Making Embedded Systems
by Elecia White

Final Project Report: Remote Controller for UVC LED LIGHTS

Contents

Contents 2
Introduction 3
Part Selection 4
Hardware Block Diagram 6
Software Block Diagram System overview 7
Remote System software overview 8
Local System software overview 9
State Machine Implementation 10
Conclusion 11
Self-assessment 12
References 13

Introduction

I'm not exactly sure what to call this project, I will try to explain the application and hopefully this should clarify the project design as well as the feature associated with the project. The project idea came as a result of the SARS-CoV-2 pandemic. Much earlier at the beginning of the pandemic there were a lot of information regarding virus transmission by surface contacts. There were also varying information on how long the virus could survived on surfaces. Me being a curious person by nature, I felt compelled to research existing approach for surface decontamination. One approach that stood out to me was using UV light as a way to destroy viruses on surfaces. I then proceeded to check amazon for existing products and there were plenty of options available. I still felt the need to design my own system with specific feature in mind, because of course I can make it much better!

Project Features:

- Local and Remote system.
- Local and Remote system should communicate wirelessly.
- Local system feature:
    - Button to arm the remote system.
    - Status indication to that system is arm.
    - Need to be mountable on a door.
    - Need a sensor to detect when the door is open when system is armed.
    - Need to be flexible so can be moved to different room.
    - Battery power to allow flexibility of movement.

- Remote system features:
    - Power over USB with a 5V adapter.
    - Need an IO to control UVC lights.
    - Need to listen to Local system for incoming commands.
    - Needs to disarm the system if communication with local system is lost.

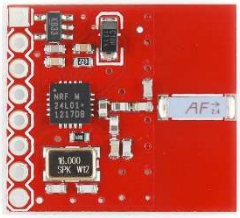
Rationale for chosen features:

UV light specifically UV-C light occupying the wavelength of 200 to 280 nm is very effective at destroying pathogens and also unfortunately can cause skin and eye damages to both human and pets. Due to this reason I had to choose some feature that would allow me to turn on the Light remotely, as well as prevent accidental exposure to the lights for pets and human being.

Part Selection

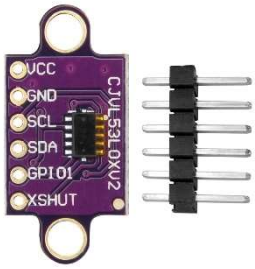
To accomplish he project feature the following parts were selected:

1. nRF24L01+ Transceiver breakout board from spark fun.
   Features:
   - Operation band: 2.4GHZ ISM band
   - Speed: 250kbs,1Mbs,2Mbs on air rate
   - 126 RF channels
   - SPI interface
   - Electrical rating:
     - 11.3mA TX on 0dBm output power, 13.5mA RX at 2Mbs air data rate.
     - 900nA in power down
     - 26uA in standby-1
     - 1.9 to 3.6V supply range



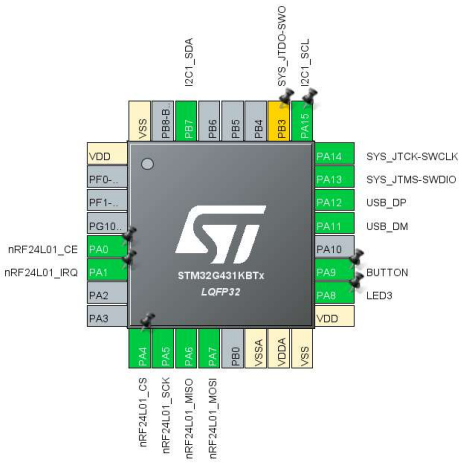
2. VLX53L0X Time of Flight(ToF) sensor:

   - Measure ranges up to 2m
   - Class 1 laser device
   - I2C interface
   - Programable I2C address
   - Electrical rating:
     - 2.6 to 3.5 V supply range
     - 19mA active range average consumption
     - HW standby 3 – 7 uA, typical 5uA, SW standby 4 – 9 uA, typical 6uA



3. STM32G431KBT6 microcontroller:

- Core: Arm® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator) allowing 0-wait-state execution from Flash memory, frequency up to 170 MHz with 213 DMIPS, MPU, DSP instructions
- 128 Kbytes of Flash
- 32 KB SRAM
- USB 2.0 full-speed interface with LPM and BCD support
- 3 x SPIs, 4 to 16 programmable bit frames,
- 3 x I2C Fast mode plus (1 Mbit/s) with 20 mA current sink
- 1 x LPUART
- 4 x USART/UARTs (ISO 7816 interface, LIN, IrDA, modem control)
- Single precision FPU
- Electrical rating: VDD, VDDA voltage range: 1.71 V to 3.6 V

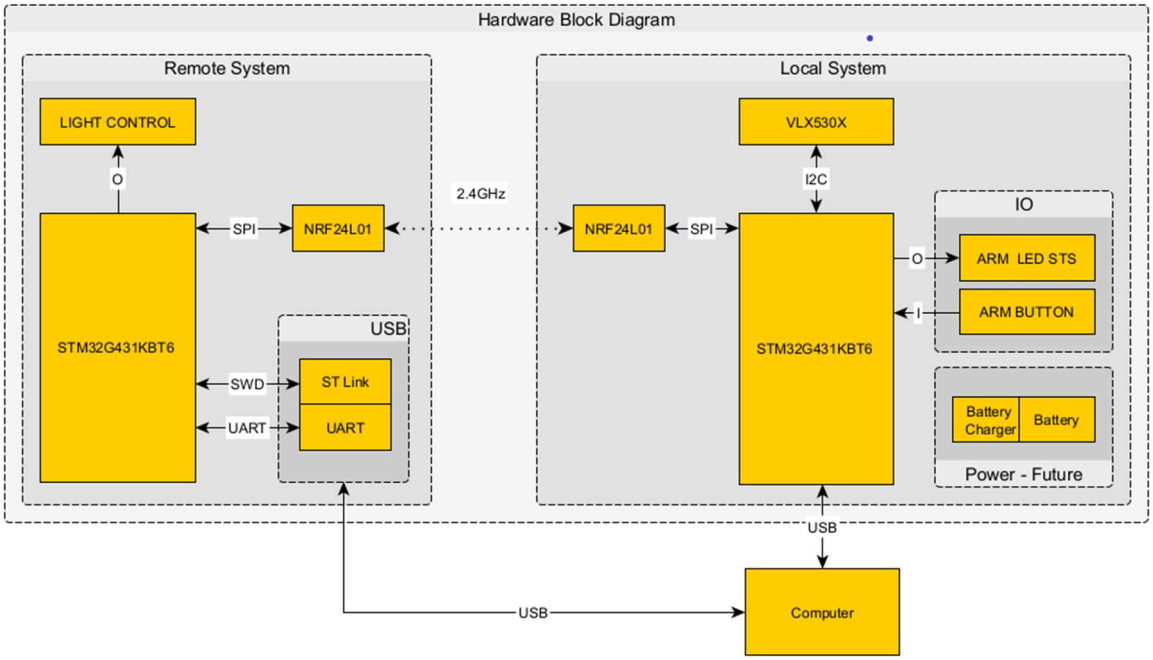


Hardware Block Diagram

Shown below is the hardware design block diagram showing how each module interface with the Microcontroller (MCU).

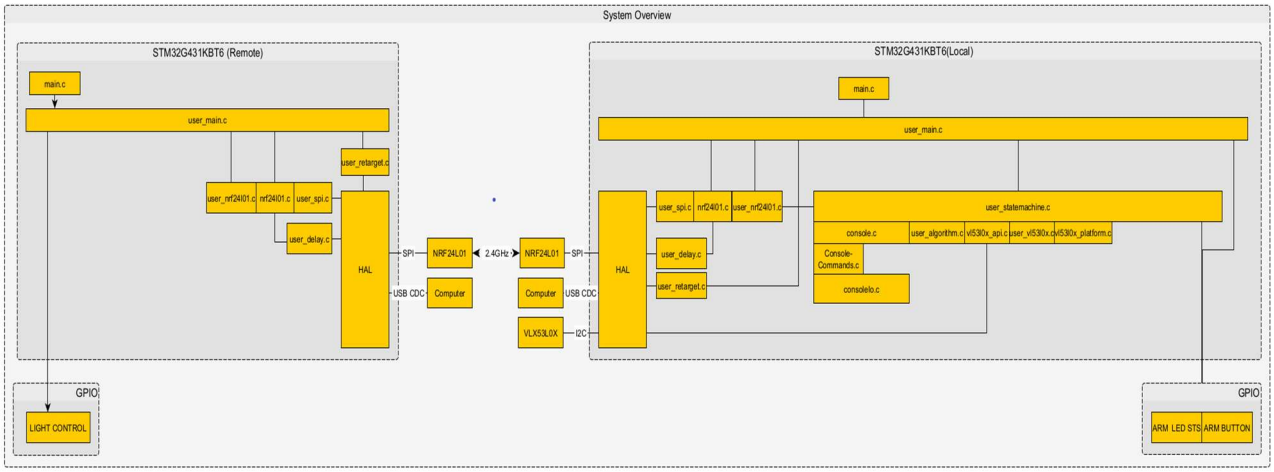
On the local system, the user interacts with the local system using the ARM Button and ARM LED Status through GPIO. To arm or disarm the system, the MCU sends commands over SPI to the NRF24L01 transceiver. Once the system is armed, the VLX530X TOF system is used to detect when the door is open through I2C. For debugging on the local system, the USB interface is used to communicate with a computer using the USB CDC class.

On the remote system, the MCU monitor for commands coming from the local system through the NRF24L01 through SPI. The MCU then enable or disable Light through an output GPIO. For debugging purpose, the UART is used to communicate with the PC through the ST link debugger on the Nucleo-g431kb board running as the Remote system.



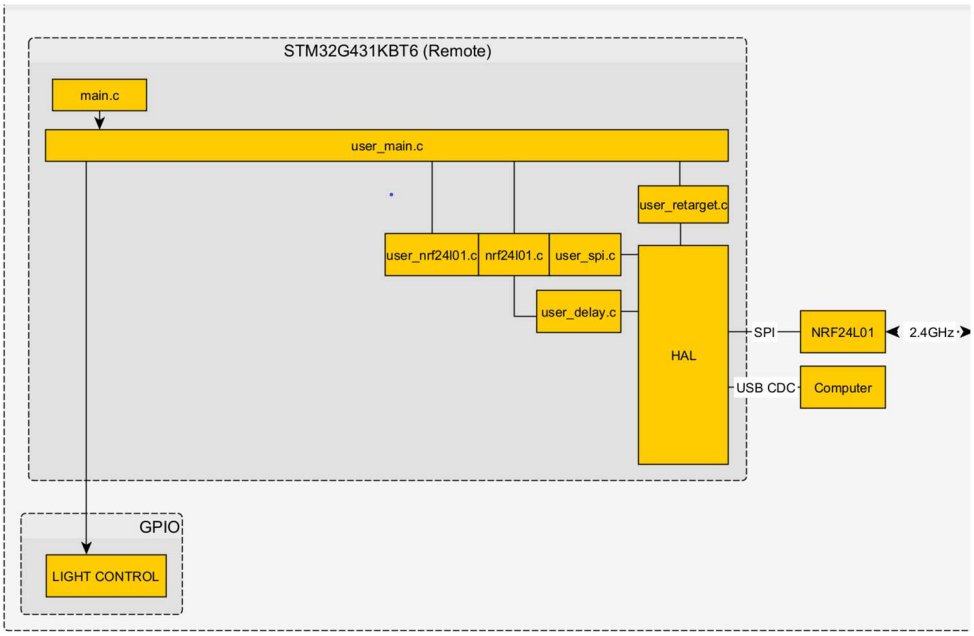
Software Block Diagram System overview

To actualize the described system so far, the following system overview was created to show how each module fits into the entire system. There was an emphasis on the interaction between function call between module, lesser attention was given to data type shared between modules.



Remote System software overview

Here's a detail overview of the remote system software design overview

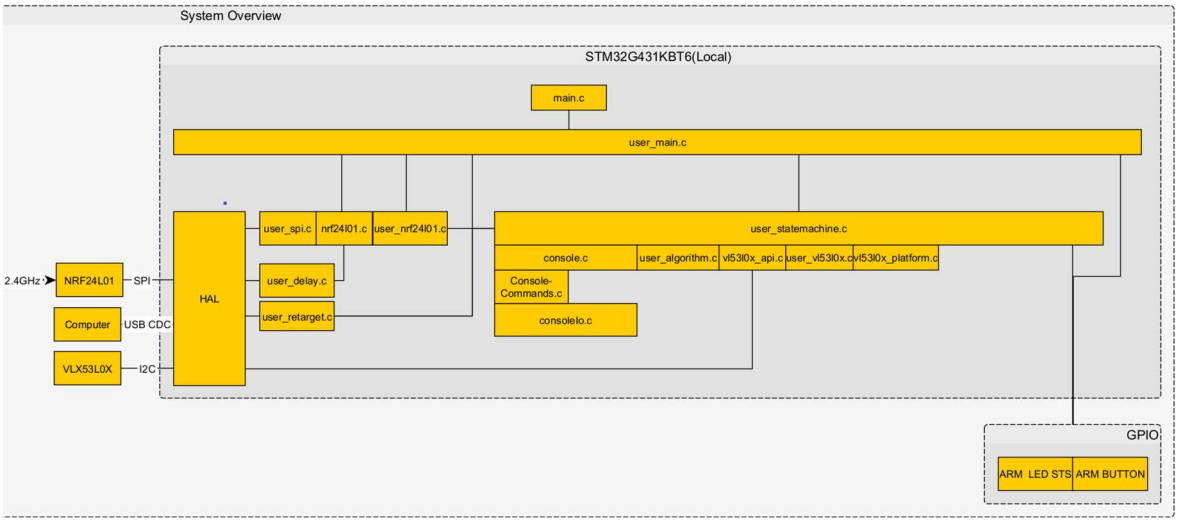


Description of files:

1. main.c : This is the entry point into the program. From here user_main.c is called this is to allow isolation between STM32CubeMX project and the user defined projects files that are not auto generated.
2. user_main.c: this is where project while loop exist and call to the rest of the system. In this file the following tasks are being taken care off: Reading data from the local system, monitoring for heartbeat when system is armed, enabling and disabling light control when command is received from the local system.
3. user_nrf24l01.c: this where project specific function resides to communicate with the nrf24l01 module. This file makes call to the nrf24l01.c file which actually takes care of the low-level details.
4. nrf24l01.c is a library I'm using to communicate with the nfr24L01 module, the library provides all necessary function to set and configure the nrf module, including hardware abstraction to map the SPI peripheral and GPIO to the library.
5. user_spi.c and user_delay.c this two are supporting module for the nrf24l01 library. They allow the library to read and write data to spi, and provide hardware timer with micro seconds accuracy.

Local System software overview

Here's a detail overview of the local system software design overview



Description of files:

1. The nrf24l01 interface described above for the remote system applies here as well, the code was reused as indicated in the diagramed above.
2. The user_statemachine.c: this is called by the user_main.c file as shown above, I will provide more information below in describing the state machine.
3. console.c this module is called by the state_machine at a regular interval, this is the entry point to the debugging console. This calls both the consoleIo.c and consoleCommands.c to respond to user commands from terminal.
4. consoleCommands.c this is where the commands from the terminal are mapped to function pointer. This allows actioning to request from user.
5. ConsoleIo.c this is the interface between the actual hardware incoming and outgoing command request are processed here.
6. user_algorithm.c : this contains a set of function to calculate a running average and update a fifo of unsigned 16 bit array of eight element.
7. user_Vl53l0x.c: this is used to initialize the TOF sensor before usage.
8. Vl53l0x_api.c, Vl53l0x_platform.c are part of ST micro library for the TOF sensor. They are used to read data from the sensor over I2C.

State Machine Implementation

A Table-driven state machine was implemented for the local systems, shown below is a state transition table describing the states, events, next state and condition in which each state operate.

#	STATE	EVENT	NEXT_STATE	CONDITION
1	ST_INIT	EV_IDLE	ST_IDLE	START
2	ST_IDLE	EV_ARM	ST_ARM	ARM_CONDITION
3	ST_IDLE	EV_CHECK_RX	ST_CHECK_RX	NOT_ARMED
4	ST_ARM	EV_POLL	ST_POLL	ARMED
5	ST_POLL	EV_DISARM	ST_DISARM	DISARM_CONDITION
6	ST_POLL	EV_UPDATE_HEARTBEAT	ST_UPDATE_HEARTBEAT	ARMED
7	ST_UPDATE_HEARTBEAT	EV_CHECK_RX	ST_CHECK_RX	ARMED
8	ST_CHECK_RX	EV_DISARM	ST_DISARM	DISARM_CONDITION
9	ST_CHECK_RX	EV_POLL	ST_POLL	ARMED
10	ST_CHECK_RX	EV_IDLE	ST_IDLE	NOT_ARMED
11	ST_CHECK_RX	EV_ARM	ST_ARM	ARM_CONDITION
12	ST_DISARM	EV_IDLE	ST_IDLE	ARMED

To elaborate on the state machine, the condition column determine which events can occurred in each define states. For example, while the system in not ARMED we ensure that no polling of the time-of-flight sensor or update heartbeat is sent to the remote system. The only thing we do in NOT_ARMED state is poll for user console commands and going back to idle.

Conclusion

This has been an amazing course so far; I've pushed myself further as far as coding experience and being more comfortable reading other people codes and importing external library from other sources. I have a pretty good idea where my skill set stands, I hope to push further in improving my coding ability and spending more time designing system then coding them.

Project specific there's still a lot to do, I ran into some issue with getting the transceiver to send a request for data and received it. I spent some time to troubleshoot but believe I was running out of time. This is something I would like to implement. I believe that would probably be a good candidate for a finite state machine. I still need to design the local system remote switch for battery power. I have some good idea how to approach this and which steps to take next.

I have some work I need to do to work on updating the code to non-blocking delay. I've implemented the function and have tested it on the remote system to keep track of the heartbeat timeout. I would like to replace the blocking the delay in the code with non-blocking.

I still need to design the hardware as well. All in all, plenty of work to do still!

Self-assessment

CRITERIA	GRADE	COMMENTS
Project meets minimum project goals	2.5	I have implemented a state machine, button interrupt, I have 2.5 peripheral 2 spi transceiver a 1 I2C sensor. I don't know if it counts sinsce they're both spi.
Completeness of deliverables	2.0	Code is readable on its own, without the report. I believe the report might be lacking plus I'm submitting this very late.
Clear intentions and working code	3	The system work as intended I still need to work on making the device low power.
Reusing code	3	Code has been reused where applicable. There's room for refactoring.
Originality/Scope	2.5	I like my approach to the problem solved.
Bonus:	0	Described, has graphs, and is accurate
Bonus: Version control was used	1	I do have some versioning.
TOTAL		A total of 11.5

References

Resources for the VLX53L01 TOF sensor

Interfacing with the TOF sensors:

Link with help discussion thread on the VLX53L01

https://community11.st.com/s/question/0D50X0000AtiU4VSQU/how-to-add-vl53l0x-drivers-to-stm32-project

Digikey forum Link: Adding the VL53L1X Driver to an STM32Cube Project

This link provided some help in getting communication with the module. Some of the information was not relevant but this still provided some needed help.

Provided some help in getting USB CDC Device working:

https://controllerstech.com/usb-cdc-device-and-host-in-stm32/

Resources for the NRF24L01 Transceiver

Very helpful and detail tutorial on working with the NRF24L01 for the LPC2148. I repurposed this tutorial to allow working with STM32.

http://blog.diyembedded.com/

Resource for finite state machine implementation: A Function Pointer Based State Machine

Resource for implementing a command interface console:
https://github.com/eleciawhite/reusable