|  |
| --- |
| // RH\_NRF24.h |
|  | // Author: Mike McCauley |
|  | // Copyright (C) 2012 Mike McCauley |
|  | // $Id: RH\_NRF24.h,v 1.21 2020/06/15 23:39:39 mikem Exp $ |
|  | // |
|  |  |
|  | #ifndef RH\_NRF24\_h |
|  | #define RH\_NRF24\_h |
|  |  |
|  | #include <RHGenericSPI.h> |
|  | #include <RHNRFSPIDriver.h> |
|  |  |
|  | // This is the maximum number of bytes that can be carried by the nRF24. |
|  | // We use some for headers, keeping fewer for RadioHead messages |
|  | #define RH\_NRF24\_MAX\_PAYLOAD\_LEN 32 |
|  |  |
|  | // The length of the headers we add. |
|  | // The headers are inside the nRF24 payload |
|  | #define RH\_NRF24\_HEADER\_LEN 4 |
|  |  |
|  | // This is the maximum RadioHead user message length that can be supported by this library. Limited by |
|  | // the supported message lengths in the nRF24 |
|  | #define RH\_NRF24\_MAX\_MESSAGE\_LEN (RH\_NRF24\_MAX\_PAYLOAD\_LEN-RH\_NRF24\_HEADER\_LEN) |
|  |  |
|  | // SPI Command names |
|  | #define RH\_NRF24\_COMMAND\_R\_REGISTER 0x00 |
|  | #define RH\_NRF24\_COMMAND\_W\_REGISTER 0x20 |
|  | #define RH\_NRF24\_COMMAND\_ACTIVATE 0x50 // only on RFM73 ? |
|  | #define RH\_NRF24\_COMMAND\_R\_RX\_PAYLOAD 0x61 |
|  | #define RH\_NRF24\_COMMAND\_W\_TX\_PAYLOAD 0xa0 |
|  | #define RH\_NRF24\_COMMAND\_FLUSH\_TX 0xe1 |
|  | #define RH\_NRF24\_COMMAND\_FLUSH\_RX 0xe2 |
|  | #define RH\_NRF24\_COMMAND\_REUSE\_TX\_PL 0xe3 |
|  | #define RH\_NRF24\_COMMAND\_R\_RX\_PL\_WID 0x60 |
|  | #define RH\_NRF24\_COMMAND\_W\_ACK\_PAYLOAD(pipe) (0xa8|(pipe&0x7)) |
|  | #define RH\_NRF24\_COMMAND\_W\_TX\_PAYLOAD\_NOACK 0xb0 |
|  | #define RH\_NRF24\_COMMAND\_NOP 0xff |
|  |  |
|  | // Register names |
|  | #define RH\_NRF24\_REGISTER\_MASK 0x1f |
|  | #define RH\_NRF24\_REG\_00\_CONFIG 0x00 |
|  | #define RH\_NRF24\_REG\_01\_EN\_AA 0x01 |
|  | #define RH\_NRF24\_REG\_02\_EN\_RXADDR 0x02 |
|  | #define RH\_NRF24\_REG\_03\_SETUP\_AW 0x03 |
|  | #define RH\_NRF24\_REG\_04\_SETUP\_RETR 0x04 |
|  | #define RH\_NRF24\_REG\_05\_RF\_CH 0x05 |
|  | #define RH\_NRF24\_REG\_06\_RF\_SETUP 0x06 |
|  | #define RH\_NRF24\_REG\_07\_STATUS 0x07 |
|  | #define RH\_NRF24\_REG\_08\_OBSERVE\_TX 0x08 |
|  | #define RH\_NRF24\_REG\_09\_RPD 0x09 |
|  | #define RH\_NRF24\_REG\_0A\_RX\_ADDR\_P0 0x0a |
|  | #define RH\_NRF24\_REG\_0B\_RX\_ADDR\_P1 0x0b |
|  | #define RH\_NRF24\_REG\_0C\_RX\_ADDR\_P2 0x0c |
|  | #define RH\_NRF24\_REG\_0D\_RX\_ADDR\_P3 0x0d |
|  | #define RH\_NRF24\_REG\_0E\_RX\_ADDR\_P4 0x0e |
|  | #define RH\_NRF24\_REG\_0F\_RX\_ADDR\_P5 0x0f |
|  | #define RH\_NRF24\_REG\_10\_TX\_ADDR 0x10 |
|  | #define RH\_NRF24\_REG\_11\_RX\_PW\_P0 0x11 |
|  | #define RH\_NRF24\_REG\_12\_RX\_PW\_P1 0x12 |
|  | #define RH\_NRF24\_REG\_13\_RX\_PW\_P2 0x13 |
|  | #define RH\_NRF24\_REG\_14\_RX\_PW\_P3 0x14 |
|  | #define RH\_NRF24\_REG\_15\_RX\_PW\_P4 0x15 |
|  | #define RH\_NRF24\_REG\_16\_RX\_PW\_P5 0x16 |
|  | #define RH\_NRF24\_REG\_17\_FIFO\_STATUS 0x17 |
|  | #define RH\_NRF24\_REG\_1C\_DYNPD 0x1c |
|  | #define RH\_NRF24\_REG\_1D\_FEATURE 0x1d |
|  |  |
|  | // These register masks etc are named wherever possible |
|  | // corresponding to the bit and field names in the nRF24L01 Product Specification |
|  | // #define RH\_NRF24\_REG\_00\_CONFIG 0x00 |
|  | #define RH\_NRF24\_MASK\_RX\_DR 0x40 |
|  | #define RH\_NRF24\_MASK\_TX\_DS 0x20 |
|  | #define RH\_NRF24\_MASK\_MAX\_RT 0x10 |
|  | #define RH\_NRF24\_EN\_CRC 0x08 |
|  | #define RH\_NRF24\_CRCO 0x04 |
|  | #define RH\_NRF24\_PWR\_UP 0x02 |
|  | #define RH\_NRF24\_PRIM\_RX 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_01\_EN\_AA 0x01 |
|  | #define RH\_NRF24\_ENAA\_P5 0x20 |
|  | #define RH\_NRF24\_ENAA\_P4 0x10 |
|  | #define RH\_NRF24\_ENAA\_P3 0x08 |
|  | #define RH\_NRF24\_ENAA\_P2 0x04 |
|  | #define RH\_NRF24\_ENAA\_P1 0x02 |
|  | #define RH\_NRF24\_ENAA\_P0 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_02\_EN\_RXADDR 0x02 |
|  | #define RH\_NRF24\_ERX\_P5 0x20 |
|  | #define RH\_NRF24\_ERX\_P4 0x10 |
|  | #define RH\_NRF24\_ERX\_P3 0x08 |
|  | #define RH\_NRF24\_ERX\_P2 0x04 |
|  | #define RH\_NRF24\_ERX\_P1 0x02 |
|  | #define RH\_NRF24\_ERX\_P0 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_03\_SETUP\_AW 0x03 |
|  | #define RH\_NRF24\_AW\_3\_BYTES 0x01 |
|  | #define RH\_NRF24\_AW\_4\_BYTES 0x02 |
|  | #define RH\_NRF24\_AW\_5\_BYTES 0x03 |
|  |  |
|  | // #define RH\_NRF24\_REG\_04\_SETUP\_RETR 0x04 |
|  | #define RH\_NRF24\_ARD 0xf0 |
|  | #define RH\_NRF24\_ARC 0x0f |
|  |  |
|  | // #define RH\_NRF24\_REG\_05\_RF\_CH 0x05 |
|  | #define RH\_NRF24\_RF\_CH 0x7f |
|  |  |
|  | // #define RH\_NRF24\_REG\_06\_RF\_SETUP 0x06 |
|  | #define RH\_NRF24\_CONT\_WAVE 0x80 |
|  | #define RH\_NRF24\_RF\_DR\_LOW 0x20 |
|  | #define RH\_NRF24\_PLL\_LOCK 0x10 |
|  | #define RH\_NRF24\_RF\_DR\_HIGH 0x08 |
|  | #define RH\_NRF24\_PWR 0x06 |
|  | #define RH\_NRF24\_PWR\_m18dBm 0x00 |
|  | #define RH\_NRF24\_PWR\_m12dBm 0x02 |
|  | #define RH\_NRF24\_PWR\_m6dBm 0x04 |
|  | #define RH\_NRF24\_PWR\_0dBm 0x06 |
|  | #define RH\_NRF24\_LNA\_HCURR 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_07\_STATUS 0x07 |
|  | #define RH\_NRF24\_RX\_DR 0x40 |
|  | #define RH\_NRF24\_TX\_DS 0x20 |
|  | #define RH\_NRF24\_MAX\_RT 0x10 |
|  | #define RH\_NRF24\_RX\_P\_NO 0x0e |
|  | #define RH\_NRF24\_STATUS\_TX\_FULL 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_08\_OBSERVE\_TX 0x08 |
|  | #define RH\_NRF24\_PLOS\_CNT 0xf0 |
|  | #define RH\_NRF24\_ARC\_CNT 0x0f |
|  |  |
|  | // #define RH\_NRF24\_REG\_09\_RPD 0x09 |
|  | #define RH\_NRF24\_RPD 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_17\_FIFO\_STATUS 0x17 |
|  | #define RH\_NRF24\_TX\_REUSE 0x40 |
|  | #define RH\_NRF24\_TX\_FULL 0x20 |
|  | #define RH\_NRF24\_TX\_EMPTY 0x10 |
|  | #define RH\_NRF24\_RX\_FULL 0x02 |
|  | #define RH\_NRF24\_RX\_EMPTY 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_1C\_DYNPD 0x1c |
|  | #define RH\_NRF24\_DPL\_ALL 0x3f |
|  | #define RH\_NRF24\_DPL\_P5 0x20 |
|  | #define RH\_NRF24\_DPL\_P4 0x10 |
|  | #define RH\_NRF24\_DPL\_P3 0x08 |
|  | #define RH\_NRF24\_DPL\_P2 0x04 |
|  | #define RH\_NRF24\_DPL\_P1 0x02 |
|  | #define RH\_NRF24\_DPL\_P0 0x01 |
|  |  |
|  | // #define RH\_NRF24\_REG\_1D\_FEATURE 0x1d |
|  | #define RH\_NRF24\_EN\_DPL 0x04 |
|  | #define RH\_NRF24\_EN\_ACK\_PAY 0x02 |
|  | #define RH\_NRF24\_EN\_DYN\_ACK 0x01 |
|  |  |
|  |  |
|  | ///////////////////////////////////////////////////////////////////// |
|  | /// \class RH\_NRF24 RH\_NRF24.h <RH\_NRF24.h> |
|  | /// \brief Send and receive unaddressed, unreliable datagrams by nRF24L01 and compatible transceivers. |
|  | /// |
|  | /// Supported transceivers include: |
|  | /// - Nordic nRF24 based 2.4GHz radio modules, such as nRF24L01 http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01 |
|  | /// and other compatible transceivers. |
|  | /// - nRF24L01p with PA and LNA modules that produce a higher power output similar to this one: |
|  | /// http://www.elecfreaks.com/wiki/index.php?title=2.4G\_Wireless\_nRF24L01p\_with\_PA\_and\_LNA |
|  | /// - Sparkfun WRL-00691 module with nRF24L01 https://www.sparkfun.com/products/691 |
|  | /// or WRL-00705 https://www.sparkfun.com/products/705 etc. |
|  | /// - Hope-RF RFM73 http://www.hoperf.com/rf/2.4g\_module/RFM73.htm and |
|  | /// http://www.anarduino.com/details.jsp?pid=121 |
|  | /// and compatible devices (such as BK2423). nRF24L01 and RFM73 can interoperate |
|  | /// with each other. |
|  | /// |
|  | /// This base class provides basic functions for sending and receiving unaddressed, unreliable datagrams |
|  | /// of arbitrary length to 28 octets per packet. Use one of the Manager classes to get addressing and |
|  | /// acknowledgement reliability, routing, meshes etc. |
|  | /// |
|  | /// The nRF24L01 (http://www.sparkfun.com/datasheets/Wireless/Nordic/nRF24L01P\_Product\_Specification\_1\_0.pdf) |
|  | /// is a low-cost 2.4GHz ISM transceiver module. It supports a number of channel frequencies in the 2.4GHz band |
|  | /// and a range of data rates. |
|  | /// |
|  | /// This library provides functions for sending and receiving messages of up to 28 octets on any |
|  | /// frequency supported by the nRF24L01, at a selected data rate. |
|  | /// |
|  | /// Several nRF24L01 modules can be connected to an Arduino, permitting the construction of translators |
|  | /// and frequency changers, etc. |
|  | /// |
|  | /// The nRF24 transceiver is configured to use Enhanced Shockburst with no acknowledgement and no retransmits. |
|  | /// TX\_ADDR and RX\_ADDR\_P0 are set to the network address. If you need the low level auto-acknowledgement |
|  | /// feature supported by this chip, you can use our original NRF24 library |
|  | /// at http://www.airspayce.com/mikem/arduino/NRF24 |
|  | /// |
|  | /// Naturally, for any 2 radios to communicate that must be configured to use the same frequency and |
|  | /// data rate, and with identical network addresses. |
|  | /// |
|  | /// Example Arduino programs are included to show the main modes of use. |
|  | /// |
|  | /// \par Packet Format |
|  | /// |
|  | /// All messages sent and received by this class conform to this packet format, as specified by |
|  | /// the nRF24L01 product specification: |
|  | /// |
|  | /// - 1 octets PREAMBLE |
|  | /// - 3 to 5 octets NETWORK ADDRESS |
|  | /// - 9 bits packet control field |
|  | /// - 0 to 32 octets PAYLOAD, consisting of: |
|  | /// - 1 octet TO header |
|  | /// - 1 octet FROM header |
|  | /// - 1 octet ID header |
|  | /// - 1 octet FLAGS header |
|  | /// - 0 to 28 octets of user message |
|  | /// - 2 octets CRC |
|  | /// |
|  | /// \par Connecting nRF24L01 to Arduino |
|  | /// |
|  | /// The electrical connection between the nRF24L01 and the Arduino require 3.3V, the 3 x SPI pins (SCK, SDI, SDO), |
|  | /// a Chip Enable pin and a Slave Select pin. |
|  | /// If you are using the Sparkfun WRL-00691 module, it has a voltage regulator on board and |
|  | /// can be should with 5V VCC if possible. |
|  | /// The examples below assume the Sparkfun WRL-00691 module |
|  | /// |
|  | /// Connect the nRF24L01 to most Arduino's like this (Caution, Arduino Mega has different pins for SPI, |
|  | /// see below). Use these same connections for Teensy 3.1 (use 3.3V not 5V Vcc). |
|  | /// \code |
|  | /// Arduino Sparkfun WRL-00691 |
|  | /// 5V-----------VCC (3.3V to 7V in) |
|  | /// pin D8-----------CE (chip enable in) |
|  | /// SS pin D10----------CSN (chip select in) |
|  | /// SCK pin D13----------SCK (SPI clock in) |
|  | /// MOSI pin D11----------SDI (SPI Data in) |
|  | /// MISO pin D12----------SDO (SPI data out) |
|  | /// IRQ (Interrupt output, not connected) |
|  | /// GND----------GND (ground in) |
|  | /// \endcode |
|  | /// |
|  | /// For an Arduino Leonardo (the SPI pins do not come out on the Digital pins as for normal Arduino, but only |
|  | /// appear on the ICSP header) |
|  | /// \code |
|  | /// Leonardo Sparkfun WRL-00691 |
|  | /// 5V-----------VCC (3.3V to 7V in) |
|  | /// pin D8-----------CE (chip enable in) |
|  | /// SS pin D10----------CSN (chip select in) |
|  | /// SCK ICSP pin 3----------SCK (SPI clock in) |
|  | /// MOSI ICSP pin 4----------SDI (SPI Data in) |
|  | /// MISO ICSP pin 1----------SDO (SPI data out) |
|  | /// IRQ (Interrupt output, not connected) |
|  | /// GND----------GND (ground in) |
|  | /// \endcode |
|  | /// and initialise the NRF24 object like this to explicitly set the SS pin |
|  | /// NRF24 nrf24(8, 10); |
|  | /// |
|  | /// For an Arduino Due (the SPI pins do not come out on the Digital pins as for normal Arduino, but only |
|  | /// appear on the SPI header). Use the same connections for Yun with 5V or 3.3V. |
|  | /// \code |
|  | /// Due Sparkfun WRL-00691 |
|  | /// 3.3V-----------VCC (3.3V to 7V in) |
|  | /// pin D8-----------CE (chip enable in) |
|  | /// SS pin D10----------CSN (chip select in) |
|  | /// SCK SPI pin 3----------SCK (SPI clock in) |
|  | /// MOSI SPI pin 4----------SDI (SPI Data in) |
|  | /// MISO SPI pin 1----------SDO (SPI data out) |
|  | /// IRQ (Interrupt output, not connected) |
|  | /// GND----------GND (ground in) |
|  | /// \endcode |
|  | /// and initialise the NRF24 object with the default constructor |
|  | /// NRF24 nrf24; |
|  | /// |
|  | /// For an Arduino Mega: |
|  | /// \code |
|  | /// Mega Sparkfun WRL-00691 |
|  | /// 5V-----------VCC (3.3V to 7V in) |
|  | /// pin D8-----------CE (chip enable in) |
|  | /// SS pin D53----------CSN (chip select in) |
|  | /// SCK pin D52----------SCK (SPI clock in) |
|  | /// MOSI pin D51----------SDI (SPI Data in) |
|  | /// MISO pin D50----------SDO (SPI data out) |
|  | /// IRQ (Interrupt output, not connected) |
|  | /// GND----------GND (ground in) |
|  | /// \endcode |
|  | /// and you can then use the constructor RH\_NRF24(8, 53). |
|  | /// |
|  | /// For an Itead Studio IBoard Pro http://imall.iteadstudio.com/iboard-pro.html, connected by hardware SPI to the |
|  | /// ITDB02 Parallel LCD Module Interface pins: |
|  | /// \code |
|  | /// IBoard Signal=ITDB02 pin Sparkfun WRL-00691 |
|  | /// 3.3V 37-----------VCC (3.3V to 7V in) |
|  | /// D2 28-----------CE (chip enable in) |
|  | /// D29 27----------CSN (chip select in) |
|  | /// SCK D52 32----------SCK (SPI clock in) |
|  | /// MOSI D51 34----------SDI (SPI Data in) |
|  | /// MISO D50 30----------SDO (SPI data out) |
|  | /// IRQ (Interrupt output, not connected) |
|  | /// GND 39----------GND (ground in) |
|  | /// \endcode |
|  | /// And initialise like this: |
|  | /// \code |
|  | /// RH\_NRF24 nrf24(2, 29); |
|  | /// \endcode |
|  | /// |
|  | /// For an Itead Studio IBoard Pro http://imall.iteadstudio.com/iboard-pro.html, connected by software SPI to the |
|  | /// nRF24L01+ Module Interface pins. CAUTION: performance of software SPI is very slow and is not |
|  | /// compatible with other modules running hardware SPI. |
|  | /// \code |
|  | /// IBoard Signal=Module pin Sparkfun WRL-00691 |
|  | /// 3.3V 2----------VCC (3.3V to 7V in) |
|  | /// D12 3-----------CE (chip enable in) |
|  | /// D29 4----------CSN (chip select in) |
|  | /// D9 5----------SCK (SPI clock in) |
|  | /// D8 6----------SDI (SPI Data in) |
|  | /// D7 7----------SDO (SPI data out) |
|  | /// IRQ (Interrupt output, not connected) |
|  | /// GND 1----------GND (ground in) |
|  | /// \endcode |
|  | /// And initialise like this: |
|  | /// \code |
|  | /// #include <SPI.h> |
|  | /// #include <RH\_NRF24.h> |
|  | /// #include <RHSoftwareSPI.h> |
|  | /// Singleton instance of the radio driver |
|  | /// RHSoftwareSPI spi; |
|  | /// RH\_NRF24 nrf24(12, 11, spi); |
|  | /// void setup() { |
|  | /// spi.setPins(7, 8, 9); |
|  | /// .... |
|  | /// \endcode |
|  | /// |
|  | /// |
|  | /// For Raspberry Pi with Sparkfun WRL-00691 |
|  | /// \code |
|  | /// Raspberry Pi P1 pin Sparkfun WRL-00691 |
|  | /// 5V 2-----------VCC (3.3V to 7V in) |
|  | /// GPIO25 22-----------CE (chip enable in) |
|  | /// GPIO8 24----------CSN (chip select in) |
|  | /// GPIO11 23----------SCK (SPI clock in) |
|  | /// GPIO10 19----------SDI (SPI Data in) |
|  | /// GPIO9 21----------SDO (SPI data out) |
|  | /// IRQ (Interrupt output, not connected) |
|  | /// GND 6----------GND (ground in) |
|  | /// \endcode |
|  | /// and initialise like this: |
|  | /// \code |
|  | /// RH\_NRF24 nrf24(RPI\_V2\_GPIO\_P1\_22, RPI\_V2\_GPIO\_P1\_24); |
|  | /// \endcode |
|  | /// See the example program and Makefile in examples/raspi. Requires bcm2835 library to be previously installed. |
|  | /// \code |
|  | /// cd examples/raspi |
|  | /// make |
|  | /// sudo ./RasPiRH |
|  | /// \endcode |
|  | /// \code |
|  | /// |
|  | /// You can override the default settings for the CSN and CE pins |
|  | /// in the NRF24() constructor if you wish to connect the slave select CSN to other than the normal one for your |
|  | /// |
|  | /// Caution: on the Raspberry Pi Zero, the hardware SPI, is only connected to the |
|  | /// ICSP-header. So in order to use the RF, one must either connect it to the SPI-pins |
|  | /// of the ICSP-header or use the software SPI provided by RHSoftwareSPI. |
|  | /// the mapping of the SPI-Pins for each board here: |
|  | /// https://www.arduino.cc/en/Reference/SPI |
|  | /// Arduino (D10 for Diecimila, Uno etc and D53 for Mega) |
|  | /// |
|  | /// Caution: on some Arduinos such as the Mega 2560, if you set the slave select pin to be other than the usual SS |
|  | /// pin (D53 on Mega 2560), you may need to set the usual SS pin to be an output to force the Arduino into SPI |
|  | /// master mode. |
|  | /// |
|  | /// Caution: this module has not been proved to work with Leonardo, at least without level |
|  | /// shifters between the nRF24 and the Leonardo. Tests seem to indicate that such level shifters would be required |
|  | /// with Leonardo to make it work. |
|  | /// |
|  | /// It is possible to have 2 radios conected to one arduino, provided each radio has its own |
|  | /// CSN and CE line (SCK, SDI and SDO are common to both radios) |
|  | /// |
|  | /// \par SPI Interface |
|  | /// |
|  | /// You can interface to nRF24L01 with with hardware or software SPI. Use of software SPI with the RHSoftwareSPI |
|  | /// class depends on a fast enough processor and digitalOut() functions to achieve a high enough SPI bus frequency. |
|  | /// If you observe reliable behaviour with the default hardware SPI RHHardwareSPI, but unreliable behaviour |
|  | /// with Software SPI RHSoftwareSPI, it may be due to slow CPU performance. |
|  | /// |
|  | /// Initialisation example with hardware SPI |
|  | /// \code |
|  | /// #include <RH\_NRF24.h> |
|  | /// RH\_NRF24 driver; |
|  | /// RHReliableDatagram manager(driver, CLIENT\_ADDRESS); |
|  | /// \endcode |
|  | /// |
|  | /// Initialisation example with software SPI |
|  | /// \code |
|  | /// #include <RH\_NRF24.h> |
|  | /// #include <RHSoftwareSPI.h> |
|  | /// RHSoftwareSPI spi; |
|  | /// RH\_NRF24 driver(8, 10, spi); |
|  | /// RHReliableDatagram manager(driver, CLIENT\_ADDRESS); |
|  | /// \endcode |
|  | /// |
|  | /// \par Example programs |
|  | /// |
|  | /// Several example programs are provided. |
|  | /// |
|  | /// \par Radio Performance |
|  | /// |
|  | /// Frequency accuracy may be debatable. For nominal frequency of 2401.000 MHz (ie channel 1), |
|  | /// my Yaesu VR-5000 receiver indicated the center frequency for my test radios |
|  | /// was 2401.121 MHz. Its not clear to me if the Yaesu |
|  | /// is the source of the error, but I tend to believe it, which would make the nRF24l01 frequency out by 121kHz. |
|  | /// |
|  | /// The measured power output for a nRF24L01p with PA and LNA set to 0dBm output is about 18dBm. |
|  | /// |
|  | /// \par Radio operating strategy and defaults |
|  | /// |
|  | /// The radio is enabled all the time, and switched between TX and RX modes depending on |
|  | /// whether there is any data to send. Sending data sets the radio to TX mode. |
|  | /// After data is sent, the radio automatically returns to Standby II mode. Calling waitAvailable() or |
|  | /// waitAvailableTimeout() starts the radio in RX mode. |
|  | /// |
|  | /// The radio is configured by default to Channel 2, 2Mbps, 0dBm power, 5 bytes address, payload width 1, CRC enabled |
|  | /// 2 byte CRC, No Auto-Ack mode. Enhanced shockburst is used. |
|  | /// TX and P0 are set to the Network address. Node addresses and decoding are handled with the RH\_NRF24 module. |
|  | /// |
|  | /// \par Memory |
|  | /// |
|  | /// Memory usage of this class is minimal. The compiled client and server sketches are about 6000 bytes on Arduino. |
|  | /// The reliable client and server sketches compile to about 8500 bytes on Arduino. |
|  | /// RAM requirements are minimal. |
|  | /// |
|  | class RH\_NRF24 : public RHNRFSPIDriver |
|  | { |
|  | public: |
|  |  |
|  | /// \brief Defines convenient values for setting data rates in setRF() |
|  | typedef enum |
|  | { |
|  | DataRate1Mbps = 0, ///< 1 Mbps |
|  | DataRate2Mbps, ///< 2 Mbps |
|  | DataRate250kbps ///< 250 kbps |
|  | } DataRate; |
|  |  |
|  | /// \brief Convenient values for setting transmitter power in setRF() |
|  | /// These are designed to agree with the values for RF\_PWR in RH\_NRF24\_REG\_06\_RF\_SETUP |
|  | /// To be passed to setRF(); |
|  | typedef enum |
|  | { |
|  | // Add 20dBm for nRF24L01p with PA and LNA modules |
|  | TransmitPowerm18dBm = 0, ///< On nRF24, -18 dBm |
|  | TransmitPowerm12dBm, ///< On nRF24, -12 dBm |
|  | TransmitPowerm6dBm, ///< On nRF24, -6 dBm |
|  | TransmitPower0dBm, ///< On nRF24, 0 dBm |
|  | // Sigh, different power levels for the same bit patterns on RFM73: |
|  | // On RFM73P-S, there is a Tx power amp, so expect higher power levels, up to 20dBm. Alas |
|  | // there is no clear documentation on the power for different settings :-( |
|  | RFM73TransmitPowerm10dBm = 0, ///< On RFM73, -10 dBm |
|  | RFM73TransmitPowerm5dBm, ///< On RFM73, -5 dBm |
|  | RFM73TransmitPowerm0dBm, ///< On RFM73, 0 dBm |
|  | RFM73TransmitPower5dBm ///< On RFM73, 5 dBm. 20dBm on RFM73P-S2 ? |
|  |  |
|  | } TransmitPower; |
|  |  |
|  | /// Constructor. You can have multiple instances, but each instance must have its own |
|  | /// chip enable and slave select pin. |
|  | /// After constructing, you must call init() to initialise the interface |
|  | /// and the radio module |
|  | /// \param[in] chipEnablePin the Arduino pin to use to enable the chip for transmit/receive |
|  | /// \param[in] slaveSelectPin the Arduino pin number of the output to use to select the NRF24 before |
|  | /// accessing it. Defaults to the normal SS pin for your Arduino (D10 for Diecimila, Uno etc, D53 for Mega, |
|  | /// D10 for Maple) |
|  | /// \param[in] spi Pointer to the SPI interface object to use. |
|  | /// Defaults to the standard Arduino hardware SPI interface |
|  | RH\_NRF24(uint8\_t chipEnablePin = 8, uint8\_t slaveSelectPin = SS, RHGenericSPI& spi = hardware\_spi); |
|  |  |
|  | /// Initialises this instance and the radio module connected to it. |
|  | /// The following steps are taken:g |
|  | /// - Set the chip enable and chip select pins to output LOW, HIGH respectively. |
|  | /// - Initialise the SPI output pins |
|  | /// - Initialise the SPI interface library to 8MHz (Hint, if you want to lower |
|  | /// the SPI frequency (perhaps where you have other SPI shields, low voltages etc), |
|  | /// call SPI.setClockDivider() after init()). |
|  | /// -Flush the receiver and transmitter buffers |
|  | /// - Set the radio to receive with powerUpRx(); |
|  | /// \return true if everything was successful |
|  | bool init(); |
|  |  |
|  | /// Reads a single register from the NRF24 |
|  | /// \param[in] reg Register number, one of RH\_NRF24\_REG\_\* |
|  | /// \return The value of the register |
|  | uint8\_t spiReadRegister(uint8\_t reg); |
|  |  |
|  | /// Writes a single byte to the NRF24, and at the same time reads the current STATUS register |
|  | /// \param[in] reg Register number, one of RH\_NRF24\_REG\_\* |
|  | /// \param[in] val The value to write |
|  | /// \return the current STATUS (read while the command is sent) |
|  | uint8\_t spiWriteRegister(uint8\_t reg, uint8\_t val); |
|  |  |
|  | /// Reads a number of consecutive registers from the NRF24 using burst read mode |
|  | /// \param[in] reg Register number of the first register, one of RH\_NRF24\_REG\_\* |
|  | /// \param[in] dest Array to write the register values to. Must be at least len bytes |
|  | /// \param[in] len Number of bytes to read |
|  | /// \return the current STATUS (read while the command is sent) |
|  | uint8\_t spiBurstReadRegister(uint8\_t reg, uint8\_t\* dest, uint8\_t len); |
|  |  |
|  | /// Write a number of consecutive registers using burst write mode |
|  | /// \param[in] reg Register number of the first register, one of RH\_NRF24\_REG\_\* |
|  | /// \param[in] src Array of new register values to write. Must be at least len bytes |
|  | /// \param[in] len Number of bytes to write |
|  | /// \return the current STATUS (read while the command is sent) |
|  | uint8\_t spiBurstWriteRegister(uint8\_t reg, uint8\_t\* src, uint8\_t len); |
|  |  |
|  | /// Reads and returns the device status register NRF24\_REG\_02\_DEVICE\_STATUS |
|  | /// \return The value of the device status register |
|  | uint8\_t statusRead(); |
|  |  |
|  | /// Sets the transmit and receive channel number. |
|  | /// The frequency used is (2400 + channel) MHz |
|  | /// \return true on success |
|  | bool setChannel(uint8\_t channel); |
|  |  |
|  | /// Sets the chip configuration that will be used to set |
|  | /// the NRF24 NRF24\_REG\_00\_CONFIG register when in Idle mode. This allows you to change some |
|  | /// chip configuration for compatibility with libraries other than this one. |
|  | /// You should not normally need to call this. |
|  | /// Defaults to NRF24\_EN\_CRC| RH\_NRF24\_CRCO, which is the standard configuration for this library |
|  | /// (2 byte CRC enabled). |
|  | /// \param[in] mode The chip configuration to be used whe in Idle mode. |
|  | /// \return true on success |
|  | bool setOpMode(uint8\_t mode); |
|  |  |
|  | /// Sets the Network address. |
|  | /// Only nodes with the same network address can communicate with each other. You |
|  | /// can set different network addresses in different sets of nodes to isolate them from each other. |
|  | /// Internally, this sets the nRF24 TX\_ADDR and RX\_ADDR\_P0 to be the given network address. |
|  | /// The default network address is 0xE7E7E7E7E7 |
|  | /// \param[in] address The new network address. Must match the network address of any receiving node(s). |
|  | /// \param[in] len Number of bytes of address to set (3 to 5). |
|  | /// \return true on success, false if len is not in the range 3-5 inclusive. |
|  | bool setNetworkAddress(uint8\_t\* address, uint8\_t len); |
|  |  |
|  | /// Sets the data rate and transmitter power to use. Note that the nRF24 and the RFM73 have different |
|  | /// available power levels, and for convenience, 2 different sets of values are available in the |
|  | /// RH\_NRF24::TransmitPower enum. The ones with the RFM73 only have meaning on the RFM73 and compatible |
|  | /// devces. The others are for the nRF24. |
|  | /// \param [in] data\_rate The data rate to use for all packets transmitted and received. One of RH\_NRF24::DataRate. |
|  | /// \param [in] power Transmitter power. One of RH\_NRF24::TransmitPower. |
|  | /// \return true on success |
|  | bool setRF(DataRate data\_rate, TransmitPower power); |
|  |  |
|  | /// Sets the radio in power down mode, with the configuration set to the |
|  | /// last value from setOpMode(). |
|  | /// Sets chip enable to LOW. |
|  | void setModeIdle(); |
|  |  |
|  | /// Sets the radio in RX mode. |
|  | /// Sets chip enable to HIGH to enable the chip in RX mode. |
|  | void setModeRx(); |
|  |  |
|  | /// Sets the radio in TX mode. |
|  | /// Pulses the chip enable LOW then HIGH to enable the chip in TX mode. |
|  | void setModeTx(); |
|  |  |
|  | /// Sends data to the address set by setTransmitAddress() |
|  | /// Sets the radio to TX mode |
|  | /// \param [in] data Data bytes to send. |
|  | /// \param [in] len Number of data bytes to send |
|  | /// \return true on success (which does not necessarily mean the receiver got the message, only that the message was |
|  | /// successfully transmitted). |
|  | bool send(const uint8\_t\* data, uint8\_t len); |
|  |  |
|  | /// Blocks until the current message (if any) |
|  | /// has been transmitted |
|  | /// \return true on success, false if the chip is not in transmit mode or other transmit failure |
|  | virtual bool waitPacketSent(); |
|  |  |
|  | /// Indicates if the chip is in transmit mode and |
|  | /// there is a packet currently being transmitted |
|  | /// \return true if the chip is in transmit mode and there is a transmission in progress |
|  | bool isSending(); |
|  |  |
|  | /// Prints the value of all chip registers |
|  | /// to the Serial device if RH\_HAVE\_SERIAL is defined for the current platform |
|  | /// For debugging purposes only. |
|  | /// \return true on success |
|  | bool printRegisters(); |
|  |  |
|  | /// Checks whether a received message is available. |
|  | /// This can be called multiple times in a timeout loop |
|  | /// \return true if a complete, valid message has been received and is able to be retrieved by |
|  | /// recv() |
|  | bool available(); |
|  |  |
|  | /// Turns the receiver on if it not already on. |
|  | /// If there is a valid message available, copy it to buf and return true |
|  | /// else return false. |
|  | /// If a message is copied, \*len is set to the length (Caution, 0 length messages are permitted). |
|  | /// You should be sure to call this function frequently enough to not miss any messages |
|  | /// It is recommended that you call it in your main loop. |
|  | /// \param[in] buf Location to copy the received message |
|  | /// \param[in,out] len Pointer to available space in buf. Set to the actual number of octets copied. |
|  | /// \return true if a valid message was copied to buf |
|  | bool recv(uint8\_t\* buf, uint8\_t\* len); |
|  |  |
|  | /// The maximum message length supported by this driver |
|  | /// \return The maximum message length supported by this driver |
|  | uint8\_t maxMessageLength(); |
|  |  |
|  | /// Sets the radio into Power Down mode. |
|  | /// If successful, the radio will stay in Power Down mode until woken by |
|  | /// changing mode it idle, transmit or receive (eg by calling send(), recv(), available() etc) |
|  | /// Caution: there is a time penalty as the radio takes a finite time to wake from sleep mode. |
|  | /// \return true if sleep mode was successfully entered. |
|  | virtual bool sleep(); |
|  |  |
|  | protected: |
|  | /// Flush the TX FIFOs |
|  | /// \return the value of the device status register |
|  | uint8\_t flushTx(); |
|  |  |
|  | /// Flush the RX FIFOs |
|  | /// \return the value of the device status register |
|  | uint8\_t flushRx(); |
|  |  |
|  | /// Examine the receive buffer to determine whether the message is for this node |
|  | void validateRxBuf(); |
|  |  |
|  | /// Clear our local receive buffer |
|  | void clearRxBuf(); |
|  |  |
|  | private: |
|  | /// This idle mode chip configuration |
|  | uint8\_t \_configuration; |
|  |  |
|  | /// the number of the chip enable pin |
|  | uint8\_t \_chipEnablePin; |
|  |  |
|  | /// Number of octets in the buffer |
|  | uint8\_t \_bufLen; |
|  |  |
|  | /// The receiver/transmitter buffer |
|  | uint8\_t \_buf[RH\_NRF24\_MAX\_PAYLOAD\_LEN]; |
|  |  |
|  | /// True when there is a valid message in the buffer |
|  | bool \_rxBufValid; |
|  | }; |
|  |  |
|  | /// @example nrf24\_client.pde |
|  | /// @example nrf24\_server.pde |
|  | /// @example nrf24\_encrypted\_client.pde |
|  | /// @example nrf24\_encrypted\_server.pde |
|  | /// @example nrf24\_reliable\_datagram\_client.pde |
|  | /// @example nrf24\_reliable\_datagram\_server.pde |
|  | /// @example RasPiRH.cpp |
|  |  |
|  | #endif |