CC1101 'Random' RX FIFO Overflow

Resolved

Jeremy Langford

Hello TI Community,

Sorry to get multiple threads going, but I have a few questions that I would like answered and I didn't want to confuse my other thread.

I am wondering if there is anything fundamentally wrong with waiting until the a packet has been received with CRC Good and CRC auto-flush enabled. I have been experimenting with writing code for a non-ti micro-controller to receive data from the cc1101 dev kit acting as a transmitter.

What normally happens is that I can receive data fine, but after some arbitrary number of successful successive receives, normally in the thousands, my radio will fall into RX-FIFO-Overflow. I would not expect this to happen since I automatically discard any packets that are received with a CRC failure, and I am transmitting a packet once every 25,50, or 100 ms.

I have been able to verify via the oscilloscope, that from the time that I receive the CRC good signal from the radio, and I enter my ISR and finish the ISR, takes around 300 us, which should be a short enough of a time span so that I don't have any overlapping packets.

I was wondering if you guys could point me in the correct direction and perhaps ask me the questions that I should be asking my self that I haven't thought of.

I am looking forward to your responses!

Jeremy

Jeremy Langford

Jeremy Langford

Just an update here, I was able to get in and grab my register configurations, so that should help quite a bit.

Could it be possible that I am getting RX Timeout with these configurations then going into idle mode, which would cause the fifo to overflow at that point? I have the TI Smart RF studio transmitting automatically at a given period.

```
volatile uint8 rfSettings[] = {
    0x29,  // IOCFG2              GDO2 Output Pin Configuration
    0x2E,  // IOCFG1              GDO1 Output Pin Configuration
    0x07,  // IOCFG0              GDO0 Output Pin Configuration, Asserts when packet re
    0xFF,  // FIFOTHR
    0xD3,  // SYNC1               Sync Word, High Byte
    0x91,  // SYNC0               Sync Word, Low Byte
    0xFF,  // PKTLEN              Packet Length
    0x0C,  // PKTCTRL1            Packet Automation Control :added autoflush
    0x45,  // PKTCTRL0            Packet Automation Control
    0x00,  // ADDR                Device Address
    0x1E,  // CHANNR              Channel Number
    0x0c,  // FSCTRL1             Frequency Synthesizer Control
    0x00,  // FSCTRL0             Frequency Synthesizer Control
    0x22,  // FREQ2               Frequency Control Word, High Byte
    0xBB,  // FREQ1               Frequency Control Word, Middle Byte
    0x0C,  // FREQ0               Frequency Control Word, Low Byte
    0x0E,  // MDMCFG4             Modem Configuration //changed to 500kb
    0x3B,  // MDMCFG3             Modem Configuration
    0x03,  // MDMCFG2             Modem Configuration
    0x23,  // MDMCFG1             Modem Configuration
    0xFF,  // MDMCFG0             Modem Configuration
    0x62,  // DEVIATN             Modem Deviation Setting
    0x07,  // MCSM2               Main Radio Control State Machine Configuration
    0x3c,  // MCSM1               Main Radio Control State Machine Configuration chanhe
    0x18,  // MCSM0               Main Radio Control State Machine Configuration
    0x1D,  // FOCCFG              Frequency Offset Compensation Configuration
    0x6c,  // BSCFG               Bit Synchronization Configuration
    0xC7,  // AGCCTRL2            AGC Control
    0x00,  // AGCCTRL1            AGC Control
    0xB2,  // AGCCTRL0            AGC Control
    0x87,  // WOREVT1             High Byte Event0 Timeout
    0x6B,  // WOREVT0             Low Byte Event0 Timeout
    0xFB,  // WORCTRL             Wake On Radio Control
    0xB6,  // FREND1              Front End RX Configuration
    0x10,  // FREND0              Front End TX Configuration
    0xEA,  // FSCAL3              Frequency Synthesizer Calibration
    0x2A,  // FSCAL2              Frequency Synthesizer Calibration
    0x00,  // FSCAL1              Frequency Synthesizer Calibration
    0x1F,  // FSCAL0              Frequency Synthesizer Calibration
    0x41,  // RCCTRL1             RC Oscillator Configuration
    0x00,  // RCCTRL0             RC Oscillator Configuration
    0x59,  // FSTEST              Frequency Synthesizer Calibration Control
    0x7F,  // PTEST               Production Test
    0x3F,  // AGCTEST             AGC Test
    0x88,  // TEST2               Various Test Settings
    0x31,  // TEST1               Various Test Settings
    0x09  // TEST0               Various Test Settings
};
```

Oleg Pushkarev

In reply to Jeremy Langford:

Hi, Jeremy,

May be you could show here your Reciever code?

Below is well working code. It 's from TI actually. As you see there is RX FIFO overflow managing. Without this portion of code my RX stops in 2...24 hours.

```
// infinite loop
  while(1)
  {
    // wait for packet received interrupt
    if(packetSemaphore == ISR_ACTION_REQUIRED)
    {
      cc11xLSpiReadReg(CC110L_RXBYTES,&rxBytesVerify,1);

      do
      {
        rxBytes = rxBytesVerify;
        cc11xLSpiReadReg(CC110L_RXBYTES,&rxBytesVerify,1);
      }
      while(rxBytes != rxBytesVerify);


      // Check that we have bytes in FIFO
      if(rxBytes != 0) {

        // Read MARCSTATE to check for RX FIFO error
        cc11xLSpiReadReg(CC110L_MARCSTATE, &marcState, 1);

        // Mask out MARCSTATE bits and check if we have a RX FIFO ERROR (0x11)
        if((marcState & 0x1F) == RXFIFO_OVERFLOW) {

          // Flush RX FIFO
          trxSpiCmdStrobe(CC110L_SFRX);
        } else {

          cc11xLSpiReadRxFifo(rxBuffer,(rxBytes));

          // check CRC ok (CRC_OK: bit7 in second status byte)
          if(rxBuffer[rxBytes-1] & 0x80)
          {
            // toggle led
            halLedToggle(LED1);
            // update packet counter
            packetCounter++;
          }
        }
      }
      // reset packet semaphore
      packetSemaphore = ISR_IDLE;

      // set radio back in RX
      trxSpiCmdStrobe(CC110L_SRX);

    }
  }
```

Oleg

Jeremy Langford

In reply to Oleg Pushkarev:

Oleg, Thanks for the advice, I changed my register configuration to go from RX to Idle after every read, and I then put it back into RX manually. It seems to have helped quite a bit, but it still seems to have the same fundamental problem.

Out of curiosity, what kind of data rates are you shooting for in your device? I am trying to get as close to 500 kb/s as possible. Every now and then, my device will still stop working on me, but I noticed that in smart RF studio, that as I increased the period of transmission to my receiver, that the read failure seemed to take longer to happen.

I have a hunch, that the TI dev kit get's its packet TX timings from the PC that it is connected to, and that when you ask for 5 ms, that is more like 5 ms best service and that it could certainly take longer than that for the packet tx signal to be sent, but I am wondering if it could also take shorter than 5 ms to receive the packet tx signal, which could be problematic if my receiver was not ready for the data yet.

Is there a way to make the radio reject a new packet if there is already one currently in the FIFO that is being read? That way there is not a situation were a new packet is received while the old one is being read. The other question is, if this is the problem, why is my radio entering a state were it can not receive new information.

Here is how I look right now, this code is called from inside of my ISR that triggers whenever I get the Packet with CRC good RX signal from cc1101.

```
FIFOLength = CC1101_Read_Register(CC1101_RXBYTES);


RadioSlaveSlaveSelect_Write(0x00);/*start of spi transmision*/
while(1 == RadioMISO_Read());


/*Lets clear the RX FIFO*/
RadioSPIM_ClearFIFO();

RadioSPIM_WriteTxData(CC1101_RX_BURST_ACCESS);
while(0 == RadioSPIM_GetRxBufferSize());/*wait for Radio status byte to be clocked in*/
RadioStatus = RadioSPIM_ReadRxData();

RadioSPIM_WriteTxData(0x00);
while(0 == RadioSPIM_GetRxBufferSize());/*wait for PayloadLength byte to be clocked in*
PayloadLength = RadioSPIM_ReadRxData();


while(Count < PayloadLength)
{
    RadioSPIM_WriteTxData(0x00);
    while(0 == RadioSPIM_GetRxBufferSize());/*wait for data to clock in*/
    Data[Count] = RadioSPIM_ReadRxData();
    Count++;
}

/*read rssi*/
RadioSPIM_WriteTxData(0x00);
while(0 == RadioSPIM_GetRxBufferSize());/*wait for RSSI to clock in*/
RSSI = RadioSPIM_ReadRxData();

/*read packet status including crc*/
RadioSPIM_WriteTxData(0x00);
while(0 == RadioSPIM_GetRxBufferSize());/*wait for PacketStatus to clock in*/
PacketStatus = RadioSPIM_ReadRxData();



```

```
RadioSlaveSlaveSelect_Write(0x01);/*end of SPI transmission*/
/*Flush out Garbage*/
FIFOLength -= (Count + 3);
if(FIFOLength > 0)
{
    CC1101_Send_Command(CC1101_SFRX);
}
CC1101_Send_Command(CC1101_SRX);


return;
```

Thanks a ton for your post it was helpful!

Oleg Pushkarev

In reply to Jeremy Langford:

Hi,

TI's code works with 1,2 kb data speed. It's far from your 500 kb case. My packets go at 1 sec rate. Receiver works for a few days without any stumble.

To be onest I do not understand your code, but it seems that you do not check RX FIFO OVERFLOW, don't you?

Jeremy Langford

In reply to Oleg Pushkarev:

The first thing that I do is I check how many bytes are in the FIFO, then at the end of the code after I have read how many bytes are in actual data packet as dictated by the length byte, I then see if there is extra data in the FIFO, if there is, I give the SFRX strobe.

x = bytes in fifo

y = bytes in packet

z = status bytes length

a= x - y - z

if(a < 0){flush fifo}