Design Report: Karaoke Machine implemented on the DE2 Board Using SCOMP and the Wolfson Audio Codec

Patrick Dillon
Sahil Gupta
Eric Patterson
Josh Stirnemann
Prahlad Venkatesh

ECE 2031, Digital Design Lab
Section L06
Spring 2012

Submitted
24 April 2012

Georgia Institute of Technology
School of Electrical and Computer Engineering

Executive Summary

This report describes the solution for implementing a karaoke machine using a simple computer (SCOMP), Wolfson WM8731 audio codec and an I^2C controller on a DE2 board. I^2C controller was written in VHDL which helped in managing the data between SCOMP, the audio codec and several other peripheral devices. For this project, the seven peripheral devices used were switches, pushbuttons, LEDs, LCD screen, microphone, speaker and an iPod. I^2C controller sends two bytes of data when writing and receives one byte of data when reading. It sends 8-bit commands to any 7-bit addresses on I^2C bus. It responds to the in and out commands of the SCOMP. An assembly program was written and downloaded to the DE2 board to control the audio codec and all the peripheral devices. This assembly program consisted of four main states, namely, AudioInit, HandleButtons, HandleSwitches, and HandleState. AudioInit takes care of the initializing the settings of the audio codec, HandleButtons and HandleSwitches manage the user response of the pushbuttons and the switches respectively, and HandleState ensures correct transition between the states. This execution provides a robust design that allows the user to easily communicate to other I^2C enabled devices for future enhancements. The future work which the team wants to do with the current project includes ideas such as integrating the video codec to display the song lyrics and visual backgrounds, adding clock synchronization and playing music from a memory card or USB storage. Moreover, the main advantage of this implementation is discretization of fades and preventing the blocking of the user. It also allows state transition to stop in the middle of the fade. The design met all the requirements of the project and was very successful during the demonstration.

Design Report: Karaoke Machine implemented on the DE2 Board Using SCOMP and the Wolfson Audio Codec

1.0 Introduction

This report explains a solution to the implementation of a karaoke machine on the DE2 board using the I^2C controller, the SCOMP and the audio codec. The I^2C controller and the SCOMP were written in VHDL and downloaded to the DE2 board. The I^2C controller facilitates data exchange between the SCOMP and the audio codec. The audio codec and all the peripheral devices were controlled by an assembly program downloaded to the DE2 board.

1.1 Problem Description

The goal of this project is to design a general-purpose I^2C controller with a standard Simple Computer (SCOMP) I/O interface and to establish communication between the SCOMP and the Wolfson WM8731 audio codec. The following were the requirements to achieve the project goal:

1.1.1 I^2C Controller

- Responds to standard I/O commands (IN, OUT) at one or more IO address beginning at 0x10.
- Sends arbitrary 8-bit commands to arbitrary 7-bit addresses on the I^2C bus.
- Sends write commands from the SCOMP.
- Reads and relays a response from the SCOMP to the I^2C enabled device.

1.1.2 Audio Codec

- The I^2C controller should communicate with the audio codec chip on the DE2 board midway through the project.

- Develop a user interface using several peripheral devices on the DE2 board such as seven segment displays, LCD screen, LEDs, pushbuttons, etc.
- Create a demonstration SCOMP program using assembly to achieve the above requirements.
- Develop a specific application of the audio codec to meet all the requirements.

1.2 Design Solution

To meet all the above requirements, the team decided to design a karaoke machine on the DE2 board. I^2C controller sends two bytes of data when writing and receives one byte of data when reading. It sends 8-bit commands to any 7-bit addresses on I^2C bus. It responds to the in and out commands of the SCOMP. For this project, the seven peripheral devices used were switches, pushbuttons, LEDs, LCD screen, microphone, speaker and an iPod. An assembly program was written and downloaded to the DE2 board to control the audio codec and all the peripheral devices. This assembly program consisted of four main states, namely, AudioInit, HandleButtons, HandleSwitches, and HandleState. AudioInit takes care of the initialization of the settings of the audio codec, HandleButtons and HandleSwitches manage the user response of the pushbuttons and the switches respectively, and HandleState ensures correct transition between the states.

The design met all the requirements of the project and was very successful during the demonstration.

The controller will regulate the information being passed between the SCOMP and the I^2C enabled device by using two subroutines, an audio codec and I^2C controller driver, which will handle the address and the different functionalities of each device. The I^2C controller will initially start in an ideal state. Once the controller receives a valid data string, it will decide to go into a sending or receiving state. When the data transfer is complete, the SCOMP will give instructions to send the information to the audio codec over the SDA (serial data line). With the SCOMP being the master control device, the speed of the SCL (serial clock line) is dependent on the initial clock value defined in the SCOMP.

Technical Approach

Proposed Overall Design

To fulfill the goal of the project, several design phases were created to ensure the completion of each requirement. The team aims to complete a phase prior to moving forward in the design process. These phases are implemented to yield a highly efficient design; as a result, debugging and executing each design phase will be easier and independent of other phases. Because a team member is responsible for the completion of a design phase, the project progression will be optimized due to the minimization of duplicated work.

The karaoke machine will be implemented by using the I^{2}C controller as the means of communication between several peripheral devices (LEDs, switches, and push buttons), the SCOMP programmed in the DE2 board, and the Wolfson WM8731 audio codec integrated on the DE2 board. Figure 1 provides a graphical representation of the karaoke machine the team will design.

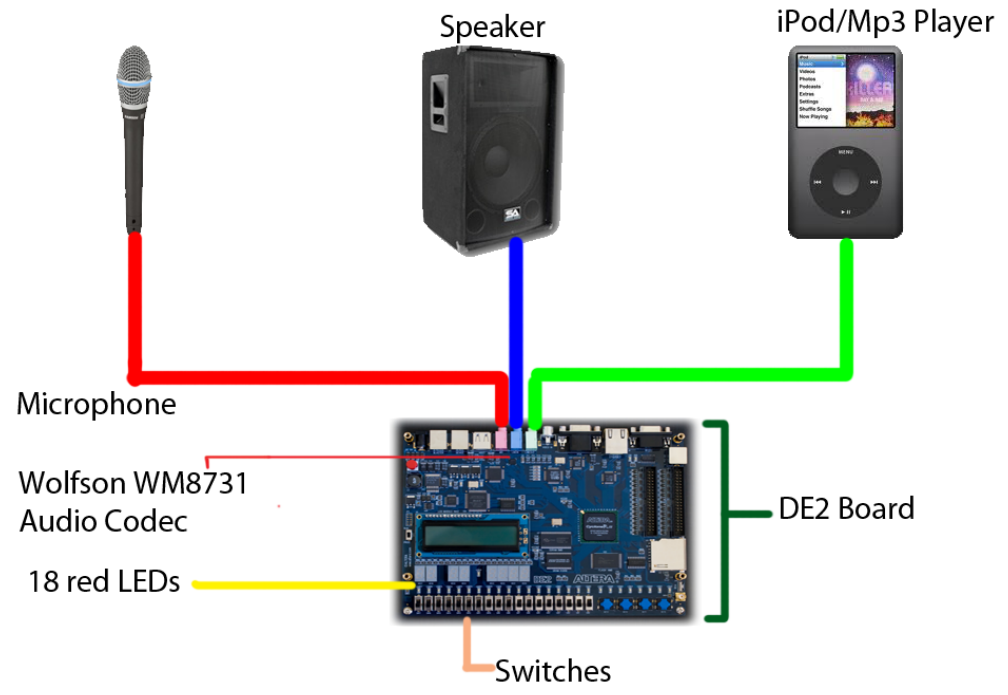


Figure 1. The setup of the karaoke machine with all the connections between devices shown.

In the port declaration of I2C_Controller.vhd, three standard logic inputs are included to distinguish different words sent over the I/O data bus. These input ports are three individual chip selects, which are utilized by the I^2C controller for managing state machine transitions and driving the tri-state buffer. Integer counting within the controller will keep track of the number of bits sent over the I^2C bus. The I^2C addresses are seven bit and the data will be sent in one byte.

I^2C Controller Design

For the development of the I^2C controller, an Altera DE2 board will be programmed with the Quartus II Web Edition software version 9.1 with SP2. The logic and states of the I^2C controller will be written in VHDL due to its compatibility with the Quartus software. This involves the creation of two state machines - one for the SCOMP to I^2C interface and one that sends data over the I^2C bus.

By default, the I^2C controller is in an idle state. Therefore, it waits for the address of an I^2C device to communicate with. The I^2C controller enters the sending or receiving state of the state machine depending on whether the eighth bit of the address is a zero (write) or one (read). In the 'sending' state, the I^2C controller waits to send data. If SCL and SDA are both one, it starts sending a byte and in the next clock cycle, transitions to a send bit state where it will stay for the next seven cycles. After eight bits have been transferred, it waits for an acknowledge (ACK) from the slave. If the ACK is received, then the state machine will start transferring the next byte. If there are no bytes left to send, it will send the stop sequence, and the I^2C controller will return to the idle state. If the ACK is not received, then it will transition back to start sending the byte without proceeding to the next byte to be sent.

If the I^2C controller receives a one as the eighth bit of the address in the idle state, then it goes to the 'receiving' part of the state machine. In this state, the process is similar to that in the 'sending' part of the state machine. The I^2C controller start by sending the address of the I^2C requested device over the SDL. Then, it loops while receiving 8 bits from the slave, while controlling the clock. Once an entire byte is received, the I^2C controller will pull the SDA low to acknowledge that the byte was received. Finally, the I^{2}C controller sends the stop sequence and stores the received byte in a register to be retrieved by SCOMP using IN I2C_DATA and transitions to the idle state.

Interaction with Audio Codec

The I^{2}C controller has two bidirectional lines, the Serial Data Line and the Serial Clock Line that are connected to the audio codec.

For this part, the plan is to create a karaoke machine that receives music from a MP3 player. This task will be achieved by the usage of the three jacks in the audio codec on the DE2 board, i.e., the line-in, line-out, and mic-in jacks. The line-in jack will be used as an interface between the MP3 player and the audio codec. The mic-in jack will be used to send the input of human voice inputs by means of a microphone. The line-out jack will be used to connect the audio codec to the speakers, so as to have an output for the sound. There are also pushbuttons that send a signal to reset the sound. Additionally, there is a digital to analog and analog to digital converter present in the audio codec that will allow for automatic conversion into the desired format.

The objective is to be able to communicate with the audio codec, i.e., prompt it to start using the I^{2}C controller. This start state will be accomplished by determining the address of the audio codec and finding a specific command to set it running from the I^{2}C controller. Afterwards, the team plans to program the LEDs and the switches in a way that they can have some sort of impact on the output volume by utilizing a subroutine. In the subroutine, the use of commands will enable the manipulation of the output volume like ‘increase volume’, ‘decrease volume’, ‘mute’, etc.

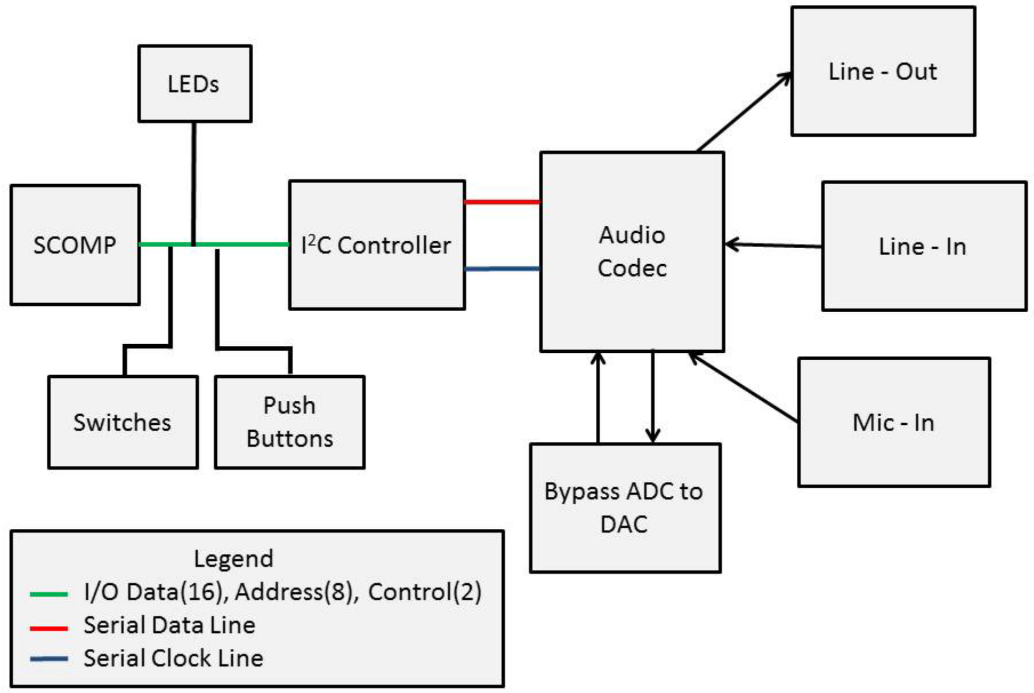


Figure 2. I/O subsystem showing interconnection of audio codec with I^{2}C.

SCOMP Program Demonstration

The functionality of the I^{2}C controller will be shown through the demonstration of a karaoke machine. As mentioned before, this karaoke machine will require a user to attach a microphone, speaker, and a music player. In the initial state, the karaoke machine will set the volume of the audio output to zero, waiting for the user to press a button (KEY [0]) before fading-in the volume to 50%. Pressing that button again will fade the volume back down to 0. The buttons will be checked via polling their respective I/O addresses and will be performed continuously in between sending commands to the audio codec. Commands will be sent to the audio codec using two sets of subroutines. The first set of subroutines is the audio codec driver. This driver manages the audio codec's address and the different functions that can be performed by the audio codec. In order to communicate with the audio codec, the audio codec driver will use a set of subroutines, which compose the I^{2}C controller driver. The I^{2}C controller driver consists of the I^{2}C controller's I/O address, which is a command to set the target I^{2}C device address, and commands, which perform a send or receive on the I^2C controller. The I^2C controller then transfers or requests the corresponding data using the SDA and SCL lines on the I^2C bus.

For the I2C_WRITE subroutine, the accumulator is expected to contain the data that needs to be sent, and a specific location in memory is expected to hold the address of the I^2C device to send the data to. The I2C_WRITE subroutine starts by storing the data it needs to send. It then loads the address of the I^2C device, sets the write bit, and sends the address to the I^2C controller using OUT I2C_ADDR. Next, the subroutine loads the data to send back into the accumulator and executes OUT I2C_DATA. Finally, it enters a loop, where it waits for IN I2C_TEST to input 0 into the accumulator before returning.
The I2C_READ subroutine expects the address of the I^2C device be in the accumulator. It starts by sending the address to the I^2C controller using OUT I2C_ADDR. It then waits, by checking if IN I2C_TEST sets the accumulator to 0, for the I^2C controller to enter the IDLE state which means the data has been sent successfully before returning.

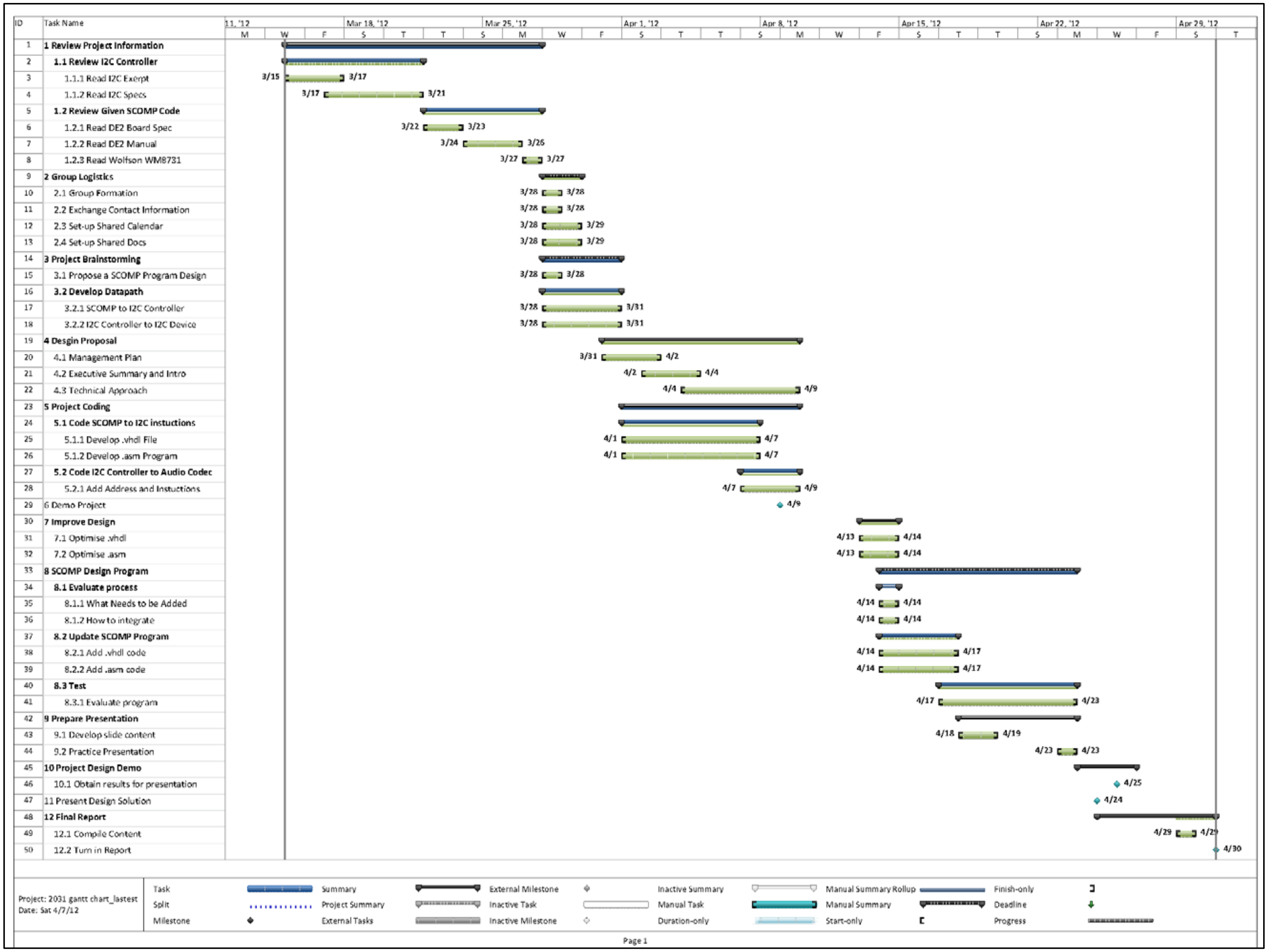
Management Plan

Appendix A contains a Gantt chart that provides a visual representation of the team's project schedule.

Contingency Plan

In the event that the I^2C controller needs to be able to enter a slave mode, the I^2C controller's state machine will be adapted to contain a slave state, where it will buffer any incoming data and operate based on the clock the master device drives over the SCL. Once the buffer is full, the I^2C controller will stretch the clock by holding SCL low until SCOMP has received the data using the assembly instruction IN I2C_CONTROLLER. If the option to request that a byte be resent is desired, then it will be unnecessary for the I^2C controller to have a buffer. After receiving a byte, the I^2C controller would stretch the clock either before or after the master pulses the SCL to read the ACK. The I^2C controller has no ability to check whether the data received is correct, so it would have to wait for the assembly to read in the data received over the I^{2}C bus, check the data, and send a new command, OUT I2C_ACK, back to the I^{2}C controller. The OUT I2C_ACK command sends the value in the accumulator to the I^{2}C controller and would be expected to be 1 for acknowledge and 0 for not acknowledge (NACK). Once an acknowledge decision has been made by the assembly, the I^{2}C controller will ACK or NACK, and then it will release the SCL causing the clock to stop stretching.

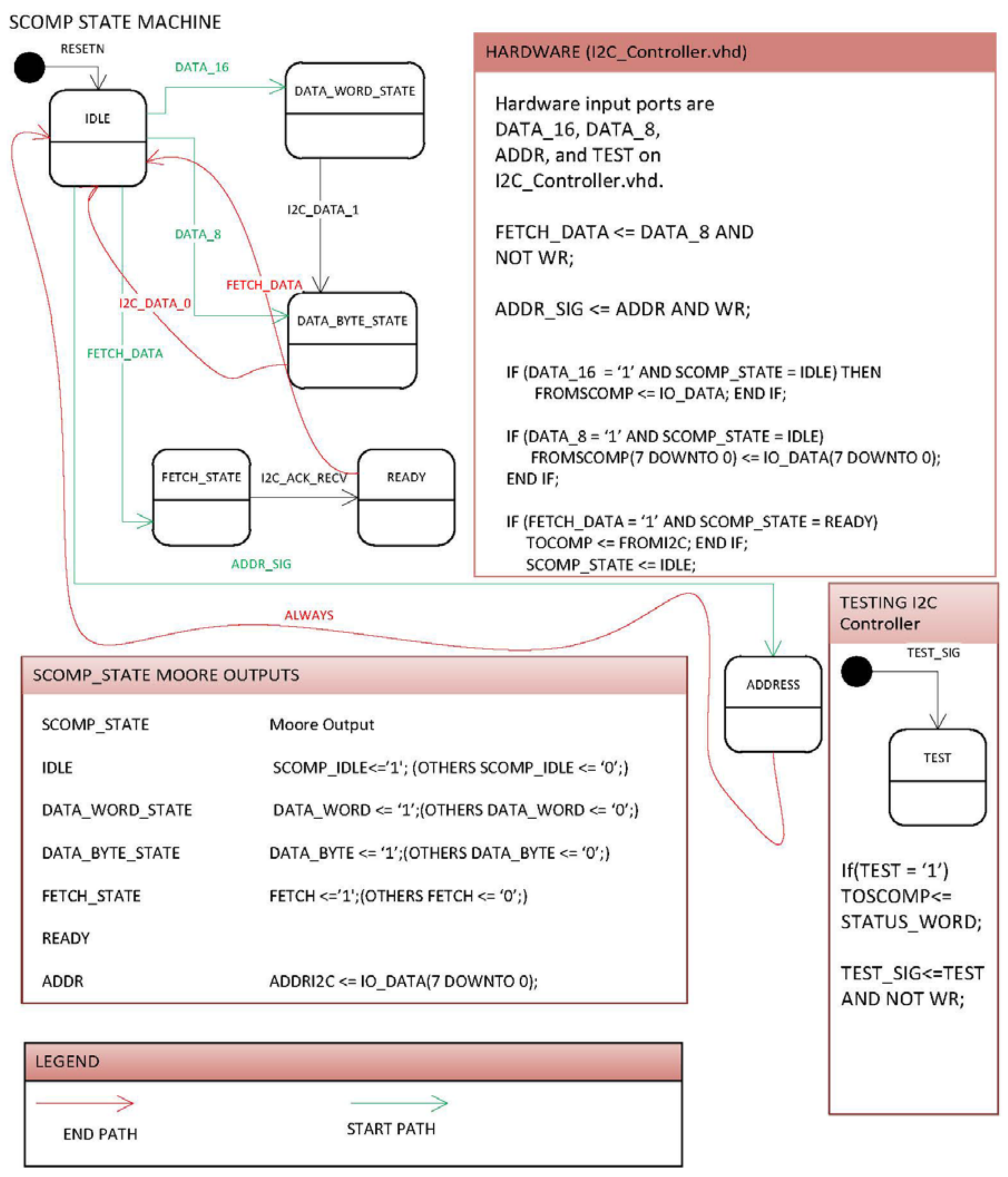
Appendix A: Gantt Chart



ID	Task Name
1	**1 Review Project Information**
2	**1.1 Review I2C Controller**
3	1.1.1 Read I2C Exerpt
4	1.1.2 Read I2C Specs
5	**1.2 Review Given SCOMP Code**
6	1.2.1 Read DE2 Board Spec
7	1.2.2 Read DE2 Manual
8	1.2.3 Read Wolfson WM8731
9	**2 Group Logistics**
10	2.1 Group Formation
11	2.2 Exchange Contact Information
12	2.3 Set-up Shared Calendar
13	2.4 Set-up Shared Docs
14	**3 Project Brainstorming**
15	3.1 Propose a SCOMP Program Design
16	**3.2 Develop Datapath**
17	3.2.1 SCOMP to I2C Controller
18	3.2.2 I2C Controller to I2C Device
19	**4 Desgin Proposal**
20	4.1 Management Plan
21	4.2 Executive Summary and Intro
22	4.3 Technical Approach
23	**5 Project Coding**
24	**5.1 Code SCOMP to I2C instuctions**
25	5.1.1 Develop .vhdl File
26	5.1.2 Develop .asm Program
27	**5.2 Code I2C Controller to Audio Codec**
28	5.2.1 Add Address and Instuctions
29	6 Demo Project
30	**7 Improve Design**
31	7.1 Optimise .vhdl
32	7.2 Optimise .asm
33	**8 SCOMP Design Program**
34	**8.1 Evaluate process**
35	8.1.1 What Needs to be Added
36	8.1.2 How to integrate
37	**8.2 Update SCOMP Program**
38	8.2.1 Add .vhdl code
39	8.2.2 Add .asm code
40	**8.3 Test**
41	8.3.1 Evaluate program
42	**9 Prepare Presentation**
43	9.1 Develop slide content
44	9.2 Practice Presentation
45	**10 Project Design Demo**
46	10.1 Obtain results for presentation
47	11 Present Design Solution
48	**12 Final Report**
49	12.1 Compile Content
50	12.2 Turn in Report

Figure A.1. The Gantt chart outlining team’s tasks and their dates.

Appendix B: UML Statecharts



HARDWARE (I2C_Controller.vhd)

Hardware input ports are DATA_16, DATA_8, ADDR, and TEST on I2C_Controller.vhd.

```
FETCH_DATA <= DATA_8 AND
NOT WR;

ADDR_SIG <= ADDR AND WR;

IF (DATA_16  = '1' AND SCOMP_STATE = IDLE) THEN
    FROMSCOMP <= IO_DATA; END IF;

IF (DATA_8 = '1' AND SCOMP_STATE = IDLE)
    FROMSCOMP(7 DOWNTO 0) <= IO_DATA(7 DOWNTO 0);
END IF;

IF (FETCH_DATA = '1' AND SCOMP_STATE = READY)
    TOCOMP <= FROMI2C; END IF;
    SCOMP_STATE <= IDLE;
```

SCOMP_STATE MOORE OUTPUTS

SCOMP_STATE	Moore Output
IDLE	SCOMP_IDLE<='1'; (OTHERS SCOMP_IDLE <= '0';)
DATA_WORD_STATE	DATA_WORD <= '1';(OTHERS DATA_WORD <= '0';)
DATA_BYTE_STATE	DATA_BYTE <= '1';(OTHERS DATA_BYTE <= '0';)
FETCH_STATE	FETCH <='1';(OTHERS FETCH <= '0';)
READY	
ADDR	ADDRI2C <= IO_DATA(7 DOWNTO 0);

```
If(TEST = '1')
TOSCOMP<=
STATUS_WORD;

TEST_SIG<=TEST
AND NOT WR;
```

Figure B.1. UML statechart for SCOMP state machine.

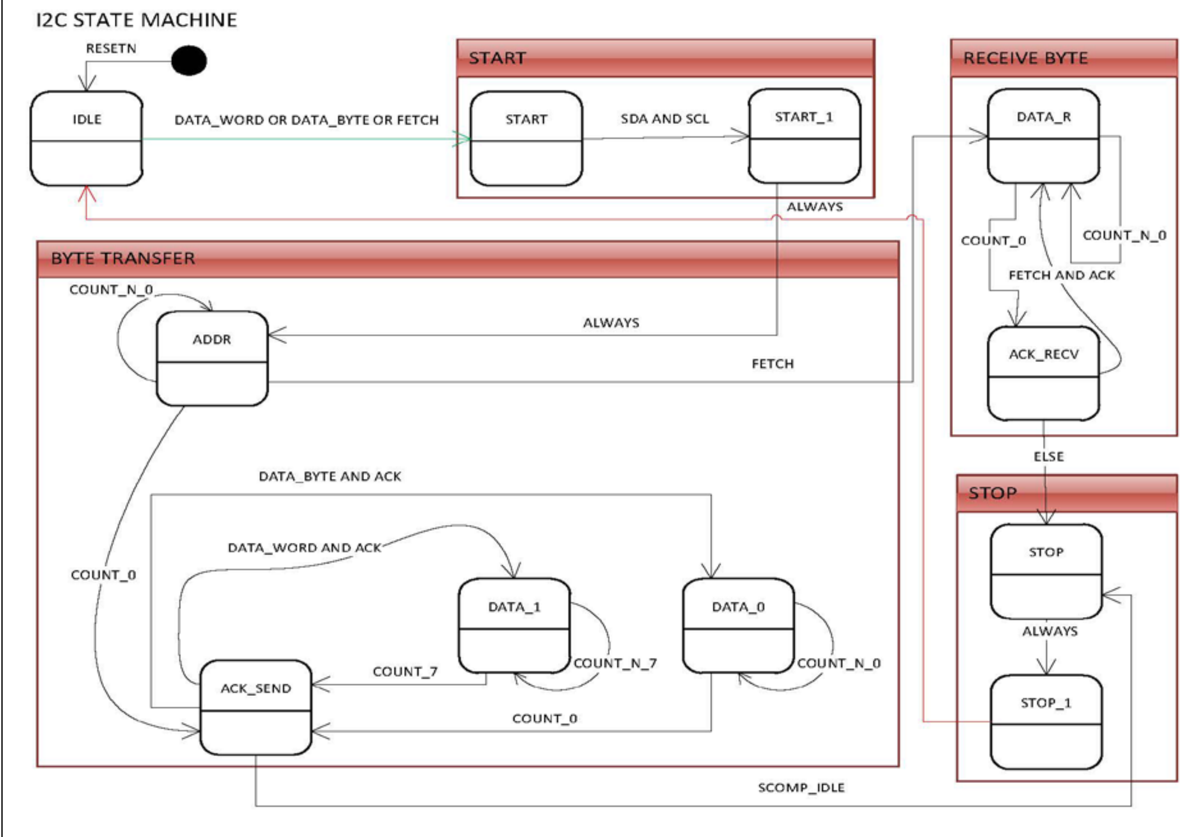


Figure B.2. UML statechart for I^{2}C state machine.

Appendix C: Schematic of I^2C Controller Implementation

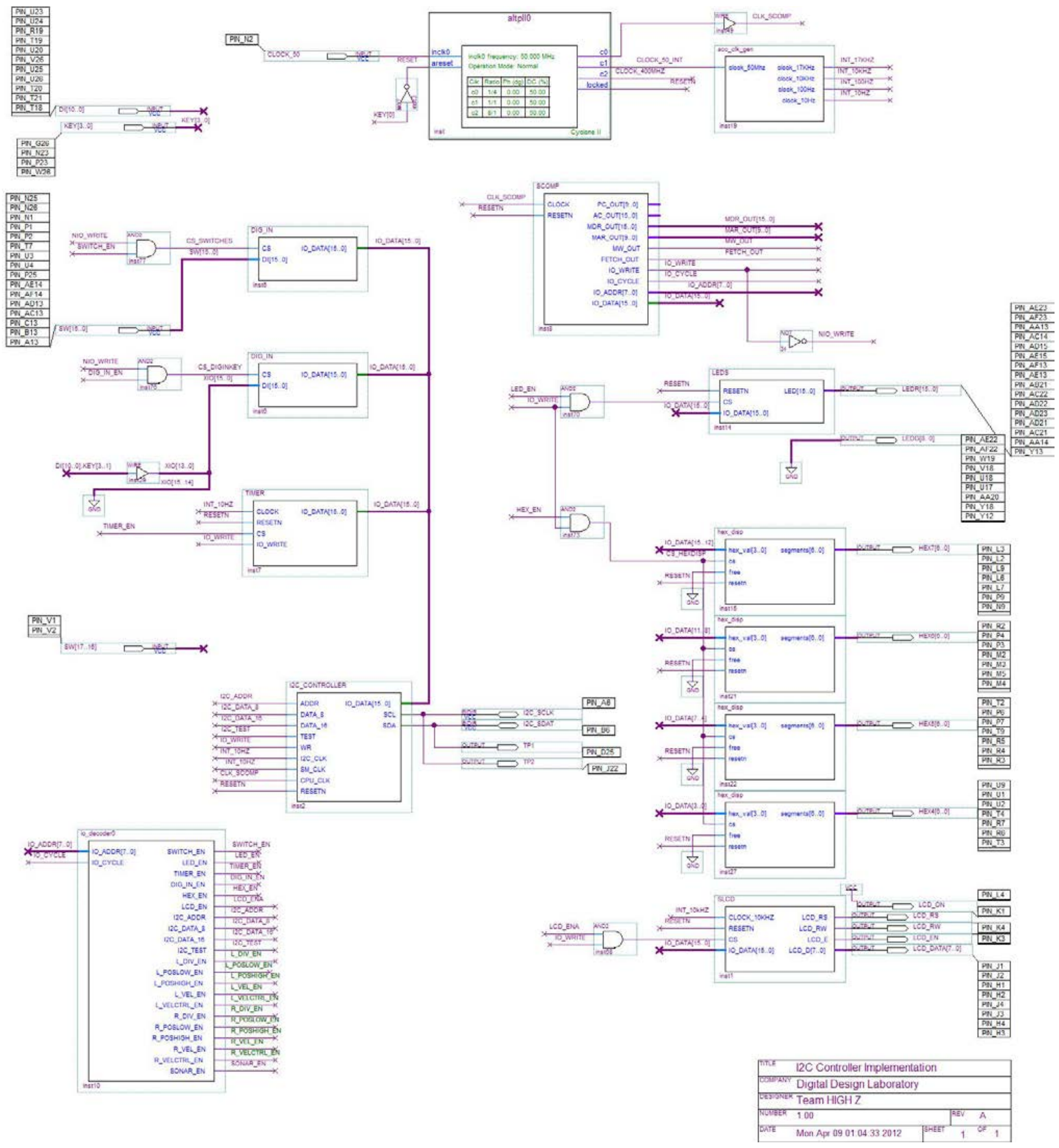


Figure C.1. Circuit schematic showing all the pin assignments and connections between all the devices used in the implementation of the karaoke machine, namely, the I^{2}C controller, the SCOMP, the LEDs , the keys and the switches.