15/05/14

CC1100 driver notes

This document applies to the (so-called) new version of the driver which became official on 13/03/07 (with PG130307). The old version, still available as an option, is described in a separate document.

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, including the WOR mode) 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 as long as the output queue is nonempty, but when the queue gets drained, the chip will be powered down again. Thus, if you have several packets to transmit in the RXOFF mode, and they won't be arriving fast enough to keep the queue nonempty, you may consider turning on the receiver for the duration of that sequence. Note that this problem is now (much) less serious than it was in the old driver (probably to the point of complete irrelevance), because (in contrast to the old driver) the new driver doesn't

completely reset the chip on power up.

When the chip is powered down to the WOR (wake on radio) mode (as described farther in this document), there is a non-trivial (configurable) delay (timeout) following any activity (reception or transmission) and preceding the subsequent automatic return to the WOR state. If a reception or transmission happens during that delay, the transition to the WOR state will be postponed again, i.e., the timeout will be reset. Thus, if the chip is in WOR mode, there is some extra amount of time following each transmission during which the chip remains in RX.

Note that there are no TXON/TXOFF physopts. In a sense, the transmitter is assumed to be on for as long as there are packets to transmit. When the queue of packets runs out, and the receiver is off, the chip is powered down (or put into WOR state, if the WOR option is on).

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 praxis).

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_cc1100 when the driver was initialized.

PHYSOPT_RESET

Resets the chip, reloads the registers, reverts the receiver status and all options to defaults, i.e., as they appeared after initializing the driver (with phys_cc1100). Note that the default state of the receiver is off, no WOR.

In the present version of the driver, the chip is never reset internally following the initialization, except in response to PHYSOPT_RESET.

Optional calls

These operations are only available, if the respective options are selected at compilation. To select an option, you have to define some symbol to a nonzero value either in options.sys (locally for the praxis) or in board_options.sys (at the board definition). See *"Static configuration options"* below for the list of option symbols.

PHYSOPT_CAV

Forces an explicit backoff (blocks the transmitter) for the specified number of (PicOS) milliseconds. This call requires RADIO_CAV_SETTABLE to be defined as 1.

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 (according to PATABLE). This call, as well as GETPOWER, requires RADIO_POWER_SETTABLE to be defined as 1. The default setting of transmitter power is 7, i.e., max (note that it was 2 in the old driver). The option is disabled (despite RADIO_POWER_SETTABLE being defined as 1) when RADIO_OPTIONS includes

RADIO_OPTION_PXOPTIONS, as explained below.

PHYSOPT_GETPOWER

Returns the current transmitter power setting (a number between 0 and 7).

PHYSOPT_SETCHANNEL

Sets the RF channel to the value of the lower byte of the specified word (it is a number between 0 and 255). This call, as well as GETCHANNEL, requires RADIO_CHANNEL_SETTABLE to be defined as 1. The default channel number is 0.

PHYSOPT_GETCHANNEL

Returns the current setting of the RF channel.

PHYSOPT_SETRATE

Sets the bit rate to one of four options: 0 – 5000 bps, 1 – 10000 bps (the default), 2 – 38,400 bps, 3 – 200,000 bps. This call, as well as GETRATE, requires RADIO_BITRATE_SETTABLE to be defined as 1. Note that the argument of SETRATE must be a number between 0 and 3, inclusively.

PHYSOPT_GETRATE

Returns the current bit rate option (a number between 0 and 3).

PHYSOPT_RESET (option)

When RADIO_OPTION_REGWRITE is defined (see below), the RESET call accepts an optional argument. When the argument is NULL, the operation behaves exactly as its standard variant (described above). Otherwise, the argument points to an array of bytes interpreted as register assignments (described below). The operation puts new values into the specified registers *without* resetting the chip.

Note that in the old version of the driver, the (optional) argument of RESET was interpreted as a "supplement" table specifying new register settings to be applied at every chip reset. That way the register change was non-volatile until explicitly revoked. It was important, because the old driver would reset the chip internally on many problematic or suspicious events. In the new driver, a register change will not survive a subsequent reset. Note, however, that the new driver never resets the chip except upon an explicit request from the praxis (via argument-less RESET).

PHYSOPT_RXOFF (option)

When RADIO_WOR_MODE is set to 1 (i.e., the WOR option is compiled in), RXOFF accepts an optional argument. If the argument is absent (NULL) or points to a word containing zero, the behaviour of RXOFF is standard. Otherwise (i.e., if the argument points to a nonzero word), the chip will enter a low-power listening mode known as *wake on radio* (see section *"Wake on radio"* for details).

PHYSOPT_ERROR

This operation is only available when RADIO_OPTION_STATS is defined. It manages the performance statistics collected by the driver when that option is on (see below).

PHYSOPT_SETPARAMS

This call is only available if RADIO_WOR_MODE is set to 1. It allows the praxis to change some parameters of the WOR mode (see below).

Compilation options (static configuration)

The header file PicOS/cc1100.h assigns default values to some constants that configure the operation of the driver. All the constants named RADIO_... appearing close to the top of the file (as well as some other, less prominent, constants) can be defined in options.sys (board_options.sys) thus overriding the default definitions. Here is a brief description of their meaning.

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. These bits of RADIO_OPTIONS (represented by symbolic constants named RADIO_OPTION_...) are meaningful and interpreted as follows:

Bit 0 (0x001, RADIO_OPTION_RBKF) causes the driver to insert into word 1 of every outgoing packet (the word immediately following Network ID) the amount of backoff (in PicOS milliseconds) suffered by the packet. Note that the praxis must be aware of that and never use that word as part of the payload.

Bit 1 (0x002, RADIO_OPTION_DIAG) switches on diag messages on various important (or suspicious) events, e.g., power down, reset, packet reception.

Bit 2 (0x004, RADIO_OPTION_STATS) enables PHYSOPT_ERROR, a special physopt request (see above), whereby the praxis can retrieve from the driver six counters pertaining to its performance. This is the format of the control request:

```
word counters [6];
...
tcv_control (sfd, PHYSOPT_ERROR, counters);
```

The specified array is filled with the following word-sized values (in this order):

0. The total number of attempted packet receptions triggered by the recognition of a sync word signalling the beginning of a packet.
1. The total number of successfully received packets. This number is never bigger than the previous number.
2. The total number of packets submitted to the transmitter, i.e., ones that have been extracted from the driver's PHY queue to be transmitted.
3. The total number of packets submitted for transmission that exceeded the limit on the number of transmission attempts (see below). This number is never bigger than the previous number. This number is always zero if RADIO_LBT_MODE (see below) is not 3 (which is the default).
4. The congestion indicator describing the average backoff delay suffered by a transmitted packet (see below).
5. The maximum backoff so far, i.e., the maximum delay suffered by a packet.

Cumulative counters wrap around at 64K (65536). Counters 0 and 1, as well as 2 and 3, are coupled in the sense that when the first one wraps around the maximum (and is zeroed), the second one is zeroed as well.

The congestion indicator is calculated as an exponential moving average of the backoff delay over all transmitted (or dropped) packets. The formula is

$$C_n = C_{n-1} \times 0.75 + d_n \times 0.25 ,$$

where C_n is the new value of the indicator, after accommodating the delay (d_n) of the current packet ,

C_{n-1} was the previous value (calculated for the previous packet), and d_n is the delay (in PicOS milliseconds) suffered by the current packet.

All counters can be zeroed by providing a NULL second argument to tcv_control, i.e.,

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

Bit 3 (0x008, RADIO_OPTION_CCHIP) activates code to safely check for the presence of CC1100 module when the driver is initialized (with phys_cc1100). This is to prevent indefinite hangups in tests designed for possibly faulty boards. Normally an attempt to start the driver on a board without a CC1100 chip (or with a defective chip) may hang the system. Note that this option is void on CC430 (where the radio is integrated with the CPU).

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 5 (0x020, RADIO_OPTION_WORPARAMS) selects the test mode of WOR. Its sole purpose at present is to extend the list of options settable with the SETPARAMS physopt – to make it possible to change some parameters of WOR from the praxis. The details are described in *"Wake on radio"*.

Bit 6 (0x040, RADIO_OPTION_REGWRITE) extends the function of PHYSOPT_RESET (see above). With this option, RESET accepts an optional argument pointing to a byte array and providing new values to be stored in indicated registers. Note that the chip is not reset by this variant of the operation: only the selected registers are modified according to the table. This makes it possible for the praxis to reconfigure the chip in a practically unlimited way (including ways that may potentially render the chip unresponsive and even hang the driver). The array consists of two bytes per register setting (the register number and its new value) and terminates with byte 255 (0xff). For example this sequence:

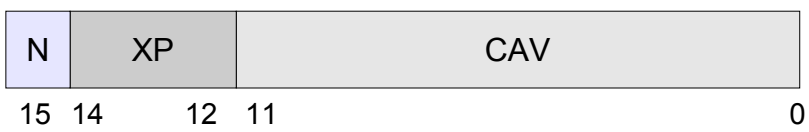
```
byte subst [] = { 0x3E, 0xC4, 1C, 0x48, 255 };
...
tcv_control (sfd, PHYSOPT_RESET, (address)subst);
```

sets the first entry in PATABLE (register 0x3E) to 0xC4 and register AGCCTRL1 (specifying LBT thresholds – see below) to 0x48.

Note that a RESET request with NULL argument will reset the chip and remove any register modifications previously made by the praxis.

Bit 7 (0x080, RADIO_OPTION_ENTROPY) enables the code for initializing the node's entropy (affecting random number generation) from RSSI when the driver is initialized (by phys_cc1100). This is only effective if the global system options select entropy collection, i.e., the compilation constant ENTROPY_COLLECTION is defined as nonzero. In such a case, when the driver is initialized, it will take 8 samples of RSSI at 1 msec intervals and incorporate the least significant 4 bits of each sample into a 32-bit initial *entropy* value. Note that regardless of this bit, *entropy* will be updated from RSSI on every packet reception (assuming that ENTROPY_COLLECTION is nonzero).

Bit 8 (0x100, RADIO_OPTION_PXOPTIONS) enables per-packet parametrization of transmission power and collision avoidance vectors. 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_CRC_MODE (default setting 0) selects the way of calculating the CRC for a packet, which can be one of:

0	built-in hardware calculation; this is the recommended default setting
1	built-in CC2400-compatible CRC
2	like 0 + AUTOFLUSH (do not use – see below)
3	like 1 + AUTOFLUSH (do not use)
4	software CRC (our own software implementation of ISO 3309 CRC)

At first sight, the AUTOFLUSH option appears attractive as (theoretically) it does not wake up the driver until a packet has been received correctly (its hardware CRC is OK). This means that it delegates most of the reception processing to hardware, thus minimizing the overhead in the driver. However, as it turns out, the aggregate reception event is presented in an inconsistent way, which makes it difficult to organize the driver's activities around it (and, for example, avoid hangups caused by lost events).

Despite its present buggy status, the option has not been removed from the driver, as newer versions of the chip may fix the problem (providing for a more foolproof way of triggering reception events).

As a side note, some of my experiments would (surprisingly) suggest that software CRC gives a slightly better performance (better packet delivery fraction at a given distance) than hardware CRC. I would not bet on it too heavily, but one possible reason why hardware CRC might result in worse reception is the electromagnetic interference caused by the calculation being carried out in parallel with the reception (note that software CRC is calculated *after* the complete packet has been received). I have observed deteriorating performance of the chip under conditions of heavy CPU activity, which can only be attributed to increased electromagnetic noise caused by the CPU. Perhaps a similar kind of noise is generated by the chip itself when it calculates the CRC online. To support this hypothesis, I have seen a line in one of TI application notes stating that polling the chip (in a loop, instead of awaiting an interrupt) reduces RF sensitivity. That statement wasn't elaborated upon, but it strongly suggests that the amount of stress on the chip during reception should be reduced as much as possible.

RADIO_CRC_TRANSPARENT (default setting 0) indicates whether packets with bad CRC should be receivable, i.e., should be accepted by the driver and queued for reception by the application. If the flag

is nonzero, than a packet with bad CRC (formally qualifying for reception otherwise) will be received. The user program can examine the most significant byte of the link quality indicator (LQI) byte which is 1 when the CRC was correct and 0 otherwise. The flag applies to hardware and software CRC calculation. Note that it doesn't affect the CRC for output packets.

RADIO_DEFAULT_POWER (default setting 7) selects the default transmission power setting. This is the number of PATABLE entry (between 0 and 7 inclusively) to be used as the initial power level immediately after the chip's initialization. This is also the power level to be assumed when the argument of a PHYSOPT_SETPOWER request is NULL.

RADIO_POWER_SETTABLE (default setting 0) indicates whether the *physopt* requests SETPOWER and GETPOWER should be available. Note that they are disabled by default, in which case, the initial and immutable power setting is determined by RADIO_DEFAULT_POWER.

RADIO_DEFAULT_CHANNEL (default setting 0) selects the default channel. Legitimate values are between 0 and 255.

RADIO_CHANNEL_SETTABLE (default setting 0) indicates whether the *physopt* requests SETCHANNEL and GETCHANNEL should be available. Note that they are disabled by default, in which case, the immutable channel number is determined by RADIO_DEFAULT_CHANNEL.

RADIO_DEFAULT_BITRATE (default setting 10000) selects the default bit rate roughly in bits per seconds. Only these four values are legitimate: 5000, 10000, 38400, and 200000.

RADIO_DEFAULT_BITRATE can be set to a rate (e.g., 38400) as well as to the ordinal (index) of that rate (e.g., 2). For example, the settings 38400 and 2 are equivalent.

RADIO_BITRATE_SETTABLE (default setting 0) indicates whether the *physopt* requests SETRATE and GETRATE should be available. Note that they are disabled by default, in which case, the immutable bit rate is determined by RADIO_DEFAULT_BITRATE.

RADIO_CAV_SETTABLE (default setting 0) indicates whether the praxis wants to use the CAV *physopt* request to directly set the backoff timer (see below). Note that the option is disabled by default, i.e., the request is not available.

RADIO_WOR_MODE (default setting 0) indicates whether the wake on radio (WOR) mode option should be compiled in. The WOR mode is described in detail in section *"Wake on radio"*.

RADIO_SYSTEM_IDENT (default setting 0xAB35) is the layout of the 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 frequency band 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_RECALIBRATE (default setting 0) tells whether the the module's frequency synthesizer should be periodically recalibrated if nothing happens in the meantime that would trigger automatic recalibration. Normally, when RADIO_CALIBRATE is zero, the synthesizer is recalibrated automatically in these circumstances:

1. before switching on RX (including after every packet reception)
2. before switching on TX, but only if the receiver (RX) has been inactive (OFF); this includes the case of transmitting in the OFF mode, or powering the chip ON (switching the receiver ON) after OFF

Theoretically, if no packet is received for a long time while the chip remains in the RX state, the synthesizer may lose its calibration. The praxis can always trigger recalibration explicitly by powering

the module down (going OFF) and then back ON, but the driver can also do it periodically (and more efficiently) if RADIO_RECALIBRATE is nonzero.

If nonzero, RADIO_RECALIBRATE tells the number of seconds after which the synthesizer will be forcibly recalibrated if the module remains in RX and there has been no reception. The maximum duration of the recalibration interval is 63 seconds.

If RADIO_RECALIBRATE is nonzero, then automatic recalibration is also forced after every packet transmission (including the situation when the receiver is ON). This is accomplished by switching the module to IDLE and then back to RX after every transmission. Note that the driver doesn't use the explicit calibration strobe (SCAL) which is messy and probably not much more efficient than going through the IDLE state.

Note: for compatibility with the old driver, I have temporarily reverted the official default settings of RADIO_POWER_SETTABLE, RADIO_CAV_SETTABLE, RADIO_BITRATE_SETTABLE, and RADIO_CHANNEL_SETTABLE to 1.

Collision avoidance (LBT) options

RADIO_LBT_MIN_BACKOFF (default setting 2). This is the minimum value of a backoff (in PicOS milliseconds) generated, e.g., when the driver fails to grab the channel for a packet transmission. To understand the interpretation of this and other backoff (LBT) related options, you need some understanding of the channel access protocol for transmission, which works as follows.

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 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.

One constant describing how congestion should be handled is:

RADIO_LBT_MODE (default setting 3). The legitimate values are 0, 1, 2, and 3. When the constant is set to 0, congestion is simply ignored. In other words, the chip is set up in such a way that it will never refuse to be put into the transmit state. This is another way of saying that all transmissions are blind and the chip never checks whether they do not interfere with other RF activities in the neighbourhood, including a situation when the chip happens to be receiving a packet at the very moment. Formally, this mode of operation is selected by setting CCA_MODE (bits 4-5 of the chip's MCSM1 register) to 0, which disables the LBT (listen before transmit) feature altogether.

By setting RADIO_LBT_MODE to 1 you choose a slightly less aggressive behaviour (corresponding to CCA_MODE = 2). While the chip will still not care about the general interference level in the neighbourhood, it will refuse to be put into the transmit state, if a packet is currently being received by the chip. Formally, that means that the *sync word* of an incoming packet has been recognized by the receiver and the RX FIFO is being filled.

If RADIO_LBT_MODE is 2 (corresponding to CCA_MODE = 3), then, additionally, the chip will be monitoring the RSS level of the received signal (regardless of the formal reception status) and refuse to be put into the transmit state if that level is above a certain threshold. That threshold (or rather two thresholds) are described by these two constants (see PicOS/cc1100.h):

RADIO_CC_THRESHOLD	(default = 1, legitimate values 0 – 15)
RADIO_CC_THRESHOLD_REL	(default = 1, legitimate values 0 – 3)

The value zero of a threshold constant disables the respective condition. RADIO_CC_THRESHOLD applies to the absolute RSSI level. Increasing (nonzero) values correspond to higher absolute levels, so 1 is the finest setting (the most sensitive active threshold). The second threshold constant (RADIO_CC_THRESHOLD_REL) refers to a relative threshold, which basically means an increase in what the chip considers to be the (possibly intermittent) level of background noise. Again, 0 switches off this condition (the present default setting), while 1, 2, and 3 correspond to increasing thresholds. The rather convoluted way those values translate into dBm is explained in the chip's documentation. Note that the values of the above two constants are only relevant if RADIO_LBT_MODE is set to 2 (they are completely ignored otherwise).

For tests, you may either change the values of the above constants and recompile the praxis, or, alternatively, you can modify (dynamically, i.e., using the respective variant of PHYSOPT_RESET – see above) the contents of register AGCCTRL1 (number 1C). That register stores the values of the two constants in a somewhat obfuscated way (because of the two's complement encoding). The second hex digit corresponds to RADIO_CC_THRESHOLD offset by 0x08, which means that 0x08 translates into CC_THRESHOLD = 0 (off), 0x09 corresponds to 1, ..., 0x0F corresponds to 7, 0x00 corresponds to 8, ..., 0x07 corresponds to 15. The first digit is 0x40 or'red with the absolute value of RADIO_CC_THRESHOLD_REL, e.g., 0x60 corresponds to RADIO_CC_THRESHOLD_REL = 2.

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 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. This option is not recommended. It was used in conjunction with 200 kbps transmission rate for high bandwidth transmission of real-time data in the heart datalogger project at SFU.

If RADIO_LBT_BACKOFF_RX is defined as 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 (there is no forced after-reception backoff). Note that, intuitively, after reception backoff is important. This is because several nodes in the neighbourhood may want to respond to the same received packet, which means that receptions are not unlikely to trigger contention.

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.

A successful channel assessment is always followed by a transmission. In a sense it does automatically start a transmission, because following it, the chip immediately begins to transmit the packet preamble. Formally, starting with a successful assessment, the entire transmission involves no delays (that would allow other threads to interleave with the driver while it happens), as it boils down to writing the packet to the TX FIFO and immediately removing it from the PHY queue (one exception is transmitting a long preamble to wake up a WOR receiver – see below). Then the driver waits for the TX status to be removed from the chip, which will indicate the physical end of the transmission. This is accomplished in an interrupt-free fashion: the driver estimates the transmission time (based on the packet length) – in fact underestimating it slightly – and then, at the end of the waiting time, polls the chip at millisecond intervals for the actual status change. Following the status change, 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.

Note that with RADIO_LBT_MODE = 1 or 2 the driver will keep trying to acquire the channel (backing off at every failed attempt) until it succeeds, using the same preset and fixed LBT sensitivity parameters (for case 2) at each attempt (determined by RADIO_CC_THRESHOLD and

RADIO_CC_THRESHOLD_REL). When RADIO_LBT_MODE is 3, the driver will limit the number of channel acquisition attempts to 8 reducing the sensitivity of the LBT thresholds after every failed attempt. For the first 5 attempts, it will use CCA_MODE = 3 (full LBT), for the subsequent two attempts it will set CCA_MODE = 2 (yielding to receptions only), and for the last (eighth) attempt it will switch LBT off completely (setting CCA_MODE to zero). Thus, the last attempt must necessarily succeed (formally speaking).

The specific sensitivity settings used for the first 5 attempts when RADIO_LBT_MODE = 3 are determined by this table:

```
0x59, 0x5A, 0x6B, 0x7D, 0x4F
```

which lists the values consecutively stored into AGCCTRL1. Note that the first entry corresponds to RADIO_CC_THRESHOLD = 1 and RADIO_CC_THRESHOLD_REL = 1, i.e., the most sensitive setting, while the last one has the largest threshold value for the absolute signal (7) with the relative assessment completely switched off.

Setting/adjusting the base RF frequency

The base frequency (the RF frequency of channel zero) is determined by three constants defined (conditionally) in PicOS/cc1100.h:

```
#ifndef CC1100_FREQ_FREQ2_VALUE
      #define CC1100_FREQ_FREQ2_VALUE 0x22
#endif

#ifndef CC1100_FREQ_FREQ1_VALUE
      #define CC1100_FREQ_FREQ1_VALUE 0xC4
#endif

#ifndef CC1100_FREQ_FREQ0_VALUE
      #define CC1100_FREQ_FREQ0_VALUE 0xEC
#endif
```

The conditions mean that the definitions can be easily overridden by definitions in options.sys (on a per-praxis basis), or board_options.sys (on a per-board basis) should a need arise. For example, in some prototype boards, the crystal (oscillator) frequency may deviate from nominal; in such cases, the deviation can be compensated by adjusting the above constants.

To transform the constant settings into frequency, you should combine the three values into a single three-byte hexadecimal number with FREQ2 representing the most significant byte. For the default settings (see above), the number is 0x22C4EC (2,278,636 decimal). Then you have to multiply this number by the crystal frequency (26,000,000) and divide by 2^{16}, i.e., 65,536. When you do that with the above number, you will get 903,999,877 Hz (i.e., almost exactly) 904 MHz. The reverse procedure is equally straightforward. Suppose you would like to adjust the base frequency by +100 kHz, i.e., 1/2 of the channel width. So your base frequency is 904.1 MHz, i.e., 904,100,000 Hz. You multiply this number by 65,536 and then divide it by 26,000,000 which yields 2,278,888. In hexadecimal this number is 0x22C5E8. So the first constant does not change, and the remaining two are:

```
#define     CC1100_FREQ_FREQ1_VALUE 0xC5
#define     CC1100_FREQ_FREQ0_VALUE 0xE8
```

The above definitions should be put, e.g., into options.sys of the praxis. Note that incrementing (or decrementing) FREQ0 by 1 affects the base frequency by about 396.73 Hz (assuming 26 MHz crystal). One unit of FREQ1 corresponds to 101.56 kHz. One unit of FREQ2 translates (exactly) into 26 MHz (i.e., the crystal frequency).

Wake on radio

The WOR mode is an optional feature of the driver enabled by setting RADIO_WOR_MODE to 1. When compiled in, it allows you to put the chip into a special state whereby it mostly sleeps using the minimum amount of power, but periodically opens the receiver for a short while to check the channel for activity. The kind of activity to trigger a wakeup (and a packet reception attempt) is configurable to some extent.

All the discussion of the CC1100 WOR feature in the official technical notes, as well as the (naive) code examples listed there, address essentially the same scenario where extremely short (one byte) packets are transmitted at a high bit rate, so the entire packet can be received within the tiny listening slot of the rigid WOR cycle. For some reason, the authors assume that the primary concern of the user is to keep the cycles synchronized at the communicating nodes, so they go to great pains to show how to accomplish that goal while circumventing three or four bugs in the chip design, which render the task almost impossible. The underlying premise of those case studies is to never exit the WOR mode, assuming that the high regularity of sleep/wakeup cycles is of utmost and primary importance.

This mode of operation is clearly quite different from our needs and thus completely useless to us. We would like to be able to transmit spontaneously, possibly after a long period of inactivity, packets of non-trivial length, without expecting the receiver to know the exact millisecond of the packet's commencement. In other words, the role of the WOR mechanism in our setting would be to wake up nodes that have been dormant for (possibly) an extended amount of time (e.g., measured in hours), then switch them to a normal RX mode to exchange one or more packets and, at the end of the processing cycle, put the nodes (automatically) back to WOR sleep.

In our driver, the WOR mode is logically viewed as a special case of switching the receiver off and powering down the chip. Normally, without WOR, a PHYSOPT_RXOFF call switches off the receiver and, if the transmit queue is empty, puts the chip to SLEEP in which mode it uses the absolute minimum of battery power and, unfortunately, remains completely unresponsive to RF events. If there are packets to be transmitted at the moment the RXOFF request is issued, the operation of physically powering down the chip is postponed until the transmit queue has been completely drained.

With WOR configured in, PHYSOPT_RXOFF accepts an optional argument. If that argument points to a word containing nonzero, the operation will put the chip into WOR mode. Similar to entering the SLEEP state, that action will be postponed until the transmit queue becomes empty.

While in WOR mode, the chip will sleep for some time (effectively consuming the same amount of power as in the completely dumb SLEEP state), then wake up to switch the receiver on for a short while, then go to sleep again. That cycle is highly regular. Its duration is determined by the symbolic constant WOR_EVT0_TIME whose default value is 65535 (which also happens to be the maximum). This constant (as well as the remaining constants discussed in this section) can be redefined, e.g., in options.sys. The value is transformed into seconds by this formula:

$$t_{EVT0} = \frac{750 \times \text{WOR\_EVT0\_TIME}}{26{,}000{,}000}$$

Thus, for the default (and maximum) setting, the cycle length is ca. 1.89 sec. [N.B.: should we decide that it is worthwhile, the cycle length can be increased by resorting to some (messy) parameters which I am not discussing here.] Note that the denominator in the above formula is the frequency of the high-speed (RF) crystal.[1]

1 On CC430, the WOR clock is simply ACLK, so in the formula the denominator becomes 32768 and the numerator loses the 750 factor. The ratio w.r.t. the CC110x variant is 0.945 and the maximum setting of 65535 translates into 2 seconds. To bring it to the same value as for CC110x, the maximum setting is assumed to be 61944 (0xF1F8).

Another parameter of the WOR mode is the duration of the reception period within the cycle. It is described by the constant WOR_RX_TIME whose default value is 6. The range of values is from 0 to 6, and they approximately translate into a percentage of the total cycle, such that 0 yields 12.5% and each subsequent value halves the time, e.g., 5 translates into 0.391% and 6 (the smallest fraction) into 0.195%. For the default (maximum) setting of WOR_EVT0_TIME, these numbers yield 7.4 msec and 3.7 msec, respectively.

Having entered the RX state within a WOR cycle, the chip will first wait for the minimum amount of time required to obtain the first valid RSSI reading and compare that reading against a threshold. The details of how those thresholds exactly work are a bit complicated (and I do not understand everything); suffice it to say that the threshold level is represented by a simple value from 0 to 15 (see section *"Collision avoidance"*), with 0 meaning switched off (i.e., zero threshold or unused), 1 being the minimum actual threshold, and 15 being the maximum. If the RSSI reading is below the threshold, the chip immediately gets back to sleep ignoring the remainder of the RX period (thus additionally conserving power). Otherwise, the chip continues monitoring the channel until the end of the RX period and, at the end of that period, carries out the so-called preamble quality assessment. If that assessment turns out positive, meaning that the chip is sensing something that looks like a packet preamble, it will trigger an interrupt to wake up the driver. The constant describing the RSSI threshold setting is named WOR_RSSI_THR and its default value is 2.

The required quality of the assessed preamble is described by another constant, WOR_PQ_THR, whose range is 1 through 7 with the default setting of 5. The preamble quality estimator maintains an internal counter which is incremented by one each time a bit is received that is different from the previous bit, and decremented by 4 each time the bits are identical. The assessment is positive when the counter reaches 4 * WOR_PQ_THR.

Yet another constant parameterizing WOR is WOR_EVT1_TIME with the range 0 through 7 and the default setting of 0. Its meaning is somewhat nebulous, except that a larger value supposedly wastes more time/energy in the cycle and a smaller value increases the chance that the chip may not be fully ready when the RX slice formally commences. I haven't noticed any problems with 0 (or any impact of that parameter, for that matter), but my tests so far have been hardly exhaustive.

Two more parameters refer to the operation of the driver, rather than the behaviour of the chip. They are:

RADIO_WOR_IDLE_TIMEOUT (default setting 3072, i.e, 3 seconds) giving the number of PicOS milliseconds for the inactivity timeout. In response to a positive preamble assessment, the driver will leave the WOR mode and begin listening for packets (in a regular active mode with the receiver switched on). While in that mode, the driver also waits on a delay whose first argument is RADIO_WOR_IDLE_TIMEOUT. Any activity (reception or transmission) will reset that wait. If nothing happens for its entire duration, the chip is put back into the WOR sleep state.

RADIO_WOR_PREAMBLE_TIME (default setting 2048, i.e., 2 seconds) giving the duration of a waking preamble to be sent by the transmitter to wake up a node sleeping in WOR. Such a long preamble will be generated for any packet marked as *urgent* (in VNETI parlance). This only happens if the WOR option is compiled in.

The idea is quite straightforward. When the driver sees an urgent outgoing packet it will precede its transmission with an extra long preamble. That is accomplished by delaying the proper transmission following a successful channel assessment (see above). The preamble has enough duration to wake up the receiver (note that it is longer than the complete WOR cycle) while providing enough spare length for the packet's reception when the chip has been switched into normal RX mode by the driver in response to the positive preamble assessment event.

Note that long waking preambles are (somewhat) incompatible with LBT mode 3 (RADIO_LBT_MODE = 3), not to mention modes 0 and 1. This is is because two or more nodes trying to start long preambles at roughly the same time will always interfere (such that one of the two transmissions will be overwritten by the preamble of the other). The multiple attempts to back off (of mode 3) will be ineffective against a sustained channel occupancy lasting seconds, because they take much less than that and the last of them is always allowed. Perhaps I should make sure that the waking preambles are always issued with LBT mode 2, even if mode 3 happens to be the compiled option.

With WOR compiled in, the driver provides an extra physopt call, PHYSOPT_SETPARAMS, whereby the praxis can redefine the two time parameters. The argument is a two-word array with the first word providing the new value of RADIO_WOR_IDLE_TIMEOUT and the second word giving the new setting of RADIO_WOR_PREAMBLE_TIME. When the chip is reset (e.g., PHYSOPT_RESET), the values will be reverted to defaults.

When RADIO_OPTION_WORPARAMS is set, the argument to PHYSOPT_SETPARAMS is extended by 5 additional bytes immediately following the original two-word array. The meaning of those bytes is as follows:

0. WOR_EVT0_TIME	(the more significant byte only)
1. WOR_RX_TIME	(0 – 6)
2. WOR_PQ_THR	(1 – 7)
3. WOR_RSSI_THR	(0 – 15)
4. WOR_EVT1_TIME	(0 – 7)

Examples

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

```
...
#include "phys_cc1100.h"
#include "plug_null.h"
...
sint sfd;
...
      phys_cc1100 (0, CC1100_MAXPLEN);
      tcv_plug (0, &plug_null);
      sfd = tcv_open (WNONE, 0, 0);
      ...
```

Note that the symbol CC1100_MAXPLEN (defined by the driver) represents the maximum possible length of a radio packet, including the two CRC bytes. That length is either 62 (when hardware CRC is selected) or 60 (with software CRC). This is because with software CRC the CRC bytes are sent as part of the packet payload (and are stored in the FIFO), whereas the hardware variant of CRC is treated separately (using no FIFO space).

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.

If WOR has been configured into the driver, you can put the radio into the WOR sleep mode with this operation:

```
...
word par;
...
      par = 1;
      tcv_control (sfd, PHYSOPT_OFF, &par);
      ...
```

You can exit the mode explicitly in one of three ways:

1. resetting the module: tcv_control (PHYSOPT_RESET, NULL), 2)
2. executing tcv_control (sfd, PHYSOPT_OFF, NULL), or with the argument equal 0
3. executing tcv_control (sfd, PHYSOPT_ON, NULL)

Here is how to send a packet with a long preamble intended to wake up a node waiting on WOR:

```
...
state XWAKE:
      address packet;
      packet = tcv_wnpu (XWAKE, sfd, 4);
      packet [1] = 0;
```

```
            tcv_endp (packet);
            ...
```

This is an “urgent” packet of the smallest possible length. Its sole purpose is to wake up the receiver (which will be accomplished by the extra long preamble).