



Basic SigmaDSP Microcontroller Integration

ANALOG DEVICES, INC.

www.analog.com

Table of Contents

Introduction	3
SigmaStudio and Standalone Application Overview.....	4
Output Files.....	5
Data Capture Window	7
Sequencer Window.....	7
Microcontroller Implementation	12
Accessing Single Registers Or SigmaStudio Cell Parameters	14
Implementing Sequencer Modes.....	14

Introduction

SigmaStudio software and any standalone application software developed by Analog Devices Inc (ADI), under the Digital Audio Unit (DAU) software group supervision, can drive any SigmaDSP family of audio specific processors. SigmaStudio and its standalone applications software share the same highly intuitive user interface for audio system development and tuning. Hence, most of the tools and tools menus are similar and execute similar tasks. This document is intended to illustrate as clear as possible the integration process of ADI user interface software into a micro controller system. For simplicity reasons, the examples used in this document refer to a specific microcontroller and microcontroller graphical environment to show a proof of concept. All the procedures and methodologies described in this manual are applicable to any kind of microcontroller and their graphical environment.

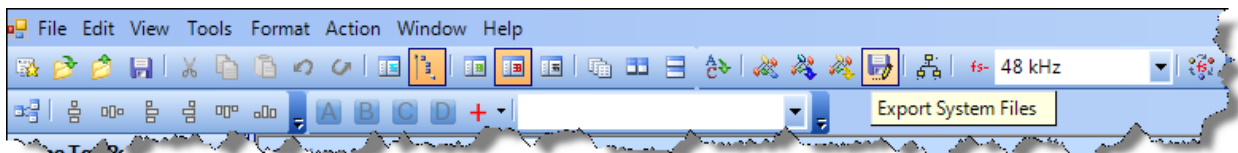
SigmaStudio and Standalone Application Overview

While using SigmaStudio and the project created is ready to be ported to the microcontroller (for details, please refer to the Microcontroller Implementation section), there are a few steps we must consider:

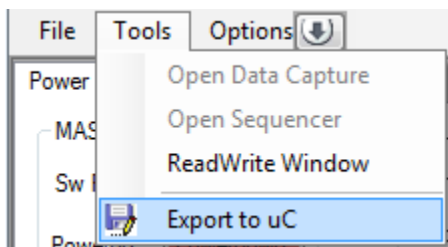
1. Locate and press the “Link Compile Download” button



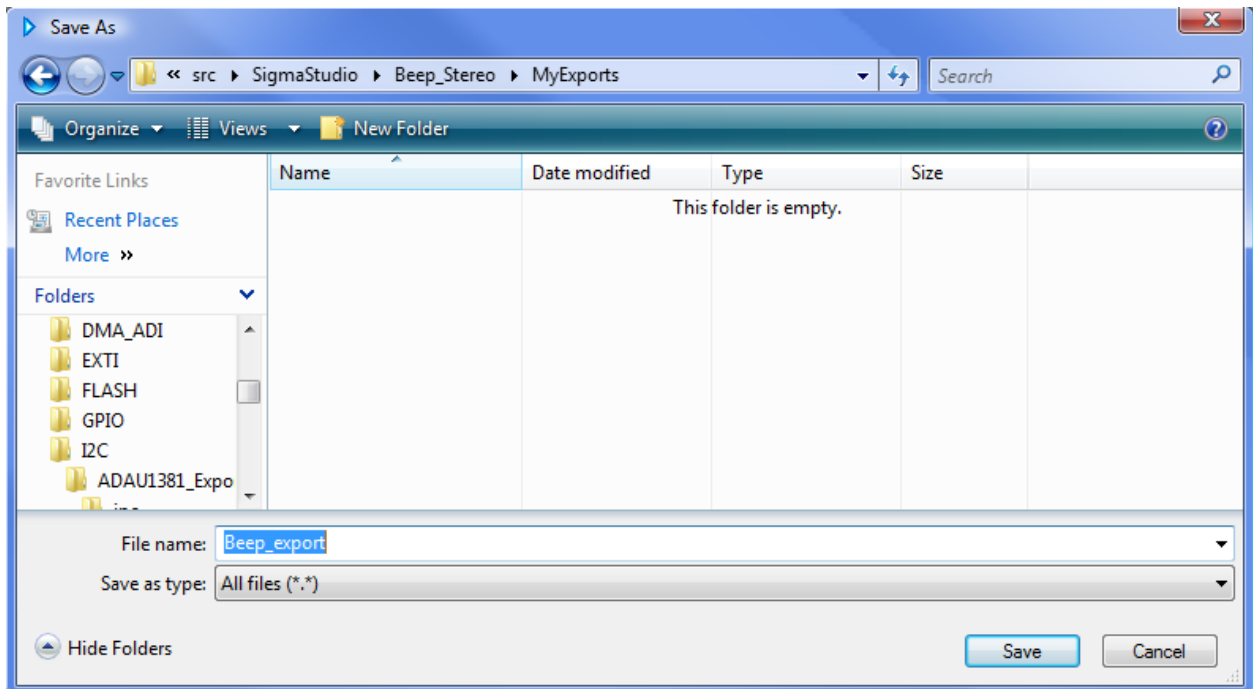
2. After pressing the button, the “Export System Files” button will be enabled



3. In a similar fashion, if using any standalone application, locate the “Export System Files” or “Export to uC” button



4. Pressing the “Export System Files” or “Export to uC” button will prompt a dialog box that will allow the user to choose a folder where all the files would be saved. It is always good practice to give a meaningful name.



Output Files

The following is a list of all files that are automatically generated, where the '*' represents the given name:

- *.hex
 - This file is the Param Data of the System
- *.params
 - Gives detailed information of Cell Name, Parameter Name, Address, Value, and Data. For example:
 - Cell Name = Tone1
 - Parameter Name = sin_lookupAlg19401mask
 - Parameter Address = 8
 - Parameter Value = 255
 - Parameter Data :
0X00, 0X00, 0X00, 0XFF,

- *.h
 - This file contains all the information needed to access the IC registers, program data, and param data using a method named “default_download()” (where applicable). The “default_download()” method calls all the registers and commands exactly as they appear on the capture window. For any SigmaDSP that has no engine (i.e. CODECs, ADCs, DACs, etc) only the register information will be displayed, and a method should be manually implemented under the user’s needs.

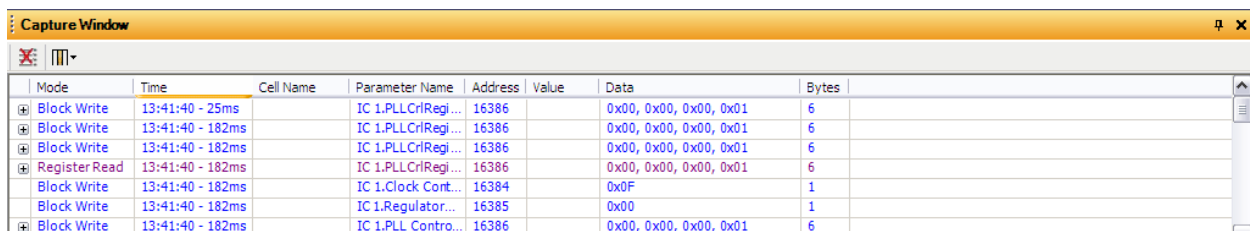
Mode	Time	Cell Name	Parameter Name	Address	Value	Data	Bytes
Block Write	10:16:38 - 334ms	1	IC 1.Start Pulse	16619		0x0A	1
Block Write	10:16:38 - 379ms	2	IC 1.Sound Eng...	16630		0x00	1
Block Write	10:16:38 - 400ms	3	IC 1.Clock Cont...	16384		0x0E	1
Block Write	10:16:38 - 401ms		IC 1.Regulator...	16385		0x00	1
Block Write	10:16:38 - 403ms		IC 1.PLL Contro...	16386		0x00, 0x80, 0x00, 0x00,	6
DELAY	10:16:38 - 405ms		IC 1.Delay			0x00, 0x64	2
Read Request	10:16:38 - 426ms		IC 1.PLL Control	16386			6
Block Write	10:16:38 - 447ms		IC 1.Microphon...	16392		0x00	1
Block Write	10:16:38 - 448ms		IC 1.Delay P...	16393		0x00, 0x00, 0x00, 0x00,	8

- *_PARAM.h
 - Contains the parameter definitions of each independent module. For example:
 - /* Module Tone1 - Sine Tone*/
 - #define MOD_TONE1_COUNT 3
 - #define MOD_TONE1_DEVICE "IC1"
 - #define MOD_STATIC_TONE1_ALGO_MASK_ADDR 8
 - #define MOD_STATIC_TONE1_ALGO_MASK_FIXPT 0x000000FF
- *_REG.h
 - Contains the Register definition for the SigmaDSP IC. Example:
 - /* MCLK Pad Control - Registers (IC 1) */
 - #define REG_MCLK_PAD_CONTROL_ADDR 0x4031
 - #define REG_MCLK_PAD_CONTROL_VALUE 0x2

- “SigmaStudioFW.h” is the only file that is not automatically generated
 - It is located on [Program Files]\Analog Devices\Sigma Studio 3.x folder, or where the executable file resides.
 - It contains a template with pre defined macros. The user has to implement these macros manually to better suit his needs on a specific microcontroller family.
 - One macro that is very helpful is the command “SIGMA_WRITE_REGISTER_BLOCK (int devAddress, int address, int length, ADI_REG_TYPE *pData)”, which contains basic information such as device address, instruction address , length, and instruction data.

Data Capture Window

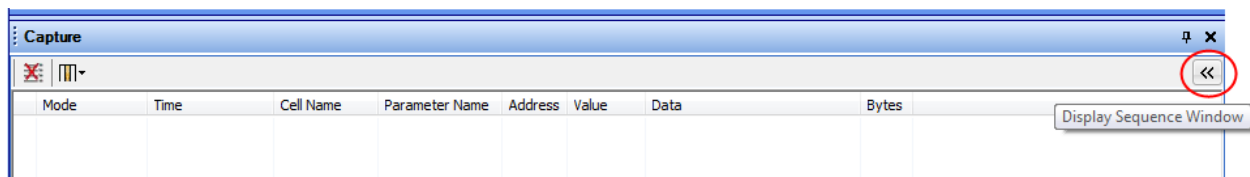
The Data Capture Window is a tool that is basically used to see what has been written or read over the USB communications link. It also serves as a log window where the user can grab as much lines as desired and drop them into the Sequencer Window (explained later). It contains the Parameter/Register name, address, data, number of bytes, etc. This window is very useful for application development and debugging.



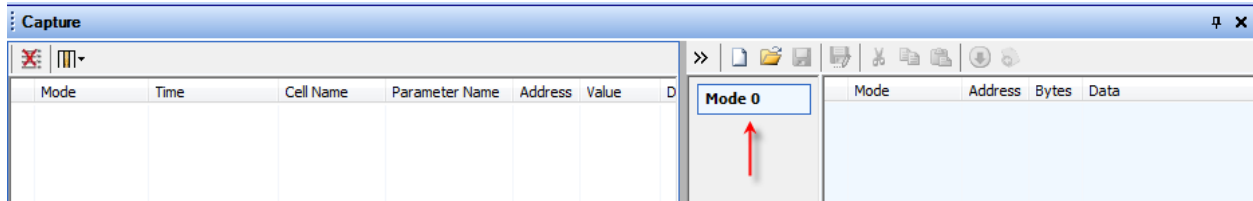
Mode	Time	Cell Name	Parameter Name	Address	Value	Data	Bytes
Block Write	13:41:40 - 25ms		IC 1.PLLCrIRegi...	16386		0x00, 0x00, 0x00, 0x01	6
Block Write	13:41:40 - 182ms		IC 1.PLLCrIRegi...	16386		0x00, 0x00, 0x00, 0x01	6
Block Write	13:41:40 - 182ms		IC 1.PLLCrIRegi...	16386		0x00, 0x00, 0x00, 0x01	6
Register Read	13:41:40 - 182ms		IC 1.PLLCrIRegi...	16386		0x00, 0x00, 0x00, 0x01	6
Block Write	13:41:40 - 182ms		IC 1.Clock Cont...	16384		0x0F	1
Block Write	13:41:40 - 182ms		IC 1.Regulator...	16385		0x00	1
Block Write	13:41:40 - 182ms		IC 1.PLL Contro...	16386		0x00, 0x00, 0x00, 0x01	6

Sequencer Window

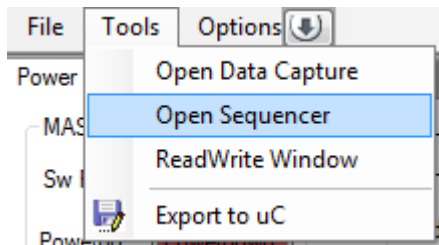
The Sequencer window is a powerful tool that creates sets of instructions for a specific task. To get access to it from SigmaStudio, simply open the Data Capture window (Ctrl + 5) and click on the upper right double arrow button.



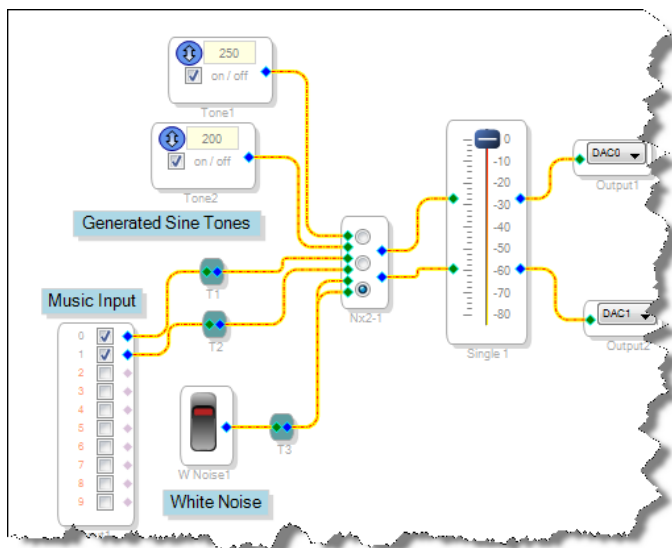
Then, the Sequencer Window will appear as part of the Data Capture



For standalone applications look into the main menu, under “Tools” and select “Open Sequencer”. Notice that the Data Capture Window is a separate window.

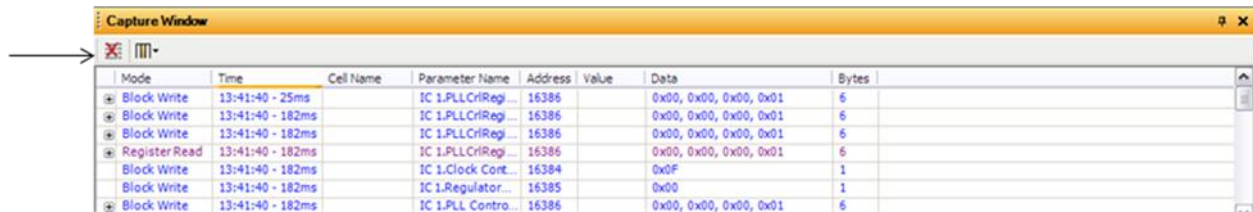


Suppose that we have the following SigmaStudio schematic:

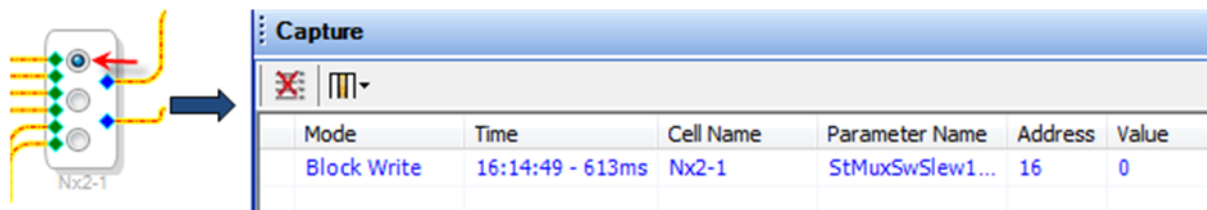


Our goal is to compile this SigmaStudio schematic or project and port it to a microcontroller.

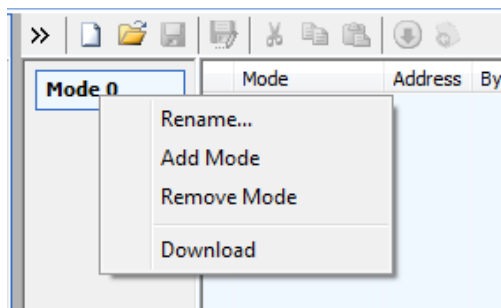
At this point we are able to create sequences. Using the Data Capture window as a tool, we can start populating the sequencer window with data. For example, we want to switch between internal generated sine tones, external input (such as music), and white noise via Nx2-1 Multiplexer. Instead of using this Nx2-1 control, we are going to use the sequencer window. To get started, clear all possible information that currently may exist on the data capture window by clicking the red “x” button.



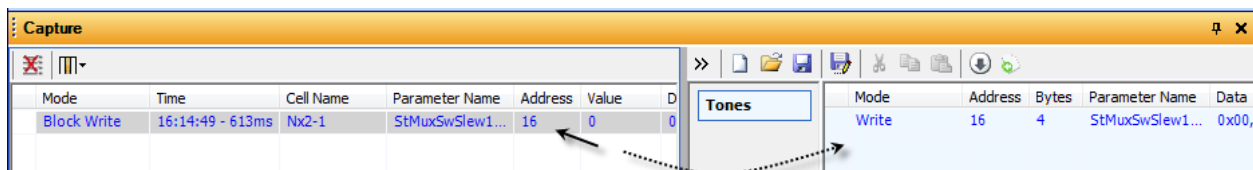
Now, it's time to create modes using the data capture window. Clicking at the first selection on our Nx2-1 control will write the necessary information to the data capture to route the Sine Tones.



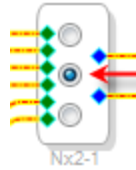
Going to the sequencer window, let's create a mode named "Tones" with the information displayed on the data capture window. Right click the mouse over the "Modes" button and select "Rename" and type "Tones"



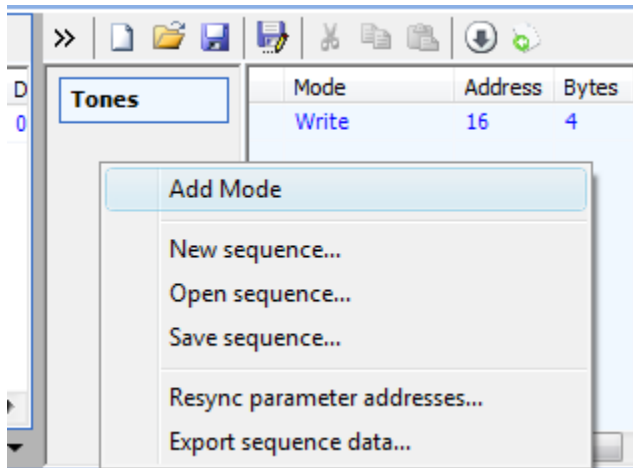
Creating modes using the data capture window is as simple as dragging and dropping the information displayed on the data capture window to the sequencer window.



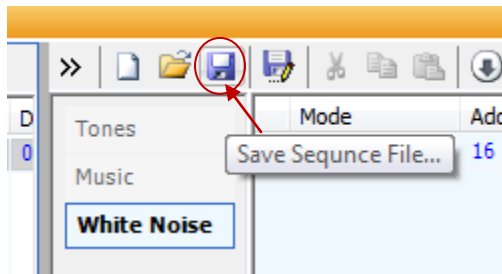
To create another mode within our Sequencer Window, clear off the information on the data capture and select the next routing path from the Nx2-1 control.



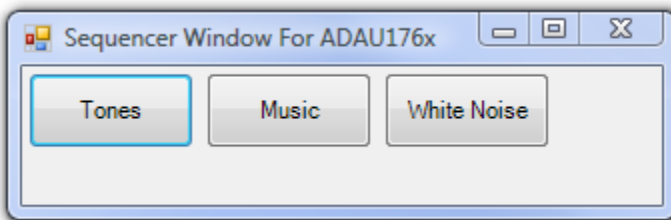
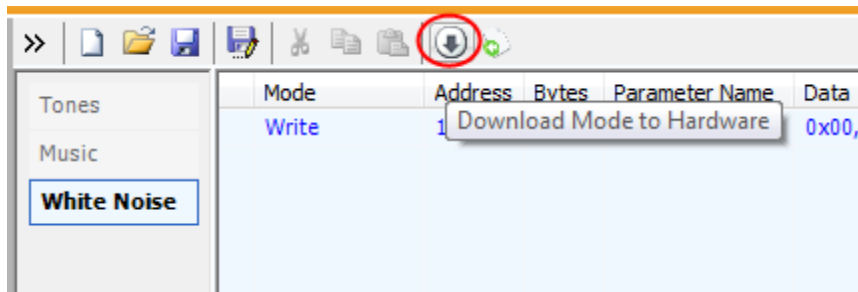
Go to the Sequencer Window, right click the mouse and select “Add Mode” and rename it to “Music”.



Drag and drop the information from the data capture window into the sequencer window. Repeat for the last option and create a mode named “White Noise”. Finally, save the sequencer file on a folder by pressing the “Save Sequence File” button.

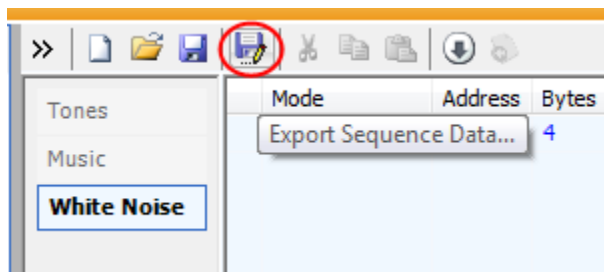


Test the modes by clicking one of two buttons: One could be the “Download Mode to Hardware” button which downloads everything on the current selected mode, where in this case it will only download the “White Noise” mode. The other button is the “Launch Sequencer Window” button which creates a floating window with all modes created. This way, the user could download any mode in any order by pressing the desired button.



Once we tested our sequencer with all its modes, and we already have our basic program into the microcontroller, the next step is to include some sequences and map each mode to a GPIO port on our ARM M3 series uC. It could be any other microcontroller.

First, we click on the “Export Sequence Data” button.



Then, we give it a name. For this example we’ll name it “1761_Demo.h”. After pressing “OK” the following files will be created:

- 1761_Demo.h. This file contains references for all modes
- Tones_Modes.h, Music_Modes.h, and White Noise_Modes.h. Each independent file contains its mode data. For example:
 - `ADI_REG_TYPE TONES_0[4] = {0x00, 0x00, 0x00, 0x00};`
 - `void TONES_download()`
 - `{`
 - `SIGMA_WRITE_REGISTER_BLOCK(DEVICE_ADDR_IC_1, 0x0010, 6,`
`TONES_0); /* StMuxSwSlew1coeffname */}`

Microcontroller Implementation

Now that we had a SigmaStudio overview and learned which files are generated and its function, we are ready to implement them to our target microcontroller. For demonstration purposes, we're going to use an ARM Cortex M3 series, and Keil graphical environment. Assuming that the user already has defined the basic microcontroller configuration such System Clocks, NVIC, GPIO, and the communication protocol configurations (in our case I2C), then we move to our "main.c" file. On this file we must include the following header files:

- SigmaStudioFW.h (Assuming that the macros had been already defined by the user)
- *.h, and *_PARAM.h (where applicable).

The main function should look similar to this

```

061 /******
062 * Function Name : main
063 * Description : Main program
064 * Input : None
065 * Output : None
066 * Return : None
067 *****
068 int main(void)
069 {
070
071
072 #ifdef DEBUG
073 debug();
074 #endif
075
076 /* System clocks configuration -----
077 RCC_Configuration();
078
079 /* NVIC configuration -----
080 NVIC_Configuration();
081
082 /* GPIO configuration -----
083 GPIO_Configuration();
084
085 /* I2C1 configuration -----
086 I2C1_Config();
087
088 /* Load DSP Main Program*/
089 default_download();
090

```

Notice that we are calling "default_download()". This function will actually grab all of the SigmaStudio parameters and load it to the microcontroller. If using a standalone application software, the user must define a function similar to the "default_download()" function. This function looks like this:

```

3 void default_download() {
4     SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_START_PULSE_ADDR, REG_START_PULSE_BYTE, R0_START_PULSE_ADDR, REG_START_PULSE_BYTE, R0_START_PULSE_BYTE );
5     SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_SOUND_ENGINE_RUN_ADDR, REG_SOUND_ENGINE_RUN_BYTE, R1_SOUND_ENGINE_RUN_ADDR, REG_SOUND_ENGINE_RUN_BYTE, R1_SOUND_ENGINE_RUN_BYTE );
6     SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R2_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R2_PLL_CONTROL_BYTE );
7     SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R3_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R3_PLL_CONTROL_BYTE );
8     SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R4_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R4_PLL_CONTROL_BYTE );
9     SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_CLOCK_CONTROL_ADDR, REG_CLOCK_CONTROL_BYTE, R5_CLOCK_CONTROL_ADDR, REG_CLOCK_CONTROL_BYTE, R5_CLOCK_CONTROL_BYTE );
10    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_REGULATOR_CONTROL_ADDR, REG_REGULATOR_CONTROL_BYTE, R6_REGULATOR_CONTROL_ADDR, REG_REGULATOR_CONTROL_BYTE, R6_REGULATOR_CONTROL_BYTE );
11    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R7_PLL_CONTROL_ADDR, REG_PLL_CONTROL_BYTE, R7_PLL_CONTROL_BYTE );
12    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_DIGITAL_MIC_AND_BEEP_CONTROL_ADDR, REG_DIGITAL_MIC_AND_BEEP_CONTROL_BYTE, R8_DIGITAL_MIC_AND_BEEP_CONTROL_ADDR, REG_DIGITAL_MIC_AND_BEEP_CONTROL_BYTE, R8_DIGITAL_MIC_AND_BEEP_CONTROL_BYTE );
13    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_RECORD_PWR_MANAGEMENT_ADDR, REG_RECORD_PWR_MANAGEMENT_BYTE, R9_RECORD_PWR_MANAGEMENT_ADDR, REG_RECORD_PWR_MANAGEMENT_BYTE, R9_RECORD_PWR_MANAGEMENT_BYTE );
14    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_SERIAL_PORT_CTRL_0_ADDR, REG_SERIAL_PORT_CTRL_0_BYTE, R10_SERIAL_PORT_CTRL_0_ADDR, REG_SERIAL_PORT_CTRL_0_BYTE, R10_SERIAL_PORT_CTRL_0_BYTE );
15    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_CONVERTER_CTRL_0_ADDR, REG_CONVERTER_CTRL_0_BYTE, R11_CONVERTER_CTRL_0_ADDR, REG_CONVERTER_CTRL_0_BYTE, R11_CONVERTER_CTRL_0_BYTE );
16    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_ADC_CONTROL_0_ADDR, REG_ADC_CONTROL_0_BYTE, R12_ADC_CONTROL_0_ADDR, REG_ADC_CONTROL_0_BYTE, R12_ADC_CONTROL_0_BYTE );
17    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_PLAYBACK_MIXER_LEFT_CONTROL_ADDR, REG_PLAYBACK_MIXER_LEFT_CONTROL_BYTE, R13_PLAYBACK_MIXER_LEFT_CONTROL_ADDR, REG_PLAYBACK_MIXER_LEFT_CONTROL_BYTE, R13_PLAYBACK_MIXER_LEFT_CONTROL_BYTE );
18    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_DAC_CONTROL_0_ADDR, REG_DAC_CONTROL_0_BYTE, R14_DAC_CONTROL_0_ADDR, REG_DAC_CONTROL_0_BYTE, R14_DAC_CONTROL_0_BYTE );
19    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_SERIAL_PORT_PAD_CTRL_0_ADDR, REG_SERIAL_PORT_PAD_CTRL_0_BYTE, R15_SERIAL_PORT_PAD_CTRL_0_ADDR, REG_SERIAL_PORT_PAD_CTRL_0_BYTE, R15_SERIAL_PORT_PAD_CTRL_0_BYTE );
20    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_COMM_PORT_PAD_CTRL_0_ADDR, REG_COMM_PORT_PAD_CTRL_0_BYTE, R16_COMM_PORT_PAD_CTRL_0_ADDR, REG_COMM_PORT_PAD_CTRL_0_BYTE, R16_COMM_PORT_PAD_CTRL_0_BYTE );
21    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_MCLK_PAD_CTRL_ADDR, REG_MCLK_PAD_CTRL_BYTE, R17_MCLK_PAD_CTRL_ADDR, REG_MCLK_PAD_CTRL_BYTE, R17_MCLK_PAD_CTRL_BYTE );
22    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_LOW_LATENCY_REG_ADDR, REG_LOW_LATENCY_REG_BYTE, R18_LOW_LATENCY_REG_ADDR, REG_LOW_LATENCY_REG_BYTE, R18_LOW_LATENCY_REG_BYTE );
23    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_DIGITAL_POWER_DOWN_0_ADDR, REG_DIGITAL_POWER_DOWN_0_BYTE, R19_DIGITAL_POWER_DOWN_0_ADDR, REG_DIGITAL_POWER_DOWN_0_BYTE, R19_DIGITAL_POWER_DOWN_0_BYTE );
24    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_GPIO_0_CONTROL_ADDR, REG_GPIO_0_CONTROL_BYTE, R20_GPIO_0_CONTROL_ADDR, REG_GPIO_0_CONTROL_BYTE, R20_GPIO_0_CONTROL_BYTE );
25    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_START_PULSE_ADDR, REG_START_PULSE_BYTE, R21_START_PULSE_ADDR, REG_START_PULSE_BYTE, R21_START_PULSE_BYTE );
26    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_ROUTING_MATRIX_INPUTS_ADDR, REG_ROUTING_MATRIX_INPUTS_BYTE, R22_ROUTING_MATRIX_INPUTS_ADDR, REG_ROUTING_MATRIX_INPUTS_BYTE, R22_ROUTING_MATRIX_INPUTS_BYTE );
27    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_ROUTING_MATRIX_OUTPUTS_ADDR, REG_ROUTING_MATRIX_OUTPUTS_BYTE, R23_ROUTING_MATRIX_OUTPUTS_ADDR, REG_ROUTING_MATRIX_OUTPUTS_BYTE, R23_ROUTING_MATRIX_OUTPUTS_BYTE );
28    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_SERIAL_DATAGPIO_PIN_CONFIG_ADDR, REG_SERIAL_DATAGPIO_PIN_CONFIG_BYTE, R24_SERIAL_DATAGPIO_PIN_CONFIG_ADDR, REG_SERIAL_DATAGPIO_PIN_CONFIG_BYTE, R24_SERIAL_DATAGPIO_PIN_CONFIG_BYTE );
29    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_SERIAL_PORT_SAMPLE_RATE_SETTING_ADDR, REG_SERIAL_PORT_SAMPLE_RATE_SETTING_BYTE, R25_SERIAL_PORT_SAMPLE_RATE_SETTING_ADDR, REG_SERIAL_PORT_SAMPLE_RATE_SETTING_BYTE, R25_SERIAL_PORT_SAMPLE_RATE_SETTING_BYTE );
30    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_NON_MODULO_RAM_1_ADDR, REG_NON_MODULO_RAM_1_BYTE, R26_NON_MODULO_RAM_1_ADDR, REG_NON_MODULO_RAM_1_BYTE, R26_NON_MODULO_RAM_1_BYTE );
31    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, PROGRAM_ADDR, PROGRAM_SIZE, Program_Data );
32    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, PARAM_ADDR, PARAM_SIZE, Param_Data );
33    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_SOUND_ENGINE_RUN_ADDR, REG_SOUND_ENGINE_RUN_BYTE, R27_SOUND_ENGINE_RUN_ADDR, REG_SOUND_ENGINE_RUN_BYTE, R27_SOUND_ENGINE_RUN_BYTE );
34    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_START_PULSE_ADDR, REG_START_PULSE_BYTE, R28_START_PULSE_ADDR, REG_START_PULSE_BYTE, R28_START_PULSE_BYTE );
35    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_DEJITTER_REGISTER_CTRL_ADDR, REG_DEJITTER_REGISTER_CTRL_BYTE, R29_DEJITTER_REGISTER_CTRL_ADDR, REG_DEJITTER_REGISTER_CTRL_BYTE, R29_DEJITTER_REGISTER_CTRL_BYTE );
36    SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_DEJITTER_REGISTER_ADDR, REG_DEJITTER_REGISTER_BYTE, R30_DEJITTER_REGISTER_ADDR, REG_DEJITTER_REGISTER_BYTE, R30_DEJITTER_REGISTER_BYTE );
37

```

Notice that the SIGMA_WRITE_REGISTER_BLOCK(int devAddress, int address, int length, ADI_REG_TYPE *pData) macro is used. This macro is defined on the SigmaStudioFW.h file as follows:

```

049  /*
050  void SIGMA_WRITE_REGISTER_BLOCK(int devAddress, int address, int length, ADI_REG_TYPE *pData )
051  {
052
053  int ii=0;
054  int zz=0;
055  Tx_Idx = 0;
056
057  /*----- Transmission Phase -----*/
058  ThisBufferSize = Address_Length + length;
059
060  I2C1_Buffer_Tx[0] = (address & 0xFF00)>>8;
061  I2C1_Buffer_Tx[1] = address & 0x00FF;
062
063  for(zz=0;zz<length;zz++)
064  {
065  I2C1_Buffer_Tx [zz+Address_Length] = pData[zz];
066  }
067  Tx_Idx = 0;
068  for(ii =0;ii < ThisBufferSize;ii++)
069  {
070
071  NextBufferEnd = ThisBufferSize;//I2C1_numbytes[ii];
072  if(ii == 0) I2C_GenerateSTART(I2C1, ENABLE);
073  /* Send data */
074  while(Tx_Idx < NextBufferEnd)
075  {
076
077  }
078
079  }
080

```

Accessing Single Registers Or SigmaStudio Cell Parameters

As an example, we're going to turn on/off sine Tone 1 (Left Tone) and sine Tone 2 (Right Tone).

1. Declare a value vector of type "ADI_REG_TYPE" with the desired value to be modified.
2. Call the "SIGMA_WRITE_BLOCK" macro with the correspondent parameters

```

5  /*****
6  * Function Name   : Mute_Left_Tone
7  * Description    : Mutes Left Tone
8  * Input          : None
9  * Output         : None
0  * Return        : None
1  *****/
2  void Mute_Left_Tone(void)
3  {
4      ADI_REG_TYPE Val[4] = {0x00, 0x00, 0x00, 0x00};
5      ADI_REG_TYPE Val2[4] = {0x00, 0x80, 0x00, 0x00};
6
7      /* Mutes the Left Tone */
8      SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, MOD_TONE1_ALGO_ON_ADDR, 4, Val );
9
10     /* Unmutes the Right Tone */
11     SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, MOD_TONE2_ALGO_ON_ADDR, 4, Val2 );
12 }

```

Implementing Sequencer Modes

On the "main.c" file, we need to include the "1761_Demo.h" file. On the microcontroller graphical environment we need to add the auto generated "1761_Demo.c" file, which contains all the call functions (the Modes) from the Sequencer.