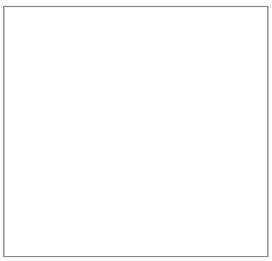


IS1200/IS1500

Lab 3 – I/O Programming
2015-09-28

Introduction

Welcome to our third lab! In this laboratory exercise you will learn the fundamentals of input/output, timers, and interrupt programming. After finishing this lab, you should be able to

1. Write code for general-purpose input/output (GPIO) ports.
2. Write code to initialize and use simple input/output devices, such as buttons and LEDs.
3. Write code to initialize and use devices with handshaking, such as a programmable timer.
4. Write code to initialize and handle interrupts.
5. Have a general understand for how to read non-trivial technical manuals, data sheets, and header files.

Preparations

You must do the lab work in advance, except for the Surprise Assignment. The teachers will examine your skills and knowledge at the lab session.

We recommend that you book a lab-session well in advance, and start preparing at least a week before the session.

You can ask for help anytime before the lab session. There are several ways to get help; see the course website for details and alternatives.

During the lab session, the teachers work with examination as well as offering help. Make sure to state clearly that you want to *ask questions*, rather than being examined.

Sometimes, our teachers may have to refer help-seekers to other sources of information, so that other students can be examined.

Resources and Reading Guidelines

- Lectures 5 and 6.
- Exercise 2 (including the suggested solutions).
- Harris & Harris (2013) course book, chapters 8.5-8.7 (available online). You may focus on sections 8.6.2-8.6.5 about GPIO, Timers, and Interrupts.
- PIC32 Reference Manuals and ChipKIT Reference Manuals.
- The C header file pic32mx.h, that is available here:
  `/opt/mcb32/include/pic32mx.h` on Windows and Linux, and here:
  `/Applications/mcb32tools.app/Contents/Resources/Toolchain/include/pic32mx.h`
  on Mac OS (after that you have installed the MCB Tools).
- See the course web page *Literature and Resources* for links and details.

Examination

Examination is for each assignment. Examining one assignment takes 5–15 minutes. Make sure to call the teacher immediately when you are done with an assignment.

Please state clearly that you want to *be examined*, rather than getting help.

The teacher checks that your program behaves correctly, and that your code follows all coding requirements and is easily readable. In particular, all assembly-language code **must** conform to the calling conventions for MIPS32 systems. Compile-time warnings are not acceptable.

The teacher will also ask questions, to check that your knowledge of the design and implementation of your program is deep, detailed and complete. When you write code, make detailed notes on your choices of algorithms, data-structures, and implementation details. With these notes, you can quickly refresh your memory during examination, if some particular detail has slipped your mind.

You must be able to answer all questions. In the unlikely case that some of your answers are incorrect or incomplete, we follow a specific procedure for re-examination. This procedure is posted on the course website.

Assignments

Through-out all assignments, test your code continually by running it on the Uno32 board with Basic IO Shield.

Assignment 1: Polling switches

In this assignment, you will write code that (repeatedly) examines the values of one or more input bits. This is called polling, and is the simplest way to check input from a switch or push-button. You will also write code to send output-data to LEDs.

Parts a and b of this assignment are preliminaries, laying the foundation for your work.

a) Create a new directory `time4io`. Then find your previous lab work from the assembly-programming lab 1. Copy all your files from lab 1 (including `labwork.S`) into the new directory `time4io`.

b) Open a Terminal window (an msys2 window on Windows). Change to the new directory and type the command

```
make
```

Look for warnings and errors, and correct them all. Then connect the Uno32 board and type

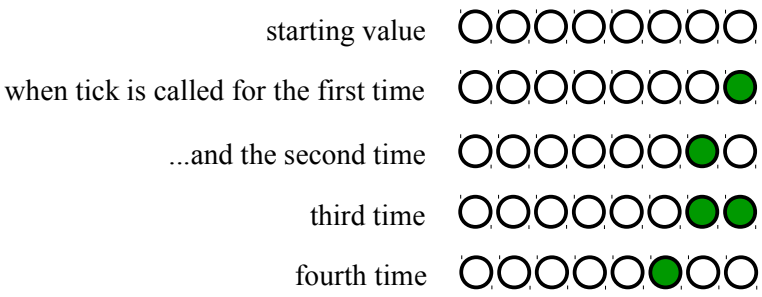
```
make install
```

Check that the code runs correctly on the Uno32 board (with Basic IO Shield). After these steps, you'll know that your old code still works (or you'll know that you have things to fix before starting this lab!).

Parts c and d deal with output-data, which is often a little simpler than input.

c) In file `mipslabwork.c`, add code in function `labinit` to initialize Port E so that bits 7 through 0 of Port E are set as outputs (i.e., the 8 least significant bits). These bits are connected to 8 green LEDs on the Basic IO Shield. Register TRISE has address `0xbf886100`. You should initialize the port using your own volatile pointer, that is, you should not use the definitions in `pic32mx.h`, yet. Do not change the function (direction) of any other bits of Port E.

d) In file `mipslabwork.c`, add code in function `labwork` to increase the binary value shown on the 8 green LEDs once each time the function `tick` is called. Initialize the value to 0, so that the LEDs show how many "ticks" have occurred since the program was started. See below.



The remaining parts of assignment 1 deal with input.

e) In file `mipslabwork.c`, add code in function `labinit` to initialize port D so that bits 11 through 5 of Port D are set as inputs. You should do this by using the definitions in `pic32mx.h` Do not change the function (direction) of any other bits of Port D.

f) Create a new file `time4io.c`. Begin the file with the following three lines:

```
#include <stdint.h>
#include <pic32mx.h>
#include "mipslab.h"
```

In this file, write a C function that returns the status of the toggle-switches on the board, with the following specification.

Function prototype: `int getsw( void );`
Parameter: none.
Return value: The four least significant bits of the return value should contain data from switches SW4, SW3, SW2, and SW1. SW1 corresponds to the least significant bit. All other bits of the return value must be zero.
Notes: The function `getsw` will never be called before Port D has been correctly initialized. The switches SW4 through SW1 are connected to bits 11 through 8 of Port D.

g) In file `time4io.c`, add a C function that returns the current status of the push-buttons BTN2, BTN3, and BTN4 with the following specification[1].

Function prototype: `int getbtns(void);`
Parameter: none.
Return value: The 3 least significant bits of the return value must contain current data from push buttons BTN4, BTN3, and BTN2. BTN2 corresponds to the least significant bit. All other bits of the return value must be zero.
Notes: The function `getbtns` will never be called before Port D has been correctly initialized. The buttons BTN4, BTN3, and BTN2, are connected to bits 7, 6 and 5 of Port D.

h) In file `mipslabwork.c`, modify the function `labwork` to also call `getbtns`. If any button is pressed, the function `getsw` must be called, and the value of the variable `mytime` updated as follows (see next page).

- Pressing BTN4 copies the value from SW4 through SW1 into the first digit of mytime. If SW4 through SW1 are set to 0100, time would change from, say, 16:53 into 46:53.
- Pressing BTN3 copies the value from SW4 through SW1 into the second digit of mytime. If SW4 through SW1 are set to 0100, time would change from, say, 16:53 into 14:53.
- Pressing BTN2 copies the value from SW4 through SW1 into the third digit of mytime. If SW4 through SW1 are set to 0100, time would change from, say, 16:53 into 16:43.

The figures below illustrate this, with an example.

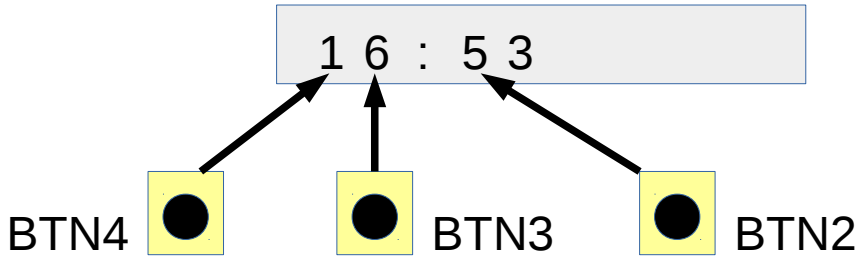
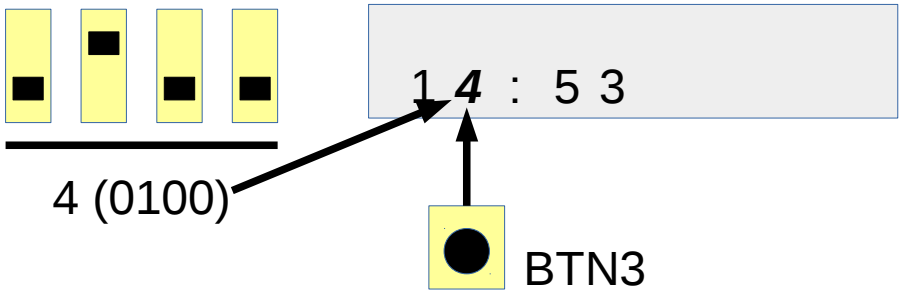


Diagram showing which digit should be updated, depending on which button is pressed.

If the toggle-switches (SW4 through SW1) have been set to 0100, pressing a button would change the corresponding digit into a '4'.



In this example, the switches are set to 4 (0100 binary), so the selected digit changes into a '4'.

When running your code on the Uno32 board with Basic IO Shield, verify that you can set all three digits: minutes (two digits) and seconds (only the tens digit).

1 You may also (optionally) implement this function to support all four push buttons: BTN1, BTN2, BTN3, and BTN4. This makes the function a bit more complicated because you need to use more than one Port. If you decide to do this optional and more complicated exercise, please inform your lab assistant about this when you are being examined on this exercise. Also, please note that you need to update the code for exercise h) accordingly.

Questions for Assignment 1

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

- If you press BTN3 and BTN2 at the same time, what happens? Why?
- If you press BTN3 quickly, does the time update reliably? Why, or why not?
- Three device-registers for input/output control are TRISE, TRISESET, and TRISECLR. Their functions are related. How? What are the differences? Did you use one or more of these device-registers in your code? Why, or why not?
- In the generated assembly code, in which MIPS register will the return values from functions `getbtns` and `getsw` be placed in. You should be able to answer this question without debugging the generated assembly code.
- In this exercise, we explained which bits that should be used in Port D and Port E. How can you find this information in the PIC32 and ChipKIT manuals? Be prepared to explain the procedure for how to find this information in the manuals.
  Advice: check the lecture slides from lecture 5 for ideas.

Assignment 2: Timer

In this assignment, you will use a timer device. Timers are controlled directly by a very accurate pulse-generator (called a crystal oscillator). Because of this, timers can can be used for very precise measurements of time. The resolution of these measurements depends on the pulse-frequency and is usually well into the sub-microsecond range. Some examples are:

- Measuring the interval between input-pulses from a sensor on a wheel, to calculate the speed of a vehicle.
- Examining the execution time of some part of a program, to determine if additional optimization could be useful.
- Repeating some action at precise intervals, such as updating the display of a clock every second. This is what you will do in the current assignment.

a) Create a new directory `time4timer`. Copy all files from assignment 1 into the new directory.

b) In file `mipslabwork.c`, add code to initialize Timer 2 for timeouts every 100 ms (that is 10 time-outs per second). Be prepared to justify your choices of clock-rate divider and time-out period.
In file `mipslabwork.c`, change the `labwork` function so that it never calls `delay`. Instead, the `labwork` function should test the time-out event flag.
If the flag indicates a time-out event, your code must reset the flag. Make the calls to `time2string`, `display_string`, `display_update`, and `tick` conditional, so that these functions are only called in case of a time-out event. Run your code on the board and verify that the time-display is updated 10 times per second.
Note: T2CON, IFS(0), and other hardware addresses, are defined by file `pic32mx.h`.

Note: Don't let the call to `getbtns` depend on a time-out. The function `getbtns` can be called in every loop iteration. On the other hand, calling the display functions in every iteration would lead to unpleasant flickering.

c) In file `mipslabwork.c`, add a global counter `timeoutcount`. In function `labwork`, use the counter to count up to 10 time-out events. Change your code so that `time2string`, `display_string`, `display_update` and `tick` are only called once in 10 time-out events. Run your code on the board and verify that the time-display is updated once per second.

Questions for Assignment 2

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

- How is the time-out event-flag checked?
- When the time-out event-flag is a "1", how does your code reset it to "0"?
- What would happen if the time-out event-flag was not reset to "0" by your code? Why?
- Make a rough estimate of how many times per second your code checks the time-out event-flag: 1, 10, 100, 1000, 10 000, 100 000, a million, 10 million, 100 million times? How did you calculate your answer?
- Does your code write to T2CON, T2CONSET, and/or T2CONCLR? How many times do you write to each of these device-registers? Why?
- Which device-register (or registers) must be written to define the time between time-out events? Describe the function of that register (or of those registers).
- If you press BTN3 quickly, does the time update reliably? Why, or why not? If not, would that be easy to change? If so, how?

Assignment 3: Interrupts

In the previous assignments, you have used polling of input-data and polling of a timer event-flag. The downside of polling is that it keeps the processor busy, checking input and/or event flags.

With interrupts, the processor can perform any kind of useful computation instead. When input data changes, or when the timer event-flag goes on, ordinary computation is temporarily suspended; the processor starts executing an interrupt-service routine (ISR) instead.

However, the fact that ordinary program execution can be interrupted at any time makes it hard to debug problems related to interrupt programming. We will therefore limit this assignment to one source of interrupts: the timer event-flag.

Helpful hint: During debugging, you can use the function `display_debug` to examine the contents of a memory address or variable. See file `mipslab.h` for more information.

Parts a, b, c, and d of this assignment are preliminaries, laying the foundation for your work.

a) Create a new directory `time4int`. Copy all files from assignment 2 into the new directory.

b) In file `mipslabwork.c`, add the following line near the beginning of the file:

```
    int prime = 1234567;
```

c) In file `mipslabwork.c`, change the `labwork` function to look like this:

```
void labwork( void ) {
  prime = nextprime( prime );
  display_string( 0, itoaconv( prime ) );
  display_update();
}
```

d) In file `mipslabwork.c`, change the function `user_isr` to look like this:

```
void user_isr( void ) {
  time2string( textstring, mytime );
  display_string( 3, textstring );
  display_update();
  tick( &mytime );
}
```

In parts e and f, you write code that will execute on an interrupt.

e) In file `mipslabwork.c`, add code into the function `user_isr` to acknowledge interrupts from Timer 2.

f) In file `mipslabwork.c`, change function `user_isr` to update and check the global counter `timeoutcount`, so that that `time2string`, `display_string`, `display_update` and `tick` are only called once in 10 invocations of `user_isr`. This is similar to what you did in Assignment 2.

In parts g and h, you write code to enable interrupts. All the other parts must be in place before interrupts are enabled.

g) In file `labwork.S`, add a function `enable_interrupt` that executes the `ei` instruction (and then returns to the caller). The `ei` instruction enables interrupts globally.

h) In file `mipslabwork.c`, add code to the function `labinit`, to enable interrupts from Timer 2, and of course to enable interrupts globally. *Note:* IEC(0) is at `0xbf881060`, and IPC(2) at `0xbf8810b0`.

Questions for Assignment 3

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

- How is the time-out event-flag checked?
- When the time-out event-flag is a "1", how does your code reset it to "0"?
- What would happen if the time-out event-flag was not reset to "0" by your code? Why?
- For this assignment, please make a new estimate of how many times per second your code checks the time-out event-flag: 1, 10, 100, 1000, 10 000, 100 000, a million, 10 million, 100 million times? How did you calculate your answer?
- From which part of the code is the function `user_isr` called? Why is it called from there?
- Why are registers saved before the call to `user_isr`? Why are only some registers saved?
- Is the function `user_isr` called from the `labwork` function? Why, or why not?
- Which part (or parts) of your code is (are) required to enable interrupts from the timer?
- If the time-out event-flag changes to a "1" while the function `display_string` is running, could the display show some kind of incorrect pattern – perhaps wrong numbers or so? If so, for how long would the pattern be displayed? Make a reasonable estimate. These two questions are difficult, so it's okay not to have a perfect answer.

Assignment 4: Surprise assignment

You will get a surprise assignment at the lab session.

You must finish the surprise assignment during the session.