The George Washington University

Department of Computer Science

CS339 Final Project

Project Final Report

Z8 Speaks

Apr. 25, 2006

Duoduo Liao

dliao@gwu.edu

Table of Contents

Project Abstract 4
Project Abstract 4
Project Status 4
1 Design Overview 4
1.1 The Purpose and Requirements 4
1.2 The Hardware Design 5
1.2.1 The hardware block diagram 5
1.3 The Software Design 7
2 Specifications 8
2.1 Hardware Modules 8
2.1.1 Z8 Encore!™ Flash Microcontroller Development Kit 9
2.1.2 VOICEDIRECT Speech Recognition Kit 9
2.1.3 SpeakJet Speech Synthesis Chip 12
2.1.4 DS1307 Real-Time Clock (RTC) Module 13
2.1.5 MAX6610 Temperature Sensor 14
2.2.6 Parts List 15
2.2 Software Modules 15
3 Implementation & Construction 16
3.1 Speech Recognition 17
3.1.1 VOICEDIRECT Board Stand-Alone Test 17
3.1.2 VOICEDIRECT Driver Development for Z8 19
3.2 Speech Synthesis 20
3.2.1 SpeakJet Chip Stand-Alone Test 20
3.2.2 SpeakJet Driver Development for Z8 21
3.3 Real-time Clock Reading/Setting 23
3.3.1 I^2C Protocol Control 23
3.3.2 Data Transfer 24
3.3.3 Time/Calendar Reading/Setting 25
3.4 Temperature Driver and Reading 26
3.5 Phrase Allophone Editing 27
3.5.1 Temperature Allophone Editing 28
3.5.2 Time/Calendar Allophone Editing 28
3.6 System Integration and Main Application Flow Chart 29
4 Conclusions 31
5 Attachments List 32

Table of Figures

Figure 1: The Block Diagram of Hardware Design 5
Figure 2: The Schematic for the Design 7
Figure 3: The Block Diagram of the Software Design 8
Figure 4: Main Hardware Components of Z8 Speaks 9
Figure 5: VOICEDIRECT Board (front and back) 10
Figure 6: VOICEDIRECT Pinout 10
Figure 7: SpeakJet Chip (Left) and Pinout (Right) 12
Figure 8: SpeakJet Pin Details and Electrical Specifications 13
Figure 9: RTC Module (Left) and RTC Module Schematic (Right) 13
Figure 10: DS1307 Address Map and Timekeeper Registers 14
Figure 11: MAX6610 Temperature Sensor Configuration 14
Figure 12: The Final Hardware Connection for the Z8 Speaks System 16
Figure 13: Stand-Alone Test for VoiceDirect Hardware 18
Figure 14: Hardware Connection for VoiceDirect Communication with Z8 19
Figure 15: Hardware Test for SpeakJet Demo Mode 21
Figure 16: The Schematic for SpeakJet Communication with Z8 21
Figure 17: DS1307 Data Transfer 24
Figure 18: The Flow Chart of Z8 Speak Main Application 30

Project Abstract

This final project report describes in detail how to create a Z8 speaks like a talking robot. The Z8 Speaks is mainly composed of Z8 microcontroller board, VoiceDirect speech recognition board, SpeakJet speech synthesis chip, real-time clock, and temperature sensor. The user can interact with the Z8 Speaks through speech communication based on speech recognition and synthesis. It has the capabilities to display current temperature, time, calendar, name, music, reset, and other information on the LEDs in the Z8 board upon spoken request as well as speaking corresponding information. This project provides twelve different operations, such as temperature, time, calendar, reset, etc., which have been already set well in the program. Before the user talks with the robot, his or her voice commands need to be trained correspondingly for the preset operations and stored as speech patterns. Then the user can ask the questions for the robot. If the voice does not match any of trained voice commands, a speech instruction – "words cannot be recognized" or "Repeat, look for" will be given to allow the user to try again.

Project Status

The overall work for the final project works very well as planned in the project proposal. Specifically, the final project meets the proposed requirements as follows,

- Read the temperature from ADC temperature sensor
- Read the Real-Time Clock (RTC) using I2C bus
- Control the speech recognition board using GPIO and hardware switches
- Control speech synthesis through serial data line
- Interact with the user over the speech communication

In addition, more voice operations and speech controls are added into the final project beyond the proposal as follows,

- Write/Reset the RTC using I2C bus
- Software decoding to reach up to 15 word commands for speech recognition
- Allophone phase editing
- Numerical pronunciation (0 – 69)
- Calendar pronunciation (Jan. 1 – Dec. 31, Monday - Sunday)
- Non-standard language speech synthesis (Chinese, AM/PM)
- Robot music playing

Project Final Demonstration Video (13MB):
http://home.gwu.edu/~dliao/cs339/Liao_z8Speaks_320x240.mpg

1 Design Overview

1.1 The Purpose and Requirements

The purpose of this project is to create a Z8 speaks. It has the capabilities to display current temperature, real-time date, day, time, name, and other information on the LEDs in the Z8 board upon spoken request as well as speaking corresponding information. The

Z8 Speaks mainly consists of Z8 microcontroller board, VOICEDIRECT speech recognition board, SpeakJet speech synthesis chip, real-time clock, and temperature sensor. The Z8 microcontroller will control temperature reading, RTC reading/writing, speech recognition, speech synthesis and output.

The proposed requirements that the project must meet are:

- Read the temperature from ADC temperature sensor
- Read the Real-Time Clock (RTC) using I2C bus
- Control the speech recognition board using GPIO and hardware switches
- Control speech synthesis through serial data line
- Control LED display with corresponding information
- Interact with the user over the speech communication

1.2 The Hardware Design

1.2.1 The hardware block diagram

The block diagram of overall hardware design is shown in Figure 1 as follows.

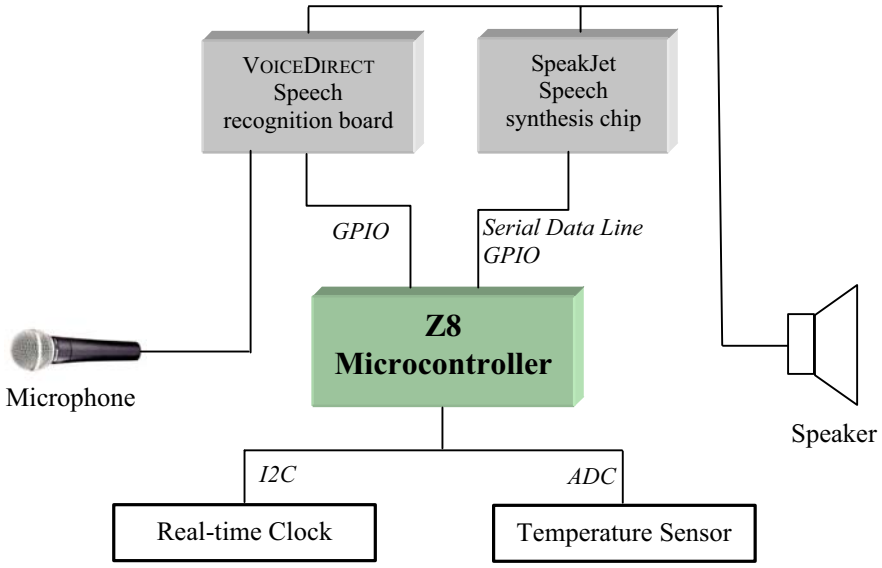
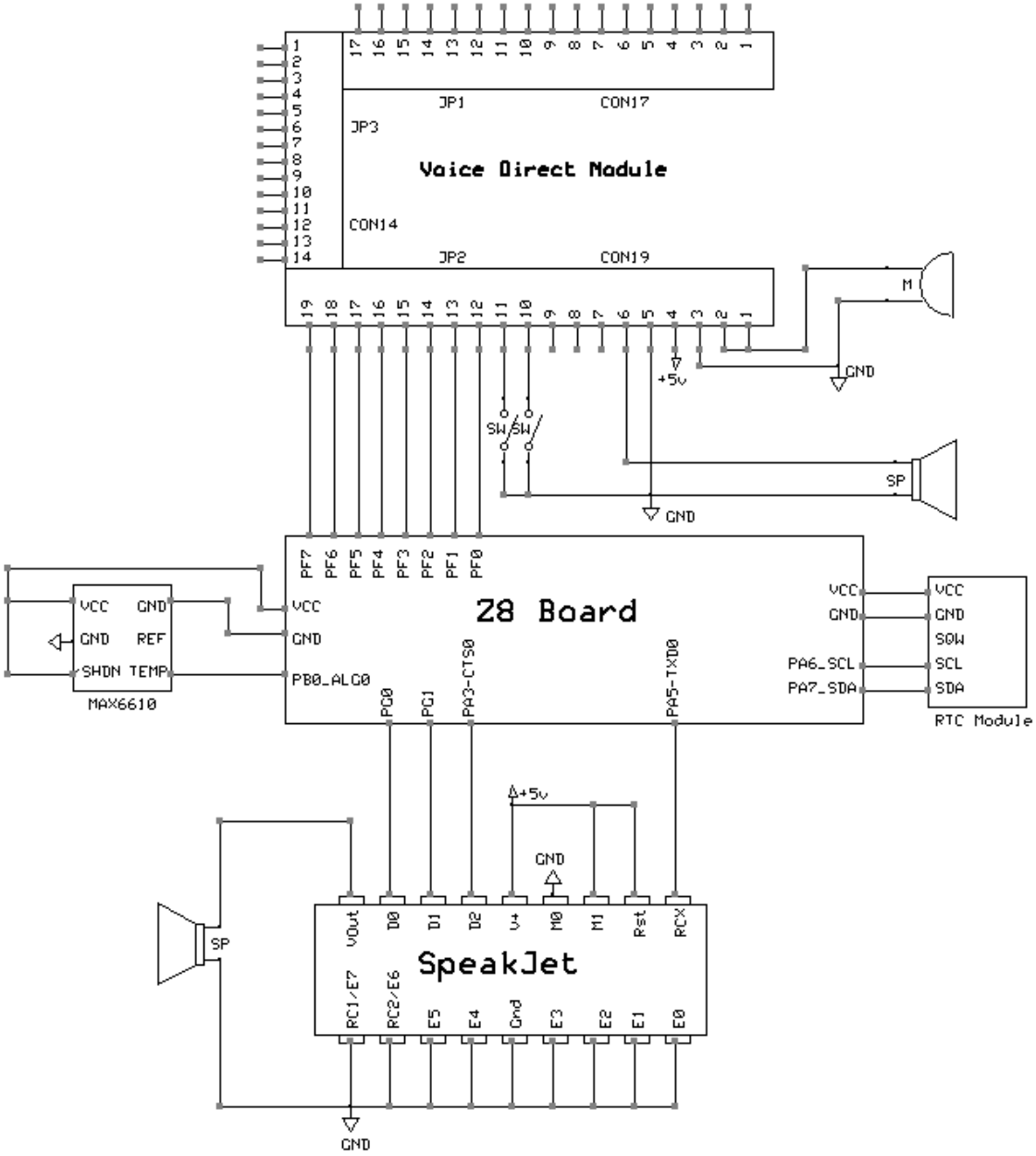


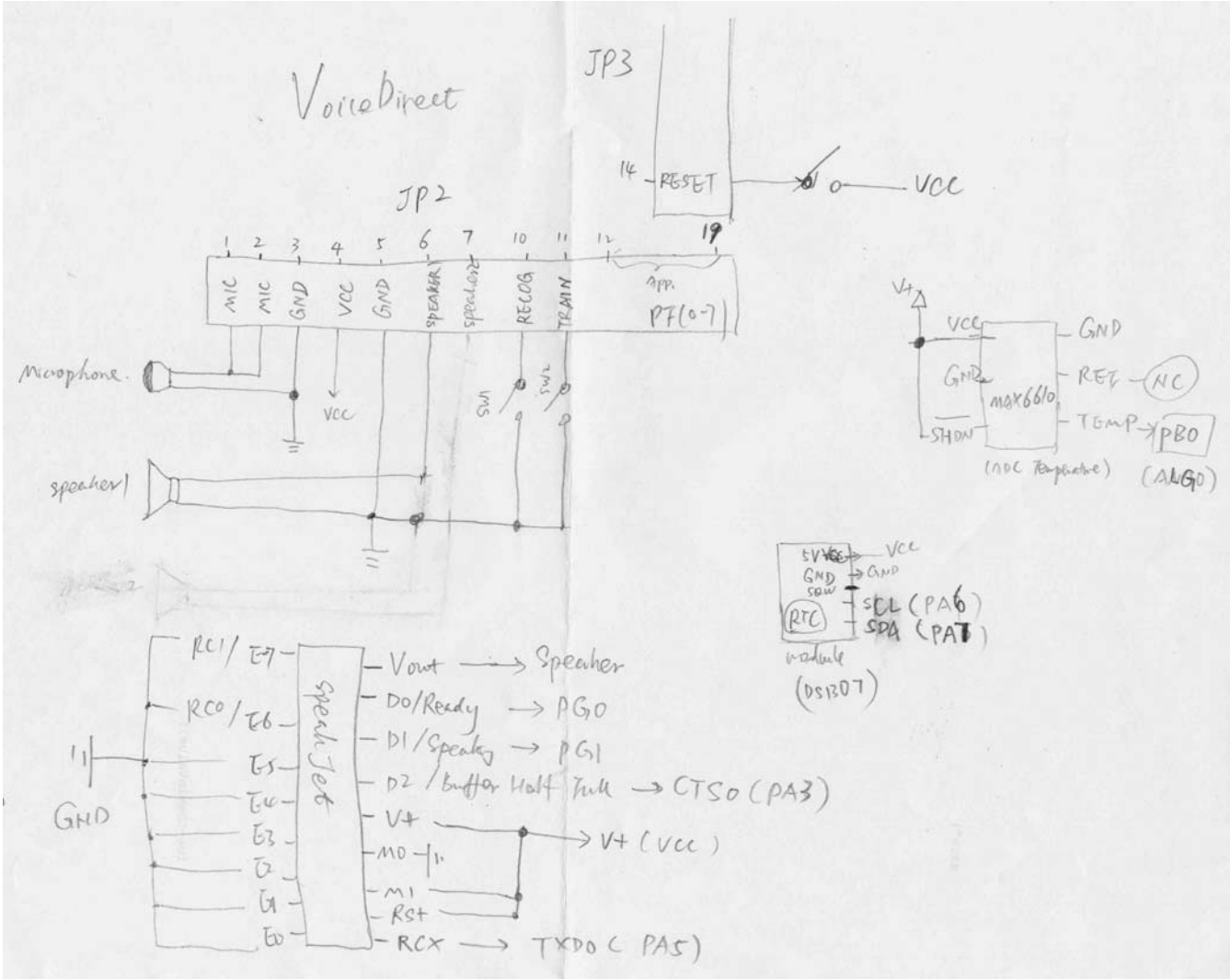
Figure 1: The Block Diagram of Hardware Design

1.2.2 The Schematic for the Design

The interfacing of the Z8 microcontroller to the VioceDirect board, SpeakJet chip, RTC module, MAX6610 temperature sensor, microphone, switches, and speakers is shown in the following schematic for the design in Figure 2.



(a) The Formal Schematic for the Design (Final Version)



(b) The Schematic Scratch (Work Progress)

Figure 2: The Schematic for the Design

1.3 The Software Design

The software is designed for four major modules, Speech Recognition Manager, Sensor Manager, Speech Synthesis Manager, and Driver Manger. The block diagram of the overall software design is shown in Figure 3 as follows.

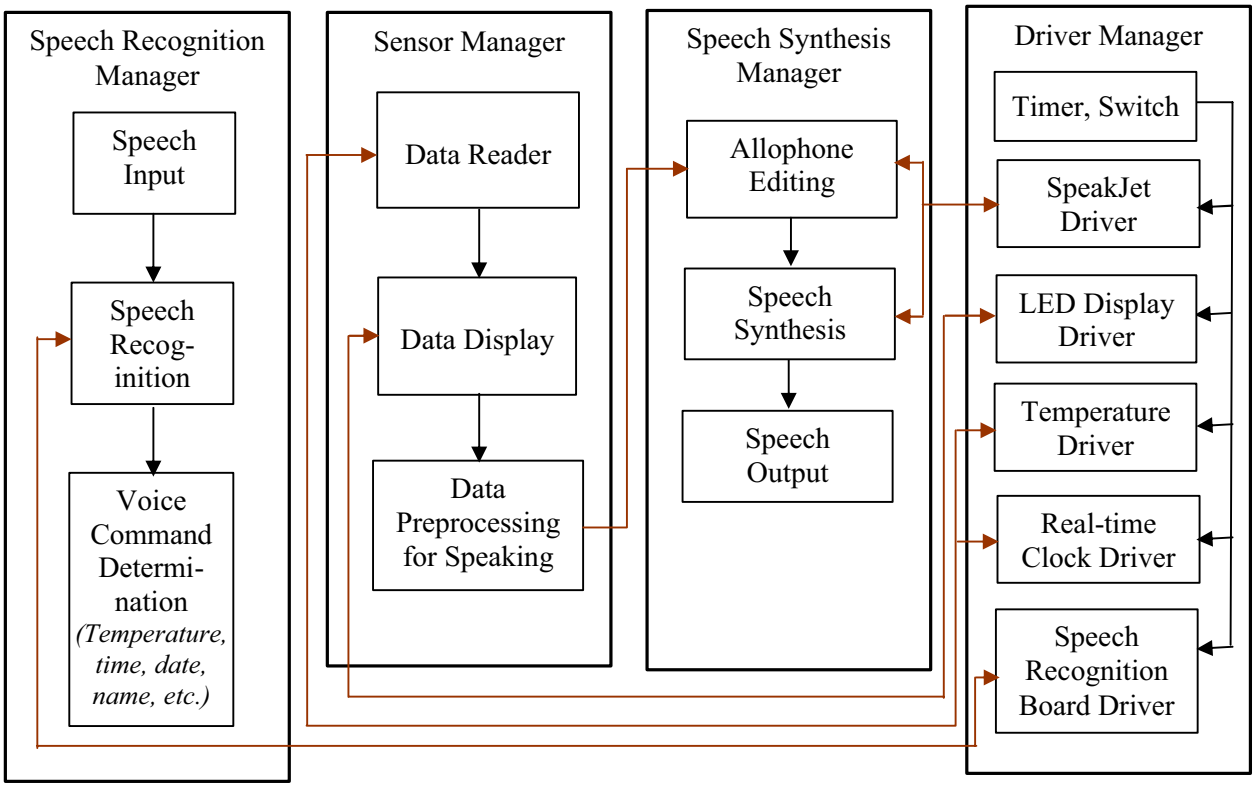


Figure 3: The Block Diagram of the Software Design

2 Specifications

2.1 Hardware Modules

The project design consists of five main hardware modules: Z8 Encore!™ Flash Microcontroller Development Kit, VOICEDIRECT speech recognition board, SpeakJet speech synthesis chip, Sparkfun DS1307 Real-Time Clock Module, and MAX6610 temperature sensor.

The main hardware components for Z8 Speaks are shown in Figure 4.

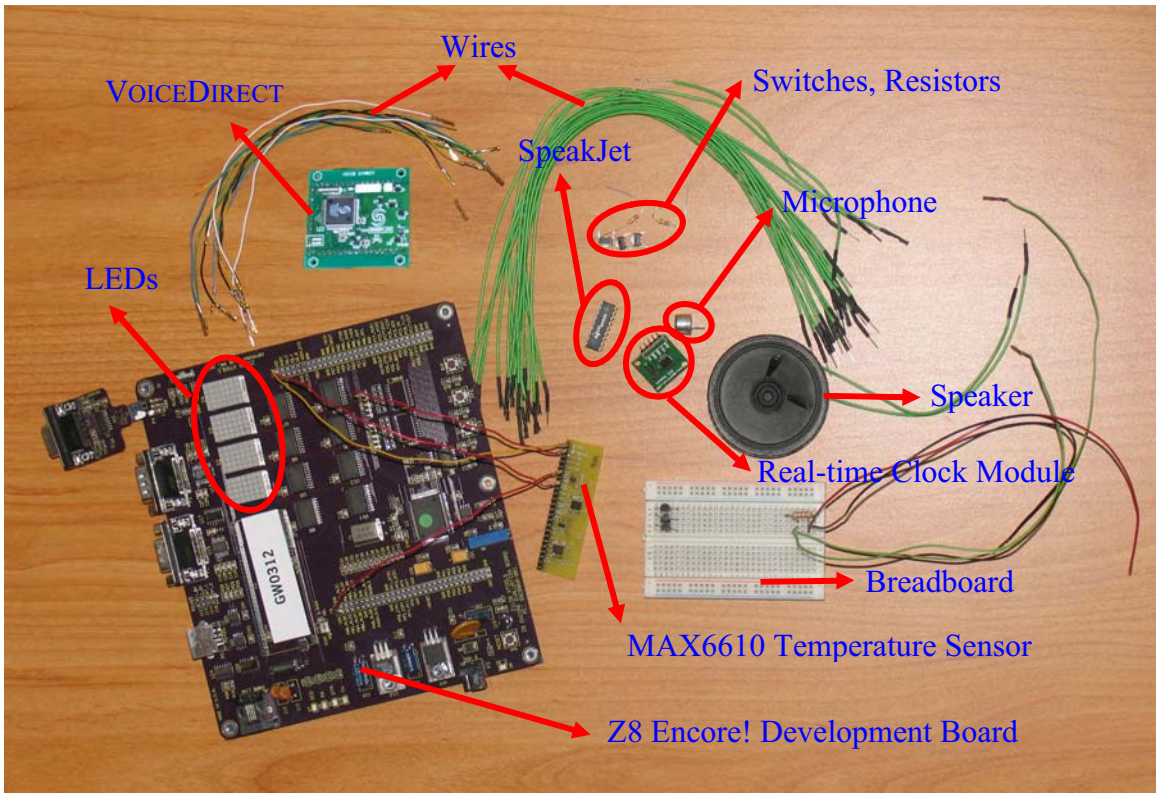


Figure 4: Main Hardware Components of Z8 Speaks

2.1.1 Z8 Encore!™ Flash Microcontroller Development Kit

The Z8 Encore!™ Flash Microcontroller Development Kit allows the user to design and evaluate projects using the eZ8 microcontroller. The kit contains a Z8F6423 module, which contains the Z8F6423 device running at 18.432 MHz, with 64 Kbytes of Flash memory and 4 Kbytes of register RAM. This evaluation board provides 12-channel 10-bit A/D converter, four 16-bit timers, a watch-dog timer, 60 General-Purpose I/Os (GPIO), and 24 vectored interrupts. The board also contains SPI, I2C, and 2 UART ports with IrDA encode/decoder, 3-channel DMA controller, four 7x5 LED arrays, three pushbuttons, and embedded modem socket. Furthermore, it contains Zilog's proprietary ZDS II for debugging and programming. The adapter provides 9VDC for the evaluation board. The board supports 3.0-3.6V operating voltage with 5V-tolerant inputs.

2.1.2 VOICEDIRECT Speech Recognition Kit

VOICEDIRECT Speech Recognition Kit includes speech recognition board (as shown in Figure 5), one microphone, one speaker, three microswitches, and two 100KOhms resistors.

VOICEDIRECT is a speaker-dependent speech recognition module, allowing training of up to 15 words. Using sophisticated speech recognition technology, VOICEDIRECT maps spoken commands to system control functions. Each time one of the words is recognized, a corresponding output pin on the module is toggled for one second.

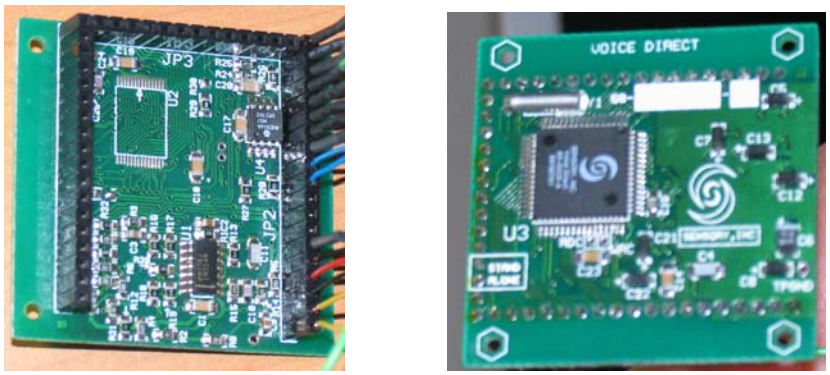


Figure 5: VOICEDIRECT Board (front and back)

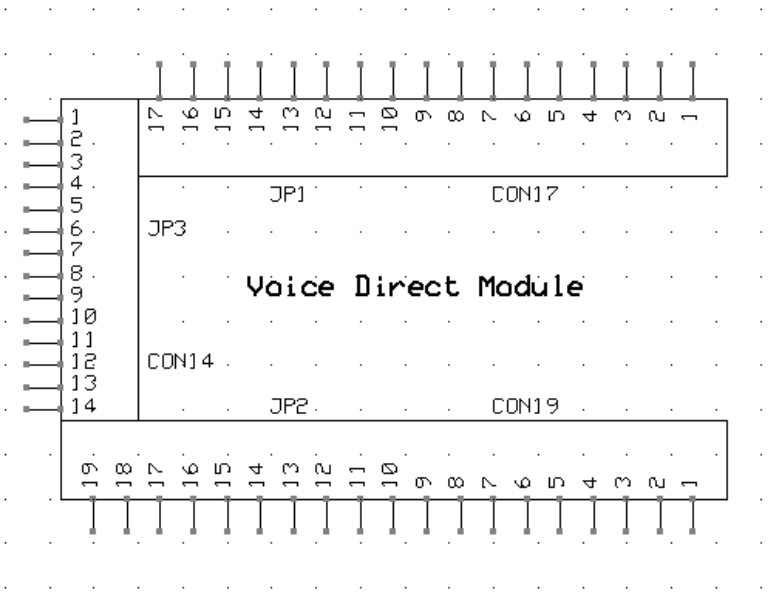


Figure 6: VOICEDIRECT Pinout

The module pinout layout is shown in Figure 6 and major module pinout used for this project is listed in the Table 1 as follows.

Table 1: Major Module Pinout

Name	Module Pin	Description	Connection		I/O
PREAMP IN	JP2 - 1	Microphone Input Connection	Connect to microphone, other microphone connection to GND		I
MIC BIAS	JP2 - 2	Mic Bias (Elec. Microphone)	If powered mic, being used, NC, otherwise connect JP2-1		I
AGND	JP2 – 3, 5	Analog Ground. For noise reasons, analog and digital grounds should connect together only at VOICE DIRECT.	GND		
-+5V	JP2 - 4	5 Volt(+) Power Supply Connection	VCC		-
PWM1	JP2 - 6	Pulse Width Modulator Output 1 (multiplexed)	Connect to 8–32Ohm speaker. Provides approximately 0.15 Watts of audio power into 32-Ohms.		O
PWM0	JP2 - 7	Pulse Width Modulator Output 0	Connect to 8–32Ohm speaker. Provides approximately 0.15 Watts of audio power into 32-Ohms.		O
DAOUT	JP2 - 8	Analog Output (unbuffered)	High-impedance (22kOhm) analog audio output. Must be powered amplified to drive a speaker, and should be low-pass filtered with a corner frequency around 20KHz. Better speech quality than the PWM output. Recommended for applications requiring either louder volume or better speech quality.		O
-RECOG	JP2 - 10	Recognition sensitivity selection and active recognition	To start recognition, pull the –RECOG line to GND for at least 100ms. To erase all recorded words, pull both the –TRAIN and –RECOG pins to GND for at least 100ms.	To erase all recorded words, pull both the –TRAIN and –RECOG pins to GND for at least 100ms.	I
-TRAIN	JP2 - 11	Training sensitivity selection and active training	To start training, pull the –TRAIN line to GND for at least 100ms.		I
OUT1-OUT7	JP2 – 12-18	Stand Alone mode output ports 1-7	Connect to user application's control lines.		O
HIGH/OUT8	JP2 - 19	Stand Alone mode output ports 8	Connect to user application's control lines.		O

In this project, -TRAIN open circuit mode is used for pin configuration. This is relaxed training mode – easier to train, accepts more similar sounding words (i.e., fewer rejections).

Since the module can recognize 15 words, but only has 8 output pins, word 9 through 15 are represented in binary form, as show in the Table 2 on the right. Software decoding method instead of hardware decoding circuits is applied to get word 9 and word 15.

Table 2: 15 Words and Corresponding Pinout

Word 1	Output 1
Word 2	Output 2
Word 3	Output 3
Word 4	Output 4
Word 5	Output 5
Word 6	Output 6
Word 7	Output 7
Word 8	Output 8
Word 9	Output 8 and Output 1
Word 10	Output 8 and Output 2
Word 11	Output 8 and Output 3
Word 12	Output 8 and Output 4
Word 13	Output 8 and Output 5
Word 14	Output 8 and Output 6
Word 15	Output 8 and Output 7

2.1.3 SpeakJet Speech Synthesis Chip

The SpeakJet is a completely self contained, single chip voice and complex sound synthesizer as shown in Figure 7. SpeakJet uses Mathematical Sound Architecture (MSA) technology which controls an internal five channel sound synthesizer to generate on-the-fly, unlimited vocabulary speech synthesis and complex sounds without the use of analog or digitally recorded samples. The SpeakJet has a built in library of 72 speech elements (allophones), 43 sound effects, and 12 DTMF Touch Tones. Through the selection of these MSA components and in combination with the control of the pitch, rate, bend, and volume parameters, the user has the ability to produce unlimited phrases and sound effects, with thousands of variations, at any time.

The SpeakJet can be controlled simultaneously by logic changes on any one of its eight Event Input lines, and/or by a Serial Data line from a CPU (such as Z8 or PC) allowing for both CPU-Controlled and Stand-Alone operations. Other features include an internal 64 byte input buffer, internal programmable EEPROM, three programmable outputs, and direct user access to the internal five channel sound synthesizer.

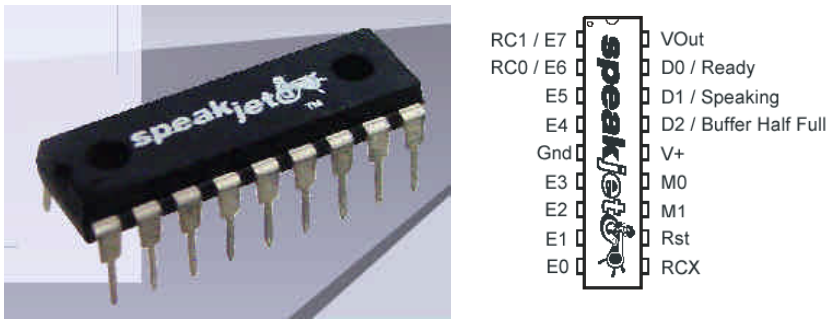


Figure 7: SpeakJet Chip (Left) and Pinout (Right)

The SpeakJet pin details and electrical specifications are shown in Figure 8.

Pin #	Description	Functional Details
1	RC1/E7	Event Input 7
2	RC0/E6	Event Input 6
3	E5	Event Input 5
4	E4	Event Input 4
5	Gnd	Ground
6	E3	Event Input 3
7	E2	Event Input 2
8	E1	Event Input 1
9	E0	Event Input 0
10	RCX	Serial Input TTL (0.0v to Vcc)
11	Rst	Master Reset
12	M1	Mode Select 1 (Baud Configure)
13	M0	Mode Select 0 (Demo Mode)
14	V+	Power input +2.0 to +5.5 volts DC
15	D2/Buffer Half Full	Data Out 2 (External) / Buffer Half Full (Internal)
16	D1/Speaking	Data Out 1 (External) / Speaking (Internal)
17	D0/Ready	Data Out 0 (External) / Ready (Internal)
18	VOut	Voice Output

Electrical Specifications:

Supply voltage	2.0 to 5.5 VDC
Supply Current:	
Idle:	<5ma. Plus loads
Speaking:	<5ma. Plus loads
Sink/Source Current:	
Outputs	25ma.
All Inputs levels:	
High	Supply
Low	GND
EEPROM:	
Max. Write cycles	Typical 1,000,000 times

Mechanical Specification:

Thermal storage:	-60 to +140 Degrees C
Thermal operating	-18 to +60 Degrees C

The thermal specifications are preliminary and may change as testing is completed.

Figure 8: SpeakJet Pin Details and Electrical Specifications

In the final project, Event Input E0-E7 is not used and must be connected to GND. The Serial Input, RCX, is used to communicate between SpeakJet and external devices such as Z8. PA5-TXD0 on Z8 board is connected to RCX. Voice Output, VOut, modulates the SpeakJet's voice on a square wave carrier of 32khz.

2.1.4 DS1307 Real-Time Clock (RTC) Module

This is a custom-designed module for the DS1307 Real Time Clock as shown in Figure 9. The module comes fully assembled and pre-programmed with the current time. The included Lithium coin cell battery (CR1225 41mAh) will run the module for a minimum of 9 years (17 years typical) without external 5V power. The DS1307 is accessed via the I^2C protocol. It provides seconds, minutes, hours, AM/PM, day, month, date, year, leap year compensation, accurate calendar up to year 2100, 1Hz output pin, and 56 Bytes of non-volatile memory available to user.

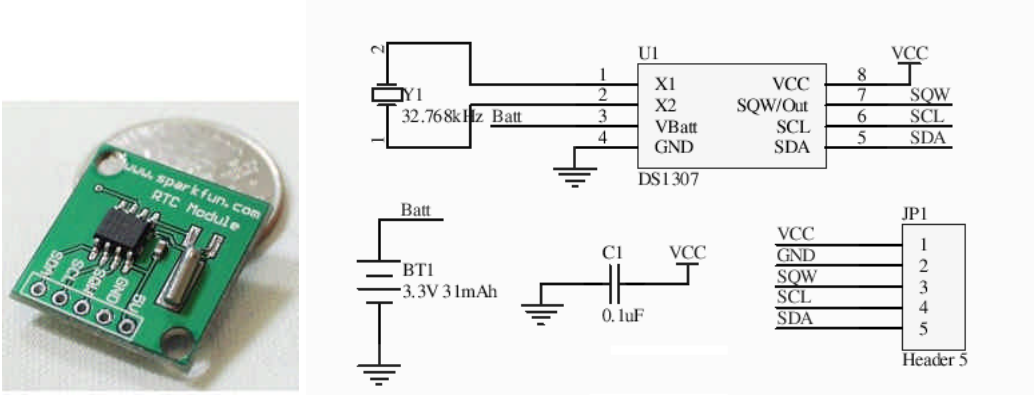


Figure 9: RTC Module (Left) and RTC Module Schematic (Right)

V_{BAT} is a battery input for any standard 3V lithium cell or other energy source. Vcc is the +5V input. When 5V is applied within normal limits, the device is fully accessible and data can be read and written. When a 3V battery is connected to the device and Vcc is below 1.25xV_{BAT}, reads and writes are inhibited. However, the timekeeping function continues unaffected by the lower input voltage. SCL (Serial Clock Input) is used to synchronize data movement on the serial interface. SDA (Serial Data Input/Output) is the input/output pin for 2-wire serial interface. The SDA pin is open drain. SQW is used for square wave output.

The DS1307 Serial RTC is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via 2-wire, bi-directional I^2C bus. DS1307 address map and timekeeper registers are shown in Figure 10.

	BIT7							BIT0	
00H	CH	10 SECONDS			SECONDS				00–59
	0	10 MINUTES			MINUTES				00–59
	0	12/24	10 HR A/P	10 HR	HOURS				01–12 00–23
	0	0	0	0	0	DAY			1–7
	0	0	10 DATE		DATE				01–28/29 01–30 01–31
	0	0	0	10 MONTH	MONTH				01–12
	10 YEAR				YEAR				00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	

Figure 10: DS1307 Address Map and Timekeeper Registers

2.1.5 MAX6610 Temperature Sensor

The MAX6610 is precise, low-power analog temperature sensors combined with a precision voltage reference as shown in Figure 11. An 8-bit ADC's LSB is equal to 1°C, while a 10-bit ADC's LSB corresponds to 0.25°C. The MAX6610 operates from 3.0V to 5.5V and has a 2.560V reference output. Power-supply current is less than 150µA. The MAX6610is available in a 6-pin SOT23 package and operate from -40°C to +125°C.

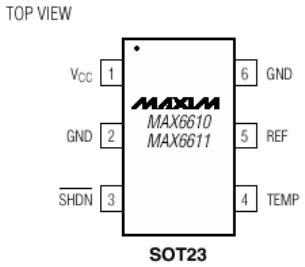


Figure 11: MAX6610 Temperature Sensor Configuration

2.2.6 Parts List

Table 3 contains the part numbers used for this project.

Part Name	Quantity	Description
Z8 Encore! Development Kit	1	Development package
VoiceDirect Board	1	Speech recognition board
SpeakJet	1	Speech synthesis chip
Switch	2	Control speech recognition and Training
SparkFun Real-time Clock Module	1	Real-time clock and calendar
MAX6610 Analog Temperature Sensor	1	Analog temperature sensor
Microphone	1	For speech recognition
Speakers	2+	For speech recognition and synthesis output
Breadboard	1	For prototyping design and test
Wires	many	For connection

2.2 Software Modules

As shown in Figure 3, all hardware control and application software modules listed in the block diagram of the software design are written by myself in C based on Z8 SDK. All of them are compiled on Zilong XTools ZDS II – Z8 Encore! Family 4.9.6 (build 05110402). ZDS II runs on the Windows XP platform. In addition, PhaseALator is used for some basic English words or phrases editing for SpeakJet Allophone code creation. The detailed software module development will be discussed in Section 3 Implementation & Construction.

3 Implementation & Construction

The hardware construction is exactly followed by the schematic for the design in Figure 2. Figure 12 shows the final Z8 Speaks system.

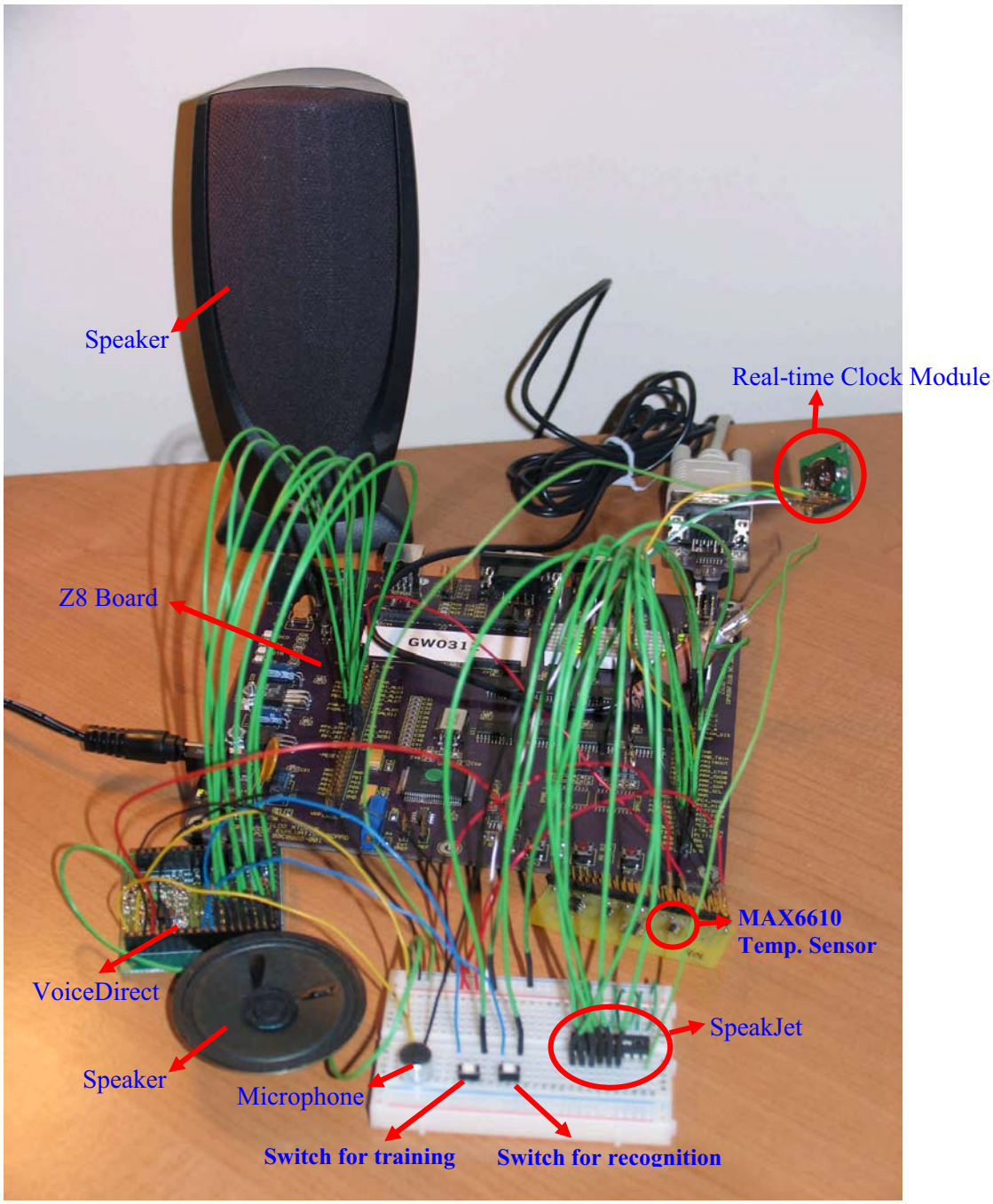


Figure 12: The Final Hardware Connection for the Z8 Speaks System

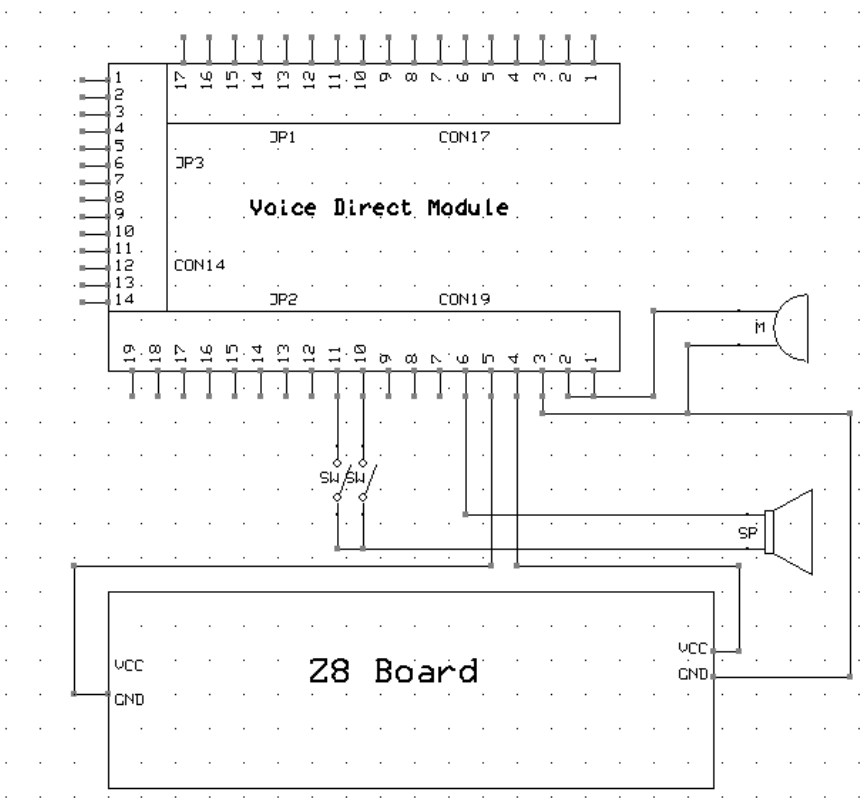
During the system development, for each major part or module needs to be tested on both hardware and software side, respectively. In the following sections, the main modules including hardware test and software development will be discussed in details. These main modules are speech recognition, speech synthesis, real-time clock reading and setting, temperature driver and reading, and phrase allophone editing. All other modules, such as LED display, timer control, in the software block diagram as shown in Figure 3 can be implemented as easily as in the basic real-time embedded system. Their details will not be given in this report. At the end of this section, the system integration and flow chart will be described.

3.1 Speech Recognition

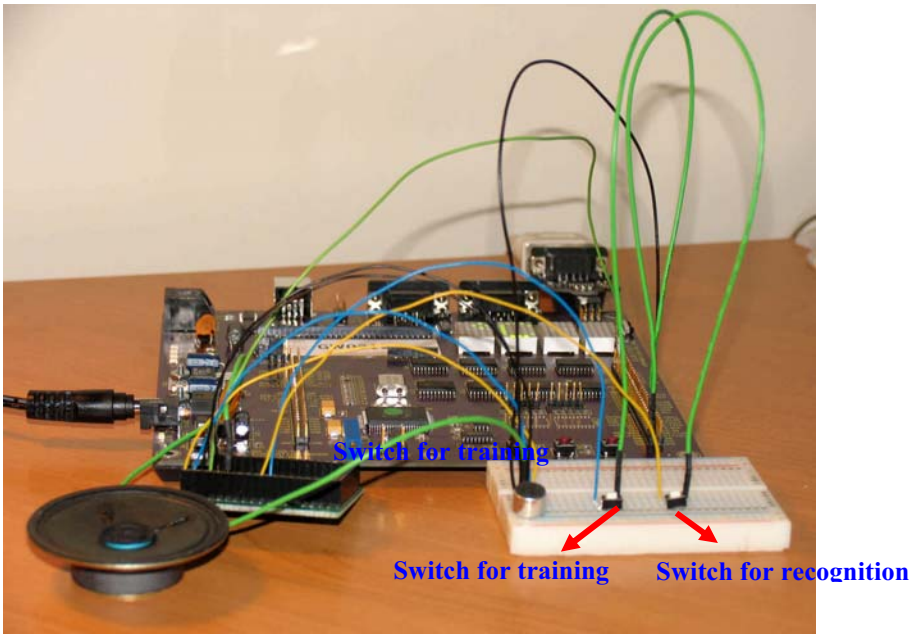
VOICEDIRECT board is used for speech recognition in this project. Firstly, hardware structure needs to be tested and verified. Then the driver for Z8 needs to be written for communication between VOICEDIRECT and Z8 microcontroller.

3.1.1 VOICEDIRECT Board Stand-Alone Test

VoiceDirect speech recognition board is connected in the stand-alone mode as shown Figure 13 (a) and (b). The Z8 in the figure is only used to be a power supply and ground connection. In (b), push left button at least 100ms, training will begin. VoiceDirect will prompt “Say word x” (where x is number from 1 to 15 corresponding to the word to be trained). After training, push right button, the VoiceDirect will prompt “Say a word”. If the word matches the training records, the VoiceDirect will return to say the number of the word record. Otherwise, it will return “Word not recognized”. Through this test, hardware configuration and connection for the VoiceDirect board can be verified.



(a) The Schematic for VoiceDirect Stand-Alone Test

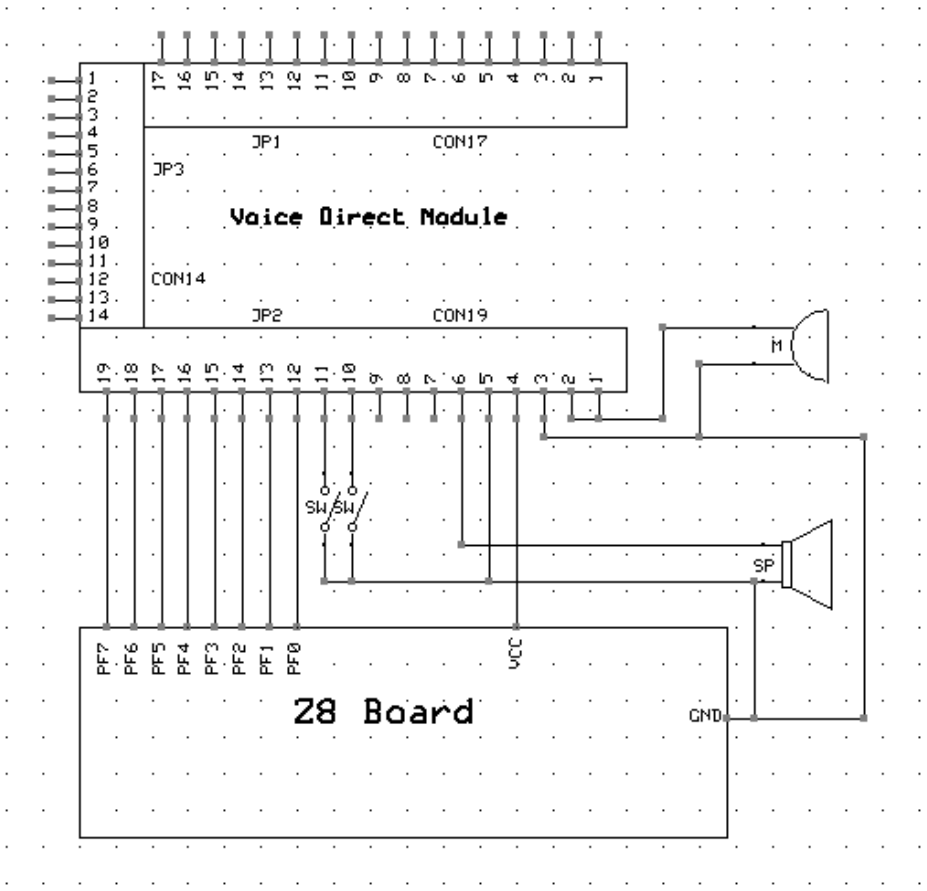


(b) The Hardware Connection for VoiceDirect Stand-AloneTest

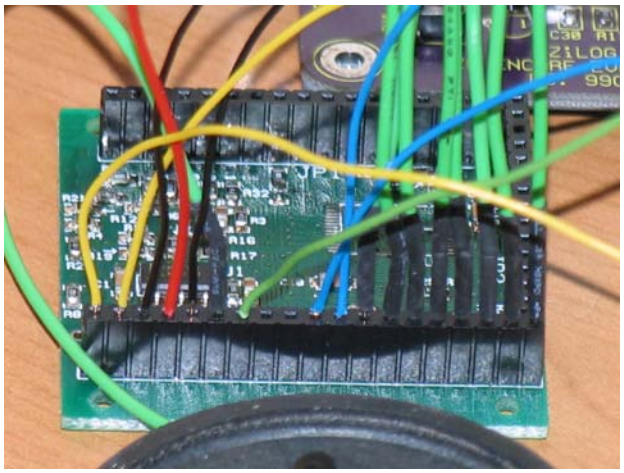
Figure 13: Stand-Alone Test for VoiceDirect Hardware

3.1.2 VOICEDIRECT Driver Development for Z8

The VoiceDirect can output the signals of OUT1-OUT8 to communicate with Z8 microcontroller through the connections to Z8 GPIOs. I use PF0-PF7 to connect VoiceDirect OUT1-OUT8 as shown in Figure 14.



(a) The Schematic for VoiceDirect Communication with Z8



(b) Hardware Connection for VoiceDirect Communication with Z8

Figure 14: Hardware Connection for VoiceDirect Communication with Z8

The following driver code snippets show how Z8 communicate with VoiceDirect. The total 15 words are represented in binary form as shown in Table 2 and decoded as follows.

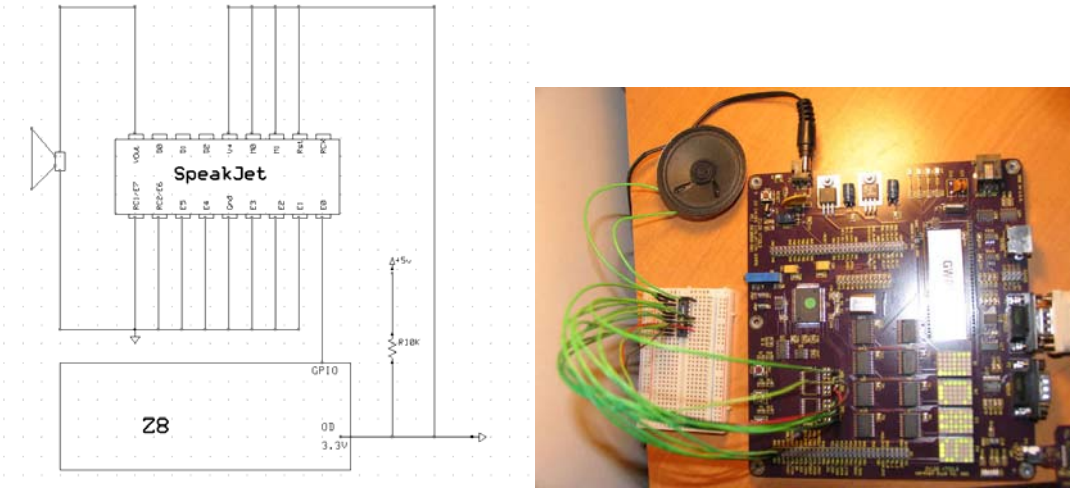
```
#define VOICE_COMMAND_1          0x01
#define VOICE_COMMAND_2          0x02
#define VOICE_COMMAND_3          0x04
#define VOICE_COMMAND_4          0x08
#define VOICE_COMMAND_5          0x10
#define VOICE_COMMAND_6          0x20
#define VOICE_COMMAND_7          0x40
#define VOICE_COMMAND_8          0x80
#define VOICE_COMMAND_9          0x81
#define VOICE_COMMAND_10         0x82
#define VOICE_COMMAND_11         0x84
#define VOICE_COMMAND_12         0x88
#define VOICE_COMMAND_13         0x90
#define VOICE_COMMAND_14         0xA0
#define VOICE_COMMAND_15         0xC0

// Initialize the ports for Voice Direct speech recognition board
void init_VoiceDirect(void)
{
        // initialize Port F(0-7)
        PFADDR = 0x02;
        PFCTL &= 0x00;          // no alterate function
        PFADDR = 0x01;          // data direction
        PFCTL &= 0x00;          // clear
        PFCTL |= 0xFF;          // input PF(0-7)
}

// Interpret the word code and take corresponding action
        switch(PFIN) {
        case VOICE_COMMAND_1:  action1(); break;
        case VOICE_COMMAND_2:  action2(); break;
        …
        case VOICE_COMMAND_15:  action15(); break;
        default: break;
        }
```

3.2 Speech Synthesis

SpeakJet chip is used for speech synthesis in this project. Firstly, hardware structure needs to be tested and verified. Then the driver for Z8 needs to be written for communication between SpeakJet and Z8 microcontroller.

3.2.1 SpeakJet Chip Stand-Alone Test

SpeakJet speech synthesis chip is connected in the demo (i.e. stand-alone) mode as shown Figure 15 (a) and (b). The Z8 in the figure is only used to be a power supply and ground connection. In such a mode, if all hardware works well, SpeakJet will play the demo sounds.



(a) The Schematic (b) The Hardware Connection

Figure 15: Hardware Test for SpeakJet Demo Mode

3.2.2 SpeakJet Driver Development for Z8

SpeakJet provides Serial Input and Event Inputs to communicate with Z8 microcontroller through Z8 GPIOs. In Figure 16, Z8's PG0 and PG1 connect to SpeakJet D0 and D1. CTS0 (PA3 alternate function) connects to D2. Serial data line (TXD0) in Z8 board is used to connect the RCX in SpeakJet. In this project, any Event Inputs are not used so all of them connect to GND.

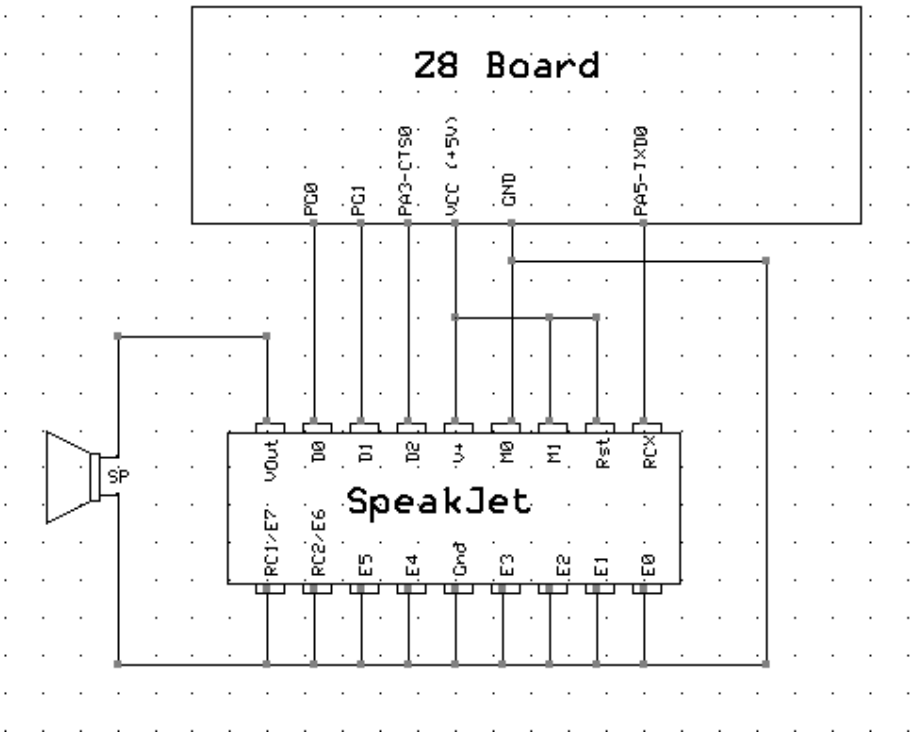


Figure 16: The Schematic for SpeakJet Communication with Z8

The SpeakJet can receive the commands that can be any of 256 commands listed in the Table D & E in the attachment A, SpeakJet User Manual. There are 7 operational groups of commands, SCP, Allophones, Sound Effects, DTMF, Pauses, Levels, and Controls. In this project, SCP commands are not be used. Thus, each command received is buffered into a 64-byte input buffer and executed by the MSA in FIFO.

Serial Data is the main command of communicating with the SpeakJet to execute commands or create voices and sounds. The SpeakJet serial configuration is fixed at 8bits, No-parity, and 1 stop bit and non-inverted. The SpeakJet can be configured to accept Baud rates from 2400 to 19200. The factory default setting is 9600 baud. In this project, default setting is used.

Therefore, the key for SpeakJet Z8 driver development is make serial data line TXD0 work. The driver code snippets are as follows,

```
#define FREQ 18432000                                   // 18.432MHz
#define BAUD1 9600                                      //  9.6K baud for UART0
#define BRG1 FREQ/((unsigned long)BAUD1*16)
#define UART_TXD_EN                     0x80

// initialize SpeakJet speech synthesis chip
// Initialize the ports
void init_SpeakJet(void)
{
        // initialize Port PA3 and PA5
        PAADDR = 0x02;
        PACTL |= 0x28;          // alterate function: PA3-CTS0, PA5-TXD0
        PAADDR = 0x01;          // data direction
        PACTL &= 0xD7;          // ouput:PA3-CTS0, PA5-TXD0

        U0BRH = (char)(BRG1 >> 8);
        U0BRL = (char)(BRG1 & 0xff);
        U0CTL0 = UART_TXD_EN;       // Transmit enable, No Parity, 1 Stop
}
// Input the speech string into SpeakJet through serial data line
void Speak(char *speech)
{
        puts(speech);
}
```

In this project, I directly use Allophones to edit any speech. For example, the following allophone code string represents the English speech “Welcome to Z8 Speaks”.

```
unsigned char s_welcome[LEN_WELCOME] = {252, 252, 147, 159, 194, 134, 140, 8, 191, 162, 8, 167, 128, 128, 154, 4, 191, 187, 198, 8, 128, 196, 187, 1};  // “Welcome to Z8 Speaks”
```

3.3 Real-time Clock Reading/Setting

SparkFun Real-time clock module is used for read/set the time or calendar in this project. RTC hardware specifications are described in Section 2.1.4. The schematics for SparkFun DS1307 RTC module is shown in Figure 9. The driver for communication between RTC and Z8 microcontroller contains the I^2C protocol, data transfer, and time/calendar reading or setting. These will be described in detail in this section.

3.3.1 I^2C Protocol Control

I^2C protocol is used for this module to communicate with Z8 microcontroller board. The following functions are used for I^2C communication. Note that DS1307 operates in the regular mode (100KHz) only.

```
//Intialize I2C Interface
void init_I2C(void)
{
        // BRG = 18432KHz/(4*100KHz) = 46 = 0x2E  [mode = 100kHz for DS1307)
        I2CBRH = 0x00;                  // BRG High
        I2CBRL = 0x2E;                  // BRG Low

        PAADDR = 0x02;
        PACTL |= 0xC0;                  // alterate function: PA6-SCL, PA7-SDA for I2C port
        I2CCTL = I2C_ENABLE;
}
void I2C_start (void)  {
        I2CCTL |= SEND_START;                   // I2C Start bit
}
void I2C_stop (void)  {
        I2CCTL |= SEND_STOP;                    // I2C Stop bit
        while ((I2CCTL & SEND_STOP) == SEND_STOP)
          ;
}
void I2C_write_byte (unsigned char data)  {
        I2CDATA = data;                         // Write I2C data
}
void I2C_Transmit_Data_Empty (void) {
        while ((I2CSTAT & TRANSMIT_DATA_REG_EMPTY) == 0x00) // Wait for Transmit
                ;                                                       // Buffer Empty
}
unsigned char I2C_read_byte (void)  {
        unsigned char data;
        data = I2CDATA;                         //Read I2C data
        return (data);
}
void I2C_Acknowledge (void)  {
        while ((I2CSTAT & RECEIVED_ACK) == RECEIVED_ACK) ;      // wait for ACK
}
unsigned char I2C_AckNack (void)  {
        unsigned char data;
        while (1) {
        // Wait for the last byte Transmit/Receive ACK/NACK
         if ((I2CSTAT & RECEIVED_ACK) == RECEIVED_ACK)
```

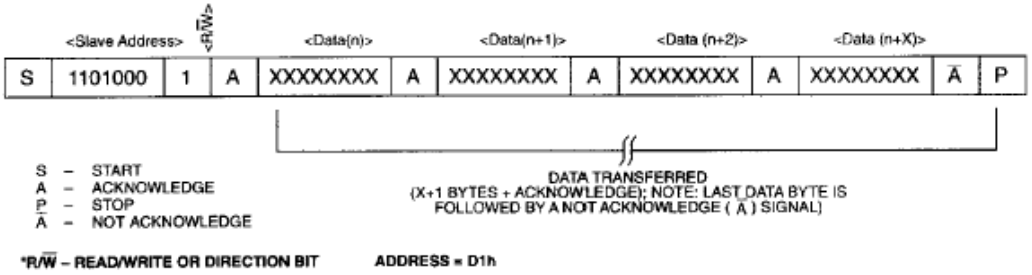
```
            return (ACK);
          else if ((I2CSTAT & RECEIVED_NACK) == RECEIVED_NACK) {
                  I2CCTL |= FLUSH_TXD;
                  I2C_start ();
                  I2C_stop ();
                  data = I2C_read_byte ();                    // Clear I2C Receive buffer
                  printf ("\n\n\rI2C Error - NACK Received");
                   return (NACK);
} } }
void I2C_Send_NACK (void)  {
          I2CCTL |= SEND_NACK;                                // Send NOT Acknowledge
}
void I2C_Receive_Data_Full (void)  {
          while ((I2CSTAT & RECEIVE_DATA_REG_FULL) == 0x00)  //Wait for Receive Buffer Full
            ;
}
```

3.3.2 Data Transfer

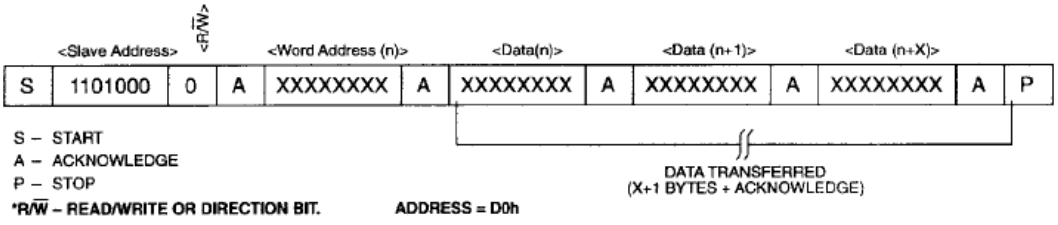
The DS1307 device address is 7-bit, 1101000. The last bit is for direction, 0 for write and 1 for read. The definition of the device identification code in the program is as follows,

```
#define EEPROM_Read_Address  0xD1          //I2C EEPROM Read Address
#define EEPROM_Write_Address 0xD0          //I2C EEPROM Write Address
```

The data transfer between Z8 and RTC is followed by 2-wire serial data bus. That is, the bus is controlled by Z8 microcontroller that generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until STOP condition is executed. Each receiving device, when addressed, is obliged to generate an acknowledge after reception of each byte. The DS1307 data write and read are shown in Figure 17.



(a) Data Read



(b) Data Write

Figure 17: DS1307 Data Transfer

The following code snippets show how the data transfer works (i.e. read/write) based on I2C protocol.

```
unsigned char ReadEEPROM(unsigned char addrL)
{
        unsigned char data;

        I2C_start ();                           // I2C Start
        I2C_write_byte (EEPROM_Write_Address);  // EEPROM Address; Write
        I2C_Transmit_Data_Empty ();             // wait for transmit buffer empty
        I2C_Transmit_Data_Empty ();             // wait for transmit buffer empty
        I2C_write_byte (addrL);                 // LSM EEPROM Address
        if (!I2C_AckNack())                     // wait for ACK/NACK from EEPROM
        return;
        I2C_Transmit_Data_Empty ();             // wait for transmit buffer empty
        I2C_start ();                           // I2C Start
        if (!I2C_AckNack())                     // wait for ACK/NACK from EEPROM
        return;
        I2C_Send_NACK ();                       // send NACK to EEPROM
         I2C_write_byte (EEPROM_Read_Address);  // EEPROM Address; Read
        I2C_Receive_Data_Full ();               // wait for receive data
        data = I2C_read_byte ();                // EEPROM Read data
        I2C_stop ();                            // I2C Stop

        return data;
}

void WriteEEPROM(unsigned char addrL, unsigned char data)
{
        I2C_start ();                           // I2C Start
        I2C_write_byte (EEPROM_Write_Address);  // I2C EEPROM Address; Write

        I2C_Transmit_Data_Empty ();             // wait for transmit buffer empty
        I2C_write_byte (addrL);                 // LSB EEPROM Address
        if (!I2C_AckNack())                     // wait for ACK/NACK from EEPROM
        return;
         I2C_Transmit_Data_Empty ();            // wait for transmit buffer empty
        I2C_write_byte (data);                  // EEPROM Write data
        if (!I2C_AckNack())                     // wait for ACK/NACK from EEPROM
        return;
        I2C_Transmit_Data_Empty ();             // wait for transmit buffer empty
        I2C_stop ();                            // I2C Stop
        if (!I2C_AckNack())                     // wait for ACK/NACK from EEPROM
        return;
}
```

3.3.3 Time/Calendar Reading/Setting

DS1307 Address Map and Timekeeper Registers are shown in Figure 10. The following function shows how to read the time and calendar from RTC.

```
#define SECOND          0
#define MINUTE          1
#define HOUR            2
```

```
#define DAY             3
#define DATE            4
#define MONTH           5
#define YEAR            6

// Read data from DS1307 RTC through I2C
unsigned char Read_RTC(int mode)
{
        unsigned char data, tmp, tms;
        data = ReadEEPROM(mode);
        if(mode == HOUR) {
                if(data&0x20)
                    AM = 1;         // AM
                else AM =0;         // PM
           tmp = data & 0x10;
           tmp >>= 4;
           tms = data & 0x0F;
           data =  tms + tmp*10;
        }
        else if(mode == DAY) {
           data &= 0x07;
        }
        else {
           tmp = data & 0xF0;
           tmp >>= 4;
           tms = data & 0x0F;
           data = tms + tmp*10;
        }
   return data;
}
```

Likewise, using function WriteEEPROM(mode, address) can set the time or calendar.

3.4 Temperature Driver and Reading

MAX6610 analog temperature sensor is used for temperature reading in the final project. I directly use MAX6610 sensor on the Lab3 board for this project. MAX6610 hardware specifications are described in Section 2.1.5. The MAX6610 configuration and schematics for Z8 are shown in Figure 11 and Figure 2, respectively. The driver for communication to Z8 microcontroller contains ADC initialization and calculation for the temperature as shown in the following code functions. ALG0 – PB0 port is used for temperature input to Z8.

```
// Initialize the ADC
void init_ADC(int mode)
{
        PBADDR = 0x01;          // set up port B to use as ADC input
        PBCTL |= 0x01;          // PB0
        PBADDR = 0x02;          // Port B alternate function
        PBCTL |= 0x01;          // enable PB0 = ANA0 ADC INPUT 0
        ADCCTL = 0x90;          // ADC Control Register
                                // CEN =1, 0, VREF=0, CONT=1, ANAIN=0000
```

```
        IRQ0ENH |= 0x01;          // Set Interrupt Highest Priority
        IRQ0ENL |= 0x01;
        SET_VECTOR(ADC, isr_adc);      //set the interupt vector
        EI();
        // CEN will be 0 at first complete conversion result 5129+24 system clock cycles
        while(ADCCTL & 0x80)
                ; // Do nothing     but wait for the end of the first A/D conversion
}

//Interrupt routine
//ADC interupt.
#pragma interrupt
void isr_adc(void)
{
        adc_data = calADC();
}

unsigned int calADC(void)          // get temperature
{
        unsigned char tmp;
        unsigned int data;

        tmp = ADCD_L;              // get low bits
        tmp >>= 6;                 // move to the right side
        tmp &= 0x03;               // clean up
        data = ADCD_H;
        data <<= 2;                // move to the left side
        data &= 0x03FC;            // clean up
        data |= tmp;               // ADC value

        return data;
}
```

3.5 Phrase Allophone Editing

To produce speech, a list of selected allophones is sent to the SpeakJet. As the SpeakJet is vocalizing this list of allophones, MSA actively and continuously calculates all the sound components of the allophones including the transitional sounds made between the allophones, producing the same sounds that the human mouth does as it moves one position to another position.

Selecting the appropriate combination of allophones and pauses can thusly create any English word or phrase. Further tuning with the Rate, Pitch, Bend and Volume parameters adds to the delivery of the phrase and can change the emotion in which the phrase is perceived.

Stressing the Rate, Pitch, Bend and Volume parameters to levels outside the human range can result in some interesting sounds that go way beyond what a normal human mouth can produce. In addition, several other sounds effects, which are included in the MSA Sound Component Database, of which, some use vocalization and some do not, can be integrated into the phrases.

The result is a system that gives the user the ability to not only produce an unlimited vocabulary, but also to produce slang, gibberish, moans, groans, yodels and other weird vocalized sounds not normally included in a canned TTS system.

All allophones in this final project are edited directly by hands and PhaseALator software. These allophones mainly contain temperature, time, calendar, songs, etc. Although allophones are mainly used for English language, they can be used for simulation of some foreign language pronunciation.

3.5.1 Temperature Allophone Editing

The temperature phrase contains two major parts. One is the numerical phrase. The other is just "degree" word. The allophone code of the "degree" can be directly translated from PhaseALator. However, numerical phrase should be composed with 10 basic digit numbers (i.e, 0, 1,…, 9) and 9 basic high digit numbers like 10, 20,…, 90, 100, 1000,... In this project, all the allophones for these basic phrase elements are first stored in the ROM. When used, they will be copied into the RAM. The following function shows how to compose the speech of the temperature degree.

```
void SpeakTemperature(float temperature)
{
        int T, T1, T0, lc;
        unsigned char tmp[20];
        T = (int)(temperature + 0.5);
        if(T > 0) {
                if(T <= 20){
                        lc = copy_data_digitCode(speech, CODE_1, T, 0);
                        datacat_rom(speech, s_degree, lc, LEN_DEGREE);
                }
                else {
                        T1 = (int)(T/10);
                        T0 = (int)(T%10);
                        lc = copy_data_digitCode(speech, CODE_10, T1-2, 0);
                        copy_data_digitCode(tmp, CODE_1, T0, 0);
                        datacat(speech, tmp, lc, lens[T0]);
                        datacat_rom(speech, s_degree, lc+lens[T0], LEN_DEGREE);
                }
        }
        puts(speech);   // Speak the temperature
}
```

3.5.2 Time/Calendar Allophone Editing

The time phrase contains two major parts. One is the numerical phrase. The other is just "AM" or "PM" word. Likewise, the allophone code of the "AM" or "PM" can be directly translated from PhaseALator. Numerical phrases are composed with 10 basic digit numbers and 9 basic high digit numbers from 10 to 60. all the allophones for these basic phrase elements are first stored in the ROM. When used, they will be copied into the RAM. The following function shows how to compose the speech of the time.

```
void SpeakTime(unsigned char hour, unsigned min, int am)
{
```

```
        unsigned char tmp[MAX_CODE_LEN], lc, T1, T0;
        lc = copy_data_digitCode(speech, CODE_1, hour, 0);
        if(min <= 20) {
                lc = copy_data_digitCode(speech, CODE_1, min, lc);
        }
        else {
                T1 = (int)(min/10);
                T0 = (int)(min%10);
                lc = copy_data_digitCode(speech, CODE_10, T1-2, lc);
                copy_data_digitCode(tmp, CODE_1, T0, 0);
                datacat(speech, tmp, lc, lens[T0]);
                lc += lens[T0];
        }
        if(am)
                datacat_rom(speech, s_AM, lc, LEN_AM);
        else
                datacat_rom(speech, s_PM, lc, LEN_PM);
        puts(speech);       // Speak the time
}
```

The calendar phrases contains more elements, including day, date, month, and year. Likewise, the calendar speech code strings can be composed by these elements using similar methods.

3.6 System Integration and Main Application Flow Chart

The system integration contains two aspects, hardware integration and software integration. Hardware components are integrated as shown in Figure 2 and Figure 12. Software modules are integrated into one main application based on the software design block diagram in Figure 3. This application includes 12 sub-applications corresponding to 15 voice commands.

In this VoiceDirect board for the final project, since OUT5 PIN16 does not work, the Command 5 and Command 13 cannot be executed. In my program, Command 4 is not used either. Thus, the total 12 voice commands or questions are listed as follows,

1) Command 1: welcome	-- "Hello!"
2) Command 2: temperature	-- "Can you tell me the temperature?"
3) Command 3: time	-- "What time is it now?"
4) Command 6: date	-- "Can you tell me the date?"
5) Command 7: name	-- "What's your name?"
6) Command 8: day	-- "What day is today?"
7) Command 9: math	-- "7 plus 8"
8) Command 10: sing	-- "Can you sing?"
9) Command 11: music	-- "pa, pa" clap the hands
10) Command 12: goodbye	-- "Thank you! Bye-bye!"
11) Command 14: set the time	-- "Reset the time"
12) Command 15: Ni Hao	-- "Ni Hao" Hello in Chinese

The flow chart of the main application for Z8 Speaks final project is shown in Figure 18 as follows.

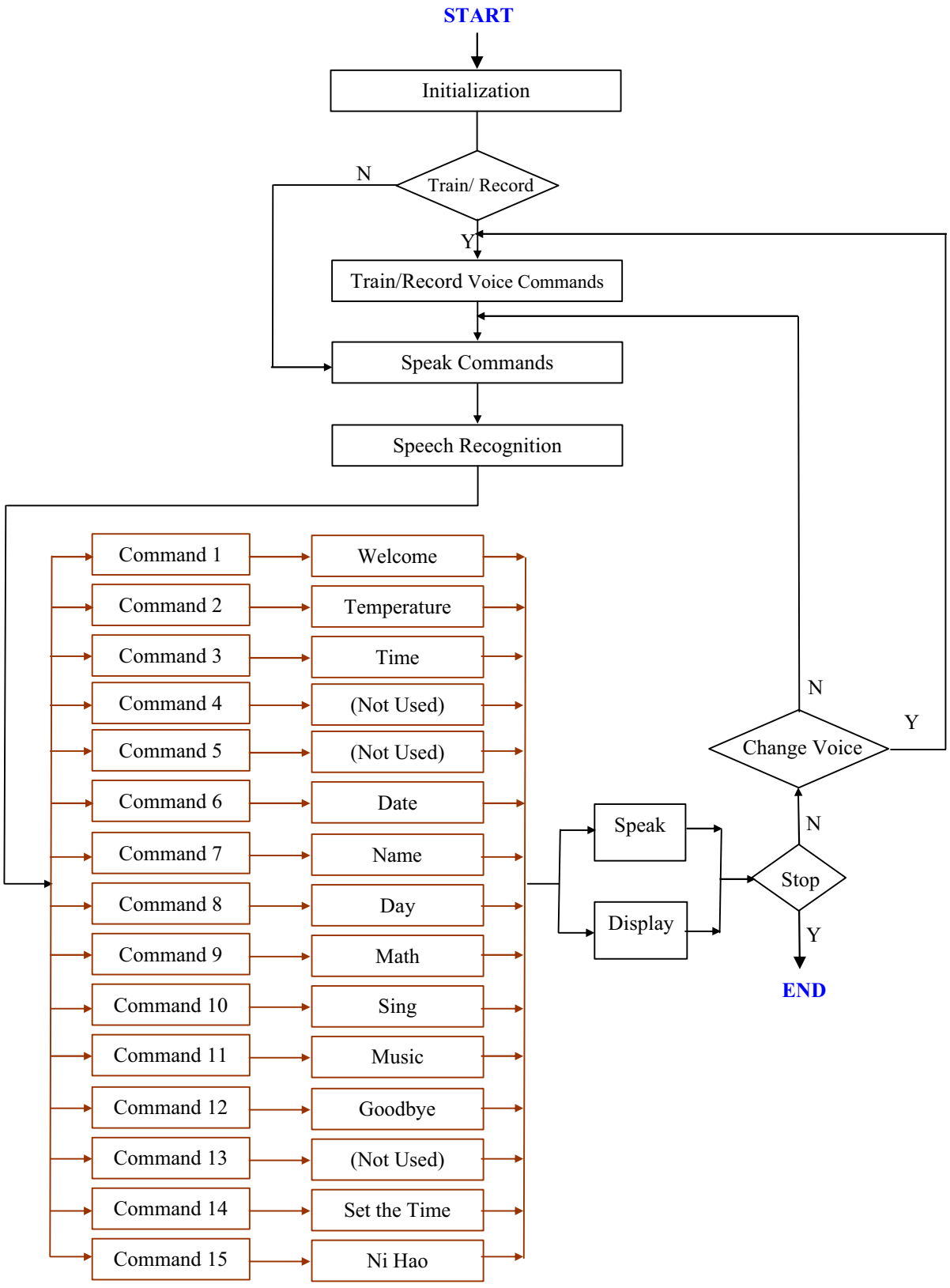


Figure 18: The Flow Chart of Z8 Speak Main Application

4 Conclusions

The Z8 Speaks System is a hardware-software mix project. It integrates Z8 microcontroller, speech recognition, speech synthesis, real-time clock, temperature sensing, LED display, timer control, and other hardware and software module technologies.

The overall work for the final project works very well as planned in the project proposal. This final work makes the user be able to interact with Z8 Speaks (i.e., a talking robot) over the speech communication. The final project meets very well all the proposed requirements, such as read the temperature from ADC temperature sensor, read the Real-Time Clock (RTC) using I2C bus, control the speech recognition board using GPIO and hardware switches, and control speech synthesis through serial data line. In addition, more voice operations and speech controls are added into the final project beyond the proposal.

Just as described in Section 3, Implementation and Construction, each step is an important design decision I made for this project. After each step passes test and verification, the system can be integrated correctly. In summary, these decisions are as follows,

- Test and verify speech recognition
  - → VOICEDIRECT board stand-alone test
  - → OICEDIRECT driver development for Z8
- Test and verify speech synthesis
  - → SpeakJet chip stand-alone test
  - → SpeakJet driver development for Z8
- Test and verify SparkFun real-time clock module
  - → Driver development through I2C
  - → Time/Calendar management
- MAX6610 Temperature Driver development through ADC
- Phrase allophone editing
- System integration and main application development

During my implementation, strictly following each step not only made all of them work very well but also reduce mistake or errors further accelerated the whole project development. This further proves that decision making is the key for the development.

This is also one key point I learnt from this project. In addition, what else I have learnt from this project is as follow,

- Hardware design and Z8 programming
- Speech recognition board use and development
- Speech synthesis chip use and development
- I2C control of Real-time Clock
- ADC control of temperature sensor

5 Attachments List

The list of the attachments for this final project is as follows,

Data Sheets
1) Sensory VoiceDirect Speech Recognition Kit (hardcopy)
2) SpeakJet User's Manual (PDF)
3) DS1307 64x8 Serial Real-Time Clock (PDF)
4) SparkFun RTC Module Schematic (PDF)
5) MAX6610/6611 Temperature Sensors and Voltage References (PDF)

Source code
1) Speaks ZDSPROJ file
2) main.c, main.h
3) gpio.c, gpio.h
4) adc.c, adc.h
5) timer.c, timer.h
6) eeprom.c, eeprom.h
7) I2C.c, I2C.h
8) rtc.c, rtc.h
9) speakjet.c, speakjet.h
10) voicedirect.c, voicedirect.h
11) zsldevinit.asm

Demonstration Video (13MB)
Liao_z8Speaks_320x240.mpg(13MB)

Reports
Project proposal (PDF)
Project Status Report (PDF)
Final Project Report – Z8 Speaks (PDF)
Project Summary brochure (PDF)