Implementing RTOS in self-leveling Stewart platform Intermediate Project

Sylwester Kubicki

January 26, 2016

Wrocław University of Technology, Faculty of Electronics

Intermediate Project, Witold Paluszyński

Abstract

Goal of the project [1] is to develop software for self-leveling Stewart platform based on RTOS using CubeMX. Project also includes researching available real time operating systems for STM32 microcontrollers and deciding which one most suitable. All procedures are divided into threads in order to make them work concurrently. Implemented software involves reading data from sensors (gyroscope and accelerometer), analysing raw data, fusing them into more reliable information about orientation. Finally by using inverted kinematics equations values for servomechanisms are calculated and set.

1 Introduction

1.1 Physical model

Physical model of Stewart platform was designed and constructed outside of the scope of this project. Detailed description of development is available in [1]. Image 1 shows the platform. It consists of:

- 6 servomechanisms
- STM32F3 Discovery evaluation board
- Construction made of Plexiglas

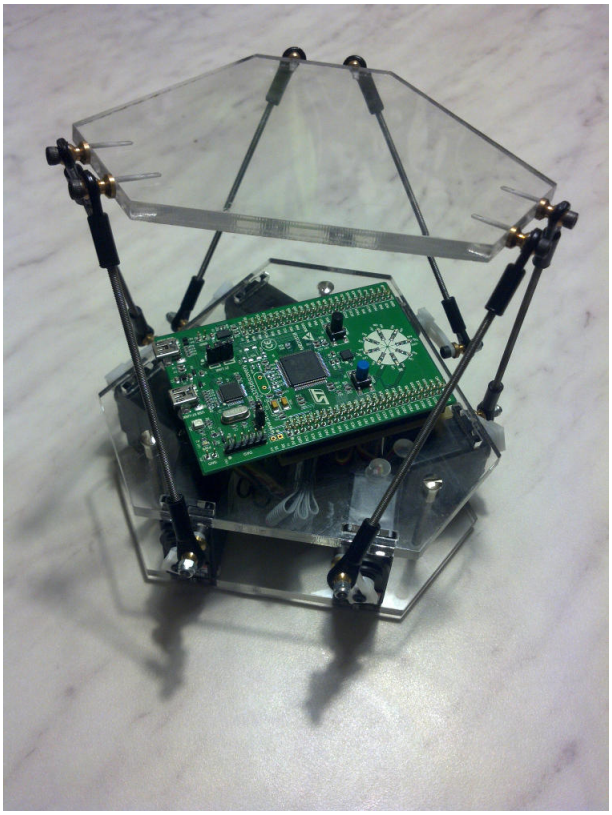

Figure 1: Platform used for implementation and tests

1.2 Assuptions and constraints

First main assumption of the project is to use available programs which provides versatile solution for developing software. Used programs should allowed to rapidly design and change the configuration of microcontroler without the need to use technical documentation in order to find correct values of registers and to avoid conflicts on pins.
Second main assumption is to make certain that all operations are real time. Those operations are:

- Reading data from sensors
- Fusing readings from sensors

- Calculating control based on inversed kinematics
- Providing terminal communication with user

It was decided that RTOS as a framework, should ensure concurrency of threads thanks to the system scheduler.
Last but not least main assumption is to develop described solution in a way that guarantees limitation of complexity in software structure and prevents defects during further development. In order to ensure that assumption additional focus was put on using abstract libraries:

- HAL - Hardware Abstraction Layer
- BSP - Board Support Package
- RTOS - Real Time Operating System

Thanks to that implementation is not dependant directly on hardware. Furthermore if there is a need to change the hardware (in order to increase performance or to lower the power consumption) little or no major changes in software are needed.

2 Configuration generation with CubeMX

Developing software for microcontrollers is a complex task which usually involves spending time browsing technical documentation searching for names of registers, correct values etc. This process can be greatly simplified with CubeMX software. CubeMX is a graphical creator for STM32 products. It's functions are based on Hardware Abstraction Layer, which can provide easy way to change microcontroller without significant alteration to code. More detailed information can be found in [2], [3] and official documentation [4].

It should be noted that can generate projects for

- Keil MDK
- IAR Workbench
- Atollic TrueStudio
- STM SW4STM32
- unofficially eclipse based IDEs

Three main aspects of CubeMX involve:

- Configuring GPIO port
- Configuring peripherals
- Configuring MiddleWares (FreeRTOS, FATFS, USB Device)
- Configuring clocks in graphical interface
- Power consumption calculations

2.1 Pins configuration

Designing pins configuration is essential during software development for microcontrollers. In many cases there are conflicts between pins in configaration. Number of processor outputs and assigned to them functions is limited. Because of that some configurations are infeasible. CubeMX provides simple, graphical solution in order to avoid described conflicts. It can be seen on figure 2. Status of each pin is marked in color:

gray Pin is free, not configured

green Pin is configured

orange Pin in not fully configured

yellow Pin has hardware implementation which cannot be changed

Names of pins correspon to the names in generated code. For example pins PD12-PD15 amd PC6, PC7 are connected to TIM3 and TIM4. Those pins are configured correctly and denoted with green color. In addition to graphical description of pins peripherals have als color description:

black Peripheral is not configured, not used

green Peripheral is configured correctly

red Use of that peripheral is infeasable due to configuration of other peripherials

yellow mark Peripheral is not configured, but can used in some restricted way due to the used of other peripherals

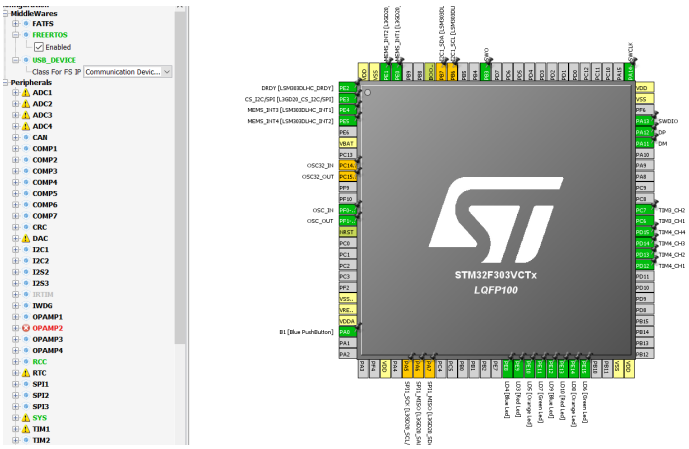

Figure 2: Setting pins function in CubeMX

2.2 PWM configuration

In order to drive the platform correct PWM signals should be generated and sent to servomechanisms. That is achieved with a use of two timers TIM3 and TIM3. Both of them have 4 channels each. Basic configuration of each timer can be seen on the figure 3.

Additional configuration is done in peripheral configuration which involves choosing settings for such elements like:

- prescaler
- counter mode
- counter period
- settings specific for each channel

Detailed image of configuration can be seen of figure 4

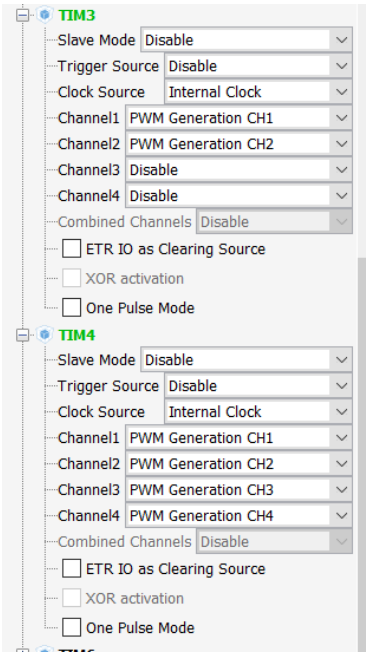

Figure 3: Configuring timers for PWMs

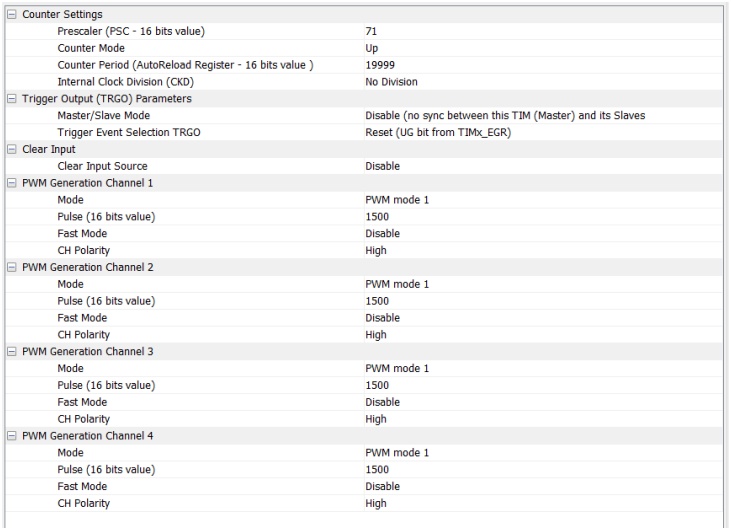

Figure 4: Configuring timers for PWMs

2.3 Clocks configuration

Another very important aspect of microcontroller configuration is clock speed selection. Each peripheral required specific frequency to work correctly. Core of microcontroller in

Discovery F3 board can operate at maximum frequency 72MHz. This configuration can also be done graphically in CubeMX. Based on peripheral configuration clocks table may slightly differ. In order to use HSE oscillator RCC needs to be activated. Then with the use of PLL and frequency prescalers desired configuration can be achieved. What is worth noting CubeMX will detect and highlight invalid configuration. After that will ask if the conflicts should be resolved automatically, this option usually works correctly and provide desired configuration without a need to use microcontroller technical specification.

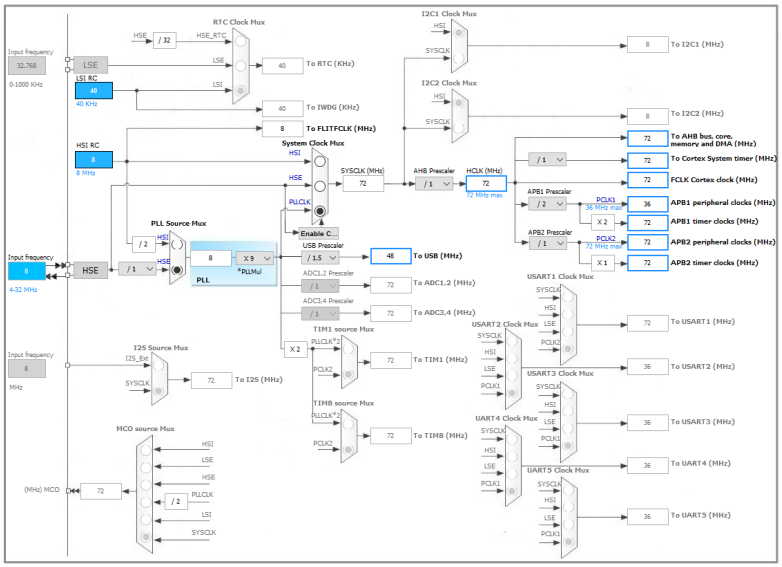

Figure 5: Clocks configuration

3 Board Support Package

Operation of fixed rotary Stewart platform is based on determining orientation of a base of the platform. It is done with readings from gyroscope, accelerometer, fusing those readings and using them in order to compute control for servomechanisms. That results in stabilizing upper platform in horizontal plane. Determining orientation of base platform is crucial. Therefore IMU needs to be implements. Discovery F3 board is equipped with gyroscope (L3GD20 [6]) and accelerometer with magnetometer (LSM303DLHC [7]). In order to use those sensors specified driver are needed. Those sensors are specific for Discovery F3 board therefore are not included in CubeMX configuration but are included in BSP [5]. It should be noted that BSP is specific for used equipment and some alterations are need to use different microcontroller or sensor. CubeMX dedicated package from STM32 [8] includes BSP drivers and examples how to use them correctly. Having project generated with CubeMX two folders should be added from STM32CubeF3 package:

STM32F3-Discovery gyroscope and accelerometer implementation for DiscoveryF3 sensors

Components implementations for numerous sensors including L3GD20 and LSM303DLHC

It should be noted that Components folder contains Common implementation for various sensors and peripherals like: accelerometer, gyroscope. thermometers, camera, lcd, audio etc. Each of those implementation acts as an interference and can be used in designing and developing drivers for sensors not provided by vendor.This way newly developed drivers

are compatible with BSP drivers and can be exchanged with relatively small impact on software structure.

4 FreeRTOS

4.1 Introduction

Concurrency of tasks is crucial in embedded solutions. Majority of embedded devices require some sort of response to the data in an instant, it deterministic time. Furthermore if the time window requirement is not met (either by doing something earlier or too late) some non-deterministic behavior can occur, which is undesirable in an embedded system. Real Time Operating System are optimized for those embedded/real time applications. The primary objective is to ensure timely and deterministic response to events which occur in the system. Using any RTOS allows writing an application as a set of independent threads that communicate with each other by message queues and semaphores.

There are many solutions available on the, some of them officially support STM32 devices, for others there are ports created. Some of those system are listed below:

- Free RTOS
- ECos
- Chibios/Rt
- CooCox
- ScmRTOS
- Nut/OS
- BeRTOS
- Contiki
- TinyOS
- RODOS
- RT-Thread

In most of them C++ can be used in order to create tasks out of a box, in others there are additional extension which allow to use it.

FreeRTOS is one of those operating systems intended for small microcontrollers. Most RTOS implementations offer the following in one way or another:

Tasks (sometimes called a process or thread) Independent piece of code that executes in its own context. OS should allow several tasks to run concurrently.

Scheduler Decides which task can run at a given time.

Semaphore Mechanism which allows communication and sharing resources between task in a safe way.

Timers Like other timers in system, it can be used to continuously call a function at a given interval or to time a single event.

FreeRTOS is used in this project because of it's well documented functions and possibility of configuration in CubeMX, which improves ease of portability of developed application to other devices.

4.2 Comparison of APIs

Because of vast amount of RTOS available of STM32 there is a generic CMSIS-OS wrapping layer provided by ARM in CubeMX. Thanks to that applications which use this layer can be directly ported on any other RTOS without any change in code, only the CMIS-OS wrapper has to be changed. This can make possible to investigate which of available real time operating system is best suited for chosen application.

FreeRTOS official API differs from API provided with the use of CubeMX (CMISIS-RTOS API). Detailed description of differences between those APIs is available in technical user manual from STM32 [9].

Example with the use of CMISIS-RTOS API:

```
ACCThread1Handle = osThreadCreate(osThread(accelerometer), NULL);
osKernelStart();
```

Example with the use of FreeRTOS API

```
xTaskCreate( vStartAccelerometer, "Accelerometer reading", 240, NULL, 1, NULL );
vTaskStartScheduler()
```

Both solutions can be used in developing with a use of CubeMX, but in case of standard API additional header files should be included:

```
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
```

Regardless of chosen API the final result is the same, created threads can communicate with each other and work independently. In this project CMISIS-RTOS API is chosen because of it's generic character and flexibility of changing underling RTOS to another without the need of altering the code.

4.3 Threads implemented during the project

Correct operation of self-leveling Stewart platform requires reading data from sensors, analysing those reading, determining control for servomechanisms and communication via terminal. All function are separated into independent threads which communicate via messages queues:

accThread Reading data from accelerometer

gyroThread Reading data from gyroscope

fusionThread Analysing reading from sensor (sensors fusion algorithm - Mahony)

controlThread Calculating and sending control signal via PWM to servomechanism

terminalThread Communicating via terminal (virtual com port), sending information about orientation

5 Summary

Nowadays developing embedded applications can be vastly simplified with a use of tools like CubeMX. Furthermore Developed software can be ported to another device with relatively small amount of work. What is more libraries like HAL and DSP are developed with a lot of care about quality and bugs are very rare. That makes software based on those libraries more reliable than software which operates on hardware directly without any API.
There are many RTOS available for STM32 devices. FreeRTOS is one of those systems. It is integrated in CubeMX as a middleware. Tasks, message queues can be declared with a use of creator, that improves portability to other devices.
Previously implemented software for self-leveling platform was divided into threads and can work correctly. It is important that any future changes into the design (both software and hardware) can be made much easier in comparison to standard one thread design in a loop.

References

[1] Self-leveling Stewart Platform - Sylwester Kubicki

[2] http://www.emcu.it/STM32Cube/STM32Cube.html#Tutorial

[3] http://empa.com/dokumanlar/STM32Cube-presentation.pdf

[4] http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/user_manual/DM00104712.pdf

[5] http://www.freertos.org/FreeRTOS-Plus/BSP_Solutions/st/STM32Cube.html

[6] https://www.pololu.com/file/0J563/L3GD20.pdf

[7] http://www.st.com/web/en/resource/technical/document/datasheet/DM00027543.pdf

[8] http://www.st.com/web/en/catalog/tools/PF260613

[9] http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/user_manual/DM00105262.pdf

[10] http://www.freertos.org/RTOS.html