



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 07 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 dụng 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. Các so sánh và nhận xét giữa các tín hiệu điều chế thực tế trên KIT C5515 và tín hiệu trong mô phỏng MATLAB được đưa ra để xây dựng các kết luận.

Mục lục

LỜI MỞ ĐẦU	1
DANH SÁCH HÌNH VẼ	3
1 LÝ THUYẾT ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ FSK	4
1.1 Điều chế và giải điều chế BPSK	4
1.1.1 Điều chế BFSK	4
1.1.2 Giải điều chế BFSK	6
1.2 Ưu, nhược điểm và các ứng dụng của điều chế FSK	7
2 ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ 4-FSK VỚI MATLAB	9
2.1 MATLAB code thực hiện điều chế 4-FSK	9
2.2 MATLAB code thực hiện giải điều chế 4-FSK	14
3 ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ 4-FSK VỚI TMS320C5515	17
3.1 Giới thiệu TMS320C5515	17
3.2 Thực hiện điều chế và giải điều chế 4-FSK với TMS320C5515	19
3.2.1 Một số chuẩn bị và kết nối	19
3.2.2 Code thực hiện điều chế 4-FSK	21
4 KẾT LUẬN	31
4.1 Kiến thức đạt được	31

Danh sách hình vẽ

1.1	Các dạng sóng dịch tần dựa theo pha của tín hiệu	5
1.2	Sơ đồ khối điều chế BFSK	6
1.3	Sơ đồ khối các bộ giải điều chế BFSK	7
2.1	Chuỗi bit tín hiệu cần điều chế.	11
2.2	Các mức tần số tương ứng với chuỗi bits.	12
2.3	Tín hiệu sau điều chế 4-FSK.	13
2.4	Chuỗi bits thu được ở bộ thu.	16
3.1	KIT xử lý số tín hiệu TMDX5515eZdsp	18
3.2	Sơ đồ khối chức năng TMS320C5515	19
3.3	Sơ đồ kết nối KIT C5515 với PC	20
3.4	Kết nối C5515 với máy tính	20
3.5	Sơ đồ khối thực hiện điều chế trên KIT C5515.	21
3.6	Tín hiệu ngõ vào và ra của KIT C5515	29
3.7	Phổ của tín hiệu sau điều chế 4-FSK	30

Chương 1

LÝ THUYẾT ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ FSK

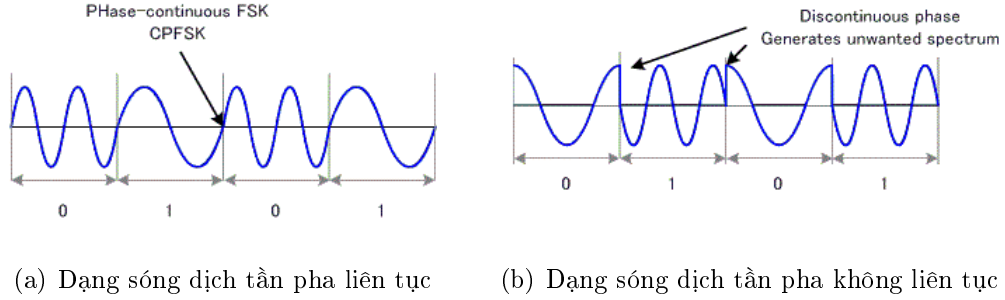
Điều chế số dịch tần FSK (Frequency Shift Keying) là dạng điều chế truyền chuỗi bit nhị phân dưới dạng tín hiệu có các mức tần số khác nhau. Với phương thức truyền từng bit nhị phân BFSK (Binary FSK), tín hiệu sau điều chế sẽ có hai mức tần số ứng với hai bit nhị phân '0' và '1'. Với phương thức truyền 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 dịch 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).



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 $b(t) = \text{logic1}$ (hoặc $d(t) = +1$)
- $f = f_L$ khi $b(t) = \text{logic0}$ (hoặc $d(t) = -1$)

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

$$v_{BFSK}(t) = A_c \cdot \cos [2\pi(f_0 + d(t) \cdot \Delta f)t] \quad (1.1)$$

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

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

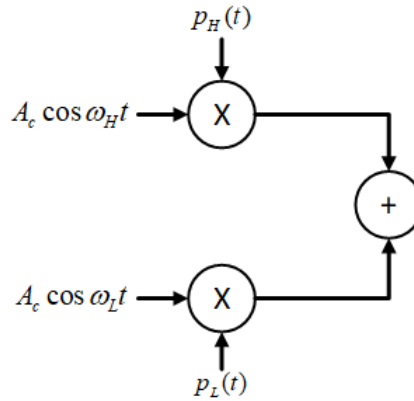
$$v_{BFSK}(t) = A_c \cdot p_H(t) \cdot \cos 2\pi f_H t + A_c \cdot p_L(t) \cdot \cos 2\pi f_L t \quad (1.2)$$

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 .

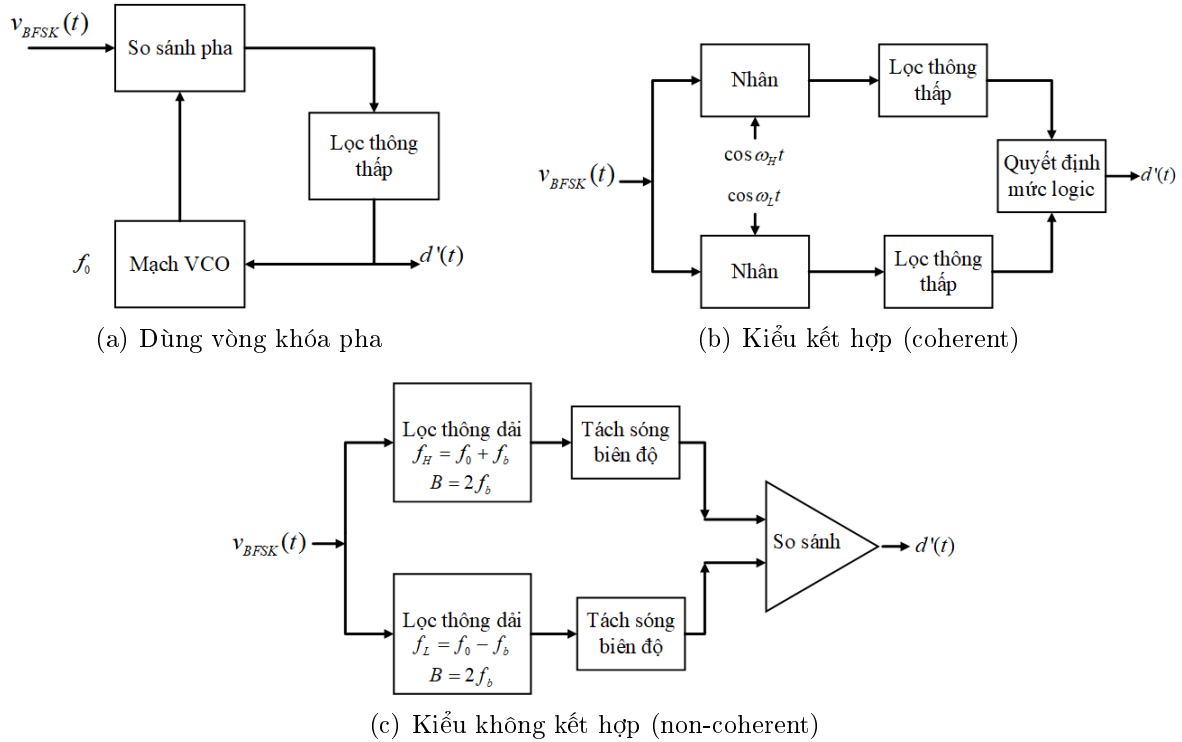
Nếu chọn f_H và f_L quá gần nhau, sẽ xảy ra hiện tượng chồng lấn giữa hai vùng phổ; ngược lại, nếu cách nhau quá xa sẽ gây lãng phí dải tần. Phương pháp được sử dụng là chọn f_H và f_L sao cho cả búp phổ chính có chung điểm cắt như hình ???. Khi đó, băng thông của tín hiệu BFSK được cho bởi công thức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

$$BW \approx 4f_b = \frac{4}{T_b} \quad (1.3)$$

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).



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. Điều chế FSK cũng không phải là một ngoại lệ, 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.

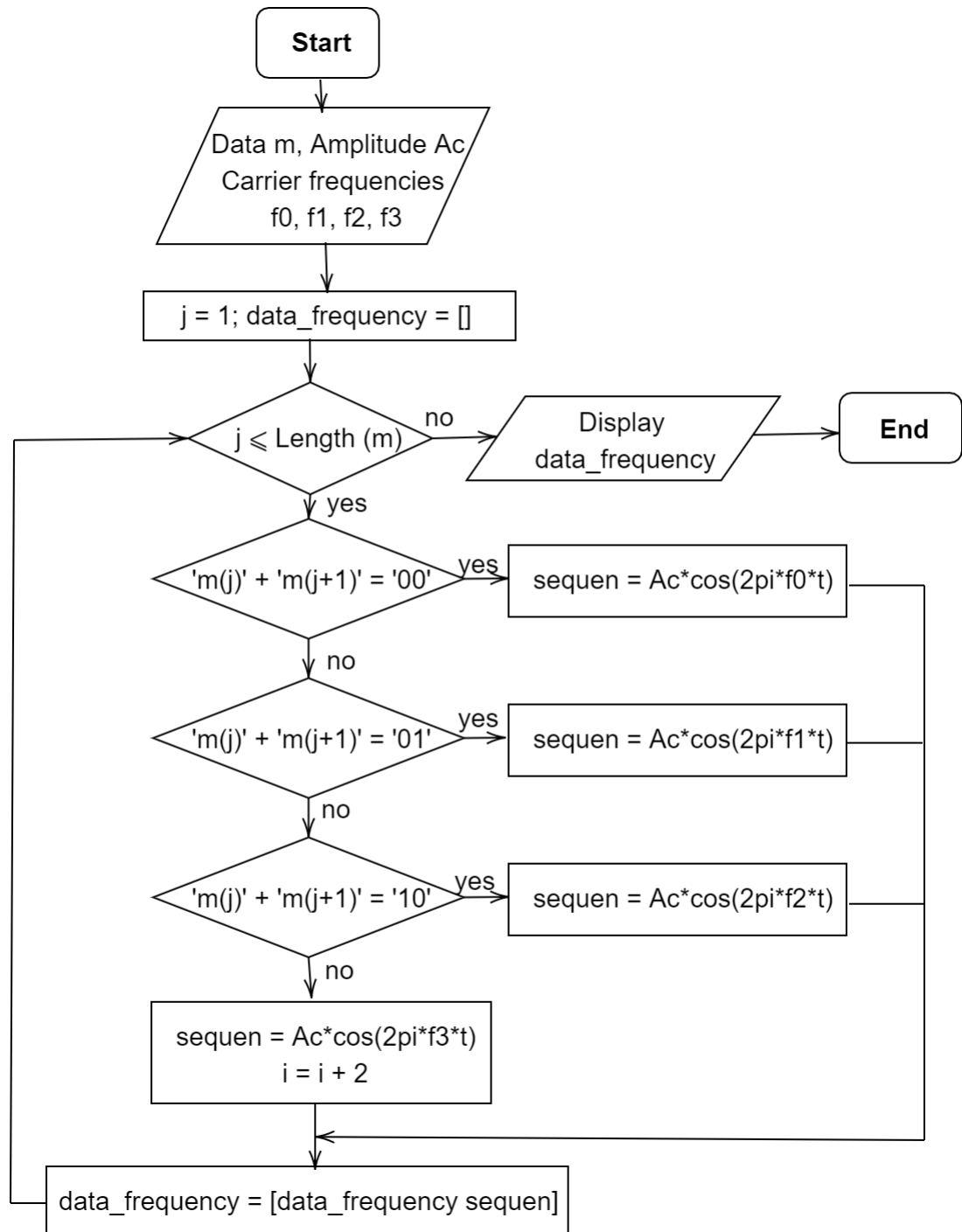
Nội dung ví dụ như sau: Cho một chuỗi các bits tín hiệu m , sóng mang có biên độ $A_c = 5$ không đổi, các mức tần số tương ứng với chuỗi bit được cho như bảng 2.1. Thực hiện mô phỏng MATLAB thể hiện tín hiệu sau điều chế 4-FSK và chuỗi tín hiệu thu được.

Bảng 2.1: Các tần số thể hiện chuỗi bit

Mức tần số	Tần số (MHz)	Chuỗi bit
1	2	00
2	4	01
3	6	10
4	8	11

2.1 MATLAB code thực hiện điều chế 4-FSK

Sơ đồ khối điều chế 4-FSK:



Hiển thị tín hiệu

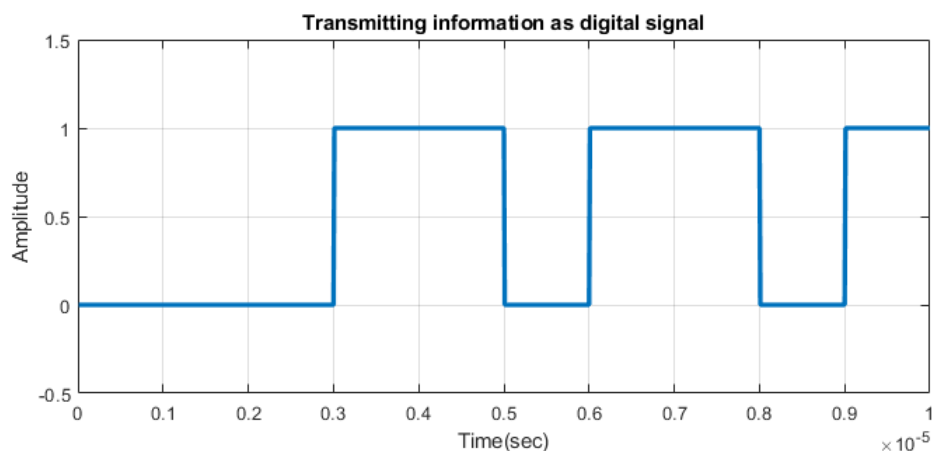
```

clc
clear all

%% Information
unit = 0.000001;
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);
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^{-6} 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.

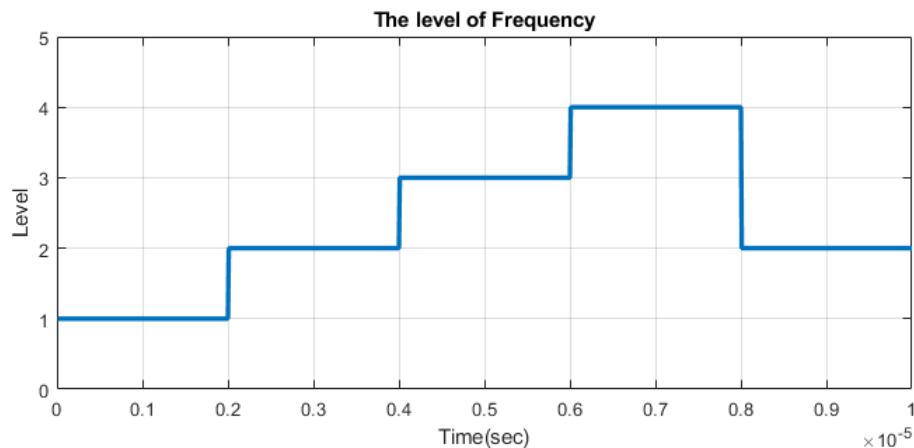
Hiển thị mức tần số cần mã hóa

```

%% 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);
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 carrier signal

```

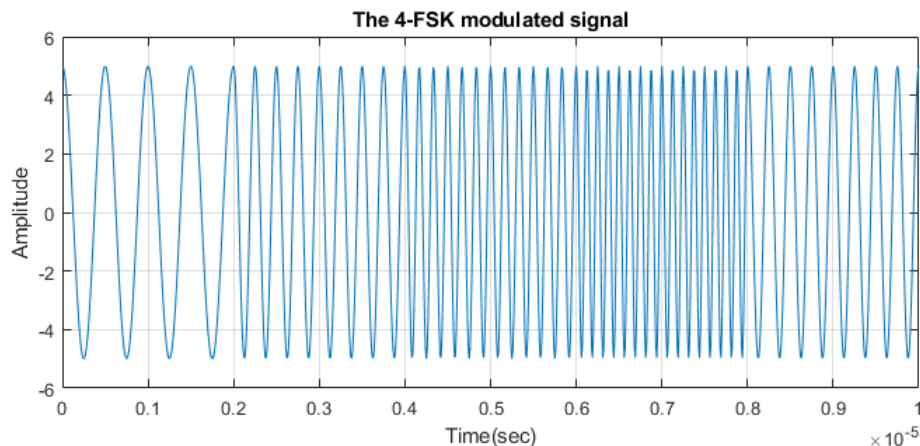
```

f0 = 2*1/unit; % Frequency for 00
f1 = 4*1/unit; % Frequency for 01
f2 = 6*1/unit; % Frequency for 10
f3 = 8*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);
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');

```

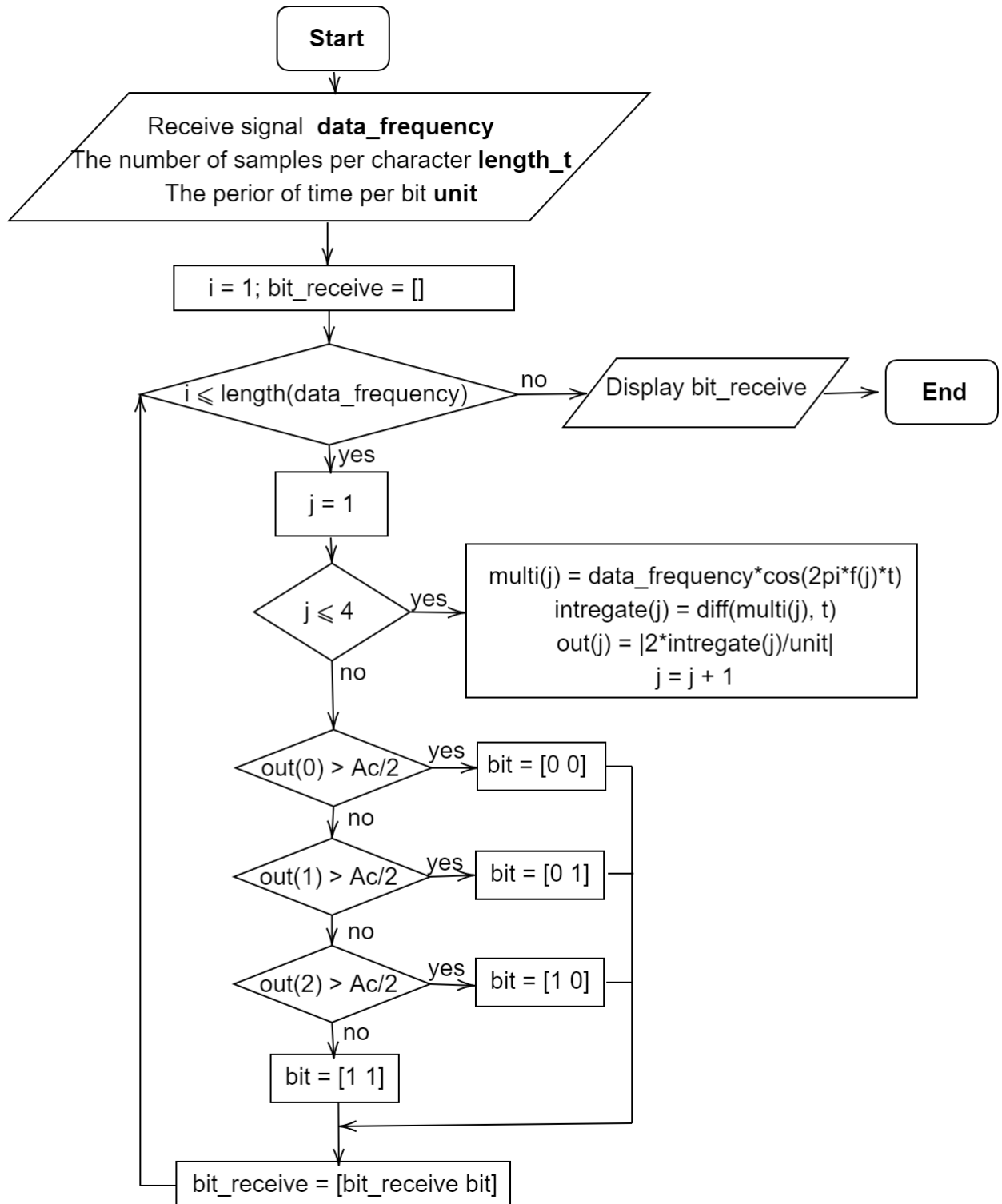
Biến ma trận *data_frequency* dùng để lưu giá trị thể hiện tín hiệu sau điều chế 4-FSK. Các tần số f_0 , f_1 , f_2 và f_3 là các mức tần số tương ứng với các chuỗi bits mã hóa trong bảng 2.1. Tín hiệu dùng để truyền đi trên đường truyền sau khi điều chế 4-FSK của chuỗi bits được thể hiện trên hình 2.3.



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

2.2 MATLAB code thực hiện giải điều chế 4-FSK

Sơ đồ khối giải điều chế 4-FSK:



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 tín hiệu hình sin có các mức tần số trong bảng 2.1. Khi tín hiệu có cùng mức tần 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 tổng 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.

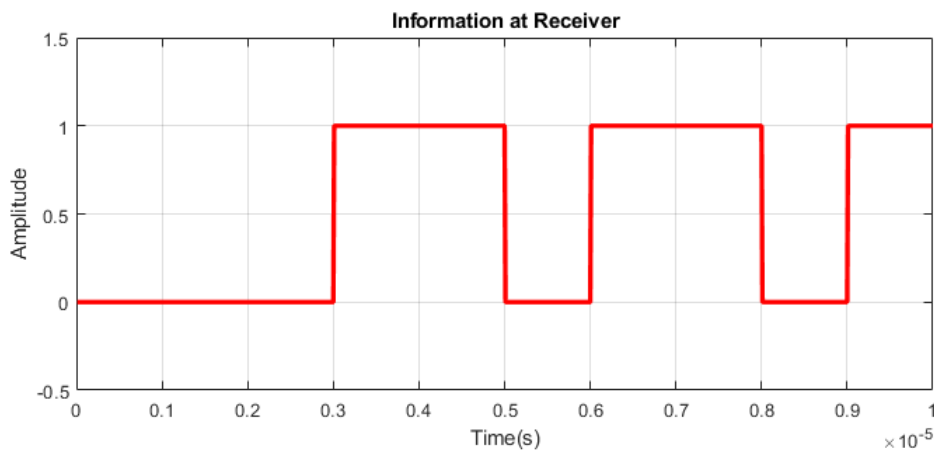
Hiển thị tín hiệu sau điều chế

```

%% 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 2.4. 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 2.4: Chuỗi bits thu được ở bộ thu.

Chương 3

ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ 4-FSK VỚI TMS320C5515

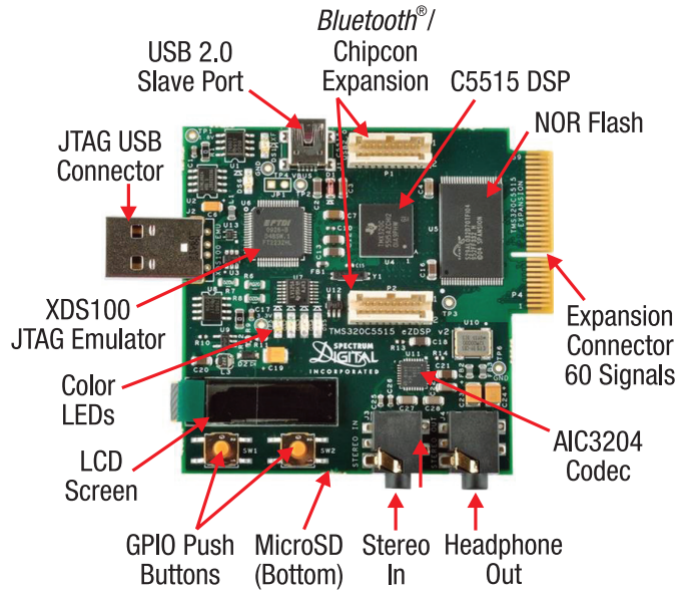
TMS320C5515 được đánh giá là vi xử lý xử lý số tín hiệu 16-bit công suất thấp nhất. Trong chương này, KIT xử lý số tín hiệu C5515eZdsp và phần mềm tạo dao động AudMeS-20080606 được giới thiệu. Sau đó, chúng sẽ được ứng dụng trong điều chế và giải điều chế 4-FSK với cùng bài toán trên MATLAB đã được trình 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:

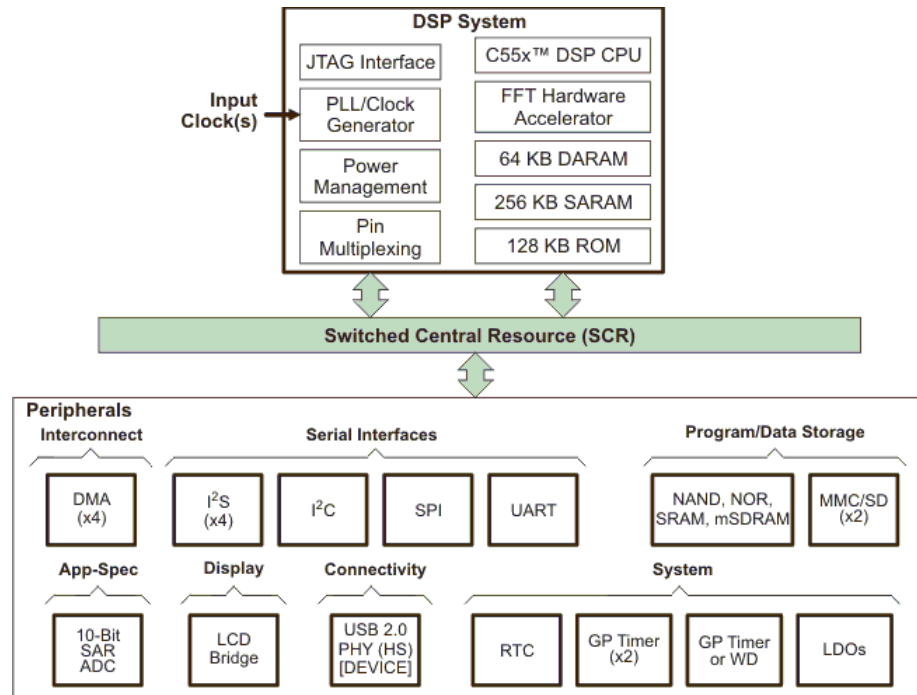
- Hiệu suất cao, công suất tiêu thụ thấp.



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.

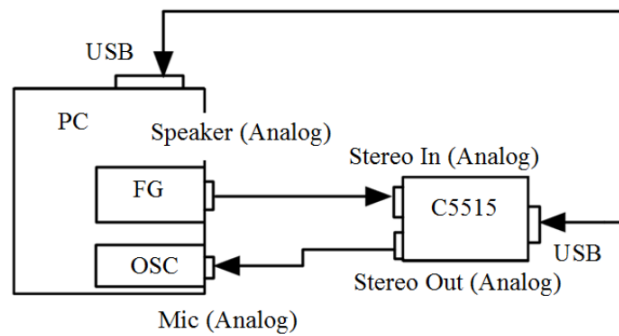
3.2 Thực hiện điều chế và giải điều chế 4-FSK với TMS320C5515

3.2.1 Một số chuẩn bị và kết nối

Sơ đồ kết nối KIT C5515 với PC được thể hiện trên hình 3.3.

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

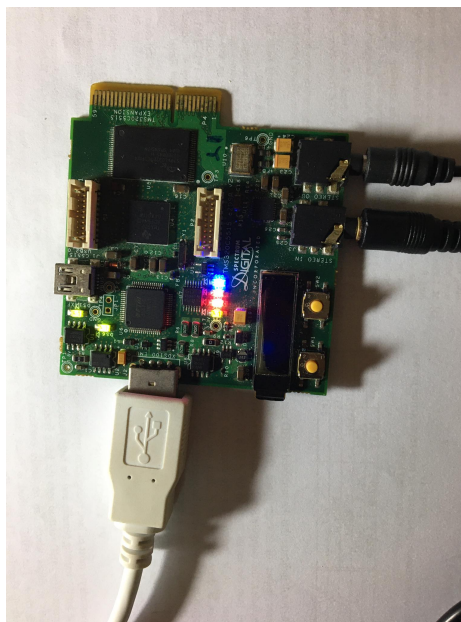


Hình 3.3: Sơ đồ kết nối KIT C5515 với PC

Input 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:

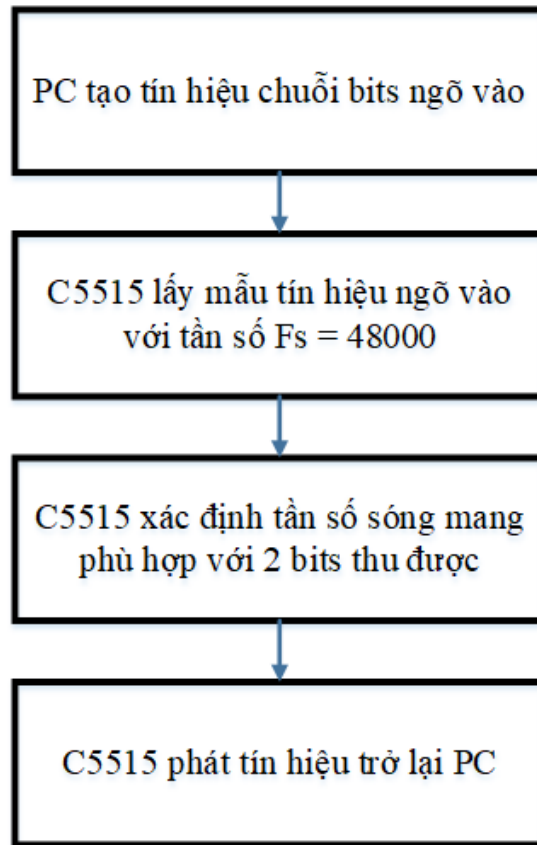


(a) Kết nối với KIT C5515



(b) Kết nối IN và OUT của PC

Hình 3.4: Kết nối C5515 với máy tính



Hình 3.5: Sơ đồ khối thực hiện điều chế trên KIT C5515.

3.2.2 Code thực hiện điều chế 4-FSK

Sơ đồ khối thực hiện điều chế 4-FSK trên KIT C5515 thông qua kết nối với máy tính được mô tả như hình 3.5:

Chương trình sử dụng các thư viện có sẵn được cung cấp bởi TI bao gồm: *usbstk.h*, *aic3204.h*. Nội dung của chương trình chính *main.c* thực hiện điều chế 4-FSK như sau:

```

/*
 * 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 *aic3204_tone_headphone* (xác định trong file *aic3204_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. Bên cạnh đó, hàm *aic3204_loop_stereo_in1* xác định trong file *aic3204_loop_stereo_in1.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"

/* ----- */
/*
*
*   AIC3204 Loop
*   Output input from STEREO IN 1 through the HEADPHONE jack
*
* ----- */

#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, data1, data2, data3, data4;
    Int16 sample, left_output, right_output;
    Int16 left_input, right_input;
    Int16 pre_value, current_value;
    /* Configure AIC3204 */
    //signal = (double*) malloc( nsample * sizeof(double));
    short ptsig0[100];
    short ptsig1[100];
    short ptsig2[100];
    short ptsig3[100];
    short twobit[2];

    /* Create 4 sine signals*/
    Int16 nsample = 100; 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 offset = 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 oversampling
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 );

/* Play Tone */
for ( i = 0 ; i < 50 ; i++ )
{
    for ( j = 0 ; j < 1000 ; j++ )
    {
        for ( sample = 0 ; sample < nsample ; sample++ )
        {
            aic3204_codec_read(&left_input, &right_input);
            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;
                    }
                case 2:
                    // ... (rest of the code for 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(left_input>0){
    left_output = ptsig0[sample]; //16383
}
if(left_input<0){
    left_output = ptsig3[sample];
}*/

    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;
    //left_output = 0;
    //right_output = 0;
    aic3204_codec_write(left_output, right_output);

}

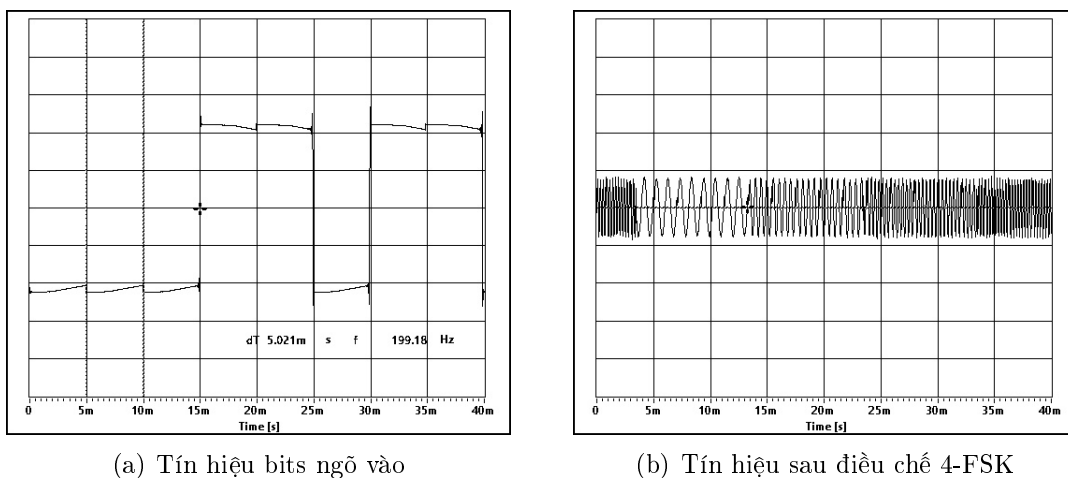
}

/* Disable I2S */
I2S0_CR = 0x00;

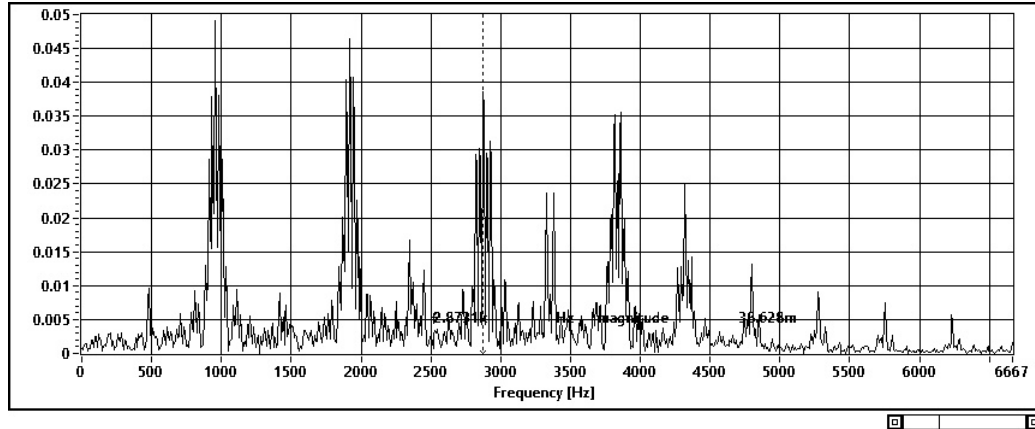
return 0;
}

```

Trong đó, các tần số sóng mang được chọn lần lượt là $f_0 = 1\text{kHz}$, $f_1 = 2\text{kHz}$, $f_2 = 3\text{kHz}$, $f_3 = 4\text{kHz}$; tần số lấy mẫu $F_s = 48000$ mẫu/giây. Khối quan trọng nhất trong chương trình là khối nằm trong lệnh *switch*. Ở dòng lệnh này các dòng code mô tả quá trình lấy mẫu tín hiệu chuỗi bit ngõ vào để xác định tần số sóng mang tương ứng. Hình 3.6 mô tả tín hiệu chuỗi bits được tạo bởi máy tính và tín hiệu ngõ ra từ KIT C5515 sau khi điều chế 4-FSK.



Hình 3.6: Tín hiệu ngõ vào và ra của KIT C5515



Hình 3.7: Phổ của tín hiệu sau điều chế 4-FSK

Các bits được truyền vào bằng cách lặp lại chuỗi 00-01-10-11. Từ đó, các tần số dùng để điều chế là $f_0 = 1\text{kHz}$, $f_1 = 2\text{kHz}$, $f_2 = 3\text{kHz}$, $f_3 = 4\text{kHz}$. Với tốc độ bit là $\text{bitrate} = 200$, 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 3.6(b), ta nhận thấy mỗi tần số sẽ thay đổi sau 10ms nên tín hiệu sau điều chế phù hợp về mặt thời gian.

Phổ của tín hiệu sau điều chế được thể hiện trong hình 3.7. Trong hình 3.7, có 4 tín hiệu chủ đạo ở các tần số lần lượt là $f_0 = 1\text{kHz}$, $f_1 = 2\text{kHz}$, $f_2 = 3\text{kHz}$, $f_3 = 4\text{kHz}$.

Chương 4

KẾT LUẬN

4.1 Kiến thức đạt được

Điều chế dịch tần FSK được ứng dụng để truyền tín hiệu 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 C5515eZdsp 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 C5515eZdsp được thiết kế chuyên dùng cho xử lý tín hiệu.
- Lập trình bộ điều chế 4-FSK trên KIT C5515eZdsp.

Những hạn chế và thiếu sót của bài báo cáo:

- Chưa thực hiện được code giải điều chế 4-FSK. Nguyên nhân là do chưa tìm được hàm thay thế cho hàm tính tích phân rời rạc (trapz) trong MATLAB.

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] TMS320C55x DSP Library Programmer's Reference, Texas Instrument