Debugging with Code Composer Studio

TI Precision Labs – Microcontrollers

Presented by Matthew Pate

Prepared by Santosh Jha

Overview

In this video, we will cover the following topics

- How to debug applications using TI Code Composer Studio or similar Integrated Development Environment (IDE)
- Explain how to launch debug session
- What happens when debug session is launched
- Explain various debugging tools like breakpoint, variables, CPU registers, memory view
- Graphing tools

* Images are for TI Code Composer Studio(CCS). There are similar functionalities in other IDEs

Setup for a Debugging Application

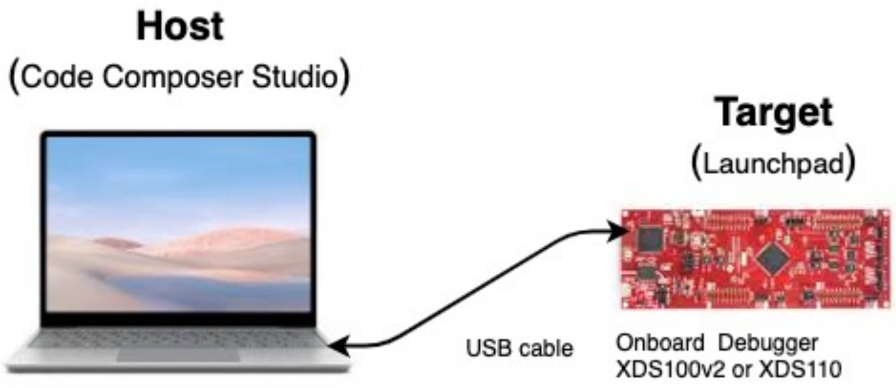
Software Setup:

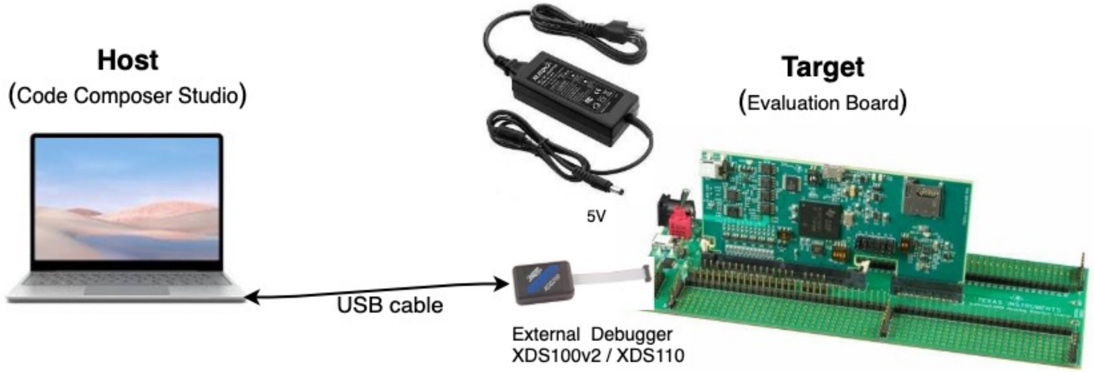
- Install TI Code Composer Studio
- Launch CCS Studio

Hardware Setup:

Select one option depending on EVM

- Option1 : Connect EVM using USB cable to board with onboard debugger
- Option2 : Provide external power to EVM and connect the EVM with an external debugger using USB cable



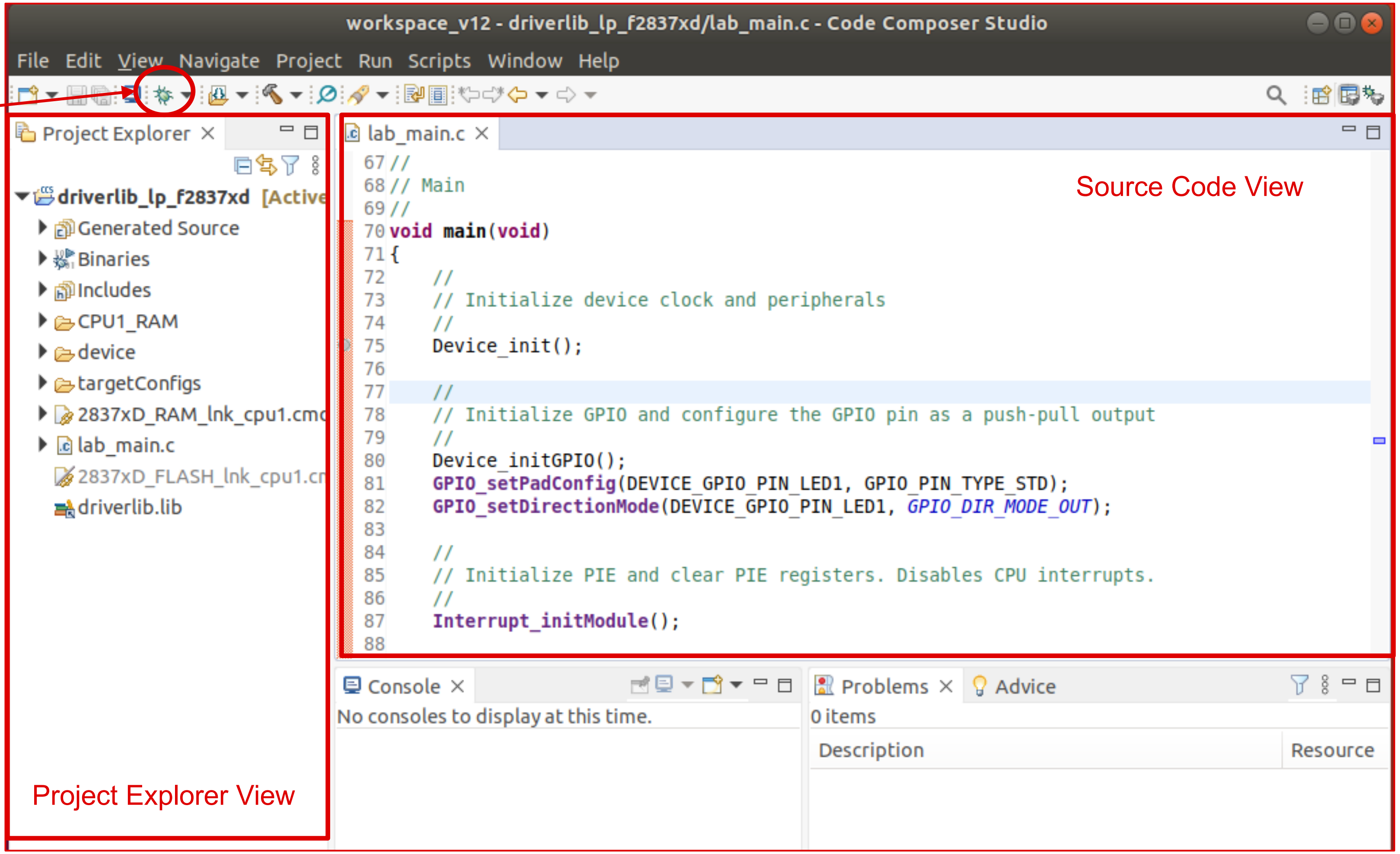


Launch Debug Session

- Open TI Composer Project
- Click the ‘**Bug**’ Icon on toolbar to launch debug session

When debug session is launched:

- Multiple panes are opened for debugging purpose
- CCS uses the Target Configuration File (.ccxml) to connect to the target device
- Hardware initialization is done through the GEL script



Source Code View

```
//
// Main
//
void main(void)
{
    //
    // Initialize device clock and peripherals
    //
    Device_init();

    //
    // Initialize GPIO and configure the GPIO pin as a push-pull output
    //
    Device_initGPIO();
    GPIO_setPadConfig(DEVICE_GPIO_PIN_LED1, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED1, GPIO_DIR_MODE_OUT);

    //
    // Initialize PIE and clear PIE registers. Disables CPU interrupts.
    //
    Interrupt_initModule();

```

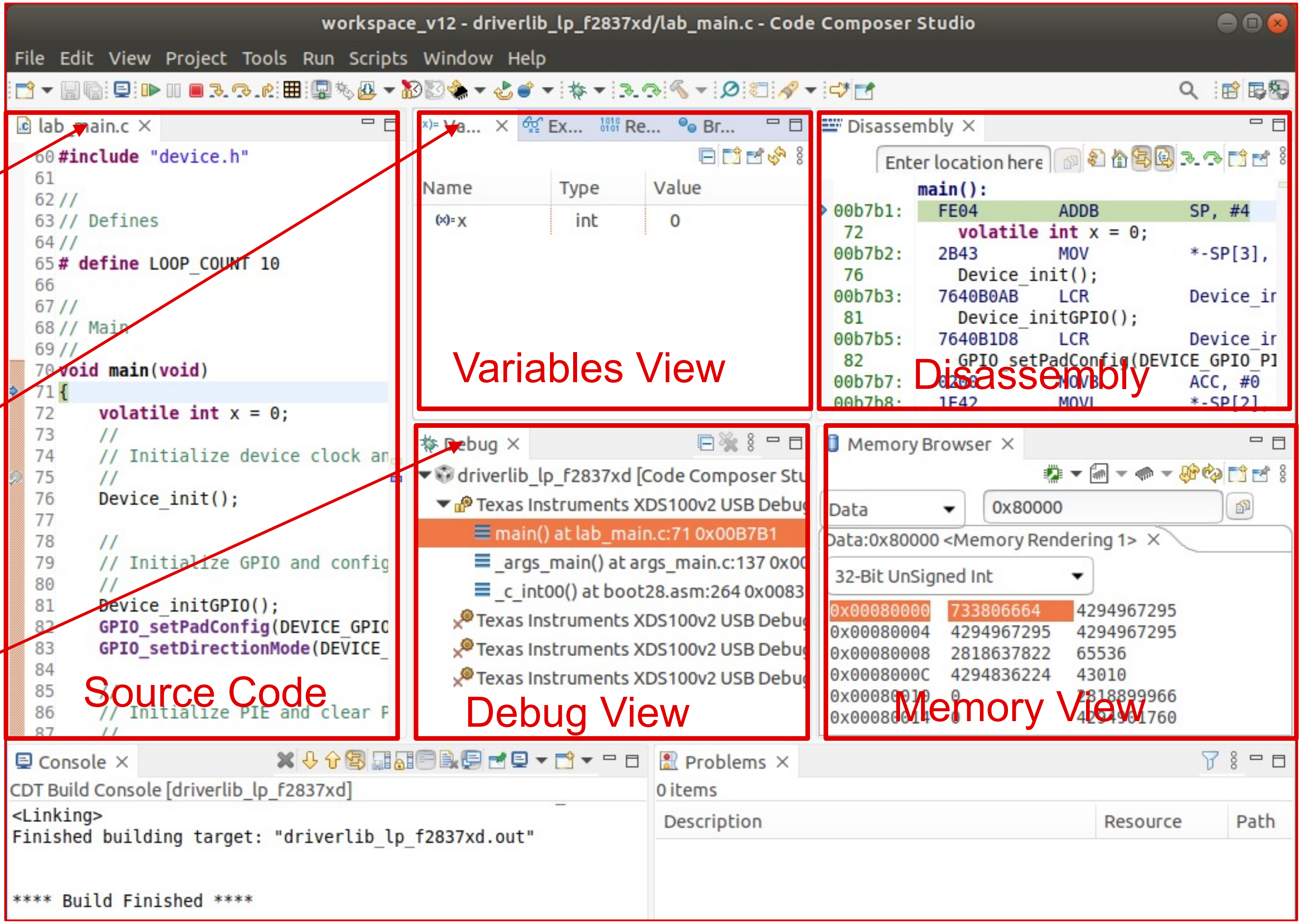
No consoles to display at this time.

0 items

Project Explorer View

After Debug Launch

- IDE opens multiple windows useful for debugging an application
- The source-code view shows the program halted at the beginning of main() function
- The **Variables**, **Expressions**, and **Registers** views are also opened by default
- **Debug** view lists all the cores on the device and call-stack for each core
- **Disassembly** and **Memory View** may also be visible if enabled



```
#include "device.h"

//
// Defines
//
# define LOOP_COUNT 10

//
// Main
//
void main(void)
{
    volatile int x = 0;
    //
    // Initialize device clock an
    //
    Device_init();

    //
    // Initialize GPIO and config
    //
    Device_initGPIO();
    GPIO_setPadConfig(DEVICE_GPIC
    GPIO_setDirectionMode(DEVICE_

    //
    // Initialize PIE and clear F
    //
```

Name	Type	Value
x	int	0

```
main():
00b7b1:   FE04        ADDB        SP, #4
 72         volatile int x = 0;
00b7b2:   2B43        MOV         *-SP[3],
 76         Device_init();
00b7b3:   7640B0AB    LCR         Device_ir
 81         Device_initGPIO();
00b7b5:   7640B1D8    LCR         Device_ir
 82         GPIO_setPadConfig(DEVICE_GPIO_PI
00b7b7:   0200        MOVB        ACC, #0
00b7b8:   1E42        MOVL        *-SP[2],
```

```
driverlib_lp_f2837xd [Code Composer Stu
  Texas Instruments XDS100v2 USB Debug
    main() at lab_main.c:71 0x00B7B1
    _args_main() at args_main.c:137 0x00
    _c_int00() at boot28.asm:264 0x0083
  Texas Instruments XDS100v2 USB Debug
  Texas Instruments XDS100v2 USB Debug
  Texas Instruments XDS100v2 USB Debug
```

Data 0x80000

Data:0x80000 <Memory Rendering 1>

32-Bit UnSigned Int

```
0x00080000  733806664   4294967295
0x00080004  4294967295  4294967295
0x00080008  2818637822  65536
0x0008000C  4294836224  43010
0x00080010  0           2818899966
0x00080014  0           4294901760
```

CDT Build Console [driverlib_lp_f2837xd]

```
<Linking>
Finished building target: "driverlib_lp_f2837xd.out"


**** Build Finished ****
```

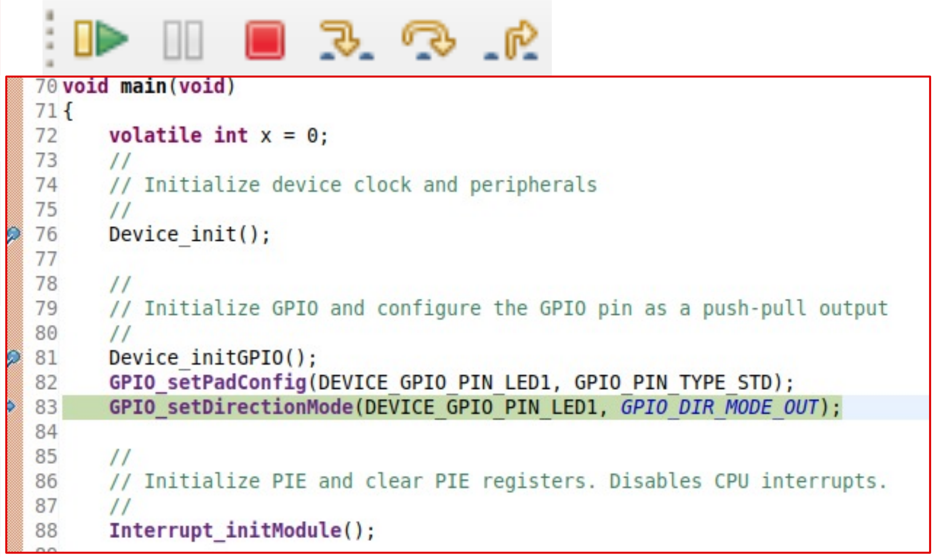
0 items

Description	Resource	Path

Debugging Commands

Basic commands needed for application debugging:

Icon	Command	Description
▶	Resume	Starts code execution
⏸	Suspend	Halts the code execution
■	Terminate	Disconnects the target and terminates Debug session
↴	Step Into	Executes a single source line, jumping into subroutines or functions
↷	Step Over	Running the code one line at a time
↱	Step Return	Run all lines until Program Counter (PC) reaches caller of the function



```
void main(void)
{
    volatile int x = 0;
    //
    // Initialize device clock and peripherals
    //
    Device_init();

    //
    // Initialize GPIO and configure the GPIO pin as a push-pull output
    //
    Device_initGPIO();
    GPIO_setPadConfig(DEVICE_GPIO_PIN_LED1, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED1, GPIO_DIR_MODE_OUT);

    //
    // Initialize PIE and clear PIE registers. Disables CPU interrupts.
    //
    Interrupt_initModule();
```

Breakpoints

Breakpoints stop code-execution and allow users to view the values of variables/expressions

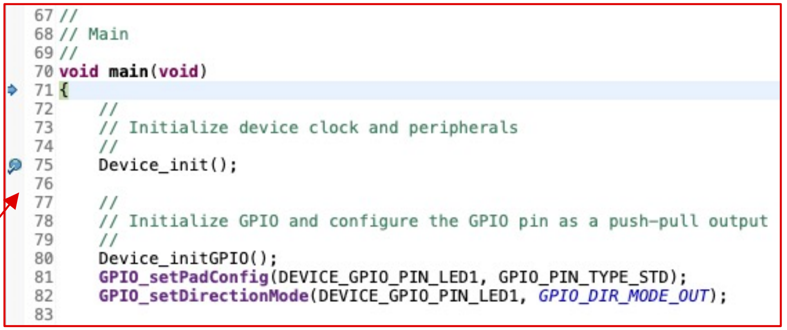
Two types of Breakpoints – Software & Hardware

How to set breakpoint:

- Double-click the shaded area in code next to the line number
- It can also be added by right-clicking in .c file and selecting breakpoint.

Software Breakpoint:

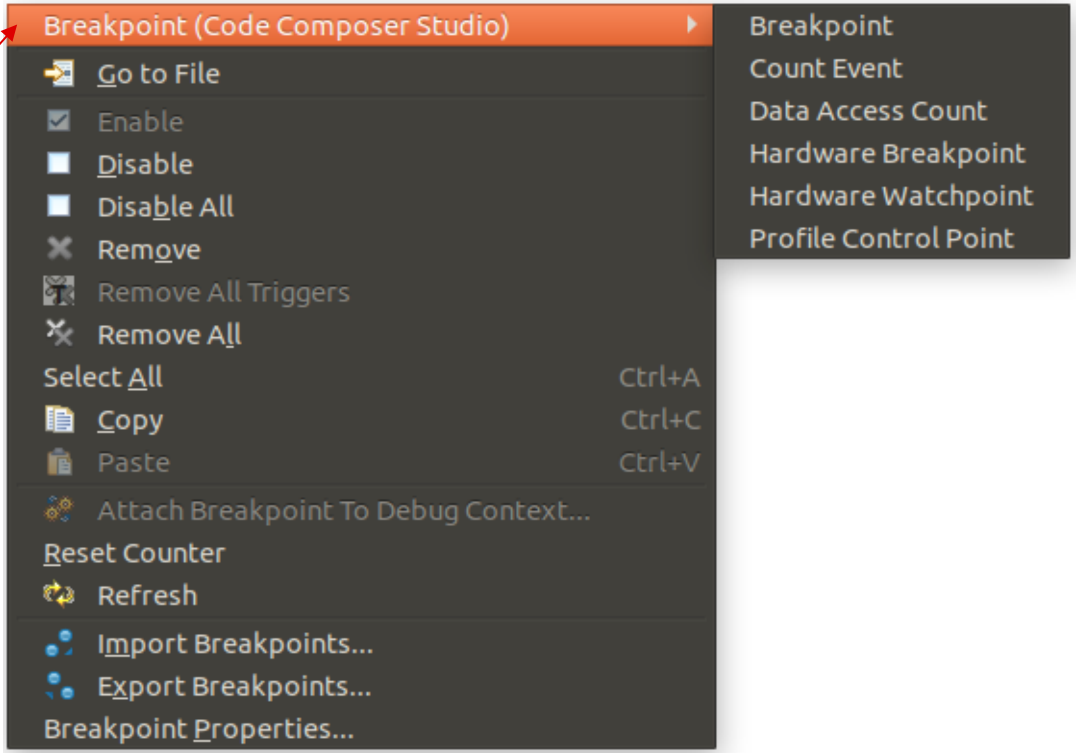
- Can only be set in memory regions with write access (RAM)
- No theoretical limit to the number of software breakpoints



```
//
// Main
//
void main(void)
{
    //
    // Initialize device clock and peripherals
    //
    Device_init();

    //
    // Initialize GPIO and configure the GPIO pin as a push-pull output
    //
    Device_initGPIO();
    GPIO_setPadConfig(DEVICE_GPIO_PIN_LED1, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED1, GPIO_DIR_MODE_OUT);

```



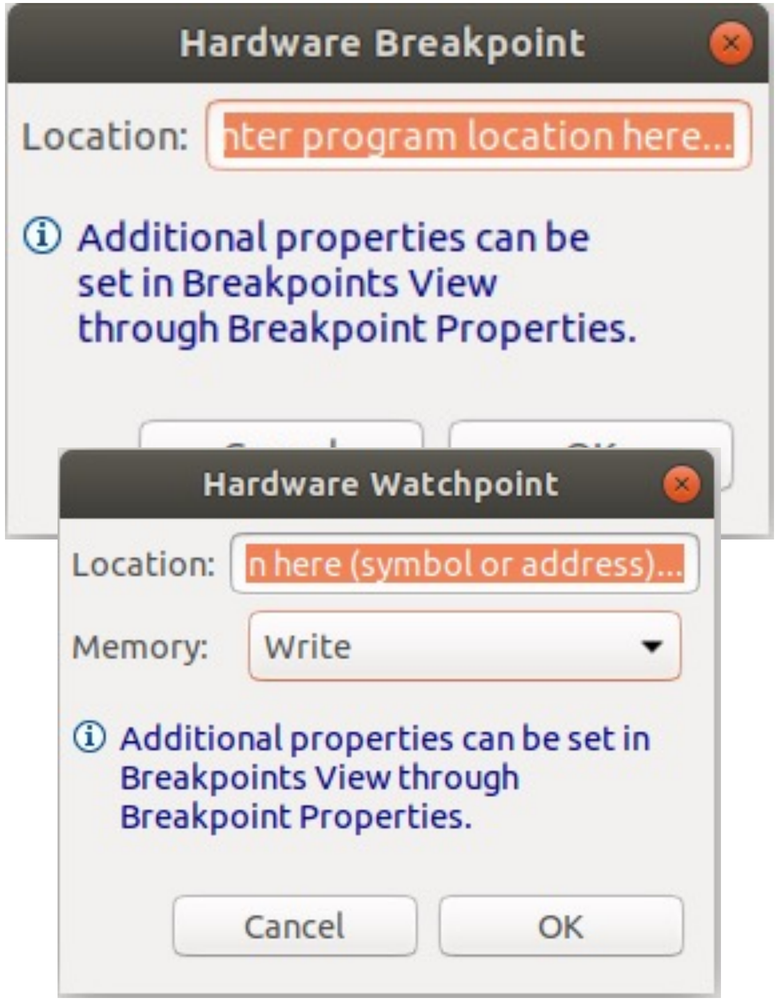
Breakpoints

Hardware Breakpoint:

- Dependent on device - implemented internally by the target hardware
- Debugger writes the address to a register on the device and sets a flag to enable breakpoints
- It can be set in any memory type - RAM, Flash, or ROM

Hardware Watchpoint:

- Special category of HW breakpoints - triggered by memory accesses instead of instruction acquisitions



Hardware Breakpoint

Location: nter program location here...

Additional properties can be set in Breakpoints View through Breakpoint Properties.

Hardware Watchpoint

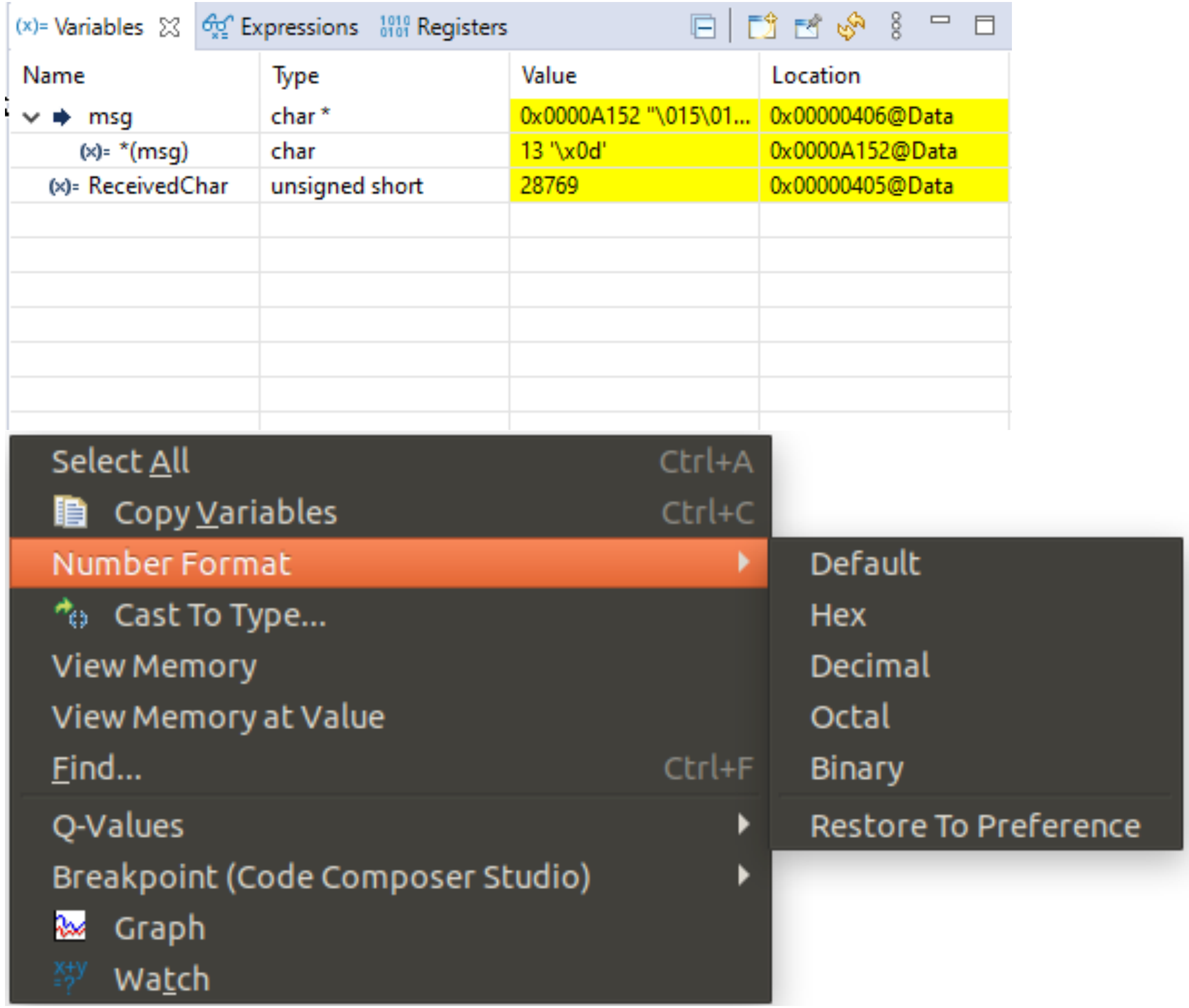
Location: n here (symbol or address)...

Memory: Write

Additional properties can be set in Breakpoints View through Breakpoint Properties.

Debug System – Variables

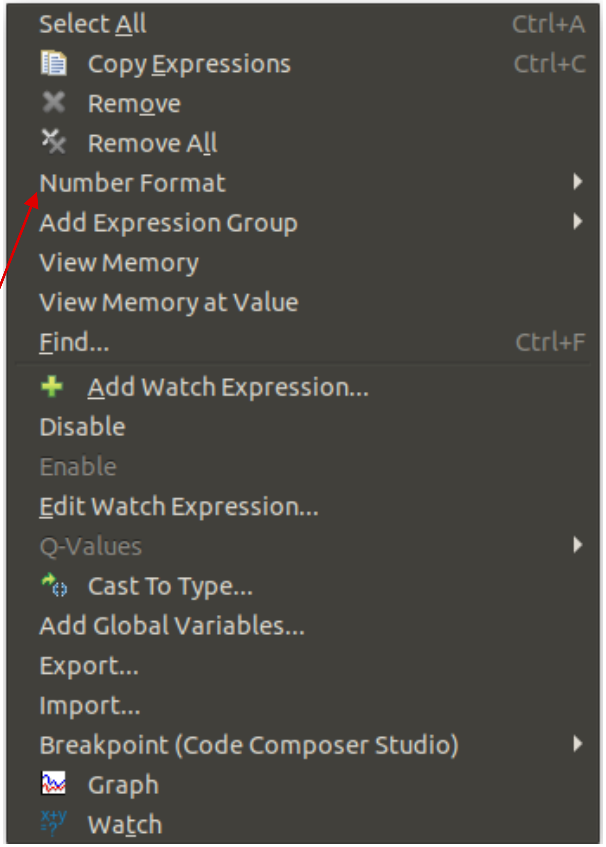
- **Local Variables** can be viewed in Variables window
- Variables whose values have changed are highlighted in a yellow background.
- The value of a variable may be modified by clicking its **Value** column and entering a new value.
- In devices that have separate program and data pages, this is suffixed by the at symbol (@) followed by the page name (Program, Data, IO).
- If the variable is allocated to a register, this field will show the word ‘Register’ followed by its register name.
- For formatting of variable, right-click on a given variable, and select desired format from context menu

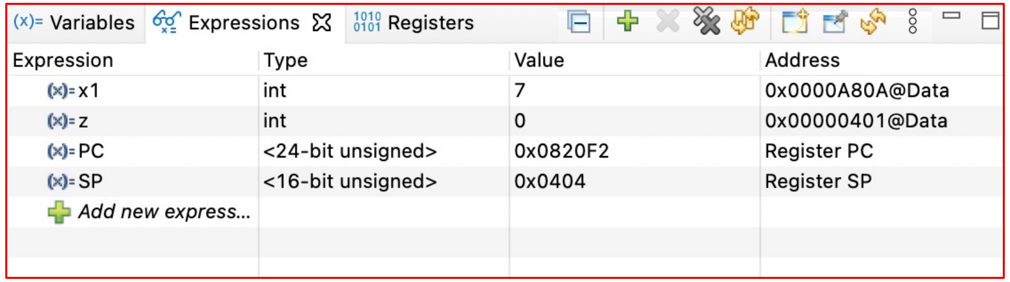


Name	Type	Value	Location
msg	char *	0x0000A152 "\015\01...	0x00000406@Data
*(msg)	char	13 '\x0d'	0x0000A152@Data
ReceivedChar	unsigned short	28769	0x00000405@Data

Debug System – Watch Expression

- Variables (local, global, static), C-valid expressions, and registers can be monitored.
- Expressions that contain more than one element, such as arrays, structures, or pointers, are displayed with either a plus sign (+) or minus sign (-) immediately preceding the expression name.
- Expressions whose values have changed since the last time they were seen are highlighted in a yellow background.
- For formatting of **expression**, right-click on a given variable, and select desired format from context menu



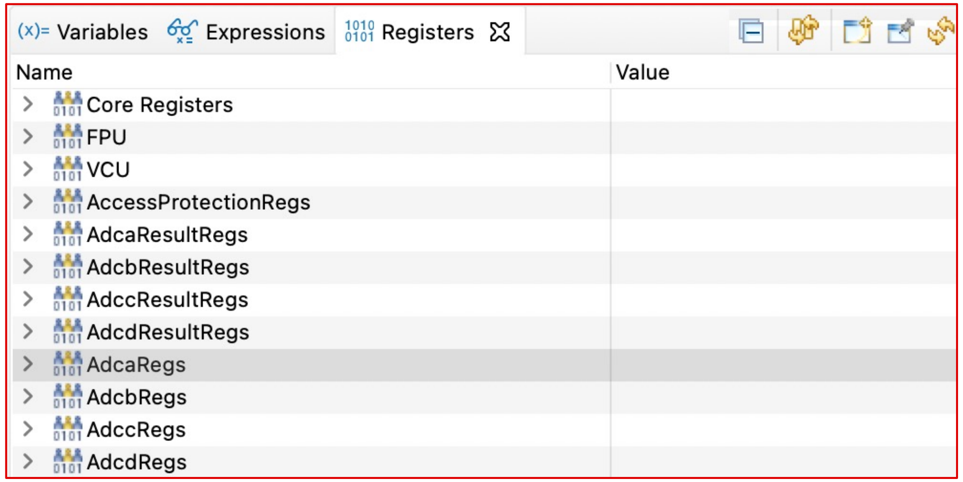


Expression	Type	Value	Address
(x)= x1	int	7	0x0000A80A@Data
(x)= z	int	0	0x00000401@Data
(x)= PC	<24-bit unsigned>	0x0820F2	Register PC
(x)= SP	<16-bit unsigned>	0x0404	Register SP
Add new express...			

Debug System – Register, Memory View

Register View

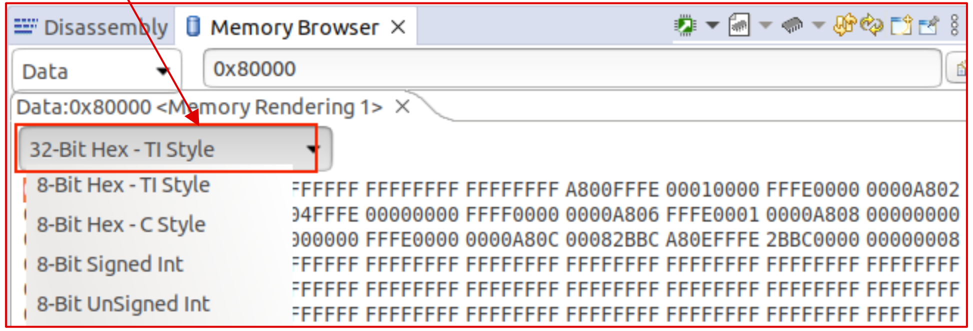
- View and edit CPU core registers
- Changed registers are highlighted in a yellow background



Name	Value
Core Registers	
FPU	
VCU	
AccessProtectionRegs	
AdcaResultRegs	
AdcbResultRegs	
AdccResultRegs	
AdcdResultRegs	
AdcaRegs	
AdcbRegs	
AdccRegs	
AdcdRegs	

Memory Browser Pane

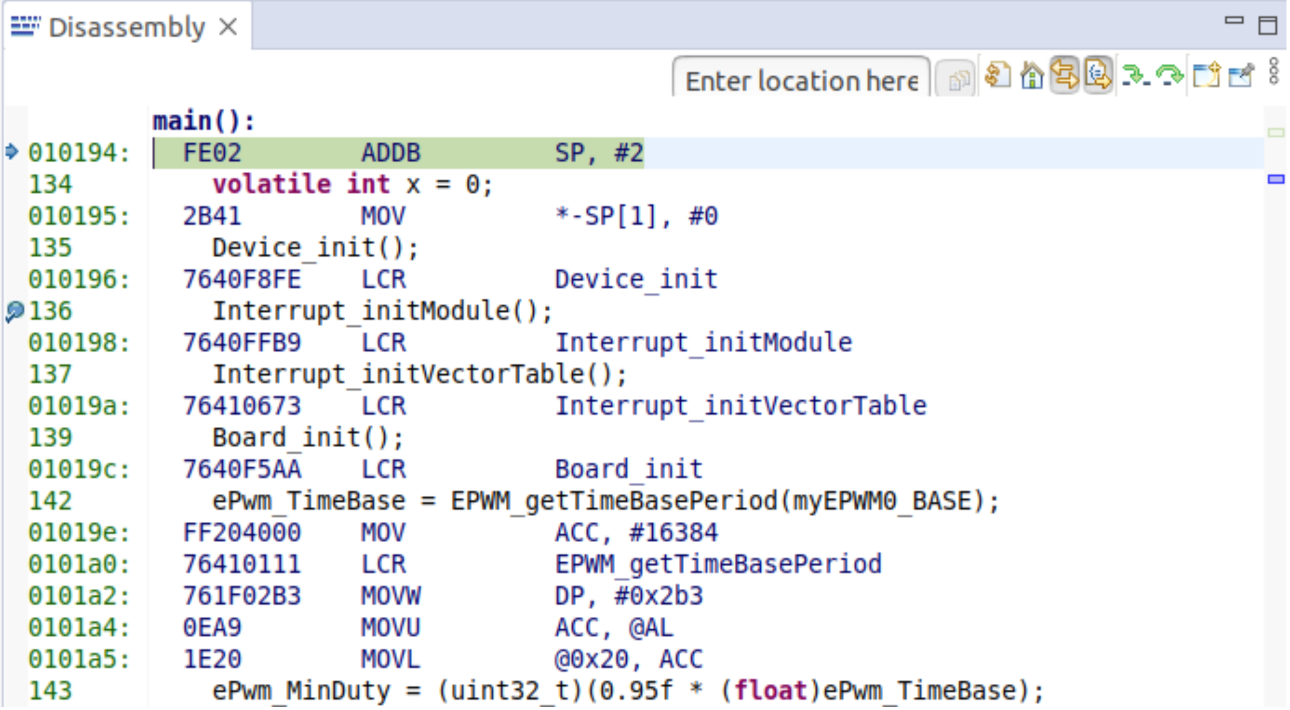
- Open by going to CCS menu **View → Memory Browser**
- Shows the contents of the target memory starting at a specified address
- Data can be viewed in different formats. Format can be selected from dropdown list



```
FFFFF FFFFFFFF FFFFFFFF A800FFFE 00010000 FFFE0000 0000A802
)4FFFE 00000000 FFFF0000 0000A806 FFFE0001 0000A808 00000000
)00000 FFFE0000 0000A80C 00082BBC A80EFFFE 2BBC0000 00000008
FFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Debug System – Disassembly View

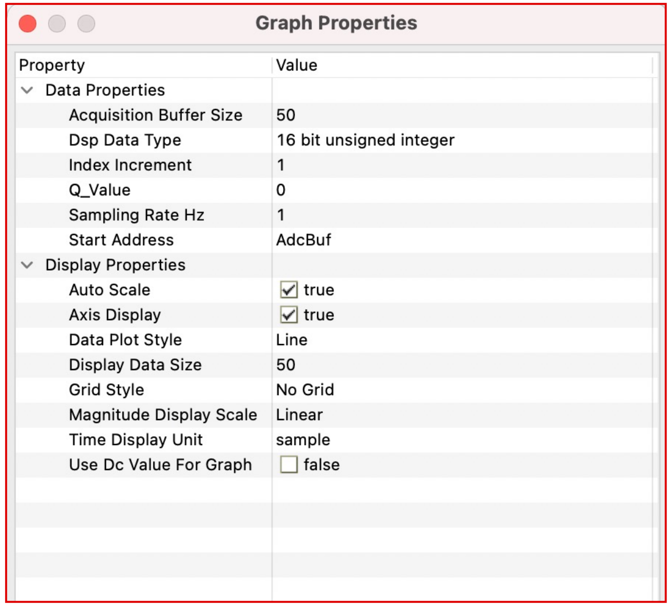
- **Disassembly** view translates machine language into assembly language.
- Displays the disassembled instructions and symbolic information needed for debugging.
- Can be viewed by going to CCS menu **View** → **Disassembly**



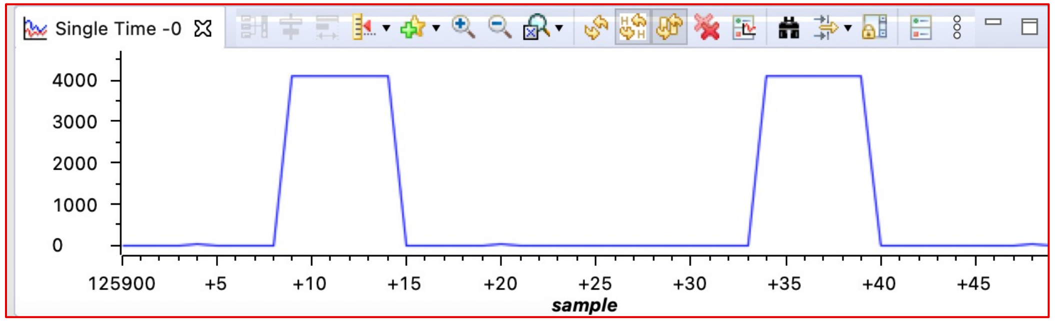
```
        main():
010194:   FE02        ADDB          SP, #2
134         volatile int x = 0;
010195:   2B41        MOV           *-SP[1], #0
135         Device_init();
010196:   7640F8FE    LCR           Device_init
136         Interrupt_initModule();
010198:   7640FFB9    LCR           Interrupt_initModule
137         Interrupt_initVectorTable();
01019a:   76410673    LCR           Interrupt_initVectorTable
139         Board_init();
01019c:   7640F5AA    LCR           Board_init
142         ePwm_TimeBase = EPWM_getTimeBasePeriod(myEPWM0_BASE);
01019e:   FF204000    MOV           ACC, #16384
0101a0:   76410111    LCR           EPWM_getTimeBasePeriod
0101a2:   761F02B3    MOVW          DP, #0x2b3
0101a4:   0EA9        MOVU          ACC, @AL
0101a5:   1E20        MOVL          @0x20, ACC
143         ePwm_MinDuty = (uint32_t)(0.95f * (float)ePwm_TimeBase);
```

CCS – Graph Tools

- An advanced graph and image visualization tool is available in CCS using CCS menu **Tools → Graph**
- Displays the data in a X-Y plot format.
- The data formatting and plotting is entirely done by the host but using the data present on the target device’s memory.
- The graph tool does not modify the data on the target memory but only fetches it via the Debug Probe connection to update its view.
- Use manual H**alt** or **Breakpoint** to update the graph or enable the option **Continuous update** in the **Graph Toolbar**.


Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	50
Dsp Data Type	16 bit unsigned integer
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	AdcBuf
Display Properties	
Auto Scale	☑ true
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	50
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	□ false



To find more Microcontroller and Processor technical resources and search products, visit **https://www.ti.com/microcontrollers**.