CY3280-CPM1 CapSensePlus Module Development Kit Guide

Spec. # 001-51922 Rev. **

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

Welcome to the CY3280-CPM1 CapSensePlus Module Development Kit. CPM represents CapSensePlus Module. CapSensePlus is defined as capacitive sensing with other value-added features, such as: LED color mixing, fan control, motor control, temperature sensing, and more. Each feature that the PSoC can integrate in addition to CapSense is defined as a "PLUS" feature.

This kit showcases the advanced CapSensePlus features provided by CY8C22X45 and CY8C28XXX. It is a daughter board of CY3280-22X45 and CY3280-28XXX Universal CapSense Controller Development Kits. Some examples are created to demonstrate the new features of CY8C22X45 and CY8C28XXX, which include RTC, 10-bit SAR ADC, Variable Length SPI, and PWM.

This kit can also serve as a development platform. You can easily reuse the source code for your own CapSensePlus applications.

This kit user guide discusses CapSensePlus lab implementations.

1. This chapter lists kit contents, CD-ROM Directory Structure, CY3280-CPM1 features, and Trouble Shooting.
2. Chapter 2 provides step by step instructions to run all the labs.
3. Chapter 3 describes firmware implementation. You need to understand the source code then reuse the code.

1.1 Review of Kit Components

The following items are included in the kit:

1. One CapSensePlus Module board
2. Kit CD
3. Documentation

1.2 Directory Structure

This list describes the higher level directory structures in the CD-ROM, but does not explore the lower level directories.

|---Docs 'Docs' contains the kit documentation in PDF form
|
|---Hardware 'Hardware' contains the design file used in development of the Kit
| |---Schematic
| |---BOM
| |---SilkScreen
| |---Gerber
|
|---Firmware 'Firmware' contains firmware of example projects
| |--- RTC_Lab
| |--- SAR10_Lab
| |--- IPWMDB_Deadband_Lab
| |--- IPWMDB_MultiShot_Lab
| |--- SHIFTREG8_Lab
| |--- SPIVL_Master_Lab
| |--- SPIVL_Master_Slave_Communication_Lab
| |--- Thermistor_Lab
|
|---Hex Files 'Hex Files' contains hex files of example projects
| |--- 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 Features

1. Two mechanical buttons: SW1 (P3.5) and SW2 (P3.7).
2. An LED panel is driven by a serial expanding chip 74HC164. This chip is controlled by the 8-bit SPI master module inside PSoC. The LED panel contains Four-digital 7-segment LED and one dot LED in the center. They are controlled in a time sharing method. Each 7-segment LED is controlled by a transistor (Q1~Q5), which is connected to P4.0~P4.4.
3. One Potentiometer attaches to a 10-bit SAR ADC through P0.0.
4. Six LEDs (LED1~LED6) are connected to P4.0~P4.5.
5. An LED bar is used to demonstrate the variable length SPI UM. It is enabled by a transistor (Q6) that is connected to P4.5.
6. Loop-back connection jumpers (JP3 and JP4) are used to demonstrate the variable length SPI master-slave communication.
7. Audio with different tones is generated through a DAC (P0.1). A speaker is used to play the audio (available for CY3280-28XXX Universal CapSense Controller Development Kit only).
8. NTC thermistor is used to measure the temperature. It is driven by VCC through a resistor divider. The voltage on NTC thermistor is connected to the 10-bit SAR ADC for measurement.
9. Thermocouple is to precisely measure the temperature. The thermocouple is connected to the CY8C28XXX through the INSAMP UM and a low-pass filter, and then goes to the Delta-Sigma ADC (available for CY3280-28XXX Universal CapSense Controller Development Kit only).
10. Test points/pads for power and ground.
11. Four rubber feet for mechanical stability.

1.4 Trouble Shooting

1.4.1 External Crystal Oscillator Does Not Work Correctly

The External Crystal Oscillator (ECO) circuit multiplex PSoC Device pins (P10 and P11) with ISSP interface. If you select ECO for 32 KHz clock source, then you must disconnect PSoC MiniProg from the ISSP interface. Otherwise, the ECO does not work correctly.

1.5 Document Revision History

Table 1-1. Revision History

Revision	PDF Creation Date	Origin of Change	Description of Change
**	3/13/09	WCAI/ AESA	New Kit Guide

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. Quick Start

2.1 RTC Lab

2.1.1 Introduction

This Real Time Clock (RTC) user module is a new hardware module in CY8C22X45/CY8C28XXX devices. RTC provides the real time without firmware maintenance. It supports the Hour:Minute:Second format. You can set the time displayed by reading out the 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 according to the selected clock source. Refer to the RTC data sheet for more information.

2.1.2 Lab Description

This lab demonstrates the Real Time Clock (RTC) feature provided by CY8C22X45/CY8C28XXX devices. It displays the real time on the LED panel.

2.1.3 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of the RTC lab by MiniProg, through the ISSP interface. Then disconnect the MiniProg from the board.
3. Disconnect JP3.
4. Disconnect JP4.
5. Power on the CY3280-22X45/CY3280-28XXX board.

2.1.4 Lab Result

The time is displayed on the LED panel, which starts from 00:00(min:sec) and elapses in pace with real time.

2.1.5 Lab Interaction

Find solutions for these questions:

1. How to start the RTC from a specific time point?
2. How to make the LED panel display a specific time (for example, 05:00), then decrease time until reaching 00:00, and finally generate an interrupt?

2.2 10-Bit SAR ADC Lab

2.2.1 Introduction

The SAR10 ADC user module is built for the optimized ADC hardware for CY3280-22X45/CY3280-28XXX devices. It is an 10-bit Successive Approximation Register (SAR) ADC converter that 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. Refer to the SAR10 data sheet for more information.

2.2.2 Lab Description

This lab demonstrates the SAR10 ADC feature provided by CY3280-22X45/CY3280-28XXX devices. A potentiometer is serial connected between VCC and Ground, and 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.

2.2.3 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of SAR10 lab by MiniProg, through the ISSP interface. Disconnect MiniProg from the board.
3. Connect JP2_1 and JP2_2.
4. Disconnect JP2_3 and JP2_4.
5. Disconnect JP2_5 and JP2_6.
6. Disconnect JP3.
7. Disconnect JP4.
8. Power on CY3280-22X45/CY3280-28XXX board.
9. Tune the potentiometer and see the result.

2.2.4 Lab Result

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

2.2.5 Lab Interaction

Find solutions for this question:

SAR10 supports three modes of analog-to-digital conversion: Software Trigger, Hardware Trigger, and Free run. The current firmware implements the free run mode. Could you try the other modes?

2.3 IPWMDB Labs

2.3.1 Introduction

The CY3280-22X45/CY3280-28XXX device introduces a new user module: IPWMDB (Integrated Pulse Width Modulator Dead Band). It includes PWMDB8L and PWMDB16L. The PWMDB8L is an enhanced version of PWM8 that can support dead band feature in one digital block. It has improvements, such as one-shot/multi-shot. In addition, the PWMDB16L is an enhanced version of the PWM16, which can support the dead band feature and consumes only two digital blocks. It also has one-shot/multi-shot feature. Refer to the PWMDB8L and PWMDB16L data sheets for more information.

2.3.2 IPWMDB Dead Band Lab

2.3.2.1 Lab Description

This lab demonstrates the IPWMDB dead band feature provided by CY3280-22X45/CY3280-28XXX device. This lab implements two modules in PSoC chip:

- An PWMDB8L module with 50% duty and 2s period:

  It is used to drive two LEDs: LED1 and LED2. Dead time is inserted between the 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.

2.3.2.2 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of IPWMDB deadband lab by MiniProg, through ISSP interface. Disconnect MiniProg from the board.
3. Connect JP2_1 and JP2_2.
4. Disconnect JP2_3 and JP2_4.
5. Disconnect JP2_5 and JP2_6.
6. Disconnect JP3.
7. Disconnect JP4.
8. Power on the CY3280-22X45/CY3280-28XXX board.
9. Tune the potentiometer and see the result.

2.3.2.3 Lab Result

- When the potentiometer voltage is below the reference voltage of the comparator, then both LED1 and LED2 are off.
- When the potentiometer voltage exceeds the reference voltage of the comparator, then LED1 and LED2 flash periodically and alternately. A dead time (LED1 and LED2 are both off) occurs during flashing.

2.3.2.4 Lab Interaction

Find solutions for these questions:

1. How do you modify the dead time of the IPWMDB and what is the sequential result?
2. How do you modify the reference voltage of the comparator and what is the sequential result?

2.3.3 IPWMDB Multi-Shot Lab

2.3.3.1 Lab Description

This lab demonstrates the IPWMDB multi-shot feature provided by the CY3280-22X45/CY3280-28XXX device. One LED(LED1) and two buttons(SW1 and SW2) are used in this lab. SW2 acts as the start input of the IPWMDB module. When SW2 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.

2.3.3.2 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of the IPWMDB multi-shot lab by MiniProg, through the ISSP interface. Disconnect MiniProg from the board.
3. Disconnect JP4.
4. Disconnect JP5.
5. Power on the CY3280-22X45/CY3280-28XXX board.

2.3.3.3 Lab Result

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

2.3.3.4 Lab Interaction

Find solutions for this question:

How do you modify the multi-shot number of IPWMDB and what is the sequential result?

2.4 Shifter Lab

2.4.1 Introduction

The new user module SHIFTREG8 is built for the digital signal shifting in applications such as FSK. It is a modular linear feedback shift register (LFSR) that delays a 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. Refer to the SHIFTREG8 data sheet for more information.

2.4.2 Lab Description

This lab demonstrates the Shifter feature provided by the CY3280-22X45/CY3280-28XXX devices. Three LEDs (LED1, LED2, and LED4) and one button (SW1) are employed in this lab. LED4 indicates the period of the shift clock. LED1 is directly controlled by the SW1 input, while LED2 is controlled by the SW1 input with a shifter register between them. LED2 changes after LED1, following a delay. The delay time depends on the value of the shifter register.

2.4.3 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of the Shifter lab by MiniProg, through ISSP interface. Disconnect MiniProg from the board.
3. Disconnect JP3.
4. Disconnect JP4.
5. Power on the CY3280-22X45/CY3280-28XXX board.

2.4.4 Lab Result

- After powering on, LED4 flashes with the same period of the shift clock.
- After powering on, the LED2 is on for a while, because of the default reset status and 'Delay Cycle Number' setting in SHIFTREG8.
- Press SW1 once. LED1 flashes once first. Then LED2 also flashes once after a delay. Note that the time duration when SW1 is pressed must be more than the period of the shift clock, which is indicated by LED4.
- Press SW1 and hold it. LED1 is turned on first. Next, LED2 is also turned on after a delay. The delay time depends on 'Delay Cycle Number' setting in SHIFTREG8. Until SW1 is released, LED1 is turned off first, then LED2 is also turned off after a delay.

2.4.5 Lab Interaction

Find solutions for this question:

How do you change the shifter delay time and what is the sequential result?

2.5 Variable length SPI labs

2.5.1 Introduction

The CY3280-22X45/CY3280-28XXX device introduces a new user module: Variable length SPI. This user module provides a SPI user module that you can configure as variable data length. You can configure the arbitrary data length from 9 to 16 bits. It includes two types: SPI master (SPIMVL) and SPI slave (SPISVL). Refer to the SPIMVL and SPISVL data sheets for more information.

2.5.2 Variable Length SPI Master Lab

2.5.2.1 Lab Description

This lab demonstrates the SPIMVL feature provided by CY3280-22X45/CY3280-28XXX devices. The LED bar is used in this lab. The SPIMVL sends out data that is in different lengths (from 9 to 16 bits) periodically. The sent data is then displayed on the LED bar. The number of LEDs turned on indicates the SPI data length parameter. You can see that the number of LEDs turned on also changes from 9 to 16 periodically.

2.5.2.2 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of the Variable length SPI master lab by MiniProg, through the ISSP interface. Disconnect MiniProg from the board.
3. Disconnect JP3.
4. Disconnect JP4.
5. Power on the CY3280-22X45/CY3280-28XXX board.

2.5.2.3 Lab Result

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

2.5.2.4 Lab Interaction

Find solutions for this question:

How do you periodically change the number of LEDs turned on from 10 to 15?

2.5.3 Variable Length SPI Master-Slave Communication Lab

2.5.3.1 Lab Description

This lab demonstrates the SPIMVL and SPISVL features provided by CY3280-22X45/CY3280-28XXX devices. One SPIMVL UM and one SPISVL UM are employed in this lab. The 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 received data equals to the sent data, the data is displayed on the LED bar. Otherwise, the LED bar is always off.

2.5.3.2 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of Variable length SPI master-slave communication lab by MiniProg, through the ISSP interface. Disconnect MiniProg from the board.
3. Connect JP3.
4. Connect JP4.
5. Power on the CY3280-22X45/CY3280-28XXX board.

2.5.3.3 Lab Result

- The LED Bar is always on, which indicates 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 and rotationally. This indicates that the data lengths of both SPIMVL and SPISVL change from 9 to 16 bits periodically, at the same pace.

2.5.3.4 Lab Interaction

Find solutions for these questions:

1. Currently, SPISVL works in polling mode. How do you make it work in interrupt mode?
2. Try to change the mode or clock polarity of both SPIMVL and SPISVL at the same time.

2.6 NTC Thermistor Lab

2.6.1 Introduction

There is no new feature to introduce in this lab. This lab only shows a temperature measurement method with NTC thermistor. You can make your own temperature measurement application in this lab.

2.6.2 Lab Description

This lab demonstrates how to display the environment temperature on an LED panel.

2.6.2.1 Step by Step Procedure

1. Connect the CY3280-CPM1 board to the CY3280-22X45/CY3280-28XXX board.
2. Program the CY3280-22X45/CY3280-28XXX board with the hex file of NTC thermistor lab by MiniProg, through the ISSP interface. Disconnect MiniProg from the board.
3. Connect JP2_3 and JP2_4.
4. Disconnect JP2_1 and JP2_2.
5. Disconnect JP2_5 and JP2_6.
6. Disconnect JP3.
7. Disconnect JP4.
8. Power on the CY3280-22X45/CY3280-28XXX board.

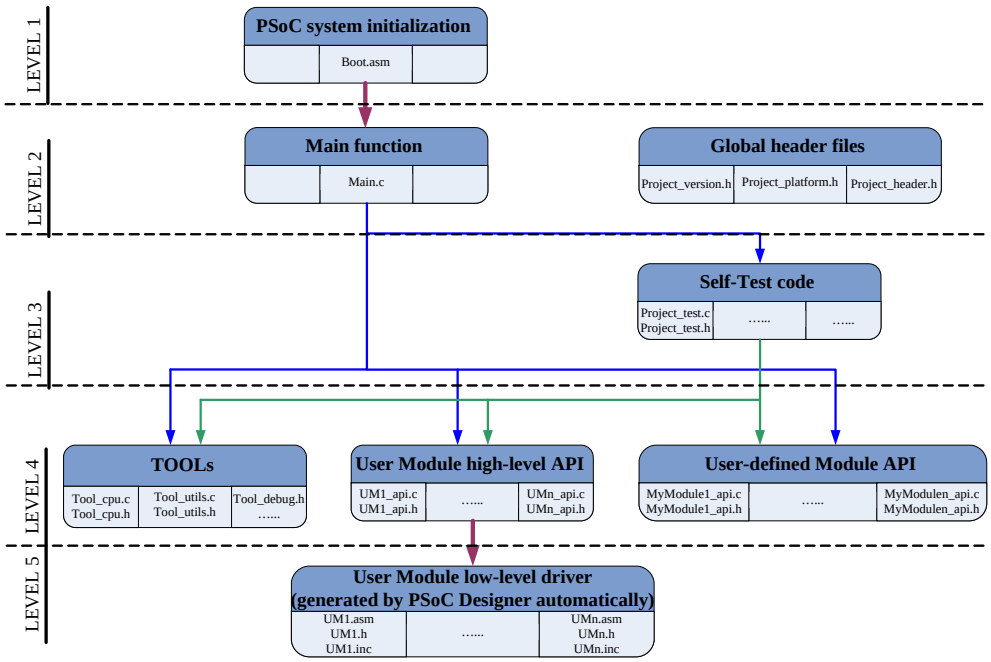
2.6.2.2 Lab Result

The temperature is displayed on LED panel.

3. Firmware

All CapSensePLUS labs are based on the same firmware architecture shown in Figure 3-1. The entire architecture contains five levels. These levels are discussed in the following sections. The levels may appear complex, but it is very effective for code reuse and can speed up project development.

Figure 3-1. Firmware Architecture



3.1 Level 1

This level implements PSoC system initialization and directly calls main functions. The related source files are automatically generated by PSoC Designer. The two important two source files are *boot.asm* and *PSoCConfig.asm*. These files are briefly discussed in this section. Refer to the *IDE User Guide.pdf* and source code for more information.

3.1.1 Boot.asm

This is startup file of the PSoC firmware system that resides in the source tree under Source Files. This file defines the boot sequence. The components of the boot sequence are:

- Define and allocate the reset and interrupt vectors
- Initialize device configuration
- Initialize C environment if using the C Compiler
- Call main to begin executing the application code

When a project is created, the template file, *boot.tpl*, is copied into the project directory. Every time the project is generated, the *boot.asm* file is generated from the local *boot.tpl* file. *Boot.asm* is regenerated every time the device configurations change and application files are generated. This is done to ensure that the 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 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 automatically overwrites *PSocConfig.asm* 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. Most of 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 more information.

3.2.2 Project_version.h

This file defines project version related MACROs. These MACROs work as conditionally compiling flags. Modifying their values get different compiling results, which represents the different versions of the code for:

Different PSoC hardware configurations. PSoC devices are very flexible and configurable. Accordingly, the firmware must be flexible enough to support different hardware configurations. As a result, when you change the hardware configuration, you do not need to modify the firmware. You may only need to modify some MACRO definitions.

Different PSoC parts. PSoC devices have a series of families, such as: CY8C29xxx, CY8C24xxx, and more. They are similar and compatible with each other to an extent. Quite often, the firmware from one family is immigrated to another family. The firmware architecture must be flexible enough to support different families when necessary. As a result, you can implement the firmware immigration with few modifications to the firmware.

Different compiler configurations. During firmware debugging and testing stages, self testing or debugging code may need to be added in the source code. However, after debugging or testing, these codes must be eliminated quickly and completely. The *project_version.h* defines several MACROs, such as DEBUG, SELF-TEST, and RELEASE. These MACROs are applied in the source code to control compiling results to support different firmware development stages.

Different hardware platforms. PSoC targets at small and flexible systems. The hardware platforms may have different versions with few differences, such as different pin assignments. The firmware architecture must be flexible enough to be compatible with different hardware platforms. As a result, you can implement the firmware immigration between different hardware platforms with less firmware modification.

With *project_version.h* file, the firmware architecture can be easily applied into different but similar projects. Therefore, the source code can be reused as much as possible. Refer to the source code for more information.

3.2.3 Project_platform.h

This file defines hardware platform related information, generally pin assignment information. PSoC targets at small and flexible systems. The hardware platforms may have different versions with few differences, such as different pin assignments. If that is the case, the firmware must be compatible with different hardware platforms. You can list all versions of hardware platforms information in this file, and then combine it with *Project_version.h.* This can control the compiler to generate different versions of programming files for different hardware platforms. Refer to the source code for more information.

3.2.4 Project_header.h

PSoC Designer generated a header file *psocapi.h* to include all the header files of the user modules. Similarly, the firmware architecture discussed here also contains a header file *project_header.h* to include all the project-related header files. Then you can reference all the firmware resource (such as, global constants, global variables, and global functions) in your code by adding this file in your source file. Refer to the source code for more information.

3.3 Level 3

During firmware development, you may need to add self test code in debugging or testing stages, especially in cases that lack emulating tools. This level implements self test code that is in project range (accordingly, there exists self test code in single user module range. Refer to *UM_api.c*). When a bug appears, the first step is to locate the bug. If the bug is possibly triggered by a single module, then add some self test code in *UM_api.c* to find it. If the bug is triggered in the system level, it means that the bug appears when the modules are combined, but not when the modules work individually. In this case, add some self test code in Project_test.c to find the bug. When the bug is located, you can eliminate the self test code from the final programming file. This is done by modifying the MACRO defined in Project_version.h, instead of removing them from the source code. You can keep them for future use.

3.3.1 Project_test.c

This file implements project-ranged self test code. You can combine all lower level (level 4 ~ level 5) code here to test the system functionalities. These codes are conditionally compiled into the final hex file by the MACRO defined in *Project_version.h*. It also provides the interface to call UM-ranged self test code. Refer to the source code for more information.

3.3.2 Project_test.h

This is header file of *project_test.c*. Refer to the source code for more information.

3.4 Level 4

This is the core level in the firmware architecture. You can reuse the code in this level for other projects. It is expandable: the more code in this level, the more code can be shared. Anyone can contribute to this level. 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

PSoC devices are flexible and configurable. Cypress provides readymade user modules for customers. Every UM is integrated with completed low level firmware drivers. You can reference these drivers directly in the source code. However, the UM low level driver is fixed and generated automatically, and cannot be modified. The high level API is based on the low level driver. Similar to an expandable UM API library that can be shared, you can combine any low level function to create a new high level function to satisfy specific features. You can also expand the UM library and share it with other projects.

3.4.1.1 *UM_api.c*

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

Generally, *UM_api.c* has these function:

Implement New Low Level Functions

This function supplements the original low level driver automatically generated by PSoC Designer when necessary.

Implement High Level Functions

This function combines low level functions to implement new specific features.

Implement UM-Ranged Self Test Functions

This function implements UM-ranged self test code that is used only to test the UM's features. You need to differentiate it from the project-ranged self test code implemented in Project_test.c

Create Data Members for UM

This is similar to object-oriented language, such as C++, whose object always has 2 fields: 'member functions' and 'data members'. *UM_api.c* also takes the responsibility of creating data members for the UM when necessary.

Be Compatible with Different UMs in the Same Category

In some cases, several UMs may fall into the same category. For timer, PSoC Designer provides 'sleep timer' and 'normal timer'; for ADC, there are 'ADCINC' and 'SAR', and so on. These user modules are similar, especially, their high level APIs are almost the same. As a result, *UM_api.c* must be compatible with different UMs in the same category.

3.4.1.2 *UM_api.h*

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

- Define MACROs for conditionally compiling *UM_api.c* into different versions to be compatible with different UMs in the same category.

  For example:
  ```
  #define ADC_TYPE_SAR 1
  #define ADC_TYPE_ADCINC 2
  #define ADC_TYPE_SELECTION     ADC_TYPE_SAR
  ```
- Define UM related constants.

  For example:
  ```
  #define ADC_RESOLUTION 10
  ```
- Define UM related data type.

  For example, if there are 8-bit and 10-bit ADCs, then you must define a new data type(ADC_WORD) to support the different ADC resolutions.
  ```
  ////////////////////////////////////////////////////////////////
  //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)
  ////////////////////////////////////////////////////////////////
  ```

- Define UM related data type for UM's 'data members'. Here is an example for the RTC module:
```
typedef struct {
unsigned char bHour;
unsigned char bMinute;
unsigned char bSecond;
unsigned char bBCDHour;
unsigned char bBCDMinute;
unsigned char bBCDSecond;
} RTCAPI_PARAMS_STRUCT;
```
- Declare UM related global variables to make them globally visible.
```
extern  RTCAPI_PARAMS_STRUCT  RtcApi_tParams;
```
- Declare UM related global functions to make them globally visible.
- Define MACROs to conditionally compile *UM_api.c* into different versions to support UM ranged self test features.

3.4.2 User Defined Module API

In addition to readymade UMs provided by PSoC Designer, there must be more modules. These modules could be software modules that implement only arithmetic, or new modules created by combining other readymade UMs to implement complex functionalities. These modules are called 'user defined modules' and must be packed and integrated into this level for maximum code reusage.

3.4.2.1 MyModule_api.c

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

3.4.2.2 MyModule _api.h

This is the header file of *MyModule_api.c*.

3.4.2.3 Example1

Here is an example for user defined module: Five digits 7-Segment LED

- The 7-Segment LED is driven by the 74HC164 device whose interface with MCU is 8-bit SPI. This requires an 'SPIM' user module.
- Five digits require five pins to turn them on and off. This requires a 5 'LED' user module.
- One 'timer8' to implement the 7-Segment LED scanning period

The user defined module '5 digits 7-Segment LED' is the combination of one 'SPIM' user module, one 'timer8', and five 'LED' user modules. The source code of '5 digits 7-Segment LED' must be based on the APIs of 'SPIM', 'timer8', and 'LED' to implement '5 digits 7-Segment LED' related functionalities.

3.4.2.4 Example2

The button is a basic component in the embedded system. You need to add a delay and check the button status to prevent fake button trigger. It is better to create a new user defined module for the button, and implement the Button_fIsPressed() function in it for code reusage.

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, which resembles a C standard library. These tools can be used by everyone. You can create your own tools and increase the contents of the tool box. Here are some examples for tools.

3.4.3.1 Tool_debug.h

This file defines a series of debugging tools that are useful during code debugging stage, especially in cases that lack emulation tools. You can insert these tools in your source code directly to set a test point. These tools could be conditionally compiled into a final hex file by the MACRO defined in *Project_version.h*. Refer to the source code for more information.

3.4.3.2 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 the HI-TECH compiler can eliminate unused code automatically.

A typical example that can be added into *tool_utils.c* is 'delay subroutine'. The 'delay subroutine' satisfies all the rules listed here. It is small, does not depend on hardware, and occupies less ROM space. Refer to the source code for more information.

3.4.3.3 Tool_utils.h

This is the header file of *tool_utils.c*. Refer to the source code for more information.

3.4.3.4 Tool_cpu.c

This file defines the miscellaneous functions to implement low level cpu related features that could be reused in any project. Refer to the source code for more information.

3.4.3.5 Tool_cpu.h

This is the header file of *tool_cpu.c*. Refer to the source code for more information.

3.5 Level 5

PSoC devices are flexible and configurable. Cypress provides many readymade user modules for customers. Every UM is integrated with completed low level firmware driver and a detailed data sheet.

3.5.1 User Module Low Level Driver

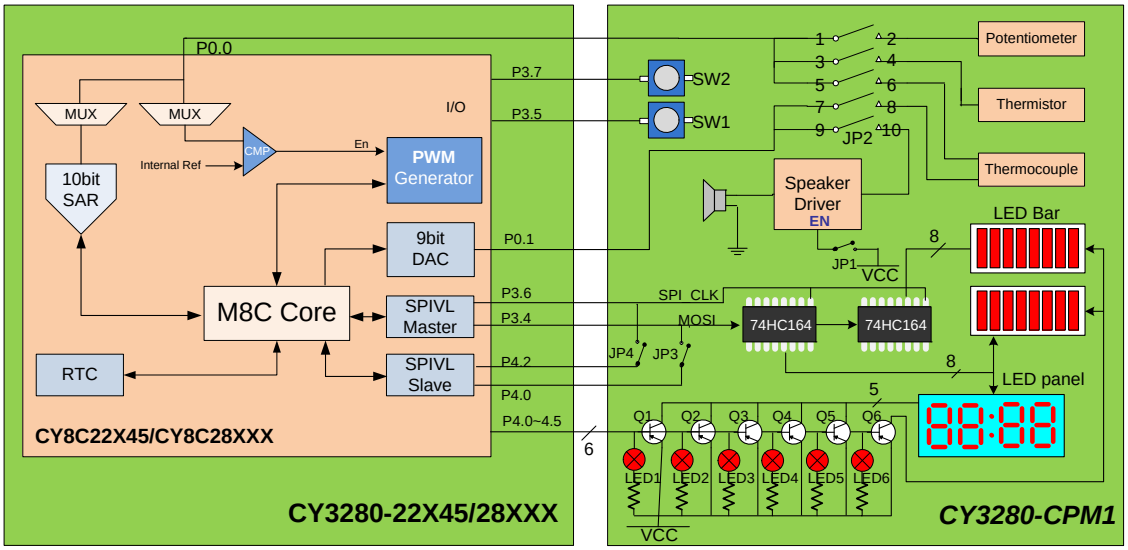
PSoC Designer automatically generates the source code of low level drivers for the UM employed in Device Editor. Then you can reference these drivers directly in the source code. Generally, the driver is a composite of *UM.asm*, *UM.h,* and *UM.inc*. Refer to the UM's data sheet for more information.

A. Appendix

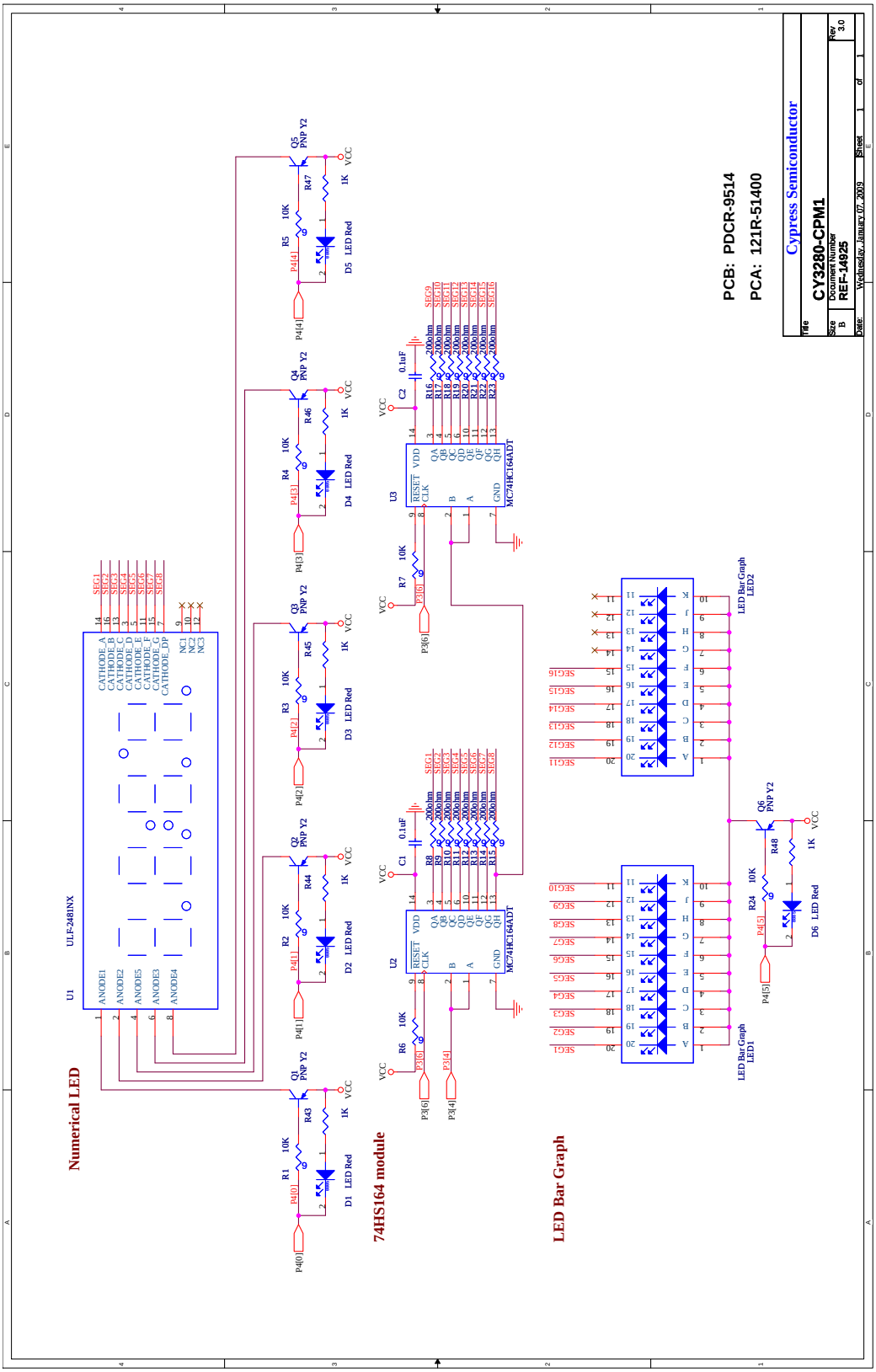
A.1 System Block Diagram

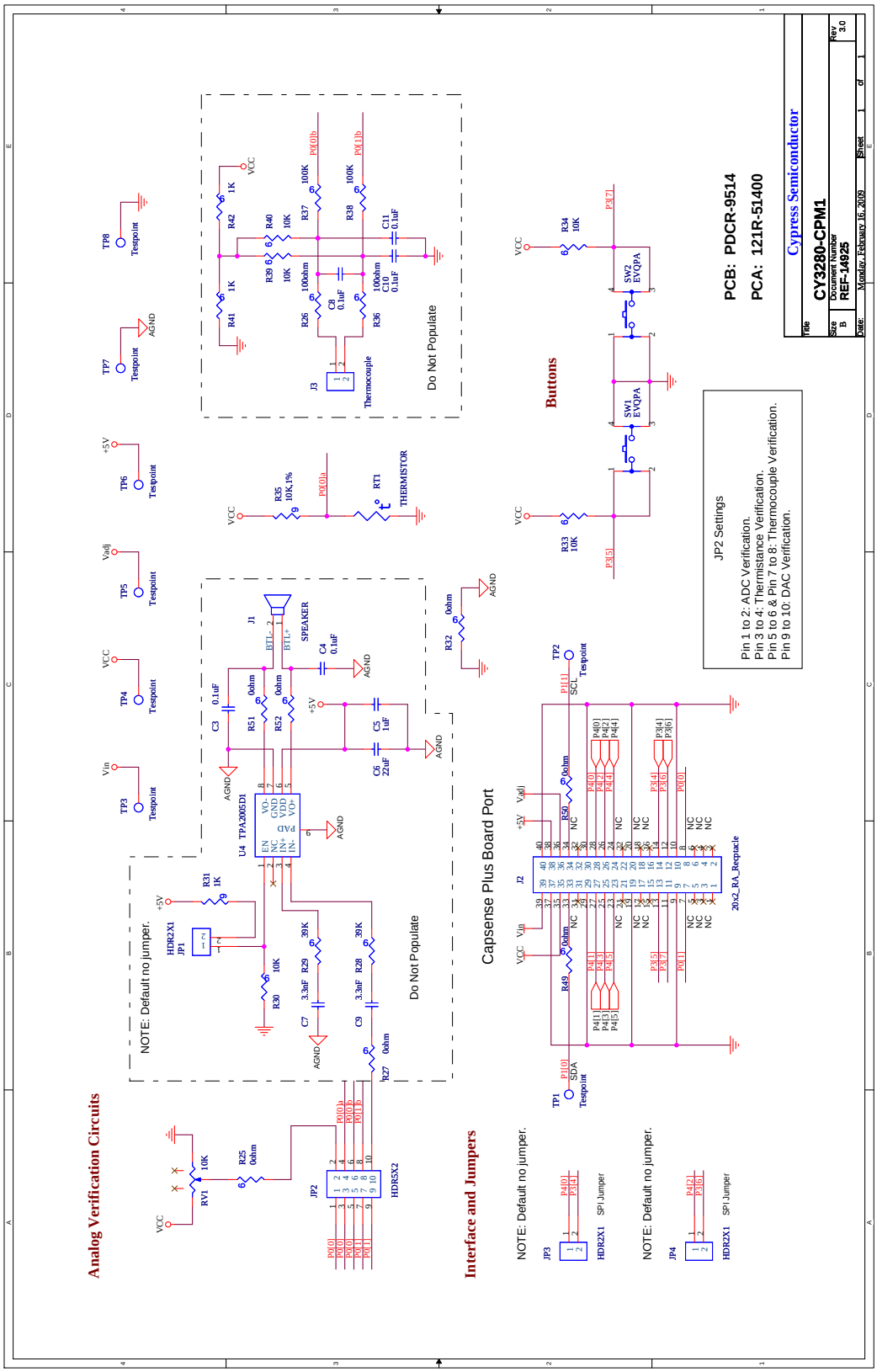
The CY3280-CPM1 board is a daughter board of the CY3280-22X45 and CY3280-28XXX Universal CapSense Controller Development Kits. You must use them together to study the advanced CapSensePLUS features provided by CY8C22X45/CY8C28XXX.

Figure A-1. CapSensePLUS System Structure Diagram

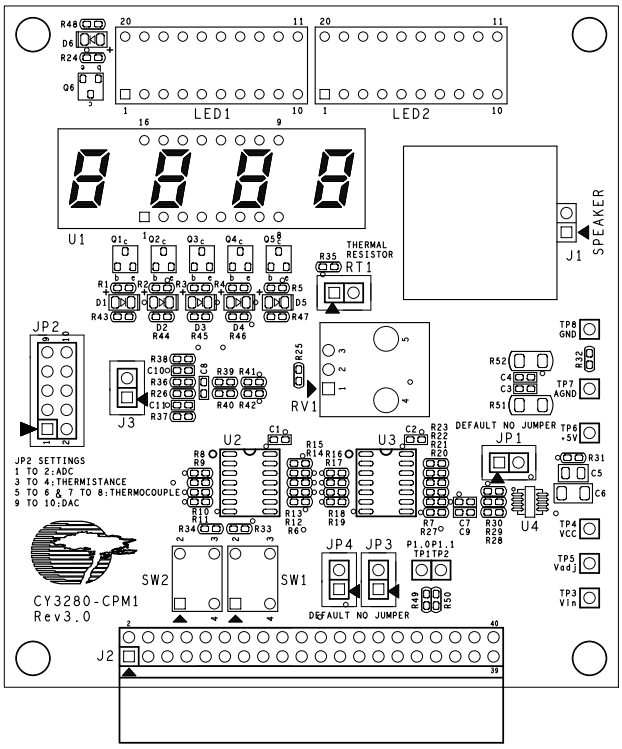


A.2 Schematic




Analog Verification Circuits
Interface and Jumpers
Capsense Plus Board Port
Buttons
Do Not Populate
NOTE: Default no jumper.
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
CY3280-CPM1
REF-14925
Monday, February 16, 2009

A.3 Top Silk Screen



A.4 Bill Of Material (BOM)

Item	Qty	Reference	Description
1	7	C1,C2,C3,C4,C8,C10,C11	CAP CER 0.10UF 25V X7R 10% 0603
2	1	C5	CAP CER 1.0UF 25V X7R 10% 0805
3	1	C6	CAP CER 22UF 16V X5R 20% 1206
4	2	C7,C9	CAP CER 3300PF 10V X7R 10% 0603
5	6	D1,D2,D3,D4,D5,D6	LED RED 635NM DIFF LENS 0805
6	3	JP1,JP3,JP4	CONN HEADER 2POS .100" STR TIN
7	1	JP2	CONN HEADER 10POS .100 STR TIN
8	1	J1	SPEAKER INTERFACE
9	1	J2	CONN HEADER 40POS .100" R/A TIN
10	1	J3	PROBE INTERFACE
11	2	LED1,LED2	LED BAR GRAPH 10-SEGMENT GREEN
12	6	Q1,Q2,Q3,Q4,Q5,Q6	TRANSISTOR SWITCHING PNP SOT-23
13	1	RT1	THERMISTOR NTC 10K OHM 5% RAD
14	1	RV1	POT 10K OHM 9MM VERT NO BUSHING
15	13	R1,R2,R3,R4,R5,R6,R7,R24, R30,R33,R34,R39,R40	RES 10K OHM 1/10W 5% 0603 SMD
16	16	R8,R9,R10,R11,R12,R13,R14,R15, R16,R17,R18,R19,R20, R21,R22,R23	RES 200 OHM 1/10W 5% 0603 SMD
17	5	R25,R27,R32,R49,R50	RES ZERO OHM 1/10W 5% 0603 SMD
18	2	R26,R36	RES 100 OHM 1/10W 5% 0603 SMD
19	2	R28,R29	RES 39K OHM 1/10W 5% 0603 SMD
20	9	R31,R41,R42,R43,R44,R45, R46,R47,R48	RES 1.0K OHM 1/10W 5% 0603 SMD
21	1	R35	RES 10.0K OHM 1/10W 1% 0603 SMD
22	2	R37,R38	RES 100K OHM 1/10W 5% 0603 SMD
23	2	R51,R52	RES ZERO OHM 1/4W 5% 1206 SMD
24	2	SW1,SW2	LT SWITCH 6MM H=5MM 130GF
25	8	TP1,TP2,TP3,TP4,TP5,TP6, TP7,TP8	TEST POINT PC MINI .040"D BLACK
26	1	U1	LED 7-SEG .4" 4DGT SUPER RED Common Anode
27	2	U2,U3	IC SHIFT REG 8BIT SER/PAR 14SOIC
28	1	U4	IC 1.1W CLASS-D AUDIO AMP 8-SON