Chapter 3 Peripherals

A typical embedded system continuously communicates with its environment. In the previous chapter we assumed that data provided by sensors[1] and switches[2] are read by a (dummy) function call. In this chapter we will first work with timers that are important system peripherals. Then we will work with a typical technique for preprocessing input signals. We will see an example on the target platform (bare-machine i386 emulated by Qemu) and then another example on the host platform (e.g., lab's Linux Mint).

1.1 Timing

Timers are used to keep time, as their name suggests. Some hardware platforms provide several timers, other systems use system calls (software-defined). Correct timing is essential for a vast range of the embedded systems including control systems. For example, in an open-loop scenario (as discussed in section 2.1.5) many aspects of system operations are controlled by precise timing. A coffee machine, for example, may be designed to dispense appropriate amount of coffee based on the time-interval that the pump is kept on. Wrong timing results in overflow or too little coffee. A hazardous situation is when traffic lights switch. There should be appropriate amount of time between one direction receiving a red signal and the other direction receiving a green signal.

- Some timers can be set to give hardware-based or software-based interrupts. As always, polling is another solution. Moreover, in some situations only one timer (or system time) can be used to keep multiple time periods.
- When working with hardware-based timers and counters, one must pay attention to the meaning of each bit. They might not match the data-type that we prefer to use.

1.2 Bare-Machine

Copy "peripherals target" from "skeleton" directory to your local directory (we assume: userID/TDDI11/peripherals_target)

```
cp -r /home/TDDI11/lab/skel/peripherals_target /home/userID/TDDI11
```

Now let us change to the new directory and check it:

```
cd /home/userID/TDDI11/peripherals_target
ls
```

Check to see if "peripherals target" directory contains the following:

- main.c
- Makefile
- floppy.img
- mtools.conf
- makeNrun.sh

[1] Like temperature and humidity sensors.
[2] Simple ones like door lock and complex ones like fan speed selector.

Similar to Chapter 1, the source code is in “main.c”. Compiler and linker commands are listed in “Makefile”. The binary file that will be generated by this process must be placed in the floppy image file “floppy.img”. Placing the binary file into the floppy image must be done with “mcopy” command. The necessary settings for “mcopy” are defined in the configuration file “mtools.conf”. The floppy image is used to start the virtual machine “Qemu”. The overall process is described in the script “makeNrun.sh”.

A question will be asked by “mcopy” about what to do with the existing binary file. This is the skeleton binary that you must overwrite with your newly generated binary file. Select “o”.

The source code (main.c) consists of four sections. The code works with timing and main system counter. In this section, the assignment is to read and analyze the code. Understanding the datatypes and used functions is needed.

1.2.1 Assignments

1. Why “stopTime” is defined as “DWORD32”?

2. Why “finalCount” is defined as “QWORD64”?

3. What does “Now_Plus()” do?

4. What does “CPU_Clock_Cycles()” do?

5. What does “x” (or similarly “xM”) represent?

6. What does “Milliseconds()” do?

7. Compare numbers printed by section 2 and sections 3 of the code

    7.1. Are they different?

    7.2. Should they be similar?

    7.3. How do you explain that these two sections may report different values?

    7.4. Which approach (section 2 or 3) is more appropriate? Why?

8. Section 4 prints two different numbers. What causes their difference?

9. Consider an operation (e.g., multiplication, addition, …) and approximate its execution time. A technique is to perform multiple of the selected operation in a loop and divide the overall time by the number of executions it had in the loop.

1.2.2 Demonstrations

Run the program and show the code to the lab assistant. Briefly discuss the answers to the above questions. Be prepared to answer questions that evaluate your understating of the matter.

Run the code for assignment 9 and show the source code to the lab assistant.

1.2.3 Deliverables

- Answers to the above questions
- The code that approximates the execution time (assignment 9).

Email them to your lab assistant. Write in the subject: TDDI11 Chapter 3.

1.3 Preprocessing

Many input devices might not provide a very stable readout, especially in an embedded system where inexpensive hardware is used. Even a typical push button will not return always "0" and only one "1" when one presses it only once. This is called bouncing. So if the embedded system views every "1" readout as an activation, it will receive many activations when the button is pressed only once. You can read more on this, by searching online for "push button bouncing". The solution is sometimes called "debounce".

The sensors output might be changing rapidly due to either noise or their (unnecessarily) high resolution. For example, an ambient light sensor might react to a fly flying by. It will be annoying for user if the screen brightness changes with such minor fluctuations. Therefore, in many cases the data from sensors are directly sent to a filtering mechanism (low-pass/averaging) before being used. We did not do so in the previous chapter to keep it simple. The results could be, for example, that the humidity warning light goes on and off like it is blinking randomly when the humidity is around 60%. This will look like the system cannot make its mind. A simple filtering technique is moving average. It is basically an averaging over consecutive readout samples. We will use it in an example in which the system tries to make its minds over mouse movements being major or minor.

1.4 Host-Machine[3]

Copy "peripherals host" from "skeleton" directory to your local directory (we assume: userID/TDDI11/peripherals_host)

```
cp -r /home/TDDI11/lab/skel/peripherals_host /home/userID/TDDI11
```

Now let us change to the new directory and check it:

```
cd /home/userID/TDDI11/peripherals_host
ls
```

Check to see if "peripherals host" directory contains the following:

- main.c

The source code (main.c) consists of three sections. The code works with mouse (pointing device) as a motion sensor in section 2 of the code. Section 3 of the code is timing related and simulates a blinking light. In this section, the assignment requires reading and analyzing the code. Understanding the datatypes and used functions is needed. Afterwards, we run the code, analyze the output, and improve some constant values in the code.

[3] This may not execute correctly on other computers that are not provided in the lab rooms.

Compile, run, and exit:

```
gcc main.c -lm -lX11
./a.out
ctrl + c
```

1.4.1 Assignments

One objective here is to design a system that can make its mind about mouse movements being large or not. Un-comment the lines that print "horizontalPosition", "verticalPosition", "horizontalMotion", "verticalMotion", and "combinedMotion". Move the mouse, observe the results, and pay attention to the range of values returned by mouse. This gives us an idea about the appropriate value for "motionThreshold".

The system must not react to small mouse movements or to sporadic and seldom large movements. The following line of code does it:

```
combinedMotion = sqrt(pow((double)horizontalMotion,2.0) + pow((double)verticalMotion,2.0));

motionMetric = alpha * combinedMotion + (1 - alpha) * motionMetric;
```

The term "`sqrt(pow((double)horizontalMotion,2.0) + pow((double)verticalMotion,2.0))`" is used to indifferently combine horizontal and vertical mouse movement. It is a Euclidean metric. Pay attention that negative values will have the same effect as positive ones.

The suggested range for "alpha" is between 0 and 1. Change the value of "alpha" and observe the difference it makes in the system response to the mouse movements.

Section 3 in the code simulates a blinking light by printing "X" for on and "." for light being off. The rate is controlled by "blinkHalfCycleSeconds". Too high blinking rate makes the light look like being always on. Too low blinking rate makes it difficult to realize in a glans that the light is blinking. Change "blinkHalfCycleSeconds" and find an appropriate rate.

Answer to the following questions with no more than a few sentences:

1. Assume a circle around the current mouse position. Will any straight-line movement that moves the mouse from its current position at the center of the circle to arbitrary points on the circle have the same effect on the "motionMetric"? Why?

2. Name an alternative for the Euclidean metric that we used (i.e., $\sqrt{h^2 + v^2}$). Briefly explain how it works.

3. What are advantages and disadvantages of Euclidean metric?

4. Modify the code, compile, and run it for alpha equal to 0.01, 0.5, and 0.99. Move the mouse around with minor and major movements and analyze the system behavior. You may find it helpful to uncomment some of the lines in "main.c" to print out more details. Name a difference that each one of these alpha values causes in the system behavior. Choose an alpha value that you think is useful.

5. (Optional): Name an alternative for the moving average filtering mechanism that we have used (i.e., $y_n = \alpha \times x_n + (1 - \alpha) \times y_{n-1}$). Briefly explain how it works.

6. (Optional): How does a typical push-button "debounce" work?

Modify the code with your chosen values for "alpha", "motionThreshold", and "blinkHalfCycleSeconds".

Optional assignment: Modify the code with alternative way of combining horizontal and vertical movements (instead of Euclidean metric). This is related to question 2 above.

Optional assignment: Modify the code with alternative filtering (instead of moving average). This is related to question 5 above.

1.4.2 Demonstrations

Explain why the values you picked for "alpha", "motionThreshold", and "blinkHalfCycleSeconds" are reasonable. Do this by running the modified program. Small mouse movements or sporadic and seldom large movements must not trigger motion detection. Be prepared to answer questions that evaluates your understating of the matter.

Briefly discuss the answers to the above questions.

Optional assignment(s), if you decided to do it (them).

1.4.3 Deliverables

- Answers to the above questions
- The code modified with values you picked for "alpha", "motionThreshold", and "blinkHalfCycleSeconds".
- (Optional): The code with alternative way of combining horizontal and vertical movements (instead of Euclidean metric).
- (Optional): The code with alternative filtering (instead of moving average).
- Feedback questionnaire

Email them to your lab assistant. Write in the subject: TDDI11 Chapter 3.

1.5 Resources[4]

An educational outcome of these labs is that the students find resources, online. Therefore, please try to search for the relevant information on the internet.

For convenience, you may also use the following:

- Course on operating systems, TDIU25, 2nd lecture
- https://www.ida.liu.se/~TDDI11/labs/pdf/libepc_doc.pdf

[4] Concepts visited in this chapter are simple and widely available on the web