Sebastien Cantin Lab 2 Report ECSE 324
#260979759 Fall 2021

Part 1

Brief description

The task was to write several drivers for the two given IO devices (pushbuttons and 7 segment hex displays) and to write a program that would print the hexadecimal number in the first four switches to the hex display corresponding to the push button that was de-clicked.

The hex display needed a driver to clear individual bit, flood them or write a specific hexadecimal number to them.

The push buttons needed a driver that would read the value of the buttons currently pressed, one that would check the values of the pushbuttons that had been de-clicked and one that would clear the register that held those values. Finally drivers were needed that would enable and disable interrupt requests from particular push buttons.

Approach taken

Clear and flood drivers were needed for the hex display and both used a very similar logic. They check each bit on the one hot encoded index register and bitclear or or the value in the hex's display. The write function is very similar for the hex displays except it also has to find which value it is going to display in the hex

The drivers for the pushbutton were very simple to write, it was simply loading the register from the IO device and reading it or clearing it.

The code for the actual program was not that complicated. The edge capture register of the push buttons was constantly polled at the same time as the value from the switches was read. The value of the switches was constantly written to the LEDs. When the value was something other than 0 for the push button edge capture register, the value from the switches would be written to the hexes in the indices that the pushbuttons had been de-clicked on.

Challenges faced

It was challenging at first to understand how to interact with some registers in order to clear them, but eventually it became clear and easier to understand. Not many problems were faced writing the program itself.

Possible improvements

Polling is not an efficient was of making a computer work. This is a very simple program and using interrupts would allow the cpu to do more important tasks than just constantly checking the values of IO device registers. The write, flood and clear hex drivers could all use improvement, a loop could be made instead of simply copy pasting the code 6 times, but it was not a proper use of time at the point of writing the code since it was very complicated breaking the loop after the 4th iteration to store the value and then load the new value and change the address to store at.

Part 2

Brief description

The task was to write several drivers for the timer and write two programs, one which simply cycled from 0 to 15, iterating after each second and displaying the number on the LEDs and the first hex display. The second function was a stopwatch that had a button that started counting, one that stopped and one that reset the value displayed on the hexes

The drivers needed for the timer were a configure timer, which stored arguments in the load register and control registers, a read driver, which loaded the condition of the interrupt status and a clear, which cleared the interrupt status.

Approach taken

For the first program, the approach was rather simple, always read the timer, if it is not 1, branch back to the top, otherwise add one to the register holding the count value, if the count value was greater than 15, set it back to 0, and then print the new value on Hex0.

The second program was far more complicated, instead of having just one main loop, there were two. One for the stop state and one for the on state, the reset was just a label inside of stop. In stop, the push button edge captures were checked and if they had the value 1, the program branched to start, otherwise it either branched back to stop or continued to reset depending on the value. In the start loop, it constantly checked the timer interrupt status, if it was not 1, it would branch back to start. Then the push button edge captures were checked and if they were 2 or 4, the program branched back to stop or reset. All the values to be printed one the hexes were held in 1 register, so there was a subroutine named increment which added 1 to the count and checked all the bits to ensure they were in the correct range for the hex they represented. Then there was a write_sequence subroutine that grabbed each bit in the count register and made its corresponding hex write it. The program then branched back to start.

Challenges faced

There was trouble the first time trying to print the count value the first time, because the program was just trying to print the whole value in all the hexes instead of each value being printed to its corresponding hex, so a write_sequence was made that did an and with the byte needed and right shifted it so that it would be in the correct bit then printed it. When reading the lab instructions, there was a slight lack of clarity regarding how the clear interrupt status was supposed to work. Inside it, it was calling the read interrupt status and then storing the value after clearing it, instead of just using the value that was loaded in register 0 already from running it before running the clear. The result was that the timer was operating at two thirds of its speed since sometimes two interrupt flags were essentially cleared in one clear.

Possible improvements

Polling is not an efficient was of making a computer work. This is a simple program and using interrupts would allow the cpu to do more important tasks than just constantly checking the values of IO device registers. The beginning of the start loop could also be improved, since technically after a push button is de-clicked, it would take up to ten milliseconds to read the edge capture register since the interrupt status of the timer had to be 1 in order to read the push buttons

Part 3

Brief description

This program is another version of part 2.2. It is a very similar stopwatch except for the fact it uses interrupts and deals with the interrupt requests. This program started with a lot of given code however there were a couple things to change, and then the code to make the stopwatch had to be added as well.

Approach taken

The first instructions were to add a configure timer and enable push buttons in the _start label, however the enable push buttons were not necessary as that code was already included in the given code. The code to determine which service request was occurring also had to be updated to deal with two different types of service requests instead of just the push buttons. A service request had to be written for the timer that wrote the value of the interrupt status in the tim_int_flag in memory and then cleared the interrupt status request. The push button service also had to be rewritten to do the exact same thing except store it in the pb_int_flag. The service request for the timer also had to be configured, which was simply just copying the three lines of code that configured the push buttons and changing the value to that of the timer. Finally most of the code from 2.2 could be copy pasted with a few changes, mostly just changing the polling from the edgecp register and interrupt status to the two flags in memory.

Challenges faced

A very simple mistake was made, the timer service request was storing its value inside the push button flag, and it was a very, very challenging mistake to find, because it was not in a part of the code occurring often visibly inside the disassembly, but after a lot of wasted time and tampering it was found and fixed. Another problem was faced when reading the push buttons, the program used to clear the push button flag, but that would result in lost button clicks, since there were a couple extra checks of the flag that did not result in the correct transition occurring.

Possible improvements

This program did not actually use interrupts properly, it was still a polling machine except instead of polling the I/O devices directly, it polled flags in memory that held the values from the I/O devices. A better way to do it would be to continue having the push button values in a flag, however incrementing the values on the hexes ought to occur inside the timer interrupt. Inside the push button interrupt, disabling and enabling the timer from sending interrupt statuses could even make it more efficient depending on what state the stopwatch was supposed to be in.