Lab Objectives

1. Generate a project using the GR-Sakura web compiler
2. Edit/Compile/Build/Debug the project using the web compiler

Lab Materials

Please verify you have the following materials at your lab station.

- GR-Sakura
- Laptop
- Internet Access

Skill Level

1. Introductory course
2. Base knowledge of C programming

Time to Complete Lab

60 Minutes

Lab Sections

1 Setting up an account with MyRenesas....2
2 Lab #1 - Getting Started with GR-Sakura....3
3 Lab #2 - Modifying the previous code....8
4 Lab #3 - Building a new project....10
5 Lab #4 – Debugging over serial....12
6 Appendix....16

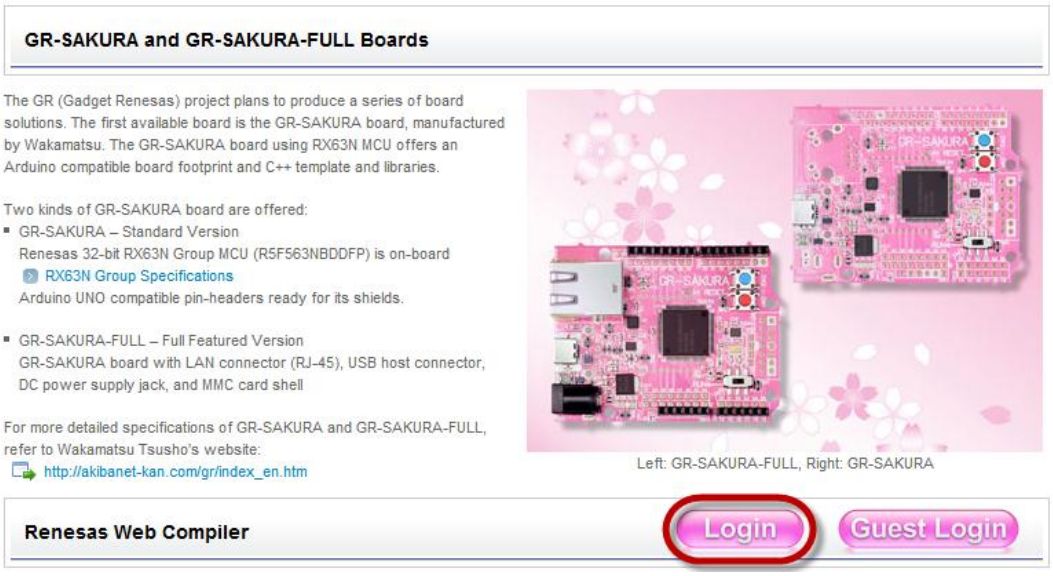
1 Setting up an account with MyRenesas

Overview:

This lab section will show you how to register with MyRenesas so your files can be properly preserved.

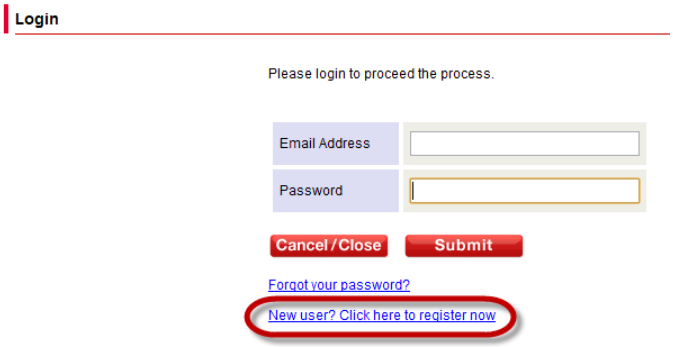
Procedural Steps

Step 1.1 Go to the web-compiler which can be found here: http://www.renesas.com/products/promotion/gr/index.jsp or at http://tool-cloud.renesas.com/ to log on as a guest. Please open the desired webpage up on Firefox and not IE8 or Chrome.



Step 1.2 You have the ability to login as a registered user or as a guest. One downside with a guest login is that your workspace will not be saved between logins. If you have never registered with this site please go to section 4 "Setting up and account with MyRenesas" and return here when finished.

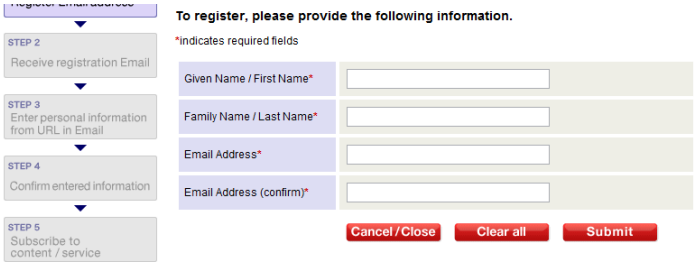
Step 1.3 After clicking on “Login” click on register when prompted to login.



Step 1.4 Choose your preferred language when the next window appears.



Step 1.5 Fill in the required information on the next screen as seen.



Step 1.6 An E-mail will be sent to the desired E-mail address you used. This will require you to have access to that e-mail that way you can access the confirmation e-mail.

Step 1.7 Once you receive this e-mail, click on the link within the e-mail that will allow you to finish setting up your account such as password, address, and further information.

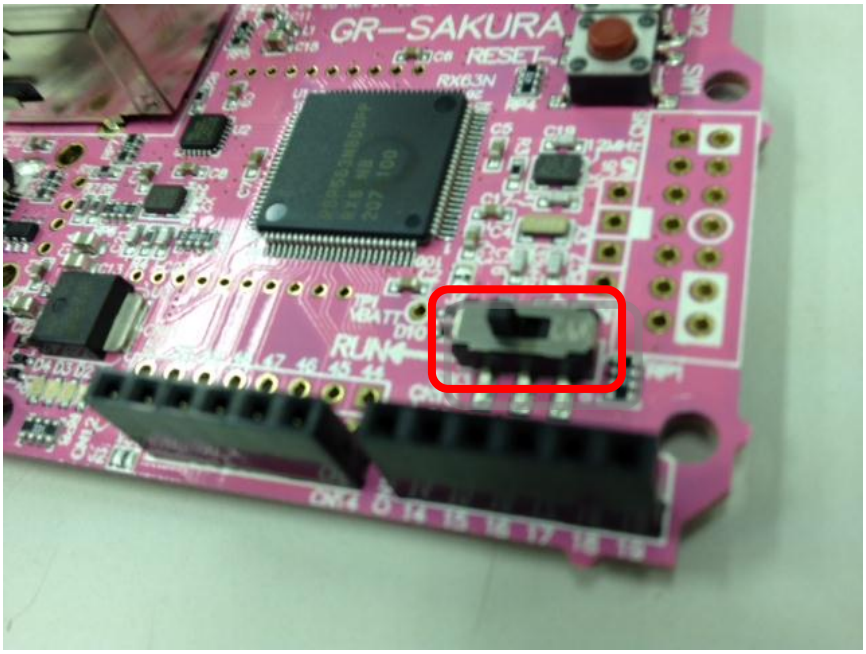
2 Lab #1 - Getting Started with GR-Sakura

Overview:

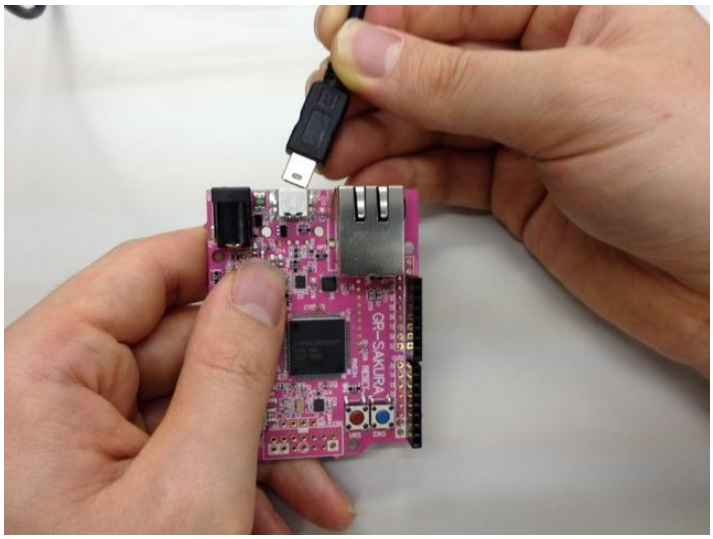
This lab section will show you how to properly start and compile your first ever project on the newly designed development board known as “GR-Sakura”. This will be completed by registering with the GR-Sakura web compiler’s webpage along starting your first project.

Procedural Steps

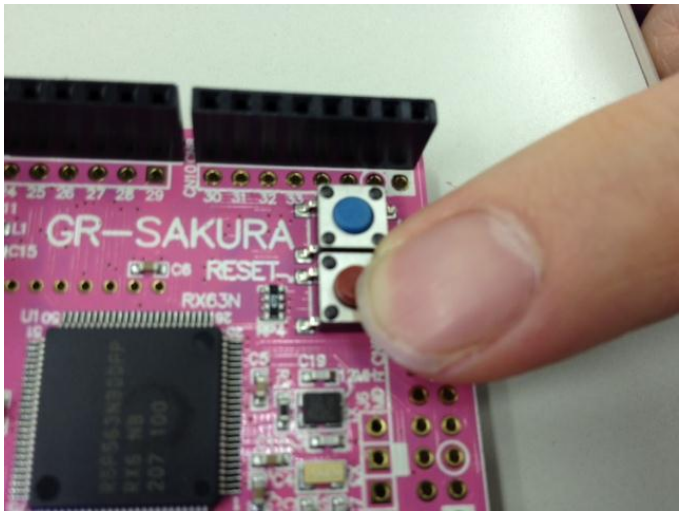
Step 2.1 Make sure that SW3 is in “run mode”



Step 2.2 Connect the GR-Sakura board to your computer with a USB to mini-USB cable.



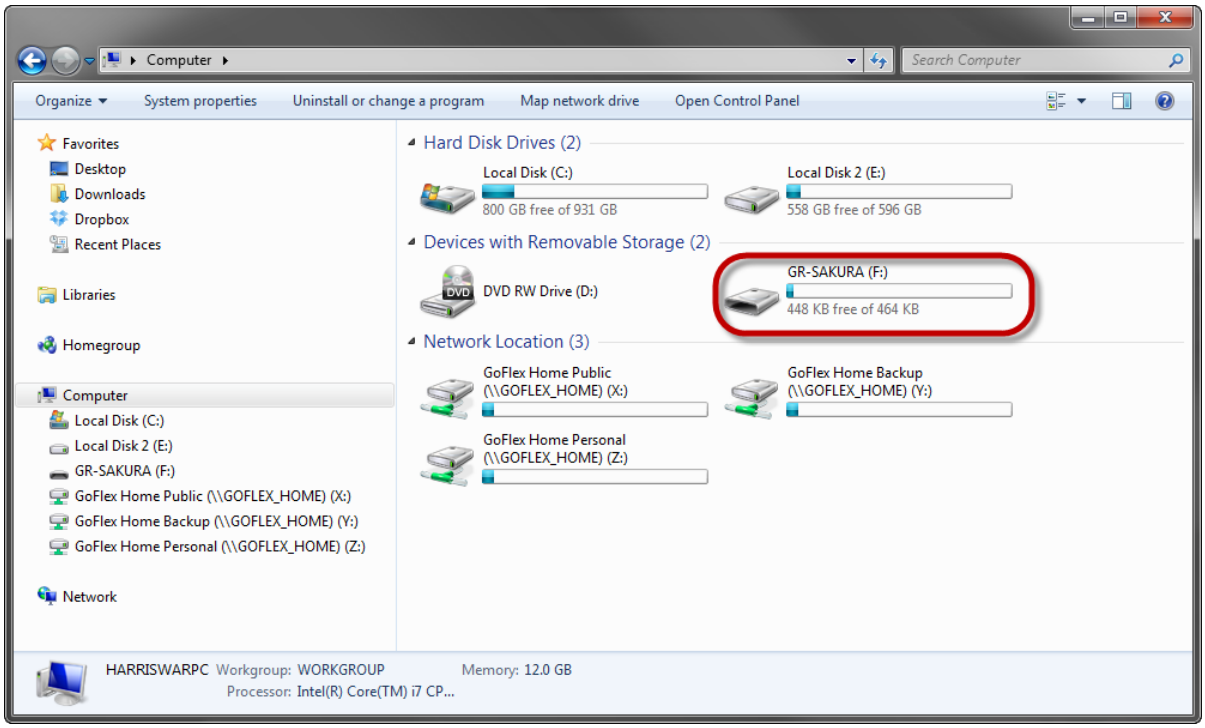
Step 2.3 Press the red switch to reset the board.



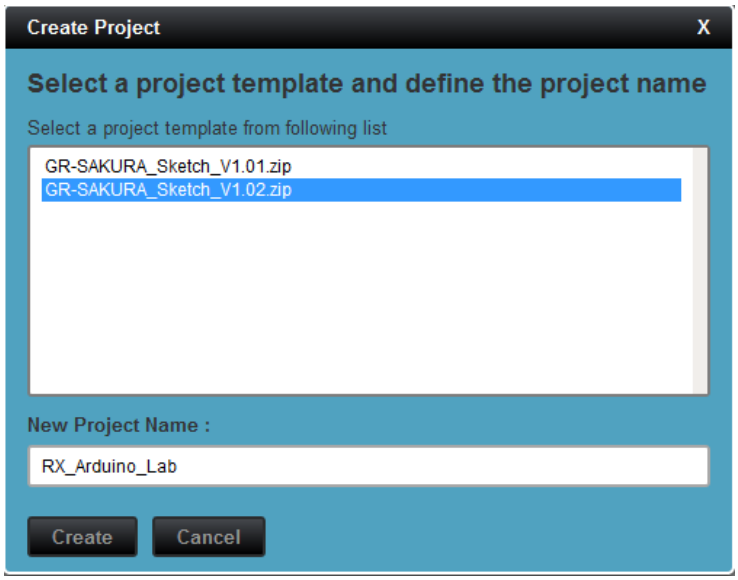
Step 2.4 The boards LEDs should be lit up blue; this indicates that the board is ready for programming.

Step 2.5 Go to “computer” by clicking Start > Computer. You should see the device show up here, if the device does not show up it means that either the driver has not finished installing or your system does not support this hardware.

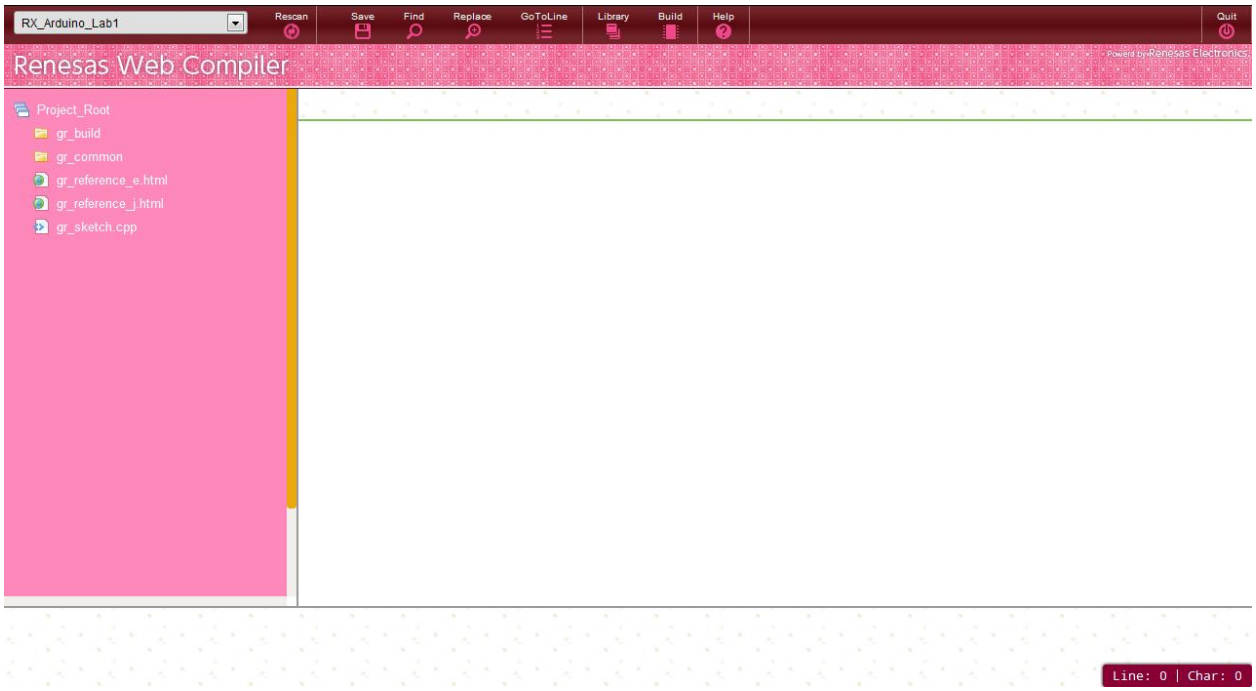
If the GR-Sakura board does not show up within the drive window as seen below, please download the serial drivers from Moodle and update the Sakura drivers.



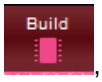
Step 2.6 If it is your first time logging in a window will pop-up asking you to create a project. You will need to select a “template” and a project name as seen below. The project name can be anything that you desire.



Step 2.7 Your window should now look like this, if it doesn’t please try to recreate a project.

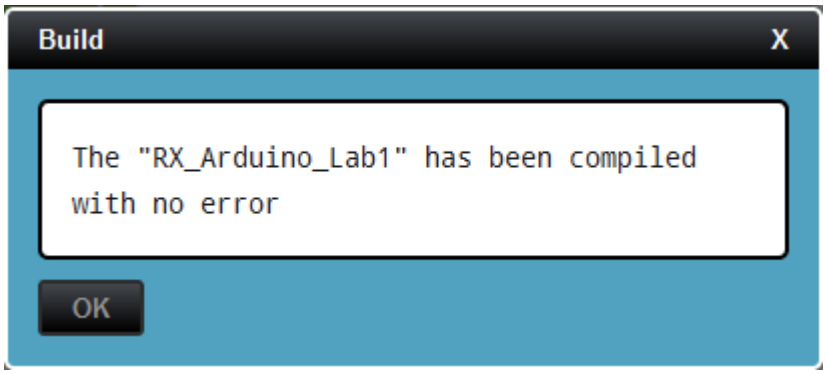


Step 2.8 The project comes preloaded with a “test” program. This program is designed to blink the LED’s on the board in a wave method. To open the project, double click on “gr_sketch.cpp” on the left panel of the web compiler.

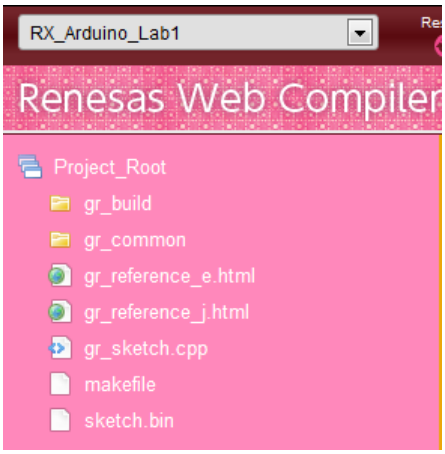
Step 2.9 With the file open click on the build function, 

, in the top bar of the web compiler to build the project.

Step 2.10 The project should build without errors. When the window below pops up go ahead and click ok.



Step 2.11 Your project tree in the left panel has now changed to:



Step 2.12 In the bottom of the window you will notice the compiler reports all the web tool operations (GNU Compiler, Linker, Make process). This will look like the snippet below.

```
rx-elf-gcc -Wall -g -O2 -I./gr_build -I./gr_common -I./gr_common/include -
I./gr_common/include/rxduino -I./gr_common/include/tkdnhal -I./gr_common/lib -c
gr_sketch.cpp -o gr_sketch.o
rx-elf-gcc -Wall -g -O2 -I./gr_build -I./gr_common -I./gr_common/include -
I./gr_common/include/rxduino -I./gr_common/include/tkdnhal -I./gr_common/lib -c
gr_common/intvect.c -o gr_common/intvect.o
rx-elf-gcc -Wall -g -O2 -I./gr_build -I./gr_common -I./gr_common/include -
I./gr_common/include/rxduino -I./gr_common/include/tkdnhal -I./gr_common/lib -c
gr_common/lowlevel.c -o gr_common/lowlevel.o
gr_common/lowlevel.c: In function 'fstat':
gr_common/lowlevel.c:128:3: warning: overflow in implicit constant conversion
[-Woverflow]
```

```
gr_common/lowlevel.c:129:3: warning: large integer implicitly truncated to
unsigned type [-Woverflow]
gr_common/lowlevel.c:134:3: warning: overflow in implicit constant conversion
[-Woverflow]
rx-elf-ld -
Map ./gr_build/sketch.map ./gr_common/gstart.o ./gr_sketch.o ./gr_common/intvec
t.o ./gr_common/lowlevel.o ./gr_common/lib/libc.a ./gr_common/lib/libg.a ./gr_c
ommon/lib/libgcc.a ./gr_common/lib/libm.a ./gr_common/lib/librxduino.a ./gr_com
mon/lib/libsim.a ./gr_common/lib/libstdc++.a ./gr_common/lib/libsupc++.a ./gr_c
ommon/lib/libtkdnhal.a ./gr_common/lib/libtkdnip.a -T ./gr_common/gr_sakura.gsi
-L./gr_common/lib/ -lrxduino -lstdc++ -lsupc++ -lc -lsim -lgcc -lm -ltkdnhal -
nostartfiles -o sketch.elf
C:/PROGRA~2/Renesas/Hew/Tools/KPIT/GNURX-~1/v12.01/rx-elf/bin/rx-elf-ld.exe:
warning: section `.bss' type changed to PROGBITS
rx-elf-objcopy -O binary sketch.elf sketch.bin
rm -f *.o
rm -f sketch.elf
Make process completed
```

Step 2.13 The “sketch.bin” file is the file that has just been created by the compiler. Right click on this file and select “download files”.

Step 2.14 Find your downloaded file, it should be located in the “downloads” folder within “computer” or at the bottom of the internet browser in the download bar by default. Drag and drop this file over to the “GR-Sakura” drive folder.

3 Lab #2 - Modifying the previous code

Overview:

This lab section will show you how to modify and add code to the previous project that was created in section one of this lab.

Procedural Steps

Step 3.1 Reset the board at this point by pressing the red button to remove the previous sections code. This will allow you to load a new program to the board.

Step 3.2 Within the "gr_sketch.cpp" file modify the code to allow the onboard switch to interface with the board. Add the code that is labeled with "//add this line"

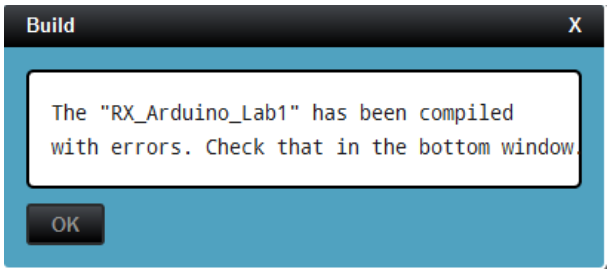
```
void setup()
{
    pinMode(PIN_LED0,OUTPUT);
    pinMode(PIN_LED1,OUTPUT);
    pinMode(PIN_LED2,OUTPUT);
    pinMode(PIN_LED3,OUTPUT);
    pinMode(PIN_SW,INPUT);          // <= add this line
}
```

Step 3.3 While in the same file modify the void loop function to include:

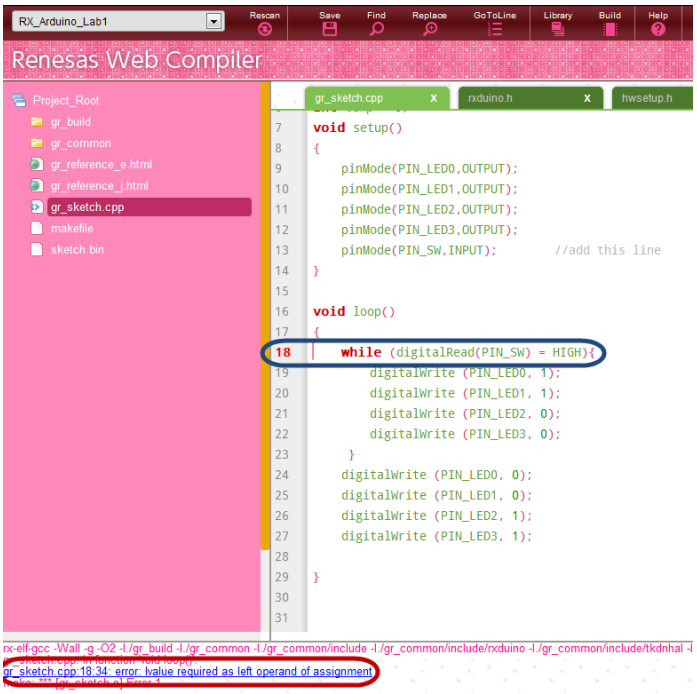
```
void loop()
{
```

```
        while (digitalRead(PIN_SW) = HIGH){
          digitalWrite (PIN_LED0, 1);
          digitalWrite (PIN_LED1, 1);
          digitalWrite (PIN_LED2, 0);
          digitalWrite (PIN_LED3, 0);
        }
        digitalWrite (PIN_LED0, 0);
        digitalWrite (PIN_LED1, 0);
        digitalWrite (PIN_LED2, 1);
        digitalWrite (PIN_LED3, 1);
}
```

Step 3.4 Attempt to rebuild the desired the newly developed code as described previously. The code that you have generated should cause an error with the Sakura web compiler like the one seen below.



Step 3.5 Click on the "OK" button within this window to get it to close. Once the window is closed go to the bottom of the web compiler and click on the blue highlighted link: "gr_sketch.cpp:18:34: error: lvalue required as left operand of assignment".



What you should notice is that after you click on the link, the number "18" or the line that contains "digitalRead(PIN_SW) = HIGH)" becomes **bolded**. This is because this is where the compiler has determined that an error resides within the code.

Step 3.6 To correct the error within the code modify the while statement to:

```
while (digitalRead(PIN_SW) == HIGH){
```

Step 3.7 Rebuild the code an load it onto the Sakura. With the code properly loaded to the board you should notice that when switch 2, the blue button, is pressed the left two blue LED's should light up and the other two should light up when the button is released.

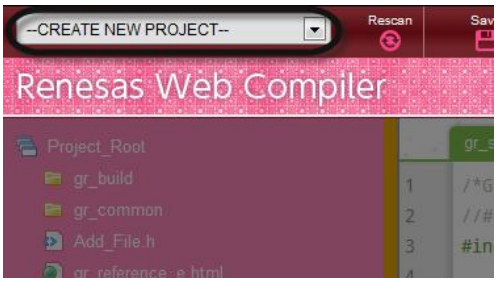
4 Lab #3 - Building a new project

Overview:

This lab section will show you how to create a new project and how to import a user defined *.h file.

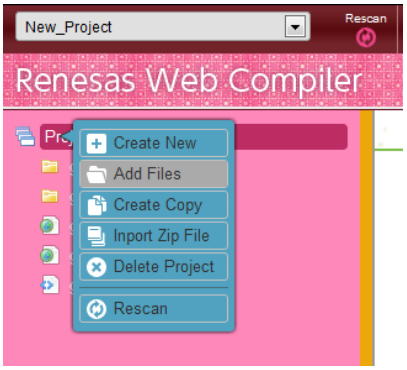
Procedural Steps

Step 4.1 In the upper left hand corner of the Web compiler in the drop down bar change the selection to "-CREATE NEW PROJECT-"



Step 4.2 Select the template that you would like to use, "GR-SAKURA_Sketch_v1.02.zip" and a project name. Once that is completed click Create.

Step 4.3 With the new project created, right-click on the Project_Root file within the project tree and select "Add Files"



Step 4.4 With the next pop-up click on “Open” and browse to the directory where you downloaded the “Add_File.h” from Moodle and select it. This will allow for a new *.h file to be uploaded to the Web Compiler. You should see the file in your project tree if it was added correctly.

Step 4.5 Go ahead and double click on the "gr_sketch.cpp" and the "Add_File.h" to view the code that is within the two files. To include the "Add_File.h" in the project build you must include it within the "gr_sketch.cpp" file by adding a line of code after #include <rxduino.h>

```
#include "Add_File.h"
```

Step 4.6 In the "gr_sketch.cpp" file remove the code that is in the setup loop and replace it with the code shown below:

```
int count = 0;                  //add this line
int sw2Pressed = 0;             //add this line

void setup()
{
    initializeLEDs();           //Replace lines with
}
```

Step 4.7 Add the following lines to the void loop function:

```
void loop()
{
    while(ButtonPress()){               //Replace lines with
        if (sw2Pressed == 1){           //Replace lines with
            sw2Pressed = 0;             //Replace lines with
            count = count + 1;          //Replace lines with
            if (count>= 16){            //Replace lines with
                count= 0;               //Replace lines with
            }                           //Replace lines with
            BinaryCountDisplay(count); //Replace lines with
        }                               //Replace lines with
    }                                   //Replace lines with
    sw2Pressed = 1;                     //Replace lines with
}
```

Step 4.8 Build the project like normal; there shouldn't be an error with this build, if there is please go through the code you have added to insure that it matches exactly. With the project built go ahead and load it to the GR-Sakura as described earlier.



What you should observe is that every time you press switch 2 the four LED's act like a binary counter and keep track of how many times you have pressed the button.

5 Lab #4 – Debugging over serial

Overview:

In this lab section we will look into alternative ways of debugging projects. In this alternative method we will utilize a virtual UART communication to determine any errors/mistakes that are within the code.

Procedural Steps

Step 5.1 Even though the LEDs are a great way to see how the board is properly working sometimes you have to debug the system. The easiest way to do this with a device that doesn't have a built in debug function is to set-up a serial communication with the board. To do this add the following line to the setup function within "gr_sketch.cpp" file. This added line of code will initialize the virtual COM port on the RX63N at a baud rate of 38400.

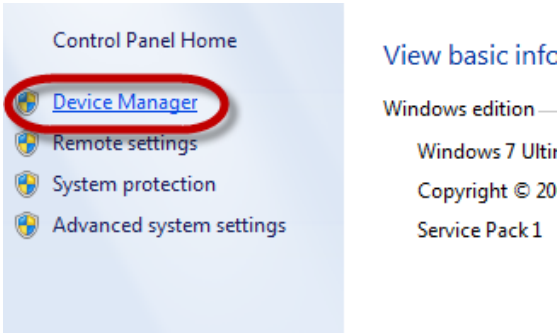
```
Serial.begin(38400);
```

Step 5.2 Then within the loop function within the same file add the following line. This line of code will print out a line of the variable count in a binary format.

```
      If (count >= 16){
            count= 0;
            }
            BinaryCountDisplay(count);
            Serial.println(count,BIN);          // <= add this line
        }
```

Step 5.3 Build, download, and load the program to the GR-Sakura board to test out the newly modified code.

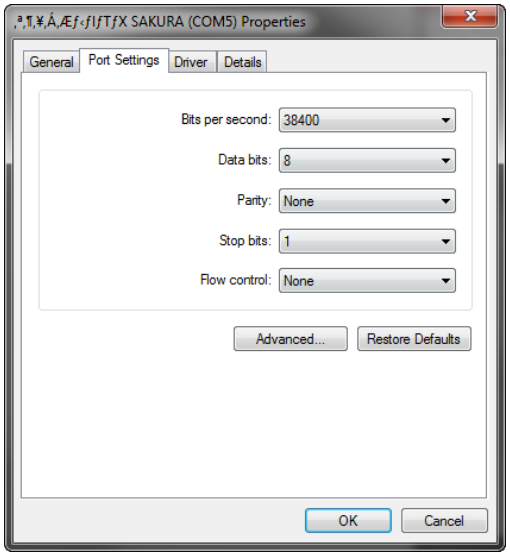
Step 5.4 Once the code is properly loaded to the board a driver should automatically install on the computer. This driver is for the virtual COM port. To verify the COM's number please go to the device manager by clicking start then right-clicking on computer and selecting properties. Once in the system properties window in the upper left-hand corner select "Device Manager"



Step 5.5 Once the device manager is up, in the device tree select "Ports (COM & LPT)" and right click on the COM device that ends with "SAKURA" and select properties.

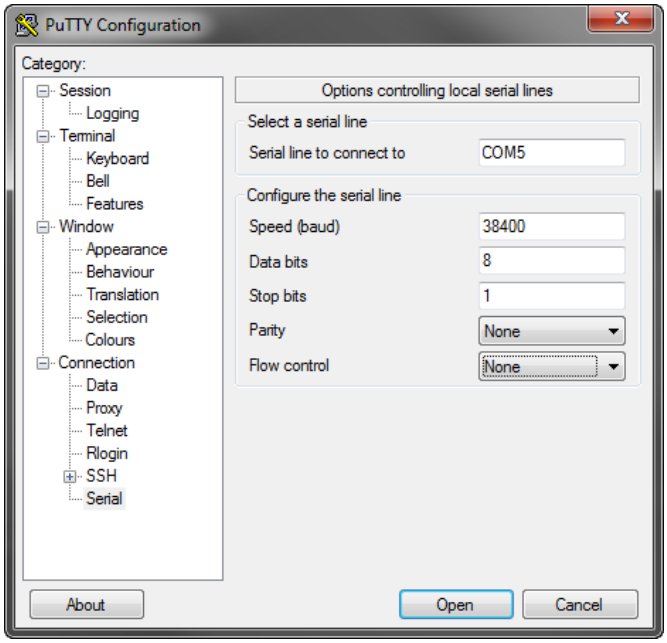
Write down the COM# associated with this device here: _____

Step 5.6 Once in properties of the SAKURA board go to the Port Settings tab and change all of the settings to match the image below and click “OK” or “Apply”.

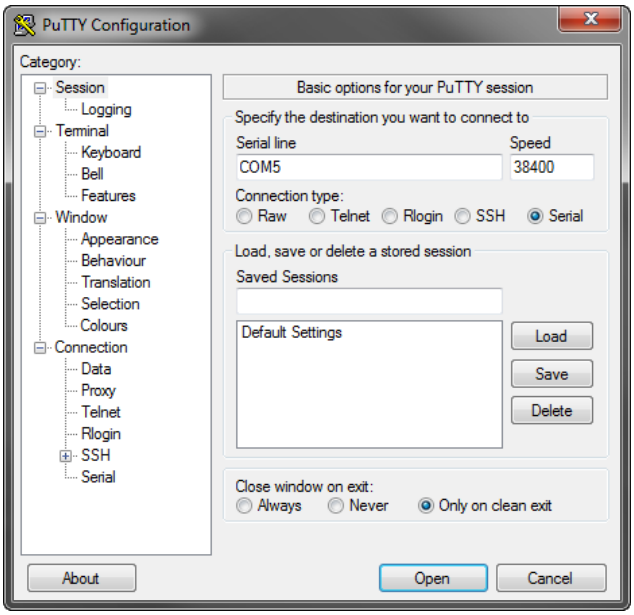


Step 5.7 Go ahead and click ok when finished and then open up the "PuTTy.exe" or HyperTerminal which can be found on the desktop. This application will be used to create a COM port with the GR-Sakura board. If you are using your laptop download PuTTy.

Step 5.8 In the category tree of the program go to the very bottom and select "Serial". Once in the Serial section change the information on this page to match the image below. Make sure to use the COM port you defined earlier.



Step 5.9 Go back to "Session" category within the Category tree. Once there change the "Connection type:" to "Serial" and click open. This step can be skipped if using HyperTerminal.



Step 5.10 The window should close and another window should open up. You are now able to test out the program that you have developed and put onto the board. To do this press switch 2 as many times as you would like and you should notice the on board LEDs changing, and you should see a corresponding Binary number show up within the PuTTy application. Below is an example of output that you should receive.



What you should observe is that the onboard LED's increment by 1 or sometimes more than one every time you press the switch. But if you look at the output of the Serial you will notice that the system still counts up by 1 every time. This is due to a phenomenon known as a "bouncing" switch. The way to fix this is by creating a debouncing circuit or to take care of it within the code.

Step 5.11 To develop a software debounce please insert the below code **above** the void setup function within the gr_sketch.cpp file.

```
int counter = 0;              // how many times we have seen new value
int reading;                  //the current value read from the input pin
int current_state = LOW;      // the debounced input value
long time = 0;                // the last time the output pin was sampled
int debounce_count = 10;      // number of millis/samples to consider
                              // before declaring a debounced input

int DebounceButtonPress(void){
    // If we have gone on to the next millisecond
    if(millis() != time){
        reading = digitalRead(PIN_SW);
        if(reading == current_state && counter > 0){
            counter--;
        }
        if(reading != current_state){
            counter++;
        }
        // If the Input has shown the same value for long enough let's
        // switch it
        if(counter >= debounce_count){
            counter = 0;
            current_state = reading;
        }
        time = millis();
      }
      return(current_state);
}
```

Step 5.12 In the void loop change while(ButtonPress()) to:

```
    while(DebounceButtonPress()){
```

Step 5.13 You should now be able to build, download, and upload the code to the GR-Sakura as before. You will have to exit out of your PuTTy session and reopen another session with all the previous information to reestablish a communication channel after the code has properly been loaded to the GR-Sakura board.

Step 5.14 You should now be able to test out your program once again at this point. What you should notice is that the LEDs no longer jump count but actually count in succession. This can also be seen in the PuTTy terminal.

6 Appendix

Overview:

This section has information about the GR-Sakura IDE and will allow you to program other projects on your own.

GR-Sakura IDE Language Summary

Structure

- setup() – is called once when a sketch starts. Used to initialize variables, pin modes, start using libraries, etc.
- loop() - as its name implies, this function loops consecutively the code inside it to allow your program to respond and change.

Variables

- HIGH|LOW – will output a “high” value to a pin or a “low” value to a pin respectively.
- INPUT|OUTPUT – determines the direction of a desired pin

Functions

Digital I/O

- pinMode(pin, mode) – configures the pin to behave as either an input or an output
  - pin – refers to the number of pin to be configured
  - mode – refers to what mode the pin is to be set (INPUT or OUTPUT)
- digitalWrite(in, value) – writes a high (5v) or low (0v) value to a digital pin.
  - Value – HIGH or LOW
- digitalRead(pin) – reads the value from a specified digital pin if it is HIGH or LOW

Analog I/O

- analogRead(pin) – reads the value of the specified analog pin. The voltage is converted to an integer.
- analogWrite(pin, value) – writes an analog value (PWM wave) to a pin.
  - Value – the duty cycle of the PWM wave.

Time

- Millis() – returns the number of milliseconds since the GR-sakura board began running the current program
- Micros() – returns the number of microseconds since the GR-Sakura began running the current program
- Delay(ms) – pauses the program for the amount of time specified as a parameter.
  - Ms – the number of milliseconds to pause
- delayMicroseconds(us) – same as delay but with different unit of time.
  - Us – the number of microseconds to pause.

Math

- Min(x,y) – returns the smaller number
  - X – to be compared with y
  - Y – to be compared with x
- Max(x,y) – returns the larger number
  - X – to be compared with y
  - Y – to be compared with x
- Abs(x) – computes the absolute value of a number
  - X – the number
- Pow(base, exponent) – returns the result of the exponent equation
  - Base – the base number
  - Exponent – the exponent of the base
- Sqrt(x) – returns the square root of x
  - X – the number that will be sought of its square root