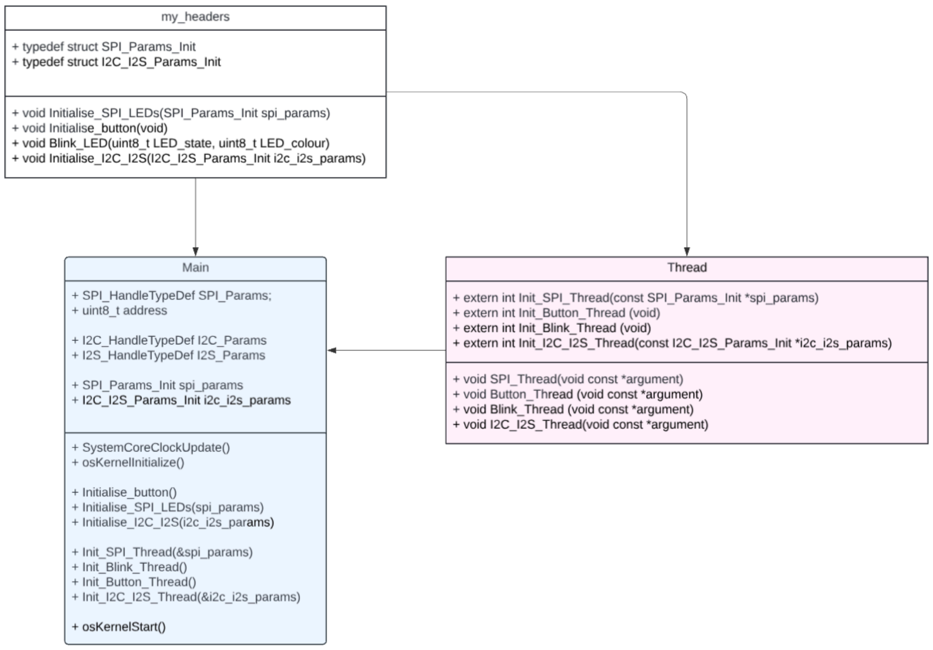
Overview

For this assignment, four threads have been created; the SPI thread, LED flasher thread, Button thread, and finally, the I2C and I2S thread. The Button thread will run concurrently with all other threads in the program, providing control over which thread runs at any given time. The execution of SPI thread will run when the program is initially uploaded. And the other threads will start working when the button is clicked. Here is a UML overview of the program implementation:


my_headers
+ typedef struct SPI_Params_Init
+ typedef struct I2C_I2S_Params_Init
+ void Initialise_SPI_LEDs(SPI_Params_Init spi_params)
+ void Initialise_button(void)
+ void Blink_LED(uint8_t LED_state, uint8_t LED_colour)
+ void Initialise_I2C_I2S(I2C_I2S_Params_Init i2c_i2s_params)
Main
+ SPI_HandleTypeDef SPI_Params;
+ uint8_t address
+ I2C_HandleTypeDef I2C_Params
+ I2S_HandleTypeDef I2S_Params
+ SPI_Params_Init spi_params
+ I2C_I2S_Params_Init i2c_i2s_params
+ SystemCoreClockUpdate()
+ osKernelInitialize()
+ Initialise_button()
+ Initialise_SPI_LEDs(spi_params)
+ Initialise_I2C_I2S(i2c_i2s_params)
+ Init_SPI_Thread(&spi_params)
+ Init_Blink_Thread()
+ Init_Button_Thread()
+ Init_I2C_I2S_Thread(&i2c_i2s_params)
+ osKernelStart()
Thread
+ extern int Init_SPI_Thread(const SPI_Params_Init *spi_params)
+ extern int Init_Button_Thread (void)
+ extern int Init_Blink_Thread (void)
+ extern int Init_I2C_I2S_Thread(const I2C_I2S_Params_Init *i2c_i2s_params)
+ void SPI_Thread(void const *argument)
+ void Button_Thread (void const *argument)
+ void Blink_Thread (void const *argument)
+ void I2C_I2S_Thread(void const *argument)

figure(1): UML design for the program

Level 1 - Creating a Dynamic Two-Axis Tilt Switch Interface

1. In the 'my_headers.h' file, a struct named 'SPI_Params_Init' will be created to hold SPI initialization parameters. This struct includes four variables, with two of them being pointers to track changes to these variables.
2. A thread with the main function named 'SPI_Thread' will be created in the 'Thread.c' file to accomplish the functionality of this task using an RTOS. Furthermore, the struct created will be passed as a parameter in the initialization function 'Init_SPI_Thread.'
3. Access the X and Y axes in the LIS3DSH, with X and Y as signed 16-bit integers holding XL, XH, YL, and YH values read from the corresponding addresses in the LIS3DSH datasheet. Set sensitivity values for X and Y at 500 and -500 for negative X and Y, this is because the acceleration typically ranges from -32768 to +32767. Specific LEDs will turn on and off based on the direction of tilt when the device exceeds these values.

4. Now, before the while loop in the main thread function for SPI, which is 'SPI_Thread,' the statement ‘osSignalSet(tid_SPI_Thread, 0x01)’ is included to enable the execution of SPI when the program is initially uploaded. Inside the while loop, the first line of code will be ‘osSignalWait(0x01, osWaitForever)’. This will cause the program to pause if the push button is clicked, as the flag associated with this thread is cleared.

Level 2 - Control with Push Button

1. Two threads will be created in this section, each with main thread functions named 'Button_Thread' and 'Blink_Thread' in the file 'Thread.c'.
2. For the blinking thread, a red flasher will be created for 0.5 seconds. The first line of code will include 'osSignalWait(0x01, osWaitForever)' to control its operation using the button.
3. We need the push button to work as a latch switch, maintaining its current state until intentionally toggled. I've implemented a global variable 'flag' with an initial state of zero. Clicking the button with flag at 0 triggers one function and sets the flag to one; clicking with flag at one triggers another function and resets the flag to zero, this will make the button work as a latch switch.
4. When the button is clicked with a state of zero, all LEDs will turn on for one second without the need for a debouncing delay. We will then stop the SPI thread by clearing its flag and set flags for the blinking, I2C, and I2S threads.
5. If the button is clicked with a state of one, a 200-millisecond delay will be added to remove the debouncing effect of the button. The flags for the blinking, I2C, and I2S threads will be cleared, and the SPI thread flag will be set.

Level 3 - Generating a Beep Sound using CS43L22

1. Similar to level one, a struct will be created to hold the initialization parameters for I2C and I2S. This is because the STM32F4 controls the audio DAC through the I2C interface and processes digital signals through the I2S connection or analog input signal.
2. Based on the datasheet of the CS43L22, specifically in section 4.11 titled 'Required Initialization Settings', the implementation of the CS43L22 initialization settings using I2C involves writing these values to the crossbounding addresses of CS43L22 through the 'HAL_I2C_Master_Transmit' function. Similarly, in section 7.17, Beep & Tone Configuration (Address 1Eh), 0xC0 will be written to register 0x1E to configure a continuous beep occurrence. The same applies to setting the 'Power Ctl 1' register (0x02) to 0x9E.
3. Finally, a thread with the main function named 'I2C_I2S_Thread' will be created in the 'Thread.c' file to transmit the data of the square wave through I2S to generate sound using the function 'HAL_I2S_Transmit_IT'.