

Sonic Alarm

By: Chris and Vincent Paiano

Associated Project: Yes

Associated Part Family: CY8C25xxx, CY8C26xxx

PSoC Designer Version: 4.1

Abstract

The PSoC is configured to listen to a condenser microphone through two stages of PGA gain, a BPF2 and a comparator (in that order), to provide an activity-detection scheme based on audio. Audible warnings and an alarm are sounded after an entrance and exit delay, to convince any unwanted visitors to vacate the premises.

Introduction

This simple PSoC alarm provides inexpensive and effective security with minimal installation effort – simply hook up the audio output to your existing stereo system, turn it up, and hide the touch switch.

Light, unobtrusive warning sounds will be given through an amplifier and speaker to provide time for either the occupants to leave after setting the alarm, or for someone to disarm the alarm after re-entry before the alarm goes nuts. If nobody disarms the device, a very loud and annoying siren/klaxon sound will be produced to further defer unwanted visitors.

Software/PSoC Implementation

The PSoC implementation of the Sonic Alarm utilizes an analog pathway for ambient room audio to enter through a condenser microphone, which uses a PGA to pass out AGND.

The pathway begins with two stages of PGA gain, which feed a band-pass filter (configured with the Filter Design Wizard, a handy tool found by right-clicking on the filter module once placed). This filter (BPF2) is set for a center frequency of 500 Hz, a bandwidth of 100 Hz, and a gain of 18 dB. The output of the BPF2 effectively singles out the noise associated with general activity from other sounds such as trucks passing by, and is rectified externally before being input to a comparator to detect this activity. A PWM is used for the warning tones, and another is used for the alarm tones. These two are mixed externally to the same output, with the warning sounds attenuated significantly. This allows the user to really crank the stereo system up to full volume for the alarm to cause a ruckus, while the warning chirps will be background noise.

The PSoC code, illustrated in Figure 2, demonstrates how all this might be achieved in practice. A flowchart of the code is shown in Figure 3, and the configuration of PSoC resources appear in Figure 4, Figure 5, Figure 6, and Figure 7.

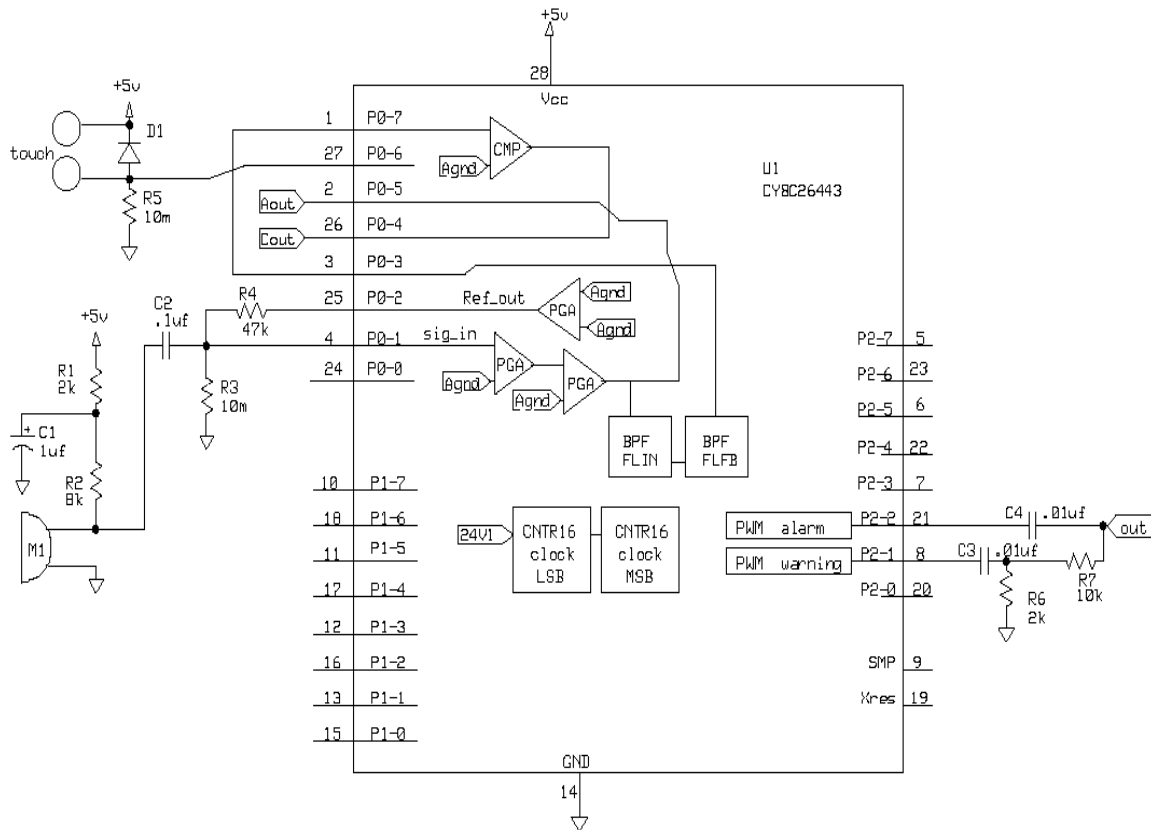


Figure 1. Schematic

Hardware Implementation

The condenser, mic M₁, is phantom powered through R₂ from the +5V supply, filtered by R₁/C₁. C₂ provides DC blocking, allowing the input P0-1 to float a little below AGND, set by R₃ and R₄. This offset was found to be necessary when gain is set at a maximum on both PGAs. C₃, C₄, R₅, and R₆ mix the output so that the warning signal is much lower in volume than the alarm.

```

#include <m8c.h>          //Part-specific constants and macros
#include "PSoCAPI.h"     //PSoC API definitions for all User Modules
#include "globalparams.h"
#include "ports.h"
#define ArmTouchBitMask 64    //P0_6
#define NumSecondsWarning 30
#define KlaxonMinPeriod 3
#define KlaxonMaxPeriod 45
#define KlaxonToneChangeDelay 3000
#define NumKlaxons 1000
#define WarningToneTime 2500
#define ArmedLED(b) (PRT2DR = (b==0) ? (PRT2DR&0xF7) : (PRT2DR|0x08))    //P2_3
#define BIT(bitNumber) ( 1 << (bitNumber) )
const char WarnChirpPeriods[]={16,4,24,9,16};
BOOL Activity=0;
BOOL Armed=0;
char TonePeriod;
char Warning;
unsigned int Alarm;
void ArmSystem(void);
void WarningChirp(void);
void SoundKlaxon(void);
void PollArmTouch(void);
    #define ProcessorSpeedInMHz 24
    #define WaitMSCorrectionFactor 5*ProcessorSpeedInMHz //to calibrate Wait function
for MS
    #define KlaxonBreakInMS 350
    #define KlaxonBreak KlaxonBreakInMS*WaitMSCorrectionFactor
    #define ArmTouchDelayInMS 1000
    #define ArmTouchDelay ArmTouchDelayInMS*WaitMSCorrectionFactor
    #define OneSecondInMS 1000
    #define OneSecond OneSecondInMS*WaitMSCorrectionFactor
    long int Counter = 0;
    void Wait(long int);
void main()
{
    CNT16_Clock_Start();
    PGA_Ref_Start(PGA_Ref_HIGHPOWER);
    BPF_Audio_Start(BPF_Audio_HIGHPOWER);
    PGA_Audio_Start(PGA_Audio_HIGHPOWER);
    PGA_Audio2_Start(PGA_Audio2_HIGHPOWER);
    CMP_Alarm_Start(CMP_Alarm_HIGHPOWER);
    PWM_Alarm_DisableInt();
    M8C_EnableIntMask(INT_MSK0, INT_MSK0_ACOLUMN_2);
    M8C_EnableGInt;
    while(1)
    {
        PollArmTouch();
        if (Armed)
        {
            Activity=0;
            while(!Activity)    //Check for activity
            {
                PollArmTouch();    //Check for disarm/arm
                if (!Armed) Activity=1;
            }
            if (Armed)    //Warning
            {
                for (Warning=0; Warning<NumSecondsWarning; Warning++)
                {
                    WarningChirp();
                    PollArmTouch();
                    if (!Armed) Warning=NumSecondsWarning;
                    Wait(OneSecond);
                }
            }
        }
    }
}

```

```

        if (Armed)    //Sound alarm!
        {
            for (Alarm=0; Alarm<NumKlaxons; Alarm++)
            {
                SoundKlaxon();
                PollArmTouch();
                if (!Armed) Alarm=NumKlaxons;
            }
        }
    }
}
void WarningChirp()
{
    PWM_Warn_WritePeriod(WarnChirpPeriods[0]);
    PWM_Warn_WritePulseWidth(WarnChirpPeriods[0]/2);
    PWM_Warn_Start();
    Wait(WarningToneTime);
    PWM_Warn_WritePeriod(WarnChirpPeriods[1]);
    PWM_Warn_WritePulseWidth(WarnChirpPeriods[1]/2);
    Wait(WarningToneTime);
    PWM_Warn_WritePeriod(WarnChirpPeriods[2]);
    PWM_Warn_WritePulseWidth(WarnChirpPeriods[2]/2);
    Wait(WarningToneTime);
    PWM_Warn_WritePeriod(WarnChirpPeriods[3]);
    PWM_Warn_WritePulseWidth(WarnChirpPeriods[3]/2);
    Wait(WarningToneTime);
    PWM_Warn_WritePeriod(WarnChirpPeriods[4]);
    PWM_Warn_WritePulseWidth(WarnChirpPeriods[4]/2);
    Wait(WarningToneTime);
    PWM_Warn_Stop();
}
void SoundKlaxon()
{
    TonePeriod=KlaxonMaxPeriod;
    PWM_Alarm_WritePeriod(TonePeriod);
    PWM_Alarm_WritePulseWidth(TonePeriod/2);
    PWM_Alarm_Start();
    Wait(KlaxonToneChangeDelay);
    while(TonePeriod>KlaxonMinPeriod)
    {
        PWM_Alarm_WritePeriod(--TonePeriod);
        PWM_Alarm_WritePulseWidth(TonePeriod/2);
        Wait(KlaxonToneChangeDelay);
    }
    while(TonePeriod<KlaxonMaxPeriod)
    {
        PWM_Alarm_WritePeriod(++TonePeriod);
        PWM_Alarm_WritePulseWidth(TonePeriod/2);
        Wait(KlaxonToneChangeDelay);
    }
    PWM_Alarm_Stop();
}
void ArmSystem()
{
    for (Warning=0; Warning<NumSecondsWarning; Warning++)
    {
        PollArmTouch();
        if (!Armed) Warning=NumSecondsWarning;
        WarningChirp();
        Wait(OneSecond);
    }
}
void PollArmTouch()
{
    if (PRTODR & ArmTouchBitMask)
    {
        Wait(ArmTouchDelay);
        if (PRTODR & ArmTouchBitMask)    //If still, touched. Toggle armed
state
        {

```

```

        if (Armed) { Armed=0; ArmedLED(0); }
        else { Armed=1; ArmedLED(1); ArmSystem(); }
    }
}
#pragma interrupt_handler CMP_Alarm_INT
void CMP_Alarm_INT()
{
    Activity=1;
}
void Wait(long int ToWait){ for (Counter = 0; Counter < ToWait; Counter++); }
    
```

Figure 2. PSoC C Code

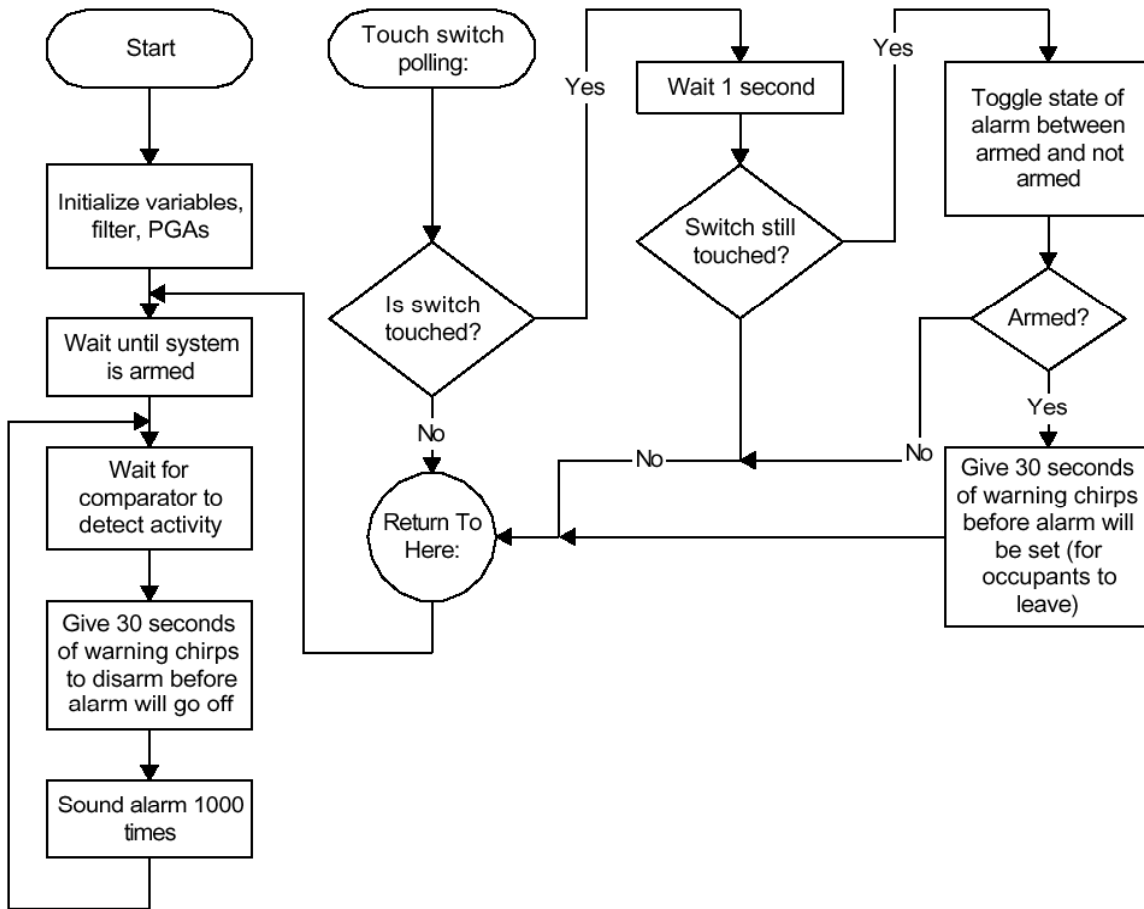


Figure 3. Program Flowchart

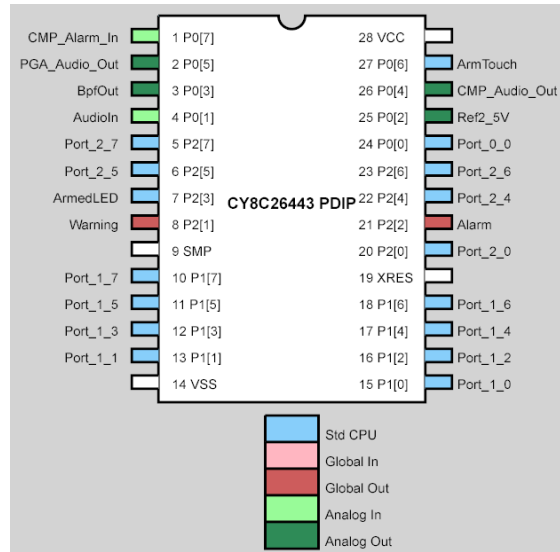


Figure 4. PSoC Pin Configuration

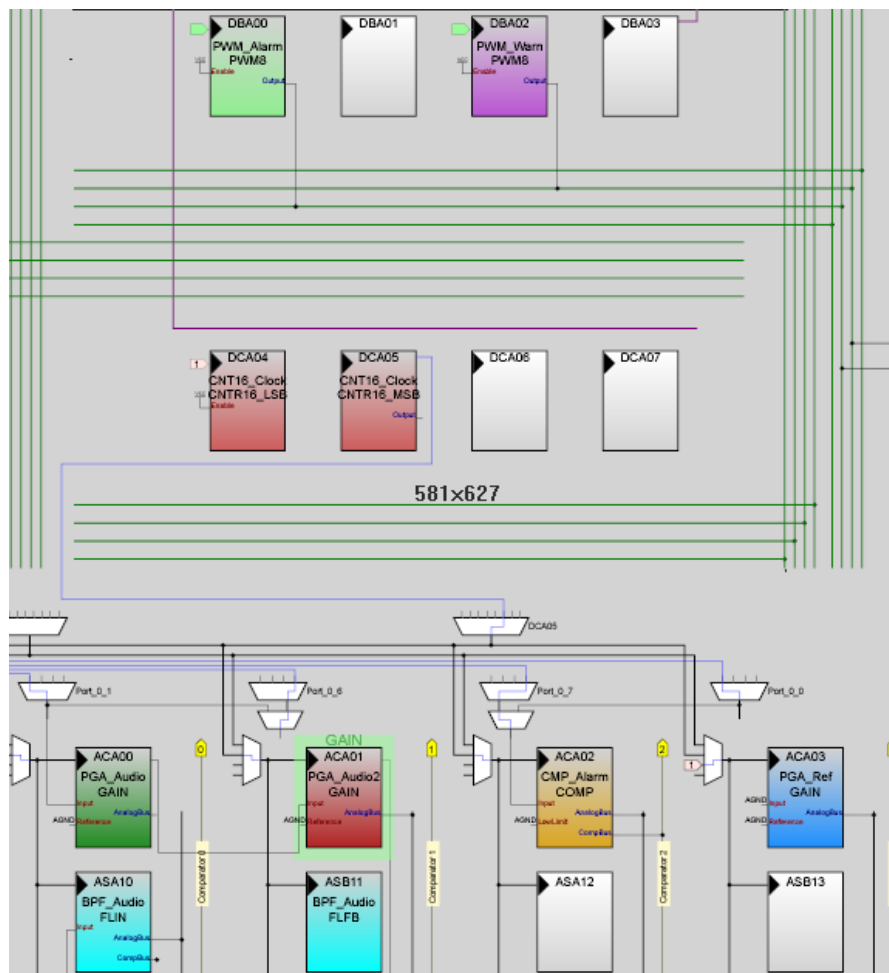


Figure 5. PSoC User Module Configuration

BPF_Audio	
User Module Parameters	
C1	13
C2	9
C3	9
C4	6
CA	32
CB	32
Input	ACA01
AnalogBus	AnalogOutBus_0
CompBus	DISABLE
Polarity	Non-Inverting

CMP_Alarm	
User Module Parameters	
AnalogBus	AnalogOutBus_2
CompBus	ComparatorBus_2
Input	AnalogColumn_InputSelect_2
LowLimit	AGND
RefValue	0.625

CNT16_Clock	
User Module Parameters	
Clock	24V1
Enable	High
Output	None
Period	549
CompareValue	275
CompareType	Less Than Or Equal
InterruptType	Terminal Count

PGA_Audio	
User Module Parameters	
Gain	16.00
Input	AnalogColumn_InputMUX_0
Reference	AGND
AnalogBus	Disable

PGA_Audio2	
User Module Parameters	
Gain	16.00
Input	ACA00
Reference	AGND
AnalogBus	AnalogOutBus_1

PGA_Ref	
User Module Parameters	
Gain	1.000
Input	AGND
Reference	AGND
AnalogBus	AnalogOutBus_3

PWM_Alarm	
User Module Parameters	
Clock	CPU_32_KHz
Enable	High
Output	Global_OUT_2
Period	0
PulseWidth	0
CompareType	Less Than Or Equal
InterruptType	Terminal Count

PWM_Warn	
User Module Parameters	
Clock	CPU_32_KHz
Enable	High
Output	Global_OUT_1
Period	0
PulseWidth	0
CompareType	Less Than Or Equal
InterruptType	Terminal Count

Figure 6. PSoC User Module Parameter Settings

Global Resources		Name	Port	Select	Drive	Interrupt
CPU_Clock	24_MHz	Port_0_0	P0[0]	StdCPU	Pull Down	DisableInt
32K_Select	Internal	AudioIn	P0[1]	AnalogInp	High Z	DisableInt
PLL_Mode	Disable	Ref2_5V	P0[2]	AnalogOu	High Z	DisableInt
Sleep_Timer	512_Hz	BpfOut	P0[3]	AnalogOu	High Z	DisableInt
24V1= 24MHz/N	1	CMP_Audio_Ou	P0[4]	AnalogOu	High Z	DisableInt
24V2= 24V1/N	1	PGA_Audio_Ou	P0[5]	AnalogOu	High Z	DisableInt
Analog Power	SC On/Ref Med	ArmTouch	P0[6]	StdCPU	High Z	DisableInt
Ref Mux	(Vcc/2)+/(Vcc/2)	CMP_Alarm_In	P0[7]	AnalogInp	High Z	DisableInt
Op-Amp Bias	Low	Port_1_0	P1[0]	StdCPU	Pull Down	DisableInt
A_Buff_Power	Low	Port_1_1	P1[1]	StdCPU	Pull Down	DisableInt
SwitchModePump	OFF	Port_1_2	P1[2]	StdCPU	Pull Down	DisableInt
Trip Voltage [LVD (SMP)]	4.64V (5.00V)	Port_1_3	P1[3]	StdCPU	Pull Down	DisableInt
Supply Voltage	5.0V	Port_1_4	P1[4]	StdCPU	Pull Down	DisableInt
Watchdog Enable	Disable	Port_1_5	P1[5]	StdCPU	Pull Down	DisableInt
		Port_1_6	P1[6]	StdCPU	Pull Down	DisableInt
		Port_1_7	P1[7]	StdCPU	Pull Down	DisableInt
		Port_2_0	P2[0]	StdCPU	Pull Down	DisableInt
		Warning	P2[1]	Global_OU	Strong	DisableInt
		Alarm	P2[2]	Global_OU	Strong	DisableInt
		ArmedLED	P2[3]	StdCPU	Pull Up	DisableInt

Figure 7. PSoC Global Resource Settings

About the Authors

Name: Chris and Vincent Paiano
Title: B.S., Computer Engineer and Electronic Engineer

Background: 22+ years programming/computer experience.
 40+ years electronics/design and troubleshooting experience.

Contact: psoc@chrispaiano.com
engineering@chrispaiano.com

Cypress MicroSystems, Inc.
 2700 162nd Street SW, Building D
 Lynnwood, WA 98037
 Phone: 800.669.0557
 Fax: 425.787.4641

<http://www.cypress.com/> / <http://www.cypress.com/support/mysupport.cfm>

Copyright © 2004 Cypress MicroSystems, Inc. All rights reserved.

PSoC™, Programmable System-on-Chip™, and PSoC Designer™ are trademarks of Cypress MicroSystems, Inc. All other trademarks or registered trademarks referenced herein are the property of their respective owners. The information contained herein is subject to change without notice. Made in the U.S.A.