17/04/10

CC1350 RF driver notes

The present driver of the RF module available on CC1350 only covers the so-called “proprietary mode” of the chip to the minimum extent that allows us to run TARP. No ZigBee/Bluetooth support is available yet, although such support will be implemented as soon as the need arises.

Controlling the driver (a summary of *physopt* calls)

Here is the full list of *physopts* accepted by the driver. A physopt is identified by a symbolic constant, whose name always starts with the prefix PHYSOPT_, appearing as the first argument of a physopt call, which, in its most general format, looks like this:

```
val = tcv_control (PHYSOPT_..., arg);
```

When we say that a physopt accepts an argument, it means that the second argument of tcv_control is interpreted by the operation. If so, that argument is of type address: whatever that address points to depends on the physopt. The argument may happen to be optional, i.e., the pointer can be NULL. If no argument is mentioned in a physopt's description, it means that the physopt takes no argument (if specified, the argument will be ignored). Some physopts return (always simple) values of type int, some others return no value of importance. In the former case, if the argument is not NULL, the returned value is also stored at the argument pointer (as a word). In the latter case, the actual value returned by tcv_control is zero and nothing is stored at the argument pointer, even if it isn't NULL.

Unconditional (i.e., non-optional) calls

These operations are available *always*, i.e., regardless of the compilation options of the driver.

PHYSOPT_STATUS

> Returns the driver status. It can only be 2 (if the receiver is off) and 3 (if the receiver is on). Bit number 1 (second from the right) is a legacy status of the transmitter (which these days is formally always on). I am thinking of retiring this feature (i.e., PHYSOPT_STATUS) altogether. The only reason it is still around is that LibComms uses it (and I prefer not to mess with Wlodek's code).

PHYSOPT_RXON

> Switches the receiver on.

PHYSOPT_RXOFF

> Switches the receiver off. Note: a spontaneous transmission when the receiver is switched off takes slightly more time than when the receiver is on. This is because the chip must be powered up (some of its registers reloaded) before the transmission. Following the first transmission in such a state, the chip will remain powered up for *offdelay* milliseconds expecting more outgoing packets. The default value of *offdelay* (settable with PHYSOPT_SETPARAMS, see below) is 256. Even if *offdelay* is zero, the transmitter will remain on for as long as the output queue of packets is nonempty. The countdown begins when the queue has been drained.

PHYSOPT_SETSID

Sets the “station” ID (which is really the network ID). Note that values 0x0000 and 0xFFFF are special. With the network ID set to any of these two values, the driver will accept all physically receivable packets, regardless of their setting of network ID. If the network ID is 0xFFFF, it will not be copied to the header of an outgoing packet (honouring the value inserted there by the user program).

PHYSOPT_GETSID

Returns the current setting of the network ID.

PHYSOPT_GETMAXPL

Returns the maximum packet length, i.e., the maximum value that can be specified as an argument of tcv_wnps. This is the value that was provided as the second argument of phys_cc1350 when the driver was initialized.

PHYSOPT_ERROR

The name is a bit confusing, but I am just recycling physopt names from cc1100 in a way that keeps them somewhat related.

Writes to the argument address a data structure with the following layout:

```
struct  rfc_propRxOutput_s {
        // Number of packets received and not ignored (CRC OK, address OK, and so on);
        // note that we don't use built-in addressing
        word            nRxOk;
        // Number of received packets with CRC error
        word            nRxNok;
        // Number of received packets with CRC OK ignored due to address missmatch
        // (this should be zero in out setup)
        byte            nRxIgnored;
        // Number of packets stopped by the filter (should be zero in our setup)
        byte            nRxStopped;
        // Number of packets that have been received and discarded due to lack of buffer
        // space (this can happen in principle, although isn't very likely)
        byte            nRxBufFull;
        // RSSI of the last received packet
        byte            lastRssi;
        // Time stamp of last received packet; this is the reading of the so-called RAT
        // timer running at 4MHz
        lword           timeStamp;
};
```

PHYSOPT_CAV

Forces an explicit backoff (blocks the transmitter) for the specified number of (PicOS) milliseconds.

PHYSOPT_GETPOWER

Returns the current transmitter power setting (a number between 0 and 8). Note that 8 means a fixed-power long-distance configuration (RADIO_DEFAULT_POWER == 8). Transmitter power is not settable in that configuration.

PHYSOPT_GETCHANNEL

Returns the current setting of the RF channel which is a number between 0 and 7.

PHYSOPT_SETCHANNEL

Sets the RF channel to the indicated (word) value which must be between 0 and 7. A value larger than 7 is treated as 7.

PHYSOPT_GETRATE

Returns the current bit rate option (a number between 0 and 3). Note that 0 means a special low rate (see below) intended for long-distance communication which is configured at compilation (RADIO_BITRATE_INDEX == 0) and is not changeable.

PHYSOPT_SETPARAMS

Sets the value of *offdelay* (see above).

Optional calls

These operations are only available, if the respective options are selected at compilation.

PHYSOPT_SETPOWER

The argument is between 0 and 7 (any larger value translates into 7). Chooses one of the 8 discrete power levels for the transmitter. This option is only available if:

RADIO_DEFAULT_POWER < 8 and

RADIO_OPTION does not include PXOPTIONS (as explained below)

PHYSOPT_SETRATE

Sets the bit rate to one of three options: 1 – 10000 bps, 2 – 38,400 bps, 3 – 50,000 bps (the default). This call is only available, if RADIO_BITRATE_INDEX is not zero. Otherwise, this compilation parameter selects the hardwired rate number 0 (625 kbps) intended for long-distance communication, which cannot be changed.

Compilation options (static configuration)

The header file PicOS/CC13XX/cc1350.h assigns default values to some constants that configure the operation of the driver. In my effort to maintain compatibility with the driver of our most popular RF module (CC1100), most of the constants used by the CC1350 RF driver inherit their names and meaning from its C1100 predecessor.

All the constants named RADIO_... defined in cc1350.h (as well as some other, less prominent, constants) and encapsulated into #ifndef statements can be defined in options.sys (board_options.sys) thus overriding the default definitions. Here is the list:

Flag options

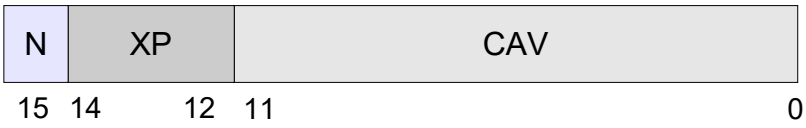
The symbolic constant RADIO_OPTIONS (with the default value of 0) is a configuration of bit flags, which are mostly used to select diagnostics (for debugging or performance evaluation), but also some options that might be of interest to a production praxis.

Two flags are presently used by the driver:

Bit 4 (0x010, RADIO_OPTION_NOCHECKS) is intended to save some code by removing consistency checks in various places where the driver interfaces to the praxis.

Bit 8 (0x100, RADIO_OPTION_PXOPTIONS) enables per-packet parametrization of transmission

power and collision avoidance vectors in exactly the same way as in the CC1100 driver. When this option is compiled in, the last two bytes of every outgoing packet are interpreted as a word with this layout:



where XP is the transmission power for the packet, CAV is the collision avoidance vector, i.e., the initial delay in PicOS milliseconds preceding the packet's transmission attempt, and N is a flag telling whether LBT should be disabled for the packet (when N is 1, the packet is transmitted blindly without LBT).

Note that when acquiring a buffer for outgoing packet (with tcv_wnp) the praxis program must reserve two bytes at the end for the status information regardless whether RADIO_OPTION_PXOPTIONS is selected or not. Normally the initial contents of those bytes are irrelevant. When RADIO_OPTION_PXOPTIONS is compiled in, the praxis program is expected to set those two bytes for every outgoing packet according to the above format. PHYSOPT_SETPOWER is then useless and disabled, because every outgoing packet explicitly declares its own transmission power. PHYSOPT_GETPOWER is still available to return the transmission power of the last transmitted packet. Note that the maximum CAV settable on a per-packet basis is 4095 amounting to about 4 seconds. PHYSOPT_CAV is still formally legal and will be applied as an extra delay to the first outgoing packet.

Note that, with RADIO_OPTION_PXOPTIONS compiled in, whenever a packet, e.g., received from the network, is cloned to the transmit queue (or otherwise *disposed* there) by the praxis (including the VNETI plugin), the last (status) word of the packet must be set to the intended value, as otherwise the driver will derive the transmission parameters of the packet from the accidental contents of the status bytes.

General options

RADIO_DEFAULT_POWER (default setting 7) selects the default transmission power setting. This value indexes an array of settings (somewhat reminiscent of PATABLE for CC1100) which determine the actual transmit power in some messy sort of way. The array is defined in cc1350.h. I have obtained it from SmartRF Studio and (according to the Studio) the entries correspond to these powers in dBm (starting from 0): -10, 0, 2, 4, 6, 8, 10, 12.

When RADIO_DEFAULT_POWER is set to 8, it selects the extra high transmit power of 14dBm. That setting requires a special enabling configuration in the chip configuration which makes it impossible to redefine the transmit power dynamically (or at least, I don't want to deal with this issue at the present stage). Thus, the setting disables PHYSOPT_SETPOWER. It is also incompatible with RADIO_OPTION_PXOPTIONS.

RADIO_DEFAULT_CHANNEL (default setting 0) selects the default channel. Legitimate values are between 0 and 7.

Note that (unlike CC1100) CC1350 has not built in mechanism for handling channels. Thus, the channel setting simply means base frequency adjustment. The base frequency is set at present to 868 MHz (constant CC1350_BASEFREQ) and the channel number translates into the number of MHz added to the base (e.g., channel 3 = 871 MHz).

RADIO_DEFAULT_BITRATE (default setting 3) selects the default bit rate to one of four options: 0 – 625 kbps, 1 – 10000 bps, 2 – 38,400 bps, 3 – 50,000 bps (the default). The constant can be set to 0,

1, 2, 3, or to 625, 10000, 38400, 50000, respectively. The authoritative constant, derived from RADIO_DEFAULT_BITRATE, is RADIO_BITRATE_INDEX whose values are 0, 1, 2, 3.

If RADIO_BITRATE_INDEX is zero, the default rate (625 kbps) cannot be changed (PHYSOPT_SETRATE becomes disabled). This (lowest possible) rate is intended for long-distance communication.

RADIO_SYSTEM_IDENT (default setting 0xAB3553BA) is the layout of the 32-bit sync word to precede the first byte of a transmitted packet. Nodes using different sync words won't be able to communicate, even if they use the same base frequency and the same channel. This attribute can be viewed as a hard (not modifiable by the praxis) extension of the station (network) ID providing one more level of separation for networks operating in the same neighbourhood.

RADIO_DEFAULT_OFFDELAY (default setting 256) is the default delay in PicOS milliseconds after the last transmission before turning off the radio, if the receiver is switched off.

Collision avoidance (LBT) options

The LBT option is compiled in when RADIO_LBT_SENSE_TIME is greater than zero. The default setting is 2. The constant stands for the number of PicOS milliseconds during which the channel should be sensed before a transmission [this is probably too much, we need a sub millisecond setting]. Note that with PXOPTIONS, outgoing packets may individually disable LBT.

The actual procedure is a bit complicated (see Section 23.7.5.5.1 of the swcu117g document by TI) and my understanding of it is far from complete. It involves two independently assessed criteria: RSSI-based and correlation-based. The channel is deemed busy, if any of the two criteria says it is busy.

The RSSI-based procedure consists in monitoring RSSI whenever a new reading is available (I am not sure about the frequency of RSSI updates). If RADIO_LBT_RSSI_NBUSY (default = 4) consecutive RSSI readings are above RADIO_LBT_RSSI_THRESHOLD (default = 70),[1] then the channel is assumed busy. When experimenting, just assume that increasing RADIO_LBT_RSSI_NBUSY or/and increasing RADIO_LBT_RSSI_THRESHOLD will tend to make the collision avoidance scheme less aggressive.

As far as I understand the foggy explanation in the manual, the correlation threshold is related to an attempt to detect the carrier by counting correlation peaks within a time interval determined by RADIO_LBT_CORR_PERIOD (expressed in RAT ticks, where 1 tick = 0.25μs). The default setting of RADIO_LBT_CORR_PERIOD is 512 (128μs). RADIO_LBT_CORR_NINV (default = 3) stands for the number of correlation peaks separated by (the correlation period or less) to reach the first step for the busy decision. The decision will will be affirmative if RADIO_LBT_CORR_BUSY (default = 3) correlation peaks are subsequently detected separated by less than the correlation interval. If RADIO_LBT_CORR_BUSY is 0, then reaching the first step is enough for the busy decision.

If RADIO_LBT_CORR_PERIOD is 0, the correlation criterion is switched off.

If RADIO_LBT_SENSE_TIME is greater than zero (and LBT is not disabled for the packet [with PXOPTIONS]), then the channel assessment is carried out before every transmission attempt, up to RADIO_LBT_MAX_TRIES (default = 16) per packet. When that number is reached, the next transmission is carried out with LBT switched off. The value of 255 or higher stands for infinity.

Similar to CC1100, the driver uses a backoff timer, which is a *utimer*, i.e., a millisecond counter decremented automatically (by the kernel) towards zero. The first thing that the driver does before

1 The RSSI threshold is expressed in the same units as the RSSI values returned in received packets. The range is formally the same as for CC1100. The driver converts the internal signed representation of RSSI into an unsigned one by adding 128.

transmitting a packet is to check whether the backoff timer is zero. If not, the driver delays its next attempt at transmission/retransmission by the current value of the timer.

While waiting for the backoff timer to reach zero, the driver also waits for a generic *attention* event. Such an event is triggered whenever something changes in the driver's environment that may require it to do something. A packet reception (nonempty RX FIFO) is one example of such an event. Generally, the attention event is the most important event awaited by the driver. Whenever the driver is waiting for something (anything at all), it is also waiting for an attention event.

Normally, the backoff timer is decremented towards zero at millisecond rate. Suppose that the driver has a packet to transmit, so it looks at the backoff counter and sees that its value is some $d > 0$. The driver will issue a delay request for d milliseconds along with a wait (*when*) request for the attention event. If nothing special happens in the meantime (no attention event), the driver will wake up exactly at the moment when the backoff timer has reached zero. Then it will have another go at the outgoing packet.

The backoff timer can be reset. For example, it is reset upon a packet reception. It can also be set explicitly by PHYSOPT_CAV request (from the praxis). In any case, when the backoff timer is reset, an attention event is triggered, which (among other things) means that the driver will immediately see the new value of the backoff timer.

When there is a packet to be transmitted and the backoff timer is zero, the driver will initiate the procedure of transmitting the packet. The first step of this procedure is an attempt to grab the channel (the so-called channel status assessment). To this end, the driver tries to switch the chip to the transmit (TX) state and then immediately checks the effective state of the chip. If the effective state is in fact TX, it means that channel access has been granted, so the driver proceeds to transmit the packet. The chip may refuse to be put into the transmit state if it thinks the channel is busy. In such a case, we say that the packet is experiencing a congestion.

If the channel assessment fails, i.e., the chip refuses to enter the transmit state, the driver generates a random number using this formula:

```
bckf_timer = RADIO_LBT_MIN_BACKOFF + rnd () & mask;
```

where RADIO_LBT_MIN_BACKOFF (default setting 2) is the minimum backoff and mask is always a power of two minus 1, i.e., its binary representation consists of a number of ones from the right. The mask size is determined by this constant:

RADIO_LBT_BACKOFF_EXP (default setting 6) which specifies the power of two to be used for the mask (equal to the number of ones + 1). Thus, the default setting translates into 5 ones (2^6–1 = 63). This is the mask to be applied after a failed attempt to grab the channel by the transmitter. With the default setting, the backoff interval can be anywhere between 2 and 2+63=65 milliseconds.

Note that the driver will not re-attempt the transmission for as long as the backoff timer remains nonzero. The backoff timer will be reset after a successful packet reception using the mask described by this constant:

RADIO_LBT_BACKOFF_RX (default setting 3). The idea is that having received a packet we remove at least one of the reasons why a previous transmission attempt may have failed. Thus it may make sense to reconsider the failed transmission under new opportunities. On the other hand, multiple nodes in the neighbourhood may develop the same idea (synchronizing to the end of the same packet), so some randomization may be in order. Note that it isn't absolutely clear whether the new backoff should be statistically shorter or longer than the previous setting (one following a failed channel assessment), but it may make sense to use different ranges in the two cases. The default setting translates into a randomized delay between 2 and 9 milliseconds.

If RADIO_LBT_BACKOFF_EXP is defined as zero, the transmitter behaves aggressively retrying channel access at millisecond intervals.

If RADIO_LBT_BACKOFF_RX is zero, then the backoff timer is never reset after a packet reception. In particular, if it was zero, it stays at zero after the reception.

The backoff timer can be set explicitly by the praxis using this (optional) control call:

```
word bckf;
...
tcv_control (sfd, PHYSOPT_CAV, &bckf);
```

where the value of bckf provides the new setting of the timer in milliseconds. If the last argument of tcv_control is NULL, the driver generates a randomized interval using RADIO_LBT_BACKOFF_EXP for the mask.

Following a packet transmission, the driver sets the backoff timer to: RADIO_LBT_XMIT_SPACE (default setting 2) milliseconds. The idea is to provide a breathing room for the receiver to accommodate the received packet before being fed a new one. If RADIO_LBT_XMIT_SPACE is zero, the backoff timer is not set at the end of a transmission.

Setting/adjusting the base RF frequency

The base frequency (the RF frequency of channel zero) is determined by CC1350_BASEFREQ (default = 868) which simply specifies the frequency in MHz.

Examples

Here is the standard sequence to initialize the driver in combination with the NULL plugin:

```
...
#include "phys_cc1350.h"
#include "plug_null.h"
...
sint sfd;
...
      phys_cc1350 (0, CC1350_MAXPLEN);
      tcv_plug (0, &plug_null);
      sfd = tcv_open (WNONE, 0, 0);
      ...
```

Note that the symbol CC1350_MAXPLEN (defined by the driver) represents the maximum possible length of a radio packet, including the two extra bytes for CRC. That length is 250 bytes (i.e., over four times more than for CC1100).

The rest is basically the same as for CC1100, except that there is no WOR (yet).

Note that following the above initialization, the receiver is OFF and the Network ID is 0. Do this to switch the receiver ON:

```
tcv_control (sfd, PHYSOPT_RXON, NULL);
```

To switch the receiver OFF, use PHYSOPT_RXOFF as the second argument of tcv_control. You can use PHYSOPT_ON and PHYSOPT_OFF instead of PHYSOPT_RXON and PHYSOPT_RXOFF. The "RX" symbols are a legacy of the old scheme whereby the transmitter was switched on and off separately and independently of the receiver. Note that tcv_control formally accepts PHYSOPT_TXON and PHYSOPT_TXOFF as the second argument (for compatibility with the old scheme), but the resultant action is completely void.

The operation of changing the Network ID involves an argument which must be stored in memory, e.g.,

```
...
word nid;
...
      nid = 0xBADD;
      tcv_control (sfd, PHYSOPT_SETSID, &nid);
      ...
```

Sending and receiving packets is straightforward, e.g.,

```
...
state XMIT:
      address packet;
      packet = tcv_wnp (XMIT, sfd, 16);
      ... fill in the packet ...
      tcv_endp (packet);
      ...
```

Note that the length specified as the third argument of tcv_wnp must be an even number between 4 and the maximum packet length (specified as a the second argument to phys_cc1100). It covers the complete packet, including the Network ID and CRC. You are not supposed to fill the last two bytes of the packet, but if you do, whatever you put there will be ignored (if software CRC is in effect, those bytes will be overwritten by the CRC).

Here is sample reception code:

```
...
state RECV:
      address packet;
      sint plength;
      packet = tcv_rnp (RECV, sfd);
      plength = tcv_left (packet);
      ...
```

Note that the packet length (plength) includes the Network ID and two CRC bytes at the end.