RMIT University

School of Engineering

EEET2096 – Embedded Systems Design and Implementation

Lab Experiment #2

The ARM parallel I/O ports

Lecturer: Dr Jidong Wang

Tutor: Tuan Phan

Students: Jaime De Esteban Uranga s3762844,

Jasmine Rehal s3541157, Traratuch Wattanatorn s3610249

(Identical Report)

Group: Tuesday 12:30 – 14:30

Submission Due Date: Sunday 14/04/19 23:59

Introduction:

There are 2 main purposes of laboratory 2, which are to configure the general purpose input and output of the ARM microcontroller in order to correspond to the circuitry that is accessible on the Keil ARM EVB and to write a program that monitors the input switches as well as controls the LEDs (light emitting diodes) on the EVB. The ARM EVB was to be used for the laboratory as it has a set of LEDs, which are wired to bits 8 to 15 of Port E and there are 3 push button switches that are labelled User (PB7), Tamper (PC13) as well as WakeUp (PA0). These elements were utilised in order to simulate on and off inputs from the user [1].

Preliminary:

1. Part of this laboratory will require turning the LEDs on and off at a visible rate. From your results to Questions in Lab 1 estimate the initial value of R0 in the following code fragment to give a visible delay.

In order to modify the base code that is given with the laboratory and to ensure that the LEDs have a delay that is noticeable to a human, it was required to understand how many instructions are being run by the processor every second. Since the loop consists of two instructions, being the "adds" and the branch instruction, and knowing that the processor runs approximately 6 million instructions a second (6.059 Mhz), it can be estimated that with an initial value of 1 million, it will take the processor around 0.5 seconds to run the delay program.

Knowing this, we can construct our code to make sure our delay is correct. For this, the value of 1.000.000 was used in Hex code, which converts to 0xF4240.

```
DELAY           EQU        0xF4240

Delay
     LDR R0,=DELAY
Loop
     ADDS R0,#-1 ;note minus 1
     BNE Loop
```

When this delay is checked using the Mixed Signal Oscilloscope (MSO), the time between the LEDs turning on and off can be accurately measured.

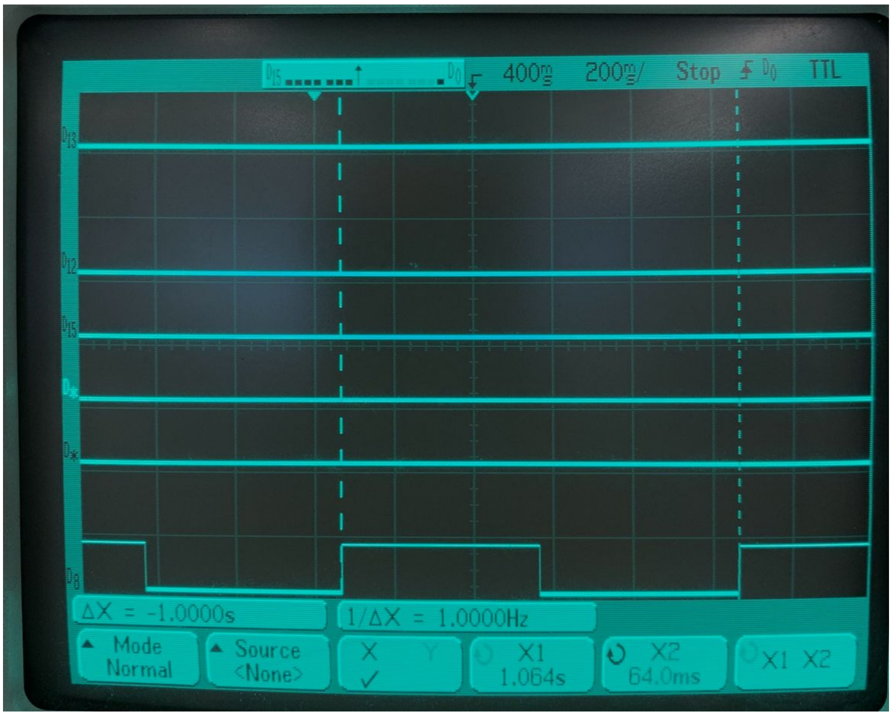

Figure 1. Snapshot of the MSO displaying the time between the LEDs turning on and off.

2. From reference 3 STM32F10xxx Cortex-M3 programming manual read section 3.3.3 and document the restrictions on the value of **constant*****. The USER switch in this laboratory is wired to GPIOB. This will require GPIOB to be configured. (See preliminary 4)***

According to Table 4, it can be observed that the amount in bytes for options is 16, which means that the maximum amount of options the MOV instruction can have is 16B.

Table 4. Flash module organization (low-density devices) (continued)

Block	Name	Base addresses	Size (bytes)
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash memory interface registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

Table 1. Flash module organisation of low-density devices.

3. Complete tables, which are the clock enable and Input/Output register [2].

Port	Base Address	Clock enable in RCC_APB2ENR (address 0x40021018)
GPIOA	0x40010800-0x40010BFF	IOPAEN (Bit 2)
GPIOB	0x40010C00-0x40010FFF	IOPBEN (Bit 3)
GPIOC	0x40011000-0x400113FF	IOPCEN (Bit 4)
GPIOD	0x40011400-0x400117FF	IOPDEN (Bit 5)
GPIOE	0x40011800-0x40011BFF	IOPEEN (Bit 6)

Table 2. GPIO Base Address and Clock enables.

Register	Mnemonic	Address offset
Configuration Register Low	GPIOx_CRL	0x40011000 OR 0x00
Configuration Register High	GPIOx_CRH	0x40011004 OR 0x04
Input data register	GPIOx_IDR	0x40011008 OR 0x08h
Output data register	GPIOx_ODR	0x4001100C OR 0x0C

Table 3. Input output registers.

4. Using the code provided in section 16.3.9 as a guide read through the chapter on the ARM_IO and then develop the code to configure the pin/port containing the user switch.

In order to configure the input pin for the user button, it was needed to set the required bits in the configuration register in `GPIOB_CRL`., since our user button is in the lower bank of pins in the port E.

It was then required to set the button either to input-input floating or input-input pull up. In this case, the input floating was choses, as for this lab the results are the same and it produces a cleaner result in the Oscilloscope.

It was required to write 0100 (0x4) in the `GPIOB_CRL` for pin 7, which is the MSB in the register. For that, the code to set the pin is shown below as:

```
mov r6,#0x40000000       ;configure port B - input user switch
ldr r7,=GPIOB_CRL
str r6,[r7]
```

Table 20. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General purpose output	Push-pull	0	0	01 10 11 see Table 21		0 or 1
	Open-drain		1			0 or 1
Alternate Function output	Push-pull	1	0			don't care
	Open-drain		1			don't care
Input	Analog	0	0	00		don't care
	Input floating		1			don't care
	Input pull-down	1	0			0
	Input pull-up					1

Table 21. Output MODE bits

MODE[1:0]	Meaning
00	Reserved
01	Max. output speed 10 MHz
10	Max. output speed 2 MHz
11	Max. output speed 50 MHz

Table 4. Port bit configuration table and Table 5. Output MODE bits.

5. Develop the code to implement the flow chart of figure 16.13. Note in the procedure this code will inserted after the LEDs have been turned on. The LEDs will remain on when the switch is pressed.

In order to run the code given in 16.13, it was needed to modify the original code to read the button "User".

To do this, firstly the clock in port B was enabled, since our button is pinned to Port B pin 7. For that we run the code shown below as:

```
mov r6,#0x48          ;enable clock to port E and B
ldr r7,=RCC_APB2ENR
str r6,[r7]
```

We use 0x48 because we are enabling the clock to port E and B at the same time. Then we are going to configure the button as a floating input. For that we write into `GPIOB_CRL`, with the exact configuration.

```
mov r6,#0x40000000       ;configure port B - input user switch
```

```
ldr r7,=GPIOB_CRL
str r6,[r7]
```

Once we have set the button correctly we need to read the button, for that we will need to read `GPIOB_IDR` and read bit 7 as it is the one corresponding to our enabled pin.

```
ldr r6,=GPIOB_IDR
ldr r7,[r6]
```

With all of that set, we can assembled our final program.

```
GPIOE_CRH       EQU   0x40011804
RCC_APB2ENR     EQU   0x40021018
GPIOE_ODR       EQU   0x4001180c
DELAY           EQU        0xF4240
GPIOB_IDR       EQU        0x40010c08
GPIOB_CRL       EQU        0x40010c00
     AREA RESET, DATA, READONLY
     EXPORT __Vectors
__Vectors
                      DCD 0x20000200
                      DCD Reset_Handler

     AREA |.text|, CODE, READONLY
     EXPORT Reset_Handler
;
Reset_Handler PROC
     mov r6,#0x48          ;enable clock to port E and B
     ldr r7,=RCC_APB2ENR
     str r6,[r7]
     mov r6,#0x33333333    ;configure port E - o/p to LEDs
     ldr r7,=GPIOE_CRH
     str r6,[r7]
     mov r6,#0x40000000      ;configure port B - input user switch
     ldr r7,=GPIOB_CRL
     str r6,[r7]
     ;
ledON
     mov r6,#0xff00        ; turn LEDs on
     ldr r7,=GPIOE_ODR
     str r6,[r7]
```

```
Delay
	ldr R0,=DELAY
CLoop
	mov r6,=GPIOB_IDR
	mov r5,#0x80
	ldr r7,[r6]		;reading switch press
	tst r5,r7	;AND r5 r7, checking for button press
	beq Loop
	ADDS R0,#-1 ;note minus 1
	BNE CLoop
	mov r6,#0x00; ;turn LEDs off
	ldr r7,=GPIOE_ODR
	str r6,[r7]
	ldr R0,=DELAY
CLoop2
	ADDS R0,#-1 ;note minus 1
	BNE CLoop2

Loop
	b Loop
	ENDP
	END
```

With this simple program we can follow the chart, making the LEDs stay on if the button is pressed.

Procedure:

In order to make the Figure 16.14 program, it was required to divide the LEDs into two different groups, and depending on registering a button press we will call a function to flash the top LEDs or the bottom LEDs.

The set up is exactly the same as the previous code, we enable clock to the port B, set up the button as a floating input, and then create our code.

```
GPIOE_CRH	EQU	0x40011804
RCC_APB2ENR	EQU	0x40021018
GPIOE_ODR	EQU	0x4001180c
DELAY		EQU	0xF4240
GPIOB_IDR	EQU	0x40010c08
GPIOB_CRL	EQU	0x40010c00
	AREA RESET, DATA, READONLY
	EXPORT __Vectors
__Vectors
			DCD 0x20000200
			DCD Reset_Handler
```

```
      AREA |.text|, CODE, READONLY
      EXPORT Reset_Handler
;
Reset_Handler PROC
      mov r6,#0x48              ;enable clock to port E and B
      ldr r7,=RCC_APB2ENR
      str r6,[r7]
      mov r6,#0x33333333        ;configure port E - o/p to LEDs
      ldr r7,=GPIOE_CRH
      str r6,[r7]
      mov r6,#0x40000000        ;configure port B - input user switch
      ldr r7,=GPIOB_CRL
      str r6,[r7]

CLoop
      ldr r6,=GPIOB_IDR
      mov r5,#0x0000F710
      ldr r7,[r6]             ;reading switch press
      cmp r5,r7               ;AND r5 r7
      beq flashFour
return
      mov r6,#0xf000          ; turn upper 4 LEDs on
      ldr r7,=GPIOE_ODR
      str r6,[r7]
      bl delay
      mov r6,#0x0000          ; turn upper 4 LEDs off
      ldr r7,=GPIOE_ODR
      str r6,[r7]
      bl delay
      b CLoop
flashFour
      mov r6,#0x0f00          ; turn upper 4 LEDs on
      ldr r7,=GPIOE_ODR
      str r6,[r7]
      bl delay
      mov r6,#0x0000          ; turn upper 4 LEDs off
      ldr r7,=GPIOE_ODR
      str r6,[r7]
      bl delay
      ldr r6,=GPIOB_IDR
      mov r5,#0x80
      ldr r7,[r6]             ;reading switch press
      tst r5,r7         ;when button is pressed, GPIOB_IDR will be
0x80
      beq flashFour
```

```
        b return
delay
        ldr r0,=DELAY
delayLoop
        adds r0,#-1
        bne delayLoop
bx lr
Endp
```

First we need to explain the setup part of the program. What we do is enable the clock in the ports B and E of the board by writing 0x48 into `RCC_APB2ENR`, once the clocks are enabled we set up the pins accordingly. We set up the LEDs as 0011 which is an General purpose output, push pull and a max speed of 50Mhz. We then set up the the input button by setting the bits to General purpose input floating.

What's happening is, there's a main loop that flashes the top 4 LEDs every 0.5s and will continue to do that forever. However, when a button press is detected, using the TST instruction which will only branch to “flashFour” when the exact pin 7 is turned to 1.

Once the branch to “flashFour” happens, the other 4 LEDs are flashed for half a second and then the button is read again, if it's still being pressed then we run “flashFour” again, but if the button has been lifted, we run the original routine and flash the normal 4 LEDs and loop the program again.

The following screenshots show the top group of LEDs flashing, the bottom group flashing and the input button where two short presses has been detected.

6.

a) MSO trace of the channel probing the input switch.
b) A second probe on one of the first group of LEDs
c) The third probe on one of the lower group of LEDs

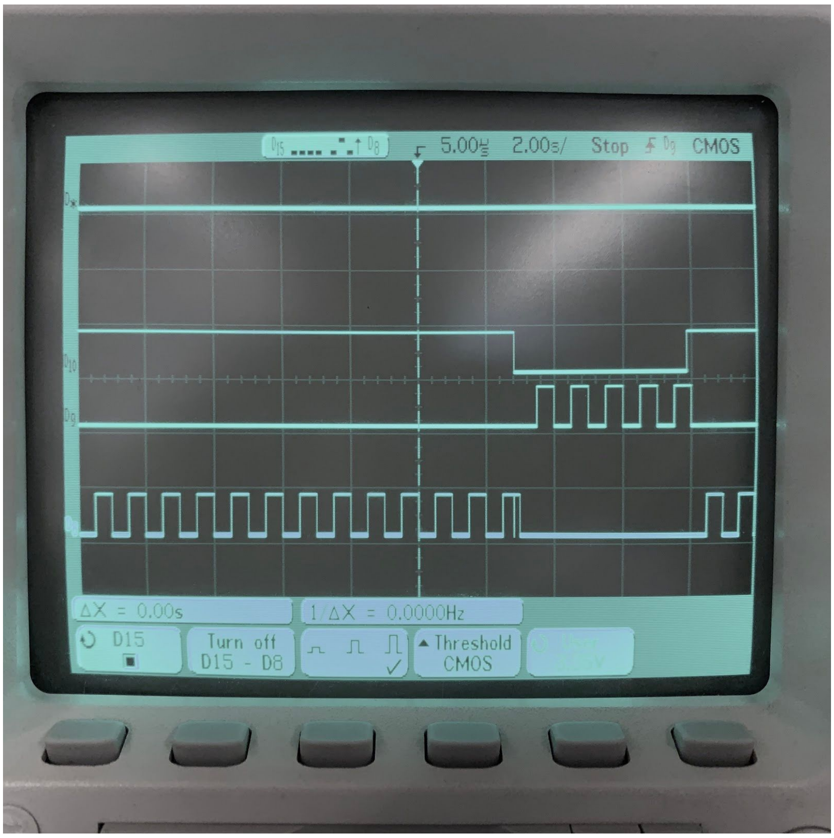

Figure 2. MSO Trace with both LED banks and Button Press.

Questions:

1. ***Often several outputs need to be wired together in an open drain/collector configuration. What is open drain and how does it work? How may the open drain be configured on one of the ARM output pins?***

An open-drain or open-collector output pin is driven by a single transistor, which pulls the pin to only one voltage (generally, to ground). When the output device is off, the pin is left floating. When the transistor is off, the signal can be driven by another device or it can be pulled up or down by a resistor. The resistor prevents an undefined, floating state. (https://www.maximintegrated.com/en/glossary/definitions.mvp/term/Open-drain/gpk/1174)

The open drain can be configured in the `GPIO` pins by writing into the corresponding register (`GPIOx_CRL` / `GPIOx_CRH`) and selecting the different configuration bits. For example for a General Purpose output can be configured by writing 0111(0x7) into the corresponding register.

Table 20. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General purpose output	Push-pull	0	0	01 10 11 see Table 21		0 or 1
	Open-drain		1			0 or 1
Alternate Function output	Push-pull	1	0			don't care
	Open-drain		1			don't care
Input	Analog	0	0	00		don't care
	Input floating		1			don't care
	Input pull-down	1	0			0
	Input pull-up					1

Table 21. Output MODE bits

MODE[1:0]	Meaning
00	Reserved
01	Max. output speed 10 MHz
10	Max. output speed 2 MHz
11	Max. output speed 50 MHz

Table 6. Port bit configuration table and Table 7. Output MODE bits.

2. In this laboratory the switch is sampled at one instant of time. If this happened in, for example a pedestrian traffic light controller, the request could be missed. What is needed is a circuit that remembers that a switch has been pressed. It is suggested that the RS flip flop might be an appropriate component. Explain how the RS flip flop could be implemented in the design of a traffic light controller.

A pedestrian traffic light controller is used to control the flow of pedestrians crossing roads. At a pedestrian crossing, when a pedestrian wants to cross the road, they are required to press a button and after their request is received by the traffic light controller, after a certain amount of time the traffic lights for the cars changes from green to red, and the traffic lights for the pedestrians becomes green.

In this case, it is important that the circuit remembers that a switch has been pressed, otherwise the pedestrian might be waiting at the pedestrian crossing for a while and they will not get a green crossing light, causing them to wait at the crossing for an unnecessarily long time. This is why an RS flip flop would be an appropriate component to resolve this issue. A flip flop is known as a bi-stable device, which has 3 classes that are latches, edge-triggered and pulse-triggered flip flops. In regards to the flip flops, ‘set’ means that the output of the circuit is 1 and ‘reset’ means that the output of the circuit is 0. The two different types of flip flops are the RS flip and the JK flip flop [3]. In regards to the RS flip flop circuit, an additional control input is applied. This additional control input controls the timings to when the state of the circuit is going to change and the additional input is the clock pulse [4]. The way in which an RS flip flop can be implemented in the design of a traffic light controller is that when an additional control input is applied, it can decide when the circuit is going to change from a red to a green light and vice versa using the clock pulse that is the additional input.

3. Your code is supposed to read the switch regularly to catch the switch pressing. Without RS flip flop, The detection of switch pressing is really an edge detection (for either up edge or down edge). What is your consideration for reading frequency (or the interval between each reading)?

It has been found using the MSO, that a short press takes approximately 188ms. Therefore it can be approximated that by checking the GPIOB_IDR in an interval less than 188ms, we can read pretty much every single button press.

The image shown below is the result of a short press on the MSO, which demonstrates the points mentioned above.

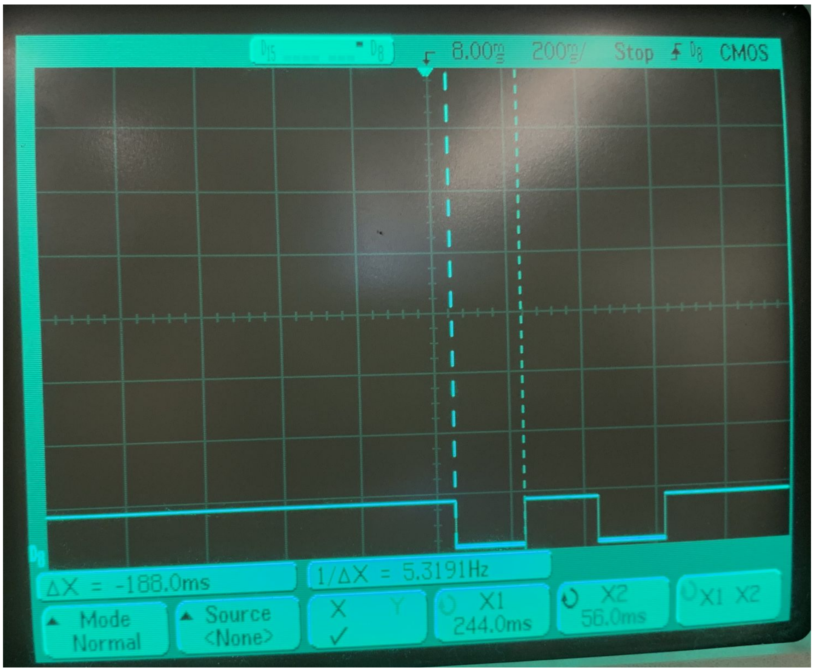

Figure 3. Snapshot of the MSO displaying the result of a short press of a button.

In our code, we are checking the input register every time the LEDs flash so we are able to recognise when the button is being pressed without the user having to hold it down for a long time, a short press will trigger the flashFour routine immediately.

References:

[1] Jidong Wang, “Lab 2_Parallel _IO.docx”, ed. Melbourne, Australia: RMIT University, 2019. [Online]. Available at: https://rmit.instructure.com/courses/49819/files/6385622/download?verifier=7ru7b7Szt7fzbVIbWy9f9K57UH0NcvHNJA996cTJ&wrap=1

[2] Procedure call standard for the ARM architecture. Ep2012 [Online at rmit]. Available at: http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042f/IHI0042F_aapcs.pdf

[3] Circuit Globe, “RS Flip Flop”, April 2019. [Online]. Available at: https://circuitglobe.com/rs-flip-flop.html

[4] Bright Engineering Hub, “Types of flip-flop circuits explained – RS, JK, D & T”, 2019. [Online]. Available at: https://www.brighthubengineering.com/diy-electronics-devices/46493-types-of-flip-flop-circuits-explained-rs-jk-d-t/