Freescale Semiconductor

Getting started with CDE in Kinetis Design Studio

By: Carlos Mendoza / Technical Information Center

freescale semiconductor

About this document

This document introduces the usage of Component Development Environment (CDE) by creating a software component that will use the Kinetis Software Development Kit (KSDK) drivers to control a 16x2 LCD.

The code provided on the below document will be used as starting point for the creation of the component:

Driving 16x2 LCD using KSDK drivers: https://community.freescale.com/docs/DOC-329190

Software versions

The steps described in this document are valid for the following versions of the software tools:

- KDS v3.0.0
- KSDK v1.3.0

The links to download the software can be found on the **Appendix A.**

Contents

1. Glossary

CDE ***Component Development Environment*:** Eclipse plug-in tool that provides a graphical interface to create, edit, and package embedded components.

KDS ***Kinetis Design Studio*:** Integrated Development Environment (IDE) software for Kinetis MCUs.

KSDK ***Kinetis Software Development Kit*:** Set of peripheral drivers, stacks and middleware layers for Kinetis microcontrollers.

2. Overview and concepts

2.1 Processor Expert Software

Processor Expert Software is a development system to create, configure, optimize, migrate, and deliver software components that generate source code for Freescale silicon. Processor Expert software covers Freescale's S08/RS08, S12(X), ColdFire, ColdFire+, Kinetis, DSC 56800/E, QorIQ and some other Power Architecture® processors. Processor Expert software is available as part of the CodeWarrior tool suite or as an Eclipse-based plug-in feature for installation into an independent Eclipse environment.

You can find more information about processor Expert on the following link:

http://mcuoneclipse.com/2015/10/18/overview-processor-expert/

2.1.1 Embedded Components

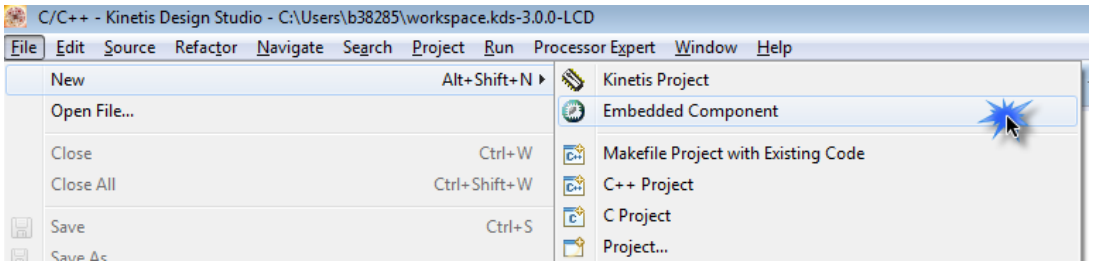
An embedded component is a software entity that exposes a specific set of methods, properties, and events providing an abstraction for peripheral I/O and CPUs. An embedded component may also encapsulate/package a software stack or RTOS adapter.

2.1.1.1 Terminology

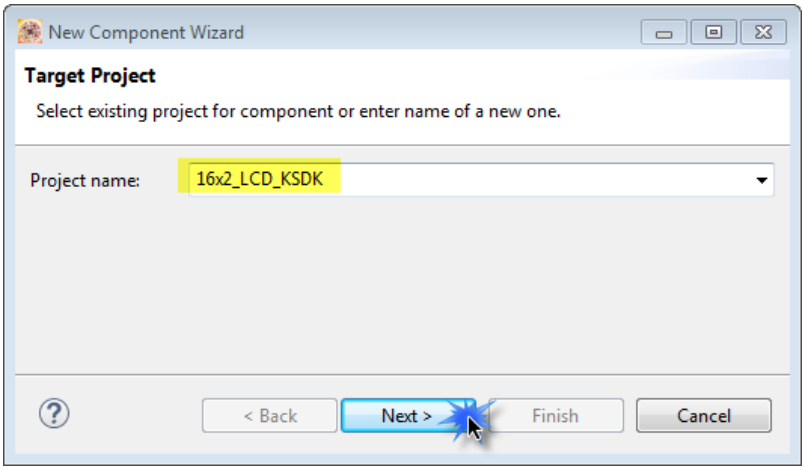
- **Properties-** The initialization state and features supported at runtime by the component; the properties can only be defined at design time.
- **Methods-** These are the functions exposed by the component, available at runtime and are used to set and read the component state or implement behavior.
- **Events-** These are callback functions exposed by the component to attend to asynchronous events such as interrupts.
- **Driver-** A driver contains the source code of all methods and events generated by a component. Every component, except the CPU, has a driver associated with it.
- **Inheritance-** The process that allows using and/or redefining methods and events of another component.
- **Interfaces-** Defines the methods and events needed by a new component that uses the interface.

3. Creating a new embedded component

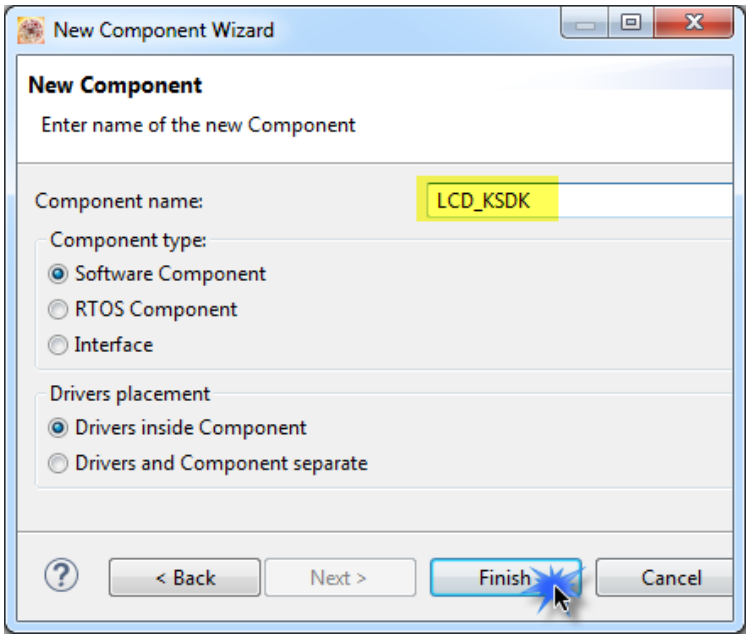
- In order to start creating a **new** component create a new Embedded Component project in KDS, to do this go to menu **File** > **New** > **Embedded Component:**



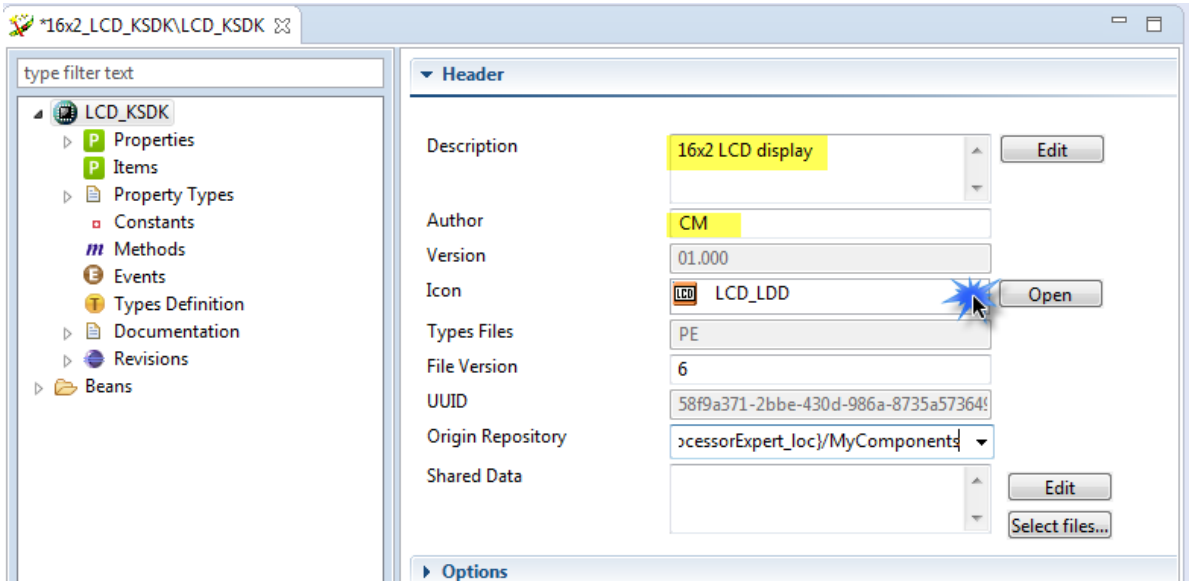
- Give a name to your project and click on **Next**:



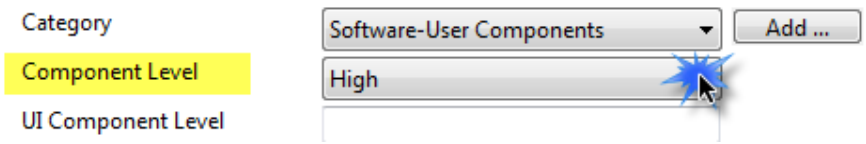
- Give a name to the component, for this example choose the **Component type** to be **Software Component**, the drivers placement to be inside the component and click on Finish:



- The next step is to fill the component's general information. In the **Header** tab set a brief **description** of the component, the **author**, select an **Icon** or choose a custom one, specify the **Origin Repository** where the component will be distributed, in this case it will be MyComponents:



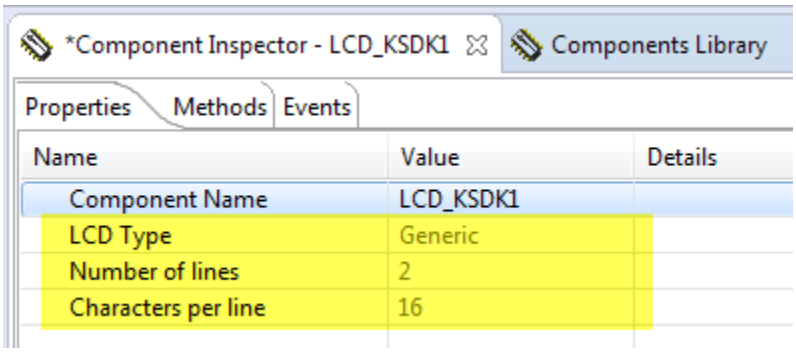
- On the **Options** tab, leave everything as is except for the **Component Level**, change this option to **High**.



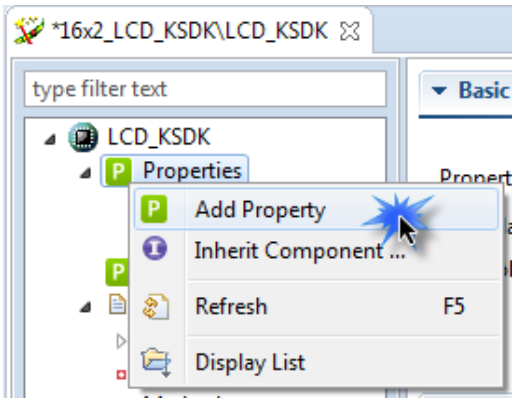
- And for the **Help** and **Compiler** tab leave everything as is.

3.1 Adding properties

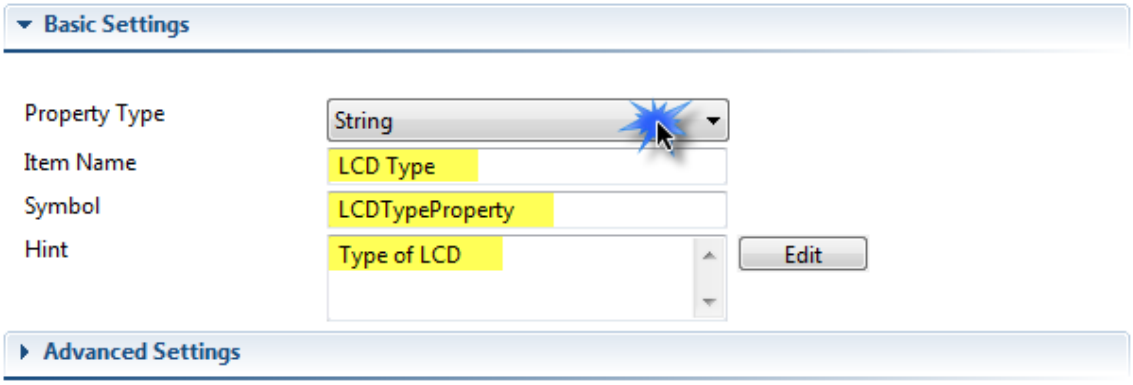
- The component will have 3 read-only properties: **LCD Type, Number of lines** and **Characters per line**.



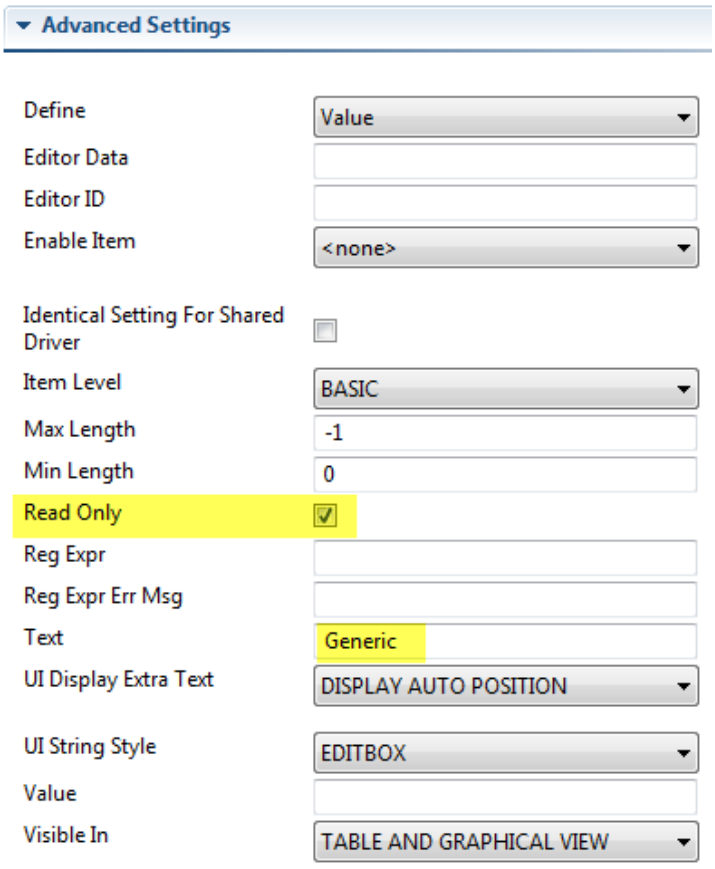
- To add these properties, right-click on the **Properties** in the Component editor and select **Add Property** option:



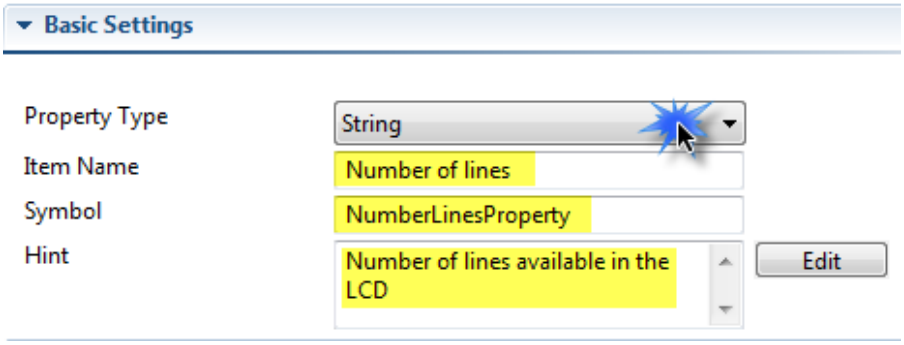
- The Properties editor will open, on the **Basic Settings** change the Property Type to String and set the **Name**, **Symbol** and **Hint** for the "LCD Type" property:



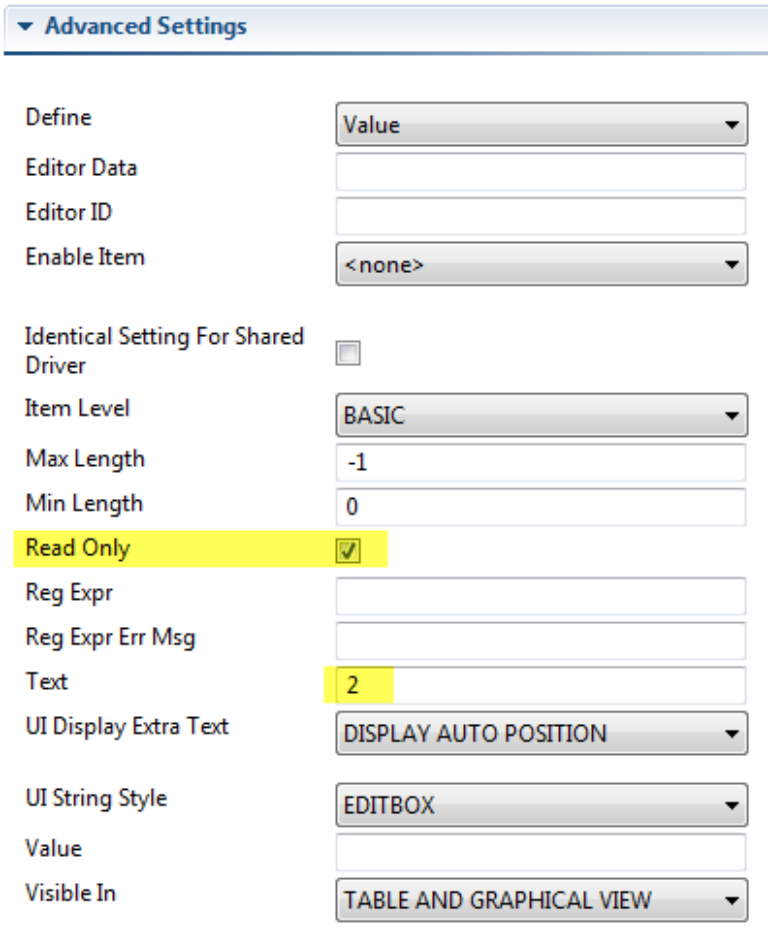
- On the **Advanced Settings** tab and enable the **Read Only** option and set the value for the property, in this case **Generic** will be the value for the **LCD Type** property.



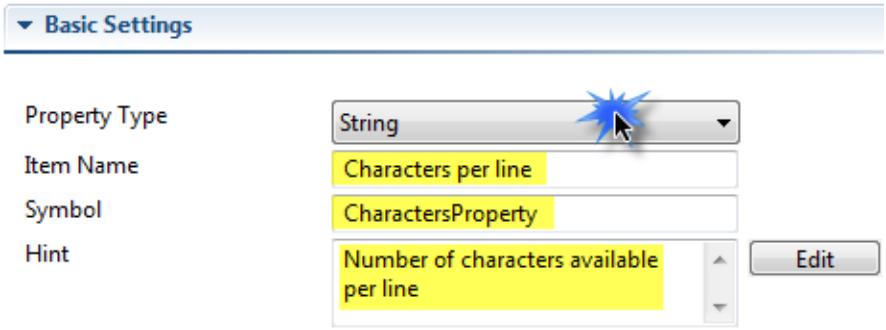
- The same steps apply for the **Number of lines** property:



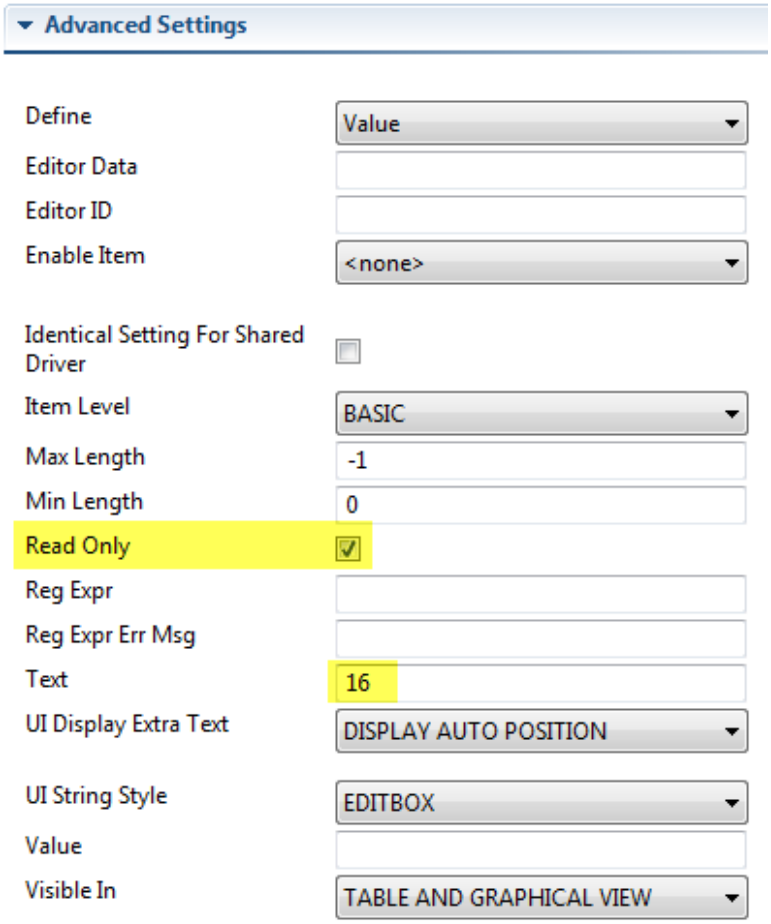
- **2** will be the value for the **Number of lines** property.



- And for the **Characters per line** property:



- **16** will be the value for the **Characters per line** property.



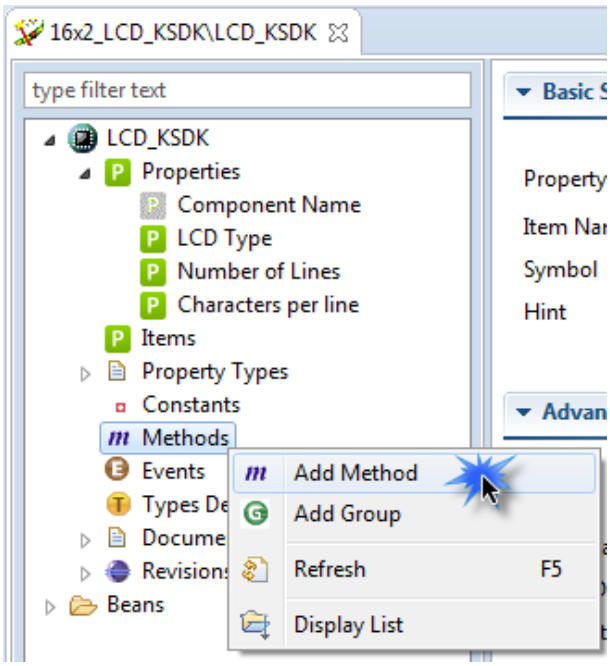
3.2 Adding a method

The document "**Driving 16x2 LCD using KSDK drivers"** (https://community.freescale.com/docs/DOC-329190) uses the *lcd_init* function to print a message on the LCD:

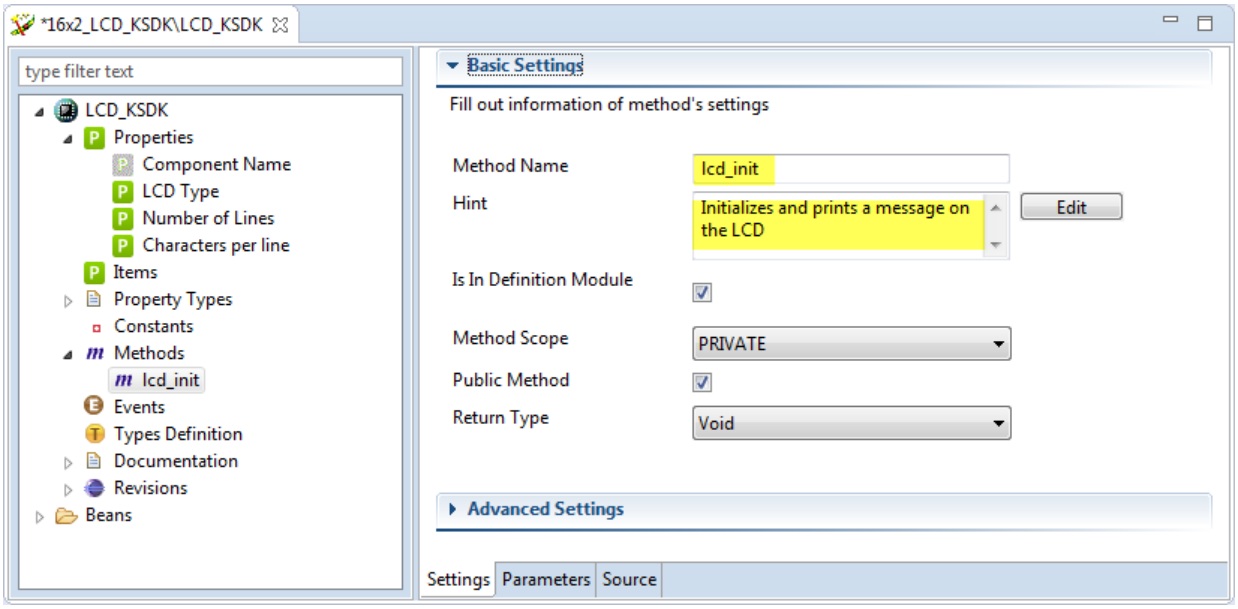
```
//-----------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------

int main(void)
{

    // Configure board specific pin muxing
    hardware_init();

    // Initialize UART terminal
    dbg_uart_init();

    PRINTF("\r\nRunning the myProject project.\r\n");

    lcd_init();

    for (;;)                                        // Forever loop
    {
        __asm("NOP");
    }


}
```

So add this function as a method in the component.

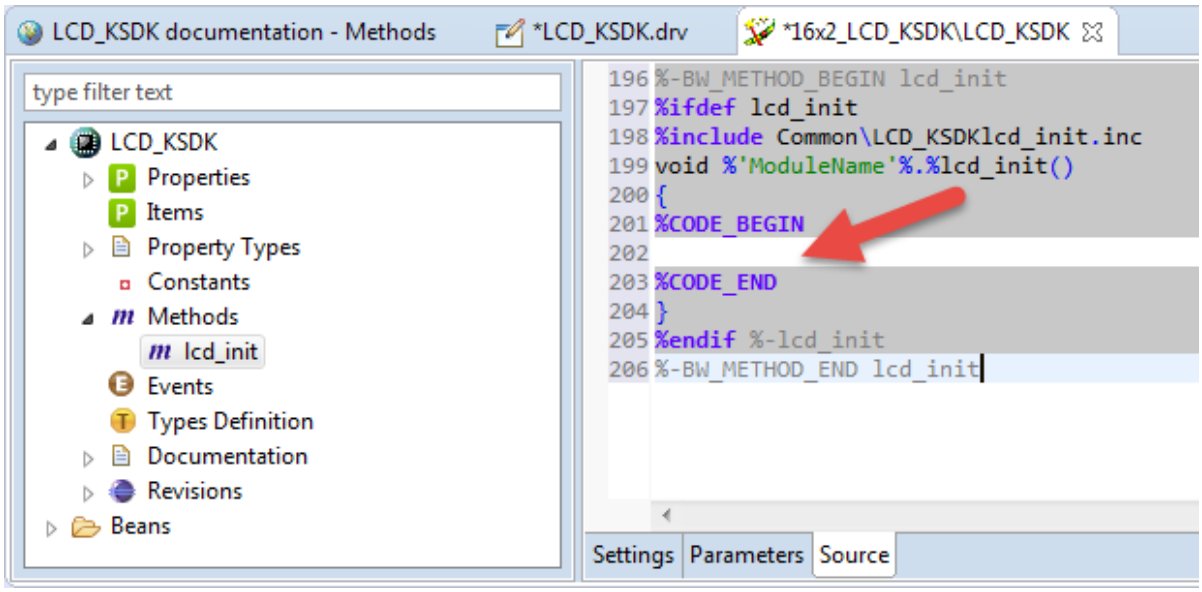
- To do this make a right click on **Methods** and click on **Add Method**:



- The Method editor will open, on the **Basic Settings** set the Name and Hint for the "**lcd_init**" method:



- Go to the **Source** tab and locate the section pointed below:



- Then add the following code on that section:

```
	LCD_Pin_Enable();   // Enable pins

	delay(100);         //Display initialization

	SetUp();

	instruction(0x80); //First line
	text((unsigned char *)&upper_line[0]);

	instruction(0xC0); //Second line
	text((unsigned char *)&lower_line[0]);

	instruction(0x0F); //Cursor on blinking
}

//-------------------------------------------------------------------------

void enable(void){
	LCD_ENABLE_ON;
	delay(4);
	LCD_ENABLE_OFF;
}

//-------------------------------------------------------------------------

void SetUp(){
	unsigned char a = 0;

	while(initLCD[a])
		{
			instruction(initLCD[a]);
			a++;
		}
}

//-------------------------------------------------------------------------

void instruction(unsigned char x){
		LCD_RS_OFF;

		lcd_data(x&0xF0);
		enable();

		lcd_data((x<<4)&0xF0);
		enable();
}
```

```
//---------------------------------------------------------------------

void lcd_data(unsigned char x){
      //Bit 7
      if (x&0x80)  {
        LCD_D7_ON;
      }
      else   {
        LCD_D7_OFF;
      }

      //Bit 6
      if (x&0x40)  {
        LCD_D6_ON;
      }
      else   {
        LCD_D6_OFF;
      }

      //Bit 5
      if (x&0x20)  {
        LCD_D5_ON;
      }
      else   {
        LCD_D5_OFF;
      }

      //Bit 4
      if (x&0x10)  {
        LCD_D4_ON;
      }
      else   {
        LCD_D4_OFF;
      }
}

//---------------------------------------------------------------------

void text (unsigned char *b){
      while(*b)
      {
            info(*b);
            b++;
      }
}
```

```
//---------------------------------------------------------------------

void info(unsigned char x)
{
	if (x == 'ñ') x = 238;

	LCD_RS_ON;

	lcd_data( x&0xF0 );
	enable();

	lcd_data( (x<<4)&0xF0 );
	enable();
}


//---------------------------------------------------------------------

void LCD_Pin_Enable(void){
	LCD_ENABLE_EN;
	LCD_RS_EN;
	LCD_D7_EN;
	LCD_D6_EN;
	LCD_D5_EN;
	LCD_D4_EN;
}

//---------------------------------------------------------------------

void delay(unsigned long time){
	while (count <= time){}
	count = 0;
```

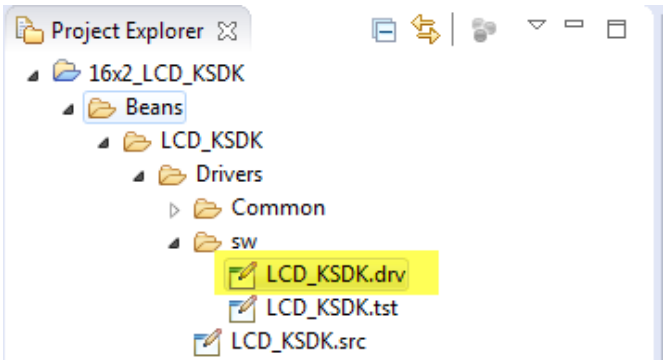
Note: The first and last Curly braces were intentionally left out, these will be added automatically.

- The result should be as follows:

```
%-BW_METHOD_BEGIN lcd_init
%ifdef lcd_init
%include Common\LCD_KSDKlcd_init.inc
void %'ModuleName'%.%lcd_init()
{
%CODE_BEGIN
  LCD_Pin_Enable();   // Enable pins

  delay(100);     //Display initialization

  SetUp();

  instruction(0x80); //First line
  text((unsigned char *)&upper_line[0]);

  instruction(0xC0); //Second line
  text((unsigned char *)&lower_line[0]);

  instruction(0x0F); //Cursor on blinking
}
```

```
//-------------------------------------------------------------------------

void LCD_Pin_Enable(void){
  LCD_ENABLE_EN;
  LCD_RS_EN;
  LCD_D7_EN;
  LCD_D6_EN;
  LCD_D5_EN;
  LCD_D4_EN;
}

//-------------------------------------------------------------------------

void delay(unsigned long time){
  while (count <= time){}
  count = 0;
%CODE_END
}
%endif %-lcd_init
%-BW_METHOD_END lcd_init
```

- The next step is to add the necessary includes, macros, variable definitions and function prototypes needed by the method, all this needs to be added on the following file:



- Double click on it and the editor will open, the includes will be added in the following section:

```
80 %-BW_CUSTOM_INCLUDE_START_H
81 %- Write your own includes here ...
82 %-   Example:
83 %-     #include "header_name.h"
84 #include "board.h"
85 #include "lcdPins.h"
86 %-
87 %-BW_CUSTOM_INCLUDE_END_H
88 %-
```

Code:

```
#include "board.h"
#include "lcdPins.h"
```

- The macro definitions and function prototypes will be added in the following section:

```
%-BW_METHOD_MACROS_START
/* lcd mapping */
#define LCD_ENABLE_EN (GPIO_DRV_OutputPinInit(&lcdPins[0])) /*!< Enable target LCD Enable */
#define LCD_RS_EN (GPIO_DRV_OutputPinInit(&lcdPins[1])) /*!< Enable target LCD RS */
#define LCD_D7_EN (GPIO_DRV_OutputPinInit(&lcdPins[2])) /*!< Enable target LCD D7*/
#define LCD_D6_EN (GPIO_DRV_OutputPinInit(&lcdPins[3])) /*!< Enable target LCD D6*/
#define LCD_D5_EN (GPIO_DRV_OutputPinInit(&lcdPins[4])) /*!< Enable target LCD D5*/
#define LCD_D4_EN (GPIO_DRV_OutputPinInit(&lcdPins[5])) /*!< Enable target LCD D4*/

#define LCD_ENABLE_OFF (GPIO_DRV_WritePinOutput(lcdPins[0].pinName, 0)) /*!< Turn off target LCD Enable */
#define LCD_RS_OFF (GPIO_DRV_WritePinOutput(lcdPins[1].pinName, 0)) /*!< Turn off target LCD RS */
#define LCD_D7_OFF (GPIO_DRV_WritePinOutput(lcdPins[2].pinName, 0)) /*!< Turn off target LCD D7*/
#define LCD_D6_OFF (GPIO_DRV_WritePinOutput(lcdPins[3].pinName, 0)) /*!< Turn off target LCD D6*/
#define LCD_D5_OFF (GPIO_DRV_WritePinOutput(lcdPins[4].pinName, 0)) /*!< Turn off target LCD D5*/
#define LCD_D4_OFF (GPIO_DRV_WritePinOutput(lcdPins[5].pinName, 0)) /*!< Turn off target LCD D4*/

#define LCD_ENABLE_ON (GPIO_DRV_WritePinOutput(lcdPins[0].pinName, 1)) /*!< Turn on target LCD Enable */
#define LCD_RS_ON (GPIO_DRV_WritePinOutput(lcdPins[1].pinName, 1)) /*!< Turn on target LCD RS */
#define LCD_D7_ON (GPIO_DRV_WritePinOutput(lcdPins[2].pinName, 1)) /*!< Turn on target LCD D7 */
#define LCD_D6_ON (GPIO_DRV_WritePinOutput(lcdPins[3].pinName, 1)) /*!< Turn on target LCD D6 */
#define LCD_D5_ON (GPIO_DRV_WritePinOutput(lcdPins[4].pinName, 1)) /*!< Turn on target LCD D5 */
#define LCD_D4_ON (GPIO_DRV_WritePinOutput(lcdPins[5].pinName, 1)) /*!< Turn on target LCD D4 */

/* Function Prototypes */
void delay(unsigned long time);
void LCD_Pin_Enable(void);
void enable(void);
void lcd_init();
void SetUp ();
void instruction (unsigned char x);
void lcd_data(unsigned char x);
void text (unsigned char *b);
void info(unsigned char x);
%-BW_METHOD_MACROS_END
```

Code:

```
/* lcd mapping */
#define LCD_ENABLE_EN (GPIO_DRV_OutputPinInit(&lcdPins[0])) /*!< Enable target LCD Enable */
#define LCD_RS_EN (GPIO_DRV_OutputPinInit(&lcdPins[1])) /*!< Enable target LCD RS */
#define LCD_D7_EN (GPIO_DRV_OutputPinInit(&lcdPins[2])) /*!< Enable target LCD D7*/
#define LCD_D6_EN (GPIO_DRV_OutputPinInit(&lcdPins[3])) /*!< Enable target LCD D6*/
#define LCD_D5_EN (GPIO_DRV_OutputPinInit(&lcdPins[4])) /*!< Enable target LCD D5*/
#define LCD_D4_EN (GPIO_DRV_OutputPinInit(&lcdPins[5])) /*!< Enable target LCD D4*/

#define LCD_ENABLE_OFF (GPIO_DRV_WritePinOutput(lcdPins[0].pinName, 0)) /*!< Turn off target LCD
Enable */
#define LCD_RS_OFF (GPIO_DRV_WritePinOutput(lcdPins[1].pinName, 0)) /*!< Turn off target LCD RS */
#define LCD_D7_OFF (GPIO_DRV_WritePinOutput(lcdPins[2].pinName, 0)) /*!< Turn off target LCD D7*/
#define LCD_D6_OFF (GPIO_DRV_WritePinOutput(lcdPins[3].pinName, 0)) /*!< Turn off target LCD D6*/
#define LCD_D5_OFF (GPIO_DRV_WritePinOutput(lcdPins[4].pinName, 0)) /*!< Turn off target LCD D5*/
#define LCD_D4_OFF (GPIO_DRV_WritePinOutput(lcdPins[5].pinName, 0)) /*!< Turn off target LCD D4*/

#define LCD_ENABLE_ON (GPIO_DRV_WritePinOutput(lcdPins[0].pinName, 1)) /*!< Turn on target LCD
Enable */
#define LCD_RS_ON (GPIO_DRV_WritePinOutput(lcdPins[1].pinName, 1)) /*!< Turn on target LCD RS */
#define LCD_D7_ON (GPIO_DRV_WritePinOutput(lcdPins[2].pinName, 1)) /*!< Turn on target LCD D7 */
#define LCD_D6_ON (GPIO_DRV_WritePinOutput(lcdPins[3].pinName, 1)) /*!< Turn on target LCD D6 */
#define LCD_D5_ON (GPIO_DRV_WritePinOutput(lcdPins[4].pinName, 1)) /*!< Turn on target LCD D5 */
#define LCD_D4_ON (GPIO_DRV_WritePinOutput(lcdPins[5].pinName, 1)) /*!< Turn on target LCD D4 */

/* Function Prototypes */
void delay(unsigned long time);
void LCD_Pin_Enable(void);
void enable(void);
void lcd_init();
void SetUp ();
void instruction (unsigned char x);
void lcd_data(unsigned char x);
void text (unsigned char *b);
void info(unsigned char x);
```

- Finally the variable definitions will be added in the following section:

```
%-BW_CUSTOM_VARIABLE_START
%- Write your static variables here
%-   Example:
%-     static int counter1;
%-     int %'ModuleName'%.counter2;
%-
const unsigned char  upper_line[] = "K64F 16x2 LCD";
const unsigned char  lower_line[] = "Using PEx KSDK";

const unsigned char  initLCD[8]={0x02, 0x28, 0x0C,0x06,0x01,0x00};
volatile long      count;


%-BW_CUSTOM_VARIABLE_END
```

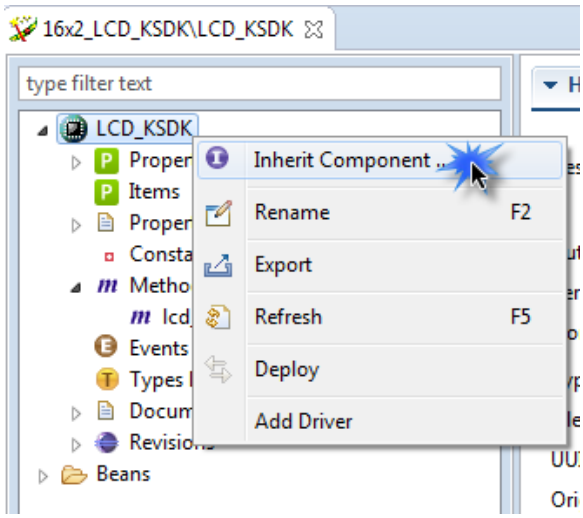
Code:

```
const unsigned char  upper_line[] = "K64F 16x2 LCD";
const unsigned char  lower_line[] = "Using PEx KSDK";

const unsigned char  initLCD[8]={0x02, 0x28, 0x0C,0x06,0x01,0x00};
volatile long              count;
```

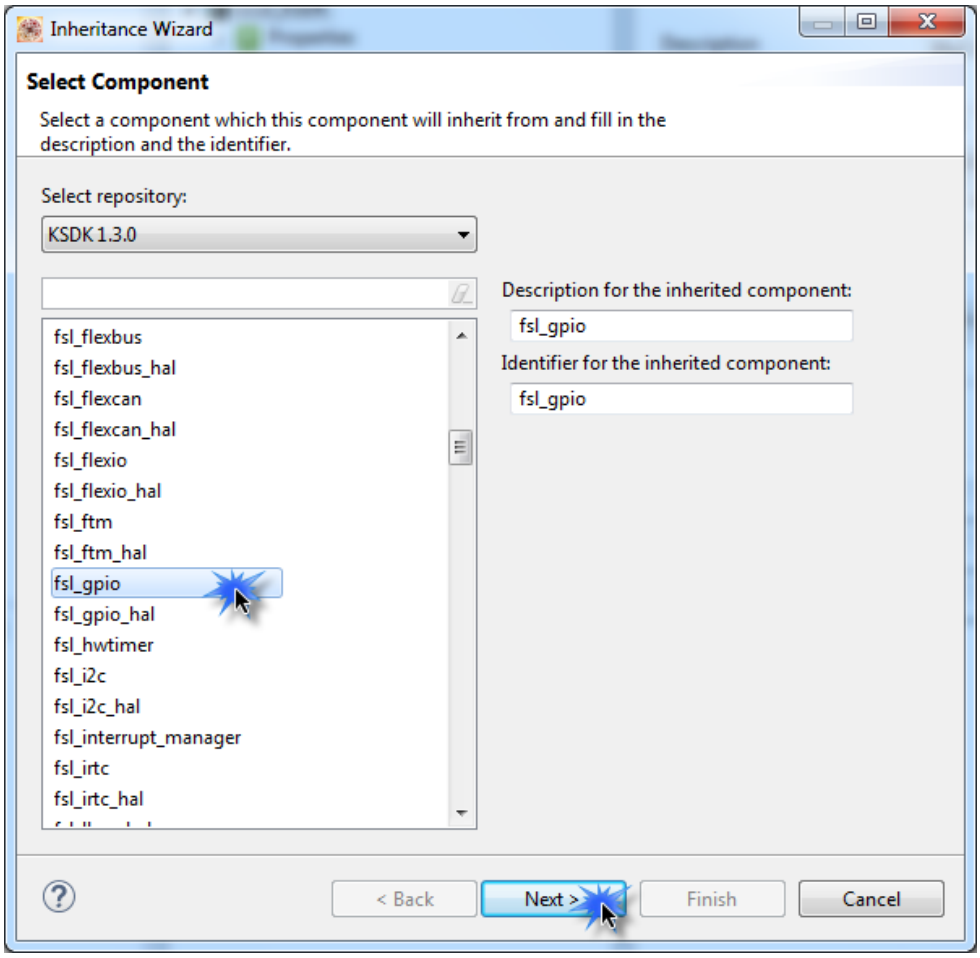
3.3 Adding inherited components

The **fsl_gpio** and **fsl_pit** components are needed for the **LCD_KSDK** component to work, so these need to be inherited.

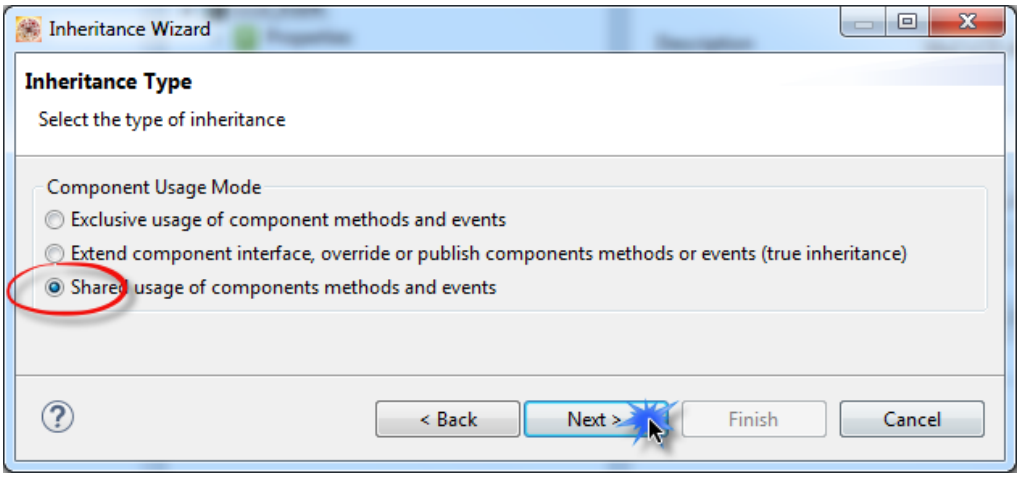
- Make a right click on the **LCD_KSDK** component and select **Inherit Component…**:



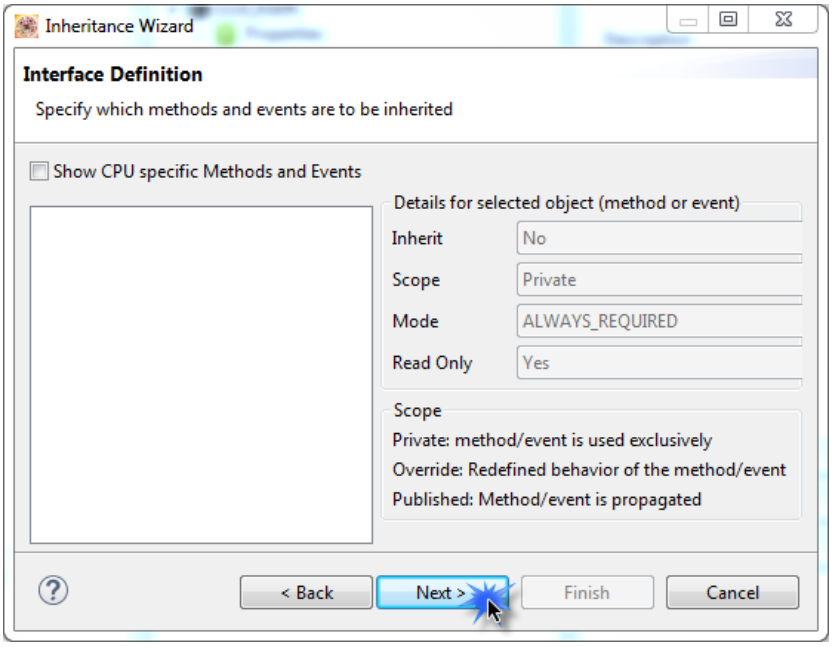
- The **Inheritance Wizard** will open, change the repository to KSDK 1.3.0 and choose the **fsl_gpio** component:



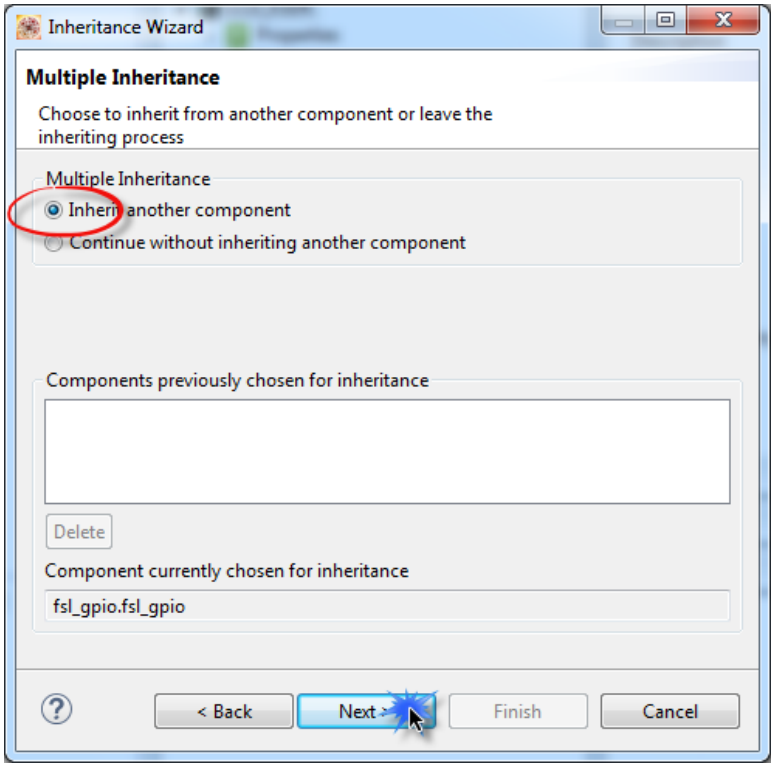
- On the next step, select the **Shared usage of components methods and events** option, click on **Next**:



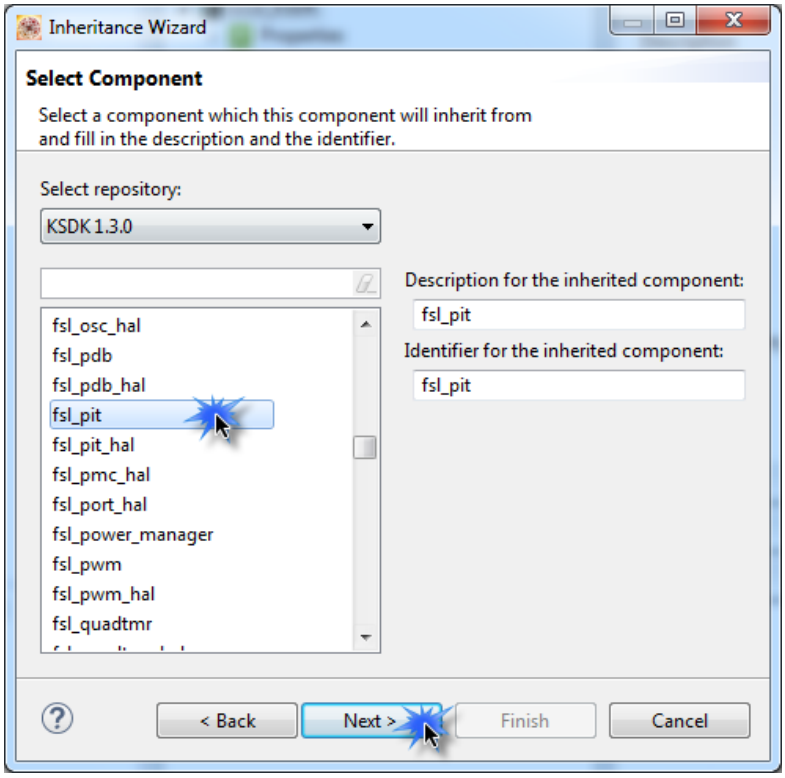
- On the following window nothing changes, click on **Next**:

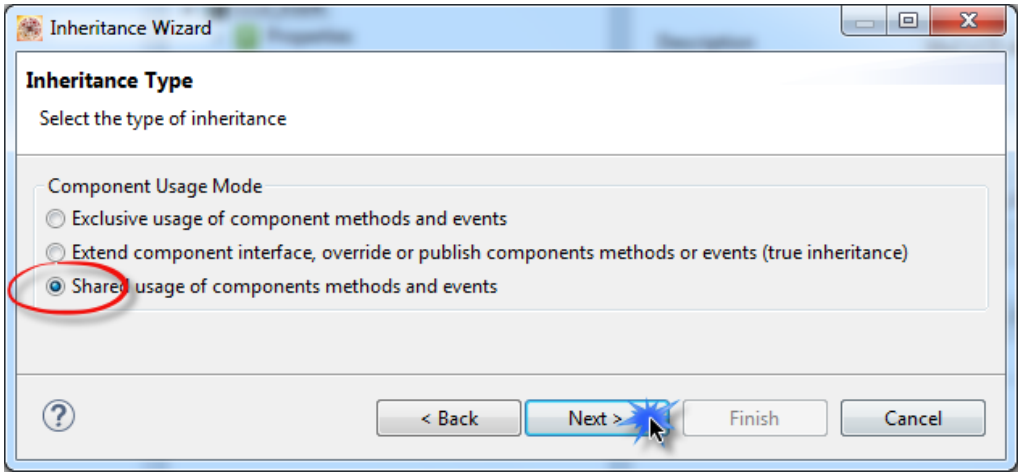


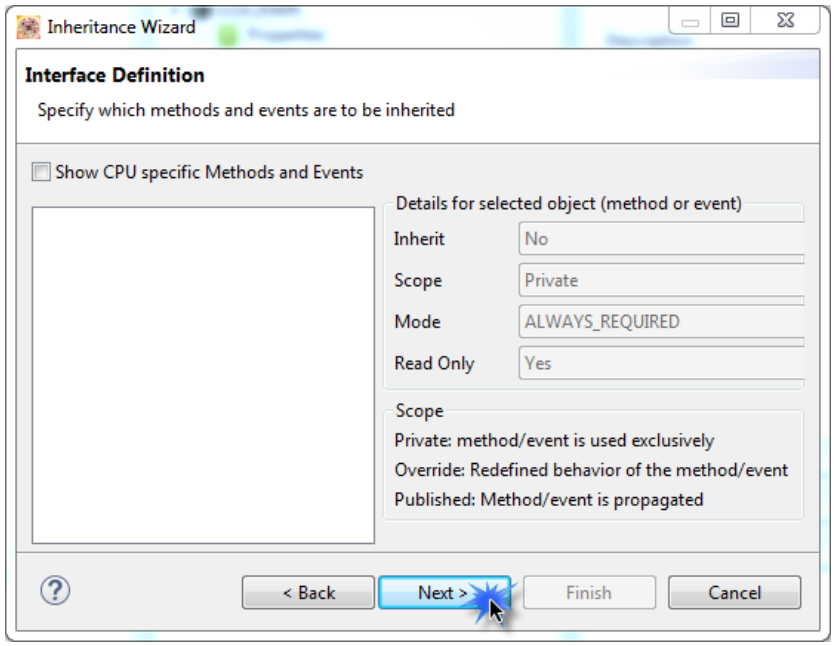
- On the **Multiple Inheritance** window select **Inherit another component** and click on **Next**:



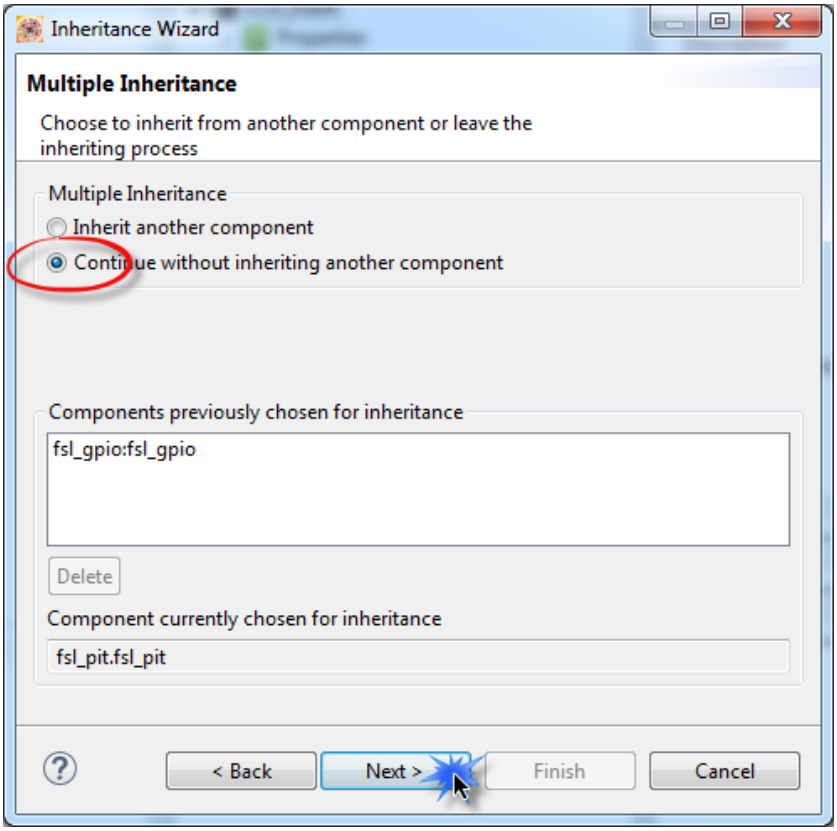
- And follow the same process as above only this time select the **fsl_pit** component:



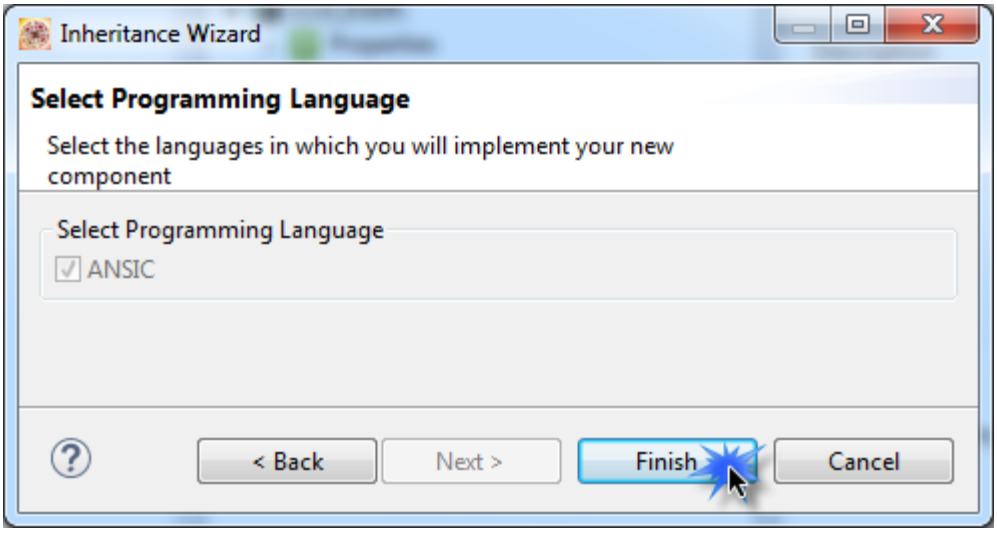




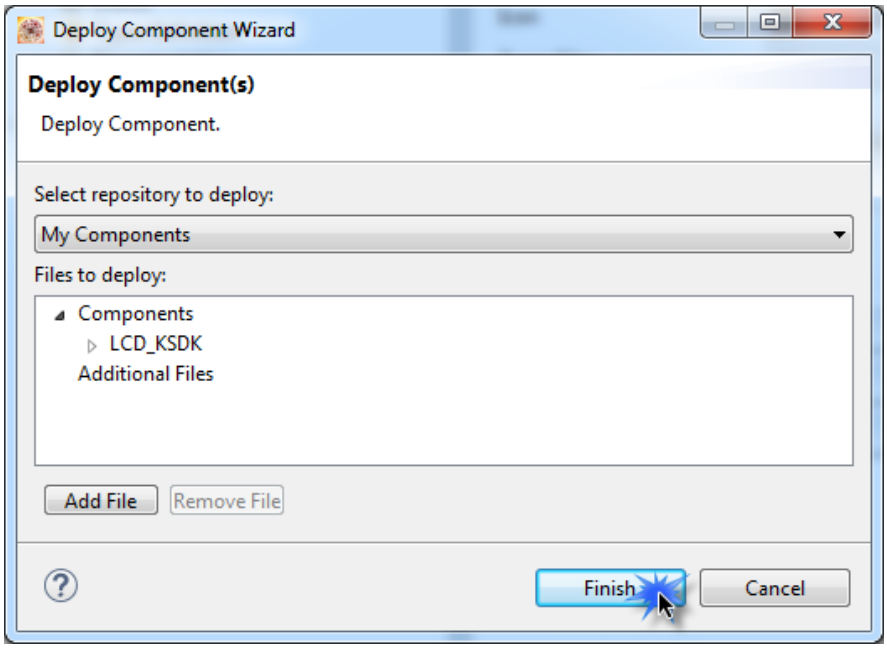
- This time, on the **Multiple Inheritance** window, select **Continue without inheriting another component** and click on **Next**:



- And click on **Finish**:



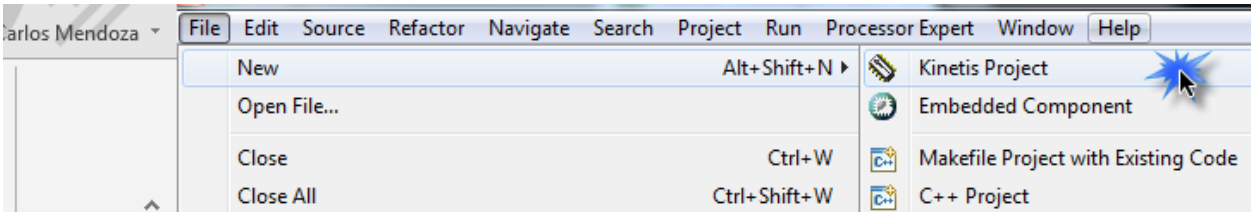
- Save the changes and the component will be deployed to the **My Components** repository:



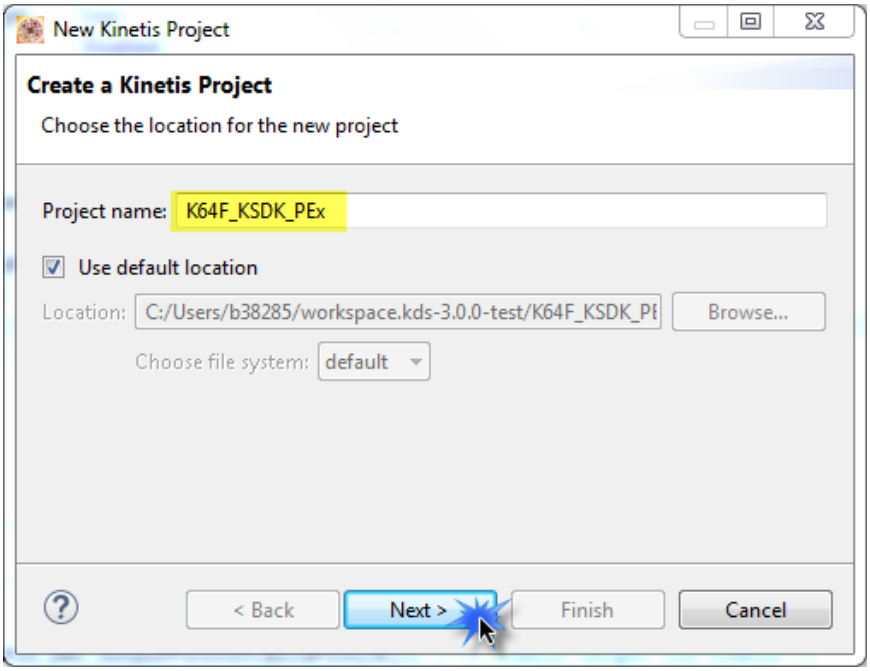
4. Testing component.

To test the component create a new KSDK project with Processor Expert enabled.

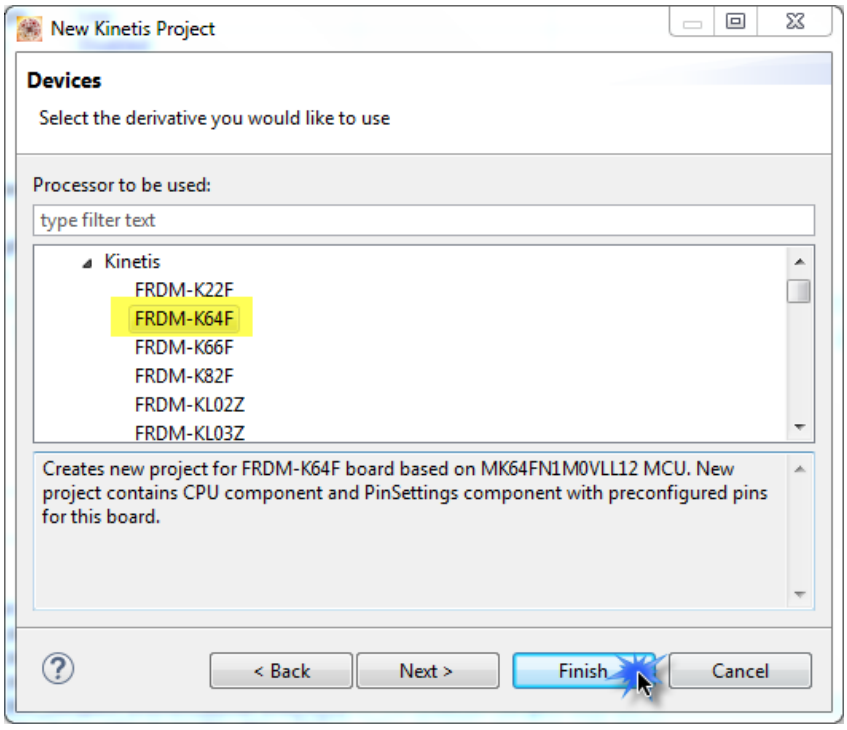
- Go to menu **File**> **New**> **Kinetis Project**:



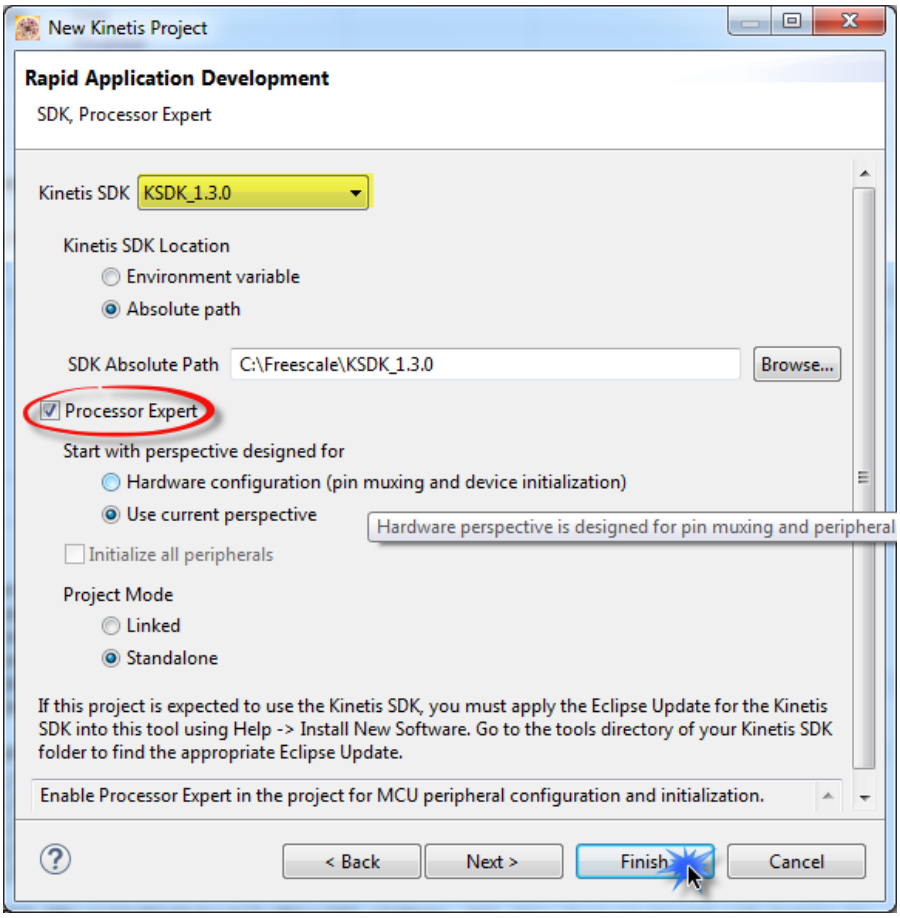
- The new project wizard will open, give a name to the project:



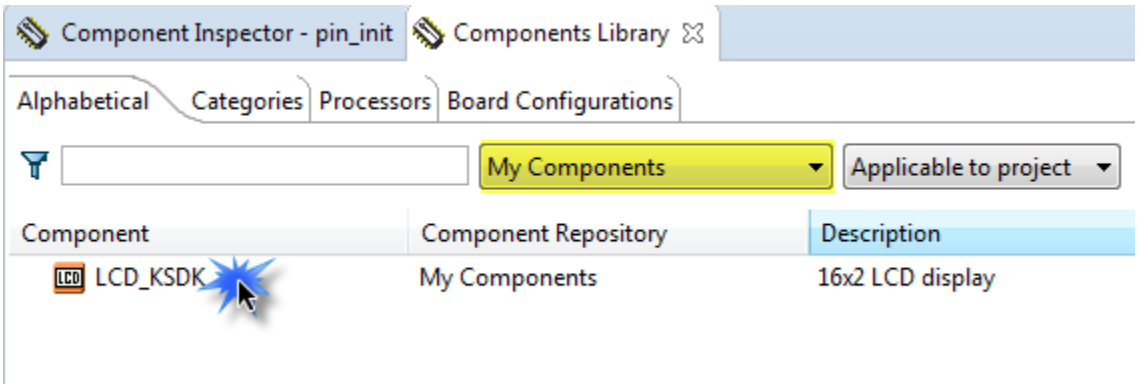
- Select the board or processor to be used, for this example the FRDM-K64F board was used:



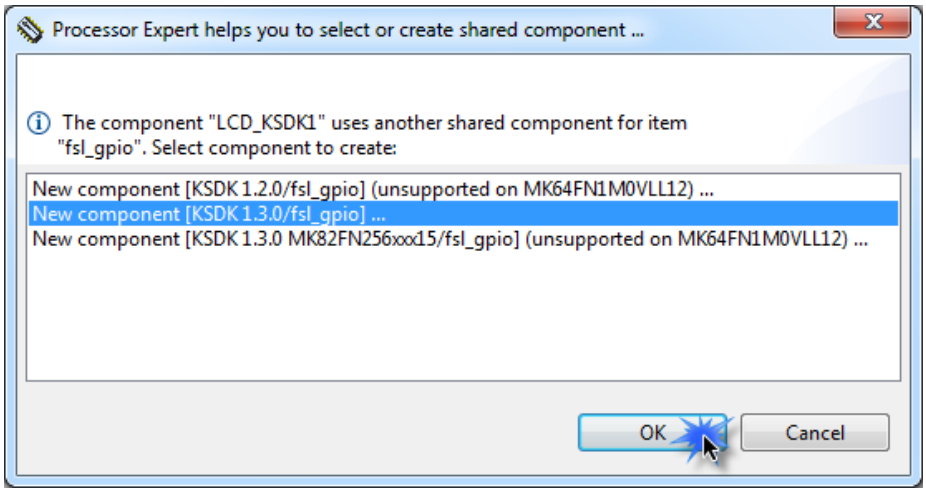
- Select the KSDK version 1.3 and enable the **Processor Expert** option:

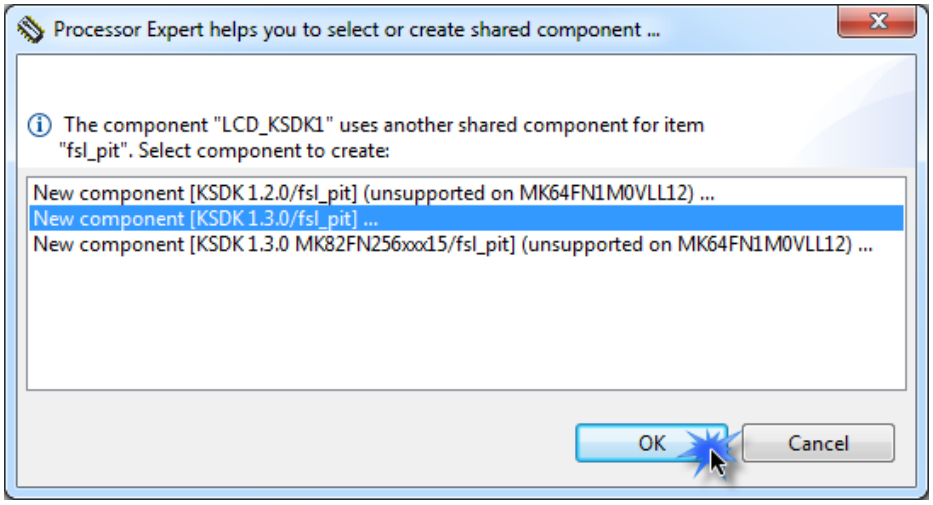


- Now go to the **Components library**, select the repository named **My Components** and add the **LCD_KSDK** component to the project:

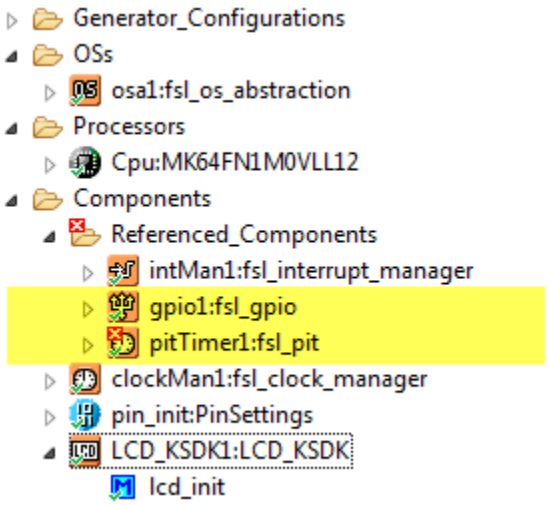


- The below dialogs will show up indicating that shared components need to be created, the **fsl_gpio** and the **fsl_pit** components:





- These components will be located inside the **Referenced_Components** folder:

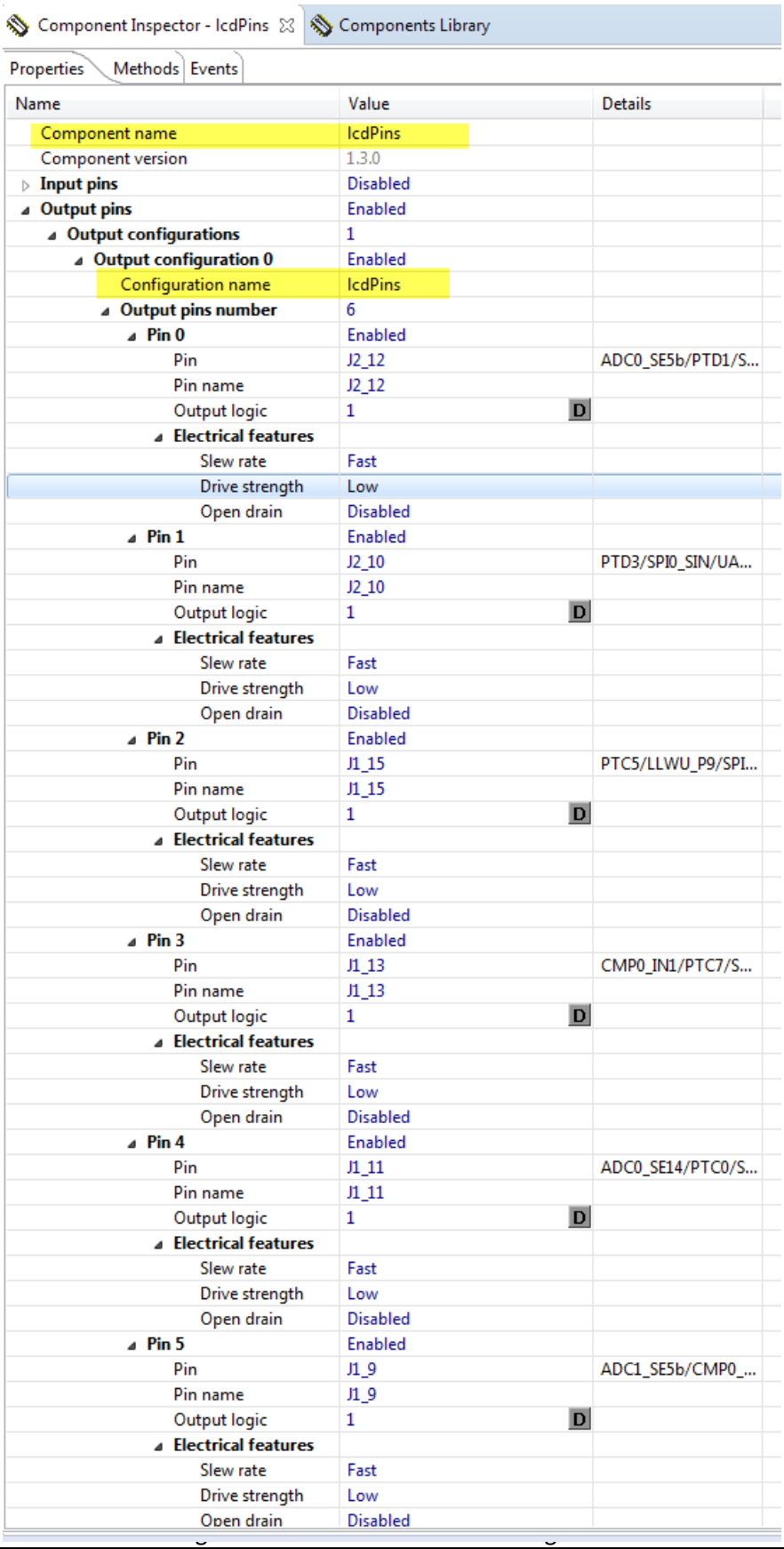


- Now the **fsl_gpio** component needs to be modified, these are the needed changes:

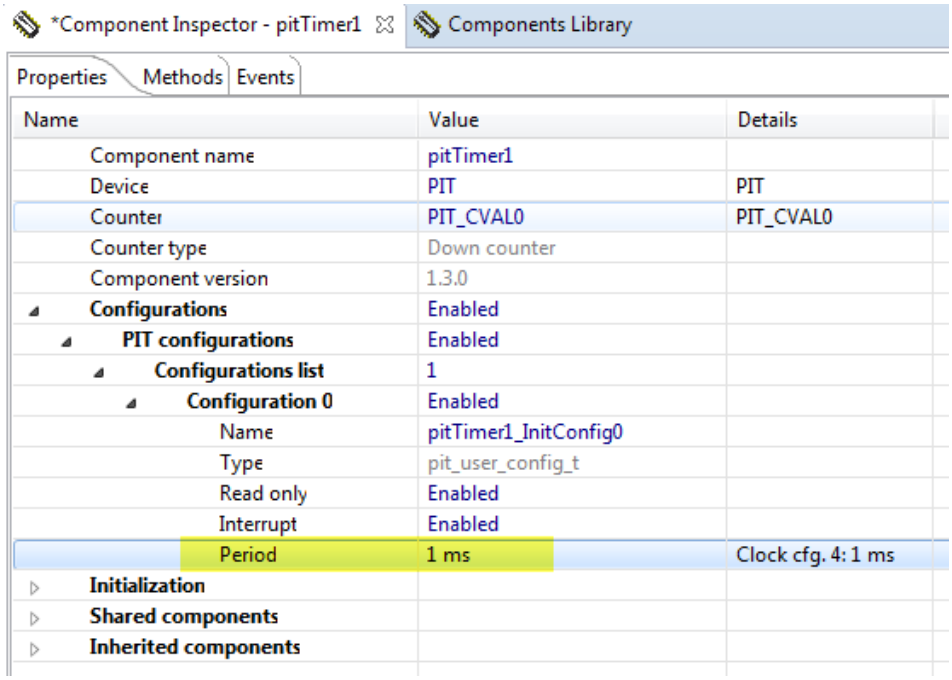
    - Change the component name to **lcdPins**, this is how the LCD_KSDK component will recognize it.
    - Disable the Input pins.
    - Change the "Output configuration 0" name to **lcdPins**.
    - Change the "Output pins number" of the "Output configuration 0" from 1 to 6.
    - Disable the Auto initialization.
    - Assign the 6 pins to be used to control the LCD, here is an example of the pins used in the FRDM-K64F:

Output configuration 0	Function	Pin
Pin 0	Enable	PTD1
Pin 1	RS	PTD3
Pin 2	D7	PTC5
Pin 3	D6	PTC7
Pin 4	D5	PTC0
Pin 5	D4	PTC9

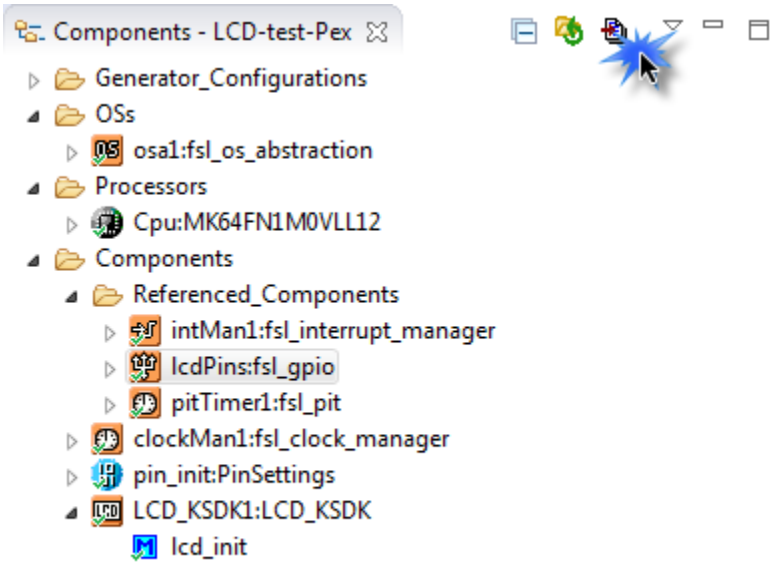
    - Disable the Open drain option for the pins.
    - Change the Slew rate option to fast for the pins.


Component Inspector - lcdPins
Components Library
Properties
Methods
Events
Name
Value
Details
Component name
lcdPins
Component version
1.3.0
Input pins
Disabled
Output pins
Enabled
Output configurations
1
Output configuration 0
Enabled
Configuration name
lcdPins
Output pins number
6
Pin 0
Enabled
Pin
J2_12
ADC0_SE5b/PTD1/S...
Pin name
J2_12
Output logic
1
D
Electrical features
Slew rate
Fast
Drive strength
Low
Open drain
Disabled
Pin 1
Enabled
Pin
J2_10
PTD3/SPI0_SIN/UA...
Pin name
J2_10
Output logic
1
D
Electrical features
Slew rate
Fast
Drive strength
Low
Open drain
Disabled
Pin 2
Enabled
Pin
J1_15
PTC5/LLWU_P9/SPI...
Pin name
J1_15
Output logic
1
D
Electrical features
Slew rate
Fast
Drive strength
Low
Open drain
Disabled
Pin 3
Enabled
Pin
J1_13
CMP0_IN1/PTC7/S...
Pin name
J1_13
Output logic
1
D
Electrical features
Slew rate
Fast
Drive strength
Low
Open drain
Disabled
Pin 4
Enabled
Pin
J1_11
ADC0_SE14/PTC0/S...
Pin name
J1_11
Output logic
1
D
Electrical features
Slew rate
Fast
Drive strength
Low
Open drain
Disabled
Pin 5
Enabled
Pin
J1_9
ADC1_SE5b/CMP0_...
Pin name
J1_9
Output logic
1
D
Electrical features
Slew rate
Fast
Drive strength
Low
Open drain
Disabled

- For the fsl_pit component these are the needed changes:
    - Change the **Period** to 1 ms.



- Now click on the **Generate code** button:



- Add the following function call to the **lcd_init** method on the **main.c** file from the source folder:

```
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/
  /* Write your code here */
  /* For example: for(;;) { } */
  LCD_KSDK1_lcd_init();
  /*** Don't write any code pass this line, or it will be deleted during code
generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T
MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is
defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
```

- Edit the **events.c** file from the source folder as following:

```
/* User includes (#include below this line is not maintained by Processor Expert) */
extern volatile long count;
volatile bool              pitIsrFlag[2] = {false};

#ifdef pitTimer1_IDX
/*
** ===================================================================
**     Interrupt handler : pitTimer1_IRQHandler
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void pitTimer1_IRQHandler(void)
{
  /* Clear interrupt flag.*/
  PIT_HAL_ClearIntFlag(g_pitBase[pitTimer1_IDX], pitTimer1_CHANNEL);
  pitIsrFlag[0] = true;
  count++;
```

- Finally compile and run the code, the following message will be printed on the LCD:

Appendix A - References

- Driving 16x2 LCD using KSDK drivers:
  https://community.freescale.com/docs/

- Tutorial: Creating a Processor Expert Component for an Accelerometer:
  http://mcuoneclipse.com/2013/03/31/tutorial-creating-a-processor-expert-component-for-an-accelerometer/

- KDS webpage:
  www.freescale.com/kds

- KSDK webpage:
  www.freescale.com/ksdk

- Kinetis Design Studio videos:

  - Installation of KDS and Kinetis SDK: https://community.freescale.com/videos/3281
  - Installation of OpenSDA Firmware: https://community.freescale.com/videos/3282
  - Debugging with KDS: https://community.freescale.com/videos/3283
  - Building the KSDK demo applications: https://community.freescale.com/videos/3378