

Technical Introduction to Embedded System with BBC micro:bit

Atsushi Matsubara, Takayuki Makihara, Seiko Yamano

Production Division

1 Introduction

BBC micro:bit, an educational microcontroller board, has been actively incorporated programming into the curriculum of primary and secondary education institutions as an aspect of education. The micro:bit has the modules that stimulate the human sensory organs: button, speaker, microphone and LED matrix. The users thus can quickly program these modules to operate. Furthermore, they can easily connect the micro:bit to the external modules related to measurement and control fields because the edge connectors of it has distinctive hardware specifications.

Microsoft's MakeCode, an open source platform, has microPython and Scratch-type languages, although many development environments for the micro:bit vary depending on the programming languages. The former language is Python optimized for the microcontroller and the latter one is a visual-based one which creates a program combining or stacking the processing blocks. In addition, Arduino IDE, the integrated development environment with C-style language, allows users to develop for the micro:bit after installing libraries for it.

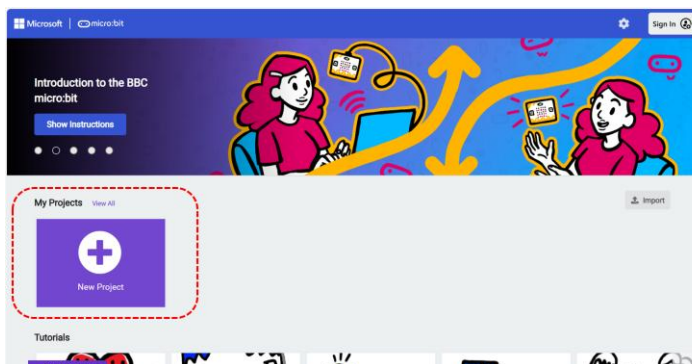
This paper discusses fundamental skills for the embedded system with the micro:bit such as setting up the development environment and controlling the modules with it. We specifically focus on the following: the first is to effectively use MakeCode and Arduino IDE, the second is to install the libraries for modules and the third is to input/output signal from/to the modules and to precisely control them.

2 Development Environment

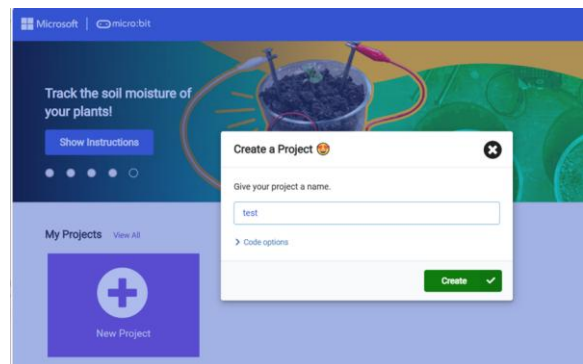
2.1 MakeCode

Microsoft provides the development environment “MakeCode” for the micro:bit on the online site (URL: <https://makecode.microbit.org/>). The starting step in programming is to create a new project using “New Project” (see Figure 1).

The development environment shown in Figure 2 has three main areas: left area is “simulator” that can simulate the created program and check if it properly works, center one “toolbox” organized processing blocks by categories, right one “workspace” for coding a program. Additionally, the environment has event-based button controls “Download” and



(a) Online site



(b) A new project

Figure 1. MakeCode

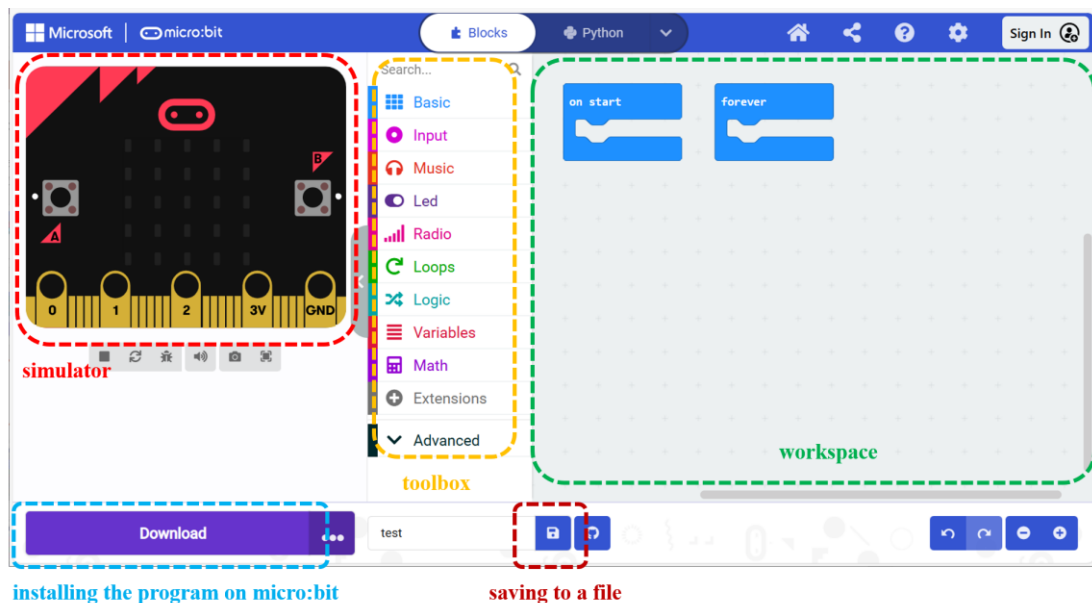


Figure 2. Development environment

“Save” which transfer a program to micro:bit and save it as a file to a terminal or a PC.

Two processing blocks in the “workspace” area are “on start” and “forever” ones: the mission of the former is to initialize global variables and functions, and that of the latter to loop through the code in itself. The processing order for these blocks is that the latter follows the former.

The “toolbox” area has the following categories for tools: “Basic” stocks the blocks that display a text or a geometric image on 5×5 LED Matrix and pause the code processing in addition to the above two blocks; “Input” does them that process input signals and handle events from the modules (e.g., “a button is pressed”); “Music” that plays a sound or a music using the micro:bit’s speaker; “Led” that control 5×5 LED Matrix (i.e., turn a LED on/off). “Loop”, “Logic”, “Variables” and “Math” take on repetition, conditional branch, storing a value and calculation, respectively; “Extensions” allows users to install libraries for external connection modules such as other company’s sensors and controllers.

The blocks in these categories are placed on the “workspace” area using drag and drop operations. A program thus completes if repeatedly combining the multiple blocks like pieces of a jigsaw puzzle.

2.2 Arduino IDE

As mentioned above (chapter 1), “Arduino IDE”¹⁾, the development environment for “Arduino” micro-controller device, allows users to create a program for the micro:bit when installing its own libraries.

“Arduino IDE” needs “Nordic Semiconductor nRF5 Boards Manager” because the micro:bit version 2.2.1, reported in this paper, has the nRF52833 processor provided by Nordic Semiconductor.

After installing “Arduino IDE”, the

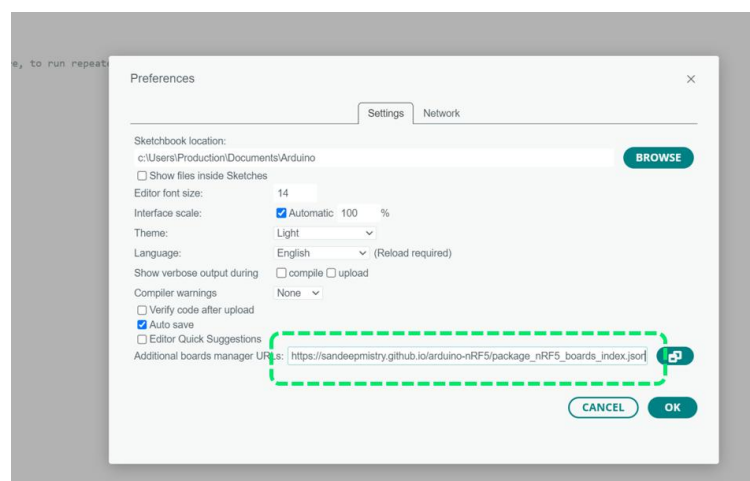


Figure 3. “References” window

registration of the board for micro:bit is based on the following procedure: the first is to input the below URL to “Additional boards manager URLs” in “References” window displayed after selecting the “File” → “References...” menu (see Figure 3).

https://sandeepmistry.github.io/arduino-nRF5/package_nRF5_boards_index.json

the second is to install “Nordic Semiconductor nRF5 Boards” using the “Tools” → “Board” → “Boards Manager ...” menu. The board for the target (micro:bit version 2.2.1) is compatible with “BBC micro:bit V2” registered in “Board” menu (see Figure 4).



Figure 4. Registering the target board

“Arduino IDE” with two processing areas, `setup()` and `loop()` functions, has the same system configuration as “MakeCode”.

3 micro: bit

3.1 Overview

As shown in Figure 5, the front side of micro:bit version 2.2.1 has the modules that stimulate the human’s five senses such as two tact switches (Button A and B), a microphone and a 5×5 LED Matrix display whereas the back side of it does a speaker, a tact switch for reset processing, sensors (acceleration sensor and magnetic one) and two power supply

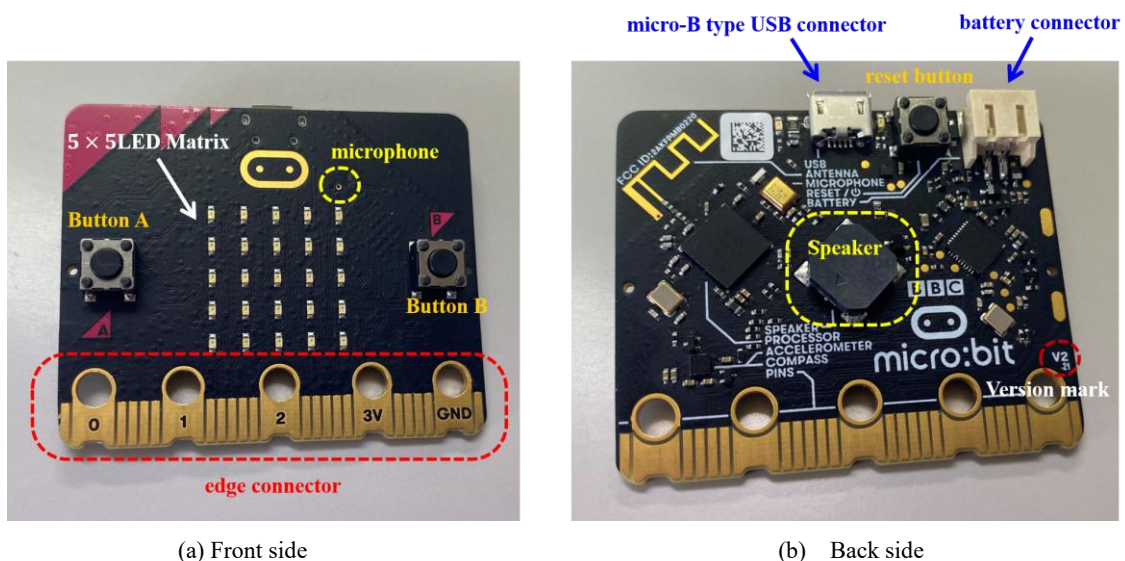


Figure 5. Overview of micro:bit

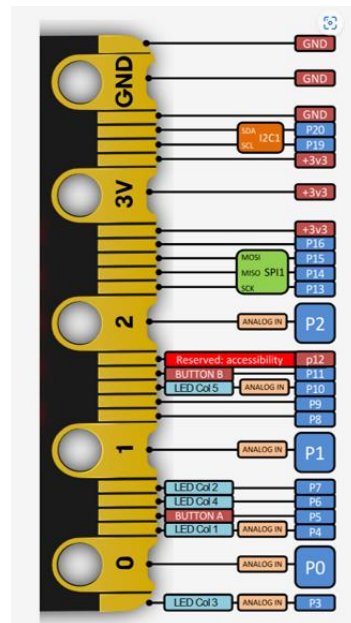
connectors (one is micro-B type USB connector and the other is JST standard one for an battery box). Moreover, the bottom of it equips with an edge connector, which can connect to external modules as analog/digital inputs/outputs. The edge connector is especially designed to use preferentially five pins: three pins, “P0”, “P1”, “P2”, are for analog inputs, the other two ones are for a power supply voltage and a ground terminal.

Figure 6 shows assignment for pins of the micro:bit. The number of the pins with the edge connector is 25, and its details are 19 for analog/digital inputs/outputs, 3 for power supply voltages and 3 for ground terminals [see the figure (a)].

Furthermore, the micro:bit has 14 types of potential pins (“P21”-“P34”) on the board. Assignment information for these pins is stored in the following (hidden) file. The figure (b) shows part of the file “variant.cpp” contents.

C:\Users\¥[user name on PC]¥AppData¥Local¥Arduino15¥packages¥sandeepmistry¥hardware¥nRF5¥0.8.0¥variants¥BBCmicrobitV2¥variant.cpp

For instance, operating each LED on the 5×5 LED Matrix display needs combining the pin (“P4”, “P7”, “P3”, “P6” or



(a) Edge connector pins*

```

/* Pins for micro:bit V2 */
#include "variant.h"

const uint32_t g_ADigitalPinMap[] = {
  // 0 - 4
  0, // A0, LEFT PAD
  1, // A1, MIDDLE PAD
  2, // A2, RIGHT PAD
  3, // A3, COL3
  4, // A4, COL1

  // 5 - 9
  5, // BTN A
  6, // COL4, P1.05
  7, // COL2
  8, // NFC2
  9, // NFC1

  // 10 - 16
  10, // A5, COL5
  11, // BTN B
  12, // SCK
  13, // MISO
  14, // MOSI
  15, // P1.02

  // 17 - 18
  17, // (uint32_t)-1, // 3.3V
  18, // (uint32_t)-1, // 3.3V

  // 19 - 20
  19, // SCL, P1.00
  20, // SDA, P1.00

  // 21 - 25
  21, // ROW1
  22, // ROW2
  23, // ROW3
  24, // ROW4
  25, // ROW5
};

```

(b) Part of pins assigned to the board

Figure 6. Assignment for pins

* see URL: <https://tech.microbit.org/hardware/edgeconnector/>

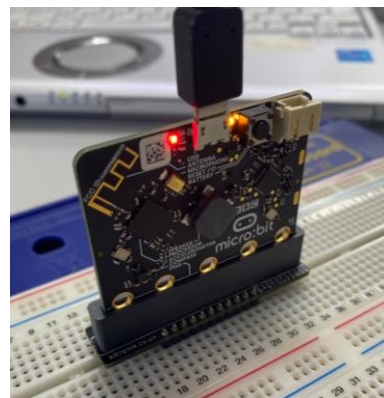


Figure 7. Breadboard Breakout

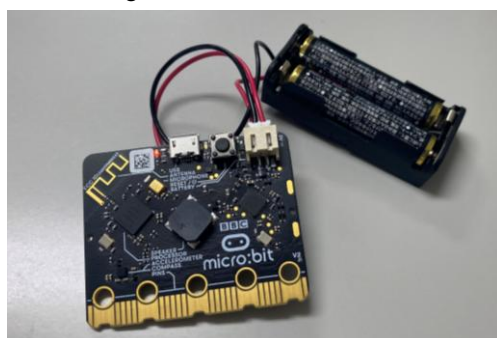


Figure 9. Battery box

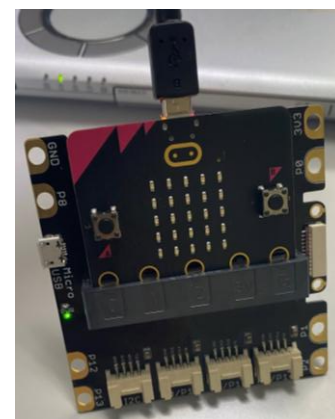
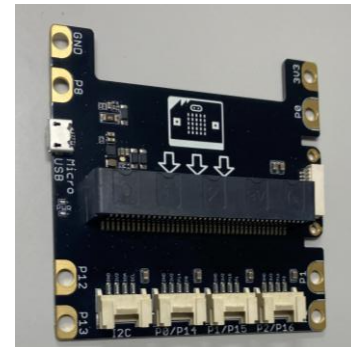


Figure 8. Grove Shield

“P10” corresponds to “LED Col1”, … , or “LED Col5”, respectively) on the edge connector with the pin (“P21”, …, “P25” does to “LED Row1”, … , or “LED Row5”, respectively) on the board (see section 4.1).

3.2 Accessories

BBC micro:bit has various support accessories to enhance the compatibility with the modules without the same connector standard.

Kitronik Ltd’s “Breadboard Breakout” shown in Figure 7, a conversion board to spread out pins of the micro:bit on a bread board, is generally used when forming a circuit network using many pins. Figure 8 shows SseedStudio “Grove Shield”. The shield has Grove connector system which has been widely used because of a high convenience. The micro:bit thus easily links to measuring instruments and controllers with the connector system.



Figure 10. A-micro B type USB

The battery box shown in Figure 9 drives the micro:bit through JST standard cable. The box needs two AAA batteries, which is useful for portable power supply. The use of A-micro B type USB cable is to transfer a program to the micro:bit and drive it (see Figure 10).

4 Examples of implementation²⁾

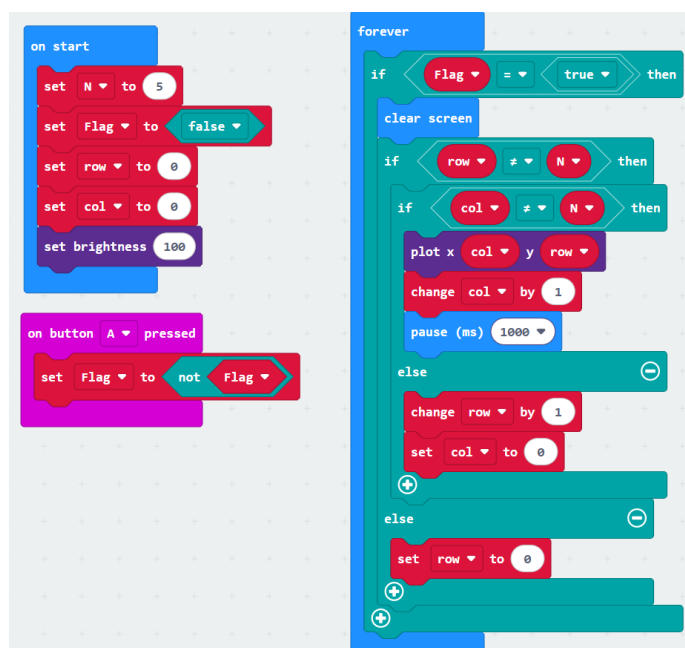
This chapter presents five operation cases using the following modules: first, 5×5 LED Matrix display on the micro:bit, next, Adafruit NeoPixel Stick, then, Kitronik Zip Tile, after that, SseedStudio 4 Digit Display and finally, speaker on the micro:bit.

4.1 5×5 LED Matrix display

We design that the lighting location of an LED forming a 5×5 LED Matrix display changes sequentially the next location of one. Code 1 is an event driven program to operate the matrix display using Button A on the micro:bit. Raising of the event that Button A is pressed leads to begin to turn a LED on. This type of event will be called “button event” or “Button A (or B) event” from here. Handle for the button event implements using “on button X pressed” block belongs to category “Input” in the “toolbox” (where $X \in \{A, B, A+B\}$).

We define two states (“start or restart” and “pause”) for the event that the lighted location of a LED changes. Two effects are produced whenever raising the Button A event: one is that the two states are cyclically switched and the other is that Boolean variable “Flag”, the judgement criteria to select the tasks depending on the states, is done or reversed polarities (“true” or “false”).

A “plot” block in category “Led” is used to turn a LED on. The block is set coordinate information (x, y) as its arguments: x is horizontal, y vertical.



Code 1. Program code to change the turn-on location (MakeCode)

Symmetrically, Code 2 shows a program coded on “Arduino IDE”. The program includes the header file “Adafruit_Microbit.h” as the library to control and operate the matrix display. The library is available when installing three libraries – “BLEPeripheral Library”, “Adafruit GFX Library” and “Adafruit microbit Library”, in “LIBRARY MANAGER” window displayed through menu steps: “Sketch”→”Include Library”→”Manege Libraries ...”.

Adafruit_Microbit_Matrix class instance is initialized in setup () function using the begin () one of the class.

Turning a LED on performs using Adafruit_Microbit_Matrix.drawPixel() function with three arguments: the first specifies a horizontal location of a LED, the second does a virtual one of it and the final the digital value “1” (means “LED on”).

The task for raising of the button event typically uses a combination of the digitalRead() and delay() functions. However, the use of delay () function could cause delayed or no response for the event because the function prevents other processing while working. Hence, the program applies the interrupt function “attachInterrupt()” which breaks out of a loop processing in order to call a user-defined function that performs the desired processing when raising an event. The

```
#include <Adafruit_Microbit.h>
#define Btn 5
#define N 5
Adafruit_Microbit_Matrix microbit;
bool flag;
int row, col;
void setup() {
  flag=false;
  pinMode(Btn,INPUT_PULLUP);
  microbit.begin();
  row=col=0;
  attachInterrupt(digitalPinToInterrupt(Btn),Statement,FALLING);
}
void loop() {
  if(flag==true){
    microbit.clear();
    microbit.drawPixel(col, row, 1);
    if(col != N) {
      col++;
      delay(1000);
    }
    else {
      col=0;
      row++;
    }
    if(row==N) row=0;
  }
}
void Statement(){
  flag=!flag;
}
```

Code 2. Program code to change a turn-on location (Arduino IDE)

attachInterrupt() function specifies the interrupt number assigned to the micro:bit board as the first argument, a user-defined function to call as the second one and an event type as the third one.

The interrupt number is equal to the pin one “5” (=P5) assigned to Button A as edge connector because the raising and handling of the Button A or B event drives as an interrupt action. For reference, digitalPinToInterrupt() function wrapping the pin number will be used if the interrupt number is unknown. The our-defined function “Statement” has the same task as “MakeCode” that the variable “flag” switches polarities whenever raising the button event. The third argument of attachInterrupt() function set “FALLING”. The mode “FALLING” means that raising of the event is associated with state of the pin transiting from “HIGH” to “LOW”. It is due to the fact that the micro:bit is connected to a power supply voltage (V_{cc}) via a pull-up resistor.

As mentioned in section 3.1, a configuration for an interface to transfer signals on “micro:bit – matrix display” line is based on a combination of a pin for the board and one for the edge connector. A program needs two integer arrays, ROW [5] = {21,22,23,24,25} and COL [5] = {4,7,3,6,10}, if be not used the above “Adafruit microbit Library” (and the other two libraries).

Operation of a LED specifically depends on the following condition of digitalWrite() function : “turning on ” sets ROW[] as “HIGH” and COL[] as “LOW”. “turning off” ROW[] as either and COL[] as “HIGH”.

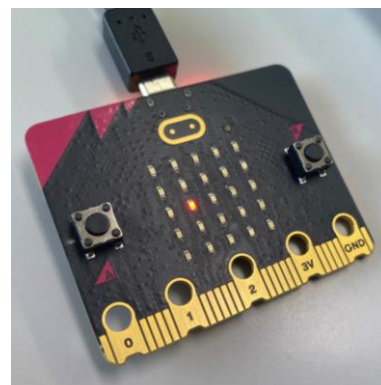


Figure 11. an example to turn a LED on

Figure 11 shows an example to turn a LED on.

4.2 Adafruit's NeoPixel Stick

We next present an example operating Adafruit NeoPixel Stick with eight full color LEDs. Figure 12 shows a wiring diagram that the eight LEDs respectively begin to blink “While”, “Red”, “Orange”, “Yellow”, “Lime”, “Aqua”, “Blue”, and “Purple” colors when the externally connected tact switch is pressed.

The State for the blinking alternates between “start” and “stop” whenever raising the tact switch event, as the example in the section 4.1.

Kitronik Breadboard breakout links the micro:bit to the Stick and the tact switch via the pin numbers “P0” and “P1”, respectively. Moreover, the micro:bit forms not only “P0-

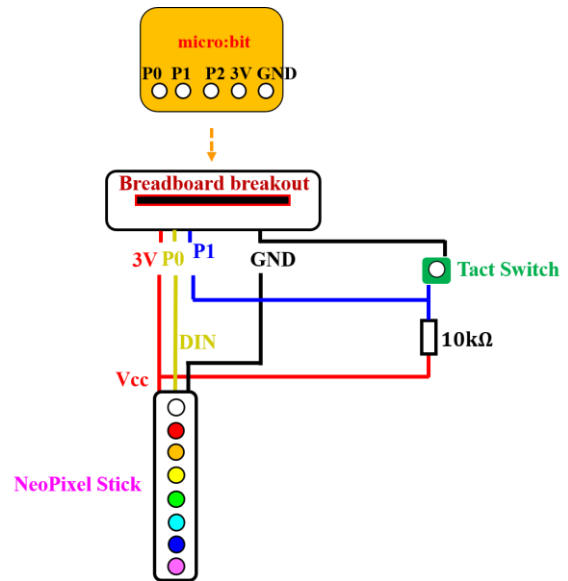


Figure 12. A wiring diagram

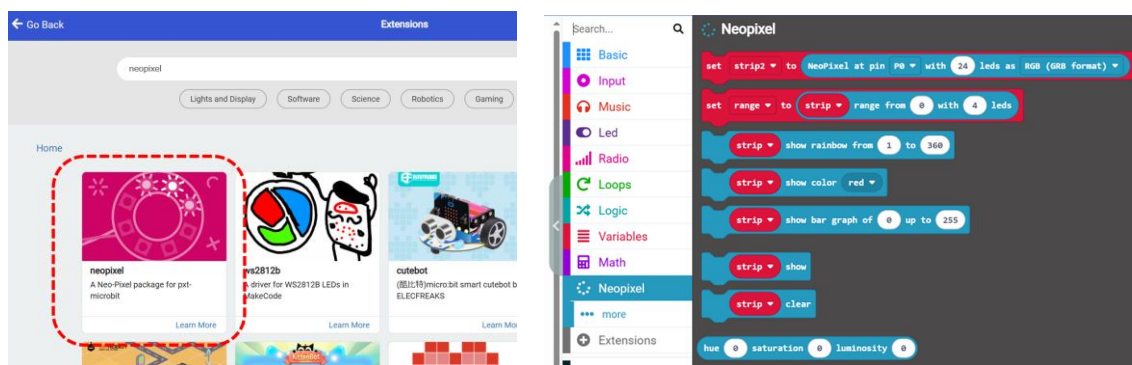
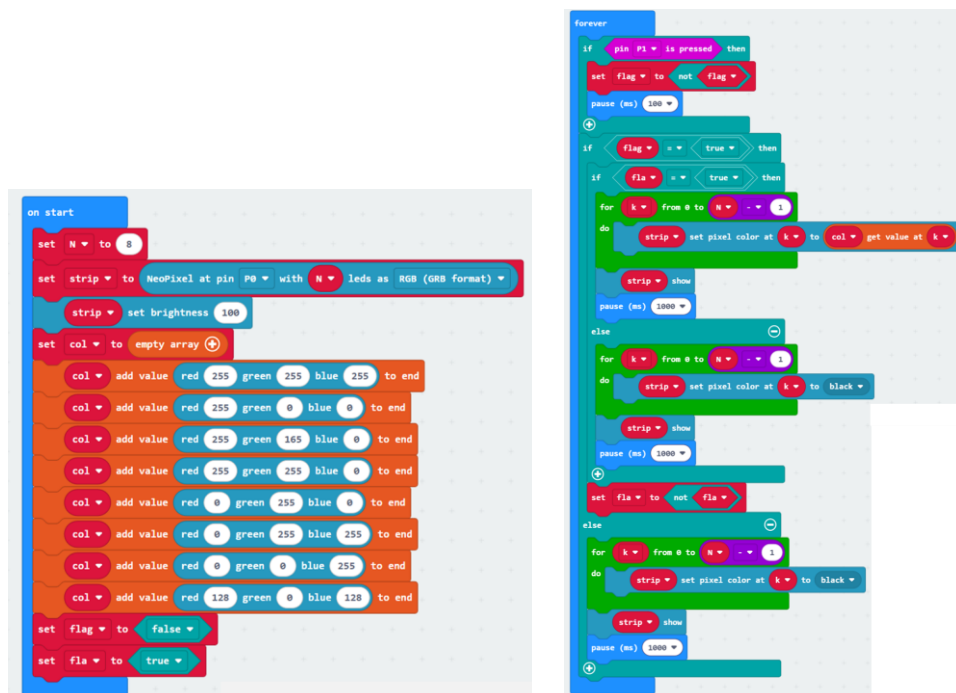


Figure 13. Adding category for NeoPixel to “toolbox” area



Code 3. A sample program for NeoPixel Stick (MakeCode)

DIN” line but also “3V- V_{cc} ” and GND lines with the Stick, and it connects to the tact switch using the pull-up resistor “10k Ω ”. The pull-up connecting means that inputs digital value “0” through “P1” due to GND if the switch is pressed, does “1” by V_{cc} otherwise.

The use of the Stick needs adding the library for NeoPixel to “toolbox” area (see Figure 13). Code 3 is a sample program.

The variable “strip” block in the added category “Neopixel” can set a pin number for digital output (= “P0”) and a format for color information. The color information of each LED is stored in our-defined array “col []” as RGB information.

Code 4 parallelly shows a sample program coded on “Arduino IDE”.

This program uses the header file “Adafruit_NeoPixel.h” as library for the Stick. Installing the library, as in section 4.1, just selects “Adafruit NeoPixel” in “LIBRARY MANAGER” window.

Regarding “Adafruit_NeoPixel” class, its constructor specifies the following three values when creating the instance: the number of LEDs (=8), a pin number (=0) and driving frequency of a LED (=800kHz). The class instance is initialized using begin() function. The handling of the tact switch event depends on the value of Boolean variable “flag” using attachInterrupt() function. setPixelColor() and show() functions in

```
#include<Adafruit_NeoPixel.h>
#define N 8
#define PIN0 0
#define PIN1 1
Adafruit_NeoPixel pixels =Adafruit_NeoPixel(N,PIN0,NEO_GRB+NEO_KHZ800);
bool flag;
void setup() {
  pixels.begin();
  pixels.setBrightness(32);
  flag=false;
  attachInterrupt(digitalPinToInterrupt(PIN1),Statement,FALLING);
}
```

Code 4. A sample program for NeoPixel Stick (setting: Arduino IDE)

```
void loop() {
  if(flag==true){
    for(int i=0;i<N;i++){
      switch(i){
        case 0:
          pixels.setPixelColor(i,pixels.Color(255,255,255));
          break;
        case 1:
          pixels.setPixelColor(i,pixels.Color(255,0,0));
          break;
        case 2:
          pixels.setPixelColor(i,pixels.Color(255,165,0));
          break;
        case 3:
          pixels.setPixelColor(i,pixels.Color(255,255,0));
          break;
        case 4:
          pixels.setPixelColor(i,pixels.Color(0,255,0));
          break;
        case 5:
          pixels.setPixelColor(i,pixels.Color(0,255,255));
          break;
        case 6:
          pixels.setPixelColor(i,pixels.Color(0,0,255));
          break;
        case 7:
          pixels.setPixelColor(i,pixels.Color(128,0,128));
          break;
      } // switch
    } // for
    pixels.show();
    delay(1000);
    LEDOFF();
    delay(1000);
  }
  else LEDOFF();
}
void Statement(){
  flag=!flag;
}
void LEDOFF(){
  pixels.clear();
  pixels.show();
}
```

Code 4. A sample program for NeoPixel Stick
(processing: Arduino IDE)

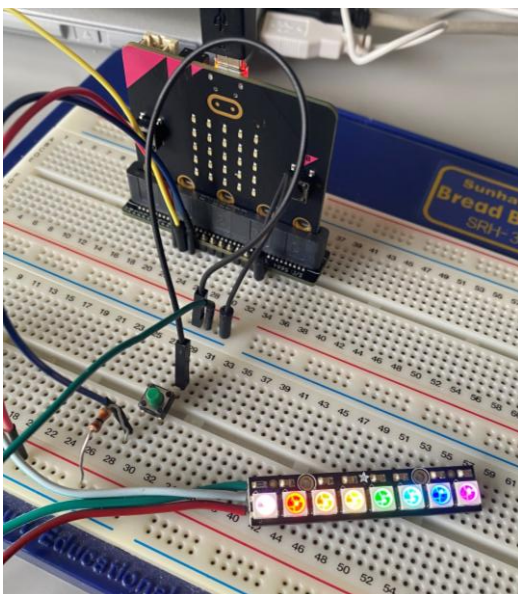


Figure 14. An example to turn LEDs on

Adafruit_NeoPixel class are respectively used to set the designated color information of each LED and make the LED emit. The task of turning the LEDs off uses a combination of the class's clear () and show () functions.

Figure 14 thus shows that each of the LEDs emits its designated color.

4.3 Kitronik's Zip Tile

Kitronik Zip Tile (8×8 Full Color Matrix) is shown in Figure 15. It is very easy to connect the Tile to micro:bit because the connector of the former is the same as that of the latter. A driven voltage/power for micro:bit with the downloaded program is supplied from not an A-micro B type USB cable but two batteries in the battery box linked to the Tile through JST connector. This is due to the fact that the battery box prevents an electrical overload from damaging micro:bit and causing it to unstably drive.

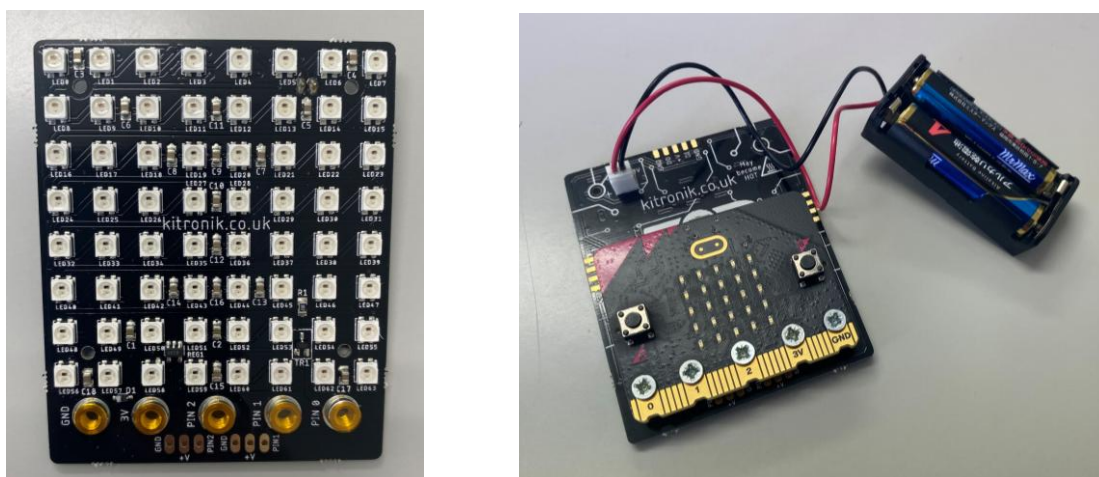


Figure 15. Zip Tile

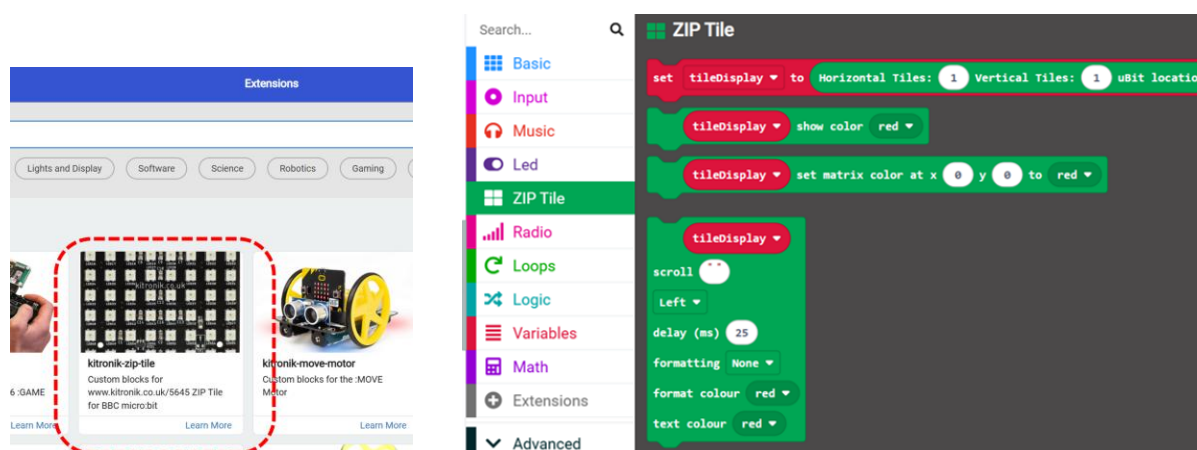


Figure 16. Adding category for Zip Tile to “toolbox” area

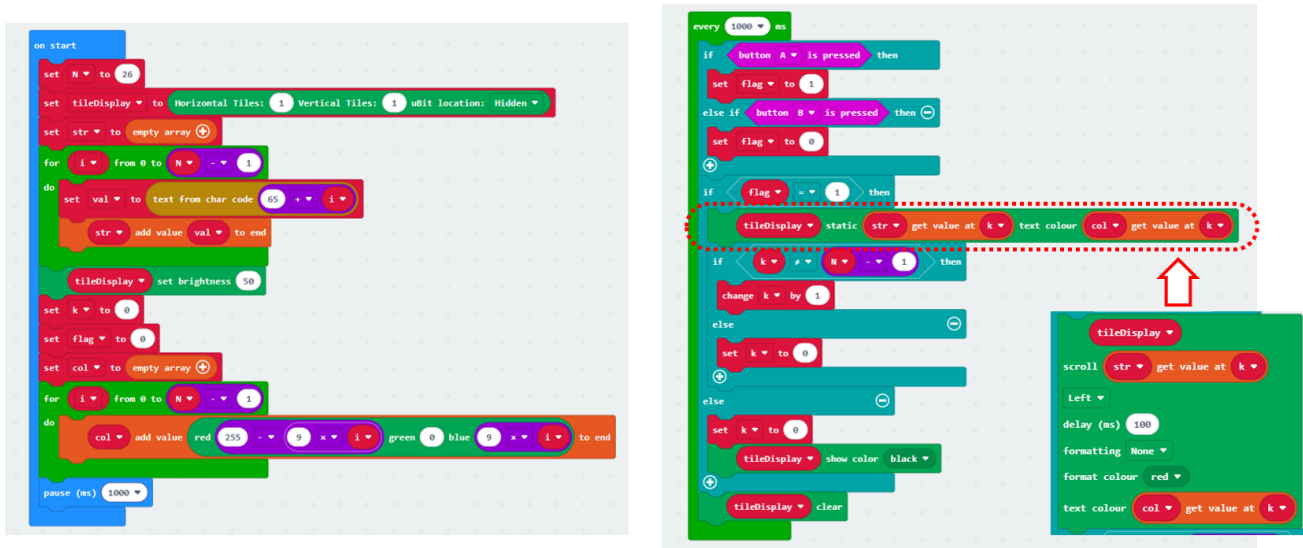
Figure 16 shows that the category “ZIP Tile” needs installing in “toolbox” via “Extensions”. Code 5 shows a sample program (created on “MakeCode”) that flashes and displays the 26 letters of the alphabet one by one sequentially on the Tile.

The variable “tileDisplay” in the “on start” block specifies a condition of the hardware configuration that means overlapping micro:bit with the back of the Tile. The letters of the alphabet are those of the uppercase (“A”–“Z”). These letters, however, are generated from the integer values (“65”–“90”) assigned in ASCII code using casting: integer-to-

character explicit conversion.

The Button A is used to display a letter whereas the Button B is done to hide it. It is enabled scrolling of the letters that the red dotted area in the figure is replaced with a “Scroll” block shown directly below. A letter is presented on Tile like a scroll while shifting the locations of turning on the LEDs from right to left every 100ms.

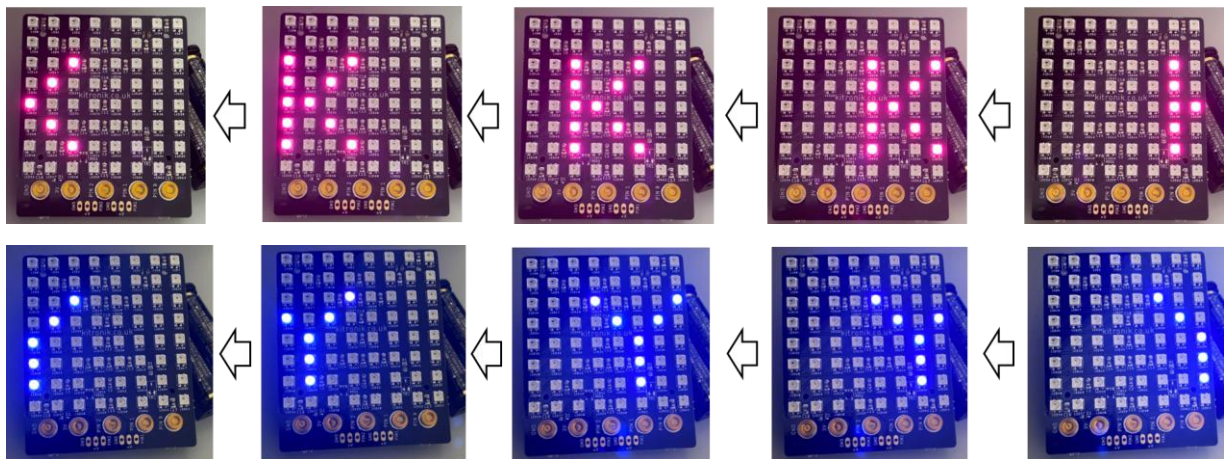
We show examples of flashing and scrolling displays in Figure 17.



Code 5. Program to operate Zip Tile (MakeCode)



(a) flashing displays (3 cases)



(b) scrolling displays (2 cases: left←right)

Figure 17. Examples for Zip Tile (makeCode)

On the other hand, programming on “Arduino IDE” has the following two notes: one is that there is not an easily available library for the Tile and the other we must prepare our own characters considering the structure of the Tile (8×8

LED matrix). Hence, we try to code a sample program based on the policy that the first solution is to substitute the header file “Adafruit_NeoPixel.h” used in section 4.2 and the next one to store self-made characters as two-dimensional information in three-dimensional arrays (the third dimension represents the number of characters). The reason for adopting the same library as Adafruit NeoPixel Stick is based on the similarity or compatibility of the Tile’s and the Stick’s interfaces to transfer data (signal).

A sample program uses 10 types of numerals “1”, “2”, ..., “9”, “0” as our-defined characters although it shows in Code 6. Emitting the LEDs means assigning digital values 1s to the points of the LEDs where locate along the brushstroke of a letter. The order of the 8×8 (= 64) LEDs ascends from top left to right bottom. The setting of a LED number, the first argument of Adafruit_NeoPixel.setPixelColor () function, is based on the ascending rule.

Our-defined function “Color ()” makes each letter light up in its own designated color (e.g., “1” is white, “2” red, ..., “9” purple, “0” dark pink [fuchsia]) using Adafruit_NeoPixel.Color() function although the detailed discussion of Cord 6 is omitted for the reason that it conforms to Code 4 shown in section 4.2.

Furthermore, we have confirmed that the scrolling display for the above numerals is stably and accurately done using our separately coded program.

Figure 18 is examples of flashing and scrolling displays using Code 6.

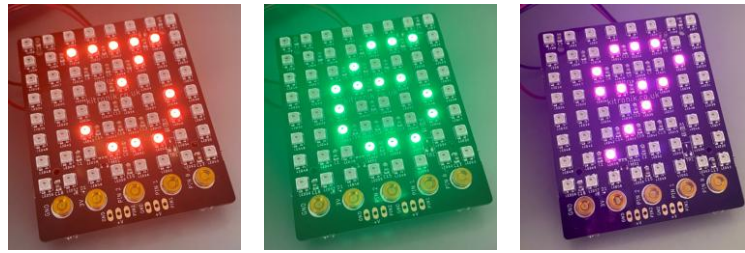
```
#include<Adafruit_NeoPixel.h>
#define N 8
#define PIN0 0
#define PIN5 5
#define num 10
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(N*N,PIN0, ...);
char chara[][N][N+1]={ {
"00000000", //1
"00001000",
"00011000",
"00001000",
"00001000",
"00001000",
"00011100",
"00000000"
},{
... abbreviates from '2' to '9' ...
},{
"00000000", //0
"00001100",
"00100010",
"01000010",
"01000010",
"01000100",
"00111000",
"00000000"
}
};
bool flag;
int k=0;
void setup() {
  pixels.begin();
  pixels.setBrightness(32);
  attachInterrupt(PIN5,Statement,FALLING);
  flag=false;
}
void loop() {
  if(flag==true){
    uint32_t clr= user-defined_Color();
    for(int row=0;row<N;row++){
      for(int col=0;col<N;col++){
        int pos=N*row+col;
        if(chara[k][row][col]=='1') pixels.setPixelColor(pos,clr);
        else pixels.setPixelColor(pos,pixels.Color(0,0,0));
      }
    }
    pixels.show();
    delay(1000);
    LEDOFF();
    delay(1000);
    if(k==num-1) k=0;
    else k++;
  } // flag
  else LEDOFF();
}
```

Code 6. Program to operate Zip Tile (Arduino IDE)

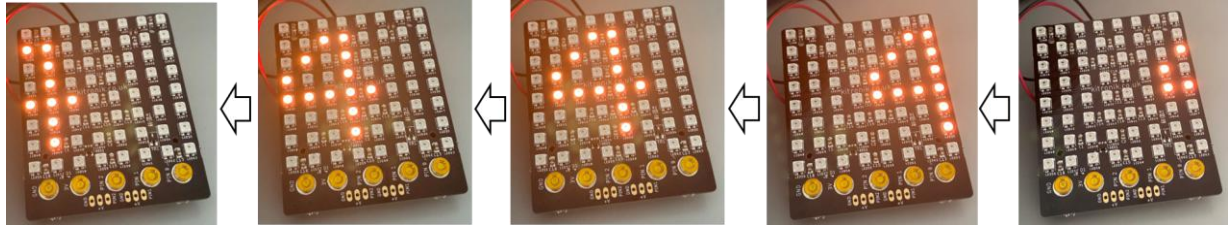
4.4 SeedStudio’s 4 Digits Display

SeedStudio 4 Digits Display shown in Figure 19 has four modules that draws one numeral using 7 LEDs, which has been frequently used as an item to show a ticking like a timer or a clock. The Display has SeedStudio Grove connector bundling four signal lines: “ V_{cc} ”, “GND”, “DIO”, “CLK”, which is easily connected to micro:bit using “Grove Shield” (Figure 8).

Code 7 is a sample program for countdown created after adding the category for the Display using “Extensions” in “MakeCode” (see Figure 20). The “set” block with variable “4digit” in the category “Grove” sets two pin “P0” and “P14”,



(a) flashing displays (3 cases)



(b) a scrolling display (left→right)

Figure 18. Examples for Zip Tile (Arduino IDE)

correspond to the clock signal “CLK” and the digital input-output “DIO”, respectively, based on one port of “Grove Shield” to use. The outline of the countdown follows: (1) the variable “start” stores the elapsed time since the start of the program using “running time(ms)” block in the “on start” one; (2) the countdown begins or progresses if a condition “ $val - start \geq 1000ms$ ” is true, although the variable “val” obtains a value using “running time(ms)” block whenever processing a loop.



Figure 19. Grove 4 Digit Display

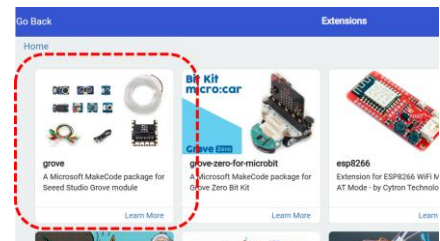
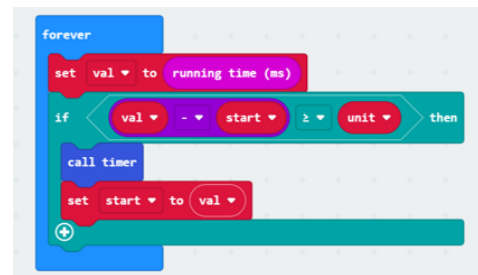
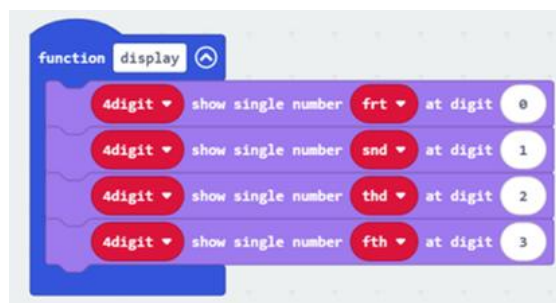


Figure 20. Adding category for Grove



Code 7. Program for countdown (loop processing : MakeCode)

Our-defined function “display” shows that each of 4 digits is associated with its own variable “frt”, “snd”, “thd” or “fth”. The stored values of the two variables “frt”



Code 7. Program for countdown(parameter settings : MakeCode)



Code 7. Program for countdown (calculation processing: MakeCode)

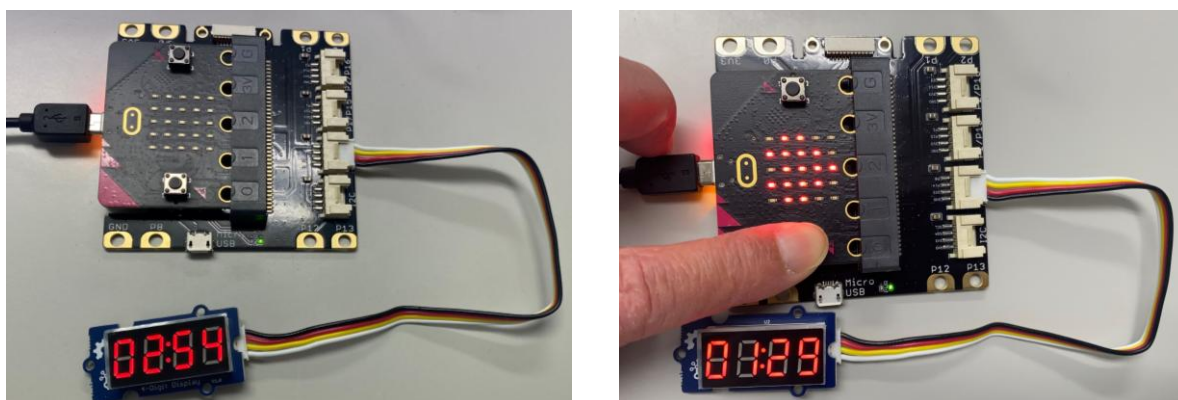


Figure 21. Examples of countdown and its pause

and “snd” represent higher and lower digits as a unit of “minutes” whereas that of “thd” or “fth” do them as one of “seconds”. Furthermore, as 1 set of these two variables, the values for higher and lower digits are respectively stored the quotient and the remainder when the counter variable “minutes” or “second” is divided by 10 using the operator “/” (see our-defined function “arrange” in the Code).

Figure 21 shows an example of the countdown as 3 minutes. The countdown pauses while the Button A is pressed whereas it begins again if the button is released.

We provide a program coded on “Arduino IDE” in Code 8. The program includes the header file “TM1637.h” after installing the library “Grove_4-Digit_Display” for the reason that the Display has an integrated circuit (IC) called “TM1637”. The TM1637 class can generate its instance using the pins for “CLK” and “DIO” signals. The display ()

```

#include<TM1637.h>
#include "nrf.h"
#define setttime 3
#define CLK 0
#define DIO 14
#define BtnA 5
TM1637 tm1637(CLK, DIO);
int8_t TimeDisp[] = {0x00, 0x00, 0x00, 0x00};
int minutes=settime;
unsigned int second;
int frt,snd,thd,fth;
bool flag;
extern "C" void TIMER2_IRQHandler(void) {
    NRF_TIMER2->EVENTS_COMPARE[0] = 0;
    if(flag==true){
        if(second==0) second=60;
        arrange();
        display();
        if(second==60) minutes--;
        if(minutes< 0) minutes=settime;
        else second--;
    }
}

void setup() {
    setup2();
    flag=false;
    pinMode(BtnA,INPUT_PULLUP);
    tm1637.init();
    tm1637.set(BRIGHT_TYPICAL);
    tm1637.point(POINT_ON);
    second=60;
    attachInterrupt(BtnA,Statement,FALLING);
}

```

Code 8. Program for countdown (first half: Arduino IDE)

function of the class allows the Display to show the values for 4 digits stored in int8_t array “TimeDisp[]”.

We use a clock system of “nRF52833”, the processor of micro:bit board, in the code. The header file “nrf.h” defines the structure “NRF_TIMER2”

required for timer controls. The use of the structure takes advantage of the follows: first, it drives on a thread that is independent of main one, next, it has the system that notifies the elapsed time through interrupt processing. This program of “Arduino IDE” will thus be more accurate than the above that of “MakeCode” in terms of a ticking.

The members of the structure “NRF_TIMER2” are initialized or set parameters in our-defined function “setup2”. Their details refer to the literature³⁾, and we especially explain the noteworthy items in them.

The processor “nRF52833”³⁾, which operates at a clock frequency of 64MHz, has a controller that provides multiple clock signals. Operation of timer needs peripheral devices such as prescaler, counter and comparator along with 16MHz clock signal. Prescaler is a divider that generates a signal with lower frequency than a clock signal. A sign to notify the elapsed time as a timer is an interrupt event raising when an accumulation (a stacking) value obtained from a count up task of the counter reaches a setting one for the comparator.

A parameter “PRESCALER” for the prescaler, one of the member variables of the structure, contributes to a division “ $2^{\text{PRESCALER}}$ ” in Code 8 (second half), which is $2^8 = 256$ when setting “PRESCALER” as 8. Though a fundamental mechanism of a timer depends on the count up task of the counter, the calculated value f_{clk}/F_d is equivalent to clock frequency f_{cnt} for the counter if setting clock frequency as f_{clk} and division as $F_d (= 2^{\text{PRESCALER}})$, example of which

```

void setup2(){
    NRF_TIMER2->TASKS_STOP = 1;
    NRF_TIMER2->TASKS_CLEAR = 1;
    NRF_TIMER2->MODE = TIMER_MODE_MODE_Timer;
    NRF_TIMER2->PRESCALER = 8;
    NRF_TIMER2->BITMODE =
        TIMER_BITMODE_BITMODE_16Bit;
    NRF_TIMER2->CC[0] = 62500;
    NRF_TIMER2->INTENSET =
        (TIMER_INTENSET_COMPARE0_Enabled
         << TIMER_INTENSET_COMPARE0_Pos);
    NRF_TIMER2->SHORTS =
        (TIMER_SHORTS_COMPARE0_CLEAR_Enabled
         << TIMER_SHORTS_COMPARE0_CLEAR_Pos);
    NVIC_SetPriority(TIMER2_IRQn, 3);
    NVIC_ClearPendingIRQ(TIMER2_IRQn);
    NVIC_EnableIRQ(TIMER2_IRQn);
    NRF_TIMER2->TASKS_START = 1;
}

void loop() {
}

void arrange(){
    frt=minutes/10;
    snd=minutes%10;
    if (second<60){
        thd=second/10;
        fth=second%10;
    }
    else{
        thd=0;
        fth=0;
    }
}

void display(){
    TimeDisp[0]= (int8_t)frt;
    TimeDisp[1]= (int8_t)snd;
    TimeDisp[2]= (int8_t)thd;
    TimeDisp[3]= (int8_t)fth;
    tm1637.display(TimeDisp);
}

void Statement(){
    flag=!flag;
}

```

Code 8. Program for countdown (second half: Arduino IDE)

$f_{cnt} = 16 \times 10^6 / 2^8 = 62,500\text{Hz}$ if $f_{clk} = 16\text{MHz}$. In this case, timing t_{itp} for interrupt event is represented in the following form if setting a counting value as N_{cnt} :

$$t_{itp} = \frac{N_{cnt}}{f_{cnt}} = \frac{N_{cnt} F_d}{f_{clk}} \quad (1)$$

Therefore, t_{itp} is temporarily equal to 1[s] if $N_{cnt} = f_{cnt}$, which makes it possible to raise interrupt every one second as timer.

The member “NRF_TIMER2” of the structure sets “16 bit (=2¹⁶ = 65536)”, maximum countable value of the counter and the member “INTENSET” does a defined-parameter that means to compare counter with comparator. The setting for the comparator is done using array variable CC[·], a register for it, which sets CC[·] as $f_{clk} = 62,500$. For reference, $t_{itp} = 0.5\text{s}$ if $\text{CC}[\cdot] = 62,500/2 = 31,250$.

Event handler “TIMER2_IRQHandler”, which is defined in assembler source file “gcc_startup_nrf52.S”, is used to implement the interrupt processing as shown in Code 8 (first half). The assembler source file, however, exists outside the area that the program shown in the Code can directly access. Thus, extern “C”, one of linkage specifiers, allows the program to call the handler according to the rules of C language.

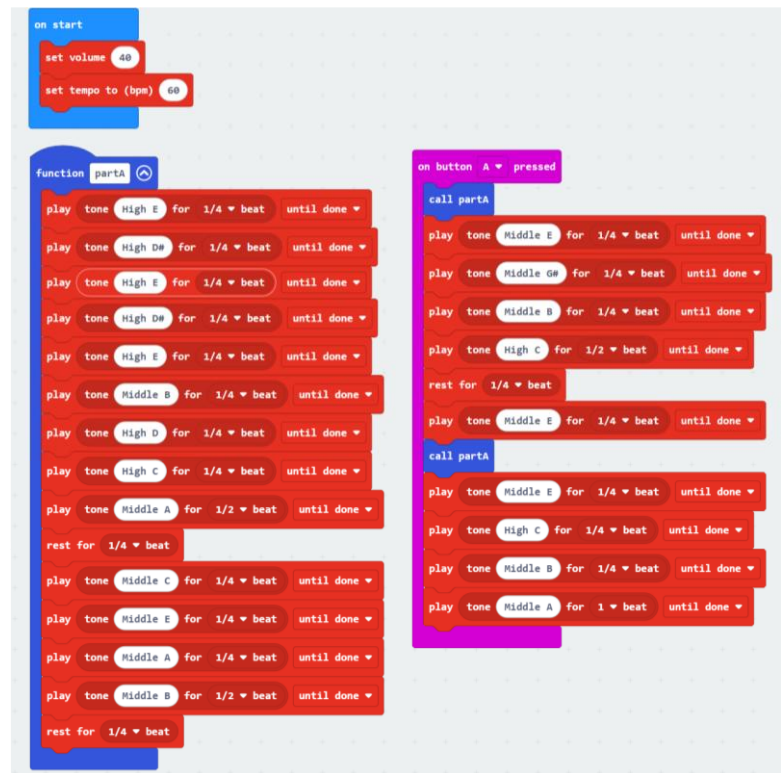
4.5 Speaker

As final example, we present a sample using speaker of micro:bit. Code 9 is a program to play part of music “Für Elise”: “Bagatelle 'Für Elise' a-moll WoO.59” by Ludwig van Beethoven⁴⁾ (see Appendix).

Volume of the speaker and tempo (beats per minute) of the music are set in “on start” block. The temporal length of one beat is 1s when setting tempo as 60. Thus, 0.5s and 0.25s if half of the beat and quarter of it.

“play” block, which is included in category “Music” and used to output to the speaker, designates music scale as tone and length of beats as that of tone.

“Middle” and “High” strings adding to musical scales of alphabetical notation means respectively the forth and the fifth octave areas.



Code 9. Program for music “Für Elise” (MakeCode)

The musical scales are gradually, regularly and ascendingly arranged notes according to their pitch (frequency). “MIDI (Musical Instruments Digital Interface)” is a unified standard that multiple linked electronic devices can mutually transfer and share data whereas “MIDI Tuning Standard” is a standard to clarify and control musical intervals (pitch intervals of two adjacent tones) of less than a semitone based on the protocol of MIDI. An octave area is divided into 12 musical scales by these standards, which known as “twelve equal temperament”. Each of the scales is named using a combination

of one alphabet and octave number. Furthermore, the semitone higher tone between two adjacent tones trails sharp sign “#” to the end of its name.

One musical scale used when tuning instruments is called “concert pitch”. A tone of 440Hz has been used as the concert pitch based on national or regional characteristics and their historical backgrounds. This tone is named “A4” combining alphabet “A” as starting point and “4” meaning that its tone is located in the fourth octave area. In addition, the tone “A4” has note “69” (MIDI note number) although an identified number named “note” is assigned to each of the scales.

Relating frequency f_X of musical scale X with its note number N_X represents as follows using the concert pitch “A4” and $N_X = N_{A4} + 12\log_2[f_X/f_{A4}]$:⁵⁾

$$f_X = f_{A4} \times 2^{\left[\frac{N_X - N_{A4}}{12}\right]} \quad (2)$$

Assuming parallelly f_X to be frequency of a scale included in the k octave area, f_X depends on the following form using difference d of two scales:

$$f_X = f_{A4} \times 2^{\left[(k-4) + \frac{d}{12}\right]} \quad (3)$$

Form (2) and (3) equations,

$$N_X = N_{A4} + 12(k - 4) + d \quad (4)$$

These equations can derive relation of f_X , N_X and d . Table 1, for reference, shows relationality of the scales in the fourth octave area.

We also code a program for music in “Arduino IDE”. The program, omitting is details, has the following noticeable points: (i) to include header file “MicrobitV2-HHS.h” after installing library “MicrosoftV2-HHS”; (ii) to calculate the frequencies of the scales in the fourth octave using (2)~(4) equations and then to do those of the scales in the other areas using bitwise shift operators (“<<” or “>>”); (iii) to output each of musical scales using analogWritePin() function with three arguments (pin number, temporal length of a scale and frequency of it).

scale X	scale difference d	note N_X	f_X [Hz]
C4	-9	60	262
C4#	-8	61	277
D4	-7	62	294
D4#	-6	63	311
E4	-5	64	330
F4	-4	65	349
F4#	-3	66	370
G4	-2	67	392
G4#	-1	68	415
A4	0	69	440
A4#	1	70	466
B4	2	71	494

Table 1. examples for musical scales (the fourth octave area)

Regarding (ii), we can obtain the frequency of the scale in the next higher octave if shifts that of the scale in the fourth octave to the left by one bit using the operator (“<<”) whereas can do the frequency of the scale in the next lower octave if does that of it to the right by one bit using the operator (“>>”) (e.g., “C5” if “C4” << 1 whereas “C3” if “C4” >> 1).

The multiple pieces of implemented music besides the above example show as follows⁶⁾:

1. James Lord Pierpont, “Jingle Bells”
2. Takeshi Inoue, “Japanese Children’s Song: Tulip”
3. Johann Sebastian Bach, “Air on the G String: Air from Orchestral Suite no.3 in D major, BWV 1068”
4. Johann Pachelbel, “Canon: Canon and Gigue in D major for three Violins and Basso continuo”⁷⁾

We implement these songs using musical scales in maximum of four octaves (from the third to the sixth) based on scores () of them. Note however that the songs are not necessarily implemented dependably by the scores but are arranged to suit our tastes, for the two follows: one is, for songs made of chords, to appropriately choose one of several scales, and

the other is to change the temporal length of the beat for each of the musical scales. These attempts will lead to creating a covered melody with more groove such as an upbeat music or a relaxed feel as a healing one.

5 Conclusion

We have regularly held a programming seminar with microcontrollers since April 2024. The seminar is not necessary one of methods to develop experts or creators for programming (as engineers) although has the widely range of topics and tasks to acquire knowledge and skills for it from basics to advanced. Considering the recently advance of information technology society, the purposes and policies of the seminar are as follows: first is to provide its members with the fun and the excitement of programming using simple and compact items like microcontrollers, next to familiarize them with a concept of programming, an algorithmic idea and a logical one, and last to reduce the threshold of programming that beginners and non-experts hold, which would help increase the number of the member. Those of the seminar thus lead to expand not only engineer's skills for programming but the number of people interested in it.

Figure 22 shows scenes from the seminar using micro:bit. We slowly and step by step tackle each task in the seminar because many of us are not programming engineers. The examples of task are to create a simple operating program for novelty modules when getting them and to share and solve tasks that each of us sets for ourselves.

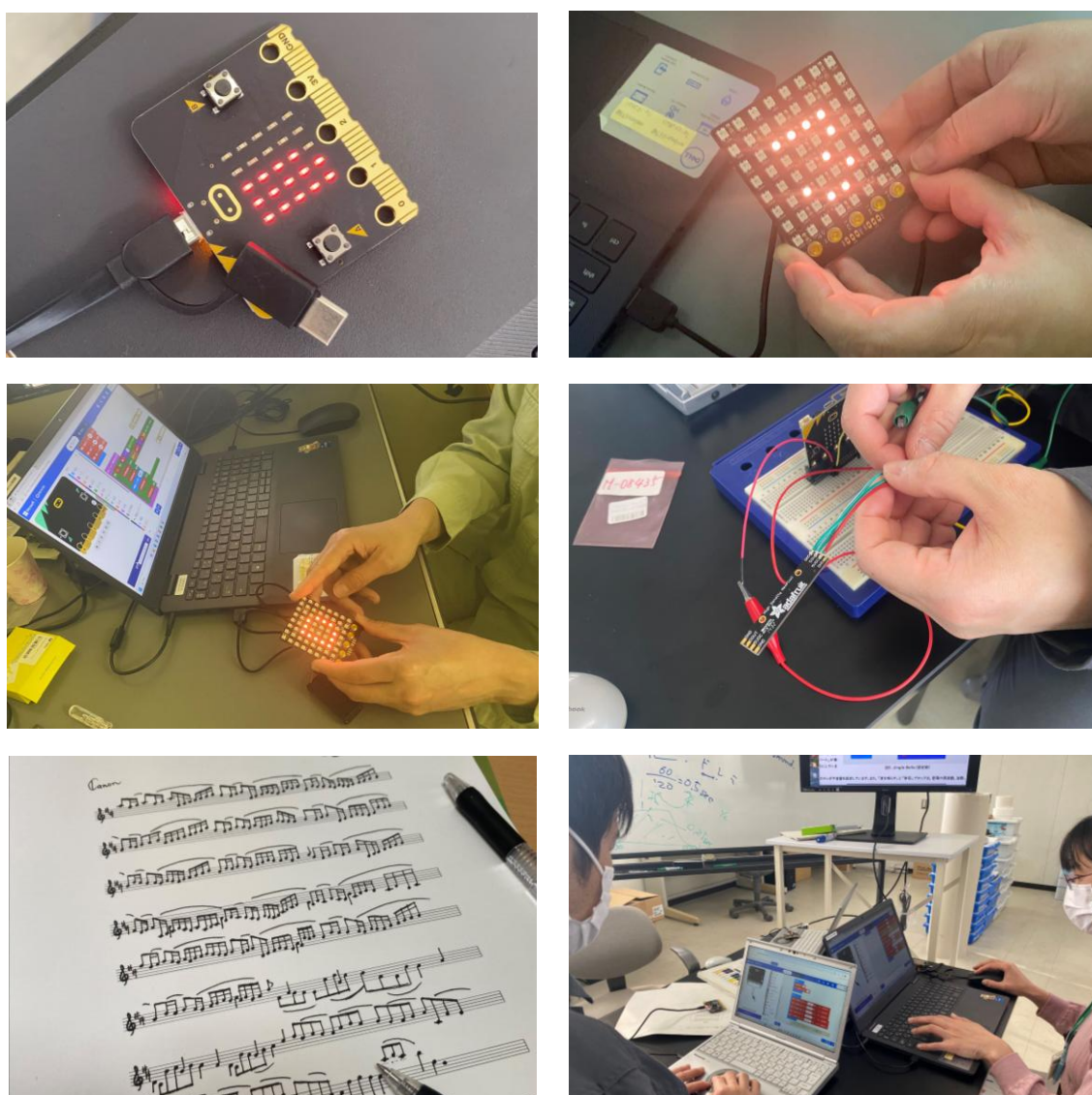


Figure 22. Programming seminar

In closing, this paper has been constructed based on worthwhile or useful comments and advice as well as proofreading of its English document provided by Dr. Shota Nakashima and Dr. Ryosuke Adachi in Graduate School of Sciences and Technology for Innovation, Yamaguchi University. We are really grateful to the two scientists who gladly accept to review it.

References

- 1) Arduino IDE download Site, <https://www.arduino.cc/en/software>
- 2) Arduino Documentation (Libraries/Language Reference), <https://docs.arduino.cc/libraries/> (or [/language-reference/](https://docs.arduino.cc/language-reference/))
- 3) nRF52833 Product Specification v1.7, NORDIC SEMICONDUCTOR, nRF52833_PS_v1.7.pdf, pp.672-684, https://docs.nordicsemi.com/bundle/ps_nrf52833/page/keyfeatures_html5.html
- 4) FÜR ELISE OP.173 L. van Beethoven, MUSIC FOR PIANO NO.2, Zen-On Music Company Limited
- 5) Meinard Müller, Fundamentals of Music Processing -Audio, Analysis, Algorithms, Applications-, Springer, pp.13-24, 2015
- 6) For instance, <https://www.youtube.com/watch?v=QzBTv9QL6qU>,
<https://www.youtube.com/watch?v=PZzLkVjrDkk>, https://www.youtube.com/watch?v=ZHCjU_rcQno
- 7) KANON Johann Pachelbel, MUSIC FOR PIANO NO.448, Zen-On Music Company Limited, 1987

Appendix

Figure 23 shows music score of “Für Elise” handwritten by S. Yamano, one of the authors. Code 9 is created based on the first eight bars (within the dotted line in the figure) of this score.



Figure 23. handwritten music score: “Für Elise”