Leonardo Fusser (1946995) October 8th, 2020

Microcontroller & Microprocessor Systems

Lab 2

Day Yann Fong

PART A:

1a)

PIC16F887 Clock Modes
(1) EC – External clock with I/O on OSC2/CLKOUT
(2) LP – 32kHz Low-power crystal mode
(3) XT – Medium gain crystal or ceramic resonator oscillator mode
(4) HS – High gain crystal or ceramic resonator mode
(5) RC – External resistor-capacitor (RC) with Fosc/4 output on OSC2/CLKOUT
(6) RCIO – External resistor-capacitor (RC) with I/O on OSC2/CLKOUT
(7) INTOSC – Internal oscillator with Fosc/4 output on OSC2 and I/O on OSC1/CLKIN
(8) INTOSCIO – Internal oscillator with I/O on OSC1/CLKIN and OSC2/CLKOUT

b) The default frequency of the internal oscillator is 4 MHz

3a) The pin on the PIC16F887 that outputs the clock (CLKOUT pin) is RA6/14 (OSC2/CLKOUT).

bi) The frequency observed is around 1MHz and the peak-to-peak voltage is around 5V *(take a look at scope screenshot below)*.

ii) (Calculation based on my observation from my oscilloscope)

Scope configuration:

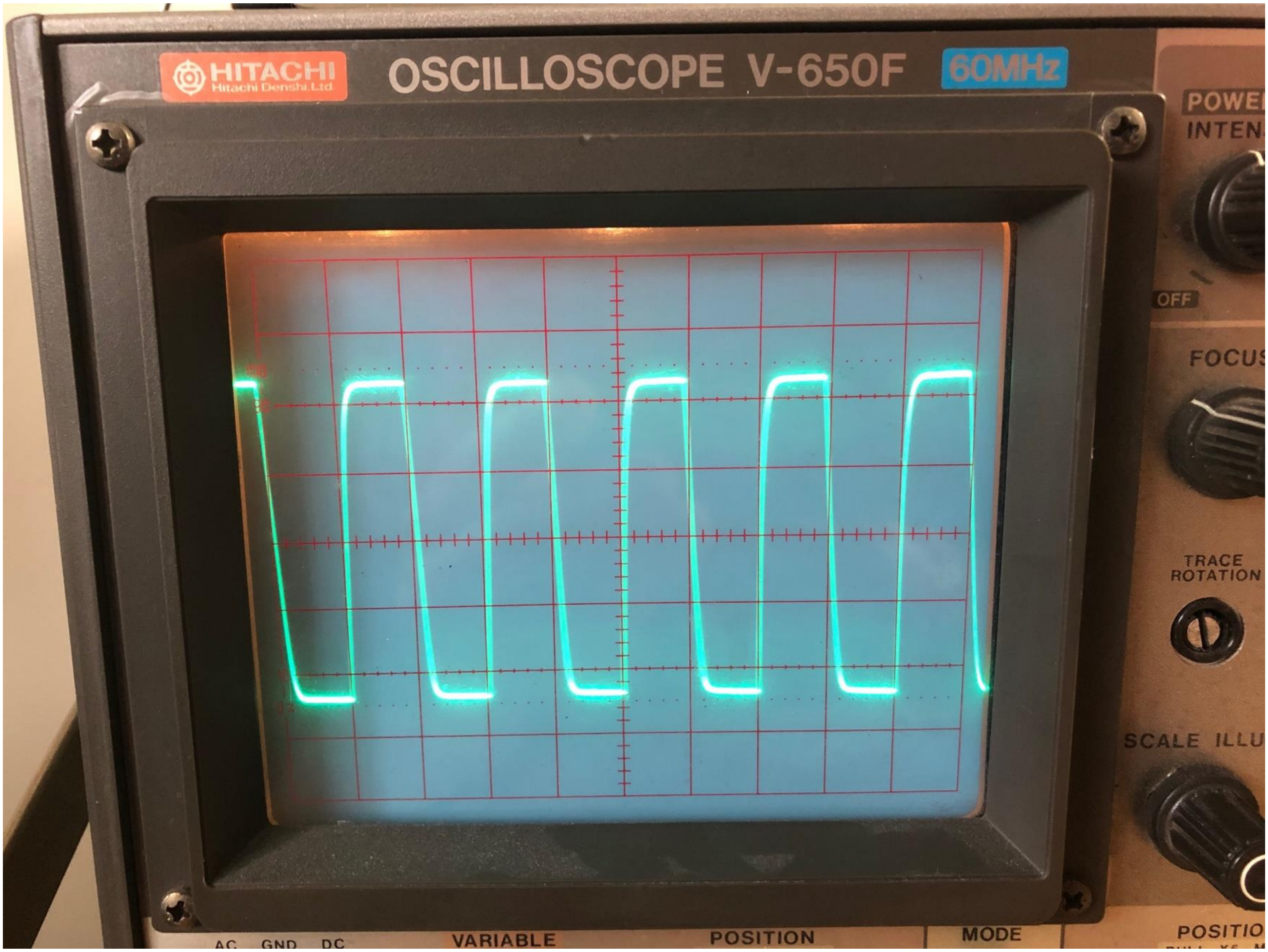
TIME/DIV: 1V per-division (there are 5 divisions)

TIME/DIV: 0.5uS per-division (there is 2 divisions)

$$V_{PP} = 5V$$

$$Frequency = \frac{1}{1uS} = 1MHz$$

These results from my calculations match my expectations based on the scope screenshot given to me in the Lab2 instructions.



Signal observed from CLKOUT (pin 14) on PIC16F887

(VOLTS/DIV: 1V & TIME/DIV: 0.5uS)

PART B:

8) Step over (F8): Executes one source line of the program. If the line is a function call, executes the entire function then stops.

Step into (F7): Executes one source line of the program. If the line is a function call, executes the program up to the function's first statement and stops.

Run to cursor (F4): Runs the current project to the cursor's location in the file and stops program execution.

Set PC at cursor: Sets the program counter (PC) value to the line address of the cursor.

https://microchipdeveloper.com/mplabx:debug-toolbar

10)

Assembly code	PC	Wreg	Status bits			Remarks
			Z	DC	C	
`movlw .123`	`0x0`	`0x7b` (123 in decimal) 0x7b -> (W)	x	x	x	Load "123" into W register.
`clrw`	`0x1`	`0x0` 00h -> (W)	1	x	x	W register cleared. Z is set.
`movlw CONST1`	`0x2`	`0x3c*` CONST1 -> (W)	1(x)	x	x	Load "CONST1" into W register.
`addlw 'F'`	`0x3`	`0x82` (W)+F -> (W) 0x3c("CONST1") + 0x46("F")** = 0x82	0	1	0	Contents of W register added to "F". Result stored in W register.
`xorlw CONST2`	`0x4`	`0xab` (W) .XOR. CONST2 -> (W) 0x82 .XOR. 0x29*** = 0xab	0	1(x)	x	Contents of W register is XOR'ed with "CONST2". Result stored in W register.
`movwf var1`	`0x5`	`0xab` (W) -> (var1) 0xab moved to "var1"	x	1(x)	x	Data from W register moved to register "var1".
`rlf var1, w`	`0x6`	`0x56`	x	1(x)	1	Contents "var1" rotated one bit to the left through Carry flag. Result stored in W register.

Table 1: Debugging assembly code

*Value of CONST1 in HEX.

**0x46 is HEX value for literal "F".

***0x29 is HEX value for binary number "00101001".

11)

12)

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
000	00	00	07	1B	00	00	00	00	00	00	00	00	00	00	00	00	....
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
020	AB	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
080	00	FF	07	1B	00	FF	FF	FF	FF	0F	00	00	00	00	10	60	....`
090	00	00	FF	00	00	FF	00	00	00	00	00	00	00	01	00	00	....
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
100	00	00	07	1B	00	08	00	00	00	02	00	00	00	00	00	00	....
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
180	00	FF	07	1B	00	00	FF	40	FF	3F	00	00	00	00	--	--	.......@ .?....--
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

(Screenshot taken from MPLAB X IDE – "File Registers" window)

By going to the "File Registers" window in MPLAB X, I was able to see the value in data register "var1", located at 0x20 (defined in the code). Once I reached the final instruction through the debugging process, 0x20 showed the final value of the register in the "File Registers" window (shown above). Here, the final value was 0xab (highlighted in dark blue).

For final values of all other related registers, refer to "Table1: Debugging assembly code" on the previous page.

16)

Debugger: A "**debugger**" provides equivalent access using on-chip debugging hardware with standard production processors.

Simulator: A "**simulator**" is a software model that provides similar functionality but no hardware is used. The code is executed in the IDE environment simulating the processor in software.

https://www.microchip.com/forums/m882471.aspx#:~:text=A%20%22debugger%22%20provides%20equivalent%20access,hardware%20with%20standard%20production%20processors.&text=A%20%22simulator%22%20is%20a%20software,simulating%20the%20processor%20in%20software.