MESH NETWORK COMMUNICATION

Supervisor: Sandra French

TEAM MEMBERS

Swathi Thekkuveettil

Reshma Kallungal Paul

Manisha Arora

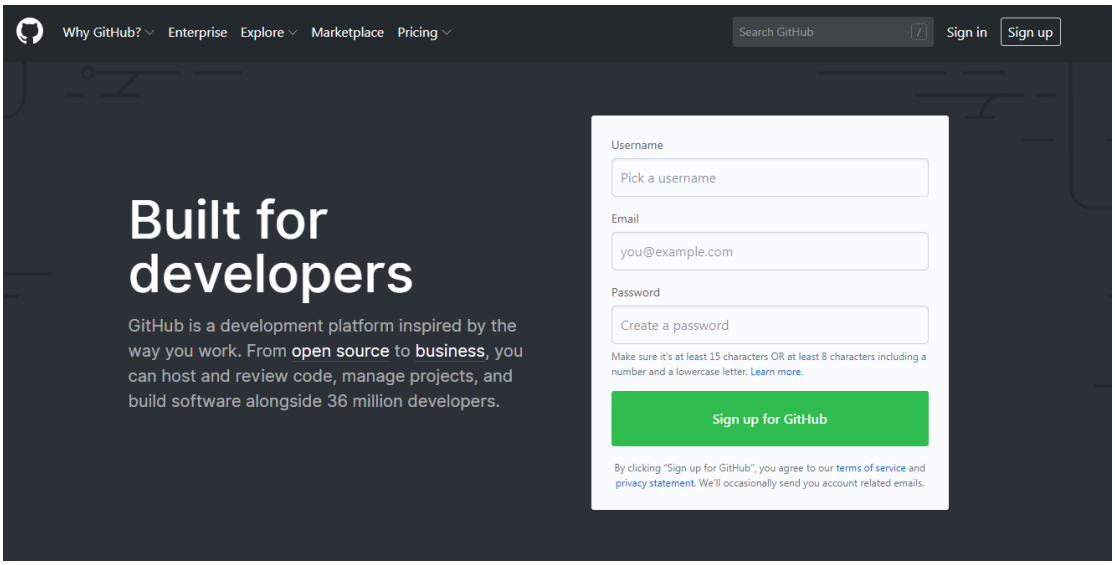
INTRODUCTION

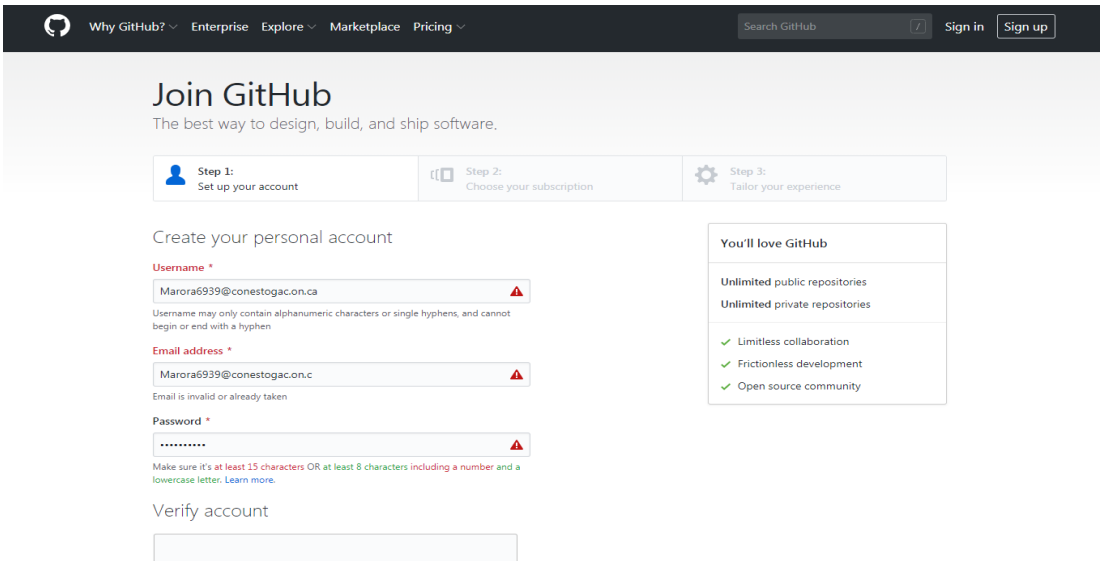
Nowadays, communication between devices is an inevitable part in technology development which paved the way for the discovery of different communication networks. Of these, mesh communication network is an important type made up of radio nodes arranged in a mesh topology. Mesh refers to the interconnection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

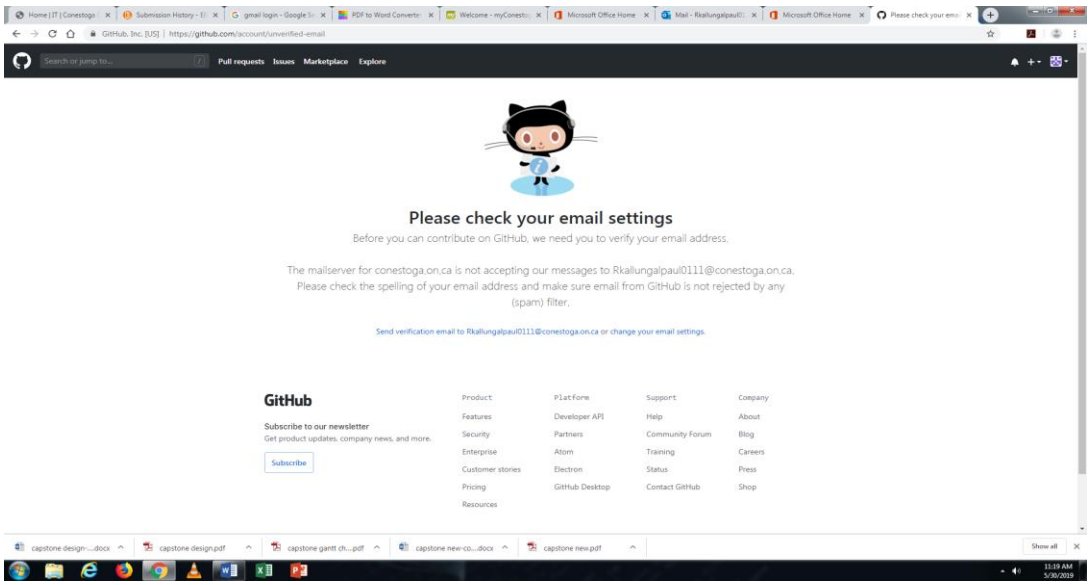
Creating account at GitHub:

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:

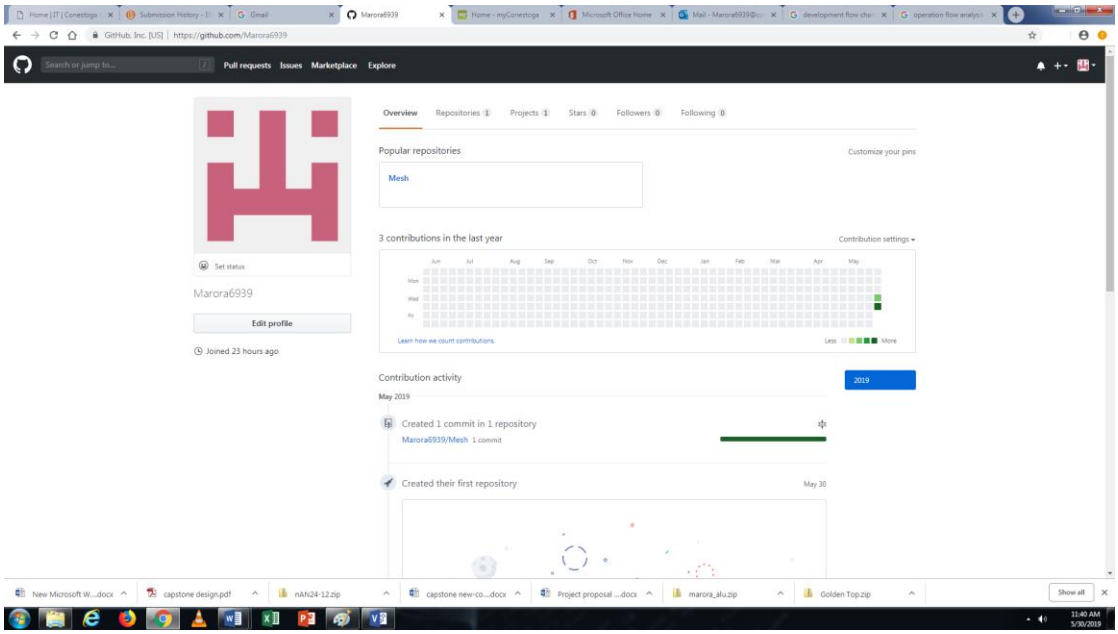




After creating the profile, we have to verify that with our mail id.



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

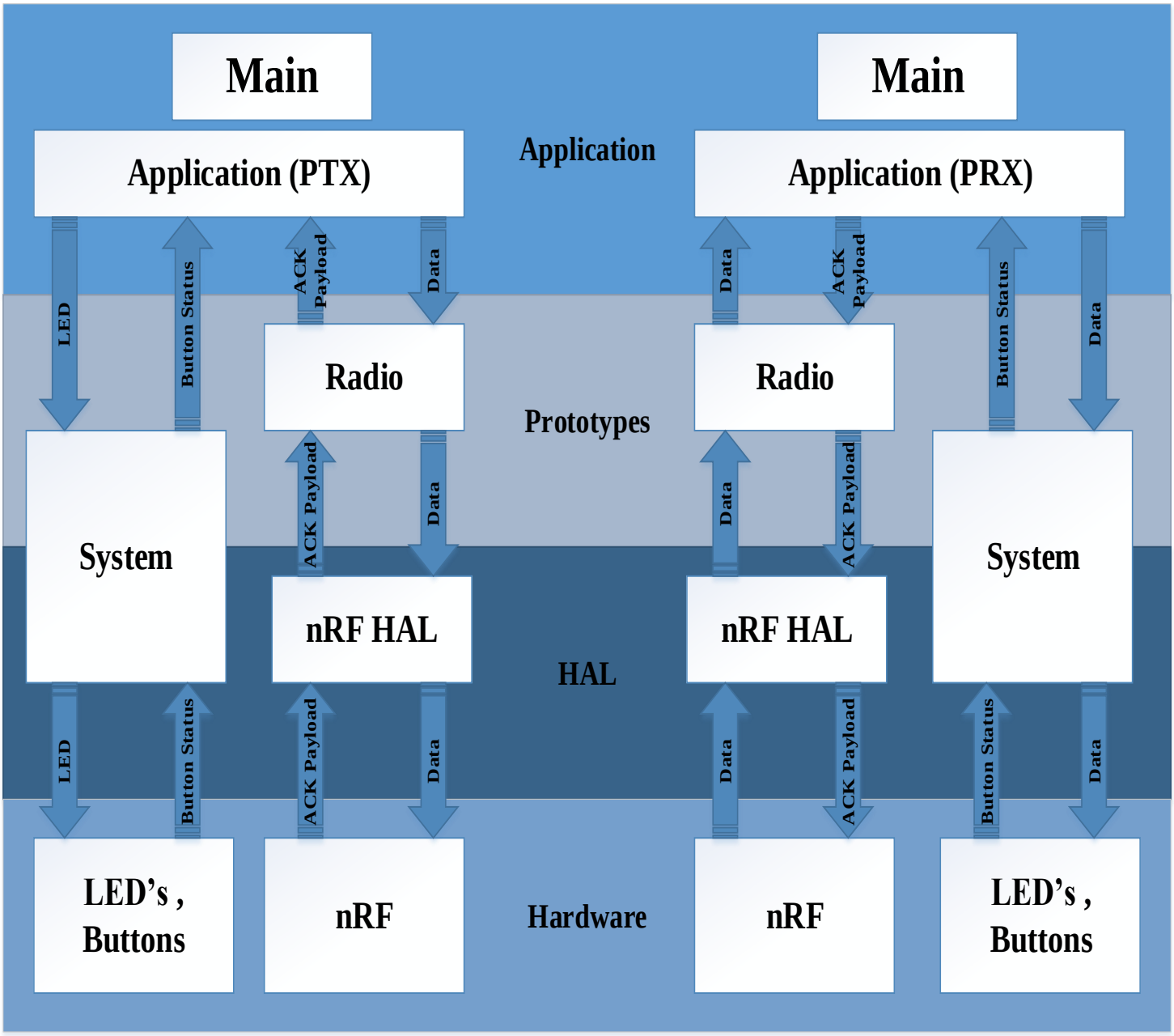


PROJECT OVERVIEW AND WORKING

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.

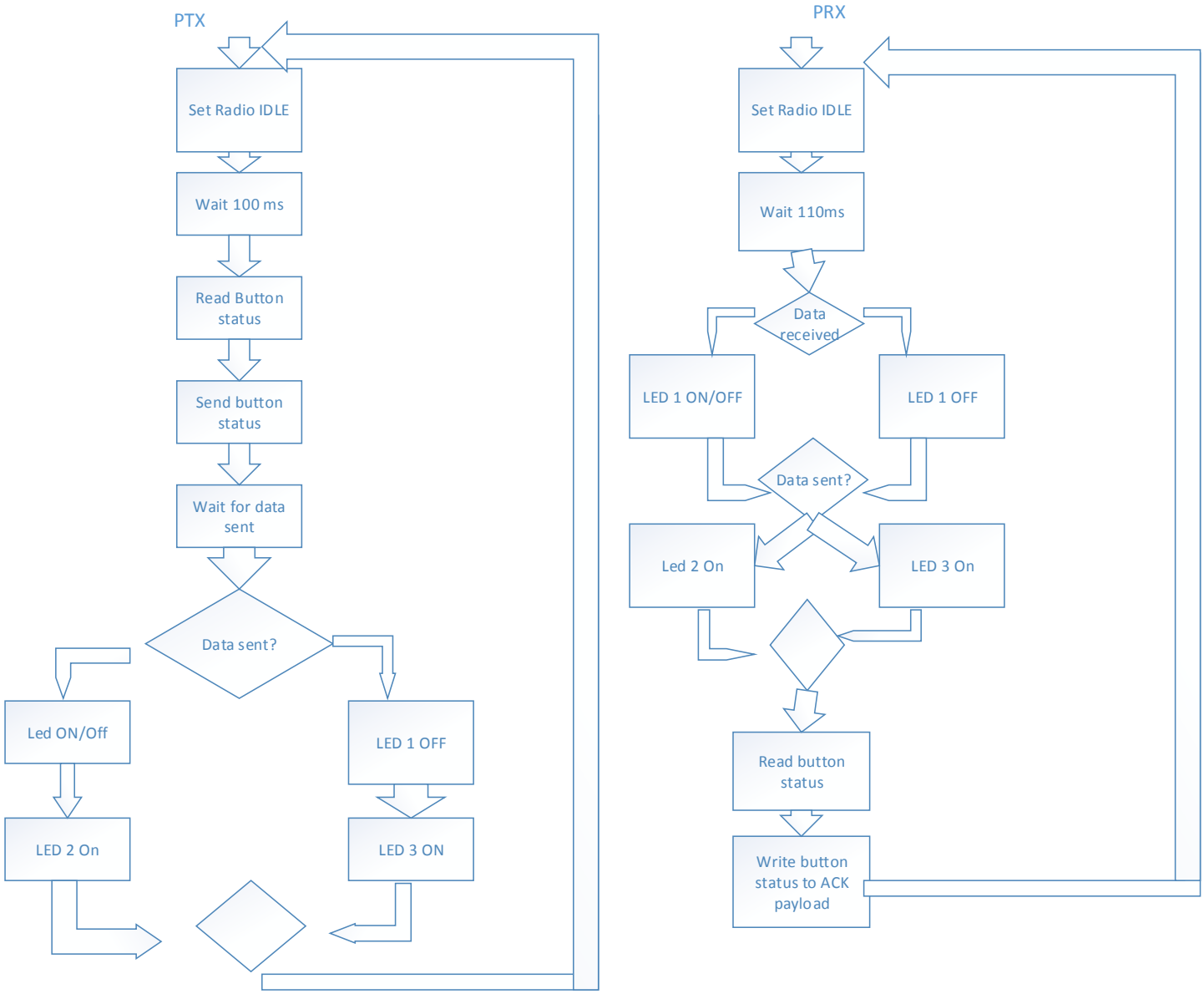
As a part of doing the software part of project, we three members started signing into GitHub account through the following steps:

INITIAL SOFTWARE CREATION PLANNING

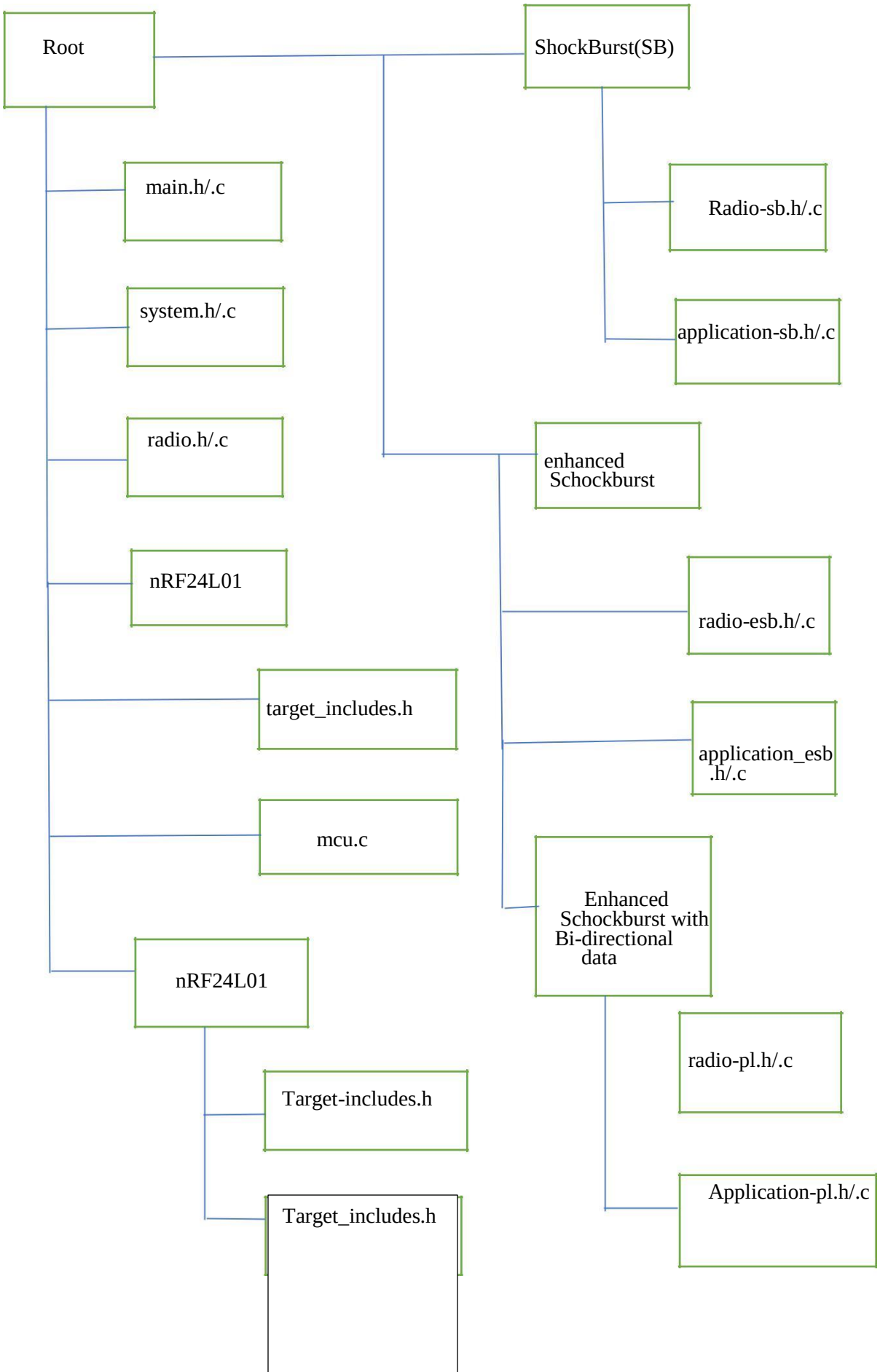


PTX – Transmitter
PRX -Receiver

FLOW CHART


PTX
Set Radio IDLE
Wait 100 ms
Read Button status
Send button status
Wait for data sent
Data sent?
Led ON/Off
LED 2 On
LED 1 OFF
LED 3 ON
PRX
Set Radio IDLE
Wait 110ms
Data received
LED 1 ON/OFF
LED 1 OFF
Data sent?
Led 2 On
LED 3 On
Read button status
Write button status to ACK payload

SOURCE CODE FLOWCHART



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();
}
```