AN667

GETTING STARTED WITH THE SILICON LABS PRECISION32™ IDE

1. Introduction

This application note describes the Precision32™ Integrated Development Environment (IDE) for the Silicon Labs 32-bit microcontrollers (SiM3xxxx). This IDE is based on the Eclipse platform.

2. Key Points

- Importing and building projects with the Precision32 IDE
- IDE Overview
- Debugging with the Precision32 IDE
- Getting Started: Blinky Example

3. Relevant Documentation

- Precision32 Help: In the Precision32 IDE, go to **Help→Help Contents**
- Eclipse documentation: http://help.eclipse.org/indigo/index.jsp

4. Hardware Requirements

The Precision32 IDE has the following hardware requirements:

- A standard x86 PC with 1 GB RAM minimum (2 GB recommended) and 500 MB+ of available disk space, running one of the following operating systems (both 32-bit and 64-bit systems are supported):
  - Microsoft® Windows XP (SP2 or greater)
  - Microsoft® Windows Vista
  - Microsoft® Windows 7

An internet connection is required to request and activate license keys.

5. Workspaces

A workspace in the Precision32 IDE is a grouping of active projects. The workspace contains the top-level IDE settings, including the global defaults, view window positions, and the projects in the workspace. The workspace information is contained in the metadata (.metadata) subdirectory in the workspace directory. Any projects added to the workspace will be copied to this location if the copy projects into workspace option is selected.

After launching Precision32, the workspace dialog shown in Figure 1 prompts for a workspace location. To select a workspace, either browse to an existing workplace or select a new workspace location.

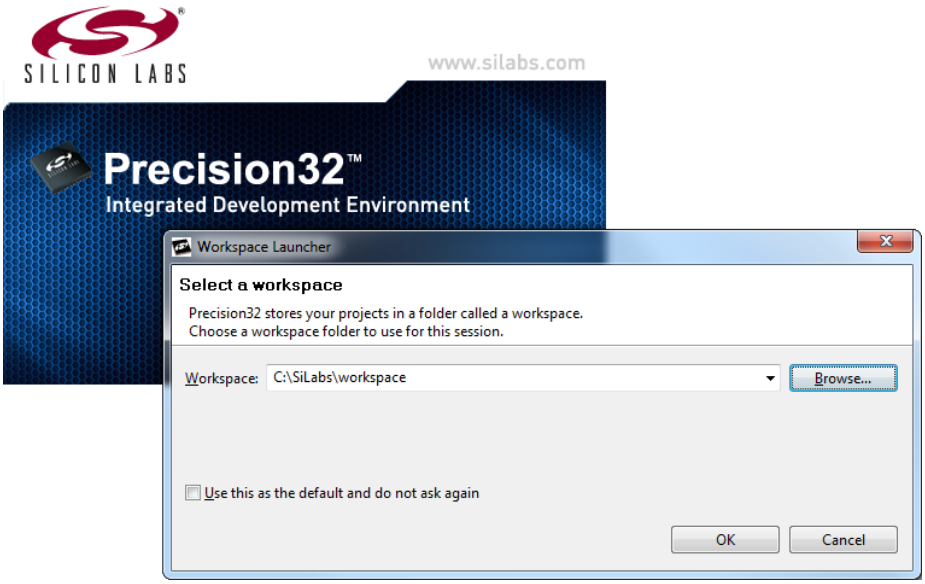

Figure 1. Selecting a Workspace

To change the workspace while the IDE is open, go to **File→Switch Workspace**.

6. Activating Precision32 IDE

Prior to registering the Precision32 IDE, the size of the debug image is limited to 8 kB and Trace functionality is disabled.

To register the Precision32 IDE, follow the instructions on the initial landing page after opening the software. This process will require an internet connection.

1. In the IDE, go to **Help→Product Activation→Create Serial Number and Activate**.
2. Write down the displayed serial number or copy it to the clipboard.
3. Press OK and a web browser will open on the Activations page. A mysilabs account is required to enter the serial number in the page.
4. Complete the rest of the form and press the button to request the activation code.
5. Copy the activation code.
6. Navigate to **Help→Product Activation→Enter Activation Code**.
7. Enter the activation code and press OK.

Once activated, the landing page will update and all features are available.

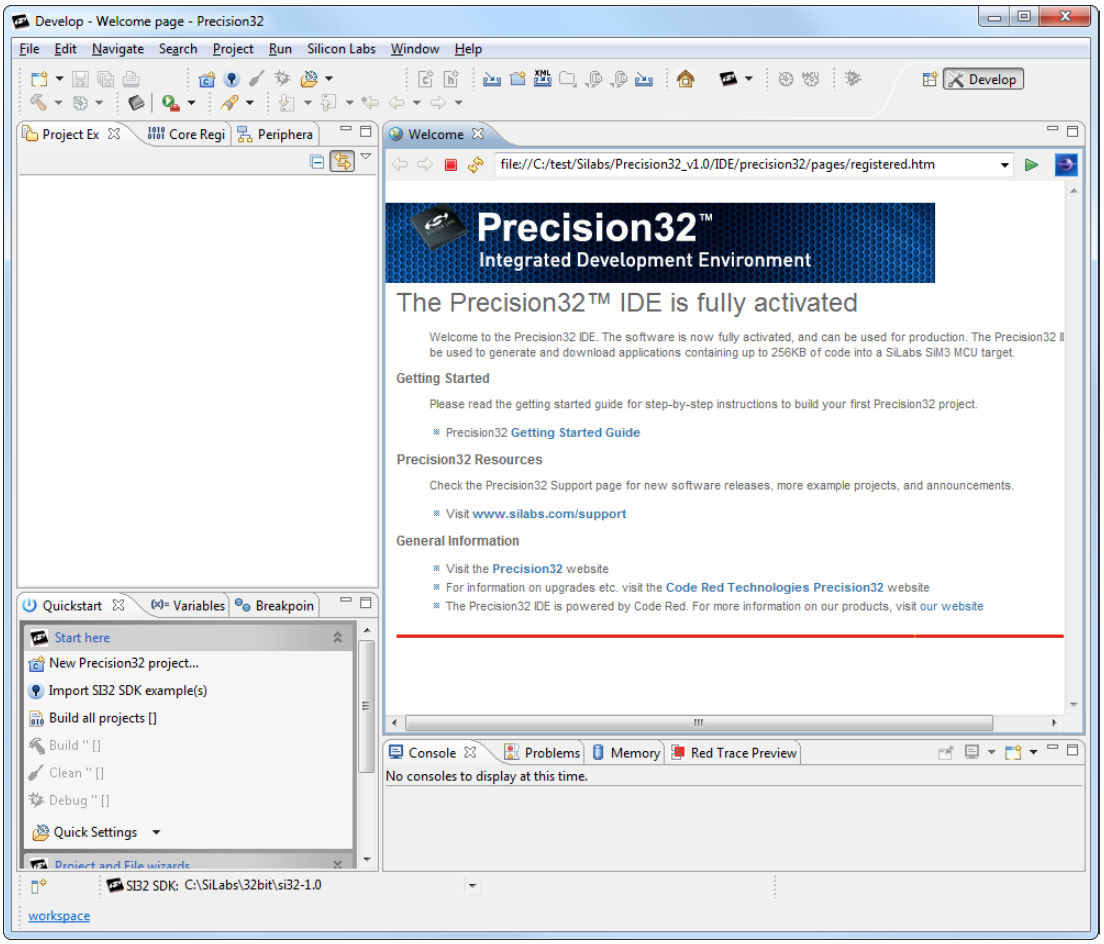

Figure 2. Registered Landing Page

7. Projects

A Precision32 project contains files, build options, and project settings. Projects generally exist as a directory containing sub-directories and files. The project structure seen in the IDE is replicated physically on the disk. However, a project may also contain linked files or directories which are just pointers to files or folders outside of the project directory.

7.1. Project Structure

The projects generated by the Silicon Labs AppBuilder and the SDK follow a recommended project structure. These projects have three categories of source files:

1. The my* files and any user generated files are located in a **src** subdirectory under the project directory. AppBuilder will generate my* files if they do not exist, but will not modify existing copies. This ensures that user updates to my* files are safe when re-exporting a project from AppBuilder.
2. Generated (g*) files are located in **src\generated**. These files contain all the automatically generated AppBuilder code. These files will always be overwritten when a project is re-exported, and any user changes to these files will be lost.
3. The si32 SDK firmware package consisting of the HAL, code examples, and si32Library is located in a linked folder at **C:\SiLabs\32bit\si32-x.y** by default, where x is the major version and y is the minor version.

Each project is compiled against a version of the HAL or si32Library.

7.1.1. Setting the Silicon Labs SDK Path

The Silicon Labs HAL revision is selected using the drop-down menu in the Precision32 IDE footer shown in Figure 3.

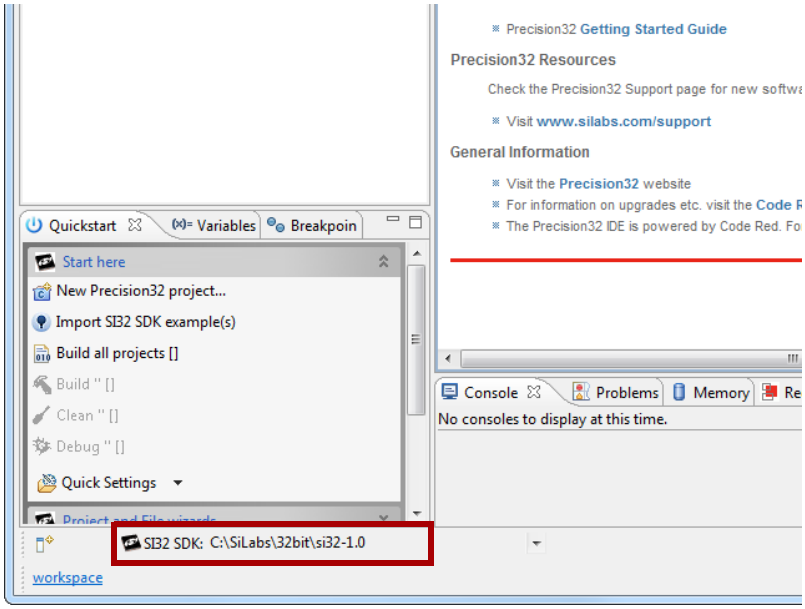

Figure 3. Selecting the Silicon Labs SDK Path

A warning is displayed if the IDE cannot find the previously-selected path.

For projects that do not use the Silicon Labs SDK, this drop-down menu can be toggled on or off by clicking on **Silicon Labs→Show SDK Selector**.

7.2. Creating a New Project

To create a new code project, launch the Silicon Labs AppBuilder software by selecting **New Precision32 project...** in the **Quickstart** view to generate the device initialization code for the 32-bit device. Then, checking the **Open After Export** option in AppBuilder will automatically open the exported AppBuilder workspace and project in the IDE.

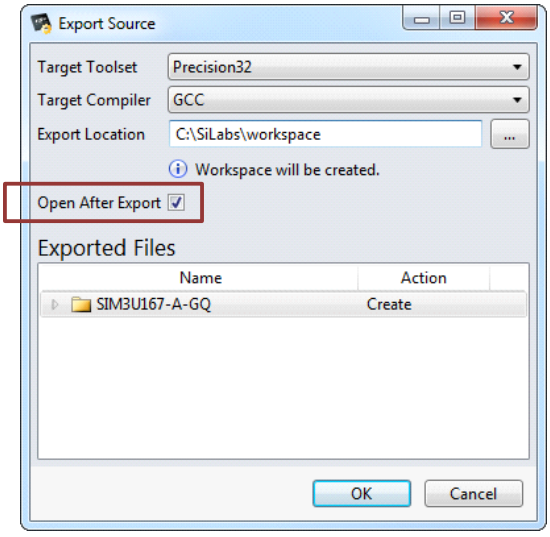

Figure 4. Automatically Opening a New Workspace

7.3. Importing a Project

To import an existing project such as a Precision32 code example:

1. Select the si32 SDK path as described in “7.1.1. Setting the Silicon Labs SDK Path”.
2. Click the **Import SI32 SDK example(s)** link in the **Quickstart** view. If the SDK path is appropriately set, all available projects in the SDK will automatically populate the Projects area.
3. (Optional) Press the **Browse...** button to select a device-specific set of examples.
4. Select the checkboxes for the projects to import.
5. Ensure the **Copy projects into workspace** checkbox is checked.
6. Press the **Finish** button.

Figure 5 shows the process of importing a project.

When an existing project is imported into a workspace, it will be copied into the workspace directory if the **Copy projects into workspace** option is selected. After the import, all changes to the copied files are local and specific to that workspace, which ensures the master copy of the project is unchanged. If a project is added to a workspace without making a copy, any modifications to the workspace modify the original project. Any modifications to the linked SDK files will modify the master copy.

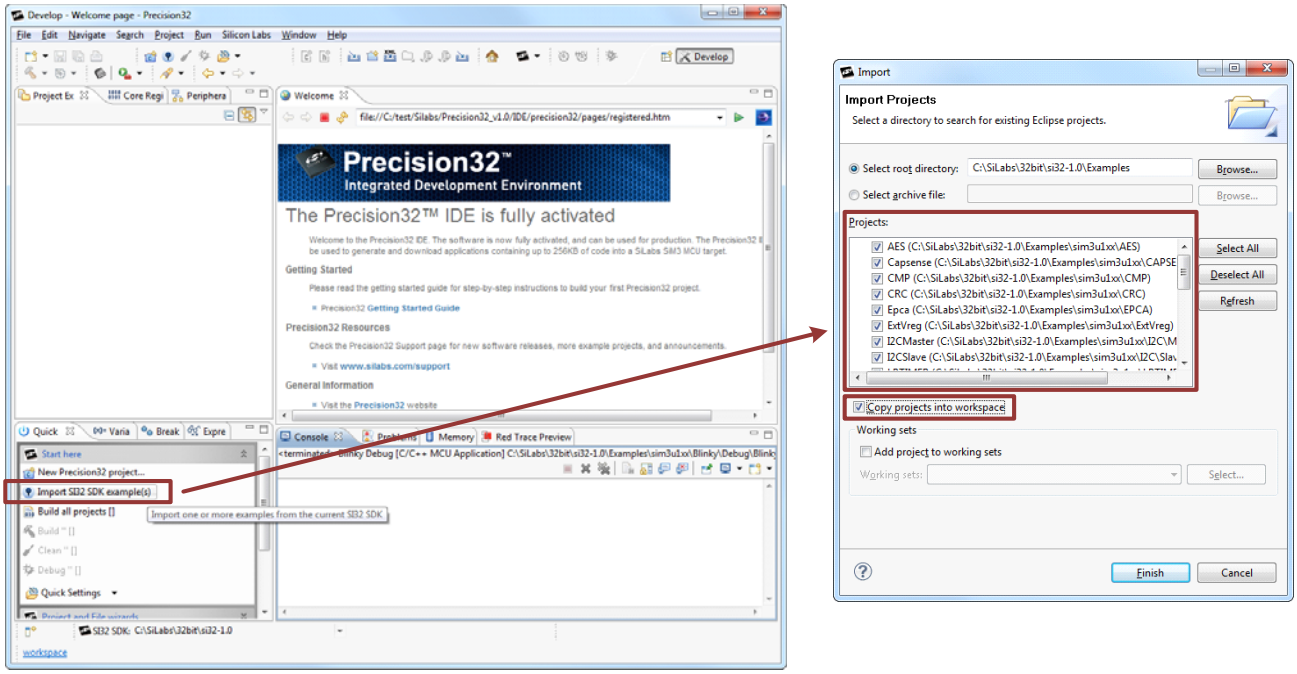

Figure 5. Importing a Project

7.4. Building a Project

Once a project is open and selected in the workspace, the **Quickstart** view will update with a **Build 'project' [Debug]** option shown in Figure 6. Selecting this option will compile all the files associated with the project. The output of the build will appear in the **Console** view.

Section "8. Build Options" discusses the full build options and configurations available.

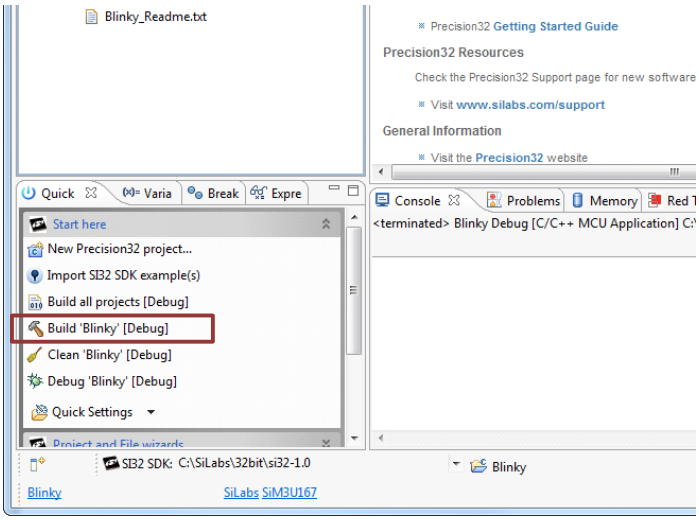

Figure 6. Building a Project

7.5. Refreshing a Project

The IDE will not immediately reflect files added to a project outside of the IDE. To refresh a project and reload the references from the disk, right-click on the **Project Explorer** view and select **Refresh** as shown in Figure 7.

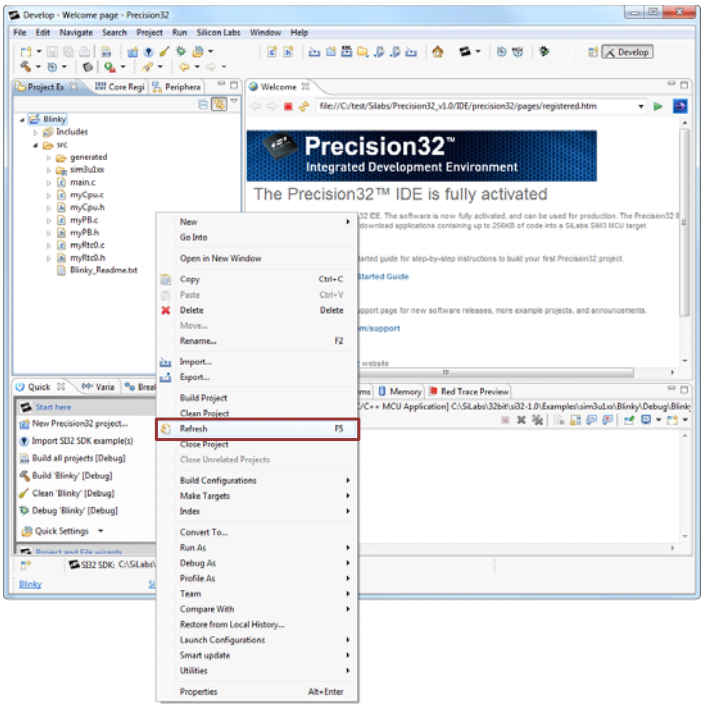

Figure 7. Refreshing a Project

7.6. Cleaning a Project

To clean the project build files from a workspace directory, select the **Clean 'project' [Debug]** option in the **Quickstart** view. Figure 8 shows the **Clean** menu option.

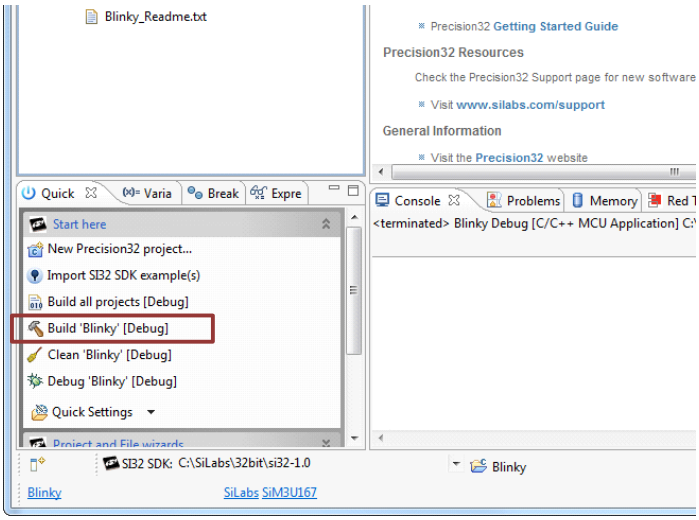

Figure 8. Cleaning a Project

7.7. Removing a Project from a Workspace

The project can either be removed from the workspace or physically deleted from the disk. To remove a project from a workspace, right-click on the project name in the **Project Explorer** view and select **Delete**. A dialog will open with a **Delete project contents on disk (cannot be undone)** option that will physically delete the project files, if selected.

8. Build Options

Each project stores the project build settings. These settings can be accessed through the **Project→Properties** menu or by right clicking in the **Project Explorer** view and selecting **Properties**.

8.1. Settings

The **C/C++ Build→Settings** section contains most of the build settings for the project. The **View build options for 'project'** option in the **Build and Settings** section of the **Quickstart** view is another way to open this same build settings view.

The **C/C++ Build→Settings→Tool Settings** view has many options for optimization levels, display warnings, and other build settings.

The **C/C++ Build→Settings→Build Steps** view has options for pre- and post-build steps.

These settings will not need to change for most applications.

8.2. Paths and Symbols

The **C/C++ General→Paths and Symbols** section has settings for the project include paths for each configuration and language combination. When adding a path, select the **Add to all configurations** and **Add to all languages** options unless the paths should be omitted from configurations or languages. This settings view also contains libraries and library path information.

8.3. Build Configurations

A build configuration is a named collection of build options. The Precision32 has two build configurations by default: Debug and Release. The Debug build configuration has no optimization and includes all debug symbols in the build. The Release build option has high optimization and includes no debug symbols in the build.

9. IDE Layout Overview

The default Precision32 IDE has four main areas: Project Explorer, Code Editor, Quickstart, and Status. These view windows and areas are completely configurable.

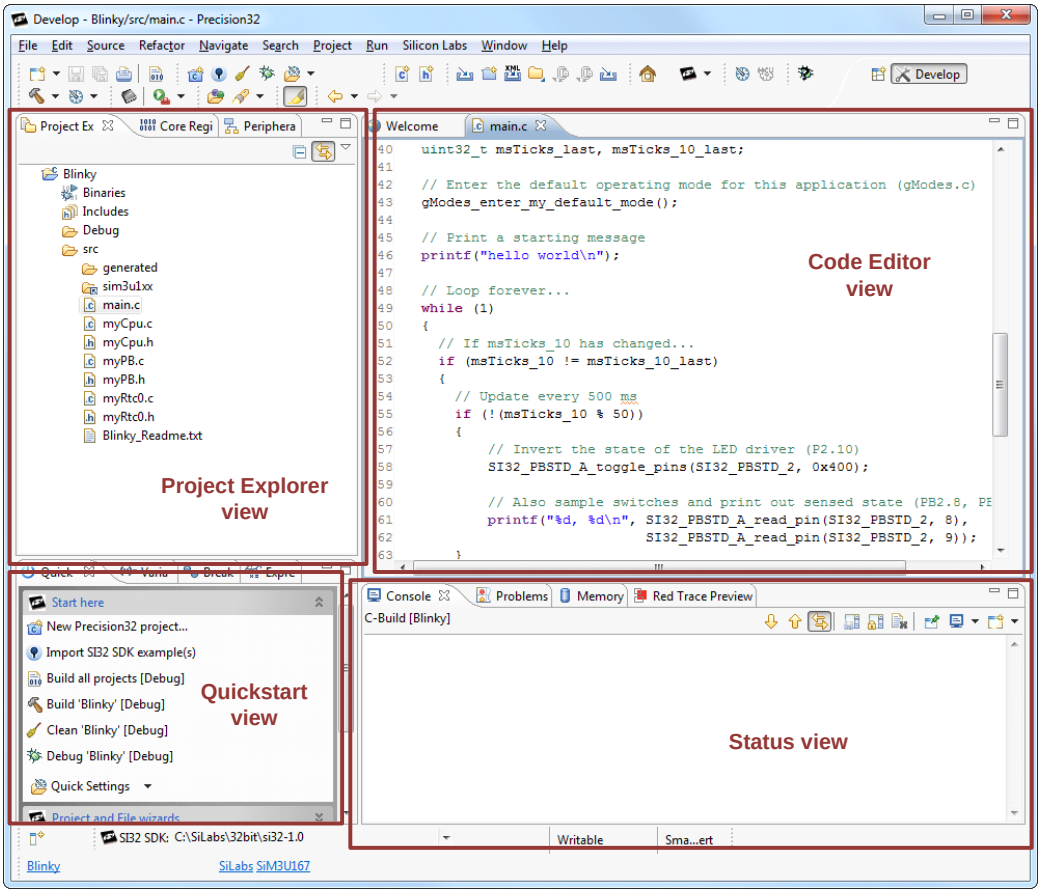

Figure 9. Precision32 IDE Layout Overview

9.1. Project Explorer

The **Project Explorer** view displays the projects associated with the active workspace. Each project has a set of subdirectories, though some of these subdirectories may not always be present:

- After a successful build, **Binaries** includes the binary files associated with the project.
- **Includes** contains the external include files outside the project structure.
- The **Debug** folder contains the build information, include make files, object files, and other build outputs.
- The **src** folder contains all of the project source files.

The **Project Explorer** also displays several icons to provide extra information about the project. A small red x on a folder indicates that the file didn't build correctly. In addition, icons indicate when linked resources cannot be found.

Double clicking any files in the **Project Explorer** will open that file in the **Code Editor** view.

Each file can also be expanded to show a list of includes, defines, and functions used by that file. Double clicking one of these will jump to the corresponding line in the file.

9.2. Quickstart

The **Quickstart** view is the primary portal to all Precision32 development tasks. This view will be extended in the future to include more features and tasks.

Most of the functions available in the **Quickstart** view are available in other locations in the IDE. However, these alternate methods sometimes behave differently than the **Quickstart** link. The **Quickstart** view is the quickest and easiest way to navigate the development tasks in the IDE.

9.3. Code Editor

The **Code Editor** view has many customizable options. This section discusses the default behavior in the Precision32 IDE.

Hovering over a symbol in the **Code Editor** view will open a type-specific viewer. For variables, this pop-up viewer displays the declaration of the variable. The viewer displays an expanded version of macros. Finally, hovering over a function will display the first few lines of the function.

Right-clicking on a symbol provides a long list of options. For example, selecting **Source→Correct Indentation** will adjust the indentation of the line, and **Source→Toggle Comment** will toggle between commenting or uncommenting the line. The **Declarations**, **References**, **Search Text** options will do various searches and open a **Search** view in the **Status** view.

The panel to the right of the scroll bar displays tags as shown in Figure 10. A red tag here indicates a build error, while a yellow tag indicates a warning. A blue tag indicates an informational message. Hovering over the tag with the mouse cursor provides more information about the problem.

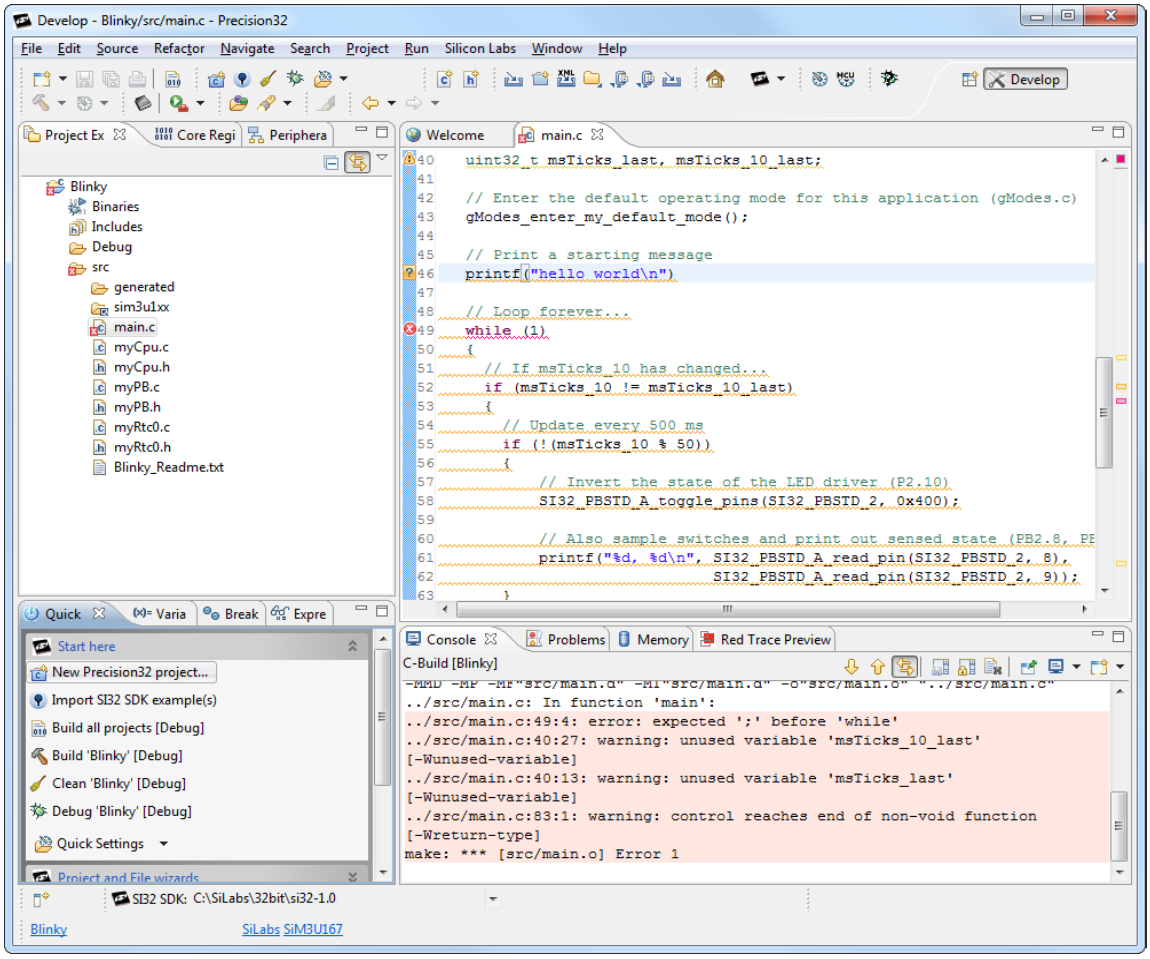

Figure 10. Code Editor Tags

9.4. Status

The **Status** area contains status information for the project. The **Console** view contains context-sensitive information. For example, after clicking the **Build 'project' [Debug]** link in the **Quickstart** view, the **Console** view displays the build feedback; during a debugging session, the **Console** view can display debugging printf() information.

Clicking on a project or other element may change or restrict the information displayed in the **Console** view. The **Display Selected Console** button on the right side of the **Console** view shown in Figure 11 can be used to manually switch between consoles.

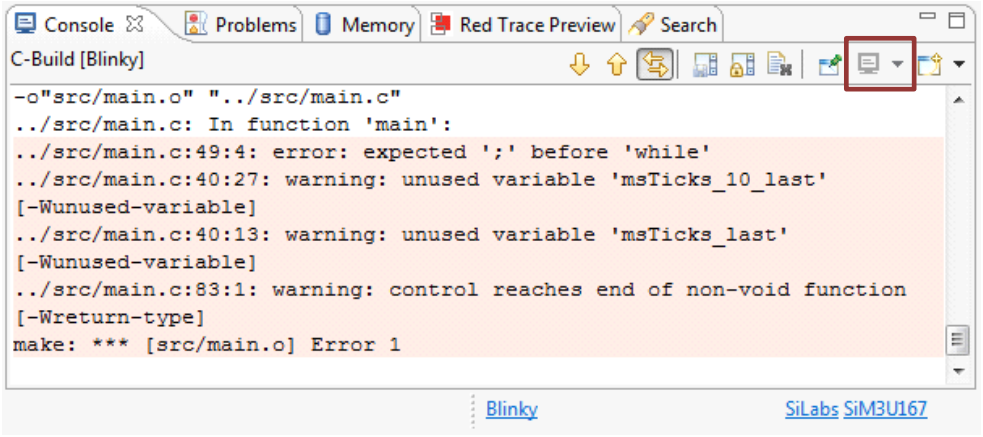

Figure 11. Display Selected Console Button

Any build errors in the **Console** view will be highlighted in red. Double clicking on an error will jump to the error in code.

The **Console** view for each process (building or debugging) is reset upon the reinitialization of that process. For example, the console with build errors for a project will reset when another build of the same project starts.

9.4.1. Problems View

The **Problems** view in the **Status** view aggregates all errors and warnings encountered during a build and displays them in a hierarchical fashion. This saves the need to search through the build log to find a list of the errors in a multiple-file project. Double-clicking on these entries will jump to the error or warning in the corresponding file. Figure 12 shows the **Problems** view.

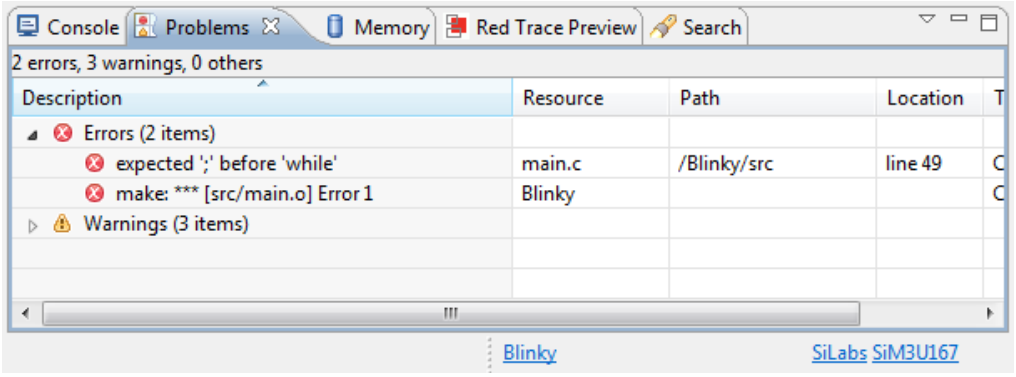

Figure 12. Console Problems View

9.4.2. Other Views

The Precision32 IDE has many options for additional tabs in the **Status** view area. These views can be added by using **Window→Show View→Other**.

9.5. Perspectives

A perspective in the Precision32 IDE is a collection of views and their placement. The current perspective is highlighted in the upper-right corner of the IDE view as shown in Figure 13. Additional perspectives can be added.

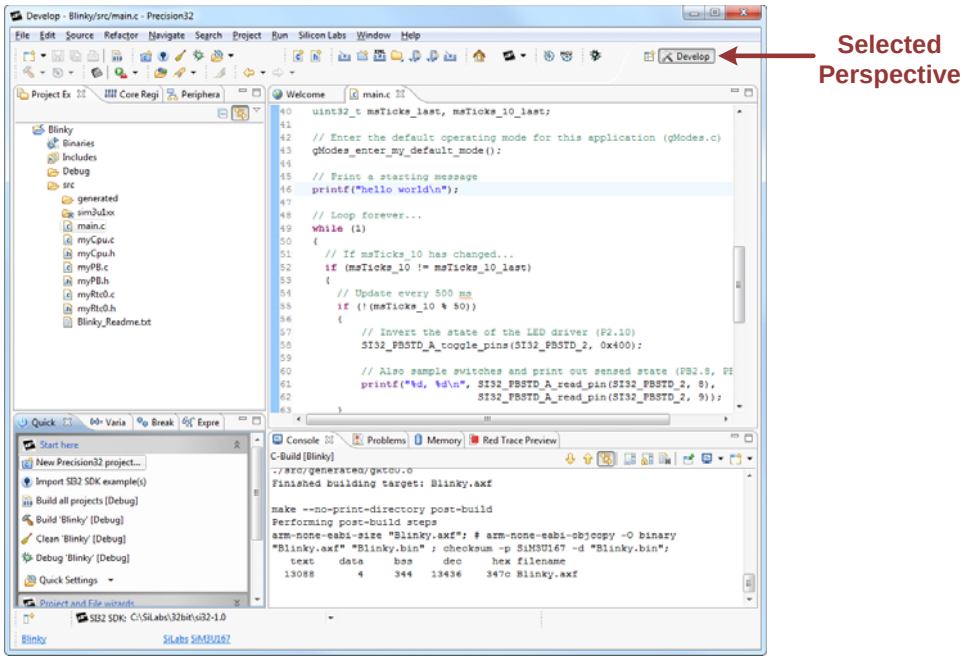

Figure 13. IDE Perspectives

10. Debugging

The **Debug 'project' [Debug]** link in the **Quickstart** view starts a debugging session. Starting a debug session will automatically recompile the project (if required), connect to and Flash the device, and run to first line of main(). The IDE will automatically open a **Debug** view if is not open when a debugging session begins.

To restart a debug session, first exit debug mode to free the debug adapter by clicking the red square **Terminate** button. Then, press the **Debug 'project' [Debug]** link in the **Quickstart** view.

When debugging, hovering over a symbol will have context-sensitive information. For example, hovering over a function in debug mode shows the address of the function instead of the first lines of code.

Figure 14 shows the default debug configuration of the IDE.

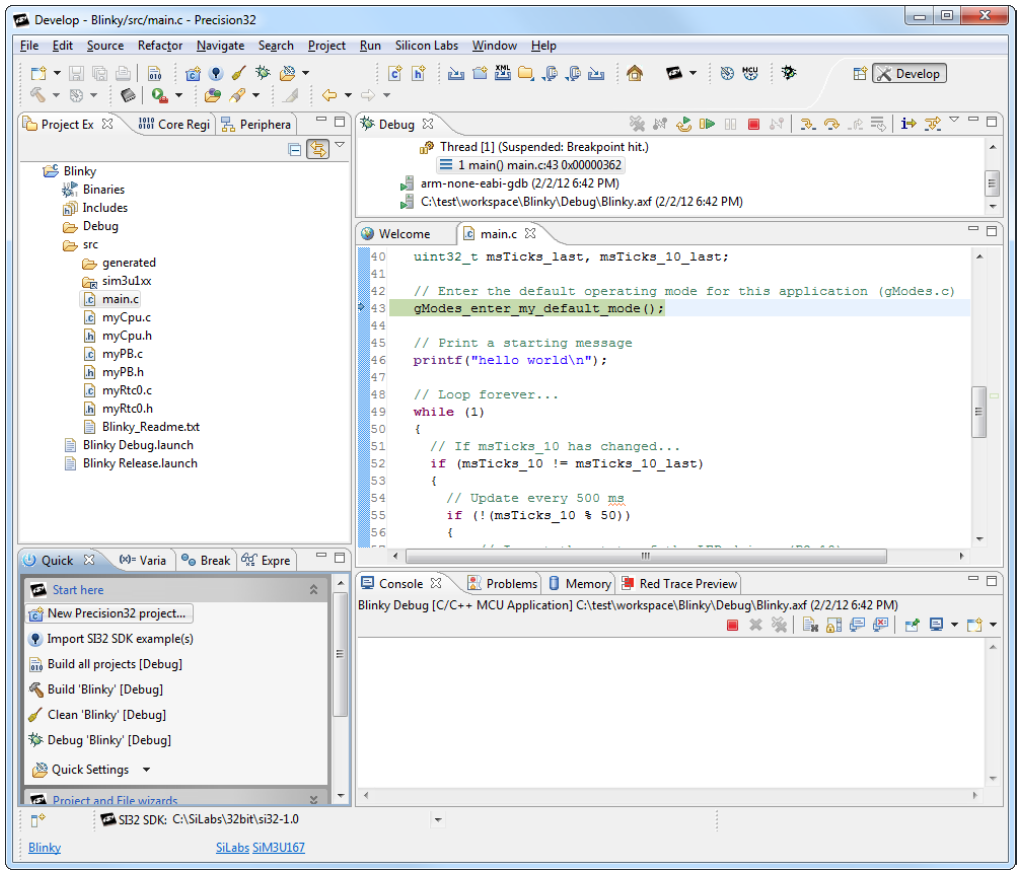

Figure 14. Debugging in the Precision32 IDE

10.1. The printf() Function

The redlib semi-hosting library is used by default to route printf() statements to the **Console** view. Projects compiled using the semi-hosting library will hang when printf() statements are encountered and the device is not connected to the IDE. Projects built with the no-host library will work properly without the IDE, but will not have printf statements directed to the **Console** view.

To switch the library used when building the project:

- Open the **myLinkerOptions_p32.ld** file located in the project directory in any text editor.
- To enable debugging printf(), uncomment line 6 of the file (**GROUP(libcr_semihost.a libcr_c.a libcr_eabihelpers.a)**) and comment out line 7 of the file.
- To enable normal printf(), uncomment line 7 of the file (**GROUP(libcr_nohost.a libcr_c.a libcr_eabihelpers.a)**) and comment out line 6 of the file.

Figure 15 shows the printf() options in the linker file.

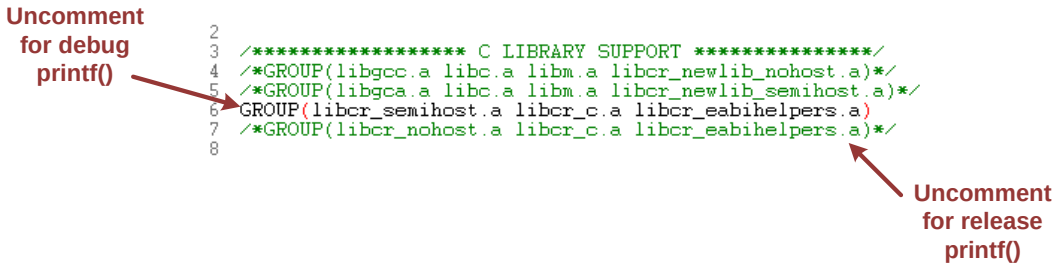

Figure 15. Switching Printf() Options

10.2. Debug View

The **Debug** view shown in Figure 16 has debugging control buttons and the stack trace view. The buttons at the top of the view can run (**Resume**), stop (**Pause**), and reset (**Restart**) the device. The **Terminate** button exits debug mode.

When halted at a breakpoint, the **Debug** view has options to step into, over, and out of functions.

The code view is lined with the stack trace, and highlighting a stack call will jump to that call in code. The stack viewer shows the device process stack (GDB Debugger) and entries for the host GDB and debug process.

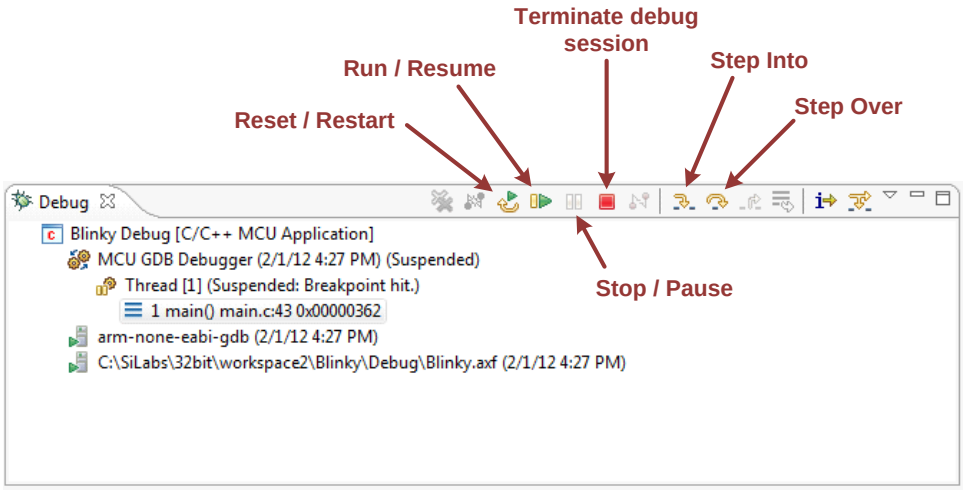

Figure 16. Debug View

10.3. Breakpoints

Add a breakpoint to a project by double-clicking on a blue area to the left of the **Code Editor** view. Lines that are not marked with blue are ineligible for breakpoints.

The **Breakpoint** view in Figure 17 shows the current breakpoints and has options to enable or disable breakpoints without removing them. Double-clicking on breakpoint jumps to the related code in the **Code Editor** view.

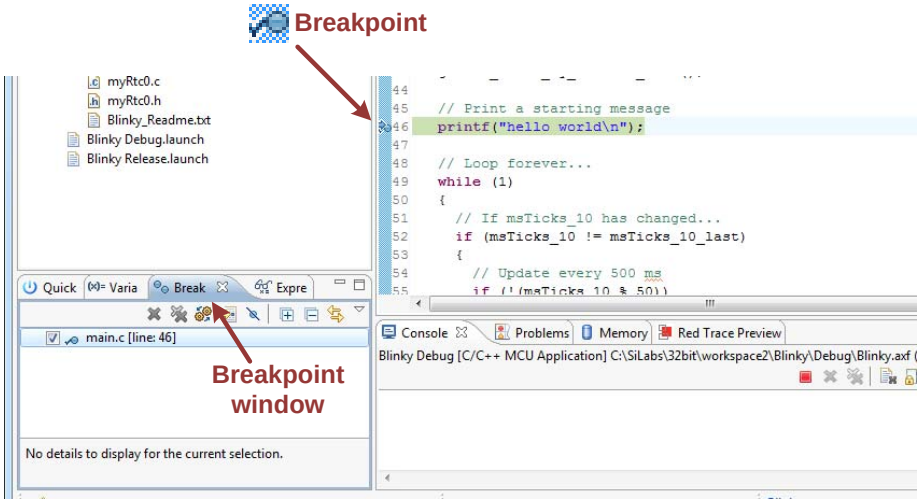

Figure 17. Breakpoint View

When stopped at a breakpoint, right-clicking on the **Code Editor** enables options to run to a particular line of code or add variables to the **Expressions** watch view.

10.4. Peripherals

The **Peripherals** view contains a list of peripherals for the device connected to the IDE. Selecting the checkbox next to a peripheral adds the peripheral to the **Memory** view. If the **Memory** view is not visible when a peripheral is added, the IDE focus will switch to the **Memory** view.

Figure 18 displays the **Peripherals** view.

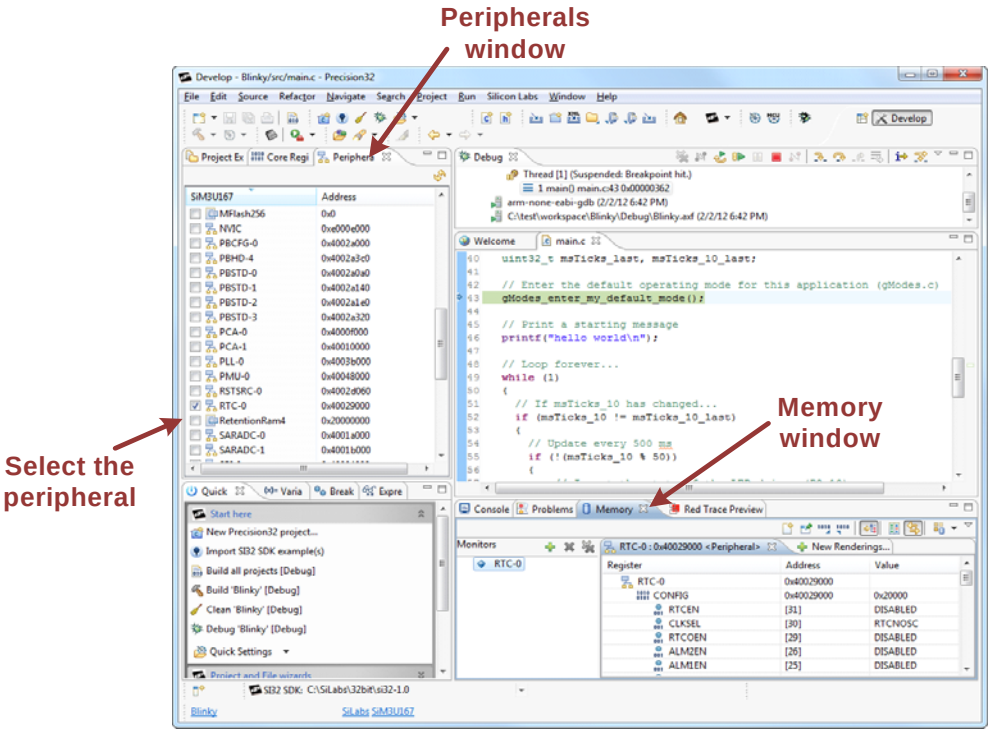

Figure 18. Peripherals View

10.5. Memory

The **Memory** view performs all memory viewing, which includes the device peripherals. For peripherals, the **Memory** view has a **Registers→Bit Fields** hierarchy. Hovering the mouse cursor over a register or bit field causes a verbose tool tip to open. This view allows the state of the device to be changed while stopped at a breakpoint.

Non-peripheral memory views are added to the **Memory** view by selecting the green icon.

Figure 19 shows the **Memory** view.

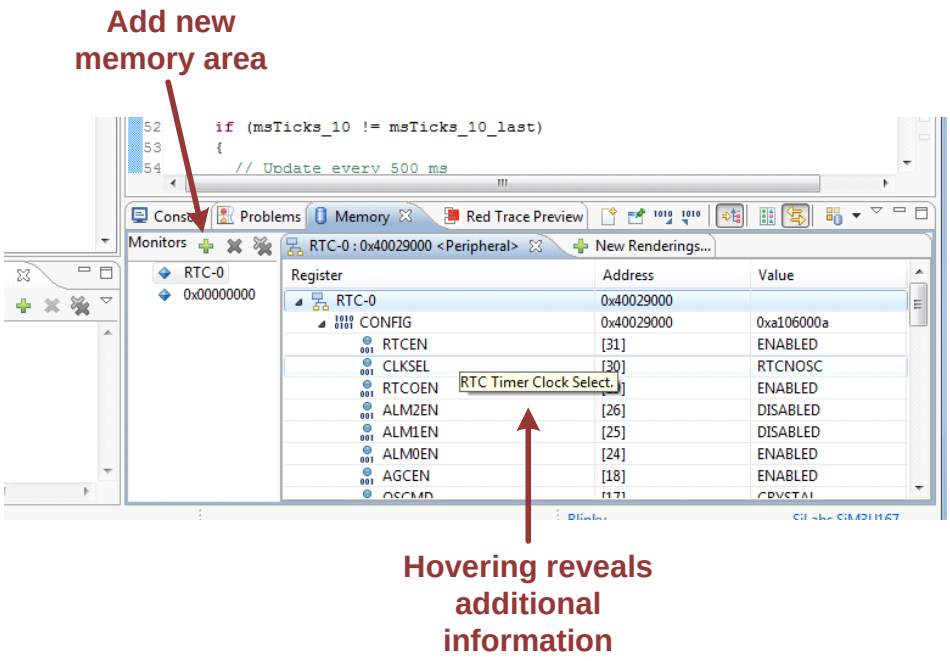

Figure 19. Memory View

10.6. Core Registers

The **Core Registers** view shown in Figure 20 displays the core registers of the device. Some core peripherals such as the NVIC are included with the peripherals.

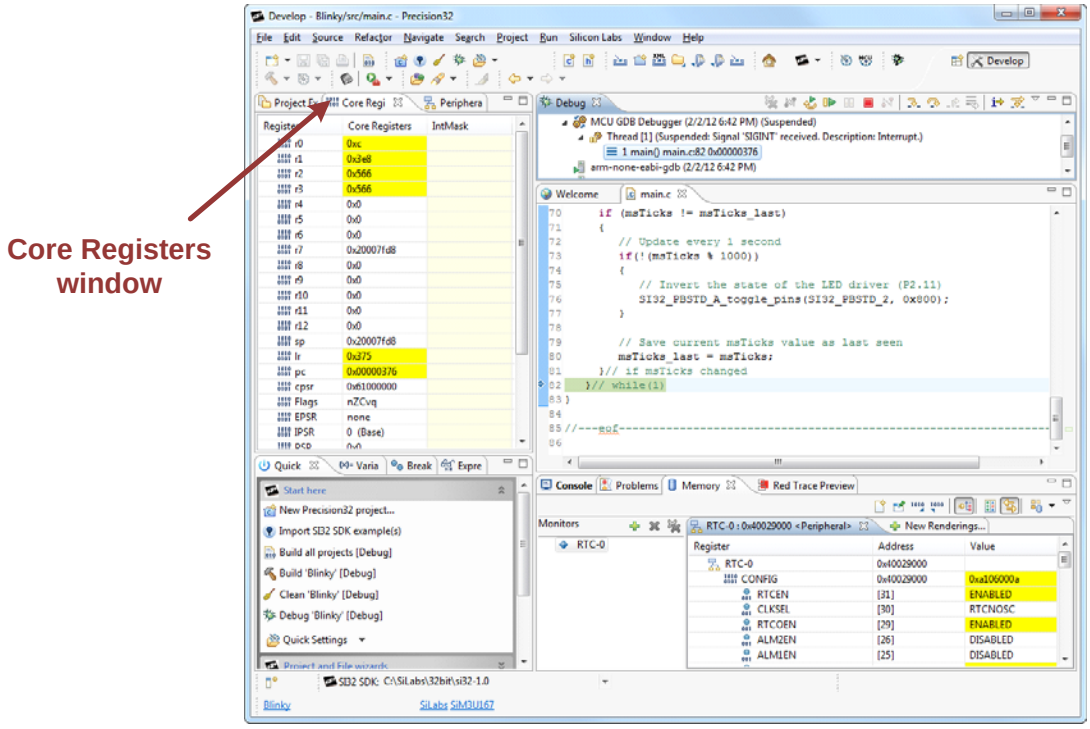

Figure 20. Core Registers View

10.7. Expressions

The **Expressions** view opens automatically when a new watch variable is added by right-clicking on the **Code Editor** view and selecting **Add Watch Expression...**. This view displays the contents of variables, structures, and other data types added to the watch list.

Figure 21 shows the **Expressions** view.

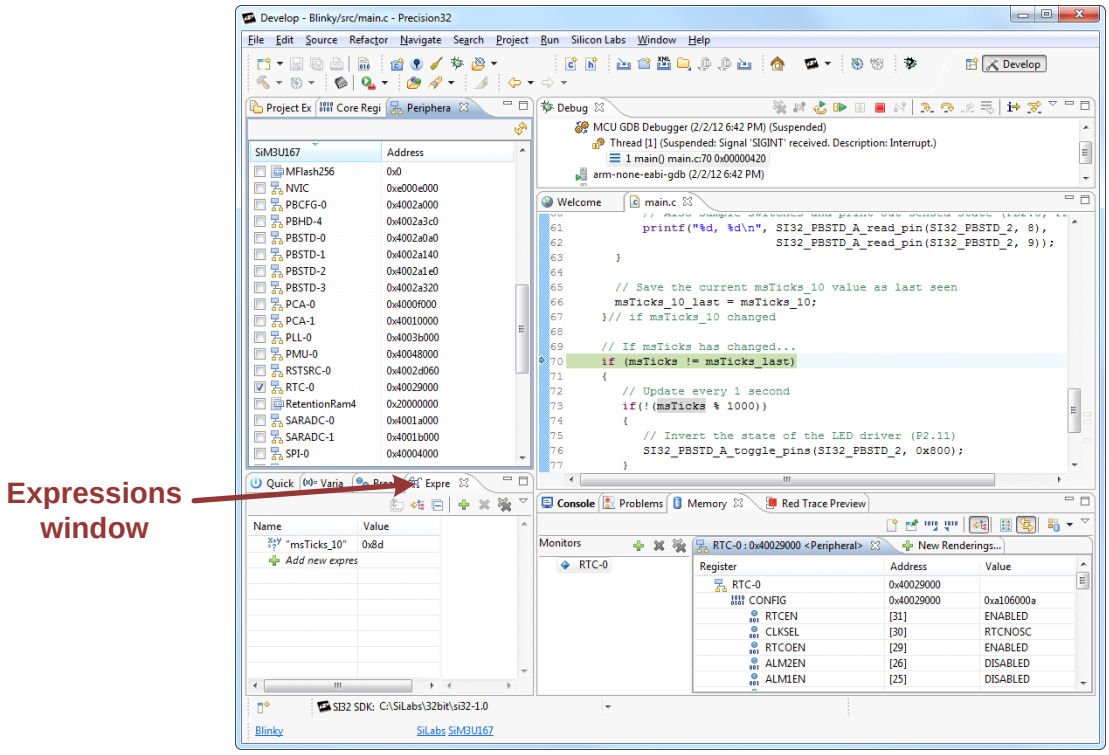

Figure 21. Expressions View

11. Getting Started: Blinky Example

To run and debug the Blinky example:

1. Open the Precision32 IDE.
2. (Optional) Register the IDE using the steps listed in “6. Activating Precision32 IDE”.
3. Set the Silicon Labs SDK path (**C:\SiLabs\32bit\si32-x.y** by default) in the drop-down menu at the bottom of the IDE.
4. Import the Blinky example project:
   a. Click the **Import SI32 SDK example(s)** link in the **Quickstart** view.
   b. Select the checkbox for the Blinky project.
   c. Select the **Copy projects into workspace** checkbox.
   d. Press the **Finish** button.
5. Select the **Blinky** project in the **Project Explorer** view.
6. Build the project by selecting **Quickstart→Build ‘Blinky’ [Debug]**.
7. Connect the Silicon Labs Debug Adapter ribbon cable to the header on the MCU card.
8. Connect the Silicon Labs Debug Adapter to the PC.
9. Start a debug session by selecting **Quickstart→Debug ‘Blinky’ [Debug]**.
10. Press the **Debug→Resume** button to run the code. The LEDs on the MCU card will blink and the **Console** view will print the status of the switches.

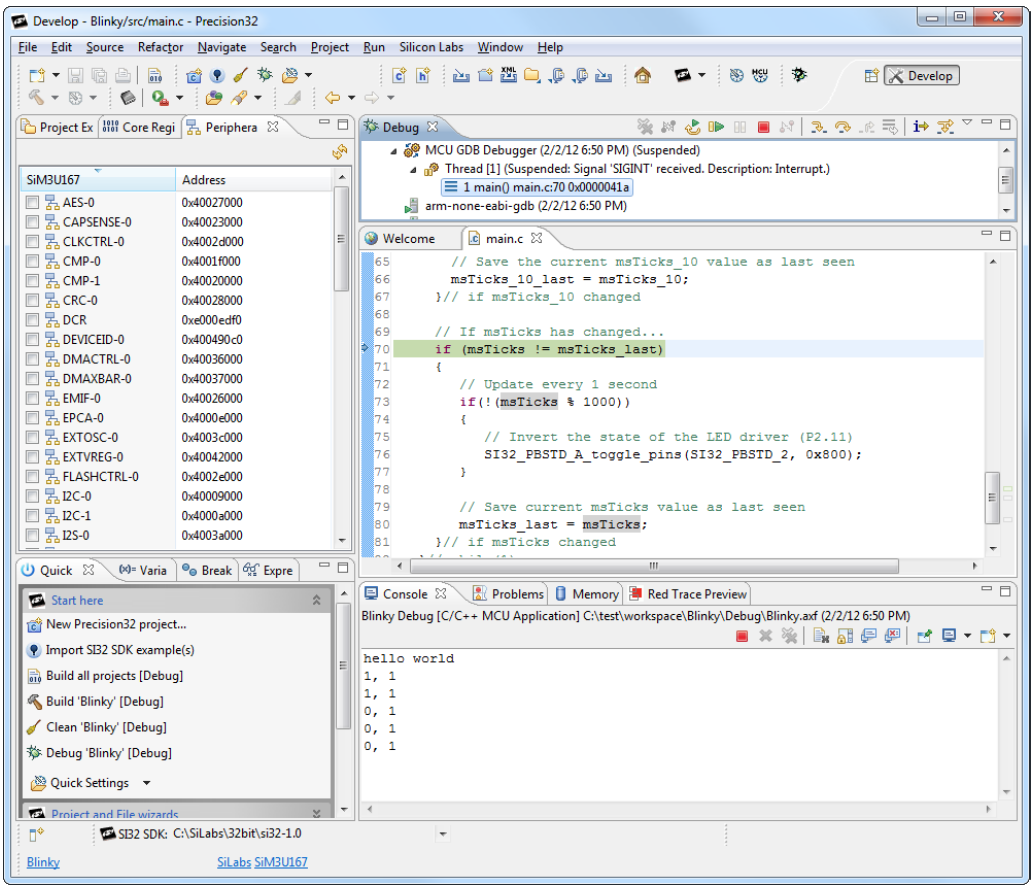

Figure 22. Blinky Example

Contact Information

Silicon Laboratories Inc.

400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Please visit the Silicon Labs Technical Support web page:
https://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.