KTH Information and
Communication Technology

IS1200/IS1500

Lab 3 – I/O Programming
2024-07-30
v1.4

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 to use simple input/output devices, such as switches and LEDs.
2. Write code to initialize and use devices with handshaking, such as a programmable timer.
3. Write code to initialize and handle interrupts.
4. Have a general understand for how to read non-trivial technical manuals, data sheets, and C-code

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 (2021) course book, chapters 9.2-9.4 (available online).
  - **Note:** unlike the H&H board, which contains a built-in timer inside the processor, we our DTEK-V board deliberately disabled this feature. Hence, you must read the documentation on the timer (on the Canvas page) to understand how to program it and how to acknowledge interrupts from that source.
- See the course web page *Literature and Resources* for links and details. In particular, the documentation about the timer and the general-purpose I/O (which is the buttons and switches are).

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 RISC-V (RV32IM) 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.

This lab is examined in three parts. You can only be examined on one part at a time.

Part 1 – Assignment 1.

Part 2 – Assignment 2.

Part 3 – Assignment 3 and the Surprise Assignment.

Assignments

Through-out all assignments, test your code continually by running it on the DTEK-V board.

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) Copy all your files from the first laboratory exercise (lab1) to a new directory and delete the `labmain.S` file. Next, copy and unzip the content of the `files-lab3.zip` from the Canvas homepage into the same directory. The `labmain.c` file from the third lab essentially does the same as the `timetemplate` (from lab1) function did but in C-code.

Note: you need to make the the functions time2string, tick, delay, display_string globally visible in the `timetemplate.S` file (otherwise they cannot be called from C).

Note: in the `labmain.c` file, in the main function, you need to update the delay parameter to reflect what was set in your timetemplate function.

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

```
make
```

Then connect the DE10-Lite board and type:

```
dtekv-run main.bin
```

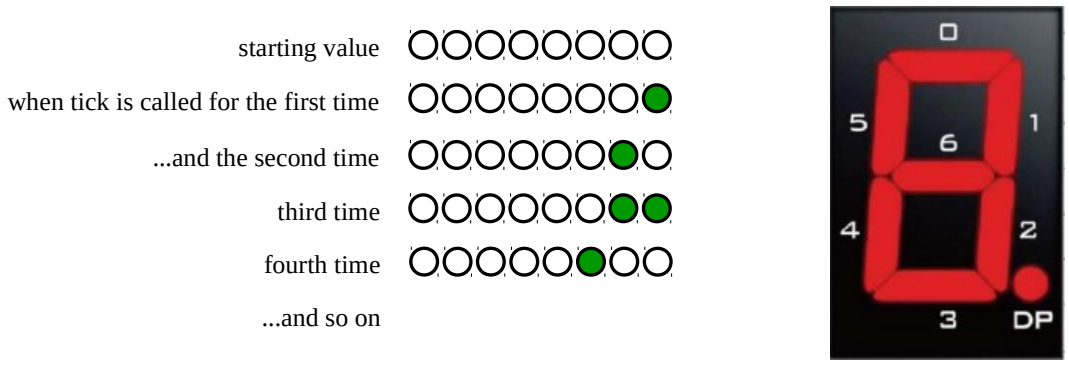
Check that the code runs correctly on the DTEK-V board (that is, identical to your lab1 output). After these steps, you'll know that the board is working.

Parts c and d deal with output-data.

c) In the lab`main.c` file, add a function to turn the LEDs on or off. The LEDs' address is `0x04000000`. Use a volatile pointer to write to this address. The DTEK-V board has 10 LEDs, which can be toggled on and off by setting the 10 least significant bits (LSBs) to either 1 or 0, respectively. For example, writing `0x7` (which is `0111` in binary) will toggle on the 1st, 2nd, and 3rd LEDs, and turn off the remaining ones.

Function prototype: `void set_leds(int led_mask);`
Parameter: `led_mask`, an integer where the 10 less significant bits correspond to the state of the corresponding led.
Return value: nothing.

d) In the `labmain.c` file, add code that runs at the start of your program. This code should increment the binary value displayed on the first 4 LEDs once each “second” has passed. Initialize the LEDs' value to 0, so that the LEDs show how many "seconds" have elapsed since the program started. The program should stop when all the first 4 LEDs are turned on simultaneously. You can use the function you created in part (c) to toggle the LEDs. Note that, at this stage, each “second” is an approximation and not an exact one.



***Left**: Example of how the LEDs should be blinking at the beginning of your application*
***Right**: a 7-Segment display showing which bits control which LEDs (DP is controlled by bit 7).*

e) Create a C function that sets the value to be shown on the 7-segment displays. There are six of these displays. The address of the first display is `0x04000050`, and each subsequent display has an offset of `0x10`. Therefore, the address of the last display is `0x040000a0`. *Hint*: writing '0' to a bit position lights up the LED.
The function should have the following specification:

Function prototype: `void set_displays(int display_number, int value);`
Parameter:
- display_number, corresponds to the display the user wants to set.
- value, corresponds to the value to display.
Return value: nothing.

The remaining parts of assignment 1 deal with input.

f) Create a C function that returns the status of the toggle-switches on the board. The toggles (or switches) are mapped to memory address `0x04000010`. This function should have the following specification.

Function prototype: `int get_sw( void );`
Parameter: none.
Return value: The ten least significant bits of the return value should contain data from switches. SW1 corresponds to the least significant bit. All other bits of the return value must be zero.

g) Create a C function that returns the current status of the second push-button. The button is mapped to memory address `0x040000d0`. Note, the first push-button on the board is used to reset the board.

Function prototype: `int get_btn(void);`
Parameter: none.
Return value: The least significant bit of the return value must correspond to the status of the second button.

h) After the start sequence (developed in part [d]), enter an infinite loop. In this infinite loop, you should count the both seconds,minutes *and hours* passed and display them on the 7-segment displays (use the previously developed `set_displays`). The two rightmost displays show the seconds, the middle two displays show the minutes passed, and the two leftmost displays show the hours passed.

Within the infinite loop, call `get_btn` to check if a button is pressed. If a button press is detected, read the switch values with `get_sw` and update the display as follows:[1]

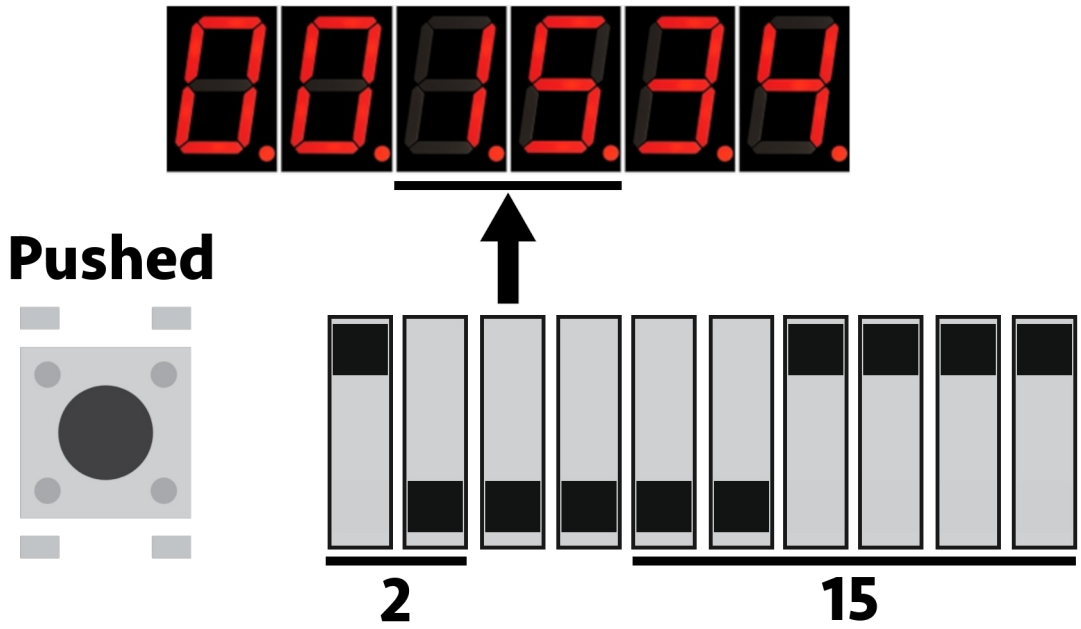
1 To show that you understand use one of the remaining switches to exit the infinit loop and end the program.

- The state of the two left-most (most significant) switches determine which pair of displays to update:
  - If the state is `01` (left-switch down, right-switch up) modify the second counter,
  - If the state is `10` (left-switch up, right-switch down) modify the minute counter,
  - If the state is `11` (both switches up) modify the hour counter
- The value to be updated is defined in binary by state of the six right-most switches (the 6 least significant switches)

Example: If the two most significant switches are set to `10` (binary for 2), selecting the middle pair of displays (for minutes), and the least significant switches are set to `001111` (binary for 15), then the minutes should be updated to 15.

Refer to the following figure for a visual explanation.

Note: When running your code on the DE10-Lite board, verify that you can set the seconds, minutes, and hours correctly.



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.

- How many “seconds” in theory have passed after the start sequence developed in part (d)?
- In the generated assembly code, in which RISC-V register will the return values from functions `get_btn` and `get_sw` be placed in. You should be able to answer this question without debugging the generated assembly code.

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 `labmain.c`, in the `labinit` function, add code to initialize Timer for timeouts every 100 ms (that is 10 time-outs per second). Remember that the DTEK-V board frequency is 30 MHz. .
In file `labmain.c,`, change the `main` function so that it never calls `delay`. Instead, insert code in the `main` function to test for 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: The timer is memory-mapped to address 0x04000020-0x0400003F. Look up the timer documentation (on the Canvas homepage) to find out how to program the timer.

Note: Don't let the call to `get_sw` 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 `labmain.c`, add a global counter `timeoutcount`. In function `main`, 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.

- 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?
- 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 BTN1 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.

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 `labmain.c`, add the following line near the beginning of the file:

```
    int prime = 1234567;
```

c) In file `labmain.c`, change the `main` function to look like this:

```
int main ( void ) {
  labinit();
  while (1) {
    print (”Prime: ”);
    prime = nextprime( prime );
    print_dec( prime );
    print(”\n”);
    }
}
```

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

```
void handle_interrupt ( unsigned cause ) {
  display_time (mytime);
  tick (&mytime);
}
```

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

e) In file `labmain.c`, add code into the function `handle_interrupt` to acknowledge interrupts from the Timer. Note that function `handle_interrupt` should include the code that is called every time an interrupt occurs. It should not include the code that initializes the interrupt routine.

f) In file `labmain.c`, change function `handle_interrupt` 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 `handle_interrupt`. 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 `boot.S`, add a function `enable_interrupt` (at the end of the file) that enables interrupts and allows interrupts from the timer (remember to make the function globally visible).

h) In file `labmain.c`, add code to the function `labinit`, that enables interrupts (see [g]), allows interrupts from the timer, and enable the timer to generate interrupts. *Note:* Before enabling interrupts globally, all other initialization must be complete.

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.

- 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?
- From which part of the code is the function `handle_interrupt` called? Why is it called from there?
- Why are registers saved before the call to `handle_interrupt`?
- Which device-register (or registers), and which processor-register (or registers) must be written to enable interrupts from the timer? Describe the functions of the relevant registers.

Assignment 4: Surprise assignment

You will get a surprise assignment at the lab session.

You must finish the surprise assignment during the session.

The surprise assignment is demonstrated together with Assignment 3.

Version history

1.4: Ported to new DTEK-V board. 2024-06-30

1.3: Clarified and condensed questions, and specified the parts explicitly. 2016-01-13.

1.2: Updated assignment 3e. Clarified the meaning of function `user_isr`. 2015-10-06.

1.1: Updated assignment 1d, specifying that `pic32mx.h` not should be used, 2015-09-29.

1.0.1: Added KTH logotype, 2015-09-28.

1.0: First released version, 2015-09-28.