

Keil uVision Debugging Programs

**WEL Labs, IITB
2016**

Development Tools (Revision)

- **Coding – Editor => Entry of code into file(s)**
- **Translation – Assembler or Compiler**
=> Generate machine code from source code
- **Execution check – using Debugger to verify operation of program (on Simulator)**
- **Program – Programmer**
=> Put machine code in the chip

Single Point Solution – IDE e.g. Keil

Keil uVision IDE ...

- **Project** : A **collection of files related to a particular programming task**.
- **Build** : The process in which **only the files modified since last build** are assembled/compiled for the chosen microcontroller device.
- **Rebuild** : The process in which **all files are assembled/compiled** irrespective of their modification state.
- **Debug** : The process of finding errors happening during program execution and removing them.

Simulator and Debugger

- **Simulation of microcontroller behavior while executing the user program**
- **The Debug mode UI allows the user to perform the following**
 - 1. Observe** microprocessor Registers, Memory, Ports and Peripherals
 - 2. Place breakpoints** to stop simulation as specific instruction or on condition
 - 3. Monitor code** under execution
 - 4. Modify data** variables
 - 5. Monitor timing** of execution

E:\ACADEMIC\PG\RA\pt-51\programs\avproj - μVision4

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Target 1

Project

- Target 1
 - Source Group 1
 - simple_move.asm

```
1  src equ 30h
2  dst equ 40h
3  cnt equ 5
4
5  mov r0,#src
6  mov r1,#dst
7  mov r2,#cnt
8
9  back:  mov a,@r0
10         mov @r1,a
11         inc r0
12         inc r1
13
14  djnz r2,back
15
16  loop:  sjmp loop
17
18  end
```

Assuming you have already built a program and obtained binary code file for it, We can use the simulator available with keil for debugging programs

Build Output

```
Build target 'Target 1'
linking...
Program Size: data=8.0 xdata=0 code=14
"a" - 0 Error(s), 0 Warning(s).
```

Simulation L:18 C:4 CAP NUM

After a successful build of the project, following are the resulting files

1. Binary code is placed in **filename.obj** for each source file in a project.
2. All these obj files are linked into a single binary named using the project name without any extension.
3. If enabled by user **<<filename.hex>>** file is created for the project.

The user can now start the Debug session to check execution of the program.

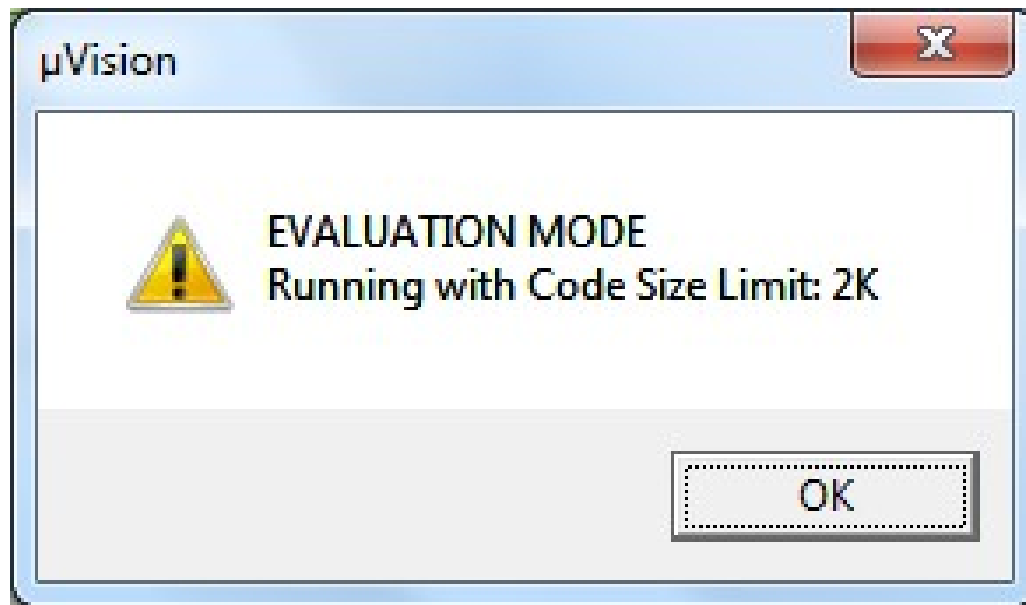
Build Output

```
Build target 'Target 1'
linking...
Program Size: data=8.0 xdata=0 code=14
"a" - 0 Error(s), 0 Warning(s).
```

Enter or leave a debug session

Simulation

CAP NUM SCR



A popup of Evaluation version is presented which shows the limitation of the mode.

On clicking "OK" the popup closes and the user is presented with the **Debug Mode** user interface (UI)

User interface in Debug mode

The screenshot shows a debugger interface with several windows. Four callout boxes highlight specific areas:

- Register window:** Located on the left, it shows a list of registers (r0-r7, Sys, a, b, sp, sp_max, PC, auxr1, dptr, states, sec, psw) and their values. A callout box labeled "Register window" points to this area.
- Disassembly window:** Located at the top right, it shows assembly code with addresses and opcodes. A callout box labeled "Disassembly window" points to this area.
- User program window:** Located in the center, it shows the source code for "simple_move.asm". A callout box labeled "User program window" points to this area.
- Memory window:** Located at the bottom right, it shows memory addresses and their contents. A callout box labeled "Memory window" points to this area.

Other visible windows include "Registers" (top left), "Command" (bottom left), and "Memory1" (bottom right). The status bar at the bottom shows "Simulation" mode, "t1: 0.00000000 sec", and "CAP NUM SCRL OVR R/W".

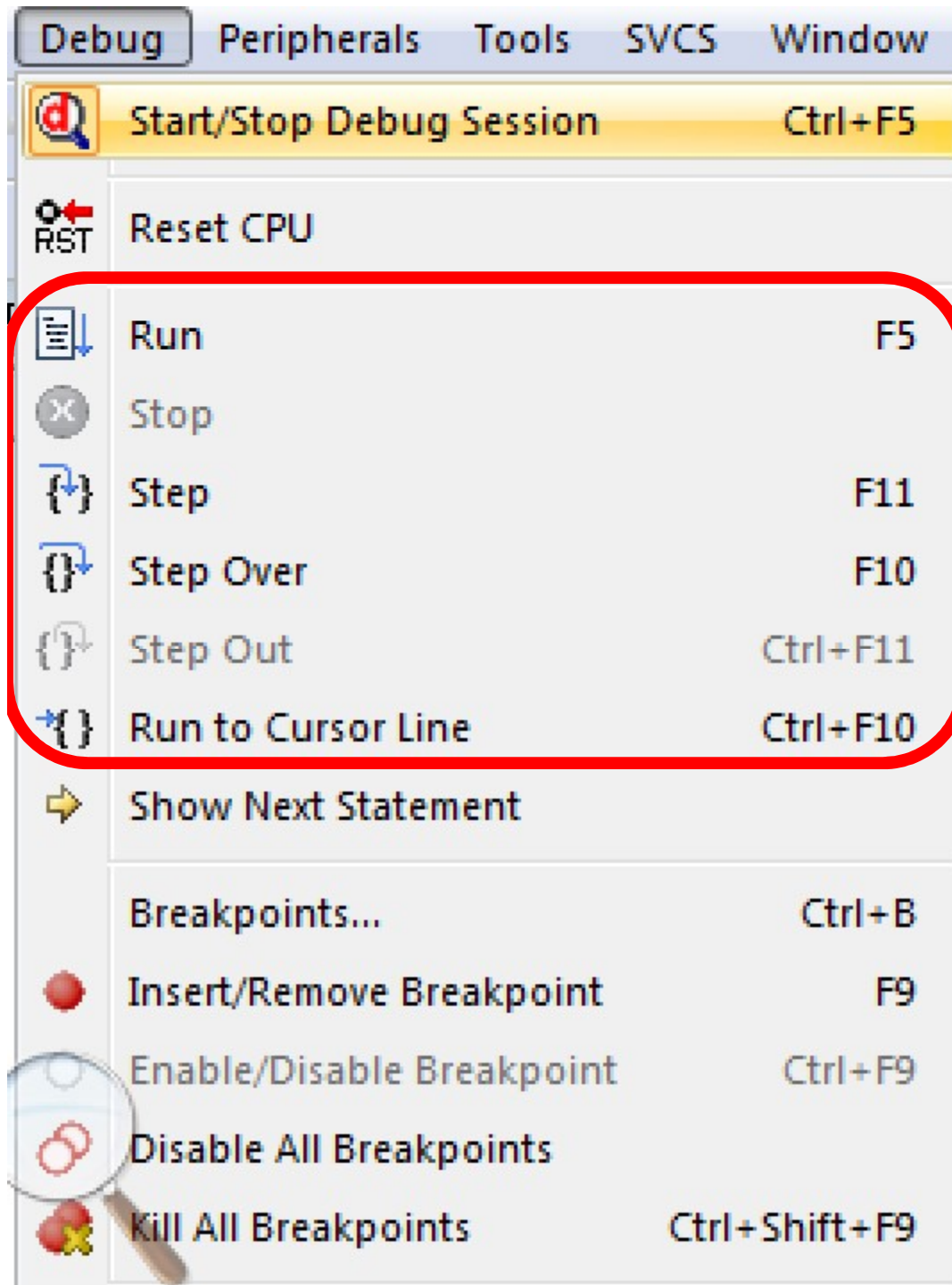
Disassembly window – shows the Opcodes for the Program loaded in the debugger.

Register window

User program window – shows the Assembly or High Level Program loaded in the debugger.

Command window

Memory window



User can execute instructions in multiple modes :

1. **Run (F5)** – Continues executing the program until the next active breakpoint is reached or till the program termination.
2. **Step (F11)**-- Executes a single-step into a function; Executes the current instruction line.
3. **Step Over (F10)** – Executes a single-step over a function.
4. **Run to Cursor Line (Ctrl+F10)**
Allows user to place a cursor and run the program till that line.

Details of Disassembly and Memory window

Disassembly

```
10: CLR P1.4
C:0x0100 C294 CLR 0x90.4
11: MOV R0, #31D
12: START:
C:0x0102 781F MOV R0, #0x1F
13: SETB EA
C:0x0106 D2AF SETB ETO(0xA8.1)
15: MOV TMOD, #0001B ; Sets Timer 0 to MODE1 (16 bit timer). Timer 1 is not used
C:0x0108 7589 MOV TMOD(0x89), #0x01
16: MOV TH0, #00H ; Loads TH0 register with FCH
C:0x010B 758C00 MOV TH0(0x8C), #0x00
17: MOV TLO, #00H ; Loads TLO register with 18H
```

Disassembly window

Address

Instruction mnemonic in assembly language

Opcode

Memory 1

Address: C:0x100 ← Enter address here

Memory window

C:0x0100	C2	94	78	1F	D2	AF	D2	A9	75	89	01	75	8C	00	75	8A	00	D2	8C	80	FE	C2	8C	C2
C:0x0118:	8D	08	04	B2	94	78	1F	D2	8C	22	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0130:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Data at address 0x0100

Call Stack + Locals | Memory 1

Code and Data memory access

Memory 1

Address: `d:00h`

d: refers to data memory

D:0x00:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D:0x18:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D:0x30:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D:0x48:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D:0x60:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D:0x78:	00	00	00	00	00	00	00	00	00	FF	07	00	00	00	00	00	10	00	00	00	00	00	00	00	00
D:0x90:	FF	00	00	00	F8	FF	FE	00	00	00	00	00	00	00	00	00	00	FF	00	00	00	00	00	00	00

Call Stack + Locals | Memory 1

Simulation t1: 0.00000000 sec L:11 C:9 CAP NUM SCRL OVR R/W

Memory 1

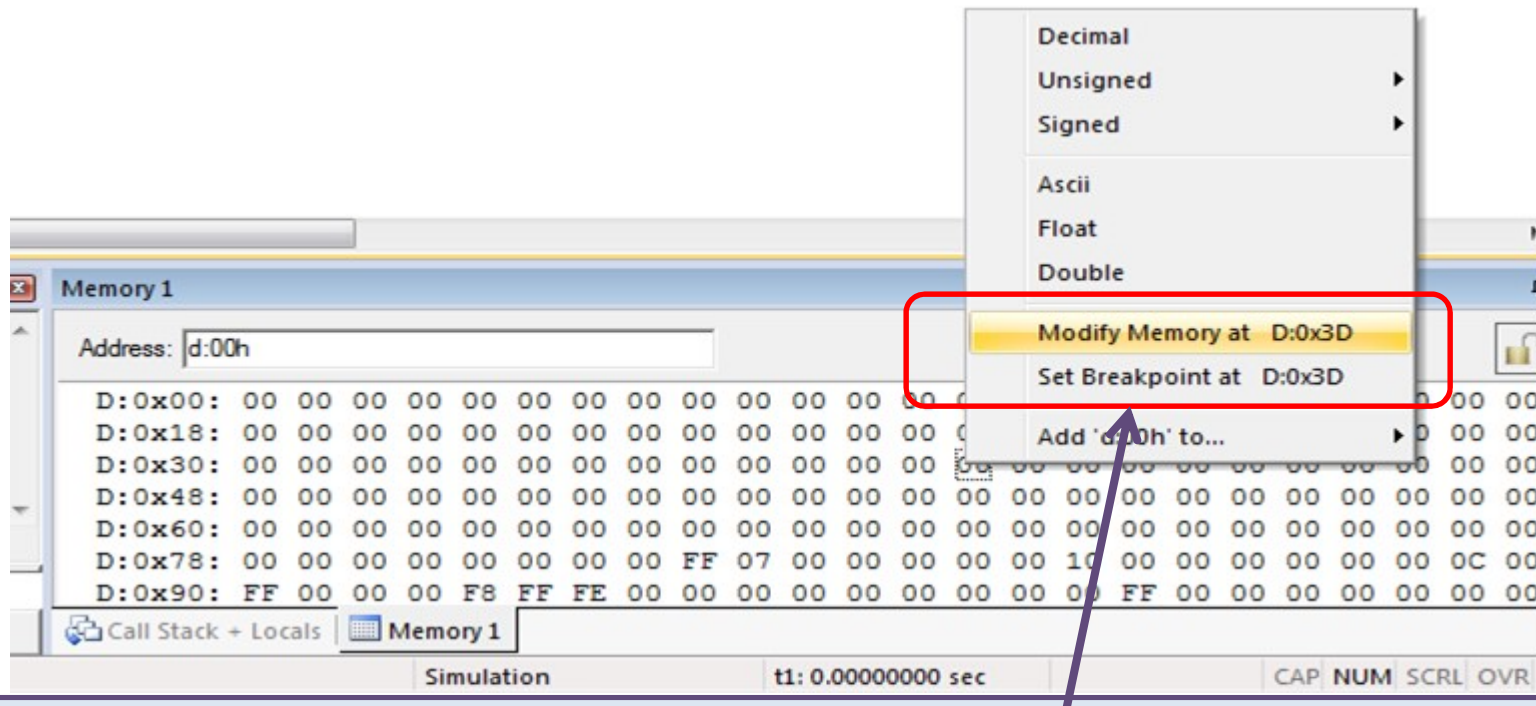
Address: `c:00h`

c: refers to code segment of the memory

C:0x0000:	78	30	79	40	7A	05	E6	F7	08	09	DA	FA	80	FE	00	00	00	00	00	00	00	00	00	00	00
C:0x0018:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0048:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0078:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0090:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Call Stack + Locals | Memory 1

Simulation t1: 0.00000000 sec L:10 C:14 CAP NUM SCRL OVR R/W



During execution, user can right click on the required memory location in the memory window to modify RAM data. Functionality for selecting the number system in which the memory contents are to be displayed is also available .

Note: To initialize memory contents on hardware, user has to add necessary instructions in the program code.

Register	Value
[-] Regs	
r0	0x30
r1	0x40
r2	0x05
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
[-] Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x0000
auxr1	0x00
[-] dptr	0x0000
[0]	0x0000
[1]	0x0000
states	0
sec	0.00000000
[-] psw	0x00
p	0
F1	0
ov	0
rs	0
f0	0
ac	0
cy	0

The Registers window provides access to all the registers including the flag register, DPTRs etc.

E:\ACADEMIC\PG\RA\pt-51\programs\a.uvproj - μVision4

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Registers

Register

Regs

r0

r1

r2

r3

r4

r5

r6

r7

Sys

a

b

sp

sp_

PC

aux

dptr

state

sec

psw

Project

Command

Running

Load "E:\ACADEMIC\PG\RA\pt-51\programs\a"

ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess

View

- Status Bar
- Toolbars
- Project Window
- Books Window
- Functions Window
- Templates Window
- Source Browser Window
- Build Output Window
- Find In Files Window
- Command Window
- Disassembly Window
- Symbol Window
- Registers Window
- Call Stack Window
- Watch Windows
- Memory Windows
 - Memory 1
 - Memory 2
 - Memory 3
 - Memory 4
- Serial Windows
- Analysis Windows
- Trace
- System Viewer
- Toolbox Window
- Periodic Window Update

Assembly

```
5: mov r0,#src
0000 7830 MOV R0,#0x30
6: mov r1,#dst
0002 7940 MOV R1,#0x40
7: mov r2,#cnt
8:
```

simple_move.asm

```
src equ 30h
dst equ 40h
cnt equ 5

mov r0,#src
mov r1,#dst
mov r2,#cnt

djnz r2,back
```

Memory 1

Address: d:00h

D:0x00:	35 45 00 00 00 00 00 00
D:0x18:	00 00 00 00 00 00 00 00
D:0x30:	00 00 00 00 00 00 00 00
D:0x48:	00 00 00 00 00 00 00 00
D:0x60:	00 00 00 00 00 00 00 00
D:0x78:	00 00 00 00 00 00 00 00
D:0x90:	FF 00 00 00 F8 FF FE 00

Call Stack + Locals Memory 1

Simulation

If some windows are not being displayed then use the "View" menu to get them on the window.

r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
[-] Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x00...
auxr1	0x00
[+] dptr	0x0000
states	0
sec	0.0000...
[+] psw	0x00

```
simple_move.asm
1  src equ 30h
2  dst equ 40h
3  cnt equ 5
4
5  mov r0,#src
6  mov r1,#dst
7  mov r2,#cnt
8
9  back:  mov a,@r0
10       mov @r1,a
11       inc r0
12       inc r1
13
14       djnz r2,back
15
16  loop:  sjmp 1
17
18  end
```



To set a breakpoint, user can click in the marked area against the corresponding line of code

The image shows a debugger interface with three main panes. On the left is the 'Registers' pane, in the middle is the 'Disassembly' pane, and on the right is the 'Assembly' pane. A red dot in the assembly pane indicates a breakpoint set on line 12.

Register	Value
Regs	
r0	0x30
r1	0x40
r2	0x05
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x00...
auxr1	0x00
dpnr	0x0000
states	0
sec	0.0000...
psw	0x00

```
11:      inc r0
C:0x0008  08      INC      R0
12:      inc r1
13:
C:0x0009  09      INC      R1
```

simple_move.asm

```
1  src equ 30h
2  dst equ 40h
3  cnt equ 5
4
5  mov r0,#src
6  mov r1,#dst
7  mov r2,#cnt
8
9  back:  mov a,@r
10         mov @r1,a
11         inc r0
12         inc r1
13
14  djnz r2,back
15
16  loop:  sjmp loop
17
18  end
```

The breakpoint is shown as a red dot against the line.

The breakpoint is automatically displayed at the equivalent line in the disassembly window too.

Peripherals menu

The screenshot displays a development environment with the 'Peripherals' menu open. The menu items include: Interrupt, I/O-Ports (highlighted), Serial, Timer, TWI, SPI, Keyboard Interrupt, EEPROM Data, Flash Memory, and Clock Control. The 'I/O-Ports' sub-menu is also open, showing Port 0 through Port 4. A callout box points to the 'Peripherals' menu with the text: 'Various Peripherals can be accessed through the "Peripherals" menu.'

The 'Registers' window shows the following data:

Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x0000
auxr1	0x00
dptr	0x0000
states	0
sec	0.00000000
psw	0x00

The assembly code window shows the following code:

```
1 src equ 30h
2 dst equ 40h
3 cnt equ 5
4
5 mov r0,#src
6 mov r1,#dst
7 mov r2,#cnt
8
9 back: mov a,@r0
10      mov @r1,a
11      inc r0
12      inc r1
13
14 djnz r2,back
```

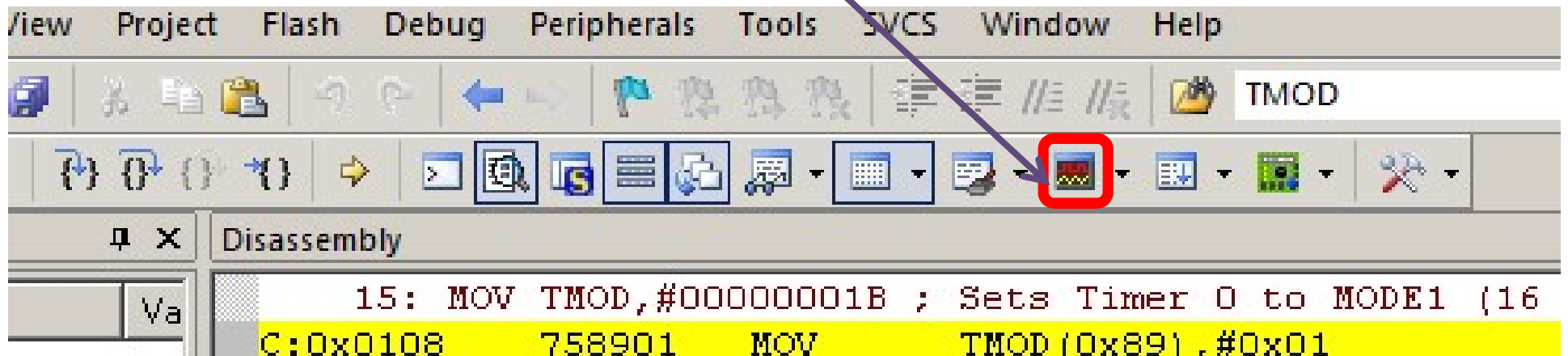
The 'Timer/Counter 0' configuration window is open, showing the following settings:

- Mode: 0: 13 Bit Timer/Counter
- Timer: Timer
- TCON: 0x00, TMOD: 0x00
- TH0: 0x00, TLO: 0x00
- TO Pin, TFO
- Control: Status: Stop
- TRO, GATE, INTO#

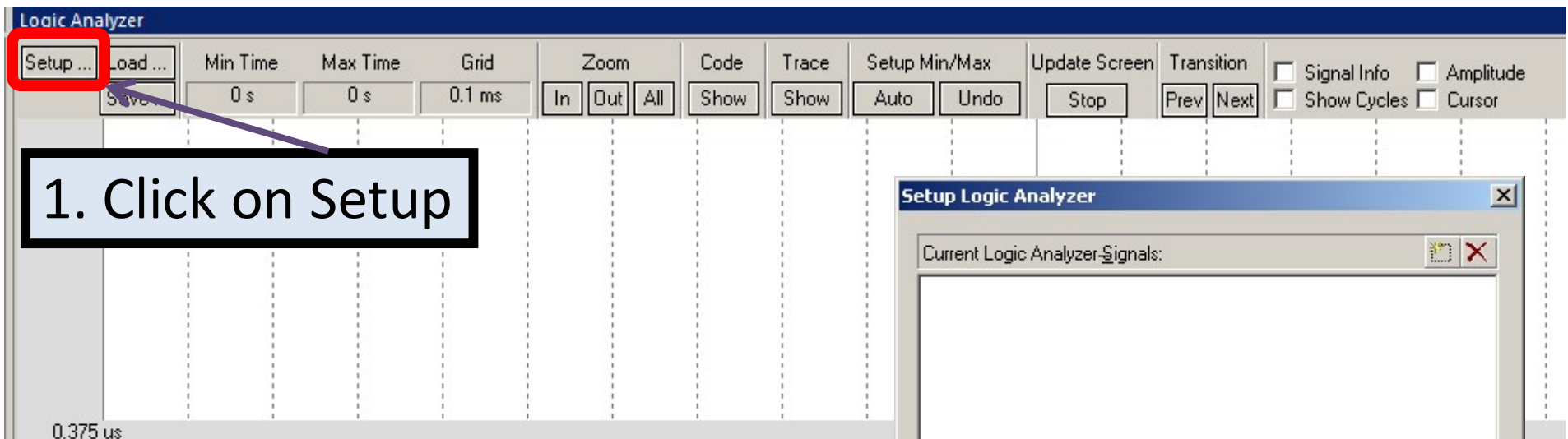
A callout box points to the 'Timer/Counter 0' window with the text: 'Window corresponding to Timer 0'.

Logic analyzer

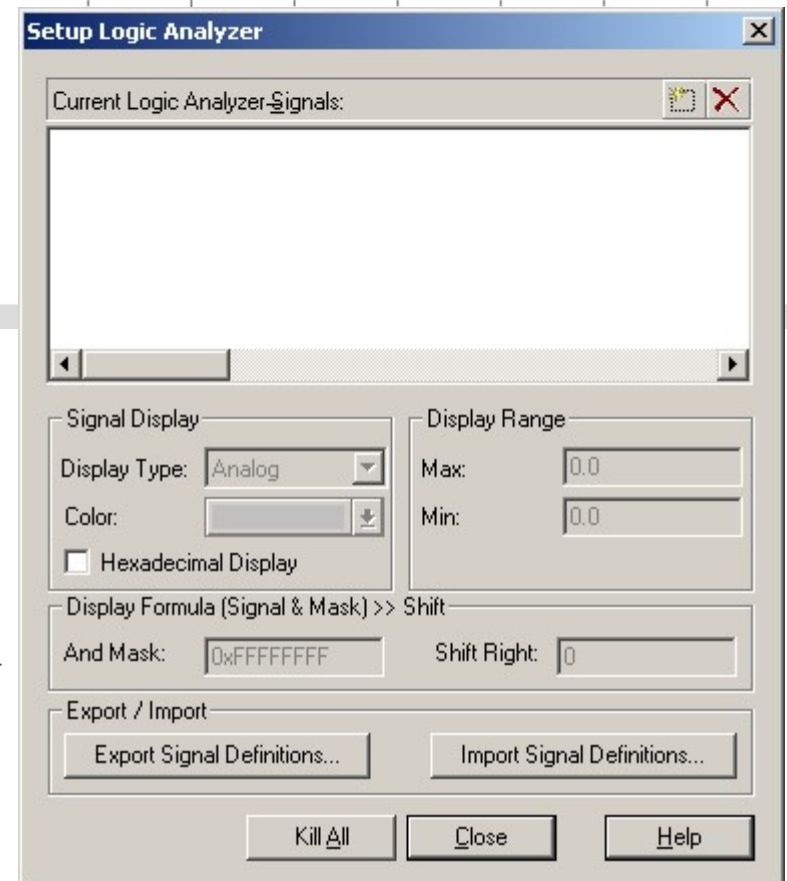
To start the logic analyzer click on the highlighted icon or go to **View > Analysis Window > Logic Analyzer**.



Logic Analyzer window



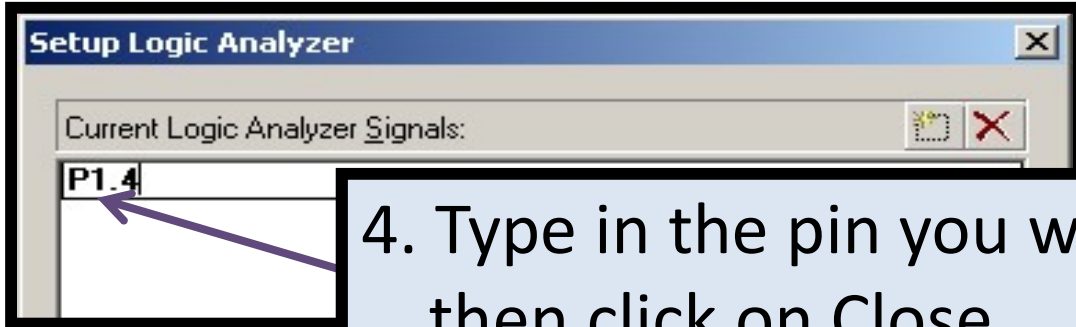
1. Click on Setup



2. The setup window appears.

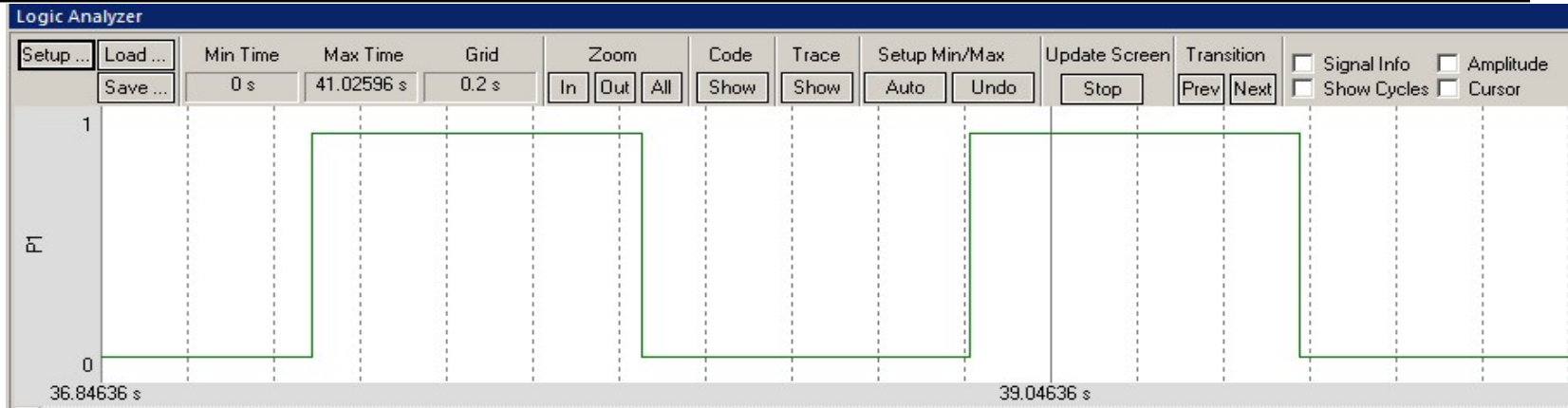


3. Click on the “New/insert” icon.



4. Type in the pin you want to monitor then click on Close.

5. After running a simulation, you can pause it and look at the timing waveforms to debug your code.



Questions ?

Thank you

WEL Labs, IITB
2016

For doubts/errors in this PPT contact :
Suryakant Toraskar e-mail: smtoraskar.iitbombay@gmail.com location : WEL5