CodeWarrior Tutorial
How to create a new project

1 Environment setup
- You must have CodeWarrior for 68HC08 Release 2.1 Special Edition installed on your computer.

2 Project we are going to create
- We are going to create step by step the demo run in the executive tutorial. The application uses a potentiometer and a LED. The potentiometer represents analog information such as level of a liquid in a tank, a temperature, and the LED gives information like being OFF for low values, ON for high values and blinking for values in the middle. In this example, the thresholds are set at 1/3 and 2/3 of the full range.

3 Creating the project
- Launch CodeWarrior:
  Start > Programs > Metrowerks > CodeWarrior CW08 V2.1 > CodeWarrior IDE
- Select File > New
- Make sure the ‘Project’ tab is active, Select HC08 Stationery
- Enter a project name like ‘MyQT4project’
- Change the directory if you want (Location, Set...)
- Click OK. A new window opens to let you select the device and the language.
- Chose the type, derivative and language:
  Select QT_QY > QT4 > C with Processor Expert
- Click OK.

A new project is created using the template for 68HC908QT1 and development using the Processor Expert tool.

Several windows are visible:
3.1 ‘Target CPU’ window

The ‘Target CPU’ window in the center shows a footprint of the processor selected for the development. In the device, we see the different on-chip modules such as CPU, Timer, A/D converter. Modules with an icon attached to them are modules used by the application. The pins that are used to connect external functions are indicated by a line and an icon, symbol of the function attached (CPU and Port A).

Place the cursor of the mouse on the pins to see a description of their functions.

3.2 ‘Project Panel’ window

The ‘Project Panel’ window shows and keeps track of the beans that have been created for this application. A click on the [+] next to a bean shows a list of methods and/or events related to the bean. A green tick indicate if the named methods or event is selected and a red cross that code has not been generated.

3.3 ‘Bean Selector’ window

The ‘Bean Selector’ window offers the developer a list of beans to add to the project. Some of the beans may not be usable with the version of CodeWarrior installed. The Standard and Professional Editions offer a wider range of hardware and software beans than the Special Edition. In the example, the A/D converter and standard I/O port beans have been expanded. They contain the two beans that will be used for the project: ADfast and BitIO. Unfold the tree to make them visible!
3.4 ‘Project Manager’ window

The ‘Project Manager’ window (top right) is a standard CodeWarrior window showing the files making the current project. Processor Expert adds several folders and files to the project.

Select the ‘Targets’ tab:

Note that several targets have been created, allowing the user to switch from simulation to hardware platform with a simple click.

We stay with the ‘P&E PEDebug’ target at this time.

4 Adding the beans to the project

We are going to add two beans to the project to build our application: one bean for the A/D converter receiving the analog input and one bean for the I/O port driving the LED.

4.1 Creating the A/D bean

- In the Bean Selector window, double-click on the ‘ADfast’ bean once. This opens the ‘Bean Inspector’ window that will be used to set up the parameters for this function. (Ignore the red exclamation mark error message at this time.)

Select ‘Properties’ tab:
- For ‘Interrupt Service/Event’ click the curved arrow to set to Disabled. (If you do not see the extended screen, go to ‘view’ menu and select ‘Expert view’).

- For ‘A/D channel (pin)’ of channel 0 (only one channel selected) click the down-arrow and select PTA5_OSC1_AD3_KBI5

- Click the (…) for ‘conversion time’ and for ‘Init.Value-Requested conv. Time:’ enter 34 (make sure the unit is µS) and click on OK to close the dialog.
- Verify that ‘Low speed’ and ‘Slow speed’ modes are disabled (if not, click on the curved arrow.)
Select the ‘Methods’ tab for the software drivers:

- Click on the curved arrows to ‘do not generate code’ for all of the methods except for ‘Measure’ and ‘GetValue’.

These methods should have the green tick on the left and ‘generate code’ selected.

At this stage, the Target CPU window shows the AD converter activated on pin 2 of the chip.

4.2 Creating the I/O bean

- In the Bean Selector window, double-click on the ‘BitIO’ bean. This loads the ‘Bean Inspector’ window with information about the bean. We can set up the parameters for this function.

Select the ‘Properties’ tab:
- Click the down-arrow for ‘Pin for I/O’ and select AD1_TCH1_PTA1_KBI1
- Click the down-arrow for ‘Pull resistor’ and select No pull resistor
- Click the down-arrow for ‘Open drain’ and select No open drain
- Click the down-arrow for ‘Direction’ and select Output

Select the ‘Methods’ tab for the software drivers:

- Click on the curved arrows to ‘do not generate code’ for all of the methods except ‘PutVal’.

‘PutVal’ method should have the green tick on the left and ‘generate code’.
The ‘Target CPU’ window should now be updated:

- Pin 2 shows the AD converter function (A/D icon) activated.
- Pin 6 the digital output function (up-down arrow) activated.

5 Validation of the design, generation of driver code

We are going to generate the code for the I/O drivers and the files for the user code. We select the ‘Make’ icon in the Project Manager window (or the menu bar Project > Make or [F7]).

Processor Expert shows several messages:
One message indicates that we have started the code generation

The second message shows the progress with the information processed and the code generated.

Another window shows Compiling and Linking progress,

Finally, the CodeWarrior compiler returns a message that we had a successful code generation.
6 Verification of the files created

We can verify the folders created by Processor Expert:

- **‘User Modules’ contains**
  - A file “Events.C” that is the placeholder for the user to enter the code to execute when the events attached with the peripherals take place.
  - A file “MyQT4project.C” that is the placeholder for the main procedure and any other procedure desired by the user. These other procedures can of course be placed in additional files.

- **‘Generated Code’ contains**
  - The .C files for the code associated with the beans added to the project. This includes initialization, input, output and the declarations necessary for the use of the functions.

7 Setting the environment for the simulator

The simulator, by definition, is not connected to any real hardware. However, it simulates instructions to read peripheral registers that will be connected to external signals in the end user application. It is necessary for the simulator to know what value should be read during a read operation. One way to achieve this is to issue a command for the simulator before execution of such instructions. We use a feature of the debugger, the capability to automatically execute commands after loading of the application and before execution of the code.

7.1 Creating ‘postload.cmd’

In the Project Manager window, open the folder ‘Debugger Command files’ and double-click on ‘Postload.cmd’ to open the command file. The file is open for editing.

Below the comments, enter the line:

```plaintext
inputa 00
```

Save and close the file.

Upon execution, this will tell the simulator that reading of Port A should return the value 0x00.
8 Entering the user code

We want the application to read the A/D input, compare the value read against two values (thresholds) and take an action depending on the value. We are going to enter the code for these tasks.

8.1 Opening ‘MyQT4project.c’

In the Project Manager window, double-click on ‘MyQT4project.c’ to open the file that has been created by Processor Expert as a placeholder for the user code. The editor now displays the source code. After several lines of comments and a series of '#include' statements, the main procedure looks like:

```c
void main(void)
{
    /*write your code here*/
    for(;;);
}
```

8.2 Entering the application code

Replace the whole procedure by the following text containing variables definition and new procedure (CodeWarrior will color the key words as they are recognized):

```c
static byte myValues[1]; /* Number of channels */
void main(void) {
    byte min = 255/3;
    byte max = (255*2)/3;
    byte err;
    bool s;
    int i;

    for(;;) {
        /* run measurement with set wait for result */
        err = AD1_Measure(TRUE);

        /* Get results */
        err = AD1_GetValue((byte *)myValues);

        /* Compare value and action LED */
        if (myValues[0] > max) {
            s = 1;    /* set LED ON */
        } else {
            if (myValues[0] < min) {
                s = 0;    /* set LED OFF */
            } else {
                s = s^1;    /* invert LED ON <-> OFF */
            }
        }
        for (i = 0; i < 5000; i++);  /* delay */
        Bit1_PutVal(s);
    }
}
```
9 Compiling

When the source code is entered, save and close the ‘MyQT4project.C’ file.

We can verify that everything is correct by clicking the icon ‘Make’ in the project manager (or menu Project > Make or [F7].)
In normal operation, calling the debugger will force a ‘Make’ command if the ‘Build’ panel is set for this and if a file has been modified.

A message from CodeWarrior compiler/linker will indicate that we have a successful build. We can now use the simulator to run the code and verify our implementation.

10 Loading the Debugger and Simulator

CodeWarrior provides debugging capability through a powerful debugger. There are several standard windows open automatically when opening the debugger. The opening window can be entirely customized to display the information needed by the user. This setup will be remembered by the debugger and reopened each time the debugger is invoked. These settings can also be different from one project to another, and even from one target to another, thanks to the flexible project settings available in the IDE.

Invoke the Debugger by clicking on the debugger icon in the Project Manager window.

This will load the debugger, the simulator (set by default in the stationery) and then load the object code of the application. Once opened, the debugger display several windows:

10.1 ‘Source’ window

The ‘Source’ window shows the code and allows to set, delete or disable breakpoints. From this window we can find and go to other procedures.

There is a close relation with the assembly window: instructions can be dragged from one window to the other window and corresponding instructions are highlighted. This is useful when trying to relate optimized code to the original lines of C code.
10.2 ‘Assembly’ window

The ‘Assembly’ window shows addresses, code and dis-assembled instructions. Information such as calculated addresses and hexadecimal code is also displayed. Variables and pointers can be displayed in ‘symbolic’ format.

Breakpoints can be set in this window. There is a close relation with the source window: instructions can be dragged from one window to the other window and corresponding instructions are highlighted.

10.3 ‘Register’ window

The ‘Register’ window shows the register set for the 68HC08 family and their contents. Last changes are highlighted in red. Display can be done in all usual formats such as binary, decimal, hexadecimal and a few others. Register contents can be edited by typing the value in the corresponding box.

10.4 ‘Memory’ window

The ‘Memory’ window is used to display bytes, words or long words of memory in many formats such as decimal, hexadecimal or binary. Changes from the last refresh are indicated in red. Memory can be edited by typing the values either in the byte area or the ASCII area. Memory can also be filled, saved to and reloaded from disk.

It is possible to open multiple instances of the memory window to display non-contiguous areas and make the best use of the screen space.

10.5 ‘Data’ windows

The ‘Data’ windows are used to display the variables used by the application. Representation of the values can be done in several different formats. Multiple copies of the data window can be created to display global and local variables.
11 Adding Stimulation and Data Visualization

In order to facilitate the testing of the code, CodeWarrior offers the capability to stimulate the built-in analog and digital inputs, as well as to display in different forms, the status of the outputs.

11.1 I/O Stimulation

For this example we are going to stimulate the analog input. This will simulate the action of the potentiometer in the application (information from a sensor in the final application).

From the Debugger menu, select: Component > Open
A window opens with a list of components,
- select `Visualizationtool`
- click OK

A window opens in ‘Edit’ mode. Resize it.
Right-click in the window and select: Add New Instrument > Bar

The cursor is now dragging a box that we drop in the Data Visualization window (click) and we resize the box as a thin vertical box.

A right click in this box opens a window to set the properties of the box. Set the following parameters:
- Rounding box: Sunken
- Kind of Port: Memory
- Port to display: 0x3E
- Bar direction: Down-Up

Close the property box.
Leave the ‘edit’ mode (press the switch icon or right click in the window, then deselect Edit Mode.)
The box will be used as a sliding potentiometer.

Let’s verify the good stimulation of the I/O:
In the ‘Memory’ window,
- Right-click and select Mode > Periodical and enter the value 3.
(This will automatically refresh the window to show changes in the values.)
- Click Ok to close the window.

Drag up and down the red column of the data stimulation (if none is visible, click in the box and a red column will appear at the cursor) and observe memory location 0x3E in the Memory window. Its contents should appear in red and change according to the displacement.
At this stage we have implemented a very powerful feature of CodeWarrior: the ability to stimulate an analog input to a device and to stimulate real-time.

12 Adding Data Visualization

If stimulating the inputs to a device is an important feature, it is equally important to be able to visualize the actions that the processor is programmed to take during the execution of the code for the application. We are going to simulate the LED that we defined at the beginning of the document.

- Right-click in the Data Visualization window and select ‘edit’ mode,
- Right-click and select **Add New Instrument > LED**

The cursor is now dragging a box that we drop in the data visualization window (click) and we resize the box to a visible LED.

A right click in this box opens a window to set the properties of the box. Set the following parameters:

- **Rounding box:** No Box
- **Kind of Port:** Memory
- **Port to display:** 0x00
- **Bit number to display:** 1
- **Color if Bit==0:** (grey color)
- **Color if Bit==0:** (yellow color)

Close the property box

The LED is now connected to Port A bit 1 (address 0x00).

We need the LED to refresh periodically to indicate changes when they happen. Right-click on the Data Visualization window and select **Properties**.

For Refresh Mode, select **Periodical** and enter a **Refresh Rate of 1**. Close the window.

Leave the ‘edit’ mode (press the switch icon or right click in the window)

We now have a complete simulation of the features that will be implemented in the end application. The potentiometer and the LED are connected to the same pins as the final application and the polarities of the signal can be set to match the final application.

At this stage we have implemented another very powerful feature of Code Warrior: the ability to display the output of a peripheral and reverse the function to stimulate the pin if needed. This provides a much more user-friendly display of the activity of the processor under test.
12.1 Free running Stimulation and Visualization

The final step is to see if we can test the application in its full operation:
- LED ON when potentiometer is in above 2/3 of the range,
- LED blinking when the potentiometer is between 1/3 and 2/3 of the range,
- LED OFF when the potentiometer is below 1/3 of the range.

We also want to have the LED blink without having to keep stepping through the code.

Run the program and drag the column simulating the potentiometer to different positions (if no red part is visible, click anywhere in the column, the red portion will appear to the cursor):
- In the range 0 to 1/3, the LED should be steadily OFF (grey),
- Between 1/3 and 2/3, the LED should blink, even when not moving the cursor,
- Above 2/3, the LED should be steadily ON (yellow).

Note: the blinking rate does not exhibit a regular 50% ratio. This is not a defect but due to the asynchronous refresh of the data visualization with regard to the program. This causes the reading of the memory to not coincide with the occurrences of ON and OFF.

12.2 Halting and closing the program

- Click on ‘Stop’
- Save your settings, including Data Visualization (File > Save Configuration)
- Close the Debugger
- Close CodeWarrior

CONGRATULATION, you have completed the design of an application using many unmatched features of CodeWarrior: the Processor Expert, the CodeWarrior IDE, the Debugger/Simulator with I/O Stimulation and Data Visualization.

We have implemented very powerful features of CodeWarrior:
- Stimulation of an input to a device,
- Stimulation of an analog input,
- Visualization of a digital output,
- Real-time stimulation and visualization from an operator point of view.

This saves a lot of disk space, comparing with the alternative of creating large numbers of files with various data patterns, and provides the flexibility to test in a more realistic way any situation the user may chose to evaluate.