# Verification Document

| Team member | Student Number |
|---|---|
| Ilia Sosner | 49425259 |
| Cole Shanks | 54950860 |
| Guoyu Zhao | 65442865 |
| Reuben Singh Joginder | 89291884 |

Team Number: 8
Instructors: Dr. Paul Lusina and Dr. Cristian Grecu
Teaching Assistants: Hamed Noori and Fredy Alves
Course: ELEC 391
Document: Verification
Total Number of Pages: 17
Date: 17th June, 2020

# Table of Contents

## Full system verification (source to sink)

The full system implementation has 3 major aspects when it comes to verification.

- Bit error rate of the channel as well as source to sink.
- Audio transmission verification
- Text transmission verification
- Latency

We measured the bit error rate by running a simulation in modelsim to check 200,000 bits. The simulation bypasses error correction for this calculation. We chose to do this because checking BER with ECC included would take days of simulation time or at least many hours running on actual hardware in order to get a reasonably accurate result.

Since our ECC method is capable of correcting 1 wrong bit among a set of 512 bits, and given that our errors are relatively far apart (not burst errors), It is reasonable to assume that our end-to-end BER of 10^-9 is satisfied. The results of the BER simulation are shown in Figure 1.

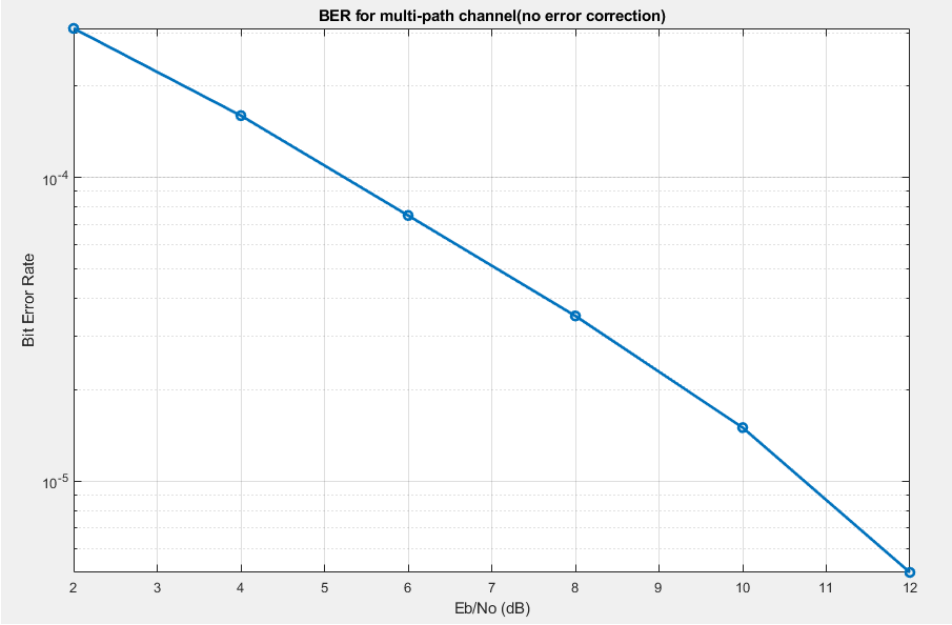



Fig-1: BER simulation results

We successfully verified that we could transmit a segment of a podcast played from a phone through our fpga system and out to the headphones. The audio was preprocessed with a 4 kHz low-pass filter to fit our audio bandwidth constraint. The audio quality was satisfactory and we were able to confirm the audio bandwidth limitation by

playing a frequency sweep and listening for the point where aliasing starts to happen (around 4 kHz).

Text transmission was verified by inspecting the memory contents before and after transmission using the in-system memory viewer in quartus. The viewer allows conversion to ascii so the text can be read. We read through the transmitted text segment and confirmed there were no errors. See Fig-A1 and Fig-A2 for memory contents.

In our modelsim simulations we were able to determine that there is an audio latency of 2.1E-4 seconds. The text transmission latency depends on the file size but we were able to verify in simulation that we can transmit 8000 characters per second with a minimum latency of 2.1E-4 seconds.

In terms of FPGA implementation the overall system used 4,625 ALM's (14%), 90,512 block memory bits (2%) and 87 DSP blocks (100%). Please see Fig-A3 and Fig-A4 in the appendix for more detailed resource and timing information.

# Compression encoding / decoding Verification

The list below contains elements of the compansion module that were implemented and verified:

## A-law Companding

- Correct mapping of the input signal to the output signals with minimal latency[1].
- Whether the linear approximation was sufficient.

## Huffman Companding

- Buffering at least 4 different ASCII characters, so that a compression ratio of around 3:1 is reached.
- Buffering incoming characters while processing is taking place.
- Verification of required clock cycles
- Variable code length
- Verification of latency

## A-law Companding Verification

For verification of the A-law companding module, a testbench and transmission of actual data was used. To verify whether the correct mapping was taking place with minimal latency, a testbench covering generic and extreme cases[2] was generated. See Figure 2 below.

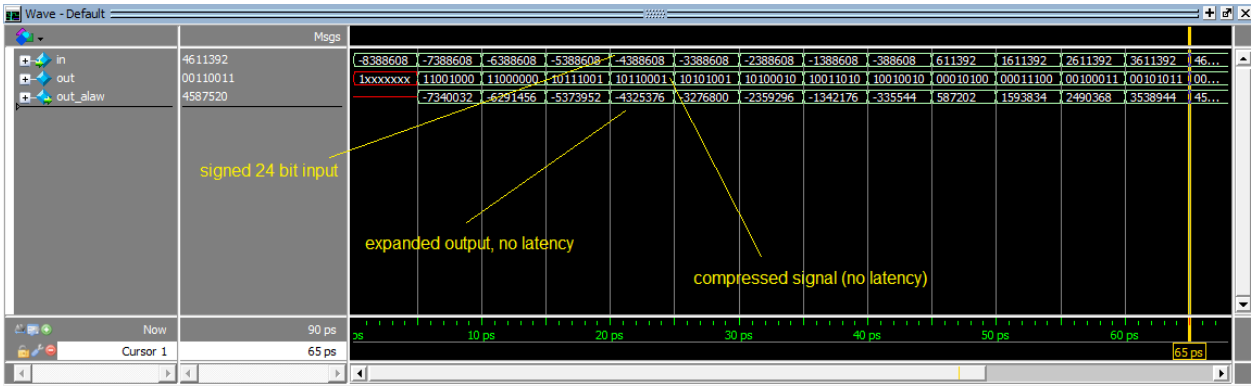



Fig-2: Testbench examining A-law Companding Module

[1] Minimal latency, minimal clock cycles, combinational logic.
[2] such as boundary inputs, error inputs, or inputs out of bounds.

A linear approximation was applied to the A-law companion curve (see appendix FigA-4). To verify the linear approximation was indeed reliable, the testbench in Figure 2, as well as transmission of audio through the entire system was used. The expanded values were reasonably close to the input values, and an audible output, similar to output received in our MATLAB Simulink simulation of A-law companion confirmed that the linear approximation was sufficient.

## Huffman Companion Verification

To verify that our Huffman Companion module indeed met the required objectives, a text message was encoded in 8 bit ASCII sequence, stored on a .mem file and was initialized in a testbench on ModelSim. The input, algorithmic manipulations stored in additional memory blocks, encoded sequence, states, and decoded output were investigated using the Waveform viewer. The analysis will be explained in the series of screenshots below, followed by how the functionality satisfies the project design objectives.

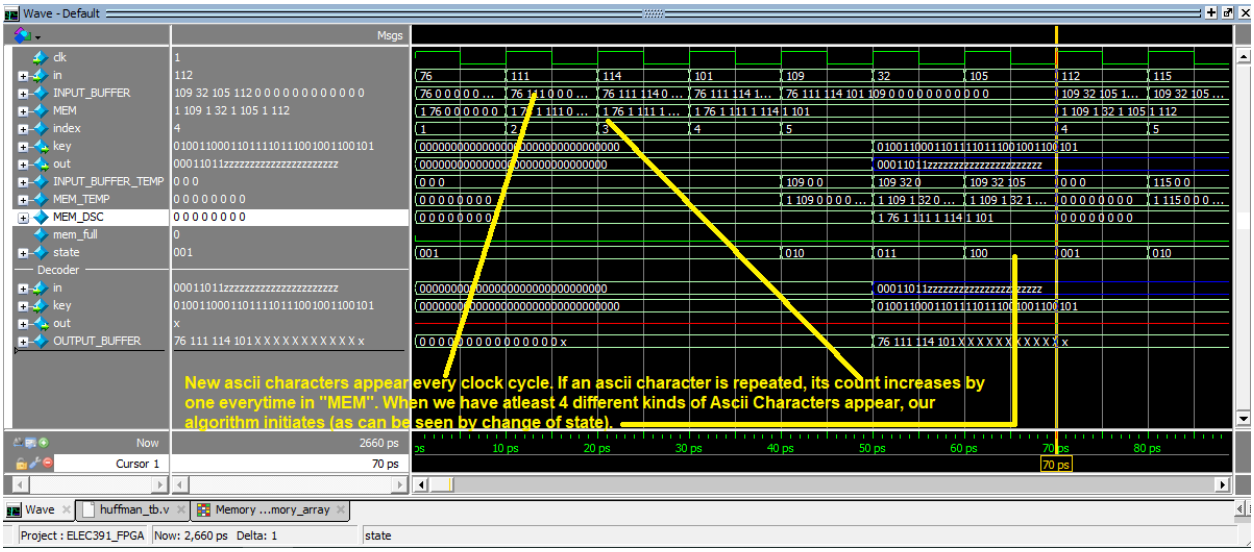



Fig-3: Annotated Huffman Testbench - ASCII characters buffered

As it can be seen in the screenshot, buffering at least 4 different ASCII characters, and taking into account the repeating characters allows for a fairly reasonable compression ratio (3:1), allowing for smaller transmission data sizes.

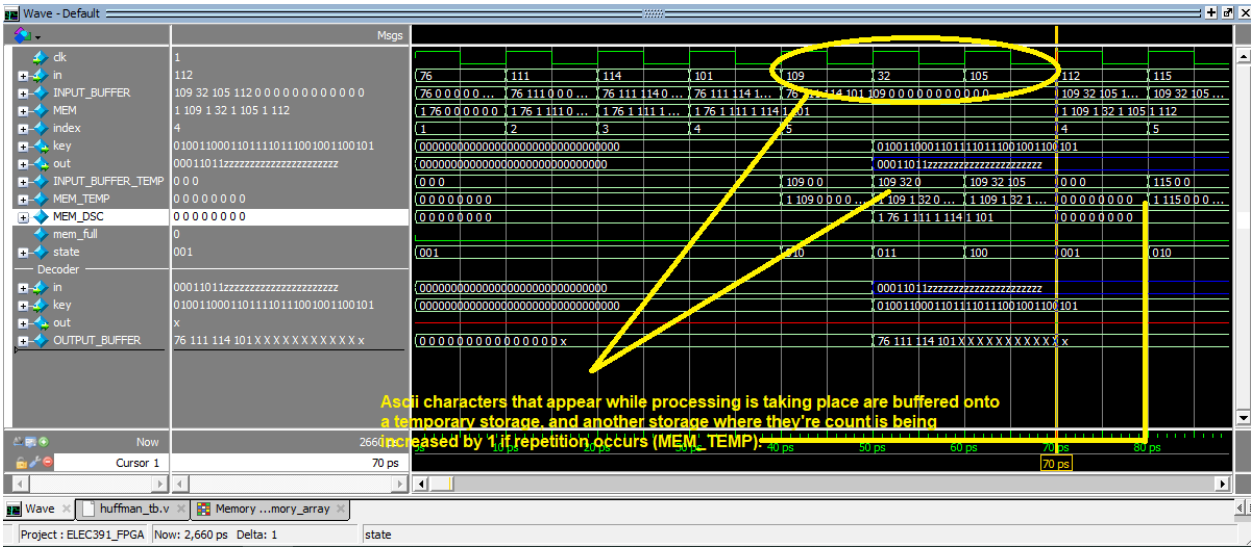



Fig-4: Annotated Huffman Testbench - Incoming ASCII characters buffered.

Buffering of ASCII characters that appear while processing is taking place, ensures no ASCII characters are missed as the system is running.

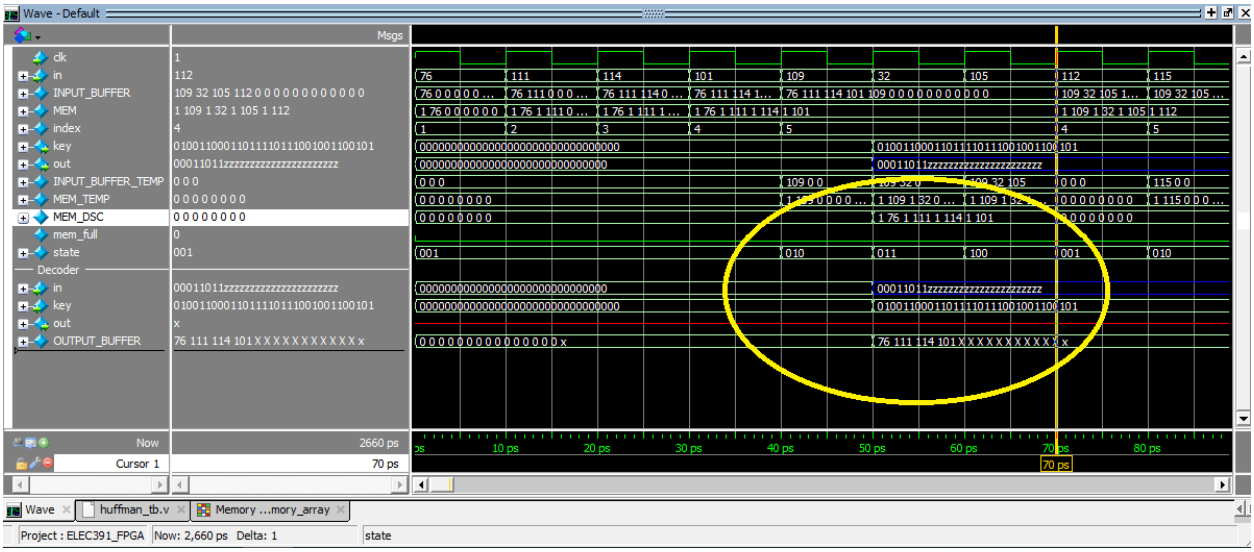



Fig-5: Annotated Huffman Testbench - Combinational Logic

The processing of ASCII characters (applying algorithm) and production of an encoded sequence is produced within 3 clock cycles, which are the minimum, ensuring low latency. The algorithm also accounts for variable length codewords (fed into the decoder). Finally, the decoder has minimal latency as can be seen by the output within the same clock cycle.

# Error Correction Encoding / Decoding Verification

To verify the Error Correction Encoding/Decoding process, we first started with a very simple testbench to ensure that data could be reliably encoded, sent across the channel, and then be decoded at the other end. After this was confirmed, the next step was to introduce noise into the channel and see how much corrupted data could be restored. For the Hamming (72, 64) encoding we are using, the decoder can correct 1-bit errors and detect 2-bit errors although it cannot correct them. The relevant flag signals are

- *err_detected_sig* indicates that an error has been detected.
- *err_corrected_sig* says a single-bit error has been found and corrected.
- *err_fatal_sig* denotes that a double-bit error has been found, but not corrected. You must not use the data if this signal is asserted.

To do this, we will look at sending the value 45 across the channel which has a binary representation of 101101. Noise will have a value of 2 and will be added to the signal.

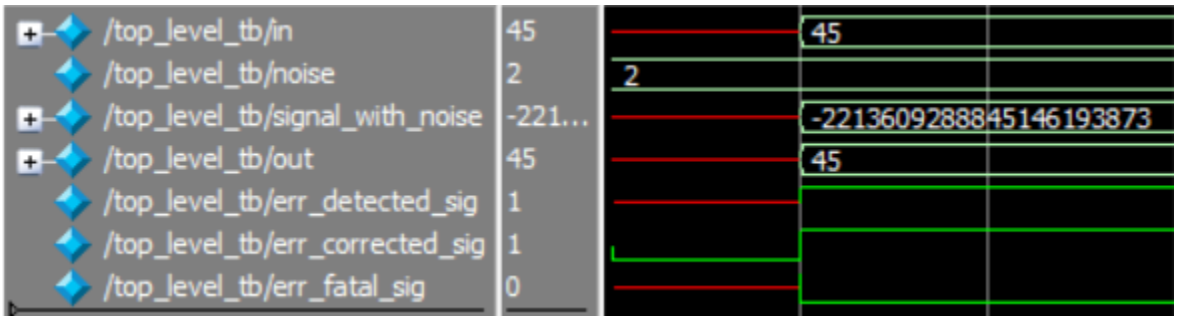



Fig-6: Encoding Example - Transmitted Value of 45

*signal_with_noise* now has the value of 47 for it's data bits and 8 parity bits attached which is why the value is so different. 47 in binary is 101111 which is only a 1-bit difference from our original data. It can be seen here that the decoder is able to correctly restore the data. Flags are up that indicate that an error has been detected and corrected.

Next, Noise will have a value of 1 which will change the value of our data in the channel to be 46

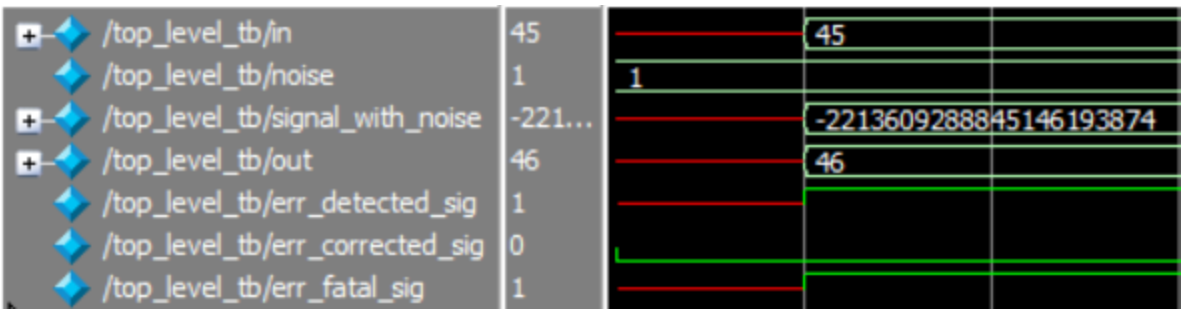


Fig-7: Encoding Example: Noise set to 1

46 in binary is 101110. This is now a 2-bit difference from our original data. It can be seen that the decoder is unable to restore the data. It simply outputs the corrupted message. Flags are up that indicate multiple errors have been detected but the data has not been corrected. *err_fatal_sig* is up and this tells us that we should not use this data.

If we add more noise the results are the same. Here, Noise is 3 which leads to a value of 48 in the channel.

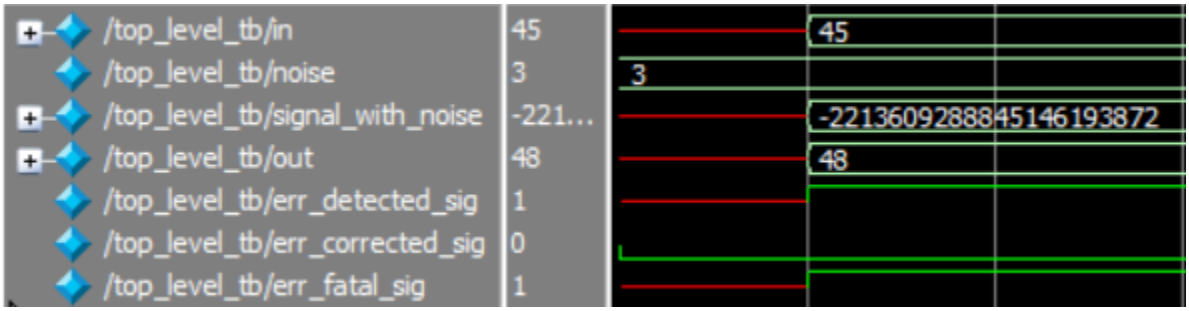


Fig-8: Encoding Example: Noise set to 3

48 in binary is 110000. This is now a 5-bit difference from our original data. It can be seen again that *err_fatal_sig* is up and this data should be thrown away.

# Modulation / Demodulation

The BPSK modulation/ demodulation implementation has the following components tested and verified:

- The incoming bit to the modulator is mapped to the correct symbol within the same clock cycle (combinational logic).
- The incoming symbol to the demodulator is converted to the correct bit value, also using combinational logic.

## BPSK Modulation/ Demodulation Verification

A testbench was created to verify if the BPSK modules met the required objectives. For readability, the Bit-Symbol / Symbol-Bit mapping is shown in the table below.

| Bit | Symbol |
|---|---|
| 0 | -1 |
| 1 | 1 |

Table-1: Bit to Symbol Mapping

The figures below show annotated screenshots of the simulation for modulation and demodulation, verifying that the required objectives are met and correct mapping is achieved.

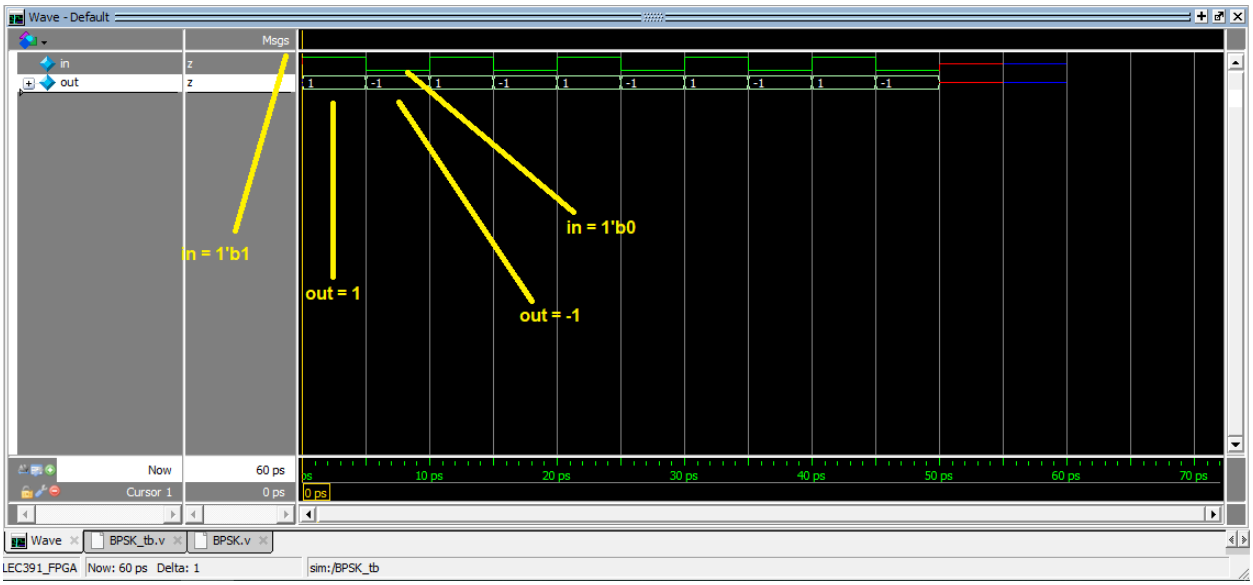



Fig-9: BPSK Modulation Testbench

For modulation, as it can be seen from the simulation screenshot above, there is minimal latency in mapping the bit to the symbol.

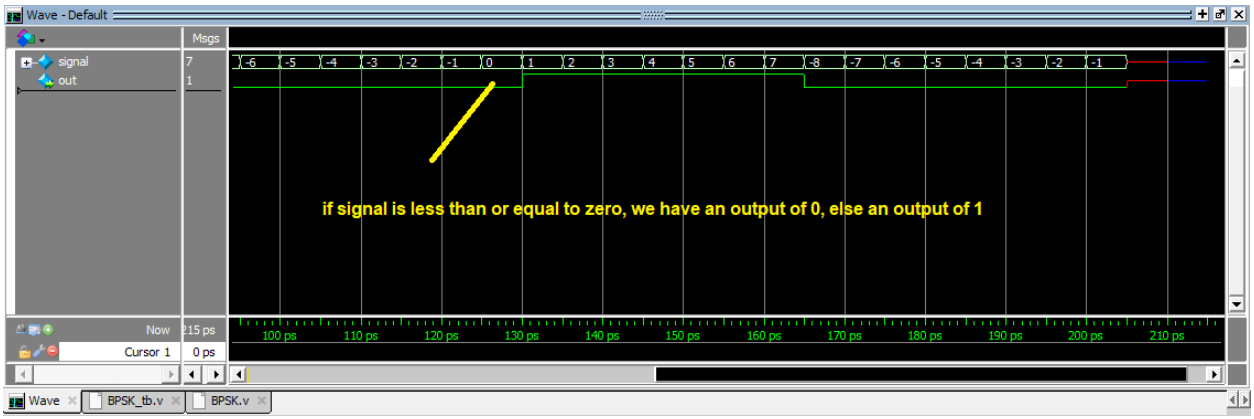



Fig-10: BPSK Demodulation Testbench

For demodulation, as it can be seen from the simulation screenshot above, there is minimal latency, and any value less than or equal to 0 is mapped to a 0, else it is mapped to a 1.

# Transmitter and Receiver Verification

## Transmitter

The transmitter implementation has 4 main parts that were implemented and verified. These parts are as listed below:

- Waveform data generation and frequency analysis in matlab
- Storage of waveform data in FPGA RAM
- Retrieval of waveform samples
- Multiplication by the input symbol

To generate the waveform data we use a matlab script that generates a frequency analysis of the pulse and also a table of waveform sample values. These values are converted to Hex in Excel and stored in a quartus memory initialization file (.mif). These values were also plotted in excel to verify the general shape of the gaussian pulse. One period of the waveform consists of 60 32-bit samples.

These samples are initialized into memory on compilation. The samples are then accessed cyclically at 10 clock cycles per sample. This signal is then multiplied with the input symbol (+/- 1) to create the final output waveform. Some of these verification steps are shown in Figures 11-12 below.

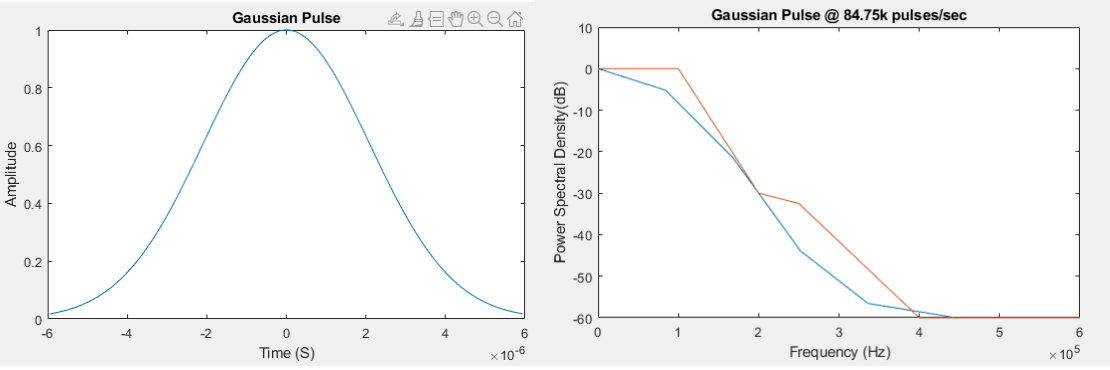



Fig 11-12. Gaussian pulse shape and frequency domain analysis with spectral power mask.

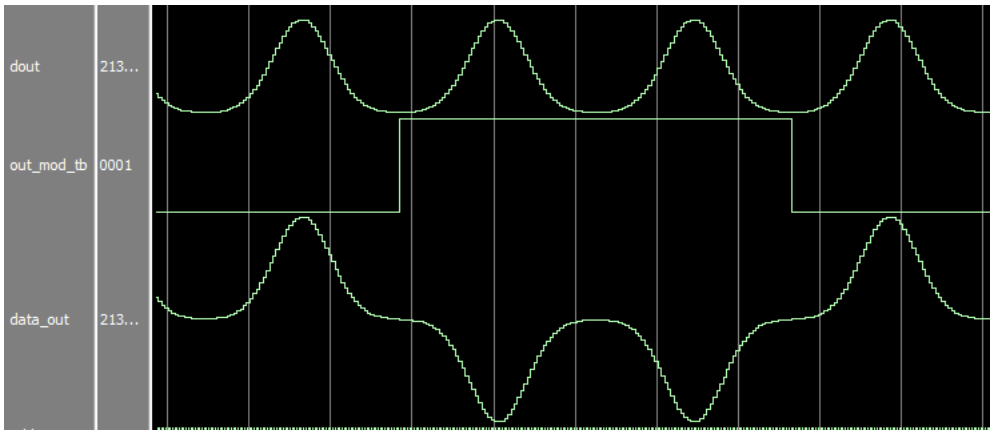


Fig-12: Memory retrieved waveform, modulation input, and resulting output transmission waveform.

In terms of frequency domain verification we relied on matlab simulation and the fact that our overall system performed as expected in terms of overall data-rates and audio bandwidth capabilities. We were able to achieve our target transmission rate of 83.3 kHz. With a required audio data rate of only 72 kbps, this transmission rate was sufficient.

## Receiver

The receiver has 3 main parts that we implemented and verified:

- Filter noise out from input signal with the use of a matched filter
- Sample the output of the matched filter at symbol period intervals
- Reconstruct the original data from the mixed path signal using a delay and scaling.

Much like the transmitter, the 60 coefficients for the matched filter were generated in matlab and loaded into memory on the fpga. The matched filter removes the majority of the gaussian noise and recovers a slightly distorted multipath signal.
This signal is then sampled every 600 clock cycles (1 symbol period). To recover the original transmitted signal we used the delay and scaling method shown in Figure 13. The transmitted data is then successfully recovered at the demodulator.

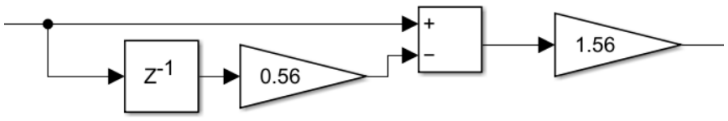


Fig-13: The multipath recovery circuit with a delay of one symbol period.

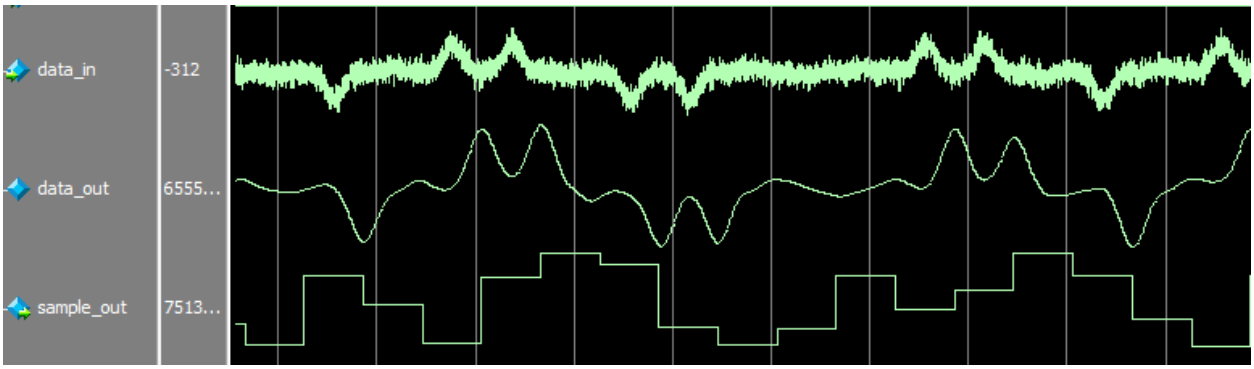


Fig-14: From top to bottom: Noisy multipath input signal, output of matched filter, modulation symbols recovered from multipath circuit (output of receiver).

## Channel Verification

The channel has 4 functions that have to be implemented and verified:

- Create two “paths” from the source signal, both scaled to conserve the original single path energy, one delayed by a symbol period.
- Combine the two paths back into one.
- Attenuate the combined signal.
- Add a specific amount of additive white gaussian noise (AWGN)

To verify the first three points we use modelsim to inspect the various internal waveforms of the channel verilog module. Figures 15-17 below show the input waveform, the 2 scaled and shifted signals and the resulting combined signal.

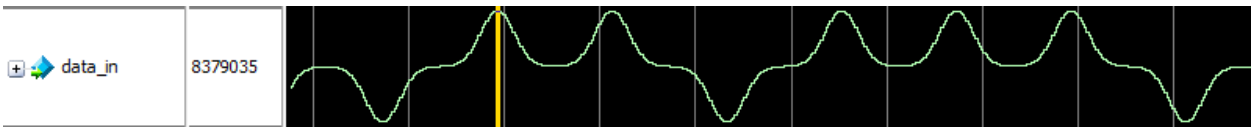


Fig-15: Single Path input signal with an amplitude of~8.37M.

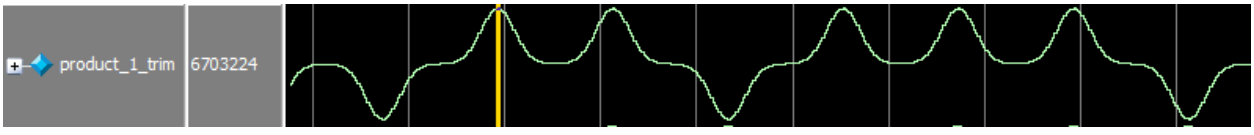


Fig-16: Path one scaled by 0.8 to amplitude of ~6.7M (no delay).

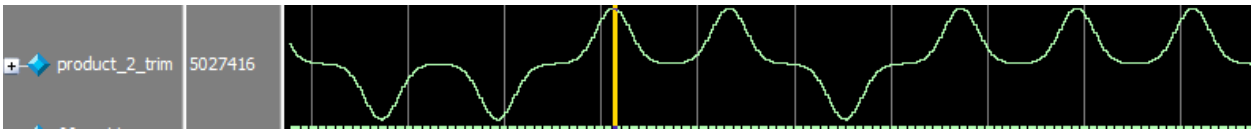


Fig-17: Path two scaled by 0.6 to amplitude of ~5M (delayed one symbol period).

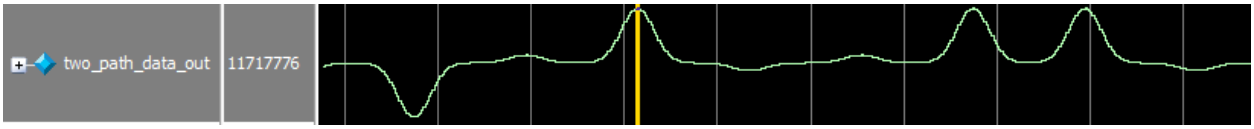


Fig-18: Combined signal.

Attenuation for verification was set according to the following formula for free space loss.

$$FSPL = 20\ log_{10}(d) + 20\ log_{10}(f) + \ 20log_{10}\left(\frac{4\pi}{c}\right)$$

Where d is distance (we chose 110m), f is 900 MHz and c is the speed of light. This results in an attenuation of 71.2 dB or a division by 4096. The resulting waveform looks exactly like Figure 18 but with a smaller magnitude.

To verify that the energy of the original pulse was being conserved, the waveform data was exported to excel and the signal energy was calculated for the source pulse as well as the 2 scaled paths. The computed results confirm that the sum of the two path energies is equal to the original pulse. Please see Excel file named Energy_and_BER.xls for details. The computed values are shown in table 2 below.

| Single Path Pulse energy | Path 1 | Path 2 | Path 1+ Path 2 |
|---|---|---|---|
| 12.963 | 8.297 | 4.667 | 12.963 |

Table 2- Computed pulse energies for multipath

The last step in the channel is the addition of noise. We used an AWGN module from OpenCores.org. To attain various SNRs we measured the symbol energy, and calculated the required noise variance and noise energy. To change the variance of the AWGN module we also used a scaling factor as shown in Table 3.

| Required Variance | SNR(dB) | Eb/No | Eb | No | Required Noise Scaling Factor |
|---|---|---|---|---|---|
| 135696.7121 | 12 | 3.9805 | 12.9633802 | 3.256721222 | 0.180344192 |
| 170843.9187 | 10 | 3.1616 | 12.9633802 | 4.100254212 | 0.20235636 |
| 215136.808 | 8 | 2.51069 | 12.9633802 | 5.163283597 | 0.227108959 |
| 270754.0593 | 6 | 1.99495 | 12.9633802 | 6.498097682 | 0.25477903 |
| 341040.7813 | 4 | 1.5838 | 12.9633802 | 8.184979079 | 0.285903951 |
| 429111.4058 | 2 | 1.25874 | 12.9633802 | 10.29867415 | 0.3207511 |

Table 3- Desired SNRs and calculated AWGN parameters.

The resulting noise signal is then added to the attenuated multipath signal to form the output of the channel module. These steps are shown below in Figure 19 and Figure 20.

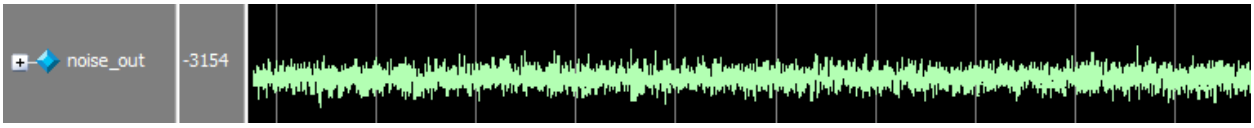



Fig-19: Noise output from AWGN module

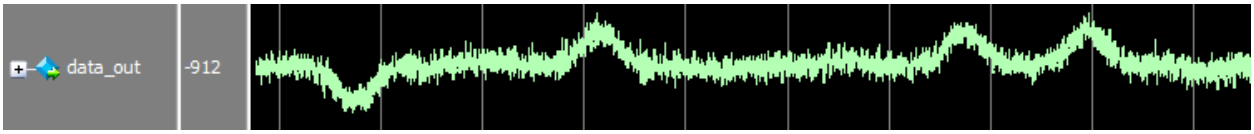



Fig-20: Noise added to multipath signal

# Technical Appendix

Instance 1: SRC

```
000000  43 48 41 50 54 45 52 20 49 0A 0A 4F 6E 20 61 6E 20 65  CHAPTER I..On an e
000012  78 63 65 70 74 69 6F 6E 61 6C 6C 79 20 68 6F 74 20 65  xceptionally hot e
000024  76 65 6E 69 6E 67 20 65 61 72 6C 79 20 69 6E 20 4A 75  vening early in Ju
000036  6C 79 20 61 20 79 6F 75 6E 67 20 6D 61 6E 20 63 61 6D  ly a young man cam
000048  65 20 6F 75 74 20 6F 66 0A 74 68 65 20 67 61 72 72 65  e out of.the garre
00005a  74 20 69 6E 20 77 68 69 63 68 20 68 65 20 6C 6F 64 67  t in which he lodg
00006c  65 64 20 69 6E 20 53 2E 20 50 6C 61 63 65 20 61 6E 64  ed in S. Place and
00007e  20 77 61 6C 6B 65 64 20 73 6C 6F 77 6C 79 2C 20 61 73   walked slowly, as
000090  20 74 68 6F 75 67 68 0A 69 6E 20 68 65 73 69 74 61 74   though.in hesitat
0000a2  69 6F 6E 2C 20 74 6F 77 61 72 64 73 20 4B 2E 20 62 72  ion, towards K. br
0000b4  69 64 67 65 2E 0A 0A 48 65 20 68 61 64 20 73 75 63 63  idge...He had succ
0000c6  65 73 73 66 75 6C 6C 79 20 61 76 6F 69 64 65 64 20 6D  essfully avoided m
0000d8  65 65 74 69 6E 67 20 68 69 73 20 6C 61 6E 64 6C 61 64  eeting his landlad
0000ea  79 20 6F 6E 20 74 68 65 20 73 74 61 69 72 63 61 73 65  y on the staircase
0000fc  2E 20 48 69 73 0A 67 61 72 72 65 74 20 77 61 73 20 75  . His.garret was u
00010e  6E 64 65 72 20 74 68 65 20 72 6F 6F 66 20 6F 66 20 61  nder the roof of a
000120  20 68 69 67 68 2C 20 66 69 76 65 2D 73 74 6F 72 69 65   high, five-storie
000132  64 20 68 6F 75 73 65 20 61 6E 64 20 77 61 73 20 6D 6F  d house and was mo
000144  72 65 0A 6C 69 6B 65 20 61 20 63 75 70 62 6F 61 72 64  re.like a cupboard
000156  20 74 68 61 6E 20 61 20 72 6F 6F 6D 2E 20 54 68 65 20   than a room. The
000168  6C 61 6E 64 6C 61 64 79 20 77 68 6F 20 70 72 6F 76 69  landlady who provi
00017a  64 65 64 20 68 69 6D 20 77 69 74 68 20 67 61 72 72 65  ded him with garre
00018c  74 2C 0A 64 69 6E 6E 65 72 73 2C 20 61 6E 64 20 61 74  t,.dinners, and at
```

FigA-1: Quartus In-system Memory Viewer showing source memory.

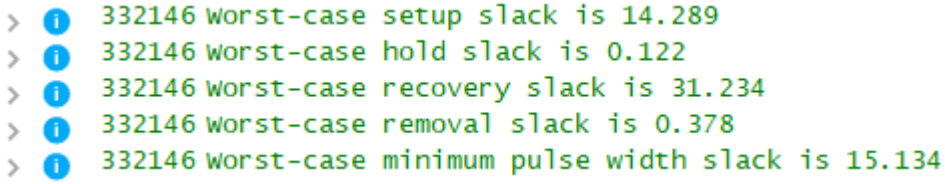
Instance 2: SINK

```
000000  43 48 41 50 54 45 52 20 49 0A 0A 4F 6E 20 61 6E 20 65  CHAPTER I..On an e
000012  78 63 65 70 74 69 6F 6E 61 6C 6C 79 20 68 6F 74 20 65  xceptionally hot e
000024  76 65 6E 69 6E 67 20 65 61 72 6C 79 20 69 6E 20 4A 75  vening early in Ju
000036  6C 79 20 61 20 79 6F 75 6E 67 20 6D 61 6E 20 63 61 6D  ly a young man cam
000048  65 20 6F 75 74 20 6F 66 0A 74 68 65 20 67 61 72 72 65  e out of.the garre
00005a  74 20 69 6E 20 77 68 69 63 68 20 68 65 20 6C 6F 64 67  t in which he lodg
00006c  65 64 20 69 6E 20 53 2E 20 50 6C 61 63 65 20 61 6E 64  ed in S. Place and
00007e  20 77 61 6C 6B 65 64 20 73 6C 6F 77 6C 79 2C 20 61 73   walked slowly, as
000090  20 74 68 6F 75 67 68 0A 69 6E 20 68 65 73 69 74 61 74   though.in hesitat
0000a2  69 6F 6E 2C 20 74 6F 77 61 72 64 73 20 4B 2E 20 62 72  ion, towards K. br
0000b4  69 64 67 65 2E 0A 0A 48 65 20 68 61 64 20 73 75 63 63  idge...He had succ
0000c6  65 73 73 66 75 6C 6C 79 20 61 76 6F 69 64 65 64 20 6D  essfully avoided m
0000d8  65 65 74 69 6E 67 20 68 69 73 20 6C 61 6E 64 6C 61 64  eeting his landlad
0000ea  79 20 6F 6E 20 74 68 65 20 73 74 61 69 72 63 61 73 65  y on the staircase
0000fc  2E 20 48 69 73 0A 67 61 72 72 65 74 20 77 61 73 20 75  . His.garret was u
00010e  6E 64 65 72 20 74 68 65 20 72 6F 6F 66 20 6F 66 20 61  nder the roof of a
000120  20 68 69 67 68 2C 20 66 69 76 65 2D 73 74 6F 72 69 65   high, five-storie
000132  64 20 68 6F 75 73 65 20 61 6E 64 20 77 61 73 20 6D 6F  d house and was mo
000144  72 65 0A 6C 69 6B 65 20 61 20 63 75 70 62 6F 61 72 64  re.like a cupboard
000156  20 74 68 61 6E 20 61 20 72 6F 6F 6D 2E 20 54 68 65 20   than a room. The
000168  6C 61 6E 64 6C 61 64 79 20 77 68 6F 20 70 72 6F 76 69  landlady who provi
00017a  64 65 64 20 68 69 6D 20 77 69 74 68 20 67 61 72 72 65  ded him with garre
00018c  74 2C 0A 64 69 6E 6E 65 72 73 2C 20 61 6E 64 20 61 74  t,.dinners, and at
```

FigA-2: Quartus In-system Memory Viewer showing transmitted text stored in sink memory.

| | |
|---|---|
| Flow Status | Successful - Wed Jun 16 22:30:34 2021 |
| Quartus Prime Version | 19.1.0 Build 670 09/22/2019 SJ Lite Edition |
| Revision Name | final_demo_top |
| Top-level Entity Name | final_demo_top |
| Family | Cyclone V |
| Device | 5CSEMA5F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 4,625 / 32,070 ( 14 % ) |
| Total registers | 5696 |
| Total pins | 91 / 457 ( 20 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 90,512 / 4,065,280 ( 2 % ) |
| Total DSP Blocks | 87 / 87 ( 100 % ) |
| Total HSSI RX PCSs | 0 |
| Total HSSI PMA RX Deserializers | 0 |
| Total HSSI TX PCSs | 0 |
| Total HSSI PMA TX Serializers | 0 |
| Total PLLs | 1 / 6 ( 17 % ) |
| Total DLLs | 0 / 4 ( 0 % ) |

FigA-3*: Quartus Compilation report for Top-Level Design.*




FigA-4*: Quartus Timing Analysis for Top-Level Design*