

# XMC Dev Day 2015

Hands-on with DAVEv4 & XMCLib

Alberto Squarcina IFI ATV SMD EMEA  
Alessandro Manfre IFI ATV SMD EMEA



# Agenda

- LLD of XMCLib: use of ACMP
- Blink an LED with PWM APP
- Use ADC APP to update PWM dutycycle

# Agenda

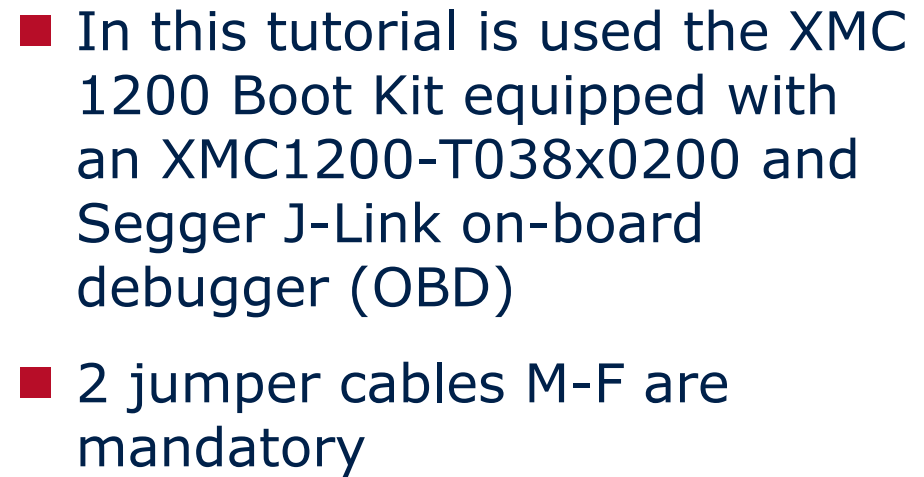
- LLD of XMCLib: use of ACMP

- Blink an LED with PWM APP

- Use ADC APP to update PWM dutycycle

# Summary

- Required XMC kit
- HOT Session: Key points
- ACMP: details and connections
- Starting DAVE
- Work with LLD: overview
- Work with LLD: details
- Digital IO configuration & Alternate Functions



# HOT Session: Key points

- The target for the example of HOT session is to use the ACMP peripheral, in combination with a Digital IO
- The example is made mixing the use of XMCLib by LLD and the APPs of DAVE4
- The example is made in order to see the Alternate Functions (both as Input or Output) of each pin
- Output of the example is the switch ON/OFF of a LED accordingly to the output of ACMP

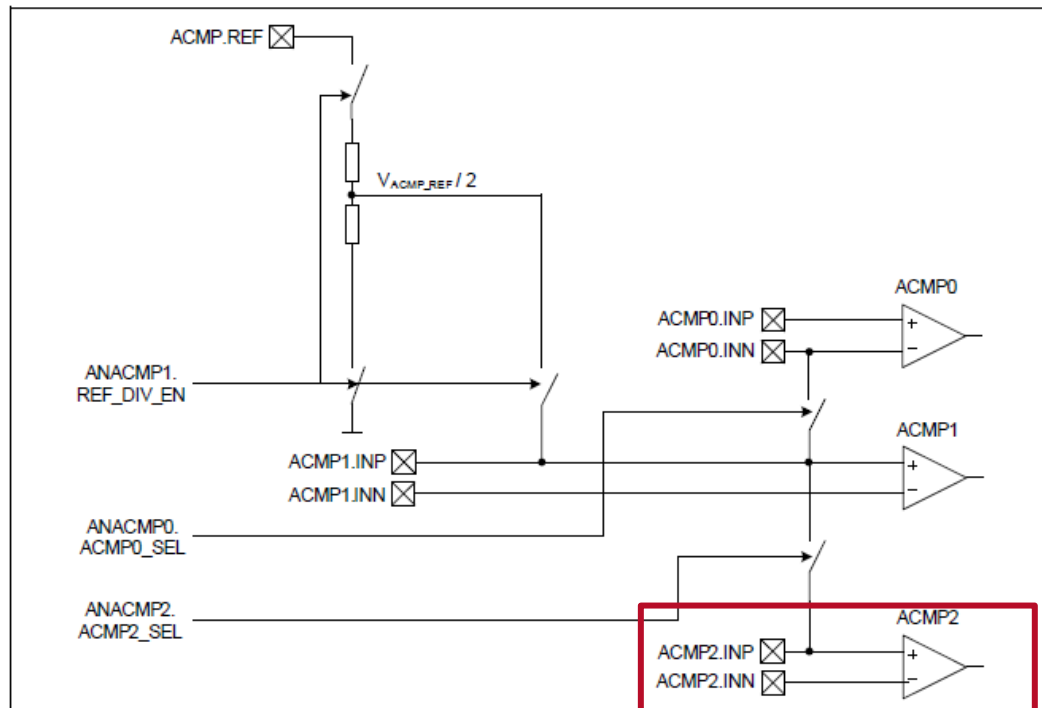
# ACMP: details [1/2]

- In the XMC1200 there are up to 3 analog comparators.  
In this session ACMP2 will be used

- Pins used

- Inputs: P2.1,P2.2

- Output: P0.5



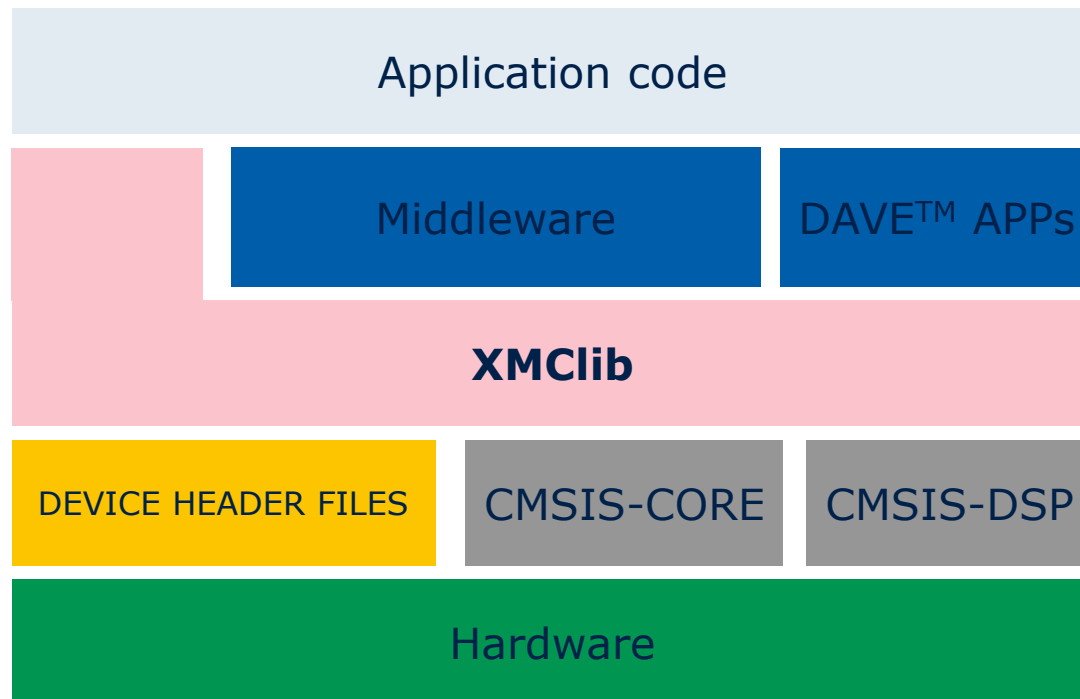
# ACMP: details [2/2]

Input/Output	I/O	Connected To	Description
ACMP0.INN	I	P2.8	"-" input of ACMP0
ACMP0.INP	I	P2.9	"+" input of ACMP0
ACMP1.INN	I	P2.6	"-" input of ACMP1
ACMP1.INP	I	P2.7	"+" input of ACMP1
ACMP2.INN	I	P2.2	"-" input of ACMP2
ACMP2.INP	I	P2.1	"+" input of ACMP2
ACMP.REF	I	P2.11	Reference input of ACMP
ACMP0.OUT	O	P0.10 P2.10 ERU0.0A0 BCCU0.IN5	output of ACMP0
ACMP1.OUT	O	P1.0 ERU0.1A0 BCCU0.IN0 P2.5.HW0 pull control	output of ACMP1
ACMP2.OUT	O	P0.5 P1.2 ERU0.2A0 BCCU0.IN3 P2.3.HW0 pull control	output of ACMP2

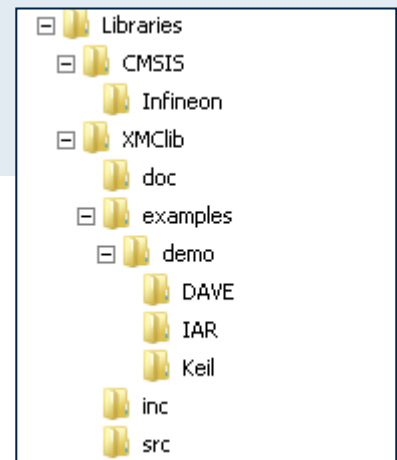


# Work with LLD: overview

- XMC Lib consists of low-level drivers for the XMC product family peripherals
- Collection of routines and data structures which covers all peripheral functions
- Peripheral register abstraction by a set of stateless APIs

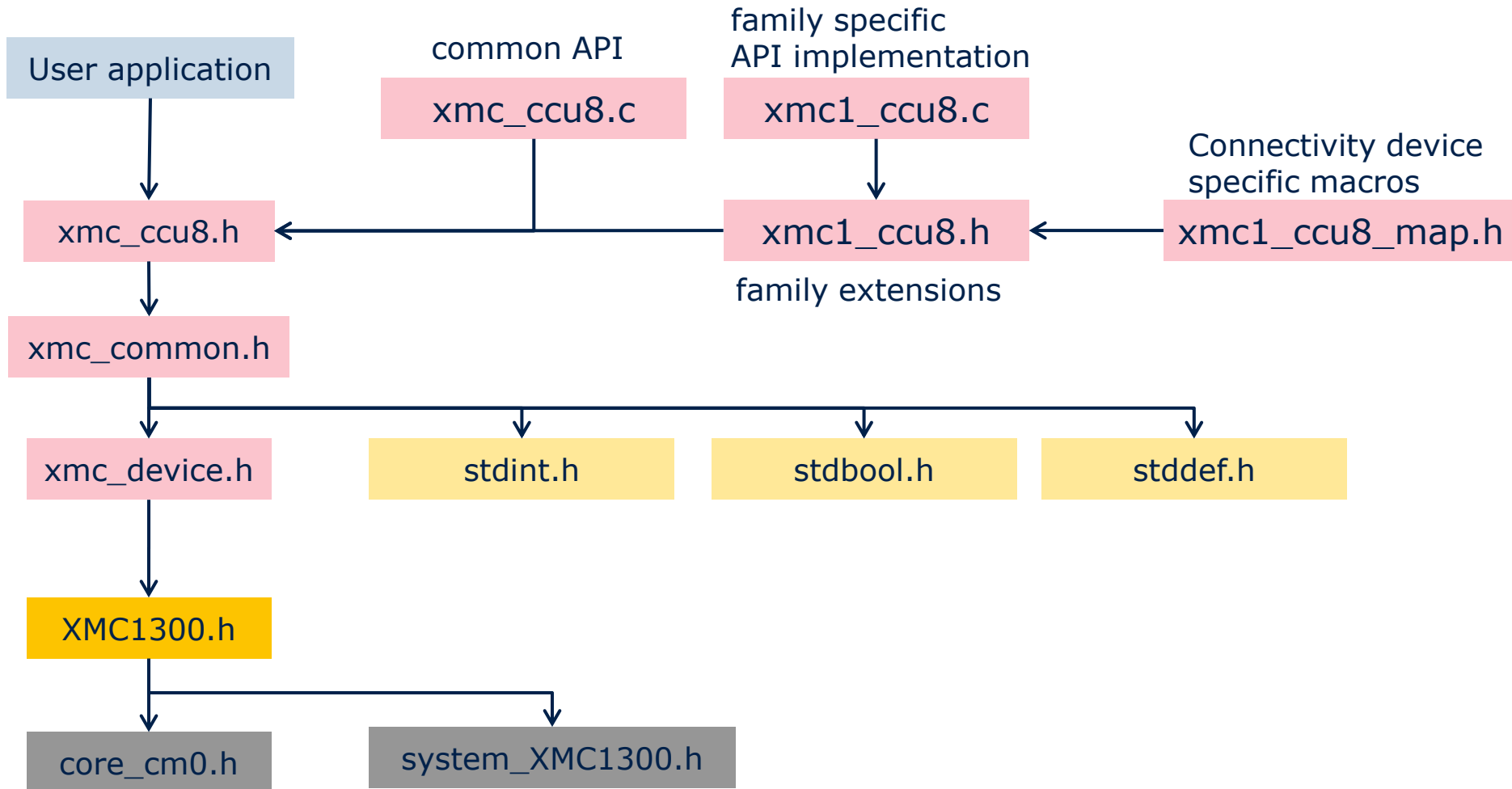


# Package Description



- **inc** folder contains peripheral API header files
  - *xmc\_common.h*: common functionality and definitions
  - *xmc\_device.h*: device selection
  - *xmc\_<peripheral>.h* (one header file per peripheral): Function prototypes, data structures and enumeration
  - *xmc<family>\_<peripheral>.h* (optional): Family extensions
- **src** folder contains peripheral API implementation
  - *xmc\_common.c*: common functionality and definitions
  - *xmc\_<peripheral>.c*: Implementation of common peripheral functionality
  - *xmc<family>\_<peripheral>.c* (optional): Family specific implementation
- **doc** folder contains API documentation generated from sources
- **examples** folder contains extensive set of examples projects for different toolchains

# Package Description



# Coding rules and conventions

- C99, in addition unions and bitfields are used for more compact code and data section.
- Use only standard data types
- Use enumerations
- Naming convention:
  - Variables use only lower case, underscore separated words.
  - Functions use CamelCase convention.
- Non blocking APIs.
- Runtime error checking can be enabled by user
  - XMC\_ASSERT() used to check input parameters of functions, result of calculations, ...
  - XMC\_DEBUG() used to monitor status of application

# Coding rules and conventions

Object = attributes + methods

Attributes are the peripheral data struct.

Methods are functions that take a pointer to the peripheral data struct as the first argument.

XMC Lib uses naming conventions to bind the data struct and the functions that operate on it

**XMC\_PERI**\_DoSomething(XMC\_PERI\_t const \*peri, ...);

Each driver defines its own "namespace", XMC\_PERI.

# XMC Lib APIs

- Each driver has an API to get at runtime the driver version

```
XMC_DRIVER_VERSION_t XMC_PERI_GetDriverVersion(void);
```

- Each driver has an initialization function

```
XMC_PERI_STATUS_t  
XMC_PERI_Init(XMC_PERI_t const *peri,  
              const XMC_PERI_CONFIG_t const *config);
```

- Most drivers have enable/disable function

```
void XMC_PERI_Enable(XMC_PERI_t const *peri);  
void XMC_PERI_Disable(XMC_PERI_t const *peri);
```

# XMC Lib APIs

## ■ Event handling

```
void XMC_PERI_Enable(XMC_PERI_t const *peri);  
void XMC_PERI_Disable(XMC_PERI_t const *peri);  
void XMC_PERI_EnableEvent(XMC_PERI_t const *peri);  
void XMC_PERI_DisableEvent(XMC_PERI_t const *peri);  
void XMC_PERI_TriggerEvent(XMC_PERI_t const *peri);  
uint32_t XMC_PERI_GetEventStatus(XMC_PERI_t const *peri);  
void XMC_PERI_SetInterruptNode(XMC_PERI_t const *peri);
```

## ■ Control

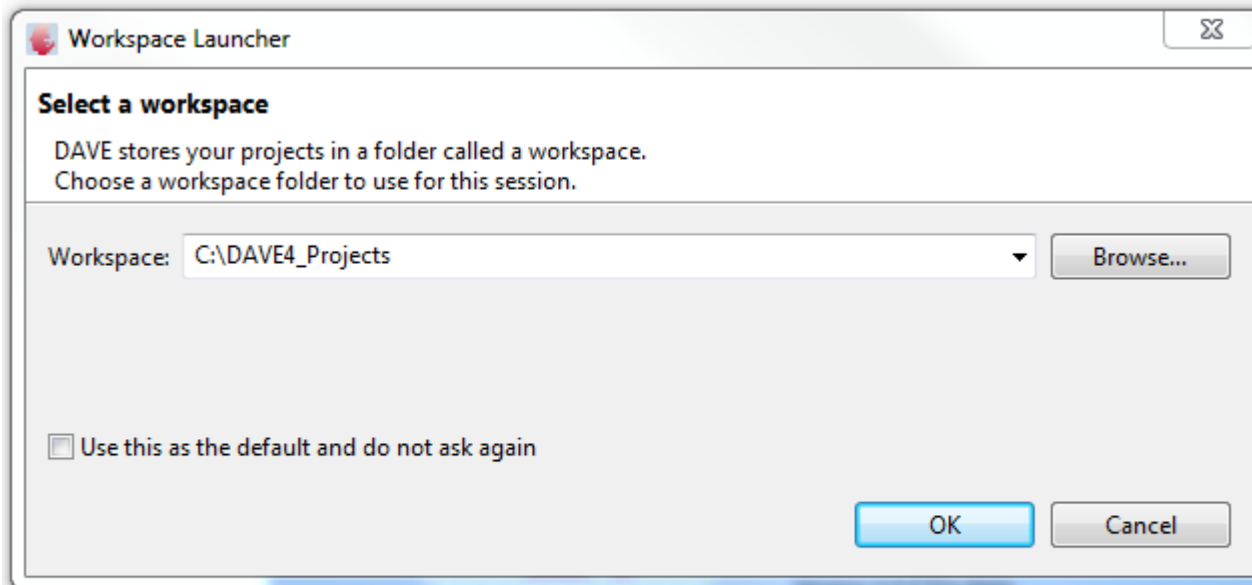
```
void XMC_PERI_Start(XMC_PERI_t const *peri);  
void XMC_PERI_Stop(XMC_PERI_t const *peri);  
void XMC_PERI_Suspend(XMC_PERI_t const *peri);  
void XMC_PERI_Resume(XMC_PERI_t const *peri);
```

## ■ Get/setters (run time API to get or modify state of peripheral)

```
void XMC_PERI_SetSomething(XMC_PERI_t const *peri, ...)  
uint32_t XMC_PERI_GetSomething(XMC_PERI_t const *peri, ...)
```

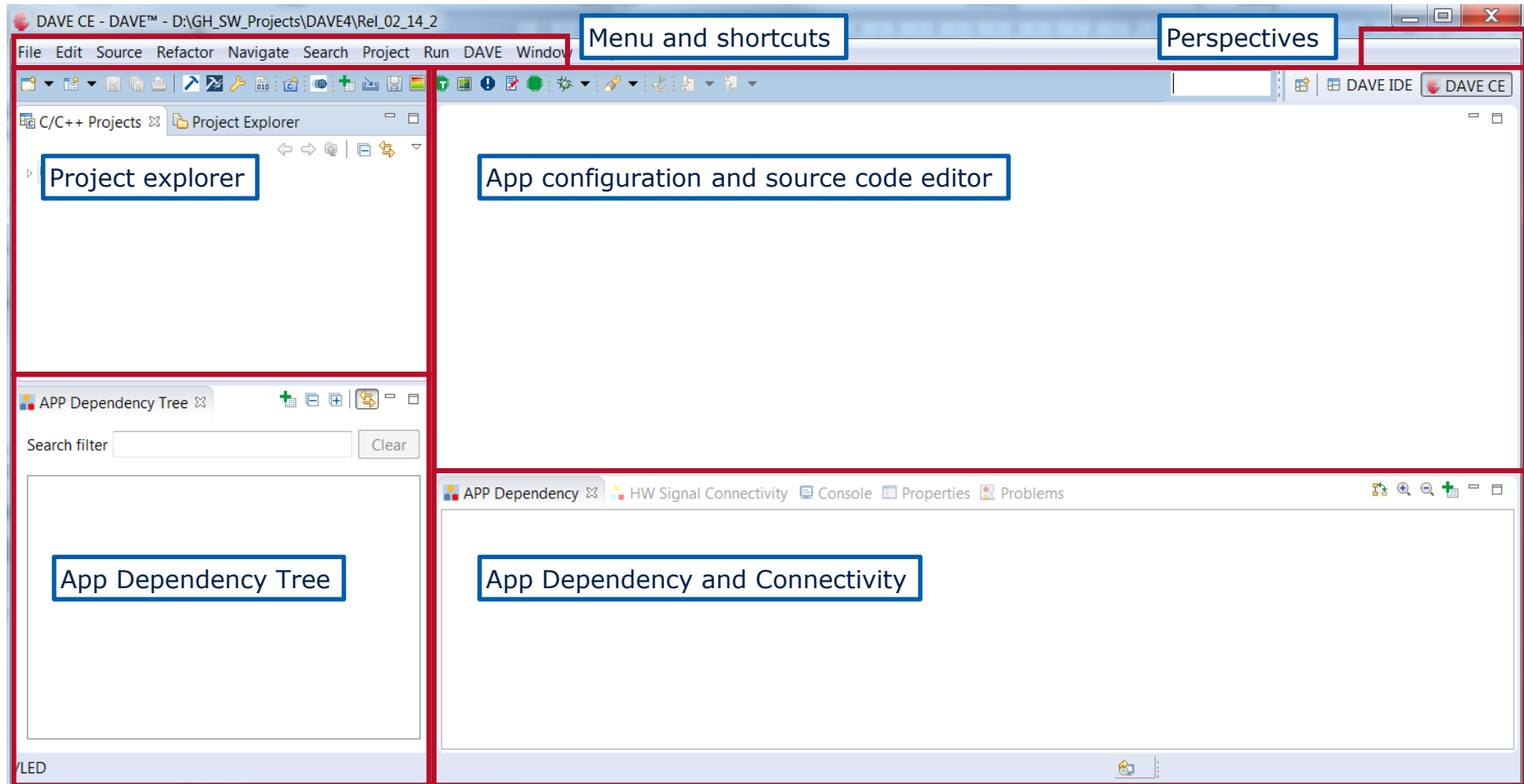
# Starting DAVE™ for the first time

- Start DAVE
- Enter path to workspace folder
  - Please chose a new workspace folder, not an existing workspace folder form an earlier DAVE™ version

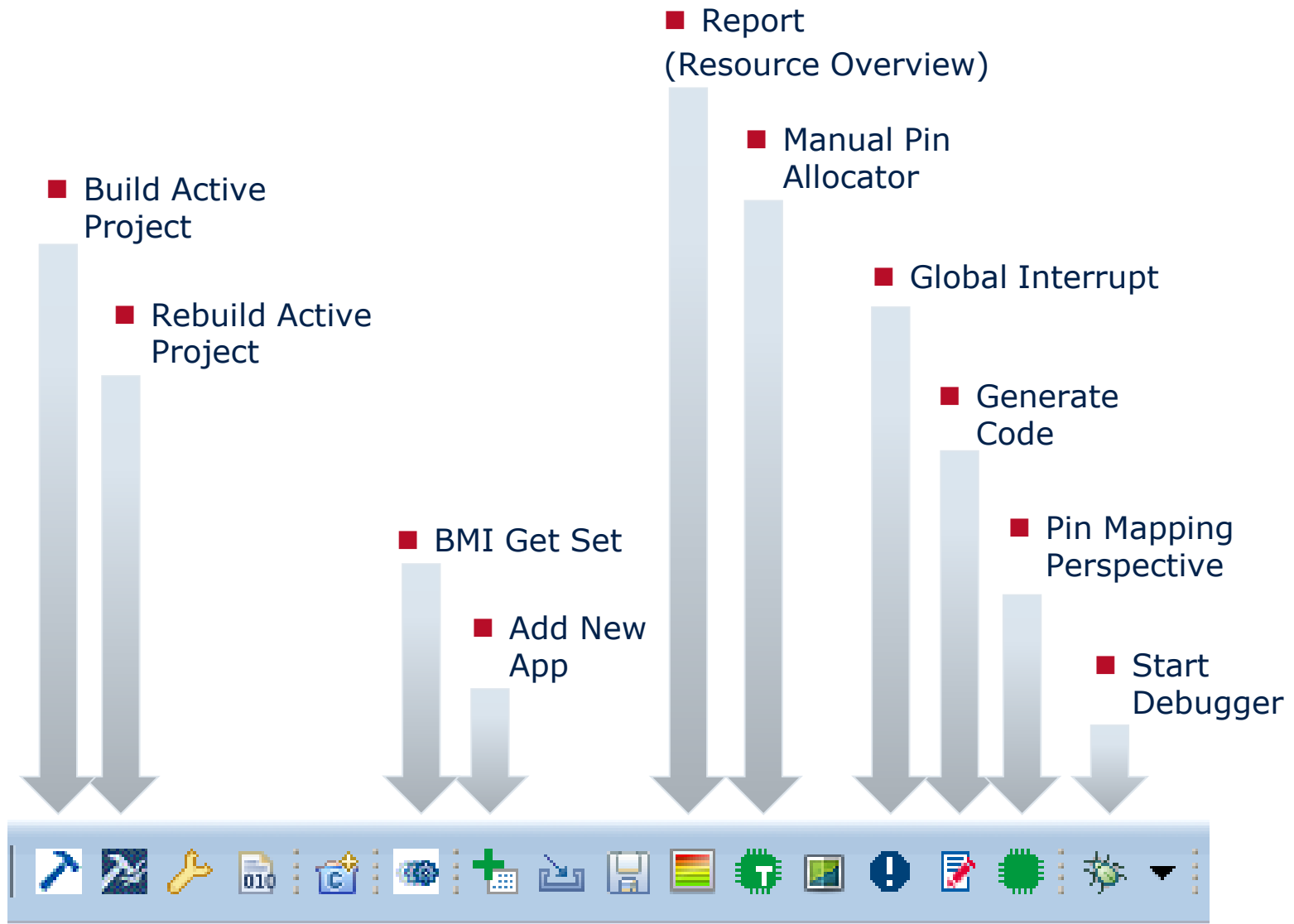




# DAVE™ CE Workspace



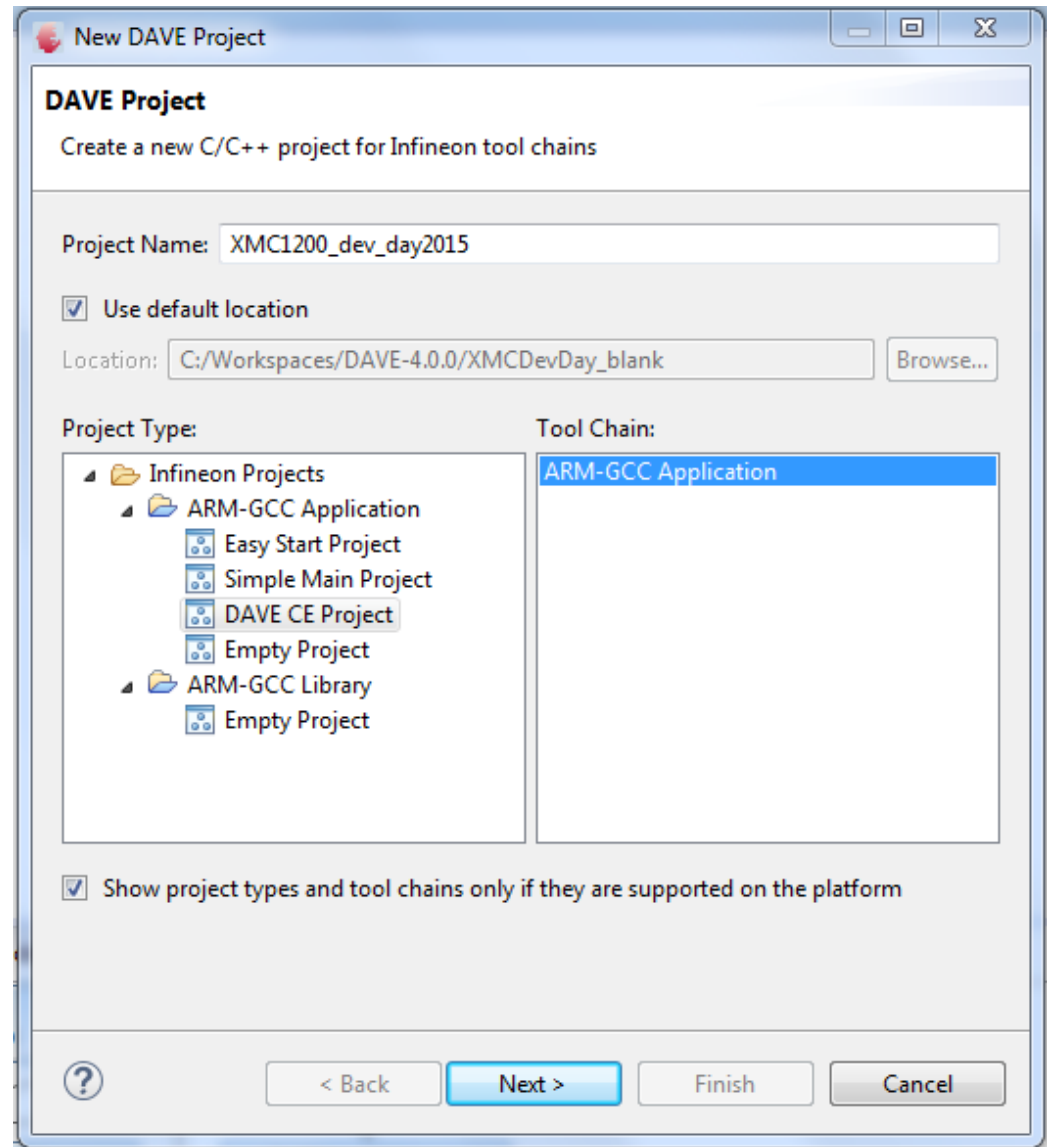
# Tool Panel



# How to create a DAVE™ CE Project [1/2]

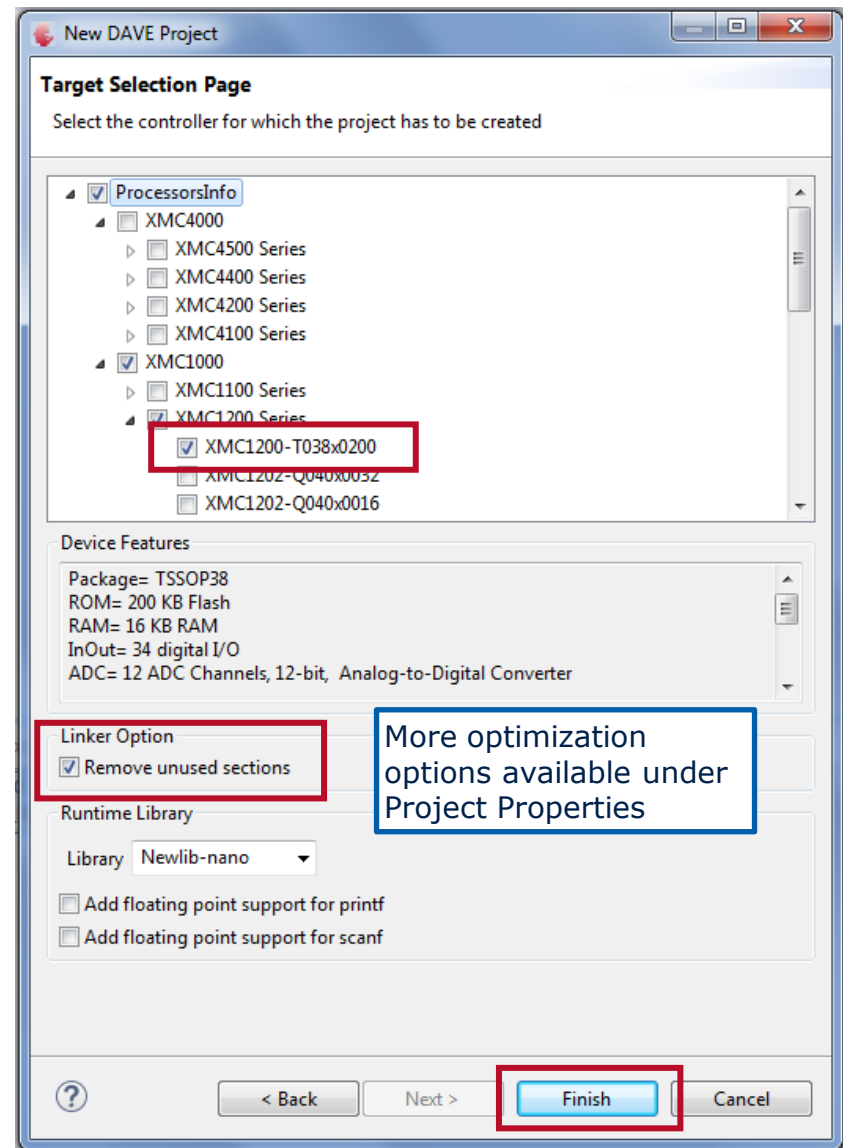
## ■ Create DAVE™ Code Engine (CE) Project

1. Go to File → New → DAVE Project
2. Select DAVE CE Project
3. Type a Project Name
4. Click Next

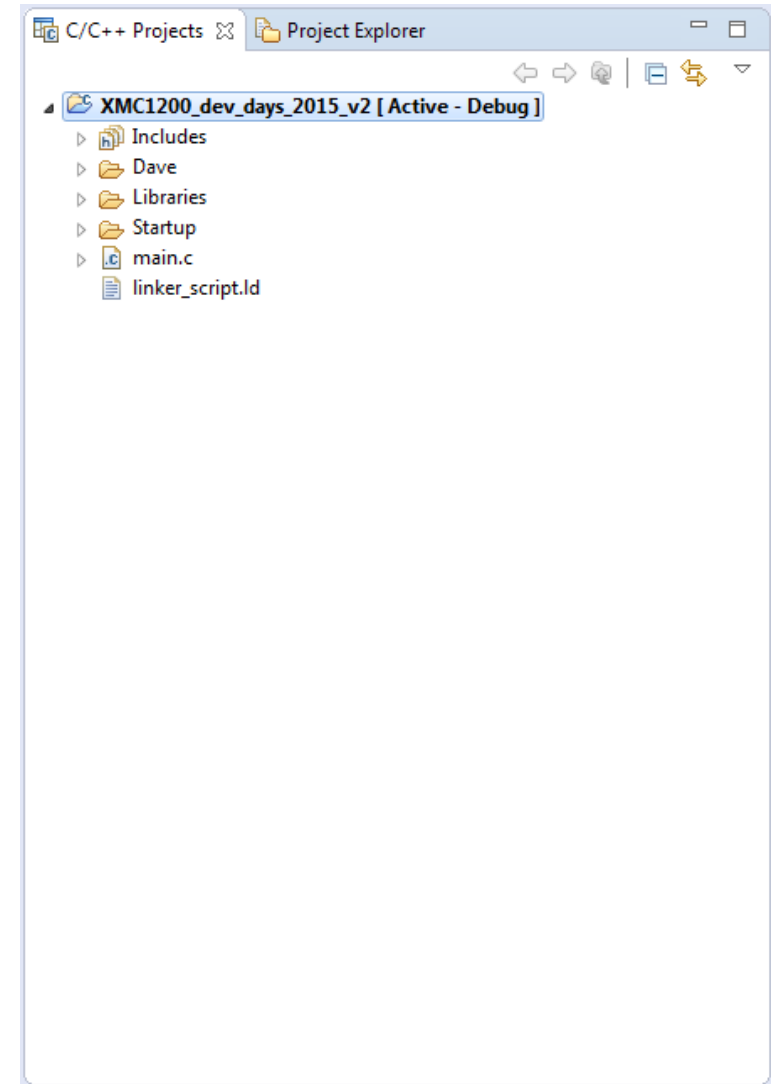
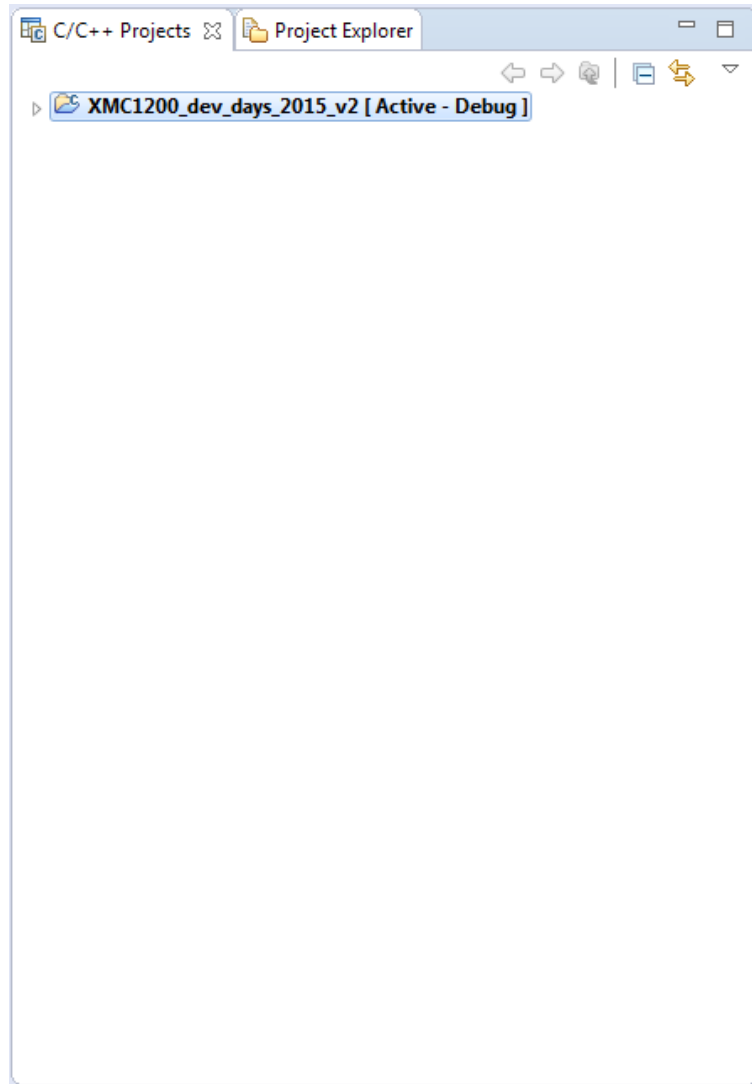


# How to create a DAVE™ CE Project [2/2]

- Select the appropriate microcontroller
- In this case
  - XMC1200-T038x0200
- Click Finish



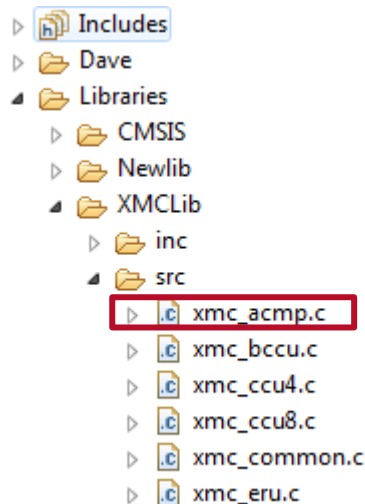
# Project View



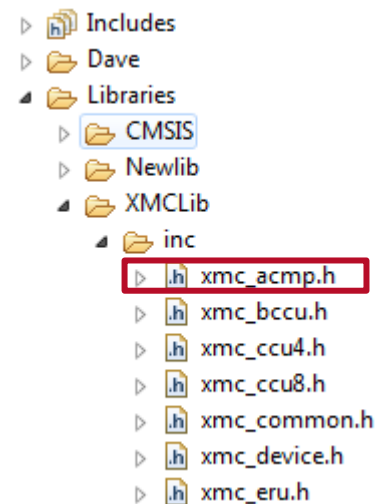
# Work with LLD: project details [1/2]

## ■ XMCLib is always available in each project

### □ Source files



### □ Header files



# Work with LLD: project details [2/2]

## LLD Code for ACMP



```
#include <XMC1200.h>
#include <DAVE.h>           //Declarations from DAVE Code Generation (includes SFR declaration)
#include <xmc_acmp.h>

#define ACMP_SLICE_NUMBER (2)

/* Hardware reference to ACMP Slice */
XMC_ACMP_t *peripheral;

/* ACMP Slice configuration */
XMC_ACMP_CONFIG_t g_acmp_config = {
    .filter_disable = 0U,
    .output_invert = 0U,
    .hysteresis = XMC_ACMP_HYSTERESIS_20
};

int main(void)
{
    DAVE_STATUS_t status;

    peripheral = COMPARATOR;

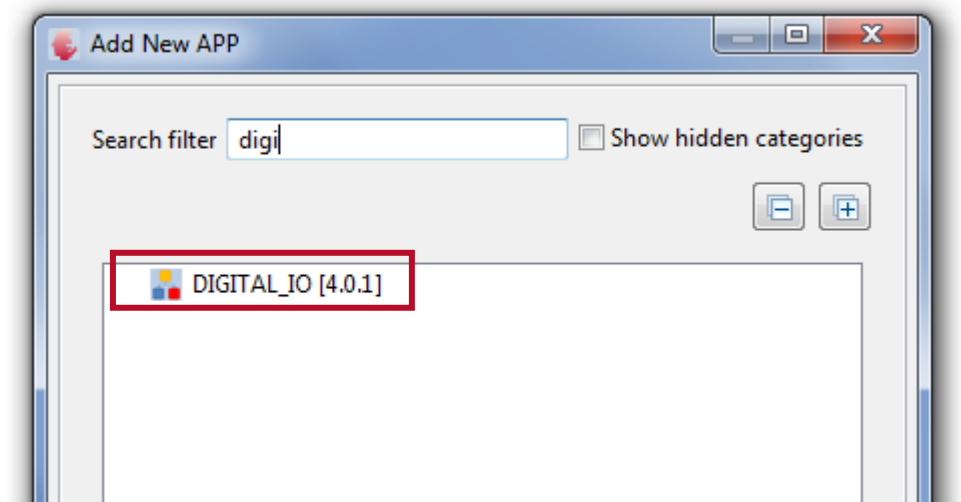
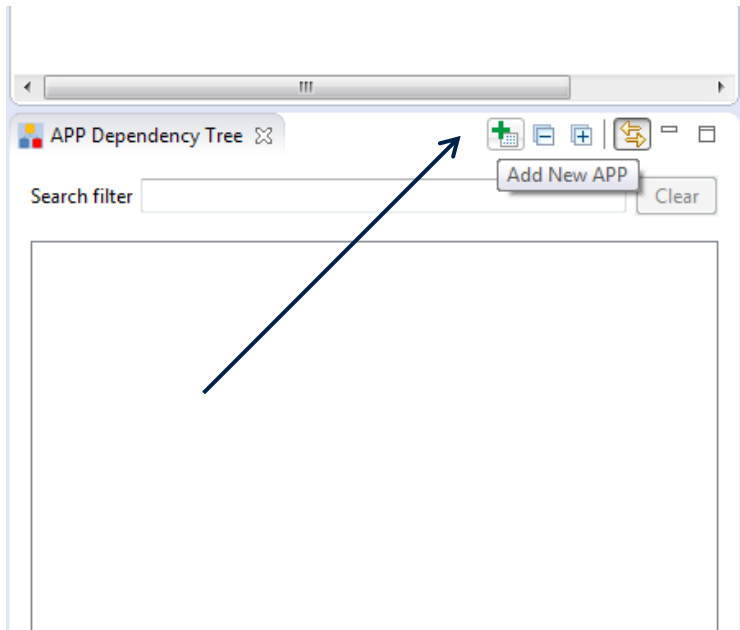
    /* Enable and Initialize ACMP Slice-2 */
    XMC_ACMP_EnableComparator(peripheral, ACMP_SLICE_NUMBER);
    XMC_ACMP_Init(peripheral, ACMP_SLICE_NUMBER, &g_acmp_config);

    XMC_ACMP_SetInput(peripheral, ACMP_SLICE_NUMBER, XMC_ACMP_INP_SOURCE_STANDARD_PORT);

    status = DAVE_Init();    /* Initialization of DAVE Apps */
}
```

# Digital IO configuration [1/3]

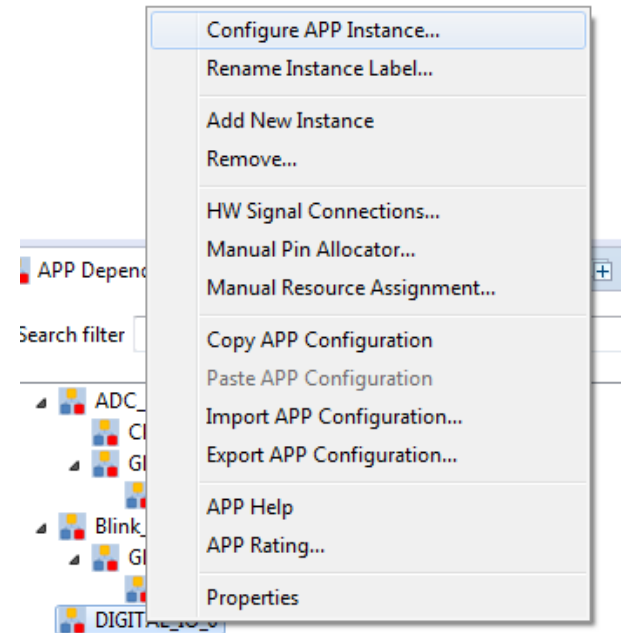
- The next step for the session is to configure a Digital IO pin as output of ACMP2 (P0.5) by the use of relative APP of DAVE
  - Click on "Add new APP" on bottom left menu
  - Type "digital" on Search filter field
  - Double click on DIGITAL\_IO



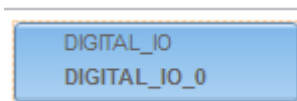


## Digital IO configuration [2/3]

- Perform right-click on App name in the App Dependency Tree view
- Select Configure APP Instance

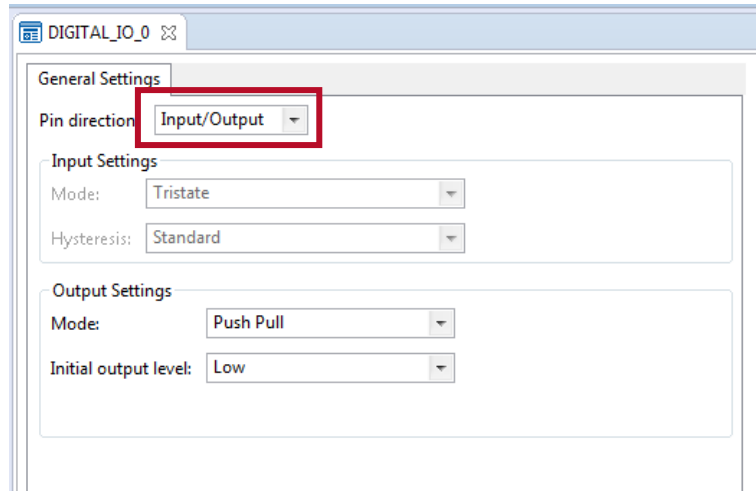


- Second Possibility:
- Double-click on App Name in App Dependency View



# Digital IO configuration [3/3]

## ■ Set pin as Input/Output



DIGITAL\_IO\_0

General Settings

Pin direction: **Input/Output**

Input Settings

Mode: Tristate

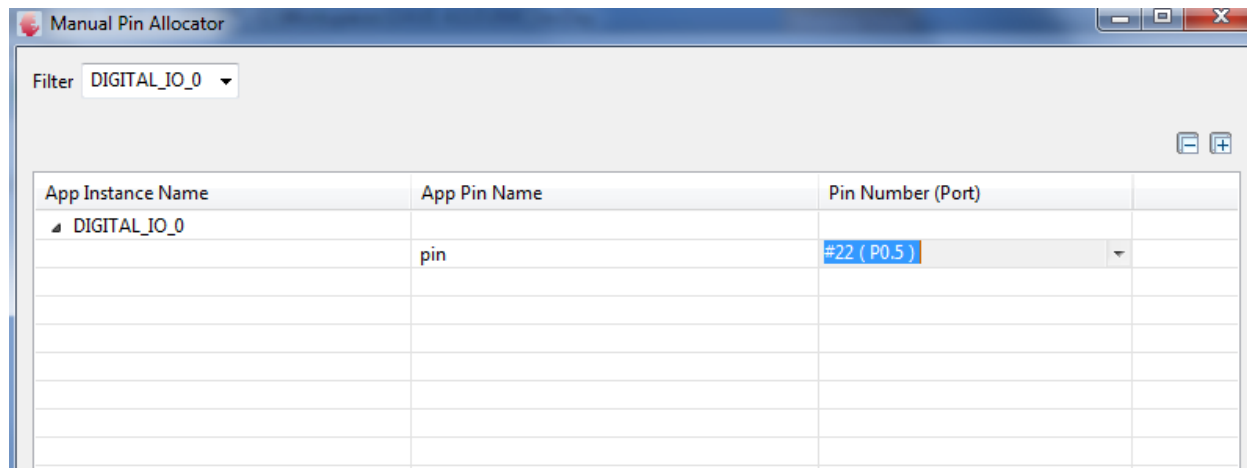
Hysteresis: Standard

Output Settings

Mode: Push Pull

Initial output level: Low

## ■ Allocate P0.5 as output by “Manual Pin Allocator” menu




Manual Pin Allocator

Filter: DIGITAL\_IO\_0

App Instance Name	App Pin Name	Pin Number (Port)
▲ DIGITAL_IO_0	pin	#22 (P0.5)

# DAVE: Generation code

- One touch code generation

1. Click  in the tool panel
2. Generated code can be found under C/C++ Projects window, DAVE → Generated

- Start Compiler tools to build the project

- Click  in the tool panel

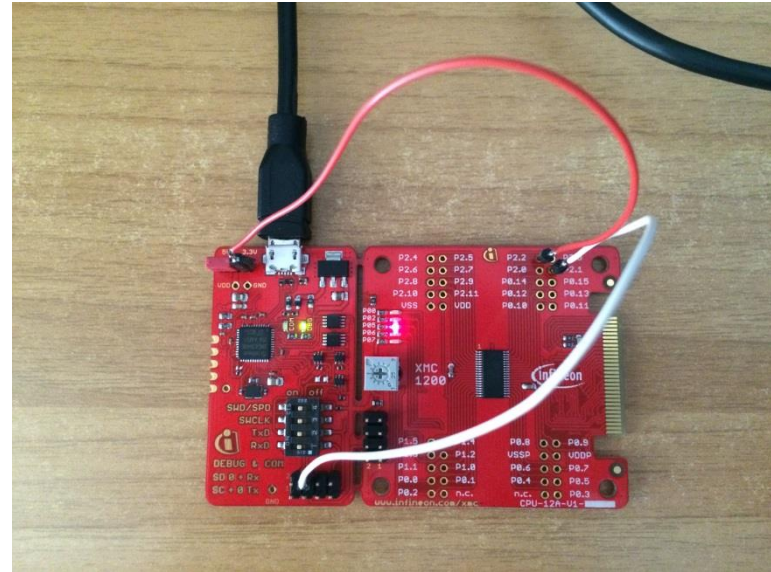
- Ensure that Compiler finished building in Console window

# ACMP: Hardware connections

## ■ Case 1:

ACMP2.INN 3.3V (P2.2)

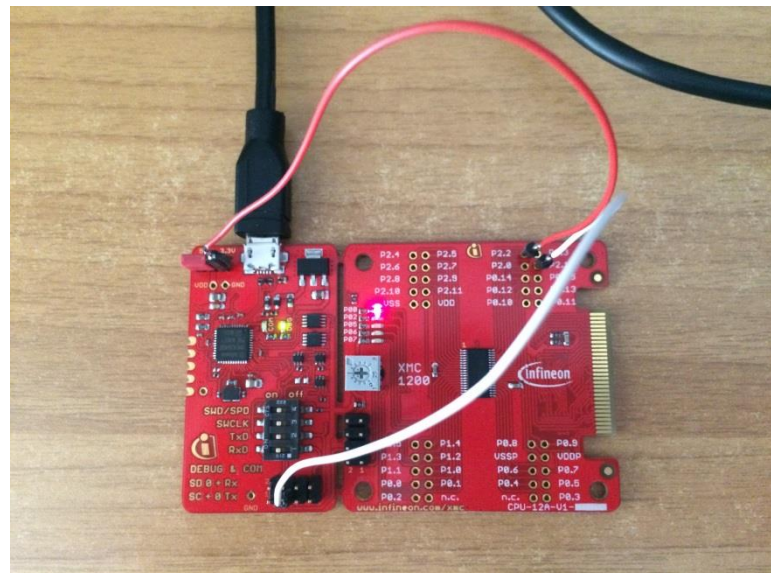
ACMP2.INP 0V (P2.1)



## ■ Case 2:

ACMP2.INN 3.3V (P2.2)

ACMP2.INP 5V (P2.1)



# DAVE: Flash and Debug [1/2]

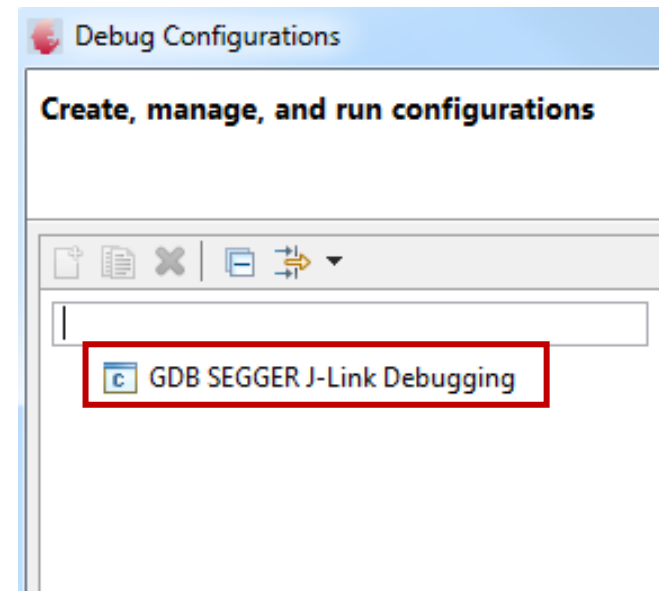
## ■ Start Debug Session

- Click  in the tool panel

## ■ Create a new Debug Configuration

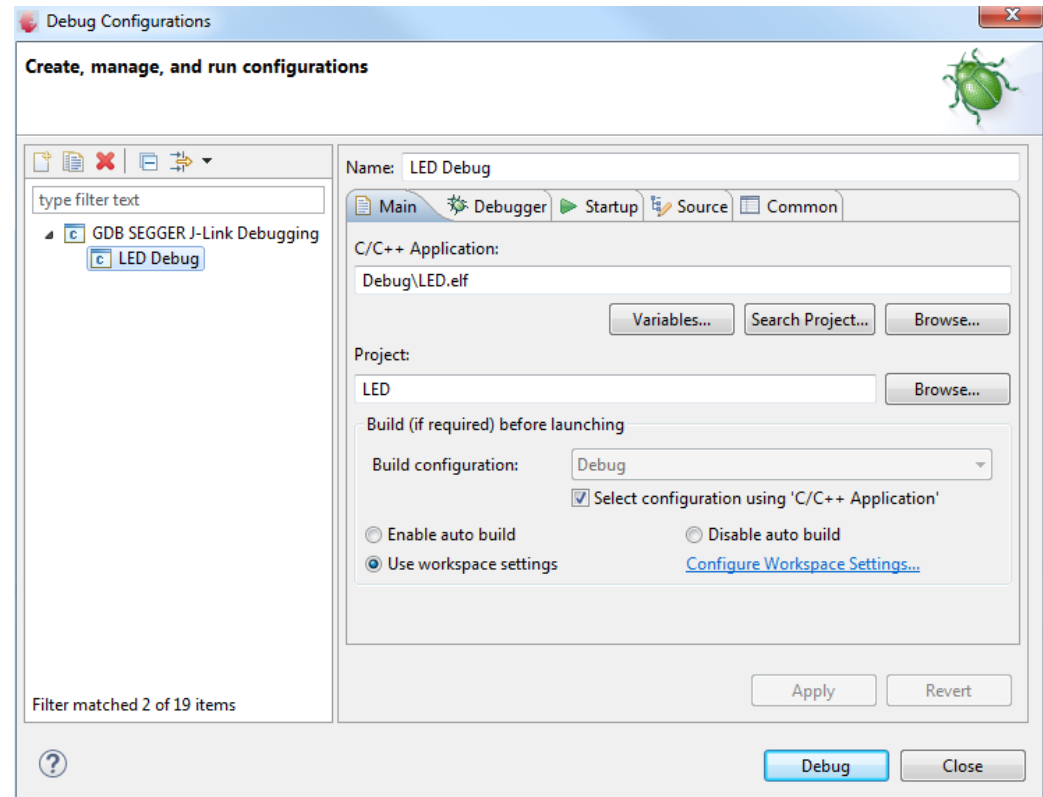
- Double-click  
"GDB SEGGER J-Link Debugging"

Segger J-link Driver software  
4.96h or above needs to be  
installed



# DAVE: Flash and Debug [2/2]

- Click "Debug"
- The flashing process is started and DAVE automatically switches to Debug Perspective

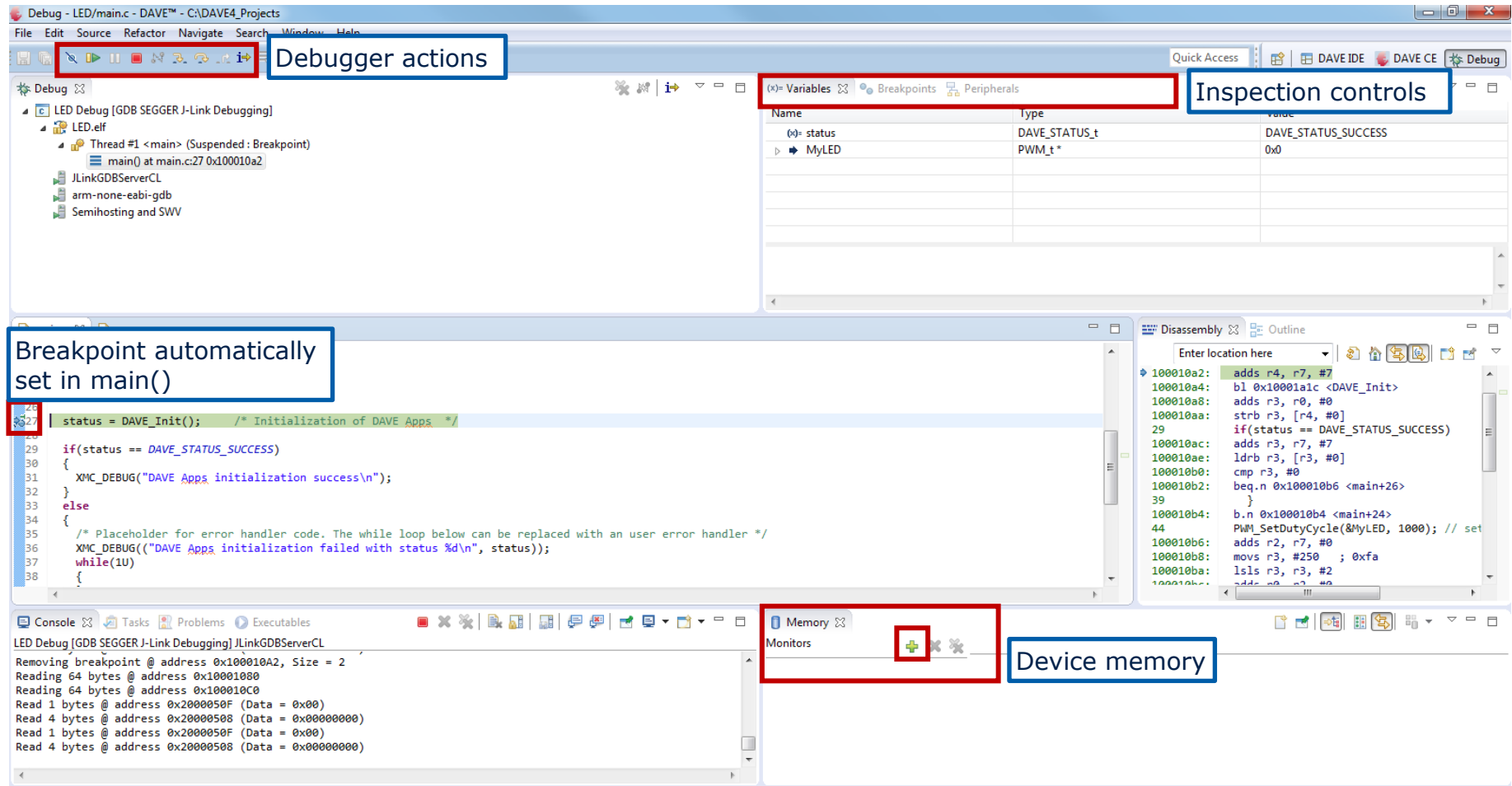


- Hint: To switch to Project Workspace Perspective, click DAVE CE at upper right corner of window



# The Debug Perspective [1/4]

## Debug Workspace



The screenshot displays the Infineon DAVE IDE's Debug Perspective, which is used for debugging applications. The interface is divided into several panes:

- Debugger actions:** A toolbar at the top left containing icons for running, stepping, and other debugging actions.
- Inspection controls:** A panel on the top right showing variables and their values. It includes tabs for Variables, Breakpoints, and Peripherals. The Variables tab is active, showing a table of variables.
- Breakpoint automatically set in main():** A callout box pointing to a breakpoint icon in the source code editor.
- Source code editor:** The main area showing the C source code for `main.c`. A breakpoint is set at line 29, which is highlighted in green.
- Disassembly:** A panel on the bottom right showing the assembly code corresponding to the source code.
- Console:** A panel at the bottom left showing the output of the program, including messages about breakpoint removal and data reading.
- Memory:** A panel at the bottom right showing the memory addresses and values of the program.
- Device memory:** A callout box pointing to the Memory panel.

Name	Type	Value
(*) status	DAVE_STATUS_t	DAVE_STATUS_SUCCESS
MyLED	PWM_t*	0x0

```
29 status = DAVE_Init(); /* Initialization of DAVE Apps */
30
31 if(status == DAVE_STATUS_SUCCESS)
32 {
33     XMC_DEBUG("DAVE Apps initialization success\n");
34 }
35 else
36 {
37     /* Placeholder for error handler code. The while loop below can be replaced with an user error handler */
38     XMC_DEBUG(("DAVE Apps initialization failed with status %d\n", status));
39     while(1)
40     {
41     }
```

```
100010a2: adds r4, r7, #7
100010a4: bl 0x10001a1c <DAVE_Init>
100010a8: adds r3, r0, #0
100010aa: strb r3, [r4, #0]
29      if(status == DAVE_STATUS_SUCCESS)
100010ac: adds r3, r7, #7
100010ae: ldrb r3, [r3, #0]
100010b0: cmp r3, #0
100010b2: beq.n 0x100010b6 <main+26>
39      }
100010b4: b.n 0x100010b4 <main+24>
44      PWM_SetDutyCycle(&MyLED, 1000); // set
100010b6: adds r2, r7, #0
100010b8: movs r3, #250 ; 0xfa
100010ba: lsls r3, r3, #2
100010bc: adds r0, r7, #0
```

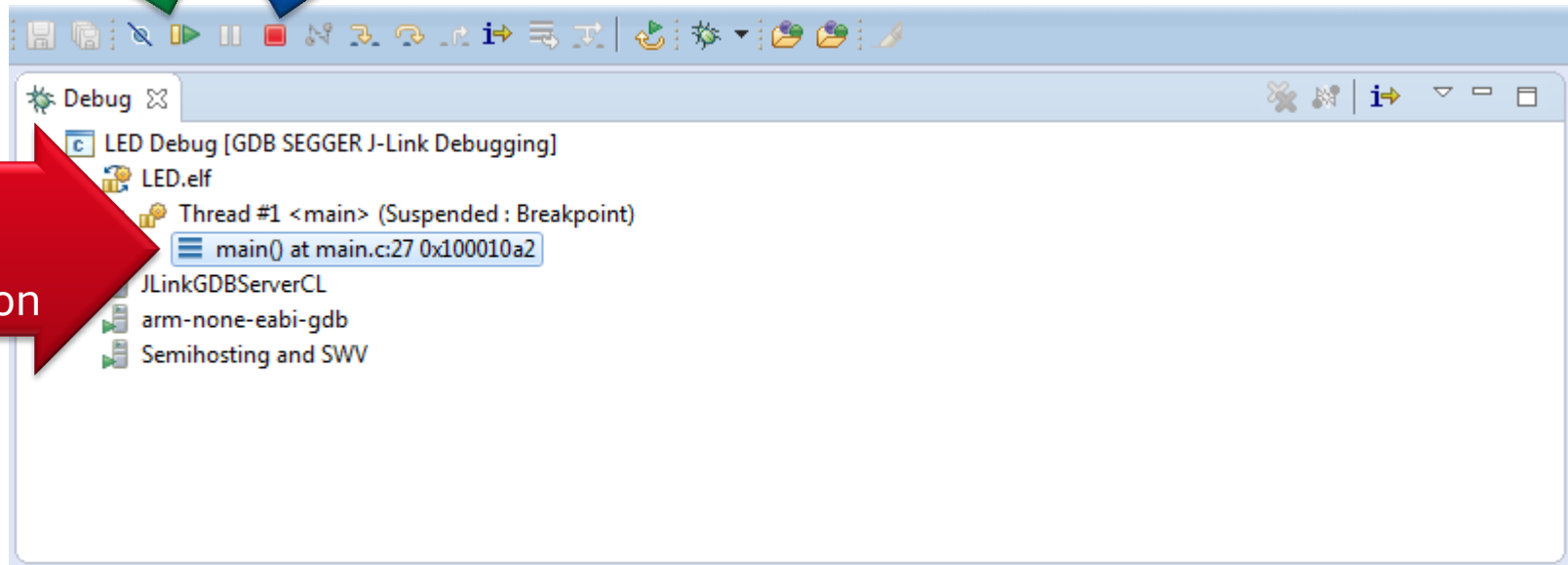
Removing breakpoint @ address 0x100010A2, Size = 2  
Reading 64 bytes @ address 0x10001080  
Reading 64 bytes @ address 0x100010C0  
Read 1 bytes @ address 0x2000050F (Data = 0x00)  
Read 4 bytes @ address 0x20000508 (Data = 0x00000000)  
Read 1 bytes @ address 0x2000050F (Data = 0x00)  
Read 4 bytes @ address 0x20000508 (Data = 0x00000000)

# The Debug Perspective [2/4]

## Debug Window



Debug  
Session  
information

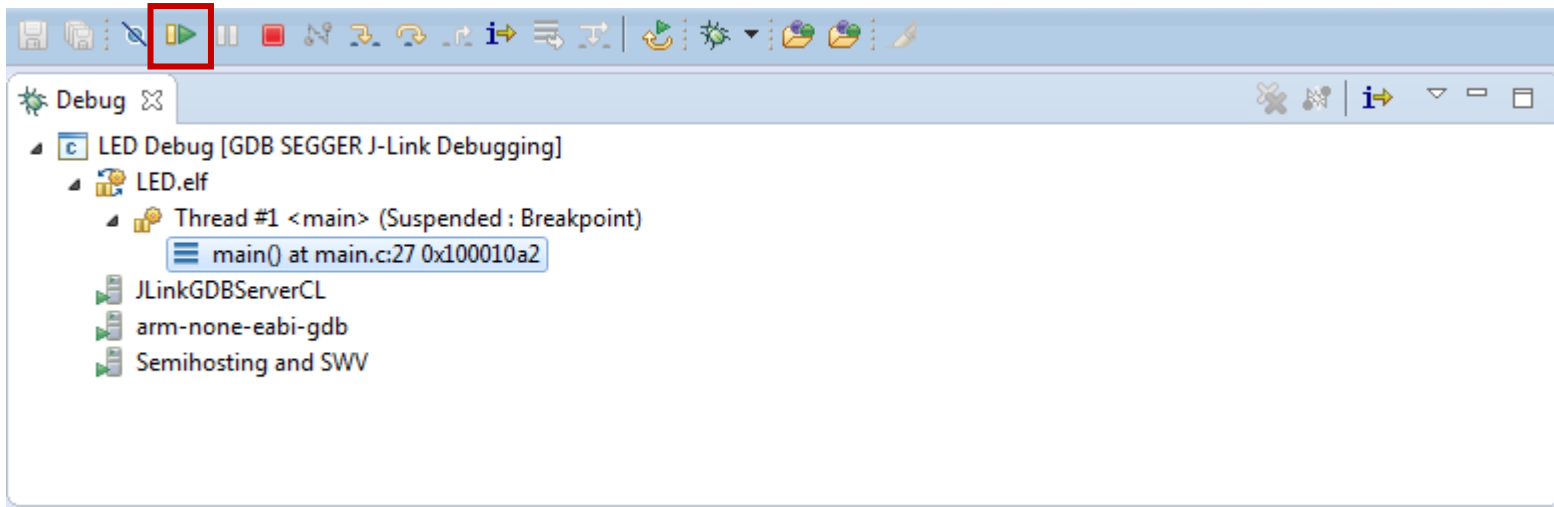




# The Debug Perspective [3/4]

## Start Program

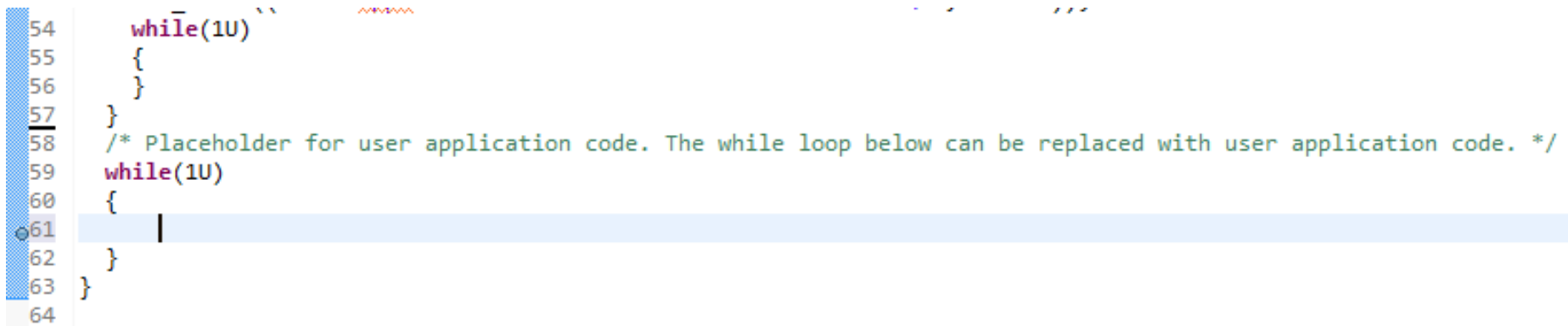
- Click on the Resume button to start code execution



# The Debug Perspective [4/4]

## Breakpoints

- To place a breakpoint, double-click on the blue bar at the line of code



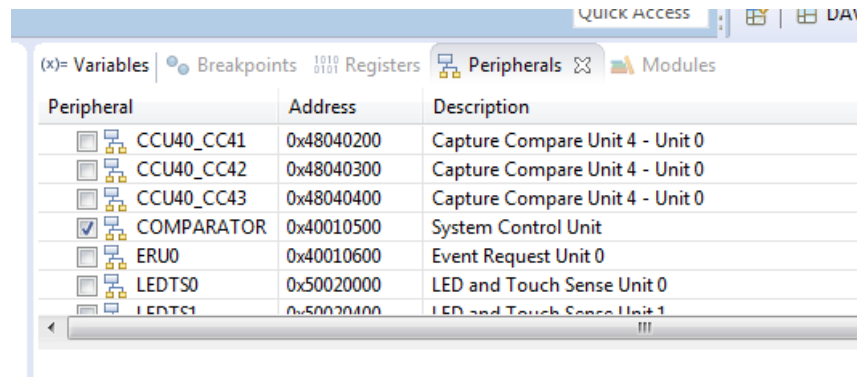
The screenshot shows a code editor with a light blue background. On the left, a vertical blue bar indicates line numbers from 54 to 64. Line 54 is highlighted in blue. The code is as follows:

```
54     while(1U) ``  
55     {  
56     }  
57 }  
58 /* Placeholder for user application code. The while loop below can be replaced with user application code. */  
59 while(1U)  
60 {  
61 |  
62 }  
63 }  
64
```

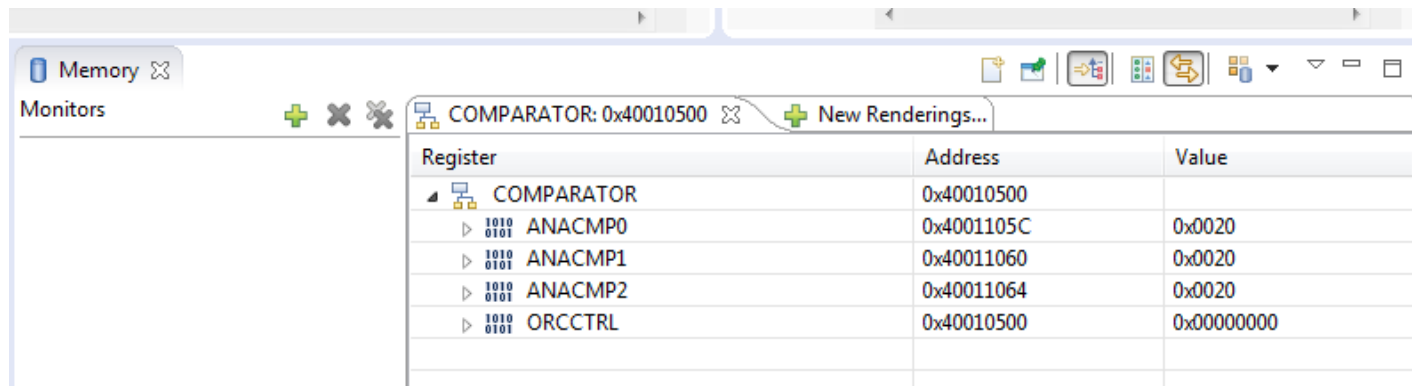
A vertical blue bar is positioned at the start of line 61, and a small black circle (the breakpoint) is placed on it. A horizontal blue bar highlights the entire line 61.

# Debug Session

- Select COMPARATOR in the Peripherals menu of Inspection controls part of Debug Perspective

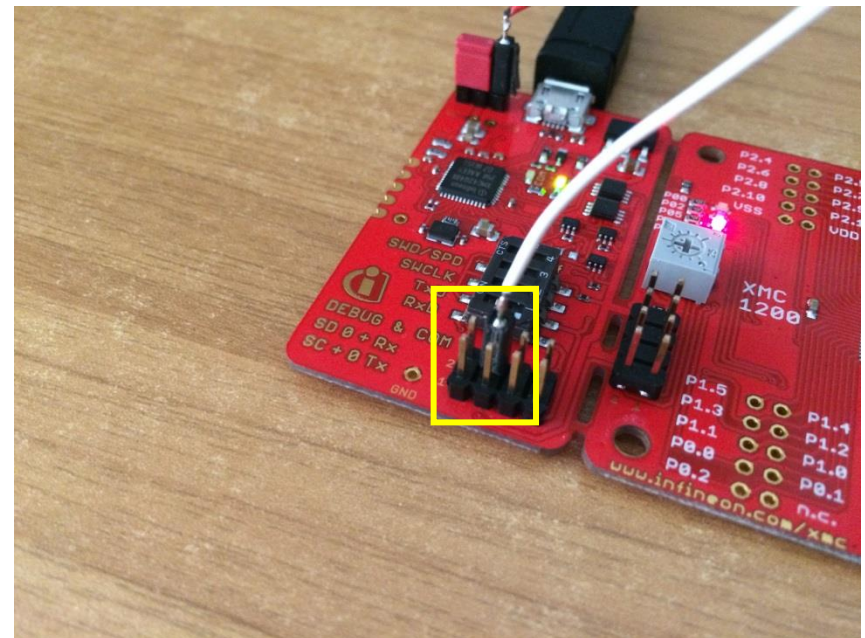
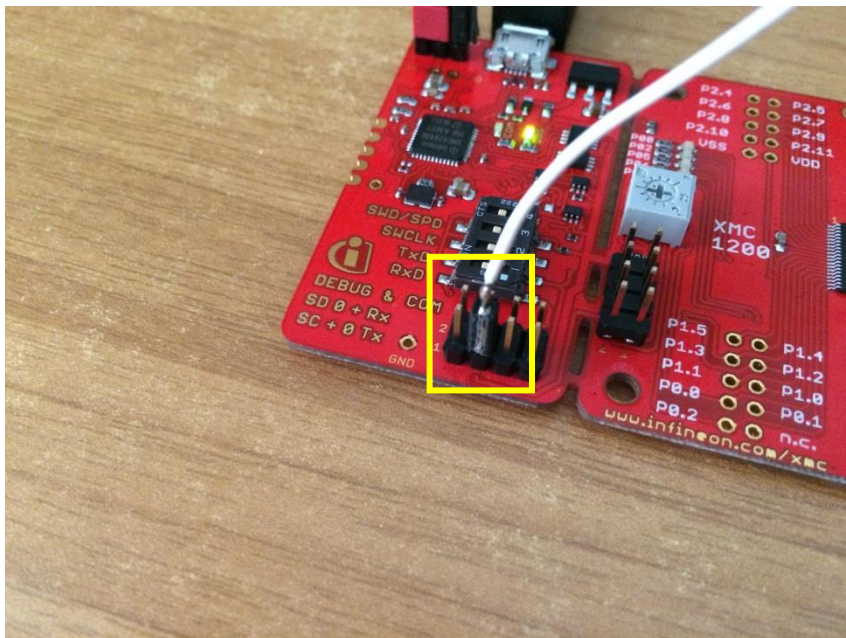


- In the Memory window will appear the registers of chosen peripheral



# Debug session

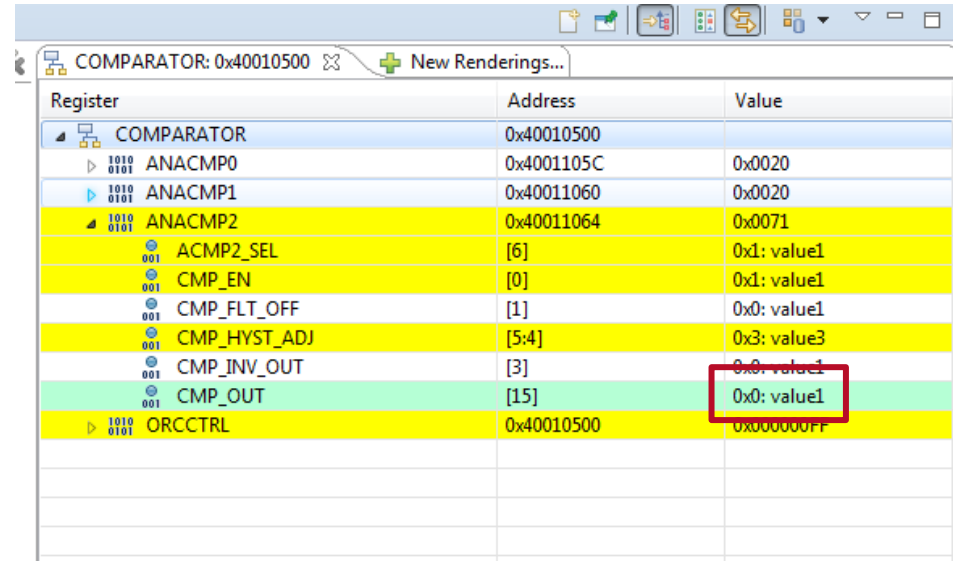
- Switching input P2.1 from "+" to "0" it is possible to see the change of CMP\_OUT register



# Debug Session

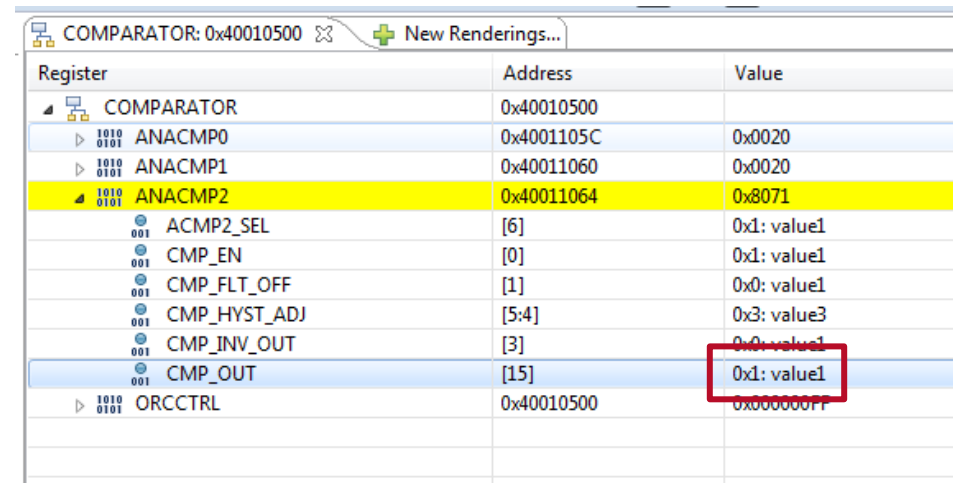
## ■ Outputs of CMP\_OUT

□ P2.1 connected to "+"



Register	Address	Value
COMPARATOR	0x40010500	
ANACMP0	0x4001105C	0x0020
ANACMP1	0x40011060	0x0020
ANACMP2	0x40011064	0x0071
ACMP2_SEL	[6]	0x1: value1
CMP_EN	[0]	0x1: value1
CMP_FLT_OFF	[1]	0x0: value1
CMP_HYST_ADJ	[5:4]	0x3: value3
CMP_INV_OUT	[3]	0x0: value1
CMP_OUT	[15]	0x0: value1
ORCTRL	0x40010500	0x000000FF

□ P2.1 connected to "0"



Register	Address	Value
COMPARATOR	0x40010500	
ANACMP0	0x4001105C	0x0020
ANACMP1	0x40011060	0x0020
ANACMP2	0x40011064	0x8071
ACMP2_SEL	[6]	0x1: value1
CMP_EN	[0]	0x1: value1
CMP_FLT_OFF	[1]	0x0: value1
CMP_HYST_ADJ	[5:4]	0x3: value3
CMP_INV_OUT	[3]	0x0: value1
CMP_OUT	[15]	0x1: value1
ORCTRL	0x40010500	0x000000FF

# Debug Session

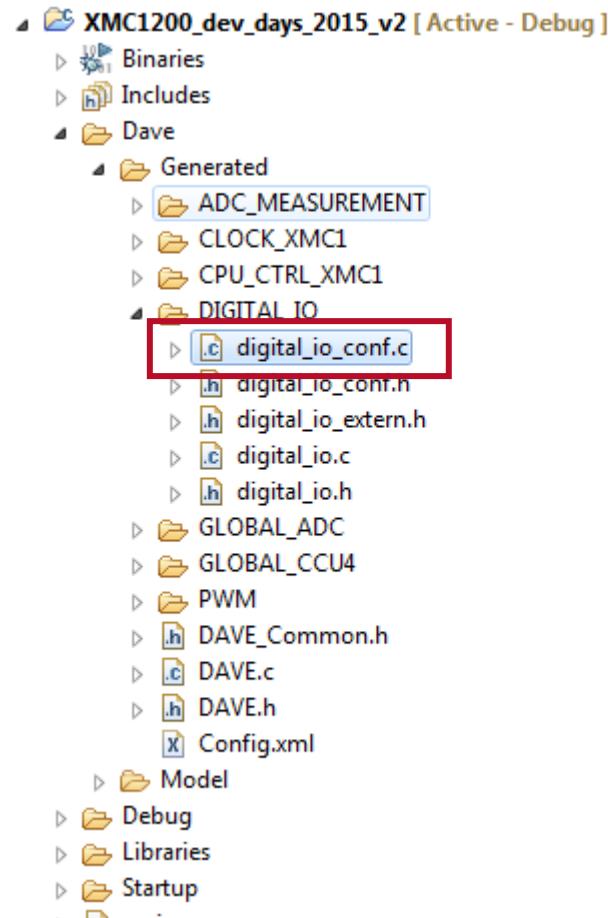
- The P0.5 doesn't change its status. Why?
  - It is not set as ACMP2. OUT!!
- Each pin has many functions, called Alternate Functions
- In every Reference Manual, it is present a table which describe this characteristics

# Alternate Functions

Function	Outputs							Inputs							
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	Input	Input	Input	Input	Input	Input	Input	Input
P0.0	ERU0. PDOUT0	LEDS0. LINE7	ERU0. GOUT0	CCU40. OUT0		USIC0_CH 0.SELO0	USIC0_CH 1.SELO0	BCCU0. TRAPINB	CCU40.IN0 C			USIC0_CH 0.DX2A	USIC0_CH 1.DX2A		
P0.1	ERU0. PDOUT1	LEDS0. LINE8	ERU0. GOUT1	CCU40. OUT1		BCCU0. OUT8	SCU. VDROP		CCU40.IN1 C						
P0.2	ERU0. PDOUT2	LEDS0. LINE5	ERU0. GOUT2	CCU40. OUT2		VADC0. EMUX02			CCU40.IN2 C						
P0.3	ERU0. PDOUT3	LEDS0. LINE4	ERU0. GOUT3	CCU40. OUT3		VADC0. EMUX01			CCU40.IN3 C						
P0.4	BCCU0. OUT0	LEDS0. LINE3	LEDS0. COL3	CCU40. OUT1		VADC0. EMUX00	WWDT. SERVICE_ OUT								
P0.5	BCCU0. OUT1	LEDS0. LINE2	LEDS0. COL2	CCU40. OUT0		ACMP2. OUT									
P0.6	BCCU0. OUT2	LEDS0. LINE1	LEDS0. COL1	CCU40. OUT0		USIC0_CH 1.MCLKOU T	USIC0_CH 1.DOUT0		CCU40.IN0 B			USIC0_CH 1.DX0C			
P0.7	BCCU0. OUT3	LEDS0. LINE0	LEDS0. COL0	CCU40. OUT1		USIC0_CH 0.SCLKOU T	USIC0_CH 1.DOUT0		CCU40.IN1 B			USIC0_CH 0.DX1C	USIC0_CH 1.DX0D	USIC0_CH 1.DX1C	
P0.8	BCCU0. OUT4	LEDS1. LINE0	LEDS0. COLA	CCU40. OUT2		USIC0_CH 0.SCLKOU T	USIC0_CH 1.SCLKOU T		CCU40.IN2 B			USIC0_CH 0.DX1B	USIC0_CH 1.DX1B		
P0.9	BCCU0. OUT5	LEDS1. LINE1	LEDS0. COL6	CCU40. OUT3		USIC0_CH 0.SELO0	USIC0_CH 1.SELO0		CCU40.IN3 B			USIC0_CH 0.DX2B	USIC0_CH 1.DX2B		
P0.10	BCCU0. OUT6	LEDS1. LINE2	LEDS0. COL5	ACMP0. OUT		USIC0_CH 0.SELO1	USIC0_CH 1.SELO1					USIC0_CH 0.DX2C	USIC0_CH 1.DX2C		
P0.11	BCCU0. OUT7	LEDS1. LINE3	LEDS0. COL4	USIC0_CH 0.MCLKOU T		USIC0_CH 0.SELO2	USIC0_CH 1.SELO2					USIC0_CH 0.DX2D	USIC0_CH 1.DX2D		
P0.12	BCCU0. OUT8	LEDS1. LINE4	LEDS0. COL3	LEDS1. COL3		USIC0_CH 0.SELO3		BCCU0. TRAPINA	CCU40.IN0 A	CCU40.IN1 A	CCU40.IN2 A	CCU40.IN3 A	USIC0_CH 0.DX2E		
P0.13	WWDT. SERVICE_ OUT	LEDS1. LINE5	LEDS0. COL2	LEDS1. COL2		USIC0_CH 0.SELO4						USIC0_CH 0.DX2F			
P0.14	BCCU0. OUT7	LEDS1. LINE6	LEDS0. COL1	LEDS1. COL1		USIC0_CH 0.DOUT0	USIC0_CH 0.SCLKOU T					USIC0_CH 0.DX0A	USIC0_CH 0.DX1A		

# Alternate Function: how to configure

- Step 1: change perspective to DAVE CE and open the digital\_io\_conf.c file with double click





# Alternate Function: how to configure

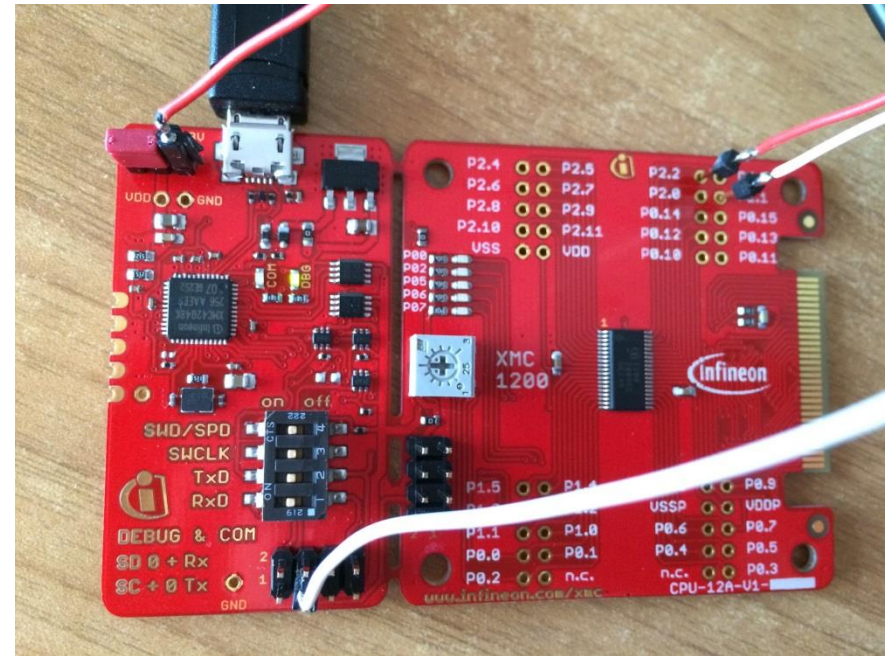
- Step 2: Change the conf mode of the DIGITAL\_IO from ***XMC\_GPIO\_MODE\_OUTPUT\_PUSH\_PULL*** to ***XMC\_GPIO\_MODE\_OUTPUT\_PUSH\_PULL\_ALT6***

```

78
79 XMC_GPIO_CONFIG_t DIGITAL_IO_0_config =
80 {
81     .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT6,
82     .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW,
83
84 };
85
86 DIGITAL_IO_t DIGITAL_IO_0 =
87 {
88     .gpio_port = XMC_GPIO_PORT0,
89     .gpio_pin = 5U,
90     .gpio_config = &DIGITAL_IO_0_config,
91 };
92
93

```

- 



# Agenda

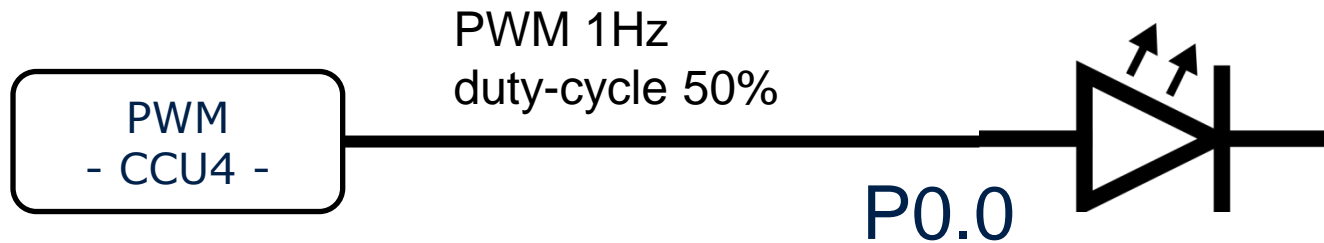
- LLD of XMCLib: use of ACMP

- **Blink an LED with PWM APP**

- Use ADC APP to update PWM dutycycle

# PWM to Blink an LED

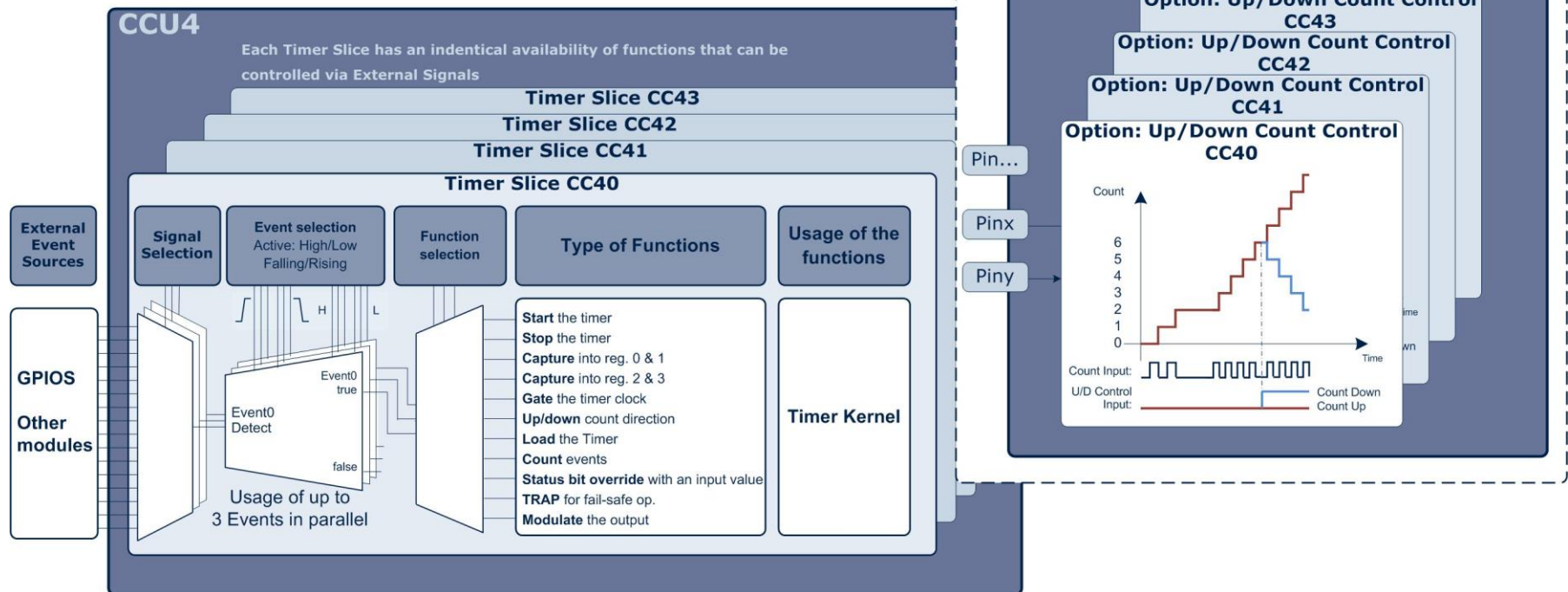
- Creation of a PWM signal that control a LED



# CCU4

## Modular Timer Approach

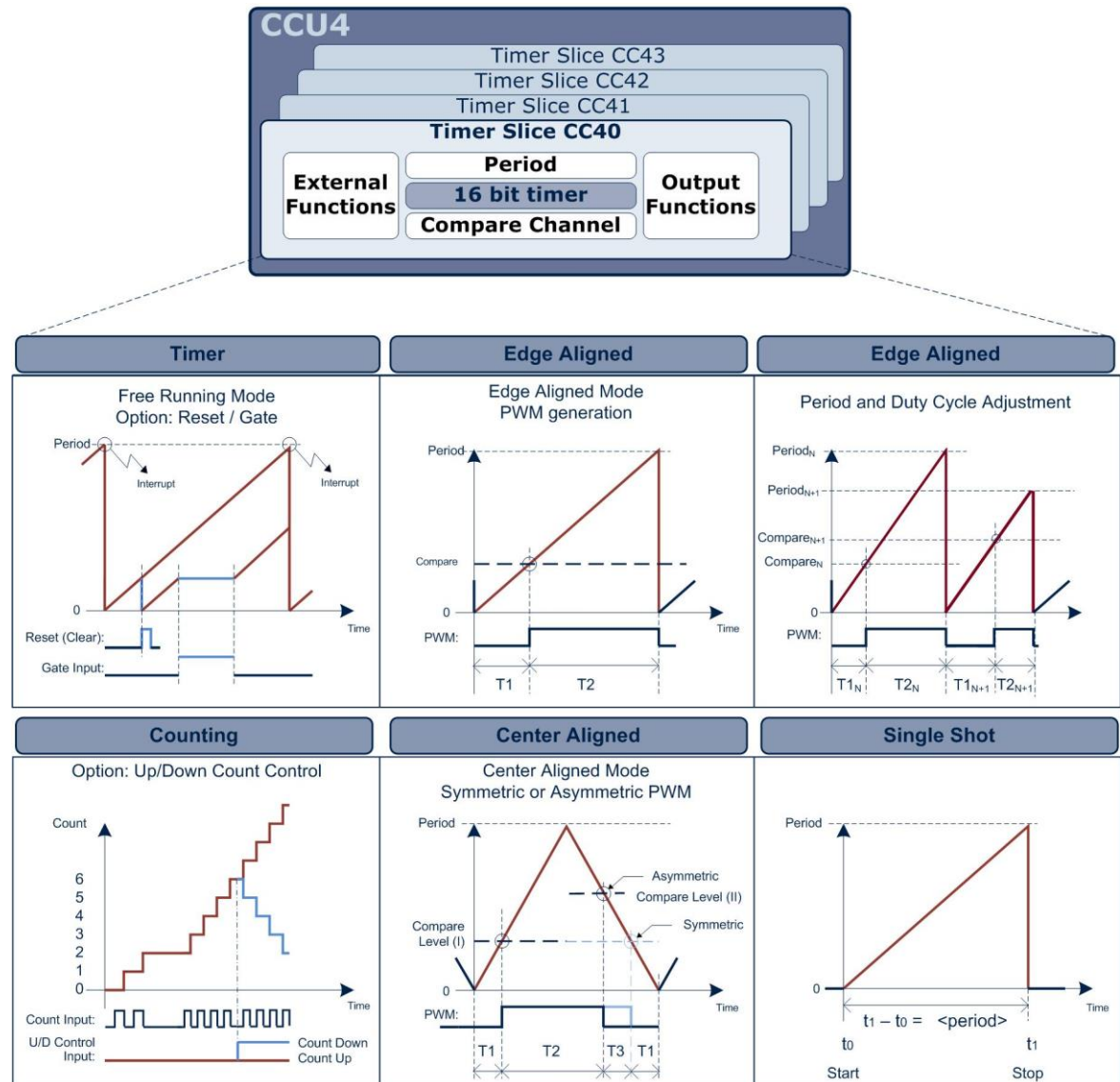
- Equal structure and same availability of features for each of the 4 Timer Slices
- Functions controllable via external signals do not depend on the selected signal
- Portability of code is not dependent on the used Timer Slice
- High amount of configurable external functions (11), make each Timer Slice a very flexible HW resource for signal conditioning



# CCU4

## Flexible PWM generation

- Each Timer Slice of the CCU4 can operate in center aligned or edge aligned mode
- Additional operation modes like single shot, counting or dithering modes are also available
- Update of the Duty Cycle and Period can be done on-the-fly to accommodate different operation requirements
- Additional external controllable functions give another degree of PWM manipulation (e.g. timer gate, timer load, timer clear, etc)

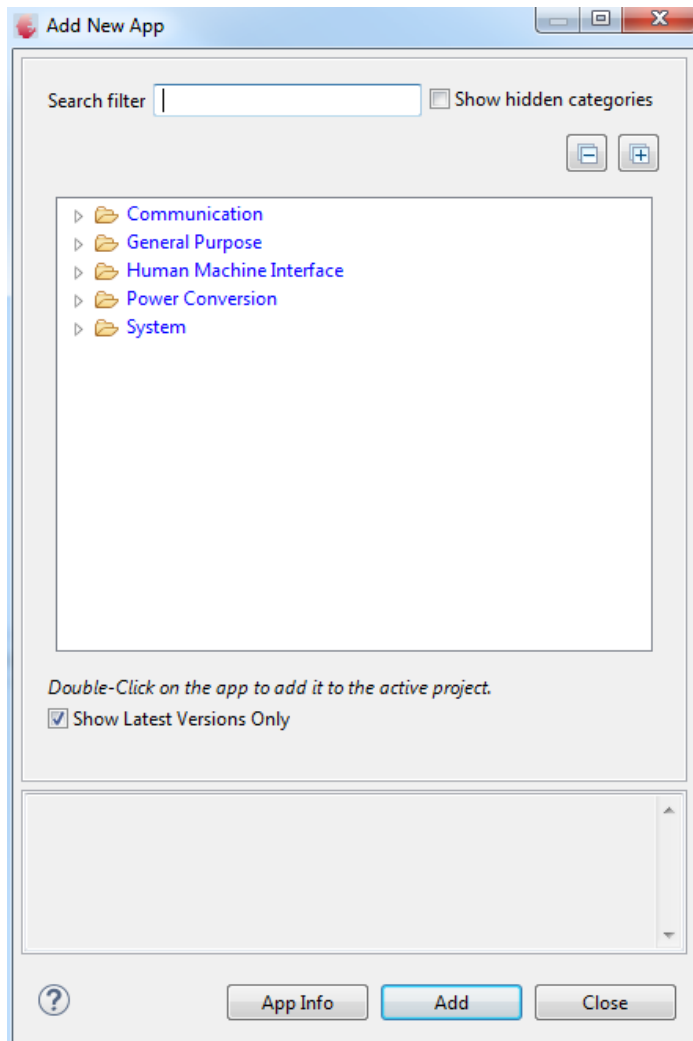


# Learning Outcome

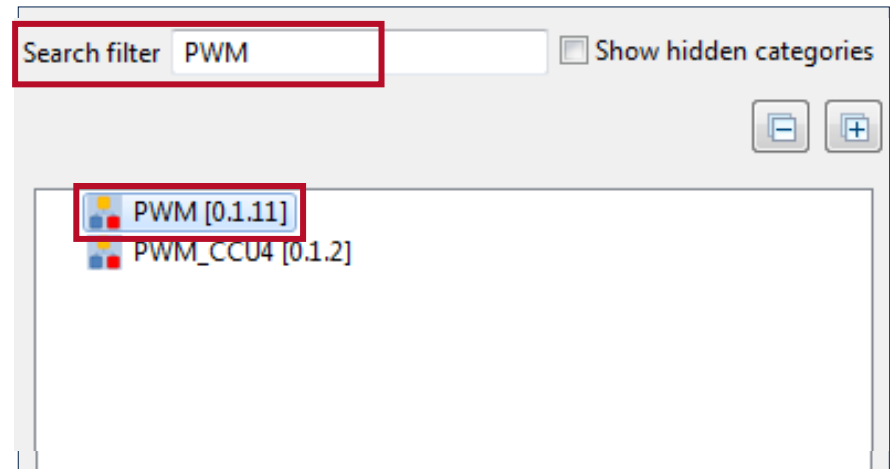
- Learn the basic principles of DAVE™:
  - Required XMC kit
  - Create DAVE™ Project
  - GUI based DAVE™ APPs configuration
  - Graphical pin mapping
  - One touch code generation
  - Download and debug code
  - DAVE™ updates
  - Expert support



# Add DAVE™ APP from Code Repository



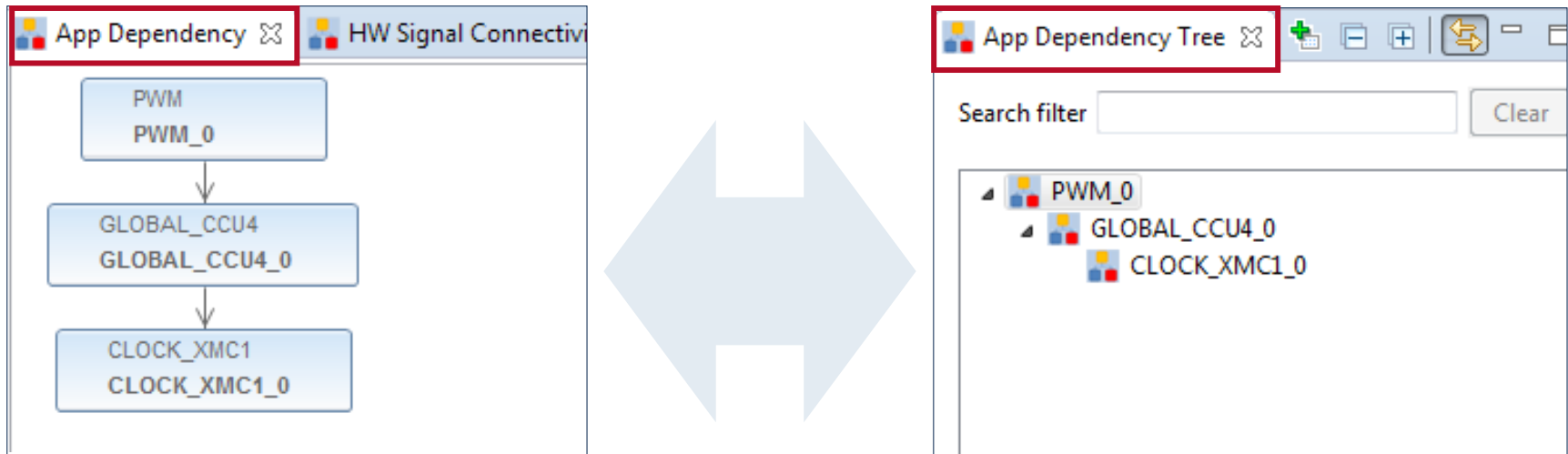
- Add DAVE™ APP to project
  1. Click  in Tool Panel, or
  2. DAVE → APP New APP
  3. Type "PWM" in the search filter field, and double-click PWM APP



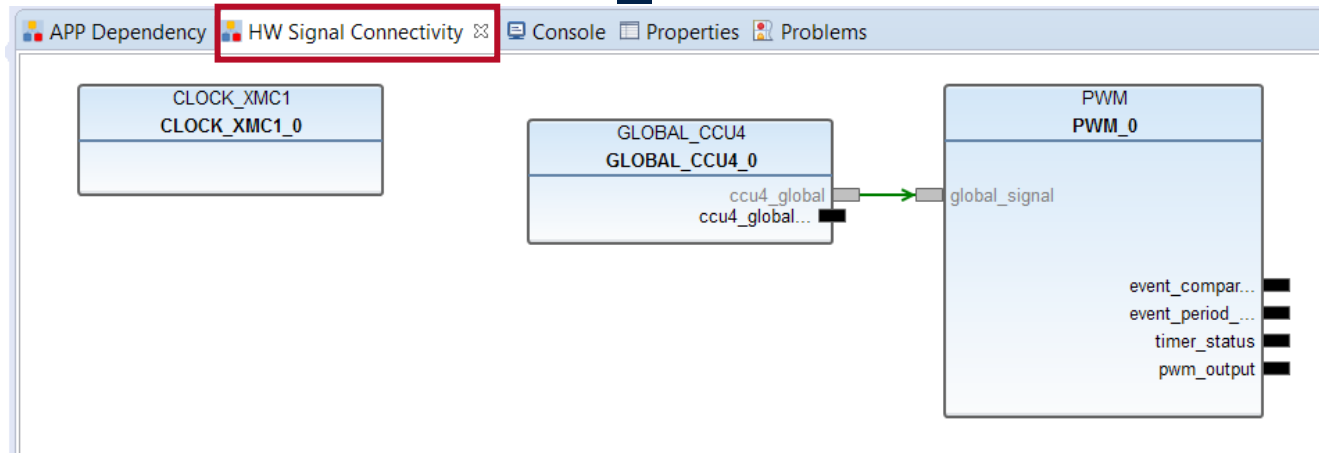


# More Project Views

- All APPs included in the Project are displayed in different views:



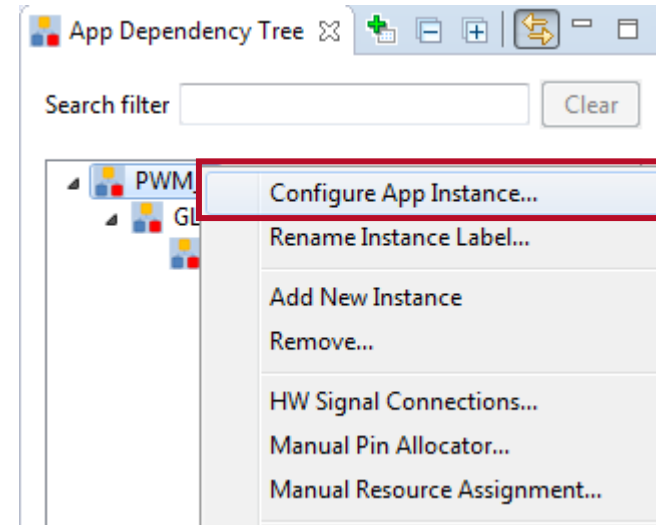
- The number behind “\_” identifies the instance of an APP



# DAVE™ APPs Editor View

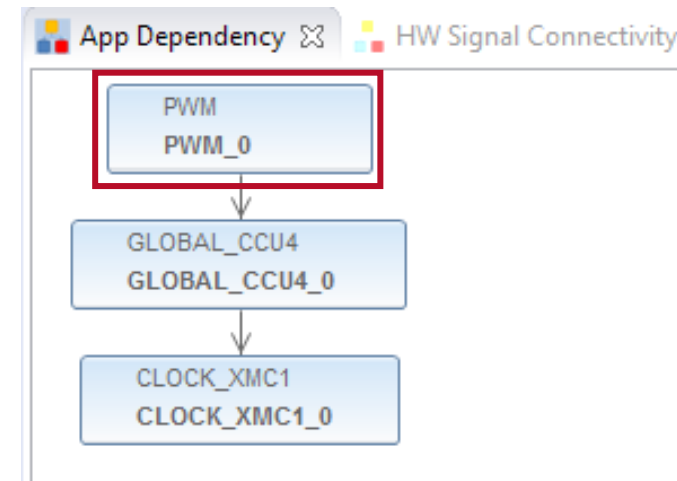
## ■ Either

1. Right-click APP name in the App Dependency Tree view
2. Select Configure App Instance



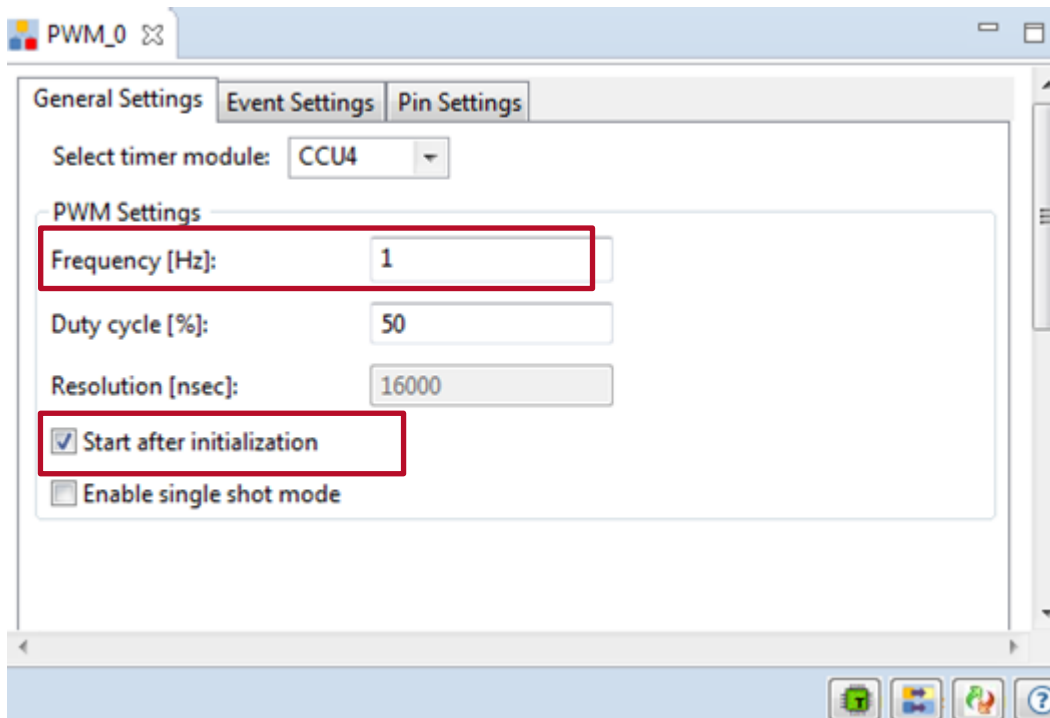
## ■ Or

- Double-click APP name in the App Dependency view



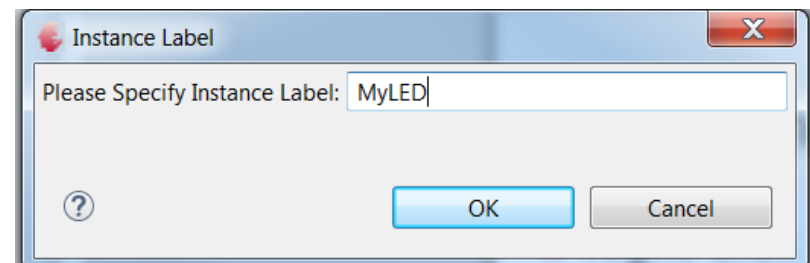
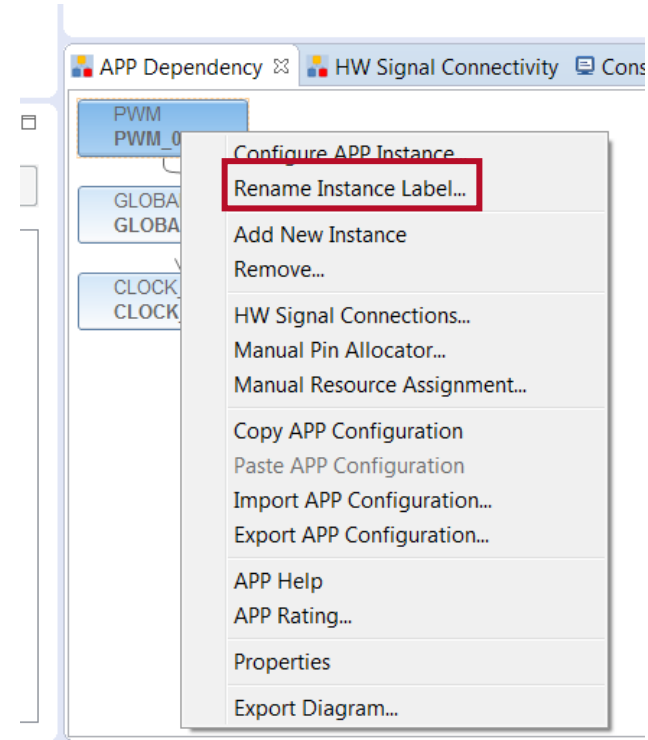
# PWM APP Configuration

- Configure PWM APP via graphical user interface editor
  1. Set PWM Frequency to 1 Hz
  2. LED blinks every 0.5 second
  3. Check "Start Timer After Initialization"



# Rename the Instance Label of the PWM APP

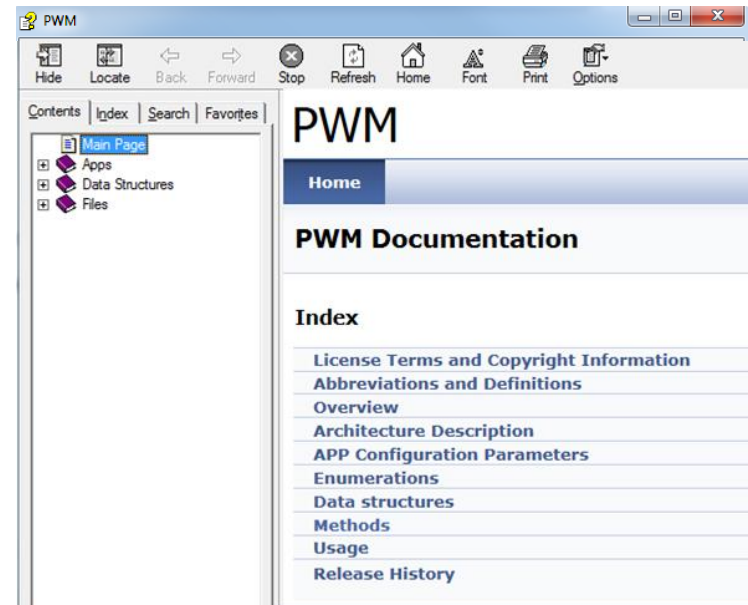
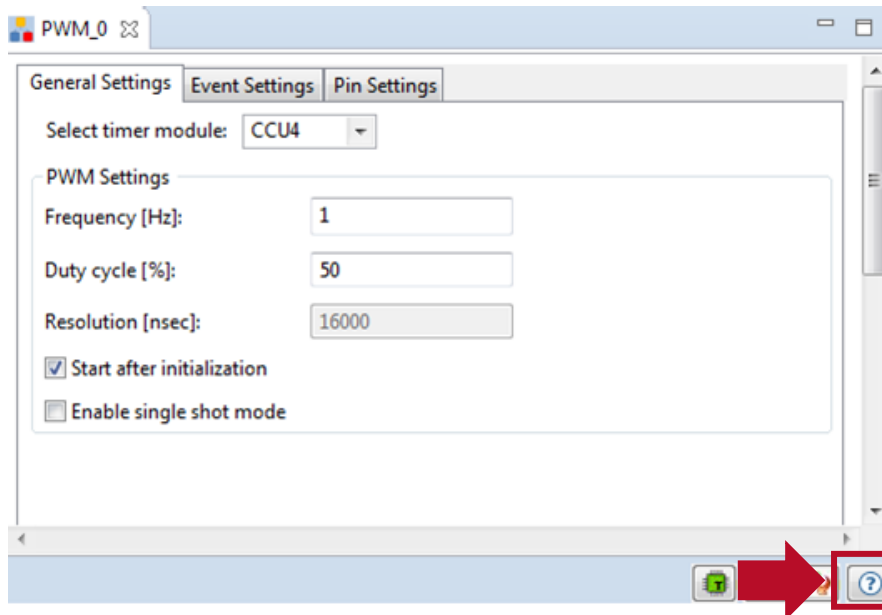
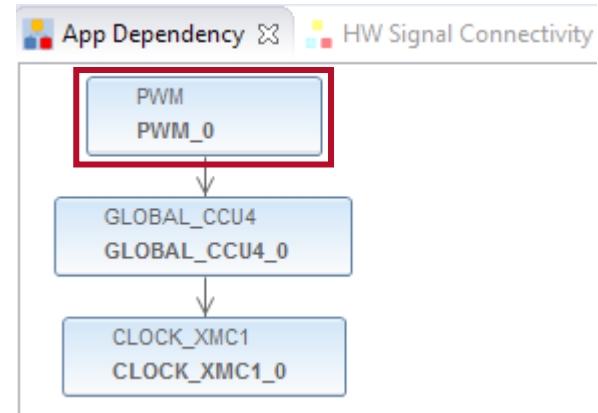
- Right click on the PWM APP
- Select  
Rename Instance Label...
- Type in: MyLED
- Now "MyLED" can be used as handler in the APIs of the PWM APP to reference this instance



# Hint: Additional Information about APPs


## ■ Reference to DAVE™ APP information

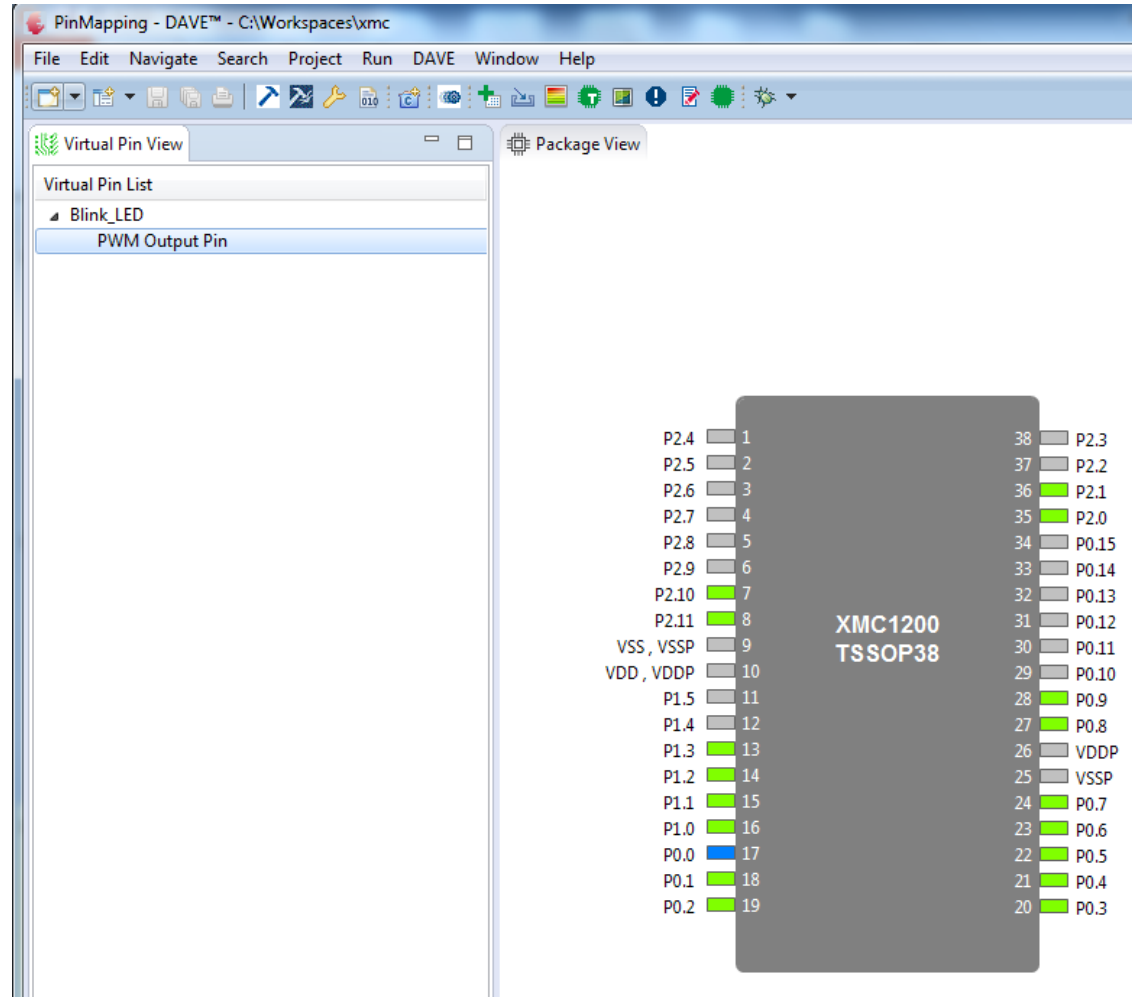
1. Double-click DAVE™ APP (e.g. PWM\_0) in App Dependency View
2. Click Help icon



# Pin Mapping for PWM App (1/2)

■ Assign signal to pin with graphical pin mapping view

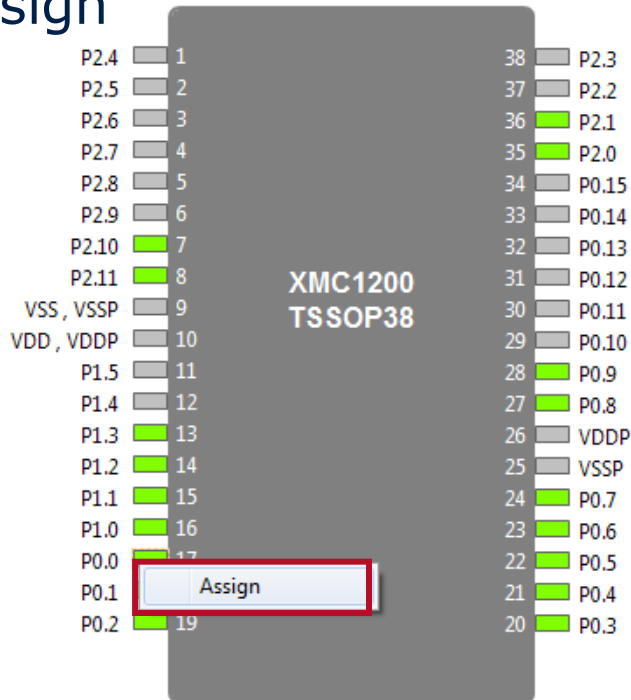
1. Click  to open Pin Mapping Perspective
2. Under Virtual Pin List, select PWM Output Pin
  - Green pin: All possible pins for selected signal
  - Blue pin: User assigned pin



# Pin Mapping for PWM App (2/2)

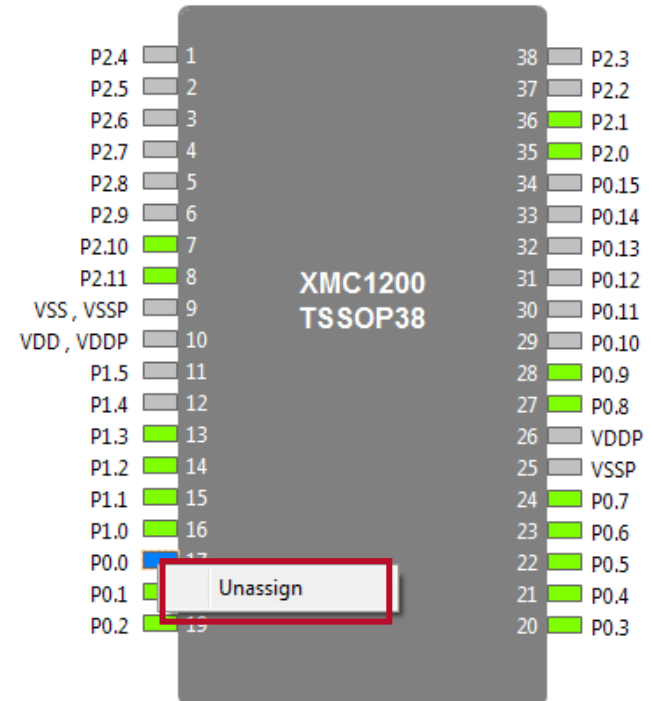
## ■ To assign pin:

- Right-click on a **green** pin → Assign



## ■ To unassign pin:

- Right-click on a **blue** pin → Unassign




## ■ Assign PWM Output Pin to User LED1 at P0.0, #17

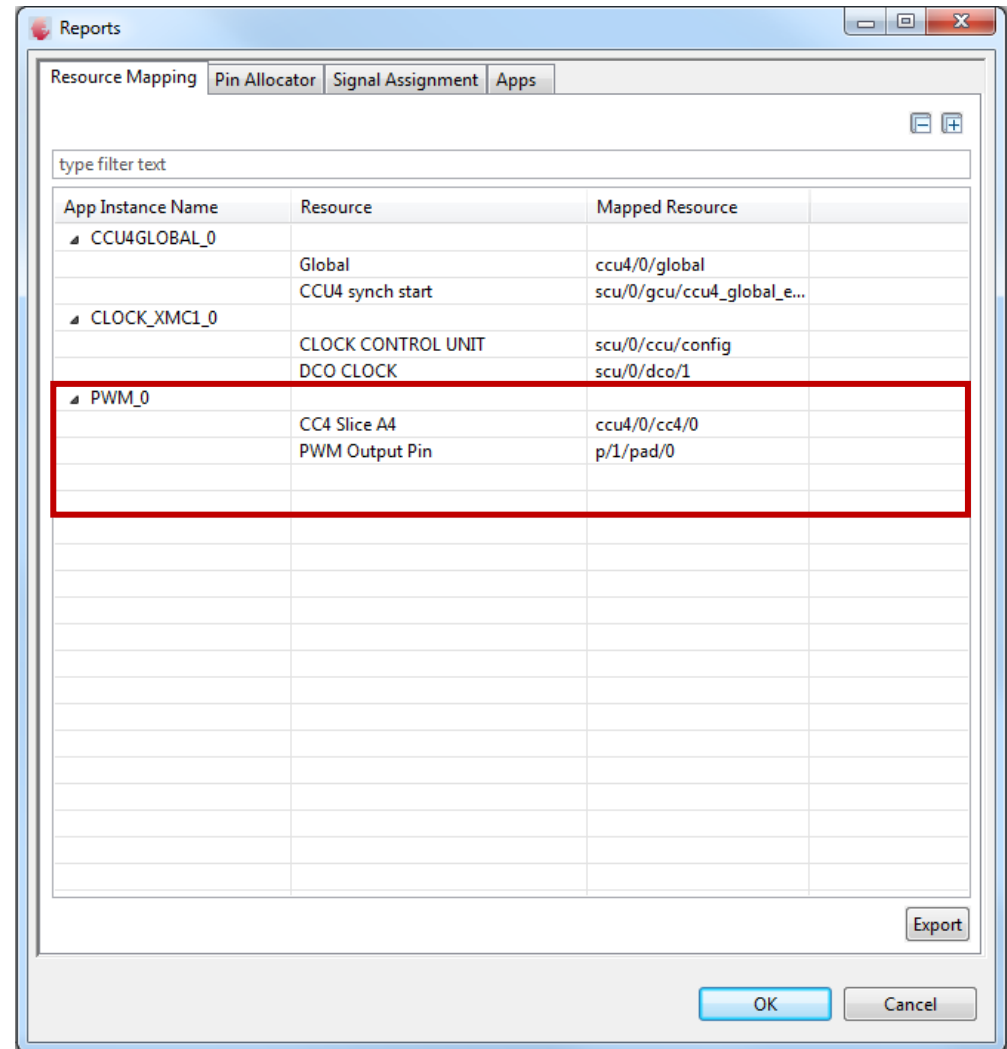
- Right-click on pin 17 → Assign
- In case you use a different board / device please select a pin that is connected to a LED

# Hint: Check Correct Resource Mapping

## ■ Check resource mapping

□ Click  to open

Reports in DAVE CE perspective




App Instance Name	Resource	Mapped Resource
▲ CCU4GLOBAL_0	Global	ccu4/0/global
	CCU4 synch start	scu/0/gcu/ccu4_global_e...
▲ CLOCK_XMC1_0	CLOCK CONTROL UNIT	scu/0/ccu/config
	DCO CLOCK	scu/0/dco/1
▲ PWM_0	CC4 Slice A4	ccu4/0/cc4/0
	PWM Output Pin	p/1/pad/0



# Generate Code and add a few Lines of Code to change the Duty Cycle of the PWM and Compile Code



## ■ One touch code generation

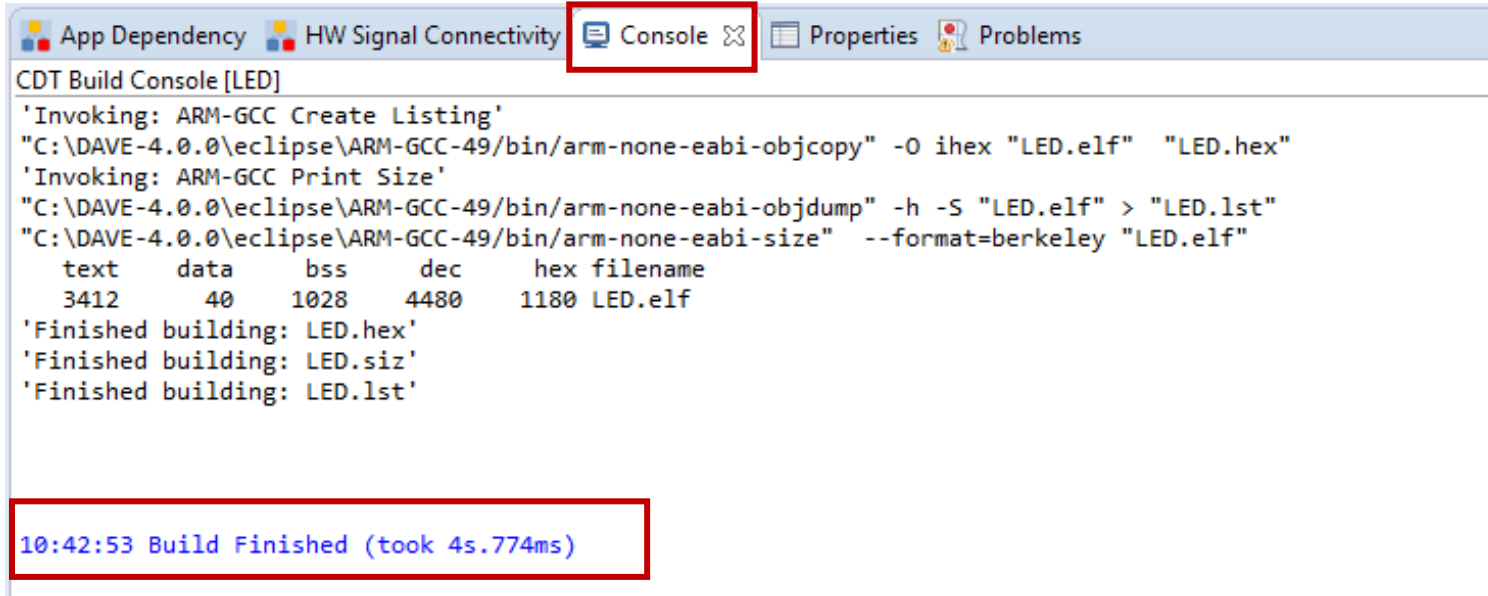
1. Click  in the tool panel
2. Generated code can be found under C/C++ Projects window, DAVE → Generated

## ■ Start Compiler tools to build the project

- Click  in the tool panel

# Check Compiler Results

- Ensure that Compiler finished building in Console window

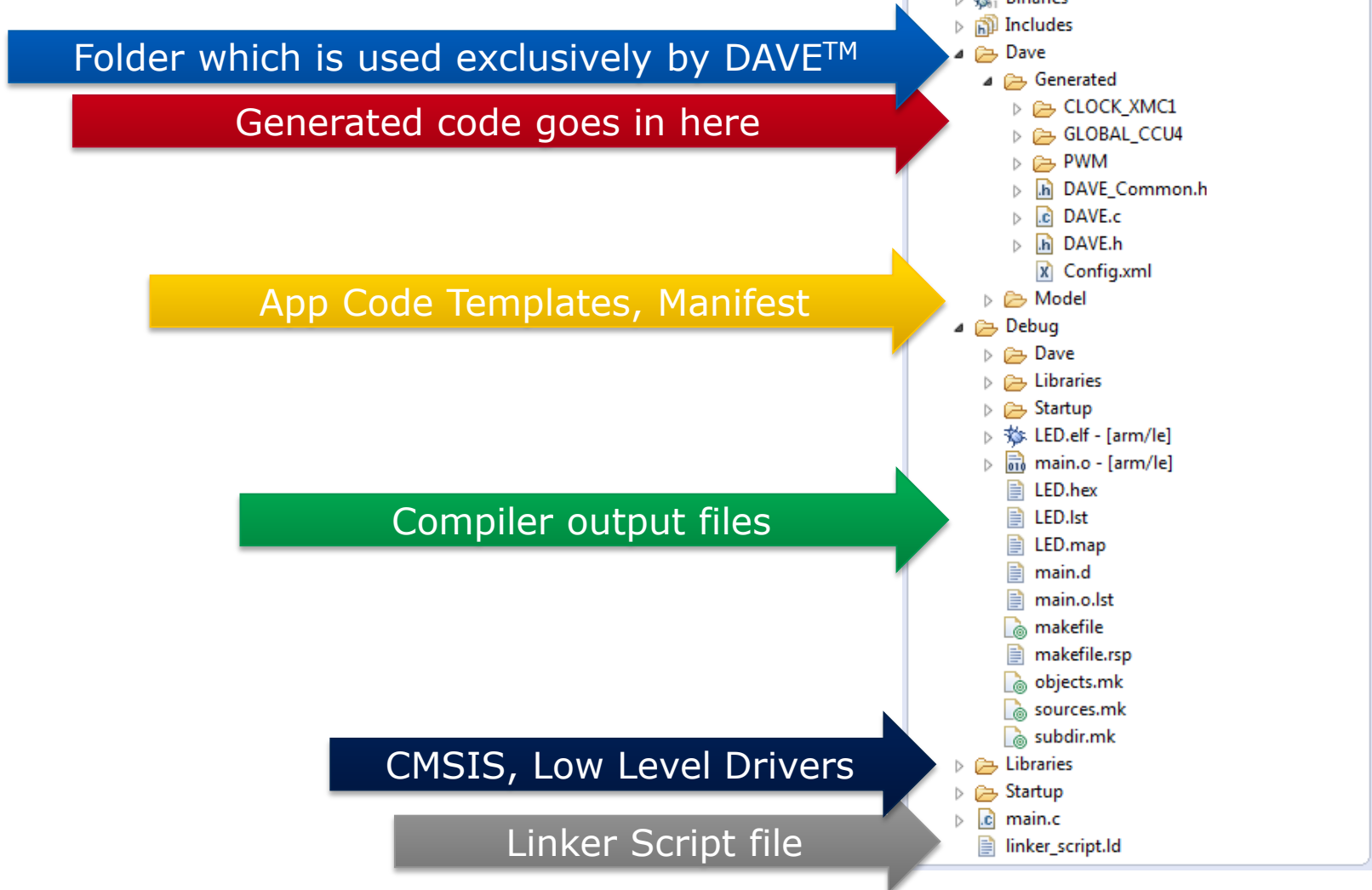


CDT Build Console [LED]

```
'Invoking: ARM-GCC Create Listing'
"C:\DAVE-4.0.0\eclipse\ARM-GCC-49/bin/arm-none-eabi-objcopy" -O ihex "LED.elf" "LED.hex"
'Invoking: ARM-GCC Print Size'
"C:\DAVE-4.0.0\eclipse\ARM-GCC-49/bin/arm-none-eabi-objdump" -h -S "LED.elf" > "LED.lst"
"C:\DAVE-4.0.0\eclipse\ARM-GCC-49/bin/arm-none-eabi-size" --format=berkeley "LED.elf"
  text    data    bss     dec     hex filename
  3412     40   1028   4480   1180 LED.elf
'Finished building: LED.hex'
'Finished building: LED.siz'
'Finished building: LED.lst'
```

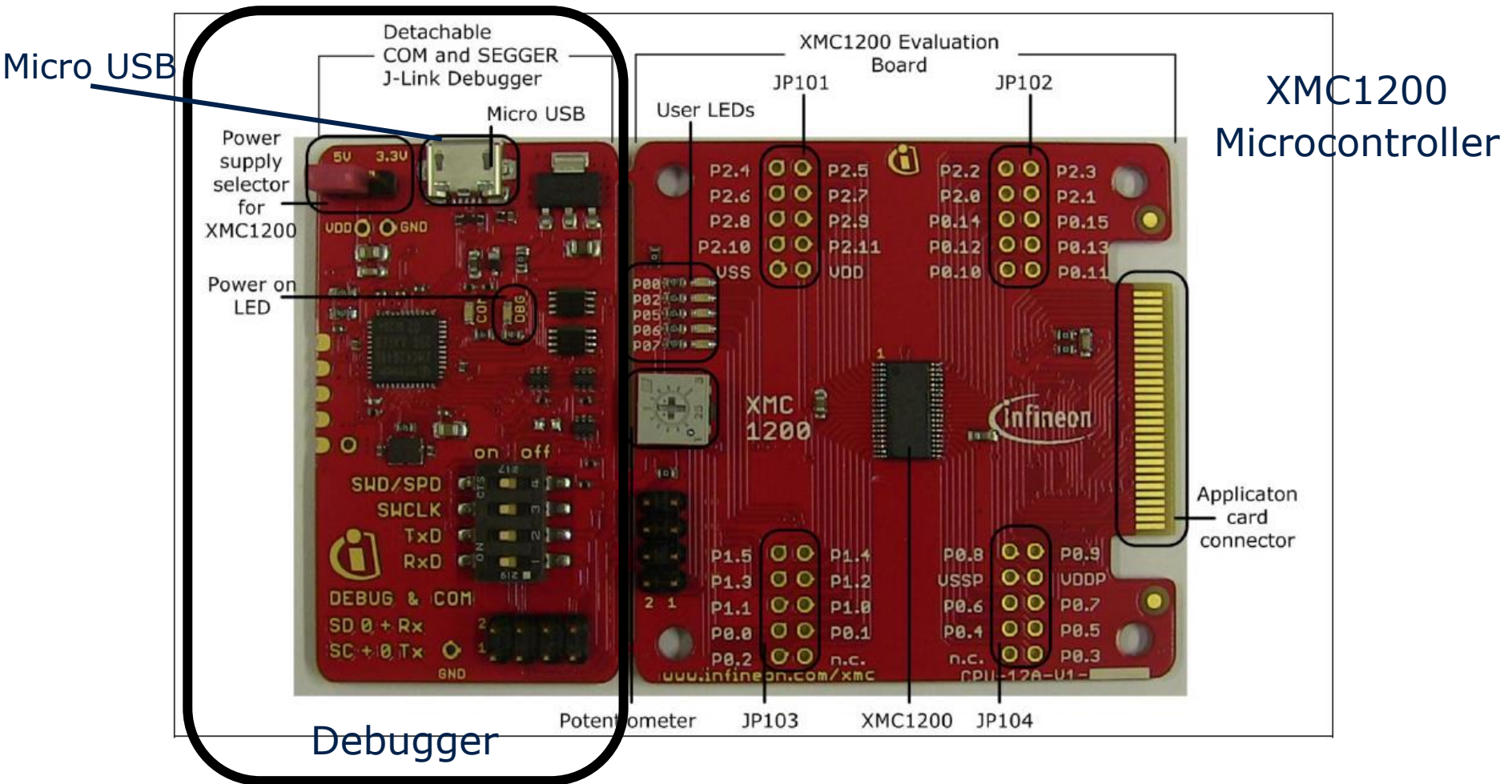
10:42:53 Build Finished (took 4s.774ms)

# The Project Folder



# Flash and Debug (1/3)

- Ensure the Debugger of the XMC 1200 BootKit is connected to your PC via USB



# Flash and Debug (2/3)

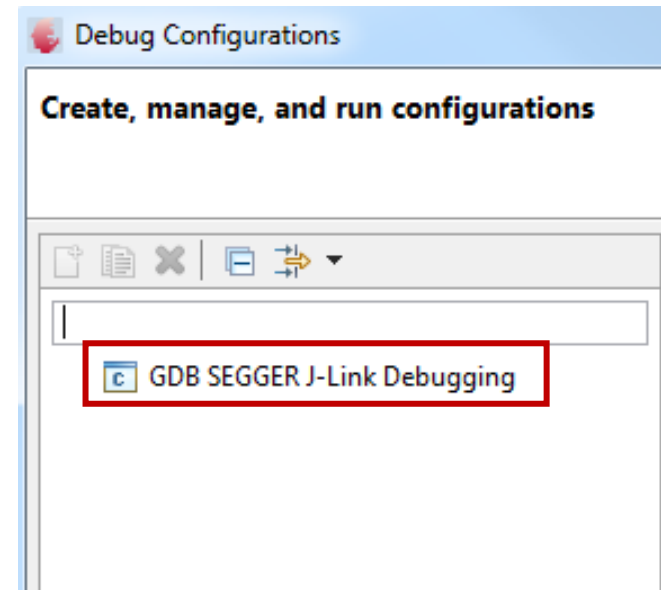
## ■ Start Debug Session

- Click  in the tool panel

## ■ Create a new Debug Configuration

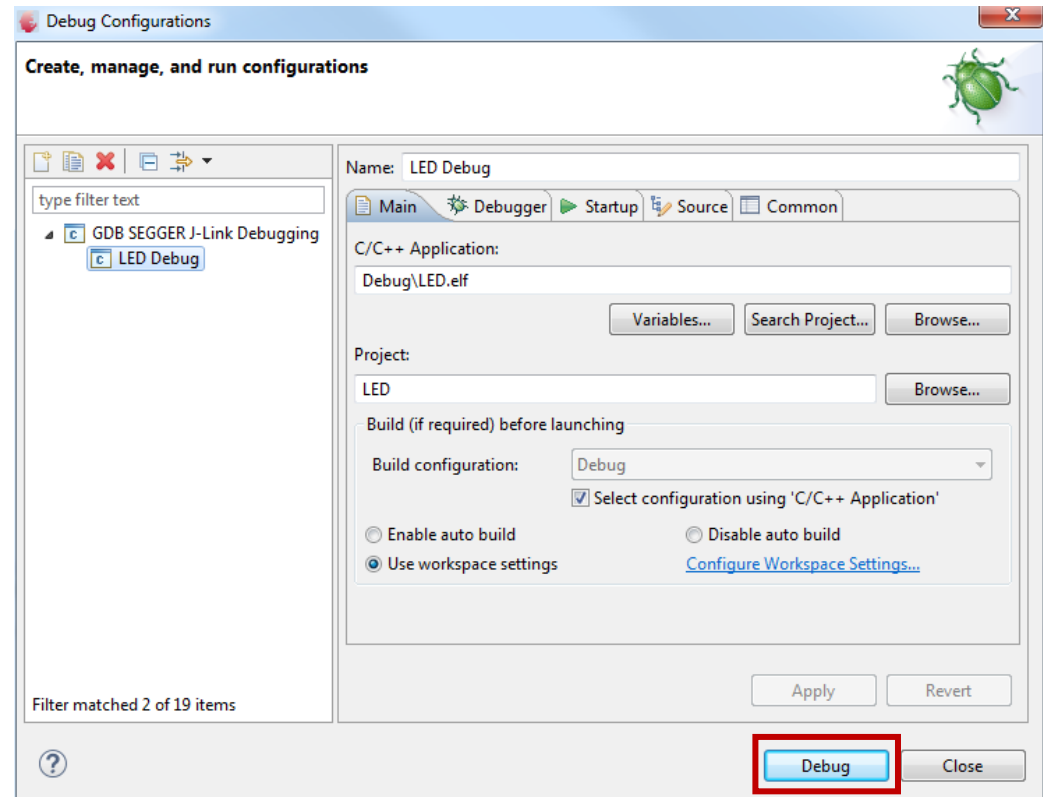
- Double-click  
"GDB SEGGER J-Link Debugging"

Segger J-link Driver software  
4.96h or above needs to be  
installed



# Flash and Debug (3/3)

- Click "Debug"
- The flashing process is started and DAVE automatically switches to Debug Perspective

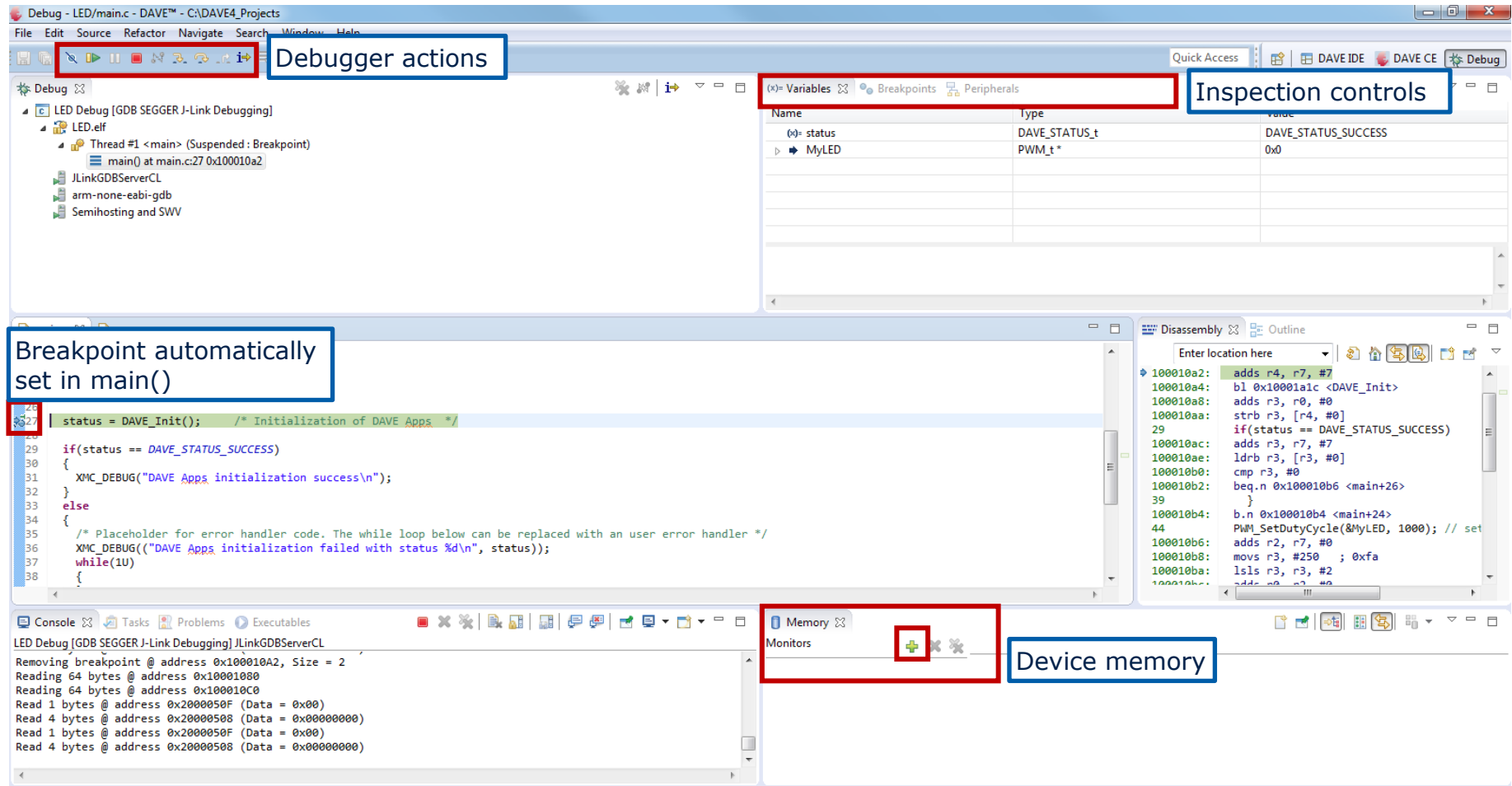


- Hint: To switch to Project Workspace Perspective, click DAVE CE at upper right corner of window



# The Debug Perspective (1/6)

## Debug Workspace



The screenshot displays the Infineon DAVE IDE's Debug Perspective. The interface is divided into several panels:

- Debugger actions:** A toolbar at the top left containing icons for running, stepping, and other debugging actions.
- Inspection controls:** A panel on the top right showing the current state of the debug session, including variables and breakpoints.
- Breakpoint automatically set in main():** A callout pointing to a breakpoint set at line 29 of the `main()` function in the source code.
- Device memory:** A panel at the bottom right showing the memory map of the device, with a red box highlighting the memory address range.

The main source code window shows the following code:

```
29 status = DAVE_Init(); /* Initialization of DAVE Apps */
30
31 if(status == DAVE_STATUS_SUCCESS)
32 {
33     XMC_DEBUG("DAVE Apps initialization success\n");
34 }
35 else
36 {
37     /* Placeholder for error handler code. The while loop below can be replaced with an user error handler */
38     XMC_DEBUG(("DAVE Apps initialization failed with status %d\n", status));
39     while(1)
40     {
41     }
```

The console window at the bottom left shows the output of the debug session:

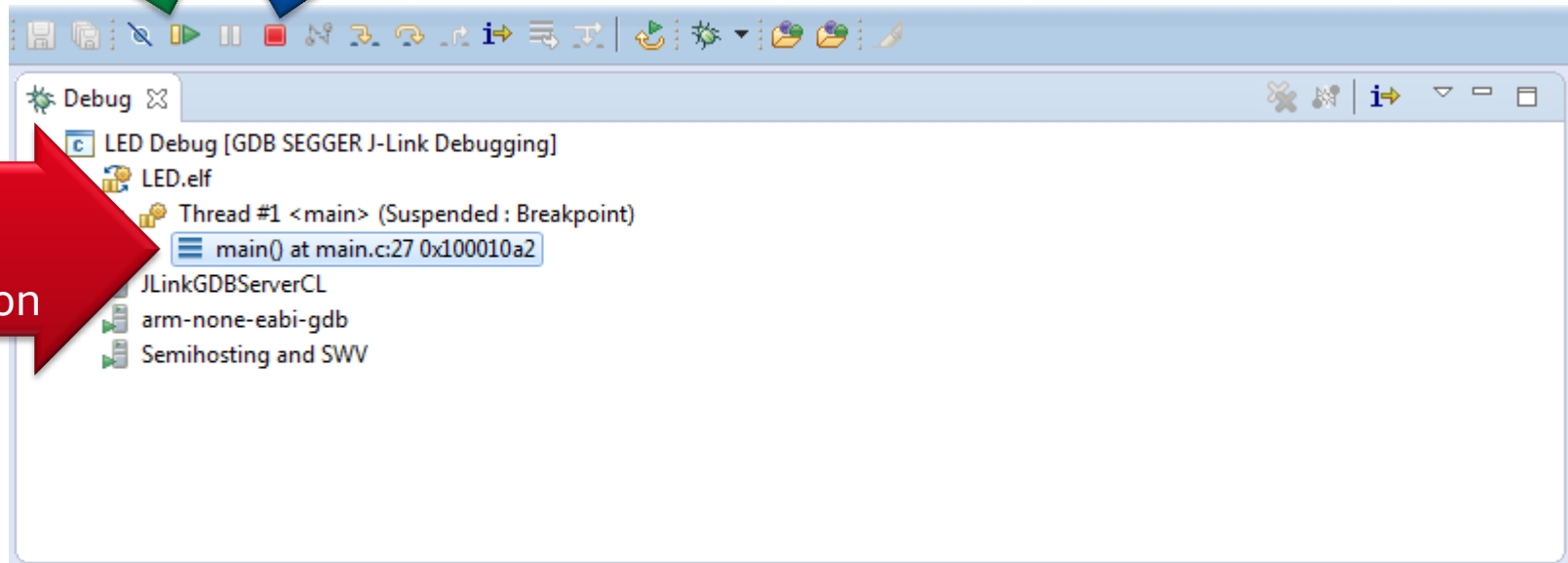
```
LED Debug [GDB SEGGER J-Link Debugging] JLinkGDBServerCL
Removing breakpoint @ address 0x100010A2, Size = 2
Reading 64 bytes @ address 0x10001080
Reading 64 bytes @ address 0x100010C0
Read 1 bytes @ address 0x2000050F (Data = 0x00)
Read 4 bytes @ address 0x20000508 (Data = 0x00000000)
Read 1 bytes @ address 0x2000050F (Data = 0x00)
Read 4 bytes @ address 0x20000508 (Data = 0x00000000)
```

# The Debug Perspective (2/6)

## Debug Window



Debug  
Session  
information

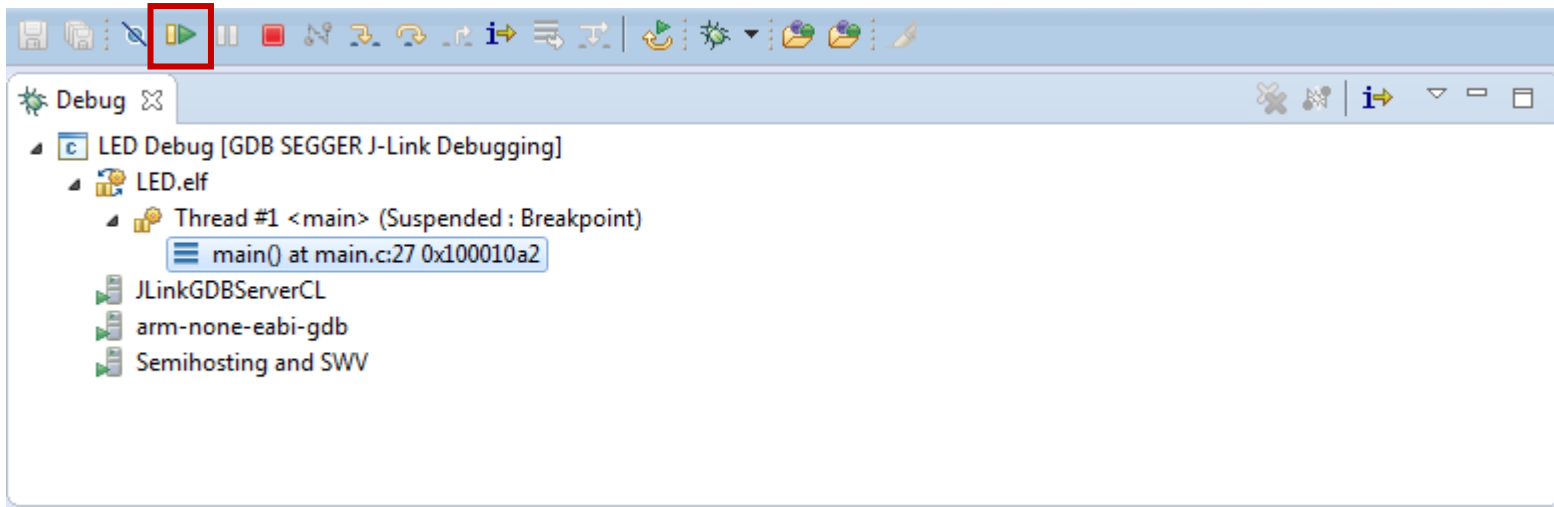




# The Debug Perspective (3/6)

## Start Program

- Click on the Resume button to start code execution

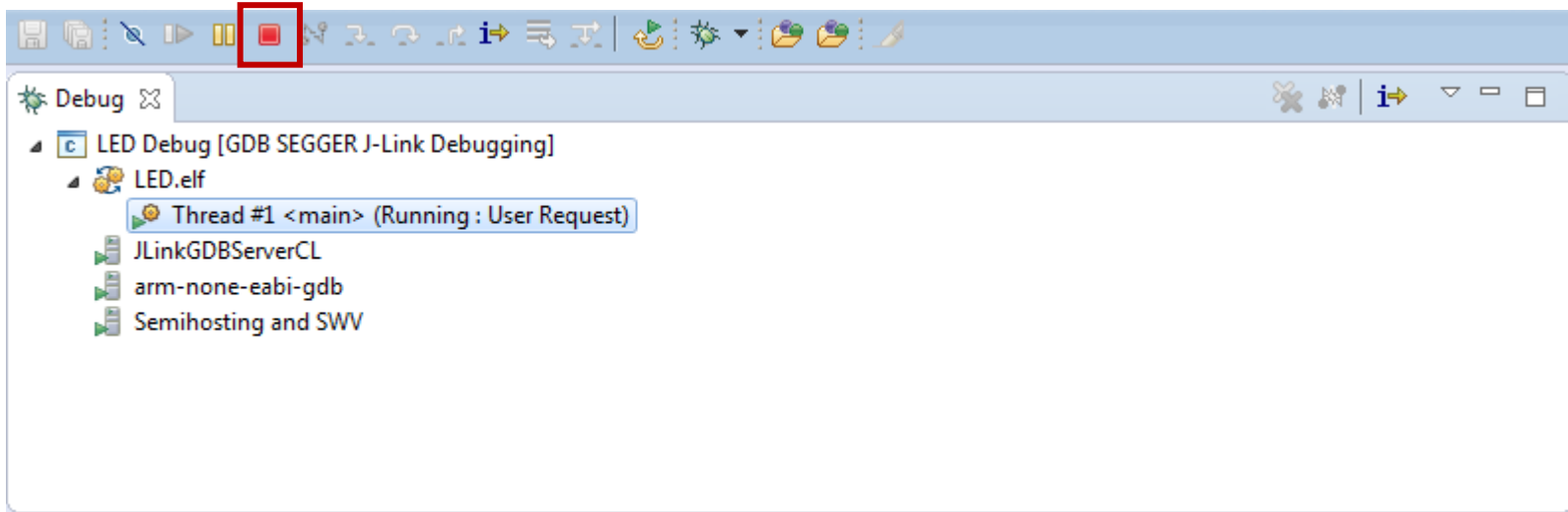


- User LED1 (P0.0) on XMC1200 BootKit board should be blinking

# The Debug Perspective (6/6)

## End Debug Session

- Always end a debug session by clicking the Terminate Button

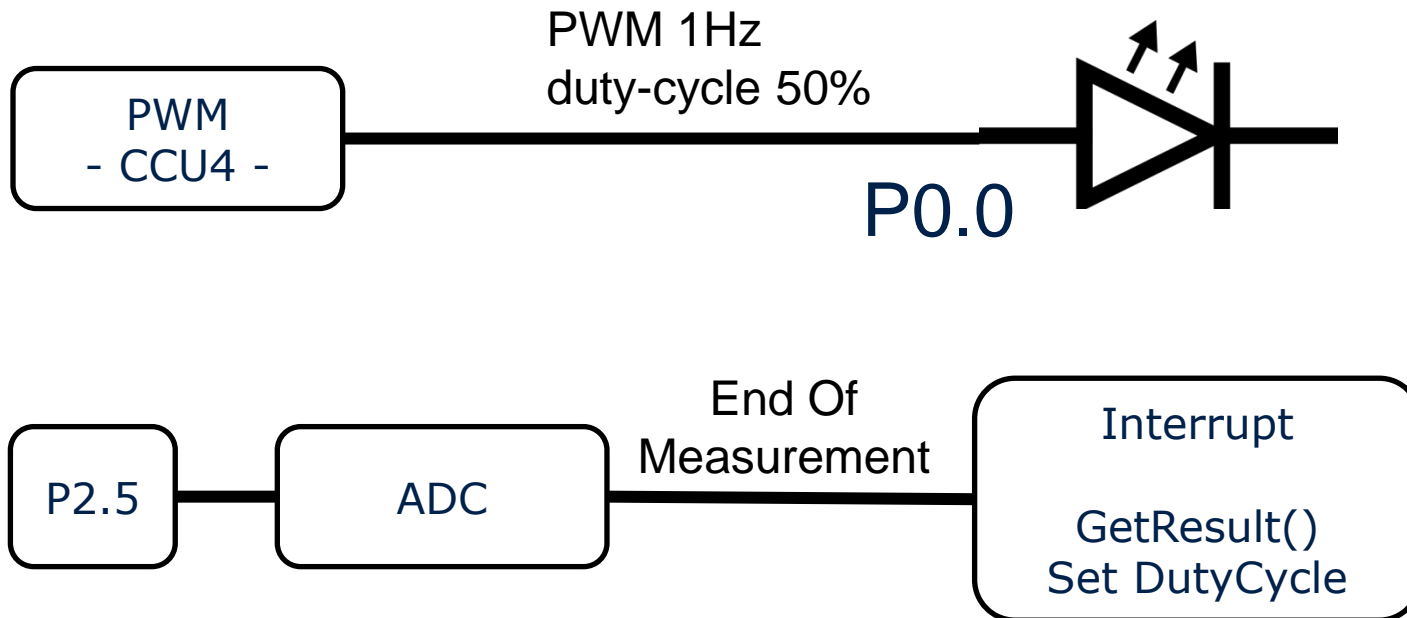


# Agenda

- LLD of XMCLib: use of ACMP
- Blink an LED with PWM APP
- Use ADC APP to update PWM dutycycle

# PWM to Blink an LED

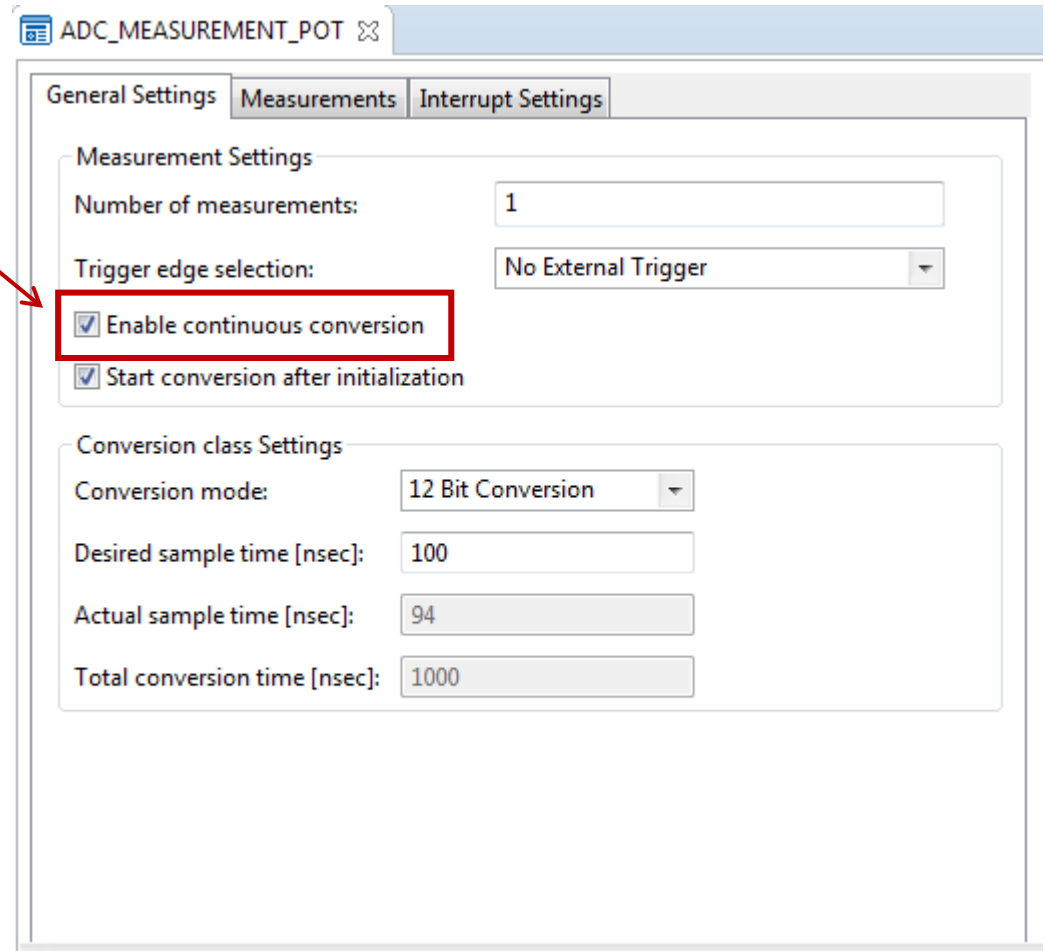
- Add ADC Measurement APP to read potentiometer, then update PWM dutycycle when ADC is ready.





# Configure ADC\_MEASUREMENT APP [1/2]

## ■ Enable Continuous conversion



ADC\_MEASUREMENT\_POT

General Settings | **Measurements** | Interrupt Settings

Measurement Settings

Number of measurements: 1

Trigger edge selection: No External Trigger

☒ Enable continuous conversion

☒ Start conversion after initialization

Conversion class Settings

Conversion mode: 12 Bit Conversion

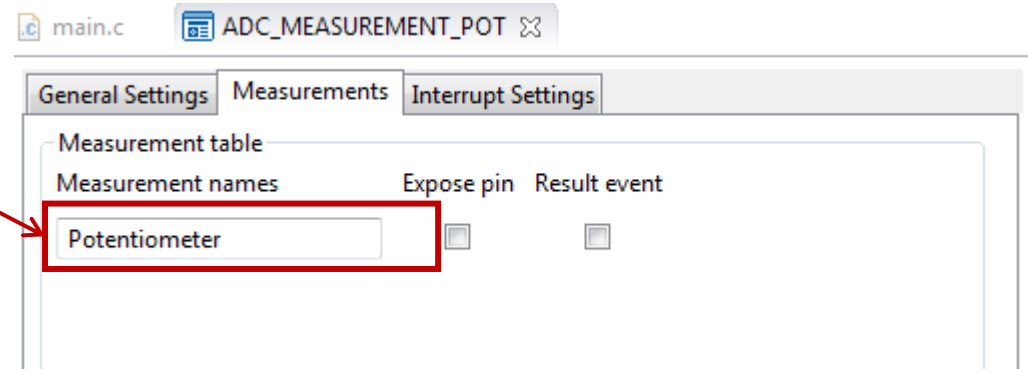
Desired sample time [nsec]: 100

Actual sample time [nsec]: 94

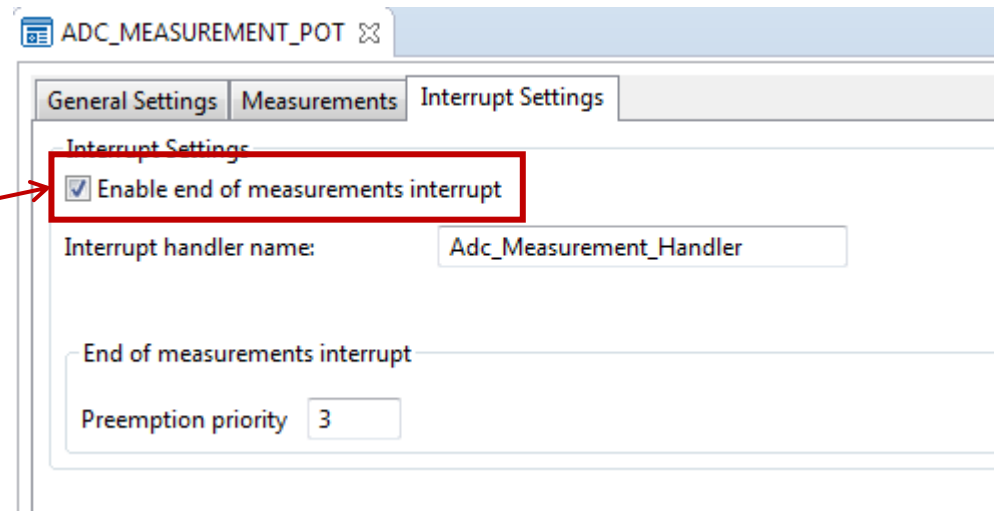
Total conversion time [nsec]: 1000

# Configure ADC\_MEASUREMENT APP [2/2]

- In "Measurements"  
Change name from  
"Channel\_A" to  
"Potentiometer"



- In "Interrupt setting"  
Set "Enable end of  
measurements  
interrupt"



# Generate Code and add a few Lines of Code to change the Duty Cycle of the PWM

## ■ One touch code generation

1. Click  in the tool panel

## ■ ADD this code

```
58     ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_0);
59     /* Placeholder for user application code. The while loop below can be replaced with user application code. */
60     while(1U)
61     {
62     }
63 }
64
65 void Adc_Measurement_Handler()
66 {
67     uint32_t result;
68     result = ADC_MEASUREMENT_GetResult(&ADC_MEASUREMENT_Potentiometer_handle);
69     result = (result*MyLED.config_ptr->config_value->period_value)>>12;
70     PWM_SetDutyCycle(&MyLED, result);
71 }
```

## ■ Start Compiler tools to build the project

□ Click  in the tool panel

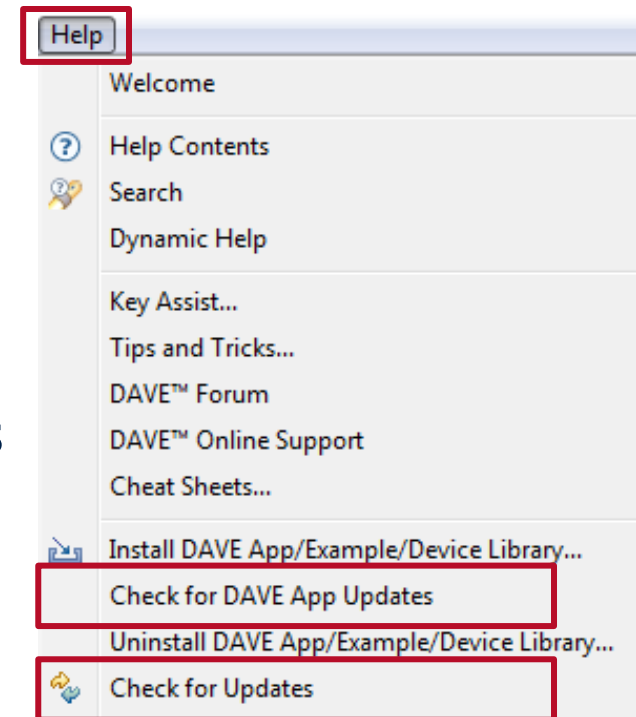
## ■ Start Debug Session

□ Click  in the tool panel



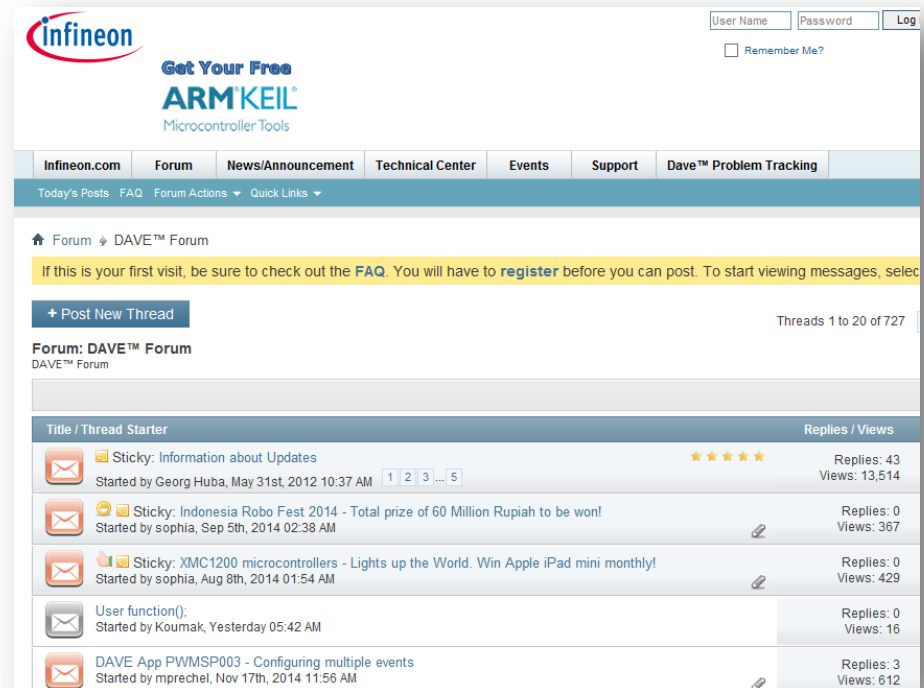
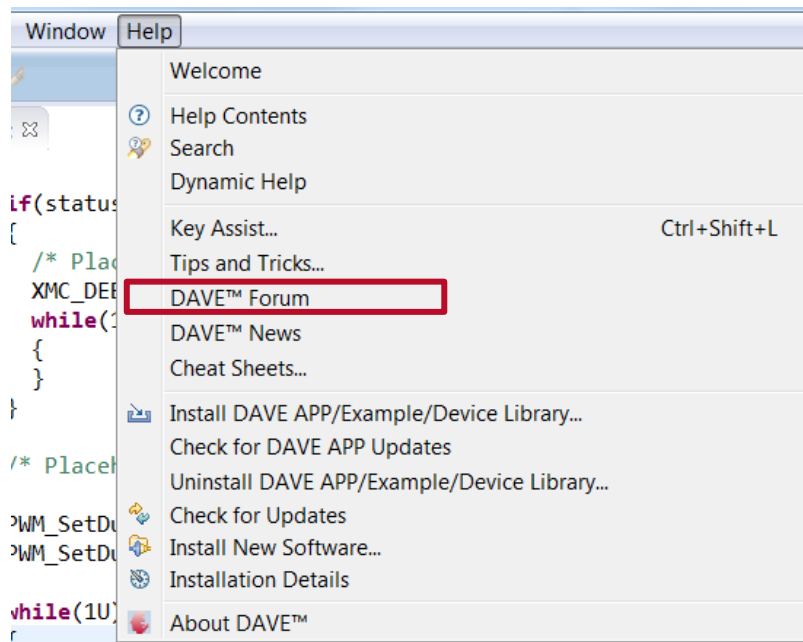
# One-click DAVE™ update

- DAVE™ APPs and device support can be updated locally
- Re-installation not required
  
- Update DAVE™ system
  - Help → Check for Updates
  
- Update DAVE™ APPs and device support
  - Help → Check for DAVE App Updates



# Expert Support

- Easy access to DAVE™ technical support, downloads and information updates



**DAVE™ Forum**

# ENERGY EFFICIENCY MOBILITY SECURITY

Innovative semiconductor solutions for energy efficiency, mobility and security.

