PAH8002EP: PPG Data Acquisition for Heart Rate Calculation

Application Note AN32

Related Part Ordering Information

Part Number	Type
PAH8002EP-IP	Low Power Optical Heart Rate Detection Sensor, 22-Pin LGA Package

For any additional inquiries, please contact us at http://www.pixart.com/contact.asp

Table of Contents

List of Figures

List of Tables

1.0 Introduction

1.1 System Design

As a Heart Rate Detection (HRD) sensor, the PAH8002EP has integrated DSP to attain those processed PPG (Photoplethysmogram) data for use in deducing heart-beat information. The PAH8002EP is always implemented as a slave device where a controller or processor is required to be the host functioning as a complete system design. The PPG data is acquired by the host via serial interface. Being processed with the sophisticated algorithm library either in APP or Firmware level as shown in the Figure 1. System Block Diagram, the heart rate data and waveform could be determined for display on Heart Rate Monitor (HRM) device. As a solution provider, PixArt is providing set of algorithm libraries (Refer to Section 4.1Pixart Algorithm API) as reference in calculating the heart rate.

1.2 System Block Diagram

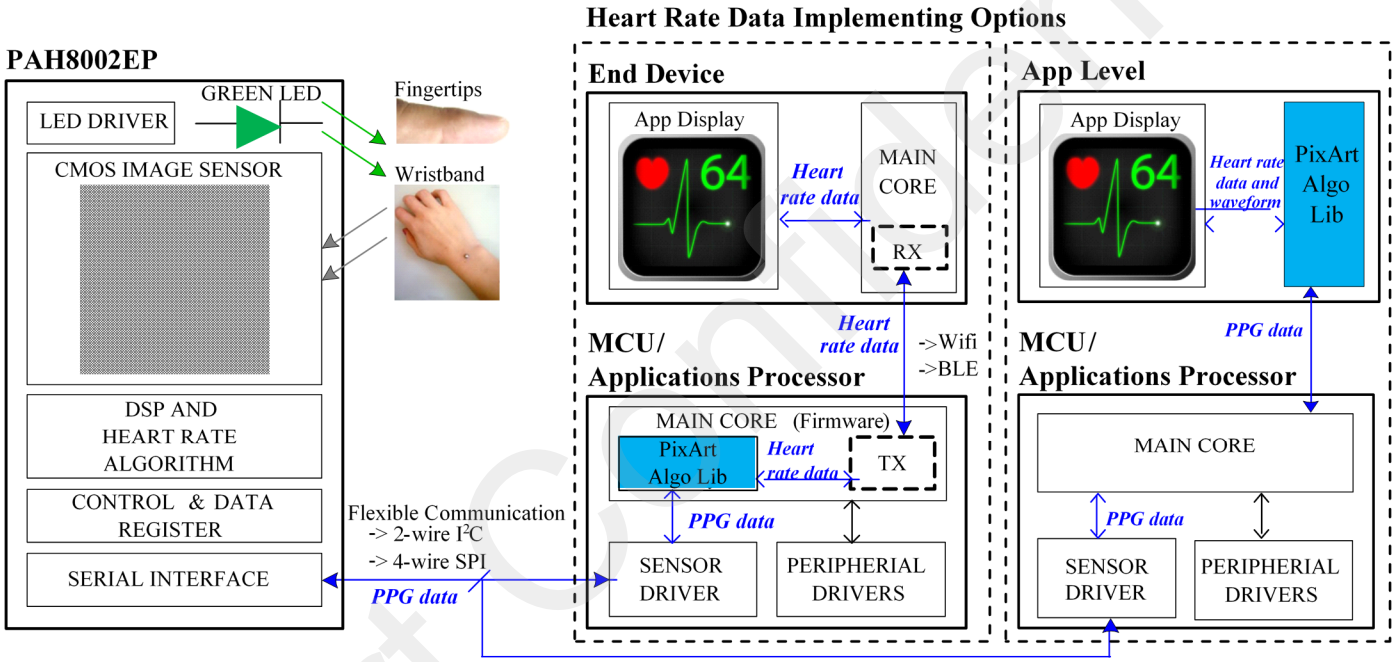

Figure 1. System Block Diagram

2.0 Operation Flow

2.1 Touch Mode

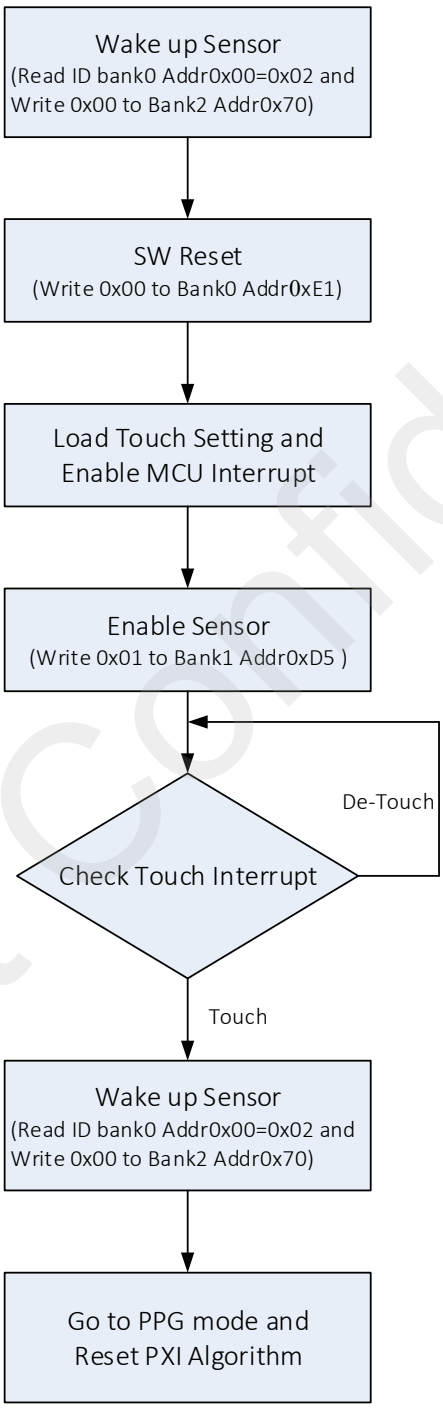

Figure 2. Touch Mode Flow Chart

2.2 PPG Mode

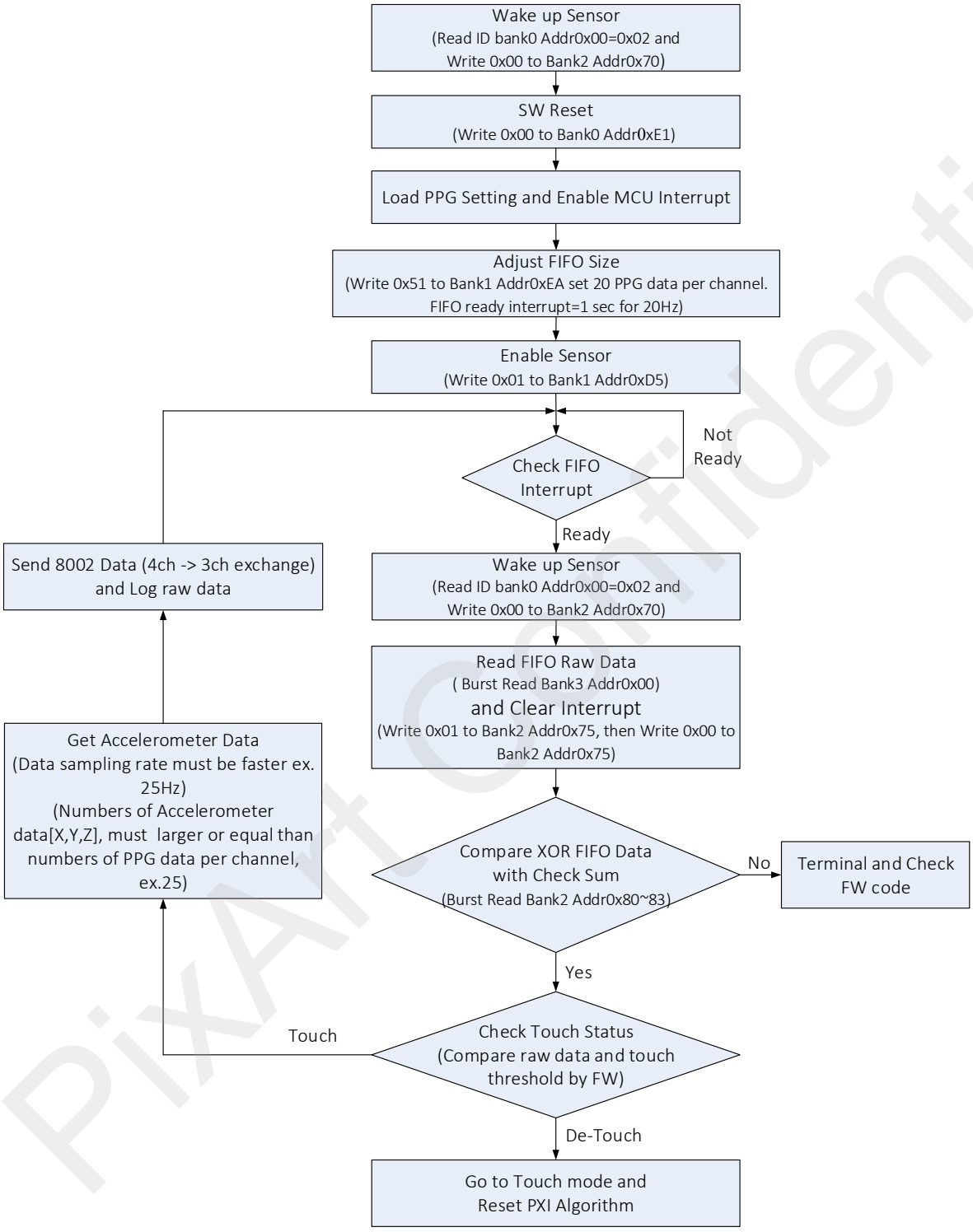

Figure 3. PPG Mode Flow Chart

3.0 Heart Rate FIFO Control for PPG Data and Accelerometer Data Asynchronous

1. The first PPG and Accelerometer data of FIFO must be synchronous.

2. The second data point to the last data point of Accelerometer data in FIFO must be interpolated to meet PPG time period.

Example:

Numbers of PPG data is 20 for 20Hz report rate；Numbers of Accelerometer data is 25 for 25Hz report rate

In one second interrupt interval, Accelerometer data of FIFO must be interpolated to 20 data points matching with PPG data.

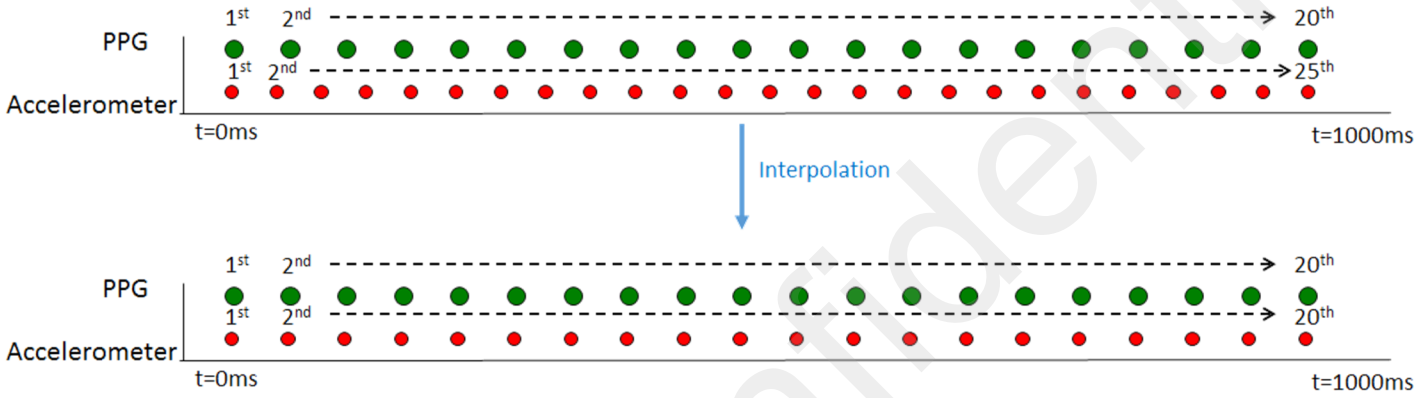

Figure 4. Data Interpolation of Accelerometer to match with PPG

4.0 Software References

4.1 Pixart Algorithm API

Algorithm Library Version must update to V212 or later.

Algorithm Lib. need 12kB RAM Size for dynamic memory allocation (malloc). Please set heap Size more than 0x3000 .

Table 1. Pixart Algorithm API Function Calls

Function Calls	Description
PXIALG_API uint32_t pah8002_version();	Call this function to determine the version of PXI algorithm.
PXIALG_API uint32_t pah8002_query_open_size(void);	Call this function to query the size (in bytes) of function pah8002_open required.
PXIALG_API uint8_t pah8002_open(void *pBuffer);	Call this function to open PXI algorithm. It requires that the client allocates enough memory (at least the return size of pah8002_query_open_size).
PXIALG_API uint8_t pah8002_close();	Call this function to close/reset PXI algorithm.
PXIALG_API uint8_t pah8002_set_param(pah8002_param_idx_t idx, float p1);	Call this function to set 1 parameter. Ex: PAH8002_PARAM_IDX_GSENSOR_MODE, The MEMS Scale of Motion Sensor. 0: +-2G, 1: +-4G, 2: +-8G, and 3: +-16G.
PXIALG_API uint8_t pah8002_get_param(pah8002_param_idx_t idx, float *p1);	Call this function to get 1 parameter.
PXIALG_API uint8_t pah8002_entrance(pah8002_data_t *data8002);	Call this function to send 8002 data to PXI algorithm.
PXIALG_API uint8_t pah8002_get_signal_grade(int16_t *grade);	Call this function to get PPG signal grade. High signal grade is meant high signal quality.
PXIALG_API uint8_t pah8002_get_hr(float *hr);	Call this function to get heart rate value.

API Return Value (pah8002_entrance)
enum
{
MSG_SUCCESS = 0,
MSG_ALG_NOT_OPEN,
MSG_ALG_REOPEN,
MSG_MEMS_LEN_TOO_SHORT,
MSG_NO_TOUCH,
MSG_PPG_LEN_TOO_SHORT,
MSG_FRAME_LOSS,
MSG_HR_READY = 0x30,
MSG_SIGNAL_POOR = 0x40,
MSG_INVALID_ARGUMENT = 16,
};
	0: Normal
	1: Without call pah8002_open(); function
	2: Call pah8002_open(); function more than 1 time
	3: Numbers of Accelerometer data(X,Y,Z) is shorter than Numbers of PPG data per channel
	4: De-touch
	5: Numbers of PPG data per channel is too short, ex. < 5 at 20Hz will return this flag
	6: Frame is not continuous
	0x30: Heart rate value ready flag.
	MCU can get heart rate after getting this flag
	0x40: Input signal data is poor.
	16: Invalid argument to function input.

4.2 PAH8002EP Data Format Description and Record Data Example

```
typedef struct pah8002_data {

      uint8_t      frame_count;        //Frame Count
      uint32_t     time;               //FIFO Data Ready Interval, unit ms
      uint8_t      touch_flag;         //Touch Status, 1 for Touch and 0 for De-Touch
      uint32_t     nf_ppg_channel;     //Using channel numbers, ex.3
      uint32_t     nf_ppg_per_channel; //Numbers of PPG data per channel, ex.20
      int32_t      *ppg_data;          //Pointer to FIFO Raw Data
      uint32_t     nf_mems;            //Numbers of Accelerometer data(X,Y,Z), must larger or equal
                                       //than numbers of PPG data per channel, ex.25
      int16_t      *mems_data;         //Pointer to Accelerometer data
} pah8002_data_t;
```

Record Data Example:

```
PPG CH#, 3 // Using channel numbers
Frame Count, 0 // frame count for data loss check
Time, 1020 //FIFO data ready interval, unit ms：ex. 20 data for 20Hz setting, time is about 1000ms
PPG, 1, 20, 0, 2493254, 1518686, 0, 2492862, 1517980, 0, 2492126, 1517858, 0, 2492962, 1518008, 0,
2491192, 1518126, 0, 2489914, 1517328, 0, 2490798, 1517536, 0, 2490260, 1518096, 0, 2491386,
1518952, 0, 2492320, 1519750, 0, 2488392, 1518480, 0, 2482542, 1516676, 0, 2479342, 1514576, 0,
2479538, 1515138, 0, 2480324, 1515640, 0, 2479144, 1515374, 0, 2477624, 1514310, 0, 2476150,
1513366, 0, 2476148, 1513484, 0, 2478262, 1514254,

//ppg data, 1 is touch flag, 20 is numbers of ppg data per channel, then ch0(IR) data, ch1(G)
data, ch2(G) data sequence, so total is 20 data.
MEMS, 25, 1792, 7872, 896, 1792, 7936, 896, 1856, 8000, 960, 1664, 7936, 832, 1728, 7872, 768,
1792, 7936, 896, 1728, 7872, 1024, 1792, 7936, 832, 1792, 7936, 896, 1792, 7872, 1024, 1856, 8000,
832, 1728, 7936, 832, 1856, 7936, 960, 1792, 7936, 896, 1920, 8000, 960, 1856, 7872, 832, 1920,
7936, 896, 1920, 7936, 832, 1792, 8000, 832, 1792, 7936, 832, 1856, 7872, 960, 1920, 7808, 832,
1856, 7936, 832, 1856, 8000, 896, 1920, 7872, 896,
//mems data, 25 is numbers of accelerometer data per channel, then X data, Y data, Z data
sequence, so total is 25 data. If no use, please key 0 in all value.
```

4.3 Reference Code of MCU for Use with PixArt's HRD Algo Library

4.3.1 pah8002.c

```
#include "pah8002.h"
#include "pah8002_comm.h"
#include "pah8002_api_c.h"

#include "board.h"

#include "dd_vendor_1.h"
#include "uart.h"
#include "main.h"
#include "accelerometer.h"
#include <stdint.h>
#include <string.h>
#include <stdlib.h>

#define TOTAL_CHANNELS          4          //Using channel numbers
#define HEART_RATE_MODE_SAMPLES_PER_CH_READ (20)        //Numbers of PPG data per channel
#define HEART_RATE_MODE_SAMPLES_PER_READ (TOTAL_CHANNELS* HEART_RATE_MODE_SAMPLES_PER_CH_READ)
#define TOTAL_CHANNELS_FOR_ALG      3
#define MEMS_ZERO     //Default Accelerometer data are all zero


#define PPG_MODE_ONLY

enum{
        SUSPEND_MODE = 0,
        TOUCH_MODE,
        NORMAL_MODE,
        NORMAL_LONG_ET_MODE,
        STRESS_MODE,
        NONE,
};


static uint8_t _mode = NONE ;
static uint8_t pah8002_ppg_data[HEART_RATE_MODE_SAMPLES_PER_READ * 4] ;
static uint8_t _touch_flag = 0 ;
static volatile uint8_t _pah8002_interrupt = 0 ;
static pah8002_data_t _pah8002_data;
static uint32_t _timestamp = 0 ;

#ifdef MEMS_ZERO
static int16_t _mems_data[HEART_RATE_MODE_SAMPLES_PER_READ * 3] ;
#endif
```

```
static uint8_t _ir_dac = 0 ;
static uint8_t _ir_expo = 0 ;
static uint8_t _chip_id = 0 ;

static void *_pah8002_alg_buffer = NULL;

static bool pah8002_sw_reset(void);
static bool pah8002_start(void);
static int pah8002_wakeup(void);
static int pah8002_check(void);


//++++++++++++++++++++++PAH8002 functions++++++++++++++++++++++++++++++++++++++++++++++++++++++++
static bool pah8002_sw_reset()
{
		uint8_t data ;
		debug_printf(">>> pah8002_sw_reset \r\n");
		pah8002_wakeup();

		if(0 != pah8002_write_reg(0x7f, 0x00))
		{
			goto RTN;
		}
		if(0 != pah8002_read_reg(0, &data))
		{
			goto RTN;
		}
		debug_printf("ID = %d\r\n", data);
		if(data != 0x02)
		{
			goto RTN;
		}
		if(0 != pah8002_write_reg(0xe1, 0))	//write 0 to trigger Software Reset
		{
			goto RTN;
		}

		//delay 5ms
		delay_ms(5);
		debug_printf("<<< pah8002_sw_reset \r\n");
		return true;
RTN:
		return false;
}
```

```
static bool pah8002_start()
{
		uint8_t data = 0;
		int samples_per_read =  HEART_RATE_MODE_SAMPLES_PER_READ ;
		debug_printf(">>> pah8002_start \r\n");

		pah8002_wakeup();

		if(0 != pah8002_write_reg(0x7f, 0x01))
		{
			goto RTN;
		}
		else if(0 != pah8002_write_reg(0xea, (samples_per_read+1)))
		{
			goto RTN;
		}
		else if(0 != pah8002_write_reg(0xd5, 1)) //TG enable. REQTIMER_ENABLE
		{
			goto RTN;
		}
		else if(0 != pah8002_read_reg(0xd5, &data))	//TG enable. REQTIMER_ENABLE
		{
			goto RTN;
		}
		pah8002_check();
		debug_printf("<<< pah8002_start %d\r\n",data);

		return true;

RTN:
		return false;
}
static bool pah8002_touch_mode_init()
{
	int i =0 ;

	debug_printf(">>> pah8002_touch_mode_init \r\n");
	pah8002_wakeup();
	for(i = 0; i < INIT_TOUCH_REG_ARRAY_SIZE;i++)
	{
	  if ( pah8002_write_reg(  init_touch_register_array[i][0],
			init_touch_register_array[i][1]) != 0 )
	  {
	    goto RTN;
	  }
	}
```

```
        debug_printf("<<< pah8002_touch_mode_init \r\n");
        return true;
RTN:
        return false;
}


static bool pah8002_normal_mode_init()
{
        int i =0 ;
        debug_printf(">>> pah8002_normal_mode_init \r\n");
        pah8002_wakeup();
        for(i = 0; i < INIT_PPG_REG_ARRAY_SIZE;i++)
        {
           if ( pah8002_write_reg( init_ppg_register_array[i][0],
                    init_ppg_register_array[i][1]) != 0 )
           {
             goto RTN;
           }
        }

        debug_printf("<<< pah8002_normal_mode_init \r\n");

        return true;
RTN:
        return false;
}

static bool pah8002_stress_mode_init()
{
        int i =0 ;

        debug_printf(">>> pah8002_stress_mode_init \r\n");
        pah8002_wakeup();
        for(i = 0; i < INIT_STRESS_REG_ARRAY_SIZE;i++)
        {
           if ( pah8002_write_reg(  init_stress_register_array[i][0],
                    init_stress_register_array[i][1]) != 0 )
           {
             goto RTN;
           }
         }

        debug_printf("<<< pah8002_stress_mode_init \r\n");

        return true;
```

```
RTN:
        return false;
}

static uint8_t pah8002_get_touch_flag_ppg_mode()
{
 static uint8_t touch_sts_output = 1 ;
 int32_t *s = (int32_t *)pah8002_ppg_data ;
 int32_t ch0 ;
 int32_t ch1 ;
 int64_t ir_rawdata;
 int i;
 static int touch_cnt = 0, no_touch_cnt = 0 ;

 #define TouchDetection_Upper_TH (600)
 #define TouchDetection_Lower_TH (512)

 #define TouchDetection_Count_TH (3)                    //(3+1)*50ms = 200ms
 #define NoTouchDetection_Count_TH (3)                  //(3+1)*50ms = 200ms

 for(i=0; i<HEART_RATE_MODE_SAMPLES_PER_READ; i+=TOTAL_CHANNELS)
 {
        ch0 = *s;
        ch1 = *(s+1);
        ir_rawdata = ch0 - ch1 ;
        ir_rawdata = (ir_rawdata * _ir_dac * _ir_expo)>>20 ;

        if( ir_rawdata > TouchDetection_Upper_TH)
        {
                touch_cnt++;
                no_touch_cnt = 0;
        }
        else if( ir_rawdata < TouchDetection_Lower_TH)
        {
                no_touch_cnt++;
                touch_cnt = 0 ;
        }
        else
        {
                touch_cnt = 0 ;
                no_touch_cnt = 0;
        }

        s+=TOTAL_CHANNELS;
 }
```

```
 if(touch_cnt > TouchDetection_Count_TH)
 {
        touch_sts_output = 1;
 }
 else if( no_touch_cnt > NoTouchDetection_Count_TH)
 {
        touch_sts_output = 0;
 }


 debug_printf("<<< pah8002_get_touch_flag_ppg_mode %d, %d\n",touch_cnt, no_touch_cnt);
 debug_printf("<<< pah8002_get_touch_flag_ppg_mode %d\n", touch_sts_output);

 return touch_sts_output;
}

static bool pah8002_enter_normal_mode()
{
        debug_printf(">>> pah8002_enter_normal_mode\r\n");
        if(_mode == NORMAL_MODE) return true;

        //1. software reset
        if( !pah8002_sw_reset())
                goto RTN;


        //2. load registers for normal mode
        if( !pah8002_normal_mode_init())
                goto RTN;

        pah8002_write_reg(0x7f, 0x00); //Bank0
        pah8002_read_reg(0x0D, &_ir_expo);    // IR Exposure Time
        pah8002_write_reg(0x7f, 0x01); //Bank1
        pah8002_read_reg(0xBA, &_ir_dac);      //IR Led DAC

        //3. enable sensor
        if( !pah8002_start())
                goto RTN;
        _mode = NORMAL_MODE;
        debug_printf("<<< pah8002_enter_normal_mode ir_dac %x, ir_expo %x\r\n", _ir_dac, _ir_expo);
        return true;
RTN:
        return false ;
}
```

```
static bool pah8002_enter_stress_mode()
{
	debug_printf(">>> pah8002_enter_stress_mode\r\n");
	if(_mode == STRESS_MODE) return true;

	//1. software reset
	if( !pah8002_sw_reset())
		goto RTN;

	//2. load registers for normal mode
	if( !pah8002_stress_mode_init())
		goto RTN;

	pah8002_write_reg(0x7f, 0x00); //Bank0
	pah8002_read_reg(0x0D, &_ir_expo);    // IR Exposure Time
	pah8002_write_reg(0x7f, 0x01); //Bank1
	pah8002_read_reg(0xBA, &_ir_dac);      //IR Led DAC

	//3. enable sensor
	if( !pah8002_start())
		goto RTN;

	_mode = STRESS_MODE;
	debug_printf("<<< pah8002_enter_stress_mode \r\n");
	return true;
RTN:
	return false ;
}

static bool pah8002_enter_touch_mode()
{
	debug_printf(">>> pah8002_enter_touch_mode\r\n");
	if(_mode == TOUCH_MODE) return true;

	//1. software reset
	if( !pah8002_sw_reset() )
		goto RTN;

	//2. load registers for touch mode
	if( !pah8002_touch_mode_init())
		goto RTN;

	//3. enable sensor
	if( !pah8002_start())
		goto RTN;
```

```
        _mode = TOUCH_MODE;
        debug_printf("<<< pah8002_enter_touch_mode\r\n");
        return true;

RTN:
        return false ;
}

static bool pah8002_get_touch_flag( uint8_t *touch_flag)
{
        debug_printf(">>> pah8002_touch_status \r\n");
        pah8002_wakeup();
        if(0 != pah8002_write_reg(0x7f, 0x02))
        {
                goto RTN;
        }
        else if(0 != pah8002_read_reg(0x45, touch_flag))          //
        {
                goto RTN;
        }

        debug_printf("<<< pah8002_touch_status %d\r\n", *touch_flag);

        return true;

RTN:
        return false;
}

static int pah8002_wakeup()
{
        int retry = 0 ;
        int success = 0 ;
        uint8_t data = 0 ;
        pah8002_read_reg(0, &data);
        pah8002_read_reg(0, &data);

        do
        {
                pah8002_write_reg(0x7f, 0x00);
                pah8002_read_reg(0, &data);
                if(data == 0x02) success++;
                else success = 0 ;

                if(success >=2) break;
```

```
		retry ++;

	}while(retry < 20);

	if(_chip_id == 0)
	{
	 pah8002_read_reg(0x02, &data);

		_chip_id = data & 0xF0 ;
		if(_chip_id != 0xD0)
	 {
			debug_printf("Not support anymore\r\n");
			while(1){};
		}
	}

	pah8002_write_reg(0x7f, 0x02);
	pah8002_write_reg(0x70, 0x00);

	debug_printf("pah8002_wakeup retry %d \r\n", retry);

	return retry;
}

static int pah8002_check()
{
	int retry = 0 ;
	int success = 0 ;
	uint8_t data = 0 ;
	uint8_t b1_0xd5 = 0 ;
	uint8_t b1_0xe6 = 0 ;
	pah8002_read_reg(0, &data);
	pah8002_read_reg(0, &data);

	do
	{
		pah8002_write_reg(0x7f, 0x00);
		pah8002_read_reg(0, &data);
		if(data == 0x02) success++;
		else success = 0 ;

		if(success >=2) break;
		retry ++;
	}while(retry < 20);

	pah8002_write_reg(0x7f, 0x01);
```

```
        pah8002_read_reg(0xd5, &b1_0xd5);
        pah8002_read_reg(0xe6, &b1_0xe6);
        debug_printf("pah8002_check retry %d \r\n", retry);
        if(b1_0xd5 != 1)
                debug_printf("pah8002_check error Bank1 0xD5 0x%x \r\n", b1_0xd5);
        if(b1_0xe6 != 0xC8)
                debug_printf("pah8002_check error Bank1 0xE6 0x%x \r\n", b1_0xe6);
        return retry;
}

static bool pah8002_enter_suspend_mode()
{
        int i = 0 ;

        debug_printf("pah8002_enter_suspend_mode");
        pah8002_sw_reset();

        for(i = 0; i < SUSPEND_REG_ARRAY_SIZE;i++)
        {
           if ( pah8002_write_reg(suspend_register_array[i][0],
                    suspend_register_array[i][1]) != 0 )
           {
             return false;
           }
 }
        _mode = SUSPEND_MODE;
 pah8002_check();
 return true;
}

static bool _pah8002_task()
{
        uint8_t cks[4] ;
        uint8_t int_req = 0;

        debug_printf(">>> pah8002_task\n");
        pah8002_wakeup();
        if(0 != pah8002_write_reg(0x7f, 0x02))
        {}
        else if(0 != pah8002_read_reg(0x73, &int_req))
        {}
        else
        {
                if( (int_req & 0x04) != 0)
                {
                        //overflow
```

```
		while(1);
}

if( (int_req & 0x02) != 0)
{
		//touch
		debug_printf("touch interrupt\n");
}

if( (int_req & 0x08) != 0)
{
		//overflow
		while(1);
}

if( (int_req & 0x01) != 0)
{
int samples_per_read = HEART_RATE_MODE_SAMPLES_PER_READ ;
		debug_printf("FIFO interrupt\n");
		//pah8002_get_touch_flag(&state->pah8002_touch_flag);
		if(0 != pah8002_write_reg(0x7f, 0x03))
		{}
		else if(0 != pah8002_burst_read_reg(0, pah8002_ppg_data, samples_per_read*4))
		{}
		else if(0 != pah8002_write_reg(0x7f, 0x02))
		{}
		else if(0 != pah8002_burst_read_reg(0x80, cks, 4))
		{}
		else if(0 != pah8002_write_reg(0x75, 0x01))	//read fifo first, then clear SRAM FIFO interrupt
		{}
		else if(0 != pah8002_write_reg(0x75, 0x00))
		{}
		else
		{
			uint32_t *s = (uint32_t *)pah8002_ppg_data ;
			uint32_t cks_cal = *s;
			uint32_t cks_rx = *((uint32_t *)cks) ;
			uint32_t i ;

			//checksum compare
			for(i=1; i<samples_per_read; i++)
			{
				cks_cal = cks_cal ^ (*(s+i)) ;
			}

			if(cks_cal != cks_rx)
```

```
                                {
                                        debug_printf("checksum error\r\n");
                                }
                                else
                                {
                                        debug_printf("checksum OK %d\r\n", cks_cal);
                                }
                                _touch_flag = pah8002_get_touch_flag_ppg_mode();
                        }
                }
                else
                {
                        debug_printf("not fifo interrupt%d\r\n", int_req);
                }
        }

        debug_printf("<<< pah8002_task\n");
        return true;
}

static bool pah8002_normal_long_et_mode_init()
{
        int i =0 ;
        debug_printf(">>> pah8002_normal_long_et_mode_init \r\n");
        pah8002_wakeup();
        for(i = 0; i < INIT_PPG_LONG_REG_ARRAY_SIZE;i++)
        {
          if ( pah8002_write_reg( init_ppg_long_register_array[i][0],
                  init_ppg_long_register_array[i][1]) != 0 )
          {
            goto RTN;
          }
        }

        debug_printf("<<< pah8002_normal_long_et_mode_init \r\n");
        return true;
RTN:
        return false;
}
static bool pah8002_enter_normal_long_et_mode()
{
        debug_printf(">>> pah8002_enter_normal_long_et_mode\r\n");
        if(_mode == NORMAL_LONG_ET_MODE) return true;

        //1. software reset
        if( !pah8002_sw_reset())
```

```
            goto RTN;

        //2. load registers for normal mode
        if( !pah8002_normal_long_et_mode_init())
            goto RTN;

        pah8002_write_reg(0x7f, 0x00); //Bank0
        pah8002_read_reg(0x0D, &_ir_expo);    // IR Exposure Time
        pah8002_write_reg(0x7f, 0x01); //Bank1
        pah8002_read_reg(0xBA, &_ir_dac);      //IR Led DAC

        //3. enable sensor
        if( !pah8002_start())
            goto RTN;
        _mode = NORMAL_LONG_ET_MODE;
        debug_printf("<<< pah8002_enter_normal_long_et_mode ir_dac %x, ir_expo %x\r\n", _ir_dac, _ir_expo);
        return true;
RTN:
        return false ;
}

static void pah8002_dyn_switch_ppg_mode()
{
        uint8_t b2a4, b2a5 ;
        uint16_t value ;
        pah8002_wakeup();
        pah8002_write_reg(0x7F, 0x02);
        pah8002_read_reg(0xa4, &b2a4);
        pah8002_read_reg(0xa5, &b2a5);
        value = b2a5 ;
        value <<= 8 ;
        value += b2a4 ;
        if (value >  4639)
        {
            pah8002_enter_normal_long_et_mode();
        }
}
//-----------------------------------PAH8002 functions-----------------------------------------------


bool pah8002_init(void)
{
        uint8_t ret = 0;
        uint32_t open_size = 0;

        //Algorithm initialization
```

```
	_pah8002_data.frame_count = 0 ;
	_pah8002_data.nf_ppg_channel = TOTAL_CHANNELS_FOR_ALG;
	_pah8002_data.nf_ppg_per_channel = HEART_RATE_MODE_SAMPLES_PER_CH_READ;
	_pah8002_data.ppg_data = (int32_t *)pah8002_ppg_data;
#ifdef MEMS_ZERO
	memset(_mems_data, 0, sizeof(_mems_data));
	_pah8002_data.nf_mems = HEART_RATE_MODE_SAMPLES_PER_CH_READ;
	_pah8002_data.mems_data = _mems_data;
#endif
	open_size = pah8002_query_open_size();
	_pah8002_alg_buffer = malloc(open_size);
	ret = pah8002_open(_pah8002_alg_buffer);
	if (ret != MSG_SUCCESS)
		return false;

	// Set 0: +/-2G, 1: +/-4G, 2: +/-8G, 3: +/-16G
	if (MSG_SUCCESS != pah8002_set_param(PAH8002_PARAM_IDX_GSENSOR_MODE, 1))
		return false;

	log_printf("PPG CH#, %d\n", TOTAL_CHANNELS_FOR_ALG);
	delay_ms(300);
#ifdef PPG_MODE_ONLY
	return pah8002_enter_normal_mode();
#else
	return pah8002_enter_touch_mode();
#endif
}

void pah8002_deinit(void)
{
	pah8002_enter_suspend_mode();

	pah8002_close();

	if (_pah8002_alg_buffer)
	{
		free(_pah8002_alg_buffer);
		_pah8002_alg_buffer = NULL;
	}
}

void pah8002_log(void)
{
	int i = 0 ;
	uint32_t *ppg_data = (uint32_t *)_pah8002_data.ppg_data ;
	int16_t *mems_data = _pah8002_data.mems_data ;
```

```
	log_printf("Frame Count, %d \n", _pah8002_data.frame_count);
	log_printf("Time, %d \n", _pah8002_data.time);
	log_printf("PPG, %d, %d, ", _pah8002_data.touch_flag, _pah8002_data.nf_ppg_per_channel);
	for(i=0; i<_pah8002_data.nf_ppg_channel * _pah8002_data.nf_ppg_per_channel; i++)
	{
		log_printf("%d, ", *ppg_data);
		ppg_data ++;
	}
	log_printf("\n");
	log_printf("MEMS, %d, ", _pah8002_data.nf_mems);
	for(i=0; i<_pah8002_data.nf_mems*3; i++)
	{
		log_printf("%d, ", *mems_data);
		mems_data ++;
	}
	log_printf("\n");
}

static void data_convert_4ch_to_3ch(uint32_t *pdata, uint32_t len)
{
	uint32_t i = 0, j = 0;
	for(i=0, j=2; j<len; i+=3, j+=4)
	{
		*(pdata+i+1) = *(pdata+j);
		*(pdata+i+2) = *(pdata+j+1);
	}
}

void pah8002_task(void)
{
	uint8_t ret;
	float hr = 0 ;
	uint32_t sys_tick;
	if(_pah8002_interrupt == 1)
	{
		_pah8002_interrupt = 0;

		if(_mode == TOUCH_MODE)
		{
			pah8002_enter_normal_mode();
			_timestamp = get_sys_tick();
			accelerometer_start();
		}
		else if(_mode == NORMAL_MODE || _mode == NORMAL_LONG_ET_MODE)
		{
			_pah8002_task();
```

```
                pah8002_dyn_switch_ppg_mode();

#ifdef PPG_MODE_ONLY
#else
                if(_touch_flag == 0)
                {
                        pah8002_enter_touch_mode();
                        accelerometer_stop();
                }
#endif

                //process algorithm
#ifdef MEMS_ZERO
#else
                accelerometer_get_fifo(&_pah8002_data.mems_data, &_pah8002_data.nf_mems);
#endif
                sys_tick = get_sys_tick();
                _pah8002_data.time = sys_tick - _timestamp;
                _timestamp = sys_tick;
                _pah8002_data.touch_flag = _touch_flag;
                data_convert_4ch_to_3ch((uint32_t                                *)pah8002_ppg_data,
HEART_RATE_MODE_SAMPLES_PER_READ);

                // log 3ch ppg_data before pah8002_entrance()
                pah8002_log();

                ret = pah8002_entrance(&_pah8002_data);
                if((ret & 0x0f) != 0)
                {
                        switch(ret) //check error status
                        {
                                case MSG_ALG_NOT_OPEN:
                                        debug_printf("Algorithm is not initialized.\r\n");
                                        break;
                                case MSG_MEMS_LEN_TOO_SHORT:
                                        debug_printf("MEMS data length is shorter than PPG data length.\r\n");
                                        break;
                                case MSG_NO_TOUCH:
                                        debug_printf("PPG is no touch.\r\n");
                                        break;
                                case MSG_PPG_LEN_TOO_SHORT:
                                        debug_printf("PPG data length is too short.\r\n");
                                        break;
                                case MSG_FRAME_LOSS:
                                        debug_printf("Frame count is not continuous.\r\n");
                                        break;
```

```
                        }
                    }
                    if((ret & 0xf0) == MSG_HR_READY)
                    {
                        pah8002_get_hr(&hr) ;
                        debug_printf("HR = %d\r\n", (int)(hr));
                    }
                    _pah8002_data.frame_count++;
                }
            }
}

void pah8002_intr_isr(void)
{
    _pah8002_interrupt = 1 ;
}
```

4.3.2 pah8002_comm_i2c.c

```
#include "pah8002_comm.h"
#include "i2c.h"

#define I2C_ID_PAH8002  0x15  //I2C 7-bit ID

uint8_t pah8002_write_reg(uint8_t addr, uint8_t data)
{
    return i2c_write_reg(I2C_ID_PAH8002, addr, data);
}
uint8_t pah8002_read_reg(uint8_t addr, uint8_t *data)
{
    return i2c_read_reg(I2C_ID_PAH8002, addr, data);
}
uint8_t pah8002_burst_read_reg(uint8_t addr, uint8_t *data, uint32_t rx_size)
{
    return i2c_burst_read_reg(I2C_ID_PAH8002, addr, data, rx_size);
}
```

4.3.3 pah8002_comm_spi.c

```
#include "pah8002_comm.h"
#include "spi.h"

static uint8_t		_real_bank = 0xFF;		// 0x00 ~ 0x03
static uint8_t		_spi_bank = 0xFF;		// 0x00 ~ 0x07

static int _set_bank(uint8_t addr);
static int _write_reg(uint8_t addr, uint8_t data);
static int _read_reg(uint8_t addr, uint8_t *data, uint32_t size);

uint8_t pah8002_write_reg(uint8_t addr, uint8_t data)
{
	uint8_t ret = 0;

	if (addr == 0x7F)
	{
		ret = _write_reg(0x7F, data);
		if (ret == 0)
		{
			_real_bank = data;
			_spi_bank = data;
		}
		return ret;
	}

	ret = _set_bank(addr);
	if (ret != 0)
		return ret;

	return _write_reg(addr, data);
}
uint8_t pah8002_read_reg(uint8_t addr, uint8_t *data)
{
	uint8_t ret = 0;

	ret = _set_bank(addr);
	if (ret != 0)
		return ret;

	return _read_reg(addr, data, 1);
}
uint8_t pah8002_burst_read_reg(uint8_t addr, uint8_t *data, uint32_t rx_size)
{
	uint8_t ret = 0;
```

```
	ret = _set_bank(addr);
	if (ret != 0)
		return ret;

	return _read_reg(addr, data, rx_size);
}

static int _set_bank(uint8_t addr)
{
	bool highpart = (addr >= 0x80);
	uint8_t spi_bank = _real_bank;

	if (highpart)
		spi_bank += 4;

	if (_spi_bank != spi_bank)
	{
		_spi_bank = spi_bank;

		// change bank
		return _write_reg(0x7F, spi_bank);
	}
	return 0;
}

static int _write_reg(uint8_t addr, uint8_t data)
{
	addr |= (0x80);  //write, bit7 = 1                ==> Write, bit 7 is 1
	return spi_write_reg(addr, data);
}

static int _read_reg(uint8_t addr, uint8_t *data, uint32_t size)
{
	addr &= (0x7F); //read, bit7 = 0                  ==> Read, bit 7 is 0
	return spi_burst_read_reg(addr, data, size);
}
```

4.4 Test Pattern

4.4.1 pah8002_testpattern.h

```
#ifndef _PAH8002_TESTPATTERN_
#define _PAH8002_TESTPATTERN_

#include <stdint.h>
#define NF_PPG_CHANNEL 3
#define NF_PPG_PER_CHANNEL 20
#define NF_MEMS_MAX 25
static uint32_t ppg[][NF_PPG_PER_CHANNEL*NF_PPG_CHANNEL] =
{

{ 23247, 87148, 101183, 23245, 87026, 100931, 23256, 86890, 100761, 23251, 86891, 100744, 23250, 86906, 100788,
23257, 87002, 100851, 23259, 87030, 100901, 23258, 87095, 100908, 23255, 87097, 100982, 23264, 87258, 101101,
23272, 87393, 101246, 23265, 87441, 101342, 23260, 87561, 101523, 23280, 87739, 101722, 23280, 87842, 101866,
23270, 87808, 101610, 23257, 87533, 101271, 23254, 87332, 101005, 23242, 87285, 100899, 23252, 87198, 100762,}

,{ 23221, 87146, 100801, 23223, 87153, 100691, 23242, 87152, 100583, 23229, 87162, 100671, 23241, 87175,
100666, 23255, 87252, 100826, 23249, 87382, 101026, 23262, 87523, 101165, 23245, 87603, 101341, 23262, 87737,
101518, 23263, 87849, 101535, 23257, 87731, 101236, 23237, 87539, 100986, 23237, 87504, 100905, 23234, 87446,
100881, 23220, 87435, 100792, 23228, 87462, 100807, 23240, 87548, 100838, 23246, 87620, 100950, 23250, 87661,
101015,}

,{ 23254, 87728, 101123, 23255, 87781, 101174, 23255, 87881, 101280, 23248, 87955, 101391, 23245, 88075,
101500, 23244, 88105, 101639, 23242, 88137, 101398, 23237, 87926, 101058, 23217, 87737, 100790, 23230, 87619,
100571, 23216, 87529, 100433, 23205, 87560, 100470, 23213, 87513, 100396, 23208, 87503, 100352, 23219, 87479,
100286, 23208, 87545, 100437, 23232, 87602, 100478, 23241, 87727, 100645, 23221, 87787, 100743, 23243, 87925,
100869,}

,{ 23240, 87996, 101056, 23247, 88139, 101197, 23243, 88129, 101007, 23249, 88020, 100723, 23223, 87882,
100526, 23234, 87799, 100395, 23216, 87748, 100370, 23231, 87766, 100337, 23215, 87750, 100338, 23222, 87723,
100240, 23203, 87681, 100240, 23212, 87762, 100317, 23213, 87803, 100367, 23208, 87839, 100443, 23212, 87935,
100529, 23209, 87989, 100604, 23211, 88050, 100728, 23210, 88123, 100808, 23213, 88083, 100559, 23189, 87871,
100291,}

,{ 23196, 87762, 100212, 23196, 87700, 100091, 23182, 87716, 100086, 23195, 87723, 100047, 23185, 87624,
100012, 23198, 87623, 100010, 23193, 87656, 100051, 23197, 87707, 100169, 23188, 87826, 100325, 23206, 87867,
100410, 23196, 87943, 100519, 23211, 88035, 100641, 23204, 88120, 100797, 23199, 88210, 100877, 23186, 88110,
100648, 23176, 87938, 100420, 23194, 87857, 100240, 23176, 87744, 100164, 23179, 87753, 100192, 23190, 87775,
100114,}

,{ 23177, 87720, 100154, 23176, 87760, 100120, 23184, 87767, 100119, 23184, 87797, 100153, 23167, 87810,
100186, 23164, 87900, 100356, 23181, 87991, 100397, 23177, 88007, 100507, 23178, 88060, 100613, 23145, 88143,
100793, 23174, 88114, 100619, 23162, 87948, 100306, 23150, 87716, 100108, 23144, 87676, 100045, 23155, 87674,
99994, 23152, 87642, 100014, 23148, 87602, 99874, 23151, 87517, 99911, 23145, 87573, 99945, 23157, 87652,
99990,}

,{ 23152, 87658, 100082, 23147, 87695, 100189, 23159, 87779, 100244, 23152, 87861, 100429, 23157, 87907,
100566, 23167, 88017, 100760, 23169, 88089, 100911, 23157, 88086, 100717, 23150, 87886, 100379, 23142, 87707,
```

100210, 23144, 87607, 100076, 23136, 87620, 100107, 23137, 87594, 100009, 23137, 87510, 99951, 23147, 87529, 99922, 23141, 87466, 99916, 23143, 87472, 99900, 23121, 87454, 99941, 23133, 87523, 100010, 23115, 87558, 100063,}

,{ 23118, 87585, 100217, 23120, 87674, 100480, 23115, 87783, 100571, 23118, 87832, 100500, 23114, 87701, 100222, 23108, 87543, 99986, 23085, 87426, 99895, 23086, 87374, 99881, 23091, 87358, 99899, 23086, 87353, 99840, 23105, 87322, 99826, 23090, 87355, 99909, 23100, 87401, 100025, 23105, 87484, 100102, 23098, 87504, 100246, 23115, 87557, 100316, 23099, 87668, 100442, 23102, 87730, 100625, 23100, 87828, 100805, 23097, 87923, 100927,}

,{ 23102, 87968, 100870, 23103, 87785, 100498, 23086, 87573, 100264, 23090, 87492, 100108, 23088, 87423, 100106, 23093, 87423, 100030, 23083, 87377, 99976, 23074, 87332, 99918, 23080, 87393, 99948, 23092, 87358, 100017, 23092, 87383, 100007, 23091, 87441, 100092, 23090, 87489, 100168, 23069, 87486, 100263, 23084, 87620, 100398, 23084, 87621, 100437, 23074, 87659, 100479, 23071, 87583, 100205, 23076, 87424, 99970, 23059, 87301, 99794,}

,{ 23071, 87253, 99781, 23052, 87247, 99715, 23064, 87244, 99621, 23054, 87140, 99517, 23053, 87091, 99496, 23063, 87094, 99533, 23058, 87104, 99547, 23042, 87134, 99560, 23057, 87196, 99690, 23037, 87240, 99736, 23050, 87304, 99887, 23061, 87395, 100002, 23062, 87447, 100096, 23045, 87397, 99862, 23040, 87231, 99536, 23031, 87102, 99328, 23033, 87068, 99227, 23027, 86968, 99083, 23016, 86954, 98970, 23013, 86886, 98804,}

};

static int16_t mems[][NF_MEMS_MAX*3] =

{

{ 16128, -640, -2112, 16064, -512, -2112, 16000, -384, -2112, 16128, -384, -2112, 16128, -704, -1920, 16192, -576, -1856, 16064, -768, -1920, 16128, -704, -1920, 16000, -576, -1856, 16128, -704, -1920, 16064, -640, -1856, 16128, -640, -1920, 16128, -512, -1856, 16128, -640, -1984, 16192, -512, -1792, 16192, -576, -2048, 16192, -640, -1920, 16128, -448, -1984, 16128, -640, -2112, 16256, -384, -1856, 16128, -576, -1792, 16128, -576, -1792, 16192, -512, -1856, 16064, -576, -1792,}

,{ 16064, -576, -1856, 16128, -704, -1728, 16128, -576, -1920, 16128, -448, -2048, 16128, -448, -1920, 16256, -704, -1728, 16064, -512, -1728, 16192, -512, -1920, 16128, -448, -1984, 16320, -704, -1792, 16320, -640, -1792, 15936, -704, -1792, 16064, -448, -1920, 16128, -384, -1792, 16192, -576, -1792, 16128, -640, -1792, 16064, -448, -1792, 16128, -448, -1856, 16128, -640, -1664, 16128, -640, -1728, 16192, -512, -1920, 16000, -512, -2112, 16128, -512, -1920,}

,{ 16192, -576, -1856, 16128, -768, -1792, 16064, -512, -1792, 16128, -448, -1856, 16128, -512, -1792, 16064, -704, -1728, 16064, -512, -1792, 16128, -640, -1792, 16192, -512, -1920, 16128, -512, -1856, 16064, -448, -2112, 16128, -448, -1920, 16192, -640, -1728, 16192, -512, -1664, 16256, -576, -1792, 16064, -576, -1856, 16000, -640, -1856, 16128, -576, -1920, 16256, -512, -1856, 16128, -448, -2048, 16064, -640, -1856, 16192, -512, -1984, 16128, -512, -1984, 16128, -512, -1920,}

,{ 16128, -512, -1920, 16128, -512, -2112, 16192, -448, -1792, 16256, -704, -1728, 16128, -576, -1856, 16192, -448, -1792, 16000, -512, -1920, 16192, -576, -1856, 16064, -512, -1856, 16128, -512, -1728, 16192, -576, -1728, 16128, -512, -1728, 16128, -512, -1728, 16064, -576, -1920, 16128, -640, -1856, 16128, -704, -1856, 16064, -384, -1856, 16128, -320, -1856, 16064, -640, -1792, 16192, -448, -1728, 16192, -512, -1792, 16192, -448, -1856, 16000, -512, -1984,}

```
,{ 16128, -640, -1856, 16064, -512, -1920, 16192, -448, -1792, 16128, -512, -1984, 16064, -448, -1920, 16128, -512, -1792, 16128, -512, -1728, 16128, -448, -1792, 16128, -576, -1856, 16128, -576, -1856, 16128, -448, -1920, 16192, -576, -1728, 16192, -512, -1856, 16128, -448, -1856, 16064, -576, -1792, 16128, -576, -1920, 16128, -448, -1856, 16128, -576, -1856, 16192, -512, -1984, 16128, -448, -1984, 16128, -448, -1984, 16192, -576, -1920, 16128, -448, -1920, 16064, -384, -1984,}

,{ 16000, -640, -1856, 16128, -512, -1792, 16192, -576, -1856, 16128, -384, -1920, 16064, -448, -1920, 16064, -512, -1984, 16128, -576, -1792, 16192, -512, -2048, 16256, -448, -1920, 16192, -576, -1920, 16128, -512, -1984, 16128, -448, -1856, 16128, -512, -1728, 16128, -512, -1792, 16192, -448, -1856, 16064, -512, -1920, 16128, -320, -1856, 16064, -448, -1856, 16128, -512, -1856, 16128, -384, -1856, 16128, -384, -1728, 16256, -448, -1984, 16128, -448, -1792,}

,{ 16128, -320, -1856, 16128, -576, -1792, 16192, -576, -1920, 16128, -448, -1920, 16128, -320, -1856, 16064, -512, -1856, 16192, -576, -1920, 16064, -320, -1920, 16192, -448, -1920, 16064, -512, -1792, 16192, -512, -1920, 16064, -512, -1984, 16192, -640, -1920, 16192, -448, -1920, 16128, -512, -1920, 16128, -448, -2048, 16192, -256, -2112, 16128, -576, -1856, 16128, -448, -1920, 16192, -512, -1792, 16064, -512, -1920, 16128, -512, -1856, 16128, -512, -1856, 16128, -384, -1984,}

,{ 16000, -384, -2048, 16128, -448, -1856, 16192, -448, -1856, 16192, -448, -1984, 16128, -320, -1920, 16064, -640, -1792, 16064, -448, -1920, 16128, -384, -1920, 16128, -448, -1920, 16256, -320, -1856, 16192, -640, -1792, 16128, -512, -1728, 16256, -512, -1792, 16064, -576, -1856, 16128, -512, -1856, 16192, -320, -1856, 16320, -576, -1920, 16192, -640, -1728, 16128, -256, -1856, 16128, -384, -1792, 16128, -384, -1856, 16064, -512, -1856, 16128, -448, -1792,}

,{ 16192, -512, -1728, 16000, -448, -1856, 16256, -256, -1792, 16128, -256, -1792, 16192, -448, -1856, 16192, -576, -1792, 16192, -320, -1920, 16128, -512, -1728, 16192, -320, -1856, 16256, -448, -1856, 16128, -576, -1728, 16128, -384, -1856, 16192, -448, -1856, 16128, -640, -1920, 16256, -512, -1856, 16000, -320, -1856, 16064, -384, -1920, 16128, -384, -1920, 16192, -512, -1920, 16128, -448, -1856, 16128, -576, -1856, 16128, -448, -1920, 16192, -512, -1856, 16128, -512, -1856,}

,{ 16128, -384, -1856, 16192, -448, -1792, 16128, -512, -1856, 16128, -384, -1856, 16128, -448, -1600, 16256, -448, -1984, 16128, -256, -1920, 16192, -320, -1792, 16128, -512, -1728, 16128, -448, -1792, 16128, -384, -1792, 16192, -448, -1856, 16128, -512, -1792, 16192, -320, -1856, 16064, -512, -1792, 16256, -448, -1792, 16128, -448, -1856, 16192, -384, -1728, 16128, -576, -1920, 16256, -448, -1728, 16128, -448, -1728, 16192, -384, -1856, 16128, -384, -1664,}


};

static const uint32_t nf_mems[] =

{
        24, 23, 24, 23, 24, 23, 24, 23, 24, 23,
};

#endif
```

4.4.2 pah8002_test_algorithm.c

```
#include "pah8002_api_c.h"
#include "pah8002_testpattern.h"

#include <stdlib.h>
#include <string.h>

float test_library(void)
{
	uint8_t ret = 0;
	uint32_t version = 0;
	float myHR = 0;
	int i = 0;
	void *pah8002_buffer = NULL;
	pah8002_data_t pah8002_data;

	pah8002_buffer = malloc(pah8002_query_open_size());
	ret = pah8002_open(pah8002_buffer);
	if (ret != MSG_SUCCESS)
		return -1.0;

	version = pah8002_version();

	memset(&pah8002_data, 0, sizeof(pah8002_data));
	pah8002_data.time = 0;
	pah8002_data.nf_ppg_channel = NF_PPG_CHANNEL;
	pah8002_data.nf_ppg_per_channel = NF_PPG_PER_CHANNEL;
	pah8002_data.touch_flag = 1;

	for (i = 0; i < sizeof(nf_mems)/sizeof(nf_mems[0]); i++)
	{
		pah8002_data.frame_count = i ;
		pah8002_data.nf_mems = nf_mems[i] ;
		pah8002_data.mems_data = (int16_t *)mems[i] ;
		pah8002_data.ppg_data = (int32_t *)ppg[i];

		ret = pah8002_entrance(&pah8002_data);
		if ((ret& (3<<4)) == (3<<4))
		{
			pah8002_get_hr(&myHR);

			if (myHR > 0)
			{
				break;
			}
		}
```

```
        }

        pah8002_close();

        return myHR;
}
```

5.0 Example of Android ADSP Driver

1. Launch ""QSensorTest" App.

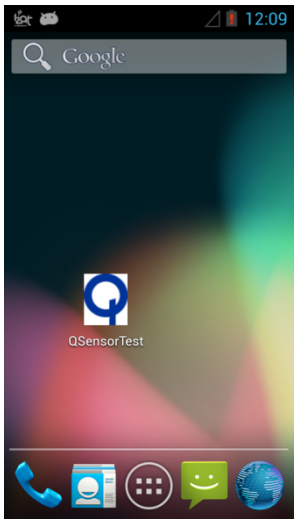

Figure 5. Step 1 – QsensorTest APP Launch

2. Select "SELF TEST"

Figure 6. Step 2 - Self Test

3. Run "MAG: PIXART_PAH8002#". To call for function "sns_dd_pixart_run_test" in driver. In this function, we check the sensor ID and do software reset.

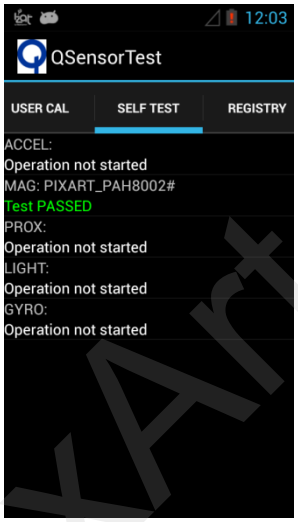

Figure 7. Step 3 – Enable Driver

4. Select "STREAMIMG"

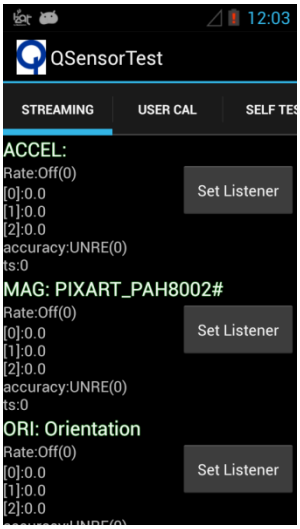

Figure 8. Step 4 – Streaming

5. Click “Set Listener” button of “MAG: PIXART_PAH8002@” and fill “100000” for IR Touch Detection (IR LED channel only) and Click “Submit” button to start Touch Detection function.

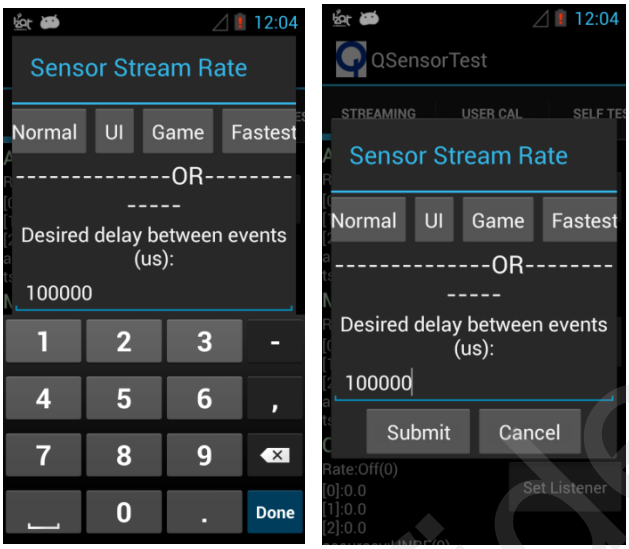

Figure 9. Step 5 – IR Touch Detection Setting

6. Enable mini-dim to get debug message. (For example: mini-dm –comport com41).

When touch the sensor, it appears … touch flag = 128 …., and INT pin keeps high.
When de-touch, it appears … touch flag = 0 …., and INT pin keeps low.

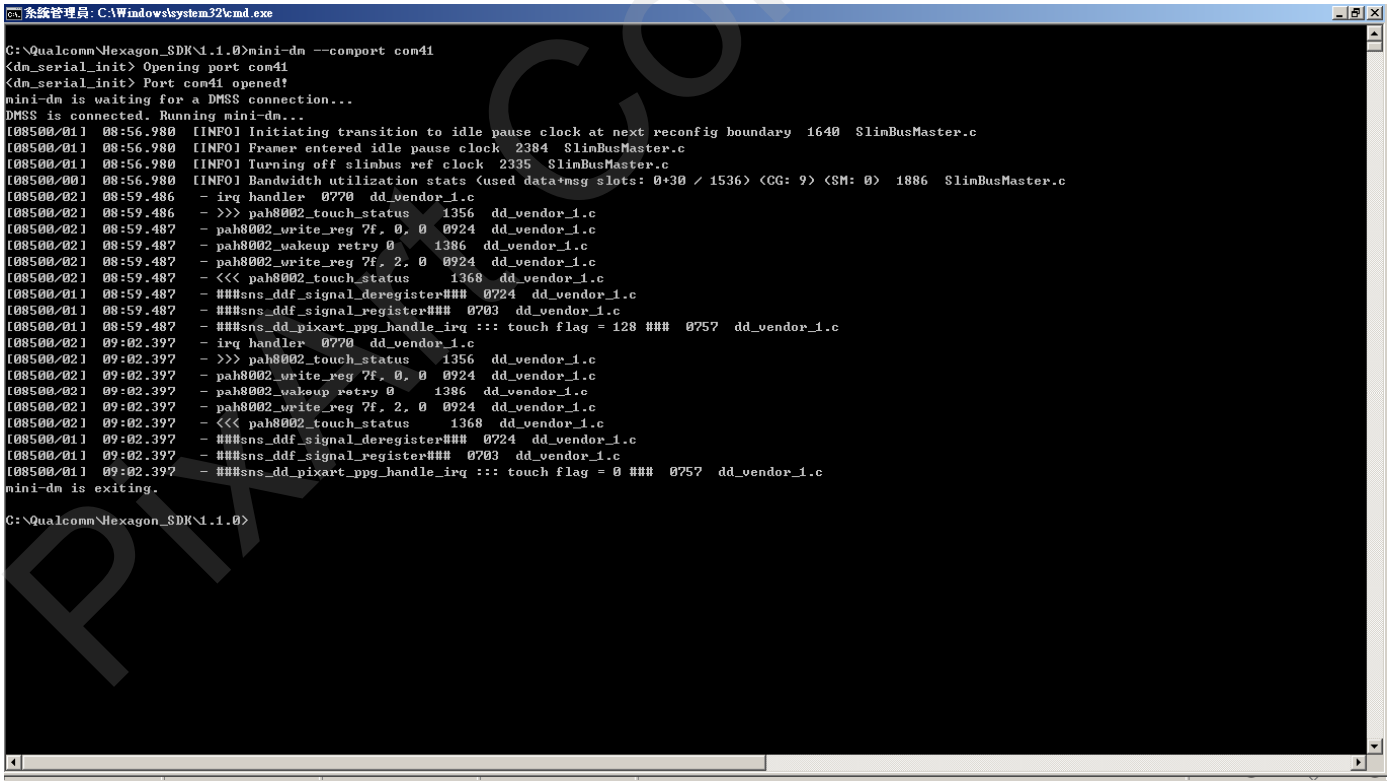

Figure 10. Step 7 – Enable Mini-Dim

7. Click “Remove Listener” button, sensor will enter sleep mode.
8. Re-click “Set Listener” button of “MAG: PIXART_PAH8002#”, fill in “50000” as delay for PPG mode (1 IR channel + 2 Green LED channels), and click “Submit” button to start Heart Rate PPG Detection

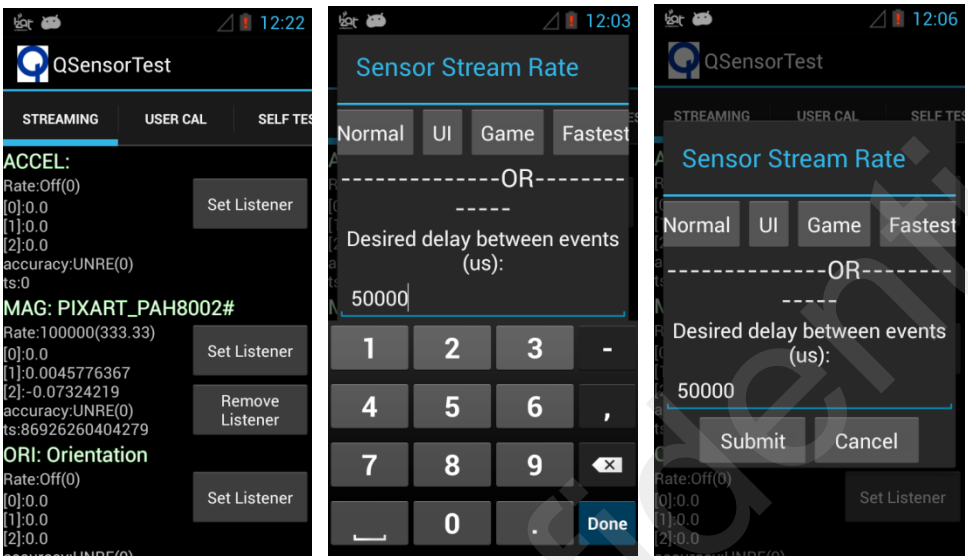

Figure 11. Step 9 – Heart Rate PPG Detection Setting

9. In mini-dim message window, you can see the message dumps the latest raw data of three channels.

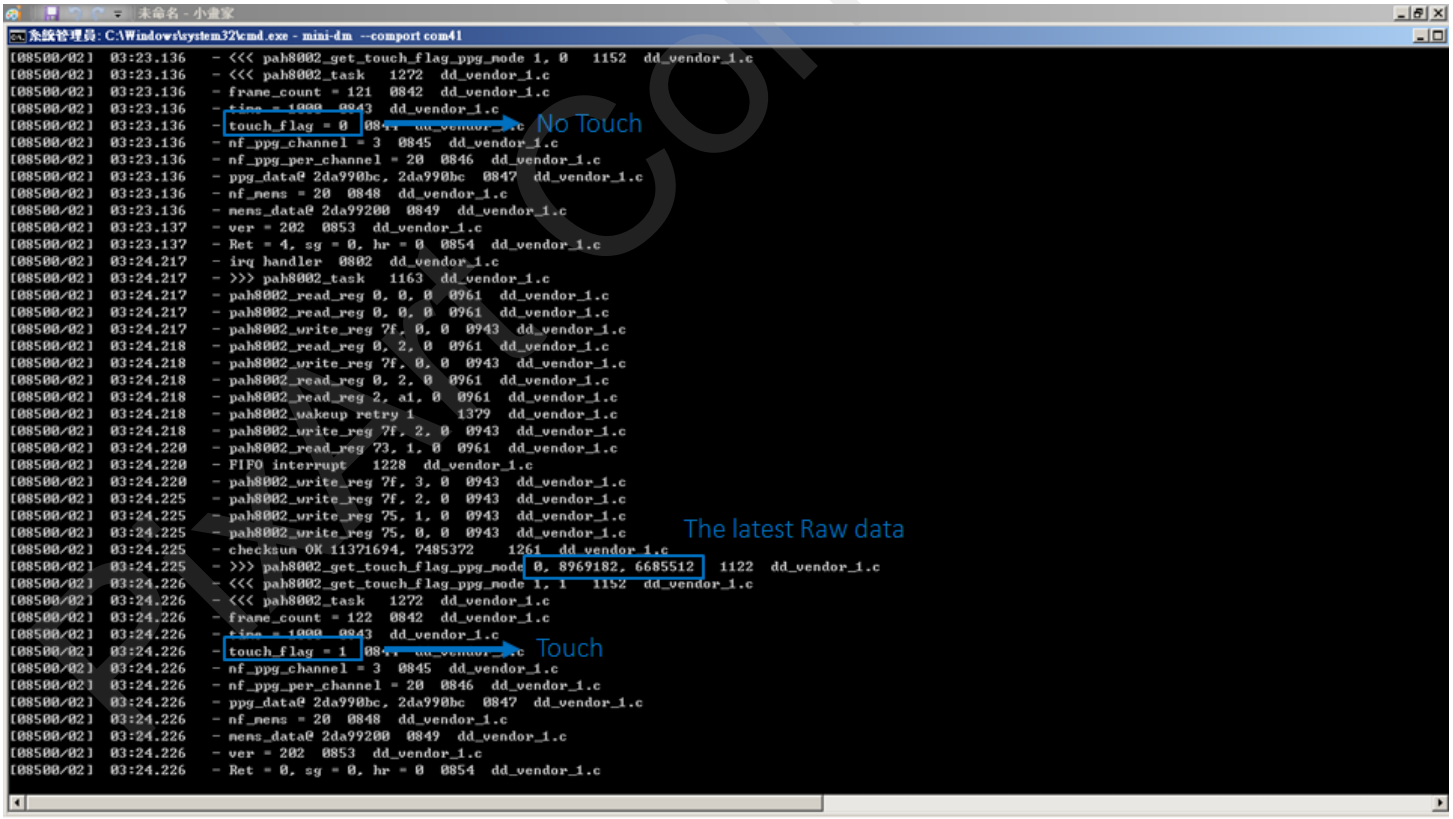

Figure 12. Step 10 – Mini-Dim Message Dump

10. In PPG mode, when FIFO interrupt occurs, the function “sns_dd_pixart_ppg_irq_handler” is called. If the “PXIALG_INSIDE” flag is defined, it will feed Pixart’s algorithm with PPG data for data processing then print out the Heart Rate result message as “HR = 98, ret = 0 ….”.

11. In PPG mode, only FIFO (Raw) interrupt is supported. INT pin will output a pulse signal when FIFO data is ready. The “Touch” status is being recognized by FW algorithm. Refer to “pah8002_get_touch_flag_ppg_mode” function.

    When touch, it shows “touch_flag = 1”.
    When de-touch, it shows “touch_flag = 0”.

    Please be noted that PAH8002 outputs numbers of data per interrupt (driver default is 60 data for three channels per interrupt) in the order of CH#0, CH#1, CH#2, CH#0, CH#1, CH#2, and so on from the FIFO.

12. The ADSP driver registry setting is as below.

```
-----------------------------
item-registry-ID: name: value...
1934-1935: UUID: (UNKNOWN DRV) 61c69537-9675-40c1-bc46-fd1b43b8250b
1936: off_to_idle:        0     1937: idle_to_ready:      0     1938: i2c_bus:         0x0c
1939: reg_group_id:    1020     1940: cal_grp_id:     65535     1941: gpio1:         0x0049
1942: gpio2:          0xffff    1943: sensor_id:         20     1944: i2c_address:     0x15
1945: data_type1:         2     1946: data_type2:         0     1947: rel_sns_idx:       -1
1948: sens_default:       0     1949: flags:           0x80
1984: device_select:      0
drv_cfg[3]
```

Figure 13. Step 14 – ADSP Driver Registry Setting

6.0 Appendices

6.1 Touch Setting of 3.8Hz for INT Application

```
const uint8_t init_touch_register_INT_array[][2] = {
{0x7f, 0x01},//switch to bank1
{0xe6, 0xC8},
{0xe7, 0x00},
{0xF1, 0x00},
{0x07, 0x01},
{0xAE, 0x06},
{0xAF, 0x07},
{0xBA, 0x7C},
{0x6C, 0x10},
{0x6D, 0x10},
{0x7A, 0x01},
{0x6F, 0x10},
{0x7F, 0x00},//switch to bank0
{0x08, 0xFF},
{0x09, 0x03},
{0x5A, 0x01},
{0x5C, 0x58},//Touch Threshold
{0x5D, 0x02},
{0x60, 0x00},
{0x61, 0x02},
{0x64, 0x01},
{0x65, 0x01},
{0x35, 0x80},
{0x36, 0x02},
{0x8C, 0x00},
{0x8E, 0x00},
{0xDE, 0x00},
{0xD9, 0x01},
{0xDD, 0x04},
{0x3B, 0x01},
{0x43, 0x00},
{0x47, 0x01},
{0x48, 0x00},
{0x49, 0x00},
{0x4A, 0x01},
{0x4D, 0x01},
{0x16, 0x01},
{0x13, 0x01},
{0x14, 0x01},
{0x15, 0x01},
{0x50, 0x01},
{0x51, 0x01},
{0x59, 0x00},
{0x57, 0x00},
{0x6B, 0x00},
{0x6C, 0x00},
{0x3E, 0x00},
{0x0D, 0x78},
{0x0E, 0x00},
```

```
{0x7F, 0x02},//switch to bank2
{0x17, 0x00},
{0x18, 0x00},
{0x1B, 0x01},
{0x1C, 0x01},
{0x25, 0x02},
{0x29, 0x00},
{0x2d, 0x01},
{0x4F, 0x0C},
{0x66, 0x01},
{0x67, 0x01},
{0x68, 0x01},
{0x69, 0x01},
{0x6A, 0x01},
//{0x6B, 0x01},
//{0x6C, 0x01},
{0x6D, 0x01},
{0x6E, 0x01},
{0x6F, 0x01},
{0x70, 0x01},
{0x74, 0x01},
{0x76, 0x00},
{0x7A, 0x01},
{0x7B, 0xFF},
{0x8D, 0x01},
{0x8F, 0x01},
{0x92, 0x00},
{0x7F, 0x01},//switch to bank1
{0xA2, 0x40},
{0x7C, 0x01},
{0x4C, 0x01},
{0x4F, 0x07},
{0x3F, 0x04},
{0x0C, 0x05},
{0x4D, 0x05},
{0x52, 0x05},
{0x86, 0x50},
{0x92, 0x1C},
{0x98, 0x1D},
{0x9A, 0x42},
{0x81, 0x01},
{0x3B, 0x00},
{0xEA, 0xC9},
{0xA4, 0x50},
{0xA5, 0x00},
{0xA6, 0x52},
{0xA7, 0x00},
{0xA8, 0x53},
{0xA9, 0x00},
{0xAD, 0x00},
{0xD6, 0xFF},
{0xD7, 0x1F},
{0xD8, 0x01},
```

```
{0xD9, 0x00},
{0xDA, 0x10},
{0xDB, 0x00},
{0xDC, 0x16},
{0xDD, 0x00},
{0xDE, 0x17},
{0xDF, 0x00},
{0xE0, 0xFE},
{0xE1, 0x1F},
};
#define INIT_TOUCH_INT_ARRAY_SIZE (sizeof(init_touch_register_INT_array)/sizeof(init_touch_register_INT_array[0]))
```

6.2 Touch Setting of 3.8Hz for Touch Flag Application

```
const uint8_t init_touch_register_array[][2] = {
{0x7f, 0x01},       //switch to bank1
{0x4C, 0x00},
{0xe6, 0xC8},
{0xe7, 0x00},
{0xF1, 0x00},
{0x07, 0x01},
{0xAE, 0x06},
{0xAF, 0x07},
{0xBA, 0x7C},
{0x6C, 0x10},
{0x6D, 0x10},
{0x7A, 0x01},
{0x6F, 0x10},
{0x7F, 0x00},       //switch to bank0
{0x08, 0xFF},
{0x09, 0x03},
{0x5A, 0x01},
{0x5C, 0x58},       //Touch Threshold
{0x5D, 0x02},
{0x60, 0x00},
{0x61, 0x02},
{0x64, 0x01},
{0x65, 0x01},
{0x35, 0x80},
{0x36, 0x02},
{0x84, 0x78},
{0x8C, 0x00},
{0x8E, 0x00},
{0xDE, 0x00},
{0xD9, 0x01},
{0xDD, 0x04},
{0x3B, 0x01},
{0x43, 0x00},
{0x47, 0x01},
{0x48, 0x00},
{0x49, 0x00},
{0x4A, 0x01},
{0x4D, 0x01},
{0x16, 0x01},
```

```
{0x13, 0x01},
{0x14, 0x01},
{0x15, 0x01},
{0x50, 0x01},
{0x51, 0x01},
{0x59, 0x00},
{0x57, 0x00},
{0x6B, 0x00},
{0x6C, 0x00},
{0x3E, 0x00},
{0x0D, 0x78},
{0x0E, 0x00},
{0x7F, 0x02},       //switch to bank2
{0x17, 0x00},
{0x18, 0x00},
{0x1B, 0x01},
{0x1C, 0x01},
{0x25, 0x04},
{0x29, 0x00},
{0x2d, 0x01},
{0x4F, 0x0C},
{0x66, 0x01},
{0x67, 0x01},
{0x68, 0x01},
{0x69, 0x01},
{0x6A, 0x01},
//{0x6B, 0x01},
//{0x6C, 0x01},
{0x6D, 0x01},
{0x6E, 0x01},
{0x6F, 0x01},
{0x70, 0x01},
{0x74, 0x01},
{0x76, 0x01},
{0x8D, 0x01},
{0x8F, 0x01},
{0x92, 0x00},
{0x7F, 0x01},       //switch to bank1
{0xA2, 0x40},
{0x7C, 0x01},
{0x4F, 0x07},
{0x3F, 0x04},
{0x0C, 0x05},
{0x4D, 0x05},
{0x52, 0x05},
{0x86, 0x50},
{0x92, 0x1C},
{0x98, 0x1D},
{0x9A, 0x42},
{0x81, 0x01},
{0x3B, 0x00},
{0xEA, 0xC9},
{0xA4, 0x50},
```

```
{0xA5, 0x00},
{0xA6, 0x52},
{0xA7, 0x00},
{0xA8, 0x53},
{0xA9, 0x00},
{0xAD, 0x00},
{0xD6, 0xFF},
{0xD7, 0x1F},
{0xD8, 0x01},
{0xD9, 0x00},
{0xDA, 0x10},
{0xDB, 0x00},
{0xDC, 0x16},
{0xDD, 0x00},
{0xDE, 0x17},
{0xDF, 0x00},
{0xE0, 0xFE},
{0xE1, 0x1F},
};
#define INIT_TOUCH_REG_ARRAY_SIZE (sizeof(init_touch_register_array)/sizeof(init_touch_register_array[0]))
```

6.3 PPG Setting of 20Hz

```
const uint8_t init_ppg_register_array[][2] = {
{0x7f, 0x01},       //switch to bank1
{0xE6, 0xC8},
{0xE7, 0x00},
{0x07, 0x01},
{0xAE, 0x06},
{0xAF, 0x07},
{0x4D, 0x00},
{0xBA, 0x7C},
{0xBB, 0x7C},
{0xBC, 0x7C},
{0xBD, 0x06},
{0xBE, 0x06},
{0xBF, 0x06},
{0xB1, 0x06},
{0xB2, 0x06},
{0xB3, 0x06},
{0x6A, 0x00},
{0x6B, 0x01},
{0x6C, 0x10},
{0x6D, 0x10},
{0x7A, 0x00},
{0x6F, 0x10},
{0x7F, 0x00},       //switch to bank0
{0x08, 0xFF},
{0x09, 0x03},
{0x4F, 0x0C},
{0xE6, 0x07},
{0x8C, 0x00},
{0xAE, 0x01},
```

```
{0xD0, 0x01},
{0x8E, 0x00},
{0xD2, 0x01},
{0xB0, 0x01},
{0x27, 0x80},
{0x28, 0x12},
{0x35, 0xC0},
{0x36, 0x12},
{0x37, 0xC0},
{0x38, 0x12},
{0x39, 0xC0},
{0x3A, 0x12},
{0xDE, 0x00},
{0xD9, 0x01},
{0xDD, 0x04},
{0x3B, 0x01},
{0x3C, 0x15},
{0x3D, 0x15},
{0x47, 0x01},
{0x48, 0x01},
{0x49, 0x01},
{0x4A, 0x01},
{0x4B, 0x00},
{0x4C, 0x00},
{0x4D, 0x00},
{0x16, 0x00},
{0x13, 0x01},
{0x14, 0x01},
{0x15, 0x01},
{0x50, 0x00},
{0x59, 0x00},
{0x56, 0x00},
{0x57, 0x00},
{0x6B, 0x01},
{0x6C, 0x00},
{0x8F, 0x01},
{0xB1, 0x01},
{0x3E, 0x02},
{0x3F, 0x04},
{0x40, 0x04},
{0x0D, 0x78},
{0x0E, 0x00},
{0x0F, 0xF0},
{0x10, 0x00},
{0x11, 0xF0},
{0x12, 0x00},
{0x6D, 0xF0},
{0x6E, 0x00},
{0x6F, 0x00},
{0x70, 0x02},
{0x71, 0x10},
{0x72, 0x00},
{0x77, 0x00},
```

```
{0x78, 0x0C},
{0x79, 0x00},
{0x7A, 0x08},
{0x7B, 0x00},
{0x7C, 0x0B},
{0x7D, 0x00},
{0x7E, 0x09},
{0x80, 0x00},
{0x81, 0x0D},
{0x82, 0x00},
{0x83, 0x07},
{0x85, 0x01},
{0x90, 0xF0},
{0x91, 0x00},
{0x92, 0x20},
{0x93, 0x12},
{0x94, 0x10},
{0x95, 0x00},
{0x9A, 0x00},
{0x9B, 0x0C},
{0x9C, 0x00},
{0x9D, 0x08},
{0x9E, 0x00},
{0x9F, 0x0B},
{0xA0, 0x00},
{0xA1, 0x09},
{0xA2, 0x00},
{0xA3, 0x0D},
{0xA4, 0x00},
{0xA5, 0x07},
{0xA7, 0x01},
{0xB2, 0xF0},
{0xB3, 0x00},
{0xB4, 0x20},
{0xB5, 0x12},
{0xB6, 0x10},
{0xB7, 0x00},
{0xBC, 0x00},
{0xBD, 0x0C},
{0xBE, 0x00},
{0xBF, 0x08},
{0xC0, 0x00},
{0xC1, 0x0B},
{0xC2, 0x00},
{0xC3, 0x09},
{0xC4, 0x00},
{0xC5, 0x0D},
{0xC6, 0x00},
{0xC7, 0x07},
{0xC9, 0x01},
{0x7F, 0x02},       //switch to bank2
{0x17, 0x00},
{0x18, 0x00},
```

```
{0x1B, 0x01},
{0x1C, 0x01},
{0x25, 0x02},
{0x29, 0x00},
{0x2d, 0x01},
{0x4F, 0x10},
{0x66, 0x01},
{0x67, 0x01},
{0x68, 0x01},
{0x69, 0x01},
{0x6A, 0x01},
//{0x6B, 0x01},
//{0x6C, 0x01},
{0x6D, 0x01},
{0x6E, 0x01},
{0x70, 0x01},
{0x7B, 0xFF},
{0x7F, 0x01},       //switch to bank1
{0xA2, 0x40},
{0x7C, 0x01},
{0x4F, 0x07},
{0x3F, 0x04},
{0x0C, 0x05},
{0x4D, 0x05},
{0x52, 0x05},
{0x86, 0x50},
{0x92, 0x1C},
{0x98, 0x1D},
{0x9A, 0x42},
{0x81, 0x01},
{0x3B, 0x00},
{0xEA, 0xC9},
{0xA4, 0x50},
{0xA5, 0x00},
{0xA6, 0x52},
{0xA7, 0x00},
{0xA8, 0x53},
{0xA9, 0x00},
{0xD6, 0x40},
{0xD7, 0x06},
{0xD8, 0x01},
{0xD9, 0x00},
{0xDA, 0x11},
{0xDB, 0x00},
{0xDC, 0x84},
{0xDD, 0x02},
{0xDE, 0x85},
{0xDF, 0x02},
{0xE0, 0x3F},
{0xE1, 0x06},
};
#define INIT_PPG_REG_ARRAY_SIZE (sizeof(init_ppg_register_array)/sizeof(init_ppg_register_array[0]))
```

6.4 PPG Setting of 20Hz Long Exposure Time

```
const uint8_t init_ppg_long_register_array[][2] = {
{0x7f, 0x01},       //switch to bank1
{0xE6, 0xC8},
{0xE7, 0x00},
{0x07, 0x01},
{0xAE, 0x06},
{0xAF, 0x07},
{0x4D, 0x00},
{0xBA, 0x7C},
{0xBB, 0x7C},
{0xBC, 0x7C},
{0xBD, 0x06},
{0xBE, 0x06},
{0xBF, 0x06},
{0xB1, 0x06},
{0xB2, 0x06},
{0xB3, 0x06},
{0x6A, 0x00},
{0x6B, 0x01},
{0x6C, 0x10},
{0x6D, 0x10},
{0x7A, 0x00},
{0x6F, 0x10},
{0x7F, 0x00},       //switch to bank0
{0x08, 0xFF},
{0x09, 0x03},
{0x4F, 0x0C},
{0xE6, 0x07},
{0x8C, 0x00},
{0xAE, 0x01},
{0xD0, 0x01},
{0x8E, 0x00},
{0xD2, 0x01},
{0xB0, 0x01},
{0x27, 0x40},
{0x28, 0x25},
{0x35, 0x80},
{0x36, 0x25},
{0x37, 0x80},
{0x38, 0x25},
{0x39, 0x80},
{0x3A, 0x25},
{0xDE, 0x00},
{0xD9, 0x01},
{0xDD, 0x04},
{0x3B, 0x01},
{0x3C, 0x0A},
{0x3D, 0x0A},
{0x47, 0x01},
{0x48, 0x01},
{0x49, 0x01},
{0x4A, 0x01},
```

```
{0x4B, 0x00},
{0x4C, 0x00},
{0x4D, 0x00},
{0x16, 0x00},
{0x13, 0x01},
{0x14, 0x01},
{0x15, 0x01},
{0x50, 0x00},
{0x59, 0x00},
{0x56, 0x00},
{0x57, 0x00},
{0x6B, 0x01},
{0x6C, 0x00},
{0x8F, 0x01},
{0xB1, 0x01},
{0x3E, 0x02},
{0x3F, 0x02},
{0x40, 0x02},
{0x0D, 0x78},
{0x0E, 0x00},
{0x0F, 0xC0},
{0x10, 0x12},
{0x11, 0xC0},
{0x12, 0x12},
{0x6D, 0xF0},
{0x6E, 0x00},
{0x6F, 0x00},
{0x70, 0x02},
{0x71, 0x10},
{0x72, 0x00},
{0x77, 0x00},
{0x78, 0x0C},
{0x79, 0x00},
{0x7A, 0x08},
{0x7B, 0x00},
{0x7C, 0x0B},
{0x7D, 0x00},
{0x7E, 0x09},
{0x80, 0x00},
{0x81, 0x0D},
{0x82, 0x00},
{0x83, 0x07},
{0x85, 0x01},
{0x90, 0xF0},
{0x91, 0x00},
{0x92, 0x00},
{0x93, 0x25},
{0x94, 0x10},
{0x95, 0x00},
{0x9A, 0x00},
{0x9B, 0x0C},
{0x9C, 0x00},
{0x9D, 0x08},
```

```
{0x9E, 0x00},
{0x9F, 0x0B},
{0xA0, 0x00},
{0xA1, 0x09},
{0xA2, 0x00},
{0xA3, 0x0D},
{0xA4, 0x00},
{0xA5, 0x07},
{0xA7, 0x01},
{0xB2, 0xF0},
{0xB3, 0x00},
{0xB4, 0x00},
{0xB5, 0x25},
{0xB6, 0x10},
{0xB7, 0x00},
{0xBC, 0x00},
{0xBD, 0x0C},
{0xBE, 0x00},
{0xBF, 0x08},
{0xC0, 0x00},
{0xC1, 0x0B},
{0xC2, 0x00},
{0xC3, 0x09},
{0xC4, 0x00},
{0xC5, 0x0D},
{0xC6, 0x00},
{0xC7, 0x07},
{0xC9, 0x01},
{0x7F, 0x02},       //switch to bank2
{0x17, 0x00},
{0x18, 0x00},
{0x1B, 0x01},
{0x1C, 0x01},
{0x25, 0x02},
{0x29, 0x00},
{0x2d, 0x01},
{0x4F, 0x10},
{0x66, 0x01},
{0x67, 0x01},
{0x68, 0x01},
{0x69, 0x01},
{0x6A, 0x01},
//{0x6B, 0x01},
//{0x6C, 0x01},
{0x6D, 0x01},
{0x6E, 0x01},
{0x70, 0x01},
{0x7B, 0xFF},
{0x7F, 0x01},       //switch to bank1
{0xA2, 0x40},
{0x7C, 0x01},
{0x4F, 0x07},
{0x3F, 0x04},
```

```
{0x0C, 0x05},
{0x4D, 0x05},
{0x52, 0x05},
{0x86, 0x50},
{0x92, 0x1C},
{0x98, 0x1D},
{0x9A, 0x42},
{0x81, 0x01},
{0x3B, 0x00},
{0xEA, 0xC9},
{0xA4, 0x50},
{0xA5, 0x00},
{0xA6, 0x52},
{0xA7, 0x00},
{0xA8, 0x53},
{0xA9, 0x00},
{0xD6, 0x40},
{0xD7, 0x06},
{0xD8, 0x01},
{0xD9, 0x00},
{0xDA, 0x11},
{0xDB, 0x00},
{0xDC, 0xC4},
{0xDD, 0x02},
{0xDE, 0xC5},
{0xDF, 0x02},
{0xE0, 0x3F},
{0xE1, 0x06},
};
#define INIT_PPG_LONG_REG_ARRAY_SIZE
(sizeof(init_ppg_long_register_array)/sizeof(init_ppg_long_register_array[0]))
```

6.5 PPG Setting of 200Hz

```
const uint8_t init_stress_register_array[][2] = {
{0x7f, 0x01},       //switch to bank1
{0xE6, 0xC8},
{0xE7, 0x00},
{0x07, 0x01},
{0xAE, 0x06},
{0xAF, 0x07},
{0x4D, 0x00},
{0xBA, 0x7C},
{0xBB, 0x7C},
{0xBC, 0x7C},
{0xBD, 0x06},
{0xBE, 0x06},
{0xBF, 0x06},
{0xB1, 0x06},
{0xB2, 0x06},
{0xB3, 0x06},
{0x6A, 0x00},
{0x6B, 0x01},
```

```
{0x6C, 0x10},
{0x6D, 0x10},
{0x7A, 0x00},
{0x6F, 0x08},
{0x7F, 0x00},       //switch to bank0
{0x08, 0xFF},
{0x09, 0x03},
{0x4F, 0x0C},
{0xE6, 0x07},
{0x8C, 0x00},
{0xAE, 0x01},
{0xD0, 0x01},
{0x8E, 0x00},
{0xD2, 0x01},
{0xB0, 0x01},
{0x27, 0x60},
{0x28, 0x0F},
{0x35, 0xA0},
{0x36, 0x0F},
{0x37, 0xA0},
{0x38, 0x0F},
{0x39, 0xA0},
{0x3A, 0x0F},
{0xDE, 0x00},
{0xD9, 0x01},
{0xDD, 0x04},
{0x3B, 0x01},
{0x3C, 0x02},
{0x3D, 0x02},
{0x43, 0x00},
{0x44, 0x00},
{0x45, 0x00},
{0x47, 0x01},
{0x48, 0x01},
{0x49, 0x01},
{0x4A, 0x01},
{0x4B, 0x00},
{0x4C, 0x00},
{0x4D, 0x00},
{0x16, 0x00},
{0x13, 0x01},
{0x14, 0x01},
{0x15, 0x01},
{0x50, 0x00},
{0x59, 0x00},
{0x56, 0x00},
{0x57, 0x00},
{0x6B, 0x01},
{0x6C, 0x00},
{0x8F, 0x01},
{0xB1, 0x01},
{0x3E, 0x00},
{0x3F, 0x00},
```

```
{0x40, 0x00},
{0x0D, 0x78},
{0x0E, 0x00},
{0x0F, 0xF0},
{0x10, 0x00},
{0x11, 0xF0},
{0x12, 0x00},
{0x6D, 0xF0},
{0x6E, 0x00},
{0x6F, 0x00},
{0x70, 0x02},
{0x71, 0x10},
{0x72, 0x00},
{0x77, 0x00},
{0x78, 0x0C},
{0x79, 0x00},
{0x7A, 0x08},
{0x7B, 0x00},
{0x7C, 0x0B},
{0x7D, 0x00},
{0x7E, 0x09},
{0x80, 0x00},
{0x81, 0x0D},
{0x82, 0x00},
{0x83, 0x07},
{0x85, 0x01},
{0x90, 0xF0},
{0x91, 0x00},
{0x92, 0x40},
{0x93, 0x0F},
{0x94, 0x10},
{0x95, 0x00},
{0x9A, 0x00},
{0x9B, 0x0C},
{0x9C, 0x00},
{0x9D, 0x08},
{0x9E, 0x00},
{0x9F, 0x0B},
{0xA0, 0x00},
{0xA1, 0x09},
{0xA2, 0x00},
{0xA3, 0x0D},
{0xA4, 0x00},
{0xA5, 0x07},
{0xA7, 0x01},
{0xB2, 0xF0},
{0xB3, 0x00},
{0xB4, 0x40},
{0xB5, 0x0F},
{0xB6, 0x10},
{0xB7, 0x00},
{0xBC, 0x00},
{0xBD, 0x0C},
```

```
{0xBE, 0x00},
{0xBF, 0x08},
{0xC0, 0x00},
{0xC1, 0x0B},
{0xC2, 0x00},
{0xC3, 0x09},
{0xC4, 0x00},
{0xC5, 0x0D},
{0xC6, 0x00},
{0xC7, 0x07},
{0xC9, 0x01},
{0x7F, 0x02},       //switch to bank2
{0x17, 0x00},
{0x18, 0x00},
{0x1B, 0x01},
{0x1C, 0x01},
{0x25, 0x02},
{0x29, 0x00},
{0x2d, 0x01},
{0x4F, 0x0C},
{0x66, 0x01},
{0x67, 0x01},
{0x68, 0x01},
{0x69, 0x01},
{0x6A, 0x01},
{0x6D, 0x01},
{0x6E, 0x01},
{0x70, 0x01},
{0x7B, 0xFF},
{0x7F, 0x01},       //switch to bank1
{0x22, 0x50},
{0x48, 0x50},
{0xA2, 0x40},
{0x7C, 0x01},
{0x4F, 0x07},
{0x3F, 0x04},
{0x0C, 0x05},
{0x4D, 0x05},
{0x52, 0x05},
{0x86, 0x50},
{0x92, 0x1C},
{0x98, 0x1D},
{0x9A, 0x42},
{0x81, 0x01},
{0x3B, 0x00},
{0xEA, 0xC9},
{0xA4, 0x50},
{0xA5, 0x00},
{0xA6, 0x52},
{0xA7, 0x00},
{0xA8, 0x53},
{0xA9, 0x00},
{0xD6, 0xA0},
```

```
{0xD7, 0x00},
{0xD8, 0x01},
{0xD9, 0x00},
{0xDA, 0x2C},
{0xDB, 0x00},
{0xDC, 0x8B},
{0xDD, 0x00},
{0xDE, 0x8C},
{0xDF, 0x00},
{0xE0, 0x9F},
{0xE1, 0x00},
};
#define INIT_STRESS_REG_ARRAY_SIZE (sizeof(init_stress_register_array)/sizeof(init_stress_register_array[0]))
```

6.6 Sleep Setting

```
const uint8_t suspend_register_array[][2] = {
{0x7f, 0x01},       //switch to bank1
{0x09, 0x01},
{0x23, 0x01},
{0xB4, 0x01},
{0xB7, 0x01},
{0xE6, 0xC8},
{0xE7, 0x00},
{0xF1, 0x00},
{0x07, 0x01},
{0xAE, 0x06},
{0xAF, 0x07},
{0xBA, 0x7C},
{0x6C, 0x10},
{0x6D, 0x10},
{0x7A, 0x00},
{0x6F, 0x10},
{0x7F, 0x00},       //switch to bank0
{0x08, 0xFF},
{0x09, 0x03},
{0xD6, 0x01},
{0x5C, 0x00},
{0x5D, 0x05},
{0x60, 0x00},
{0x61, 0x03},
{0x64, 0x05},
{0x65, 0x05},
{0x35, 0x80},
{0x36, 0x02},
{0x8C, 0x00},
{0x8E, 0x00},
{0xDE, 0x00},
{0xD9, 0x01},
{0xDD, 0x04},
{0x3B, 0x01},
{0x47, 0x01},
{0x48, 0x00},
```

```
{0x49, 0x00},
{0x4A, 0x00},
{0x4D, 0x00},
{0x16, 0x00},
{0x13, 0x01},
{0x14, 0x01},
{0x15, 0x01},
{0x50, 0x01},
{0x51, 0x01},
{0x59, 0x00},
{0x57, 0x00},
{0x6B, 0x00},
{0x6C, 0x00},
{0x3E, 0x00},
{0x43, 0x00},
{0x0D, 0x78},
{0x0E, 0x00},
{0x7F, 0x02},       //switch to bank2
{0x17, 0x00},
{0x18, 0x00},
{0x1B, 0x01},
{0x1C, 0x01},
{0x1F, 0x00},
{0x29, 0x00},
{0x2d, 0x01},
{0x2B, 0x00},
{0x2C, 0x00},
{0x31, 0x00},
{0x4F, 0x10},
{0x66, 0x01},
{0x67, 0x01},
{0x68, 0x01},
{0x69, 0x01},
{0x6A, 0x01},
{0x6B, 0x01},
{0x6C, 0x01},
{0x6D, 0x01},
{0x6E, 0x01},
{0x6F, 0x01},
{0x70, 0x01},
{0x74, 0x00},
{0x76, 0x01},
{0x78, 0x01},
{0x7A, 0x01},
{0x7B, 0xFF},
{0x8D, 0x01},
{0x8F, 0x01},
{0x92, 0x00},
{0x7F, 0x01},       //switch to bank1
{0xA2, 0x40},
{0x7C, 0x01},
{0x4C, 0x01},
{0x4F, 0x07},
```

```
{0x3F, 0x04},
{0x0C, 0x05},
{0x4D, 0x05},
{0x52, 0x05},
{0x86, 0x50},
{0x92, 0x1C},
{0x98, 0x1D},
{0x9A, 0x42},
{0x81, 0x01},
{0x3B, 0x00},
{0xEA, 0xC9},
{0xA4, 0x50},
{0xA5, 0x00},
{0xA6, 0x52},
{0xA7, 0x00},
{0xA8, 0x53},
{0xA9, 0x00},
{0xD6, 0xFF},
{0xD7, 0x1F},
{0xD8, 0x01},
{0xD9, 0x00},
{0xDA, 0x10},
{0xDB, 0x00},
{0xDC, 0x13},
{0xDD, 0x00},
{0xDE, 0x14},
{0xDF, 0x00},
{0xE0, 0xFE},
{0xE1, 0x1F},
{0x7F, 0x01},       //switch to bank1
{0xd5, 0x01},
};
#define SUSPEND_REG_ARRAY_SIZE (sizeof(suspend_register_array)/sizeof(suspend_register_array[0]))
```

Document Revision History

Revision Number	Date	Description
0.1	22 May 2015	1st Creation, Preliminary version
0.2	01 Jul 2015	1. Added Table of Contents 2. Updated Introduction 3. Added Heart rate FIFO control for PPG data and Accelerometer data asynchronous section 4. Added Reference code of MCU by using PXI HRD Library 5. Added Appendices setting 6. Updated PXI Algorithm API 7. Updated Data Format Description
0.3	01 Sep 2015	1. Update PPG setting of 20Hz and 200Hz and Touch Setting and Sleep Setting 2. Update Software References
0.4	15 Sep 2015	1. Add PPG setting of 20Hz Long Exposure Time and mode 2. Modify LED0 DAC value (Bank1 0xBA) 3. Update Software References 4. Modify PPG Mode Operation Flow
0.5	02 Oct 2015	1. Mask bank2 0x6B and 0x6C of Touch Setting and PPG Setting of 20Hz
0.6	26 Oct 2015	1. Update code of "void pah8002_task()"
0.7	09 Nov 2015	1. Add pah8002_comm_spi reference code
0.8	16 Dec 2015	1. Modify Touch setting (Bank1 0xAD) 2. Modify setting for current leakage for I2C (Bank2 0x17,0x18,0x1B,0x1C)
0.9	18 Feb 2016	1. Change data log position before pah8002_entrance 2. Update FW code for system memory saving when algorithm disable. 3. Add test pattern section for ALG V212 4. Add Pixart Algorithm API Note