DESAIN DAN IMPLEMENTASI *CONTROL UNIT* DAN FAST FOURIER TRANSFORMATION MENGGUNAKAN *LIBRARY* FFTW3 UNTUK SISTEM AKUISISI DATA RADAR FMCW PADA KAPAL FERI

TUGAS AKHIR

AHMAD THARIQ SUHERMAN

13112003

PROGRAM STUDI TEKNIK ELEKTRO

JURUSAN SAINS

INSTITUT TEKNOLOGI SUMATERA

LAMPUNG SELATAN

2017

DESAIN DAN IMPLEMENTASI
***CONTROL UNIT* DAN FAST FOURIER TRANSFORMATION**
MENGGUNAKAN *LIBRARY* FFTW3 UNTUK SISTEM AKUISISI
DATA RADAR FMCW PADA KAPAL FERI

Oleh:
AHMAD THARIQ SUHERMAN

Tugas Akhir ini telah diterima dan disahkan
sebagai persyaratan untuk memperoleh gelar
SARJANA TEKNIK
di
PROGRAM STUDI TEKNIK ELEKTRO
JURUSAN SAINS
INSTITUT TEKNOLOGI SUMATERA

Lampung Selatan, September 2017
Disetujui oleh :

Pembimbing I,	Pembimbing II,
Arif Sasongko, S.T., M.T., Ph.D.	**Dean Corio, S.T., M.T.**
NIP. 132 320 058	**NIP. 198 606 222 015 041 003**

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan benar.

Nama **: .....**

NIM **: .....**

Tanda Tangan **: .....**

Tanggal **: .....**

DESAIN DAN IMPLEMENTASI *CONTROL UNIT* DAN FAST FOURIER TRANSFORMATION MENGGUNAKAN *LIBRARY* FFTW3 UNTUK SISTEM AKUISISI DATA RADAR FMCW PADA KAPAL FERI

ABSTRAK

Oleh

AHMAD THARIQ SUHERMAN

NIM : 13112003

PROGRAM STUDI TEKNIK ELEKTRO

Seiring dengan berkembangnya teknologi radar, maka berkembang pula teknologi akuisisi data yang mampu bekerja secara optimal dengan input dari karakteristik sinyal radar tersebut. Dalam hal ini sistem radar FMCW memancarkan gelombang mikro secara kontinyu yang stabil dimodulasikan secara frekuensi dengan suatu gelombang frekuensi rendah dengan periode tertentu. Prinsipnya, perbedaan antara frekuensi gelombang yang ditransmisikan dan diterima dapat digunakan untuk mendeteksi jarak dan kecepatan target.

Pada sistem akuisisi data tugas akhir ini terdapat bagian pengolahan berupa *control unit* yang yang terdiri dari *frequency divider* dan *trigger*, ekspansi bit data, serta FFT yang umum ditemui untuk mempermudah analisis jarak dari sinyal menggunakan *library* C FFTW. Pengolahan sinyal secara digital tersebut diimplementasikan pada SoCKit Evaluation Board yang mengintegrasikan FPGA Altera Cyclone V yang berdaya rendah dalam beroperasi dengan ARM Cortex-A9.

Pada tugas akhir ini, untuk *frequency divider* akan membagi sumber clock dari osilator 50 Mhz menjadi 2,083 MHz. *Trigger* yang mengendalikan kecepatan penulisan data pada FIFO yang dengan memanfaatkan counter dan memperhitungkan *guard time* sinyal radar. Ekspansi bit yang akan menambahkan bit LSB untuk tiap 1 data dengan lebar data 14-bit menjadi 16 bit. Serta FFT yang bisa diimplementasikan pada HPS SoCKit Evaluation Board dan digunakan untuk mengoperasikan deretan sinyal input.

Kata Kunci: FMCW Radar, FFT, SoCKit, trigger, bexpansion, HPS, frequency divider.

DESIGN AND IMPLEMENTATION OF CONTROL UNIT AND FAST FOURIER TRANSFORMATION USING FFTW3 LIBRARY FOR DATA ACQUISITION SYSTEM ON FERRY

ABSTRACT

By

AHMAD THARIQ SUHERMAN

NIM :13112003

ELECTRICAL ENGINEERING STUDY PROGRAM

As the technology of radar developed, thus the technology of data acquisition also developed to optimize the acquisition of radar signal. In this case, FMCW radar system transmitting microwave continuously which stable and can be modulated by frequency by a certain period of low-frequency wave. Theoretically, the difference between transmitted and received frequency wave can be used for detecting distance and velocity of the target.

In this final project: Data Acquisition System, it contains by two parts of a control unit which consisted of frequency divider and trigger, Bit expansion of data, and Fast Fourier Transformation which can be found in any data acquisition system and simplify the analysis of the signal. This project using FFTW as its FFT library. Those digital processing units are implemented on SoCKit Evaluation Board which integrates FPGA Altera Cyclone V with ARM Cortex-A9.

In this work, frequency divider will divide the clock source from 50 MHz Oscillator to 2,083 MHz, this frequency will be used as the sampling frequency of ADC. Trigger controlling the rate of data writing on FIFO by calculating the guard time of radar signal and taking the advantage of counter. The bit of data can be adding 2 bits of LSB for a 14-bit width of data for each signal input. Also, FFT is able to be implemented on HPS of SoCKit Evaluation Board and used for the operation of sequence data from FPGA.

Keyword: FMCW Radar, Fast Fourier Transformation, SoCKit, Trigger, bit expansion, HPS, frequency divider.

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir serta menyusun tugas akhir ini .

Selama melaksanakan tugas akhir ini, penulis telah banyak memperoleh bantuan dan dukungan dari berbagai pihak. Oleh karena itu penulis menyampaikan rasa terima kasih kepada:

1. Orangtua, Kakak dan Adik penulis yang selalu memberikan do'a, dukungan, dan kasih sayang kepada penulis;
2. Bapak Arif Sasongko selaku dosen pembimbing I dan Bapak Dean Corio selaku dosen pembimbing II , yang telah memberikan petunjuk dan bimbingan yang berarti dalam proses menyelesaikan tugas akhir ini
3. Arvin Ditto Amalsyah dan Ibrahim Amyas Aksar Tarigan sebagai sahabat dan rekan satu team yang selalu menyemangati dan memberikan nasihat yang berharga selama mengerjakan tugas akhir.
4. Bapak Ratna Indrawijaya dan Yaya Sulaeman sebagai pembimbing kuliah praktek di LIPI Bandung yang banyak memberikan kesempatan untuk berbagi ilmu yang sangat bermanfaat dalam pengerjaan tugas akhir ini.
5. Pihak-pihak lain yang juga telah banyak memberikan bantuan kepada penulis yang tidak dapat penulis sebutkan satu per satu;

Penulis menyadari bahwa tugas akhir ini bukanlah yang paling akhir dan masih perlu dikembangkan lebih lanjut, untuk itu kritik yang membangun dan saran sangatlah penulis harapkan.

Lampung Selatan, September 2017

Ahmad Thariq Suherman

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR SINGKATAN

Singkatan	Kepanjangan	Pemakaian pertama kali pada halaman
FMCW	Frequency Modulated Continous Wave	1
DAQ	Data Acquisition	1
PC	Personal Computer	1
FPGA	Field Programmbale Gate Array	1
SoCKit	System on Chip Kit	2
FIFO	First In First Out	2
HPS	Hard Processor System	2
FFT	Fast Fourier Transformation	2
VHDL	Very High Defnition Language	5
FFTW	The Fastest Fourier Transformation in the West	6
ARM	Acorn RISC Machine	6
OS	Operating System	7
LSB	Least Significant Bit	19
ADC	Analog-to-Digital Converter	15
DSP	Digital Signal Processing	20

DAFTAR TABEL

BAB I
PENDAHULUAN

2.4 Latar Belakang

Radar merupakan singkatan dari *Radio Detection and Ranging* berfungsi untuk menentukan jarak, posisi, dan kecepatan dari suatu benda. Di awal pengembangannya, jenis radar yang sangat populer digunakan adalah *pulse-radar*. Sayangnya radar jenis ini memiliki kelemahan yaitu menggunakan daya yang cukup besar untuk dapat digunakan dan juga memiliki masalah untuk pengukuran jarak pendek. Atas dasar ini, maka dilakukan pengembangan untuk menghasilkan radar yang dapat melakukan pengukuran pada jarak yang pendek, mudah digunakan, dan hemat daya penggunaannya. Diciptakanlah FMCW radar, yang memenuhi semua kebutuhan itu. Sistem radar FMCW memancarkan gelombang mikro secara kontinyu yang stabil dimodulasikan secara frekuensi dengan suatu gelombang frekuensi rendah dengan periode tertentu. Prinsipnya, perbedaan antara frekuensi gelombang yang ditransmisikan dan diterima dapat digunakan untuk mendeteksi jarak dan kecepatan target.

Karena keluaran dari hasil penggabungan sinyal yang dipancarkan dan diterima masih berupa sinyal analog, informasi dan data yang diberikan oleh radar tidak begitu praktis. Artinya, informasi sinyal analog ini tidak dapat diolah lebih lanjut, yaitu dimanipulasi dan diteruskan kembali. Sinyal ini harus dikonversi dulu menjadi sinyal digital, kemudian diproses di PC agar dapat dimengerti oleh manusia. Proses konversi inilah akan dilakukan oleh sistem DAQ.

Oleh karena itu, dibutuhkan proses digital yang sering digunakan dalam mengakuisisi sinyal analog, yaitu mengendalikan bit-bit dan laju peyimpanan dalam rangkaian digital, atau bisa disebut sebagai *control unit*. Selain itu, operasi yang tak jarang dilakukan dalam menerjemahkan informasi radar adalah Fast Fourier

Transformation guna memudahkan analisis dengan menggunakan SoCKit Evaluation Board.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang disampaikan sebelumnya, laporan tugas akhir ini akan membahas beberapa bagian sistem akuisisi data, yaitu *control unit* berupa trigger dan pembagi frekuensi, serta Fast Fourier Transformation. Rumusan masalah pada tugas akhir ini sebagai berikut:

1. Bagaimana cara mengendalikan aliran data yang akan diteruskan ke FIFO?
2. Bagaimana cara mengekspansi bit data?
3. Bagaimana cara membagi frekuensi dari sumber clock yang digunakan?
4. Bagaimana cara menggunakan FFT pada HPS *board* SoCKit?

1.3 Tujuan

Tujuan yang ingin dicapai dari tugas akhir ini adalah:

1. Melakukan perancangan pengendalian aliran data ke FIFO dengan menggunakan counter bersyarat sesuai dengan karakteristik sinyal radar.
2. Melakukan penambahan bit pada data yang masuk sehingga data yang masuk dapat masih dapat digunakan pada modul yang lain
3. Melakukan perancangan pembagi frekuensi yang akan digunakan sebagai frekuensi sampling dan clock bagi modul yang lain.
4. Melakukan perancangan dan implementasi FFT pada HPS sehingga data sinyal yang masuk dapat memiliki informasi yang dapat dipahami.

1.4 Batasan Masalah

Batasan masalah pada tugas akhir ini adalah sistem akuisisi data yang dirancang untuk radar FMCW yang data karakteristik sinyalnya diperoleh dari Lembaga Ilmu Pengetahuan Indonesia. Kemudian modul *control unit* dan ekspansi bit dibuat untuk mengendalikan laju data dan menambahkan data tiap bit dengan mempertimbangkan besar frekuensi yang disampling dan karakteristik sinyal dari radar FMCW, serta FFT dengan menyesuaikan mikroprosesor yang digunakan sehingga informasi yang diolah tidak berubah. Pembatasan masalah akan

1.5 Metodologi Penelitian

Metodologi yang dipakai untuk melakukan perancangan dan implementasi tugas akhir ini adalah sebagai berikut:

1. Tinjauan Pustaka

   Pada langkah ini akan dibahas teori-teori dan hal-hal teknis lainnya mengenai spesifikasi alat-alat yang digunakan dalam merancang modul.
2. Perancangan

   Pada langkah kedua ini akan dilakukan perancangan yang mempertimbangkan karakteristik dari sinyal yang akan diakuisisi
3. Implementasi

   Pada langkah ketiga ini akan dibahas mengenai proses implementasi dari hasil rancangan.
4. Pengujian

   Langkah keempat ini akan dibahas mengenai pengujian yang dilakukan pada modul-modul yang telah diimplementasikan dan apakah hasil dari perancangan yang dilakukan sudah memenuhi spesifikasi atau belum.
5. Simpulan

   Langkah terakhir akan membahas hasil dari pengujian yang telah dilakukan serta keseluruhan langkah dari tahap tinjauan pustaka hingga pengujian.

1.6 Sistematika Penulisan

Laporan tugas akhir ini secara umum terbagi menjadi tiga bagian: pendahuluan, isi, dan penutup. Setiap bagian dapat terbagi menjadi satu atau lebih bab. dengan penyajian sebagai berikut:

- BAB I PENDAHULUAN
  Bab ini membahas latar belakang tugas akhir, rumusan masalah, tujuan, batasan masalah, metodologi penelitian dan sistematika penulisan.
- BAB II TINJAUAN PUSTAKA
  Bab ini berisi dasar-dasar teori dan spesifikasi alat pada perancangan dan algoritma yang digunakan.
- BAB III PERANCANGAN DAN IMPLEMENTASI
  Bab ini berisi perancangan *control unit*, ekspansi bit dan FFT yang digunakan.
- BAB IV PENGUJIAN DAN VERIFIKASI SISTEM
  Bab ini menjelaskan mengenai pengujian dari sistem yang telah diimplementasikan dan verifikasi sistem yang telah dirancang.
- BAB V PENUTUP
  Bab ini berisi simpulan dari hasil yang didapat pada tugas akhir ini, beserta saran untuk pengembangan sistem selanjutnya.

BAB II

TINJAUAN PUSTAKA

1.1. FPGA

Salah satu bagian dari SoCKit, yaitu *Field Programmable Gate Array* (FPGA) yang merupakan sebuah sirkuit digital terintegrasi yang tak jarang digunakan untuk mengimplementasikan rancangan rangkaian digital karena memiliki kemampuan pemrosesan data yang sangat cepat, serta rendah dalam konsumsi daya. Interkoneksi gerbang logika yang dimiliki FPGA mampu dikonfigurasikan satu sama lainnya. FPGA digunakan untuk melakukan proses komputasi dari alogritma integrasi numerik yang memiliki performa yang sangat baik serta kemampuan yang tinggi dalam melakukan integrasi gerbang-gerbang logika.

Gambar 1. Altera SoCKit Evaluation Board

Sumber : www.terasic.com

Pada tugas akhir ini digunakan SoCKit yang menggunakan Altera Cyclone V yang nantinya board ini akan diprogram menggunakan bahasa VHDL dan atau Verilog.

1.2. Fast Fourier Transformation

Fast Fourier Transformation adalah suatu algoritma untuk menghitung transformasi Fourier diskrit (DFT) dengan cepat dan efisien daripada harus menghitung satu persatu point dari serangkaian data yang masuk. Transformasi Fourier Diskrit ditunjukkan pada persamaan (1)

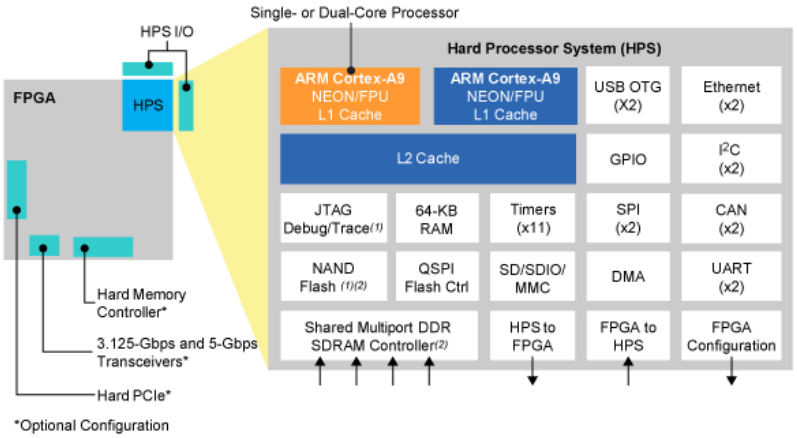
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \dots, N-1 \qquad \text{.....(1)}$$

Ketika menghitung deret Fourier secara langsung , kita memerlukan operasi aritmatika sebanyak O (N^2). Sedangkan algoritma FFT hanya memerlukan operasi sebanyak O(N log N) untuk menghitung deret yang sama..

Pada tugas akhir ini dipergunakan library FFT yang dikeembangkan oleh FFTW 3.3.6 , yaitu sebuah library subrutin dalam bahasa C untuk menghitung transformasi Fourier diskrit dalam dimensi satu atau lebih, beraneka ukuran input, baik data real dan kompleks. FFTW dikembangkan dari tahun ke tahun untuk bermacam-macam bahasa dan arsitektur, serta optimalisasi algoritma serta alokasi memori yang akan dipergunakan untuk melakukan operasi FFT secara digital.

1.3. Hard Processor System

Sejak tahun 2012 Altera memproduksi SoC FPGA, Pada SoCKit Evaluation Board yang digunakan terdapat bagian yang disebut dengan *Hard Processor System* yang diintegrasikan secara penuh dengan FPGA melalui satu perangkat. Basis HPS yang digunakan, yakni ARM-Cortex A9 yang merupakan mikroprocessor 32-bit dengan arsitektur ARMv7-A yang menyediakan 4 cache-coherent core, beserta NEON SIMD.



Gambar 2. Diagram Block HPS Pada SoCKit

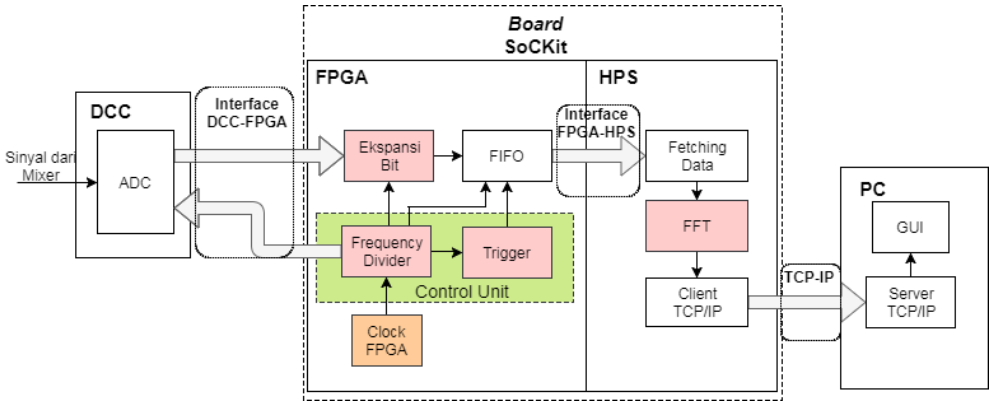
Sumber : www.terasic.com

Untuk mengekseksi program yang akan digunakan pada HPS, maka diperlukan suatu program minicom yang akan mengemulasi OS pada HPS menggunakan komunikasi serial.

BAB III

PERANCANGAN DAN IMPLEMENTASI DESAIN

3.1. Spesifikasi



Gambar 3. Diagram Blok Sistem Akuisisi Data

Pada tugas akhir ini yang akan menjadi pembahasan adalah modul ekspansi bit, frequency divider, trigger dan FFT yang terdapat pada sistem akuisi data (lihat **Gambar 3)**. Oleh karena itu, ditentukan spesifikasi untuk *control unit* yang terdiri dari frequency divider dan trigger pada FPGA seperti **Tabel 1** di bawah ini

Tabel 1. Spesifikasi *control unit*

Input	*Frequency divider :* *Clock* frekuensi 50 MHz *Trigger* : *Clock* frekuensi 1.041 MHz
Output	*Frequency divider :* *Clock* frekuensi 2.083 MHz dan 1.041 MHz *Trigger* : *Falling edge trigger pulse.*

Fungsi	*Frequency divider* : - Sebagai pembagi clock sumber menjadi frekuensi sampling - Input untuk trigger *Trigger :* - Mengatur laju penulisan data pada FIFO

Spesifikasi untuk ekspansi bit pada FPGA seperti **Tabel 2** di bawah ini

Tabel 2. Spesifikasi ekspansi bit

Input	1 data dengan lebar 14 bit
Output	l data dengan lebar 16 bit
Fungsi	Untuk mempermudah akuisisi data pada modul yang lain, tanpa mengubah informasi yang ada.

Spesifikasi untuk Fast Fourier Transformation pada HPS seperti **Tabel 3** di bawah ini

Tabel 3. Spesifikasi Fast Fourier Transformation

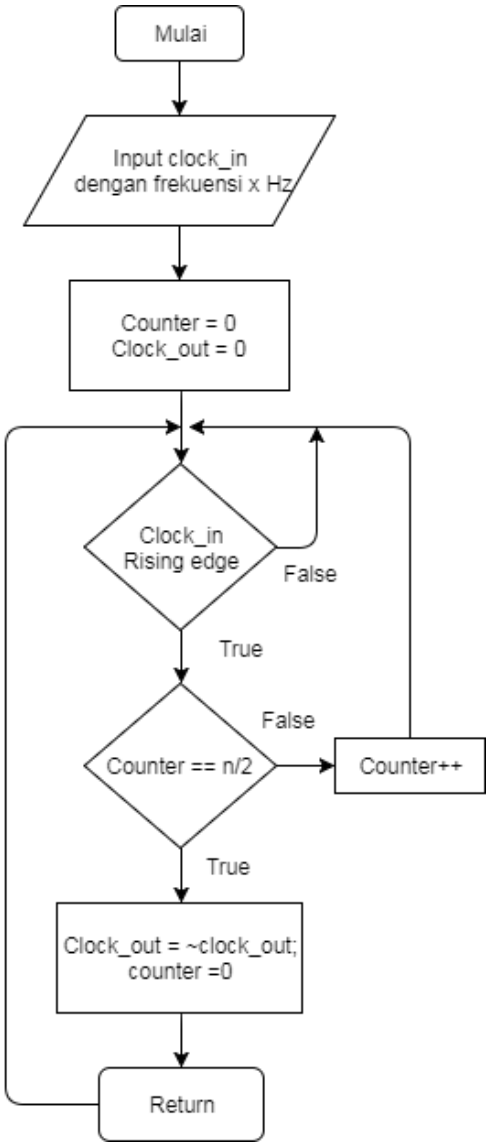
Input	n data dengan lebar data 32 bit (float), domain waktu
Output	n data dengan lebar data 32 bit (float), domain frekuensi
Fungsi	Memberikan informasi berupa frekuensi yang terukur dari sekuens data yang keluar dari modul FIFO.

3.2. Perancangan dan Implementasi

3.2.1. Control Unit

3.2.1.1. Frequency Divider

Frequency divider berfungsi untuk menurunkan frekuensi internal FPGA sehingga outputnya dapat digunakan oleh modul-modul lain sebagai *clock*, misalnya modul FIFO.



Gambar 4. Diagram Alir *Frequency Divider*

Input : *Clock* dengan frekuensi X.

Output : *Clock* dengan frekuensi X/n. Nilai n adalah bilangan genap.

Proses :

1. Menerima input Clock_In dengan frekuensi X.
2. Inisialisasi variabel Counter dan Clock_Out.
3. Untuk setiap *rising edge* dari Clock_In (perubahan dari *active low* menjadi *active high*), kerjakan proses berikut :
    a. Memeriksa nilai variabel Counter.
    b. Apabila nilai Counter = n/2, maka Clock_Out dinegasi (*active low* menjadi *active high*, atau sebaliknya).
    c. Apabila nilai Counter != n/2, maka Counter akan bertambah 1.
    d.

Dengan algoritma ini, modul *frequency divider* mampu membagi frekuensi input sebesar n, dengan n merupakan bilangan genap. Modul tidak bisa membagi dengan nilai bilangan ganjil.

Board SoCKit memiliki osilator dengan frekuensi 50 MHz. Frekuensi ini perlu diturunkan agar dapat digunakan oleh yang lainnya. Untuk itu, diperlukan *frequency divider* yang dapat membagi frekuensi menjadi lebih kecil. Pertama-tama, *clock* dengan frekuensi 50 MHz diteruskan sebagai masukan ke *frequency divider* yang mampu membagi frekuensi sebesar 6, dengan *source code* sebagai berikut:

```
// Frequency Divider by 6.



module freqdivider6(clk,rst,clk_out);

// Input adalah clock 50 MHz
```

```
input clk,rst;

// Output adalah 50/6 = 8.333 MHz

output reg clk_out;

reg [5:0] counter;



always @(posedge clk or negedge rst)

begin

      if(!rst)

            begin

            counter <= 5'd0;

            clk_out <= 1'b0;

            end

      else

     if(counter == 5'd2)       //divide by 6

                  begin

                        counter <= 5'd0;

                        clk_out <= ~clk_out;

                  end

            else

                  begin
```

```
                        counter <= counter+5'd1;

                    end

end



endmodule
```

Output adalah *clock* dengan frekuensi sebesar 50/6 = 8.333 MHz, yang kemudian diteruskan ke *frequency divider* yang mampu membagi frekuensi sebesar 2, 4, atau 8, dengan *source code* sebagai berikut:

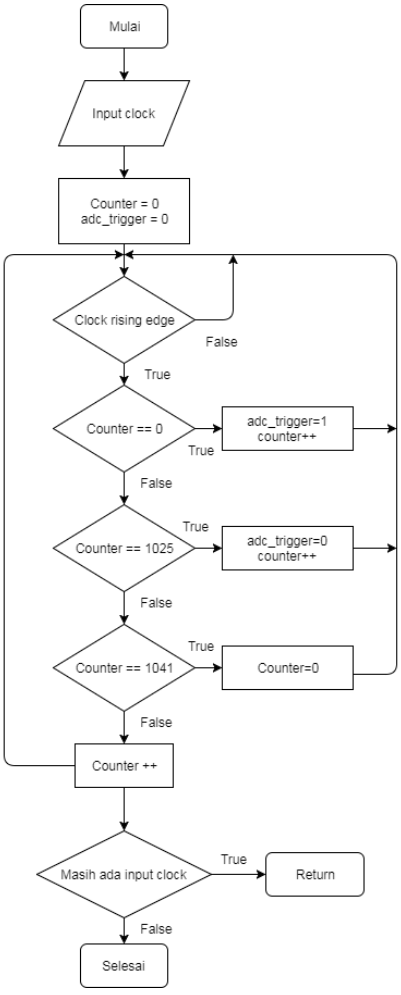
```
// Frequency Divider by 2, 4, & 8



module freqdivider8(clk,rst,clk_out,sync_out);

//Input adalah clock dengan frekuensi 8.333 MHz

input clk, rst;

output clk_out, sync_out;

reg [4:0] counter;



always @(posedge clk or negedge rst)

begin

      if(!rst)

            begin
```

```
            counter <= 4'd0;
            end
      else
     if(counter == 4'd7)
                  begin
                        counter <= 4'd0;
                  end
            else
                  begin
                        counter <= counter+4'd1;
                  end
end
//0->clk/2   //1->clk/4   // 2->clk/8
//clk_out adalah output yang mengambil bit ke-1 dari counter.
//besar frekuensinya adalah 8.333/4 = 2.08333 MHz
assign clk_out = counter[1];
//sync_out adalah output yang mengambil bit ke-2 dari counter.
//besar frekuensinya adalah 8.333/8 = 1.04166 MHz
assign sync_out = counter[2];
```

```
endmodule
```

Frequency divider ini memiliki 2 *output*. *Output* yang pertama adalah clk_out berupa *clock* dengan frekuensi sebesar 8.333/4 = 2.083 MHz. Sinyal clk_out akan digunakan sebagai frekuensi *sampling* ADC, *clock* FIFO, serta *clock* ekspansi bit. Output kedua adalah sync_out berupa *clock* dengan frekuensi sebesar 8.333/8 = 1.0416 MHz. Sinyal sync_out akan digunakan sebagai *clock trigger*

3.2.1.2. Trigger



Gambar 5. Diagram Alir *Trigger*

Trigger berfungsi untuk mengatur kerja dari FIFO. Ketika *trigger* memberikan output bernilai 1, maka FIFO akan menulis data ke dalam *buffer*. Input dari *Trigger* adalah *clock* dari *frequency divider* sebesar 1,041 MHz, dan output adalah *trigger pulse.*

Input : *Clock*.

Output : Variabel adc_trigger.

Proses :

1. Menerima input Clock
2. Inisialisasi variabel Counter dan adc_trigger
3. Untuk setiap *rising edge* dari Clock (perubahan dari *active low* menjadi *active high*), kerjakan proses berikut :
    a. Memeriksa nilai variabel Counter.
    b. Apabila nilai Counter = 0, nilai adc_trigger menjadi 1, dan Counter bertambah 1.
    c. Apabila nilai Counter = 1025, maka adc_trigger menjadi 0, dan Counter bertambah 1.
    d. Apabila nilai Counter = 1041, maka nilai Counter menjadi 0.

Apabila syarat b. sampai d. tidak terpenuhi, maka Counter bertambah 1.

Trigger berfungsi sebagai penentu waktu untuk menuliskan data yang masuk ke FIFO dengan bekerja seiring *clock*. *Trigger* merupakan upaya untuk melakukan efisiensi ruang memori berdasarkan karakteristik sinyal radar yang memiliki waktu jaga (*guard time*), oleh karena itu *trigger* akan mengehentikan penulisan data tepat saat waktu jaga (*guard time*) radar. Untuk memudahkan dalam implementasi trigger dibuatlah *counter* yang bersandingan dengan *clock frequency divider.*

Berikut adalah *source code* pada *Trigger:*

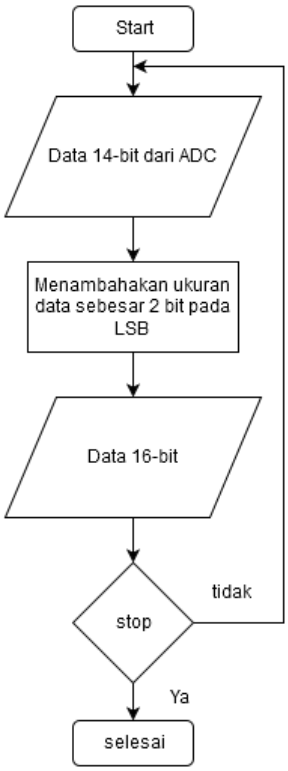
```
// Trigger


module
```

```
trigger(clk,rst,adc_trig,ramp_mode,dds_trig_1ramp,dds_trig_2ramp,dds_tr
ig);

input clk, rst, ramp_mode;

output dds_trig;

output reg adc_trig;

output reg dds_trig_1ramp;

output reg dds_trig_2ramp;

reg [11:0] counter;


always @(posedge clk or negedge rst)

Begin

      if(!rst)

            Begin

                  counter <= 11'd0;

                  adc_trig <= 1'b0;    // hanya adc trigger yang
digunakan

                  dds_trig_1ramp <= 1'b0;

                  dds_trig_2ramp <= 1'b0;

            end

      else

            if(counter==11'd0)   //start ramp

                  begin

                        adc_trig <= 1'b1;

                        dds_trig_1ramp <= 1'b1;

                        dds_trig_2ramp <= 1'b1;

                        counter <= counter+11'd1;

                  end
```

```
            else if(counter==11'd1025)    //stop ramp
                  begin
                        adc_trig <= 1'b0;
                        dds_trig_1ramp <= 1'b1;
                        dds_trig_2ramp <= 1'b0;
                        counter<=counter+11'd1;
                  end
            else if(counter==11'd1041)    //guard end (pri)
                  begin
                        counter<=11'd0;
                  end
            else
                  begin
                        dds_trig_1ramp <= 1'b0;
                        dds_trig_2ramp <= 1'b0;
                        counter<=counter+11'd1;
                  end

end

assign dds_trig = (ramp_mode) ? dds_trig_1ramp : dds_trig_2ramp;

endmodule
```

3.2.2 Ekspansi Bit

Penambahan ukuran data sebesar 2 bit pada LSB juga bertujuan untuk mempertahankan data dalam bentuk *two's complement*, mengingat komputer melakukan pengolahan data dengan *two's complement*.



Gambar 6. Diagram alir ekspansi bit

Input : Masukkan yag akan digunakan oleh DSP adalah data tiap-tiap hasil sampling yang berukuran 14-bit pada masing-masing channel

Output : Hasil keluaran dari ekspansi adalah data-data yang berukuran 16-bit dengan 2 bit nol tambahan pada LSB pada masing-masing channel, yang digabungkan menjadi 32 bit.

Proses :

1. Input yang masuk berukuran 14-bit disimpan dalam register

2. Data yang berukuran 14-bit ditambahkan ukurannya sebesar 2-bit pada LSB pada masing-masing channel
3. Data yang telah berukuran 16-bit pada masing-masing channel diurutkan menjadi channel channel A terlebih dahulu, lalu channel B sehingga lebar data menjadi 32 bit

Untuk memudahkan dalam pengiriman data digital yang nantinya diterima oleh HPS, maka diperlukan ekspansi bit dari 14 bit ke 16 bit, untuk masing-masing channel A dan B. Dikarenakan implementasi dikerjakan dengan tipe *two's complement*, maka penambahan yang tepat dilakukan pada LSB sehingga tidak mengubah informasi dari data-data yang telah diakuisisi.

Berikut adalah *source code* data asli disimpan dalam register `tempa2da_data` dan `tempa2db_data`:

```
//--- Temp Channel A

always @(negedge reset_n or posedge addaADA_DCO)

begin

      if (!reset_n) begin

            tempa2da_data     <= 14'd0;

      end

      else begin

            tempa2da_data     <= addaADA_D;

      end

end
```

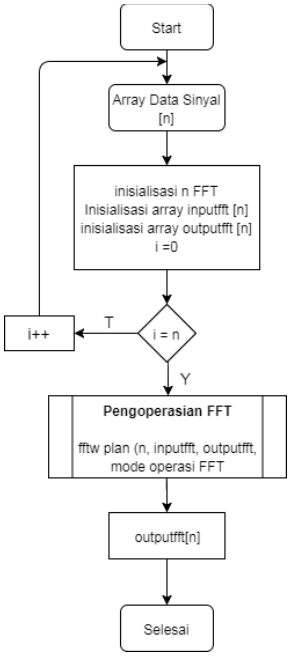
```
//--- Temp Channel B
always @(negedge reset_n or posedge addaADB_DCO)
begin
      if (!reset_n) begin
            tempa2db_data     <= 14'd0;
      end
      else begin
            tempa2db_data     <= addaADB_D;
      end
end
```

Source code penambahan dua bit LSB pada masing-masing channel:

```
//---  Channel A
always @(negedge reset_n or posedge freq2M)
begin
      if (!reset_n) begin
            a2da_data    <= 14'd0;
      end
      else begin
            a2da_data    <= {tempa2da_data, 2'b00};  //Data ADC Fix
channel A
      end
```

```
end

//---  Channel B

always @(negedge reset_n or posedge freq2M)

begin

      if (!reset_n) begin

            a2db_data    <= 14'd0;

      end

      else begin

            a2db_data    <= {tempa2db_data, 2'b00}; //Data ADC Fix
channel B

      end

end
```

3.2.3 Transformasi Fourier



Gambar 7. Diagram Alir FFT menggunakan *library* FFTW

Fungsi FFT dapat digunakan dengan memasukkan library fftw pada *source code* yang akan dikompilasi silang melalui Linux, FFT tersebut melakukan DFT 4096 point dengan input single channel dan akan dikirimkan dengan TCP/IP 2048 point . Berikut *source code* yang digunakan:

```
#include <fftw3.h>

#include <math.h>

#include <float.h>



#define NUM_POINTS 4096

#define NUM_POINTS_MAG 1024

#define REAL 0

#define IMAG 1


unsigned short int setengah = NUM_POINTS_MAG/2;


int main (){

   fftwf_complex in[NUM_POINTS];

   fftwf_complex out[NUM_POINTS];

   float mag2[setengah];


   while (1) {

      if (full ==1){


      short int i = 0;
```

```
      while(i < NUM_POINTS){


/////////////////    FFT     /////////////////


         for (i = 0; i < NUM_POINTS ; ++i){

            in[i][REAL] = ch_a_si[i]/25000.0;

            in[i][IMAG] = 0;

         }


         fftwf_plan   p   =   fftwf_plan_dft_1d(NUM_POINTS,   in,   out,
FFTW_FORWARD, FFTW_ESTIMATE);


         fftwf_execute(p);  //repeat as needed


         for (i=0; i < setengah; ++i){

            mag2[i]=sqrt((out[i][REAL] * out[i][REAL]) + (out[i][IMAG]
* out[i][IMAG]));

            fprintf(fp3,"%f\n", mag2[i]) ;

         }


         for (i=0; i < setengah; ++i){

            mag2[i]=sqrt((out[setengah+i][REAL]*out[setengah+i][REAL])
+ (out[setengah+i][IMAG]*out[setengah+i][IMAG]));

            fprintf(fp3,"%f\n", mag2[i]) ;

         }
```

```
          for (i=0; i < setengah; ++i){

             mag2[i]=sqrt((out[(2*setengah)       +       i][REAL]       *
out[(2*setengah)   +   i][REAL])   +   (out[(2*setengah)   +   i][IMAG]   *
out[(2*setengah)+ i][IMAG]));

            fprintf(fp3,"%f\n", mag2[i]) ;



          }



          for (i=0; i < setengah; ++i){

             mag2[i]=sqrt((out[(3*setengah)       +       i][REAL]       *
out[(3*setengah)   +   i][REAL])   +   (out[(3*setengah)   +   i][IMAG]   *
out[(3*setengah)+ i][IMAG]));

             fprintf(fp3,"%f\n", mag2[i]) ;
          }



          fclose (fp3);

          fftwf_destroy_plan(p);

          fftwf_cleanup();



}



}

   return 0;

}
```

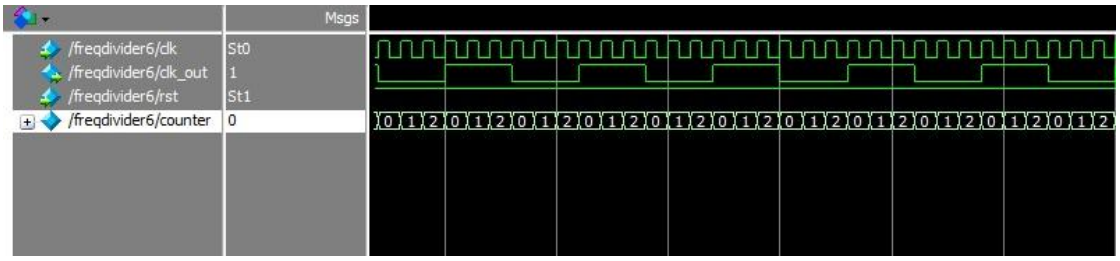
(Halaman ini sengaja dikosongkan)

BAB IV

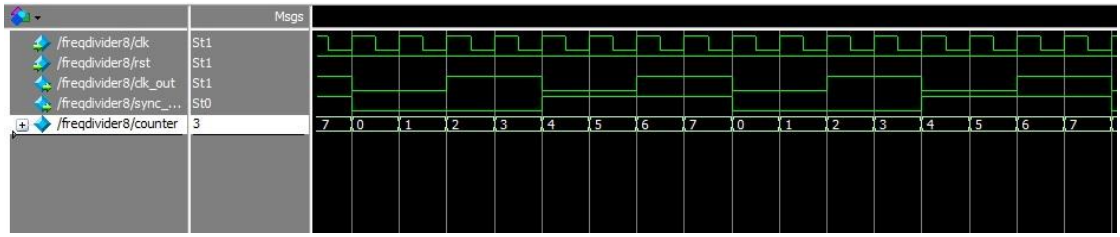
PENGUJIAN DAN HASIL

4.1. Control Unit

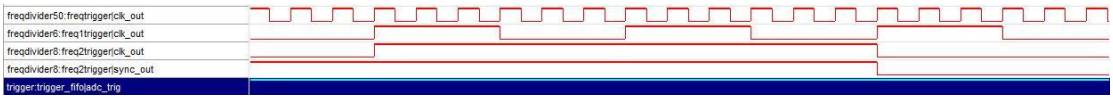
4.1.1. Frequency Divider



Gambar 8. Hasil Simulasi Modelsim pada *Frequency Divider* 6



Gambar 9. Hasil Simulasi Modelsim pada *Frequency Divider* 8



Gambar 10. *Output Frequency Divider* pada SignalTap Analyzer II

Input dari *frequency divider* adalah *clock* internal FPGA dengan frekuensi 50 MHz. Output yang diinginkan dari modul *frequency divider* adalah *clock* dengan frekuensi 2.0833 MHz (sinyal `clk_out`) dan 1.0416 MHz (sinyal `sync_out`). Maka *frequency divider* harus mampu membagi sebesar 50 MHz/2.083 MHz = 24 untuk l `clk_out` serta 50 MHz/1.0416 MHz = 48 untuk sinyal `sync_out`.

Agar lebih mudah dalam implementasi, *frequency divider* dibagi menjadi 2 bagian, yaitu *frequency divider 6* yang membagi frekuensi input sebesar 6, dan *frequency divider* 8 yang membagi frekuensi input sebesar 4 atau 8.

Dari hasil simulasi pada **Gambar 8.** terlihat bahwa algoritma *frequency divider 6* sudah benar. Frekuensi output sudah berhasil didapat sebesar (frekuensi input)/6. Hal ini terlihat dari panjang satu gelombang output sama dengan panjang enam gelombang input.

Dari hasil simulasi pada **Gambar 9.** terlihat bahwa algoritma *frequency divider 8* sudah benar. Frekuensi output sudah berhasil didapat, yaitu `clk_out` sebesar frekuensi input/4, dan `sync_out` sebesar frekuensi input/8. Hal ini terlihat dari panjang gelombang. Panjang satu gelombang `clk_out` sama dengan panjang empat gelombang input. Panjang satu gelombang `sync_out` sama dengan panjang delapan gelombang input.

Maka, hasil dari simulasi menunjukkan bahwa `clk_out` memiliki frekuensi sebesar 50/(6*4) = 2.0833 MHz dan `sync_out` memiliki frekuensi sebesar 50/(6*8) = 1.0416 MHz.

Kemudian, *frequency divider* diperiksa kembali setelah diimplementasikan pada *board*. Pemeriksaan dilakukan menggunakan SignalTap Analyzer, dan terlihat pada **Gambar 10**. Dari gambar tersebut terlihat kembali bahwa panjang satu gelombang `clk_out` sama dengan panjang 24 gelombang input 50 MHz. Panjang satu gelombang `sync_out` sama dengan panjang 48 gelombang input 50 MHz.

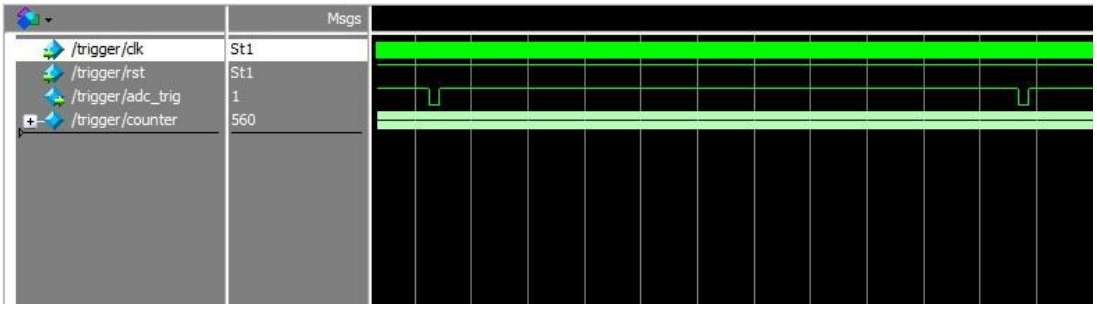
Maka, hasil pemeriksaan SignalTap Analyzer membuktikan bahwa *frequency divider* telah bekerja dengan benar, dan memiliki output berupa *clock* dengan frekuensi 2.0833 MHz dan 1.0416 MHz.

4.1.2 Trigger

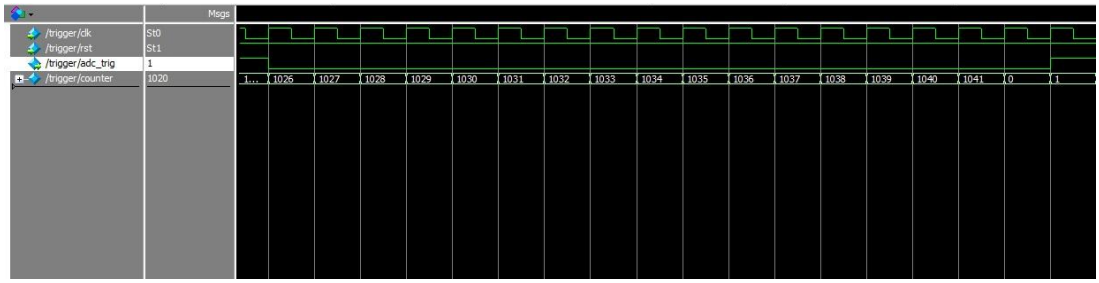
trigger digunakan sebagai pengendali laju penulisan data yang akan dilakukan oleh FIFO. *Trigger* dipergunakan pada saat *guard time*. Sinyal radar yang saat ini diasumsikan dengan menggunakan generator sinyal. Berdasarkan persamaan (2) didapatkan jumlah data yang tidak akan dimasukkan ke FIFO selama selama *guard time* berlangsung.

$$Data\ loss = \frac{t_{guard\ time}}{t_{PRI}} \times trigger\ clock \qquad \text{.... (2)}$$

Berikut hasil simulasi *timing* diagram dari *trigger*:



(a)

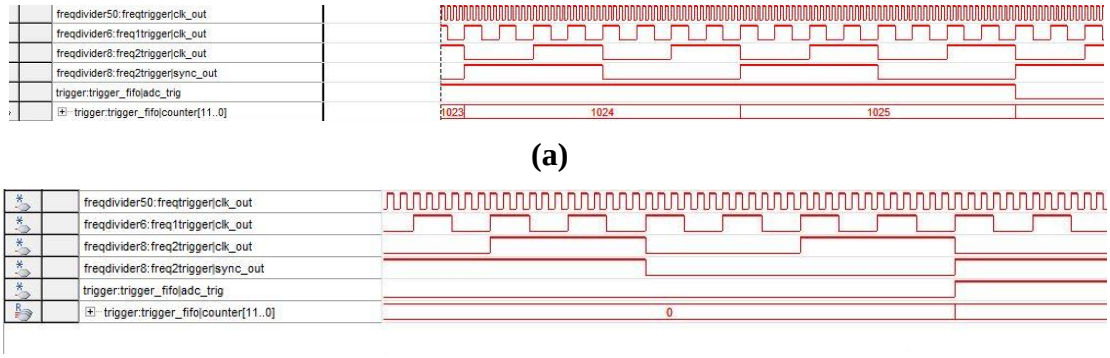


(b)

Gambar 11. Hasil simulasi Modesim pada modul trigger

Pada simulasi tersebut trigger akan mengatur `write_en` data pada FIFO ketika counter dari submodul tersebut mencapai 1025 sampai 1041. Selama counter 1025 mencapai

1041 berarti akan mengubah `write_en` pada FIFO menjadi 0, selama hal tersebut terjadi akan ada 16 data yang tidak akan dimasukkan ke dalam FIFO sebelum `write_en` akan diubah kembali menjadi 1. Karena *clock* dari *trigger* dua kali lebih lambat dari *clock* FIFO ini berarti akan ada kurang lebih 32 data yang tidak dimasukkan selama 2052 data yang akan masuk ke dalam FIFO.



Gambar 12. Output *trigger* pada SignalTap Analyzer II. (a) Mulai *trigger*, (b) Akhir *trigger*

Pengujian *trigger* pada SignalTap Analyzer II, **Gambar 12 (a)** merupakan kondisi yang sesuai dengan simulasi menggunakan modelsim dan mengalami kondisi yang sama dengan simulasi pada **Gambar 11**. Jadi, submodul *trigger* yang telah diimplementasikan sudah bekerja dengan benar.

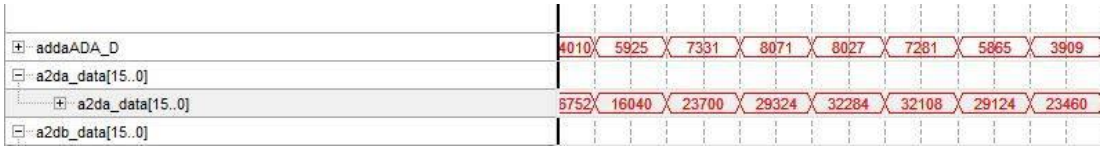
4.2 Ekspansi Bit

Untuk kemudahan dalam mengakuisisi data dari FPGA ke HPS maka diperlukan penambahan bit untuk tiap *channel* yang digunakan sebesar 2 bit pada LSB. Seperti contoh ilustrasi berikut ini:

x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0
Channel B																Channel A															

Gambar 13. Ilustrasi Penambahan 2 Bit LSB pada Masing-Masing Channel

Penambahan 2 bit pada LSB bisa dilakukan jika dan hanya jika *sampling* sinyal dari awal dilakukan dalam mode *two's complement* untuk menghindari perubahan nilai dari data yang masuk.



Gambar 14. Hasil Pengujian Ekspansi Bit Menggunakan SignalTap Analyzer II pada `a2da_data` (Channel A)

Berdasarkan **Gambar 14**, penambahan 2 bit LSB ini menyebabkan naiknya nilai 4 kali lipat sesuai dengan ilustrasi pada **Gambar 13**. Pada pengujian menggunakan SignalTap Analyzer II, secara konsisten tiap-tiap data yang ditambahkan bit-nya naik menjadi 4 kali lipat dan sebanding untuk semua data yang masuk sehingga tidak perlu dikhawatirkan terjadinya *error* yang mengubah informasi mengenai frekuensi yang diukur. Jadi, ekspansi bit yang diimplementasikan sudah bekerja sesuai yang diharapkan.

4.3 Fast Fourier Transformation

Pada pengujian submodul FFT dilakukan dengan tipe data float (32 bit), sedangkan tipe data yang masuk ke dalam HPS adalah tipe data short integer 16 bit sehingga perlu dilakukan operasi data input dengan tipe data float yang akan menghasilkan tipe data float tanpa mengubah nilai dari data input. Hal ini dilakukan karena submodul FFT yang digunakan adalah *single precision* data, yakni tipe data float.

Tabel 4. Input FFT 128 Poin dengan Frekuensi 100 KHz dan 500 KHz

n	Frekuensi (KHz)		n	Frekuensi (KHz)		n	Frekuensi (KHz)		n	Frekuensi (KHz)	
	100	500		100	500		100	500		100	500
1	0,3616	1,27968	**33**	-0,66448	-1,31072	**65**	0,8672	0,43632	**97**	0,84464	0,8992
2	0,71728	-0,0896	**34**	-0,972	0,0136	**66**	1,12544	-1,20384	**98**	1,10128	0,96736
3	1,0024	-1,31072	**35**	-1,19792	1,28432	**67**	1,26144	-0,61424	**99**	1,25648	-0,80432
4	1,2144	-0,11056	**36**	-1,31072	0,11888	**68**	1,29456	1,09872	**100**	1,29952	-1,09728

5	1,28912	1,27808	**37**	-1,31072	-1,2992	**69**	1,20832	0,72832	**101**	1,22192	0,64224
6	1,272	0,23632	**38**	-1,19792	-0,31168	**70**	-1,21488	-1,03936	**102**	1,0336	1,1504
7	1,11744	0,03088	**39**	-0,9752	1,23072	**71**	-1,00096	-0,88464	**103**	0,75008	-0,52096
8	0,872	0,0136	**40**	-0,664	0,4376	**72**	-0,69728	0,89808	**104**	0,39552	-1,2416
9	0,54672	1,28656	**41**	-0,29936	-1,20544	**73**	-0,33312	0,97056	**105**	0,00832	0,336
10	0,1688	0,12096	**42**	0,0976	-0,61904	**74**	0,05888	-0,80448	**106**	-0,38624	1,2608
11	-0,22784	-1,2984	**43**	0,4816	1,09824	**75**	0,44512	-1,09952	**107**	-0,74208	-0,20768
12	-0,60784	-0,312	**44**	0,8264	0,72384	**76**	0,79136	0,64048	**108**	-1,03504	-1,30768
13	-0,92096	1,23056	**45**	1,07424	-1,03616	**77**	1,06368	1,14896	**109**	-1,23424	0,01456
14	-1,16896	0,44288	**46**	1,27216	-0,8856	**78**	1,2376	-0,52224	**110**	-1,31072	1,28784
15	-1,30336	-1,20704	**47**	1,28608	0,89776	**79**	1,29856	-1,2416	**111**	-1,29648	0,1192
16	-1,31072	-0,61856	**48**	1,2512	0,96752	**80**	1,23952	0,3392	**112**	-1,15088	-1,3016
17	-1,21936	-0,61856	**49**	1,0144	-0,80592	**81**	1,07008	1,2576	**113**	-0,90544	1,23328
18	-1,01984	1,10064	**50**	0,78	-1,09856	**82**	0,80528	-0,2088	**114**	-0,5792	0,43376
19	-0,72112	0,72656	**51**	0,42864	0,63376	**83**	0,4576	-1,31072	**115**	-0,2016	-1,20544
20	-0,35744	-1,03632	**52**	0,0296	1,15056	**84**	0,07184	0,01136	**116**	0,19424	-0,61456
21	0,03488	-0,88576	**53**	-0,35552	-0,5216	**85**	-0,32192	1,28848	**117**	0,56976	1,10192
22	0,42	0,89664	**54**	-0,72384	-1,2448	**86**	-0,6872	0,11856	**118**	0,89376	0,72464
23	0,77008	0,97024	**55**	-1,00816	0,33536	**87**	-0,99136	-1,29744	**119**	1,13504	-1,03408
24	1,05136	-0,80336	**56**	-1,23632	1,25696	**88**	-1,2088	-0,30832	**120**	1,27024	-0,8856
25	1,23024	-1,09984	**57**	-1,31072	-0,20864	**89**	-1,31072	1,23248	**121**	1,2952	0,90112
26	1,29664	0,63488	**58**	-1,30896	-1,31072	**90**	-1,31072	0,43648	**122**	1,19424	0,96912
27	1,25056	1,15184	**59**	-1,16384	0,01504	**91**	-1,18496	-1,20864	**123**	0,99264	-0,80224
28	1,0888	-0,52256	**60**	-0,92928	1,28624	**92**	-0,95568	-0,61424	**124**	0,6952	-1,09776
29	0,8224	-1,24448	**61**	-0,61136	0,11728	**93**	-0,63936	1,09648	**125**	0,3344	0,63888
30	0,4832	0,3384	**62**	-0,22448	-1,2992	**94**	-0,26592	0,72944	**126**	-0,05776	1,15024
31	0,10336	1,25856	**63**	0,16144	-0,3112	**95**	0,12608	-1,03552	**127**	-0,44864	-0,52288
32	-0,29568	-0,20896	**64**	0,54224	1,23088	**96**	0,50912	-0,88448	**128**	-0,79568	-1,24272

Tabel 5. Hasil FFT 128/2 Poin, Frekuensi 100 KHz dan 500 KHz dengan Menggunakan Submodul FFT dan MATLAB

MATLAB						Submodul					
n	Frekuensi (KHz)		n	Frekuensi (KHz)		n	Frekuensi (KHz)		n	Frekuensi (KHz)	
	100	500		100	500		100	500		100	500
1	8,8918	0,631	**33**	1,6537	15,8974	**1**	8,8918	0,631	**33**	1,6537	15,8974
2	5,3994	1,2969	**34**	1,8878	8,6261	**2**	5,3994	1,2969	**34**	1,8878	8,6261
3	10,6667	1,3133	**35**	2,0141	5,6723	**3**	10,6667	1,3133	**35**	2,0141	5,6723
4	13,4075	1,3248	**36**	1,5110	4,1187	**4**	13,4075	1,3248	**36**	1,5110	4,1187
5	10,326	1,3874	**37**	2,1802	3,2455	**5**	10,326	1,3874	**37**	2,1802	3,2455
6	44,1450	1,444	**38**	1,1302	2,7873	**6**	44,1450	1,444	**38**	1,1302	2,7873
7	11,0531	1,5113	**39**	2,1964	2,6227	**7**	11,0531	1,5113	**39**	2,1964	2,6227
8	60,2495	1,6177	**40**	0,7814	2,5725	**8**	60,2495	1,6177	**40**	0,7814	2,5725
9	9,6264	1,7282	**41**	2,1682	2,6618	**9**	9,6264	1,7282	**41**	2,1682	2,6618
10	16,3859	1,8377	**42**	0,9154	2,6880	**10**	16,3859	1,8377	**42**	0,9154	2,6880
11	8,7449	1,9546	**43**	1,9999	2,7120	**11**	8,7449	1,9546	**43**	1,9999	2,7120
12	7,8102	2,0591	**44**	1,1905	2,7266	**12**	7,8102	2,0591	**44**	1,1905	2,7266
13	7,9051	2,1407	**45**	1,8162	2,66	**13**	7,9051	2,1407	**45**	1,8162	2,66
14	3,9021	2,2393	**46**	1,3995	2,579	**14**	3,9021	2,2393	**46**	1,3995	2,579
15	6,8088	2,2627	**47**	1,4176	2,4598	**15**	6,8088	2,2627	**47**	1,4176	2,4598
16	1,9085	2,2899	**48**	1,6665	2,2534	**16**	1,9085	2,2899	**48**	1,6665	2,2534
17	5,8001	2,2658	**49**	1,0346	2,0205	**17**	5,8001	2,2658	**49**	1,0346	2,0205
18	1,7146	2,1749	**50**	1,866	1,7354	**18**	1,7146	2,1749	**50**	1,866	1,7354
19	4,813	2,0492	**51**	0,8155	1,439	**19**	4,813	2,0492	**51**	0,8155	1,439
20	2,2746	1,8723	**52**	1,8849	1,1126	**20**	2,2746	1,8723	**52**	1,8849	1,1126
21	3,7017	1,5955	**53**	0,6004	0,7585	**21**	3,7017	1,5955	**53**	0,6004	0,7585
22	2,7825	1,2436	**54**	1,8201	0,3960	**22**	2,7825	1,2436	**54**	1,8201	0,3960
23	2,7032	0,8398	**55**	0,897	0,1644	**23**	2,7032	0,8398	**55**	0,897	0,1644
24	3,1051	0,3202	**56**	1,8087	0,419	**24**	3,1051	0,3202	**56**	1,8087	0,419
25	1,8952	0,3655	**57**	1,1313	0,7040	**25**	1,8952	0,3655	**57**	1,1313	0,7040
26	3,1942	1,1885	**58**	1,5434	1,0575	**26**	3,1942	1,1885	**58**	1,5434	1,0575
27	1,172	2,2766	**59**	1,1959	1,3475	**27**	1,172	2,2766	**59**	1,1959	1,3475
28	3,0595	3,7870	**60**	1,092	1,6174	**28**	3,0595	3,7870	**60**	1,092	1,6174
29	1,0381	6,1671	**61**	1,5834	1,8308	**29**	1,0381	6,1671	**61**	1,5834	1,8308

30	2,8252	10,8967	**62**	0,9531	2,0258	**30**	2,8252	10,8967	**62**	0,9531	2,0258
31	1,3068	27,7354	**63**	1,7619	2,1526	**31**	1,3068	27,7354	**63**	1,7619	2,1526
32	2,4297	73,2666	**64**	0,695	2,2455	**32**	2,4297	73,2666	**64**	0,695	2,2455

Pada uji coba yang dilakukan, sinyal generator yang digunakan merupakan sinyal sinusoid dengan frekuensi 100 KHz dan 500 KHz. Untuk mengetahui frekuensi yang diukur perlu dicari resolusi terlebih dahulu dengan persamaan (3)

$$Resolusi = \frac{1}{fft\ point} \times \frac{f_{max}}{2} \qquad \text{.... (3)}$$

Setelah mendapatkan resolusi dari FFT bisa diketahui frekuensi yang diukur dengan cara mengalikan resolusi yang diperoleh dengan data ke-n yang magnitudonya paling tinggi. Untuk uji coba ini digunakan FFT 128/2 poin sehingga resolusi yang didapatkan adalah

$$Resolusi = \frac{1}{64} \times \frac{1{,}041\ MHz}{2} = 16{,}265\ KHz \qquad \text{.... (4)}$$

Sebagai pembanding dan verifikasi bahwa data telah benar, maka diuji coba pula data input pada MATLAB yang juga mempergunakan *library* FFT yang sama dengan submodul yang diimplementasikan. Jika diperhatikan pada **Tabel 5 .** untuk frekuensi 100 KHz magnitudo tertinggi ada pada data ke-8, berarti frekuensi yang terukur adalah 130,125 KHz, sedangkan untuk 500 KHz ada pada data ke-32, yang berarti frekuensi terukur adalah 520 kHz baik pada submodul FFT dan MATLAB.

Dari pengujian tersebut membuktikan bahwa yang digunakan telah benar. Selain itu semakin tinggi poin FFT yang digunakan, maka semakin kecil resolusi dan pengukuran frekuensi semakin akurat.

(Halaman ini sengaja dikosongkan)

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan pembahasan yang telah dijelaskan sebelumnya baik pada bagian peranannya, implementasi dan pengujian dapat disimpulkan bahwa :

- Pengendalian kecepatan penulisan data oleh FIFO dapat dilakukan dengan memperhitungan *guard time* dan PRI radar. Untuk membuat data saat *guard time* tidak masuk ke dalam FIFO maka dibuat counter bersyarat untuk disambungkan dengan FIFO sehingga efisiensi memori dapat dilakukan.
- Penambahan bit yang dilakukan hanya bisa dilakukan apabila tipe data pada sistem yang dioperasikan dalam bentuk *two's complement* mengingat tipe data two's complement dapat dengan mudah dimanipulasi untuk keperluan lain sehingga penambahan bit tidak akan mengubah informasi yang diterima dan mempermudah akuisisi data yang akan dilakukan oleh HPS nanti.
- Pembagi frekuensi atau *frequency divider* juga dapat dibuat dengan memanfaatkan *counter* yang bersanding dengan clock sumber sehingga frekuensi sampling yang akurat dan presisi dapat dicapai tanpa menggunakan PLL.
- Untuk mendapatkan resolusi yang tinggi membutuhkan point FFT yang besar pula. FFT dengan point yang besar mempergunakan banyak memori pula. Selain itu, FFT sudah sesuai dengan tipe processor yang digunakan, yaitu *single precision* yang mengharuskan operasi dari FFT dilakuakan dalam tipe data float.

5.2. Saran

Berdasarkan hasil perancangan, implementasi dan pengujian yang telah dilakukan, sara yang dapat diberikan diantaranya :

- Untuk melakukan FFT yang poinnya besar harus diimplementasikan pada processor dengan kapasitas RAM yang besar.
- Modul-modul yang telah dikerjakan boleh diuji dengan menggunakan sinyal masukan dari mixer radar FMCW yang sebenarnya.

DAFTAR PUSTAKA

[1] http://www.fftw.org/fftw2_doc/fftw_6.html : Installation and Customization FFTW

[2] Oppenheim. 1989. Discrete-Time Signal Processing. New Jersey. Prentice Hall

[3] Suto, Kyohei , & Sumantyo, J.T Sri. Development SAR Base-Band Signal Processor Using FPGA and On-Board PC. IEEE. Pp 672-675.

[4] Matteo Frigo and Steven G. Johnson, "The Design and Implementation of FFTW3" *Proceedings of the IEEE* 93 (2), 216-231 (2005).

[5] Hyun, Eugin, & Kim, Sang-Dong. FPGA Based Signal Processing Module Design and Implementation for FMCW Vehicle Radar Systems.

[6] Kreyszig, Erwin. 2015. Advanced Engineeering Mathematics. New Jersey. Wiley.

[7]