Université d'Ottawa
Faculté de genie

École d'ingénierie et de technologies de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information Technology and Engineering

COURSE: CEG3136/CEG3536
SEMESTER: Fall 2010

PROFESSORS: Gilbert Arbez
DATE: Dec 17, 2010
TIME: 14h00 – 17h00

Computer Architecture II

FINAL EXAMINATION

Solution

NAME and STUDENT NUMBER:_____/_____

Instructions:

- Answer ALL questions on the examination paper.
- This is a close-book examination**.**
- Use the provided space to answer the following questions. If more space is needed, use the back of the page.
- Show all your calculations to obtain full marks.
- Calculators are allowed.
- Read all the questions carefully before you start.

There are three (3) parts in this examination.

Part 1	**Short Answer (Theory)**	**20 marks**	
Part 2	**Assembler Prog.**	**5 marks**	
Part 3	**Update to the Alarm System**	**35 marks**	
Total		**60 marks**	

Total Pages: 11

Part 1 Short Answer Questions (Theory) (total 20 points)

Mark the correct answer or give a short answer to the following questions.

1. (2 points) In which bus would you find interrupt lines going to the CPU?
   Control bus.

2. (2 points) What is the role of the interrupt vector?
   To provide the address of an ISR for the corresponding interrupt.

3. (2 points) Would you configure the IRQ to be level triggered or edge triggered if more that one device was connected to the IRQ pin?
   Level triggered, since devices would be wire-ored.

4. (2 points) Is is possible to write to the SCI Receive Data Register? Explain your answer.
No, the Transmit Data Register and Receive Data Register occupy the same address, thus when writing to the address, it is the contents of the Transmit Data Register is written, not the Receive Data Register.

5. (4 points) What component does an timer channel configure for input-capture use: the comparator or the latch? Explain. Give the three possible input capture event configurations.
   The latch is used to store the current value of the counter into the data register when an input capture event occurs in the signal applied to the channel pin. The possible input capture event configurations are rising edge, falling edge or both rising/falling edge.

6. (2 points) What is the typical action taken to clear a flag when fast clear is NOT enabled?
   Write a 1 to the flag to reset it to 0.

7. (4 points) Consider that 2 signals, one containing frequences from 10 kHz to 20 kHz (10000 to 20000 Hz) and the other frequences from 5 kHz to 8 kHz, applied to two adjacent pins on the HCS12 ATD port. Assuming that samples from the signals are converted to 8-bit binary values using the SCAN mode and an 8 cycles used for the 2nd phase of the sampling time, determine and select an ATD clock frequency available in the HCS12 ATD module to process these signals. Assume a 24 MHz system clock.
   Sampling frequency = 40 kHz, sampling time = conversion sequence time= 25 µsec. Number of conversion cycles/conversion = 8+2+8 = 18 ATD clock cycles/conversion, and thus 2 x 18 = 36 cycles per conversion sequence. ATD clock cycle = 25 µsec/conversion sequence divided by 36 ATD clock cycles/conversion = 0.694 µsec/ ATD clock cycles. Minimum ATD frequency = 1.44 MHz. Use 1.50 MHz clock available in the ATD module.

8. (2 points) Which channel in the timer can be used to synchronize signals from multiple channels?

   Channel 7

Part 2 – Assembler Programming (total 5 points)

Translate the following C Pseudo-code to assembler.

C Function

```
/*-------------------------
Function: getKCode()
Description:
   Gets a code from PORTA
   that corresponds
   to a keypress.
-------------------------*/
byte getKCode()
{
  byte code;
  PORTA = 0b00001110;
  code = PORTA;
  if(code == 0b11111110)
  {
    PORTA = 0b00001101;
    code = PORTA;
    if(code == 0b11111101)
    {
      PORTA = 0b00001011;
      code = PORTA;
      if(code == 0b11111011)
      {
        // Assume bit 3 produces code
        PORTA = 0b00000111;
        code = PORTA;
      }
    }
  }
  PORTA = 0x00;
  return(code);
}
```

Assembler Code

```
%-----------------------------------
Subroutine: getKCode()
No parameters
Returns code in acc B.
Description:
   Gets a code from PORTA
   that corresponds
   to a keypress.
%-----------------------------------%
getKCode: movb #%00001110,PORTA
          ldab PORTA
          cmpb #%11111110
          bne endif
          movb #%00001101,PORTA
          ldab PORTA
          cmpb #%11111101
          bne endif
          movb #%00001011,PORTA
          ldab PORTA
          cmpb #%11111011
          bne endif
          movb #%00000111,PORTA
          ldab PORTA
endif :   clr PORTA
          rts
```

Part 3 – Updating the Alarm System (35 points)

A New Keypad Module

Recall that the the keypad module (assembler module) in the Alarm System project is based on polling and had the side effect of occuping the CPU when a key is pressed and not released. The objective of this question is to redesign the Keypad Module to remove this side effect. The challenge is to apply interrupts to monitoring the keypad. Given that the keypad is a passive device (i.e. cannot generate interrupts), a timer channel will be used to invoke a sequence of interrupts for checking the state of the key pad. All logic for debouncing (both at the pressing and releasing of a key) and determining the key pressed is implemented in the interrupt service routine.

Assumptions:

1. The timer main counter increments every 1.3333 micro-seconds.
2. The entry points to the new module remain `initKeyPad()`, `readKey()`, and `pollReadKey()`.
3. Assume that the following functions are available (that is, you do not need to create these functions).
    a. `byte getKCode()` – returns a code value of the key that is pressed using Port A. This function is called after debouncing the key press (see Part 2).
    b. `char getAscii(byte code)` – returns the ASCII value corresponding to the key code.
4. Timer channels are used by other modules in the Alarm System Project as follows:
    a. Channel 0 – used by the Delay Module
    b. Channel 1 – used by the Segment Display Module
    c. Channel 2 – used by the Thermistor Module
    d. Channel 5 – used by the Siren Module
    e. Channel 6 – used by the Alarm System Module

A. (10 points) Provide an overall design of your module. Use **English** text (possibly diagrams) to explain the logic of each of the entry points and the ISR and any other functions you are using (do not provide design for the `getKCode` and `getASCII` functions).

- **`initKeyPad` – setup the timer channel 4 to generate an interrupt Setup the timer channel (4) to generate an interrupt every 10 ms (with a clock period of 1.333 microsecionds, count 7500 counter tics). The 10 ms is chosen to debounce for that period of time. PORTA is configured by setting DDRA to 0x0F, and enabling pull-up resistors.**
- **`pollReadKey` – check the value of the global variable `keyCode`. If it is not equal to `NOKEY`, translate the code to an ASCII value using `getAscii`, set `keyCode` to `NOKEY` and return the value found; otherwise return `NOKEY`.**
- **`readKey` – Wait until asciiCode does not contain `NOKEY`. Then copy the value, set `keyCode` to `NOKEY` and return the value found, followed by a call to `getAscii(code)` to translated the code to an ASCII character.**

ISR Design

- **Define a variable, `state`, that will assume one of the following values**
  - **WAITING_FOR_KEY – waiting to detect a change in the status of the keypad, i.e. to see the start of the key press. This is done by checking that PORTA is different from 0xF0. If so, then copy the value of PORTA into a static variable, say code, and then change state to DEB_KEYPRESS.**
  - **DEB_KEYPRESS – state will have this value after a keypress was detected. PORTA is checked and compared to the value of code. If it is different, then state is reset to WAITING_FOR_KEY, otherwise, getKCode() is called to get the code of the current keypress, Then state is updated to WAITING_FOR_REL – to wait for release.**
  - **WAITING_FOR_REL – PORTA is checked for the value 0xF0. If detected, then state is updated to the value DEB_REL, to indicate that debouncing of the release is being done.**
  - **DEB_REL – if PORTA is 0xF0, then set state to WAITING_FOR_KEY, otherwise, reset to WAITING_FOR_REL. At this point the key code is saved in the global variable `keyCode`.**

B. (15 points) First give the contents of the header file KeyPad.h. Then provide the C code for each of the entry points, the ISR, and any other functions (other than `getKCode` and `getAscii`).

KeyPad.h file

```
/*-------------------------------------------------------------------------
   File: keyPad.h
   Description:  Contains definitions and prototypes for the Keypad Module
-------------------------------------------------------------------------*/
#ifndef _KEYPAD_H
#define _KEYPAD_H

//C Prototypes to assembler subroutines - Entry Points
void initKeyPad(void);
char pollReadKey(void);
char readKey(void);

// Some Definitions
#define NOKEY 0  // See KeyPad.c - to indicate no key pressed
#define BADCODE 0xFF // indicates that key code was not mapped to ASCII char


#endif /* _KEYPAD_H */
```

KeyPad.c file

```
/*-------------------------------------------------------
    File: keyPad.c
    Description: Module for reading the keypad using interrupts
                 and timer channel 4.
-------------------------------------------------------*/

#include "mc9s12dg256.h"
#include "keyPad.h"
#define BIT4 0b00010000;

#define TENMSEC 7500  // 10 ms = 7500 x 1 1/3 micro-second

// Global variables
volatile byte keyCode;

// Local Function Prototypes
char getAscii(byte);
byte getKCode(void);

/*-------------------------------------------
Function: initKeyPad
Description: initializes hardware for the
             KeyPad Module.
-------------------------------------------*/
void initKeyPad(void)
{
  // Sets up Port A register
  DDRA = 0x0F;  // Set output direction for lower 4 bits of PORT A
  PORTA = 0x00;  // set all outputs to 0 - should read 0xF0
  PUCR |= 0x01;  // setup pullup resistors on Port A.
  // Sets up timer channel to generate interrupts
  // Assume that timer is enabled elsewhere with 1 1/3 microsec ticks
  // for controlling displays
      TIOS |= BIT4; // Set ouput compare for TC4
```

```
      TIE |= BIT4; // Enable interrupt on TC4

      TC4 = TCNT + TENMSEC; // enables timeout on channel 4
      keyCode = NOKEY;
}

/*-------------------------------------------------
Interrupt: readKey
Description: Waits for a key and returns its ASCII
             equivalent.
-------------------------------------------------*/
char readKey()
{
    char ch;
    while(keyCode == NOKEY) /* wait */;
    ch = getAscii(keyCode);
    keyCode = NOKEY;
    return(ch);
}
/*-------------------------------------------------
Interrupt: pollReadKey
Description: Checks for a key and if present returns its ASCII
             equivalent; otherwise returns NOKEY.
-------------------------------------------------*/
char pollReadKey()
{
    char ch;
    if(keyCode == NOKEY) ch = NOKEY;
    else
    {
        ch = getAscii(keyCode);
        keyCode = NOKEY;
    }
    return(ch);
}

/*-------------------------------------------------
Interrupt: key_isr
Description: Display interrupt service routine
             that checks keypad every 10 ms.
-------------------------------------------------*/
// State values
#define WAITING_FOR_KEY 0
#define DEB_KEYPRESS    1
#define WAITING_FOR_REL 2
#define DEB_REL         3

void interrupt VectorNumber_Vtimch4 key_isr(void)
{
  static byte state = WAITING_FOR_KEY;  // state of keypad check
  static byte code;

  switch(state)
  {
    case WAITING_FOR_KEY:
      code = PORTA;
      if(code != 0xF0) state = DEB_KEYPRESS;
      break;
    case DEB_KEYPRESS:
      if(PORTA != code) state = WAITING_FOR_KEY;
      else
```

```
      {
         code = getKCode();
         state = WAITING_FOR_REL;
      }
      break;
    case WAITING_FOR_REL:
      if(PORTA == 0xF0) state = DEB_REL;
      break;
    case DEB_REL:
      if(PORTA != 0xF0) state = WAITING_FOR_REL;
      else
      {
          keyCode = code;  // save ASCII code value
          state = WAITING_FOR_KEY;
      }
      break;
  }
      // Set up next interrupt (also clears the interrupt)
      TC4 = TC4 + TENMSEC;
}
```