#include <stdint.h>

#include <stdbool.h>

#include <stdio.h>

#include <ctype.h>

#include "common.h"

/\*\* 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

#define TIMER1\_OVERFLOW() (TF1 == 1) /\*\*< Checks for overflow on timer1 \*/

#define MAX\_RUNTIME 65 /\*\* The maximum runtime in ms that a timer might run. Calculated by

\* taking the maximum number your timer might take and divide that by

\* @b CYCLES\_PR\_MS. On the nRF24L01 evaluation kit this is 0xFFFF / 1000 =

\* 65 (aprox).

\*/

#define MAX\_TIME (0 - MAX\_RUNTIME \* CYCLES\_PR\_MS) /\*\* This constant defines the value the timer should be set to for running for

\* the maximum time.

\*/

/\*\* Start timer1. \*/

#define T1\_START() do{ TR1 = 1; \

ET1 = 0; \

TF1 = 0; \

EA = 1; \

}while(0)

/\*\* Stop timer1. \*/

#define T1\_STOP() do{ TR1 = 0; \

ET1 = 0; \

}while(0)

/\*\* Initialise Timer 1 in mode 1 (16-bit timer) \*/

#define T1\_MODE1() (TMOD |= 0x10)

/\*\* Sets the low bit of timer 1 \*/

#define T1\_SET\_LB(x) (TL1 = x)

/\*\* Sets the high bit of timer 1 \*/

#define T1\_SET\_HB(x) (TH1 = x)

#define RADIO\_ACTIVITY() (IE0 == 1) /\*\*< The register on the radio indicating

\* activity on the radio \*/

/\*\* How to reset the activity register \*/

#define RESET\_RADIO\_ACTIVITY() (IE0 = 0)

/\*\* Tx side System setup \*/

SPI\_HandleTypeDef hspi1;

UART\_HandleTypeDef huart2;

/\*\* Tx side System setup end \*/

/\* Private function prototypes for Radio -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_SPI1\_Init(void);

static void MX\_USART2\_UART\_Init(void);

/\* Private function prototypes for Radio ends -----------------------------------------------\*/

int pinState1=1,pinState2=1,pinState3=1;

static uint8\_t timer\_rounds;

static radio\_status\_t status;

static xdata uint8\_t pload\_sb[RF\_PAYLOAD\_LENGTH];

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;

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 \*/

};

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};

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);

/\*\* the following function is for hardware device setup.

\*/

void system\_init (void);

/\*\* Blinks the LED's to check the start device

\*/

void device\_boot\_msg(void);

/\*\* Blinks the LED's to check the start device

\*/

void LED\_ALL\_OFF(void);

/\*\* the following is for checking the buttons are pressed or not

\*/

int B1\_PRESSED(void);

/\*\* the following is for checking the buttons are pressed or not

\*/

int B2\_PRESSED(void);

/\*\* the following is for checking the buttons are pressed or not

\*/

int B3\_PRESSED(void);

/\*\* the following is for turning on the corresponding LED

\*/

void LED1\_ON(void);

/\*\* the following is for turning on the corresponding LED

\*/

void LED2\_ON(void);

/\*\* the following is for turning on the corresponding LED

\*/

void LED3\_ON(void);

/\*\* the following is for timing controll units

\*/

void start\_timer (uint16\_t time);

/\*\* the following is for timing controll units

\*/

void wait\_for\_timer (void);

/\*\* the following is for setting up the radio in SB mode

\*/

void radio\_sb\_init (const uint8\_t \*address, hal\_nrf\_operation\_mode\_t operational\_mode);

/\*\* the following is for getting the radio status

\*/

void radio\_set\_status (radio\_status\_t new\_status);

/\*\* the following is for receiving mode

\*/

void device\_prx\_mode\_sb(void);

/\*\* the following is for radio setup

\*/

void radio\_irq(void);

/\*\* the following is for getting the status of the radio

\*/

radio\_status\_t radio\_get\_status (void);

/\*\* the following is for getting timing value

\*/

bool timer\_done (void);

/\*\* the following is for getting timing value

\*/

uint8\_t radio\_get\_pload\_byte (uint8\_t byte\_index);

/\*\* the following is for initialising the radio in esb mode

\*/

void radio\_esb\_init (const uint8\_t \*address, hal\_nrf\_operation\_mode\_t operational\_mode);

/\*\* the following is for initialising the radio in pl mode

\*/

void radio\_pl\_init (const uint8\_t \*address, hal\_nrf\_operation\_mode\_t operational\_mode);

/\*\* the following is for transmitting in SB mode

\*/

void device\_ptx\_mode\_sb(void);

/\*\* this is to send a data in air

\*/

void radio\_send\_packet(uint8\_t \*packet, uint8\_t length);

/\*\* the following is for transmitting in ESB mode

\*/

void device\_ptx\_mode\_esb(void);

/\*\* the following is for receivingin ESB mode

\*/

void device\_prx\_mode\_esb(void);

/\*\* the following is for transmitting in PL mode

\*/

void device\_ptx\_mode\_pl(void);

/\*\* the following is for receivingin PL mode

\*/

void device\_prx\_mode\_pl(void);

void radio\_send\_packet(uint8\_t \*packet, uint8\_t length)

{

hal\_nrf\_write\_tx\_pload(packet, length); // load message into radio

//CE\_PULSE(); // send packet // i dont know where to find this section yet

radio\_set\_status (RF\_BUSY); // trans. in progress; RF\_BUSY

}

void radio\_pl\_init (const uint8\_t \*address, hal\_nrf\_operation\_mode\_t operational\_mode)

{

hal\_nrf\_close\_pipe(HAL\_NRF\_ALL); // First close all radio pipes

// Pipe 0 and 1 open by default

hal\_nrf\_open\_pipe(HAL\_NRF\_PIPE0, true); // Then open pipe0, w/autoack

hal\_nrf\_set\_crc\_mode(HAL\_NRF\_CRC\_16BIT); // Operates in 16bits CRC mode

hal\_nrf\_set\_auto\_retr(RF\_RETRANSMITS, RF\_RETRANS\_DELAY);

// Enables auto retransmit.

// 3 retrans with 250ms delay

hal\_nrf\_set\_address\_width(HAL\_NRF\_AW\_5BYTES); // 5 bytes address width

hal\_nrf\_set\_address(HAL\_NRF\_TX, address); // Set device's addresses

hal\_nrf\_set\_address(HAL\_NRF\_PIPE0, address); // Sets recieving address on

// pipe0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Changed from esb/radio\_esb.c \*

\* Enables: \*

\* - ACK payload \*

\* - Dynamic payload width \*

\* - Dynamic ACK \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

hal\_nrf\_enable\_ack\_pl(); // Try to enable ack payload

// When the features are locked, the FEATURE and DYNPD are read out 0x00

// even after we have tried to enable ack payload. This mean that we need to

// activate the features.

if(hal\_nrf\_read\_reg(FEATURE) == 0x00 && (hal\_nrf\_read\_reg(DYNPD) == 0x00))

{

hal\_nrf\_lock\_unlock (); // Activate features

hal\_nrf\_enable\_ack\_pl(); // Enables payload in ack

}

hal\_nrf\_enable\_dynamic\_pl(); // Enables dynamic payload

hal\_nrf\_setup\_dyn\_pl(ALL\_PIPES); // Sets up dynamic payload on

// all data pipes.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* End changes from esb/radio\_esb.c \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(operational\_mode == HAL\_NRF\_PTX) // Mode depentant settings

{

hal\_nrf\_set\_operation\_mode(HAL\_NRF\_PTX); // Enter TX mode

}

else

{

hal\_nrf\_set\_operation\_mode(HAL\_NRF\_PRX); // Enter RX mode

hal\_nrf\_set\_rx\_pload\_width((uint8\_t)HAL\_NRF\_PIPE0, RF\_PAYLOAD\_LENGTH);

// Pipe0 expect

// PAYLOAD\_LENGTH byte payload

// PAYLOAD\_LENGTH in radio.h

}

hal\_nrf\_set\_rf\_channel(RF\_CHANNEL); // Operating on static channel

// Defined in radio.h.

// Frequenzy =

// 2400 + RF\_CHANNEL

hal\_nrf\_set\_power\_mode(HAL\_NRF\_PWR\_UP); // Power up device

start\_timer(RF\_POWER\_UP\_DELAY); // Wait for the radio to

wait\_for\_timer(); // power up

radio\_set\_status (RF\_IDLE); // Radio now ready

}

void radio\_esb\_init (const uint8\_t \*address, hal\_nrf\_operation\_mode\_t operational\_mode)

{

hal\_nrf\_close\_pipe(HAL\_NRF\_ALL); // First close all radio pipes

// Pipe 0 and 1 open by default

hal\_nrf\_open\_pipe(HAL\_NRF\_PIPE0, true); // Then open pipe0, w/autoack

// Changed from sb/radio\_sb.c

hal\_nrf\_set\_crc\_mode(HAL\_NRF\_CRC\_16BIT); // Operates in 16bits CRC mode

hal\_nrf\_set\_auto\_retr(RF\_RETRANSMITS, RF\_RETRANS\_DELAY);

// Enables auto retransmit.

// 3 retrans with 250ms delay

// Changed from sb/radio\_sb.c

hal\_nrf\_set\_address\_width(HAL\_NRF\_AW\_5BYTES); // 5 bytes address width

hal\_nrf\_set\_address(HAL\_NRF\_TX, address); // Set device's addresses

hal\_nrf\_set\_address(HAL\_NRF\_PIPE0, address); // Sets recieving address on

// pipe0

if(operational\_mode == HAL\_NRF\_PTX) // Mode depentant settings

{

hal\_nrf\_set\_operation\_mode(HAL\_NRF\_PTX); // Enter TX mode

}

else

{

hal\_nrf\_set\_operation\_mode(HAL\_NRF\_PRX); // Enter RX mode

hal\_nrf\_set\_rx\_pload\_width((uint8\_t)HAL\_NRF\_PIPE0, RF\_PAYLOAD\_LENGTH);

// Pipe0 expect

// PAYLOAD\_LENGTH byte payload

// PAYLOAD\_LENGTH in radio.h

}

hal\_nrf\_set\_rf\_channel(RF\_CHANNEL); // Operating on static channel

// Defined in radio.h.

// Frequenzy =

// 2400 + RF\_CHANNEL

hal\_nrf\_set\_power\_mode(HAL\_NRF\_PWR\_UP); // Power up device

start\_timer(RF\_POWER\_UP\_DELAY); // Wait for the radio to

wait\_for\_timer(); // power up

radio\_set\_status (RF\_IDLE); // Radio now ready

}

void radio\_sb\_init (const uint8\_t \*address, hal\_nrf\_operation\_mode\_t operational\_mode)

{

hal\_nrf\_close\_pipe(HAL\_NRF\_ALL); // First close all radio pipes

// Pipe 0 and 1 open by default

hal\_nrf\_open\_pipe(HAL\_NRF\_PIPE0, false); // Open pipe0, without/autoack

hal\_nrf\_set\_crc\_mode(HAL\_NRF\_CRC\_16BIT); // Operates in 16bits CRC mode

hal\_nrf\_set\_auto\_retr(0, RF\_RETRANS\_DELAY); // Disables auto retransmit

hal\_nrf\_set\_address\_width(HAL\_NRF\_AW\_5BYTES); // 5 bytes address width

hal\_nrf\_set\_address(HAL\_NRF\_TX, address); // Set device's addresses

hal\_nrf\_set\_address(HAL\_NRF\_PIPE0, address); // Sets recieving address on

// pipe0

if(operational\_mode == HAL\_NRF\_PTX) // Mode depentant settings

{

hal\_nrf\_set\_operation\_mode(HAL\_NRF\_PTX); // Enter TX mode

}

else

{

hal\_nrf\_set\_operation\_mode(HAL\_NRF\_PRX); // Enter RX mode

hal\_nrf\_set\_rx\_pload\_width((uint8\_t)HAL\_NRF\_PIPE0, RF\_PAYLOAD\_LENGTH);

// Pipe0 expect

// PAYLOAD\_LENGTH byte payload

// PAYLOAD\_LENGTH in radio.h

}

hal\_nrf\_set\_rf\_channel(RF\_CHANNEL); // Operating on static channel

// Defined in radio.h.

// Frequenzy =

// 2400 + RF\_CHANNEL

hal\_nrf\_set\_power\_mode(HAL\_NRF\_PWR\_UP); // Power up device

//hal\_nrf\_set\_datarate(HAL\_NRF\_1MBPS); // Uncomment this line for

// compatibility with nRF2401

// and nRF24E1

start\_timer(RF\_POWER\_UP\_DELAY); // Wait for the radio to

wait\_for\_timer(); // power up

radio\_set\_status (RF\_IDLE); // Radio now ready

}

void device\_prx\_mode\_pl(void)

{

while(true)

{

// Setup and put the ACK payload on the FIFO

pload\_pl[0] = 0;

if(B1\_PRESSED())

{

pload\_pl[0] = 1;

}

hal\_nrf\_write\_ack\_pload (0, pload\_pl, RF\_PAYLOAD\_LENGTH);

start\_timer(110);

// Run until either 110ms has lapsed

// OR there is data on the radio

do

{

radio\_irq ();

} while ((radio\_get\_status () == RF\_IDLE) && !timer\_done());

// Blink LED2 if ACK payload was sent, LED3 if not

if ((radio\_get\_status ()) == RF\_TX\_DS

|| (radio\_get\_status ()) == RF\_TX\_AP)

{

LED2\_BLINK();

}

else

{

LED3\_BLINK();

}

if ((radio\_get\_status ()) == RF\_RX\_DR

|| (radio\_get\_status ()) == RF\_TX\_AP)

{

// Get the payload from the PTX and set LED1 accordingly

if (radio\_get\_pload\_byte (0) == 1)

{

LED1\_ON();

}

else

{

LED1\_OFF();

}

}

else

{

LED1\_OFF();

}

// Set radio status to idle

radio\_set\_status (RF\_IDLE);

}

EX0 = 1;

}

void device\_ptx\_mode\_pl(void)

{

while(true)

{

// Wait til the packet is sent

do {

radio\_irq ();

} while((radio\_get\_status ()) == RF\_BUSY);

// Blink LED2 if ACK is recieved, LED3 if not

if (((radio\_get\_status ()) == RF\_TX\_DS)

|| ((radio\_get\_status ()) == RF\_TX\_AP))

{

LED2\_BLINK();

}

else

{

LED3\_BLINK();

}

// If ACK payload was recieved, get the payload

if (radio\_get\_status () == RF\_TX\_AP)

{

// Get the payload from the PRX and set LED1 accordingly

if (radio\_get\_pload\_byte (0) == 1)

{

LED1\_ON();

}

else

{

LED1\_OFF();

}

}

// Sleep 100ms

HAL\_Delay(100);

// Set up the payload according to the input button 1

pload\_pl[0] = 0;

if(B1\_PRESSED())

{

pload\_pl[0] = 1;

}

//Send the packet

radio\_send\_packet(pload\_pl, RF\_PAYLOAD\_LENGTH);

}

}

void device\_prx\_mode\_esb(void)

{

while(true)

{

start\_timer(110);

// Run until either 110ms has lapsed

// OR there is data on the radio

do

{

radio\_irq ();

} while ((radio\_get\_status () == RF\_IDLE) && !timer\_done());

if ((radio\_get\_status ()) == RF\_RX\_DR)

{

// Get the payload from the PTX and set LED1

if (radio\_get\_pload\_byte (0) == 1)

{

LED1\_ON();

}

else

{

LED1\_OFF();

}

}

else

{

LED1\_OFF();

}

// Set radio status to idle

radio\_set\_status (RF\_IDLE);

}

}

void device\_ptx\_mode\_esb(void)

{

while(true)

{

// Wait til the packet is sent

do {

radio\_irq ();

} while((radio\_get\_status ()) == RF\_BUSY);

// Blink LED2 if ACK is recieved, LED3 if not

if (((radio\_get\_status ()) == RF\_TX\_DS))

{

LED2\_BLINK();

}

else

{

LED3\_BLINK();

}

// Sleep 100ms

start\_timer(100);

wait\_for\_timer();

// Set up the payload according to the input button 1

pload\_esb[0] = 0;

if(B1\_PRESSED())

{

pload\_esb[0] = 1;

}

//Send the packet

radio\_send\_packet(pload\_esb, RF\_PAYLOAD\_LENGTH);

}

}

void device\_ptx\_mode\_sb(void)

{

while(true)

{

// Wait til the packet is sent

do {

radio\_irq ();

} while((radio\_get\_status ()) == RF\_BUSY);

// Sleep 100ms

start\_timer(100);

wait\_for\_timer();

// Set up the payload according to the input button 1

pload\_sb[0] = 0;

if(B1\_PRESSED())

{

pload\_sb[0] = 1;

}

//Send the packet

radio\_send\_packet(pload\_sb, RF\_PAYLOAD\_LENGTH);

}

}

void device\_prx\_mode\_sb(void)

{

while(true)

{

start\_timer(110);

// Run until either 110ms has lapsed

// OR there is data on the radio

do

{

radio\_irq ();

} while ((radio\_get\_status() == RF\_IDLE) && !timer\_done());

if ((radio\_get\_status ()) == RF\_RX\_DR)

{

// Get the payload from the PTX and set LED1 accordingly

if (radio\_get\_pload\_byte(0) == 1)

{

LED1\_ON();

}

else

{

LED1\_OFF();

}

}

else

{

LED1\_OFF();

}

// Set radio status to idle

radio\_set\_status(RF\_IDLE);

}

}

uint8\_t radio\_get\_pload\_byte (uint8\_t byte\_index)

{

return pload[byte\_index];

}

bool timer\_done (void)

{

bool retval = false;

if (TIMER1\_OVERFLOW())

{

timer\_rounds--;

if (timer\_rounds > 0)

{

run\_timer (MAX\_TIME);

}

else

{

retval = true;

T1\_STOP();

}

}

return retval;

}

radio\_status\_t radio\_get\_status (void)

{

return status;

}

void radio\_irq(void)

{

if (RADIO\_ACTIVITY()) // Check if an interupt is

{ // triggered

switch(hal\_nrf\_get\_clear\_irq\_flags ())

{

case (1<<HAL\_NRF\_MAX\_RT): // Max retries reached

hal\_nrf\_flush\_tx(); // flush tx fifo, avoid fifo jam

radio\_set\_status (RF\_MAX\_RT);

break;

case (1<<HAL\_NRF\_TX\_DS): // Packet sent

radio\_set\_status (RF\_TX\_DS);

break;

case (1<<HAL\_NRF\_RX\_DR): // Packet received

while (!hal\_nrf\_rx\_fifo\_empty ())

{

hal\_nrf\_read\_rx\_pload(pload);

}

radio\_set\_status (RF\_RX\_DR);

break;

case ((1<<HAL\_NRF\_RX\_DR)|(1<<HAL\_NRF\_TX\_DS)): // Ack payload recieved

while (!hal\_nrf\_rx\_fifo\_empty ())

{

hal\_nrf\_read\_rx\_pload(pload);

}

radio\_set\_status (RF\_TX\_AP);

break;

default:

break;

}

RESET\_RADIO\_ACTIVITY();

}

}

void radio\_set\_status (radio\_status\_t new\_status)

{

status = new\_status;

}

static void run\_timer (uint16\_t time)

{

if (time != 0)

{

T1\_MODE1(); // Setting up mode 1 on timer 1 (16-bit timer)

T1\_SET\_LB((uint8\_t)time);

T1\_SET\_HB((uint8\_t)(time >> 8));

T1\_START();

}

}

void wait\_for\_timer (void)

{

while (timer\_rounds > 0)

{

while (!TIMER1\_OVERFLOW())

;

timer\_rounds--;

if (timer\_rounds > 0)

{

run\_timer (MAX\_TIME);

}

}

T1\_STOP();

}

void start\_timer (uint16\_t time)

{

uint16\_t setuptime;

uint16\_t firstruntime;

firstruntime = (uint16\_t)(time % MAX\_RUNTIME);

setuptime = 0 - (firstruntime \* CYCLES\_PR\_MS);

time -= firstruntime;

timer\_rounds = (uint8\_t)(time / MAX\_RUNTIME) + 1;

if (setuptime == 0)

{

setuptime = MAX\_TIME;

timer\_rounds--;

}

run\_timer (setuptime);

}

void LED1\_ON(void)

{

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,1);

}

void LED2\_ON(void)

{

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,1);

}

void LED3\_ON(void)

{

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,1);

}

int B1\_PRESSED(void)

{

pinState1 = HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_3);

if(pinState1==0)

{

return 1;

}

else

{

return 0;

}

}

int B2\_PRESSED(void)

{

pinState2 = HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_4);

if(pinState2==0)

{

return 1;

}

else

{

return 0;

}

}

int B3\_PRESSED(void)

{

pinState3 = HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_5);

if(pinState3==0)

{

return 1;

}

else

{

return 0;

}

}

void LED\_ALL\_OFF(void)

{

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,0);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,0);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,0);

}

void device\_boot\_msg(void)

{

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,1);

HAL\_Delay(500);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,0);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,1);

HAL\_Delay(500);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,0);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,1);

HAL\_Delay(500);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,0);

HAL\_Delay(500);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,1);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,1);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,1);

HAL\_Delay(500);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,0);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,0);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,0);

}

/\*\* the following function is for system clock configuration, if the mcu is resetting we need to remove this

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct;

RCC\_ClkInitTypeDef RCC\_ClkInitStruct;

/\*\*Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE2);

/\*\*Initializes the CPU, AHB and APB busses clocks

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = 16;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLM = 16;

RCC\_OscInitStruct.PLL.PLLN = 336;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV4;

RCC\_OscInitStruct.PLL.PLLQ = 7;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

\_Error\_Handler(\_\_FILE\_\_, \_\_LINE\_\_);

}

/\*\*Initializes the CPU, AHB and APB busses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

\_Error\_Handler(\_\_FILE\_\_, \_\_LINE\_\_);

}

/\*\*Configure the Systick interrupt time

\*/

HAL\_SYSTICK\_Config(HAL\_RCC\_GetHCLKFreq()/1000);

/\*\*Configure the Systick

\*/

HAL\_SYSTICK\_CLKSourceConfig(SYSTICK\_CLKSOURCE\_HCLK);

/\* SysTick\_IRQn interrupt configuration \*/

HAL\_NVIC\_SetPriority(SysTick\_IRQn, 0, 0);

}

/\*\* this section here is for setting the three buttons input and 3 LED's\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct;

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOB, CSNpin\_Pin|CEpin\_Pin, GPIO\_PIN\_RESET);

/\*Configure GPIO pins : CSNpin\_Pin CEpin\_Pin \*/

GPIO\_InitStruct.Pin = CSNpin\_Pin|CEpin\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

GPIO\_InitTypeDef GPIO\_InitStruct;

GPIO\_InitStruct.Pin = (GPIO\_PIN\_0| GPIO\_PIN\_1 |GPIO\_PIN\_2);

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;

GPIO\_InitStruct.Alternate = 0;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

GPIO\_InitTypeDef GPIO\_InitStruct;

GPIO\_InitStruct.Pin = (GPIO\_PIN\_3| GPIO\_PIN\_4 |GPIO\_PIN\_5);

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;

GPIO\_InitStruct.Alternate = 0;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

printf("Switches and LED's initialised successfully\n");

}

/\*\* this section here is for setting Controller in SPI mode\*/

static void MX\_SPI1\_Init(void)

{

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_LOW;

hspi1.Init.CLKPhase = SPI\_PHASE\_1EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_32;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 10;

if (HAL\_SPI\_Init(&hspi1) != HAL\_OK)

{

\_Error\_Handler(\_\_FILE\_\_, \_\_LINE\_\_);

}

printf("SPI initialised successfully\n");

}

/\*\* this section here is for setting the USART which can be used or not\*/

static void MX\_USART2\_UART\_Init(void)

{

huart2.Instance = USART2;

huart2.Init.BaudRate = 115200;

huart2.Init.WordLength = UART\_WORDLENGTH\_8B;

huart2.Init.StopBits = UART\_STOPBITS\_1;

huart2.Init.Parity = UART\_PARITY\_NONE;

huart2.Init.Mode = UART\_MODE\_TX\_RX;

huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;

if (HAL\_UART\_Init(&huart2) != HAL\_OK)

{

\_Error\_Handler(\_\_FILE\_\_, \_\_LINE\_\_);

}

printf("USART initialised successfully\n");

}

void system\_init (void)

{

SystemClock\_Config();

MX\_GPIO\_Init();

MX\_SPI1\_Init();

MX\_USART2\_UART\_Init();

printf("Device initialised successfully\n");

}

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();

}

}

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 // this shoul be added in the last if needed

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;

}

}

}