Part IA Computer Science Tripos

Hardware Practical Classes

Year: 2024 – 2025

Dr. I. J. Wassell,

Mr. N. Batterham.

Introduction

The Hardware Practical Classes accompany the Digital Electronics series of lectures. The aim of the Practical Classes is to enable students to get hands-on experience of designing, building, and testing and debugging of digital electronic circuits. The labs will be held in the Intel Lab. (SW11) located in the William Gates Building (WGB).

The Digital Electronics lecture series occupies 12 lectures in the first 4 weeks of the Michaelmas Term, while the Practical Classes occupy the latter 6 weeks of the Michealmas Term and all 8 weeks of the Lent Term. The Practical Classes take the form of 4 workshops. Each Workshop has an accompanying worksheet that is included in this document, specifically:

- Workshop 1 – Electronic Die;
- Workshop 2 – Shaft Position Encoder;
- Workshop 3 – Debouncing a Switch;
- Workshop 4 – Framestore for an LED Array.

In general, the workshops require some preparatory work to be done prior to the practical session. These tasks are highlighted at the beginning of each Worksheet. Typically this involves preparing a design that you will then build, test and modify during the practical class. Note that insufficient preparation prior to the practical classes may compromise effective use of your time in the laboratory.

In the Practical Classes you will usually work on your own, and you are expected to complete one Workshop in each of your scheduled sessions. The days of the week and the times that the Practical Classes run depends upon the term. Demonstrators are available during the sessions to assist you with any queries or problems you may have. Here are some additional points to bear in mind regarding the Practical Classes:

- Please make sure you stick to your timetabled session, otherwise the numbers per session become unbalanced giving rise to problems in terms of equipment availability and demonstrator support;
- Arrive promptly and begin your work in a timely manner. This will maximise the probability of completing a workshop before the end of the laboratory session;
- Make sure you keep up with the workshop schedule and get each workshop 'ticked';
- During week 2 (Lecture 4) of the accompanying Digital Electronics lecture series, a lecture to introduce the Hardware Practical Classes will be given**. It is important that you attend this lecture, particularly if you have no prior experience of electronics.**

Circuit Building Guidance

Many materials are available on which to build prototype circuits. The material chosen will depend on the required life and use of the circuit. For laboratory use or other short-term applications where a permanent circuit is not needed and the operating frequency is below 10 MHz, solderless breadboards as used in the prototyping box shown below are suitable.

The prototyping box comprises:

- 3 breadboards on which your circuits will be constructed
- A 5 volt power supply required for the logic integrated circuits (chips)
- A variable frequency clock which operates in either manual or free running mode
- 8 switches, which can provide inputs to your circuit
- 2 push button switches (not debounced)
- A debounced push button switch
- Two 7-segment light emitting diode (LED) display
- A 10x7 LED array
- 8 user input LEDs
- A potentiometer
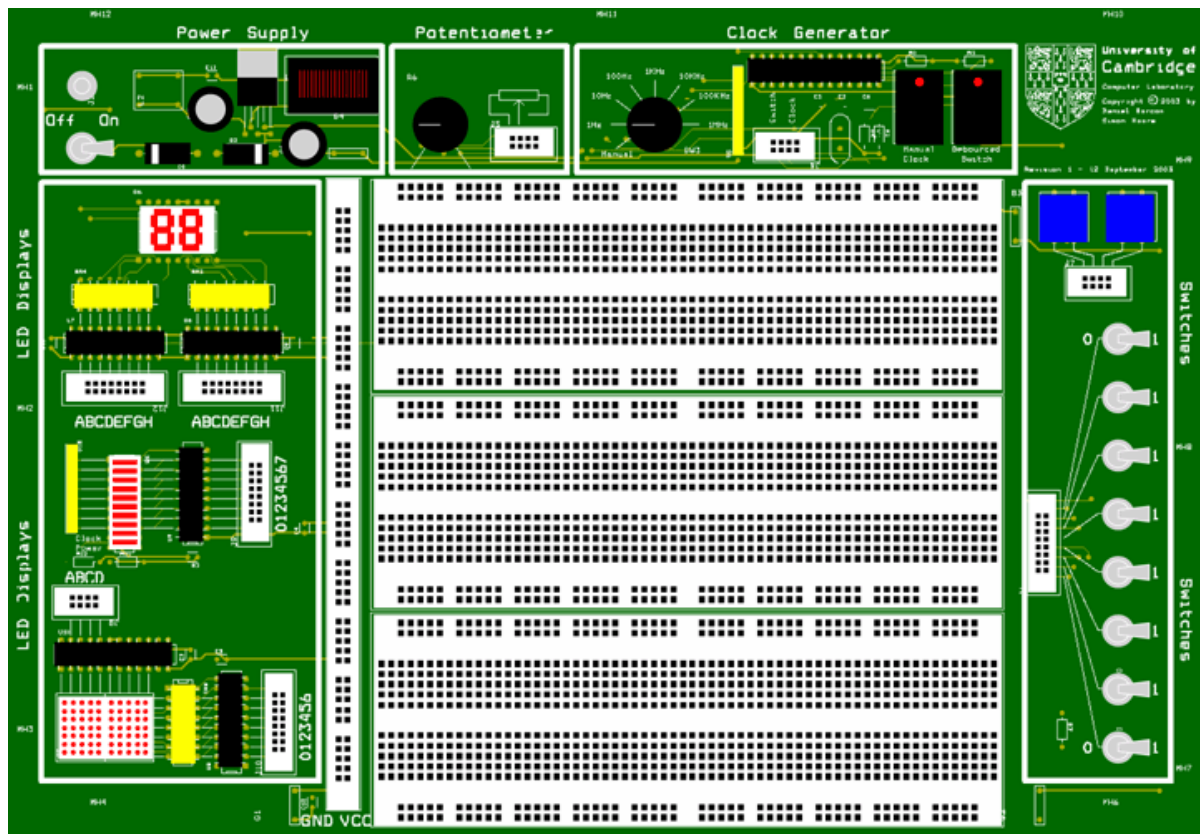
There are three banks of LEDs on the left hand side of the board for signal output:

- At the top there is a pair of 7-segment LED displays which are connected via PALs. The PALs have been configured to do binary to 7-segment hexadecimal character conversion. Inputs A to D provide the binary inputs.
- In the middle there is an LED bar graph. 8 of the LEDs are connected via a buffer. The other two indicate when the power is on and the state of the clock.
- At the bottom you will find an LED matrix. The column decoding is provided by a 4-

to-16 line decoder (inputs A to D) and the row value is supplied via a buffer (inputs 0 to 6).

Using the Breadboard

A breadboard has many strips of metal which run underneath the board. The metal strips are laid out as shown in red below, horizontally along the two lines at the top and bottom, and vertically to join six holes together in the central section.

These strips connect the holes on the top of the board. This makes it easy to connect components together to build circuits. To use the breadboard, the legs of components are placed in the holes (the sockets). The holes are made so that they will hold the component in place. Each hole is connected to one of the metal strips running underneath the board.

The long top and bottom row of holes are usually used for power supply connections. The rest of the circuit is built by placing components and connecting them together with jumper wires. Chips can be placed in the middle of the board so that half the legs are on one side of the middle line and half are on the other side. A completed circuit might look like the following.
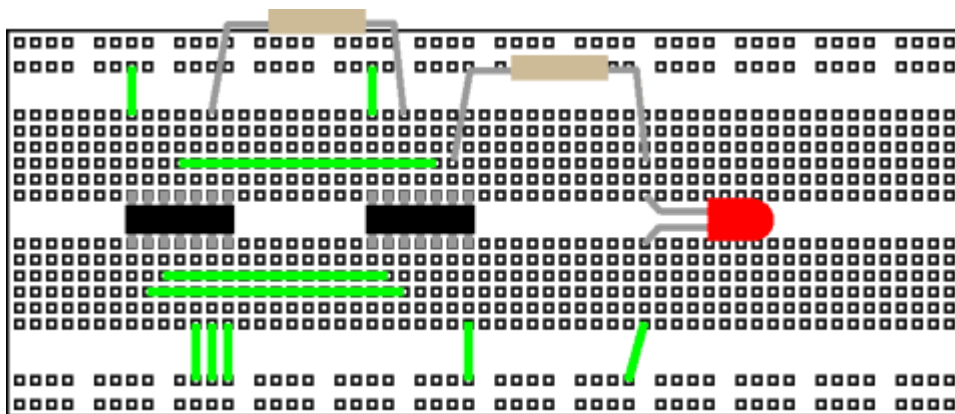
Careful planning of the circuit layout simplifies wiring, minimizes errors and makes debugging easier. Try to arrange the circuit for a logical signal flow, usually inputs on the left and outputs on the right. This helps anyone looking at the board to find easily sections of the circuit and trace signals through it. Wherever possible, all ICs should be pointed in the same direction to reduce the chance of one being put in backwards. This also makes it easier to keep track of pin numbers when wiring and debugging. Colour coded wiring is an easy way to reduce wiring errors and aid the process. If possible, build and test one section of the circuit at a time to simplify debugging before connecting the sections

together. Keep connecting wires as short as possible and route them around ICs so that a defective IC can be replaced without removing and often incorrectly replacing the wires. While all this may seem laborious, the time spent in careful construction and checking is well repaid by having more circuits work first time.

The pin outs of all the ICs used in the hardware labs are available as a booklet in the lab. Never take the pin out of an IC for granted! Below are two similar chips: the 7400 and 7402, are quad NAND and quad NOR gates respectively. Note the different pin out.

7400   7402
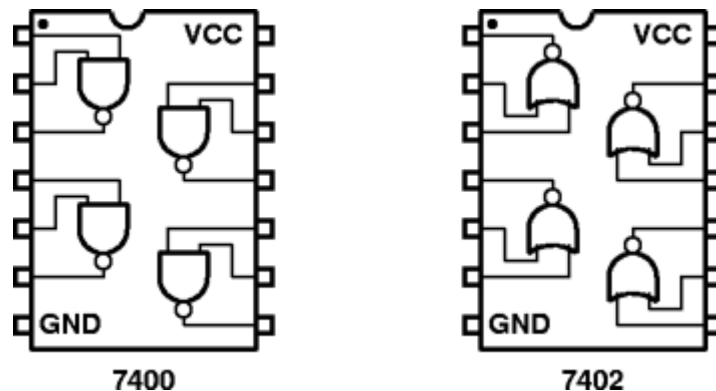
More details concerning the chips you will be using are available in the lab booklet and also on the course web page.

Common Ticking Criteria

All of the workshops should be written up in full in the style of a laboratory log book. You may use an actual log book or else loose sheets with page numbers. From your notes it should be possible for somebody else to be able to reproduce your work. This is good scientific practise. When an exercise is complete and written up, including answers to the questions, the final page will be signed by a demonstrator and a tick entered in the tick sheet.

We recommend that you draw out circuits in your log book before wiring them up. Put pin numbers for the connections to each chip. Cross out connections on the circuit diagram as you add them to the real circuit – this will help prevent wiring omissions/errors.

Appendix 1 – Resistor Colour Code

Resistors are normally coded with coloured bands to enable quick identification of their value. The colour code specifies the value of the resistor, in ohms, and the maximum deviation from the stated value (the tolerance). Most resistors have a manufacturer's tolerance with is adequate for most electronic applications.

Interpreting the Colour Code

The four colour code bands are at one end of the component. Counting from the end, the first three (or sometimes four) bands give the resistance value and the last the tolerance. The significance of the colours is shown in the table below:

First 3 (or 4) bands		Tolerance band	
Black	0	Brown	1%
Brown	1	Red	2%
Red	2	Gold	5%
Orange	3	Silver	10%
Yellow	4	No band	20%
Green	5		
Blue	6		
Violet	7		
Grey	8		
White	9		

Reading the value (three band)

The first two bands are used to specify the first two digits of the resistor's value. The third gives the number of noughts to be added. Thus in this example, yellow and violet give the first two digits as 4 and 7 and the number of zeros is 3 (orange) giving a value of 47000 ohms.
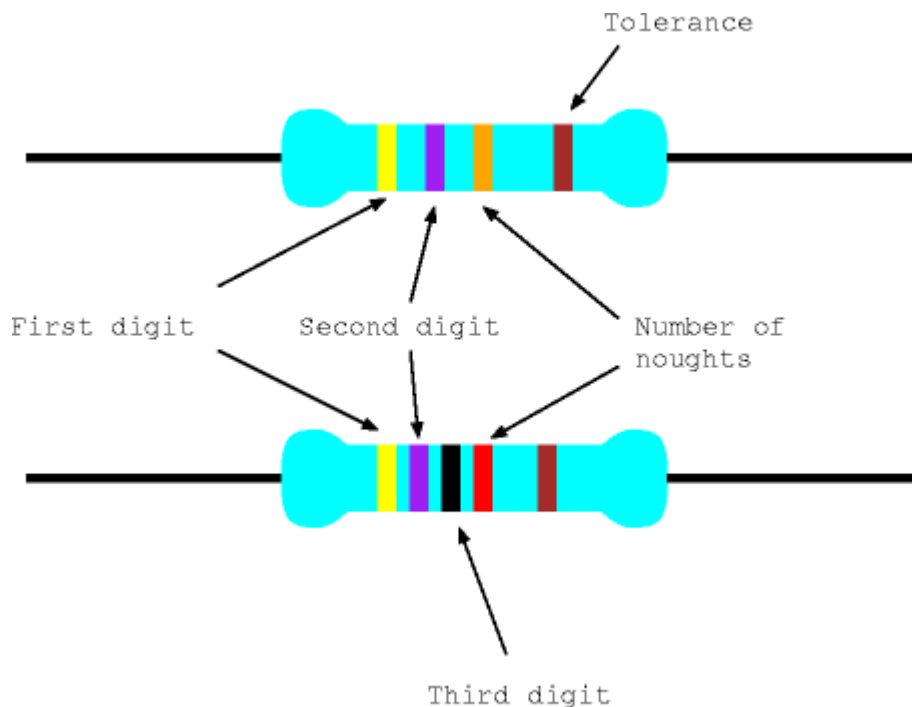
Four Band Resistors

More modern resistors use four bands to specify the resistance, the first three giving the three most significant digits and the fourth the number of zeros. Yellow, violet, black, red decodes as 4, 7, 0, 00 i.e., 47000 ohms.

Workshop One – Electronic Die

Introduction

The purpose of this lab is to build a simple electronic die using light emitting diodes (LEDs) as the display device. Digital logic chips will be used to implement the required functionality. You will construct the die on a prototyping breadboard that is housed in the 1A workshop prototyping box. Before you start wiring up the circuits, make sure you understand how the holes in the breadboard are connected and also how the pins are numbered on the logic chips – see the description in the provided Circuit Building Guidance.

Equipment and Components

1A Workshop Prototyping Box
Connecting wire
Red LEDs (7)
220Ω resistors (7)
10kΩ resistor
74HC00 – Quad 2 input NAND gates (2)
74HC193 – Synchronous 4 bit up/down counter (1)

Step 1 – Wiring up one LED

First we need to see how to turn on one LED; note there will be seven in the completed die. If a LED is connected directly across the 5V power supply used to power the digital logic chips, it may well draw enough current to destroy itself. To prevent this happening we need to connect a current limiting resistor in series with the LED. The required circuit is below, followed by the calculation to select an appropriate value for the current limiting resistor, R.

Please make sure you use the correct value resistor, otherwise you may destroy the LED.
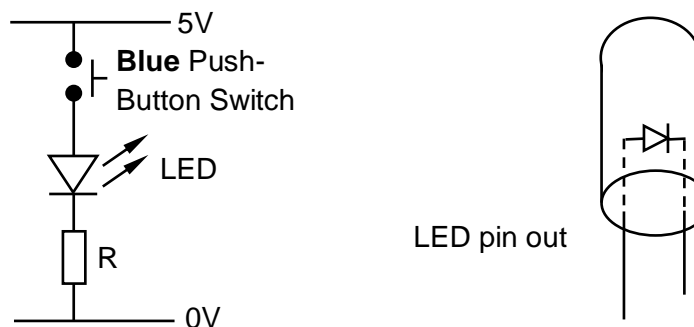
When the LED is lit correctly, the voltage across it will be about 1.6V and the current flowing through it should be about 15mA. To work out the appropriate value for R we need to use Ohm's law, i.e., V = IR. In this case the voltage across R is (5 - 1.6)V, so,

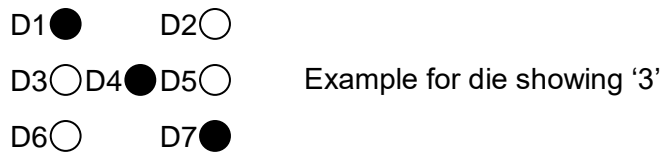$$R = V/I = (5 - 1.6)/0.015 = 226\Omega.$$

Note that the nearest available value (known as the preferred value) is 220Ω.

Task: Construct the above circuit and check that the LED lights when the button is pressed.

Step 2 – Binary to Die Decoder

To construct the die you will need to arrange the 7 LEDs in the following pattern to emulate the spots on a die.

D1●   D2○

D3○D4●D5○   Example for die showing '3'

D6○   D7●

Task: Arrange the 7 LEDs on your breadboard to make the above pattern; making sure that each LED has a 220Ω resistor in series with it. Then construct the following circuit using a 74HC00 quad 2-input NAND chip, i.e., each chip contains 4 NAND gates. Test the circuit by connecting the 3 inputs to the toggle switches in the prototyping box. Remember to connect V_{cc} (pin 14) to 5V and GND (pin 7) to 0V. Also unused gate **inputs** must be connected to 0V to prevent unstable operation of the chips.
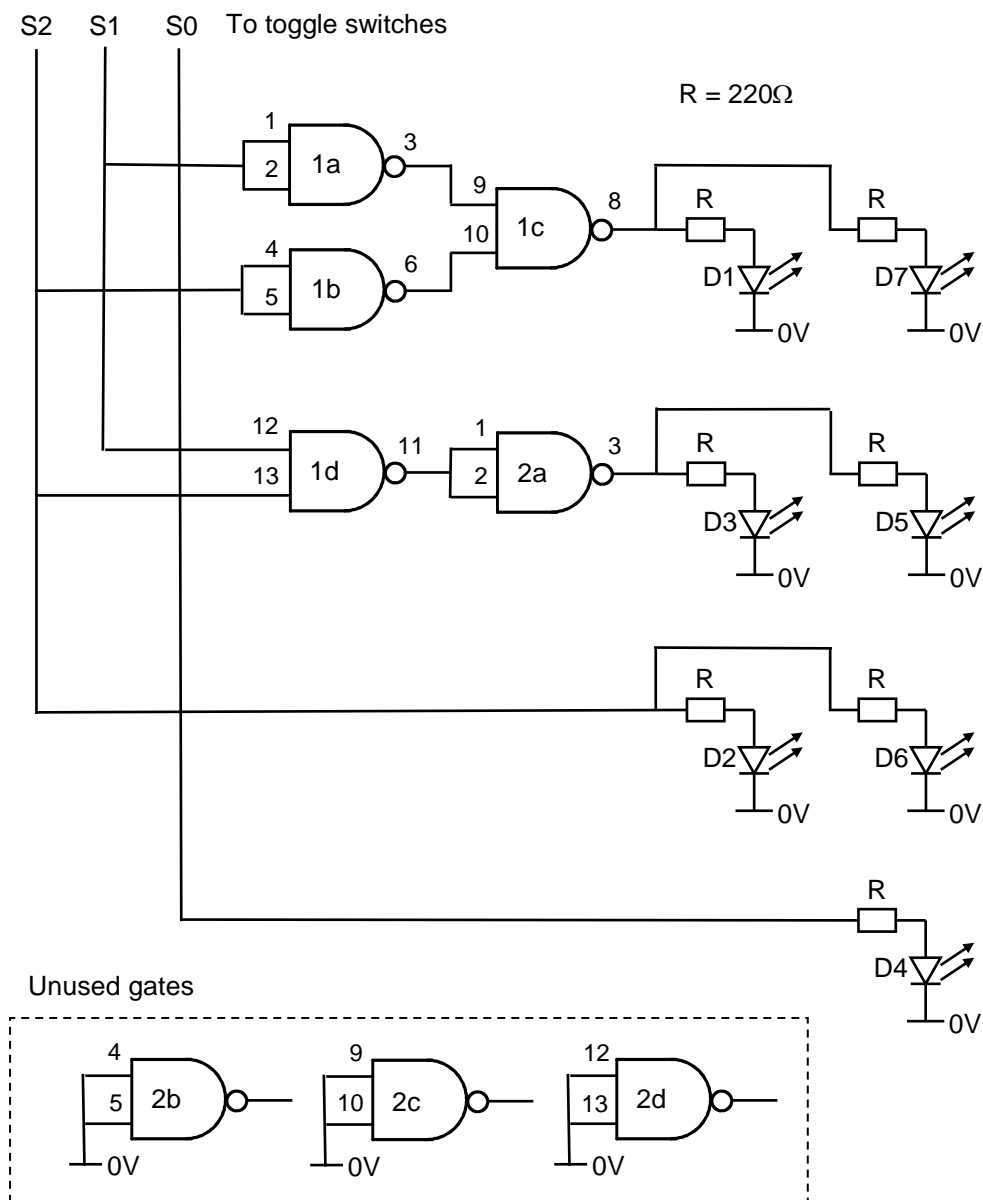
Step 3 – Make the die spin

Task: Use a 74HC193 counter chip as a replacement for the toggle switches. Wire up the circuit as shown below. Set the clock generator in the prototyping box to 100kHz. Note that to make the counter pausable, the 100 kHz clock is connected through one of the blue push button switches. Remember to disconnect S0, S1, S2 from the toggle switches and to connect them to the appropriate outputs from the counter, namely Q0, Q1, Q2. Also remember to connect V_{cc} (pin 16) to 5V and GND (pin 8) to 0V. You should now have a usable 8-sided die!
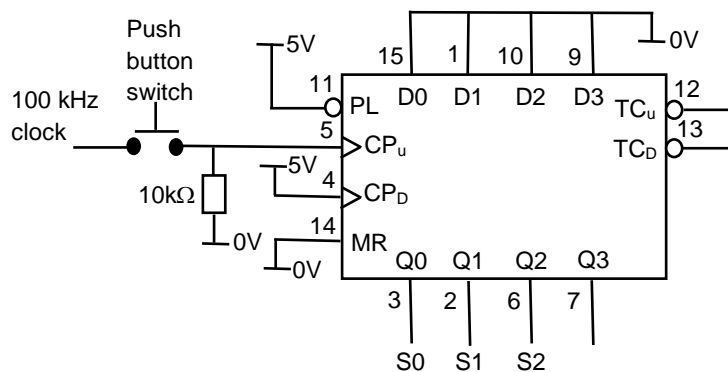
## Step 4 – Count from 1 to 6

Task: Modify the previous circuit as shown below so that the die counts from 1 through 6. The decode logic detects the count of '7' and forces the counter to parallel load with the count of '1'.
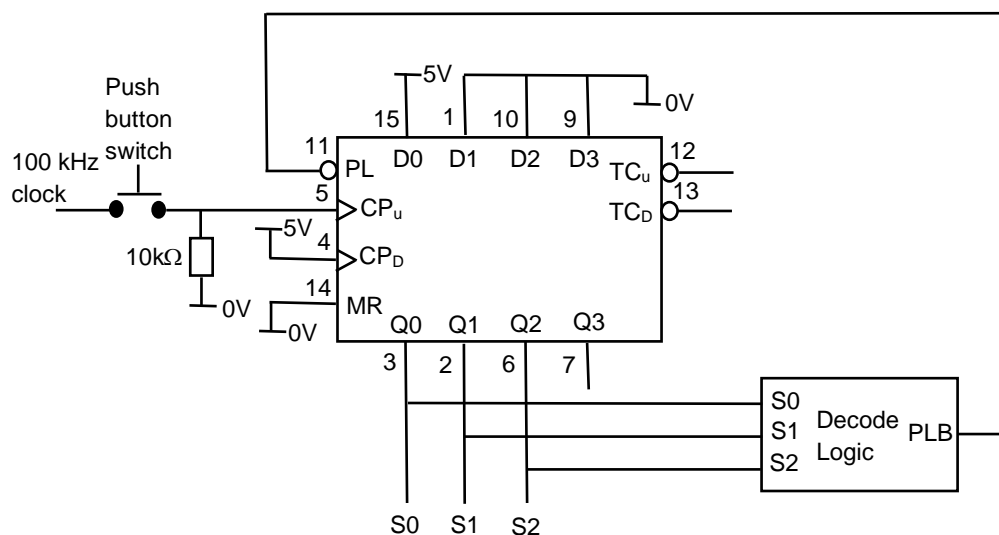
The decode logic has the following circuit diagram. Note that it uses the previously unused NAND gates from the original die design – remember to disconnect their inputs from 0V!
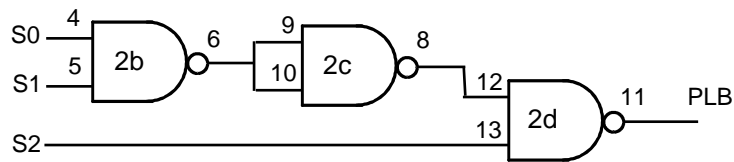
Assessment

Ticking criteria: Demonstrate that your 8-sided die works and answer the following questions.

Once your work has met the Common Ticking Criteria (see Introduction), get your work ticked by an assessor. Remember that you need to hand-in this assessment exercise as part of your portfolio of work (see Head of Department's notice).

In future workshops, it will be often be convenient to monitor the logical value of a signal using LEDs. Rather than wiring up discrete LEDs with serial resistors as we have done in this workshop, it will be more convenient to use the bank of 8 pre-wired LEDs which is available on the left hand side of the prototyping box.

Questions

1. What die patterns are displayed for binary inputs 000 and 111?
2. How much current can each NAND gate on the 74HC00 drive?
3. How would you arrange an LED (and resistor) so that it turns on when the output of a NAND gate is low?
4. How could this reduce the number of gates you have used?

I. J. Wassell   6-9-22

Workshop Two – Shaft Position Encoder

Introduction

Do Step 1 before the practical session.

Some industrial automation applications require control systems which know the rotational position of a shaft. Similar devices are also used for digital volume controls, etc., on domestic appliances. A shaft encoder is used to record the rotation typically in the form of a digital Gray code. In this workshop you need to build a decoder for such an encoder.

The shaft encoder you will be using produces a 2-bit Gray code. Such encoders are usually optical (e.g., those commonly used in a mouse), but in this case, we will be using a mechanical encoder. A disk is connected to the shaft which rotates with it. Electrical contacts are made with this disk to produce one of two Gray code sequences shown below depending upon the direction of rotation.
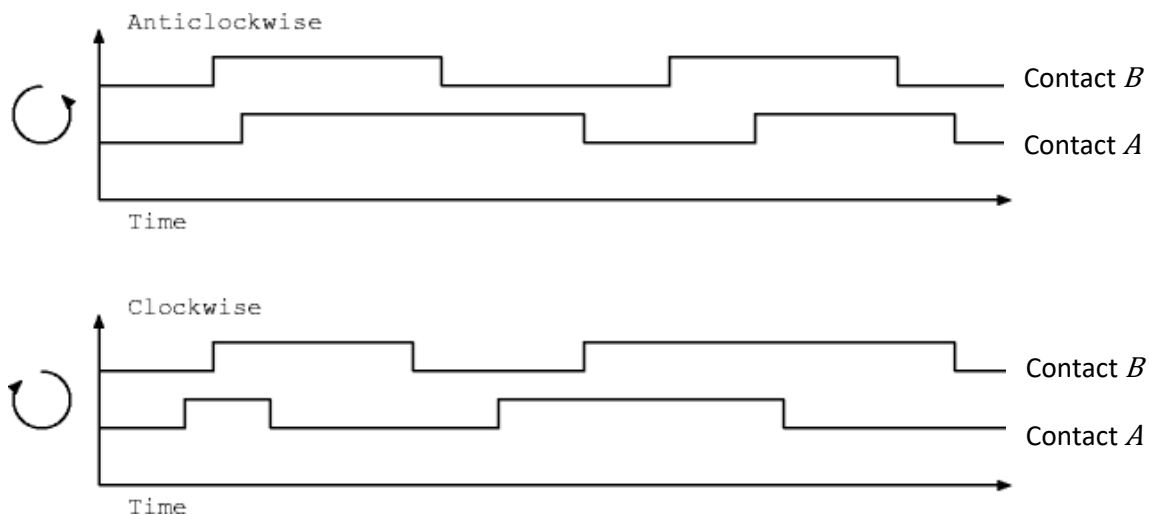
A shaft decoder module is required to convert this 2-bit Gray code (B, A) into a 4-bit (i.e., 0 to 15) position count. The 4-bit position count should be *incremented* every time the input from the encoder changes owing to a *clockwise* rotation (e.g., from 00 to 01, or from 10 to 00). Similarly, the 4-bit position count should be *decremented* every time the input from the encoder changes owing to an *anticlockwise* rotation (e.g., from 00 to 10, or from 01 to 00). Firstly, a Finite State Machine (FSM) must be designed to generate the clockwise and anticlockwise rotation detection signals. These signals are then used to control a separate 4-bit binary up/down counter that generates the current position (i.e., the counter output value) that is then displayed on one of the 7-segment LED displays (with built-in hexadecimal to 7-segment display decoder) housed in the prototype box.

Components

- 1A prototyping box
- rotary shaft encoder and two 4.7kΩ pull-down resistors
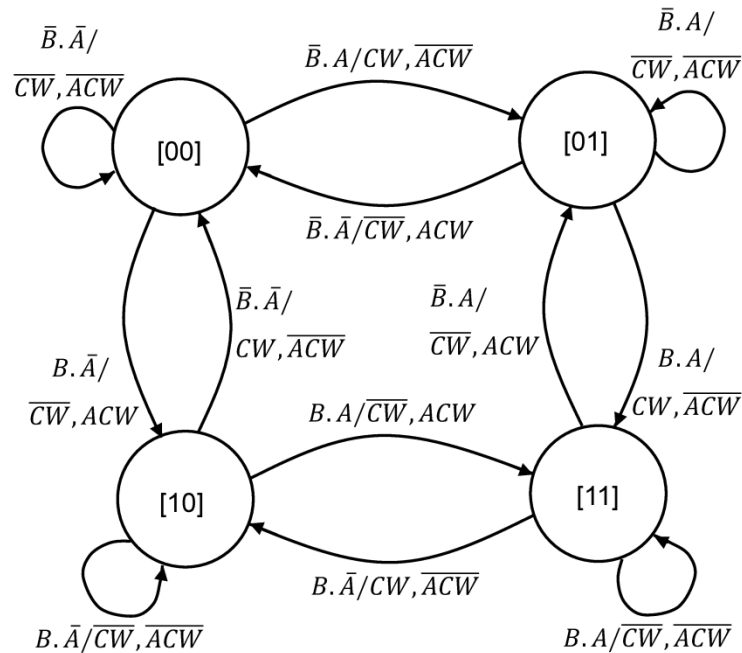- a PAL - GAL16V8
- 74HC193 - 4-bit up/down counter

Note: The recommended circuit for the shaft encoder is given in the lab booklet - remember to use the 4.7kΩ pull-down resistors to turn the switch open state into a firm logic value. Also

note that the encoder provided steps through **all four** of the output combinations for each 'click' position, consequently you need to turn it **very slowly** between click positions.

Step 1 – Do before the practical session

Design Shaft decoder to generate clockwise/anticlockwise signals

Here is the state diagram to decode this movement.

$\bar{B}.\bar{A}/$
$\overline{CW},\overline{ACW}$

$\bar{B}.A/CW,\overline{ACW}$

$\bar{B}.A/$
$\overline{CW},\overline{ACW}$

[00]

[01]

$\bar{B}.\bar{A}/\overline{CW},ACW$

$\bar{B}.\bar{A}/$
$CW,\overline{ACW}$

$\bar{B}.A/$
$\overline{CW},ACW$

$B.\bar{A}/$
$\overline{CW},ACW$

$B.A/$
$CW,\overline{ACW}$

$B.A/\overline{CW},ACW$

[10]

[11]

$B.\bar{A}/CW,\overline{ACW}$

$B.\bar{A}/\overline{CW},\overline{ACW}$

$B.A/\overline{CW},\overline{ACW}$

FF labels: $[Q_B \; Q_A]$ Transition labels:
Inputs/Outputs

Note that this 4-state finite state machine (FSM) represents a Mealy machine since the inputs and the outputs are shown on the transitions between states (arcs). It generate two independent outputs, one that goes high for one clock cycle when the shaft rotates one output combination change clockwise, namely CW, and the other for one output combination change anticlockwise, namely ACW. The 4 states, i.e., 00, 01, 11 and 10, are represented using 2, D-type flip flops with outputs Q_B and Q_A. Also note with the chosen state assignment **the next state is given by the current input values B and A**. For example, assume we are in state [00] and the shaft rotates one output combination *anticlockwise* giving current input values, $B = 1$, $A = 0$, the next state will be [10] and the outputs are $ACW = 1$ and $CW = 0$.

Design task 1: From the state diagram, create a state transition table. Note that this table will have columns for the Current State, the Next State (that is the same as the current inputs) and for the clockwise (CW) and anticlockwise (ACW) outputs.

Very Important Point Concerning the 74HC193 4-bit up/down Counter

Fundamentally, the CW and ACW signals are used to control whether the counter increments or decrements its current count value respectively, i.e., whether it up or down

counts. The up/down counter has 2 separate clock inputs, one for up counting (CPU) and one for down counting (CPD). For example to make the counter increment, then a rising edge has to be applied to CPU input (essentially a pulse signal created by changing the CPU input signal from a binary '1' to a binary '0' and then back again), while the CPD input is held at binary '1'. The reverse situation applies concerning the clock inputs to make the counter decrement its value. Consequently, when generating the clockwise and anticlockwise signals, we actually **require the complement** of the CW and ACW signals (as defined previously) i.e., $\overline{CW}$ and $\overline{ACW}$, to provide the CPU and CPD clock signals respectively.

Use K-maps to determine the simplified Boolean expressions for $\overline{CW}$ and $\overline{ACW}$. Recall we should group the '0's and any useful don't care states (rather than grouping the '1's and useful don't care states) in the map since we require complemented output functions. In terms of the don't care conditions, one example is the 00 to 11 state transition that we know cannot ever occur.

We now wish to implement a synchronous (clocked from an independent free-running clock) FSM inside a programmable array logic (PAL) device. The PAL has 8 output pins labelled P19 to P12 and 8 inputs labelled P2 to P9. Connect P2 to Contact A and P3 to Contact B.

This exercise can be completed by designing a four-state FSM, where the D-type flip-flops (FFs) having outputs labelled P18 and P19 hold the states. The equations you need to enter in to the PAL compiler should look something like this:

```
P19 = P2
P18 = P3
P15 = <Some function of P3, P2, P18 and P19>
P14 = <Some function of P3, P2, P18 and P19>
P17 = P14 (note: not strictly necessary)
P16 = P15 (note: not strictly necessary)
```

This means that FF output P19 is a registered version of its input P2 (that is connected to A) and FF output P18 is a registered version of its input P3 (that is connected to B). So the Next State is given by the current inputs P2 and P3, and the Current State is FF outputs P19 and P18. The outputs labelled P15 and P14 are the $\overline{CW}$ and $\overline{ACW}$ signals respectively and are functions of the current state (P19 and P18) and the inputs (P2 and P3).

Since the design is a Mealy machine, the outputs P15 and P14 generated by its combinational logic can apparently change directly in response to the inputs P2 and P3. Thus owing to 'contact bounce' in the switch, the inputs P2 and P3 will experience multiple transitions giving rise to unwanted multiple transitions at the output of the combinational logic. However, because we have set the PAL in to registered mode, in effect, the outputs P15 and P14 have already been registered via a D type FF on each output and so in this case we do not have to worry. However, to be on the safe side, we have also explicitly included an additional D-type FF on each output. Thus P17 ($\overline{ACW}$) is a registered version of P14 and P16 ($\overline{CW}$) is a registered version of P15. Thus in reality, we actually have a 'registered' Mealy machine.

Implementation task 1:

NOTE: Make sure that the PAL compiler is in **registered mode**.

Now enter your equations in to the PAL compiler. **Read the Appendix on PALs and the**

PAL compiler if you are unfamiliar with them. The compiler should be running on a stand-alone laptop connected to the programmer.

Implementation task 2: Now connect your PAL to a rotary encoder and your two outputs to 2 LEDs, each connected in series with a current limiting 220Ω resistor. Connect the clock to pin 1 and make sure that the output enable (pin 11) is connected to ground. To test your design start by setting the clock to 10Hz. Make sure that each LED flashes once as the rotary encoder is moved one output combination clockwise or anticlockwise.

Step 2 - count clockwise/anticlockwise pulses

Implementation task: Use the clockwise/anticlockwise signals as up/down count signals for the 74HC193 counter. Connect the output of the counter to one of the 7-segment displays. Note that the 4-bit input to each of the 7-segment displays is labelled ABCD, where A represents the most significant bit (MSB) and D the least significant bit (LSB). Also, make sure the **unused inputs** of the 74HC193 and connected to either 0V or 5V as appropriate.

Assessment

Ticking criteria: Write up your design and answer the following questions. Demonstrate that your rotary position counter works. Once your work has met the **Common Ticking Criteria** (see Introduction), get your work ticked by an assessor. Remember that you need to hand in this assessed exercise as part of your portfolio of work (see the Head of Department's notice).

Questions

1. Why didn't we ask you to debounce the inputs from the rotary encoder?
2. How fast can the rotary encoder be rotated before your circuit fails to count correctly?

Appendix – PALs and the PAL Compiler

A PAL (programmable array logic) is a circuit which can be configured by the user to perform a logic function. It consists of an AND array followed by an OR array, the former being programmable. Inputs are fed into the AND array, which performs the desired AND functions and generates product terms. The various product terms are then fed into the OR array. In the OR array, the outputs of the various product terms are combined to produce the desired outputs.

You will be using a small, relatively modern PAL called a GAL16V8. It has between 0 and 16 inputs and 8 to 0 outputs respectively, a common configuration is for up to 8 inputs and up to 8 outputs. Each output is driven by a single 8-input OR gate, which in turn is driven by eight 32-input AND gates. These 32 inputs can be programmed to be connected to the 16 possible inputs to the chip and their inverses. Note that when used to implement an FSM up to 8 of these inputs are actually outputs from the D-type FFs in each of the 8 output macrocells.
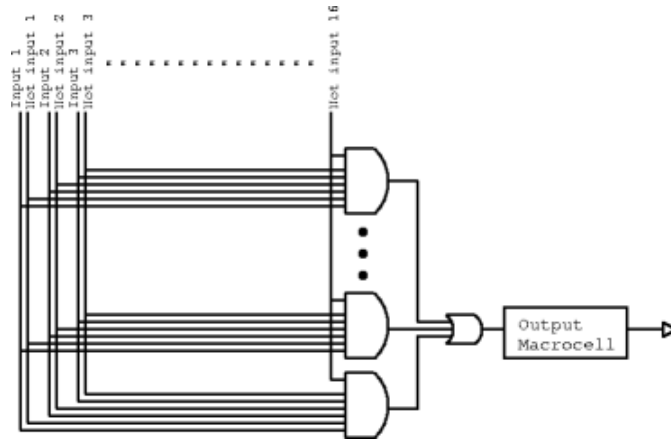
The output from the each OR gate is connect to an output pin through an output macrocell. This can be programmed to latch (using a D-type FF) the output or feed the output signal back into AND array. For a more detailed description of this PAL, read the datasheet.

PAL Compiler

To program the PAL with the functions necessary to implement a state machine, you will need a PAL compiler. This will produce a JEDEC file which is needed by the programmer. You will be using a web-based compiler. You need to type in equations using variables P2 to P9 on the input side (representing the input pins) and P12 to P19 on the output side (representing output pins).

When using the PAL Compiler you should have the above diagram in mind. The compiler expects 8 equations, one to drive each output logic macrocell (OLMC). The OLMCs in turn drive pins 19 to 12. What the 16 inputs are depends on the configuration mode of the PAL. Pins 2 to 9 are always inputs, but in **registered** mode pins 12 to 19 make the other 8, in **complex** mode pins 1, 11 and 13 to 18 are used and in **simple** mode pins 1, 11 to 14 and 17 to 19. The mode can be selected using the check boxes.

Note: We always use the PAL in **Registered Mode** because we need to use D-type FFs in order to implement finite state machines. Indeed, the lab compiler also forces us to use the **'Registered Configuration'** that ensures that the Output Logic Macro Cell (OLMC) contains a D-type FF. Incidentally, a 'Combinatorial Configuration' for the Registered Mode is also available where the OLMC does not include a D-type FF. However, we do not make use of this possibility.
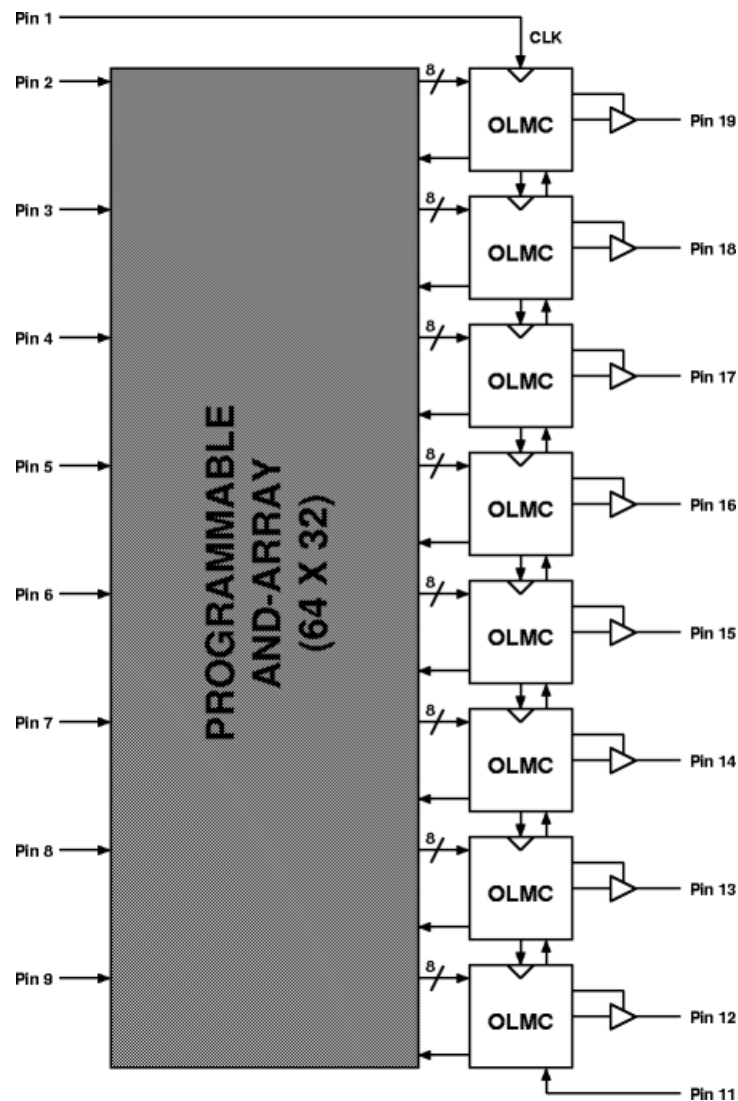
The equations for each output (called P19 to P12, in that order) can consist of inputs, operators and brackets. The input must be referred to by the "P" followed by a number. Operators are &, |, and ! for AND, OR, and NOT respectively.

Examples:

P19 = P2	A simple assignment latch
P18 = P2 & P3	2-input AND gate
P19 = !P19	A 1-bit counter

When you have typed in your equations, click the **compile** button.

Page last updated on 22-Sep-2022

Workshop Three – Debouncing a Switch

Introduction

Do the RS latch design required in the Task in Step 3 and the design required in Step 4 before the practical session.

When using a switch as an input to a digital circuit, several pit-falls may be encountered. The problem of switch `bounce' is the subject of this experiment, what it is, and ways (including an all-digital method) of overcoming it.

Components

- 1A prototyping box
- connecting wire
- 2 x 4K7 resistor
- 74HC00 - NAND gates
- 74HC193 counter
- A PAL - GAL16V8
- Single pole double throw (SPDT) switch

Step 1 - Watch a switch bouncing

Task: Build the following circuit on a breadboard using the **blue** non-debounced push-switch. The supplementary hand-out shows how the voltage at the Output varies as a function of time as the switch is pressed and released. This plot was captured using an oscilloscope using a 'one-shot' trigger mode and varying the timebase setting until the appropriate detail could be seen.
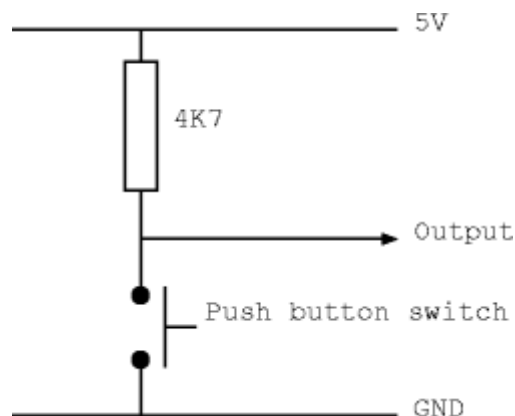
Note: The 4K7 resistor is often known as a 'pull-up' resistor since the Output is pulled-up to 5V when the push button switch is open (not pressed).

Step 2 - Count bounces

Task: Connect the output of the circuit in Step 1 to the clock input of the counter (Note: This is the 74HC193 counter device used in the previous workshop, but make sure you connect the inputs appropriately so that it operates as an up-counter). Connect the counter output to the 7-segment display as was done in the previous workshop. What should happen? Why doesn't it work? The object of the next step is to debounce the switch so that counter

increments just once as the switch contact is made.

Step 3 - Debouncing a single pole double-throw switch

You may be surprised to find that a single pole double-throw (SPDT) switch is easier to debounce than the single-throw switch employed in the previous circuit, so in the first instance we will begin by debouncing a SPDT switch. The following circuit shows a SPDT switch and two pull-up resistors. The switch you will be using is a break before make switch, i.e., it is mechanically incapable of connecting both switch terminals together. What is the relationship between the two outputs if bounces only appear on one contact at a time?
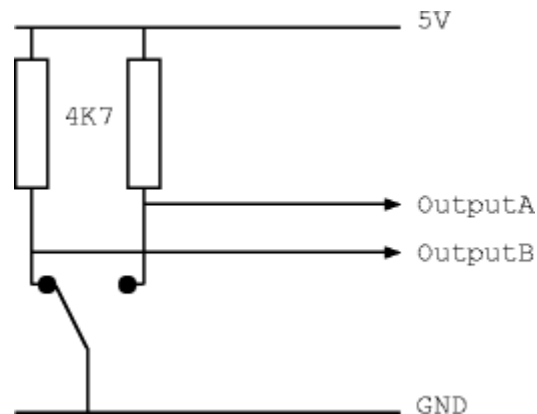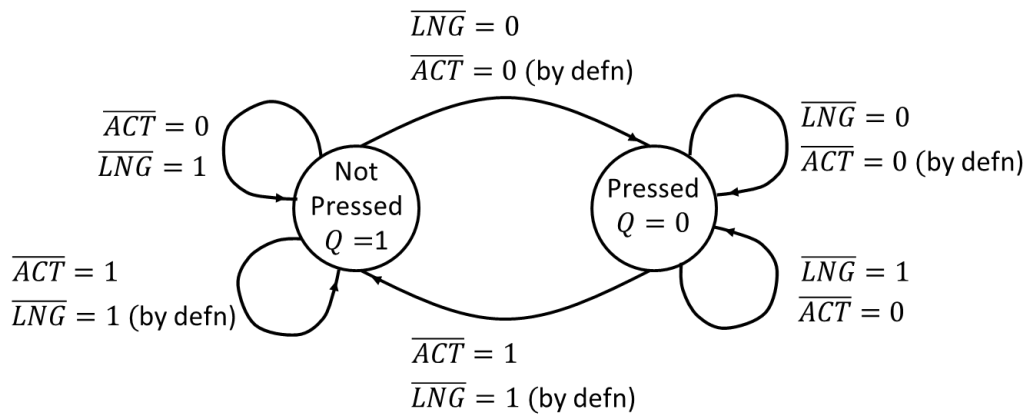

Task: Build the above circuit. Use an RS latch that takes OutputA and OutputB as inputs and produces a debounced output. **Note:** The switch connection arrangement used here produces a 'High' (i.e., 5V) output when a switch contact is open, and a 'Low' (i.e., 0V) output when a switch contact is closed. Thus the logical output from this arrangement is known as 'active low'. This has the opposite sense to what is required at the inputs to the NOR gate based RS latch shown in the Digital Electronics course notes. For example from the course notes we see that a 'High' on the S input (and a 'low' on the R input) gives the 'Set' condition. Use bubble logic to reconfigure the NOR based RS latch to use active low inputs, i.e., where for example the 'set' condition is given by a 'Low' on the S input and a 'High' on the R input.

Step 4 - Debouncing the push button switch

Debouncing a switch can be done in many ways, using a variety of analogue or digital techniques. We will now implement a debouncer using a synchronous digital approach. Consequently we will use a PAL to implement a length 7 shift register (i.e., 7, D-type flip-flops) with the switch output as its input. You should not use the output of the first flip-flop since it may go metastable if the input changes too near to the active clock edge. When all of the other flip-flops (i.e., 6) in the shift register all become 'Low', the output from the debounce circuit should go 'Low'. Similarly, when all the shift register outputs become 'High', the output of the debouncer should go 'High'.

Design: Use a 6-input OR gate to generate a signal, say $\overline{LNG}$ (i.e., active 'Low'), to detect when the shift register outputs are all 'Low'. In addition use a 6-input AND gate to generate a signal, say $\overline{ACT}$ (active 'Low') that detects when the shift register outputs are no longer all 'High', i.e., some switch activity has occurred. Use these 2 signals as the inputs to a 2-state Moore machine having the following state diagram. Note that Q is the required debounced output signal and the transitions between states are determined by the generated 'input' signals $\overline{LNG}$ and $\overline{ACT}$.

The state diagram shows two states "Not Pressed $Q=1$" and "Pressed $Q=0$" with transitions:

- Self-loop on Not Pressed (top left): $\overline{ACT}=0$, $\overline{LNG}=1$
- Self-loop on Not Pressed (bottom left): $\overline{ACT}=1$, $\overline{LNG}=1$ (by defn)
- Transition Not Pressed → Pressed (top): $\overline{LNG}=0$, $\overline{ACT}=0$ (by defn)
- Self-loop on Pressed (top right): $\overline{LNG}=0$, $\overline{ACT}=0$ (by defn)
- Self-loop on Pressed (bottom right): $\overline{LNG}=1$, $\overline{ACT}=0$
- Transition Pressed → Not Pressed (bottom): $\overline{ACT}=1$, $\overline{LNG}=1$ (by defn)

If the machine is in state $Q=1$, then if $\overline{ACT}=1$, $\overline{LNG}=1$ (i.e., button has not been pressed) the machine remains in state $Q=1$. If the machine is in state $Q=1$, then if $\overline{ACT}=0$, $\overline{LNG}=1$ (i.e., button has been pressed, but not for very long) the machine remains in state $Q=1$. The machine will only move to state $Q=0$ once $\overline{LNG}=0$, i.e., the switch has been pressed for a sufficiently long time. The machine will remain in state $Q=0$ if the switch remains pressed, i.e., $\overline{LNG}=0$. The machine also remains in state $Q=0$ if $\overline{LNG}=1$, $\overline{ACT}=0$, since this indicates that the switch has been released, but there is still some switch activity owing to contact bounce. The machine only returns to the not pressed state $Q=1$ once the switch activity has ceased, i.e., $\overline{ACT}=1$.

Write down the state transition table associated with the above state diagram and use a Karnaugh Map to generate the required next state logic for the single D-type flip-flop. Don't forget to make use of don't care conditions (observe that $\overline{ACT}=1, \overline{LNG}=0$ does not occur) to simplify the required combinational logic.

Task: Design, build and test this digital switch debouncer.

Assessment

Ticking criteria: Write up your experimental data and final design, and then answer the following questions. Demonstrate that your debouncer works.

Once your work has met the **Common Ticking Criteria** (see Introduction), get your work ticked by an assessor. Remember that you need to hand in this assessed exercise as part of your portfolio of work (see the Head of Department's notice).

Questions

1. Why is a switch debouncer necessary?
2. How should you choose the clock frequency for the PAL based debouncer?

Workshop Four - Framestore for an LED Array

Introduction

Do the Control Logic design Task required in Step 3 before the practical session.

This aim of this experiment is design, build and test a framestore (a piece of equipment that stores an image) implemented using a Static Random Access Memory (SRAM), two light emitting diode (LED) arrays to display the image and some control logic.

The design uses the SRAM as a framestore for two 5x7 LED arrays. The LED display is made up from individual LEDs which are connected in a grid. Connecting a row (through a current limiting resistor) to the 5V power supply and a column to ground (i.e., 0V) lights the LED where they cross. This means that although each LED can be lit individually, an arbitrary pattern cannot be displayed merely by applying constant voltages to the row and column inputs. Instead, the array must be scanned, preferably at a rate at which the display does not appear to flicker. In this experiment, the pattern to be displayed is to be read from an SRAM.
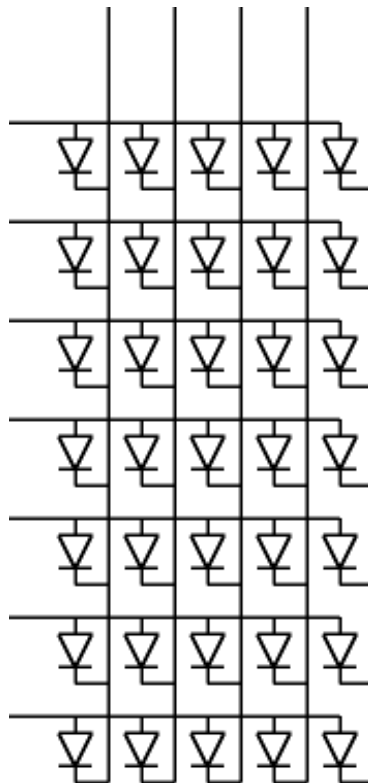

Components

- 1A prototyping box
- connecting wires
- Holtek HT6116-70 SRAM
- 74HC541 octal tri-state buffer
- 74HC00 quad 2-input NAND gates
- 74HC193 4-bit synchronous binary counter
- 74HC74 dual D-type flip-flop
- Binary-coded decimal (BCD) hex switch

Step 1 - Make the LED matrix light up

Task: Use the 74HC193 to produce a 4-bit counter. Make sure that any unused inputs and control inputs are connected to appropriate logic levels to ensure correct operation. Use the 4 bit output value from the counter (Q0-Q3) as the column address for scanning the LED array (Note that the least significant bit (LSB) of the LED array column address is labelled 'A'). Connect the array row inputs to logic 1 (i.e., to 5V). Note that devices within the dotted box labelled 'Prototype Box' are pre-built and you just need to provide connections to the array row and column inputs.

Step 2 - Connect the SRAM to the LED matrix

Task: Also use the 4-bit counter output value (Q0-Q3) as the address input (A0-A3) of the SRAM and connect the data output (the 7 lower data bits, i.e., D0-D6) from the SRAM to the rows of the array (make sure you first remove the connections between the row inputs to the array and the 5V supply you made in Step 1). Also make sure that the $\overline{CS}$, $\overline{OE}$, and $\overline{WE}$ inputs (note are all active low) are connected to appropriate logic levels so that the SRAM is in read mode. Unused address and data inputs on the SRAM should be connected to ground (i.e., 0V).
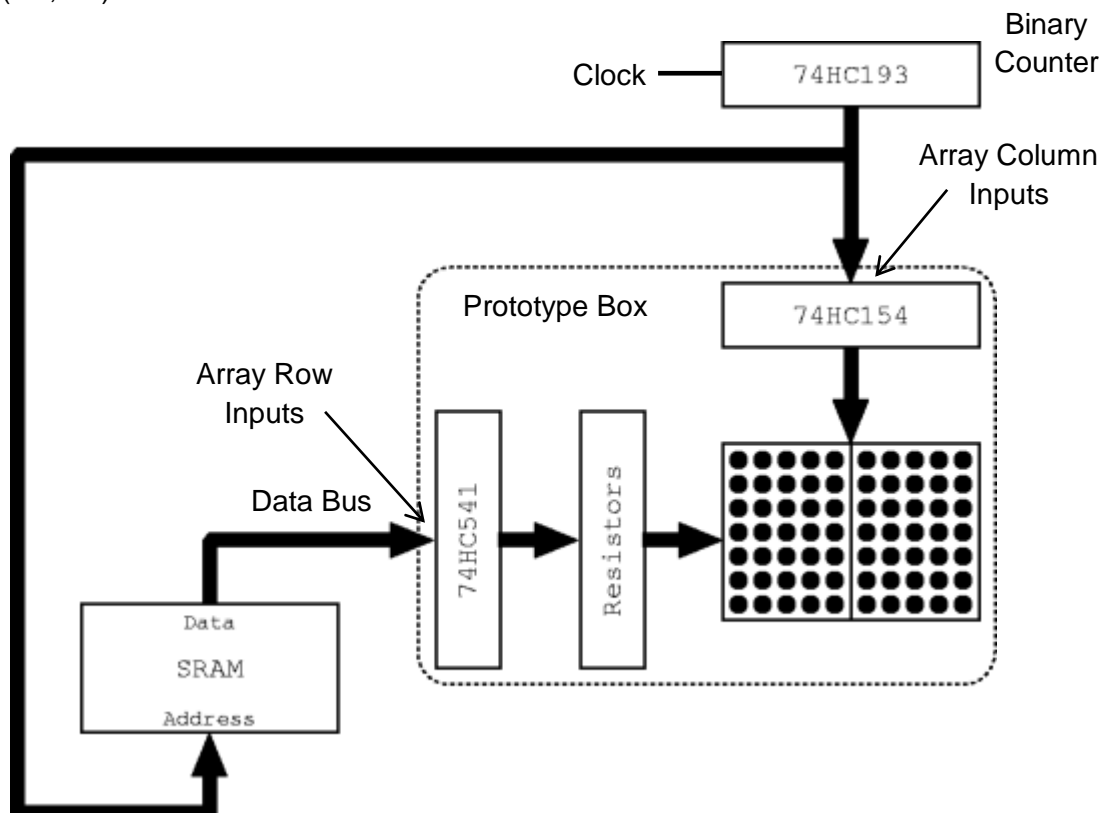
The array should now display the random contents (since we have not written any specific data to the SRAM yet) held in the SRAM. Turn your power supply on and off several times to see if the pattern changes on the LED arrays. Make sure you leave the circuit turned-off for at least 20s!

Step 3 - Add an SRAM write circuit

The next part of this experiment is to add some control logic to allow values to be written into the SRAM. The complete circuit should look something like this:
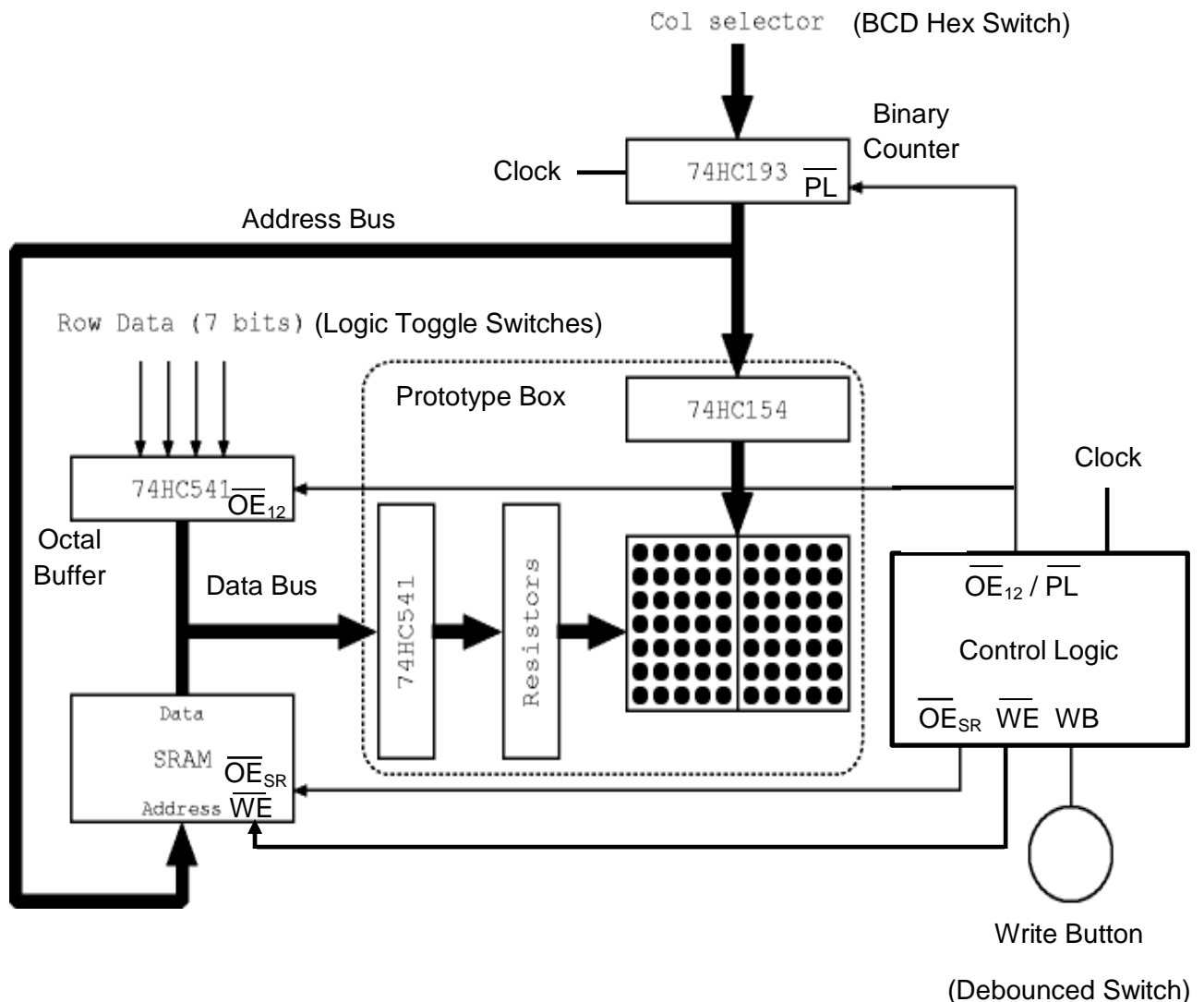
Connect 7 of the outputs of the 74HC541 octal buffer (Y0 to Y6) to the 7-bit Data Bus (i.e., 7 data pins (D0 to D6) of the SRAM), and use seven logic toggle switches as its data input (A0 to A6). Note the 74HC541 has 2 active low output enable ($\overline{OE}$) inputs that should be connected together (labelled $\overline{OE_{12}}$). Now connect the binary coded decimal (BCD) hex switch (see hardware labs data book) to the parallel data input of the binary counter. The BCD switch acts as a column selector, i.e., it allows a desired column in the display to be selected so that it can be written to. When complete, you should be able to write the data from the logic toggle switches into the SRAM at the address supplied by the column selector. Note however, that this should only happen when the Write Button is pressed.

Task: Design the Control Logic – Do before the practical session

The required state machine has 2 states, i.e.,

(i) Display the stored image (**Disp**) and;
(ii) Load the desired image to the chosen LED array column, i.e., the chosen SRAM address (**Load**).

We can represent the required operation in the form of a state diagram. On the transitions (arcs) between states we will indicate the input, i.e., the Write Button (WB), and the outputs, $\overline{PL}$, $\overline{OE_{12}}$, $\overline{OE_{SR}}$ $\overline{WE}$. In doing so, we are implicitly describing a Mealy Machine implementation.
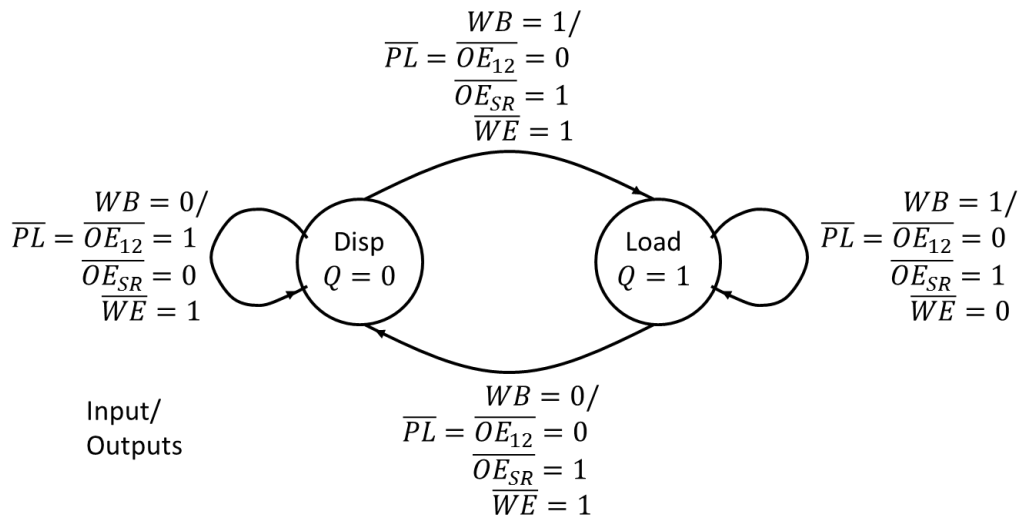
Since we have 2 states, we can implement the machine using a single D-type FF. In addition, notice that the next state Q is given by the output of the write button (WB). So if we are currently in the Disp ($Q = 0$) state and $WB = 0$ (switch not pressed), then the machine remains in the Disp state, the SRAM output is enabled and its addressed content is put onto the Data Bus and so into the LED array. Note that the Octal Buffer output is disabled since we should only connect one output device at a time to the Data Bus (in order to prevent data corruption).

If we are in the Disp state and the write button is pressed ($WB = 1$) we enable the counter to Parallel Load ($\overline{PL} = 0$) the selected column address to the SRAM, the SRAM output is disabled, and the Data Bus now carries data from the output of the now enabled Octal Buffer (that has as its input the desired row data set on the Logic Toggle Switches). Following the clock edge, the machine moves into the Load state ($Q = 1$) and if the write button is still pressed ($WB = 1$), we enable the SRAM write mode ($\overline{WE} = 0$). When the write button is released, the SRAM write control is disabled ($\overline{WE} = 1$) and it is this 0 to 1 transition of the $\overline{WE}$ control signal that causes the data from the Octal Buffer to be written into the selected SRAM address. At the next clock edge we move from the Load to the Disp state. If the write button remains not pressed ($WB = 0$), the machine remains in the Disp state and the output of the SRAM is enabled and the contents of the addressed location is put onto the Data Bus and hence to the LED array.

Write down the state transition table for the machine given in the previous state diagram and use it to generate the required combinational logic to give the required control output signals. Note that:

- The Write Button (called the Debounced Switch on the prototyping box) is connected to the D input of the D-type FF (note 2 are available on the 74HC74).
- Note that $\overline{OE}_{12}$ is the complement of $\overline{OE}_{SR}$ since only one of either the Octal Buffer output or the SRAM output should access the Data Bus at a time.
- The parallel load of the binary counter is asynchronous. The output value changes immediately, and does not wait for the next rising edge of the clock.
- The BCD hex switch (i.e., the column selector) needs 4 x 4K7 pull-down resistors on its 4 outputs. See the Hardware Labs data book for details.
- You might like a demonstrator to check before proceeding to the final design.

The Control Logic can be implemented using a 74HC74 dual D-type flip-flop (FF) and a 74HC00 quad 2-input NAND gate.

Assessment

Ticking criteria: Write up your experimental data and final design, and then answer the following questions. Demonstrate that your circuit works correctly: you should be able to specify a column on the LED array with the column selector and the data pattern for that column with the toggle switches. Pressing the write button should make the pattern permanent.

Once your work has met the Common Ticking Criteria (see Introduction), get your work ticked by an assessor. Remember that you need to hand in this assessed exercise as part of your portfolio of work (see the Head of Department's notice). Note: One tick is for completing steps 1 and 2 and the other for step 3 and answering the questions.

Questions

1. Each LED in the matrix can draw a maximum current of 5mA. What is the maximum current that the two LED arrays can draw during the operation of the frame store?
2. What would happen if the write button was released before the first active clock edge after it was pressed?
3. Note that the debounced Write Button is synchronised with the system clock. What problem may occur if it was not?
4. How might the implementation be changed such that the outputs from the Control Logic change synchronously with the clock?