ARM® Cortex®-M4

32-bit Microcontroller

NuMicro® Family
NuMaker Brick
Design Guide

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

List of Figures

List of Tables

1 OVERVIEW

NuMaker Brick is a highly integrated platform designed to support IoT (Internet of Things) for users to develop their designs quickly. Every sensor has its own function. The combination module allows the NuMaker Brick platform to create application-specific products with high flexibility. Besides, the NuMaker Brick provides an expansion board such that users can create their own specific sensor and extend the function on the NuMaker Brick platform. Since the biggest challenge for developers is not to add new functions, but to integrate the other module, the NuMaker Brick platform is equipped with a unified communication protocol such that users can add a new module designed only in accordance with the specifications for the communication protocol. This can significantly reduce the complexity of developing the new modules and help developers to focus more on developing new modules to meet the product schedule.

This document guides users to add a new module on the NuMaker Brick platform step by step from basic platform architecture introduction to software design concept and communicated protocol. This document also assists developers to build a complete platform operation concept based on detailed explanation of a practical example.

2 NUMAKER BRICK PLATFORM INTRODUCTION

2.1 NuMaker Brick Platform Concept

The NuMaker Brick platform consists of a master module and some slave sensor modules. The master module collects and processes all the parameter received from each of the sensor modules. Before receiving or transmitting parameter from/to the master module, each of sensor modules always needs to wait for the call and execute the received commands from master module. Figure 2-1 shows the combination example of NuMaker Brick Platform between a master module and two slave sensor modules.

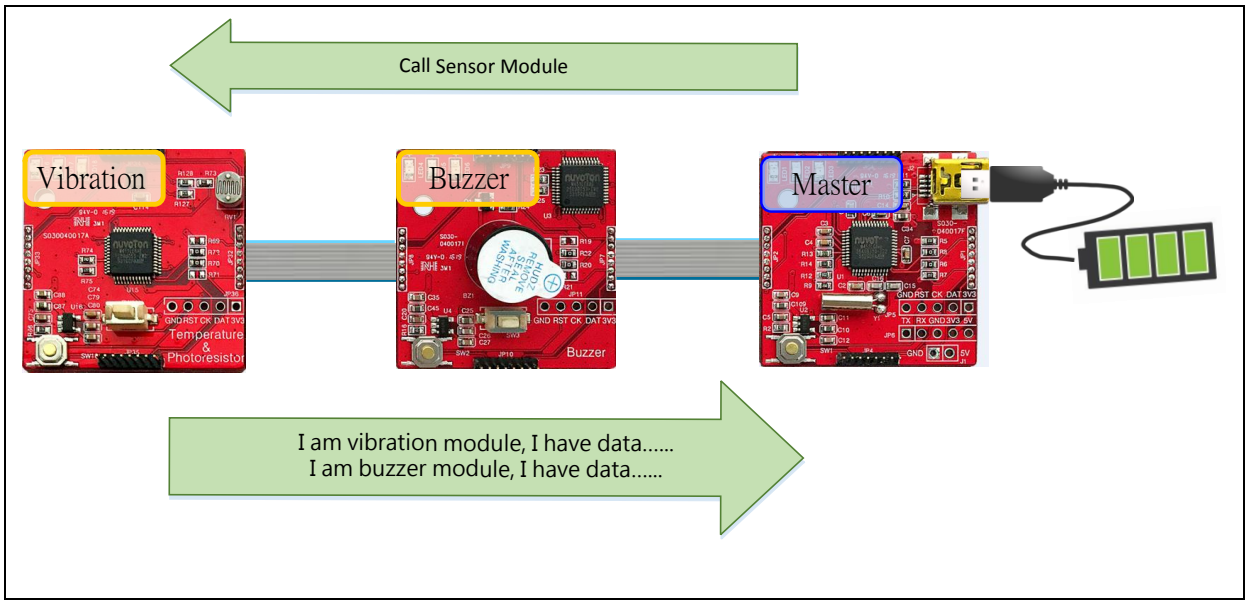

Figure 2-1 Combination Example of NuMaker Brick Platform

This chapter introduces the NuMaker Brick platform program and its main function. The main function is to put in main.c and can be divided into two parts: setup() and loop(). Setup() is used to do the initial operation; loop() is where the NuMaker Brick platform executes program. The next chapter will introduce the two parts and the entire framework of the platform.

To modularize program, the modules are classified by the following functions:

- **Module initialization:** Initialization function for each module
- **Function executing program periodically:** Module is required to execute periodically.
- **Function executing program immediately:** Module is required to execute immediately.

Figure 2-2 shows all the function-calls of sensor modules in modular programming table and the classification of sensor functions. Users can easily add new functions of new modules into this structural function table according to their demands in application system.

```
APFN_FUNC_T pfnDevFunc[MAX_TID_DEV] =
{
      /*
         Functions for each device:

          {Initial, Period, Pulling, Report}

          Initial : Init and configure device
          Period  : Process device sensor parameter  or set sensor once per 0.1
seconds.
          Polling : Process device sensor parameter or set sensor frequently.
          Report  : Report process parameter to master
      */

      /* for embedded device boards */
```

Sensor module initialization | Sensor functions executed periodically | Sensor functions executed immediately

```
      {Battery_Init,    MasterControl,    NULL,              report_battery},
      {Buzzer_Init,     NULL,             Buzzer_Control,    report_buzzer},
      {Led_Init,        NULL,             Led_Control,       report_led},
      {AHRS_Init,       AHRS_Control,     AHRS_Cal,          report_ahrs},
      {SonarInit,       SonarDetect,      SonarTimeOutCheck, report_sonar},
      {HTU21D_Init,     WaitHTU21D,       GetHTU21DTemp,     report_temp},
      {Gas_Init,        GetGas,           NULL,              report_gas},
      {IR_Init,         IR_Control,       IR_Check,          report_ir},
      {key_init,        NULL,             NULL,              report_key},

      /* for custom device boards
         fill your four board functions here to specified ID. */
```

Users can define sensor functions by themselves

```
      {Lr_Init,    GetLr, NULL, report_resDev9},    // For device ID 9
      {NULL,       NULL,  NULL, report_resDev10},   // For device ID 10
      {NULL,       NULL,  NULL, report_resDev11},   // For device ID 11
      {NULL,       NULL,  NULL, report_resDev12},   // For device ID 12
      {NULL,       NULL,  NULL, report_resDev13},   // For device ID 13
      {NULL,       NULL,  NULL, report_resDev14}    // For device ID 14
};
```

Figure 2-2 Function-Calls of Sensor Modules in Modular Programming Table

2.2 Function-Call Setup() Description

Figure 2-3 shows the setup() function-call initialization flowchart.

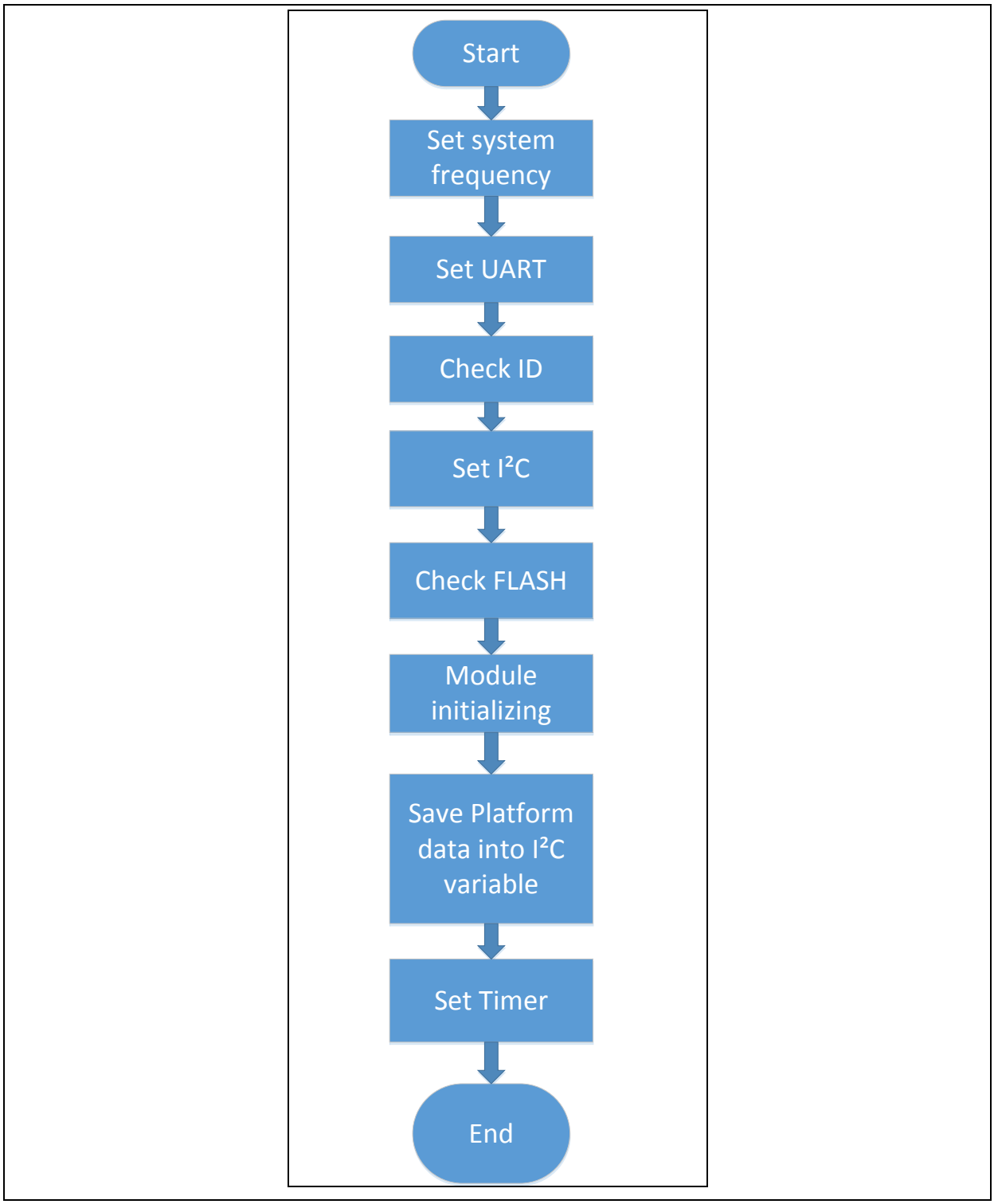

Figure 2-3 Setup() Initialization Flowchart

The setup() function-call flow is described as follows. Figure 2-4 shows the whole program code of setup() function-call.

1. Set the system clock as 72MHz for CPU.
2. Set the baud rate as 115200bps for UART.
3. Check ID to identify this module is Master or Slave device on I^{2}C bus of this NuMaker Brick platform.
4. Initialize the I^{2}C peripheral of this module by ID.
5. Initialize the Flash memory.
6. Initialize the module, including four steps:
   - Check ID
   - Initialize module function
   - Check whether there is a corresponding parameter in Flash. Read Flash data according to the value of ID
   - Set the period of function executing
7. Set the module parameter for I²C
   - The master module saves the next parameter that will be transmitted to the sensor module.

```
void setup()
{
        int i;

        setupSystemClock();
        setupUART();
        setup_system_tick(SYSTEM_TICK_FREQ);
        ID_Init();
        GetID();

        /* Init AHRS I2C */
        /* ----Initialize I2C slave mode--- */
        if(devNum == 0)
                I2C_MS_Master_Init();
        else
                I2C_MS_Slave_Init();

        TIDMstFirstInitFIN = 0;
        TIDMstInitFIN = 0;
        TIDMstStage = 0;
        TIDMstInitDevState = 1;
        FlashInit();
```

1. Set system clock
2. Set UART baud rate
3. Check ID
4. I^{2}C initialization according to ID
5. Initialization for Flash

6. Initialization for each sensor module

```
// Set function pointers for default sensor boards.
for(i=0; i<MAX_TID_DEV; i++)
{
      if (pTidList[i])
            SetDeviceFunction(pTidList[i], &(pfnDevFunc[i]));
}

// Call board init function
if (pTidList[devNum]->func.pfnSetup)
      pTidList[devNum]->func.pfnSetup();

// Load TID feature from flash
GetFlashTID(&(pTidList[devNum]->Feature), devNum);

// Set Timer1 to (1000/value) in (Hz)
Timer1Init(1000 / (pTidList[devNum]->Feature.data1.value));

/* TID initialize */
SlvDataInit();
MstDataInit();
}
```

7. Save module parameters into I²C variables

Figure 2-4 Setup() Function-Call Program

2.2.1 Initialization of Sensor Module

The following section introduces the sensor module initialization program content. Battery_init() code is used as an example. At the initializing program, the first part is the module initialize; the second part is the data settings transfer between master and slave. Following is the data setting format.

1. First, set the data needed. The photoresistor module's data is as follows.
   - **Sleep Period:** The periodic time to monitor the sensor module
   - **Trigger Alarm Value:** To trigger the alarm when the sensor value is above/below the value
   - **Sensor Value:** The current value of sensor module
   - **Alarm:** To turn on/off alarm

2. Status classification: Feature, Output and Input
   - **Feature:** Some settings of sensor module that don't need to transmit or receive data frequently.
   - **Output:** Master transmits the information to the slave sensor module.
   - **Input:** Master receives the information from the slave sensor module.

For example, a battery sensor module has two Feature parameters: “Sleep Period” to monitor the battery voltage periodically and “Trigger Alarm Value” to trigger alarm when the battery voltage is below this setting value. Moreover, the battery sensor module has two Input parameters: “Sensor Value” to show the current battery voltage value and “Alarm“ to show the alarm on/off status on the battery sensor module.

3. Set the parameters of each sensor module:

   - **Maximum, minimum:** Show in brown block (as the following figure), which is set by data range that is required.
   - **Position (arg):** Indicate the data position of the classification and the location should be increased sequentially.
   - **Length (datalen):** Represent the length of data and its unit is Byte.
   - **Value:** Represents the current setting value of data.
   - **Data number (dataNum):** Represent the number of data that will be transmitted or received to/from the master module. For example, there are two Input data for the sensor module and the dataNum should be set as 2.

```
void Battery_Init()
{
```

Initialization of sensor module

```
    SYS_UnlockReg();
    /* Enable EADC module clock */
    CLK_EnableModuleClock(EADC_MODULE);
    /* EADC clock source is 72MHz, set divider to 8, ADC clock is 72/8 MHz */
    CLK_SetModuleClock(EADC_MODULE, 0, CLK_CLKDIV0_EADC(8));
    SYS_LockReg();
    /* Configure the GPB0 - GPB3 ADC analog input pins.  */
    SYS->GPB_MFPL &= ~SYS_GPB_MFPL_PB1MFP_Msk;
    SYS->GPB_MFPL |= SYS_GPB_MFPL_PB1MFP_EADC_CH1;
    GPIO_DISABLE_DIGITAL_PATH(PB, BIT1);
    //LED
    SYS->GPA_MFPL &= ~(SYS_GPA_MFPL_PA2MFP_Msk);
    SYS->GPA_MFPL |= SYS_GPA_MFPL_PA2MFP_GPIO;
    GPIO_SetMode(PA,BIT2,GPIO_MODE_OUTPUT);
    PA2 = 1;

    /* Set the ADC internal sampling time, input mode as single-end and enable
the A/D converter */
    EADC_Open(EADC, EADC_CTL_DIFFEN_SINGLE_END);
    EADC_SetInternalSampleTime(EADC, 6);

    /* Configure the sample module 0 for analog input channel 1 and software
trigger source.*/
    EADC_ConfigSampleModule(EADC, 1, EADC_SOFTWARE_TRIGGER, 1);

    /* Clear the A/D ADINT0 interrupt flag for safe */
    EADC_CLR_INT_FLAG(EADC, 0x2);

    /* Enable the sample module 0 interrupt */
    EADC_ENABLE_INT(EADC, 0x2); //Enable sample module A/D ADINT0 interrupt
    EADC_ENABLE_SAMPLE_MODULE_INT(EADC, 1, 0x2); //Enable sample module 0
interrupt
```

Settings of sensor module will be passed to master module

```
    BatDev.DevDesc.DevDesc_leng = 26;       //Report descriptor
    BatDev.DevDesc.RptDesc_leng = 36;       //Report descriptor
    BatDev.DevDesc.InRptLeng = 5;           //Input report
    BatDev.DevDesc.OutRptLeng = 0;          //Output report
    BatDev.DevDesc.GetFeatLeng = 6;         //Get feature
    BatDev.DevDesc.SetFeatLeng = 6;         //Set feature
```

```
    BatDev.DevDesc.CID = 0;                    //manufacturers ID
    BatDev.DevDesc.DID = 0;                    //Product ID
    BatDev.DevDesc.PID = 0;                    //Device firmware revision
    BatDev.DevDesc.UID = 0;                    //Device Class type
    BatDev.DevDesc.UCID = 0;                   //reserve

    /* Feature */
    BatDev.Feature.data1.minimum = 0;          //Sleep period
    BatDev.Feature.data1.maximum = 1024;
    BatDev.Feature.data1.value = 100;
    BatDev.Feature.data2.minimum = 0;          //Battery alerm value
    BatDev.Feature.data2.maximum = 100;
    BatDev.Feature.data2.value = 50;
    BatDev.Feature.arg[0] = 1;
    BatDev.Feature.arg[1] = 2;
    BatDev.Feature.datalen[0] = 2;
    BatDev.Feature.datalen[1] = 2;
    BatDev.Feature.dataNum = 2;

    /* Input */
    BatDev.Input.data1.minimum = 0;            //Battery value
    BatDev.Input.data1.maximum = 100;
    BatDev.Input.data1.value = 100;
    BatDev.Input.data2.minimum = 0;            //Over flag
    BatDev.Input.data2.maximum = 1;
    BatDev.Input.data2.value = 0;
    BatDev.Input.arg[0] = 1;
    BatDev.Input.arg[1] = 2;
    BatDev.Input.datalen[0] = 2;
    BatDev.Input.datalen[1] = 1;
    BatDev.Input.dataNum = 2;

    /* Output */
    BatDev.Output.dataNum = 0;
}
```


Figure 2-5 Initialization Program of Battery Sensor Module

2.3 Function-Call Loop()

Figure 2-6 represents the flowchart of loop() function-call for the master module and slave sensor module. The loop() function-call is the main program code that executed in master or sensor module. The left flowchart is for master module and the right flowchart is for the slave sensor module.

The loop() function-call can be separated into two sections: Period section and non-periodic section.

- Period section works once at every setting time. For example, the vibration sensor module will update data every 0.1 second.
- Non-periodic section always is executed. Typically, it is used to check or react immediately on the sensor module. For example, the sonar sensor module collects continuously the distance information from the surrounding.

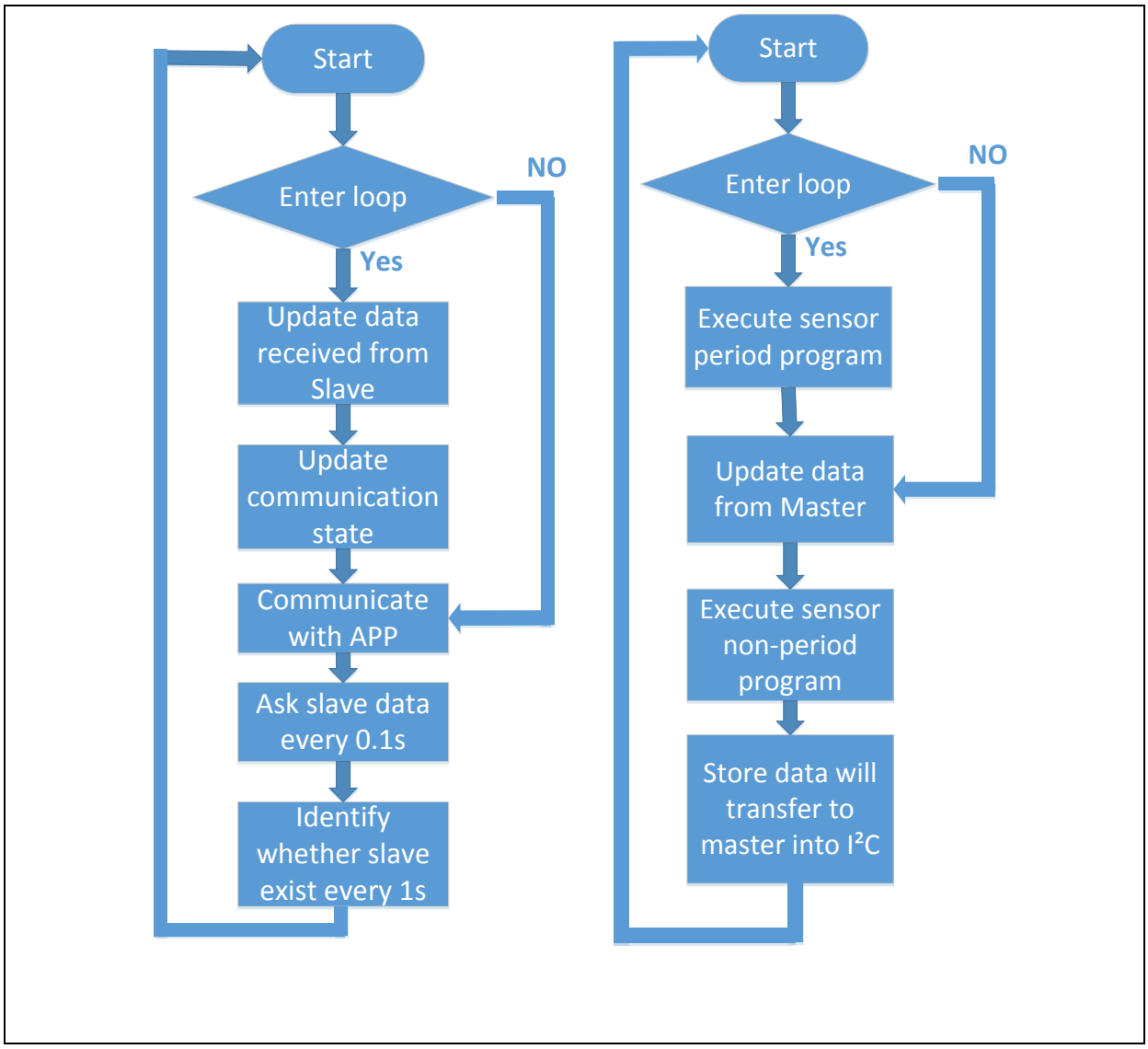

Figure 2-6 Loop() on Master and Sensor Module Flowchart

2.3.1 Loop() Overview

The following describes the loop() function-call. Figure 2-7 shows the whole program code of loop() function-call.

Period execution section (inside of if(TMR1INTCount != RecentTimeCounter) area):

1. Master module
    I. Communication with APP.
    II. Update parameter which will be transmitted to slave module.
    III. Update communication state with each sensor module (using I^2C).

    Master will communicate with each slave module periodically to get the newest parameter.
2. Sensor module:
    I. Execute the period program of sensor module.

Non-periodic execution section (outside of if(TMR1INTCount != RecentTimeCounter) area):

3. Master module
    I. Ask each sensor module by I^2C every 0.1s, including Set Feature, Get Feature, Set Output and Get Input.
    II. Check whether the sensor module is connected or disconnected every 1 second, including Device Descriptor and Report Descriptor receiving.
4. Sensor module:
    I. Update output signal from master.
    II. Executes non-periodic program.
    III. Store parameter which will be transmitted to master by I^2C.

```
void loop()
{
      CommandProcess();
      TaskScheduler();
      PowerControl();

      // ******************************************************
      //                Get parameter for once every 0.1s
      // ******************************************************
      if(TMR1INTCount != RecentTimeCounter)
      {
            MainTimeMSCounter = getTickCount();

            if (pTidList[devNum]->func.pfnPeriod)
                  pTidList[devNum]->func.pfnPeriod();

            RecentTimeCounter = TMR1INTCount;
      }

      // =============TID Master handle received parameter===============
      // -----------TID Master handle received parameter------------
      if(devNum == 0)
      {
            if(TIDMstInitFIN==0)
            {
                  TIDMst_GetDev();
            }
            else if(TIDMstInitFIN==1)
            {
                  TIDMstDevTRx();
            }
      }
      // ==================== SLAVE PART  =====================
      // -----------TID slave handle received parameter--------------
      else
      {
            if(I2CMS_SlvRxFin==1)
                  TID_SlvRxUpdate();
      }
```

1 - Ⅰ : Communication with APP

1- Ⅱ : Update parameter which will be transmitted to slave

1 - Ⅲ : Update communication state with each sensor module

2 - Ⅰ : Periodic program of sensor module

3 - Ⅰ : Master asks each sensor for parameter by I^2C every 0.1s

3 - Ⅱ : Master identifies whether sensor module is present or absent every 1 second

4 - Ⅰ : Sensor updates the received parameter from master

```
        // -----------Slave part execute subfunction--------------

        if (pTidList[devNum]->func.pfnPulling)
              pTidList[devNum]->func.pfnPulling();

        // --------------TID Slave Store parameter-----------------
        if(devNum != 0)
              SlvDataStore();
}
```

4 - Ⅱ: Sensor module executes non-periodic function

4 - Ⅲ: Sensor module stores parameter which will be transmitted to master into variables for I^2C

Figure 2-7 Loop() Program Description

2.3.2 Detailed Description of Loop()

The NuMaker Brick platform supports two stages to meet the needs of different conditions. The first stage is the function program executed periodically for sensor modules as shown in Figure 2-8.

```
// ******************************************************
//                Get parameter for once every 0.1s
// ******************************************************
if(TMR1INTCount != RecentTimeCounter)
{
      MainTimeMSCounter = getTickCount();

      if (pTidList[devNum]->func.pfnPeriod)
            pTidList[devNum]->func.pfnPeriod();

      RecentTimeCounter = TMR1INTCount;
}
```

2 - I : Periodic program of sensor module

Figure 2-8 Periodic Program of Sensor Module

The sensing program executed periodically shown in Figure 2-8 must correspond with the periodic function of a master or sensor module, as shown in Figure 2-9.

```
APFN_FUNC_T pfnDevFunc[MAX_TID_DEV] =
{
      ...
      /* for embedded device boards */
```

Sensor functions executed periodically

```
      {Battery_Init,    MasterControl,   NULL,              report_battery},
      {Buzzer_Init,     NULL,            Buzzer_Control,    report_buzzer},
      {Led_Init,        NULL,            Led_Control,       report_led},
      {AHRS_Init,       AHRS_Control,    AHRS_Cal,          report_ahrs},
      {SonarInit,       SonarDetect,     SonarTimeOutCheck, report_sonar},
      {HTU21D_Init,     WaitHTU21D,      GetHTU21DTemp,     report_temp},
      {Gas_Init,        GetGas,          NULL,              report_gas},
      {IR_Init,         IR_Control,      IR_Check,          report_ir},
      {key_init,        NULL,            NULL,              report_key},

      ...
};
```

Figure 2-9 Structural Function Table of Sensor Modules

The following description uses the vibration sensor module as an example.

```
void AHRS_Control(void)
{
	SensorsRead(SENSOR_ACC|SENSOR_GYRO,1);
	nvtUpdateAHRS(SENSOR_ACC|SENSOR_GYRO);
	AhrsRead(AHRSDev.Feature.data2.value, AHRSDev.Feature.data3.value,
AHRSDev.Output.data1.value); //(Vibration Level, alerm time, clear flag)
}
```

Figure 2-10 Non-Periodic Program of Sensor Module

In the second stage, the function program will be checked or executed continuously and this program section is outside of the if(TMR1INTCount != RecentTimeCounter){} block.

```
	// -----------Slave part execute subfunction--------------

	if (pTidList[devNum]->func.pfnPulling)
		pTidList[devNum]->func.pfnPulling();
```

4 - Ⅱ : Sensor module executes non-periodic function

Figure 2-11 Non-Periodic Program of Sensor Module

The sensing program executed non-periodically shown in Figure 2-11 must correspond with the non-periodic function of a master or sensor module as shown in Figure 2-12.

```
APFN_FUNC_T pfnDevFunc[MAX_TID_DEV] =
{
	...
	/* for embedded device boards */
```

Sensor functions executed immediately

```
	{Battery_Init,	MasterControl,	NULL,		report_battery},
	{Buzzer_Init,	NULL,		Buzzer_Control,	report_buzzer},
	{Led_Init,	NULL,		Led_Control,	report_led},
	{AHRS_Init,	AHRS_Control,	AHRS_Cal,	report_ahrs},
	{SonarInit,	SonarDetect,	SonarTimeOutCheck,	report_sonar},
	{HTU21D_Init,	WaitHTU21D,	GetHTU21DTemp,	report_temp},
	{Gas_Init,	GetGas,		NULL,		report_gas},
	{IR_Init,	IR_Control,	IR_Check,	report_ir},
	{key_init,	NULL,		NULL,		report_key},

	...
};
```

Figure 2-12 Structural Function Table of Sensor Modules

After setting the sensor module is finished, set the master, including setting trigger source from a new module. The Master Control program can be found in pfnDevFunc which is located at main.c. TIDMstUpdateDevState is in Master Control. Refer to Figure 2-13.

```
void MasterControl(void)
{
	/* Master Store parameter */
	TIDMstUpdateDevState();
	/* Master recheck device */
	if(TIDMstFirstInitFIN==1)
	{
		if(TMR1TimerCounter > 10)
		{
			if(I2CMstEndFlag==1)
			{
				TIDMstInitFIN=0;
				TMR1TimerCounter=0;
				I2C_Close(I2C_MS_PORT);
				I2C_MS_Master_Restart();
			}
		}
		else
		{
			TIDMstInitFIN = 1;
			TMR1TimerCounter++;
		}
	}
	GetBattery();
}
```

Figure 2-13 The Feedback Setting from Master module to sensor modules

The example of TIDMstUpdateDevState() is shown below:

```
// *****************************************************
//     Master Handle Device state
// *****************************************************
void TIDMstUpdateDevState()
{
	...
                The received parameter from vibration module is 1
	if((AHRSDev.Input.data2.value==1) && (TIDMstInitDevState & (1<<AHRSDEV)))
```

```
        {
            if(BuzDev.dTod.dTHR==1)
            {
                BuzDev.Output.data1.value = 1;   //Song
                TIDMstDevState[BUZZERDEV] |= 2;  //buzzer operate: output(2)
            }

        ...
}
```

There is relationship between buzzer and vibration module

Master ask buzzer to beep

Figure 2-14 Sensor Module Trigger settings

Taking the vibration module as an example, this program uses the following function:

If the received parameter from vibration module is 1: if(AHRSDev.Input.data2.value==1)

If there is relationship between the buzzer and vibration module: if (BuzDev.dTod.dTHR==1)

Master asks buzzer to beep: BuzDev.Output.data1.value = 1

2.4 Program File

The NuMaker Brick function can be divided into two parts:

- **Master function:** Includes the NuMaker Brick platform architecture and communication protocol with APP. Refer to Table 2-1.
- **Slave function:** The Sensor module program. Refer to Table 2-2.

Master Module	Program Name	Program Description
I²C	i2c_ms	I²C communication setting on the platform
	i2cdev	I²C communication function-call
Platform parameter	tid	Platform parameter setting
	tidmst	Master platform parameter storage methods
	tiddev	Slave platform parameter storage methods
ID check	devCheck	ID check in each module
APP communication	report	APP communication setting
	report_AP	APP communication function-call

Table 2-1 Programs of Master Module

Sensor Module	Program Name	Program Description
Battery	battery	Use ADC detect battery
LED	led	Use PWM driving LED
Buzzer	buzzer	Use PWM driving buzzer
	music	Music mapping table
Gas	gas	Use ADC detect gas
IR	Ir	Use PWM driving IR and using capture to detect IR
	PWM0P2	IR's PWM setting and interrupt executing
Sonar	sonar	Use PWM capture get sonar signal
Temperature	temp_HTU21D	Use I²C to receive temperature and humidity signal
	i2c_HTU21D	I²C communication setting on the temperature module
Vibration	AHRS	Use I²C to receive vibration signal
	calibrate	Calibrate vibration module
	mpu6050	I²C communication setting on the vibration module
	sensors	Arrange vibration module parameter

Button	KEY	Button module function
	GPC_IRQ	Use PC interrupt to receive button state
	GPE_IRQ	Use PE interrupt to receive button state
Period Setting	timerIRQ1	Set the execution period at each module

Table 2-2 Programs of Sensor Module

3 COMMUNICATION LAYER OF NUMAKER BRICK PLATFORM

This chapter introduces the whole data transmission throughout the expansion platform. The first section will explain how to reach to the two targets that mentioned previously on this platform.

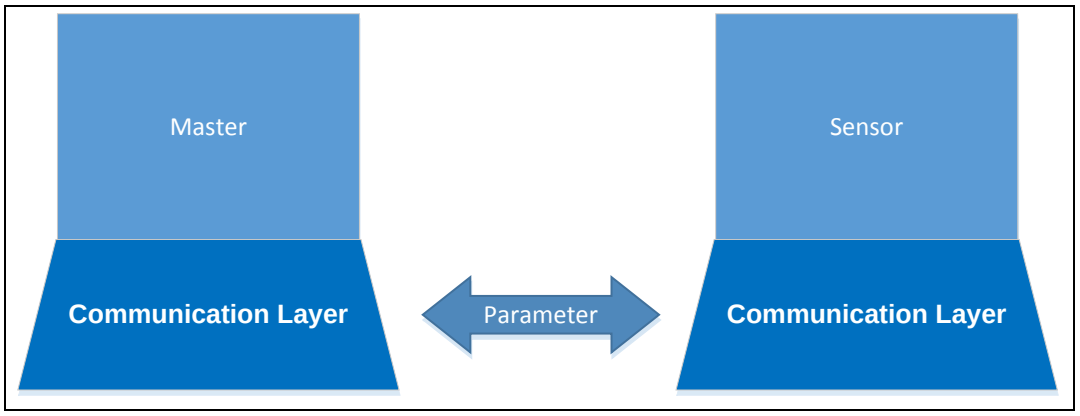

Figure 3-1 Communication Architechture between Master and Sensor Module

The NuMaker Brick platform can be divided into three parts:

- **Master Module:** Arrange the data which is received from other module or send parameter to other module.
- **Sensor Module:** To monitor or detect any changes in the surrounding and transmit these parameter to master module.
- **Communication Layer:** Communication layer dispersed in Master and sensor module. It is established to accomplish two objectives mentioned previously – device scalability and parameter consistency. Therefore, the parameter must be standardized that common market sensor module have. Let Master can gather information and send command to sensor module.

In the platform, the master collects information from sensor module, and sends command to other device. It is important to let master know sensor module parameter.

- **Parameter classification:** Set consistent format for each sensor parameter (including maximum value, minimum value and the current value).
- **Module expansibility:** Separate each sensor information into three classes (Feature, Input and Output). It also records the length and position of each parameter on each class.

The second section will describe the protocol between master module and sensor modules. The last section will explain their parameter formats of sensor modules.

The communication between master and sensor module can be processed by I^2C which is defined as communication layer. Each parameter defined in the Communication layer for all sensor modules is shown in Table 3-1.

	Parameter Class	Parameter Content
Sensor module	Feature	Maximum
		Minimum

		Length
		order
		value
	Output	Maximum
		Minimum
		Length
		order
		value
	Input	Maximum
		Minimum
		Length
		order
		value

Table 3-1 Parameter Specification of Sensor Module

Each parameter has the following information:

- Parameter range: The maximum and minimum values of parameter.
- Parameter length: The length of parameter.
- Parameter order: The position of parameter in the classification.
- Parameter content: The current Parameter Value.

The communication level also classifies all parameter for all sensor modules to avoid the unnecessary transmission of parameter during the communication. There are three types of classes as described below.

- **Feature:** The specific information of sensor module, e.g. the frequency of buzzer, the distance for sonar to alarm.
- **Input:** The information transmitted from sensor module to master module, e.g. the real distance of sonar, the vibration event.
- **Output:** The command transmitted from master module to sensor module, e.g. LED blink command.

These classes are all the same on both master module and sensor module. The Input and Output definition are from the view of master module.

3.1 Communication Protocol of NuMaker Brick Platform

The communication protocol and flow can be classified into three stages as shown in Figure 3-2.

- **Sensor module type:** Sensor module will send “Device Descriptor” back to master module in this stage.
- **Sensor module parameter:** Sensor module will send “Report Descriptor” back to master module in this stage.
- **Parameter transaction:** The master module and sensor modules can transact parameter to each other, and can be defined as four kinds of types: “Get Input”, “Set Output”, “Get Feature” and “Set Feature”.

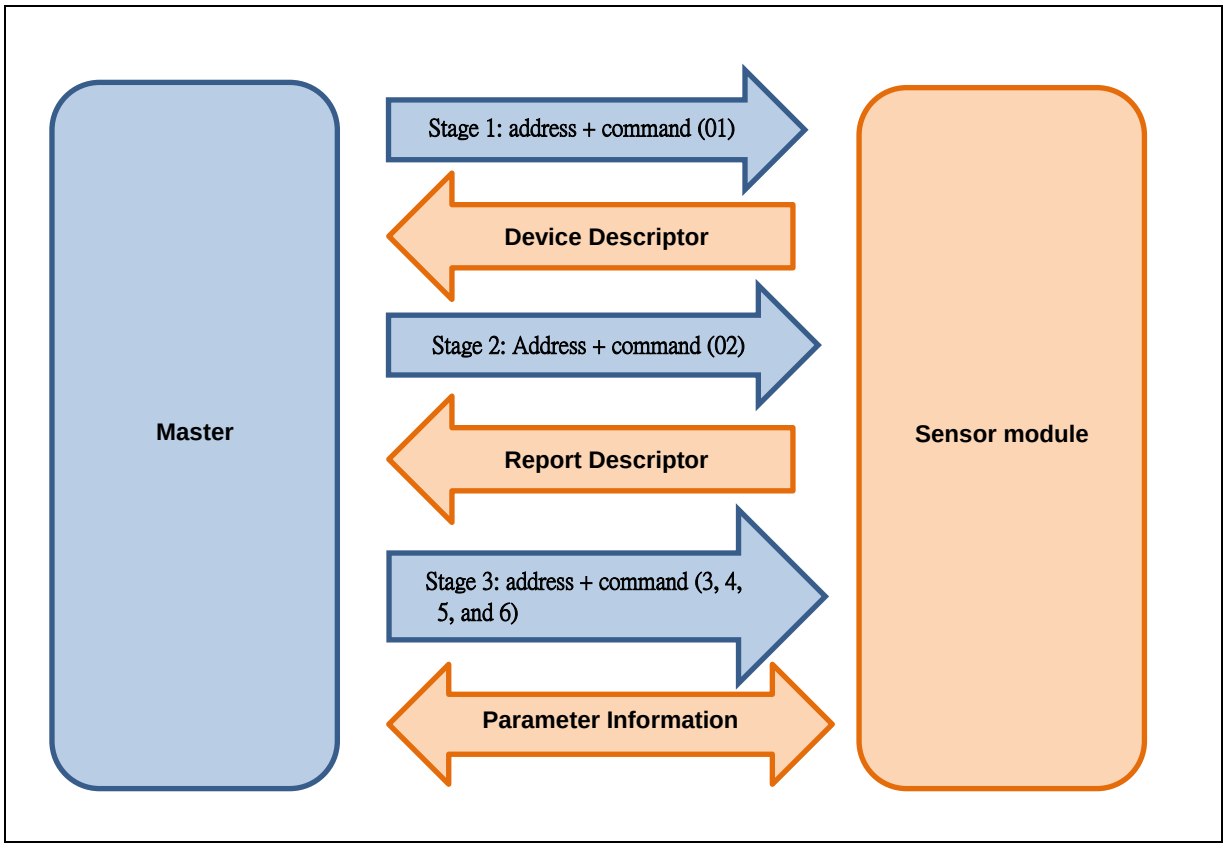

Figure 3-2 Communication Protocol between Master and Sensor Module

At the beginning of system operation, the master should identify the sensor module before it can transmit or receive parameter to/from the sensor modules. The commands and transaction flows that master communicates with sensor module are explained and shown below.

1. Master identifies the sensor module: Call the sensor module by I^2C, if master receives any responses from the sensor module that it indicates the sensor module is connected.

2. Master asks the sensor module to report its “Device Descriptor” (Stage 1). Ex. parameter length and ID……

Device Descriptor: (command 0x01)

Start	Address	W	ACK	Command (0x01)	ACK	Command (0x00)	ACK	Sr
Address	R	ACK	TID Descriptor	ACK	· · ·	TID Descriptor	NACK	

Note: Also see Table 3-2.

3. Master asks the sensor module respond to “Report Descriptor” (Stage 2).

Report Descriptor: (command 0x02)

Start	Address	W	ACK	Command (0x02)	ACK	Command (0x00)	ACK	Sr
Address	R	ACK	Report Descriptor	ACK	· · ·	Report Descriptor	NACK	

Note: Also see Table 3-3.

4. Master asks the sensor module to report its parameters (Stage 3). There are four kinds of types: “Get Input”, “Set Output”, “Get Feature” and “Set Feature”.

Get Input: (command 0x03)

Start	Address	W	ACK	Command (0x03)	ACK	Command (0x00)	ACK	Sr
	Address	R	Data (Length Field, LSB)	ACK	Data (Length Field, MSB)	ACK		
			Data (Report Field, LSB)	ACK	· · ·	Data (Report Field, MSB)	NACK	

Note: Also see Table 3-4.

Set Output: (command 0x04)

Start	Address	W	ACK	Command (0x04)	ACK	Command (0x00)	ACK
Data (Length Field, LSB)	ACK	Data (Length Field, MSB)	ACK				
Data (Report Field, LSB)	ACK	· · ·	Data (Report Field, MSB)	ACK	P		

Note: Also see Table 3-4.

Get Feature: (command 0x05)

Start	Address	W	ACK	Command (0x05)	ACK	Command (0x00)	ACK	Sr
	Address	R	Data (Length Field, LSB)	ACK	Data (Length Field, MSB)	ACK		
			Data (Report Field, LSB)	ACK	· · ·	Data (Report Field, MSB)	NACK	

Note: Also see Table 3-4.

Set Feature: (command 0x06)

Start	Address	W	ACK	Command (0x06)	ACK	Command (0x00)	ACK
Data (Length Field, LSB)	ACK	Data (Length Field, MSB)	ACK				
Data (Report Field, LSB)	ACK	· · ·	Data (Report Field, MSB)	ACK	P		

Note: Also see Table 3-4.

User can randomly set the length and range of parameter, giving the maximum flexibility for all users.

3.2 Parameter Format

The Master module plays the most important role in the whole NuMaker Brick platform. Master should identify all the sensor modules on the same I^2C bus before transacting parameters with sensor modules. After recognizing all the sensor modules, master will receive parameters from sensor modules or transmit parameter to sensor modules through the I^2C bus.

3.2.1 Device Descriptor

As shown in Figure 3-2, in the Stage 1, the master asks the sensor module to pass back the Device Descriptor, and the parameter length of each parameter is 2 bytes in this device descriptor. Master can get the related parameters, for example, the company ID, device ID, product ID, and the hardware version to know the sensor module from the reporting of Device Descriptor. Table 3-2 shows the complete format of Device Descriptor.

Data Name	Data Content	Data Length	Function Description
Start flag	257 (1, 1)	2 bytes	
Device Descriptor Length	26	2 bytes	First stage (Device Descriptor) data length
Report Descriptor Length	Variable	2 bytes	Second stage (Report Descriptor) data length
Get Feature Length	Variable	2 bytes	Third stage (Get Feature) data length
Set Feature Length	Variable	2 bytes	Third stage (Set Feature) data length
Get Input Length	Variable	2 bytes	Third stage (Get Input) data length
Set Output Length	Variable	2 bytes	Third stage (Set Output) data length
Company Code	Variable	2 bytes	Company code which produce sensor module
Device Code	Variable	2 bytes	Sensor module's Device Code
Product Code	Variable	2 bytes	Sensor module's Product Code
Hardware Version	Variable	2 bytes	Sensor module's Hardware Version
Device Class Code	Variable	2 bytes	Sensor module's Device Class Code
Reserved	Variable	2 bytes	
Reserved	Variable	2 bytes	

Table 3-2 Device Descriptor Format

3.2.2 Report Descriptor

As shown in Figure 3-2, in the Stage 2, the master asks the sensor module to report the Report Descriptor. After confirming each length of class in the sensor module, users can start getting the length of every data. In this stage, transfer whole data length first. Second, transfer feature data. In feature data, transfer 257 represent following data are feature data and then start to transfer single data. The first single data to transfer is address information. (e.g. 1st data:5, 2nd data:17, 3rd data:29, 4th data:41, 5th data:53, 6th data:65, 7th data:77, 8th data:89, 9th data:101, 10th data:113,). Then data length (usually 1~2 bits) is transferred. At last, maximum (8+1 or 8+2) and minimum (10+1 or 10+2) are transferred.

Data Name	Data Content	Data Length	Function Description
Report Descriptor Length	variable		The total data length of Report Descriptor
Parameter Class (Feature)	257 (1, 1)	2 bytes	Declare the following parameter type: Feature
Parameter Address	5	1 byte	Declare this parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare this parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum parameter value (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum parameter value (12) and the next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
Parameter Address	17	1 byte	Declare this parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare the parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum parameter value (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum parameter value (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
…	…	…	…
Parameter Class (Input)	258 (1, 2)		Declare the following parameter type: Input
Parameter Address	5	1 byte	Declare this parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare this parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum parameter value (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum parameter value (12) and next column length (1 or 2)

Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
Parameter Address	17	1 byte	Declare this parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare this parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum parameter value (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum parameter value (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
…	…	…	…
Parameter Class (Output)	259 (1, 3)		Declare the following information type: Output
Parameter Address	5	1 byte	Declare this parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare this parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum parameter value (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum parameter value (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
Parameter Address	17	1 byte	Declare this parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare the parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum parameter value (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum parameter value (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
…	…	…	…

Table 3-3 Report Descriptor Format

At this time, the master must save the information in the following order:

After receiving the parameter address, the length of parameter will be stored at corresponding datalen and the arg parameter also will be updated by the current value. For example, when receiving address is 29 and length is 2, write 2 to the third datalen and 3 to the third arg.

3.2.3 Parameter Information

After master receives the Report Descriptor done from the sensor module, users can get or set the content values of parameters. There are four kinds of read or write options that correspond to different parameter types (Feature, Input and Output).

- Write sensor module Output parameter
- Read sensor module Input parameter
- Write sensor module Feature parameter
- Read sensor module Feature parameter

Data Name	Data Content	Data Length	Function Description
Get Feature Length **(Set Feature Length)** **(Get Input Length)** **(Set Output Length)**	Variable	2 bytes	The total length of Data information
First Parameter Value	Variable	1~2 bytes	The first parameter in the classification
Second Parameter Value	Variable	1~2 bytes	The second parameter in the classification
...	...	...	...

Table 3-4 Parameter Format

The parameter is determined according to the arg parameter at this stage. If the arg is not zero, it indicates that the parameter is meaningful. If the arg parameter is zero, it indicates that this parameter can be ignored.

4 ADDING A NEW SENSOR MODULE

The procedure of adding a new sensor module can be separated into two stages: hardware stage and software stage. On the hardware stage, it is supposed that user has a workable sensor, for example, the photoresistor sensor. Figure 4-1 shows how to add the hardware circuit about the photoresistor sensor on the expansion module board. Section 4.2 will guide user how to modify the firmware program for this new photo resistor sensor module.

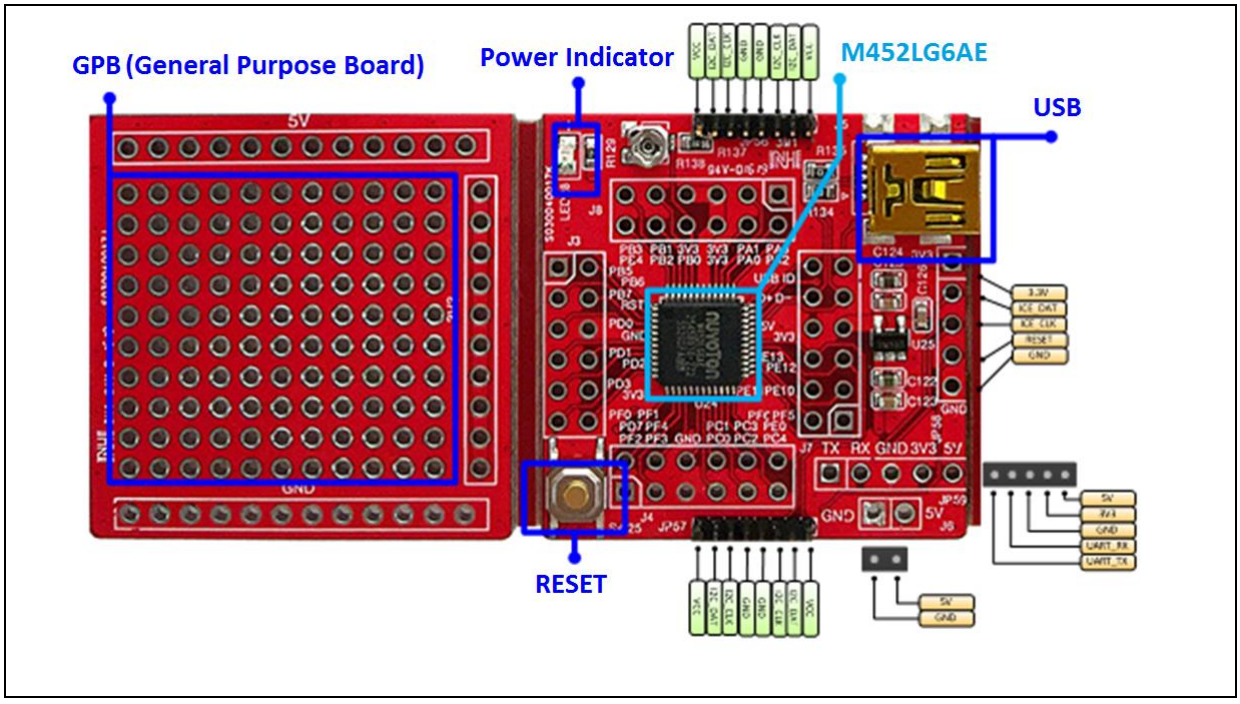

Figure 4-1 Expansion Module Board

4.1 Hardware Stage

The hardware stage includes two steps: setting ID for new sensor module and adding new hardware circuit on expansion module board. Following is an example of adding a photoresistor sensor module.

4.1.1 Setting ID for New Sensor Module

This chapter migrates the workable photoresistor sensor circuit to the expansion module board. On this expansion module board, it provides many through holes (vias) and four 2x6 connectors that one by one mapped to all pins of MCU to expand the GPIO functions of pins and to solder conveniently the extra hardware circuits.

Figure 4-2 shows the partial schematics of MCU and ID checking of this expansion module board. The MCU M452LG6AE chip is one of the M452 series microcontrollers and this chip has very rich peripherals, for example, UART, Smart Card, SPI, I^2C, I^2S, USB, PWM, ACMP, DAC and ADC.

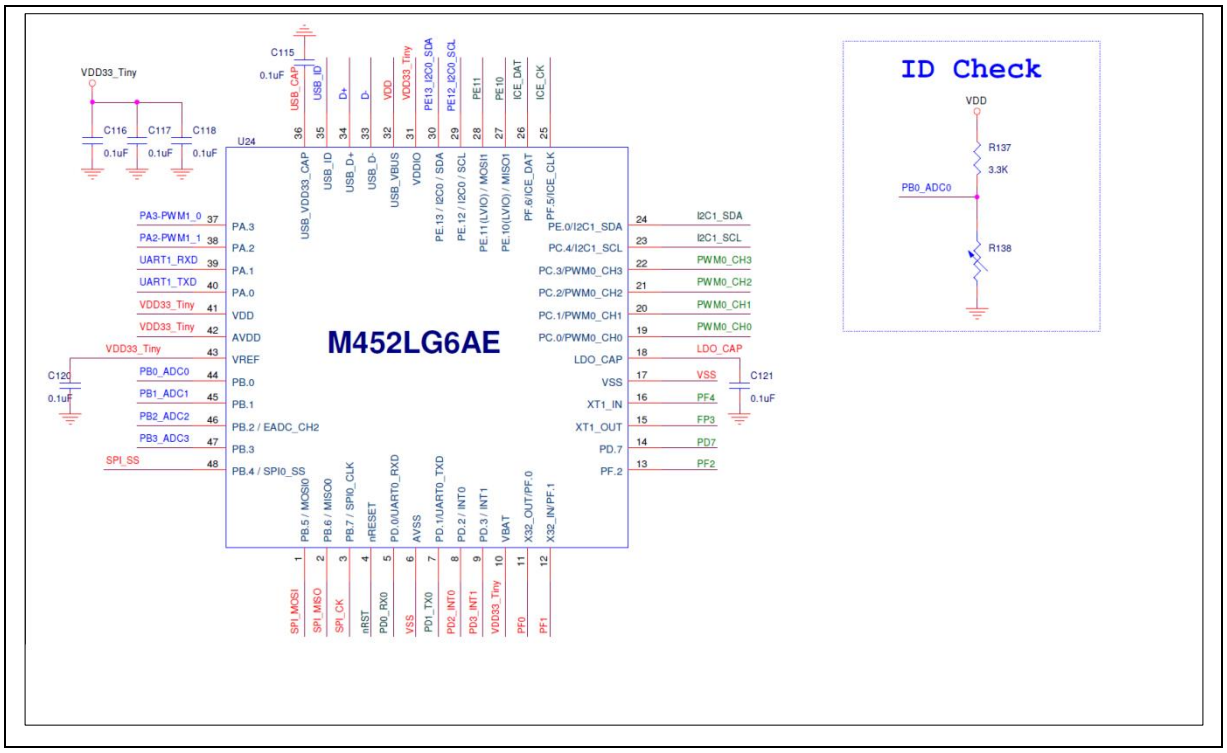

Figure 4-2 Partial Schematics of MCU and ID Check of Expansion Module Board

Before adding a new sensor module into the working NuMaker Brick platform, it is very important for user to define the module's ID for this new sensor module. The resistor R138 for the ID check as shown in Figure 4-2 is a variable resistor for user to change the ADC input voltage on the PB.0 pin of M452LG6AE MCU to define the ID value for this new sensor module that created by the expansion module board. In this example of photoresistor sensor module, the resistance value of R138 should be adjusted to 5.1KΩ, the 9th sensor module (ID = 9), to distinguish with the other sensor modules (ID = 0~8) that had been pre-defined in the NuMaker Brick. Table 4-1 shows the ID values from 9 to 14 and their corresponding resistance values and ADC voltages that input to PB.0 pin of MCU M452LG6AE if user has the other new sensor modules that will be added in the NuMaker Brick platform.

ID	Target Voltage Range	Resistance Value Connected to V_{DD}	Resistance Value Connected to GND	Voltage Value
9	1.9V ~ 2.05V	3.3KΩ	5.1KΩ	1.99V
10	2.1V ~ 2.25V	3.3KΩ	6.8KΩ	2.2V
11	2.3V ~ 2.45V	3.3KΩ	9.1KΩ	2.41V
12	2.5V ~ 2.65V	3.3KΩ	12KΩ	2.59V
13	2.7V ~ 2.85V	3.3KΩ	18KΩ	2.8V
14	2.9V ~ 3.05V	3.3KΩ	33KΩ	3V

Table 4-1 ID Identifying by Different Resistance and ADC Voltage for New Sensor Module

4.1.2 Adding New Hardware Circuit on Expansion Module Board

After setting the resistance value of R138 for the ID check, user can add additional photoresistor sensor circuit on the expansion module board, and then connect the power required by the module and the pin that needs to be connected to the MCU to the appropriate connector. Figure 4-3 shows the complete schematic diagram of the new photoresistor sensor module.

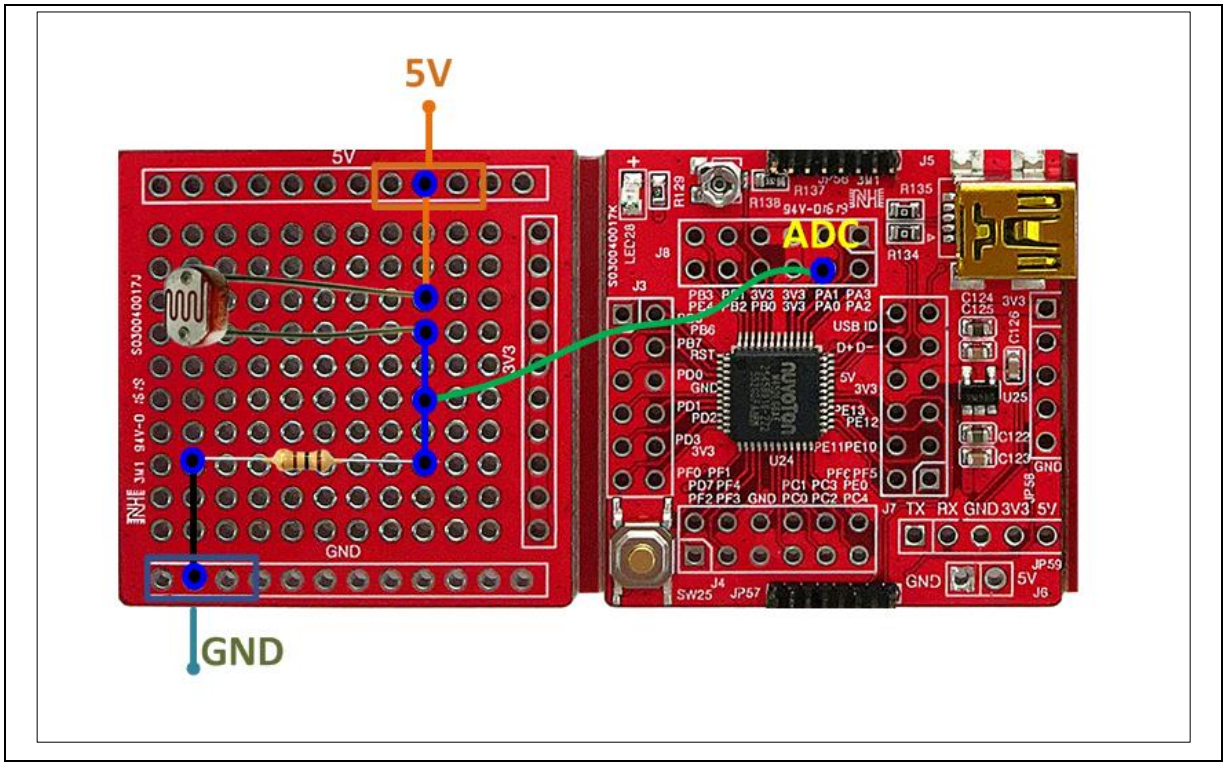

Figure 4-3 Complete Schematic Diagram of New Photoresistor Sensor Module

4.2 Software Stage

The software stage can be separated into two sections, program initialization and program execution. Table 4-2 shows that software should be modified or added onto both master module and new sensor module.

At program initialization, user doesn't need to modify any programs for the master module, but for the new sensor module user should first identify the ID, and then do hardware initialization for this new sensor module and define the related parameters for the NuMaker Brick platform.

At program execution, user should add the response program into the master module for the new sensor module due to this sensor module joining and add the main execution program into the new sensor module.

Software Section	Master Module	New Sensor Module
Initialization	It's not necessary to modify	1. ID checking 2. Hardware initialization of new sensor module 3. Parameters Initialization for the platform
Execution	To add the response program for the new sensor module	1. To add the execution program of new sensor module 2. To combine the program of platform into the program of new sensor module

Table 4-2 Software Modification on Master Module and New Sensor Module

Figure 4-4 shows the function-calls of new sensor modules in modular programming table. User just only enters the correct function-call names into this modular programming table. For the new photoresistor sensor module, user only fills the initialization function-call "Lr_Init" and the periodic execution function-call "GetLr" as shown in Figure 4-4.

```
APFN_FUNC_T pfnDevFunc[MAX_TID_DEV] =
{
      /*
         Functions for each device:

          {Initial, Period, Pulling, Report}

          Initial : Init and configure device
          Period  : Process device sensor parameter or set sensor once per 0.1
seconds.
          Polling : Process device sensor parameter or set sensor frequently.
          Report  : Report process parameter to master
      */

      /* for embedded device boards */
```

```
        {Battery_Init,      MasterControl,      NULL,               report_battery},
        {Buzzer_Init,       NULL,               Buzzer_Control,     report_buzzer},
        {Led_Init,          NULL,               Led_Control,        report_led},
        {AHRS_Init,         AHRS_Control,       AHRS_Cal,           report_ahrs},
        {SonarInit,         SonarDetect,        SonarTimeOutCheck,  report_sonar},
        {HTU21D_Init,       WaitHTU21D,         GetHTU21DTemp,      report_temp},
        {Gas_Init,          GetGas,             NULL,               report_gas},
        {IR_Init,           IR_Control,         IR_Check,           report_ir},
        {key_init,          NULL,               NULL,               report_key},

        /* for custom device boards
           fill your four board functions here to specified ID. */
        {Lr_Init,   GetLr,  NULL,  report_resDev9},    // For device ID 9
        {NULL,      NULL,   NULL,  report_resDev10},   // For device ID 10
        {NULL,      NULL,   NULL,  report_resDev11},   // For device ID 11
        {NULL,      NULL,   NULL,  report_resDev12},   // For device ID 12
        {NULL,      NULL,   NULL,  report_resDev13},   // For device ID 13
        {NULL,      NULL,   NULL,  report_resDev14}    // For device ID 14
};
```

Sensor module's initialization

Sensor function executed periodically

Sensor function executed immediately

Figure 4-4 Function-Calls of New Sensor Module in Modular Programming Table

4.2.1 Program Initialization Section

Figure 4-5 shows the program initialization of photoresistor sensor module, the Lr_init() function-call. The program initialization includes ADC hardware initialization based on the new photoresistor sensor module and setting the related parameters including Device Descriptor and Report Descriptor, which will be transacted between master module and this new sensor module.

The Report Descriptor includes three types of classified parameters about this new photoresistor sensor module: Feature, Input and Output classes.

There are four parameters about this new photoresistor sensor module that should be transacted with master module, the Sleeping Period, Alarm Trigger Value, Light Value and Alarm Flag that are put in the Feature and Input classes individually.

For each parameter, user also needs to define the maximum value, minimum value, content of parameter, position of parameter, length of parameter and number of parameter for each parameter in individual class.

- **maximum and minimum:** set by the required parameter.
- **value:** the current Parameter Value.
- **arg:** the position of parameter, increasing by one.
- **datalen:** the length of parameter, the unit is byte.
- **dataNum:** the total number of parameter in this class.

```
void Lr_Init()
{
```

Initialization of sensor module

```
    SYS_UnlockReg();
    /* Enable EADC module clock */
    CLK_EnableModuleClock(EADC_MODULE);
    /* EADC clock source is 72MHz, set divider to 8, ADC clock is 72/8 MHz */
    CLK_SetModuleClock(EADC_MODULE, 0, CLK_CLKDIV0_EADC(8));
    SYS_LockReg();
    /* Configure the GPB0 - GPB3 ADC analog input pins */
    SYS->GPB_MFPL &= ~SYS_GPB_MFPL_PB1MFP_Msk;
    SYS->GPB_MFPL |= SYS_GPB_MFPL_PB1MFP_EADC_CH1;

    GPIO_DISABLE_DIGITAL_PATH(PB, BIT1);

    /* Set the ADC internal sampling time, input mode as single-end and enable the A/D converter */
    EADC_Open(EADC, EADC_CTL_DIFFEN_SINGLE_END);
    EADC_SetInternalSampleTime(EADC, 6);

    /* Configure the sample module 0 for analog input channel 1 and software trigger source.*/
    EADC_ConfigSampleModule(EADC, 1, EADC_SOFTWARE_TRIGGER, 1);

    /* Clear the A/D ADINT0 interrupt flag for safe */
    EADC_CLR_INT_FLAG(EADC, 0x2);

    /* Enable the sample module 0 interrupt.  */
    EADC_ENABLE_INT(EADC, 0x2);//Enable sample module A/D ADINT0 interrupt
    EADC_ENABLE_SAMPLE_MODULE_INT(EADC, 1, 0x2);//Enable sample module 0 interrupt.
```

Settings of sensor module will be passed to master module

Device Descriptor

```
    ResDev9.DevDesc.DevDesc_leng = 26;      //Report descriptor
    ResDev9.DevDesc.RptDesc_leng = 36;      //Report descriptor
    ResDev9.DevDesc.InRptLeng = 5;          //Input report
    ResDev9.DevDesc.OutRptLeng = 0;         //Output report
    ResDev9.DevDesc.GetFeatLeng = 6;        //Get feature
    ResDev9.DevDesc.SetFeatLeng = 6;        //Set feature
    ResDev9.DevDesc.CID = 0;                //manufacturers ID
    ResDev9.DevDesc.DID = 0;                //Product ID
    ResDev9.DevDesc.PID = 0;                //Device firmware revision
    ResDev9.DevDesc.UID = 0;                //Device Class type
    ResDev9.DevDesc.UCID = 0;               //reserve
```

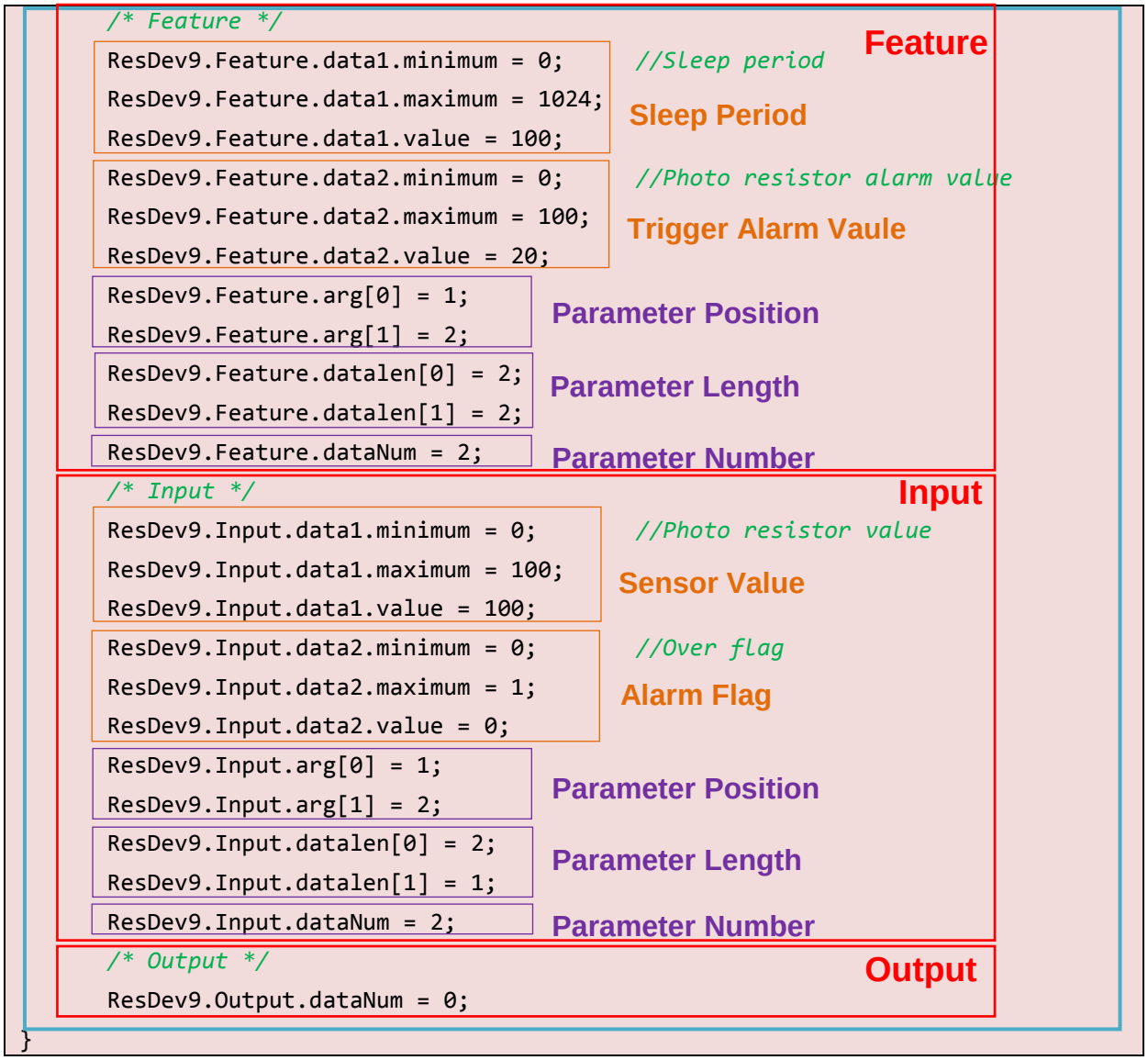

Figure 4-5 Photoresistor Sensing Module Initialization Program

At this time, a new sensor module initialization is completed. Then, proceed to the execution phase.

4.2.2 Program Execution Section

The execution stage can be divided into two parts:

- Sensor module modification:

  Add original program into NuMaker Brick platform

- Master program modification:

  The modification of the master is to communicate with new modules. The communication includes setting the objects that can be triggered by a new device, and the object that can trigger the new device. At TIDMstUpdateDevState(), which is in tismst.c .

The functions of the photoresistor module used in this program are described as follows:

If the parameter that photoresistor module sends back is 1;

1. If a buzzer and the photoresistor module are related to each other;
2. Master asks the buzzer starts to beep;
3. If the LED and photoresistor module are related to each other;
4. Master asks the LED to start to blink.

```
// *****************************************************
//     Master Handle Device state
// *****************************************************
void TIDMstUpdateDevState()
{
       ...
                    If the parameter that photoresistor module return is 1 11 1
       else if((ResDev9.Input.data2.value == 1) && (TIDMstInitDevState &
(1<<RESDEV9)))
       {    BuzDev.dTod.dNINE represent that buzzer and photoresistor module have relationship 11
            if(BuzDev.dTod.dNINE==1)
            {          Master asks buzzer to start to beep
                  BuzDev.Output.data1.value = 1;   //Song
                  TIDMstDevState[BUZZERDEV] |= 2;  //buzzer operate:
                  output(2)
            }LedDev.dTod.dNINE represent that LED and photoresistor module have relationship 11 1
            if(LedDev.dTod.dNINE==1)
            {          Master asks LED to start to beep
                  LedDev.Output.data1.value = 1;   //Period
                  TIDMstDevState[LEDDEV] |= 2;     //led operate:
                  output(2)
            }
       }


      ...
}
```

Figure 4-6 Structural Function Table of Sensor Modules

5 NUMAKER BRICK CONNECTED WITH ARDUINO

User can also use Arduino as a new sensor module to connect with the NuMaker Brick. The procedure can be separated into two steps as described in the following sections. The first section will discuss the necessary programs that all sensor modules should execute. The second section will describe how to add the new Arduino module into NuMaker Brick platform.

5.1 Necessary Function-calls for All Sensor Modules

This section introduces the necessary program function-calls of setup() and loop() that all sensor modules in the NuMaker Brick platform should execute.

5.1.1 Function-Call Setup()

As shown in Figure 5-1, there are three steps in the initialization program in the function-call setup() for any sensor modules.

1. Set UART: Set the UART module and its baud rate.
2. I^2C initialization: Set the I^2C pin and its bit rate.
3. Save the module parameter to the variables that will be transacted by I^2C bus.

```
void setup()
{
     . . .

      Serial_begin(9600);          1. Set UART port

      I2C_Init();                  2. I2C Initialization

       LED_Init();

      SlvDevInit(&LedDev);         3. Save module parameter into
      SlvDataStore();              variables that will be transacted by
                                   I2C bus

      . . .
}
```

Figure 5-1 Initialization Program of Setup() for All Sensor Modules

5.1.2 Function-Call Loop()

As shown in Figure 5-2, in the function-call loop() the following steps need to be completed for any of the sensor modules.

1. Arrange and handle the parameter received from master.
2. Save parameter into the variables that will be transacted with master module by I^2C bus.

```
void loop()
{
     . . .

      TID_SlvRxUpdate();

      /* LED code */
      if(LedDev.Output.data2.value==0)
      {
            if(LedDev.Output.data1.value == 1)
           {
                 Led_Blink_Start();
                 LedDev.Output.data1.value=0;
           }
      }
      else
      {
            Led_Stop();
      }
      Led_Blink_Check();

      //update TID parameter
      SlvDataStore();

     . . .
}
```

Arrange and handle the parameter received from

Save parameter into the variables that will be transacted by I^2C bus

Figure 5-2 Program Execution of loop() for All Sensor Modules

5.2 Adding a New Arduino Module

5.2.1 Hardware Stage

In the example of adding a new Arduino module, Arduino MEGA is used to implement as a new sensor module when it connects with the Arduino RGB LED module. In Figure 5-3, these three R, G and B connectors on the Arduino MEGA should be connected with the pins that have the same name on the Arduino RGB LED module to control the display color of LED. The SDA and SCL connectors on the Arduino MEGA are the I^2C interface that should be connected to the NuMaker Brick platform. Finally, the ground of Arduino MEGA should also be connected with the ground of NuMaker Brick for the common ground.

Note: When the NuMaker Brick is connected with the new Arduino LED sensor module, the original LED sensor module that belongs to the NuMaker Brick must be removed from the NuMaker Brick platform.

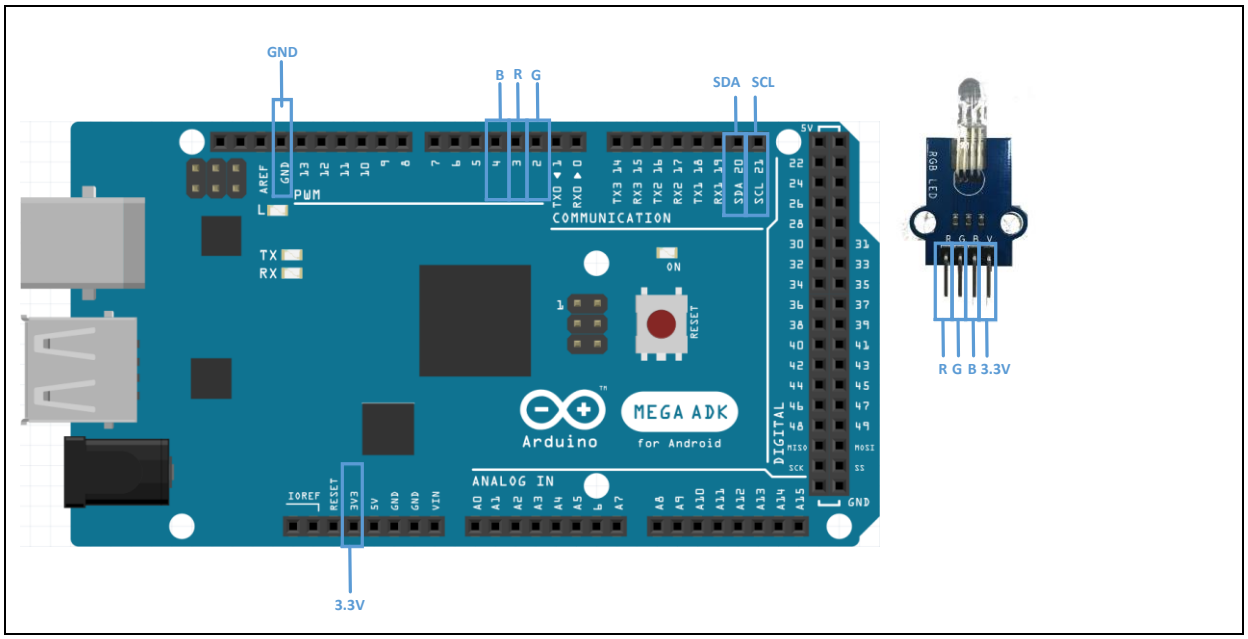

Figure 5-3 Schematic Diagram of Arduino MEGA and Arduino RGB LED Module

5.2.2 Software Stage

When adding a new Arduino module, two parts of software should be modified: setup() and loop() function-calls.

- Function-call setup():

The new program (LED Initialization) inside setup() function-call for the Arduino module is almost the same as the original program of the LED sensor module in NuMaker Brick platform, as shown in Figure 5-4.

```
void setup()
{
     . . .

       Serial_begin(9600);

       I2C_Init();

       LED_Init();

       SlvDevInit(&LedDev);
       SlvDataStore();

      .  .  .
}
```

LED Initialization

Figure 5-4 LED Initialization Program in setup() for Arduino LED Module

- Function-call loop():

The loop() function-call can be separated into two parts:

- **Program Executed Periodically:** It is not necessary to modify any programs for the Arduino LED module.
- **Program Executed Non-Periodically:** To create the non-periodic execution programs as shown in Figure 5-5 for the Arduino LED module.

```
void loop()
{
     . . .

      TID_SlvRxUpdate();

      /* LED code */                          Non-periodic execution program
      if(LedDev.Output.data2.value==0)
      {
            if(LedDev.Output.data1.value == 1)
            {
                  Led_Blink_Start();
                  LedDev.Output.data1.value=0;
            }
      }
      else
      {
            Led_Stop();
      }
      Led_Blink_Check();

      //update TID parameter
      SlvDataStore();

     . . .
}
```

Figure 5-5 LED Execution Program in loop() for Arduino LED Module

6 APPENDIX A - APP DESIGN CONCEPT

The whole platform consists of a master module and some sensor modules, and thus user needs to set different parameters for different situations, for example, the humidity alarm value and the sonar alarm distance. It is very tedious when there is a little change in this whole platform and it requires users to re-program all the flashes of chips on the whole platform system. So planning to design the APP can communicate with the master and command the master to modify the parameter of the sensor module which will be greatly improved the convenience of the user..

There are three main pages that can be selected on APP:

- **Index Page:** User can briefly understand the parameter or status of each sensor module and set the connection port on this page.
- **Device Link Page:** User can easily set the relationships between each sensor module.
- **Sensor Module Page:** User can see the detailed parameter or status on each sensor module.

The APP is communicated with the master module through the UART interface. Some basic commands that APP can send to master are listed below:

1. Start to send data: @ss
2. Stop sending data: @sp
3. Set transmission format: @tx

The following sections will introduce some additional features.

6.1 Index Page

Figure 6-1 shows the Index page on APP. This page has two main functions:

- To display how many sensor modules those are connected at the moment to master module.
- To display the current Input parameter from each sensor module.

The following describes how the master module communicates the two kinds of parameters mentioned above with sensor modules. The data length is two bytes for each data.

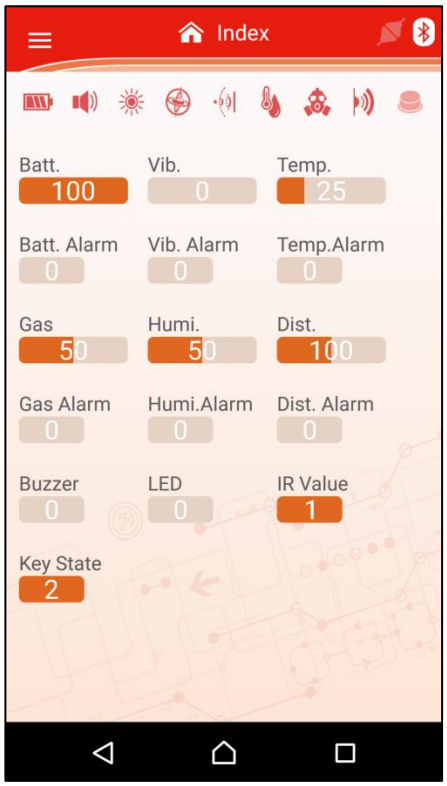

Figure 6-1 Index Page

The transmission process on maser module for this index page can be classified into two stages:

- To transmit how many sensor modules are connected to the master module.
- To transmit the parameter that received from each sensor module.

If there are any errors occur under transmission, the transmission stage will be back to stage 1 to reconfirm the status of connection between the master module and sensor modules.

The transmission commands used to switch to the Index page on APP are listed below in sequence:

1. Send @tx, to request master to send the information on Index page
2. Send @ss, to request master to start to send information
3. send @sp, to request master to stop sending data if there are any errors occur
4. Send @tx, to request master to switch to this page again
5. wait 500 ms
6. Send @ss, to request master to start to send information

The following describes the report descriptor that APP receives from the master module. Table 6-1 lists the data format that APP received in the first stage of Index page.

Data Name	Data Content	Data Length	Description
Inform Start Flag	257 (1, 1)	2 bytes	Start to transmit data
First Stage Data Length	8	2 bytes	There are total 4 data in the first stage, 8 bytes
Sensor Module Number	8	2 bytes	The total number of sensor modules
Connected Module Number	Variable	2 bytes	The number of sensor modules that connected
Second Stage Data Length	30	2 bytes	There are total 15 data in the second stage, 30 bytes

Table 6-1 Data Format that APP Received in the First Stage of Index Page

After the forth data (Connected Module Number) is received and handled, user can get the status of "Link to device" from the icons that are turned on or turned off on this APP Index page. From the analyzing of the forth data, the 1st bit is the INDBATTERY and the battery icon on the APP Index page will be turned on if this bit is 1. The 2nd bit is the INDBUZZER and the buzzer icon on the APP Index page will be turned on if this bit is 1. The 3rd bit is the INDLED and the LED icon on the APP Index page will be turned on if this bit is 1. The 4th bit is the INDAHRS and the AHRS icon on the APP Index page will be turned on if this bit is 1. The 5th bit is the INDSONAR and the sonar icon on the APP Index page will be turned on if this bit is 1. The 6th bit is the INDTEMP and the temperature icon on the APP Index page will be turned on if this bit is 1. The 7th bit is the INDGAS and the gas icon on the APP Index page will be turned on if this bit is 1. The 8th bit is the INDIR and the IR icon on APP Index page will be turned on if this bit is 1. The 5th received data are used to confirm the received data whether are correct in the second stage. This procedure of stage 1 will re-start if any errors occur during transmission.

Table 6-2 lists the data format that APP received in the second stage of Index page.

Data Name	Data Content	Data Length	Description
Second Stage Data Length	30	2 bytes	To re-start stage 1 if the value is not 30
Battery Value	Variable	2 bytes	
Low Power Alarm	Variable	2 bytes	1 indicates the low power state
Buzzer Status	Variable	2 bytes	1 indicates the buzzer is working
LED Status	Variable	2 bytes	1 indicates the LED is working
Vibration Value	Variable	2 bytes	
Vibration Alarm	Variable	2 bytes	1 indicates the vibration alarm is triggered
Sonar Value	Variable	2 bytes	

Distance Alarm	Variable	2 bytes	1 indicates the distance alarm is triggered
Temperature	Variable	2 bytes	
Humidity	Variable	2 bytes	
Temperature Alarm	Variable	2 bytes	1 indicates the temperature alarm is triggered
Humidity Alarm	Variable	2 bytes	1 indicates the humidity alarm is triggered
Gas Value	Variable	2 bytes	
Gas Alarm	Variable	2 bytes	1 indicates the gas alarm is triggered

Table 6-2 Data Format that APP Received in the Second Stage of Index Page

6.2 Device Link Page

Figure 6-2 shows the Device Link Page on APP and the link relationships between each sensor module. The sensor modules can be classified into two types: the input type (Battery, Vibration, Sonar, Temperature and Gas modules) and the output type (Buzzer and LED modules). The transmission process on the device link page can be separated into two stages:

- To display the sensor modules that connected to master module.
- To display the link relationships between each sensor module.

The two stages are classified into three aspects:

- To display the sensor modules those are currently connected to the NuMaker Brick platform.
- To display the sensor module's relationship to each other.
- To receive the sensor module's relationship between each other without updating.

The transmission commands used to switch to the Device Link page on APP are listed below in sequence:

1. **Send @td:** to request master to send information on Device Link page
2. **Send @ss:** to request master to start to send information
3. **send @sp:** to request master to stop sending data if there are any errors occur
4. **Send @tx:** to request master to switch to this page again
5. wait 500 ms
6. **Send @ss:** to request master to start to send information

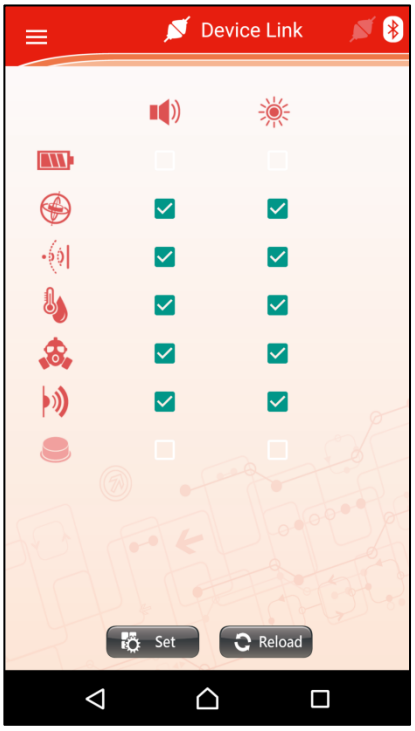

Figure 6-2 Device Link Page

The received data is shown below:

Table 6-3 lists the data format that APP received in the first stage of Device Link page.

Data Name	Data Content	Data Length	Description
Inform Start Flag	257 (1, 1)	2 bytes	Start to transmit data
First Stage Data Length	8	2 bytes	There are total 4 data in the first stage, 8 bytes
Sensor Module Number	8	2 bytes	The total number of sensor modules
Connected Sensor Module Number	Variable	2 bytes	The number of sensor modules that connected
Second Stage Data Length	6	2 bytes	There are total 3 data in the second stage, 6 bytes

Table 6-3 Data Format that APP Received in the First Stage of Device Link Page

Similar to the Index page, the 4th data is used to confirm the status of sensor modules which are currently connected to the platform. The fifth received data are used to confirm the data length in second and the third stages of Device Link page.

Table 6-4 lists the data format that APP received in the second stage of Device Link page.

Data Name	Data Content	Data Length	Description
Second Stage Data Length	6	2 bytes	To re-start to stage 1 if this value is not 6
Connection Status between Buzzer and the Other Modules	Variable	2 bytes	N/A
Connection Status between LED and the Other Modules	Variable	2 bytes	N/A

Table 6-4 Data Format that APP Received in the Second Stage of Device Link Page

The main purpose of the second stage is to show that the buzzer and LED can be controlled by which sensor module (1 indicates that their relationships are connected with each other and the icons will be turned on). The 1st bit of the received second data for the buzzer module indicates the link relationship with the battery sensor module. The 2nd and 3rd bits are skipped out. The 4th bit indicates the link relationship between the buzzer module and vibration module. The 5th bit indicates the link relationship between the buzzer module and sonar module. The 6th bit indicates the link relationship between the buzzer module and the temperature and humidity module. The 7th bit indicates the link relationship between the buzzer module and gas module. The meaning of bits of the received third data for LED module is the same as that for the buzzer module.

The information data received in third stage is exactly the same as that in the second stage. But the difference is that it does not need to update the display status in the third stage. User can click the Reload button on this page to update the last display status. This is because the APP cannot achieve to transact the data with master module and update the status on this page at the same time.

The transmission processing is shown below and can be separated into two parts: Reload and Set.

There are two steps in the Reload stage – the master module requested by the @ss command to transmit the values continuously and to update the display status. Because it just only to receive the values but not to update the display status in the third stage. It is provided to update the display status without switching any the pages. The Set stage is to change the link status of master module with the sensor modules when there are setting changes on the APP by users, the APP will send commands to the master module and the master module will reconnect to these sensor modules by this new link commands. Table 6-5 shows the commands supported in the Set stage.

Transfer Data	Description
“@mz40”+1or0+”\r”	Modify the state between buzzer and battery
“@mz43”+1or0+”\r”	Modify the state between buzzer and vibration
“@mz44”+1or0+”\r”	Modify the state between buzzer and sonar
“@mz45”+1or0+”\r”	Modify the state between buzzer and temperature
“@mz46”+1or0+”\r”	Modify the state between buzzer and gas
“@ml40”+1or0+”\r”	Modify the state between LED and battery
“@ml43”+1or0+”\r”	Modify the state between LED and vibration
“@ml44”+1or0+”\r”	Modify the state between LED and sonar
“@ml45”+1or0+”\r”	Modify the state between LED and temperature
“@ml46”+1or0+”\r”	Modify the state between LED and gas

Table 6-5 Commands Supported in the Set Stage

6.3 Sensor Module Page

There are a total of seven pages for the sensor modules (battery, buzzer, LED, vibration, sonar, temperature and gas) on the APP. Each sensor module page is used to:

- Display the information of sensor module; and
- Modify the data of sensor module

There are three types of each sensor module's data: Input, Output and Feature. In this chapter, the types are simplified into two types: the control type (Output and Feature) and the state type (Input). The data of Input cannot be modified by user so it is classified as the state type. But user can modify the data of both Output and Feature can be modified so they are classified as the control type.

The data of each sensor module can be classified into four steps:

1. Get Device Descriptor
2. Get Report Descriptor
3. Get data information
4. Get data information but only update the data of state type

The transmission commands used to switch the Sensor Module page are listed below in sequence:

1. **Send @tb(*)**, to request master to send information on the sensor (@tb is for battery) module's page
2. **Send @ss**, to request master to start to send information
3. **send @sp**, to request master to stop sending data if there are any errors occur
4. **Send @tx**, to request master to switch to this page again
5. wait 500 ms
6. **Send @ss**, to request master to start to send information

(*): the other commands for sensor modules are listed below:

Sensor module	Order
Battery	@tb
Buzzer	@tz
LED	@tl
Vibration	@ta
Sonar	@ts
Temperature	@tt
Gas	@tg

Table 6-6 Commands to Switch to Sensor Module Page

The buzzer module page is shown in Figure 6-3. Its data can be simplified into two types: control type (Output and Feature) and state type (Input).

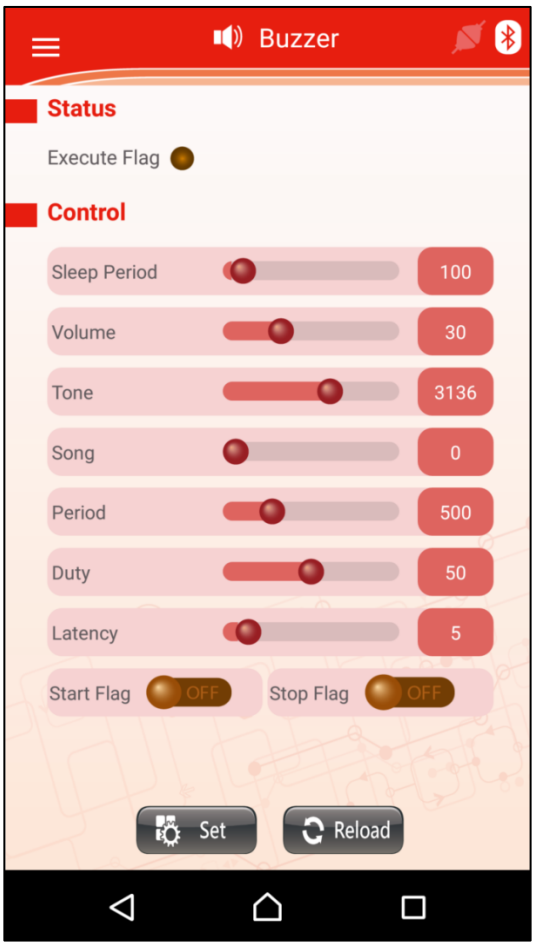

Figure 6-3 Buzzer Module Page

There are three stages for the received data from the sensor module. The first stage is to get the Device Descriptor from the sensor module. If the first byte of the received data is not equal to 257, it indicates that the data has errors for the APP. Table 6-7 lists the data format of received data from the sensor module in the first stage.

Data Name	Data Content	Data Length
Inform Start Flag	257 (1, 1)	2 bytes
Device Descriptor Length	26	2 bytes
Report Descriptor Length	Variable	2 bytes
Get Feature Length	Variable	2 bytes
Set Feature Length	Variable	2 bytes
Get Input Length	Variable	2 bytes
Set Output Length	Variable	2 bytes
Company Code	0	2 bytes

Device Code	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 6-7 Received Data on Sensor Module Page in the First Stage

The second stage is to receive the Report Descriptor from the sensor module. After receiving the parameter address, the parameter length will be saved in the corresponding “datalen” category variables and writes the current value to “arg” variables. For example, the received address is 41 and its length is 1, write the fourth “datalen” variable to 1 and write the “arg” variable to 4). If the first byte of the received data in the second stage is not equal to the length of Report Description received in the first stage, it means that there are some errors inside the received data in this second stage. Table 6-8 lists the data format of received data from the sensor module in the second stage.

Data Name	Data Content	Data Length	Function Description
Report Descriptor Length	Variable		The total length at this stage.
Parameter Class (Feature)	258(1, 2)		Declare the following information type: Feature
Parameter Address	5	1 byte	Declare the parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare the parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum value of the data (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is The maximum value of the data (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
Parameter Address	5	1 byte	Declare the parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare this parameter length(1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum value of the data (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is The maximum value of the data (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value

…	…	…	…
Parameter Class (Input)	258(1, 2)		Declare the following information type: Input
Parameter Address	5	1 byte	Declare the parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare the parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum value of the data (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is The maximum value of the data (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
Parameter Address	5	1 byte	Declare the parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare the parameter length(1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum value of the data (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum value of the data (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
…	…	…	…
Parameter Class (Output)	258(1, 2)		Declare the following information type: Output
Parameter Address	5	1 byte	Declare the parameter address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare the parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum value of the data (8) and next column length (1 or 2)
Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is the maximum value of the data (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
Parameter Address	5	1 byte	Declare the data address (5, 17, 29, 41, 53, 65, 77, 89, 101, 113)
Parameter Length	1~2	1 byte	Declare the parameter length (1~2 bytes)
Minimum Parameter Value	9 or 10	1 byte	Declare the next column is the minimum value of the data (8) and the next column length (1 or 2)

Minimum Parameter Value	variable	1~2 bytes	The minimum parameter value
Maximum Parameter Value	13 or 14	1 byte	Declare the next column is The maximum value of the data (12) and next column length (1 or 2)
Maximum Parameter Value	variable	1~2 bytes	The maximum parameter value
…	…	…	…

Table 6-8 Received Data on Sensor Module Page in the Second Stage

The third stage is to receive all the control and state types (Input, Output and Feature classes) at one time. If the 1st data of the received data in the third stage is not equal to the sum of "Get Input Length", "Set Output Length" and "Get Feature Length" received in the first stage, it means that there are some errors inside the received data in the third stage. It is judged based on the value of “arg” variable of the parameter. If the value of “arg” variable is not equal to zero, it means that the data is meaningful. If the value of “arg” variable is equal to zero, it means that it is ended. Table 6-9 lists the data format of received data from sensor module in the third stage.

Data Name	Data Content	Data Length	Description
Inform Start Flag	Variable	2 bytes	The sum of Get Feature Length, Set Output Length and Get Input Length that received in the first stage
First Feature Parameter Value	Variable	1~2 bytes	
Second Feature Parameter Value	Variable	1~2 bytes	
…	Variable	1~2 bytes	It is decided whether to end by the “arg” variable.
First Input Parameter Value	Variable	1~2 bytes	
Second Input Parameter Value	Variable	1~2 bytes	
…	Variable	1~2 bytes	It is decided whether to end by the “arg” variable.
First Output Parameter Value	Variable	1~2 bytes	
Second Output Parameter Value	Variable	1~2 bytes	
…	Variable	1~2 bytes	It is decided whether to end by the “arg” variable.

Table 6-9 Received Data on Sensor Module Page in the Third Stage

The data received in the fourth stage is almost the same as the data received in the third stage, but only the parameter of state type (Input class) is updated. The transmission processing is shown below and can be separated into two parts: Reload and Set. The commands are sent as follows:

1. Send @sp, to request master to stop sending data if there are any errors occur
2. Send @tb*, to request master to switch to the sensor (@tb is for battery) module’s page
3. wait for 500 ms
4. Send @ss, to request master to start to send information

The link status of the sensor module is changed by the APP and it will be transmitted to the master module. The master will follow these commands to re-connect with the sensor modules by the new link status. Table 6-10 lists the updating commands of APP for master module to update the related parameter of sensor module.

(*): the other commands for sensor modules are shown in Table 6-11:

Transfer Command	Description
“@m*10”+data+”\r”	To update the first Feature parameter
“@m*11”+data+”\r”	To update the second Feature parameter
…	It is decided whether to end by the “arg” variable.
“@m*20”+data+”\r”	To update the first Input parameter
“@m*21”+data+”\r”	To update the second Input parameter
…	It is decided whether to end by the “arg” variable.
“@m*30”+data+”\r”	To update the first Output parameter
“@m*31”+data+”\r”	To update the second Output parameter
…	It is decided whether to end by the “arg” variable.

Table 6-10 Updating Commands of Sensor Module

Sensor Module	Command
Battery	@mb
Buzzer	@mz
LED	@ml
Vibration	@ma
Sonar	@ms
Temperature	@mt
Gas	@mg

Table 6-11 Commands for Modifying Each Sensor Module Parameter

7 APPENDIX B - SENSOR MODULE PARAMETER INFORMATION

The transaction parameter between master and sensor modules can be separated into three parts:

- Device Descriptor
- Report Descriptor
- Parameter information

The parameter of each module can be separated into two parts: Device Description and parameter. Parameter can be classified into three classes: Feature, Output and Input. There are seven sensor modules: battery, buzzer, LED, vibration, sonar, temperature and humidity, and gas in the original NuMaker Brick platform. The following tables list the parameters for each sensor module, including the Device Descriptor, Output, Input and Feature. The values of brackets inside the maximum and minimum cells of the tables represent the maximum or minimum parameter of the next frame.

7.1 Battery Module

Data Name	Data Content	Data Length
I²C Address	20	2 bytes
Device Descriptor Data Length	26	2 bytes
Report Descriptor Data Length	36	2 bytes
Get Input Data Length	5	2 bytes
Set Output Data Length	0	2 bytes
Get Feature Data Length	6	2 bytes
Set Feature Data Length	6	2 bytes
Company Code	0	2 bytes
Device Code	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 7-1 Device Descriptor of Battery Module

Data Name	Data Address	Data Length	Minimum (Position)	Maximum (Position)
Sleep Period	5	2 bytes	0 ms (10)	1024 ms (14)
Low Battery Alarm Value	17	2 bytes	0 % (10)	100 % (14)

Table 7-2 Feature Parameter of Battery Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Battery Value	5	2 bytes	0 (10)	1024 (14)
Low Battery Alarm Flag	17	2 bytes	0 (9)	100 % (14)

Table 7-3 Input Parameter of Battery Module

7.2 Buzzer Module

Data Name	Data Content	Data Length
I²C Address	21	2 bytes
Device Descriptor Data Length	26	2 bytes
Report Descriptor Data Length	74	2 bytes
Get Input Data Length	3	2 bytes
Set Output Data Length	4	2 bytes
Get Feature Data Length	12	2 bytes
Set Feature Data Length	12	2 bytes
Company Code	0	2 bytes
Device Code	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 7-4 Device Descriptor of Buzzer Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Sleep Period	5	2 bytes	0 ms (10)	2048 ms (14)
Volume	17	1 byte	0 % (9)	100 % (13)
Frequency	29	2 bytes	0 Hz (10)	5000 Hz (14)
Song	41	1 byte	0 (9)	2 (13)
Period	53	2 bytes	0 ms (10)	2048 ms (14)
Duty	65	1 byte	0 % (9)	100 % (13)
Alarm Time	77	1 byte	0 s (9)	60 s (13)

Table 7-5 Feature Parameter of Buzzer Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Executing Flag	5	1 byte	0 (9)	1 (13)

Table 7-6 Input Parameter of Buzzer Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Start Executing Flag	5	1 byte	0 (9)	1 (13)
Stop Executing Flag	17	1 byte	0 (9)	1 (13)

Table 7-7 Output Parameter of Buzzer Module

7.3 LED Module

Data Name	Data Content	Data Length
I²C Address	22	2 bytes
Device Descriptor Data Length	26	2 bytes
Report Descriptor Data Length	74	2 bytes
Get Input Data Length	3	2 bytes
Set Output Data Length	4	2 bytes
Get Feature Data Length	12	2 bytes
Set Feature Data Length	12	2 bytes
Company Code	0	2 bytes
Device Code	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 7-8 Device Descriptor of LED Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Sleep Period	5	2 bytes	0 ms (10)	1024 ms (14)
Light	17	1 byte	0 % (9)	100 % (13)
Color	29	2 bytes	0 (10)	2048 (14)
Blink	41	1 byte	0 (9)	2 (13)
Period	53	2 bytes	0 ms (10)	2048 ms (14)
Duty	65	1 byte	0 % (9)	100 % (13)
Alarm Time	77	1 byte	0 s (9)	60 s (13)

Table 7-9 Feature Parameter of LED Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Executing Flag	5	1 byte	0 (9)	1 (13)

Table 7-10 Input Parameter of LED Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Start Executing Flag	5	1 byte	0 (9)	1 (13)
Stop Executing Flag	17	1 byte	0 (9)	1 (13)

Table 7-11 Output Parameter of LED Module

7.4 Vibration Module

Data Name	Data Content	Data Length
I^2C Address	23	2 bytes
Device Descriptor Data Length	26	2 bytes
Report Descriptor Data Length	34	2 bytes
Get Input Data Length	5	2 bytes
Set Output Data Length	0	2 bytes
Get Feature Data Length	5	2 bytes
Set Feature Data Length	5	2 bytes
Company Code	0	2 bytes
Device Code	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 7-12 Device Descriptor of Vibration Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Sleep Period	5	2 bytes	0 ms (10)	1024 ms (14)
Vibration Alarm Value	17	1 byte	0 % (9)	10 (13)

Table 7-13 Feature Parameter of Vibration Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Vibration Value	5	2 bytes	0 (10)	720 ℃ (14)
Vibration Alarm Flag	17	1 byte	0 (9)	1 (13)

Table 7-14 Input Parameter of Vibration Module

7.5 Sonar Module

Data Name	Data Content	Data Length
I²C Address	24	2 bytes
Device Descriptor Data Length	26	2 bytes
Report Descriptor Data Length	36	2 bytes
Get Input Data Length	5	2 bytes
Set Output Data Length	0	2 bytes
Get Feature Data Length	6	2 bytes
Set Feature Data Length	6	2 bytes
Company Code	0	2 bytes
Device Coe	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 7-15 Device Descriptor of Sonar Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Sleep Period	5	2 bytes	0 ms (10)	1024 ms (14)
Distance Alarm Value	17	2 bytes	0 cm (10)	200 cm (14)

Table 7-16 Feature Parameter of Sonar Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Distance Value	5	2 bytes	0 (10)	400 cm (14)
Distance Alarm Flag	17	1 byte	0 (9)	1 (13)

Table 7-17 Input Parameter of Sonar Module

7.6 Temperature Module

Data Name	Data Content	Data Length
I²C Address	25	2 bytes
Device Descriptor Data Length	26	2 bytes
Report Descriptor Data Length	58	2 bytes
Get Input Data Length	8	2 bytes
Set Output Data Length	0	2 bytes
Get Feature Data Length	8	2 bytes
Set Feature Data Length	8	2 bytes
Company Code	0	2 bytes
Device Code	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 7-18 Device Descriptor of Temperature Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Sleep Period	5	2 bytes	0 ms (10)	1024 ms (14)
Temperature Alarm Value	17	2 bytes	0 ℃ (10)	100 ℃ (14)
Humidity Alarm Value	29	2 bytes	0 % (10)	100 % (14)

Table 7-19 Feature Parameter of Temperature Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Temperature Value	5	2 bytes	0 (10)	100 ℃ (14)
Humidity Value	17	2 bytes	0 (10)	100 % (14)
Temperature Alarm Flag	29	1 byte	0 (9)	1 (13)
Humidity Alarm Flag	41	1 byte	0 (9)	1 (13)

Table 7-20 Input Parameter of Temperature Module

Data Name	Data Content	Data Length
I²C Address	24	2 bytes
Device Descriptor Data Length	26	2 bytes
Report Descriptor Data Length	36	2 bytes
Get Input Data Length	5	2 bytes
Set Output Data Length	0	2 bytes
Get Feature Data Length	6	2 bytes
Set Feature Data Length	6	2 bytes
Company Code	0	2 bytes
Device Code	0	2 bytes
Product Code	0	2 bytes
Firmware Version	0	2 bytes
Device Class Code	0	2 bytes
Reserved	0	2 bytes
Reserved	0	2 bytes

Table 7-21 Device Descriptor of Gas Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Sleep Period	5	2 bytes	0 ms (10)	1024 ms (14)
Gas Alarm Value	17	2 bytes	0 cm (10)	100 % (14)

Table 7-22 Feature Parameter of Gas Module

Parameter Name	Parameter Address	Parameter Length	Minimum (Position)	Maximum (Position)
Gas Value	5	2 bytes	0 (10)	100 % (14)
Gas Alarm Flag	17	1 byte	0 (9)	1 (13)

Table 7-23 Input Parameter of Gas Module

8 REVISION HISTORY

Date	Revision	Description
2016.12.23	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.