JUMP START: IN-SYSTEM, FLASH-BASED PROGRAMMING FOR SILICON LABS' TIMING PRODUCTS

1. Introduction

Silicon Labs' timing products provide unparalleled performance, flexibility, and functionality, freeing designers to focus on other value-adding aspects of their designs. To take full advantage of the flexibility of these timing devices and optimize in-system performance, Silicon Labs provides numerous device controls. Computer-based configuration software provides the easiest method of configuring these timing devices. To maximize in-system potential, ease hardware debugging, and help jump-start the designer's timing path, Silicon Labs presents a hardware-friendly solution called JumpStart.

JumpStart provides a simple solution for in-system programming of the timing subsystem. It can be used for both on-board frequency plan development and manufacturing-level programming. JumpStart relies on a small form-factor, stand-alone microcontroller (MCU) and code that accepts frequency plans for Silicon Labs' timing devices. The MCU firmware is partitioned to allow easy editing of the device configuration. Once created, the device configuration is simply downloaded to the MCU's flash memory. The provided JumpStart reference design code then steps through the user-supplied configuration and programs the target timing device(s). Since the flash memory can be reconfigured as needed, hardware developers are free to iterate through as many design scenarios as desired. Once the final firmware is established, the MCU can be preconfigured at Silicon Labs with customized firmware. Therefore, the system configuration can be easily established during development without complicating the manufacturing flow.

The JumpStart reference design code provides serial port access to Silicon Labs' full lineup of timing clock devices, such as the DSPLL™-based Any-Frequency Precision Clocks (e.g., Si5326 and Si5368) as well as the MultiSynth based Any-Frequency, Any-Output Clock Generators (e.g., Si5338, Si5356). Since the serial bus standards allow connection to multiple devices, expanded configurations with many devices can be easily created. The reference design code is provided royalty-free and can be modified to support connections to as many devices as needed.

2. JumpStart Usage Model

1. Use the device evaluation board and its configuration software for the desired timing device to create a configuration and corresponding register map.

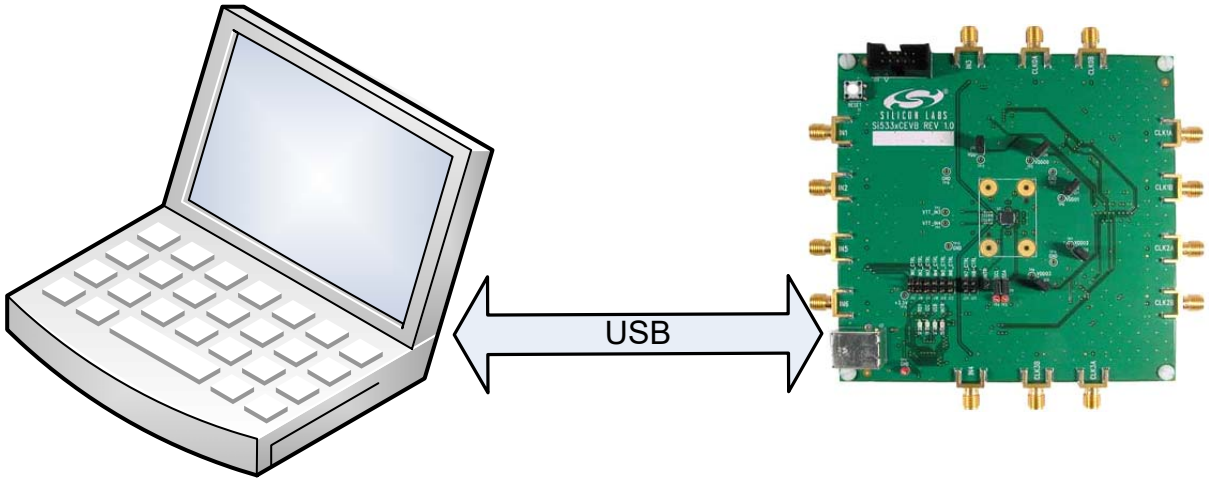

Figure 1. Step 1: Creating the Register Map

2. Using the Silicon Labs MCU Integrated Development Environment (IDE), compile the register map with the C-based MCU firmware, and download the firmware to the MCU.

Note: C2 is a Silicon Labs two-wire serial communication protocol for in-system programming and debugging of Silicon Labs MCUs. The USB debug adapter and C2 interface are included with the MCU development kits.

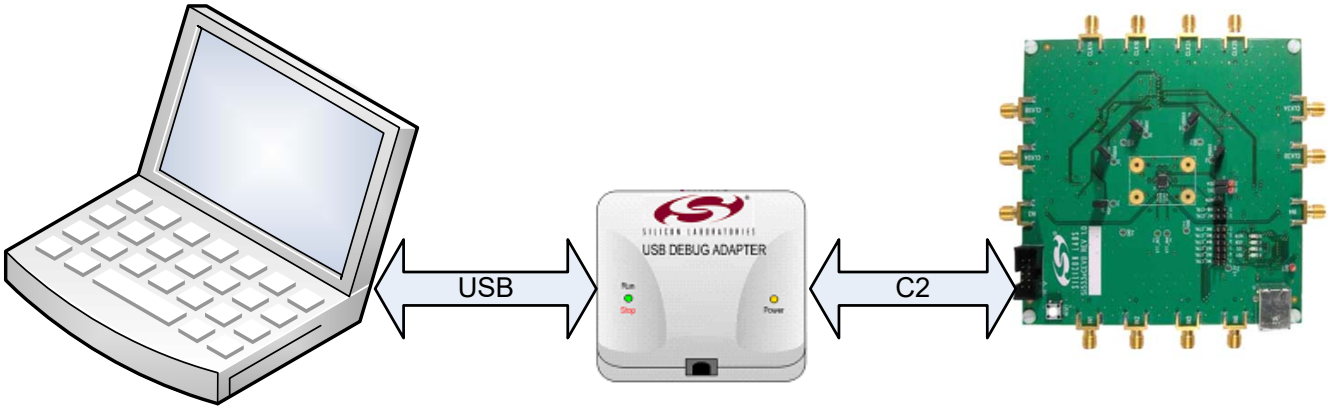

Figure 2. Step 2: Sending the Register Map to the MCU

3. After reset or powerup, the MCU will write the register map into the target timing device. The timing device is now fully configured. Silicon Labs' MCUs provide their own internal oscillator; so, the MCU can operate independently of the timing devices' outputs.

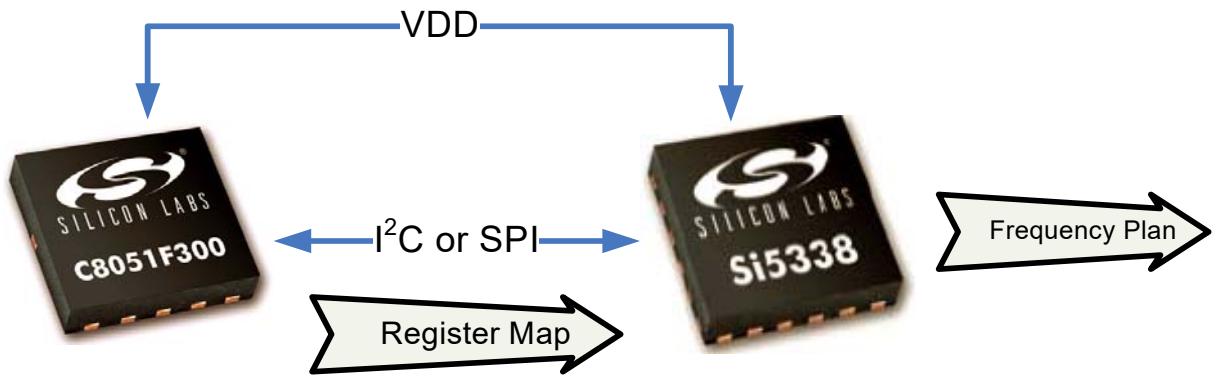

Figure 3. Step 3: Programming the Timing Device with the Register Map

4. Once the desired firmware has been set, different options may be used to program the MCUs for production. Contact Silicon Laboratories for more information, or refer to Silicon Laboratories “AN136: Production Programming Options for Silicon Labs Devices” for information on different options including other in-system programming methods with the C2 or JTAG interfaces and how to obtain pre-programmed MCUs. Factory-customized MCUs are assigned a unique orderable part number to simplify procurement.

3. Choosing the MCU

For JumpStart, consider these MCU features:

- Physical size
- Memory size
- Serial interface (I^2C or SPI)
- Digital pin count

It is possible to switch to one-time programmable (OTP) MCUs after the code has been developed in flash-based MCUs to save cost.

Use the online parametric search for choosing the best MCU. You can find this search tool on http://www.silabs.com. Select **Products** → **MCUs**, and click the **Parametric Search** link near the right side of the page.

4. Example Using the C8051F30

A Silicon Labs C8051F301 (F301) MCU (3 x 3 mm) and an Any-Frequency, Any-Output Clock Generator (Si5338N) connected via I^2C is configured for a gigabit Ethernet (GigE) application. The frequency plan required has a 25 MHz input with a crystal and two output frequencies of 25 MHz with 3.3 V CMOS on CLK0 and 125 MHz using 3.3 V LVDS on CLK1.

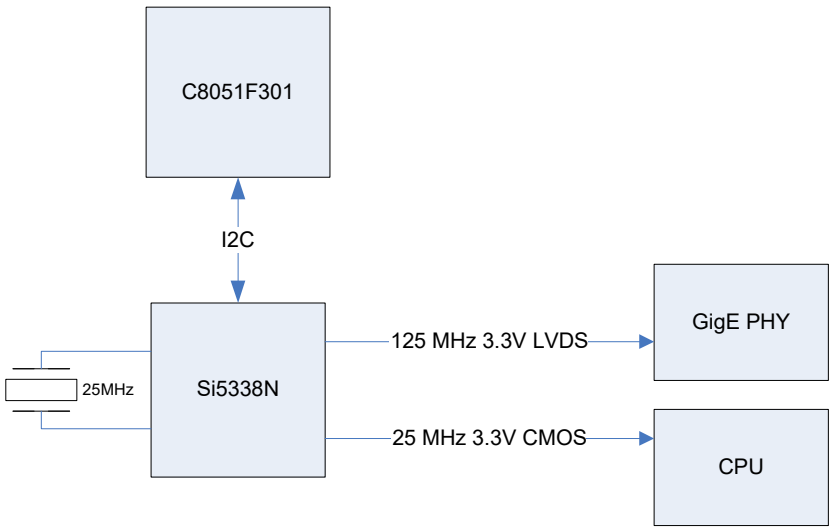

Figure 4. Example Block Diagram

4.1. Firmware Description

The full code listing of this example is available in the appendix as well as online at www.silabs.com. To download the source code, type “AN428” in the keyword search box, and download the AN428 Jumpstart Software from the result list.

1. The Si5338 data sheet has more details on writing a custom configuration to the Si5338's RAM. See section 3.5.3. The example code as seen in the appendix implements this procedure.
2. The F301 has eight digital I/O signals configured as open-drain with weak pull-ups. Initially, all the signals are set to logic 1.
3. The I^2C master state machine is adapted from Silicon Laboratories “AN141: SMBUS Communication for Small Form Factor Device Families” for use with the F301 (see “6.3 EEPROM Example” in AN141). SDA is on P0.0, and SCL is on P0.1 of the MCU using the SMBus. Timer 1 is configured to be the SMBus or I^2C clock source and is set to 400 kbps. Timer 2 is configured to be a timeout detection that the SMBus automatically uses during the state machine operation. Use of this timer is optional (see section 13.4.1 in the C8051F30x data sheet and section 3.1.2 of AN141 for more information).
4. The I^2C address of the Si5338 is set to the default of 0x70 (hexadecimal). The I^2C read and write functions will left-shift the address and fill in the I^2C R/W bit correctly.

4.2. Encoding the Register Map

For this example, the Si5338-EVB kit and ClockBuilder Pro program are used to develop a frequency plan configuration and to save the plan to a register map text file.

The goal is to take a line formatted like this from the register map text file format:

```
<address>,<data>h
```

to the register map array format in the MCU firmware like this:

```
{<address>,0x<data>,0x<mask>},
```

The address field represents the register address and is in decimal. The data field is the value to be sent to the register located at the corresponding register address. The mask denotes which bits are used and which are ignored in the register data. Any bits in the mask field that are a one are used, and any bits that are a zero are ignored. The data and mask fields are in hexadecimal. All three fields are eight bit integers with the range being 0 to 255 (0xFF).

After the formatting is complete, open the file register_map.h in the JumpStart project, and paste the register map data into the array declaration called Reg_Store between the curly braces. Update the NUM_REGS_MAX definition to be the number of elements in the array. Note that not all registers need to be written to the Si5338 depending on what features are used; see the Si5338 data sheet for more information on the register categories and masks.

4.3. Using the Header File Generation Function (Save C Code Header File)

Starting in version 2.36 of the ClockBuilder Pro software, there is a function to automatically generate the header file for use with the JumpStart project that includes the register map addresses, masks, and values. This function uses the current frequency plan information present in the Programmer and converts it to the format described in the previous section. To create this file, click on **Export** → **Register File** → **C Code Header File** in the software.

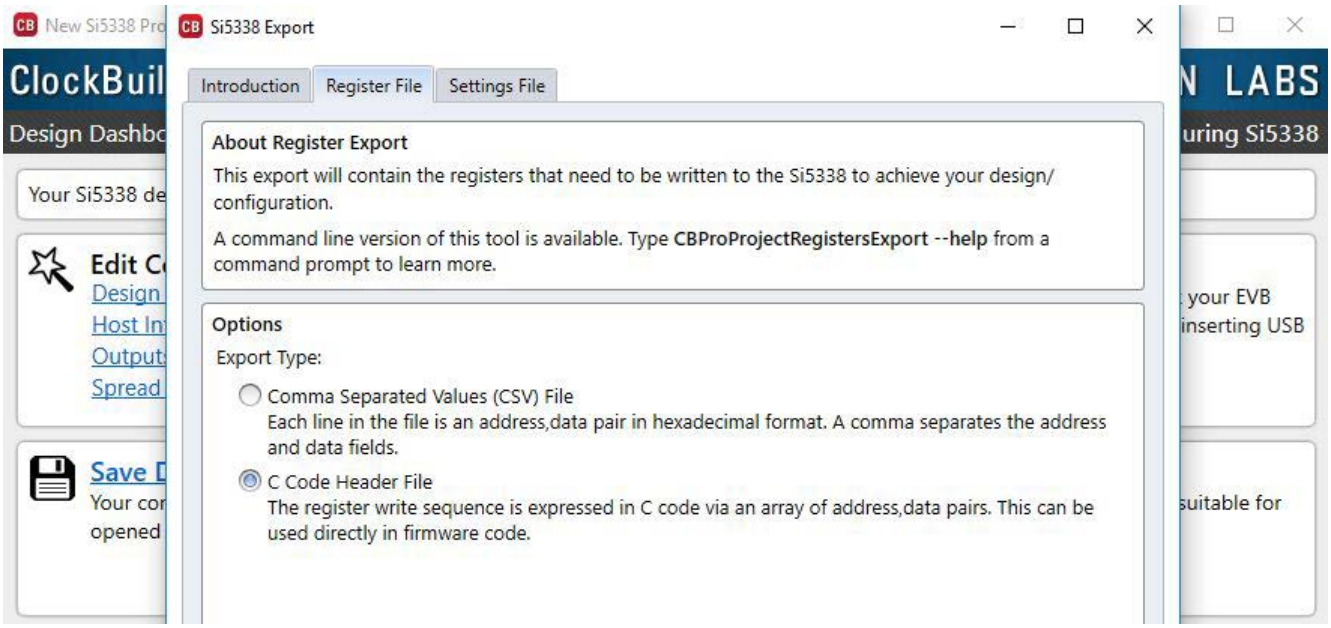

Figure 5. Selecting the Save C Code Header File Function

After it is generated, the header file should be compiled with the latest version of the JumpStart firmware project for use in the MCU, replacing whatever register_map.h file was there previously. The main Jumpstart code includes the header file already.

The generated header file contains the following:

- All the output driver, Multisynth, input configuration, spread spectrum, and phase and frequency adjustment register information
- Page bit and special register addressing (to deal with register locations greater than address 255)
- NUM_REGS_MAX calculation
- Correct C code syntax

Note that the header file is not optimized compared to one created manually, where a subset of the registers is used. For example, if spread spectrum and phase adjustment are not needed in a frequency plan, the generated file will use more flash memory space and take slightly longer to program the device since the registers related to those unused features are included in the file.

4.4. Register Map Header File

```
//Register map for use with AN428 (JumpStart)
//http://www.silabs.com/clocks
//Copyright 2012 Silicon Laboratories
//#BEGIN_HEADER
//Date = Monday, August 05, 2019 3:55 PM
//File version = 3
//Software Name = ClockBuilder Pro
//Software version = 2.35.8.405
//Software date = 8 1, 2019
//Chip = Si533x
//Part Number = Si533x
//#END_HEADER
//Input Frequency (MHz) = 25.000000000
//Input Type = Crystal
//P1 = 1
//Input Mux = XoClk
//FDBK Input Frequency (MHz) = 25.000000000
//FDBK Input Type = OFF
//P2 = 1
//FDBK Mux = NoClk
//PFD Input Frequency (MHz) = 25.000000000
//VCO Frequency (GHz) = 2.500000
//N = 100  (100.0000)
//Internal feedback enabled
//Output Clock 0
// Output Frequency (MHz) = 125.000000000
// Mux Selection = IDn
// MultiSynth = 20  (20.0000)
// R = 1
//Output Clock 1
// Output Frequency (MHz) = 25.000000000
// Mux Selection = IDn
// MultiSynth = 100  (100.0000)
// R = 1
```

```
//Output Clock 2
// Output is off
//Output Clock 3
// Output is off
//Driver 0
// Enabled
// Powered on
// Output voltage = 3.30
// Output type = 3.3V LVDS
// Output state when disabled = StopLow
//Driver 1
// Enabled
// Powered on
// Output voltage = 3.30
// Output type = 3.3V CMOS on A and B
// Output state when disabled = StopLow
//Driver 2
// Disabled
// Powered off
// Output voltage = 3.30
// Output type = 3.3V LVDS
// Output state when disabled = StopLow
//Driver 3
// Disabled
// Powered off
// Output voltage = 3.30
// Output type = 3.3V LVDS
// Output state when disabled = StopLow
//Clock 0 phase inc/dec step size (ns) = 0.000
//Clock 1 phase inc/dec step size (ns) = 0.000
//Clock 2 phase inc/dec step size (ns) = 0.000
//Clock 3 phase inc/dec step size (ns) = 0.000
//Phase increment and decrement pin control is off
//Frequency increment and decrement pin control is off
//Frequency increment and decrement is disabled
//Initial phase offset 0 (ns) = 0.000
```

```
//Initial phase offset 1 (ns) = 0.000
//Initial phase offset 2 (ns) = 0.000
//Initial phase offset 3 (ns) = 0.000
//SSC is disabled

#define NUM_REGS_MAX 350

typedef struct Reg_Data{
   unsigned char Reg_Addr;
   unsigned char Reg_Val;
   unsigned char Reg_Mask;
} Reg_Data;

Reg_Data const code Reg_Store[NUM_REGS_MAX] = {
{  0,0x00,0x00},
{  1,0x00,0x00},
{  2,0x00,0x00},
{  3,0x00,0x00},
{  4,0x00,0x00},
{  5,0x00,0x00},
{  6,0x08,0x1D},
{  7,0x00,0x00},
{  8,0x70,0x00},
{  9,0x0F,0x00},
{ 10,0x00,0x00},
{ 11,0x00,0x00},
{ 12,0x00,0x00},
{ 13,0x00,0x00},
{ 14,0x00,0x00},
{ 15,0x00,0x00},
{ 16,0x00,0x00},
{ 17,0x00,0x00},
{ 18,0x00,0x00},
{ 19,0x00,0x00},
{ 20,0x00,0x00},
{ 21,0x00,0x00},
```

```
{ 22,0x00,0x00},
{ 23,0x00,0x00},
{ 24,0x00,0x00},
{ 25,0x00,0x00},
{ 26,0x00,0x00},
{ 27,0x70,0x80},
{ 28,0x16,0xFF},
{ 29,0x90,0xFF},
{ 30,0xB0,0xFF},
{ 31,0xC0,0xFF},
{ 32,0xC0,0xFF},
{ 33,0xE3,0xFF},
{ 34,0xE3,0xFF},
{ 35,0x00,0xFF},
{ 36,0x06,0x1F},
{ 37,0x03,0x1F},
{ 38,0x00,0x1F},
{ 39,0x00,0x1F},
{ 40,0xE3,0xFF},
{ 41,0x0E,0x7F},
{ 42,0x23,0x3F},
{ 43,0x00,0x00},
{ 44,0x00,0x00},
{ 45,0x00,0xFF},
{ 46,0x00,0xFF},
{ 47,0x14,0x3F},
{ 48,0x3A,0xFF},
{ 49,0x00,0xFF},
{ 50,0xC4,0xFF},
{ 51,0x07,0xFF},
{ 52,0x10,0xFF},
{ 53,0x00,0xFF},
{ 54,0x08,0xFF},
{ 55,0x00,0xFF},
{ 56,0x00,0xFF},
{ 57,0x00,0xFF},
```

```
{ 58,0x00,0xFF},
{ 59,0x01,0xFF},
{ 60,0x00,0xFF},
{ 61,0x00,0xFF},
{ 62,0x00,0x3F},
{ 63,0x10,0xFF},
{ 64,0x00,0xFF},
{ 65,0x30,0xFF},
{ 66,0x00,0xFF},
{ 67,0x00,0xFF},
{ 68,0x00,0xFF},
{ 69,0x00,0xFF},
{ 70,0x01,0xFF},
{ 71,0x00,0xFF},
{ 72,0x00,0xFF},
{ 73,0x00,0x3F},
{ 74,0x10,0xFF},
{ 75,0x00,0xFF},
{ 76,0x00,0xFF},
{ 77,0x00,0xFF},
{ 78,0x00,0xFF},
{ 79,0x00,0xFF},
{ 80,0x00,0xFF},
{ 81,0x00,0xFF},
{ 82,0x00,0xFF},
{ 83,0x00,0xFF},
{ 84,0x00,0x3F},
{ 85,0x10,0xFF},
{ 86,0x00,0xFF},
{ 87,0x00,0xFF},
{ 88,0x00,0xFF},
{ 89,0x00,0xFF},
{ 90,0x00,0xFF},
{ 91,0x00,0xFF},
{ 92,0x00,0xFF},
{ 93,0x00,0xFF},
```

```
{ 94,0x00,0xFF},
{ 95,0x00,0x3F},
{ 96,0x10,0x00},
{ 97,0x00,0xFF},
{ 98,0x30,0xFF},
{ 99,0x00,0xFF},
{100,0x00,0xFF},
{101,0x00,0xFF},
{102,0x00,0xFF},
{103,0x01,0xFF},
{104,0x00,0xFF},
{105,0x00,0xFF},
{106,0x80,0xBF},
{107,0x00,0xFF},
{108,0x00,0xFF},
{109,0x00,0xFF},
{110,0x40,0xFF},
{111,0x00,0xFF},
{112,0x00,0xFF},
{113,0x00,0xFF},
{114,0x40,0xFF},
{115,0x00,0xFF},
{116,0x80,0xFF},
{117,0x00,0xFF},
{118,0x40,0xFF},
{119,0x00,0xFF},
{120,0x00,0xFF},
{121,0x00,0xFF},
{122,0x40,0xFF},
{123,0x00,0xFF},
{124,0x00,0xFF},
{125,0x00,0xFF},
{126,0x00,0xFF},
{127,0x00,0xFF},
{128,0x00,0xFF},
{129,0x00,0x0F},
```

```
{130,0x00,0x0F},
{131,0x00,0xFF},
{132,0x00,0xFF},
{133,0x00,0xFF},
{134,0x00,0xFF},
{135,0x00,0xFF},
{136,0x00,0xFF},
{137,0x00,0xFF},
{138,0x00,0xFF},
{139,0x00,0xFF},
{140,0x00,0xFF},
{141,0x00,0xFF},
{142,0x00,0xFF},
{143,0x00,0xFF},
{144,0x00,0xFF},
{145,0x00,0x00},
{146,0xFF,0x00},
{147,0x00,0x00},
{148,0x00,0x00},
{149,0x00,0x00},
{150,0x00,0x00},
{151,0x00,0x00},
{152,0x00,0xFF},
{153,0x00,0xFF},
{154,0x00,0xFF},
{155,0x00,0xFF},
{156,0x00,0xFF},
{157,0x00,0xFF},
{158,0x00,0x0F},
{159,0x00,0x0F},
{160,0x00,0xFF},
{161,0x00,0xFF},
{162,0x00,0xFF},
{163,0x00,0xFF},
{164,0x00,0xFF},
{165,0x00,0xFF},
```

```
{166,0x00,0xFF},
{167,0x00,0xFF},
{168,0x00,0xFF},
{169,0x00,0xFF},
{170,0x00,0xFF},
{171,0x00,0xFF},
{172,0x00,0xFF},
{173,0x00,0xFF},
{174,0x00,0xFF},
{175,0x00,0xFF},
{176,0x00,0xFF},
{177,0x00,0xFF},
{178,0x00,0xFF},
{179,0x00,0xFF},
{180,0x00,0xFF},
{181,0x00,0x0F},
{182,0x00,0xFF},
{183,0x00,0xFF},
{184,0x00,0xFF},
{185,0x00,0xFF},
{186,0x00,0xFF},
{187,0x00,0xFF},
{188,0x00,0xFF},
{189,0x00,0xFF},
{190,0x00,0xFF},
{191,0x00,0xFF},
{192,0x00,0xFF},
{193,0x00,0xFF},
{194,0x00,0xFF},
{195,0x00,0xFF},
{196,0x00,0xFF},
{197,0x00,0xFF},
{198,0x00,0xFF},
{199,0x00,0xFF},
{200,0x00,0xFF},
{201,0x00,0xFF},
```

```
{202,0x00,0xFF},
{203,0x00,0x0F},
{204,0x00,0xFF},
{205,0x00,0xFF},
{206,0x00,0xFF},
{207,0x00,0xFF},
{208,0x00,0xFF},
{209,0x00,0xFF},
{210,0x00,0xFF},
{211,0x00,0xFF},
{212,0x00,0xFF},
{213,0x00,0xFF},
{214,0x00,0xFF},
{215,0x00,0xFF},
{216,0x00,0xFF},
{217,0x00,0xFF},
{218,0x00,0x00},
{219,0x00,0x00},
{220,0x00,0x00},
{221,0x0D,0x00},
{222,0x00,0x00},
{223,0x00,0x00},
{224,0xF4,0x00},
{225,0xF0,0x00},
{226,0x00,0x00},
{227,0x00,0x00},
{228,0x00,0x00},
{229,0x00,0x00},
{231,0x00,0x00},
{232,0x00,0x00},
{233,0x00,0x00},
{234,0x00,0x00},
{235,0x00,0x00},
{236,0x00,0x00},
{237,0x00,0x00},
{238,0x14,0x00},
```

```
{239,0x00,0x00},
{240,0x00,0x00},
{242,0x02,0x02},
{243,0xF0,0x00},
{244,0x00,0x00},
{245,0x00,0x00},
{247,0x00,0x00},
{248,0x00,0x00},
{249,0xA8,0x00},
{250,0x00,0x00},
{251,0x84,0x00},
{252,0x00,0x00},
{253,0x00,0x00},
{254,0x00,0x00},
{255, 1, 0xFF}, // set page bit to 1
{  0,0x00,0x00},
{  1,0x00,0x00},
{  2,0x00,0x00},
{  3,0x00,0x00},
{  4,0x00,0x00},
{  5,0x00,0x00},
{  6,0x00,0x00},
{  7,0x00,0x00},
{  8,0x00,0x00},
{  9,0x00,0x00},
{ 10,0x00,0x00},
{ 11,0x00,0x00},
{ 12,0x00,0x00},
{ 13,0x00,0x00},
{ 14,0x00,0x00},
{ 15,0x00,0x00},
{ 16,0x00,0x00},
{ 17,0x01,0x00},
{ 18,0x00,0x00},
{ 19,0x00,0x00},
{ 20,0x90,0x00},
```

```
{ 21,0x31,0x00},
{ 22,0x00,0x00},
{ 23,0x00,0x00},
{ 24,0x01,0x00},
{ 25,0x00,0x00},
{ 26,0x00,0x00},
{ 27,0x00,0x00},
{ 28,0x00,0x00},
{ 29,0x00,0x00},
{ 30,0x00,0x00},
{ 31,0x00,0xFF},
{ 32,0x00,0xFF},
{ 33,0x01,0xFF},
{ 34,0x00,0xFF},
{ 35,0x00,0xFF},
{ 36,0x90,0xFF},
{ 37,0x31,0xFF},
{ 38,0x00,0xFF},
{ 39,0x00,0xFF},
{ 40,0x01,0xFF},
{ 41,0x00,0xFF},
{ 42,0x00,0xFF},
{ 43,0x00,0x0F},
{ 44,0x00,0x00},
{ 45,0x00,0x00},
{ 46,0x00,0x00},
{ 47,0x00,0xFF},
{ 48,0x00,0xFF},
{ 49,0x01,0xFF},
{ 50,0x00,0xFF},
{ 51,0x00,0xFF},
{ 52,0x90,0xFF},
{ 53,0x31,0xFF},
{ 54,0x00,0xFF},
{ 55,0x00,0xFF},
{ 56,0x01,0xFF},
```

```
{ 57,0x00,0xFF},
{ 58,0x00,0xFF},
{ 59,0x00,0x0F},
{ 60,0x00,0x00},
{ 61,0x00,0x00},
{ 62,0x00,0x00},
{ 63,0x00,0xFF},
{ 64,0x00,0xFF},
{ 65,0x01,0xFF},
{ 66,0x00,0xFF},
{ 67,0x00,0xFF},
{ 68,0x90,0xFF},
{ 69,0x31,0xFF},
{ 70,0x00,0xFF},
{ 71,0x00,0xFF},
{ 72,0x01,0xFF},
{ 73,0x00,0xFF},
{ 74,0x00,0xFF},
{ 75,0x00,0x0F},
{ 76,0x00,0x00},
{ 77,0x00,0x00},
{ 78,0x00,0x00},
{ 79,0x00,0xFF},
{ 80,0x00,0xFF},
{ 81,0x00,0xFF},
{ 82,0x00,0xFF},
{ 83,0x00,0xFF},
{ 84,0x90,0xFF},
{ 85,0x31,0xFF},
{ 86,0x00,0xFF},
{ 87,0x00,0xFF},
{ 88,0x01,0xFF},
{ 89,0x00,0xFF},
{ 90,0x00,0xFF},
{ 91,0x00,0x0F},
{ 92,0x00,0x00},
```

```
{ 93,0x00,0x00},
{ 94,0x00,0x00},
{255, 0, 0xFF} }; // set page bit to 0
//End of file
```

4.5. Programming the Si5338

At start-up, the MCU and the Si5338 will power-up simultaneously. Ensure that the input crystal frequency is present when the MCU and Si5338 are powered on. The MCU will initialize faster than the Si5338; so, the firmware has delay added to pause the MCU for about 12 ms. After that, the MCU will increment through the Reg_Store array. It will perform an I^2C write of the register data when the mask is a 0xFF, and it will perform a read-modify-write when the mask is something other than 0xFF. For more information, see " Appendix: JumpStart Program Using the C8051F301".

4.6. Requirements

- C8051F300DK development kit, which includes the Silabs IDE, Keil evaluation compiler, and USB Debug Adapter
- Si5338-EVB kit
- ClockBuilder Pro

4.7. Recommended Documents

- C8051F30x Data Sheet
- Si5338 Data Sheet
- AN141: SMBUS Communication for Small Form Factor Device Families
- C8051F30x Development Kit User's Guide
- Si5330/34/38 Evaluation Board User's Guide

5. Enhancements

The following descriptions provide ideas for modifying or expanding the programming technique described above.

- **Customizing the Si5338**
  The Si5338 has non-volatile memory (NVM), which can customize the start-up of the Si5338. Adding an MCU to program the custom Si5338 after it boots provides access to other features in the device, such as spread spectrum or frequency adjust on the output clocks. In addition, fewer registers will have to be programmed, which uses less memory in the MCU. Contact Silicon Laboratories for more information on custom Si5338s.
- **MCU-Controlled Power Supplies**
  The MCU in this application can control a power supply circuit to power the timing device's VDD. The MCU can send commands over the serial bus to turn on the power supplies after the MCU boots.
- **Multiple Plans**
  Depending on the MCU chosen, there may be extra digital I/O signals available on the MCU. These signals can be defined to select different register maps to load into the target device at power-up.
- **Multiple Slave Devices**
  The MCU can be used to send individual register maps to multiple slave (timing) devices using the same serial bus (I^2C or SPI). In the case of I^2C, the correct I^2C slave address must be used when addressing the slave device and correspond in the firmware with the desired register map. For SPI's slave select (SS) line, the extra digital I/O pins on the MCU can be used, or an external MUX can be added, depending on the number of slave devices.

6. Conclusion

The JumpStart model allows system designers to reconfigure Silicon Labs' timing devices as needed during hardware development without sacrificing manufacturability. The example given in this document demonstrates the ease with which configurations can be made and changed.

APPENDIX: JUMPSTART PROGRAM USING THE C8051F301

```
//-----------------------------------------------------------------------------
// F300_JumpStart.c
//-----------------------------------------------------------------------------
// Copyright 2010 Silicon Laboratories, Inc.
// http://www.silabs.com
//
// Program Description:
//
// - Interrupt-driven SMBus implementation
// - Only master states defined (no slave or arbitration)
// - Timer1 used as SMBus clock source
// - Timer2 used by SMBus for SCL low timeout detection
// - SCL frequency defined by <SMB_FREQUENCY> constant
// - Address for the slave I2C device is defined as SLAVE_ADDR
// - ARBLOST support included
// - Pinout:
//    P0.0 -> SDA (SMBus)
//    P0.1 -> SCL (SMBus)
//    all other port pins unused
//
//
// Target:         C8051F301
// Tool chain:     Keil 7.20
//
// Release 1.0
//    -Initial Revision (ACA)
//    -June 2009
//
// Release 1.1
//    - Moved the I2C address to this file from register_map.h
//    - This project works with the new register_map.h file generation
//      function in the Multisynth Clock Programmer
//
// Release 1.2 (August 2010)
//    - Added test for when the mask = 0x00 to skip that register access
```

```
//    - Updated Si5338 register mask list in register_map.h
//
// Release 1.3 (September 2010)
//    - Updated the register_map.h and main() to reflect the Si5338
//      datasheet updates
//
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <compiler_defs.h>
#include <c8051F300_defs.h>           // SFR declarations
#include <register_map.h>

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------



// Device address (7 bits) for slave target
//  0x70 is default for the Si5338
#define  SLAVE_ADDR      0x70

#define  SYSCLK            24500000       // System clock frequency in Hz

#define  SMB_FREQUENCY     400000         // Target SCL clock rate
                                          // Can be 100kHz or 400kHz

#define  WRITE             0x00           // SMBus WRITE command
#define  READ              0x01           // SMBus READ command

// Status vector - top 4 bits only
#define  SMB_MTSTA        0xE0            // (MT) start transmitted
#define  SMB_MTDB         0xC0            // (MT) data byte transmitted
#define  SMB_MRDB         0x80            // (MR) data byte received
// End status vector definition
```

```
//#define SI5338_DELAY 4800  //2ms
//#define SI5338_DELAY 24000 //10ms
#define SI5338_DELAY 28800 //12ms

#define LOCK_MASK 0x15
#define LOS_MASK 0x04


//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------


U8* pSMB_DATA_IN;               // Global pointer for SMBus data
                                // All receive data is written here

U8 SMB_SINGLEBYTE_OUT;          // Global holder for single byte writes.

U8* pSMB_DATA_OUT;              // Global pointer for SMBus data.
                                // All transmit data is read from here

U8 SMB_DATA_LEN;                // Global holder for number of bytes
                                // to send or receive in the current
                                // SMBus transfer.

U8 WORD_ADDR;                   // Global holder for the  word
                                // address that will be accessed in
                                // the next transfer

U8 TARGET;                      // Target SMBus slave address

volatile bit SMB_BUSY = 0;      // Software flag to indicate when the
                                // I2C_ByteRead() or
                                // I2C_ByteWrite()
```

```
                                // functions have claimed the SMBus

bit SMB_RW;                     // Software flag to indicate the
                                // direction of the current transfer

bit SMB_SENDWORDADDR;           // When set, this flag causes the ISR
                                // to send the 8-bit <WORD_ADDR>
                                // after sending the slave address.


bit SMB_RANDOMREAD;             // When set, this flag causes the ISR
                                // to send a START signal after sending
                                // the word address.
                                // The ISR handles this
                                // switchover if the <SMB_RANDOMREAD>
                                // bit is set.

bit SMB_ACKPOLL;                // When set, this flag causes the ISR
                                // to send a repeated START until the
                                // slave has acknowledged its address

SBIT(SDA, SFR_P0, 0);           // SMBus on P0.0
SBIT(SCL, SFR_P0, 1);           // and P0.1

SBIT (P0_5, SFR_P0, 5);
SBIT (P0_6, SFR_P0, 6);

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------

void SMBus_Init (void);
void Timer1_Init (void);
void Timer2_Init (void);
void Port_Init (void);
INTERRUPT_PROTO(SMBus_ISR, INTERRUPT_SMBUS0);
```

```
INTERRUPT_PROTO(Timer2_ISR, INTERRUPT_TIMER2);
void I2C_ByteWrite (U8 addr, U8 dat);
U8 I2C_ByteRead (U8 addr);


//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void){
   U16 counter;
   U8 curr_chip_val, clear_curr_val, clear_new_val, combined, reg;
   Reg_Data curr;

   U8 i;                          // Temporary counter variable used in for loops
   PCA0MD &= ~0x40;               // WDTE = 0 (disable watchdog timer)
   OSCICN |= 0x03;                // Configure internal oscillator for
                                  // its maximum frequency (24.5 Mhz)

   // If slave is holding SDA low because of an improper SMBus reset or error
   while(!SDA)
   {
      // Provide clock pulses to allow the slave to advance out
      // of its current state. This will allow it to release SDA.
      XBR1 = 0x40;                     // Enable Crossbar
      SCL = 0;                         // Drive the clock low
      for(i = 0; i < 255; i++);        // Hold the clock low
      SCL = 1;                         // Release the clock
      while(!SCL);                     // Wait for open-drain
                                       // clock output to rise
      for(i = 0; i < 10; i++);         // Hold the clock high
      XBR1 = 0x00;                     // Disable Crossbar
   }

   Port_Init ();                       // Initialize Crossbar and GPIO

   CKCON = 0x10;                       // Timer 1 is sysclk
```

```
                                    // Timer 2 is sysclk/12 (see TMR2CN)

   Timer1_Init ();                  // Configure Timer1 for use as SMBus
                                    // clock source

   Timer2_Init ();                  // Configure Timer2 for use with SMBus
                                    // low timeout detect

   SMBus_Init ();                   // Configure and enable SMBus


   EIE1  |= 0x01;                // Enable SMBus interrupt
   EA     = 1;                   // Global interrupt enable****MUST BE LAST****

   //---------------------------------------------------------------
   // See Si5338 datasheet Figure 9 for more details on this procedure

   // delay added to wait for Si5338 to be ready to communicate
   // after turning on
   counter = 0;
   while(counter < SI5338_DELAY) { counter++; }

   I2C_ByteWrite(230, 0x10);                  //OEB_ALL = 1
   I2C_ByteWrite(241, 0xE5);                  //DIS_LOL = 1

   //for all the register values in the Reg_Store array
   //get each value and mask and apply it to the Si5338
   for(counter=0; counter<NUM_REGS_MAX; counter++){

      curr = Reg_Store[counter];

      if(curr.Reg_Mask != 0x00) {

         if(curr.Reg_Mask == 0xFF) {
            // do a write transaction only
            // since the mask is all ones
```

```
            I2C_ByteWrite(curr.Reg_Addr, curr.Reg_Val);

         } else {
            //do a read-modify-write
            curr_chip_val = I2C_ByteRead(curr.Reg_Addr);
            clear_curr_val = curr_chip_val & ~curr.Reg_Mask;
            clear_new_val = curr.Reg_Val & curr.Reg_Mask;
            combined = clear_new_val | clear_curr_val;
            I2C_ByteWrite(curr.Reg_Addr, combined);
         }
      }
   }

   // check LOS alarm for the xtal input
   // on IN1 and IN2 (and IN3 if necessary) -
   // change this mask if using inputs on IN4, IN5, IN6
   reg = I2C_ByteRead(218) & LOS_MASK;
   while(reg != 0){
      reg = I2C_ByteRead(218) & LOS_MASK;
   }

   I2C_ByteWrite(49, I2C_ByteRead(49) & 0x7F); //FCAL_OVRD_EN = 0
   I2C_ByteWrite(246, 2);                      //soft reset
   I2C_ByteWrite(241, 0x65);                   //DIS_LOL = 0

   // wait for Si5338 to be ready after calibration (ie, soft reset)
   counter = 0;
   while(counter < SI5338_DELAY) { counter++; }
   counter = 0;
   while(counter < SI5338_DELAY) { counter++; }

   //make sure the device locked by checking PLL_LOL and SYS_CAL
   reg = I2C_ByteRead(218) & LOCK_MASK;
   while(reg != 0){
      reg = I2C_ByteRead(218) & LOCK_MASK;
   }
```

```
   //copy FCAL values
   I2C_ByteWrite(45, I2C_ByteRead(235));
   I2C_ByteWrite(46, I2C_ByteRead(236));
   // clear bits 0 and 1 from 47 and
   // combine with bits 0 and 1 from 237
   reg = (I2C_ByteRead(47) & 0xFC) | (I2C_ByteRead(237) & 3);
   I2C_ByteWrite(47, reg);

   I2C_ByteWrite(49, I2C_ByteRead(49) | 0x80); // FCAL_OVRD_EN = 1
   I2C_ByteWrite(230, 0x00);                   // OEB_ALL = 0

   //-----------------------------------------------------------

   // wait forever
   while(1); //comment this out if the power-down option is used below

   /*
   // power down the MCU
   while(SMB_BUSY);  // wait until the I2C transactions are complete
   EA = 0;           // turn off interrupts
   SMB0CF &= ~0x80;  // turn off SMBus
   RSTSRC = 0x00;    // turn off the MCD reset source
   PCON = 0x02;      // stop the MCU - only a reset will wake it up
   */
}


//-------------------------------------------------------------------------------
// Initialization Routines
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
// SMBus_Init
//-------------------------------------------------------------------------------
//
```

```
// Return Value : None
// Parameters   : None
//
// The SMBus peripheral is configured as follows:
// - SMBus enabled
// - Slave mode disabled
// - Timer1 used as clock source. The maximum SCL frequency will be
//   approximately 1/3 the Timer1 overflow rate
// - Setup and hold time extensions enabled
// - Free and SCL low timeout detection enabled
//
void SMBus_Init (void)
{
   SMB0CF = 0x5D;                      // Use Timer1 overflows as SMBus clock
                                       // source;
                                       // Disable slave mode;
                                       // Enable setup & hold time extensions;
                                       // Enable SMBus Free timeout detect;
                                       // Enable SCL low timeout detect;

   SMB0CF |= 0x80;                     // Enable SMBus;
}


//-----------------------------------------------------------------------------
// Timer1_Init
//-----------------------------------------------------------------------------
//
// Return Value : None
// Parameters   : None
//
// Timer1 is configured as the SMBus clock source as follows:
// - Timer1 in 8-bit auto-reload mode
// - SYSCLK / 12 as Timer1 clock source
// - Timer1 overflow rate => 3 * SMB_FREQUENCY
// - The maximum SCL clock rate will be ~1/3 the Timer1 overflow rate
```

```
// - Timer1 enabled
void Timer1_Init (void)
{
// Make sure the Timer can produce the appropriate frequency in 8-bit mode
// Supported SMBus Frequencies range from 10kHz to 100kHz.  The CKCON register
// settings may need to change for frequencies outside this range.
   TMOD = 0x20;                           // Timer1 in 8-bit auto-reload mode
   TH1 = 0xFF - (SYSCLK/SMB_FREQUENCY/3) + 1; // 100kHz or 400kHz for SCL
   TL1 = TH1;                                 // Init Timer1
   TR1 = 1;                                   // Timer1 enabled
}


//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// Return Value : None
// Parameters   : None
//
// Timer2 configured for SCL low timeout detection as
// follows:
// - Timer2 in 16-bit auto-reload mode
// - SYSCLK/12 as Timer2 clock source
void Timer2_Init (void)
{
   TMR2CN = 0x00;                     // Timer2 configured for 16-bit auto-
                                      // reload, low-byte interrupt disabled
                                      // Timer2 uses SYSCLK/12 (see CKCON)

   TMR2RLL = 0x5F;
   TMR2RLH = 0x88;
   TMR2L = TMR2RLL;
   TMR2H = TMR2RLH;
   TF2LEN = 0;
   TF2H = 0;
```

```
   TF2L = 0;
   ET2  = 1;                           // Timer2 interrupt enable
   TR2 = 1;                            // Start Timer2
}

//-----------------------------------------------------------------------------
// Port_Init
//-----------------------------------------------------------------------------
//
// Return Value : None
// Parameters   : None
//
// Configure the Crossbar and GPIO ports.
//
// P0.0   digital   open-drain    SMBus SDA
// P0.1   digital   open-drain    SMBus SCL
//
// all other port pins unused
//
// Note: If the SMBus is moved, the SCL and SDA sbit declarations must also
// be adjusted.
//
void Port_Init (void)
{
   P0MDOUT = 0x00;                 // All P0 pins open-drain output

   XBR1 = 0x04;                    // Enable SMBus
   XBR2 = 0x40;                    // Enable crossbar and weak pull-ups
   P0   = 0xFF;                    // set all the outputs to logic 1 explicitly

}



//-----------------------------------------------------------------------------
// SMBus Interrupt Service Routine (ISR)
//-----------------------------------------------------------------------------
```

```
//
// SMBus ISR state machine
// - Master only implementation - no slave or arbitration states defined
// - All incoming data is written starting at the global pointer <pSMB_DATA_IN>
// - All outgoing data is read from the global pointer <pSMB_DATA_OUT>
//
INTERRUPT(SMBus_ISR, INTERRUPT_SMBUS0)
{
   bit FAIL = 0;                       // Used by the ISR to flag failed
                                       // transfers

   static char i;                      // Used by the ISR to count the
                                       // number of data bytes sent or
                                       // received

   static bit SEND_START = 0;          // Send a start

   switch (SMB0CN & 0xF0)              // Status vector
   {
      // Master Transmitter/Receiver: START condition transmitted.
      case SMB_MTSTA:
         SMB0DAT = TARGET;             // Load address of the target slave
         SMB0DAT &= 0xFE;              // Clear the LSB of the address for the
                                       // R/W bit
         SMB0DAT |= SMB_RW;            // Load R/W bit
         STA = 0;                      // Manually clear START bit
         i = 0;                        // Reset data byte counter
         break;

      // Master Transmitter: Data byte (or Slave Address) transmitted
      case SMB_MTDB:
         if (ACK)                      // Slave Address or Data Byte
         {                             // Acknowledged?
            if (SEND_START)
            {
               STA = 1;
```

```
      SEND_START = 0;
      break;
   }
   if(SMB_SENDWORDADDR)       // Are we sending the word address?
   {
      SMB_SENDWORDADDR = 0;   // Clear flag
      SMB0DAT = WORD_ADDR;    // Send word address

      if (SMB_RANDOMREAD)
      {
         SEND_START = 1;      // Send a START after the next ACK cycle
         SMB_RW = READ;
      }

      break;
   }

   if (SMB_RW==WRITE)         // Is this transfer a WRITE?
   {

      if (i < SMB_DATA_LEN)   // Is there data to send?
      {
         // send data byte
         SMB0DAT = *pSMB_DATA_OUT;

         // increment data out pointer
         pSMB_DATA_OUT++;

         // increment number of bytes sent
         i++;
      }
      else
      {
        STO = 1;              // Set STO to terminte transfer
        SMB_BUSY = 0;         // Clear software busy flag
      }
```

```
         }
         else {}                    // If this transfer is a READ,
                                    // then take no action. Slave
                                    // address was transmitted. A
                                    // separate 'case' is defined
                                    // for data byte recieved.
      }
      else                          // If slave NACK,
      {
         if(SMB_ACKPOLL)
         {
            STA = 1;                // Restart transfer
         }
         else
         {
            FAIL = 1;               // Indicate failed transfer
         }                          // and handle at end of ISR
      }
      break;

   // Master Receiver: byte received
   case SMB_MRDB:
      if ( i < SMB_DATA_LEN )       // Is there any data remaining?
      {
         *pSMB_DATA_IN = SMB0DAT;   // Store received byte
         pSMB_DATA_IN++;            // Increment data in pointer
         i++;                       // Increment number of bytes received
         ACK = 1;                   // Set ACK bit (may be cleared later
                                    // in the code)

      }

      if (i == SMB_DATA_LEN)        // This is the last byte
      {
               ACK = 0;                   // Send NACK to indicate last byte
                                    // of this transfer
```

```
            STO = 1;                    // Send STOP to terminate transfer
            SMB_BUSY = 0;               // Free SMBus interface

         }

         break;

      default:
         FAIL = 1;                      // Indicate failed transfer
                                        // and handle at end of ISR
         break;
   }

   if (FAIL)                            // If the transfer failed,
   {
      SMB0CF &= ~0x80;                  // Reset communication
      SMB0CF |= 0x80;
      STA = 0;
      STO = 0;
      ACK = 0;

      FAIL = 0;
      SMB_BUSY = 0;                     // Free SMBus
   }

   SI = 0;                              // Clear interrupt flag
}


//-----------------------------------------------------------------------------
// Timer2 Interrupt Service Routine (ISR)
//-----------------------------------------------------------------------------
//
// A Timer2 interrupt indicates an SMBus SCL low timeout.
// The SMBus is disabled and re-enabled if a timeout occurs.
//
```

```
INTERRUPT(Timer2_ISR, INTERRUPT_TIMER2)
{
   SMB0CF &= ~0x80;                    // Disable SMBus
   SMB0CF |= 0x80;                     // Re-enable SMBus
   TF2H = 0;                           // Clear Timer2 interrupt-pending flag
   SMB_BUSY = 0;                       // Free bus
}


//-----------------------------------------------------------------------------
// Support Functions
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// I2C_ByteWrite
//-----------------------------------------------------------------------------
//
// Return Value : None
// Parameters   :
//   1) unsigned char addr - address to write in the device via I2C
//      range is full range of character: 0 to 255
//
//   2) unsigned char dat - data to write to the address <addr> in the device
//      range is full range of character: 0 to 255
//
// This function writes the value in <dat> to location <addr> in the device
// then polls the device until the write is complete.
//
void I2C_ByteWrite (U8 addr, U8 dat)
{
   while (SMB_BUSY);                   // Wait for SMBus to be free.
   SMB_BUSY = 1;                       // Claim SMBus (set to busy)

   // Set SMBus ISR parameters
   TARGET = SLAVE_ADDR << 1;                 // Set target slave address
   SMB_RW = WRITE;                     // Mark next transfer as a write
```

```
   SMB_SENDWORDADDR = 1;              // Send Word Address after Slave Address
   SMB_RANDOMREAD = 0;                // Do not send a START signal after
                                      // the word address
   SMB_ACKPOLL = 1;                   // Enable Acknowledge Polling (The ISR
                                      // will automatically restart the
                                      // transfer if the slave does not
                                      // acknoledge its address.

   // Specify the Outgoing Data
   WORD_ADDR = addr;                  // Set the target address in the
                                      // device's internal memory space

   SMB_SINGLEBYTE_OUT = dat;          // Store <dat> (local variable) in a
                                      // global variable so the ISR can read
                                      // it after this function exits

   // The outgoing data pointer points to the <dat> variable
   pSMB_DATA_OUT = &SMB_SINGLEBYTE_OUT;

   SMB_DATA_LEN = 1;                  // Specify to ISR that the next transfer
                                      // will contain one data byte

   // Initiate SMBus Transfer
   STA = 1;
}


//-----------------------------------------------------------------------------
// I2C_ByteRead
//-----------------------------------------------------------------------------
//
// Return Value :
//   1) unsigned char data - data read from address <addr> in the device
//      range is full range of character: 0 to 255
//
// Parameters   :
```

```
//   1) unsigned char addr - address to read data from the device
//      range is full range of character: 0 to 255
//
// This function returns a single byte from location <addr> in the device then
// polls the <SMB_BUSY> flag until the read is complete.
//
U8 I2C_ByteRead (U8 addr)
{
   U8 return_val;                     // Holds the return value

   while (SMB_BUSY);                  // Wait for SMBus to be free.
   SMB_BUSY = 1;                      // Claim SMBus (set to busy)

   // Set SMBus ISR parameters
   TARGET = SLAVE_ADDR<<1;               // Set target slave address
   SMB_RW = WRITE;                    // A random read starts as a write
                                      // then changes to a read after
                                      // the repeated start is sent. The
                                      // ISR handles this switchover if
                                      // the <SMB_RANDOMREAD> bit is set.
   SMB_SENDWORDADDR = 1;              // Send Word Address after Slave Address
   SMB_RANDOMREAD = 1;                // Send a START after the word address
   SMB_ACKPOLL = 1;                   // Enable Acknowledge Polling


   // Specify the Incoming Data
   WORD_ADDR = addr;                  // Set the target address in the
                                      // devices's internal memory space

   pSMB_DATA_IN = &return_val;            // The incoming data pointer points to
                                      // the <retval> variable.

   SMB_DATA_LEN = 1;                  // Specify to ISR that the next transfer
                                      // will contain one data byte

   // Initiate SMBus Transfer
```

```
   STA = 1;
   while(SMB_BUSY);                    // Wait until data is read

   return return_val;

}
```

Revision History

Revision 0.7

March, 2020

- Updated document to reference CBPro instead of CBDesktop.
- Updated old CBDesktop C-Code header file example to a recently-generated CBPro C-Code header file.

Revision 0.6

October, 2010

- Updated the entries in register_map.h and the Si5338 programming in the Appendix to reflect what is in the Si5338 data sheet revision 0.6.

Revision 0.5

August, 2010

- Updated the register_map.h file to reflect latest masks.
- Replaced Any-Rate with Any Frequency.
- Replace Multisynth Clock Programmer with ClockBuilder Desktop.
- Updated Appendix code listing to skip register access where the mask is 0x00.

Revision 0.4

December, 2009

- Updated Sections 4.3, 4.4, and Appendix to illustrate how to use the header file generation feature of the Multisynth Clock Programmer.

Revision 0.3

October, 2009

- Added Si5355/56 devices.

Revision 0.2

August, 2009

- Updated Figures 1, 2, and 3.
- Updated "4.1. Firmware Description" on page 5.
- Updated "4.2. Encoding the Register Map" on page 5.
- Added "4.3. Using the Header File Generation Function (Save C Code Header File)" on page 6.

Revision 0.1

July, 2009

- Initial release.

ClockBuilder Pro

One-click access to Timing tools, documentation, software, source code libraries & more. Available for Windows and iOS (CBGo only).

www.silabs.com/CBPro

Timing Portfolio
www.silabs.com/timing

SW/HW
www.silabs.com/CBPro

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com