THE UNIVERSITY OF MELBOURNE

Department of Electrical and Electronic Engineering

ELEN90066 Embedded System Design

PRACTICE EXAM - SOLUTIONS

Reading/ printing time: 30 minutes, Writing time: 180 minutes

Scanning and submission time: 30 minutes

This examination paper has 8 pages

Authorised materials:

This is a Zoom-supervised, hand-written exam where only the following materials are permitted:

- Printed copy of this exam paper and/or an offline electronic PDF reader.
- Any material available in hardcopy or stored electronically on a device disconnected from communication networks.
- A4 paper, pens, pencils, ruler. Electronic devices may not be used for writing your answers.
- Any calculator model.

Instructions to students:

- This PRACTICE exam is provided purely as an example of the format and sorts of questions that will form part of the final exam for the subject and is for your own study purposes. It is not indicative of the actual content of the final exam.
- The questions carry weight in proportion to the marks stated for each question number. These marks total 100 marks.
- During Reading Time, you may print out a hardcopy of the exam. Alternatively, you may download the exam to an electronic PDF reading device, which must then be disconnected from the internet.
- During Writing Time you may only interact with the device running the Zoom session with supervisor permission. The printed test paper (or offline device containing the test paper) and any other working sheets must be visible to Zoom invigilators.
- Write your exam answers using pens/pencils and A4 paper. Start each question on a new page and write down the question numbers.
- During Scan/Upload Time, assemble your pages in question number order, and use a mobile phone or tablet to scan and combine them into a single PDF file. Check that all pages are included and clearly readable.
- Submit your PDF file to the Gradescope Assignment corresponding to this exam. Confirm with your Zoom supervisor that you have received confirmation of your submission.
- Collusion is **not allowed under any circumstances**. Collusion includes, but is not limited to, talking to, phoning, emailing, texting or using the internet to communicate with others.
- Plagiarism, through the use of sources without proper acknowledgement or referencing, **is not permitted** and can attract serious penalties. Plagiarism includes copying and pasting from the Internet without clear acknowledgement and paraphrasing or presenting someone else's work as your own.

Question 1 (14 marks)

The cliff-edge sensors on the Kobuki robot used in your project were a type of analog distance measuring sensor, using a beam of reflected infrared light to sense the distance between the sensor and a reflective target (i.e. the ground). The range to an object is proportional to the reciprocal of the sensor's output voltage.

An excerpt from the sensor's data sheet is given below

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Measuring distance range	ΔL	(Note 1)	2	-	15	cm
Output terminal voltage	Vo	L=15cm (Note 1)	0.25	0.4	0.55	V
Output voltage difference	ΔVo	Output change at L change (15cm → 2cm) (Note 1)	1.35	1.65	1.95	V
Average supply current	Icc	L=15cm (Note 1)	-	12	22	mA

※ L: Distance to reflective object

(a) **[2 marks]** What are the typical output voltages for the cliff-edge sensor at both ends of its stated measuring range?

The sensor's measurement range is between 2cm and 15cm. At the edge of its operating range, 15cm, the typical sensor output voltage is 0.4V. The typical output voltage difference when the range is 2cm is 1.65V.

Therefore, at 2cm, the typical output voltage is $0.4 + 1.65 = 2.05V$

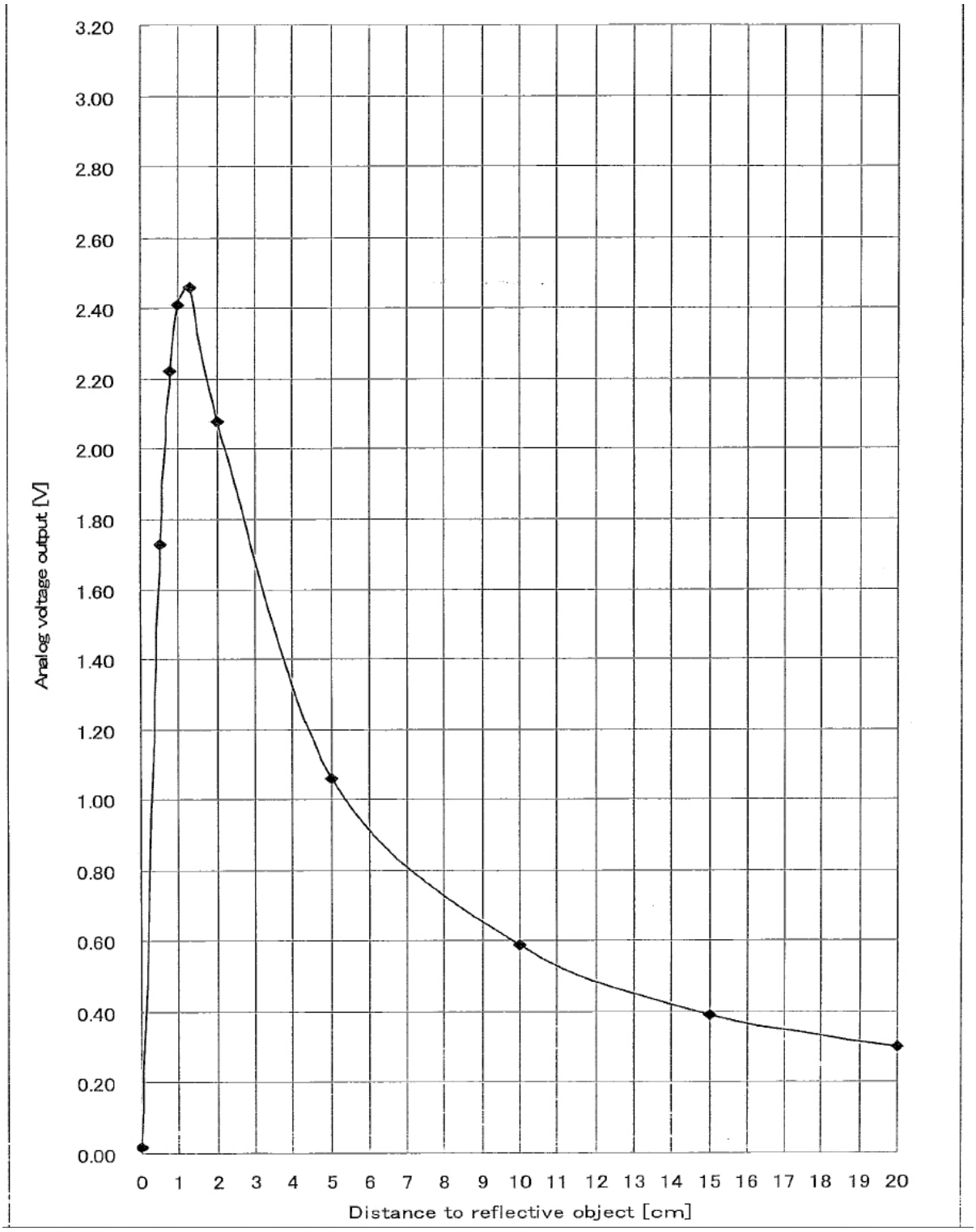
(b) **[2 marks]** Using ONLY the data from the table above and assuming an *affine* model for the sensor, is it possible to determine the *bias* and *sensitivity* parameters of the sensor? If yes, explain how to determine them. If no, explain why not.

Assuming an affine model for the sensor, $f(x) = ax + b$, given any two points we can determine the values of a, the sensitivity, and b, the bias.

Note that determining these parameters from the data sheet using the typical values may not give accurate results due to manufacturing differences in sensors. If the actual sensor parameters were closer to the min or max values, then the bias and sensitivity parameters could be quite different, and would cause inaccurate measurements to be performed.

Question 1 (continued)

(c) **[5 marks]** Some testing data for the sensor, using a particular surface, is given in the figure below



Question 1 (continued)

The equation mapping the sensor voltage V_O back to the physical range R (cm) is given by

$$R = K_s \left(\frac{1}{V_O} \right) + K_o$$

where K_s and K_o are parameters characteristic of the sensor.

Use the data in the graph to estimate K_s and K_o over the normal operating range of the sensor.

Reading values from the graph, we have the following

Distance (cm)	Voltage V_O	$1/V_O$
0	0.02	50
0.5	1.72	0.58
0.8	2.22	0.45
1	2.42	0.41
1.4	2.46	0.41
2	2.08	0.48
5	1.06	0.94
10	0.58	1.72
15	0.4	2.5
20	0.3	3.33

The approximate linear range of the sensor when considering the mapping $\frac{1}{V_O} \rightarrow R$ is between 2cm and 15cm. For simplicity, you could pick any two points in that range and calculate K_s and K_o from them.

A more thorough calculation would fit a linear line to multiple data points. Doing this yields $K_s = 6.43V \cdot cm$, $K_o = -1.07cm$

Question 1 (continued)

(d) Assume now that the output terminal voltage of the cliff-edge sensor, V_O, can range between 0V and 3.3V. The output signal from the cliff-edge sensor is converted to a 12-bit binary number via an Analog to Digital Converter for use in an obstacle avoidance algorithm by dividing this voltage range into equally spaced voltage steps.

(i) **[2 marks]** What is the precision of this sensor?

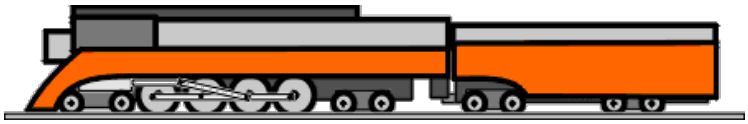
The precision of the sensor is $\frac{3.3}{2^{12}} = \frac{3.3}{4096} = 8.06 \times 10^{-4}$ V.

(ii) **[3 marks]** What is the dynamic range of the sensor, expressed in dB?

$$D_{dB} = 20 \log_{10} \left(\frac{H - L}{p} \right)$$
$$= 20 \log_{10} \left(\frac{3.3 - 0}{p} \right) = 72.25 dB$$

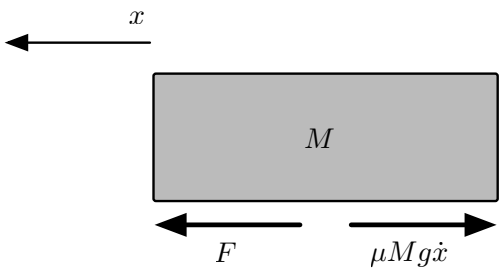
Question 2 (10 marks)

Consider a model of a train shown in the figure below.



Assuming that the train only travels in one dimension (along the track), we want to model the train so that it starts and comes to rest smoothly, and so that it can track a constant speed command with minimal error in steady state.

The train will be considered as one rigid mass, represented as M. The force F represents the force generated between the wheels of the engine and the track, while μ represents the coefficient of rolling friction. The free-body diagram is shown below.



From Newton's second law, the sum of the forces acting on a body is equal to the product of the mass of the body and its acceleration. In this case, the forces acting on the train in the horizontal direction are the rolling resistance, and the force generated at the wheel/track interface.

The rolling resistance forces are modelled as being linearly proportional to the product of the corresponding velocities and normal forces (which are equal to the weight forces).

Applying Newton's second law in the horizontal direction based on the above free-body diagram leads to the following governing equation for the train system.

$$F - \mu M g \dot{x} = M \ddot{x}$$

(a) **[3 marks]** Assuming that the train is initially at rest, rewrite the train system equation above as an *integral* equation for $x(t)$.

Write the equation for $\ddot{x}$ as

$$\ddot{x} = \frac{1}{M}\left(F - \mu M g \dot{x}\right)$$

so that we can obtain $x(t)$ by integrating the right-hand side twice

$$x(t) = \int \int \frac{1}{M}\left(F - \mu M g \dot{x}\right) dt d\tau$$

We could try to tidy this equation up a little but it will be easier for the next sub question to leave it in this form.

Question 2 (continued)

(b) [**7 marks**] Assuming that $x(t)$ is an output, construct an actor model (a block diagram) that models the train. You should use only primitive actors such as *integrators* and basic arithmetic actors such as *scale* and *adder*.

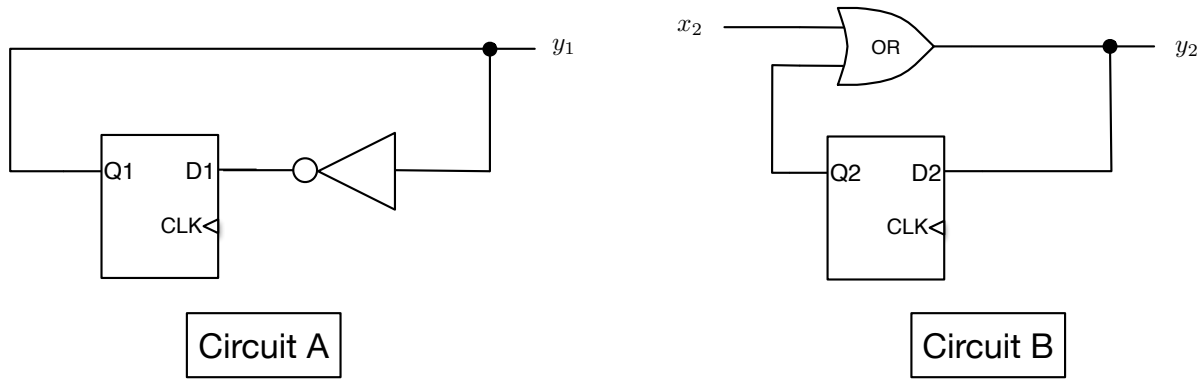
From the previous sub question,

$$x(t) = \int \int \frac{1}{M} \left(F - \mu M g \dot{x}\right) dt d\tau$$

The actors we will use for implementing this equation are 2 integrators, 2 scales and an adder.

Question 3 (24 marks)

(a) Consider the following two synchronous logic circuits below, Circuit A and Circuit B.

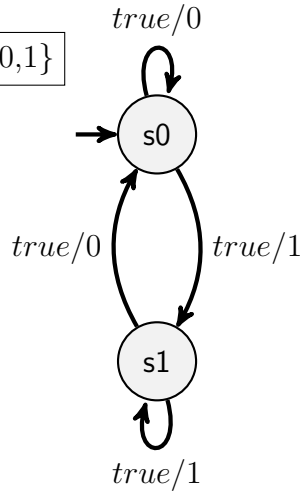


Note that the clock inputs on the D flip-flops are not explicitly shown, but the flip-flops react on the positive edge of the clock in the normal fashion (e.g. $Q <= D$).

(i) **[2 marks]** Construct a Finite State Machine diagram representing the behaviour of Circuit A.

Assume that the flip-flop initialises with $Q1 = 0$.

output: y_1 :{0,1}

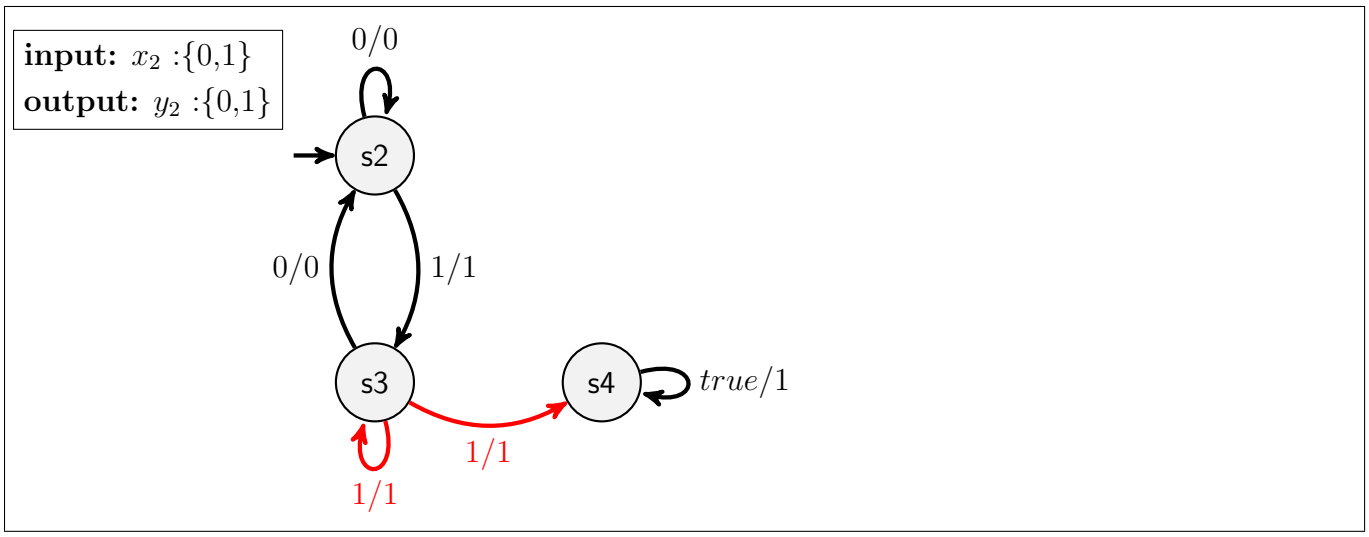


Note that the machine reacts on the CLK signal, which changes the state of the flip-flop (due to the presence of the NOT gate). The CLK signal is not specified explicitly as an input to the FSM in this case and we do not know *how* the flip-flop reacts to the CLK signal (e.g. negative or positive edge triggered), so the transitions only have a guard of *true* on them.

(ii) **[5 marks]** Construct a Finite State Machine diagram representing the behaviour of Circuit B.

As this circuit has an input x_2, in addition to a sequential element (the flip-flop), the FSM will have to model reactions to both a change in input and a clock edge on the CLK input.
Let's consider the machine reacting to a change in input first, assuming that the initial state of the flip-flop is $Q2 = 0$. If $x_2 = 1$, then irrespective of $Q2$, the output $y_2 = 1$. If $x_2 = 0$, then we need to know the state of the flip-flop in order to determine y_2. What happens if the flip-flop reacts to an edge on its CLK input? If $x_2 = 0$ and $y_2 = 0$, then that state is preserved. However, if, at that moment, $x_2 = 1$ and $y_2 = 1$, then $Q2 <= 1$ and that state is preserved. The trick is modelling the reaction of the circuit to the clock edge, depicted in the FSM diagram as the non-deterministic transition from s3 to s4. Once in s4 the machine cannot exit this state and stutters.

Question 3 (continued)



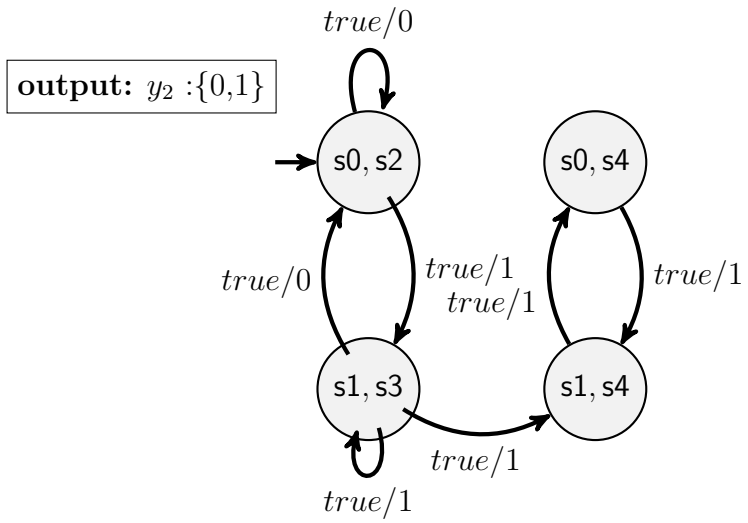
Question 3 (continued)

(iii) [**4 marks**] It is claimed that Circuit B could potentially 'get stuck' forever with $y_2 = 1$, independent of the input x_2.

Write this proposition in Linear Temporal Logic (LTL) and then either prove or disprove it.

FG$(y_2 = 1)$
This is true if $x_2 = 1$ when the flip-flop reacts to a clock edge, as the machine will transition to and remain in s3, where the output $y_2 = 1$.

(iv) [**10 marks**] Consider the *synchronous composition* of the cascaded connection of Circuits A and B – that is the output of Circuit A (y_1) is connected to the input of Circuit B (x_2). Construct a Finite State Machine diagram representing this composition.



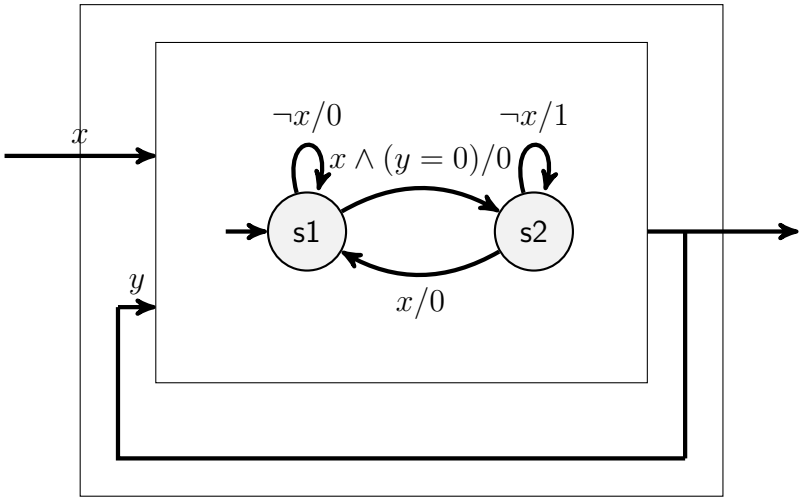
More space is provided for your answer on the following page.

Question 3 (continued)

(iv) (continued)

Question 3 (continued)

(b) Consider the following synchronous feedback composition shown below:



(i) **[3 marks]** Is the feedback composition *constructive*? Explain your answer.

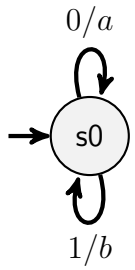
To check constructiveness, it suffices to do a must-may analysis of the feedback composition.
In s1, both possible transitions have the output y being 0, irrespective of the input. The guard for the transition to s2 is also formulated in terms of $y = 0$.

In s2, the output cannot be determined without knowing the input - if x is absent, then the output is 1, whereas if x is present then the output is 0. Therefore, we conclude that the machine is not constructive.

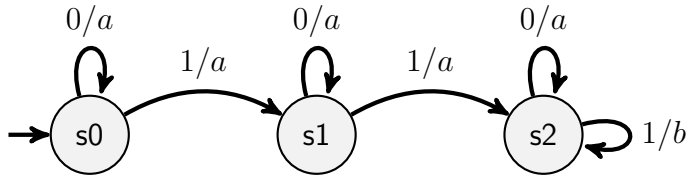
Question 4 (26 marks)

(a) Suppose that a finite state machine has a periodic sequence 0101010101 . . . as its only input. For each of the infinite sequences below, either construct a state machine (specify the state transition diagram and the initial state) that will yield the sequence as output, or argue that it is impossible to do so.

(i) [**2 marks**] abababababab...



(ii) [**5 marks**] aaaaababababababababab...

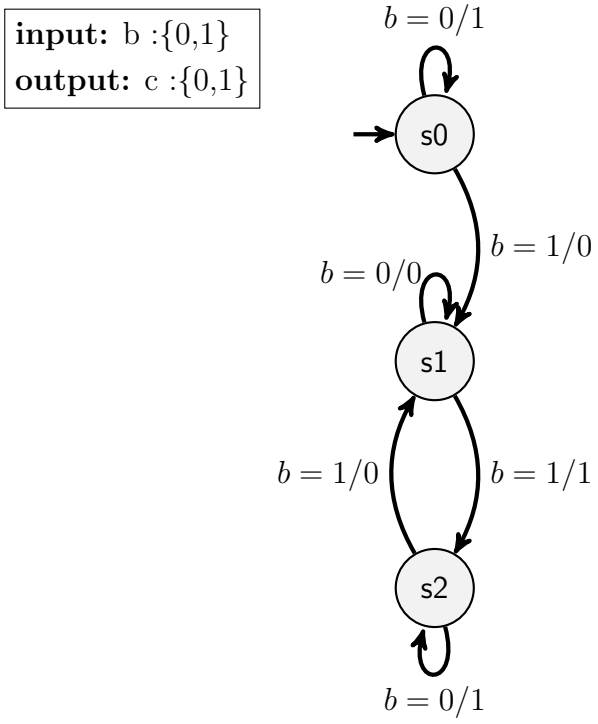


(iii) [**2 marks**] abaabbaaabbbaaaabbbb...

This is impossible as you need an infinite amount of states to remember where you are up to in the sequence (i.e how many a's and b's you need to output).

Question 4 (continued)

(b) The finite state machine shown below has the property that it outputs 1 if an input string of bits has *even parity*, that is the total number of 1s in the string is <u>even</u>.



(i) **[6 marks]** For each of the following LTL formulae, determine whether it is TRUE or FALSE. You **MUST** give an explanation for your answer. If FALSE, give a counterexample.

$\mathbf{G}(b = 1) \implies \mathbf{F}\mathsf{s2}$

This is TRUE. If $b = 1$ in every state, then the machine will eventually transition to state s2. It will not stay there, however.

Question 4 (continued)

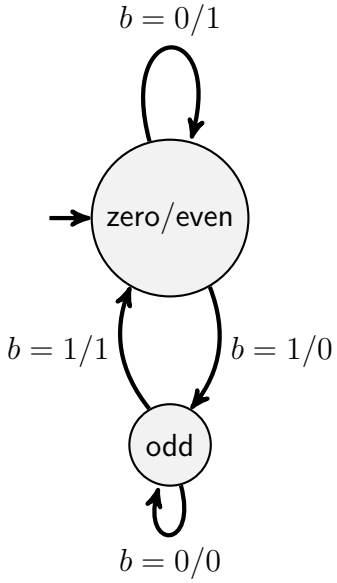
$\mathbf{GF}(b = 0) \implies \mathbf{FG}\mathsf{s0}$

This is FALSE. A counterexample is if $b = 1$ at any time after initialisation, then from the FSM diagram it is clear that the machine can never transition to (and remain in) s0.

(ii) [**8 marks**] Construct a **two-state** state machine that *simulates* this one and preserves its behaviour. Give the simulation relation.

Let M_1 denote the original three-state FSM. We are to construct a two-state machine M_2 that simulates M_1.

input: b :{0,1}
output: c :{0,1}



To test that M_2 simulates M_1, we must ensure that the machines are type equivalent and play the "matching" game : if M_1 always moves first, then M_2 can always match the move. Given the FSM diagram above, this is clearly the case and the simluation relation is

$$\{(\mathsf{s0}, \mathsf{zero/even}), (\mathsf{s1}, \mathsf{odd}), (\mathsf{s2}, \mathsf{zero/even})\}$$

Question 4 (continued)

(iii) **[3 marks]** Is your machine from part (ii) *bisimilar* to the original one? Explain your answer.

To test whether the machines are bisimilar (already knowing that they are type equivalent), we must play the "matching" game again, but this time allow the possibility of either machine moving first in any round of moves. You could do this by showing traces (as these machines are deterministic) to prove that they are bisimilar. The bisimilar relation is given by

$$S = \{(\mathsf{s0}, \mathsf{zero/even}), (\mathsf{s1}, \mathsf{odd}), (\mathsf{s2}, \mathsf{zero/even})\}$$

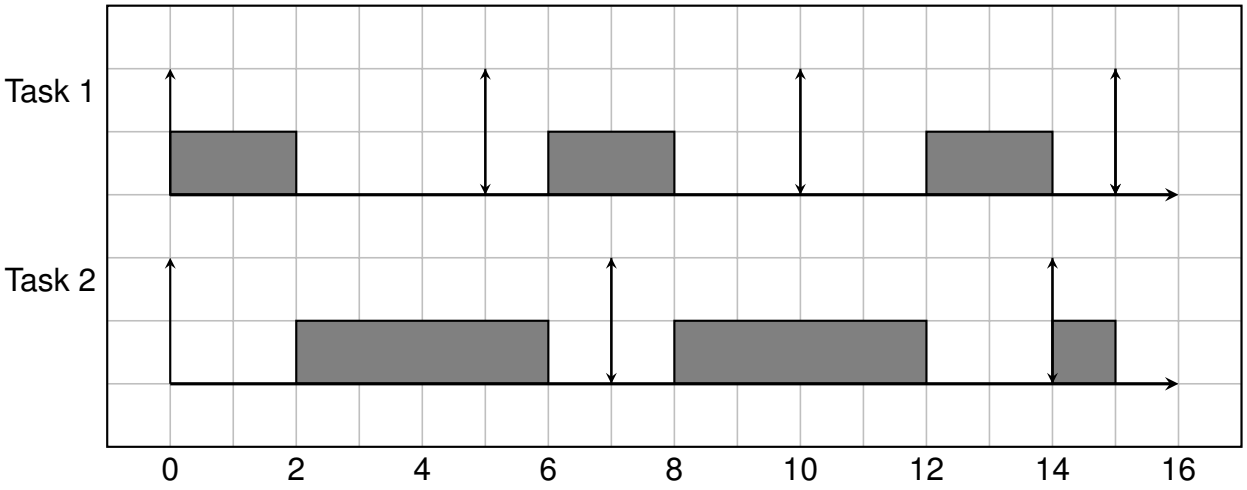
or, alternatively,

$$S' = \{(\mathsf{zero/even}, \mathsf{s0}), (\mathsf{odd}, \mathsf{s1}), (\mathsf{zero/even}, \mathsf{s2})\}$$

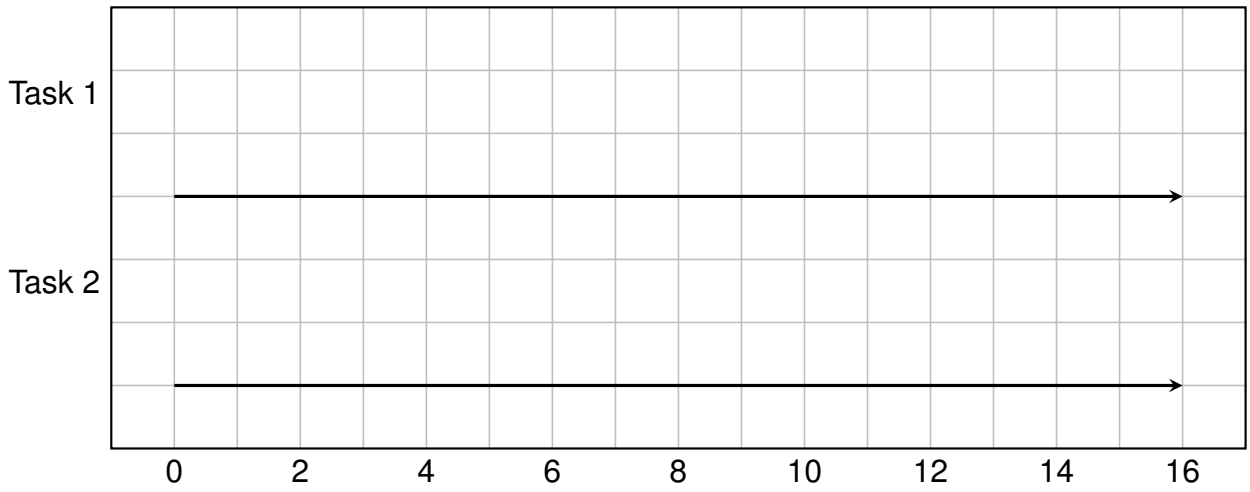
Question 5 (13 marks) Consider two tasks to be executed periodically on a single processor, where Task 1 has period $T_1 = 5$ and Task 2 has period $T_2 = 7$. Assume Task 1 has Worst Case Execution Time (WCET) $C_1 = 2$, and Task 2 has WCET $C_2 = 4$.

(a) **[6 marks]** Choose a scheduling algorithm covered in the lectures (RMS or EDF) that ensures that the two tasks have a *feasible schedule*. On the axes below, sketch the schedule for the tasks for 15 time units.

Final version



Rough working



A feasible schedule using EDF is given above.

Question 5 (continued)

(b) **[2 marks]** What is the relationship between threads and processes?

Processes are imperative programs with their own memory spaces. These programs cannot refer to each others' variables, and consequently they do not exhibit the same difficulties as threads. Communication between the programs must occur via mechanisms provided by the operating system, microkernel, or a library.

A processes is a container for threads. Threads are imperative programs that run concurrently and share a memory space. They can access each others' variables. A process may contain one or more threads, which share that memory space, all of the file descriptors and other attributes. The threads are the units of execution within the process, they posess a register set, stack, program counter, and scheduling attributes - per thread.

(c) **[2 mark]** Could threads within the same process share data with one another by passing pointers to objects on their stacks? Explain why / why not.

Yes. Since all the threads are executing in the same address space, they can pass pointers to one another and share data through these pointers. Such possible sharing includes objects allocated on the stack.

Question 5 (continued)

(d) **[3 mark]** Interrupt disabling and enabling is a common approach to implementing *mutual exclusion*. What are its advantages and disadvantages?

Advantage: It's easy and efficient on a uniprocessor.

Disadvantages:
(1) It doesn't work on multiprocessors as other processors can access resources in parallel - even if all interrupts are disabled on all processors.
(2) User-space programs shouldn't be allowed to do it as they can disable preemption and monopolise the CPU.
(3) Slows interrupt response time.

Question 6 (13 marks)

(a) **[2 marks]** Explain the concept of *pipelining* with respect to microprocessors in embedded systems. What benefits does it provide?

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the actions needed to execute an instruction. Each step operates in parallel with the other steps, although on a different instruction.

Pipelining yields a reduction in the average execution time per instruction. The reduction can be viewed as decreasing the number of clock cycles per instruction (CPI), as decreasing the clock cycle time, or as a combination. It has the additional advantage that it is not visible to the programmer.

(b) **[3 marks]** The security of embedded systems is notoriously behind that of desktop computing and consumer mobile devices. Give three reasons why this is the case.

- Many don't see embedded systems as potentially dangerous as networked desktop computers.
- Massive under-investment in security (viewed as an obstacle or as somebody else's problem).
- Consumers have to manage security themselves. In practice, it is managed by no one:
  - Updates rarely or never applied, if they're available at all.
  - Defaults throw users to the wolves. If they don't lock things down, it's their own fault.
- Applications completely unconstrained.
- Massive number of heterogeneous systems to protect.

Question 6 (continued)

(c) [**4 marks**] Name an additional sensor you would add to the Kobuki robot in order for it to perform better with either obstacle avoidance or hill climbing / descending? Explain how the sensor data would be used by your main navigation algorithm.

An example of a useful sensor would be an infra-red range finder (e.g. Workshop 1) in order to detect the distance to objects so that the Kobuki could avoid them before bumping into them. This could speed up the traversal of the obstacle course and also provide more detailed data of the course than the binary nature of the 3 bump sensors (either ON or OFF).

The sensor would need to be bench tested in order to measure its sensitivity and bias over a linear region of operation. Once this is done, it would need to be integrated into the Kobuki microcontroller via one of the Analog inputs and calibrated against several obstacles of differing sizes and shapes. The sensor may need to be placed low enough on the Kobuki to detect the objects used in the workshops, but sufficiently protected against bumps.

How the sensor data is actually used depends somewhat on the algorithm that you developed for the Kobuki. Assuming the sensor is mounted on the front, I would imagine that it would be continually polled while the robot is in forward motion. When the robot reacts to the crossing of a minimum distance threshold, it will need to rotate in order to avoid the obstacle. This avoidance could be done in an "open loop" fashion by simply rotating the robot by a fixed amount and continuing on forward, or by using a 'closed loop" feedback approach, where the robot rotates smoothly until the distance reported by the sensor increases above the minimum threshold.

Care must be taken upon approaching an incline that the robot does not misidentify it as an obstacle.

Question 6 (continued)

(d) [**4 marks**] Give an example of a *safety* requirement and a *liveness* property satisfied by your Kobuki robot algorithm.

This depends on the algorithm that you developed for the Kobuki. A safety property is one specifying that "nothing bad happens" during execution. More formally, a property p is a safety property if a system execution does not satisfy p if and only if there exists a finite-length prefix of the execution that cannot be extended to an infinite execution satisfying p.

An example of a safety property might be that "the robot does not perform the obstacle avoidance procedure while climbing a hill".

Similarly, a liveness property specifies that "something good will happen" during execution. More formally, p is a liveness property if every finite-length execution trace can be extended to an infinite execution that satisfies p. Liveness properties specify performance or progress requirements on a system.

An example of a liveness property might be that "when the robot bumps into an obstacle, it will eventually reverse away from it".

END OF PRACTICE EXAMINATION