CoffeeMaker Simulation Project in FreeRTOS Embedded Systems

Author: Tom H.
Date: Sep. 2021

Abstract

I create this CoffeeMaker for the purpose to practice use of several features of FreeRTOS and STM32CudeIDE using an evaluation board called “F767ZI Nucleo” from STMicro. It is a continuing project whenever there is a need to fix, and improve its functionality and performance.

Summary

Hardware:
F767ZI Nucleo evaluation board
A small and simple prototype board containing switches, LEDs, a potentiohmeter, resisters and capacitors, and jumper wires to the F767ZI Nucleo.
Usb cable connect the F767ZI Nucleo to the host computer.

Software:
MAC OS, Windows, or Linux. STM32CubeIDE

Firmware:
Use STM32CubeIDE ver. 1.6 or 1.7. to create this project and download the .elf or .bin output file to the F767ZI Nucleo.

Evaluation:
Refer to the schematics to connect the jumper wires between the prototype board assembly and the F767ZI Nucleo via its onboard SWD interface.
To run the exercise open a screen or serial console session on the host PC. The serial communication parameter is 115200 Baud. Press the blue button to start the simulation. Exercise operation behaviors by pressing the buttons and adjusting the potentiohmeter and different times and sessions.

This simulation project strives to represent scenarios with some coffee maker operations. There is a cup size and regular/strong brew selection. The adjustment on potentiohmeter as a sensor can be used to simulate the water level detection and event report. After the power button is pressed at the ST_OFF state the event 'EVENTGROUPBIT_POWER' runs the state machine to the ST_CHECKWATER state as it starts looking at the water level sensor event 'EVENTGROUPBIT_FILLWATER'. Only if the water level OK signal is posted as an event the state machine transitions to the warmup state ST_WARMUP. Once the warmup timer expired this is followed by the cup size select state ST_CUPSELECT triggered by the 'EVENTGROUPBIT_CUPSELECT'. Next the state is transitioned to ST_BREW while the brew timer is started. Then the state goes to ST_SERVE. Finally it goes back to the ST_CHECKWATER state, again and again. And if the auto-off timer button was pressed the auto-off timer has started. In this case then this state or the ST_WARMUP state will be preempted by the event EV_TIMEROFF from the auto-off timer expiration. This will stop at the very first state ST_OFF. We can refer all these operations to the state diagram in the note.

Design architecture

The hardware design includes the use of the following on the STM32F767 –

GPIO mapped to LEDs and external interrupt for their inputs to the buttons.
STM32 UART is exported as a virtual RS232 visible to the host via USB.
The GPIO interrupt handler registers the Brew, auto-off, and power button press.

I made the prototype board such that the LED drive logic opposite to those LEDs on the Nucleo board. I use a simple hardware debounce RC circuit for the buttons.

The software design includes the use of FreeRTOS and STM32 MX peripheral library data structure and I/O HAL_ functions set. I use the following component from the FreeRTOS:

task scheduler, software timer, and event group.

I add an event driven finite state machine (FSM) to collaborate with these FreeRTOS components. Instead of using the state machine handler tasks are used to handle the event triggered by the GPIO external interrupt when any button press arrives asynchronously.

Software timers are used to simulate the time elapse for warmup, brew, and auto-off timer. As aforementioned there are 3 bits in the event group object created in the main().

There is a simple check for multiple button press simultaneously in the GPIO interrupt handler to improve the software robustness.

The usart channel exists to provide an auxiliary operation user interface. The state machine are kept simple and light weight. It is the tasks that do most of the works that collaborate with the state machine.

I like to partition the software with as many modules as I see it fits, reducing the individual module to have single and only relevant responsibility.

Summary of implementation:

Start with the `statemachine` module.
The `StateMachine_Run(stateMachine_t * operation, sm_event_enum_t event)`
is the major function that runs switches from one of the six states to another based on the event. It checks the current state given to and matches the event in the internal transition matrix of twelve state entries. If both are matched to any entry, the current state will be switched to the new state. The new state will be archived to be ready for the next new state later.

The FreeRTOS tasks are periodically checking the events from user buttons and water sensor. These inputs are used to decide which new state to go to. The events are driven by the GPIO input pins external interrupts in the `void EXTI15_10_IRQHandler(void)` to provide trigger for the FreeRTOS eventgroup `xEventGroupSetBits()` and `xEventGroupWaitBits()` library functionality. Since the wait function is a blocking call and thus it is in the FreeRTOS tasks instead of the state machine handlers. The main() function has a line:

```
xEventGroup = xEventGroupCreate();
```

The main() function creates six tasks. Each task either stays in blocked state for about half second if no work to do; or blocked by calling the `xEventGroupWaitBits()`. Once this wait returns the StateMachineRun() will be called.

The main() function also creates three timer callback background tasks once the timer is expired at the predefined period. Until users press the auto-off button the simulator will go back to start from the ST_Checkwater and run forever.

According to the FreeRTOS user guide to enable the FreeRTOS timer interrupt the FreerRISConfig.c is modified to have this line:

```
#define configUSE_TIMERS                        1
```

The prototype.c module for now is just simply used to drive my prototype board LEDs. Whatever future expansion or feature it is, I can use this module to handle it without touching the main.c.

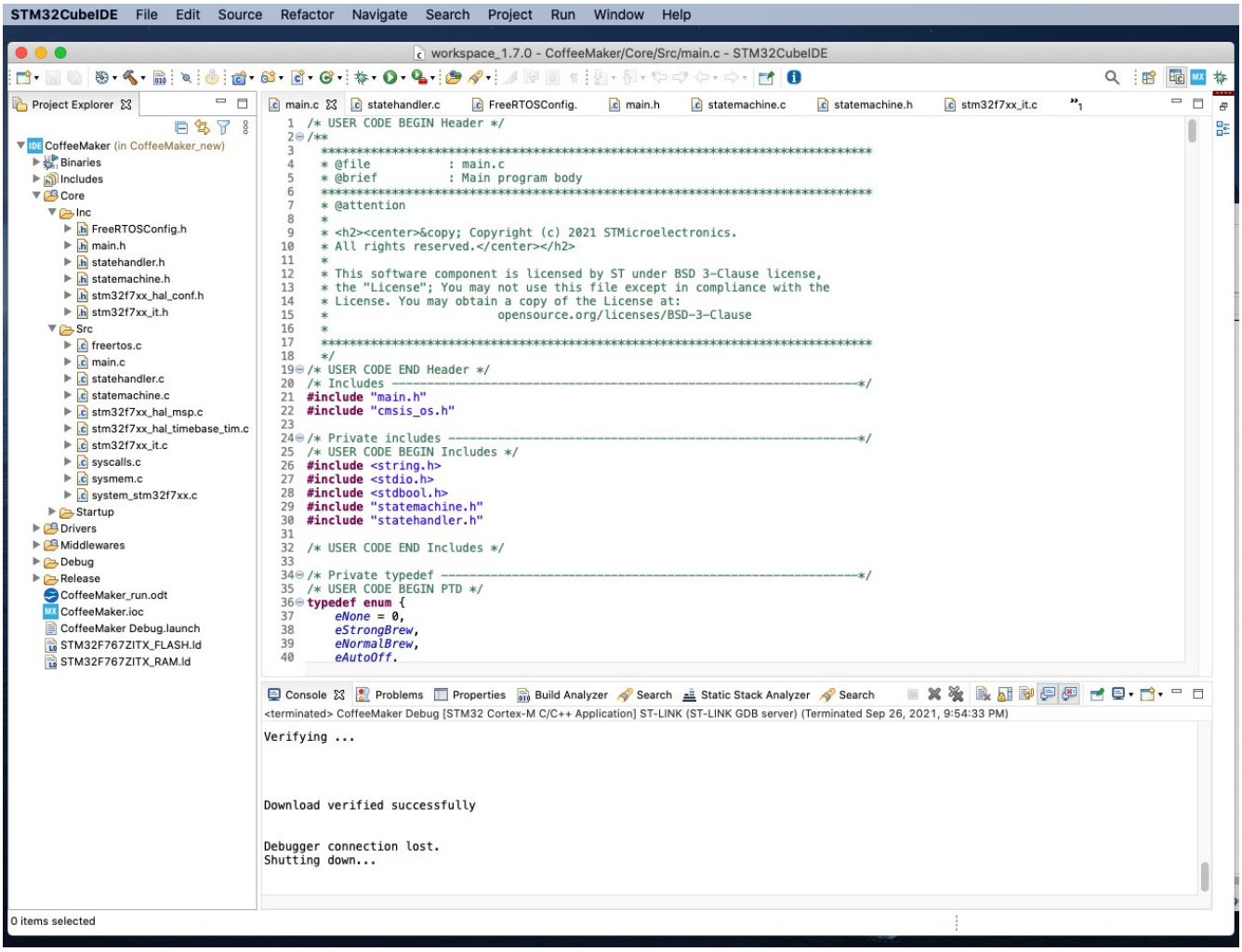
SysInit.c module helps most system related initialization. Likewise the usart.c module do its relevant works. I don't want to put everything in the main.c module.

The brew strength selection for now just drives the LED as an indication. It can be added later to simulate the making a stronger brew by extending the brew timer period.

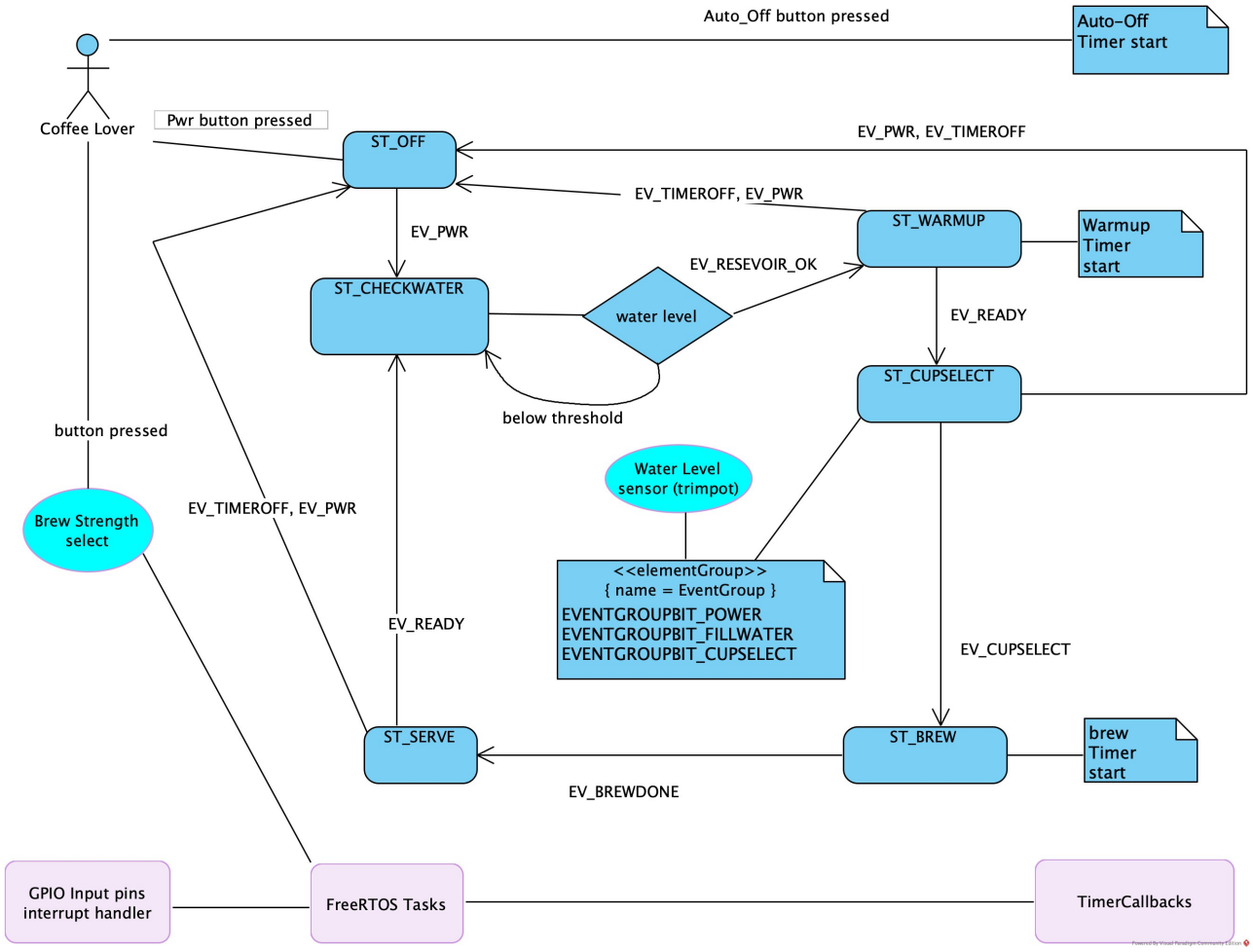
I use the Nucleo onboard ST-Link serial interface for Code download and debug with STM32CubeIDE.

I do this project based on the guidance of an embedded systems course by UC San Diego Extension. I also learned the state machine design and implementation from the original author at blog.mbedded.ninja website.

Figure 1. Run sessions


STM32CubeIDE File Edit Source Refactor Navigate Search Project Run Window Help
workspace_1.7.0 - CoffeeMaker/Core/Src/main.c - STM32CubeIDE
Project Explorer
CoffeeMaker (in CoffeeMaker_new)
Binaries
Includes
Core
Inc
FreeRTOSConfig.h
main.h
statehandler.h
statemachine.h
stm32f7xx_hal_conf.h
stm32f7xx_it.h
Src
freertos.c
main.c
statehandler.c
statemachine.c
stm32f7xx_hal_msp.c
stm32f7xx_hal_timebase_tim.c
stm32f7xx_it.c
syscalls.c
sysmem.c
system_stm32f7xx.c
Startup
Drivers
Middlewares
Debug
Release
CoffeeMaker_run.odt
CoffeeMaker.ioc
CoffeeMaker Debug.launch
STM32F767ZITX_FLASH.ld
STM32F767ZITX_RAM.ld
main.c
statehandler.c
FreeRTOSConfig.
main.h
statemachine.c
statemachine.h
stm32f7xx_it.c
/* USER CODE BEGIN Header */
/**
* @file : main.c
* @brief : Main program body
* @attention
* <h2> © Copyright (c) 2021 STMicroelectronics.
* All rights reserved. </h2>
* This software component is licensed by ST under BSD 3-Clause license,
* the "License"; You may not use this file except in compliance with the
* License. You may obtain a copy of the License at:
* opensource.org/licenses/BSD-3-Clause
*/
/* USER CODE END Header */
/* Includes */
#include "main.h"
#include "cmsis_os.h"
/* Private includes */
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include <stdbool.h>
#include "statemachine.h"
#include "statehandler.h"
/* USER CODE END Includes */
/* Private typedef */
/* USER CODE BEGIN PTD */
typedef enum {
eNone = 0,
eStrongBrew,
eNormalBrew,
eAutoOff,
Console
Problems
Properties
Build Analyzer
Search
Static Stack Analyzer
Search
<terminated> CoffeeMaker Debug [STM32 Cortex-M C/C++ Application] ST-LINK (ST-LINK GDB server) (Terminated Sep 26, 2021, 9:54:33 PM)
Verifying ...
Download verified successfully
Debugger connection lost.
Shutting down...
0 items selected

Coffee Maker Simulation State Diagram




Add Water
KEURIG