# IEEE P802.15
# Wireless Personal Area Networks

| | |
|---|---|
| Project | IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs) |
| Title | Description 15.4z HRP UWB PHY Test Vectors |
| Date Submitted | [20 April, 2020] |
| Source | Manuel Lafer (NXP Semiconductors)<br>Frank Leong (NXP Semiconductors) |
| Re: | [If this is a proposed revision, cite the original document.]<br>[If this is a response to a Call for Contributions, cite the name and date of the Call for Contributions to which this document responds, as well as the relevant item number in the Call for Contributions.]<br>[Note: Contributions that are not responsive to this section of the template, and contributions which do not address the topic under which they are submitted, may be refused or consigned to the "General Contributions" area.] |
| Abstract | [Description 15.4z HRP UWB PHY test vectors] |
| Purpose | [Describe in detail contents of 802.15.4z HRP UWB PHY test vector zip file] |
| Notice | This document has been prepared to assist the IEEE P802.15. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein. |
| Release | The contributor(s) acknowledge(s) and accept(s) that this contribution becomes the property of IEEE and may be made publicly available by P802.15. |

# 1 Introduction

Based on the existing IEEE 802.15.4 HRP UWB PHY [1], the 15.4z amendment [2] includes means to enhance the ranging device (RDEV) by defining additional modes associated with the HRP enhanced ranging device (HRP-ERDEV). The main PHY enhancements are the inclusion of the Scrambled Timestamp Sequence (STS) field in the basic HRP PPDU format, and increased pulse repetition frequency (PRF) during preamble and data fields. An associated zip file [3] contains test vectors to facilitate HRP-ERDEV interoperability. The present document describes the contents of this zip file.

# 2 Zip File Contents

Within the zip file, various PHY test vectors are included in ".mat" file format. Additionally, the file "packet_info.txt" describes the contents of each of these test vectors in terms of the general and per-field parameters used to generate each vector. The file "plot_4z_hrp_packet.py" is a script that can be used in combination with a (freely available) Python 3 distribution to generate a graphical representation of any one of the test vectors. Finally, the file "readme.txt" provides a brief overview of the other files included in the zip file. The ".txt" files and the ".py" file use UTF-8 encoding [4] (typically associated with UNIX/Linux).

## 2.1 Format of the Test Vectors

The zip file contains test vectors associated with both mandatory operating parameter sets and optional sets. Variations of the mandatory parameter sets are indicated by an additional character after the set number, e.g., "BPRF Set #1a" indicates an optional variation of mandatory BPRF Set #1 from Table 57 in [2]. BPRF parameter set numbers which are not listed in Table 57 in [2] and HPRF parameter set numbers which are not listed in Table 58 in [2] also represent optional operating parameter sets. Note that most test vectors represent optional operating parameter sets, which ERDEVs are not required to support.

The PSDU data patterns are repetitive; for the default data length of 20 bytes, a 2-octet pattern (0xBABE) is repeated 10 times, while for data lengths of {127,1023,2047,4095} octets (odd numbers), a single-octet pattern (0xAB) is repeated {127,1023,2047,4095} times, respectively.

If an STS field is present in a test vector, its DRBG seed value is also included in the associated entry in "packet_info.txt", in two forms, both as complete seed and as separate (key,data) pair.

For all ".mat" test vector files, the BPSK modulation patterns are contained in a variable named "pulses_all". The elements of "pulses_all" contain a ternary {-1,0,1} pulse modulation sequence equidistantly spaced on the 499.2 MHz nominal chip timing grid.

## 2.2 Python Test Vector Plotter

The zip file also contains a Python script capable of plotting any of the provided test vectors. The script essentially processes a test vector to show the associated waveform as it would be generated by a Radio Frequency (RF) transmitter. The main signal processing steps are:

1. Convolve the test vector's ternary pulse modulation sequence with a reference pulse
2. Multiply the resulting baseband waveform with a sinusoidal RF carrier

The structure of the script is reviewed below and example script output is provided.

### 2.2.1 Script Structure

The "plot_4z_hrp_packet.py" script was tested using Anaconda Prompt, part of the freely obtainable Python distribution "Anaconda3-2019.07-Windows-x86_64", on MS-Windows 10. Trials suggest that it is also possible to run the script within Jupyter Notebook in the Firefox browser.

Within the script, the dependencies are listed as below.

```
import argparse
import decimal
import scipy.io as sio
from scipy.signal import hilbert
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import EngFormatter
from commpy.filters import rrcosfilter
```

Of these dependencies, all except "commpy" are included in "Anaconda3-2019.07-Windows-x86_64". "commpy" can be added by executing the following in Anaconda Prompt.

```
pip install scikit-commpy
```

The first main signal processing step is convolution with a reference pulse. The script uses the Root Raised Cosine (RRC) pulse from §16.4.5 in [1], with parameters $T_p$ = 1/(499.2 MHz) ≈ 2.00 ns and $T_w$ = 0.5 ns, as reference pulse. The associated code is shown below.

```
# Set chip rate associated with the test vector
chiprate = 499.2e6

# Define oversampled linear-phase RRC reference pulse
T_p = 1/chiprate # approximately 2 ns
rrcpulse_tmp = rrcosfilter(8*int(osr), 0.5, T_p, chiprate*int(osr))[1]
```

The reference pulse is oversampled w.r.t. the chip rate, by a factor defined in variable “osr”. This variable can be set from the command line using parameter “-o”. Large values of “osr” result in higher waveform fidelity, while smaller values result in faster execution. For the larger test vector files, sticking with the default factor of 64 is recommended.

Command line parameter “-m” can be used to apply a Discrete Hilbert Transform (DHT) based minimum-phase transform to the RRC reference pulse. The part of the script implementing the core of the minimum-phase transform is shown below.

```
def min_phase(firseq, fft_pts):
    """A function to convert a FIR impulse response to min-phase

    A function that calculates the minimum-phase equivalent of a FIR
    impulse response based on a Discrete Hilbert Transform
    applied in the frequency domain.
    """
    # Recommended fft_pts >= 32768
    #
    # For guidelines on the required number of FFT points, see:
    # N. Damera-Venkata, B. L. Evans and S. R. McCaslin,
    # "Design of optimal minimum-phase digital FIR filters using
    # discrete Hilbert transforms," in IEEE Transactions on Signal
    # Processing, vol. 48, no. 5, pp. 1491-1495, May 2000.
    max_phase = np.real(np.fft.ifft( \
        np.exp(hilbert(np.real(np.log(np.fft.fft( \
        firseq, fft_pts)))))))
    tmp_min_phase = max_phase[::-1] # reverse max phase
    return tmp_min_phase[0:len(firseq)] # maintain length
```

Depending on the presence of the “-m” or “--minphase" command-line parameter, the script executes the minimum-phase transform using 65536 (i.e., 2^16) Fast Fourier Transform (FFT) points. The associated part of the script is shown below.

```
# Check for "--minphase" => Set pulse shape accordingly
if args.minphase:
    # Set FFT size to be large, ensuring accuracy
    fftsize = 65536
    # Convert reference pulse to min-phase
    rrcpulse = min_phase(rrcpulse_tmp, fftsize)
    print("Pulse shape is minimum-phase RRC")
else:
    # Keep original linear-phase reference pulse
    rrcpulse = rrcpulse_tmp
    print("Pulse shape is linear-phase RRC")
```

After the reference pulse has been convolved with the ternary modulation sequence, upconversion with the carrier at the specified frequency is performed using “np.sin()”, as part of the “plt.plot()” command shown below.

```
# Plot the upconverted signal on specified carrier (default 8.0 GHz)
plt.plot(timeaxis, \
    bbmod*np.sin(timeaxis*2*np.pi*float(carriermult)*chiprate))
```

The carrier frequency can be set in multiples of the 499.2 MHz chip rate via the command-line parameter “-c”. For example, a carrier frequency of 6.5 GHz (Channel 5) is set using “-c 13”.

Modifying the following part of the script according to the inline comment will cause the script to plot the baseband modulation waveform (i.e., omitting the upconversion step).

```
# Uncomment next line to also plot the baseband modulation (envelope)
# plt.plot(timeaxis, bbmod)
```

### 2.2.2 Example Output

The built-in help can be accessed by executing the script with the “-h” parameter from the command prompt as follows.

```
python plot_4z_hrp_packet.py -h
```

If the above is executed, the script will provide the following text output.

```
usage: plot_4z_hrp_packet.py [-h] [--input INPUT] [--carrier CARRIER]
                             [--osr OSR] [--minphase]

Script to plot pulses from a test vector MAT file.

optional arguments:
  -h, --help            show this help message and exit
  --input INPUT, -i INPUT
                        set input data file (default is "bprf_1.mat")
  --carrier CARRIER, -c CARRIER
                        set carrier frequency multiplier x 499.2 MHz
  --osr OSR, -o OSR     set oversampling ratio (default is 64)
  --minphase, -m        set minimum-phase pulse (default is linear-phase)
```

The “bprf_1.mat” test vector (assumed to be located in the script's run directory) is plotted when the script is run without any parameters, as follows.

```
python plot_4z_hrp_packet.py
```

If the above is executed, the script will provide the following text output and the graphical output shown in Figure 1.

```
Processing input data file   : bprf_1.mat
Using carrier multiplier     : 16
Using carrier frequency [Hz] : 7.9872E+9
Using oversampling ratio     : 64
Pulse shape is linear-phase RRC
```

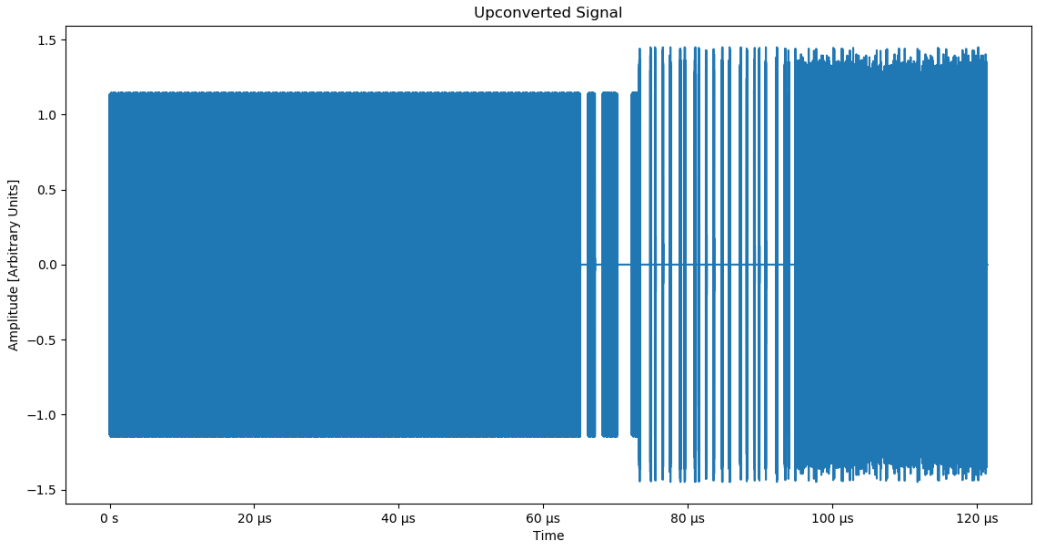


Figure 1: Graphical output with default script parameters

The example below plots “hprf_1.mat” using the built-in minimum-phase pulse shape.

```
python plot_4z_hrp_packet.py -i hprf_1.mat -o 256 -m
```

If the above is executed, the script will provide the following text output and the graphical output shown in Figure 2 (after some zooming has been performed by the user).

```
Processing input data file   : hprf_1.mat
Using carrier multiplier     : 16
Using carrier frequency [Hz] : 7.9872E+9
Using oversampling ratio     : 256
Pulse shape is minimum-phase RRC
```

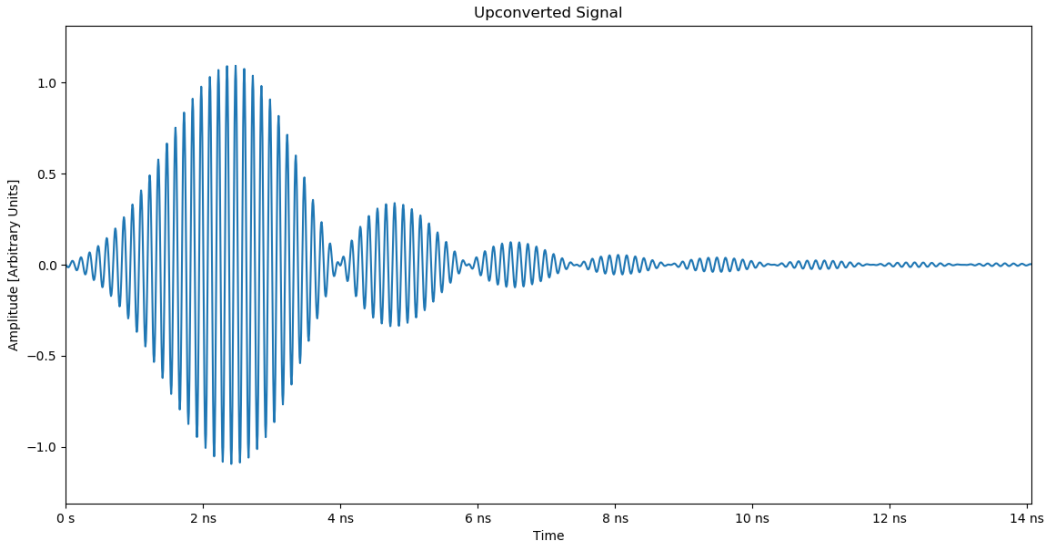


Figure 2: Graphical output with modified script parameters, zoomed in to the 0-14 ns range

Note that all of the above commands can also be run, in slightly modified form, within Jupyter Notebook. In that case, it is recommended to first execute a cell containing the following.

```
%matplotlib inline
```

Then, the example with modified script parameters can be invoked by a cell as shown below.

```
%run plot_4z_hrp_packet.py -i hprf_1.mat -o 256 -m
```

Here, it is assumed that the ".ipynb" notebook is located in the same directory as the other files.

# 3 References

[1] IEEE Standards Association, IEEE Std 802.15.4™-2015 – IEEE Standard for Low-Rate Wireless Personal Area Networks (WPANs).

[2] IEEE Standards Association, P802.15.4z™/D08 – Draft Standard for Low-Rate Wireless Networks.

[3] Manuel Lafer, Frank Leong, "15.4z HRP UWB PHY Test Vectors", IEEE 802.15, document number 15-20-0002, the latest version available at: https://mentor.ieee.org/802.15/documents.

[4] UTF-8, a transformation format of ISO 10646, available at: https://tools.ietf.org/html/rfc3629.