OB1203

OB1203 Heart Rate, Blood Oxygen Concentration, Pulse Oximetry, Proximity, Light and Color Sensor: Sensor Programming

Abstract

This application note provides a quick-start guide for programming the Renesas OB1203 all-in-one PPG biosensor, proximity sensor and color sensor.

Contents

1. Application Circuit

It is suggested to use low noise power supply sources, such low-noise low dropout (LDO) linear regulators. Independent supplies for analog and digital supply (VDD) and LED supply (LED_VDD or LVDD) are suggested. Pull-up resistors in the appropriate range to microcontroller logic voltage of 1.8–3.3V are necessary for I2C SDA, I2C SCL and INTB pins. A decoupling cap between VDD and analog ground (GND) of 0.1µF and optionally 1 µF for longer leads is suggested. 4.7–10 µF capacitors are useful between LVDD and LED_GND (LGND). The analog ground (GND) and LED power ground (LGND) should be connected at a low impedance node to avoid ground bounce effects when the LED current is high. Trace lengths should be minimized and wider trace/via widths used for current carrying LVDD and LGND lines. All the LED pins should be connected to a low thermal impedance pad.

For bench test functionality purposes it is sufficient to connect LVDD and VDD to 3.3V source relative to LGND and GND, and connect the I2C and INTB lines to the logic level (typically 1.8V or 3.3V) via pull-up resistors in the appropriate range. Note that the internal digital and analog levels are set by an internal LDO regulator.

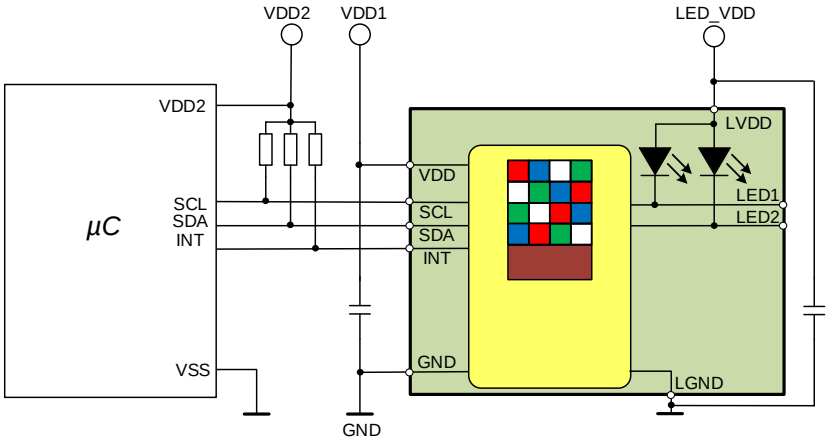

Figure 1. Application Circuit

2. OB1203 Overview

OB1203 communicates via I2C on the SDA (data) and SCL (clock) pins, along with an active-low (INTB) interrupt pin.

2.1 Address

By default, the 8-bit device address is 0xA6 for write operations and 0xA7 for read operations. The 7-bit address used by some I2C libraries is obtained by a bit shift to remove the write/read bit, namely the default 7-bit address is 0x53 (resulting in 0xA6 for write operations). A factory programmable option is the one-time-programmable (fuse) DEVICE_CFG bit which sets the write/read address to 0xA4/0xA5. When DEVICE_CFG is fused the new 7-bit I2C address permanently becomes 0x52 (resulting in 0xA4 for write operations).

Table 1. I2C Address

DEVICE_CFG bit 1	I2C Address		
	7-bit to be Left Shifted	Write 8-bit	Read 8-bit
0 – default	b: 1010 011	0xA6	0xA7
1 – programmed	b: 1010 010	0xA4	0xA5

2.2 Register Organization

The internal register map is organized into four main sections:

1. Status and data (0x00 through 0x12)
2. Settings (0x13 through 0x37)
3. FIFO (0x38 through 0x3B)
4. Miscellaneous registers (0x3C, 0x3D, 0x42, 0x43, 0x4D)

Convenient for writing and reading of multiple registers in a single I2C transaction, a read or write operation always moves (indexes) the internal pointer to the next register, **with one important exception**. The FIFO_DATA (0x3B) data register is special, providing access to an internal RAM that stores the biosensor data. Successive reads of the FIFO_DATA register index through the RAM, not the register map. So the register map essentially "breaks" at 0x3B. To access registers beyond 0x3B a write operation must be performed.

It is necessary for the FIFO and optionally its read, write and overflow registers to be read in a single burst (aka "block") read. This is described in the datasheet. Remember: "burst read or bad data".

Programming tip: To do a register dump, which is common in SW/FW debugging, separate block reads should be performed for registers before and after 0x3B.

3. Status and Data Registers (0x00–0x12)

3.1 Status and Interrupts

Device status can be obtained by reading registers 0x00 and 0x01 (burst read is best). Status bits are typically cleared when the status register is read.

Interrupts can be asserted for some of the status bits. Interrupts are enabled/disabled using the interrupt configuration registers at 0x2B and 0x2C. The function of the proximity sensor (PS) interrupt can be set to either assert with each reading beyond a threshold (state), or to assert the first time a value exceeds a threshold (state change).

An interrupt asserts by pulling the INTB pin low. For instance, the controller can attach an ISR to a falling transition on the interrupt pin. When a status register is read, the status bit is cleared and any interrupt is also released. Reading the FIFO will also clear an AFULL interrupt.

For light sensor and proximity operation, sample rates are slow enough that the controller can typically poll data on schedule and check for new data via the status register.

Programming tip: There are two clocks on the chip. There is a (high power) 1% accurate one for Bio and a (low power) low accuracy clock for PS and LS. So don't expect the LS and PS sample rates to be more than 10 or 20% accurate. In other words, if you set a polling rate and expect new data every time, you might not get it, or you might miss a sample occasionally. This is no problem for typical LS and PS applications.

3.2 Light Sensor and Proximity Sensor Data

Light sensor (LS) and proximity sensor (PS) data is unsigned. (So is the FIFO data). So all data from OB1203 is unsigned.

PS data is 16 bit. The first byte is the lowest significant byte (the lower digits). (That is consistent throughout the data registers and FIFO.)

Consider an I2C block read function with the following prototype where unsigned char is equivalent to uint8_t data type:

i2c_read(unsigned char device_address, unsigned char start_register, unsigned char *dataBuffer, unsigned char bytes_to_read)

The following operation reads two bytes into the array psDataBytes.

```
i2c_read(OB1203_ADDR,0x02,psDataBytes,2);
```

The PS data is constructed by casting as unsigned integers and appending the first byte to the second.

```
uint16_t ps_data = ( ((uint16_t)psDataBytes[1]) << 8 ) | uint16_t)psDataBytes[0];
```

Similarly, the light sensor data for the LS channels is constructed by collecting three bytes from desired channel, or block reading all of the bytes. As an example the following function reads all the light sensor data into a buffer lsDataBytes.

```
i2c_read(OB1203_ADDR,0x04,lsDataBytes,15);
```

The LS data from all the channels can be collected into an array ls_data by casting as unsigned integers and appending the data bytes.

```
for(int n=0; n<5; n++) {
      ls_data[n] = ( ((uint16_t)lsDataBytes[3*n+2]) << 16 )
       | ((uint16_t)lsDataBytes[3*n+1])<<8
       | ((uint16_t)lsDataBytes[3*n];
}
white = ls_data[0];
green = ls_data[1];
blue = ls_data[2];
red = ls_data[3];
comp = ls_data[4];
```

The comp or dark channel reports how many counts were subtracted respectively from each of the light sensor channels to compensate for leakage current in the photodiode—a temperature dependent affect. “Comp” is a reference photodiode shielded by metal so it isn’t sensitive to light and measures only temperature-dependent leakage current. Note that the subtraction is automatic and already included in the data. If the algorithm does not need to know the comp value, you can omit reading it and spare some cycles.

The register map is designed so that a controller may read the status, ps and ls data in a single burst read for subsequent parsing.

Calibration note: The proximity sensor value is scaled digitally by a factory trim code in register 0x42 (or 0x43 if using the red LED). If the LED is particular bright, the measurement is scaled down (trim < 1). The saturation count value is proportionally reduced, so do not expect all devices to have the same saturation value. And avoid setting proximity thresholds within about 60% of the nominal saturation value. This is not a problem for most applications. The proximity sensor uses the same ADC as the PPG (heart rate) sensor, but the trim code is not intended for us in PPG mode, where the operation near the saturation is desired for autogain algorithms. **For PPG mode (heart rate or SpO2), write 0x00 to register 0x42 and 0x43. Writing to the power on reset will restore the factory trim. Or, you can first read the original trim code on power-up and write it back when you switch to proximity mode.**

3.3 Settings Registers (0x13–0x37)

3.3.1. Hidden Registers

The settings registers contain various bits which can be set or cleared to adjust the measurement parameters. The most important thing to note is that **some registers contain bits which have a default setting that must be kept**. For example, writes to registers 0x19, 0x2E and 0x35 should write ones to the bits noted in the data sheet. These registers include essential information for the proper working of the chip. So make sure the bits in the register map marked "1" are set that way using a mask.

Programming tip: Aside from the "1's" in the hidden bits for registers 0x19, 0x2E and 0x35, zeros can be written to all unlabeled bits in the register map. Also, don't write ones to any of the three LSBs of register 0x2F as these bits relate to disabling ambient light cancellation.

3.3.2. Modes of Operation (how to turn it on)

OB1203 starts up in a low power mode with no measurements enabled. This default standby mode is a convenient feature, as no special register write is required to wake or sleep the chip. Whenever no measurements are enabled, OB1203 is automatically in a low power standby mode. To change from standby, simply enable either the proximity and/or ambient light measure, or bio measurement.

To start measurements the main configuration registers (0x15 and 0x16) the respective measurement enable bits must be set. When Bio (PPG) mode is enabled, LS and PS measurements are not performed. Mutually exclusive modes of operation for PPG and LS/PS is not a concern because if someone's finger is on top of the sensor for a bio measurement, the LS and PS measurements are not particularly useful. Details of the mode settings options are in the datasheet.

The SW reset bit in MAIN_CTRL_0 is particularly useful as this returns all internal registers to the power-on defaults.

Programming tip: Before beginning a new mode of operation such as switching from PS to Bio mode it is good practice to set the SW reset bit, wait for POR status to assert and then set the registers for the new programming mode. Note that this does not assert the power-on status register bit. To return to a low power mode without altering the other registers, simply de-assert the LS_EN *and* PPG_PS_EN bits (bit 0) *of the* MAIN_CTRL_0 *and* MAIN_CTRL_1 *registers (0x15 and 0x16)*.

3.4 FIFO Registers

The FIFO has a 32 slot RAM that is 3 bytes wide (stores 32 numbers 3 bytes long). **FIFO data** must be read with a **repeated start as described in the datasheet**. This is very important.

Programming tip: It is good practice to set the FIFO read and write pointers to zero when beginning a new measurement.

There are typically four different ways of reading out the FIFO.

1. Polling: read one (regular fast readout)
2. Polling: read some (irregular readout)
3. Polling: read all (low data rates only)
4. Interrupt driven (regular readout) – **preferred**

Polling is useful when the interrupt pin is not used due to pin limits or preference, or an I2C-USB translator like an Aardvark. Interrupt-driven readout is preferred.

In any case, for PPG2 mode (IR and red) only readout even number of samples, namely 1 each of red and IR: 2×3 = 6 bytes, 4×3 = 12 bytes, 6×3 = 18 bytes, etc. Otherwise you could screw up which samples are red and IR and that is not good.

3.4.1. Polling

Polling: read one

For a controller without other obligations, the data can be read as soon as it is available. The controller periodically polls the status register (with some delay between polling—you can't stay on the I2C line all the time or the chip might not get a chance to update the FIFO). When a new PPG data is available the MCU burst reads 6 bytes (PPG1 mode) or 9 bytes (PPG2 mode) from register 0x38 up (0x38, 0x39, 0x3A and 3 or 6 times 0x3B). Byte 1 is the write pointer location. Byte 2 is the read pointer location in the RAM. Byte 3 is the overflow pointer (counts the number of samples you missed reading after the FIFO filled up). The next 3 bytes are data bytes from the first sample in the FIFO at the read pointer location for the IR LED. The final 3 bytes (in PPG2—namely SpO2—mode) are for the red LED sample. The read pointer now increments to the next sample which will be available soon—keep checking the status register, but not continuously. If I2C is always busy, value might not update.

Polling: some samples

Allow several samples to accumulate in the FIFO then burst read the read pointer, write pointer and overflow counter. Calculate how many samples are available (the distance between the read and write pointer: write pointer – read pointer). Note that the FIFO is a circular buffer, so if the read pointer is 31 and the write pointer is 02, then there is one unread sample at 31, at 0 and at 1. (write pointer – read pointer + 32). Now go back and read the FIFO samples that are available—do it before the write pointer catches up to your read pointer and overwrites your data.

Programming tip: If FIFO overflow register shows "n" samples are missing due to FIFO overflow, just clone your most recent sample n times in the algorithm. Most algorithms are robust to lost samples when the present value is copied forward. But if samples are not cloned the heart rate value will appear to accelerate according to the percentage of samples lost (more beats per samples).

For more time between readout allow several samples to accumulate in the FIFO then burst read the read pointer, write pointer, overflow counter, and a fixed number of samples (say 2 each for IR and red). Now calculate how many samples were available. Say there were 4 available. That means you could have read 4 instead of 2. So next time read 4 samples. And so on. You can put a brake in the algorithm to prevent oscillation. Or, simply index the number of samples read up or down depending on if more or less samples were available.

Polling: read all

First check how many samples are available by reading the read and write pointers, then read all available samples. At high data rates you may be reading samples while the next sample comes in. If so, the sample will be lost and the overflow pointer will increment. You and the ASIC have to share the FIFO. When you are reading, the ASIC cannot be writing. Otherwise data could be corrupted. Another way to do this is to read all available samples up to a limit, based on the timing. If the read and write pointer are read and there is some delay before samples are read, the write pointer may move and overwrite old data. The controller will then read new samples thinking they are old ones. This will make the data look "spikey". Avoid this by minimizing delays between the two reads and/or reading more frequently to keep the read and write pointers closer and avoid the read pointer getting lapped.

3.4.2. Interrupt-driven (preferred)

Data ready interrupt method: The data ready interrupt is enabled. Each time new PPG data is ready (both IR and R have to finish in PPG2 mode) the interrupt asserts. Controller then reads the new data immediately. This method has small, but frequent I2C traffic and requires low latency in order to not miss samples if data is coming at high rates.

A_FULL interrupt method: The A_FULL interrupt is enabled. When the FIFO is full (or nearly full—you can set how early you want it to assert), the A_FULL interrupt asserts. The controller then reads a fixed number of samples. Optionally the controller can read the FIFO registers and adjust the number of samples it reads if it is falling behind.

The interrupt-driven method has the advantage that the write and read pointer need not to be read. That reduces the I2C traffic and the chance that a measurement gets lost due to a FIFO busy with I2C communication. Interrupt driven readout with the FIFO_A_FULL interrupt with “read all” becomes regular in time, where the amount of time between interrupts is determined by the FIFO_A_FULL value in register FIFO_CFG. For example in SpO2 mode with a data output rate of 100Hz, setting FIFO_A_FULL to 0 will result in the A_FULL interrupt asserting every 16 samples, or 160ms. Whereas, with FIFO_A_FULL setting of 0x0C will asserting with 10 samples each of IR and Red in the FIFO. If all are read before the next sample arrives, then the FIFO_A_FULL interrupt will assert every 100ms.

3.5 FIFO Read Example (A_FULL interrupt case)

```
DigitalIn intb(INTB_PIN); //setup a pin to read the interrupt line

uint16_t ppgMode = 2; //SpO2 = 2, HR = 1;
uint16_t samples2Read = 4;
unsigned char fifoDataBytes[maxSamples2Read*3*ppgMode]; //3*#meas
unsigned char bytes2Read = 3+samples2Read*ppgMode; //FIFO registers+3*#meas
uint32_t ppgData[maxSamples2Read*ppgMode]; //parsed data buffer
unsigned char samplesAvailable;
unsigned char readPointer;
unsigned char writePointer;
unsigned char overflowCounter;

while(1) {
if(!DigitalIn.read()) { //AFULL if pin is low (optionally use falling interrupt and ISR)
      i2c_read(OB1203_ADDR,0x38,fifoDataBytes,bytes2Read);  //read  FIFO  registers +data
      readPointer = fifoDataBytes[0];
      writePointer = fifoDataBytes[1];
      overflowCounter = fifoDataBytes[2];
      if (writePointer<readPointer)
            samplesAvailable = (writePointer + 32) – readPointer – samples2Read;
      else
            samplesAvailable = writePointer – readPointer – samples2Read;
      for( int n=0; n<samples2Read^PPGmode; n++) {
            uint32_t byte0 = fifoDataBytes[2+3*n]);
            uint32_t byte1 = fifoDataBytes[2+3*n+1];
            uint32_t byte2 = fifDataBytes[2+3*n+2];
            ppgData[n] = (byte2<<16) | (byte1<<8) | byte0;
      }//end data parsing loop. If SpO2 mode every other sample is red.
}//end interrupt asserted case
//optionally include a loop delay if not using interrupt and ISR
}//end while loop
```

4. Example Part Configurations

Setting up OB1203 consists of a power-on reset, setting thresholds, interrupts, settings and then turning on the measurement.

4.1 Configuring LS

reset(); //reset registers to power-on default

setLSthresholds(); //set thresholds for LS interrupt behavior

configInterrupts(); //set LS interrupts and persistence as desired

configLSsettings(); //set LS gain, resolution and sample rate

configMain(); //turn on LS measurements

beginLSPollingThread(); //poll at about the sample measurement rate

Register setting example:

Register, Value //comment

//LS mode settings

15, 03 //LS enabled, all channels

16, 00 //PS and Bio off

22, 22 //18 bit (100ms) measurement, 100ms measurement period

23, 03 //Gain mode 18 (most sensitive)

//LS interrupt upper threshold

24, FF //upper threshold LSB (set to max)

25, FF //upper threshold middle byte (set to max)

26, 07 //upper threshold MSB (set to mid-scale) –interrupts if above x7FFFF

//LS interrupt lower threshold

27, FF //lower threshold LSB (set to 255 counts) –interrupts if below x000FF.

28, 00 //lower threshold middle byte (set to min)

29, 00 //low threshold MSB (set to min)

//interrupt config

2A, 00 //not using LS variation threshold

2B, 01 //interrupt config 0: white channel (default), LS var mode off, LS (threshold) interrupt enabled

2D, 20 //LS_persist = 2 (3 consecutive LS values out of threshold range)

Programming Tip: 50ms integration time is an even number of 50Hz and 60Hz power cycles, a good choice for avoiding light source flicker.

4.2 Configuring PS

reset(); //reset registers to power-on default

setPSthresholds(); //set thresholds for PS interrupt behavior

configInterrupts(); //set PS interrupts and persistence as desired

configPSsettings(); //set PS gain, LED current, LED pulse width, pulse number and sample rate, analog cancellation, digital cancellation

configMain(); //turn on PS measurements

beginPSPollingThread(); //poll at about the sample rate

4.3 Configuring LS and PS

Similar to above.

Register setting example:

Register, Value //comment

16, 80 //power on reset—resets factory trim for proximity sensor. Wait 10ms before next I2C write.

(wait 10 ms)

15, 03 //enable LS mode, all channels

16, 01 //PPG/PS enabled, PS mode

17, FF //LED current LSB (set to max)

18, 01 //LED current MSB (set to mid scale) LED current = 511 (125mA of max 250mA)

19, 1A //Analog cancellation off, 4 pulses, obligatory hidden register bit 1 set as required

1A, 14 //pulse width 35µs, measurement period 50ms (20 sps)

1B, 96 //digital cancellation set to subtract 150 counts of crosstalk

1C, 00 //digital cancellation MSBs (set to min)

1D, 00 //moving average hysteresis not used

1E, 00 //PS low threshold lower byte (zeroes)

1F, 04 //PS low threshold upper byte (set to 1024)

20, 00 //PS upper threshold low byte (zeroes)

21, 08 //PS upper threshold upper byte (set to 2048)

22, 32 //LS resolution 17 bit (50ms integration), measurement period 100ms

23, 00 //Gain 1x (widest range)

24, FF //LS interrupt high (maxed out—not using)

25, FF //ditto

26, 0F //ditto

27, 00 //LS interrupt low (minimized—not using)

28, 00 //ditto

29, 00 //ditto

2A, 05 //LS var interrupt set to 256 count change

2B, 03 //white channel selected for interrupt and variance mode for interrupt, interrupt enabled

2C, 01 //PS interrupt logic mode: state change, PS interrupt enabled

2D, 02 //LS persist 0, PS persistence: 2 consecutive samples above/below threshold

2E, 09 //PS gain 1x, obligatory hidden register bits 3 and 0 set

2F, 40 //PS power save on, LED flip off

4.4 Configuring PPG

reset(); //reset registers to power-on default

configPPGsettings(); //set PPG gain, LED pulse length, number of averages and sample period

setPPGcurrents(); //set IR and R led current levels

configuInterrupts(); //set conversion or AFULL interrupt as desired

resetPointers(); //set write, read and overflow to 0x00;

setDigTrim();//turn off proximity sensor digital trim to allow max range

configMain(); //turn on PPG1 measurements

beginPPGPollingThread(); //poll status register or FIFO pointers or wait for interrupt

Programming tip: If any PPG configurations are changed, such as the LED currents, reset the pointers, then disable and re-enable PPG measurements to start a new conversion.

Register setting example:

Register(hex), Value (hex) //comment

15, 02 //LS mode set to all channels (not used because LS is disabled)

2B, 00 //LS interrupts not used because LS is disabled

2C, 20 //FIFO almost interrupt enabled

2E, 0A //Gain 1x, obligatory hidden register bits 3 and 0 set

2F, 04 //PPG power save on, LED flip not set (IR first)

30, AF //IR current set to x1AF

31, 01 //ditto

32, AF //Red current set x1AF

33, 01 //ditto

34, 00 //PPG analog cancellation off for both channels

35, 4A //averaging 16 samples, obligatory hidden register bits 3 and 1 set

36, 31 //min PPG pulse width, 0.625ms measurement period (1600 sps for both LEDs—100sps with 16x averages)

37, 04 //rollover enabled, AFULL interrupt with 4 samples remaining (40ms advance warning).

38, 00 //FIFO write pointer=0 (start new samples from zero

39, 00 //FIFO read pointer=0 (haven't read any samples yet)

3A, 00 //FIFO overflow=0 (reset overflow counter of samples missed due to full FIFO)

42, 00 //set digital trim to 1x for IR channel

43, 00 //set digital trim to 1x for R channel (remember to undo when switching to proximity

16, 05 //PPG/PS mode enabled, SpO2 mode (PPG2)

5. Data Rate and Current Consumption

5.1 LS Data Rate and Resolution

Light sensing is the lowest power operation. The ADC is a first order sigma delta so resolution depends on how long the measurement runs. For display backlight applications high resolution is only needed in low lighting conditions. Otherwise quick, low resolution measurements suffice. Longer conversions (higher resolution) makes for a smaller least significant bit (LSB), which lowers the detection limit (improves sensitivity).

For example LS readout every 50ms allows for 50ms conversions which delivers 17-bit resolution. For sampling every 50ms any conversion time less than or equal to 100ms can be used. The device is simply idle between measurements. A modest, but insignificant amount of power can be saved with shorter conversions. The LS ADCs are the limiting factor on power consumption.

5.2 PS Data Rate and Resolution

Proximity sensing uses short pulse trains of IR light from an LED to check for nearby objects which reflect a tiny amount of light back to the receiver. Low overall power is achieved by the low duty cycle of the measurement.

To maximize efficiency always use the largest available practical LED current. (2× LED current = 2× SNR)

To minimize sensitivity to ambient light flicker use short pulse lengths.

To improve resolution and SNR use longer pulse lengths. SNR improves more slowly with longer pulse lengths than higher LED currents. (2× pulse length <= 1.4× SNR)

To improve SNR (and increase power) increase the number of pulses per measurement (this should only be done after first maximizing the current). (2x number of pulses <= 1.4× SNR)

To reduce overall power consumption minimize the sample rate (how often a PS measurement is performed). Usually 10–20 samples per second is sufficient for mobile applications.

To reduce overall power consumption consider using a Bayesian, or 2-step technique where if an interrupt is triggered by a low power, high noise measurements, the data rate, power, number of pulses, etc. can be quickly set to a high level by the MCU to “check” if the interrupt was real. The threshold should be adjusted such that the amount of false interrupts is low enough to achieve overall power savings with high reliability.

LED avg current = LED current setting × (125mA / 511) × pulse length [sec] × number of pulses / sample period [sec]

5.3 PPG Data Rate and Resolution

PPG data rate = 1/ (PPG period * number of averages)

A high data rate is useful for implementing discrete time digital filters. Lower data rates allow for less controller interaction. To maximize SNR the largest LED current should be used with the lowest ADC gain and highest number of averages. To reduce ambient light flicker artifacts use short LED pulse lengths. SNR improves more quickly with increased averaging than it does for increased pulse lengths.

If SNR is a concern, LED currents less than 62.5mA (0x0FF) should not be used to avoid the LED driver noise from becoming the limiting noise source.

Lower data rates ~16 sps are useful for heart rate measurements. SpO2 typically requires at least 100sps.

The LED duty cycle is below 50% for PPG1 and below 25% per LED for PPG2 mode, since half the time the device is sampling the ambient environment and half the time the LED is on. In PPG2 mode the IR and R LEDs alternate.

LED avg current (per LED) = LED current setting × (125mA/511 or x3FF) × pulse length[sec] / sample period[sec]

5.4 Autogain Methods

Optimal performance of the PPG sensor is discussed in a separate app note. The key points emphasized there are that maximizing the LED current (minimum gain) and maximum sample rate are the most useful methods.

A typical PPG host controller will use a gain control method to set a target count range at about 80% of the ADC full scale. Or about 200,000 counts.

In this method the controller should use the following, or similar procedure using the PPG interrupt mode or poll for new data.

1. Disable PPG1/2 measurements.
2. Program new current values for IR/Red channels.
3. Set FIFO read, write and overflow bits to zero.
4. Enable PPG1/2 measurements.
5. Wait for ADC interrupt or periodically poll the status register for new PPG data.
6. Read 3/6 bytes from FIFO and construct PPG count values.
7. Based on the measurement, calculate new IR/R current settings.
8. For each IR and R measurement, if the count values are within a narrows target range of counts, for instance 190,000 to 210,000 counts, then increment the respective a persistence counter.
9. If the persistence value meets the target, exit autogain for that channel.
10. If the value is outside a wider range of counts, for instance 170,000 to 230,000 then re-enter autogain for that channel.

Once autogain is complete, the usual desired readout method using the FIFO can be set, for instance FIFO_A_FULL interrupt.

Procedure details: It is recommended to use PPG interrupt during autogain. When interrupt asserts, turn off measurements and PPG interrupt, read the latest red and IR samples from the FIFO, calculate new LED settings, write the new LED current settings, reset the FIFO pointer to zero, turn on PPG interrupt and restart PPG measurements. This allows for the fastest possible AGC loop. Note, samples will not be coming at the desired data rate until AGC is complete. When AGC is complete, switch to FIFO_A_FULL interrupt mode.

Stability Note

Methods for calculating the new Red/IR current settings include simple increment/decrement methods or scaling the current linearly with the counts. Either method can work, provided the steps are larger than the differential nonlinearity (DNL) of the current source. For stability, the minimum step size in the loop should be no smaller than the decimal integer 32. E.g., do not increment current like this:

ir_current++; // DO NOT DO THIS

Rather, use larger steps:

Ir_current += 32; // DO THIS

6. Revision History

Revision	Date	Description
1.4	Jan.22.21	Removed Section 6.
1.3	Oct.20.20	Updated FIFO read and interrupt methods.
1.2	Jul.30.20	Updated trim on/off procedure and simplified code example.
1.1	Apr.30.20	Redefined document title.
1.0	Apr.23.20	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 skilled in the art 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 for development of 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 out of 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.

(Rev.1.0 Mar 2020)

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/

Trademarks

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