Tyler McKean README for Project 2 GPIO Programming in Assembly

BASIC STRUCTURE

This program utilizes the joystick of the STM32L476 Microcontroller as an input to do the following task:

- Toggles both the onboard RED and GREEN LEDs when the up button is pressed.

To achieve this task, the Clock was enabled to GPIO Ports A, B, and E by using a pre-index load of the peripheral clock enable register and executing an ORR expression with the value inside the register, which enables it. The Up bottom of the joystick is configured to PA.3 of Port A, with the Red and Green LEDs tied to PB.2 and PE.8. For the Up button to toggle the LEDs, GPIO Ports A, B, and E would all need the MODER, and OTYPER to be enable for each pin and the PUPDR register would need to be configured to pull pin PA.3 down to ground.

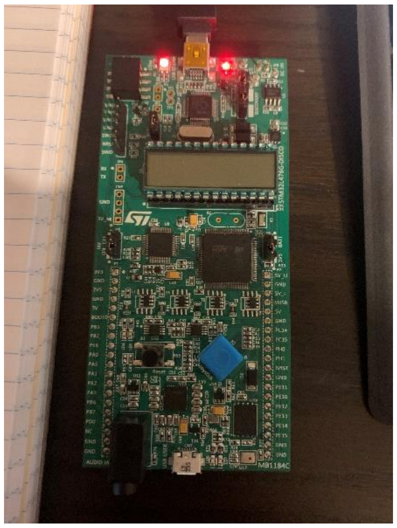
I chose to initialize all the registers for each individual Port sequentially to keep the code blocks more organized. I followed a repetitive process where I would pre-index the value within the GPIO MODER, OTYPER, and for GPIOA, the PUPDR and would store the value into a register. From there, I used a BIC expression to clear the bits I needed to initialize for these registers. For GPIOA, the MODER needed to be set as an input, which involved clearing bits 6 and 7. The OTYPER was set as an output push-pull by clearing bit 6 and in the PUPDR bits 6 and 7 were cleared and then configured as Pull-down by an ORR expression with the value 0x02. After the clearing and configuring of each register, I stored the values following the assembly operations back into each respective register. For the LEDs, I only needed to configure the MODER and OTYPER for pins PB.2 and PE.8. This was done in a similar manner: I loaded both the MODER and OTYPER into a register, r0, pre-indexed the value within it to another register, r1, and used a BIC expression to clear bits 4 and 5 for the Red LED and bits 16 and 17 for the Green LED. Both pair of bits were executed with an ORR expression to configure them as general-purpose outputs. Inside the OTYPER, I cleared bit 2 for the Red LED and bit 8 for the Green LED in order to configure them as an output push-pull. All the resulting values in the MODER and OTYPER were then stored back into their addresses. With all these registers enabled and configured, all the pins were properly initialized, and I needed to build a loop that would be able to toggle the LEDs when the up button was pressed.

I used two different labels for the loops that would toggle both LEDs when the up button was pressed. These loops were broken down into two sections with the labels, “loop1” and “loop2”. For loop1, the base address of GPIOA was loaded into a register, r0 and then the values stored in the IDR of GPIOA were pre-indexed into another register, r1. The values stored into r1 were then combined with an AND expression with a hex value of 0x08, which corresponds to Pin 3 in the IDR. A comparison instruction, CMP, was then executed to check the values stored in r1 with the hex value 0x08. A branch if not equal instruction, BNE, followed the comparison instruction and the code would branch to a label “endloop1” if the comparison was true. If the comparison is equal, the code continues by loading both the base addresses of GPIOB and GPIOE into two registers, r0. Both ODRs for each were loaded into registers, r1, and in order to toggle the LEDs, an EOR expression was executed with r1 and the corresponding hex value for both PB.2 and PE.8. For PB.2, the values in the ODR and a hex value of 0x1<<2 were executed with an EOR expression and then stored the resulting value back into the ODR for GPIOB. For PE.8, the values of the ODR and a hex value of 0x1<<8 were executed with an EOR expression and the resulting values were stored back into the ODR for GPIOE. A second loop labeled “loop2” was then written into the code in order to sustain the program. Again, the base address of GPIOA was loaded into a register, r0 and then the values of the IDR were pre-indexed into another register, r1. Another AND expression with the hex value 0x8 followed, and another comparison instruction was established. The comparison was with the values of the IDR in r1 and a hex value of 0x8 and a BEQ instruction, branch if equal, was written to loop2. If this comparison was true it would loop back to the label “loop2” and keep cycling through this loop. If the BNE instruction in loop1 was true or if the BEQ instruction in loop2 was false, the program would jump to the label “endloop1” and then branch back to loop1, which also continues to run the assembly program in an infinite loop. The __main PROC block ends after this segment of code, indicated by the ENDP directive. I did not establish any variables in my DATA block area, because this whole project could be accomplished by just coding inside the CODE block area.

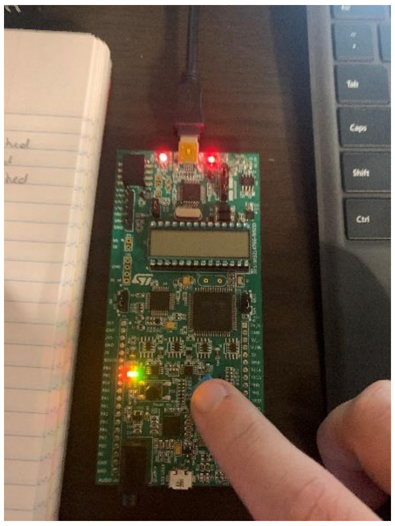
HOW TO TEST

- To test this project, simply plug the USB cable into the STM32L476 into the computer and run the project file in Keil µVision 5.
- After a successful build, click the debug option and press the reset button on the discovery board
- From here, both LEDs will be off. If you press the up button of the joystick on and off, the RED and GREEN LEDs will toggle.
- Doing these action will prove that the project works correctly.

TEST PROCEDURE



After building the project file and clicking the debug button in Keil uVision the discovery board’s LEDs will turned off.



Clicking the up button of the joystick will turn on the RED and GREEN LEDs, clicking again will turn them off.

TEST COMPLETE