Embedded Systems II
EEE3096/5S

FINAL EXAM

16 November 2021

2 hours

Examination Prepared by:
Simon Winberg and Jane Wyngaard

Last Modified: 30-Oct-2021

REGULATIONS

Dear Students,
Please note this is a **closed-book exam**. Provide your answers to the questions in the space provided. An additional page is provided after the last question page. You are recommended to detach the cheat sheets that are on the last pages of the exam paper. The exam is printed double sided. You are recommended to first scan through the questions quickly before starting, so that you can plan your strategy for answering the questions. If you are caught cheating, you will be referred to University Court for expulsion procedures. Make sure to **put** your **student name and student number** above. If you need any additional pages, are going to submit your exam pad provided as rough work, then please indicated at least your student number on the script together with the **course code EEE3096S or EEE3095S** and **a title ES2 Final Exam**.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO

Exam Structure
Marked out of 100 marks. 120 minutes.

NB

RULES

- **Answer all questions on the question paper in the space provided below the question**
- Make sure that you cross out material you do not want marked. Your first attempt at any question will be marked if two answers are found.
- Use the answer book to plan the facts for your written replies to questions, so that you produce carefully constructed responses.
- Answer all questions, and note that the time for each question relates to the marks allocated (which implies there are a few minutes to allow you to think and shuffle papers etc).

Structure of exam:

Multiple Choice, Short Answers, Design Questions to test your mettle

SECTION A:

MULTIPLE CHOICE AND SOME EASYPEASY QUESTIONS [5 x 4 = 20 marks]

Circle the letter a - d to select your choice of answer (select only one answer option for each)

Question A.1. [4 marks]

In CPU pipelined to execute in stages, various potential hazards must be handled by the hardware. Executing the following assembly instructions will cause which type of hazard.

In a standard ARM CPU pipeline, execution follows which of the following stages (select one option)

a) Decode, Execute, Memory Access, Fetch, Writeback

b) Memory Access, Decode, Execute, Fetch, Writeback

c) Memory Access, Fetch, Execute, Decode, Writeback

d) Fetch, Decode, Execute, Memory Access, Writeback

Question A.2. [4 marks]

There are various classifications for real-time systems. Which one of the following options best defines the characteristics of a periodic real-time system?

a) A system with a set of tasks that executes repeatedly at a specific period.

b) A system that meets (at least one) specific hard real-time deadline that repeats at a specific interval.

c) The term refers to a real-time system that uses polling (as opposed to e.g. interrupts).

d) It is a system for which any one of its operations has to complete within a specific time bound.

Question A.3. [4 marks]

Operating systems are responsible for and provide:

a) Process scheduling, and control of process access to hardware resources

b) Hardware management services such as mounting storage devices, controlling RAM access by different programs, and ensuring cache coherency is maintained.

c) Process scheduling, I/O, and instruction pipelining services

d) The system call memory table, boot loader storage location, and interrupt handling.

Question A.4. [4 marks]

Considering that the assembly function below is compatible with the C calling convention and compiler optimizations in use, what would be returned by func(133,1025) if called in a C function?

```
@ func(unsigned int, unsigned int):
        add     r0, r1, r0
        and     r0, r0, #127
        bx      lr
```

a) 0

b) 6

c) 8

d) 1158

Question A.5. [4 marks]

Consider that we want to count the number of bits that are clear (i.e., 0) in the input register *x*. The starting point to the module has been provided below. You need to select which option below is the right answer for which of these code options you can replace the '*put code here*' line to achieve the objective.

```
module numbits ( x, n );
  // define the ports
  input  [4:0] x;
  output [2:0] n;
  // define the registers
  reg n;

  // put code here!

endmodule
```

Select the right code option below:

a)

```
always @(x)
  begin
    n = (x&1) + (x&2) + (x&4) + (x&8) + (x&16);
  end
```

b)

```
always @(x)
  begin
    n = 5;
         if (x&1)  n = n - 1;
    else if (x&2)  n = n - 1;
    else if (x&4)  n = n - 1;
    else if (x&8)  n = n - 1;
    else if (x&16) n = n - 1;
  end
```

c)

```
always @(x)
  begin
    n = x[3]+x[2]+x[1]+x[0];
  end
```

d)

```
always @(x)
  begin
    n = x[0]^x[1]^x[2]^x[3];
  end
```

SECTION B:

SHORT ANSWERS – NOT QUITE SO EASY QUESTIONS [23 marks]

Question B1 [8 marks]

B1.1. There are many ARM processors that provide the facility of executing Thumb instructions. For example the ARM91TDMI is one such example. Briefly explain what a Thumb instruction is and how it may be of benefit when thinking of choosing a processor to use in an embedded system.

[3 marks]

B1.2. Name another processor design extension that ARM processors may support (the name of the core used in the above subquestion gives some hints but you need to describe the extension not just its letter).

[2 marks]

B1.3. Answer the following true/false questions by circling True/False below.

	Answer options:	
● The ARM processors can only run programs compiled in C	True	False
● RISC processors generally have less complex instructions that CISC processors	True	False
● When calling functions in C using the standard ABI for GCC you can only use a maximum of 7 parameters (as only 7 registers are permitted for function calls)	True	False

[3 marks]

Question B2 [7 marks]

B2.1. Why are IPCs necessary? Give 2 examples of IPCs and where they would be used.

[3 marks]

B2.2. Explain what is meant by the concepts of 'spatial locality' and 'temporal locality', emphasising how they differ (2 marks). Briefly explain why, if at all, these concepts lead to the usefulness of cache memory in computer system designs, instead of processors just using external memory that is cheaper and has lots of space available.

[4 marks]

Question B3 [8 marks]

Consider that an InfraRed Module (IRM) is hooked up to three GPIO pins on an ARM processor. The diagram below shows the block diagram of the IRM module.The module has a somewhat non-standard interface. It has the following digital signal lines and operation is explained below:

IR_DOTX	: input signal to IRM to request it to sent bit on DATX line over IR
IR_DORX	: input signal to IRM to request it to return latest received bit on DARX line
IR_DATX	: input signal to IRM to indicate data bit to send over IR
IR_DARX	: output signal from IRM to indicate latest data bit received over IR
IR_RDY	: output signal from IRM to say it is ready (bit sent or bit received)

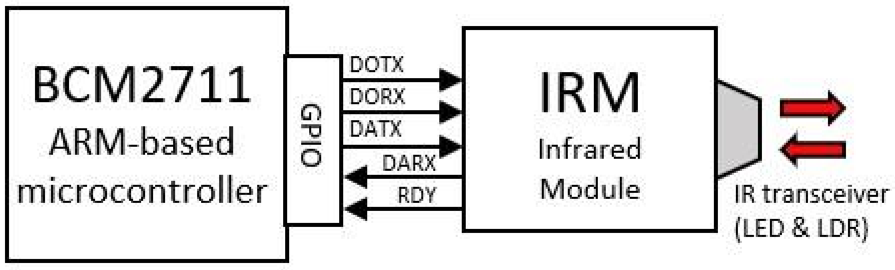


Figure showing ARM microprocessor to IRM connections

Note that IRM responds onto to positive edges on the IR_DOTX and IR_DORX lines; whenever it receives a positive edge on either of these lines it will immediately lower the IR_RDY line, and then only raise that line wither once it has sent a bit or receives a bit over IR. To send a bit, the microprocessor must wait for RDY to be high. Then, the DATX must first be set with the bit to send and then only can DOTX be raised to tell the module to transmit the bit on DATX over IR.
The IRM module is reset by setting both IR_DOTX and IR_DORX to high (it will only exit reset mode once either of these lines have been lowered, this is already done in the baseline code).

Have a look at the code below that configures the GPIO and resets the IRM. It coincidentally also indicates a piece of code that is missing, that you are asked to think about in questions B3.1 and B3.2 below.

```
/* Program to send the binary sequence 1010 over IR and then wait until a reply
   is sensed. */

#include <stdio.h>
#include <wiringPi.h>

/* device connected to pins 0 to 4 on BCM_GPIO 17 */
#define IR_DOTX    0 // request IRM to sent bit on DATX line over IR
#define IR_DORX    1 // request IRM to return latest received bit on DARX line
#define IR_DATX    2 // tell IRM what bit to send over IR
#define IR_DARX    3 // indicates what data bit IRM received over IR
#define IR_RDY     4 // if pin high then IRM is ready to receive command signals
                     // if pin low  then is is busy or waiting for tx/rx to complete
int main (void)
{
  printf ("Welcome to IRM tester\n") ;

  // set up GPIO to connect to IRM
  wiringPiSetup ();
  pinMode (IR_DOTX, OUTPUT);
  pinMode (IR_DORX, OUTPUT);
  pinMode (IR_DATX, OUTPUT);
  pinMode (IR_DARX, INPUT);
  pinMode (IR_RDY,  INPUT);
  // reset IRM by raising both DOTX and DORX
  digitalWrite (IR_DOTX, 1);
  digitalWrite (IR_DORX, 1);
  // lower both DOTX and DORX so that IRM is running
  digitalWrite (IR_DOTX, 0);
  digitalWrite (IR_DORX, 0);

  while (1)
  {
    // send the binary sequence 1010 over the IRM

    // >>>>>  Question B3.1 CODE TO BE ADDED  HERE!!!!  <<<<<

    // wait for 500ms
    delay (500) ;
    // now see if anything has come back, if so write SUCCESS otherwise try again …
    // BUT is this code good enough to do so? -- see Question B3.2
    if (digitalRead(IR_DARX)==1) {
        printf("SUCCESS!\n");
        return 0;
        }
    // nothing got returned so just delay a moment and try sending again
    delay (500) ;
  }
  return 0 ;
}
```

B3.1. The code provided above is not complete. Fill in the code, to be placed after the comment "Question B3.1 CODE TO BE ADDED ", to make the microprocessor tell the IRM module to send out over IR the sequence of bits 1, 0, 1, 0. (You need to provide some code in your solution, not necessarily all of it; your answer can incorporate some code and some explanation of how you would solve this). [4]

B3.2. Look at the code above, where it says "*BUT is this code good enough to do so?*". At that point in the program, the program is essentially waiting to see if the IRM module has received an incoming IR message and has a bit available to be received. Discuss whether or not the code provided will achieve this objective, if you think so briefly explain why, or if you don't think it would work explain how you might fix it. (Note: You do not have to provide code in your answer, a written explanation is sufficient). [3]

B3.3. .If you were not using WiringPi for using GPIO suggest an alternate approach or library that you could make use of instead. [1]

SECTION C:

TESTING YOUR METTLE QUESTIONS [34 marks]

The Following Scenario Relates to Questions C.1 - C.4 that follow

Take some time to read through this aspect briefly; the time for the exam is planned on you taking about 15 minutes reading and making sense of the scenario, and then answer the questions that follow.

Scenario: Configurable Wind-Powered Monitor (CWPM) IoT Platform

Consider that you are involved in developing an embedded platform system, called the Configurable Wind-Powered Monitor (CWPM) platform. The CWPM connects to a mini wind turbine that generates power, and charges a battery for times when the wind is not blowing. The CWPM design is shown in Fig. 1

As indicated, the rotor connects via the axle to the generator. Different size rotors can be used, from small to large, and the generator part has automatic gearing to account for changes in torque and speed depending on the rotor in use (but no need to worry about that at the moment). The CWPM comes standard with a low power ARM-based microcontroller. There are two SPI expansion slots to connect up daughter boards (e.g. for additional transducers or sampling sensors). There is also an 4-channel 10-bit ADC and a single channel 14-bit DAC. The ADC channels ch2 to ch4 each connect to an input socket that can link to an analog sensor (that is restricted to signals in the range 0V to 5V). The first ADC channel, ch1, is used to sample the battery charge level (which will also return an analogue voltage from 0V to 5V), to see how much battery power is left. The dynamo has a switch that triggers on each rotation (i.e. if it is generating some current), each rotation sends a pulse on the rota_IRQ line that invokes an IRQ on the ARM processor.

Baseline application code that has already been developed is shown on the next page, after which the questions for this section follow.

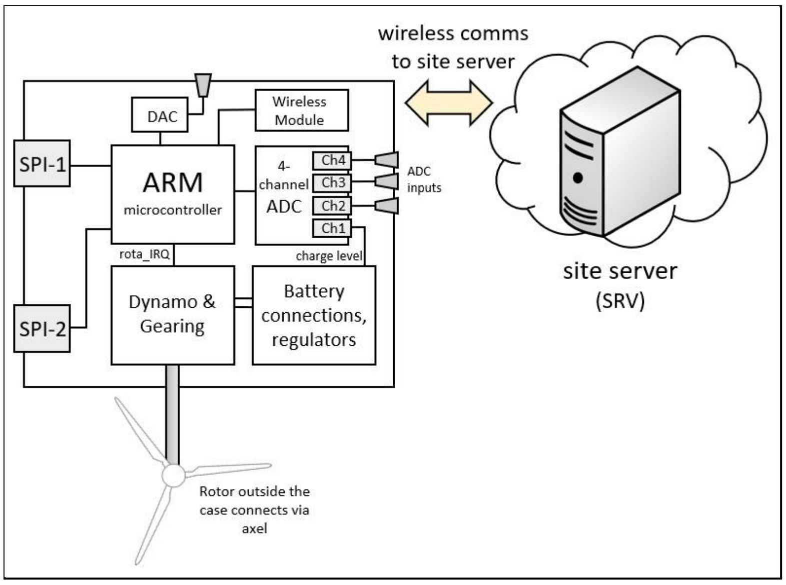


Fig. 1: Illustration of Configurable Wind-Power Monitor (CWPM) IoT Product

Data Capture and Buffering

A significant design decision for the CWPM was that it would gather input from its various sensors and store these in an array of structures for each sensor. These structs would essentially be used as a FIFO, new data will be inserted to the struct and data will be pulled out from the tail when ready to transmit. The struct would have the following form:

```
/** structure to store time information */
typedef struct {
  unsigned char day, month, year;  /* year is offset from 2000 */
  unsigned char hour, minute, second;
 } Time;
/** structure to store temperature sample */
typedef struct {
   Time  tstamp; /* date and time */
   float tempC;  /* temperature in degrees C */
   } TempSensor;

/* FIFO buffer used to store temperature data */
#define FIFO_LEN 0x100        /* specify number items in FIFO */
TempSensor temps[FIFO_LEN];   /* recorded temperature samples */
int head=0, tail=0, ntemps=0; /* FIFO head and tail, and count of samples */
```

Functions available for timing

The following functions have already been defined elsewhere in the program:

```
/* Externally defined functions */
extern unsigned long getticks_us(); /* time from system startup in microsec
*/
extern void delay_ms(int ms);  /* Cross-platform millisecond sleep function
*/
void getTime( Time* t ); /* incomplete function */
```

Sending data to the server

The sendall function below is invoked in report mode when the IoT device has gathered enough data. The implementation of this function is shown below (and relates to some questions that follow).

```
void sendall () {
  while (ntemps>0) {
    tail = (tail + 1)%LEN;
    ntemps--;
    wsend_data(MSG_TEMP,&temps[tail],sizeof(temps[tail]));
    }
}
```

Baseline main function

The following starting point main() function has been put together:

```
/** Main module: entry point to CWPM baseline */
void main () {
   float tempC; /* temporary variable to store temperature */
   printf("CWPM client started!\n");
   /* baseline just collects temperature and sends it to site server */
   while (1) {
     tempC = tempsens_read(); /* get the temperature in deg C */
     if (ntemps<FIFO_LEN) {
        temps[head].tempC = tempC;     /* add to FIFO */
        getTime(&temps[head].tstamp); /* get RTC */
        head = (head + 1)%FIFO_LEN;    /* move FIFO head on */
        ntemps++;                      /* another sample added */
     }
     /* wait 1s and then check if need to send all samples */
     delay_ms(1000);
    if (ntemps>204) sendall();
    }
}
```

Question C.1 [9 marks]

This question provides a reattempt at GA5-B.1 Embedded System design as a complex process.

The CWPM device involves an ARM microprocessor that connects to memory mapped peripherals. The plan is to provide C code, and GCC, for using the devices on the platform. Answer the following questions:

C.1.1. Consider that you have been tasked with deciding the Development Environment that engineers working on the embedded software of the CWPM will be utilizing[1]. Explain what is meant by the development environment (1 mark) and explain why a development environment for a custom embedded system may be different to that used for building application software for a standardized PC. (2 marks).

[3]

C.1.2. The CWPM has the *rota_IRQ* line hooked up to the IRQ line of the ARM processor. But there may likely be a need for other peripherals, such as those that connect to daughterboards, to provide interrupt signals. Discuss how you could work around this problem, why it could be possible, even if hardware changes to the platform are needed, so that the ARM can handle more than one interrupt.

[3]

C.1.3. In the design, there is a likely need to call manually written assembly code functions from C functions. Briefly explain how a C function could be connected to an assembly function. (While you can score full marks explaining ARM C ABI approach, you can still get marks for suggesting a potentially effective approach for making such a connection. Assembly code snippets can be added to aid explanation).

[3]

[1] You can assume all the engineers in the team have C programming and other ES development skills.

Question C.2 [8 marks]

This question provides a reattempt at GA5-A.3 analysis of embedded software designs and performance.

C.2.1. The **Data Capture and Buffering** section above indicates the struct and FIFO array used to store up to FIFO_LEN samples from the temperature sensor. To send the data from the CWPM to the site server, the *sendall* function shown above is used. The amount of data sent seems quite wasteful considering the information transferred. Such as, the temperature is sent as a float (even though it ranges from -49.9°C to 120.9°C accurate only to 1/10 of a degree). Suggest an approach to cut down on the amount of data being sent. (explain the amount of data if any your suggestion might save).

[3]

C.2.2. The CWPM has the *rota_IRQ* line that drives the IRQ request line on the ARM. The starting point, in C, for this routine is shown below.

What is needed is a means to track how fast the dynamo is turning, and we want to **determine an averaged speed of rotation over a period of 120s**. If the *rota_event* ISR function is called for each rotation of the dynamo, explain how you could go about working out the speed in RPM (i.e. rotations per minute) that the dynamo is turning. Keep in mind the functions, mentioned earlier, that are available: getticks_us, delay_ms and getTime. You want to try keeping the code in the ISR minimal. You can describe or annotate the code below regarding how you might change things in order to achieve this requirement.. [5]

```
/** CWPM interrupt service

The __irq declaration ensures that the return from the function actually implements a
return from interrupt changing back to the prior processor mode. You can access global
variables as normal in the interrupt routine. */

void __irq rota_event () {
   // currently does nothing just returns
}

/** Main function would be here, it is shown on page 9*/
void main ()
{
 ... existing code ...
}
```

Question C.3 [8 marks]

This question provides a reattempt at GA5-B.5, using (cross-)compiler for developing ES software.

C.3.1. GCC will be used for the C software needed for the CWPM. Installing IDEs, GCC, and related libraries would put much demand on the low-power processor and its limited memory. Accordingly, use of cross-compiler tools are considered. Briefly explain what advantages would be obtained from using a cross-compiler for this project. What sort of configuration might be needed, what would you need and do to get compiled code to run on the target (i.e. CWPM) platform?

[4]

C.3.2. There are various coding strategies that can improve the efficiency of code. One of these is the process or loop unrolling. Briefly explain what is meant by the process of loop unrolling and provide a brief example showing what sort of saving can be achieved (for your example you could formulate a rough estimate in terms of the saving, if any, provided).

[4]

Question C.4 [8 marks]

This question provides a reattempt at GA-B.6 standard embedded systems communication protocols

The CWPM is designed around supporting additional plug-on daughter boards that will be connected to via a SPI port. The SPI port interface has four signals: SCLK, MOSI, MISO, and ISS.

C.4.1. Let's consider that we want to send over SPI the byte 0x5B from the master to the daughterboard (an SPI slave). Assume lines are in idle state, that neither master or slaves are communicating.. For your answer, you can either show a diagram of how the signals may look or you can explain in text/code how the communication will work to send this byte to the slide (assume the slave successfully receives the data sent).

[5 marks]

C.4.2. The I2C protocol has a number of advantages, but also some disadvantages compared to SPI. Briefly discuss one advantage of I2C over SPI, and one advantage that of SPI has over I2C.

[3 marks]

SECTION D:

TESTING YOUR METTLE QUESTIONS [24 marks]

Question D.1 Finite State Machines [8 marks]

This question provides a reattempt at GA-B.2 Use of design diagrams, including FSMs

D.1.1 You're writing the code for the microprocessor used to control a new beans-to-cup coffee machine. Draw the FSM that would most efficiently implement the following system functionality.

Making coffee requires a user to select to make a cup of coffee. The machine will then grind the beans, press the ground beans into the steam cup, pressurise and heat the water, and then force the water through the grounds (percolate the coffee) into the cup, and then perform a self clean. This machine does a series of self checks whenever it's turned on and after it self cleans (checks water level, bean level, if it's grounds dump container is full or not, how long ago the last clean was done, and if there are ground beans in the steam cup). If any of the self checks indicate a user needs to do something (refill the water, refill the beans, empty the dump container), then it will indicate the required action in a touch screen and wait for the user to do the action, once done it will then continue to wait for the user to select to make coffee. The self check also looks at how recently it last did a clean and if more than 12hrs has past it will do a clean. At any point in time, if an error occurs it will do a self check and once the error is resolved it will return to waiting for a user to select to make coffee.

Do not show inputs and outputs on the FSM diagram. [4]

D.1.2 How do mealy and moore FSMs differ? (mention at least four points)

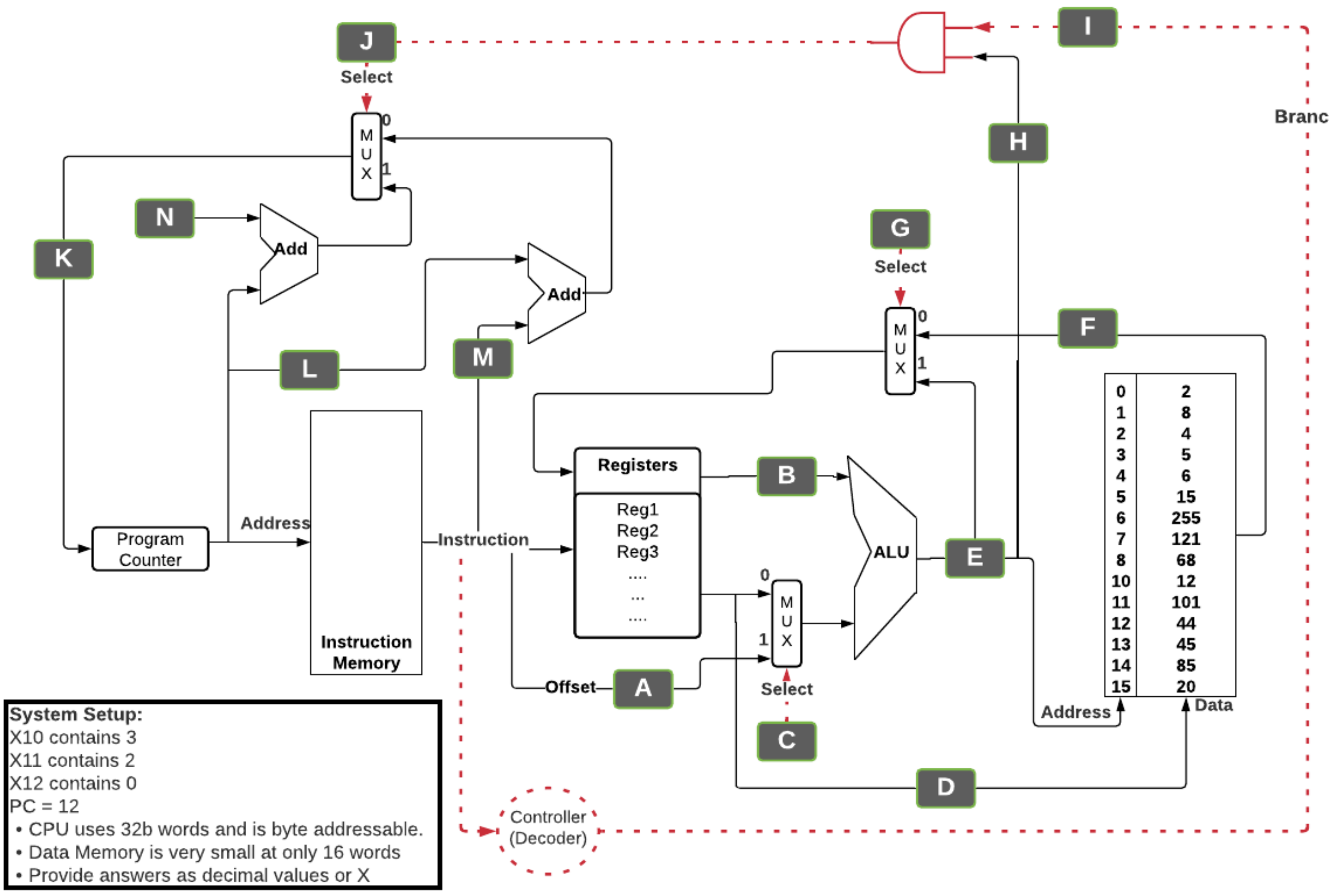
[4]

Question D.2 Computer and microprocessor architecture [8 marks]

This question provides a reattempt at GA-A.2 Understanding of computer and microprocessor architectures pertinent to use in low power, wireless, and or otherwise edge devices.

D.2.1 Fill in the lettered boxes (see next page) if the following instruction was to be executed:

LDR X10, [X11, 4]



0	2
1	8
2	4
3	5
4	6
5	15
6	255
7	121
8	68
10	12
11	101
12	44
13	45
14	85
15	20

System Setup:
X10 contains 3
X11 contains 2
X12 contains 0
PC = 12
- CPU uses 32b words and is byte addressable.
- Data Memory is very small at only 16 words
- Provide answers as decimal values or X

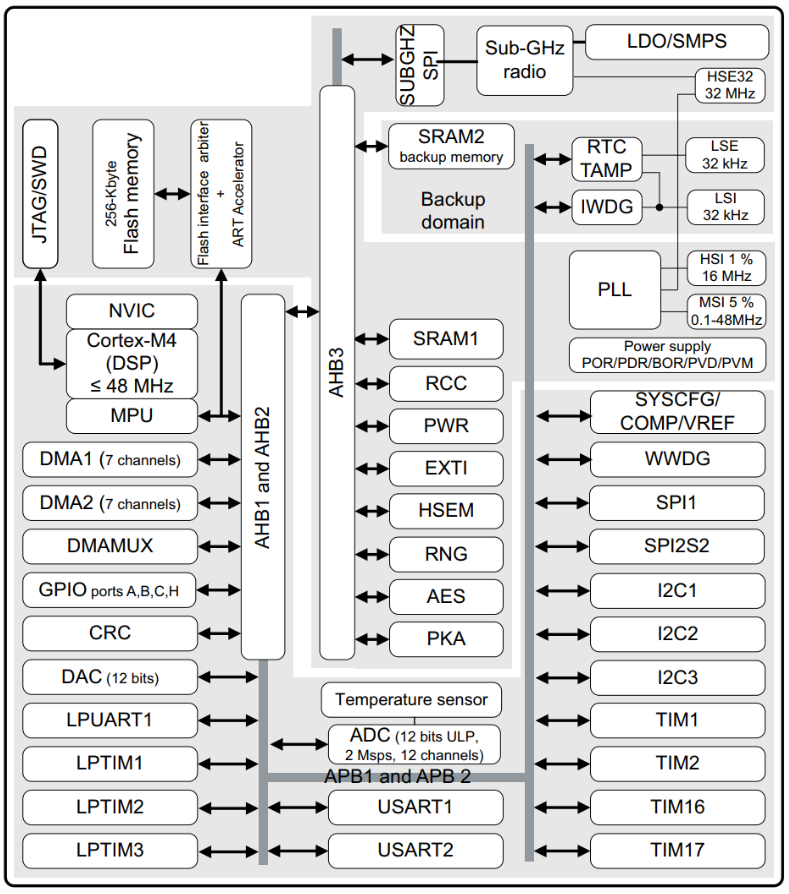
[4]

Fill in the values for the signals in the boxes below:

A		H	
B		I	
C		J	
D		K	
E		L	
F		M	
G		N	

D.2.2 The figure below shows the core functional components in a STM32WLE5. This microprocessor includes 2 CPU cores, a LoRa radio, and multiple interfaces.

Figure 1. STM32WLE5/E4xx block diagram



a) Give an example of of an application that this shown IC would be appropriate for and explain why:

[2]

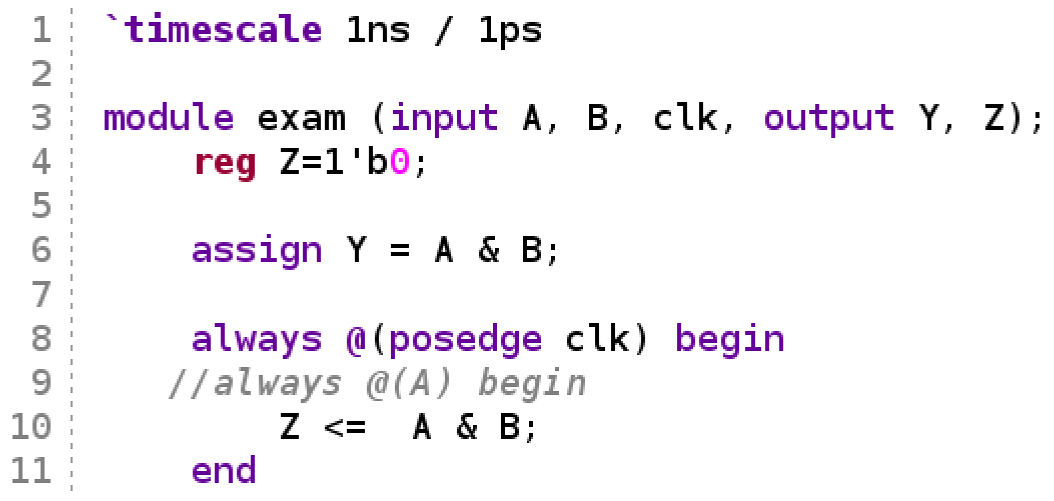
b) The IC diagram shows 2 SRAM units and a portion of flash memory. Why would both flash and SRAM memory be required? List 2 reasons.

[2]

Question D.3 Verilog [8 marks]

This question provides a reattempt at GA-A.1 Understanding of fundamental tools used in the design process for Embedded Systems

All questions refer to the following code snippet



```verilog
`timescale 1ns / 1ps

module exam (input A, B, clk, output Y, Z);
    reg Z=1'b0;

    assign Y = A & B;

    always @(posedge clk) begin
   //always @(A) begin
        Z <=  A & B;
    end
```

D.3.1 Given the above code as it is, list 3 differences between Y and Z?

[3]

D.3.2 If line 8 is commented out instead and line 9 uncommented what would change about the synthesized logic's behaviour?

[2]

D.3.3 Write the 1 line of verilog required to instantiate the 'exam module' as if in a testbench.

[2]

D.3.4 Write down the code that will correct the 1 coding error in the above module implementation.

[1]

END OF EXAMINATION

Appendix A: Verilog Cheat sheet

Numbers and constants

Example: 4-bit constant 10 in binary, hex and in decimal: 4’b1010 == 4’ha -- 4’d10

(numbers are unsigned by default)

Concatenation of bits using {}

4’b1011 == {2’b10 , 2’b11}

Constants are declared using parameter:

parameter myparam = 51

Operators

Arithmetic: and (+), subtract (-), multiply (*), divide (/) and modulus (%) all provided.

Shift: left (<<), shift right (>>)

Relational ops: equal (==), not-equal (!=), less-than (<), less-than or equal (<=), greater-than (>), greater-than or equal (>=).

Bitwise ops: and (&), or (|), xor (ˆ), not (˜)

Logical operators: and (&&) or (||) not (!) note that these work as in C, e.g. (2 && 1) == 1

Bit reduction operators: [n] n=bit to extract

Conditional operator: ? to multiplex result

Example: (a==1)? funcif1 : funcif0

The above is equivalent to:

```
 ((a==1) && funcif1)
 || ((a!=1) && funcif0)
```

Registers and wires

Declaring a 4 bit wire with index starting at 0:

wire [3:0] w;

Declaring an 8 bit register:

reg [7:0] r;

Declaring a 32 element memory 8 bits wide:

reg [7:0] mem [0:31]

Bit extract example:

r[5:2]   returns 4 bits between pos 2 to 5 inclusive

Assignment

Assignment to wires uses the assign primitive outside an always block, e.g.:assign mywire = a & b

Registers are assigned to inside an always block which specifies where the clock comes from, e.g.:

```
always@(posedge myclock)
  cnt = cnt + 1;
```

Blocking vs. unblocking assignment <= vs. =

The <= assignment operator is non-blocking (i.e. if use in an always@(posedge) it will be performed on every positive edge. If you have many non-blocking assignments they will all be updated in parallel. The <= operator must be used inside an always block – you can’t use it in an assign statement.

The blocking assignment operator = can be used in either an assign block or an always block. But it causes assignments to be performed in sequential order. This tends to result in slower circuits, so avoid using it (especially for synthesized circuits) unless you have to.

Case and if statements

Case and if statements are used inside an always block to conditionally update state. e.g.:

```
always @(posedge clock)
  if (add1 && add2) r <= r+3;
  else if (add2) r <= r+2;
  else if(add1) r <= r+1;
```

Note that we don’t need to specify what happens when add1 and add2 are both false since the default behavior is that r will not be updated. Equivalent function using a case statement:

```
always @(posedge clock)
  case({add2,add1})
  2’b11  : r <= r+3;
  2’b10  : r <= r+2;
  2’b01  : r <= r+1;
  default: r <= r;
endcase
```

Module declarations

Modules pass inputs, outputs as wires by default.

```
module ModName (
  output reg [3:0] result,  // register output
  input [1:0] bitsin,  input clk, inout bidirectnl  );
 … code …
endmodule
```

Verilog Simulation / ISIM commands

```
$display ("a string to display");
$monitor ("like printf. Vals: %d %b", decv,bitv);
#100   // wait 100ns or simulation moments
$finish  // end simulation
```

Appendix B: ARM Assembly Language Cheatsheet

Memory access instructions

LDR Rd, [Rn]	**; load 32-bit number at [Rn] to Rd**
LDR Rd, [Rn,#off]	**; load 32-bit number at [Rn+off] to Rd**
STR Rt, [Rn]	**; store 32-bit Rt to [Rn]**
STR Rt, [Rn,#off]	**; store 32-bit Rt to [Rn+off]**
PUSH {Rt}	**; push 32-bit Rt onto stack**
POP {Rd}	**; pop 32-bit number from stack into Rd**
MOV{S} Rd, <op2>	**; set Rd equal to op2**
MOV Rd, #im16	**; set Rd equal to im16, im16 is 0 to 65535**
MVN{S} Rd, <op2>	**; set Rd equal to -op2**

Branch instructions

B label	**; branch to label always**
BEQ label	**; branch if Z == 1 Equal**
BNE label	**; branch if Z == 0 Not equal**
BCS label	**; branch if C == 1 Higher or same, unsigned ≥**
BHS label	**; branch if C == 1 Higher or same, unsigned ≥**
BCC label	**; branch if C == 0 Lower, unsigned <**
BLO label	**; branch if C == 0 Lower, unsigned <**
BMI label	**; branch if N == 1 Negative**
BPL label	**; branch if N == 0 Positive or zero**
BVS label	**; branch if V == 1 Overflow**
BVC label	**; branch if V == 0 No overflow**
BHI label	**; branch if C==1 and Z==0 Higher, unsigned >**
BLS label	**; branch if C==0 or Z==1 Lower or same, unsigned ≤**
BGE label	**; branch if N == V Greater than or equal, signed ≥**
BLT label	**; branch if N != V Less than, signed <**
BGT label	**; branch if Z==0 and N==V Greater than, signed >**
BLE label	**; branch if Z==1 or N!=V Less than or equal, signed ≤**
BX Rm	**; branch indirect to location specified by Rm**
BL label	**; branch to subroutine at label**
BLX Rm	**; branch to subroutine indirect specified by Rm**

Logical instructions

AND{S} {Rd,} Rn, <op2>	**; Rd=Rn&op2 (op2 is 32 bits)**
ORR{S} {Rd,} Rn, <op2>	**; Rd=Rn|op2 (op2 is 32 bits)**
EOR{S} {Rd,} Rn, <op2>	**; Rd=Rn^op2 (op2 is 32 bits)**
BIC{S} {Rd,} Rn, <op2>	**; Rd=Rn&(~op2) (op2 32 bits)**
ORN{S} {Rd,} Rn, <op2>	**; Rd=Rn|(~op2) (op2 32 bits)**
LSR{S} Rd, Rm, Rs	**; logical shift right Rd=Rm>>Rs (unsigned)**
LSR{S} Rd, Rm, #n	**; logical shift right Rd=Rm>>n (unsigned)**
ASR{S} Rd, Rm, Rs	**; arithmetic shift right Rd=Rm>>Rs (signed)**
ASR{S} Rd, Rm, #n	**; arithmetic shift right Rd=Rm>>n (signed)**
LSL{S} Rd, Rm, Rs	**; shift left Rd=Rm<<Rs (signed, unsigned)**
LSL{S} Rd, Rm, #n	**; shift left Rd=Rm<<n (signed, unsigned)**

Arithmetic instructions

ADD{S} {Rd,} Rn, <op2>	**; Rd = Rn + op2**
ADD{S} {Rd,} Rn, #im12	**; Rd = Rn + im12, im12 is 0 to 4095**
SUB{S} {Rd,} Rn, <op2>	**; Rd = Rn - op2**
SUB{S} {Rd,} Rn, #im12	**; Rd = Rn - im12, im12 is 0 to 4095**
RSB{S} {Rd,} Rn, <op2>	**; Rd = op2 - Rn**
RSB{S} {Rd,} Rn, #im12	**; Rd = im12 – Rn**
CMP Rn, <op2>	**; Rn – op2 sets the NZVC bits**
MUL{S} {Rd,} Rn, Rm	**; Rd = Rn * Rm signed or unsigned**

Notes

Ra Rd Rm Rn Rt represent 32-bit registers
value any 32-bit value: signed, unsigned, or address
{S} if S is present, instruction will set condition codes
#im12 any value from 0 to 4095
#im16 any value from 0 to 65535
{Rd,} if Rd is present Rd is destination, otherwise Rn
#n any value from 0 to 31
#off any value from -255 to 4095
label any address within the ROM of the microcontroller
op2 the value generated by <op2>