Application Description:

Reaction Game! is what this project is all about. Pushing the user button starts the game. Based on the displayed color the user must rotate the board in the specified direction. This action if not performed within a specific time or, moving in the incorrect direction leads to the end of the game.

For example, if red color is shown then move along Y in anti-clockwise (left) direction, if green is shown then the other direction (clockwise / right). If you do not in a specified time or, in the wrong direction, the game is over.

Hardware Description:

The project used STM32F429I (RevD) Discovery Board.

It used the following on board components,

Gyroscope (I3G4250D) – to get the direction

Display (ILI9341) – to show color and direction, as well as other user instructions

Temperature sensor connected internal to ADC – (I was initially going to use a speaker and generate sound during the game but other things came up at work so switched to using this one as my 3rd peripheral) – used it to select the next direction of rotation – so that the direction becomes random.

USART for serial interface (needs further enhancements)

LEDs – to display the direction to go in

Software Description:

The project uses bare-metal single control loop to run.

The state machine described here in the homework is pretty close to what I ended up implementing, with some exception of naming and other minor variations of actions taken.

After the initial set-up is performed, the game begins in the GAME_STATE_WAITING_START state, waiting for the user to push button, once input is received (button interrupt driven) it moves to the next state - GAME_STATE_SELECT_DIRECTION where it reads the ADC to get the current temperature and compares it to the last temperature recorded. And based on the comparison it picks the next direction to rotate, so that there is a randomness to what the next direction will be. (This logic can be made more interesting than what it currently is – playing around with the raw ADC value perhaps? Some bit manips, maybe?). Based on the direction it either goes to

GAME_STATE_SETUP_CW or GAME_STATE_SETUP_ACW and does the setup which involves – lighting up the correct colored led, changing the display background, showing the correct text, and starting the timer to denote how much time the user must rotate the board in order to stay in the game.

Next, it goes to GAME_STATE_WAITING_CW or GAME_STATE_WAITING_ACW depending on the direction picked earlier, reading the gyroscope co-ordinates to see if change is detected, if within tolerance and in the correct direction. The time remaining is continuously updated on the screen. Failure to receive the correct direction before timer runs out takes us to GAME_STATE_OVER where final score is displayed and it goes back to GAME_STATE_WAITING_START to restart the game. On the other hand, if valid input is received it goes back to GAME_STATE_SELECT_DIRECTION and the game continues.

Gyroscope: Interface for reading the gyroscope coordinates, also contains the calibration routine. All written on top of the BSP code from ST.

Moving Average Filter: Used to reduce noise in readings. This has been written long back for my personal projects, just reused here. My 'work sample' code if asked for, actually.

Circular Buffer: To store new readings when calculating the moving average. This has been written long back for my personal projects, just reused here. My 'work sample' code if asked for, actually.

Temperature: This is the interface for reading the ADC1 temperature sensor.

Display: LCD interface on top of the BSP functions provided by ST. It has a table of 'attributes' so to speak, for, the text, coordinates, font, etc. that should be used when displaying each message.

Game: Contains the state machine implementation and core game logic.

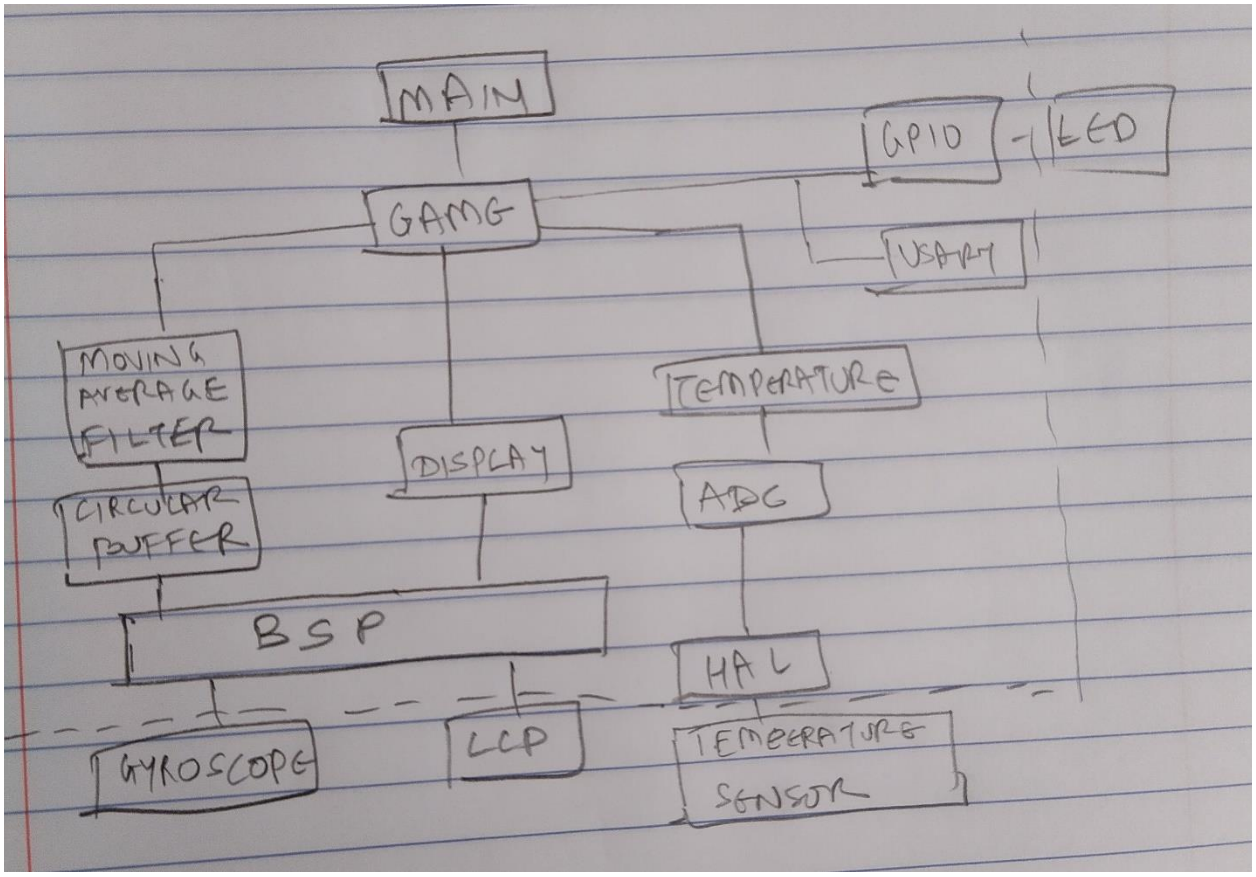
Re-used code:

ConsoleIO from here.

HAL / BSP – provided by STM with licensing information here.

Diagram(s) of the architecture:

Here is what I roughly ended up with:



Build Instructions:

Hardware: I did not use any external components so, all you need is STM32F429I-DISC1 development board.

Software: The application code can be found here. It uses stm32cube framework. I used platformio / vscode as my development environment.

The one thing to note is that, I had a RevD kit so had to ensure USE_STEM32F429I_DISCOVERY_REVD was #defined. This is done in platformio.ini. Also, the drivers need to be v1.27.0 for this to work correctly; found here.

(I have other installation instructions stashed somewhere locally – update later)

Debug and Testing:

My primary tool for debugging was the built-in debugger in vscode, using breakpoints and watch windows.

It took me a while to get my clocks correct and the timer interrupt to fire when expected.

Another, 'why does this not work?' type of debugging was after I configured my clocks, I did something on the serial port only to see that it had stopped working. I had no idea why would clocks have any effect on the serial port output. The issue was I was using a pointer without dereferencing it in a completely different place, except that I tested two different things at the same time and led to spending time debugging the broken (not broken) serial port.

As far as testing goes – a lot has to be done to this end, especially the false cases, what happens on other axes, random button presses. Should we run gyro cal routine more often? Say if we deviate more form a certain tolerance?

Power:

Powered using on-board USB. I also experimented with an external power supply, and that worked well too. In the future, given that this is more like a handheld game, we can start with couple of AA batteries.

Future:

What would be needed to get this project ready for production?

Hardware: Design board with just the bare essential components. Optimizing power supply. I think we can move down as far as the processor core is concerned, nothing as fancy as a 4 needed here.

Software: Need to create a better build system! Move away from the IDE as it will give me more control and consistency. It should make the job easier when upgrading to use newer BSP/HAL packages. Also, doing a clean installation will be simpler.

Are there other features you'd like?

Oh, yes! One too many, probably.

So, like I said previously, audio is probably going to be the first thing that I would add.

Firmware Update comes next as I've done bank-based swapping before so I think it is probably going to be all about porting that code over – as parsing and writing the .hex file is going to be the same for most part – primary work is going to be around editing linker file to ensure storage area is correctly defined, swapping code runs out of RAM, and most importantly, flash read / write works optimally.

Which brings me to the next one i.e., the need for enhancing the serial interface, as the pieces above will need to be tested independently.

And the other big-ticket item that I do want to add is syncing 2 boards with BLE to allow multi-player configuration.

Other minor additions are around user interfaces to make the system more interesting - configurable time based on typical ‘game levels’, allowing user to pick colors. The touch screen functionality can be utilized for some of these user interactions.

Grading:

Project meets minimum project goals: 1 – Barely meets most requirements. Command line interface should have been more enhanced. Should have used more external peripherals instead of all on-board as the joy (read pain) of debugging hardware is unmatched.

Completeness of deliverables: 1 – Submitted all work that I have completed.

Clear intentions and working code: 2 – Video, report, and the code does line up.

Reusing code: 1 – Reused HAL/BSP code from STM, rest is mine.

Originality and scope of goals: 0 – used ideas from the class presentation

Power analysis – Had done my week 10 homework – not comprehensive but still did put in some work, here.

Version Control – the use here could have been easily much better – could have used branching model. The uart issue that I mentioned above did come to bite me later so have learnt it the hard way (embarrassed to say, but not for the first time).