**ĐẠI HỌC QUỐC GIA TP.HCM**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HCM**

**CHƯƠNG TRÌNH KĨ SƯ CHẤT LƯỢNG CAO VIỆT PHÁP**

# BÁO CÁO PROJECT:

# MÔ PHỎNG ĐIỀU CHẾ VÀ GIẢI ĐIỀU CHẾ 4-QAM DÙNG MATLAB VÀ USBSTK5515

**GVHD: GS.TS LÊ TIẾN THƯỜNG**

**NHÓM: 7** **ĐỀ TÀI: 12**

**THÀNH VIÊN:**

| STT | HỌ VÀ TÊN | MSSV |
|---|---|---|
| 31 | Lương Hoài Thiện | 1513202 |
| 29 | Phạm Thái Hoàng | 1511147 |
| 28 | Nguyễn Hữu Khoa Minh | 1511993 |

**THÁNG 11/2018**

# LỜI NÓI ĐẦU

Ngành Viễn thông của Việt Nam đang đứng trước một cơ hội để có thể phát triển mạnh mẽ khi mà vào năm 2020 "Việt Nam sẽ là những nước đầu tiên triển khai Mạng di động thế hệ thứ 5 (5G) trên thế giới" – trích lời bộ trưởng Thông tin và Truyền thông ông Nguyễn Mạnh Hùng tại Hội thảo "Đổi mới sáng tạo Việt Nam" tổ chức vào ngày 14/11/2018. Mạng 5G được kì vọng sẽ tạo ra cuộc cách mạng về tốc độ trong các thiết bị di động khi mà tốc độ lý thuyết của mạng 5G có thể đạt đến 10 Gbps, qua đó tạo điều kiện thúc đẩy sự phát triển của các ngành công nghệ mới cần tốc độ truyền dữ liệu cao như xe tự hành, mạng lưới vạn vật kết nối (Internet of Things) với hàng ngàn, hàng tỉ bộ cảm biến có thể giao tiếp với nhau,…

Là những sinh viên thuộc khối ngành Điện, việc nắm bắt xu hướng của thế giới và đất nước để chuẩn bị hành trang cần có để đáp ứng nhu cầu tương lai trong thời gian học Đại học là điều cần thiết. Nhưng mà để có thể nắm bắt được những công nghệ mới như 5G, nền tảng các công nghệ cơ bản của mỗi sinh viên là điều cực kì quan trọng. Do đó, trong khoảng thời gian từ tháng 8 đến tháng 11, dưới sự hướng dẫn của thầy GS. TS Lê Tiến Thường, nhóm đã thực hiện nghiên cứu lý thuyết và thực hiện mô phỏng quá trình Điều chế trực pha 4 mức (4 Quadrature Amplitude Modulation) trên MATLAB và kit TMS 320C5515 eZdsp$^{TM}$ USB Stick.

Dưới sự hỗ trợ về mặt phần cứng và kiến thức của thầy GS. TS Lê Tiến Thường, nhóm đã hoàn thành dự án đúng hạn và đạt được các mục tiêu đề ra. Nhưng do thời gian có hạn, cũng như kiến thức còn nhiều hạn chế, nhóm chắc chắn không tránh khỏi những thiếu sót. Mong rằng nhóm sẽ nhận được các góp ý quý báu từ Thầy cũng như các bạn để có thể hiểu rõ hơn về công nghệ 4-QAM này.

Nhóm chân thành cảm ơn sự hỗ trợ và giảng dạy nghiêm khắc của Thầy trong suốt thời gian qua và cảm ơn những người bạn cùng nhóm vì đã cùng nhau hoàn thành dự án này.

TP Hồ Chí Minh, ngày 21/11/2018

Nhóm sinh viên thực hiện

# TÓM TẮT

Trong dự án môn học, nhóm chúng tôi đã thực hiện nghiên cứu lý thuyết cơ bản về Điều chế biên độ trực pha (Quadrature Amplitude Modulation-QAM) và một phân loại là Điều chế biên độ trực pha 4 mức – 4QAM. Sau đó, với những lý thuyết đã tìm hiểu, chúng tôi đã thực hiện mô phỏng quá trình điều chế theo phương pháp này trên SIMULINK của MATLAB và trên kit TMS 320C5515 eZdsp$^{TM}$ USB Stick đến từ nhà sản xuất Texas Instrument. Sau đó, để có thể đánh giá quá trình, nhóm cũng đã thực hiện giải điều chế để tìm lại dạng sóng tín hiệu ban đầu. Dưới đây là bảng báo cáo chi tiết kết quả thực hiện dự án này trong suốt 3 tháng vừa qua.

# BẢNG PHÂN CHIA CÔNG VIỆC

| | |
|---|---|
| Lương Hoài Thiện | Tìm hiểu lý thuyết, viết chương trình điều chế trên kit và phân tích kết quả bằng Multi-Instrument, làm báo cáo. |
| Nguyễn Hữu Khoa Minh | Tìm hiểu lý thuyết, viết chương trình mô phỏng điều chế và phân tích kết quả trên MATLAB, làm báo cáo. |
| Phạm Thái Hoàng | Tìm hiểu lý thuyết, viết chương trình giải điều chế các tín hiệu nhận được từ kit và phân tích kết quả bằng MATLAB, làm báo cáo. |

# DANH MỤC HÌNH ẢNH

# DANH MỤC BẢNG BIỂU

# MỤC LỤC

# I. Lý thuyết về Điều biên trực pha - Quadrature Amplitude Modulation (QAM):

### 1) Định nghĩa QAM:

Điều biên trực pha QAM hay còn được gọi là điều chế QASK (Quadrature Amplitude Shift Keying) là sự kết hợp giữ điều pha và điều biên của sóng mang đối với chuỗi số.

Tên gọi "trực pha" bắt nguồn từ việc 2 thành phần sóng mang của nó có dạng hình sin cùng tần số những vuông pha với nhau. Một thành phần được gọi là I hay thành phần đồng pha (In-phase component), và thành phần còn lại được gọi là Q hay thành phần vuông pha (Quadrature component).

Tín hiệu của Điều biên trực pha có dạng tổng quát là:

$$s(t) = I(t)\cos(\omega_c t) - Q(t)\cos(\omega_c t) = Re\{[I(t) + jQ(t)]e^{j2\pi f_c t}\} \quad [1]$$

Với:

$j^2 = -1$; $f_c$ là tần số sóng mang và Re{} là phần thực của số phức.

I(t) và Q(t) sẽ có dạng khác nhau, tùy thuộc vào mức (4, 16, 256, …, $4^n$) hoặc loại (số hoặc tương tự) của QAM mà ta đang xét.

Sau đây, ta sẽ khảo sát các tính chất của 4-QAM, 1 dạng của Điều biên trực pha QAM.

### 2) Phân tích trên miền tần số của QAM:

Trên miền tần số, QAM có dạng phổ giống như của điều chế DSB-SC và có công thức là:

$$S(f) = \frac{1}{2}[M_I(f - f_0) + M_I(f + f_0) + \frac{j}{2}[M_Q(f - f_0) - M_Q(f + f_0)] \quad [2]$$

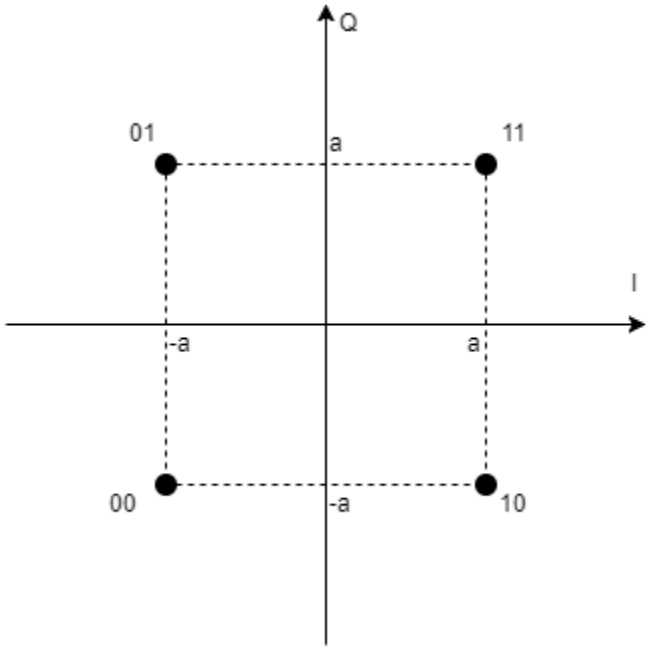
Với S(f), $M_I(f)$ và $M_Q(f)$ là các biến đổi Fourier tương ứng của s(t), I(t) và Q(t) trong công thức [1]

### 3) Điều biên trực pha 4-QAM:

#### a) Các điểm trạng thái pha trong không gian tín hiệu:

4-QAM có nghĩa là sẽ có 4 trạng thái của tín hiệu s(t) được biểu thị bởi 2 bits số nhị phân. Như vậy, mỗi kí hiệu (symbol) được truyền đi sẽ tương ứng với 1 chuỗi số liệu dài 2 bits: 00, 10, 01 hoặc 11.

Các trạng thái tín hiệu của 4-QAM được vẽ như hình:




*Hình 1: Các điểm trạng thái pha và giá trị bit tương ứng trong 4-QAM*

Trục hoành của đồ thị liên quan đến thành phần đồng pha I của sóng mang ( $\cos(\omega_c t)$) và biên độ của I được thể hiện qua giá trị hoành độ của điểm. Còn trục tung thì liên quan đến thành phần vuông qua Q ( $-sin(\omega_c t)$ ) và biên độ của Q được thể hiện qua giá trị tung độ của điểm.

Biên độ và góc pha của tín hiệu 4-QAM lần lượt được xác định bằng độ lớn của vector nối gốc tọa độ với điểm đó và góc pha hợp bởi vector đó và trục hoành.

Trong trường hợp 4-QAM, biên độ của các vector có giá trị là $a\sqrt{2}$ và các góc pha có thể là $45°, 135°, 225°, 315°$.

b) **Biểu thức của tín hiệu 4-QAM:**

$$s(t) = \sqrt{\frac{2}{T_s}}(k_1 a \cos(2\pi f_c t) - k_2 a \sin(2\pi f_c t)), 0 < t \leq T_s \quad [3]$$

Với:

Tham số $T_s$ là thời gian lấy mẫu tín hiệu.

Tham số $k_1$, $k_2$ có thể nhận giá trị là $\pm 1, \pm 3, \ldots, \pm(\sqrt{M} - 1)$ với M là một số nguyên lũy thừa của 4 ( 4, 16, 64, …).

Tham số a có liên quan đến năng lượng trung bình chuẩn hóa của mỗi trạng thái ($E_s$) theo công thức sau:

$$a = \sqrt{\frac{3E_s}{2(M-1)}} \quad [4]$$

Trong trường hợp 4-QAM thì M = 4 → $k_1$ , $k_2$ có thể nhận giá trị $\pm 1$ và $a = \sqrt{0.5E_s}$.

c) **Sơ đồ điều chế và giải điều chế 4-QAM:**

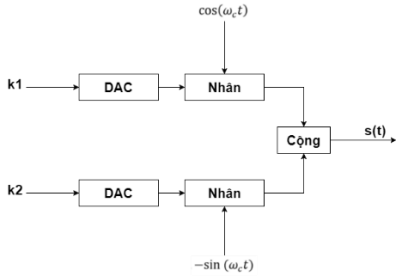
**i) Điều chế 4-QAM:**

Chuỗi thông tin mà ta muốn gửi đi sẽ được chia làm nhiều chuỗi con có độ dài là 2 bit. Gọi $k_1$,$k_2$ lần lượt là giá trị của bit đầu tiên và bit thứ hai trong từng chuỗi con đó.

| … | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | … |
|---|---|---|---|---|---|---|---|---|---|
| | | | $k_1$ | $k_2$ | | | | | |

*Bảng 1: Minh họa việc biểu diễn thông tin đầu vào để chuẩn bị điều biến bằng 4-QAM*

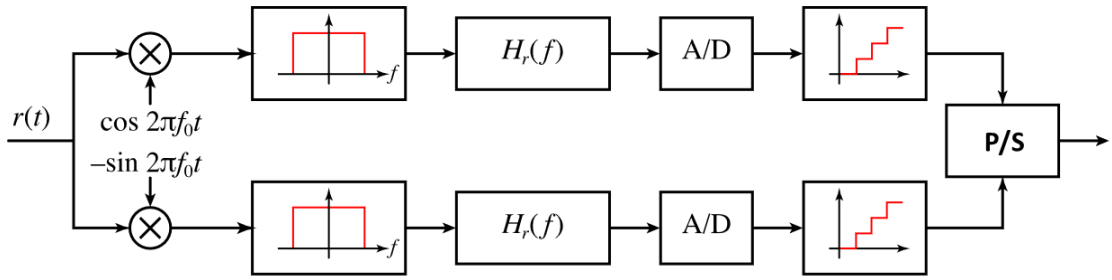
Các tín hiệu số $k_1$, $k_2$ sẽ được chuyển thành các tín hiệu tương tự bằng các bộ DAC (Digital-to-Analog Converter), sau đó sẽ được kết hợp với các sóng mang tương ứng $\cos(\omega_c t)$ , $-\sin(\omega_c t)$ thông qua các bộ nhân tín hiệu. Tín hiệu đầu ra của 2 bộ nhân sẽ được kết hợp lại để tạo thành tín hiệu đã được điều chế và được truyền đi. Sơ đồ tổng quát như sau:




*Hình 2: Sơ đồ điều chế 4-QAM*

**ii) Giải điều chế 4-QAM:**

Ở nơi thu, sau khi nhận được tín hiệu, ta có thể giải điều chế bằng cách kết hợp các bộ nhân, bộ lọc thông thấp, chuyển đổi tương tự số (ADC), bộ lượng tử hóa và bộ chuyển đổi song song – nối tiếp (P/S) theo sơ đồ sau:




*Hình 3: Sơ đồ giải điều chế 4-QAM ( Với $H_r(f)$ là đáp ứng tần số của máy thu)*

Thật vậy, ta có tín hiệu nhận được là:

$$r(t) = k_1 * \cos(2\pi f_c t) - k_2 * \sin(2\pi f_c t) \quad [5]$$

Xét nhánh trên trong trường hợp lý tưởng, sau khi qua bộ nhân, ta sẽ được tín hiệu g(t) như sau:

$$g(t) = r(t) * \cos(2\pi f_c t) = k_1 * \cos^2(2\pi f_c t) - k_2 * \sin(2\pi f_c t) * \cos(2\pi f_c t) \quad [6]$$

$$\rightarrow g(t) = \frac{k_1}{2}(1 + \cos(4\pi f_c t)) - \frac{k_2}{2}\sin(4\pi f_c t) \quad [7]$$

Bộ lọc thông thấp sau đó sẽ lọc đi các thành phần chưa tần số cao ( chứa $4\pi f_c t$) và chỉ để lại thành phần $k_1$. Trong trường hợp lý tưởng, ta sẽ sẽ thu được thành phần này một cách độc lập.

Tương tự, ta sẽ thu được $k_2$ theo quy trình ở nhánh dưới của sơ đồ bằng cách lọc thông thấp tín hiệu:

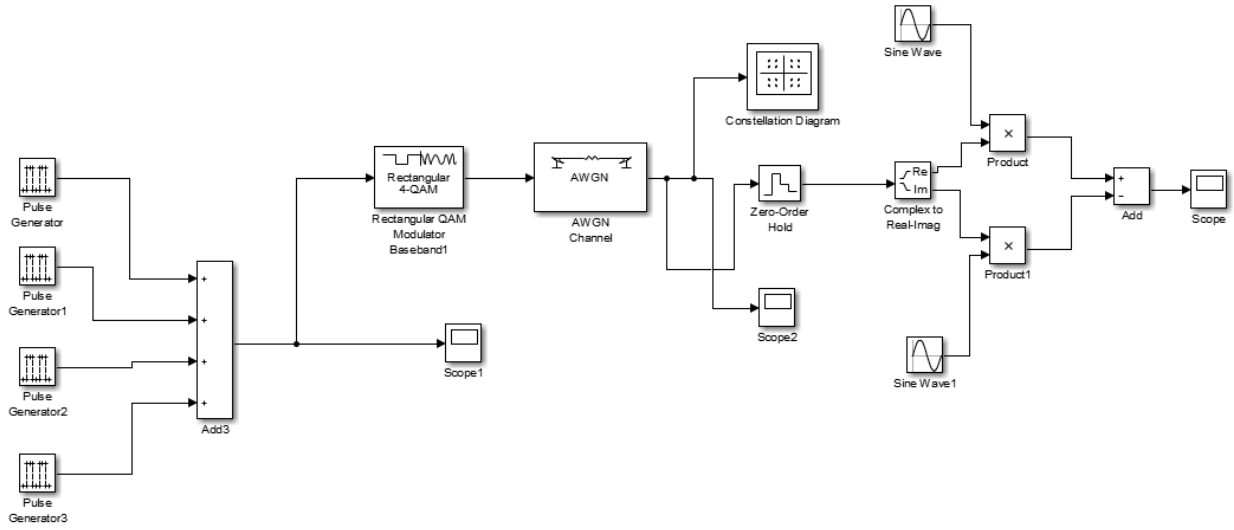
$$h(t) = \frac{k_1}{2}\cos(4\pi f_c t) - \frac{k_2}{2}(1 - \cos(4\pi f_c t)) \quad [8]$$

# II. Yêu cầu của dự án:

- Thực hiện điều chế 4-QAM trên kit eZDSP C5515 và trên MATLAB với tần số sóng mang $f_c$ = 6 kHz, tần số số $f_{dig}$ = 150 Hz.
- Máy tính phải sử dụng các phần mềm có tính năng tạo sóng để gửi tín hiệu đến kit thông qua một cáp kết nối 3.5 mm. Khi nhận được tín hiệu trên cổng STEREO-IN của kit, kit sẽ điều chế 4-QAM và gửi trả kết quả về máy tính thông qua cổng STEREO-OUT của kit. Máy tính hiện kết quả nhận được trên một phần mềm giả lập Oscilloscope.

# III. Mô phỏng điều chế 4-QAM trên MATLAB:

## 1) Sơ đồ Simulink:




*Hình 4: Sơ đồ khối SIMULINK của bộ giải điều chế 4-QAM*

Ở đầu vào của mô hình, nhóm đặt các khối Pulse Generator để tạo thông tin dưới dạng tín hiệu xung vuông. Nhờ các khối này, nhóm có thể hiệu chỉnh biên độ, chu kì, độ rộng xung, độ trễ pha, thời gian mẫu ở các khối để tạo tín hiệu mong muốn.

Kế tiếp, đằng sau khối tạo thông tin, nhóm sẽ đặt khối Rectangular QAM Modulator Baseband để chuyển các giá trị đầu vào thành các thành phần I,Q tương ứng .

Sau đó, nhóm đặt một khối AWGN Channel để mô phỏng ảnh hưởng của Additive White Gaussian Noise đến việc truyền tín hiệu. AWGN là một mô hình nhiễu cơ bản được sử dụng trong lý thuyết thông tin để bắt chước tác động của nhiều quá trình ngẫu nhiên xảy ra trong tự nhiên đến việc truyền nhận dữ liệu.

Và tiếp theo, nhóm đã đặt khối Constellation Diagram để vẽ giản đồ chòm sao và thực hiện đo đạc các thông số MER và EVM.

Khối Zero-Order Hold được đặt vào để chuyển đổi tín hiệu rời rạc đầu vào thành tín hiệu liên tục ở đầu ra.

Khối Complex to Real-Imag để tách tín hiệu phức thành 2 phần thực và phần ảo.

Hai khối Sine Wave có chức năng tạo sóng mang dạng Sin và Cos nhằm thực hiện điều chế

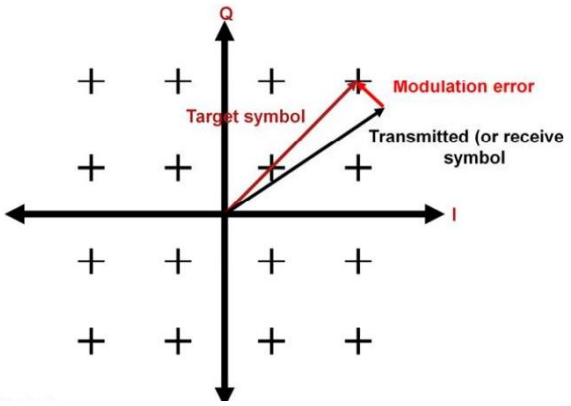
Và cuối cùng, nhóm đặt các khối cộng, khối nhân và khối vẽ đồ thị để thu được dạng sóng sau khi đã điều chế 4-QAM.

**2) Phương pháp đo MER và EVM:**

a) **Phương pháp đo Modulation Error Ratio - MER:**

The Modulation Error Ratio (MER) là thông số đánh giá chất lượng điều chế. Cụ thể với QAM nói chung, ta có công thức:

$$\overrightarrow{Modulation\ error} = \overrightarrow{Transmitted\ symbol} - \overrightarrow{Target\ symbol}$$




*Hình 5: Hình minh họa các vector sai số điều chế, vector truyền và vector mục tiêu.*

Và ta có MER tính theo dB:

$$MER = 10log\left(\frac{P_{signal}}{P_{error}}\right) \quad [9]$$

Trong đó, $P_{signal}$ là năng lượng hiệu dụng của tín hiệu chuẩn được truyền. $P_{error}$ là năng lượng hiệu dụng của vectơ lỗi.

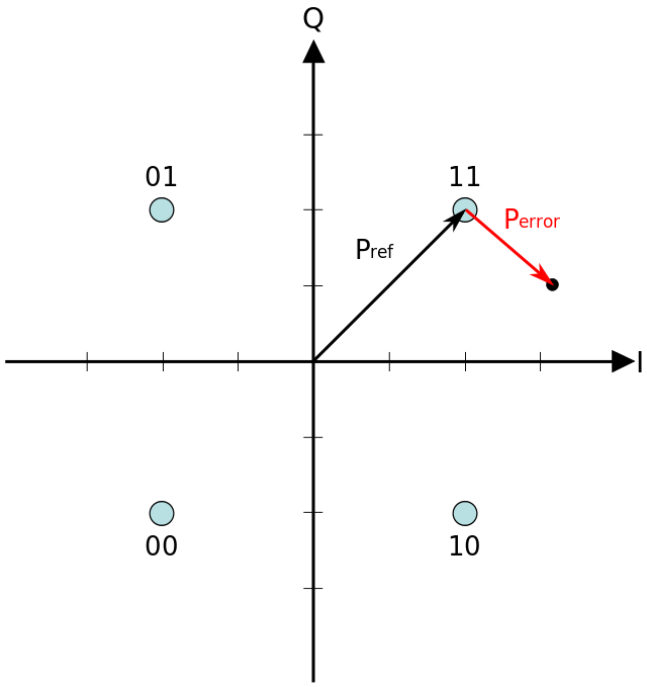
Do đó, MER càng lớn thì tỉ lệ lỗi trong quá trình điều chế càng cao.

b) **Phương pháp đo Error Vector Magnitude – EVM:**

The Error Cector Magnitude (EVM) là phương pháp đánh giá đánh giá chất lượng của quá trình phát thanh kỹ thuật số của bộ thu và bộ nhận. Một tín hiệu được gửi bởi bộ truyền lý tưởng đến bộ thu sẽ luôn có giản đồ chòm sao ở dạng lý tưởng, có nghĩa là tín hiệu phát ra ở máy phát sẽ được tái tạo một cách hoàn hảo ở máy thu. Tuy nhiên, trên thực tế, các tác động bên ngoài như rò rỉ tín hiệu sóng mang, nhiễu pha… đã gây ra sự dịch chuyển các điểm chòm sao thực tại lệch khỏi các điểm sao lý tưởng. EVM là một phương pháp đo độ sai lệch của những điểm ấy so với các điểm chòm sao lý tưởng. Hoặc ta có thể hiểu đơn giản là vector sai số được chuẩn hóa theo biên độ tín hiệu thông tin chính là EVM. Ta có công thức:

$$EVM(\%) = \sqrt{\frac{P_{error}}{P_{reference}}} x100\% \quad [10]$$

Với $P_{error}$ là biên độ hiệu dụng của vector sai số, $P_{reference}$ là biên độ hiệu dụng của tín hiệu cao nhất trong giản đồ chòm sao. Do đó EVM càng lớn thì tín hiệu sai lệch trong việc điều chế 4-QAM càng cao.

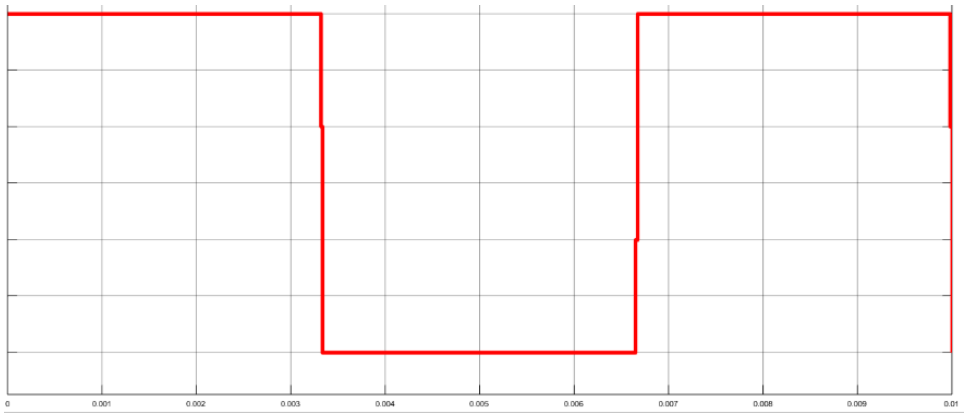



*Hình 6: Hình minh họa vector sai số và vector tham chiếu lý tưởng.*

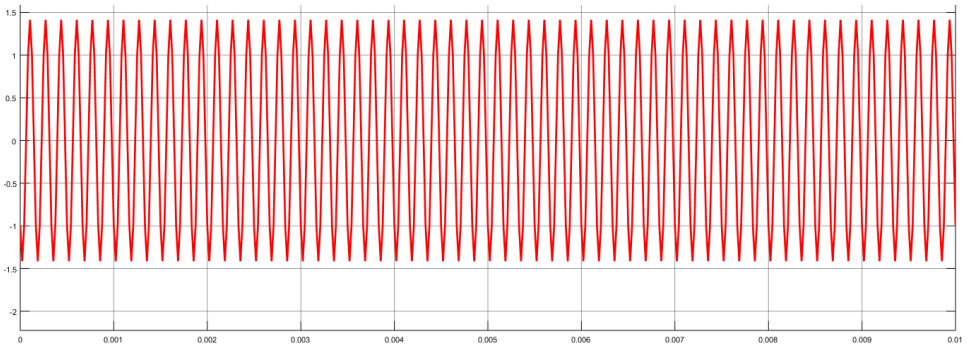
### 3) Kết quả điều chế 4-QAM:

**<u>Trường hợp 1</u>**: Tín hiệu đầu vào là sóng vuông với biên độ 1, có tần số $f_{dig}$ = 150 Hz, bên cạnh đó là sóng mang là tín hiệu hình sin với tần số $f_c$ = 6000 Hz. Nhóm lấy tần số lấy mẫu $f_s$ = 48000Hz.

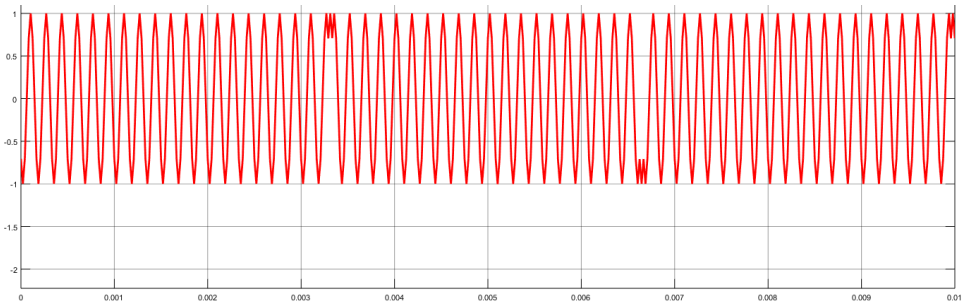
- Nếu ta điều chỉnh bộ AWGN channel với $\frac{E_b}{N_o} = 100dB$. Ta thu được kết quả:



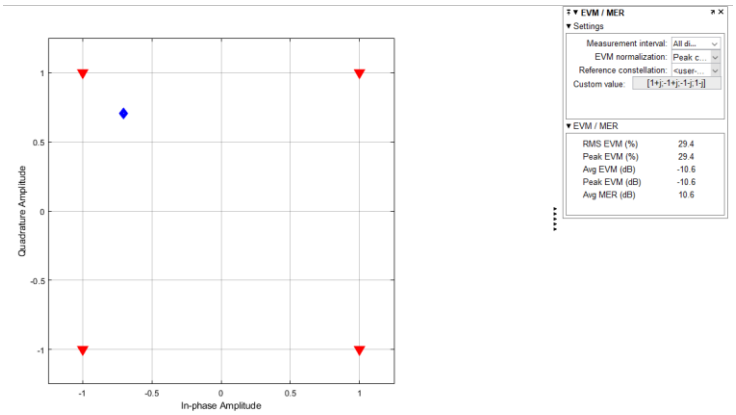
*Hình 7: : Hình tín hiệu thông tín dạng sóng vuông đầu vào với fdig = 150 Hz*



*Hình 8: Hình tín hiệu sóng mang fc = 6000 Hz*

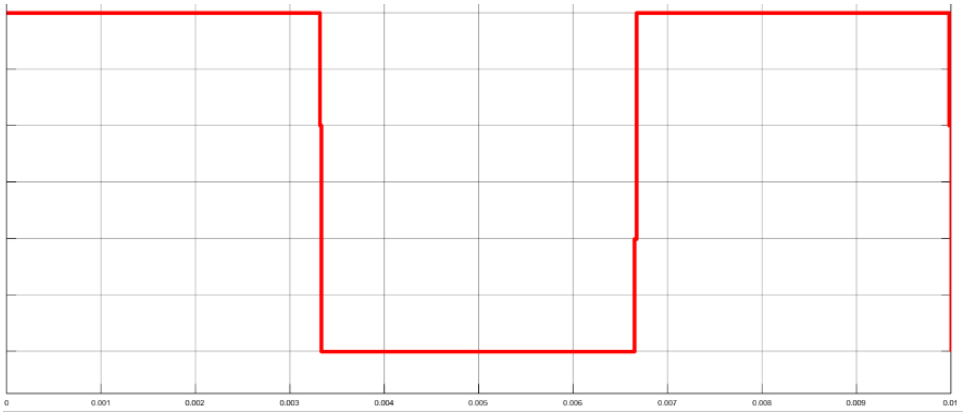


*Hình 9: Hình tín hiệu được điều chế 4-QAM*

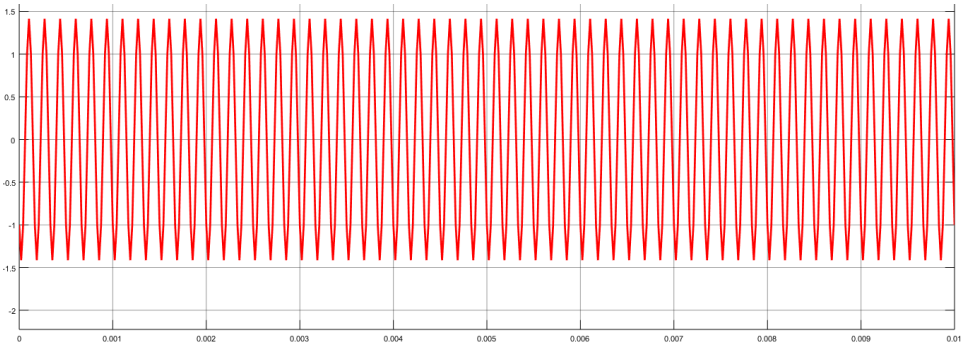



*Hình 10: Hình giản đồ chòm sao các điểm lý tưởng, điểm thực tế nhận được và các kết quả MER, EVM*

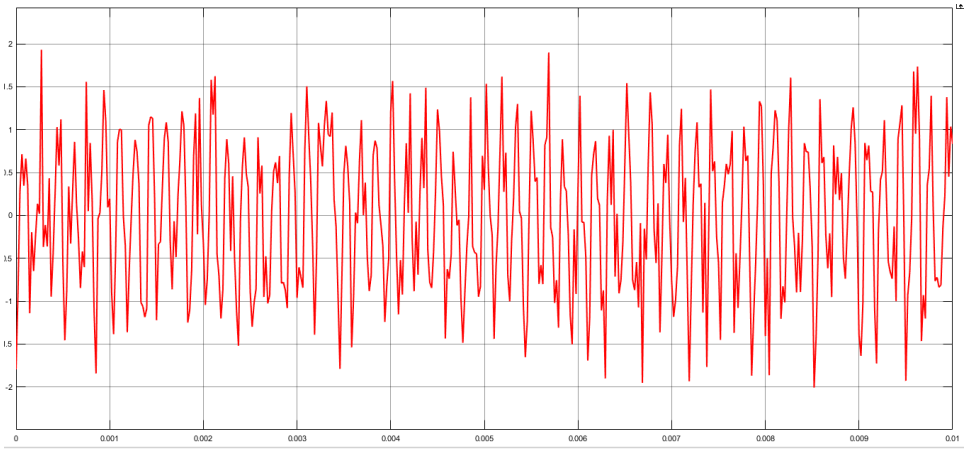
- Nếu ta điều chỉnh bộ AWGN channel với $\frac{E_b}{N_o} = 50dB$. Ta thu được kết quả:



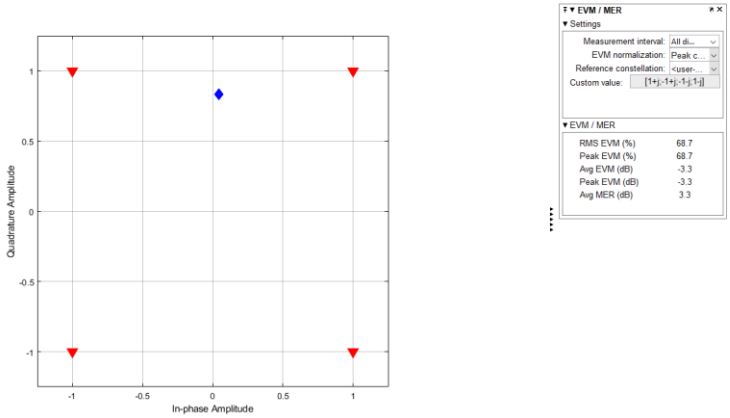
*Hình 11: Hình tín hiệu thông tin đầu vào dạng sóng vuông với fdig = 150 Hz*



*Hình 12: Hình tín hiệu sóng mang fc = 6000 Hz*



*Hình 13: Hình tín hiệu được điều chế 4-QAM*




*Hình 14: Hình giản đồ chòm sao các điểm lý tưởng, điểm thực tế nhận được và các kết quả MER, EVM*

**Nhận xét kết quả MER (dB) và EVM (%):**

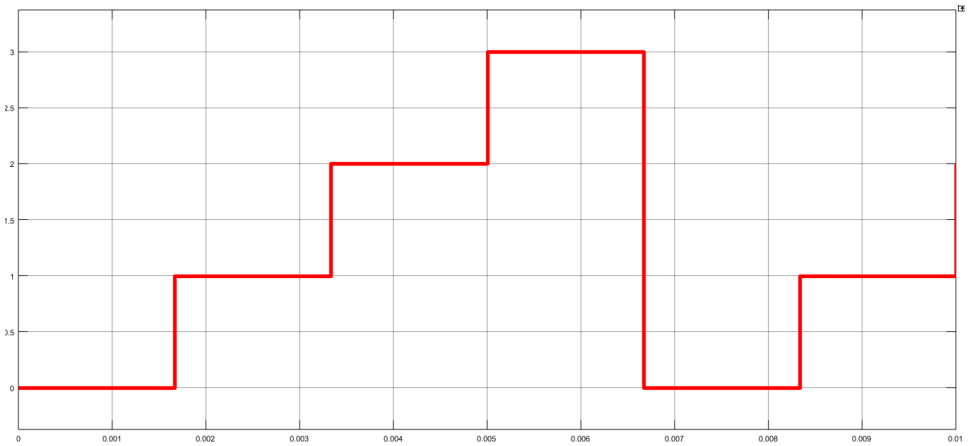
+ Với $\frac{E_b}{N_o} = 100\ dB$, ta có $EVM_{RMS}$ (%) = 29.4% và $MER_{AVG}$ = 10.6 dB

+ Với $\frac{E_b}{N_o} = 50\ dB$, ta có $EVM_{RMS}$ (%) = 68.7% và $MER_{AVG}$ = 3.3 dB

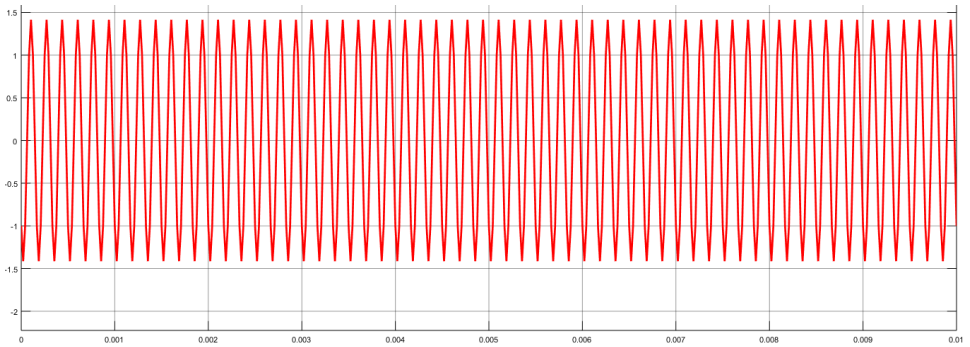
Kết quả trên là hợp lý, vì với tỷ lệ $\frac{E_b}{N_o}$ càng lớn thì tỉ lệ tương đương tỷ lệ SNR ( Signal to noise ratio ) càng lớn, việc nhiễu tín hiệu sẽ gây ít tác động hơn. Ta có công thức $MER(dB) = 10log\left(\frac{P_{signal}}{P_{error}}\right)$ và $EVM(\%) = \sqrt{\frac{P_{error}}{P_{reference}}}x100\%$. Nên với tỷ lệ $\frac{E_b}{N_o} = 100dB$ sẽ có MER(dB) lớn hơn và EVM(%) nhỏ hơn so với tỷ lệ $\frac{E_b}{N_o} = 50dB$ .

**<u>Trường hợp 2:</u>** Tín hiệu đầu vào là sóng vuông có các mức 0 1 2 3, có tần số $f_{dig}$ = 150 Hz, bên cạnh đó là sóng mang là tín hiệu hình sin với tần số $f_c$ = 6000 Hz. Nhóm lấy tần số lấy mẫu $f_s$ = 600Hz.

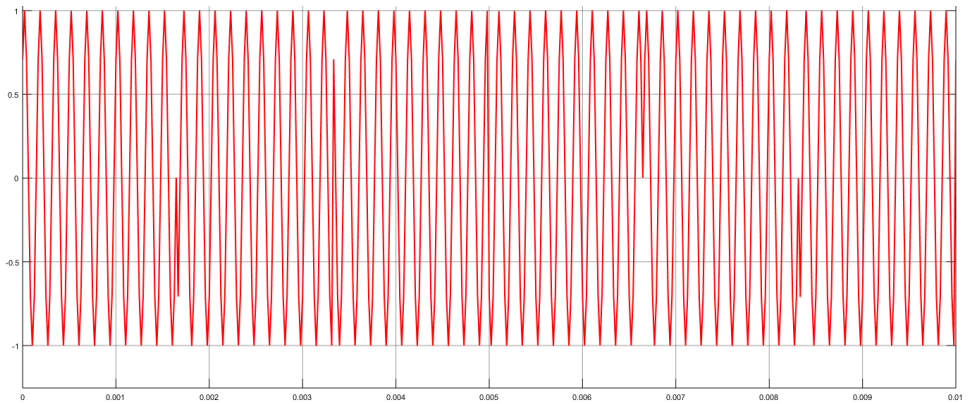
- Nếu ta điều chỉnh bộ AWGN channel với $\frac{E_b}{N_o} = 100dB$. Ta thu được kết quả:



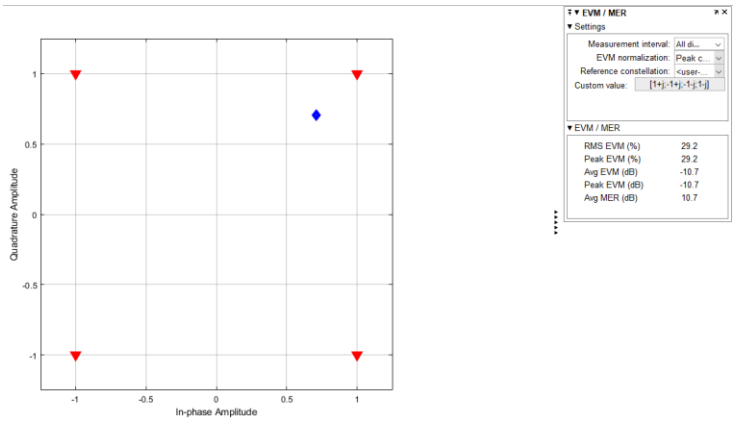
*Hình 15: Tín hiệu đầu vào xung vuông có các mức 0 1 2 3 với fdig = 150 Hz*



*Hình 16: Hính tín hiệu sóng mang với fc = 6000 Hz*

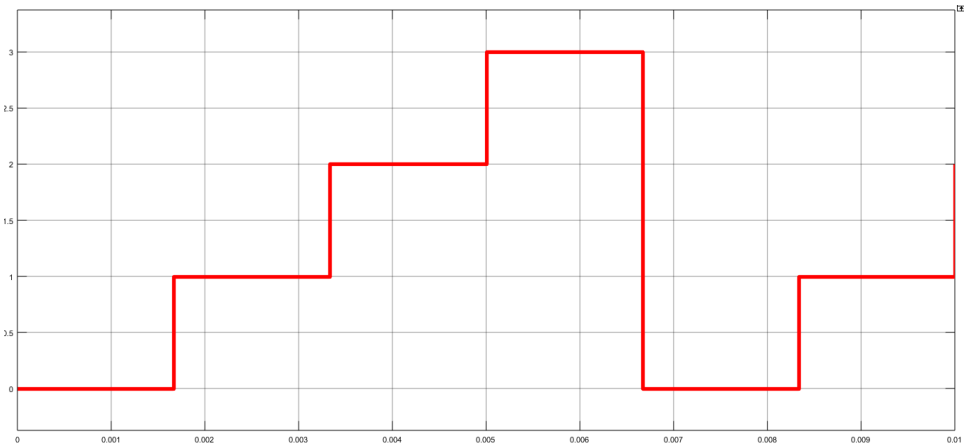


*Hình 17: Hình tín hiệu sóng được điều chế 4-QAM*

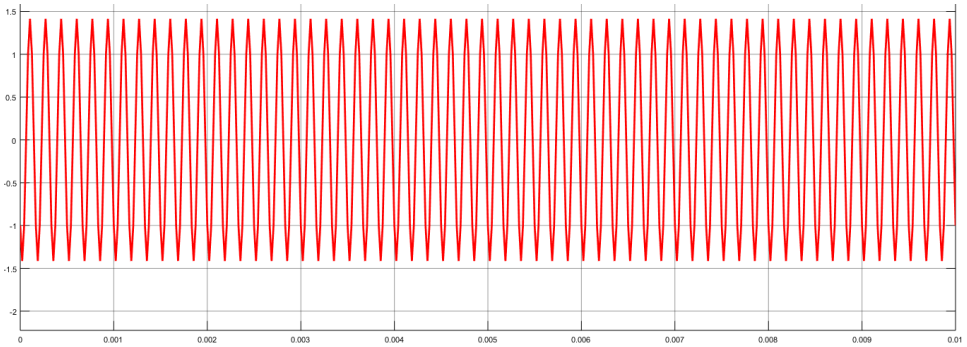



*Hình 18: Hình giản đồ chòm sao các điểm lý tưởng, điểm thực tế nhận được và các kết quả MER, EVM*

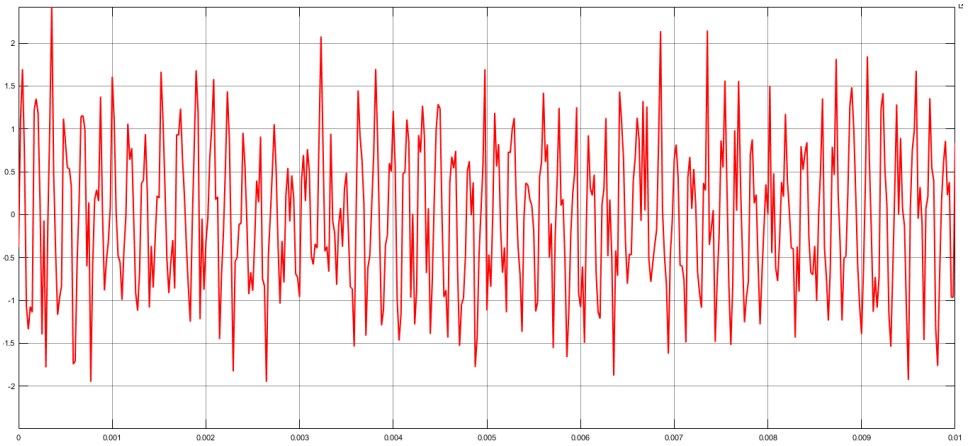
- Nếu ta điều chỉnh bộ AWGN channel với $\frac{E_b}{N_o} = 50dB$. Ta thu được kết quả:



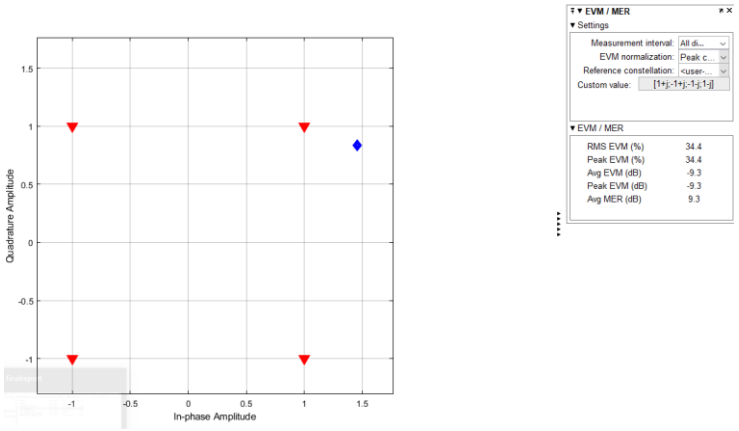
*Hình 19: Hình tín hiệu đầu vào xung vuông với fdig = 150 Hz*



*Hình 20: Hình tín hiệu sóng mang fc = 6000 Hz*



*Hình 21: Hình tín hiệu được điều chế 4-QAM*




*Hình 22: Hình giản đồ chòm sao các điểm lý tưởng, điểm thực tế nhận được và các kết quả MER, EVM*

**Nhận xét kết quả MER(dB) và EVM (%)**

+ Với $\frac{E_b}{N_o} = 100\ dB$, ta có $EVM_{RMS}$ (%) = 29.2% và $MER_{AVG}$ = 10.7 dB

+ Với $\frac{E_b}{N_o} = 50\ dB$, ta có $EVM_{RMS}$ (%) = 34.4% và $MER_{AVG}$ = 9.3 dB

Kết quả trên là hợp lý, vì với tỷ lệ $\frac{E_b}{N_o}$ càng lớn thì tỉ lệ tương đương tỷ lệ SNR ( Signal to noise ratio ) càng lớn, việc nhiễu tín hiệu sẽ gây ít tác động hơn.

Ta có công thức $MER(dB) = 10log\left(\frac{P_{signal}}{P_{error}}\right)$ và $EVM(\%) = \sqrt{\frac{P_{error}}{P_{reference}}} x100\%$. Nên với tỷ lệ $\frac{E_b}{N_o} = 100dB$ sẽ có MER(dB) lớn hơn và EVM(%) nhỏ hơn so với tỷ lệ $\frac{E_b}{N_o} = 50dB$ .

**Nhận xét, kết luận từ 2 trường hợp trên:**

Ta quan sát thấy được khi $\frac{E_b}{N_o} = 100dB$ thì tín hiệu đầu ra được điều chế 4-QAM đã được điều chế dựa trên việc thay đổi pha của sóng mang. Ví dụ ở mức trạng thái bit 3 xuống bit 0 của tín hiệu thông tin, ta có thể quan sát được sự đảo pha một cách rõ ràng của tín hiệu điều chế với tín hiệu sóng mang. Việc điều chế bằng Matlab đạt được kết quả mong đợi.

Khi $\frac{E_b}{N_o} = 50dB$ thì tín hiệu đầu ra đã bị tác động bởi nhiễu tín hiệu khá là nhiều. Tín hiệu điều chế 4-QAM không còn giữ được hình dạng biên độ của sóng mang ban đầu. Phương pháp điều chế 4-QAM hoạt động hiệu quả hơn nếu $\frac{E_b}{N_o} > 50dB$.

# IV. Giới thiệu về kit TMS 320C5515 eZdsp™ USB Stick và khối mã hóa/ giải mã âm thanh:

## 1) Giới thiệu kit TMS 320C5515 eZdsp™ USB Stick :

### a) Tổng quan :

TMS320C5515 eZdsp™ USB Stick là 1 bộ công cụ phát triển đến từ hãng Texas Instruments (TI) để hỗ trợ người dùng phát triển các ứng dụng dựa trên con chip DSP 16-bit TMS320C5515 của TI.

TMS320C5515 là bộ xử lý tín hiệu số 16 bit sử dụng năng lượng thấp nhất trong công nghiệp, do đó giúp tiết kiệm năng lượng và kéo dài thời gian sử dụng pin. Với tốc độ xử lí 240 MIPS (Millions of Instructions Per Second), 320KB bộ nhớ, một bộ tăng tốc phần cứng để tính toán FFT, C5515 là một bộ xử lý thích hợp cho các ứng dụng DSP như mấy ghi âm, nhạc cụ điện tử và các hàng điện tử tiêu dùng khác.

### b) Tính năng của TMS320C5515 eZdsp™ :

- **Của kit TMS320C5515 eZdsp:**

- Kích thước nhỏ gọn, công suất thấp, hiệu suất cao.
- Bộ xử lí tín hiệu số TMS320C5515 của Texas Instrument.
- Bộ giải mã stereo ( stereo in, stereo out) TLV320AIC3204 của Texas Instrument.
- Đầu đọc thẻ nhớ Micro SD.
- Hỗ trợ USB 2.0 để trao đổi dữ liệu và cấp nguồn cho Kit hoạt động.
- Bộ nhớ NOR Flash 32 Mb.

- Màn hình OLED gắn sẵn dùng giao tiếp $I^2C$.
- 5 đèn LED và 2 nút nhấn đều có thể lập trình được.
- Có nhiều khe cắm mở rộng.
- Tương thích với phần mềm Code Composer Studio của Texas Instrument

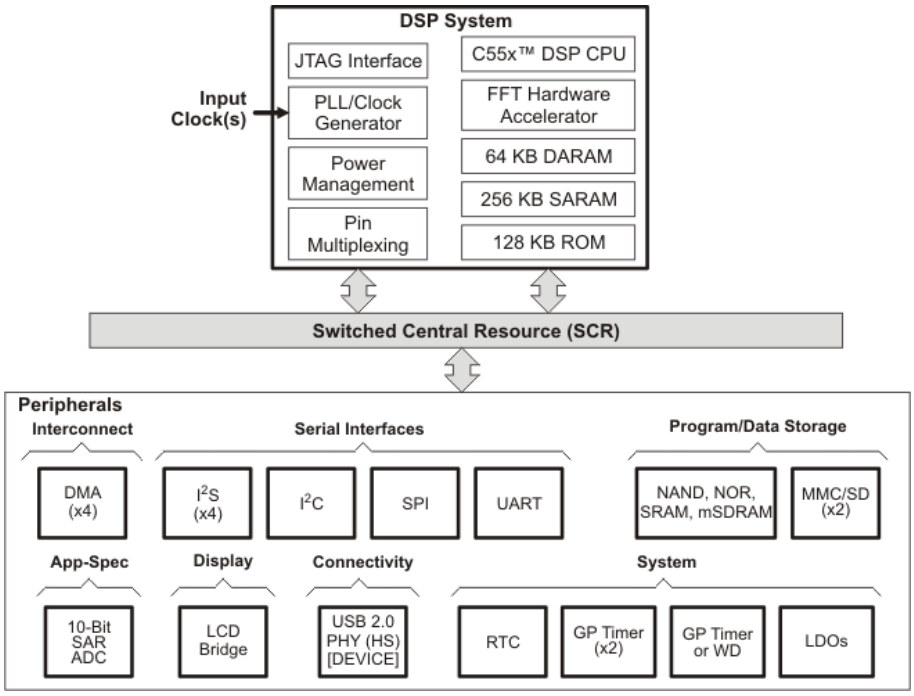
- **Của bộ xử lý DSP TMS320C5515:**

Toàn bộ các tính năng của bộ xử lí được thể hiện qua 2 bảng dưới đây:

| HARDWARE FEATURES | | | C5515 |
|---|---|---|---|
| Peripherals<br>Not all peripheral pins are available at the same time (for more detail, see the Device Configuration section). | External Memory Interface (EMIF) | | Asynchronous (8/16-bit bus width) SRAM, Flash (NOR, NAND), SDRAM and Mobile SDRAM (16-bit bus width)(1) |
| | DMA | | Four DMA controllers each with four channels, for a total of 16 channels |
| | Timers | | 2 32-Bit General-Purpose (GP) Timers<br>1 Additional Timer Configurable as a 32-Bit GP Timer and/or a Watchdog |
| | UART | | 1 (with RTS/CTS flow control) |
| | SPI | | 1 with 4 chip selects |
| | $I^2C$ | | 1 (Master/Slave) |
| | $I^2S$ | | 4 (Two Channel, Full Duplex Communication) |
| | USB 2.0 (Device *only*) | | High- and Full-Speed Device |
| | MMC/SD | | 2 MMC/SD, 256 byte read/write buffer, max 50-MHz clock for SD cards, and signaling for DMA transfers |
| | LCD Bridge | | 1 (8-bit or 16-bit asynchronous parallel bus) |
| | ADC (Successive Approximation [SAR]) | | 1 (10-bit, 4-input, 16-µs conversion time) |
| | Real-Time Clock (RTC) | | 1 (Crystal Input, Separate Clock Domain and Power Supply) |
| | FFT Hardware Accelerator | | 1 (Supports 8 to 1024-point 16-bit real and complex FFT) |
| | General-Purpose Input/Output Port (GPIO) | | Up to 26 pins (with 1 Additional General-Purpose Output (XF) and 4 Special-Purpose Outputs for Use With SAR) |
| On-Chip Memory | Size (Bytes) | | 320KB RAM, 128KB ROM |
| | Organization | | • 64KB On-Chip Dual-Access RAM (DARAM)<br>• 256KB On-Chip Single-Access RAM (SARAM)<br>• 128KB On-Chip Single-Access ROM (SAROM) |
| JTAG BSDL_ID | JTAGID Register (Value is: 0x1B8F E02F) | | see Figure 5-45 |
| CPU Frequency | MHz | 1.05-V Core | 60 or 75 MHz |
| | | 1.3-V Core | 100 or 120 MHz |
| Cycle Time | ns | 1.05-V Core | 16.67, 13.$\overline{3}$ ns |
| | | 1.3-V Core | 10, 8.33 ns |
| Voltage | Core (V) | | 1.05 V (60, 75 MHz) |
| | | | 1.3 V (100, 120 MHz) |
| | I/O (V) | | 1.8 V, 2.5 V, 2.75 V, 3.3 V |
| LDOs | DSP_LDO | | 1.3 V or 1.05 V, 250 mA max current for DSP CPU ($CV_{DD}$) |
| | ANA_LDO | | 1.3 V, 4 mA max current for PLL ($V_{DDA\_PLL}$), SAR, and power management circuits ($V_{DDA\_ANA}$) |
| | USB_LDO | | 1.3 V, 25 mA max current for USB core digital (USB_$V_{DD1P3}$) and PHY circuits (USB_$V_{DDA1P3}$) |
| PLL Options | Software Programmable Multiplier | | x4 to x4099 multiplier |
| BGA Package | 10 x 10 mm | | 196-Pin BGA (ZCH) |

*Bảng 2: Các tính năng của C5515*

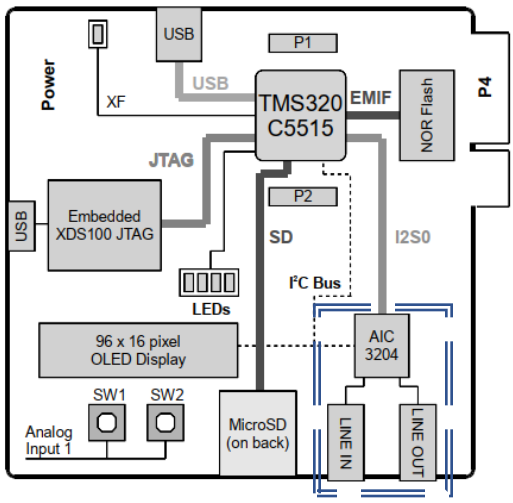
c) **Sơ đồ khối chức năng:**




*Hình 23: Các khối chức năng của C5515*

**2) Khối mã hóa/ giải mã âm thanh dùng chip TLV320AIC3204:**

Vị trí của khối mã hóa/ giải mã âm thanh dùng chip TLV320AIC3204 nằm ở góc dưới bên phải như hình sau:




*Hình 24: Vị trí của khối mã hóa/ giải mã âm thanh trên kit*

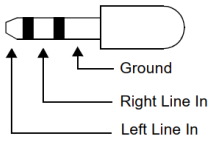
a) **J3, Audio In Connector:**

Cổng vào Audio, J3, được dùng để truyền tín hiệu vào bộ giải mã AIC3204. Các tín hiệu trên các chân như bảng sau:

| Pin # | Signal Name | AIC3204 Pin # |
|---|---|---|
| 1 | GND-AIC | |
| 2 | AIC_LINE2L | 15 |
| 3 | AIC_LINE2R | 16 |
| 4 | No connect | |
| 5 | No connect | |

*Bảng 3: J3, cổng vào Audio*

Dưới đây là cấu trúc của cổng kết 3.5 mm TRS.




*Hình 25: Cổng kết nối 3.5 mm dạng TRS*

b) **J4, Audio Out Connector:**

Cổng ra Audio, J4, được dùng để đưa tín hiệu từ AIC3204 ra bên ngoài. Các tên tín hiệu như bảng sau:

| Pin # | Signal Name | AIC3204 Pin # |
|---|---|---|
| 1 | GND-AIC | |
| 2 | HEADPHONE_LOUT | 25 |
| 3 | HEADPHONE_ROUT | 27 |
| 4 | No connect | |
| 5 | No connect | |

*Bảng 4: J4, cổng ra Audio*

c) **Chip TLV320AIC3204:**

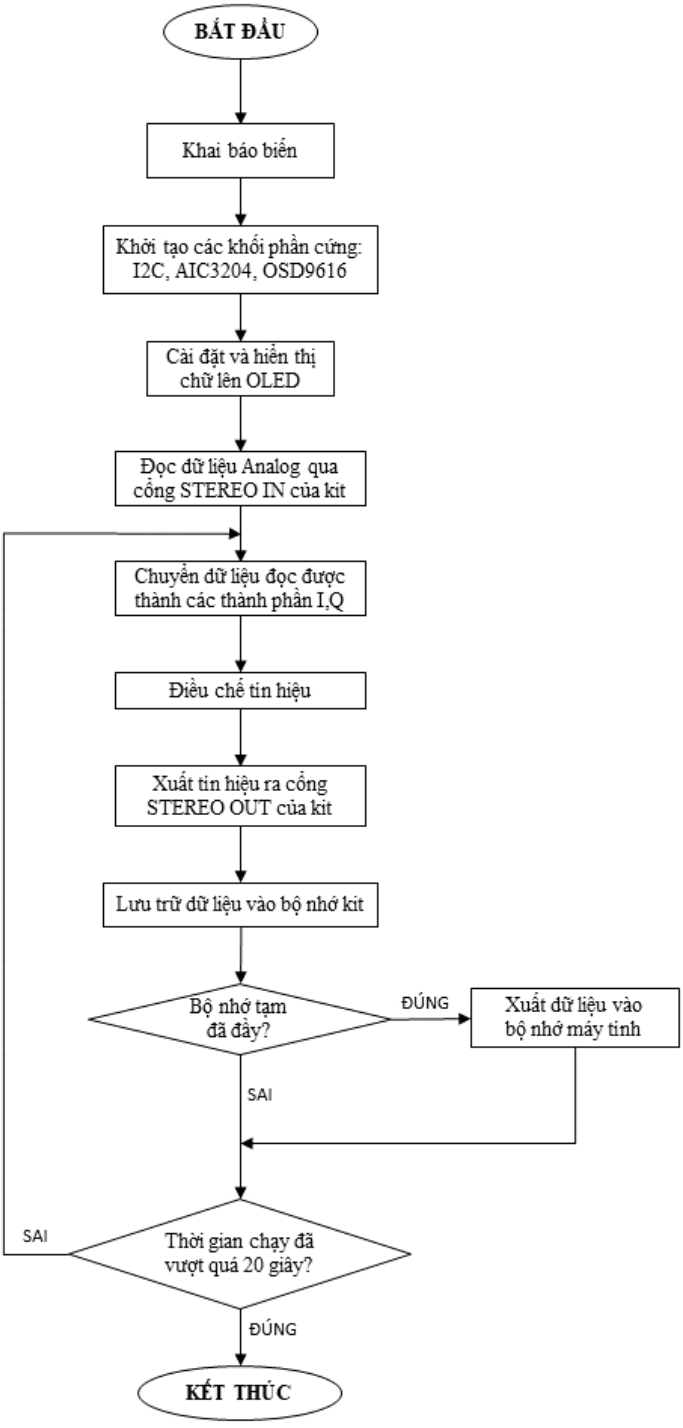
Chip TLV320AIC3204 từ hãng TI là một bộ mã hóa/ giải mã âm thanh với điện áp, công suất thấp và có khả năng lập trình input và output, có tích hợp các vòng khóa pha (Phase-Locked Loop) và các ổn áp LDOs (Low-DropOut Regulators). Chip này được ứng dụng nhiều trong các thiết bị di động và thiết bị nghe nhạc cầm tay.

Các tính năng của nó như sau:

- 2 Bộ chuyển đổi âm thanh số-tương tự với 100dB SNR.
- 2 Bộ chuyển đổi âm thanh tương tự-số với 93dB SNR.
- 6 ngõ vào và 4 ngõ ra tương tự.
- Sử dụng các giao tiếp âm thanh số như: L, R, I2S, TDM, DSP
- Tần số lấy mẫu tối đa: 192 kHz
- Kích thước 5 mm x 5 mm với 32 chân.

# V. Mô phỏng điều chế 4-QAM trên kit eZDSP:

Dưới đây là sơ đồ khối mô tả quá trình điều chế 4-QAM trên kit eZDSP.




*Hình 26: Sơ đồ khối mo tả thuật toán chạy trên kit*

1) **Giải thích sơ đồ khối:**

a) **Khai báo biến:**

**main.c**

```
// Các biến liên quan đến vòng lặp:
      Int8 j = 0;
      Int16 i = 0;
      Int16 id = 0;

// Các biến liên quan đến việc lấy mẫu và điều chế:
      Int8 sample;
      Int16 mes1, mes2;
      Int8  I, Q;
      Int16 modulated_signal;

// Các biến liên quan đến việc lưu dữ liệu vào máy tính:
       static Int16 message_bin[3200];
       *pmes = message_bin;
       static Int16 carrier_bin[1600];
        *pcar = carrier_bin;
       static Int16 modulated_sig_bin[1600];
       *pmod = modulated_sig_bin;
       Int8 logFlag = 0;
```

b) **Khởi tạo các khối phần cứng I2C, AIC3204, OSD9616:**

Dưới đây là các hàm con dùng để khởi tạo I2C, AIC3204, OSD9616 trên kit eZDSP. Chi tiết từng code được trình bày trong Phụ lục B

**main.c (tiếp theo)**

```
// Khởi tạo các chức năng chung của USBSTK5515
     USBSTK5515_init( );

// Khởi tạo chức năng I2C của kit
      SYS_EXBUSSEL = 0x6100;
      USBSTK5515_I2C_init( );

// Khởi tạo chip âm thanh AIC3204
      AIC3204_init();

// Khởi tạo màn hình OLED OSD9616:
      OSD9616_init();
```

c) **Cài đặt và hiển thị chữ lên OLED:**

Bằng cách sử dụng hàm con dưới đây, nhóm đã xuất dòng giới thiệu bằng tiếng Việt ra màn hình OLED của kit. Dòng chữ đó là:

**"VP15-HK1-4QAM**

**Nhóm 7 - Thiện - Minh - Hoàng"**

Chi tiết hàm con này được được trình bày trong phụ lục B, phần oled.h và oled.c

**main.c (tiếp theo)**

```
    // Hiển thị dòng chữ giới thiệu:
        OSD9616_displayText();
```

d) **Đọc dữ liệu Analog qua ngõ vào STEREO IN của kit:**

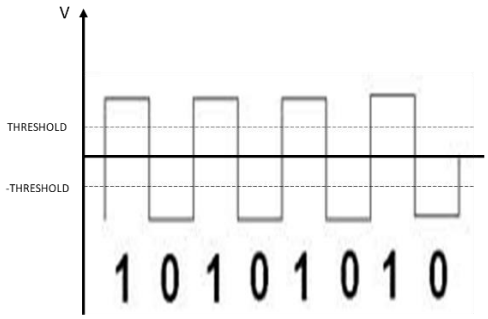
Đề tài của nhóm là điều chế 4-QAM, do đó trong đoạn code sau, nhóm chúng tôi đã lập trình để lấy mẫu 2 lần liên tiếp bằng bộ ADC 16 bit tích hợp sẵn tại tần số lấy mẫu là $f_s$ = 48 kHz. Dữ liệu đọc được có dạng số nguyên có dấu 16 bit.

**main.c (tiếp theo):**

```
    // Đọc dữ liệu Audio dưới dạng Analog
    while((Rcv & I2S0_IR) == 0);  // Chờ cho đến khi kit
sẵn sàng để đọc dữ liệu mới.
    mes1 = I2S0_W0_MSW_R;  // Đọc dữ liệu lần 1 trên kênh
trái của kết nối.

    while((Rcv & I2S0_IR) == 0);  // Chờ cho đến khi kit
sẵn sàng để đọc dữ liệu mới.
    mes2 = I2S0_W0_MSW_R;  // Đọc dữ liệu lần 2 trên kênh
trái của kết nối.
```

e) **Điều chế tín hiệu:**

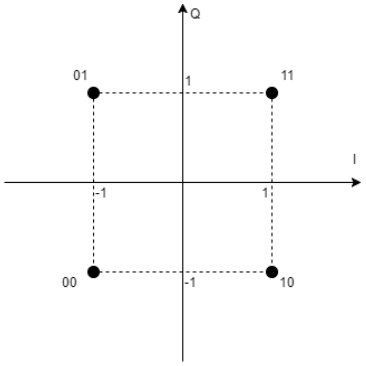
Bằng việc so sánh 2 giá trị vừa đọc được với các mức ngưỡng, kit có thể phân biệt được giá trị điện áp nào đại diện cho bit 1, giá trị nào đại diện cho bit 0. Sau đây là sơ đồ mô tả các giá trị xung vuông ngõ vào và các giá trị bit tương ứng.




*Hình 27: Mô tả chuỗi xung vuông ngõ vào cùng với các giá trị bit mà chúng đại diện*

Sau khi đã nhận diện được giá trị bit, kit sẽ thực hiện gộp 2 bit liền kề lại và chuyển thành các thành phần In-phase và Quadrature ( I và Q) trong điều chế 4-QAM.

Trong trường hợp điện áp nằm giữa -THRESHOLD và THRESHOLD, kit sẽ xem như là không có tín hiệu cần gửi và đặt các giá trị I, Q bằng 0. Biểu đồ chòm sao dưới đây thể hiện mối liên hệ giữa các dibit và các thành phần I, Q tương ứng của chúng.




*Hình 28: Biểu đồ chòm sao thể hiện các giá trị dibit và các thành phần I, Q tương ứng*

**main.c (tiếp theo)**

```
// Chuyển đổi thông tin nhận được thành các thành phần I,Q:
     if ((mes1 <= -THRESHOLD) && (mes2 <= -THRESHOLD)){
     I = -1; Q = -1;
     } else if ((mes1 <= -THRESHOLD) && (mes2 > THRESHOLD)){
     I = -1; Q =  1;
     } else if ((mes1 > THRESHOLD) && (mes2 <= -THRESHOLD)){
     I =  1; Q = -1;
     } else if ((mes1 > THRESHOLD) && (mes2 > THRESHOLD)) {
     I =  1; Q =  1;
     } else{
     I =  0; Q =  0;
     }
```

f) **Điều chế tín hiệu:**

Để có thể áp dụng công thức điều chế [1], kit cần các sóng mang sine và cosine ở tần số sóng mang $f_{carrier}$ = 6 kHz. Có nhiều phương pháp để tạo ra dữ liệu sine, cosine này như: dùng hàm toán học trong thư viện math.h có sẵn, dùng phương trình sai phân hoặc dùng bảng tra. Với mục tiêu là kit có thể tính toán, xử lí trong 1 khoảng thời gian bé hơn $1/f_c = 1/48000 \approx 21$ ms, nhóm chúng tôi đã chọn cách dùng bảng tra các giá trị sine và cosine vì sự đơn giản và tính toán nhanh chóng của phương pháp. Các bảng tra này được lưu trữ trong file sinetable.h như sau:

**sinetable.h**

```
/*16-bit signed samples,4 samples/period,f_carrier =6 kHz*/
    static Int8 samplesPerPeriod = 4;
    static Int16 sinetable[4]   = {
                  0,       16384,
                  0,    -16384 };
    static Int16 cosinetable[4] = {
              16384,          0,
             -16384,          0 };
```

Để có thể xác định số điểm cần thiết cho bảng tra của sóng mang, chúng tôi tính toán như sau. Tần số lấy mẫu là $f_c$=48kHz thì khoảng cách giữa 2 mẫu là 1/48000 (s). Nhóm lấy mẫu 2 lần liên tiếp rồi mới lấy giá trị sine, cosine để tính toán tín hiệu đã điều chế. Do đó, khoảng cách cần thiết giữa 2 điểm sine hoặc cosine liền kề là 2/48000 = 1/24000 (s). Chu kì sóng mang là 1/6000 (s). Do đó, số mẫu cần thiết của sóng mang trong 1 chu kì là (1/6000) / (1/24000) = 4 điểm.

Việc lựa chọn biên độ của sóng mang cũng được lựa chọn thận trọng để tránh các hiện tượng quá điện áp và gây nguy hiểm cho phần cứng của kit cũng như của máy tính. Do đó, nhóm đã chọn giá trị là 16384, bằng một nửa giá trị cực đại mà các bộ DAC 16 bit có thể chuyển đổi.

Dưới đây là đoạn code dùng để điều chế 4-QAM trên kit:

**main.c (tiếp theo)**

```
// Điều chế tín hiệu
// I*cos(2*pi*fc*t) - Q*sin(2*pi*fc*t)
modulated_signal=I*cosinetable[sample]-Q*sinetable[sample];
```

g) **Xuất tín hiệu ra cổng STEREO OUT trên kit:**

Việc xuất tín hiệu được thực hiện bằng việc ghi 1 giá trị Int16 đến thanh ghi I2S0_W0_MSW_W. Sau khi thanh ghi này được ghi, AIC3204 sẽ đọc giá trị trong thanh ghi này và mã hóa giá trị Digital thành tín hiệu Analog thông qua bộ DAC 16 bit của nó.

Việc xuất tín hiệu này chỉ được thực hiện trên 1 kênh trong 2 kênh của ngõ ra (kênh trái – Left Channel). Kênh còn lại được gán giá trị 0. Nhóm chọn cách này bởi vì ngõ vào microphone trên máy tính là ngõ Mono (1 kênh), trong khi ngõ ra Stereo Out của kit là Stereo (2 kênh). Do vậy, nếu truyền tín hiệu trên cả 2 kênh thì có thể xảy ra hiện tượng nhiễu xuyên âm (Cross-talk) và dồn kênh, ảnh hưởng đến chất lượng dữ liệu đọc được trên máy tính.

Dưới đây là đoạn code thực hiện việc xuất tín hiệu ra cổng STEREO OUT:

**main.c (tiếp theo)**

```
// Chờ cho đến khi kit sẵn sàng để gửi tín hiệu
     while((Xmit & I2S0_IR) == 0);

// Gửi tín hiệu đã điều chế vào kênh trái của bộ DAC 16 bit
     I2S0_W0_MSW_W = modulated_signal;
     I2S0_W0_LSW_W = 0;

// Tắt việc gửi tín hiệu trên kênh phải
     I2S0_W1_MSW_W = 0;
     I2S0_W1_LSW_W = 0;
```

h) **Thu thập và lưu trữ giữ liệu:**

Nhóm nhận thấy rằng thời gian sử dụng kit có hạn và có nhiều việc để làm với kit trong thời gian giới hạn đó, nhóm đã nảy ra ý tưởng là lưu lại dưới dạng tập tin nhị phân .bin dữ liệu của tín hiệu ADC đọc được của kit, tín hiệu sóng mang tương ứng với từng điểm của ADC và tín hiệu đã điều chế. Khi đã có các dữ liệu này rồi, nhóm sẽ dùng MATLAB để đọc và thực hiện việc phân tích, giải điều chế tín hiệu sau khi đã trả kit về cho giảng viên.

Còn việc lưu trữ dữ liệu dưới dạng tập tin nhị phân .bin được thực hiện trên kit. Các tập tin này được kit thu thập vào những thời điểm cuối của thời gian chạy chương trình ( khoảng giây thứ 19 tính từ lúc bắt đầu chạy). Dữ liệu về tín hiệu ADC đọc được có độ dài 3200 mẫu, mỗi mẫu là một số nguyên có dấu 16 bit. Còn dữ liệu về sóng mang và sóng đã điều chế có độ dài 1600 mẫu và mỗi mẫu là một số nguyên 16 bit.

Dưới đây là đoạn code thực hiện việc ghi nhận và lưu trữ dữ liệu dạng .bin:

**main.c (tiếp theo)**

```
// Thu thập dữ liệu:
     if ((sample ==0) && (j == 8000) && (i == 9))
     logFlag = 1;
     if (logFlag == 1){
          message_bin[2*id] = mes1;
          message_bin[2*id+1] = mes2;
          carrier_bin[id] = sinetable[sample];
          modulated_sig_bin[id] = modulated_signal;
          id = id + 1;

// Lưu lại thông điệp, sóng mang và sóng đã điều chế:
          if (id == 1600){
               logFlag = 0;
               ExportFile("..\\output\\message.bin",
3200,(Int16*) pmes );
               ExportFile("..\\output\\carrier.bin",
1600,(Int16*) pcar );
               ExportFile("..\\output\\modulated.bin",
1600,(Int16*) pmod );
          }
     }
```

## 2) Kết quả thực hiện:

### a) Kết nối giữa máy tính và kit:

Để có thể kết nối giữa máy tính và kit, nhóm dùng 2 loại kết nối sau:

1) Chuẩn kết nối tuần tự đa dụng (Universal Serial Bus – USB): kết nối này được dùng để nạp code vào kit từ máy tính và lưu dữ liệu .bin vào máy tính. Nhóm đã sử dụng dây cắm USB đi kèm kit để thực hiện kết nối này.

2) Chuẩn truyền dữ liệu âm thanh dạng Analog qua USB (Generic USB audio): kết nối này được sử dụng để truyền dữ liệu từ cổng Microphone đến cổng STEREO IN trên kit và nhận dữ liệu phát ra từ cổng STEREO OUT của kit trên máy tính. Nhóm đã sử dụng một USB soundcard làm trung gian kết nối giữa kit và máy tính.

Ngoài ra, khi chạy chương trình trên Code Composer Studio (CCS), màn hình OLED cũng sẽ hiện 2 dòng chữ chạy giới thiệu nhóm và tên đề tài.

Dưới đây là hình ảnh mô tả quá trình kết nối máy tính với kit eZDSP:

*Hình 29: Tổng quan về các kết nối của hệ thống.*

*Hình 30: Cáp USB đi kèm kit được sử dụng để kết nối qua cổng USB. Còn kết nối Audio được thực hiện với 2 sợi cáp Audio 3.5 mm chuẩn TRS.*


*Hình 31: Một USB âm thanh (USB Soundcard) được sử dụng làm trung gian kết nối giữa máy tính và kit.*

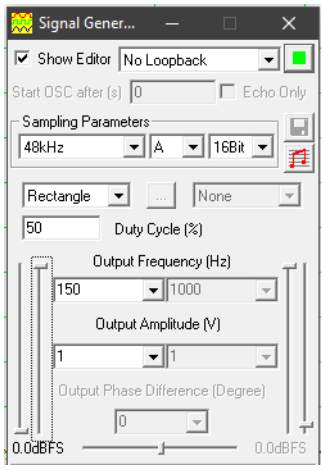
*Hình 32: Dòng chữ giới thiệu nhóm và đề tài trên OLED*

**Kết luận:**

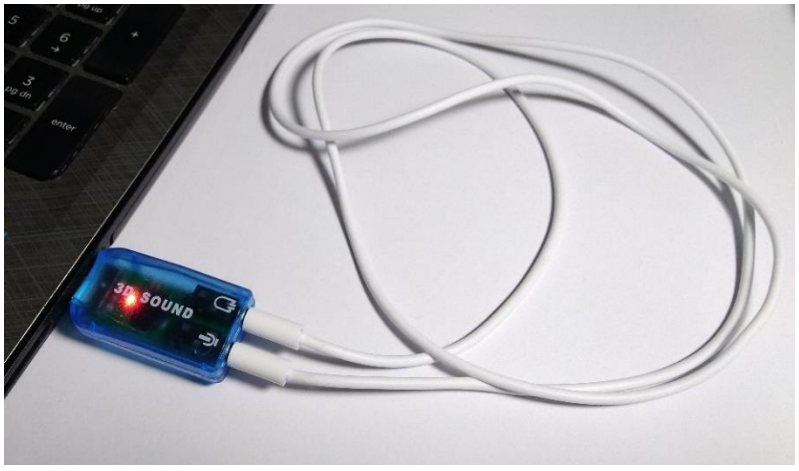
- Thông qua kết nối USB, code đã được nạp thành công trên kit. Dòng chữ OLED đã hoạt động đúng như yêu cầu. Khi kết hợp với phần mềm Multi-Instrument, việc truyền nhận dữ liệu qua cổng âm thanh đã hoạt động tốt.

b) **Tín hiệu sóng đầu ra của USB âm thanh:**

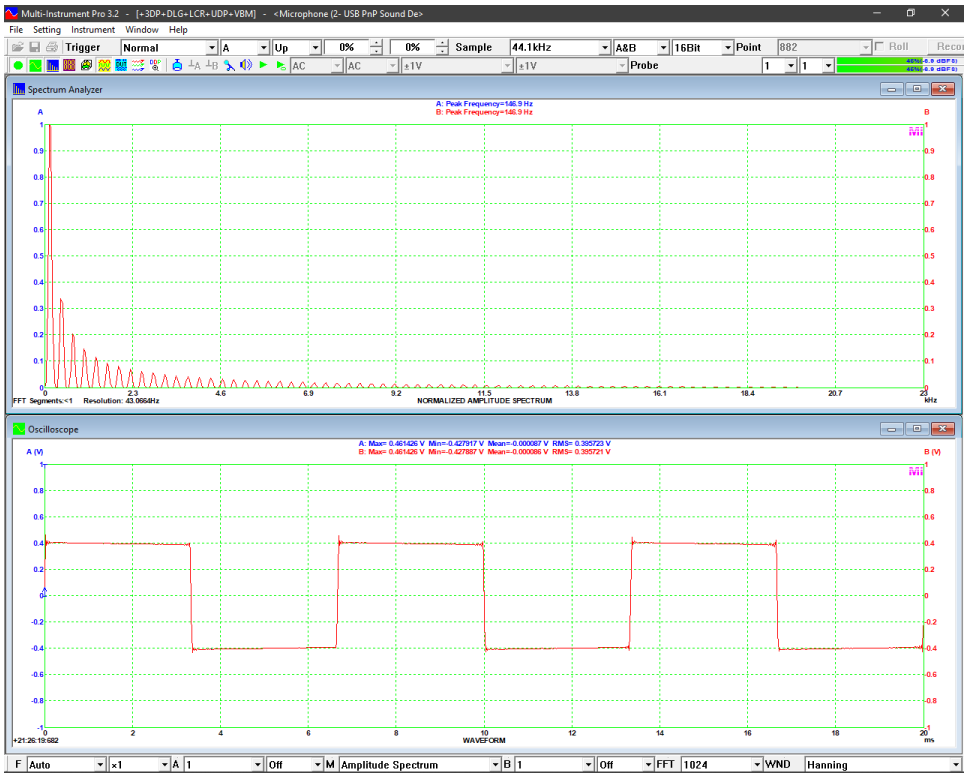
Để đo đạc dạng sóng đầu ra của USB, nhóm đã dùng một dây cáp 3.5 mm và kết nối 2 đầu của USB bằng dây cáp này. Sau đó, nhóm dùng chức năng Signal Generator của Multi-Instrument (MI) để phát một tín hiệu sóng vuông có chu kì nhiệm vụ 50% và tần số 150 Hz qua cổng Speaker của USB âm thanh. Tiếp theo, nhóm dùng chức năng Oscilloscope của Multi-Instrument để vẽ lại dạng sóng thu được trên cổng Microphone của USB âm thanh. Chi tiết về 2 chức năng này của phần mềm MI được trình bày trong Phụ lục A. Kết quả thực hiện được miêu tả ở hình dưới.




*Hình 33: Thông số của Signal Generator trên MI.*




*Hình 34: Cách nối dây để đo đạc sóng phát ra từ phần mềm*




*Hình 35: Kết quả phân tích phổ biên độ và dạng sóng trên máy tính*

**Nhận xét:**

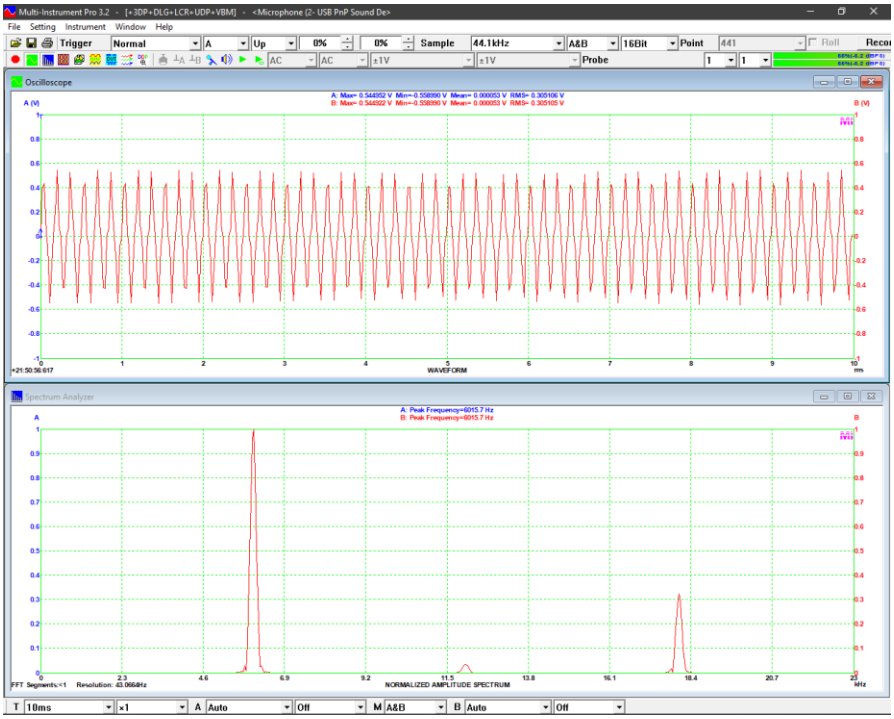
- Quan sát phổ biên độ, nhóm nhận thấy là sóng hài cơ bản có tần số là 146.9 Hz. Sai số tương đối của biên độ này so với giá trị mong muốn là $\frac{|146.9-150|.}{150} * 100\% = 2.07\%$ có thể chấp nhận được.

- Ngoài ra, trên màn hình dao động ký, sóng phát ra có dạng xung vuông có biên độ là 0.8 V với 2 đỉnh là -0.4 V và 0.4 V. Biên độ này có thể được thay đổi bằng thanh trượt Amplitude trong Signal Generator hoặc bằng nút Volume trong bảng điều khiển âm thanh của Windows 10.

- Do đó, có thể kết luận là USB âm thanh đã hoạt động tốt, có thể thực hiện gửi và nhận tín hiệu qua 2 cổng của USB âm thanh này. Còn chức năng Signal Generator của phần mềm MI đã phát sóng có dạng và tần số đúng có sai số không đáng kể và chấp nhận được.

c) **Tín hiệu sóng mang phát ra bởi kit:**

Để đo đạc và vẽ lại dạng sóng của sóng mang phát ra bởi kit, nhóm đã thực hiện thiết lập hệ thống như đã trình bày ở mục 3.a và chỉnh sửa lại code trên kit để kit có thể phát ra tín hiệu sóng mang $f_c$ = 6 kHz. Kết quả thực hiện như sau:



*Hình 36: Dạng sóng sine 6 kHz thu được trên MI.*

**Nhận xét:**

- Quan sát phổ biên độ của sóng, nhóm nhận thấy là tần số hài cơ bản của sóng là 6015.7 Hz, sai số tương đối so với giá trị mong muốn là $\frac{|6015.7-6000|}{6000} * 100\% = 0.26\ \%$

- Ngoài ra, với tần số lấy mẫu sóng 44 100 Hz, tần số cao nhất có thể đo được là 22100 Hz theo định lý lấy mẫu Nyquist. Do đó, chức năng Oscilloscope đã đo được 2 tần số của sóng hài bậc cao khác là tần số 12 000 Hz và 18 000 Hz.

- Lý do đo được 2 tần số 12 000 Hz và 18 000 Hz trong dạng sóng mang sine của nhóm là vì nhóm lấy 4 mẫu / 1 chu kì sóng mang như đã trình bày ở phần 2.h). Do đó, sóng mang sine chính là các sóng tam giác có tần số 6000 Hz. Khi phân tích Fourier dạng sóng tam giác tạo ra bởi Function Generator, ta sẽ thu được công thức như sau:

$$f(t) = \sum_{n=1,3,5,..}^{\infty} \frac{-8}{n^2\pi^2}\cos(n\omega t)\,, với\ \omega = 2\pi * 6000 \quad [11]$$

- Quan sát công thức ta thấy sóng tam giác các sóng hài bậc cao ở 6000 * k Hz với k lẻ trong trường hợp lý tưởng. Trong thực tế, USB âm thanh đã thu được thành phần sóng có biên độ rất bé, gần bằng 0 ở 12 000 Hz và thành phần sóng có biên độ sóng lớn hơn tại 18 000 Hz, phù hợp với công thức trên.

- Do đó nhóm kết luận rằng với 4 mẫu / 1 chu kì, sóng mang tạo ra bởi kit có tần số là 6 000 Hz đúng như yêu cầu của project. Tuy nhiên, do có xuất hiện sóng hài ở 12000 Hz và 18000 Hz, việc thực hiện lọc thông thấp sau khi đã điều chế trên kit là một việc cần thiết để lọc bỏ các tín hiệu tần số cao và đảm bảo tín hiệu đã điều chế là một tín hiệu bandpass.

d) **Tín hiệu đã điều chế tính toán trên kit và nhận được máy tính:**

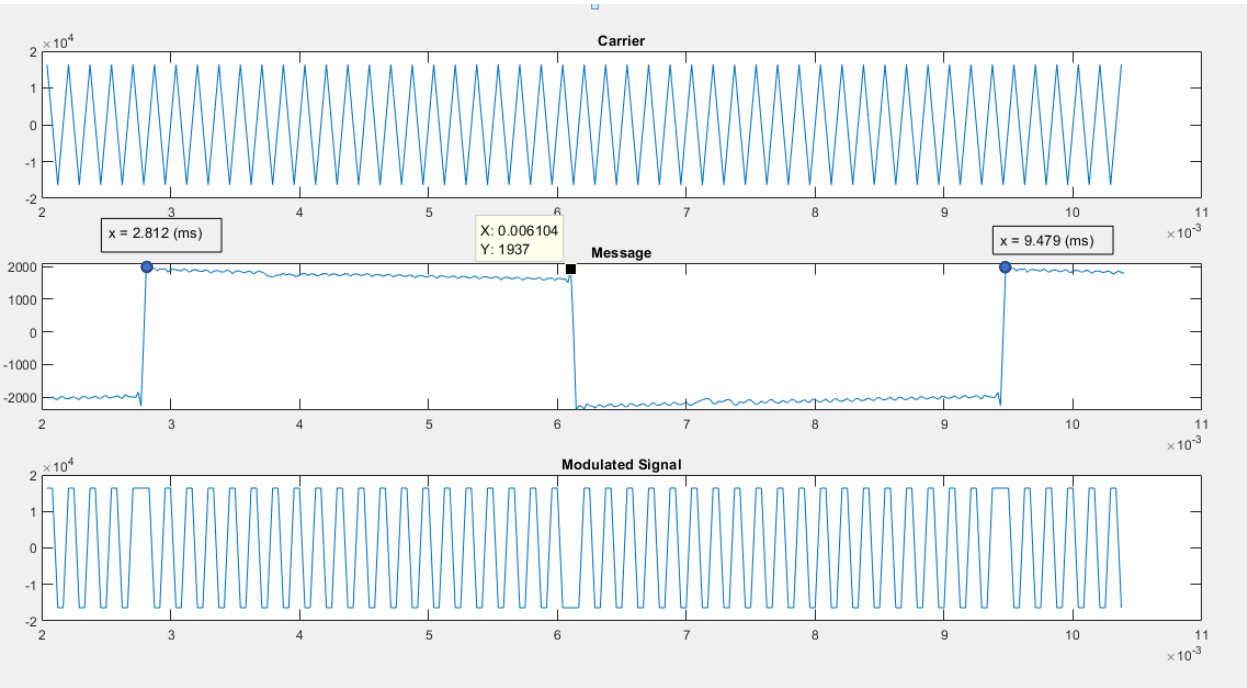
**i) Tín hiệu đã điều chế tính toán trên kit:**

Để vẽ lại tín hiệu này, nhóm đã viết một đoạn code trên MATLAB để đọc các .bin xuất ra từ kit. Đoạn code sau sẽ vẽ các đồ thị về thành phần sóng mang sine, thông điệp và tín hiệu đã điều chế trong cùng 1 figure của MATLAB:

**bin_reader.m**

```
clear all; clc; close all;
name1 = 'carrier.bin';
name2 = 'message.bin';
name3 = 'modulated.bin';
t1 = 0:1/48000:(3199*1/48000);
t2 = 0:2/48000:(3199*1/48000);

f = fopen(name1, 'rb');
md = fread(f, Inf, 'int16','ieee-be');
md = md';
fclose(f);
subplot(3,1,1);
plot(t2(50:250), md(50:250));
title('Carrier');

f = fopen(name2, 'rb');
md = fread(f, Inf, 'int16','ieee-be');
md = md';
fclose(f);
subplot(3,1,2);
plot(t1(100:500), md(100:500));
title('Message');

f = fopen(name3, 'rb');
md = fread(f, Inf, 'int16','ieee-be');
md = md';
fclose(f);
subplot(3,1,3);
plot(t2(50:250),md(50:250));
title('Modulated Signal');
```

Kết quả thu được như sau:




*Hình 37: Kết quả phân tích dữ liệu về sóng mang, thông điệp đọc được và tín hiệu đã điều chế trên file .bin.*

**Nhận xét:**

- Khi xét tín hiệu có 2 mức điện áp là 1 và 0, ta có công thức điều chế là:

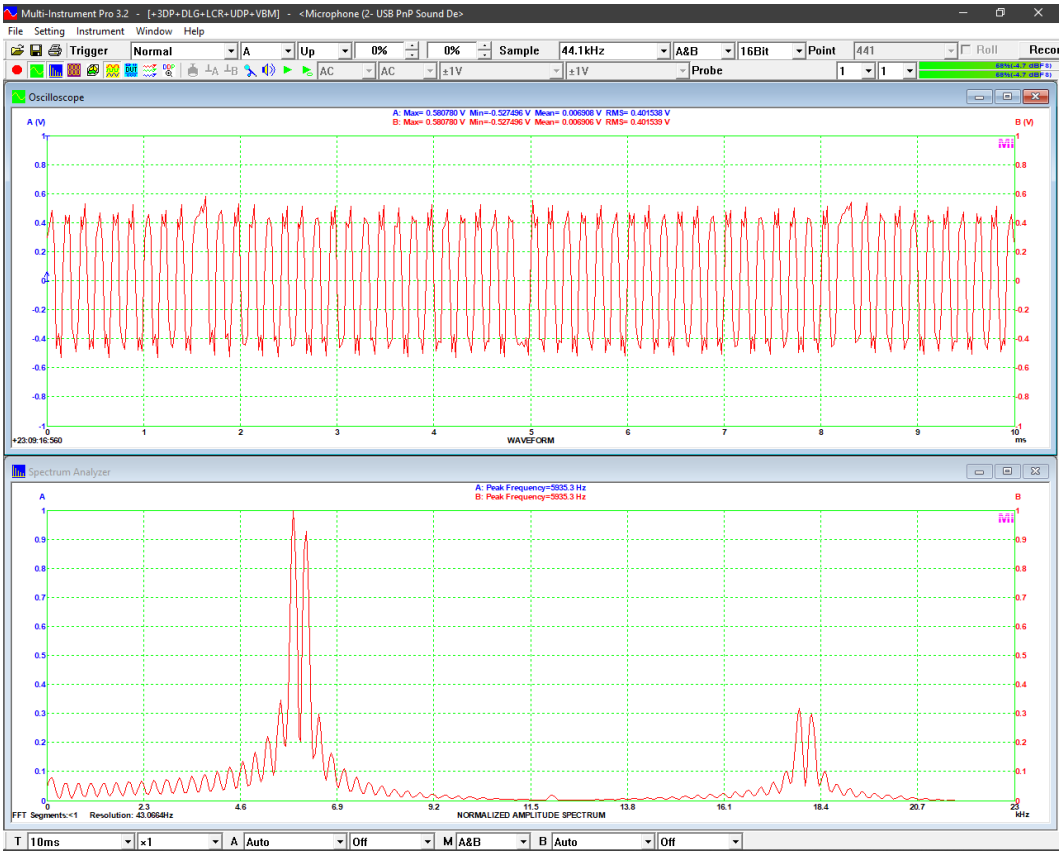
$$x_{mod} = \begin{cases} \cos(\omega_c t) - \sin(\omega_c t) = \sqrt{2}\sin\left(\omega_c t - \frac{5\pi}{4}\right), \text{ứng với mức } 1, I = Q = 1 \\ -\cos(\omega_c t) + \sin(\omega_c t) = \sqrt{2}\sin\left(\omega_c t - \frac{\pi}{4}\right), \text{ứng với mức } 0, I = Q = -1 \end{cases} \quad [12]$$

- Quan sát đồ thị trên, sử dụng chức năng Cursor, nhóm tính được chu kì của thông điệp là (9.479-2.812)*$10^{-3}$ = 6.667*$10^{-3}$ s, tương ứng 149.99 Hz. Mặc khác, xét tại thời điểm t = 4 ms, ứng với giá trị 1 của thông điệp, nhóm nhận thấy sóng đã điều chế có 1 sự trễ pha $\frac{5\pi}{4}$ ( ứng với trễ 5 mẫu ) so với sóng mang sine. Còn khi xét tại thời điểm t = 8 ms, ứng với giá trị 0 của thông điệp, sóng đã điều chế có sự trễ pha $\frac{\pi}{4}$ (ứng với trễ 1 mẫu) so với sóng mang sine. Như vậy, các giá trị tính toán trên kit của sóng đã điều chế phù hợp với công thức tính toán trên.

**ii) Tín hiệu điều chế nhận được Oscilloscope của phần mềm Multi-Instrument:**

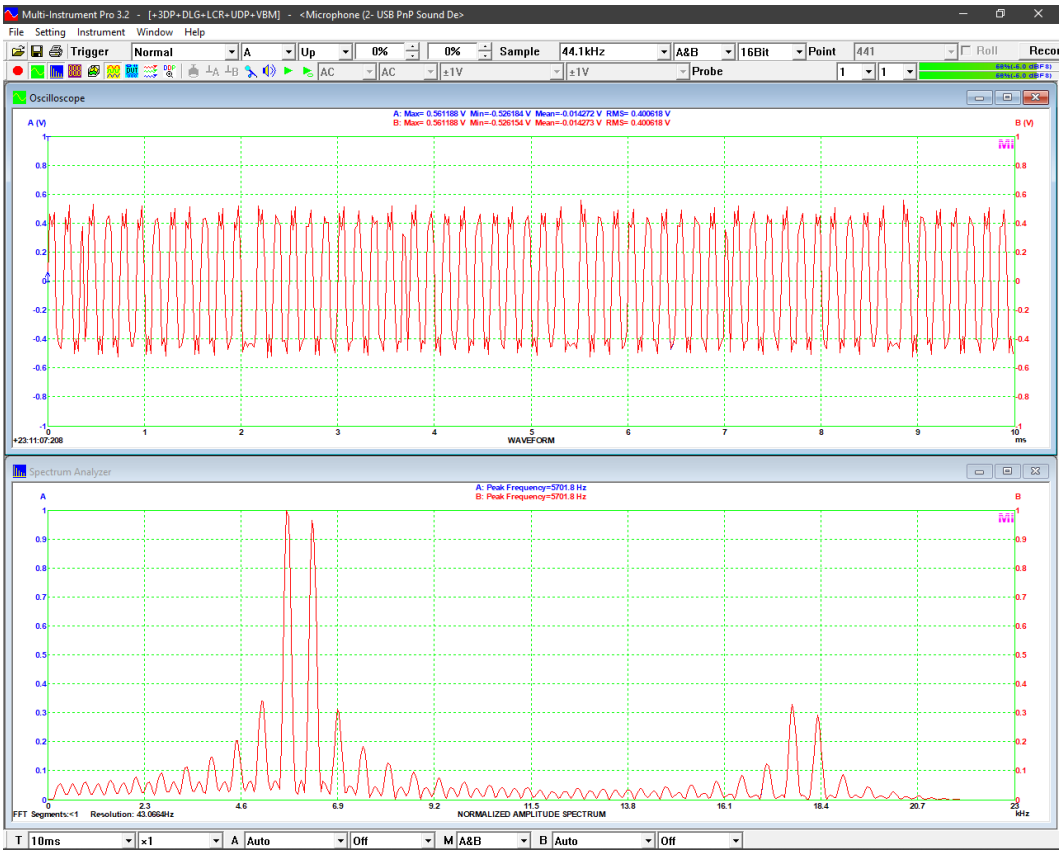
Sau khi đã hoàn thành điều chế, kit sẽ gửi kết quả đến AIC3204 dưới dạng một số nguyên 16 bit có dấu. Thông qua bộ DAC 16 bit, AIC3204 dựa vào giá trị này để xuất ra tín hiệu điện áp Analog tương ứng. Để kiểm chứng công thức về phổ tần số ở [2], nhóm đã dùng chức năng Oscilloscope và Spectrum Analyzer của phần mềm Multi-Instrument để xem dạng sóng theo thời gian và phổ biên độ của sóng nhận được. Mặc khác, nhóm cũng sẽ thay đổi tần số số của sóng vuông đầu vào để quan sát sự thay đổi của phổ biên độ. Nhóm đã thu được kết quả như sau:

- Đối với sóng vuông 150 Hz, chu kì nhiệm vụ 50%, biên độ là 1:



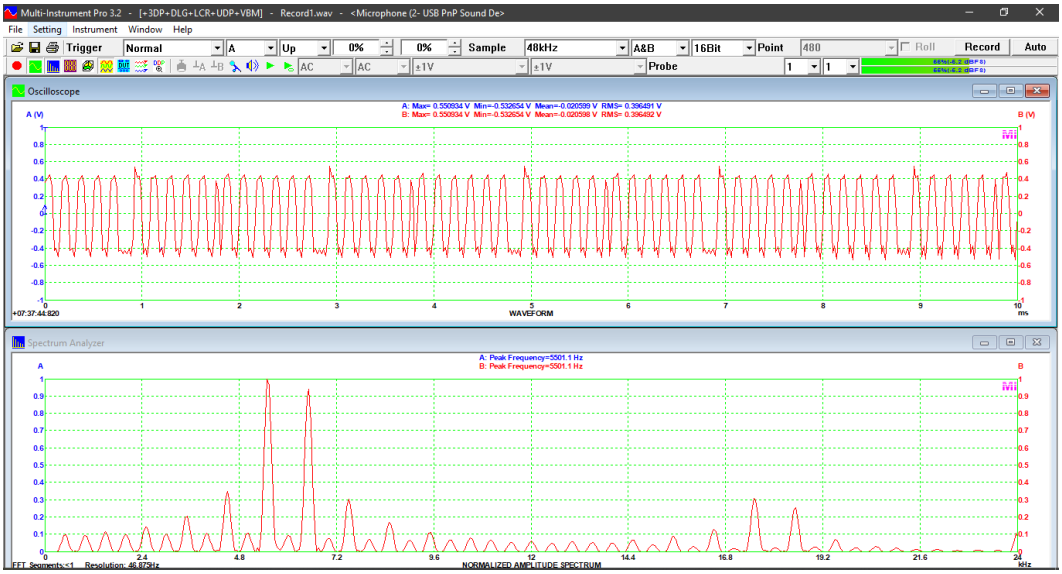

*Hình 38: Tín hiệu đã điều chế thu được trên Multi-Instrument đối với sóng vuông có tần số 150 Hz.*

- Đối với sóng vuông 300 Hz, chu kì nhiệm vụ 50%, biên độ là 1:



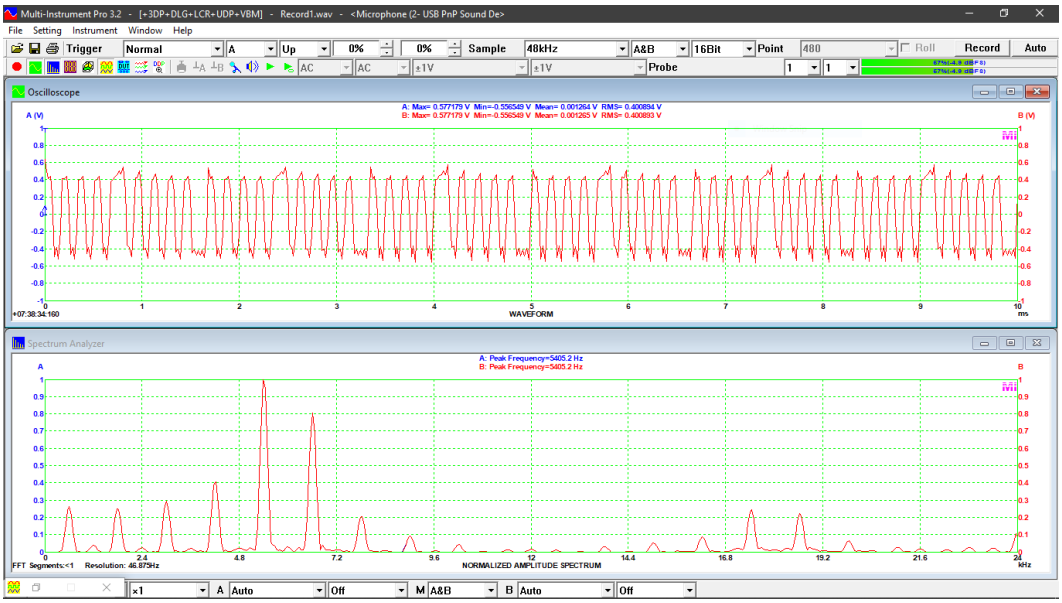

*Hình 39: Tín hiệu đã điều chế thu được trên Multi-Instrument đối với sóng vuông có tần số 300 Hz.*

- Đối với sóng vuông 500 Hz, chu kì nhiệm vụ 50%, biên độ là 1:




*Hình 40: Tín hiệu đã điều chế thu được trên Multi-Instrument đối với sóng vuông có tần số 500 Hz*

- Đối với sóng vuông 600 Hz, chu kì nhiệm vụ 50%, biên độ là 1:




*Hình 41: : Tín hiệu đã điều chế thu được trên Multi-Instrument đối với sóng vuông có tần số 600 Hz*

**Nhận xét:**

- Theo công thức [2] , các thành phần sóng hài chủ yếu nằm ở f-$f_c$ = 6000 – 150 = 5850 Hz và f+$f_c$ = 6000 +150 = 6150. Nhưng trong thực tế, ở 150 Hz, Spectrum Analyzer đã hiển thị thành phần sóng hài có biên độ lớn nhất là 5935.3 Hz và chênh lệch $\frac{|5935.3-5850|}{5850} * 100\% = 1.45\%$ so với mong đợi. Còn trong các trường hợp 300 Hz, 500 Hz, 600 Hz, các thành phần sóng hài có biên độ lớn nhất đều đáp ứng phương trình [2].

- Ngoài ra, nhờ Spectrum Analyzer, nhóm cũng đã quan sát được các thành phần sóng hài bậc cao ở $18\,000 \pm f_m$ (Hz). Các sóng hài này xuất hiện vì trên kit, nhóm chưa thực hiện công đoạn lọc thông thấp các tín hiệu đã điều chế trước khi xuất ra cổng STEREO OUT. Nhưng nhờ đã lưu lại dữ liệu sóng trên máy tính, nhóm có thể thực hiện công đoạn này sau bằng cách sử dụng các bộ lọc trên phần mềm MATLAB trước khi giải điều chế.

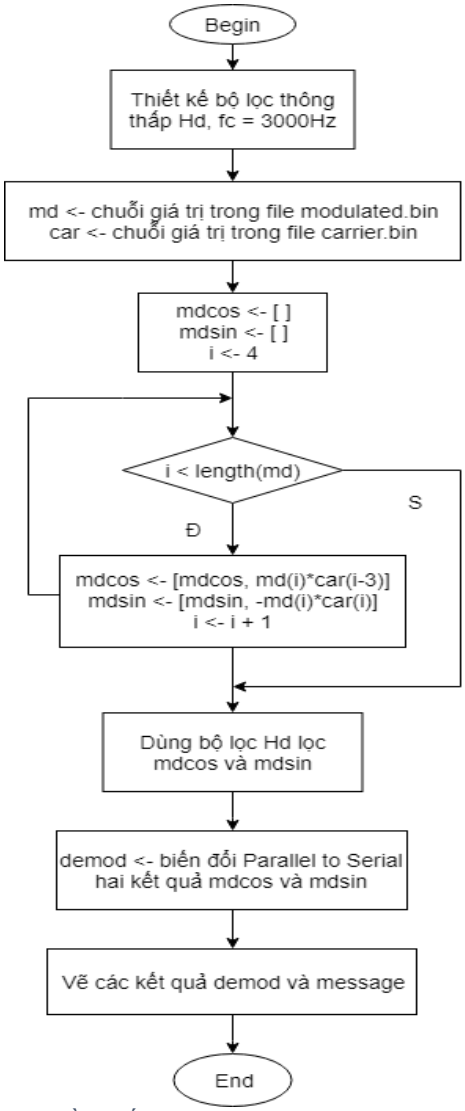
- Nhóm chỉ mới thu thập dữ liệu đối với dữ liệu 2 mức là 1 và 0, ứng với dibit là 11 và 00 theo quy ước của nhóm. Do đó, ta có thể nhận biết được ranh giới giữa 2 bit 1 và 0 cạnh nhau bằng cách tìm các đỉnh của sóng nhận được trên Oscilloscope có bề rộng lớn gấp đôi so với các đỉnh bình thường kề cận. Hiện tượng này gây ra bởi sự đảo biên độ của sóng nhận được so với sóng mang (vì theo công thức [12], 2 sóng này ngược pha nhau) bằng cách xem xét các đỉnh của sóng đã điều chế trên Oscilloscope tại các điểm phân cách giữa 2 bit 1 và 0 cạnh nhau. Nếu kết hợp với

các sóng vuông có 4 mức 0, 1, 2, 3, nhóm dự đoán rằng ta có thể quan sát thêm được sự trễ pha của sóng đã điều chế.

- Cuối cùng, tổng kết lại thì việc thực hiện điều chế trên kit đã thành công, có dạng sóng và phổ biên độ như mong muốn. Tuy nhiên do thiếu sót về mặt thu thập dữ liệu, nhóm đã chưa thể kiểm chứng được chương trình khi đầu vào là sóng vuông có 4 mức biên độ.

**III. Giải điều chế 4-QAM bằng MATLAB**

Phần này mô tả giải thuật và trình bày code để giải điều chế 4-QAM sử dụng phần mềm MATLAB R2014a.

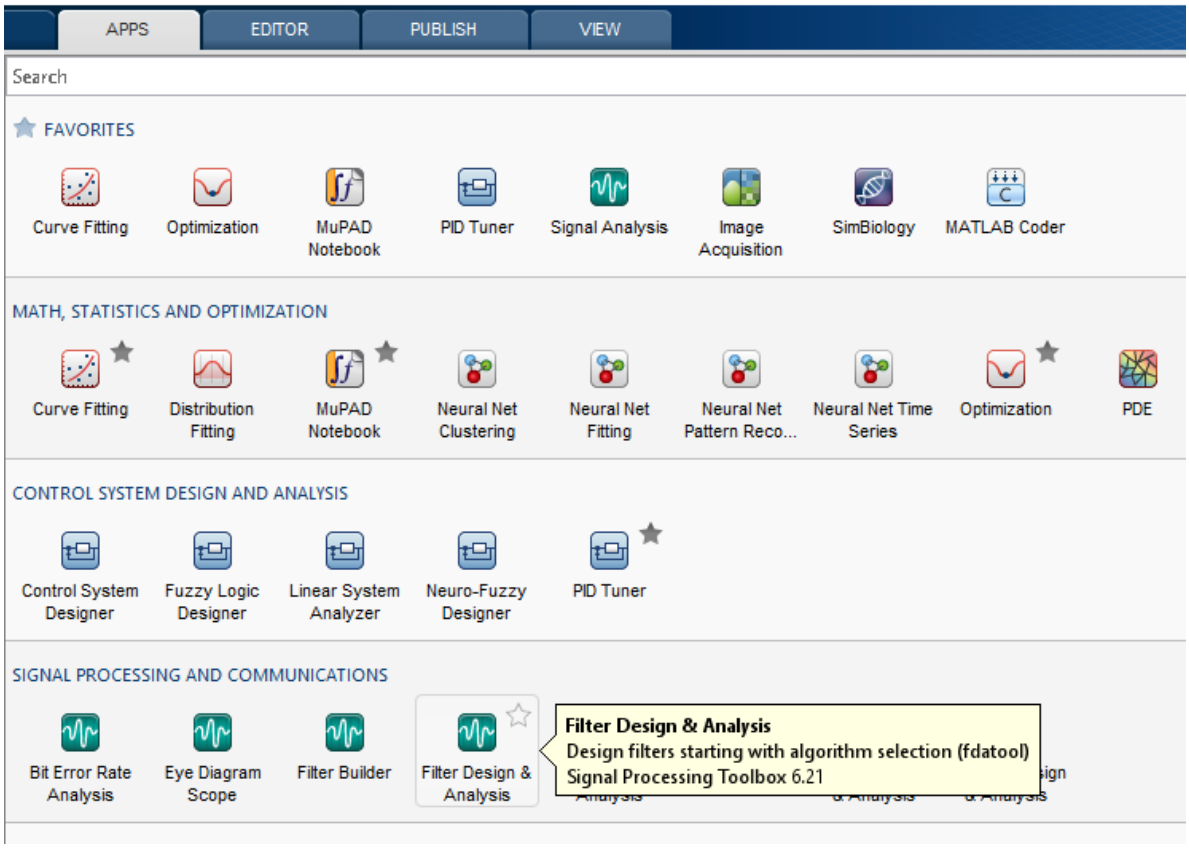



*Hình 42: Sơ đồ giải thuật giải điều chế 4-QAM các dữ liệu thu được từ kit trên MATLAB*

Sử dụng sơ đồ giải điều chế (hình 3) được nêu trong phần I., ta xây dựng được sơ đồ giải thuật cho MATLAB (hình 42). Dữ liệu vào sử dụng 03 file binary chứa các tín hiệu sóng mang carrier, tín hiệu đã được điều chế modulated signal và tín hiệu điều chế message. Kit C5515 được lập trình để lưu lại các file này. Do dữ liệu đầu vào là các chuỗi số nên khối A/D và bộ lượng tử hóa không cần thiết. Kết thúc quá trình giải điều chế, tín hiệu demodulated signal và tín hiệu message sẽ được vẽ để so sánh dạng sóng.

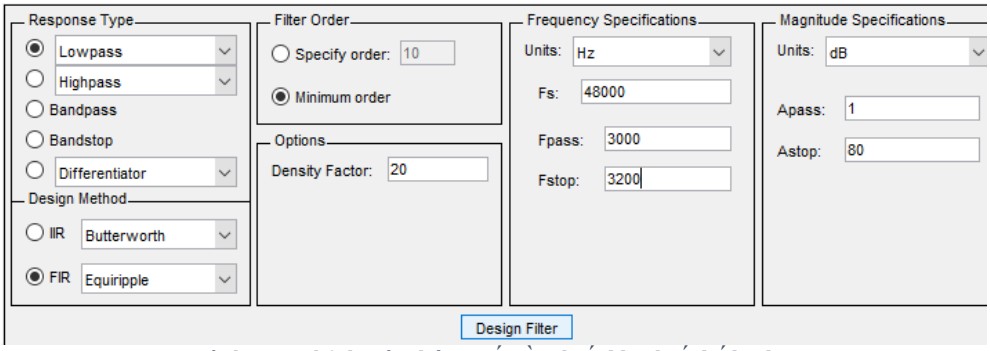
Trong 1 chu kỳ, tín hiệu carrier được lấy 4 giá trị, khoảng cách giữa 2 giá trị kề nhau là 1/24000 giây. Thật vậy, ta có: $4 \times 1/24000 = 1/6000$ đúng với chu kỳ yêu cầu của sóng carrier ($f_c$ = 6000 Hz). Điều này quan trọng vì sóng carrier được sử dụng cho quá trình giải điều chế để tránh hiện tượng crosstalk.

Thuật toán sử dụng bộ lọc thông thấp với tần số cắt $f_c$ = 3000Hz đóng vai trò lọc các tín hiệu mang tần số cao của sóng carrier. Bộ lọc được tạo nhờ App Filter Design & Analysis có sẵn trong MATLAB. Để tạo bộ lọc phù hợp, ta chọn biểu tượng Filter Design & Analysis trong thẻ APPS.

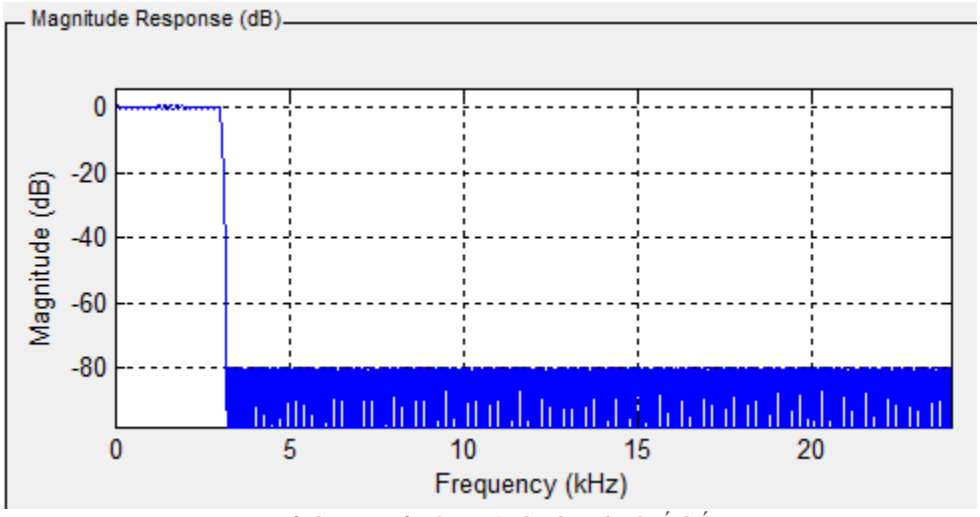



*Hình 43: App Filter Design & Analysis trong MATLAB R2014*

Chỉnh thông số: trong ô Response Type chọn Lowpass, ô Frequency Specifications thay đổi các giá trị Tần số lấy mẫu Fs = 48000Hz, Fpass = 3000Hz, Fstop = 3200Hz. Sau khi chỉnh xong, ta nhấn Design Filter để tiếp tục, lúc này MATLAB sẽ tự tạo bộ lọc theo các thông số được điều chỉnh.

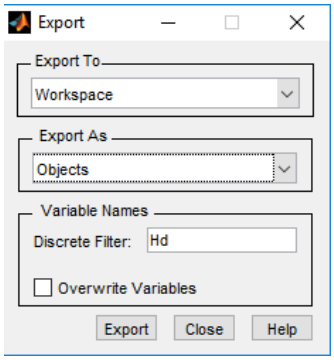



*Hình 44: Chỉnh các thông số cần thiết khi thiết kế bộ lọc*




*Hình 45: Đáp ứng của bộ lọc đã thiết kế*

Chọn File -> Export, trong thẻ Export as chọn Object để tạo biến object Hd trong Workspace.




*Hình 46: Tạo biến object Hd trong Workspace.*

Chọn File -> Generate MATLAB Code -> Filter Design Function, chọn địa chỉ lưu file và đặt tên file. Lúc này MATLAB sẽ sinh code cho bộ lọc vừa tạo với định dạng file .m. Hàm được sinh ra chứa các giá trị của bộ lọc đã được thiết kế trước đó.

**LPF.m**

```
function Hd = LPF
%LPF Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 8.3 and the DSP System Toolbox
8.6.
% Generated on: 20-Nov-2018 09:25:25

% Equiripple Lowpass filter designed using the FIRPM
function.

% All frequency values are in Hz.
Fs = 48000;  % Sampling Frequency

Fpass = 3000;            % Passband Frequency
Fstop = 3200;            % Stopband Frequency
Dpass = 0.057501127785;  % Passband Ripple
Dstop = 0.0001;          % Stopband Attenuation
dens  = 20;              % Density Factor

% Calculate the order from the parameters using
FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fpass, Fstop]/(Fs/2), [1 0],
[Dpass, Dstop]);

% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);

% [EOF]
```

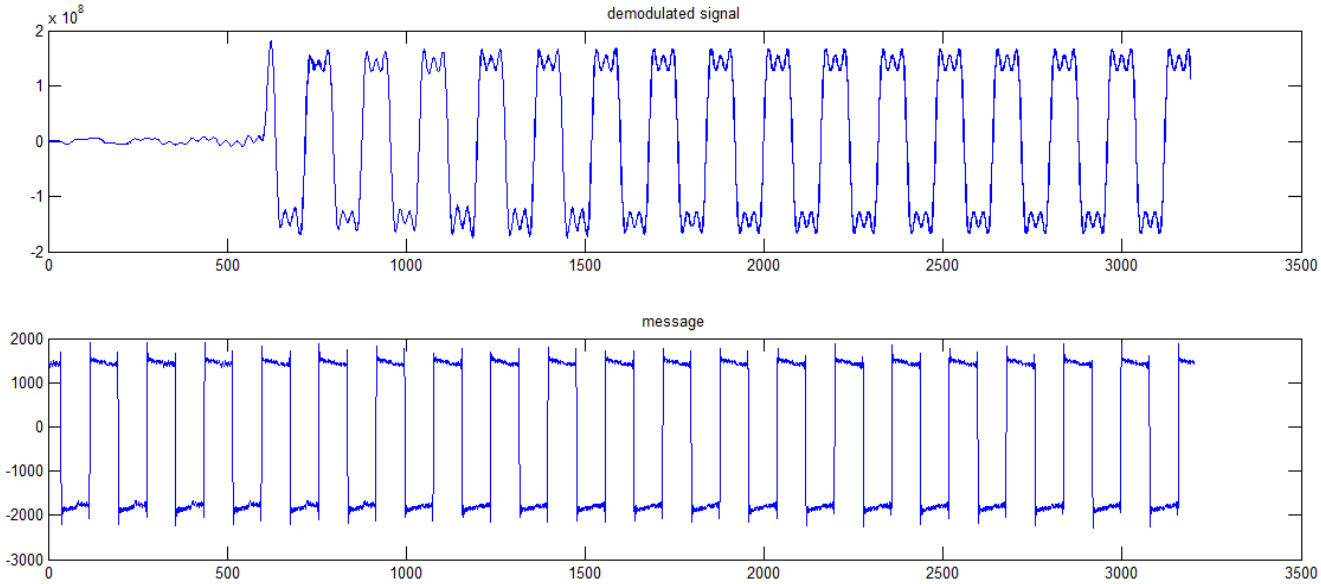
Biến md và car lần lượt là mảng lưu các giá trị tín hiệu đã được điều chế (modulated) và các giá trị của sóng mang (carrier). Hai mảng mdcos và mdsin lúc đầu được khai báo rỗng để chứa các giá trị I và Q. Sóng mang carrier được dùng là sóng sin, nên để tìm sóng mang cosine và áp dụng vào công thức [1] $r(t) = k_1 * \cos(2\pi f_c t) - k_2 * \sin(2\pi f_c t)$ , tín hiệu car được dời trái 3 giá trị để thành tín hiệu cos. Như vậy i được khởi tạo bằng 4.

Sau khi phân tích, ta có code MATLAB giải điều chế như sau:

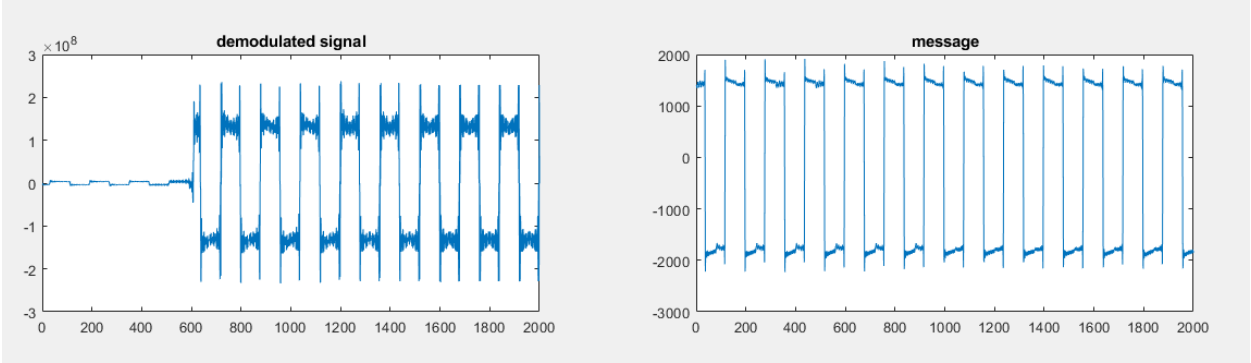
**demod.m**

```
clear all
clc

Hd = LPF;

%Read file modulated.bin
name = 'D:\matlab_bin\qam4mod\fdig300rect0.5\modulated.bin';
f = fopen(name,'rb');
md = fread(f,Inf,'int16','ieee-be');
md = md';

name = 'D:\matlab_bin\qam4mod\fdig300rect0.5\carrier.bin';
f = fopen(name,'rb');
car = fread(f,Inf,'int16','ieee-be');
car = car';

%Multiplication part
mdcos = [];
mdsin = [];

i = 4;
while i < length(md)
    mdcos = [mdcos, md(i)*car(i-3)];
    mdsin = [mdsin, -md(i)*car(i)];
    i = i + 1;
end
%LPF
filtercos = filter(Hd,mdcos);
filtersin = filter(Hd,mdsin);

%demodulated signal signal
demod = reshape([filtercos;filtersin],1,[]);
subplot(2,1,1);
t = 1:1:length(demod);
plot(t,demod);
title('demodulated signal signal');

%message
name = 'D:\matlab_bin\qam4mod\fdig300rect0.5\message.bin';
f = fopen(name,'rb');
mes = fread(f,Inf,'int16','ieee-be');
mes = mes';
subplot(2,1,2);
t = 1:1:length(mes);
plot(t,mes);
title('message');
```

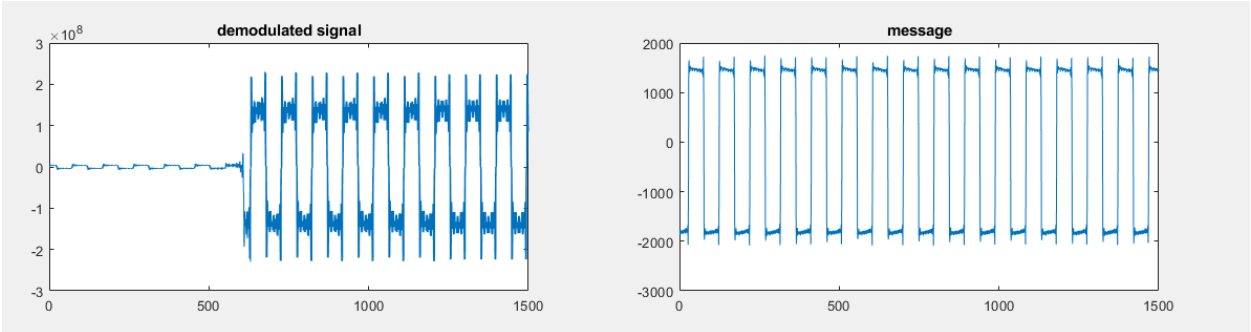
Kết quả giải điều chế:




*Hình 47: Kết quả giải điều chế 4-QAM tín hiệu xung vuông đầu vào 150 Hz*




*Hình 48: Kết quả giải điều chế 4-QAM tín hiệu xung vuông đầu vào 300 Hz*




*Hình 49: Kết quả giải điều chế 4-QAM tín hiệu xung vuông đầu vào 300 Hz*

.

**Nhận xét:**

Ta thấy dạng sóng của tín hiệu giải điều chế và tín hiệu message giống nhau, tuy nhiên biên độ hai tín hiệu khác nhau do 4-QAM không đảm bảo giữ nguyên biên độ tín hiệu message.

Ở tín hiệu đã giải điều chế, ta thấy tồn tại một độ trễ vào khoảng 600 mẫu ( ứng với 600*1/24000 = 0.025 s) tại những thời điểm đầu của tín hiệu.

Như vậy là việc điều chế và giải điều chế đã hoàn thành, dạng sóng đã được khôi phục lại trên MATLAB.

# VI. Kết luận:

Trong dự án này, nhóm chúng tôi đã hoàn thành các việc điều chế và giải điều chế đã thực hiện thành công trên MATLAB lẫn trên kit. Tuy nhiên do kiến thức còn hạn chế, nhóm đã chưa nghĩ tới việc thử nghiệm với tín hiệu đầu vào sóng vuông 4 biên độ để có thể thử nghiệm với chuỗi dibit là 10 và 01.

Trong suốt quá trình thời gian vừa qua, cả nhóm đã học được rất nhiều kiến thức mới liên quan như điều chế 4QAM, thiết kế bộ lọc số, lập trình C trên kit eZDSP, phân tích dữ liệu âm thanh bằng MATLAB. Những kiến thức này là những kiến thức cơ bản quý giá để nhóm có thể đi tiếp trong chặng đường sắp tới.

# VII. Danh mục tài liệu tham khảo:


A. Bruce Carlson, P. B. (2002). *COMMUNICATION SYSTEMS: AN INTRODUCTION TO SIGNALS AND NOISE IN ELECTRICAL*. McGraw-Hill.

Leon W. Couch, I. (2013). *DIGITAL AND ANALOG COMMUNICATIONS SYSTEMS, EIGHTH EDITION*. Pearson.

Rodger E. Ziermer, W. H. (2014). *PRINCIPLES OF COMMUNICATIONS: SYSTEMS, MODULATION AND NOISE*. Wiley.

Spectrum Digital, I. (2010). *TMS320C5515 EZDSP USB STICK TECHNICAL REFERENCE*.

Texas Instruments. (2014). *TLV320AIC3204 ULTRA LOW POWER STEREO AUDIO CODEC DATASHEET*.

# PHỤ LỤC A: Giới thiệu phần mềm giả lập máy tạo sóng và dao động ký trên PC – Multi-Instrument 3.2:

### 1) Giới thiệu chung:

Multi-Instrument 3.2 là một phần mềm nổi tiếng đến từ hãng Virtins Technology với nhiều chức năng mô phỏng lại các hệ thống đo lường, phân tích tín hiệu trên miền thời gian, tần số và thời gian-tần số như dao động ký, máy đo đa năng, máy tạo sóng, …

Bằng cách tận dụng tính năng ADC và DAC trên các card âm thanh có sẵn trong các hệ thống máy tính cá nhân, Multi-Instrument có thể biến các máy tính thành các thiết bị đo lường đa chức năng với tần số lấy mẫu có thể lên đến 768 kHz và độ phân giải lên đến 24 bits (tùy thuộc vào card âm thanh của mỗi máy). Thông thường, các card âm thanh trong các máy tính có đủ khả năng để phân tích tín hiệu trong dải âm tần (20 đến 20 000 Hz)

Để có thể kết nối giữa máy tính và kit, nhóm cần 2 loại cáp:

1- Một sợi dây cáp âm thanh có 2 đầu đực loại TRRS cổng 3.5mm.

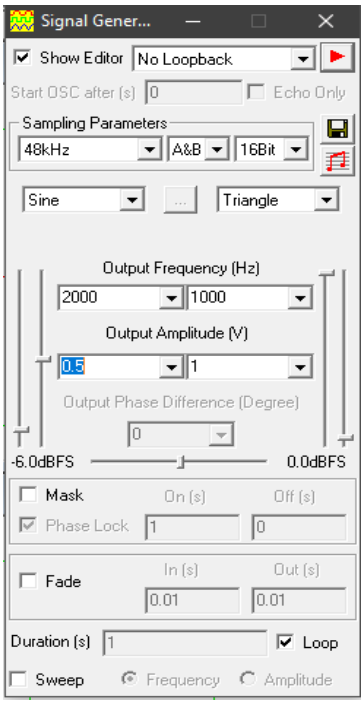
2- Một sợi dây cáp chia dữ liệu từ 1 cổng TRRS cái thành 2 cổng tai nghe – microphone dạng TRS đực riêng biệt.

Sở dĩ nhóm chọn 2 loại cáp như vậy vì trên Laptop của các thành viên trong nhóm, nhà sản xuất đã tích hợp các chức năng tai nghe – microphone vào một cổng 3.5 mm cái dạng TRRS. Còn trên kit eZDSP, 2 ngõ Line In – Line Out tồn tại tách biệt nhau dưới dạng 2 cổng cái loại TRS. Do đó cần một bộ chuyển đổi giữa 2 dạng này.

*Hình 50: Cáp chia dữ liệu ra 2 đầu tai nghe - microphone riêng biệt*

*Hình 51: Cáp 3.5mm với 2 đầu đực dạng TRRS*

## 2) Máy tạo sóng – Signal Generator:




*Hình 52: Giao diện của chức năng Signal Generator*

Một vài thông số quan trọng của Máy tạo sóng như sau:

+ Tần số lấy mẫu $F_s$ của bộ DAC: phần mềm hỗ trợ các tần số lấy mẫu từ 2 kHz đến 200 kHz.

+ Độ sâu bit: hỗ trợ 8, 16 và 24 bit.

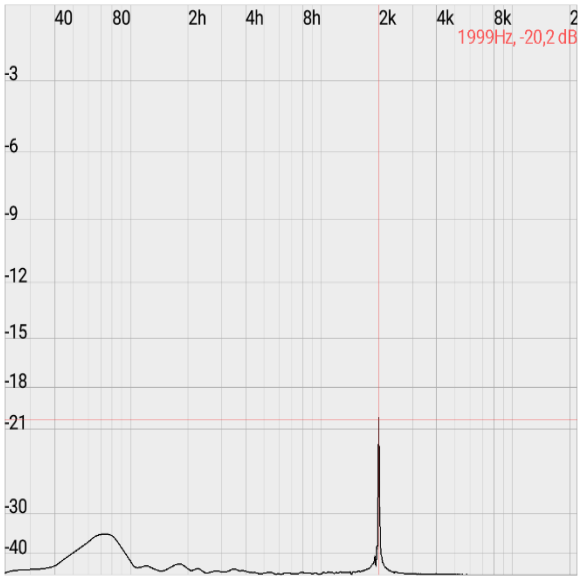
+ Số kênh: phần mềm hỗ trợ xuất sóng ra trên tối đa 2 kênh: A và B.

+ Dạng sóng: gồm các loại Sine, Rectangle, Triangle, Saw Tooth, …

+ Tần số sóng (Output Frequency): Dải tần số sóng ra khác nhau tùy thuộc vào tần số lấy mẫu và nằm trong khoảng từ 0 đến $F_s/2$ Hz.

+ Biên độ của điện áp sóng ra: Biên độ nằm trong khoảng từ 0 đến 1 V.

Nhóm đã thử nghiệm phát âm thanh dạng Sine tần số 2000Hz trên PC với máy tạo sóng trên, sau đó dùng phần mềm "Oscope" trên smartphone Android để phân tích phổ tần số. Kết quả thu được như hình dưới.




*Hình 53: Kết quả phân tích phổ tần số bằng phần mềm Oscope*

**Kết luận**:

Với 1 đỉnh có biên độ lớn nhất tại 1999 Hz trên phần mềm Oscope, chức năng Signal Generator trên Multi Instrument đã phát ra tín hiệu âm thanh có dạng sóng và tần số nằm trong khoảng chấp nhận được.

### 3) Dao động ký - Oscilloscope:

Một vài thông số quan trọng của Dao động ký như sau:

+ Tần số lấy mẫu $F_s$ của bộ ADC : tần số này nằm trong khoảng từ 2 kHz đến 200 kHz.

+ Các kênh đầu vào: phần mềm hỗ trợ hiển thị tối đa 2 kênh đầu vào.

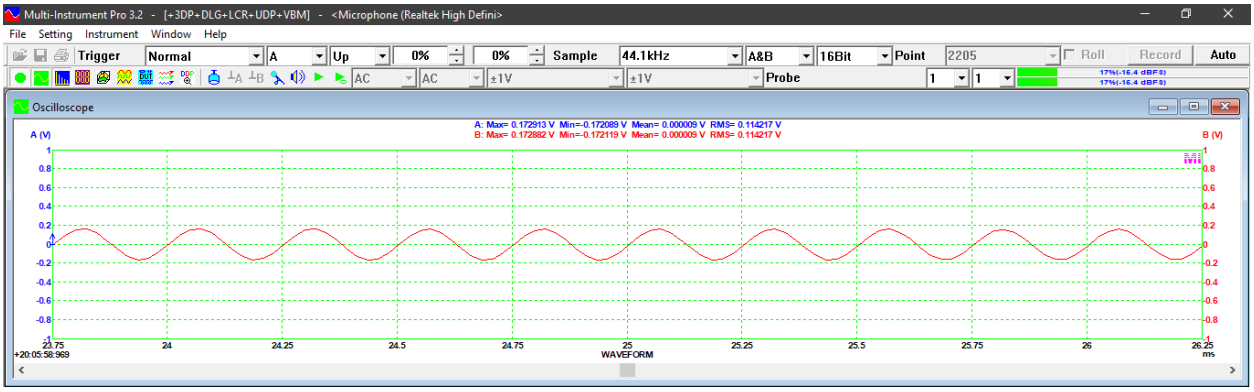
+ Độ sâu bit: hỗ trợ 8, 16 hoặc 24 bit>

+ Thời gian T của một bản ghi: thể hiện độ dài của một bản ghi và được dùng để tính toán số lượng mẫu trong 1 bản ghi qua công thức: $n = T*F_s$

+ Khoảng biên độ của sóng: thể hiện khoảng đo của điện áp đầu vào.

+ Kênh hiển thị: có thể lựa chọn hiển thị các kênh riêng biệt (A&B) và các phép toán dựa trên tín hiệu của các kênh như A+B, A-B, A*B, A/B.

Để thử nghiệm chức năng này, nhóm đã sử dụng một phần mềm tạo sóng Sine trên smartphone Android có tên là "Frequency Generator" với tần số 4000 Hz và xem dạng sóng thu được trên chức năng Oscilloscope của Multi Instrument. Kết quả thu được như sau:




*Hình 54: Kết quả dạng sóng trên Oscilloscope*

Từ đồ thị, nhóm xác định được chu kì của sóng là 0.25 ms, tương ứng với 4000 Hz. Do đó nhóm kết luận rằng, Oscilloscope đã thu được chính xác dạng sóng và âm thanh phát ra từ smartphone và từ đó có thể hiển thị lên biểu đồ theo thời gian thực.

### 4) Máy phân tích phổ - Spectrum Analyzer:

Các chức năng của máy phân tích phổ là:

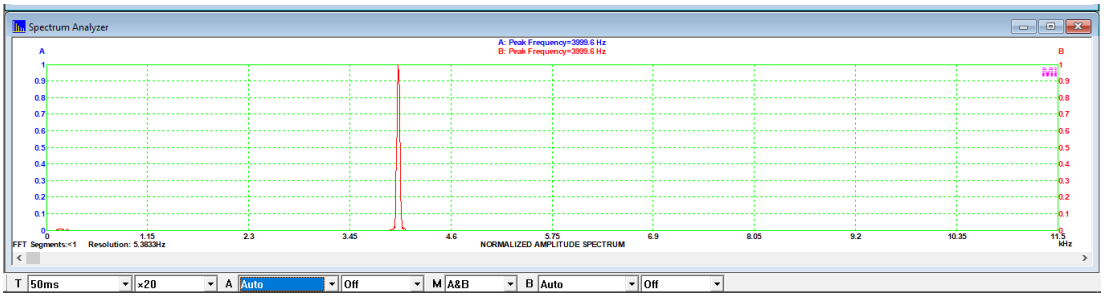
+ Dải tần số phân tích: phần mềm hỗ trợ từ 1 Hz đến 100 GHz.

+ Loại thông số hiển thị: phần mềm hỗ trợ tính toán và hiển thị các thông số như: Amplitude Spectrum, Phase Spectrum, Auto Correlation, Transfer Function,…

+ Số điểm FFT: phần mềm hỗ trợ tính toán FFT với $2^k$ điểm, k từ 7 đến 22.

+ Cửa sổ FFT: phần mềm phân tích phổ với thuật toán FFT với các loại cửa sổ như: Rectangle, Triangle, Hanning, Hamming, Blackman,…

Nhóm tiếp tục dùng phần mềm "Frequency Generator" để thử nghiệm với sóng Sine có tần số 4000 Hz. Kết quả phân tích Amplitude Spectrum thu được trên Spectrum Analyzer của Multi Instrument như hình dưới:




*Hình 55: Kết quả phân tích phổ biên độ trên Spectrum Analyzer.*

**Kết luận:**

Với 1 đỉnh hài cao nhất tại 4000 Hz, chức năng Spectrum Analyzer đã hoạt động tốt và cho kết quả đúng như mong muốn.

### 5) Kết luận:

Yêu cầu về tín hiệu trong project của nhóm là $f_{dig}$ = 150 Hz, $f_c$ = 6000 Hz, các tần số yêu cầu đều nằm trong dải âm tần. Tất cả các chức năng mà nhóm cần để thực hiện project là máy tạo sóng, dao động ký và máy phân tích phổ mật độ công suất đều được hỗ trợ trong phần mềm.

Ngoài ra, bằng cách thực hiện thu/ phát dữ liệu trên một smartphone có cài các phần mềm phát sóng/ phân tích phổ khác nhau như: Frequency Generator, Oscope, nhóm có thể kết luận rằng phần mềm Multi-Instrument đã hoạt động đúng như yêu cầu. Do đó, nhóm quyết định chọn Multi-Instrument 3.2 để sử dụng trong project này.

## PHỤ LỤC B: Chi tiết toàn bộ code C trên kit eZDSP:

**Main.c**

```
     #include "AIC3204.h"
     #include "sinetable.h"
     #include "oled.h"
     #include "utilities.h"

     #define THRESHOLD  200

     void main( void )
     {
          // Declare variables:
         Int8 j = 0;
         Int16 i = 0;
         Int16 id = 0;

         Int8 sample;
         Int16 mes1, mes2;
         Int8  I, Q;
         Int16 modulated_signal;

          static Int16 message_bin[3200],       *pmes =
message_bin;
          static Int16 carrier_bin[1600],           *pcar =
carrier_bin;
          static Int16 modulated_sig_bin[1600],     *pmod =
modulated_sig_bin;

          Int8 logFlag = 0;

         // Initialize BSL
         USBSTK5515_init( );

          // Enable and Initialize I2C:
          SYS_EXBUSSEL = 0x6100;
          USBSTK5515_I2C_init( );

         // Configure AIC3204:
          AIC3204_init();

          // Configure OSD9616:
          OSD9616_init();
```

```
          // Display text:
          OSD9616_displayText();

         // Receiving and modulating the message for 10
second = 480000 samples:
         for ( i = 0 ; i < 10 ; i++ )
         {
          id = 0;
              for ( j = 0 ; j < 12000 ; j++ )
              {
                  for ( sample = 0 ; sample <
samplesPerPeriod ; sample++ )
                  {
                    // Read Analog Audio
                    while((Rcv & I2S0_IR) == 0);  // Wait for
interrupt pending flag
                      mes1 = I2S0_W0_MSW_R;  // 16 bit left
channel received 1st audio data.

                    while((Rcv & I2S0_IR) == 0);  // Wait for
interrupt pending flag
                      mes2 = I2S0_W0_MSW_R;  // 16 bit left
channel received 2nd audio data.

                        // Convert message to I,Q component:
                        if ((mes1 <= -THRESHOLD) && (mes2 <=
-THRESHOLD))
                        {    I = -1;
                             Q = -1;
                        }
                        else if ((mes1 <= -THRESHOLD) &&
(mes2 > THRESHOLD))
                        {    I = -1;
                             Q = 1;
                        }
                        else if ((mes1 > THRESHOLD) && (mes2
<= -THRESHOLD))
                        {    I = 1;
                             Q = -1;
                        }
                        else if ((mes1 > THRESHOLD) && (mes2
> THRESHOLD))
                        {    I = 1;
                             Q = 1;
```

```
                        }
                        else
                        {
                             I = 0;
                             Q = 0;
                        }

                        // Modulate signal
                        modulated_signal =
I*cosinetable[sample] - Q*sinetable[sample]; //
I*cos(2*pi*fc*t) - Q*sin(2*pi*fc*t)

                        // Write the modulated signals to
left channel:
                      while((Xmit & I2S0_IR) == 0);
          // Wait for interrupt pending flag
                         I2S0_W0_MSW_W = modulated_signal;
          // Write the modulated signal to 16 bit left
channel.
                      I2S0_W0_LSW_W = 0;

                      I2S0_W1_MSW_W = 0;
                      I2S0_W1_LSW_W = 0;

                      // Log the data:
                   if ((sample ==0) && (j == 8000) && (i ==
9))
                        logFlag = 1;
                      if (logFlag == 1)
                      {
                          message_bin[2*id] = mes1;
                          message_bin[2*id+1] = mes2;
                          carrier_bin[id] =
sinetable[sample];
                          modulated_sig_bin[id] =
modulated_signal;
                        id = id + 1;

                      // Save the carrier, message and
modulated signal;
                        if (id == 1600)
                        {
                             logFlag = 0;
```

```
    ExportFile("..\\output\\message.bin",
3200,(Int16*) pmes );

    ExportFile("..\\output\\carrier.bin",
1600,(Int16*) pcar );

    ExportFile("..\\output\\modulated.bin",
1600,(Int16*) pmod );
                        }
                      }
                  }
              }
         }
         /* Disable I2S */
         I2S0_CR = 0x00;


    }
```

**AIC3204.c**

```
     #include "AIC3204.h"
     #include "sinetable.h"
     #include "oled.h"
     #include "utilities.h"

     #define THRESHOLD  200

     void main( void )
     {
          // Declare variables:
         Int8 j = 0;
         Int16 i = 0;
         Int16 id = 0;

         Int8 sample;
         Int16 mes1, mes2;
         Int8  I, Q;
         Int16 modulated_signal;

          static Int16 message_bin[3200],       *pmes =
message_bin;
          static Int16 carrier_bin[1600],           *pcar =
carrier_bin;
          static Int16 modulated_sig_bin[1600],     *pmod =
modulated_sig_bin;

          Int8 logFlag = 0;

         // Initialize BSL
         USBSTK5515_init( );

          // Enable and Initialize I2C:
          SYS_EXBUSSEL = 0x6100;
          USBSTK5515_I2C_init( );

         // Configure AIC3204:
          AIC3204_init();

          // Configure OSD9616:
          OSD9616_init();

          // Display text:
```

```
          OSD9616_displayText();

         // Receiving and modulating the message for 10
second = 480000 samples:
         for ( i = 0 ; i < 10 ; i++ )
         {
          id = 0;
              for ( j = 0 ; j < 12000 ; j++ )
              {
                  for ( sample = 0 ; sample <
samplesPerPeriod ; sample++ )
                  {
                    // Read Analog Audio
                    while((Rcv & I2S0_IR) == 0);  // Wait for
interrupt pending flag
                      mes1 = I2S0_W0_MSW_R;  // 16 bit left
channel received 1st audio data.

                    while((Rcv & I2S0_IR) == 0);  // Wait for
interrupt pending flag
                      mes2 = I2S0_W0_MSW_R;  // 16 bit left
channel received 2nd audio data.

                        // Convert message to I,Q component:
                        if ((mes1 <= -THRESHOLD) && (mes2 <=
-THRESHOLD))
                        {    I = -1;
                             Q = -1;
                        }
                        else if ((mes1 <= -THRESHOLD) &&
(mes2 > THRESHOLD))
                        {    I = -1;
                             Q = 1;
                        }
                        else if ((mes1 > THRESHOLD) && (mes2
<= -THRESHOLD))
                        {    I = 1;
                             Q = -1;
                        }
                        else if ((mes1 > THRESHOLD) && (mes2
> THRESHOLD))
                        {    I = 1;
                             Q = 1;
                        }
```

```
                        else
                        {
                             I = 0;
                             Q = 0;
                        }

                        // Modulate signal
                        modulated_signal =
I*cosinetable[sample] - Q*sinetable[sample]; //
I*cos(2*pi*fc*t) - Q*sin(2*pi*fc*t)

                        // Write the modulated signals to
left channel:
                      while((Xmit & I2S0_IR) == 0);
          // Wait for interrupt pending flag
                        I2S0_W0_MSW_W = modulated_signal;
          // Write the modulated signal to 16 bit left
channel.
                      I2S0_W0_LSW_W = 0;

                      I2S0_W1_MSW_W = 0;
                      I2S0_W1_LSW_W = 0;

                      // Log the data:
                   if ((sample ==0) && (j == 8000) && (i ==
9))
                        logFlag = 1;
                      if (logFlag == 1)
                      {
                          message_bin[2*id] = mes1;
                          message_bin[2*id+1] = mes2;
                          carrier_bin[id] =
sinetable[sample];
                          modulated_sig_bin[id] =
modulated_signal;
                        id = id + 1;

                      // Save the carrier, message and
modulated signal;
                        if (id == 1600)
                        {
                             logFlag = 0;
```

```
    ExportFile("..\\output\\message.bin",
3200,(Int16*) pmes );

    ExportFile("..\\output\\carrier.bin",
1600,(Int16*) pcar );

    ExportFile("..\\output\\modulated.bin",
1600,(Int16*) pmod );
                        }
                      }
                  }
              }
          }
          /* Disable I2S */
          I2S0_CR = 0x00;
```

**Oled.c**

```
    #include "oled.h"

    //Initialize OSD9616
    void OSD9616_init()
    {    Uint8 cmd[10];    // For multibyte commands

        /* Initialize I2C */
        USBSTK5515_I2C_init( );

        /* Initialize LCD power */
        USBSTK5515_GPIO_setDirection( 12, 1 );  // Output
        USBSTK5515_GPIO_setOutput( 12, 1 );     // Enable
13V

        /* Initialize OSD9616 display */
        OSD9616_send(0x00,0x00); // Set low column address
        OSD9616_send(0x00,0x10); // Set high column address
        OSD9616_send(0x00,0x40); // Set start line address

        cmd[0] = 0x00 & 0x00FF;  // Set contrast control
register
        cmd[1] = 0x81;
        cmd[2] = 0x7f;
        USBSTK5515_I2C_write( OSD9616_I2C_ADDR, cmd, 3 );

        OSD9616_send(0x00,0xa1); // Set segment re-map 95
to 0
        OSD9616_send(0x00,0xa6); // Set normal display

        cmd[0] = 0x00 & 0x00FF;  // Set multiplex ratio(1
to 16)
        cmd[1] = 0xa8;
        cmd[2] = 0x0f;
        USBSTK5515_I2C_write( OSD9616_I2C_ADDR, cmd, 3 );

        OSD9616_send(0x00,0xd3); // Set display offset
        OSD9616_send(0x00,0x00); // Not offset
        OSD9616_send(0x00,0xd5); // Set display clock
divide ratio/oscillator frequency
        OSD9616_send(0x00,0xf0); // Set divide ratio

        cmd[0] = 0x00 & 0x00FF;  // Set pre-charge period
```

```
        cmd[1] = 0xd9;
        cmd[2] = 0x22;
        USBSTK5515_I2C_write( OSD9616_I2C_ADDR, cmd, 3 );

        cmd[0] = 0x00 & 0x00FF;  // Set com pins hardware
configuration
        cmd[1] = 0xda;
        cmd[2] = 0x02;
        USBSTK5515_I2C_write( OSD9616_I2C_ADDR, cmd, 3 );

        OSD9616_send(0x00,0xdb); // Set vcomh
        OSD9616_send(0x00,0x49); // 0.83*vref

        cmd[0] = 0x00 & 0x00FF;  //--set DC-DC enable
        cmd[1] = 0x8d;
        cmd[2] = 0x14;
        USBSTK5515_I2C_write( OSD9616_I2C_ADDR, cmd, 3 );

        OSD9616_send(0x00,0xaf); // Turn on oled panel
    }

    //Sends 2 bytes of data to the OSD9616
    Int16 OSD9616_send( Uint16 comdat, Uint16 data )
    {
         Uint8 cmd[2];
         cmd[0] = comdat & 0x00FF; // Specifies whether
data is Command or Data
         cmd[1] = data; // Command / Data
         return USBSTK5515_I2C_write( OSD9616_I2C_ADDR,
cmd, 2 );
    }

    //Sends multiple bytes of data to the OSD9616
    Int16 OSD9616_multiSend( Uint8* data, Uint16 len )
    {
         Uint16 x;
         Uint8 cmd[10];
         for(x=0;x<len;x++) {// Command / Data
              cmd[x] = data[x];
         }
         return USBSTK5515_I2C_write( OSD9616_I2C_ADDR,
cmd, len );
    }
```

```
    // Print letter to OLED:
    Int16 printLetter(Uint16 l1,Uint16 l2,Uint16 l3,Uint16
l4)
    {
         OSD9616_send(0x40,l1);
        OSD9616_send(0x40,l2);
        OSD9616_send(0x40,l3);
        OSD9616_send(0x40,l4);
        OSD9616_send(0x40,0x00);
        return 0;
    }

    // Print the texts to OLED:
    Int16 OSD9616_displayText()
    {
         Int16 i;
         Uint8 cmd[10];    // For multibyte commands

        /* Fill page 0 */
        OSD9616_send(0x00,0x00);   // Set low column
address
        OSD9616_send(0x00,0x10);   // Set high column
address
        OSD9616_send(0x00,0xb0+0); // Set page for page 0
to page 5
        for(i=0;i<128;i++)
        {
         OSD9616_send(0x40,0xff);
        }
        /* Write to page 0 */
        OSD9616_send(0x00,0x00);   // Set low column
address
        OSD9616_send(0x00,0x10);   // Set high column
address
        OSD9616_send(0x00,0xb0+0); // Set page for page 0
to page 5
           for(i=0;i<16;i++)
        {
         OSD9616_send(0x40,0x00);  // Spaces
        }

         printLetter(0x7f,0x02,0x02,0x7f);    //M
         printLetter(0x7e,0x09,0x09,0x7e);    //A
         printLetter(0x5e,0x21,0x21,0x1e);    //Q
```

```
          printLetter(0x7f,0x08,0x08,0x0f);    //4
          printLetter(0x00,0x18,0x18,0x00);    //-
          printLetter(0x00,0x7f,0x02,0x00);    //1
          printLetter(0x41,0x22,0x14,0x7f);    //K
          printLetter(0x7f,0x08,0x08,0x7f);    //H
          printLetter(0x00,0x18,0x18,0x00);    //-
          printLetter(0x31,0x49,0x49,0x4f);    //5
          printLetter(0x00,0x7f,0x02,0x00);    //1
          printLetter(0x06,0x09,0x09,0x7f);    //P
          printLetter(0x3f,0x40,0x20,0x1f);    //V

         for(i=0;i<61;i++)
         {
          OSD9616_send(0x40,0x00);  // Spaces
         }
         /* Fill page 1*/
         OSD9616_send(0x00,0x00);   // Set low column
address
         OSD9616_send(0x00,0x10);   // Set high column
address
         OSD9616_send(0x00,0xb0+1); // Set page for page 0
to page 5
         for(i=0;i<170;i++)
         {
          OSD9616_send(0x40,0xff);
         }

         /* Write to page 1*/
         OSD9616_send(0x00,0x00);   // Set low column
address
         OSD9616_send(0x00,0x10);   // Set high column
address
         OSD9616_send(0x00,0xb0+1); // Set page for page 0
to page 5

          printLetter(0x3f,0x49,0x49,0x26);    //g
          printLetter(0x40,0x78,0x08,0x78);    //n
          printLetter(0x20,0x78,0x4a,0x79);    //af
          printLetter(0x78,0x48,0x48,0x78);    //o
          printLetter(0x7f,0x08,0x08,0x7f);    //H
          printLetter(0x00,0x18,0x18,0x00);    //-
          printLetter(0x78,0x08,0x08,0x7f);    //h
          printLetter(0x40,0x78,0x08,0x78);    //n
          printLetter(0x00,0x7a,0x00,0x00);    //i
```

```
          printLetter(0x7f,0x02,0x02,0x7f);    //M
          printLetter(0x00,0x18,0x18,0x00);    //-
          printLetter(0x40,0x78,0x08,0x78);    //n
          printLetter(0x22,0x2d,0x7d,0x3a);    //eej
          printLetter(0x00,0x7a,0x00,0x00);    //i
          printLetter(0x78,0x08,0x08,0x7f);    //h
          printLetter(0x01,0x07f,0x7f,0x01);   //T
          printLetter(0x00,0x18,0x18,0x00);    //-
          printLetter(0x0f,0x79,0x01,0x01);    //7
          printLetter(0x78,0x10,0x10,0x78);    //m
          printLetter(0x79,0x4a,0x48,0x78);    //os
          printLetter(0x78,0x08,0x08,0x7f);    //h
          printLetter(0x7f,0x20,0x04,0x7f);    //N

         for(i=0;i<19;i++)
         {
         OSD9616_send(0x40,0x00);  // Spaces
         }

         /* Set vertical and horizontal scrolling */
         cmd[0] = 0x00;
         cmd[1] = 0x29;  // Vertical and Right Horizontal
Scroll
         cmd[2] = 0x00;  // Dummy byte
         cmd[3] = 0x00;  // Define start page address
         cmd[4] = 0x03;  // Set time interval between each
scroll step
         cmd[5] = 0x01;  // Define end page address
         cmd[6] = 0x08;  // Vertical scrolling offset
         OSD9616_multiSend( cmd, 7 );
         OSD9616_send(0x00,0x2f);
         /* Keep first 8 rows from vertical scrolling  */
         cmd[0] = 0x00;
         cmd[1] = 0xa3;  // Set Vertical Scroll Area
         cmd[2] = 0x08;  // Set No. of rows in top fixed
area
         cmd[3] = 0x08;  // Set No. of rows in scroll area
         OSD9616_multiSend( cmd, 4 );

          return 0;
     }
```

**Utilities.c**

```
#include "utilities.h"

// Get .bin file length:
Uint32 GetBinLen(const char *path)
{
     Uint32 length;
//   char c;
     FILE *fp;
     /* Open file for both reading and writing */
     fp = fopen(path, "rb");
     /* Seek to the beginning of the file */
     fseek(fp, SEEK_SET, 0);

     length=0;
     /*get the length of the text*/
     while (1){
          fgetc(fp);
          if(feof(fp)){
               break;
          }
          length++;
     }
     fclose(fp);
     return length;
}

//Save file to .bin file
Uint16 ExportFile(const char *path, Uint32 binLen, Int16
*p_buffer_data )
{
     FILE *fp ;
     Uint32 i ;
     Int8 temp;

     fp = fopen(path,"wb") ;
     if ( fp == (FILE*) NULL)
          {
               return 1 ;
          }
     for (i = 0; i < binLen; i++ )
          {
               p_buffer_data[i] = p_buffer_data[i] & 0xFFFF;
```

```
                temp = p_buffer_data[i] >> 8;
                fputc(temp, fp);
                temp = p_buffer_data[i] & 0xFF;
                fputc(temp, fp);
          }
     fclose(fp) ;
     return 0 ;
}

// Load data from .bin file:
Uint16 ImportFile(const char *path,  Uint32 binLen, Int16
*p_buffer_data )
{
     FILE *fp ;
     Int16 data , pdata[1] ;
     Uint16 i ;


     fp = fopen(path,"rb") ;
     if ( fp == (FILE*)NULL)
          {
          return 1 ;
          }

     for( i = 0 ; i < binLen; i++ )
{
          fread(pdata, 1, 1, fp);
          data = *(pdata);
          p_buffer_data[i] = data ;
}
     fclose(fp) ;
     return 0;
}
```

**Sinetable.h**

```
#ifndef SINETABLE_H_
#define SINETABLE_H_
    /* Pre-generated sine and cosine wave data
     * 16-bit signed samples, 4 samples/period, f_carrier =
6000 kHz */
    static Int8 samplesPerPeriod = 4;
    static Int16 sinetable[4]   = {
              0,       16384,
              0,    -16384 };
    static Int16 cosinetable[4] = {
          16384,           0,
         -16384,           0 };


#endif /*SINETABLE_H_*/
```

Ngoài ra, nhóm còn sử dụng bộ thư viện hỗ trợ từ Texas Instrument để cấu hình phần cứng usbstk5515bsl.lib.