Name: ______________________________ Student # : ______________________

# COMP ENG 4DS4
# Embedded Systems
# Midterm Exam on February 12, 2009

This midterm exam includes 4 pages and 2 questions. Write your name and student # at the top of this page. Answer all questions. Only the responses in the answer spaces will be graded. This is a closed book midterm examination. No reference material of any kind is permitted. No calculators of any kind are permitted. Do all questions. **The duration is 45 minutes**.

## Question 1 (*8 marks*)

Consider the enclosed program written in C:

```
#include <stdio.h>
#include <stdlib.h>

void* return_message(void *message, short int type)
{

    int offset = 3;
    static void *return_pointer = NULL;

    switch (type) {
        case 0: return_pointer=(char *)message+(offset--);
                break;
        case 1: return_pointer=(short int *)message+(offset--);
                break;
        case 2: return_pointer=(float *)message+(offset--);
                break;
        case 3: return_pointer=(double *)message+(offset--);
                break;
    }

    return (void *)return_pointer;
}

int main(void)
{
    char message[] = "A message for this problem";

    for (int i=0; i<4; i++)
        printf("message %d: %s\n", i, (char *)return_message((void *)message, i));

    return 0;
}
```

***(a)*** In the box provided below write down the messages printed as the given program executes.

```
message 0: essage for this problem
message 1: age for this problem
message 2: r this problem
message 3: em
```

***(b)*** Consider that the variable `offset` from the function `return_message` is declared as follows:

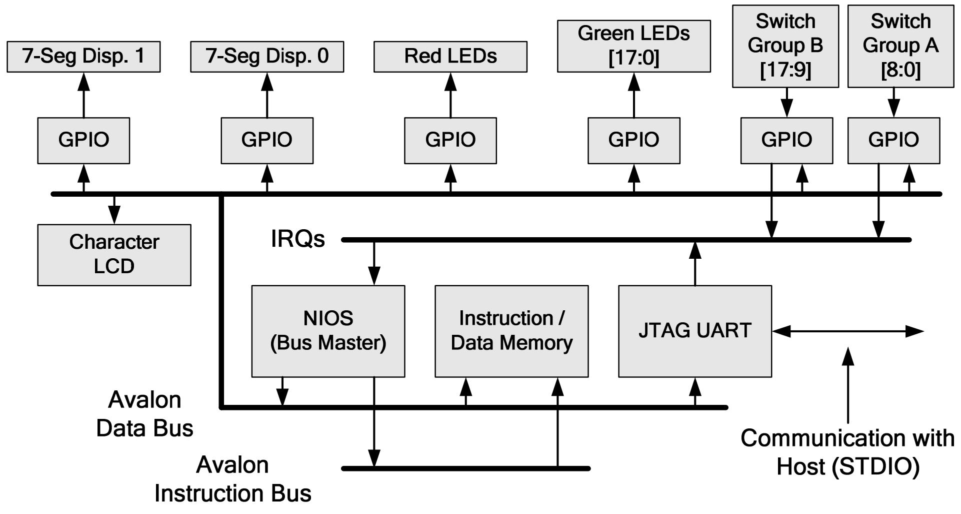
```
static int offset = 3;
```

In the box provided below write down the messages printed as the modified program executes.

```
message 0: essage for this problem
message 1: ssage for this problem
message 2: ssage for this problem
message 3: A message for this problem
```

## Question 2 (*12 marks*)

Consider an embedded system as shown in the figure below.



Name: ___________________________________ Student # : _________________________

The two input peripherals have three registers addressable through offset 0 (data register); offset 2 (interrupt mask register) and offset 3 (edge capture register). The edge capture register will trigger the interrupts whenever any of its bits becomes a 1 (either edge on an input will set the respective bit in the edge register to a 1). Note, the interrupt service routine can read the edge register, however it must reset it when the interrupt service is complete. The output peripheral has only one data register at offset 0 which can only be written by the embedded processor. The outputs are active high. The setup can be used to prototype a vending machine, as explained below.

The four least significant bits of Group A are connected to input switches, which are used to feed money (coins) to the vending machine. Whenever they toggle (either positive or negative edge) the `amount` accumulated for the current transaction is updated using the following values:

- Position 3 of Group A – 25 cents
- Position 2 of Group A – 10 cents
- Position 1 of Group A – 5 cents
- Position 0 of Group A – 1 cent

The four least significant bits of Group B are connected to four input switches, which give the commands to the vending machine as follows:

- Position 3 of Group B – cancel the current transaction
- Position 2 of Group B – purchase item 2 – note: the price of item 2 is 70 cents
- Position 1 of Group B – purchase item 1 – note: the price of item 1 is 45 cents
- Position 0 of Group B – purchase item 0 – note: the price of item 0 is 20 cents

The four least significant bits of the output peripheral Green LEDs are showing the following:

- Position 3 of Green LED – activated for 3 seconds only (after a transaction completes)
- Position 2 of Green LED – activated for 3 seconds only (after a transaction completes) if change is provided
- Position 1 of Green LED – activated for 3 seconds only when there is no sufficient `amount` for the current transaction (for generating a signal for 3 seconds you can use a `delay(3)`function made available to you)
- Position 0 of Green LED – activated so long as there is an active transaction in progress

In addition to the behavior described above, we have the following simplifying assumptions:

- A new transaction becomes active only if the `amount` is 0 and any of the Group A switches is toggled
- If a purchase/cancel command is given when a transaction is not active then no action will be taken
- During an active transaction, the `amount` is updated (using the values dependent on the switch positions, as described above) when any of the Group A switches is toggled (either edge will trigger the interrupt request)
- An active transaction completes either when the cancel command is given or when an item was purchased
- An item has been purchased if, during an active transaction, a purchase command was given for a particular item and the `amount` accumulated for the current transaction is greater than or equal to the price of the respective item
- If a purchase item command has been given for an active transaction when there is no sufficient `amount` in the vending machine then output LED from position 1 will be activated for 3 seconds (as described above); note however, the current transaction will continue with the same `amount` as before the command was given
- When a transaction is complete (either through a purchase or a cancel) it is assumed that the machine has enough coins to return the exact change (at which time the `amount` will become 0)
- The machine will never overflow (neither in the number of coins nor for the representation of `amount`)
- To simplify the interrupt handling, it is assumed that no two (or more) inputs will ever change at the same time in our environment (note, it does not matter if the inputs are from the same group or not); hence both interrupt service routines will work under the assumption that only one bit has been changed in the edge register
- While an interrupt is serviced, it is assumed that no other interrupt request will come from the environment (note, it does not matter whether the serviced interrupts or the interrupt requests are from Group A or B)
- It is assumed that no interrupt occurs before the program execution enters the `while (1);` loop

The backbone of the source code is provided on the following page. The interrupt service routines are already setup and the context that is passed is the `amount` of money available for the transaction in progress (note: if there is no transaction in progress then the `amount` is obviously 0). You need to complete the source code for the two interrupt service routines in order to satisfy the behavior described above. No changes are permitted in the `main` function. VERY IMPORTANT NOTE: In `SW_GRPB_interrupt` you are permitted to write the code for handling only one of the purchase commands (because the basic principles for handling the other two purchase commands are the same). However, you must explain briefly using comments in the code what would be the differences for handling the other two purchase commands. Note, the handling of the cancel transaction command must be coded!

```c
void SW_GRPA_interrupt(int *amount) {
    // the test for *amount == 0 is safe, but also redundant according to the spec
    // (because no interrupt request happens when an existing interrupt is serviced)
    if (*amount == 0) IOWR(LED_GREEN_O_BASE, 0, 0x1);

    if ((IORD(SWITCH_GRPA_I_BASE, 3) & 0xF) == 0x1) *amount += 1;
    if ((IORD(SWITCH_GRPA_I_BASE, 3) & 0xF) == 0x2) *amount += 5;
    if ((IORD(SWITCH_GRPA_I_BASE, 3) & 0xF) == 0x4) *amount += 10;
    if ((IORD(SWITCH_GRPA_I_BASE, 3) & 0xF) == 0x8) *amount += 25;

    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0);
}

void SW_GRPB_interrupt(int *amount) {

    if (*amount != 0) {
        if ((IORD(SWITCH_GRPB_I_BASE, 3) & 0xF) ==0x4) {
            if (*amount < 70) {
                IOWR(LED_GREEN_O_BASE, 0, 0x3);
                delay(3);
                IOWR(LED_GREEN_O_BASE, 0, 0x1);
            }
            else if (*amount >= 70) {
                if (*amount == 70) IOWR(LED_GREEN_O_BASE, 0, 0x8);
                else IOWR(LED_GREEN_O_BASE, 0, 0xC);
                delay(3);
                IOWR(LED_GREEN_O_BASE, 0, 0x0);
                *amount = 0;
            }
        }

        // to purchase items 1 or 0 then we can change the above code as follows:
        // the edge register will be compared against 0x2 and 0x1 respectively
        // the amount will be compared against 45 and 20 respectively

        if ((IORD(SWITCH_GRPB_I_BASE, 3) & 0xF) ==0x8) {
            IOWR(LED_GREEN_O_BASE, 0, 0xC);
            delay(3);
            IOWR(LED_GREEN_O_BASE, 0, 0x0);
            *amount = 0;
        }
    }
    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0);
}

int main(void)
{
    volatile int amount = 0; // amount of money for the current transaction

    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0); // edge capture register
    IOWR(SWITCH_GRPA_I_BASE, 2, 0xF); // IRQ mask
    alt_irq_register(SWITCH_GRPA_I_IRQ, &amount, (void*)SW_GRPA_interrupt);

    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0);
    IOWR(SWITCH_GRPB_I_BASE, 2, 0xF);
    alt_irq_register(SWITCH_GRPB_I_IRQ, &amount, (void*)SW_GRPB_interrupt);

    IOWR(LED_GREEN_O_BASE, 0, 0x0);
    while (1);
    return 0;
}
```