

ICAR Hardware Lab 2 : Logic functions & Flip-flops

The aim of this lab is to demonstrate how complex Boolean logic functions can be implemented using simple logic gates i.e. AND, OR and NOT gates. These in turn can be used to construct more complex circuits e.g. capable of performing Binary arithmetic. Finally we will look at ICs that can store data i.e. flip-flops, and how these components can be combined with Boolean logic to implement Binary counters i.e. the key building blocks of any computer.

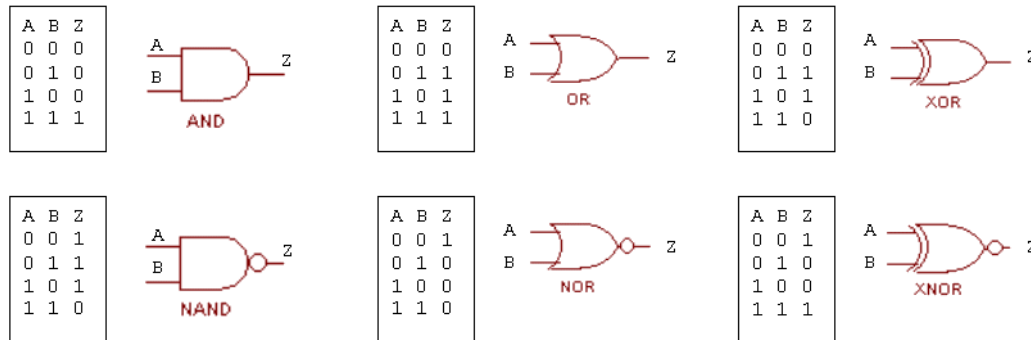


Figure 1: Logic gate truth tables

Task 1

The cockroaches have evolved, they have learnt to only partially enter the trap, if the trap then closes they can then pull themselves out i.e. wriggle backwards. To enable the operator to detect when this has occurred each trap needs to remember if either of the LDR sensors have been triggered during the night. This memory function can be implemented using a flip-flop i.e. a one bit memory device. The logic tutor board contains four J-K flip-flops, as shown in figure 2.

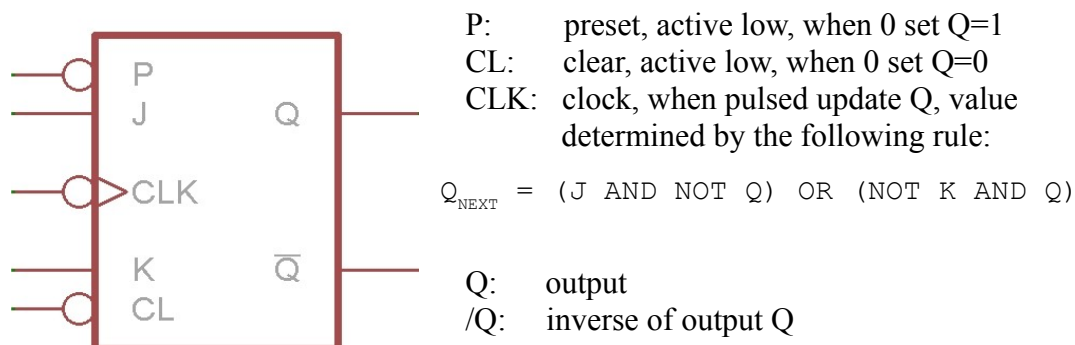


Figure 2: J-K Flip-flop

Using J-K Flip-flop FF3 we can implement a simple SET-RESET memory. First connect the J, K and CLK inputs to a fixed logic 0. Then connect the P input to switch S2 and the CL input to switch S1. Move **both** toggle switches into the inactive state i.e. logic 1. Connect the Q output to LED LP3, as shown in figures 3 and 4.

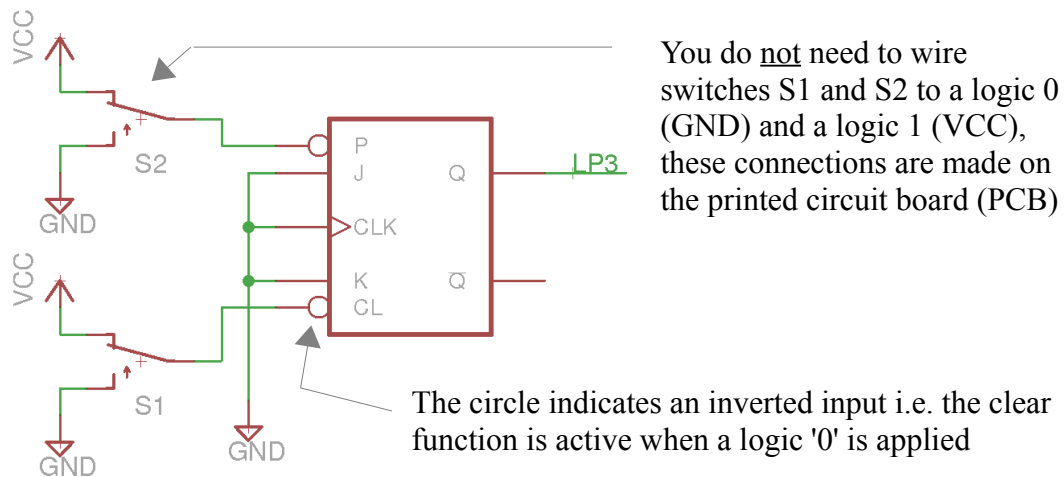


Figure 3: Set – Reset circuit diagram

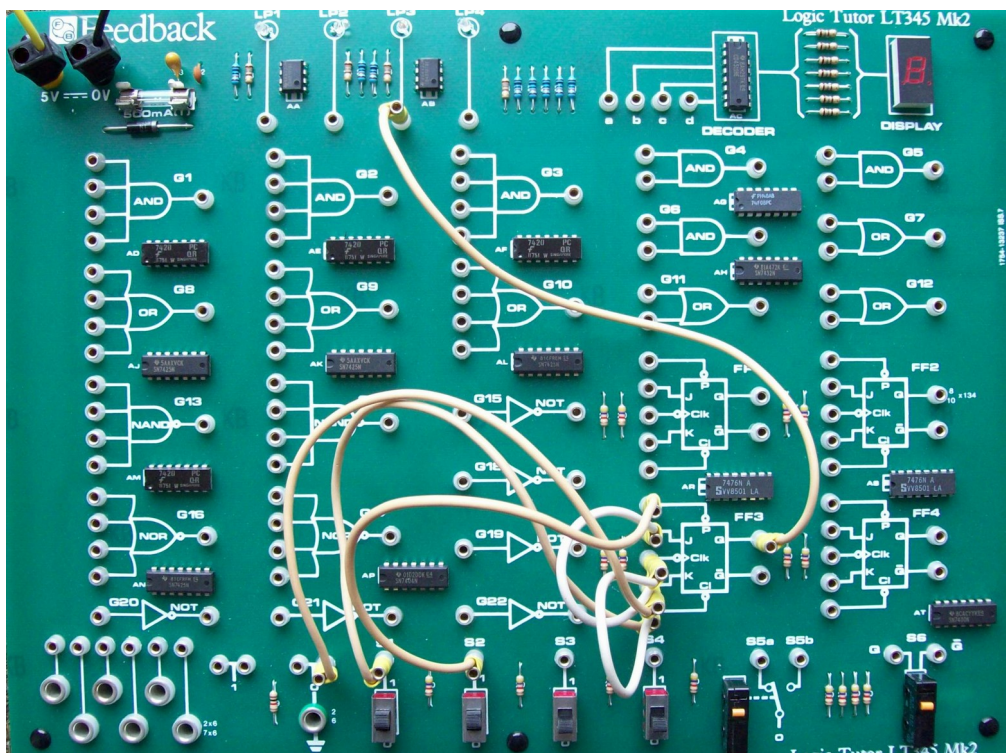
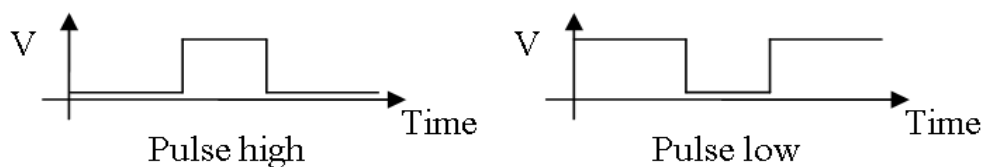


Figure 4: Set – Reset configuration

Question: Pulse switch S1 low i.e. $1 \rightarrow 0 \rightarrow 1$, what is the output Q i.e. LED LP3?
Pulse toggle switch S2 low i.e. $1 \rightarrow 0 \rightarrow 1$, what is the output Q?



You should have observed that when switch S2 is pulsed low the output Q goes to a logic 1 i.e. set, or pre-set. The flip-flop will now store this value until switch S1 is pulsed low, where upon the output Q goes to a logic 0 i.e. reset, or clear.

Hint, the most common mistake made when testing this circuit is not starting with switches S1 and S2 in the logic '1' position.

Question: Using this Set-Reset (SR) flip-flop, design a circuit such that LED LP3 will be set when either of the LDR sensors are triggered and will remain illuminated until switch S1 is pulsed low i.e. manual reset. Implement and test this circuit, design tips are given below.

Tips, use a NOT gate and an OR gate to produce a NOR gate. Disconnect the P input from S2 and connect it to the output of this NOT gate.

Warning, some power supplies generate significant amounts of noise, which can prevent this circuit from working i.e. false signals. In these cases do ask for assistance as we can manually add additional filtering, or change power supplies.

Task 2

To 'counter' this new breed of super cockroaches the rules that control the trap need to be updated to only trigger (enable) the trap if either LDR sensor has been active for approximately 1.5 seconds i.e. the cockroach has fully entered the trap and is eating the bait. To implement this function the block diagram shown in figure 5 will be used.

DO NOT try to build this circuit. The construction of this block diagram is described in Tasks 3 – 7. You may disassemble any previous circuits.

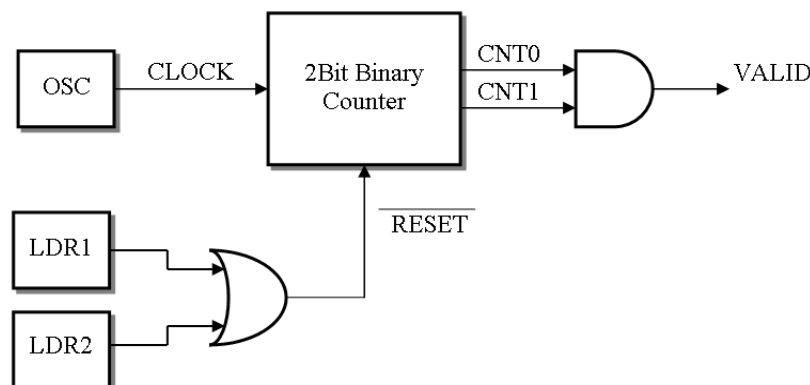


Figure 5 : delayed enable block diagram

In this circuit when the LDR sensors are not triggered the OR gate will output a logical '0', holding the counter in reset i.e. CNT = 00. When either sensor is triggered the RESET line is set to a logic '1', the counter will then increment its value each time the OSC clock input is pulsed low i.e. the counter will generate the output states: 00,01,10 and 11. When the count value reaches the value 3 i.e. 11, the VALID signal is set to a logic 1, triggering the trap. This counter circuit requires a repeating square wave waveform i.e. a clock, as shown in figure 6. Connect the square wave output (OSC) socket on the trap to an LED. Observe the frequency (speed) of this square wave and estimate the waveform's period i.e.

Frequency = number of pulses per second (units Hertz)
 Period = $1 \div$ Frequency (units Seconds)

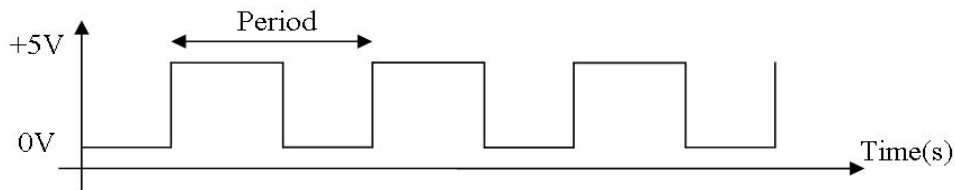


Figure 6 : square waveform

Optional Task

To view this electrical waveform we can also use an oscilloscope (scope), as shown in figures 7. This piece of test equipment allows you to view voltage signals as a graph, the vertical axis is the voltage level and the horizontal axis time.

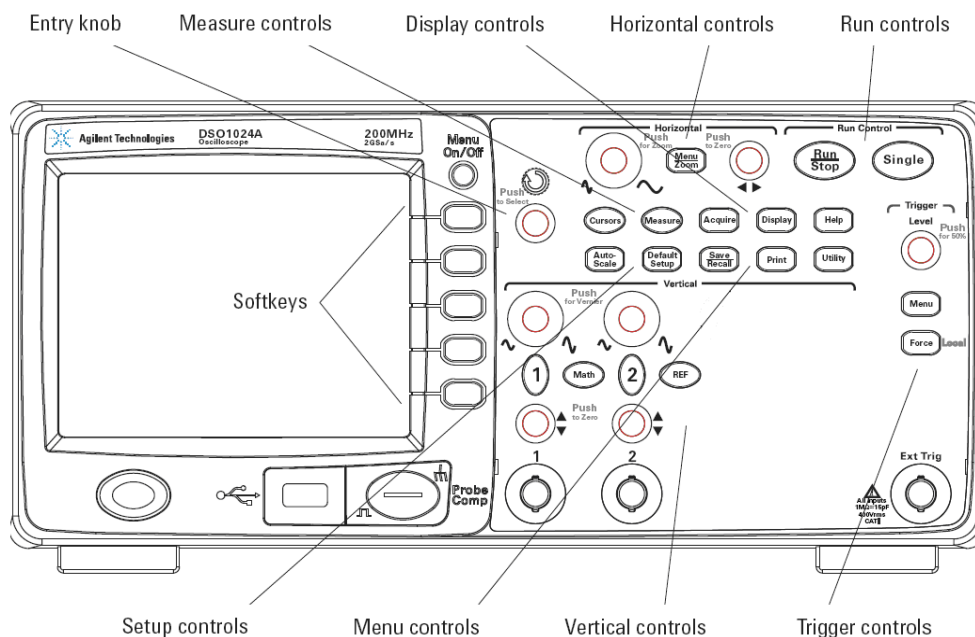



Figure 7 : An oscilloscope

To switch the scope on press the oval power button  at the bottom left of the case. The main control buttons will be illuminated as the scope initialises itself to the last user configuration.

To connect a scope probe, align, push, then twist the connector onto the silver BNC connectors labelled 1 located at the bottom middle (as discussed in Lab 1). Clip the probe onto the square wave output (OSC), then connect the black crocodile clip to the logic '0' (0V) socket, as shown in figure 8. To view this waveform, press the 'Auto-scale' button within the Setup controls panel (located in the middle of the scope) as shown in figure 7. This will configure the scopes internal circuitry to correctly display this waveform. Using the scope measure the signals maximum and minimum voltage level and the square wave period (time high + time low).

Note, sometimes auto-scale doesn't work, you may have to manually set up the scope as shown in the bottom right of figures 8 and 9. If you are not sure how to do this, do ask.

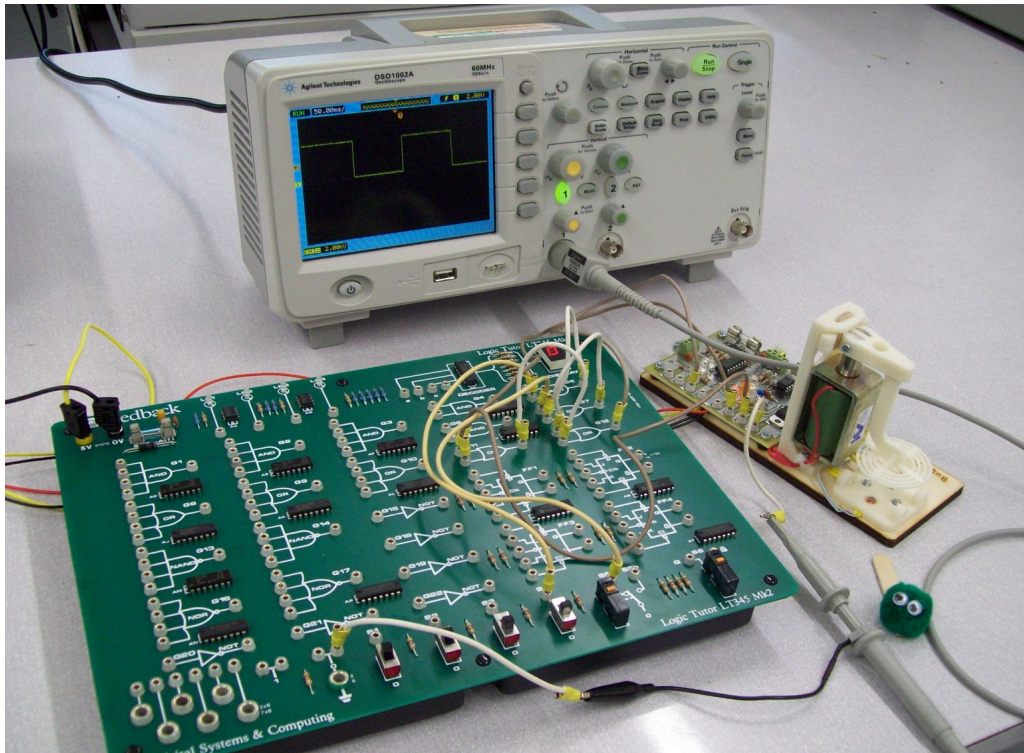


Figure 8 : oscilloscope setup

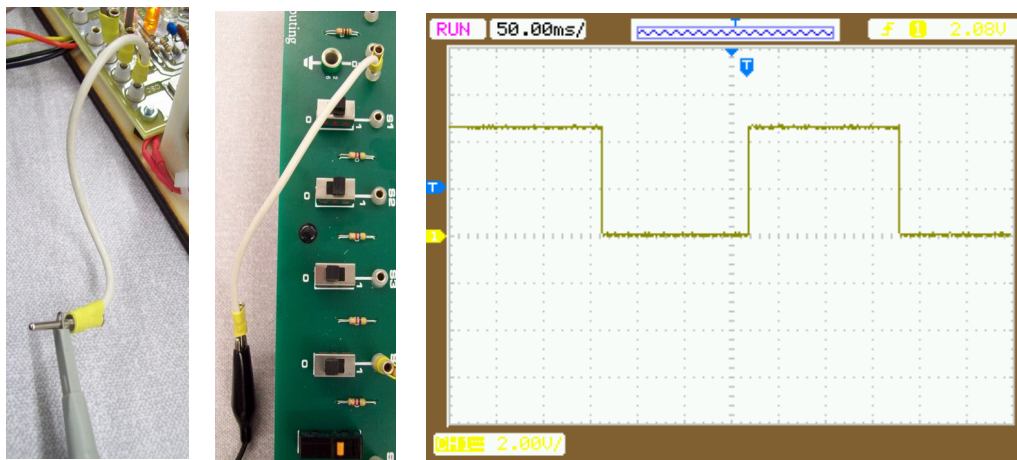


Figure 9 : Connecting the oscilloscope probe, oscilloscope screen (bottom)

INPUT	OUTPUT	Description
$A_1 A_0$	$Z_1 Z_0$	
0 0	0 1	$0+1 = 1$
0 1	1 0	$1+1 = 2$
1 0	1 1	$2+1 = 3$
1 1	0 0	$3+1 = 4$ (only stores lower two bits)

Figure 10 : increment truth table

Task 3

A binary counter is constructed from two basic functional blocks:

- Adder : a logic circuit to perform an **increment** function to generate the next count value i.e. $\text{count} = \text{count} + 1$.
- Memory : flip-flops, a **register** to store the current count value i.e. in this case a 2 bit number.

Question: The truth table for a two bit binary incrementer is shown in figure 10. What two logic gates are needed to implement this circuit i.e. generate outputs Z_1 and Z_0 ?

Hint, what is the relationship between A_0 and Z_0 , what truth table in figure 1 matches output Z_1 ?

The required logic gates needed to implement a 2 bit incrementer are shown in figure 11. Do not construct this circuit as we can make some optimisations to reduce the number of components used.

Task 4

The input to the incrementer i.e. A_1 and A_0 , is the current count value. This data must be stored in a 2bit register. To store this data the J-K flip-flops can be configured as Data flip-flops or D-type flip-flops, as shown in figure 12.

[https://en.wikipedia.org/wiki/Flip-flop_\(electronics\)#D_flip-flop](https://en.wikipedia.org/wiki/Flip-flop_(electronics)#D_flip-flop)

Using J-K Flip-flop FF3 and NOT gate G21, construct the circuit shown in figure 12. Next connect the signal DATA_IN (NOT gate input) to S1, the RESET signal (C1) to S2 and the CLOCK signal (Clk) to the Q output of micro-switch S6. Finally connect the signal DATA_OUT (Q output of FF3) to LED LP2.

Questions:

- Move the toggle switch S2 (RESET) into the inactive state i.e. logic 1, and the toggle switch S1 to logic 1 i.e. DATA_IN.
- Reset the flip-flop by pulsing low the RESET signal i.e. S2, $1 \rightarrow 0 \rightarrow 1$. What is the state of DATA_OUT?
- Clock the flip-flop by pulsing high the CLOCK input i.e. press the red button on micro-switch S6. What is the state of DATA_OUT?
- Set DATA_IN to a logic 0, what is the state of DATA_OUT? Next clock the flip-flop again. What is the state of DATA_OUT now?

What you should observe is that when switch S2 is pulsed low the output Q goes to a logic 0 i.e. the flip-flop is cleared / reset. The flip-flop will now store this value until switch S6 is pulsed high, where upon the output Q is updated with the value on the input DATA_IN. Unlike the SR flip-flop we only need to pulse the CLK pin to update the output i.e. you only need to place the data to be stored on the DATA_IN input, you do not need to select different inputs (Set / Reset) to store different logic values.

Note, only a voltage transition from a logic '1' to a logic '0' on the CLK pin will trigger the output to be updated i.e. an update is performed on the falling edge,

indicated by the circle on the CLK input, not by the logic level.

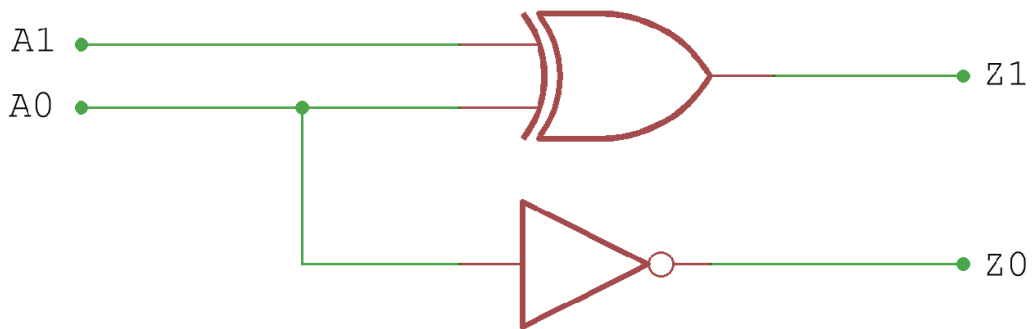


Figure 11: 2bit incrementer

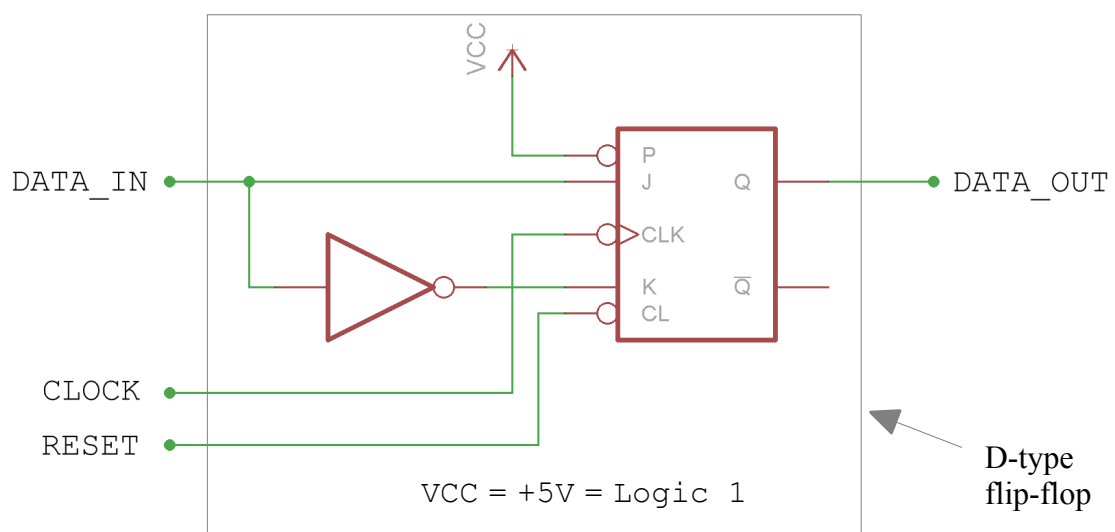


Figure 12 : D-Type Flip-flop

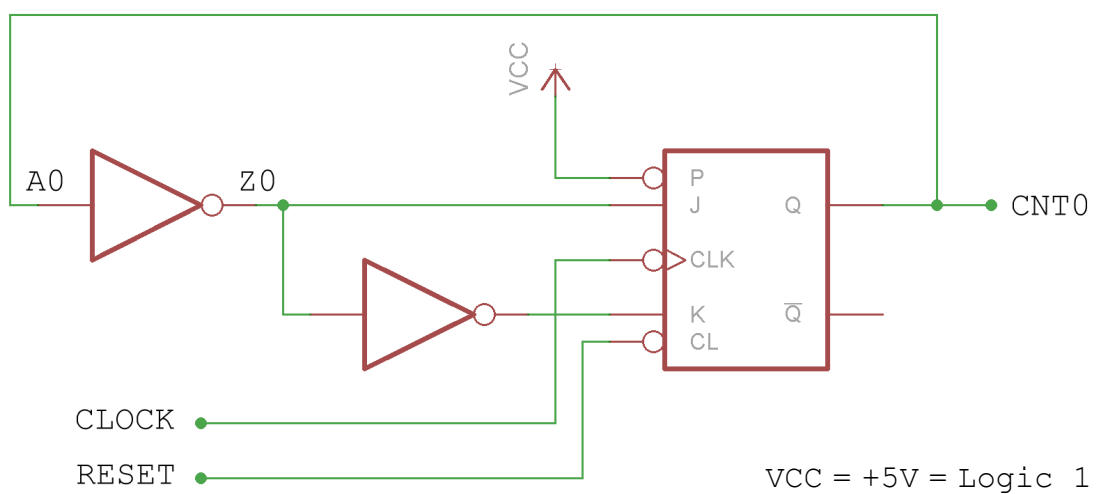


Figure 13 : Counter circuit for bit CNT0

Task 5

To construct a counter the circuits in figure 11 and 12 need to be combined. Each bit of the 2bit counter i.e. CNT0 and CNT1 will need to be stored in a flip-flop. The combined circuits to implement the least significant digit CNT0 is shown in figure 13. However, using logic minimisation techniques we can reduce the number of logic gates required. The J-K flip-flop already has a inverted output i.e. \bar{Q} . An alternative implementation is shown in figure 14, confirm that the functionality of this circuit is the same as figure 13?

Hint, $\text{NOT}(\text{NOT } A) = A$.

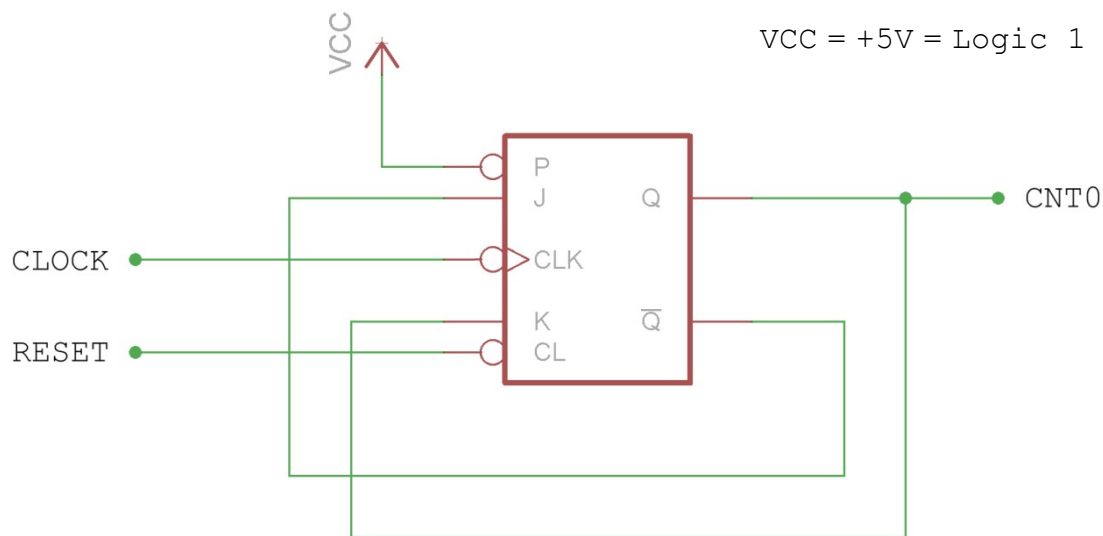


Figure 14 : Alternative implementation for counter bit CNT0

Using J-K Flip-flop FF4 construct the circuit shown in figure 14. Next connect the RESET input to S2 and the CLOCK input to the Q output of micro-switch S6. Finally connect the Q output of FF4 to LED LP1.

Question: To test this circuit:

- Reset the flip-flop by pulsing low the RESET signal i.e. S2.
- Repeatedly clock the flip-flop by pulsing high the CLOCK input i.e. press the red button on micro-switch S6. What is the state of DATA_OUT, how does this output change each time the flip-flop is clocked?

$$Q_{\text{NEXT}} = (J \text{ AND NOT } Q) \text{ OR } (\text{NOT } K \text{ AND } Q)$$

What you should observe is that when the flip-flop is clocked the output CNT_0 should toggle i.e. alternate between a logic 0 and a logic 1, as shown in the truth table in figure 10.

Task 6

The circuit to generate the most significant bit of the counter: CNT1, is an XOR gate, as shown in figure 11. Unfortunately the Logic Tutor board does not contain

that this circuit implements a 2bit binary counter i.e. it will generate the binary sequence: 00, 01, 10, 11, then overflow back to 00.

Task 7

Using the 2 bit binary counter constructed in task 6 construct the block diagram shown in figure 5. Connect the `VALID` signal to an LED, check that this signal goes high when the cockroach has been in the trap for approximately 2 – 3 seconds. Then use this signal to drive the `SOLI` input on the bug trap, triggering the solenoid to close.

Tips, disconnect toggle switch `S2` and micro-switch `S6`. Use AND Gate `G5` and OR Gate `G7` to generate the `VALID` and `RESET` signals.

Task 8

You should observe that the `VALID` signal does not remain high i.e. the count will overflow back to 00. Therefore, to hold the cockroach in the trap we need a circuit to remember that the trap has been triggered i.e. a Set-Reset (SR) flip-flop, as described in task 1.

Using the J-K Flip-flop `FF2` and NOT gate `G15` implement a SR flip-flop. Connect this flip-flops clear line `CL` to switch `S2` (manual reset) and set it to a logic '1', its inactive state. Connect the `Q` output of `FF2` to an LED, does this signal now remain high when the trap is triggered?

If your circuit works correctly, connect the `Q` output of `FF2` to the `SOLI` input on the bug trap, then test if your new and improved bug trap works.

Additional Task

A more complex bug trap is discussed on:

http://www-users.cs.york.ac.uk/~mjf/bug_trap/index.html

Can you improve upon its functionality?

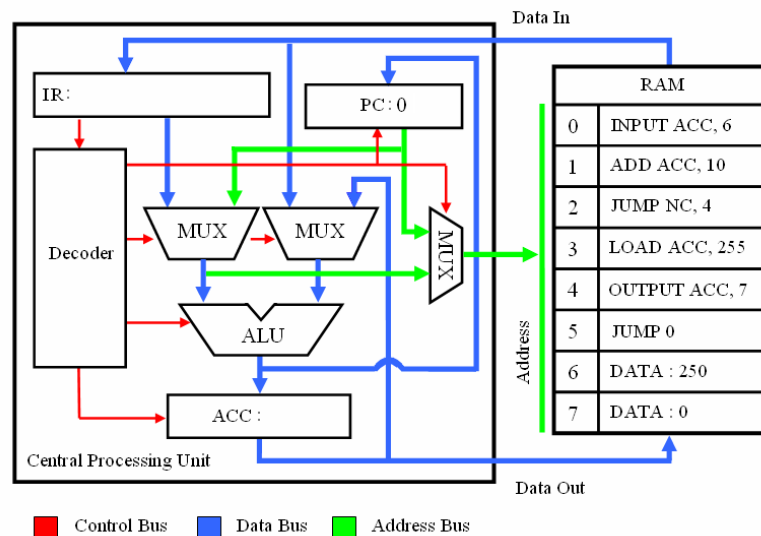


Figure 16 : A typical computer architecture

Summary

The counter implemented in this lab uses the three main building blocks found within a computer i.e. data processing, memory and control.

Data processing : the simple incrementer circuit shown in figure 11 can be expanded to implement hardware capable of performing addition, subtraction, multiplication and division functions. These functions are normally grouped together within a computer's arithmetic and logic unit (ALU), as shown in figure 16. As computers become more specialised increasingly complex hardware modules are being developed to increase processing performance, however, these are still built upon basic logic gates i.e. AND, OR and NOT gates.

Memory : to perform arithmetic operations a computer must also contain memory, storage locations for the data to be processed and any results produced. These are typically implemented using one or more registers, each storing multiple bits e.g. 8, 16, 32 bit etc. Each bit within a register is stored in a flip-flop e.g. if we have a 8 bit register there will be 8 flip-flops. Registers are classified as special or general purpose. Special purpose registers contain specific information the computer requires to function. These are not generally accessible to the programmer e.g. the instruction register (IR), stores the current instruction being processed, or the program counter (PC), address of the instruction to be executed. General purpose registers contain operands and results processed / produced by the ALU. A computer may have one or more general purpose registers, if it has only one this register is typical called the Accumulator (ACC), as shown in figure 16.

Control : a computer must generate a sequence of control signals (bits), instructing each functional block e.g. multiplexer (MUX), ALU, or register, within the computer what it should do and when it should do it. The 2 bit counter shown in figure 15 is an example of a simple state machine. This functionality can be expanded to control any number of bits, generating any sequence of control signals i.e. a micro-sequencer. This is represented in figure 16 as the decoder block, generating the step-by-step sequence of events (control signals) within the computer to process each instruction.

Appendix A

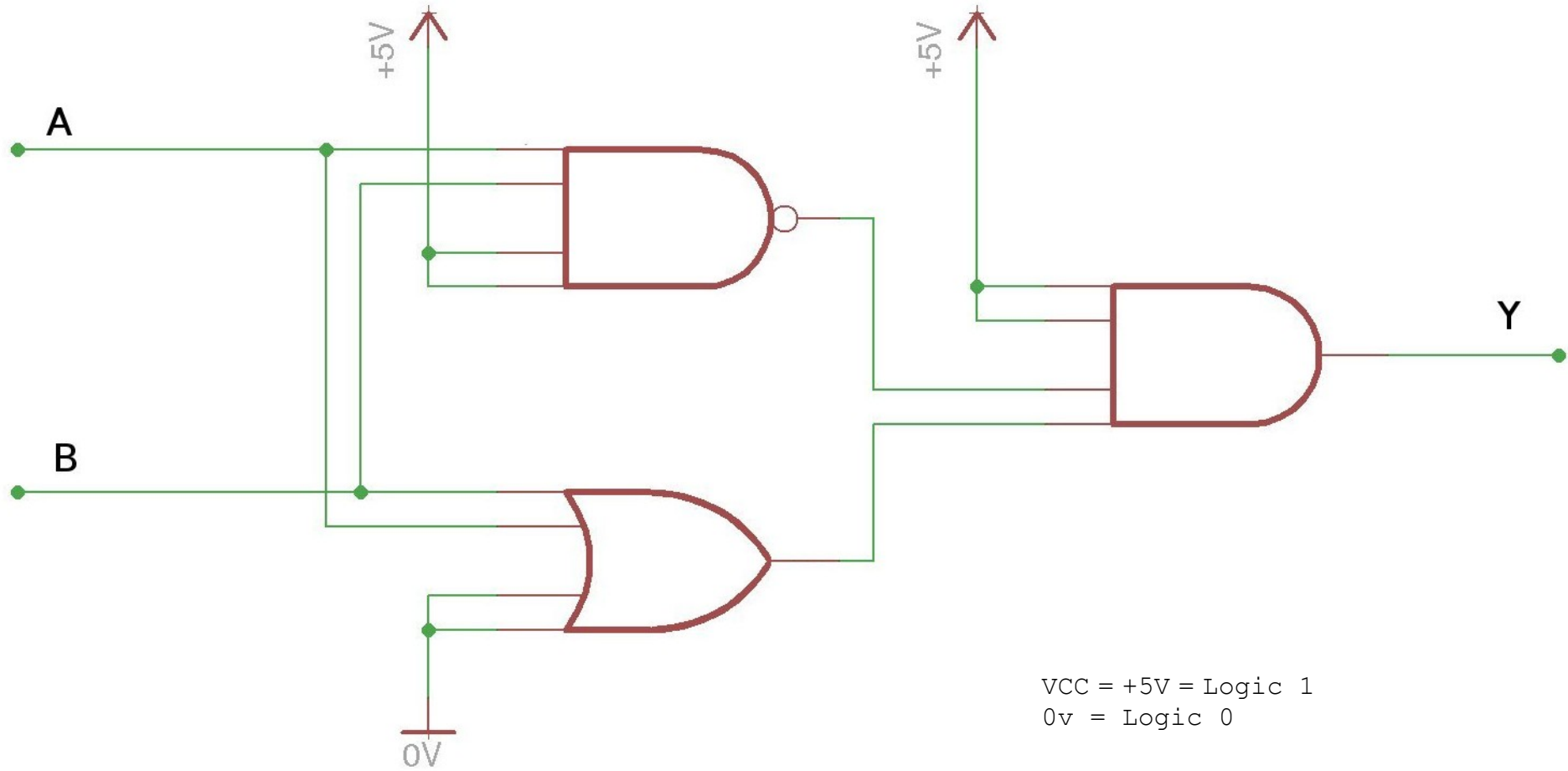


Figure A1: XOR gate circuit, implementation 3