CY3280-CPM1 CapSense® Plus Module Development Kit Guide

Doc. # 001-51922 Rev. *B

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Copyrights

Flash Code Protection

Cypress products meet the specifications contained in their particular Cypress PSoC Data Sheets. Cypress believes that its family of PSoC products is one of the most secure families of its kind on the market today, regardless of how they are used. There may be methods, unknown to Cypress, that can breach the code protection features. Any of these methods, to our knowledge, would be dishonest and possibly illegal. Neither Cypress nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Cypress is willing to work with the customer who is concerned about the integrity of their code. Code protection is constantly evolving. We at Cypress are committed to continuously improving the code protection features of our products.

Contents

1. Introduction

Thank you for your interest in the CY3280-CPM1 CapSense® Plus Module Development Kit. CapSense Plus is defined as capacitive sensing plus other value-added features, such as LED color mixing, fan control, motor control, and temperature sensing. Each feature that the PSoC® can integrate in addition to CapSense is defined as a "Plus" feature.

This kit showcases the CapSense Plus features provided by CY8C22x45 and CY8C28xxx. The CapSense Plus Module (CPM) is a daughter board of CY3280-22x45 and CY3280-28xxx Universal CapSense Controller development kits. This document demonstrates some examples that highlight the new features of CY8C22x45 and CY8C28xxx. These features include:

- Real-time clock (RTC)
- 10-bit successive approximation register (SAR) ADC
- Variable length serial peripheral interface (SPI)
- Pulsewidth modulator (PWM)

This kit also serves as a development platform; you can reuse the source code for your CapSense Plus applications.

1.1 Kit Contents

The following items are included with the kit:

- CY3280-CPM1 CapSense Plus Module Board
- CY3280-CPM1 CapSense Plus Kit CD
- Quick Start Guide

1.2 Directory Structure

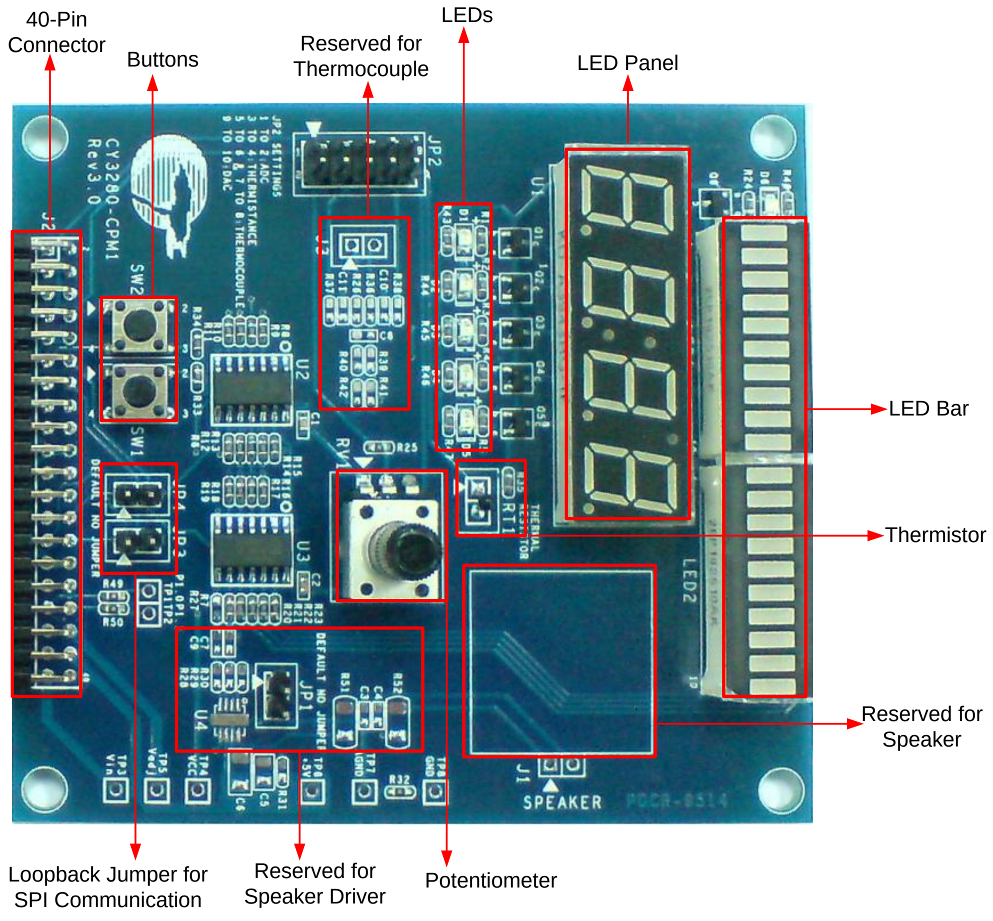
The following is a high-level directory structure in the CY3280-CPM1 kit CD-ROM.

```
|---Docs              The 'Docs' folder contains the kit documentation in PDF form
|
|---Hardware        The 'Hardware' folder contains design files used in kit development
|       |---Schematic
|       |---BOM
|       |---SilkScreen
|       |---Gerber
|
|---Firmware         The 'Firmware' folder contains firmware of code examples
|       |--- RTC_Lab
|       |--- SAR10_Lab
|       |--- IPWMDB_Deadband_Lab
|       |--- IPWMDB_MultiShot_Lab
|       |--- SHIFTREG8_Lab
|       |--- SPIVL_Master_Lab
|       |--- SPIVL_Master_Slave_Communication_Lab
|       |--- Thermistor_Lab
|
|---Hex Files         The 'Hex Files' folder contains hex files of code examples
|       |--- RTC_Lab.hex
|       |--- SAR10_Lab.hex
|       |--- IPWMDB_Deadband_Lab.hex
|       |--- IPWMDB_MultiShot_Lab.hex
|       |--- SHIFTREG8_Lab.hex
|       |--- SPIVL_Master_Lab.hex
|       |--- SPIVL_Master_Slave_Communication_Lab.hex
|       |--- Thermistor_Lab.hex
```

1.3 CY3280-CPM1 Module Board Hardware

The following figure illustrates the components of the CY3280-CPM1 Module board.

Figure 1-1. Kit Components


40-Pin Connector
Buttons
Reserved for Thermocouple
LEDs
LED Panel
LED Bar
Thermistor
Reserved for Speaker
Loopback Jumper for SPI Communication
Reserved for Speaker Driver
Potentiometer

The features of the CY3280-CPM1 Module board components are briefly described here:

- Two mechanical buttons - SW1 (P3.5) and SW2 (P3.7)
- LED panel - The LED panel is driven by a serial expanding chip 74HC164; the chip is controlled by the 8-bit SPI master module inside PSoC. The LED panel contains four digital 7-segment LEDs with one dot LED in the center. Each 7-segment LED is controlled by a transistor (Q1–Q5), which is connected to P4.0–P4.4
- Potentiometer - Attached to a 10-bit SAR ADC through P0.0
- Six LEDs - LED1–LED6 are connected to P4.0–P4.5
- LED bar - Demonstrates the variable length SPI user module. It is enabled by a transistor (Q6), which is connected to P4.5
- Loop-back connection jumpers (JP3 and JP4) - Demonstrates the variable length SPI master-slave communication
- Audio - Audio with different tones is generated through a DAC (P0.1); a speaker is used to play the audio (available in the CY3280-28xxx Universal CapSense Controller Development Kit only)
- NTC thermistor - Measures the temperature and is driven by VCC through a resistor divider. The voltage on NTC thermistor is then connected to the 10-bit SAR ADC for measurement

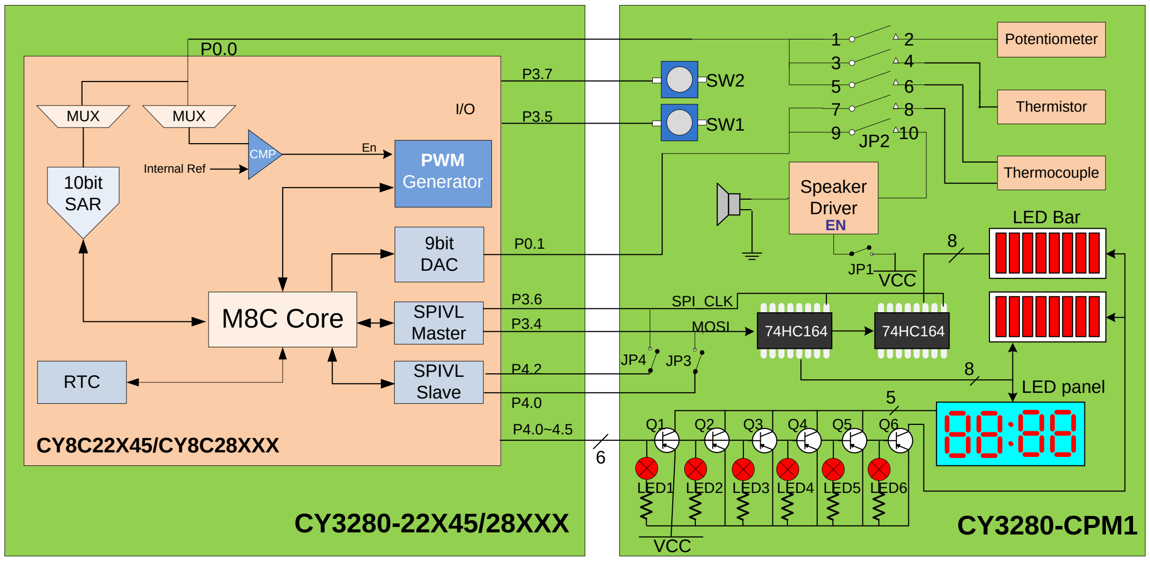
- Thermocouple - Connected to the CY8C28xxx through the INSAMP user module, then through a low-pass filter, and then goes to the Delta-Sigma ADC (available for CY3280-28xxx Universal CapSense Controller Development Kit only)
- Test points/pads - Used for power and ground
- Four rubber feet - Provides mechanical stability

Note The external crystal oscillator (ECO) circuit multiplexes PSoC device pins (P10 and P11) with the ISSP interface. If you select ECO for a 32-kHz clock source, then disconnect PSoC MiniProg from ISSP interface. Otherwise, the ECO will not work correctly.

1.4 System Block Diagram

The CY3280-CPM1 board is a daughter board of CY3280-22x45 and CY3280-28xxx Universal CapSense Controller Development Kits. Use these kits together to analyze the advanced CapSense Plus features provided by CY8C22x45/CY8C28xxx.

Figure 1-2. CapSense Plus System Structure Diagram



1.5 Document Revision History

Table 1-1. Revision History

Revision	PDF Creation Date	Origin of Change	Description of Change
**	04/14/09	WCAI/AESA	New kit guide
*A	01/08/09	WCAI/AESA	Updated Section 2.3.3.2, bullet 3 to disconnect JP3 and bullet 4 to disconnect JP4
*B	03/21/11	JIAO	Extensive edits and content reorganization

1.6 Documentation Conventions

Table 1-2. Document Conventions for Guides

Convention	Usage
`Courier New`	Displays file locations, user entered text, and source code: `C:\ ...cd\icc\`
Italics	Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
[**Bracketed, Bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > Open	Represents menu paths: File > Open > New Project
Bold	Displays commands, menu paths, and icon names in procedures: Click the **File** icon and then click **Open**.
Times New Roman	Displays an equation: $2 + 2 = 4$
Text in gray boxes	Describes Cautions or unique functionality of the product.

2. Code Examples

This section describes the code examples that accompany the CY3280-CPM1 kit.

2.1 RTC

This example demonstrates the real time clock (RTC) feature provided by CY8C22x45 and CY8C28xxx devices. It shows real time on the LED panel.

2.1.1 Introduction

The RTC user module is a new hardware module in CY8C22x45 and CY8C28xxx devices. RTC provides real time without firmware maintenance. It supports the Hour:Minute:Second format. Time is displayed by reading data from related registers. Interrupts may be generated based on the value of the corresponding user-configurable parameter. RTC supports two functional modes: general timer and real time clock.

2.1.2 Procedure

1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board
2. Program the CY3280-22x45/CY3280-28xxx board with the RTC lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Disconnect JP3 and JP4
5. Power the CY3280-22x45/CY3280-28xxx board

2.1.3 Result

The elapsed time is displayed in real time on the CY3280-CPM1 board LED panel.

2.2 10-Bit SAR ADC

This example demonstrates the SAR10 ADC feature provided by CY3280-22x45 and CY3280-28xxx devices. A potentiometer is connected in serial between VCC and ground; the voltage across the potentiometer is attached to the SAR10 ADC module through P00. The potentiometer voltage is measured by the SAR10 ADC. The ADC result is displayed on the LED panel.

2.2.1 Introduction

The SAR10 ADC user module is built for optimized ADC hardware for CY3280-22x45 and CY3280-28xxx devices. It converts an input voltage to a digital code using a SAR block. It produces a 10-bit unsigned value for each sample. This user module supports three modes of analog-to-digital conversion: software trigger, hardware trigger, and freerun.

2.2.2 Procedure

1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board
2. Program the CY3280-22x45/CY3280-28xxx board with the SAR10 lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Connect JP2_1 and JP2_2
5. Disconnect JP2_3 and JP2_4
6. Disconnect JP2_5 and JP2_6
7. Disconnect JP3 and JP4
8. Power the CY3280-22x45/CY3280-28xxx board
9. Tune the potentiometer to view the result

2.2.3 Result

The ADC value is displayed on the LED panel. The value ranges from 0 to 1023 and reflects the voltage on the potentiometer.

2.3 IPWMDB Examples

The CY3280-22x45 and CY3280-28xxx devices introduce a new user module: Integrated Pulse Width Modulator Dead Band (IPWMDB). It includes PWMDB8L and PWMDB16L. The PWMDB8L is an enhanced version of PWM8, which can support dead band feature in one digital block. It also has improved features such as one-shot and multi-shot. The PWMDB16L is an enhanced version of PWM16, which can support dead band feature and consumes only two digital blocks.

2.3.1 IPWMDB Dead Band

This example demonstrates the IPWMDB dead band feature provided by the CY3280-22x45 and CY3280-28xxx devices. It implements two modules in PSoC chip:

- PWMDB8L module with 50 percent duty and 2s period

  It is used to drive two LEDs: LED1 and LED2. Dead time is inserted between PWM output transition. The dead time can be measured by scope or is visible during the transition.
- Voltage comparator
  - One input of the comparator is connected to the potentiometer voltage output (P00).
  - The other input of comparator is connected to the internal voltage reference. The voltage threshold is set in source code.
  - The comparator output controls the PWMDB8L kill pin.

2.3.1.1 Procedure

1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board
2. Program the CY3280-22x45/CY3280-28xxx board with the IPWMDB deadband lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Connect JP2_1 and JP2_2
5. Disconnect JP2_3 and JP2_4
6. Disconnect JP2_5 and JP2_6
7. Disconnect JP3 and JP4

8. Power the CY3280-22x45/CY3280-28xxx board
9. Tune the potentiometer to view the result

2.3.1.2 *Result*

- When potentiometer voltage is below reference voltage of comparator, both LED1 and LED2 are off
- When potentiometer voltage exceeds reference voltage of comparator, LED1 and LED2 will flash periodically and alternately, with a dead time (LED1 and LED2 are off) during flashing

2.3.2 IPWMDB Multi-shot

This example demonstrates the IPWMDB multi-shot feature provided by the CY3280-22x45 and CY3280-28xxx devices. One LED (LED1) and two buttons (SW1 and SW2) are used. SW2 acts as the start input of the IPWMDB module; when it is pressed, the IPWMDB is triggered and generates the multi-shot PWM output pulses. SW1 acts as the kill input of the IPWMDB module. The PWM output drives LED1 directly.

2.3.2.1 *Procedure*

1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board
2. Program the CY3280-22x45/CY3280-28xxx board with the IPWMDB multi-shot lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Disconnect JP3 and JP4
5. Power the CY3280-22x45/CY3280-28xxx board

2.3.2.2 *Result*

- Leave SW1 and SW2 unpressed; LED1 remains off
- Press SW2 once; LED1 flashes five times depending on the multi-shot number set in source code
- Press SW2 again; LED1 flashes five times again
- When SW1 is pressed, LED1 is off

2.4 Shifter

This example demonstrates the Shifter feature provided by the CY3280-22x45 and CY3280-28xxx devices. Three LEDs (LED1, LED2, and LED4) and one button (SW1) are used. LED4 indicates the period of shift clock. LED1 is directly controlled by the SW1 input, while LED2 is controlled by the SW1 input with a shifter register between them, so that LED2 changes with a little delay after LED1; the delay depends on the value of the shifter register.

2.4.1 Introduction

The new user module SHIFTREG8 is built for digital signal shifting in applications such as the FSK. It is a modular linear feedback shift register (LFSR) that delays an input bit stream. The Delay Cycle Number value can be specified to define its output, delayed up to eight PSoC block clocks. Digital blocks can be cascaded to build up the shifter register chain.

2.4.2 Procedure

1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board

2. Program the CY3280-22x45/CY3280-28xxx board with the Shifter lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Disconnect JP3 and JP4
5. Power the CY3280-22x45/CY3280-28xxx board

2.4.3 Result

- After powering on, LED4 will flash with the same period as the shift clock
- After powering on, the LED2 will be on for a while due to the default reset status and 'Length' setting in SHIFTREG8
- Press SW1 once, LED1 will flash once, LED2 will also flash once after a delay. Note that duration of pressing SW1 should be larger than period of shift clock, which is indicated by LED4
- Press SW1 and hold it; LED1 is turned on. LED2 will also be turned on after a delay; the delay depends on the 'Length' setting in SHIFTREG8. When SW1 is released, LED1 is turned off, then LED2 is also turned off after a delay

2.5 Variable Length SPI Examples

The CY3280-22x45 and CY3280-28xxx devices introduce a new user module, Variable Length SPI. This user module provides an SPI user module that can be configured as variable data length. You can configure arbitrary data length from 9 to 16 bits. It includes two types: SPI master (SPIMVL) and SPI slave (SPISVL).

2.5.1 Variable Length SPI Master

This example demonstrates the SPIMVL feature provided by the CY3280-22x45 and CY3280-28xxx devices using the LED bar. The SPIMVL sends out data of different length (from 9 to 16 bits) periodically; this data is displayed on the LED bar. The number of LEDs turned on indicates the SPI data length parameter, therefore, the number of LEDs turned on also changes from 9 to 16 periodically.

2.5.1.1 Procedure

1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board
2. Program the CY3280-22x45/CY3280-28xxx board with the Variable Length SPI Master lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Disconnect JP3 and JP4
5. Power the CY3280-22x45/CY3280-28xxx board

2.5.1.2 Result

The number of LEDs turned on changes from 9 to 16 periodically. This indicates that SPI data length also changes from 9 to 16 bits periodically.

2.5.2 Variable Length SPI Master-Slave Communication

This example demonstrates the SPIMVL and SPISVL features provided by the CY3280-22x45 and CY3280-28xxx devices. One SPIMVL UM and one SPISVL UM are used. Data length parameters of SPIMVL and SPISVL are always set to the same value, which changes from 9 to 16 bits periodically. SPIMVL sends out a specific data, which is a function of SPI length parameter; SPISVL receives the data. If the data received equals the data sent, the data is displayed on the LED bar. Otherwise, the LED bar remains off.

2.5.2.1 Procedure

1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board
2. Program the CY3280-22x45/CY3280-28xxx board with the Variable Length SPI Master-Slave Communication lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Connect JP3 and JP4
5. Power the CY3280-22x45/CY3280-28xxx board

2.5.2.2 Result

- The LED bar is always on, indicating that the communication between SPIMVL and SPISVL is successful
- The number of LEDs turned on changes periodically
  - The lower 8 bits of LEDs are always on
  - The higher 8 bits of LEDs are turned on individually in rotation. This indicates that the data lengths of both SPIMVL and SPISVL change from 9 to 16 bits periodically and at the same pace.

2.6 NTC Thermistor

2.6.1 Introduction

This example demonstrates a temperature measurement method with the NTC thermistor. You can make your own temperature measurement application using this example.

2.6.2 Procedure

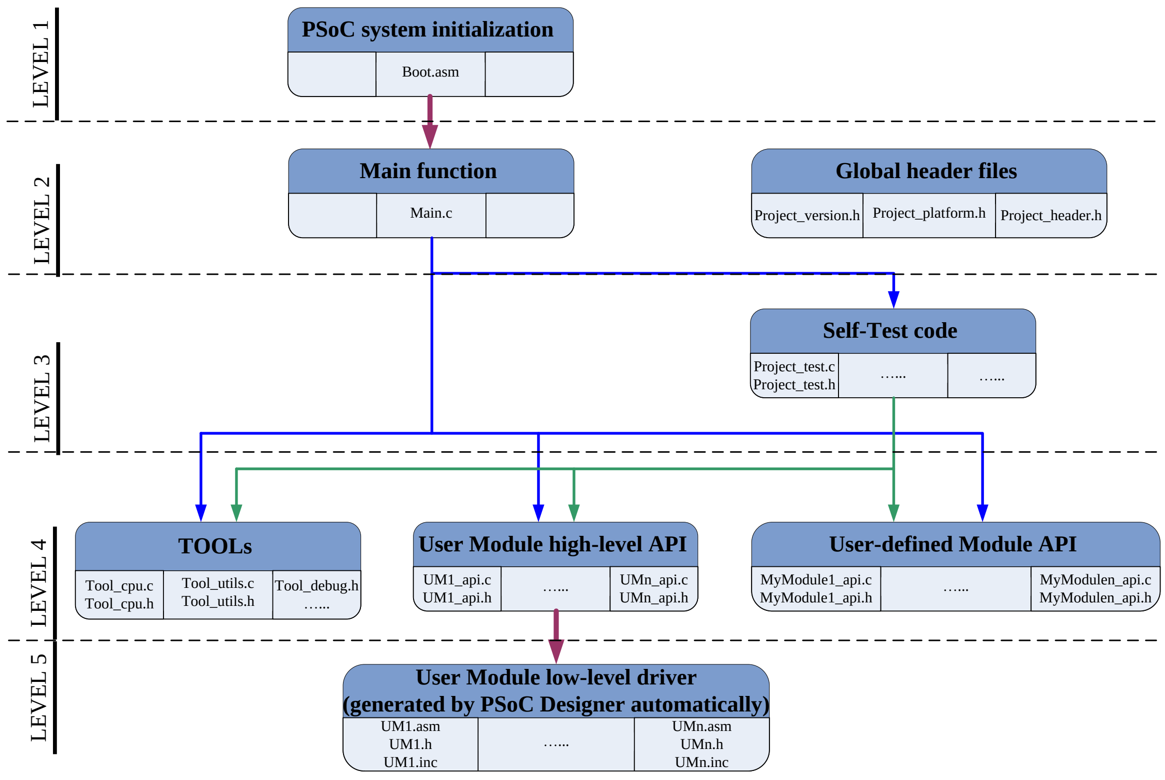
1. Attach the CY3280-CPM1 board to the CY3280-22x45/CY3280-28xxx board
2. Program the CY3280-22x45/CY3280-28xxx board with the NTC thermistor lab hex file using MiniProg via the ISSP interface
3. Unplug MiniProg from the CY3280-22x45/CY3280-28xxx board
4. Connect JP2_3 and JP2_4
5. Disconnect JP2_1 and JP2_2
6. Disconnect JP2_5 and JP2_6
7. Disconnect JP3 and JP4
8. Power the CY3280-22x45/CY3280-28xxx board

2.6.3 Result

The temperature is displayed on the LED panel.

3. Firmware

All CapSense Plus examples are based on firmware architecture similar to the following diagram. The architecture includes five levels and each level is discussed in detail in the following sections. These levels are effective for code reuse and can speed up project development.



3.1 Level 1

This level implements PSoC system initialization, then calls main functions directly. The related source files are generated by PSoC Designer™ automatically. The important source files are *boot.asm* and *PSoCConfig.asm*. They are briefly discussed here; for more details, refer to the *IDE User Guide.pdf* and source code.

3.1.1 Boot.asm

This is the startup file of PSoC firmware system and is located in the Source Files folder. It defines the boot sequence:

- Define and allocate reset and interrupt vectors
- Initialize device configuration

- Initialize C environment if using the C compiler
- Call main function to begin executing the application code

When a project is created, the template file, *boot.tpl*, is copied into the project directory. Each time the project is generated, the *boot.asm* file is also generated from the local *boot.tpl* file. *Boot.asm* is regenerated every time device configurations change and application files are generated. This ensures that interrupt handlers are consistent with the configuration. If you make changes to *boot.asm* that you do not want overwritten, modify the local project *boot.tpl* file and then regenerate the file.

3.1.2 PSoCConfig.asm

This is a required Library Source file because it contains the configuration that is loaded at system power-up. PSoC Designer overwrites *PSocConfig.asm* automatically when a device configuration changes and application files are regenerated, with no exceptions.

3.2 Level 2

This level includes main function and global header files. Almost all the user interfaces are implemented here:

- Implement main() in *Main.c*
- Configure firmware into different versions in *Project_version.h*
- Define different hardware platforms in *Project_platform.h*

3.2.1 Main.c

The C entry function main() is implemented here. You can combine all lower levels (level 3 ~ level 5) code in main() to implement the whole system functionality. Refer to the source code for detailed information.

3.2.2 Project_version.h

This file defines project version related macros. These macros work as compiling flags; modifying their value provides different compiling results, which represent different versions of code.

Different PSoC hardware configurations. The PSoC device is flexible and configurable, so the firmware should also be flexible enough to support different hardware configurations. This way, when you change hardware configuration, you need to make minimum modifications such as changing a macro’s definition.

Different PSoC parts. The PSoC device has a series of families, such as: CY8C29xxx and CY8C24xxx. They are similar and compatible with each other to some extent. It may be necessary to migrate the firmware from one family to another. The firmware architecture should be flexible enough to support different families to allow firmware migration with minimum modification.

Different compiler configurations. Sometimes it is necessary to add self-testing or debugging code to the source code. This needs to be removed after completing debugging or testing. The *project_version.h* defines several macros, such as DEBUG, SELF-TEST, and RELEASE. They are applied in the source code to control compiling results and support different firmware development stages.

Different hardware platforms. PSoC targets small and flexible systems. The hardware platform may have different versions with little difference, such as different pin assignments. The firmware architecture should be flexible enough to be compatible with different hardware platforms. This will allow firmware migration between different hardware platforms with minimum firmware modification.

With *project_version.h* file, the firmware architecture can be easily applied into similar projects. Therefore, the source code can be reused as much as possible.

3.2.3 Project_platform.h

This file defines hardware platform related information such as pin assignment. The hardware platform may have different versions with little difference. You can list all versions of hardware platforms in this file, and then combine with *Project_version.h* to control the compiler to generate different versions of programming files for different hardware platforms.

3.2.4 Project_header.h

PSoC Designer generates a header file *psocapi.h* to include all user module header files. Similarly, the firmware architecture discussed here also contains a header file *project_header.h* to include all project-related header files. You can reference all the firmware resource, such as global constants, global variables, and global functions, in your code by simply adding this file to your source file.

3.3 Level 3

During firmware development, it is necessary to add self-test code in debugging or testing stages, especially when there are no emulating tools. This level implements self-test code, which is in the project range. If there is a bug, locate it and confirm if it is triggered by a single module. Then, add the self-test code in *UM_api.c*. If the bug is triggered in system level, all modules will work individually, but the bug appears when they are combined. In this case, add self-test code in *Project_test.c*. After the bug is located, you can eliminate the self-test code from the final programming file by modifying the macro defined in *Project_version.h*. This can be reused later.

3.3.1 Project_test.c

This file implements project-range self-test code. Combine all lower level (level 4 to level 5) code to test the system functions. These codes are conditionally compiled into final hex file by the macro defined in *Project_version.h*. It also provides and interface to call the user module range self-test code. See the source code for details.

3.3.2 Project_test.h

This is the header file of *project_test.c*. See the source code for details.

3.4 Level 4

This is the core level in the firmware architecture. All code in this level is reusable and expandable. It includes the following categories:

- User module high-level API
- User defined module API
- Embedded firmware tool

3.4.1 User Module High-Level API

Cypress provides several ready-to-use user modules. Every user module is integrated with a complete low-level firmware driver, which can be referenced directly in the source code. The user module low-level driver is fixed and generated automatically. The high-level API is based on the low-level driver, and similar to an expandable and sharable UM API library, you can combine any low-level functions to create a new high-level function to satisfy specific features.

3.4.1.1 *UM_api.c*

This is a high-level API code for specific user module, the 'UM' represents the name of the user module; for example, you can create 'Timer_api.c' for timer.

UM_api.c does the following tasks:

Implement new low-level functions. This is to supplement the original low-level driver generated by PSoC Designer automatically when necessary

Implement high-level functions. This is to combine low-level functions to implement specific features.

Implement user module range self-test functions. This is to implement self-test code, which is only used to test user module features. It is different from the project-ranged self-test code implemented in Project_test.c

Create data members for user modules. This is similar to object-oriented language, such as C++, whose object always has two fields: 'member functions' and 'data members'. *UM_api.c* also creates data members for user modules when necessary.

Be compatible with different user modules in the same category. In some cases, several user modules may fall into the same category. For timer, PSoC Designer provides 'sleep timer' and 'normal timer'; For ADC, there are 'ADCINC' and 'SAR'. These user modules, especially their high-level APIs, are similar. Therefore, *UM_api.c* should be compatible with different user modules in the same category.

3.4.1.2 *UM_api.h*

This is header file of *UM_api.c.* It performs the following tasks:

- Defines macros for conditionally compiling *UM_api.c* into different versions to be compatible with different user modules in the same category

  for example:

  ```
  #define ADC_TYPE_SAR 1
  #define ADC_TYPE_ADCINC 2
  #define ADC_TYPE_SELECTION     ADC_TYPE_SAR
  ```
- Defines user module related constants

  for example:

  ```
  #define ADC_RESOLUTION 10
  ```
- Defines user module related data type

  for example, for 8-bit ADC and 10-bit ADC, defines a new data type (ADC_WORD) to support different ADC resolution

  ```
  ////////////////////////////////////////////////////////////////////
  //Define ADC data type based on ADC_RESOLUTION
  #if (ADC_RESOLUTION <= 8)
  typedef  unsigned char  ADC_WORD;
  #else
  typedef  unsigned int   ADC_WORD;
  ```

#endif//(ADC_RESOLUTION <= 8)

//

- Defines user module related data type for 'data members', see the RTC module example:

  typedef struct {

  unsigned char bHour;

  unsigned char bMinute;

  unsigned char bSecond;

  unsigned char bBCDHour;

  unsigned char bBCDMinute;

  unsigned char bBCDSecond;

  } RTCAPI_PARAMS_STRUCT;
- Declares user module related global variables

  extern RTCAPI_PARAMS_STRUCT RtcApi_tParams;
- Declares user module related global functions
- Defines macros to conditionally compile *UM_api.c* into different versions and support user module range self-test feature

3.4.2 User Defined Module API

Besides ready-made user modules provided by PSoC Designer, you may need customized modules such as a software module that only implements arithmetic or a new module created by combining other available user modules to implement complex functionality. Such modules are called 'user defined module' and are packed and integrated into this level for maximum code reuse.

MyModule_api.c. There is no common coding pattern for *MyModule_api.c*. It depends on specific functions.

MyModule _api.h. This is header file of *MyModule_api.c*.

3.4.2.1 Example1

Following is an example of a user defined module, 5-digit 7-segment LED

- The 7-segment LED is driven by 74HC164 chip whose interface with MCU is 8 bits SPI, so an 'SPIM' user module is required.
- Five digits need 5 pins to turn them on and off, so you need five LED user modules.
- One 'timer8' is required to implement 7-segment LED scanning period.

Therefore, this user defined module is the combination of one SPIM, one timer8, and five LED user modules. The source code of '5-digit 7-segment LED' should be based on APIs of 'SPIM', 'timer8', and 'LED'.

3.4.2.2 Example2

Button is a basic component in embedded system. It is used to add delay and confirm button status to prevent fake button trigger. A new user defined module can be created to implement Button_fIsPressed() function for code reuse.

3.4.3 Embedded Firmware Tool

This part of the code contains tools for debugging, testing, or speeding up firmware development. It is similar to an embedded firmware library or a C standard library. You can make your own tool and expand the tool box. The following are some of the existing tools.

Tool_debug.h. This file defines a series of debugging tools that are useful during code debugging, especially when there are no emulation tools. Insert these tools in your source code directly where you want to set a test point. These tools can be conditionally compiled into the final hex file by the macro defined in *Project_version.h*. See the source code for details.

Tool_utils.c. This file defines miscellaneous functions that satisfy the following rules:

- simple functions used frequently
- hardware independent
- small code size (this rule is optional, because HI-TECH compiler can eliminate unused code automatically)

A typical example to be added into *tool_utils.c* is the 'delay subroutine'. The 'delay subroutine' satisfies all the rules listed above, it is small and does not depend on any hardware. It also occupies little ROM space. See the source code for details.

Tool_utils.h. This is the header file of *tool_utils.c*. See the source code for details.

Tool_cpu.c. This file defines miscellaneous functions to implement low-level CPU related features which can be reused in any project. See the source code for details.

Tool_cpu.h. This is the header file of *tool_cpu.c*. See the source code for details.

3.5 Level 5

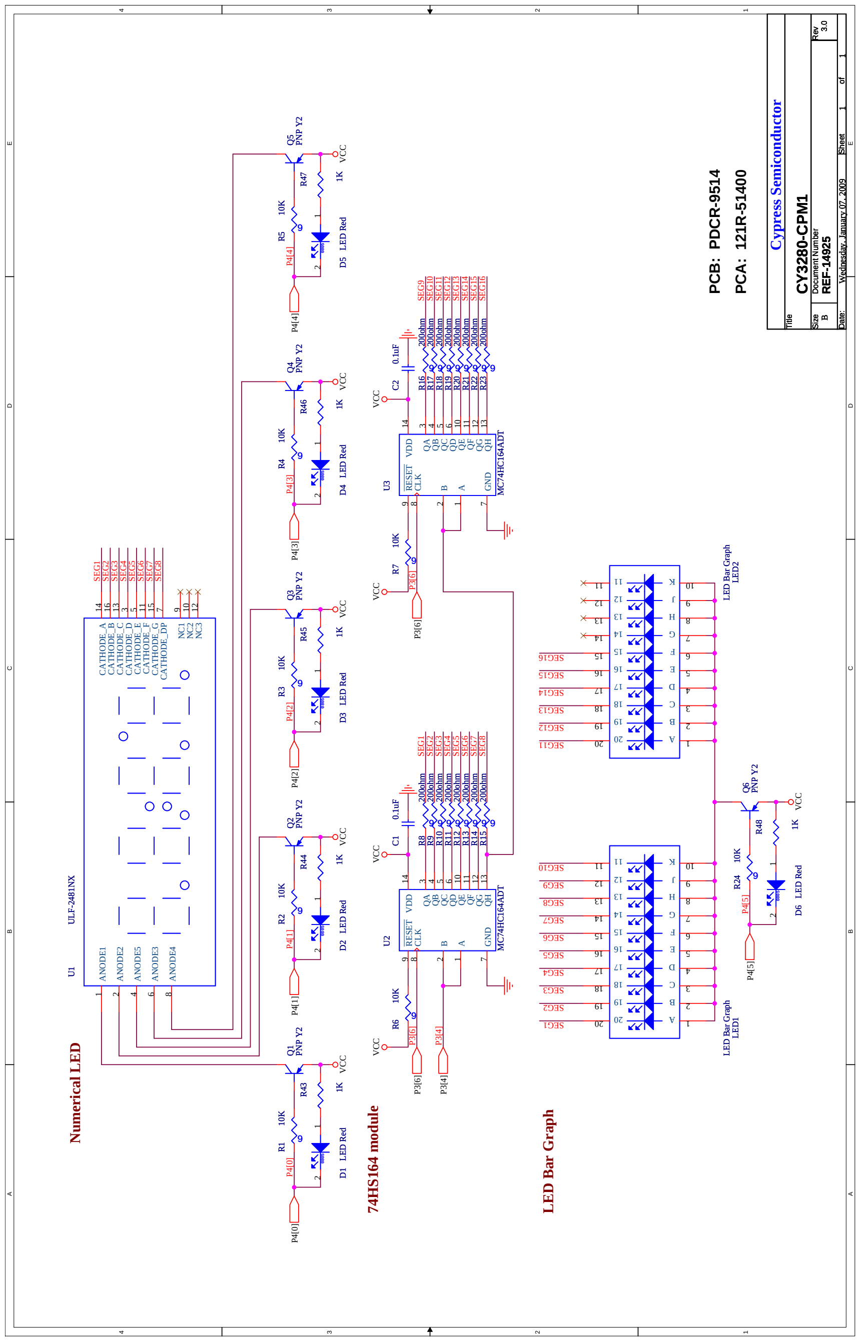
The PSoC device is highly flexible and configurable. Cypress provides several ready-to-use user modules. Every user module is integrated with complete low-level firmware driver and a detailed data sheet.

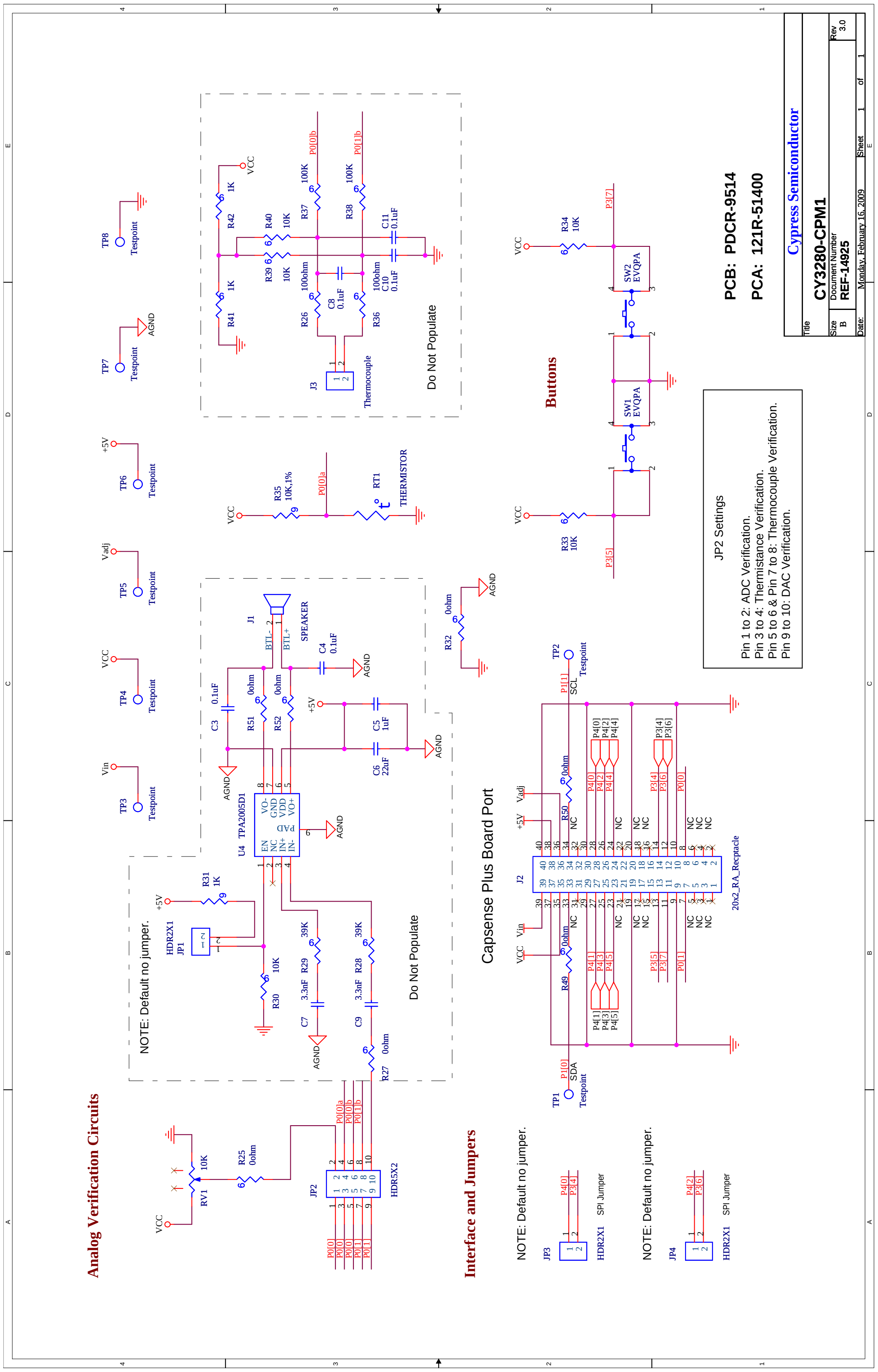
3.5.1 User Module Low-level Driver

PSoC Designer generates the source code of low-level drivers for a user module in the Device Editor. These drivers can be referenced directly in the source code. The driver is a composite of *UM.asm*, *UM.h*, and *UM.inc*. For more information, refer to the respective user module data sheet.

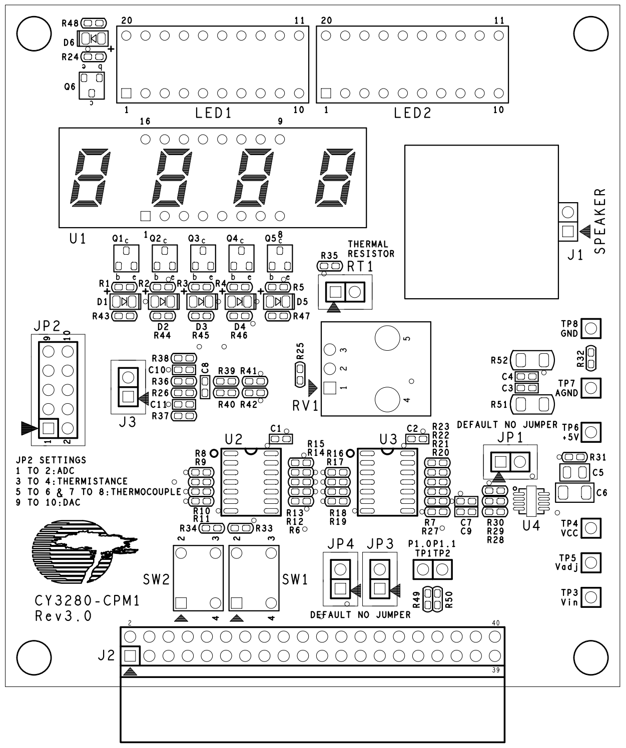
A. Appendix

A.1 Schematic




Analog Verification Circuits
Interface and Jumpers
Capsense Plus Board Port
Buttons
NOTE: Default no jumper.
Do Not Populate
JP2 Settings
Pin 1 to 2: ADC Verification.
Pin 3 to 4: Thermistance Verification.
Pin 5 to 6 & Pin 7 to 8: Thermocouple Verification.
Pin 9 to 10: DAC Verification.
PCB: PDCR-9514
PCA: 121R-51400
Cypress Semiconductor
Title
CY3280-CPM1
Size B
Document Number REF-14925
Rev 3.0
Date: Monday, February 16, 2009
Sheet 1 of 1
U4 TPA2005D1
J1 SPEAKER
RT1 THERMISTOR
J3 Thermocouple
J2 20x2_RA_Recptacle
JP2 HDR5X2
JP3 HDR2X1 SPI Jumper
JP4 HDR2X1 SPI Jumper
SW1 EVQPA
SW2 EVQPA
TP1 Testpoint
TP2 Testpoint
TP3 Testpoint
TP4 Testpoint
TP5 Testpoint
TP6 Testpoint
TP7 Testpoint
TP8 Testpoint

A.2 Top Silk Screen



A.3 Bill of Material

Item	Qty.	Reference	Description	Manufacturer	Mfr Part Number
1	7	C1,C2,C3,C4,C8,C10,C11	CAP CER 0.10UF 25V X7R 10% 0603	AVX Corporation	06033C104KAT2A
2	1	C5	CAP CER 1.0UF 25V X7R 10% 0805	TDK Corporation	C2012X7R1E105K
3	1	C6	CAP CER 22UF 16V X5R 20% 1206	Murata Electronics North America	GRM31CR61C226ME 15L
4	2	C7,C9	CAP CER 3300PF 10V X7R 10% 0603	AVX Corporation	0603ZC332KAT2A
5	6	D1,D2,D3,D4,D5,D6	LED RED 635NM DIFF LENS 0805	Sharp Microelectronics	LT1D40A
6	3	JP1,JP3,JP4	CONN HEADER 2POS .100" STR TIN	Molex Connector Corpo-ration	90120-0122
7	1	JP2	CONN HEADER 10POS .100 STR TIN	FCI	67997-410HLF
8	1	J1	SPEAKER INTERFACE	N/A	N/A
9	1	J2	CONN HEADER 40POS .100" R/A TIN	Molex Connector Corpo-ration	90122-0140
10	1	J3	PROBE INTERFACE	N/A	N/A
11	2	LED1,LED2	LED BAR GRAPH 10-SEGMENT GREEN	Lite-On Inc.	LTA-1000G
12	6	Q1,Q2,Q3,Q4,Q5,Q6	TRANSISTOR SWITCHING PNP SOT-23	Fairchild Semiconductor	MMBT3702
13	1	RT1	THERMISTOR NTC 10K OHM 5% RAD	EPCOS Inc.	B57891M0103J000
14	1	RV1	POT 10K OHM 9MM VERT NO BUSHING	Panasonic - ECG	EVU-F3AF30B14
15	13	R1,R2,R3,R4,R5,R6,R7,R24, R30,R33,R34,R39,R40	RES 10K OHM 1/10W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEYJ103V
16	16	R8,R9,R10,R11,R12,R13,R1 4,R15,R16, R17,R18,R19,R20,R21,R22, R23	RES 200 OHM 1/10W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEYJ201V
17	5	R25,R27,R32,R49,R50	RES ZERO OHM 1/10W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEY0R00V
18	2	R26,R36	RES 100 OHM 1/10W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEYJ101V
19	2	R28,R29	RES 39K OHM 1/10W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEYJ393V
20	9	R31,R41,R42,R43,R44,R45, R46,R47,R48	RES 1.0K OHM 1/10W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEYJ102V
21	1	R35	RES 10.0K OHM 1/10W 1% 0603 SMD	Panasonic - ECG	ERJ-3EKF1002V
22	2	R37,R38	RES 100K OHM 1/10W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEYJ104V
23	2	R51,R52	RES ZERO OHM 1/4W 5% 1206 SMD	Panasonic - ECG	ERJ-8GEY0R00V
24	2	SW1,SW2	LT SWITCH 6MM H=5MM 130GF	Panasonic - ECG	EVQ-PAD05R
25	8	TP1,TP2,TP3,TP4,TP5,TP6, TP7,TP8	TEST POINT PC MINI .040"D BLACK	Keystone Electronics	5001
26	1	U1	LED 7-SEG .4" 4DGT SUPER RED Com-mon Anode	Lite-On Inc.	LTC-4627JR
27	2	U2,U3	IC SHIFT REG 8BIT SER/PAR 14SOIC	ON Semiconductor	MC74HC164ADG
28	1	U4	IC 1.1W CLASS-D AUDIO AMP 8-SON	Texas Instruments	TPA2005D1DRBR
Special Installation:					
29	1	For J1	SPEAKER 8OHM .3W 87DB 15MM SMD	PUI Audio	SMS-1508-2-R
30	1	For J3	PROBE TEMP "K" 4' INSULATED LEAD	TPI (Test Products Int)	GK11M
31	4		BUMPER CLEAR .440X.20" DOME	Richco Plastic Co	RBS-2
32	6		JUMPER, CONN JUMPER SHORTING TIN	Sullins Electronics Corp	STC02SYAN