Project 3:

Wireless DTMF Transceiver Design Using USRP

ECE 4271

Rahmaan Lodhia

Daniel Weiss

Santhosh Karnik

1. Transceiver and DTMF Overview

The DTMF transceiver was created based on the flowchart shown in Figure 1.

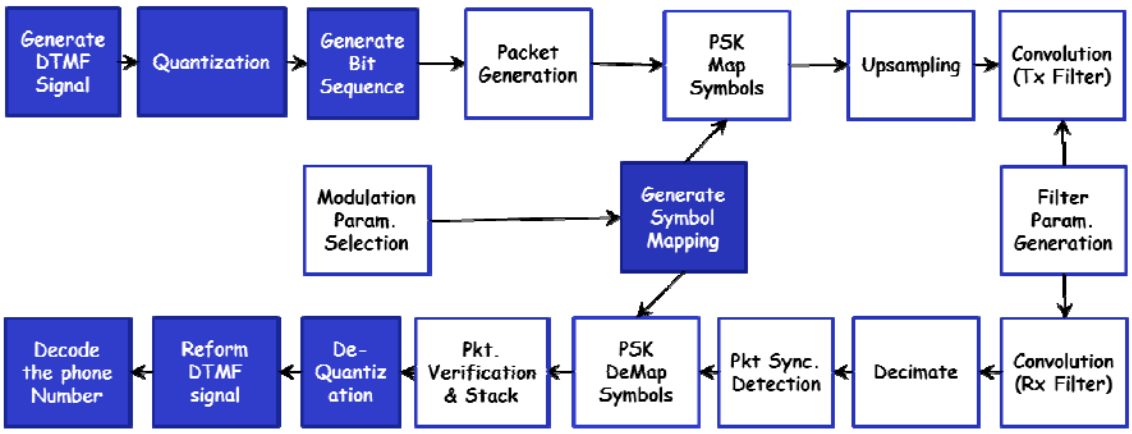

Figure 1. Block diagram of entire DTMF tranciever program.

2. LabVIEW subVI Overview

2.1. Sub_DTMF_Encoder.vi

The encoder subVI takes in the desired phone number to create and whether or not the signal should contain the dial tone. This data is passed through a MathScript module that creates the desired phone signal, outputs the data as a double array, and plays the resulting dial sound.

2.2. Sub_DTMF_Quantize_BitEncoding.vi

The quantize/bit-encoding subVI takes in the double array of the DTMF waveform and outputs the bit stream to be transferred. First, the quantization equations are applied using labVIEW blocks. The bit sequences that represent each quantization level are created in MathScript for easy of matrix entry. Each row in the matrix

represents on level with the columns representing the bits that make up that level. The switch statement in the MathScript outputs the proper matrix based on the quantization size and the Boolean gray encoding signal. A bit matrix is then initialized to the proper size and a for-loop is created in labVIEW to insert the proper bits based on the quantized signal.

2.3. Sub_DTMF_BitPadding.vi

The bit padding subVI takes in the received bit-stream and the number of quantization levels. Based on this, a choose labVIEW VI is used to either pass the input bit-stream to the output or to append zeros to the received bit-stream. If the remainder of the length of the received bit-stream divided by the logarithm base two of the quantitation levels is zero then the input is directly passed to the output. Otherwise, the difference between the quantization levels and the remainder is taken and that difference gives the number of zeros that are appended to the signal.

2.4. Sub_DTMF_Dequantize_BitDecoding.vi

The dequantize and bit decoding subVI takes in the received bit-stream with the padded zeros along with the quantization level, L, and whether or not gray code is used. The VI then uses a math script to convert each group of $\log_2(L)$ bits and converts the bits into the appropriate quantization level, and then into a signal level between –1 and 1. The math script uses matrix and vector operations instead of for loops to optimize run time and computational efficiency. This signal is then passed to the DTMF Decoder VI.

2.5. Sub_DTMF_Decoder.vi

The DTMF decoder is build within a MathScript block and has the following main processing steps:

1. Separate the signal into high band and low band using two FIR filters (designed using the MATLAB FDAtool). This simplifies the logic needed to find the two frequencies since only one need to be found in each band.
2. Create non-overlapping windows L=106 with each signal band. This band size represents a time of 13.3ms but with a frequency resolution of 75 Hz.
3. Pad the signal with zeros and calculate the necessary DFTs using the Goertzel algorithm using N=205 for the fundamental frequencies and N=201 for the harmonic frequencies.
4. Calculate the max and average signal power in each window for each band.
5. Check that the max power is greater than the set threshold (1×10^{-3}) and that the power ratios (max/avg) and harmonic to fundamental power ratio are greater than the set threshold (2).
6. Test if both a frequency from the column and a frequency from the row passed the previous test for a window then apply duration and interruption rules to form digit string (included adding a mode function to the labVIEW project; shown in Appendix A).

The digit string is then formatted as a phone number, with the necessary dashes, using labVIEW VI blocks depending on length of the digit string. The algorithm can

handles signals with lengths 5,7,10, and 11. The flowchart for this subVI can be seen in Figure 2.

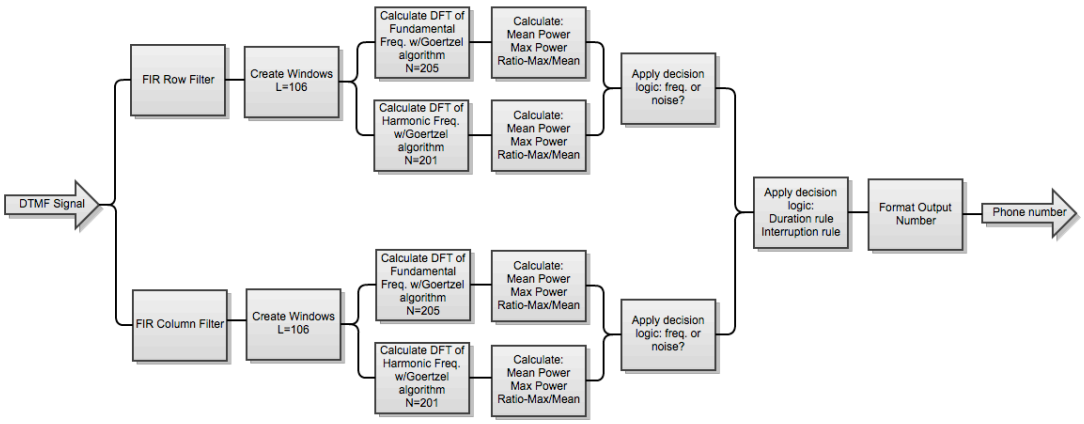

Figure 2. DTMF decoder VI flowchart.

3. Added Features

The team added the ability for the DTMF Packet Receiver VI to attempt to decode the signal before 100% of the packets are received. The logic for the choose statement which decodes runs the DTMF decoder was changed from checking to ensure that all packets were received to checking the AND of all of the logic below:

- Ensure that the percentage of packets received is greater than the set threshold.
- Ensure that an addition packet was received since the last loop iteration.
- Check the mod 2 of the number of packets received and return true if the mod 2 is zero.

The loop will break as soon as the received phone number string matches the phone number input to the receiver.

4. Troubleshooting

The team encountered a number of difficulties when connecting the USRP devices to the computers. The two MacBook Pro computers running Windows 7 through Parallels virtualization software required a very particular software setup in order for the USRP to communicate properly with labVIEW. The static IP address (192.168.10.1) had to be set on the Ethernet settings in the mac system preferences dialog and the network settings for the virtual machine had to be set to shared network. It was possible to test the connection without the use of the included utility by pining the known IP address of the device form either mac terminal or windows command prompt. The one windows computer did not have a Gigabit Ethernet adapter and thus was unable to communicate with the USRP device.

General troubleshooting of the various labVIEW VIs was accomplished using the Highlight Execution tool as well as the probe tool.

5. Recorded Measurements

5.1. Overview of Test Runs

The performance of the code was tested through two different test cases displaying two different environments. The first environment had the USRP devices right next to each, while the second environment had the USRP devices at a sizeable distance. Each test case measured the percentage of packets required to decode the signal and whether or not the program received an error across different constellation sizes and quantization levels. Each combination of constellation size and quantization

level was tested four times. The average percentage of packets from each combination was taken into consideration when determining the effectiveness of the program. The decoder aspect of the program proved to be accurate in decoding the signal in the environment the test was conducted, so the only error taken into account during the trials were the possible error that could have occurred when the receiver could not obtain all the packets, which will be labeled as a packet error in the following tables.

5.2. Test Case 1

Figure 3. Test case one transmitter/receiver positioning.

Table 1. Results from Test Case One

Test Case 1 Measurements						
BPSK Measurements						
Transmiter Gain:	**30dB**					
Receiver Gain:	**20dB**					
Frequency:	**1GHz**					
Quantization Level	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Average % of Packets Required**	**Error Present**
2	82%	94%	85%	94%	88.75%	None
4	98%	100%	100%	100%	99.50%	None
8	91%	98%	94%	94%	94.25%	None
16	100%	100%	100%	98%	99.50%	None
QPSK Measurements						
Quantization Level	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Average % of Packets Required**	**Error Present**
2	88%	83%	100%	94%	91.25%	None
4	98%	95%	52%	81%	81.50%	None
8	52%	52%	52%	52%	52.00%	None
16	93%	98%	100%	89%	95.00%	None
8PSK Measurements						
Quantization Level	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Average % of Packets Required**	**Error Present**
2	Packet Err	Packet Err	Packet Err	Packet Err	N/A	Packet Error
4	85%	74%	74%	66%	74.75%	None
8	95%	68%	98%	81%	85.50%	None
16	46%	53%	52%	50%	50.25%	None

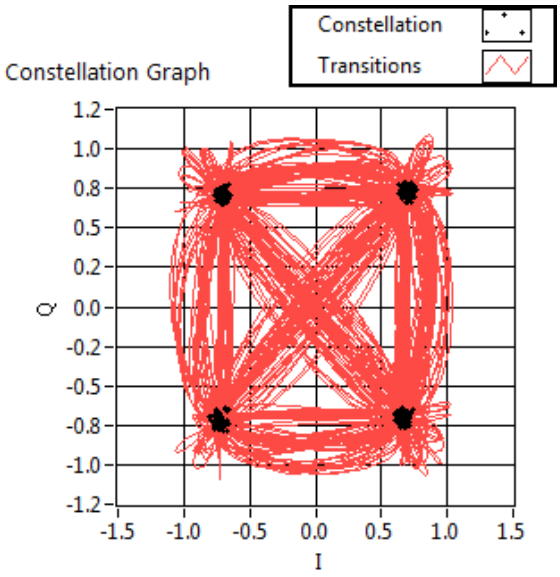

Figure 4. An example constelation graph from QPSK encoding.

5.3. Test Case 2

Figure 5. Test case two transmitter/receiver positioning.

Table 2. Results from Test Case Two

Test Case 2 Measurements						
Transmiter Gain:	**30dB**					
Receiver Gain:	**20dB**					
Frequency:	**567MHz**					
BPSK Measurements						
Quantization Level	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Average % of Packets Required**	**Error Present**
2	88%	83%	100%	83%	88.50%	None
4	83%	92%	92%	89%	89.00%	None
8	98%	83%	92%	92%	91.25%	None
16	92%	99%	99%	92%	95.50%	None
QPSK Measurements						
Quantization Level	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Average % of Packets Required**	**Error Present**
2	Packet Err	Packet Err	Packet Err	Packet Err	N/A	Packet Error
4	97%	47%	50%	47%	60.25%	None
8	52%	52%	52%	49%	51.25%	None
16	98%	100%	59%	100%	89.25%	None
8PSK Measurements						
Quantization Level	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Average % of Packets Required**	**Error Present**
2	Packet Err	Packet Err	Packet Err	Packet Err	N/A	Packet Error
4	87%	100%	90%	63%	85.00%	None
8	93%	69%	75%	82%	79.75%	None
16	45%	46%	47%	48%	46.50%	None

5.4. Conclusions

From the trials conducted above, the following observations were reached in regards to the program's performance. The designed transceiver performed best when it was set to QPSK at quantization level 4 and at 8PSK at quantization level 16, as it was able to decode the signal with only about 50% of the packets being received at the receiver end. Both test runs had similar results, so distance between devices did not prove to be a major issue. However, the program was prone to fail at receiving packets which prevented it from completing the decode process. This packet error occurred during some constellation sizes at quantization level 2. This error most likely occurred because of the limitations of the laptops and the USRP used, as the data processed by the laptops were properly sent or received by the hardware. Overall, our program was successful in decoding in several cases and only faltered due to hardware limitations.

Appendix A: Code Listing

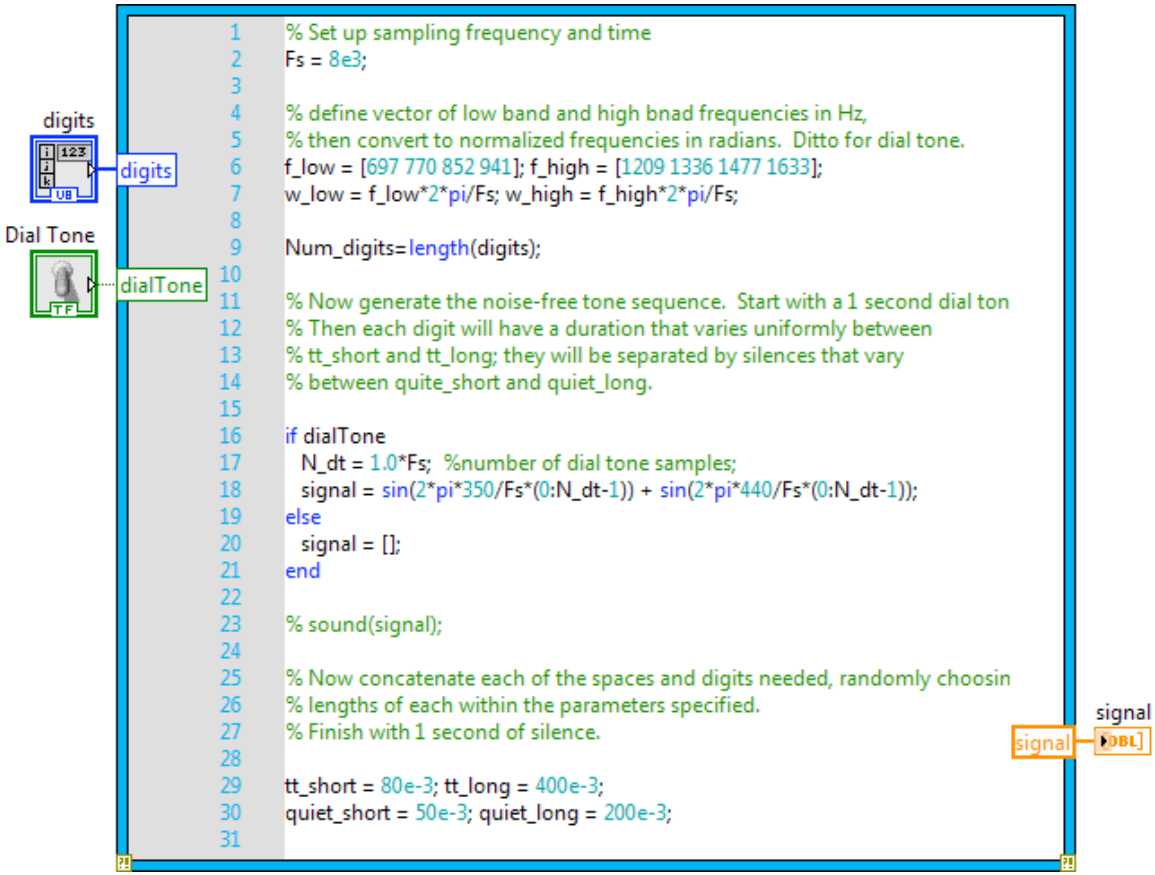

Figure 6. Sub_DTMF_Encoder.vi.

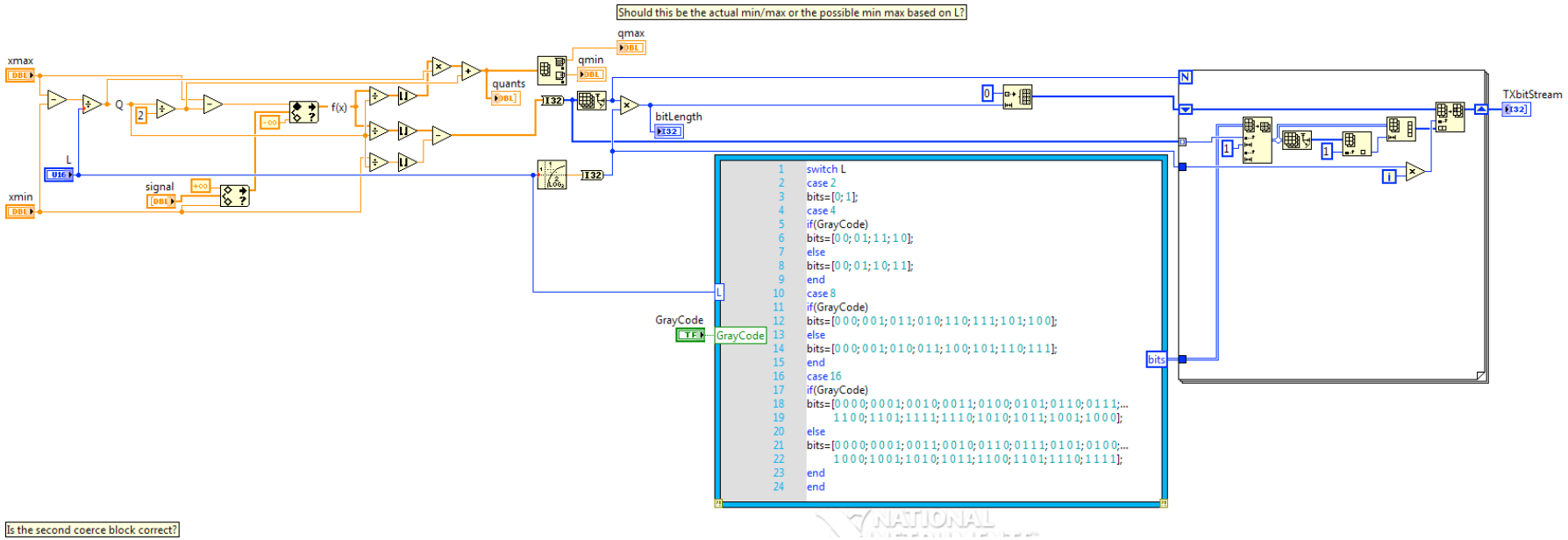

Figure 7. Sub_DTMF_Quantize_BitEncoding VI file.

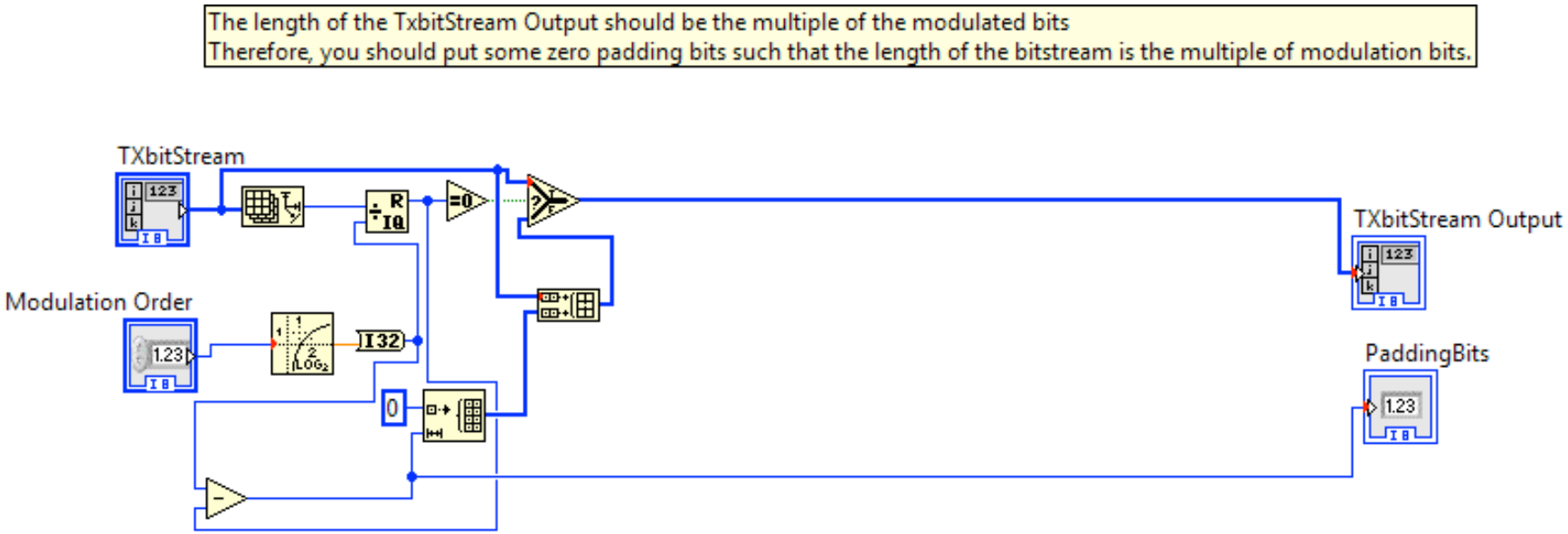

Figure 8. Sub_DTMF_BitPadding VI.

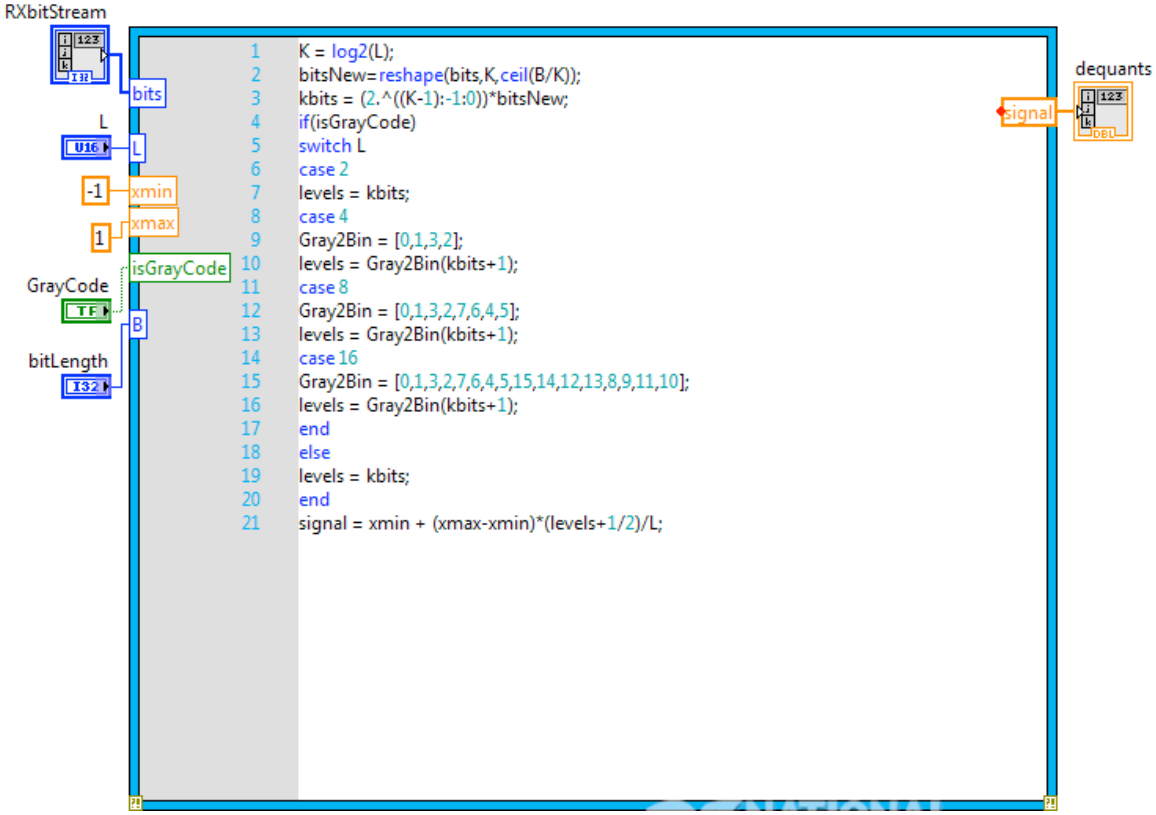

Figure 9. Sub_DTMF_Dequantize_bitDecoding VI.

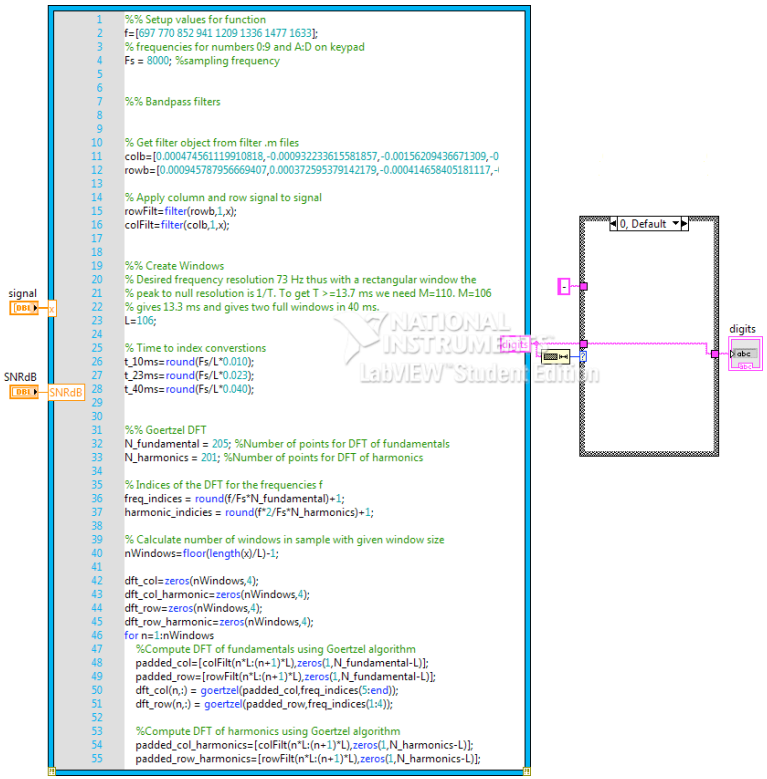

Figure 11. Sub_DTMF_Decoder VI.

```
function mode = mode(x)
X = sort(x); indices   =  find(diff([X realmax]) > 0); % indices where repeated values change
[modeL,i] =  max (diff([0 indices]));     % longest persistence length of repeated values
mode      =  X(indices(i));
```

Figure 10. Mode function implemented as a MathScript Function.

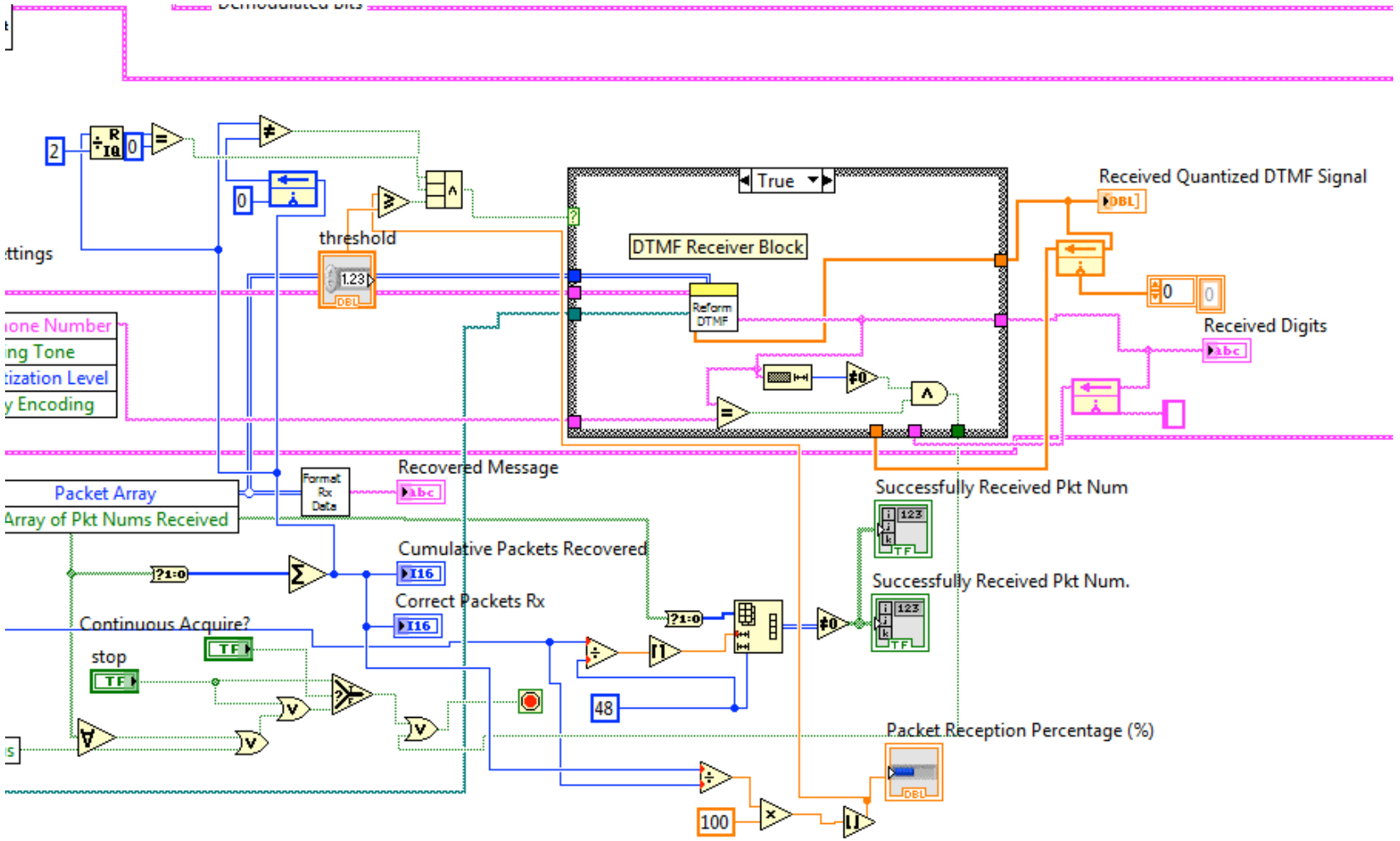

Figure 12. Modified section of DTMF Packet Receiver VI which attempts to decode DTMF signal when certain conditions are met.