YKAMELEAN24

Native Firmware Reference

March, 2018

YEPKiT

Contents

1. Overview

Microcontrollers (MCU) are an integrating piece of many electronic systems providing the ability to implement complex control systems and interface with a wide range of peripherals. They provide a high flexibility, cost efficiency and footprint savings.

Nonetheless starting the development of a MCU based system from “green field” may be a daunting task due to the learning curve associated with programming it. This is the main reason why the “accelerated” development MCU boards and tools proliferate nowadays, boards like arduino and the tools associated with it brought the power of MCU environments to the general public and leveraged the development of new an exiting DIY projects and systems.

We use extensively MCU development boards on our development process, testing development hypothesis and for implementing “one-shot” projects/systems. With time the features we looked for become more apparent and the high level design for our “ideal” development MCU board was born.

The main features we wanted in a MCU development board were:

1. Out of the box functionality that allowed us to plug and play with it.
2. Quick and easy programming capability.
3. Uncompromising in terms of evolution to production level.

YKAMELEAN24 was developed to provide the above features to developers. It provides a solid functional baseline to build upon and enrich.

The native firmware provides a fully functional baseline that can be used in a plug and play manner together with a PC with an USB connecting to provide out-of-the-box capability for the user to interact with the low level electronic interfaces using the MCU IOs and interfaces/peripherals. A software for Linux and Windows is available for download at the YKAMELEAN24 product page which implements the commands used to control the board, peripherals and add-on modules. Add-on modules implement the interface with additional boards like sensors and actuators[1], more details on add-on modules are presented in section 5 at page 7.

The native firmware is structured to facilitate the inclusion of additional code from the user and is suitable to serve a baseline to be built upon. It will take care of initialization the basic configuration of the YKAMELEAN24 board (for example the USB interface configuration) saving time to the developer. Section 3 on page 3 provides further insight on how user code can be added to the baseline firmware to expand it.

A USB bootloader is used to enable direct loading of the firmware to the board through USB, this way a specific hardware programmer is not required facilitating the development process. To load the firmware through USB we provide a PC software application available for download at the product page on yepkit.com. For details refer to the appendix B on page 15.

While having a way to quickly implement “something” is useful we also wanted to do it with a production level orientation where code efficiency, minimize rework, ability to use all of the MCU features, to incorporate native Manufacturer libraries and tools is important. By using the native C programming language, libraries and compilers we ensure uncompromising development from inception to production. For details on the firmware build process and tools check appendix A on page 15.

[1] Check the product page for the most up to date list of supported add-on modules/boards

2. Using the native firmware as a baseline to build upon

A flow to build you own version of the firmware, customized for your specific application, can be the following.

1. Add your code to the native firmware;
2. Compile and build using MPLABX IDE and XC16 compiler;
3. Upload the firmware hex file to YKAMELEAN24 board using the board BootloaderApp for PC[2].

2.1. Adding user code to the native firmware

While you are free to change the native firmware any way you want it, we recommend for user code to be included using function calls in the functions in `src/app/user_app.c`. On the native firmware, in the `user_app.c` source file you can find the following function implementation.

```
void user_app(void)
{
        //place here user application front-controller functions
}
```

This function is called in every cycle of the main program eternal loop.

Now let's assume that we wanted to implement an oven temperature controller that periodically polls a temperature sensor to find out the temperature of the oven and based on that runs a PID (Proportional Integral Deferential) algorithm to switch a relay which controls the oven power. To do this we could add a function called, for example, `oven_temperature_control_tasks()` which implements all the control. Now the `user_app()` function looks like the following.

```
void user_app(void)
{
        //place here user application front-controller functions
        oven_temperature_control_tasks();
}
```

3. Firmware source code project structure

The firmware source code project uses the following structure.

```
/ykamelean24_firmware/
      + src/
          + app/
          + inc/
          + USB/
          + peripheral/
              + io/
              + lcd/
```

[2]For details on how to use the BootloaderApp refer to appendix B

```
+ modules/
    + ykemb/
    + ykhat/
    + ts100/
    + ykur/
```

src consolidates all the project source code.

app contains the “applicational” components and user code.

USB has the USB communication library.

peripheral consolidates the MCU peripheral libraries.

modules consolidates the add-on modules libraries.

4. Peripherals

The native firmware provides peripheral interface libraries. To use the functions exported by each of the libraries you must include the respective header file in the your code.

An example of the use of the peripheral libraries is the USB peripheral interface functions in the `src/app/app.c` source file. For details on using the USB interface refer to section 6 on page 11.

4.1. GPIO peripheral library

One of the basic but useful features of the MCU are the digital Input/Output (I/O) pins. The **gpio** peripheral library provides functions to write (output) and read (input) digital I/O, taking care of all the configuration of the respective MCU pins to function as a digital input or output depending if a read or write functions is called.

The following I/O pins are supported by the **gpio** peripheral library.

Pin	I/O	Input Buffer Type	Pin Index
RB15	Input/Output	Schmitt Trigger	1
RB14	Input/Output	Schmitt Trigger	2
RB13	Input/Output	Schmitt Trigger	3
RB3	Input/Output	Schmitt Trigger	4
RA1	Input/Output	Schmitt Trigger	5
RB9	Input/Output	Schmitt Trigger	6
RB8	Input/Output	Schmitt Trigger	7
RB7	Input/Output	Schmitt Trigger	8
RB4	Input only	Schmitt Trigger	9
RB1	Input/Output	Schmitt Trigger	10
RA0	Input/Output	Schmitt Trigger	11
RA2	Input/Output	Schmitt Trigger	12
RA3	Input/Output	Schmitt Trigger	13
RA4	Input only	Schmitt Trigger	14
RB2	Input/Output	Schmitt Trigger	15

This peripheral library source code is located in the `src/peripherals/io/` firmware project folder.

4.1.1. Write/output

The following function configures a I/O pin to digital output and sets it to High or Low depending if a 1 or 0 was written.

```
void gpio_write(unsigned char pin_index, unsigned char value)
```

`pin_index` - I/O pin index as defined in the table above.

`value` - 0 or 1 for High and Low, respectively.

4.1.2. Read/input

The read function configures the I/O to digital input and returns 1 or 0 depending if the input is at High or Low level, respectively.

```
unsigned char gpio_read(unsigned char pin_index)
```

`pin_index` - I/O pin index as defined in the table above.

This function will return the digital the value read from the pin, 0 if Low and 1 if High.

5. Modules

The module libraries export functions to interact with add-on boards or components. Currently the following modules are supported.

- **ykemb** - YKEMB board interface module.
- **ykhat** - YKHaT Humidity and Temperature board interface module.
- **ts100** - RTD PT100 based temperature sensor interface module.
- **ykur** - YKUR board control interface module.

The module libraries are located in the `src/modules/` folder. Each module library will have it's own subfolder.

5.1. YKEMB module

YKEMB is an EEPROM board that can be used with YKAMELEAN24 as an add-on board to provide persistent data memory. These add-on boards are connected using the I^2C bus and multiple EEPROMs can be stacked as they can be configured with different addresses. Please check /www.yepkit.com/product/300107/YKEMB for details.

The **ykemb** module library provides functions for writing and reading bytes to and from YKEMB boards. To use this functions in your code you need to include the `ykemb.h` header file.

5.1.1. Write byte

To write a byte to a YKEMB board EEPROM memory use the following function.

```
unsigned char ykemb_byteWrite(unsigned char dataByte,
                              unsigned char deviceAddr,
                              unsigned char *wrtAddr)
```

`dataByte` - Byte to be written to the YKEMB EEPROM.

`deviceAddr` - YKEMB device I^2C address.

`wrtAddr` - Pointer to the two byte memory address of the byte to be written.

5.1.2. Read byte

To read a byte from a YKEMB board EEPROM memory use the following function.

```
unsigned char ykemb_i2cSelByteRead(unsigned char deviceAddr,
                                   unsigned char *readPrevAddr,
                                   unsigned char *byteBuff)
```

`deviceAddr` - YKEMB device I^2C address.

`readPrevAddr` - Pointer to the two byte address of the byte to be read.

`byteBuff` - Pointer to the byte that will store the byte read.

5.2. YKHaT module

YKHaT is a breakout board of a Temperature and Humidity sensor. The board is connected to YKAMELEAN24 boards using the I^2C bus. For details on the YKHaT please visit www.yepkit.com/product/300111/YKHAT.

To use the YKHaT module functions in your own firmware code you need to include the `src/modules/ykhat/ykhat.h` header file on your source code.

5.2.1. Get Temperature in ºC

Function that fetches the current temperature reading from the sensor and returns the value in Celsius unit.

```
int ykhat_get_tempC(char addr)
```

`addr` - I^2C address of the target YKHaT sensor board.

This board returns an `int` with the temperature reading in ºC unit.

5.2.2. Get Temperature in ºF

The temperature can also be obtained in Fahrenheit unit, for that use the following function.

```
int ykhat_get_tempF(char addr)
```

`addr` - I^2C address of the target YKHaT sensor board.

This board returns an `int` with the temperature reading in ºF unit.

5.2.3. Get Humidity in %RH

To get the relative humidity level reading, from the sensor, use the following function.

```
int ykhat_get_hum(char addr)
```

`addr` - I^2C address of the target YKHaT sensor board.

This board returns an `int` with the humidity reading in %RH unit.

5.2.4. YKHaT module usage example

The native firmware makes available through the USB interface the functionalities of the YKHaT module library, at the implementation of this interface can be used as reference on how to use the `ykhat` library functions.

In the `src/modules/app/app.c` source file there is a handler function example for using the YKHaT module functions. For example, the following handler function is called when a YKHaT command is received through USB by the YKAMELEAN24 board.

```
char app_ykhat(unsigned char *ReceivedDataBuffer, unsigned char *ToSendDataBuffer)
```

Where:

`ReceivedDataBuffer` - Pointer to the 64 byte buffer containing the received command through USB interface.

`ToSendDataBuffer` - Pointer to the 64 byte buffer to be transmitted back to the USB host.

5.3. TS100 module

TS100 is a Platinum (Pt100) Resistance Temperature Detector (RTD) sensor board with a Serial Peripheral Interface (SPI). For details on the TS100 board please visit *www.yepkit.com/product/300103/TS100*.

The module library makes available functions to:

- Enable the sensor;
- Get raw sensor reading;
- Get temperature reading in Celsius;
- Get calibration value.

5.3.1. Enable the sensor

The following function will configure the SPI to communicate with TS100 board and enable the sensor. For details on how to connect the TS100 to a YKAMELEAN24 board refer to "Ykamelean24 add-on modules reference" document and "TS100 setup guide" page at *yepkit.com*.

```
void ts100_spi_enable(void)
```

5.3.2. Get raw sensor reading

Gets from the sensor the raw reading of Pt100 digitalized to 22 bit by the inboard ADC. Please check the TS100 board documentation for details on how to convert the raw reading into temperature.

The function is the following.

```
unsigned long ts100_get_sensor_reading(void)
```

5.3.3. Get temperature reading in Celsius

To get the temperature already converted to Celsius unit, use the following function.

```
double ts100_get_temperature(void)
```

5.3.4. Get calibration value

A calibration factor should be used to get the most accurate temperature values. To calibrate set the TS100 calibration jumper[3] and call the following function to fetch the calibration value to be considered.

```
double get_calibration_val(void)
```

5.4. YKUR module

To control the YKUR board[4] through I^2C you can use the functions made available by this module. The functionalities exported by the module are:

- Send command;

[3] Please refer to TS100 product page for documentation on how to set the board into calibration.

[4] www.yepkit.com/product/300106/YKUR

- Get response.

YKUR board in the I^{2}C interface is configured in slave mode being the responsibility of the master, the Ykamelean24 board, to send commands and ask for responses when applicable.

5.4.1. Send command

The YKUR documentation, that can be found at the product page[5], under the I^{2}C control interface has a mapping between byte value OpCodes and actions to be executed by the YKUR. The send command function, bellow, sends this OpCode value to a YKUR board for it to execute the corresponding action/command.

```
char ykur_i2cSendCommand(unsigned char cmd_buffer, char num_bytes, unsigned char address)
```

5.4.2. Get response

In some situations it is possible to fetch information from the YKUR board[6]. To do so use the following function.

```
char ykur_i2cGetResponse(char *cmd_buffer, char address)
```

[5] www.yepkit.com/product/300106/YKUR
[6] Check documentation on the YKUR product page.

6. USB Communication Protocol

A key feature of YKAMELEAN24 is the level of access and control the user has through the USB interface. Most of the board functionalities can be configured, accessed and controlled through the USB connection. This allows the user to explore the board from a PC with an USB connection.

An USB host software to control the YKAMELEAN24 board is available for download in the product page.

YKAMELEAN24 board will appear to the USB host as an HID device and the communication protocol is based on 64 byte messages.

6.1. Communication Packet Structure

- Byte **0** - Scope/Interface
- Byte **1** - Message Type
- Byte **2** - Action
- Byte **3** to **63** - Action specific

Communication from the USB host (typically the user PC) to the board must have a target scope or interface, for example if the user wants to control an I/O pin the scope will be the GPIO. This scope is defined by the first byte of the USB message/report sent from the host.

There are two main groups of scopes/interfaces, the so called **inner scopes** and the **modules scopes**.

Inner scopes are related to board features and peripherals (e.g., I/O's, ADC, I^2C and other).

Modules scopes are related to add-on boards that can be connected to the YKAMELEAN24.

Bellow are the scopes supported by the current version of the firmware.

Byte 0 value	Scope	Scope type	Description
0x01	gpio	Inner	Control general I/O pins
0x10	ykemb	Module	Interface with YKEMB board
0x11	ykur	Module	Interface with YKUR board for control through I^2C
0x12	ykhat	Module	Interface with YKHaT temperature and humidity sensor board

6.1.1. General Purpose I/O (GPIO)

In this scope the GPIO's are all configured to digital I/O. The configuration is done pin by pin when a **gpio** command is received and just for the targeted pin. No pre-configuration is assumed, so every time a command is received the proper configuration to execute such command is performed, for example if a Write command is received for pin `RB14` the high level steps are the following:

1. Configure pin `RB14` as digital output;
2. Set output pin level (high/low) as defined by the command received.

Note that if this specific pin was previously configured as being used by a peripheral that configuration will be overwritten by the one associated with the command received.

The structure of a **gpio** scope command is the following.

Byte number	Value
0	0x01
1	0x01 - Command 0x02 - Response
2	0x01 - Write command 0x02 - Read command
3	Pin index
4	Pin value (1 or 0)
7 to 63	don't care

Pin index is defined in the following table.

Index	Pin
1	RB15
2	RB14
3	RB13
4	RB3
5	RA1
6	RB9
7	RB8
8	RB7
9	RB4
10	RB1
11	RA0
12	RA2
13	RA3
14	RA4
15	RB2

Pin value is the value to be written to the addressed pin or the value read from the pin. A 0 value is a pin at Low level. A 1 value is a pin at High level.

6.1.2. YKEMB Interface

The **ykemb** interface allows the user read/write access from the USB host to a YKEMB board connected to the YKAMELEAN24 board. Multiple YKEMB boards can be connected to a single YKAMELEAN24 as they can be set up with distinct I^2C addresses.

The following table describes the command structure for the ykemb interface scope.

Byte number	Value
0	0x10
1	0x01 - Command 0x02 - Response
2	0x01 - Write command 0x02 - Read command
3	I²C device address
4	Memory address High byte
5	Memory address Low byte
6	Data byte
7 to 63	don't care

I²C device address is the three less significant bits of the ykemb device I²C address which are configured by the user through the on board three way DIP switch.

Memory address High byte is the most significant byte of the two byte memory address to/from which the data should be written/read.

Memory address Low byte is the less significant byte of the two byte memory address to/from which the data should be written/read.

Data byte is the data byte to be written.

6.1.3. YKUR Interface

The firmware implements an USB interface to interact with the YKUR i²C control interface. With it the YKAMELEAN24 works as a bridge between USB and I²C. This is useful for testing the I²C connection between the boards or to improve integration in a use case or setup where a both a YKAMELEAN24 board and YKUR board is used and the user does not want to have two USB cables (one for YKAMELEAN24 and another for the YKUR).

The USB message structure is the following for a command sent by the USB host (e.g., PC) to YKAMELEAN24.

Byte 0	Byte 1	...	Byte 63
0x11	opcode	don't care	

When the YKAMELEAN24 receives this command message it will forward the `opcode` to the YKUR board through I²C.

6.1.4. YKHaT Interface

The following table describes the command structure for the ykhat interface scope.

Byte number	Value
0	0x12
1	0x01 - Command 0x02 - Response
2	0x01 - Get temperature in °C 0x02 - Get temperature in °F 0x03 - Get humidity in %RH 0x11 - Start sensor reading
3	I^2C device address
4	Response value MSB
5	Response value LSB
6 to 63	Don't care

7. The firmware and the bootloader

YKAMELEAN24 boards are preloaded with a bootloder and the latest stable version of the firmware. The main purpose of the bootloader in the YKAMELEAN24 boards is to enable the user to load new versions or their own custom versions of the firmware using the USB interface instead of requiring a PIC programmer.

The user can build their own version of the firmware, creating a hex file and then load it to the board directly from the PC through USB using the Host Bootloader application, which is available for download from the product page.

Nonetheless, if the user has a PIC programmer it can be used, as usual, to program the firmware to the microcontroller (MCU).

To build a custom firmware the user should use the MPLABX software from Microchip, freely available for download, and include the `src/hid_boot_p24FJ128GB202.gld` linker file in their own firmware project.

Also the bootloader and the firmware should share the same configuration bits. The configuration bits used in the bootloader are in the `src/inc/configuration_bits.h` file in the firmware project folder.

A. Build process and tools

To build this firmware the following tools are used.

- MPLABX IDE
- XC8 Compiler

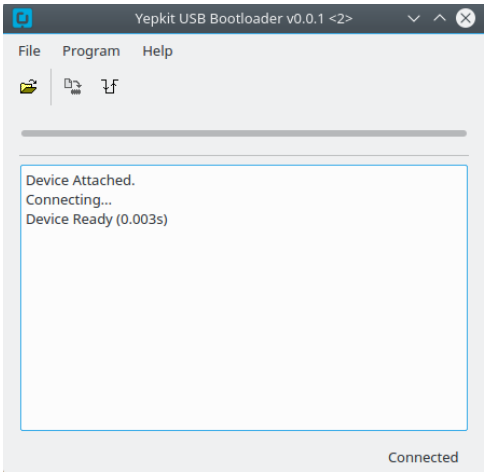
Both can be downloaded from http://www.microchip.com and a free license is available.

B. Using the bootloader to update the firmware through USB

The YKAMELEAN24 board bootloader has the capability to program the device with firmware images received through USB. To do so a host application capable of interacting with the board bootloader is required.

The bootloader is compatible with the Microchip MLA utilities HIDBootloader application which can be used to program new firmwares to the YKAMELEAN24 boards.

To update the firmware using the USB you will need the YKBootloader application which allows to import firmware images and load them to Ykamelean24 boards using the USB.



This application is available for download at the YKAMELEAN24 product page.

Steps to update the firmware version on YKAMELEAN24 board through USB with the bootloader application:

1. Download the firmware binary file (hex file);
2. Connect the board to your PC through USB;
3. Set the board into bootloader mode by connecting `SDA/RB` and `SCL/RB8` pins to `GND` and resetting the device (press `RESET` button), at which point the indicator LED should start to fast blink indicating that the board is in bootloader mode;
4. Open the USB Bootloader Application (YKBootloader for Linux and HIDBootloader for Windows);

5. Load the firmware binary file by clicking `Import Firmware Image`;
6. Program the new firmware image by clicking `Erase/Program/Verify Device`;
7. Once the programming process has finished, reset the device by clicking `Reset Device` or pressing the board `RESET` button.

B.1. Bootloader application for Windows operating systems

A package with the Microchip MLA HIDBootloader application binaries for Windows operating systems is available for download at the YKAMELEAN24 product page. Refer to *http://www.microchip.com/mplab/microchip-libraries-for-applications/mla-license* for MLA licensing details.

Bellow are the steps to run the bootloader application on Windows systems.

Start by downloading the Bootloader Application for Windows at the YKAMELEAN24 product page and extract the files into a folder of your choosing. Open a command line console (cmd) and navigate to the folder containing the extracted files and run the following command.

```
HIDBootloader.exe 0x04D8 0xF25C
```

Where `0x04D8` and `0xF25C` are the board **VID** and **PID** respectively.

Note that if you click on the executable `HIDBootloader.exe` file the application will open but it will not recognize the YKAMELEAN24 board as the VID and PID were not provided.

B.2. Bootloader application for Linux operating systems

For Linux the application can be build from the source code. Both the source code and build instructions for the most up-to-date version is available at the *https://github.com/Yepkit/ykam24cmd* github repository.

Once built, run this application just by clicking on the **YKBootloader** executable file. In this case there is no need to call from the command line and provide the VID and PID.

C. Revisions

1.0.0 New source code structure and new components.

- New source code structure
- GPIO peripheral added.
- YKHaT module added.
- YKUR module added.

0.1.0 Initial release.