

# MESH NETWORK

**COMMUNICATION**

**Supervisor: Sandra French**

## TEAM MEMBERS

Swathi Thekkuveettil Reshma Kallungal Paul Manisha Arora

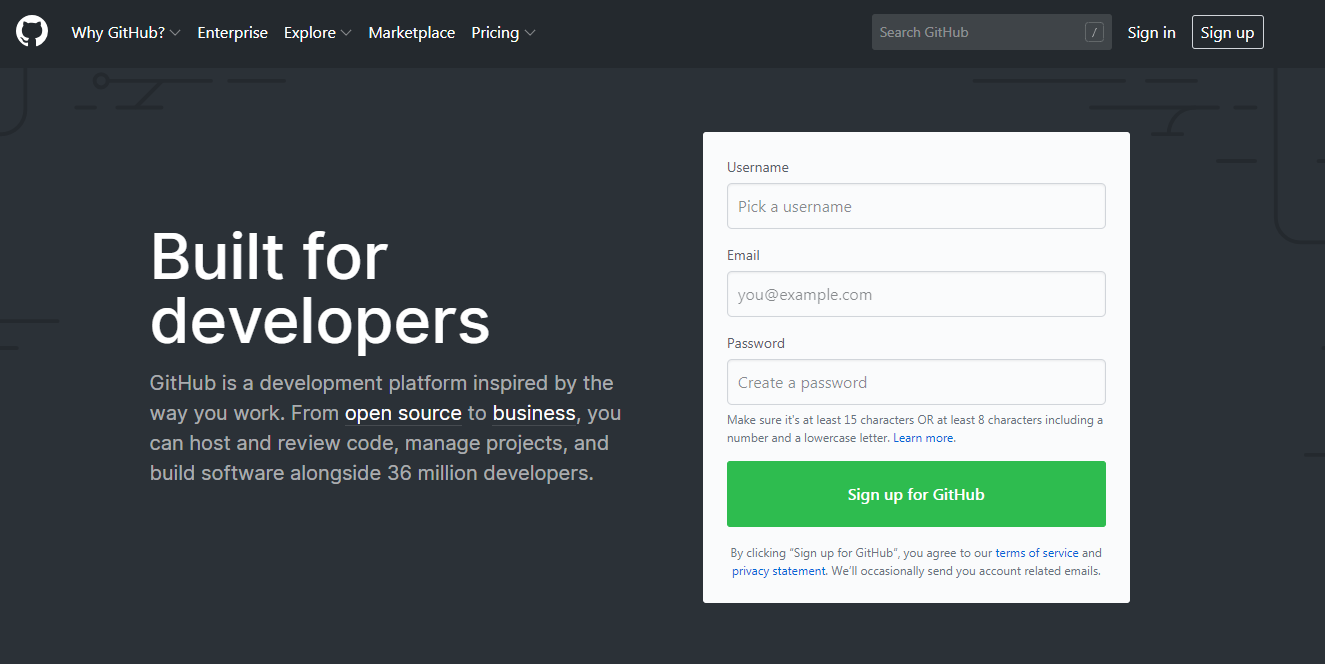
## INTRODUCTION

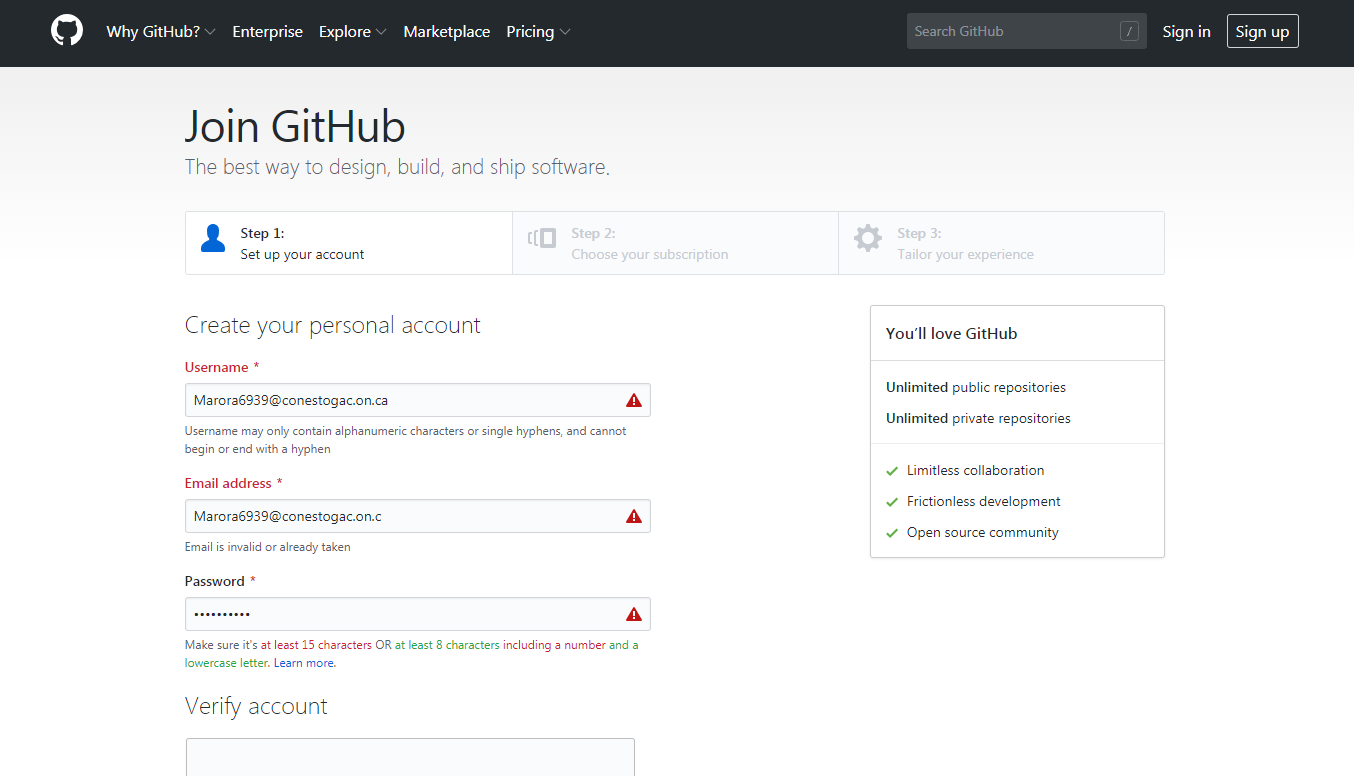
Nowadays, communication between devices is an inevitable part in technology development which paved the way for the discovery of different communication networks. Of these, mesh communication network is an important type made up of radio nodes arranged in a mesh topology. Mesh refers to the interconnection among nodes and devices.

In our project we are trying to implement a mesh networking which make the communication among the connected nodes in the network possible. Here, we can send data wirelessly from one node to another. This project have wide application in different areas include colleges, industries, offices etc and we are collaborating with github for getting more revised code for our project.

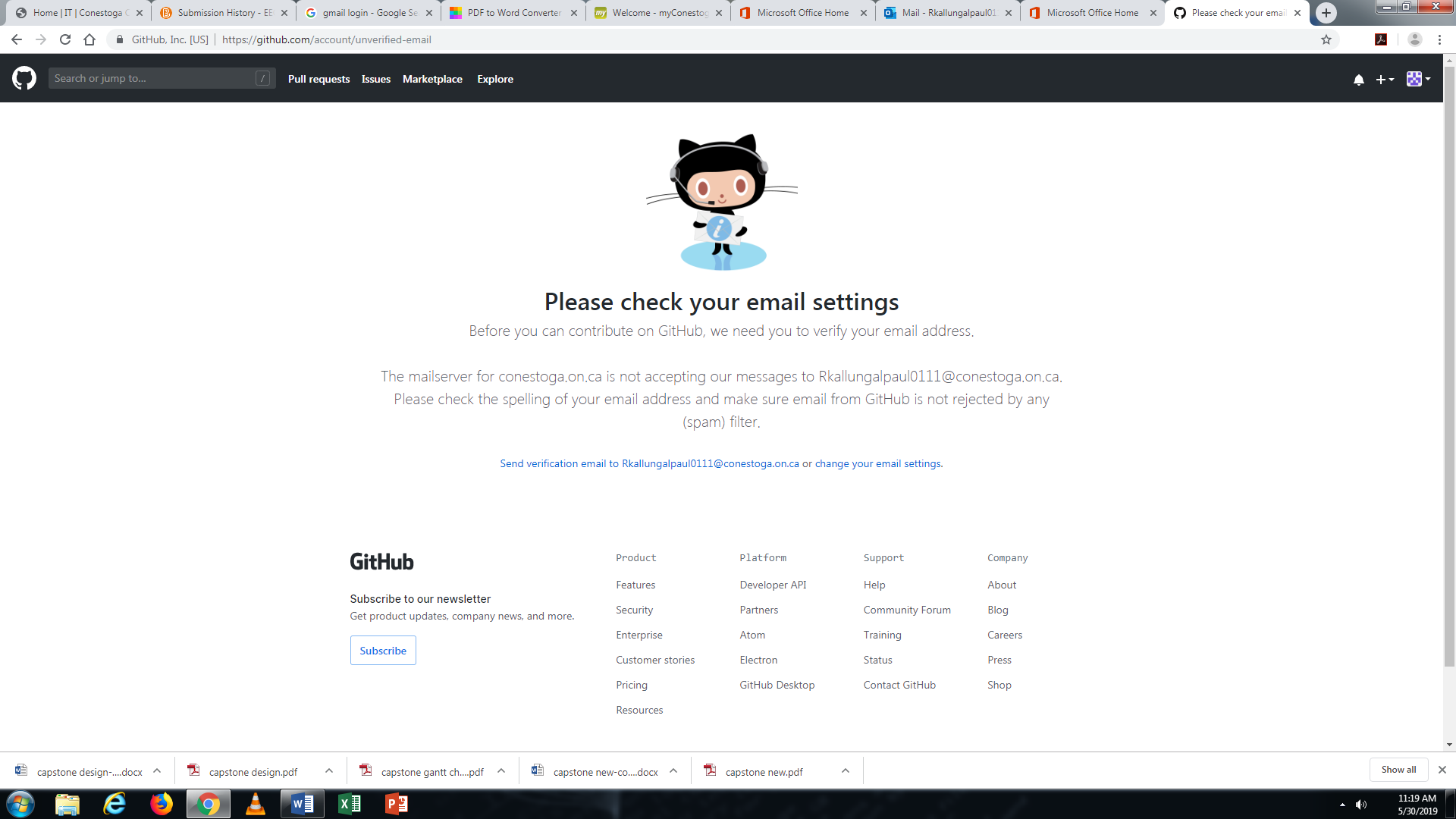
**Creating account at GitHub:**

We have to sign in with our basic profile information’s. As a part of doing the software part of project, we three members started signing into GitHub account through the following steps:

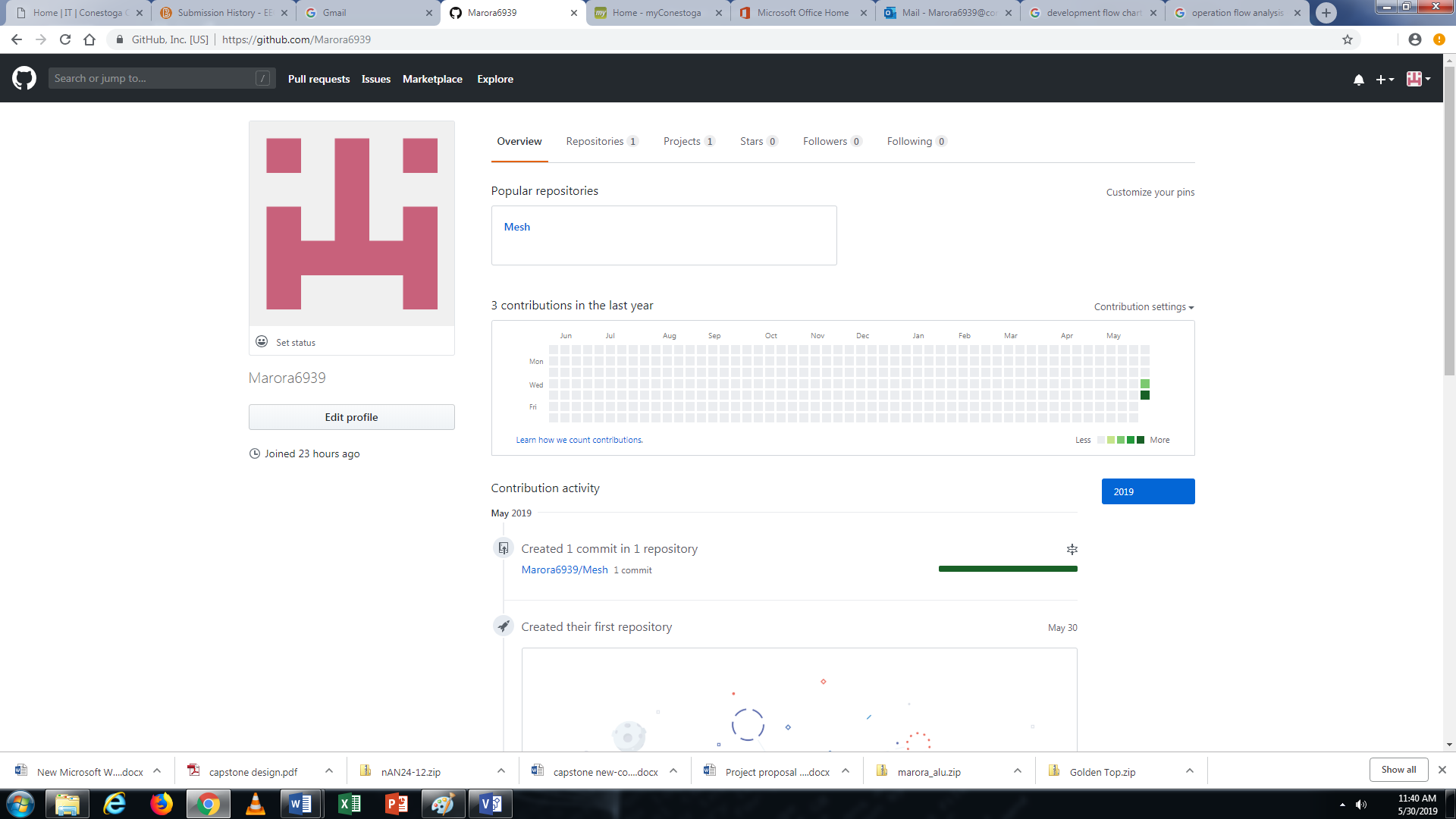
****

****

After creating the profile, we have to verify that with our mail id.



After that, this will be our GitHub account profiler looks like



## PROJECT OVERVIEW AND WORKING

In this project we are planning to make a network with three nodes in which each node can communicate with any of the other nodes in the network and at the same time they can work as both transmitters and receiver. We can further develop the project to make a larger network. The Main requirements of the project are, three NRF24L01 which works as nodes and a microcontroller to make a network, for this project we are planning to use arm controller. A single NRF24L01 can listen up to 6 other modules at the same time.

As a part of doing the software part of project, we three members started signing into GitHub account through the following steps:

**INITIAL SOFTWARE CREATION PLANNING**

****

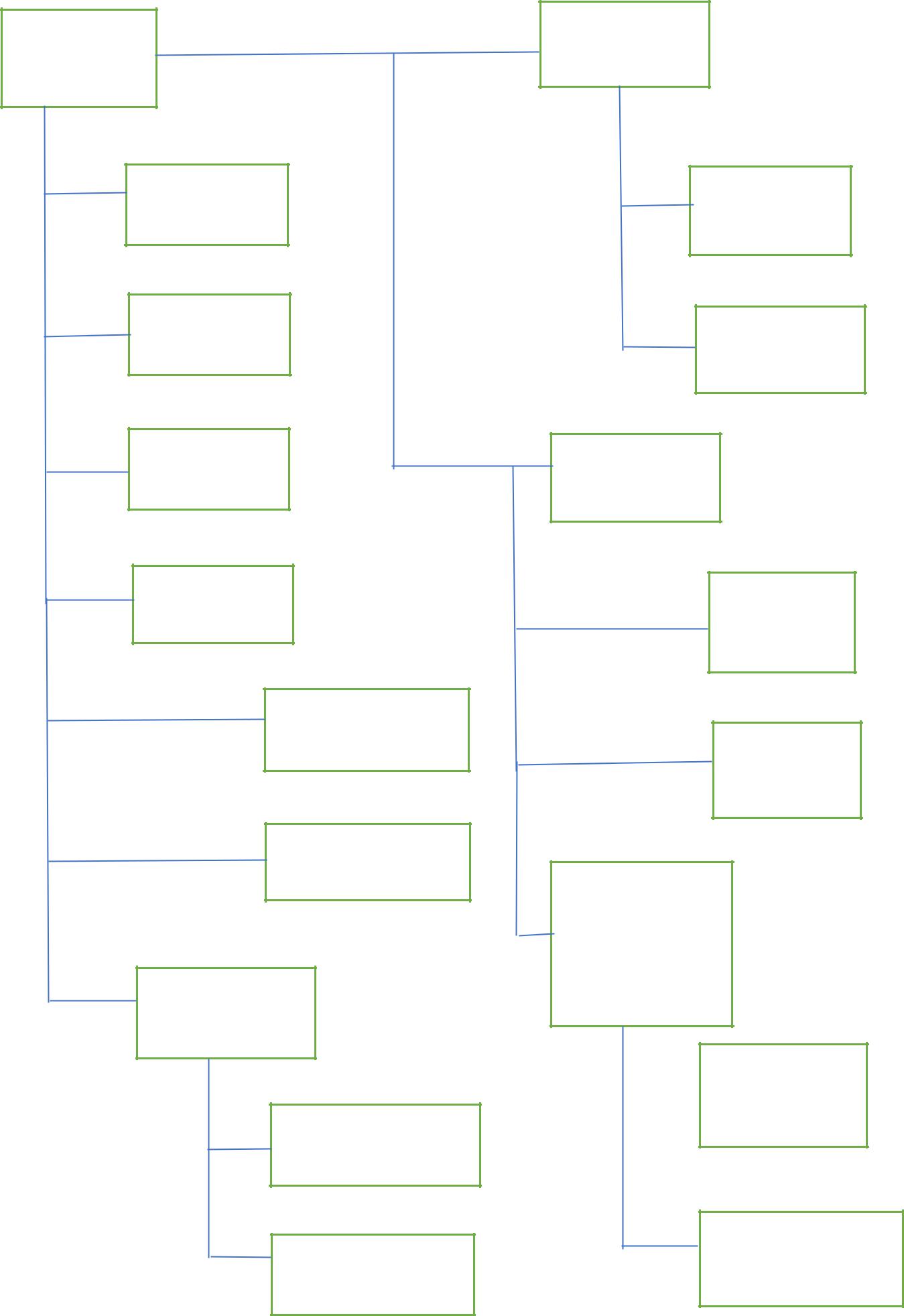
PTX – Transmitter

PRX -Receiver

**FLOW CHART**



**SOURCE CODE FLOWCHART**



Root ShockBurst(SB)

main.h/.c

Radio-sb.h/.c

system.h/.c

application-sb.h/.c

radio.h/.c

enhanced

Schockburst

nRF24L01

radio-esb.h/.c

target\_includes.h

application\_esb

.h/.c

mcu.c

Enhanced

Schockburst with

Bi-directional

data

nRF24L01

radio-pl.h/.c

Target-includes.h

Application-pl.h/.c

Target\_includes.h

**SOURCE CODE**

/\* Copyright (c) 2007 Nordic Semiconductor. All Rights Reserved.

\*

\* The information contained herein is property of Nordic Semiconductor ASA.

\* Terms and conditions of usage are described in detail in NORDIC

\* SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.

\*

\* Licensees are granted free, non-transferable use of the information. NO

\* WARRENTY of ANY KIND is provided. This heading must NOT be removed from

\* the file.

\*

\* $LastChangedRevision: 2477 $

\*/

/\*\*

\* @file

\* @ingroup Main

\* This file contain the main initialisation and allows users to make their

\* choices of operational mode. Implements a state machine through the enum

\* @c state, the @c state\_machine array, and the get\_next\_state() function.

\*

\* To choose between the differnt modes, after startup press:

\* - B1 for PTX mode

\* - B1 for ShockBurst in PTX mode. Indicate with @b LED1 on

\* - B2 for Enhanced ShockBurst in PTX mode. Indicate with @b LED2 on

\* - B3 for Enhanced ShockBurst with Bidirectional data in PTX mode. Indicate with @b LED1 and @b LED2 on

\* - B2 for PRX mode

\* - B1 for ShockBurst in PRX mode. Indicate with @b LED1 and @b LED3 on

\* - B2 for Enhanced ShockBurst in PRX mode. Indicate with @b LED2 and @b LED3 on

\* - B3 for Enhanced ShockBurst with Bidirectional data in PRX mode. Indicate with @b LED1, @b LED2 and @b LED3 on

\*

\* @author Per Kristian Schanke

\*/

#include <stdint.h>

#include <stdbool.h>

#include "hal\_nrf.h"

#include "target\_includes.h"

/\*\* Contain the common radio functions, implemented in radio.c \*/

#include "radio.h"

/\*\* Contain the specific radio functions for a radio in ShockBurst,

\* implemented in sb/radio\_sb.c \*/

#include "sb/radio\_sb.h"

/\*\* Contain the specific radio functions for a radio in Enhanced ShockBurst,

\* implemented in esb/radio\_esb.c \*/

#include "esb/radio\_esb.h"

/\*\* Contain the specific radio functions for a radio in Enhanced ShockBurst

\* with Bidirectional data, implemented in pl/radio\_pl.c \*/

#include "pl/radio\_pl.h"

/\*\* Contain the application functions for a radio in ShockBurst,

\* implemented in sb/application\_sb.c \*/

#include "sb/application\_sb.h"

/\*\* Contain the application functions for a radio in Enhanced ShockBurst,

\* implemented in esb/application\_esb.c \*/

#include "esb/application\_esb.h"

/\*\* Contain the application functions for a radio in Enhanced ShockBurst

\* with Bidirectional data, implemented in pl/application\_pl.c \*/

#include "pl/application\_pl.h"

/\*\* Contain the functions for delays, system functions and some timers,

\* implemented in system.c \*/

#include "system.h"

/\*\*

\* The possible states of the system.

\*/

typedef enum {

DEVICE\_IDLE = 0, /\*\*< The device is idle \*/

DEVICE\_PRX\_IDLE, /\*\*< The device will operate in @b PRX mode \*/

DEVICE\_PTX\_IDLE, /\*\*< The device will operate in @b PTX mode \*/

DEVICE\_PRX\_SB, /\*\*< The device will operate in @b PRX mode with ShockBurst functionailty \*/

DEVICE\_PRX\_ESB, /\*\*< The device will operate in @b PRX mode with Enhanced ShockBurst functionailty \*/

DEVICE\_PRX\_PL, /\*\*< The device will operate in @b PRX mode with Enhanced ShockBurst functionailty with Bidirectional data \*/

DEVICE\_PTX\_SB, /\*\*< The device will operate in @b PTX mode with ShockBurst functionailty \*/

DEVICE\_PTX\_ESB, /\*\*< The device will operate in @b PTX mode with Enhanced ShockBurst functionailty \*/

DEVICE\_PTX\_PL, /\*\*< The device will operate in @b PTX mode with Enhanced ShockBurst functionailty with Bidirectional data \*/

NO\_CHANGE /\*\*< No state change \*/

} state\_t;

/\*\*

\* The state transistion table. Indicates which state the statemachine

\* should jump to as next state.

\*

\* Example on use:@code

next\_state = state\_machine[current\_state][button\_pressed];

if (next\_state == NO\_CHANGE)

next\_state = current\_state;

@endcode

\*/

static const state\_t state\_machine[][3] =

// B1 B2 B3 CURRENT STATE

{ {DEVICE\_PTX\_IDLE, DEVICE\_PRX\_IDLE, NO\_CHANGE}, /\*\*< DEVICE\_IDLE \*/

{DEVICE\_PRX\_SB, DEVICE\_PRX\_ESB, DEVICE\_PRX\_PL}, /\*\*< DEVICE\_PRX\_IDLE \*/

{DEVICE\_PTX\_SB, DEVICE\_PTX\_ESB, DEVICE\_PTX\_PL}, /\*\*< DEVICE\_PTX\_IDLE \*/

{NO\_CHANGE, NO\_CHANGE, NO\_CHANGE}, /\*\*< DEVICE\_PRX\_SB \*/

{NO\_CHANGE, NO\_CHANGE, NO\_CHANGE}, /\*\*< DEVICE\_PRX\_ESB \*/

{NO\_CHANGE, NO\_CHANGE, NO\_CHANGE}, /\*\*< DEVICE\_PRX\_PL \*/

{NO\_CHANGE, NO\_CHANGE, NO\_CHANGE}, /\*\*< DEVICE\_PTX\_SB \*/

{NO\_CHANGE, NO\_CHANGE, NO\_CHANGE}, /\*\*< DEVICE\_PTX\_ESB \*/

{NO\_CHANGE, NO\_CHANGE, NO\_CHANGE} /\*\*< DEVICE\_PTX\_PL \*/

};

/\*\* LED should be on \*/

#define ON 1

/\*\* LED should be off \*/

#define OFF 0

/\*\* Function should loop for 0 seconds \*/

#define SEK\_0 0

/\*\* Function should loop for aprox 1 seconds \*/

#define SEK\_1 10

/\*\* Function should loop for aprox 2 seconds \*/

#define SEK\_2 20

/\*\* Function should loop for aprox 3 seconds \*/

#define SEK\_3 30

/\*\* Defines the leds that should be turned on by the show\_status() function

\* and how long the light should be on. Column 1 is LED1 ON/OFF, column 2

\* is LED2 ON/OFF, column 3 is LED3 ON/OFF,

\* column 4 indicates wheter all light should be turned off (OFF) or if the

\* pattern already lit up should stay on (ON), column 5 is the time the lights

\* should stay in a locking loop (rounds of 100ms).

\*/

static const uint8\_t show\_state[][5] =

//LED1, LED2, LED3, ALL off after?, Time,

{{ON , ON , ON , ON, SEK\_0}, /\*\*< DEVICE\_IDLE \*/

{OFF, OFF, ON , ON, SEK\_0}, /\*\*< DEVICE\_PRX\_IDLE \*/

{OFF, OFF, OFF, ON, SEK\_0}, /\*\*< DEVICE\_PTX\_IDLE \*/

{ON , OFF, ON , OFF, SEK\_3}, /\*\*< DEVICE\_PRX\_SB \*/

{OFF, ON , ON , OFF, SEK\_3}, /\*\*< DEVICE\_PRX\_ESB \*/

{ON , ON , ON , OFF, SEK\_3}, /\*\*< DEVICE\_PRX\_PL \*/

{ON , OFF, OFF, OFF, SEK\_3}, /\*\*< DEVICE\_PTX\_SB \*/

{OFF, ON , OFF, OFF, SEK\_3}, /\*\*< DEVICE\_PTX\_ESB \*/

{ON , ON , OFF, OFF, SEK\_3}, /\*\*< DEVICE\_PTX\_PL \*/

};

/\*\* The address of the radio. Parameter to the radio init \*/

static code const uint8\_t address[HAL\_NRF\_AW\_5BYTES] = {0x22,0x33,0x44,0x55,0x01};

/\*\* Implementation of the state transition. Changes state based on the

\* current state and the value of a pressed button. Waits til button is released

\* before it returns.

\*

\* @param current\_state The current state of the statemachine

\* @return The next state. Returns @b current\_state if state\_machine

\* indicated @c NO\_CHANGE

\*/

static state\_t get\_next\_state(state\_t current\_state);

/\*\* Function that runs in a loop until all buttons are released.

\*/

static void wait\_for\_button\_release(void);

/\*\* Shows the state the state\_machine is in.

\*/

static void show\_status(state\_t operation);

/\*\* Function that initialises everything. Calls @b system\_init () which is

\* hardware dependant, and @b device\_boot\_msg () from @b system.c.

\* It implementes a simple statemachine to handle the input from the user on

\* the evaluation board. With two clicks, the user can choose between

\* primary transmitter mode (PTX) and primary reciever mode (PRX), and between

\* the functionality levels ShockBurst (sb), Enchanced ShockBurst,

\* and Enhanced ShockBurst with Bidirectional data (pl).

\*/

void main(void)

{

state\_t current\_state = DEVICE\_IDLE;

system\_init(); //Hardware dependant system initialisation

device\_boot\_msg(); //Flashes LED's in a simple pattern

GLOBAL\_INT\_ENABLE(); //Ensure that all interupts are turned on

LED\_ALL\_OFF(); //Turn off all lights

wait\_for\_button\_release (); //Ensure that all buttons are released

//Implemenation of a simple state machine.

while (true)

{

current\_state = get\_next\_state (current\_state);// Go to next state

wait\_for\_button\_release (); // Ensure that all

// buttons are released

show\_status (current\_state);

switch (current\_state)

{

case DEVICE\_IDLE: // No operation chosen yet

break;

case DEVICE\_PRX\_IDLE: // In PRX mode, but still lack

break; // functionality

case DEVICE\_PTX\_IDLE: // In PTX mode, but still lack

break; // functionality

case DEVICE\_PRX\_SB: // Start as PRX in ShockBurst

radio\_sb\_init (address, HAL\_NRF\_PRX);

device\_prx\_mode\_sb ();

break;

case DEVICE\_PRX\_ESB: // Start as PRX in Enhanced

radio\_esb\_init (address, HAL\_NRF\_PRX);// ShockBurst

device\_prx\_mode\_esb ();

break;

case DEVICE\_PRX\_PL: //Start as PRX in Enhanced

radio\_pl\_init (address, HAL\_NRF\_PRX); //ShockBurst with ACK payload

device\_prx\_mode\_pl ();

break;

case DEVICE\_PTX\_SB: //Start as PTX in ShockBurst

radio\_sb\_init (address, HAL\_NRF\_PTX);

device\_ptx\_mode\_sb ();

break;

case DEVICE\_PTX\_ESB: //Start as PTX in Enhanced

radio\_esb\_init (address, HAL\_NRF\_PTX);//ShockBurst

device\_ptx\_mode\_esb ();

break;

case DEVICE\_PTX\_PL: // Start as PTX in Enhanced

radio\_pl\_init (address, HAL\_NRF\_PTX); // ShockBurst with ACK payload

device\_ptx\_mode\_pl ();

break;

default: // If in an illegal state, set to

current\_state = DEVICE\_IDLE; // default state (DEVICE\_IDLE)

break;

}

}

}

static state\_t get\_next\_state (state\_t current\_state)

{

state\_t next\_state = NO\_CHANGE;

if (B1\_PRESSED()) // Swap state according to state\_machine

{ // array with button input and

// current\_state as input

next\_state = state\_machine[current\_state][0];

}

else if (B2\_PRESSED())

{

next\_state = state\_machine[current\_state][1];

}

else if (B3\_PRESSED())

{

next\_state = state\_machine[current\_state][2];

}

if (next\_state == NO\_CHANGE) // If no statechange should occur, return

{ // previous state

next\_state = current\_state;

}

else // As it takes some time for the button to

{ // stabalise as pressed, give it a short

delay\_10ms(); // delay to stabalise

}

return next\_state;

}

static void wait\_for\_button\_release (void)

{

while (B1\_PRESSED() || B2\_PRESSED() || B3\_PRESSED()) // Wait until all

; // buttons are released

delay\_10ms(); // Delay to stabalise

}

static void show\_status (state\_t operation)

{

uint16\_t time;

LED\_ALL\_OFF();

if (show\_state[operation][0] == ON)

{

LED1\_ON();

}

if (show\_state[operation][1] == ON)

{

LED2\_ON();

}

if (show\_state[operation][2] == ON)

{

LED3\_ON();

}

// If there is to be a delay where LED's are shown, but no input is

// accepted, delay for the period indicated in show\_state[operation][4]

if (show\_state[operation][4] > 0)

{

time = (uint16\_t)(show\_state[operation][4] \* 100);

start\_timer(time);

wait\_for\_timer();

}

// If the radio goes into an operational mode, all LED's should be turned off

// before entering that mode

if (show\_state[operation][3] == OFF)

{

LED\_ALL\_OFF();

}