/\*

Library: NRF24L01/NRF24L01+

Written by: Mohamed Yaqoob (MYaqoobEmbedded YouTube Channel)

Date Written: 10/11/2018

Last modified: -/-

Description: This is an STM32 device driver library for the NRF24L01 Nordic Radio transceiver, using STM HAL libraries

References: This library was written based on the Arduino NRF24 Open-Source library by J. Coliz and the NRF24 datasheet

- https://github.com/maniacbug/RF24

- https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss\_Preliminary\_Product\_Specification\_v1\_0.pdf

\* Copyright (C) 2018 - M. Yaqoob

This is a free software under the GNU license, you can redistribute it and/or modify it under the terms

of the GNU General Public Licenseversion 3 as published by the Free Software Foundation.

This software library is shared with puplic for educational purposes, without WARRANTY and Author is not liable for any damages caused directly

or indirectly by this software, read more about this on the GNU General Public License.

\*/

#include "stm32f3xx\_hal.h"

#include "stm32f3xx\_hal\_gpio.h"

#include "stm32f3xx\_hal\_spi.h"

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#include <stdio.h>

#include <stdint.h>

#include <ctype.h>

#include "common.h"

#define \_BV(x) (1<<(x))

/\* Memory Map \*/

#define REG\_CONFIG 0x00

#define REG\_EN\_AA 0x01

#define REG\_EN\_RXADDR 0x02

#define REG\_SETUP\_AW 0x03

#define REG\_SETUP\_RETR 0x04

#define REG\_RF\_CH 0x05

#define REG\_RF\_SETUP 0x06

#define REG\_STATUS 0x07

#define REG\_OBSERVE\_TX 0x08

#define REG\_CD 0x09

#define REG\_RX\_ADDR\_P0 0x0A

#define REG\_RX\_ADDR\_P1 0x0B

#define REG\_RX\_ADDR\_P2 0x0C

#define REG\_RX\_ADDR\_P3 0x0D

#define REG\_RX\_ADDR\_P4 0x0E

#define REG\_RX\_ADDR\_P5 0x0F

#define REG\_TX\_ADDR 0x10

#define REG\_RX\_PW\_P0 0x11

#define REG\_RX\_PW\_P1 0x12

#define REG\_RX\_PW\_P2 0x13

#define REG\_RX\_PW\_P3 0x14

#define REG\_RX\_PW\_P4 0x15

#define REG\_RX\_PW\_P5 0x16

#define REG\_FIFO\_STATUS 0x17

#define REG\_DYNPD 0x1C

#define REG\_FEATURE 0x1D

/\* Bit Mnemonics \*/

#define MASK\_RX\_DR 6

#define MASK\_TX\_DS 5

#define MASK\_MAX\_RT 4

#define BIT\_EN\_CRC 3

#define BIT\_CRCO 2

#define BIT\_PWR\_UP 1

#define BIT\_PRIM\_RX 0

#define BIT\_ENAA\_P5 5

#define BIT\_ENAA\_P4 4

#define BIT\_ENAA\_P3 3

#define BIT\_ENAA\_P2 2

#define BIT\_ENAA\_P1 1

#define BIT\_ENAA\_P0 0

#define BIT\_ERX\_P5 5

#define BIT\_ERX\_P4 4

#define BIT\_ERX\_P3 3

#define BIT\_ERX\_P2 2

#define BIT\_ERX\_P1 1

#define BIT\_ERX\_P0 0

#define BIT\_AW 0

#define BIT\_ARD 4

#define BIT\_ARC 0

#define BIT\_PLL\_LOCK 4

#define BIT\_RF\_DR 3

#define BIT\_RF\_PWR 6

#define BIT\_RX\_DR 6

#define BIT\_TX\_DS 5

#define BIT\_MAX\_RT 4

#define BIT\_RX\_P\_NO 1

#define BIT\_TX\_FULL 0

#define BIT\_PLOS\_CNT 4

#define BIT\_ARC\_CNT 0

#define BIT\_TX\_REUSE 6

#define BIT\_FIFO\_FULL 5

#define BIT\_TX\_EMPTY 4

#define BIT\_RX\_FULL 1

#define BIT\_RX\_EMPTY 0

#define BIT\_DPL\_P5 5

#define BIT\_DPL\_P4 4

#define BIT\_DPL\_P3 3

#define BIT\_DPL\_P2 2

#define BIT\_DPL\_P1 1

#define BIT\_DPL\_P0 0

#define BIT\_EN\_DPL 2

#define BIT\_EN\_ACK\_PAY 1

#define BIT\_EN\_DYN\_ACK 0

/\* Instruction Mnemonics \*/

#define CMD\_R\_REGISTER 0x00

#define CMD\_W\_REGISTER 0x20

#define CMD\_REGISTER\_MASK 0x1F

#define CMD\_ACTIVATE 0x50

#define CMD\_R\_RX\_PL\_WID 0x60

#define CMD\_R\_RX\_PAYLOAD 0x61

#define CMD\_W\_TX\_PAYLOAD 0xA0

#define CMD\_W\_ACK\_PAYLOAD 0xA8

#define CMD\_FLUSH\_TX 0xE1

#define CMD\_FLUSH\_RX 0xE2

#define CMD\_REUSE\_TX\_PL 0xE3

#define CMD\_NOP 0xFF

/\* Non-P omissions \*/

#define LNA\_HCURR 0

/\* P model memory Map \*/

#define REG\_RPD 0x09

/\* P model bit Mnemonics \*/

#define RF\_DR\_LOW 5

#define RF\_DR\_HIGH 3

#define RF\_PWR\_LOW 1

#define RF\_PWR\_HIGH 2

#define CSNpin\_Pin GPIO\_PIN\_0

#define CEpin\_Pin GPIO\_PIN\_1

SPI\_HandleTypeDef hspi2;

UART\_HandleTypeDef huart1;

typedef enum {

RF24\_PA\_m18dB = 0,

RF24\_PA\_m12dB,

RF24\_PA\_m6dB,

RF24\_PA\_0dB,

RF24\_PA\_ERROR

}rf24\_pa\_dbm\_e ;

//2. NRF24\_setDataRate() input

typedef enum {

RF24\_1MBPS = 0,

RF24\_2MBPS,

RF24\_250KBPS

}rf24\_datarate\_e;

//3. NRF24\_setCRCLength() input

typedef enum {

RF24\_CRC\_DISABLED = 0,

RF24\_CRC\_8,

RF24\_CRC\_16

}rf24\_crclength\_e;

void NRF24\_DelayMicroSeconds(uint32\_t uSec);

//1. Chip Select function

void NRF24\_csn(int mode);

//2. Chip Enable

void NRF24\_ce(int level);

//3. Read single byte from a register

uint8\_t NRF24\_read\_register(uint8\_t reg);

//4. Read multiple bytes register

void NRF24\_read\_registerN(uint8\_t reg, uint8\_t \*buf, uint8\_t len);

//5. Write single byte register

void NRF24\_write\_register(uint8\_t reg, uint8\_t value);

//6. Write multipl bytes register

void NRF24\_write\_registerN(uint8\_t reg, const uint8\_t\* buf, uint8\_t len);

//7. Write transmit payload

void NRF24\_write\_payload(const void\* buf, uint8\_t len);

//8. Read receive payload

void NRF24\_read\_payload(void\* buf, uint8\_t len);

//9. Flush Tx buffer

void NRF24\_flush\_tx(void);

//10. Flush Rx buffer

void NRF24\_flush\_rx(void);

//11. Get status register value

uint8\_t NRF24\_get\_status(void);

//12. Begin function

void NRF24\_begin(GPIO\_TypeDef \*nrf24PORT, uint16\_t nrfCSN\_Pin, uint16\_t nrfCE\_Pin, SPI\_HandleTypeDef nrfSPI);

//13. Listen on open pipes for reading (Must call NRF24\_openReadingPipe() first)

void NRF24\_startListening(void);

//14. Stop listening (essential before any write operation)

void NRF24\_stopListening(void);

//15. Write(Transmit data), returns true if successfully sent

bool NRF24\_write( const void\* buf, uint8\_t len );

//16. Check for available data to read

bool NRF24\_available(void);

//17. Read received data

bool NRF24\_read( void\* buf, uint8\_t len );

//18. Open Tx pipe for writing (Cannot perform this while Listenning, has to call NRF24\_stopListening)

void NRF24\_openWritingPipe(uint64\_t address);

//19. Open reading pipe

void NRF24\_openReadingPipe(uint8\_t number, uint64\_t address);

//20 set transmit retries (rf24\_Retries\_e) and delay

void NRF24\_setRetries(uint8\_t delay, uint8\_t count);

//21. Set RF channel frequency

void NRF24\_setChannel(uint8\_t channel);

//22. Set payload size

void NRF24\_setPayloadSize(uint8\_t size);

//23. Get payload size

uint8\_t NRF24\_getPayloadSize(void);

//24. Get dynamic payload size, of latest packet received

uint8\_t NRF24\_getDynamicPayloadSize(void);

//25. Enable payload on Ackknowledge packet

void NRF24\_enableAckPayload(void);

//26. Enable dynamic payloads

void NRF24\_enableDynamicPayloads(void);

void NRF24\_disableDynamicPayloads(void);

//27. Check if module is NRF24L01+ or normal module

bool NRF24\_isNRF\_Plus(void) ;

//28. Set Auto Ack for all

void NRF24\_setAutoAck(bool enable);

//29. Set Auto Ack for certain pipe

void NRF24\_setAutoAckPipe( uint8\_t pipe, bool enable ) ;

//30. Set transmit power level

void NRF24\_setPALevel( rf24\_pa\_dbm\_e level ) ;

//31. Get transmit power level

rf24\_pa\_dbm\_e NRF24\_getPALevel( void ) ;

//32. Set data rate (250 Kbps, 1Mbps, 2Mbps)

bool NRF24\_setDataRate(rf24\_datarate\_e speed);

//33. Get data rate

rf24\_datarate\_e NRF24\_getDataRate( void );

//34. Set crc length (disable, 8-bits or 16-bits)

void NRF24\_setCRCLength(rf24\_crclength\_e length);

//35. Get CRC length

rf24\_crclength\_e NRF24\_getCRCLength(void);

//36. Disable CRC

void NRF24\_disableCRC( void ) ;

//37. power up

void NRF24\_powerUp(void) ;

//38. power down

void NRF24\_powerDown(void);

//39. Check if data are available and on which pipe (Use this for multiple rx pipes)

bool NRF24\_availablePipe(uint8\_t\* pipe\_num);

//40. Start write (for IRQ mode)

void NRF24\_startWrite( const void\* buf, uint8\_t len );

//41. Write acknowledge payload

void NRF24\_writeAckPayload(uint8\_t pipe, const void\* buf, uint8\_t len);

//42. Check if an Ack payload is available

bool NRF24\_isAckPayloadAvailable(void);

//43. Check interrupt flags

void NRF24\_whatHappened(bool \*tx\_ok,bool \*tx\_fail,bool \*rx\_ready);

//44. Test if there is a carrier on the previous listenning period (useful to check for intereference)

bool NRF24\_testCarrier(void);

//45. Test if a signal carrier exists (=> -64dB), only for NRF24L01+

bool NRF24\_testRPD(void) ;

//46. Reset Status

void NRF24\_resetStatus(void);

//47. ACTIVATE cmd

void NRF24\_ACTIVATE\_cmd(void);

//48. Get AckPayload Size

uint8\_t NRF24\_GetAckPayloadSize(void);

//\*\*\*\*\*\*\*\*\*\* DEBUG Functions \*\*\*\*\*\*\*\*\*\*//

//1. Print radio settings

void printRadioSettings(void);

//2. Print Status

void printStatusReg(void);

//3. Print Config

void printConfigReg(void);

//4. Init Variables

void nrf24\_DebugUART\_Init(UART\_HandleTypeDef nrf24Uart);

//5. FIFO Status

void printFIFOstatus(void);

static void MX\_GPIO\_Init(void);

static void MX\_SPI2\_Init(void);

static void MX\_USART1\_UART\_Init(void);

uint64\_t RxpipeAddrs = 0x11223344AA;

char myRxData[50];

char myAckPayload[32] = "Ack by STMF7!";

int Receiver(void)

{

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_SPI2\_Init();

MX\_USART1\_UART\_Init();

/\* USER CODE BEGIN 2 \*/

NRF24\_begin(GPIOA, GPIO\_PIN\_8, GPIO\_PIN\_15, hspi2);

nrf24\_DebugUART\_Init(huart1);

printRadioSettings();

NRF24\_setAutoAck(true);

NRF24\_setChannel(52);

NRF24\_setPayloadSize(32);

NRF24\_openReadingPipe(1, RxpipeAddrs);

NRF24\_enableDynamicPayloads();

NRF24\_enableAckPayload();

NRF24\_startListening();

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

if(NRF24\_available())

{

NRF24\_read(myRxData, 32);

printf("the data received is %s\n",myRxData);

NRF24\_writeAckPayload(1, myAckPayload, 32);

myRxData[32] = '\r'; myRxData[32+1] = '\n';

HAL\_UART\_Transmit(&huart1, (uint8\_t \*)myRxData, 32+2, 10);

}

}

/\* USER CODE END 3 \*/

}

/\* SPI2 init function \*/

static void MX\_SPI2\_Init(void)

{

hspi2.Instance = SPI2;

hspi2.Init.Mode = SPI\_MODE\_MASTER;

hspi2.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi2.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi2.Init.CLKPolarity = SPI\_POLARITY\_LOW;

hspi2.Init.CLKPhase = SPI\_PHASE\_1EDGE;

hspi2.Init.NSS = SPI\_NSS\_SOFT;

hspi2.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_32;

hspi2.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi2.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi2.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi2.Init.CRCPolynomial = 7;

hspi2.Init.CRCLength = SPI\_CRC\_LENGTH\_DATASIZE;

hspi2.Init.NSSPMode = SPI\_NSS\_PULSE\_ENABLE;

if (HAL\_SPI\_Init(&hspi2) != HAL\_OK)

{

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

}

}

/\* USART1 init function \*/

static void MX\_USART1\_UART\_Init(void)

{

huart1.Instance = USART1;

huart1.Init.BaudRate = 115200;

huart1.Init.WordLength = UART\_WORDLENGTH\_8B;

huart1.Init.StopBits = UART\_STOPBITS\_1;

huart1.Init.Parity = UART\_PARITY\_NONE;

huart1.Init.Mode = UART\_MODE\_TX\_RX;

huart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart1.Init.OverSampling = UART\_OVERSAMPLING\_16;

huart1.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

huart1.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

if (HAL\_UART\_Init(&huart1) != HAL\_OK)

{

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

}

}

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\_GPIOI\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_15|GPIO\_PIN\_8, GPIO\_PIN\_RESET);

/\*Configure GPIO pins : PA15 PA8 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_15|GPIO\_PIN\_8;

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

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

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

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

}