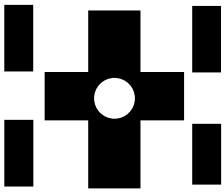


Modulio™ Modular I/O system

Modulio™ is a trademark of PastelMagic , Inc.

Ver1.0

Title Index

1. What is Modulio 5
2. I2C Bus 7
3. RJ-11 telephone splitter 8
4. Modulio Connector Pin Assignments 9
5. Example System of Modulio 10
6. Modulio Electrical Specification 11
7. Modulio PCB Design proposal 14
8. Modulio bus switch board 15
9. Reference design of bus switch board 17
10. Modulio Software 20
10.1. Example of accessing device 21
10.2. MI2CADRS structure 23
10.3. Modulio common library functions 24
10.3.1. void MI2C_Start(void) Initialize hardware 24
10.3.2. void MI2C_Setup(MI2CADRS *adrs, UINT8 bswadrs, UINT8 bswch, UINT8 devadrs, UINT8 option) Initialize Modulio management structure 25
10.3.3. UINT8 MI2C_WriteBytes(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length) 26
10.3.4. UINT8 MI2C_ReadBytes(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length) 26
10.3.5. UINT8 MI2C_WriteByte(MI2CADRS *adrs, UINT16 Reg, UINT8 Data) 27
10.3.6. UINT8 MI2C_ReadByte(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data) 27
10.3.7. void MI2C_Waitms(UINT16 delay) 27
10.4. Modulio Microcontroller dependent library 28
10.4.1. void MI2C_Startup(void) 28
10.4.2. UINT8 MI2C_fSendStart(UINT8 bSlaveAddr, UINT8 fRW) 28
10.4.3. UINT8 MI2C_fSendRepStart(UINT8 bSlaveAddr, UINT8 fRW) 29
10.4.4. void MI2C_SendStop(void) 29
10.4.5. UINT8 MI2C_fWrite(UINT8 bData) 30
10.4.6. UINT8 MI2C_bRead(UINT8 fACK) 30
10.4.7. void MI2C_WaitMills(UINT16 delay) 30
11. Recommended style of device driver 32
11.1. Device management structure 32
11.2. Naming of the structure 33

12. XXX_Setup() function (constructor) 33
12.1. Start() function 33

Figure 1: Example system using I2C bus system 7
Figure 2:RJ11 telephone splitter 8
Figure 3: I2C bus signal and power line assignment on Modulio.....9
Figure 4: Example system of temperature and pressure sensing equipment.....10
Figure 5: The standard way to use 3.3V device (recommended). 11
Figure 6: All devices work at +3.3V 12
Figure 7: Mixed voltage system(3.3V bus system) 13
Figure 8:Modulio board design proposal.....14
Figure 9:Mix voltage system. 16
Figure 10: Bus switch board schematic 18
Figure 11:Modulio bus switch board.....19
Figure 12: Modulio software hierarchy 20
Figure 13: Example program using Modulio 21
Figure 14:Example system.....22
Figure 15:The example program using Modulio 23
Figure 16: Example structure of Modulio device driver 32

1. What is Modulio

Modulio is flexible generic I/O sysytem based on I2C-bus and RJ11(6P4C) modular cable. It is so easy for you to plug-in/out the cable, to change connection, and to add/remove modules, thus Modulio is very convenient for prototyping and education.

Modulio libraries provides common APIs that does not depend on microcontroller/platform. These are written in 'C' language and has a hierarchical structure. Thus, it is easy to move Modulio to different platform and user applications can work on various microprocessor/platforms only to re-compile. If you want to port it to your favorite microcontroller, all you have to do is to impliment basic I2C bus access functions. Actually, Modulio has been ported to Cypress PSoC1 (CY8C29466, CY8C27143) Microchip PIC24F08 & ATMega328P, RaspberryPi , RaspberryPi Pico

Modulio supports up to eight PCA9546A 4channnel I2C bus switch. The bus switch allows multiple devices with same I2C bus address to be used.

Moudlio also suggests standard PCB sizes.

Figure 1 is a environment measurement system consists of 2xLCD panel, pressure sensor, acceleration sensor, full-color LED and MPU (Raspberrypi Pico)

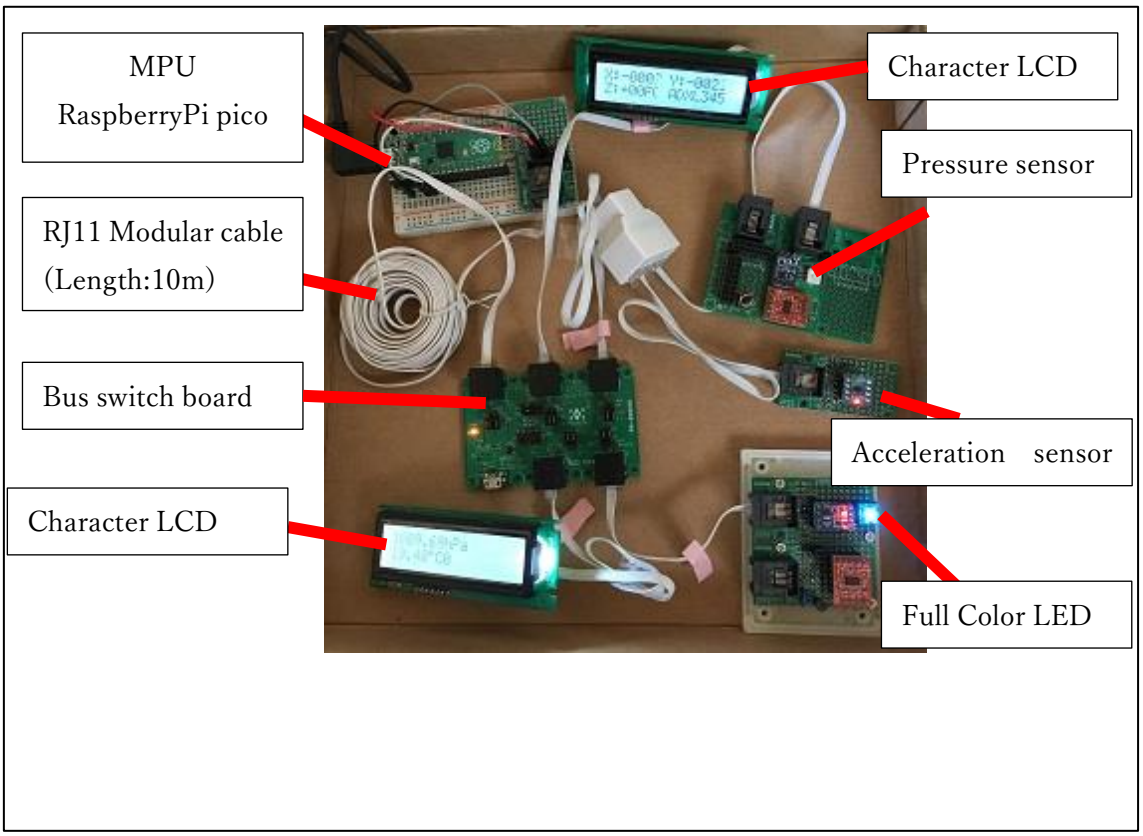


Figure 1:Environment measurement system using Modulio

2. I2C Bus

I2C Bus is a two-wire synchronous serial bus invented by Philips Semiconductors (known today as NXP Semiconductors).

The physical layer of I2C bus consists of the serial clock (SCL) and serial data (SDA) line (See Figure 2). Both signals are driven by open-drain/open collector driver, and should be connected to VDD through a pull-up resistor.

Typical I2C bus system has one controller device and one or more target devices (called as 'master device' and 'slave device' in previous documents).

Each target devices have a unique address. Generally, it is 7-bit length. When reading/writing I2C target device, the controller uses its address.

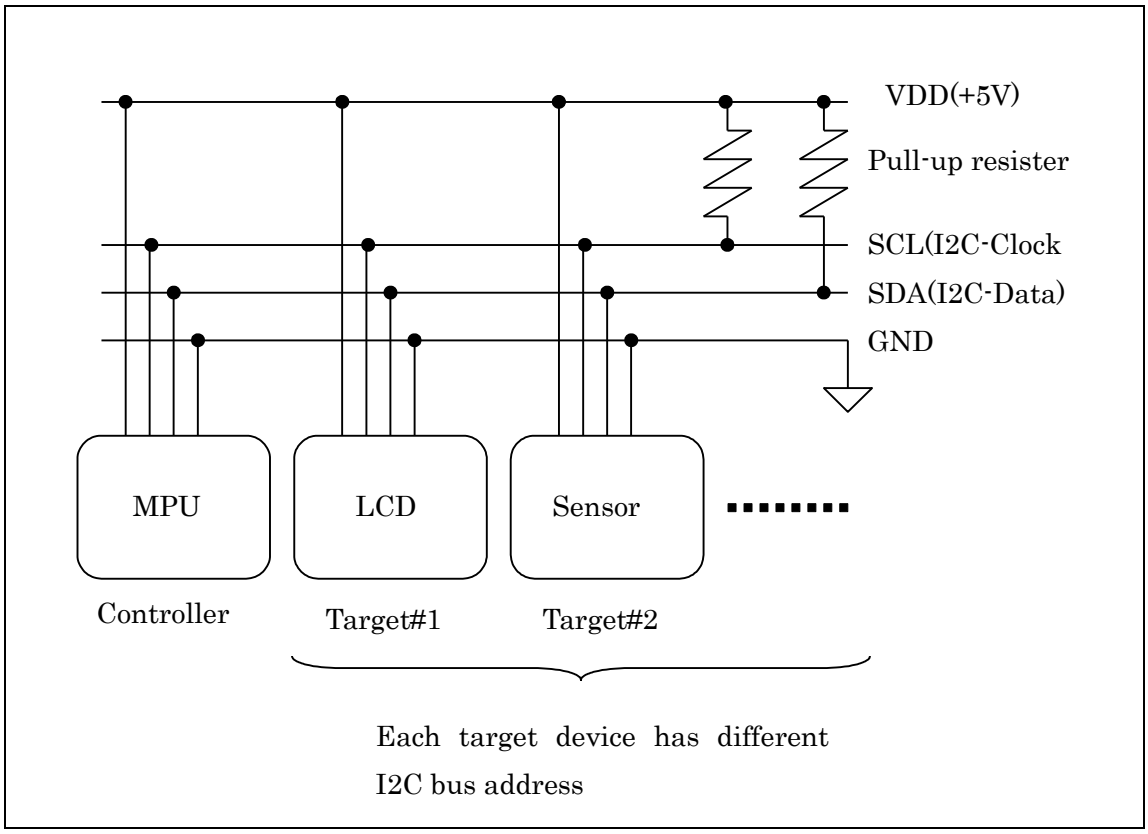


Figure 2: Example system using I2C bus system

3. RJ-11 telephone splitter

I2C bus uses 4 lines – two bus signal lines (SCL/SDA) and two power lines (VDD/GND). The RJ-11 (6P4C) connector well known as telephone connector also has 4 signal lines. If more than one device is used on a telephone line, a telephone splitter can be used.

The pin connection of telephone splitter is shown as Figure 3. The RJ-11 connector has 6-positon 4-contacts. Pin number 2 to 4 are used, pin 1 and 6 are N.C(Not Connected). All connectors are connected in parallel.

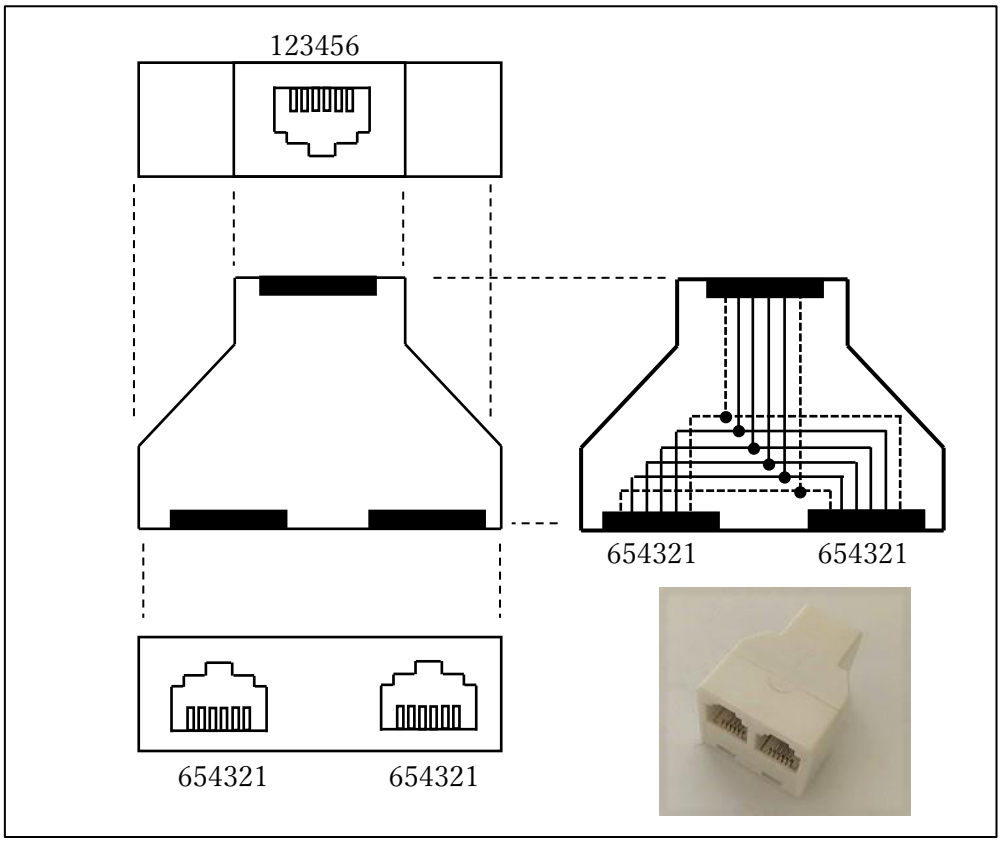


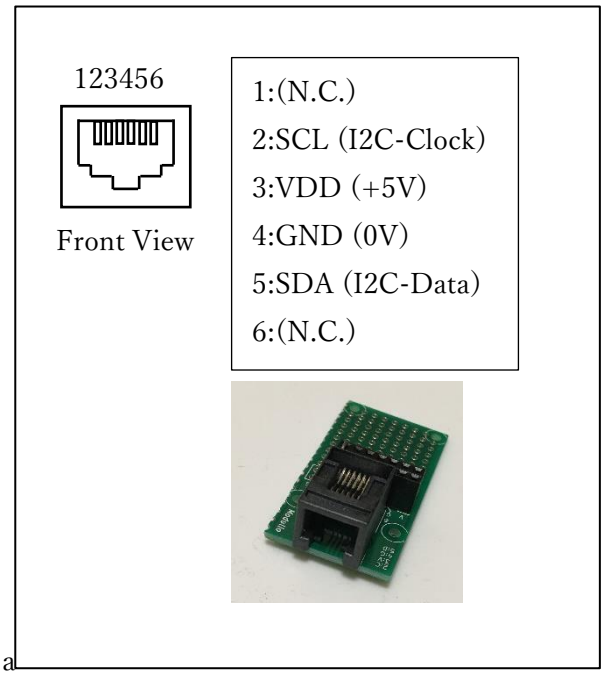
Figure 3:RJ11 telephone splitter

4. Modulio Connector Pin Assignments

I2C bus signals are assigned to RJ11 connector as shown in Figure 4. The inner pin (pin 3 and 4) are power lines and outer pin (pin 2 and 5) are I2C clock (SCL) and I2C data (SDA). This assignment conforms to the recommendations of the I2C bus standards.

VDD power voltage is +5V. In case of using 3.3V devices, use 5V to 3.3V voltage regulator for power supply and level conversion device for I2C bus signals.

The PCA9546A I2C bus switch device which is a standard bus switch device of Modulio has signal level conversion function, thus when using it, there is no needs to use a level conversion IC.



a

Figure 4: I2C bus signal and power line assignment on Modulio

5. Example System of Modulio

Figure 5 shows an example of a temperature and pressure measuring equipment. This equipment consists of three modules: a microcontroller module, a character LCD panel and a sensor module.

An I2C signals of microcontroller are drawn to RJ11 connector. The I2C bus is connected to a RJ11 telephone splitter.

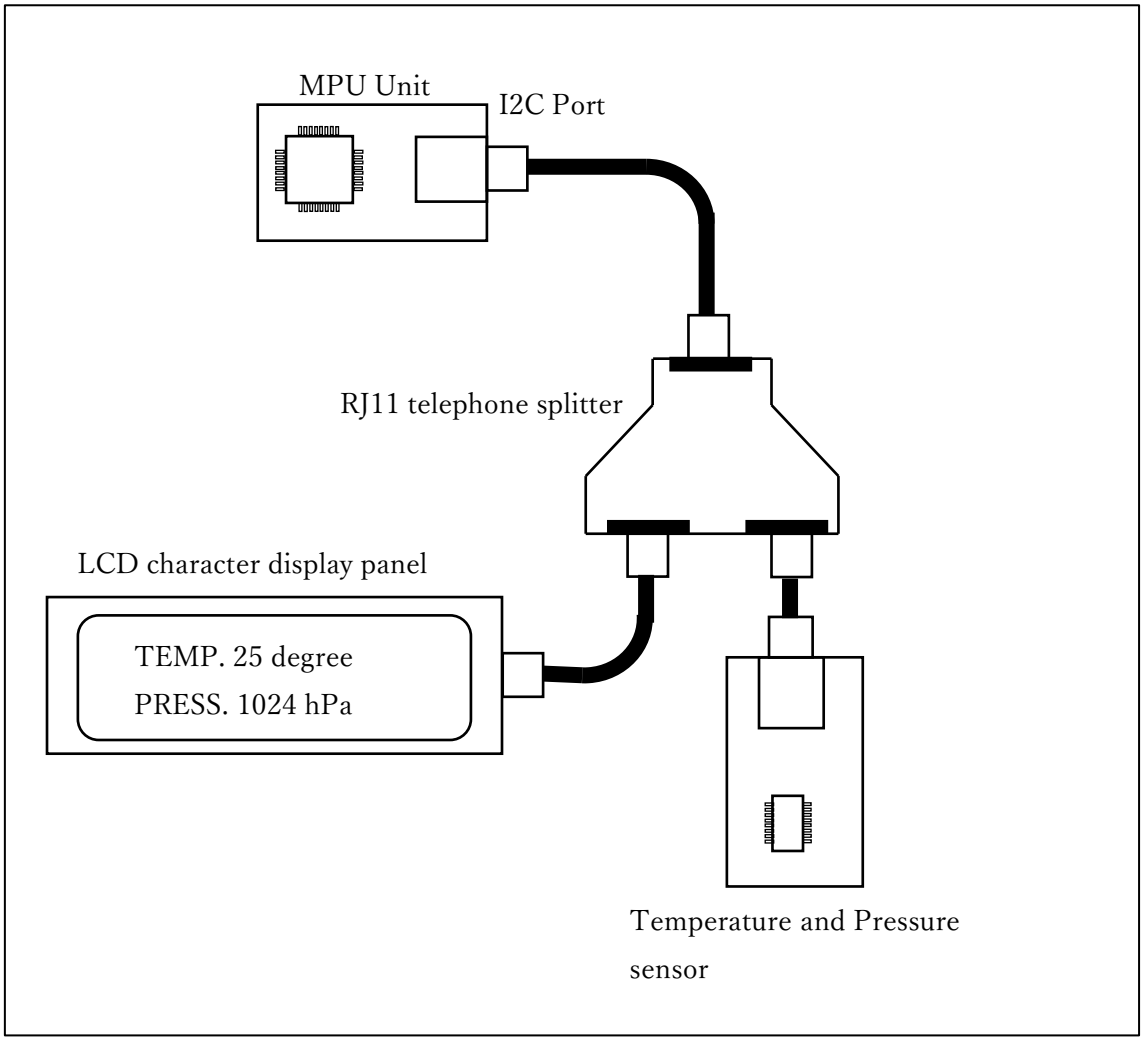


Figure 5: Example system of temperature and pressure sensing equipment

6. Modulio Electrical Specification

Modulio recommended specification is as follows.

・VDD Voltage：+5V

・Pull-up resister：4.7k-ohm

・Bus speed：50kbps(*)

*: It can be set to 100kbps or more. Though, ACM1602 character display panel that is easily available was not work properly at bus speed higher than 50kbps.

Figure 6 shows you the way to use the I2C device worked at 3.3V. It is recommended to use 5V to 3.3V regulator and I2C bus level converter (ex. NXP PCA9306) .

It is possible to set VDD voltage to 3.3V if all devices work at +3.3V (See Figure 7). And in some case, it is also possible to connect pull-up resister with +3.3V in mixed voltage (+5V and +3.3V) system(Figure 8). In this case, note that the minimum VIH (Input high level voltage) of signal (SCL/SDA) level of device is typically 0.7xVDD, thus, minimum VIH of the device worked at +5V is +3.5V (=5.0x0.7), it is close to 3.3V.

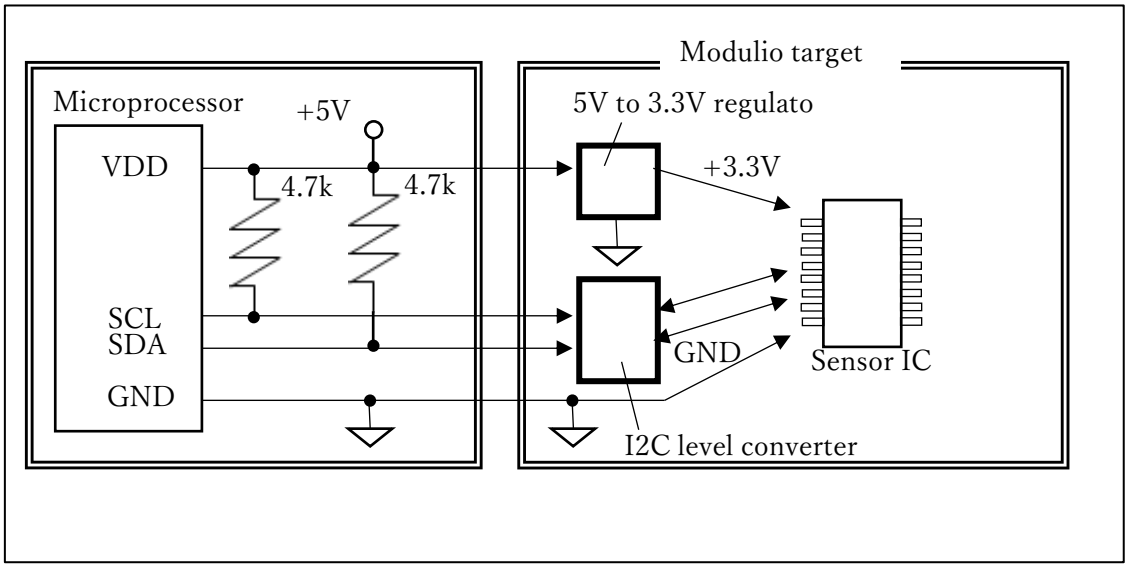


Figure 6: The standard way to use 3.3V device (recommended).

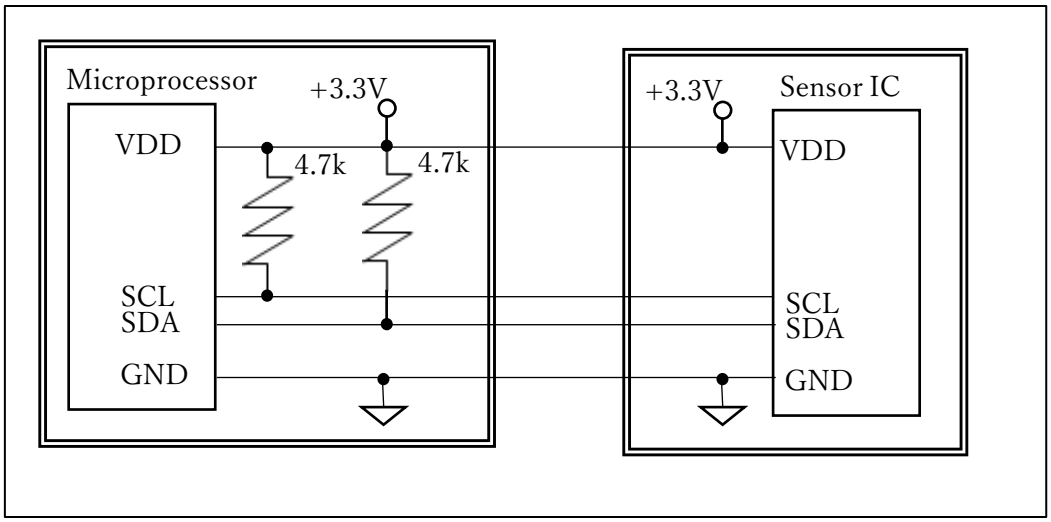


Figure 7: All devices work at +3.3V

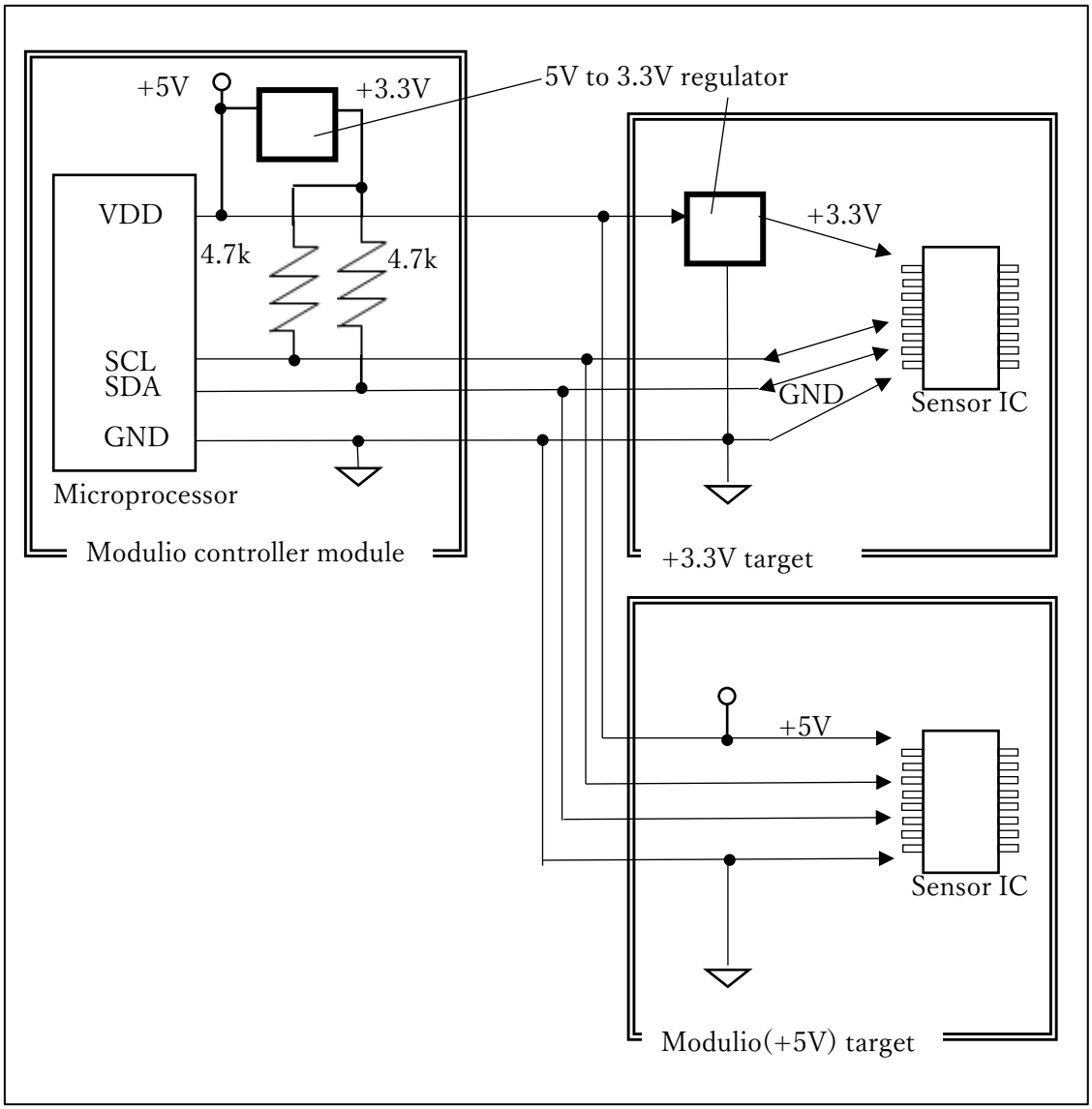


Figure 8: Mixed voltage system(3.3V bus system)

7. Modulio PCB Design proposal

Figure 9 shows a proposed PCB design. If you design larger size PCB, integer multiple size of this is recommended. If you are annoying to decide PCB size, this may be a good suggestion.

Off course, it does not restrict you from designing your own size PCB.

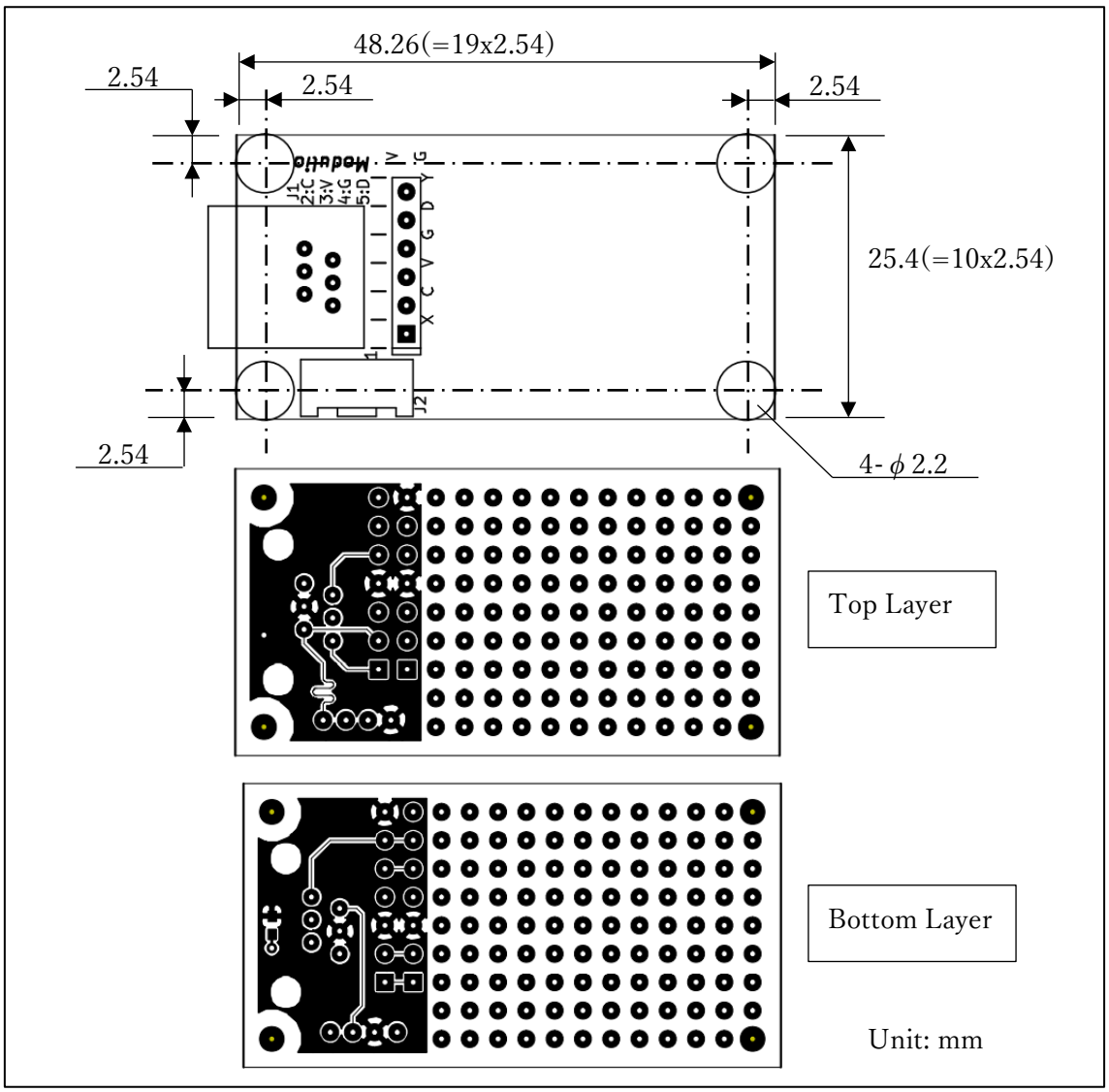


Figure 9:Modulio board design proposal

8. Modulio bus switch board

Modulio supports NXP PCA9546 as standard bus switch IC. PCA9546 is a 4-channnel I2C bus switch with signal level conversion. It is useful for mixed voltage system.

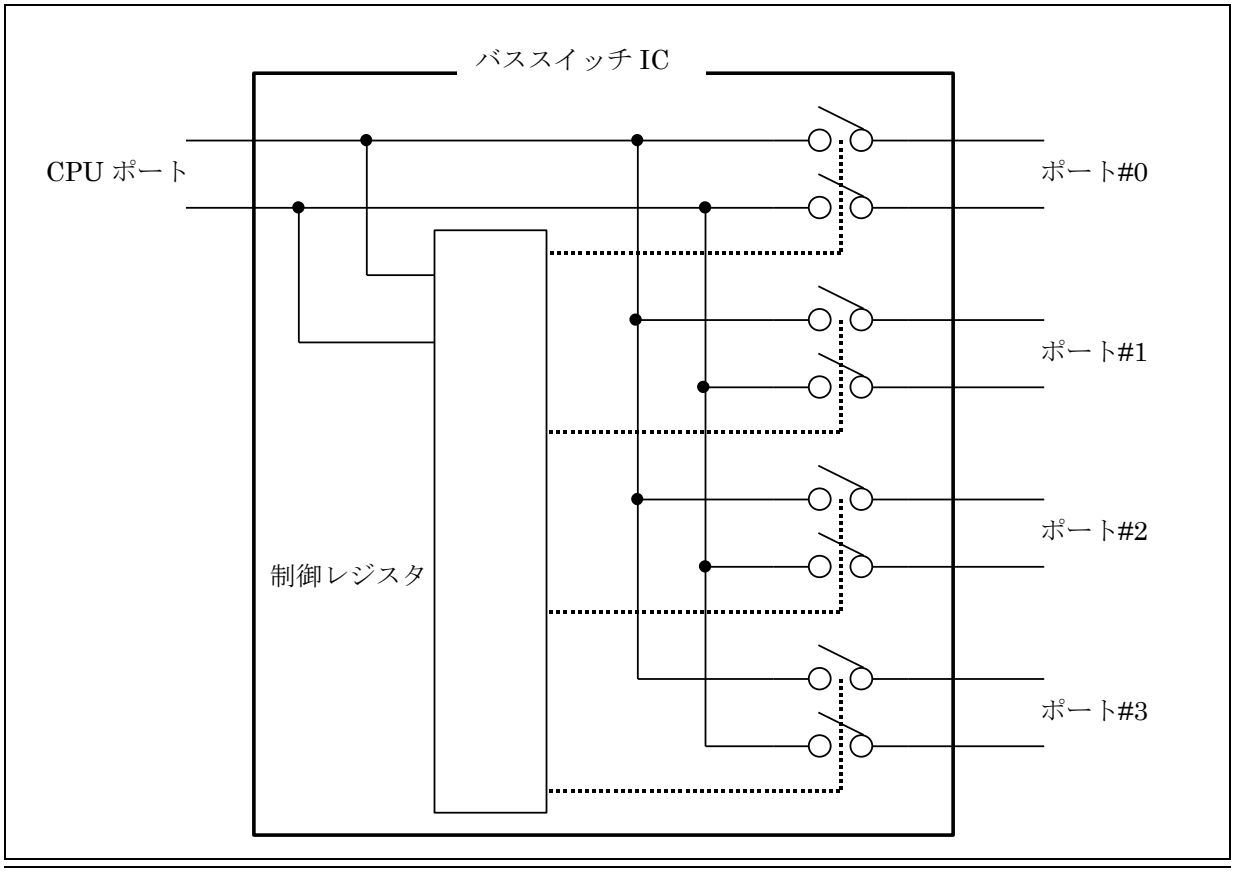


Figure 10 shows a mix voltage system using bus switch (PCA9632). PCA9632 has 1 controller-port and 4-target port. PCA9632 has its own control register accessed via I2C bus control internal switch. Though PCA9632 operates at +5V, all of ports are 5V tolerant. Thus, it can connect directly to device operating at +5V.

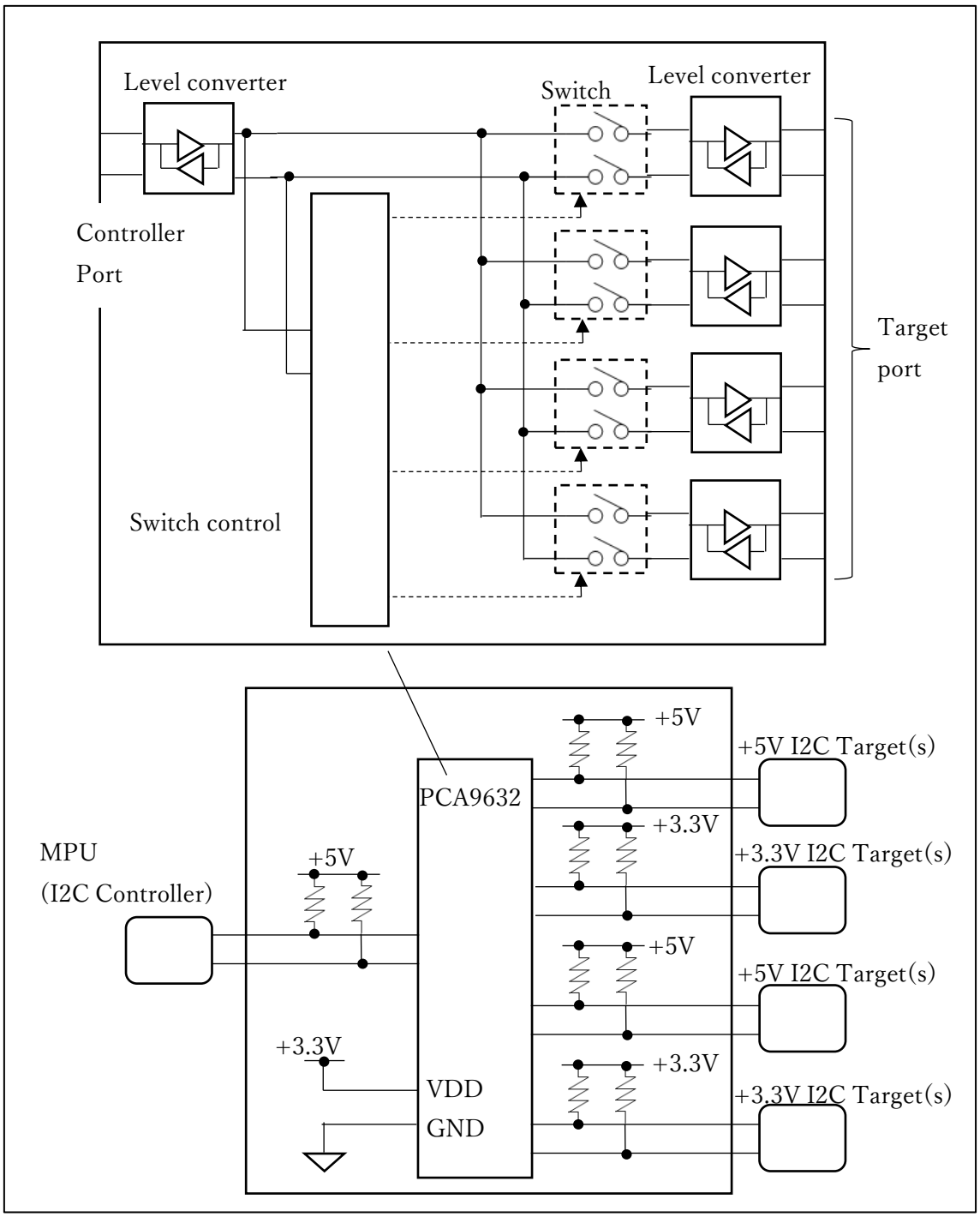


Figure 10:Mix voltage system.

9. Reference design of bus switch board

Figure 10 shows a reference of bus switch board circuit. The micro-USB connector is used to supply +5V power. The pull-up voltage of each I2C port (1-controller port, 4-target port) can be selected from +5V, +3.3V or none (open). This is a simple reference, so there are no protection circuit such as reverse current protection, surge protection, etc.

Similarly, Figure 12 shows a reference design for PCB layout. It is tree times size of the standard Modulio PCB.

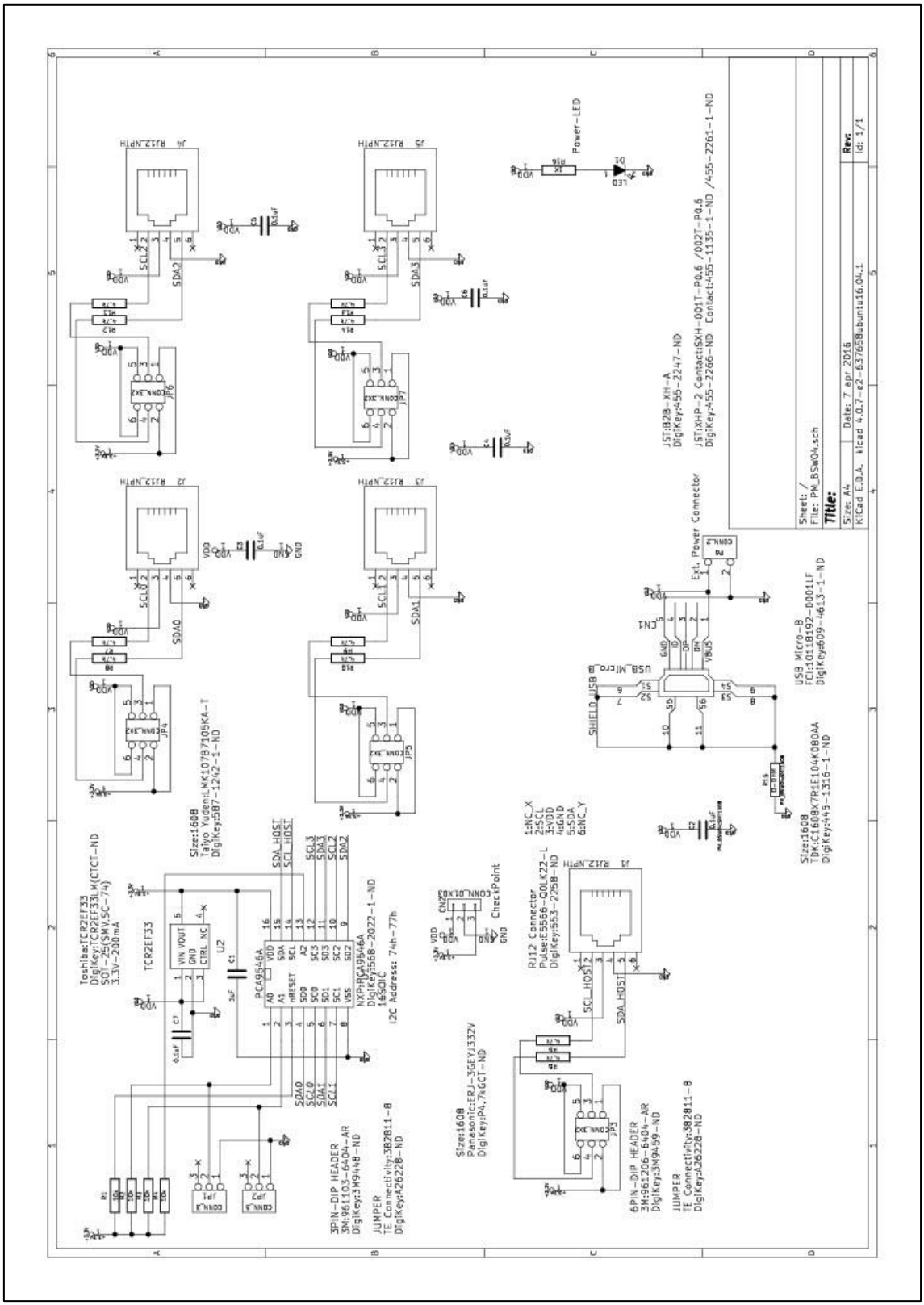


Figure 11: Bus switch board schematic

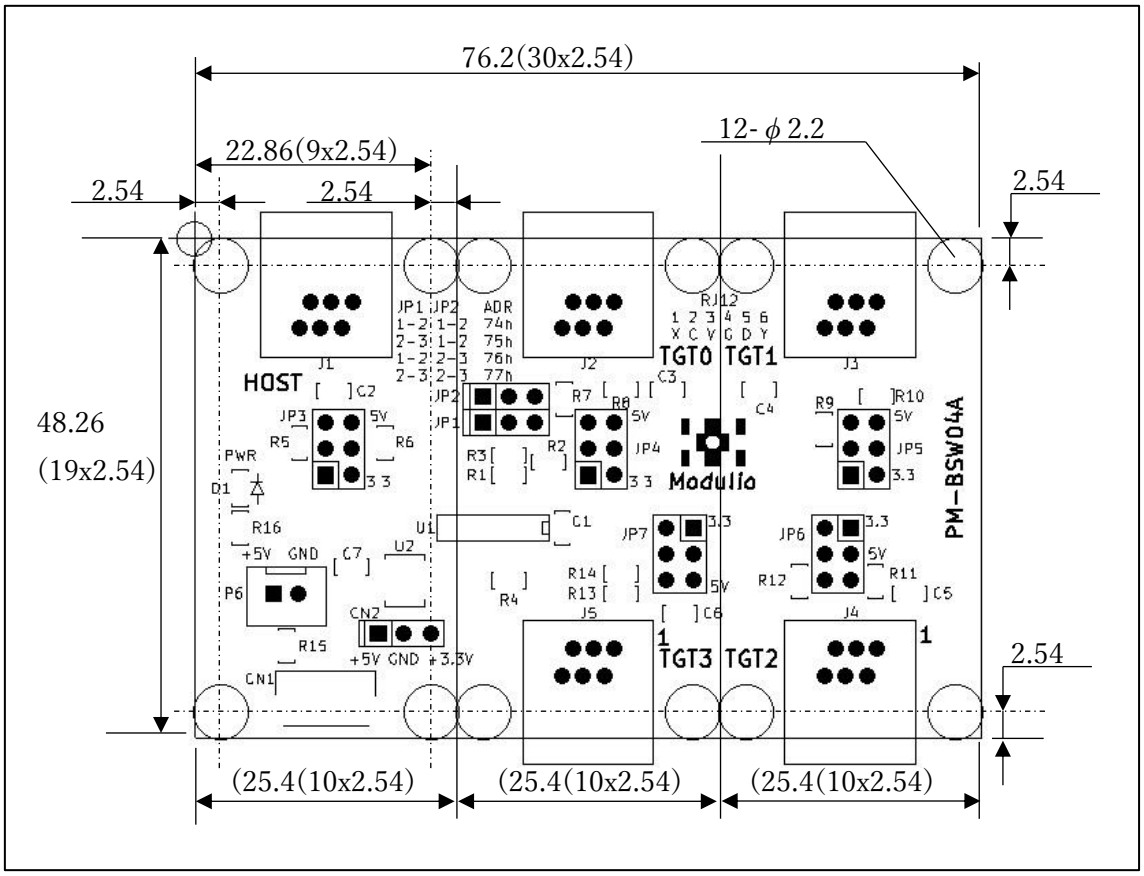


Figure 12:Modulio bus switch board

10. Modulio Software

Figure 13 show hierarchy of Modulio software. Modulio software is composed of three categories. I/O device-independent library (common library), I/O device-dependent library (device driver) and Microcontroller dependent library (HAL library).

User application uses only the I/O device-dependent library and MI2C_Start() function that requests hardware initialization.

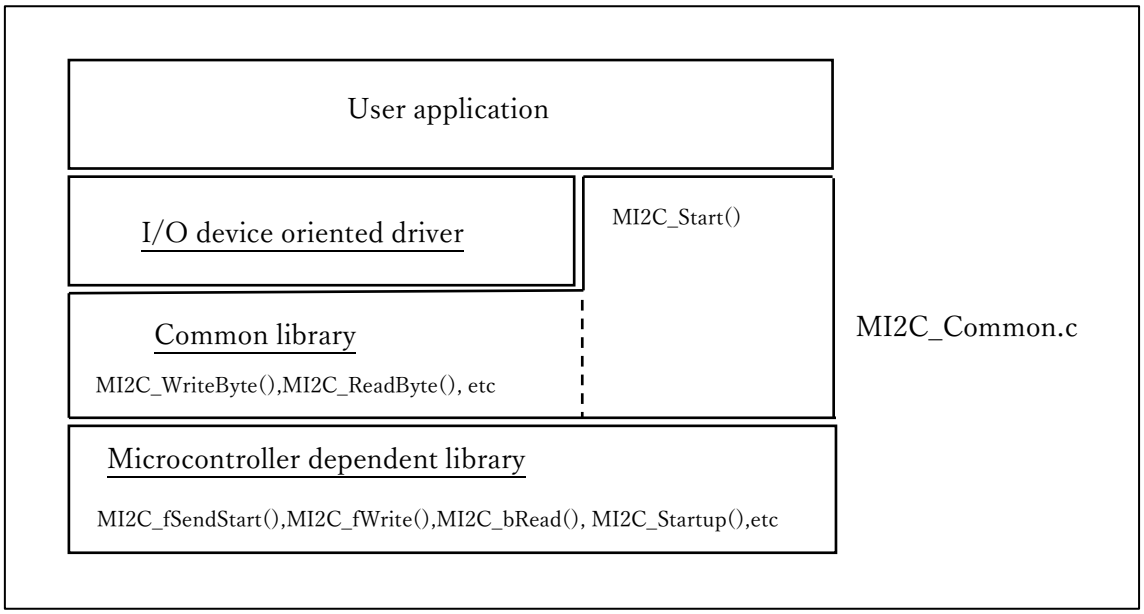


Figure 13: Modulio software hierarchy

10.1. Example of accessing device.

Figure 16 is a example program using Modulio library and Figure 15 shows example system. Modulio library provides 'class' (in C++) like way to access device. The 'class' is similar to 'struct' in the C language, though struct can't have 'this pointer'. Thus, the first parameter of all functions of device driver is a pointer to the device object that contain the device-independent MI2CADRS structure (described as follow) and device-dependent functions and data.

ACM1602_Setup()/LPS25H_setup() function initialize per-device structure. The second and third parameter specifies the I2C address and port of bus switch.

For instance, ACM1602_Setup(&lcd2, BSW_ADRS_DEFAULT, 3, ACM1602_ADRS0, 0) shows ACM1602 LCD panel is connected to bus switch with default I2C address and to the port#3 of bus switch.

```
#include "Modulio.h"
#include "ACM1602.h"
#include "LPS25H.h"
……
ACM1602 lcd1, lcd2; // 16x2 character LCD display panel
LPS25H     psense;   // Pressure & temperature sensor
……

void main(void)
{
   MI2C_Start(); // Initialize hardware

   //---- Initialize I/O devices
   ACM1602_Setup(&lcd1, BSW_ADRS_DEFAULT, 0, ACM1602_ADRS0, 0);
   ACM1602_Setup(&lcd2, BSW_ADRS_DEFAULT, 3, ACM1602_ADRS0, 0);
   LPS25H_Setup(&psense, BSW_ADRS_DEFAULT, 1, LPS25H_ADRS0, 0);
……
   //--- Start working I/O evices
   lcd1.Start(&lcd1);
   lcd2.Start(&lcd2);
   psense.Start(&psense);
……
   psense.ReadPress(&psense);       // read pressure register value
   psense.wConvPress(&psense);    // data conversion
   lcd1.Position(&lcd1, 0, 2);          // Set cursor of LCD1 to (2,0) Y=0, X=2
   lcd2.Position(&lcd2, 0, 0);          // Set cursor of LCD2 to (0,0) Y=0, X=0
   sprintf(s," %04d.%02dhPa",
       (psense.PRESS[0] << 8)|(psense.PRESS[1]), psense.PRESS[2]);
   Lcd1.PrString(&lcd1, s);             // Display to LCD1
   sprintf(s," %02d.%02d%cC", psense.TEMP[0], psense.TEMP[1], 0xdf);
   lcd2.PrString(&lcd2, s);               //
……
}
```

Figure 14: Example program using Modulio

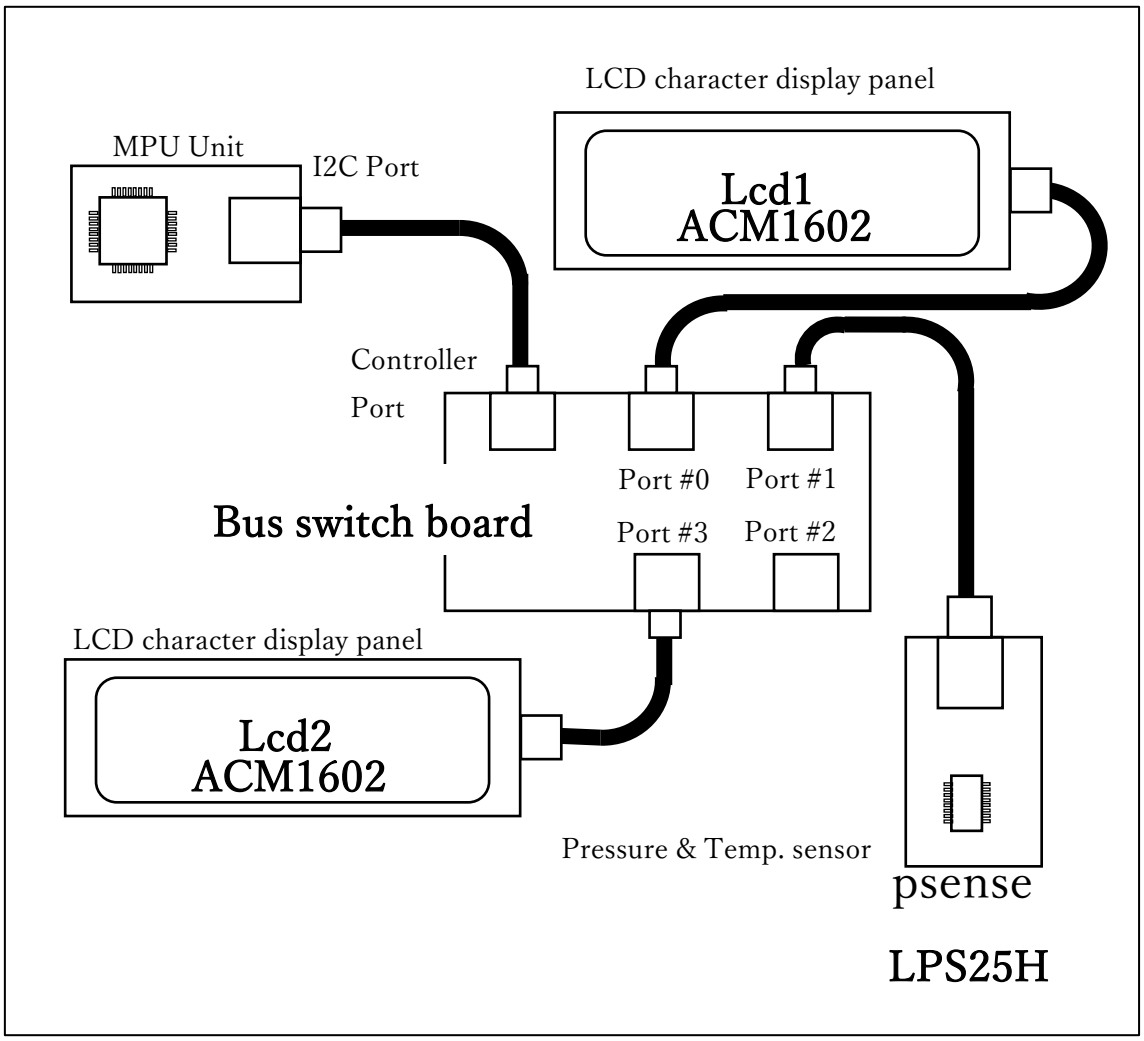


Figure 15:Example system

10.2. MI2CADRS structure

Modulio manage basic information of each device by per-device MI2CADRS structure. This structure contains device-independent (common) functions and data and is initialized in the MI2C_Setup() function.

```
---------------------------------------
typedef struct MI2CADRS_STRUC {
  UINT8   BSW_ADRS;          // The address of bus switch that connected to the device.
  UINT8   BSW_CH;            // The channel number of bus switch that connected to the device.
  UINT8   DEV_ADRS;          // The address of the device
  UINT8   OPTION_1;          // The optional data. Reserved for future use.
                             // The pointer to function that write multiple bytes to I2C bus.
  UINT8   (*WriteBytes)(struct MI2CADRS_STRUC *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length);
                             // The pointer to function that read multiple bytes to I2C bus.
  UINT8   (*WriteByte)(struct MI2CADRS_STRUC *adrs, UINT16 Reg, UINT8 Dat);
                             // The pointer to function that write a byte to I2C bus.
  UINT8   (*ReadBytes) (struct MI2CADRS_STRUC *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length);
                             // The pointer to function that read a byte to I2C bus.
  UINT8   (*ReadByte) (struct MI2CADRS_STRUC *adrs, UINT16 Reg, UINT8 *Data);
} MI2CADRS;
```

Figure 16:The example program using Modulio

10.3. Modulio common library functions

Modulio common library functions are as follows. UINT8 means unsigned 8-bit length value, UINT16 means unsigned 16-bit length value.

With the exception of M2C_Start()function, almost all API of common library functions are prepared for writing device driver.

```
---------------------------------------
void MI2C_Start(void)
void    MI2C_Setup(MI2CADRS *adrs, UINT8 bswadrs, UINT8 bswch, UINT8 devadrs, UINT8 option)
UINT8 MI2C_WriteBytes(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length)
UINT8 MI2C_ReadBytes(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length)
UINT8 MI2C_WriteByte(MI2CADRS *adrs, UINT16 Reg, UINT8 Data)
UINT8 MI2C_ReadByte(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data)
void MI2C_Waitms(UINT16 delay)
---------------------------------------
```

10.3.1. void MI2C_Start(void)

Initialize hardware

Input: None

ReturnValue: None

MI2C_Start() function is used to initialize hardware set transfer speed and GPIO pins, etc.

In User application, call MI2C_Start() function first of all other Modulio functions (includes device driver).

MI2C_Start()function is defined as follow in the common library (Modulio_Common.c),

```
---------------------------------------
void MI2C_Start(void)
{
        MI2C_Startup();
}
---------------------------------------
```

MI2C_Start() calls M2C_Startup(). That is written by the person who ported Modulio for each microprocessor.

10.3.2. void MI2C_Setup(MI2CADRS *adrs, UINT8 bswadrs, UINT8 bswch, UINT8 devadrs, UINT8 option)

Initialize Modulio management structure.

Initializes the MI2CADRS structure that is associated with the device.

Input:

MI2CADRS *adrs

The pointer to MI2CADRS structure that specifies the device.

UINT8 bswadrs

I2C bus address of bus switch that is connected with the device. If the device is connected to controller (MPU), use BSW_ADRS_NONE for this field.

UINT8 bswch

The channel number of I2C bus switch which connected to device.

UINT8 devadrs

The I2C address of the device.

Return value:

None

10.3.3. UINT8 MI2C_WriteBytes(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length)

Write multiple data bytes to the specified register of the device.

Input:

MI2CADRS *adrs

The address of MI2CADRS structure that specifies the device.

UINT16 Reg

The register address of the device specified by “adrs” field.

UINT8 *Data

The address of the buffer that contains data to be written to the device.

UINT8 Length

Specifies the data length to be written in byte.

Retrun value

MI2C_NAKslave (0x00) Error occurred on I2C bus

MI2C_ACKslave (0x01) Transfer complete with no error(s).

10.3.4. UINT8 MI2C_ReadBytes(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data, UINT8 Length)

Input:

MI2CADRS *adrs

The address of MI2CADRS structure that specifies the device.

UINT16 Reg

The register address of the device specified by “adrs” field.

UINT8 *Data

The address of the buffer in which store data from device.

UINT8 Length

Specifies the data length to be read in byte.

Retrun value

MI2C_NAKslave (0x00) Error occurred on I2C bus

MI2C_ACKslave (0x01) Transfer complete with no error(s).

10.3.5. UINT8 MI2C_WriteByte(MI2CADRS *adrs, UINT16 Reg, UINT8 Data)

Write a byte to specified device.

Input:

MI2CADRS *adrs

The address of MI2CADRS structure that specifies the device.

UINT16 Reg

The register address of the device specified by “adrs” field.

UINT8 Data

A data byte to be written.

Retrun value

MI2C_NAKslave (0x00) Error occurred on I2C bus

MI2C_ACKslave (0x01) Transfer complete with no error(s).

10.3.6. UINT8 MI2C_ReadByte(MI2CADRS *adrs, UINT16 Reg, UINT8 *Data)

Read a data from specified device.

Input:

MI2CADRS *adrs

The address of MI2CADRS structure that specifies the device.

UINT16 Reg

The register address of the device specified by “adrs” field.

UINT8 Data

A data byte buffer in which store data to be read.

Retrun value

MI2C_NAKslave (0x00) Error occurred on I2C bus

MI2C_ACKslave (0x01) Transfer complete with no error(s).

10.3.7. void MI2C_Waitms(UINT16 delay)

Wait (pause) the specified number of milliseconds.

Input:

UINT16 delay

Specified the time to wait(pause) in milliseconds. 1000 means 1sec.

Return value:
None.

10.4. Modulio Microcontroller dependent library

Microcontroller-dependent library contains functions that perform the basic operation of I2C bus. This library includes the following functions.

In case you want to use Modulio on your favorite microcontroller, all you have to do is to port these functions.

---

void MI2C_Startup(void)
UINT8 MI2C_fSendStart(UINT8 bSlaveAddr, UINT8 fRW)
UINT8 MI2C_fSendRepStart(UINT8 bSlaveAddr, UINT8 fRW)
void MI2C_SendStop(void)
UINT8 MI2C_fWrite(UINT8 bData)
UINT8 MI2C_bRead(UINT8 fACK)
void MI2C_WaitMills(UINT16 delay)

10.4.1. void MI2C_Startup(void)

Initialize hardware (I2C controller, GPIO Pin, etc).

Input:
None
Return value:
None

10.4.2. UINT8 MI2C_fSendStart(UINT8 bSlaveAddr, UINT8 fRW)

Perform start condition on I2C bus.

Input:

UINT8 bSlaveAddr

The I2C bus address of the device.

UINT8 fRW

MI2C_WRITE (0x00) Perform write transfer in following transaction(s)

MI2C_READ (0x01) Perform read transfer in following transaction(s)

Return value:

MI2C_NAKslave (0x00) Error occurred on I2C bus

MI2C_ACKslave (0x01) Transfer complete with no error(s).

10.4.3. UINT8 MI2C_fSendRepStart(UINT8 bSlaveAddr, UINT8 fRW)

Perform repeated start condition on I2C bus.

Input:

UINT8 bSlaveAddr

The I2C bus address of the device.

UINT8 fRW

MI2C_WRITE (0x00) Perform write transfer in following transaction(s)

MI2C_READ (0x01) Perform read transfer in following transaction(s)

Return value:

MI2C_NAKslave (0x00) Error occurred on I2C bus

MI2C_ACKslave (0x01) Transfer complete with no error(s).

10.4.4. void MI2C_SendStop(void)

Perform stop condition on the I2C bus. The data transfer is completed and the I2C bus goes idle.

Input:

None

Return value:

None

10.4.5. UINT8 MI2C_fWrite(UINT8 bData)

Write a byte data to I2C bus.

Input:

UINT8 bData data to be written to target device

Return value:

MI2C_NAKslave (0x00) Error occurred on I2C bus

MI2C_ACKslave (0x01) Transfer complete with no error(s).

10.4.6. UINT8 MI2C_bRead(UINT8 fACK)

Read a byte from I2C bus.

Input:

UINT8 fACK

MI2C_fACK (0x01) send ACK after read a byte from I2C device.

MI2C_fNAK (0x00) send NAK after read a byte (indicate last byte to be read) from I2C device.

Return value:

Data read from I2C device

10.4.7. void MI2C_WaitMills(UINT16 delay)

Wait (pause) the specified number of milliseconds. This function is called from MI2C_Waitms() function.

Input:

UINT16 delay

Specified the time to wait(pause) in milliseconds. 1000 means 1sec.

Return value:

None.

11. Recommended style of device driver

Though Modulio does not restrict coding style of device driver, the following coding styles are recommended.

11.1. Device management structure

Figure 17 is a example of device management structure of device driver. MI2CADRS structure is placed at the top of device management structure. Device-specific data/function pointers are placed following the MI2CADRS structure.

Each object of the structure is associated with physical device. In case of using multiple identical devices, you declare multiple objects of same structure. Each structure has unique MI2CADRS information that is associated with device, and first parameter of device-specific functions is a device management structure itself (similar to 'this' pointer in C++).

In this way, each function can access to associated device.

```
typedef struct LPS25H_STRUC {
  MI2CADRS madrs;          ——————  Device-independent structure
  UINT8      ID;
  UINT8      STS;
  UINT8      PRESS[3];     }       Device-specific data
  UINT8     TEMP[2];
  UINT8     STATUS;

  UINT8     (*Start)(struct LPS25H_STRUC *p);
  UINT8     (*ReadPress)(struct LPS25H_STRUC *p);
  Void      (*wConvPress)(struct LPS25H_STRUC *p);   }   Pointers to device-
  UINT8     (*ReadTemp)(struct LPS25H_STRUC *p);         specific function
  Void      (*wConvTemp)(struct LPS25H_STRUC *p);
  UINT8     (*ReadStatus)(struct LPS25H_STRUC *p);
} LPS25H;
```

Figure 17: Example structure of Modulio device driver

11.2. Naming of the structure

It is recommended that the name of the structure be the same as the device number or product name.

12. XXX_Setup() function (constructor)

Unlike 'class' of C++, the structure does not have mechanism of constructor, thus the function that setup function pointer of structure should place outside of the structure.
It is recommended that the function name be XXX_Start() ('XXX' is same name as the device management structure).
It is recommended that the first parameter of Setup()function is pointer to device management structure, second and third parameter be the address and port of bus switch to which the device that associated with the first parameter is connected..

12.1. Start() function

Start() function used to activate (start working) the device. There are many devices that do not need start operation. Even in such case, it is recommended to prepare empty Start() function for unified style.

Revision History

Version	Description	Date
1.0	First release	3/12 2022