The University of Melbourne
Department of Computing and Information Systems

Examination

Semester 1, 2014

SWEN90010 High Integrity Systems Engineering

Exam Duration: 2 hours

Total marks for this paper: 50

Reading Time: 15 minutes

This paper has 9 pages.

Authorised materials:
Students may not bring any material into the room.

Instructions to Invigilators:
Each student should initially receive a script with 9 pages.
Students may NOT keep the exam paper after the examination.

Instructions to Students:

- Attempt *all* questions.
- Answer questions only on the lined pages in a script book. The blank pages can be used for rough working.
- Start your answer to each question on a new page.
- Be sure to clearly number your answers.

The exam consists of 6 questions. The number of marks for each question is shown at the start of the question. Remember that rationale and justification is important in answering questions BUT also be as concise as possible with your answers. Point form is acceptable for more descriptive answers.

Part A – Safety engineering

Question 1 **[13 marks]**

In the projects in this subject, we have studied *implantable cardioverter-defibrillators* (ICDs). This question is based on the same domain.

For this, assume that we have the ICD as specified in assignment 1, *except* that there are three redundant heart-rate monitors operating in parallel, to mitigate the problem of faulty heart-rate monitors. In this system, the heart-rate monitors each provide a single reading to the ICD component, which uses approximate agreement majority voting to calculate a value for the heart rate.

Figure 1 show ICD system architecture. Note the three heart rate monitors.

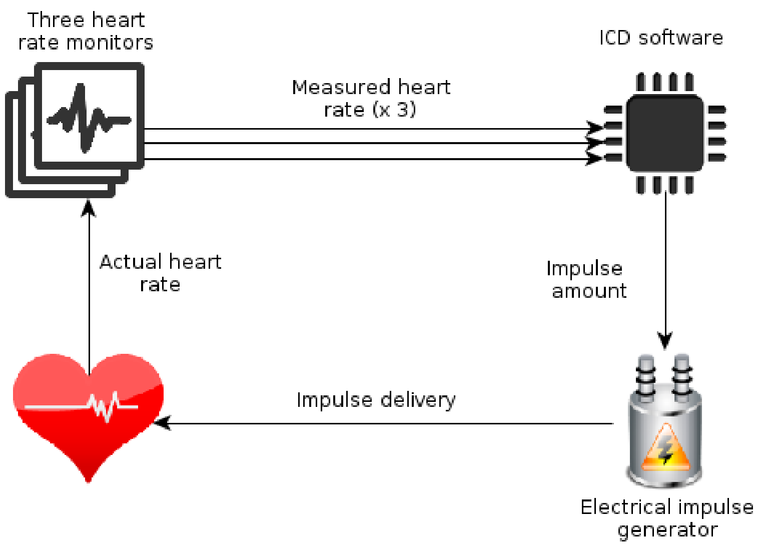

Three heart rate monitors
ICD software
Measured heart rate (x 3)
Actual heart rate
Impulse amount
Impulse delivery
Electrical impulse generator

Figure 1: A model of the ICD system architecture.

Safety conditions

The following safety condition must be met by the ICD system:

> If the patient's heart is beating in a regular manner, they shall receive no shocks from the impulse generator.

If this safety concern is violated, a hazard occurs.

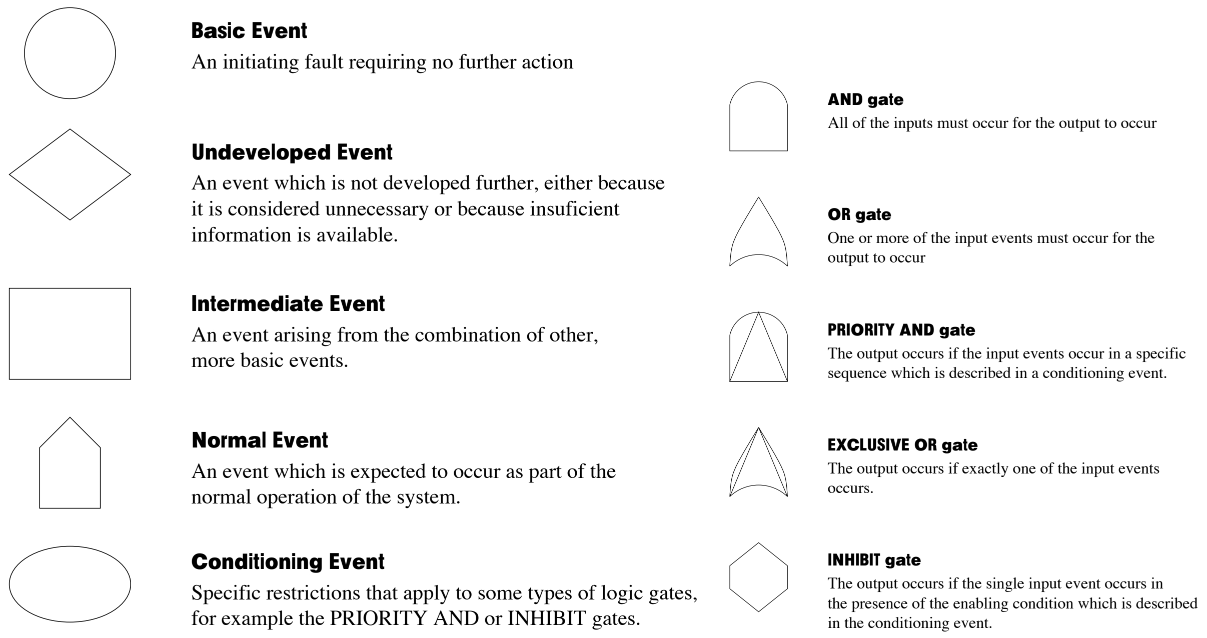
Perform a fault tree analysis on the ICD software component for this hazard for the case when the heart receives a 30 joule shock. Your fault tree should outlines how the failure can occur, caused by the ICD software only. Briefly justify each choice that you make in your analysis, stating why you believe each level is *necessary* or *sufficient*.

The top-level event for the fault tree is the case in which the heart is shocked at 30 joules while having a regular heartbeat.

For this question, assume that the ICD software has the following functions:

- Receive the three heart rate readings (assume that heart rate monitors are functioning correctly).
- Vote on the three readings using an approximate agreement majority voting algorithm.
- Calculate the heart status: fibrillation, tachycardia, or normal. Fibrillation is when the standard deviation of the five previous readings is above a certain threshold.
- Calculate the joules to deliver.
- Send the intended amount of joules to the impulse generator (assume that impulse generator is functioning correctly).

Figure 2 shows the syntax for fault trees, which you should use in your answer.



Basic Event
An initiating fault requiring no further action

Undeveloped Event
An event which is not developed further, either because it is considered unnecessary or because insuficient information is available.

Intermediate Event
An event arising from the combination of other, more basic events.

Normal Event
An event which is expected to occur as part of the normal operation of the system.

Conditioning Event
Specific restrictions that apply to some types of logic gates, for example the PRIORITY AND or INHIBIT gates.

AND gate
All of the inputs must occur for the output to occur

OR gate
One or more of the input events must occur for the output to occur

PRIORITY AND gate
The output occurs if the input events occur in a specific sequence which is described in a conditioning event.

EXCLUSIVE OR gate
The output occurs if exactly one of the input events occurs.

INHIBIT gate
The output occurs if the single input event occurs in the presence of the enabling condition which is described in the conditioning event.

Figure 2: Fault tree symbols

Part B – Model-based specification

Question 2 **[12 marks]**

Consider a messaging module of a system that maintains incoming and outgoing messaging queues for each process, and can send/receive messages from these queues.

The following Sum specification models the data types and state of the system. Total functions are used to model the message queue for each process.

```
Bit == {0,1};
Msg == seq Bit;

[ProcessID];

schema state is
dec
  incoming : ProcessID --> seq Msg;
  outgoing : ProcessID --> seq Msg
end state;
```

To get the incoming message queue for a process, `p`, we can use the expression `incoming(p)`.

For this question, the following Sum functions may be useful:

- `head(s)`: returns the first element in the sequence `s`. For example, `head(<3,2,1>)` is equal to 3.
- `tail(s)`: returns a sequence containing all but the first element in the sequence `s`. For example, `tail(<3,2,1>)` is equal to `<2,1>`.
- `restrict(s, e)`: where `s` is a sequence and `e`. Returns the sub-sequence of `s` containing only those elements in `e`. For example, `restrict(<4,3,2,1>, {5,3,1})` is equal to `<3,1>`.

(a) [**6 marks**] Model the following two operations:

- `Receive`, which takes as input, a message and a process ID, and adds the message to the incoming queue of the process ID.
- `Send`, which takes as input, a process ID, and outputs the first message in the outgoing queue of that process, and removes the message from the queue.

(b) [**6 marks**] Model the following operation:

- `ProcessIncoming`, which takes as input, a process ID, and then outputs the first message in the *incoming queue* of that process and removes that message from the queue, if and only if the party bit is correct. If not, an error is returned.
  This assumes that 1-bit parity coding is used for each message. That is, a single bit is added to added to the *start* of the message (recall that the message is a sequence of bits): 0 if there are an odd number of 1s, and 1 if there are an even number of 1s. To process a received message from the incoming queue, the operation shall first check that the parity bit is correct.

You do not need to get the syntax 100% to obtain full marks for this question.

Part C – Fault-tolerant design

Question 3 **[7 marks]**

N-version programming uses N different implementations to achieve fault tolerance. Consider two design scenarios: one in which N=2, and one in which N=3 (that is, a pair of redundant components, and a triple of redundant components).

Compare the 2-version programming and 3-version programming from the following points of view:

1. fault detection: that is, their ability to detect faults;
2. fault tolerance: that is, their ability to tolerate faults;
3. types of voting systems that could be used.

Question 4 **[6 marks]**

Interlaces parity-bit coding can tolerate *and detect* a 1-bit error in a string of bits by storing $N/2$ extra bits (where N is the length of the information bits). The table for determining which bit is in error for an 8-bit information string is below.

	Error in parity bit			
Bit error	**P3**	**P2**	**P1**	**P0**
w_0	×	×	×	×
w_1	×	×	×	
w_2	×	×		×
w_3	×		×	×
w_4	×	×		
w_5		×	×	
w_6			×	×
w_7		×		×
P_0				×
P_1			×	
P_2		×		
P_3	×			

Assume a system in which a parity bit is 0 if there are an odd number of 1s, and 1 if there are an even number of 1s.

If a process receives the following 12-bits, with the first 8 containing the information and the remaining 4 containing the parity bits:

11111000-0110

Is any of the bits in this string in error? If so, which one? (Assume that at most 1 bit can be corrupted). Show your working.

Part D – Correctness by construction

Question 5 **[1 mark]**

Who is the programming language *Ada* named after, and why is this person famous?

Question 6 **[11 marks]**

Figure 3 shows an Ada program for calculating the average of an array of integers, and then overwrites the first element of the array with the value of the average. The program includes a contract with the postcondition:

`Result = summation(1, List~'Length, List~)/List~'Length and List(1) = Result;`

where `summation(A,B,L)` = $\sum_{i=A}^{B}$ `L(`i`)`.

The program loops through all elements in the array, keeping a running tally in the variable `Sum`. The loop invariant has been given in the assertion. At the end of the loop, the total sum is divided by the list length to give the average, unless the list is empty, in which case the result is 0. Finally, it puts the average value in the first element of the list.

Using Hoare logic, show that this program either establishes its contract, or fails to establish its contract. The Hoare logic rules are given in Figure 4.

```
type FloatArray is array(<>) of Float;

procedure ListAverage(List : in FloatArray; Result : out Float)
--# derives Result from List & List from List;
--# post Result = summation(1, List~'Length, List~) / List~'Length
--#      and List(1) = Result;
is
  I, N : Integer;
  Sum : Float;
begin
  Sum := 0.0;
  I := 1;
  N := List'Length;

  while I /= N loop
    --# assert Sum = summation(1, I, List);
    I := I + 1;
    Sum := Sum + List(I);
  end loop;

  if N = 0 then
    Result := 0.0;
  else
    Result := Sum / N;
  end if;

  List(1) := Result;

end ListAverage;
```

where `summation(A,B,L)` = $\sum_{i=A}^{B}$ `L`(i).

Figure 3: An Ada program for calculating the average from an array of integers.

$\{P[E/x]\}\ x := E\ \{P\}$ (assignment axiom)

$\{P\}\ \mathbf{skip}\ \{P\}$ (empty statement axiom)

$$\frac{P' \to P,\ Q \to Q',\ \{P\}\ S\ \{Q\}}{\{P'\}\ S\ \{Q'\}}$$ (consequent rule)

$$\frac{\{P\}\ S_1\ \{R\},\quad \{R\}\ S_2\ \{Q\}}{\{P\}\ S_1; S_2\ \{Q\}}$$ (sequential composition rule)

$$\frac{\{P \land B\}\ S_1\ \{Q\},\quad \{P \land \neg B\}\ S_2\ \{Q\}}{\{P\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{endif}\ \{Q\}}$$ (conditional rule)

$$\frac{\{P \land B\}\ S\ \ \{P\}}{\{P\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{done}\ \{\neg B \land P\}}$$ (iteration rule)

$$\frac{\{P\}\ S\ \{Q\}}{\{P[E_1/v_1, E_n/v_n]\}\ p(E_1, \ldots, E_n)\ \{Q[E_1/v_1, \ldots, E_n/v_n]\}}$$ (procedure call rule)

$\{P[a\{NE \to E\}/a]\}\ a[NE] := E\ \{P\}$ (array assignment rule)

Figure 4: Rules for Hoare logic.

— End of exam —