Communication Systems Lab
Final Project: ZigBee Receiver Implementation

Bing-Ang Xie, *B03901195*
Hsin-Ju Chen, *B02901036*

I. Introduction

OUR final project uses a software defined radio (SDR) NI USRP-2932 [1] to receive XBee S2 (Series 2) [2] signals. Based on IEEE 802.15.4 Standard (ZigBee) [3], our work is mainly focused on PHY layer and uses LabVIEW 2014 [4] to implement the software design of the receiver. By controlling XBee via X-CTU [5], we receive and demodulate the signals by LabVIEW front panel.

II. Overview of IEEE 802.15.4 Standard

A. 900 MHz

According to the standard, the modulations used for 900 MHz are mainly O-QPSK (offset quadrature phase shift keying), BPSK (binary PSK with differential encoder), and GFSK (Gaussian frequency shift keying with data whitening).

B. 2.4 GHz

At 2.4 GHz, ZigBee uses O-QPSK modulation with DSSS (direct sequence spread spectrum) (Fig. 1).

Fig. 1 Modulation Diagram

To realize DSSS, the original binary data is multiplied with a 32-bit binary sequence (chips) (Fig. 2). Then, the even-indexed chips are modulated into I (in-phase) carrier while the odd-indexed chips are modulated into Q (quadrature-phase) carrier. There is a time shift between I and Q carriers to minimize the phase change between each chip so that it never exceeds 90° (Fig. 3). By using half-sine pulse shaping (Fig. 4), the overall effect of the modulation will be equivalent to MSK (minimum shift keying) [6].

Data symbol	Chip values (c_0 c_1 … c_{30} c_{31})
0	1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0
1	1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0
2	0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0
3	0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1
4	0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1
5	0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0
6	1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1
7	1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1
8	1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1
9	1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1
10	0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1
11	0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0
12	0 0 0 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0
13	0 1 1 0 0 0 0 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1
14	1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 0 0
15	1 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0

Fig. 2 Chip Table

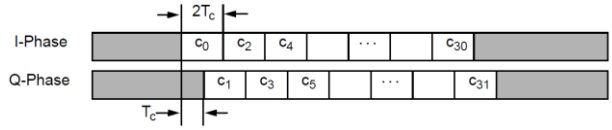

Fig. 3 O-QPSK Chip Offsets

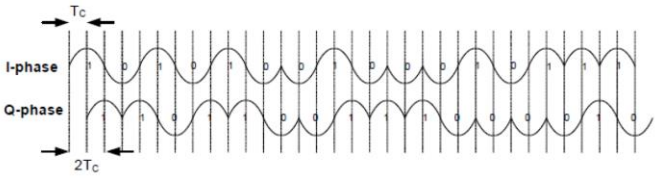

Fig. 4 Sample Chips with Half-Sine Pulse Shaping

As for the packet format, each PPDU includes SHR, PHR, and PHY payload. SHR is 40 bits (10 symbols) long and has a preamble of 32 zeros followed by an SFD 1110 0101. The PHR is 8 bits long with the first 7 bits indicating the length of the PHY payload and the last bit as a reserved bit. All bits are ordered with the LSB (least significant bit) first.

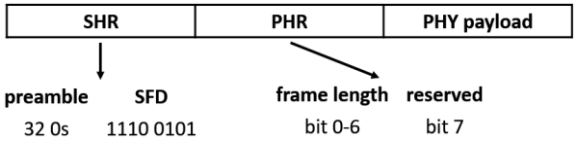

Fig. 5 PPDU Format

III. System Implementation Setup

A. USRP Rx

We do our main control here.

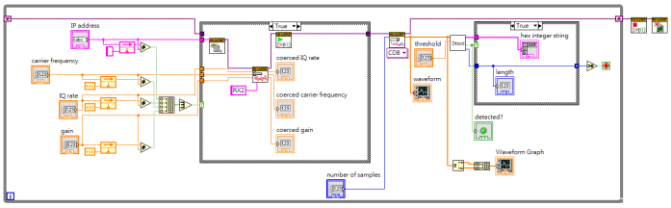
Fig. 6 USRP Rx Block Diagram

According to the control panel of X-CTU, the operating channel is channel C, which is channel 12 in decimal. In IEEE 802.15.4 (10.1.2.2), the channel frequency follows the following equation:

$$F_c(\text{MHz}) = 2405 + 5(k - 11)$$

where $k = 11,12,\cdots,26$.

Therefore, the operating channel is at 2.41 GHz.

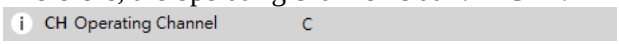

Fig. 7 Operating Channel on X-CTU Control Panel

Since the data rate is 250 kbps and that each symbol is consisted of 4 bits that maps to 32 chips, the chip rate is 2 M chips/s. To satisfy sampling theorem, the sampling rate must be at least twice the original bandwidth, so we set the IQ rate to 4 M while the samples to chip ratio (samples/chip) is 2.

B. Demodulator

The demodulator consists of a few parts: energy detector, carrier frequency offset, received waveform to binary, and packet locator.

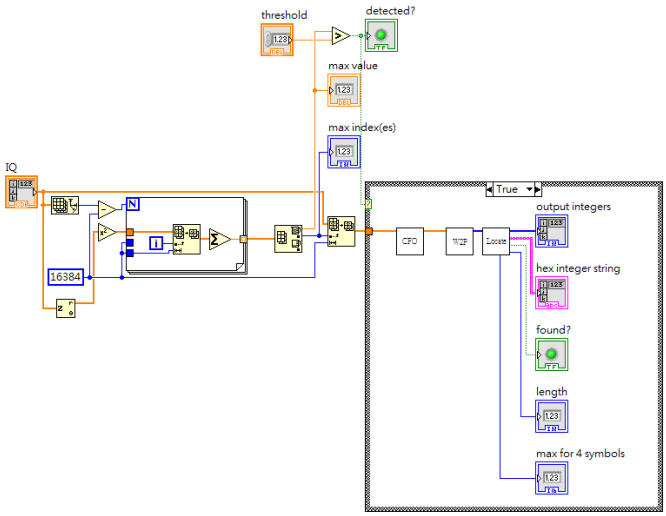

Fig. 8 Demodulator Block Diagram

Energy detector is implemented in the demodulator block. By summing the total energy of a certain length of window throughout the received signals, the following function blocks will only deal with the highest energy window. The length of the window is determined by the largest possible packet and according to IEEE 802.15.4 is 127 octets (we use 128 for convenience), which is 16384 in our experiment.

C. Carrier Frequency Offset

From the symbol synchronizer in lab 4 [7], we changed some of the parameters to fit our requirements (N = L = 128). The corrected signals are:

$$x[n] = e^{-j2\pi f_d n} y[n]$$

where $f_d = -\frac{1}{2\pi N}\arg R(k_{max})$, $k_{max} = \arg\max_k |R(k)|$, $R(k) = \frac{1}{L}\sum_{i=0}^{\infty} y[k+i]y^*[k+i+N]$.

So, we calculate the auto correlation of signals spaced by 128 (length of 1 preamble symbol at sample level). The highest value will occur in the first, second, or third symbol of the preamble. After acquiring the CFO, we corrected all the signals and passed those after the highest value to the next step.

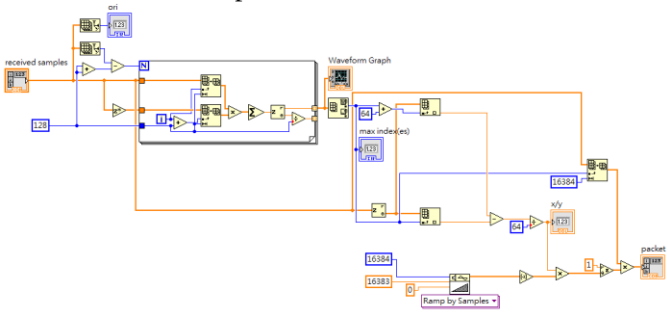

Fig. 9 Carrier Frequency Offset Block Diagram

D. Received Waveform to Binary

For this part, we have different versions of demodulation.

At 900 MHz, if O-QPSK is used, the chip table is a 16 square table. If BPSK is used, then a differential encoder is prior to BPSK in modulation (Fig. 10). If GFSK is used, then a data whitening process will be prior to GFSK in modulation (Fig. 11). The data whitening process is the XOR of the PHY payload field with a PN9 sequence (seed of nine ones).

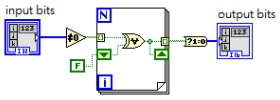

Fig. 10 Differential Encoder Block Diagram

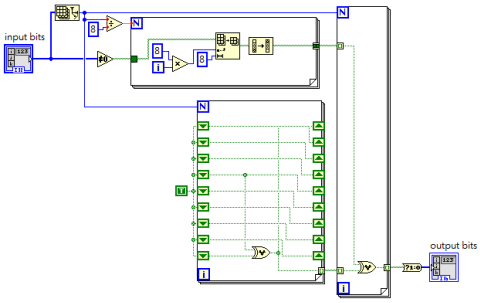

Fig. 11 Data Whitening Block Diagram

At 2.4 GHz, O-QPSK is used. We have tried three schemes here. The first one is to modulate the chip table with half-sine pulse shaping into waveforms, and then find the preamble waveform in the received signals. The second one is to transform the waveform into bits with direct demodulation (Fig. 12).

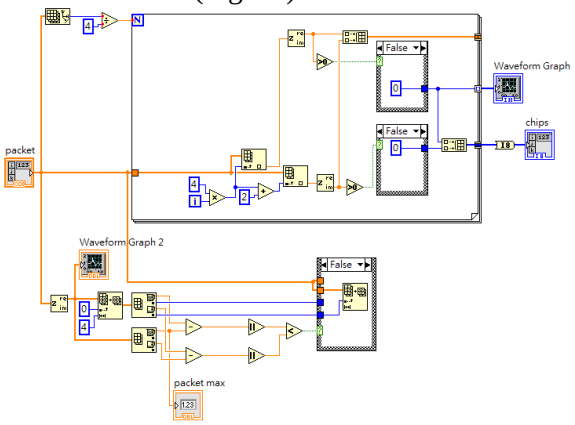

Fig. 12 Received Waveform to Chips Block Diagram

The third and last one is done by extracting the phase difference of each chip (Fig. 13). The first loop calculates the phase difference between each sample, and then the second one decides which two transitions are part of a phase difference between adjacent chips. Then the third loop combines the phase differences by grouping them two at a time. The last loop converts the phase differences into binary bits with 1 for positive difference and 0 for negative difference.

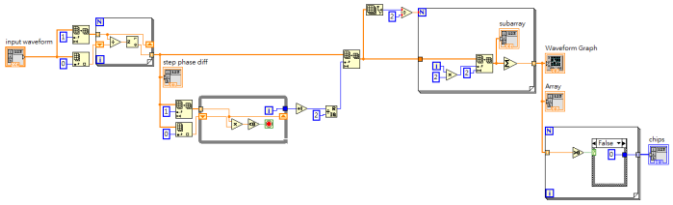

Fig. 13 Received Waveform to Phase Difference Chips Block Diagram

E. Locator

For the locator of the phase-difference scheme (Fig. 14), the original chip table is first converted into a phase difference chip table (Fig. 15, 16). Since there are only 31 phase differences for 32 chips when adjacent symbols are unknown, we made the 32nd value 0 for convenience. Therefore, since the 32nd value can be seen as random, if the matched bit count is either 31 or 32, we can consider it as a 100% match. By comparing the new chips between symbols, we found that the largest inter-symbol identical bit count is 19. Therefore, if the match is near 19, the symbol is considered an error.

Fig. 14 Locator Block Diagram

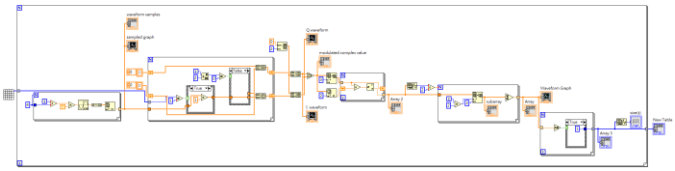

Fig. 15 Conversion of Chip Table Block Diagram

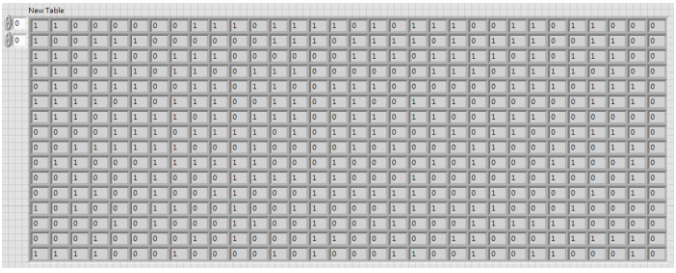

Fig. 16 Phase Difference Chip Table

We follow similar steps as the received signal conversion to phase difference for the chip table conversion. The original table is modulated with O-QPSK and half-sine pulse shaping. We then accumulated the step phase differences and acquired the phase difference chip table.

By searching for the last symbol of the preamble along with SFD in the received phase difference, we can locate the PHR and PHY payload. The similarities between the received chips and symbols in the chip table are also calculated. Then we convert the next two symbols back into bits to get the payload length by dropping the last bit. The last step extracts the payload according to the length and shows it in hexadecimal.

F. *Hardware Setup*

For 900 MHz, we use the module XBee-PRO 900HP [8] and SDR NI USRP-2920 [9]. The operating frequency of XBee is around 920 MHz and USRP can cover up to 2.2 GHz.

For 2.4 GHz, we use XBee S2 and NI USRP-2932. The operating frequency of XBee is around 2.45 GHz and USRP can cover up to 4.4 GHz.

IV. Main Achievements

Our experiment is mainly done at 2.4 GHz.

At our demonstration in class, we presented the first scheme mentioned above and received broadcasting packets.

However, after a few revisions, we built the third scheme after TA's suggestion of using non-coherent detection with phase differences and achieved a near 100% correct rate of receiving the symbols from the broadcasting packets.

Here is a screen recording video of a short 30-second demo of the third scheme: https://goo.gl/h47uPj (please view with the video quality of 720p or higher). The window on the right shows the matched bit count which is always 31 or 32, meaning that we have indeed found the packet.

V. Experimental Results

The journey of our final project starts at 900 MHz using XBee-PRO 900HP.

At first, we were not sure about the modulation scheme of the module, so we followed the IEEE 802.15.4 standard and tried to implement an O-QPSK version with the help of some of the references we have found [10]. Without much success, we also tried BPSK with differential encoder stated in the standard. Then, we found that the modulation type of the module we were using was GFSK [11]. So we used GFSK with data whitening and still cannot receive signals with higher energy level. Then, since the datasheet of XBee-PRO 900HP stated that FHSS is used, we tried multiple values and found that 920 MHz is one of the main channels.

After our unsuccessful decoding of XBee-PRO 900HP, TA lend us XBee S2 modules and a higher frequency USRP.

Our first attempt at 2.4 GHz is done by modulating the chip table in the standard to the same form as the transmit end should be, and try to recover the symbols by comparing the modulated table with the received waveform. During this attempt, we realized that the XBee module sends out broadcasting signals and that they are the main packet that we are receiving.

However, this method is rather rough and that demodulation should better be done at bit level than at waveform level, so we tried a different structure even though we were receiving consistent symbols at the time.

The next scheme was to switch the waveform back into chips. By extracting I and Q waveforms and transform them into chips, we interleave them according to O-QPSK and put back a received bit sequence. But the performance was bad, and with some help from TA, we started to implement a third method.

The last and most successful method is done by extracting phase differences. Since the prior scheme may have a big problem if the chips were all rotated for more than $\pi/4$, the use of phase differences in place of the original chips can solve the problem. So we converted the chip table into a phase difference table and dealt with the received signals at chip level.

The results were very different from before. While the intra-symbol matching of the preamble and SFD search was less than 70% in the second scheme, the last method nearly always achieve a striking 100%. Not only does the SHR has a high correct rate, the payload symbol matching rates were also always 100% (if we consider 31/32 as 100% due to the randomness of the last bit) (Fig. 17), and the demo video in section IV shows it all.

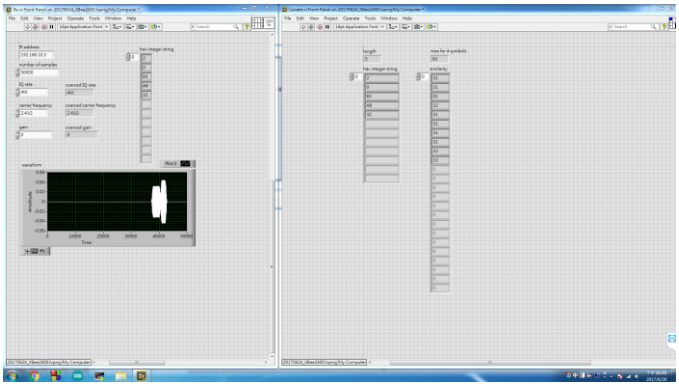

Fig. 17 Screenshot after Receiving Payload

There are two packet lengths that we keep receiving: one is 5 and the other is 12. Since the decoded symbols are consistent for the first few symbols but always different for the rest, we suspect that the messages we received are broadcasting signals from two of the modules we use. It is possible that the broadcasting signals consist time-variant messages.

While transmitting self-defined packets, we can sometimes receive packets with lengths other than 5 or 12. With larger distance, we are mysteriously more capable of discovering our packets compared with only seeing broadcasting signals when the distance is small. However, due to some possible payload encryption that we cannot turn off, even though the received payload has a 100% matching rate with the table, we cannot recover the payload message.

VI. Discussions

Our original assignment was to implement a receiver of ZigBee at 900MHz. The module we had was XBee-PRO 900HP, and we encountered a few problems:

- The main difficulty is its spread spectrum technique, FHSS (frequency-hopping spread spectrum). Though FHSS is different from DSSS, it still uses a similar idea that includes sequence. Unfortunately, IEEE 802.15.4 does not mention FHSS at all. Since we also read from Wikipedia that the communication protocol of the module we were using was proprietary [12], we were more confident that the PHY layer of XBee-PRO 900HP does not follow IEEE 802.15.4 standard. So in order to overcome this issue, we turned to some XBee forum and found that someone had testified that the FHSS sequence of XBee-PRO 900HP is indeed not public. We also found that some other person used SDR RTL2382U combined with GNU radio to try guessing the FHSS sequence via reverse engineering technique [13]. But since this article only captures a small part of one FHSS sequence and that the module supports 7 sequences, we realized that a clear understanding of the FHSS mechanism is time-consuming and almost impossible without knowing the hopping count after a few tries.
- The secondary difficulty is the lack of information of its modulation scheme. By the standard, it supports 19 types of modulation schemes and 3 of them are in 900MHz, which are O-QPSK, BPSK, and GFSK. However, in XBee-PRO 900HP datasheet, it does not mention the modulation scheme it adopts. Thus, we tried all of the 3 modulation schemes and found that GFSK performed best among them.

During the experiment, we found that if we put the XBee modules close to the USRP, the packets we receive are almost correct.

Since we receive packets with fixed period spacing, we suspect that the packets we are getting are broadcast signals. However, because we cannot find the packet structure of broadcast signal from the datasheet nor the standard, we cannot be sure what the actual content of the broadcast signal should be. Also, we discovered that if we put XBee modules further away from USRP, we can get both the broadcast packet and the packet we transmitted.

VII. Course Feedback

In order to give the teacher and TA some feedback, we racked our brains and end up with nothing.

The experiment and course arrangements are good but rather inversely proportional to the course credit. However, we did not regret taking this class because we gained the knowledge of both theoretical and practical aspects of communications system. Also, with the diligent TA, we could always seek help in time and acquire satisfactory hints.

References

[1] NI USRP-2932. http://www.ni.com/pdf/manuals/375988c.pdf
[2] XBee S2 (Series 2) by Digi International. http://www.hobbytronics.co.uk/datasheets/xbee_2mw_s2_90000976_G.pdf
[3] IEEE 802.15.4 standard. http://standards.ieee.org/getieee802/download/802.15.4-2015.pdf
[4] NI LabVIEW 2014. http://www.ni.com/labview/release-archive/2014/
[5] X-CTU. https://www.digi.com/resources/documentation/digidocs/PDFs/90001458-13.pdf
[6] MSK and O-QPSK material. http://www.dsplog.com/2009/06/16/msk-transmitter-receiver/
[7] *National Taiwan University Communication Systems Lab: Lab4 Handout*. 2017.
[8] XBee-PRO 900 HP by Digi International.
[9] NI USRP-2920. http://www.ni.com/pdf/manuals/375839b.pdf
[10] A laboratory exercise on 802.15.4 communication between software defined radio (SDR) and XBee (Boris Kolev)
[11] Radio Emission Designation and Modulation Type for Digi RF Radios. http://knowledge.digi.com/articles/Knowledge_Base_Article/Radio-Emission-Designation-And-Modulation-Type-For-Digi-RF-Radios
[12] XBee on Wikipedia. https://en.wikipedia.org/wiki/XBee
[13] Reverse engineering of XBee PRO PHY layer. http://xn--thibaud-dya.fr/phy_xbee_p1.html