Practical Module 1: Logic Circuits

This document contains the background information and exercises for the first module of the 2IC30 Computer Systems practical.

Overview

The first three sessions of the 2IC30 Computer Systems Practicals are devoted to the design and implementation of digital circuits, also called logic circuits. The designs' final solutions, in the form of a circuit, will be physically implemented. For this purpose, a set of digital-circuit experimental boxes containing gates and other circuits is made available.

Working as a pair you will need to present your solutions to the Tutor or Student Assistants. Required elements of the solution are highlighted in **bold** in the question text. *If your partner is not present when the solution is presented then they will not receive a grade*. The exercises that are to be graded are indicated in the schedule below.

Session Schedule

The below schedule aligns the questions with the contents of the lectures. You are free to submit solutions in advance (e.g. session 2 questions in session 1), or behind schedule (e.g. session 2 questions in session 3). Submission of solutions outside of sessions 1-3 is only possible by prior agreement with the tutor and may incur a grade penalty.

Session	Preparation	Graded Exercises
1 Truth Tables and Boolean Functions	Watch Lecture 1 Book – Chapter 1 Watch – Practical Overview	1.3 Clock Selector (3 points) 1.4 Logic Unit (3 points) 1.5 Not-a-minority element (3 points) 1.5 bonus (1 point)
2 Synthesis and Optimization	Lectures 1 – 3 Book – Chapter 1	2.1 2-Level Multiplexer (3 points) 2.2 Multi-level circuits (3 points) 2.3 Carry look-ahead adders (3 points) 2.3 bonus (1 point)
3 Sequential Circuits	Lectures 1 – 6 Book – Chapters 1-3	3.1 On-Off switch with Synchronous Reset (3 points) 3.2 An abstract state machine (3 points) 3.3 4-bit Register with Parallel Load and Rotate (3 points) 3.3 bonus (1 point)

TOP

Grading Scheme

The grade is applied to the entire question, not the sub-elements. Unless specified or agreed with the tutor, simulations are not accepted as substitutes for implemented circuits. You are permitted to resubmit solutions, but only in entirety.

0 points:	Circuit does not behave as expected and design elements not present/as expected
1 point:	Circuit behaves as expected though some design elements not present/as expected
2 points:	Circuit does not behave as expected but all design elements as expected
3 points:	Circuit behaves as expected and all design elements as expected

There is a total of 27 points. Bonus points can be used to increase your total, but there is a cap of 27 points.

Submission Requirements / Style Guidelines

To ensure submissions are 'as expected' (see above), please see the following;

- Submissions (excluding the physical circuit) may be presented on paper or electronically.
- Designs must be legible (e.g. block script rather than cursive, aligned cells in table columns and rows)
- Truth tables should use binary row ordering, inputs on the left of the table, outputs on the right of the table.
- Karnaugh maps must use Gray code, and groupings must be indicated.
- Boolean expressions should use the same notation as presented in lectures
- Circuit Diagrams: input/output signals, terminals, busses etc should be labelled. Use block print to aid in clarity. Logic gate symbols should be as presented in the lectures/textbook. Minimise use of duplicate input signals, and do not use logic operators on input labels. Use appropriate symbols to differentiate where signal wires cross or join.
- State diagrams: All states should be labelled. Input and output signals should be clearly differentiated, i.e. inputs should be on edges (use Boolean expressions where necessary), output on nodes.

These requirements apply to the practical sessions AND the final examination. You should use the practical sessions as an opportunity to ensure you are well versed in producing full, structured and legible solutions – *you will not be prompted for these requirements in the final examination*.

TOP

IMPORTANT:
Working Methodology

Prior to the practical session:
- Work with your partner to ensure you both understand the problems and the solutions.
- Simulate, if needed, any circuits to check your design.
- Make a note of any questions you may have for tutors – ask these at the start of the session when tutors are not so busy grading other groups.

During the practical session:
- You will be given a numbered box to work with
- Tutors may make a note of the number
- You are responsible for the equipment; please report any faults/breakages to tutors
- There will be a 15 minute break between hours 1+2 and 3+4 (no assistance or grading)
- The last 10 minutes of every session is for tidying up and returning equipment (no assistance or grading, op is op!).

While building/changing your circuits:
- Keep the power of the BOX *OFF* to avoid accidentally short circuits.
- An output must only be connected to an input – never to another output.
- One output may be connected to several different inputs, but inputs must only ever be connected to a single output.
- Remove wires by pulling the base – do not pull the wire itself; this causes the wire to become detached from the plug although visibly they will not appear broken.
- Use wires of the same colour to connect input switches/signals to the main circuit, and another colour to connect circuit outputs to LEDs. This will assist you in tracing errors and in explaining how your circuits work when being assessed.
- Similar to programming, you should build your circuit in stages and test as you build. This will help you avoid errors caused by malfunctioning chips or wires.

Getting help and feedback:
- In general, a working circuit is indicative of a 'correct' design. For some exercises there are multiple solutions.
- If your circuit does not function as intended then tutors will first check the validity of your design. They will help you locate any errors in your design. If the design is 'correct' then you may be advised to rebuild the circuit in stages and test as you build.

Assumptions that often cause issues:
- All hardware is fully functional
- Unconnected gate inputs default to logic 0
- A single output can be connected to many, many inputs
- You cannot approach tutors outside of the instruction sessions to ask for assistance in understanding the problems set or how to approach them

TOP

Background

The TTL-family is the oldest family of digital integrated circuits in existence: they were introduced in 1963 and are still in existence, although nowadays CMOS circuits usually are preferred. As an introduction to working with digital circuits, TTL is as good as any other family.

> The use of the word "family" suggests, rightly so, that there is some coherence between different circuits within the same family: all devices within a single family adhere to the same standard regarding voltage levels and currents, both in their inputs and in their outputs. As a result, devices from the same family can be freely connected without knowledge of these voltages and currents. That is, one can use these circuits without having to know how the logical values 1 (or: true) and 0 (or: false) are represented by physical voltages and currents.
>
> Only when one wishes to connect components that do not belong to the family, like push buttons or LED lights, one must know the voltages and currents involved exactly: in order that the circuits work as they should, their input/output specifications must be respected at all times. Similarly, when one wishes to connect digital devices belonging to different families, with different I/O specifications, one must thoroughly understand these specifications. To give a simple example: the signals in the TTL-family are currents[1], whereas the signals in the CMOS-families are voltages[2].
>
> In the case of our experimental boxes we do not have to worry about the physical signals, though, even when we wish to connect push buttons or LEDs: this has been taken care of by the designers of the experimental boxes already. As a result, you do not have to know anything about the TTL-family: you can treat the circuits on the experimental box as abstract logical devices. Isn't that nice?

In the practical instructions the experimental box on which you will build your circuits will be simply referred to as "the BOX". Inputs to your circuits will generally be one or more of the toggle switches. The outputs will be connected to LEDs.

Where you are asked to perform measurements this will involve building a circuit and cycling through all the possible input combinations by toggling the switches and noting the output state of the LEDs. You can simulate all the circuit designs required in software. The recommended package is CEDAR Logic. Note that the BOX does not have an unlimited supply of gates, it may be necessary to construct gates such as NAND, NOR from other gates.

Independently of which logic family is used one very important design rule must be adhered to under all circumstances: ***never ever connect two (different) outputs!*** If the values of the two outputs happen to be equal – both 0 or both 1 – , nothing will happen but if their values happen to be different a connection between these outputs is equivalent to a short circuit: both circuits will be destroyed immediately, silently, and usually without fire or smoke. But possibly a bad smell.

For the same reason an output may never be connected to (one of the points named) "VCC" or "GND" – the "plus" and the "minus" of the power supply – : these are to be considered as outputs too. The connections "VCC" and "GND" represent constant logical values: "VCC" represents a constant value 1 whereas "GND" represents a constant value 0 . These should be used where an input is required to have a defined value.

[1] No current is a logical 1 and a negative current is a logical 0
[2] A high voltage, like 5 V , is a logical 1 , whereas a low voltage, like 0 V , is a logical 0

TOP

Session 1 - Truth Tables and Boolean Functions

During this session we will focus on building circuits from logic expressions. The first two activities are for you to become accustomed to using the box and are not graded.

Pre-Practical Session

The lectures have covered sections 1.1 – 1.9 of the course text. The material is this chapter will be needed to complete the following exercises.

Work through the exercises in Chapter 1 of the course text. Ensure you complete and understand exercises 1.5.1 and 1.6.4. You should prepare in advance by answering Exercise 1.3, parts a-e and the design for g

1.1 Introduction to the board

Construct the following circuit on the BOX;

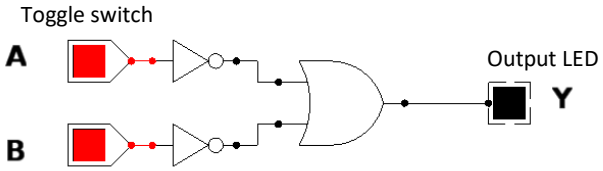


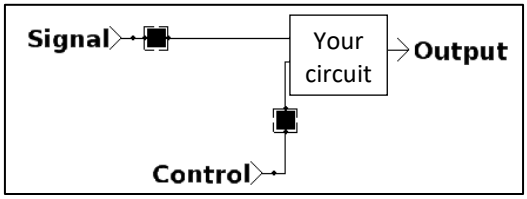
Figure 1 Two inverters and an OR-gate

a) Determine the truth table for this circuit by means of measurements. Construct an empty truth table with 3 columns (A, B, Y). To "measure" the circuit, connect two switches to the two inputs, and connect an LED to the output. Cycle through all the combination of inputs and note the output.
b) Examine the truth table; what logical function does this circuit realize?
c) Would it be possible to replace this circuit by an equivalent single gate from the "family" on the BOX? If so, which one?
d) What would you think that the word "equivalent" is supposed to mean here?

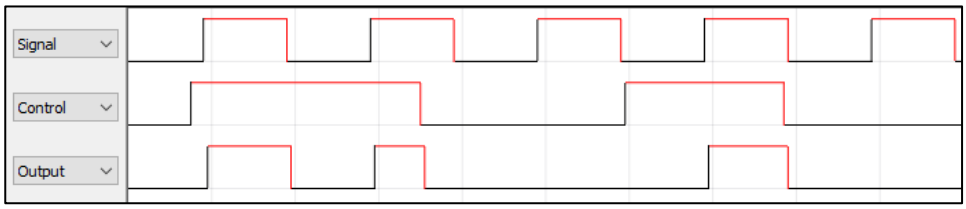
1.2 Behaviours to Truth Tables to Circuits

a) A gate circuit is required with the following behaviour:

When Control=1 : Output=Signal
When Control=0 : Output=0

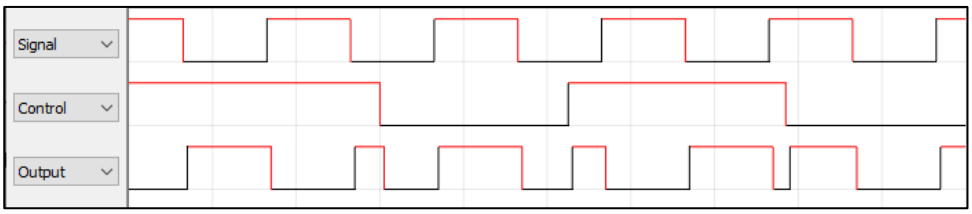


The behaviour of which can be seen in the below time-line:



Construct a truth table for the above behaviour. Label the inputs as Signal and Control. Use the textbook or other resources to identify the single logic gate that can implement this behaviour.

b) Examine the following timeline and define the behaviour in a manner similar to part a) above. i.e. write two statements beginning '*When control = …..*". Construct the truth table with Signal and Control as the inputs and identify the logic gate that can be used to implement the required behaviour.



1.3 Clock Selector

a) Draw the **circuit diagram** for a 4-input multiplexer using individual logic gates.
b) **Implement the circuit** on the BOX. Use toggle switches for the selectors, and use 4 output frequencies from the oscillator as the inputs to the multiplexer. Connect the output of the multiplexer to an LED.
c) Check that the circuit functions as intended; each combination of selector inputs should result in the LED flashing at a different frequency. Adjust the circuit to ensure there is a sensible relationship between the binary value of the selectors and the frequency selected.
d) Model the behaviour of the circuit using a **truth table** to show the relationship between selector combination (input) and frequency (output).

1.4 Logic Unit

a) Adjust the **circuit** from 1.3 above so that the multiplexer inputs are now the output from each of the following 4 functions: *a^b*, *a v b*, a ⊕ b, ¬a. Use two additional toggle switches to represent *a* and *b*.
b) Draw a **circuit diagram** represent the circuit in part a).
c) Model the behaviour of the circuit using a **truth table** to show the relationship between the selector combination and the selected logic function.

TOP

1.5 Not-a-Minority Circuit

The 'Not-a-Minority' circuit will output a logic 1 if at least half of the inputs are 1.

a) Model the desired behaviour of a 4 input Not-a-Minority circuit using a **truth table**.
b) Model the behaviour of the circuit mathematically by using the truth table to construct
   i. the **conjunctive normal form** of the Boolean expression for the circuit
   ii. the **disjunctive normal form** of the Boolean expression for the circuit

c) Examine the expressions obtained in b) above. Select one of these, and if possible, try to **reduce it by algebraic manipulation**. Why did you choose that particular expression?
d) Verify your reduced expression by simulation or by implementing the circuit on the BOX.
e) Construct the **Karnaugh map** for the Not-A-Minority function. From the map **determine an optimal expression**. You may have to try both the min and maxterm expressions.
f) **Implement** the optimal expression from e) above.

For a bonus point:

g) **Implement** the circuit for f) above using only NAND or NOR gates. You should determine if this is possible on the BOX. If not, then you may simulate the circuit.

TOP

Session 2 – Processor Components

2.1 2-level Multiplexer

a) Model the behaviour of an 8-input multiplexer using a **truth table**. Do so efficiently: is it possible to avoid writing down 2048 lines and yet provide full information?
b) Model the behaviour of an 8-input multiplexer using a **two-level Boolean expression**. You can derive this directly from the truth table.
c) Draw the **circuit diagram** for an 8-input multiplexer using individual logic gates.
d) A multiplexer with *k* selectors can be used to implement *any* circuit with 2^k inputs where the 2^k inputs are individually connected to either logic 1 or 0. Use the circuit from c) above to **implement an even parity bit generator** (the total number of bits=1 is an even number). Before you actually do this in your circuit, model the behaviour using a **truth table**.

2.2 Multi-level circuits with multiple outputs – binary numbers

a) **Design** (truth table or Karnaugh map and expressions) and **implement** a circuit with 3 inputs (a_1 , a_2 and a_3) and 2 outputs o_0 (representing 2^0) and o_1 (representing 2^1). The output must give the binary representation of the number of inputs that are 1 .
b) **Simplify the expressions** for o_0 and o_1 . Can you find a 2 -level implementation for o_0 with only 2 gates? It may help you to make a truth table for parts of the expression you will get for o_0 .
c) **Implement the circuit** for the expressions o_0 and o_1 on the BOX. If you cannot build this exact design on the BOX due to insufficient gates then you may simulate the circuit.

2.3 Carry look-ahead adders

A fully combinatorial adder for two n-bit binary numbers can be constructed as a cascade of n Full Adders. The worst-case propagation delay of this circuit is determined by the longest path through the circuit, which is usually the path from one of the inputs a_0 , b_0 , c_0 to output c_n. In this exercise we are studying such an adder, for 2 bits, as shown in Figure 2. In order to complete the following exercises you are advised to draw the circuit using individual logic gates.

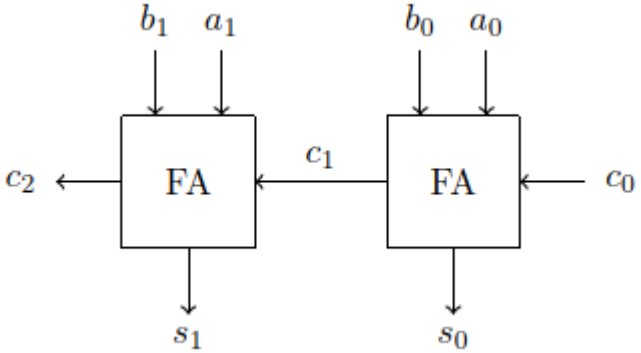


Figure 2 A 2-bit adder

a) Assuming that each of the two Full Adders in our 2-bit adder is constructed from two Half Adders and an OR-gate – as shown during the Lectures – , **what is the gate delay from input c0 to output c2** ? Count every AND-gate and OR-gate as 1 unit of delay but count every EXLUSIVE-OR-gate as 2 units of delay. **What is the gate delay from inputs a_0 and b_0 to output c_2** ? If we would cascade n of these 2-bit adders to form a 2∗n-bit adder, then **what would be the gate delay from a0 to c2∗n** ?

b) Assuming that now each of the two Full Adders in our 2-bit adder is constructed as a dedicated 2-level circuit, answer the **same questions as under (a)**. Remark: To answer this question you do not need to design the 2-level circuit version of the Full Adder.

c) In order to speed up the carry chain you should **design** and **simulate** a dedicated circuit for our 2-bit adder. This must be done in such a way that the gate delay from any of the inputs a_0 , b_0 , c_0 , a_1 , b_1 to output c_2 is (at most) 2 . As a result, a cascade of n of these 2-bit adders will exhibit a propagation delay from any input to c_{2*n} of at most $2 * n$ gates: quite an improvement!

Note that a design includes truth tables or Karnaugh maps, expressions and derivations as well as the final **circuit diagram**.

Hint: Your design must contain a dedicated 2-level circuit for c_2 ; the speed of the sum outputs s_1 and s_0 is less important: for these you may use the same EXLUSIVE-OR implementations as in the old Full Adder. Think about how you approach this problem: would you use a 32-line truth table? No! Can you construct a 5-input Karnaugh map? No! So, then what? Calculate! First, write down a minimal sum-of-products formula for c_1 , in terms of a_0 , b_0 , c_0 ; next, write down such a formula for c_2 , in terms of a_1 , b_1 , c_1 , and then . . . ?

For a bonus point :

d) **Implement** your answer to part c) on the BOX, and test it fully.

TOP

Session 3 – Sequential Circuits

In synchronous systems, i.e. systems in which (all) flip-flops are triggered by a central clock signal, it is considered bad practice to use this clock signal for any other purpose. Do not use the clock in guards of transitions of your finite automata. In particular, using the "CLK" signal in a guard is pointless, as the flip-flops are triggered by the upgoing edges of "CLK". The value of "CLK" just before these edges always will be 0 .

3.1 On-Off switch, with synchronous Reset

The following question is based on the introductory state machine example of an On-Off Switch in the textbook (section 3.5)

We will extend the On-Off Switch example with an additional input, named *r* . Whenever r = 0 the Switch behaves exactly as before. When, however, r = 1 the Switch will return to its initial state immediately, that is, at the very first rising clock transition after r has become 1 . (So, just as is the case with p , the responses of the circuit to r occur synchronously with the clock.) As long as r = 1 the Switch will remain in its initial state, so as long as r = 1 , the value of input p is irrelevant.

The following state assignment should be used;

State	*s*	*T*
A	0	0
B	0	1
C	1	0
D	1	1

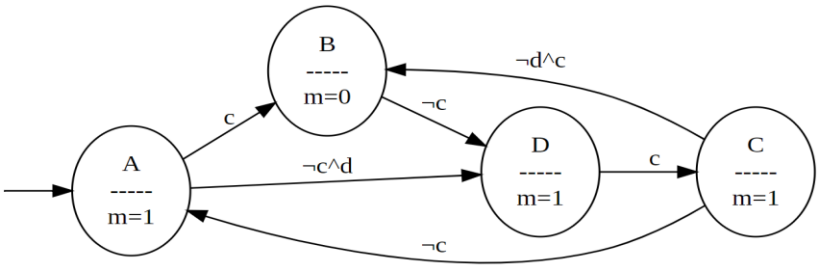
a) Design the sequential circuit to implement the switch with the new functionality. You will need to produce:
   - A revised **state machine diagram**
   - A revised **state transition table**
   - A revised **truth table** for the output x
   - Revised **Karnaugh diagrams** for each of these three values, plus the resulting Boolean expressions.
   - A **circuit diagram** using edge-triggered D-Flip-Flops for the state variables

b) **Implement** this circuit on the BOX. Test it with a clock frequency of 1/4 Hz : How responsive is this circuit to input changes? Test it with a clock frequency of 1 Hz : How responsive is the circuit now? Determine experimentally what is the smallest of the available – on the BOX – clock frequencies such that, to a human observer, the circuit appears to respond "immediately" to input changes.

TOP

3.2 An abstract state machine

Consider the following state machine for some arbitrary system. There are two input signals c and d. There is one output signal m. If there are inputs that are not explicitly drawn, then these will cause the state machine to stay in the same state. The states must have the following internal encoding: A: 00, B: 01, C:10, and D: 11.



a) Design the sequential circuit to implement the above system. You will need to produce:
   - A **state transition table**
   - A **truth table** for the output(s)
   - A **circuit diagram** using edge-triggered D-Flip-Flops for the state variables (indicate clearly how you derived the circuit diagram)

   Note: part a) above is an examination style question. The use of Karnaugh maps is *optional*.

b) **Implement** the circuit on the box.

General Purpose Registers

A general purpose register R_x, is an interconnected group of flip flops with a bit-width W_R denoting the number of flip flops. The register can execute several operations, for example;

Pre-, or Parallel, load	: R_x <= value (the value is sourced from a bus which may be connected to another register R_y or an external device)
Clear	: R_x <= 0
Increment (or count up)	: R_x <= R_x +1
Shift left k bits	: R_x <= R_x . 2^k (where k < W_R - $\lfloor \log_2 \rfloor$ - 1, otherwise 0)
Shift right k bits	: R_x <= $\lfloor R_x / 2^k \rfloor$ (where k < W_R - $\lfloor \log_2 \rfloor$ - 1, otherwise 0)
Rotate left	: R_x <= (R_x << 1) || (R_x >> (W_R -1)) (|| is logical OR)
Rotate right	: R_x <= (R_x >> 1) || (R_x << (W_R -1))

3.3 4-bit Register with Parallel Load and Rotate

Design a 4-bit register capable of rotating its contents to the right. This is similar to the shift register circuit shown in Fig. 3.8 in the text book, although now there is a wrap-around from D_0 (Least Significant Bit) to D_3 (Most Significant Bit), and an additional input, PRE. When PRE=1 this allows a 4-bit value (P_3, P_2, P_1, P_0)to be loaded in parallel to each of the 4 D-flip-flops. When PRE=0 the circuits rotates the D-flip-flop contents 1 bit to the right with each clock pulse.

TOP

a) Produce simplified **state diagram*S*** for such a register. Note that attempting to account for all combinations of P_3, P_2, P_1, P_0 when PRE=1 will result in a single *very* large state diagram. For now, just consider the rotate operation for different D_3-D_0 values; this will result in smaller diagrams.

b) Complete the full **state transition table**. Now you should account for the PRE input. The first line has been done for you.

Current State				Input					Next State			
D_3	D_2	D_1	D_0	PRE	P_3	P_2	P_1	P_0	$D3_{next}$	$D2_{next}$	$D1_{next}$	$D0_{next}$
0	0	0	0	0	x	x	x	x	0	0	0	0

Note that where PRE=1 the next state will be the value on inputs P_3 - P_0, though the actual value of these inputs is irrelevant.

c) Derive the **Boolean expressions** for the next-state values. Note that the rotate *functionality* of the register as described above is not the implementation logic (there is no logical OR of the register with itself).

d) Create the **circuit diagram**.

For a bonus point:

e) **Implement** the circuit on the box.

TOP