OB1203 Custom Application Example Code

This document discusses the OB1203 example code provided for evaluating the US082-OB1203EVZ breakout board for OB1203 using an RL78 or RA microcontroller. It describes how to configure options for evaluation and how to debug for custom development.

Contents

1. Notice

The source code is provided for reference, demonstration, and/or evaluation. Note the disclaimer in the source code `oximeter.c` file header. No performance is guaranteed or warranted, nor is fitness or suitability for any medical device application claimed. Renesas will not indemnify customers using any part of the provided reference code. The OEM is responsible for performance of products made using any of the provided reference code. For general questions, contact Renesas. In addition, for medical devices, Renesas strongly recommends to consult a medical device regulatory compliance advisor.

2. Device Specification

A fully developed API supporting a selected set of OB1203 features is available on the OB1203 product page for simple integration projects. This document describes example code that supports custom development projects requiring more flexibility.

This software is provided for evaluating the US082-OB1203EVZ breakout board evaluation kit using an RL78/G13 (R5F100BGA) MCU as an example, or any RA series Renesas microcontroller evaluation kit

Tips:

- For best operation, a finger cradle or support is recommended to stabilize the finger on the sensor. These can be 3D printed. If you need a replacement or 3D file for the finger support, contact Renesas Sales.
- Custom boards developed by Renesas customers should follow the OB1203 Pulse Oximeter Module Electrical, Thermal, and Optical Design Guide, with particular attention paid to reduction of noise and EMI on VDD and heat sinking at the LED supply pins.

3. Resources

3.1 RA Series Microcontroller Resources

The example RA code is intended for use with FSP, which is a plugin for Renesas E2 Studio (an Eclipse-based IDE for firmware development). FSP includes a library functioning as a hardware abstraction layer as well as software tools inside E2 Studio for configuring firmware features, visualization tools for port assignments, and configuration of firmware modules. The example project is for the RA2A1-EK but it can be easily ported to other RA boards.

FSP (which will install the latest, and a separate version of E2 Studio) can be downloaded here and the FSP homepage is here.

3.2 RL78 Series Microcontroller Resources

For evaluating the OB1203 using the RL78/G13 multipurpose MCU, refer to the following resources:

- RL78 microcontroller source code, application notes, datasheets, and resources are located on the OB1203 product page. Submit a request for the source code via the website download link, and then return to same link to download the code once the request has been approved.
- Reference code for other microcontrollers such as Renesas ARM-based RA devices and Renesas Dialog Bluetooth microcontrollers are also available. Contact Renesas and/or check the OB1203 product page for updates and new releases.
- The reference code can be easily ported to any custom design featuring an RL78 microcontroller with sufficient RAM (~12kb) and ROM (~80kb) and processor speed (32MHz).
- The Eclipse-based IDE E^2 Studio for firmware development can be downloaded from the e2 studio product page.
- Note, a free trial license for the RL78 CC-RL compiler is available with the e^2 Studio download. Depending on the program size a commercial license may be required. Inquire on the Compiler Licenses product page.
- The Renesas flash programmer tool such as E2 Lite is required to program RL78 microcontrollers. It can be purchased from Renesas distributors.
- The free Renesas flash programmer utility can be downloaded from the Renesas Flash Programmer product page.

3.3 General Resources

- All downloads and application notes are available on the OB1203 product page. For example, detailed explanations of the Spo2, heartrate, and respiration rate algorithms are provided in a separate OB1203 Application Note.
- Please do not hesitate to contact Renesas for assistance! We have offices and sales teams around the world to support you.

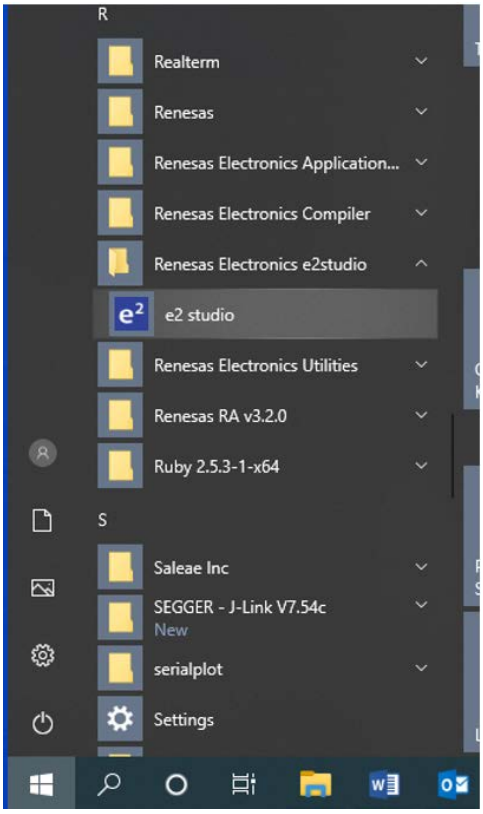
4. RL78 MCU Instructions

4.1 Opening the Project in E^2 Studio IDE

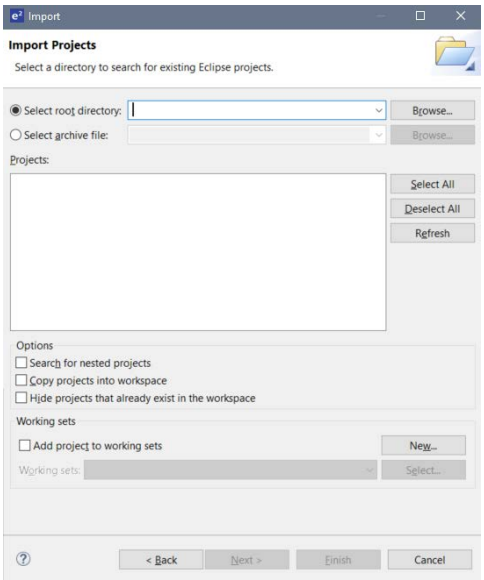
After downloading the source code from the Renesas website you will have a zip file.

1. Extract the zip file to the folder of your choice and note the folder location.
2. Download and install E^2 Studio integrated development environment and required components RL78. Be aware the RA series MCU uses the FSP platform which is a separate E^2 Studio platform, so there is no need to select RA for this install. Renesas Dialog MCUs also use a separate IDE.

3. Open E^2 Studio. It is possible to have multiple installations/versions of E^2 Studio. Important E^2 Studio updates with new features, tools, and capabilities are released a few times a year. We recommend always using the latest version.



4. After dismissing any start-up windows, click **File > Import** to open the import dialog.
5. Select Existing Projects into Workspace and click Next.



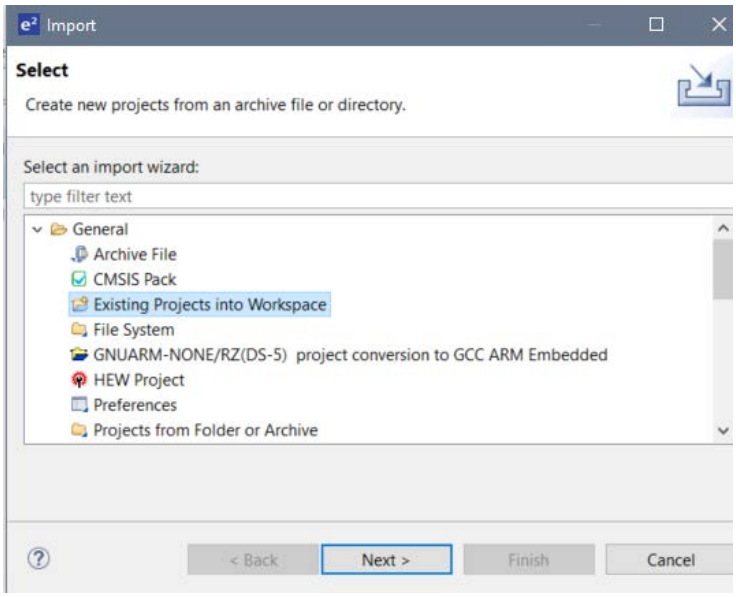
6. In the following dialog click **Browse** and navigate to the folder you unzipped.
7. Select the project **OB1203SD_RL2_EVK** identified in the **Projects** list and click **Finish**.

Tips:

- Use a new workspace for this project. It will save some possible confusion (File > Switch Workspace and change the name of the workspace folder to something like “ob1203sd_rl2”).
- If you have not used E^2 Studio before, we recommend watching some tutorial videos (e.g., on YouTube) to get familiar with the workspace, windows, features, etc.

4.2 Configuring Settings

1. In the project explorer window, expand the project, if necessary, by clicking on the > button in front of the project name. Note the `HardwareDebug` folder. This folder is auto-generated when the project is compiled and it includes the binary or hex file for programming the microcontroller, which can be done via the Renesas Programming Utility, for example. In this example we program via E^2 Studio’s (Eclipse) debugger interface.
2. Expand the `src` (source) folder by clicking on the > button next to the folder name.



3. Open `oximeter.c` by double clicking on it. This is the main program.
4. Note the disclaimer in the header. Reference code is useful as a starting point, but verification on your device and in your application is advised.
5. There are several options that can be configured in the header `oximstruct.h`, `spo2.h` and `ob1203.h`. The options are described near the top of the file in and with additional detail here.
   - Removing the `//` comment in front of a #define option enables it.
   - Typing `//` in front of a #define option disables it.

4.2.1. UART Output

Algorithm results are printed on the serial UART pin at 230400 baud, along with raw data on the serial UART. Data can be live-view graphed and saved with the free program SerialPlot, or view as text with a free serial terminal program like Teraterm or Putty. Connect the ground pin (VSS) and the data out pin (two away from VSS) a UART to USB adapter to record the data via a PC.

When `PRINT_RAW` and `PRINT_ALG_RESULTS` are defined, data is printed at 100 lines per second including both raw data and algorithm outputs. The print lines contain the following comma-separated: timestamp, IR data, Red data, IR LED current setting, Red LED current setting, SpO2, heartrate, respiration rate, infrared perfusion index, RMS variation of perfusion index (motion check), pressure factor (a measure related to how hard a user is pressing), and an error/status code related to estimated SpO2 and HR data reliability. Raw data is unique on

each line, but algorithm outputs reproduced on each line update only once every 400ms. Raw data print can be disabled by commenting out `#define PRINT_RAW`. In that case one line of algorithm results is output every 400ms. Printing of the IR and Red LED driver current settings can be independently toggled by defining `PRINT_PPG_AND_CURRENT`.

The raw data output is particularly useful for looking at sensor noise. For example, you can place a stationary (heavy) target over the sensor to check the noise level or you can view data live using SerialPlot and see how finger pressure affects the signal, or log raw data from a subject for later analysis.

4.2.2. Data from UART

The `DATA_FROM_UART` option is for algorithm verification and tuning. After recording raw data, you can define this mode by uncommenting `#define DATA_FROM_UART`. Also, undefine `PRINT_RAW` and `UPDATE_DISPLAY`. The device will start up and do nothing until it gets raw data sent to it over UART. Connect ground and the Tx and Rx pins of the UART to USB adapter (e.g., FDTI chip or cable) in that order to the pins on the EVK. To send data over UART use the Python 3 script provided in the software download. By default (`HANDSHAKE` defined), the firmware performs a handshake by returning a character after each received data pair. If it runs slowly, open Windows Device Manager, right-click on the COM port device, choose Properties > Port Settings > Advanced > Latency = 1ms.

Additionally, you can choose an option in the script that reads in data which includes a time stamp, and/or LED current data. Examples are given in a subsequent section. After receiving the required number of samples, the MCU computes the algorithm results and sends them back via UART. The Python script stores the data to a file. This is useful so you can run the same test data set with different algorithm settings and compare the results, or compare the results to a record from a reference device that was recording at the same time as the raw data was recorded. Choose `#define PPG_AND_CURRENT` to read in data that includes IR and red LED current settings in columns after the IR and red data. This is necessary to reproduce expected algorithm results when autogain changes the LED current.

Note: To verify large amounts of data while tuning an algorithm, contact Renesas for a C program version of the algorithm that can be run on a PC.

4.2.3. Step 0: Define Board Type (RL78 legacy EVK option)

For legacy RL78 evaluation kits, you can choose to define OB1203SD-RL2-EVK (limited distribution EVK version) via `#define RL2_EVK`; otherwise, leave it commented out for settings appropriate to OB1203SD-RL-EVK (original, non-Bluetooth EVK).

4.2.4. Step 1: I/O Options

In this option, you will choose whether the device will:

1. Display the results on the display
2. Print raw data out over the serial UART pin
3. Import previously recorded data over UART

4.2.5. Step 2: Heartrate Only or SpO2

By default, the firmware configures the OB1203 to use both IR and red LEDs to measure blood oxygen concentration (SpO2). When only heartrate is necessary, the red LED can be turned off and LED power consumption reduced by approximately half. To enable heartrate only mode define `HR_ONLY`, otherwise leave it commented out.

You can also change the rate at which the algorithm runs. For SpO2 and HR only applications, where respiration rate does not need to be calculated, the algorithm can be run only once per second by commenting out COMPUTE_RR in the file SPO2.h. Note, the more verification was done for the SpO2 option. By default, COMPUTE_RR is enabled and the algorithm runs (40ms) every 400ms. The algorithm runs on fewer samples when it runs more frequently in this mode so the overall difference is only about 20% less computation for not computing RR.

In the case of turning off COMPUTE_RR in _50sps mode the algorithm runs every 80ms.

In 100sps mode (default), with COMPUTE_RR off the algorithm runs every 100ms.

4.2.6. Step 3: SNR vs. Power Consumption Option

In this step you can optimize for lower power or signal to noise ratio. This demonstrates how a developer might attempt to optimize the operation for a particular signal range. In this step you set how frequently the OB1203 samples.

The signal to noise ratio (SNR) is a function of how many measurements are made and averaged on the OB1203 before recording a value to the FIFO register. Choosing BEST_SNR runs the ADC and LEDs at maximum speed. Some heat sinking care is needed for this mode. Other modes can be defined with successively less pulse averaging and pulse rate.

With the default MID_POWER configuration OB1203 8x oversampling and outputs the averaged (binned) data 100 times a second. The oversampling can be set at 16x by defining BEST_SNR or 4x by defining LOW_POWER.

The relative signal to noise ratios for BEST_SNR, MID_POWER, and LOW_POWER configurations are 1.4, 1.2, and 1 respectively, with RMS noise in the bandwidth of interest for normal heart rate range and mid-range LED of approximately 10, 12, and 14 counts.

Another option for reducing power is to decrease the number of samples per second. This cuts the algorithm computation time in half at constant LED power. This option is enabled by defining_50sps in SPO2.h. If this mode is defined, BEST_SNR uses the average of 32 measurements per output data point, MID_POWER uses 16 averages per data point, and LOW_POWER uses 8 average per data point. Note, the most clinical verification was performed for 100sps and spot checks were performed for 50sps.

There are also options for successively less sampling and averaging: LOW_POWER_2 and LOW_POWER_3, or for _50sps mode LOW_POWER_4.

Note, the algorithm's high-pass filter performs a moving average of the incoming data. For 100 samples per second (saps) (default) 8 samples are averaged, reducing noise at constant data rate. For 50sps, 4 samples are averaged. The resulting bandwidth and noise reduction from averaging are similar. This has the advantage of reducing RAM requirements and computation at the expense of some accuracy at very high heart rates.

4.2.7. Step 4: Display Output

To enable algorithm output and user interactions with a 128x64 I2C OLED display make sure #define UDPATE_DISPLAY is uncommented. To show the pulse waveform on the LCD display, define SHOW_PPG_WAVE. This increases the display update time and associated MCU processing time but visibly shows what the actual signal looks like. It is enabled by default.

4.2.8. Step 5: Select Strictness

Define CHECK_PI to enable checking for measurement errors from motion or low signal.

Defining CHECK_PI mode allows users to define thresholds below which the device will not display SpO2 results. During evaluation or development it is recommended to comment out this mode. When preparing for regulatory review, contact Renesas to discuss using CHECK_PI to prevent display of potentially incorrect SpO2 readings for very low perfusion index or very low count readings. This involves some evaluation of the system noise performance and typical and worst case perfusion index.

4.2.9. Step 6: SpO2 Debugging

Normally, SpO2 algorithm output is clipped to a maximum of 100%, regardless of the calculated ratio of ratios “R” value. In order to calibrate the SpO2 for a particular device, it is advisable to define “SPO2_DEBUG” to avoid clipping data at the upper or lower extremes during hypoxia tests.

4.2.10. Step 7: Menu Operation

Define `USE_MENU` to enable using a press of the button to access menu options for enabling alarms, etc. This feature is a work in progress. Leave it undefined to leave menu features off.

4.2.11. Step 8: Auto Power Off

Enable auto power off for hardware with GPIO and shutoff/wake/enable circuit.

4.2.12. Step 9: Miscellaneous Debugging

Define ten_x_pi to output 10x larger values of the perfusion index to the display, which can be useful when doing tests at low perfusion levels.

Define DEBUG to disable auto power off, for instance, for collecting long datasets.

5. Code Organization

The code consists of auto-generated code for configuring and controlling the RL78 and custom application-specific code. Files that start with “r_cg_” are files originally created by the RL78 code generator. Everything else is application-specific code.

For RA autogenerate files are in folders other than `src`.

`Oximeter.c` is the main program and has two support files `oximeter.h` which has the things related to UART serial print and display, and `oximstruct.h` which has things related to running the algorithm.

`OB1203.c` and `OB1203.h` provide code for configuring and reading out data from the OB1203 bio sensor ASIC.

`KALMAN.c` and `KALMAN.h` provide code for doing outlier filtering and averaging of data. This code is a zero-order Kalman filter where only the variance in the data is analyzed to determine outliers and the predictor is based on a moving average. `KALMAN.h` defines parameters that set the duration of the averaging for displayed SpO2 and HR values and also the related parameters for several averages used internally in the algorithm.

`I2C.c` and `I2C.h` are low-level bit-bang I2C code originally used to work around some hardware issues. Customers can replace this with RL78 hardware I2C modules, if desired.

Files in the `OLED` and `OLED_Display` folder are specific to the OLED display features.

`SPO2.c` and `SPO2.h` contain the bulk of the algorithm code, which is described in the algorithm application note. `SPO2.h` defines many tunable parameters of the algorithm such as the duration of averaging used for the displayed respiration rate, as described in the OB1203 algorithm application note.

`SAVGOL.c` and `SAVGOL.h` are files related to the high-performance smoothing filter.

`menu.c`, `menu.h`, `button.c`, and `button.h` are related to features of the display controllable by pressing the button.

6. Compiling and Debugging Code for RA

This function is pretty straightforward. Click compile to compile, and click debug to debug. Contact a Renesas FAE or the Renesas Rulz forum for specific help.

Other features of FSP that may be useful are visible in the FSP Configuration view by opening configuration.xml. This allows the user to select the MCU target, EVK board, GPIO pins, interrupts, and other features.

7. Compiling and Debugging Code for RL78

When importing an existing project into a new version of E^2 Studio, occasionally there are few things that need to be cleaned up.

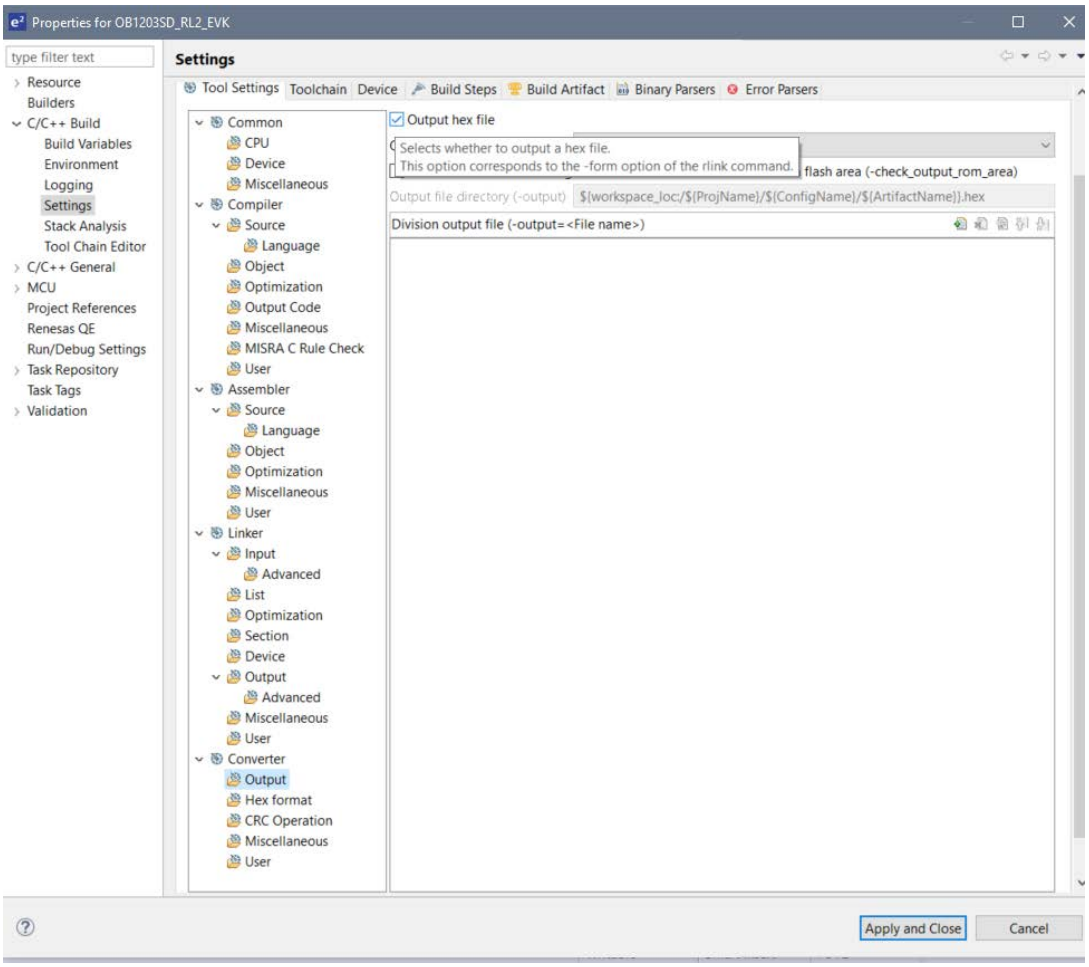
First, you will need to clean the project, which removes existing compiled code. Then build the project:

1. Right-click anywhere in the project explorer window and click **Clean Project**.
2. Right-click anywhere in the project explorer window and click **Build Project**. If the console shows that the build was successful, you have the necessary components installed.

7.1 Optionally Clearing a Compile Error from Upgrading E^2 Studio/Compiler

In some cases, when upgrading to the latest version there may be an error related to a circular reference or an error that mentions an output file such as an .x or .abs file. If that is the case, you can clear the error by doing the following:

1. In the project explorer window, right-click on the **HardwareDebug** folder and click **Delete**. Don't worry. This is an auto-generated folder. It does not have any source code in it.
2. In the project explorer window, right-click on the project title itself and then select **properties**. The properties dialog opens. (If you right-click on one of files instead of the project name at the top, then you get a subset of the options in the dialog box, so make sure to select properties for the entire project.) You can also access this from the **Project > Properties** using the top menu. This will open the properties for the current active project, so make sure that you either have only one project in the workspace or only one project is open.
3. In the properties dialog box, in the left-side panel, expand C/C++ Build and choose **Settings**.

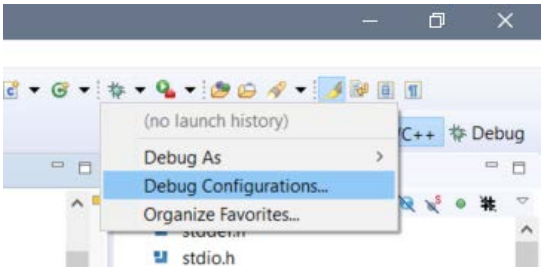


4. Scroll down to the **Converter** sub-menu and click on **Output**. Deselect “Output hex file”. Click **Apply and Close**.
5. Build the project.
6. Re-open the properties dialog, select “Output hex file”. Click **Apply and Close**.
7. Build the project.

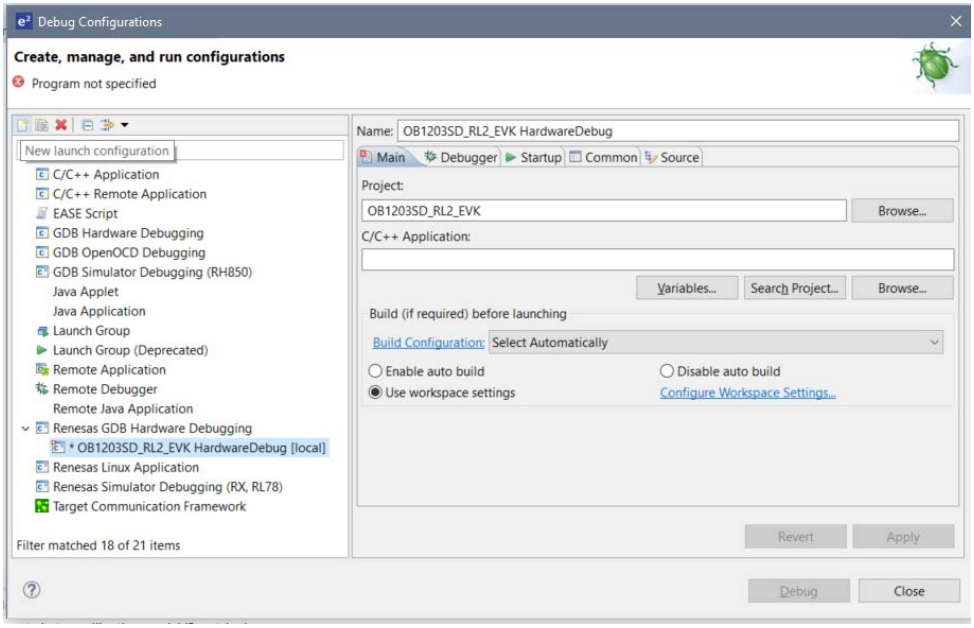
This clears the error associated with older projects being opened in a newer version of E^2 Studio/CC-RL compiler. At this point you have a hex file for debugging. The next step is to teach E^2 Studio how to debug this project.

7.2 Debug Configuration

1. From the **Run** menu select **Debug Configurations…** or click on the drop-down next to the small bug icon.

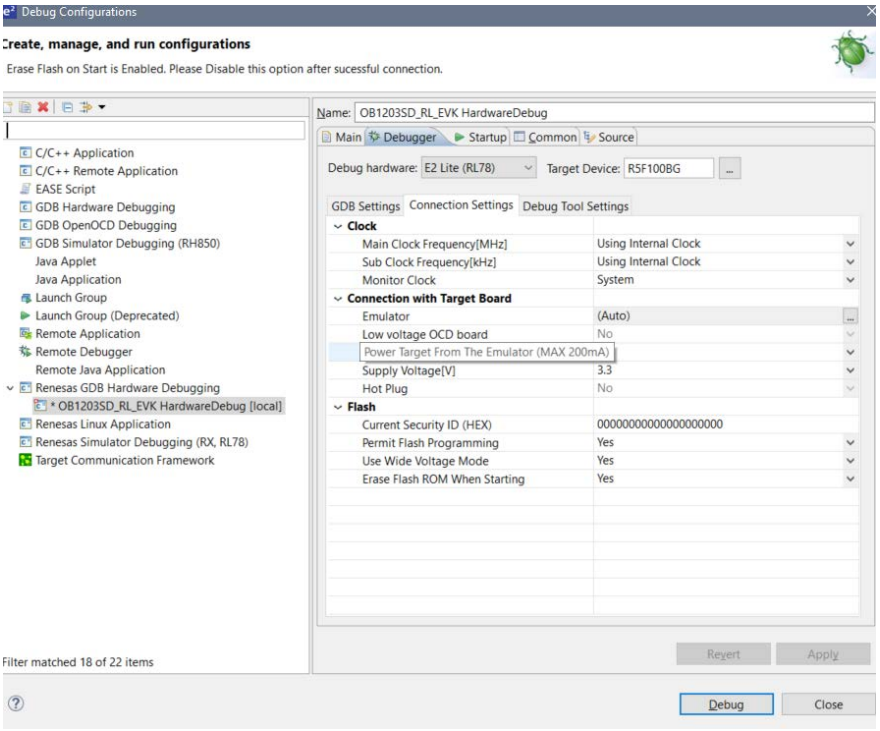


2. In the **Debug Configurations** dialog, in the left panel, scroll down and select **Renesas GDB Hardware Debugging**. If a debug instance exists, you can click on it; otherwise, click the **+** button at the top of the left panel to create a new instance of Renesas GDB Hardware debugging.



3. On the right-hand panel in the **Main** tab, select **Search Project** and choose the **.x** file.
4. Select the second tab called **Debugger**.
5. On the Debugger tab, under Debug hardware, select your debugger (e.g., “E2 Lite (RL78)” or “E1/E20 (RL78) depending on the debugger you are going to use.
6. On the Debugger tab, for Target Device, select **RL78 > G13 > R5F100BG**.

7. On the debugger tab, there are three sub-menu tabs. Select **Connection Settings** and set **Power Target From The Emulator** to **Yes**.



8. Optionally, switch to the **Startup** tab and scroll down to the **Set breakpoint at main** option and deselect it if you do not like the code to stop at the first point of user code in addition to stopping at the first point in assembly code. Then, one click of the play button during debugging will start the code immediately from cstart.asm.
9. Click **Apply**. You can leave this dialog box open.

7.3 Set Up the Hardware and Begin Debugging

You are now ready to debug. The next step is to get the hardware set up and start the debugger. If you do not want to debug and just want to push the program to the hardware, skip this section.

Before connecting your programmer to the PC, connect an adapter (see “Device Specification

A fully developed API supporting a selected set of OB1203 features is available on the OB1203 product page for simple integration projects. This document describes example code that supports custom development projects requiring more flexibility.

This software is provided for evaluating the US082-OB1203EVZ breakout board evaluation kit using an RL78/G13 (R5F100BGA) MCU as an example, or any RA series Renesas microcontroller evaluation kit

Tips:

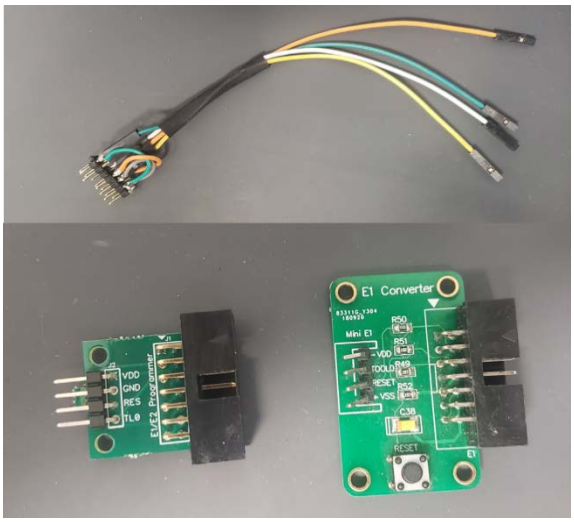
For best operation, a finger cradle or support is recommended to stabilize the finger on the sensor. These can be 3D printed. If you need a replacement or 3D file for the finger support, contact Renesas Sales.

Custom boards developed by Renesas customers should follow the OB1203 Pulse Oximeter Module Electrical, Thermal, and Optical Design Guide, with particular attention paid to reduction of noise and EMI on VDD and heat sinking at the LED supply pins.

- Resources
- RA Series Microcontroller Resources

- The example RA code is intended for use with FSP, which is a plugin for Renesas E2 Studio (an Eclipse-based IDE for firmware development). FSP includes a library functioning as a hardware abstraction layer as well as software tools inside E2 Studio for configuring firmware features, visualization tools for port assignments, and configuration of firmware modules. The example project is for the RA2A1-EK but it can be easily ported to other RA boards.
- FSP (which will install the latest, and a separate version of E2 Studio) can be downloaded here and the FSP homepage is here.

1. RL78 Series Microcontroller Resources") between the programmer and the header pins on the EVK. The following figure shows some adapter cables/boards. You can build your own or try to obtain one from a Renesas sales engineer.



2. Double-check that you have not crossed the power (VCC/VDD) and ground (VSS) lines or shorted them.
3. Connect your debugger to the PC via USB cable.
4. Click "Debug" from the debug configuration menu. You will be presented with a dialog box asking if you want to switch to debug view. Select the box to always do this in the future and click yes.

The debugger now starts up and the view is switched to debugger view. In this view you have access to buttons for controlling the program operation. If the buttons do not show you may be in the C/C++ view. You can switch views with this tool:

Other views, like the RL78 code generator, are accessible with the item that looks like a spreadsheet with a yellow "+".

The debugger halts at the first line of assembly code. To run the code, click the green play button on the debug toolbar. To pause, click the pause button. The single step, step over a function, step into a function, or step to function return (step out) use the respective buttons on the debug toolbar. You can hover over the buttons to see their function. To reset the program, use the reset button (little green bug with arrow to the right).

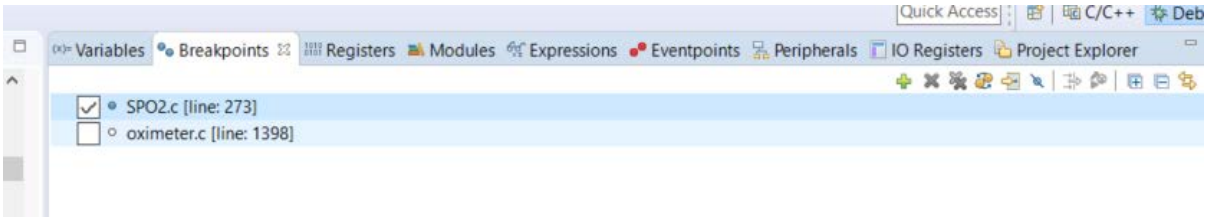
7.4 Creating Debug Points

Breakpoints for pausing code operation during debugging are created by double clicking in the region to the left of the code, where the code line numbers show up.

By default, E^2 Studio inserts a hardware breakpoint for which RL78 has only one. Hardware breakpoints show up as a blue square instead of blue circle. To switch the breakpoint type, right-click on the breakpoint and select

“switch default breakpoint type to software”. Then clear any hardware breakpoint and double-click to create a new software breakpoint. Now you can create as many as you want.

In the debug view, there is a pane with several tabs of which one is called “Breakpoints” and has an icon made up of two blue dots (see below). There you can see all your breakpoints, toggling them individually by clicking on the check box or remove one or all by right clicking and selecting the appropriate option.



Another useful pane while debugging is the Variables view. When an operation is paused, either at a breakpoint or by clicking the pause button, this view shows the values of local variables in whatever function operation is at.

The Expression view shows any variables with global scope that you type in. You can also type numeric computations including variables, so two variables added together or scaling a variable. This can make the value either easier to view or see on a chart (live view graph). When you have variables that are an array, you can expand the variable, and then select items and copy and paste them into a text editor.

Most importantly, you can right-click on an expression and select “Enable real-time refresh”, then right-click and set the refresh interval to make the expression update in real-time. This live view is useful while debugging.

In addition, for any variables that are enabled for real-time refresh, you can right-click and select “add to chart” and a chart will appear showing that variable, making a plot of that variable’s history. You can right-click on the chart and change the line color or delete items or reset the chart as needed. You can also set the chart refresh interval to make it scroll faster or slower.

Tip: When debugging and you have expressions updating in real-time, you cannot add new expressions. First, disable real-time refresh by clicking the icon that looks like a refresh with a slash through it. Then add your new expressions, then re-enable real-time refresh.

7.5 When the Debugger Does Not work (Debugging the Debugger)

A few quick checks before checking for bugs in the debugger:

1. Is the debugger plugged into a powered USB port and does the PC recognize a device? Windows will make a sound when you plug it in.
2. Is the part connected to the debugger correctly? Jumper wires correct? Jumper wires not broken? Jumper wire connector not plugged in backwards (unplug now!).
3. Correct target device set in debugger configuration (i.e., R5F100BG)?
4. Correct programmer/emulator type selected (e.g., E2 Lite (RL78) if you are using the programmer with the clear plastic cover)?
5. Programmer set to “Power Target from Emulator”?

If all those check pass, the next step is to check for software issues. Sometimes E^2 Studio has lost track of the programmer. Sometimes it does not properly close out a service, etc. This can happen if the programmer gets disconnected from the PC during debugging, or for other unknown reasons.

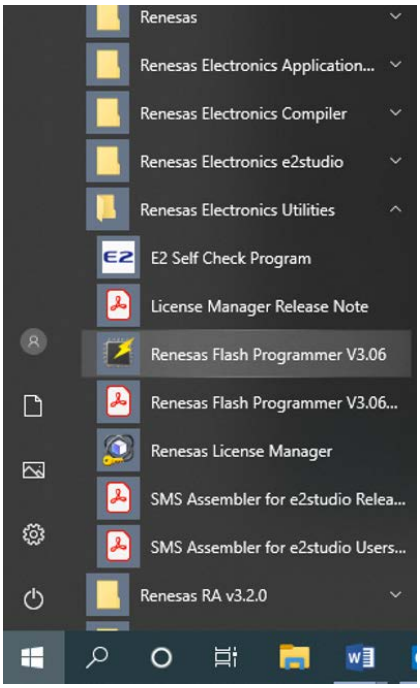
1. Unplug and reconnect in the programmer/emulator.
2. Close and reopen E^2 Studio.
3. Open task manager in Windows (Ctrl-Alt-Delete > Task Manager > Details) and end the e2-server-gdb.exe task. (Logging out of Windows and logging back in or restarting your computer has the same effect but will take longer.)

4. Try pushing the .MOT file from the project HardwareDebug folder using the Renesas Flash Programmer utility. This procedure is described below in detail in the Flash Programmer section. If this succeeds, then there are no problems with the hardware; only software or debugger settings. If it fails claiming the emulator is already in use, then kill the e2-server-gdb.exe task as described above. If it fails claiming it cannot find the device, then you have a hardware connection issue or bad hardware. If it fails claiming it cannot find the emulator, then you may have bad emulator hardware or a bad USB cable or a bad USB port. Sometimes a USB port gets a current overload and can be reset by restarting the PC.
5. Contact Renesas for assistance. Your time is valuable to us.

8. Using the Renesas Flash Programmer Utility

To program the device without debugging, make sure your project has a compiled .MOT file.

1. Navigate to your project folder > HardwareDebug and look for a .MOT file. If it is there, continue; otherwise, you will need to build the project (see “Compiling and Debugging Code”).
1. Download Renesas Flash Programmer. If you installed E^2 Studio, you probably already have it. If not, follow the link in “Resources” to download and install the Renesas Flash Programmer Utility.
2. Run the Renesas Flash Programmer tool.



3. From the File menu select **New Project**.
4. In the **New Project** dialog, choose RL78 as the microcontroller.
5. Enter a project name such as OB1203SD-RL2-EVK.
6. Under **communication**, select your programmer/emulator tool such as “E2 emulator lite.”
7. Under **tool details**, select 3.3V for the supply power.
8. Connect your programmer/emulator via USB and connect the EVK programming header to the emulator via an adapter board and jumper wires.
9. In the Renesas Flash Programmer dialog, select **Connect**. If you get a “Tool is already in use” error then stop the debugger in E^2 Studio and try again.
10. Select the program file by clicking **Browse** and navigating to your project folder’s HardwareDebug folder. Choose the MOT file to program to the EVK.

11. Click **START**. The Renesas Flash Programmer will now program the IC. After programming, the MCU is held in reset by the programmer.
12. Disconnect the programmer from the EVK.
13. Turn off the battery switch, then turn it on and press the start button on the EVK to run the program.
14. In the Renesas Flash Programmer tool from the file menu, choose **Save Project**.
15. Close the Renesas Flash Programmer tool if you have no additional devices to program.

9. Data from UART Examples

For sending data to the EVK and reading back algorithm results, there is a python script (and several simple program variables to set), as described below.

9.1 Prerequisites

- Install Python 3 or above on your system.
- Install SciPy, Numpy, and pySerial libraries.
- Connect EVK GND, and UART Tx/Rx lines to a USB to 3.3V serial UART adapter such as an FTDI board or cable. The line labeled (O) on the EVK connects to the FTDI Rx pin. The line labeled (I) connects to the FTDI board Tx pin.

9.2 Record Data

The first step is to record data using the PRINT_RAW option and suitable terminal program. For Windows SerialPlot is a good option, or TeraTerm. Typically two to five minutes of data is useful. At 100sps this means the data file is 12,000 to 30,000 lines long. Adjust the buffer in TeraTerm, if needed, to ensure all data is captured, or use SerialPlot or a Python script to capture serial prints.

A library of pre-recorded data for various user skin types, heart rates, breathing rates, perfusion index (finger temperature), etc., is invaluable in algorithm tuning and verification.

9.3 Program Variables

Save the data file to the folder with the python script or vice-versa.

Open the python script write_serial_test_data.py or write_serial_test_data_handshake.py using your preferred python editor or text editor such as Notepad++:

1. Enter the correct COM port number as `port_number`. If you do not know the COM port number, open Windows device manager and check which COM port shows up when you plug in your USB-to-serial adapter. Alternatively, you can note which COM port number is new in a list provided in a terminal program like TeraTerm, RealTerm, or SerialPlot. (Connect your FTDI before opening the terminal program or refresh the list.) Set the baud rate to 230400.
2. Select the correct `interval`. If your data was taken at 100sps, select 40 for the interval. If your data was taken with 50sps, select 20 for the interval.
3. Choose the `startline` to be the first row of data from the file that will be sent to the EVK. 0 is the minimum value. You can increase this to avoid any odd data at the beginning from hand motion or to skip a line with column header information.
4. Enter the data file to be analyzed as the `filename`. This Python script will save the algorithm outputs as a new txt file that matches the input data filename but with “_results” appended to the filename.
5. Enter the total number of lines to read from the file as `lines2read`. For example, if there are 16,000 lines in the file and you want to consider all the lines, choose lines2read = 16000-startline.

6. Note the columns of your data corresponding to the IR and red ADC data and the delimiter between the columns. For example, if there is a column of IR data, a comma then red data, the data file will look like the following:

```
1  205887,198778
2  205876,198733
3  205903,198660
4  205909,198699
5  205874,198753
6  205850,198756
7  205911,198677
```

   Set `delim` equal to "," or ',. Set `ir_data_column` to 0 and `r_data_column` to 1. If, for example, there is a column of time stamp data or other in the first column then set the IR data column to 1 and the red data column to 0. If the columns are separated by tabs use "\t" as the delimiter, or for columns separated by a space use ' ' (space between quotes).

7. Set `inc_current` to `True` if you have IR and Red LED current settings and want to send them to the algorithm and have the algorithm consider jumps in the LED levels. This is necessary for any data taken where the LED current changes. In this case you must also define `PPG_AND_CURRENT` in `oximeter.c` so the MCU knows to receive the IR values and respond according.
8. Set `decimate` configuration. If you have data taken at 100sps and want to simulate 50sps by skipping every other data point, set `decimate` to `True` and choose 20 for interval.
9. Set `handshake` configuration. Set `handshake` as `True` if the program hangs during operation due to irregular timing (default FW uses handshake). This happens for some drivers. If you set this to `True`, it runs slightly slower but more reliably. To avoid a speed-up operation, go into Windows device manager and select the com port Properties > Port Settings > Advanced > Latency = 1ms.
10. Set number of columns `NUM_COLS`. If set to 3, the program expects 3 integer values such as SpO2, HR, and RR (respiration rate). If set to 4, the program expects 3 integer values and one float, such as R (ratio of ratios) or PI (perfusion index).

9.4 Running the Python Script

1. Open a **command window** or **Windows PowerShell** in the folder where the data file and the Python script are located.
2. Program the EVK with a version of the firmware where **DATA_FROM_UART** and optionally **PPG_AND_CURRENT** are defined.
3. Connect the EVK UART and either start the debugger and push play so the program is running (waiting for data) or push the button to start the program.
4. Wait until the EVK is past the start-up screen, and then at the command prompt type `Python write_serial_test_data.py` (or if Python 3 is defined in your system path as Python3 type at the `Python3 write_serial_test_data.py`). The algorithm outputs will appear in the terminal window and be stored in the results file when the program finishes running.

If the Python script operation hangs or algorithm gives unexpected results, check the following:

- The EVK is not connected to the UART
- The EVK is not powered on
- The EVK is not running (halted in debugger)
- The Python program was run while the EVK was still on its startup screen
- The EVK is not running code with DATA_FROM_UART defined or UPDATE_DISPLAY or PRINT_RAW is defined (only DATA_FROM_UART should not defined)
- The EVK code interval is 40 (100sps) while the python code interval is 20 (50sps) or vice-versa

- The EVK code is expecting handshake and the Python code is not
- The EVK code is expecting LED current values and Python is not, or vice-versa

10. Algorithm Parameters and Tuning

Several settings and parameters should be tuned by customers based on the application, the optomechanical design, and the achieved signal-to-noise ratio. These parameters are found in spo2.h.

10.1 SpO2 Calibration Parameters

10.1.1. R-curve

Spo2 calibration (R curve) is set by defined values `Rsquare`, `Rlinear`, and `Rconstant`, for example:

```
#define Rsquare (0)
#define Rlinear (-48)
#define Rconstant (118.5)
```

Spo2 is calculated according to the formula:

$$spo2 = Rsquare * R^2 + Rlinear * R + Rconstant$$

Where R is the ratio of ratios (Red AC/Red DC) / (IR AC/ IR DC).

Spo2 level is clipped below `MIN_SPO2` because such low values are not verifiable in a hypoxia lab. When `DEBUG_SPO2` is defined the clipping does not occur.

10.1.2. Finger Pressure Correction

For applications where variable finger pressure is of concern, customers can enable finger-pressure correction by defining USE_PRESSURE_CORRECTION.

```
#define USE_PRESSURE_CORRECTION
```

To disable finger pressure correction, comment out this line.

Finger-pressure correction is not the same for each individual due to variation in PPG waveform shapes (e.g., the size of the dicrotic notch and blood pressure). Therefore, finger-pressure correction tuning parameters should be chosen conservatively. An example is provided below with explanatory comments

```
#define MIN_PI_FOR_PRESSURE_CORRECTION 0.1 //no finger correction below this PI
#define PRESSURE_FACTOR_THRESHOLD 70 //pressure factor below this has no effect
#define PRESSURE_SLOPE_DENOM 9 //larges means less correction
#define PRESSURE_FACTOR_MAX 140 //pressure factor above this adds no correction
```

10.1.3. Noise Correction

To extend the useful range of PI a fixed noise can be subtracted from the data by defining `USE_NOISE_CORRECTION`. For details on characterizing noise, see the OB1203 Algorithm Application Note.

If the power supply noise dominates the SNR, the device will have noise that does not depend on LED intensity, but rather there is fixed noise associated with the receiver. For such applications, define the typical RMS (average deviation) noise with `TYP_NOISE`. This depends on the power settings (`BEST_SNR`, `MID_POWER`, or `LOW_POWER`) and the hardware. To reduce supply noise use an LDO on VDD, or in the case of the switching power supply use at least an RF choke resonant damper to reduce pulsation on VDD and VLED.

The RL2 and BT2 evaluation kits are power supply-limited noise limited. *To achieve lower noise you can short the boost regulator PMIC input and output pins and power directly from batteries (provided the batteries are fresh) or a clean DC supply*. Note, the EK1 evaluation kit has 2x lower noise than the other EV kits due to lower PMIC ripple (RF choke).

If the application has noise that primarily scales with LED intensity (Tx noise limited) define SCALE_TX_NOISE. Then define the max noise at IR LED current setting of 1023.

```
#define MAX_NOISE 40 //Tx noise at max IR current (0x3F)
```

10.1.4. CHECK_PI

Define CHECK_PI to limit the output of potentially erroneous SpO2 values for cases of extremely low perfusion.

MIN_RMS defines the RMS signal amplitude for the IR channel, below which SpO2 values are not displayed. The actual measurement is average deviation, which is 1.25x smaller than RMS for PPG signals. The measurement has 3 bits of fixed precision, so it is 8x the actual low limit. For example, MIN_RMS of 152 indicates an actual limit of 152/8 = 19, which corresponds to an AC peak to peak signal of 19*2*sqrt(2)*1.25 = 67 counts.

```
#define MIN_RMS (152) //8x the minimum allowable RMS (avg dev) for IR channel
```

We also need to check the variation of the PI level. If it is varying wildly then the SpO2 value will be lower than expected and we should freeze the SpO2 value until a better reading is available.

```
#define NORMAL_PI_MAX_RMS (0.3) //RMS fraction of PI allowable with decent PI
```

At low PI levels, we need to be more stringent about the level of motion and other sources of variation that we can tolerate. LOW_PI_LEVEL defines the floating point PI percentage below which we set a lower allowed variance.

```
#define LOW_PI_LEVEL (0.15) //below this we use a more stringent check on PI rms
#define LOW_PI_MAX_RMS (0.15) //RMS fraction of PI allowable with crappy PI
```

10.1.5. SG Filter Tuning

For hardware where the SNR is not ideal, if higher than expected SpO2 is observed, the SG filter length should be shortened because the additional noise affects the smoothed red channel amplitude more than the IR channel.

To avoid this effect, either change the calibration or reduce the SG filter length. For the case of reduced SG filter length, in the following section of code, replace the short and long SG values by ½ of the nominal value. For example, replace 8 by 4. 4 indicates an actual SG filter length of 8.

```
#ifdef _50sps
      #define SHORT_SG 2 //must be a power of 2
      #define LONG_SG 4 //must be a power of 2
      #define SHORT_PER1F_THRESH_DOWN 200 //120 BPM
      #define SHORT_PER1F_THRESH_UP 222 //110 BPM
#else
      #define SHORT_SG 4 //must be a power of 2
      #define LONG_SG 8 //must be a power of 2
      #define SHORT_PER1F_THRESH_DOWN 400 //120 BPM
      #define SHORT_PER1F_THRESH_UP 444 //110 BPM
#endif
```

In the same section of code, you can set the threshold for heart rate period (including 3 bits of fixed precision) for which we shorten the SG filter to allow higher bandwidth heart rate signals. For example, the threshold at which we switch to shorter SG filter occurs at 400, which is a heart rate period of 50 samples or 6000/50 = 120 bpm.

11. Revision History

Revision	Date	Description
1.00	Sep 9, 2022	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.