ĐẠI HỌC BÁCH KHOA TP HỒ CHÍ MINH
KHOA ĐIỆN - ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG

PROJECT

HỆ THỐNG THÔNG TIN NÂNG CAO

ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ 4-FSK TRÊN TMS320C5515

Thầy hướng dẫn: GS.TS LÊ TIẾN THƯỜNG
Học viên: BÙI THANH TÍNH - 1970464

TP Hồ Chí Minh, ngày 29 tháng 08 năm 2020

LỜI MỞ ĐẦU

Trong những năm qua, kỹ thuật viễn thông đã có những bước tiến không ngừng với tốc độ nhanh. Đặc biệt trong thời đại công nghệ thông tin số hiện nay, các kỹ thuật xử lý số càng được xem trọng và cải thiện dựa trên các kỹ thuật nền tảng, một trong số các kỹ thuật đó là điều chế dịch tần FSK (Frequency Shift Keying). Trong nội dung của bài báo cáo này, các lý thuyết của điều chế và giải điều chế FSK được trình bày cụ thể. Sau đó, để áp dụng các lý thuyết đã đưa ra, tôi sử chương trình MATLAB để mô phỏng quá trình điều chế, cũng như giải điều chế 4-FSK. Tuy nhiên, quá trình mô phỏng MATLAB sẽ không có các vấn đề về nhiễu luôn xảy ra trong các hệ thống thực tế. Do đó, KIT C5515 - một bộ xử lý công suất thấp 16 bits chuyên dụng cho các ứng dụng xử lý tín hiệu, được sử dụng để xử lý các tín hiệu điều chế 4-FSK tương tự bài toán mô phỏng trong chương trình MATLAB. Sau khi thực hiện điều chế và giải điều chế 4-FSK trên KIT C5515, các so sánh và nhận xét giữa các tín hiệu điều chế thực tế và tín hiệu trong mô phỏng MATLAB được đưa ra để xây dựng các kết luận.

Các source code sẽ được đăng trên Github cá nhân với đường dẫn `https://github.com/XNCV/4-FSK-Modulation-and-Demodulation-with-TMS320C5515.git`.

Mục lục

Danh sách hình vẽ

Chương 1

LÝ THUYẾT ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ FSK

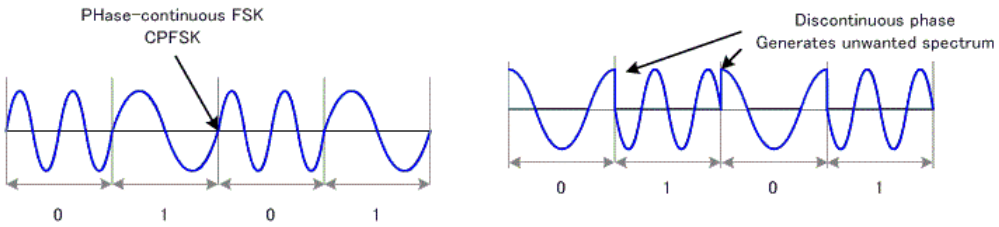
Điều chế số dịch tần FSK (Frequency Shift Keying) là dạng điều chế truyền chuỗi bit nhị phân dưới dạng tín hiệu có các mức tần số khác nhau. Với phương thức truyền từng bit nhị phân BFSK (Binary FSK), tín hiệu sau điều chế sẽ có hai mức tần số ứng với hai bit nhị phân '0' và '1'. Với phương thức truyền n bits nhị phân một, tức là ứng với mỗi ký tự truyền đi sẽ có n bits, ta cần $M = 2^n$ mức tần số để tổng hợp nên tín hiệu sau điều chế, và dạng này được gọi là điều chế tần bậc M (M-ary FSK). Nội dung chương này trình bày các lý thuyết về điều chế và giải điều chế đối với dạng cơ bản, nền tảng của FSK là điều chế dịch tần nhị phân BFSK và giới thiệu dạng mở rộng M-ary FSK. Ngoài ra, những ưu và nhược điểm của điều chế dịch tần cũng được đề cập trong nội dung còn lại.

1.1 Điều chế và giải điều chế BPSK

1.1.1 Điều chế BFSK

Đối với điều chế dịch tần nhị phân BFSK, mỗi bit nhị phân được truyền dưới dạng một tần số sóng mang tương ứng. Do đó, tần số sóng mang f sẽ nhận lần lượt một trong hai giá trị f_H tương ứng với bit 1 và tần số f_L tương ứng với bit 0. Dạng sóng của tín

hiệu sau điều chế BFSK $v_{FSK}(t)$ trong miền thời gian được thể hiện ở hình 1.1. Trong đó, dạng sóng dich tần pha liên tục $v_{CPFSK}(t)$ (Continuous Phase FSK) và dạng sóng dịch tần pha không liên tục $v_{NCPFSK}(t)$ (Non Continuous Phase FSK) được minh họa lần lượt trong hình 1.1(a), 1.1(b).



(a) Dạng sóng dịch tần pha liên tục (b) Dạng sóng dịch tần pha không liên tục

Hình 1.1: Các dạng sóng dịch tần dựa theo pha của tín hiệu

Hai tần số sóng mang f_H và f_L tương ứng với chuỗi bit $b(t)$ như sau:

- $f = f_H$ khi $b(t) = logic1$ (hoặc $d(t) = +1$)
- $f = f_L$ khi $b(t) = logic0$ (hoặc $d(t) = -1$)

Biểu thức toán học của tín hiệu sau điều chế BFSK có dạng:

$$v_{BFSK}(t) = A_c.\cos\left[2\pi(f_0 + d(t).\Delta f)\right]\mathrm{t} \tag{1.1}$$

trong đó $f_0 = \frac{f_H+f_L}{2}$ và $f_0 = \frac{f_H-f_L}{2}$.

Ngoài ra, biểu thức tín hiệu sau điều chế $v_{BFSK}(t)$ cũng có thể được viết dưới dạng khác như sau:

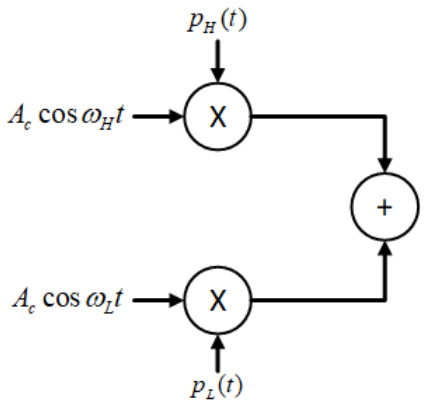
$$v_{BFSK}(t) = A_c.p_H(t).\cos 2\pi f_H\mathrm{t} + A_c.p_L(t).\cos 2\pi f_L\mathrm{t} \tag{1.2}$$

trong đó, hàm $p_H(t)$ và $p_L(t)$ có giá trị được cho trong bảng 1.1. Khi giá trị của hàm $p_H(t)$ bằng $+1$ và hàm $p_L(t)$ có giá trị 0 thì tín hiệu đang thể hiện mức logic 1. Ngược lại, nếu giá trị của hàm $p_L(t)$ bằng $+1$ và hàm $p_H(t)$ có giá trị 0 thì mức logic 0 đang được

Bảng 1.1: Giá trị hàm $p_H(t)$ và $p_L(t)$ tương ứng với các trạng thái nhị phân $d(t)$

d(t)	$p_H(t)$	$p_L(t)$
+1	+1	0
-1	0	+1

truyền. Từ công thức 1.2, sơ đồ khối thực hiện điều chế BFSK được thể hiện như hình 1.2.



Hình 1.2: Sơ đồ khối điều chế BFSK

Ở công thức 1.2, $v_{BFSK}(t)$ gồm hai thành phần sóng điều chế ASK (Amplitude Shift Keying) ở hai tần số khác nhau là f_H và f_L. Do đó, phổ của tín hiệu sau điều chế BFSK là tổng hợp của hai phổ ASK có tần số lần lượt là f_H và f_L.

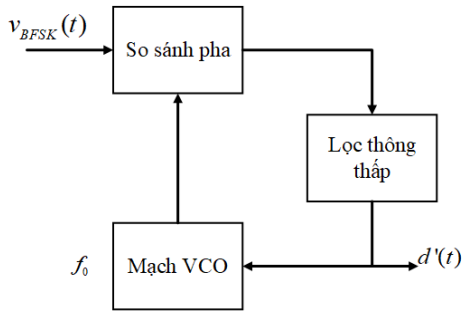
Nếu chọn f_H và f_L quá gần nhau, sẽ xảy ra hiện tượng chồng lấn giữa hai vùng phổ; ngược lại, nếu cách nhau quá xa sẽ gây lãng phí dải tần. Phương pháp được sử dụng là chọn f_H và f_L sao cho cá búp phổ chính có chung điểm cắt như hình **??**. Khi đó, băng thông của tín hiệu BFSK được cho bởi công thức 1.3.

$$BW \approx 4f_b = \frac{4}{T_b} \tag{1.3}$$

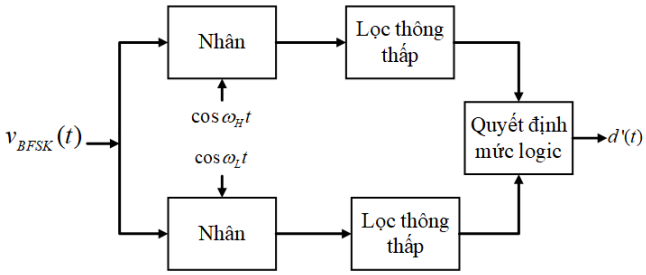
1.1.2 Giải điều chế BFSK

Để giải điều chế tín hiệu BFSK, hai dạng phương pháp được sử dụng. Dạng đầu tiên xem tín hiệu $v_{BFSK}(t)$ như là tín hiệu điều chế tần số và dùng các phương pháp giải điều

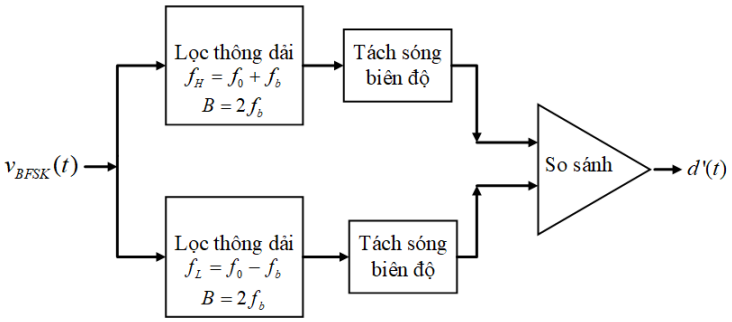
tần. Hình 1.3(a) mô tả phương pháp giải điều chế dùng vòng khóa PLL. Mặt khác, công thức 1.2 mô tả $v_{BFSK}(t)$ là tổng hợp của hai tín hiệu ASK ở hai tần số f_H và f_L khác nhau. Do đó, có thể dùng các phương pháp tách sóng biên độ hoặc tách sóng nhân kết hợp để giải điều chế như mô tả trong hình 1.3(b) và hình 1.3(c).



(a) Dùng vòng khóa pha



(b) Kiểu kết hợp (coherent)



(c) Kiểu không kết hợp (non-coherent)

Hình 1.3: Sơ đồ khối các bộ giải điều chế BFSK

1.2 Ưu, nhược điểm và các ứng dụng của điều chế FSK

Bất cứ hệ thống điều chế và giải điều chế nào cũng có những ưu điểm và hạn chế riêng. Tùy thuộc vào đặc tính sử dụng cũng như điều kiện môi trường mà các hệ thống phù hợp được lựa chọn. Diều chế FSK cũng không phải là một ngoại lê, một số điểm tích cực và hạn chế phải kể đến như:

Ưu điểm:

- Mạch đơn giản

- Xác suất lỗi bit thấp
- Tỷ số tín hiệu trên nhiễu SNR (signal to noise ratio) cao
- Ổn định với nhiễu tốt hơn ASK: Trong điều chế BFSK chỉ quan tâm đến tần số tín hiệu nên các nhiễu cộng ở biên độ không ảnh hưởng.

Nhược điểm:

- Băng thông yêu cầu lớn hơn ASK và PSK (phase shift keying): điều này làm cho FSK chỉ được sử dụng trong các hệ thống tốc độ thấp (tốc độ bit 1200 bits/sec).

Chương 2

ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ 4-FSK VỚI MATLAB

Trong chương này, một ví dụ mô phỏng điều chế và giải điều chế 4-FSK được thực hiện bằng phần mềm MATLAB.

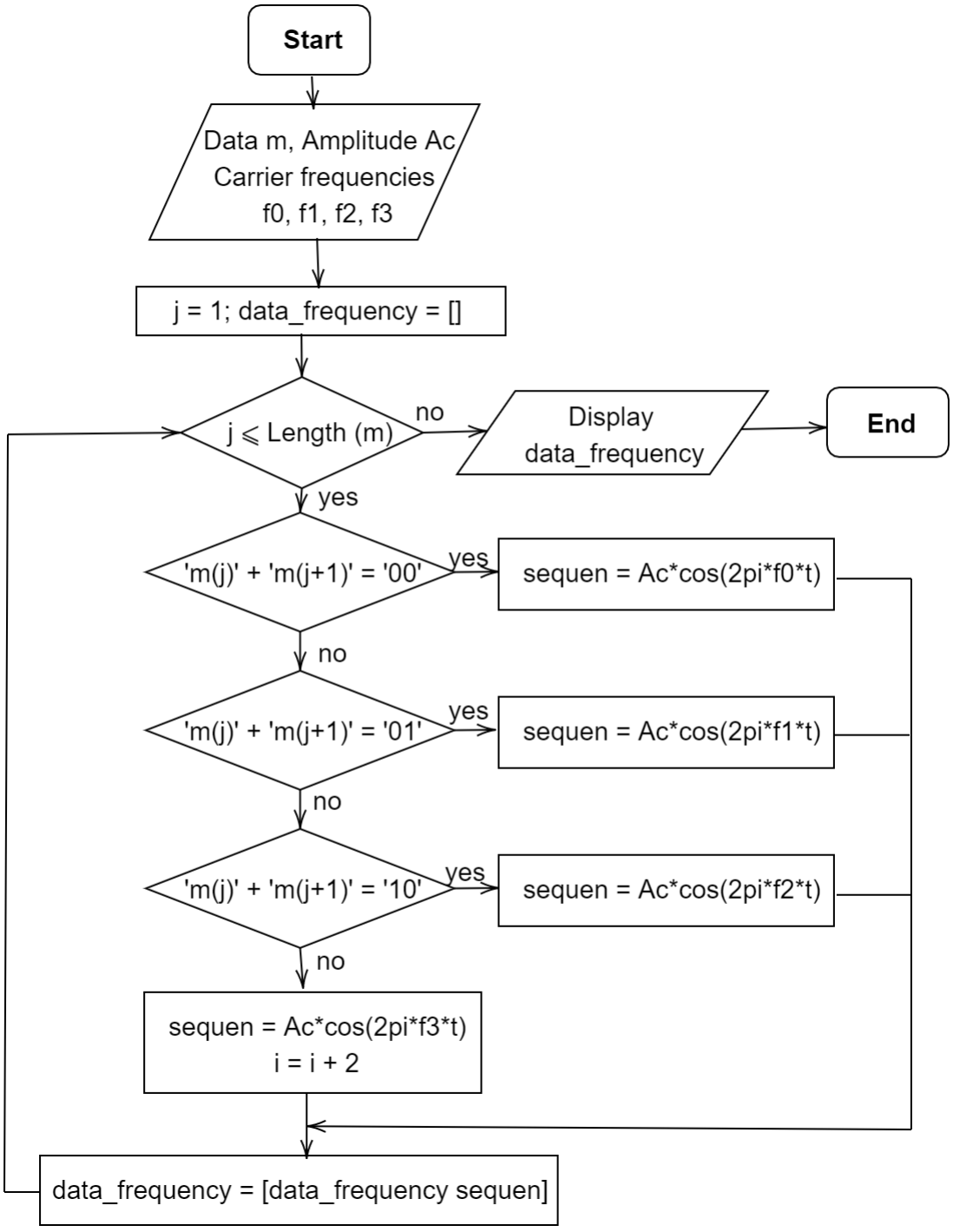
Nội dung ví dụ như sau: Cho một chuỗi các bits tín hiệu m, sóng mang có biên độ $A_c = 5$ không đổi, các mức tần số tương ứng với chuỗi bit được cho như bảng 2.1. Thực hiện mô phỏng MATLAB thể hiện tín hiệu sau điều chế 4-FSK và và chuỗi tín hiệu thu được.

Bảng 2.1: Các tần số thể hiện chuỗi bit

Mức tần số	**Tần số (kHz)**	**Chuỗi bit**
1	1	00
2	2	01
3	3	10
4	4	11

2.1 MATLAB code thực hiện điều chế 4-FSK

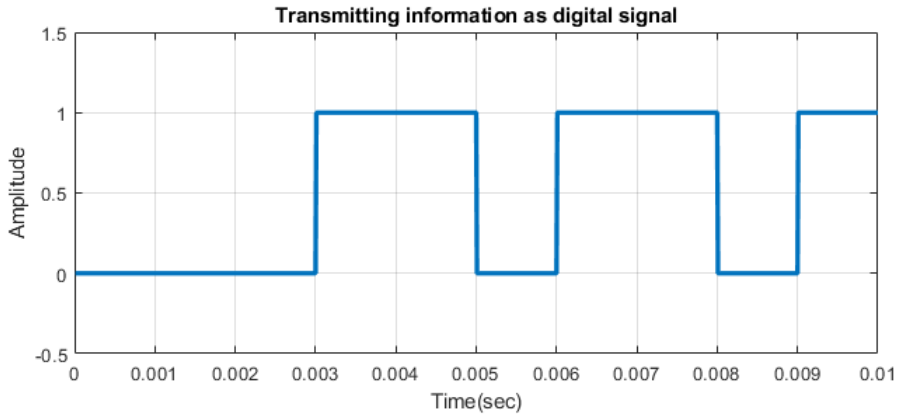
Sơ đồ khối điều chế 4-FSK:


Start
Data m, Amplitude Ac
Carrier frequencies
f0, f1, f2, f3
j = 1; data_frequency = []
j ⩽ Length (m)
no
Display
data_frequency
End
yes
'm(j)' + 'm(j+1)' = '00'
yes
sequen = Ac*cos(2pi*f0*t)
no
'm(j)' + 'm(j+1)' = '01'
yes
sequen = Ac*cos(2pi*f1*t)
no
'm(j)' + 'm(j+1)' = '10'
yes
sequen = Ac*cos(2pi*f2*t)
no
sequen = Ac*cos(2pi*f3*t)
i = i + 2
data_frequency = [data_frequency sequen]

Hiển thị tín hiệu

```
clc
clear all

%% Information
unit = 0.001;
m = [0 0 0 1 1 0 1 1 0 1];
disp(m);
%% Display data
data_line = [];
for i = 1:length(m)
    if m(i) == 0
        sequen = zeros(1,100);
    else
        sequen = ones(1,100);
    end
    data_line = [data_line sequen];
end
t_data = unit/100:unit/100:100*length(m)*(unit/100);
%subplot(4,1,1);
figure;
plot(t_data, data_line, 'lineWidth', 2.5); grid on;
axis([0 unit*length(m) -0.5 1.5]);
ylabel('Amplitude');
xlabel('Time(sec)');
title('Transmitting information as digital signal');
```

Ở đoạn code trên, chuỗi bit được lưu dưới dạng một ma trận một chiều. Các tín hiệu được xử lý ở tần số MHz nên biến đơn vị *unit* của thời gian có giá trị là 10^{-3} s. Biến ma trận *data_line* được dùng để vẽ chuỗi bit trên miền thời gian. Kết quả sau khi chạy đoạn code được minh họa trong hình 2.1.

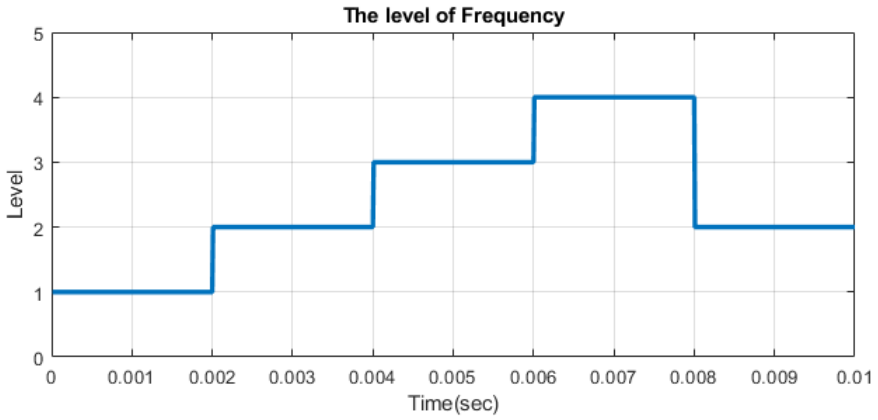


Hình 2.1: Chuỗi bit tín hiệu cần điều chế.

Hiển thị mức tần số cần mã hóa

```
%% Display the level of Frequency
data_level = [];
for j = 1:2:length(m)
    level = bin2dec(string(m(j))+string(m(j+1)));
    if level == 0
        sequen = ones(1,200);
    elseif level == 1
        sequen = 2*ones(1,200);
    elseif level == 2
        sequen = 3*ones(1,200);
    else
        sequen = 4*ones(1,200);
    end
    data_level = [data_level sequen];
end
%subplot(4,1,2);
figure;
plot(t_data, data_level, 'lineWidth', 2.5); grid on;
axis([0 unit*length(m) 0 5]);
ylabel('Level');
xlabel('Time(sec)');
title('The level of Frequency');
```

Dựa vào bảng 2.1, đoạn code trên thực hiện qui đổi các mức tần số ứng với chuỗi bit đã cho để tạo ra tín hiệu sau điều chế. Hình 2.2 thể hiện các mức tần số ứng với chuỗi bits theo thời gian thực.



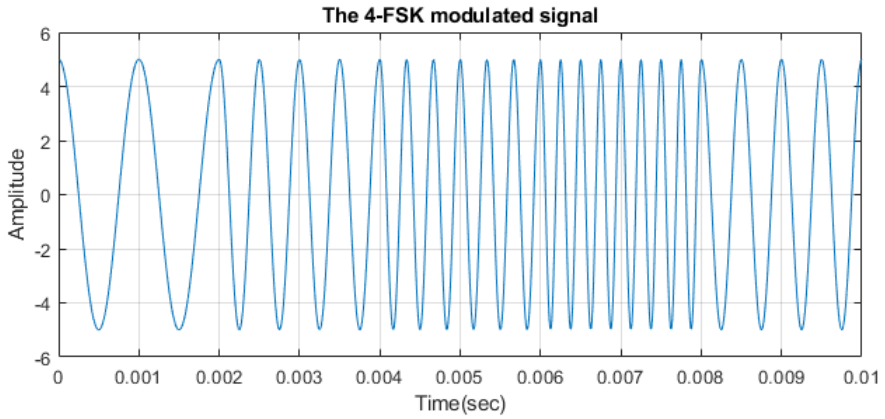
Hình 2.2: Các mức tần số tương ứng với chuỗi bits.

Tín hiệu sau điều chế

```
%% Display the 4-FSK modulated signal
Ac = 5; % The amplitude of carier signal
```

```
f0 = 1*1/unit; % Frequency for 00
f1 = 2*1/unit; % Frequency for 01
f2 = 3*1/unit; % Frequency for 10
f3 = 4*1/unit; % Frequency for 11

data_frequency = [];
t = unit/100:unit/100:2*unit; % 2*unit because of 4-FSK
for j = 1:2:length(m)
    level = bin2dec(string(m(j))+string(m(j+1)));
    if level == 0
        sequen = Ac*cos(2*pi*f0*t);
    elseif level == 1
        sequen = Ac*cos(2*pi*f1*t);
    elseif level == 2
        sequen = Ac*cos(2*pi*f2*t);
    else
        sequen = Ac*cos(2*pi*f3*t);
    end
    data_frequency = [data_frequency sequen];
end
%subplot(4,1,3);
figure;
plot(t_data, data_frequency); grid on;
axis([0 unit*length(m) -6 6]);
ylabel('Amplitude');
xlabel('Time(sec)');
title('The 4-FSK modulated signal');
```

Biến ma trận *data_frequency* dùng để lưu giá trị thể hiện tín hiệu sau điều chế 4-FSK. Các tần số $f0$, $f1$, $f2$ và $f3$ là các mức tần số tương ứng với các chuỗi bits mã hóa trong bảng 2.1. Tín hiệu dùng để truyền đi trên đường truyền sau khi điều chế 4-FSK của chuỗi bits được thể hiện trên hình 2.3.

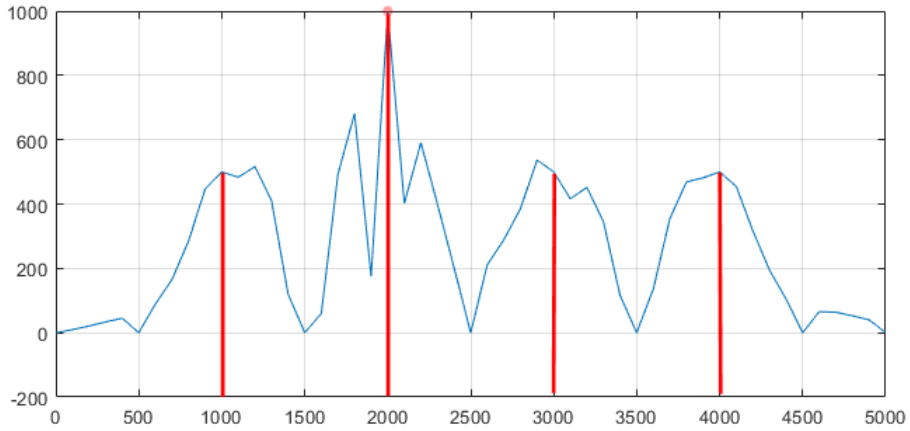


Hình 2.3: Tín hiệu sau điều chế 4-FSK.

Để quan sát phổ của tín hiệu sau khi điều chế, đoạn code MATLAB sau được thực thi:

```
%% test spectrum analyzer
fs = 10^5; % sampling frequency
Ts = 1/fs; % the duration between two sampling times
N = length(t_data);
f_data = -1/(2*Ts):1/(N*Ts):1/(2*Ts)-1/(N*Ts);
y_fft = fft(data_frequency); %%%
y_fhst = fftshift(y_fft);  %%%
figure;
plot(f_data, abs(y_fhst)); grid on;
axis([0 5*10^3 -200 1000]);
```

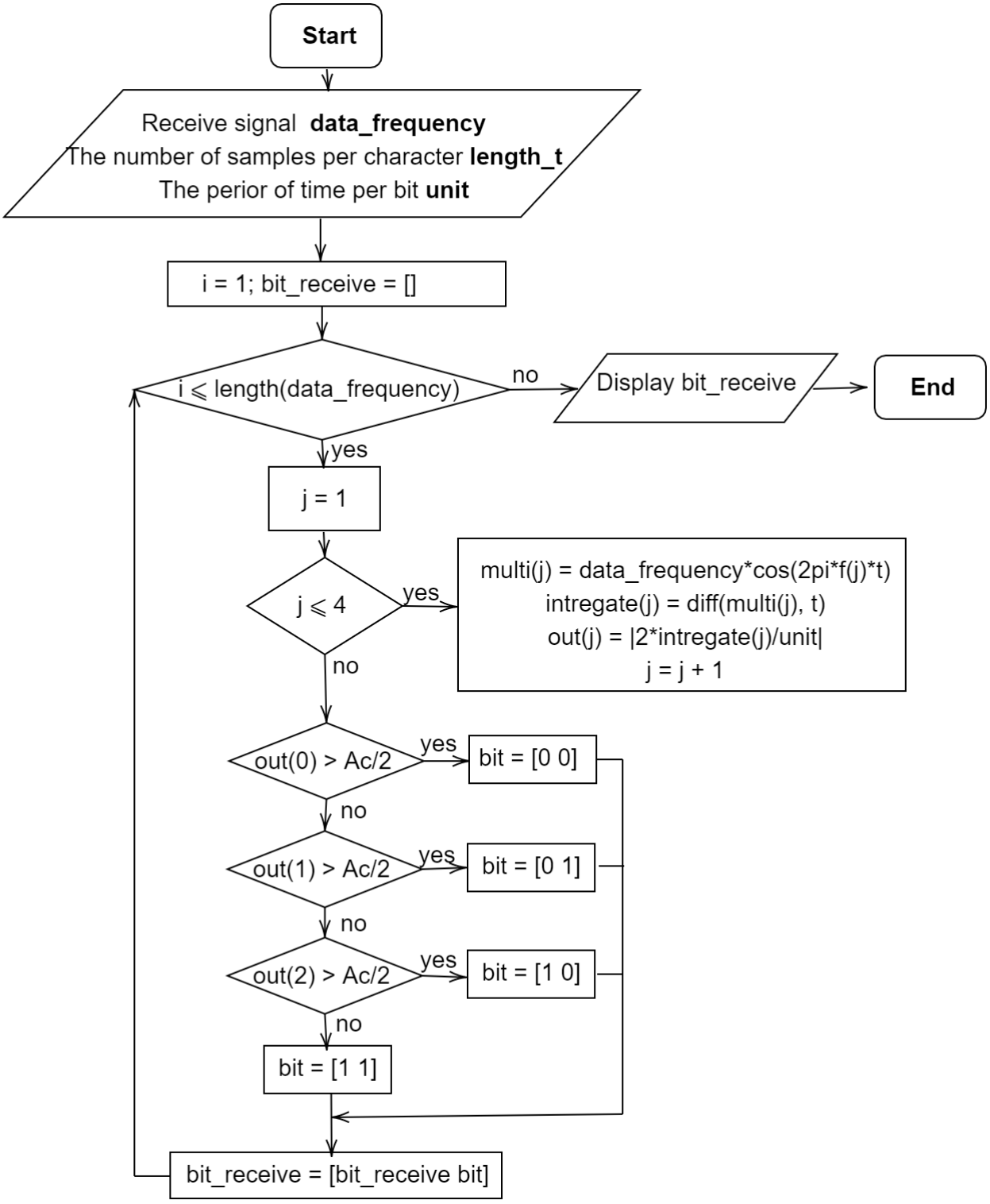
Hình 2.4 thể hiện dạng phổ của tín hiệu sau khi điều chế 4-FSK sử dụng phép biến đổi FFT rời rạc.



Hình 2.4: Phổ của tín hiệu sau điều chế 4-FSK

2.2 MATLAB code thực hiện giải điều chế 4-FSK

Sơ đồ khối giải điều chế 4-FSK:


Start
Receive signal data_frequency
The number of samples per character length_t
The perior of time per bit unit
i = 1; bit_receive = []
i ⩽ length(data_frequency)
no
Display bit_receive
End
yes
j = 1
j ⩽ 4
yes
multi(j) = data_frequency*cos(2pi*f(j)*t)
intregate(j) = diff(multi(j), t)
out(j) = |2*intregate(j)/unit|
j = j + 1
no
out(0) > Ac/2
yes
bit = [0 0]
no
out(1) > Ac/2
yes
bit = [0 1]
no
out(2) > Ac/2
yes
bit = [1 0]
no
bit = [1 1]
bit_receive = [bit_receive bit]

Giải điều chế

```
%% Demodulation
c0 = cos(2*pi*f0*t);
c1 = cos(2*pi*f1*t);
c2 = cos(2*pi*f2*t);
c3 = cos(2*pi*f3*t);

length_t = length(t);
bit_receive = [];
intregate_00 = [];
for i = 1:length_t:length(data_frequency)
    multi0 = data_frequency(i:i+length_t-1).*c0;
    multi1 = data_frequency(i:i+length_t-1).*c1;
    multi2 = data_frequency(i:i+length_t-1).*c2;
    multi3 = data_frequency(i:i+length_t-1).*c3;
    intregate_0 = trapz(t,multi0);
    intregate_1 = trapz(t,multi1);
    intregate_2 = trapz(t,multi2);
    intregate_3 = trapz(t,multi3);
    out0 = round(2*intregate_0/unit);
    out1 = round(2*intregate_1/unit);
    out2 = round(2*intregate_2/unit);
    out3 = round(2*intregate_3/unit);
    if out0 > Ac/2
        bit = [0 0];
    elseif out1 > Ac/2
        bit = [0 1];
    elseif out2 > Ac/2
        bit = [1 0];
    else
        bit = [1 1];
    end
    intregate_00 = [intregate_00 intregate_0];
    bit_receive = [bit_receive bit];
end
disp(' Binary information at Receiver :');
disp(bit_receive);
```

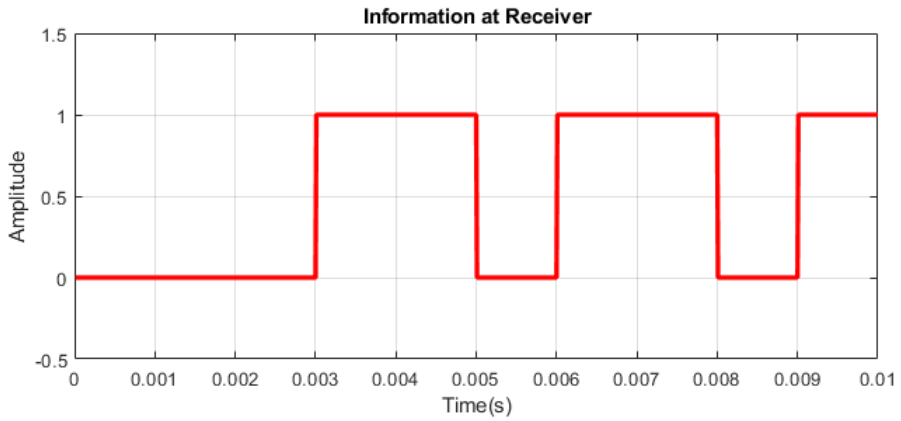
Trong đoạn code trên, bộ giải điều chế 4-FSK kiểu kết hợp (coherent) được áp dụng. Tín hiệu thu được $data\_frequency$ được nhân lần lượt với các các tín hiệu hình sin có các mức tần số trong bảng 2.1. Khi tín hiệu có cùng mức tấn số với tín hiệu nhân vào (các biến cx) thì một giá trị DC được sinh ra. Khi sử dụng hàm tích phân $trapz$ thì cổng ra của nhánh có cùng mức tần số với tín hiệu sẽ có giá trị lớn nhất và gần bằng với A_c. Ta chọn mức so sánh là $A_c/2$ để xác định mức tần số hiện tại của tín hiệu thu được, rồi

qui đổi sang mã nhị phân tương ứng sẽ thu được tín hiệu ở bộ thu tín hiệu 4-FSK.

Hiển thị tín hiệu sau điều chế

```
%% Show the information at Receiver
bit_receive_show = [];
for i = 1:length(bit_receive)
    if (bit_receive(i) == 0)
        sequence = zeros(1, 100);
    elseif (bit_receive(i) == 1)
        sequence = ones(1, 100);
    end
    bit_receive_show = [bit_receive_show sequence];
end
t_data_receive = unit/100:unit/100:100*length(bit_receive)*(unit/100);
subplot(4,1,4); grid on;
plot(t_data_receive, bit_receive_show, 'red', 'lineWidth', 2.5);
axis([0 length(bit_receive)*unit -0.5 1.5]);
ylabel('Amplitude');
xlabel('Time(s)');
title('Information at Receiver');
```

Đoạn code trên dùng để thể hiện tín hiệu chuỗi bit thu được trên miền thời gian thể hiện trong hình 2.5. Ta có kết luận rằng tín hiệu thu được trùng khớp với tín hiệu phát khi áp dụng code mô phỏng 4-FSK trong môi trường hoàn toàn không có nhiễu.



Hình 2.5: Chuỗi bits thu được ở bộ thu.

Chương 3

ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ 4-FSK VỚI TMS320C5515

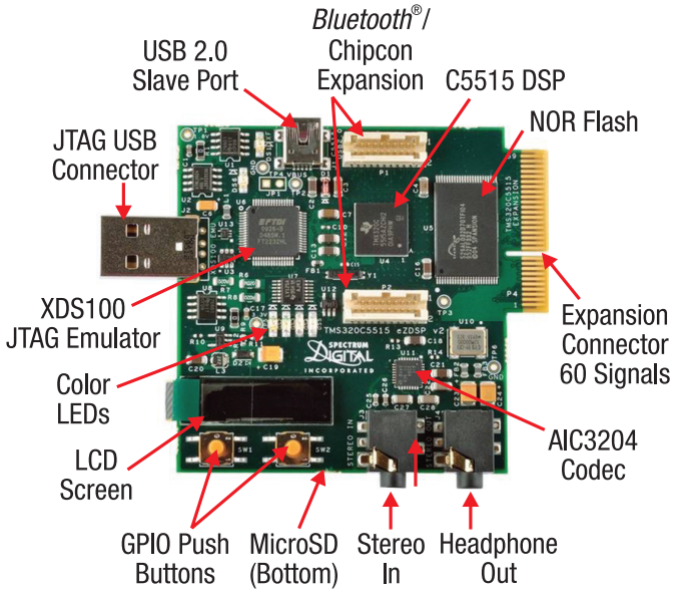
TMS320C5515 được đánh giá là vi xử lý xử lý số tín hiệu 16-bit công suất thấp nhất. Trong chương này, KIT xử lý số tín hiệu C5515eZdsp và phần mềm tạo dao động AudMeS-20080606 được giới thiệu. Sau đó, chúng sẽ được ứng dụng trong điều chế và giải điều chế 4-FSK với cùng bài toán trên MATLAB đã được trình bày ở chương 2. Từ đó, các kết quả được đưa ra và có một số nhận xét.

3.1 Giới thiệu TMS320C5515

KIT C5515eZdsp là công cụ xử lý số tín hiệu công suất thấp của công ty Texas Instruments (TI), sử dụng vi xử lý TMS320C5515 - được đánh giá là vi xử lý xử lý số tín hiệu 16-bit có công suất thấp nhất. Bộ KIT này cung cấp nhiều lựa chọn tương tác như USB 2.0 và SD. Cổng USB có thể cung cấp đủ công suất để vận hành KIT mà không cần thêm bất cứ nguồn nào khác. Hình 3.1 mô tả khái quát một số bộ phận có trên KIT C5515eZdsp.

TMS320C5515 có một số tính năng nổi bật sau:

- Hiệu suất cao, công suất tiêu thụ thấp.



Hình 3.1: KIT xử lý số tín hiệu TMDX5515eZdsp

- Sử dụng phép toán số học dấu chấm tĩnh.
- Một/hai lệnh được thực hiện mỗi chu kỳ.
- Giao tiếp với các ngoại vi:

  - 8/16 bit bộ nhớ ngoài tích hợp: bộ nhớ bất đồng bộ, SARAM, Flash (NOR, NAND), SARAM di động với độ rộng 16 bit.

  - Bốn bộ điều khiển DMA, mỗi bộ gồm có 4 kênh (tổng cộng có 16 kênh).

  - Hỗ trợ chuẩn truyền thông giao tiếp bất đồng bộ (UART).

  - Ba bộ timer dùng chung 32 bit (GP timer), trong đó 1 timer cấu hình như 1 Watchdog and/or GP.

  - Hỗ trợ giao tiếp theo chuẩn I2C theo phương thức Master – Slave (Chủ – tớ).

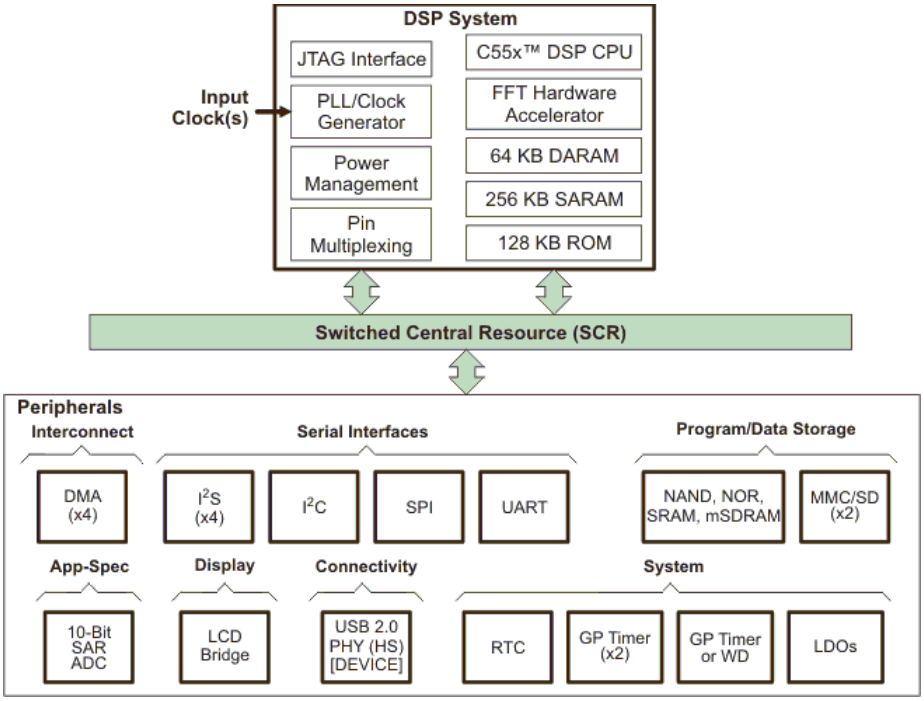
  - Hỗ trợ việc truyền dữ liệu theo chuẩn I2S.

  - Hỗ trợ USB 2.0 cho phép việc trao đổi dữ liệu với tốc độ cao.

  - Hỗ trợ LCD với giao diện bất đồng bộ.

  - ADC 10 bit với 4 bộ ghi xấp xỉ liên tiếp (SAR).

TMS320C5515 được chia làm 3 khối: (được thể hiện trong hình 3.2)



Hình 3.2: Sơ đồ khối chức năng TMS320C5515

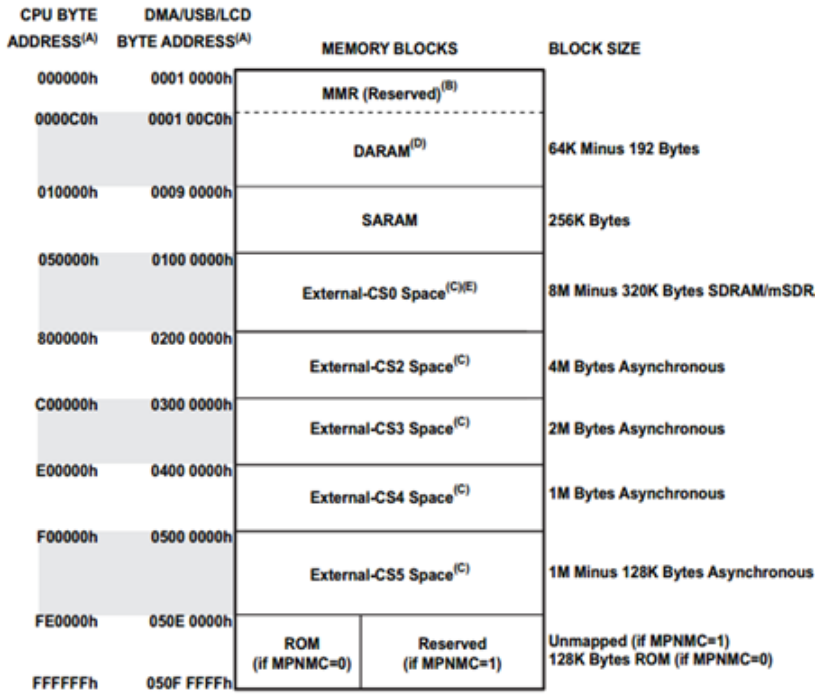
- Khối 1 (DSP System): hệ thống xử lý DSP.
- Khối 2 (Switched Central Resource): dữ liệu trung tâm chuyển mạch.
- Khối 3 (Peripherals): thiết bị ngoại vi.

Sơ đồ bộ nhớ

Trong lập trình, đặc biệt là lập trình hệ thống nhúng, một yếu tố quan trọng không thể bỏ qua trong quá trình thực hiện code là bộ nhớ của vi xử lý sử dụng. Đối với các chương trình nhỏ và đơn giản, việc này có thể không cần thiết. Tuy nhiên, việc cân nhắc xử dụng bộ nhớ trong các chương trình phức tạp không những giúp tránh sử dụng vượt quá bộ nhớ mà còn giúp chương trình thực thi nhanh hơn. Hình **3.3** mô tả cấu trúc các khối bộ nhớ của KIT C5515.

- On-Chip Dual-Access RAM (DARAM):

- DARAM chiếm 64k byte được nằm trong khoảng 000000h đến 00FFFFH.
- Mỗi khối DARAM có thể thực hiện 2 truy cập mỗi chu trình (2 đọc, 2 viết hoặc 1 đọc 1 viết).



Hình 3.3: Sơ đồ các khối bộ nhớ KIT C5515

- DARAM có thể được truy cập bởi chương trình bên trong, data hoặc DMA buses.

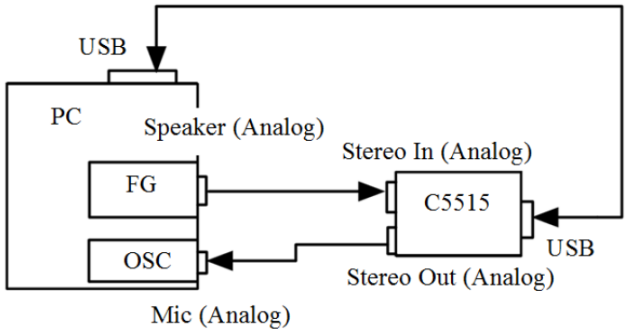
- On-Chip Single-Access RAM (SARAM):

- SARAM chiếm 256k byte được nằm trong vùng bộ nhớ từ 010000h đến 04FFFFh.
- Mỗi SARAM block cần thực hiện 1 truy cập mỗi chu kỳ (1 đọc hoặc 1 viết).
- SARAM có thể được truy cập bởi chương trình bên trong, data hoặc DMA buses. Riêng SARAM cũng có thể được truy cập bởi USB and LCD DMA buses.

- On-Chip Read-Only Memory (ROM):

- ROM chiếm 128k bytes được lưu trữ ở vùng ô nhớ có địa chỉ FE0000h đến FFFFFFh.
- Địa chỉ ROM có thể được truy cập bởi phần mềm bộ nhớ ngoài hoặc bộ nhớ bên trong ROM.

- I/O memory:



Hình 3.4: Sơ đồ kết nối KIT C5515 với PC

- Thiết bị bao gồm 64k bytes I/O trong vùng bộ nhớ thanh ghi trong thiết bị ngoại vi DSP và hệ thống thanh ghi được sử dụng kiểm soát idle, giám sát tình trạng và cấu hình hệ thống.

3.2 Thực hiện điều chế và giải điều chế 4-FSK với TMS320C5515

3.2.1 Một số chuẩn bị và kết nối

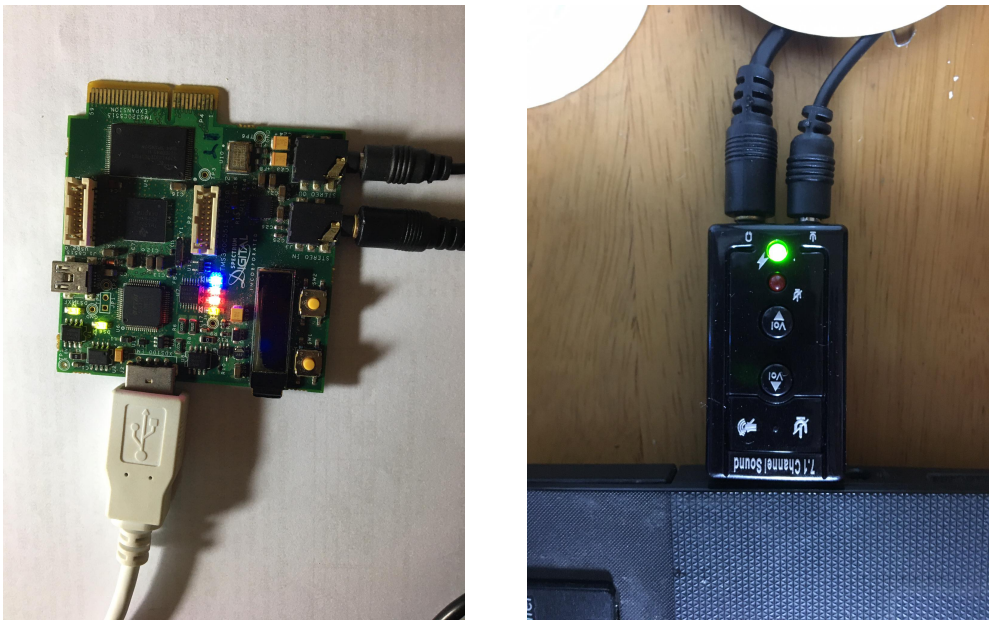
Sơ đồ kết nối KIT C5515 với PC được thể hiện trên hình 3.4.

Kết nối giữa PC với KIT C5515 thông qua cáp USB. Đầu USB Speaker của PC được kết nối với cổng Stereo In của C5515 và ngược lại, cổng Stereo Out của C5515 sẽ được kết nối với USB MIC của PC.

Chương trình được viết trên PC thông qua CCS sẽ được nạp vào C5515 qua cổng USB. Chương trình này sẽ điều khiển AIC3204 codec nhận tín hiệu analog từ cổng Stereo In chuyển sang tín hiệu digital và điều khiển DSP xử lý tín hiệu này. Sau đó, KIT sẽ thực hiện xử lý tín hiệu trở lại dạng analog và xuất ra cổng Stereo Out.

Tín hiệu chuỗi bit được tạo trên PC và truyền qua cổng USB speaker và nhận lại tín hiệu đã được xử lý từ cổng USB MIC. Sau đó, tín hiệu được đưa vào phần mềm Soundcard OScilloscope để nhận diện kết quả.

Một số hình ảnh kết nối mạch thực tế:



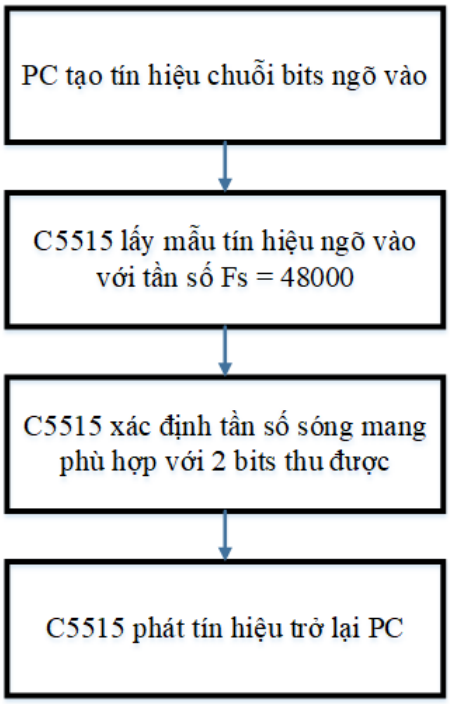
(a) Kết nối với KIT C5515 (b) Kết nối IN và OUT của PC

Hình 3.5: Kết nối C5515 với máy tính

3.2.2 Điều chế 4-FSK với C5515

Sơ đồ khối thực hiện điều chế 4-FSK trên KIT C5515 thông qua kết nối với máy tính được mô tả như hình 3.6:

Chương trình sử dụng các thư viện có sẵn được cung cấp bởi TI bao gồm: *usbstk*5515.*h*, *aic*3204.*h*. Nội dung của chương trình chính *main.c* thực hiện điều chế 4-FSK như sau:



Hình 3.6: Sơ đồ khối thực hiện điều chế trên KIT C5515.

```
/*
 * 4-FSK Modulation
 * Bui Thanh Tinh
 */

#include "stdio.h"
#include "usbstk5515.h"

void main( void )
{
    /* Initialize BSL */
    USBSTK5515_init( );
    SYS_EXBUSSEL = 0x6100;       // Enable I2C bus
   USBSTK5515_I2C_init( );      // Initialize I2C

   // Test connection
    printf("EXBUSSEL = %02x\n", SYS_EXBUSSEL);
    printf("Test the connection\n");
    aic3204_tone_headphone();
    USBSTK5515_wait( 100 );
    printf("Testing the connection is done.\n 4-FSK Modulation Mode\n");

    // Main loop
    aic3204_loop_stereo_in1();

    printf( "\n***END***\n" );
```

```
    SW_BREAKPOINT;
}
```

Trong đó, hàm *aic*3204*_tone_headphone* (xác định trong file *aic*3204*_tone_headphone.c*) dùng để phát một tín hiệu có dạng sóng vuông từ KIT C5515 đến máy tính, để đảm bảo rằng các kết nối được thiết lập chính xác. Nếu sau khi chạy chương trình, trước tiên nếu không có sóng vuông hiển thị trên Oscilloscope thì cần xem lại các kết nối. Bên cạnh đó, hàm *aic*3204*_loop_stereo_in*1 xác định trong file file *aic*3204*_loop_stereo_in*1*.c*, với nội dung như sau:

```
/*
 *  AIC3204 Tone
 *  Bui Thanh Tinh
 */
#include "stdio.h"
#include "usbstk5515.h"
extern Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
#define Rcv 0x08
#define Xmit 0x20
#include "math.h"

#define Fs 48000
#define F0 1000
#define F1 2000
#define F2 3000
#define F3 4000
#define PI 3.14159264

Int16 count=0;
Int16 state=0;

extern void aic3204_codec_read(Int16* left_input, Int16* right_input)
{
      while((Rcv & I2S0_IR) == 0); // Wait for interrupt pending flag
      *left_input = I2S0_W0_MSW_R; // 16 bit left channel received audio data
      //data1 = I2S0_W0_LSW_R;
      *right_input = I2S0_W1_MSW_R; // 16 bit right channel received audio data
      //data2 = I2S0_W1_LSW_R;
}

extern void aic3204_codec_write(Int16 left_output, Int16 right_output)
{
      /* Write Digital audio */
         while((Xmit & I2S0_IR) == 0); // Wait for interrupt pending flag
      I2S0_W0_MSW_W = left_output; // 16 bit left channel transmit audio data
         I2S0_W0_LSW_W = 0;
```

```
        I2S0_W1_MSW_W = right_output; // 16 bit right channel transmit audio
            data
        I2S0_W1_LSW_W = 0;
}


Int16 aic3204_loop_stereo_in1( )
{
    /* Pre-generated sine wave data, 16-bit signed samples */
    Int16 level, j, i = 0;
    Int16 sample, left_output, right_output;
    Int16 left_input, right_input;
    Int16 pre_value, current_value;
    short ptsig0[96];
   short ptsig1[96];
   short ptsig2[96];
   short ptsig3[96];
   short twobit[2]; // Use for sampling input bits

    /* Create 4 sine signals*/
    Int16 nsample = 96; int x;
   for (x=0; x < nsample; x++){
      ptsig0[x] = 1333*sin(2*PI*x*F0/Fs);
   }
   for (x=0; x < nsample; x++){
      ptsig1[x] = 1333*sin(2*PI*x*F1/Fs);
   }
   for (x=0; x < nsample; x++){
      ptsig2[x] = 1333*sin(2*PI*x*F2/Fs);
   }
   for (x=0; x < nsample; x++){
      ptsig3[x] = 1333*sin(2*PI*x*F3/Fs);
   }

   // Initially set up for AIC3204
    AIC3204_rset( 0, 0 );       // Select page 0
    AIC3204_rset( 1, 1 );       // Reset codec
    AIC3204_rset( 0, 1 );       // Select page 1
    AIC3204_rset( 1, 8 );       // Disable crude AVDD generation from DVDD
    AIC3204_rset( 2, 1 );       // Enable Analog Blocks, use LDO power
    AIC3204_rset( 0, 0 );       // Select page 0
    /* PLL and Clocks config and Power Up */
    AIC3204_rset( 27, 0x0d );   // BCLK and WCLK is set as o/p to
        AIC3204(Master)
    AIC3204_rset( 28, 0x00 );   // Data ofset = 0
    AIC3204_rset( 4, 3 );       // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN
        <-PLL CLK
    AIC3204_rset( 6, 7 );       // PLL setting: J=7
    AIC3204_rset( 7, 0x06 );    // PLL setting: HI_BYTE(D=1680)
```

```
AIC3204_rset( 8, 0x90 );    // PLL setting: LO_BYTE(D=1680)
AIC3204_rset( 30, 0x88 );   // For 32 bit clocks per frame in Master mode
   ONLY
                            // BCLK=DAC_CLK/N =(12288000/8) = 1.536MHz =
                               32*fs
AIC3204_rset( 5, 0x91 );    // PLL setting: Power up PLL, P=1 and R=1
AIC3204_rset( 13, 0 );      // Hi_Byte(DOSR) for DOSR = 128 decimal or
   0x0080 DAC oversamppling
AIC3204_rset( 14, 0x80 );   // Lo_Byte(DOSR) for DOSR = 128 decimal or
   0x0080
AIC3204_rset( 20, 0x80 );   // AOSR for AOSR = 128 decimal or 0x0080 for
   decimation filters 1 to 6
AIC3204_rset( 11, 0x82 );   // Power up NDAC and set NDAC value to 2
AIC3204_rset( 12, 0x87 );   // Power up MDAC and set MDAC value to 7
AIC3204_rset( 18, 0x87 );   // Power up NADC and set NADC value to 7
AIC3204_rset( 19, 0x82 );   // Power up MADC and set MADC value to 2
/* DAC ROUTING and Power Up */
AIC3204_rset( 0, 0x01 );    // Select page 1
AIC3204_rset( 12, 0x08 );   // LDAC AFIR routed to HPL
AIC3204_rset( 13, 0x08 );   // RDAC AFIR routed to HPR
AIC3204_rset( 0, 0x00 );    // Select page 0
AIC3204_rset( 64, 0x02 );   // Left vol=right vol
AIC3204_rset( 65, 0x00 );   // Left DAC gain to 0dB VOL; Right tracks Left
AIC3204_rset( 63, 0xd4 );   // Power up left,right data paths and set
   channel
AIC3204_rset( 0, 0x01 );    // Select page 1
AIC3204_rset( 16, 0x00 );   // Unmute HPL , 0dB gain
AIC3204_rset( 17, 0x00 );   // Unmute HPR , 0dB gain
AIC3204_rset( 9, 0x30 );    // Power up HPL,HPR
AIC3204_rset( 0, 0x00 );    // Select page 0
USBSTK5515_wait( 500 );     // Wait

/* ADC ROUTING and Power Up */
AIC3204_rset( 0, 1 );       // Select page 1
AIC3204_rset( 0x34, 0x30 ); // STEREO 1 Jack
                           // IN2_L to LADC_P through 40 kohm
AIC3204_rset( 0x37, 0x30 ); // IN2_R to RADC_P through 40 kohmm
AIC3204_rset( 0x36, 3 );    // CM_1 (common mode) to LADC_M through 40 kohm
AIC3204_rset( 0x39, 0xc0 ); // CM_1 (common mode) to RADC_M through 40 kohm
AIC3204_rset( 0x3b, 0 );    // MIC_PGA_L unmute
AIC3204_rset( 0x3c, 0 );    // MIC_PGA_R unmute
AIC3204_rset( 0, 0 );       // Select page 0
AIC3204_rset( 0x51, 0xc0 ); // Powerup Left and Right ADC
AIC3204_rset( 0x52, 0 );    // Unmute Left and Right ADC

AIC3204_rset( 0, 0 );
USBSTK5515_wait( 200 );     // Wait
/* I2S settings */
I2S0_SRGR = 0x0;
```

```
I2S0_CR = 0x8010;  // 16-bit word, slave, enable I2C
I2S0_ICMR = 0x3f;  // Enable interrupts
USBSTK5515_wait( 50 );

/* MAIN LOOP */
for ( i = 0 ; i < 50 ; j++ )
{
    for ( j = 0 ; j < 1000 ; j++ )
    {
        for ( sample = 0 ; sample < nsample ; sample++ )
        {
            aic3204_codec_read(&left_input, &right_input); //read the
                value from sterio-in
            switch(state)
            {
             case 0:
                pre_value = left_input;
                if(pre_value!=0)
                {
                   state = 1;
                }
                break;
             case 1:
                current_value = left_input;
                // Catch up the change of bits
                if(((pre_value < 0)&&(current_value > 0))||((pre_value >
                    0)&&(current_value < 0)))
             {
                state = 2;
                twobit[0]=0;twobit[1]=0;
             }
             break;
          case 2:
             count++;
             if(count == 120) twobit[0] = left_input;
             if(count == 360) twobit[1] = left_input;
             if((twobit[0] < 0)&&(twobit[1] < 0)) level = 3;
             else if ((twobit[0] < 0)&&(twobit[1] > 0)) level = 2;
             else if ((twobit[0] > 0)&&(twobit[1] < 0)) level = 1;
             else if ((twobit[0] > 0)&&(twobit[1] > 0)) level = 0;
             if (count==480) {
                count = 0; twobit[0]=0;twobit[1]=0;}
             break;
            }

            if(level == 0){
          left_output = ptsig0[sample];
        }
        if(level == 1){
```

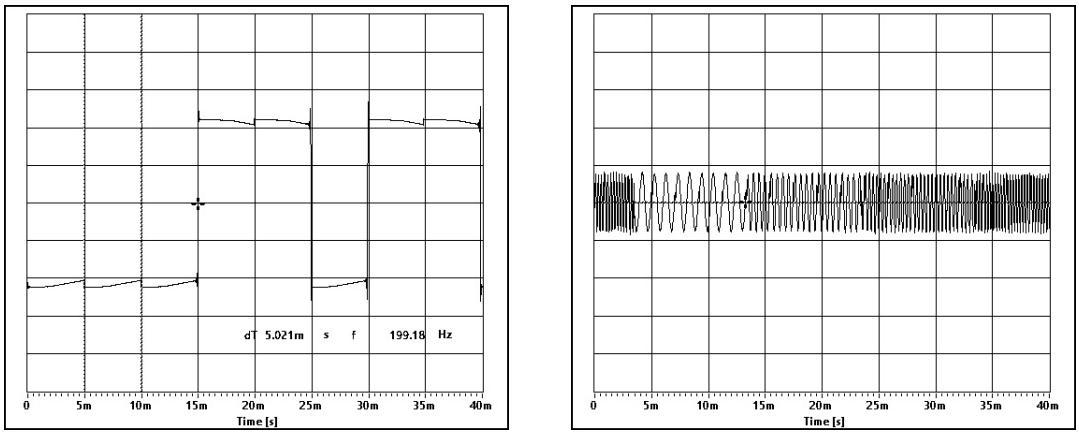
```
            left_output = ptsig1[sample];
         }
         if(level == 2){
            left_output = ptsig2[sample];
         }
         if(level == 3){
            left_output = ptsig3[sample];
         }
         right_output = left_output;

              aic3204_codec_write(left_output, right_output); //write the
                  value to sterio-out
         }
      }
   }
   /* Disble I2S */
   I2S0_CR = 0x00;
   return 0;
}
```

Trong đó, các tần số sóng mang được chọn lần lượt là f0 = 1kHz, f1 = 2kHz, f2 = 3kHz, f3 = 4kHz; tần số lấy mẫu Fs = 48000 mẫu/giây.

Khối quan trọng nhất trong chương trình là khối nằm trong lệnh *switch*. *state* == 0 dùng để khởi tạo giá trị ban đầu cho biến *pre_value* đọc từ ngõ stereo in. *state* == 1 thực hiện nhận biết sự thay đổi bit từ 0 sang 1 hoặc ngược lại, nếu có sự thay đổi bit thì chuyển sang *state* == 2. Khi *state* == 2, theo tính toán, mỗi bit sẽ được lấy mẫu 240 lần. Do đó, mẫu trung tâm của 2 bits liền kề nhau là mẫu thứ 120 và 360. Dựa vào giá trị của hai mẫu này, mức tần số sóng mang tương ứng để truyền mỗi hai bits sẽ được xác định thông qua biến *level*. Trong phần tiếp theo, ứng với mỗi mức tần số ta sẽ truyền với tần số sóng mang tương ứng trong thời gian 2 bits (10ms).

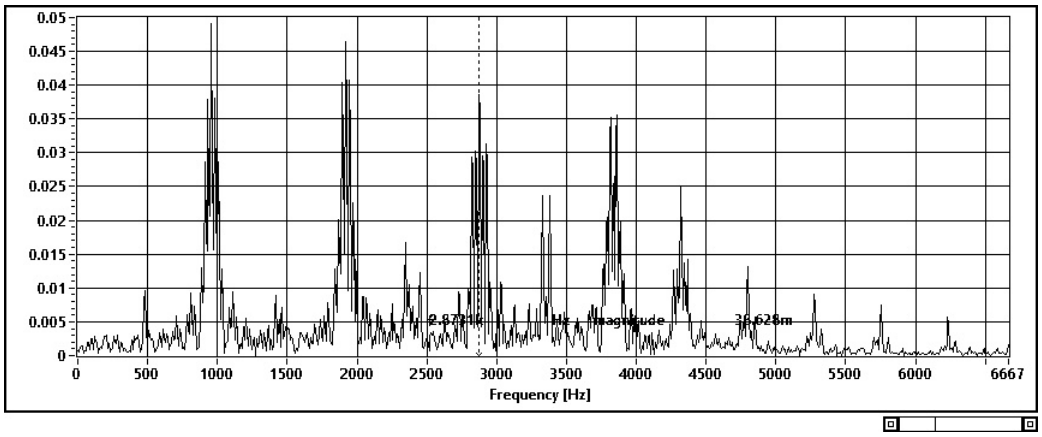
Chuỗi bits ngõ vào trong bài báo cáo này được tạo bằng cách ghi tín hiệu xung vuông vào file "*input_wave.wav*" và dùng phần mềm *WavePad Sound Editor* để chỉnh sửa chuỗi bits thành 00-01-10-11 lặp lại.

Các bits được truyền vào bằng cách lặp lại chuỗi 00-01-10-11. Từ đó, các tần số dùng để điều chế là f0 = 1kHz, f1 = 2kHz, f2 = 3kHz, f3 = 4kHz. Với tốc độ bit là *bit_rate* = 200, tức là thời gian truyền một bit là 5ms, thời gian một tần số mã hóa cho 2 bits sẽ tương ứng với 10ms. Từ hình **3.7(b)**, ta nhận thấy mỗi tần số sẽ thay đổi sau 10ms nên tín hiệu



(a) Tín hiệu bits ngõ vào (b) Tín hiệu sau điều chế 4-FSK

Hình 3.7: Tín hiệu ngõ vào và ra của KIT C5515



Hình 3.8: Phổ của tín hiệu sau điều chế 4-FSK

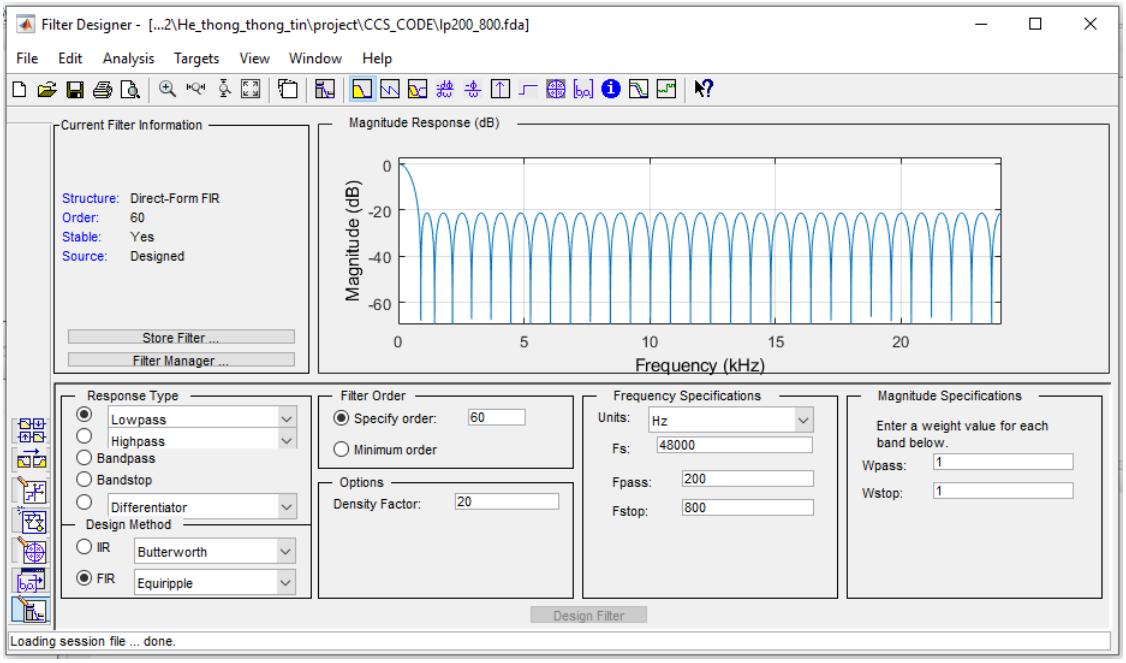
sau điều chế phù hợp về mặt thời gian.

Phổ của tín hiệu sau điều chế được thể hiện trong hình **3.8**. Từ hình thu được, có 4 tín hiệu chủ đạo ở các tần số lần lượt là f0 = 1kHz, f1 = 2kHz, f2 = 3kHz, f3 = 4kHz.

3.2.3 Giải điều chế 4-FSK với C5515

Trong nội dung này, việc giải điều chế 4-FSK sử dụng cấu trúc giải điều chế dạng kết hợp (được mô tả trong hình **1.3(b)**).

Trước tiên, phần mềm MATLAB được sử dụng để tạo bộ lọc số FIR thông thấp có các thông số như sau: $Fpass = 200Hz$, $Fstop = 800Hz$, $Filterorder = 60$.



Hình 3.9: Bộ lọc số thông thấp FIR được thiết kế bằng MATLAB

Hình **3.9** thể hiện các cấu hình để thiết kế bộ lọc. Ở đây, sau quá trình chạy và thử nghiệm nhiều lần trên KIT C5515, một kinh nghiệm được rút ra là chọn bậc của bộ lọc sao cho phù hợp không quá lớn dẫn đến quá nhiều dữ liệu được lưu trên KIT dẫn đến tràn bộ nhớ. Trong trường hợp này, bậc của bộ lọc được chọn là 60. Sau đó, sử dụng công cụ "Targets > Generate C header..." để tạo một file header chứa các thông số của bộ lọc đã thiết kế. Sau một số chỉnh sửa cho phù hợp với ngôn ngữ C, file header chứa thông số của bộ lọc được mô tả như sau:

```
/*
 * Filter Coefficients (C Source) generated by the Filter Design and Analysis
    Tool
 * Generated by MATLAB(R) 9.3 and Signal Processing Toolbox 7.5.
 * Generated on: 27-Aug-2020 16:41:25
 */

/*
 * Discrete-Time FIR Filter (real)
 * -------------------------------
 * Filter Structure : Direct-Form FIR
 * Filter Length    : 61
 * Stable           : Yes
 * Linear Phase     : Yes (Type 1)
 */
```

```
/* General type conversion for MATLAB generated C-code */
/*
 * Expected path to tmwtypes.h
 * C:\Program Files\MATLAB\R2017b\extern\include\tmwtypes.h
 */
/*
 * Warning - Filter coefficients were truncated to fit specified data type.
 *   The resulting response may not match generated theoretical response.
 *   Use the Filter Design & Analysis Tool to design accurate
 *   int16 filter coefficients.
 */
#define LPL 61
Int16 LP[61] = {
     1544,   255,   274,   295,   315,   335,   356,   376,   396,
      416,   436,   456,   475,   494,   512,   530,   547,   563,
      578,   591,   605,   616,   628,   638,   645,   654,   660,
      665,   668,   670,   670,   670,   668,   665,   660,   654,
      645,   638,   628,   616,   605,   591,   578,   563,   547,
      530,   512,   494,   475,   456,   436,   416,   396,   376,
      356,   335,   315,   295,   274,   255,  1544
};
```

Ngoài ra, để giảm bớt khối lượng tính toán trong chương trình, các tín hiệu hình sine tương ứng với 4 tần số sóng mang được tính sẵn sử dụng đoạn code MATLAB sau, và được lưu vào file *ptsig.h*. Việc này cũng đồng thời giảm bớt dung lượng biến sử dụng, thay vì sử dụng các 4 biến mảng dạng *short* (16 bits) như trong quá trình điều chế thì các mảng này được lưu dưới dạng hằng số trong ROM của KIT.

```
ptsig0 = [];
ptsig1 = [];
ptsig2 = [];
ptsig3 = [];
PI = 3.14159264;
nsample = 96;
for i=1:1:96
    a0 = round(1333*sin(2*PI*(i-1)*1000/48000));
    a1 = round(1333*sin(2*PI*(i-1)*2000/48000));
    a2 = round(1333*sin(2*PI*(i-1)*3000/48000));
    a3 = round(1333*sin(2*PI*(i-1)*4000/48000));
    ptsig0 = [ptsig0 a0];
    ptsig1 = [ptsig1 a1];
    ptsig2 = [ptsig2 a2];
    ptsig3 = [ptsig3 a3];
end
fileID0 = fopen('ptsig0.txt','w');
ptsig0file = fprintf(fileID0,'%d, ',ptsig0);
```

```
fileID1 = fopen('ptsig1.txt','w');
ptsig1file = fprintf(fileID1,'%d, ',ptsig1);

fileID2 = fopen('ptsig2.txt','w');
ptsig2file = fprintf(fileID2,'%d, ',ptsig2);

fileID3 = fopen('ptsig3.txt','w');
ptsig3file = fprintf(fileID3,'%d, ',ptsig3);
```

Sau khi tạo bộ lọc thông thấp, quá trình xử lý tín hiệu đã được điều chế 4-FSK được thực hiện thông qua đoạn code sau:

```
/*
 *  AIC3204 Tone _4-FSK Demodulation code
 *  Bui Thanh Tinh
 */
#include "stdio.h"
#include "usbstk5515.h"
extern Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
#define Rcv 0x08
#define Xmit 0x20
#include "math.h"
#include "ptsig.h"
#include <stdio.h>
#include <Dsplib.h>
#include "low_pass_200_800.h"


#define Fs 48000
#define F0 1000
#define F1 2000
#define F2 3000
#define F3 4000
#define PI 3.14159264

Int16 count=-1;
Int16 state=0;
Int16 level=0;

extern void aic3204_codec_read(Int16* left_input, Int16* right_input)
{
      while((Rcv & I2S0_IR) == 0); // Wait for interrupt pending flag
      *left_input = I2S0_W0_MSW_R; // 16 bit left channel received audio data
      //data1 = I2S0_W0_LSW_R;
      *right_input = I2S0_W1_MSW_R; // 16 bit right channel received audio data
      //data2 = I2S0_W1_LSW_R;
}
```

```
extern void aic3204_codec_write(Int16 left_output, Int16 right_output)
{
      /* Write Digital audio */
         while((Xmit & I2S0_IR) == 0); // Wait for interrupt pending flag
      I2S0_W0_MSW_W = left_output; // 16 bit left channel transmit audio data
         I2S0_W0_LSW_W = 0;
         I2S0_W1_MSW_W = right_output; // 16 bit right channel transmit audio
             data
         I2S0_W1_LSW_W = 0;
}


Int16 aic3204_loop_stereo_in1( )
{
    /*  */
    Int16 j, i, ii;
    Int16 data0, data1, data2, data3;
    Int16 sample;
    Int16 left_input, right_input;
    Int16 pre_value, current_value, current_level;

    /* Configure AIC3204 */
   Int16 left_input_matrix[1];
   Int16 dbuffer0[LPL+2]={0};
   Int16 dbuffer1[LPL+2]={0};
   Int16 dbuffer2[LPL+2]={0};
   Int16 dbuffer3[LPL+2]={0};
   Int16 left_output_matrix[1];

   Int16 temp_output_matrix0[1];
   Int16 temp_output_matrix1[1];
   Int16 temp_output_matrix2[1];
   Int16 temp_output_matrix3[1];

   Int16 signal_before_filter[1];
   long int mul0, mul1, mul2, mul3;
    Int16 nsample = 96;

   // Initially set up for AIC3204
    //... Insert the same code with modulation mode to set up AIC3204 ...//

    /* Play Tone */
    for ( i = 0 ; i < 50 ; j++ )
    {
        for ( j = 0 ; j < 1000 ; j++ )
        {
            for ( sample = 0 ; sample < nsample ; sample++ )
            {
```

```
        aic3204_codec_read(left_input_matrix, &right_input);


mul0 = (long int)left_input_matrix[0] * (long int)ptsig0[sample];
mul1 = (long int)left_input_matrix[0] * (long int)ptsig1[sample];
mul2 = (long int)left_input_matrix[0] * (long int)ptsig2[sample];
mul3 = (long int)left_input_matrix[0] * (long int)ptsig3[sample];
for ( ii = 0; ii < 4; ii++ )
{
   if(ii==0)
   {
      signal_before_filter[0] = (Int16)(mul0/666);
      fir(signal_before_filter,     // input
      LP,     // coef
      temp_output_matrix0,  // output
      dbuffer0,  // Z-1 blocks and more
      1,    // number to process
      LPL     // number of parameters
      );
   }
   if(ii==1)
   {
      signal_before_filter[0] = (Int16)(mul1/666);
      fir(signal_before_filter,     // input
      LP,     // coef
      temp_output_matrix1,  // output
      dbuffer1,  // Z-1 blocks and more
      1,    // number to process
      LPL     // number of parameters
      );
   }
   if(ii==2)
   {
      signal_before_filter[0] = (Int16)(mul2/666);
      fir(signal_before_filter,     // input
      LP,     // coef
      temp_output_matrix2,  // output
      dbuffer2,  // Z-1 blocks and more
      1,    // number to process
      LPL     // number of parameters
      );
   }
   if(ii==3)
   {
      signal_before_filter[0] = (Int16)(mul3/666);
      fir(signal_before_filter,     // input
      LP,     // coef
      temp_output_matrix3,  // output
      dbuffer3,  // Z-1 blocks and more
```

```
                1,    // number to process
                LPL     // number of parameters
                );
            }
        }
        data0 = abs(temp_output_matrix0[0]); data1 =
            abs(temp_output_matrix1[0]); data2 =
            abs(temp_output_matrix2[0]); data3 =
            abs(temp_output_matrix3[0]);
        if((data0>data1)&&(data0>data2)&&(data0>data3)) level = 0;
        if((data1>data0)&&(data1>data2)&&(data1>data3)) level = 1;
        if((data2>data0)&&(data2>data1)&&(data2>data3)) level = 2;
        if((data3>data0)&&(data3>data1)&&(data3>data2)) level = 3;

        switch(state)
             {
              case 0:
                 pre_value = level;
                 state = 1;
                 break;
              case 1:
                 current_value = level;
                 // Catch up the change of level
                 if(pre_value!=current_value) state = 2;
              break;
            case 2:
              count++;
              if (count == 240) current_level = level; // Sampling at 240th
                  value of level to avoid instability
              if (current_level == 0) left_output_matrix[0] = 0;
              else if (current_level == 3) left_output_matrix[0] = 1333;
              else if (current_level == 2)
              {
                 if(count < 240) left_output_matrix[0] = 0;
                 else left_output_matrix[0] = 1333;
              }
              else
              {
                 if(count < 240) left_output_matrix[0] = 1333;
                 else left_output_matrix[0] = 0;
              }
              if (count==480) count = 0;
              break;
             }
             aic3204_codec_write(left_output_matrix[0],
                 left_output_matrix[0]);
        }
    }
}
```

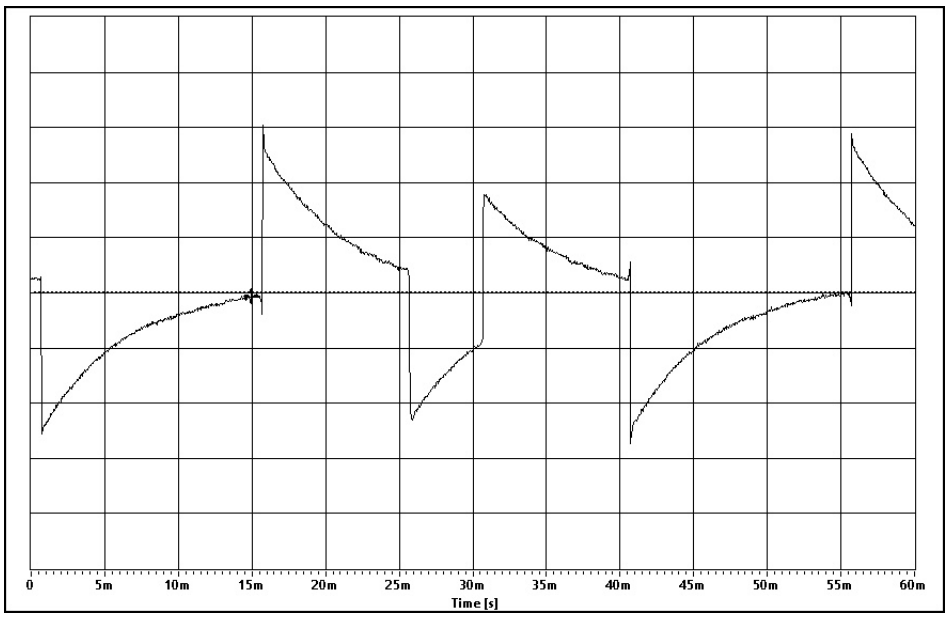
```
    /* Disble I2S */
    I2S0_CR = 0x00;

    return 0;
}
```

Các quá trình diễn ra theo phương pháp giải điều chế kiểu kết hợp. Tín hiệu đầu vào được nhân mới 4 sóng mang hình sine đã tạo sẵn và được lưu trong các biến 32 bits $mul0, mul1, mul2, mul3$. Sau đó, đầu ra cho qua bộ lọc thông thấp (đã được tạo) để loại bỏ các tần số cao, chỉ giữ lại giá trị DC (khi tín hiệu cùng tần số với sóng mang nhân vào). Đối với các sóng mang có tần số khác với tín hiệu đầu vào thì sẽ không có mức DC. Chính vì thế, tín hiệu ngõ vào sẽ có tần số tương ứng với sóng mang khi nhân vào tín hiệu tạo ra mức DC. Từ giá trị tần số này, các mức tần số tương thể hiện qua biến *level*. Trong phần tiếp theo của chương trình, việc giải mã từ các mức level để tạo ra mỗi hai bits tương ứng được thực hiện. Cấu trúc *switch case* tương tự với quá trình điều chế. Lưu ý quan trong là biến $current\_level$ được thêm vào để chọn tại mẫu trung tâm 240th nhằm tránh sự mất ổn định khi vừa mới thay đổi mức tần số. Các bits 0 được thể hiện qua mức DC 0 và các bits 1 được thể hiện qua việc truyền giá trị 1333 qua ngõ ra đủ để KIT có thể nhận biết được. Sau khi hoàn thành việc xuất các giá trị DC qua stereo out, quá trình giải điều chế hoàn tất.

Cho tín hiệu 4-FSK được ghi lại trong file $modulated_sig.wav$ (ở quá trình điều chế) qua KIT C5515 chứa đoạn code giải điều chế. Sau đó, tín hiệu chuỗi bits ngõ ra của KIT sẽ được truyền lại PC thông qua cổng Stereo Out. Kết quả quan sát dạng sóng của tín hiệu thu được trong một khoảng 60ms được thể hiện trên hình **3.10**. Từ dạng sóng ta có thể dễ dàng rút ra được chuỗi bits là 00-01-10-11 (mỗi bit truyền trong 5ms).

SO SÁNH VỚI MATLAB:

Trong cả hai quá trình điều chế và giải điều chế 4-FSK thực hiện trên KIT C5515, các kết quả thể hiện về chuỗi bits cũng như dạng sóng của tín hiệu sau điều chế đều giống với các kết quả thực hiện trên MATLAB trong nội dung chương **2**. Tuy nhiên, các tín hiệu tạo ra có thêm những thành phần nhiễu sinh ra bởi mạch KIT và đường truyền thông qua cổng stereo; chuỗi bits ngõ ra sau khi giải điều chế đúng với chuỗi bits ngõ vào nhưng dạng xung vuông bị méo dạng.



Hình 3.10: Tín hiệu chuỗi bits thu được sau giải điều chế 4-FSK

Chương 4

KẾT LUẬN

Điều chế dịch tần FSK được ứng dụng để truyền tín hiêu số bằng cách thay đổi tần số sóng mang giữa các tần số rời rạc xác định. Bài báo cáo này đã trình bày nền tảng cơ bản của điều chế và giải điều chế FSK, cũng như ứng dụng vào các ví dụ cụ thể trong mô phỏng MATLAB và KIT C5515 xử lý thực nghiệm. Các kiến thức đạt được:

- Nắm được các kiến thức lý thuyết điều chế và giải điều chế FSK.
- Áp dụng điều chế và giải điều chế 4-FSK trong các mô phỏng MATLAB.
- Thiết lập và kết nối hệ thống xử lý tín hiệu thông qua TMS320C5515 và máy tính.
- Làm quen với bộ KIT C5515 được thiết kế chuyên dùng cho xử lý tín hiệu.
- Kỹ năng giải quyết liên tiếp những lỗi xảy ra trong quá trình lập trình.
- Lập trình bộ điều chế và giải điều chế 4-FSK trên KIT C5515.
- So sánh đánh giá kết quả của cùng một bài toán 4-FSK thực hiện trên MATLAB và KIT C5515.

Những hạn chế và thiếu sót của bài báo cáo:

- Chưa thực hiện lập trình tính tỷ lệ lỗi bit.
- Chưa thực hiện tạo tín hiệu tương ứng với chuỗi bits được tạo ngẫu nhiên để làm đầu vào cho bộ điều chế.

TÀI LIỆU THAM KHẢO

[1] Vũ Đình Thành, "Chương 5 Thông tin số", *Giáo trình nguyên lý thông tin tương tự số*, Tái bản lần thứ 2, Nhà xuất bản Đại học quốc gia TP Hồ Chí Minh, 2012, ISBN: 978-604-73-0874-3.

[2] A. B. Carlson, P. B. Crilly, "Communication Systems", McGraw-Hill, 2010, 5th Edi. or later, ISBN 978-0-07-338040-7.

[3] Leon W.Couch, II, "Digital and Analog Communication Systems", Prentice Hall, 2001, 6th Edi. or later, ISBN 0-13- 089630-6.

[4] Texas Instruments, "C5515 eZDSP USB Stick Development Tool description and features", https://www.ti.com/tool/TMDX5515EZDSP, retrieved on February, 27th 2014.

[5] CHOI, J.H., CHANG, J.H. and LEE, S.R., 2010. TMS320C55x DSP Library Programmer's Reference TMS320C55x DSP Library Programmer's Reference, 2002. *IEICE transactions on fundamentals of electronics, communications and computer sciences,* 93(9), pp.1684-1687.

[6] Md. Salim Raza. MATLAB code for BFSK modulation and demodulation (`https://www.mathworks.com/matlabcentral/fileexchange/44821-matlab-code-for-fsk-modulation-and-demodulation`), MATLAB Central File Exchange, 06 Jun 2018.

[7] Alfred Hero, EECS 452: Digital Signal Processing Design Laboratory - Labs 2, University of Michigan, 2014.