LAB REPORT

Submission Required Element

a) What operating system (including revision) did you use for your code development?

Answer) Operating system used for code development was "Windows 7 Enterprise – Service Pack 1.

b) What compiler (including revision) did you use?

Answer) The below compilers were used for compiling code:

as31 revision 2.0– Paulmon.asm, extra.asm

SDCC Rev 2.3.1 – Lab3 Required element, Supplemental Element and Challenges

asm51 – ASM code for test read/write to XRAM

asx8051 – Challenge #19

c) What exactly (include name/revision if appropriate) did you use to build your code (what IDE, make/makefile, or command line)?

Answer) Code::Blocks IDE Release 12.11 Rev 8629 was used to build the C code and the ASM code was edited in Notepad.

d) Did you install and use any other software tools to complete your lab assignment?

Answer) Tera Term terminal emulator and Flip 3.4.7 Programming utility were the other software tools installed to complete the lab assignment.

e) Did you experience any problems with any of the software tools? If so, describe the problems.

Answer) No significant problems were encountered with the software tools.

Required Element output demonstration:

- Paulmon, Extra and lab#3 required element programs were written to the microcontroller flash.

The required element program to allocate buffers and store characters was run using Paulmon.

Below is the user interface for the program:

```
                Welcome to PAULMON2 v2.1, by Paul Stoffregen

  See PAULMON2.DOC, PAULMON2.EQU and PAULMON2.HDR for more information.

Program Name                       Location      Type
  List                               1000      External command
  Single-Step                        1400      External command
  Memory Editor (VT100)              1800      External command

PAULMON2 Loc:5000 > Jump to memory location

Jump to memory location (5000), or ESC to quit: 3000

running program:

----------------------------------------------------
This program echoes characters entered from the terminal
It stores the 'storage' characters in XRAM buffer0
Buffers 0 and 1 will be allocated memory in XRAM
Buffer 2 can be used to maintain a copy of storage characters in buffer0
It is possible to generate a heap report of the storage characters in buffers

Enter buffer size between 32 and 3200 bytes:
320

'A-z a-z 0-9' - Storage characters'?' - Heap Report and empty the buffer
'=' - Display characters in the buffer and provide their memory location
'*' - Copy contents of buffer0 to buffer2
'@' - Heap Report and empty the buffer

Enter character:
d

Character stored
d

Enter character:
d

Character stored
d

Enter character:
f

Character stored
f

Enter character:
t

Character stored
t

Enter character:
6
```

- The ‘?’ feature provides shows a heap report and empties the buffer. Below is the snapshot of ‘?’ output:

```
Enter character:
?

------------------------------------------
------------------------------------------
Heap Report
Buffer: Buffer0
Buffer Start address: X:0x0005
Buffer End Address: X:0x0016
Total allocated size in bytes: 18
Number of storage characters in the buffer: 18
Free space in the buffer: 302
Number of characters since last '?': 19

------------------------------------------
------------------------------------------

------------------------------------------
------------------------------------------
Heap Report
Buffer: Buffer1
Buffer Start address: X:0x0149
Buffer End Address: X:0x0170
Total allocated size in bytes: 0
Number of storage characters in the buffer: 0
Free space in the buffer: 40
Number of characters since last '?': 0

------------------------------------------
------------------------------------------

Buffer0:
h h a s h d o h q o h i d d n q w h

Enter character:
```

- The ‘=’ feature shows characters in the respective buffers in hex with their addresses.

Additional Feature “*”

The ‘*’ key is used to enable this feature. This additional feature was added to copy the contents of buffer0 to a separate buffer. This can be used to copy the contents of the buffer before emptying it using ‘?’. Below is the snapshot of the output:

```
Enter character:
*

------------------------------------------
------------------------------------------
Heap Report
Buffer: Buffer2
Buffer Start address: X:0x0175
Buffer End Address: X:0x0189
Total allocated size in bytes: 21
Number of storage characters in the buffer: 21
Free space in the buffer: 299
Number of characters since last '*': 0

------------------------------------------
------------------------------------------
d d f t 6 6 8 u h h n j n h w s h s w h w

Enter character:
=

Buffer0:

X:0x0005: 0x64  0x64  0x66  0x74  0x36  0x36  0x38  0x75  0x68  0x68  0x6E  0x6A  0x6E  0x68  0x77  0x73
X:0x0015: 0x68  0x73  0x77  0x68  0x77
Buffer1:

X:0x0149: 0x0
Buffer2:

X:0x0175: 0x64  0x64  0x66  0x74  0x36  0x36  0x38  0x75  0x68  0x68  0x6E  0x6A  0x6E  0x68  0x77  0x73
X:0x0185: 0x68  0x73  0x77  0x68  0x77

Enter character:
```

- Error Handling was done in the program by not allowing the user to enter special characters and buffer size. Below is the snapshot:

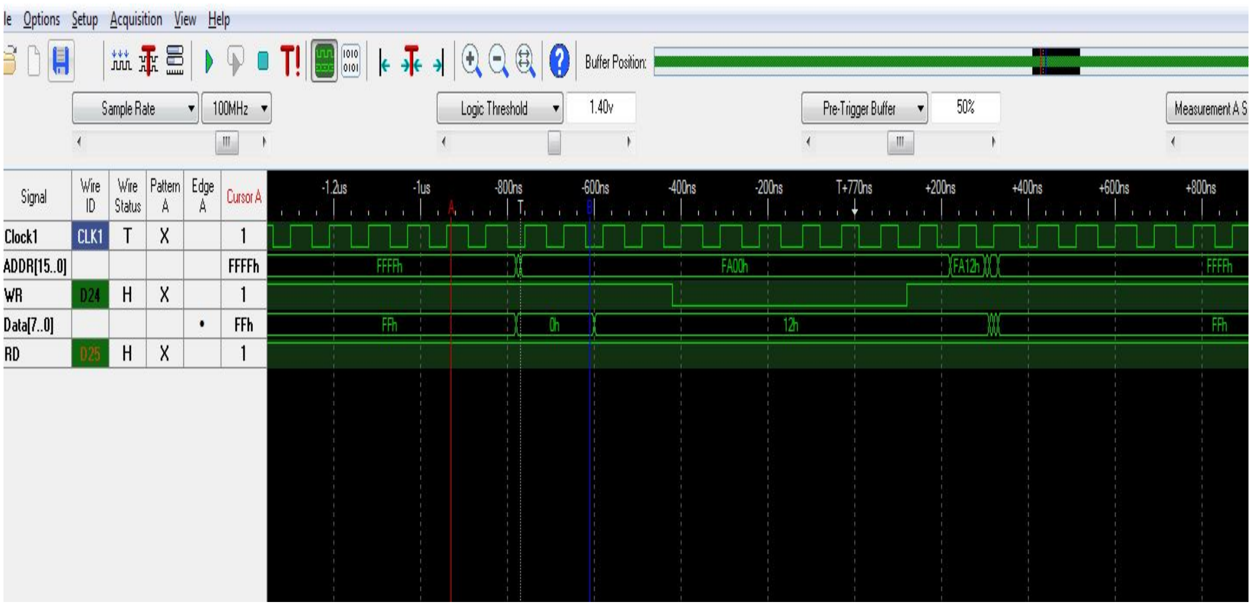
```
-----------------------------------------------------
This program echoes characters entered from the terminal
It stores the 'storage' characters in XRAM buffer0
Buffers 0 and 1 will be allocated memory in XRAM
Buffer 2 can be used to maintain a copy of storage characters in buffer0
It is possible to generate a heap report of the storage characters in buffers

Enter buffer size between 32 and 3200 bytes:
43^
Please enter integer input. Special characters are not allowed.

-----------------------------------------------------
This program echoes characters entered from the terminal
It stores the 'storage' characters in XRAM buffer0
Buffers 0 and 1 will be allocated memory in XRAM
Buffer 2 can be used to maintain a copy of storage characters in buffer0
It is possible to generate a heap report of the storage characters in buffers

Enter buffer size between 32 and 3200 bytes:
<
Please enter integer input. Special characters are not allowed.

-----------------------------------------------------
This program echoes characters entered from the terminal
It stores the 'storage' characters in XRAM buffer0
Buffers 0 and 1 will be allocated memory in XRAM
Buffer 2 can be used to maintain a copy of storage characters in buffer0
It is possible to generate a heap report of the storage characters in buffers

Enter buffer size between 32 and 3200 bytes:
456
Incorrect buffer size
Enter buffer size in even multiples of 16
456

-----------------------------------------------------
This program echoes characters entered from the terminal
It stores the 'storage' characters in XRAM buffer0
Buffers 0 and 1 will be allocated memory in XRAM
Buffer 2 can be used to maintain a copy of storage characters in buffer0
It is possible to generate a heap report of the storage characters in buffers

Enter buffer size between 32 and 3200 bytes:
15
Incorrect buffer size
Enter buffer size in even multiples of 16
15

-----------------------------------------------------
This program echoes characters entered from the terminal
It stores the 'storage' characters in XRAM buffer0
Buffers 0 and 1 will be allocated memory in XRAM
Buffer 2 can be used to maintain a copy of storage characters in buffer0
It is possible to generate a heap report of the storage characters in buffers

Enter buffer size between 32 and 3200 bytes:
640


'A-z a-z 0-9' - Storage characters'?' - Heap Report and empty the buffer
'=' - Display characters in the buffer and provide their memory location
'*' - Copy contents of buffer0 to buffer2
'@' - Heap Report and empty the buffer


Enter character:
```

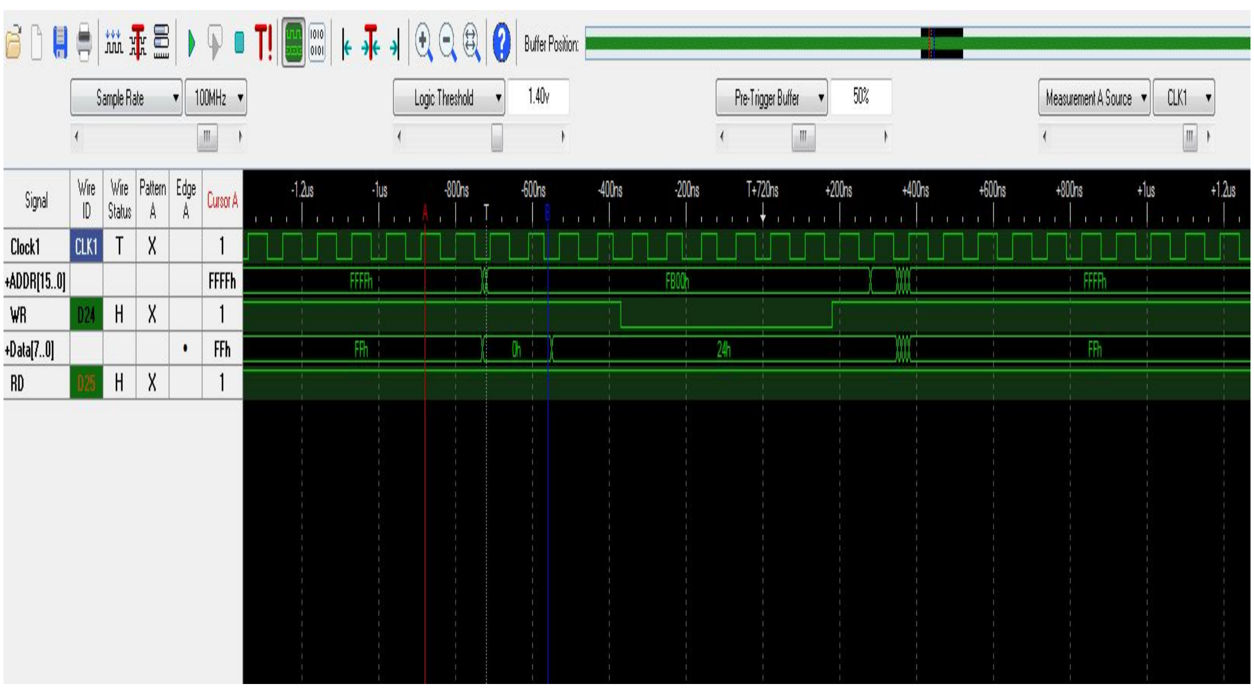
- Below is a snapshot of Block Fill done using Paulmon. It can be seen Paulmon has enabled the internal 1KB of XRAM. Using block fill, ‘U’ was written to the memory addresses from 0000 to 7FFFh.

```
DATA        8051 External Memory Editor, Paul Stoffregen, 1996
ADDR: +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F   ASCII EQUIVILANT
------------------------------------------------------------------------
7F00: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F10: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F20: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F30: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F40: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F50: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F60: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F70: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F80: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F90: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FA0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FB0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FC0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FD0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FE0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FF0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
------------------------------------------------------------------------
^E-Edit  ^G=Goto  ^C=Code  ^D=Data  ^L=Redraw  ^Q=Quit
```

```
DATA        8051 External Memory Editor, Paul Stoffregen, 1996
ADDR: +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F   ASCII EQUIVILANT
------------------------------------------------------------------------
0000: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0010: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0020: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0030: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0040: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00A0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
------------------------------------------------------------------------
^E-Edit  ^G=Goto  ^C=Code  ^D=Data  ^L=Redraw  ^Q=Quit
```

- Debug port functionality: Multiple debug ports were used. One of the debug ports was activated when '?' feature is pressed. This port writes 12h to address FA00h.
  The other debug port is activated when '=' is pressed. It writes 24h to address FB00h.





Supplemental Element output demonstration:

PCA Modes demonstrated:

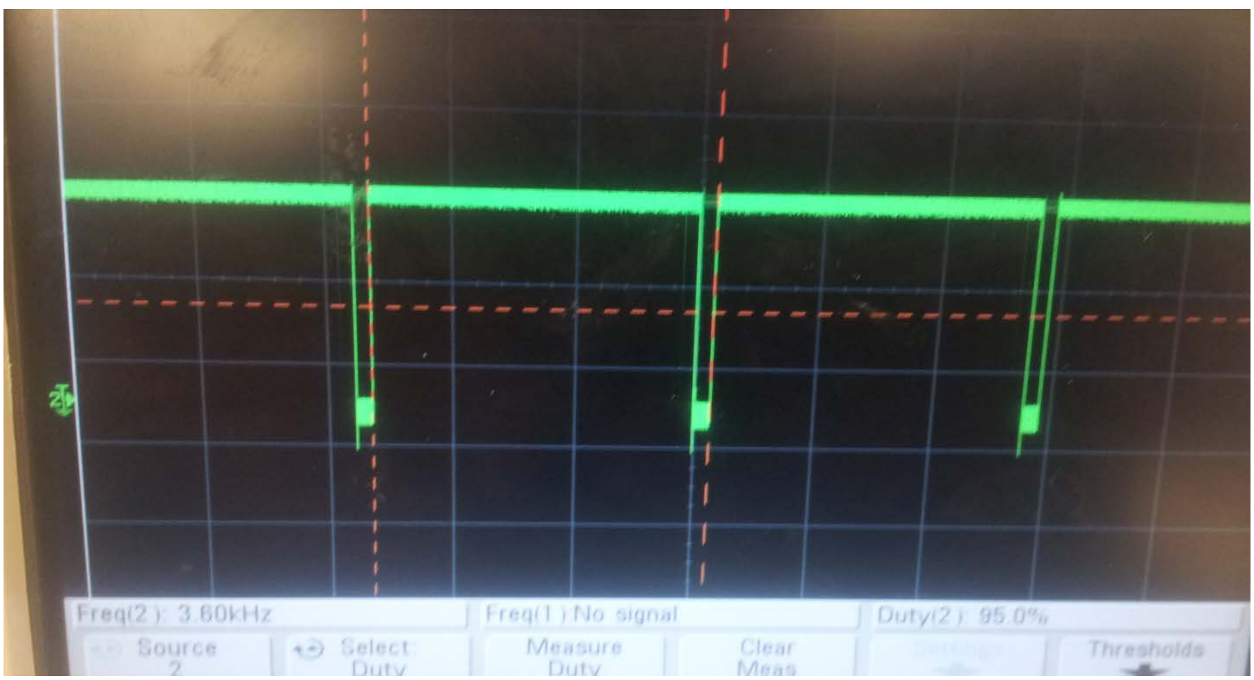
- **a)** PWM mode
- **b)** 16 bit internal timer
- **c)** High speed output
- **d)** Watchdog timer

a) PWM mode:

Run PWM

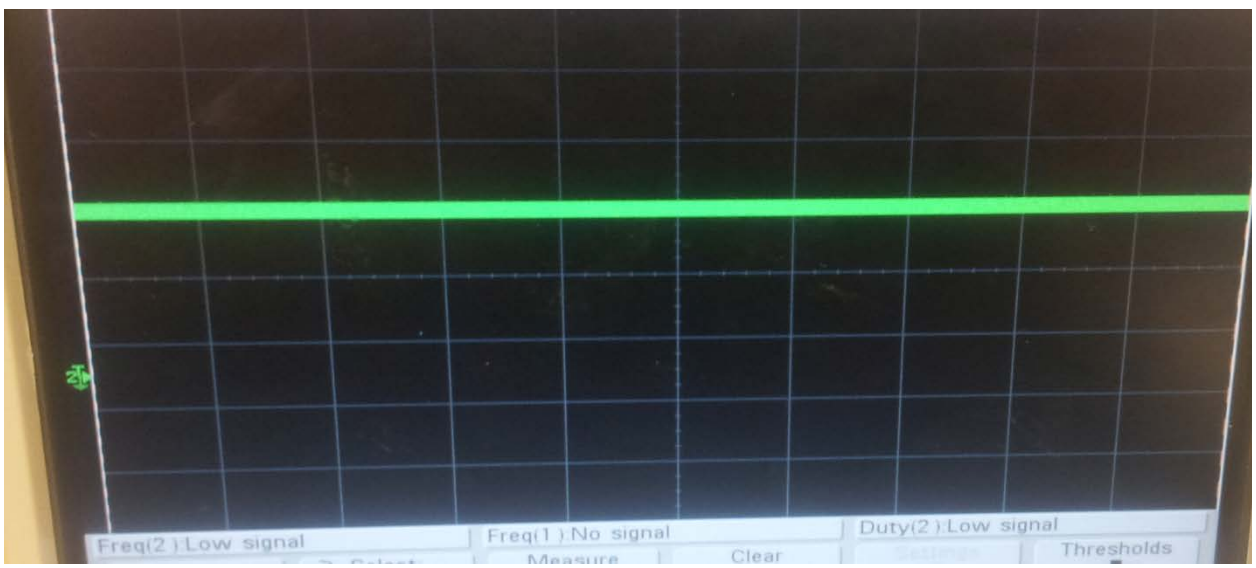
```
PWM - a

16-bit software timer - b

High speed output - c

Watchdog timer - d
a
Entered PWM mode
Choose one of the options below:

PWM - 1
Stop PWM - 2
Increase PWM duty cycle - 3
Decrease PWM duty cycle - 4
Enter Idle Mode - 5
Enter Power Down Mode - 6
1
Run PWM
6
Entered Power Down mode

Select PCA mode:
PWM - a

16-bit software timer - b

High speed output - c

Watchdog timer - d
a
Entered PWM mode
Choose one of the options below:

PWM - 1
Stop PWM - 2
Increase PWM duty cycle - 3
Decrease PWM duty cycle - 4
Enter Idle Mode - 5
Enter Power Down Mode - 6
1
Run PWM
5
Entered Idle mode
```

Increase pulse width:



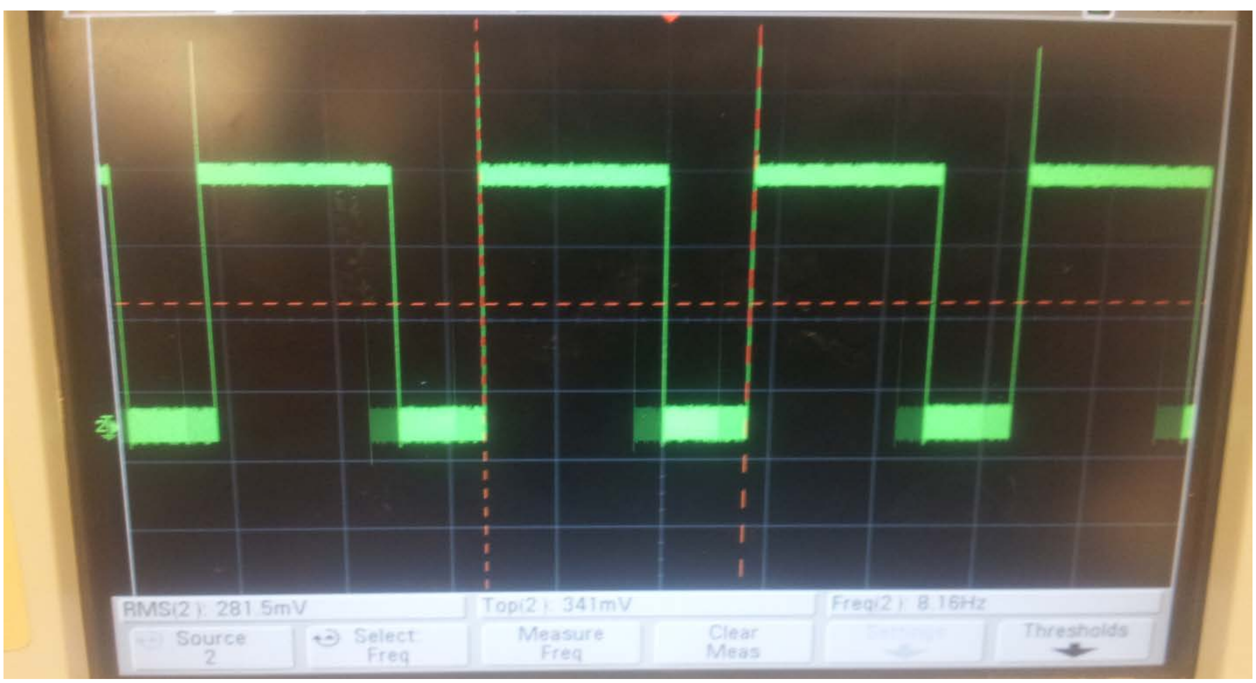
Decrease pulse width:

Idle mode:

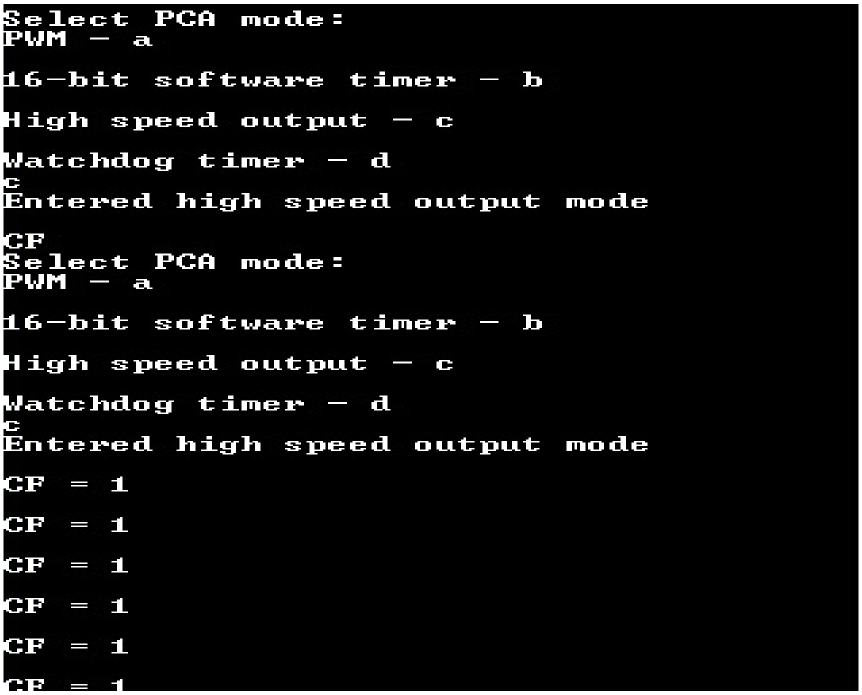


b) 16 bit software timer:

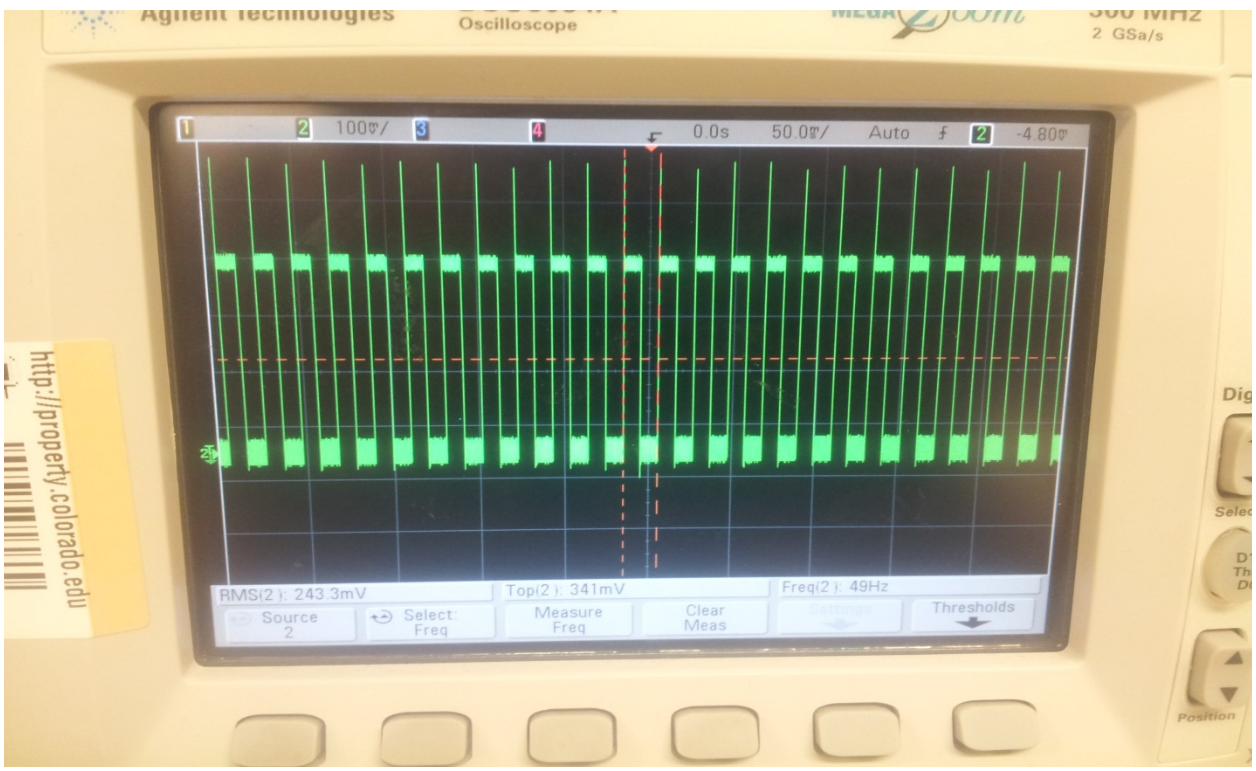
```
Select PCA mode:
PWM - a

16-bit software timer - b

High speed output - c

Watchdog timer - d
b
Entered 16 bit software timer mode

CF = 0

CF = 0

CF = 1

Entered 16 bit software timer mode

CF = 0

CF = 0

CF = 0

CF = 0

CF = 0

CF = 0

CF = 0

CF = 1
```



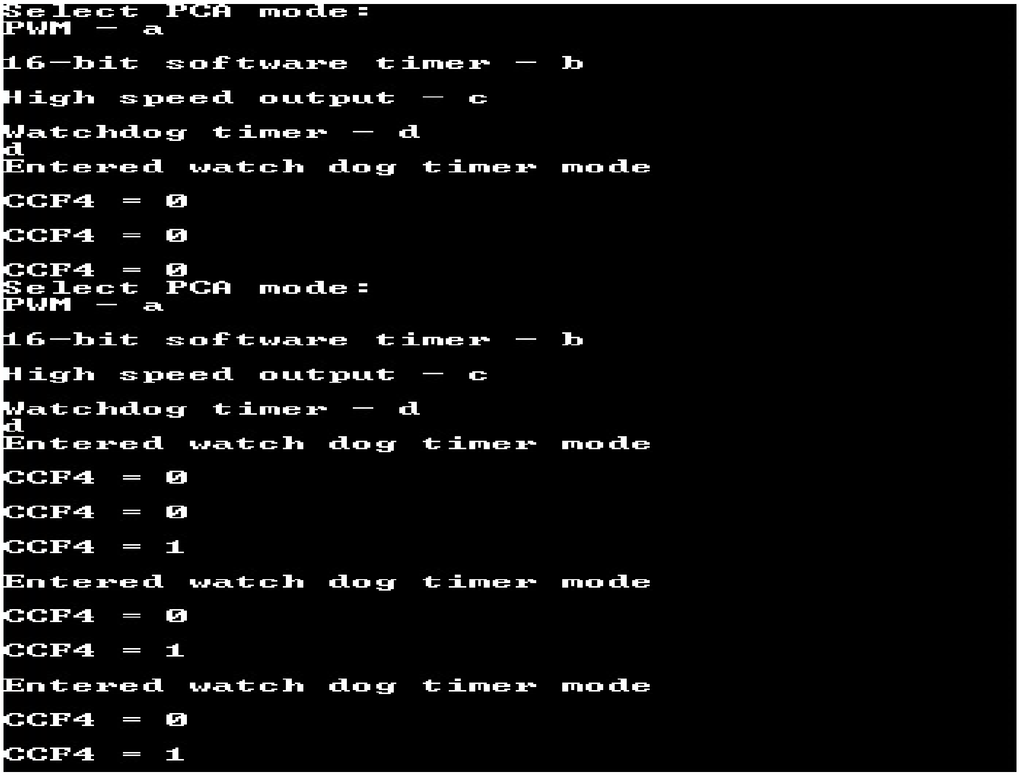
c) High speed output mode



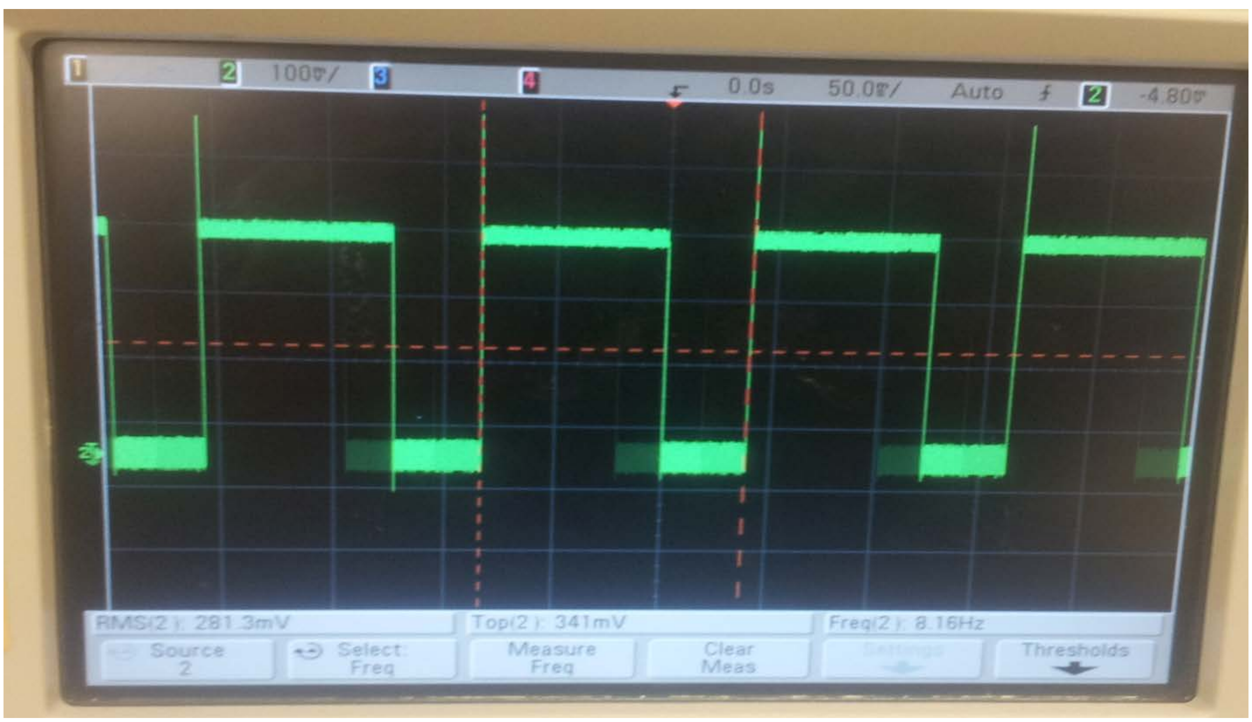
```
Select PCA mode:
PWM - a

16-bit software timer - b

High speed output - c

Watchdog timer - d
c
Entered high speed output mode

CF
Select PCA mode:
PWM - a

16-bit software timer - b

High speed output - c

Watchdog timer - d
c
Entered high speed output mode

CF = 1

CF = 1

CF = 1

CF = 1

CF = 1

CF = 1
```



d) Watchdog timer

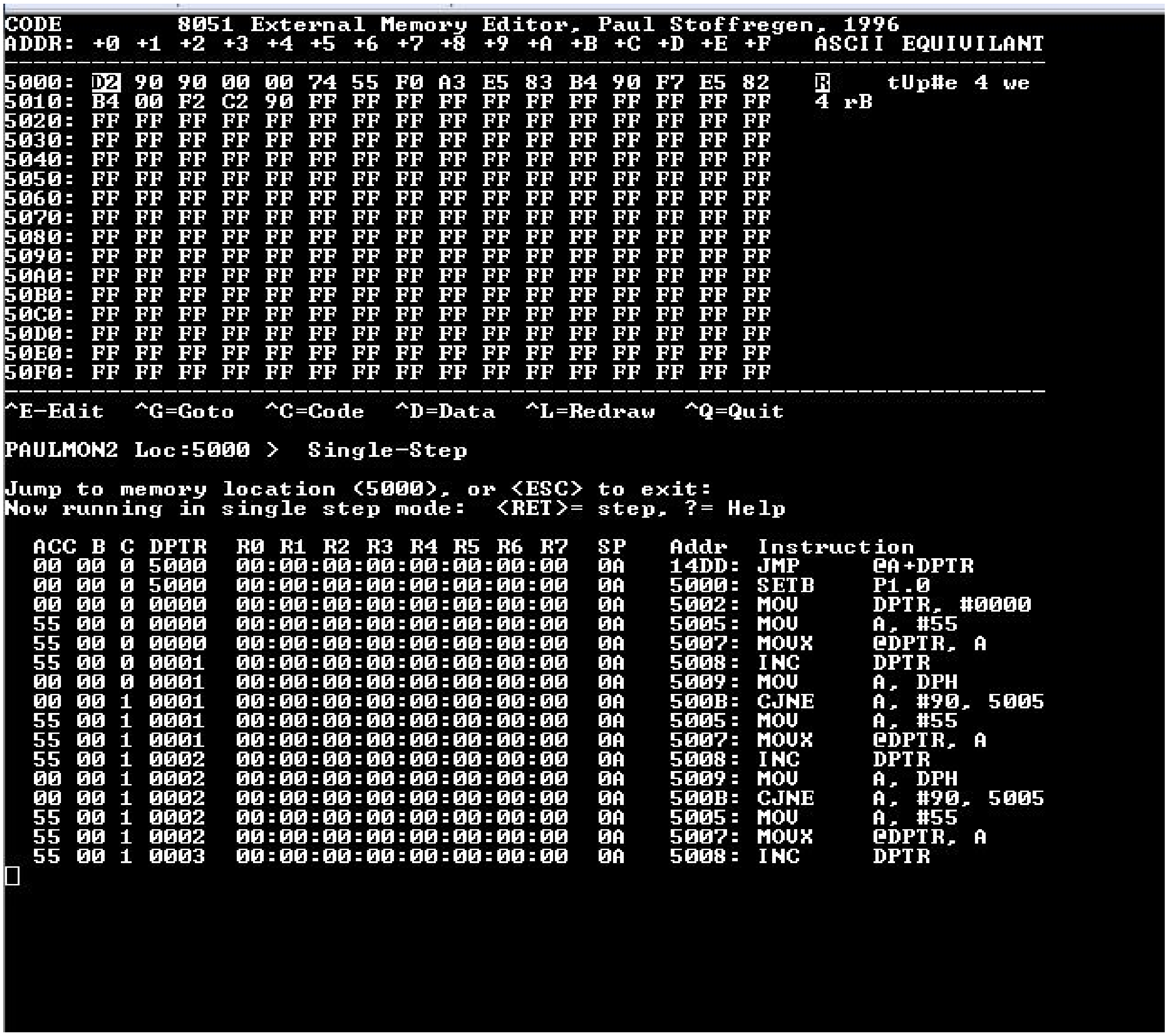


```
Select PCA mode:
PWM - a
16-bit software timer - b
High speed output - c
Watchdog timer - d
d
Entered watch dog timer mode
CCF4 = 0
CCF4 = 0
CCF4 = 0
Select PCA mode:
PWM - a
16-bit software timer - b
High speed output - c
Watchdog timer - d
d
Entered watch dog timer mode
CCF4 = 0
CCF4 = 0
CCF4 = 1
Entered watch dog timer mode
CCF4 = 0
CCF4 = 1
Entered watch dog timer mode
CCF4 = 0
CCF4 = 1
```



Challenges:

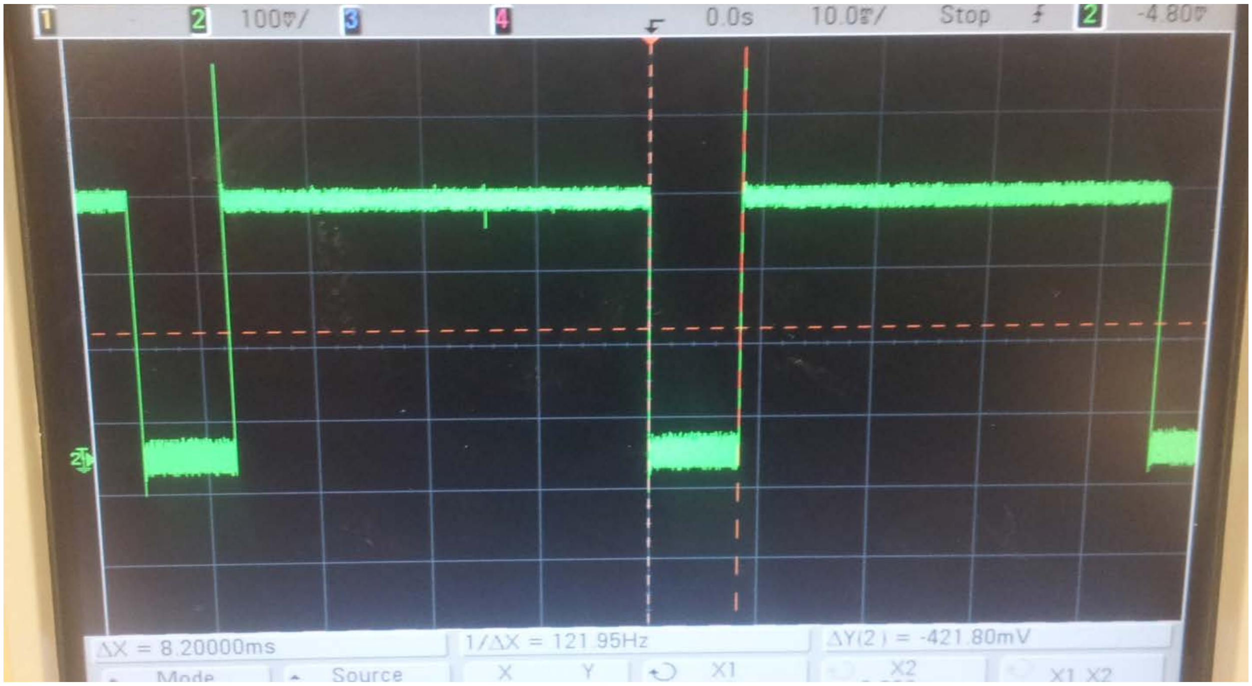
1. PAULMON Run command + single stepping:



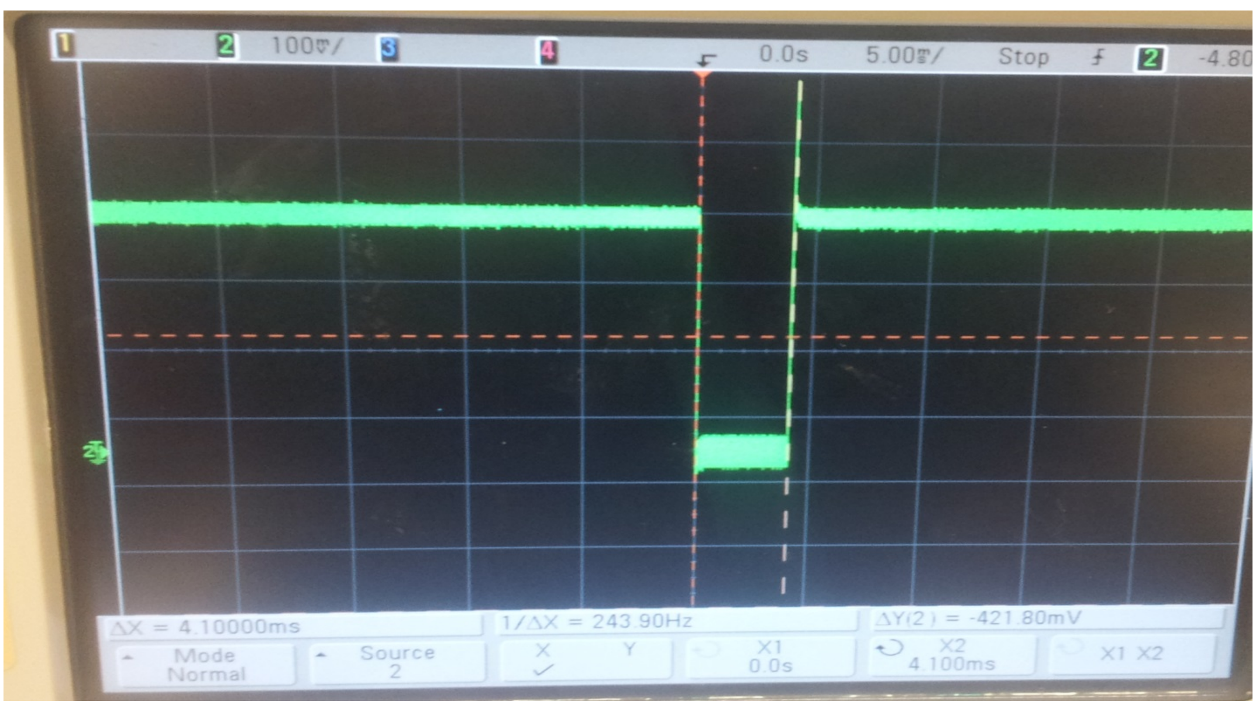
```
CODE          8051 External Memory Editor, Paul Stoffregen, 1996
ADDR: +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F   ASCII EQUIVILANT
--------------------------------------------------------------------------------
5000: D2 90 90 00 00 74 55 F0 A3 E5 83 B4 90 F7 E5 82   R     tUp#e 4 we
5010: B4 00 F2 C2 90 FF FF FF FF FF FF FF FF FF FF FF   4 rB
5020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5060: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5070: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5090: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
50A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
50B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
50C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
50D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
50E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
50F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
--------------------------------------------------------------------------------
^E-Edit  ^G=Goto  ^C=Code  ^D=Data  ^L=Redraw  ^Q=Quit

PAULMON2 Loc:5000 >  Single-Step

Jump to memory location (5000), or <ESC> to exit:
Now running in single step mode:  <RET>= step, ?= Help

  ACC B  C DPTR  R0 R1 R2 R3 R4 R5 R6 R7  SP   Addr  Instruction
  00 00 0 5000  00:00:00:00:00:00:00:00  0A   14DD: JMP     @A+DPTR
  00 00 0 5000  00:00:00:00:00:00:00:00  0A   5000: SETB    P1.0
  00 00 0 0000  00:00:00:00:00:00:00:00  0A   5002: MOV     DPTR, #0000
  55 00 0 0000  00:00:00:00:00:00:00:00  0A   5005: MOV     A, #55
  55 00 0 0000  00:00:00:00:00:00:00:00  0A   5007: MOVX    @DPTR, A
  55 00 0 0001  00:00:00:00:00:00:00:00  0A   5008: INC     DPTR
  00 00 0 0001  00:00:00:00:00:00:00:00  0A   5009: MOV     A, DPH
  00 00 1 0001  00:00:00:00:00:00:00:00  0A   500B: CJNE    A, #90, 5005
  55 00 1 0001  00:00:00:00:00:00:00:00  0A   5005: MOV     A, #55
  55 00 1 0001  00:00:00:00:00:00:00:00  0A   5007: MOVX    @DPTR, A
  55 00 1 0002  00:00:00:00:00:00:00:00  0A   5008: INC     DPTR
  00 00 1 0002  00:00:00:00:00:00:00:00  0A   5009: MOV     A, DPH
  00 00 1 0002  00:00:00:00:00:00:00:00  0A   500B: CJNE    A, #90, 5005
  55 00 1 0002  00:00:00:00:00:00:00:00  0A   5005: MOV     A, #55
  55 00 1 0002  00:00:00:00:00:00:00:00  0A   5007: MOVX    @DPTR, A
  55 00 1 0003  00:00:00:00:00:00:00:00  0A   5008: INC     DPTR
```

2. **Floating Point performance in X1 & X2 mode:**

X1 mode:



X2 mode:



A port pin has been toggled on performing an intensive floating point calculation.

As it can be seen, in X1 mode it takes 8.2ms to perform the calculation and 4.1ms in X2 mode to perform the same calculation.

This demonstrates that the frequency is doubled in X2 mode and it can be used to perform intensive math calculations.

3. Assembly and C interfacing

A C function c_func() is called from main() in the C file. The c_func in turn calls asm_func() which is an assembly language function and passes two parameters. The asm_func() calls abc() a function in the C file and passes a parameter. This parameter passed is printed as “2” on the first line of the output. Then the asm_func() adds two numbers that were passed and returns control to c_func() which in turn returns control to main().

The addition of the two parameters is printed as “8” below.

```
2
Addition of number a and number b is 8
```

Below are the commands used to compile and link the ASM and C code.

```
C:\sdcc\bin>asx8051 -losg asmfunc.asm

C:\sdcc\bin>sdcc cfunc.c asmfunc.rel

C:\sdcc\bin>
```