TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN ĐIỆN TỬ - VIỄN THÔNG

BÀI TẬP LỚN

MÔN HỆ THỐNG VIỄN THÔNG

Đề tài:

Thiết kế hệ thống thu phát 16-QAM trên FPGA

Giảng viên hướng dẫn: PGS.TS. VŨ VĂN YÊM

Nhóm thực hiện:

TRƯƠNG VĂN BÌNH	SHSV:20090254	ĐT 11 – K54
HÀ VĂN PHÚ	SHSV:20092021	ĐT 04 – K54
TRẦN THỊ THANH HUYỀN	SHSV:20091272	ĐT 02 – K54
NGUYỄN THANH TÙNG	SHSV:20093128	ĐT 04 – K54

Hà Nội, 11-2012

MỤC LỤC

Phần 1. Giới thiệu ý tưởng và xác định chỉ tiêu kỹ thuật của sản phẩm

1.1 Nhu cầu và sự cần thiết của thuật toán

Hệ thống thông tin số hiện nay đang sử dụng mã hóa QPSK là chủ yếu, nhưng tài nguyên tần số là có hạn cần đưa vào áp dụng các phương pháp mã hóa mới tận dụng tối đa dung lượng cũng như tốc độ kênh truyền.

Bài báo này đưa ra phương pháp mã hóa 16-QAM, Góp phần vào việc phát triển sử dụng thuật toán 16-QAM trong mã hóa các hệ thống hiện tại.Bài viết trình bày một thiết kế hoàn chỉnh cho một máy Phát-Thu 16-QAM trên Kit FPGA spartan-6.

1.2 Các công trình bài báo nghiên cứu đã có và có liên quan

Hiện nay ở việt Nam và trên thế giới đều đã có nhóm cùng làm về đề tài System 16-QAM Transmitter and Receiver Design Based on FPGA. Thiết kế đều tập trung trên kit Virtex4 FPGA.Sử dụng dải tần IF 12MHz cho kết quả rất khả quan.

Bài viết này chúng em có điểm mới là linh hoạt sử dụng thuật toán CORDIC trong việc khôi phục sóng mang và xoay pha Chòm Sao. Giúp tăng mạnh mẽ tốc độ tính toán và tiết kiệm tài nguyên bộ nhớ. Hệ thống sử dụng phương thức giao tiếp Mic-Loa môi trường truyền sóng là không gian tự do tần số sóng mang f = 10MHz, không có vật cản.

Ngoài ra bài viết đáp ứng được tất cả các yêu cầu cơ bản của một hệ thống Phát-Thu 16-QAM.

Các module được test trên Modelsim cho kết quả chính xác. Mô phỏng thực hiện trên Simulink của Matlab. Vì lí do khách quan nên kết quả BER thu được còn khiêm tốn.

1.3 Tổng Quan Sơ Lược Về QAM

1.3.1. Tổng quan về QAM

- QAM là dạng điều chế mà thông tin được chứa cả trong biên độ và pha của sóng mang được truyền. Các trạng thái thường gặp của điều chế QAM là 4-QAM, 16-QAM, 64-QAM, 256-QAM..

❖ Điều chế QAM

Một tín hiệu điều chế biên độ vuông góc QAM sử dụng hai sóng mang vuông góc là $\cos 2\pi f_c t$ và $\sin 2\pi f_c t$, mỗi sóng mang được điều chế bởi một chuỗi độc lập các bit mang thông tin. Các sóng tín hiệu truyền đi có dạng

$U_m(t)=A_{mc}g_T(t)\cos 2\pi f_c t + A_{ms}g_T(t)\sin 2\pi f_c t \qquad m=1,2\ldots,M$

Trong đó $\{A_{mc}\}$ và $\{A_{ms}\}$ là tập các mức biên độ nhận được bằng cách ánh xạ các chuỗi k bit thành các biên độ tín hiệu. Tổng quát hơn, QAM có thể xem như một dạng hỗn hợp của điều chế biên độ số và điều chế pha số.

1.3.2. Giải điều chế và tách tín hiệu QAM

Giả sử rằng một lượng dịch pha sóng mang được đưa vào trong quá trình truyền dẫn tín hiệu. Thêm vào đó tín hiệu thu được bị nhiễu loạn bởi tạp âm cộng Gauss, vì vậy tín hiệu thu được r(t) có thể biểu diễn là :

$$R(t)=A_{mc}g_T(t)\cos(2\pi f_c t+\Theta) + A_{ms}g_T(t)\sin(2\pi f_c t+\Theta)+n(t)$$

Trong đó Θ là lượng dịch pha của sóng mang và $n(t)=n_c(t)\cos 2\pi f_c t - n_s(t)\sin 2\pi f_c t$

Như vậy giải điều chế QAM thực hiện ước lượng dịch pha sóng mang Θ của tín hiệu thu được , bù lại lượng dich pha này và khôi phục tín hiệu gần giống với tín hiệu ở bộ phát đã phát đi

❖ Đặc điểm của tín hiệu QAM.

Tín hiệu QAM là sự kết hợp của điều chế biên độ FSK và điều chế pha PSK do đó nó mang các đặc điểm của FSK và PSK , ngoài ra có một số đặc điểm khác như:

- Số mức biên độ hoặc pha của sóng mang trong điều chế PSK và FSK càng lớn cho phép mang nhiều thông tin hơn, nhưng số lượng này bị giới hạn do nhiễu kênh truyền. Số mức càng tăng kéo theo độ phức tạp trong mạch điều chế và giải điều chế cũng tăng.
- Với điều chế n-PSK sóng mang truyền đồng thời N bit thông tin. Số lượng pha cần có 2^n , n tăng làm cho độ lệch giữa hai pha kế tiếp là $2\pi/2^n$ giảm rất nhanh do đó rất dễ bị nhiễu tác động làm lỗi bit. Đối với những hệ thống dùng

hơn 4 bit để truyền thông tin thì người ta thường dùng QAM thay cho PSK vì xác suất lỗi thấp hơn và khả năng kháng nhiễu tốt hơn.

1.3.3. Chỉ tiêu kỹ thuật của thuật toán.

1.3.3.1. Khối phát

Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (10MHz)
Rst_n	1 bit	Input	Reset
Data_in	1 bit	Input	Dữ liệu vào
Ce	1 bit	Input	Cho phép mạch hoạt động
Data_out	16bit	Output	Dữ liệu ra sau khi đã được xử lý. Trong đó có 4 bit cao biểu diễn phần thập phân, 12 bit thấp biểu diễn phần sau dấu phẩy.
Ready	1 bit	Output	Báo hiệu có dữ liệu ra

1.3.3.2. Hàm truyền đạt của khối phát

Tín hiệu vàosẽ được xử lý qua các bước sau:

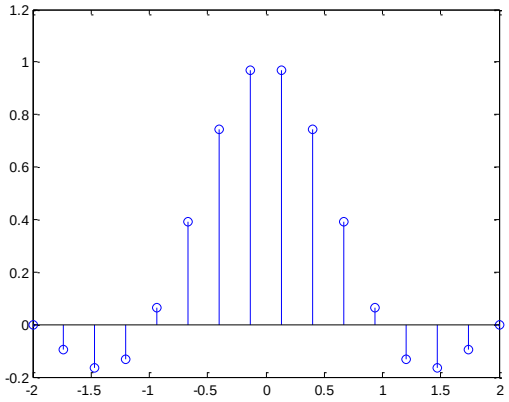
- ✓ **Bước 1 – Cyclic prefix**: Có chức năng đóng gói dữ liệu. Cứ sau mỗi 90bit đi vào khối, thì dữ liệu sẽ được chèn thêm 10 bit 1. Đầu ra của khối này là 1bit.
- ✓ **Bước 2 – Mapper:** Có chức năng mã hóa dữ liệu theo bảng sau:

b_0b_1	I	b_2b_3	Q
00	-3	00	-3
01	-1	01	-1
11	+1	11	+1
10	+3	10	+3

Khối này sẽ tách 4bit liên tiếp thành 2 phần (Phần I và phần Q) sau đó tín hiệu sẽ được ánh xạ 1:1 thành các mức biên độ -3, -1, 1, 3. Khi này tín hiệu được chia thành 2 kênh: Kênh I và kênh Q

Biểu diễn các mức biên độ bằng số 16bit, trong đó có 12bit sau dấu phẩy

✓ **Bước 3 – Bộ lọc cos nâng:** Tín hiệu đi qua bộ lọc cos nâng



✓ **Bước 4 – Sóng mang:** Đối với kênh I, tín hiệu được nhân với sóng mang sin. Đối với sóng mang Q, tín hiệu được nhân với sóng mang cos. Sau đó 2 kênh này sẽ được cộng điểm – điểm với nhau, và tạo ra 1 đường dữ liệu 16bit.

✓ **Bước 5 – Chèn Pilot:** Dùng để đồng bộ tín hiệu nhận được bên thu. Cứ sau 25 mẫu sóng mang, tín hiệu lại được chèn thêm 10 số 7 (Mỗi số 7 được biểu diễn bằng 4 bit phần thập phân và 12 bit sau dấu phẩy). Đầu ra của khối này là đầu ra cuối cùng của bộ phát, chúng ta phát tín hiệu này qua anten.

1.3.3.3. Khối thu

Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (10MHz)
Rst_n	1 bit	Input	Reset
Data_in	16 bit	Input	Dữ liệu ra sau khi đã được xử lý. Trong đó có 4 bit cao biểu diễn phần thập phân, 12 bit thấp biểu diễn phần sau dấu phẩy.
start	1 bit	Input	Cho phép mạch hoạt động
Data_out	1 bit	Output	Dữ liệu ra sau khi đã được xử lý
Ready	1 bit	Output	Báo hiệu có dữ liệu ra

1.3.3.4. Hàm truyền đạt của khối thu

Tín hiệu thu được gồm 16 bit, trong đó có 12bit sau dấu phẩy. Tín hiệu này được xử lý qua các bước sau:

- ✓ **Bước 1 – Đồng bộ theo thời gian:** dữ liệu thu được từ Mic sẽ được cắt khoảng có độ dài bằng 2 lần chiều dài của 1 khung dữ liệu.

  Sử dụng 2 cửa Sổ Trượt có độ rộng 10 mẫu cách nhau 400 mẫu, cho 2 cửa sổ chạy, lấy tích tương quan giá trị của 2 cửa sổ đó và lưu lại trong thanh ghi. Khi cửa sổ trượt hết khoảng dữ liệu vừa cắt ta so sánh và tìm max trong các giá trị đó. Nếu max thu được chưa thỏa mãn ngưỡng như yêu cầu thì công việc sẽ lặp lại từ đầu cho đến thì tìm được max.

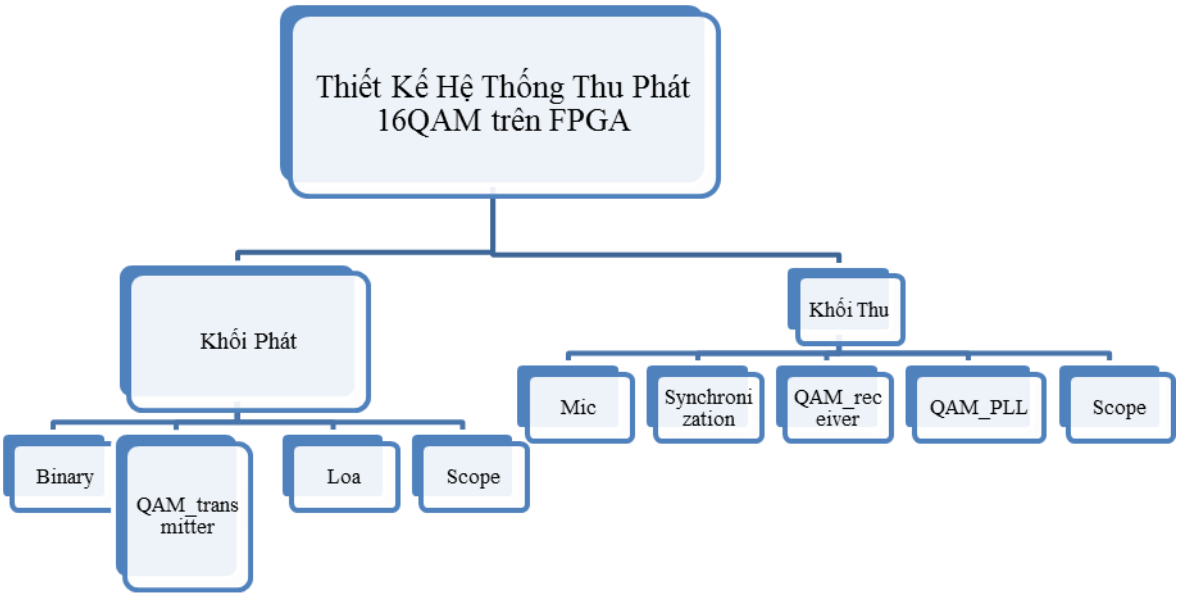
  Từ giá trị max thu được ta dịch trở lại 10 mẫu chính là điểm đầu của khung dữ liệu, cũng chính là điểm bắt đầu phần **Pilot.**
- ✓ **Bước 2 – Cắt Pilot:** Sauphần đồng bộ ta bắt được điểm bắt đầu **Pilot**. Bước này có nhiệm vụ cắt đi tất cả các phần **Pilot** của các khung dữ liệu.
- ✓ **Bước 3 – Nhân Sóng mang:** Tín hiệu sau khi cắt **Pilot** được đưa qua 2 đường nhân với bộ sin và cos để hình thành 2 kênh dữ liệu I và Q
- ✓ **Bước 4 – Bộ lọc cos nâng:** Bộ lọc này đóng vai trò là bộ lọc thông thấp, lọc đúng dạng của dữ liệu
- ✓ **Bước 5 – Phase Lock Loop:** Khôi phục pha của tín hiệu, dựa vào việc hồi tiếp các tín hiệu.(Adapter filter)d
- ✓ **Bước 6 – Demapper:** Đối chiếu tín hiệu thu được trên 2 kênh I và Q với bảng mapper thu được dòng bit thông tin.

1.3.4. Các yêu cầu phi chức năng

Không tính tới

Phần 2. Phân tích chức năng và lập kế hoạch

2.1 Phân tích chức năng



Hình 2.1. Sơ đồ chức năng của sản phẩm

2.1.1. Khối Phát

Đầu vào là dạng Binary, đầu ra là tín hiệu kiểu int16 bao gồm 4 bit phần nguyên và 12bit biểu diễn phần thập phân.

2.1.2. Khối Thu

Đầu vào là tín hiệu int16 thu được từ Mic và đầu ra là 2 dòng bit trên 2 kênh I và Q. Dòng bit này được biểu diễn trên đồ thị chòm sao.

2.2 Lập kế hoạch

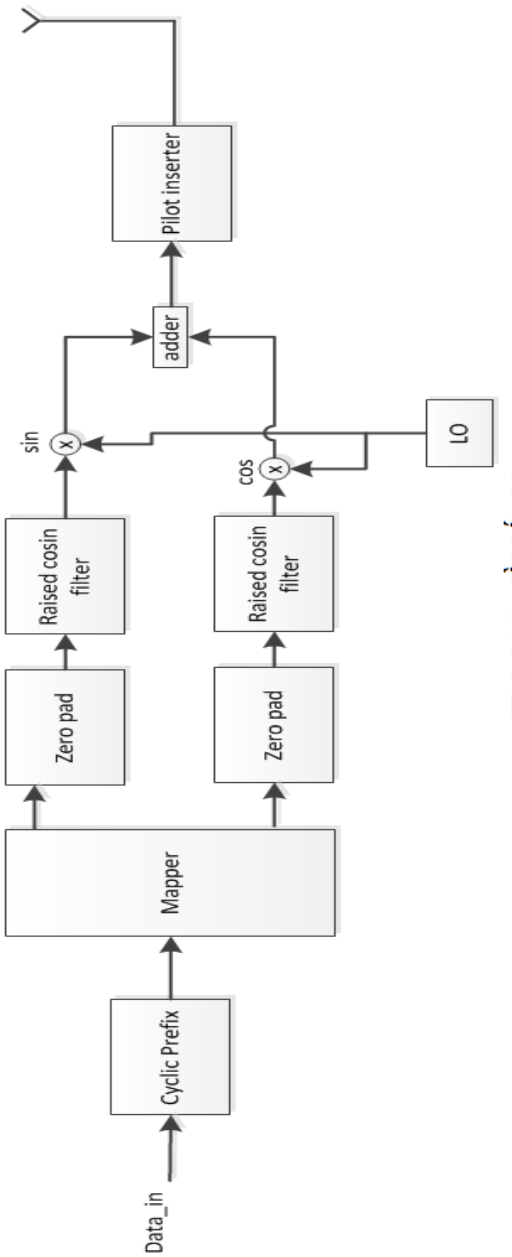
Mục	Công việc	Người thực hiện	Bắt đầu	Kế hoạch kết thúc	Ngày kết thúc
1.	Viết verilog cho các khối cơ bản phần phát, phần thu: bộ sóng mang, bộ chèn không, cyclic prefix, lọc, mapper,	Huyền	01/10	20/11	20/11

	pilot, cắt pilot,...				
2	Viết specification, viết verilog cho khối PLL và phân công công việc.	Phú	01/10	20/11	20/11
3.	Thuật toán và Simulink.	Bình	01/10	20/11	20/11
4.	Tổng hợp synopsys.	Tùng	01/10	20/11	20/11

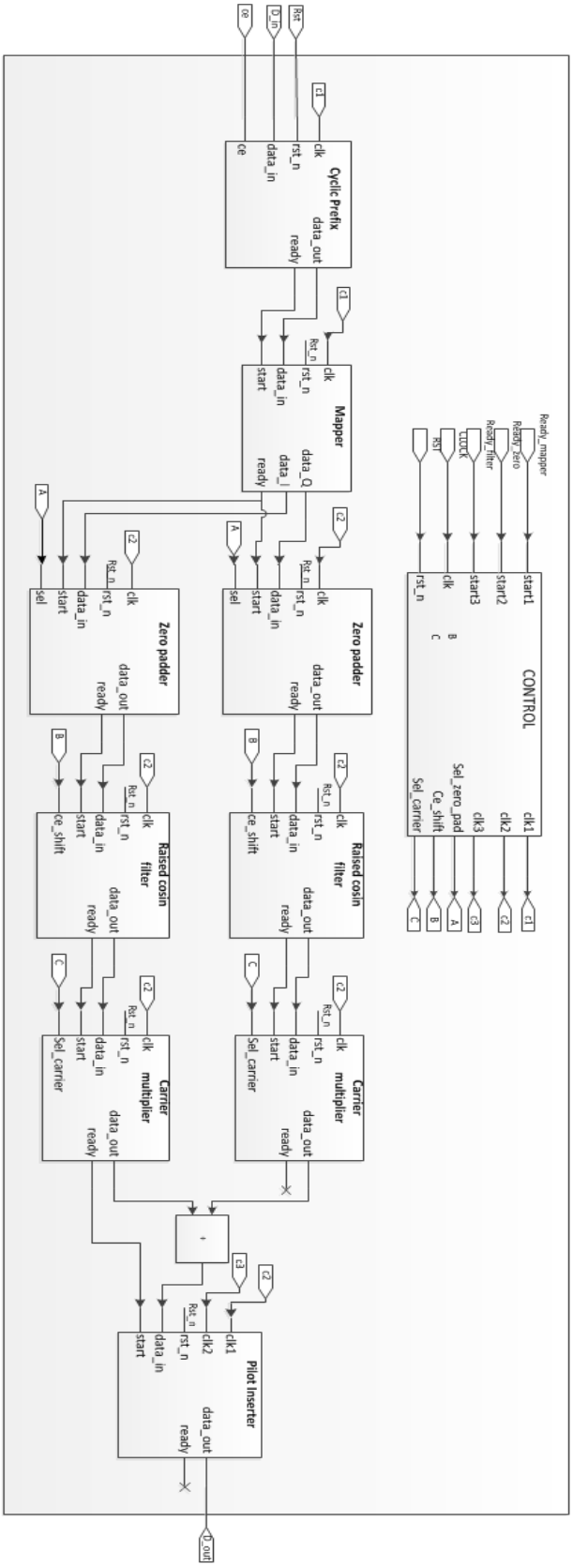
Phần 3. Lựa chọn phương án kỹ thuật

3.1 Sơ đồ khối của sản phẩm

3.1.1 Sơ đồ khối phát:



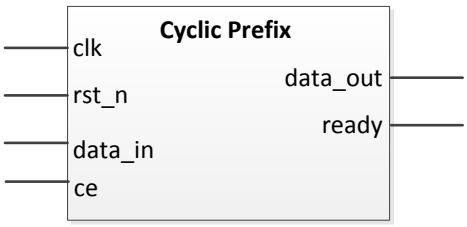
Hình 3.1 Sơ đồ khối phát



Hình 3.2 Sơ đồ ghép nối khối phát

a. Cyclic Prefix

Block diagram



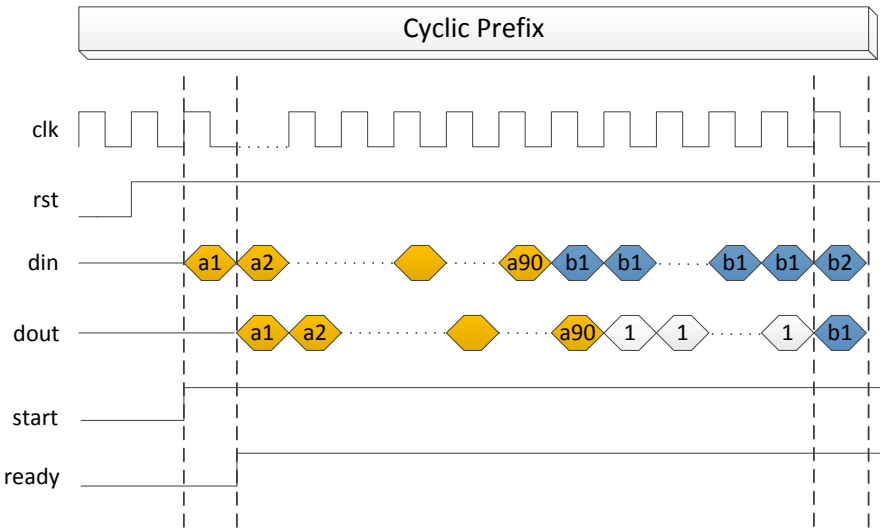
I/O port

Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (2.5MHz)
Rst_n	1 bit	Input	Reset
Data_in	1 bit	Input	Dữ liệu vào
ce	1 bit	Input	Cho dữ liệu vào
Data_out	1bit	Output	Dữ liệu ra
Ready	1 bit	Output	Báo hiệu dữ liệu ra

Operation

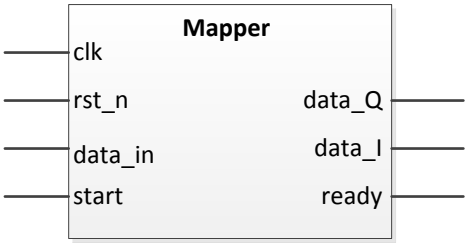
- Dữ liệu vào sau 90 bit sẽ được chèn khoảng bảo vệ 10 bit ‘1’.
- Cần 1 thanh ghi-xbit lựa chọn giữa 90 bit đầu vào và 10 bit bảo vệ ‘1’
- Ban đầu gán tín hiệu vào cho 1 dây dẫn A (wire), cùng lúc đó bật Start từ 0 lên 1
- Khi có sườn lên của xung clk, đầu ra bằng với giá trị dây dẫn A
- Khi có dữ liệu ra, tín hiệu Ready sẽ chuyển từ 0 lên 1

Waveform



b. Mapping

Block diagram



I/O port

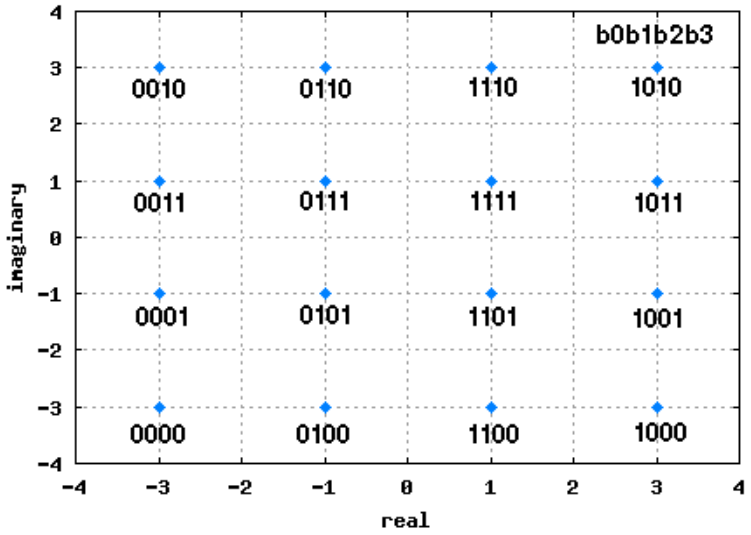
Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (2.5MHz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Data_in	1 bit	Input	Dữ liệu vào
Data_I	16 bit	Output	Dữ liệu phần thực (4 bit thập phân trong tổng số 16 bit)
Data_Q	16 bit	Output	Dữ liệu phần ảo(4 bit thập phân trong tổng số 16 bit)
Ready	1 bit	Output	Báo hiệu dữ liệu ra

Operation

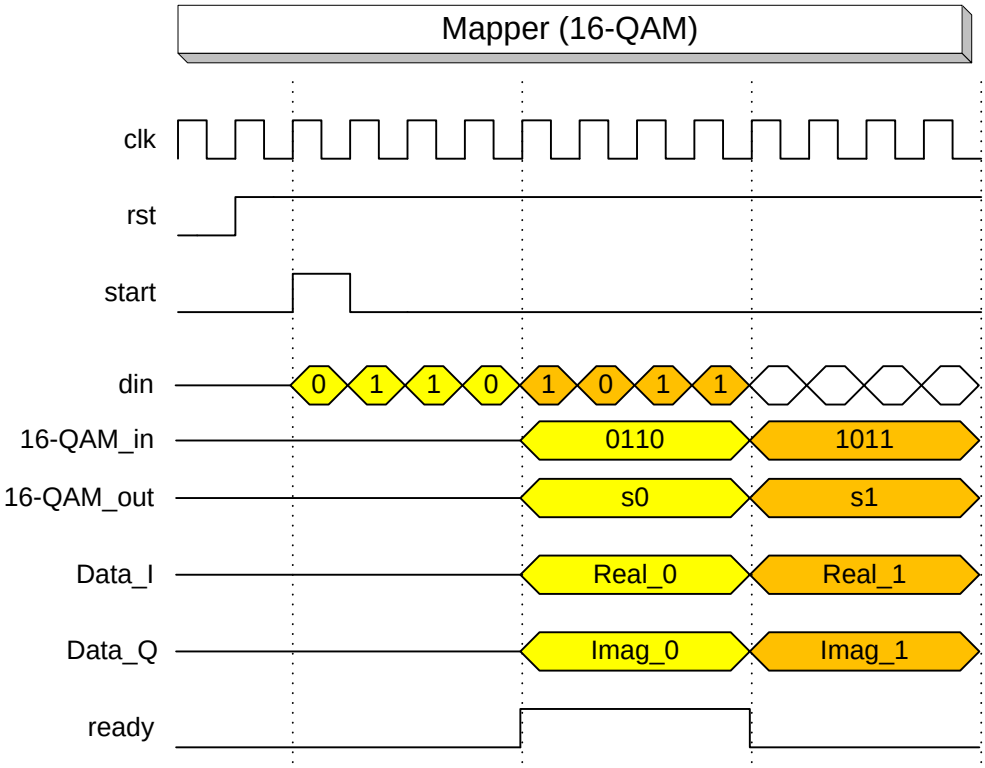
- Khi start =1 gán dữ liệu vào cho 1 dây dẫn để xử lý trên dây dẫn đó
- Mã hóa

b_0b_1	I	b_2b_3	Q
00	-3	00	-3
01	-1	01	-1
11	+1	11	+1
10	+3	10	+3

- Chòm sao điều chế

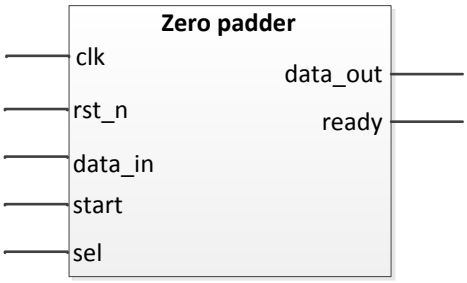


Waveform



c. Zero padder

Block diagram

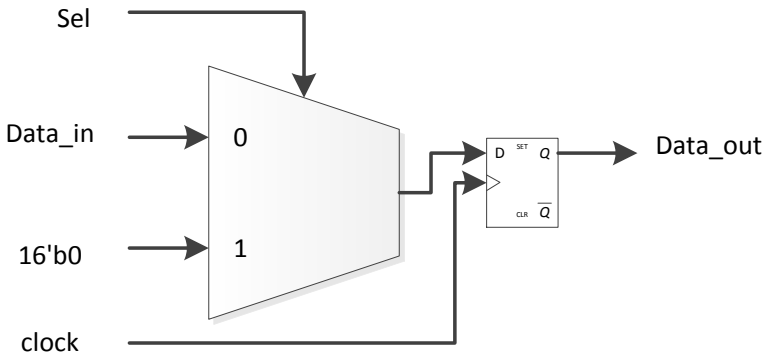


I/O port

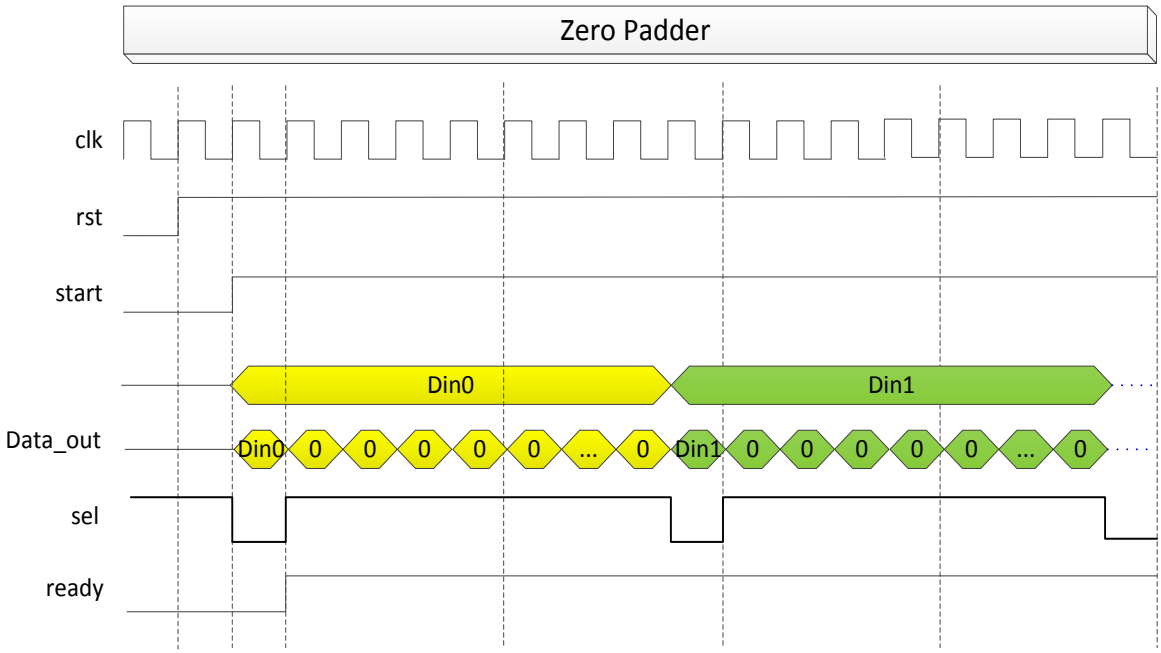
Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock(10Mhz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Data_in	16 bit	Input	Dữ liệu vào
Sel	1 bit	Input	Tín hiệu chọn giữa đầu vào và số 0-16bit
Data_out	16 bit	Output	Dữ liệu ra
Ready	1 bit	Output	Báo hiệu dữ liệu ra

Operation

- Chèn 15 số 0 vào giữa các phần tử
- Khi có tín hiệu start, đưa dữ liệu vào vào khối (dây A)
- Sel lựa chọn giữa dây A và 16'b0
- Tại sườn dương của clk, gán tín hiệu được chọn cho đầu ra
- Tín hiệu sel là tín hiệu điều khiển có 1 bit:
  - Sel = 0: chọn data_in
  - Sel= 1: chọn 16'b0

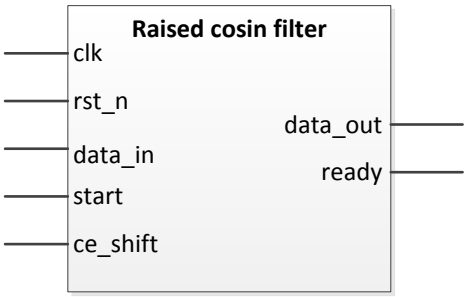


Waveform


Zero Padder
clk
rst
start
Din0
Din1
Data_out
Din0
0
0
0
0
0
...
0
Din1
0
0
0
0
0
...
0
sel
ready

d. Raised cosin filter

Block diagram

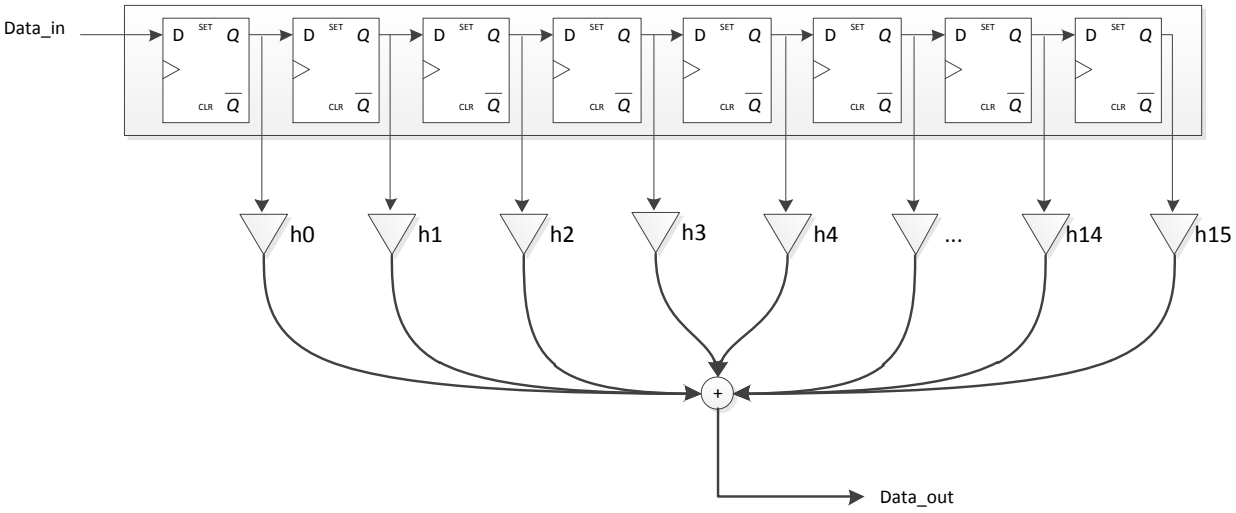


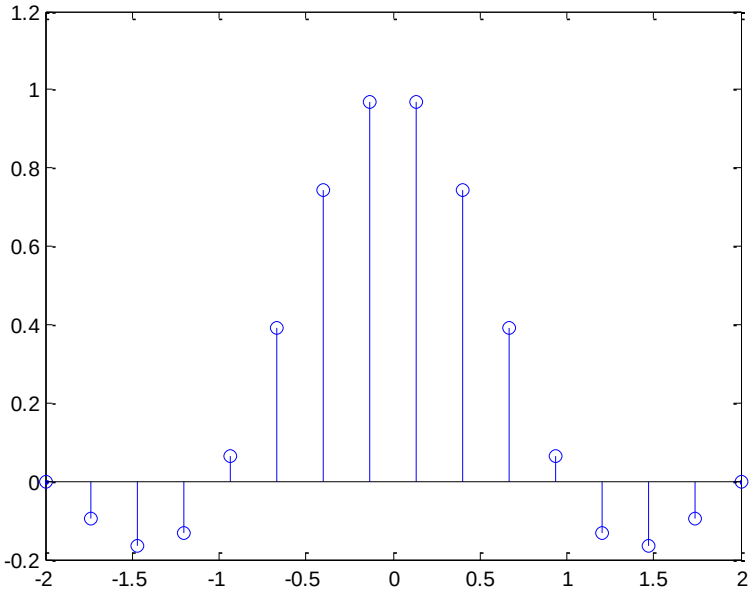
I/O port

Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (10MHz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Data_in	16 bit	Input	Dữ liệu vào
Ce_shift	1 bit	Input	Ce_shift=1 : cho phép dịch bit
Data_out	16 bit	Output	Dữ liệu ra
Ready	1 bit	Output	Báo hiệu dữ liệu ra

Operation

- Khi Ce_shift = 1: cho phép thanh ghi dịch hoạt động
- Tín hiệu được dịch vào và được nhân với 1 bảng gồm 16 giá trị- là giá trị của các đáp ứng xung





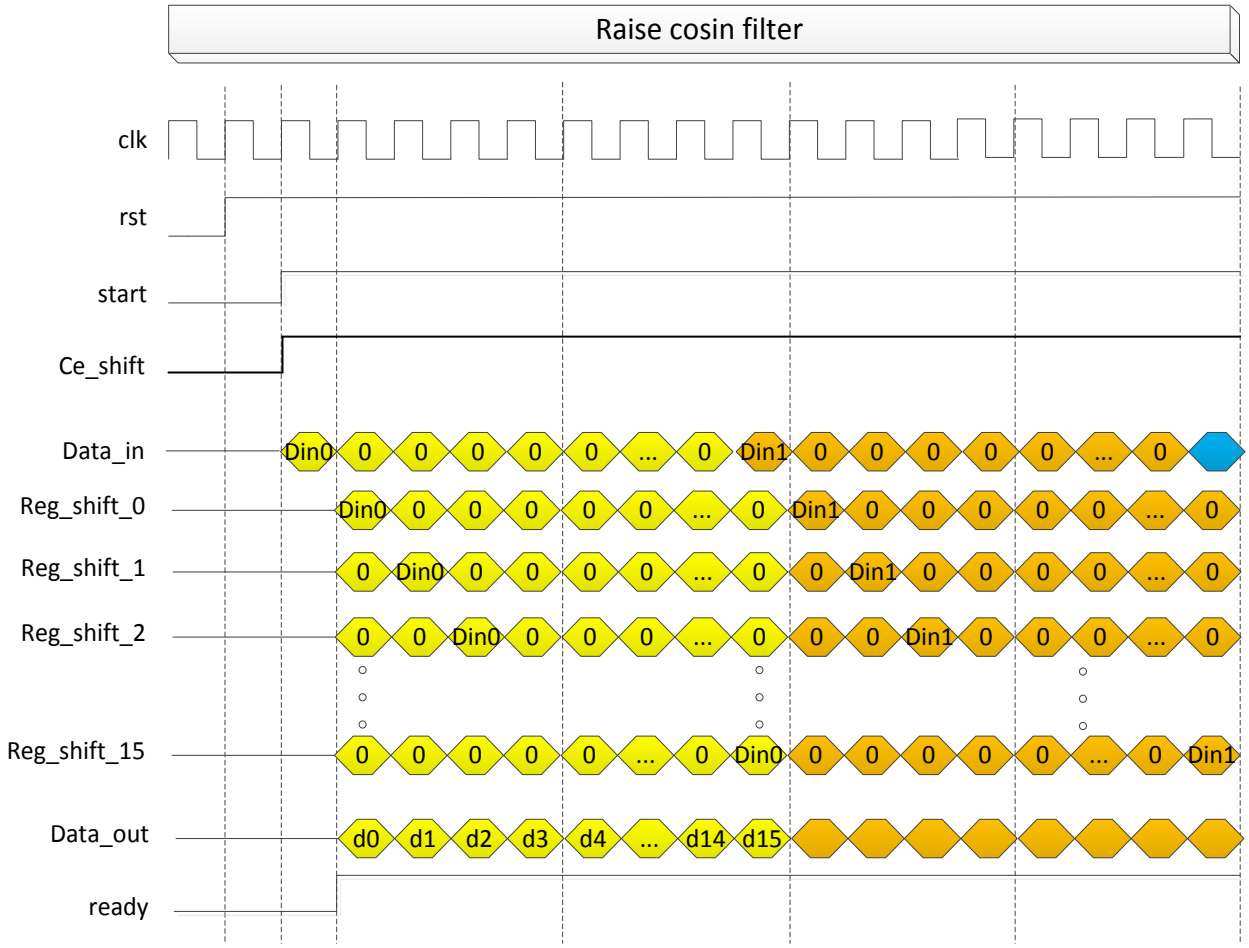
Đáp ứng xung – 16 điểm

- Bảng giá trị của đáp ứng xung:

##	Giá trị
h0	0.0000
h1	-0.0951
h2	-0.1672
h3	-0.1317
h4	0.0641
h5	0.3928
h6	0.7431
h7	0.9690
h8	0.9690
h9	0.7431
h10	0.3928
h11	0.0641
h12	-0.1317
h13	-0.1672
h14	-0.0951
h15	0.0000

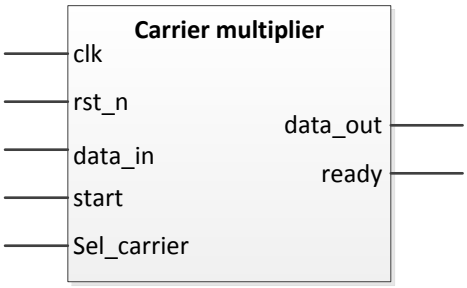
Bảng 1: Giá trị của đáp ứng xung

Waveform


Raise cosin filter
clk
rst
start
Ce_shift
Data_in
Reg_shift_0
Reg_shift_1
Reg_shift_2
Reg_shift_15
Data_out
ready
Din0
Din1
d0
d1
d2
d3
d4
d14
d15

e. Carrier multiplier

Block diagram

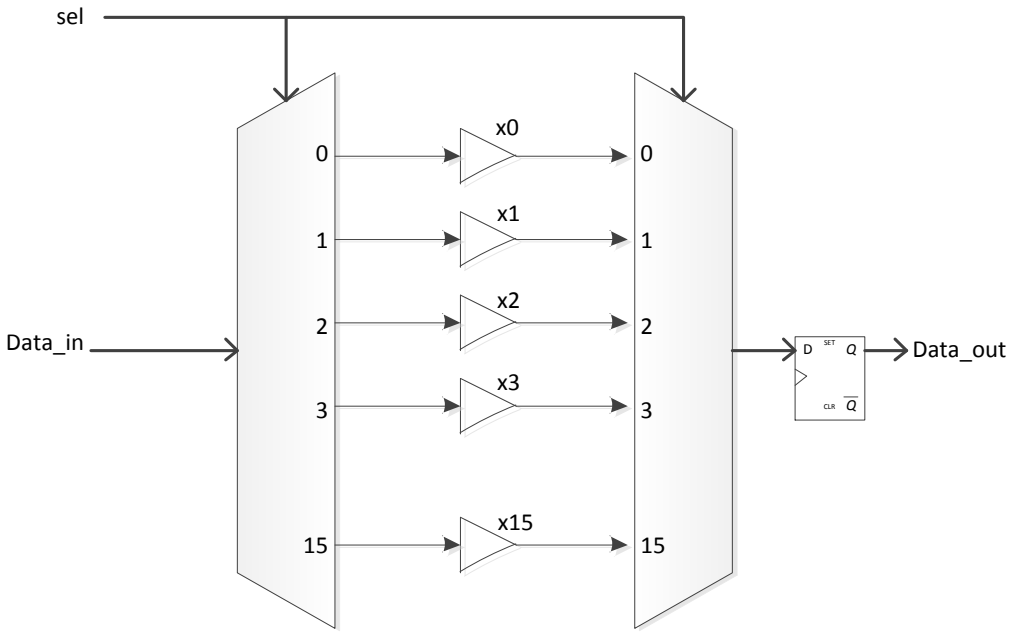


I/O port

Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (10MHz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Sel_carrier	4 bit	Input	Lựa chọn nhân hằng số
Data_in	16 bit	Input	Dữ liệu vào
Data_out	16 bit	Output	Dữ liệu ra
Ready	1 bit	Output	Báo hiệu dữ liệu ra

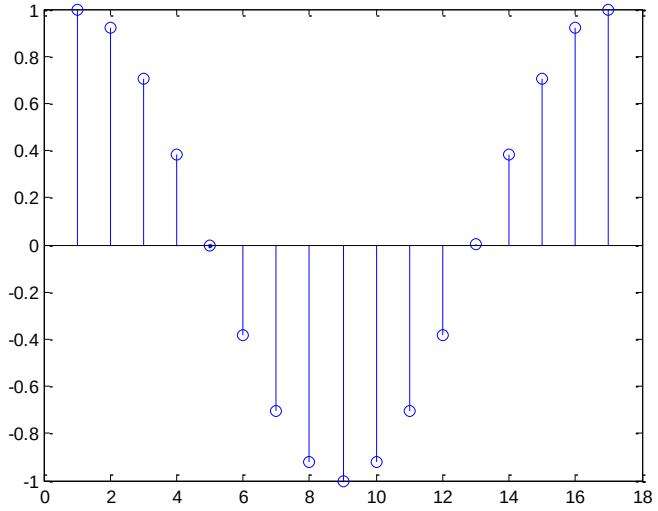
Operation

- Sóng mang là cosin hoặc sin sẽ được cắt ra thành 16 điểm với giá trị như trong bảng 2
- Sơ đồ:

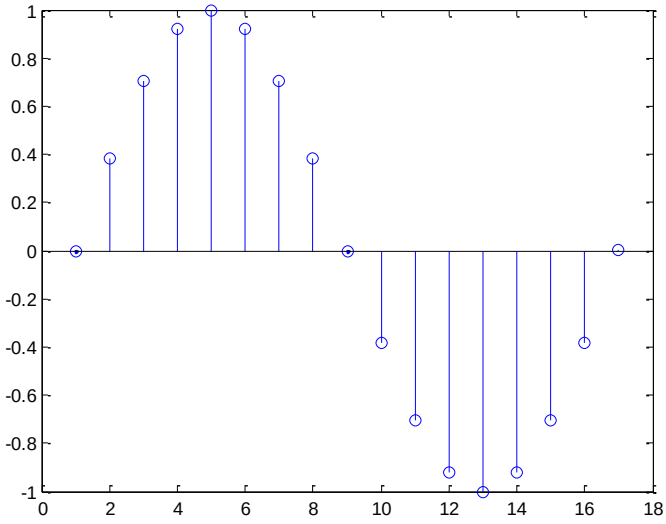


Với x0..15 là giá trị của sin hoặc cos đã được lấy mẫu

Sóng mang sin và cos được chia thành 16 điểm



Sóng mang cos

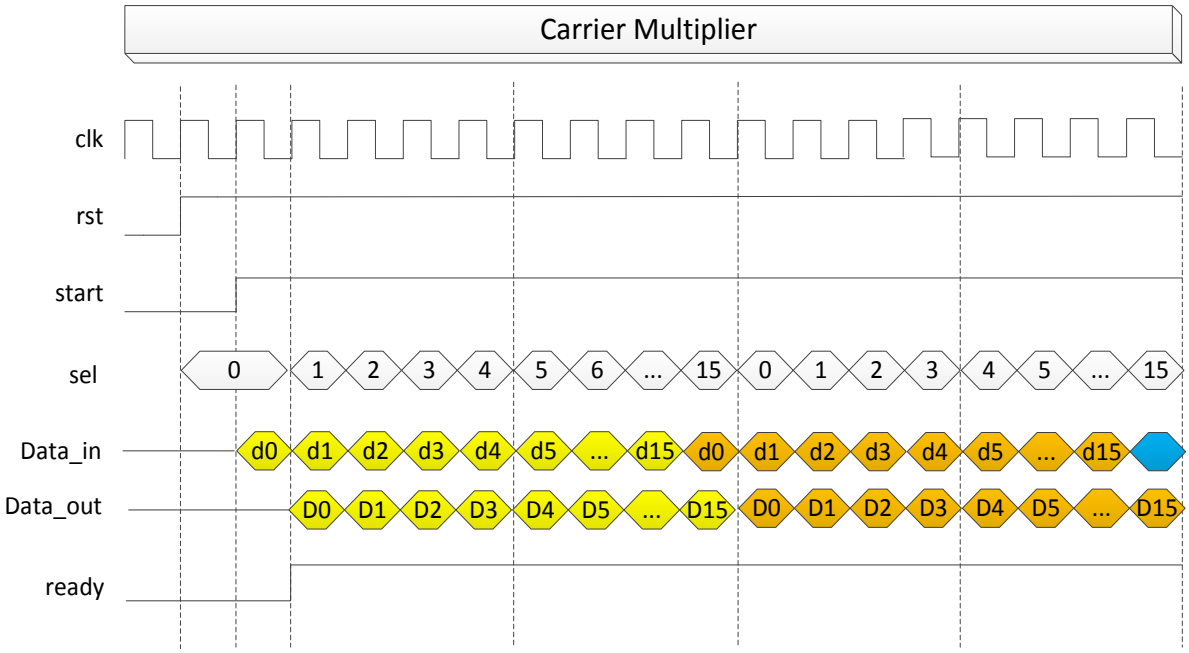


Sóng mang sin

##	Giá trị	##	Giá trị
sin0	0.0000	cos0	1.0000
sin1	0.3827	cos1	0.9239
sin2	0.7071	cos2	0.7071
sin3	0.9239	cos3	0.3827
sin4	1.0000	cos4	0.0000
sin5	0.9239	cos5	-0.3827
sin6	0.7071	cos6	-0.7071
sin7	0.3827	cos7	-0.9239
sin8	0.0000	cos8	-1.0000
sin9	-0.3827	cos9	-0.9239
sin10	-0.7071	cos10	-0.7071
sin11	-0.9239	cos11	-0.3827
sin12	-1.0000	cos12	0.0000
sin13	-0.9239	cos13	0.3827
sin14	-0.7071	cos14	0.7071
sin15	-0.3827	cos15	0.9239

Bảng 2. Giá trị của sóng mang sin và cos

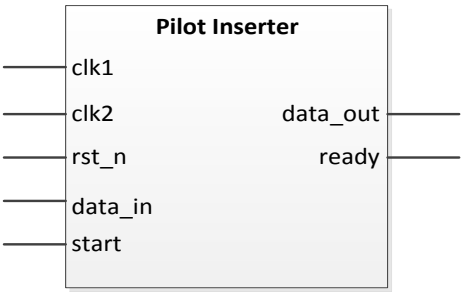
Waveform



Với d0..15 là 16bit dữ liệu vào. Và D0..15 là dữ liệu ra sau khi lấy dữ liệu vào nhân với giá trị x0..15

f. Pilot Interter

Block diagram



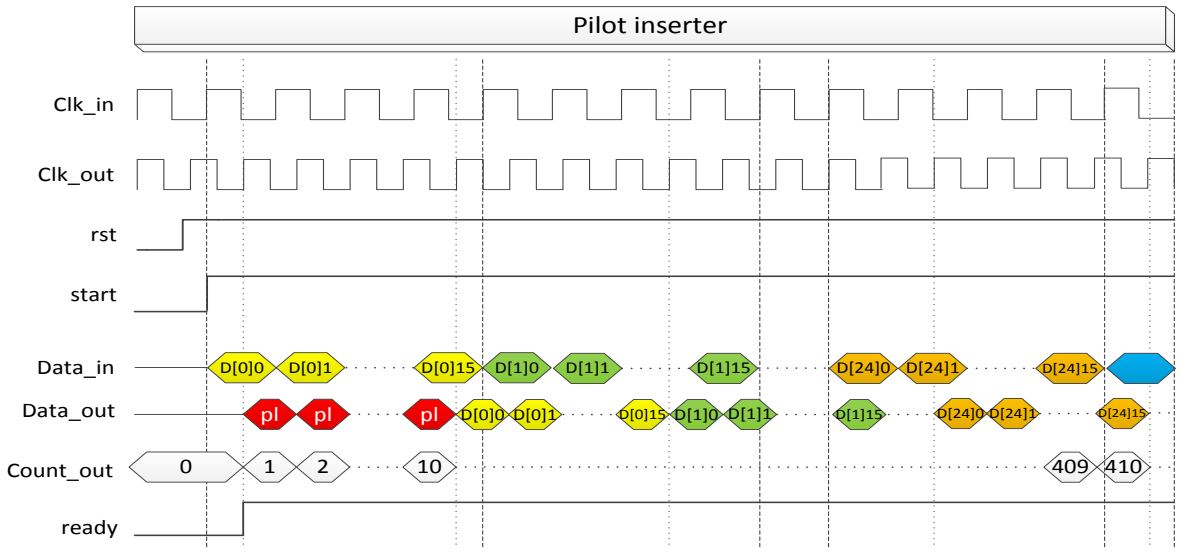
I/O port

Port	Data Width	Direction	Description
Clk1	1 bit	Input	Clock (10MHz)
Clk2	1 bit	Input	Clock (10.25MHz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Data_in	16 bit	Input	Dữ liệu vào
Data_out	16bit	Output	Dữ liệu ra
Ready	1 bit	Output	Báo hiệu dữ liệu ra

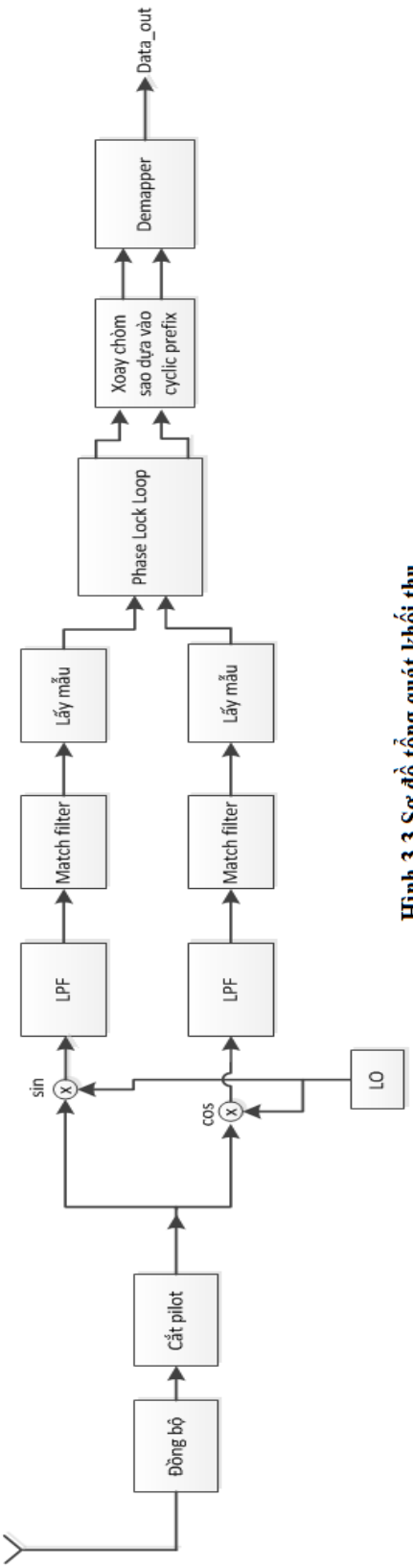
Operation

- Sau 25 điểm chòm sao (mỗi một chu kì sóng mang đại diện cho một điểm) phải chèn thêm 10 số 7 (mỗi số 7 được biểu diễn bằng 16 bit)
- Mỗi sóng mang được chia thành 16 điểm như ở các bộ phía trên
- Để chèn 10 số 7 thì tỷ lệ clk đọc vào và đọc ra là: 400/410

Waveform



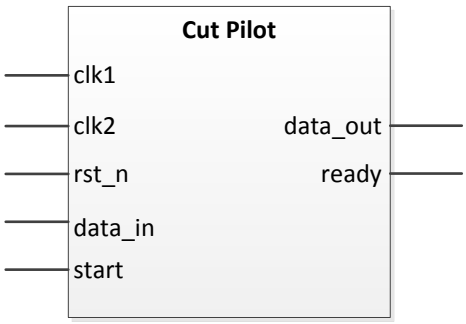
3.1.1 Sơ đồ khối thu:



Hình 3.3 Sơ đồ tổng quát khối thu

a. Cut pilot

Block diagram



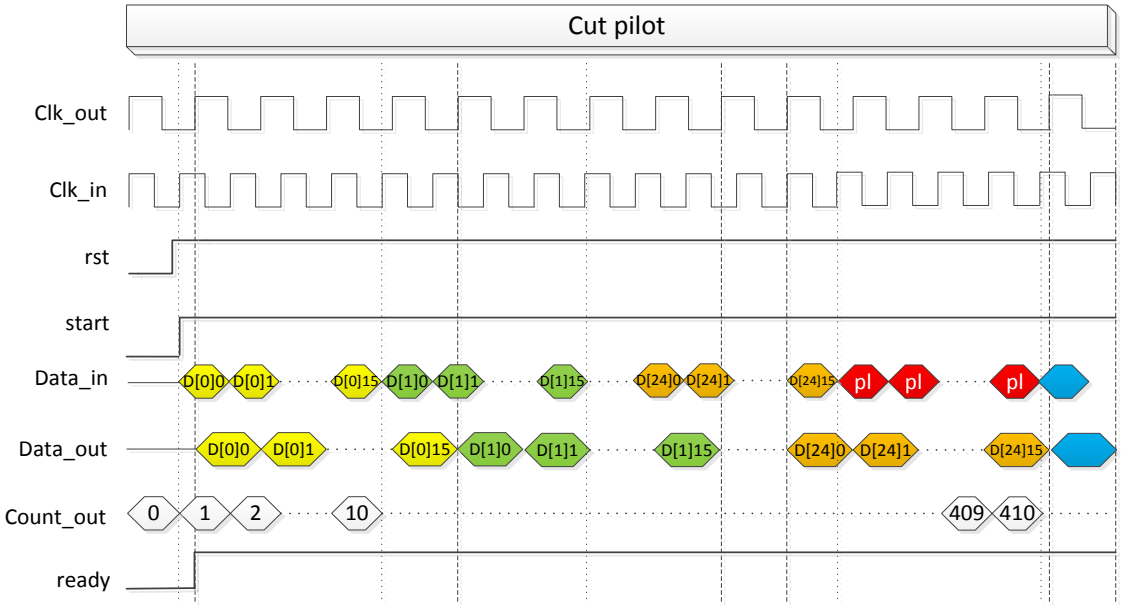
I/O port

Port	Data Width	Direction	Description
Clk1	1 bit	Input	Clock in(10.25MHz)
Clk2	1 bit	Input	Clock out (10MHz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Data_in	16 bit	Input	Dữ liệu vào
Data_out	16 bit	Output	Dữ liệu ra
Ready	1 bit	Output	Báo hiệu dữ liệu ra

Operation

- Sau 410 mẫu dữ liệu (tính từ lúc có tín hiệu start) phải cắt đi 10 số 7 (mỗi số 7 được biểu diễn bằng 16 bit)
- clk đọc vào và đọc ra là: 410/400

Waveform



b. Carrier multiplier

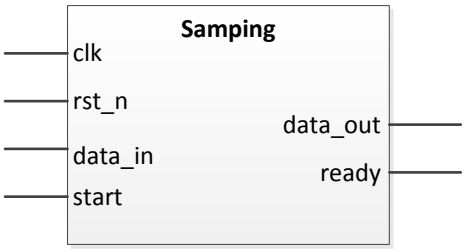
Khối này được kế thừa từ khối ở phần phát, với thông số kỹ thuật giống hệt nhau.

c. Raised cosin filter

Khối này được kế thừa từ khối ở phần phát, với thông số kỹ thuật giống hệt nhau.

d. Sampling

Block diagram



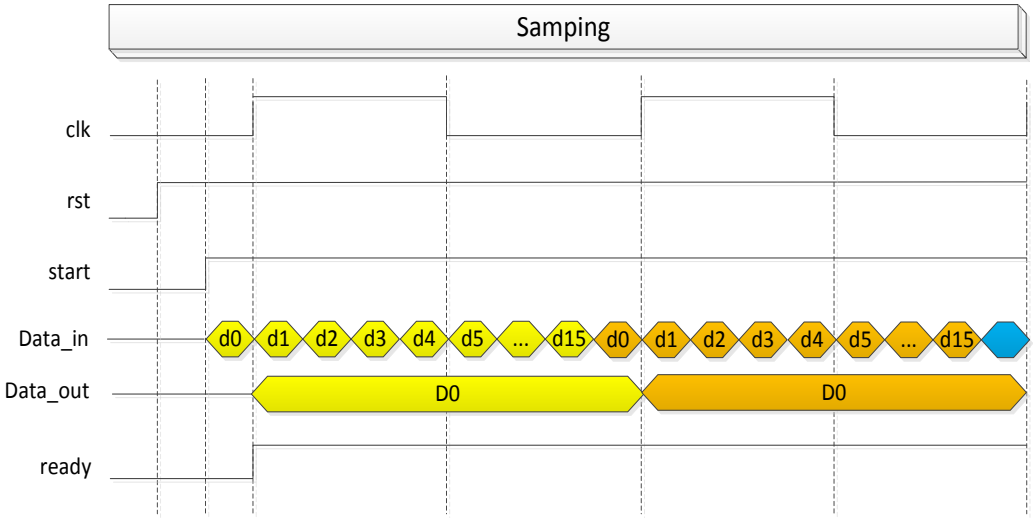
I/O port

Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (0.625MHz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Data_in	16 bit	Input	Dữ liệu vào
Data_out	16bit	Output	Dữ liệu ra
Ready	1 bit	Output	Báo hiệu dữ liệu ra

Operation

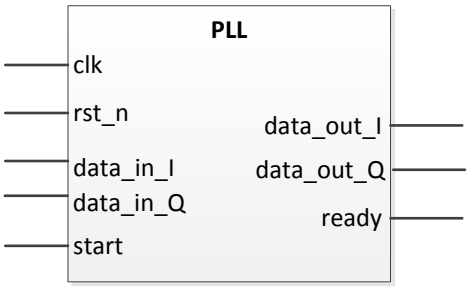
- Lấy mẫu tín hiệu vào với tốc độ: Cứ 16 mẫu thì lấy 1 mẫu

Waveform



e. Phase lock loop

Block diagram

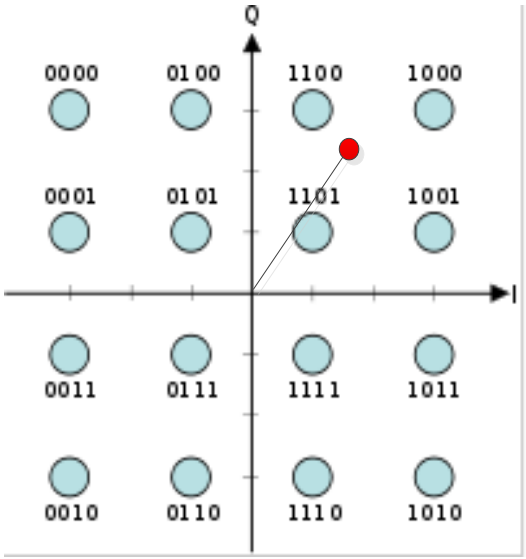


I/O port

Port	Data Width	Direction	Description
Clk	1 bit	Input	Clock (23.125MHz)
Rst_n	1 bit	Input	Reset
Start	1 bit	Input	Báo hiệu dữ liệu vào
Data_in_I	16 bit	Input	Dữ liệu vào kênh I
Data_in_Q	16 bit	Input	Dữ liệu vào kênh Q
Data_out_I	16 bit	Output	Dữ liệu ra kênh I
Data_out_Q	16 bit	Output	Dữ liệu ra kênh Q
Ready	1 bit	Output	Báo hiệu dữ liệu ra

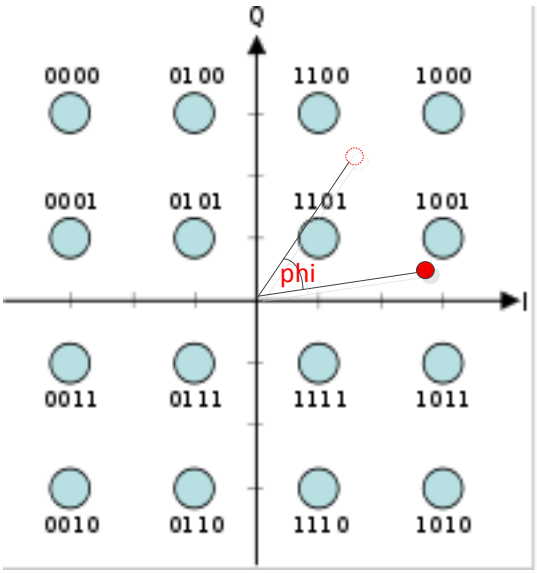
Operation – Hoạt động của bộ PLL:

Giả sử sau khi lấy mẫu ta thu được một cặp dữ liệu vào (I,Q) là chấm tròn nhỏ như ở hình 3.4:



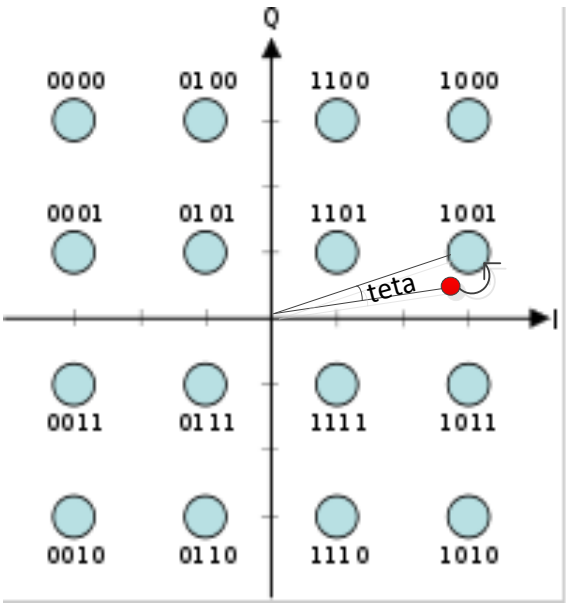
Hình 3.4 tọa độ điểm (x,y) trước bộ PLL

Tiếp theo ta phải xoay điểm "chấm nhỏ" đó theo một góc **phi_in**, góc **phi_in** này là tín hiệu hồi tiếp ở cuối bộ PLL. Ban đầu khi có dữ liệu đầu tiên vào thì góc phi_in này bằng 0, sau một chu kỳ thực hiện khối PLL, thì góc phi_in này sẽ được tính toán lại. Sau khi xoay góc phi_in, ta được hình 3.5:

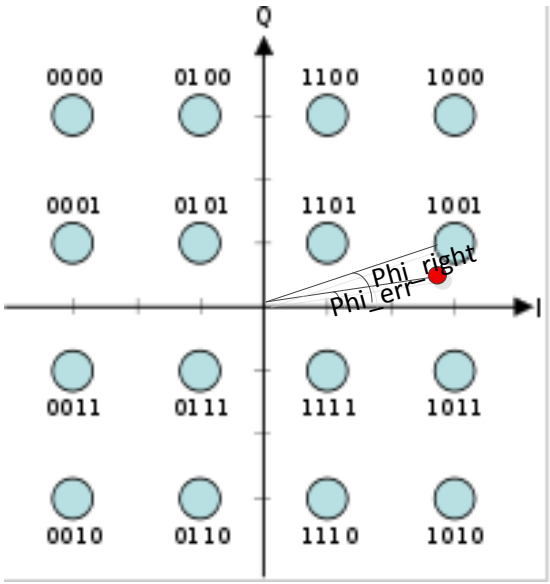


Hình 3.5 Xoay điểm (x,y) theo góc phi

Sau đó ta đưa điểm vừa quay đến một điểm sao gần nhất với nó, ở đây là điểm 1001, đồng thời tính được góc **teta** (hình 3.6)**.** Để tính được góc teta này ta phải tính góc hợp bởi điểm "chấm nhỏ" với trục I – Góc **phi_err**. Và ta đã biết góc hợp bởi vector đi qua điểm 1001 với trục I là **phi_right.** Sau đó ta tính teta=|phi_err-phi_right| (Hình 3.7)



Hình 3.6 Làm tròn điểm



Hình 3.7 Tính góc teta

Sau khi tính được góc teta, chúng ta sử dụng các công thức sau để tính được góc phi hồi tiếp cho dòng dữ liệu (I,Q) kế tiếp:

$$Teta=|phi\_err-phi\_right| \quad (3.1)$$

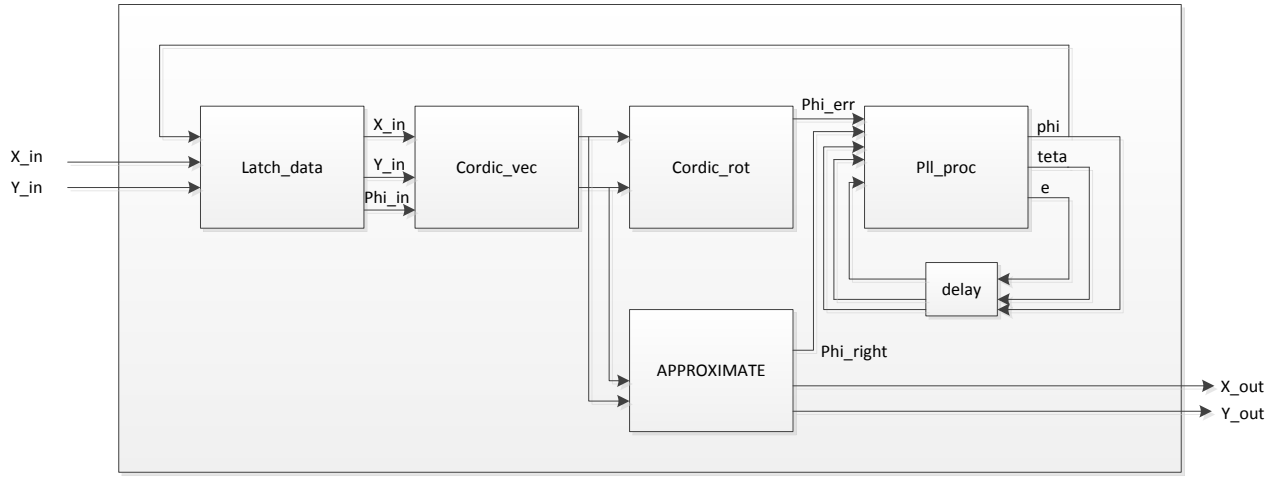
$$e=e'+k_p*teta+(k_i-k_p)*teta' \quad (3.2)$$

$$phi=phi'+e \quad (3.3)$$

với :

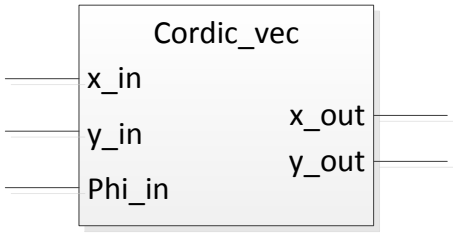
- $k_p=0.026$, $k_i=6.9 * 10^{-4}$
- e', teta', phi' là kết quả của lần dữ liệu trước

Thực hiện mạch:

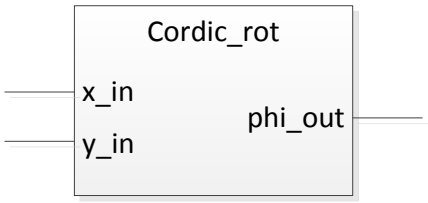


Bao gồm các khối:

- Latch data: Chốt dữ liệu, điều khiển đồng bộ dữ liệu. Đảm bảo dữ liệu hoạt động trong 37 chu kỳ đồng hồ
- Cordic vector: đầu vào là tọa độ (x,y) và góc cần quay, đầu ra là tọa độ (x',y') sau khi quay góc phi. Để thực hiện khối này, chúng ta cần đến thuật toán cordic. Thực hiện xoay góc trong vòng 16 chu kỳ đồng hồ. Thuật toán xoay Cordic sẽ được trình bày ở phần **3.2.1**

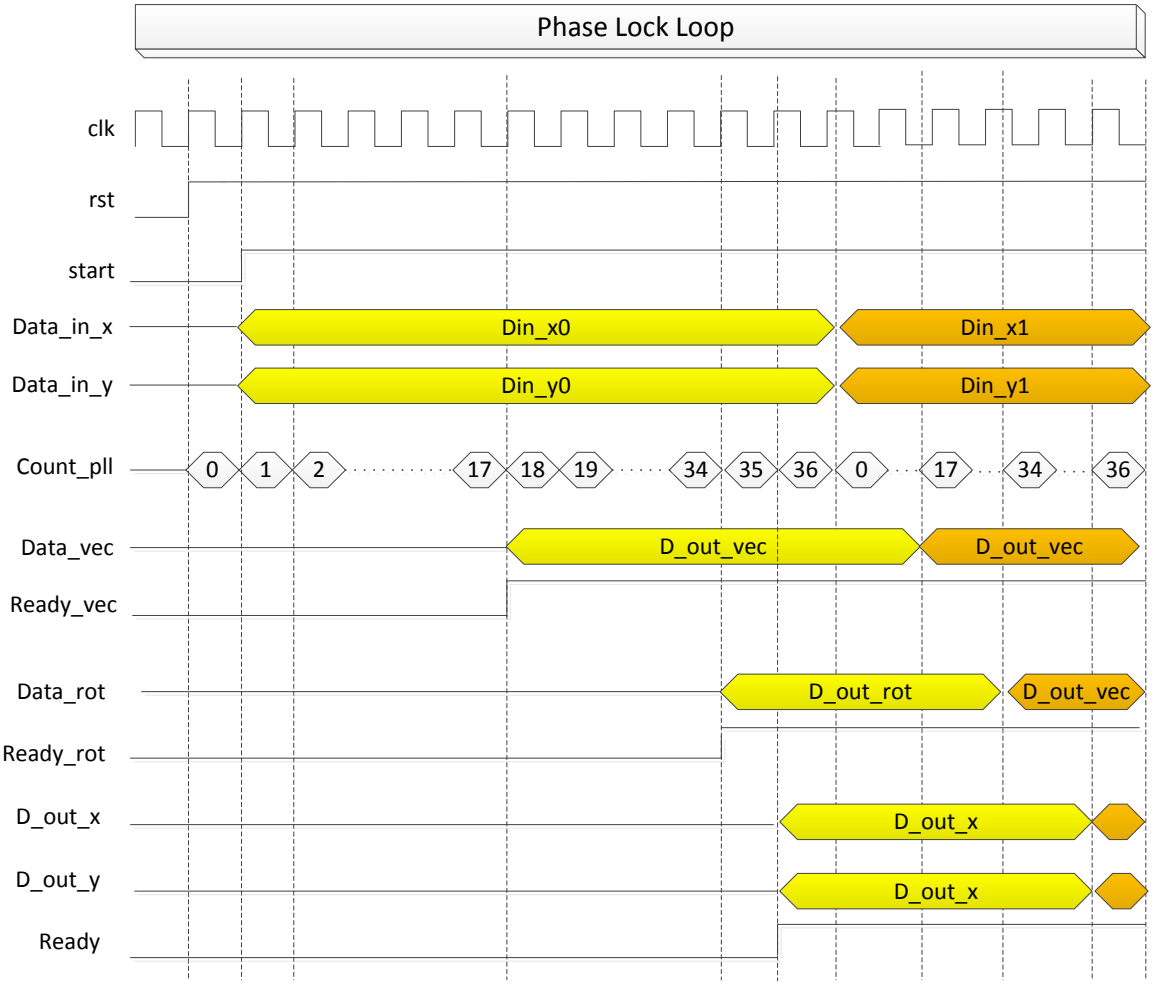


- Cordic rotation: đầu vào là tọa độ (x,y) đầu ra là góc sau khi quay tọa độ đó về điểm ($\sqrt{x^2+y^2}, 0$) (trục hoành). Để thực hiện khối này, chung ta sử dụng thuật toán xoay Cordic để xoay vector có tọa độ (x,y) về trục hoành, từ đó đưa ra góc quay



- Approximate: Làm tròn tọa độ đầu vào (x,y) về điểm gần nhất trên chòm sao, đồng thời đưa ra góc phi_right, nghĩa là góc tính từ điểm sao đó về đến trục hoành.
- Pll_proc: thực hiện tính toán ra góc phi hồi tiếp

Waveform


Phase Lock Loop
clk
rst
start
Data_in_x
Din_x0
Din_x1
Data_in_y
Din_y0
Din_y1
Count_pll
0
1
2
17
18
19
34
35
36
0
17
34
36
Data_vec
D_out_vec
D_out_vec
Ready_vec
Data_rot
D_out_rot
D_out_vec
Ready_rot
D_out_x
D_out_x
D_out_y
D_out_x
Ready

3.2 Phương án thiết kế mạch

3.2.1. Khối PLL

Thuật toán PLL sử dụng rất nhiều đến các phép toàn lượng giác và phép toàn nhân phức tạp, khó thực thi và triển khai trên phần cứng.

Thuật toán Cordic đã được đề cập trong bài nhằm thay thế các phép toàn phức tạp bằng các phép toán đơn giản có thể thực hiện được trên phần cứng(dịch và cộng), góp phần tiết kiệm tối đa tài nguyên phần cứng cũng như thời gian thực thi thuật toán Phase Lock Loop(PLL).

CORDIC (COordinate Rotation DIgital Computer), còn được biết đến với tên gọi Thuật toán Volder, là phương pháp đơn giản và hiệu quả để tính toán các hàm lượng giác và hypepol. Nó thường được dùng khi bộ nhân phần cứng không khả dụng (Ví dụ: các vi xử lý đơn giản, FPGAs), và được xây dựng từ các bộ cộng trừ, dịch bit, bảng tham chiếu.

Thuật toán CORDIC được xuất phát từ yêu cầu quay 1 vector về vector khác trên trục De-cac:

$$\begin{bmatrix} c_\theta & -s_\theta \\ s_\theta & c_\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x' \\ y' \end{bmatrix} \text{ hay } \begin{cases} x' = xc_\theta - ys_\theta \\ y' = yc_\theta + xs_\theta \end{cases}$$

Phương trình trên cho thấy phép quay 1 vector trong mặt phẳng Đề-các đi một góc θ. Phép toán này có thể được viết lại như sau:

$$\begin{cases} x' = \cos\theta.\left(x - y\tan\theta\right) \\ y' = \cos\theta.\left(y + x\tan\theta\right) \end{cases}$$

Nếu các góc quay được xác định sao cho $\tan\theta = \pm 2^{-i}$, phép nhân với tanθ có thể được đơn giản hóa bằng một phép dịch bit. Một góc quay tùy ý được thay thế bằng việc quay liên tiếp nhiều góc nhỏ. Khi đó cosθ là một hằng số. Quá trình quay liên tiếp này được thể hiện:

$$\begin{cases} x_{i+1} = K_i\left(x_i - s_i.y_i.2^{-i}\right) \\ y_{i+1} = K_i\left(y_i + s_i.x_i.2^{-i}\right) \end{cases}$$

Trong đó

$$K_i = \cos\left(\tan^{-1}\left(2^{-i}\right)\right) = 1\Big/\sqrt{1+2^{-2i}}$$

$$s_i = \pm 1$$

Loại bỏ các hằng số K_i khỏi phương trình trong mỗi vòng lặp. Tích của các hằng sốK_i sẽ được bổ sung vào kết quả cuối cùng. Tích này tiến tới giá trị 0.60725 khi lặp vô hạn lần:

$$K = \prod_{i=0}^{\infty} \frac{1}{\sqrt{1+2^{-2i}}} = 0.60725$$

Góc quay tổng hợp cũng được xác định từ các góc quay nhỏ tại mỗi lần lặp. Việc tính toán góc quay này bổ sung thêm phương trình thứ 3 vào thuật toán CORDIC.

$$z_{i+1} = z_i - s_i . \tan^{-1}\left(2^{-i}\right)$$

$Tan^{-1}(2^{-i})$ là hằng số, các giá trị này được lưu lại trong 1 bảng hằng số .Trên phần cứng, bảng này được lưu lại trong ROM.

Bảng 3. Hằng số Arctan

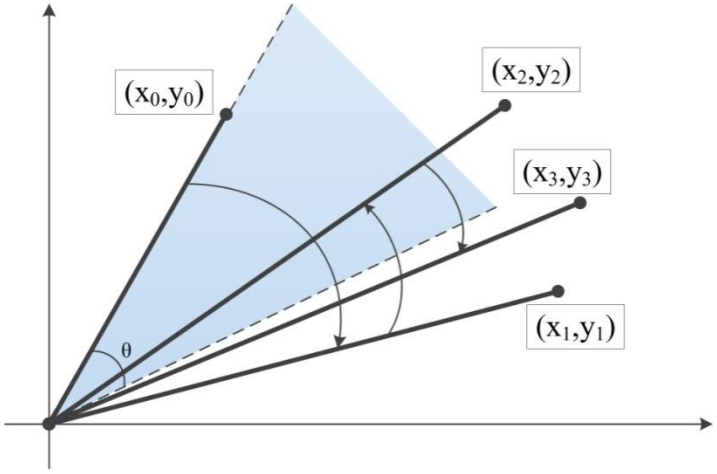
i	$\tan^{-1}(2^{-i})$
0	0.785398
1	0.463648
2	0.244979
3	0.124354
4	0.062419
5	0.031239
…	…

Hệ phương trình cho thuật toán CORDIC:

$$\begin{cases} x_{i+1} = x_i - s_i y_i 2^{-i} \\ y_{i+1} = y_i + s_i x_i 2^{-i} \\ z_{i+1} = z_i - s_i \tan^{-1}(2^{-i}) \end{cases}$$

CORDIC làm việc ở 2 chế độ. Chế độ thứ nhất gọi là chế độquay (Rotation). Trong chế độ này, vector được quay đi một góc xác định bằng tham số đầu vào. Chế độ thứ hai là chế độ vector (Vectoring). Vector được quay về trục x (hay nói cách khác, y bị triệt tiêu) đồng thời ghi lại góc quay được.

3.2.1.1. Chế độ quay (Rotation)



Hình 1. Phép quay CORDIC chế độ Rotation

Trong chế độquay (Rotation), thành phần lưu giá trị góc quay(thành phần z) được khởi tạo bằng giá trị góc cần quay. Việc quyết định hướng quay tại mỗi lần

lặp được thực hiện nhằm giảm dần sự sai khác giữa góc quay được và góc cần quay. Điều kiện cho hệ phương trình CORDIC trong chế độ này có dạng:

$$s_i = \begin{cases} 1 & \text{khi } z_i \geq 0 \\ -1 & \text{khi } z_i < 0 \end{cases}$$

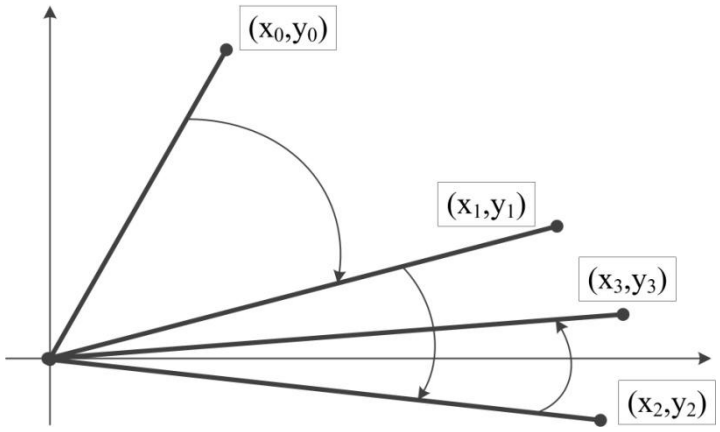
Thông số đầu vào của phép toán:

$$\begin{cases} x_0 = x_{in} \\ y_0 = y_{in} \\ z_0 = \theta \end{cases}$$

Thống số đầu ra của phép toán:

$$\begin{cases} x_n = \dfrac{1}{K}\left(x_{in}\cos z_0 - y_{in}\sin z_0\right) \\ y_n = \dfrac{1}{K}\left(y_{in}\cos z_0 + x_{in}\sin z_0\right) \\ z_n = 0 \end{cases}$$

3.2.1.2. Chế độ vector (Vectoring)



Hình b.2. Phép quay CORDIC chế độ Vectoring

Trong chế độ vector (Vectoring), CORDIC thực hiện xoay vector đầu vào cho đến khi vector này trùng với trục x. Việc quyết định hướng quay tại mỗi vòng lặp nhằm đưa thành phần y tiến dần về 0. Điều kiện cho hệ phương trình CORDIC trong chế độ này có dạng:

$$s_i = \begin{cases} 1 & \text{khi } y_i < 0 \\ -1 & \text{khi } y_i \geq 0 \end{cases}$$

Thông số đầu vào của phép toán:

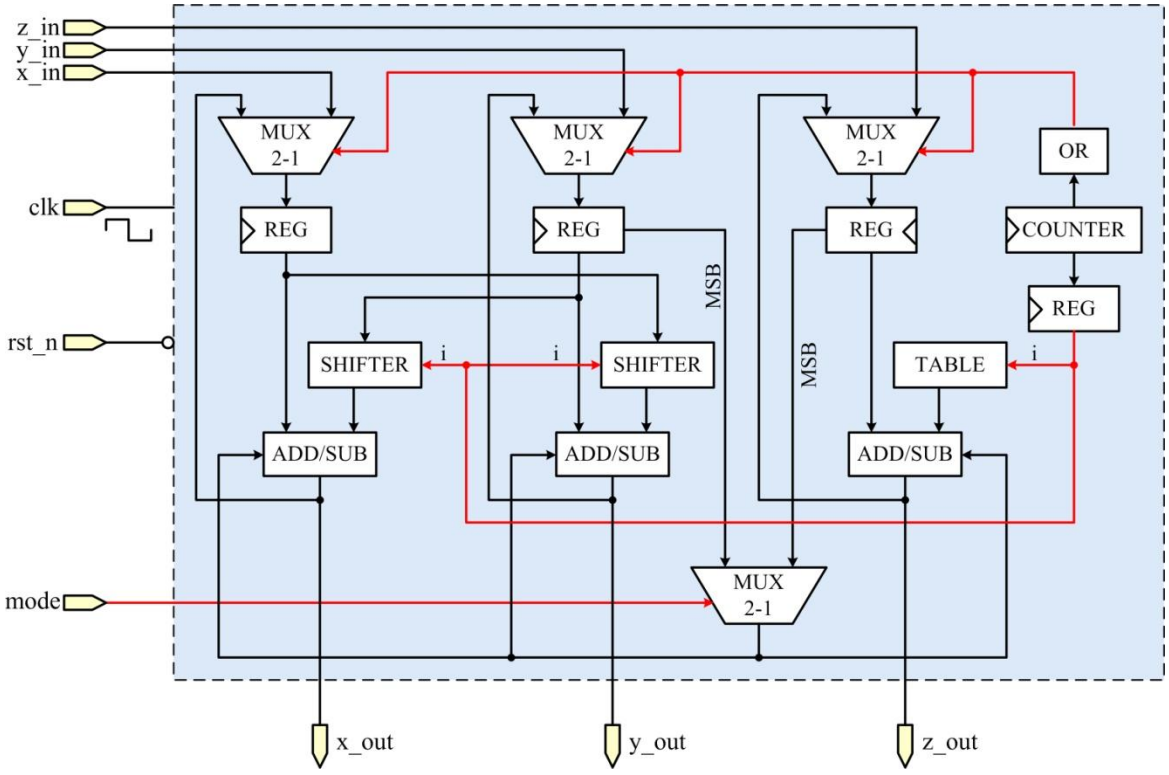
$$\begin{cases} x_0 = x_{in} \\ y_0 = y_{in} \\ z_0 = z_{in} \end{cases}$$

Thông số đầu ra của phép toán:

$$\begin{cases} x_n = \frac{1}{K}\sqrt{x_{in}^2 + y_{in}^2} \\ y_n = 0 \\ z_n = z_{in} + \tan^{-1}\left(\frac{y_{in}}{x_{in}}\right) \end{cases}$$

3.2.1.3. Triển khai CORDIC trên phần cứng

Từ hệ phương trình thuật toán CORDIC, có thể đưa ra sơ đồthiết kếsơ bộ cho thuật toán này như sau.



Hình 3. Sơ đồ thiết thuật toán CORDIC

Sơ đồ khối CORDIC thiết kế theo phương án hồi tiếp. Khối xử lý bao gồm các bộ MUX, bộ dịch phải, bộ cộng trừ có điều khiển và các thanh ghi, được thiết kế dựa theo sơ đồ thiết kế thuật toán sơ bộ. Khối điều khiển đơn giản là một Counter. Giá trị của counter xác định số bit cần dịch trong mỗi lần lặp, cũng như định vị dữ liệu cần tham chiếu trong bảng hằng số. Ngoài ra, việc thực hiện OR tất cả các bit của Counter

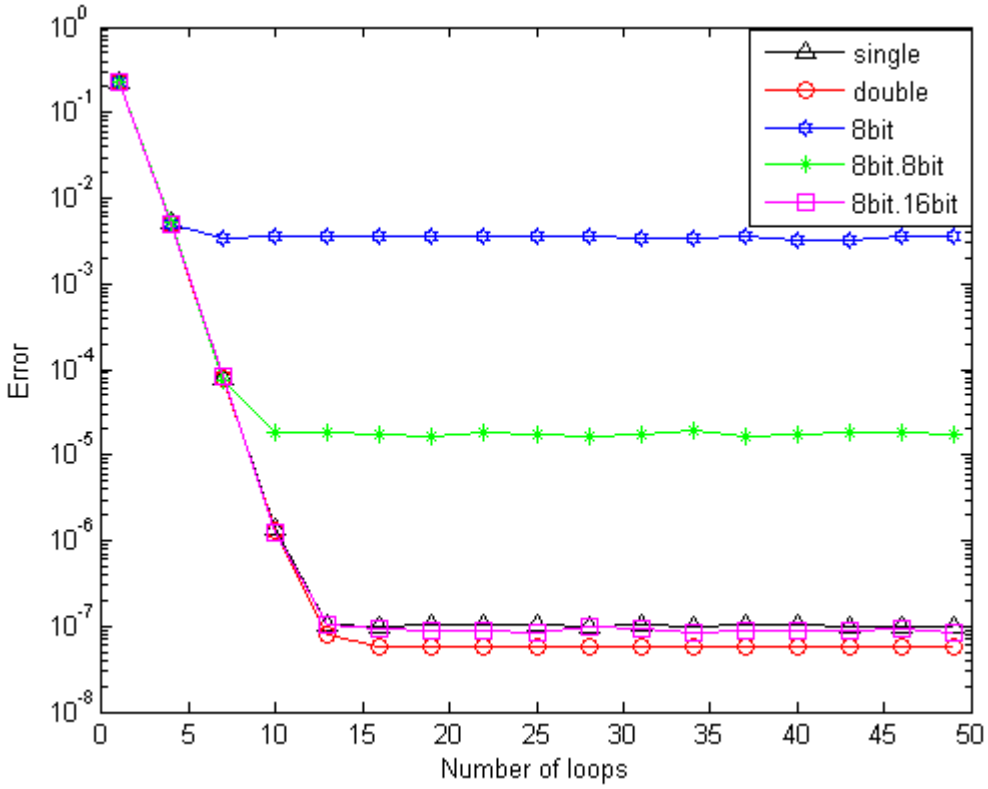
giúp điều khiển các khối MUX đầu vào. Khi tín hiệu điều khiển bằng 0 (Counter = 0), dữ liệu đầu vào được nạp và xử lý. Khi tín hiệu điều khiển khác 0 (Counter > 0), dữ liệu hồi tiếp được chọn để xử lý. Vì các khối Dịch bit, cộng trừ đều là mạch logic tổ hợp nên mỗi vòng lặp của phép toán được thực hiện trong 1 chu kỳ đồng hồ.

3.2.1.4. Đánh giá thuật toán CORDIC

Một hạn chế khi sử dụng khối CORDIC, cụ thể là với CORDIC – Vectoring, đó là góc quay trong từng vòng lặp có giá trị cố định xác định trước. Như vậy, không thể tránh khỏi trường hợp góc quay tổng hợp không đạt được tới góc cần quay.

Dưới đây là sơ đồ đánh giá sự phụ thuộc của độ chính xác vào kiểu dữ liệu và số vòng lặp.

(kết quả này được đánh giá dựa vào việc thực hiện code bằng phần mềm Matlab).



Theo thuật toán CORDIC – Vectoring, khi kết thúc phép toán, vector (x, y) được quay về trục x, hay nói cách khác, lúc này x mang giá trị $r = \sqrt{x^2 + y^2}$, còn y bị triệt tiêu. Sai số ở đây được tính bằng sai khác giữa giá trị cuối cùng của x và r.

$$\delta = \frac{|r - x|}{r}$$

Cũng qua đồ thị khảo sát trên hình, số vòng lặp cần thiết để thực hiện thuật toán CORDIC là 16. Giá trị này được chọn để cân đối giữa độ chính xác và thời gian thực

thi phép toán, bên cạnh đó, nó cũng giúp cho việc tận dụng hết khả năng lưu trữ của 1 thanh ghi 4 bit.

Trong bài phần CORDIC dữ liệu đầu vào chọn kiểu int16 bao gồm 8bit nguyên và 8bit phần thập phân.

3.2.2. Khối bộ lọc và khối nhân sóng mang

Thay vì sử dụng phép nhân 16bit, ta sử dụng cách dịch bit

Ví dụ: nhân số 16 bit với số 0.607252925634384, ta biến đổi số 0.607252925634384 về dạng nhị phân: 0.10011011011110100111011011. Sau đó chỉ cần lấy số nhân dịch bit theo số nhị phân ấy.

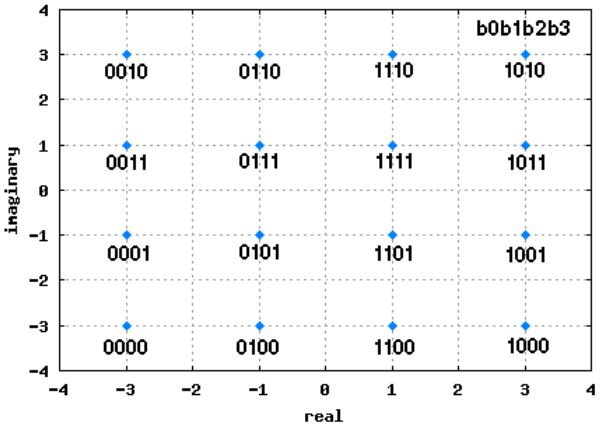
Một đoạn code cho ví dụ trên:

```
module mulconst
#(
  parameter WIDTH = 16          // Data width
)
(
  input signed [WIDTH-1 : 0]        in,
  output       [WIDTH-1 : 0]        out
);
  // 0.10011011011110100111011011 = 0.607252925634384
  assign out =
(in>>>1)+(in>>>4)+(in>>>5)+(in>>>7)+(in>>>8)+(in>>>10)+(in>>>11)+(in>>>12)

+(in>>>14)+(in>>>17)+(in>>>18)+(in>>>19)+(in>>>21)+(in>>>22)+(in>>>24)+(in
>>>25);

Endmodule
```

3.2.3. Khối Mapper

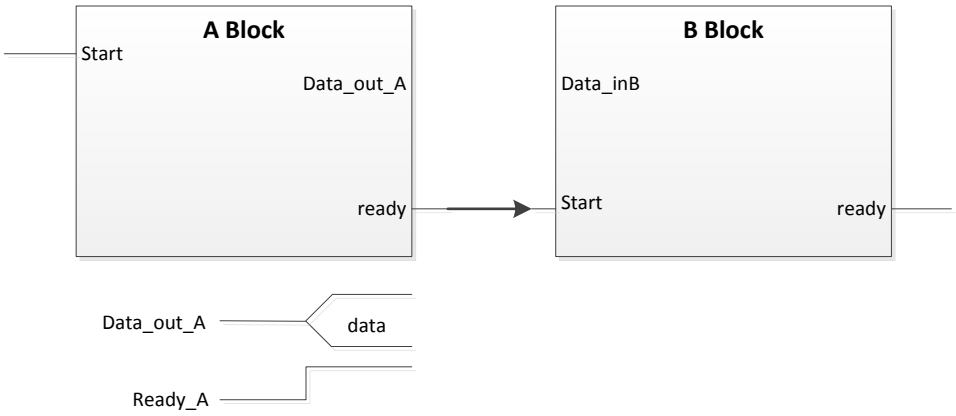
Thực hiện mạch bằng phương pháp Look up table. Tín hiệu vào sẽ được ánh xạ lên chòm sau theo bảng sau:

b_0b_1	I	b_2b_3	Q
00	-3	00	-3
01	-1	01	-1
11	+1	11	+1
10	+3	10	+3



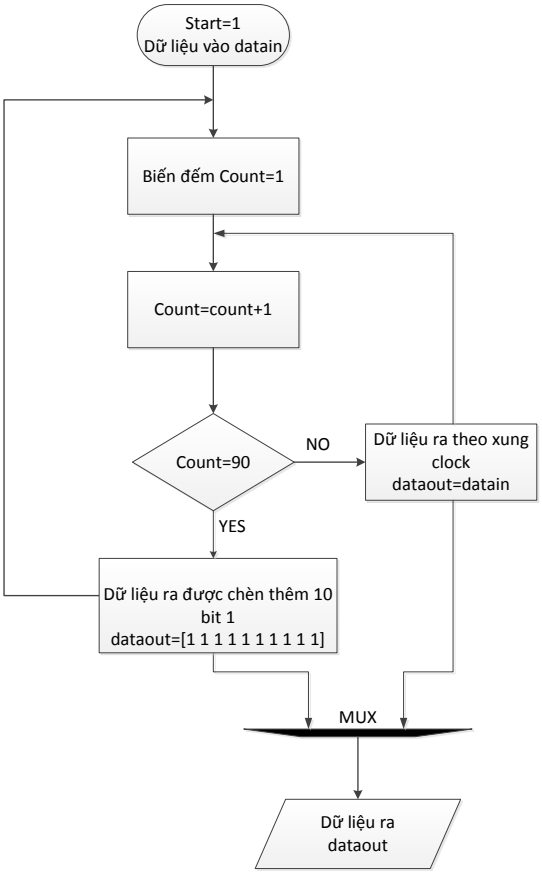
3.2.4. Cách ghép nối các khối

Để ghép nối một cách đồng bộ giữa các khối, nhóm em sử dụng 2 tín hiệu start và ready. Tín hiệu **Start** để báo có dữ liệu vào, và khi đó mạch bắt đầu hoạt động. Khối **Ready** để báo dữ liệu ra, tín hiệu này có vai trò kích thích khối liền sau nó hoạt động bằng cách nối vào chân **Start**của khối đó.

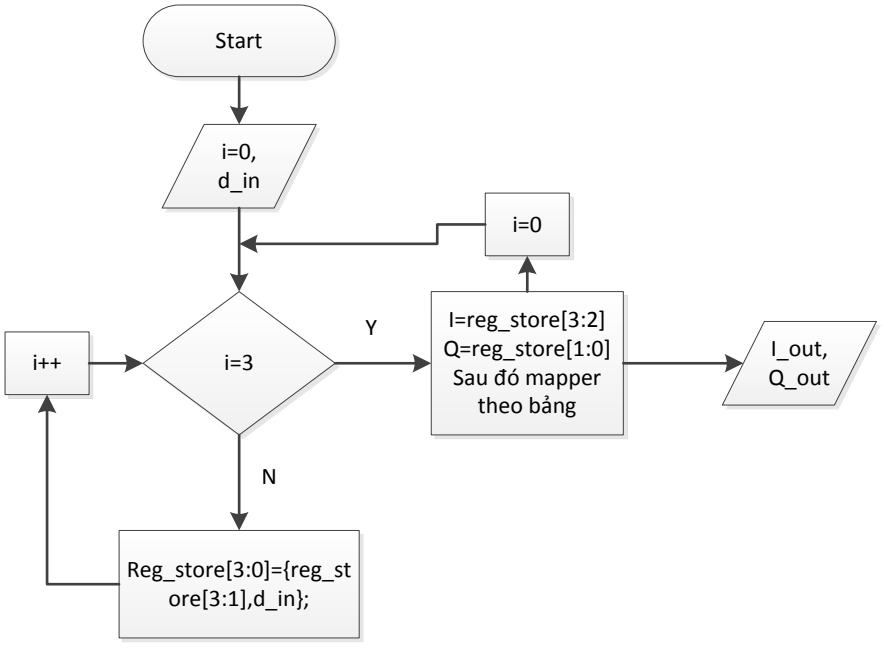


Phần 4. Thiết kế mạch

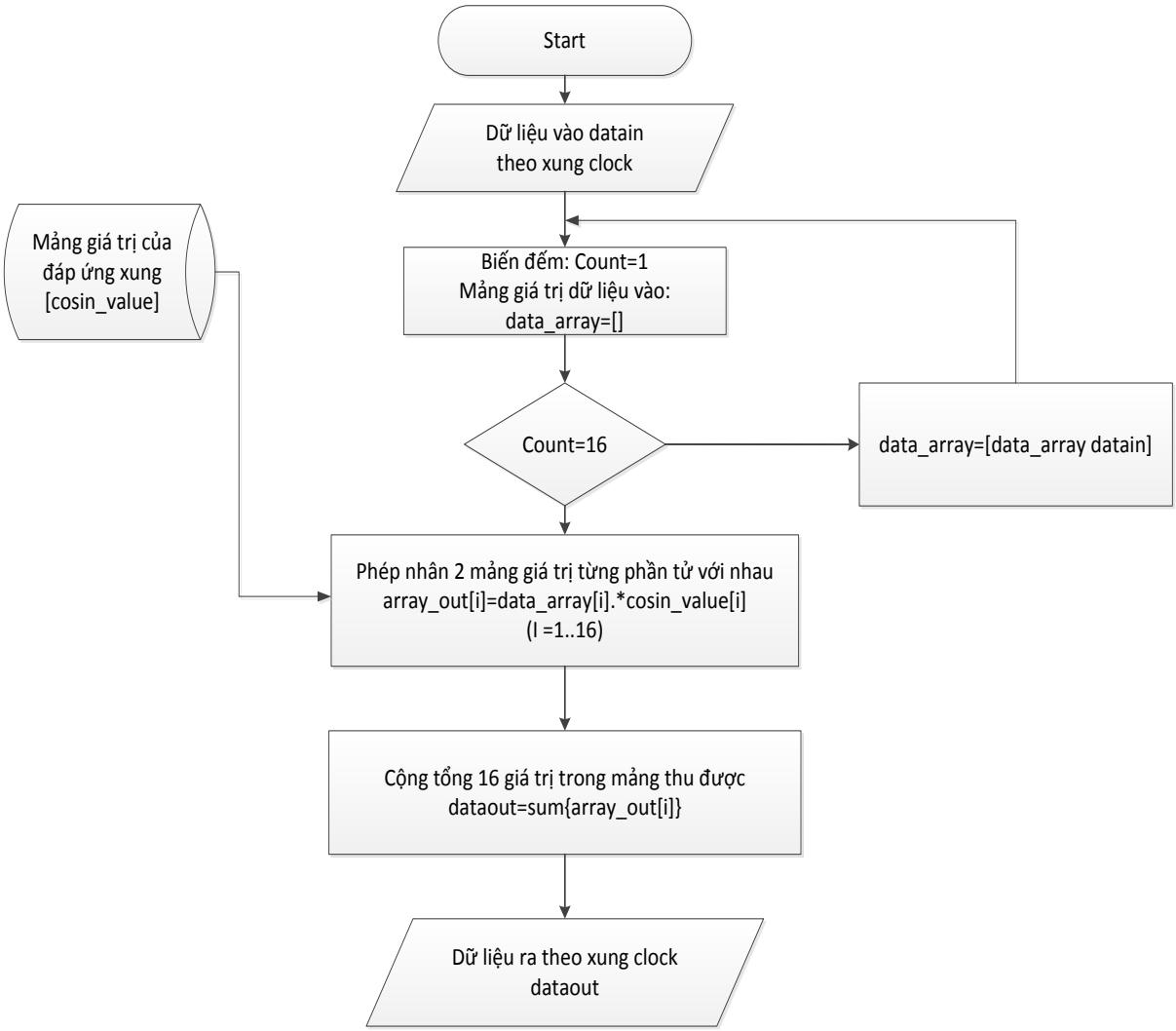
4.1 Lưu đồ thuật toán khối Cyclic Prefix



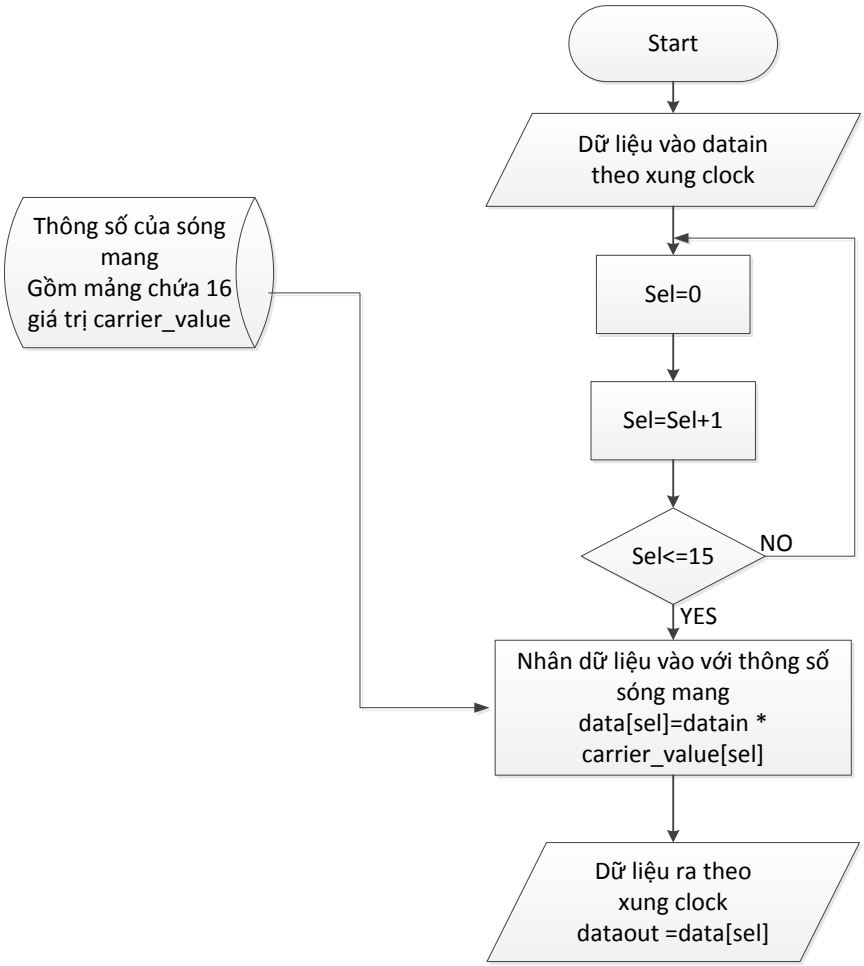
4.2 Lưu đồ thuật toán khối Mapper



4.3 Lưu đồ thuật toán khối bộ lọc

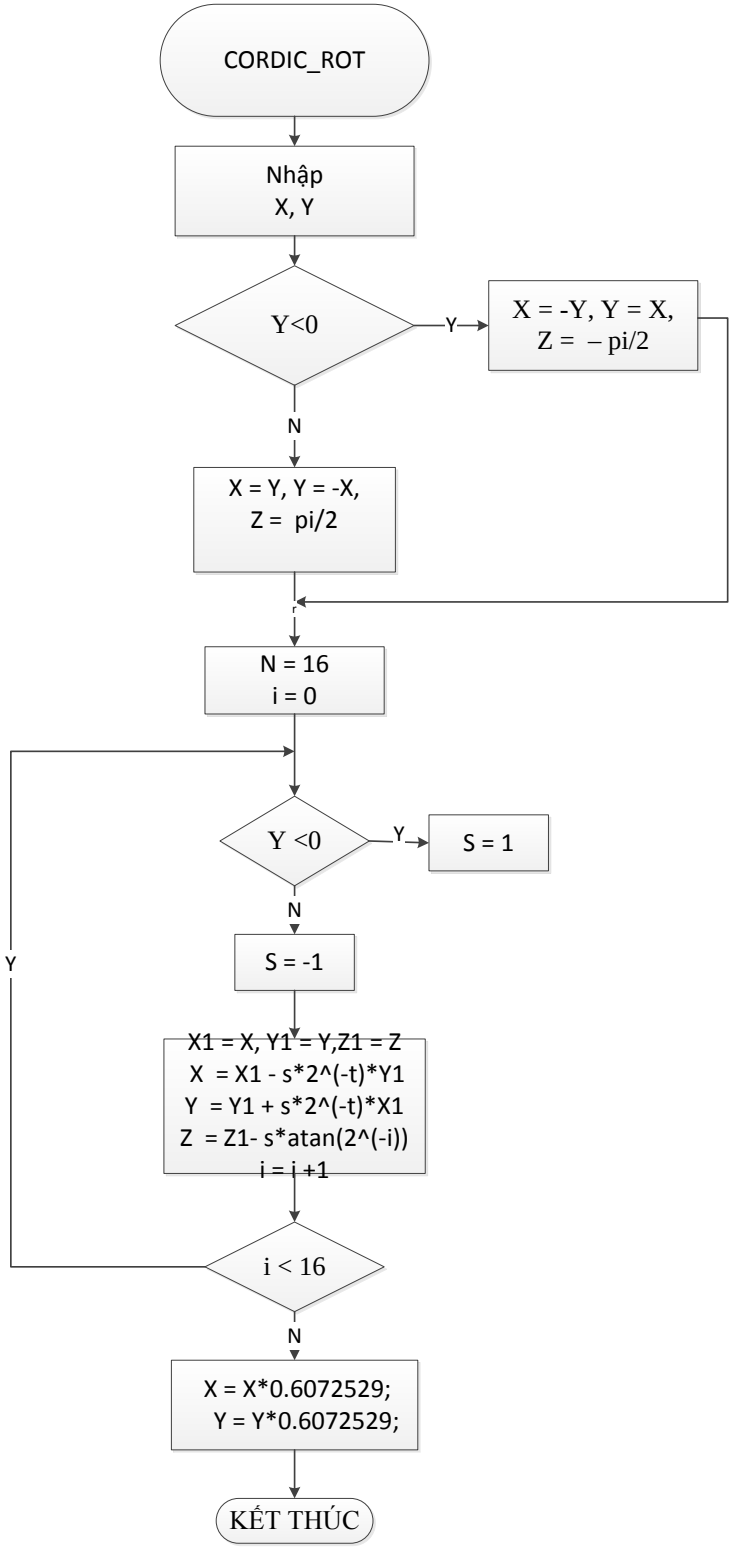


4.4 Lưu đồ thuật toán khối nhân sóng mang

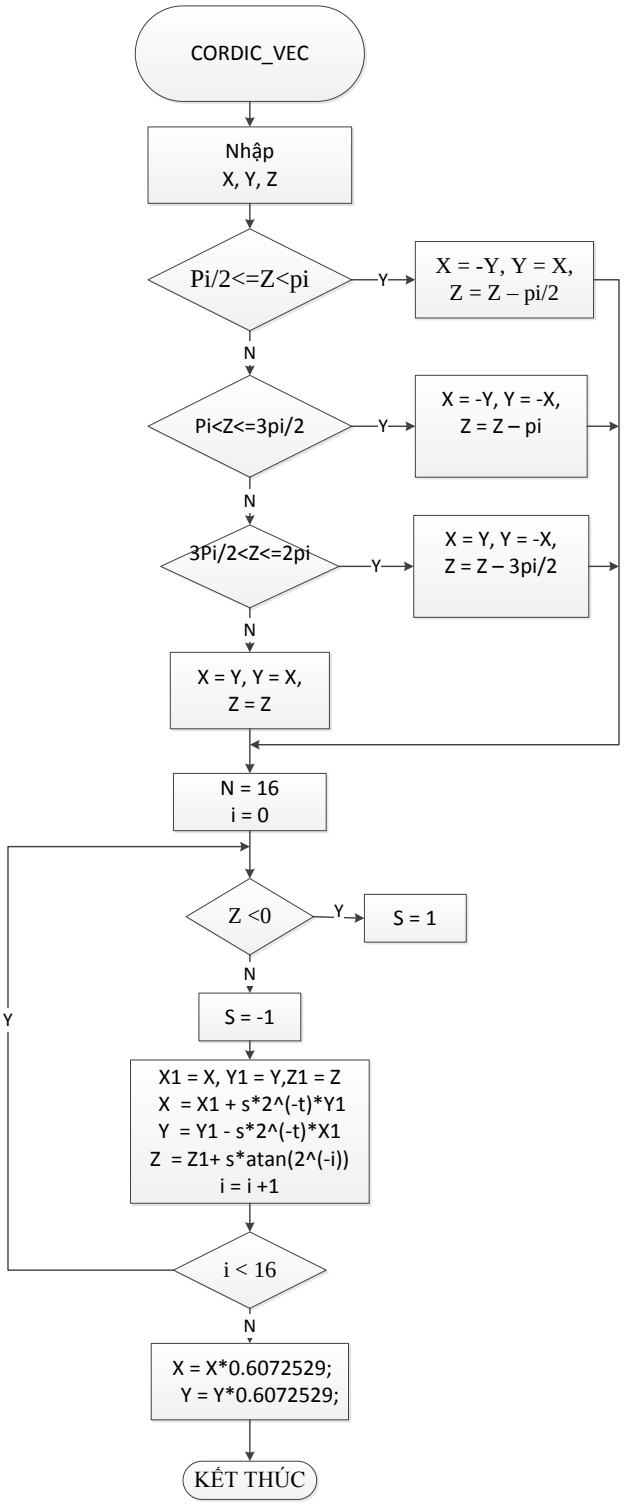


4.5 Lưu đồ thuật toán khối PLL

4.5.1. Lưu đồ thuật toán khối Cordic rotation



4.5.2. Lưu đồ thuật toán khối Cordic vector



Phần 5. Triển khai

Phần này trình bày việc triển khai cụ thể một số khối. Tuy nhiên do khối lượng code quá lớn, vì vậy toàn bộ code chương trình sẽ được lưu trong đĩa CD

5.1 Khối phát

5.1.1. Chèn khoảng bảo vệ - Cyclic prefix

```
module cyclic_prefix
(
      input                   clk,
      input                   rst_n,
      input                   data_in,
      input                   start,
      output      reg   data_out,
      output      reg   ready
);

      reg   data_in_r;
      reg   [6:0] count;
      reg   ready_temp; //lam tre ready 1 chu ki theo spec

      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  count<=0;
            end
            else if(start) begin
                  if(count==7'd99)  count<=0;
                  else  count<=count+1;
            end
      end

      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  data_in_r<=0;
                  ready_temp<=0;
            end
            else if(start) begin
                  data_in_r<=data_in;
                  ready_temp<=1'b1;
            end
      end

      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  data_out<=0;
                  ready<=0;
            end
            else if(start) begin
                  ready<=ready_temp;
                  if(count<7'd90) data_out<=data_in_r;
                  else  data_out<=16'b1;
            end
      end

endmodule
```

5.1.2. Mapper

```
module mapper
#(
      parameter width_data = 16
)
(
      input                         data_in,
      input                         clk,
      input                   rst_n,
      input                   start,
      output reg [width_data-1:0]   data_I,
      output reg [width_data-1:0]   data_Q,
      output reg                    ready
);
      wire        data_in_wire;//Day noi voi data_in
      reg         [3:0] reg_sh;//thanh ghi dich
      reg         [1:0] count;
      reg         [3:0] reg_I,reg_Q;
      reg   start_a,start_b,start_c,start_d;

      assign data_in_wire = (start) ?     data_in : 1'b0;

      always @ (posedge clk or negedge rst_n)begin    //Khoi  tao  count  va
dich thanh ghi reg_sh
            if(!rst_n)begin
                        count<=0;
                        reg_sh<=0;
            end
            else if(start) begin
                        count<=count+1;
                        reg_sh<={data_in_wire,reg_sh[3:1]};
            end
      end

      always @ (posedge clk or negedge rst_n)begin //xu ly ready
            if(!rst_n)begin
                  ready<=0;
                  start_a<=0;
                  start_b<=0;
                  start_c<=0;
                  start_d<=0;
            end
            else if(start) begin
                  start_a<=1;
                  start_b<=start_a;
                  start_c<=start_b;
```

```
                start_d<=start_c;
                ready<=start_d;
        end
    end

    always @ (posedge clk or negedge rst_n)begin//data out
        if(!rst_n)begin
                data_I<=0;
                data_Q<=0;
        end
        else begin
                data_I<={reg_I,12'b0};
                data_Q<={reg_Q,12'b0};
        end
    end

    always @ (count or reg_sh or start)begin  // Mapper
        if((count==0)&&(start))begin
                    case(reg_sh[3:2])
                        2'b00:
                                    reg_I=-4'd3;
                        2'b01:
                                    reg_I=-4'd1;
                        2'b10:
                                    reg_I=4'd3;
                        default:
                                    reg_I=4'd1;
                    endcase

                    case(reg_sh[1:0])
                        2'b00:
                                    reg_Q=-4'd3;
                        2'b01:
                                    reg_Q=-4'd1;
                        2'b10:
                                    reg_Q=4'd3;
                        default:
                                    reg_Q=4'd1;
                    endcase
        end
        else begin
                    reg_I=4'b0000;
                    reg_Q=4'b0000;
        end
    end
    endmodule
```

5.1.3. Chèn không - Zero padder

```
module zero_padder
#(
      parameter width_data = 16
)

(
      input                                          clk,
      input                                          rst_n,
      input                                          start,
      input                                          sel,
      input       [width_data-1:0]  data_in,
      output      [width_data-1:0]  data_out,
      output      reg                            ready
);

      reg         [width_data-1:0]  data;
      wire        [width_data-1:0]  data_in_wire;     //Day luu gia tri
data_in

      assign data_in_wire = (start) ? data_in : 16'b0;
      assign data_out = data;

      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  data<=16'b0;
                  ready<=1'b0;
            end
            else begin
                  if(!sel)    begin
                                          data<=data_in_wire;
                                          ready <=1;
                                    end
                  else        data<=16'b0;
            end
      end

endmodule
```

5.1.4. Bộ lọc cos nâng - Raised cosin filter

Trong CD

5.1.5. Bộ nhân sóng mang - Carry multiplier

Bộ nhân sóng mang cos

```
module carrier_multi_cos
#(
      parameter width_sym     = 16,       // do rong cua 1 ky tu
      parameter width_sel           = 4         // do rong cua bien sel
)
(
      input             clk,
      input             rst_n,
```

```
      input                    [width_sel-1:0]         sel,
      input       signed       [width_sym-1:0]         data_in,
      input              start,
      output reg         [width_sym-1:0]         data_out,
      output reg         ready             //Bao da nap xong du lieu
);

      wire [width_sym-1:0] data_out_0    ;
      wire [width_sym-1:0] data_out_1    ;
      wire [width_sym-1:0] data_out_2    ;
      wire [width_sym-1:0] data_out_3    ;
      wire [width_sym-1:0] data_out_4    ;
      wire [width_sym-1:0] data_out_5    ;
      wire [width_sym-1:0] data_out_6    ;
      wire [width_sym-1:0] data_out_7    ;
      wire [width_sym-1:0] data_out_8    ;
      wire [width_sym-1:0] data_out_9    ;
      wire [width_sym-1:0] data_out_10   ;
      wire [width_sym-1:0] data_out_11   ;
      wire [width_sym-1:0] data_out_12   ;
      wire [width_sym-1:0] data_out_13   ;
      wire [width_sym-1:0] data_out_14   ;
      wire [width_sym-1:0] data_out_15   ;
      wire [width_sym-1:0]    data_in_temp;
      wire [width_sym-1:0]    data_in_temp2;
      reg  [width_sym-1:0]    data_out_temp;
      reg                                  data_in_temp3;
      reg                                  ready_delay;

      assign data_in_temp2= start? data_in:0;
      assign data_in_temp= data_in_temp2[15]?
(~(data_in_temp2)+16'b1):data_in_temp2;


      assign data_out_0  = 0;
      assign data_out_1  =
(data_in_temp>>>2)+(data_in_temp>>>3)+(data_in_temp>>>8)+(data_in_temp>>>9
)+(data_in_temp>>>10)+(data_in_temp>>>11)+(data_in_temp>>>12);
      assign data_out_2  =
(data_in_temp>>>1)+(data_in_temp>>>3)+(data_in_temp>>>4)
+(data_in_temp>>>6) +(data_in_temp>>>8);
      assign data_out_3  =
(data_in_temp>>>1)+(data_in_temp>>>2)+(data_in_temp>>>3)
+(data_in_temp>>>5) +(data_in_temp>>>6) +(data_in_temp>>>9);
      assign data_out_4  = data_in_temp;
      assign data_out_5  =
(data_in_temp>>>1)+(data_in_temp>>>2)+(data_in_temp>>>3)
+(data_in_temp>>>5) +(data_in_temp>>>6) +(data_in_temp>>>9);
      assign data_out_6  =
(data_in_temp>>>1)+(data_in_temp>>>3)+(data_in_temp>>>4)
+(data_in_temp>>>6) +(data_in_temp>>>8);
      assign data_out_7  =
(data_in_temp>>>2)+(data_in_temp>>>3)+(data_in_temp>>>8)+(data_in_temp>>>9
)+(data_in_temp>>>10)+(data_in_temp>>>11)+(data_in_temp>>>12);
      assign data_out_8  = 0;
      assign data_out_9  =
~((data_in_temp>>>2)+(data_in_temp>>>3)+(data_in_temp>>>8)+(data_in_temp>>
>9)+(data_in_temp>>>10)+(data_in_temp>>>11)+(data_in_temp>>>12))+16'b1;
      assign data_out_10 =
~((data_in_temp>>>1)+(data_in_temp>>>3)+(data_in_temp>>>4)
+(data_in_temp>>>6) +(data_in_temp>>>8))+16'b1;
```

```
      assign data_out_11 =
~((data_in_temp>>>1)+(data_in_temp>>>2)+(data_in_temp>>>3)
+(data_in_temp>>>5) +(data_in_temp>>>6) +(data_in_temp>>>9))+16'b1;
      assign data_out_12 = ~( data_in_temp)+16'b1;
      assign data_out_13 =
~((data_in_temp>>>1)+(data_in_temp>>>2)+(data_in_temp>>>3)
+(data_in_temp>>>5) +(data_in_temp>>>6) +(data_in_temp>>>9))+16'b1;
      assign data_out_14 =
~((data_in_temp>>>1)+(data_in_temp>>>3)+(data_in_temp>>>4)
+(data_in_temp>>>6) +(data_in_temp>>>8))+16'b1;
      assign data_out_15 =
~((data_in_temp>>>2)+(data_in_temp>>>3)+(data_in_temp>>>8)+(data_in_temp>>
>9)+(data_in_temp>>>10)+(data_in_temp>>>11)+(data_in_temp>>>12))+16'b1;

      always @ (posedge clk or negedge rst_n  ) begin
            if(!rst_n) begin
                  data_out <= 0 ;
                  ready<=0;
                  ready_delay<=0;
                  data_out_temp<=0;
            end
            else  begin
                              case(sel)
                                    4'b0000: data_out_temp<= data_out_4  ;
                                    4'b0001: data_out_temp<= data_out_5  ;
                                    4'b0010: data_out_temp<= data_out_6  ;
                                    4'b0011: data_out_temp<= data_out_7  ;
                                    4'b0100: data_out_temp<= data_out_8  ;
                                    4'b0101: data_out_temp<= data_out_9  ;
                                    4'b0110: data_out_temp<= data_out_10 ;
                                    4'b0111: data_out_temp<= data_out_11 ;
                                    4'b1000: data_out_temp<= data_out_12 ;
                                    4'b1001: data_out_temp<= data_out_13 ;
                                    4'b1010: data_out_temp<= data_out_14 ;
                                    4'b1011: data_out_temp<= data_out_15 ;
                                    4'b1100: data_out_temp<= data_out_0  ;
                                    4'b1101: data_out_temp<= data_out_1  ;
                                    4'b1110: data_out_temp<= data_out_2  ;
                                    default: data_out_temp<= data_out_3  ;
                              endcase
                              data_in_temp3<=data_in_temp2[15];//
data_in_temp3 la tre cua data_in_temp de viec xac dinh am duong se van giu
duoc den khi ra data_out
                              if(!data_in_temp3) begin
                                    data_out<=data_out_temp;

                              end
                              else begin
                                    data_out<=~(data_out_temp)+16'b1;
                              end
                              ready<=ready_delay;
                              ready_delay<=1;
                  end
      end
endmodule
```

Bộ nhân sóng mang sin
Trong CD

5.1.6. Bộ chèn pilot - pilot inserter

```
Trong CD
```

5.1.7. Bộ điều khiển – Controller

```
TOP Module
module      controller_qam_16
#(
      parameter   wid_count = 4
)

(
      input       clk,
      input       rst_clk,
      input       rst_n,
      input       ready_mapper,
      input       ready_zero,
      input       ready_filter,
      output      clk_12,     //clk_12=12clk
      output      clk_3,      //clk_3=3clk
      output      clk_2,      //clk_2=2clk      : su dung cho khoi pilot
      output      sel_zero_pad,
      output      ce_shift,
      output      [wid_count-1:0]   sel_carrier
);
////////////////////////////////////////////////
//Sinh clock tu clock chuan
gen_clk
      my_gen_clk(
                  .clk(clk),
                  .rst_clk(rst_clk),
                  .clk_12(clk_12),
                  .clk_3(clk_3),
                  .clk_2(clk_2)
      );
////////////////////////////////////////////////
//dau ra tin hieu dieu khien
proc_qam    #     (.wid_count(wid_count))
      process_qam(
                  .clk(clk_3),
                  .rst_n(rst_n),
                  .ready_mapper(ready_mapper),
                  .ready_filter(ready_filter),
                  .ready_zero(ready_zero),
                  .sel_zero_pad(sel_zero_pad),
                  .sel_carrier(sel_carrier),
                  .ce_shift(ce_shift)
      );

endmodule
```

```
Tạo clock
module gen_clk
(
      input                   clk,
      input                   rst_clk,
      output      clk_12,     //clk_12=12clk
      output      clk_3,      //clk_3=3clk
      output      reg   clk_2 //clk_2=2clk      : su dung cho khoi pilot
```

```
);
      reg   [1:0] count_3;
      reg   [3:0] count_12;

      assign clk_3 = (count_3==0) ? 1'b1:1'b0;
      assign clk_12 = (count_12==0) ? 1'b1 : 1'b0;

      always @ (posedge clk or negedge rst_clk)begin
            if(!rst_clk)begin
                  clk_2<=0;
            end
            else begin
                  clk_2<=~clk_2;
            end
      end

      always @ (posedge clk or negedge rst_clk)begin
            if(!rst_clk)begin
                  count_3<=2'b10;
            end
            else begin
                  if(count_3==2'd2) count_3<=0;
                  else  count_3<=count_3+1;
            end
      end

      always @ (posedge clk or negedge rst_clk)begin
            if(!rst_clk)begin
                  count_12<=4'd11;
            end
            else begin
                  if(count_12==4'd11)     count_12<=0;
                  else  count_12<=count_12+1;
            end
      end

endmodule
```

```
module proc_qam
#(
      parameter   wid_count = 4
)

(
      input       clk,
      input       rst_n,
      input       ready_mapper,
      input       ready_zero,
      input       ready_filter,
      output      reg                           sel_zero_pad,
      output                                          ce_shift,
      output      [wid_count-1:0]   sel_carrier
);
      reg   [wid_count-1:0]   count_zero;
      reg   [wid_count-1:0]   count_carrier;

      assign      ce_shift = ready_zero;
      assign      sel_carrier = count_carrier;

//////////////////////////////////////////////////////////////
//Khoi tao bo dem
```

```
      always @ (posedge clk or negedge rst_n)begin    //Bo dem cua khoi
zero_padder
            if(!rst_n)begin
                  count_zero<=0;
            end
            else if(ready_mapper)begin
                  count_zero<=count_zero+1;
            end
      end

      always @ (posedge clk or negedge rst_n)begin    //Bo dem cua khoi
multiplier_carrier
            if(!rst_n)begin
                  count_carrier<=4'b0;
            end
            else if(ready_filter)begin
                  count_carrier<=count_carrier+1;
            end
      end
////////////////////////////////////////////////////////////////
//Dieu khien tin hieu sel_zero_pad
      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  sel_zero_pad<=1'b1;
            end
            else begin
                  if((count_zero==0)&&(ready_mapper)) sel_zero_pad=1'b0;
                  else  sel_zero_pad<=1'b1;
            end
      end

endmodule
```

5.2 Khối thu

5.2.1. Cắt pilot – Cut Pilot

```
Trong CD
```

5.2.2. Nhân sóng mang – Carry multiplier

```
Đã trình bày bên khối phát
```

5.2.3. Bộ lọc cos nâng – Raised cosin filter

```
Đã trình bày bên khối phát
```

5.2.4. Lấy mẫu – Sampling

```
module      sampling
#(
      parameter width=16
)
(
      input       [width-1:0] data_in,
      input                                 clk,
      input                                 rst_n,
      input                                 start,
      output      [width-1:0] data_out,
      output      reg               ready
);
      reg         [width-1:0] data;

      assign data_out=data;

      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  data<=0;
                  ready<=0;
            end
            else if(start)begin
                  ready<=1'b1;
                  data<=data_in;
            end
      end
endmodule
```

5.2.5. Vòng khóa pha – Phase lock loop

Dưới đây là 5 file top module lớn nhất trong tổng số 20 file .v, các file được lưu trong đĩa CD

File Top Phase Lock Loop

```
module      top_pll
#(
      parameter   width_data = 16,
      parameter   width_reg   =18,
      parameter   COUNT_WIDTH = 4,
      parameter kp=0.026,//0.0000011010100111 6 7 9 11 14 15 16
      parameter ki=0.00069,//0.00000000001011010011
      parameter k =0.02531//kp-ki=0.0000011001111010 6 7 10 11 12 13 15
)
(
      input       [width_data-1:0]  x_in,
      input       [width_data-1:0]  y_in,
      input                                          clk,rst_n,
      input                                          start,
      output      [width_data-1:0]  x_out,
      output      [width_data-1:0]  y_out,
      output                                         ready
);
      wire        [width_data-1:0]  phi_ht;//Phi hoi tiep
      wire        [width_data-1:0]  x_latch,y_latch,phi_latch;
      wire        [width_reg-1:0]   x_vec,y_vec;
      wire        [width_data-1:0]  phi_err,phi_right,phi_right_out;
      wire        [width_data-1:0]  phi_temp,teta_temp,e_temp;//hoi tiep
cua pll_proc
      wire        [width_data-1:0]  e_o,teta_o;//Output cua khoi pll_proc
      wire        [width_data-1:0]  x_appr,y_appr;
      wire        ready_latch;
      wire        ready_vec;
      wire        ready_rot;
      wire        ce_rot,ce_vec;


latch_data  #(.width(width_data))
      latch_data_inst(
                  .x_in       (x_in),
                  .y_in       (y_in),
                  .phi_in     (phi_ht),
                  .e_in       (e_o),
                  .teta_in    (teta_o),
                  .phi_right_in(phi_right),
                  .x_i        (x_appr),
                  .y_i        (y_appr),
                  .start      (start),
                  .clk        (clk),
                  .rst_n      (rst_n),
                  .x_out      (x_latch),
                  .y_out      (y_latch),
                  .phi_out (phi_latch),
                  .x_o        (x_out),
                  .y_o        (y_out),
                  .ce_vec     (ce_vec),
                  .ce_rot     (ce_rot),
```

```
                    .phi         (phi_temp),
                    .teta        (teta_temp),
                    .e                (e_temp),
                    .phi_right_out(phi_right_out),
                    .ready       (ready)
      );

vec_cordic_top #(
                                    .COUNT_WIDTH      (COUNT_WIDTH),
                                    .WIDTH_IN         (width_data),
                                    .WIDTH_OUT        (width_reg)
                                )
      cordic_to_vec(
                                .clk        (clk),
                                .start      (ce_vec),
                                .rst_n      (rst_n),
                                .x_in       (x_latch),
                                .y_in       (y_latch),
                                .z_in       (phi_latch),
                                .x_out      (x_vec),
                                .y_out      (y_vec),
                                .ready      (ready_vec)
      );

top_cordic_rot #(
                                    .COUNT_WIDTH      (COUNT_WIDTH),
                                    .WIDTH                  (width_data),
                                    .WIDTH_WIRE       (width_reg)
                                )
      cordic_to_rot(
                                .clk        (clk),
                                .start      (ce_rot),
                                .rst_n      (rst_n),
                                .x_in       (x_vec),
                                .y_in       (y_vec),
                                .x_out(),
                                .y_out(),
                                .z_out      (phi_err),
                                .ready      (ready_rot)
      );

appr_cordic #(
                                .width_in   (width_reg),
                                .width_out  (width_data)
                           )
      appr_cordic_inst(
                                .x_in(x_vec),
                                .y_in(y_vec),
                                .start(ready_vec),
                                .clk(clk),
                                .rst_n(rst_n),
                                .x_out(x_appr),
                                .y_out(y_appr),
                                .phi_right(phi_right)
      );

pll_proc #(
                           .kp   (kp),
                           .ki   (ki),
                           .k    (k),
                           .width(width_data)
```

```
                  )
      pll_proc_inst(
                        .phi_err          (phi_err),
                        .phi_right  (phi_right_out),
                        .phi_in           (phi_temp),
                        .e_in             (e_temp),
                        .teta_in          (teta_temp),
                        .clk              (clk),
                        .rst_n            (rst_n),
                        .start            (ready_rot),
                        .e_out            (e_o),
                        .teta_out   (teta_o),
                        .phi_out          (phi_ht)
      );

endmodule
```

Module chốt dữ liệu: Điều khiển dữ liệu đồng bộ với nhau

```
module latch_data
#(
      parameter width=16
)
(
      input       [width-1:0] x_in,
      input       [width-1:0] y_in,
      input       [width-1:0] phi_in,e_in,teta_in,phi_right_in,
      input       [width-1:0] x_i,y_i,//Giu tin hieu x_output va y_output
den clk thu 37 roi dua ra
      input                                 start,clk,rst_n,
      output      [width-1:0] x_out,
      output      [width-1:0] y_out,
      output      [width-1:0] phi_out,
      output      [width-1:0] x_o,y_o,//Dieu chinh clk cho dau ra x,y cua
top_pll
      output      reg               ce_vec,ce_rot,
      output      reg [width-1:0] phi,teta,e,phi_right_out,
      output      reg               ready
);
      reg         [5:0] count;
      reg         [width-1:0] x_temp,y_temp,phi_temp,x_io_temp,y_io_temp;
      reg         [width-1:0] e_tp,phi_tp,teta_tp;//Lay dau vao
e_in,teta_in,phi_in

      assign x_out=x_temp;
      assign y_out=y_temp;
      assign phi_out=phi_temp;
      assign x_o=x_io_temp;
      assign y_o=y_io_temp;
//Dung
      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  count<=6'd0;
            end
            else if(start)begin
                  if(count==6'd36)  count<=6'd0;
                  else  count<=count+6'd1;
            end
      end

      always @ (posedge clk or negedge rst_n)begin
```

```
            if(!rst_n)begin
                  x_temp<=0;
                  y_temp<=0;
                  phi_temp<=0;
            end
            else if((start)&&(count==0))begin
                  x_temp<=x_in;
                  y_temp<=y_in;
                  phi_temp<=phi_in;
            end
      end
//
      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  x_io_temp<=0;
                  y_io_temp<=0;
            end
            else if((start)&&(count==6'd36)) begin
                  x_io_temp<=x_i;
                  y_io_temp<=y_i;
            end
      end

//Dieu khien ce cua 2 top module VEC va ROT
      always @ (posedge clk or negedge rst_n)begin    //CE_VEC
            if(!rst_n)begin
                  ce_vec<=0;
            end
            else if(start) begin
                  if((count>=6'd0)&&(count<=6'd15)) ce_vec<=1'b1;
                  else  ce_vec<=1'b0;
            end
      end

      always @ (posedge clk or negedge rst_n)begin//CE_ROT
            if(!rst_n)begin
                  ce_rot<=0;
            end
            else if(start)begin
                  if((count>=6'd17)&&(count<=6'd32))  ce_rot<=1'b1;
                  else  ce_rot<=1'b0;
            end
      end

      always @ (posedge clk or negedge rst_n)begin// dieu khien
e',phi',teta'
            if(!rst_n)begin
                  e_tp<=0;
                  teta_tp<=0;
                  phi_tp<=0;
                  ready<=0;
            end
            else if((start)&&(count==6'd36))begin
                  e_tp<=e_in;
                  phi_tp<=phi_in;
                  teta_tp<=teta_in;
                  ready<=1'b1;
            end
      end

      always @ (posedge clk or negedge rst_n)begin
```

```
            if(!rst_n)begin
                  e<=0;
                  teta<=0;
                  phi<=0;
                  phi_right_out<=0;
            end
            else if((start)&&(count==6'd34))begin
                  e<=e_tp;
                  phi<=phi_tp;
                  teta<=teta_tp;
                  phi_right_out<=phi_right_in;
            end
      end

endmodule
```

Module: Xoay tọa độ (x_in,y_in) thành (x_out,y_out) theo góc z

```
module vec_cordic_top
#(
  parameter COUNT_WIDTH = 4,   // Counter width
  parameter WIDTH_IN = 16,          // Counter width
  parameter WIDTH_OUT =18            //Mo rong bit dau
)
(
  input                              clk,
  input                              start,
  input                              rst_n,
  input signed [WIDTH_IN-1 : 0]     x_in,
  input signed [WIDTH_IN-1 : 0]     y_in,
  input signed    [WIDTH_IN-1 : 0]     z_in,
  output reg   [WIDTH_OUT-1 : 0]   x_out,
  output reg   [WIDTH_OUT-1 : 0]   y_out,
  output                ready
);
  wire                               sign;
  wire         [COUNT_WIDTH-1 : 0] shift_bit;
  wire                               mux_sel;
  wire         [WIDTH_OUT-1 : 0]   x_out_wire;
  wire            [WIDTH_OUT-1 : 0]   y_out_wire;

  always    @ (posedge clk or negedge rst_n)begin     //Giu gia tri
x_out,y_out trong 16 chu ki
      if(!rst_n)begin
            x_out<=0;
            y_out<=0;
      end
      else if (shift_bit==4'd15)begin
            x_out<=x_out_wire;
            y_out<=y_out_wire;
      end
  end

  // CORDIC
  vec_eda_cordic
  #(
    .WIDTH_IN     (WIDTH_IN),
       .COUNT_WIDTH  (COUNT_WIDTH),
       .WIDTH_OUT (WIDTH_OUT)
  )
  eda_cordic_inst
```

```
  (
    .x_in          (x_in),
    .y_in          (y_in),
       .z_in                    (z_in),
       .clk           (clk),
    .ce            (start),
    .rst_n         (rst_n),
    .mux_sel       (mux_sel),
    .shift_bit     (shift_bit),
    .sign_in       (sign),
    .x_out         (x_out_wire),
    .y_out         (y_out_wire),
    .sign_out      (sign)
  );

  // CTRL
  vec_cordic_ctrl
  #(
    .COUNT_WIDTH  (COUNT_WIDTH)
  )
  eda_cordic_ctrl_inst
  (
    .clk           (clk),
    .ce            (start),
    .rst_n         (rst_n),
    .shift_bit     (shift_bit),
    .mux_ctrl      (mux_sel),
       .ready                   (ready)
  );
endmodule
```

Module: Xoay tọa độ (x,y) về trục hoành vừa đưa ra góc vừa xoay

```
module top_cordic_rot
#(
  parameter COUNT_WIDTH = 4,   // Counter width
  parameter WIDTH = 16,         // do rong du lieu cua goc
  parameter WIDTH_WIRE =18          //data width
)
(
  input                               clk,
  input                               start,
  input                               rst_n,
  input signed [WIDTH_WIRE-1 : 0]   x_in,
  input signed [WIDTH_WIRE-1 : 0]   y_in,
  output reg   [WIDTH_WIRE-1 : 0]   x_out,
  output reg   [WIDTH_WIRE-1 : 0]   y_out,
  output reg      [WIDTH-1 : 0]                 z_out,
  output                ready
);
  wire                                sign;
  wire         [COUNT_WIDTH-1 : 0]  shift_bit;
  wire                                mux_sel;
  wire                  [WIDTH_WIRE-1 : 0]   x_out_wire;
  wire            [WIDTH_WIRE-1 : 0]   y_out_wire;
  wire                  [WIDTH-1 : 0]                 z_out_wire;
  wire                  [WIDTH-1 : 0]
      z_out_wire_temp;//Chuyen z_out_wire ve [0-2pi]

  assign    z_out_wire_temp = (z_out_wire[WIDTH-1])   ?
      (z_out_wire+16'd25735)  :     (z_out_wire);//pi->25735(16bit)
```

```
  always    @ (posedge clk or negedge rst_n)begin     //Giu gia tri
x_out,y_out trong 16 chu ki
      if(!rst_n)begin
            x_out<=0;
            y_out<=0;
            z_out<=0;
      end
      else if (shift_bit==4'd15)begin
            x_out<=x_out_wire;
            y_out<=y_out_wire;
            z_out<=z_out_wire_temp;
      end
  end

  // CORDIC
  rot_eda_cordic
  #(
    .WIDTH        (WIDTH),
       .COUNT_WIDTH  (COUNT_WIDTH),
       .WIDTH_WIRE      (WIDTH_WIRE)
  )
  rot_cordic_inst
  (
    .x_in         (x_in),
    .y_in         (y_in),
       .clk          (clk),
    .ce           (start),
    .rst_n        (rst_n),
    .mux_sel      (mux_sel),
    .shift_bit    (shift_bit),
    .sign_in      (sign),
    .x_out        (x_out_wire),
    .y_out        (y_out_wire),
       .z_out                (z_out_wire),
    .sign_out     (sign)
  );

  // CTRL
  rot_cordic_ctrl
  #(
    .COUNT_WIDTH  (COUNT_WIDTH)
  )
  rot_cordic_ctrl_inst
  (
    .clk          (clk),
    .ce           (start),
    .rst_n        (rst_n),
    .shift_bit    (shift_bit),
    .mux_ctrl     (mux_sel),
       .ready                (ready)
  );
endmodule
```

Module: Làm tròn (x,y) về điểm gần nhất trên chòm sao

```
/*
      Lam tron (I,Q)->(I',Q') dua ve diem gan nhat tren chom sao
*/
module appr_cordic
```

```
#(
      parameter width_in = 18,      //data width_in
      parameter width_out = 16      //data width_out
)
(
      input signed      [width_in-1:0]    x_in,
      input signed      [width_in-1:0]    y_in,
      input                                                   start,
      input                                                   clk,
      input                                                   rst_n,
      output                  [width_out-1:0] x_out,
      output                  [width_out-1:0] y_out,
      output                  [width_out-1:0] phi_right
);

      reg               [width_out-1:0]
buffer_x1,buffer_x2,buffer_y1,buffer_y2,buffer_phi1,buffer_phi2;//Dem du
lieu de dam bao chac chan khong bi mat sau 16 chu ki
      reg               [width_out-1:0] x_temp,y_temp,phi_temp;

      assign      x_out = buffer_x2;
      assign      y_out = buffer_y2;
      assign      phi_right=buffer_phi2;

      always @ (x_in)begin// lam tron x_in
            if((x_in<-18'd8192)&&(x_in[width_in-1])) begin  //<-2 va x_in
am
                  x_temp=-16'd12288;//=-3
            end
            else if((x_in>=-18'd8192)&&(x_in[width_in-1])) begin//-2<=x_in
&& x_in la so am
                  x_temp=-16'd4096;//=-1
            end
            else if((x_in>=18'd0) && (x_in<18'd8192) && (~x_in[width_in-
1]))begin//0<=x_in<2 va x_in duong
                  x_temp=16'd4096;//=1
            end
            else begin
                  x_temp=16'd12288;//=3
            end
      end

      always @ (y_in)begin// lam tron y_in
            if((y_in<-18'd8192)&&(y_in[width_in-1])) begin  //<-2 va y_in
am
                  y_temp=-16'd12288;//=-3
            end
            else if((y_in>=-18'd8192)&&(y_in[width_in-1])) begin//-2<=y_in
va y_in am
                  y_temp=-16'd4096;//=-1
            end
            else if((y_in>=18'd0) && (y_in<18'd8192) && (~y_in[width_in-
1]))begin//0<=y_in<2 va y_in duong
                  y_temp=16'd4096;//=1
            end
            else begin
                  y_temp=16'd12288;//=3
            end
      end

      always @ (x_temp or y_temp)begin
```

```
            if(((x_temp==16'd4096)&&(y_temp==16'd4096)) ||
((x_temp==16'd12288)&&(y_temp==16'd12288)))begin
                  phi_temp=16'd3217;
            end
            else if(((x_temp==-16'd4096)&&(y_temp==-16'd4096)) ||
((x_temp==-16'd12288)&&(y_temp==-16'd12288)))begin
                  phi_temp=16'd16085;
            end
            else if(((x_temp==-16'd4096)&&(y_temp==16'd4096)) ||
((x_temp==-16'd12288)&&(y_temp==16'd12288)))begin
                  phi_temp=16'd9651;
            end
            else if(((x_temp==16'd4096)&&(y_temp==-16'd4096)) ||
((x_temp==16'd12288)&&(y_temp==-16'd12288)))begin
                  phi_temp=16'd22519;
            end
            else if ((x_temp==16'd12288)&&(y_temp==16'd4096))begin
                  phi_temp=16'd1318;
            end
            else if ((x_temp==-16'd12288)&&(y_temp==-16'd4096))begin
                  phi_temp=16'd14186;
            end
            else if ((x_temp==16'd4096)&&(y_temp==16'd12288))begin
                  phi_temp=16'd5116;
            end
            else if ((x_temp==-16'd4096)&&(y_temp==-16'd12288))begin
                  phi_temp=16'd17984;
            end
            else if ((x_temp==-16'd4096)&&(y_temp==16'd12288))begin
                  phi_temp=16'd7752;
            end
            else if ((x_temp==16'd4096)&&(y_temp==-16'd12288))begin
                  phi_temp=16'd20619;
            end
            else if ((x_temp==-16'd12288)&&(y_temp==16'd4096))begin
                  phi_temp=16'd11550;
            end
            else begin
                  phi_temp=16'd24418;
            end
      end

      always @ (posedge clk or negedge rst_n)begin
            if(!rst_n)begin
                  buffer_x1<=0;
                  buffer_x2<=0;
                  buffer_y1<=0;
                  buffer_y2<=0;
                  buffer_phi1<=0;
                  buffer_phi2<=0;
            end
            else if(start)begin
                  buffer_x1<=x_temp;
                  buffer_x2<=buffer_x1;
                  buffer_y1<=y_temp;
                  buffer_y2<=buffer_y1;
                  buffer_phi1<=phi_temp;
                  buffer_phi2<=buffer_phi1;
            end
      end
```

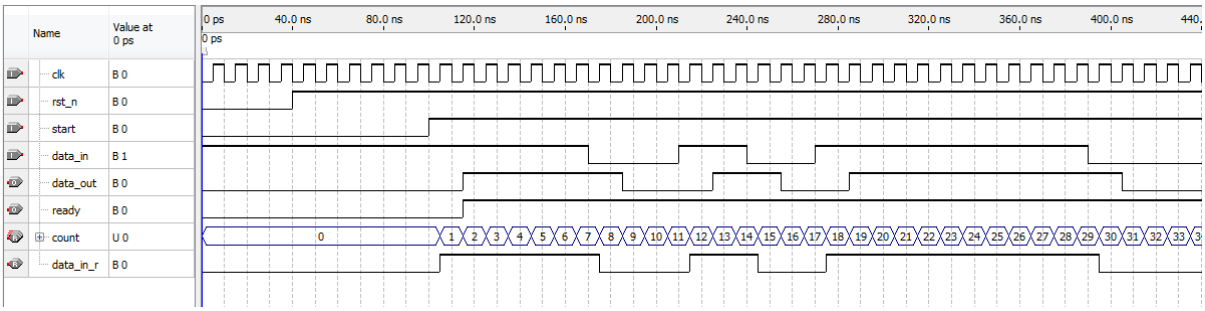
```
endmodule
```

Phần 6. Kế hoạch kiểm tra

Phần này sử dụng công cụ mô phỏng VERTOR WAVEFORM của ALTERA để mô phỏng cho các khối được nếu trong phần 5.

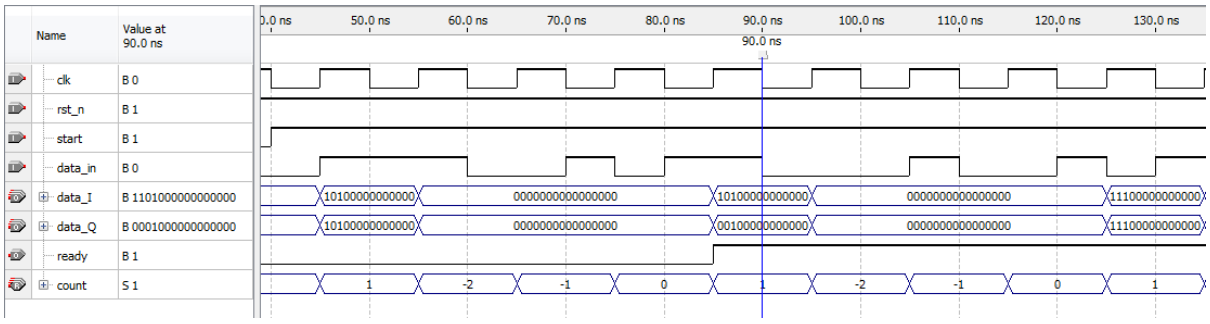
6.1 Phần phát

6.1.1. Bộ Cyclic Prefix



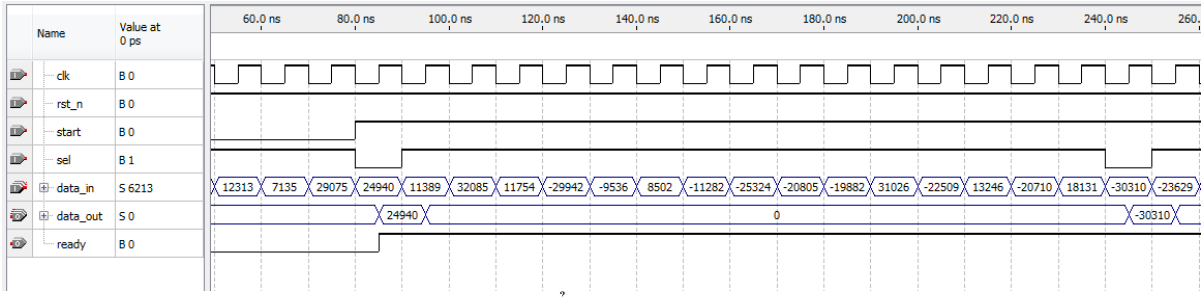
Hình 6.1 Kiểm tra bộ cyclic prefix.

6.1.2. Bộ mapper



Hình 6.2 Kiểm tra bộ mapper.

1.1.1. Bộ zero pad

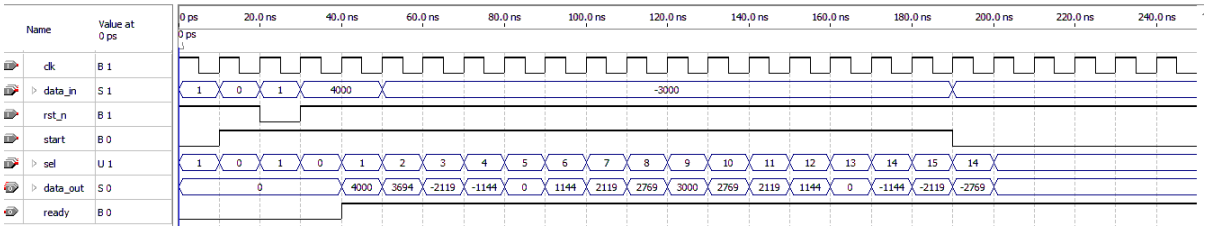


Hình 6.3 Kiểm tra bộ zero pad.

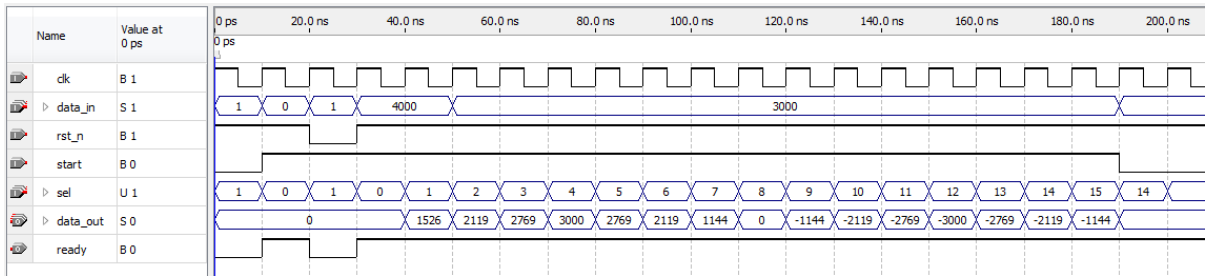
6.1.3. *Bộ raised cosin filter*

Hình 6.4 Kiểm tra bộ raised cosin filter.

1.1.2. *Bộ carry multiplier*

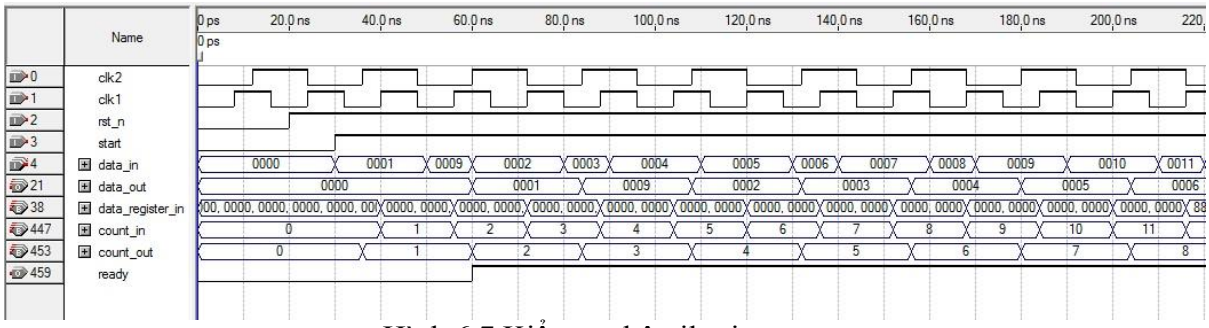


Hình 6.5 Kiểm tra bộ carry cos multiplier.



Hình 6.6 Kiểm tra bộ carry sin multiplier.

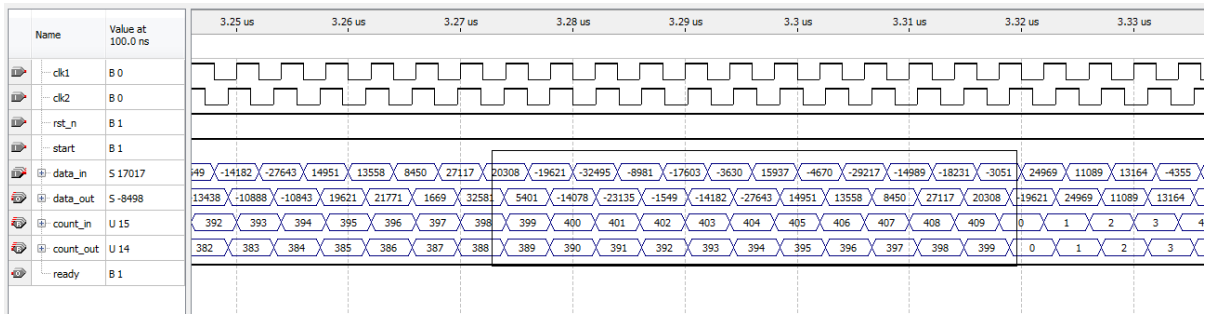
1.1.3. *Bộ pilot inserter*



Hình 6.7 Kiểm tra bộ pilot inserter

6.2 Phần thu

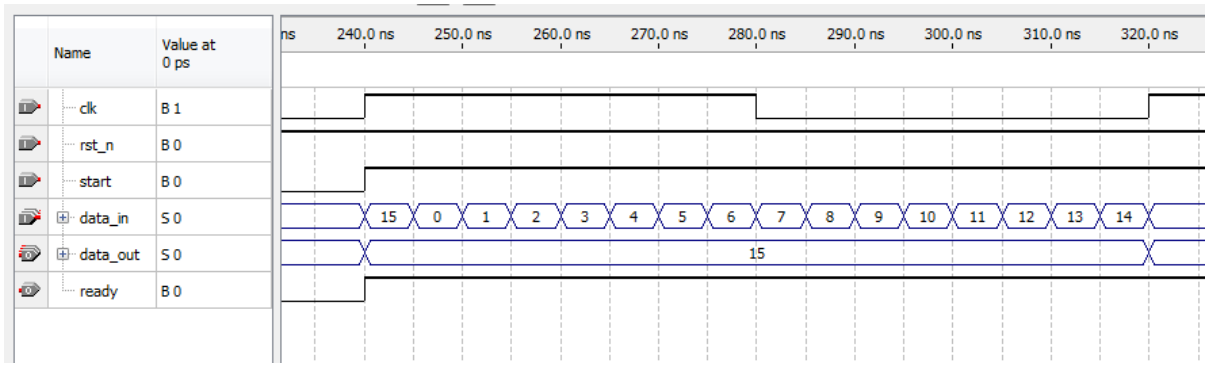
1.1.4. *Bộ cắt pilot*



1.1.5. *Bộ nhân song mang (giống ở trên)*

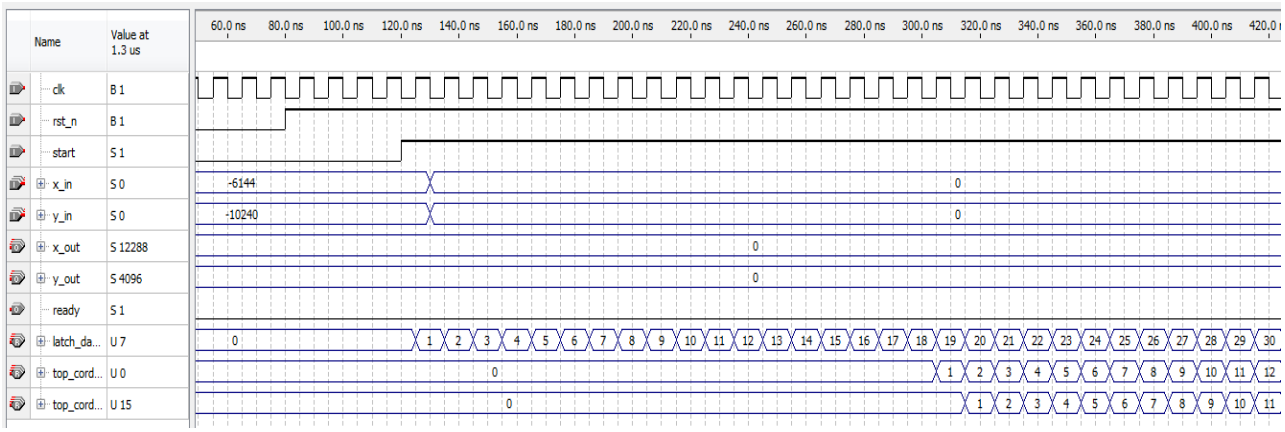
1.1.6. *Bộ lọc (Như phần phát)*

1.1.7. *Bộ lấy mẫu*



Hình 6.8 Kiểm tra bộ sampling

1.1.8. Phase Lock Loop



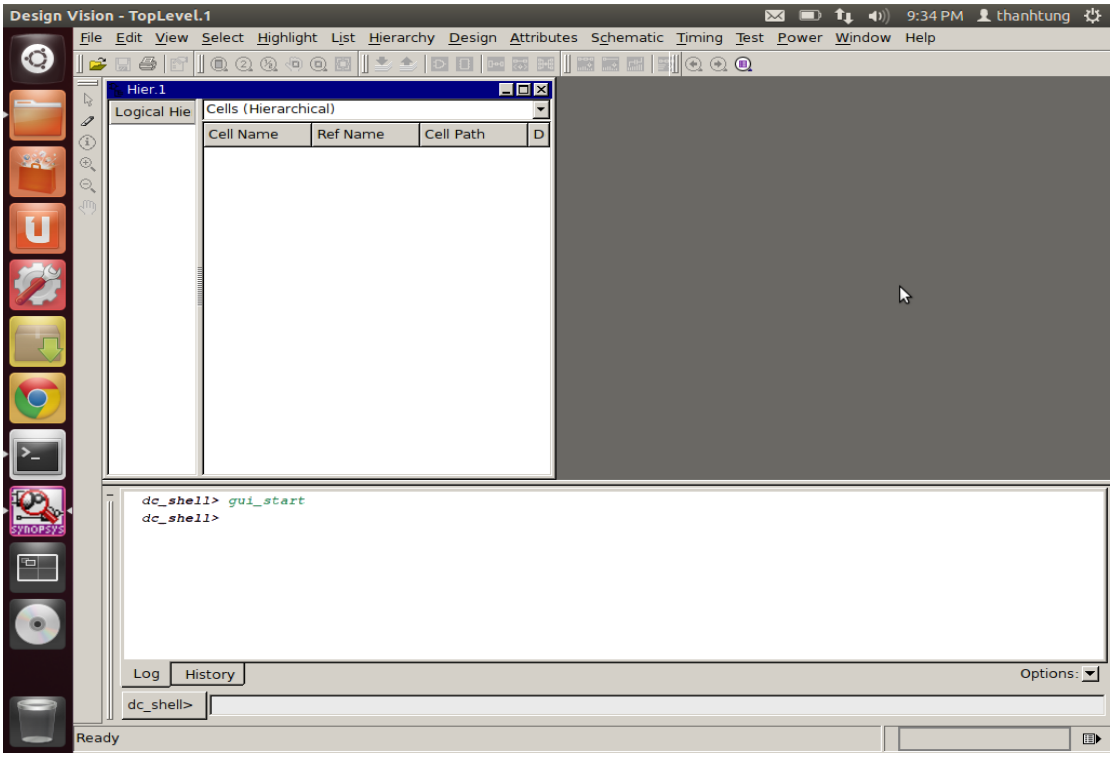
Phần 7. Tổng hợp mạch bằng phần mềm SYNOPSYS

Phần này sử dụng phần mềm SYNOPSYS chạy trên nên LINUX để tổng hợp thành mạch thật, có thể đi đặt ở các nhà máy. Phần này setup các thông số về thời gian cho mạch như: input delay, output delay, t_setup, t_hold…

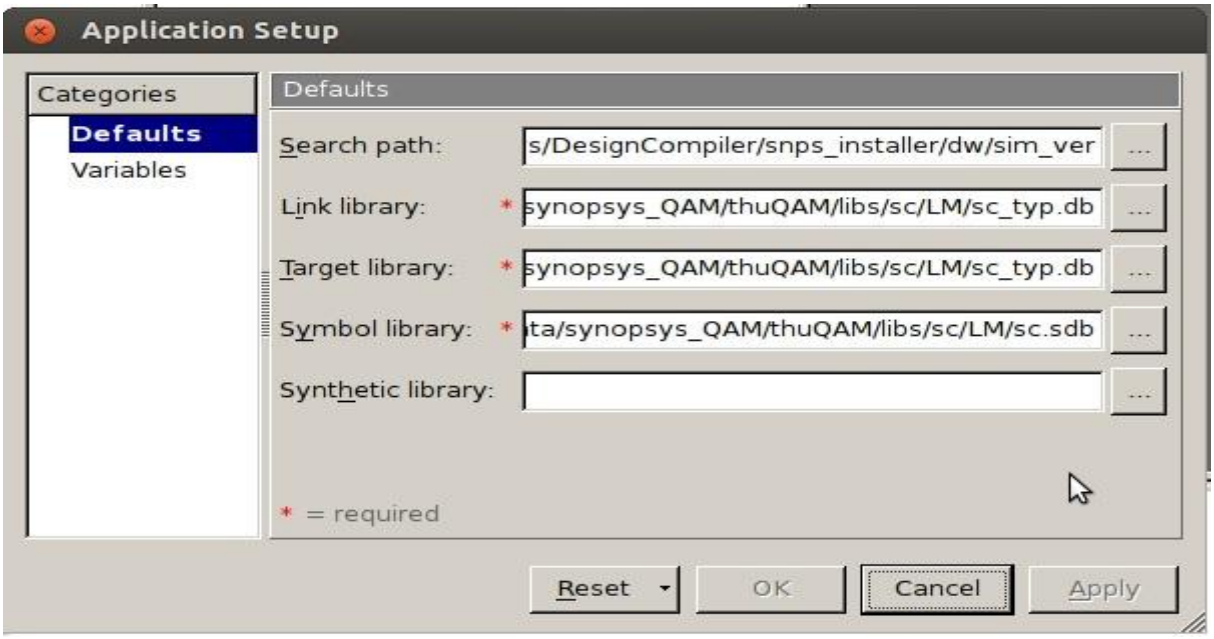
7.1 Các bước thực hiện

Bước 1: Từ cửa sổ terminal, gõ lệnh: dc_gui

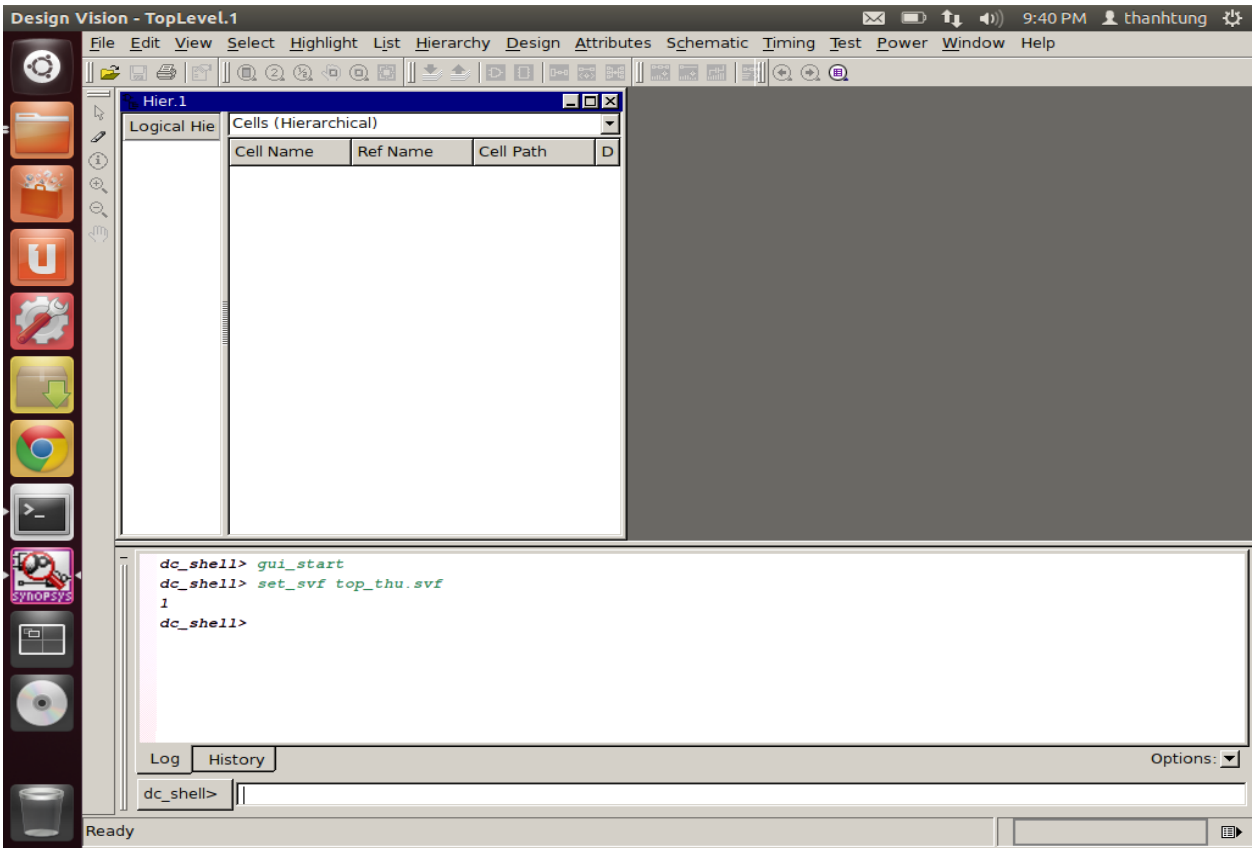
Màn hình giao diện của chương trình:



Bước 2: Thiết lập các thư viện

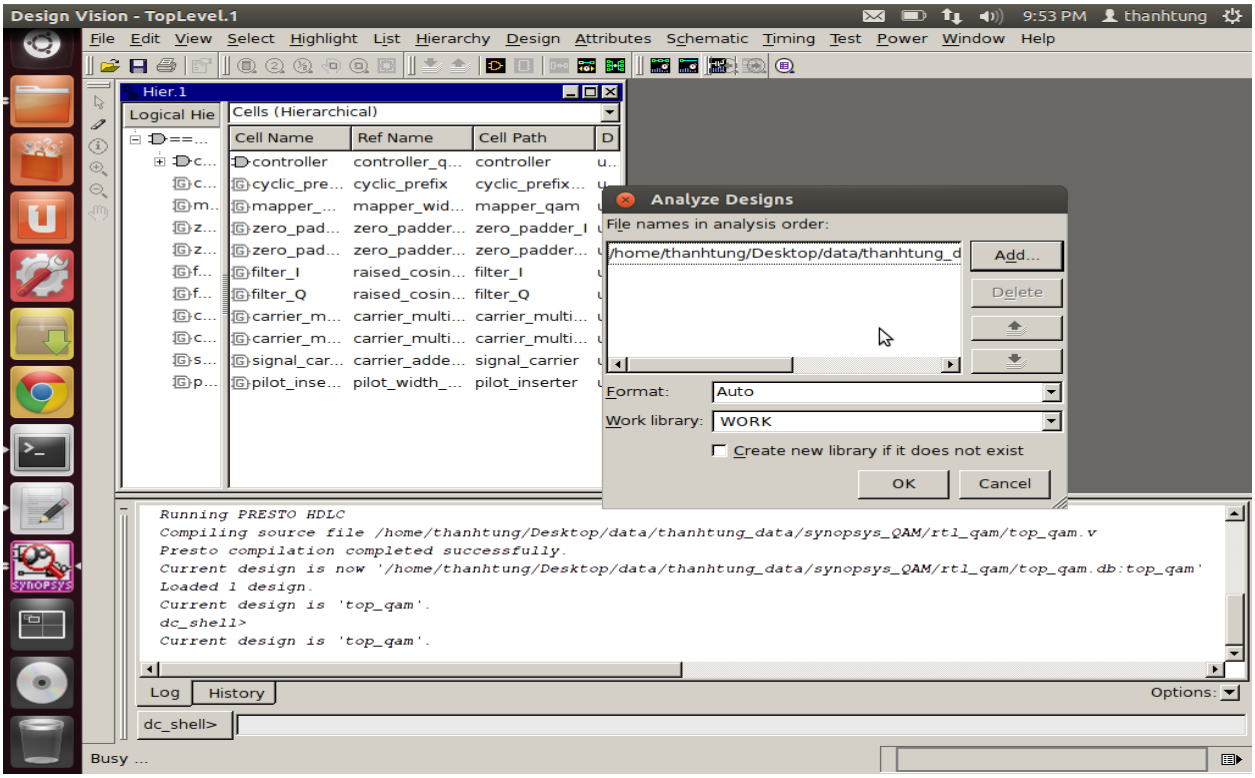


Bước 3: Tạo file lưu tiến trình của quá trình tổng hợp bằng lệnh: set_svf top_thu.svf

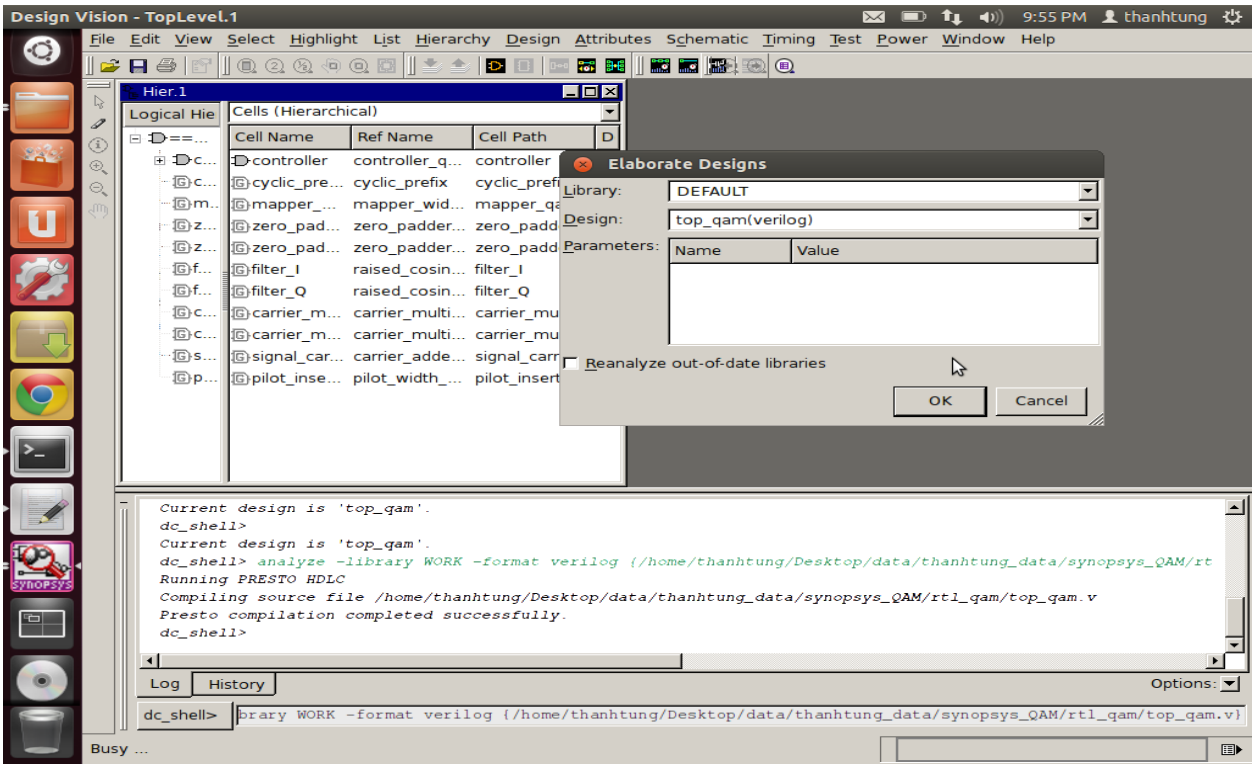


Bước 4: đọc file top module: *File → Read → "top_module.v"*

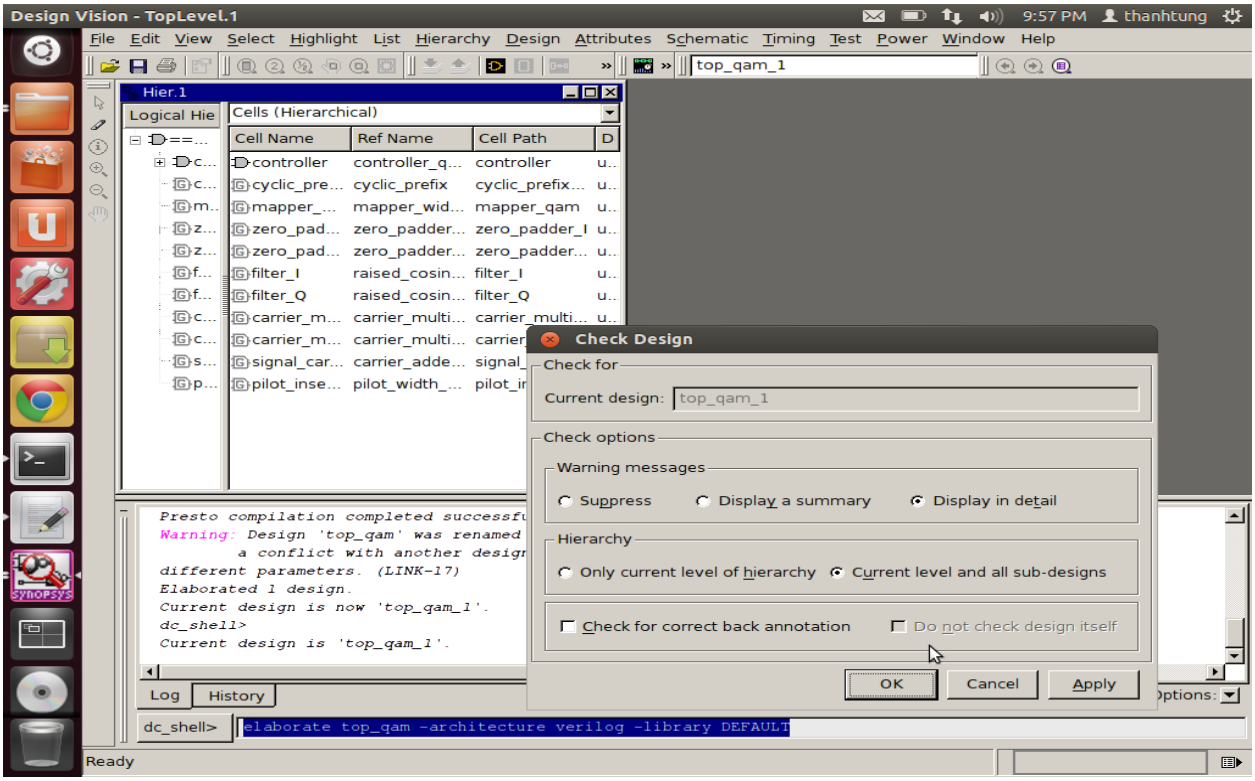
Bước 5: Analyze: *file →analyze → Add→ < top_module.v >→OK*



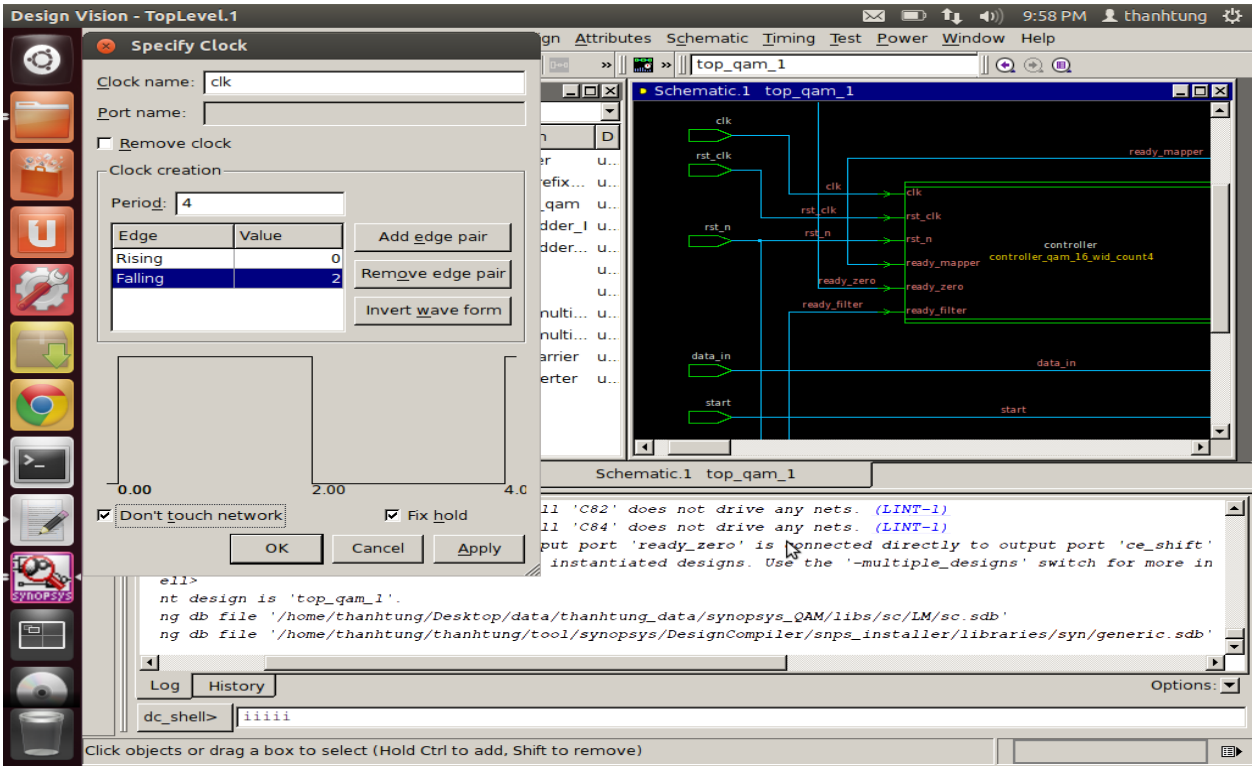
Bước 6: Elaborate



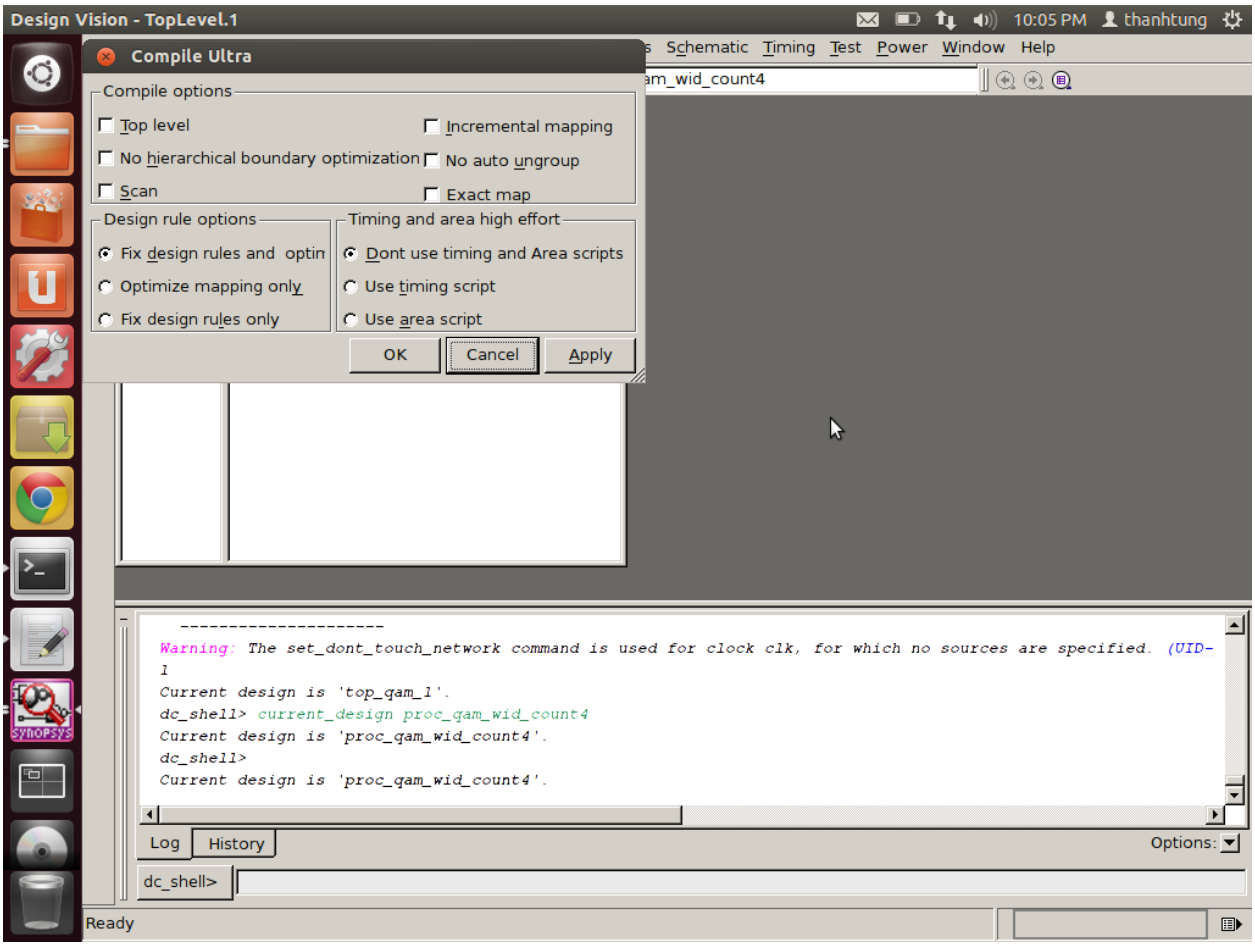
Bước 7: Check Design*: Design→Check design→OK*



Bước 8: Set clock contraints:



Bước 9: Optimized the design: Design → *Compile Ultra...*



7.2 Kết quả, nhận xét

7.2.1. Các lệnh sử dụng

```
gui_start
set_svf top.svf
read_file -format verilog
{/home/thanhtung/Desktop/data/thanhtung_data/synopsys_QAM/rtl_
qam/top_qam.v}
analyze -library WORK -format verilog
{/home/thanhtung/Desktop/data/thanhtung_data/synopsys_QAM/rtl_
qam/top_qam.v}
elaborate top_qam -architecture verilog -library DEFAULT
uplevel #0 check_design
create_clock -name "clk" -period 4 -waveform { 0 2  }
set_dont_touch_network  [ find clock clk ]
set_fix_hold  [ find clock clk]
compile -exact_map
current_design proc_qam_wid_count4
compile_ultra
```

```
uplevel #0 { report_design }
uplevel #0 { report_hierarchy }
uplevel #0 { report_resources -hierarchy }
uplevel #0 { report_constraint -significant_digits 2 }
uplevel #0 { report_reference -nosplit }
uplevel #0 { report_port }
uplevel #0 { report_cell }
uplevel #0 { report_clock -nosplit }
uplevel #0 { report_area -nosplit }
uplevel #0 { report_compile_options -nosplit }
uplevel #0 { report_power -analysis_effort low }
```

7.2.2. Các file report của quá trình tổng hợp

7.2.2.1. Area

```
****************************************
Design : top_qam_1
****************************************
Combinational area:          0.000000
Noncombinational area:       0.000000
Net Interconnect area:      21.055969
Total cell area:             0.000000
Total area:                 21.055969
****************************************
Design : top_qam_1
****************************************
Combinational area:       19578.500000
Noncombinational area:    57968.500000
Net Interconnect area:    22762.068830
Total cell area:          77547.000000
Total area:               100309.068830
****************************************
Design : proc_qam_wid_count4
****************************************
Combinational area:         27.000000
Noncombinational area:      59.500000
Net Interconnect area:       5.698282

Total cell area:            86.500000
Total area:                 92.198282
```

7.2.2.2. Nets

```
Total 340 nets         8236        340    1021.45         56.65
8576
Maximum                6715          1    1008.98         30.72
6716
Average               24.22       1.00       3.00          0.17
25.22
```

7.2.2.3. Power

```
****************************************
Design : top_qam_1
****************************************
Design        Wire Load Model            Library
------------------------------------------------
top_qam_1              ForQA             cb13fs120_tsmc_max
Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW
****************************************
Design : top_qam_1
****************************************
Design        Wire Load Model            Library
------------------------------------------------
top_qam_1              140000            cb13fs120_tsmc_max
raised_cosin_filter_width_data16_No_reg16_0
                       8000              cb13fs120_tsmc_max
raised_cosin_filter_width_data16_No_reg16_1
                       8000              cb13fs120_tsmc_max
carrier_multi_sin_width_sym16_width_sel4
                       8000              cb13fs120_tsmc_max
carrier_multi_cos_width_sym16_width_sel4
                       8000              cb13fs120_tsmc_max
pilot_width_data16     70000             cb13fs120_tsmc_max


Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
```

```
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  = 151.4091 uW   (14%)
  Net Switching Power  = 968.7601 uW   (86%)
                         ---------
Total Dynamic Power    =   1.1202 mW  (100%)

Cell Leakage Power     = 404.4937 uW
****************************************
Design : proc_qam_wid_count4
****************************************
Design        Wire Load Model            Library
------------------------------------------------
proc_qam_wid_count4    ForQA             cb13fs120_tsmc_typ


Global Operating Voltage = 1.2
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =  21.3789 uW   (86%)
  Net Switching Power  =   3.5870 uW   (14%)
                         ---------
Total Dynamic Power    =  24.9659 uW  (100%)

Cell Leakage Power     = 198.7400 nW
```

7.2.2.4. Reference

```
Warning: Design 'top_qam_1' has '9' unresolved references.
For more detailed information, use the "link" command.
(UID-341)

****************************************
Report : reference
Design : top_qam_1
Version: D-2010.03-SP5-2
Date   : Wed Nov 21 15:19:55 2012
```

```
****************************************

Attributes:
    b - black box (unknown)
   bo - allows boundary optimization
    d - dont_touch
   mo - map_only
    h - hierarchical
    n - noncombinational
    r - removable
    s - synthetic operator
    u - contains unmapped logic

Reference           Library       Unit Area   Count    Total
Area   Attributes
--------------------------------------------------------------
------------------
carrier_adder                              0.000000          1
0.000000  b
carrier_multi_cos                          0.000000          1
0.000000  b
carrier_multi_sin                          0.000000          1
0.000000  b
controller_qam_16                          0.000000          1
0.000000  b
cyclic_prefix                              0.000000          1
0.000000  b
mapper                                     0.000000          1
0.000000  b
pilot                                      0.000000          1
0.000000  b
raised_cosin_filter                        0.000000          2
0.000000  b
zero_padder                                0.000000          2
0.000000  b
--------------------------------------------------------------
------------------
Total                      9                       references
0.000000

****************************************
Report : reference
Design : top_qam_1
Version: D-2010.03-SP5-2
Date   : Thu Nov 22 09:41:26 2012
****************************************
```

```
Reference           Library       Unit Area   Count    Total
Area   Attributes
-------------------------------------------------------------
------------------
ad01d0              cb13fs120_tsmc_max     5.000000       14
70.000000 r
ah01d0              cb13fs120_tsmc_max     3.000000        8
24.000000 r
an02d1              cb13fs120_tsmc_max     1.250000       18
22.500000
an04d1              cb13fs120_tsmc_max     2.000000        2
4.000000
aoi211d1            cb13fs120_tsmc_max     2.500000        1
2.500000
aor221d1            cb13fs120_tsmc_max     2.500000        1
2.500000
bufbd1              cb13fs120_tsmc_max     1.000000        2
2.000000
carrier_multi_cos_width_sym16_width_sel4         1715.000000
1  1715.000000 h, n
carrier_multi_sin_width_sym16_width_sel4         1681.500000
1  1681.500000 h, n
clk2d2              cb13fs120_tsmc_max     5.500000        1
5.500000
decrq1              cb13fs120_tsmc_max     8.250000       14
115.500000 n
deprq1              cb13fs120_tsmc_max     8.000000        3
24.000000 n
dfcrb1              cb13fs120_tsmc_max     6.000000        1
6.000000 n
dfcrn1              cb13fs120_tsmc_max     5.500000        1
5.500000 n
dfcrq1              cb13fs120_tsmc_max     5.500000       48
264.000000 n
dfprb1              cb13fs120_tsmc_max     6.750000       11
74.250000 n
inv0d0              cb13fs120_tsmc_max     0.750000        1
0.750000
inv0d1              cb13fs120_tsmc_max     0.750000       14
10.500000
inv0d2              cb13fs120_tsmc_max     1.000000        1
1.000000
invbdf              cb13fs120_tsmc_max     4.250000        1
4.250000
mx02d0              cb13fs120_tsmc_max     2.000000       19
38.000000
nd02d0              cb13fs120_tsmc_max     1.000000        3
```

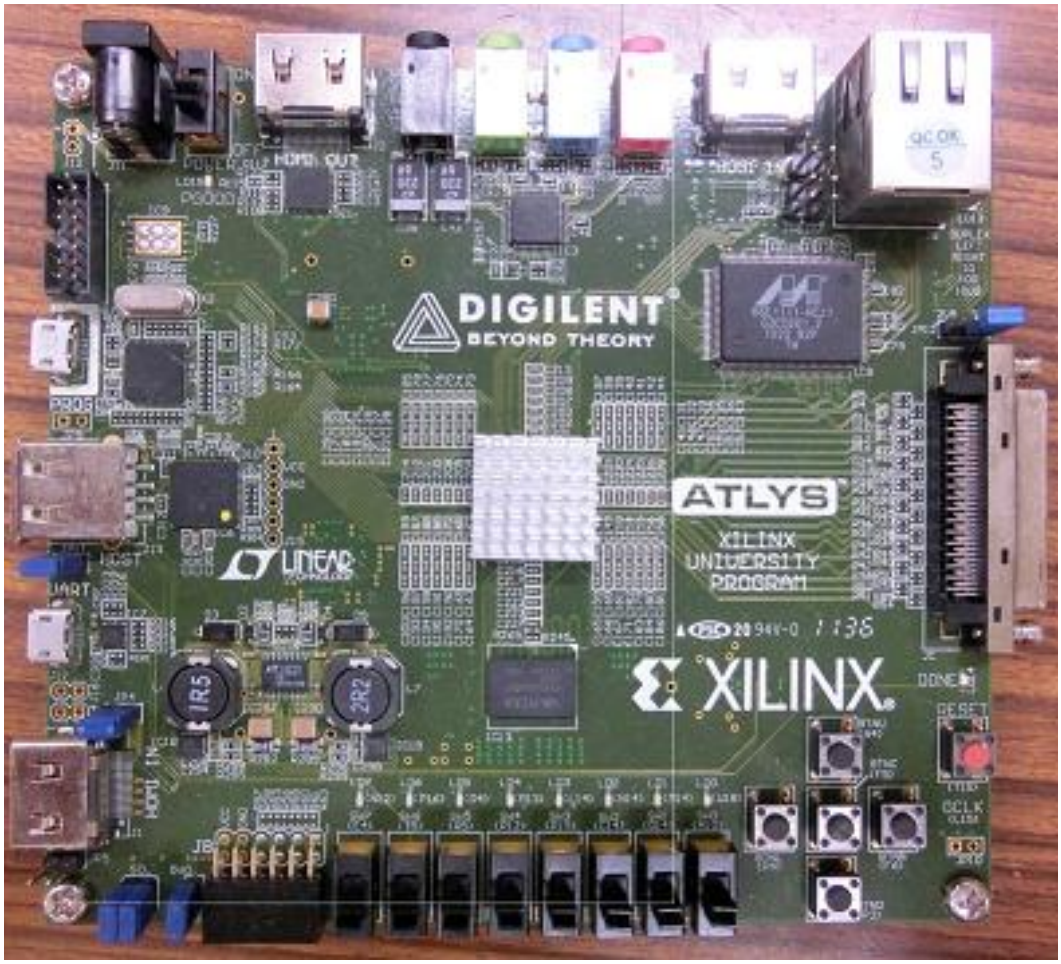
```
3.000000
nd02d1              cb13fs120_tsmc_max     1.000000        7
7.000000
nd03d0              cb13fs120_tsmc_max     1.250000        2
2.500000
nd04d0              cb13fs120_tsmc_max     1.750000        2
3.500000
nd12d0              cb13fs120_tsmc_max     1.250000        1
1.250000
nr02d0              cb13fs120_tsmc_max     1.000000       19
19.000000
nr03d0              cb13fs120_tsmc_max     1.250000        4
5.000000
nr04d0              cb13fs120_tsmc_max     1.500000        3
4.500000
or02d1              cb13fs120_tsmc_max     1.250000        3
3.750000
or03d1              cb13fs120_tsmc_max     1.750000        1
1.750000
or04d1              cb13fs120_tsmc_max     2.000000        2
4.000000
ora21d1             cb13fs120_tsmc_max     2.000000        6
12.000000
ora211d1            cb13fs120_tsmc_max     2.250000        4
9.000000
pilot_width_data16                  63631.250000          1
63631.250000  h, n
raised_cosin_filter_width_data16_No_reg16_0      4914.750000
1  4914.750000 h, n
raised_cosin_filter_width_data16_No_reg16_1      4847.250000
1  4847.250000 h, n
xr03d1              cb13fs120_tsmc_max     4.250000        1
4.250000
------------------------------------------------------------
------------------
Total                      38                      references
77547.000000

****************************************
Report : reference
Design : proc_qam_wid_count4
Version: D-2010.03-SP5-2
Date   : Sun Nov 25 22:13:15 2012
****************************************
Reference          Library       Unit Area   Count    Total
Area   Attributes
------------------------------------------------------------
```

```
------------------
an02d1              cb13fs120_tsmc_typ     1.250000        1
1.250000
deprq1              cb13fs120_tsmc_typ     8.000000        3
24.000000 n
dfcrq1              cb13fs120_tsmc_typ     5.500000        4
22.000000 n
dfprb1              cb13fs120_tsmc_typ     6.750000        2
13.500000 n
inv0d0              cb13fs120_tsmc_typ     0.750000        3
2.250000
inv0d1              cb13fs120_tsmc_typ     0.750000        1
0.750000
mx02d0              cb13fs120_tsmc_typ     2.000000        4
8.000000
nd02d0              cb13fs120_tsmc_typ     1.000000        3
3.000000
nd03d0              cb13fs120_tsmc_typ     1.250000        1
1.250000
nr02d0              cb13fs120_tsmc_typ     1.000000        3
3.000000
nr04d0              cb13fs120_tsmc_typ     1.500000        1
1.500000
ora21d1             cb13fs120_tsmc_typ     2.000000        3
6.000000
------------------------------------------------------------
------------------
Total                     12                      references
86.500000
```

Phần 8. Thực nghiệm

Phần này mô tả việc kiểm tra sản phẩm, chạy thử sản phẩm.Phần này nên trình bày ở dưới dạng ảnh sản phẩm. Bên XILINX có một số công cụ giúp chung ta có thể mô phỏng phần cứng trực tiếp trên Simulink, như vậy chúng ta vừa có thể sử dụng phần cứng được nạp trên kit FPGA vừa có thể sử dụng một số công cụ hiển thị trên Simulink. Chip FPGA mà nhóm thực hiện là SpartanR-6 của XILINX

Board ： ATLYS by DIGILENT co.
FPGA ：Xilinx SpartanR-6 LX45 FPGA in a 324-pin BGA package
Operating frequency：100MHz

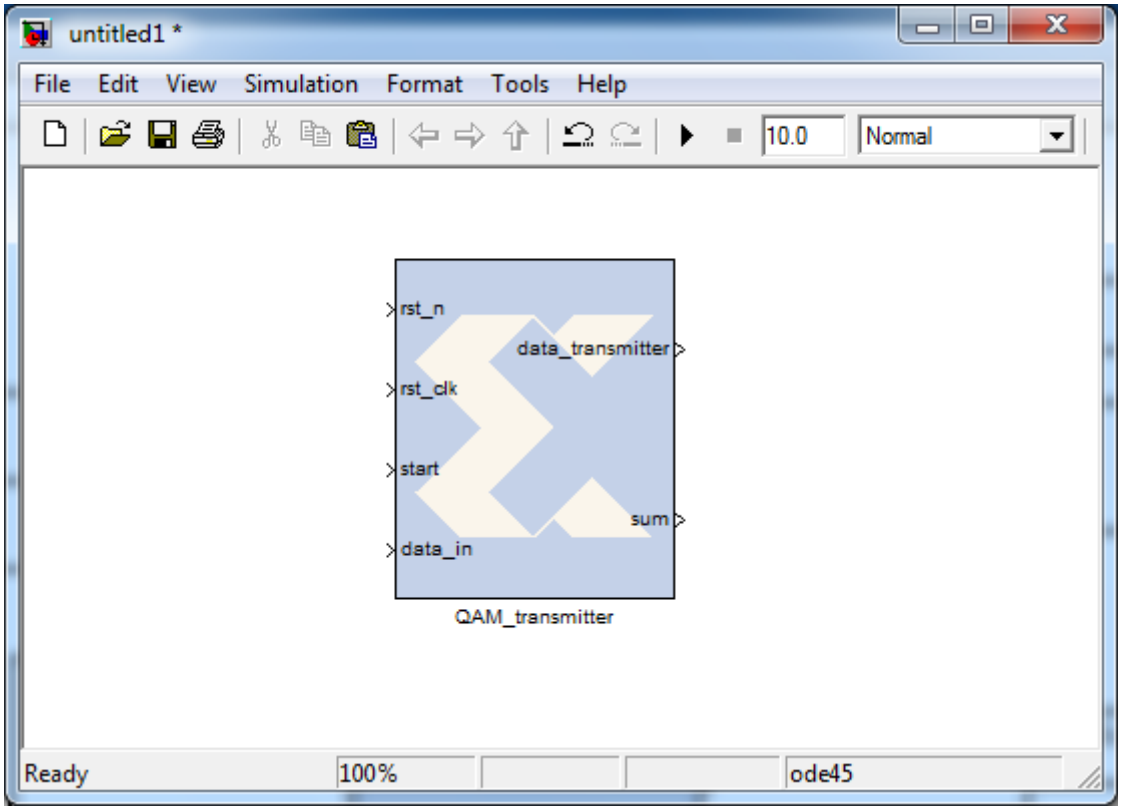


Tiến hành mô phỏng:

Phầm mô phỏng được chạy trên kit SpartanR-6 FPGA của XILINX. Kết quả đối chiếu với khối Xilinx trong Matlab.

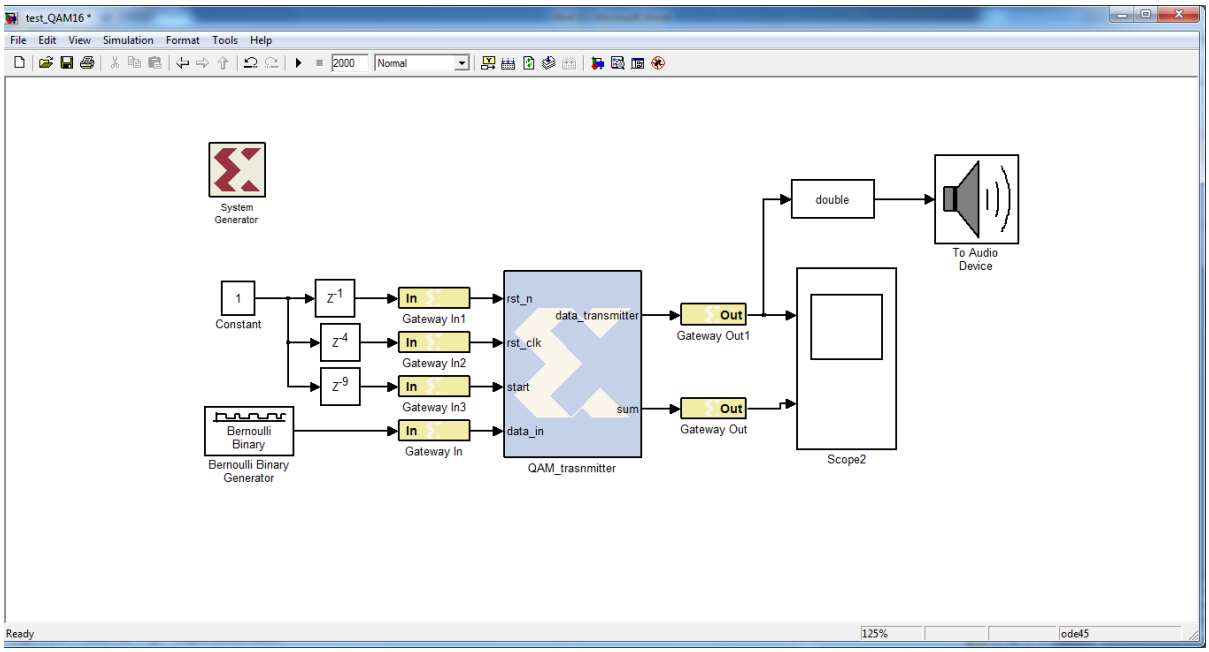
Trước tiên, chúng ta import tất cả code phần cứng vào khối Blackbox trong ***system generation*** của Simulink theo đường dẫn **Simulink Library Browser → Xilinx Blockset → BasicElements → Blackbox.**

Import file top.v vào Blackbox, xuất hiện file *_config.m, tiến hành hiệu chỉnh clock, định dạng dữ liệu đầu vào đầu ra cho khối Blackbox trên file *_config.m. ta thu khối Blackbox hoàn chỉnh.



Hình 8.1 khối Blackbox đã nạp code verilog.

Tiến hành ghép nối với đầu vào là các khối khởi tạo dữ liệu có sẵn trong Simulink.



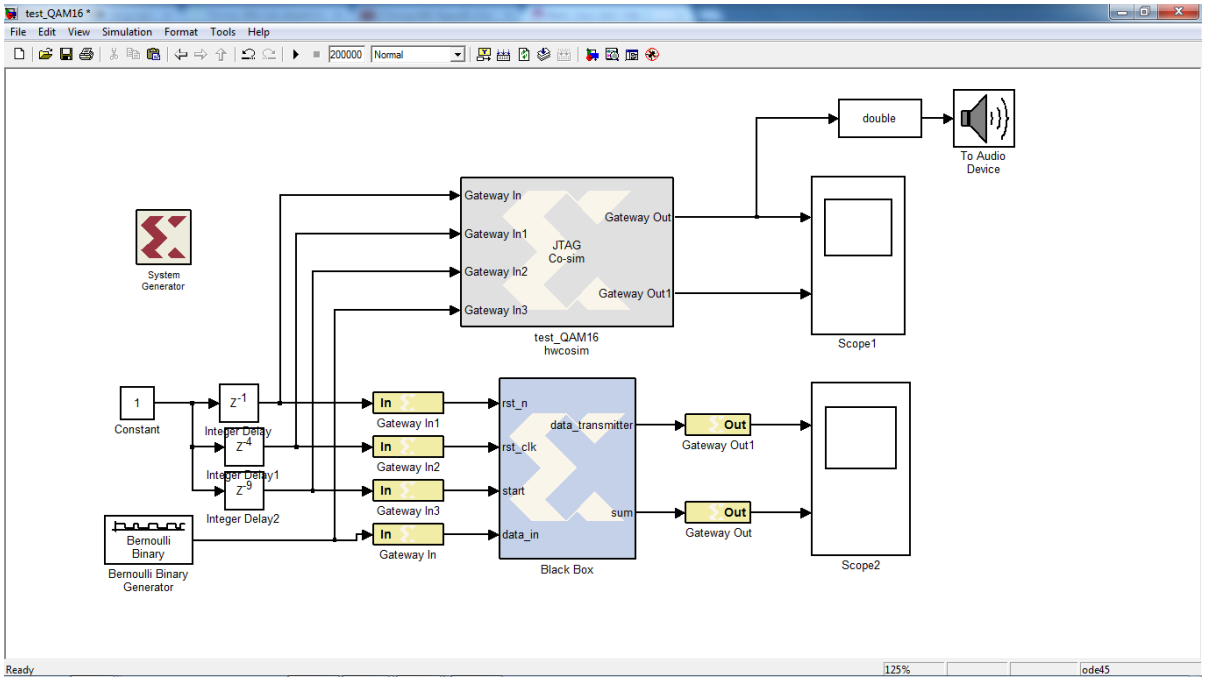
Hình 8.2 Khối phát trên Simulink

Generator khối Blackbox ở trên để chạy trên kit SpartanR-6 FPGA.

Lựa chọn thông số của system gennerator như sau:

- Compilation: **Hardware Co-Simulink → Atlys**
- Hardware description language: **Verilog**
- Chọn Generate

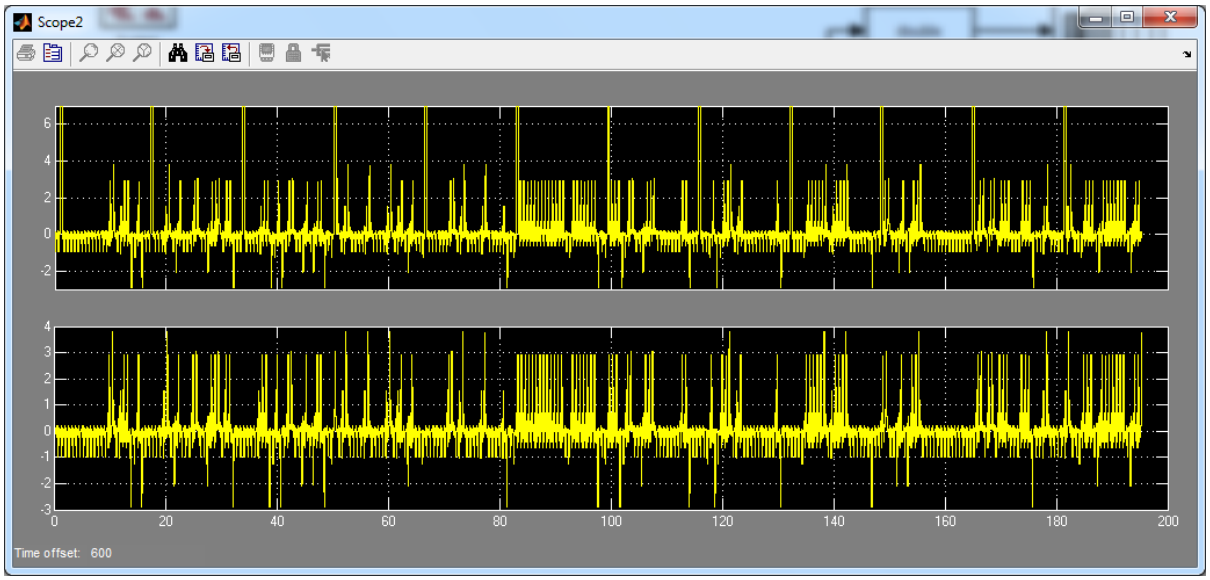
Kết quả thu được khối **JTAG Co-sim** (khối màu Xám). Kết nối khối JTAG Co-sim với đầu vào ra của simulink.



Khối phát báo gồm

- ✓ Binary chuỗi bit phát đi.
- ✓ Constan điều khiển các tín hiệu restart.
- ✓ QAM_transmitter chèn các phần tử 0, lọc dạng tín hiệu và nhân sóng mang.
- ✓ JTAG Co-sim kết quả generator khối QAM_transmitter
- ✓ Loa phát tín hiệu được ghép sóng mang ra ngoài môi trường.

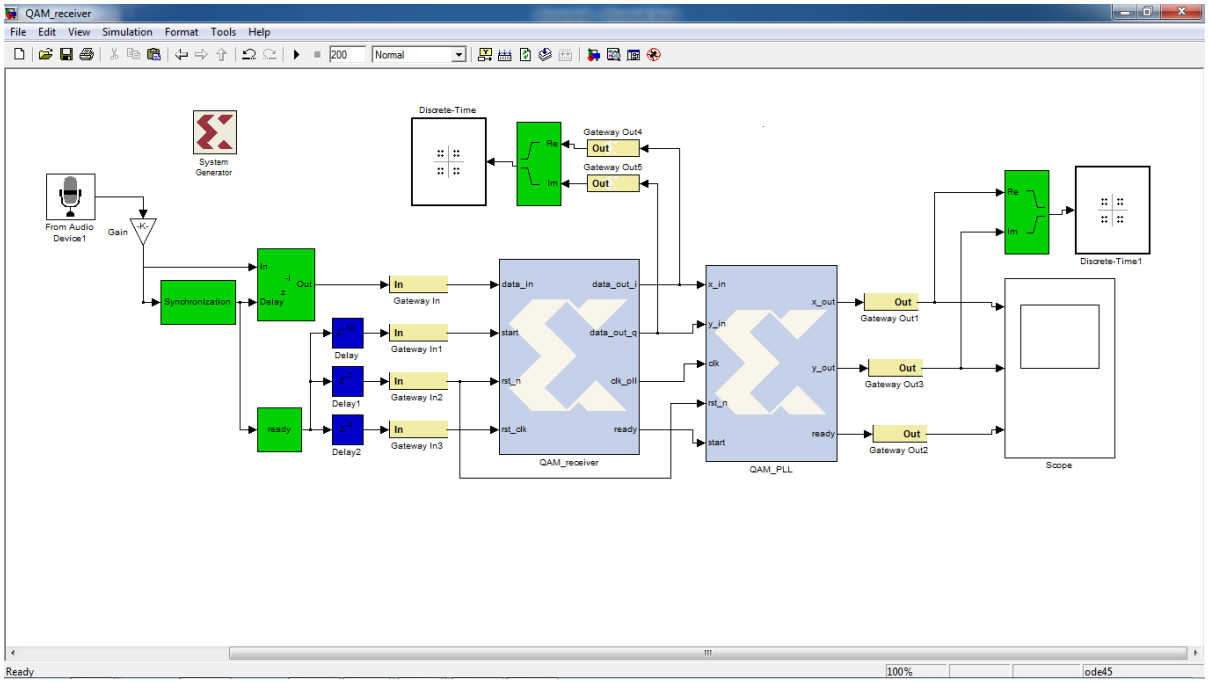
Dạng sóng hiển thị trên Scope khi thực hiện phát 16-QAM. Sum là tín hiệu thu được sau bộ ghép sóng mang. Data_transmitter là tín hiệu ra đã được chèn pilot vào sum.



Hình 8.3 Dạng sóng thu được trước khi phát qua Loa.

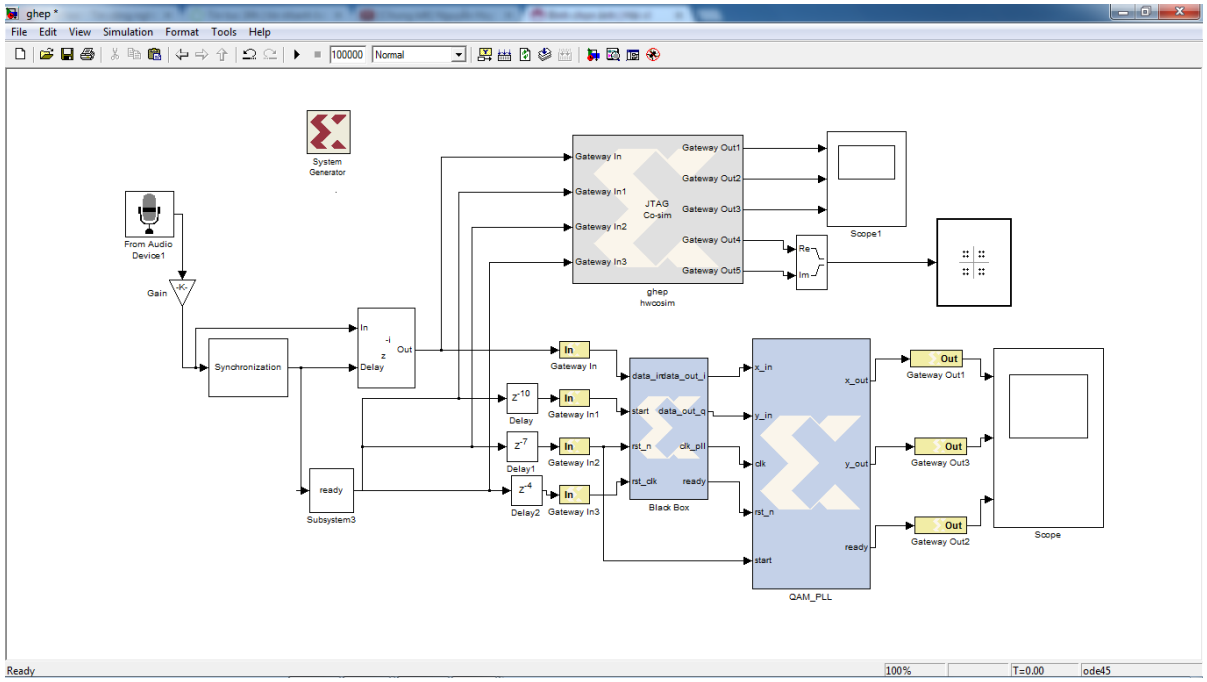
Bên khối Thu bao gồm:

- ✓ Mic thực hiện thu tín hiệu.
- ✓ Khối Synchronization thực hiện đồng bộ tín hiệu thu đưa ra điểm đầu khung và tín hiệu ready diều khiểm khối QAM_receiver và QAM_PLL.
- ✓ QAM_receiver nhân sóng mang với phase khởi tạo ban đầu và lọc LPF
- ✓ QAM_PLL phục hồi phase đúng của tín hiệu, ánh xa lên Chòm Sao.



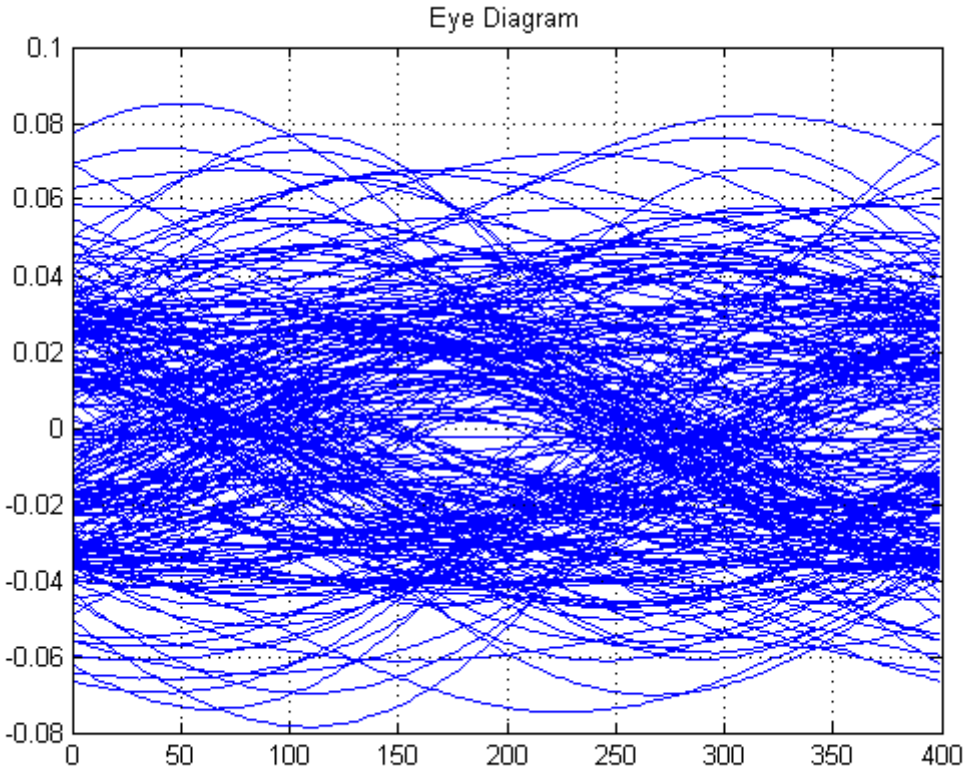
Hình 8.4 Khối Thu trên Simulink

Kết quả sau khi generator khối Thu.



Khi này, nhóm em sử dụng 2 kit SpartanR-6 để chạy một bên phát, một bên thu như sau:

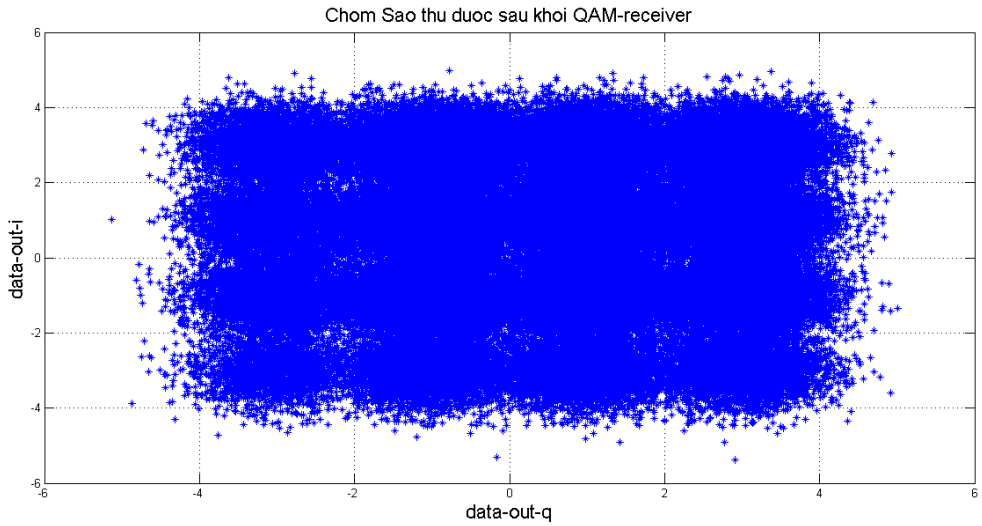
Sau khi cắt Pilot tín hiệu được xếp chồng lên nhau tao thành đồ thị Mắt.



Hình 8.5 Đồ thị Mắt thu được trước khi lấy mẫu

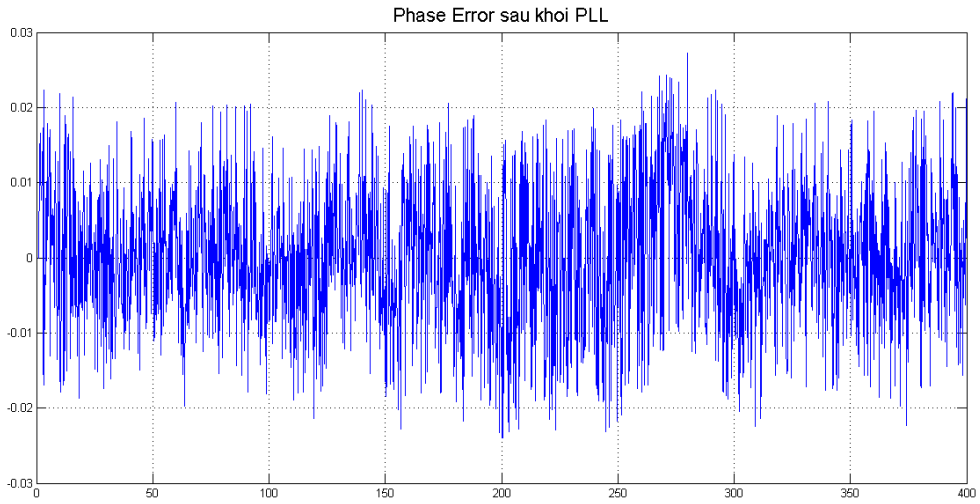
Tín hiệu thu được sau khối QAM_receiver.

Sai số lớn nhưng vẫn có thể nhận thấy chòm sao hình thành 16 điểm.

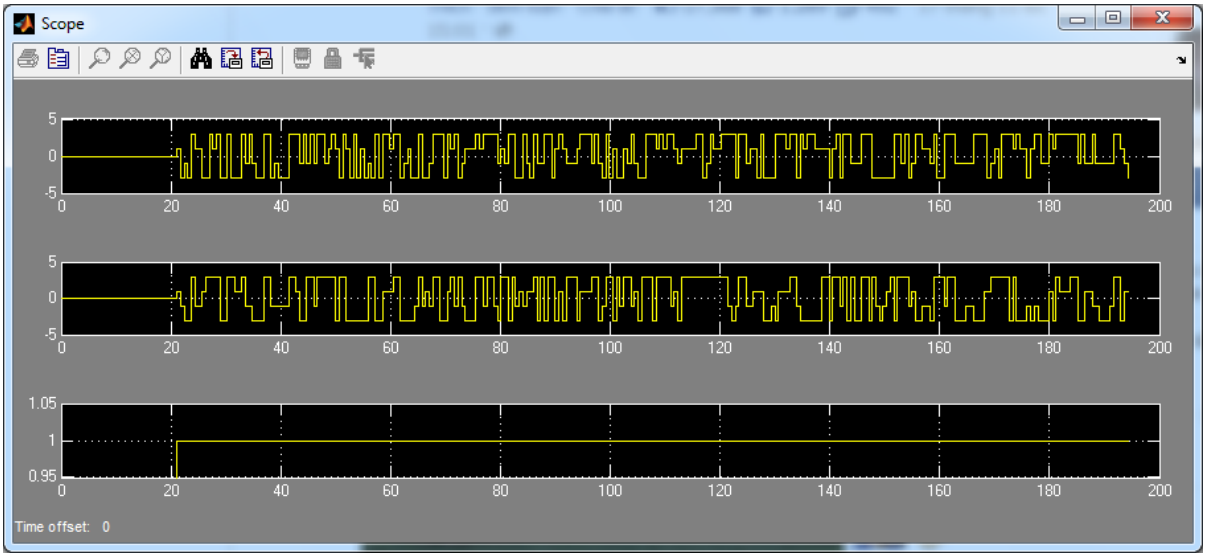


Hình 8.6 Chòm Sao sau khối QAM_receiver

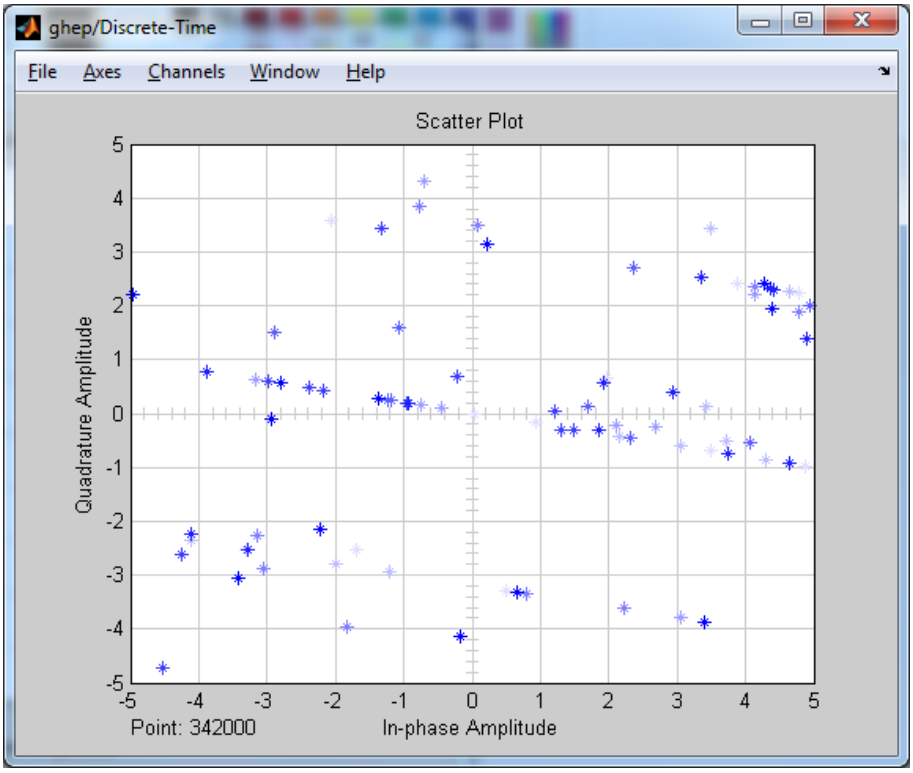
Phase Error trong quá trình khôi phục phase đúng của tín hiệu.



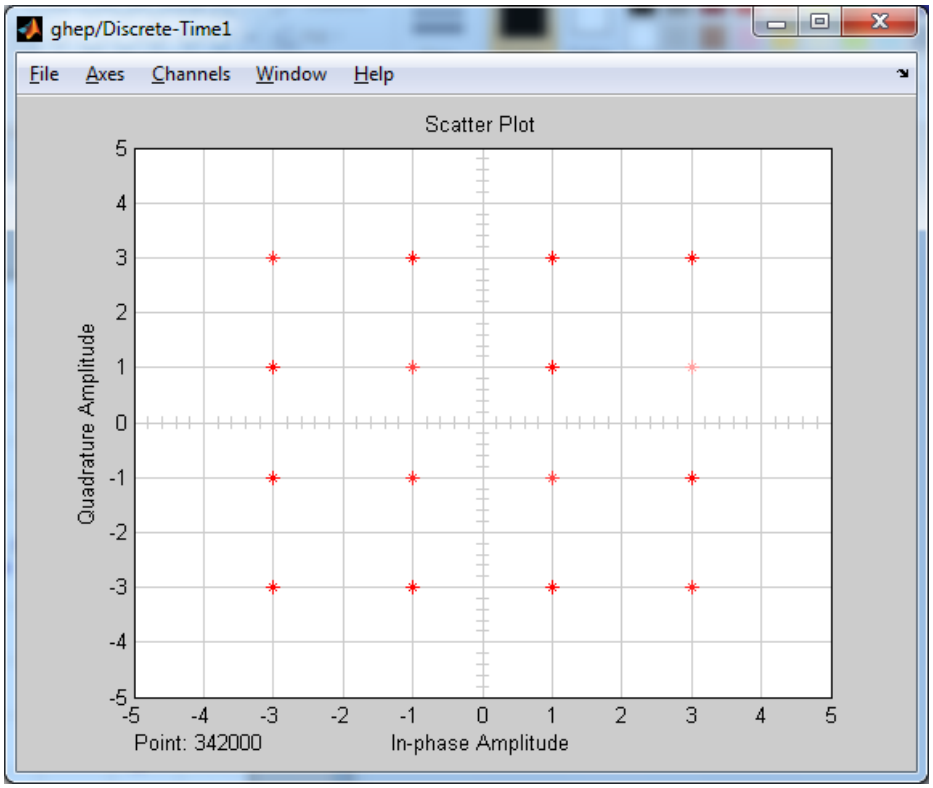
Hình 8.7 Phase error khi thực hiện dịch pha PLL



Hình 8.8 Kết quả trên kênh I, Q và tín hiệu ready.



Hình 8.9 Chòm Sao lỗi trước khi qua bộ PLL



Hình 8.10 Chòm Sao đúng sau khi qua bộ PLL

Tài liệu tham khảo

[1] Consultative Committee for Space Data Systems, "CCSDS 101.0-B-5 Recommendation for Space Data System Standards – Telemetry Channel Coding," June 2001.

[2] Xilinx User Guide. Retrieved on July 23 2005 from http://www.xilinx.com /products/software/sysgen/app_docs/user_guide.htm.

[3] Y. H. Hu, "CORDIC-Based VLSI Architectures for Digital Sig-nal Processing," IEEE Signal Processing Magazine , July 1992,pp. 16–35

[4] J.E. Volder, "The CORDIC Trigonometric Computing Tech-nique," IRE Trans. on Electronic Computers ,v ol. 8, no. 3, 1959,pp. 330–334

[5] T. Rappaport, Wireless Communications – Principles & Practice , 2nd edition. Prentice-Hall, Upper Saddle River, NJ, 1996.

[6] The Mathworks Inc., Simulink, Dynamic System Simulation for Matlab, Using Simulink. Natick, Massachusetts, USA.1999.

[7] Carrier Synchronization for 3- and 4-bit-per-Symbol Optical Transmission Ezra Ip and Joseph M. Kahn, Fellow, IEEE.

[8] J. P. Gordon and L. F. Mollenauer, "Phase noise in photonic commu-nications systems using linear amplifiers," Opt. Lett. , vol. 15, no. 23,pp. 1351–1353, Dec. 1990

[9] J. R. Barry and J. M. Kahn, "Carrier synchronization for homodyneand heterodyne detection of optical quadriphase-shift keying,"J. Lightw.Technol., vol. 10, no. 12, pp. 1939–1951, Dec. 1992

[10] G. P. Agrawal, Fiber Optic Communication System, 3rd ed. New York: Wiley, 2002

[11] Implementation of the CORDIC Algorithm in a Digital Down-Converter.

[12] http://www.xilinx.com/support/sw_manuals/sysgen_user.pdf