10/07/30, 10/11/07, 12/02/01

CC1100 driver notes

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

RADIO_OPTIONS (with the default value of 0) is a configuration of bit flags, which are mostly used to select various levels of diagnostics (for debugging or performance evaluation). Eight least significant bits of the constant's value are meaningful and interpreted as follows:

Bit 0 (0x01) switches on diag messages indicating abnormal conditions. Those messages that indicate serious problems (that normally should not occur for as long as the chip is healthy) end with two exclamation marks.

Bit 1 (0x02) switches on diag messages on various important (albeit normal) events, e.g., power down, reset, packet reception.

Bit 2 (0x04) enables **PHYSOPT_ERROR**, a special control request, whereby the praxis can retrieve from the driver 6 counters reflecting 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 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) and were dropped. 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 (including packets dropped due to congestion, when **RADIO_LBT_MODE** is 3).

Accumulative 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

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

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 (0x08) activates code to check for the presence of CC1100 chip on system reset. 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 chip (or with a defective chip) will hang the system.

Bit 4 (0x10) causes the special guard process to be activated and run in parallel with the driver. The role of the guard process is to detect hangups of the driver resulting from a faulty CC1100 chip. Normally, the guard process is not required.

Bit 5 (0x20) switches on extra consistency checks (along the same line as the guard process). Such checks are not normally required, but they may be occasionally useful for diagnosing broken chips.

Bit 6 (0x40) extends the semantics of **PHYSOPT_RESET** (this is a control request resetting the chip) by using its argument as a pointer to the substitution table providing new (replacement) values for selected registers of the chip. 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 hang the driver). The substitution table consists of two bytes per register setting (the register number + its 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 the praxis should not alter the substitution table after submitting it to the driver. This is because the same table will be used on every subsequent reset (which may occur spontaneously for quite legitimate internal reasons). Alternatively, if the praxis knows what it is doing, it may just modify

some values in the substitution table and trigger a reset (e.g., by switching the transmitter/receiver off and back on) to effect the new register settings.

By specifying a NULL substitution table pointer, you remove the last substitution table from the driver's view. Also, each new substitution table completely revokes the previous one. Those registers that are not mentioned in the table receive standard settings.

Bit 7 (0x80) enables the code for initializing the node's entropy (affecting random number generation) from RSSI upon system reset. This is only effective if the global system options select entropy collection, i.e., the 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).

`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 should be the recommended setting (I am not saying that it *is* the recommended setting, because there is still some controversy as to whether the old software CRC calculation is not better)

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).

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

Note: one possible reason why hardware CRC might result in worse reception is the electromagnetic interference cause by the calculation being carried out in parallel with the reception. I have observed deteriorating performance of the RF 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 on line.

`RADIO_DEFAULT_POWER` (default setting 2) 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 to a **PHYSOPT_SETPOWER** control request is NULL.

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

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_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 constants, you need some understanding of the channel access protocol for transmission, which is outlined below.

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 present 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 perform an action. A packet reception (nonempty RX FIFO) is one example of such an event. Generally, the attention event is the most important event awaiting 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 at exactly the moment when the backoff timer has reached zero.

The backoff timer can be reset. For example, it is reset upon a packet reception. It can also be set explicitly by a **PHYSOPT_CAV** control call (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 contains 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 assessment). To this end, the driver tries to switch the chip to “transmit” state and then immediately checks the effective state of the chip. If the effective state is in fact “transmit”, 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 some other RF activities in the neighbourhood, including a situation when the chip happens to be receiving a packet. 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 feature altogether.

We start by describing the first two nontrivial modes, i.e., 1 and 2. The description of mode 3 comes a bit later.

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 *start word* of a 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 = 6, legitimate values 0 – 15)
RADIO_CC_THRESHOLD_REL	(default = 0, 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 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) current level of background noise. Again, 0 switches off this condition (the present default setting), while 1, 2, and 3 correspond to increasing thresholds. The 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 a substitution table – 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 (certainly, it makes no sense with **RADIO_LBT_RETRY_LIMIT** or **RADIO_LBT_RETRY_STAGED** – see below). 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 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. Formally, such a 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. 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 defined as 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 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.

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 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.

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 904.1MHz, 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).