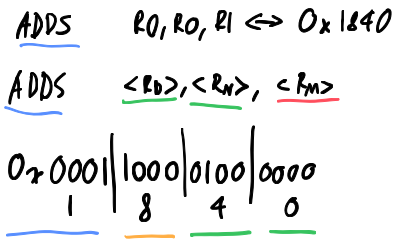


PCT Lab: Bit Manipulations

1 Introduction

In this lab you will learn to use different techniques for the manipulation of individual bits in registers. You will use these to read dip switches and buttons, as well as to write out values. Specifically, you will detect events on the four buttons T0 – T3.

2 Goals

- You can apply the bitwise operators in C for setting and clearing individual register bits.
- You can explain the purpose of an edge-detection and are able to implement it in C.
- You know how to use the bit shifting operators in C.

3 Tasks

3.1 Task – Control individual LEDs

Download the given project frame. Expand the program so that LED7 and LED6 are always on (bright) and LED5 and LED4 are always off (dark). As a result LED3 to LED0 will be controlled by the settings of dip switches S3 to S0 whereas the settings of the 4 leftmost dip switches will be ignored.

Tip: Define (`#define`) a mask for the “bright” as well as for the “dark” bits and use the assignment operators `|=` and `&=` to set/clear the corresponding bits.

Verify the correct behavior with different positions of the dip switches.

3.2 Task – How often does a button get pressed?

a) Expand the program from task 3.1. Read the state of the buttons T3 to T0 into a variable which needs to be defined. Mask the unused upper 4 bits.

   Define an 8 bit wide counter variable. Increment this variable each time the button T0 is pressed i.e. whenever Bit 0 is high. Output the variable on LED15 to LED8 during each loop iteration.

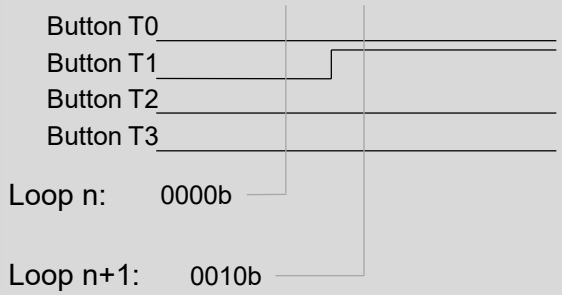
b) What do you notice when testing? By which value does the counter increase when you press the button? What is the cause?

c) Expand the program with another counter variable that counts the “push events” of the button. Count the “push-events” independent on how long the button is pressed by the user. Output the variable on LED31 to LED24.

   Implement an edge detection, which detects if one or more buttons were not pressed (low) during the last loop iteration and are now pressed (high).

- Save the value of the button state in a new variable (at the end of the loop).
- Implement the edge detection for all four buttons at the same time. You can do this by linking the button state from the last loop iteration with the state of the current loop iteration. Use the operators for the one's complement (~) and bitwise AND (&).

Example: pressing button T1




```
n:          0000b → 1111b
n + 1:            → 0010b
AND                 0010b
```

→ Button T1 pressed

- Afterwards use a bit mask to only consider button T0.

3.3 Task – Additional functions when you press a button

Expand your program further. Depending on the button pressed one of the following actions shall be executed on a variable of type **`uint8_t`**.

Button T3: The variable shall be assigned the value set on the dip switches S7 to S0.

Button T2: The value of the variable shall be bit-wise inverted (one's complement).

Button T1: The value of the variable shall be shifted by one bit to the left (**<<** operator).

Button T0: The value of the variable shall be shifted by one bit to the right (**>>** operator).

In the case where more than one button is pressed simultaneously only one action shall be carried out. Continuously output the value of the variable on LED23 to LED16.

3.4 Task (optional)

Change the program from task 3.3, such that by pressing button T2 only the bits at the position 5 to 2 get toggled (bitwise inversion). The other bits are left as they are.

Tip: You can toggle a bit with the XOR operation (**^**). Create a mask in which the corresponding bits are set (1) and the others are cleared (0).

4 Final program

In Figure 1 you see all the in- and outputs of all the subtasks.

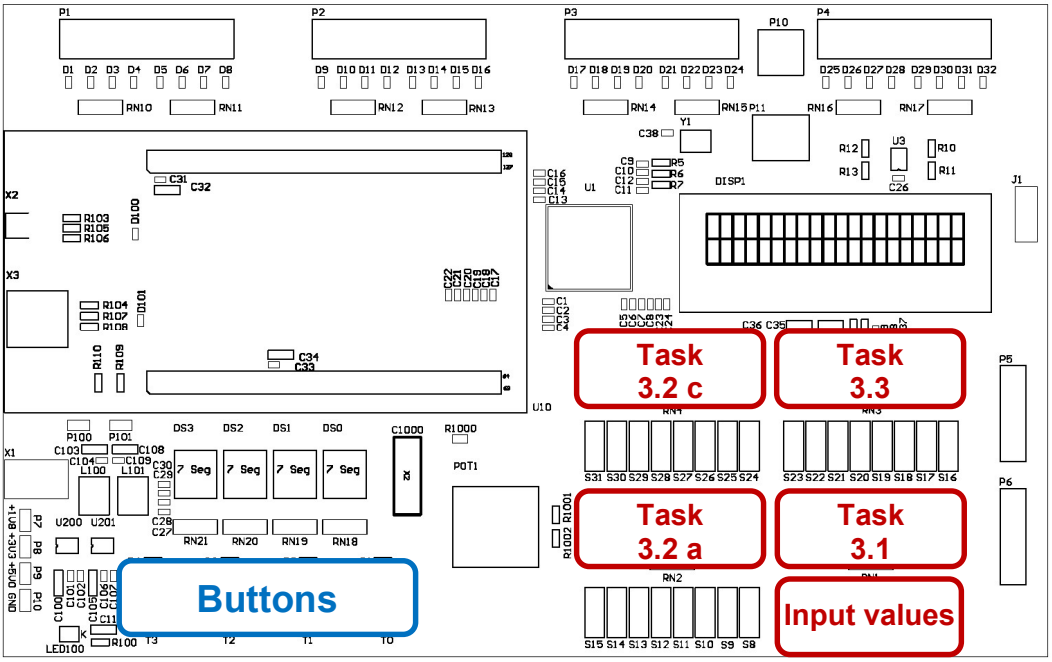


Figure 1: In- and output

5 Grading

The working programs have to be presented to the lecturer. The student has to understand the solution / source code and has to be able to explain it to the lecturer.

Criteria	Weight
“Control individual LEDs” meets the required functionality.	1/4
“How often does a button get pressed?” complies with the required functionality in part a). You can explain the cause of the behavior in part b).	1/4
“How often does a button get pressed?” uses in part c) the described method to detect edges.	1/4
“Additional functions when you press a button” meets the required functionality.	1/4