



SILICON LABS

## PRECISION32™ IDE AND APPBUILDER DETAILED TUTORIAL AND WALKTHROUGH

### 1. Introduction

This document provides a step-by-step tutorial walkthrough for the Precision32 Development Tools (IDE and AppBuilder) using the **SiM3U1xx-B-DK** hardware and **Silicon Labs SDK version 1.0.1**. The first series in the walkthrough imports a pre-existing example into the IDE and discusses the IDE debugging basics. The second series guides the creation of a **TIMER0 Blinky** example using AppBuilder and adds some additional code editing and debugging tips for the IDE.

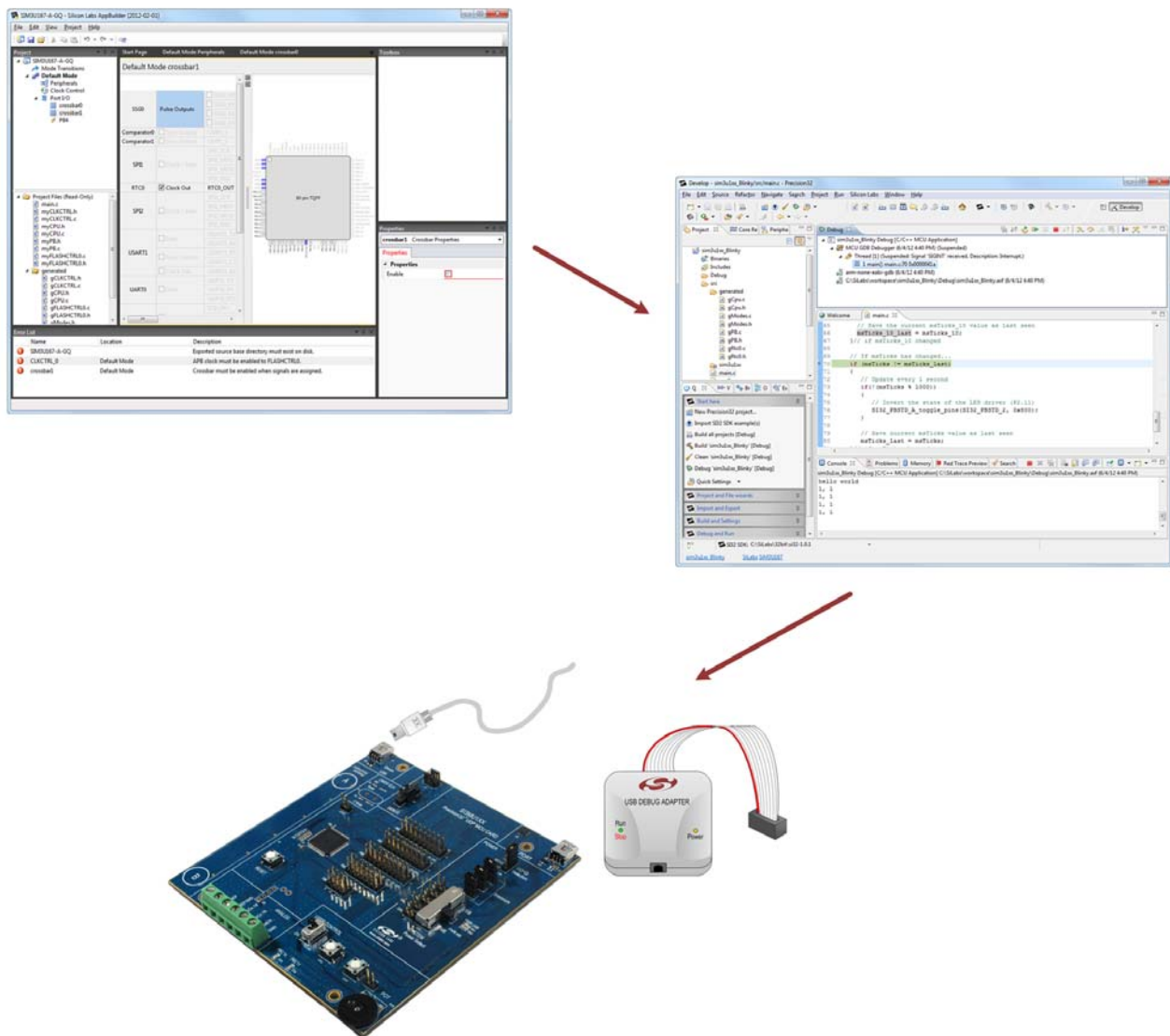


Figure 1. Precision32 IDE and AppBuilder Walkthrough Overview

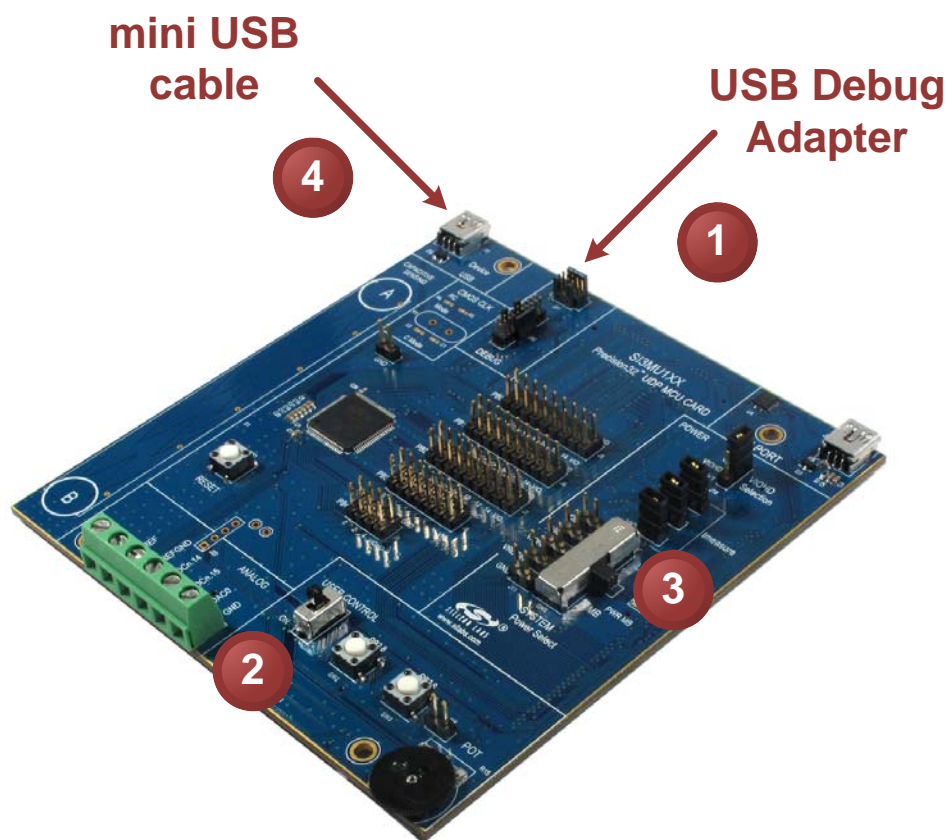
## 2. Relevant Documentation

Precision32 Application Notes are listed on the following website: [www.silabs.com/32bit-appnotes](http://www.silabs.com/32bit-appnotes).

- **AN667: Getting Started with the Silicon Labs Precision32™ IDE**—provides a description of the IDE features and environment.
- **AN670: Getting Started with the Silicon Labs Precision32™ AppBuilder**—provides a description of the AppBuilder features.

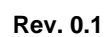
## 3. Hardware Setup

To set up the hardware for this walkthrough:



**Figure 2. Hardware Setup Diagram**

1. Connect the USB Debug Adapter to J31 and connect the Debug Adapter to the PC with a USB cable.
2. Move the **USER CONTROL** MCU card switch (SW4) to ON.
3. Move the **SYSTEM Power Select** switch (SW5) to the USB position.
4. Connect a mini USB cable to the Device USB connector (J13). Connect the cable to the PC to power the board.



3

5. Verify the shorting blocks are installed as shown in Figure 3:
  - a. J18 top two pins
  - b. J14
  - c. J15
  - d. J17
  - e. J16

## 4. Lab Software Setup

1. Install the Precision32 package from [www.silabs.com/32bit-software](http://www.silabs.com/32bit-software).
2. Open the Precision32 IDE and activate it, if it's not activated already. Instructions on how to do this are available on the Welcome page in the IDE.

## 5. Walkthrough Series #1: Running Blinky in the Precision32 IDE

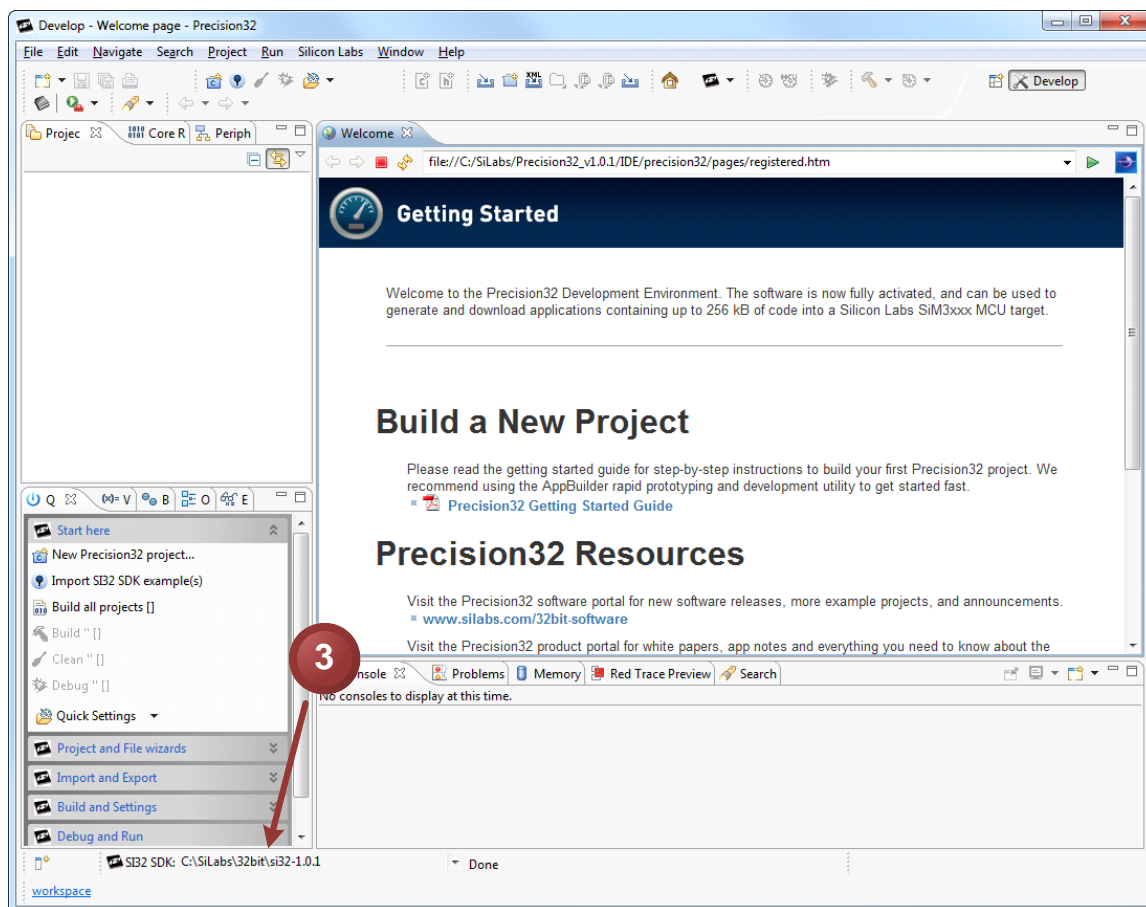
### 5.1. Objectives

The objectives of series are:

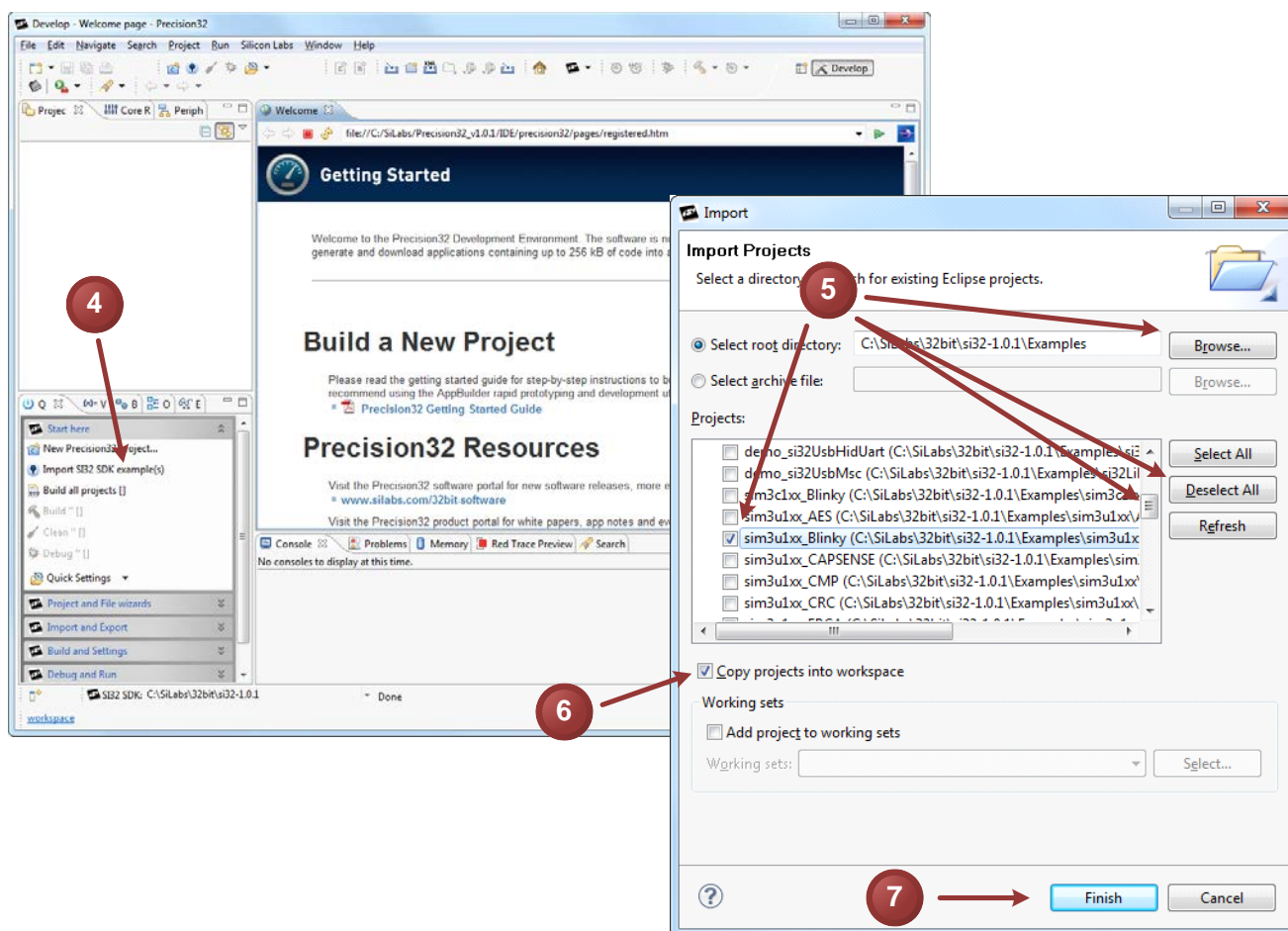
- Learn how to import an SDK example into the Precision32 IDE.
- Learn how to download a program to a device.
- Explore some of the debugging features of the IDE.

### 5.2. Instructions

1. Launch the **Precision32 IDE 1.0.1** (default path: C:\SiLabs\Precision32\_v1.0.1\IDE) and select a workspace (**C:\SiLabs\workspace**).
2. Activate the IDE using the instructions on the landing page, if the IDE isn't already activated.
3. Set the Silicon Labs SDK path to **C:\SiLabs\32bit\si32-1.0.1**.

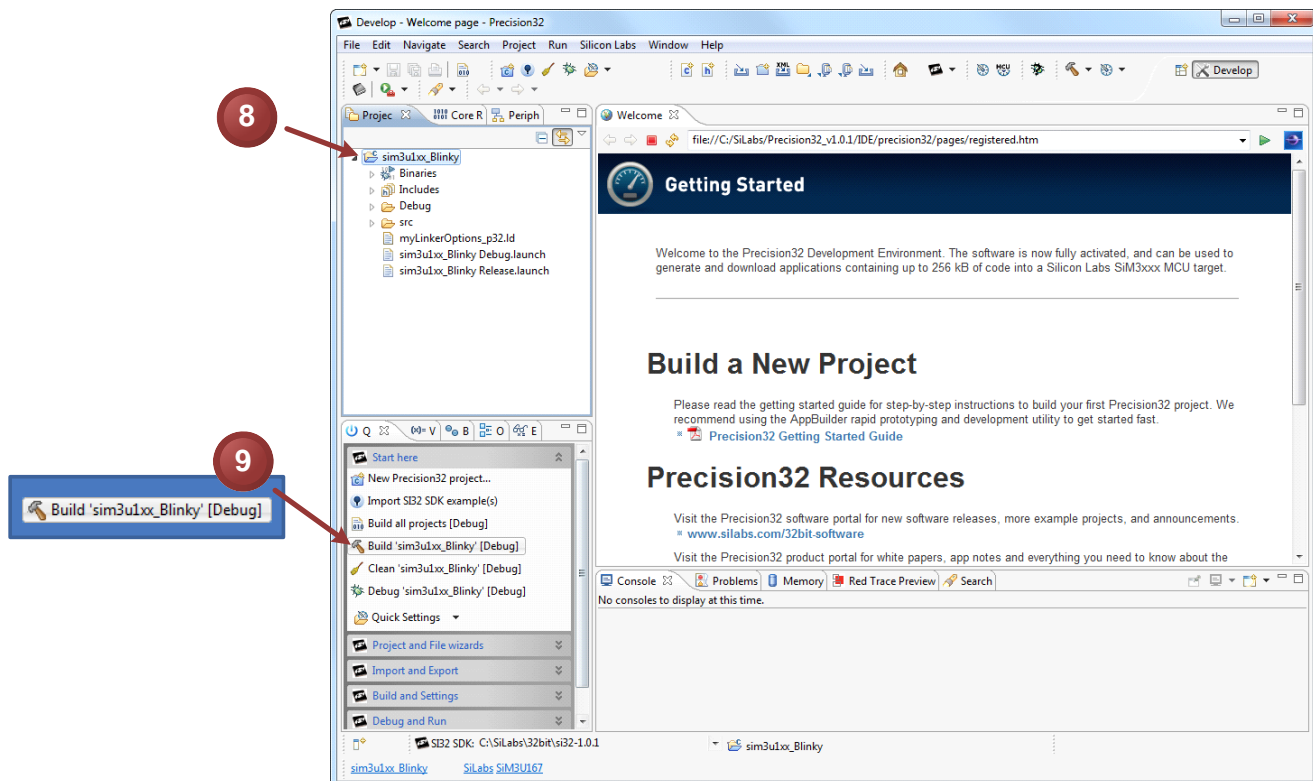


4. Click the **Import SI32 SDK example(s)** link in the **Quickstart** view.
5. Scroll down and check just the **sim3u1xx\_Blinky** example. The default path for importing projects is C:\SiLabs\32bit\si32-x.y.z\Examples, where x.y.z is the version of the SDK selected in the previous step, but the **Browse...** button can be used to navigate to a specific example directory. The **Deselect All** button is also useful for deselecting a long set of examples.
6. Ensure the **Copy projects into workspace** checkbox is enabled.
7. Click **Finish**.

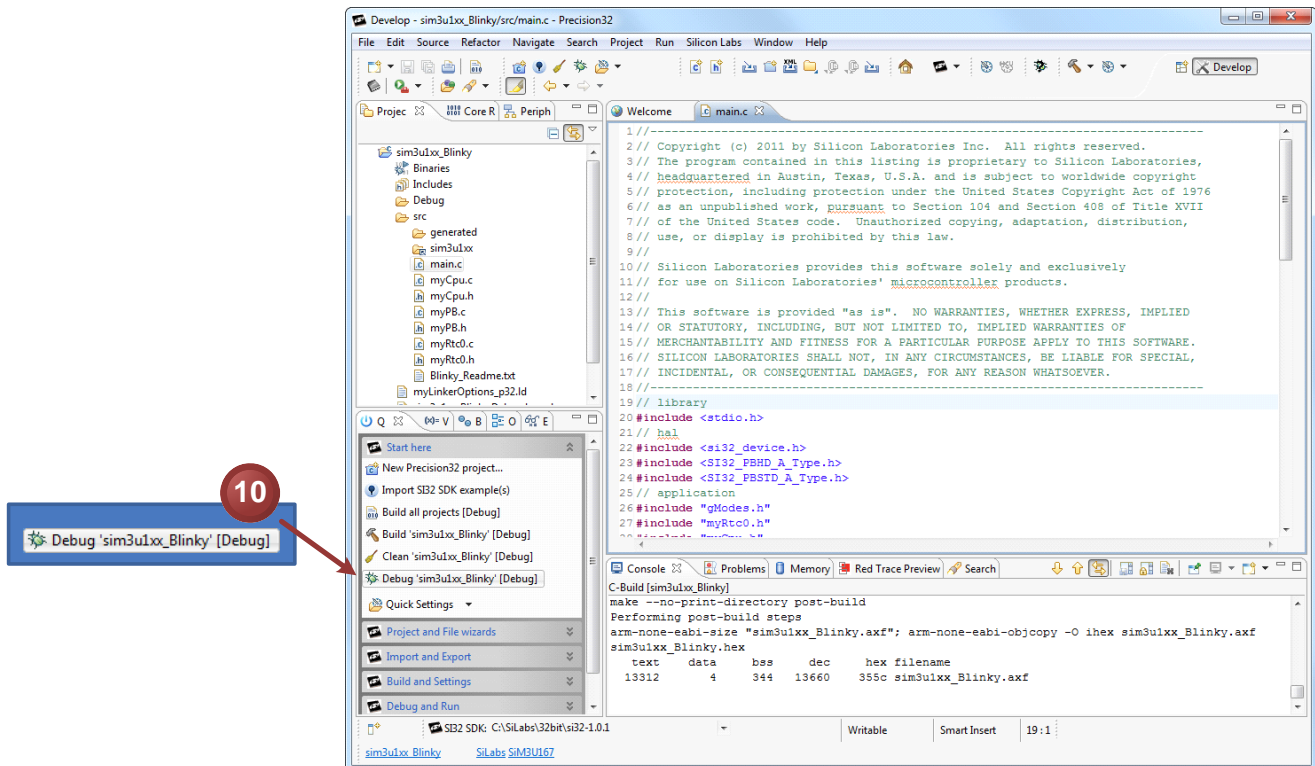




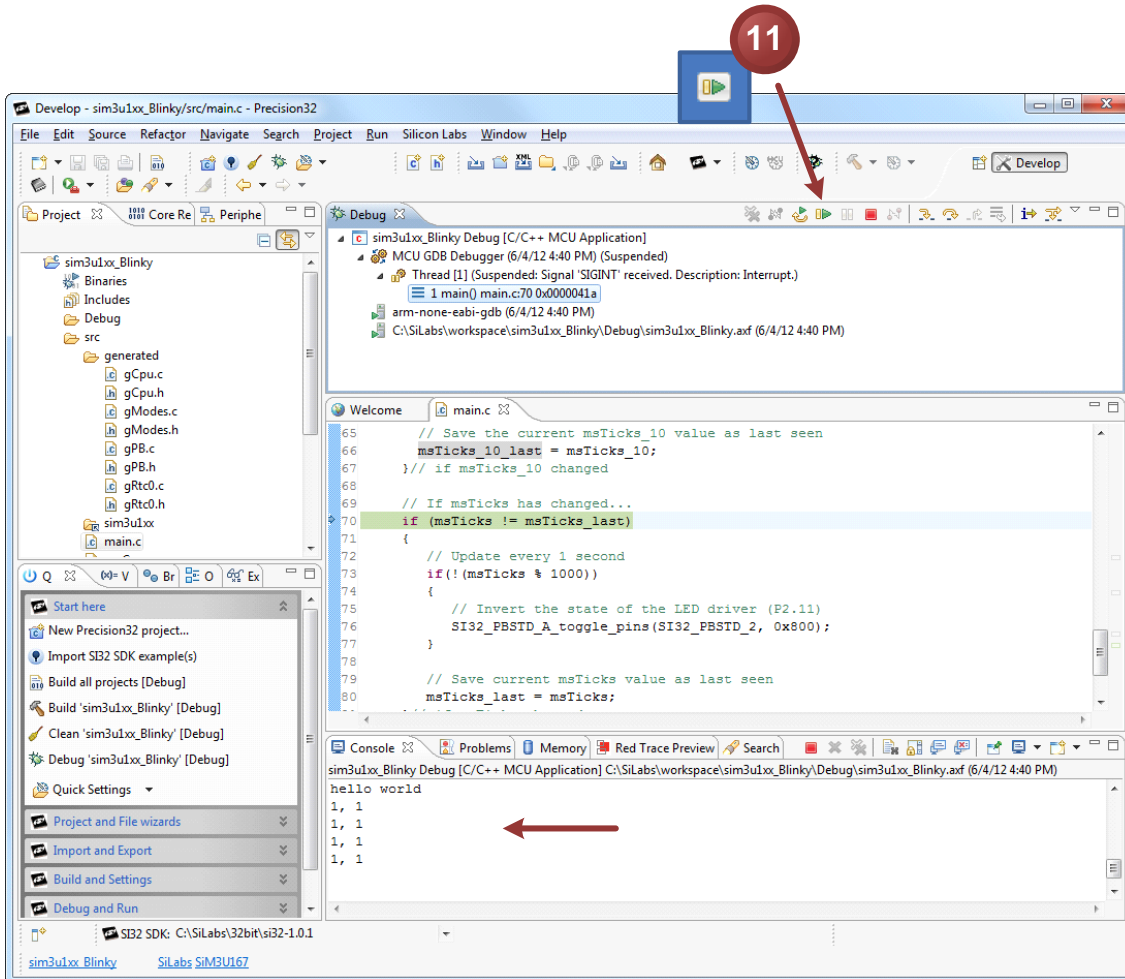
- Click the **sim3u1xx\_Blinky** project name.
- Click the **Build 'sim3u1xx\_Blinky' [Debug]** link.



- Click the **Debug 'sim3u1xx\_Blinky' [Debug]** link. The **Debug** view will open with trace information.

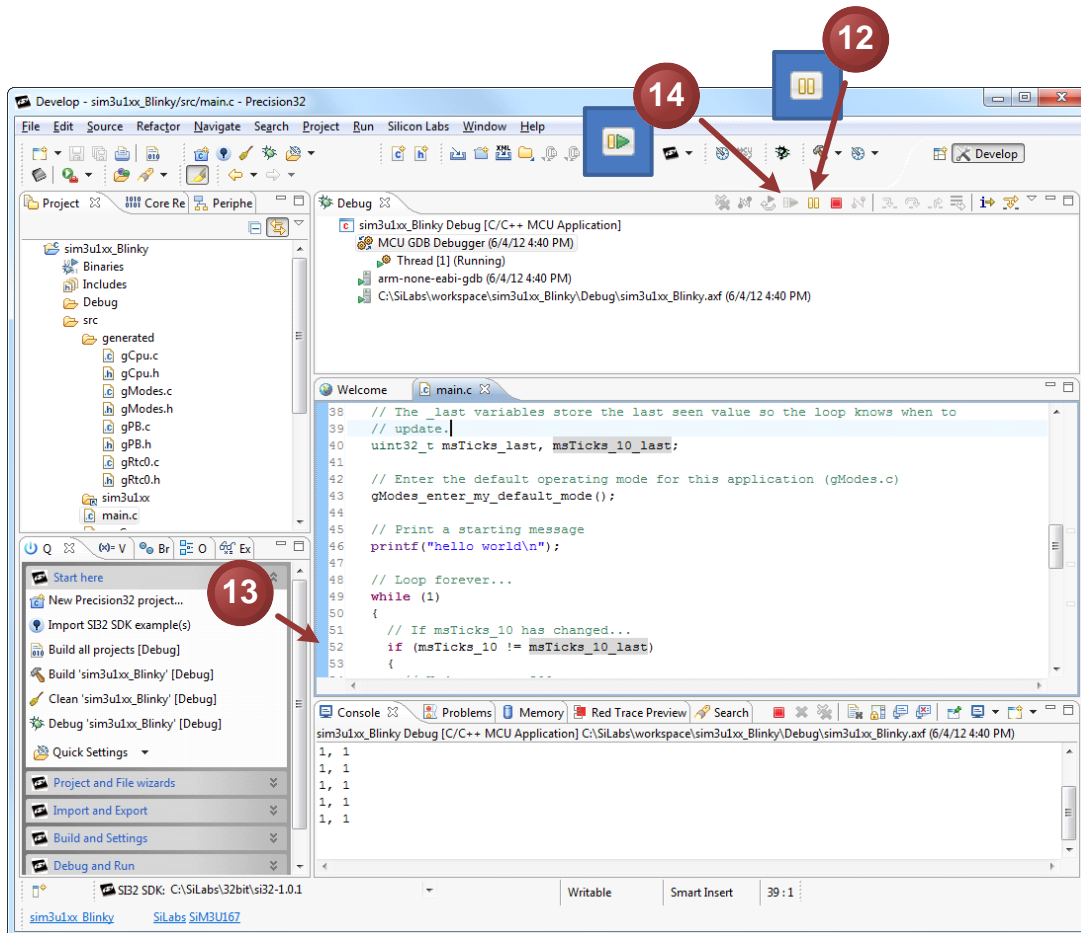


11. Start the program. PB2.10 and PB2.11 will blink after a small delay. The printf() output from the example will appear in the **Console** view.

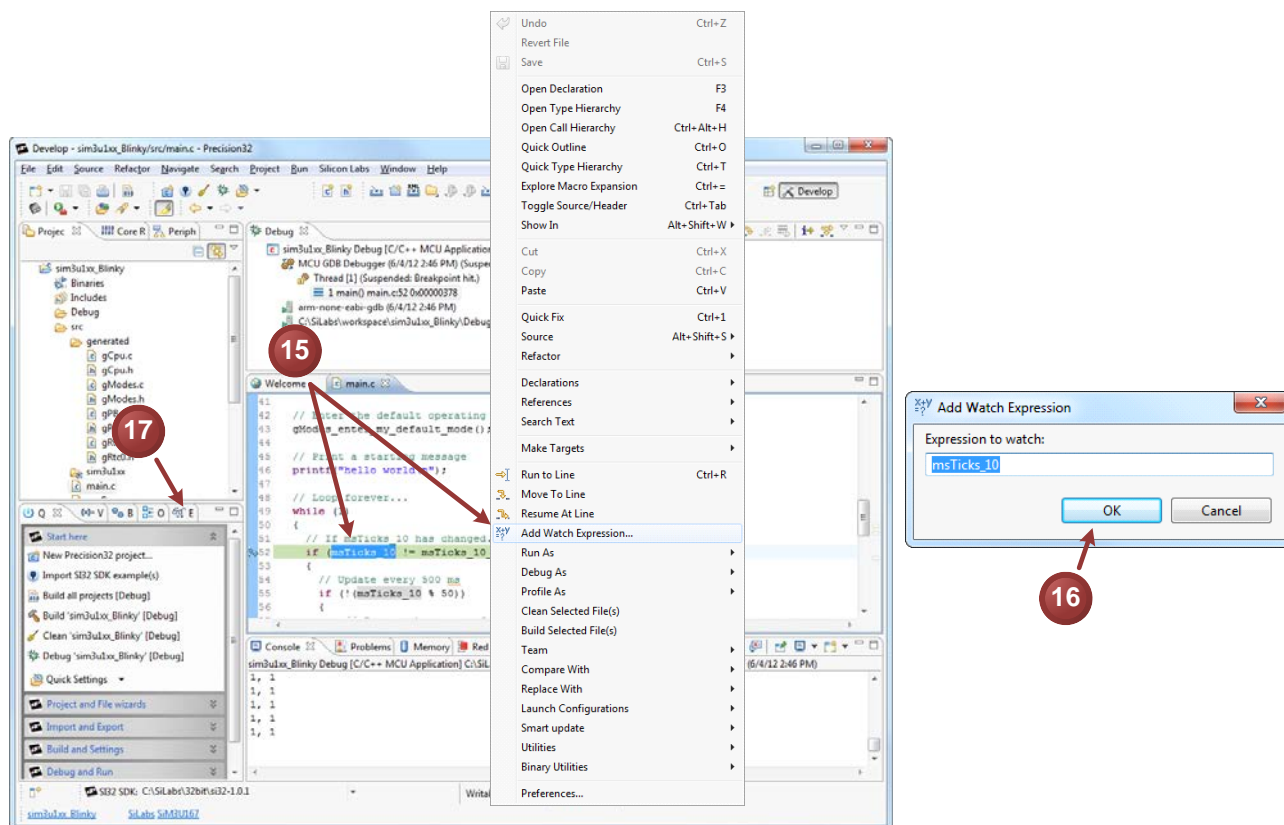




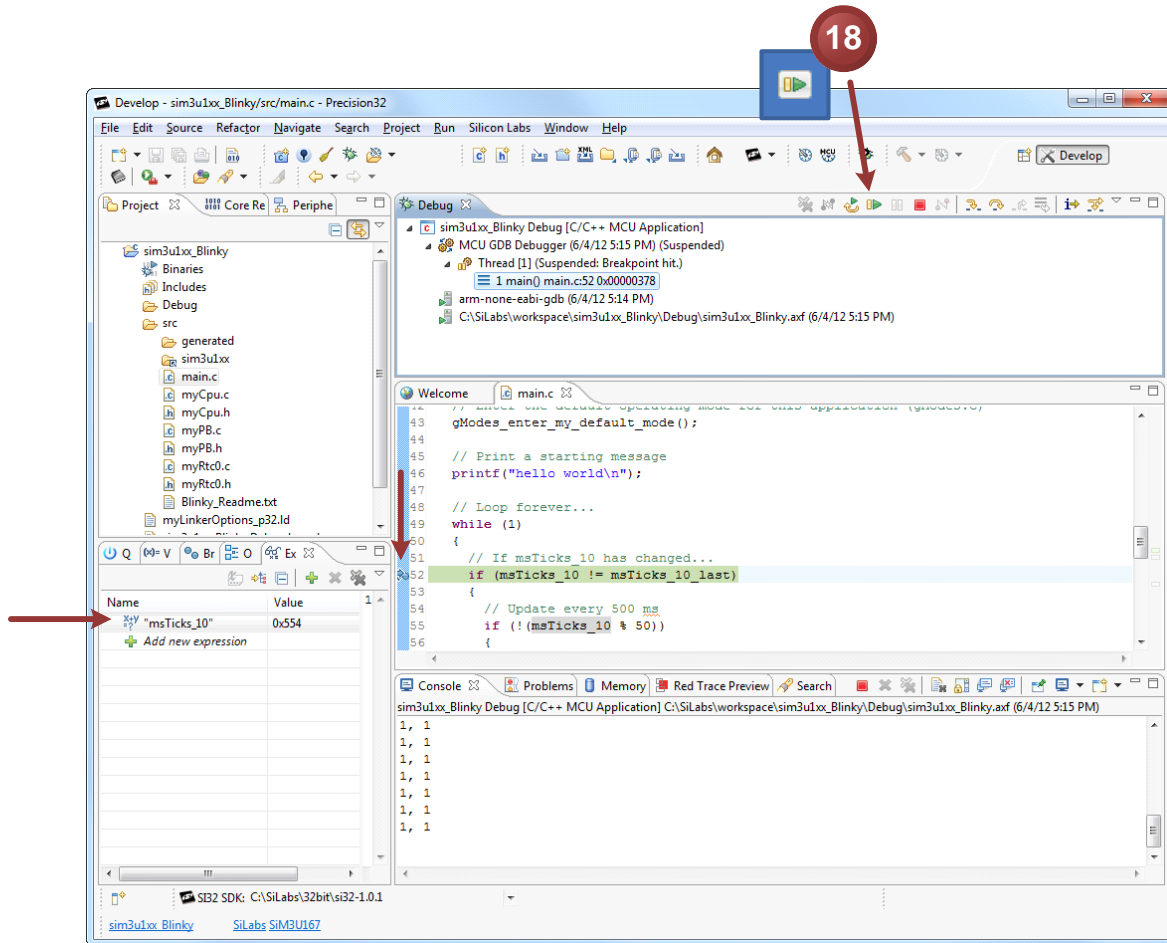
12. Pause the program.
13. Double-click the blue area next to a line to add a breakpoint.
14. Start the program again.



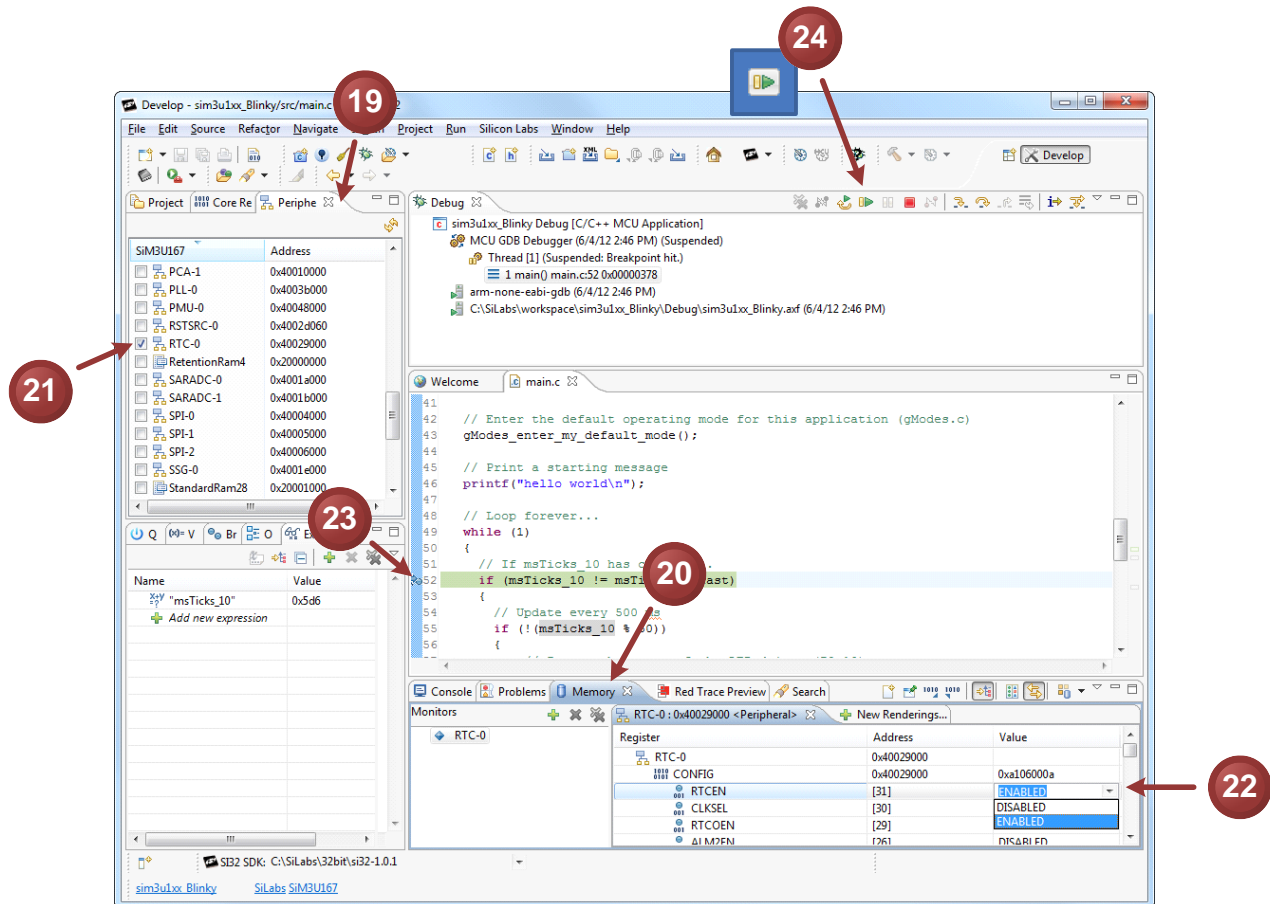
15. Highlight the **msTicks\_10** variable in the **main.c** file, right click, and select **Add Watch Expression...**
16. Press OK in the **Add Watch Expression** dialog.
17. Select the **Expression** view.



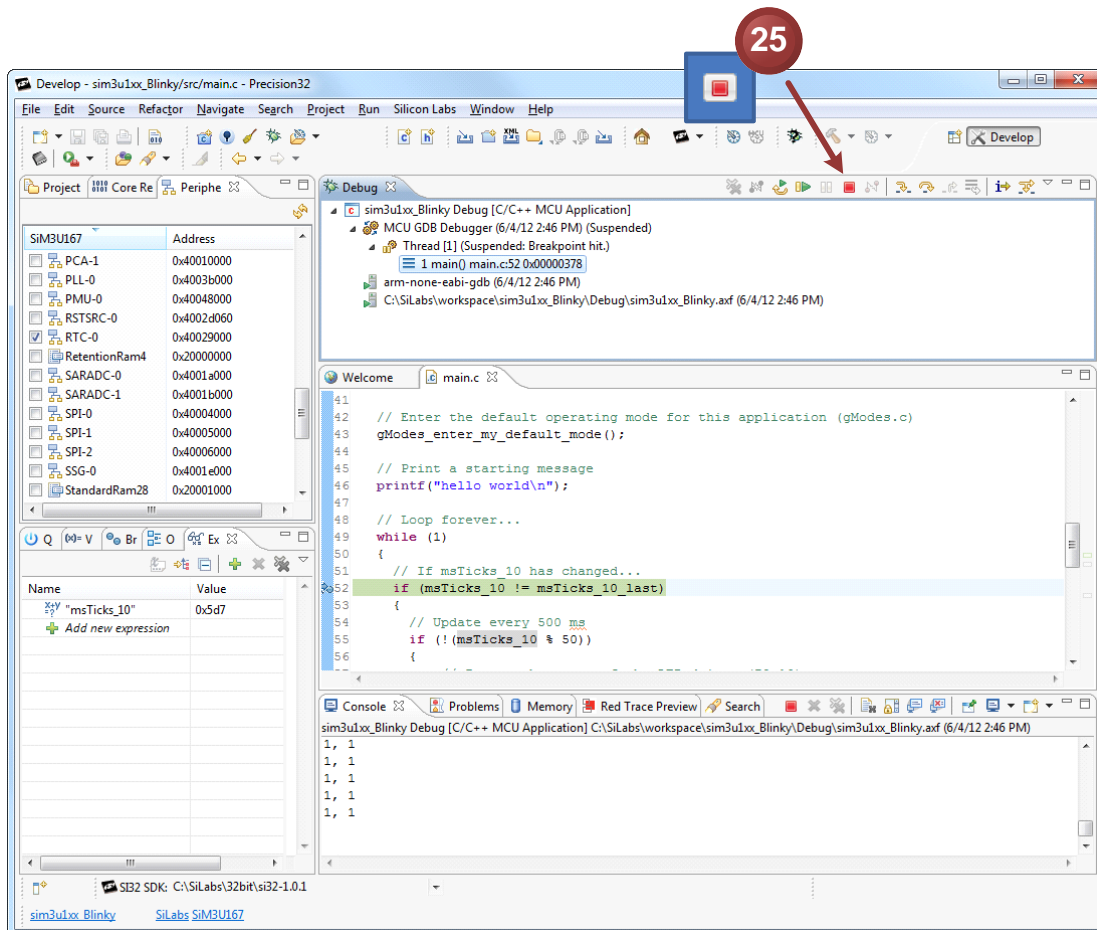
18. Click the **Resume** button a few times. The program will stop each time at the breakpoint and update the value of `msTicks_10`.



19. Select the **Peripheral** view.
20. Select the **Memory** view.
21. Enable the **RTC-0** checkbox.
22. Disable the RTC by selecting **DISABLED** for the **RTCCEN** bit.
23. Double-click on the breakpoint to remove it.
24. Start the program again. The PB2.10 LED will stop blinking since the RTC is no longer running. PB2.11 still blinks because it's running from the SysTick timer.



25. Click the **Terminate** button to end the debugging session.





## 6. Walkthrough Series #2: Creating a Timer Blinky

### 6.1. Objectives

The objectives of this series are:

- Use AppBuilder to configure the TIMER0 high 16-bit timer to generate a 12 Hz waveform on PB2.10 (red LED).
- Use the information learned in Lab 1 to download and debug the generated AppBuilder project.

### 6.2. Lab Preparation

The default SiM3U1xx oscillator is the Low Power Oscillator at 20 MHz, but this doesn't create a 12 Hz waveform cleanly:

$$\frac{20000000}{12} = 1666666.67$$

Instead, we can use the 48 MHz USB0 oscillator. The actual overflow rate desired for the timer is 12 Hz multiplied by two, since the timer must overflow for each rising and falling edge of the waveform on the pin. This means our TIMER0 effective overflow rate is:

$$\frac{48000000}{12 \times 2} = 2000000$$

The TIMER0 module has two ways to meet this overflow rate. The module can clock from a divided version of APB. This prescaler (CLKDIV and CLKDIVRL) is an integer value and can be arbitrarily chosen between 0 and 255 (effectively creating APB divided by 256 up to APB divided by 1). In addition, the Timer count interval is 16 bits in split mode, so it can be anywhere between 0 and 65535 (setting the timer count to 0xFFFF causes the timer to overflow in one clock). We can choose these values in any way such that they equal the required count of 2000000.

If we choose to set the TIMER0 APB prescaler to 40 (for a TIMER0 clock of 1.2 MHz):

$$\frac{2000000}{40} = 50000$$

Prescaler (CLKDIVRL) value from the reference manual:

$$F_{\text{prescale}} = \frac{F_{\text{APB}}}{256 - \text{CLKDIVRL}}$$

$$256 - \text{CLKDIVRL} = 256 - 40 = 216$$

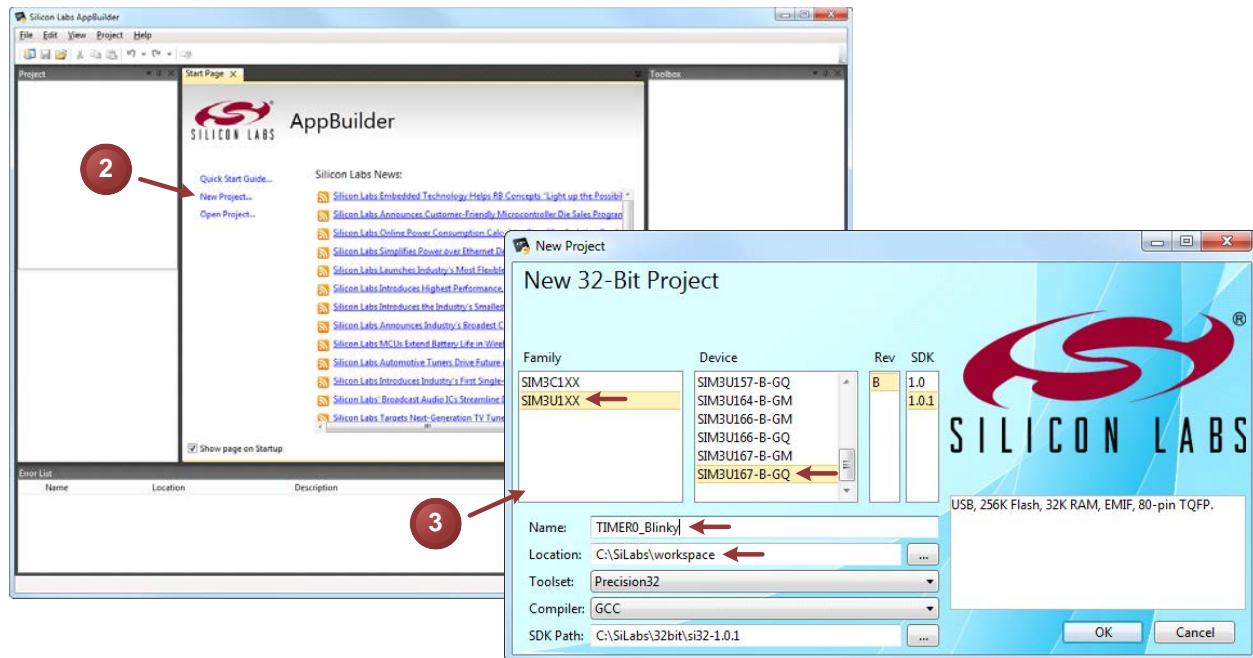
16-bit reload value (HCCR) from the reference manual:

$$\text{Overflow Rate} = \frac{F_{\text{TIMER}}}{65536 - \text{HCCR}}$$

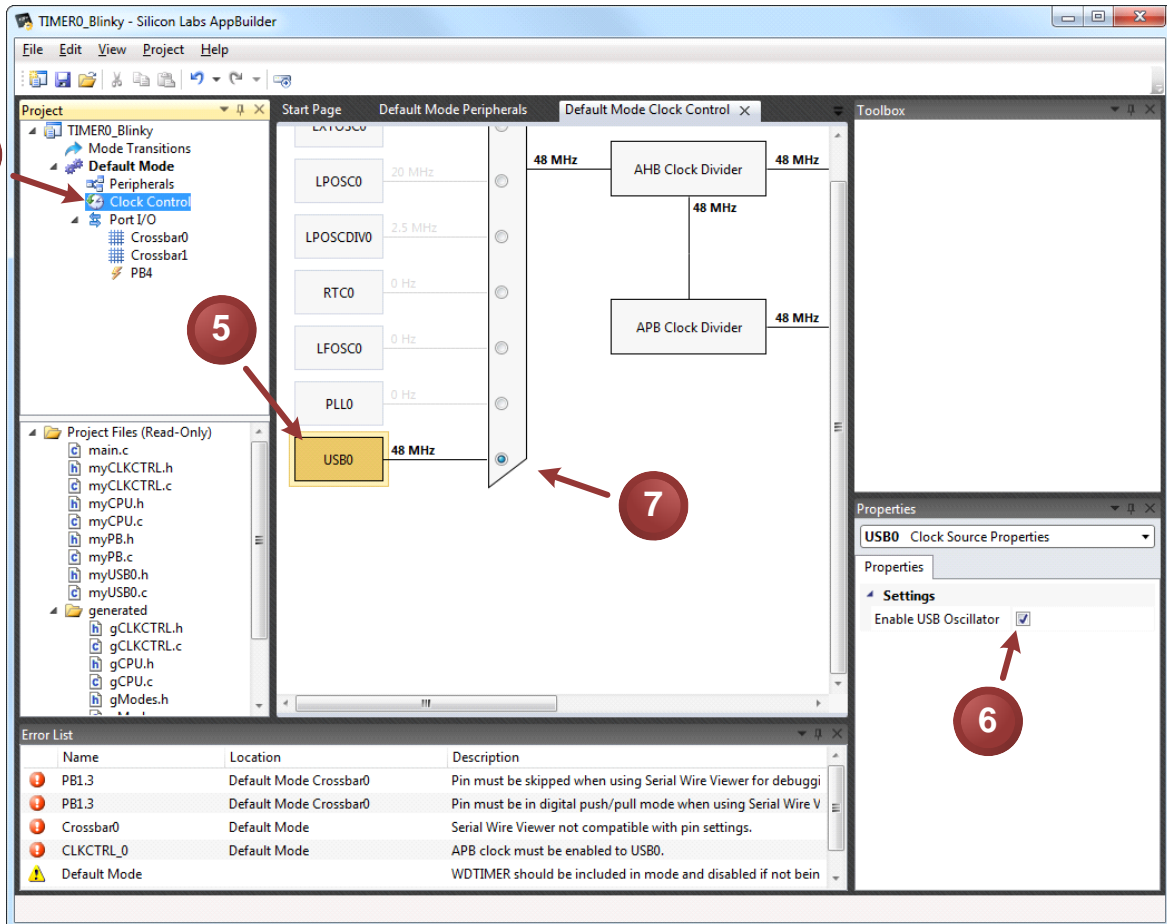
$$65536 - \text{HCCR} = 65536 - 50000 = 15536$$

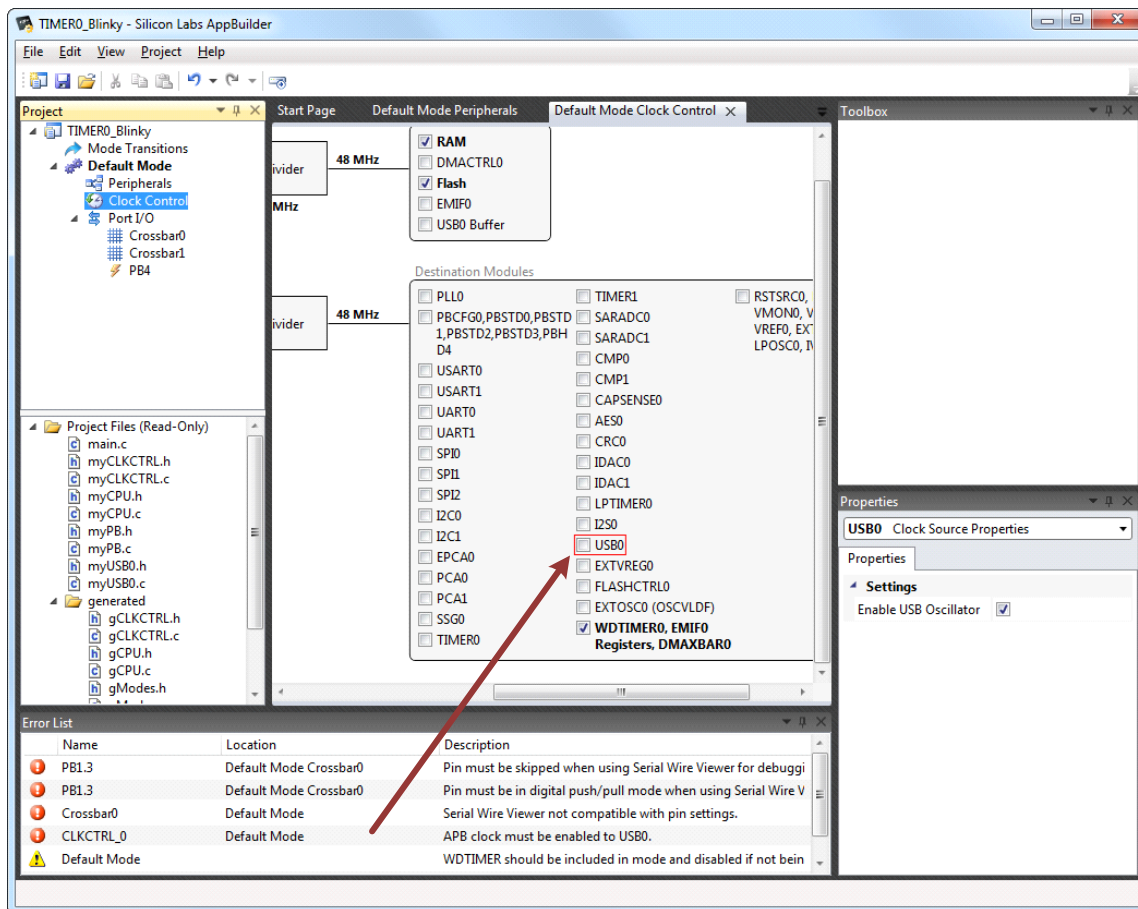
### 6.3. Instructions

1. Launch **AppBuilder 1.0.1** (default path: C:\SiLabs\Precision32\_v1.0.1\AppBuilder).
2. Select **New Project....**
3. Select the project settings. **SiM3U1xx**→**SiM3U167-B-GQ**→**B**→**1.0.1**. Update the **Name** of the project and **Location** to the workspace location. Press **OK**.

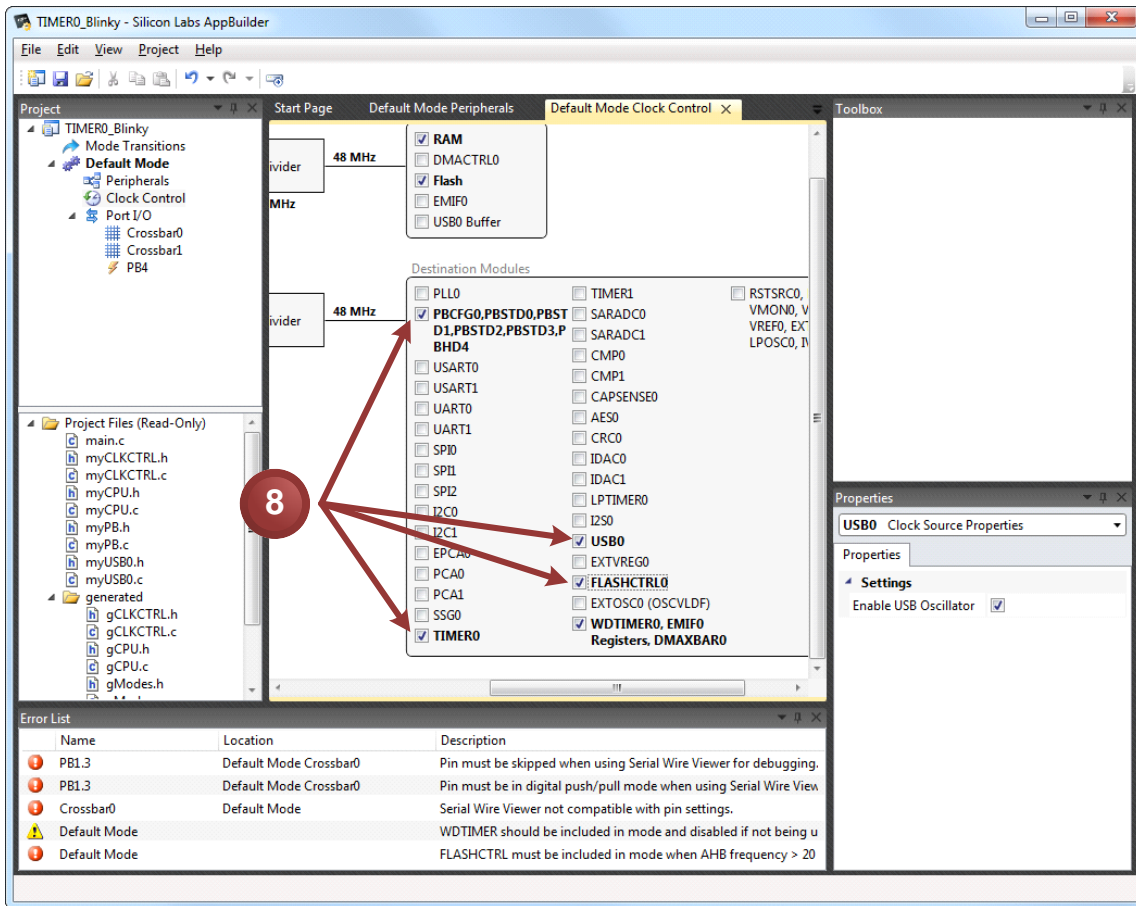


4. Double-click on **Clock Control**.
5. Select the **USB0** Oscillator.
6. Enable the USB0 Oscillator in the **Properties** window.
7. Click on the MUX radial button to select the USB0 oscillator as the AHB clock source. The diagram updates to show the current AHB and APB clock frequencies. After selecting the USB0 clock as the AHB source, an error appears indicating that the USB0 APB clock must be enabled.



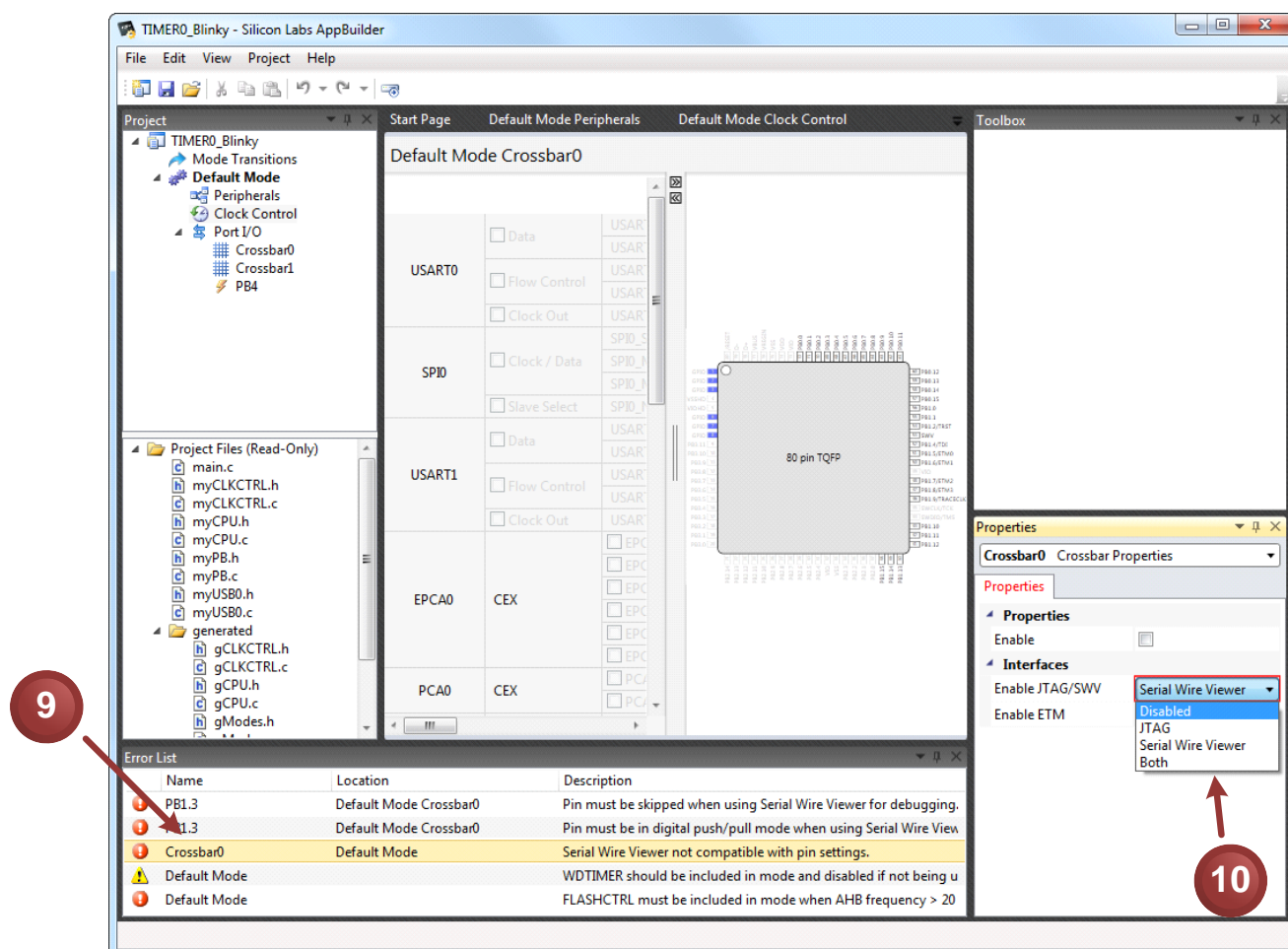


8. Enable the APB clocks to the **Ports (PBCFG0, PBSTD0, PBSTD1, PBSTD2, PBSTD3, PBHD4), TIMER0, USB0, and FLASHCTRL0**. The error and red highlight around USB0 disappear after enabling the USB0 APB clock.

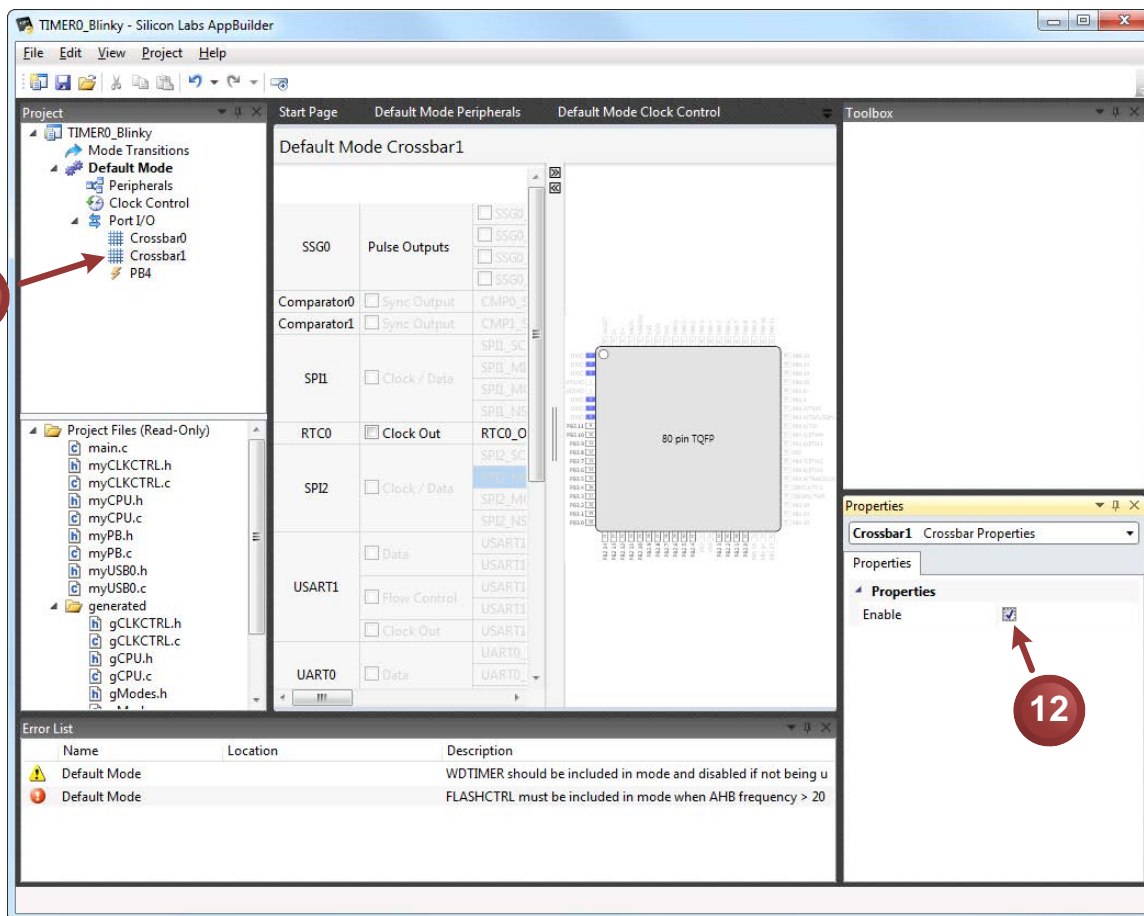




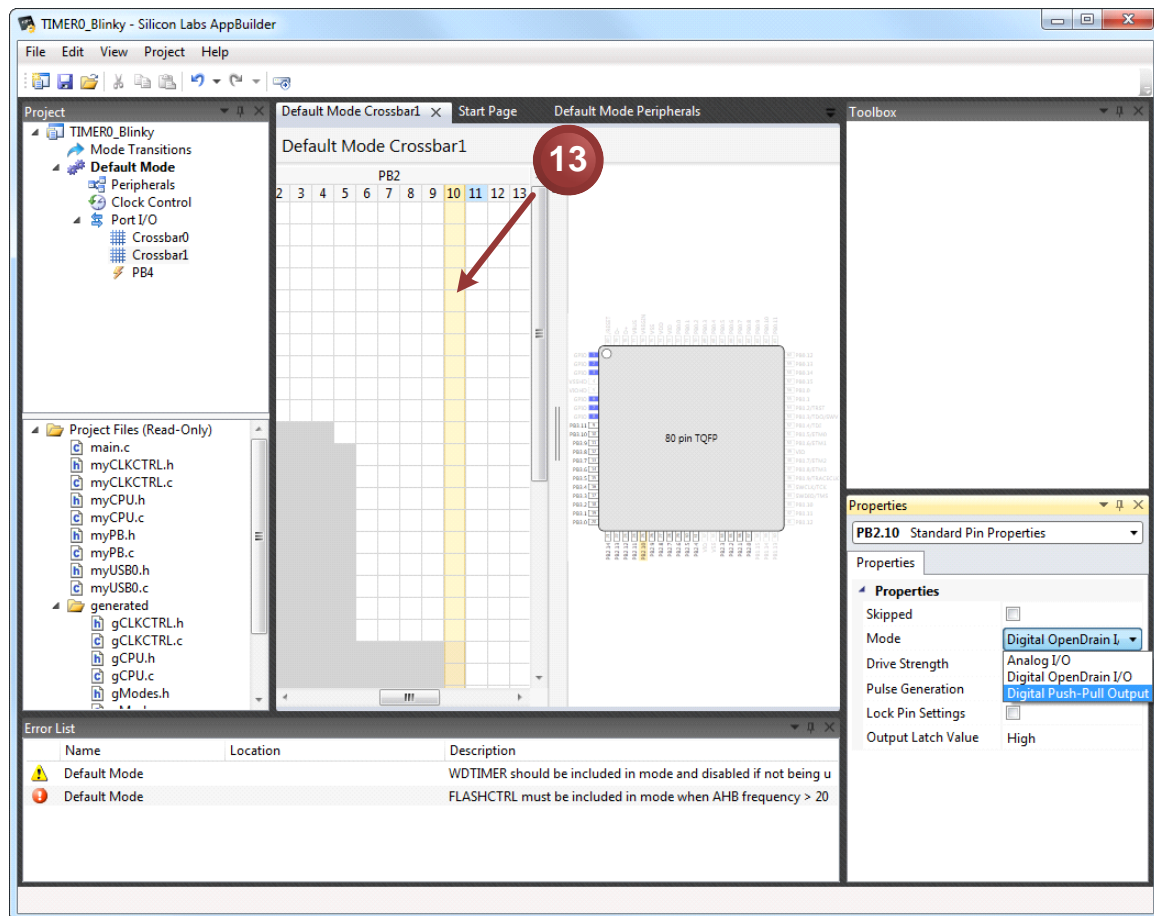
9. We can remove some errors and disable SWV since we're not using it for this application. Double-click on the crossbar 0 error in the **Error List** window.
10. Select **Disabled** in the **Enable JTAG/SWV** drop-down menu.



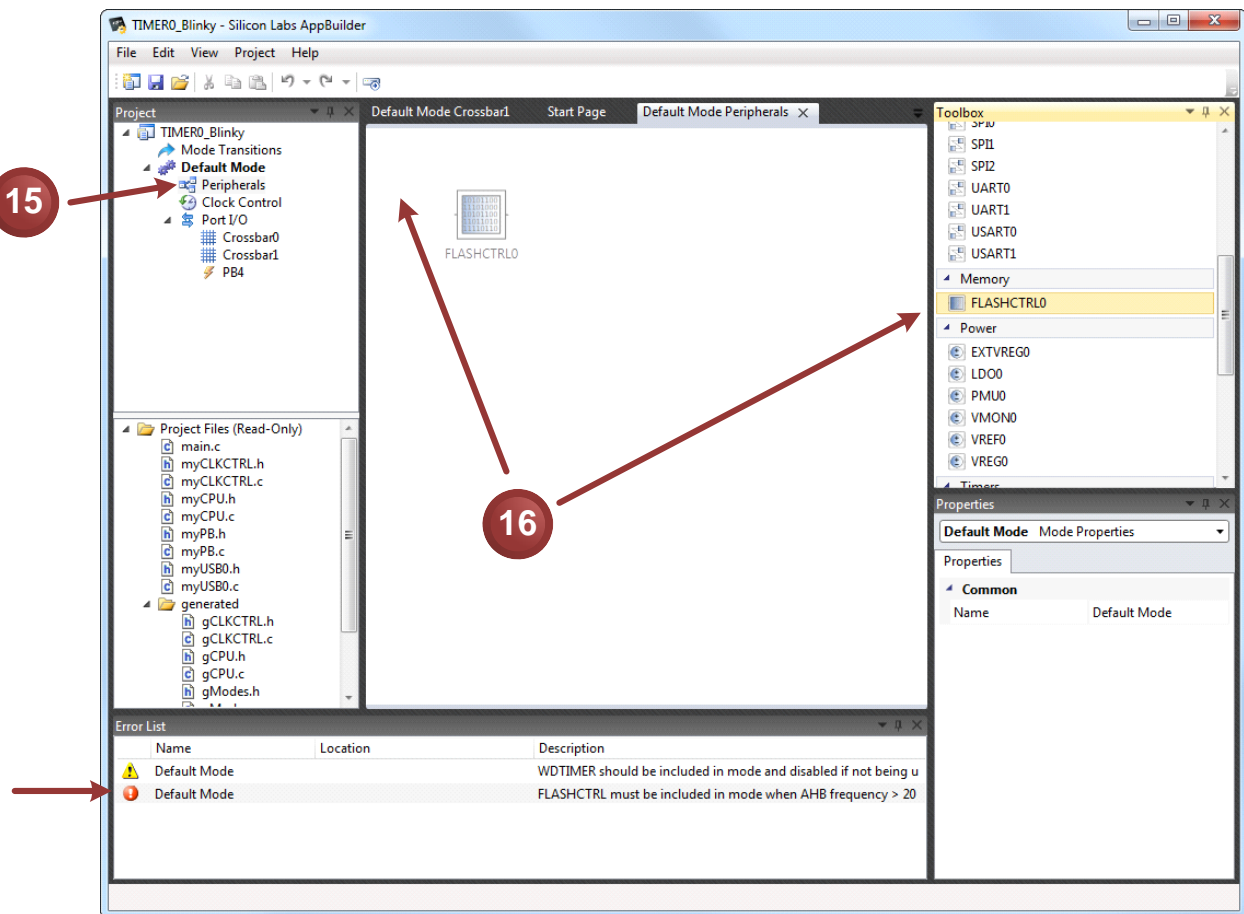
11. Double-click on **crossbar1**.
12. Enable **crossbar1**.



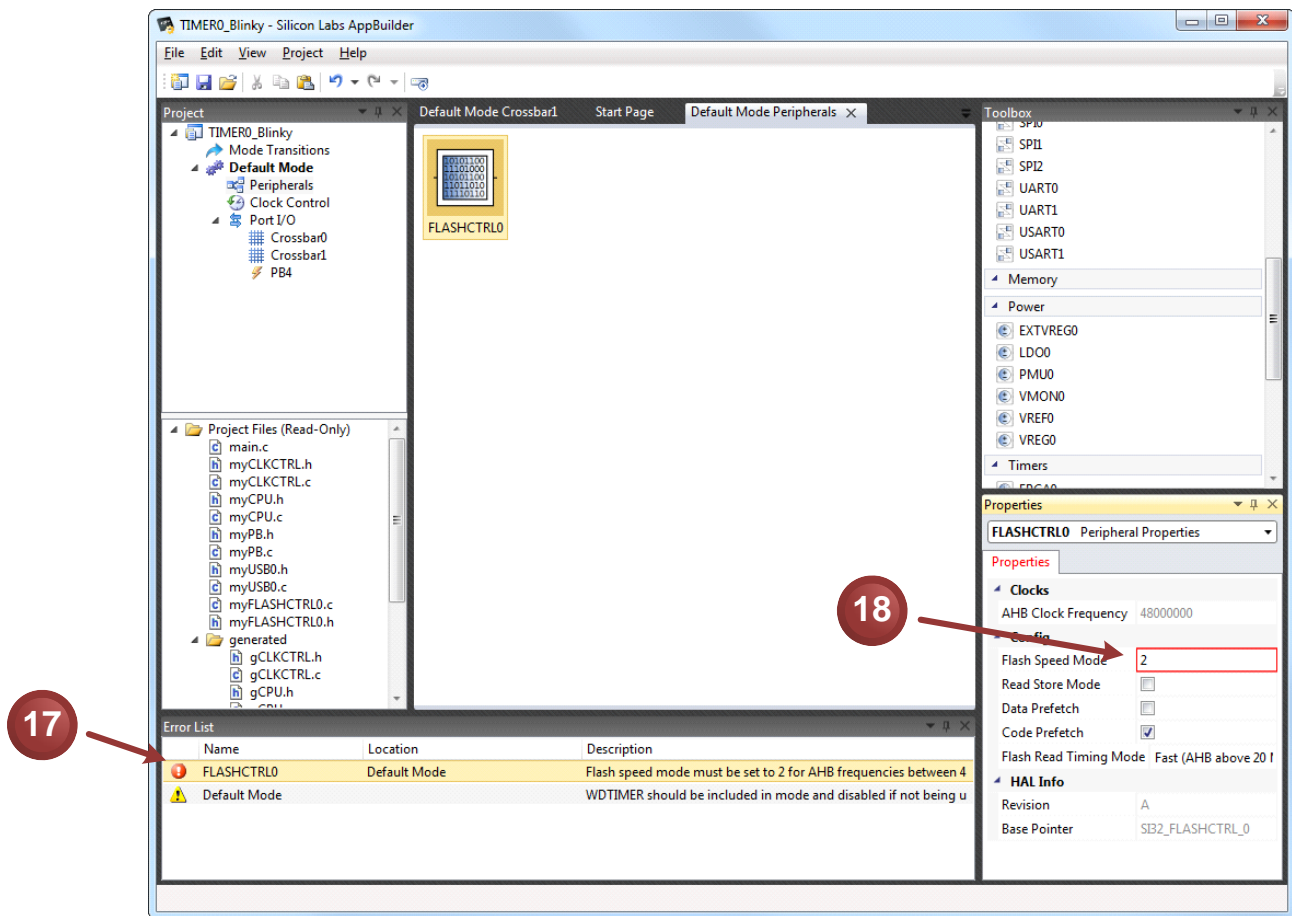
13. Click on the PB2.10 column.
14. Set PB2.10 to Push-Pull in the **Properties** window.



15. Let's fix the **FLASHCTRL0** error in the **Error List**. Double-click on **Peripherals**.
16. Drag and drop **FLASHCTRL0** to the **Peripherals** Canvas.

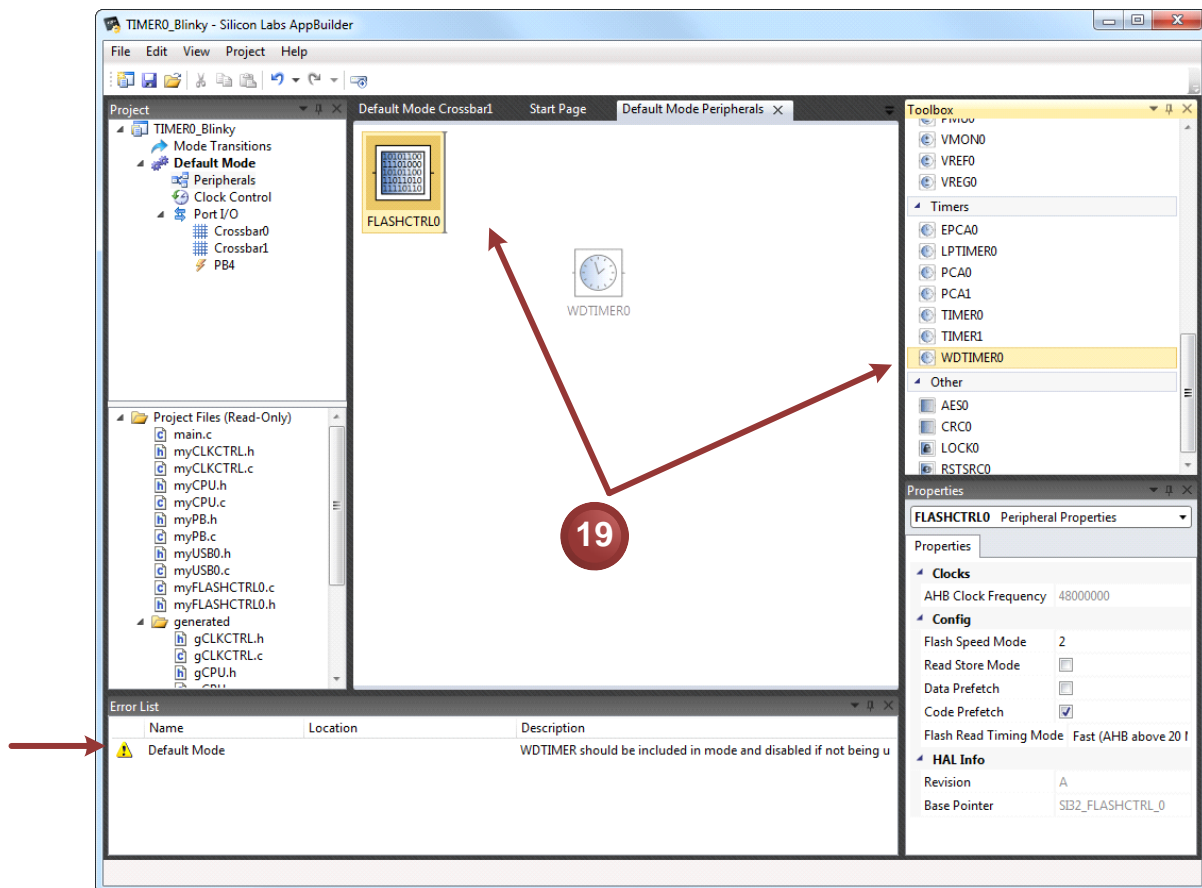


17. Double-click on the error. The program will automatically jump to the **FLASHCTRL0 Properties** window.
18. Set the **Flash Speed Mode** field to **2**. The error will disappear after hitting **Enter**.

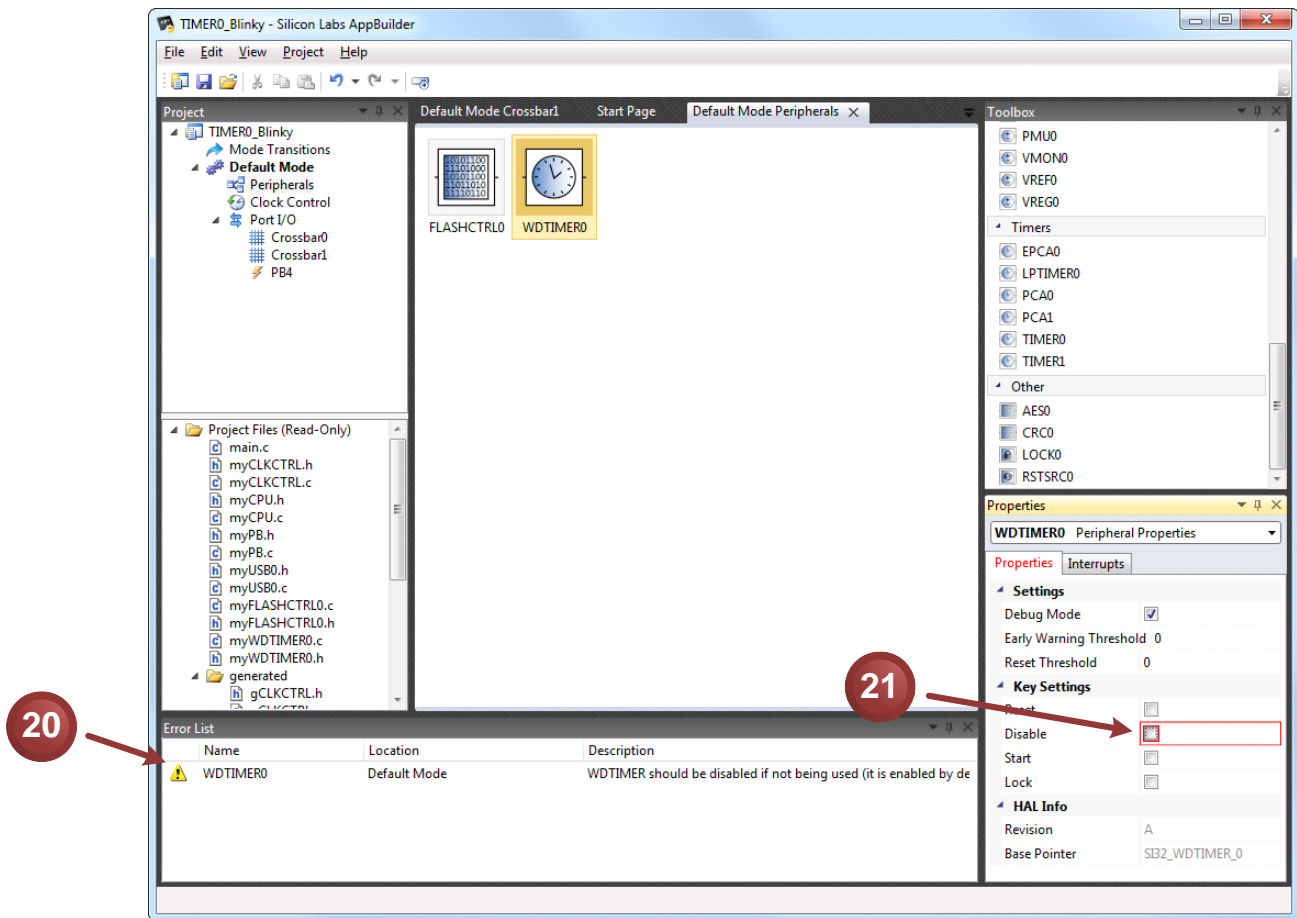




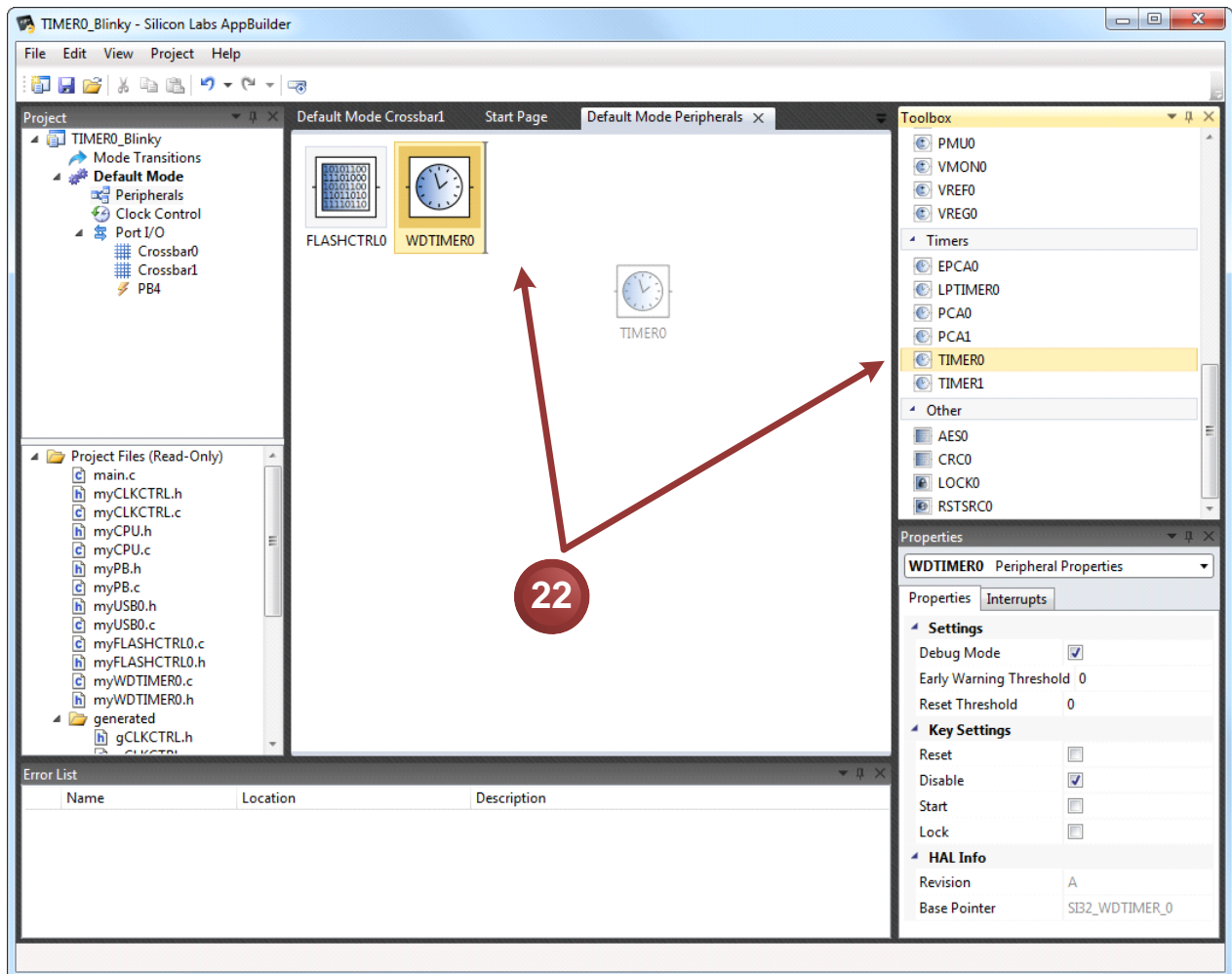
- Let's fix the last error in the Error List for the Watchdog Timer. Drag and drop **WDTIMER0** to the **Peripherals Canvas**.



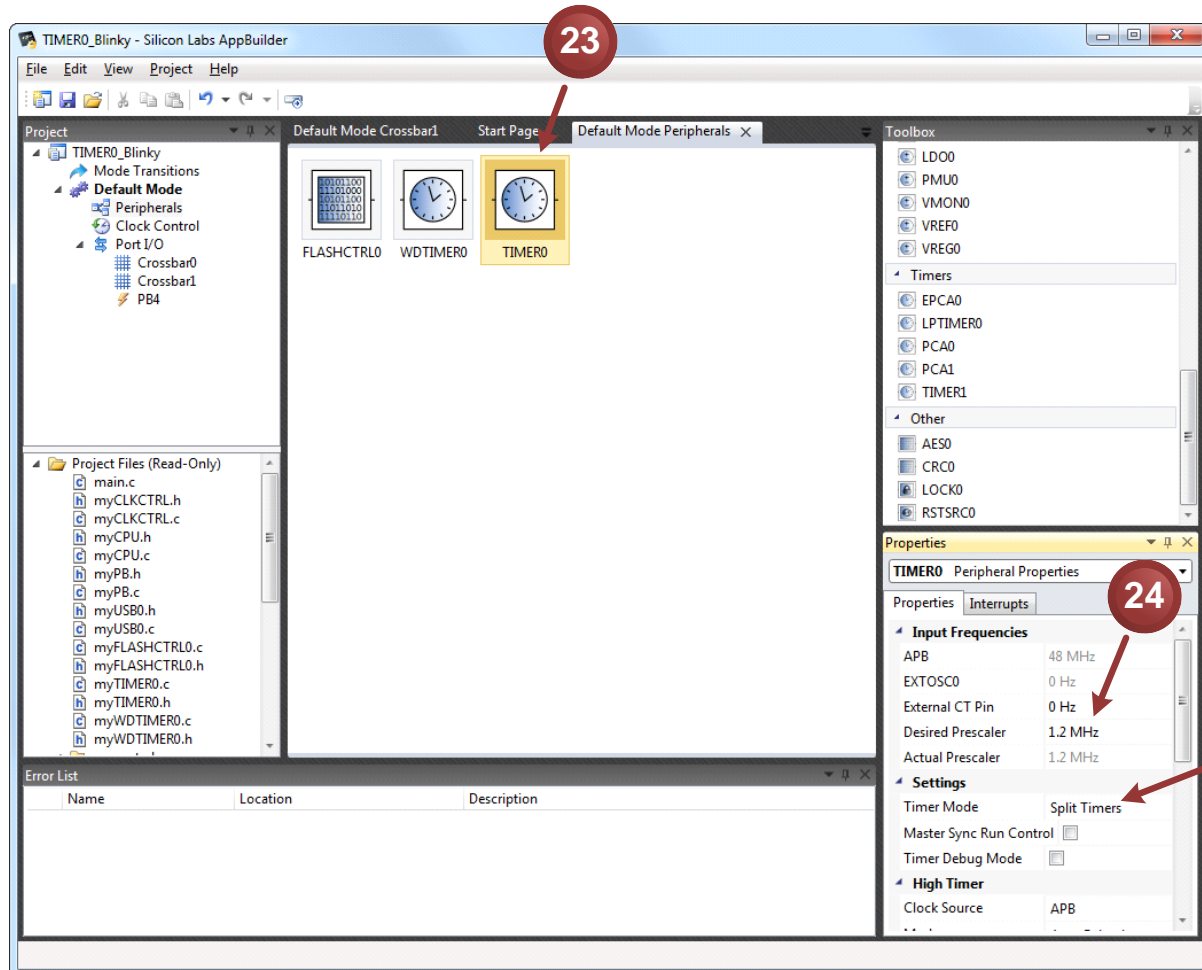
20. Double-click on the error. The program will automatically jump to the **WDTIMER0 Properties** window.
21. Disable the Watchdog Timer. The **Error List** should now be empty.



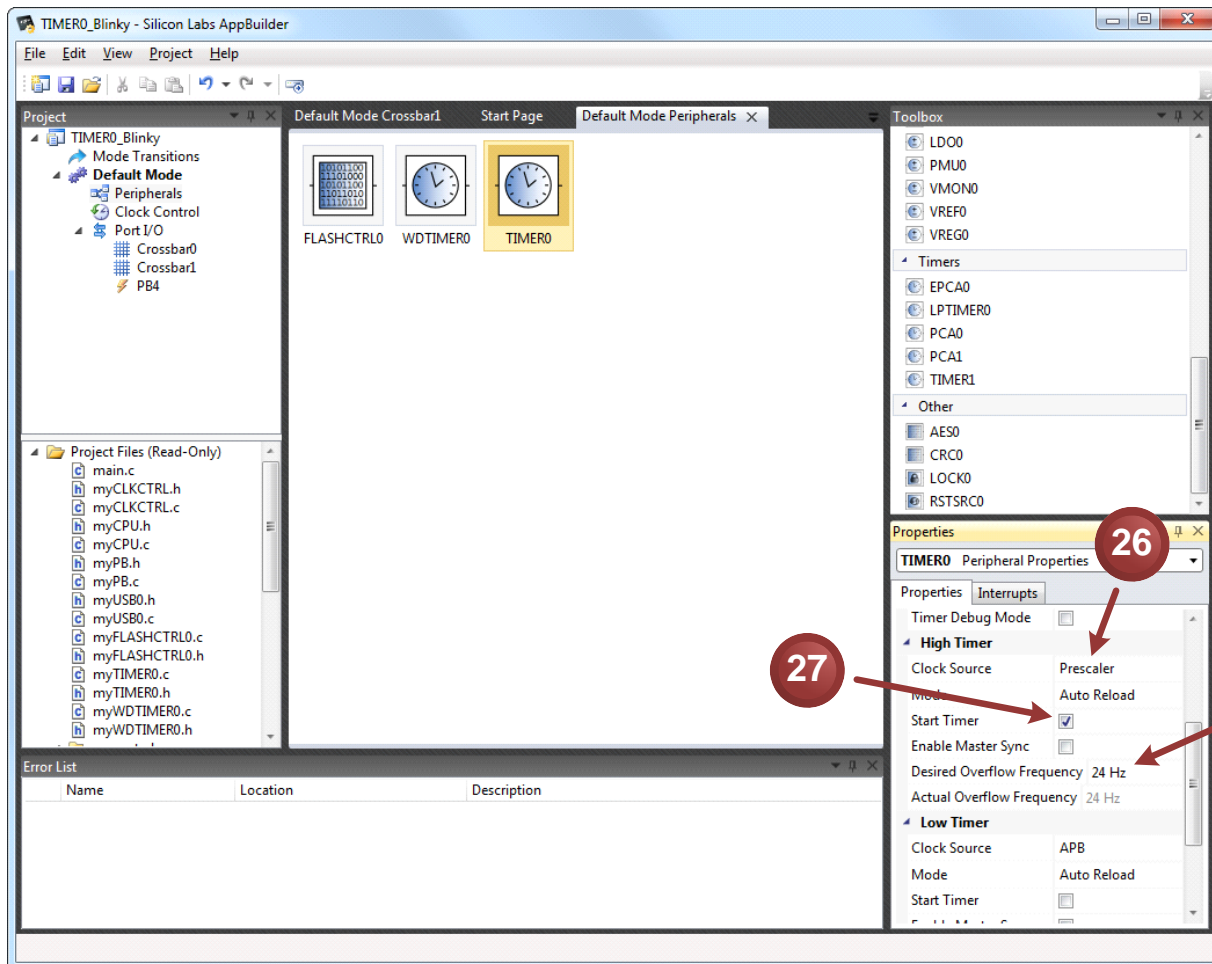
22. Drag and drop **TIMER0** to the **Peripherals Canvas**.



23. Click on TIMER0.
24. Set the **Desired Prescaler** value to **1.2 MHz**.
25. Select **Split Timers** for the **Timer Mode**.

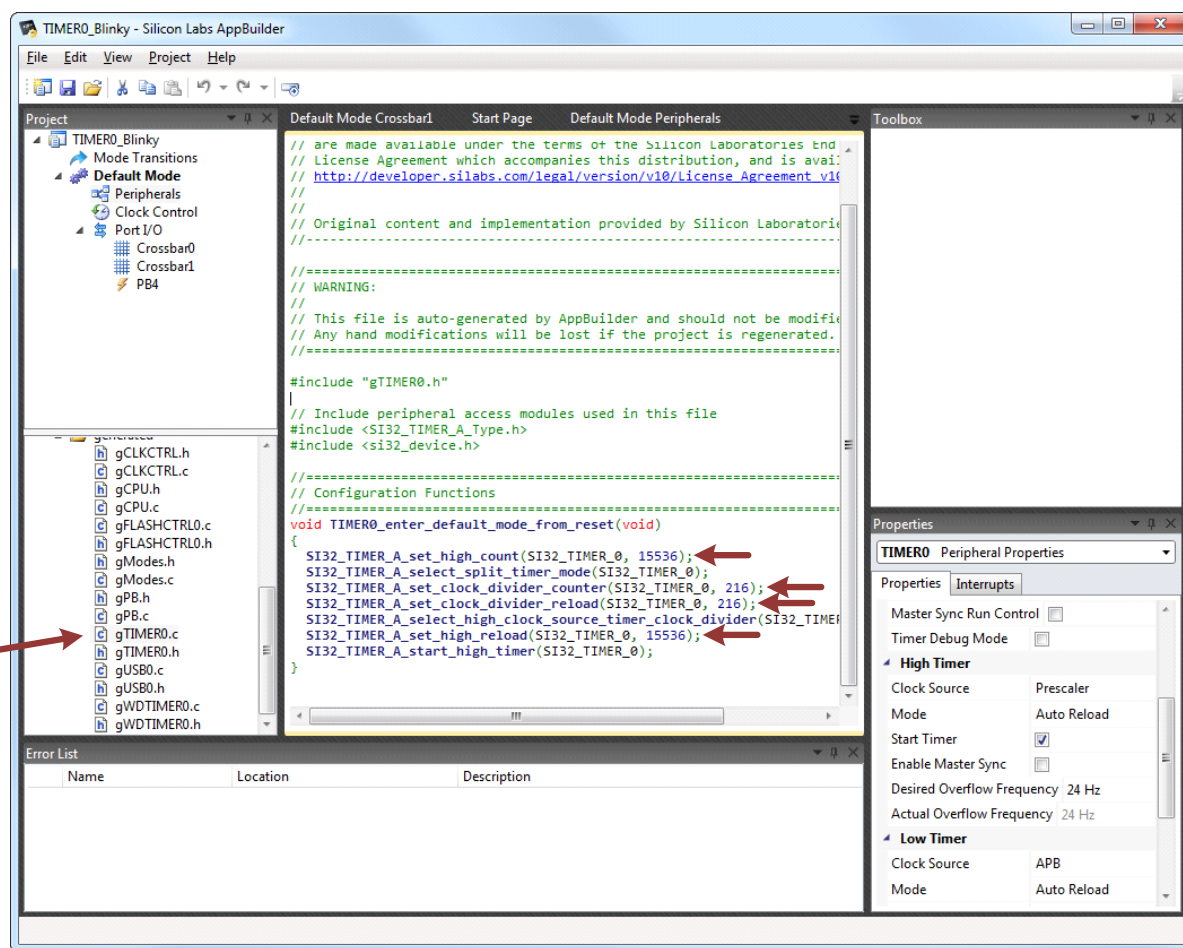


26. Select the **Prescaler** as the TIMER0 high clock source.
27. Start the high timer.
28. Set the **Desired Overflow Frequency** to **24 Hz** (12 Hz x 2 for toggling the LED on and off). TIMER0 high will be in auto-reload mode by default.

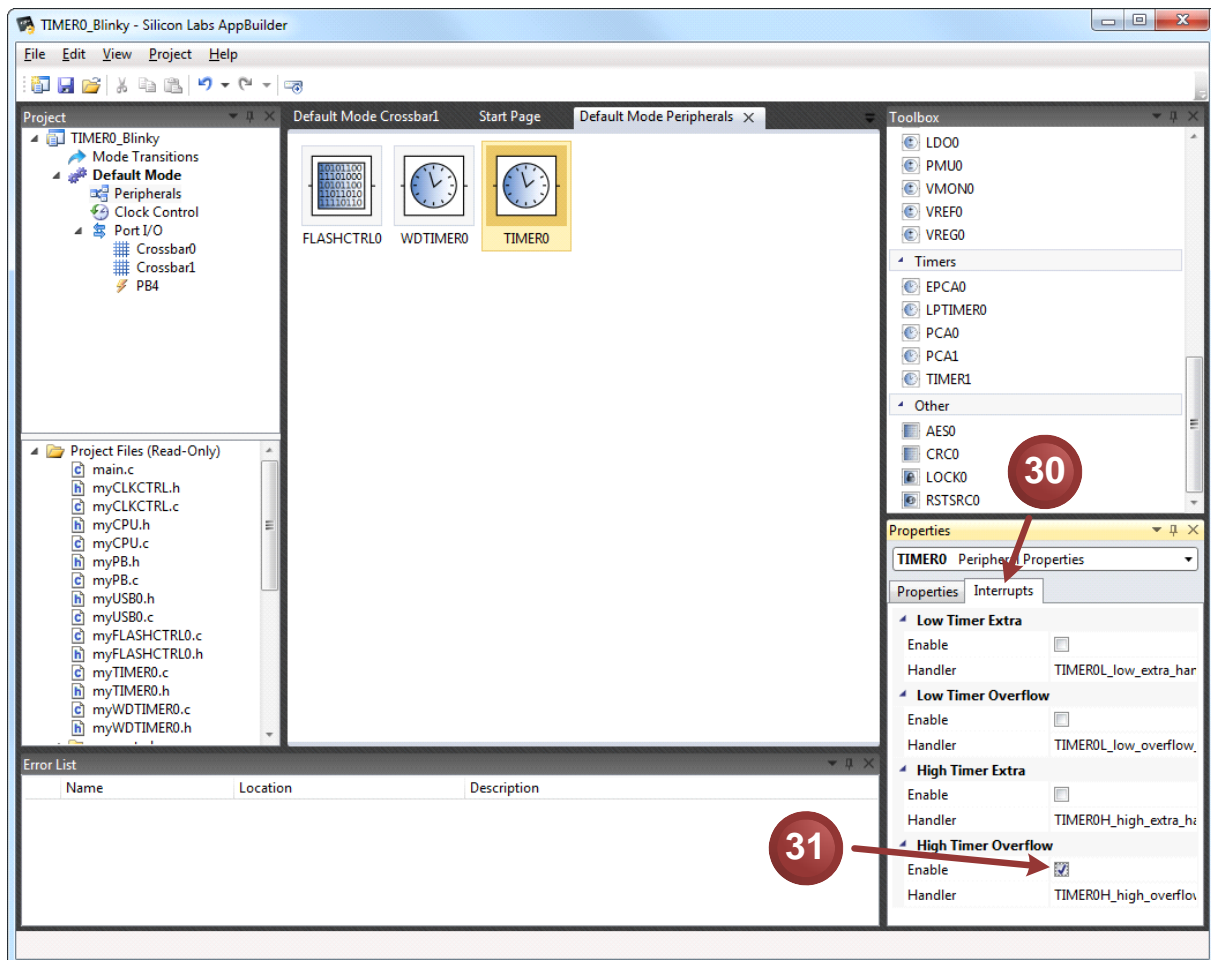




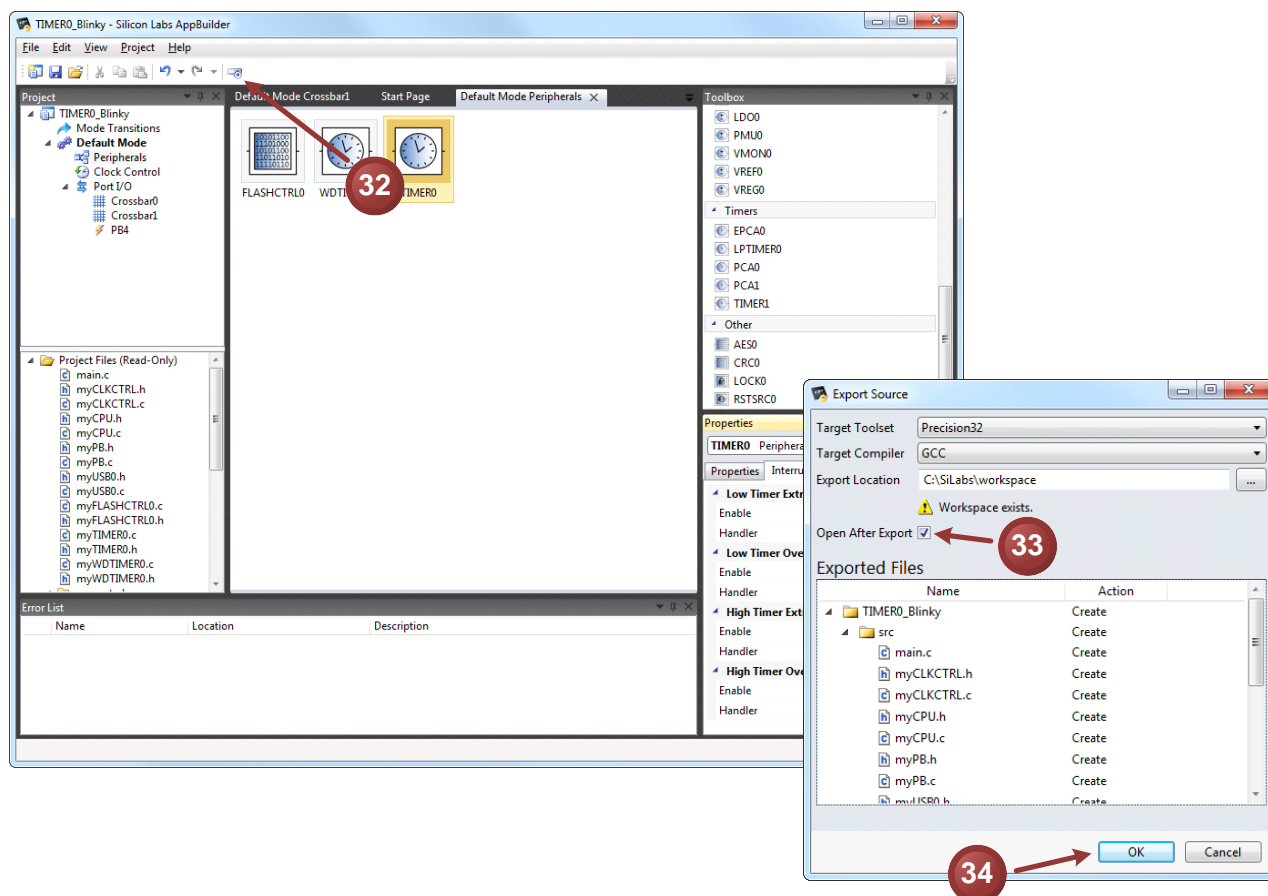
29. We can view the generated code by clicking on **gTIMER0.c** in the source files list. AppBuilder used the same values we calculated at the beginning of the lab for the prescaler (**216**) and reload values (**15536**).

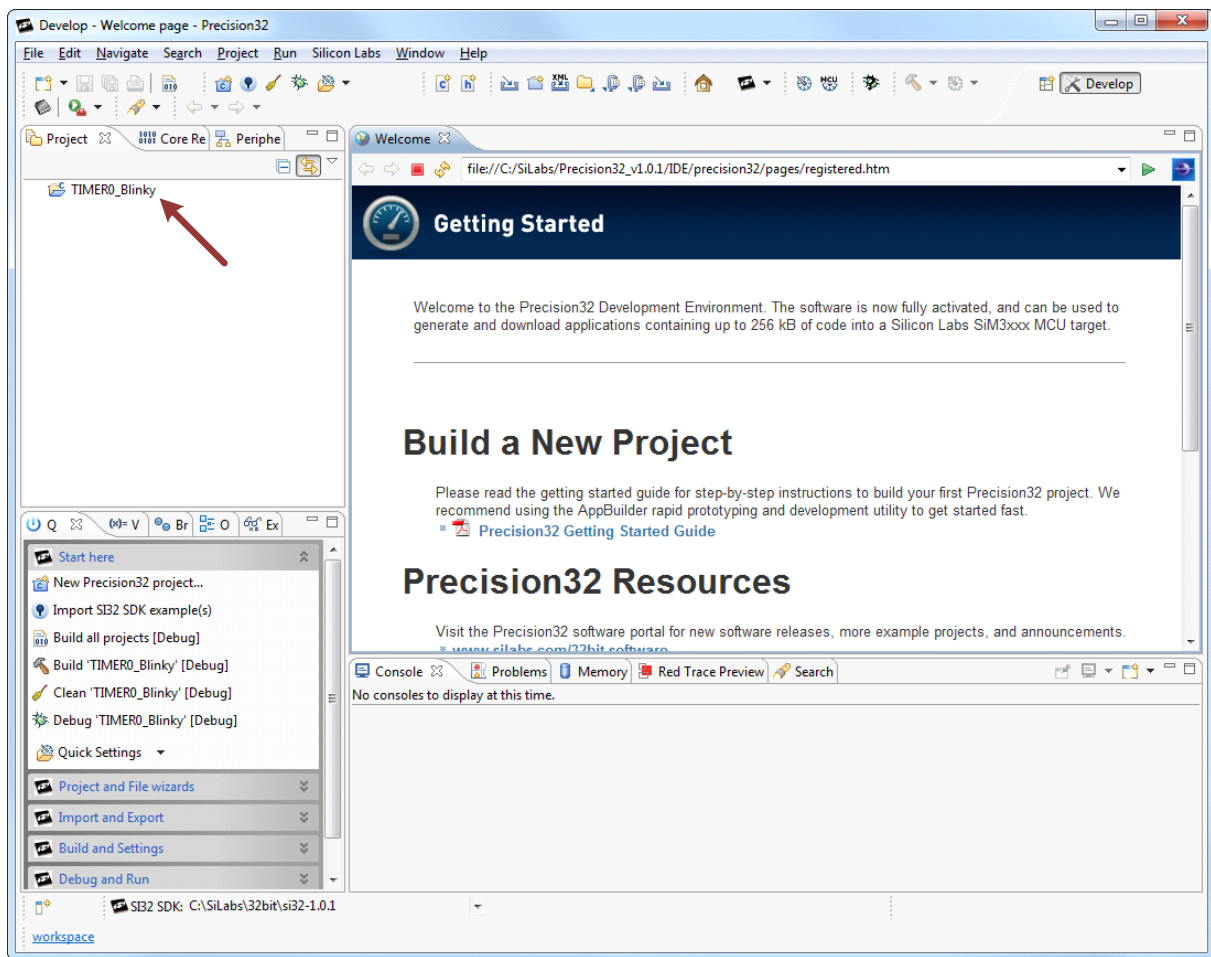


30. Click the **TIMER0 Interrupts** tab.
31. Enable **High Timer Overflow** interrupts.

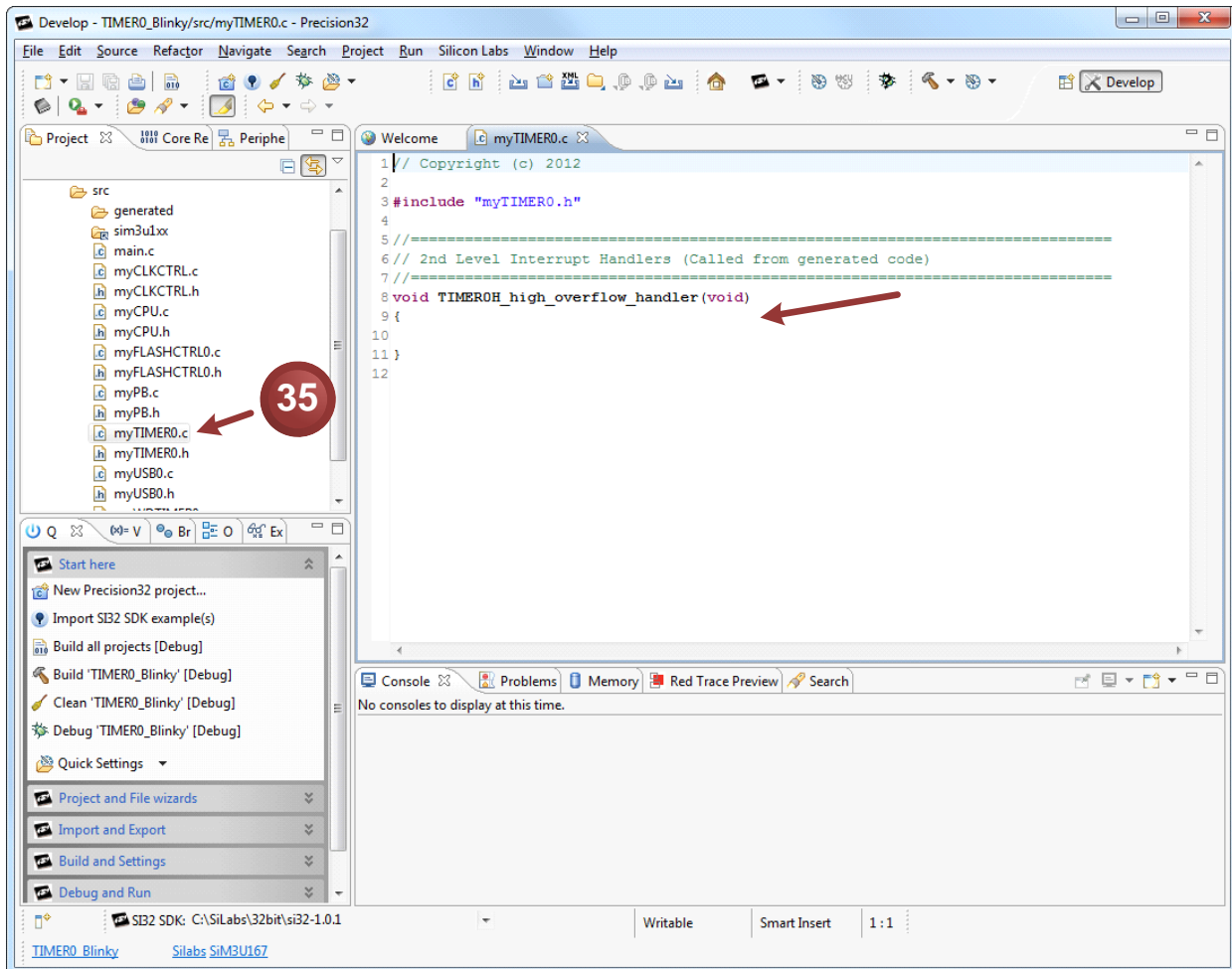


32. Click the **Export** button.
33. Verify **Open After Export** is automatically selected.
34. Press **OK**. The IDE will automatically open and have the AppBuilder project added to the workspace.





35. In the IDE, double-click on **myTIMER0.c**. The TIMER0H high overflow interrupt handler will be empty



The TIMER0 high overflow handler needs to perform two functions: 1) clear the TIMER0 high overflow interrupt flag in the TIMER0 module, and 2) toggle the PB2.10 pin.

36. Add the includes to the top of the **myTIMER0.c** file:

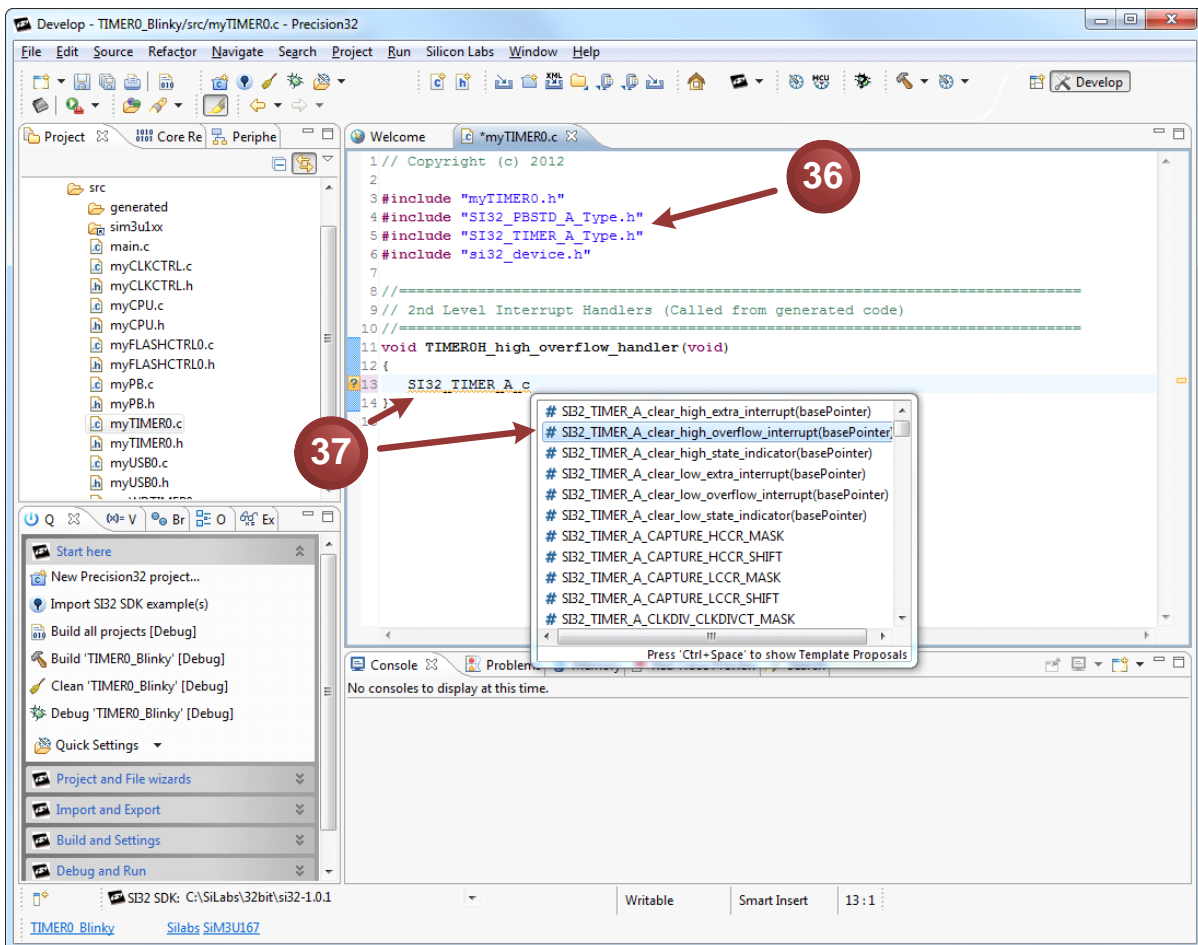
```
#include <SI32_PBSTD_A_Type.h>

#include <SI32_TIMER_A_Type.h>

#include <si32_device.h>
```

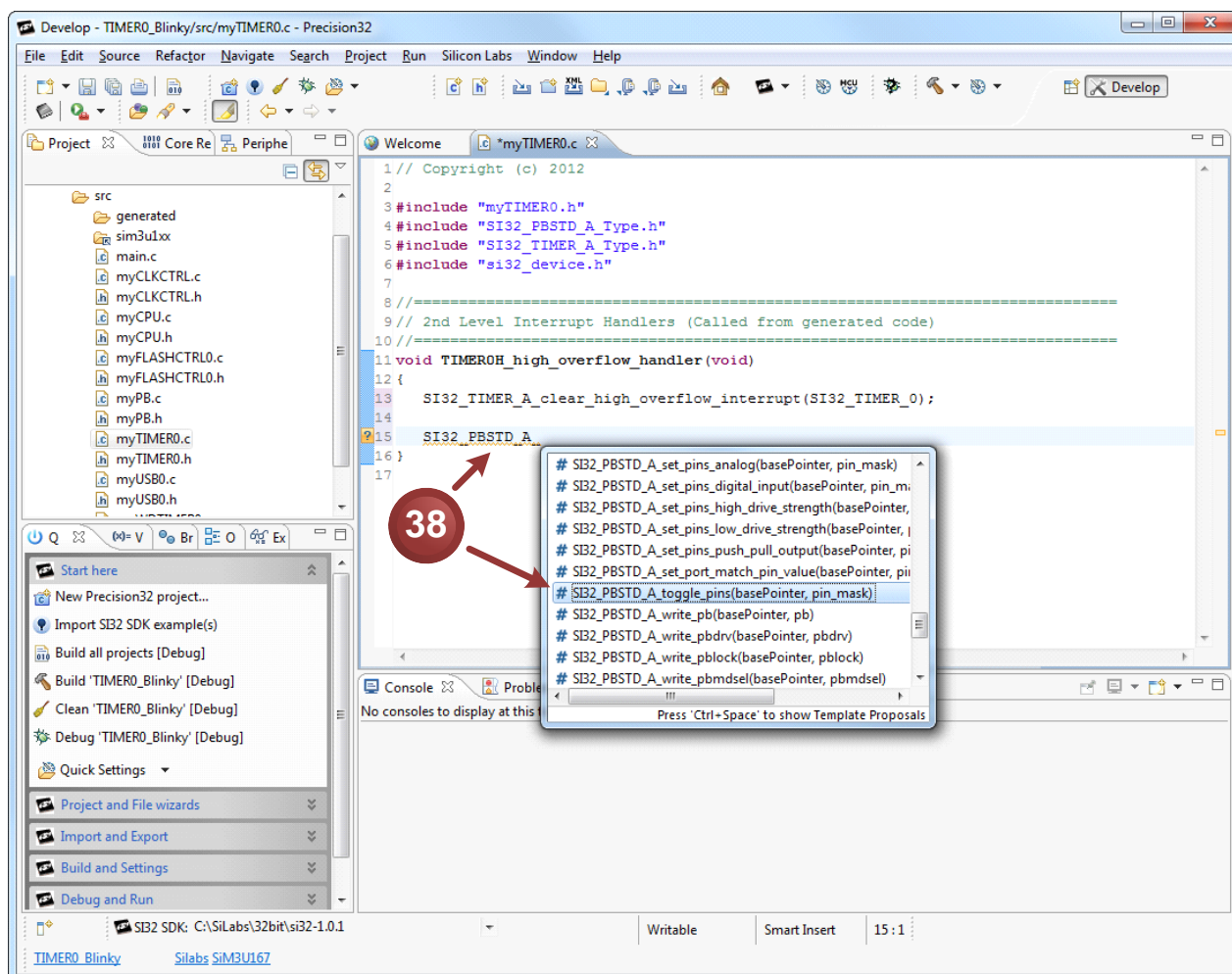
37. Because we already added the header files, we can use the auto-complete feature of the IDE to help us find the correct functions. Move the cursor inside the interrupt handler, type in “**SI32\_TIMER\_A\_c**”, and press **CTRL + SPACEBAR**. This will open a window that shows all the available defines and functions that start with SI32\_TIMER\_A\_c. The “**clear\_high\_overflow\_interrupt**” function will be the second choice. Once the correct function is selected, fill in the correct base pointer for TIMER0 (**SI32\_TIMER\_0**). The line of code should look like:

```
SI32_TIMER_A_clear_high_overflow_interrupt(SI32_TIMER_0);
```



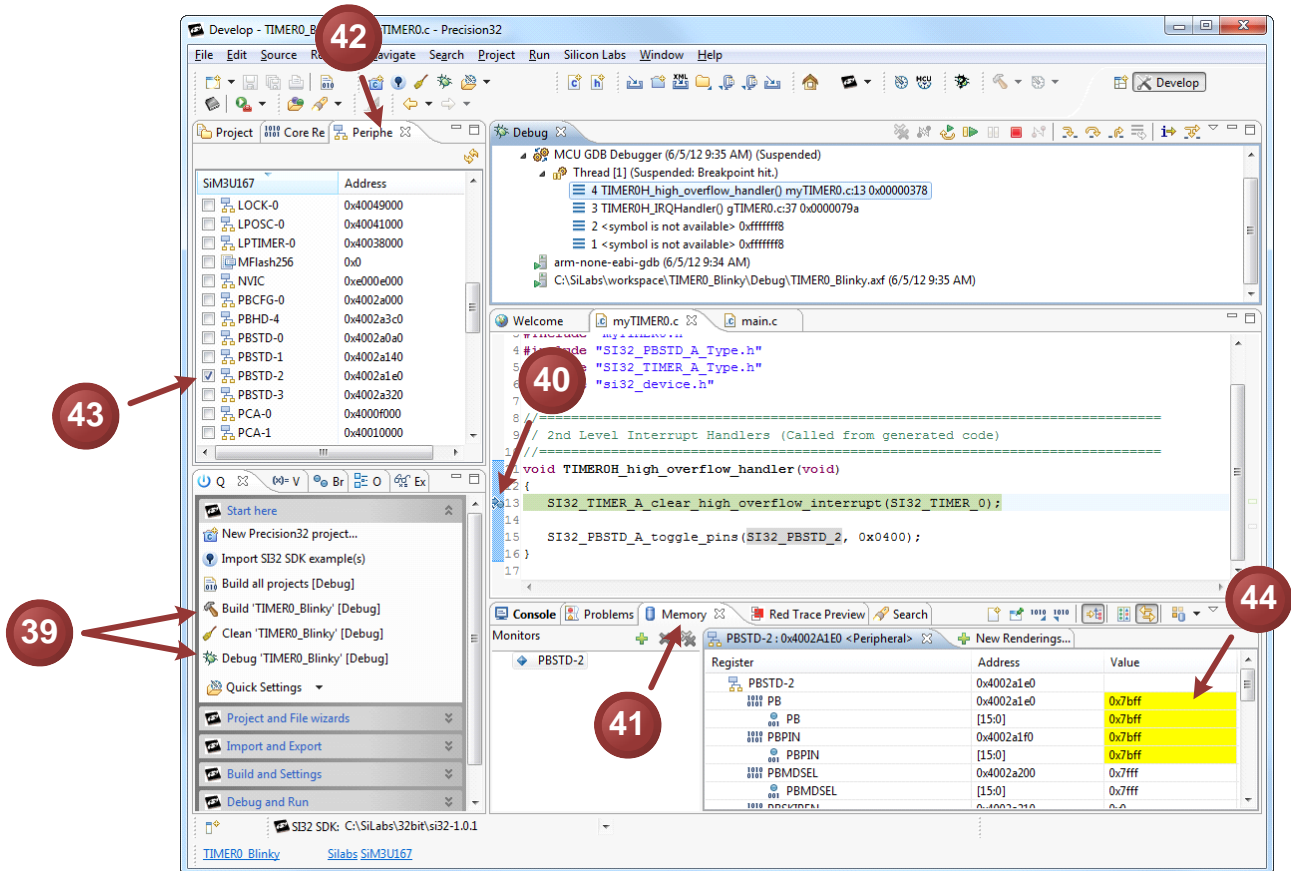
38. We can use the same process to enter in the second function, but we'll search for it in the full list this time. Move to a new line, type in "**SI32\_PBSTD\_A**", and press **CTRL + SPACEBAR**. The list will show all functions and defines for the PBSTD A module. We know we want a function that will toggle the pin, so look for a function of the appropriate name (**toggle\_pins**). Once the correct function is selected, fill in the correct base pointer for PB2 (**SI32\_PBSTD\_2**) and the correct mask for PB2.10 (**0x0400**). The line of code should look like:

```
SI32_PBSTD_A_toggle_pins(SI32_PBSTD_2, 0x0400);
```





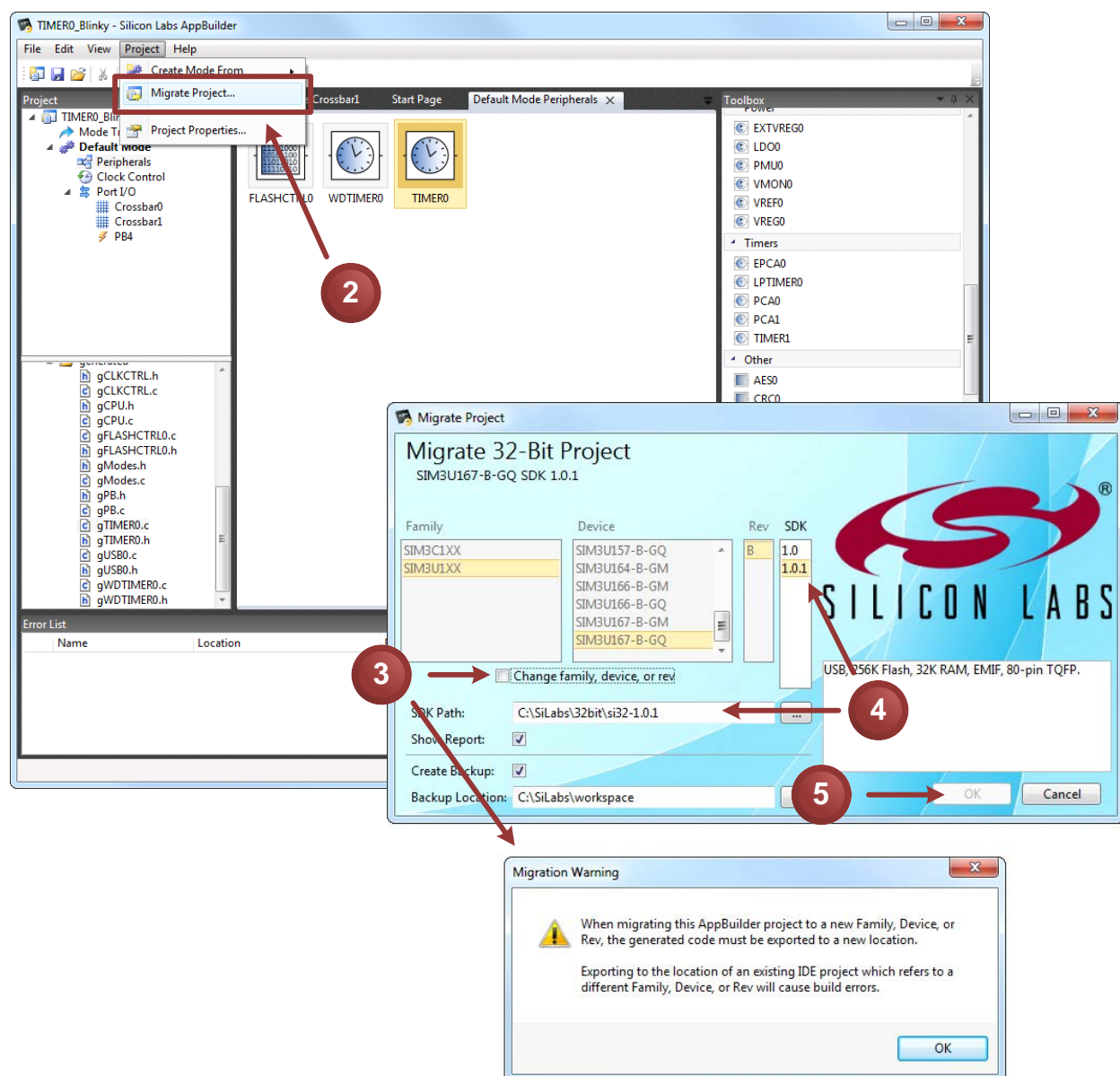
39. Build the code and start a debug session. Then start the program. The LED will blink at a 12 Hz rate.
40. Add a breakpoint in the `TIMER0` handler in `myTIMER0.c`.
41. Select the **Memory** view.
42. Select the **Peripheral** view.
43. Open the PB2 register window by selecting **PBSTD-2** in the **Peripheral** view.
44. Pressing the start program button repeatedly will update the PB2 pin value in the register window.



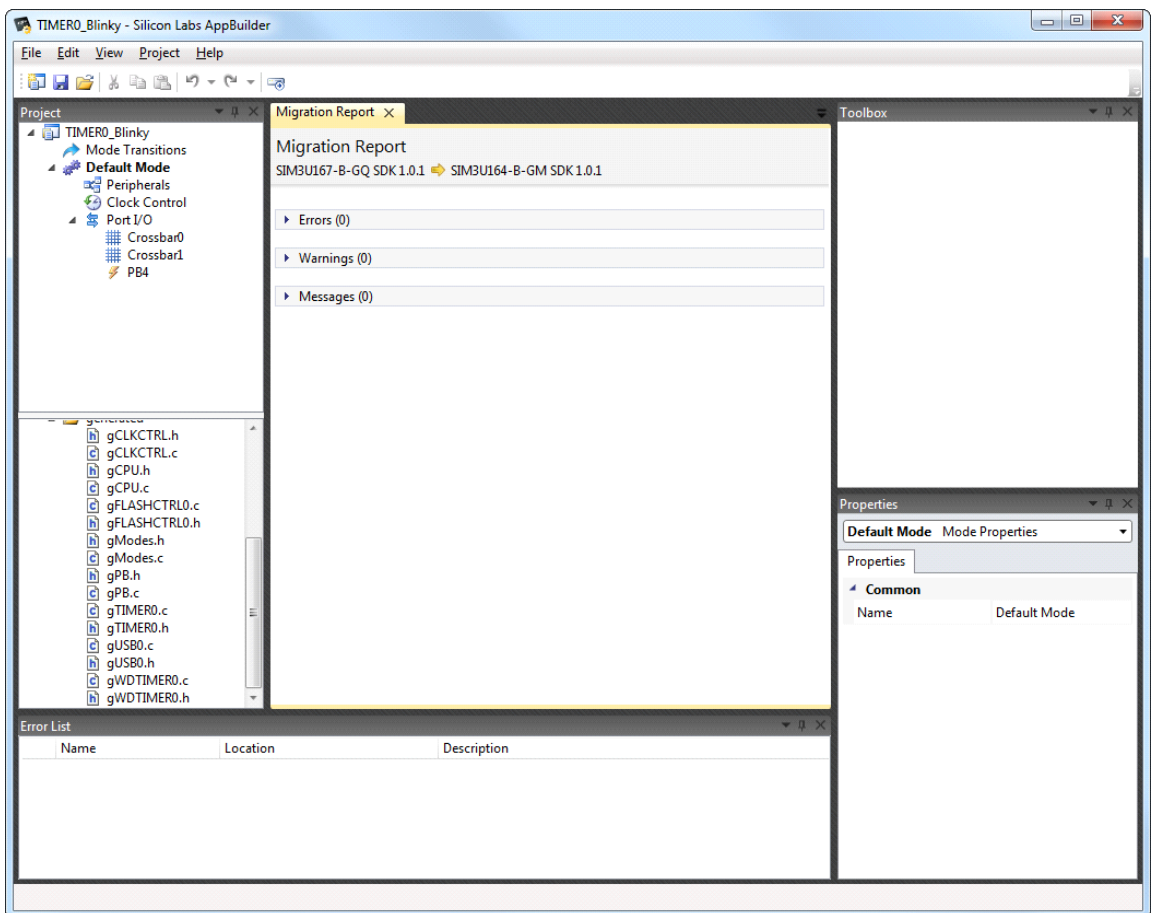
## 6.4. Additional Notes: Migrating a Project

The AppBuilder migrate feature allows a project switch to a different version of the SDK, a different device family, or device within the same family. To migrate a project:

1. Open the project in **AppBuilder 1.0.1**.
2. Go to the **Project**→**Migrate Project...** menu selection.
3. Check the **Change family, device, or rev** checkbox to change the device family, device within the selected family, or the revision of the device. This will prompt a message window that requests a new export location.
4. Changes to the SDK version do not require a new export location.
5. Make any other desired modifications to the Migrate Project dialog and press **OK**.

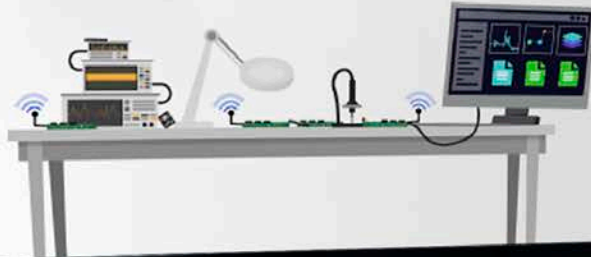


If the report option was not disabled, AppBuilder will generate a report of the migration operation. This report will output a summary of the migration operation (from→to) and any potential Errors, Warnings, or Messages.



Silicon Labs

# Simplicity Studio™4



## Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!



**IoT Portfolio**  
[www.silabs.com/IoT](http://www.silabs.com/IoT)



**SW/HW**  
[www.silabs.com/simplicity](http://www.silabs.com/simplicity)



**Quality**  
[www.silabs.com/quality](http://www.silabs.com/quality)



**Support and Community**  
[community.silabs.com](http://community.silabs.com)

### Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Labs shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

### Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR®, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOModem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress® and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.  
400 West Cesar Chavez  
Austin, TX 78701  
USA

<http://www.silabs.com>