ESE 566

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

Project Title: Talking Tiny Cognitive Architecture

Project Report 2

Team: George Madathil Saviour (114834447)

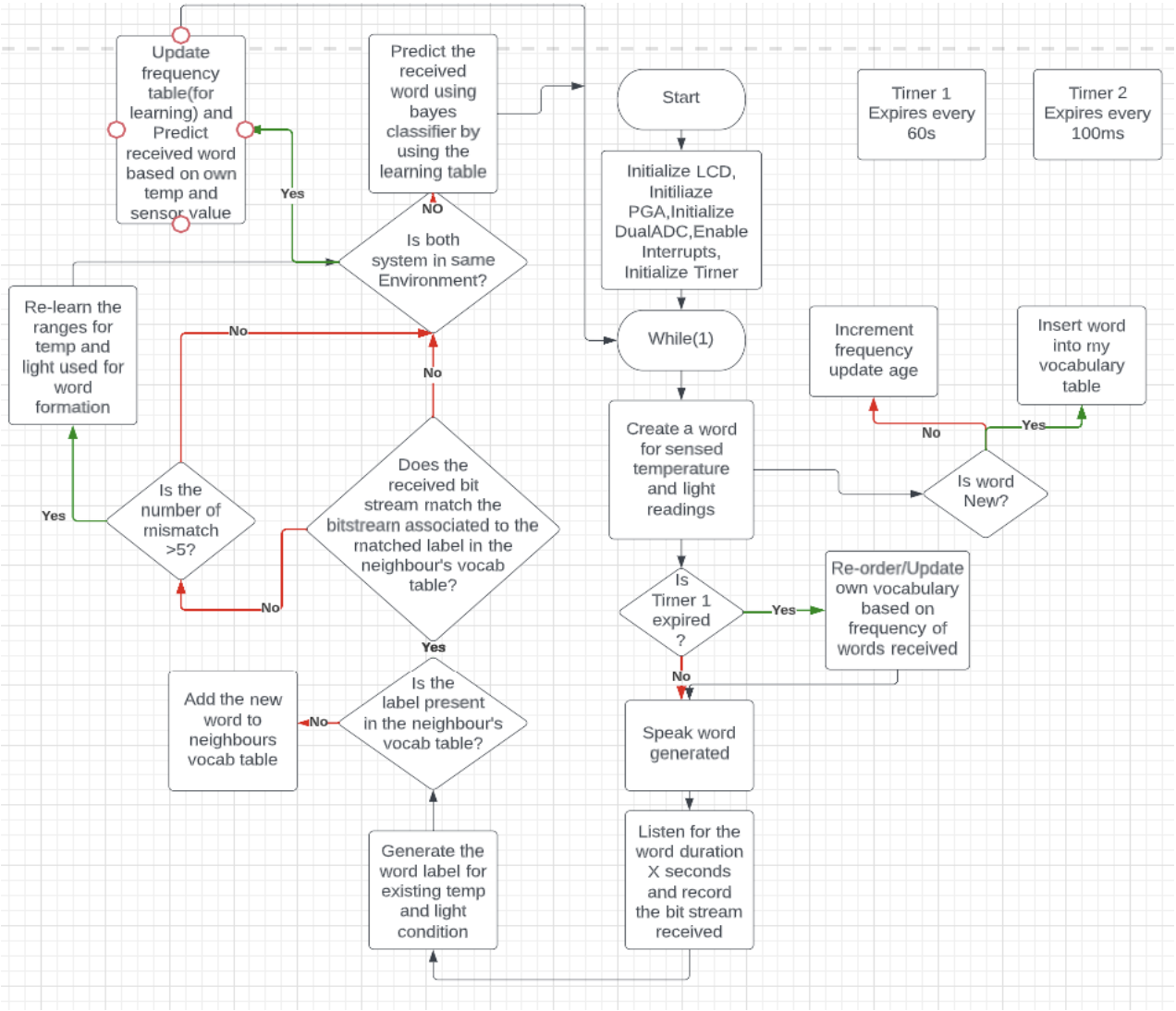
1. Algorithm

The Talking Tiny Cognitive Architecture is a system that senses a data from sensors such as temperature, light and differentiates the data based on range and creates the self-vocabulary table. Also, it communicates with other system with same environmental conditions and creates listening vocabulary table while communicating with other system.

Initially both systems are stationed in the same room in the same environment. Each system learns the other systems behavior of generating the words that denotes a temperature or light or temperature and light condition. Based on the learnings attained, both the systems should be now able to predict the words transmitted between each other when placed in different environments using Naïve Bayes Classifier.

The system performs the following operations: 1) Sensing 2) Learning and 3) Decision making.

Below is a flow chart of our algorithm which gives a high-level idea of how our design performs the above three operations.



Sensing part of the system is almost same as project 1, therefore we'll be stressing more on the learning and decision-making algorithm below.

Learning:
Learning can be briefly explained in two parts:

Part 1: On top of the own and neighbors' vocabulary table maintain a separate learning table containing label, word encoding, and frequency of each word received by the agent. This will be used to calculate the conditional probability for Bayes classification.

Part 2: Improving the data set or frequency table by re-learning the ranges of temp and light used for word generation. After listening, check if the listened word needs to be entered in the neighbor's vocab table. If the same word label is already present in the neighbor's table but with a different bit stream it means the sender agent have extended the word meaning to include a new value for temperature or light conditions. In this case, re-adjust or re learn the ranges for temperature and light. This way the agent correctly labels the received words, and thereby the learning table will have a smaller number of false positive data. The correctly predicted words will highly have likelihood when compared to the false predicted words. Thereby improving the accuracy of the system when in different environment.

Below is how a learning table will look like. This table will be populated if the system is in the same environment. The system follows a continuous learning process if it is the same environment. As soon as both agents are no more in the same environment, then the system depends on this learning table for correct prediction.

Table 1 Learning table

Word Encoding	Label	Frequency
1010 1001	DARK_COLD	10
1010 0000	DARK	8
1010 1001	DARK_MEDIUM	2
1010 1001	DARK_COLD	3
0100 1001	BRIGHT_MEDIUM	2
0100 1010	BRIGHT_COLD	15

Decision making:
We used two types of decision-making approach:

1) **In Mode == SAME_ENVIRONMENT,** when both agents are in the same environment we rely on the values from the sensor and temperature to decide on the meaning of bit stream received. Here we are assuming there is an external mechanism to know which mode the agents are operating in. For example, a button can be used to indicate the mode of operation.

2) **In Mode == DIFFERENT_ENVIRONMENT,** for this mode we use a bayes classifier to correct predict/ perceive the meaning of the received bit stream. This node can also be considered as a fallback mechanism.

- We use Naïve bayes classifier to predict the word listened from an opposite agent.
- Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation: • $P(A|B) = \{P(B|A) * P(A)\} / \{P(B)\}$
- Basically, we are trying to find probability of event A, given the event B is true.

- P(A) is called priori of A (i.e., Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B).
- P(A|B) is a posteriori probability of B, i.e., probability of event after evidence is seen
- Now, with regards to our dataset, we can apply Bayes' theorem to create a classifier model. Consider the receiving agent has received a bitstream 10101001and now it wants to predict the label (class) it belongs to using priori probability data (listening table) created during the learning process. So, we can check that P("DARK_COLD") given bitstream 10101001.

P("DARK_COLD" | 10101001) = {P(10101001| "DARK_COLD") * P("DARK_COLD")} / {P(10101001)}

Denominator probability value will be constant for all other probability, so we can ignore it.
P("DARK_COLD"| 10010101) = {P(10010101| "DARK_COLD") * P("DARK_COLD")}

Using the above table 1, we can calculate the best conditional probability which can be used to predict the word label with the highest likelihood.

If received bitstream is 10010101	
P(10101001|"DARK_COLD")*P("DARK_COLD")	(10/40) *(13/40)
P(10101001|" DARK")*P("DARK")	0
P(10101001|" DARK_MEDIUM")*P("DARK_MEDIUM")	(2/40) *(13/40)
P(10101001|" DARK_COLD")*P("DARK_COLD")	0
P(10101001|" BRIGHT_MEDIUM")*P("BRIGHT_MEDIUM")	0
P(10101001|" BRIGHT_COLD")*P("BRIGHT_COLD")	0

Based on above table we can see that probability of received word "DARK_COLD" is maximum. Hence as per Naïve Bayesian formula, 10101001can be categorized as "DARK_COLD".

2. Routines

2.1 **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.

2.2 **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.

2.3 **create_word**(): This API will generate labels for each light and temp value read and based on the generated labels a bit stream is generated. The generated word encoding describes a light – temp condition. This bit stream is used during the speak operation.

2.4 **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.

2.5 **quicksort**(): A quick sort for the vocabulary tables is performed based on the frequency values of each word entry in the table

2.6 **remove**(): This API is used to remove word entries below a threshold frequency from the reorder vocabulary tables.

2.7 **speak**(): This API is used to output the words generated using speakers connected. The PWM generates the signals required to drive the speaker. The PWM operates at 1.4Khz. The PWM is turned off and on based on 1s and 0s bit values of the bit stream generated by the system.

2.8 **blocking_delay**(): This API will perform a blocking delay operation

2.9 **Swap**(): Used for the reordering algorithm

2.10 **Part**(): Partitioning for the quick sort algorithm.

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

2.12 **Timer16_1_ISR**() : This ISR used to maintain a clock for the system. This timer interrupt is triggered every 100ms. This is used for the listening operation. A global variable **wElapsedTime** will be incremented every time this ISR is triggered.

2.13 **Listen**(): This API performs the listening operations. It will listen for X seconds of duration,and listen for the one stamp at a time. At X seconds the API will be able to detect the bit stream that the other agent is transmitted via sound waves. If the average ADC value of the mic exceeds more that loud stamp threshold value of 0x200 we detect that a loud stamp is transmitted by the other agent whereas if the average ADC value is less this threshold, we detect it as silent stamp.

2.14 **check_if_labelPresent():** This API is used to create a label based on the agent's temp and light value and then see if this label is present in the neighbors' table.

2.15 **check_if_bit_stream_match():** This API checks if the bitstream associated to the received word matches with the bitstream of the word which is already present in the neighbors table.

2.16 **re_learn_ranges**(): This API is used to re-adjust the ranges of temp/light which is used to generate the bitstream/word encoding for the system. This will help in reducing the frequency of wrongly predicted words in the learning table. Thereby improving the accuracy of the prediction of the bayes classifier. This API has the capability to understand is the words listened is a combination of light and temperature or only light or only temperature.

2.17 **highest_conditional_prob**(): This API is used to predict the word based on the word/bitstream received, when not in the same environment. This API will return the label of the predicted class. More details of the working of the API is explained above.

3. Data structures

An array is used to store the two vocabulary tables. Each element of the array is a strut consisting of three elements. Similarly, an array is used to store one learning table. The learning table have element to store word encoding, label, frequency and frequency of mismatch.

```
typedef struct vocabTable
{
    BYTE word;
    enum comb label;
    BYTE frequency;
    BYTE age;
}vocabTable_type;
typedef  struct learn_table
{
    BYTE word;
    enum comb label;
    BYTE frequency;
    BYTE frequency_of_mismatch;
}learn_table_type;

vocabTable_type my_table[SIZE]= {0};
vocabTable_type neighours_table[SIZE] = {0};
learn_table_type bayesian_learn_table[18] = {0};
```

4. Interfacing of Signals to PSoC1

4.1 **Temperature Sensor MCP9808:**
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.

4.2 **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.

4.3 **Microphone:** 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.

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

5. Hardware resources

5.1 **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 are 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 = 1.4Khz. Therefore, the audio signal generated by speaker will be 1.4Khz.

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

5.2 **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

5.3 **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

5.4 **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 SC

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

5.5 **Timer module and LCD:** In addition, we are using LCD and Timer user modules. This is a 16-bit timer 1 that generates an interrupt signal every 100ms (T)seconds. It is placed in DCC12 and DCC13 digital blocks.

6. Timing Complexity and Size:

6.1 Timing (clock cycles)and size(bytes) of APIs:

API	clk cycles	size
getLux	52	19
read_temperature	30	11
create_word	188	60
update_table	38	13
quick_sort	44	16
remove	24	9
speak	10	4
blocking_delay	8	2
swap	13	4
part	13	4
TimerISR	24	7
Timer_16_1_ISR	18	6
Listen	13	4
check_if_labelPresent	13	4
check_if_bit_stream_match	46	14
re_learn_ranges	36	13

Ram and ROM usage:

```
ROM 45% full. 7252 out of 16384 bytes used (does not include absolute areas).
RAM 25% full. 188 bytes used (does not include stack usage).
Built with ICCM8C STD V7.05.00
```

6.2 **Complexity:** Re-order operation is still having highest complexity.

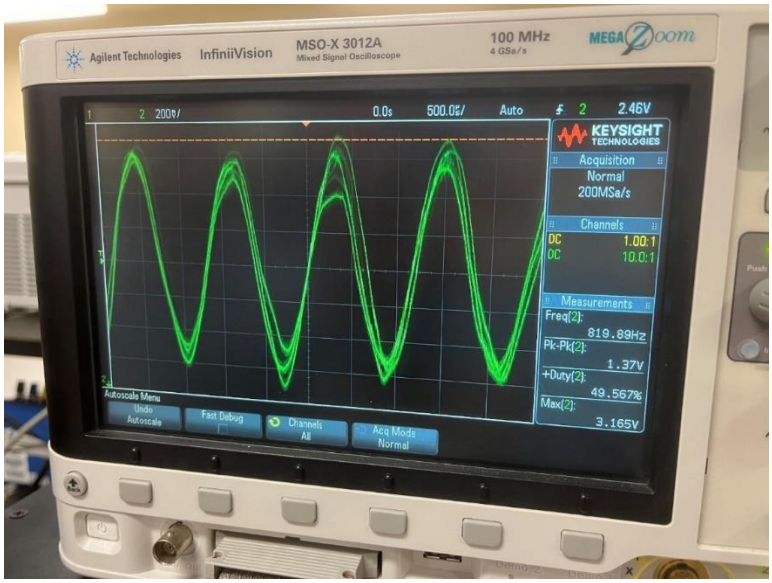
Routine	Complexity
`update_table`	O(n), n= number of words in vocabulary
`Quick sort`	O(nlogn)
`Remove`	O(n)
`Reorder (Sorting + Remove)`	O(nlogn) + O(n)

7. Tradeoff

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.

8. Testing / Debugging:

8.1 We verified that the mic is working by tapping the analog output of the PGA 1 to which the mic is connected. We used signal generator to drive the speaker, and this was kept next to the mic. The tapped signal of the mic matched the signal and frequency of the sinusoidal signal of the signal generator. Below is a photo showing the tapped signal in the CRO.



8.2 **Verify that the transmission and receiving of bit stream:** To do that we made one of the agents to transmit bitstream associated to the word “DARK_MEDIUM” which is 10101001. We were able to get the same sequence our host agent. Y and Z were given value 2 seconds. We print the average ADC value of the mic and found that for a loud stamp duration our ADC value showed an average value greater that hex value 0x200.This ADC value was print on the LCD. And for a loud stamp we observed that we the average adc value during silent stamp duration was less that 0x200 which made us verify that silent stamp was transmitted.
In the figure 1 below you can see how the receiver agent is showing a value of 298 average adc value indicating that loud stamp. In the figure 2 below you can see how the receiver agent is showing a value of 1C8 average adc value indicating that silent stamp. The transmission functionality was also verified here.

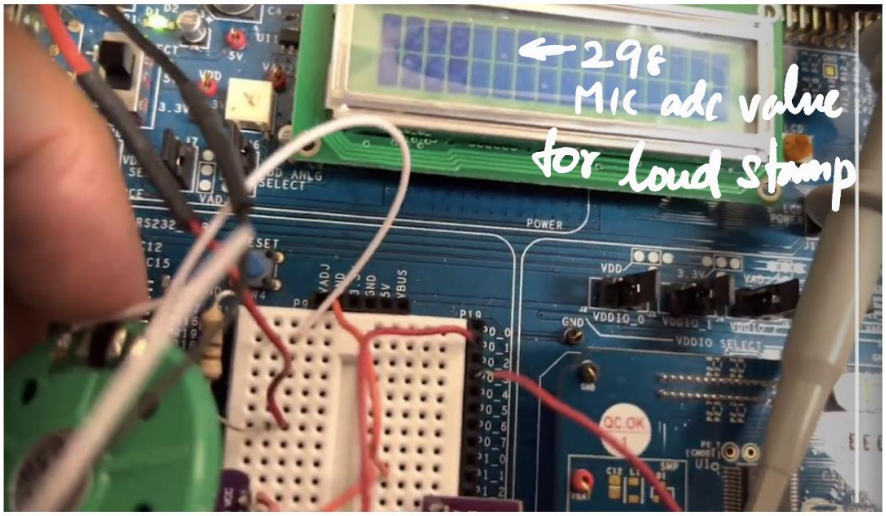


Figure 1 Loud stamp

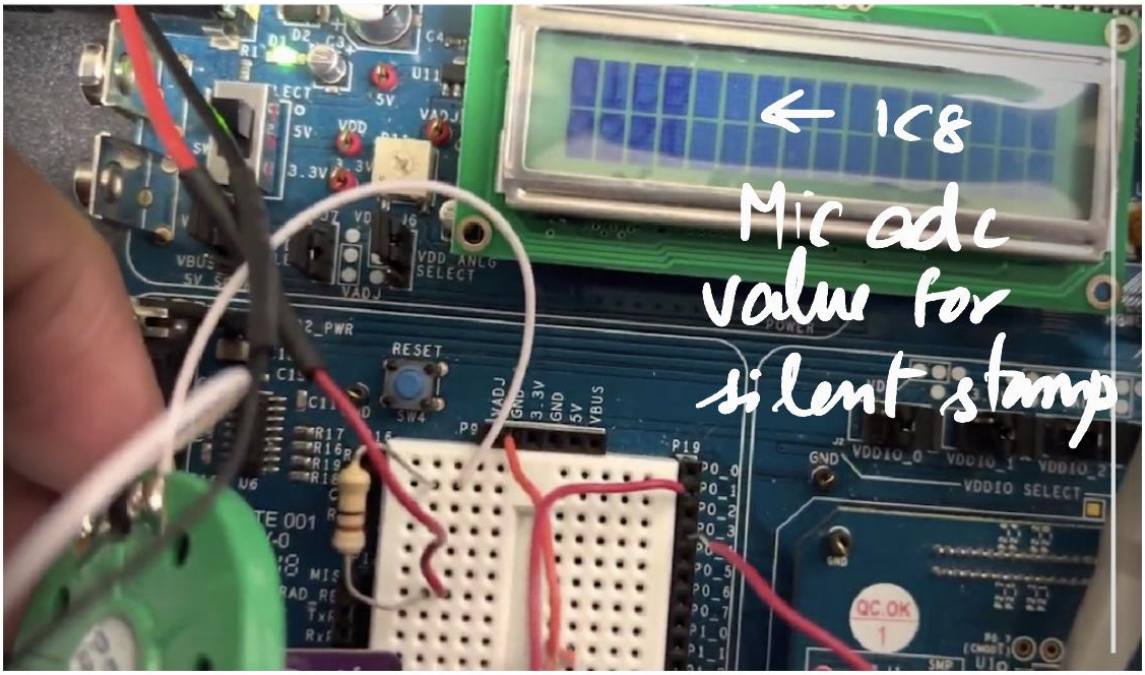


Figure 2 silent stamp

9. Result/observations:

9.1 In the below image you can see the case where two agents where working in the same environment. The figure 3 shows how one of agent is speaking the work DARK MED and other once is listening to mode. Figure 4 shows how the agent after listening was able to detect the correct word. Here what we need to understand is that both agents are following the same rule for word generation and that's why the receiving agent was able to perceive correctly, because they are in the same environment. In the figure 5, you can see the case where the sender agent extended the meaning of DARK MED, where the agent is considering temperature value of 8 in the medium category (indicating range change). But the receiver agent is still having using the same range to understand the received bit stream. The receiver agent will now kick start the range adjusting algorithm to correct this new range change. After this correction, the receiver agent will adapt its vocabulary generation rule to reflect to the adaptation of sender.

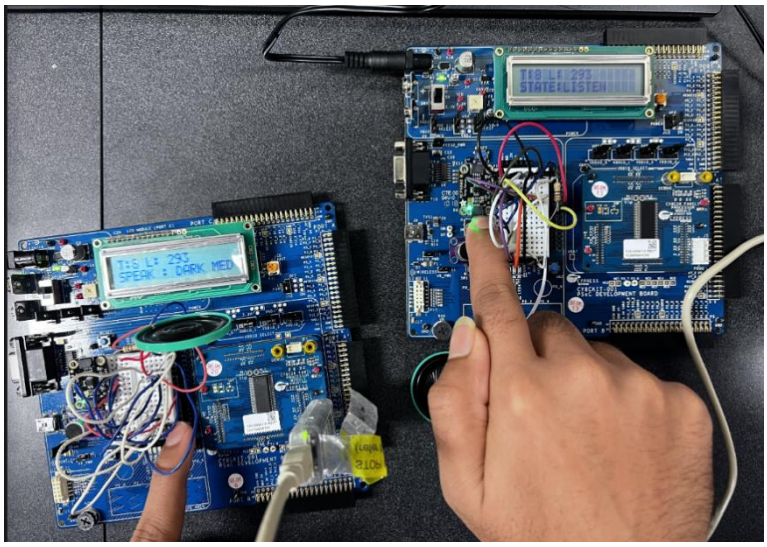


Figure 3 Speaking and listening

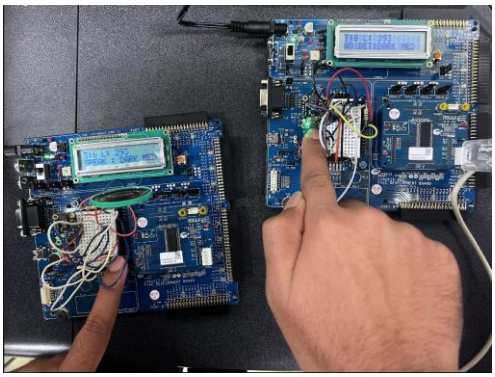


Figure 4 Detecting Correctly

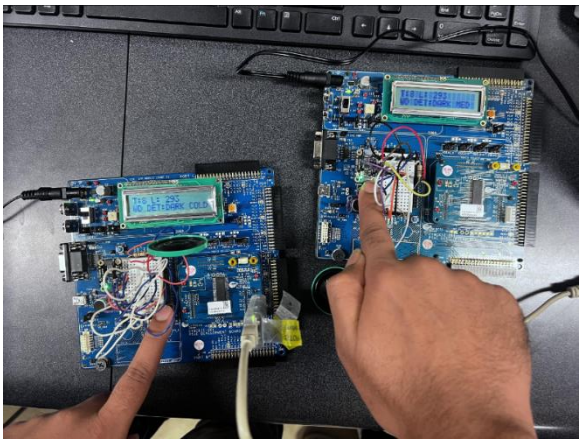


Figure 5 Detecting wrongly(trigger re-learning of ranges)

9.2 We observed that with when the X, Y, Z values where decreased form initial value of Z=Y=2s and X = 16s to Z=Y=200ms, X=1.6s, even though the speed of detection increased, we observed that the adc value of the temperature didn't settle down early, enough. So that average adc value sometime resulted in below the threshold of 0x200. This resulted in agent detecting a silent stamp instead of loud stamp. With the Z=Y=2s and X = 16s this was not observed.

9.3 In our case the maximum vocabulary size is 15, but if this value needs to be increased then the table needs to clear of less frequent word in a frequent manner. Therefore, the Q value should be reduced further from 60. Otherwise, dthe less frequent, words in the vocabulary will unnecessarily consumer more re-ordering time.

10. Improvement:

10.1 Selecting of the mode should be automatically done by the system without any external intervention. An algorithm should be detected to fall back to the bayes classifier detect approach from the sensor-based approach when seeing a drop in the accuracy of prediction resulted from the other agent moving to a different environment of different temperature and light conditions.

11. Appendix

```
#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                         1600 // 1.6s
#define Z                         200  // 200ms
#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
#define LS_NIBBLE_MASK            0x0F
#define MS_NIBBLE_MASK            0xF0

#define LOUD_STAMP_THRESHOLD      0x200

enum light  {DARK, NORMAL , BRIGHT};
enum temp   {COLD, MEDIUM, HOT};
enum comb   {DARK_COLD,
             DARK_MEDIUM,
             DARK_HOT,
             NORMAL_COLD,
             NORMAL_MEDIUM,
             NORMAL_HOT,
             BRIGHT_COLD,
             BRIGHT_MEDIUM,
             BRIGHT_HOT,
             DARK_ONLY,
             NORMAL_ONLY,
             BRIGHT_ONLY,
             COLD_ONLY,
             MEDIUM_ONLY,
             HOT_ONLY};
char lookuptable[15] = {"DARK_COLD",
                        "DARK_MEDIUM",
                        "DARK_HOT",
```

```
                                "DARK_MEDIUM",
                                "DARK_HOT",
                                "NORMAL_COLD",
                                "NORMAL_MEDIUM",
                                "NORMAL_HOT",
                                "BRIGHT_COLD",
                                "BRIGHT_MEDIUM",
                                "BRIGHT_HOT",
                                "DARK_ONLY"
                                "NORMAL_ONLY",
                                "BRIGHT_ONLY",
                                "COLD_ONLY",
                                "MEDIUM_ONLY",
                                "HOT_ONLY"
}
typedef struct vocabTable
{
    BYTE word;
    enum comb label;
    BYTE frequency;
    BYTE age;
}vocabTable_type;
typedef  struct learn_table
{
    BYTE word;
    enum comb label;
    BYTE frequency;
    BYTE frequency_of_mismatch;
}learn_table_type;

vocabTable_type my_table[SIZE]= {0};
vocabTable_type neighours_table[SIZE] = {0};
learn_table_type bayesian_learn_table[18] = {0};

// This is the initial default ranges that both the systems will start with
INT lux_range_1 = 333;
INT lux_range_2 = 666;
INT temp_range_1 = 10;
INT temp_range_2 =23;
/*
typedef struct learning_table
vocabTable_type my_table[SIZE]= {0};             // To do: finalize on the size
vocabTable_type neighours_table[SIZE] = {0};     // To do: finalize on the size
learn_table_type bayesian_learn_table[18] = {0};// To do: finalize on the size


INT wElapsedTime = 0;                       // timer with us resolution

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, char *label,INT l1,INT l2,INT t1,INT t2);
void blocking_delay(BYTE bTimes);
BOOL update_table(BYTE word_encoding, vocabTable_type* table,char * label);
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);
void remove(BYTE freq_threshold,vocabTable_type* table );
void speak(BYTE word_encoding);
BYTE listen(void);
BOOL check_if_labelPresent(char * index);
BOOL check_if_bit_stream_match(char index,BYTE word_listened);
void re_learn_ranges(BYTE word_listened);
BYTE highest_conditional_prob(void);

#pragma interrupt_handler Timer16_1_ISR
void Timer16_1_ISR(void )
{
      wElapsedTime++;
}
#pragma interrupt_handler ResetSwitchISR
// This ISR is triffered every 60 seconds
void TimerISR(void )
{
    sortVocabTables = TRUE;
}

void ResetSwitchISR(void )
{
```

```
void main(void)
{
    float luxValue; // Ambient Light illumiance value
    INT   tempValue;     // Ambient temperature
    BYTE  speak_word;
    BYTE  word_listened;
    BOOL neword;
    char * label;
    char * index;

    // 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();

    //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 ///////////////////
    speak_word= create_word(luxValue, tempValue,label, lux_range_1,lux_range_2,temp_range_1,temp_range_2);


    // Update my vocab
    update_table(speak_word,my_table,label);


    // 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(Q,my_table);
    }
    // Speak ////////////////////////////
    if(neword)
    {
        speak(speak_word);
    }
    // Listen the bit stream from the other agent
    word_listened = listen();

    // Check if the listened word needs to be entered in the neighbour's vocab table
        // Check if the received bit stream represents a light condition or temp condition or both.
        // For example, if listened bit steam is 0000 1010, since the Most Significant 4 bit representing Light
        // this word bit stream is represents only a temperature condition.
        // If the a same word label is already present but with a different bit stream it means the sender age
        // to include a new value for temperature or light conditions. In this case, re-adjust or re learn the
```

```
        if(check_if_labelPresent(index))
        {
            //Check bitstream associated to the receieved word matches with the bitstream of the word which is alrea
            if(check_if_bit_stream_match(*index,word_listened))
            {
                //If bit stream matches re_learning of ranges not required. Other agent hasn't extended its meaning.
            }
            else
            {
                //Bit stream doesn't match. Adjust the ranges of temp/light and generate the bitstream/word_encoding
                //matches with the received bit stream
                re_learn_ranges(word_listened);

                // This re learning of ranges will inturn improve the bayesian leaning process and will improve the
                // prediction when in different environment.
            }
        }
        else
        {
            //It is a new word. update the table with new word received.
            update_table(word_listened,neighours_table,NULL);
        }

        if (mode == SAME_ENVIRONMENT)
        {
            //Learn only when in same evironment
            //Used for bayesian word prediction when mode == Different environment.
            update_frequency_table(word_listened);
        }

        //-----------------Decesion Making---------------------------
        if( mode == SAME_ENVIRONMENT)
        //Use sensor values of the agent to predict the meaning of the word received
        {
            // This wil print the predicted word
            create_word(luxValue, tempValue,label, lux_range_1,lux_range_2,temp_range_1,temp_range_2);
        }

        if( mode == DIFFERENT_ENVIRONMENT)
        {
            //Use bayesian table to predict the meaning of the word associated to the received bit stream   */
```

```
        {
            //Use bayesian table to predict the meaning of the word associated to the received bit stream
            predicted_label = highest_conditional_prob();
            for(i<0;i<SIZE;i++)
            {
                if(predicted_label == i)
                {
                    LCD_2_Position(0,0);
                    LCD_2_PrCString("Word Predicted:");
                    LCD_2_Position(1,0);
                    //LCD_2_PrCString("%s",lookuptable[predicted_label]);
                }
            }
        }
    }
}

BYTE create_word(float lux, INT temp, char *label,INT l1,INT l2,INT t1,INT t2)
{
    enum light light_label;
    enum temp temp_label;
    BYTE bit_stream;

    if (lux >= 0 && lux < l1)
        light_label = DARK;
    else if(lux >= l1 && lux < l2)
        light_label = NORMAL;
    else if(lux >=l2 && lux <=1000)
        light_label = BRIGHT;

    // Check this
    if (temp < t1)
        temp_label = COLD;
    else if(temp >=t1 && temp < t2)
        temp_label = MEDIUM;
    else if(temp >= t2)
        temp_label = HOT;

    if (light_label == DARK && temp_label == COLD)
    {
        bit_stream = 0xa9;//Bit stream : 1010 1001 | Dark light and Cold condition
```

```
    {
        bit_stream = 0xa9;//Bit stream : 1010 1001 | Dark light and Cold condition
        *label = DARK_COLD;
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: DARK COLD");
    }
    else if (light_label == DARK && temp_label == MEDIUM)
    {
        bit_stream = 0xaa; //Bit stream : 1010 1010 |  Dark light and Medium temp condition
        *label = DARK_MEDIUM;
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: DARK MEDIUM");
    }
    else if (light_label == DARK && temp_label == HOT)
    {
        bit_stream = 0xa8;//Bit stream : 1010 1000 | Dark light and Hot temp condition
        *label = DARK_HOT;
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: DARK HOT");
    }
    else if (light_label == NORMAL && temp_label == COLD)
    {
        bit_stream = 0x89; //Bit stream : 1000 1001 | Normal light and cold temp condition
        *label = NORMAL_COLD;
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: NORMAL COLD");
    }
    else if (light_label == NORMAL && temp_label == MEDIUM)
    {
        bit_stream = 0x8a; //Bit stream : 1000 1010 | Normal light and medium temp condition
        *label = NORMAL_MEDIUM;
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: NORMAL MEDIUM");
    }
    else if (light_label == NORMAL && temp_label == HOT)
    {
        bit_stream = 0x88; //Bit stream : 1000 1000 | Normal light and hot temp condition
        *label = NORMAL_HOT;
        LCD_2_Position(0,0);
        LCD_2_PrCString("Word Generated is: NORMAL HOT");
    }
      else if (light_label == BRIGHT && temp_label == COLD)
      {
          bit_stream = 0x49; //Bit stream : 01001001 | Bright light and cold temp condition
          *label = BRIGHT_COLD;
          LCD_2_Position(0,0);
          LCD_2_PrCString("Word Generated is: BRIGHT COLD");
      }
      else if (light_label == BRIGHT && temp_label == MEDIUM)
      {
          bit_stream = 0x4a; //Bit stream : 01001010 | Bright light and medium temp condition
          *label = BRIGHT_MEDIUM;
          LCD_2_Position(0,0);
          LCD_2_PrCString("Word Generated is: BRIGHT MEDIUM");
      }
      else if (light_label == BRIGHT && temp_label == HOT)
      {
          bit_stream = 0x48; //Bit stream : 01001000 | Bright light and hot temp condition
          *label = BRIGHT_HOT;
          LCD_2_Position(0,0);
          LCD_2_PrCString("Word Generated is: BRIGHT HOT");
      }
      return bit_stream;// return word encoding
  }


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

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

```
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)
        {
            // Remove the old word and details and initial with zero
            table[i].word= 0;
            table[i].frequency = 0;
            table[i].age= 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)
```

```
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)
    {
        // Type(2 bits) | spacer(1 bit) | payload (17 bits) (so in total 20bits of bit stream per work speak operatio
        // One word consists of 8 stamps
        case 0xa9: //1010 1001 4 loud stamp, 4 silent stamp | Dark light and Cold condition
            PWM8_1_Start();
            blocking_delay(Z_div);//1
            PWM8_1_Stop();
            blocking_delay(Y_div);//0
            PWM8_1_Start();
            blocking_delay(Z_div);//1
            PWM8_1_Stop();
            blocking_delay(Y_div);//0
            PWM8_1_Start();
            blocking_delay(Z_div);//1
            PWM8_1_Stop();
            blocking_delay(2*Y_div);//00
            PWM8_1_Start();
            blocking_delay(Z_div);//1
            PWM8_1_Stop();

            /*y = Y_div* 16;
            blocking_delay(y);*/
            break;;
        case 0xaa:// 1010 1010 4 loud stamp, 4 rest silent stamps |  Dark light and Medium temp condition
            PWM8_1_Start();
            blocking_delay(Z_div);//1
            PWM8_1_Stop();
            blocking_delay(Y_div);//0
            PWM8_1_Start();
```

```
    case 0xaa:// 1010 1010 4 loud stamp, 4 rest silent stamps |  Dark light and Medium temp condition
        PWM8_1_Start();
        blocking_delay(Z_div);//1
        PWM8_1_Stop();
        blocking_delay(Y_div);//0
        PWM8_1_Start();
        blocking_delay(Z_div);//1
        PWM8_1_Stop();
        blocking_delay(Y_div);//0
        PWM8_1_Start();
        blocking_delay(Z_div);//1
        PWM8_1_Stop();
        blocking_delay(Y_div);//0
        PWM8_1_Start();
        blocking_delay(Z_div);//1
        PWM8_1_Stop();
        blocking_delay(Y_div);//0
        /*y = Y_div *14;
        blocking_delay(y);*/
        break;
    case 0xa8:// 1010 1000 3 loud stamp,5 rest silent stamps | Dark light and Hot temp condition
        //To do
        break;
    case 0x89:// 1000 1001  3 loud stamp,5 rest silent stamps |  Normal light and cold temp condition
        //To do
        break;
    case 0x8a:// 1000 1010  3 loud stamp and 5 rest silent stamps | Normal light and medium temp condition
        //To do
        break;
    case 0x88:// 1000 1000 2 loud stamp, 6 rest silent stamps | Normal light and hot temp condition
        //To do
        break;
    case 0x49:// 01001001 3 loud stamp,5 rest silent stamps | Bright light and cold temp condition
        //To do
        break;
    case 0x4a:// 01001010 3 loud stamp,5 rest silent stamps | Bright light and medium temp condition
        break;
    case 0x48:// 0100 1000 2 loud stamp, 6 rest silent stamps | | Bright light and hot temp condition
        //To do
```

```
BYTE listen(void)
{
    int iResult1,iResult2,avg_mic_output,count,total;
    INT start_time =wElapsedTime;
    INT bit_start_time;
    INT X_time_elaspsed = 0;
    INT ZY_time_elaspsed = 0;
    BYTE bit_stream = 0;
    // Listen for a word duration to heat the bit stream sent by the other agent
    while (X_time_elaspsed < X)
    {
        // Listen for lound or silen stamp for the Y = Z = 200ms duration
        avg_mic_output = 0;
        count = 0;
        total = 0;
        while ((ZY_time_elaspsed < Z || ZY_time_elaspsed < Y))
        {
            ZY_time_elaspsed = wElapsedTime - bit_start_time;

            while(DUALADC_1_fIsDataAvailable() == 0);  // Wait for data to be ready

            iResult1 = DUALADC_1_iGetData1();           // Get Data from ADC Input1
            iResult2 = DUALADC_1_iGetData2();

            DUALADC_1_iGetData1ClearFlag();
            DUALADC_1_iGetData2ClearFlag();
            count++;
            total = total + iResult1;
            avg_mic_output = total/count;
        }
        // Detected loud stamp
        if (avg_mic_output> LOUD_STAMP_THRESHOLD)
        {
            bit_stream = bit_stream | 0x1;
        }
        // Detected silent stamp
        else
        {
            bit_stream = bit_stream | 0x0;
```

```
        {
            bit_stream = bit_stream | 0x0;
        }
        //shift the bit stream for the next bit value
        bit_stream = bit_stream <<1;

        X_time_elaspsed = wElapsedTime - start_time;

        bit_start_time = wElapsedTime;
        ZY_time_elaspsed = 0;
    }

    return bit_stream;
}

// Check if word label is present in neighbours for the current temp&/light condition
BOOL check_if_labelPresent(char* index)
{
    float luxValue; // Ambient Light illumiance value
    INT   tempValue;     // Ambient temperature
    char * label;
    BOOL present =FALSE;
    int i;
    // Sense ////////////////////////////

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

    //Create a word label based on the sensor values
    create_word(luxValue, tempValue,label,lux_range_1,lux_range_2,temp_range_1,temp_range_2);

    // Check if this label is present in neighbour's table
    for (i= 0; i< SIZE; i++)
    {
       if (neighours_table[i].label == *label)
        {
            present =TRUE;
            *index = i;
        }
```

```
            *index = i;
        }
    }
    return present;
}
BOOL check_if_bit_stream_match(char index,BYTE word_listened)
{
    if(neighours_table[index].word == word_listened)
        return TRUE;
    else
        return FALSE;
}

void re_learn_ranges(BYTE word_listened)
{
    float luxValue; // Ambient Light illumiance value
    INT   tempValue;
    char *label;
    //Read light sensor value
    luxValue   = getLux(); // luxValue ranges from 0 to 1000
    //Read temperature value
    tempValue  = read_temperature();

    //check if the received word indicates temperature only condition or light only condition or both temperature&light condition
    if ((word_listened & MS_NIBBLE_MASK != 0) && (word_listened & LS_NIBBLE_MASK != 0))
    {
        // generate the bit stream for each different combinations of range values, the find the new range that will
        // generate the bit stream that matches with the received bit stream
        if (create_word(luxValue, tempValue,label,luxValue,lux_range_2,temp_range_1,temp_range_2) == word_listened)
        {
            lux_range_1 = luxValue;
        }
        else if (create_word(luxValue, tempValue,label,lux_range_1,luxValue,temp_range_1,temp_range_2) == word_listened)
        {
            lux_range_2 = luxValue;
        }
        else if (create_word(luxValue, tempValue,label,lux_range_1,lux_range_2,tempValue,temp_range_2) == word_listened)
        {
            temp_range_1 = tempValue;
        }
        else if (create_word(luxValue, tempValue,label,lux_range_1,lux_range_2,temp_range_1,tempValue) == word_listened)
```

```
        }
        else if (create_word(luxValue, tempValue,label,lux_range_1,lux_range_2,temp_range_1,tempValue) == word_listened)
        {
            temp_range_2 = tempValue;
        }
    }
    else if((word_listened & MS_NIBBLE_MASK ==0) && (word_listened & LS_NIBBLE_MASK != 0))
    {
        //relearn only temperature ranges
        if (create_word(luxValue, tempValue,label,lux_range_1,lux_range_2,tempValue,temp_range_2) == word_listened)
        {
            temp_range_1 = tempValue;
        }
        else if (create_word(luxValue, tempValue,label,lux_range_1,lux_range_2,temp_range_1,tempValue) == word_listened)
        {
            temp_range_2 = tempValue;
        }
    }
    else if ((word_listened & MS_NIBBLE_MASK !=0 ) && (word_listened & LS_NIBBLE_MASK == 0))
    {
        //relearn only light ranges
        if (create_word(luxValue, tempValue,label,luxValue,lux_range_2,temp_range_1,temp_range_2) == word_listened)
        {
            lux_range_1 = luxValue;
        }
        else if (create_word(luxValue, tempValue,label,lux_range_1,luxValue,temp_range_1,temp_range_2) == word_listened)
        {
            lux_range_2 = luxValue;
        }
    }
}
void update_frequency_table(BYTE word_encoding)
{
    int i;
    BOOL found = FALSE;
    char * label;

    float luxValue; // Ambient Light illumiance value
    INT   tempValue;
    char *label;
    //Read light sensor value
```

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

    //Create a word label based on the sensor values
    create_word(luxValue, tempValue,label,lux_range_1,lux_range_2,temp_range_1,temp_range_2);

    for (i= 0; i< SIZE; i++)
    {
        table[i].age++;
        if (bayesian_learn_table[i].word == word_encoding)
        {
            found =TRUE;
            table[i].frequency++;
        }
    }
    if (found == FALSE)
    {
        table[i].word = word_encoding;
        table[i].frequency = 1;
        table[i].label = *label;
        table[i].frequency_of_mismatch = 0;
    }
    return found;
}

BYTE highest_conditional_prob(BYTE word_listened)
{
    INT highest_c_prob=0;
    INT p1=0;
    int index=0;
    for (i= 0; i< SIZE; i++)
    total = total + bayesian_learn_table[i].frequency;

    for (i= 0; i< SIZE; i++)
    {
        if (bayesian_learn_table[i].label == word_listened && bayesian_learn_table[i].label == label )
        {
            p1 = bayesian_learn_table[i].frequency/total;
```

```
}

BYTE highest_conditional_prob(BYTE word_listened)
{
    INT highest_c_prob=0;
    INT p1=0;
    int index=0;
    for (i= 0; i< SIZE; i++)
    total = total + bayesian_learn_table[i].frequency;

    for (i= 0; i< SIZE; i++)
    {
        if (bayesian_learn_table[i].label == word_listened && bayesian_learn_table[i].label == label )
        {
            p1 = bayesian_learn_table[i].frequency/total;
        }
        if (bayesian_learn_table[i].label == word_listened)
        {
            p2 = bayesian_learn_table[i].frequency/total;
        }
        temp  = p1*p2;
        if (temp > highest_c_prob)
        {
            highest_c_prob = temp;
            index = i;
        }
    }
    return bayesian_learn_table[index].label;
}
```