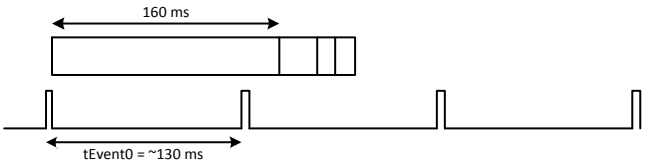
I have made an example for 1.2 kbps where the following packet is sent:
24 bytes preamble, 4 bytes sync, 2 bytes payload (1 length byte + 1 data byte), and 2 bytes CRC.
The transmitter then transmits for ((24 + 4 + 2 + 2)*8)/1200 = 213.33 ms
The radio should wake up often enough to be able to receive 4 bytes of preamble:
Wake-up interval = ((24 - 4)*8)/1200 = 133.33 ms
To add some margins, EVENT0 is set to 4506 (tEVENT0 = ~130 ms).



RX_TIME_RSSI = 1
RX_TIME_QUAL = 1
RX_TIME = 0 (12.5 % duty cycle, meaning that the RX timeout = 16.2 ms)
In this time the radio has the time to receive 19 bits (=> PQT set to 3 to add some margins)
I modified the Link program found here: http://www.ti.com/lit/zip/swrc021
The registers settings are for the 433 MHz band as this was the only EMs I had available.
The following register settings where used (both TX and RX):

```
halSpiWriteReg(CCxxx0_IOCFG0,       0x06); // SYNC (Used by app)
halSpiWriteReg(CCxxx0_IOCFG1,       0x5c); // LNA (debug only)
halSpiWriteReg(CCxxx0_IOCFG2,       0x08); // PQT Reached (debug only)
halSpiWriteReg(CCxxx0_FIFOTHR,      0x47);
halSpiWriteReg(CCxxx0_PKTCTRL0,     0x05);
halSpiWriteReg(CCxxx0_PKTCTRL1,     0x44);
halSpiWriteReg(CCxxx0_FSCTRL1,      0x06);
halSpiWriteReg(CCxxx0_FREQ2,        0x10);
halSpiWriteReg(CCxxx0_FREQ1,        0xA7);
halSpiWriteReg(CCxxx0_FREQ0,        0x62);
halSpiWriteReg(CCxxx0_MDMCFG4,      0xF5);
halSpiWriteReg(CCxxx0_MDMCFG3,      0x83);
halSpiWriteReg(CCxxx0_MDMCFG2,      0x13);
halSpiWriteReg(CCxxx0_MDMCFG1,      0x72);
halSpiWriteReg(CCxxx0_DEVIATN,      0x15);
halSpiWriteReg(CCxxx0_MCSM0,        0x18);
halSpiWriteReg(CCxxx0_FOCCFG,       0x16);
halSpiWriteReg(CCxxx0_WORCTRL,      0xFB);
halSpiWriteReg(CCxxx0_FSCAL3,       0xE9);
halSpiWriteReg(CCxxx0_FSCAL2,       0x2A);
halSpiWriteReg(CCxxx0_FSCAL1,       0x00);
halSpiWriteReg(CCxxx0_FSCAL0,       0x1F);
halSpiWriteReg(CCxxx0_TEST2,        0x81);
halSpiWriteReg(CCxxx0_TEST1,        0x35);
halSpiWriteReg(CCxxx0_TEST0,        0x09);
halSpiWriteReg(CCxxx0_PKTLEN,       61);
```

```
case TX:
  halSpiWriteReg(CCxxx0_IOCFG2,       0x5B);

  // Infinite loop
  while (TRUE) {

    while (!ebButtonPushed());

    halRfSendPacket(txBuffer, sizeof(txBuffer));

    intToAscii(++packetsSent);
    ebLcdUpdate("Sent:", asciiString);

    while (!ebButtonPushed());

    halWait(65000);
    halSpiStrobe(CCxxx0_STX); // Transmit only preamble to see how the RX behaves
    halWait(65000);halWait(65000);halWait(65000);halWait(65000);halWait(65000);halWait(65000);
    halSpiStrobe(CCxxx0_SIDLE);
  }
break;

case RX:

  halSpiWriteReg(CCxxx0_WORCTRL,      0x78);
  halSpiWriteReg(CCxxx0_WOREVT1,      0x11);
  halSpiWriteReg(CCxxx0_WOREVT0,      0x9A);
  halSpiWriteReg(CCxxx0_MCSM2,        0x18);
  halSpiWriteReg(CCxxx0_PKTCTRL1,     0x64);

  // Infinite loop
  while (TRUE) {
    halSpiStrobe(CCxxx0_SWOR);
// The below polling of GDO pins should be replaced by interrupt driven code
    // Wait for GDO0 to be set -> sync received
    while (!GDO0_PIN);

    // Wait for GDO0 to be cleared -> end of packet
    while (GDO0_PIN);

    if ((halSpiReadStatus(CCxxx0_RXBYTES) & BYTES_IN_RXFIFO)) {

      // Read length byte
      packetLength = halSpiReadReg(CCxxx0_RXFIFO);

      halSpiReadBurstReg(CCxxx0_RXFIFO, rxBuffer, packetLength);

      // Read the 2 appended status bytes (status[0] = RSSI, status[1] = LQI)
      halSpiReadBurstReg(CCxxx0_RXFIFO, status, 2);

      if (status[LQI] & CRC_OK) {
        intToAscii(++packetsReceived);
        ebLcdUpdate("Received:", asciiString);
      }
    }
  }
break;
```

The plot from the logic analyzer below shows how the receiver behaves when there are

1) No signal present (terminate due to no CS)
2) There are preamble at RX timeout but no following packet
3) and when a packet is sent