EECS 373 Midterm 1
Winter 2023

9 February 2023

No calculators, reference material, internet, or communicating with others about the exam (except course staff).

Name

UM Uniqname

Sign below to acknowledge the Engineering Honor Code: "I have neither given nor received aid on this examination, nor have I concealed a violation of the Honor Code."

Signature

The number of points per problem are not well correlated to the time required. This is intentional, as we want students with good basic knowledge to get reasonably high scores, but for students with deeper understanding to receive higher scores.

1 Short questions (12 points)

Embedded system definition and market Technology trends Embedded applications

1. Why is it common for an embedded system development team to be smaller than a team developing a general-purpose microprocessor? Select one. (3 points)
   - ○ The entire embedded systems market is smaller than the general-purpose computing market, so the design teams must be smaller.
   - ○ Embedded systems typically have fewer and looser design constraints than general-purpose computing systems, reducing the number of designers needed.
   - ○ To be general-purpose, there must be numerous variants of the microprocessor design that are dynamically selected at run time. Each variant must be designed, requiring a large design team.
   - ○ ✓ The embedded systems market is fragmented, so a particular embedded design may have a smaller market than a single microprocessor design, thus constraining the total pay justified for an engineering team.
   - ○ Embedded systems have real-time constraints, which simplifies their design process.
2. The following list of common requirements for embedded systems was presented in the first lecture. Indicate one that is equally applicable to general-purpose computers. (3 points)
   - ○ Timely (hard real-time).
   - ○ Wireless.
   - ○ Reliable.
   - ○ First time correct.
   - ○ Rapidly implemented.
   - ○ ✓**(any of these O.K.)** Low price.
   - ○ ✓**(any of these O.K.)** High performance.
   - ○ Low power.
   - ○ Embodying deep domain knowledge.
   - ○ ✓**(any of these O.K.)** Beautiful.
3. In what context would you most likely find an LPWAN used? Select one. (3 points)
   - ○ Wireless video streaming security application.
   - ○ ✓ Low data rate agricultural sensor network spanning several acres.
   - ○ Sensor network within an automobile.
   - ○ **This one received partial credit.** Infrastructure-powered home automation network.
   - ○ Wearable device communicating with a smartphone carried by the user.
4. Is it possible for peripherals on an open-collector bus that produce high outputs simultaneously (i.e., at incorrect times) but are otherwise correctly designed to produce a short-circuit? (3 points)
   - ○ Yes.
   - ○ ✓ No.

2 Debugging and Aspects of ANSI C related to Embedded Systems (12 points)

1. When debugging a hardware-software system, which of the following options is generally more effective? (3 points)
   - ○ Focusing on the the debugging style with which you are most proficient, e.g., gathering information or reasoning about potential causes.
   - ○ ✓ Periodically switching between gathering information and reasoning about potential causes.

2. When debugging a hardware-software system, if you have no reason to believe that any of the following potential problems are more likely to be the cause of a persistant error than the others, which one should you test first. (3 points)
   - ○ A coding error in a machine learning algorithm running on your microcontroller.
   - ○ A defect in the DRAM memory chip being used to store data for the machine learning algorithm.
   - ○ ✓**Either is fine.** A faulty DC-DC converter providing your system with power and ground.
   - ○ A faulty microphone being used as input to the microcontroller.
   - ○ ✓**Either is fine.** A nearby wireless transmitter that may be a source of interference with the embedded system's communication signals.

3. Indicate all the lines that very likely hold bugs in the following C program. Presume that the addresses indicated in lines 1 and 2 are valid MMIO addresses. (6 points)

   ○ 1 ○ ✓ 2 ○ 3 ○ ✓ 4 ○ 5 ○ ✓ 6 ○ 7 ○ 8 ○ 9 ○ 10

```
   typedef enum {
     RETURN_TO_DEFAULT, LEFT, RIGHT, HALT, SAFE_SHUTDOWN
   } gimbal_command_t;

   #define GIMBAL_MAX_ROTATION 5326

1  volatile int *gimbal_mmio_address = (volatile int *)(0x123450);

2  const int *gimbal_mmio_rotation = (const int *)(0x123460);

3  void write_gimbal(gimbal_command_t command) {
4    gimbal_mmio_address = command;
   }

5  double gimbal_relative_orientation(void) {
6    return *gimbal_mmio_rotation / GIMBAL_MAX_ROTATION;
   }

   int main(void) {
7    write_gimbal(RETURN_TO_DEFAULT);
8    if (gimbal_relative_orientation() > 0.5) {
9      write_gimbal(SAFE_SHUTDOWN);
     } else {
10     write_gimbal(HALT);
     }
     return 0;
   }
```

3 ARM assembly (12 points)

Write an ARM assembly language procedure that implements the C function "move_alnum" in an ABI compliant manner. Clearly comment your code. Label which registers each value represents. Code comments enable partial credit. The function "isalnum" is an ABI compliant function with the following prototype.

```
int isalnum (char c);
uint32_t move_alnum(char * input,
uint32_t input_size) {
  char * output = input;
  uint32_t output_size = 0;
  int i = 0;
  while (i != input_size) {
    char c = *(input + i);
// Continue reading in next column.
```

```
      int isalnum = isalnum(c);
      if (isalnum != 0) {
        *(output + output_size) = c;
        ++output_size;
      }
      ++i;
    }
    return output_size;
  }
```

Provide your answer here. Although you are not required to use the following assembly instructions, some might be useful in your answer: ADD, B, BEQ, BL, CMP, LDRB, MOV, POP, PUSH, and STRB.

```
move_alnum:
  PUSH {R4-R8, LR}
  MOV R4, R0 // input
  MOV R5, R0 // char* output = input;
  MOV R6, R1 // input_size
  MOV R7, #0 // output_size = 0
  MOV R8, #0 // i = 0
while_loop:
  CMP R8, R6 // if (i == input_size)
  BEQ result
  LDRB R0, [R4, R8] // char c = *(input + i)
  PUSH {R0} // put c on stack.
  BL isalnum
  POP {R1} // bring c back from stack.
  CMP R0, #0 // if (isalnum == 0)
  BEQ else
if:
  STRB R1, [R5, R7] // *(output + output_size) = c
  ADD R7, R7, #1 // ++output_size
else:
  ADD R8, R8, #1 // ++i
  B while_loop
result:
  MOV R0, R7 // return output_size
  POP {R4-R8, PC}
```

4 Building (10 points)

Based on the following nm symbol listing, indicate all of the following files that may be simultaneously linked with main.o without linking errors. This question is not asking for the minimal set of files that can be linked to main.o without errors; it's asking for the maximal set.

- ◯ ✓ a.o
- ◯ ✓ b.o

◯ ✓ c.o

◯ ✓ d.o

◯ e.o

```
a.o:
0000000000000000 T func_a

b.o:
0000000000000000 T _Z6func_ai

c.o:
0000000000000000 t func_a

d.o:
0000000000000000 T func_b

e.o:
0000000000000000 T main

main.o:
                 U func_a
0000000000000000 T main
```

5 Interrupts (15 points)

1. Fill in the circle marking the method you should use to associate interrupts with ISRs for the ARM processor we are using in lab. The "B" instruction branches to the target address. (5 points)

○ ✓	○
g_pfnVectors:	g_pfnJumps:
.word _estack	B _estack
.word Reset_Handler	B Reset_Handler
.word NMI_Handler	B NMI_Handler
.word HardFault_Handler	B HardFault_Handler
.word MemManage_Handler	B MemManage_Handler
. . .	. . .

2. Considering the code below, what condition will the data structure referenced by G_slhdr be in if the ISR executes at ***B*** when called from ***A***. [X]→NULL indicates a slnode with data X and a next pointer to NULL. [X]→[Y]→NULL indicates an slnode with data X and a next pointer to another slnode with data Y and a next pointer to NULL. Show your work, e.g., in the space to the right of the code. (5 points)

   ◯ [NULL]→[7]→[5]→NULL

   ◯ [NULL]→[unallocated memory]

   ◯ [NULL]→[5]→NULL

   ◯ ✓ [NULL]→[5]→[unallocated memory]

   ◯ [NULL]→NULL

   ◯ [7]→[5]→NULL

```
#include <malloc.h>
#include <assert.h>
```

```
struct slnode {
  void * data;
  struct slnode * next;
};

typedef struct slnode slnode;

slnode * create_slnode(void * data, slnode * next) {
  slnode * new_nd = malloc(sizeof(slnode));
  assert(new_nd);
  new_nd->data = data;
  new_nd->next = next;
  return new_nd;
}

slnode * create_slist_header(void) {
   return create_slnode(NULL, NULL);
}

slnode * slist_insert_after(slnode * loc_nd, void * data) {
  slnode * new_nd = create_slnode(data, loc_nd->next);
// ***B***
  loc_nd->next = new_nd;
  return new_nd;
}

void slist_delete_after(slnode * loc_nd) {
  slnode * delete_target = loc_nd->next;
  loc_nd->next = loc_nd->next->next;
  free(delete_target);
}

static slnode * G_slhdr = NULL;

void __attribute__((interrupt)) ISRFunc(void) {
  slist_delete_after(G_slhdr);
}

int main(void) {
  int x = 5;
  int y = 7;

G_slhdr = create_slist_header();
  slist_insert_after(G_slhdr, &x);
  slist_insert_after(G_slhdr, &y);
  slist_insert_after(G_slhdr, &x); // ***A***
  return 0;
}
```

3. After the previous example, select one bubble to indicate whether there will be memory leaks, i.e., heap-allocated objects that are no longer referenced and are therefore impractical to free? (5 points)

   - ◯ No memory leaks.
   - ◯ ✓ One slnode will leak.
   - ◯ Two slnodes will leak.
   - ◯ One slnode and one integer will leak.

◯ Two slnodes and two integers will leak.

◯ One slnode and one void will leak.

6 MMIO and APB (17 points)

You are given an APB memory-mapped input-output hardware interface that has three push buttons and one LED. Each push button has its own read-only 32-bit memory address and is mapped to bit position 31. The LED has its own write-only 32-bit memory address and is mapped to bit position 0.

Below is the implementation of the Verilog module that can be controlled to by writing to the memory-mapped input-output registers.

```
module hello_world_module(
input PCLK, input PRESERN, input PSEL, input PENABLE, input [7:0] PADDR,
output PREADY, output PSLVERR,
input PWRITE,
input [31:0] PWDATA,
output [31:0] PRDATA,
input button_a, input button_b, input button_c,
output led);

  assign PSLVERR = 0;
  assign PREADY = PENABLE;
  wire enable, read_a_en, read_b_en, read_c_en;
  assign enable = PSEL & PENABLE & PWRITE & (PADDR == 0);
  assign read_a_en = PSEL & PENABLE & ~PWRITE & (PADDR == 4);
  assign read_b_en = PSEL & PENABLE & ~PWRITE & (PADDR == 8);
  assign read_c_en = PSEL & PENABLE & ~PWRITE & (PADDR == 12);
  assign PRDATA[31] =
    (read_a_en) ? button_a :
    (read_b_en) ? button_b :
    (read_c_en) ? button_c :
    1'b0;
  always @(posedge PCLK) begin
    if (enable) begin
      led <= PWDATA[0];
    end
  end
endmodule
```

1. Write a C function that reads in the state of three buttons and turns on the LED if all of them are pressed simultaneously. Assume that PSEL is configured to be high when memory locations 0x40050000–0x4005000F are accessed. The buttons and LEDs are active low. You may assume that the buttons will not change states during the duration of this function call. (9 points)

   void hello_world_fx(void) {

```
  uint32_t reg_a_val = *((uint32_t *) 0x40050004);
  uint32_t reg_b_val= *((uint32_t *) 0x40050008);
  uint32_t reg_c_val = *((uint32_t *) 0x4005000C);
  if (((reg_a_val >> 31) | (reg_b_val >> 31) | (reg_c_val >> 31)) == 0) {
    *((uint32_t *) 0x40050000 = 0;  // turn the led on
  } else {
// else statement is not necessary. Question does not explicitly ask to
// turn off LED.
  }
  *((uint32_t *) 0x40050000 = 1;
}
```

2. Which control signal uniquely identifies this bus transaction (or uniquely targets this device)? No explanation is needed. (2 point)
   PSEL
3. Which control signal is used to distinguish between idle, setup, and access states? No explanation is needed. (2 point)
   PENABLE
4. Which control signal is used by the peripheral to signal the processor that it is available for access? No explanation is needed. (2 point)
   PREADY
5. How many wait states does this hardware generate? Justify using at most two sentences. (2 points)
   Zero. PREADY is tied to PENABLE.

7 ABI (12 points)

Consider the function "integer_binary_logarithm" below.

```
int integer_binary_logarithm(uint32_t x);

void print_debug(uint32_t current_num, int counter);

integer_binary_logarithm:
  PUSH {LR}
  MOV r1, #-1
  while:
    BL print_debug
    CMP r0, #0
    BEQ result
    LSR r0, #1
    ADD r1, r1, #1
    B while
  result:
    MOV r0, r1
    POP {PC}
```

It has one argument, uint32_t x, and one return value, which is an integer that is $\lfloor \log_2(x) \rfloor$ (note: when x = 0, the function is allowed to return -1). The function "integer_binary_logarithm" calls an ABI compliant function called "print_debug," implemented by your lab partner, that takes in two integers as arguments to print out. The implementation of "print_debug" is not shown. You create a test suite, and on your first round of testing, you find out that integer_binary_logarithm is behaving as expected and is returning the correct values. You conduct further testing, but this time, you replace your implementation of "print_debug" with your friend's implementation of "print_debug," which is still ABI compliant. You run the same test suite as before for a second round of testing. However, this time, "integer_binary_logarithm" returns incorrect output. Use at most two sentences to explain why your function failed to reveal the bug.

The function "integer_binary_logarithm" does not save the values of the caller-save registers r0 and r1 before branching to "print_debug". The original implementation of print_debug may not have altered r0 or r1, but the new implementation of print_debug must have altered r0 or r1. Note: The wording of the question may have led to some confusion since "your function" was not very clear (a clearer question would use different names for the functions and clearer names for the people). To get full credit, students needed to correctly identify that there was no caller-save and that the new implementation of print_debug overwrites r0 and/or r1.

8 Timers (10 points)

Consider a count-down timer that fires an interrupt and copies the value in MMIO register TOP to its COUNTER register when the COUNTER register reaches zero. The PRESCALER register holds an integer, n. The CPU crystal frequency of 1024 kHz is divided by 2^n before being used to clock the counter. n is 8. If you would like the timer to wait five seconds, fire an interrupt, then periodically fire interrupts once per second after that time, what values should you initialize the COUNTER and TOP registers to? You may only initialize those two registers and you may only write each once, i.e., you may not update them within your ISR. Show your work in the empty space below the question. The registers are large enough to hold any of the unsigned ints listed below.

- COUNTER: ◯ 0 ◯ 400 ◯ 500 ◯ 1000 ◯ 4000 ◯ 5000 [✓ 20000] ◯ 25000
- TOP: ◯ 0 ◯ 400 ◯ 500 ◯ 1000 [✓ 4000] ◯ 5000 ◯ 20000 ◯ 25000