PHILIPS Sensing Platform

DG1903 PSP Biosensing by PPG Integration Guide

Approved

Rev. 1.1 — 9 Feb 2023

Philips Confidential

Document information

Info	Content
Keywords	Philips Sensing Platform, PSP, PPG, Library
Abstract	Integration Guideline for the PSP Library

PHILIPS

Revision history

Rev	Date	Author	Description
1.0	20 Oct 2021	JaWo	Initial version
	24 May 2022	JaWo	Update Legal Information
	8 December 2022	StSw	Add crosstalk/shortcut and SNR measurement
1.1	9 February 2023	StSw	Updated for PSP6.1.8, added firing sequence, multi-stream handling, wavelength requirement

Table of Contents

1 Introduction

Philips Sensing Platform library offers robust and accurate healthcare solution in wrist-worn devices. The solution is supported by Philips' deep experience in advanced healthcare biometrics technology.

PSP library provides a wide variety of metrics as specified in [SY1902]. System integrators can license specific metrics from Philips and integrate into their products.

1.1 Purpose

The purpose of this document is to describe how system integrators can integrate the PSP Biosensing by PPG product to their target platforms.

1.2 Scope

The scope of this document relates to PSP Biosensing by PPG integration. It applies to PSP 6.1. Newer versions of the library might not be fully covered by this version of the document. Certain peripheral functionalities, like LED power control, Analog Front End driver and accelerometer driver, are to be implemented by the system integrators and will not be fully covered in this document. However, where such components impact the usage of the PSP library, the required interactions with the library are explained along with examples of usage where applicable.

In this document, both hardware and software aspects specific to Biosensing by PPG are discussed. The generic PSP library integration details can be found in [DG1901].

1.3 References

Document ID	Title	Version
SY1901	PSP API Specification	1.2
SY1902	PSP Metrics Specification	1.2
DG1901	PSP Library Integration Guide	1.2

1.4 Software Reference

File	Contents
psp.h	PSP functions & definitions
psp_map.h	PSP functions physical memory addresses
psp.c	PSP function implementations, connecting the prototypes in psp.h to the addresses in psp_map.h
fx_datatypes.h	datatype definitions
psp.bin	executable binary code (binary file format)
psp.hex	executable binary code (Intel Hex file format)

1.5 Definitions, Acronyms and Abbreviations

Abbreviation	Description
ACC	Acceleration signal obtained by an accelerometer
AD	Analog to Digital
ADC	Analog to Digital Converter
AHR	Average Heart Rate
API	Application Programming Interface
App	An application, especially as downloaded by a user to a mobile device.
Bluetooth	A standard for the short-range wireless interconnection of mobile phones, computers, and other electronic devices.
BT	Bluetooth
CPU	Central Processing Unit
DAC	Digital-to-Analog converter
EDA	Electrodermal Activities
HR	Heart Rate
IR	Infrared
IRQ	Interrupt Request
MCU	Micro-controller
PPG	Photoplethysmogram, the signal obtained by an optical heart-rate sensor.
PSP	Philips Sensing Platform
RAM	Random Access Memory
RHR	Resting Heart Rate
SpO2	Peripheral oxygen saturation. This reading indicates the amount of oxygen being carried by red blood cells

2 Specific Guidelines for Biosensing by PPG

2.1 Recommendation for Multi-color optical design

For measuring SpO2, red and infrared LEDs are added to the optical design, so called multi-color design. Fig 1 shows an optical layout example using two photodiodes and different LED colors and positions. Depending on the mechanical possibilities and customer preferences, this example design could be used in the design-in.

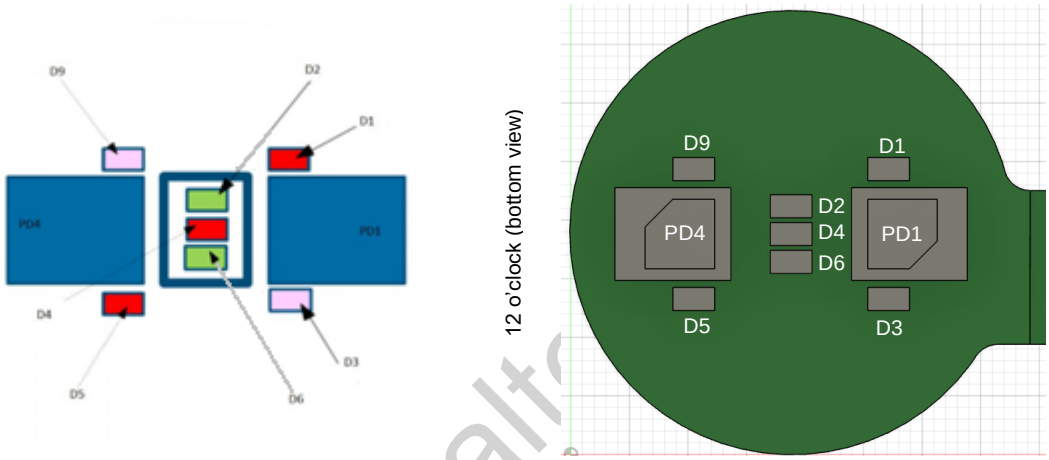

Fig 1. Philips multi-color example design with 2x photodiodes and 7x LEDs.

In Fig 1, the legend is as follows:

D1, D4, D5 = Red LED
D2, D6 = Green LED
D3, D9 = IR LED
PD1, PD4 = Photo diodes

2.1.1 Firing sequence

Depending on the AFE used, a certain number of measurements can be done during one sample period. In case of the MAX86178, 6 measurements can be done. The sample frequency required as input for the PSP6.1 is 32Hz (31.25 msec interval) and the pulse duration of each LED is 117 µs. Please check the AFE datasheet for details on timing. Note that for the SpO2 measurement, the firing of red & IR should be close together.

In case of the example design from Fig 1, the firing sequence would look as follows:

- an SpO2 measurement follows the sequence of using the red + infrared + green + ambient in combination with each photo diode. This reflects in the following order:
  M1, M2, M3, M4, M5 and M7 at each sample period(1/32s).

- an HR (or related) measurement follows the sequence using the green + motion + ambient in combination with each photo diode. This reflects in the following order:
  M5 and M7 at each sample period(1/32s).

Measurements:
M1= PD4 with D1 ('Red top')
M2= PD1 with D5 ('Red bottom')
M3= PD1 with D9 ('IR top')
M4= PD4 with D3 ('IR bottom')
M5= PD1 + PD4 with D2 & D6 (center green)
M7= PD1 + PD4 (ambient)

2.1.2 Multi-stream handling

As of version PSP6.1.8*, it is possible to feed two, instead of one, separate data streams for PPG_GREEN, PPG_Ambient, PPG_RED and PPG_IR to the library. Internally, these separate data streams are processed by a sophisticated algorithm instead of being averaged.
* For PSP6.1.0 – PSP6.1.7 versions, one data stream for PPG_GREEN and two for PPG_RED and PPG_IR are enabled.

2.1.3 Wavelength and current requirement:

Green min: 515nm, max:535nm, Typical:525nm
Red min: 655nm, max:665nm, Typical:660nm
Infrared min: 870nm, max:890nm, Typical:880nm

The wavelength requirement for Green and Red is quite standard, which is 525 and 660nm, respectively. The choice of the wavelengths is critical for SpO2. SpO2 is determined by the ratio of signals measured with RED and IR light. The absorption levels are different at different wavelengths. The wavelengths that were chosen (880nm for IR and 660nm for Red) seem to deliver the best signals. For IR the sensitivity in the wavelength difference is less sensitive, that is why you see 940nm sometimes is being used. It results likely in slightly different calibration parameters used for the calibration of the algorithm. Deviation of the 660 nm (Red) is much more influencing the calibration and sensitivity of SpO2 than the IR 880-920 choice.

The LEDs in our example design have been selected to have a high-power efficiency which provide high radiant intensity output (mW/sr) and stability of the wavelength especially for the Red LED. The system should provide enough current to drive the LEDs to operate in their linear region. We also recommend using a booster for the LED signals (check LED datasheet). The maximum LED current (for this particular example design) is set as follows:

128 mA for 1x Green LED (so 2x 128mA since there are two green LEDs)
64 mA for PPG Motion (Red LED closest to Green LED, so the center one).
128 mA for Red
128 mA for IR

2.2 Clock and Synchronization for PPG and ACC data

PSP library requires synchronized 32 Hz PPG and 32 Hz ACC input signals as input. The PPG Front End and Accelerometer do the sampling and provide the samples based on the clock sources. The ratio of PPG and ACC samples available may vary if there is drift between 2 clock sources. It is suggested the clock sources of the PPG and ACC sampling should share a common clock source, says a stable 32678Hz external clock.

In order to synchronize between the PPG and ACC signals, it is suggested to align with the interrupts from the PPG Analog Front End. The PPG Analog Front End is programmed to provide an interrupt to MCU when 32 PPG samples are available in one second. Then, transferring PPG and ACC data from PPG Analog Front End and Accelerometer can be conducted in the interrupt subroutine.

The Accelerometer is supposed to have 32 ACC samples when the MCU receives interrupt signal from PPG Analog Front End. However, the number of ACC samples may vary slightly due to the latency of the interrupt calls. Some Accelerometers may not have FIFO to contain the ACC samples in one second, so multiple interrupts from the Accelerometer are employed within a second to prevent FIFO data overflow.

The best scenario in the system would be all components (i.e. the PPG Front Ends, the accelerometer and host MCU) are synchronized with one 32678Hz clock source. Synchronization problem may occur if different clock sources are used for PPG Front Ends, Accelerometer and host MCU. This would cause the PPG signal and ACC signal having a drift in the sampling and drifting amount varies from device to device. Then it is difficult to control the consistency between different devices.

Just in case applying same clock source to PPG, ACC and host MCU is not possible, the PPG clock can be used as a reference such that the PPG resampling can be done as little as possible to preserve the signal integrity. It is recommended to align the ACC samples collected in the period to every 32 PPG samples. That is to interpolate the ACC signal (e.g. 31 samples) to 32 samples to align with 32 PPG samples.

Please note that if the PPG sampling rate is not correct and accurate, this would directly affect accuracy of those metrics requiring PPG signal, e.g. heart rate, heart beat time stamp etc.

2.2.1 Adjustment of Heart Rate Output

If the MCU clock is believed to be more accurate than the PPG and ACC clocks, the MCU clock can be used as an absolute reference. Since there is a drift between the MCU clock and the PPG clock, this drift may propagate to the output metrics, for example, Heart Rate calculation. Compensation should be applied to the output to improve the accuracy. Follow the formula below to adjust the heart rate:

Adjusted HR = Heart Rate reported from library / (T32sample in s);
T32sample is the time for PPG sensor to deliver 32 PPG samples. The T32sample can be obtained from the timestamp on the MCU when 32 PPG samples are received. For example, if the MCU takes 0.97s on average to receive 32 PPG samples, and the heart rate reported from the library is 100 bpm, then the Adjust HR should be 100/0.97 = round (103.09) BPM = 103 bpm.

However, using this method will not correct the accuracy of other metrics requiring PPG signal, especially heartbeat time stamp where the variation in sampling time will one to one reflect in the heartbeat time stamp.

2.3 Data resampling and data synchronization

The PSP library requires PPG signal input at a rate of 32 samples per second and an ACC signal input at a rate of 32 samples per second.

The system integrators may find the PPG Front End and Accelerometer is not providing any sampling rate selection that matches with the input requirements of the library specification. It is suggested to resample the PPG data and/or acceleration data in order to fit the required sampling rate.

For example, if a PPG Front End selected is providing 25 samples in one second, 32Hz PPG samples can be achieved by interpolating 25Hz PPG samples as illustrated in Fig 2.

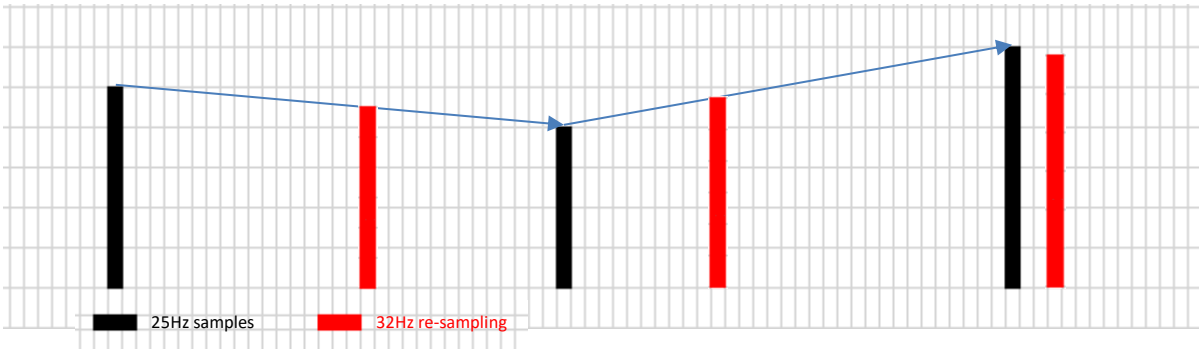

Fig 2. An example of resampling by interpolating PPG signals

With the same token, 32Hz accelerator signal could be achieved by resampling the accelerometer that does not match. Resampling ACC signal at higher sampling frequency to lower sampling frequency requires decent anti-aliasing filtering, otherwise, the performance of heart rate estimation is affected.

The best scenario is that the PPG Front Ends, the Accelerometer and the host MCU are synchronized with one 32768Hz clock source. However, some accelerometers may not get any external clock interface. This can be addressed by resampling of the ACC samples to 32Hz from different number of ACC signals due to the drift in accelerometer clock. In other words, let's PPG sensor dictate the time and ignores the accelerometer IRQ. Every (PPG) cycle a constant number of PPG samples will be retrieved and variable number of ACC samples will be read. These numbers should be dynamically resampled to the required 32/32 (PPG/ACC) Hz sampling rates.

For instance, if the PPG Front End is set to generate 25 samples per second and the Accelerometer is set to generate 120 samples per second, there would be 25 PPG samples and 120 ACC samples in one second period. PSP library has an interface that works on 32Hz PPG and 32Hz ACC. There is a mismatch in the sampling. The PPG samples will be linear interpolated to produce 32Hz samples while the ACC samples will be interpolated to nearest sampling rate which is a multiple of 32, in this case it is 128Hz, anti-aliasing filtered and then down-sampled to 32Hz. As a rule of thumb, either the PPG or ACC signals should be interpolated to nearest multiple of 32Hz, such as 32Hz, 64Hz, 128Hz, 256Hz etc, effectively anti-aliasing filtered and then down-sampled to 32Hz.

The 32 PPG and 32 ACC samples are then fed to the library via the SetMetric.

Following pseudo code segment illustrates conducting resampling of PPG and ACC from Ring buffers before feeding into PSP library. Ring buffers are used to handle the PPG and ACC data collected from the PPG and ACC interrupt calls.

```
ListRequiredMetrics ( &requiredMetricsList )
for ( each metricID in requiredMetricsList ) // PPG, acceleration, …
```

```
  If (time metric required)
    SetMetric(time metric, metricData)
  Else If (PPG metric required)
  // conduct interpolation and replication of PPG, AMB signal
    "Send PPG data from PPG ring buffer to PPG re-sampler"
    "Update metricData from PPG re-sampler"
    "Fill metricData with PPG gain and LED power"
    SetMetric(PPG metric, metricData)
  Else If (ACC metric required)
  // conduct interpolation and replication of ACC signal
    "Send ACC data from ACC ring buffer to ACC re-sampler"
    "Update metricData from ACC re-sampler"
    SetMetric(ACC metric, metricData)
  Else
    "gather metricData"        // exactly 1 value per required metric
    SetMetric(metricID, metricData )
end for
Process ( )                    // let PSP process anything it can
....
```

2.3.1 Interpolation of PPG Signal

The PPG Front end may provide PPG signals in different sampling rates from the required one for the PSP library. An efficient re-sampler for the PPG signal at different frequencies could be done by interpolation assuming no anti-aliasing filter or down-sampling is required. System integrators can use following function to cope with different sampling rate of PPG signal sources.

Below is an example code of how to interpolate the PPG signal.

```
/* Saturation unsigned 16-bit */
#define SATURATE_UINT16( X )   ( (X) < 0 ? 0 : (X) > 65535 ? 65535 : (X) )

/*-------------------------------------------------------------------------------------------------*
*                                                                                                  *
*  Name          : Interpolate                                                                     *
*                                                                                                  *
*  Description   : Generic linear interpolation from inLen to outLen                               *
*                                                                                                  *
*  Inputs        : *in       : input samples                                                       *
*                  inLen     : number of  input samples                                            *
*                  outLen    : number of output samples                                            *
*                                                                                                  *
*  Outputs       : *out      : output samples                                                      *
*                  *state    : save last sample for usage in next function call                    *
*                                                                                                  *
*  Comments      : convert inLen samples to outLen samples by interpolation                        *
*                  Last sample of *in and *out are aligned                                         *
*                   Initialise *state to zero at startup(first function call)                      *
*                  PPG signal is 16 bit unsigned int                                               *
*                                                                                                  *
*-------------------------------------------------------------------------------------------------*/
FX_VOID Interpolate
(
   FX_UINT16 * const in,
   FX_SINT       const inLen,
   FX_UINT16 * const out,
   FX_SINT     const outLen,
   FX_UINT16 * const state,
)
{
   FX_SINT32 y1;
   FX_SINT32 y2;

   FX_UINT32 y;
   FX_UINT32 x;
   FX_UINT32 x1;

   FX_SINT   i;

        /* convert from FIN Hz to FOUT Hz by interpolating the samples      */
        /* out = y1 + (y2 - y1) * (x - x1) / (x2 - x1)                      */
        /* projected on the FIN * FOUT grid                                 */
        /* this becomes                                                     */
        /* out = y1 + (y2 - y1) * (x - x1) / FOUT                           */
        x  = 0;
```

```
        x1 = 0;
        y  = 0;
        y1 = (FX_SINT32)(*state);
        y2 = (FX_SINT32)(in[y]);
        for (i = 0; i < outLen; i++)
        {
                FX_SINT32 tmpOut;

                x += inLen;
                while ( x > ( x1 + outLen ))
                {
                        x1 += outLen;
                        y++;
                        y1 = y2;
                        y2 = (FX_SINT32)(in[y]);
                }

                tmpOut = ( y2 - y1 ) * ( x - x1 );
                tmpOut /= outLen;
                tmpOut += y1;
                out[i] = (FX_UINT16) SATURATE_UINT16(tmpOut);
        }
        *state = (FX_UINT16)(in[inLen-1]);  /* save last sample for usage in next function call */
}
```

2.3.2 Resampling of Accelerometer Signal

The Accelerometer may provide ACC signals in different sampling rates from the required one for the PSP library. An efficient re-sampler for the ACC signal at different frequencies could be done by interpolation (and down sampling). System integrators can use following function to cope with different sampling rates of ACC signal sources.

Here below are some example configurations of the resampling by decimation (provide proper anti-aliasing effect) for ACC signals.

- with ~25Hz ACC input, use linear interpolation to get 32Hz signals
- with ~50Hz ACC input, use linear interpolation to get 64Hz signal followed by 2x down-sampler to get 32Hz signal
- with ~100Hz ACC input, use linear interpolation to get 128Hz signal followed by 4x down sampler to get 32Hz signal

When feeding the ACC signal to the PSP library, make sure the ACC signal is actually within +/-8g operating range by clipping the out-of-range signal.

Below are example implementations of interpolation and down sampling of ACC signals

```
/* Saturation 16-bit */
#define SATURATE_SINT16( X )    ( (X) < -32768 ? -32768 : (X) > 32767 ? 32767 : (X) )

/*-----------------------------------------------------------------------------------------*
*                                                                                          *
*  Name       : FILTER_Decimate_2x_SINT16                                                  *
*                                                                                          *
*  Description : Downsample input signal by a factor of 2 using an                         *
*                    efficient downsampler.                                                *
*                                                                                          *
*  Inputs     : nSamples : Number of input samples (multiple of 2)                         *
*               *in      : Input samples                                                   *
*               *out     : Pointer to array of nSamples/2 elements                         *
*               state    : Downsampler states                                              *
*                                                                                          *
*  Outputs    : out      : (nSamples/2) downsampled output                                 *
*                                                                                          *
*  Comments   : May be used in-place. Initialise states to zero at startup.                *
*                                                                                          *
*-----------------------------------------------------------------------------------------*/
FX_VOID Filter_Decimate_2x_SINT16
(
   FX_SINT16   nSamples,
   FX_SINT16   const in[],
   FX_SINT16   out[],
   FX_SINT32   state[2]
)
{
   FX_SINT c1;
   FX_SINT c2;
   FX_SINT32 tmp;
```

```
   for( c1 = 0, c2 = 0; c1 < nSamples; c1 += 2, c2++ )
   {
      state[1] = ( in[c1+1] - state[1] + ((FX_SINT32)state[0] * 2) ) / 2;
      state[0] = in[c1+1];
      tmp = state[1] + in[c1];

      out[c2] = (FX_SINT16)SATURATE_SINT16( tmp / 2 );
   }
}

/*-------------------------------------------------------------------------------------*
*                                                                                      *
*  Name       : FILTER_Decimate_4x_SINT16                                              *
*                                                                                      *
*  Description : Downsample input signal by a factor of 4 using an                     *
*             efficient downsampler.                                                    *
*                                                                                      *
*  Inputs     : nSamples : Number of input samples (multiple of 4)                     *
*             *in     : Input samples                                                  *
*             *out    : Pointer to array of nSamples/4 elements                        *
*             state   : Downsampler states                                             *
*                                                                                      *
*  Outputs    : out     : (nSamples/4) downsampled output                              *
*                                                                                      *
*  Comments   : May be used in-place. Initialise states to zero at startup.            *
*                                                                                      *
*-------------------------------------------------------------------------------------*/
FX_VOID Filter_Decimate_4x_SINT16
(
   FX_SINT16 nSamples,
   FX_SINT16 const in[],
   FX_SINT16 out[],
   FX_SINT32 state[4]
)
{
   FX_SINT c1;
   FX_SINT c2;
   FX_SINT32 tmp[2];

   for( c1 = 0, c2 = 0; c1 < nSamples; c1 += 4, c2++ )
   {
      state[1] = ( in[c1+1] - state[1] + ((FX_SINT32)state[0] * 2) ) / 2;
      state[0] = in[c1+1];
      tmp[0] = state[1] + in[c1];

      state[1] = ( in[c1+3] - state[1] + ((FX_SINT32)state[0] * 2) ) / 2;
      state[0] = in[c1+3];
      tmp[1] = state[1] + in[c1+2];

      state[3] = ( tmp[1] - state[3] + ((FX_SINT32)state[2] * 2) ) / 2;
      state[2] = tmp[1];
      tmp[0] = state[3] + tmp[0];

      out[c2] = (FX_SINT16)SATURATE_SINT16( tmp[0] / 4 );
   }
```

```
}

/*-------------------------------------------------------------------------------------*
*                                                                                      *
* Name       : Interpolate                                                             *
*                                                                                      *
* Description : Generic linear interpolation from inLen to outLen                      *
*                                                                                      *
* Inputs     : *in    : input samples                                                  *
*              inLen  : number of  input samples                                       *
*              outLen : number of output samples                                       *
*                                                                                      *
* Outputs    : *out   : output samples                                                 *
*                                                                                      *
* Comments   : convert inLen samples to outLen samples by interpolation    *
*              Last sample of *in and *out are aligned                                 *
*              Initialise *state to zero at startup(first function call)               *
*                                                                                      *
*-------------------------------------------------------------------------------------*/
FX_VOID Interpolate
(
   FX_SINT16 * const in,
   FX_SINT    const inLen,
   FX_SINT16 * const out,
   FX_SINT    const outLen,
   FX_SINT16 * const state,
)
{
   FX_SINT32 y1;
   FX_SINT32 y2;

   FX_UINT32 y;
   FX_UINT32 x;
   FX_UINT32 x1;

   FX_SINT   i;

        /* convert from FIN Hz to FOUT Hz by interpolating the samples      */
        /* out = y1 + (y2 - y1) * (x - x1) / (x2 - x1)                      */
        /* projected on the FIN * FOUT grid                                 */
        /* this becomes                                    */
        /* out = y1 + (y2 - y1) * (x - x1) / FOUT                           */
        x  = 0;
        x1 = 0;
        y  = 0;
        y1 = *state;
        y2 = in[y];
        for (i = 0; i < outLen; i++)
        {
                FX_SINT32 tmpOut;

                x += inLen;
                while ( x > ( x1 + outLen ))
                {
                        x1 += outLen;
                        y++;
```

```
                    y1 = y2;
                    y2 = in[y];
                }

                tmpOut = ( y2 - y1 ) * ( x - x1 );
                tmpOut /= outLen;
                tmpOut += y1;
                out[i] = (FX_SINT16)tmpOut;
        }
        *state = in[inLen-1];  /* save last sample for usage in next function call */
}
```

2.4 Output PPG/ACC metrics

In view of the importance of the PPG and ACC signal integrity, in the PSP library API, there are PPG/ACC output metrics. Those are intended mainly for testing, performance analysis and problem solving purposes, and are not part of the product offering from Philips.

The PPG and ACC signals are just mapping the input PPG/ACC metrics to the corresponding output metrics, but only at the moment at which the PSP library uses/takes over the input metrics value.

In fact, all input metrics can be retrieved at the output. There are also compressed PPG and ACC signals which are compressed versions of the PPG and ACC signals. The format is proprietary and not recommended to use. For details of those metrics please refer to [SY1902].

2.5 Verification tests

To verify the electrical and mechanical design providing sufficient optical quality of PPG signal for the library, some verification tests are recommended.

- PPG and ACC signal synchronization and integrity test (PASA)
- Crosstalk test
- SNR test
- Reflectivity test
- HR simulator test

2.5.1 PASA Test tool

Philips uses a test tool called PASA Test Tool (depicted in Fig 3) to test the PPG and ACC signal integrity. The DUT is placed on the platform which will be rotated periodically. If the PPG and ACC acquisition are correct, the signal feeding to the library should follow a synchronized sine wave for

the PPG and ACC signal. By analyzing the signal integrity, the driver implementation could be verified. Please contact Philips ITS Customer Support for details.

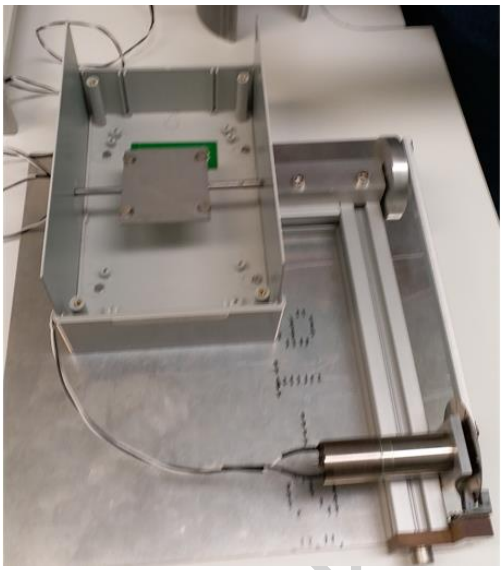

Fig 3. PASA Test Tool

2.5.2 Crosstalk / Shortcut test

Importance in the design of the optical sensor is the light path. Light should travel from the LED to the skin, and then back to the light sensor (Photodiode or CMOS sensor). Only the reflected light from the skin to the light sensor contains the heartbeat information. However, depending on the mechanical design, light could still travel directly from the LED to the light sensor. This is called crosstalk or shortcut.

The aim of the crosstalk test is to measure the amount of this (crosstalk/shortcut) light.
In practice the PPG count is measured when **firmly covering the sensor** with a black rubber patch.

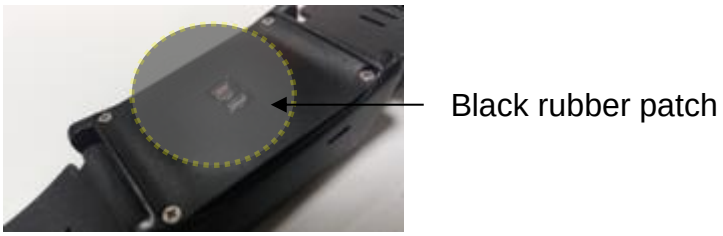

Fig 4. Black patch needs to cover the optical sensor firmly

Crosstalk always reduces the performance so it should be limited to the below described threshold to contain the performance influence.

Measurement:

- Cover the sensor by a black rubber, under 100% LED driven current and maximum Gain.
  Normally this happens automatically by the control loop.
- Measurement duration is about 30 seconds (output should be stable).
- Measure PPG_Count

Passing criteria:

- **Green LED** -> for HR related metrics:
  ADC_{max} = 19 bit (select applicable range)
  Range = 0.5 (=ADC range used)
  Min_{THD} = 0.4 (40 - 90 % range)
  Modulation acceptable = 5% (Mod_{diff} = 0.05)

  shortcut < ADC_{max} * Range * Min_{THD} * (Mod_{diff} / (1- Mod_{diff}))

  [2^19 * 0.5 * 0.4 * (0.05 / (1-0.05)]

  The ppg count for Green should be <5K out of 2^19

- **Red & IR LED** -> for SpO2:
  ADC_{max} = 19 bit (select applicable range)
  Range = 1 (=ADC range used)
  Min_{THD} = 0.5 (50 - 90 % range)
  Modulation acceptable = 0.25% (Mod_{diff} = 0.0025)

  Short < ADC_{max} * Range * Min_{THD} * (Mod_{diff} / (1- Mod_{diff}))

  [2^19 * 1 * 0.5 * (0.0025 / (1-0.0025)]

  The ppg count for Red & IR should be <600 out of 2^19

2.5.3 Signal to Noise (SNR) test

The aim of the Signal-to-Noise Ratio (SNR) test is to measure the amount of noise imposed on the desired light from the LEDs reflected on the light sensor. The higher the SNR the better small signals (low perfusion index) can be measured. The SNR is also often described in the AFE specification by the AFE manufacturer. Typically, it is around 90 dB @ 10 µA sensor current.

The test setup consists of a black box that can be closed, this is to avoid ambient light. Inside the black box there is a reflector that can be moved to increase/decrease the distance between the reflector and the Device Under Test (DUT). It is also important that the setup is mechanically stable to prevent vibrations. Although optional, placing all equipment inside a metal box would create an even better EMC compliant solution. Fig 5 shows a picture of the SNR test setup. Optimal settings for e.g., Led Power and ADC gain (control loop) to carry out the SNR test can be discussed and aligned with the Philips ITS Customer Support team and the AFE manufacturer.

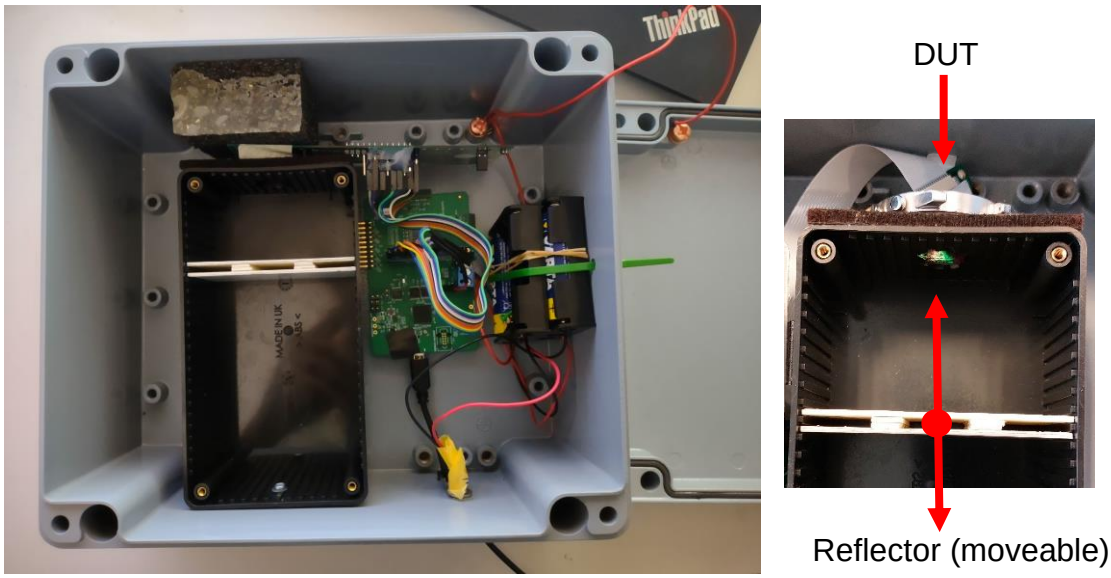

Fig 5. SNR test setup

Measurement:

In this example Philips SNR specification states minimum 90 dB @ 10 µA. For this the optical reflector in the test setup can be adjusted such that the PIN diode / TIA input current equals 10 µA. However, as it is difficult to tune the TIA current by setting the reflector distance, a different approach is taken. Measurements are done with the optical reflector at several different positions such that a ‘staircase shape’ current profile is generated. Then, in every horizontal level in the staircase, a time window of 3 seconds is selected. In each of these windows, the average value and standard deviation of the

signal is calculated. The SNR is then calculated as the ratio of mean value and standard deviation. To express SNR in [dB], the following formula is used:

SNRdB = 20 * log10(average(signal_in_window) / standard_deviation(signal_in_window))

From the above, a set of values have been generated for different sensor current I_TIA and SNRdB, then a curve fit SNRdB(Itia) and extrapolation can be applied. From this curve fit, the SNR at different currents can be derived. See Fig 6 below.

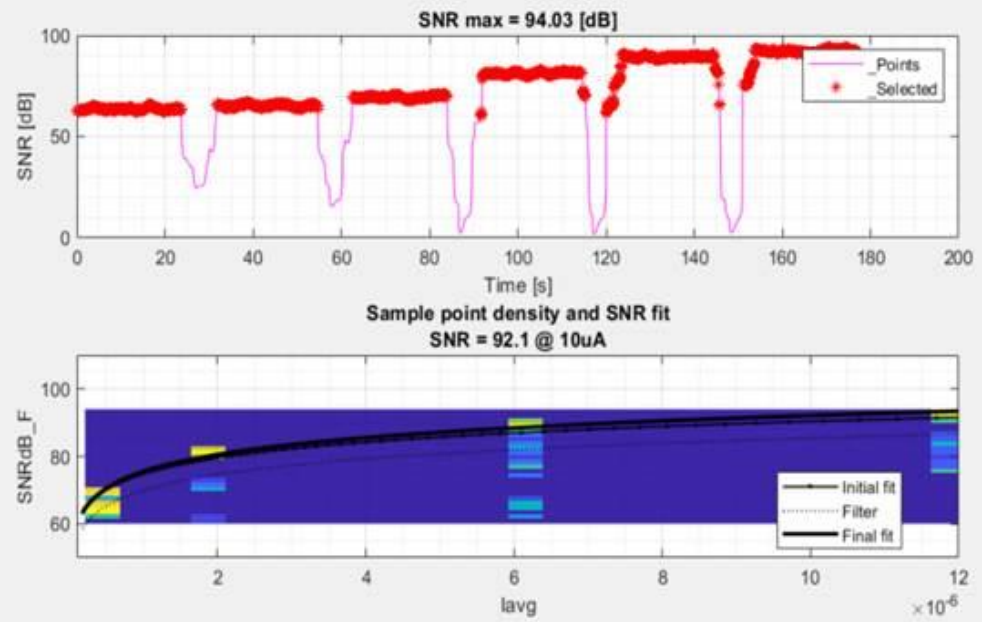

Fig 6. Upper graph shows the SNR per distance, lower graph shows interpolation of SNR.

Passing criteria:

- Check AFE datasheet to get reference SNR of corresponding Sensor-current
- SNR for Green channel >80dB at required operating current
- SNR for Red and Infrared channel >85dB at required operating current

2.5.4 Reflectivity test

The aim of the reflectivity test is to verify the amount of light from the LEDs reflected onto the light sensor. Normally this test is implemented on factory level to check e.g., for LED failures. This test could be combined with the SNR test.

Measurement:

Reflectivity = $I_{Photo\ Diode}$ / I_{LED} => [µA / mA]

Where:
$I_{Photo\ Diode}$ = $I_{Photo\ Diode\text{-}max}$ * ($ADC_{PPG\ level}$ / ADC_{max})
$ADC_{PPG\ level}$ = e.g. 20000 counts
ADC_{max} = e.g. 32786 counts (in case of 15bit ADC resolution)

Passing criteria:
Due to different architectures in the analog front end, the criteria for reflectivity test is different. The Reflectivity is a relative number. The passing criteria should be created with comparing the reflectivity in respect to a Golden Sample. The reflectivity also depends on the position of the reflective screen in the black box. The reflective screen should be calibrated and placed on a position such that the PPG Mean value for that Golden Sample is within a certain range.

2.5.5 HR simulator test

To quickly test the wearable's HR performance (still as verification test), a heart rate simulator, such as the HRS200 (Whaleteq) can be used. It's working principle is based on adding (modulated) light to the DC level of the wearable LED pulse to simulate a heartbeat signal. This signal is retrieved by the wearable photodiode as a simulated heartbeat modulation, see Fig 7. By placing the wearable on the device and selecting a BPM, the wearable should display the same value (±1BPM).

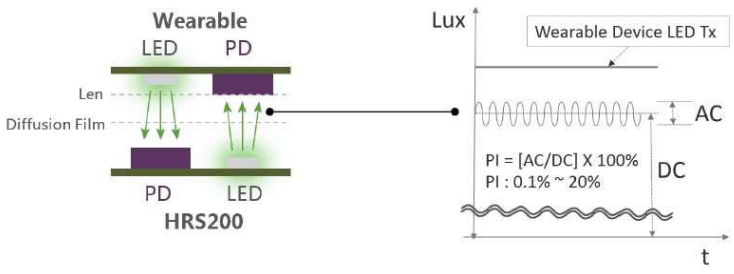

Fig 7. Heart rate simulator

2.6 Validation tests

Once the device has reached 'production quality' status, Philips recommends performing a proper validation test to get insight into the (heartrate and/or others) behavior of the device/wearable. Such a validation test consists of:

- Actual tests on humans (20 subjects; mixed gender, age, BMI, Skin tone et cetera)
- Following a pre-defined test protocol such as Google's WearOS 3.0
- Controlled circumstances…yet plenty of variables.

Testing: firmware, sensor, casing, strap, environmental, specific activity, benchmark,..

2.7 Raw Data Logging for PPG signal Analysis

In the course of design and integration of the PSP library, the problem of the system performance may arise from the defects in mechanical design, photodiode pickup, LED power and gain control, resampling process, potential buffering problems, PPG signal quality as well as the motion artifacts represented by the ACC signals. The raw data logging is a way in helping to examine problems in the PPG and ACC data providing to the PSP library. The resulting metrics logging, like the heart rate, is also helping in validation of the performance of the system.

2.8 Skin Proximity & Wearing Detection

It is advised to disable the output metrics in non-wearing periods, which will reset the metrics functionality to an initial known starting state and also save power.

The application may use the Skin Proximity metric to disable other output metrics during such non-worn periods (the Skin Proximity metric itself should stay enabled all the time, also during non-worn periods). Note that the PSP library itself does not enable or disable metrics in relation to Skin Proximity, this is the responsibility of the application.

If only PPG and ACC sensor are available (no skin conductance sensor), Skin Proximity transition from off-skin to on-skin is typically with a latency of around 30-35 seconds. This period should be long enough to reduce false on-skin values. Skin Proximity transition from on-skin to off-skin is typically with a latency from as little as 1-2 seconds, up to around 15-20 seconds, depending on various acceleration and PPG signal conditions.

In general, Skin Proximity provides reliable values when the device is either worn by the user (where the user may be moving or stationary), or stored in a stable, stationary position (e.g. on a bedside table or charging cradle). But it is important to notice that Skin Proximity has limitations under these known conditions:

1. false on-skin may occur in case the device is in motion and not worn but still facing a reflective surface, e.g. in a handbag; (This is the same situation as the device is worn on wrist if based on PPG and ACC signal only and hence additional sensor or manual switch off is required.)
2. false on-skin may occur in case the device is **NOT** in motion and not worn but still facing a reflective surface, e.g. put on a reflective table;
3. false on-skin may occur when the device is taken off under a dark environment.
4. off-skin may occur e.g. if the user is asleep in a posture hampering the detection of blood flow (e.g. body laying on the wrist wearing the device). (No PPG signal and hence no heart rate can be detected due to PPG method)

How Skin Proximity can be used strongly depends on the wearable device's intended use cases. If the limitations of Skin Proximity as mentioned above are not acceptable, then it is advisable to equip the device with other means (such as other sensors) to detect wearing/non-wearing, or to ask the user to manually switch the device's monitoring function on or off via a user interface (e.g. button). Also, in certain use cases, combinations of various methods are possible, e.g. let the user switch on the monitoring function with a button, and use Skin Proximity off-skin value to switch the function off again.

If skin conductance sensor is also available and Biosensing by EDA metrics are enabled, Skin Proximity will also take into account of skin conductance as input to improve the accuracy. The latency is expected to be the same.

It might also be possible to infer device wearing or non-wearing state from other output metrics (or combinations of metrics) of the PSP library, such as Activity Count (to detect certain amounts of motion or non-motion). The detection algorithm needs to be designed, implemented and validated by the system integrators. How feasible and useful this approach is, depends on the application use case specifics, and therefore differs per application.

2.9 Resting Heart Rate Extraction

The Resting Heart Rate (RHR) metric is extracted by collecting information during *periods of relative physical inactivity.* The information is aggregated and retained for up to seven days.

After first enabling RHR, its initial value is available after the metric's update interval of 1 hour. The PSP library will only give an initial value when the monitored person was sitting or lying calm and relaxed but not slept for at least 1 minute with good quality within that hour. Typically, the quality of initial value is still low, due to a limited amount of relevant information.

During periods of inactivity longer than 1 minute, the PSP library will request PPG only for maximally 1 minute if RHR metric is enabled. It then will wait for 15 minutes to request PPG again. So after

approximately one hour (with subject being inactive during that hour) PPG could be requested for four times but it will give only one RHR value per hour.

The quality of subsequent values will increase when cumulatively more relevant information is aggregated, depending on the monitored person's lifestyle pattern. In normal daily life wearing conditions, the RHR metric will give a stable value with Quality Index equals to 4 after approximately 12-15 hours. In case the monitored person was quite active, this can take up longer. The Quality Index will increase slowly from 1 to 4 in case it detects sufficient information in resting periods. The RHR information is aggregated and retained in history for up to seven days.

If RHR metric is disabled, no new information is collected anymore, and the aggregated information is gradually discarded as time progresses. When re-enabling RHR metric within seven days, the remaining retained information still contributes to the metric extraction but due to mentioned data discarding, the quality may be lowered. After seven or more days of disabling, all retained information is lost, and upon re-enabling, complete information will have to be collected anew. Disabling the RHR metric will not cause reset of the aggregated information. The aggregated information could be reset by re-initializing the instance of the PSP library or clearing aggregated information with a system reset.

The collection of RHR information has no connection with skin proximity and any other metrics. If RHR metric is enabled, the PSP library might measure values for RHR continuously even when the watch is not worn. The application should disable RHR metric when not worn and enable it again when worn.

The flow diagram indicates the conditions for enabling and disabling of RHR metric (could also apply to all other metric as well) depend on the wearing status. The value and quality of RHR metric is saved before updating with a new RHR metric with equal or higher quality is produced in next update interval.

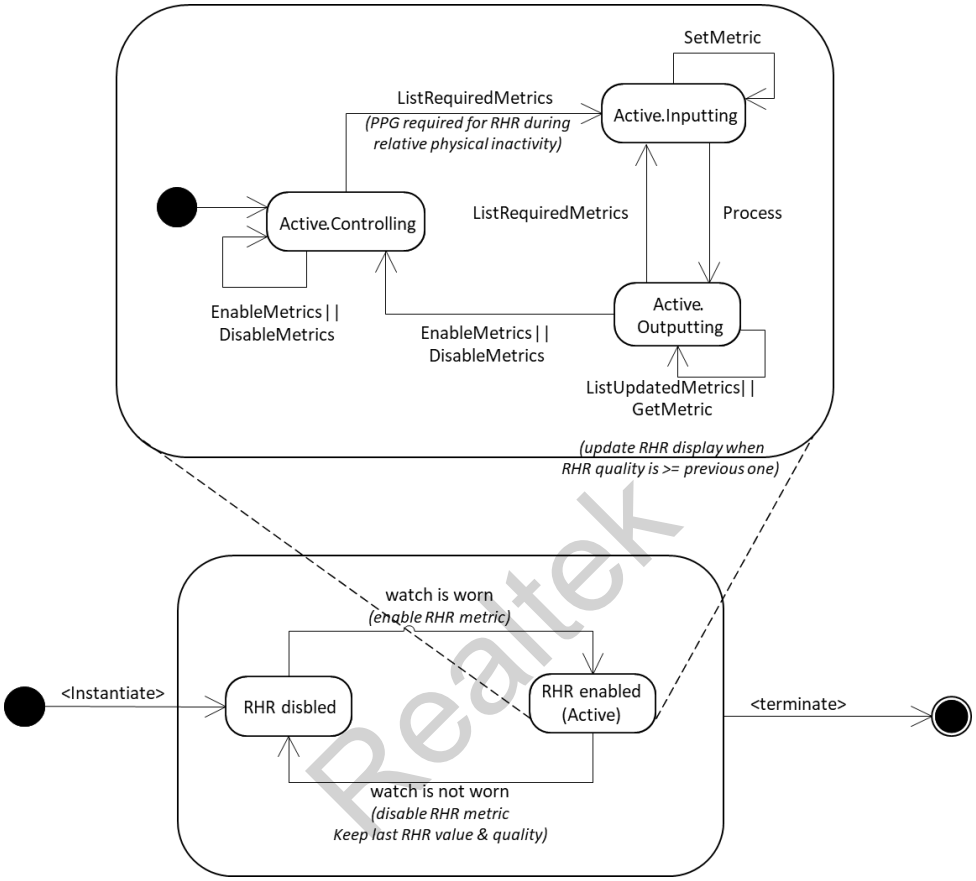

Fig 8. Flow diagram in enabling / disabling resting Heart Rate

2.10 LED Power Control Loop Recommendation

LED power control loop is a subroutine designed to achieve adaptive power control for LEDs and auto-gain control of PPG analog front end. It is a controller that tries to set optimal “Led Power” and “ADC Gain” for the PPG front-ends that have access to the power of LED and the A/D conversion gain. Please note that this is only needed if the PPG front-end does not contain automatic gain control inside the chip. It is the responsibility of the system integrators to ensure the LED power control loop working properly to provide accurate PPG data to PSP library.

Due to different skin colors of end users or “small” variations in reflectivity (due to changing perfusion), the PPG signal from the AD converter is running/drifting out of the optimal window of operation or even is clipping. An adaptive control loop is implemented to set “Led Power” and “ADC Gain” so the resulting PPG signal from AD converter is again in the optimal range.

First, the LED power and gain control should be able to adapt to the skin color of the end user when the device is worn on wrist. Then, an adaption may occur after a warm up session because of change of skin perfusion. The adaption may also occur when the signal quality and/or the modulation becomes too small. All these adaptions aim at keeping a balance between the signal quality and power consumption of the system. However, the changes of LED power and gain control introduce discontinuities in the signal which should be limited to a minimum.

Ideally, the adjustment of LED power and gain control is desired only once to adapt the skin color and skin contact of end user because unwanted discontinuity in PPG signal occurs at every adjustment. But at the same time, it is desirable to adjust as quickly as possible to follow new situations in changes of skin contact or in skin perfusion. The LED/gain figure should be specified in each feeding of PPG samples to PSP library. The update rate of LED/gain figure slower than 1 second is undesirable because most metrics are updated every second, otherwise the reaction becomes too slow. In summary, the adaptions will not occur too often when watch is worn on wrist.

In practice, system integrators are suggested to use the FIFOs of Analog Front End and Accelerometer to minimize the power for serial communication. Even though the depth of FIFO may contain samples longer than 1 second, it should not make the update rate longer than 1 second. Otherwise, the LED power and gain control reacts too slowly when the end user puts the watch on wrist.

The operation principle of the LED power and gain is that we control the LED power to bring the PPG output working around a target level with around 80dB Signal-to-noise ratio (SNR) for Green (85dB for Red). An operation window is defined with the target level laying between the lower and upper boundaries of the window. Once the PPG output exceeds either the lower or upper boundary of the operation window, the LED power output will be adjusted to bring the PPG close to the target level of PPG output.

An example of the power control loop with MAX30110 is illustrated in following pseudo code. This design could be modified to fit for different Analog Front Ends.

2.10.1 Configuration of MAX30110

The configuration of MAX30110 in the example includes:

The current range of LEDs are configured with 50mA[00] in full range.
The LED current is controlled with LED1_PA (0x11) and LED2_PA(0x12).
LED Range(0x14) is set to 0x00. (refer to spec of MAX30110 for details)

The LED power control parameter ranges from 0 to 1023.

The PPG sensor is configured with 25Hz in Single Pulse Mode[0001], 104 µs integration time[01]. The ADC Range is selectable among [00], [01] and [10]. LED settling time is set to 20 µs[11] and no sample averaging[000].

PPG Configuration 1(0x0E) is set to 0bxx00 0101
PPG Configuration 2(0x0F) is set to 0b0001 1000

The PPG gain control is switching among gain = 1 to 3

The register for FIFO is configured as
FIFO Configuration(0x08) = 0x07 with interrupt at 25 sample in FIFO
FIFO Data Control 1(0x09) = 0xCD with LED1&LED2 on Data Item1; Ambient on Data Item2
FIFO Data Control 2(0x0A) = 0x00 with no data on Data Item 3 & 4

Only a quarter of the ADC range is used, discarding upper 2 bits. It is achieved with the optimal window of operation setting in a way that

```
ADC_MAX = 524287              /* (2^19) – 1; max value of 19-bit unsigned */
thrHigh = ADC_MAX * ¼ * 90%;
thrLow  = ADC_MAX * ¼ * 45%;
```

Apparently, the resulting PPG value from MAX30110 can already be presented 17 bit (unsigned). Then, unsigned 16 bit PPG signal for PSP library is obtained by discarding lowest 1 bits. The Ambient signal is adjusted to 1x gain (gain = 0).

Here below shows the range setting for different ADC gain.

Relative ADC gain <G>	ADC gain factor	FULL SCALE (nA)	¼ RANGE USED (nA)	LSB resolution in 16-bit PPG (pA)
0	gain factor 1x	48000	12000	366.2
1	gain factor 2x	24000	6000	183.1
2	gain factor 4x	12000	3000	91.6
3	gain factor 8x	6000	1500	45.8

The example of the control loop has a preference of 4x gain (gain = 2) in calculation of new LED power and gain settings. The optimal window of operation from the AD converter of PPG signal is shown in Fig 9. The PPG signal running/drifting out of the optimal window of operation results re-calculation of the LED power and AD converter gain of PPG signal.

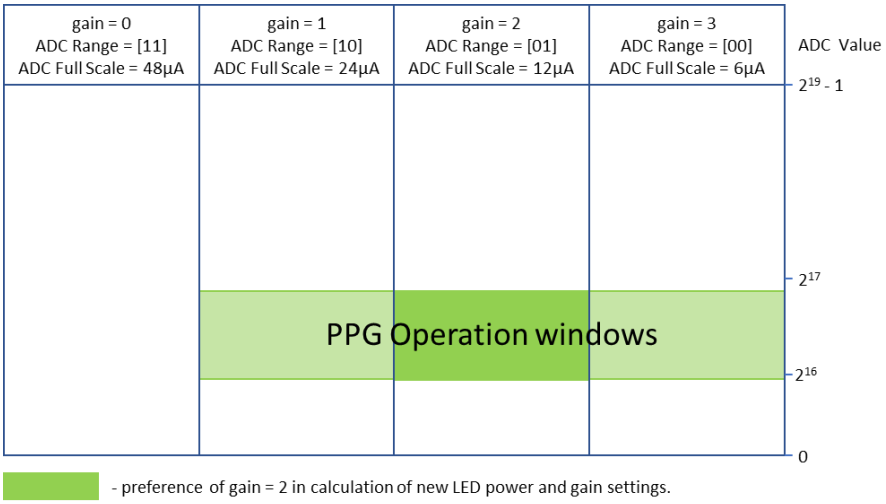

Fig 9. Optimal Window of Operation for PPG signal

2.10.2 LED Power Control Loop Example

```
#define ADC_MAX     (524287) /* (2^19) - 1 */

#define PPG_HIGH    (90) /* 90% of used scale: */
#define PPG_LOW     (45) /* 45% of used scale: */
#define LED_MIN     (10) /* 10% */
#define LED_MAX     (60) /* 60% */

/*-----------------------------------------------------------------------------*
*                                                                              *
*  Name        : ControlLoop                                                   *
*                                                                              *
*  Description : Calculate new led power + gain based upon current led power,*
*                gain and ppg values                                           *
*                                                                              *
*  Inputs      :  ppg     : ppg value from front-end (full 19 bit range)      *
*                *pwr     : led power [0..1023]                               *
*                *gain    : gain [1..3]                                       *
*                                                                              *
*  Outputs     : *pwr     : led power [0..1023]                               *
*                *gain    : gain [1..3]                                       *
*                                                                              *
*  Comments    : This function relies on defines:                              *
*                ADC_MAX    : full range (2^19)-1                             *
*                PPG_HIGH   : high threshold in %                             *
*                PPG_LOW    : low  threshold in %                             *
*                LED_MIN    : min led power in %                              *
*                LED_MAX    : max led power in %                              *
*                                                                              *
*                Only if ppg value is outside range [thrLow..thrHigh]          *
*                new led power values will be calculated to reach target       *
*                value, presuming linear relation between led power and ppg    *
*                If new led power is outside range [LED_MIN..LED_MAX]          *
*                recalculate new values taken gain factor into account, if     *
*                gain range permits                                            *
*                                                                              *
*                This function is typically called once per second             *
*                                                                              *
```

```
*                                                                                 *
*---------------------------------------------------------------------------------*/

static void ControlLoop( uint32_t ppg, uint16_t *pwr, uint8_t *gain)
{
    /* preset new values with old value */
    uint32_t pwrNew  = *pwr;
    uint8_t  gainNew = *gain;

    /* calculate thresholds for scale used: 1/4 of full so (ADC_MAX / 4) */
    uint32_t thrHigh = ((((uint32_t)PPG_HIGH) * (ADC_MAX >> 2)) / 100);
    uint32_t thrLow  = ((((uint32_t)PPG_LOW ) * (ADC_MAX >> 2)) / 100);
    uint32_t target  = (thrLow + thrHigh) / 2;   /* put target in the middle of low-high
thresholds */

    /* check if thresholds are hit */
    if ( (ppg > thrHigh) || (ppg < thrLow) || (trigLoop == true) )
    {
        /*
********************************************************************************** */
        /* based on this value, we are going to determine a new pwr setting on CURRENT gain
scale */

        /* ppg/pwrOld = target/pwrNew, so: */
        /* pwrNew = (pwrOld * target) / ppg */
        pwrNew *= target;            /* new_power already filled with old_power */
        pwrNew /= (ppg + 1);         /* + 1 to prevent division by 0 */

        /*
********************************************************************************** */
        /* check current gain and modify to (preferred) gain=2 + recalculating led pwr
accordingly */
        /* e.g we prefer to have (gain=2, pwr=20..60) iso (gain=3, pwr=10..30) */
        if ( trigLoop == false )
        {
           if ( gainNew > 2 )
           {
             gainNew -= 1;   /* switch to lower sensitivity/gain(2) */
             pwrNew <<= 1;   /* multiply by 2 */
           }
           else if ( gainNew < 2 )
           {
             gainNew += 1;   /* switch to higher sensitivity/gain(2) */
             pwrNew >>= 1;   /* divide by 2 */
           }

           /*
   ********************************************************************************************
   */
           /* if power is low then we have "large" PPG signal */
           /* therefore we can better go to lower sensitivity with 2x led power */
           if ( pwrNew < (((uint32_t)LED_MIN * 1023) / 100) )
           {
             gainNew -= 1;   /* switch to lower sensitivity/gain */
             pwrNew <<= 1;   /* multiply by 2 */
           }
           /* if power is high then we have "small" PPG signal */
           /* therefore we can better go to higher sensitivity with 2x less led power */
           else if ( pwrNew >= (((uint32_t)LED_MAX * 1023) / 100))
           {
             gainNew += 1;   /* switch to higher sensitivity/gain */
             pwrNew >>= 1;   /* divide by 2 */
           }
           else
           {
             //gainNew = 2;
           }
        }
        trigLoop = false;


/* ********************************************************************************** */
```

```
        /* clip the new led power to [20..1023] */
        if (pwrNew > 1023)
        {
            pwrNew = 1023;
        }
        else if (pwrNew < 20)
        {
            pwrNew = 20;
        }

        /* clip the new gain to [1..3] */
        if (gainNew > 3)
        {
            gainNew = 3;
        }
        else if (gainNew < 1)
        {
            gainNew = 1;
        }
    }

    /* Return the new gain value */
    *gain = gainNew;

    /* Return the new power value */
    *pwr = (uint16_t) pwrNew;
}
```

Here below are the mapping of LED power and gain value between the ControlLoop and hardware registers of MAX30110.

Translation of LED power [0..1023] to LED current control [0..255]

In MAX30110, LED current are controlled with LED1_PA (0x11) and LED2_PA(0x12).

```
  LEDx_PA = (uint8_t)(pwr >> 2);        /* 10bit to 8bit values in LEDx_PA;*/
```

Translation of gain [1..3] to ADC Range in PPG Configuration 1(0x0E)

ADC Range is 2-bit values with [10 01 00] corresponds to [1 2 3] in gain value

```
  regGain = (uint8_t)((gain ^ 0x03) << 6) | 0x04 | 0x01;         /* convert 0..3 to 3..0 */
  trigLoop = true;          /* sensitivity is changed, trigger loop again for fine tuning */
```

Translation from 19-bit PPG values to unsigned 16-bit values to PSP library:

```
uint32_t tmpValue;

tmpValue    = ( ppgFrontEnd     >> 1 );         /* presume ppg is 17b, shift by 1 for 16b */
ppg         = (tmpValue > 65535) ? 65535 : (uint16_t)tmpValue; /* clip against 16 bit max */

tmpValue    = ( ambientFrontEnd >> (1+gain) ); /*ambient in gain=0,correct with used gain */
                                               /* select proper offset correction */
ambient     = (tmpValue > 65535) ? 65535 : (uint16_t)tmpValue; /* clip against 16 bit max */

tmpValue    = (pwr / 10);                           /* convert from [0..1023] to [0..102] */
ledPower    = (tmpValue > 100) ? 100 : (uint8_t)tmpValue;            /* clip against 100 */

adcGain     = gain;                                                  /* no conversion */
```

2.10.3 Explanation of the LED Power Control Loop Example

The procedure to define the target level of PPG output and the operation window is illustrated below. The maximum range, ADC_MAX, of the ADC output of PPG signal is defined by the Analog Front End chosen, for example the range is 2^19 (for the 19 bit ADC in our reference). With the chosen LED and Photodiode components, the quality of the PPG signal should be analyzed under multiple subjects with a broad variation of required skin tones in different situations, like in cold or hot weather.

A PPG output level is found to ensure the SNR is still above 80dB while lowering the LED power level. This PPG output level is the target level required. Lowering the target level also means that less LED power is needed and therefore power is saved. The boundaries of operation window around the target level are defined by a minimum level and maximum level. As an example in our example design, the minimum level for the PPG window (PPG_LOW) is 45% and maximum level (PPG_HIGH) is 90%. We have chosen to use a quarter of the ADC Converter output, so the thresholds of the PPG window are

the maximum threshold: thrHigh = ADC_MAX / 4 * (PPG_HIGH / 100);
the minimum threshold: thrLow = ADC_MAX / 4 * (PPG_LOW / 100);
the target level: target = (thrLow + thrHigh) / 2.

When PPG signal is within the operation window, there is no change required. If PPG signal goes above maximum level or below minimum level, the ADC gain and LED power will be changed to bring the PPG signal go to the target level again. Assuming the system is linear, the new LED power and gain values are calculated using the old ones. These new values should bring the PPG output close to target level. If the system, including the LED, photodiode and AFE, is not linear (e.g. 5V booster is not used for LED power etc), a compensation should be included in the calculation.

In the Example of Power Control Loop in section 2.10.2, it is assumed the system is non-linear. trigLoop parameter is introduced. In case the system is linear, this parameter is not needed. The condition checking of (trigLoop == true) should be removed and trigLoop is always set to false.

The LED power is controlled with a parameter(*pwr) in a range of 0 to 1023. We do the calculation of new LED power with assuming a linear relation between LED power and PPG signal at unit gain, so "new LED power" = "old LED power" * (target signal / ppg signal[1] + 1).

The new LED power is calculated based on current gain setting. Assuming the system is linear, doubling the ADC gain with halving the LED power will get the same PPG signal output. With the preference of gain=2, the new LED power is calculated by setting the gain = 2 first, then followed by check if the new LED power is well located in the operation window.

The operation window of LED power is defined with LED_MIN and LED_MAX. The LED power will be doubled with decreasing the PPG gain if new power is below LED_MIN. The LED power will be halved with increasing the PPG gain if new power is beyond LED_MAX. The limits of minimum and maximum LED power levels are set to 10% and 60% respectively in the example.

Finally, the PPG signal to the library should operate between minimum level (PPG_LOW) = 45% and maximum level (PPG_HIGH) = 90% when the PPG sensor is on wrist. The LED power will be driven to 100% if there is no PPG signal (ambient only) in case that the PPG sensor is off wrist. Keeping PPG signal operating outside the window between PPG_LOW and PPG_HIGH in on-wrist situation means the ControlLoop is not in proper functioning.

2.11 PPG Signal for SpO2 measurement

The SpO2 measurement requires to calculate the ratio between the absolute level of Red and Infrared PPG signals. So in the processing of PPG signals, both the AC and DC components should be intact. If the signal is scaled in the preprocessing, the same scaling factor should be applied to both AC and DC components. If subtraction is performed in the preprocessing, the subtracted value should be known to be able to reconstruct the PPG signals for SpO2 measurement.

2.12 Auto Scaling of PPG signal

Some PPG sensors which contain automatic gain control, like Pixart sensor, will deliver 32 bit PPG signal and the dynamic range is quite high. A sophisticated auto scaling function can be used to scale the signal from 32 bit to 16 bit unsigned signal as well as maintaining the same gain factor for Red and Infrared channel. The following shows the logic of auto scaling. For the example code, please contact Philips ITS Customer Support.

[1] The target signal and ppg signal is the photodiode signal at current gain level (or hardware ADC range).

```
/*-------------------------------------------------------------------------*
*                                                                         *
*  Name        : Normalize_ppg                                            *
*                                                                         *
*  Description : Normalize Pixart normalized mode PPG signal to 0 - 65535 *
*                                                                         *
*  Inputs      : *in      : input samples float                           *
*                inLen    : number of  input samples                      *
*                *out     : output samples in 0 - 65535                   *
*                *ppgmax  : output max ppg raw data in last calibration   *
*                           period (N seconds)                            *
*                ppgsf    : PPG Scaling factor (SF1)                      *
*                adcgain  : ADC Gain                                      *
*                caltimer : Calibration timer                             *
*                                                                         *
*  Outputs     : *out     : output samples                                *
*                ppgsf    : PPG Scaling factor                            *
*                adcgain  : ADC Gain                                      *
*                caltimer : Calibration timer                             *
*                                                                         *
*  Comments    : Dynamic scaling of PPG signal to meet library requirements *
*                1. Assume the PPG signal is captured after skin detected.*
*                2. Use default scaling factor SF0                        *
*                3. Calculate scaling factor SF1 based on first N seconds.*
*                4. After N seconds, use the SF1 to scale the PPG signal. *
*                5. Set ADC gain = 2. Evaluate the PPG signal every second.*
*                                                                         *
*                If PPG signal > PPGMaxT (e.g. 90%) of maximum PPG range, *
*                divide the PPG by 2, set gain to current - 1             *
*                                                                         *
*                If PPG signal < PPGMinT (e.g. 20%) of maximum PPG range, *
*                multiply the PPG by 2, set gain to current + 1           *
*                                                                         *
*                If PPG signal reaches PPGMaxT and Gain 1 (cannot decrease*
*                gain further) Or PPG signal reaches below PPGMinT and gain 3*
*                (cannot increase gain further) ==> recalculate SF1.      *
*                                                                         *
*                                                                         *
*                 For SpO2 calculation, R and IR channel must use the same *
*                Scaling factor SF1. In addition, Pixart should operate in*
*                normalized mode.                                         *
*                                                                         *
*-------------------------------------------------------------------------*/
```

A Appendix : Typical Integration Issues

Intended to be blank.

Legal information

Disclaimers

This document may be subject to change without notice. Philips Electronics Nederland B.V. does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

This document is provided “as is” with no warranties, expressed or implied, including but not limited to any implied warranty of merchantability, fitness for a particular purpose, or freedom from infringement. Philips Electronics Nederland B.V. or its Affiliates may have patents or pending patent applications, or other intellectual property rights that relate to the described subject matter. Philips and its logo are trademarks of Koninklijke Philips N.V. All other names are the trademark or registered trademarks of their respective holders. The furnishing of this document does not provide any license, expressed or implied, by estoppels or otherwise, to any such patents, trademarks or copyrights, or other intellectual property rights. Philips Electronics Nederland B.V. assumes no responsibility for errors or omissions in this document; nor does Philips Electronics Nederland B.V. make any commitment to update the information contained herein. This document is subject to change without notice.

Patents

Notice is herewith given that the subject device uses one or more patents and that each of these patents may have corresponding patents in other jurisdictions.

Philips has been awarded a number of patents on the technology used in the Philips Sensing Platform offering in Europe, Asia and the United States.

Trademarks

Notice: All referenced brands, service names and trademarks are property of their respective owners.