THIS SPEC IS OBSOLETE

Spec No: 001-50956

Spec Title: INFRARED PROXIMITY DETECTION WITH PSOC(R) - AN50956

Sunset Owner: Rui (Rod) Wang (ROWA)

Replaced by: NONE

AN50956

Infrared Proximity Detection with PSoC®

Author: Jemmey Huang, Wayne Zhang
Associated Project: Yes
Associated Part Family: CY8C23X33, CY8C24X33
Software Version: PSoC Designer™ 5.0
Related Application Notes: None

If you have a question, or need help with this application note, contact the author at rjvb@cypress.com

OBSOLETE

Abstract

AN50956 describes the interface and signal processing for infrared proximity detection using PSoC®. An example project and a circuit are used to demonstrate the application.

Contents

Introduction 1
Infrared Proximity Sensor 1
- Problems in Infrared Proximity Detection 2
- Using PSoC Internal Band-Pass Filter and Low-Pass Filter 3
- Band-Pass Filter with Synchronous Sampling ADC 4

PSoC Project 5
- Hardware Block Diagram 5
- Device Configuration 6
- Application Firmware 7
- Testing of Example Project 7

Summary 8
Appendix A – Demonstration Board Schematic 9
Appendix B – Source Code 10

Introduction

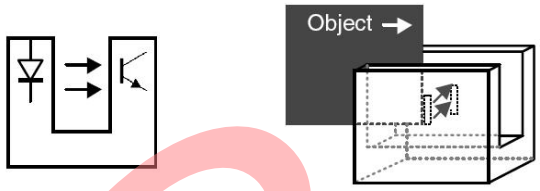
Infrared sensors are light sensors, which are active and function in the infrared part of the frequency spectrum. They primarily consist of an infrared emitter and an infrared receiver. They are found in many object detection applications such as automatic door openers and burglar alarms, surface feature detection, rotation shaft encoding, and barcode decoding. Some recently used applications in mobile devices include auto call-answering, auto speaker phone activation, and detection of flip-in-clamshell phones.

This application note is bundled with the firmware and hardware for demonstrating interface and analog signal processing of infrared proximity sensors using PSoC. It also explains infrared light sensing and the advantages of using PSoC in this solution.

Infrared Proximity Sensor

The infrared sensor is made up of the emitter (infrared LED) and detector (photodiode). The emitter emits IR light pulse and the receiver detects the corresponding light pulse. The sensor is classified into transmissive and reflective sensors. As shown in Figure 1, in a transmissive sensor, the emitter and the detector face each other. Objects are detected if they interrupt the beam of light between the emitter and the detector.

Figure 1. IR Transmissive Sensor Illustration



For a reflective sensor, the emitter and the detector are next to each other, and are separated by a barrier. Objects are detected when light is reflected off them and back into the detector. The reflective sensor can be used in proximity detection. Some vendors provide a built-in sensor that includes the receiver and transmitter in a small package. In this application note, Avago HDSL-9100 is used in the demonstration circuit.

Figure 2. IR Proximity Sensor Illustration

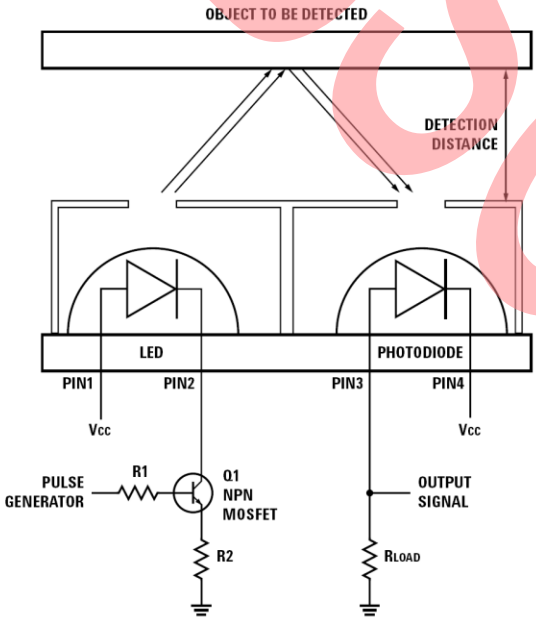


Figure 2 is the typical testing circuit for HDSL-9100, which is an analog-output reflective sensor. It produces a photocurrent that can be converted to an output voltage through an external resistor when an object is hit. The light intensity of the emitter depends on the V_{CC} to the LED (Pin 1) and the current limiting resistor (R2).

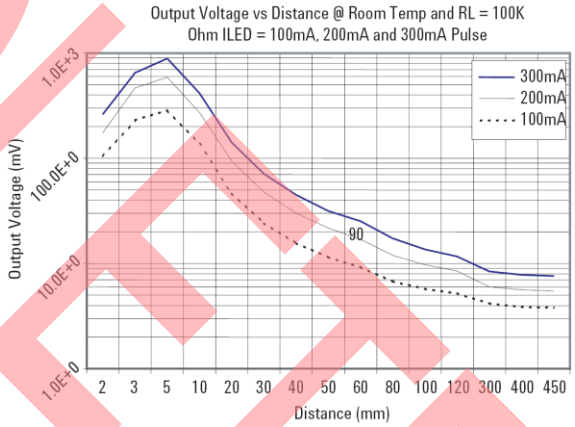
Typically, the driver from the microcontroller cannot provide a high current to drive the LED and requires a transistor (Q1) to drive the LED at a higher current (possibly higher than 100 mA peak). The infrared intensity of the emitter influences the detection range. Considering the critical requirement of the PCB size in mobile phone application, the demonstration circuit uses two GPIO pins to drive the LED in parallel and provide current to the emitter.

On the IR receiver side, the desired output voltage depends on the detection distance and the value of the load resistor (R_{LOAD}). This output voltage signal can be connected to the next stage, such as an analog amplifier, comparator, or Schmitt-Trigger, to control various functions.

The selection of the load resistor, R_{LOAD}, plays a significant role in circuit operation. If the R_{LOAD} is too large, the RC time constant increases, thereby increasing the response time. However, if the R_{LOAD} is too small, it contributes more thermal noise to the circuit than higher ones. Therefore, it is important to note the current-to-voltage transfer characteristics for this part of the circuit. According to the applications, the R_{LOAD} can be selected in a range of 50 kΩ to several MΩ. In this demonstration circuit, we use a 1 MΩ resistor for evaluation.

Figure 3 shows the relationship between the obstacle distance and HDSL-9100 receiver output voltage, under different energization currents on the HDSL-9100 emitter.

Figure 3. IR Proximity Output versus Distance



Problems in Infrared Proximity Detection

The following problems in infrared proximity detection applications need to be solved:

1. The influence from ambient light, such as sunlight or fluorescent light. Steady light falling directly on the detector reduces the sensor's sensitivity. Strong light can saturate the phototransistor and, in this condition, the sensor becomes blind. Varying ambient light results in incorrect signals and non-existent reflection changes. Therefore, the influence of ambient light must be minimized by using optical filters, inspired mechanical design, and, if necessary, using modulation operation. Figure 4 shows the output signal that the sensor is under fluorescent light. There is a distinct noise from the fluorescent, which is

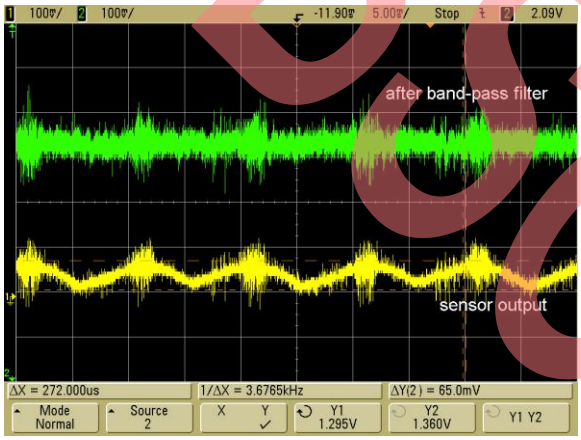
OBSOLETE

100 Hz frequency with about 70 mv peak-peak amplitude.

2. Interference from other infrared light sources, such as remote controllers, has the same influence as ambient light.
3. Immunity to the power supply system.
4. Trade-off between detection range and power consumption, especially in portable design.

One way to solve these problems is to emit modulated light, which quickly turns the emitter on and off. The modulated signal is more reliably detected by a demodulator, which is tuned to the frequency of the modulated light.

Figure 4. IR Proximity Output Under Fluorescent Light



With the modulated infrared light, PSoC uses several implementation methods to handle the signal. These include:

- Using external band-pass filter and low-pass filter with PSoC ADC to sample the filter output. This method uses external RC network to build the band-pass and low-pass filter and get the average. The disadvantages of this method are:
  - Needs external RC network; consumes more PCB space.
  - The passive filter does not have high-Q.
  - Too much delay if applied to a deep low-pass filter.
  - Implementation flexibility is not good; the modulated frequency decides the RC network value.
- Using PSoC internal band-pass filter and low-pass filter with ADC. This is discussed in the following section.

Using PSoC internal band-pass filter with synchronous sampling ADC, which is detailed in the section Band-Pass Filter with Synchronous Sampling ADC.

Using PSoC Internal Band-Pass Filter and Low-Pass Filter

The PSoC flexible on-chip resource provides different configurable settings that can be used in this application. The internal SC blocks and CT blocks can be configured as band-pass filter and low pass filter to process the modulated signal. The ADC can measure the signal output from the low-pass filter. Figure 5 shows the block diagram of the implementation.

In this diagram, a digital block is used to generate a 1.2 kHz PWM with 50 percent duty. The output drives the IR emitter through two GPIOs with 100-Ω serial resistors. The IR receiver output is connected to a PSoC internal band-pass filter, which is also configured to a 1.2-kHz centre frequency. After the band-pass filter, the signal is routed out through a pin back to the chip because the chip internal connection cannot support the signal routing directly to the low pass filter. A PGA is then used as a buffer to route the signal to a low-pass filter, in which corner frequency is set at about 500 Hz. The low-pass filter is used as an amplitude demodulator. Another CT is used as a comparator to detect the signal polarity from the output of the band-pass filter and to control the analog modulator in the SC block of the low-pass filter. For more information, see AN2042 - Sensing - Multifunctional Optical Sensor.

OBSOLETE

Figure 5. PSoC Design Block Diagram – Using Internal BPF and LPF

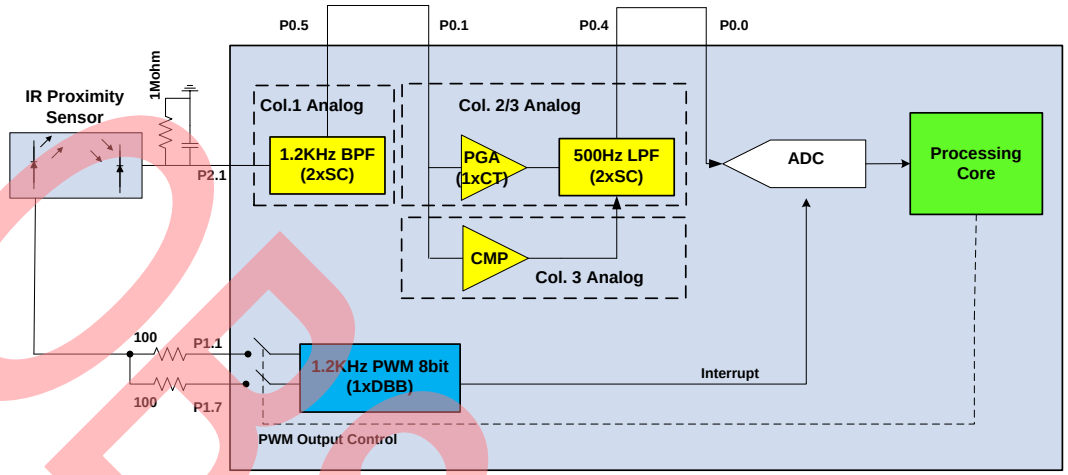
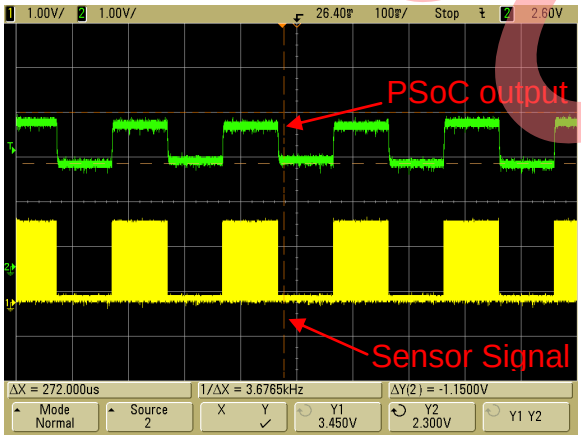


Figure 6 shows the analog implementation inside PSoC. It requires a four-column analog block device such as a diamond chip, and totally consumes two CT blocks and four SC blocks.

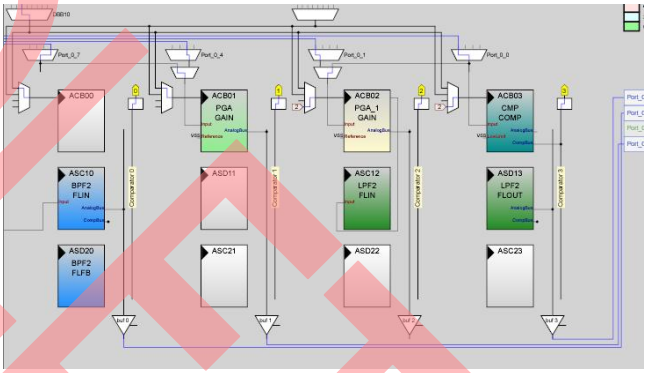
Figure 6. Waveform with BPF and LPF



The previous figure shows the signal output from the sensor and the signal output from PSoC after the processing of internal band-pass filter and low-pass filter. The PWM is applied to the emitter at an interval of 100 ms. The yellow waveform is the IR receiver output signal, which is modulated with the PWM signal. After the processing of PSoC BPF and LPF circuits, described earlier, the output signal is showed as the green waveform. It reflects the average amplitude of the input signal. So it is easy to use an ADC to sample the signal when PWM is applied to the emitter, or use a comparator to do an on and off detection.

The obvious disadvantage of this method is that it consumes too much PSoC resource, although its firmware is relatively simple. As shown in Figure 7, the configuration requires four SC blocks and three CT blocks.

Figure 7. PSoC Designer with BPF and LPF



Band-Pass Filter with Synchronous Sampling ADC

Another method for implementing the modulated signal processing in PSoC is to use a band-pass filter with a synchronous sampling ADC. The synchronous sampling ADC exists in some PSoC products such as CY8C23x33, CY8C24x33, and CY8C22x45. The dedicated SAR ADC in the PSoC is able to work with the PSoC digital block, so the ADC can be triggered at the point that aligned with the digital block counter's value. This function is useful in many applications similar to the peak current sampling in the power conversion application or BEMF signal sampling in BLDC motor control.

In the configuration, the ADC measures the peak value of the output of band-pass filter, and then the firmware calculates the amplitude of the sensor output. Figure 8 shows the signal of the sensor output and the PSoC PWM output applied on the IR emitter.

OBSOLETE

Figure 9 shows the signal after the PSoC band-pass filter. Only the modulated signal is allowed to pass the filter. Because the delay results from the capacitance load on the sensor output and the latency from the band-pass filter, the BPF output signal has some delay or phase shifting when compared with the PWM signal applied on the emitter. Different capacitance loads and different band-pass filters result in different phase delays. But, under the same circuit condition and input signal, the phase delay is the same. So it is possible to measure the peak value of the output signal by using the PWM signal to trigger an ADC at the same delay point. Figure 9 shows the relationship between the PWM signal and the filter output signal.

Figure 8. PWM and Sensor Output Waveform

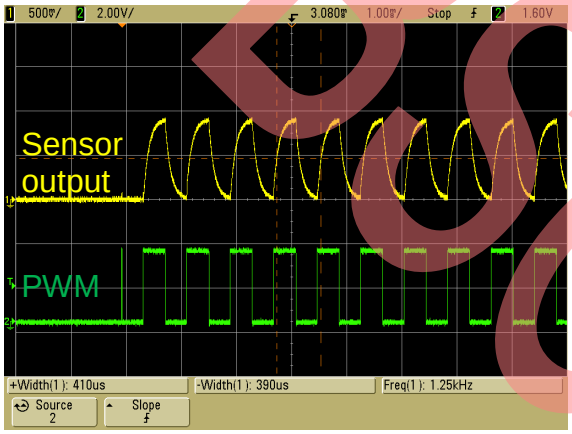
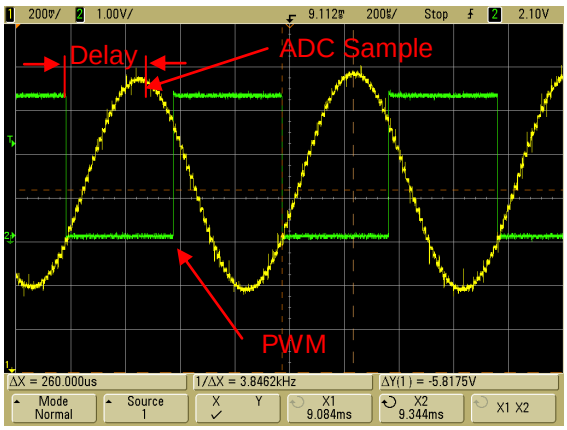


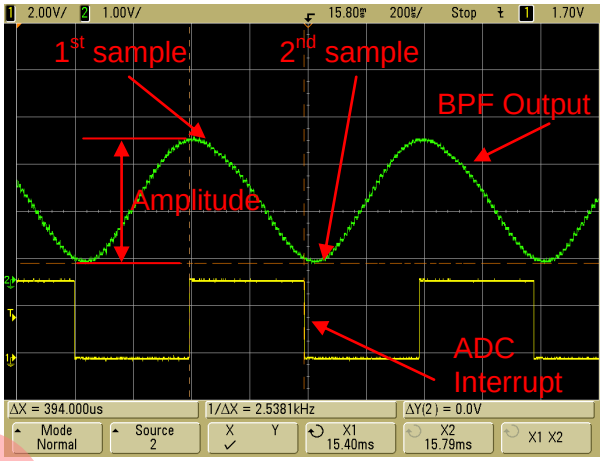
Figure 9. PWM and Signal Waveform After BPF



Because the BPF output signal is shifted to analog ground (AGND), the peak value of the BPF output is higher than AGND. For an 8-bit ADC, the ADC result of the peak signal is greater than 0x80.

It is possible to measure the waveform peak and valley values, and then work out the difference. The amplitude value of the signal should be twice as sensitive, when compared with only sampling the peak value. Based on this conception, we can use the ADC twice in a PWM period. Figure 10 shows the sampling point of the waveform. This still uses synchronous sampling ADC as mentioned before.

Figure 10. Using Double Sampling to Measure Amplitude



PSoC Project

This example project demonstrates how to interface and process the signals from IR proximity sensors with PSoC on-chip analog and digital blocks and the synchronous SAR ADC.

The hardware block diagram of the system is shown in Figure 11 and the PSoC internal configuration in PSoC Designer is shown in Figure 12. The example project performs the following tasks:

Input

- Infrared Proximity sensor.

Processing stage

- Driving the emitter of IR proximity sensor with PSoC PWM signal.
- Pre-processing IR proximity output signal by PSoC band-pass filter.
- Converting the band-pass filter output to digital through a synchronous sampling SAR ADC.
- Calculating the amplitude of the band-pass filter output signal and IR filtering in the firmware.
- Providing the amplitude data to host through I^2C interface.

Output

- Providing the amplitude data to host through I^2C interface.

Hardware Block Diagram

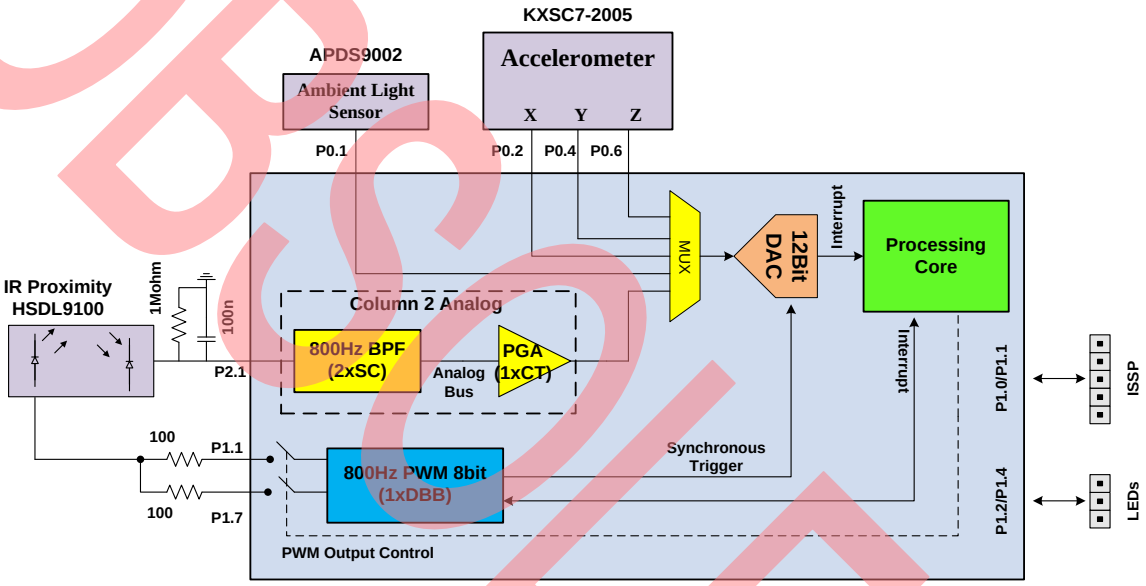
Figure 11 shows the hardware block diagram of the implementation. In this circuit, CY8C23533, which has a SAR8 ADC and can be triggered by the digital blocks, is used as the controller. The 800 Hz PWM signal drives the

OBSOLETE

LED emitter through the PSoC GPIO pin. It also triggers the SAR8 ADC at certain points in every PWM cycle to measure the peak and valley signal values of the BPF output. The triggering point of the ADC varies according to the load on the IR receiver, energization on emitter, and other testing conditions. With a fixed testing condition, the phase delay is almost the same. We can measure the delay manually to decide the ADC trigger point. The LED receiver is connected to PSoC P2.1 pin and routed to the band-pass filter directly; no amplifier is required. Use an internal CT as the buffer to route the BPF output signal to SAR8 ADC for measurement.

Other types of sensors are also included in the block diagram as this demonstration board is also used to demonstrate those sensors.

Figure 11. System Hardware Block Diagram – Band-pass Filter with Synchronous ADC



Device Configuration

The analog block layout of the PSoC project is shown in Figure 12. The PSoC project is built on the demonstration board using CY8C23533. For detailed schematic information, see Appendix A – Demonstration Board Schematic.

Figure 12. PSoC Designer Layout

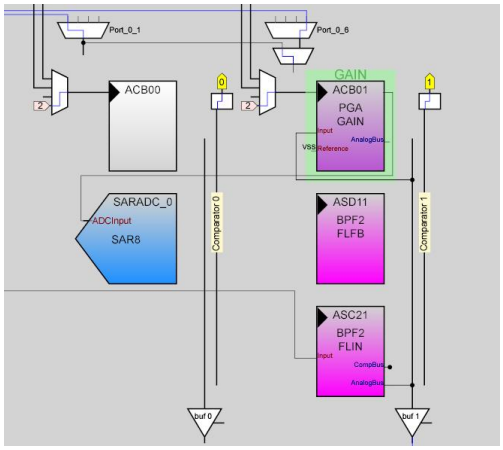


Table 1 shows the resource requirements for this application.

Table 1. Resource Requirements

User Module Requirement	PSoC Blocks Required for Each User Module	User Module Flash	User Module RAM
SAR ADC(8 bit)	Dedicated analog PSoC Block	147	0
PWM	1 Digital Block(DBB)	67	0
Band-Pass Filter	2 SC Block	109	0
PGA(optional)	1 CT Block	52	0
EzI2Cs	None	264	16

This application requires one dedicated analog PSoC block, one digital block, and two SC blocks. If you route the BPF output to ADC through an internal resource, then a CT block can be used as the buffer. Otherwise, route the signal out to an analog output buffer, and then route it to port 0 to feed into the ADC. Therefore, the remaining blocks provide additional functionality to the application.

OBSOLETE

Application Firmware

The demonstration board samples three types of sensors in the system and sends out the ADC result through I^2C interface.

CY8C23533 uses the SAR8 ADC to sample the sensors' output voltage. Among the three sensors, the ambient light sensor and accelerometer are detected directly without any pre-condition circuit before going into the ADC input. The IR proximity detection needs the band-pass filter for pre-conditioning.

The system has an 800 Hz PWM, which is used to drive the IR proximity sensor transmitter. Select the 800 Hz frequency as it is 16 times than that of the 50 Hz power grid frequency.

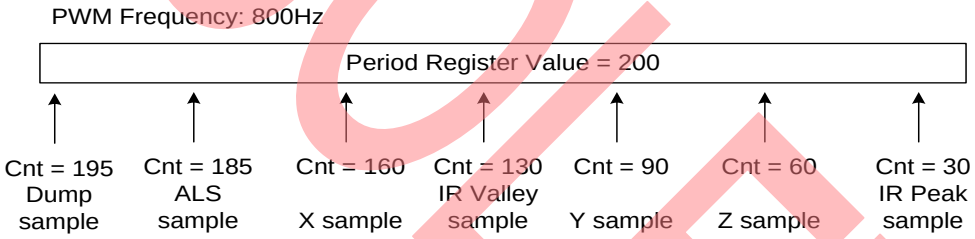
ADC samples the signals with PWM synchronization. This means the PWM triggers the ADC sample at a specific point and generates the ADC interrupt after completing conversion. The firmware responds to the ADC interrupt and sets up the next trigger point. The sample trigger points are arranged as shown in Figure 13. Remember that the PWM is a down-count register, so the firmware sets the maximum comparison value into the compare register.

Firmware samples 16 times from different sensor inputs, and accumulates them. This 16-time sampling period is about 20 ms, which is the power line cycle. After finishing the 16-time accumulation, the firmware does an IIR filter with previous measurement results, and updates the data to the I2C buffer.

When the band-pass filter starts, there is a delay before the filter output reaches the expected output value. Every time the measurement starts, there is a 4-PWM period delay to start the ADC sampling and result accumulation.

All the testing results, except the IR proximity detection, are reported in raw data mode. These are the ADC results after simple IIR filter. For the IR proximity detection, the result comes from the difference between the peak value and the valley value. The code is written in C. For detailed information, see Appendix B – Source Code.

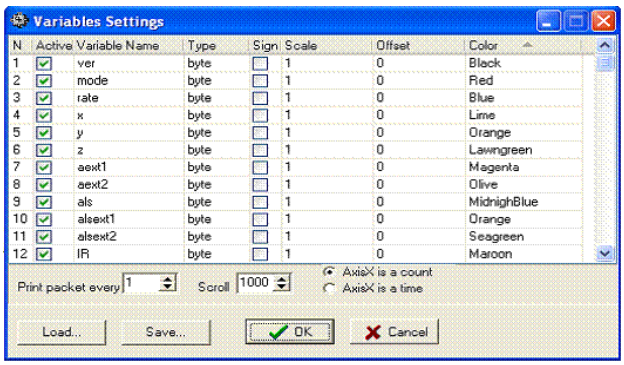
Figure 13. ADC Synchronous Trigger Point with PWM Counter



Testing of Example Project

- Hook up the sensor board with an USBtoIIC bridge.
- Open Cypress USBtoIIC GUI.
- Set up the variables as shown in Figure 14.

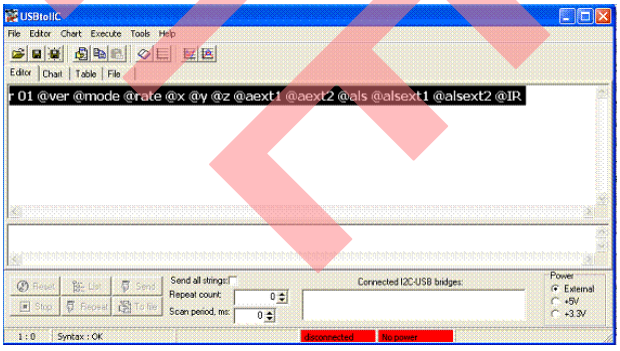
  Figure 14. USBtoIIC Variables Setting



- Set up the IIC speed to 100 Ksps or 50 Ksps

  Input the command as: r 01 @ver @mode @rate @x @y @z @aext1 @aext2 @als @alsext1 @alsext2 @IR.

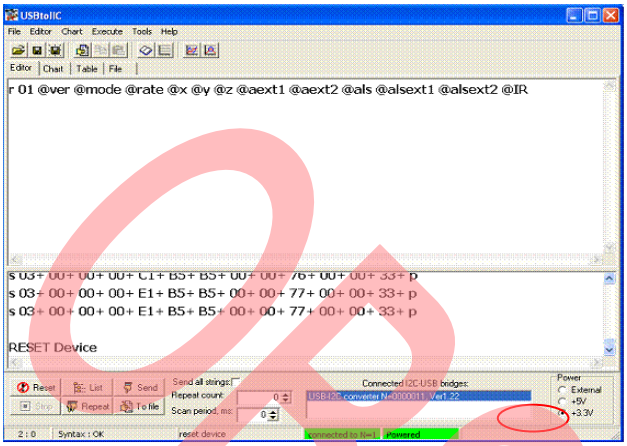
  Figure 15. USBtoIIC Command Setting



- Provide 3.3 V power supply to the board.

OBSOLETE

Figure 16. USBtoIIC - Providing 3.3 V Power Supply



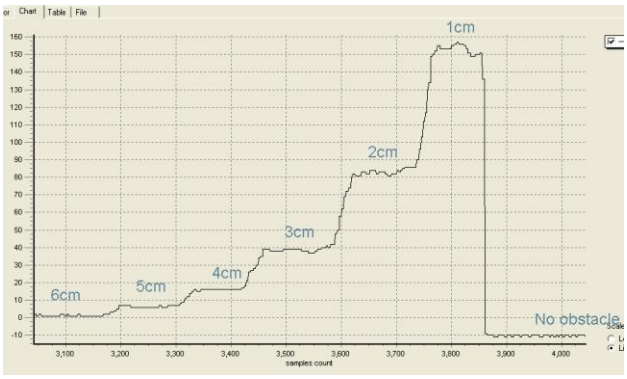
- Click “repeat” and click the “chart” tab. The next window appears. Check the X, Y, Z, ALS, and IR on the right window. The IR signal represents the IR proximity amplitude value measured by PSoC.

  Figure 17. IR Proximity Data Showing on USBtoIIC GUI



- Moving the finger close to the IR sensor changes the corresponding curve. Figure 18 shows the obstacle at a different distance above the sensor. This experiment uses a Cypress business card as the obstacle with the test condition: PWM duty – 50 percent, PWM frequency – 800 Hz, and the external components list as shown in Figure 11.

  Figure 18. Testing with a Different Distance



Summary

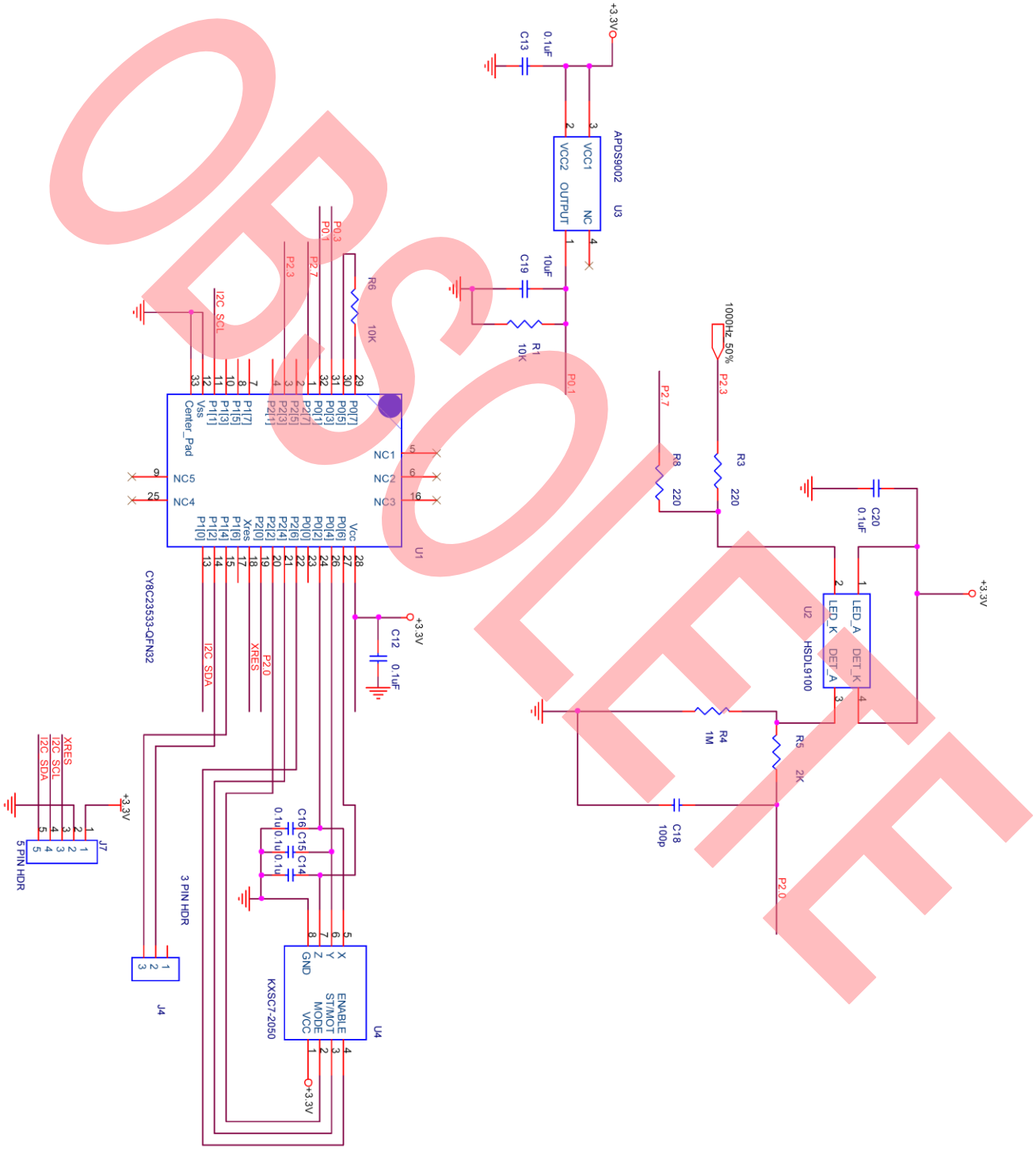
This application note describes the interface and signal processing for infrared proximity detection using PSoC®. Modulated infrared proximity detection can be implemented with the CY8C23x33 device. Its two SC blocks can be configured as band-pass filters, and the SAR8 ADC can be triggered synchronously to the internal PWM used to drive the LED emitter. The system only consumes two SC blocks and one ADC with very simple firmware processing.

About the Author

Name:	Jemmey Huang and Wayne Zhang
Title:	Application Engineer Manager Sr. and Applications Engineer Staff.
Background:	Jemmey Huang is working as Sr. Applications Engineer Manager in Cypress Semiconductor’s Asia-Pac Solution Center.
Contact:	jhu@cypress.com wzb@cypress.com

OBSOLETE

Appendix A – Demonstration Board Schematic



Appendix B – Source Code

Main.C

```
#pragma interrupt_handler ADC_Int_C();
#pragma interrupt_handler PWM_ISR_C();

void Enable_Accl_Sensor();
char scale(unsigned char data);

#define TRG_1  195
#define TRG_2  185
#define TRG_3  160
#define TRG_4  130
#define TRG_5  90
#define TRG_6  60
#define TRG_7  30

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include "i2c.h"
#include "pwm8.h"
#include "BPF2.h"

unsigned char bPort2_Data = 0;
unsigned char bPort1_Data = 0;
unsigned char bADC_cnt = 0;
unsigned char bPWM_cnt = 0;
unsigned char bFlag = 0;
unsigned char bADC_channel = 0;

int result, iResult_H, iResult_L, iResult_ALS, iResult_X, iResult_Y, iResult_Z;

struct FLAG {
unsigned PWM: 1;
unsigned ADC: 1;
unsigned others: 6;
}bIntFlags;


struct SNSData {
unsigned char bX;
unsigned char bY;
unsigned char bZ;
unsigned char bALS;
unsigned char bIR;
} SnsData;


void main()
{
    INT i,j;
    BYTE bResult;
    DWORD dwTemp1;
    INT iTemp;

    vI2CInit();
    I2CSBuff.bMode = 0;
```

OBSOLETE

```
M8C_EnableGInt;
PRT1DR = 0x88;
   PGA_Start(PGA_HIGHPOWER);
PWM8_WritePulseWidth(120);

   PWM8_EnableInt();
PWM8_Start();
PWM8_1_Start();
PWM8_2_Start();

BPF2_Start(BPF2_HIGHPOWER);
SAR8_SelectADCChannel(SAR8_ACB01);
SAR8_CONTROL_1_REG = 3; //In auto-triger mode, Low path active
SAR8_TRIGGER_SRC_REG = 1; //set DBB0 as the ADC triger source
SAR8_COMPARE_LO_REG = TRG_6;
SAR8_EnableInt();
SAR8_Start();

while(1)
{

    if(bIntFlags.PWM==1) //800HZ PWM interrupt, 1.25ms interval
    {
        bIntFlags.PWM=0;
        bPWM_cnt++;

        if(bPWM_cnt==4)//start the ADC after 4 PWM pulse, wait until    signal stable
        {
            SAR8_SelectADCChannel(SAR8_ACB01);      //first channel
            SAR8_COMPARE_LO_REG = TRG_1;      //first trigger point
            bADC_channel = 0;
            bADC_cnt = 0;

            iResult_H = 0;
            iResult_L = 0;
            iResult_ALS = 0;
            iResult_X = 0;
            iResult_Y = 0;
            iResult_Z = 0;

            SAR8_Start();
        }
        else if(bPWM_cnt==20) //after 20 pulse count, about 25ms,
        {               //output PWM control - disable
            if(I2CSBuff.bMode!=0)      //full run mode, every 25ms update 1 time
                PRT2GS = PRT2GS & 0x77;
            SAR8_Stop();
        }
        else if(bPWM_cnt>24)
        {
            if(((I2CSBuff.bMode==1)&&(bPWM_cnt==40))||((I2CSBuff.bMode==2)&&(bPWM_cnt==80)))
            {
                PRT2GS = PRT2GS | 0x88;       //switch the PWM output to pin
            }
        }

    } // end of if(bIntFlags.PWM==1) //800HZ PWM interrupt, 1.25ms interval

    if(bIntFlags.ADC==1) // ADC interrupt
    {
```

OBSOLETE

```
    switch (bADC_channel)
    {
        case 0: // dump sample
                    SAR8_COMPARE_LO_REG = TRG_2;
                    //DUMP SAMPLE to remove the channel influence
                    SAR8_SelectADCChannel(SAR8_P0_1);
                    bADC_channel = 1;
                    break;
        case 1: // sample ALS
                    SAR8_COMPARE_LO_REG = TRG_3;
                    iResult_ALS += SAR8_DATA_LO_REG;

                    SAR8_SelectADCChannel(SAR8_P0_2);//next channel IR valley

                    bADC_channel = 2;
                    break;
        case 2: // sample accelerometer sensor - X
                    SAR8_COMPARE_LO_REG = TRG_4;
                    iResult_X += SAR8_DATA_LO_REG;
        SAR8_SelectADCChannel(SAR8_ACB01);//next channel IR valley
                    bADC_channel = 3;
                    break;
        case 3: // sample valley value of IR proximity sensor

                    SAR8_COMPARE_LO_REG = TRG_5;
                    iResult_L += SAR8_DATA_LO_REG;
                    SAR8_SelectADCChannel(SAR8_P0_4);//next channel is Y
                    bADC_channel = 4;
                    break;

        case 4: // sample accelerometer sensor - Y
                    SAR8_COMPARE_LO_REG = TRG_6;
                    iResult_Y += SAR8_DATA_LO_REG;
                    SAR8_SelectADCChannel(SAR8_P0_6);//next channel is Z
                    bADC_channel = 5;
                    break;

        case 5: // sample accelerometer sensor - Z
                    SAR8_COMPARE_LO_REG = TRG_7;
                    iResult_Z += SAR8_DATA_LO_REG;
                    SAR8_SelectADCChannel(SAR8_ACB01);//next channel is IR peak
                    bADC_channel = 6;
                    I2CSBuff.bAcclExt1 = SAR8_DATA_LO_REG;
                    break;


        case 6: // sample peak value of IR proximity sensor
                    SAR8_COMPARE_LO_REG = TRG_1;
                    iResult_H += SAR8_DATA_LO_REG;
                    SAR8_SelectADCChannel(SAR8_P0_1);//next channel is ALS
                    bADC_channel = 0; // ----
                    bADC_cnt++;

                    break;
    }

if(bADC_cnt==16)
{
    if(iResult_H>iResult_L)
        result = ((iResult_H-iResult_L)>>4);//+(SnsData.bIR>>1);
    else result = 0;
```

OBSOLETE

```
            SnsData.bIR =  (SnsData.bIR>>1) + (result>>1); //Filter
            I2CSBuff.bProxiData = SnsData.bIR;
            I2CSBuff.bProxiExt1 = iResult_H>>4;
            I2CSBuff.bProxiExt2 = iResult_L>>4;
            iResult_H = 0;
            iResult_L = 0;
            PWM8_1_Stop();
            PWM8_1_WritePulseWidth(255-(SnsData.bIR>>1));
            PWM8_1_Start();

            result = (SnsData.bALS<<1) + SnsData.bALS;
            result = result+(iResult_ALS>>4);
            SnsData.bALS = result>>2;
            I2CSBuff.bAlsData=SnsData.bALS;
            iResult_ALS = 0;
            PWM8_2_Stop();
            PWM8_2_WritePulseWidth(255-SnsData.bALS);
            PWM8_2_Start();

            result = (SnsData.bX>>1) + (iResult_X>>5);
            SnsData.bX = result;
            I2CSBuff.bX=SnsData.bX;
            iResult_X = 0;

            result = (SnsData.bY>>1) + (iResult_Y>>5);
            SnsData.bY = result;
            I2CSBuff.bY=SnsData.bY;
            iResult_Y = 0;

            result = (SnsData.bZ>>1) + (iResult_Z>>5);
            SnsData.bZ = result;
            I2CSBuff.bZ=SnsData.bZ;
            iResult_Z = 0;

            bPWM_cnt=0;
            bADC_cnt = 0;
        }

        bIntFlags.ADC = 0;

    }//end of if(bIntFlags.ADC==1) // ADC interrupt

  }

}

void ADC_Int_C()
{
    bIntFlags.ADC = 1;
}


void PWM_ISR_C()
{
    bIntFlags.PWM = 1;
}
```

OBSOLETE

I2C.H

```
#ifndef __I2C_H__
#define __I2C_H__

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include "ezi2cs.h"

typedef struct
{
    BYTE bVersion;
    BYTE bMode;
    BYTE bUpdateRate;
    //ACCELEROMETER CONTROL PARAMETER
    BYTE bX;
    BYTE bY;
    BYTE bZ;
    BYTE bAcclExt1;
    BYTE bAcclExt2;
    //ALS CONTROL PARAMETER
    BYTE bAlsData;
    BYTE bAlsExt1;
    BYTE bAlsExt2;
    //PROXIMITY CONTROL PARA
    BYTE bProxiData;
    WORD bProxiExt1;
    WORD bProxiExt2;

    BYTE bExtFun1;
    BYTE bExtFun2;

}TYPE_I2CS_STRUCT;

extern TYPE_I2CS_STRUCT I2CSBuff;
//extern BYTE bgControl1,bgControl2;
void vI2CInit(void);
//void vScanI2CCmd(void);

#endif
```

OBSOLETE

Document History

Document Title: Infrared Proximity Detection with PSoC® - AN50956

Document Number: 001-50956

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2635179	JHU	01/13/2009	New Application Note
*A	3569829	RJVB	04/02/2012	Updated template. Removed reference to AN2044 as the document is obsolete. No technical updates. Completing sunset review.
*B	4728902	ROWA	05/07/2015	Obsolete Spec.

OBSOLETE

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

OBSOLETE

PSoC is a registered trademark of Cypress Semiconductor Corp. "Programmable System-on-Chip" and PSoC Designer are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2009-2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.