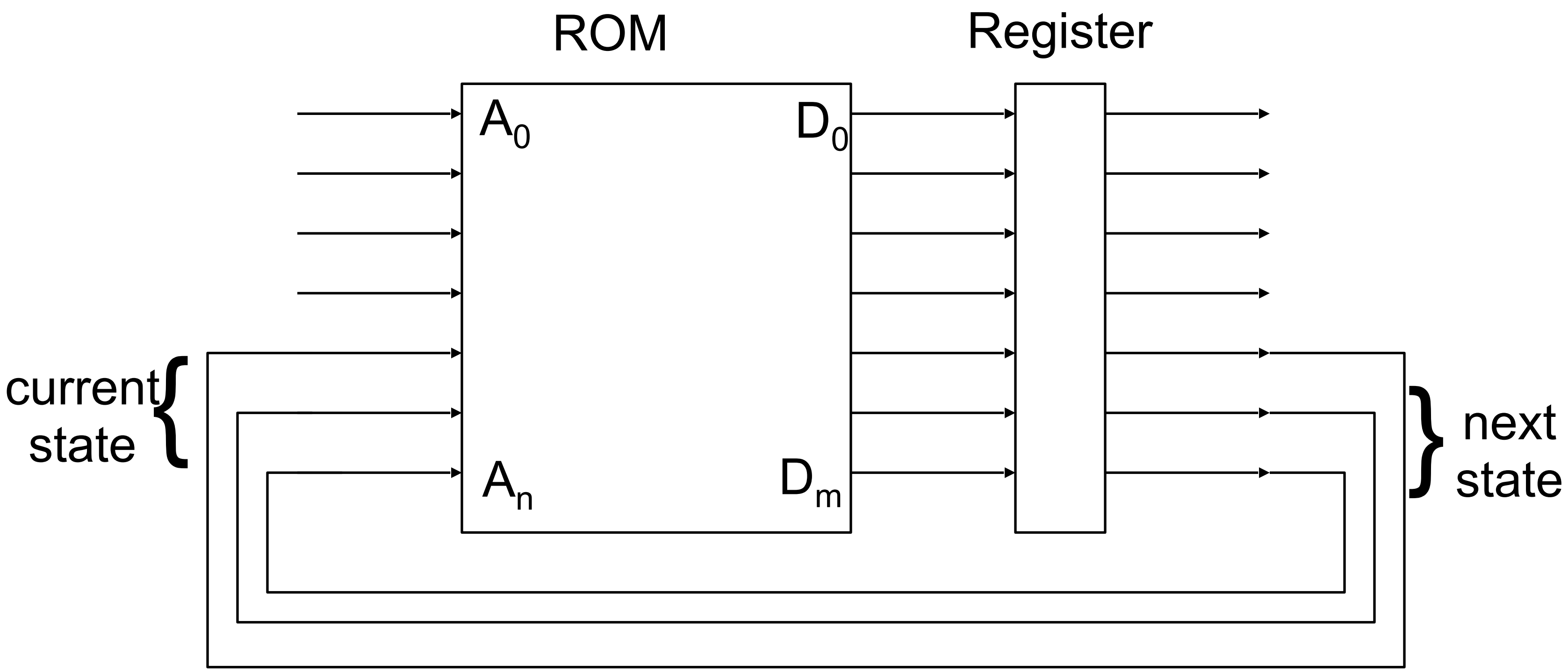
ROM for Generalised Transform

- **We need to add registers (flip flops) to maintain the output states**



Adding States

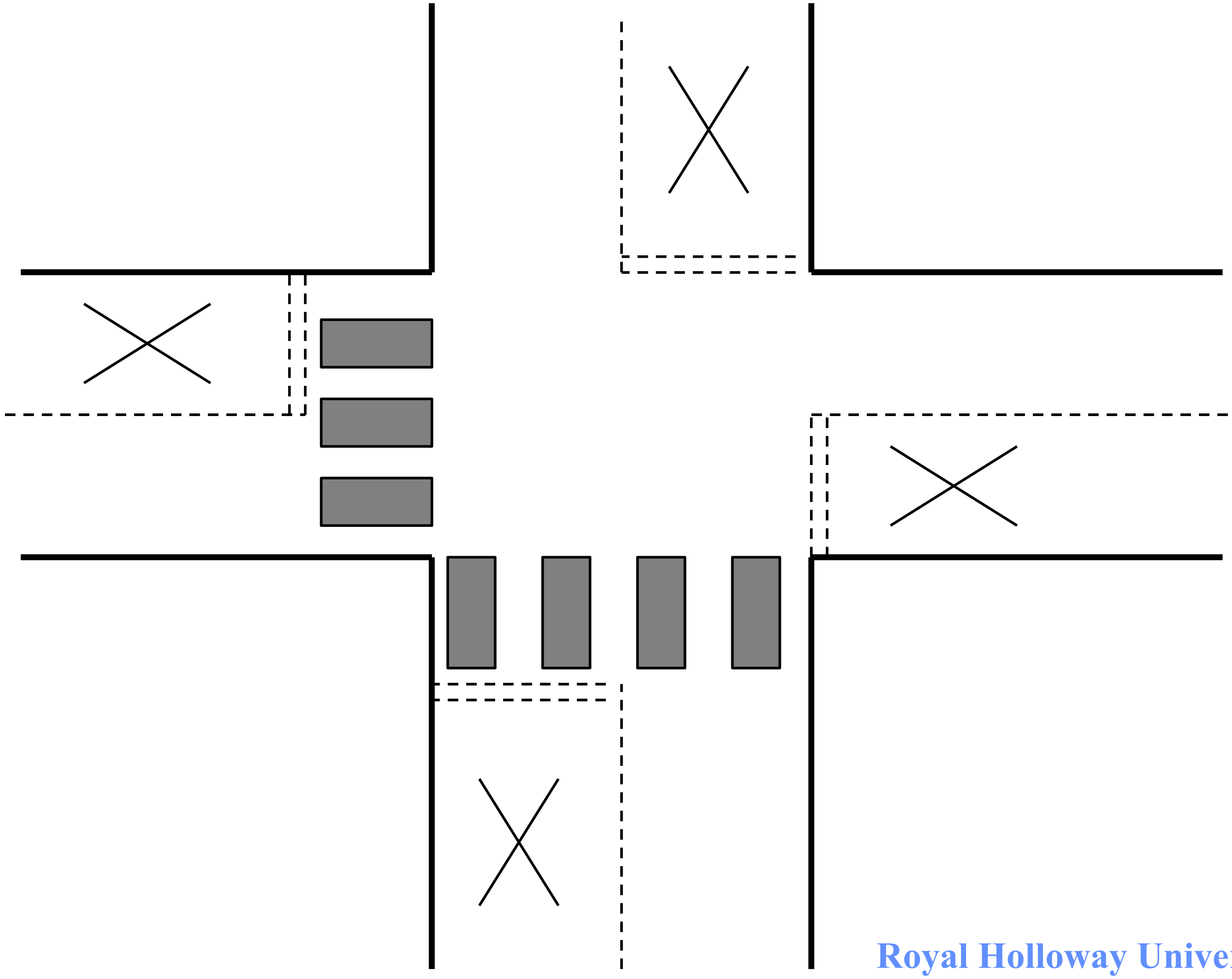
- **We use some of the address and data bits to represent the current and next states**



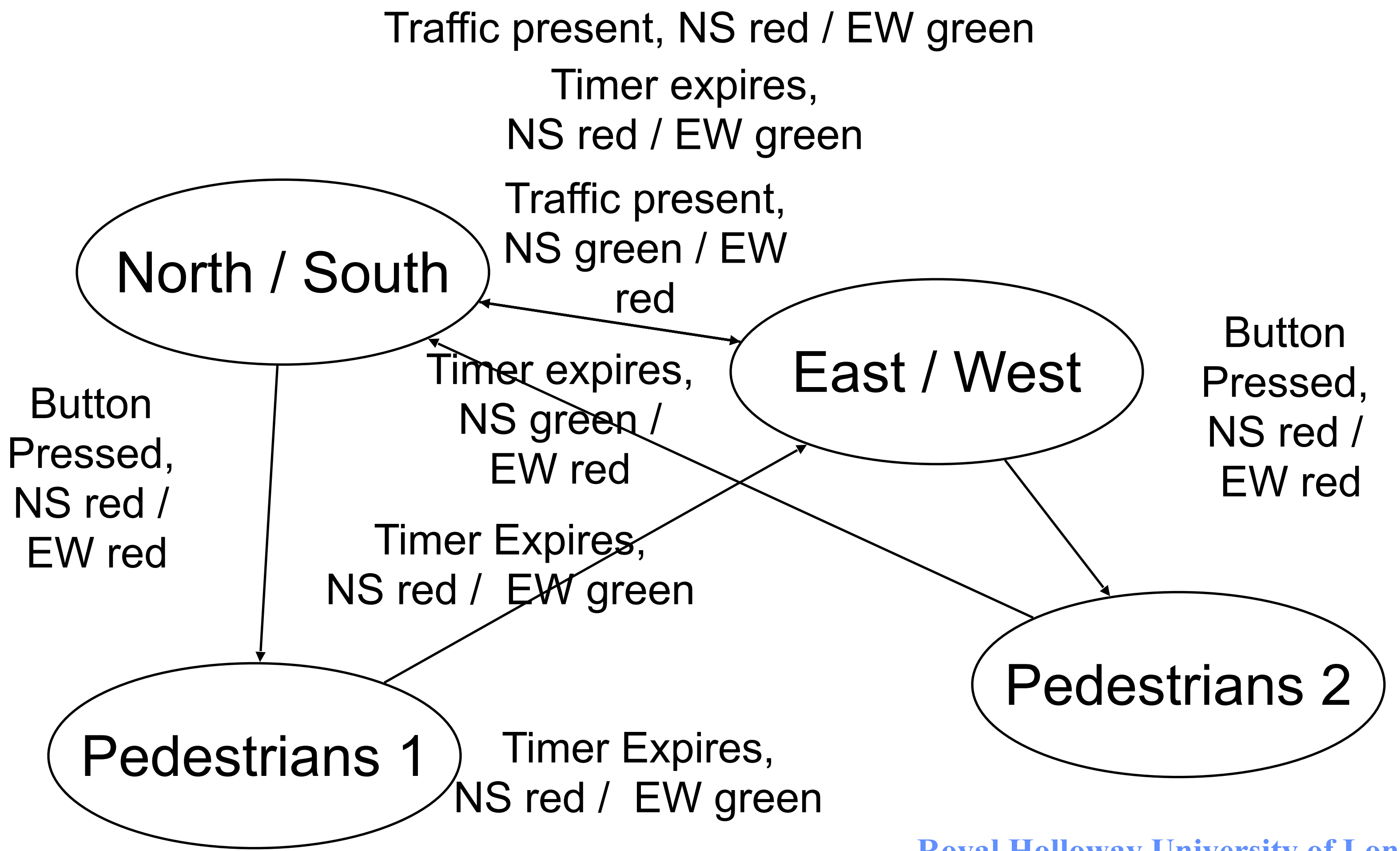
Traffic Light Example

- From previous lecture
- We will assume all timings are the same
  - We have a “timer reset” output
  - This generates a “timer expired” input after a set interval
- We will add traffic sensors
  - These generate a binary ‘1’ when vehicles are present
  - If there is traffic in the “red” direction, and no traffic in the “green” direction then
    - reset the timer
    - change the lights
- We also need a “pedestrian button” input
- And outputs for N/S and E/W lights

A Simple Traffic Light Junction



Traffic Light State Diagram



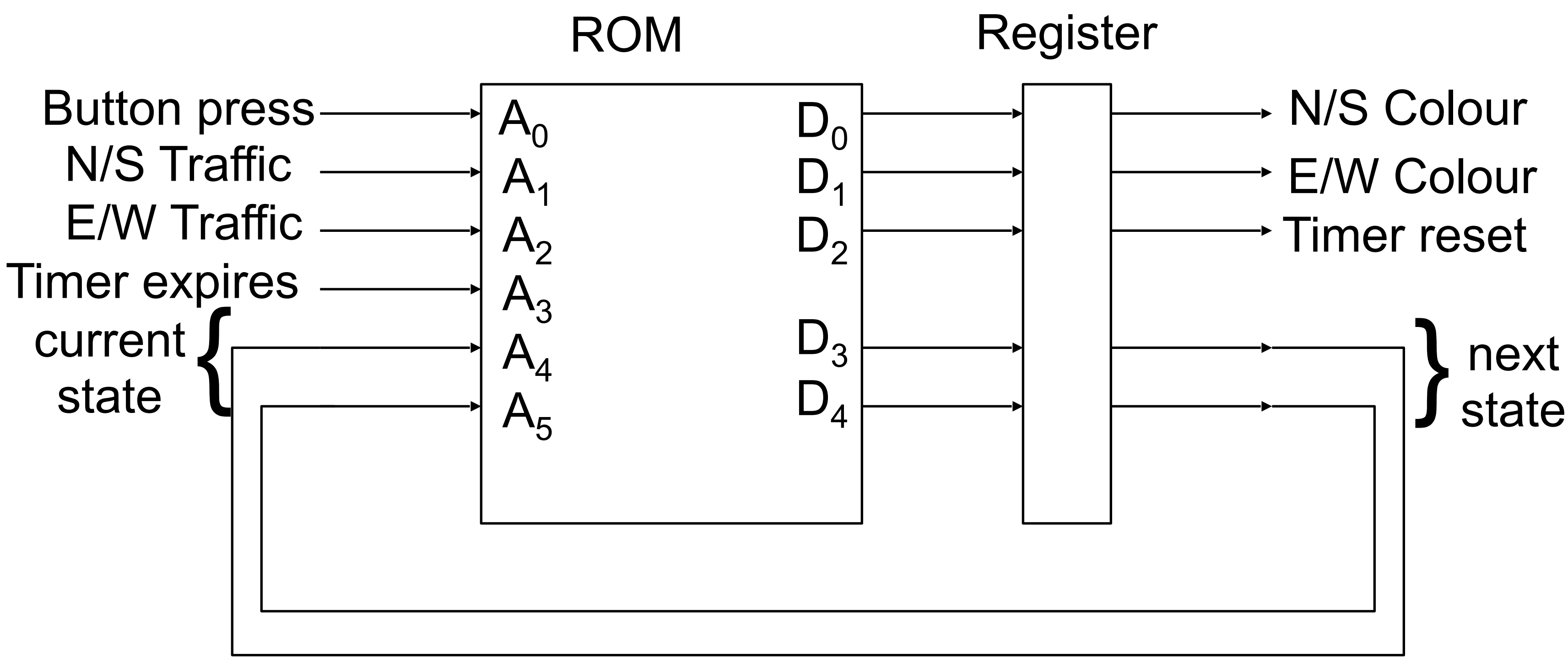
Address Assignments

- We need an input to represent a button press
  - Assign A_0 as “button press”
- We need inputs from each pair of traffic sensors
  - Assign A_1 as “NS traffic”
  - Assign A_2 as “EW traffic”
- We need an input for the timer expiry
  - Assign A_3 as “Timer expires”
- We need two bits to represent the state
  - (we have four states in total)
  - Assign A_4 and A_5 as the state bits as follows
    - 00 = North/South
    - 01 = East/West
    - 10 = Pedestrians 1
    - 11 = Pedestrians 2

Data Assignments

- We need an output to represent the state of the N/S lights
  - Assign D_0 as "n/s colour" (0 = red, 1 = green)
- Similarly for the E/W lights
  - Assign D_1 as "e/w colour" (0 = red, 1 = green)
- We will assume that if D0 = 0 and D1 = 0 then
  - Pedestrian lights are green
- We also need an output to reset the timer
  - Assign D_2 as "timer reset"
- And two output bits to represent the next state
  - Assign D_3 and D_4 as per previous slide

ROM Schematic



ROM Truth Table (extract)

Address Lines						Data Lines				
Btn prs	N/S traf	E/W traf	Tmr Exp	Crnt State		N/S	E/W	Tmr rst	Next State	
0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	1	1	0	1
1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	1	1	0	1
0	1	1	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	1	1	0
1	0	0	1	0	0	0	0	1	1	0
1	1	1	1	0	0	0	0	1	1	0

Etc...

ROM Implementation

- <u>**ALL**</u> **combinations of inputs need to be explicitly specified**
  - **However, this does make plain conflicting priorities**
    - **e.g. What happens when a button press coincides with the traffic sensor?**

- **ROM size must be a minimum of *n* words each of *m* bits long**
  - **This can get large quickly!**

Logic Implementation

- In most situations there are many “don’t care” combinations of inputs
  - In fact, the *majority* of inputs may be “don’t care”
- We can implement the truth table using combinatorial logic
  - But we need to minimise the hardware required
- Rather than discrete logic we can use programmable arrays of logic
  - More on these next week

Laboratory Tools

- We will be using the Tkgate simulation package
  - This runs under under X Windows on Linux / Unix and is available from http://www.cs.cmu.edu/~hansen/tkgate
  - It has also been installed on aspasia in C103
  - There is a tutorial available for those interested
- Tkgate is a graphical simulation package
  - "Draw" logic circuits using graphical components
  - Run the simulation, view outputs, timing diagrams etc.
- Much less dangerous than soldering!

Laboratory Preparation

- A circuit will be provided
  - As per previously shown ROM schematic
  - With the addition of clock signals etc.
  - Switches for the traffic sensors
  - LEDs for the traffic lights
- You will need to provide the ROM contents
  - 64 entries, as per previous table
  - Probably easier to provide this in address order

ROM Contents Format

- **You will need to provide a text file as input**
- **Must have extension ".mem"**
- **Blank lines and those beginning with # ignored**
- **Other lines must be composed of:**
  - **A hexadecimal address**
  - **A forward slash**
  - **one or more hexadecimal data bytes**
- **e.g.**

```
# Start loading at position 100
100/ e1 f0  0  0 e1 e0  0  0
108/ 81  0  0  0 12  1 bd  0
```

Possible Extensions

- **What is the effect on our ROM implementation of the following changes?**
  - Having different timer values for each direction?
    - Additional output required
  - Adding "filter lanes" for particular directions
    - Additional states (therefore more bits to represent them)
    - Additional outputs
  - Correctly modelling the traffic light sequence (red -> red / amber -> green -> amber)
    - Lots more states and outputs

Summary

- **State Machines can easily be implemented using ROMs**
  - **But need to fully specify ROM contents**
  - **ROM size can become large quickly**
- **Using programmable arrays of logic allows us to take advantage of “don’t care” states**
  - **But we need a mechanism to do this**

Next Week

- **We will look at how State Machines can be implemented in programmable logic**
- **A more general minimisation method than Karnaugh maps**