SILICON LABORATORIES

AN73

Si3220/Si3225 System Demonstration Kit User's Guide

Introduction

This document describes the operation and construction of the Si3220/3225 demonstration system. The boards and software can be used together to simulate up to 16-channels of POTS (plain old telephone service). The 4-channel system demonstration kit part numbers are Si3220kit4-EVB and Si3225kit4-EVB.

Setup

To set up the system demonstration using the Si3220 or Si3225, stack the boards as illustrated in Figure 1. The setup (from bottom to top) consists of an 8-bit Microcontroller Board (SiLINKMP-EVB), a Dual ProSLIC Power Supply Board (SiLINKPS-EVB), and two Dual ProSLIC Evaluation Boards (Si3220DC0-EVB or Si3225DC0-EVB). Up to eight dual-channel evaluation boards (EVBs) can be stacked above the Microcontroller and Power Supply Boards to create up to a 16-channel POTS system. The required input voltage for use with the Si3220 system is 12 V at J1 of the Dual ProSLIC Power Supply Board. JP3 and JP4 need to be set on the Power Supply Board for the appropriate V_{BAT}; the system requires a V_{BAT} of –78 V. If the Si3225 is being used, a ringing signal should be connected at JP7 on the 8-bit Microcontroller Board. With the Si3220, no ringing signal is needed. Connecting a db-9 cable between J1 and a serial port of the PC allows access to a wide variety of software configurable features. This is done using a console window such as HyperTerminal, which is provided with the Microsoft® Windows® product family. When setting up a new connection in HyperTerminal, set the appropriate baud rate according to the microcontroller clock input frequency. The system demonstration kit is configured for a baud rate of 9600 bps. Connect telephones to the RJ-11 jacks (J1 and J11) on all Dual ProSLIC evaluation boards. The microcontroller demonstrates line card functionality similar to a PBX system. (Refer to AN68 on how to program the microcontroller with the 16-channel POTS software.)

Setup checklist:

1. Program Microcontroller. (See AN68.)
2. Switch Settings: S2-1, 2, 3 must be set for an appropriate PCLK. S2-4,5,6 must all be "Off."
3. S2-7 must be "On."
4. JP6 on the Microcontroller Board is in the V_{DD} off position.
5. Power settings: Jumpers on the power supply must be configured for a 3.3 V_{DD} and a –78 V_{BAT}.
6. Stack the boards as shown in Figure 1.
7. Connect telephones to the RJ-11 jacks.
8. Apply 12 V through the barrel jack of the power supply.
9. If the Si3225 is used, supply a ringing signal at JP7.
10. Connect db-9 cable from J1 to the com port of the PC.
11. Open the hyper terminal and set the appropriate com port and configure it to the following settings:
    - Baud rate to 9600 bps
    - Data bits: 8
    - Parity: none
    - Stop bits: 1
    - Flow control: Xon/Xoff

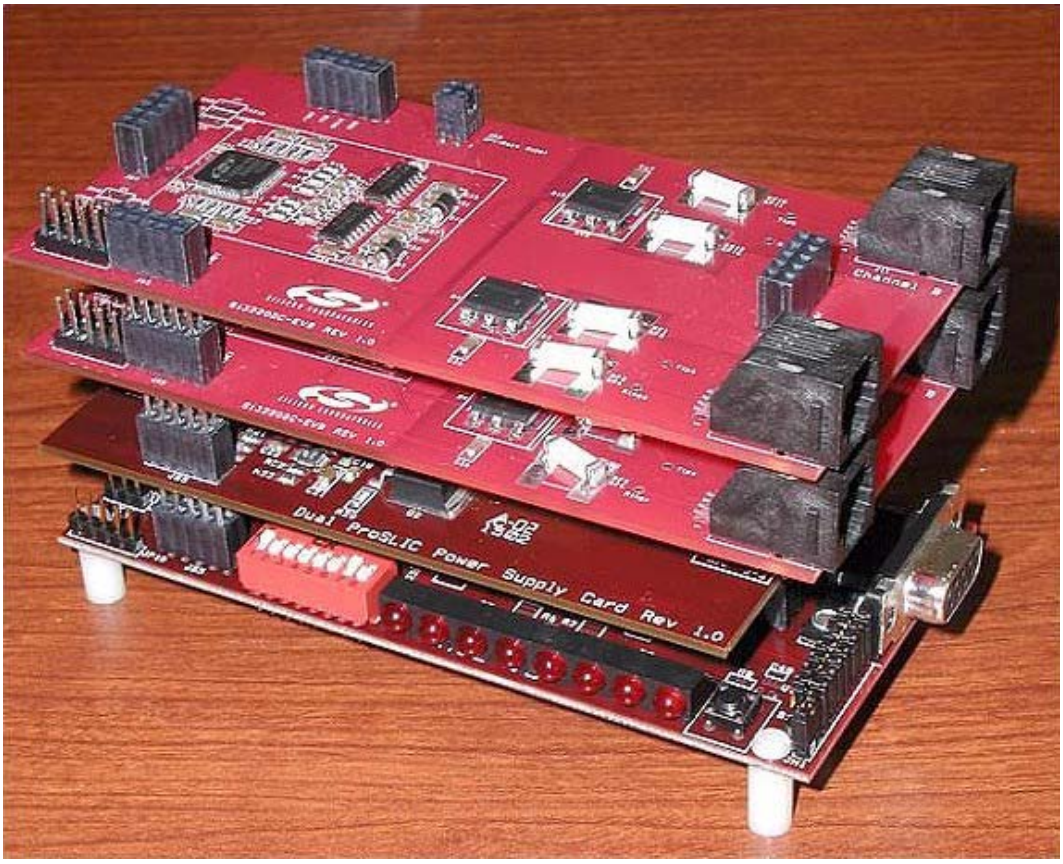

Figure 1. Four Channel Si3220 System Demonstration Setup

Operation

Once power is applied and software is running, the kit performs all basic telephone functionality. By picking up any phone, a user will hear a dial tone and can dial any of the other phones connected to the setup. The dialed phone will ring, and when answered, a normal phone conversation can take place between the two channels. The PBX is complete with busy tone, reorder tone, and congestion tone. Caller ID information is also displayed on the ringing phone if equipped with a caller ID display.

Console

The console provides a user-configurable menu. (See Figure 2.) Menu options include viewing/changing phone numbers and caller-ID information, changing between various country's tones, viewing/logging the state/events of the various channels, and reading and writing to Direct Registers and RAM. When the microcontroller is first powered on, the console lets the user know what channels are working. After initialization, the console notifies the user, and the PBX is ready to use. (The user is also notified when the LEDs on the board stop flashing.) When the menu is displayed, the user types the letter of a desired function followed by a carriage return. The console also allows assigning different phone numbers to various actions. By selecting the "Add/Change action of Phone Number" option, the user can assign phone numbers to different tones and events. Another option allows setting of the microcontroller clock frequency. After a reset, the software assumes 8.192 MHz. If a different MCU-CLK is being used, menu option "G" tells the microcontroller what clock speed is being applied; therefore, the software is able to set the MCU timing resistors correctly. If the software is not given the correct clock speed, certain call features will not work properly. There is also an option that allows the user to reset the default settings. This option changes all channel settings back to their original value. It is recommended that a user reset the defaults after programming the microcontroller or on first use. The last option allows the user to enter diagnostic mode. This allows the user to perform loop test measurements and find GR 909 fixed line faults.

Note: If the internal clock is not used for the GP32, set the correct frequency using the console.

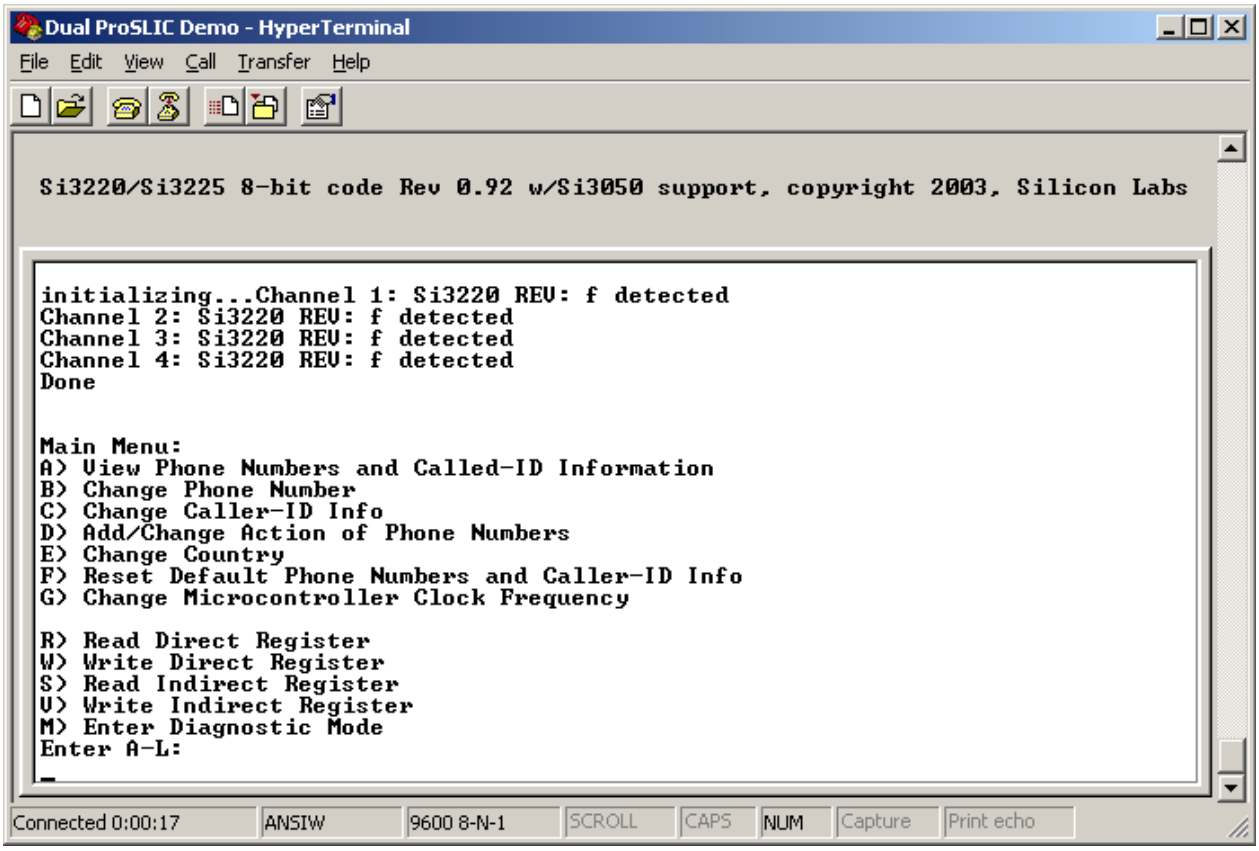

Figure 2. Menu

Software

The demo code provided is intended as a basis to create a PBX using the Si3220 or Si3225. Silicon Laboratories has two sets of demo code available. The PC version, described in ”AN75: Dual ProSLIC Software Interface”, polls interrupts to switch states, while the version provided for this setup stays idle unless awakened by the interrupt handler. The code was written in an interrupt-driven manner due to RAM and timing limitations. The software project can be opened and changed using the Metrowerks CodeWarrior™ Version 2.0 HC08 compiler.

Figure 3 gives a high-level block diagram of the layers in the software. Spi.c (associated header file: spi.h) is a low-level driver that communicates to the ProSLIC through the SPI port. Dual_i2o.c (associated header file: dual_io.h) is one layer up that sends the correct order of bytes for communication. Gr909.c, ISR.c, and SLIC.c are high-level drivers that demonstrate how to perform various common telephony tasks. (The associated header files are: SLIC.h, registers.h, SRAM.h, gr909.h.)

SLIC.c performs all basic necessary telephony functions. Table 1 explains the functions contained in SLIC.c. These are the most common necessary functions for any application.

GR909.c demonstrates how to perform GR 909 loop tests. The functions are essentially the same as those described in AN75 except they have been formatted to perform on an 8-bit microcontroller. Refer to AN75: “Dual ProSLIC Software Interface” and code comments for details on how these functions are implemented.

ISR.c contains the interrupt service routines of the program. Unlike the PC version explained in AN75 where the software polled for interrupts, the processor here goes into a low-power mode until it is awakened by the interrupt hardware. ISR.c processes the interrupt and sends the data back to the main program for the necessary action.

Additionally, two other files explain how to perform other telephony features. Caller.c demonstrates how to send caller-ID data and pulsedialing.c demonstrates how to detect dial pulses for pulse dialing. Code comments explain these algorithms in more detail.

The main program serves as a demo on how to create a PBX. The files used to accomplish the demo and console interface are explained in Table 2 on page 6. Figure 4 contains the basic PBX state machine.

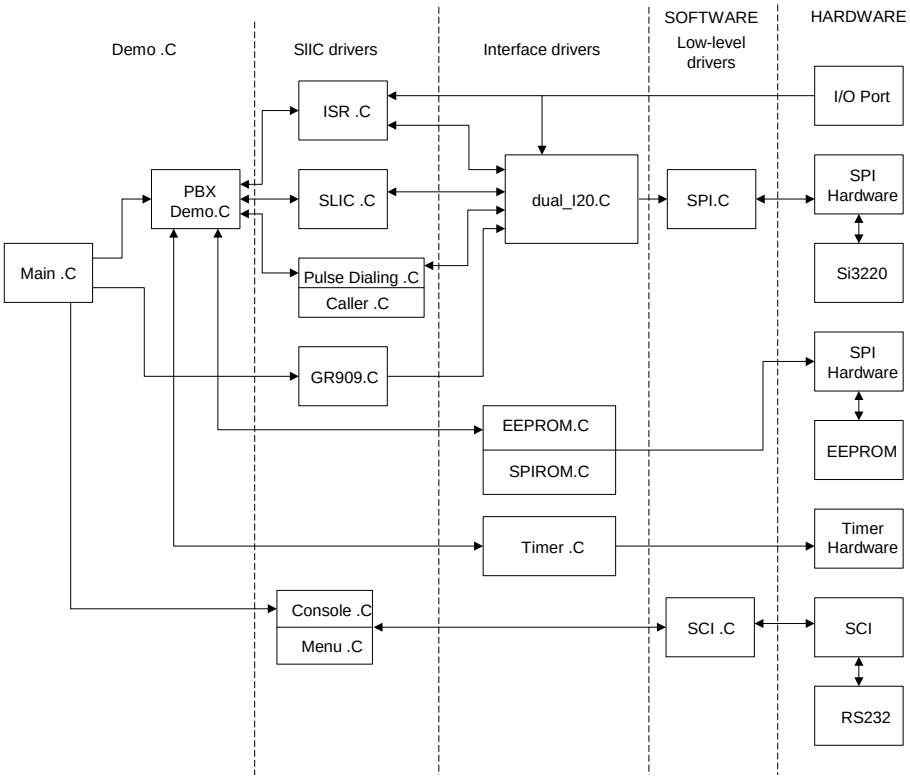

Figure 3. Software Block Diagram

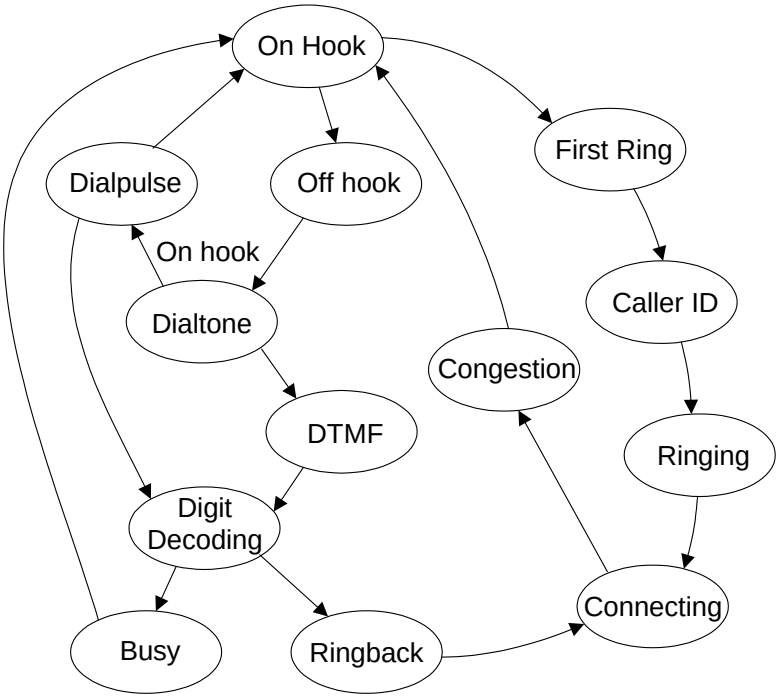

Figure 4. State Machine

Table 1. SLIC.c SLIC.h

unsigned char dualProSLIC_channelInit(void)	Initializes and calibrates the ProSLIC. This function must be run before any operation of the Si3220/Si3225. Initial RAM and register values are loaded from data structures elsewhere in this file. The function then performs the proper calibration routine. Refer to "AN58: Si3220/Si3225 Programmer's Guide" for the proper initialization procedure.
void activateRinging(void); void goActive(void); void goOpen(void); void goOHT(void);	These four functions change the LINEFEED (register 6) states of the Si3220/25. They switch between open, active, on-hook transmission, and ringing.
unsigned char get_HookStatus(void);	Returns value of the loop closure bit in the LCRRTP register (register 9). Returns 1 if off-hook and 0 if on-hook.
unsigned char get_DTMFdigit(void);	Returns DTMF digit value in TONDTMF (register 69).
void disableOscillators();	Turns off oscillators. Stops generated tones.
void breakConnection(unsigned char c1, unsigned char c2); void makeConnection(unsigned char c1, unsigned char c2);	Makes and breaks a direct PCM connection between 2 channels.
void waitForTheChannel(void)	Checks MSTRSTAT (register 3) to ensure the ProSLIC is up and running
void genTone(Oscillator *tone1, Oscillator *tone2)	Generates a tone with the internal oscillator.
void standardRinging(void)	Initializes ringing registers before ringing is activated.

Table 2. Demo Files

Files	Description
PBX Files	
Main.c	Initializes board and checks for channels.
DualDemo.c	DualDemo.c handles all PBX functionality (make/break connections, channel states, ringing, tones, etc.).
Countries.c/Countries.h	These files contain data structures of the coefficients for the tone registers for various countries. Console.c calls Countries.c when a user changes the country setting through the console. The function “changeptrs()” sets the tone pointers to the appropriate country.
Data Management Files **Note:** The microcontroller is limited in RAM space (512 bytes), so care must be taken when writing code to ensure data is managed properly.	
SLICRAM.c/SLICRAM.h	The actual channel data, which stores DTMF digits, hookstatus, ringcounts, etc., is not accessed often and therefore can be held external to the microcontroller. The software places this data in the test and diagnostics portion of the ProSLIC RAM space since these RAM locations (RAM address 125–162, excluding 155 and 156) are not used during normal operation of the Dual ProSLIC. Each channel has its own set of these ram locations; so, adding channels is not limited by RAM space in the microcontroller. The program only reads and writes the data as necessary. As data is needed, SLICRAM.c pulls it from the Dual ProSLIC and places it into the global “channelDat,” which is referenced by the global pointer “pCCData.” When the main program is done with the data, SLICRAM.c writes it back to the Dual ProSLIC.
EEPROM.c	The EEPROM contains the phone numbers and caller-ID information of the various channels. It also contains the phone numbers that perform other actions. The information is stored in 32 byte rows. The first 10 bytes of a row are the phone number for that channel. The next 16 are the caller-ID information or description of the number. The last two bytes in a row is the action that the phone number performs (call a phone, play a tone, etc.). The rest of the bytes are unused. EEPROM.c combines the drivers and data management functions of the EEPROM. See information below on EEPROM drivers.
Data.c/dual.h	Data.C and dual.h contain all the data structures and global variables used in the PBX portion of the program. The majority of the channel data is held in the RAM of the ProSLIC (see SLICRAM.c above). DualDemo.c uses “statePtr” to reference the state structure of the current channel being accessed. “channelState” is an array that holds all the state information of all the channels. Refer to the code for more details.

Table 2. Demo Files (Continued)

Files	Description
Drivers	
SPIROM.c/EEPROM.c	SPIROM contains all the functions to read and write from the EEPROM. Because the EEPROM has the ability to write an entire page at one time, the code does not use SPI.c which only sends a byte at one time. EEPROM.c has its own drivers for the EEPROM when interfacing for phone data. This makes it simple for the software to write and read multiple locations at one time when reading/writing channel data. “phone2PROM()” writes the entire default channel table to the EEPROM. Drivers to write individual parts of the data also exist in this file: (writeAction(), writeCallerInfo(), writephonenumber(), etc.). These are called upon when a user changes information using the console.
SCI.c /SCI.h	SCI.c sends and receives data from the SCI port, which communicates with the serial port of the PC. Various functions are written in SCI.c to allow sending and receiving data in various formats: characters, strings, decimal, hexadecimal, etc. The baud rate of the SCI is configured through the microcontroller SCI baud rate register (SCBR). At 8.096 MHz, SCBR is set to a divide by 13 for a 9600 bps baud rate. If the clock rate is changed, SCBR should be changed accordingly in the function “SCIinit()”. For example, If 4.9152 MHz is used, SCBR could be set to a divide by 2 for a baud rate of 38400 bps. Refer to the MC68HC908GP32 data book for information on how to set this register.
Timer.c/timer.h	Timer.c is the interface to the timer of the Dual ProSLIC. The amount of time to wait is sent to the function “sleep_uc()” in milliseconds. It halts the microcontroller for that many milliseconds.
6808.c/6808.h	These two files contain all the mnemonics of the various ports of the various MC68HC908GP32 microcontroller for easy access. 6808.c also contains the initialization functions to set all the ports configuration registers (port A-E, SCI, SPI, Timer, etc.) to appropriate values.
Console Files	
Console.c/console.h and menu.c/menu.h	The console is activated by an interrupt from the SCI receive hardware. Every time data is found, it adds it to the global string “stringinput.” When an [enter] is detected, it determines if the input was a valid response and calls the function “acceptstring()” to move on to the next menu/action.

Portability

The code can be ported to any microcontroller. To port the code to a different platform, all drivers need to be changed to support the new processor/microcontroller. Specifically, the following steps should be taken:

1. Replace SPI.C with a low-level driver to interface the chosen processor to send and receive bytes from the dual ProSLIC using the SPI bus.
2. Change the Interrupt() function to detect interrupts with the new processor.
3. Change “sleep_uc()” to suspend the new processor for a set number of ms.
4. If a RS-232 connection is desired, replace SCI.C with a low-level serial port driver.
5. Replace EEPROM drivers to match new processor’s SPI port.

Document Change List

Revision 0.1 to Revision 0.2

- Updated code description to describe latest release of demo code.
- Added Figure 3, “Software Block Diagram”.
- Added Figure 4, “State Machine”.
- Updated Table 1 to explain SLIC driver in more detail.
- Added “Appendix—Interfacing with the Si3050 DAA”

Revision 0.2 to Revision 0.3

- Updated "Software" on page 4.
- Removed Appendix.

Contact Information

Silicon Laboratories Inc.

4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: productinfo@silabs.com
Internet: www.silabs.com