Dodeca Time Tracker

Making Embedded Systems – Final Project

Graeme Gets

2022

# Contents

[Contents 2](#_Toc102728499)

[Figures 2](#_Toc102728500)

[Overview 3](#_Toc102728501)

[Features 3](#_Toc102728502)

[Hardware 3](#_Toc102728503)

[Architectural Diagrams 4](#_Toc102728504)

[Hardware Descriptions 6](#_Toc102728505)

[Accelerometer/ Gyro 6](#_Toc102728506)

[Serial Command Line Interface 6](#_Toc102728507)

[Buttons 6](#_Toc102728508)

[LED 6](#_Toc102728509)

[Battery & Fuel Gauge 6](#_Toc102728510)

[Software Descriptions 7](#_Toc102728511)

[Command Line Interface 7](#_Toc102728512)

[Commands 7](#_Toc102728513)

[Task Time Management 8](#_Toc102728514)

[Task Assignment 8](#_Toc102728515)

[Task Control 8](#_Toc102728516)

[Data logging structure 8](#_Toc102728517)

[Addressable LED’s 8](#_Toc102728518)

[Task State machine. 9](#_Toc102728519)

# Figures

[Figure 1- Hardware Block Diagram 4](#_Toc102640727)

[Figure 2 - Software Block Diagram 4](#_Toc102640728)

[Figure 3 - Software Hierarchy Diagram 5](#_Toc102640729)

# Overview

This is a **remake** of a product I saw and thought I would like to make it myself. The idea is to use a Dodecahedron to create a physical way to track time spent during the day - In other words, tracking tasks. Each side of the dodecahedron can be assigned a task such as ‘Email’, ‘Coding’, ‘Lunch’, ’meeting’ etc. As you start or stop a task you simply set the Dodecahedron with the task you are about to do facing up.

The system logs the start time, End time (including date) and duration of each task. This data can then be exported via the command terminal. For the initial project, this will be via serial, but in furture this would be via Wifi or Bluetooth to a device or server.

A set of LED’s and sound will be used to attract attention when a tasks allotted time it up.

# Features

* 12 sided Dodecahedron
* Configuration – Via Command line initially
  + Set each face task
  + Set task min/ max time
  + Set End time
  + Configure and save task configuration to flash.
* Each side has RGD lighting to indicate:
  + Flash to indicate x number of Mins
  + Task is being timed
  + Task time has ended
  + Task has been paused
* Speaker to play relevant sounds
  + End of time/start/ stop/pause
* Accelerometer & Gyro to determine which side is facing up.
* (Button on each side to Pause/Start)
* (Initially – Radio Comms for Command line - xBee)

# Hardware

|  |  |  |
| --- | --- | --- |
| Name | Description | Notes |
| Processor board | Black Pill STM32F411CEU6 | Chosen for small footprint so that it will fit into the dodecahedron. It also has a 32.7K crystal for the RTC and enough peripherals. Memory and features needed for the project. |
| Accelerometer and Gyro | GY-521 | The module uses the MPS6050 mems chip. |
| Addressable LED’s | WS2812 | Each side will have a 12 LED ring of Addressable LED’s |
| I2S Amp | Max98357A |  |
| Battery Gauge | Adafruit fuel gauge | On Semiconductor LC709203F - Smart LiB Gauge  Battery Fuel Gauge LSI  for 1-Cell Lithium-ion (Li+) |
| RF Serial | Xbee S2C | Used for serial communication. Configured for straight through serial. |

# Architectural Diagrams

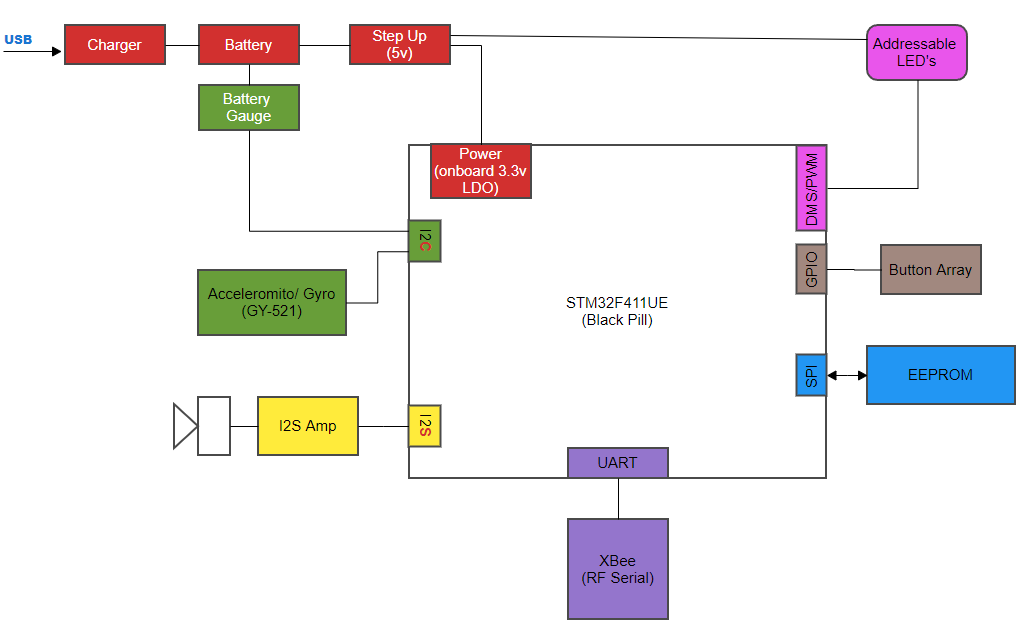


Figure 1- Hardware Block Diagram

Diagram

Description automatically generated

Figure 2 - Software Block Diagram

Diagram

Description automatically generated

Figure 3 - Software Hierarchy Diagram

# Development Environment and Tools

This project is coded using the STM32Cube IDE and the HAL layer.

Code can be found on Github here [making-embedded-projects/final-project at main · graeme-gets/making-embedded-projects (github.com)](https://github.com/graeme-gets/making-embedded-projects/tree/main/final-project)

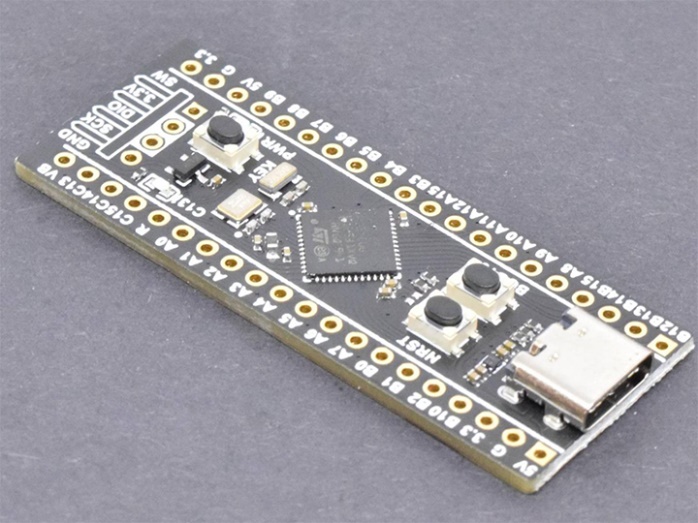
## Tools

Multimeter

Saleae logic analyser

# Hardware Descriptions

## Processor

The ‘Black Pill’ processor, developed by [WeAct Studio](https://github.com/WeActTC/WeAct-Studio-Product), was used in this project due to the ample peripherals, clock speed and Flash/Ram configuration. All allowing for more than enough program and memory space which allows for expansion.

The specific board uses the STM32F411CEU6.

Basic processor Specifications

|  |  |
| --- | --- |
| Clock | 100Mkz |
| Flash | 512K |
| Ram | 128K |
| Cortex M4 |  |
| Peripherals | ACD x 1  RTC x 1  Timers x 8  UARTS  I2C x 3  SDIO x 1  SPI x 5  USART x 3  USB\_OTG\_FS I2S x 5  CRC |

**Peripherals used in Project**

RTC for time keeping and alarms

Timer 9 for sleep wake up in interrupt

UART for xBee serial communication via RF

CRC – for configuration Checksum

The Dev board has the following Specifications

|  |  |
| --- | --- |
| HSE external crystal | 25 Mhz |
| LSE | 32.768kHz |
| LDO regulator for 5V input |  |
| USB-C |  |
| Buttons | User Key BOOT Reset |

## External Hardware

### Accelerometer/ Gyro

The heart of the project is the detection of the orientation of the dodecahedron linked with the real time logging. The system will need to detect which side of the dodecahedron is facing up and detect a change. The system detects between the specific orientation change and ignores any other movement to cater for the device being moved around the work area or bumped.

The module being used is the GY-521 module which uses the MPU-6050 3 Axis Gyroscope/Accelerometer chip

### RF Serial Module – Xbee S2C

The Xbee is used to transfer serial data via RF. The Dodec Timer cannot have cabled attached and need to be untethered from the PC.

The PAN ID’s and serial configuaration were set up manually via the Digi International configuration software called XCTU.

For the purposes of this project, I did not add in Xbee configuration from the device as only one was being made, however, the configuration of the module is easy with simple AT command via serial.

The Xbee radio uses TTL UART for communication protocol to the SMT32.

## I2S Audio Amp

A small Audio amp module using the I2S communication protocol is used for various alerts. These will be small sound bytes in the form of pleasant beeps or tones.

The module is a Max98357A driver from Adafruit (<https://www.adafruit.com/product/3006>)

## LED

Each side of the dodecahedron has a ring of 12 addressable LED’s. The face up LED’s indicate the following

The LED ring is a 12 LED 50mm ring using WS2812B LED’s.

The brightness is kept to approx. 50% in order to reduce the power consumption of the Led’s. If white, for instance, is used at full brightness we can expect there to be a 60 mA draw ***per*** LED. As there are 12 LED rings (one for each side) each with 12 RGB Led’s, the total current draw would be a wopping 144 x 60 mA = 8.64 Amps which is way over the capability of the power system.

However, only the top face is used for the continuous status of the timer limiting the amperage to a maximum of 60 mA x 12 = 720mA. Having just one colour further reduces this to 20 mA x 12 = 240mA. This is even further reduced by a lew brightness giving a measured average of around 40mA.

Side Note : In reality, for a production device of this size its very unlikely that so many LED’s would be used.

## Battery & Fuel Gauge

The fuel gauge allows the device to monitor the remaining power and let the user know if it needs charging. A slow flashing LED can be used to indicate the need for a charge.

## 3.3 v Regulator

The device regulates the incoming power from the Lipo battery to a constant 3.3v.

The regulator used is the TPS6306x by Texas Instruments. Its is a Buck-Boost converter with a max current draw of up to 2A which is far more then the device will need to consume

## Battery

The battery is a LIPO 3.7V 2000mAh from MakerFocus.

## Power Design & considerations

The system can run off 3.3V for all peripherals except the Addressable LEDs. While the LED rings are designed for 5V operation they appear to work well enough on 3.3v, particularly the logic 1 & 0 levels. However the power supply will need to be able to deliver a maximum of 500mA to the LED’s directly to cater for the power consumption. A far lower power rating is achievable if only 2 rinds at most are at full use and brightness.

# Software Descriptions

Imported Software and Licenses

The device only uses two libraries from an external source

1. DMA control for the Addressable LED’s
   1. GitHub - [hey-frnk/STM32\_HAL\_NeoPixel: SK6812 RGBW NeoPixel using STM32 HAL on NUCLEO-F042K6 STM32F042K6 STM32F0 (github.com)](https://github.com/hey-frnk/STM32_HAL_NeoPixel)
   2. The sk6812.c file was used and modifieds for this devices purpose
   3. There is no licence information. Credit to the author Frank from VDF collective
2. MPU6050 Accelerometer
   1. The main need for this library was for the Kalman filter to obtain angles in degrees from the accelerometer rating.
   2. Github [leech001/MPU6050: STM32 HAL library for GY-521 (MPU6050) with Kalman filter (github.com)](https://github.com/leech001/MPU6050)
   3. Licence – GNU Public Licence

## Command Line Interface

The system uses a serial command line interface for both system check/debugging as well as task and device management.

Note:

It would be better to use a WiFi or Bluetooth connection with an app to manage the tasks and log the data to a server. However, this project is to prove the embedded software and so the app/web interface can be achieved at a later date.

### Commands

The face ID and the Task ID are the same and are linked when the task is assigned.

|  |  |  |
| --- | --- | --- |
| **Command** | **Parameters** | **Description** |
| Help | None | Lists all the command line options |
| Reset! | None | Resets the device to factor settings. At this stage this is a set of predefined tasks allocated to different faces. |
| Reboot! | None | Software reboot |
| Ver | None | Displayes the firmware version running on the device |
| time? | None | Displays the current time from the RTC |
| date? | None | Displays the current date from the RTC |
| time | hh:mm:ss | Sets the time |
| date | dd-mm-yy | Sets the date |
| acc? |  | reads the Accelerometer values |
|  | r – read  n – number of times to read |  |
|  | g – get register value  n – register address to retrieve | Gets the hex value of a register |
|  | w – Write Register  n – Register address  v – Register value | Sets a value for a specific register |
| lipo? | None | Reads the current battery voltage |
| led |  | Controls for the Addressable LED’s |
|  | f – Set face colour  n – Face number  c- colour (r|g|b) | Set a specific dodecahedron face colour |
|  | o | turn off all LED’s |
|  | d | do led display around all faces |
| faceup? | none | Displays the face ID and X/Y angle of the accelerometer that is currently pointing up |
| save | None | saves the Dodeca task allocation to flash |
| dodeca? | None | Lists all the tasks on the system |
| dodeca |  | Sets the task information for the face that is currently facing up |
|  | n | Sets the name of the task |
|  | e | Enables the task |
|  | d | disables the task |
|  | a | Sets the Max task time |
|  | i | Sets the Min task time |
| config |  | Enters Config Mode. This puts the state machine into config mode. No other state can run unless you exit this mode |
| exit |  | Exist the Config mode and starts the state machine again. |
| dump | none | dumps the currently timed tasks to the terminal in comma delimited form |
| clear | none | clears the task timed data. |

The base code for the CLI is drawn from Elecia White’s example CLI code from Woko, however the following changed and features have been implemented

1. Use interrupts instead of polling
2. Multi level command structure
3. Actions to deal with control characters in the command string

## Task Time Management

### Task Assignment

Each face of the dodecahedron can be assigned a different task. This ranges from work tasks to entertainment tasks to meal and rest times.

Using the CLI, the user can assign tasks with the following parameters

* Task Name
* Maximum task duration
* Minimum task duration
* Task Colour
* Enable or Disable the task

A task , can be started by reorientating the dodecahedron so that the task face is facing up. This logs the start time.

A task can be stopped by either placing the dodecahedron on the STOP FACE or any of the disabled tasks.

## Configuration

The device allows for 12 tasks to be set up. The task configuration is saved to Flash.

A 2K area of flash has been reserved in the linker script at the end of the flash area

MEMORY

{

RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 128K

FLASH (rx) : ORIGIN = 0x8000000, LENGTH = 510K

CONFIG (r) : ORIGIN = 0x807E800 , LENGTH = 2k

}

.systemConfig (NOLOAD) :

{

. = ALIGN(4);

KEEP(\*(.systemConfig))

. = ALIGN(4);

} > CONFIG

The configuration uses the CRC peripheral in order to calculate the config checksum – This is used to determine of the saved config is valid or not. If not the default settings are applied and saved back to flash.

A pointer to the config area is defined in the Config controller code.

**\_\_attribute\_\_**((\_\_section\_\_(".systemConfig"))) **const** **char** systemConfigROPtr

The pointer is then set to a structure containing the definitrion of the configuration.

Once the tasks have been configured via the console a ‘save’ command must be used to commit the config to flash.

## Data logging structure

Data logging of each task is a simple array of a task struct. As, at this stage, there is only one task data type, the array is simply an array of the task data.

Each time a tasks is stopped, q task record data is created and added to the array.

When the data is ‘dummped’ there is a simple process to loop though the array and format the into a usable format.

The Device uses the time\_t type and mktime from ‘time.h’ to create a timestamp for the start end end times.

## Addressable LED’s

Design and Inspiration taken from <https://www.thevfdcollective.com/blog/stm32-and-sk6812-rgbw-led>

The driver used has been designed to cater for a large number of RGB led’s by using only two bytes within the DMA buffer to write duty cycles to the timers PWM channel. The driver uses the ‘pulse half complete’ and ‘pulse complete’ interrupts to move new data into the byte that has completed. This allows for a ‘double buffer’ type arrangement which allows for any number of LED’s to be used without the need to create a large PWM buffer to hold the entire ‘byte per bit’ PWM structure.

## System start up

The system start up initialises the HAL drivers and sets the device up.

Date and Time – As the RTC clock , at this stage, does not have battery back up (to be added), if the device is switched off the RTC information is lost.

Upon boot up the RTC date and time is checked and is not set the device is automatically placed into config mode with a message o say that the date and time need setting.

Sequence of Start up

1. Init HAL
2. Set System Clock
3. Init all configured peripherals
4. Switch off all LED’s
5. Initialise the System config
6. Read in config and reset/save if the config is invalild
7. Initialise the Dedeca tasks with the retrieved task config
8. Initialise the Data Store
9. Initialise the Console
10. Initialise the Accelerometer
11. Detect which face is currently facing up
12. Validate the current date
13. Start State machine based on the validity of the date and time.

The main loop simply calls :-

1. ConsoleProcess – to process any messages from the terminal
2. StateController – to run the state machine

## Task State machine.

The state machine is faily simple.