MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EE430 PROJECT REPORT – PART II

Frequency-Hopping Spread Spectrum (FHSS)

Student Names: Oğuzhan Oğuz - Sina Zehtaban

Student IDs: 2516631 - 2549566

1. Introduction

Nowadays, the secure and efficient transmission of data has become a basic requirement in communication systems. In particular, the resistance of data to interference and jamming has become very critical. Frequency-Hopping Spread Spectrum (FHSS) is a communication technique that aims to achieve these goals by randomly varying the carrier frequency of the signal among a predefined set of frequencies. This process increases the resistance of the signal to noise by spreading its energy over a wider bandwidth, making it more difficult to jam or interfere with the signal.

This project focuses on the implementation of FHSS communication system consisting of a transmitter and a receiver. The transmitter encodes a text message entered by the user into digital data. The signal is modulated using M-ary Frequency Shift Keying (M-FSK) and the data is transmitted over M-FSKs. The receiver records the transmitted signal. It performs time-frequency analysis and decodes the embedded message. The system is designed to support three different levels of M-FSK.

The report includes the construction stages and the results obtained during the project process.

2. Transmitter Implementation

The transmitter part is responsible for transmitting the written message according to the data entered by the user. The transmitting process is carried out as a result of the processes mentioned below. The results are displayed through spectrograms.

2.1 Text Message Encoding

- Conversion to Binary: The input text is converted to binary using ASCII values.
- Grouping for M-FSK: Binary data is grouped based on the modulation order (M) selected from the category also there exist a padding to ensures compatibility with the selected M.

Binary data is grouped into symbols according to the selected category.

• Category 1 (M= 2): 1 bit per symbol
• Category 2 (M= 4): 2 bits per symbol
• Category 4 (M= 8): 3 bits per symbol

Each grouped bit sequence is mapped to a modulation index (m_k) using a category-specific table. For example, for category 2, let's examine the data 0101001101101001. The data is grouped as [01, 01, 00, 11, 01, 10, 10, 01]. The mapping is done as

• 00 -> -2
• 01 -> -1
• 10 -> 1
• 11 -> 2

In other words, the mapping is done as [-1, -1, -2, 2, -1, 1, 1, -1]. In addition, the length of the binary data is padded with zeros so that the bits can be grouped exactly with symbols. This is called padding. For example, the data 0101001101101001 before padding is converted to 010100110110100100 for category 2.

As a result, the text message is converted into a binary format that can be used in the FHSS system. This process works in harmony with the subsequent modulation and frequency hopping steps, allowing the transmitter to produce valid signals in each category.

2.2 Frequency Hopping

A pseudo-random number generator determines the hop frequencies based on a frequency table. Frequency tables were generated using student number (i.e. $N_1N_2N_3N_4N_5N_6N_7$), following the rules for each category.

• **Category 1**: ’row 1’ = [1000+bS, 1500+bS, 2000+bS, 2500+bS, 3000+bS, 3500+bS, 4000+bS, 4500+bS, 5000+bS]

where b = 1 if N_6 is even, b = −1 otherwise. S = 100 · (N_7 (mod5))

• **Category 2**: ’row 2’ = [1500 + bS, 2500 + bS, 3500 + bS, 4500 + bS, 5500 + bS, 6500 + bS, 7500 + bS, 8500 + bS]

where b = 1 if N_5 is even, b = −1 otherwise. S = 100 · N_7.

• **Category 3**: ’row 3’ = [1000 + S, 3000 + S, 5000 + S, 7000 + S, 9000 + S, 11000 + S] where S = 100 · N_7.

2.3 Signal Generation

The FHSS signal is created by modulating the hop frequencies using M-FSK. The modulation index (Δf) and hop duration (T_h) are category-dependent. The FHSS signal is scaled to prevent clipping and played as an audio signal.

According to the category selected by the user, values such as sample frequency (f_s), hopping period (T_h), frequency sequence, modulation index (m_k) are assigned. Then, frequency hopping process explained in section 2.2 is performed. The carrier frequency of each hop is modulated with a frequency shift ($\Delta f \cdot m_k$) depending on the data. Modulated frequency can be found as:

$$f_{\text{modulated}} = f_k + \Delta f \cdot m_k \quad \text{(eqn. 2.1)}$$

This process integrates both the carrier frequency and the data into the signal at the same time. During each hoping period, a sinusoidal waveform is generated based on the modulated frequency:

$$x(t)=\sin(2\pi \cdot f_{\text{modulated}} \cdot t) \quad \text{(eqn. 2.2)}$$

Each hopping signal generated is added to the total signal to form a complete FHSS signal. Mathematical representation of a frequency hopping signal is:

$$x(t) = \sum_k \mathrm{rect}_{T_h}(\mathrm{t} - \mathrm{kT}_h)\exp\left(\mathrm{j}(2\pi(\mathrm{f}_k + \Delta\,\mathrm{fm}_k)(\mathrm{t} - \mathrm{kT}_h)\right) \text{ (eqn. 2.3)}$$

The generated signal is then normalized so that its amplitude is in the range of -1.1. This is done to avoid clipping problems. When processing complex signals, using only the sine or cosine component significantly reduces the computational load. Therefore, only the sine signal is used in the solution. By using Euler theorem:

$$\exp(j\theta) = \cos(\theta) + j\sin(\theta) \text{ (eqn. 2.4)}$$

By using eqn. 2.4:

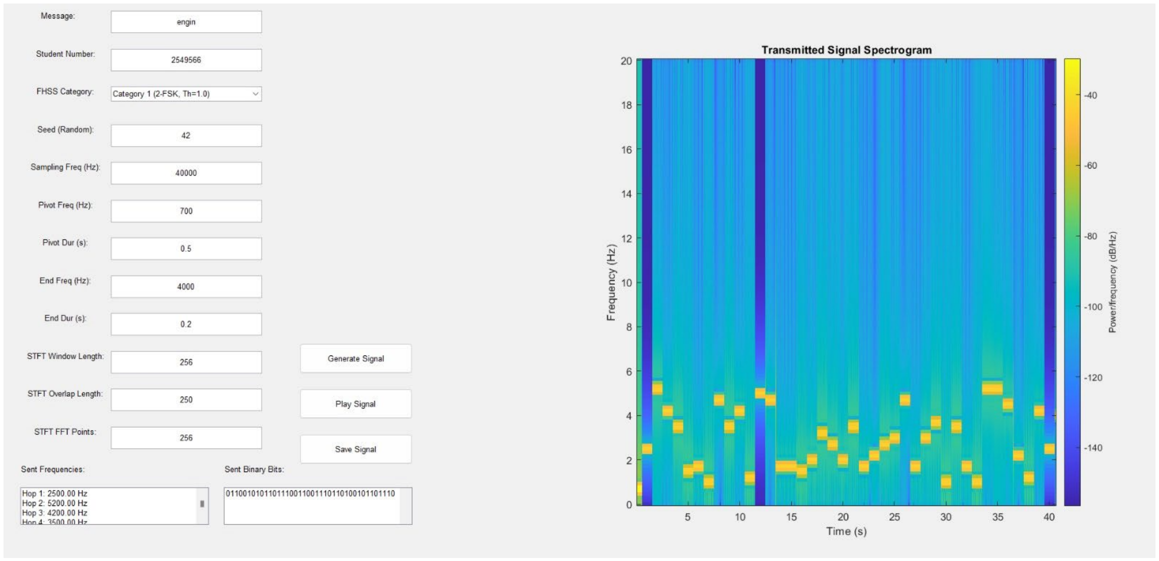
$$\exp\left(\mathrm{j}\big(2\pi(\mathrm{f}_k + \Delta\,\mathrm{fm}_k)(\mathrm{t} - \mathrm{kT}_h)\big)\right) => \sin\big(2\pi(f_k + \Delta f m_k)(t - kT_h)\big) \text{ (eqn. 2.5)}$$

Finally:

$$x(t) = \sum_k \sin\big(2\pi(f_k + \Delta f m_k)(t - kT_h)\big)\ \mathrm{rect}_{T_h}(\mathrm{t} - \mathrm{kT}_h) \text{ (eqn. 2.6)}$$

The expression in eqn. 2.6 was obtained using the signal in eqn. 2.2.

• Category 1 Signal Generation



Message:	engin
Student Number:	2549566
FHSS Category:	Category 1 (2-FSK, Th=1.0)
Seed (Random):	42
Sampling Freq (Hz):	40000
Pivot Freq (Hz):	700
Pivot Dur (s):	0.5
End Freq (Hz):	4000
End Dur (s):	0.2
STFT Window Length:	256
STFT Overlap Length:	250
STFT FFT Points:	256

Sent Frequencies:

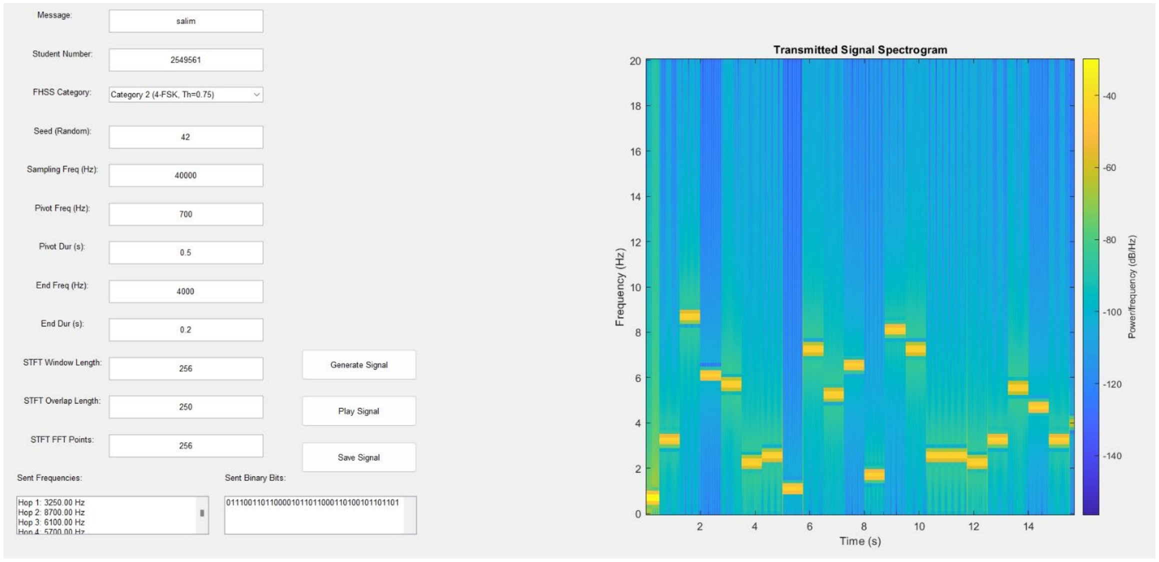
```
Hop 1: 2500.00 Hz
Hop 2: 5200.00 Hz
Hop 3: 4200.00 Hz
Hop 4: 3500.00 Hz
```

Sent Binary Bits:

```
0110010101101110011001110110100101101110
```

Figure 1. Category 1 signal

• Category 2 Signal Generation



Message:	salim
Student Number:	2549561
FHSS Category:	Category 2 (4-FSK, Th=0.75)
Seed (Random):	42
Sampling Freq (Hz):	40000
Pivot Freq (Hz):	700
Pivot Dur (s):	0.5
End Freq (Hz):	4000
End Dur (s):	0.2
STFT Window Length:	256
STFT Overlap Length:	250
STFT FFT Points:	256

Sent Frequencies:

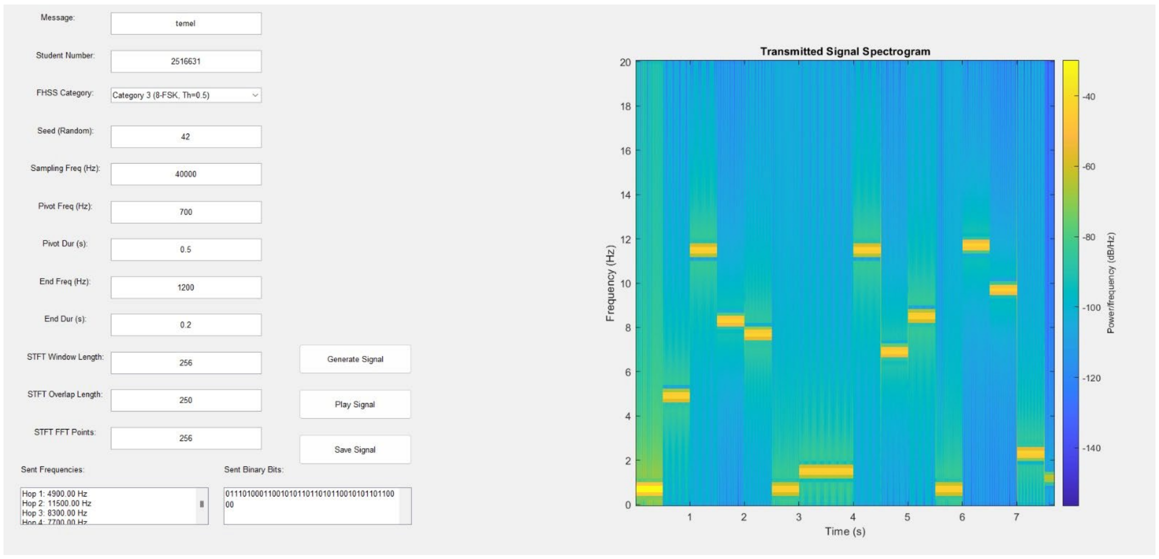
```
Hop 1: 3250.00 Hz
Hop 2: 8700.00 Hz
Hop 3: 6100.00 Hz
Hop 4: 5700.00 Hz
```

Sent Binary Bits:

```
0111001101100001011011000110100101101101
```

Figure 2. Category 2 signal

• Category 3 Signal Generation



Message:	temel
Student Number:	2516631
FHSS Category:	Category 3 (8-FSK, Th=0.5)
Seed (Random):	42
Sampling Freq (Hz):	40000
Pivot Freq (Hz):	700
Pivot Dur (s):	0.5
End Freq (Hz):	1200
End Dur (s):	0.2
STFT Window Length:	256
STFT Overlap Length:	250
STFT FFT Points:	256

Sent Frequencies:

```
Hop 1: 4900.00 Hz
Hop 2: 11500.00 Hz
Hop 3: 8300.00 Hz
Hop 4: 7700.00 Hz
```

Sent Binary Bits:

```
011101000110010101101101011001010110110000
```

Figure 3. Category 3 signal

The generated signals can be seen in Figures 1, 2 and 3. The results seen in the figures were obtained for categories 1,2,3 with the words "engin","salim" ,and "temel" and the student numbers "2549566","2549961","2516631" respectively. It can be observed that signals that meet the desired values are generated.

2.4 Spectrograms

A spectrogram is a basic visualization tool used in time-frequency analysis. It is used in systems such as FHSS (Frequency Hopping Spread Spectrum) to show how the frequency components of a signal change with time. In this project, spectrograms for the FHSS signal were generated using the spectrogram function of MATLAB.

First, the signal is divided into segments of fixed length (N_w) and a certain overlap length (N_o). The frequency content of the signal is assumed to be constant within each segment. Then, the short-time Fourier transform (STFT) is applied to each window. This process calculates the frequency spectrum of that window. Finally, the spectra of all windows are combined along the time and frequency axes. This gives the following values:

• x-axis: Time (center of each window)

• y-axis: Frequency

• z-axis (color): Amplitude (in dB, showing how intense each frequency is at a given time)

An example plot can be seen in Figure 4.

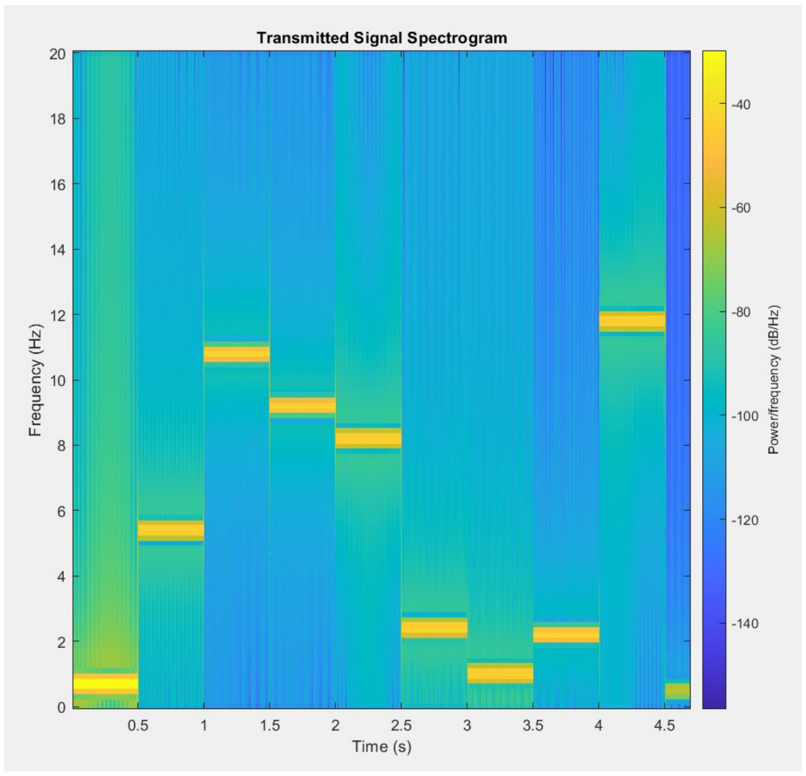


Figure 4. A STFT plot example [1]

$$X(t,f) = \int_{-\infty}^{\infty} x(\tau)w(\tau - t)\exp(-j2\pi f\tau)d\tau \quad (\text{eqn. } 2.7)$$

Where the w(t) is window function. The STFT of the function x(t) can be found using Eqn. 2.7.

2.4.1 Parameters Used in Time-Frequency Analysis and Their Effects

• **Window Length (N_w):** The signal is divided into segments of length N_w. The window length determines the time interval of each analysis. A longer window provides better frequency resolution but lower time resolution. In other words, there is a trade-off between frequency resolution and time resolution. Frequency resolution can be found with the following equation:

$$f_{resolution} = f_s / N_w \quad \text{(eqn. 2.8)}$$

• **Overlap Length (N_o):** A value that indicates how much two consecutive windows overlap each other. High overlap results in less interruption and smoother transitions, and provides more accurate amplitude information in the time axis.

• **FFT Points (N_{FFT}):** Determines how many data points each window will represent on the frequency axis. Higher N_{FFT} provides finer frequency resolution but has higher computational cost, meaning more processing power or time is required to perform the operation. N_{FFT} is expressed mathematically as:

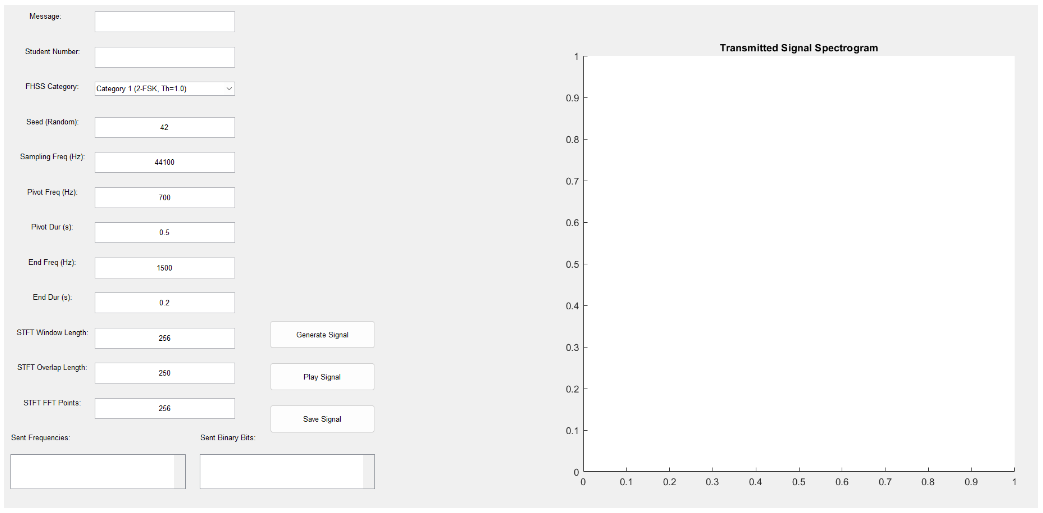
$$N_{FFT} = f_s / f_{resolution} \quad \text{(eqn. 2.9)}$$

$N_{FFT} \geq N_W$. Usually N_{FFT} is a power of 2.

• **Sampling Frequency (f_s):** Determines the maximum frequency that the spectrogram can analyze. Higher f_s allows for the analysis of higher frequency components. If the signal contains a frequency greater than the Nyquist frequency, $f_s/2$, aliasing occurs.

2.5 GUI for Transmitter

The Graphical User Interface (GUI) is designed to create and visualize FHSS signals with user parameters. In Figure 5, the GUI created for the transmitter can be seen.



Message:	
Student Number:	
FHSS Category:	Category 1 (2-FSK, Th=1.0)
Seed (Random):	42
Sampling Freq (Hz):	44100
Pivot Freq (Hz):	700
Pivot Dur (s):	0.5
End Freq (Hz):	1500
End Dur (s):	0.2
STFT Window Length:	256
STFT Overlap Length:	250
STFT FFT Points:	256
Sent Frequencies:	
Sent Binary Bits:	

Figure 5. Transmitter GUI

First of all, there is a text message section in the GUI. In this section, the user enters the message he wants to be transmitted.

Then, the category section, which can be seen in Figure 6, was created. It allows selection between categories 1, 2 and 3. A drop-down menu was created for this purpose ,also a function was used to dynamically update the parameters according to the selected category.

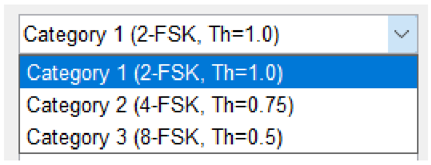

Category 1 (2-FSK, Th=1.0)
Category 1 (2-FSK, Th=1.0)
Category 2 (4-FSK, Th=0.75)
Category 3 (8-FSK, Th=0.5)

Figure 6. Category part of GUI

Apart from this, "student number", "sampling frequency", “seed", "STFT window length", "STFT overlap length", ", STFT FFT points" sections were created. The user's preferred parameters were taken with these sections. Numerical and text input fields were created using the uieditfield component.

For the spectrogram panel, an axes window was created using the uiaxes component. The imagesc function was used to visualize the spectrogram of the signal. The spectogram function was used to analyze the frequency components of the FHSS signal. An example can be seen in Figure 7.

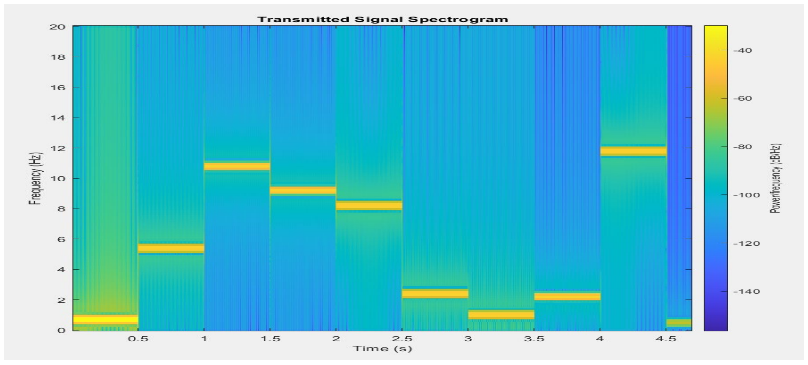


Figure 7. Spectogram part of GUI

Due to the time delay between the transmitted and received signal, we also defined a frequency called pivot frequency which will appear at the beginning and the end of the transmitted signal and receiver will be able to detect and decode the exact signal transmitted without losing any part of it. In order to allow user to adjust parameters related to this part , we also created related parts for them as it can be seen in the GUI.

Additionally, two boxes are defined: one displays the transmitted frequencies generated for each hop along with their corresponding hop numbers, and the other shows the sequence of transmitted binary bits.

Finally, 3 buttons were created to generate signal, play the signal , and save the signal as an wav file in case of necessity for debugging our system and investigate its performance.

3. Receiver Implementation

The receiver is also implemented as a MATLAB GUI. It records the transmitted signal and decodes it back into text.

3.1 Signal Recording

The GUI records the transmitted signal using the PC's microphone. Users can set the recording duration and sampling rate. Listening operations are performed with the Audiorecorder and recordblocking functions.

3.2 Noise Reduction

Several methods were used to reduce noise in the project. These methods include:

- Bandpass Filtering:

In the receiver code, a bandpass filter is used around the frequencies of interest. This cuts off very low frequencies below 300 Hz and very high frequencies above 15 kHz. This reduces noise by removing frequency content that does not belong to the transmitted message. It makes the main interest band clearer for hop detection. However, it also has a few disadvantages. If the hop frequencies are close to the selected upper limit, these frequencies can be attenuated.

- Band Stop Filter Around Pivot:

In the receiver code, after the hop region is clipped, a band-stop filter is applied. This suppresses the frequency of about 700 Hz (the pivot frequency is 700 Hz). This significantly reduces the residual of the pivot and does not affect the first hop pivot frequency. The disadvantage is that if the hop signal is designed to contain 700 Hz, it will suppress the real signal. To prevent this, the pivot frequency can be changed to an appropriate value.

- Skipping Edges in Each Hop:

In the hop detection loop, a certain number of samples are skipped from the beginning and end of each hop. A value of 3 kHz is usually used for this. When each new hop starts, the modulator switches from one frequency to another. Therefore, there may be parts of the signal that are not supposed to be at the beginning and end of each hop. By skipping the edge samples, the focus is on the stationary sine wave in the middle of the hop, which improves FFT peak detection. The disadvantages are that some of the total data of the hop is lost. Therefore, it can cause problems with short hops. Also, if the noise level is high and the edges are not well defined, a more efficient skipping may be required.

- Guard Time after Pivot:

A small guard period is added after the end index of the pivot signal. This ensures that the first real hop is not detected as the pivot signal in the FFT. If too much time is skipped, the first hop signal may be missed. Not very likely. The guard period should be long enough to eliminate the pivot signal, but short enough not to lose hop data.

Thanks to all these applied techniques, noise is reduced as much as possible before the decoding process.

3.3 Frequency Hop Tracking

In the FHSS technique, each hop is shaped around a certain fundamental frequency f_{base}. M-FSK is added to this. For this, the fundamental frequency is first created. A root frequency sequence was created using certain digits of the student number and some intervals. The 5th, 6th and 7th digits of the student number are important. For category 1, if the value of N_6 is even, b = 1, otherwise, b = -1. S = 100 * (N_7 %5). And row 1 for category 1:

’row 1’= [1000+bS, 1500+bS, 2000+bS, 2500+bS, 3000+bS, 3500+bS, 4000+bS, 4500+bS, 5000+bS]

For category 2, if the value of N_5 is even, b = 1, otherwise, b = -1. S = 100*N_7 and row2:

’row 2’= [1500 + bS, 2500 + bS, 3500 + bS, 4500 + bS, 5500 + bS, 6500 + bS, 7500 + bS, 8500 + bS]

For Category 3, S = 100*N_7 and row3:

’row 3’= [1000 + S, 3000 + S, 5000 + S, 7000 + S, 9000 + S, 11000 + S] where S = 100* N_7. As a result, a sequence like {1000,1500,…}+b×S is formed. This sequence contains the fundamental frequencies that the transmitter can “select”. During each hop, the transmitter selects a fundamental frequency. It does this by randomly selecting a certain element of the freqArr array. At the same time, it adds an offset while encoding the data to be transmitted in each hop as 2-FSK, 4-FSK or 8-FSK. For example, if M=8, m_k values will be [−4,−3,−2,−1,+1,+2,+3,+4]. In each hop:

$$f_{hop} = f_{base,hop} + m_k * \Delta f$$

Here, Δf and m_k are the frequency shifts representing the data. This allows the transmitter to determine both which fundamental frequency it jumps to and which combination it transmits. At this frequency, the transmitter generates a $\sin(2\pi f_{hop} t)$ wave and sends it for the hop period T_h. To add a pivot, the f_{pivot} wave is placed at the beginning, to add an end, the f_{end} wave is placed at the end. They are combined and normalized.

In the Receiver Side:

After receiving the record, the receiver finds the pivot and end signals and examines the intervening section as hopped data.

- Pivot and End Detection:

It finds the pivot by cross-correlating with f_{pivot}; where it finds it, it says the pivot is finished here and accepts that place as the start. Similarly, end is found by cross-correlating with f_{end}, and accepts it as the end of the hopped data.

- Splitting into Hops and Edge Skipping:

Hopped divides the data into sections of Th duration. In each section, it discards some of the beginning and end; that is, it takes the fixed tone part in the middle. Because the frequency is not fully stable at the time of the hop beginning/end transition.

- Finding Frequency with FFT:

Performs FFT on the middle part of each hop and finds the strongest frequency peak:

$$f_{dominant} = \underset{f}{\operatorname{argmin}} \left| F\{ \, x_{hop(t)} \right|$$

- Near Base Frequency and M-FSK Shift Calculation:

The receiver generates freqArr as in the transmitter. Finds the nearest f_{base}:

$$f_{base} = \min_{f_i \in freqArr} |f_i - f_{dominant}|$$

The difference between $f_{dominant}$ - f_{base} is divided by Δf:

$$\tilde{m}_k = \frac{f_{dominant} - f_{base}}{\Delta f}$$

From here, it rounds $\tilde{m}_k$ to the nearest M-FSK level and finds which bit comes next.

3.4 Demodulating Signal and Finding Δf_{mk}

Δf_{mk} is a frequency offset added to the carrier frequency (f_k) during each hop. This offset is used to encode digital data. Here we comprehensively explain how Δf_{mk} is found and how signal is demodulated.

- Extracting the Hopped Segment:

As a result of the operations explained in the "Frequency Hop Tracking" section, the pivotIndex and endIndex values are found. The intermediate part is determined as hoppedSignal. Then, this signal is divided into pieces with a duration of Th. In this way, the hopped segment is obtained.

- FFT on the Midsection of Each Hop:

The beginning and end of the hop may be the transient phase of the frequency transition. To remove this effect, a certain length of the beginning and end of each hop is discarded. FFT is applied to the remaining part. The $f_{dominant}$ value is found as mentioned before.

- Matching to Nearest Base Frequency:

The f_{base} value is found as mentioned in the “Frequency Hop Tracking” section.

- M-FSK Offset and Demodulation

The $\tilde{m}_k$ value is found as mentioned in the “Frequency Hop Tracking” section. Then round $\tilde{m}_k$ to the nearest mkMap value to decode the symbol.

- Converting Symbols to Bit Sequence:

$\tilde{m}_k$ value rounded to the nearest mkMap value. For example, for M = 4, mkMapping = {-2, -1, 1, 2}. Next, the binary values corresponding to the matched mkMap values are found. For example, the value +4 is converted to the value 11. All hops become a bit string next to each other. Groups of 8 are converted to ASCII characters. Thus, the message is decoded.

3.5 Receiver GUI

In Figure 8, the GUI created for the receiver can be seen. Fields were created for the user to enter the parameters "sampling rate", "hopping period", "delta f", "record duration". The same operations explained in section 2.5 were performed for this process. Then, the category section was created as in section 2.5. And an area was created showing the resulting spectrogram. Since all operations are the same as section 2.5, they will not be explained again. An decoder result part added additionaly. It shows the decode result.

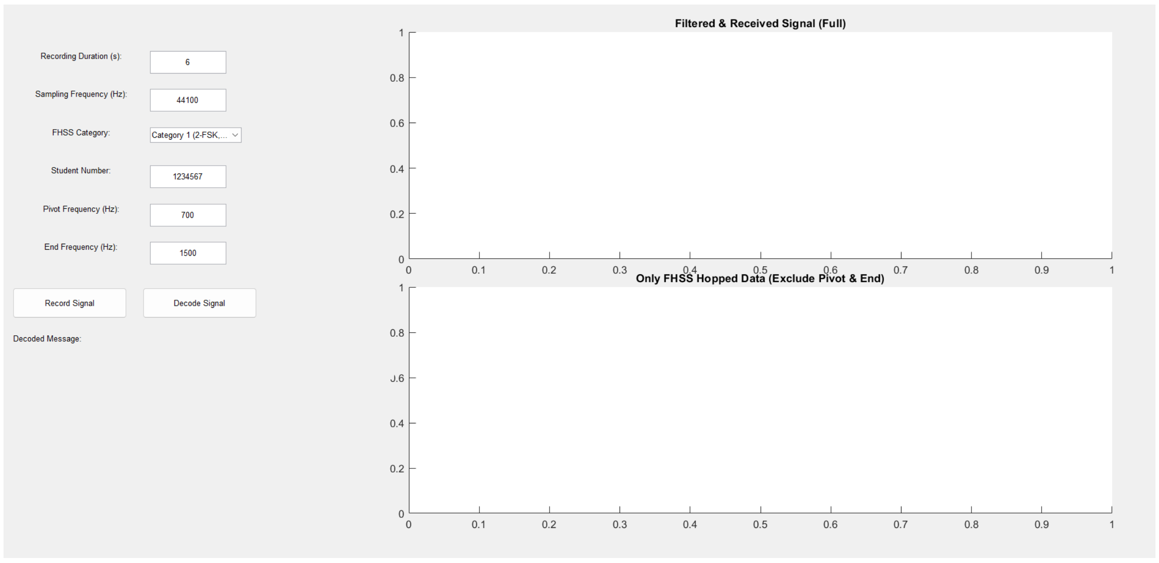

Recording Duration (s): 6
Sampling Frequency (Hz): 44100
FHSS Category: Category 1 (2-FSK,...
Student Number: 1234567
Pivot Frequency (Hz): 700
End Frequency (Hz): 1500
Decoded Message:
Record Signal
Decode Signal
Filtered & Received Signal (Full)
Only FHSS Hopped Data (Exclude Pivot & End)
0
0.2
0.4
0.6
0.8
1
0.1
0.3
0.5
0.7
0.9

Figure 8. GUI of receiver

4. Spectrograms In Receiver’s side

The spectrogram of the recorded signal is plotted. After noise reduction, the cleaned signal is shown for comparison.

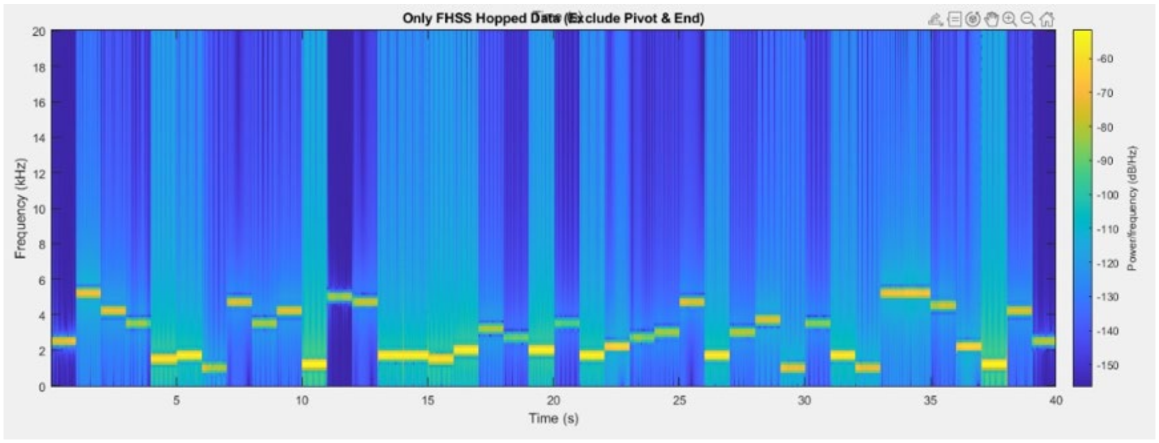


Figure 9. Receiver spectrogram plot for “engin” message in category 1

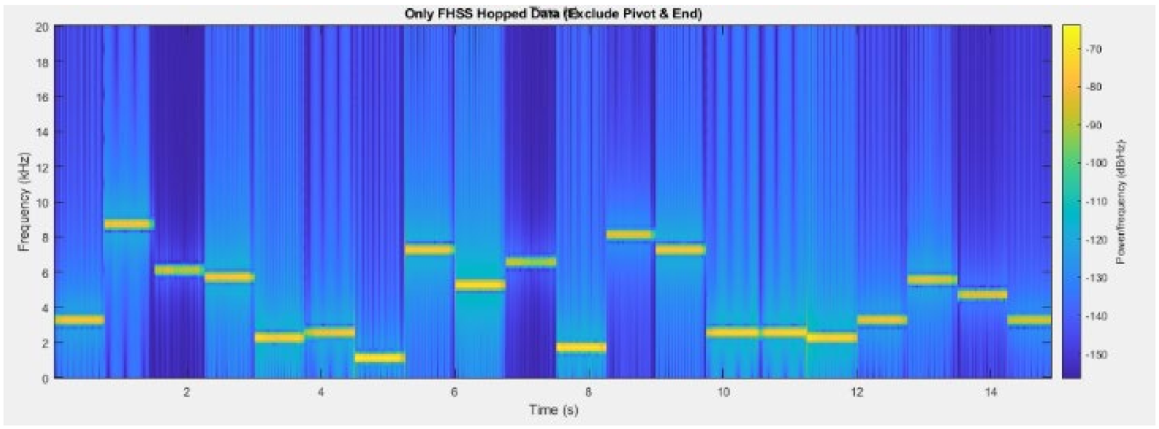


Figure 10. Receiver spectrogram plot for “salim” message in category 2

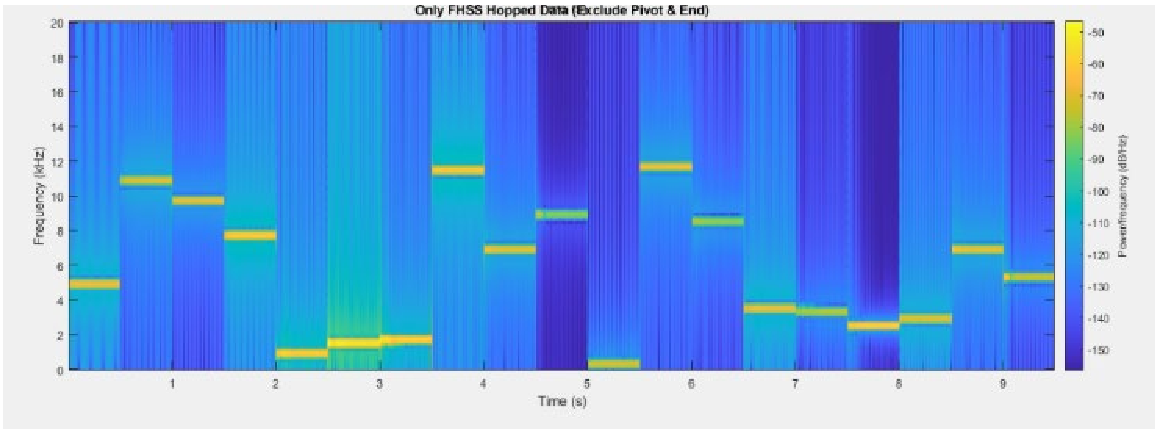
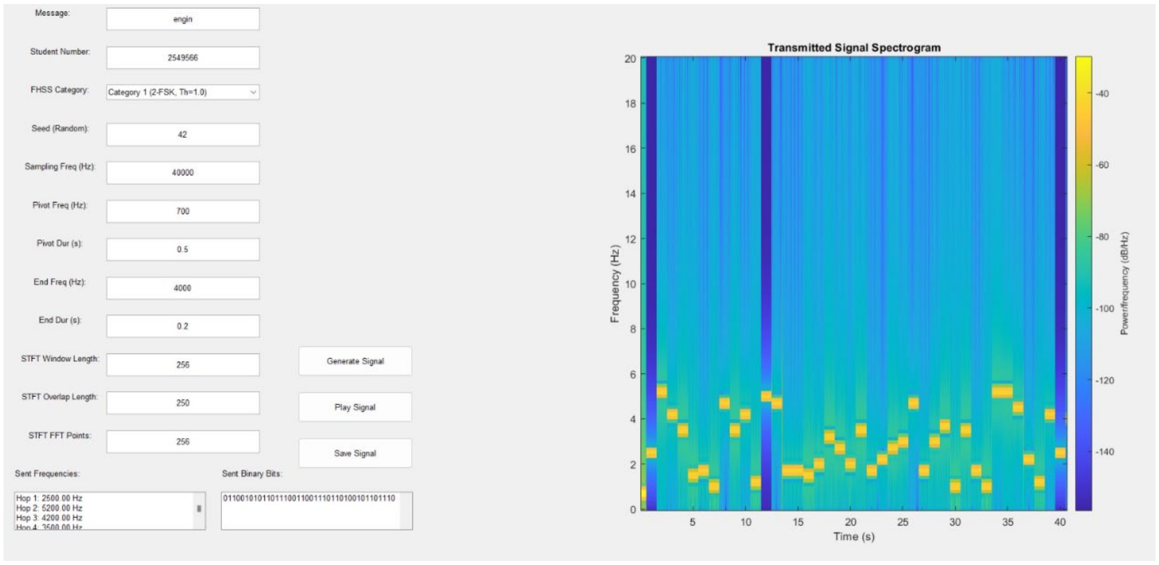


Figure 11. Receiver spectrogram plot for “ogulcan” message in category 3

5. Category 1 Test


Message:	engin
Student Number:	2549566
FHSS Category:	Category 1 (2-FSK, Th=1.0)
Seed (Random):	42
Sampling Freq (Hz):	40000
Pivot Freq (Hz):	700
Pivot Dur (s):	0.5
End Freq (Hz):	4000
End Dur (s):	0.2
STFT Window Length:	256
STFT Overlap Length:	250
STFT FFT Points:	256

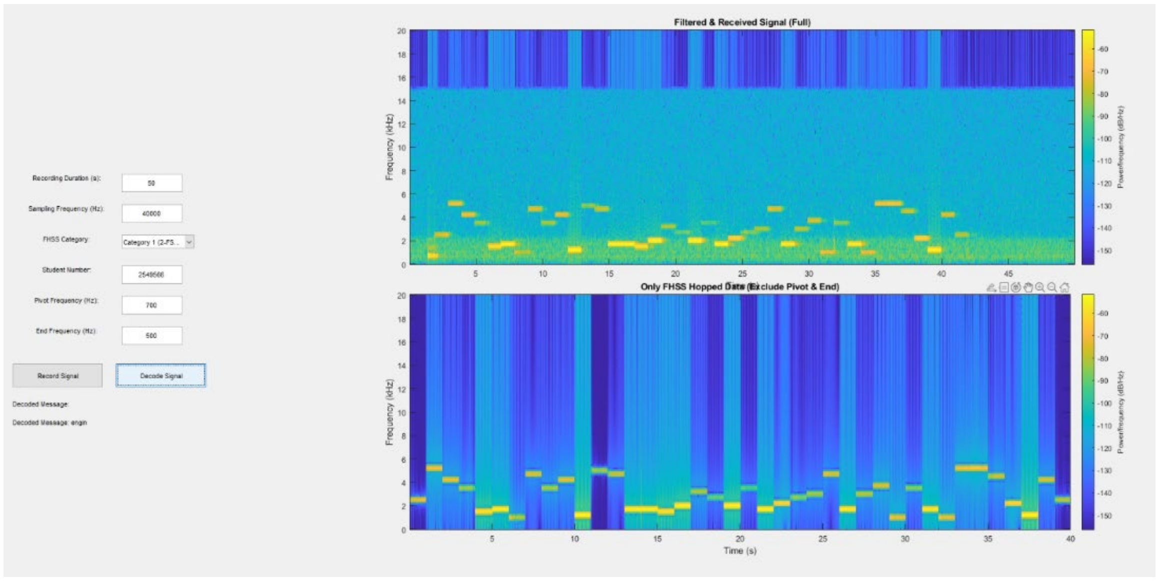
Sent Frequencies:

```
Hop 1: 2500.00 Hz
Hop 2: 5200.00 Hz
Hop 3: 4200.00 Hz
Hop 4: 3500.00 Hz
```

Sent Binary Bits:

```
0110010101101110011001110110100101101110
```

Figure 12. Transmitter GUI for “engin” message in category 1

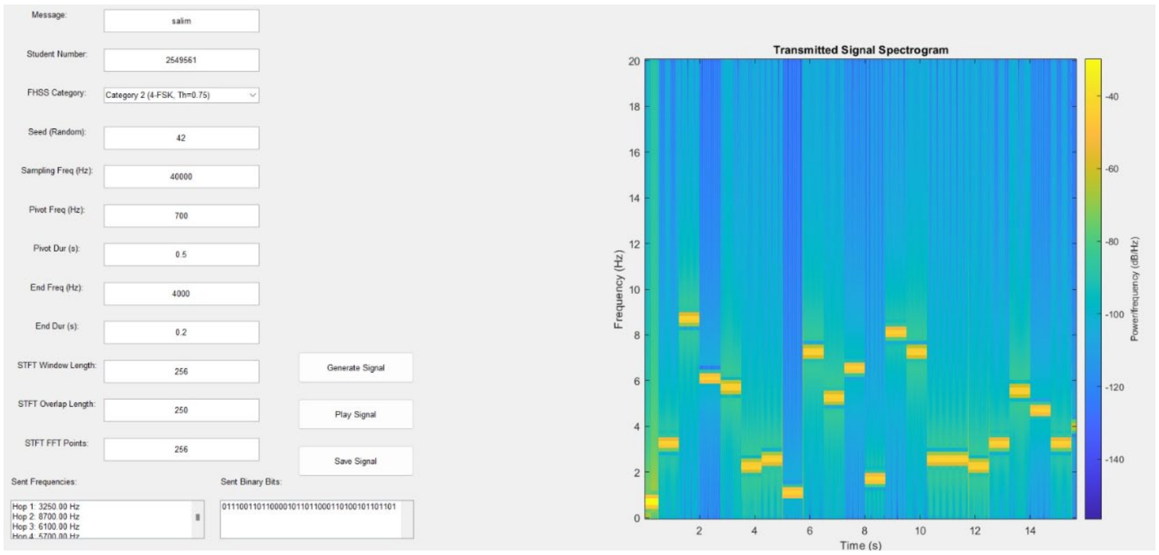

Recording Duration (s):	50
Sampling Frequency (Hz):	40000
FHSS Category:	Category 1 (2-FS...
Student Number:	2549566
Pivot Frequency (Hz):	700
End Frequency (Hz):	500

Decoded Message:

Decoded Message: engin

Record Signal
Decode Signal
Filtered & Received Signal (Full)
Only FHSS Hopped Data (Exclude Pivot & End)
Frequency (kHz)
Time (s)
Power/frequency (dB/Hz)

Figure 13. Receiver GUI for “engin” message in category 1



Message: salim

Student Number: 2549561

FHSS Category: Category 2 (4-FSK, Th=0.75)

Seed (Random): 42

Sampling Freq (Hz): 40000

Pivot Freq (Hz): 700

Pivot Dur (s): 0.5

End Freq (Hz): 4000

End Dur (s): 0.2

STFT Window Length: 256

STFT Overlap Length: 250

STFT FFT Points: 256

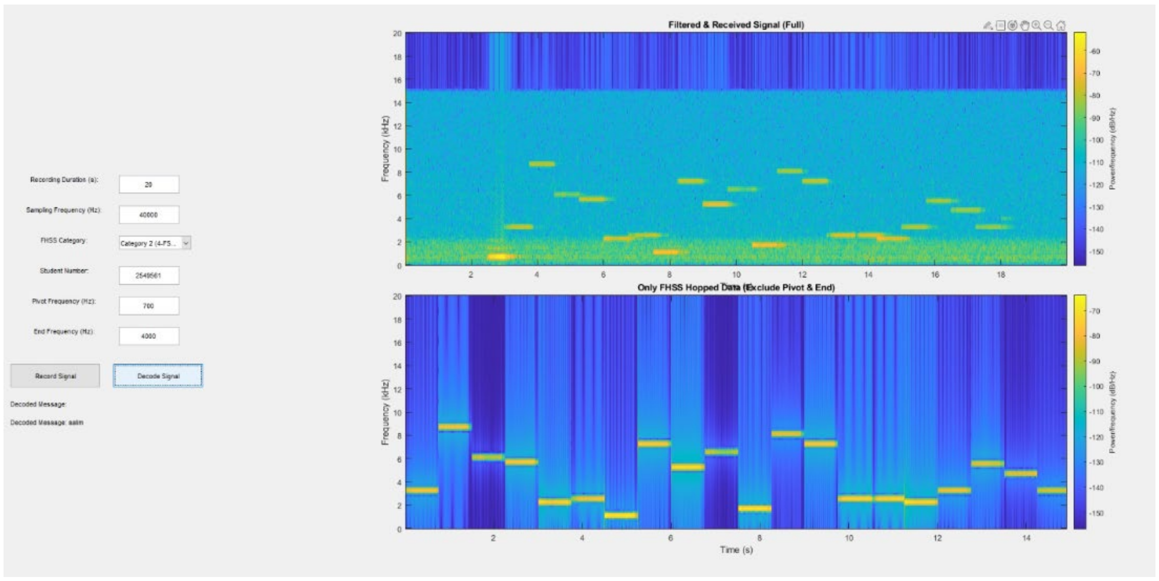
Sent Frequencies:

```
Hop 1: 3250.00 Hz
Hop 2: 8700.00 Hz
Hop 3: 6100.00 Hz
Hop 4: 5700.00 Hz
```

Sent Binary Bits:

```
01110011011000010110110001101001011101101
```

Figure 14. Transmitter GUI for "salim" message in category 2


Filtered & Received Signal (Full)
Only FHSS Hopped Data (Exclude Pivot & End)
Frequency (kHz)
Time (s)
Power/frequency (dB/Hz)
Record Signal
Decode Signal

Recording Duration (s): 20

Sampling Frequency (Hz): 40000

FHSS Category: Category 2 (4-FS...

Student Number: 2549561

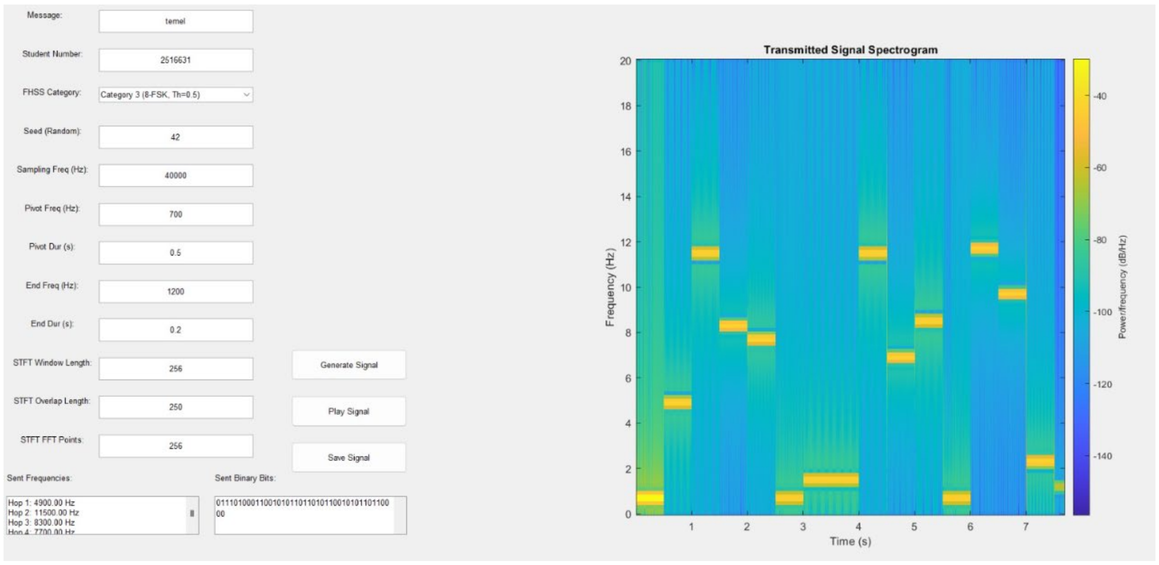
Pivot Frequency (Hz): 700

End Frequency (Hz): 4000

Decoded Message:

Decoded Message: salim

Figure 15. Receiver GUI for "salim" message in category 2



Message: temel

Student Number: 2516631

FHSS Category: Category 3 (8-FSK, Th=0.5)

Seed (Random): 42

Sampling Freq (Hz): 40000

Pivot Freq (Hz): 700

Pivot Dur (s): 0.5

End Freq (Hz): 1200

End Dur (s): 0.2

STFT Window Length: 256

STFT Overlap Length: 250

STFT FFT Points: 256

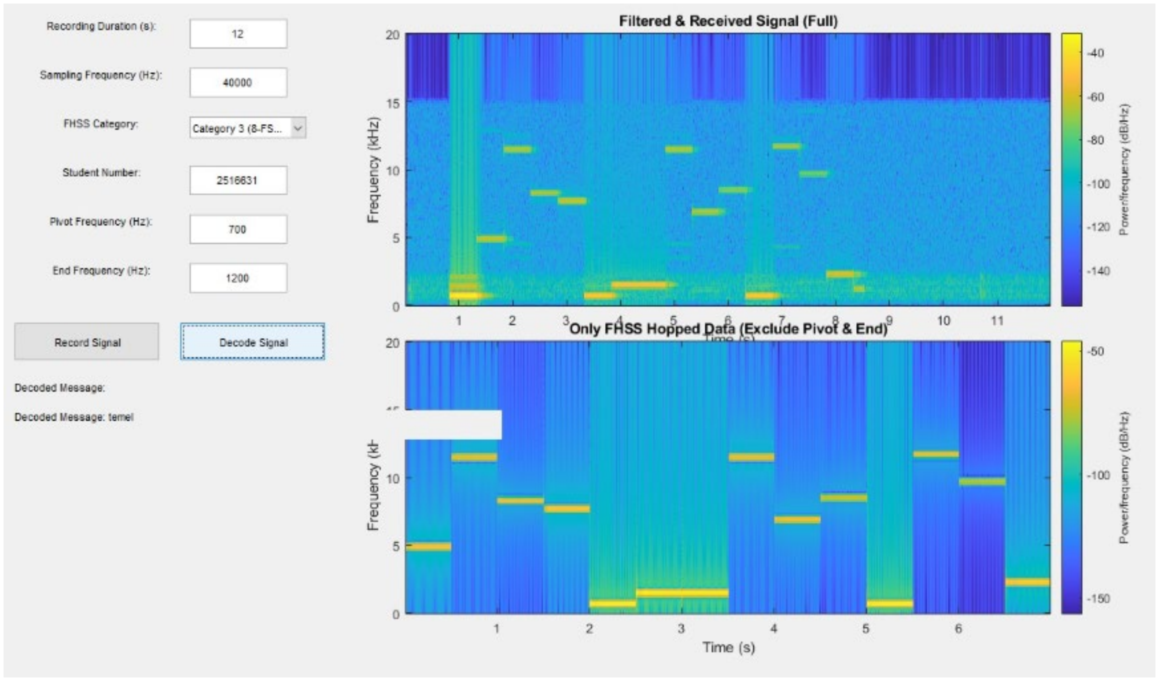
Sent Frequencies:

Hop 1: 4900.00 Hz
Hop 2: 11500.00 Hz
Hop 3: 8300.00 Hz
Hop 4: 7700.00 Hz

Sent Binary Bits:

```
0111010001100101011011010110010101101100
00
```

Figure 16. Transmitter GUI for “temel” message in category 3



Recording Duration (s): 12

Sampling Frequency (Hz): 40000

FHSS Category: Category 3 (8-FS...

Student Number: 2516631

Pivot Frequency (Hz): 700

End Frequency (Hz): 1200

Decoded Message:

Decoded Message: temel

Record Signal
Decode Signal
Filtered & Received Signal (Full)
Frequency (kHz)
Power/frequency (dB/Hz)
Only FHSS Hopped Data (Exclude Pivot & End)
Time (s)

Figure 17. Receiver GUI for “temel” message in category 3

In Figure 12-17, Three different messages with three different categories were transmitted and successfully decoded by the receiver without causing any error. Although in some cases some unexpected sounds coming from the environment caused at most 2 letters in a

6-letter message to be decoded wrongly, in 95% of the times the decoder worked without causing any error.

6. Challenges

- Synchronization: To tackle this issue that we faced in the first part of project specially for category 3, we Introduced a pivot signal to align transmitter and receiver clocks as it was comprehensively explained in its corresponding section.
- Noise Sensitivity: Due to the high error of the decoder's output, we Increased our filter's resolution and applied frequency tolerance for better detection.

7. Conclusion

Frequency-Hopping Spread Spectrum (FHSS) communication system has been successfully designed and implemented in MATLAB environment. The project starts with converting a text-based message entered by the user into digital data. The obtained digital data is modulated with frequency hopping algorithm and spread in the spectrum. The purpose of this process is to make jamming and interception more difficult. The transmitted data is demodulated by applying noise reduction on the receiver side and is decoded according to a known algorithm on both sides. Spectrograms are used to examine the signals in both transmitter and receiver sections. System parameters can be easily changed thanks to the user-friendly interface. The project provides an important infrastructure in terms of understanding, simulating and ensuring communication reliability of the basic principles of FHSS systems.

7. References

[1] Middle East Technical University, "FHSS Transmitter and Receiver Design Project," Department of Electrical and Electronics Engineering, Fall 2024. [Online]

8. Appendix

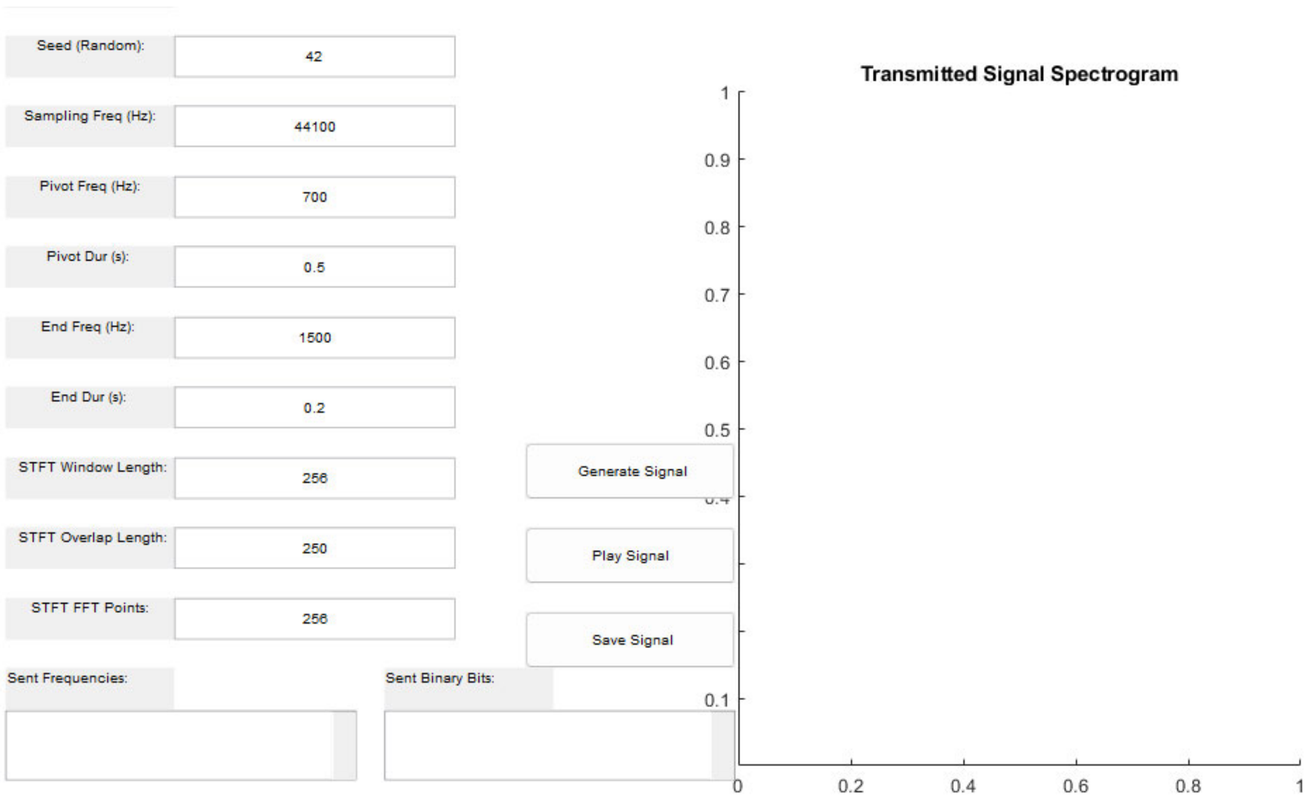
Code snippets for the transmitter and receiver are provided below:

Contents

```
function FHSS_Transmitter_GUI()
```

```
    % Create the figure for the GUI
    fig = figure('Name', 'FHSS Transmitter (Show Freqs & Bits)', ...
                 'Position', [100, 100, 1000, 600]);

    % -- UI Elements -------------------------------------------------------
     % Message
    uicontrol('Style', 'text', 'Position', [30, 700, 100, 30], 'String', 'Message:');
    msgInput = uicontrol('Style', 'edit', 'Position', [150, 700, 200, 30]);

    % Student Number
    uicontrol('Style', 'text', 'Position', [30, 650, 120, 30], 'String', 'Student Number:');
    studentNoInput = uicontrol('Style', 'edit', 'Position', [150, 650, 200, 30]);

    % FHSS Category => picks both M-FSK and hop period
    uicontrol('Style', 'text', 'Position', [30, 600, 120, 30], ...
        'String', 'FHSS Category:');
    categoryMenu = uicontrol('Style', 'popupmenu', ...
        'Position', [150, 600, 200, 30], ...
        'String', {'Category 1 (2-FSK, Th=1.0)', ...
                   'Category 2 (4-FSK, Th=0.75)', ...
                   'Category 3 (8-FSK, Th=0.5)'});

    % Seed
    uicontrol('Style', 'text', 'Position', [30, 550, 120, 30], 'String', 'Seed (Random):');
    seedInput = uicontrol('Style', 'edit', 'Position', [150, 550, 200, 30], 'String', '42');

    % Sampling Frequency
    uicontrol('Style', 'text', 'Position', [30, 500, 120, 30], 'String', 'Sampling Freq (Hz):');
    samplingFreqInput = uicontrol('Style', 'edit', 'Position', [150, 500, 200, 30], 'String', '44100');

    % Pivot freq / dur
    uicontrol('Style', 'text', 'Position', [30, 450, 120, 30], 'String', 'Pivot Freq (Hz):');
    pivotFreqInput = uicontrol('Style', 'edit', 'Position', [150, 450, 200, 30], 'String', '700');

    uicontrol('Style', 'text', 'Position', [30, 400, 120, 30], 'String', 'Pivot Dur (s):');
    pivotDurInput = uicontrol('Style', 'edit', 'Position', [150, 400, 200, 30], 'String', '0.5');

    % End freq / dur
    uicontrol('Style', 'text', 'Position', [30, 350, 120, 30], 'String', 'End Freq (Hz):');
    endFreqInput = uicontrol('Style', 'edit', 'Position', [150, 350, 200, 30], 'String', '1500');

    uicontrol('Style', 'text', 'Position', [30, 300, 120, 30], 'String', 'End Dur (s):');
    endDurInput = uicontrol('Style', 'edit', 'Position', [150, 300, 200, 30], 'String', '0.2');

    % STFT window length
    uicontrol('Style', 'text', 'Position', [30, 250, 120, 30], 'String', 'STFT Window Length:');
    windowLengthInput = uicontrol('Style', 'edit', 'Position', [150, 250, 200, 30], 'String', '256');

    % STFT Overlap Length
    uicontrol('Style', 'text', 'Position', [30,200, 120, 30], 'String', 'STFT Overlap Length:');
    overlapLengthInput = uicontrol('Style', 'edit', 'Position', [150, 200, 200, 30], 'String', '250');

    % STFT FFT Points
    uicontrol('Style', 'text', 'Position', [30, 150, 120, 30], 'String', 'STFT FFT Points:');
    fftPointsInput = uicontrol('Style', 'edit', 'Position', [150, 150, 200, 30], 'String', '256');

    % Buttons
    generateButton = uicontrol('Style', 'pushbutton', 'Position', [400, 250, 150, 40], ...
        'String', 'Generate Signal', 'Callback', @generateSignal);

    playButton = uicontrol('Style', 'pushbutton', 'Position', [400, 190, 150, 40], ...
        'String', 'Play Signal', 'Callback', @playSignal);

    saveButton = uicontrol('Style', 'pushbutton', 'Position', [400, 130, 150, 40], ...
        'String', 'Save Signal', 'Callback', @saveSignal);


    % Axes for spectrogram
    ax = axes('Parent', fig, 'Position', [0.55, 0.1, 0.4, 0.8]);
    title(ax, 'Transmitted Signal Spectrogram');

    % NEW: Multiline text boxes to show "Sent Freqs" and "Sent Bits"
    uicontrol('Style', 'text', 'Position', [30, 100, 120, 30], ...
        'String', 'Sent Frequencies:', 'HorizontalAlignment', 'left');
    sentFreqsBox = uicontrol('Style', 'edit', 'Position', [30, 50, 250, 50], ...
        'Max', 2, 'Min', 0, 'HorizontalAlignment', 'left');
```

```
uicontrol('Style', 'text', 'Position', [300, 100, 120, 30], ...
    'String', 'Sent Binary Bits:', 'HorizontalAlignment', 'left');
sentBitsBox = uicontrol('Style', 'edit', 'Position', [300, 50, 250, 50], ...
    'Max', 2, 'Min', 0, 'HorizontalAlignment', 'left');

% Global variables
global finalSignal fs sentFreqs sentBits;
```



Seed (Random): 42
Sampling Freq (Hz): 44100
Pivot Freq (Hz): 700
Pivot Dur (s): 0.5
End Freq (Hz): 1500
End Dur (s): 0.2
STFT Window Length: 256
STFT Overlap Length: 250
STFT FFT Points: 256
Sent Frequencies:
Sent Binary Bits:

Callback Functions

```
% --- Generate Signal ---
function generateSignal(~, ~)
    message    = get(msgInput, 'String');
    studentNo  = get(studentNoInput, 'String');
    catValue   = get(categoryMenu, 'Value');
    randomSeed = str2double(get(seedInput, 'String'));
    fs         = str2double(get(samplingFreqInput, 'String'));
    pivotFreq  = str2double(get(pivotFreqInput, 'String'));
    pivotDur   = str2double(get(pivotDurInput, 'String'));
    endFreq    = str2double(get(endFreqInput, 'String'));
    endDur     = str2double(get(endDurInput, 'String'));

    if isempty(message) || isempty(studentNo) || isnan(randomSeed) || ...
       isnan(fs) || isnan(pivotFreq) || isnan(endFreq)
        errordlg('Please provide valid inputs!', 'Input Error');
        return;
    end

    switch catValue
        case 1
            M=2;   Th=1.0;
            deltaF = 100;
        case 2
            M=4;   Th=0.75;
            deltaF = 150;
        case 3
            M=8;   Th=0.5;
            deltaF = 200;
    end

    [finalSignal, sentFreqs, sentBits] = generateFHSSSignal_ShowHops( ...
        message, studentNo, M, Th, randomSeed, fs, pivotFreq, pivotDur, endFreq, endDur);

    % Plot spectrogram
    axes(ax);
    spectrogram(finalSignal, 256, 250, 256, fs, 'yaxis');
    title('Transmitted Signal Spectrogram');
    xlabel('Time (s)');
    ylabel('Frequency (Hz)');
```

```
        % Show frequencies
        freqStr = "";
        for iHop=1:length(sentFreqs)
            freqStr = freqStr + sprintf('Hop %d: %.2f Hz\n', iHop, sentFreqs(iHop));
        end
        set(sentFreqsBox, 'String', freqStr);

        % Show binary data
        set(sentBitsBox, 'String', sentBits);
    end

    % --- Play Signal ---
    function playSignal(~, ~)
        if isempty(finalSignal)
            errordlg('Generate a signal first!', 'Error');
            return;
        end
        sound(finalSignal, fs);
    end

    % --- Save Signal ---
    function saveSignal(~, ~)
        if isempty(finalSignal)
            errordlg('Generate a signal first!', 'Error');
            return;
        end
        audiowrite('transmitted_fhss_signal.wav', finalSignal, fs);
        msgbox('Signal saved as transmitted_fhss_signal.wav', 'Success');
    end
```

```
end
```

---

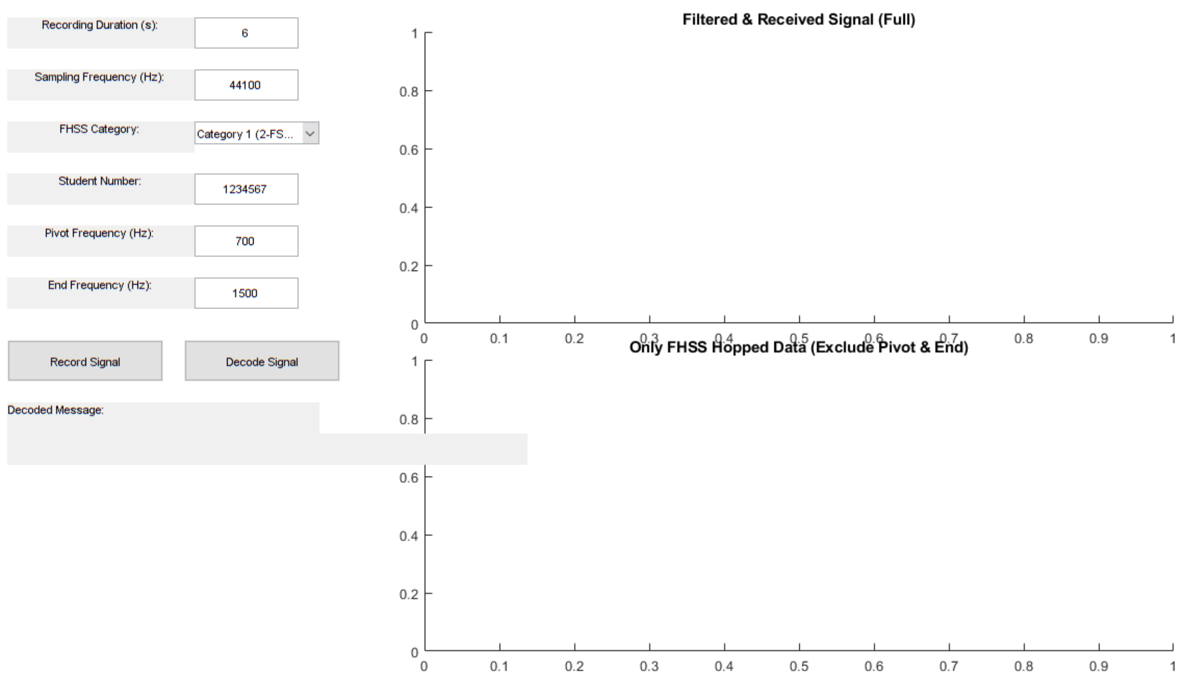
generateFHSSSignal_ShowHops Returns finalSignal, plus: sentFreqs = array of actual freq used per hop sentBits = the entire binary string transmitted

---

```
function [finalSignal, freqUsed, binDataStr] = generateFHSSSignal_ShowHops( ...
    message, studentNo, M, Th, randomSeed, fs, ...
    pivotFreq, pivotDur, endFreq, endDur)


    % Build freqArr from studentNo
    N7 = str2double(studentNo(end));
    N6 = str2double(studentNo(6));
    N5 = str2double(studentNo(5));
    switch M
        case 2
            b = mod(N6+1,2)*2-1;
            N7 = mod(N7, 5);
            S  = 100 * N7;
            array = 1000:500:5000;
            freqArr = array + S*b;
        case 4
            b = mod(N5+1,2)*2-1;
            S  = 100 * N7;
            array = 1500:1000:8500;
            freqArr = array + S*b;
        case 8
            S  = 100 * N7;
            array = 1000:2000:11000;
            freqArr = array + S;
    end
    disp(freqArr);

    % Convert message -> bits
    asciiVals  = double(message);
    binStr     = reshape(dec2bin(asciiVals, 8).', 1, []);
    bitsPerSym = log2(M);

    % mkMap skipping 0 if M=8
    switch M
        case 2
            mkMapping = [-1 +1];
            deltaF = 100;
        case 4
            mkMapping = [-2 -1 +1 +2];
            deltaF = 150;
        case 8
            mkMapping = [-4 -3 -2 -1 +1 +2 +3 +4];
            deltaF = 200;
        otherwise
            error('Unsupported M in transmitter');
    end

    % Pad so multiple of bitsPerSym
    padLen = mod(-length(binStr), bitsPerSym);
    if padLen>0
        binStr = [binStr, repmat('0',1,padLen)];
    end

    % Group bits -> mk
    groupedBits = reshape(binStr, bitsPerSym, []).';
    mkVals = mkMapping(bin2dec(groupedBits)+1).';
    disp(mkVals);
    rng(randomSeed);
    totalHops   = length(mkVals);
    hopIndices  = randi(length(freqArr), totalHops, 1);
    baseFreqs   = freqArr(hopIndices);
    freqUsed    = zeros(1, totalHops);

    t_hop = 0: 1/fs : Th - 1/fs;
    fhssSignal = [];

    for iHop=1:totalHops
        thisBase = baseFreqs(iHop);
        thisMk   = mkVals(iHop);
        modFreq  = thisBase + thisMk*deltaF;
        freqUsed(iHop) = modFreq;  % store for display

        oneHopSig = sin(2*pi*modFreq * t_hop);
        fhssSignal= [fhssSignal, oneHopSig]; %#ok<AGROW>
    end

    % Normalize hopped portion
    if max(abs(fhssSignal))<1e-12
        fhssSignal= zeros(size(fhssSignal));
    else
        fhssSignal= fhssSignal / max(abs(fhssSignal));
    end

    % Pivot
    t_pivot    = 0:1/fs : pivotDur - 1/fs;
    pivotSig   = 50 * sin(2*pi*pivotFreq*t_pivot);

    % End tone
    t_end   = 0:1/fs : endDur - 1/fs;
    endSig  = sin(2*pi*endFreq*t_end);

    % Combine
    combined= [pivotSig, fhssSignal*10, endSig];
    combined= combined / max(abs(combined)) * 0.8;
    finalSignal= combined;

    % Return binStr as is
    binDataStr = binStr;
end
```

Table of Contents

```
function FHSS_Receiver_GUI()

    % Create figure
    fig = figure('Name', 'FHSS Receiver (Pivot+End, bottom=only hopped
portion, fix pivot)', ...
                 'Position', [100, 100, 1200, 700]);

    % -- UI Controls -------------------------------------------------
    uicontrol('Style', 'text', 'Position', [20, 650, 180, 30], ...
        'String', 'Recording Duration (s):');
    durationInput = uicontrol('Style', 'edit', 'Position', [200, 650, 100,
30], ...
        'String', '6');

    uicontrol('Style', 'text', 'Position', [20, 600, 180, 30], ...
        'String', 'Sampling Frequency (Hz):');
    samplingFreqInput = uicontrol('Style', 'edit', 'Position', [200, 600,
100, 30], ...
        'String', '44100');

    % FHSS Category
    uicontrol('Style', 'text', 'Position', [20, 550, 180, 30], ...
        'String', 'FHSS Category:');
    categoryMenu = uicontrol('Style', 'popupmenu', ...
        'Position', [200, 550, 120, 30], ...
        'String', {'Category 1 (2-FSK, Th=1.0)', ...
                   'Category 2 (4-FSK, Th=0.75)', ...
                   'Category 3 (8-FSK, Th=0.5)'});

    % Student Number
    uicontrol('Style', 'text', 'Position', [20, 500, 180, 30], ...
        'String', 'Student Number:');
    studentNoInput = uicontrol('Style', 'edit', 'Position', [200, 500, 100,
30], ...
        'String', '1234567');

    % Pivot / End freq
    uicontrol('Style', 'text', 'Position', [20, 450, 180, 30], ...
        'String', 'Pivot Frequency (Hz):');
    pivotFreqInput = uicontrol('Style', 'edit', 'Position', [200, 450, 100,
30], ...
        'String', '700');
    uicontrol('Style', 'text', 'Position', [20, 400, 180, 30], ...
        'String', 'End Frequency (Hz):');
    endFreqInput = uicontrol('Style', 'edit', 'Position', [200, 400, 100,
30], ...
        'String', '1500');

    recordButton = uicontrol('Style', 'pushbutton', ...
        'Position', [20, 330, 150, 40], ...
        'String', 'Record Signal', ...
        'Callback', @recordSignal);

    decodeButton = uicontrol('Style', 'pushbutton', ...
        'Position', [190, 330, 150, 40], ...
        'String', 'Decode Signal', ...
        'Callback', @decodeSignal);

    % Decoded message
    uicontrol('Style', 'text', 'Position', [20, 280, 300, 30], ...
        'String', 'Decoded Message:', 'HorizontalAlignment', 'left');
    decodedMsgDisplay = uicontrol('Style', 'text', 'Position', [20, 250,
500, 30], ...
        'HorizontalAlignment', 'left');

    % Axes for spectrograms
    axFull = axes('Parent', fig, 'Position', [0.35, 0.55, 0.6, 0.4]);
    title(axFull, 'Filtered & Received Signal (Full)');

    axCrop = axes('Parent', fig, 'Position', [0.35, 0.1, 0.6, 0.4]);
    title(axCrop, 'Only FHSS Hopped Data (Exclude Pivot & End)');

    % Globals
    global recordedSignal fs decodedMessage;
```


Recording Duration (s): 6

Sampling Frequency (Hz): 44100

FHSS Category: Category 1 (2-FS...

Student Number: 1234567

Pivot Frequency (Hz): 700

End Frequency (Hz): 1500

Decoded Message:

Record Signal
Decode Signal
Filtered & Received Signal (Full)
Only FHSS Hopped Data (Exclude Pivot & End)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Record

```
    function recordSignal(~, ~)
        duration = str2double(get(durationInput, 'String'));
        fs = str2double(get(samplingFreqInput, 'String'));

        if isempty(duration) || duration <= 0 || isempty(fs) || fs <= 0
            errordlg('Please provide valid recording parameters!','Input
Error');
            return;
        end

        recObj = audiorecorder(fs,16,1);
        disp('Recording started...');
        recordblocking(recObj, duration);
        disp('Recording stopped.');
        recordedSignal= getaudiodata(recObj);
    end
```

Decode

```
    function decodeSignal(~, ~)
        if isempty(recordedSignal)
            errordlg('Record a signal first!','Error');
            return;
        end

        catValue= get(categoryMenu,'Value');
        switch catValue
            case 1, M=2; Th=1.0; deltaF=100;
            case 2, M=4; Th=0.75; deltaF=150;
            case 3, M=8; Th=0.5; deltaF=200;
        end

        pivotFreq = str2double(get(pivotFreqInput,'String'));
        endFreq   = str2double(get(endFreqInput,'String'));
        studentNo = get(studentNoInput,'String');

        % Remove DC
        recordedSignal= recordedSignal - mean(recordedSignal);

        % Bandpass from 300 Hz -> 12000 Hz (example to handle higher freq)
        bpFilt = designfilt('bandpassfir','FilterOrder',200, ...
            'CutoffFrequency1',300,'CutoffFrequency2',15000, ...
            'SampleRate',fs);
        filteredSignal = filtfilt(bpFilt, recordedSignal);

        % Plot top
        axes(axFull);
        spectrogram(filteredSignal,256,250,256,fs,'yaxis');
        title('Filtered & Received Signal (Full)');
        xlabel('Time (s)');
        ylabel('Frequency (kHz)');

        % Decode pivot->end & skip edges
        [decodedMessage, hoppedSignal] = ...
            decodeFHSSPivotEnd_ExcludeBoth_FixedPivot( ...
                filteredSignal, fs, pivotFreq, endFreq, M, Th, studentNo,
deltaF);

        % Plot bottom
        axes(axCrop);
        spectrogram(hoppedSignal,256,250,256,fs,'yaxis');
        title('Only FHSS Hopped Data (Exclude Pivot & End)');
        xlabel('Time (s)');
        ylabel('Frequency (kHz)');

        set(decodedMsgDisplay, 'String', ['Decoded Message: ',
decodedMessage]);
    end

end
```

------

decodeFHSSPivotEnd_ExcludeBoth_FixedPivot **Pivot fix**: hoppedStart = pivotIndex (NOT pivotIndex + pivotLen) skip ~3000 from edges each hop for FFT

```
--------------------------------------------------------------------------
------

function [decodedMessage, hoppedSignal] = ...
    decodeFHSSPivotEnd_ExcludeBoth_FixedPivot( ...
        filteredSignal, fs, pivotFreq, endFreq, M, Th, studentNo, deltaF)

    decodedMessage = '';
    hoppedSignal   = [];

    pivotDur = 0.5;
    endDur   = 0.2;

    N7 = str2double(studentNo(end));
    N6 = str2double(studentNo(6));
    N5 = str2double(studentNo(5));
    switch M
        case 2
            b = mod(N6+1,2)*2-1;
            N7 = mod(N7, 5);
            S  = 100 * N7;
            array = 1000:500:5000;
            freqArr = array + S*b;
        case 4
            b = mod(N5+1,2)*2-1;
            S  = 100 * N7;
            array = 1500:1000:8500;
            freqArr = array + S*b;
        case 8
            S  = 100 * N7;
            array = 1000:2000:11000;
            freqArr = array + S;
    end
    disp(freqArr)

    % pivot ref
    t_pivot= 0:1/fs : pivotDur-1/fs;
    pivotRef= 50*sin(2*pi*pivotFreq*t_pivot);  % amplitude 50
    if max(abs(pivotRef))>0
        pivotRef= pivotRef/ max(abs(pivotRef));
    end

    [rPivot,lagsPivot]= xcorr(filteredSignal,pivotRef);
    [~, iMaxPivot]= max(abs(rPivot));
    % pivotIndex is the END of pivot => pivotIndex = pivot start + pivotLen
    pivotIndex= lagsPivot(iMaxPivot) + length(pivotRef);

    if pivotIndex<1 || pivotIndex> length(filteredSignal)
        disp('WARNING: pivot not found => decode entire audio...');
        pivotIndex=1;
    end
    % end ref
    t_end= 0:1/fs : endDur-1/fs;
    endRef= sin(2*pi*endFreq*t_end);
    if max(abs(endRef))>0
        endRef= endRef/ max(abs(endRef));
    end

    [rEnd,lagsEnd]= xcorr(filteredSignal,endRef);
    [~, iMaxEnd]= max(abs(rEnd));
    endIndex= lagsEnd(iMaxEnd)+ length(endRef);

    if endIndex<1 || endIndex> length(filteredSignal)
        disp('WARNING: end tone not found => decode until end...');
        endIndex= length(filteredSignal);
    end

    % The fix: hoppedStart is pivotIndex, not pivotIndex + pivotLen
    hoppedStart= pivotIndex;
    hoppedStop = endIndex - length(endRef);

    if hoppedStart<1, hoppedStart=1; end
    if hoppedStop> length(filteredSignal), hoppedStop=
length(filteredSignal); end
    if hoppedStop< hoppedStart
        disp('WARNING: pivot->end invalid => decode entire audio...');
        hoppedStart=1;
        hoppedStop= length(filteredSignal);
    end

    hoppedSignal = filteredSignal(hoppedStart : hoppedStop);
    bpFilt = designfilt('bandstopfir','FilterOrder',200, ...
            'CutoffFrequency1',680,'CutoffFrequency2',720, ...
            'SampleRate',fs);
    hoppedSignal = filtfilt(bpFilt, hoppedSignal);
    disp(length(hoppedSignal))

    % Now decode hoppedSignal
    decodePortion= hoppedSignal;
    numHops= floor(length(decodePortion)/(Th*fs));
    if M == 2
        mkMap = [-1 +1];
    elseif M == 4
        mkMap = [-2 -1 +1 +2];
    elseif M == 8
        mkMap = [-4 -3 -2 -1 +1 +2 +3 +4];
    end
    bitsPerSym= log2(M);
    thresholdTol= 0.70;

    % We'll skip edges: 3000 from the start, 3000 from the end
    skipN= floor(fs*Th/10);

    decodedMk= [];
    for iHop=1 : numHops+1
        hopStart= round((iHop-1)*Th*fs)+1;
        hopEnd  = round(iHop*Th*fs);
        disp(iHop)
        if hopEnd> length(decodePortion)
            hopEnd= length(decodePortion);
        end

        % skip 3000 from start, 3000 from end
        innerStart = hopStart + skipN*5;
        innerEnd   = hopEnd   - skipN*3;

        if innerStart> innerEnd
            innerStart = hopStart + skipN*4;
            innerEnd   = hopEnd   - skipN*2;
        end

        if innerStart<1, innerStart=1; end
        if innerEnd> length(decodePortion), innerEnd= length(decodePortion);
end

        oneHop= decodePortion(innerStart : innerEnd);

        if isempty(oneHop) || max(abs(oneHop))<1e-12
            decodedMk= [decodedMk, NaN];
            continue;
        end

        % FFT
        fftLen= 2^nextpow2(length(oneHop));
        fftSpec= abs(fft(oneHop, fftLen));
        freqAxis= (0:fftLen-1)* fs/ fftLen;

        halfIdx= floor(fftLen/2);
        [~, iPeak]= max(fftSpec(1:halfIdx));
        detectedFreq= freqAxis(iPeak);
        % nearest freq
        [~, iF]= min(abs(freqArr- detectedFreq));
        fk= freqArr(iF);
        disp(iF)
        disp(fk)
        disp(detectedFreq)
        mkEst= (detectedFreq- fk)/ deltaF;
        disp(detectedFreq -fk)
        disp(mkEst)
        disp("/////////////")
        [~, iMap]= min(abs(mkMap- mkEst));
        if abs(mkMap(iMap)- mkEst)> thresholdTol
            decodedMk= [decodedMk, NaN];
        else
            decodedMk= [decodedMk, mkMap(iMap)];
        end
        part_signal = decodePortion(hopStart:hopEnd);
        bpFilt = designfilt('bandpassfir','FilterOrder',200, ...
'CutoffFrequency1',detectedFreq-50,'CutoffFrequency2',detectedFreq+50, ...
            'SampleRate',fs);
        part_signal = filtfilt(bpFilt, part_signal);

        decodePortion(hopStart:hopEnd) = part_signal;

    end
    hoppedSignal = decodePortion;
    % mk->bits
    binDecoded='';
    for val= decodedMk
        if isnan(val)
            binDecoded= [binDecoded, repmat('0',1,bitsPerSym)];
        else
            idx= find(mkMap== val,1);
            binSym= dec2bin(idx-1,bitsPerSym);
            binDecoded= [binDecoded, binSym];
        end
    end
    disp(binDecoded)
    % bits->ASCII
    padLen= mod(-length(binDecoded),8);
    if padLen>0
        binDecoded= [binDecoded, repmat('0',1,padLen)];
    end

    localMsg='';
    for i=1:8:length(binDecoded)
        byteStr= binDecoded(i:i+7);
        localMsg= [localMsg, char(bin2dec(byteStr))];
    end

    decodedMessage= localMsg;
end
```

Published with MATLAB® R2024b