

Core MCU Framework Documentation

Core MCU Framework Documentation

- [Core MCU Framework Overview](#)
 - [What is Core MCU Framework?](#)
 - [License](#)
 - [Versions](#)
 - [Releases](#)
 - [Core MCU Framework - Download](#)
- [Getting Started](#)
 - [Directory Layout CORE8-16F Version](#)
- [CORE Interface](#)
 - [CORE API Documentation - Core8-16F](#)
- [HAL Interface](#)
- [Drivers](#)

Core MCU Framework

Overview

What is Core MCU Framework?

The Core MCU Framework is a specialized software platform designed for developing applications that run on microcontroller units (MCUs).

The goal of Core MCU Framework is to make it easy to move between various MCU processor platforms.

Portability Across Different MCUs

One of the primary challenges in embedded systems development is dealing with the heterogeneity of hardware. Different MCUs have their own sets of instructions, peripherals, and memory architectures. Core MCU Framework aims to abstract these differences so that developers can write code that is portable across various MCU processors. This means you can take a program designed for one MCU and adapt it to another with minimal changes.

Simplified Development Process

Core MCU Framework provides a collection of libraries and tools that simplify common tasks in embedded systems programming.

For example, Core MCU Framework offer standardized libraries for handling I/O operations, timers, interrupts, and other peripherals common to MCUs. This can significantly speed up the development process and reduce the learning curve when moving between different MCUs.

Consistent API

A uniform application programming interface (API) across various MCUs allows developers to apply the same code structure and calls, regardless of the underlying hardware. Core MCU Framework's consistent API ensures that developers don't need to relearn command sets or libraries when switching MCUs, thus saving time and reducing errors.

License

Core MCU Framework is free for non-commercial use.

This means, if you are a hobbyist or just want to play around with Core MCU Framework, you may do so.

However, if you are selling a product, making money from something using it - we need to talk. I think it is only fair to get a cut, so I can support my hobbies and create more cool stuff. Don't you think so?

Core MCU Framework Software License Agreement

1. Definitions

- **“Framework”** refers to the Core MCU Framework, which is a specialized software platform designed for developing applications that run on microcontroller units (MCUs).
- **“Licensee”** refers to any individual or entity that uses the Framework.
- **“Commercial Use”** involves any use of the Framework in a commercial activity where the primary intention is revenue generation.

2. Grant of License

- **Non-Commercial License:** The Framework is available free of charge for non-commercial use by hobbyists, students, and developers for educational or personal projects. This license allows you to use, modify, and distribute the Framework in non-commercial settings.
- **Commercial Use License:** Use of the Framework in any commercial activity requires a separate license agreement. Interested parties should contact the Licensor to negotiate terms suitable for commercial distribution and support.

3. Portability and API Consistency

- The Framework is designed to abstract the differences across various MCUs, providing a consistent API that enhances portability and simplifies the development process. This includes standardized libraries for I/O operations, timers, interrupts, and other peripherals.

4. Restrictions

- You may not use the Framework for any commercial purposes without a separate licensing agreement.
- You shall not misrepresent the origin or ownership of the Framework.

- Modifying or extending the Framework for commercial use without the proper commercial license is strictly prohibited.

5. Rights of the Licensor

- The Licensor reserves the right to update the terms of this license and the Framework itself at any time. Notifications of such updates will be provided to all active users.

6. Limitation of Liability

- The Framework is provided “as is”, without warranty of any kind, express or implied. The Licensor shall not be liable for any claims or damages whatsoever resulting from the use of the Framework, including direct, indirect, incidental, or consequential damages.

7. Termination

- This License is effective until terminated. The License will terminate immediately without notice from the Licensor if you fail to comply with any provision of this License. Upon termination, you must cease all use of the Framework and destroy all copies, full or partial, of the Framework.

8. Miscellaneous

- This License constitutes the entire agreement between the parties relating to the use of the Framework and supersedes all prior or contemporaneous understandings regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by the Licensor.

Versions

Core MCU Framework - Version is based on the specific MCU device you're working with.

Core8-16F - Designed for PIC16 Series Devices - Compatible with:

- PIC16F15313
- PIC16F15323
- PIC16F15324
- PIC16F15325

Core8-18F - Designed for PIC18 Series Devices - Compatible with:

- PIC18F2xQ84

Releases

Core8-16F Releases

Date	Version	Notes
2024/09/09	1.0.0	Initial Version
2024/09/24	1.0.1	Added HAL Interfaces

Core8-18F Releases

Date	Version	Notes
2024/10/11	1.0.0	Initial Version - Beta

Core MCU Framework – Download

The Core MCU Framework Versions, along with drivers can be found in my Github Repo.

[Core MCU Framework Download – Link](#)

Getting Started

Directory Layout CORE8-16F Version

Directory Structure Documentation for CORE8-16F Version

Overview

This documentation provides an overview of the directory structure of the CORE8-16F microcontroller framework. The purpose of this guide is to help developers quickly understand the framework's layout, where to find specific components, and the function of each directory and file.

Top-Level Directory (`src/`)

- `main.c`: The main entry point of the program. This file contains the initial setup and execution flow for the framework.

Core Framework Directory (`core16F/`)

- `core16F.h`: The primary header file for the CORE8-16F framework, providing core definitions and function prototypes.
- `core16F_const.h`: Defines constants used throughout the core framework, ensuring consistent values are used across different modules.
- `core16F_init.c`: Initialization routines for the CORE8-16F, setting up essential configurations before main program execution.
- `core_version.h`: Tracks the version of the core framework, providing version information that can be used for compatibility checks.

Core System Subdirectory (`core16F_system/`)

- **delays/**: Provides delay routines with `delays.c` and `delays.h` for timing control in the program.
- **events/**: Manages system events, with `events.c` and `events.h` handling event triggering and processing.
- **timer/**: Includes files (`isr_core16F_system_timer.c`, `.h`) for handling system timer interrupts, crucial for timing operations and event scheduling.
- **utils/**: Utility functions (`utils.c`, `utils.h`) that provide common operations used throughout the core, like math functions or data conversions.

Core Interface Documentation

The core interface provides essential functions and definitions that are crucial for developers to interact with the CORE8-16F framework effectively. Below are the key files and their purposes:

- **core16F.h**: This is the primary interface header for the core framework, containing:
 - Function prototypes for initializing and managing core functionalities.
 - Core definitions and macros that simplify development and ensure consistent use of common patterns.
 - Inline documentation for each function, explaining its purpose, parameters, and return values.
- **core16F_const.h**: This file defines all the constants used within the core system:
 - Constants such as clock frequency, system status codes, and other fixed values that maintain uniformity throughout the framework.
 - It ensures that all core modules use the same references, reducing discrepancies and simplifying debugging.
- **core16F_init.c**: Contains functions used for initializing the core framework:
 - **void core_init(void)**: This function sets up the initial configurations, including clock settings and system timers.
 - **Initialization Routines**: Initializes critical subsystems such as GPIO, timers, and interrupts. Each of these routines is modular, allowing developers to include or exclude components based on their project requirements.
- **core_version.h**: Used for tracking the version of the core:
 - Contains macros that specify the version number of the framework.
 - Useful for ensuring compatibility when integrating different versions of the core with other modules or projects.

Device Configuration (`device/`)

- **Device Configuration Files**: Contains configuration files specific to different PIC microcontrollers, such as `16F15313_configBits.h` and `16F1532x_configBits.h`, which provide settings for configuration bits like clock sources and watchdog timers.
- **Device-Specific Headers**: Files like `16F15313_core16F_config.h` and `16F1532x_core16F_config.h` contain setup information that is specific to each microcontroller model.

Driver Directory (`drivers/`)

- **`ds18b20/`**: Contains `ds18b20.c` and `ds18b20.h` files for interfacing with the DS18B20 temperature sensor.
- **`lcd_i2c/`**: Provides the driver for I2C-connected LCD displays, including source (`lcd_i2c.c`) and header (`lcd_i2c.h`) files.
- **`led_rgb/`**: Handles RGB LED control with `led_rgb.c` and `led_rgb.h`, providing functions to adjust colors and brightness.

Hardware Abstraction Layer (`hal/`)

- **`gpio/`**: Manages GPIO pins with `gpio.c`, `gpio.h`, as well as analog-specific functionality (`gpio_analog.c`, `gpio_analog.h`). Provides functions for configuring and manipulating digital and analog pins.
- **`i2c1/`**: I2C protocol support (`i2c1.c`, `i2c1.h`), allowing communication with devices on the I2C bus.
- **`nco1/`**: Provides functionality to handle Numerically Controlled Oscillators (NCO) with `nco1.c` and `nco1.h`.
- **`one_wire/`**: Implements the One-Wire communication protocol with `one_wire.c` and `one_wire.h`, typically used for sensors like DS18B20.
- **`pps/`**: Manages the Peripheral Pin Select (PPS) feature (`pps.c`, `pps.h`) which allows flexible I/O mapping.
- **`pwm3/`, `pwm4/`, `pwm5/`, `pwm6/`**: PWM modules (`pwmX.c`, `pwmX.h`) providing functions to control Pulse Width Modulation outputs for different channels.
- **`serial1/`**: UART communication support (`serial1.c`, `serial1.h`, `serial1_isr.c`), handling serial communication including interrupt-based transmission and reception.
- **Timers (`tmr0/`, `tmr1/`, `tmr2/`)**: Provides timer support (`tmrX.c`, `tmrX.h`) for controlling time-based events and functions.

Interrupt Service Routines (`isr/`)

- **`isr_control.c`, `isr_control.h`**: Manages interrupt control, providing a central place for handling various hardware interrupts.
- **`main_isr.c`**: The main interrupt service routine, coordinating different interrupts such as timers, serial communication, and more.

Purpose and Use

The structure of the CORE8-16F framework is designed to provide modular and reusable components, allowing developers to easily add, modify, or remove features as needed. By understanding the purpose of each directory and its contents, developers can efficiently navigate the codebase and make effective contributions or adaptations to their specific needs.

CORE Interface

CORE API Documentation - Core8-16F

The CORE object is an instance of the CORE16F_System_Interface_t structure, which provides a collection of core functionalities available in the CORE8-16F framework. Below is a brief overview of the key functions and features available through CORE.

CORE Definition

```
const CORE16F_System_Interface_t CORE = {  
    .Initialize = &CORE16F_init,  
  
    #ifdef _CORE16F_SYSTEM_INCLUDE_DELAYS_ENABLE  
        .Delay_MS = &CORE16F_Delay_BlockingMS,  
    #endif / _CORE16F_SYSTEM_INCLUDE_DELAYS_ENABLE/  
  
    #ifdef _CORE16F_SYSTEM_EVENTS_ENABLE  
        .Events_Initialize = &TimedEventSystem_Init,  
        .Events_Add = &ScheduleEvent,  
        .Events_Check = &CheckEvents,  
        .Events_Remove = &CancelEvent,  
    #endif  
  
    .Make16 = &CORE_Make_16,  
    .Low4 = &CORE_Return_4bit_Low,  
    .High4 = &CORE_Return_4bit_High,  
    .Set_Bit = &CORE_Set_Bit,  
    .Clear_Bit = &CORE_Clear_Bit,  
    .FloatToString = &CORE_floatToString,  
    .IntToString = &CORE_intToString,  
};
```

Key Functionalities of CORE

- **Initialization:** `CORE.Initialize()` initializes the core system, including clock and peripheral setup.
- **Delays** (Conditional): If enabled, `CORE.Delay_MS(timeMS)` provides a blocking delay in milliseconds.
- **Event Management** (Conditional): If enabled, CORE offers several event management functions:
 - **CORE.Events_Initialize():** Initializes the event management system, allowing events to be scheduled and handled.
 - **CORE.Events_Add(delay_ms, callback, interval):** Schedules an event to occur after a specified delay in milliseconds. The callback function will be called when the event occurs, and interval specifies if the event should repeat (0 for a one-time event).
 - **CORE.Events_Check():** Checks for any pending events and executes their associated callbacks if the conditions are met.
 - **CORE.Events_Remove(callback):** Cancels a previously scheduled event by providing the callback function associated with it. This is useful for stopping recurring events or removing unwanted scheduled events.
- **Utility Functions:** Various helper functions are provided:
 - `CORE.Make16(high_byte, low_byte):` Combines two 8-bit values into a 16-bit value.
 - `CORE.Low4(byte)`, `CORE.High4(byte):` Extracts lower or higher 4 bits from an 8-bit value.
 - `CORE.Set_Bit(byte, bit_position)`, `CORE.Clear_Bit(byte, bit_position):` Sets or clears a specific bit in an 8-bit value.
 - `CORE.FloatToString(number, buffer, decimalPlaces):` Converts a float to a string representation.
 - `CORE.IntToString(number, buffer):` Converts an integer to a string representation.

The CORE object provides a convenient and centralized interface for accessing key functions of the CORE8-16F system, making it easier for developers to initialize, control, and manage the microcontroller's features effectively.

HAL Interface

Drivers