TÓM TẮT NỘI DUNG KHÓA LUẬN

Mục tiêu chính của khóa luận là thực thi thuật toán MUSIC trên kit DSP TMS320C6713 của Texas Instrument, với các cấu trúc anten mảng khác nhau như ULA, UCA... Vì vậy trọng tâm của khóa luận là mô phỏng thuật toán MUSIC với các cấu trúc mảng khác nhau, từ đó tìm các giải thuật để triển khai thuật toán trên kit DSP.

Chi tiết khóa luận bao gồm 5 chương:

Chương 1: Thực hiện thuật toán MUSIC với các cấu trúc anten mảng ULA, UCA: thực hiện việc mô phỏng thuật toán sử dụng chương trình Matlab, từ đó đánh giá ưu, nhược điểm của thuật toán với các cấu trúc anten khác nhau.

Chương 2: Giới thiệu cơ bản về kit DSP TMS320C6713. Chương này phân tích các đặc điểm của kit, từ đó nêu lý do sử dụng kit để thực thi thuật toán MUSIC. Tiếp đó giới thiệu môi trường giao tiếp với kit là chương trình Code Composer Studio, và nghiên cứu khả năng tích hợp giữa Code Composer Studio và Matlab qua tính năng "Real Time Data Exchange".

Chương 3: Thực thi thuật toán MUSIC trên kit TMS320C6713: nêu lên các công việc thực hiện được và các kết quả đạt được trong thực tế.

Cuối cùng là phần kết luận và hướng phát triển tiếp theo cho khóa luận.

LỜI CAM ĐOAN

Tôi xin cam đoan khóa luận tốt nghiệp: "Thực thi thuật toán MUSIC trên kit DSPTMS320C6713" là công trình nghiên cứu của bản thân. Những phần sử dụng tài liệu tham khảo trong khóa luận đã được nêu rõ trong phần tài liệu tham khảo. Các số liệu, kết quả trình bày trong khóa luận là hoàn toàn trung thực, nếu sai tôi xin chịu hoàn toàn trách nhiệm và chịu mọi kỷ luật của khoa và nhà trường đề ra.

Tác giả khóa luận

Nguyễn Đức Anh

LỜI CẢM ƠN

Để có thể hoàn thành tốt khóa luận này, em xin chân thành gửi lời cảm ơn tới Ths.Trần Thị Thúy Quỳnh, người đã hướng dẫn tận tình và giúp đỡ em rất nhiều trong quá trình thực hiện bài khóa luận của mình.

Ngoài ra, trong quá trình thực hiện khóa luận em còn nhận được rất nhiều sự động viên và giúp đỡ từ phía gia đình, người thân và tập thể các bạn trong lớp. Do đó kết quả cũng như tính khả dụng của bài luận văn này trong thực tế là lời cảm ơn sâu sắc nhất của em gửi tới mọi người và là nguồn động lực để em có thể tự tin vào các kiến thức mình đã thu được sau khi tốt nghiệp.

MỤC LỤC

DANH MỤC CÁC CHỮ VIẾT TẮT

Thuật ngữ	Tiếng anh	Tiếng Việt
DOA	Direction of Arrival	Hướng sóng tới
MUSIC	MUltiple Signal Classification	Phương pháp phân lớp đa tín hiệu
ESPRIT	Estimation of Signal Parameters via Rotational Invariance Techniques	Phương pháp đánh giá các thông số tín hiệu thông qua kỹ thuật quay bất biến
SNR	Signal to Noise Ratio	Tỷ số tín hiệu trên nhiễu
ULA	Uniform Linear Array	Hệ anten sắp xếp theo đường thẳng
UCA	Uniform Circular Array	Hệ anten sắp xếp theo đường tròn
DSP	Digital Signal Proccessor	Bộ xử lý tín hiệu số
FPGA	Field-programmable gate array	Vi mạch cấu trúc mảng phần tử khả trình
RTDX	Real Time Data Exchange	Trao đổi dữ liệu thời gian thực
API	Application programming interface	Giao diện lập trình ứng dụng
JTAG	Joint Test Action Group	
CCS	Code Composer Studio	

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH VẼ

MỞ ĐẦU

Anten thông minh là một công nghệ mới được quan tâm nhiều trong thời gian gần đây với các ưu điểm: cải thiện vùng phủ sóng, giảm nhiễu, tăng dung lượng, mở rộng phạm vi của hệ thống...[1].

Nói một cách đơn giản, anten thông minh gồm nhiều phần tử anten đơn giản kết hợp với bộ xử lý tín hiệu để có thể thay đổi búp sóng một cách tự động. Các cấu trúc anten mảng phổ biến là cấu trúc dạng đường thẳng cách đều (ULA), cấu trúc dạng chữ nhật cách đều (URA) hay cấu trúc dạng đường tròn cách đều (UCA).

Trong các cấu trúc anten thông minh, việc xử lý tín hiệu được thực thi bằng cách thay đổi các trọng số tại mỗi phần tử anten theo một tham số tối ưu xác định. Việc xác định sơ bộ hướng sóng đến (DOA) là một trong số các trọng số này. MUSIC là một trong những thuật toán xác định DOA với nhiều ưu điểm như độ chính xác cao, khả năng áp dụng với nhiều cấu trúc anten mảng. Vì vậy thuật toán MUSIC đang ngày càng phát triển và được ứng dụng rộng rãi trong các hệ thống quân sự, điều khiển, thông tin liên lạc, truyền thông…

Để phục vụ cho việc đo lường, lưu trữ và xử lý các tín hiệu thu thập từ anten trong thực tế, thuật toán MUSIC được thực thi trên các bộ xử lý số DSP. Các bộ xử lý số DSP được lựa chọn bởi khả năng xử lý tín hiệu số rất mạnh cùng khả năng tái lập trình đơn giản. Khóa luận tốt nghiệp thực hiện việc thực thi thuật toán MUSIC trên kit DSP TMS320C6713 nhằm mục đích tiến tới thực tiễn hóa hệ thống xác định hướng sóng tới.

CHƯƠNG 1

LÝ THUYẾT VÀ MÔ HÌNH THUẬT TOÁN MUSIC

1.1 Giới thiệu về thuật toán MUSIC

MUSIC là một thuật toán xác định hướng sóng tới (DOA-Direction Of Arrival) dựa vào những tín hiệu thu thập được từ mảng anten. Trên thế giới việc nghiên cứu phương pháp tìm hướng sóng bắt đầu từ những năm 70 của thế kỷ trước, và rất nhiều thuật toán được tìm ra và tiếp tục được nghiên cứu phát triển đến tận bây giờ. Lịch sử phát triển các thuật toán có thể được trình bày tóm tắt qua bảng sau [5]:

Bảng 1: Lịch sử phát triển của thuật toán tìm hướng sóng đến

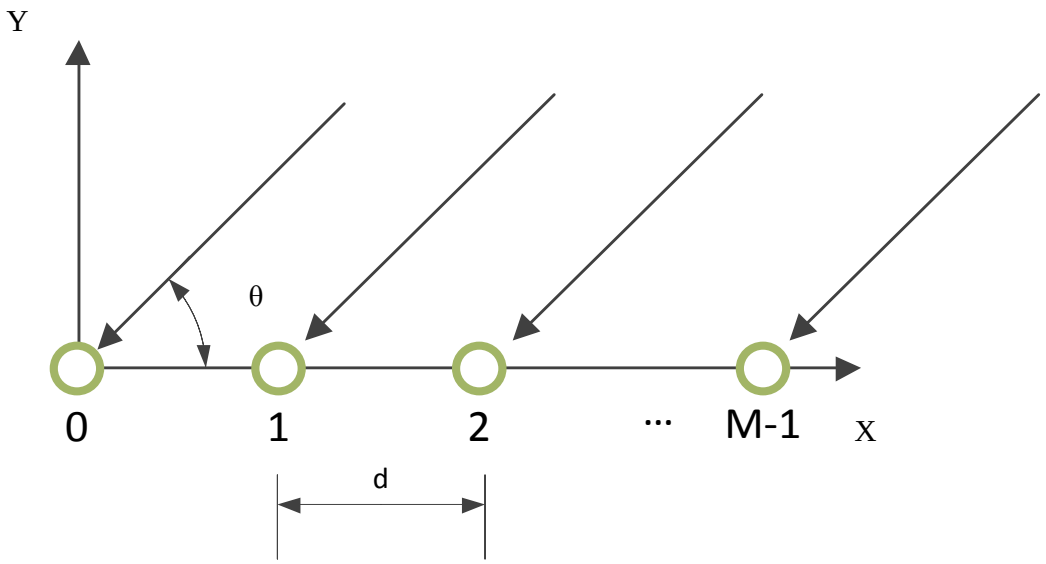
Năm tìm ra	**Tên thuật toán**	**Tác giả thuật toán**
1967	Maximum Entropy Method	Burg
1969	Maximum Likehood Method	Capon
1973	Covariance Method	Pisarenco
1977	Subspace Method –MUSIC	Schmidt
1979	Subspace Method –MUSIC	Bienvenu
1989	Subspace Method –ESPRIT	Richchard Roy,Thomas Kailath

Từ bảng ta có thể thấy: các phương pháp tìm hướng sóng đến hiện nay cơ bản dựa trên hai phương pháp không gian con: phương pháp phân lớp đa tín hiệu (MUltiple Signal Classification – MUSIC) và phương pháp đánh giá các thông số tín hiệu thông qua kỹ thuật quay bất biến (Estimation Of Signal Parameters Via Rotational Invariance Techniques – ESPRIT). Phương pháp ESPRIT ra đời sau nên tốc độ thực hiện nhanh hơn MUSIC, tuy nhiên MUSIC lại có ưu điểm là đơn giản, linh hoạt hơn trong việc áp dụng cho các cấu trúc anten khác nhau nên được nghiên cứu và ứng dụng thực tế nhiều hơn. Được tìm ra bởi những nghiên cứu độc lập của Schmidt (1977) và Bienvenu (1979), thuật toán này có thể xác định số lượng tín hiệu đến anten cũng như hướng của các tín hiệu đó, chỉ dựa trên vectơ tín hiệu đầu vào.

1.2 Thuật toán MUSIC đối với dàn anten ULA

1.2.1 Mô hình dàn anten ULA

Anten ULA hay dàn chấn tử đồng pha ULA là loại anten mảng đơn giản nhất. Dạng hình học của dàn anten này được biểu diễn qua hình vẽ sau:



Hình 1: Mô hình cấu trúc anten ULA

Cấu trúc của dàn anten ULA bao gồm M chấn tử đặt song song trên cùng một trục thẳng với cùng khoảng cách d. Mỗi phần tử trong hệ anten đóng vai trò là một nguồn đẳng hướng. Các chấn tử này hoạt động cùng pha với nhau tạo nên một hướng bức xạ duy nhất và để cho tín hiệu tới bộ xử lí giữ nguyên được pha và biên độ so với tín hiệu tới hệ anten.

1.2.2 Tín hiệu thu được sau khi qua dàn anten ULA

Như vậy, với một nguồn sóng đến, tín hiệu nhận được tại phần tử 0 (phần tử tham chiếu) được biểu diễn dưới dạng:

$$u_0(t) = a(t)\cos(2\Pi f_c t + \Psi(t) + \alpha) \qquad (1.1)$$

với: a(t): biên độ của tín hiệu

f_c : tần số sóng mang

$\psi(t)$: thành phần mang tin

α: pha của tín hiệu.

Giả sử rằng khoảng cách từ anten tới nguồn tín hiệu r >> d. Do vậy tia sóng ở

phần tử thứ i sẽ song song với tia sóng ở phần tử tham chiếu, nhưng bị trễ đi 1 khoảng thời gian truyền dẫn:

$$t = i \times \tau \tag{1.2}$$

với τ là thời gian trễ truyền dẫn giữa hai phần tử kế tiếp.

Trong dàn anten ULA, τ có giá trị bằng:

$$\tau = \frac{d \times \cos(\theta)}{c} \tag{1.3}$$

Ở đây θ: góc tới của tia sóng xuất phát từ nguồn

c: vận tốc truyền ánh sáng $c = 3.10^8$ m/s

Từ công thức (1.1), (1.2), (1.3) ta có tín hiệu nhận được ở phần tử anten thứ nhất:

$$u_1(t) = a(t)\cos(2\Pi f_c t - 2\Pi f_c \tau + \Psi(t) + \alpha) \tag{1.4}$$

Sử dụng đồng nhất thức Euler, đồng thời loại bỏ thành phần sóng mang của tín hiệu ta có dạng tín hiệu xét ở phần tử thứ nhất là:

$$\begin{aligned} u_1(t) &= a(t)e^{j(-2\Pi f_c \frac{d\cos\theta}{c} + \Psi(t) + \alpha)} \\ &= a(t)e^{j(\beta d\cos\theta + \Psi(t) + \alpha)} \end{aligned} \tag{1.5}$$

với: $\beta = \frac{2 \times \Pi}{\lambda}$ gọi là hệ số truyền sóng

Do đó với phần tử thứ k của hệ ta có dạng tín hiệu nhận được:

$$u_k(t) = a(t)e^{j(\beta kd\cos\theta + \Psi(t) + \alpha)} \tag{1.6}$$

So sánh với dạng tín hiệu nhận được ở phần tử tham chiếu:

$$s(t) = u_0(t) = a(t)e^{j(\Psi(t) + \alpha)} \tag{1.7}$$

Ta được dạng rút gọn của tín hiệu nhận được tại phần tử thứ k:

$$u_k(t) = s(t)e^{j\beta kd\cos\theta} \tag{1.8}$$

Tổng quát với D nguồn tín hiệu độc lập đến dàn anten, tín hiệu lấy mẫu tại phần tử anten thứ k lúc này sẽ là:

$$u_k(nT) = \sum_{i=1}^{D} s_i(nT) \times e^{j\beta kd\cos\theta_i} + n_k(nT) \tag{1.9}$$

Trong đó: $u_k(nT)$ là mẫu tín hiệu thu được tại phần tử thứ k tại thời điểm n.

$s_i(nT)$ là mẫu tín hiệu tới dàn anten (mẫu tín hiệu thu được tại phần tử tham chiếu) tại thời điểm n.

$n_k(nT)$ là mẫu tín hiệu nhiễu tại phần tử thứ k tại thời điểm n.

Do vậy ta có biểu diễn tổng quát của cả hệ như sau:

$$u(t) = \begin{bmatrix} u_0(t) \\ u_1(t) \\ \cdots \\ u_{M-1}(t) \end{bmatrix} = \sum_{i=1}^{D} A(\theta_i) s_i(t) + n(t) \tag{1.10}$$

Ở đây: $A(\theta)$ được gọi là vectơ lái của dàn anten.

$$A(\theta) = \begin{bmatrix} 1 \\ e^{-j\beta d\cos\theta} \\ \cdots \\ e^{-j\beta(M-1)d\cos\theta} \end{bmatrix} \tag{1.11}$$

$s_i(t)$ là các tín hiệu tới dàn anten với $i = 1 \div D$

$n(t)$ là tín hiệu nhiễu M chiều.

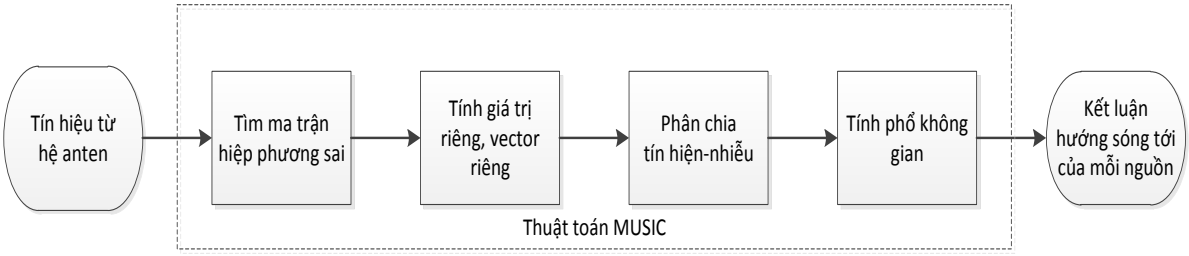
Công thức thu gọn của (1.10):

$$U = A \times S + N \tag{1.12}$$

Đây cũng là công thức thể hiện tín hiệu thu nhận được sau khi qua dàn anten.

1.2.3 Thuật toán MUSIC với dàn anten ULA

Thuật toán MUSIC thực hiện việc tìm hướng sóng tới DOA dựa trên các mẫu thu thập được từ anten [6]. Thuật toán gồm 4 bước chính, được thể hiện qua lưu đồ sau:



Hình 2: Lưu đồ thuật toán MUSIC

Nội dung của từng bước:

1. ***Tìm ma trận hiệp phương sai của các mẫu tín hiệu thu được***

Giả thiết các nguồn tín hiệu tới hệ anten không tương quan với ồn, nghĩa là:

$$E\left[s_k(t)n_l(t)\right]=0 \ \forall k \tag{1.13}$$

Khi đó ma trận hiệp biến tín hiệu lối vào sẽ là:

$$R_{UU}=E\left[UU^H\right]=AE\left[SS^H\right]A^H+E\left[NN^H\right] \tag{1.14}$$

$$\text{Vì:} \quad E\left[n_k(t)n_l(t)\right]=\begin{cases}0 & l\neq k\\ \sigma^2_{noise} & l=k\end{cases} \tag{1.15}$$

Nên (1.14) trở thành:

$$R_{UU}=E\left[UU^H\right]=AR_{SS}A^H+\sigma^2_{noise}I \tag{1.16}$$

Trong đó: R_{SS} là ma trận tương quan của các nguồn tín hiệu ban đầu.

Giả thiết các nguồn tín hiệu không tương quan với nhau, ta có đẳng thức:

$$R_{SS}=E\left[s_k(t)s_l(t)\right]=\begin{cases}0 & l\neq k\\ p_i & l=k\end{cases} \tag{1.17}$$

với p_i là công suất của tín hiệu thứ *i*.

2. ***Tính giá trị riêng, vectơ riêng của ma trận hiệp phương sai***

Nếu gọi $\{\lambda_0,\lambda_1,\ldots,\lambda_{N-1}\}$ là các giá trị riêng của ma trận tự tương quan R_{uu} thì:

$$\left|R_{UU}-\lambda_i I\right|=0 \tag{1.18}$$

Phương trình (1.10) có thể viết lại thành:

$$\left|AR_{SS}A^H+\sigma^2_{noise}I-\lambda_i I\right|=\left|AR_{SS}A^H+\left(\sigma^2_{noise}-\lambda_i\right)I\right|=0 \tag{1.19}$$

Do đó $\left(\sigma_n^2-\lambda_i\right)$ chính là các giá trị riêng của ma trận $AR_{SS}A^H$. Do ma trận $AR_{SS}A^H$ là đối xứng, các phần tử là các số phức dương, nên các giá trị riêng của nó là thực và dương.

Vì trong không gian tín hiệu chỉ có D nguồn sóng nên chỉ có D giá trị riêng tương ứng với nguồn, N – D giá trị riêng còn lại của ma trận tương ứng với nhiễu sẽ

bằng không. Trên thực tế, do ảnh hưởng của nhiễu nên khó xác định N – D giá trị riêng ứng với nhiễu. Tuy nhiên, khi xem xét giá trị riêng của ma trận $AR_{SS}A^H$ ta thấy rằng có thể phân N giá trị này thành hai không gian con [3]:

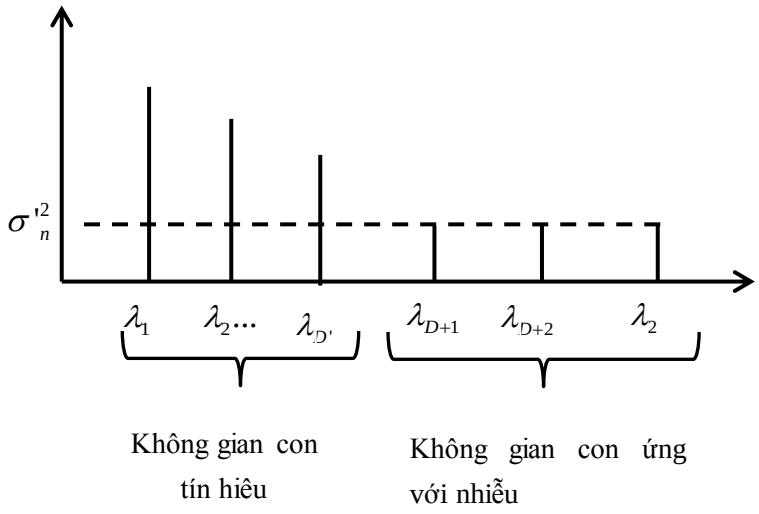
- D giá trị riêng đầu tiên hợp thành không gian con tín hiệu, được sắp xếp theo thứ tự giảm dần:

$$\lambda_1 > \lambda_2 > ... > \lambda_D \tag{1.20}$$

- N – D giá trị riêng còn lại hợp thành không gian con tương ứng với nhiễu, có cùng mức:

$$\lambda_D, \lambda_{D+1}, \ldots, \lambda_{N-1} = \sigma_n^2 \tag{1.21}$$

Sự phân bố các giá trị riêng của ma trận hiệp phương sai được thể hiện ở hình dưới:



Hình 3: Dạng phân bố các giá trị riêng của ma trận tương quan

3. ***Từ các giá trị riêng tìm được, phân tách không gian tín hiệu và không gian nhiễu***

Phương pháp xử lý ở đây là đưa ra tìm giới hạn $\sigma'^2_n > \sigma_n^2$ sao cho từ tập N giá tri riêng có một số K các giá trị riêng nhỏ nhất và số tín hiệu tới hệ anten sẽ là:

$$D' = N - K \tag{1.22}$$

Ứng với mỗi giá trị riêng sẽ tìm được một vectơ *qi* (*i= D...N-1*) thỏa mãn:

$$\left(R_{UU} - \lambda_i I\right) q_i = 0 \tag{1.23}$$

Từ (1.16), (1.21) và (1.23) ta có:

$$\left(R_{UU} - \sigma_n^2 I\right) q_i = A\mathrm{R}_{\mathrm{SS}} A^H q_i + \sigma_n^2 I - \sigma_n^2 I = 0 \quad (1.24)$$

Suy ra:

$$AR_{SS} A^H q_i = 0 \quad (1.25)$$

Trong điều kiện tích $AR_{SS} \neq 0$ (*) (A là ma trận đủ hạng và R_{SS} không phải là ma trận kì dị) thì:

$$\mathrm{A}^{\mathrm{H}} q_i = 0 \quad (1.26)$$

Hay:
$$\begin{bmatrix} a^H(\theta_0) q_i \\ a^H(\theta_1) q_i \\ \dots \\ a^H(\theta_{M-1}) q_i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} \quad (1.27)$$

Điều này có nghĩa là các vectơ riêng của $\mathrm{AR}_{\mathrm{SS}}\mathrm{A}^{\mathrm{H}}$ cũng được chia thành hai không gian: các vectơ riêng ứng với tín hiệu thì cùng phương với các vectơ lái, còn những vectơ riêng còn lại tương ứng với nhiễu thì trực giao với vectơ hướng sóng đến, hay:

$$\left\{a(\theta_0), ..., a(\theta_{D-1})\right\} \perp \left\{q_D, ..., q_{M-1}\right\} \quad (1.28)$$

4. ***Tính phổ không gian***

Thuật toán MUSIC tính phổ không gian của tín hiệu theo công thức:

$$P(\theta) = \frac{a^H(\theta) a(\theta)}{a^H(\theta) V_n V_n^H a(\theta)} \quad (1.29)$$

với $V_n = \left\{q_D, ..., q_{M-1}\right\}$ là các vectơ riêng của không gian nhiễu.

Khi thay đổi góc θ trùng với góc tới hệ anten của tín hiệu, do các vectơ lái luôn trực giao với các vectơ riêng của không gian nhiễu nên mẫu số của (1.25) sẽ tiến tới không và phổ không gian của tín hiệu sẽ đạt cực đại.

Như vậy, các điểm cực đại trên đồ thị biểu diễn $P(\theta)$ cho phép ta xác định được hướng sóng tới.

1.2.4 Mô phỏng thuật toán MUSIC với dàn anten ULA trên Matlab

- Mô hình hệ mô phỏng

Thuật toán được mô phỏng với dàn anten ULA với các thông số của nguồn cũng như của hệ anten được cho trong bảng:

Nhận xét: Kết quả mô phỏng thuật toán MUSIC với hệ anten ULA xác định được 8 nguồn sóng tới tại các góc lần lượt là [20^0, 40^0, 60^0, 80^0, 280^0, 300^0, 320^0, 340^0].

So với thông số các nguồn sóng đến được mô phỏng, kết quả mô phỏng thuật toán cho ta giá trị chính xác 4 góc tới là [20^0, 40^0, 60^0, 80^0]. Ngoài ra trong phổ không gian còn xuất hiện thêm 4 góc tới khác ở các hướng [280^0, 300^0, 320^0, 340^0].

Nguyên nhân của hiện tượng này là do phương trình biểu diễn vectơ lái hay do cấu trúc hình học của hệ anten. Từ công thức tính vectơ lái của mảng anten (1.8) ta có:

$$A(\theta_i) = e^{-j\beta d\cos(\theta_i)} \tag{1.30}$$

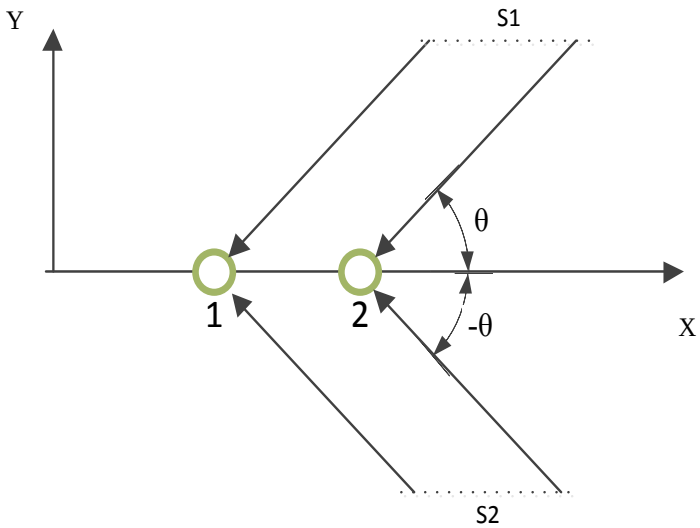
Trong cấu trúc dàn anten ULA, vì:

$$\cos(\theta_i) = \cos(-\theta_i) \forall \theta_i \tag{1.31}$$

nên

$$A(\theta_i) = A(-\theta_i) \tag{1.32}$$

Lại do cấu trúc dàn anten ULA có các phần tử sắp xếp cách đều theo đường thẳng nên vectơ lái của mảng với các góc θ và (–θ) là như nhau, dẫn đến kết quả phổ không gian đánh giá DOA ở góc θ cũng giống với góc (–θ). Vấn đề này được thể hiện rõ ràng hơn ở hình sau:



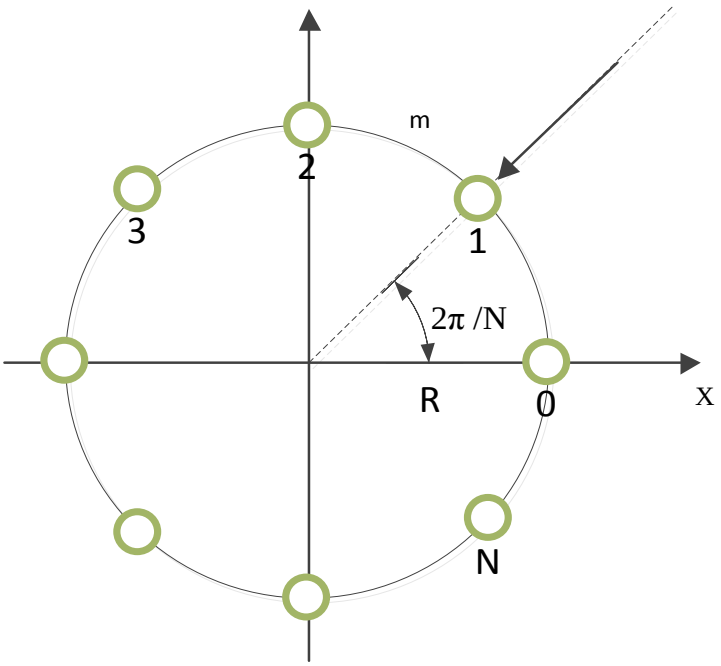
Hình 4: Hệ anten ULA trong trường hợp hai tín hiệu đối xứng

Ví dụ như ở hình trên, hệ anten không thể nhận ra được nguồn âm ở phía trên hay phía dưới của hệ. Điều này dẫn đến việc, khi xây dựng một hệ ULA cần phải thiết kế vị trí đặt hệ thích hợp để có thể che khuất nửa mặt phẳng mà hệ không phân biệt được, hoặc phải kết hợp các hệ ULA lại sao cho có thể tìm được vị trí tín hiệu cần nhận biết.

1.3 Thuật toán MUSIC đối với dàn anten UCA

1.3.1 Mô hình dàn anten UCA

Khác với cấu trúc anten ULA, cấu trúc anten sắp xếp theo đường tròn cách đều UCA được nghiên cứu từ lâu, nhưng phải tới những năm 1960 khi mà khái niệm về sự kích thích chế độ pha bắt đầu phát triển thì các nghiên cứu về hệ anten này mới đạt được những thành tựu đột phá [4]. Mô hình dàn anten UCA được thể hiện ở hình dưới:



Hình 5: Mô hình cấu trúc anten UCA

Cấu tạo của dàn anten gồm N phần tử anten đặt trong không gian thành hình tròn có bán kính R = Nmλ/2π trong mặt phẳng XY, trong đó m là độ dài cung giữa hai phần tử kề nhau, bước sóng λ.

1.3.2 Tín hiệu thu được sau khi qua dàn anten ULA

So với hệ anten mảng thẳng ULA, hệ anten mảng tròn chỉ khác cách bố trí anten trong không gian. Do vậy dạng tín hiệu nhận được tại mỗi anten riêng lẻ cũng như cấu trúc tín hiệu ở đầu vảo hệ anten là giống nhau, chỉ khác biệt ở thành phần

vectơ lái:

Vectơ lái của dàn anten ULA:

$$A(\theta_i)_{ULA} = \begin{bmatrix} 1 \\ e^{-j\beta d\cos\theta_i} \\ \cdots \\ e^{-j\beta(M-1)d\cos\theta_i} \end{bmatrix} \quad (1.33)$$

Đối với dàn anten UCA, phân tích tương tự như với dàn anten ULA ta cũng có được vectơ lái của dàn là:

$$A(\theta_i)_{UCA} = \begin{bmatrix} 1 \\ e^{j\beta R\cos(\theta_i-\Psi_1)} \\ e^{j\beta R\cos(\theta_i-\Psi_2)} \\ \cdots \\ e^{j\beta R\cos(\theta_i-\Psi_N)} \end{bmatrix} \quad (1.34)$$

Và dạng tín hiệu đầu vảo của dàn anten UCA là:

$$u(t) = \begin{bmatrix} u_0(t) \\ u_1(t) \\ \cdots \\ u_{M-1}(t) \end{bmatrix} = \sum_{i=1}^{D} A(\theta_i)s_i(t) + n(t) \quad (1.35)$$

1.3.3 Thuật toán MUSIC với dàn anten UCA

Thuật toán MUSIC trong dàn anten UCA cũng được áp dụng giống như trong dàn anten ULA.

1.3.4 Mô phỏng thuật toán MUSIC với dàn anten UCA

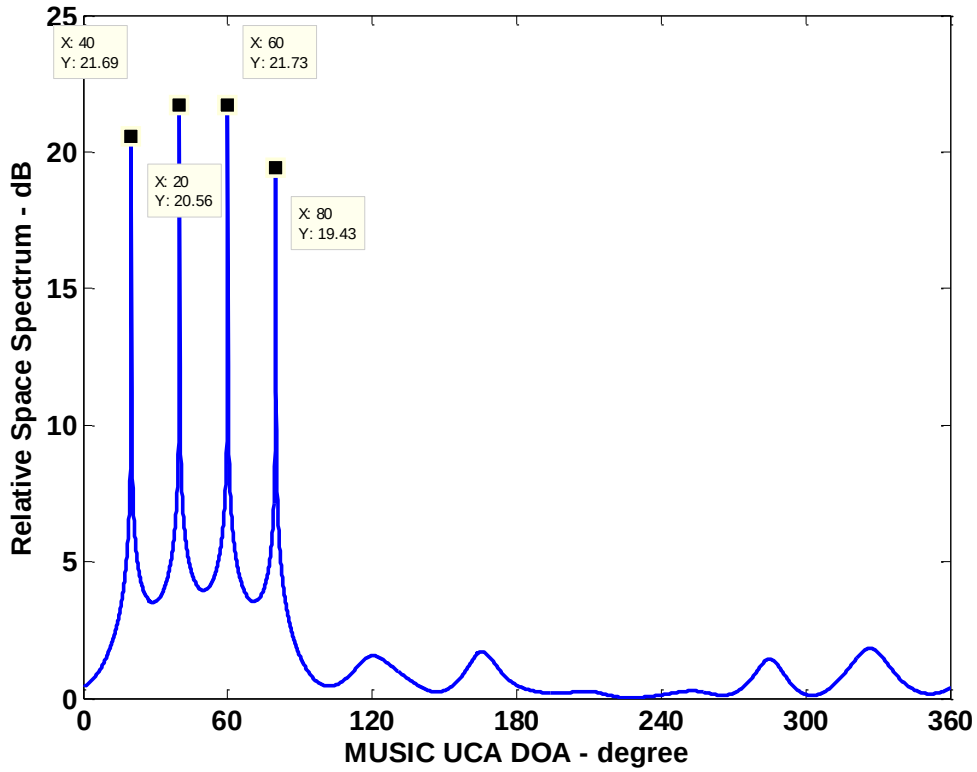
Nhằm so sánh, đánh giá hiệu quả thuật toán MUSIC với các cấu trúc dàn anten ULA và UCA, việc mô phỏng được thực hiện với cùng thông số về môi trường và nguồn sóng với dàn anten ULA.

- Mô hình hệ mô phỏng

Bảng 2: Thông số mô phỏng dàn anten

STT	Các thông số của nguồn	Các thông số của hệ anten
1	Số nguồn tín hiệu: 4	Số phần tử anten: 8
2	Góc tới của các nguồn tín hiệu: [20^o 40^o 60^o 80^o]	Độ dài cung tròn: m= λ/2 (m)
3	Tỷ số SNR: 25 dB	

- Kết quả mô phỏng



Hình 6:Kết quả mô phỏng hệ anten UCA

Nhận xét: kết quả mô phỏng hệ anten UCA thể hiện 4 đỉnh (với hệ ULA là 8 đỉnh) tương ứng với 4 nguồn sóng tới. Có thể phân biệt dễ dàng các đỉnh. Như vậy dàn anten UCA đã khắc phục nhược điểm của hệ ULA: phổ không gian thu được không xuất hiện các đỉnh phụ.

CHƯƠNG 2

TỔNG QUAN VỀ KIT DSP TMS320C6713

2.1 Lựa chọn phần cứng

Hiện nay có nhiều giải pháp xử lý tín hiệu số cho các anten thông minh, mà nổi bật nhất là sử dụng chip xử lý tín hiệu số - Digital Signal Processor (DSP) hoặc dùng các vi mạch mảng phần tử logic khả trình (FPGA), FPGA là vi mạch thuộc họ vi mạch tích hợp chuyên dụng (ASIC) lập trình được, sử dụng các ngôn ngữ đặc tả phần cứng để thiết kế những cấu trúc được tối ưu hóa cho những ứng dụng cụ thể. Ưu việt của FPGA là không thể bàn cãi, đặc biệt là khả năng xử lý nhiều tập lệnh cùng lúc cho tốc độ cao, và khả năng tiêu thụ ít điện năng hơn chip DSP. Tuy nhiên việc thực thi thuật toán trên kit DSP cũng có những đặc điểm nổi bật:

- DSP có khả năng thực hiện đa tác vụ từ điều khiển đến xử lý tín hiệu, với giá thành rẻ hơn so với FPGA.
- Để đạt được hiệu suất tối đa cho FPGA cần nhiều thời gian và kiến thức để tối ưu, trong khi đó tốc độ xử lý của kit DSP chỉ phụ thuộc chủ yếu vào xung nhịp của chip, do đó có thể đạt được hiệu suất cao hơn trong thời gian ngắn
- DSP sử dụng ngôn ngữ lập trình C, ASM tương đối phổ dụng, không đòi hỏi hiểu biết ngôn ngữ mô phỏng phần cứng như FPGA, khi cần thay đổi, lập trình lại, chip DSP cũng tỏ ra mềm dẻo hơn do chỉ cần chỉnh sửa code, trong khi đó với FPGA gặp khó khăn hơn do phải tái cấu trúc lại các cổng logic.

Dựa trên những phân tích trên, cùng với thực tế quá trình làm khóa luận trong thời gian ngắn, tập trung vào mục tiêu nghiên cứu, không đòi hỏi tối ưu điện năng tiêu thụ, ta chọn giải pháp thực thi trên chip DSP, cụ thể là kit DSP TMS320C6713của Texas Instrument.

2.2 Giới thiệu chung về kit

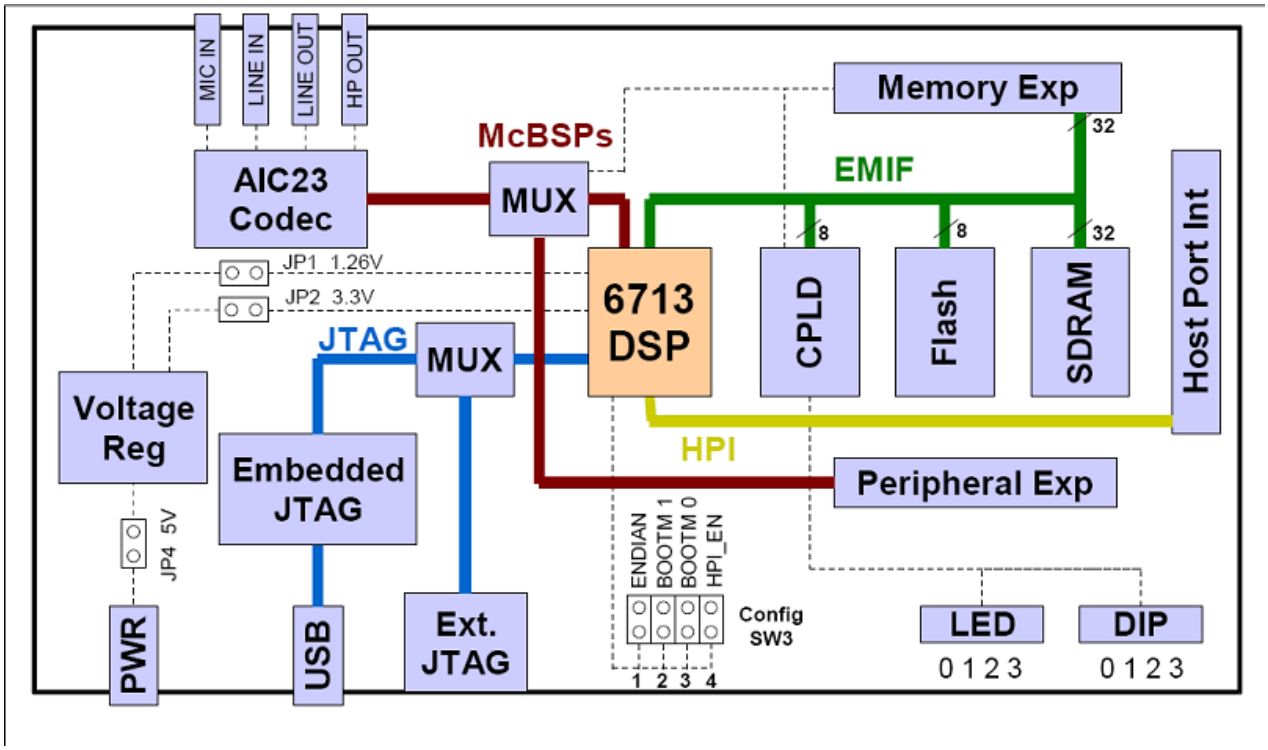
Kit TMS320C6713 DSP là giải pháp tất cả trong một cho việc lập trình trên nền DSP, cụ thể ở đây là lập trình trên chip TMS320C6713 của Texas Instrument. Các thành phần của kit bao gồm: bảng mạch sử dụng thiết kế chuẩn cho chip C6713 của TI, đĩa phần mềm chứa driver và phần mềm Code Composer Studio (CCS) để lập trình và giao tiếp với chip DSP, ngoài ra còn có sách giới thiệu, cáp USB và một adapter dùng để cấp nguồn 5V cho mạch. Hình ảnh tổng quan về các thành phần này được thể hiện ở hình dưới:

Hình 7: Kit TMS320C6713

Trong chương này sẽ đề cập đến tính năng và vai trò của hai thành phần quan trọng nhất trong kit: bảng mạch và phần mềm CCS, ngoài ra chương này cũng đề cập đến việc tích hợp giữa CCS và môi trường Matlab thông qua tính năng Real Time Data Exchange.

2.3 Bảng mạch DSP

Bảng mạch DSP của kit có thể coi như 1 hệ thống DSP hoàn chỉnh cho việc xử lý tín hiệu. Nó có tất cả các cổng kết nối để giao tiếp với máy tính qua cáp USB, khối ADC với 4 đường vào ra để nhận tín hiệu từ bên ngoài cũng như xuất tín hiệu ra. Ngoài ra trên board còn có khối JTAG giúp ta phân tích chương trình, sửa lỗi thời gian thực. Các yêu cầu phần cứng cao hơn cũng có thể được đáp ứng với các khe cắm mở rộng được thiết kế sẵn trên bảng mạch. Sơ đồ cấu tạo của bảng mạch được miêu tả ở hình vẽ dưới:

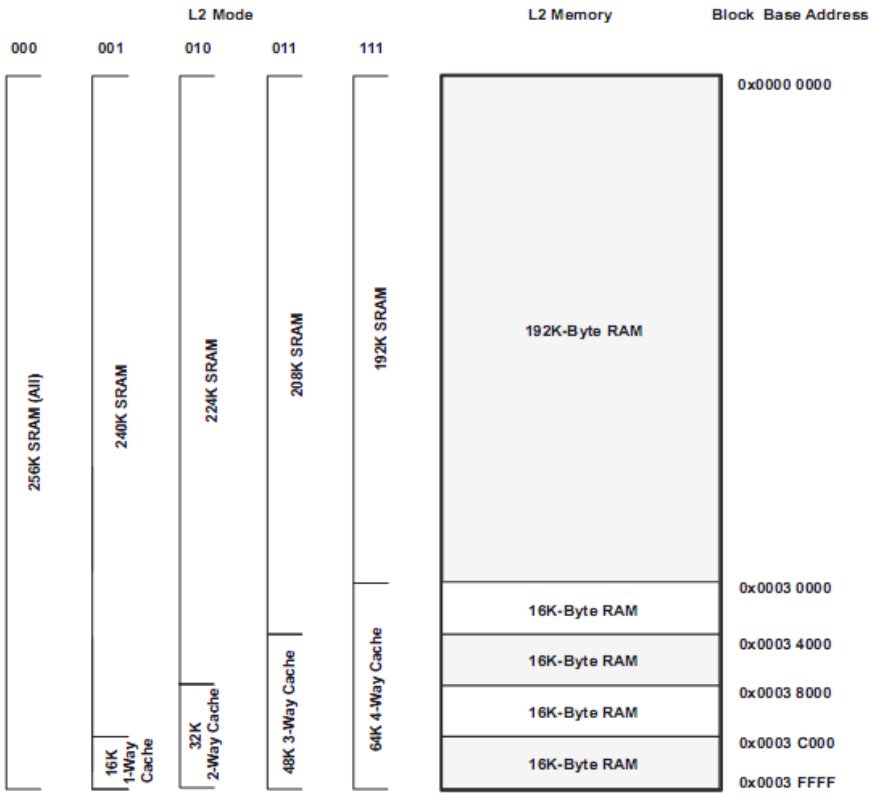


Hình 8: Sơ đồ khối kit TMS320C6713

Cụ thể cấu tạo của bảng mạch gồm các thành phần sau:

- Trung tâm của bảng mạch là chip xử lý tín hiệu TMS320C6713, chạy ở xung nhịp 225 MHz. TMS320 là tên chung cho một loạt các bộ xử lý số đến từ Texas Instrument. Nằm trong dòng chip TMS320C6x của TI, đây là dòng vi xử lý tốc độ cao, sử dụng kiến trúc đặc biệt để đáp ứng các tác vụ xử lý tín hiệu. Dựa trên kiến trúc VLIW (Very Long Instruction Word), TMS320C6713 có khả năng xử lý các số thực dấu phẩy động. và được coi là dòng chip xử lý tín hiệu mạnh nhất của TI hiện nay.
- Bộ biến đổi tín hiệu AIC23 sử dụng công nghệ sigma delta, đóng vai trò biến đổi tương tự - số và ngược lại. Tần số lấy mẫu có thể thay đổi từ 8 đến 96 kHz.
- Bốn cổng kết nối tín hiệu vào ra: MIC IN, LINE IN, LINE OUT, và HEADPHONE.
- Bốn đèn LED và công tắc DIPS. Các đèn và khóa có thể cấu hình điều khiển theo nhu cầu của người sử dụng.
- Cổng USB để giao tiếp với PC. Trên cổng cũng được thiết kế bộ JTAG nhúng giúp ta có thể sửa lỗi chương trình chạy trên chip mà không cần nối JTAG ngoài.
- Cổng PWR (+5V) cung cấp nguồn cho board. Cổng này cung cấp điện thế 1,26 V cho chip C6713 và 3,3 V để nuôi bộ nhớ và các thiết bị ngoại vi khác.

- Bộ nhớ trong: trên mạch có 264 kB bộ nhớ trong. Dưới đây là sơ đồ phân vùng và địa chỉ bộ nhớ trong của kit [9]



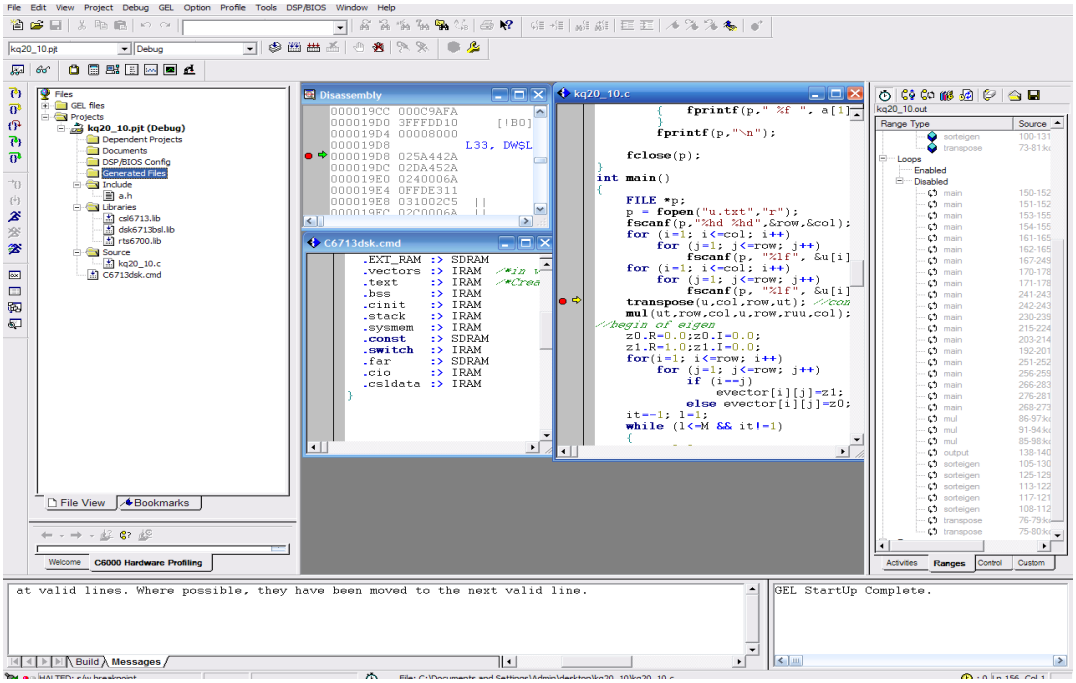
Hình 9: Sơ đồ và địa chỉ vùng nhớ L2 của kit

- Bộ nhớ ngoài: kit DSP có sẵn 16 Mb bộ nhớ ngoài. Các bộ nhớ ngoài này đều là các bộ nhớ truy cập ngẫu nhiên (RAM). Ngoài ra kit cũng có thể bổ sung bộ nhớ ngoài qua khe cắm mở rộng. Với chiều dài thanh ghi là 32 bit, kit có thể quản lý 4 GB bộ nhớ ngoài.

Có thể thấy rằng tuy xung nhịp không cao nhưng kit TMS320C6713 có dung lượng bộ nhớ lưu trữ lớn, khả năng xử lý dấu phẩy động, có sẵn JTAG nhúng thuận tiện cho debug và tính năng xử lý thời gian thực RTDX, hoàn toàn thích hợp để thực thi thuật toán MUSIC trên kit.

2.4 Code Composer Studio

Để giao tiếp giữa bộ xử lý số và PC, TI có cung cấp cho ta công cụ là Code Composer Studio (CCS). CCS được xây dựng trên nền tảng Eclipse, là một môi trường phát triển tích hợp (IDE) khá tốt. Sử dụng CCS, ta có thể thiết kế, chỉnh sửa, sửa lỗi trong code và biên dich code. CCS còn cung cấp tính năng phân tích code thời gian thực, từ đó có thể tối ưu phần cứng, phần mềm để thực hiện hệ thống thời gian thực.



Hình 10: Giao diện chương trình Code Composer Studio

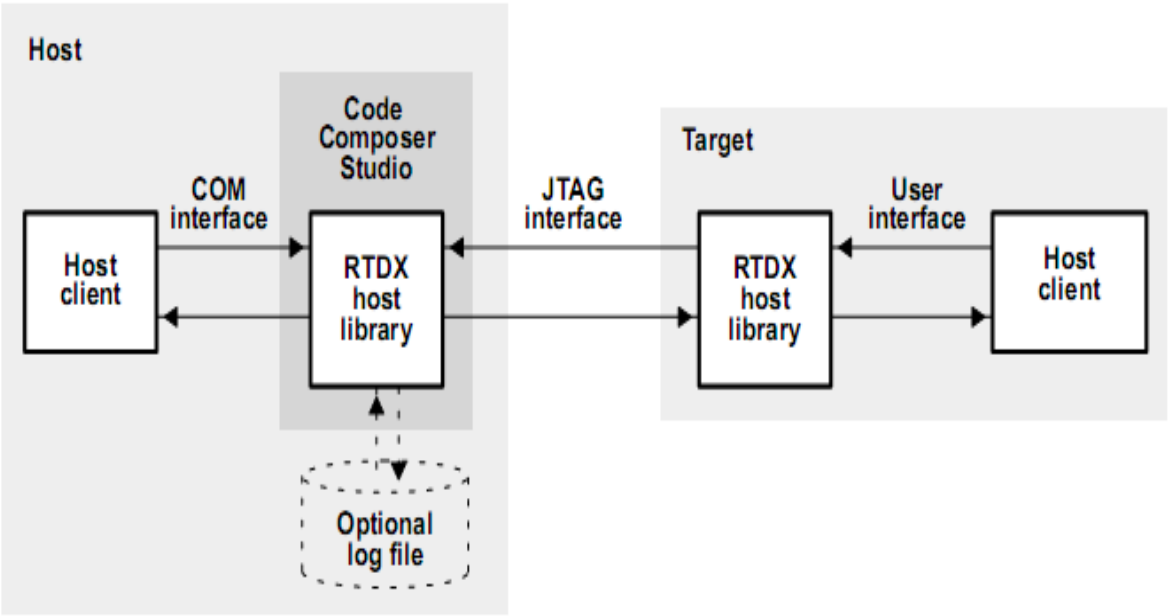
Trong CCS có nhiều dạng file với nhiều mục đích khác nhau. Bảng dưới đây giới thiệu một số dạng file và vai trò của nó [7]:

Bảng 3: Các dạng file sử dụng trong CCS

STT	***Dạng file***	***Vai trò***
1	file.pjt	Tập tin được tạo khi xây dựng 1 project để quản lý file và thông tin của project
2	file.c	Các tập tin chứa code C, phần thuật toán chính của chương trình
3	file.asm	Các tập tin chứa mã hợp ngữ được tạo ra bởi trình biên dịch C, hoặc do người sử dụng
4	file.h	Các tập tin tiêu đề hỗ trợ cho project, có thể sử dụng để chứa hàm hoặc dữ liệu
5	file.lib	Các thư viện hỗ trợ khởi tạo Chip, mạch, ADC và các thiết bị ngoại vi trên bảng mạch
6	file.cmd	Các tập tin liên kết giúp phân hoạch các vùng chương trình và dữ liệu vào bộ nhớ
7	file.out	Tập tin khả thi, được tạo bởi CMD file để có thể nạp vào kit
8	file.cdb	Tập tin cấu hình, sử dụng khi dùng tính năng DSP/BIOS

2.5 Tích hợp CCS với Matlab thông qua tính năng Real Time Data Exchange

Để tích hợp môi trường Code Composer Studio với môi trường Matlab nhằm trao đổi dữ liệu một cách liền mạch giữa kit và PC, ta có thể sử dụng tính năng RTDX. Tính năng này cho phép trao đổi tín hiệu giữa máy chủ PC và các thiết bị DSP mà không cần dừng các ứng dụng điều khiển. Mô hình giao tiếp giữa kit và máy tính được thể hiện ở hình dưới đây [8]:



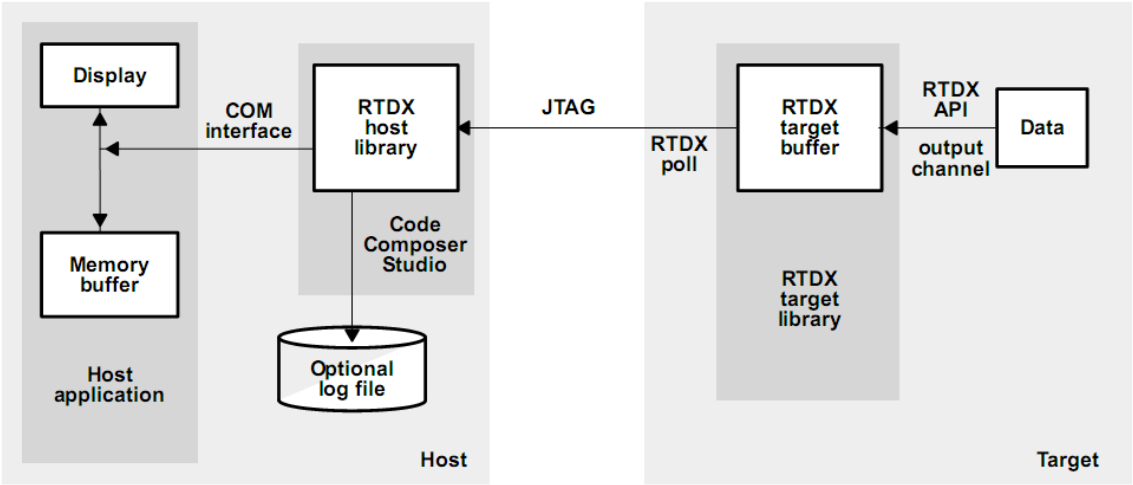
Hình 11: Mô hình giao tiếp giữa kit và PC thông qua RTDX

Tính năng RTDX bao gồm các thành phần nằm trên cả máy tính và kit DSP. Nó cho phép dữ liệu được truyền trong các kênh vào ra riêng biệt, do đó giảm thiểu sự mất mát do xung đột dữ liệu. Khi sử dụng RTDX với môi trường lập trình khác CCS (Matlab, Visual C), ta có thể kết nối với kit DSP, điều khiển kit mà không cần qua CCS, do vậy không cần truy cập vào CCS để khởi tạo và chạy chương trình. Việc sử dụng cũng đơn giản hơn khi ta có thể lập trình giao diện ở môi trường khác, rồi kết nối với CCS.

RTDX kết nối với Matlab thông qua toolbox "The Embedded Target for TI C6000 DSP" và "MATLAB Link for CCS". Quá trình kết nối được chia thành hai phần:

- Liên kết từ kit đến máy tính
- Liên kết từ máy tính đến kit

Trong quá trình liên kết từ kit đến máy tính, trước tiên 1 kênh truyền được kit khởi tạo, sử dụng các hàm API của RTDX để chuyển dữ liệu qua kênh truyền này đến bộ nhớ đệm tạo bởi thư viện RTDX. Sau đó dữ liệu được gửi tới máy tính qua giao diện JTAG. Dữ liệu được thu thập tại máy tính, sau đó được ghi vào bộ nhớ đệm của phần mềm máy tính, hoặc ghi ra log file. Quá trình này diễn ra liên tục, tự động mà không cần phải dừng xử lý trên chip. Sơ đồ biểu diễn quá trình được thể hiện qua hình dưới.



Hình 12: Quá trình liên kết từ kit đến máy tính

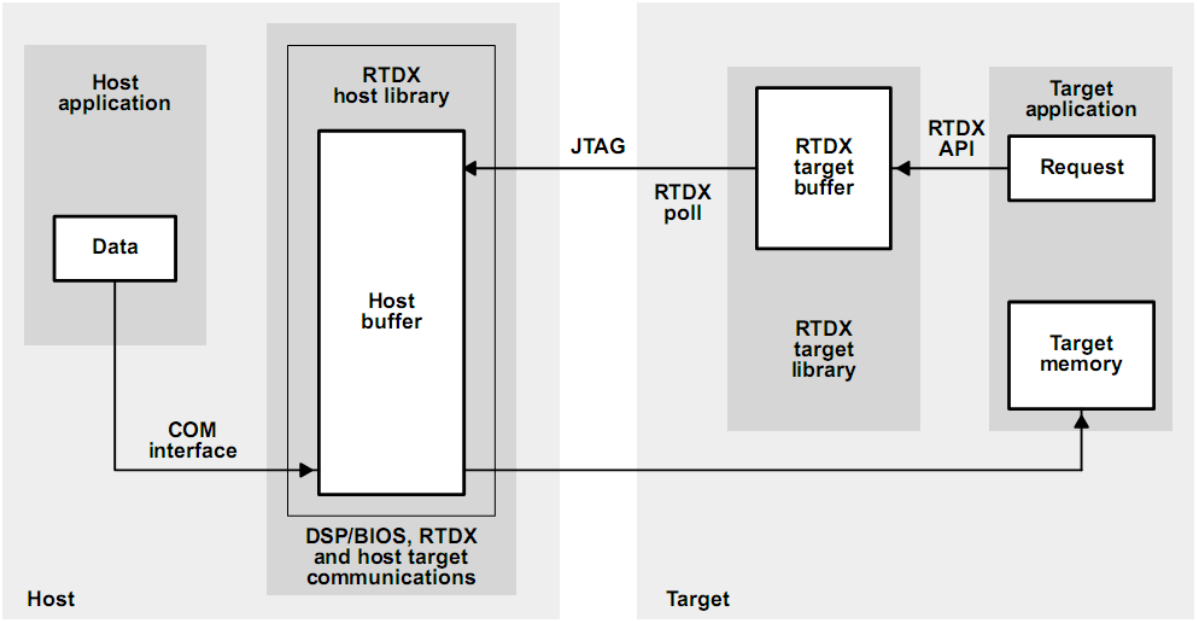
Để kết nối với RTDX, trong CCS ta cần khai báo thư viện RTDX.h. Các lệnh cơ bản để thiết lập liên kết và kênh truyền từ kit đến Matlab:

Bảng 4: Các lệnh cơ bản trong thư viện RTDX.h để liên kết CCS→Matlab

Lệnh	Tác dụng
RTDX_CreateInputChannel()	Tạo kênh truyền từ Kit đến máy tính. Tên kênh truyền phải định nghĩa giống trên Matlab
RTDX_CreateOutputChannel()	Tạo kênh truyền từ Kit đến PC
RTDX_read	Đọc dữ liệu từ kênh truyền
RTDX_write	Gửi dữ liệu đến kênh truyền

Để có thể liên kết và truyền dữ liệu từ máy tính đến kit, trước tiên ta phải định nghĩa một kênh nhận dữ liệu ở kit (xem bảng 6). Tiếp đó, khi kit yêu cầu dữ liệu từ kênh nhận, toàn bộ dữ liệu sẽ được gửi sang bộ nhớ đệm trên kit, sau đó được ghi vào từng vùng nhớ xác định. Sau khi truyền hết dữ liệu, máy tính sẽ thông báo cho RTDX để đóng kênh truyền. Quá trình này giao tiếp từ máy tính đến kit được mô tả ở hình

dưới:



Hình 13: Quá trình liên kết từ máy tính đến kit

Các lệnh cơ bản để tạo liên kết từ Matlab tới kit, khởi chạy chương trình, tạo các kênh truyền được mô tả trong bảng sau [7]:

Bảng 5: Các lệnh cơ bản của toolbox để liên kết Matlab → CCS

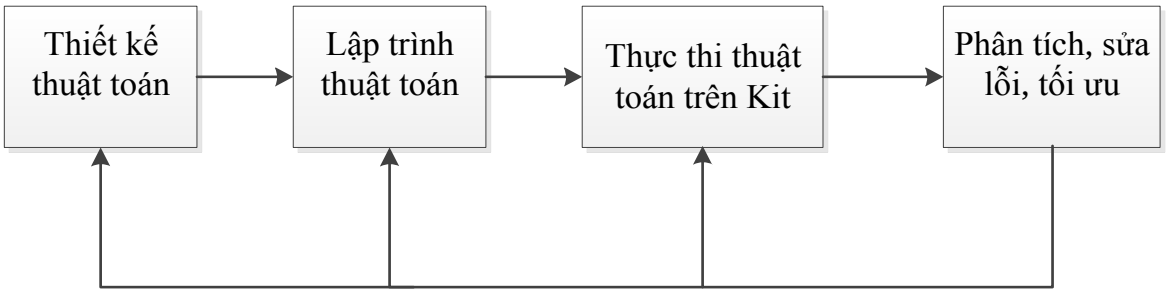
Lệnh	Tác dụng
ccsboardinfo	Lấy thông tin về bảng mạch và chip chuẩn bị kết nối
cc = ccsdsp('boardnum',0)	Tạo liên kết đến CCS
cc.rtdx.configure(num1,num2)	Tạo bộ đệm có kích cỡ num1 byte cho num2 kênh truyền
cc.rtdx.open('name','sign')	Tạo các kênh truyền dùng để đọc (sign='r') dữ liệu từ Kit hoặc viết (sign='w') chuyển dữ liệu ngược lại. Tên kênh truyền (trường name) có thể đặt tùy ý
cc.rtdx.enable	Kích hoạt kênh truyền
cc.rtdx.set('timeout', num)	Đặt lại thời gian đợi việc truyền dữ liệu. Tùy vào kích cỡ và loại dữ liệu mà thời gian này phải đặt hợp lý

CHƯƠNG 3

THỰC THI THUẬT TOÁN MUSIC TRÊN KIT TMS320C6713

Ngày nay với sự phát triển không ngừng của các phần mềm mô phỏng như Matlab, chúng ta có khả năng thiết kế thuật toán chỉ dựa vào các công cụ Simulink, sau đó sử dụng các toolbox như Real-Time Workshop để sinh ra code C, từ đó đưa vào kit để thực hiện. Tuy nhiên code được tạo ra có dung lượng lớn và khó để tùy biến tối ưu về sau. Vì vậy em lựa chọn giải pháp tự thiết kế thuật toán trên C, sau đó chuyển code vào môi trường CCS để biên dịch thành chương trình chạy cho kit DSP.

Các bước xây dựng thuật toán trên nền tảng DSP bao gồm 4 bước được mô tả bởi lưu đồ dưới đây:



Hình 14: Mô hình xây dựng thuật toán

3.1 Thiết kế thuật toán

Cụ thể về thuật toán MUSIC đã đươc nghiên cứu ở chương đầu. Tuy nhiên khi nhúng vào DSP, ta phải làm sao cho chương trình không quá nặng nề với chip DSP, bảo đảm chương trình vẫn cho ra kết quả khá chính xác với tốc độ nhanh nhất. Sau đó ta có thể mô phỏng thuật toán trên Matlab để kiểm tra tính đúng đắn của thuật toán, trước khi áp dụng vào kit DSP.

3.2 Lập trình thuật toán

Sau khi đã có một giải thuật đúng đắn, đã được kiểm chứng bằng Matlab, ở bước này ta tiến hành chuyển đổi các dòng code Matlab sang code C. Tuy nhiên do Matlab được xây dựng dựa trên hai thư viện EISPACK và LINPACK [10], vốn rất mạnh về xử lý các phương trình tuyến tính, tính toán các ma trận (LINPACK), và giải các giá trị riêng, vectơ riêng của ma trận (EISPACK), nó cung cấp nhiều hàm tính toán số học mà với ngôn ngữ C không thể giải quyết qua một vài dòng lệnh. Những khó khăn khi chuyển thuật toán MUSIC sang ngôn ngữ C được tóm tắt qua bảng dưới:

Bảng 6: So sánh thuật toán MUSIC trên ngôn ngữ Matlab và ngôn ngữ C

STT	Các bước thực hiện	Code Matlab	Code C	Khó khăn
1	Thu nhân tín hiệu vào	Có sẵn trường số phức	Không có	Phải xây dựng trường số phức với các phép tính thông thường và các phép biến đổi ma trận
2	Tính ma trận tự tương quan	Thuận tiện trong việc tính toán ma trận, có thể thực hiện bước này trong 1 dòng lệnh	Sử dụng cấu trúc mảng với các giá trị thực phức, tính toán chậm	Các biến xử lý đều là các ma trận kích cỡ lớn, mỗi phần tử đều là các số phức nên đòi hỏi tối ưu để tính toán nhanh hơn
3	Tìm giá trị riêng, vectơ riêng	Sử dụng hàm eig có sẵn trong Matlab	Không có hàm có sẵn	Đòi hỏi tìm kiếm giải thuật tính giá trị riêng, vectơ riêng.
4	Phân tách thành không gian tín hiệu và không gian nhiễu, tính phổ không gian của tín hiệu	Hàm eig tự sắp xếp các giá trị riêng và vectơ riêng tương ứng	Không có khả năng này	Phải sắp xếp các giá trị riêng, vectơ riêng theo thứ tự giảm dần

Từ bảng trên ta thấy thách thức lớn nhất khi thực hiện thuật toán MUSIC bằng ngôn ngữ C là phải tính được giá trị riêng, vectơ riêng của ma trận tự tương quan. Có 2 giải pháp đưa ra để giải quyết vấn đề này.

- Sử dụng các thư viện tích hợp sẵn, có các hàm với chức năng tương được thư viện EISPACK của Matlab.
- Thiết kế thuật toán phù hợp để tính giá trị riêng, vectơ riêng.

Việc sử dụng các thư viện tích hợp sẵn có ưu điểm về tốc độ do đã được tối ưu sẵn, tiết kiệm thời gian tìm hiểu và lập trình thuật toán tính giá trị riêng, vectơ riêng, nhất là khi các thuật toán này rất phức tạp. Tuy nhiên khi áp dụng vào khóa luận, các

thư viện tích hợp lại có những hạn chế sau:

- Kích cỡ: các thư viện tích hợp viết bằng ngôn ngữ C thường có kích cỡ lớn, một phần do phải bao quát hết tất cả các trường hợp (một thư viện chuẩn như EISPACK có các tiến trình khác nhau cho 9 trường hợp ma trận, mỗi tiến trình lại có nhiều phương pháp tính toán khác nhau). Do vậy gặp khó khăn khi đưa các thư viện này vào kit DSP vốn có bộ nhớ hữu hạn.
- Giá thành: ngoài các thư viện ở trên còn có các loại thư viện đặc biệt được tối ưu cho các hoạt động xử lý trên DSP. Chúng có kích cỡ nhỏ và tốc độ xử lý nhanh. Tuy nhiên giá các thư viện này không hề rẻ, hơn nữa do khác biệt về cấu trúc mà những thư viện này chỉ sử dụng cho một dòng chip nhất định.

Thêm vào đó, từ việc phân tích dạng ma trận tự tương quan R_{uu} cho thấy việc tìm giá trị riêng và vectơ riêng cho ma trận này chỉ là một trường hợp đặc biệt, do đó việc lập trình thuật toán trở nên đơn giản hơn. Do vậy em sử dụng phương án *lập trình thuật toán tính giá trị riêng, vectơ riêng cho ma trận tự tương quan* trong khóa luận này.

Bài toán tính giá trị riêng, vectơ riêng

1. ***Tính toán lý thuyết***

Để tìm giá trị riêng, vectơ riêng của 1 ma trận ta làm theo các bước sau [2]:

- Bước 1: Giải phương trình đặc trưng $\det(A - \lambda * I) = 0$

  Nghiệm của phương trình tìm được là các giá trị riêng λ cần tìm

- Bước 2: Ứng với mỗi giá trị riêng λ vừa tìm được, giải hệ phương trình tuyến tính thuần nhất $(A - \lambda_i \times I) \times u = 0$ với mỗi nghiệm λ_i vừa tìm được ở trên,

  Kết quả là vectơ riêng u_i ứng với giá trị riêng λ_i.

2. ***Áp dụng vào việc tính toán trên máy tính***

Từ định lý Anbơ: với m > 5 tồn tại phương trình bậc *m* mà không thể biểu diễn nghiệm tổng quát bằng cách sử dụng số hữu tỉ, phép cộng, trừ, nhân, chia và lấy khai căn.

Định lí này chỉ ra rằng không thể có thuật toán giải phương trình nào cho ra nghiệm chính xác của đa thức bậc bất kì lớn hơn 5 sau một số bước hữu hạn. Vì vậy không thể áp dụng phương pháp trên để tìm trị riêng của các ma trận có số chiều lớn

hơn 4. Điều này đòi hỏi ta phải sử dụng phương pháp khác, cụ thể ở đây là phương pháp lặp đơn Jacobi.

3. ***Phương pháp lặp đơn Jacobi***

Xét ma trận tương quan của mảng tín hiệu: R_{uu} là một ma trận Hermitian cỡ N×N, đây là dạng ma trận vuông đối xứng với các phần tử là số phức.

Một ma trận vuông đối xứng bất kì sẽ luôn chéo hóa được. Ý tưởng của phương pháp Jacobi trong khóa luận là chéo hóa ma trận R_{uu} để thu được ma trận đường chéo:

$$D = O^T R_{uu} O = \text{diag}(\lambda_1, \lambda_2, \ldots \lambda_n) \quad (4.1)$$

Khi đó các phần tử đường chéo là các gía trị riêng $\lambda_1, \lambda_2, \ldots . \lambda_n$, còn các cột của ma trận làm chéo hóa O sẽ là các vectơ riêng tương ứng với các giá trị riêng cần tìm

4. ***Lập trình phương pháp lặp đơn jacobi***

Xây dựng một dãy các ma trận $(O_k)_{k\geq 1}$ các ma trận trực giao sao cho [11]:

$$A_{k+1} = O_k^T A_k O_k = (O_1 O_2 \ldots O_k)^T A (O_1 O_2 \ldots O_k) \forall k \geq 1 \quad (4.2)$$

Biến đổi liên tục dãy này dẫn đến ma trận A hội tụ về một ma trận đường chéo

$$A_{k+1} = \begin{pmatrix} \lambda_1 & 0 & 0 & \ldots & 0 \\ 0 & \lambda_2 & .. & ... & 0 \\ 0 & \ldots & \lambda_3 & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & 0 \\ 0 & 0 & \ldots & 0 & \lambda_n \end{pmatrix} \quad (4.3)$$

Đặt $O_k = O_1 O_2 \ldots O_k$ là các ma trận trực giao có cột là các vectơ hội tụ về vectơ riêng của ma trận A

Nguyên tắc biến đổi:

$$A_k A_{k+1} \Rightarrow O_k^T A_k O_k \forall k \geq 1 \quad (4.4)$$

Triệt tiêu hết các phần tử a_{ij} (i≠j) hay các phần tử nằm ngoài đường chéo chính của ma trận:

$$\text{Đặt: } \begin{cases} A_{k+1} = (b_{ij}) \\ A_k = (a_{ij}) \\ O_k = O \end{cases} \quad (4.5)$$

Ta thực hiện các việc quay Jacobi các phần tử của ma trận A dựa vào công thức quay:

Định lý: nếu $a_{pq} \neq 0$, tồn tại một và chỉ một giá trị của $\theta \in (-\frac{\pi}{4}, 0) \cup (0, \frac{\pi}{4})$ làm cho $b_{pq} = 0$. Giá trị θ được xác định như sau:

$$\operatorname{cotan} 2\theta = \frac{a_{qq} - a_{pp}}{2a_{pq}} \tag{4.6}$$

Từ đó tính được các phần tử của ma trận A_{k+1} theo A_k:

$$\begin{cases} b_{ij} = a_{ij} \\ b_{pj} = a_{pi}\cos\theta - a_{qi}\sin\theta \\ b_{qi} = a_{pi}\sin\theta + a_{qi}\cos\theta \\ b_{pp} = a_{pp}\cos^2\theta + a_{qq}\sin^2\theta - a_{pq}\sin 2\theta \\ b_{qq} = a_{pp}\sin^2\theta + a_{qq}\cos^2\theta + a_{pq}\sin 2\theta \\ b_{pq} = b_{qp} = a_{pq}\cos 2\theta + \frac{a_{pp} - a_{qq}}{2}\sin 2\theta \end{cases} \tag{4.7}$$

Từ công thức (3.1), sử dụng các phép biến đổi lượng giác ta có:

Đặt = tan θ là nghiệm $|\theta| \leq \frac{\pi}{4}$ của phương trình bậc 2:

$$t^2 + 2mt - 1 = 0 \tag{4.8}$$

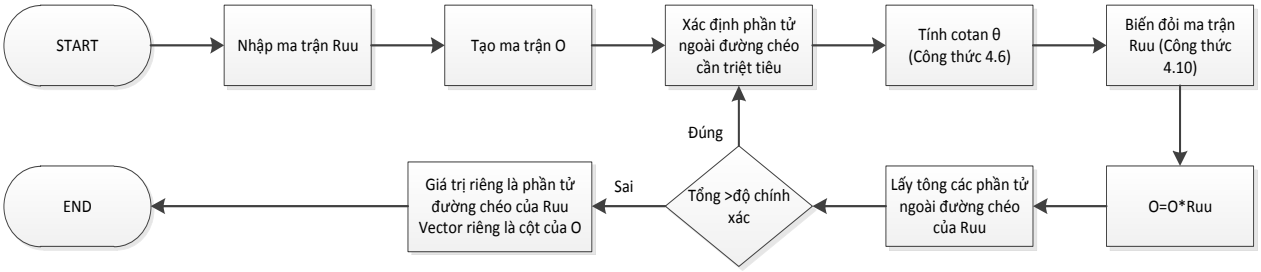
Các hàm lượng giác còn lại của θ theo tan θ:

$$\begin{cases} c = \cos\theta = \dfrac{1}{\sqrt{1+t^2}} \\ s = \sin\theta = \dfrac{t}{\sqrt{1+t^2}} \\ \sin 2\theta = \dfrac{2t}{1+t^2} \\ \cos 2\theta = \dfrac{1-t^2}{1+t^2} \end{cases} \tag{4.9}$$

Kết hợp với (4.10) ta thu được công thức quay thu gọn:

$$\begin{cases} b_{pi} = ca_{pi} - sa_{qi} \\ b_{qi} = ca_{qi} + sa_{pi} \\ b_{pp} = a_{pp} - ta_{pq} \\ b_{qq} = a_{qq} + ta_{pq} \\ b_{pq} = b_{qp} = a_{pq}\cos 2\theta + \dfrac{a_{pp} - a_{qq}}{2}\sin 2\theta \end{cases} \quad (4.10)$$

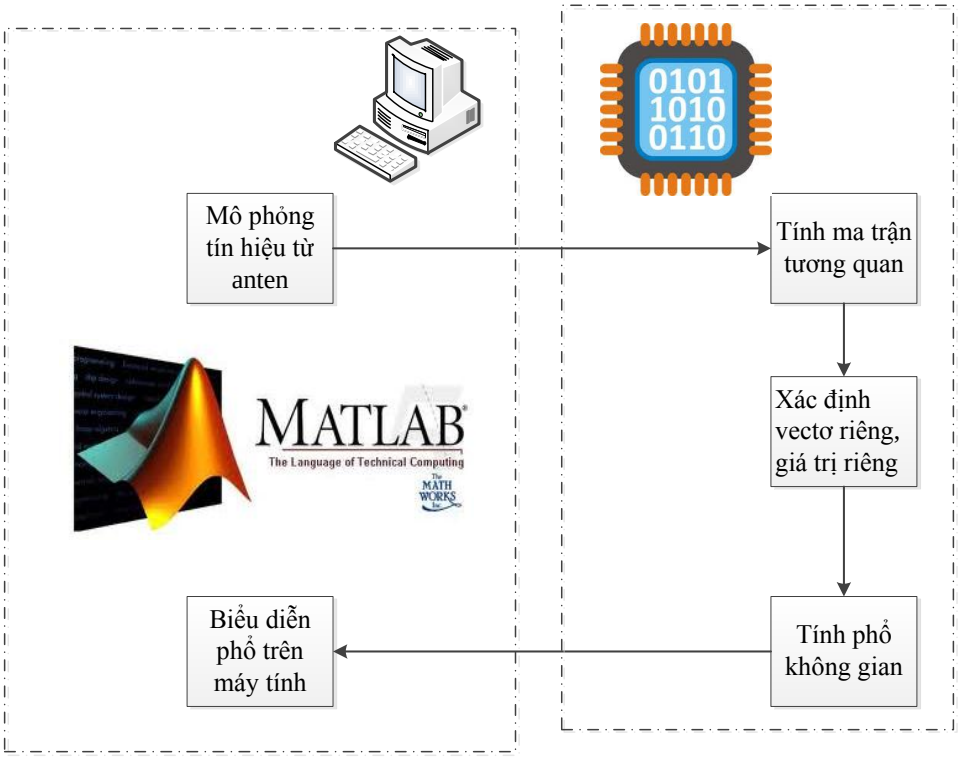
Sau mỗi bước quay, cặp phần tử nằm ngoài đường chéo chình bị triệt tiêu, tuy nhiên các phần tử bị triệt tiêu trước đó lại khác không. Tuy nhiên các giá trị này sẽ ngày càng tiến tới 0 đến khi thỏa mãn điều kiện về độ chính xác, ta nhận được ma trận với các phần tử nằm ngoài đường chéo có giá trị xấp xỉ 0. Giải thuật cài đặt trên máy tính có lưu đồ như sau:



Hình 15: Lưu đồ thuật toán Jacobi

3.3 Thực thi thuật toán trên kit

Với mục đích tập trung vào thuật toán và khả năng thực thi của thuật toán trên kit DSP, khóa luận không đi sâu vào thực hiện các phần cứng khác của hệ thống, ví dụ như hệ thống anten. Thay vào đó các tín hiệu đầu vào của dàn anten được mô phỏng trên chương trình Matlab rồi được đưa vào kit để xử lý. Sau khi kit xử lý xong, kết quả về phổ không gian của tín hiệu được đưa lại trở về máy tính để vẽ phổ tín hiệu bằng chương trình Matlab. Tóm tắt quá trình thực thi trên kit như sau:



Hình 16: Quá trình thực thi thuật toán MUSIC trên kit DSP

3.3.1 Mô tả quá trình

1. Mô phỏng tín hiệu từ anten

Các tín hiệu thu thập của hệ anten được tính bởi các yếu tố sau:

- Vectơ tín hiệu đầu vào s được xác định theo phân bố Gaussian, với giá trị trung bình $|s| = 20^{\frac{SNR}{10}}$

- Vectơ lái a(θ) phụ thuộc vào góc θ trong không gian, vị trí của mối phần tử anten trong không gian

  Vectơ nhiễu N, với các giá trị ồn giả ngẫu nhiên cho từng mẫu tín hiệu thu được.

  Khi đó vectơ cuối cùng thu được tại mỗi phần tử anten sẽ được tính bởi

$$U = AS + N \tag{4.11}$$

2. Tính ma trân tự tương quan của vectơ tín hiệu thu được

Giả thiết nhiễu không tương quan và là các biến đồng mức:

$$R_{uu} = \frac{(A \times A^T)}{N_s} \tag{4.12}$$

với N_s là số mẫu tín hiệu thu được.

3. ***Xác định các vectơ riêng và giá trị riêng tương ứng với không gian nhiễu:*** sử dụng phương pháp Jacobi đã được đề cập ở trên.

Sau khi các giá trị riêng được sắp xếp từ nhỏ nhất tới lớn nhất, chúng ta có thể chia ma trận R_{uu} thành hai không gian con như sau:

$$R_{uu} = [V_s \quad V_n] \tag{4.13}$$

Trong đó:

V_s được gọi là không gian con tín hiệu và chứa M vectơ riêng kết hợp với các tín hiệu tới. Không gian con tín hiệu là ma trận cỡ N×M.

V_n được gọi là không gian con của nhiễu và bao gồm N – M vectơ riêng kết hợp với nhiễu. Không gian con của nhiễu là một không gian cỡ N×(N – M).

4. ***Tính phổ không gian của tín hiệu***

Phổ không gian của tín hiệu được tính theo công thức:

$$P(\theta) = \frac{a^H(\theta)a(\theta)}{a^H(\theta)V_nV_n^Ha(\theta)} \tag{4.14}$$

Trong đó, khi cho góc θ thay đổi từ 0 đến 360^0, nếu góc θ trùng với góc tới của nguồn tín hiệu, P(θ) đạt cực đại và ta sẽ quan sát được một đỉnh phổ trong phổ tín hiệu quan sát ở bước 5.

5. ***Biểu diễn phổ trên máy tính***

Với dữ liệu về phổ không gian được kit trả về, thực hiện lệnh plot (θ,P) trên Matlab.

3.3.2 Phân vùng bộ nhớ của kit cho dữ liệu và chương trình

Theo mặc định ban đầu, kit chỉ sử dụng bộ nhớ trong để lưu trữ các dữ liệu cũng như chương trình được nạp vào. Do bộ nhớ này khá nhỏ (264 kB), ta cần phải cấu hình vùng nhớ mà kit có thể sử dụng thông qua file liên kết (Linker File) [7]:

Cấu trúc tập tin liên kết:

```
MEMORY
{
 IVECS:  org=0h,          len=0x220
```

```
    IRAM:       org=0x00000220,   len=0x0002FDE0 /*internal memory*/
    EDATA:      org=0x80000000,   len=0x0100000
    SDRAM:      org=0x80F00000,   len=0x01000000 /*external memory*/
    FLASH:      org=0x90000000,   len=0x00020000 */ /*flash memory*/
}
 SECTIONS
 {
   .EXT_RAM :> SDRAM
   .vectors :> IRAM
   .text    :> IRAM
   .bss     :> IRAM
   .cinit   :> IRAM
   .stack   :> IRAM
   .sysmem :> IRAM
   .const   :> SDRAM
   .switch  :> IRAM
   .far     :> SDRAM
   .cio     :> IRAM
   .csldata :> IRAM
 }
```

Để ánh xạ một trường bất kỳ sang bộ nhớ ngoài chỉ cần sửa "IRAM" thành "SDRAM". Ví dụ: .const : > SDRAM

Để ánh xạ một biến bất kỳ trong code (có thể là biến cần kích cỡ lớn mà bộ nhớ trong không chứa được) ta làm như sau:

- Định nghĩa một vùng nhớ trong bộ nhớ ngoài. Ví dụ:

EDATA: org = 0x80000000, len = 0x0100000 định nghĩa phần nhớ ngoài EDATA kích thước 1Mb.

- Trong phần khai bào biến dùng cấu trúc *#pragma* để ánh xạ biến vào vùng nhớ vừa tạo. Ví dụ với biến var:

#pragma DATA_SECTION (var,"EDATA");

3.3.3 Kết quả thực thi thuật toán:

Tiến hành thực thi thuật toán trên kit DSP với các dàn anten ULA, UCA trong trường hợp số nguồn sóng tới nhỏ hơn số phần tử anten,sử dụng mô hình của hệ giống với trường hợp mô phỏng:

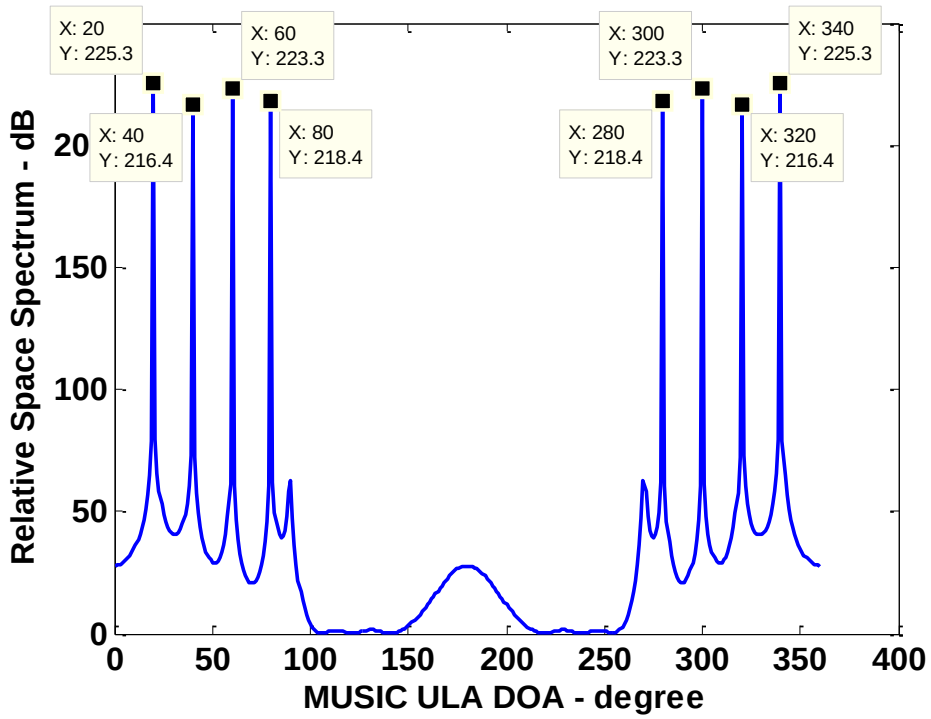
Với dàn anten ULA:

- Mô hình hệ thống:

Bảng 7: Mô hình hệ thống thực thi thuật toán MUSIC với anten ULA

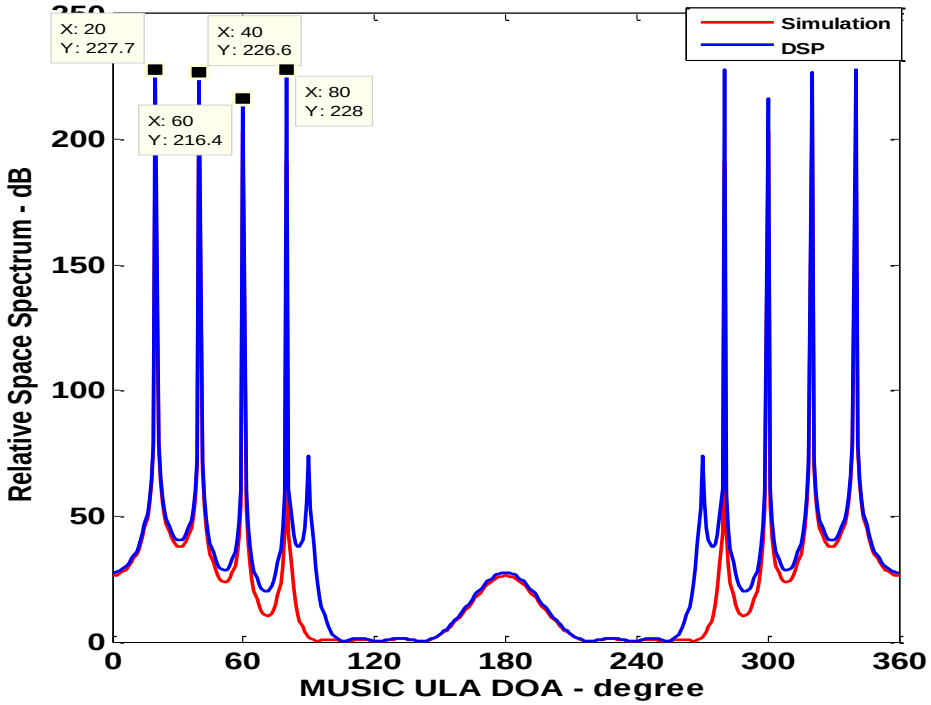
STT	Các thông số của nguồn	Các thông số của hệ anten
1	Số nguồn tín hiệu: 4	Số phần tử anten: 8
2	Góc tới của các nguồn tín hiệu: [20^o 40^o 60^o 80^o]	Độ dài cung tròn: m= λ/2 (m)
3	Tỷ số SNR: 25dB	

- Kết quả thực thi:



Hình 17: Thực thi hệ thống DSP với mảng anten ULA

Nhận xét: thực thi hệ thống trên DSP cho kết quả xác định 8 đỉnh phổ tương đương với 8 nguồn sóng tới. Trong đó hệ thống DSP xác định chính xác 4 nguồn tín hiệu ở các góc [20^0 40^0 60^0 80^0]. Các hướng còn lại là đối xứng với 4 nguồn tín hiệu, đây là hạn chế của dàn anten ULA mà ta đã nói ở trên.



Hình 18: So sánh kết quả thực thi thuật toán và mô phỏng trên dàn anten ULA

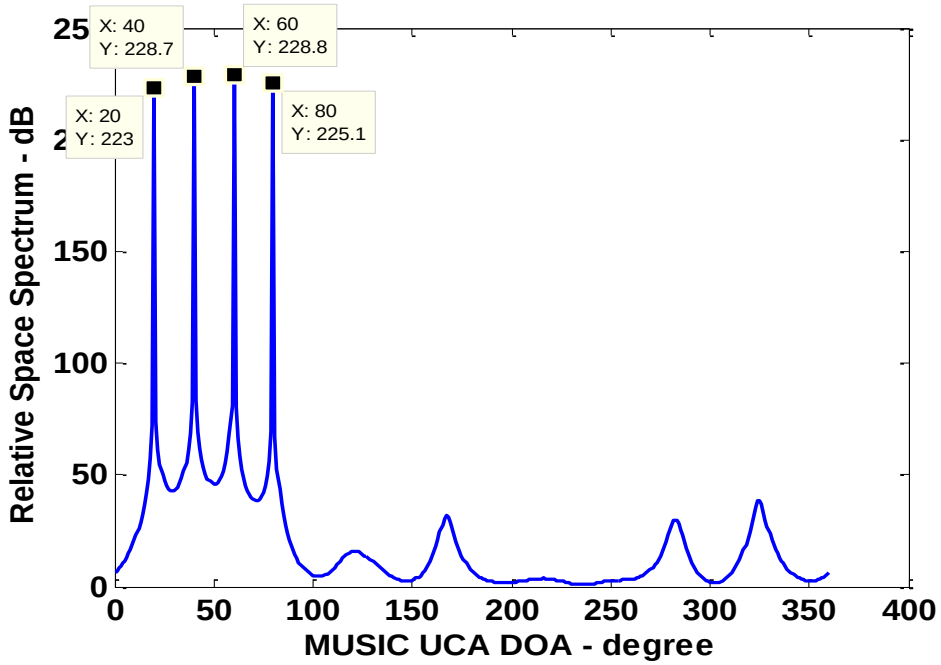
Nhận xét: so với kết quả mô phỏng, kết quả thực thi có thêm hai đỉnh với biên độ không đáng kể so với biên độ của các nguồn tín hiệu. Mặc dù cả hai cùng xác định chính xác các góc tới, ta có thể nhận thấy rằng kết quả thực thi kém hơn một chút so với mô phỏng, thể hiện ở các mức ồn cao hơn. Điều này có lẽ là do thuật toán tìm giá trị riêng không được tối ưu như thuật toán trên Matlab.

Với dàn anten UCA:

- Mô hình hệ thống:

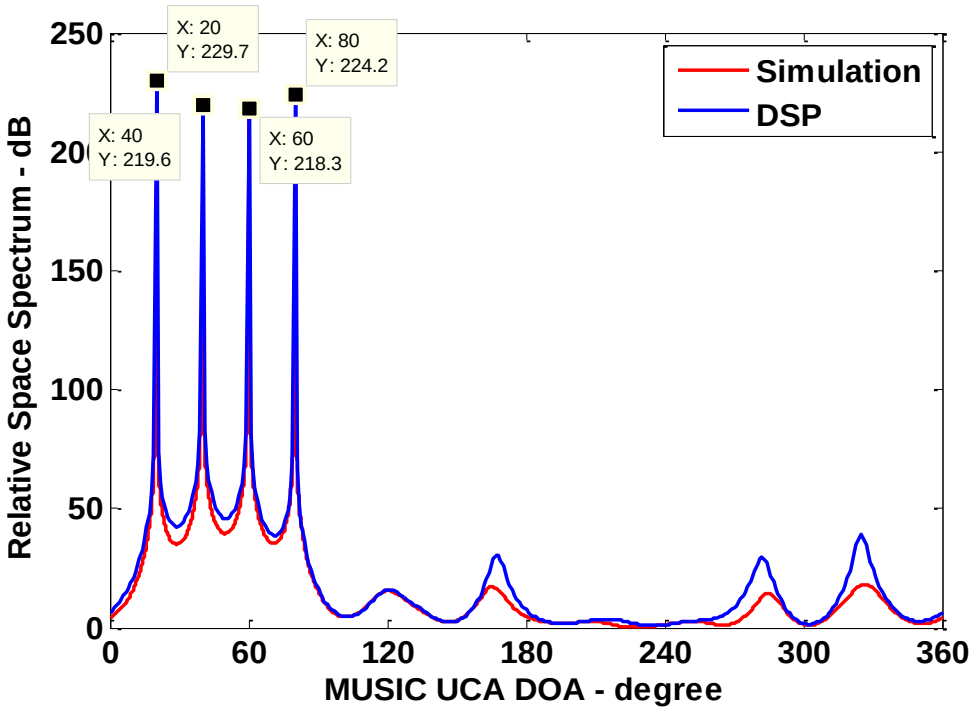
STT	Các thông số của nguồn	Các thông số của hệ anten
1	Số nguồn tín hiệu: 4	Số phần tử anten: 8
2	Góc tới của các nguồn tín hiệu: [20^o 40^o 60^o 80^o]	Khoảng cách giữa các phần tử: λ/2 (m)
3	Tỷ số SNR: 25dB	

- Kết quả thực thi:



Hình 19: Thực thi hệ thống DSP với mảng anten UCA

Nhận xét: thực thi trên hệ UCA cho ta kết quả xác định chính xác bốn góc tới ứng với các nguồn tín hiệu ở [20^0 40^0 60^0 80^0]. Kết quả đánh giá độ chính xác so với hệ mô phỏng được thể hiện ở hình dưới:



Hình 20: So sánh kết quả thực thi thuật toán và mô phỏngtrên dàn anten UCA

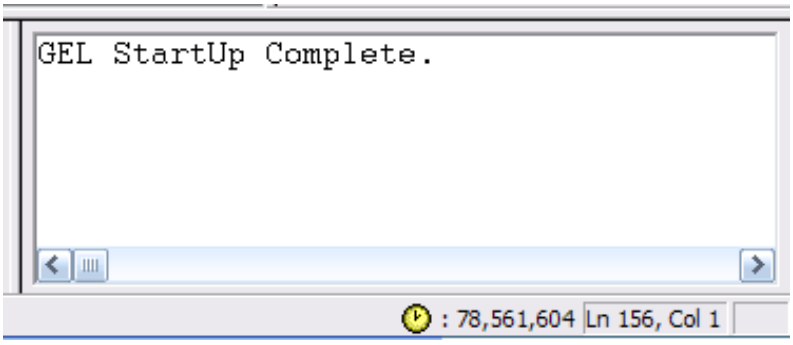
Nhận xét: ở dàn anten UCA, kết quả mô phỏng so với kết quả thực thi không khác nhau nhiều như dàn ULA. Không có thêm các đỉnh phụ, các mức ồn cũng không quá cao.

Đánh giá tốc độ thực thi trên kit:

Ở đây chỉ đánh giá tốc độ xử lý thuật toán, nên ta bỏ qua phần xử lý đọc dữ liệu của kit.

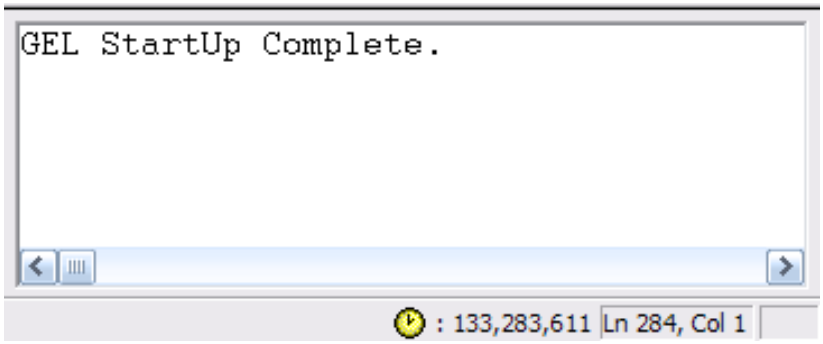
Thời gian xử lý thuật toán được tính bằng $\frac{\text{Số chu kỳ (cycle)chip đã thực hiện}}{\textit{Tốc độ của chip=225Mhz}}$

Đặt breakpoint ở cuối khối nhận dữ liệu ta nhận được 78.561.604 chu kỳ chip đã chạy



Hình 21: Số nhịp CPU của chip chạy đến khi kết thúc nhận dữ liệu

Tiếp tục đặt breakpoint ở đầu khối gửi dữ liệu, ta nhận được 133.283.611 chu kỳ.



Hình 22: Số nhịp CPU của chip chạy đến khi bắt đầu gửi dữ liệu

Thời gian xử lý thuật toán $t = \frac{133,283,611 - 78,561,604}{225 \times 10^6} = 0.216(s)$

Nhận xét: *Thời gian này là tương đối lớn so với kết quả mô phỏng, tuy nhiên việc thực thi trên chip có xung nhịp 225 MHz không thể so sánh với các hệ thống có xung nhịp lớn cỡ vài Ghz.*

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN CHO ĐỀ TÀI

Từ những phân tích về mặt lý thuyết toán học và vật lý, mô phỏng bằng chương trình MATLAB, khóa luận đã nghiên cứu thuật toán MUSIC cũng như mô tả thuật toán với các cấu trúc anten mảng sắp xếp theo đường thẳng cách đều, cấu trúc đường anten mảng sắp xếp theo đương tròn cách đều, từ đó đưa ra ưu nhược điểm của mỗi cấu trúc. Khóa luận cũng đưa ra mô hình thực thi thuật toán MUSIC trên kit TMS320C6713.

Mặc dù việc thực thi thuật toán trên kit có tốc độ khá chậm nhưng đây là điều có thể khắc phục được. Trong tương lai hệ thống có thể đưa vào áp dụng với các dữ liệu từ mảng anten trong thực tế và có thể tối ưu tốc độ xử lý để ứng dụng trong các hệ thống dò tìm, giám sát thời gian thực, với chi phí thấp và khả năng di động cao.

TÀI LIỆU THAM KHẢO

Tiếng Việt:

[1]GS. TSKH. Phan Anh, *Lý thuyết và kỹ thuật anten*, nhà xuất bản khoa học kỹ thuật, xuất bản 12- 2007.

[2]Nguyễn Đình Trí, Lê Trọng Vinh, Dương Thủy Vỹ, *Giáo trình toán học cao cấp*, xuất bản tháng 9 năm 2005, NXB Giáo Dục.

[3]Vũ Văn Yêm, Lâm Hồng Thạch, Phan Anh, "*Ứng dụng thuật toán MUSIC trong việc xác định vị trí tàu thuyền đánh cá loại vừa và nhỏ hoạt động ở vùng ven biển*", đề tài QC.06.19, Đại học Quốc Gia Hà Nội.

Tiếng Anh:

[4]A.W.Rudge, K.Milne, A.D.Olver, P.Knight, *The Handbook of antenna design*, volume 2, IEE Electromagnetic waves series 16, July. 1987

[5]Hamid Krim and Mats Viberg "*Two Decades of Array Signal Processing Research*", IEEE Signal Processing Magazine, July 1996.

[6]R.O.Schmidt,"*Multiple emitter location and signal parameter estimation*", pp.271-280, Mar 1986".

[7]Rulph Chassaing, *Digital Signal Processing and Applications with the C6713 and C6416 DSK*, Copyright© 2005 by John Wiley & Sons, Inc.

[8]Texas Instruments Incorporated, *TMS320C6713B Floating point digital signal processor*, SPRS294B, October 2005, revised June 2006.

[9]Texas Instruments Incorporated,*Real-Time Data Exchange*, SPRY012, Dallas, TX, 1998.

[10]Matlab Overview, http://www.mathworks.com/products/matlab/.

[11]William H.Press, Saul A.Teukolsky, William T.Vetterling, Brian P.Flannery, *NUMERICAL RECIPES, The Art of Scientific Computing*, 3rd Edition, Chap 11, Sep 2007.

PHỤ LỤC

1. Chương trình mô phỏng tín hiệu ở dàn anten ULA

```
Ne=8;                %So phan tu cua mang anten
Nb=1000;             %So mau tin hieu thu duoc
lamda=0.01;          %Buoc song cua tin hieu (m)
d=0.005;             %Khoang cach giua cac phan tu anten trong mang ULA (m)
%THAM SO NGUON TIN HIEU DEN [S]
D=4;                 %So nguon tin hieu
%Goc toi cua cac nguon tin hieu
angles=[20 40 60 80]*(pi/180);
SNRs=[25 25 25 25];
%Tao ma tran vecto dau vao tin hieu ban dau S[D,Nb] va ma tran vecto lai A(D,Ne)
for k=1:D
   S(k,:)=(20^(SNRs(k)/10))*exp(j*2*pi*rand(1,Nb));
   A(k,:)=exp(j*2*pi/lamda*((0:Ne-1)*d*cos(angles(k)))); %ULA normal
end
% Tao ma tran nhieu N[Nb,Ne]
N=rand(Nb,Ne)+j*rand(Nb,Ne);
% Tao ma tran du lieu thu duoc boi mang anten U[Nb,Ne]
U=S.'*A + N;
% Ghi ra file
   reU=real(U);
   imU=imag(U);
   fid = fopen('u.txt','w');
   fprintf(fid,'%d ',Ne );
   fprintf(fid,'%d ',Nb );
```

```
fprintf(fid,'%f ' ,reU');
fprintf(fid,'%f ' ,imU');
fclose(fid);
```

2. Chương trình thực thi thuật toán MUSIC

```
#include <stdio.h>
#include <math.h>
#include "a.h"
#define SIZE 2000
#define M 5000
#define PREC 0.001
short row,col,i,j,k,l,it,I0,L0,signal;
double delta,s,s0,t0,t1,w0;
typedef  struct //define struct
{ double R,I;
} COMPLEX;
typedef COMPLEX mat[SIZE][SIZE];
typedef double vec[SIZE];
COMPLEX c0,c1,c2,c3,u0,u1,z0,z1,temp;
mat ruu,evector,b,aa,aat,c,power;
vec evalue;
mat u,ut;
void ADD(COMPLEX c1, COMPLEX c2, COMPLEX *c3){
    c3->R=c1.R+c2.R;
    c3->I=c1.I+c2.I;}
void SUB(COMPLEX c1, COMPLEX c2, COMPLEX *c3){
    c3->R=c1.R-c2.R;
    c3->I=c1.I-c2.I;}
void MUL(COMPLEX c1, COMPLEX c2, COMPLEX *c3){
    c3->R=c1.R*c2.R - c1.I*c2.I;
    c3->I=c1.R*c2.I + c1.I*c2.R;}
void RMUL(double alpha, COMPLEX c, COMPLEX *c1){
    c1->R=alpha*c.R;
    c1->I=alpha*c.I;}
double SQR(double a)
```

```
{   return a*a;}
double ABS(COMPLEX c)
{   return sqrt(SQR(c.R)+SQR(c.I));}
void CONJ(COMPLEX c, COMPLEX *c1){
    c1->R=c.R;
    c1->I=-c.I;}
void CDIV(double AR, double AI, double BR, double BI, double *ZR, double *ZI)
{
 double YR,YI,W;
 YR=BR;
 YI=BI;
 if(fabs(YR) > fabs(YI)){
    W=YI/YR;
    YR=W*YI+YR;
    *ZR=(AR+W*AI)/YR;
    *ZI=(AI-W*AR)/YR;}
 else {
   W=YR/YI;
   YI=W*YR+YI;
   *ZR=(W*AR+AI)/YI;
   *ZI=(W*AI-AR)/YI;
 }
}
void transpose(mat a,short cola,short rowa, mat at) //chuyen vi lien hop
{   for (i=1; i<=cola; i++)
   {        for (j=1; j<=rowa; j++)
      {     at[j][i].R=a[i][j].R;
                    at[j][i].I=-a[i][j].I;
      }
   }
}
void mul(mat a,short cola,short rowa, mat b, short rowb, mat c,short fac)
{
    for (j=1; j<=rowb; j++)
    {       for (i=1; i<=cola; i++)
```

```
            {       temp.R=0;
                    temp.I=0;
                    c[i][j].R=0;
                    c[i][j].I=0;
                    for (k=1;k<=rowa;k++)
                    {       MUL(a[i][k],b[k][j],&temp);
                            ADD(temp,c[i][j],&c[i][j]);
                    }
                    c[i][j].R=c[i][j].R/fac;
                    c[i][j].I=c[i][j].I/fac;
            }
    }
}
void sorteigen(int N, vec R, mat Z)
{   vec Tr,Ti;
    double VR;
//  sort in ascending order
    for (j=2; j<=N; j++){
            VR=R[j];
            for (k=1; k<=N; k++){
                    Tr[k]=Z[k][j].R;
                    Ti[k]=Z[k][j].I;
            }
            for (i=j-1; i>0; i--){
                    if (fabs(R[i]) <= fabs(VR)) goto e5;
                    R[i+1]=R[i];
                    for (k=1; k<=N; k++){
                            Z[k][i+1].R=Z[k][i].R;
                            Z[k][i+1].I=Z[k][i].I;
                    }
            }
            i=0;
e5:         R[i+1]=VR;
            for (k=1; k<=N; k++){
                    Z[k][i+1].R=Tr[k];
```

```
                    Z[k][i+1].I=Ti[k];
                }
            }
        }
        void output(mat a,int row)
        {
            FILE *p;
            p = fopen("eigenvector.txt","wt");
            fprintf(p,"%d",row);
            for (j=1; j<=row; j++){
                fprintf(p," %f ", a[1][j].R);
                }
                fprintf(p,"\n");
            fclose(p);
        }
        int main()
        {
            FILE *p;
            p = fopen("u.txt","r");
            fscanf(p,"%hd %hd",&row,&col);
            for (i=1; i<=col; i++)
                for (j=1; j<=row; j++)
                    fscanf(p, "%lf", &u[i][j].R);
            for (i=1; i<=col; i++)
                for (j=1; j<=row; j++)
                    fscanf(p, "%lf", &u[i][j].I);
            transpose(u,col,row,ut);
            mul(ut,row,col,u,row,ruu,row);
        //begin of eigen
            z0.R=0.0;z0.I=0.0;
            z1.R=1.0;z1.I=0.0;
            for(i=1; i<=row; i++)
                for (j=1; j<=row; j++)
                    if (i==j)
                        evector[i][j]=z1;
```

```
                else evector[i][j]=z0;
        it=-1; l=1;
        while (l<=M && it!=1)
        {
                s=0.0;
                for (i=1; i<row; i++)
                        for (j=i+1; j<=row; j++)
                        {
                                t0=ABS(ruu[i][j]);
                                if (t0 > s)
                                {
                                        s=t0; I0=i; L0=j;
                                }
                        }
                if (s==0.0) it=1;
                else
                {
                        delta=SQR(ruu[L0][L0].R-
ruu[I0][I0].R)+4.0*SQR(ABS(ruu[I0][L0]));
                        t0=ruu[L0][L0].R-ruu[I0][I0].R + sqrt(delta);
                        t1=ruu[L0][L0].R-ruu[I0][I0].R - sqrt(delta);
                        if (fabs(t0) >= fabs(t1))
                                w0=t0;
                        else w0=t1;
                        s0=fabs(w0)/sqrt(SQR(w0)+4.0*SQR(ABS(ruu[I0][L0])));
                        t0=2.0*s0/w0;
                        RMUL(t0,ruu[I0][L0],&c0);
                        CONJ(c0,&c1);
                        for (i=1; i<I0; i++){
                                u0=ruu[i][I0];
                                MUL(c0,u0,&c2);
                                RMUL(s0,ruu[i][L0],&c3);
                                ADD(c2,c3,&ruu[i][I0]);
                                MUL(c1,ruu[i][L0],&c2);
                                RMUL(s0,u0,&c3);
```

```
            SUB(c2,c3,&ruu[i][L0]);
        }

        for (k=I0+1; k<L0; k++) {
            u0=ruu[I0][k];
            MUL(c1,u0,&c2);
            CONJ(ruu[k][L0],&u1);
            RMUL(s0,u1,&c3);
            ADD(c2,c3,&ruu[I0][k]);
            MUL(c1,ruu[k][L0],&c2);
            CONJ(u0,&u1);
            RMUL(s0,u1,&c3);
            SUB(c2,c3,&ruu[k][L0]);
        }
        for (j=L0+1; j<=row; j++)
        {
            u0=ruu[I0][j];
            MUL(c1,u0,&c2);
            RMUL(s0,ruu[L0][j],&c3);
            ADD(c2,c3,&ruu[I0][j]);
            MUL(c0,ruu[L0][j],&c2);
            RMUL(s0,u0,&c3);
            SUB(c2,c3,&ruu[L0][j]);
        }
        t0=ruu[I0][I0].R;
        t1=4.0*SQR(s0*ABS(ruu[I0][L0]))/w0;
        ruu[I0][I0].R=SQR(ABS(c0))*t0 + t1+SQR(s0)*ruu[L0][L0].R;
        ruu[L0][L0].R=SQR(s0)*t0 - t1+SQR(ABS(c0))*ruu[L0][L0].R;
        ruu[I0][L0]=z0;
        for (i=1; i<=row; i++)
        {
            u0=evector[i][I0];
            MUL(c0,u0,&c2);
            RMUL(s0,evector[i][L0],&c3);
            ADD(c2,c3,&evector[i][I0]);
```

```
                        MUL(c1,evector[i][L0],&c2);
                        RMUL(s0,u0,&c3);
                        SUB(c2,c3,&evector[i][L0]);
                }
                t0=0.0;
                for (i=1; i<row; i++)
                        for (j=i+1; j<=row; j++)
                                t0 += SQR(ABS(ruu[i][j]));
                s=2.0*t0;
                if (s < PREC)
                        it=1;
                else l++;
        }
    }
    if (it==1)
        for (i=1; i<=row; i++)
                evalue[i]=ruu[i][i].R;
//End of eigen*/
    sorteigen(row, evalue, evector);
   signal=0;
   for (i = 0; i<=row;i++)
    {
        if(evalue[i]>evalue[row]/1000)  signal++;
    }
   I0=row-signal;
    transpose(evector,row,I0,ruu);
   mul(evector,row,I0,ruu,row,b,1);
    l=0;
    for (signal=1;signal<=p2;signal++){
        for (k=1;k<=row;k++){
                aa[1][k].R=areal[l];
                aa[1][k].I=aimag[l];
                l++;
        }
        mul(aa,1,row,b,row,c,1);
```

```
        c2.R=0.0;c2.I=0.0;
        for (k=1;k<=row;k++){
                        CONJ(aa[1][k],&c0);
            MUL(c[1][k],c0,&c1);
                        ADD(c1,c2,&c2);
              }
              CDIV(row,0,c2.R,c2.I,&power[1][signal].R,&power[1][signal].I);
     }
     output(power,p2);
    return 0;
}
```