Source code manual for
USB Configurable Servo Firmware

1. About

After configuring “Servo_Firmware.ioc” file and generating code with STM32CubeMX, some code is inserted in sections ”USER CODE BEGIN/USER CODE END” in some files. And all non STM32CubeMX generated source files are placed in “Servo_Firmware/Core/Src/App/” directory.

Software used:

STM32CubeIDE - version 1.11.0
STM32 MCU Package for STM32F1 Series – version 1.8.4

Source code project files:

1. File “Servo_Firmware.ioc”
2. Inserted code in sections ”USER CODE BEGIN/USER CODE END”
3. Non STM32CubeMX generated folder “Servo_Firmware/Core/Src/App/” and everything in it
4. STM32CubeMX generated files

2. Notes about “Servo_Firmware.ioc” file

NVIC Controller configuration:

NVIC | Code generation

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Prefetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
EXTI line1 interrupt	☑	0	0
USB high priority or CAN TX interrupts	☐	0	0
TIM2 global interrupt	☐	0	0
TIM4 global interrupt	☑	0	0
TIM3 global interrupt	☑	1	0
ADC1 and ADC2 global interrupts	☑	2	0
USB low priority or CAN RX0 interrupts	☑	3	0
Time base: System tick timer	☑	4	0

Interrupts “EXTI line 1” and “TIM4 global” are responsible to PWM input signal (position signal input to board) length measurement and timeout. They have equal highest interrupt priority, interrupting them with anything what takes significant amount of MCU cycles will lead to incorrect input signal length calculation, thus incorrect position or vibration.

Interrupt “TIM3 global” calls PID calculation on every overflow. PID calculation interruption with EXTI1/TIM4 interrupts could not cause any problem. Because TIM3 is runing in continuous mode and it’s countdown is going even while it’s interrupt handler is executing or when this handler is interrupted by another interrupt. Assuming that signal interrupts take very small amount of MCU cycles, handlers of these interrupts as well as PID calculation interrupt handler will exit timely before next “TIM3 global” interrupt occurs.

Next lower priority is USB, not sure how it’s interruption with all above interrupts can affect normal operation, but even anything such occurs, we will notice it by lost or incorrect data at MCU or PC side. I have not yet observed anything such.

Next and lowest priority has “Time base: System tick timer”. We use this interrupt for functions delay of which for some time will not affect system operation:

- Calling device input signal timeout function (indicates ”signal lost”)
- LED timeout function in position change mode
- motor timeout function when it is being testing from PC app (forward/backward rotation via PC app buttons)

ADC1 configuration:

ADC1 is used for detecting potentiometer position, by measuring voltage on it. As you can notice here, ”Vrefint” channel is not used for voltage calculation accuracy and we don’t actually calculate ADC input voltage at all. If there was any need in measuring of actual voltage to calculate position, it would have terrible drift without Vrefint taking in account. But because of input voltage to potentiometer is equal to microcontroller VDD voltage (not taking in account voltage drop on wires and any electrical noise on them), ADC results will always represent potentiometer position besides of actual VDD value.

3. Inserted code in sections ”USER CODE BEGIN/USER CODE END”

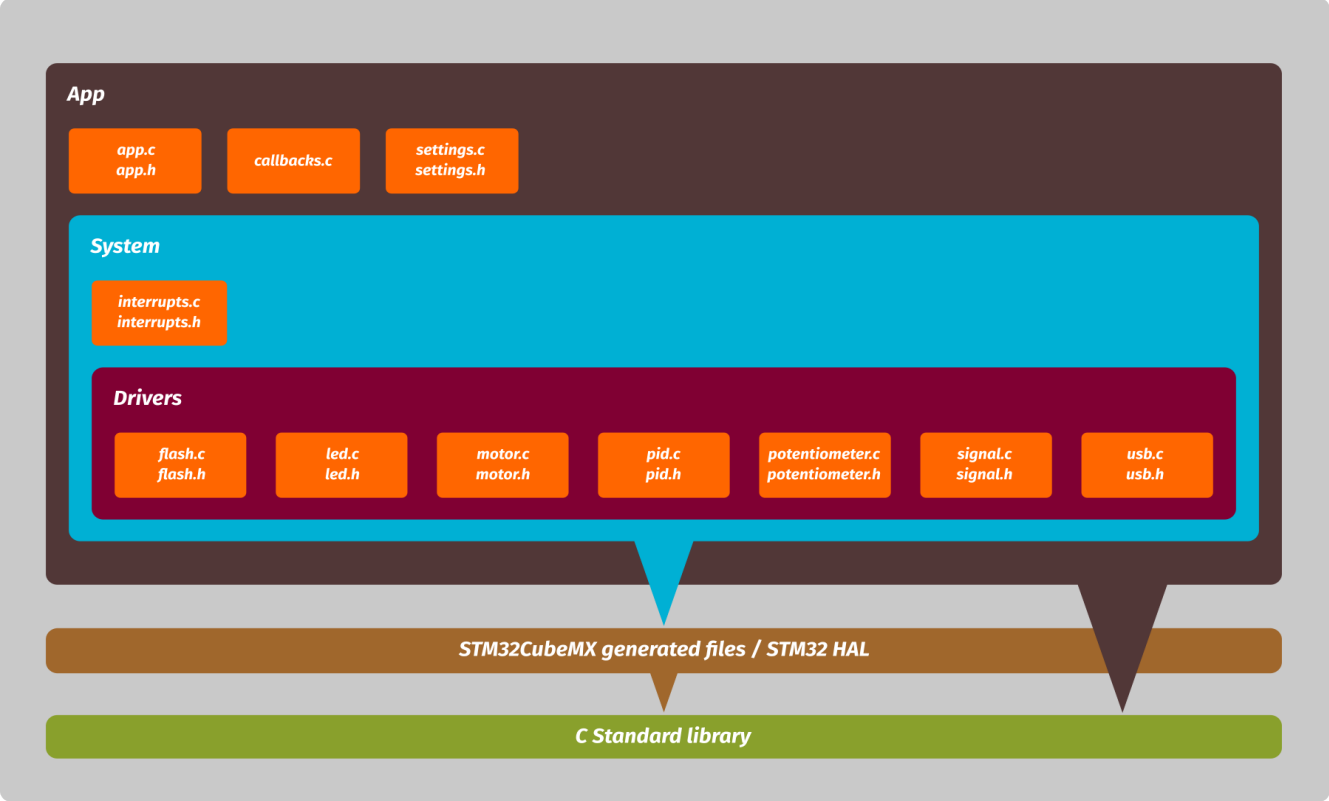
List of files and sections:

- File “Servo_Firmware\USB_DEVICE\App\usbd_cdc_if.c”
  - Section “/* USER CODE BEGIN INCLUDE */”
  - Section “/* USER CODE BEGIN 6 */”
    - Added only one line “USB_Packet_Received(&Buf[0], Len[0]);”
  - Section “/* USER CODE BEGIN INCLUDE */”
- File “Servo_Firmware\Core\Src\stm32f1xx_it.c”
  - Section “/* USER CODE BEGIN Includes */”
  - Section “/* USER CODE BEGIN SysTick_IRQn 1 */”

- File “Servo_Firmware\Core\Inc\main.h”
  - Section “/* USER CODE BEGIN Includes */”
- File “Servo_Firmware\Core\Src\main.c”
  - Section “/* USER CODE BEGIN Includes */”
  - Section “/* USER CODE BEGIN 2 */”
  - Section “/* USER CODE BEGIN WHILE */”

4. Code in “Servo_Firmware/Core/Src/App/”

Organization and dependency:



As general rule, in any folder starting from “App” all files might depend on other files from inner folder, but might not depend on files from any outer folder (unless dependency is out of ”App” folder, ST HAL library or ST provided USB middleware for example).

Everything inside “System” folder can depend on STM32CubeMX generated HAL library files, which also depends on C Standard Library. Files in “App” folder except any subfolders can only depend on C Standard Library and files from subfolders.

“Drivers” folder files:

Communication interface to c/h pairs in this folder is done by “extern” global variables in header files and functions which should be implemented externally.
Each c/h pair provides up to three data types and variables of that type:

- Configuration data type – XXXX_ctr_t
- Input data type – XXXX_i_t
- Output data type – XXXX_o_t

For example “pid.h” contains:

```
//Function should be implemented externally
void PID_Ready(float output); //called every time PID computation is complete

//Configuration variables
extern pid_ctr_t pid_ctr;
//Input data
extern pid_i_t pid_i;
//Output data
extern pid_o_t pid_o;
```

Frome here “PID_Ready” is further implemented in “callbacks.c”. For configuring PID we set variables in “pid_ctr” structure, we supply necessary input data for calculation into “pid_i” and we read output data and operation status from “pid_o” . All the necessary operations for each c/h pair to process these inputs and get output results are run by interrupts from “interrupts.c” or from “App_Loop” depending on requirements in specific c/h pair (written in funtion descriptions).

Special notes to “pid.c/h” pair:

I learned PID controller operation from Brett Beauregard’s blog, click HERE for link. Three different variants for each kp/ki/kd is my idea, also “split_1” and “split_2” variables. If you don’t found it useful just set all three “P”s same, “I”s same and “D”s same. Variables “split_1” and “split_2” sets the split points telling PID controller where to use which kp/ki/kd parameters.

For example if we set “split_1” to 1365 and “split_2” 2730. For PID input value from 0 to 1365 PID controller will use kp_1/ki_1/kd_1 for calculation. For input value from 1365 to 2730 will use kp_2/ki_2/kd_2. And for input value from 2730 to 4095 will use kp_3/ki_3/kd_3.

Operation mode “Proportional on Measurement” also is implemented and variable “pid_ctr.on_e_m” selects operation mode, PID_ON_E or PID_ON_M.

“System” folder files:

- Files pair “interrupts.c/h” implement functions for all STM32 HAL interrupts used in sources in “App” and it’s subfolders. Appropriate functions are called from “Drivers” folder source files for each interrupt depending on it’s parameters. Implementing all interrupts in single file makes code more portable to me

“App” folder files:

- Files pair “settings.c/h” provides global variable “settings_data” and functions for writing/reading from/to flash to it. Function “Settings_Apply” copies values from “settings_data” variable to appropriate global variables for every c/h pair in “Drivers” folder. Pair “settings.c/h” is used by “app.c”.
- File “callbacks.c” contains functions which are called by functions from “Drivers” folder at some specific events. For example “Signal_Received” when successfully received a signal on device input pin and calculated it’s length. Or “PID_Ready” for example, on PID calculation completion. Just separate file to keep “app.c” more readable
- Files pair “app.c/h” contain application main initialization function and single loop which are called in “main.c” (see paragraph 3 on page 3 “Inserted code in sections...”), everything else including interrupts are done in appropriate files. No other file depends on this pair, also pair does not provides any global variables accessible to other files.

Same design pattern applies to the simple “Servo_Tester” project too, included together with this firmware in repository. **For further information, function descriptions and step by step comments available in every source file.**