ESE 566

Hardware/Software Co-Design of Embedded Systems, Fall 2022

Project Title: Talking Tiny Cognitive Architecture

Project Report 1

George Madathil Saviour
114834447

Table of Contents

1. Introduction

Talking Tiny Cognitive Architecture is an embedded system consisting of a PSoC (Programmable System-on-chip) that is designed to sense data based on a parameter and perform 1) Sensing 2) Learning and 3) Decision Making. This system consists of a PSoC Microcontroller, which is interfaced with sensors i.e., sub-systems such as Temperature Sensor, Light Sensor, and Microphone.

This report of Project 1 consists of the details on the interface of Sensing sub-system.

2. Interfacing of Signals to PSoC1

2.1 Temperature Sensor MCP9808

Microchip Technology Inc's MCP9808 digital temperature sensor converts temperatures between -20 degree Celsius and +100 degree Celsius to a digital word with +-0.25/+-0.5-degree Celsius accuracy. The sensor has VDD, GND, SCL, SDA, A2,A1,A0 pins. VDD pin is connected 5V of PSOC controller. GND pin is connected to ground. SDA is connected to P1[5] pin and SCL is connected to P1[7]. A2, A1, A0 is connected to GND pin of PSOC so that a I2C slave address of 0x18 value is attained.

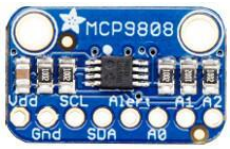


Figure 1: Temperature Sensor MCP9808

2.2 Light Sensor TEMT6000

Light sensor SIG, GND and VCC pins. GND pin is connected to ground pin. VCC pin is connected to 5V. SIG pin is connected to the P0[7]. P0[7] is internally configured as a analog input port which will will an input to the a Programmable gain amplifier. The output of PGA is connected to one of the channels of dual channel ADC configured.

Figure 2: Light Sensor TEMT6000

2.3 Microphone

The microphone amplifier MAX4466 is ideal for reading from the ADC of a microcontroller. Because the output is track-to-track, it can reach 5Vpp if the sound becomes louder. The electret microphone amplifier sensor incorporates a welded 20-20KHz electret microphone with a small trimmer potentiometer on the back for gain adjustment. We can adjust the gain from 25 to 125 times. The adjustable amplifier CMA-4544PF-W has excellent power supply noise suppression.

Microphone has SIG, VDD and GND pins. SIG pin is connected to the P0[7]. P0[7] is internally configured as a analog input port which will be an input to the Programmable gain amplifier. The output of PGA is connected to one of the channels of dual channel ADC configured.

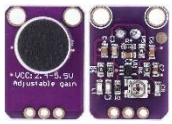
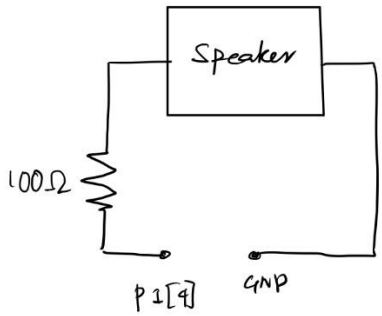


Figure 3: Microphone

2.4 Speaker

Speaker is connected to PWM output pin P1[4] as shown below.



2.5 LCD Display

LCD is interphased on the port 2 pins 0,1,2,3,4,5,6.

3. Design

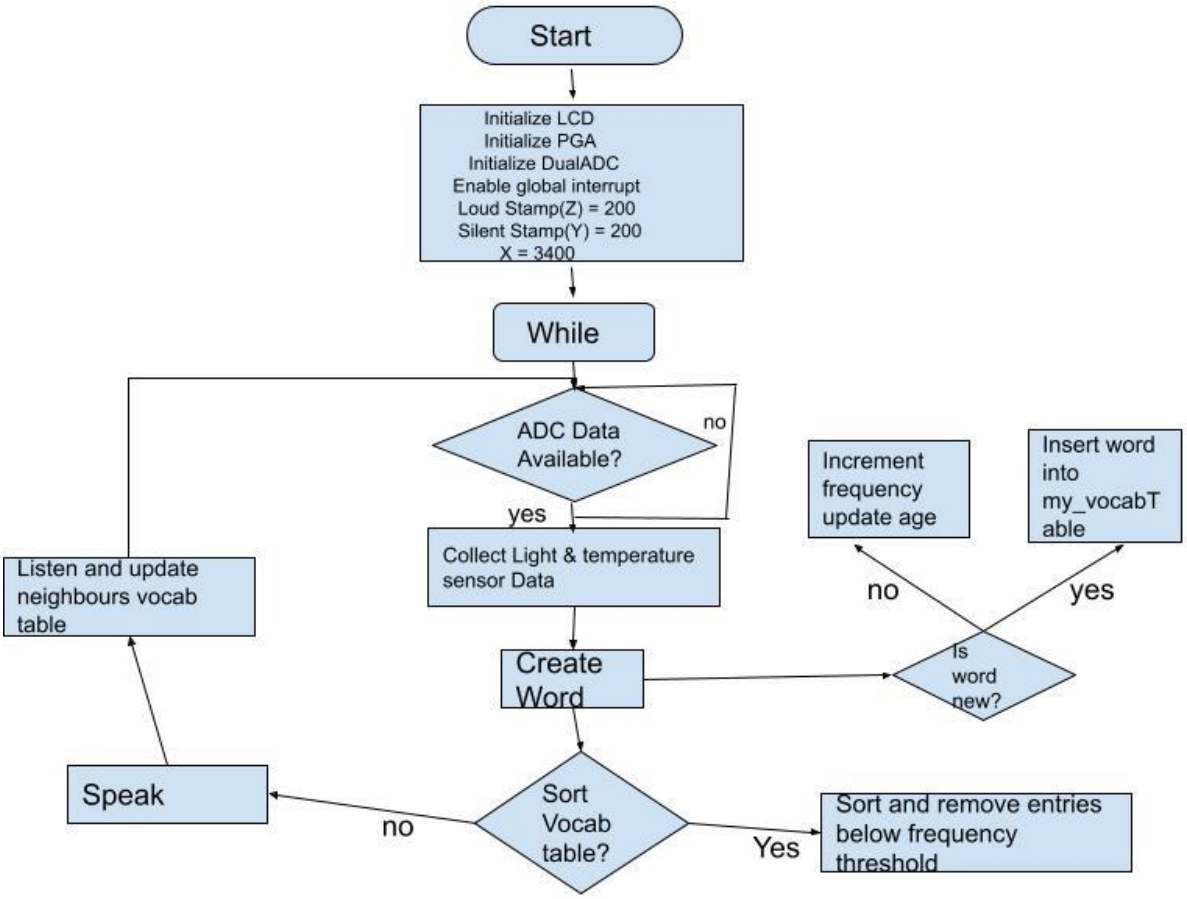
3.1 Algorithm

1. **Sensing:** The system will sense the temperature and light through MCP9808 temperature sensor and TEMT6000 light sensor
2. **Labeling:** Based on the range of the intensity of light and temperature, the system will assign a label to it.
3. **Word formation:** Based on the labels generated for temperature and light values, a single word which describes the light- temperature condition is assigned.
4. **Update Vocabulary table:** The generated word is inserted to the system's own vocabulary table if it is a new word. If it is old, just the frequency and updated for that word in the table. Every time a word is generated, the age of other words is also updated.
5. **Re-order Vocabulary table:** The system will reorder it own vocabulary table and the vocabulary table generated after communication with neighbor. This will happen periodically every 60s. The timer ISR is automatically triggered every 60s and this re-ordering is triggering when possible.
6. **Speaking:** the system will transmit this newly generated word irrespective of whether it is new or not. Sound will be generated using PWM operation.
7. **Listing:** Count the number of peaks detected in a hearing window duration, and add this info to the peer vocabulary table.

Word Encoding and speaking:

Word is a combination of 1's (loud bit) and 0's (silent bit). Every word has a fixed length of x (3.4s). Here the duration of silent bit z is 200ms and duration of y bit is 200ms. For example, if the word is "DARK COLD" the word encoding value will be 1. Which means that there will be one loud bit in the word and remaining 16 bits as silent bits. (10000000000000000) So, at the start one loud stamp lasting for 200ms is outputted. And remaining 16* 200ms = 3.2s no sound is generated. The word encoding means the number peaks/loud stamps that needs to be outputted.

4.2 Flowchart



3.3 Routines

Main Routines:

1. `getLux()` : This API reads the ADC values from the 2nd channel of the dual ADC which is connected to the light sensor. The ADC value read is converted to lux value which will range from 0 to 1000.
2. `read_temperature()` : This API reads temperature from the temp sensor which via I2C communication. Here PSOC 12C module is configured as master and talks to temperature sensor which is a i2c slave. The value read from the sensor is converted to degree Celsius.
3. `create_word()` : This API will generate labels for each light and temp value read and based on the generated labels a word encoding is generated. The generated word encoding describes a light – temp condition.
4. `update_table()` : If the word encoding generated is new, then this API will insert the new word into the its own vocab table. If it is an existing word, then only the frequency of that word is updated. This API also

increments the age value associated with every word in the vocabulary table each time when a word is generated by the system.

5. `quicksort()`: A quick sort for the vocabulary tables is performed based on the frequency values of each word entry in the table
6. `remove()`: This API is used to remove word entries below a threshold frequency from the reorder vocabulary tables.
7. `speak()`: This API is used to output the words generated using speakers connected. The PWM generates the signals required to drive the speaker.

Misc. routines:

8. `blocking_delay()`: This API will perform a blocking delay operation
9. `Swap()`: Used for the reordering algorithm
10. `Part()`: Partitioning for the quick sort algorithm

ISR:

11. `TimerISR()`: This ISR is triggered by the timer module every 60 seconds. This API triggers the re-ordering operation .
12.

3.4 Data Structure

An array is used to store the two vocabulary tables. Each element of the array is a strut consisting of three elements.

```
typedef struct vocabTable
{
    BYTE word;
    BYTE frequency;
    BYTE age;
}vocabTable_type;

vocabTable_type my_table[SIZE]= {0};        // To do: finalize on the size
vocabTable_type neighours_table[SIZE] = {0};// To do: finalize on the size
```

The size of the table is chosen 20, but this value needs to reduce and made optimum later.

4. Hardware Resources

4.1 Pulse Width Modulator (PWM):

In hardware, the PWM component provides compare outputs to generate single or continuous timing and control signals. This PWM drives the speaker. Below is the configurations of the PWM. PWM is placed the DBC11 digital block of PSoC. The output of the PWM is routed to P1[4] using the buses. PWM is configured output a 50% duty cycle signal and VC3 = 8Khz. Therefore the audio signal generated by speaker will be 8Khz.

Name	PWM8_1
User Module	PWM8
Version	2.60
Clock	VC3
Enable	High
CompareOut	Row_1_Output_0
TerminalCountOut	None
Period	255
PulseWidth	50
CompareType	Less Than
InterruptType	Terminal Count
ClockSync	Sync to SysClk
InvertEnable	Normal

4.2 Programmable Gain Amplifiers (PGA):

PGA used is mainly for making multiple GPIO ports accessible for ADC. Here the gain used is 1. Below is the configurations. In this project two PGAs are used, PGA1, is for interphasing mic and PGA 2 is for interphasing light sensor. PGA1 is takes input from P0[1], PGA2 takes input from PGA P0[4]. The output of PGA is connected to the dual channel adc. PGA1 is placed in the ACC00 and PGA2 is placed in ACC01 digital block.

Name	PGA_1
User Module	PGA
Version	3.2
Gain	1.000
Input	AnalogColumnMUXBusSwitch_0
Reference	AGND
AnalogBus	AnalogOutBus_0

Parameters - PGA_2

Name	PGA_2
User Module	PGA
Version	3.2
Gain	1.000
Input	AnalogColumnMUXBusSwitch_1
Reference	AGND
AnalogBus	AnalogOutBus_1

4.3 Analog to Digital Converters (ADC):

We are using a dual channel ADC which connects to both Microphone and Light to convert the continuous voltage signals to digital signals. Below are the configurations used for the module. Dual ADC is placed in the ASC10 and ASD11 analog blocks in the PSoC. ADC is clock sourced by VC1. Here VC1 = 12MHz and it is a 10-bit ADC.

Parameters - DUALADC_1

Name	DUALADC_1
User Module	DUALADC
Version	2.30
ADC Input1	ACC00
ADC Input2	ACC01
ClockPhase1	Norm
ClockPhase2	Norm
Clock	VC1
ADCResolution	10 Bit
CalcTime	60
DataFormat	Unsigned

4.4 I2C Module:

Here 12C is configured as a master. Below is the configuration of the block. P1[5] is configured as SDA and P1[7] is configured as SCL.

Name	I2CHW_Temp
User Module	I2CHW
Version	2.00
Read_Buffer_Types	RAM ONLY
I2C Clock	50K Standard
I2C Pin	P[1]5-P[1]7

4.5 Timer module and LCD:

In addition, we are using LCD and Timer user modules. This is a 16-bit timer that generates an interrupt signal every 60 (T)seconds. It is placed in DCC12 and DCC13 digital blocks.

Memory Usage:

ROM 40% full. 6428 out of 16384 bytes used (does not include absolute areas).
RAM 20% full. 148 bytes used (does not include stack usage).

5. Timing and Complexity

Time:

getLux – 49 clock cycles, read_temperature – 30, create_word -74, update_table – 13, quick_sort – 13, remove - 38

Routine	Complexity
`update_table`	O(n), n= number of words in vocabulary
`Quick sort`	O(nlogn)
`Remove`	O(n)
`Reorder (Sorting + Remove)`	O(nlogn) + O(n)

Tradeoff:

1) Size of the vocabulary vs Precision: If we want increased semantics that means that the vocabulary size needs to be increased. Based on the computational complexity of the operation, one cycle/iteration which consists of sensing, word generation, table updation, table reorder, speaking, and listening has a computation complexity of O(nlogn). As the size of vocabulary (n), increases, the computation complexity also increases. Since the code is executing sequentially there will be a big delay in each operation take.

5. Testing and Debugging:

1) Initially we interfaced modules one at a time and test separately. We used LCD to print debug messages to root cause issues.
2) We initially used a pulse-generated signal to initially power the speaker for outputting signals at a frequency of 8Khz. Then we used this signal generated to test the mic
3) We used an Oscilloscope to check whether the Mic continuous signal is reaching the PGA and therefore making sure that the mic is working.
4) Temperature I2C communication was not initially working, we added debug message at each check to see whether the code flow was getting stuck.

6 Improvement:

1) Can maybe modify the existing priority linked list data structure instead of the array data structure to improve the computational cost.
2) We can maybe use lesser user modules and keep some of the HW resources free for other use.
3) Instead of LCD, we can maybe make debug messages to print in the PC putty terminal using RS232 to BBB communication.
4) Rest pin is already configured, but code changes to trigger a reset is not implemented yet. This improvement will be made.

7. Appendix

```
//--------------------------------------------------------------------------------
// C main line
//--------------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include <stdio.h>
#define TEMP_SENSOR_SLAVE_ADDRESS 0x18
#define TEMP_REG                  0x05
#define MIC_ADC_CHANNEL           0x01
#define LIGHT_SENSOR_ADC_CHANNEL  0x02
#define SIZE                      20
#define X                         3400 // 3.4s
#define Z                         200  // 200m
#define Z_ns                      (Z*1000000) //ns
#define Z_div                     (Z_ns/50)
#define Y                         200  // 200ms
#define Y_ns                      (Z*1000000) //ns
#define Y_div                     (Z_ns/50)
#define Q                         200  // Have find an optimum value

enum light  {DARK, NORMAL , BRIGHT};
enum temp   {COLD, MEDIUM, HOT};

typedef struct vocabTable
{
    BYTE word;
    BYTE frequency;
    BYTE age;
}vocabTable_type;

vocabTable_type my_table[SIZE]= {0};         // To do: finalize on the size
vocabTable_type neighours_table[SIZE] = {0};// To do: finalize on the size

BOOL sortVocabTables = FALSE;
void print_LCD_debug_msg(char* msg,INT row,INT col);
INT readDualAdc(INT channel);
INT read_temperature(void);
float getLux(void) ;
BYTE create_word(float lux, INT temp);
void blocking_delay(BYTE bTimes);
void update_table(BYTE word_encoding, vocabTable_type* table );
void quickSort(vocabTable_type A[], int l, int h);
int part(vocabTable_type arr[], int l, int h);
void swap(vocabTable_type* a, vocabTable_type* b);

#pragma interrupt_handler TimerISR
#pragma interrupt_handler ResetSwitchISR
// This ISR is triffered every 60 seconds
void TimerISR(void )
{
    sortVocabTables = TRUE;
}

void ResetSwitchISR(void )
{
    INT_MSK3 = INT_MSK3 | 0x80;
    // Trigger the hardware reset ISR from the software
}

void main(void)
{
    float luxValue; // Ambient Light illumiance value
    INT   tempValue;      // Ambient temperature
    BYTE  word;
    BOOL neword;
    // PGA1 init
    PGA_1_Start(PGA_1_HIGHPOWER);
    // PGA2 init
    PGA_2_Start(PGA_1_HIGHPOWER);

    // Initial the Dual ADC
    DUALADC_1_Start(DUALADC_1_HIGHPOWER);        // Turn on Analog section
    DUALADC_1_SetResolution(10);                 // Set resolution to 10 Bits
    DUALADC_1_GetSamples(0);

    // Initializes LCD to use the multi-line 4-bit interface
    LCD_2_Start();

    // Enables the I2C HW block as a Master
    I2CHW_Temp_EnableMstr();
```

```
    // Initial the Dual ADC
    DUALADC_1_Start(DUALADC_1_HIGHPOWER);       // Turn on Analog section
    DUALADC_1_SetResolution(10);                // Set resolution to 10 Bits
    DUALADC_1_GetSamples(0);

    // Initializes LCD to use the multi-line 4-bit interface
    LCD_2_Start();

    // Enables the I2C HW block as a Master
    I2CHW_Temp_EnableMstr();

    //Enable reset button interrupt
    //To do How to trigger a software reset via button?

    // Initialize time
    Timer16_1_WritePeriod(0xffff);// Do this in the config
    Timer16_1_WriteCompareValue(0x0001);
    Timer16_1_EnableInt();//Enable Timer interrupt


    // Enable global interrupts
    M8C_EnableGInt;

    // Start the timer
    Timer16_1_Start();
    while(1)
    {

        // Sense ////////////////////////////

        //Read light sensor value
        luxValue    = getLux(); // luxValue ranges from 0 to 1000
        //Read temperature value
        tempValue   = read_temperature();\

        // Word formation ///////////////////
        word = create_word(luxValue, tempValue);


        // Update my vocab
        update_table(word,my_table);


        // Sort vocab table

        if (sortVocabTables == TRUE)
        {
            quickSort(my_table, 0 , SIZE-1);
            // Remove the table entries which does not meet the required frequency threshold
            remove(my_table,Q);
        }
        // Speak ////////////////////////////
        if(neword)
        {
            speak();
        }
        // Listen? ////////////////////////// To Do
    }
}
```

```
BYTE create_word(float lux, INT temp)
{
    enum light light_label;
    enum temp temp_label;
    BYTE word;
    if (lux >= 0 && lux < 333)
        light_label = DARK;
    else if(lux >= 333 && lux < 666)
        light_label = NORMAL;
    else if(lux >=666 && lux <=1000)
        light_label = BRIGHT;

    // Check this
    if (temp < 10)
        temp_label = COLD;
    else if(temp >=10 && temp < 23)
        temp_label = MEDIUM;
    else if(temp >= 23)
        temp_label = HOT;

    if (light_label == DARK && temp_label == COLD)
        word = 1; //Dark light and Cold condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: DARK COLD");
    else if (light_label == DARK && temp_label == MEDIUM)
        word = 2; //Dark light and Medium temp condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: DARK MEDIUM");
    else if (light_label == DARK && temp_label == HOT)
        word = 3; //Dark light and Hot temp condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: DARK HOT");
    else if (light_label == NORMAL && temp_label == COLD)
        word = 4; //Normal light and cold temp condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: NORMAL COLD");
    else if (light_label == NORMAL && temp_label == MEDIUM)
        word = 5; //Normal light and medium temp condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: NORMAL MEDIUM");
    else if (light_label == NORMAL && temp_label == HOT)
        word = 6; //Normal light and hot temp condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: NORMAL HOT");
    else if (light_label == BRIGHT && temp_label == COLD)
        word = 7; //Bright light and cold temp condition
    else if (light_label == BRIGHT && temp_label == MEDIUM)
        word = 8; //Bright light and medium temp condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: BRIGHT MEDIUM");
    else if (light_label == BRIGHT && temp_label == HOT)
        word = 9; //Bright light and hot temp condition
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: BRIGHT HOT");

    return word;// return word encoding
}


BOOL update_table(BYTE word_encoding, vocabTable_type* table)
{
    int i;
    BOOL found = FALSE;
    for (i= 0; i< SIZE; i++)
    {
        table[i].age++;
        if (table[i].word == word_encoding)
```

```
        if (table[i].word == word_encoding)
        {
            found =TRUE;
            table[i].frequency++;
        }
    }
    if (found == FALSE)
    {
        table[i].word = word_encoding;
        table[i].frequency = 1;
        table[i].age = 1;
    }
return found;
}

// Sort Algorithm for tables based on quick sort

void swap(vocabTable_type* a, vocabTable_type* b)
{
    vocabTable_type temp = *a;
    *a = *b;
    *b = temp;
}

int part(vocabTable_type arr[], int l, int h)
{
    int x = arr[h].frequency;
    int i = (l - 1);
    int j;
    for ( j = l; j <= h - 1; j++)
    {
        if (arr[j].frequency >= x)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[h]);
    return (i + 1);
}

void quickSort(vocabTable_type A[], int l, int h)
{
    if (l < h) {
        int p = part(A, l, h);
        quickSort(A, l, p - 1);
        quickSort(A, p + 1, h);
    }
}

void remove(BYTE freq_threshold,vocabTable_type table )
{
    int i;
    for(i= 0; i< SIZE; i++)
    {
        if(table[i].frequency < freq_threshold)
        {
            table[i]= {0};// Remove the old word and details and initial with zero
        }
    }

}

INT read_temperature(void)
{
    BYTE UpperByte = 0;
    BYTE LowerByte = 0;
    INT Temperature =0;
```

```
INT read_temperature(void)
{
    BYTE UpperByte = 0;
    BYTE LowerByte = 0;
    INT Temperature =0;

    I2CHW_Temp_fSendStart(TEMP_SENSOR_SLAVE_ADDRESS, I2CHW_Temp_WRITE);
    I2CHW_Temp_fWrite(TEMP_REG);
    I2CHW_Temp_SendStop();
    I2CHW_Temp_fSendStart(TEMP_SENSOR_SLAVE_ADDRESS, I2CHW_Temp_READ);
    UpperByte = I2CHW_Temp_bRead(I2CHW_Temp_ACKslave);
    LowerByte = I2CHW_Temp_bRead(I2CHW_Temp_NAKslave);
    I2CHW_Temp_SendStop();

    UpperByte = UpperByte & 0x1F; //Clear flag bits

    if ((UpperByte & 0x10) == 0x10){ //TA < 0°C
    UpperByte = UpperByte & 0x0F;//Clear SIGN
    Temperature = 256 - (UpperByte * 16 + LowerByte / 16);
    }else //TA ≥ 0°C*/
    Temperature = (UpperByte * 16 + LowerByte / 16);//Temperature = Ambient Temperature (°C)

    return Temperature;
}
float getLux(void )
{
    float volts = readDualAdc(LIGHT_SENSOR_ADC_CHANNEL)*5.0 /1024.0;
    float amps  = volts/10000.0; // Across 10,000 Ohms
    float microamps = amps * 1000000;
    float lux = microamps * 2.0;

    return lux;
}

INT readDualAdc(INT channel)
{
    int iResult1, iResult2, iResult;
    while(DUALADC_1_fIsDataAvailable() == 0);  // Wait for data to be ready
    iResult1 = DUALADC_1_iGetData1();          // Get Data from ADC Input1
    iResult2 = DUALADC_1_iGetData2ClearFlag(); // Get Data from ADC Input2
                                               // and clear data ready flag
    if (channel == 1)
    {
        iResult = iResult1;                    // Return Mic ADC readings result
    }
    else if ( channel == 2)
    {
        iResult = iResult2;                    // Return light sensor ADC readings result
    }
    return iResult;
}


void blocking_delay(BYTE bTimes)
{
    // Can be used only if the delaly required is a mutiple of 50us.
    // This sufficient for this project.
    LCD_2_Delay50uTimes(bTimes);// Reuse the delay function provided by LCD user module
}

void speak(BYTE word_encoding)
{
    BYTE y;
    switch (word_encoding)
    {
        //        10101010101010101 One word consists of 17 stamps
        case 1: //10000000000000000 1 loud stamp, 16 silent stamp (
```

```
        //        10101010101010101 One word consists of 17 stamps
        case 1: //10000000000000000 1 loud stamp, 16 silent stamp (
            PWM8_1_Start();
            blocking_delay(Z_div);
            PWM8_1_Stop();
            y = Y_div* 16;
            blocking_delay(y);
            break;;
        case 2:// 10100000000000000 2 loud stamp,rest silent stamps
            PWM8_1_Start();
            blocking_delay(Z_div);
            PWM8_1_Stop();
            blocking_delay(Y_div);
            PWM8_1_Start();
            blocking_delay(Z_div);
            PWM8_1_Stop();
            y = Y_div *14;
            blocking_delay(y);
            break;
        case 3:// 10101000000000000 3 loud stamp,rest silent stamps
            //To do
            break;
        case 4:// 10101010000000000 4 loud stamp,rest silent stamps
            //To do
            break;
        case 5:// 10101010100000000 5 loud stamp,rest silent stamps
            //To do
            break;
        case 6:// 10101010101000000 6 loud stamp,rest silent stamps
            //To do
            break;
        case 7:// 10101010101010000 7 loud stamp,rest silent stamps
            //To do
            break;
        case 8:// 10101010101010100 8 loud stamp,rest silent stamps
            //To do
            break;
        case 9:// 10101010101010101 9 loud stamp,rest silent stamps
            //To do
            break;
    }
}
```