IMPLEMENTASI *INVERSE DISCRETE COSINE TRANSFORM* (IDCT) PADA *FIELD PROGRAMMABLE GATE ARRAY* (FPGA)

SKRIPSI

KONSENTRASI REKAYASA KOMPUTER

Diajukan Untuk Memenuhi Sebagian Persyaratan

Memperoleh Gelar Sarjana Teknik

Disusun oleh :

SAFRIL WAHYU PAMUNGKAS

NIM. 0810633081 - 63

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2014

LEMBAR PERSETUJUAN

IMPLEMENTASI *INVERSE DISCRETE COSINE TRANSFORM* (IDCT) PADA *FIELD PROGRAMMABLE GATE ARRAY* (FPGA)

SKRIPSI

KONSENTRASI REKAYASA KOMPUTER

Diajukan untuk Memenuhi Persyaratan
Memperoleh Gelar Sarjana Teknik

Disusun oleh:

SAFRIL WAHYU PAMUNGKAS
NIM. 0810633081-63

Telah diperiksa dan disetujui oleh :

Pembimbing I	Pembimbing II
Waru Djuriatno, ST., MT.	**Mochammad Rif'an, ST., MT.**
NIP. 19690725 199702 1 001	**NIP. 19710301 200012 1 001**

LEMBAR PENGESAHAN

IMPLEMENTASI *INVERSE DISCRETE COSINE TRANSFORM* (IDCT) PADA *FIELD PROGRAMMABLE GATE ARRAY* (FPGA)

SKRIPSI

KOSENTRASI REKAYASA KOMPUTER

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh :

SAFRIL WAHYU PAMUNGKAS

NIM. 0810633081 – 63

Skripsi ini telah diuji dan dinyatakan lulus pada
tanggal 5 Februari 2014

Majelis Penguji :

DOSEN PENGUJI I	DOSEN PENGUJI II
Ir. Nanang Sulistiyanto, M.Eng.	**Adharul Muttaqin, ST.,MT.**
NIP. 19700113 199403 1 002	**NIP. 19760121 200501 1 001**

DOSEN PENGUJI III

Ir. Muhammad Aswin, MT.
NIP. 19640626 199002 1 001

Mengetahui,
Ketua Jurusan Teknik Elektro

Muhammad Aziz Muslim, ST., MT,. PhD.
NIP. 19741203 200012 1 001

ABSTRAK

Safril Wahyu Pamungkas., Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Januari 2014, *Implementasi Inverse Discrete Cosine Transform (IDCT) pada Field Programmable Gate Array (FPGA),* Dosen Waru Djuriatno, ST. MT. dan Mochammad Rif'an, ST. MT.

Inverse Discrete Cosine Transform (IDCT) merupakan salah satu tahap penting dalam proses pengolahan citra digital terutama dalam pemulihan citra diam (*still image*) yang tersusun atas banyak nilai piksel. Proses *Inverse Discrete Cosine Transform* (IDCT) melibatkan komputasi dengan banyak proses perkalian untuk mengolah jumlah data yang besar. Pengolahan nilai piksel ke dalam beberapa *Macro Block Unit* (MCU) (1 MCU tersusun atas 8x8 piksel) merupakan suatu cara untuk menghasilkan komputasi cepat *Inverse Discrete Cosine Transform* (IDCT).

Algoritma 1-D IDCT *Loeffler-Ligtenberg-Moschytz* (LLM) merupakan hasil pengembangan dari persamaan 1D-IDCT asli yang mampu meminimalisir penggunaaan operasi perkalian dari 64 pengali menjadi 14 pengali. Implementasi 2-D IDCT ke perangkat keras Digilent Nexys 2 Development Board FPGA Xilinx Spartan 3E dilakukan dengan implementasi 1-D IDCT untuk mengolah 8x8 data terhadap baris dan kolom.

Perancangan program VHDL (*Very high speed integrated circuit Hardware Description Languange*) 2-D IDCT menggunakan *software* Xilinx ISE, kemudian diimplementasikan pada FPGA Xilinx Spartan 3E. Sumber data untuk implementasi IDCT menggunakan mikrokontroler dengan operasi interupsi dengan pemicu *clock* FPGA. Pengujian dilakukan dengan cara membandingkan hasil komputasi FPGA dengan hasil komputasi *software* penghitung serbaguna.

Akurasi komputasi implementasi IDCT menunjukkan adanya *error* yang masih dapat ditoleransi terhadap hasil komputasi IDCT dengan *software*. Jumlah *slice* yang digunakan untuk implementasi unit 2-D IDCT sebesar 54 % dari total 4656 *slices.*

Kata Kunci: *Inverse Discrete Cosine Transform*, Algoritma *Loeffler-Ligtenberg-Moschytz*, FPGA, VHDL.

PENGANTAR

Syukur Alhamdulillah, segenap puji dan syukur penulis ucapkan kehadirat Allah SWT yang telah melimpahkan rahmat, hidayah, ridho, nikmat dan karunia- Nya sehingga penulis dapat menyelesaikan skripsi ini dengan judul "*Implementasi Inverse Discrete Cosine Transform (IDCT) pada Field Programmable Gate Array (FPGA)*" sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya Malang. Tidak lupa shalawat serta salam semoga senantiasa tercurahkan untuk Rasulullah Muhammad SAW beserta keluarga, sahabat, kerabat dan para pengikutnya sampai akhir jaman.

Skripsi ini tidak dapat terselesaikan dengan baik tanpa dukungan dan bantuan dari berbagai pihak baik secara langsung maupun tidak langsung. Oleh karena itu, pada kesempatan ini penulis ingin mengucapkan terima kasih kepada:

1. Bapak Aziz Muslim, ST., MT., Ph.D. sebagai Ketua Jurusan Teknik Elektro Universitas Brawijaya,
2. Bapak Hadi Suyono, ST., MT., Ph.D. sebagai Sekertaris Jurusan Teknik Elektro Universitas Brawijaya,
3. Bapak Mochammad. Rif'an, ST., MT. sebagai Ketua Program Studi Teknik Elektro Jurusan Teknik Elektro Universitas Brawijaya
4. Bapak Waru Djuriatno, ST. MT. sebagai Ketua Kelompok Dosen Keahlian Rekayasa Komputer Jurusan Teknik Elektro Universitas Brawijaya sekaligus sebagai Dosen Pembimbing I , atas segala kesabarannya dalam membimbing, memberikan pengarahan, ide, dan saran serta motivasi yang telah diberikan.
5. Bapak Mochammad. Rif'an, ST., MT. sebagai Dosen Pembimbing II, atas segala kesabarannya dalam membimbing, memberikan pengarahan, ide, dan saran serta motivasi yang telah diberikan,
6. Ibu Rusmi Ambarwati, ST.,MT. sebagai Dosen Pembimbing akademik atas segala bimbingan, yang telah diberikan selama menempuh studi di Teknik Elektro,
7. Ibunda Siti Masrikah, Mbak Ferny, Mbak Reni serta keluarga besar saya yang telah memberikan banyak dukungan secara moral maupun materi,

8. Bapak dan Ibu Dosen serta seluruh karyawan jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya,
9. Teman seperjuangan Yan Felix Monangin yang telah membantu memberikan saran, arahan dan dukungan hingga terselesaikannya skripsi ini,
10. Seluruh Asisten Laboratorium Informatika dan Komputer serta Asisten Laboratorium Dasar Komputer dan Pemrograman karena selalu memberikan bantuan, dukungan dan semangat hingga terselesaikannya skripsi ini,
11. Para Senior TEUB Mas Yulius Candra, Mas Aflah, Mas Gandhes, Mas Yudo, Mas Dena, Mas Wahyu, Mas Aji dan seluruh keluarga "Mampang X17" atas bantuan, dukungan dan semangat hingga terselesaikannya skripsi ini
12. Anas Setiawan,ST, Eryc Tri Juni,ST, V Nyorendra,ST, A Angga,ST, Tunggul W,ST, Okta, Blontank, Ruditta"Hlok"Devianti, dan seluruh teman-teman CONCORDES 2008 serta teman-teman paket E,
13. Semua pihak yang tidak dapat saya sebutkan satu persatu, atas dukungannya dan bantuannya.

Penulis menyadari bahwa yang tersusun dalam skripsi ini masih banyak kekurangan karena keterbatasan materi dan pengetahuan yang dimiliki. Karena itu kritik dan saran sangat diharapkan. Akhir kata, penulis mengharapkan semoga penelitian ini bermanfaat untuk kita semua, rekan-rekan mahasiswa khususnya dan bagi seluruh pembaca pada umumnya.

Malang, Januari 2014

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pemrosesan sinyal (*signal processing*) sudah umum digunakan untuk memanipulasi isyarat analog atau digital. Saat ini *signal processing* telah dikombinasikan dengan teknologi *chip/processor* yang telah banyak digunakan dalam beberapa bidang seperti telekomunikasi, *audio processing* dan juga *image processing*.

Digital Signal Processing (DSP) telah menjadi teknik yang banyak digunakan untuk mengolah isyarat secara digital daripada mengolah isyarat secara analog. DSP memberikan beberapa keuntungan dibandingakan dengan pengolahan isyarat secara analog antara lain memiliki ketahanan lebih terhadap *noise*, dan biaya yang diperlukan lebih murah dari sisi *hardware.*

Salah satu penerapan DSP adalah pada *image processing.* Pengolahan isyarat gambar (gambar diam atau video) menjadi isyarat digital diperlukan media penyimpanan yang besar jika dilakukan tanpa pemampatan atau *compression*. Demikian pula dalam hal transmisi data, tanpa pemampatan akan meningkatkan beban jaringan dan memerlukan waktu yang lebih lama dalam mentransmisikan data.

Discrete Cosine Transform (DCT) dan *Inverse Discrete Cosine Transform (IDCT)* merupakan salah satu teknik transformasi yang digunakan dalam *compression / decompression* gambar dan video. DCT melakukan transformasi dari kawasan spasial ke kawasan frekuensi sedangkan IDCT melakukan transformasi dari kawasan frekuensi ke kawasan ruang sehingga IDCT akan memulihkan isyarat hasil pengolahan DCT.

Citra/gambar merupakan isyarat dua dimensi (2D) maka dalam pengolahanya digunakan algoritma 2D-DCT dan 2D-IDCT. Pengolahan isyarat gambar dua dimensi secara langsung dengan proses 2D-DCT dan 2D-IDCT akan memerlukan banyak komputasi kompleks, oleh karena itu proses komputasi 2D dapat dilakukan dengan menggunakan satu dimensi (1D) secara horisontal dan vertikal.

Banyak algoritma komputasi yang telah ditemukan untuk mengolah 2D-DCT atau 2D-IDCT (8 x 8 data) dengan menggunakan 1D-DCT atau 1D-IDCT. Tujuan penggunaan algoritma tersebut adalah untuk mengurangi komputasi yang terjadi dan mempercepat pengolahan isyarat digital sehingga jika diimplementasikan ke dalam hardware akan mengurangi kompleksitas perancangan.

IDCT merupakan algoritma berbasis cosinus, untuk N masukan akan menghasilkan N keluaran, artinya jika diimplementasikan ke perangkat keras, IDCT untuk N masukan akan membutuhkan N memori sebagai media penyimpanannya. Sedangkan transformasi Fourier merupakan algoritma berbasis eksponensial, sebuah masukan jika difourier-kan akan menghasilkan keluaran real dan imajiner. Jika transformasi Fourier diimplementasikan ke dalam perangkat keras maka untuk N masukan akan membutuhkan 2N memori. Perbandingan pemakaian perangkat keras antara transformasi Fourier dengan IDCT menunjukan bahwa IDCT 50% lebih hemat dibandingkan dengan transformasi Fourier.

Implementasi algoritma 1D-IDCT kedalam hardware untuk pemulihan isyarat gambar dua dimensi menjadi tantangan tersendiri. Peralatan digital yang digunakan harus dapat menghasilkan unjuk kerja yang tinggi. Dalam tugas akhir ini unjuk kerja dari implementasi hardware tersebut diukur dari segi kecepatan komputasi dan seberapa banyak kapasitas design yang digunakan. Salah satu pengembangan untai terpadu yang dapat menghasilkan keseimbangan antara kecepatan dan kapasitas adalah *field programmable gate arrays (FPGA).* FPGA adalah untai terpadu yang mempunyai komponen gerbang terprogram dan sambungan terpogram.

Keunggulan utama FPGA adalah hubungan interkoneksinya yang dominan, sehingga sangat fleksibel dalam penentuan rangkaian terbaik. Untuk merancang program ke dalam FPGA salah satunya menggunakan bahasa VHDL (*Very high speed integrated circuit Hardware Description Languange*). VHDL adalah bahasa pemrograman untuk mendeskripsikan suatu perangkat keras yang digunakan dalam desain elektronis.

Tugas akhir ini akan mengimplementasikan 1D-IDCT dan 2D-IDCT pada FPGA Xilinx Spartan 3E. Hal yang perlu di perhatikan dalam implementasi algoritma IDCT pada FPGA adalah *performance* FPGA baik dalam hal kapasitas maupun kecepatan proses dan akurasi hasil perhitungan. Jika kapasitas perancangan sistem IDCT melebihi dari kapasitas yang disediakan FPGA Xilinx Spartan 3E, akan menyebabkan implementasi IDCT ke FPGA tidak bisa dilaksanakan. Optimalisasi komputasi IDCT sudah dilakukan oleh beberapa peneliti, salah satunya oleh *Loeffler, Ligtenberg* dan *Moschytz* (LLM *algorithm*) pada tahun 1989. 1D-IDCT algoritma LLM untuk 8 masukan mampu meminimalkan proses perkalian dari 64 perkalian menjadi hanya 14 perkalian. FPGA Xilinx Spartan 3E hanya menyediakan 20 pengali, untuk itu

implementasi 2D-IDCT algoritma LLM pada FPGA Xilinx Spartan 3E pengalinya akan dirancang sendiri dengan menggunakan algoritma Booth. Keluaran 1D-IDCT dan 2D-IDCT algoritma LLM akan ditampilkan melalui *Light Emiting Diode (LED)*, serta verifikasi keluaran komputasi 2D-IDCT algoritma LLM pada FPGA akan dibandingkan dengan komputasi 2D-IDCT menggunakan MATLAB.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan secara jelas diatas, maka terdapat rumusan masalah antara lain :

1. Bagaimana cara mengimplementasikan IDCT pada FPGA dengan algoritma yang tepat untuk melakukan komputasi IDCT secara efektif dan efisien.
2. Bagaimana cara merancang desain pengali yang digunakan dalam proses komputasi terkait dengan keterbatasan pengali FPGA.
3. Bagaimana akurasi dan kinerja FPGA saat diimplementasikan sebagai IDCT.

1.3 Batasan Masalah

Rumusan masalah diatas mempunyai cakupan yang sangat luas sehingga dibutuhkan batasan masalah agar pembahasan dan pembuatan laporan tidak keluar dari judul yang telah ditetapkan. Batasan masalah tersebut adalah sebagai berikut:

1. FPGA yang digunakan adalah Xilinx Spartan 3E XC3S500E
2. Perancangan IDCT berbasis VHDL.
3. Algoritma yang digunakan dalam perancangan 8 titik 1D-IDCT adalah algoritma Loeffler-Ligtenberg-Moschytz (LLM).
4. Data masukan yang dimasukkan pada IDCT merupakan hasil komputasi dari proses DCT yang dihasilkan dari software Matlab 7.0.4
5. Sumber data masukan untuk sistem dihasilkan oleh program yang dikirim melalui mikrokontroller.
6. Matlab 7.0.4 digunakan sebagai referensi dalam membandingkan hasil IDCT.

1.4 Tujuan

Tujuan tugas akhir (skripsi) ini adalah merancang dan membuat sebuah perangkat keras yang dapat melakukan komputasi IDCT menggunakan FPGA kemudian menganalisis kinerja algoritma IDCT serta mengevaluasi keluaran data dari IDCT yang dirancang.

1.5 Manfaat

Manfaat yang diharapkan dapat diperoleh melalui pengerjaan tugas akhir ini adalah :

- Bagi Penyusun :

1. Dapat mengimplentasikan IDCT pada sebuah perangkat keras, yaitu FPGA.
2. Menambah wawasan tentang pengolahan citra digital serta teknologi FPGA.
3. Memperoleh pemahaman mengenai bahasa pemrograman perangkat keras VHDL.
4. Menerapkan ilmu yang telah diperoleh dari Teknik Elekto Konsentrasi Rekayasa Komputer Universitas Brawijaya.

- Bagi Pengguna :

1. Dapat melakukan komputasi IDCT secara cepat dan efisien dengan menggunakan FPGA.

1.6 Sistematika Penulisan

Sistematika penulisan laporan tugas kahir ini adalah sebagai berikut :

BAB I Pendahuluan

Bab ini menjelaskan tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat serta sistematika penulisan.

BAB II Dasar Teori

Bab ini menjelaskan tentang konsep dasar dari Inverse Discrete Cosine Transform (IDCT), FPGA, konsep pemrograman dengan VHDL, penjelasan tentang software Xilinx ISE dan penjelasan tentang mikrokontroller.

BAB III Metodologi

Menjelaskan metode yang digunakan dalam pengerjaan tugas akhir.

BAB IV Perancangan dan Implementasi

Bab ini berisi tentang perancangan komponen 1D-IDCT dan 2D-IDCT dengan algoritma yang digunakan sehingga bisa diimplementasikan ke FPGA.

BAB V Pengujian

Bab ini akan membahas tentang pengujian perangkat lunak yang sudah diterapkan dalam FPGA meliputi pengujian akurasi nilai data hasil dari operasi IDCT, serta kinerja perancangan sistem meliputi kapasitas dan kecepatan sistem.

BAB VI Kesimpulan dan Saran

Pada bab ini berisi tentang kesimpulan yang dicapai dari hasil pembuatan perangkat lunak dengan mempertimbangkan tujuan yang ingin dicapai dari tugas akhir ini dan memberikan saran serta perbaikan yang mungkin dilakukan untuk pengembangan aplikasi yang lebih lanjut.

BAB II

DASAR TEORI

Pada bab ini dibahas mengenai dasar teori yang digunakan untuk menunjang penulisan tugas akhir mengenai implementasi IDCT pada FPGA. Beberapa dasar teori yang dimaksud diantaranya adalah IDCT dan FPGA.

2.1 IDCT (Inverse Discrete Cosine Transform)

Pada beberapa aplikasi pengolah isyarat digital, Discrete Cosine Transform (DCT) dan Inverse Discrete Cosine Transform (IDCT) merupakan algoritma yang banyak digunakan selain Discrete Fourier Transform (DFT). DCT dan IDCT merupakan komponen penting dari standar kompresi dan dekompresi gambar atau video seperti HDTV, MPEG dan JPEG. Selain itu, DCT dan IDCT juga banyak digunakan dalam beberapa aplikasi seperti *speech coding* dan pengenalan pola.

Secara teori DCT melakukan proses transformasi data dari kawasan ruang ke dalam kawasan frekuensi, sedangkan IDCT melakukan proses kebalikan dari DCT. Pada penerapanya, operasi DCT dan IDCT bekerja pada 8 x 8 blok data yang merupakan nilai-nilai piksel dari gambar. Untuk 8 x 8 sampel data dari *x(m,n)*, dua dimensi 2D-DCT dan 2D-IDCT dirumuskan pada persamaan (2.1) untuk $0 \leq k,l \leq 7$ dan persamaan (2.2) untuk $0 \leq m,n \leq 7$.

$$Y\ k,l\ = \frac{1}{4} a\ \ k\ \ a\ \ l\ \ \sum_{n=0}^{7} \sum_{m=0}^{7} x\ m,n\ \ cos\ \ \frac{2m+1\ \Pi k}{16}\ \cos\ \ \frac{(2n+1)\Pi l}{16} \qquad (2.1)$$

$$x\ m,n\ = \frac{1}{4} a\ \ k\ \ a\ \ l\ \ \sum_{k=0}^{7} \sum_{l=0}^{7} Y\ k,l\ \ cos\ \ \frac{2m+1\ \Pi k}{16}\ \cos\ \ \frac{(2n+1)\Pi l}{16} \qquad (2.2)$$

Dengan $\alpha(0) = \frac{1}{2}$ dan $\alpha(j) = 1$ untuk $j \neq 0$.

Ukuran efisiensi dari algoritma DCT/IDCT adalah jumlah dari proses perkalian yang terlibat didalamnya. Implementasi 2D-DCT dan 2D-IDCT secara langsung akan memerlukan N^4 proses perkalian untuk N x N blok data, oleh karena itu sangat jarang diimplementasikan 2D secara langsung. Jalan lain untuk mengimplementasikan 2D-DCT dan IDCT adalah dengan menggunakan proses 8 titik 1D-DCT dan IDCT yang dioperasikan pada bagian baris dari blok data selanjutnya dilakukan operasi 8 titik 1D-DCT dan IDCT pada bagian kolom dari blok data hasil pengolahan baris.

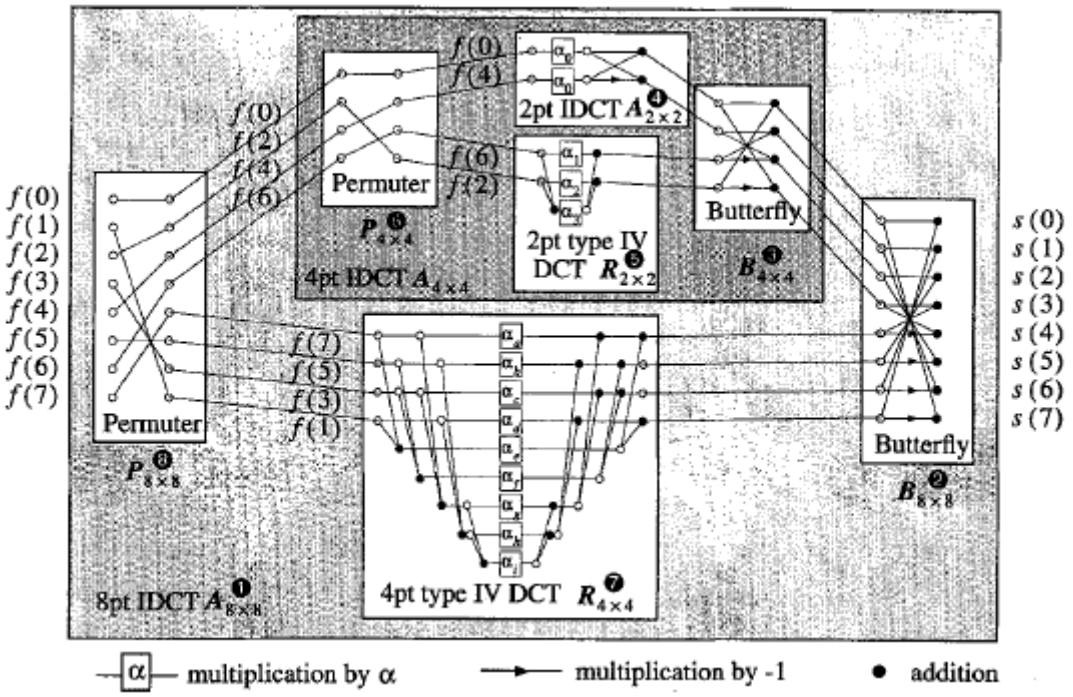
Untuk 8 titik 1D-DCT dan 8 titik 1D-IDCT dirumuskan pada persamaan (2.3) untuk $0 \leq u \leq 7$ dan persamaan (2.4) untuk $0 \leq i \leq 7$.

$$X(u) = \frac{c[u]}{2} \sum_{i=0}^{7} x[i] cos \left(\frac{(2i+1)u\Pi}{16} \right) \quad (2.3)$$

$$X(u) = \frac{c[u]}{2} \sum_{i=0}^{7} x[i] cos \left(\frac{(2i+1)u\Pi}{16} \right) \quad (2.4)$$

Dengan $c[u] = \frac{1}{\sqrt{2}}$ untuk nilai $u = 0$ dan $c[u] = 1$ untuk nilai u yang lain.

Inverse Discrete Cosine Transform (IDCT) merupakan komponen yang digunakan untuk memulihkan isyarat hasil dari proses DCT. Akurasi, kecepatan serta kapasitas *hardware* merupakan tiga hal penting dalam implementasi IDCT. Banyak algortima yang telah diperkenalkan untuk melakukan komputasi 1D-IDCT dan 2D-IDCT secara cepat dan mengurangi jumlah komputasi terutama pada proses perkalian. Pada tahun 1989, Loeffler-Ligtenberg-Moschytz (*Loeffler et al*) menemukan algoritma 8 titik IDCT yang hanya membutuhkan 14 proses perkalian, 25 proses penjumlahan dan 7 proses pengurangan seperti yang terlihat pada Gambar 2.1



Gambar 2.1 Algoritma Loeffler-Ligtenberg-Moschytz (LLM)

2.2 Algortima Booth

Algoritma Booth merupakan algoritma yang digunakan untuk perkalian bilangan biner yang memiliki tanda, atau yang mengandung representasi 2's *complement*. Algoritma Booth bisa digunakan untuk mengurangi jumlah *partial product* dan prinsip perkaliannya sama pada bilangan biner biasa, dimana dibutuhkan *partial product* yang akan dijumlahkan dan menjadi produk totalnya. Hasil algoritma Booth merupakan penjumlahan bit yang ada pada kedua *operand*, dan juga bit yang paling kiri (MSB) merupakan sebuah bit tanda (*sign*) dan tidak dapat digunakan sebagai bagian dari nilai hasil perkalian. Bilangan yang akan dikalikan selanjutnya disebut dengan terkali (*multiplicand*) dikalikan dengan bilangan pengali (*multiplier*) yang terlebih dahulu dibuat *recoded digit* nya, dimana mode operasi yang dijalankan sesuai dengan keadaan yang ada pada tabel modifikasi algoritma Booth berikut :

Tabel 2.1 *Recoded Booth Algorithm*

Multiplier			*Recoded digit*	Operasi
i+1	i	i-1		
0	0	0	0	Masukkan 0 (nol) ke partial product
0	0	1	+1	Masukkan A ke partial product
0	1	0	+1	Masukkan A ke partial product
0	1	1	+2	Geser ke kiri A satu posisi dan kemudian masukkan ke partial product
1	0	0	-2	Geser ke kiri 2's complement A satu posisi dan kemudian masukkan ke partial product
1	0	1	-1	Masukkan 2's complement A ke partial product
1	1	0	-1	Masukkan 2's complement A ke partial product
1	1	1	0	Masukkan 0 (nol) ke partial product

Untuk memudahkan dalam operasi algoritma Booth, maka perlu dibuat tabel yang berisikan setiap langkah operasi algoritma Booth sebanyak *Booth code* yang muncul. Setelah tabel lengkap, maka hasil susunan tabel dijumlahkan dengan operasi aritmatika biasa.

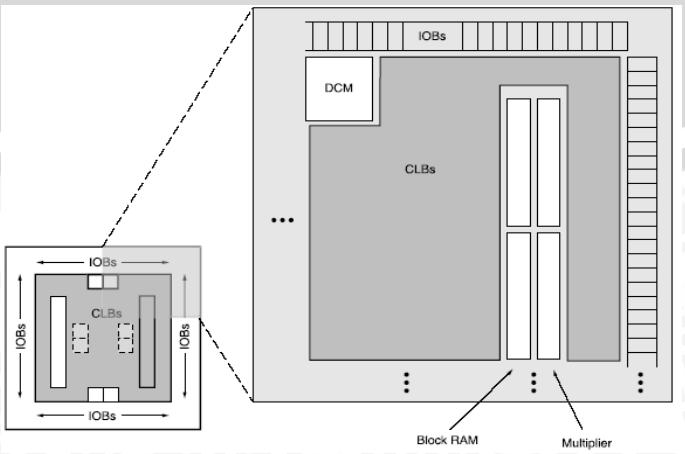
2.3 Field Programmable Gate Arrays (FPGA)

Field Programmable Gate Arrays (FPGA) adalah komponen elektronika dan semikonduktor yang mepunyai komponen gerbang terprogram (*programmable logic*) dan sambungan terpogram. Komponen gerbang terprogram yang dimiliki meliputi jenis gerbang logika biasa (AND, OR, XOR, NOT) maupun jenis fungsi matematis dan kombinatorik yang lebih kompleks (*decoder. adder, subtractor, multiplier, dll*). Blok-blok komponen di dalam FPGA bisa juga mengandung elemen memori (*register*) mulai dari flip-flop sampai pada RAM (*Random Acces Memory*).

Pengertian terprogram (*programmable*) dalam FPGA adalah mirip dengan interkoneksi saklar dalam *breadboard* yang bisa diubah oleh pembuat desain. Dalam FPGA, interkoneksi ini bisa diprogram kembali oleh pengguna maupun pendesain di dalam lab atau lapangan (*field*). Oleh karena itu jajaran gerbang logika (*Gate Array*) ini disebut *field-programmable.* Jenis gerbang logika yang bisa deprogram meliputi semua gerbang dasar untuk memenuhi kebutuhan manapun. Perangkat yang digunakan untuk memprogram FPGA berupa diagram *logic circuit* atau *source code* bahasa HDL (*Hardware Description Languange*). Bahasa HDL yang popular adalah VHDL (*Very high speed integrated circuit Hardware Description Languange*) dan Verilog.

FPGA pertama kali dirintis pada tahun 1984 oleh Ross Freeman (perusahaan Xilinx). FPGA pada waktu itu merupakan pengembangan dari teknologi CPLD (*Complex Programmable Logic Devices*) yang sudah ada sebelumnya. FPGA dan CPLD memiliki elemen *programmable logic* dalam jumlah yang besar. FPGA mempunyai jumlah gerbang logika dari puluhan ribu hingga jutaaan gerbang.

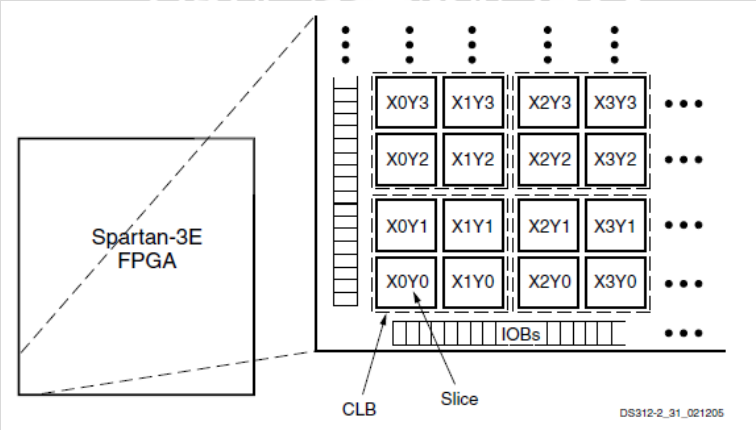
2.3.1 Arsitektur FPGA



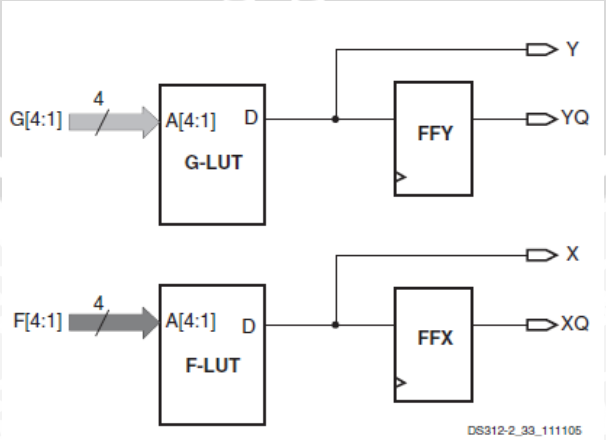
Gambar 2.2 Arsitektur FPGA Xilinx Spartan 3E

Arsitektur FPGA Xilinx Spartan 3E seperti yang terlihat pada gambar 2.2 terdiri dari 5 bagian fungsional, yaitu :

1. *Configurable Logic Block* (CLB), setiap CLB mempunyai 4 *slice*, masing-masing *slice* memiliki 2 *Look-Up Tables* (LUT). LUT mengimplementasikan fungsi logika termasuk elemen penyimpanan (*flip-flop* atau *latch*). LUT dapat digunakan sebagai memori 16 x 1 atau sebagai *shift register* 16 bit, dan unit pengali (*multiplier*) tambahan serta fungsi aritmatika. Logika-logika pada desain secara otomatis dipetakan ke dalam *slice* pada CLB-CLB. Gambar 2.3 dan Gambar 2.4 memperlihatkan lokasi CLB dan LUT.
2. *Input/Output Block* (IOB), mengatur aliran data antara pin *input/output* dan logika internal rangkaian.
3. *Block RAM*, menyediakan penyimpanan data dalam bentuk blok.
4. *Multiplier Block*,sebagai blok pengali dengan 2 buah *input* biner bertanda 18 bit.
5. *Digital Clock Manager* (DCM) *Block*, mendistribusikan, menunda, menjamak, membagi, dan menggeser fase sinyal *clock*.



Gambar 2.3 Lokasi CLB



Gambar 2.4 LUT di dalam CLB

2.3.2 Keping FPGA Xilinx Spartan-3E

Ada 5 jenis dari keluarga Xilinx Spartan 3E. Perbedaan dari kelima jenis tersebut dapat dilihat pada tabel 2.1.

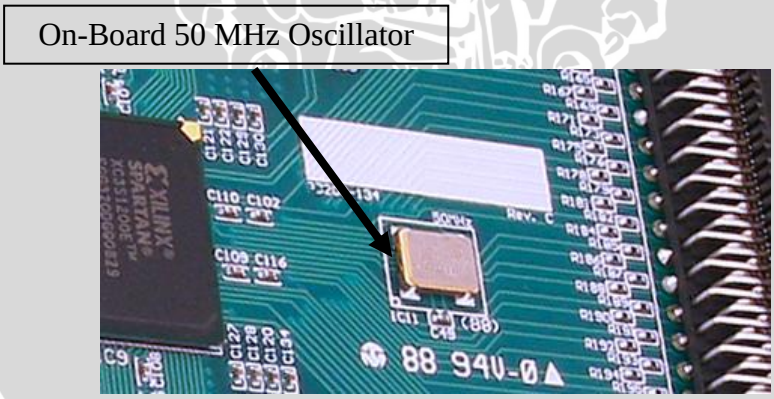
Tabel 2.2 Jenis FPGA Xilinx Spartan 3E

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40
XC3S250E	250K	5,508	34	26	612	2,448	38K	216K	12	4	172	68
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92
XC3S1200E	1200K	19,512	60	46	2,168	8,672	136K	504K	28	8	304	124
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

Dalam perancangan ini akan digunakan keeping XC3S500E Xilinx Spartan 3E. Bagian-bagian penting yang akan terlibat dalam pembuatan desain adalah *clock*, LED, *expansion connector* dan saklar geser.

2.3.2.1 Sumber Clock

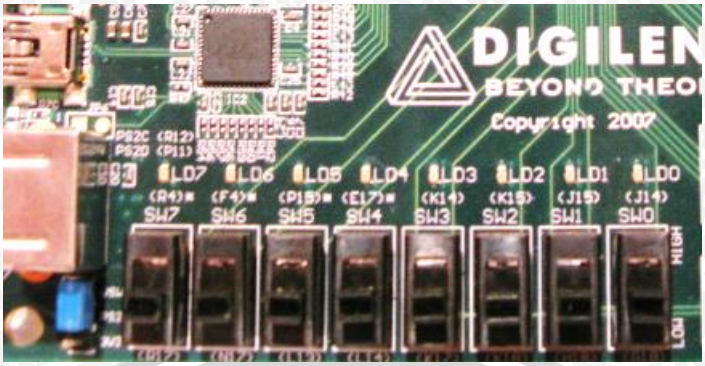
FPGA Spartan 3E mempunyai sumber *clock* internal seperti yang terlihat pada gambar 2.5 yang dibangkitkan dari *oscillator*. Frekuensi *clock* adalah 50 MHz. Sumber *clock* ini terletak pada pin B8.



Gambar 2.5 *Clock* Internal FPGA

2.3.2.2 Saklar Geser

FPGA Spartan 3E mempunyai 8 buah saklar geser. Dalam keeping FPGA terletak disebelah kiri bawah. Posisi pin SW0 = G18, SW1 = H18, SW2 = K18, SW3 = K17, SW4 = L14, SW5 = L13, SW6 = N17, SW7 = R17. Saklar geser dapat dilihat pada gambar 2.6



Gambar 2.6 Saklar geser FPGA

Ketika di geser ke atas (*up*) atau posisi ON, *switch* menghubungkan pin 3,3V FPGA, logika tinggi. Ketika digeser kebawah (*down*) atau posisi OFF, *switch* menghubungkan ke *ground* pin FPGA, logika rendah.

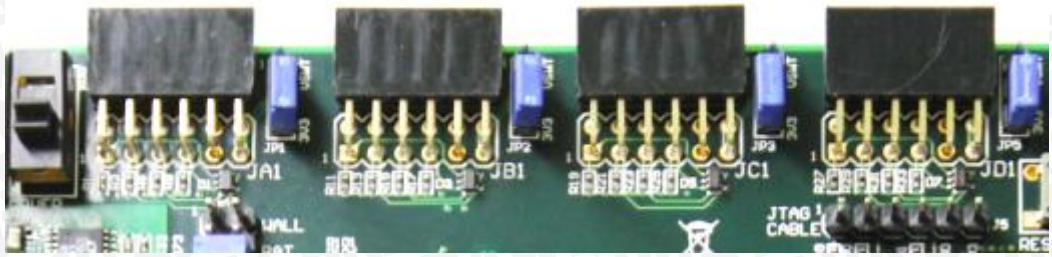
2.3.2.3 LED

FPGA Spartan 3E mempunyai 8 buah indikator LED terletak di sebelah atas saklar geser. Posisi pin dari LED0 hingga LED7 adalah J14, J15, K15, K14, E17, P15, F4 dan R4. Lokasi LED dapat dilihat pada gambar 2.7.

Gambar 2.7 LED FPGA

2.3.2.4 Expansion Connectors

Keping XC3S500E memiliki empat buah 12 pin bebas yang dapat diakses untuk kebutuhan *input* atau *output*, seperti yang terlihat pada gambar 2.8



Gambar 2.8 *Peripheral Connectors* FPGA

Alamat empat buah 12 pin bebas tersebut adalah sebagai berikut :

1. Pmod JA memiliki alamat Pin L15, K12, L17, M15, K13, L16, M14, M16 serta pin untuk Vcc dan Ground.
2. Pmod JB memiliki alamat Pin M13, R18, R15, T17, P17, R16, T18, U18 serta pin untuk Vcc dan Ground.
3. Pmod JC memiliki alamat Pin G15, J16, G13, H16, H15, F14, G16, J12 serta pin untuk Vcc dan Ground.
4. Pmod JD memiliki alamat Pin J13, M18, N18, P18, K14, K15, J15, J14 serta pin untuk Vcc dan Ground.

2.4 VHDL

VHDL merupakan sebuah bahasa pemrograman yang digunakan untuk mendeskripsikan *hardware*. VHDL merupakan singkatan dari VHSIC (*Very High Speed Integrated Circuit*) *Hardware Description Languange*. Pada awalnya dikembangkan pada awal tahun 1980 oleh Departemen Pertahanan Amerika melalui kontrak kerja dengan 3 perusahaan yaitu IBM, Texas Instruments dan Intermetrics. Ketiga perusahaan tersebut berhasil menyelesaikan metode bahasa untuk mendeskripsikan *hardware* (VHDL). Versi pertamanya VHDL yang dipublikasikan adalah versi 7.2, dirilis tahun 1985. Pada tahun 1986 *Institute of Electrical and electronics Engineers* (IEEE) menstandarisasi bahasa VHDL menjadi standar IEEE 1076-1987. Kemudian diperbaharui lagi menjadi IEEE 1076-1993 dan dirilis tahun 1994. Kemudian sebuah standar tambahan, IEEE 1164 untuk mengenalkan nilai *system logic*. Dan sampai saat ini banyak standar bahasa yang dikembangakan.

Bahasa VHDL memiliki 3 struktur bagian utama yaitu *Library*, *Entity*, dan *Architecture.*

2.4.1 Library

Bagian ini mendeklarasikan daftar *library* yang akan digunakan. Ada 3 *library* yang sering digunakan dalam merancang program, yaitu :

- `ieee.std_logic_1164` (dari *ieee library*)
- `standart` (dari *std library*)
- `work` (dari *work library*)

Sintaks :

LIBRARY nama-library;

USE nama-library.nama-package.bagian-package;

2.4.2 Entity

Bagian ini menjelaskan *port-port interface* dalam sistem. Yang dimaksud *port* adalah unit yang akan dihubungkan dengan model program VHDL lainnya atau unit yang menjadi masukan dan keluaran sistem.

Sintaks :

ENTITY nama-entity IS

PORT (daftar-port-dan-tipe);

END nama-entity;

Contoh dari komponen Multiplekser 4 to 1 :

```
entity mux4 is
        port ( masuk_1 : in std_logic_vector (11 downto 0) ;
               masuk_2 : in std_logic_vector (11 downto 0) ;
               masuk_3 : in std_logic_vector (11 downto 0) ;
               masuk_4 : in std_logic_vector (11 downto 0) ;
               sel : in std_logic_vector (1 downto 0) ;
               keluar : out_std_logic_vector (11 downto 0)) ;
        end mux4 ;
```

2.4.3 Architecture

Bagian ini menjelaskan komponen lain yang dilibatkan dalam ekskusi program. Selain itu, yang paling utama adalah menjelaskan fungsi dan perintah-perintah yang diberikan, baik perintah sekuensial maupun kombinasional.

Dalam penyajian bagian ini, ada beberapa tipe yaitu : tipe *structural*, tipe *dataflow*, tipe *behavioural*, dan tipe campuran.

Sintaks :

architecture behaviour of nama-entity is

…… deklarasi architecture

begin

…… isi dari architecture

end behaviour ;

a. Tipe Struktural

Arsitektur berupa gabungan komponen-komponen yang saling berhubungan. Tipe ini dapat menghemat jumlah sintaks jika program sangat kompleks dan mempermudah pemeriksaan kesalahan. Contohnya :

```
architecture STRUCTURE of HALF-ADDER is
component XOR2
port (X, Y: in BIT ; N : out BIT )
end component;
component AND2
port ( L,M : in BIT ; N : out BIT );
end component;
begin
X1: XOR2 port map (A,B,SUM);
A1: AND2 port map (A,B, CARRY);
end STRUCTURE ;
```

Nama arsitekturnya STRUCTURE, kemudian bagian arsitekturnya terbagi menjadi bagian deklarasi (sebelum kata "`begin`") dan bagian perintah (setelah kata "`begin`"). Dua komponen unit lain dinyatakan pada bagian deklarasi. Lalu komponen-komponen yang sudah dideklarasikan tersebut dipanggil dan dilibatkan dalam bagian perintah dengan perintah pemanggilan komponen. Yaitu "`X1 : XOR2 port map (A, B, SUM)`" dan "`A1 : AND2 port map (A, B, CARRY)`". X1 dan A1 adalah nama perintah pemanggilan komponen. Pada perintah X1, sinyal A dan B terhubung ke port X dan Y dar komponen XOR2, sedangkan port N terhubung dengan port *output* dari HALF-ADDER. Demikian juga untuk perintah A1.

b. Tipe *Dataflow*

Aliran data dalam *entity* digambarkan berupa pemberian perintah disertai pewaktuan yang berkelanjutan. Arsitektur tipe ini identic dengan perintah yang disertai batas waktu. Struktur *entity* tidak dijelaskan secara eksplisit seperti tipe struktur. Contohnya :

```
architecture DATAFLOW of HALF-ADDER is
begin
SUM <= A xor B after 8ns;
CARRY <= A and B after 4ns;
end DATAFLOW;
```

Arsitektur DATAFLOW memiliki 2 perintah sinyal yang bersamaan. Simbol "<=" berarti memberikan nilai hasil proses logika di sisi kanan ke sinyal target di sisi kiri. Keterangan "`after`" berarti menunda proses. Sehingga perintah "`A xor B`" baru akan dieksekusi setelah 8 nano sekon. Perintah "`A and B`" setelah 4 nano sekon.

c. Tipe *Behavioral*

Pada tipe ini, perintah-perintah dieksekusi secara sekuensial (berurutan). Struktur *entity* juga tidak dijelaskan secara eksplisit. Contohnya :

```
architecture BEHAVIORAL of HALF-ADDER is
begin
process ( A,B )
variable X, Y : BIT ;
begin
X := A ;
Y := B ;
SUM <= X xor Y;
CARRY <= X and Y;
end process;
end BEHAVIORAL;
```

Perintah proses memiliki bagian deklarasi (sebelum kata "`begin`") dan bagian perintah (antara kata "`begin`" dan "`end process`"). Perintah-perintah yang ada di dalam cakupan perintah proses akan diekskusi secara sekuensial. Sedangkan perintah "`process (A, B)`" berarti nilai A dan B menjadi nilai sensitif.

Sintaks "variable X, Y: BIT; " mendeklarasikan dua variabel X dan Y. Variabel berbeda dengan sinyal, saatt diberi nilai saat itu juga berubah, sedangkan sinyal ada waktu tundanya. Operator pemberian nilai pada variabel berupa tanda ":=", sedangkan pada sinyal berupa tanda "<=". Variabel yang dideklarasikan juga hanya berlaku di dalam perintah proses, sementara sinyal yang dideklarasikan berlaku lebih luas.

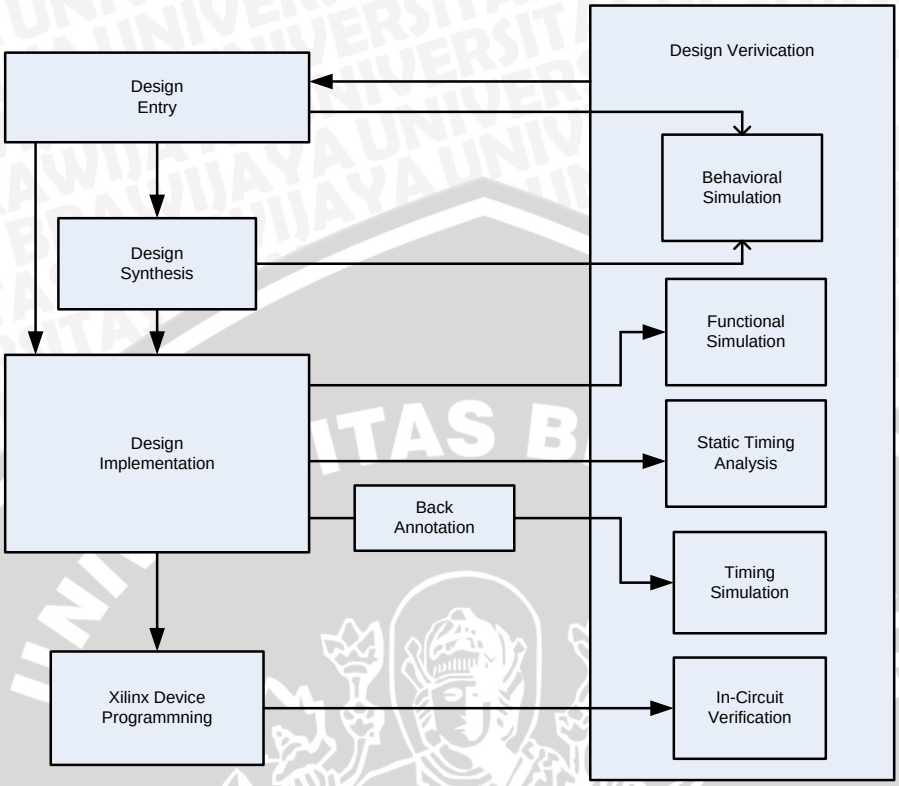
d. Tipe Campuran.

Tipe campuran berarti mengkombinasikan tipe-tipe arsitektur yang ada. Untuk program yang kompleks, seperti pada IDCT peran tipe campuran sangat diperlukan.

2.5 Pembuatan Desain dengan Software Xilinx ISE

Tahapan dalam pembuatan program sampai diimplimentasikan ke dalam FPGA antara lain pembuatan desain sistem (*design entry*), *synthesis*, *implementation*, dan *generate programming file* seperti yang terlihat pada gambar 2.9 sementara ada tahapan

lain yaitu verifikasi desain yang berada pada sisi lain dari urutan desain.



Gambar 2.9 Tahapan pembuatan desain dengan software Xilinx ISE

2.5.1 Desain Sistem (*Design Entry*)

Desain sistem (*Design Entry*) dilakukan dengan membuat *source file* bedasarkan tujuan sistem yang diinginkan. Bisa menggunakan bahasa pemrograman VHDL atau Verilog maupun dengan bahasa *Schematic*. Setelah itu, ditentukan batasan waktu, penempatan pin dan batasan area.

2.5.2 Synthesis

Pada tahap ini, *source file* yang ada diubah formatnya menjadi file *netlist*. *Synthesis* akan mengoptimalkan penggunaan logika berbasis area dan kecepatan atau kombonasi keduanya.

2.5.3 Implementation

File *netlist* yang sudah ada kemudian dikonversi lagi menjadi desain logika dalam format yang dapat di-*download* ke *device* FPGA. Pada unit *implementation*,

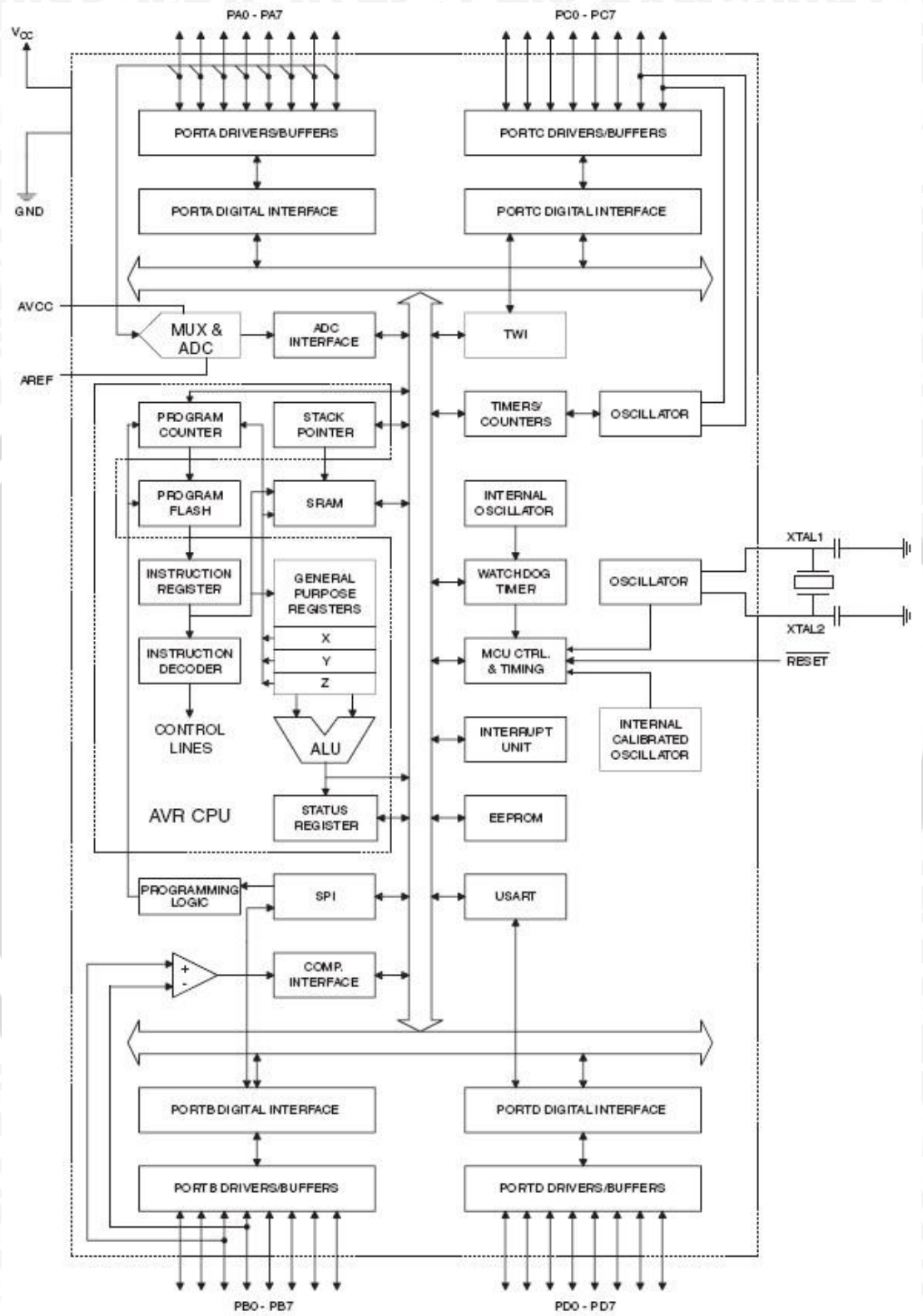
beberapa proses dilakukan. Antara lain :

a. *Translate*, menggabungkan sema *input netlist* dan desain batasan (*constraint*) lalu menghasilkan *output* file *Native Generic Database* (NGD). File NGD adalah hasil penyesuaian desain dengan batasan.
b. *Map*, memetakan logika yang diarahkan NGD ke elemen-elemen FPGA, seperti CLB dan IOB. Hasilnya berupa file *Native Circuit Description* (NCD).
c. *Place and Route*, algoritma penetuan lokasi (*placement*) digunakan untuk menempatkan masing-masing blok *array* FPGA. Hal ini bertujuan untuk meminimalkan total panjang interkoneksi yang dibutuhkan pada saat penempatan komponen. Proses *route* akan mengatur perkawatan FPGA dan menentukan saklar-saklar penghubung terkonfigurasi untuk menghubungkan antar blok-blok FPGA, memastikan 100% koneksi telah terbentuk, meminimalkan waktu tunda pada koneksi waktu kritis (*time-critical connections*). Pada tahap ini akan mengambil file NCD lalu menentukan lokasi dan rute untuk desain logika pada *device* FPGA.

2.5.4 Generate Programming File

Tahap ini akan mengkonfigurasi *chip* stelah langkah *placement* dan *routing* tuntas, keseluruhan proses memakan waktu beberapa menit. Pada tahap ini, dibuat file pemrograman (BIT) untuk memprogram FPGA. File BIT kemudian di-*download* ke FPGA dalam bentuk file PROM, ACE atau JTAG sesuai media pen-*download*-an.

2.6 ATmega8535



Gambar 2.10 Blok diagram fungsional ATmega8535

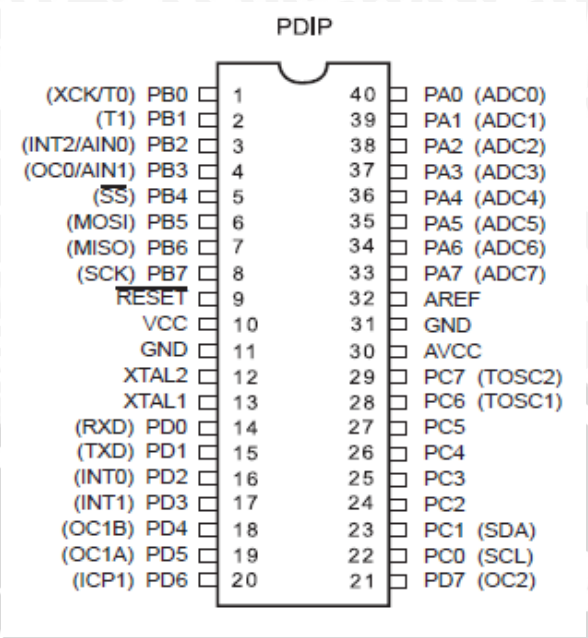
Dari gambar 2.10 dapat dilihat bahwa ATmega8535 memiliki bagian sebagai berikut :

1. Saluran I/O sebanyak 32 buah yaitu port A, port B, port C dan port D.
2. ADC 10 bit sebanyak 8 saluran.
3. Tiga buah *Timer/Counter* dengan kemampuan pembanding.
4. CPU yang terdiri atas 32 register.
5. *Watchdog Timer* dengan osilator internal.
6. SRAM sebesar 512 *byte.*
7. Memori *Flash* sebesar 8 kB dengan kemampuan *Read While Wrtie.*
8. Unit interupsi internal dan eksternal.
9. Port antarmuka SPL.
10. EEPROM sebesar 512 *byte* yang dapat diprogram saat operasi.
11. Antarmuka komparator analog.
12. Port USART untuk komunikasi serial.

2.6.1 Konfigurasi Pin ATmega8535

Konfigurasi pin ATmega8535 dilihat pada gambar 2.11. Dari gambar tersebut maka dapat dijelaskan secara fungsional konfigurasi pin ATmega8535 sebagai berikut :

1. VCC merupakan pin yang berfungsi untuk pin masukan catu daya.
2. GND merupakan pin *ground.*
3. Port A (PA0…PA7) merupakan pin I/O dua arah dan pin masukan ADC.
4. Port B (PB0…PB7) merupakan pin I/O dua arah dan pin fungsi khusus yaitu *Timer/Counter,* komparator analog dan SPI.
5. Port C (PC0…PC7) merupakan pin I/O dua arah dan pin fungsi khusus yaitu TW1, komparator analog dan *Timer Oscilator*.
6. Port D (PD0…PD7) merupakan pin I/O dua arah dan pin fungsi khusus yaitu komparator analog, interupsi eksternal dan komunikasi serial.
7. RESET merupakan pin yang digunakan untuk mereset mikrokontroler.
8. XTAL 1 dan XTAL 2 merupakan pin masukan clock eksternal.
9. AVCC merupakan pin masukan tegangan untuk ADC.
10. AREF merupakan pin masukan tegangan referensi ADC.



Gambar 2.11 Pin ATmega8535

2.6.2 Interupsi ATmega8535

AVR ATmega8535 memiliki 3 pin untuk interupsi eksternal yaitu INT0, INT1 dan INT2. Interupsi eksternal dapat dibangkitkan apabila terdapat perubahan logika atau logika 0 pada pin interupsi. Pengaturan kondisi keadaan yang menyebabkan terjadinya interupsi eksternal diatur oleh register MCUCR (MCU *Control Register*) yang terlihat seperti gambar 2.12.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar 2.12 Register MCUCR

Bit penyusunya dapat dijelaskan sebagai berikut :

a. Bit ISC11 dan ISC10 bersama-sama menentukan kondisi yang dapat menyebabkan interupsi eksternal pada pin INT1. Untuk keadaan lengkapnya dapa dilihat pada tabel 2.2

Tabel 2.3 Setting kondisi interupsi eksternal 1

ISC11	ISC10	Deskripsi
0	0	Logika 0 pada pin INT1 menyebabkan interupsi.
0	1	Perubahan logika pada pin INT1 menyebabkan interupsi
1	0	Perubahan kondisi 1 ke 0 pada pin INT1 menyebabkan interupsi
1	1	Perubahan kondisi 0 ke 1 pada pin INT1 menyebabkan interupsi

b. Bit ISC01 dan ISC00 bersama-sama menetukan kondisi yang dapat menyebabkan interupsi eksternal pada pin INT0. Untuk keadaan lengkapnya dapat dilihat pada tabel 2.3

Tabel 2.4 Setting kondisi interupsi eksternal 0

ISC01	ISC00	Deskripsi
0	0	Logika 0 pada pin INT0 menyebabkan interupsi.
0	1	Perubahan logika pada pin INT0 menyebabkan interupsi
1	0	Perubahan kondisi 1 ke 0 pada pin INT0 menyebabkan interupsi
1	1	Perubahan kondisi 0 ke 1 pada pin INT0 menyebabkan interupsi

Pemilihan pengaktifan interupsi eksternal diatur oleh register GICR (*General Interrupt Control Register*) yang terlihat pada gambar 2.13

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar 2.13 *General Control Interrupt Register*

a. Bit INT1 adalah bit untuk mengaktifkan interupsi eksternal 1. Apabila bit ini diberi logika 1 dan bi-I pada SREG (status register) juga 1 maka interupsi eksternal 1 akan aktif.
b. Bit INT0 adalah bit untuk mengaktifkan interupsi eksternal 0. Apabila bit ini diberi logika 1 dan bit-I pada SREG (status register) juga 1 maka interupsi

eksternal 0 akan aktif.

c. Bit INT2 adalah bit untuk mengaktifkan interupsi eksternal 2. Apabila bit ini diberi logika 1 dan bit-I pada SREG (status register) juga 1 maka interupsi eksternal 2 akan aktif.

BAB III

METODOLOGI

Dalam penyusunan tugas akhir ini, dirancang suatu perancangan implementasi IDCT pada FPGA. Metode penelitian yang digunakan pada penyusunan skripsi ini adalah:

3.1 Studi Literatur

Studi literatur menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut meliputi:

1. Mempelajari buku – buku yang berhubungan dengan IDCT dan FPGA.
2. Mempelajari algoritma Loeffler.
3. Mempelajari teknik – teknik dasar pemrograman dengan menggunakan VHDL.

3.2 Bahan Penelitian

a. FPGA Spartan 3E XC3S500E

   Keping XC3S500E mempunyai 500.000 gerbang dengan jumlah CLB sebesar 1164, IOB sebesar 323, pengali sebanyak 20 unit serta interkoneksi yang dapat deprogram.

b. Mikrokontroller ATmega 8535

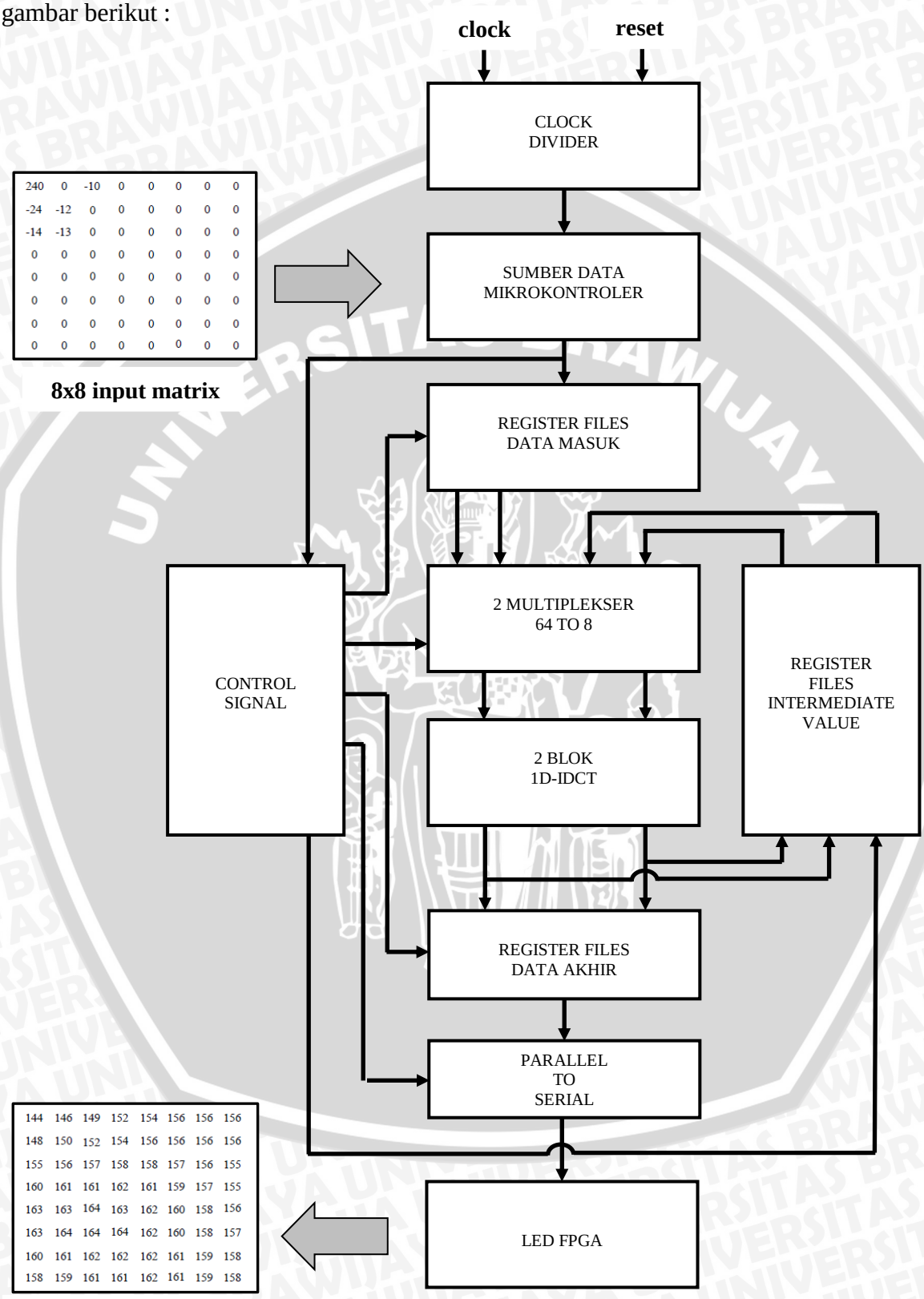
   ATmega 8535 mempunyai saluran I/O sebanyak 32 buah, ADC 10 bit sebanyak 8 saluran, 3 buah *Timer/Counter*, CPU yang terdiri atas 32 register, SRAM sebesar 512 *byte*, memori *Flash* sebesar 8 kB, serta EEPROM sebesar 512 *byte.*

3.3 Alat Penelitian

a. Satu set komputer Pentium 4
b. Sitem Operasi Windows XP Professional
c. Xilinx ISE Webpack 14.3
d. Digilent Adept
e. MATLAB 7.0.4
f. Code Vision AVR 2.5.0

3.4 Konfigurasi Sistem

Sistem 2-D IDCT ini secara umum dapat digambarkan dengan model seperti gambar berikut :



Gambar 3.1. Blok diagram sistem

Keterangan :

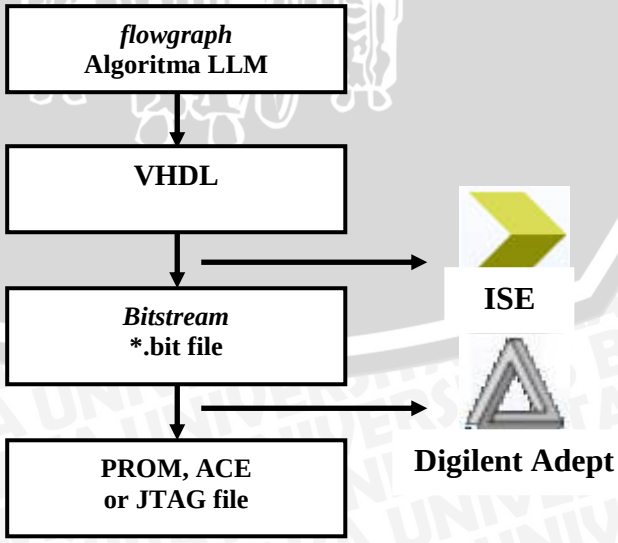
1. 8 x 8 *input* matrix : Matriks 8 x 8 dengan nilai elemen matriks berupa bilangan bulat riil pada kawasan frekuensi.
2. Mikrokontroler : Mikrontroler digunakan sebagai bagian yang akan memasukan data 4 bit menuju FPGA.
3. *Control Signal* : *Control Signal* berfungsi untuk menghasilkan sinyal-sinyal yang digunakan untuk pengendalian proses 2D-IDCT. Unit pengontrol sinyal akan menghasilkan pencacah internal untuk menghasilkan sinyal pengendali proses.
4. *Register* Files Data Masuk : *Register* ini tersusun atas 64 *register* dengan lebar 12 bit sebagai tempat untuk menyimpan 64x12 bit data masukan yang dikirimkan oleh mikrokontroler. *Register* ini bekerja sebagai *register* geser, sehingga data yang masuk pertama nantinya akan digeser sampai *register* terakhir (*register* ke-64).
5. Multiplekser : Unit multiplekser merupakan unit pemilih data, multiplekser yang digunakan adalah multiplkeser 64 ke 8, yang berarti multiplekser mempunyai 64 data masukan dan 8 data keluaran yang dikendalikan oleh sinyal yang dihasilkan unit pengontrol sinyal sebagai pemilih data mana yang akan dikeluarkan.
6. 2 Blok 1D-IDCT : Unit 1D-IDCT merupakan unit aritmatika pada sistem 2D-IDCT. Dalam sistem 2D-IDCT dibutuhkan dua unit 1D-IDCT. Proses komputasi pada masing-masing unit dikendalikan oleh sinyal yang dihasilkan oleh unit control sinyal serta sinyal *clock*.
7. *Register* Intermediate Value: *Register* ini tersusun atas 64 *register* dengan lebar 12 bit sebagai unit penyimpanan data hasil proses komputasi 1D-IDCT pada bagian baris 8x8 data

masukan. Data yang tersimpan dalam *register* ini akan diproses lagi oleh unit 1D-IDCT pada setiap kolomnya untuk menghasilkan hasil akhir proses 2D-IDCT.

8. *Register Files* Data Akhir : *Register* ini tersusun atas 64 *register* dengan lebar 8 bit sebagai unit penyimpanan nilai akhir dari proses 2D-IDCT sebelum 8x8 data dikeluarkan lewat unit *parallel to serial*.
9. *Parallel to Serial* : Unit ini merupakan unit terakhir dari sistem 2D-IDCT yang berfungsi untuk mengeluarkan 8x8 data (64 data) yang telah disimpan dalam *register files* nilai akhir satu persatu dan kemudian ditampilkan dalam data biner 8 bit pada LED FPGA.
10. LED FPGA : LED (*Light Emiting Diode*) pada FPGA digunakan sebagai bagian yang akan menampilkan hasil komputasi 2D-IDCT dari FPGA.
11. 8 x 8 *output* matrix : Matriks 8 x 8 dengan nilai elemen matriks berupa bilangan bulat riil pada kawasan spasial.

3.5 Jalan Penelitian

Jalan penelitian ini dapat digambarkan melalui gambar diagram blok 3.2 :

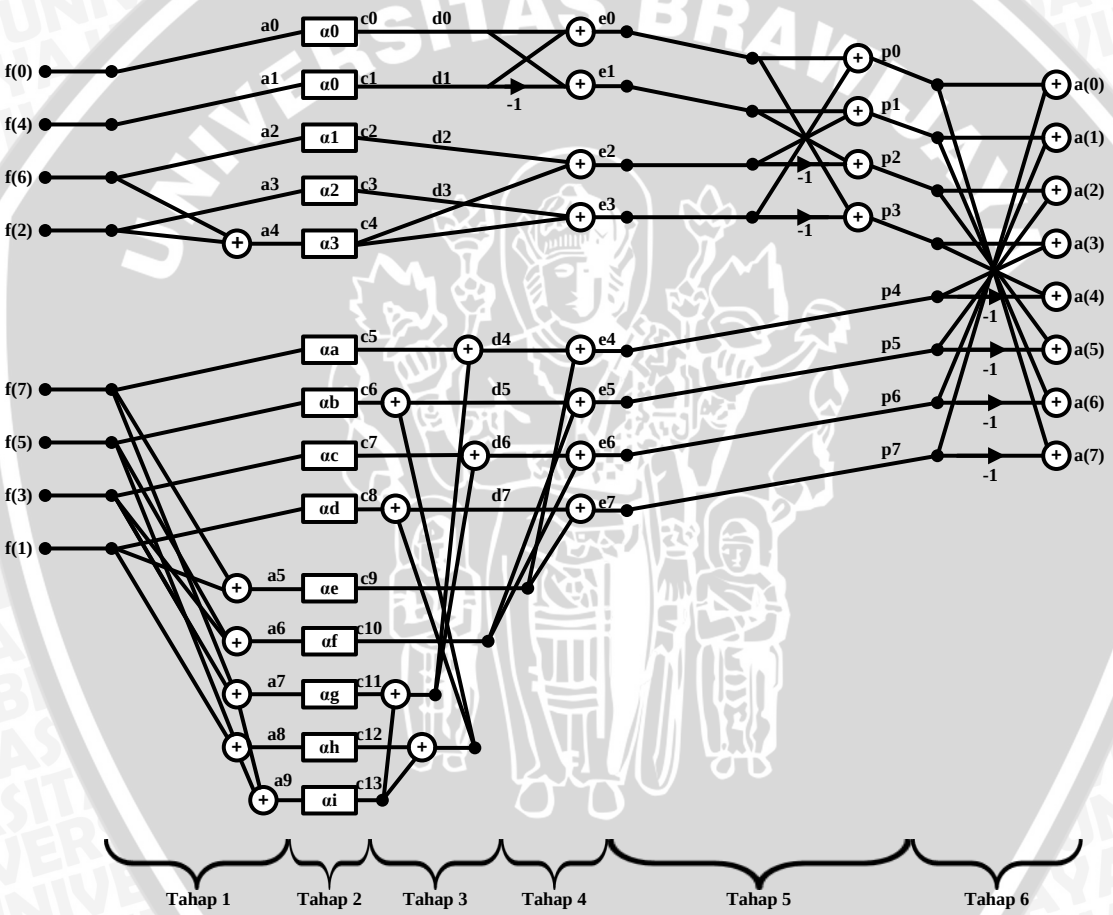


Gambar 3.2. Blok diagram implementasi algoritma

Keterangan :

1. *Signal Flow Graph* Algoritma LLM

Algoritma LLM didasarkan pada proses pemisahan 8 data masukan menjadi bagian ganjil (*odd*) dan bagian yang genap (*even*), kemudian dilakukan dua proses komputasi 4 titik IDCT lalu menggabungkan kedua proses tersebut dengan *butterfly stage*. Pelaksanaan penelitian ini dimulai dengan membagi *flow graph* algoritma LLM kedalam beberapa tahap komputasi. Terdapat 6 tahapan yang dilakukan secara berurutan yang dapat dilihat pada gambar 3.3.



Gambar 3.3. 1D-IDCT *flowgraph* algoritma LLM

Keterangan :

a. *Addition* (penjumlahan) ⊕

Seperti penjumlahan pada umumnya, penjumlahan disini menjumlahkan antara dua buah nilai *input* dan menghasilkan satu buah nilai *output*. Operasi penjumlahan dilakukan dengan memakai operator "+" yang sudah tersedia dalam *library arithmetic* pada software Xilinx ISE.

b. *Multiplication* (perkalian) 

Perkalian yaitu mengalikan sebuah nilai *input* dengan sebuah konstanta yaitu koefisien pengali algoritma LLM yang kemudian menghasilkan sebuah nilai *output*. Pada proses perkalian dapat dilakukan dengan dua macam cara, pertama menggunakan *embedded multiplier 18x18 bit* yang talah tersedia di Spartan 3E, dan yang kedua menggunkan desain pengali yang telah dirancang.

- *Embedded Multiplier*

Spartan-3E menyediakan *Dedicated Multiplier / Embedded Multiplier* sebanyak 20 unit, dalam pemanfaatany *embedded multiplier* cukup ditulis dengan bahasa VHDL sebagai berikut :

```
MULT18X18_inst : MULT18X18
Port map (
     P => P,   -- 36-bit multiplier output
     A => A,   -- 18-bit multiplier input
     B => B,   -- 18-bit multiplier input
);
```

- *Design Multiplier*

Design Multiplier adalah pengali yang dirancang sendiri menggunakan algoritma Booth. Algortima Booth digunakan karena dianggap paling efisien dalam hal komputasi bilangan biner bertanda (*signed binary*), terutama untuk perkalian.

- *Constant Multiplier*

Algoritma LLM mempunyai konstanta pengali yang mempunyai nilai masing-masing adalah α_0=0,70711, α_1=-1,30656, α_2=0,5412, α_3=0,38268, α_a=0,21116, α_b=1,45177, α_c=2,17273, α_d=1,06159, α_e=1,06159, α_f=-0,63638, α_g=-1,38704, α_h=-0,2759, α_i=0,83147 (*Loeffler et al.1989*).

c. Multiplication by -1 (pengurangan) 

Seperti pengurangan pada umumnya, yaitu melakukan operasi pengurangan antara dua buah nilai dua buah *input* dan menghasilkan satu buah nilai *output*. Operasi pengurangan dilakukan dengan memakai operator "-" yang sudah tersedia dalam *library arithmetic* pada software Xilinx ISE.

2. VHDL

VHDL (*Very high speed integrated circuit Hardware Description Language*) digunakan dalam mendesain atau mendeskripsikan fungsi perangkat keras sesuai algoritma yang digunakan agar FPGA berfungsi seperti yang diinginkan. Dalam penelitian ini digunakan FPGA Xilinx Spartan 3E yang mempunyai *compiler* Xilinx ISE. *Compiler* akan memproses VHDL menjadi sebuah bit file yang siap di *download* pada FPGA.

3. *Bitstream* *.bit file

Hasil dari proses *compile* Xilinx ISE adalah bit file yang merupakan file yang berisikan informasi konfigurasi yang akan di *download* kan pada FPGA. Bit file dapat dikirimkan pada FPGA menggunakan *software* khusus diantara nya Digilent Adept dan Xilinx Impact.

4. PROM, ACE atau JTAG

Dalam sebuah sirkuit yang kompleks pada umunya terdiri dari bagian-bagian yang spesifik, JTAG (*Joint Test Action Group*) berfungsi untuk mengatur koneksi pin-pin pada semua bagian-bagian yang ada. JTAG akan memastikan koneksi/jalur yang menghubungkan antara bagian-bagian yang ada telah tersambung dengan baik sesuai yang diinginkan.

3.6 Pengujian Sistem

Pengujian dilakukan untuk mengetahui apakah sistem yang telah dirancang dapat berfungsi sesuai apa yang diharapkan. Untuk mengetahui apakah sistem bekerja dengan baik dan sesuai dengan perancangan, maka diperlukan serangkain pengujian. Pengujian yang dilakukan ditekankan pada dua hal yaitu :

1. Pengujian akurasi sistem.

   Pengujian akurasi sistem dapat dilakukan dengan cara membandingkan dengan hasil 2D-IDCT dengan menggunakan *software* MATLAB atau dengan cara membandingkan dengan matriks asli sebelum dilakukan transformasi DCT.

2. Pengujian kinerja FPGA.

Pengujian kinerja dilakukan dengan cara mengetahui seberapa besar kapasitas yang digunakan dalam implementasi.

3.7 Kesimpulan dan Saran

Kesimpulan diperoleh dari data-data hasil setelah semua analisis dan pengujian dilakukan, sehingga diperoleh saran juga guna perbaikan aplikasi pada penelitian selanjutnya

BAB IV

PERANCANGAN SISTEM

4.1 Algoritma IDCT

Seperti yang telah dijelaskan pada bab sebelumnya, ada dua metode untuk perancangan komputasi 2D-IDCT. Metode pertama berdasar pada persamaan (2.2) yaitu melakukan proses komputasi 2D-IDCT secara langsung. Kelemahan dari metode pertama adalah dalam hal kompleksitas arsitektur tetapi menawarkan *low latency*. Metode kedua adalah dengan melakukan komputasi 2D-IDCT melalui persamaan 1D-IDCT dengan proses melakukan komputasi 1D-IDCT pada baris 8x8 blok data, kemudian menyimpan hasilnya pada memori, lalu kemudian melakukan lagi proses komputasi 1D-IDCT pada kolom 8x8 blok data hasil proses 1D-IDCT baris, sehingga hasil dari keluaran adalah 8x8 blok data 2D-IDCT. Metode kedua membutuhkan tiga tahap proses yaitu komputasi 1D-IDCT baris, menyimpan hasil komputasi dan komputasi 1D-IDCT kolom.

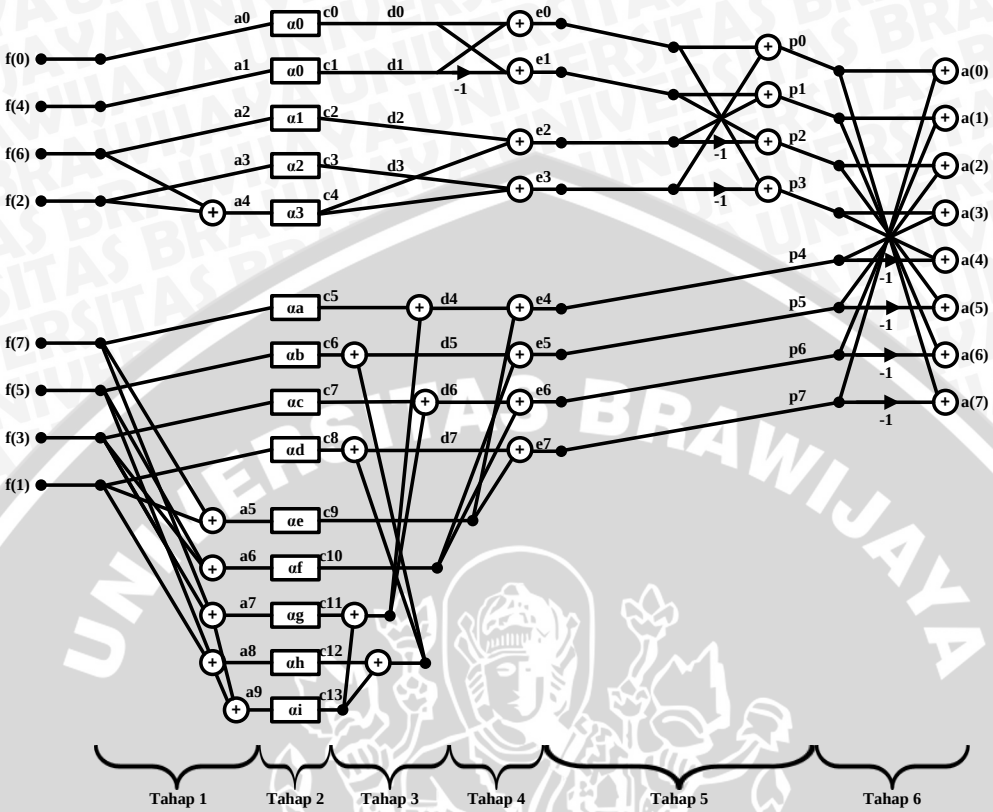
Untuk mempercepat proses komputasi dengan cara mengurangi jumlah proses perkalian, perancangan sistem akan didasarkan pada ***Loeffler-Ligtenberg-Moschytz Algorithm*** **(LLM)** yang diteliti pada tahun 1989. Algoritma ini pada dasarnya merupakan algoritma untuk menghitung komputasi 8 titik 1D-IDCT sesuai persamaan (2.4). Algoritma LLM didasarkan pada proses faktorisasi matriks A pada persamaan (4.1) yang kemudian dapat dijabarkan menjadi persamaan 4.2. Proses ini dilakukan dengan memisahkan 8 data masukan menjadi bagian ganjil (*odd*) dan bagian yang genap (*even*), kemudian dilakukan dua proses komputasi 4 titik IDCT lalu menggabungkan kedua proses tersebut dengan *butterfly stage*.

$$\boldsymbol{A}_{8\,x\,8} = \boldsymbol{B}_{8\,x\,8} \begin{matrix} \boldsymbol{A}_{4\,x\,4} & \boldsymbol{0}_{4\,x\,4} \\ \boldsymbol{0}_{4\,x\,4} & \boldsymbol{R}_{4\,x\,4} \end{matrix} \tag{4.1}$$

$$\boldsymbol{A}_{8\,x\,8} = \boldsymbol{B}_{8\,x\,8} \begin{matrix} \boldsymbol{A}_{4\,x\,4} & \boldsymbol{0}_{4\,x\,4} \\ \boldsymbol{0}_{4\,x\,4} & \boldsymbol{R}_{4\,x\,4} \end{matrix} \boldsymbol{B}_{4\,x\,4} \begin{matrix} \boldsymbol{P}_{4\,x\,4} & \boldsymbol{0}_{4\,x\,4} \\ \boldsymbol{0}_{4\,x\,4} & \boldsymbol{P}_{4\,x\,4} \end{matrix} \tag{4.2}$$

Matriks $\boldsymbol{A}_{8\,x\,8}$ adalah matriks 8 titik IDCT, matriks $\boldsymbol{A}_{4\,x\,4}$ adalah matriks 4 titik IDCT, matriks $\boldsymbol{R}_{4\,x\,4}$ merupakan matriks 4 titik DCT tipe IV dengan pengecualian bahwa kolom dan baris disusun dari komponen frekuensi tinggi ke komponen frekuensi rendah, matriks $\boldsymbol{0}_{4\,x\,4}$ adalah matriks dengan semua komponen bernilai 0, matriks $\boldsymbol{P}_{4\,x\,4}$

adalah matriks *permuter* yang menyusun kolom sehingga cocok untuk dimasukan ke dalam proses 1D-IDCT, dan matriks $\boldsymbol{B}_{4\,x\,4}$ merupakan matriks *butterfly*.



Gambar 4.1. 1D-IDCT *flowgraph* algoritma LLM (*Loeffler et al. 1989*)

Tabel 4.1 Koefisien pengali algoritma LLM (*Loeffler et al. 1989*)

Konstanta Pengali	Amplitude
a0	0,70711
a1	-1,30656
a2	0,5412
a3	0,38268
aa	0,21116
ab	1,45177
ac	2,17273
ad	1,06159
ae	-0,63638
af	-1,81225
ag	-1,38704
ah	-0,2759
ai	0,83147

Ide dasar dari algoritma LLM adalah melakukan proses penyekalaan pada semua proses perkalian dengan $\frac{1}{a0}$, sehingga akan menghilangkan proses perkalian pada

bagian $\boldsymbol{A}_{2\,x\,2}$. Keluaran dari proses penyekalaan adalah *s(x)/ao*, jika dilakukan proses 1D-IDCT maka keluaranya adalah $s(y,x)/a_0^2$. Karena $a0 = \frac{1}{\sqrt{2}}$ maka dihasilkan keluaran dua kali lipat dari hasil sebenarnya, sehingga hasil keluaran harus dibagi dua terlebih dahulu. Cara lain untuk mendapatkan hasil keluaran tanpa melakukan proses pembagian pada keluaran adalah dengan cara melakukan penyekalaan pada seluruh koefisien pengali pada tabel 4.1 dengan faktor $\frac{1}{2}$, sehingga koefisien pengali hasil penyekalaan dapat dilihat pada tabel 4.2.

Berdasarkan pada gambar 4.1 maka komputasi aritmatik dapat di bagi kedalam enam tahap (*step*) menjadi persamaan 4.3 sampai persamaan 4.16.

Tahap 1 :

a0=f(0); a1=f(4); a2=f(6); a3=f(2); a4=f(6)+f(2); (4.3)

a5=f(7)+f(1); a6=f(5)+f(3); a7=f(7)+f(3); a8=f(1)+f(5); a9=a7+a8; (4.4)

Tahap 2 :

c0=α0*a0; c1=α0*a1; c2=α1*a2; c3=α2*a3 (4.5)

c4=α3*a4; c5=αa*f(7); c6=αb*f(5); c7=αc*f(3); (4.6)

c8=αd*f(1); c9=αe*a5; c10=αf*a6; c11=αg*a7; (4.7)

c12=αh*a8; c13=αi*a9; (4.8)

Tahap 3 :

d0=c0; d1=c1; d2=c2; d3=c3; d4=c5+c11+c13; (4.9)

d5=c6+c12+c13; d6=c7+c11+c13; d7=c8+c12+c13; (4.10)

Tahap 4 :

e0=d0+d1; e1=d0+(-d1); e2=d2+c4; e3=d3+c4; (4.11)

e4=d4+c9; e5=d5+c10; e6=d6+c10; e7=d7+c9; (4.12)

Tahap 5 :

p0=e0+e3; p1=e1+e2; p2=e1+(-e2); (4.13)

p3=e0+(-e3); p4=e4; p5=e5; p6=e6; p7=e7; (4.14)

Tahap 6 :

s(0)=p0+p7; s(1)=p1+p6; s(2)=p2+p5; s(3)=p3+p4; (4.15)

s(4)=p3+(-p4); s(5)=p2+(-p5); s(6)=p1+(-p6); s(7)=p0+(-p7); (4.16)

Untuk proses 8 titik 1D-IDCT algoritma LLM hanya membutuhkan 14 kali proses perkalian, 7 proses pengurangan dan 25 proses penjumlahan. Dalam perancangan sistem semua data masukan dan data keluaran akan diproses dalam bentuk bilangan bulat (*integer*), oleh karena itu koefisien pengali akan dibulatkan dengan faktor 2^{10} atau 1024.

Tabel 4.2 Koefisien pengali algoritma LLM hasil penyekalaan

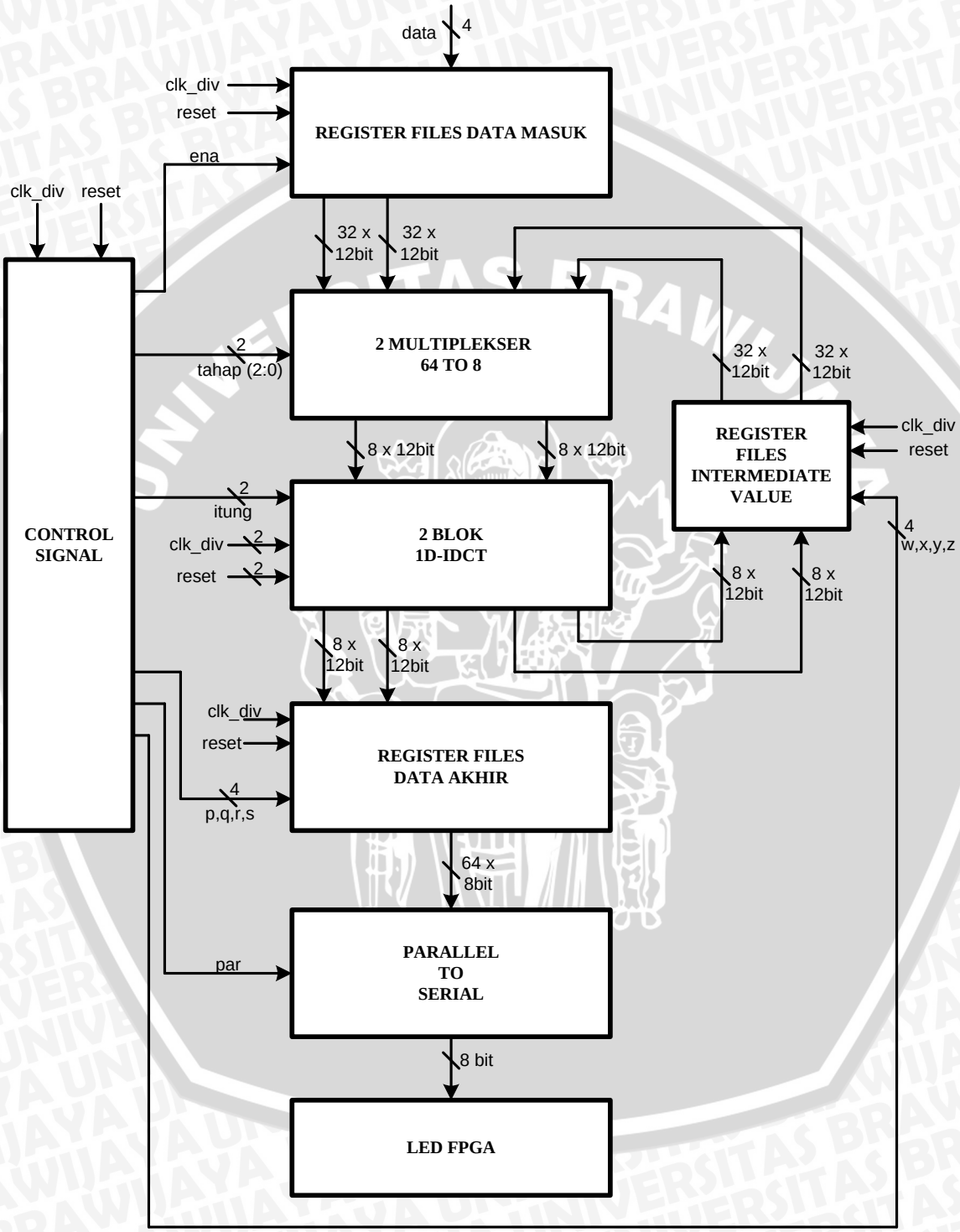
Konstanta Pengali	Amplitude	Hasil penyekalaan
a0	0,70711	0,3535
a1	-1,30656	-0,6532
a2	0,5412	0,2706
a3	0,38268	0,1913
aa	0,21116	0,1055
ab	1,45177	0,7258
ac	2,17273	1,0863
ad	1,06159	0,5308
ae	-0,63638	-0,3181
af	-1,81225	-0,9061
ag	-1,38704	-0,6935
ah	-0,2759	-0,1379
ai	0,83147	0,4157

4.2 Perancangan Implementasi 2D-IDCT

Proses 2D-IDCT dapat dilakukan dengan melakukan proses 1D-IDCT pada bagian baris dari matriks 8x8 data masukan (64 data) kemudian menyimpan hasilnya pada *register,* kemudian dilakukan proses 1D-IDCT pada bagian kolom. Proses 2D-IDCT dirancang menggunakan proses *sequential,* yang berarti proses dilakukan secara bertahap pada tiap modul atau unit yang membentuk sistem 2D-IDCT. Sumber data dikirimkan lewat mikrokontroler setiap 4 bit, pengiriman data tersebut akan dikendalikan oleh *clock* yang dihasilkan FPGA sebagai pemicu fungsi *interrupt* yang tersedia pada mikrokontroler. Mikrokontroler menggunakan fungsi *interupt0* untuk mendeteksi *rising edge* dari *clock*, sehingga ketika terdapat perubahan logika dari ‘0’ menjadi ‘1’ maka *interupt0* akan aktif dan kemudian 4 bit data dikirimkan.

Meskipun kelemahan dari proses *sequential* terletak pada kecepatan proses, terdapat cara untuk mengatasi masalah tersebut yaitu dengan menggunakan lebih dari 1

unit 1D-IDCT untuk proses 2D-IDCT. Untuk mempercepat kecepatan proses komputasi 2D-IDCT dengan mempertimbangkan kapasitas logika pada keping FPGA, maka digunakan 2 unit 1D-IDCT dalam perancangan sistem 2D-IDCT.



Gambar 4.2 Diagram alir implementasi proses 2D-IDCT

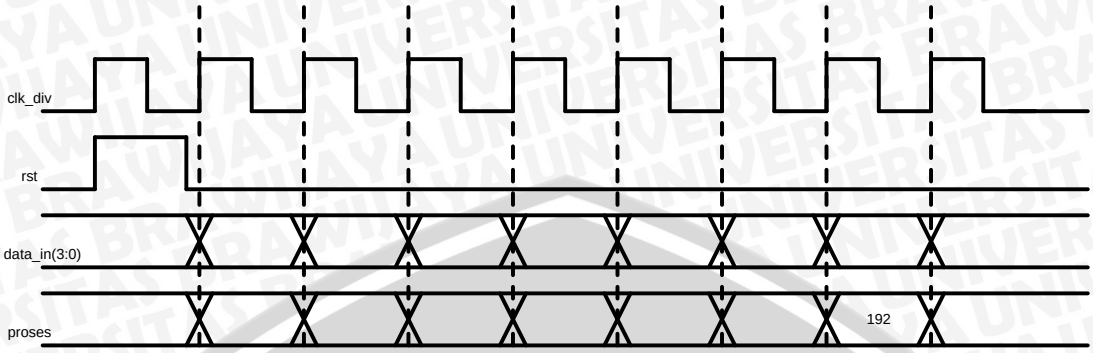
Prosesor 2D-IDCT tersusun atas unit sumber data 4 bit dari mikrokontroler, *control signal*, *register files*, multiplekser dua unit 1D-IDCT, unit *parallel to serial* dan unit penghasil detak (*clock generator*) seperti yang terlihat pada gambar 4.2. Data dikirim oleh mikrokontroler sebesar 4 bit sekali pengiriman yang kemudian ditampung dalam *register files* data masuk dan dibentuk mejadi 64 x 12bit data. Data akan dikelurakan dari *register files* data masuk dan selanjutnya akan dipilih oleh multiplekser data mana yang akan diproses oleh blok IDCT pertama. Setelah proses komputasi IDCT pertama selesai, data dikeluarkan dan ditampung oleh *register files intermediate value* yang selanjutnya akan dipilih kembali oleh multiplekser untuk diolah pada blok IDCT kedua. Hasil komputasi kedua blok IDCT akan ditampung oleh *register files* data akhir yang selanjutnya akan dikirim pada blok *parallel to serial* yang akan mengeluarkan data secara serial yang ditampilkan pada LED FPGA berupa data biner 8bit sesuai urutan indeks matriks.

Tabel 4.3 I/O *Port* sistem 2D-IDCT

Nama I/O	I/O	Fungsi	Alamat Pin
clk	I	Sinyal *clock* FPGA	B8
rst	I	Untuk me-reset sistem	B18
data	I	Data masukan 4 bit dari mikro	L15,K12,L17,M15
clk_div	O	Sinyal *clock* sistem dan berfungsi juga untuk mengaktifkan interupsi mikrokontroler	M13
serial	O	Data keluaran 8 bit hasil akhir 2D-IDCT	J14,J15,K15,K14, E17,P15,F4,R4

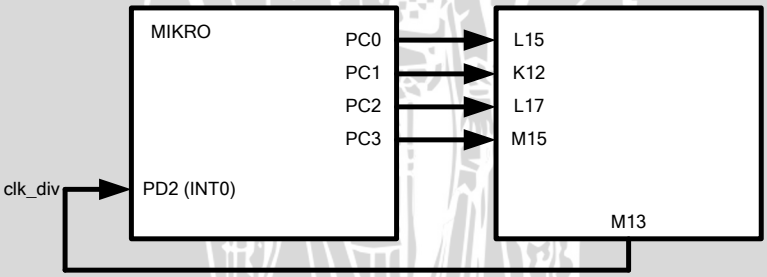
Tabel 4.3 menjelaskan I/O *port* dari perancangan sistem 2D-IDCT, “clk” sebagai *input* yang berfungsi sebagai *clock* FPGA terletak pada pin B8, “rst” sebagai *input* yang berfungsi untuk me-*reset* sistem diatur pada pin B18, “data” sebagai *input* yang berfungsi sebagai jalan masuk 4bit data dari mikrokontroler diatur pada pin L15,K12,L17,M15, “clk_div” sebagai *output* yang berfungsi untuk mengaktifkan interupsi mikrokontroler, “serial” sebagai *output* yang berfungsi sebagai jalan keluar data hasil komputasi diatur pada pin J14,J15,K15,K14,E17,P15,F4,R4.

4.2.1 Sumber Data Mikrokontroler



Gambar 4.3 *Timing diagram* data masuk

Gambar 4.3 menjelaskan *timing diagram* proses pembacaan data 4 bit yang dikirimkan oleh mikrokontroler. Pada gambar 4.3 menunjukan bahwa data 4 bit yang dikirimkan oleh mikrokontroler dibaca setiap *rising edge clock.* Proses pembacaan data ini dimulai setelah sinyal 'rst' sebagai sinyal reset di non-aktifkan (*low logic*). Data 4 bit dikirimkan tiap siklus, kemudian dalam 3 siklus *clock* data 4 bit digabungkan menjadi data 12 bit. Proses 2D-IDCT membutuhkan 64 data 12 bit, sehingga total menjadi 768 bit. Karena proses pengiriman data per 4 bit, maka untuk mengirim seluruh data membutuhkan 192 siklus *clock.*

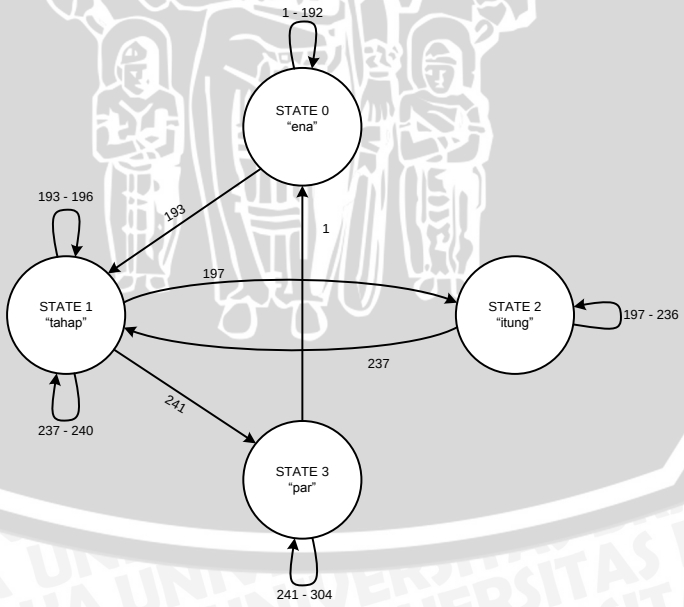


Gambar 4.4 Unit sumber data 2D-IDCT

Gambar 4.4 mejelaskan arsitektur unit sumber data 2D-IDCT. Sinyal "clk_div" dihasilkan oleh unit pembagi detak (*clock divider*), agar *clock* yang masuk ke mikrokontroler lebih lambat daripada *clock* mikrokontroler sendiri sehingga eksekusi interrupt untuk mengirimkan data dapat sinkron dengan sistem 2D-IDCT di FPGA. Sinyal "clk_div" dikeluarkan FPGA melalui pin M13. Sinyal "clk_div" terhubung dengan pin D2 sebagai *interrupt0* pada ATmega8535, sedangkan keluaranya berupa data 4 bit yang terhubung ke pin C0 sampai C3 (LSB ke MSB). Data 4 bit dari mikrokontroler masuk ke FPGA melalui pin L15,K12,L17 dan M15.

4.2.2 Control Signal

Unit pengontrol sinyal berfungsi untuk menghasilkan sinyal-sinyal yang digunakan dalam proses komputasi 2D-IDCT dari mulai data masuk hingga data siap dikeluarkan. Unit pengontrol sinyal akan menghasilkan pencacah internal untuk menghasilkan sinyal pengendali proses. Pada awal proses, unit pengontrol akan menghitung jumlah data yang masuk sampai jumlah data yang dibutuhkan sesuai proses 2D-IDCT. Setiap mikrokontroler mengirimkan 4 bit data sebanyak 3 kali pengiriman, maka akan dihasilkan sinyal "ena" yang berfungsi sebagai *clock enable* pada unit *register*. Sinyal "ena" akan dihasilkan sampai *register* ke 64 sebagai penyimpan data terkahir menerima data masukan. Setelah pencacah internal menghitung 64 data yang telah dikirim, maka akan dihasilkan sinyal "tahap" dan sinyal "itung". Sinyal "tahap" berfungsi sebagai selektor pada unit multiplekser sebelum data masuk ke unit 1D-IDCT. Sinyal "itung" berfungsi sebagai sinyal pencacah untuk pengendalian dalam proses komputasi 1D-IDCT dan bersama sinyal "tahap", kedua sinyal ini akan menghasilkan sinyal w,x,y,z,p,q,r dan s sebagai *clock enable* untuk *register* penyimpan hasil akhir proses 2D-IDCT. Dalam proses akhir unit pengontrol sinyal akan menghasilkan sinyal "par" sebagai sinyal *enable* pada unit *parallel to serial.*

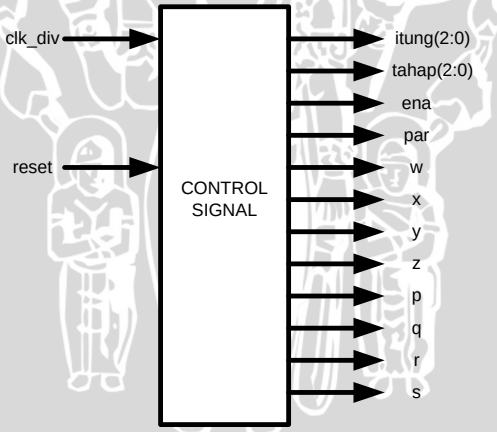


Gambar 4.5 *State Diagram control signal*

Dari gambar 4.5 dapat dijelaskan melalui Tabel 4.4 berikut :

Tabel 4.4 Operasi *Control Signal*

Cacah ke-n	State	Operasi
1 – 192	State 0	*control signal* menghasilkan sinyal "ena" yang digunakan sebagai *clock enable* pada *register files* data masuk.
193 - 196	State 1	*control signal* menghasilkan sinyal "tahap" yang digunakan untuk pemilihan data pada multiplekser.
197 - 236	State 2	*control signal* menghasilkan sinyal "itung" yang digunakan kendali komputasi unit IDCT.
237 - 240	State 1	*control signal* menghasilkan sinyal "tahap" yang digunakan untuk pemilihan data pada multiplekser.
241 - 304	State 3	*control signal* menghasilkan sinyal "par" yang digunakan unit *parallel to serial* dalam mengeluarkan data.



Gambar 4.6 Unit pengontrol sinyal 2D-IDCT

4.2.3 Register Files

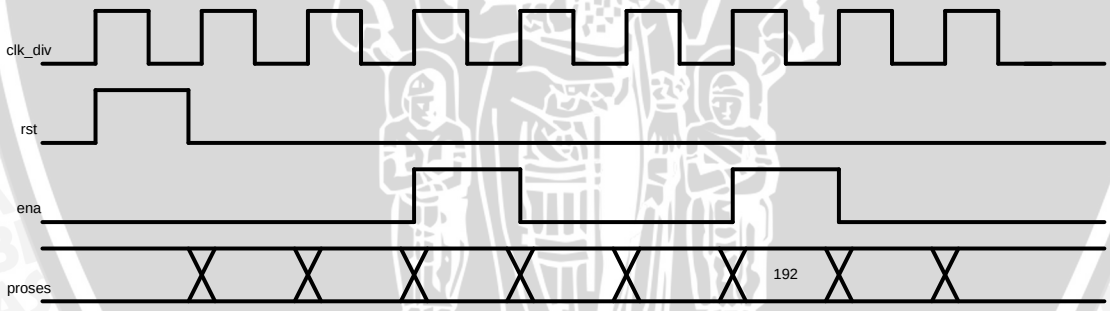
Pada sistem 2D-IDCT unit *register* terdiri dari tiga macam unit *register* yang masing-masing memilki data masukan yang berbeda. Tiga unit *register* tersebut adalah unit *register* sebagai penyimpan data masukan dari mikrokontroler, unti *register* sebagai penyimpan nilai tengah, dan unit *register* sebagai penyimpan hasil akhir proses 2D-IDCT. Jumlah dari *register-register* tersebut sama, yaitu sebanyak 64 *register* yang tersusun sebagai sebuah matriks *register* dengan ukuran 8x8. Baris pertama tersusun

dari *register* 0 sampai *register* 7, baris kedua tersusun dari *register* 8 sampai *register* 15 dan seterusnya sampai dengan baris kedelapan yang tersusun dari *register* 56 sampai *register* 63.

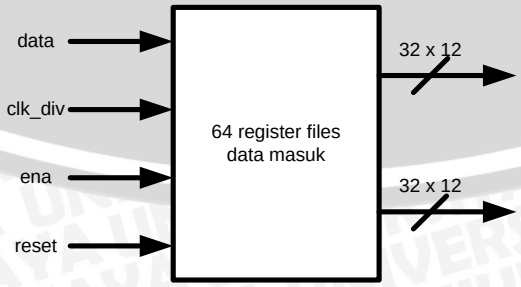
4.2.3.1 Register Files Data Masuk

Gambar 4.7 menjelaskan operasi dari unit *register files* data masuk. Sinyal 'ena' pada *timing diagram* berfungsi sebagai *clock enable* dan diaktifkan setiap 3 siklus *clock*, sehingga data yang tersimpan ke dalam *register* berukuran 12 bit. Sinyal 'ena' akan dihasilkan sampai semua data dikirimkan (sampai 192 siklus *clock*).

Register ini tersusun atas 64 *register* dengan lebar 12 bit sebagai tempat untuk menyimpan 64x12 bit data masukan yang dikirimkan oleh mikrokontroler. *Register* ini bekerja sebagai *register* geser, sehingga data yang masuk pertama nantinya akan digeser sampai *register* terakhir (*register* ke-64). Setelah semua data masuk, maka data di dalam *register* akan terhubung ke dua unit multiplekser untuk dipilih data mana yang akan diproses dalam dua unit 1D-IDCT. Baris pertama, ketiga, kelima dan ketujuh *register files* ini akan dihubungkan ke unit multiplekser satu. Sedangkan baris kedua, keempat, keenam dan kedelapan akan dihubungkan ke unit multiplekser dua.



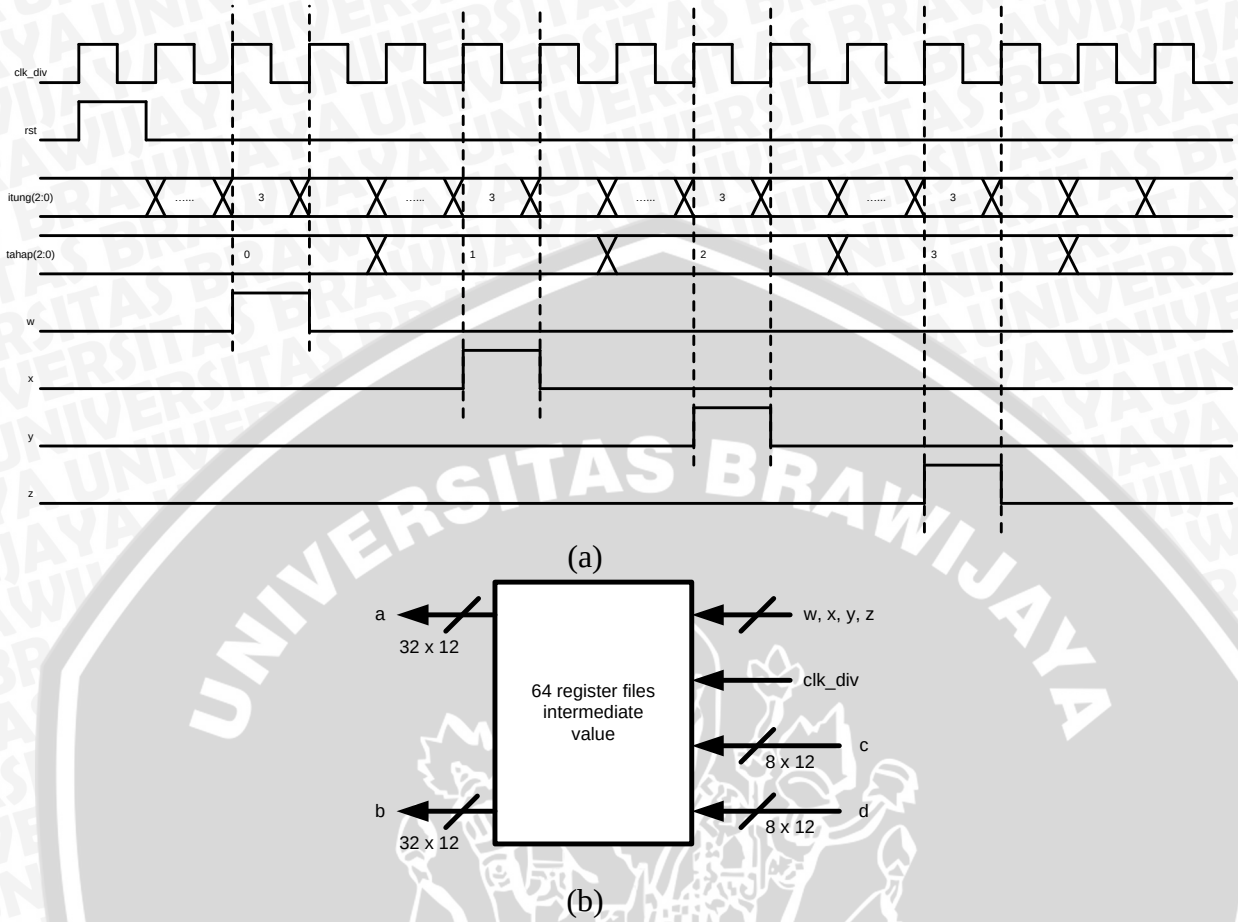
(a)



(b)

Gambar 4.7 (a) *Timing diagram register files* data masuk.
(b) Blok diagram *register files* data masuk.

4.2.3.2 Register Files Intermediate Value



Gambar 4.8 (a) *Timing diagram register files intermediate value*
(b) Blok diagram *register files intermediate value*

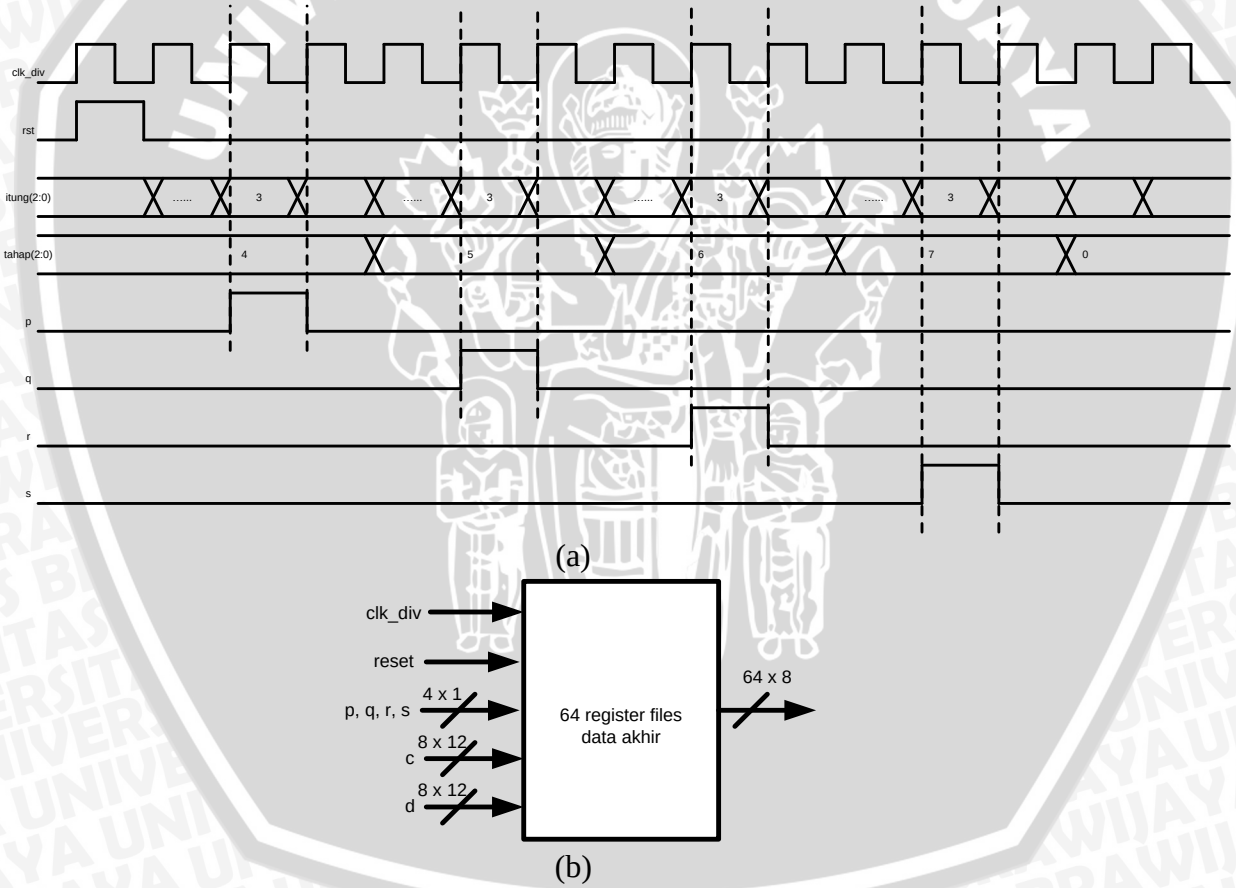
Gambar 4.8 menjelaskan operasi dari unit *register files intermediate value*. Sinyal 'w', 'x', 'y' dan 'z' pada *timing diagram* berfungsi sebagai *clock enable* dan diaktifkan oleh sinyal 'itung' dan sinyal 'tahap'. Sinyal 'w' berfungsi sebagai *clock enable* baris pertama dan kedua *register*. Sinyal 'y' berfungsi sebagai *clock enable* baris ketiga dan keempat *register*. Sinyal 'x' berfungsi sebagai *clock enable* baris kelima dan keenam *register*. Sinyal 'z' berfungsi sebagai *clock enable* baris ketujuh dan kedelapan *register*.

Register ini tersusun atas 64 *register* dengan lebar 12 bit sebagai unit penyimpanan data hasil proses komputasi 1D-IDCT pada bagian baris 8x8 data masukan. Data yang tersimpan dalam *register* ini akan diproses lagi oleh unit 1D-IDCT pada setiap kolomnya untuk menghasilkan hasil akhir proses 2D-IDCT. Sehingga *register* ini akan dihubungkan ke unit multiplekser sebelum terhubung ke unit 1D-IDCT. Kolom pertama, ketiga, kelima dan ketujuh *register* ini terhubung ke unit

multiplekser satu. Sedangkan kolom kedua, keempat, keenam dan kedelapan terhubung ke unit multiplekser dua.

4.2.3.3 Register Files Data Akhir

Gambar 4.9 menjelaskan operasi dari unit *register files* data akhir. Sinyal ‘p’, ‘q’, ‘r’ dan ‘s’ pada *timing diagram* berfungsi sebagai *clock enable* dan diaktifkan oleh sinyal ‘itung’ dan sinyal ‘tahap’. Sinyal ‘p’ berfungsi sebagai *clock enable* kolom pertama dan kedua *register*. Sinyal ‘q’ berfungsi sebagai *clock enable* kolom ketiga dan keempat *register*. Sinyal ‘r’ berfungsi sebagai *clock enable* kolom kelima dan keenam *register*. Sinyal ‘s’ berfungsi sebagai *clock enable* kolom ketujuh dan kedelapan *register*.



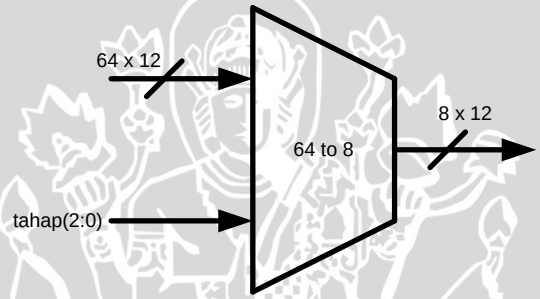
Gambar 4.9 (a) *Timing diagram register files* data akhir
(b) Blok diagram *register files* data akhir

Register ini tersusun atas 64 *register* dengan lebar 8 bit sebagai unit penyimpanan nilai akhir dari proses 2D-IDCT sebelum 8x8 data dikeluarkan lewat unit *parallel to serial*. Berbeda dengan dua *register files* sebelumnya, *register files* ini tidak

terhubung ke unit multiplekser tetapi langsung terhubung dengan unit *parallel to serial*. Pengaturan pengeluaran data di dalam *register* ini dikendalikan oleh unit *parallel to serial,* dengan data pada baris pertama *register* akan dikeluarkan terlebih dahulu, kemudian dilanjutkan dengan baris kedua seterusnya sampai baris kedelapan.

4.2.4 Multiplekser

Unit multiplekser merupakan unit pemilih data, multiplekser yang digunakan adalah multiplkeser 64 ke 8, yang berarti multiplekser mempunyai 64 data masukan dan 8 data keluaran yang dikendalikan oleh sinyal "tahap" yang dihasilkan unit pengontrol sinyal sebagai pemilih data mana yang akan dikeluarkan. Multiplekser 64 ke 8 seperti yang terlihat pada gambar 4.10 dirancang dengan multiplekser 8 ke 1 yang disusun parallel sebanyak 8 unit.



Gambar 4.10 Unit multiplekser 64 to 8

Dalam perancangan sistem 2D-IDCT dengan menggunakan 2 blok 1D-IDCT diperlukan 2 unit multiplekser 64 ke 8 dengan masing-masing unit multiplekser akan terhubung ke unit 1D-IDCT. Multiplekser pertama mendapat masukan dari baris pertama, ketiga, kelima dan ketujuh dari *register files* data masuk, serta kolom pertama, ketiga, kelima dan ketujuh dari *register files intermediate value.* Keluaran multiplekser pertama akan terhubung ke unit 1D-IDCT pertama. Multiplekser kedua mendapat masukan dari baris kedua, keempat, keenam dan kedelapan dari *register files intermediate value*. Keluaran multiplekser kedua akan terhubung ke unit 1D-IDCT kedua.

Tabel 4.4 menjelaskan fungsi selector pada unit multiplekser 1 yang dapat diartikan jika sinyal tahap bernilai 000 pada multiplekser pertama, maka multiplekser memilih delapan data (12bit) baris pertama dari *register files* data masuk sebagai *output* dari multiplekser, demikian seterusnya.

Tabel 4.5 Fungsi selektor multiplekser 1

tahap (2:0)	*Output*
000	Delapan data (12 bit) baris pertama dari *register files* data masuk
001	Delapan data (12 bit) baris ketiga dari *register files* data masuk
010	Delapan data (12 bit) baris kelima dari *register files* data masuk
011	Delapan data (12 bit) baris ketujuh dari *register files* data masuk
100	Delapan data (12 bit) kolom pertama dari *register files intermediate value*
101	Delapan data (12 bit) kolom ketiga dari *register files intermediate value*
110	Delapan data (12 bit) kolom kelima dari *register files intermediate value*
111	Delapan data (12 bit) kolom ketujuh dari *register files intermediate value*

Tabel 4.5 menjelaskan fungsi selector pada unit multiplekser 2 yang dapat diartikan jika sinyal tahap bernilai 000 pada multiplekser kedua, maka multiplekser memilih delapan data (12bit) baris kedua dari *register files* data masuk sebagai *output* dari multiplekser, demikian seterusnya.

Tabel 4.6 Fungsi selektor multiplekser 2

tahap (2:0)	*Output*
000	Delapan data (12 bit) baris kedua dari *register files* data masuk
001	Delapan data (12 bit) baris keempat dari *register files* data masuk
010	Delapan data (12 bit) baris keenam dari *register files* data masuk
011	Delapan data (12 bit) baris kedelapan dari *register files* data masuk
100	Delapan data (12 bit) kolom kedua dari *register files intermediate value*
101	Delapan data (12 bit) kolom keempat dari *register files intermediate value*
110	Delapan data (12 bit) kolom keenam dari *register files intermediate value*
111	Delapan data (12 bit) kolom kedelapan dari *register files intermediate value*

4.2.5 Unit 1D-IDCT

Unit 1D-IDCT merupakan unit aritmatika pada sistem 2D-IDCT. pada perancangan sistem 2D-IDCT dibutuhkan dua unit 1D-IDCT. 1D-IDCT hanya melakukan proses IDCT terhadap 8 data masukan dengan lebar masing-masing data

adalah 12 bit atau disebut 8 titik IDCT. Lebar data masukan dirancang sebesar 12 bit bertanda dikarenakan jika 64 data bernilai piksel maksimal (255) semua maka lebar 12 bit bertanda tersebut mampu untuk mencukupi kebutuhan lebar data hasil DCT yang memiliki nilai paling besar. Proses komputasi pada masing-masing unit dikendalikan oleh sinyal "itung" yang dihasilkan oleh unit pengontrol sinyal serta sinyal "clk_div" sebagai sinyal *clock*. Unit 1D-IDCT pertama mendapat masukan dari multiplekser pertama. Sedangkan unit 1D-IDCT kedua mendapat masukan dari multiplekser kedua.

Dalam satu unit 1D-IDCT menggunakan algortima LLM diperlukan 14 kali proses perkalian, 7 kali proses pengurangan dan 25 kali proses penjumlahan. Proses pengurangan dan penjumlahan pada unit 1D-IDCT menggunakan operator "-" dan "+" sama seperti dalam perancangan sistem 1D-IDCT. Sedangkan untuk proses perkalian hanya satu unit 1D-IDCT yang dapat menggunakan *embedded multiplier 18x18,* karena FPGA hanya menyediakan 20 buah sedangkan satu unit 1D-IDCT memerlukan 14 pengali. Dengan alasan tersebut maka unit 1D-IDCT kedua menggunakan desain pengali *constant multiplier*.

Cara pertama untuk proses perkalian dengan menggunakan *embedded multiplier 18x18 bit* memiliki keterbatasan jumlah pengali yang digunakan. Pada FPGA yang digunakan hanya menyediakan 20 pengali, tetapi dalam proses 1D-IDCT dengan algoritma LLM hanya membutuhkan 14 proses perkalian sehingga *embedded multiplier* dapat dimanfaatkan. Untuk menggunakan *embedded multiplier* adalah menambahkan kode berikut dalam listing kode VHDL :

```
MULT18x18_inst : MULT18x18
port map (
P => P,  --36-bit multiplier output
A=> A,  --18-bit multiplier input
B=> B,  --18-bit multiplier input
);
```

Cara kedua untuk melakukan proses perkalian adalah dengan cara merancang sendiri desain pengali. Prinsip perancangan perkalian adalah dengan metode *constant multiplier* (pengali konstan), yaitu mengubah 13 koefisien pengali algoritma LLM yang terlihat pada tabel 4.6 menjadi 13 program pengali. Masing-masing koefisien pengali algoritma LLM akan dibulatkan dengan faktor 2^{10} atau 1024. Dengan menggunakan tabel Booth, koefisien pengali yang telah dibulatkan akan dikodekan untuk menghasilkan *partial product,* kemudian *partial product* yang dihasilkan akan

dijumlahkan untuk memperoleh hasil perkalian. Pada skripsi ini pengali yang dirancang berukuran 12x12 bit, hal ini dikarenakan masukan unit IDCT berupa data berukuran 12 bit.

Tabel 4.7 Koefisien pengali algoritma LLM dan pembulatnya

Koefisien Pengali	Amplitudo	Pembulatan dengan faktor 2^{10}
a0	0,353555	362
a1	-0,65328	-669
a2	0,2706	277
a3	0,19134	196
aa	0,10558	108
ab	0,725885	743
ac	1,086365	1112
ad	0,530795	544
ae	-0,31819	-326
af	-0,90613	-928
ag	0,69352	-710
ah	-0,13795	-141
ai	0,415735	426

Tahap-tahap perancangan program pengali untuk masing-masing koefisien pengali algoritma LLM adalah sebagai berikut :

1. Data masukan adalah ‘A’ (*multiplicand*) berukuran 12 bit dan koefisien pengali adalah ‘b’ (*multiplier*).
2. Mengubah koefisien pengali menjadi bilangan bulat dengan cara dikali bilangan 1024.
3. Ubah koefisien pengali yang telah dibulatkan menjadi bilangan biner 12 bit, kemudian tambahkan bit ‘0’ pada LSB-nya sehingga menjadi berukuran 13 bit.
4. Mencuplik 3 bit – 3 bit pada bilangan pengali sehingga diperoleh $b_7b_6b_5, b_5b_4b_3, b_3b_2b_1 \; dan \; b_1b_0 0$. Kemudian setiap 3 bit tersebut mulai dari LSB dieksekusi dengan operasi yang sesuai dengan daftar modifikasi Booth yang digambarkan pada Tabel 4.7.
5. Menjumlahkan *partial product* dengan gerbang-gerbang logika dasar.
6. Hasil akhir perhitungan berupa bilangan 24 bit (hasil perkalian 12 bit dengan 12 bit), namun yang diambil sebagai hasil akhir adalah bit ke 21 sampai bit ke 10 karena bit ke 9 sampai bit ke 0 digeser ke kanan (dibagi 10 bit).

Tabel 4.8 Tabel modifikasi Booth

Multiplier			*Multiplicand recoded*	*Partial Product*
i+1	i	i-1		
0	0	0	0 x *multiplicand*	Nol
0	0	1	+1 x *multiplicand*	Sama dengan bilangan yang dikali
0	1	0	+1 x *multiplicand*	Sama dengan bilangan yang dikali
0	1	1	+2 x *multiplicand*	Geser ke kiri 1 bit
1	0	0	-2 x *multiplicand*	Diubah jadi *2'comp* kemudian digeser ke kiri 1 bit
1	0	1	-1 x *multiplicand*	Diubah jadi *2's comp*
1	1	0	-1 x *multiplicand*	Diubah jadi *2's comp*
1	1	1	0 x *multiplicand*	nol

Berikut ini adalah contoh perancangan program pengali untuk koefisien pengali *a0* = 0,3535.

1. 0,3535 x 1024 = 362
2. 362 = 0001 0110 1010 + '0' → 000 010 011 101 101 100
3. Membuat tabel untuk menjabarkan operasi algortima Booth. Pada Tabel 4.8 dijelaskan penjabaran awal algoritma Booth dalam menghasilkan *partial product*. Tabel 4.9 mejelaskan pengubahan bentuk *2's complement* dari A (bilangan yang dikali) menjadi bentuk *invers* ditambah dengan '1'. Kemudian Tabel 4.10 menjelaskan operasi terkahir dari desain pengali.

Tabel 4.9 Operasi awal algotima Booth

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Booth Code
A11"	A11"	A11"	A11"	A11"	A11"	A11"	A11"	A11"	A11"	A11"	A10"	A9"	A8"	A7"	A6"	A5"	A4"	A3"	A2"	A1"	A0"	100
A11"	A11"	A11"	A11"	A11"	A11"	A11"	A11"	A11"	A10"	A9"	A8"	A7"	A6"	A5"	A4"	A3"	A2"	A1"	A0"			101
A11"	A11"	A11"	A11"	A11"	A11"	A11"	A10"	A9"	A8"	A7"	A6"	A5"	A4"	A3"	A2"	A1"	A0"					101
A11	A11	A11	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	0							011
A11	A11	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0									010
0	0	0	0	0	0	0	0	0	0	0	0											000
M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0	X	X	X	X	X	X	X	X	X	X	

A" = 2's complement A

Tabel 4.10 Pengubahan bentuk 2's *complement* A

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A10'	A9'	A8'	A7'	A6'	A5'	A4'	A3'	A2'	A1'	A0'	0
A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A10'	A9'	A8'	A7'	A6'	A5'	A4'	A3'	A2'	A1'	A0'	1	
A11'	A11'	A11'	A11'	A11'	A11'	A11'	A10'	A9'	A8'	A7'	A6'	A5'	A4'	A3'	A2'	A1'	A0'	0	1		
A11	A11	A11	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	0	0	1				
A11	A11	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0								
M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0	X	X	X	X	X	X	X	X	X	X

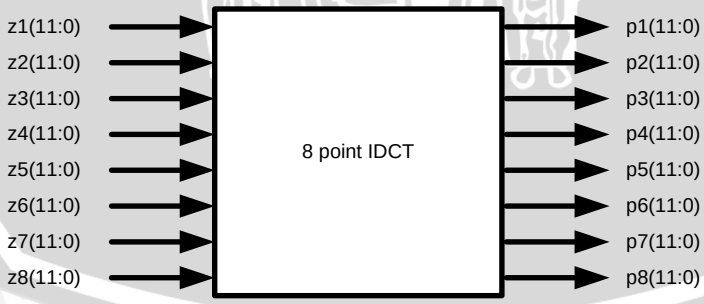
A' = invers A

Tabel 4.11 Operasi terakhir desain pengali

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A10'	A9'	A8'	A7'	A6'	A5'	A4'	A3'	A2'	A1'	A0'	0
A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A11'	A10'	A9'	A8'	A7'	A6'	A5'	A4'	A3'	A2'	A1'	A0'	1	
A11'	A11'	A11'	A11'	A11'	A11'	A11'	A10'	A9'	A8'	A7'	A6'	A5'	A4'	A3'	A2'	A1'	A0'	0	1		
S113	S113	S113	S113	S113	S113	S113	S112	S11	S10	S19	S18	S17	S16	S15	S14	S13	S12	S11	S10		
C113	C113	C113	C113	C113	C113	C113	C112	C11	C10	C19	C18	C17	C16	C15	C14	C13	C12	C11	C10		
A11	A11	A11	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	0	0	1				
S216	S216	S216	S216	S215	S214	S213	S212	S211	S210	S29	S28	S27	S26	S25	S24	S23	S22	S21	S20		
C217	C217	C217	C216	C215	C214	C213	C212	C211	C210	C29	C28	C27	C26	C25	C24	C23	C22	C21	C20		
A11	A11	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0								
S317	S317	S317	S316	S315	S314	S313	S312	S311	S310	S39	S38	S37	S36	S35	S34	S33	S32	S31	S30		
C318	C318	C317	C316	C315	C314	C313	C312	C311	C310	C39	C38	C37	C36	C35	C34	C33	C32	C31	C30		
m17	m16	m15	m14	m13	m12	m11	m10	m9	m8	m7	m6	m5	m4	m3	m2	m1	m0	X	X	X	X

C = Carry bit *S = Sum bit*

Bagian yang diwarnai pada tabel diatas dijumlahkan dengan gerbang-gerbang logika. Dan hasil akhir perkalian adalah m17 sampai m7 (12bit).



Gambar 4.11 Unit 1D-IDCT

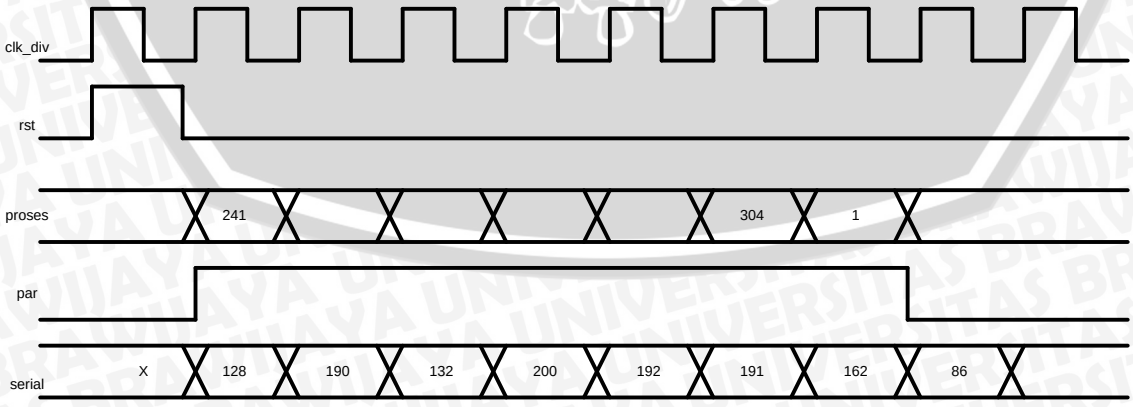
Gambar 4.11 memperlihatkan unit komputasi 8 titik IDCT dengan 8 masukan dan 8 keluaran. Data masukan dalam operasi 1D-IDCT berupa bilangan bulat, dan data yang dihasilkan juga berupa bilangan bulat, sehingga semua proses komputasi dalam

unit IDCT merupakan komputasi bilangan bulat. Seperti yang telah dijelaskan sebelumnya, algoritma LLM memilki koefisien pengali berupa bilangan pecahan untuk itu sebelum diproses ke dalam unit IDCT maka koefisien tersebut harus dibulatkan terlebih dahulu dengan faktor 1024. Proses pembulatan ini akan mengurangi kompleksitas perancangan sistem meskipun dengan mengorbankan sedikit akurasi dari hasil komputasi.

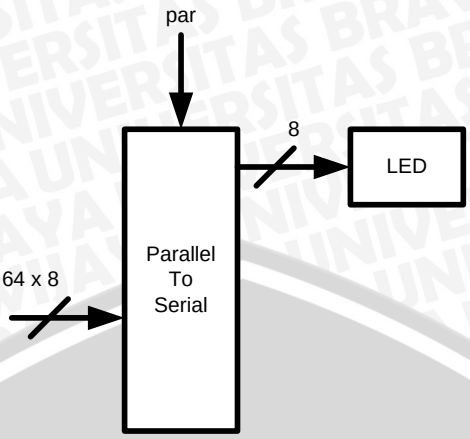
4.2.6 Parallel to Serial Converter

Unit ini merupakan unit terakhir dari sistem 2D-IDCT yang berfungsi untuk mengeluarkan 8x8 data (64 data) yang telah disimpan dalam *register files* nilai akhir satu persatu. Keluaran dari unit ini akan ditampilkan melalui lampu LED yang tersedia dalam FPGA untuk mengecek data keluaran sudah sesuai dengan yang diinginkan atau tidak.

Pengendalian keluaran unit ini diatur oleh sinyal "par" yang dihasilkan oleh unit pengontrol sinyal. Saat sinyal "par" bernilai '0' maka data di dalam *register* akhir akan di *latch* terlebih dahulu, dan jika sinyal "par" bernilai '1' maka data di dalam *register files* nilai akhir akan dikeluarkan satu per satu setiap *rising edge* dari sinyal *clock*. Gambar 4.12 menjelaskan *timing diagram parallel to serial*, sinyal "par" akan diaktifkan (logika '1') saat pencacah 'proses' bernilai 241 sampai 304 (64 cacah untuk 64 data keluar), selain nilai tersebut sinyal 'par' akan berlogika '0'. Sinyal 'serial' merupakan keluaran dari unit *parallel to serial*. Keluaran dari unit ini akan ditampilkan dalam data biner 8 bit yang dapat dilihat pada LED di FPGA.



(a)

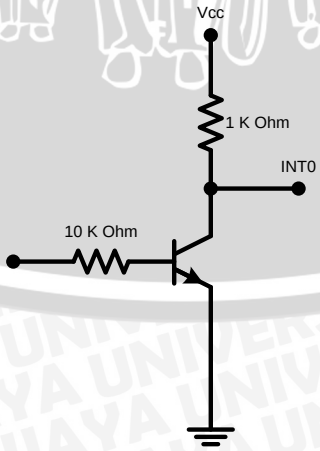


(b)

Gambar 4.12 (a) *Timing diagram parallel to serial*
(b) Blok diagram *parallel to serial*

4.2.7 Unit Penghasil Detak

Unit penghasil detak atau *clock generator* berfungsi menghasilkan sinyal "clk_div" yang akan digunakan sebagai *clock* sistem dan dikeluarkan ke pin FPGA sebagai *input* mikrokontroler untuk eksekusi interupsi eksternal. Karena interupsi mikrokontroler membutuhkan *clock* yang lebih lambat dari mikrokontroler (kurang dari 8 MHz) maka sinyal *clock* yang dibangkitkan oleh FPGA sebesar 50 MHz harus dibagi terlebih dahulu dengan program pembagi *clock* agar sinyal "clk_div" yang dihasilkan memiliki frekuensi yang lebih kecil dibandingkan dengan frekuensi *clock* mikrokontroler.



Gambar 4.13 Rangkaian pengubah level tegangan

Sinyal *clock* yang dihasilkan oleh FPGA akan dilewatkan pada rangkaian pengubah level tegangan pada gambar 4.13, hal ini dikarenakan terdapat beda level tegangan antara FPGA (3,3 Volt) dengan mikrokontroler (5 Volt). Rangkaian pengubah level tegangan ini menggunakan transistor NPN sebagai saklar. Pada saat FPGA mengirim logika '1' (3,3 Volt) maka transistor akan "on" sehingga keluaran di kaki kolektor adalah 0 Volt (logika '0') dan sebaliknya. Karena ada keterbalikan data yang dikirim maka sinyal *clock* dari FPGA akan di *invers* terlebih dahulu sebelum dikirim ke rangkaian pengubah level tegangan agar keluaran dari rangkaian akan sama dengan *clock* sistem di FPGA.

Agar hasil akhir 1D-IDCT dapat diamati oleh pengamat, maka sinyal "clk_div" dibuat sangat lambat dan harus memiliki periode antara satu sampai dua detik. Program VHDL untuk menghasilkan sinyal "clk_div" yang lebih lambat dibandingakn dengan *clock* mikrokontroler adalah sebagai berikut :

```
lambat:process(clk,rst,bagi)
begin
if(rst = '1')then
bagi <= (others => '0');
elsif (clk'event and clk = '1')then
bagi <= bagi + 1'
end if;
end process;
clk_div <= bagi (25);--sinyal clock yg dihasilkan untuk sistem
div <= not bagi(25);--sinyal clock yg dikirimkan ke
mikrokontroler
```

BAB V
PENGUJIAN DAN PEMBAHASAN

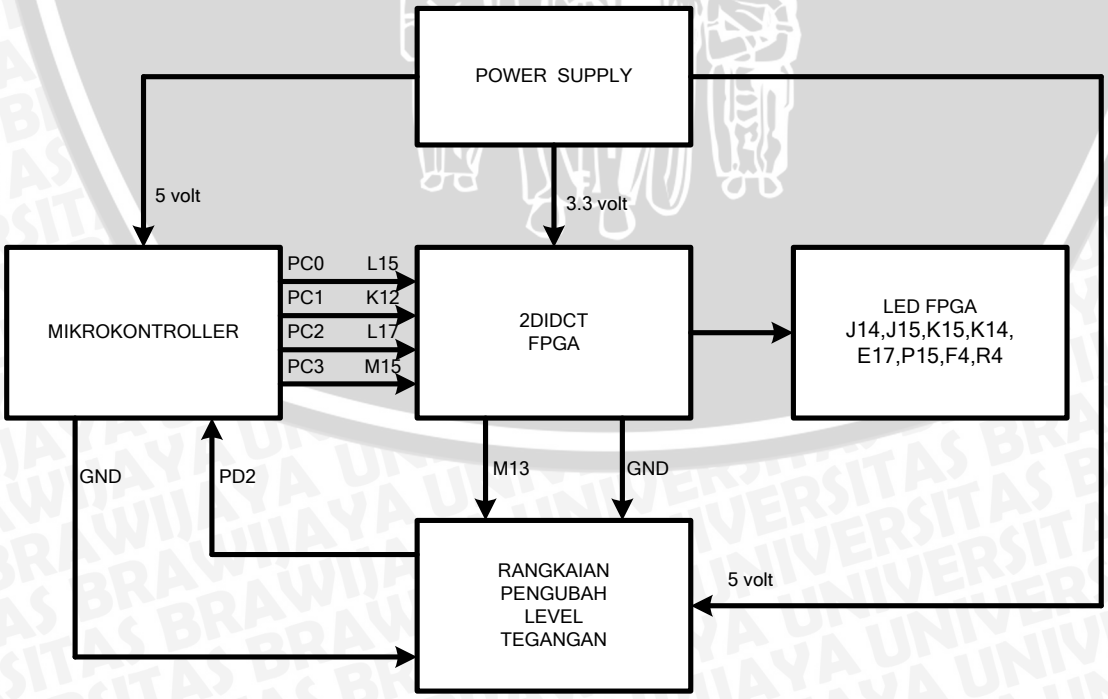
Setelah sistem dirancang dan diimplementasikan pada keeping FPGA XC3S500E maka langkah selanjutnya adalah melakukan pengujian terhadap sistem tersebut. Pengujian akan membuktikan tingkat keberhasilan perancangan sistem serta akurasi dalam proses perhitungan. Pengujian dan pembahasan sistem meliputi beberapa aspek :

1. Akurasi proses perhitungan 2D-IDCT meliputi akurasi perhitungan keseluruhan.
2. Unjuk kerja sistem pada FPGA Spratan 3E yaitu seberapa besar kapasitas yang digunakan dalam implementasi.

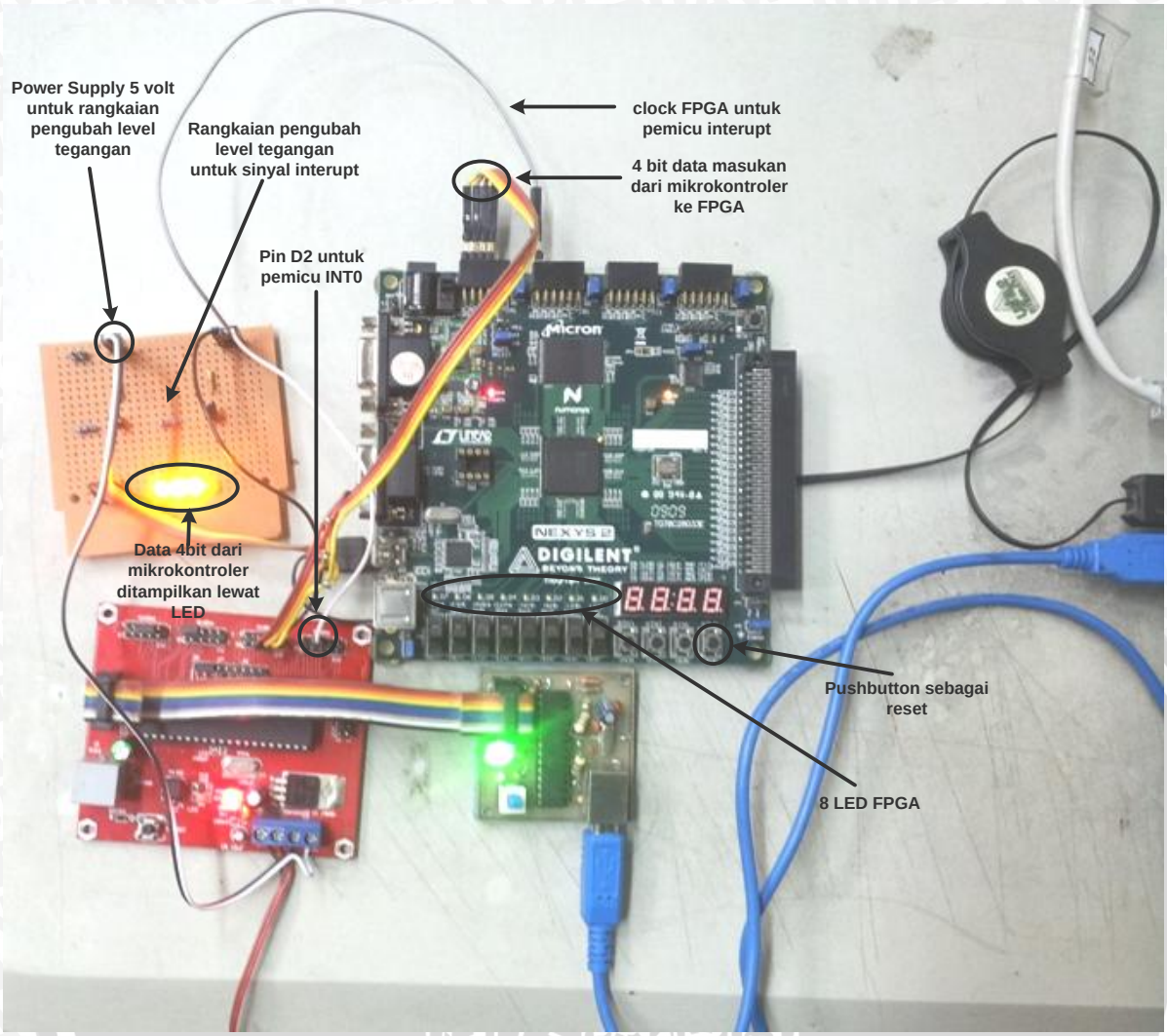
Akurasi algoritma LLM yang diterapkan pada sistem akan dibandingkan dengan hasil yang telah dihitung dengan program MATLAB. Unjuk kerja sistem akan diukur kapasitas sistem yang digunakan pada keeping FPGA (jumlah *slice* atau logika yang digunakan).

5.1 Pengujian Implementasi Sistem

Cara pengujian sistem 2D-IDCT diperlihatkan pada gambar 5.1. Gambar 5.2 memperlihatkan sistem yang telah diimplementasikan ke dalam *hardware.*



Gambar 5.1 Cara pengujian implementasi 2D-IDCT



Gambar 5.2 Implementasi sistem di FPGA Spartan 3E

5.2 Akurasi Komputasi IDCT

Akurasi merupakan salah satu aspek penting yang harus diperhatikan dalam implementasi sistem. Akurasi proses komputasi 2D-IDCT dengan algoritma LLM yang diimplementasikan pada FPGA Xilinx Spartan 3E akan dibandingkan dengan perhitungan menggunakan *tool box* 2D-IDCT MATLAB, sehingga dapat dihitung seberapa besar nilai kesalahan (*error*) yang dihasilkan dari proses komputasi 2D-IDCT pada implementasi FPGA.

Dalam pengujian akurasi yang akan diamati adalah *error* mutlak, yaitu selisih kesalahan atau perbedaan antara nilai hasil komputasi FPGA dengan data masukan asli. *error* mutlak dirumuskan pada persamaan 5.1.

Error mutlak = |UE – UF| (5.1)

UE = data masukan asli (data piksel)

UF = hasil komputasi IDCT dalam FPGA

Selain mengamati *error* mutlak, akan diamati *error* relatif yaitu persentase nilai *error* mutlak terhadap data piksel (data masukan asli) yang dapat dirumuskan pada persamaan 5.2.

$$\textit{Error}\ \text{Relatif} = \frac{\textit{error}\ \text{mutlak}}{\text{nilai data piksel}} \text{ x } 100\% \qquad (5.2)$$

5.2.1 Akurasi Komputasi 2D-IDCT

Implementasi 2D-IDCT dengan algoritma LLM pada FPGA Xilinx Spartan 3E mengujicobakan 2 jenis masukan yang berbeda. Komputasi 2D-IDCT dengan algoritma LLM pada FPGA akan dibandingkan hasilnya dengan komputasi yang dilakukan oleh *tool* 2D-IDCT MATLAB, untuk melihat besar selisih/*error* yang dihasilkan terhadap data piksel masukan. Rancangan sistem 2D-IDCT dengan algortima LLM terdiri dari 64 masukan dan 64 keluaran. Masukan 2D-IDCT dikirimkan oleh mikrokontroler per 4 bit sampai membentuk masukan 64x12 bit, sedangkan 64 keluaran ditampilkan secara berurutan dengan selang sekitar 1,342 detik melalui LED yang berupa data biner tak bertanda 8 bit. Data diujicobakan sebanyak 2 kali untuk masing-masing nilai piksel 8x8 yang berbeda.

Dengan menggunakan persamaan 5.1 dan 5.2, dari kedua percobaan tersebut didapatkan hasil sebagai berikut :

1. Percobaan I

Tabel 5.1 Akurasi komputasi 2D-IDCT algoritma LLM percobaan 1

Indeks Matriks	Data Piksel	DCT MATLAB	Keluaran IDCT		*Error* FPGA	
			MATLAB	FPGA(8bit)	Mutlak	Relatif (%)
00	110	935	110	106	4	3,636
01	110	-63	110	110	0	0
02	118	18	118	115	3	2,542
03	118	-7	118	118	0	0
04	121	7	121	120	1	0,826
05	126	13	126	125	1	0,794
06	131	-7	131	127	4	3,053
07	131	0	131	127	4	3,053
10	108	74	108	106	2	1,852

Indeks Matriks	Data Piksel	DCT MATLAB	Keluaran IDCT		*Error* FPGA	
			MATLAB	FPGA(8bit)	Mutlak	Relatif (%)
11	111	-3	111	111	0	0
12	125	-20	125	125	0	0
13	122	-21	122	121	1	0,819
14	120	-18	120	120	0	0
15	125	-11	125	125	0	0
16	134	8	134	134	0	0
17	135	5	135	135	0	0
20	106	-64	106	104	2	1,887
21	119	3	119	118	1	0,840
22	129	5	129	128	1	0,775
23	127	15	127	126	1	0,787
24	125	10	125	125	0	0
25	127	9	127	125	2	1,572
26	138	1	138	139	1	0,752
27	144	-1	144	143	1	0,694
30	110	4	110	110	0	0
31	126	3	126	126	0	0
32	130	7	130	129	1	0,769
33	133	9	133	134	1	0,752
34	133	-3	133	133	0	0
35	131	2	131	132	1	0,763
36	141	1	141	140	1	0,709
37	148	-1	148	146	2	1,351
40	115	3	115	116	1	0,869
41	116	1	116	115	1	0,862
42	119	-4	119	119	0	0
43	120	1	120	121	1	0,833
44	122	3	122	121	1	0,814
45	125	-3	125	124	1	0,800
46	137	0	137	136	1	0,730
47	139	1	139	138	1	0,719
50	115	8	115	112	3	2,608
51	106	-2	106	105	1	0,943
52	99	-1	99	96	3	3,030
53	110	0	110	109	1	0,910
54	107	0	107	105	2	1,869
55	116	-3	116	115	1	0,862
56	130	-3	130	131	1	0,769
57	127	-2	127	127	0	0
60	110	-3	110	110	0	0
61	91	0	91	91	0	0
62	82	1	82	83	1	1,219
63	101	0	101	110	1	0,990
64	99	0	99	98	1	1,010

Indeks Matriks	Data Piksel	DCT MATLAB	Keluaran IDCT		*Error* FPGA	
			MATLAB	FPGA(8bit)	Mutlak	Relatif (%)
65	104	0	104	103	1	0,961
66	120	-1	120	120	0	0
67	118	-1	118	117	1	0,847
70	103	-2	103	100	3	2,913
71	76	-1	76	77	1	1,316
72	70	2	70	69	1	1,428
73	95	0	95	95	0	0
74	92	0	92	90	2	2,174
75	91	0	91	91	0	0
76	107	1	107	105	2	1,869
77	106	2	106	104	2	1,887
Rata-rata					1,09	0,96

Tabel 5.1 menginformasikan hasil komputasi 2D-IDCT algoritma LLM setelah diimplementasikan pada FPGA dan kemudian hasilnya dibandingkan terhadap komputasi menggunakan *tool box* 2D-IDCT MATLAB. Data masukan sistem berjumlah 64 data hasil 2D-DCT menggunakan *tool box* MATLAB. Indeks matriks menunjukan penyusunan 64 data masukan menjadi matriks berukuran 8x8. Data dengan indeks matriks 27 memilki arti bahwa data terletak pada baris ketiga dan kolom ke delapan dari matriks 8x8. Tabel 5.1 juga menginformasikan bahwa percobaan 1 memberikan nilai rata-rata *error* mutlak sebesar 1,09 dan nilai rata-rata *error* relatif sebesar 0,96 % untuk kisaran data masukan 70 sampai dengan 148 dari data pembanding.

2. Percobaan 2.

Tabel 5.2 Akurasi komputasi 2D-IDCT algoritma LLM percobaan 2

Indeks Matriks	Data Piksel	DCT MATLAB	Keluaran IDCT		*Error* FPGA	
			MATLAB	FPGA(8bit)	Mutlak	Relatif (%)
00	207	1682	207	202	5	2,415
01	209	-9	209	206	3	1,435
02	211	0	211	209	2	0,948
03	210	0	210	208	2	0,952
04	210	-1	210	208	2	0,952
05	211	0	211	211	0	0
06	213	0	213	211	2	0,939

Indeks Matriks	Data Piksel	DCT MATLAB	Keluaran IDCT		*Error* FPGA	
			MATLAB	FPGA(8bit)	Mutlak	Relatif (%)
07	212	-1	212	208	4	1,887
10	209	1	209	210	1	0,478
11	210	-1	210	210	0	0
12	209	-1	209	208	1	0,478
13	210	-1	210	208	2	0,952
14	210	-2	210	209	1	0,476
15	212	0	212	211	1	0,472
16	212	0	212	211	1	0,472
17	212	1	212	212	0	0
20	208	2	208	209	1	0,481
21	210	0	210	211	1	0,476
22	211	1	211	212	1	0,474
23	209	-1	209	208	1	0,478
24	211	-1	211	209	2	0,948
25	212	0	212	211	1	0,472
26	212	0	212	212	0	0
27	211	0	211	210	1	0,474
30	208	-2	208	208	0	0
31	207	0	207	206	1	0,483
32	211	0	211	212	1	0,474
33	211	1	211	211	0	0
34	211	0	211	211	0	0
35	211	0	211	211	0	0
36	211	-1	211	212	1	0,474
37	212	-1	212	212	0	0
40	207	-1	207	208	1	0,483
41	209	-2	209	207	2	0,957
42	207	-1	207	206	1	0,483
43	210	-2	210	209	1	0,476
44	209	0	209	209	0	0
45	211	1	211	210	1	0,474
46	211	0	211	210	1	0,474
47	211	0	211	210	1	0,474
50	209	1	209	209	0	0
51	210	0	210	210	0	0
52	207	-1	207	208	1	0,483
53	209	-1	209	208	1	0,478
54	210	0	210	210	0	0,476
55	211	1	211	209	2	0,474
56	211	2	211	210	1	0,474
57	210	2	210	210	0	0
60	210	0	210	212	2	0,952
61	209	0	209	208	1	0,478
62	210	0	210	208	2	0,952

Indeks Matriks	Data Piksel	DCT MATLAB	Keluaran IDCT		*Error* FPGA	
			MATLAB	FPGA(8bit)	Mutlak	Relatif (%)
63	210	0	210	208	2	0,952
64	211	-2	211	209	2	0,948
65	211	-1	211	211	0	0
66	211	-1	211	211	0	0
67	212	0	212	212	0	0
70	209	0	209	206	3	1,435
71	210	0	210	208	2	0,952
72	209	0	209	209	0	0
73	210	0	210	210	0	0
74	210	-1	210	208	2	0,952
75	211	0	211	212	1	0,474
76	212	0	212	211	1	0,472
77	213	0	213	211	2	0,939
Rata-rata					1,109	0,528

Tabel 5.2 menginformasikan bahwa hasil komputasi 2D-IDCT percobaan 2 memberikan nilai rata-rata *error* mutlak sebesar 1,109 dan nilai rata-rata *error* relatif sebesar 0,528 % untuk kisaran data masukan 207 sampai dengan 213 dari data pembanding. Dari percobaan 1 dan percobaan 2 nilai kesalahan yang terjadi disebabkan oleh beberapa hal sebagai berikut :

a. Nilai kesalahan komputasi 2D-IDCT algoritma LLM merupakan akumulasi dari nilai kesalahan dalam dua kali proses 1D-IDCT, yaitu nilai kesalahan yang dihasilkan pada proses 1D-IDCT baris dan pada proses 1D-IDCT kolom.
b. Nilai kesalahan terjadi karena proses komputasi melibatkan pembulatan hasil komputasi, karena sistem hanya dapat mengolah bilangan bulat.
c. Adanya *error* dalam proses perkalian manggunakan desain pengali VHDL menyebabkan adanya *error* dalam komputasi 2D-IDCT.

5.3 Unjuk Kerja Implementasi 2D-IDCT

Unjuk kerja implementasi sistem 2D-IDCT dapat dilihat pada *report* implementasi. Kapasitas (*area*) merupakan aspek yang akan dianalisa pada sistem.Implementasi sistem 2D-IDCT algoritma LLM seperti gambar 4.2 pada FPGA Spartan 3E XC3S500E menghasilkan *report* implementasi yang terlihat pada tabel 5.3

Tabel 5.3 Penggunaan logika pada implementasi 2D-IDCT

Jenis Pemakaian Logika	Kapasitas yang digunakan	Kapasitas yang tersedia	Persentasi Pemakaian
Jumlah *Slices*		4656	54 %
Jumlah LUT	3532	9312	37%
Jumlah IOB	15	232	6%
Jumlah MULT18x18	14	20	70%
Jumlah GCLKs	2	24	8%

Penggunaan kapasitas logika dapat diminimalkan dengan meminimalkan perancangan sistem serta tata cara pemrograman VHDL. Dari tabel 5.3 dapat disimpulkan bahwa sistem 2D-IDCT dapat diimplementasikan pada FPGA karena pemakaian kapasitas logikanya tidak melebihi kapasitas logika yang tersedia pada FPGA.

Penggunaan *clock* pada desain 2D-IDCT dikarenakan dalam proses implementasi dilakukan dengan proses *sequential*. Untuk pengolahan 8x8 data dengan proses *sequential* seperti gambar 4.2 membutuhkan 304 siklus *clock*, jika 1 siklus *clock* memilki periode 1 μs, maka untuk memproses 1 blok data (8x8 data) dengan sistem pada gambar 4.2 membutuhkan waktu 304 μs.

Untuk proses parallel, pengolahan 8x8 IDCT (2D-IDCT) membutuhkan 16 unit 1D-IDCT, tetapi karena kapasitas *slice* FPGA yang digunakan belum dapat memenuhi untuk implementasi 8x8 IDCT dengan proses parallel, maka hanya digunakan 2 unit 1D-IDCT untuk mengolah 8x8 data. Penggunaan 2 unit 1D-IDCT menyebabkan proses pengolahan 8x8 data menjadi *sequential*.

5.4 Timing Summary

Xilinx menyediakan *tools* untuk menampilkan *timing summary* dari perancangan sistem. *Timing summary* akan diperoleh pada saat proses sintesis, hasil perhitungan *software* Xilinx. Hasil *timing summary* merupakan hasil perhitungan secara simulasi sehingga tidak dapat menggambarkan *latency time*. Perancangan sistem 2D-IDCT pada gambar 4.2 mengahsilkan *timing summary* sebagai berikut :

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 29.362ns (Maximum Frequency: 34.058MHz)
   Minimum input arrival time before clock: 4.679ns
   Maximum output required time after clock: 5.518ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

BAB VI

PENUTUP

6.1 Kesimpulan

Berdasarkan pengujian dan pembahasan implementasi sistem maka dapat diambil beberapa kesimpulan sebagai berikut :

1. Nilai kesalahan (*error*) hasil perhitungan 2D-IDCT setelah diimplementasikan pada FPGA menggunakan algoritma LLM terjadi karena proses komputasi hanya melibatkan bilangan bulat baik dalam proses perkalian maupun penjumlahan serta pengurangan.
2. implementasi 2D-IDCT menggunakan 2525 *slices* atau 54 % dari kapasitas total *slices*, 3532 LUT atau 37 % dari kapasitas total LUT, 15 IOB atau 6 % dari kapasitas total IOB, 14 *embedded multiplier* atau 70 % dari kapasitas total *embedded multiplier* dan 2 GCLKs atau 8 % dari kapasitas total GCLKs FPGA XC3S500E FG320.
3. Total waktu tunda setelah 8 data 12 bit terkirim sampai data valid keluar dari untai 1D-IDCT pada implementasi sebesar 4 siklus *clock*, jika satu siklus memiliki periode 1µs, maka total waktu tunda yang terjadi sebesar 4 µs.

6.2 Saran

Implementasi IDCT pada FPGA ini masih terdapat kekurangan dan kelemahan, oleh karena itu masih diperlukan penyempurnaan untuk pengembangan kedepannya. Berikut adalah beberapa hal membuka kemungkinan terhadap pengembangan lebih lanjut antara lain :

1. Selain menggunakan mikrokontroler sebagai sumber data masukan, modul RS-232 yang tersedia pada FPGA XC3S500E dapat dimanfaatkan sebagai sumber data masukan serial melalui PC.
2. Pada implementasi 2D-IDCT unjuk kerja sistem dapat diperbaiki dengan memanfaatkan *block RAM* yang tersedia pada FPGA XC3S500E sebagai media penyimpanan data komputasi, karena penggunaan *register files* akan lebih banyak membutuhkan *slices flip-flop*.

DAFTAR PUSTAKA

Atmel. 2013. Available at http://www.atmel.com/Images/2502s.pdf [Diakses : 22 Juni 2013]

C.-Y. Pai, W. Lynch, and A. Al-Khalili, "Low-power data-dependent 8 times;8 dct/idct for video compression,"Vision, *Image and Signal Processing, IEEE Proceedings-*, vol. 150, no. 4, pp. 245-255, aug. 2003.

Hung, A.C., and Meng, Teresa H. 1994. *A Comparison of Fast Inverse Discrete Cosine Transform Algorithms*, Multimedia System 2: 204-217. Stanford University :Stanford.

J. Shan, C. Chen and E. Yang, (2000), "High performance 2-D IDCT for Image/Video Decoding based on FPGA", *Audio, Language and Image Processing (ICALIP), 2012 International Conference on*, 2012, pp. 33-38.

Loeffler, C., Ligtenberg, A. and Moschytz, G. S.,"*Practical Fast 1-D DCT Algorithms With 11 Multipliers*" Proc. of IEEE Int. Conf. on ASSP ,1989, pp. 988-991.

N. August and D. S. Ha, "Low power design of dct and idct for low bit rate video codecs," *Multimedia, IEEE Transactions on*, vol. 6, no. 3, pp. 414 – 422, june 2004.

Swamy, R., Khorasani, M., Liu, Y., Elliott, D. and Bates, S. 2005. *A fast, pipelined implementation of a two-dimensional inverse discrete cosine transform*. pp. 665—668

Xilinx. 2013. Available at http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf [Diakses : 22 Juni 2013].

LAMPIRAN

L.1 Listing Program Pengiriman Data dari Mikrokontroler

```
#include <mega8535.h>
#include <delay.h>
#include <stdio.h>

int counter;

//External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{ ///program interrupt///
counter++;
}

//Declare your global variables here
void init(void)
{
//Declare your local variables here

PORTA=0x00;
DDRA=0xff;

PORTB=0x00;
DDRB=0x00;

//Port C initialization
//Pin data keluaran PC0,PC1,PC2,PC3
PORTC=0b00000000;
DDRC=0b00001111;

//Port D initialization
//pin D2 sebagai interupt 0 menjadi input
PORTD=0b11111011;
DDRD=0x00;

//External Interrput(s) initialization
//INT0 : On
//INT0 Mode : rising Edge
//INT1 : Off
//INT2 : Off

GICR|=0x40;
MCUCR=0x03; ///inisial interupt eksternal 0 dengan deteksi rising edge ////
MCUCSR=0x00;
GIFR=0x40;

//Analog Comparator initialization
//Analog Comparator : Off
//Analog Comparator Input Capture by Timer/Counter 1 : Off
ACSR=0x80;
SFIOR=0x00;

//Global enable interrupts
#asm("sei")

}

//program utama pengiriman data 4 bit

void main()
{
init();
counter=0;
for(;;)
{
```

```
for(;;)
{
//8 data dari baris 1 8x8 data dikirim
if(counter==1) //data 1682
        {PORTC=0b00000110;} ///bit ke 11,10,9, dan 8 dari angka 1682
else if(counter==2)
        {PORTC=0b00001001;} ///bit ke 7,6,5 dan 4 dari angka 1682
else if(counter==3)
        {PORTC=0b00000010;} ///bit ke 3,2,1 dan 0 dari angka 1682
else if(counter==4) //data -9
        {PORTC=0b00001111;}
else if(counter==5)
        {PORTC=0b00001111;}
else if(counter==6)
        {PORTC=0b00000111;}
else if(counter==7)  //data 0
        {PORTC=0b00000000;}
else if(counter==8)
        {PORTC=0b00000000;}
else if(counter==9)
        {PORTC=0b00000000;}
else if(counter==10) //data 0
        {PORTC=0b00000000;}
else if(counter==11)
        {PORTC=0b00000000;}
else if(counter==12)
        {PORTC=0b00000000;}
else if(counter==13)  //data -1
        {PORTC=0b00001111;}
else if(counter==14)
        {PORTC=0b00001111;}
else if(counter==15)
        {PORTC=0b00001111;}
else if(counter==16)  //data 0
        {PORTC=0b00000000;}
else if(counter==17)
        {PORTC=0b00000000;}
else if(counter==18)
        {PORTC=0b00000000;}
else if(counter==19)  //data 0
        {PORTC=0b00000000;}
else if(counter==20)
        {PORTC=0b00000000;}
else if(counter==21)
        {PORTC=0b00000000;}
else if(counter==22)  //data -1
        {PORTC=0b00001111;}
else if(counter==23)
        {PORTC=0b00001111;}
else if(counter==24)
        {PORTC=0b00001111;}
//8 data dari baris 2 8x8 data dikirim
else if(counter==25)  //data 1
        {PORTC=0b00000000;}
else if(counter==26)
        {PORTC=0b00000000;}
else if(counter==27)
        {PORTC=0b00000001;}
else if(counter==28)  //data -1
        {PORTC=0b00001111;}
else if(counter==29)
        {PORTC=0b00001111;}
else if(counter==30)
        {PORTC=0b00001111;}
else if(counter==31) //data -1
        {PORTC=0b00001111;}
else if(counter==32)
        {PORTC=0b00001111;}
```

```
else if(counter==33)
        {PORTC=0b00001111;}
else if(counter==34)  //data -1
        {PORTC=0b00001111;}
else if(counter==35)
        {PORTC=0b00001111;}
else if(counter==36)
        {PORTC=0b00001111;}
else if(counter==37) //data -2
        {PORTC=0b00001111;}
else if(counter==38)
        {PORTC=0b00001111;}
else if(counter==39)
        {PORTC=0b00001111;}
else if(counter==40) //data 0
        {PORTC=0b00001111;}
else if(counter==41)
        {PORTC=0b00001111;}
else if(counter==42)
        {PORTC=0b00001111;}
else if(counter==43)  //data 1
        {PORTC=0b00001111;}
else if(counter==44)
        {PORTC=0b00001111;}
else if(counter==45)
        {PORTC=0b00001111;}
else if(counter==46)  //data 1
        {PORTC=0b00001111;}
else if(counter==47)
        {PORTC=0b00001111;}
else if(counter==48)
        {PORTC=0b00001111;}
//8 data dari baris 3 8x8 data dikirim
else if(counter==49) //data 2
        {PORTC=0b00001111;}
else if(counter==50)
        {PORTC=0b00001111;}
else if(counter==51)
        {PORTC=0b00001111;}
else if(counter==52) //data 0
        {PORTC=0b00001111;}
else if(counter==53)
        {PORTC=0b00001111;}
else if(counter==54)
        {PORTC=0b00001111;}
else if(counter==55)  //data 2
        {PORTC=0b00001111;}
else if(counter==56)
        {PORTC=0b00001111;}
else if(counter==57)
        {PORTC=0b00001111;}
else if(counter==58)  //data -1
        {PORTC=0b00001111;}
else if(counter==59)
        {PORTC=0b00001111;}
else if(counter==60)
        {PORTC=0b00001111;}
else if(counter==61) //data -1
        {PORTC=0b00001111;}
else if(counter==62)
        {PORTC=0b00001111;}
else if(counter==63)
        {PORTC=0b00001111;}
else if(counter==64)  //data 0
        {PORTC=0b00000000;}
else if(counter==65)
        {PORTC=0b00000000;}
else if(counter==66)
```

```
        {PORTC=0b00000000;}
else if(counter==67)  //data 0
        {PORTC=0b00000000;}
else if(counter==68)
        {PORTC=0b00000000;}
else if(counter==69)
        {PORTC=0b00000000;}
else if(counter==70)  //data 1
        {PORTC=0b00000000;}
else if(counter==71)
        {PORTC=0b00000000;}
else if(counter==72)
        {PORTC=0b00000001;}
//8 data dari baris 4 8x8 data dikirim
else if(counter==73) //data -2
        {PORTC=0b00001111;}
else if(counter==74)
        {PORTC=0b00001111;}
else if(counter==75)
        {PORTC=0b00001110;}
else if(counter==76)  //data 0
        {PORTC=0b00000000;}
else if(counter==77)
        {PORTC=0b00000000;}
else if(counter==78)
        {PORTC=0b00000000;}
else if(counter==79)  //data 0
        {PORTC=0b00000000;}
else if(counter==80)
        {PORTC=0b00000000;}
else if(counter==81)
        {PORTC=0b00000000;}
else if(counter==82) //data 1
        {PORTC=0b00000000;}
else if(counter==83)
        {PORTC=0b00000000;}
else if(counter==84)
        {PORTC=0b00000001;}
else if(counter==85)  //data 0
        {PORTC=0b00000000;}
else if(counter==86)
        {PORTC=0b00000000;}
else if(counter==97)
        {PORTC=0b00000000;}
else if(counter==88)  //data 0
        {PORTC=0b00000000;}
else if(counter==89)
        {PORTC=0b00000000;}
else if(counter==90)
        {PORTC=0b00000000;}
else if(counter==91)  //data -1
        {PORTC=0b00001111;}
else if(counter==92)
        {PORTC=0b00001111;}
else if(counter==93)
        {PORTC=0b00001111;}
else if(counter==94)  //data -1
        {PORTC=0b00001111;}
else if(counter==95)
        {PORTC=0b00001111;}
else if(counter==96)
        {PORTC=0b00001111;}
//8 data dari baris 5 8x8 data dikirim
else if(counter==97)  //data -1
        {PORTC=0b00001111;}
else if(counter==98)
        {PORTC=0b00001111;}
else if(counter==99)
```

```
        {PORTC=0b00001111;}
else if(counter==100) //data -2
        {PORTC=0b00001111;}
else if(counter==101)
        {PORTC=0b00001111;}
else if(counter==102)
        {PORTC=0b00001110;}
else if(counter==103)  //data -1
        {PORTC=0b00001111;}
else if(counter==104)
        {PORTC=0b00001111;}
else if(counter==105)
        {PORTC=0b00001111;}
else if(counter==106) //data -2
        {PORTC=0b00001111;}
else if(counter==107)
        {PORTC=0b00001111;}
else if(counter==108)
        {PORTC=0b00001110;}
else if(counter==109) //data 0
        {PORTC=0b00000000;}
else if(counter==110)
        {PORTC=0b00000000;}
else if(counter==111)
        {PORTC=0b00000000;}
else if(counter==112) //data 1
        {PORTC=0b00001111;}
else if(counter==113)
        {PORTC=0b00001111;}
else if(counter==114)
        {PORTC=0b00000001;}
else if(counter==115) //data 0
        {PORTC=0b00000000;}
else if(counter==116)
        {PORTC=0b00000000;}
else if(counter==117)
        {PORTC=0b00000000;}
else if(counter==118) //data 0
        {PORTC=0b00000000;}
else if(counter==119)
        {PORTC=0b00000000;}
else if(counter==120)
        {PORTC=0b00000000;}
//8 data dari baris 6 8x8 data dikirim
else if(counter==121) //data 1
        {PORTC=0b00000000;}
else if(counter==122)
        {PORTC=0b00000000;}
else if(counter==123)
        {PORTC=0b00000001;}
else if(counter==124)  //data -2
        {PORTC=0b00000000;}
else if(counter==125)
        {PORTC=0b00000000;}
else if(counter==126)
        {PORTC=0b00000000;}
else if(counter==127) //data -2
        {PORTC=0b00001111;}
else if(counter==128)
        {PORTC=0b00001111;}
else if(counter==129)
        {PORTC=0b00001110;}
else if(counter==130) //data 1
        {PORTC=0b00000000;}
else if(counter==131)
        {PORTC=0b00000000;}
else if(counter==132)
        {PORTC=0b00000001;}
```

```
else if(counter==133) //data 0
        {PORTC=0b00000000;}
else if(counter==134)
        {PORTC=0b00000000;}
else if(counter==135)
        {PORTC=0b00000000;}
else if(counter==136) //data 1
        {PORTC=0b00000000;}
else if(counter==137)
        {PORTC=0b00000000;}
else if(counter==138)
        {PORTC=0b00000001;}
else if(counter==139) //data 2
        {PORTC=0b00000000;}
else if(counter==140)
        {PORTC=0b00000000;}
else if(counter==141)
        {PORTC=0b00000010;}
else if(counter==142) //data 2
        {PORTC=0b00000000;}
else if(counter==143)
        {PORTC=0b00000000;}
else if(counter==144)
        {PORTC=0b00000010;}
//8 data dari baris 7 8x8 dikirim
else if(counter==145) //data -1
        {PORTC=0b00001111;}
else if(counter==146)
        {PORTC=0b00001111;}
else if(counter==147)
        {PORTC=0b00001111;}
else if(counter==148)  //data 0
        {PORTC=0b00000000;}
else if(counter==149)
        {PORTC=0b00000000;}
else if(counter==150)
        {PORTC=0b00000000;}
else if(counter==151) //data 0
        {PORTC=0b00000000;}
else if(counter==152)
        {PORTC=0b00000000;}
else if(counter==153)
        {PORTC=0b00000000;}
else if(counter==154) //data 0
        {PORTC=0b00000000;}
else if(counter==155)
        {PORTC=0b00000000;}
else if(counter==156)
        {PORTC=0b00000000;}
else if(counter==157) //data -2
        {PORTC=0b00001111;}
else if(counter==158)
        {PORTC=0b00001111;}
else if(counter==159)
        {PORTC=0b00001110;}
else if(counter==160)  //data -1
        {PORTC=0b00001111;}
else if(counter==161)
        {PORTC=0b00001111;}
else if(counter==162)
        {PORTC=0b00001111;}
else if(counter==163) //data -1
        {PORTC=0b00001111;}
else if(counter==164)
        {PORTC=0b00001111;}
else if(counter==165)
        {PORTC=0b00001111;}
else if(counter==166) //data 0
```

```
            {PORTC=0b00000000;}
    else if(counter==167)
            {PORTC=0b00000000;}
    else if(counter==168)
            {PORTC=0b00000000;}
    //8 data dari baris 8 8x8 data dikirim
    else if(counter==169) //data 0
            {PORTC=0b00000000;}
    else if(counter==170)
            {PORTC=0b00000000;}
    else if(counter==171)
            {PORTC=0b00000000;}
    else if(counter==172) //data 0
            {PORTC=0b00000000;}
    else if(counter==173)
            {PORTC=0b00000000;}
    else if(counter==174)
            {PORTC=0b00000000;}
    else if(counter==175) //data 0
            {PORTC=0b00000000;}
    else if(counter==176)
            {PORTC=0b00000000;}
    else if(counter==177)
            {PORTC=0b00000000;}
    else if(counter==178)  //data 0
            {PORTC=0b00000000;}
    else if(counter==179)
            {PORTC=0b00000000;}
    else if(counter==180)
            {PORTC=0b00000000;}
    else if(counter==181)  //data -1
            {PORTC=0b00001111;}
    else if(counter==182)
            {PORTC=0b00001111;}
    else if(counter==183)
            {PORTC=0b00001111;}
    else if(counter==184) //data 0
            {PORTC=0b00000000;}
    else if(counter==185)
            {PORTC=0b00000000;}
    else if(counter==186)
            {PORTC=0b00000000;}
    else if(counter==187) //data 0
            {PORTC=0b00000000;}
    else if(counter==188)
            {PORTC=0b00000000;}
    else if(counter==189)
            {PORTC=0b00000000;}
    else if(counter==190) //data 0
            {PORTC=0b00000000;}
    else if(counter==191)
            {PORTC=0b00000000;}
    else if(counter==192)
            {PORTC=0b00000000;}}}}
```

L.2 Listing Program Top Level Implementasi 2D-IDCT (*top.vhd)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx primitives in this code.

library UNISIM;
use UNISIM.VComponents.all;

entity top is
        port ( clk : in std_logic;
                     rst : in std_logic;
                     div : out std_logic;
                     data_in : in std_logic_vector (3 downto 0);
                     serial : out std_logic_vector (7 downto 0));
end top;

architecture Behavioral of top is
signal reg1,reg2,reg3 : std_logic_vector (3 downto 0);
signal keluar : std_logic_vector (11 downto 0);
signal en,pen : std_logic;
type array_1 is array (0 to 63) of std_logic_vector (11 downto 0);
signal buff : array_1;
signal reg : array_1;
type array_3 is array (0 to 63) of std_logic_vector (7 downto 0);
signal mem : array_3;
signal memo : array_3;
type array_2 is array (0 to 7) of std_logic_vector (11 downto 0);
signal output_1,output_2,reg_1,reg_2 : array_2;
signal w,x,y,z,p,q,r,s,bg : std_logic;
signal sel, hitung : std_logic_vector (2 downto 0);
signal bagi : std_logic_vector (25 downto 0);

--komponen register 12 bit
component regn is
      port ( R : in std_logic_vector (11 downto 0);
                  Rin,clk,rst : in std_logic;
                  Q : out std_logic_vector (11 downto 0));
end component;

--komponen multiplekser
component mux8 is
      port ( p1,p2,p3,p4,p5,p6,p7,p8 : in std_logic_vector (11 downto 0);
                  sel : in std_logic_vector (2 downto 0);
                  keluar : out std_logic_vector (11 downto 0));
end component;

--komponen IDCT
component idct is
      port ( z1,z2,z3,z4,z5,z6,z7,z8 : in std_logic_vector (11 downto 0);
                  clk : in std_logic;
                  count : in std_logic_vector (2 downto 0);
                  idct_1,idct_2,idct_3,idct_4,idct_5,idct_6,idct_7,idct_8   :
out std_logic_vector (11 downto 0));
end component;

--komponen IDCT18
component idct_18 is
      port ( z1,z2,z3,z4,z5,z6,z7,z8 : in std_logic_vector (11 downto 0);
                  clk : in std_logic;
                  count : in std_logic_vector (2 downto 0);
                  idct_1,idct_2,idct_3,idct_4,idct_5,idct_6,idct_7,idct_8   :
out std_logic_vector (11 downto 0));
end component;

--komponen pengontrol sinyal
component ctrl is
      port ( clk : in std_logic;
                  rst : in std_logic;
                  ena : out std_logic;
                  itung : out std_logic_vector (2 downto 0);
                  par : out std_logic;
```

```
                    w,x,y,z,p,q,r,s : out std_logic;
                    tahap : out std_logic_vector (2 downto 0));
end component;

--komponen register 8 bit
component regn8 is
      port ( R : in std_logic_vector (7 downto 0);
                    Rin,clk,rst : in std_logic;
                    Q : out std_logic_vector (7 downto 0));
end component;

begin
--membuat clock lambat untuk pemicu interupt dan clock lambat sistem
lambat : process (clk,rst,bagi)
begin
      if (rst = '1') then
            bagi <= (others => '0');
            elsif (clk'event and clk = '1') then
            bagi <= bagi + 1;
      end if;
end process;
bg <= bagi (25); --clock lambat untuk operasi sistem
div <= not bagi (25); -- clock yang dikeluarkan ke port FPGA untuk pemicu
interrupt

kendali : ctrl port map (bg,rst,en,hitung,pen,w,x,y,z,p,q,r,s,sel);
ambil : process (rst,bg,data_in,reg1,reg2,reg3)
begin
      if (rst = '1') then
                    reg1 <= (others => '0');
                    reg2 <= (others => '0');
                    reg3 <= (others => '0');
                    elsif (bg'event and bg = '1') then
                    reg1 <= data_in;
                    reg2 <= reg1;
                    reg3 <= reg2;
      end if;
keluar <= reg3&reg2&reg1; --membentuk data  12 bit dari data 4 bit
end process;

--register files untuk data masuk, terdiri dari 64 register 12 bit
r0: regn port map (keluar,en,bg,rst,buff(0));
r1: regn port map (buff(0),en,bg,rst,buff(1));
r2: regn port map (buff(1),en,bg,rst,buff(2));
r3: regn port map (buff(2),en,bg,rst,buff(3));
r4: regn port map (buff(3),en,bg,rst,buff(4));
r5: regn port map (buff(4),en,bg,rst,buff(5));
r6: regn port map (buff(5),en,bg,rst,buff(6));
r7: regn port map (buff(6),en,bg,rst,buff(7));

r8: regn port map (buff(7),en,bg,rst,buff(8));
r9: regn port map (buff(8),en,bg,rst,buff(9));
r10: regn port map (buff(9),en,bg,rst,buff(10));
r11: regn port map (buff(10),en,bg,rst,buff(11));
r12: regn port map (buff(11),en,bg,rst,buff(12));
r13: regn port map (buff(12),en,bg,rst,buff(13));
r14: regn port map (buff(13),en,bg,rst,buff(14));
r15: regn port map (buff(14),en,bg,rst,buff(15));

r16: regn port map (buff(15),en,bg,rst,buff(16));
r17: regn port map (buff(16),en,bg,rst,buff(17));
r18: regn port map (buff(17),en,bg,rst,buff(18));
r19: regn port map (buff(18),en,bg,rst,buff(19));
r20: regn port map (buff(19),en,bg,rst,buff(20));
r21: regn port map (buff(20),en,bg,rst,buff(21));
r22: regn port map (buff(21),en,bg,rst,buff(22));
r23: regn port map (buff(22),en,bg,rst,buff(23));
```

```
r24: regn port map (buff(23),en,bg,rst,buff(24));
r25: regn port map (buff(24),en,bg,rst,buff(25));
r26: regn port map (buff(25),en,bg,rst,buff(26));
r27: regn port map (buff(26),en,bg,rst,buff(27));
r28: regn port map (buff(27),en,bg,rst,buff(28));
r29: regn port map (buff(28),en,bg,rst,buff(29));
r30: regn port map (buff(29),en,bg,rst,buff(30));
r31: regn port map (buff(30),en,bg,rst,buff(31));

r32: regn port map (buff(31),en,bg,rst,buff(32));
r33: regn port map (buff(32),en,bg,rst,buff(33));
r34: regn port map (buff(33),en,bg,rst,buff(34));
r35: regn port map (buff(34),en,bg,rst,buff(35));
r36: regn port map (buff(35),en,bg,rst,buff(36));
r37: regn port map (buff(36),en,bg,rst,buff(37));
r38: regn port map (buff(37),en,bg,rst,buff(38));
r39: regn port map (buff(38),en,bg,rst,buff(39));

r40: regn port map (buff(39),en,bg,rst,buff(40));
r41: regn port map (buff(40),en,bg,rst,buff(41));
r42: regn port map (buff(41),en,bg,rst,buff(42));
r43: regn port map (buff(42),en,bg,rst,buff(43));
r44: regn port map (buff(43),en,bg,rst,buff(44));
r45: regn port map (buff(44),en,bg,rst,buff(45));
r46: regn port map (buff(45),en,bg,rst,buff(46));
r47: regn port map (buff(46),en,bg,rst,buff(47));

r48: regn port map (buff(47),en,bg,rst,buff(48));
r49: regn port map (buff(48),en,bg,rst,buff(49));
r50: regn port map (buff(49),en,bg,rst,buff(50));
r51: regn port map (buff(50),en,bg,rst,buff(51));
r52: regn port map (buff(51),en,bg,rst,buff(52));
r53: regn port map (buff(52),en,bg,rst,buff(53));
r54: regn port map (buff(53),en,bg,rst,buff(54));
r55: regn port map (buff(54),en,bg,rst,buff(55));

r56: regn port map (buff(55),en,bg,rst,buff(56));
r57: regn port map (buff(56),en,bg,rst,buff(57));
r58: regn port map (buff(57),en,bg,rst,buff(58));
r59: regn port map (buff(58),en,bg,rst,buff(59));
r60: regn port map (buff(59),en,bg,rst,buff(60));
r61: regn port map (buff(60),en,bg,rst,buff(61));
r62: regn port map (buff(61),en,bg,rst,buff(62));
r63: regn port map (buff(62),en,bg,rst,buff(63));

--seleksi data masuk ke unit IDCT untuk proses baris atau kolom
m1    :    mux8    port    map    (buff(63),    buff(47),    buff(31),    buff(15),
reg(0),reg(2),reg(4),reg(6),sel,output_1(0));
m2    :    mux8    port    map    (buff(62),    buff(46),    buff(30),    buff(14),
reg(8),reg(10),reg(12),reg(14),sel,output_1(1));
m3    :    mux8    port    map    (buff(61),    buff(45),    buff(29),    buff(13),
reg(16),reg(18),reg(20),reg(22),sel,output_1(2));
m4    :    mux8    port    map    (buff(60),    buff(44),    buff(28),    buff(12),
reg(24),reg(26),reg(28),reg(30),sel,output_1(3));
m5    :    mux8    port    map    (buff(59),    buff(43),    buff(27),    buff(11),
reg(32),reg(34),reg(36),reg(38),sel,output_1(4));
m6    :    mux8    port    map    (buff(58),    buff(42),    buff(26),    buff(10),
reg(40),reg(42),reg(44),reg(46),sel,output_1(5));
m7    :    mux8    port    map    (buff(57),    buff(41),    buff(25),    buff(9),
reg(48),reg(50),reg(52),reg(54),sel,output_1(6));
m8    :    mux8    port    map    (buff(56),    buff(40),    buff(24),    buff(8),
reg(56),reg(58),reg(60),reg(62),sel,output_1(7));
m9    :    mux8    port    map    (buff(55),    buff(39),    buff(23),    buff(7),
reg(1),reg(3),reg(5),reg(7),sel,output_2(0));
m10    :    mux8    port    map    (buff(54),    buff(38),    buff(22),    buff(6),
reg(9),reg(11),reg(13),reg(15),sel,output_2(1));
m11    :    mux8    port    map    (buff(53),    buff(37),    buff(21),    buff(5),
reg(17),reg(19),reg(21),reg(23),sel,output_2(2));
```

```
m12    :    mux8    port    map    (buff(52),    buff(36),    buff(20),    buff(4),
reg(25),reg(27),reg(29),reg(31),sel,output_2(3));
m13    :    mux8    port    map    (buff(51),    buff(35),    buff(19),    buff(3),
reg(33),reg(35),reg(37),reg(39),sel,output_2(4));
m14    :    mux8    port    map    (buff(50),    buff(34),    buff(18),    buff(2),
reg(41),reg(43),reg(45),reg(47),sel,output_2(5));
m15    :    mux8    port    map    (buff(49),    buff(33),    buff(17),    buff(1),
reg(49),reg(51),reg(53),reg(55),sel,output_2(6));
m16    :    mux8    port    map    (buff(48),    buff(32),    buff(16),    buff(0),
reg(57),reg(59),reg(61),reg(63),sel,output_2(7));

--operasi IDCT menggunakan 2 blok IDCT
tahap1                :                idct                port                map
(output_1(0),output_1(1),output_1(2),output_1(3),output_1(4),output_1(5),outpu
t_1(6),output_1(7),bg,hitung,reg_1(0),reg_1(1),reg_1(2),reg_1(3),reg_1(4),reg_
1(5),reg_1(6),reg_1(7));
tahap2          :          idct_18          port          map          (output_2(0),
output_2(1),output_2(2),output_2(3),output_2(4),output_2(5),output_2(6),output
_2(7),bg,hitung,reg_2(0),reg_2(1),reg_2(2),reg_2(3),reg_2(4),reg_2(5),reg_2(6)
,reg_2(7));

--register files intermediate value, menyimpan hasil proses 1D-IDCT baris
--row 1
reg0 : regn port map (reg_1(0),w,bg,rst,reg(0));
regs1 : regn port map (reg_1(1),w,bg,rst,reg(1));
regs2 : regn port map (reg_1(2),w,bg,rst,reg(2));
regs3 : regn port map (reg_1(3),w,bg,rst,reg(3));
reg4 : regn port map (reg_1(4),w,bg,rst,reg(4));
reg5 : regn port map (reg_1(5),w,bg,rst,reg(5));
reg6 : regn port map (reg_1(6),w,bg,rst,reg(6));
reg7 : regn port map (reg_1(7),w,bg,rst,reg(7));

--row3
reg16 : regn port map (reg_1(0),x,bg,rst,reg(16));
reg17 : regn port map (reg_1(1),x,bg,rst,reg(17));
reg18 : regn port map (reg_1(2),x,bg,rst,reg(18));
reg19 : regn port map (reg_1(3),x,bg,rst,reg(19));
reg20 : regn port map (reg_1(4),x,bg,rst,reg(20));
reg21 : regn port map (reg_1(5),x,bg,rst,reg(21));
reg22 : regn port map (reg_1(6),x,bg,rst,reg(22));
reg23 : regn port map (reg_1(7),x,bg,rst,reg(23));

--row5
reg32 : regn port map (reg_1(0),y,bg,rst,reg(32));
reg33 : regn port map (reg_1(1),y,bg,rst,reg(33));
reg34 : regn port map (reg_1(2),y,bg,rst,reg(34));
reg35 : regn port map (reg_1(3),y,bg,rst,reg(35));
reg36 : regn port map (reg_1(4),y,bg,rst,reg(36));
reg37 : regn port map (reg_1(5),y,bg,rst,reg(37));
reg38 : regn port map (reg_1(6),y,bg,rst,reg(38));
reg39 : regn port map (reg_1(7),y,bg,rst,reg(39));

--row7
reg48 : regn port map (reg_1(0),z,bg,rst,reg(48));
reg49 : regn port map (reg_1(1),z,bg,rst,reg(49));
reg50 : regn port map (reg_1(2),z,bg,rst,reg(50));
reg51 : regn port map (reg_1(3),z,bg,rst,reg(51));
reg52 : regn port map (reg_1(4),z,bg,rst,reg(52));
reg53 : regn port map (reg_1(5),z,bg,rst,reg(53));
reg54 : regn port map (reg_1(6),z,bg,rst,reg(54));
reg55 : regn port map (reg_1(7),z,bg,rst,reg(55));

--row2
reg8 : regn port map (reg_2(0),w,bg,rst,reg(8));
reg9 : regn port map (reg_2(1),w,bg,rst,reg(9));
reg10 : regn port map (reg_2(2),w,bg,rst,reg(10));
reg11 : regn port map (reg_2(3),w,bg,rst,reg(11));
reg12 : regn port map (reg_2(4),w,bg,rst,reg(12));
```

```
reg13 : regn port map (reg_2(5),w,bg,rst,reg(13));
reg14 : regn port map (reg_2(6),w,bg,rst,reg(14));
reg15 : regn port map (reg_2(7),w,bg,rst,reg(15));

--row4
reg24 : regn port map (reg_2(0),x,bg,rst,reg(24));
reg25 : regn port map (reg_2(1),x,bg,rst,reg(25));
reg26 : regn port map (reg_2(2),x,bg,rst,reg(26));
reg27 : regn port map (reg_2(3),x,bg,rst,reg(27));
reg28 : regn port map (reg_2(4),x,bg,rst,reg(28));
reg29 : regn port map (reg_2(5),x,bg,rst,reg(29));
reg30 : regn port map (reg_2(6),x,bg,rst,reg(30));
reg31 : regn port map (reg_2(7),x,bg,rst,reg(31));

--row6
reg40 : regn port map (reg_2(0),y,bg,rst,reg(40));
reg41 : regn port map (reg_2(1),y,bg,rst,reg(41));
reg42 : regn port map (reg_2(2),y,bg,rst,reg(42));
reg43 : regn port map (reg_2(3),y,bg,rst,reg(43));
reg44 : regn port map (reg_2(4),y,bg,rst,reg(44));
reg45 : regn port map (reg_2(5),y,bg,rst,reg(45));
reg46 : regn port map (reg_2(6),y,bg,rst,reg(46));
reg47 : regn port map (reg_2(7),y,bg,rst,reg(47));

--row8
reg56 : regn port map (reg_2(0),z,bg,rst,reg(56));
reg57 : regn port map (reg_2(1),z,bg,rst,reg(57));
reg58 : regn port map (reg_2(2),z,bg,rst,reg(58));
reg59 : regn port map (reg_2(3),z,bg,rst,reg(59));
reg60 : regn port map (reg_2(4),z,bg,rst,reg(60));
reg61 : regn port map (reg_2(5),z,bg,rst,reg(61));
reg62 : regn port map (reg_2(6),z,bg,rst,reg(62));
reg63 : regn port map (reg_2(7),z,bg,rst,reg(63));

--register files data akhir, menyimpan hasil operasi 1D-IDCT kolom
--column1
mem0 : regn8 port map (reg_1(0) (7 downto 0),p,bg,rst,mem(0));
mem1 : regn8 port map (reg_1(1) (7 downto 0),p,bg,rst,mem(8));
mem2 : regn8 port map (reg_1(2) (7 downto 0),p,bg,rst,mem(16));
mem3 : regn8 port map (reg_1(3) (7 downto 0),p,bg,rst,mem(24));
mem4 : regn8 port map (reg_1(4) (7 downto 0),p,bg,rst,mem(32));
mem5 : regn8 port map (reg_1(5) (7 downto 0),p,bg,rst,mem(40));
mem6 : regn8 port map (reg_1(6) (7 downto 0),p,bg,rst,mem(48));
mem7 : regn8 port map (reg_1(7) (7 downto 0),p,bg,rst,mem(56));

--column2
mem8 : regn8 port map (reg_2(0) (7 downto 0),p,bg,rst,mem(1));
mem9 : regn8 port map (reg_2(1) (7 downto 0),p,bg,rst,mem(9));
mem10 : regn8 port map (reg_2(2) (7 downto 0),p,bg,rst,mem(17));
mem11 : regn8 port map (reg_2(3) (7 downto 0),p,bg,rst,mem(25));
mem12 : regn8 port map (reg_2(4) (7 downto 0),p,bg,rst,mem(33));
mem13 : regn8 port map (reg_2(5) (7 downto 0),p,bg,rst,mem(41));
mem14 : regn8 port map (reg_2(6) (7 downto 0),p,bg,rst,mem(49));
mem15 : regn8 port map (reg_2(7) (7 downto 0),p,bg,rst,mem(57));

--column3
mem16 : regn8 port map (reg_1(0) (7 downto 0),q,bg,rst,mem(2));
mem17 : regn8 port map (reg_1(1) (7 downto 0),q,bg,rst,mem(10));
mem18 : regn8 port map (reg_1(2) (7 downto 0),q,bg,rst,mem(18));
mem19 : regn8 port map (reg_1(3) (7 downto 0),q,bg,rst,mem(26));
mem20 : regn8 port map (reg_1(4) (7 downto 0),q,bg,rst,mem(34));
mem21 : regn8 port map (reg_1(5) (7 downto 0),q,bg,rst,mem(42));
mem22 : regn8 port map (reg_1(6) (7 downto 0),q,bg,rst,mem(50));
mem23 : regn8 port map (reg_1(7) (7 downto 0),q,bg,rst,mem(58));

--column4
mem24 : regn8 port map (reg_2(0) (7 downto 0),q,bg,rst,mem(3));
mem25 : regn8 port map (reg_2(1) (7 downto 0),q,bg,rst,mem(11));
```

```
mem26 : regn8 port map (reg_2(2) (7 downto 0),q,bg,rst,mem(19));
mem27 : regn8 port map (reg_2(3) (7 downto 0),q,bg,rst,mem(27));
mem28 : regn8 port map (reg_2(4) (7 downto 0),q,bg,rst,mem(35));
mem29 : regn8 port map (reg_2(5) (7 downto 0),q,bg,rst,mem(43));
mem30 : regn8 port map (reg_2(6) (7 downto 0),q,bg,rst,mem(51));
mem31 : regn8 port map (reg_2(7) (7 downto 0),q,bg,rst,mem(59));

--column5
mem32 : regn8 port map (reg_1(0) (7 downto 0),r,bg,rst,mem(4));
mem33 : regn8 port map (reg_1(1) (7 downto 0),r,bg,rst,mem(12));
mem34 : regn8 port map (reg_1(2) (7 downto 0),r,bg,rst,mem(20));
mem35 : regn8 port map (reg_1(3) (7 downto 0),r,bg,rst,mem(28));
mem36 : regn8 port map (reg_1(4) (7 downto 0),r,bg,rst,mem(36));
mem37 : regn8 port map (reg_1(5) (7 downto 0),r,bg,rst,mem(44));
mem38 : regn8 port map (reg_1(6) (7 downto 0),r,bg,rst,mem(52));
mem39 : regn8 port map (reg_1(7) (7 downto 0),r,bg,rst,mem(60));

--column6
mem40 : regn8 port map (reg_2(0) (7 downto 0),r,bg,rst,mem(5));
mem41 : regn8 port map (reg_2(1) (7 downto 0),r,bg,rst,mem(13));
mem42 : regn8 port map (reg_2(2) (7 downto 0),r,bg,rst,mem(21));
mem43 : regn8 port map (reg_2(3) (7 downto 0),r,bg,rst,mem(29));
mem44 : regn8 port map (reg_2(4) (7 downto 0),r,bg,rst,mem(37));
mem45 : regn8 port map (reg_2(5) (7 downto 0),r,bg,rst,mem(45));
mem46 : regn8 port map (reg_2(6) (7 downto 0),r,bg,rst,mem(53));
mem47 : regn8 port map (reg_2(7) (7 downto 0),r,bg,rst,mem(61));

--column7
mem48 : regn8 port map (reg_1(0) (7 downto 0),s,bg,rst,mem(6));
mem49 : regn8 port map (reg_1(1) (7 downto 0),s,bg,rst,mem(14));
mem50 : regn8 port map (reg_1(2) (7 downto 0),s,bg,rst,mem(22));
mem51 : regn8 port map (reg_1(3) (7 downto 0),s,bg,rst,mem(30));
mem52 : regn8 port map (reg_1(4) (7 downto 0),s,bg,rst,mem(38));
mem53 : regn8 port map (reg_1(5) (7 downto 0),s,bg,rst,mem(46));
mem54 : regn8 port map (reg_1(6) (7 downto 0),s,bg,rst,mem(54));
mem55 : regn8 port map (reg_1(7) (7 downto 0),s,bg,rst,mem(62));

--column8
mem56 : regn8 port map (reg_2(0) (7 downto 0),s,bg,rst,mem(7));
mem57 : regn8 port map (reg_2(1) (7 downto 0),s,bg,rst,mem(15));
mem58 : regn8 port map (reg_2(2) (7 downto 0),s,bg,rst,mem(23));
mem59 : regn8 port map (reg_2(3) (7 downto 0),s,bg,rst,mem(31));
mem60 : regn8 port map (reg_2(4) (7 downto 0),s,bg,rst,mem(39));
mem61 : regn8 port map (reg_2(5) (7 downto 0),s,bg,rst,mem(47));
mem62 : regn8 port map (reg_2(6) (7 downto 0),s,bg,rst,mem(55));
mem63 : regn8 port map (reg_2(7) (7 downto 0),s,bg,rst,mem(63));

--pengeluaran secara serial 64 data keluaran
keluarkan : process (bg,rst,mem,memo)
begin
if (rst = '1') then
memo(0) <= (others => '0'); memo(32) <= (others => '0');
memo(1) <= (others => '0'); memo(33) <= (others => '0');
memo(2) <= (others => '0'); memo(34) <= (others => '0');
memo(3) <= (others => '0'); memo(35) <= (others => '0');
memo(4) <= (others => '0'); memo(36) <= (others => '0');
memo(5) <= (others => '0'); memo(37) <= (others => '0');
memo(6) <= (others => '0'); memo(38) <= (others => '0');
memo(7) <= (others => '0'); memo(39) <= (others => '0');
memo(8) <= (others => '0'); memo(40) <= (others => '0');
memo(9) <= (others => '0'); memo(41) <= (others => '0');
memo(10) <= (others => '0'); memo(42) <= (others => '0');
memo(11) <= (others => '0'); memo(43) <= (others => '0');
memo(12) <= (others => '0'); memo(44) <= (others => '0');
memo(13) <= (others => '0'); memo(45) <= (others => '0');
memo(14) <= (others => '0'); memo(46) <= (others => '0');
memo(15) <= (others => '0'); memo(47) <= (others => '0');
memo(16) <= (others => '0'); memo(48) <= (others => '0');
```

```
memo(17) <= (others => '0'); memo(49) <= (others => '0');
memo(18) <= (others => '0'); memo(50) <= (others => '0');
memo(19) <= (others => '0'); memo(51) <= (others => '0');
memo(20) <= (others => '0'); memo(52) <= (others => '0');
memo(21) <= (others => '0'); memo(53) <= (others => '0');
memo(22) <= (others => '0'); memo(54) <= (others => '0');
memo(23) <= (others => '0'); memo(55) <= (others => '0');
memo(24) <= (others => '0'); memo(56) <= (others => '0');
memo(25) <= (others => '0'); memo(57) <= (others => '0');
memo(26) <= (others => '0'); memo(58) <= (others => '0');
memo(27) <= (others => '0'); memo(59) <= (others => '0');
memo(28) <= (others => '0'); memo(60) <= (others => '0');
memo(29) <= (others => '0'); memo(61) <= (others => '0');
memo(30) <= (others => '0'); memo(62) <= (others => '0');
memo(31) <= (others => '0'); memo(63) <= (others => '0');
elsif (bg'event and bg = '1') then
if (pen = '0') then
memo(0) <= mem(0); memo(32) <= mem(32);
memo(1) <= mem(1); memo(33) <= mem(33);
memo(2) <= mem(2); memo(34) <= mem(34);
memo(3) <= mem(3); memo(35) <= mem(35);
memo(4) <= mem(4); memo(36) <= mem(36);
memo(5) <= mem(5); memo(37) <= mem(37);
memo(6) <= mem(6); memo(38) <= mem(38);
memo(7) <= mem(7); memo(39) <= mem(39);
memo(8) <= mem(8); memo(40) <= mem(40);
memo(9) <= mem(9); memo(41) <= mem(41);
memo(10) <= mem(10); memo(42) <= mem(42);
memo(11) <= mem(11); memo(43) <= mem(43);
memo(12) <= mem(12); memo(44) <= mem(44);
memo(13) <= mem(13); memo(45) <= mem(45);
memo(14) <= mem(14); memo(46) <= mem(46);
memo(15) <= mem(15); memo(47) <= mem(47);
memo(16) <= mem(16); memo(48) <= mem(48);
memo(17) <= mem(17); memo(49) <= mem(49);
memo(18) <= mem(18); memo(50) <= mem(50);
memo(19) <= mem(19); memo(51) <= mem(51);
memo(20) <= mem(20); memo(52) <= mem(52);
memo(21) <= mem(21); memo(53) <= mem(53);
memo(22) <= mem(22); memo(54) <= mem(54);
memo(23) <= mem(23); memo(55) <= mem(55);
memo(24) <= mem(24); memo(56) <= mem(56);
memo(25) <= mem(25); memo(57) <= mem(57);
memo(26) <= mem(26); memo(58) <= mem(58);
memo(27) <= mem(27); memo(59) <= mem(59);
memo(28) <= mem(28); memo(60) <= mem(60);
memo(29) <= mem(29); memo(61) <= mem(61);
memo(30) <= mem(30); memo(62) <= mem(62);
memo(31) <= mem(31); memo(63) <= mem(63);
else
memo(62) <= memo(63); memo(30) <= mem(31);
memo(61) <= memo(62); memo(29) <= mem(30);
memo(60) <= memo(61); memo(28) <= mem(29);
memo(59) <= memo(60); memo(27) <= mem(28);
memo(58) <= memo(59); memo(26) <= mem(27);
memo(57) <= memo(58); memo(25) <= mem(26);
memo(56) <= memo(57); memo(24) <= mem(25);
memo(55) <= memo(56); memo(23) <= mem(24);
memo(54) <= memo(55); memo(22) <= mem(23);
memo(53) <= memo(54); memo(21) <= mem(22);
memo(52) <= memo(53); memo(20) <= mem(21);
memo(51) <= memo(52); memo(19) <= mem(20);
memo(50) <= memo(51); memo(18) <= mem(19);
memo(49) <= memo(50); memo(17) <= mem(18);
memo(48) <= memo(49); memo(16) <= mem(17);
memo(47) <= memo(48); memo(15) <= mem(16);
memo(46) <= memo(47); memo(14) <= mem(15);
memo(45) <= memo(46); memo(13) <= mem(14);
```

```
memo(44) <= memo(45); memo(12) <= mem(13);
memo(43) <= memo(44); memo(11) <= mem(12);
memo(42) <= memo(43); memo(10) <= mem(11);
memo(41) <= memo(42); memo(9) <= mem(10);
memo(40) <= memo(41); memo(8) <= mem(9);
memo(39) <= memo(40); memo(7) <= mem(8);
memo(38) <= memo(39); memo(6) <= mem(7);
memo(37) <= memo(38); memo(5) <= mem(6);
memo(36) <= memo(37); memo(4) <= mem(5);
memo(35) <= memo(36); memo(3) <= mem(4);
memo(34) <= memo(35); memo(2) <= mem(3);
memo(33) <= memo(34); memo(1) <= mem(2);
memo(32) <= memo(33); memo(0) <= mem(1);
memo(31) <= memo(32); serial <= mem(0);
end if;
end if;
end process;
end Behavioral;
```

L.3 Program *Control Signal* (*ctrl.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ctrl is
            port ( clk : in std_logic;
                        rst : in std_logic;
                        ena : out std_logic;
                        itung : out std_logic_vector (2 downto 0);
                        par : out std_logic;
                        w,x,y,z,p,q,r,s : out std_logic;
                        tahap : out std_logic_vector (2 downto 0));
end ctrl;

architecture Behavioral of ctrl is
signal detek : std_logic_vector (8 downto 0);
signal pen,sip,en,siap,para,c : std_logic;
signal cacah : std_logic_vector (1 downto 0);
signal tambah : std_logic_vector (6 downto 0);
signal hitung : std_logic_vector (2 downto 0);
signal nambah : std_logic_vector (2 downto 0);

begin
cch3: process (clk,rst,detek) --cacah proses keseluruhan
begin
if (rst = '1') then
                    detek <= (others => '0');
                    elsif (clk'event and clk = '1') then
                    if (detek /= "100110000") then --cacah maksimal bernilai
304
                    detek <= detek + 1;
                    else detek <= "000000001";

end if;
end if;
end process;
```

```
--sinyal untuk menandakan komputasi IDCT diaktifkan
proc: process (detek)
begin
if (detek >= "011000101" and detek <= "011101100") then
                    pen <= '1'; --aktif saat cacah proses bernilai 197 sampai
236
                    else
                    pen <= '0';
end if;
end process;
--sinyal enable untuk proses parallel to serial
pararel: process(detek,para)
begin
if (detek >= "011110001" and detek <= "100110000") then
              para <= '1'; --aktif saat cacah proses bernilai 241 sampai 304
      else para <= '0';
end if;
end process;
par <= para;
sip <= detek(8) and detek(5) and detek(4); --aktif saat cacah proses bernilai
304

--cacah data 4 bit, mencacah terus dampai 192 kali data 4 bit masuk
cch: process (clk,rst,cacah,siap,detek)
begin
if (rst = '1' or siap = '1' or detek = "100101011") then
                    cacah <= "00";
                    elsif (clk'event and clk = '1') then
                    if (cacah /= "11") then
                    cacah <= cacah + 1;
                    else cacah <= "01";
end if;
end if;
end process;

en <= cacah(1) and cacah(0); --untuk sinyal enable register files data masuk
ena <= en;

--cacah data untuk masukan 2D-IDCT, mencacah sampai 64 data
cch2: process(rst,clk,tambah,sip)
begin
if (rst = '1' or sip = '1') then
              tambah <= "0000000";
              elsif (clk'event and clk = '1') then
              if (cacah = "11") then
              tambah <= tambah + 1;
end if;
end if;
end process;
siap <= tambah(6) and (not tambah(5)) and (not tambah(4)) and (not tambah(3))
and (not tambah(2)) and (not tambah(1)) and (not tambah(0));
--cacah untuk proses sequential pada komponen IDCT
cch5: process(clk,rst,hitung,pen)
begin
if (rst = '1' or hitung = "101") then
              hitung <= (others => '0');
              elsif (clk'event and clk = '1') then
              if (pen = '1') then
              hitung <= hitung + 1;
end if;
end if;
end process;
itung <= hitung;

--selektor untuk komponen multiplekser 8 to 1
cch4: process (clk,rst,hitung,nambah)
begin
if (rst = '1') then
```

```
            nambah <= "000";
            elsif (clk'event and clk = '1') then
            if (hitung = "100") then
            nambah <= nambah + 1;
end if;
end if;
end process;
tahap <= nambah;
--sinyal enable untuk register files
c <= hitung(1) and hitung(0);
w <= c and (not nambah(2)) and (not nambah(1)) and (not nambah(0));
x <= c and (not nambah(2)) and (not nambah(1)) and nambah(0);
y <= c and (not nambah(2)) and nambah(1) and (not nambah(0));
z <= c and (not nambah(2)) and nambah(1) and nambah(0);
p <= c and nambah(2) and (not nambah(1)) and (not nambah(0));
q <= c and nambah(2) and (not nambah(1)) and nambah(0);
r <= c and nambah(2) and nambah(1) and (not nambah(0));
s <= c and nambah(2) and nambah(1) and nambah(0);

end Behavioral;
```

L.4 Program Register 12bit (*regn.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity regn is
            port ( R : in std_logic_vector (11 downto 0);
                        Rin,clk,rst : in std_logic;
                        Q : out std_logic_vector (11 downto 0));
end regn;

architecture Behavioral of regn is
begin
            process(clk,rst)
            begin
            if (rst = '1') then
            Q <= (others => '0');
            elsif (clk'event and clk = '1') then
            if (Rin = '1') then
            Q <= R;
            end if;
            end if;
            end process;
end Behavioral;
```

L.5 Program Register 8 bit (*regn8.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
```

```
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity regn8 is
            port ( R : in std_logic_vector (7 downto 0);
                        Rin,clk,rst : in std_logic;
                        Q : out std_logic_vector (7 downto 0));
end regn8;

architecture Behavioral of regn8 is
begin
            process(clk,rst)
            begin
            if (rst = '1') then
            Q <= (others => '0');
            elsif (clk'event and clk = '1') then
            if (Rin = '1') then
            Q <= R;
            end if;
            end if;
            end process;

end Behavioral;
```

L.6 Program Multiplekser (*mux8.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux8 is
            port  ( p1,p2,p3,p4,p5,p6,p7,p8  : in std_logic_vector (11 downto
0);
                        sel : in std_logic_vector (2 downto 0);
                        keluar : out std_logic_vector (11 downto 0));
end mux8;

architecture Behavioral of mux8 is

begin
process (sel,p1,p2,p3,p4,p5,p6,p7,p8)
begin
case sel is
when "000" => keluar <= p1;
when "001" => keluar <= p2;
when "010" => keluar <= p3;
when "011" => keluar <= p4;
when "100" => keluar <= p5;
when "101" => keluar <= p6;
when "110" => keluar <= p7;
when others => keluar <= p8;
```

```
end case;
end process;
end Behavioral;
```

L.7 Program IDCT dengan *Embedded Multiplier* (*idct.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity idct_18 is
            port  (  z1,z2,z3,z4,z5,z6,z7,z8  :  in  std_logic_vector  (11  downto
0);
                     clk : in std_logic;
                     count : in std_logic_vector (2 downto 0);
                     idct_1,idct_2,idct_3,idct_4,idct_5,idct_6,idct_7,idct_8  :
out std_logic_vector (11 downto 0));
end idct_18;

architecture Behavioral of idct_18 is
type tahapan is array (0 to 7) of std_logic_vector (11 downto 0);
signal p : tahapan;
type cuplik is array (0 to 7) of std_logic_vector (21 downto 0);
signal d,e,f : cuplik;

signal g_0,g_1,g_2,g_3,g_4,g_5,g_6,g_7 : std_logic_vector (21 downto 0);
signal a0,a1,a2,a3,a4,a5,a6,a7,a8,a9 : std_logic_vector (11 downto 0);
type mult_2 is array (0 to 13) of std_logic_vector (17 downto 0);
signal areg : mult_2;
type mult_3 is array (0 to 13) of std_logic_vector (35 downto 0);
signal silang : mult_3;
constant m0 : std_logic_vector (17 downto 0) := "000000000101101010";
constant m1 : std_logic_vector (17 downto 0) := "111111110101100011";
constant m2 : std_logic_vector (17 downto 0) := "000000000100010101";
constant m3 : std_logic_vector (17 downto 0) := "000000000011000100";
constant m4 : std_logic_vector (17 downto 0) := "000000000001101100";
constant m5 : std_logic_vector (17 downto 0) := "000000001011100111";
constant m6 : std_logic_vector (17 downto 0) := "000000010001011000";
constant m7 : std_logic_vector (17 downto 0) := "000000001000100000";
constant m8 : std_logic_vector (17 downto 0) := "111111111010111010";
constant m9 : std_logic_vector (17 downto 0) := "111111110001100000";
constant m10 : std_logic_vector (17 downto 0) := "111111110100111010";
constant m11 : std_logic_vector (17 downto 0) := "111111111101110011";
constant m12 : std_logic_vector (17 downto 0) := "000000000110101010";

--counter proses
begin
--ambil 8 data masukan
process (clk,count,p,z1,z2,z3,z4,z5,z6,z7,z8)
begin
if (clk'event and clk = '1') then
if (count = "001") then
p(0) <= z1;
p(1) <= z2;
p(2) <= z3;
```

```
p(3) <= z4;
p(4) <= z5;
p(5) <= z6;
p(6) <= z7;
p(7) <= z8;
end if;
end if;
end process;
--step 1 algoritma LLM
a0 <= p(0); a1 <= p(4); a2 <=p(6); a3 <= p(2);
a4 <= p(6) + p(2);
a5 <= p(7) + p(1);
a6 <= p(5) + p(3);
a7 <= p(7) + p(3);
a8 <= p(1) + p(5);
a9 <= p(7) + p(3) + p(1) + p(5);

--membuat data step 1 menjadi 18 bit
areg(0) <= a0(11)&a0(11)&a0(11)&a0(11)&a0(11)&a0(11)&a0;
areg(1) <= a1(11)&a1(11)&a1(11)&a1(11)&a1(11)&a1(11)&a1;
areg(2) <= a2(11)&a2(11)&a2(11)&a2(11)&a2(11)&a2(11)&a2;
areg(3) <= a3(11)&a3(11)&a3(11)&a3(11)&a3(11)&a3(11)&a3;
areg(4) <= a4(11)&a4(11)&a4(11)&a4(11)&a4(11)&a4(11)&a4;
areg(5) <= p(7)(11)&p(7)(11)&p(7)(11)&p(7)(11)&p(7)(11)&p(7)(11)&p(7);
areg(6) <= p(5)(11)&p(5)(11)&p(5)(11)&p(5)(11)&p(5)(11)&p(5)(11)&p(5);
areg(7) <= p(3)(11)&p(3)(11)&p(3)(11)&p(3)(11)&p(3)(11)&p(3)(11)&p(3);
areg(8) <= p(1)(11)&p(1)(11)&p(1)(11)&p(1)(11)&p(1)(11)&p(1)(11)&p(1);
areg(9) <= a5(11)&a5(11)&a5(11)&a5(11)&a5(11)&a5(11)&a5;
areg(10) <= a6(11)&a6(11)&a6(11)&a6(11)&a6(11)&a6(11)&a6;
areg(11) <= a7(11)&a7(11)&a7(11)&a7(11)&a7(11)&a7(11)&a7;
areg(12) <= a8(11)&a8(11)&a8(11)&a8(11)&a8(11)&a8(11)&a8;
areg(13) <= a9(11)&a9(11)&a9(11)&a9(11)&a9(11)&a9(11)&a9;

--step 2 algoritma LLM dengan embedded multiplier
MULT18X18_inst_1 : MULT18X18  --m0 * a0
      port map (A => areg(0),          --18 bit multiplier input
                B => m0,               --18 bit multiplier input
                   P => silang(0));   --36 bit multiplier output

MULT18X18_inst_2 : MULT18X18  --m0 * a1
      port map (A => areg(1),          --18 bit multiplier input
                B => m0,               --18 bit multiplier input
                   P => silang(1));   --36 bit multiplier output

MULT18X18_inst_3 : MULT18X18  --m1 * a2
      port map (A => areg(2),          --18 bit multiplier input
                B => m1,               --18 bit multiplier input
                   P => silang(2));   --36 bit multiplier output

MULT18X18_inst_4 : MULT18X18  --m2 * a3
      port map (A => areg(3),          --18 bit multiplier input
                B => m2,               --18 bit multiplier input
                   P => silang(3));   --36 bit multiplier output

MULT18X18_inst_5 : MULT18X18  --m3 * a4
      port map (A => areg(4),          --18 bit multiplier input
                B => m3,               --18 bit multiplier input
                   P => silang(4));   --36 bit multiplier output

MULT18X18_inst_6 : MULT18X18  --ma * p7
      port map (A => areg(5),          --18 bit multiplier input
                B => m4,               --18 bit multiplier input
                   P => silang(5));   --36 bit multiplier output

MULT18X18_inst_7 : MULT18X18  --mc * p5
      port map (A => areg(6),          --18 bit multiplier input
                B => m5,               --18 bit multiplier input
                   P => silang(6));   --36 bit multiplier output
```

```
MULT18X18_inst_8 : MULT18X18  --mc * p3
      port map (A => areg(7),          --18 bit multiplier input
                B => m6,               --18 bit multiplier input
                  P => silang(7));   --36 bit multiplier output

MULT18X18_inst_9 : MULT18X18  --md * p1
      port map (A => areg(8),          --18 bit multiplier input
                B => m7,               --18 bit multiplier input
                  P => silang(8));   --36 bit multiplier output

MULT18X18_inst_10 : MULT18X18  --me * a5
      port map (A => areg(9),          --18 bit multiplier input
                B => m8,               --18 bit multiplier input
                  P => silang(9));   --36 bit multiplier output

MULT18X18_inst_11 : MULT18X18  --mf * a6
      port map (A => areg(10),  --18 bit multiplier input
                B => m9,               --18 bit multiplier input
                   P => silang(10));--36 bit multiplier output

MULT18X18_inst_12 : MULT18X18  --mg * a7
      port map (A => areg(11),  --18 bit multiplier input
                B => m10,              --18 bit multiplier input
                   P => silang(11));--36 bit multiplier output

MULT18X18_inst_13 : MULT18X18  --mh * a8
      port map (A => areg(12),  --18 bit multiplier input
                B => m11,              --18 bit multiplier input
                  P => silang(12));--36 bit multiplier output

MULT18X18_inst_14 : MULT18X18  --mi * a9
      port map (A => areg(13),  --18 bit multiplier input
                B => m12,              --18 bit multiplier input
                  P => silang(13));--36 bit multiplier output

--step 3 algoritma LLM
d(0) <= silang(0) (21 downto 0);       d(4)  <=  silang(5)(21  downto  0)  +
silang(11)(21 downto 0) + silang(13)(21 downto 0);
d(1) <= silang(1) (21 downto 0);       d(5)  <=  silang(6)(21  downto  0)  +
silang(12)(21 downto 0) + silang(13)(21 downto 0);
d(2) <= silang(2) (21 downto 0);       d(6)  <=  silang(7)(21  downto  0)  +
silang(11)(21 downto 0) + silang(13)(21 downto 0);
d(3) <= silang(3) (21 downto 0);       d(7)  <=  silang(8)(21  downto  0)  +
silang(12)(21 downto 0) + silang(13)(21 downto 0);

--step 4 algoritma LLM
e(0) <= d(0) + d(1);                                          e(4)  <=  d(4)
+ silang(9)(21 downto 0);
e(1) <= d(0) - d(1);                                          e(5)  <=  d(5)
+ silang(10)(21 downto 0);
e(2) <= d(2) + silang(4)(21 downto 0); e(6) <= d(6) + silang(10)(21 downto 0);
e(3) <= d(3) + silang(4)(21 downto 0); e(7) <= d(7) + silang(9)(21 downto 0);

--step 5 algoritma LLM
                  f(0) <= e(0) + e(3);                  f(4) <= e(4);
                  f(1) <= e(1) + e(2);                  f(5) <= e(5);
                  f(2) <= e(1) - e(2);                  f(6) <= e(6);
                  f(3) <= e(0) - e(3);                  f(7) <= e(7);

--step 6 algoritma LLM
                  g_0 <= f(0) + f(7);           g_4 <= f(3) - f(4);
                  g_1 <= f(1) + f(6);           g_5 <= f(2) - f(5);
                  g_2 <= f(2) + f(5);           g_6 <= f(1) - f(6);
                  g_3 <= f(3) + f(4);           g_7 <= f(0) - f(7);

--keluarkan 8 data valid hasil komputasi IDCT
process(clk,count,g_0,g_1,g_2,g_3,g_4,g_5,g_6,g_7)
```

```
begin
if (clk'event and clk = '1') then
if (count = "010") then
                    idct_1 <= g_0(21 downto 10);
                    idct_2 <= g_1(21 downto 10);
                    idct_3 <= g_2(21 downto 10);
                    idct_4 <= g_3(21 downto 10);
                    idct_5 <= g_4(21 downto 10);
                    idct_6 <= g_5(21 downto 10);
                    idct_7 <= g_6(21 downto 10);
                    idct_8 <= g_7(21 downto 10);
end if;
end if;
end process;
end Behavioral;
```

L.8 Program IDCT dengan desain pengali (*idct.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity idct is
      Port ( z1,z2,z3,z4,z5,z6,z7,z8 : in std_logic_vector (11 downto 0);
                    clk : in std_logic;
                    count : in std_logic_vector (2 downto 0);
                    idct_1,idct_2,idct_3,idct_4,idct_5,idct_6,idct_7,idct_8  :
out std_logic_vector (11 downto 0));
end idct;

architecture Behavioral of idct is
type tahapan is array (0 to 7) of std_logic_vector (11 downto 0);
signal p : tahapan;
type cuplik is array (0 to 7) of std_logic_vector (11 downto 0);
signal d,e,f : cuplik;
signal g_0,g_1,g_2,g_3,g_4,g_5,g_6,g_7 : std_logic_vector (11 downto 0);
signal  a0,a1,a2,a3,a4,a5,a6,a7,a8,a9  :  std_logic_vector  (11  downto  0);  --
permuter
type multi_1 is array (0 to 13) of std_logic_vector (11 downto 0);
signal silang : multi_1;

component pengali_1 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_2 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_3 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_4 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
```

```
end component;
component pengali_5 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_6 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_7 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_8 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_9 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_10 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_11 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_12 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component pengali_13 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
             m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;

--ambil 8 data masukan untuk IDCT
begin
masuk : process(clk,count,p,z1,z2,z3,z4,z5,z6,z7,z8)
begin
if (clk'event and clk = '1') then
if (count = "001") then
p(0) <= z1;
p(1) <= z2;
p(2) <= z3;
p(3) <= z4;
p(4) <= z5;
p(5) <= z6;
p(6) <= z7;
p(7) <= z8;
end if;
end if;
end process;

--step 1 algoritma LLM
a0 <= p(0); a1 <= p(4); a2 <=p(6); a3 <= p(2);
a4 <= p(6) + p(2);
a5 <= p(7) + p(1);
a6 <= p(5) + p(3);
a7 <= p(7) + p(3);
a8 <= p(1) + p(5);
a9 <= p(7) + p(3) + p(1) + p(5);

--step 2 algoritma LLM dengan pengali buatan
t1 : pengali_1 port map (a0, silang (0));
t2 : pengali_1 port map (a1, silang (1));
```

```
t3 : pengali_2 port map (a2, silang (2));
t4 : pengali_3 port map (a3, silang (3));
t5 : pengali_4 port map (a4, silang (4));
t6 : pengali_5 port map (p(7), silang (5));
t7 : pengali_6 port map (p(5), silang (6));
t8 : pengali_7 port map (p(3), silang (7));
t9 : pengali_8 port map (p(1), silang (8));
t10 : pengali_9 port map (a5, silang (9));
t11 : pengali_10 port map (a6, silang (10));
t12 : pengali_11 port map (a7, silang (11));
t13 : pengali_12 port map (a8, silang (12));
t14 : pengali_13 port map (a9, silang (13));

--step 3 algortima LLM
d(0) <= silang(0);          d(4) <= silang(5) + silang(11) + silang(13);
d(1) <= silang(1);      d(5) <= silang(6) + silang(12) + silang(13);
d(2) <= silang(2);      d(6) <= silang(7) + silang(11) + silang(13);
d(3) <= silang(3);      d(7) <= silang(8) + silang(12) + silang(13);

--step 4 algoritma LLM
e(0) <= d(0) + d(1);                              e(4) <= d(4) + silang(9);
e(1) <= d(0) - d(1);                              e(5) <= d(5) + silang(10);
e(2) <= d(2) + silang(4);          e(6) <= d(6) + silang(10);
e(3) <= d(3) + silang(4);          e(7) <= d(7) + silang(9);

--step 5 algoritma LLM
f(0) <= e(0) + e(3);                              f(4) <= e(4);
f(1) <= e(1) + e(2);                              f(5) <= e(5);
f(2) <= e(1) - e(2);                              f(6) <= e(6);
f(3) <= e(0) - e(3);                              f(7) <= e(7);

--step 6 algoritma LLM
g_0 <= f(0) + f(7);                          g_4 <= f(3) - f(4);
g_1 <= f(1) + f(6);                          g_5 <= f(2) - f(5);
g_2 <= f(2) + f(5);                          g_6 <= f(1) - f(6);
g_3 <= f(3) + f(4);                          g_7 <= f(0) - f(7);

--keluarkan 8 data hasil komputasi IDCT
keluar: process (clk,count,g_0,g_1,g_2,g_3,g_4,g_5,g_6,g_7)
begin
if (clk'event and clk = '1') then
if (count = "0010") then
      idct_1 <= g_0;
      idct_2 <= g_1;
      idct_3 <= g_2;
      idct_4 <= g_3;
      idct_5 <= g_4;
      idct_6 <= g_5;
      idct_7 <= g_6;
      idct_8 <= g_7;

end if;
end if;
end process;
end Behavioral;
```

L.9 Program pengali 1 (*pengali_1.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_1 is
      port ( a : in STD_LOGIC_VECTOR (11 DOWNTO 0);
                  m_1 : out STD_LOGIC_VECTOR (11 DOWNTO 0));
end pengali_1;

architecture Behavioral of pengali_1 is
      signal temp_1 : STD_LOGIC_VECTOR (17 DOWNTO 0);
      signal a_inv : STD_LOGIC_VECTOR (11 DOWNTO 0);
      signal s_1 : STD_LOGIC_VECTOR (13 DOWNTO 0);
      signal s_2 : STD_LOGIC_VECTOR (16 DOWNTO 0);
      signal s_3 : STD_LOGIC_VECTOR (17 DOWNTO 0);

      signal c_1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
      signal c_2 : STD_LOGIC_VECTOR (17 DOWNTO 0);
      signal c_3 : STD_LOGIC_VECTOR (18 DOWNTO 0);
      signal p,c_m : STD_LOGIC_VECTOR (17 DOWNTO 0);
      signal g : STD_LOGIC_VECTOR (16 DOWNTO 0);
      signal c_4 : STD_LOGIC;

begin
a_inv <= not(a(11 downto 0));
--stage1
s_1(0) <= a_inv(1) xor a_inv(0) xor '1';
s_1(1) <= a_inv(2) xor a_inv(1);
s_1(10 downto 2) <= (a_inv (11 downto 3) xor a_inv (10 downto 2)) xor a_inv (8
downto 0);
s_1(11) <= (a_inv(11) xor a_inv(11)) xor a_inv(9);
s_1(12) <= (a_inv(11) xor a_inv(11)) xor a_inv(10);
s_1(13) <= (a_inv(11) xor a_inv(11)) xor a_inv(11);

c_1(0) <= a_inv(0);
c_1(1) <= (a_inv(1) and a_inv(0)) or (a_inv(1) xor a_inv(0));
c_1(2) <= a_inv(2) and a_inv(1);
c_1(11 downto 3) <= (a_inv(11 downto 3) and a_inv(10 downto 2)) or (a_inv(8
downto 0) and (a(11 downto 3) xor a(10 downto 2)));
c_1(12)  <=  (a_inv(11)  and  a_inv(11))  or  (a_inv(9)  and  (a_inv(11)  xor
a_inv(11)));
c_1(13)  <=  (a_inv(11)  and  a_inv(11))  or  (a_inv(10)  and  (a_inv(11)  xor
a_inv(11)));
c_1(14)  <=  (a_inv(11)  and  a_inv(11))  or  (a_inv(11)  and  (a_inv(11)  xor
a_inv(11)));

--stage2
s_2(0) <= s_1(0) xor c_1(0);
s_2(1) <= s_1(1) xor c_1(1);
s_2(2) <= s_1(2) xor c_1(2) xor '1';
s_2(4 downto 3) <= s_1 (4 downto 3) xor c_1(4 downto 3);
s_2(13 downto 5) <= (s_1(13 downto 5) xor c_1(13 downto 5)) xor a(8 downto 0);
s_2(14) <= (s_1(13) xor c_1(14)) xor a(9);
s_2(15) <= (s_1(13) xor c_1(14)) xor a(10);
s_2(16) <= (s_1(13) xor c_1(14)) xor a(11);
c_2(0) <= '0';
c_2(2 downto 1) <= s_1(1 downto 0) and c_1(1 downto 0);
c_2(3) <= (s_1(2) and c_1(2)) or (s_1(2) xor c_1(2));
c_2(4) <= s_1(3) and c_1(3);
c_2(5) <= s_1(4) and c_1(4);
c_2(14 downto 6) <= (s_1(13 downto 5) and c_1(13 downto 5)) or (c_1(13 downto
5) and a(8 downto 0)) or (s_1(13 downto 5) and a(8 downto 0));
c_2(15) <= (s_1(13) and c_1(14)) or (c_1(14) and a(9)) or (s_1(13) and a(9));
c_2(16)  <=  (s_1(13)  and  c_1(14))  or  (c_1(14)  and  a(10))  or  (s_1(13)  and
a(10));
```

```
c_2(17) <= (s_1(13) and c_1(14)) or (c_1(14) and a(11)) or (s_1(13) and
a(11));

--stage3
s_3(5 downto 0) <= s_2(5 downto 0) xor c_2(5 downto 0);
s_3(16 downto 6) <= (s_2(16 downto 6) xor c_2(16 downto 6)) xor a(10 downto
0);
s_3(17) <= (s_2(16) xor c_2(17)) xor a(11);

c_3(0) <= '0';
c_3(6 downto 1) <= s_2(5 downto 0) and c_2(5 downto 0);
c_3(17 downto 7) <= (s_2(16 downto 6) and c_2(16 downto 6)) or (a(10 downto 0)
and (s_2(16 downto 6) xor c_2(16 downto 6)));
c_3(18) <= (s_2(16) and c_2(17)) or (a(11) and (s_2(16) xor c_2(17)));

c_4 <= (s_3(1) and c_3(1)) or ((s_3(1) xor c_3(1)) and (s_3(0) and c_3(0)));
p(15 downto 0) <= s_3(17 downto 2) xor c_3(17 downto 2);
p(16) <= s_3(17) xor c_3(18);
p(17) <= s_3(17) xor c_3(18);
g(15 downto 0) <= s_3(17 downto 2) and c_3(17 downto 2);
g(16) <= s_3(17) and c_3(18);
c_m(0) <= c_4;

process (p,g,c_m,c_4)
begin
c_m(0) <= c_4;
            c_m(1) <= g(0) or (c_m(0) and p(0));
            inst : for i in 1 to 16 loop
            c_m(i + 1) <= g(i) or (p(i) and c_m(i));
            end loop;
end process;
temp_1 <= p(17 downto 0) xor c_m(17 downto 0);
det : process (temp_1)
begin
      if (temp_1 = "11111111111111111") then
            m_1 <= not temp_1(17 downto 6);
            else
            m_1 <= temp_1(17 downto 6);
            end if;
            end process;

end Behavioral;
```

L.10 Program pengali 2 (*pengali_2.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_2 is
      port ( a : in STD_LOGIC_VECTOR (11 DOWNTO 0);
                  m_1 : out STD_LOGIC_VECTOR (11 DOWNTO 0));
end pengali_2;

architecture Behavioral of pengali_2 is
      signal a_inv : STD_LOGIC_VECTOR (11 DOWNTO 0);
```

```
        signal s_1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
        signal s_2 : STD_LOGIC_VECTOR (12 DOWNTO 0);
        signal s_3 : STD_LOGIC_VECTOR (12 DOWNTO 0);
        signal s_4 : STD_LOGIC_VECTOR (13 DOWNTO 0);


        signal c_1 : STD_LOGIC_VECTOR (15 DOWNTO 0);
        signal c_2 : STD_LOGIC_VECTOR (13 DOWNTO 0);
        signal c_3 : STD_LOGIC_VECTOR (13 DOWNTO 0);
        signal c_4 : STD_LOGIC_VECTOR (13 DOWNTO 0);
        signal p,c_m,temp_1 : STD_LOGIC_VECTOR (17 DOWNTO 0);
        signal g : STD_LOGIC_VECTOR (16 DOWNTO 0);
        signal c_5 : STD_LOGIC;
        signal z_2,v_2 : STD_LOGIC_VECTOR (3 DOWNTO 0);
        signal z_3,v_3 : STD_LOGIC_VECTOR (4 DOWNTO 0);
        signal z_4,v_4 : STD_LOGIC_VECTOR (5 DOWNTO 0);

begin
a_inv <= not(a);
--stage1
s_1(0) <= a(2) xor a_inv(0) xor '1';
s_1(2 downto 1) <= a(4 downto 3) xor a_inv(2 downto 1);
s_1(9 downto 3) <= a(11 downto 5) xor a_inv (9 downto 3) xor a(6 downto 0);
s_1(10) <= a(11) xor a_inv(10) xor a(7);
s_1(11) <= a(11) xor a_inv(11) xor a(8);
s_1(12) <= a(11) xor a_inv(11) xor a(9);
s_1(13) <= a(11) xor a_inv(11) xor a(10);
s_1(14) <= a(11) xor a_inv(11) xor a(11);


c_1(0) <= a(1);
c_1(1) <= (a(2) and a_inv(0)) or (a(2) xor a_inv(0));
c_1(3 downto 2) <= a(4 downto 3) and a_inv(2 downto 1);
c_1(10 downto 4) <= (a(11 downto 5) and a_inv(9 downto 3)) or (a(6 downto 0)
and (a(11 downto 5) xor a_inv(9 downto 3)));
c_1(11) <= (a(11) and a_inv(10)) or (a(7) and (a(11) xor a_inv(10)));
c_1(12) <= (a(11) and a_inv(11)) or (a(8) and (a(11) xor a_inv(11)));
c_1(13) <= (a(11) and a_inv(11)) or (a(9) and (a(11) xor a_inv(11)));
c_1(14) <= (a(11) and a_inv(11)) or (a(10) and (a(11) xor a_inv(11)));
c_1(15) <= (a(11) and a_inv(11)) or (a(11) and (a(11) xor a_inv(11)));

--stage2
z_2(3 downto 0) <= s_1(3 downto 0) xor c_1(3 downto 0);
s_2(0) <= s_1(4) xor c_1(4);
s_2(10 downto 1) <= s_1(14 downto 5) xor c_1(14 downto 5) xor a_inv(9 downto
0);
s_2(11) <= s_1(14) xor c_1(15) xor a_inv(10);
s_2(12) <= s_1(14) xor c_1(15) xor a_inv(11);


v_2(0) <= '0';
v_2(3 downto 1) <= s_1(2 downto 0) and c_1(2 downto 0);
c_2(1 downto 0) <= s_1(4 downto 3) and c_1(4 downto 3);
c_2(11 downto 2) <= (s_1(14 downto 5) and c_1(14 downto 5)) or (a_inv(9 downto
0) and (s_1(14 downto 5) xor c_1(14 downto 5)));
c_2(12) <= (s_1(14) and c_1(15)) or (a_inv(10) and (s_1(14) xor c_1(15)));
c_2(13) <= (s_1(14) and c_1(15)) or (a_inv(11) and (s_1(14) xor c_1(15)));

--stage3
z_3(3 downto 0) <= z_2(3 downto 0) xor v_2(3 downto 0);
z_3(4) <= s_2(0) xor c_2(0);
s_3(0) <= s_2(1) xor c_2(1) xor '1';
s_3(11 downto 1) <= s_2(12 downto 2) xor c_2(12 downto 2) xor a_inv(10 downto
0);
s_3(12) <= s_2(12) xor c_2(13) xor a_inv(11);

v_3(0) <= '0';
v_3(4 downto 1) <= z_2(3 downto 0) and v_2(3 downto 0);
```

```
c_3(0) <= s_2(0) and c_2(0);
c_3(1) <= (s_2(1) and c_2(1)) or (s_2(1) xor c_2(1));
c_3(12 downto 2) <= (s_2(12 downto 2) and c_2(12 downto 2)) or (a_inv(10
downto 0) and (s_2(12 downto 2) xor c_2(12 downto 2)));
c_3(13) <= (s_2(12) and c_2(13)) or (a_inv(11) and (s_2(12) xor c_2(13)));

--stage4
z_4(4 downto 0) <= z_3(4 downto 0) xor v_3(4 downto 0);
z_4(5) <= s_3(0) xor c_3(0);
s_4(0) <= s_3(1) xor c_3(1) xor '1';
s_4(1) <= s_3(2) xor c_3(2);
s_4(11 downto 2) <= s_3(12 downto 3) xor c_3(12 downto 3) xor  a(9 downto 0);
s_4(12) <= s_3(12) xor c_3(13) xor a(10);
s_4(13) <= s_3(12) xor c_3(13) xor a(11);

v_4(0) <= '0';
v_4(5 downto 1) <= z_3(4 downto 0) and v_3(4 downto 0);
c_4(0) <= s_3(0) and c_3(0);
c_4(1) <= (s_3(1) and c_3(1)) or (s_3(1) xor c_3(1));
c_4(2) <= s_3(2) and c_3(2);
c_4(12 downto 3) <= (s_3(12 downto 3) and c_3(12 downto 3)) or (a(9 downto 0)
and (s_3(12 downto 3) xor c_3(12 downto 3)));
c_4(13) <= (s_3(12) and c_3(13)) or (a(10) and (s_3(12) xor c_3(13)));

c_5 <= (z_4(1) and v_4(1)) or ((z_4(1) xor v_4(1)) and (z_4(0) and v_4(0)));
p(3 downto 0) <= z_4(5 downto 2) xor v_4(5 downto 2);
p(17 downto 4) <= s_4(13 downto 0) xor c_4(13 downto 0);
g(3 downto 0) <= z_4(5 downto 2) and v_4(5 downto 2);
g(16 downto 4) <= s_4(12 downto 0) and c_4(12 downto 0);
c_m(0) <= c_5;

process (p,g,c_m,c_5)
begin
c_m(0) <= c_5;
             c_m(1) <= g(0) or (c_m(0) and p(0));
             inst : for i in 1 to 16 loop
             c_m(i + 1) <= g(i) or (p(i) and c_m(i));
             end loop;
end process;
temp_1 <= p(17 downto 0) xor c_m(17 downto 0);
det : process (temp_1)
begin
      if (temp_1 = "11111111111111111") then
             m_1 <= not temp_1(17 downto 6);
             else
             m_1 <= not temp_1(17 downto 6) + 1;
             end if;
             end process;

end Behavioral;
```

L.11 Program pengali 3 (*pengali_3.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity pengali_3 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end pengali_3;

architecture Behavioral of pengali_3 is
signal s_1 : STD_LOGIC_VECTOR(13 DOWNTO 4);
signal s_2 : STD_LOGIC_VECTOR(14 DOWNTO 3);
signal c_1 : STD_LOGIC_VECTOR(14 DOWNTO 4);
signal c_2 : STD_LOGIC_VECTOR(15 DOWNTO 3);
signal p,c_m : STD_LOGIC_VECTOR(11 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (10 DOWNTO 0);
signal c_3 : STD_LOGIC;

begin
--stage1
s_1(9 downto 4) <= a(11 downto 6) xor a(9 downto 4) xor a(7 downto 2);
s_1(10) <= a(11) xor a(10) xor a(8);
s_1(11) <= a(11) xor a(11) xor a(9);
s_1(12) <= a(11) xor a(11) xor a(10);
s_1(13) <= a(11) xor a(11) xor a(11);

c_1(10 downto 4) <= (a(11 downto 5) and a(9 downto 3)) or (a(7 downto 1) and
(a(11 downto 5) xor a(9 downto 3)));
c_1(11) <= (a(11) and a(10)) or (a(8) and (a(11) xor a(10)));
c_1(12) <= (a(11) and a(11)) or (a(9) and (a(11) xor a(11)));
c_1(13) <= (a(11) and a(11)) or (a(10) and (a(11) xor a(11)));
c_1(14) <= (a(11) and a(11)) or (a(11) and (a(11) xor a(11)));

--stage2
s_2(10 downto 3) <= s_1(13 downto 6) xor c_1(13 downto 6) xor a(7 downto 0);
s_2(11) <= s_1(13) xor c_1(14) xor a(8);
s_2(12) <= s_1(13) xor c_1(14) xor a(9);
s_2(13) <= s_1(13) xor c_1(14) xor a(10);
s_2(14) <= s_1(13) xor c_1(14) xor a(11);

c_2(3) <= (s_1(5) and c_1(5)) or ((s_1(5) xor c_1(5)) and (s_1(4) and
c_1(4)));
c_2(11 downto 4) <= (s_1(13 downto 6) and c_1(13 downto 6)) or (c_1(13 downto
6) and a(7 downto 0)) or (s_1(13 downto 6) and a(7 downto 0));
c_2(12) <= (s_1(13) and c_1(14)) or (a(8) and (s_1(13) xor c_1(14)));
c_2(13) <= (s_1(13) and c_1(14)) or (a(9) and (s_1(13) xor c_1(14)));
c_2(14) <= (s_1(13) and c_1(14)) or (a(10) and (s_1(13) xor c_1(14)));
c_2(15) <= (s_1(13) and c_1(14)) or (a(11) and (s_1(13) xor c_1(14)));

c_3 <= (s_2(4) and c_2(4)) or ((s_2(4) xor c_2(4)) and (s_2(3) and c_2(3)));
p(9 downto 0) <= s_2(14 downto 5) xor c_2(14 downto 5);
p(10) <= s_2(14) xor c_2(15);
p(11) <= s_2(14) xor c_2(15);
g(9 downto 0) <= s_2(14 downto 5) and c_2(14 downto 5);
g(10) <= s_2(14) and c_2(15);
c_m(0) <= c_3;

process(p,g,c_m,c_3)
begin
c_m(0) <= c_3;
                  c_m(1) <= g(0) or (c_m(0) and p(0));
                  inst : for i in 1 to 10 loop
                  c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                  end loop;
                  end process;
                  m_1(11 downto 0) <= p(11 downto 0) xor c_m(11 downto 0);

end Behavioral;
```

L.12 Program pengali 4 (*pengali_4.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_4 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                   m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end pengali_4;

architecture Behavioral of pengali_4 is
signal temp_1 : STD_LOGIC_VECTOR(17 DOWNTO 0);
signal a_inv : std_logic_vector(11 downto 0);
signal s_1 : STD_LOGIC_VECTOR(13 DOWNTO 0);
signal c_1 : STD_LOGIC_VECTOR(14 DOWNTO 0);
signal p,c_m : STD_LOGIC_VECTOR(17 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal c_2 : STD_LOGIC;
signal z_1, v_1 : std_logic_vector(3 downto 0);

begin
a_inv <= not(a(11 downto 0));
--stage1
z_1(3 downto 0) <= a(3 downto 0);
s_1(0) <= a(4) xor a_inv(0) xor '1';
s_1(1) <= a(5) xor a_inv(1);
s_1(7 downto 2) <= a(11 downto 6) xor a_inv(7 downto 2) xor a(5 downto 0);
s_1(8) <= a(11) xor a_inv(8) xor a(6);
s_1(9) <= a(11) xor a_inv(9) xor a(7);
s_1(10) <= a(11) xor a_inv(10) xor a(8);
s_1(11) <= a(11) xor a_inv(11) xor a(9);
s_1(12) <= a(11) xor a_inv(11) xor a(10);
s_1(13) <= a(11) xor a_inv(11) xor a(11);

v_1(0) <= '0';
v_1(3 downto 1) <= a(2 downto 0);
c_1(0) <= a(3);
c_1(1) <= (a(4) and a_inv(0)) or (a(4) xor (a_inv(0)));
c_1(2) <= a(5) and a_inv(1);
c_1(8 downto 3) <= (a(11 downto 6) and a_inv(7 downto 2)) or (a(5 downto 0)
and (a(11 downto 6) xor a_inv(7 downto 2)));
c_1(9) <= (a(11) and a_inv(8)) or (a(6) and (a(11) xor a_inv(8)));
c_1(10) <= (a(11) and a_inv(9)) or (a(7) and (a(11) xor a_inv(9)));
c_1(11) <= (a(11) and a_inv(10)) or (a(8) and (a(11) xor a_inv(10)));
c_1(12) <= (a(11) and a_inv(11)) or (a(9) and (a(11) xor a_inv(11)));
c_1(13) <= (a(11) and a_inv(11)) or (a(10) and (a(11) xor a_inv(11)));
c_1(14) <= (a(11) and a_inv(11)) or (a(11) and (a(11) xor a_inv(11)));

c_2 <= (z_1(1) and v_1(1)) or ((z_1(1) xor v_1(1)) and (z_1(0) and v_1(0)));
p(1 downto 0) <= z_1(3 downto 2) xor v_1(3 downto 2);
p(15 downto 2) <= s_1(13 downto 0) xor c_1(13 downto 0);
p(16) <= s_1(13) xor c_1(14);
p(17) <= s_1(13) xor c_1(14);
g(1 downto 0) <= z_1(3 downto 2) and v_1(3 downto 2);
g(15 downto 2) <= s_1(13 downto 0) and c_1(13 downto 0);
g(16) <= s_1(13) and c_1(14);
c_m(0) <= c_2;
```

```
process (p,g,c_m,c_2)
begin
c_m(0) <= c_2;
                    c_m(1) <= g(0) or (c_m(0) and p(0));
                    inst : for i in 1 to 16 loop
                    c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                    end loop;
end process;
temp_1 <= p(17 downto 0) xor c_m(17 downto 0);
det: process (temp_1)
begin
      if (temp_1 = "11111111111111111") then
            m_1 <= not temp_1(17 downto 6);
            else
            m_1 <= temp_1(17 downto 6);
            end if;
            end process;

end Behavioral;
```

L.13 Program pengali 5 (*pengali_5.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_5 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end pengali_5;

architecture Behavioral of pengali_5 is
signal temp_1 : STD_LOGIC_VECTOR (16 downto 0);
signal a_inv : std_logic_vector (11 downto 0);
signal s_1 : STD_LOGIC_VECTOR (14 DOWNTO 1);

signal c_1 : STD_LOGIC_VECTOR (15 DOWNTO 1);
signal p,c_m : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (15 DOWNTO 0);
signal c_2,z,x,y,t : STD_LOGIC;

begin
a_inv <= not a;
--stage1
z <= a_inv(2) xor a_inv(0) xor '1';
x <= a_inv(1);
s_1(2 downto 1) <= a_inv(4 downto 3) xor a_inv(2 downto 1);
s_1(9 downto 3) <= a_inv(11 downto 5) xor a_inv(9 downto 3) xor a(6 downto 0);
s_1(10) <= a_inv(11) xor a_inv(10) xor a(7);
s_1(11) <= a_inv(11) xor a_inv(11) xor a(8);
s_1(12) <= a_inv(11) xor a_inv(11) xor a(9);
s_1(13) <= a_inv(11) xor a_inv(11) xor a(10);
s_1(14) <= a_inv(11) xor a_inv(11) xor a(11);

y <= a_inv(0);
```

```
t <= '0';
c_1(1) <= (a_inv(2) and a_inv(0)) or (a_inv(2) xor a_inv(0));
c_1(3 downto 2) <= a_inv(4 downto 3) and a_inv(2 downto 1);
c_1(10 downto 4) <= (a_inv(11 downto 5) and a_inv(9 downto 3)) or (a( 6 downto
0) and (a_inv(11 downto 5) xor a_inv(9 downto 3)));
c_1(11) <= (a_inv(11) and a_inv(10)) or (a(7) and (a_inv(11) xor a_inv(10)));
c_1(12) <= (a_inv(11) and a_inv(11)) or (a(8) and (a_inv(11) xor a_inv(11)));
c_1(13) <= (a_inv(11) and a_inv(11)) or (a(9) and (a_inv(11) xor a_inv(11)));
c_1(14) <= (a_inv(11) and a_inv(11)) or (a(10) and (a_inv(11) xor a_inv(11)));
c_1(15) <= (a_inv(11) and a_inv(11)) or (a(11) and (a_inv(11) xor a_inv(11)));

c_2 <= (( z and t) or (z xor t)) and (x and y);
p(13 downto 0) <= s_1(14 downto 1) xor c_1(14 downto 1);
p(14) <= s_1(14) xor c_1(15);
p(15) <= s_1(14) xor c_1(15);
p(16) <= s_1(14) xor c_1(15);
g(13 downto 0) <= s_1(14 downto 1) and c_1(14 downto 1);
g(14) <= s_1(14) and  c_1(15);
g(15) <= s_1(14) and  c_1(15);
c_m(0) <= c_2;

process (p,g,c_m,c_2)
begin
c_m(0) <= c_2;
            c_m(1) <= g(0) or (c_m(0) and p(0));
            inst : for i in 1 to 15 loop
            c_m(i + 1) <= g(i) or (p(i) and c_m(i));
            end loop;
end process;
temp_1 <= p(16 downto 0) xor c_m(16 downto 0);
det : process (temp_1)
begin
      if (temp_1 = "11111111111111111") then
             m_1 <= not temp_1(16 downto 5);
             else
             m_1 <= temp_1(16 downto 5);
             end if;
             end process;
end Behavioral;
```

L.14 Program pengali 6 (*pengali_6.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_6 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end pengali_6;

architecture Behavioral of pengali_6 is
signal a_inv : STD_LOGIC_VECTOR (11 DOWNTO 0);
signal temp_1 : std_logic_vector (18 downto 0);
signal s_1 : STD_LOGIC_VECTOR (15 DOWNTO 0);
signal s_2 : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal s_3 : STD_LOGIC_VECTOR (18 DOWNTO 0);

signal c_1 : STD_LOGIC_VECTOR (16 DOWNTO 0);
```

```
signal c_2 : STD_LOGIC_VECTOR (18 DOWNTO 0);
signal c_3 : STD_LOGIC_VECTOR (18 DOWNTO 0);
signal p,c_m : STD_LOGIC_VECTOR (18 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal c_4,z_2,v_2 : STD_LOGIC;
signal z_3,v_3 : std_logic_vector (1 downto 0);

begin
a_inv <= not(a);
--stage1
s_1(0) <= a_inv(1);
s_1(1) <= a_inv(2);
s_1(3 downto 2) <= a_inv(4 downto 3) xor a(1 downto 0);
s_1(10 downto 4) <= a_inv(11 downto 5) xor a(8 downto 2) xor a_inv(6 downto
0);
s_1(11) <= a_inv(11) xor a(9) xor a_inv(7);
s_1(12) <= a_inv(11) xor a(10) xor a_inv(8);
s_1(13) <= a_inv(11) xor a(11) xor a_inv(9);
s_1(14) <= a_inv(11) xor a(11) xor a_inv(10);
s_1(15) <= a_inv(11) xor a(11) xor a_inv(11);

c_1(0) <= a_inv(0);
c_1(1) <= a_inv(1);
c_1(2) <= a_inv(2);
c_1(4 downto 3) <= a_inv(4 downto 3) and a(1 downto 0);
c_1(11 downto 5) <= (a_inv(11 downto 5) and a(8 downto 2)) or (a_inv(6 downto
0) and (a_inv(11 downto 5) xor a(8 downto 2)));
c_1(12) <= (a_inv(11) and a(9)) or (a_inv(7) and (a_inv(11) xor a(9)));
c_1(13) <= (a_inv(11) and a(10)) or (a_inv(8) and (a_inv(11) xor a(10)));
c_1(14) <= (a_inv(11) and a(11)) or (a_inv(9) and (a_inv(11) xor a(11)));
c_1(15) <= (a_inv(11) and a(11)) or (a_inv(10) and (a_inv(11) xor a(11)));
c_1(16) <= (a_inv(11) and a(11)) or (a_inv(11) and (a_inv(11) xor a(11)));

--stage2
z_2 <= s_1(0) xor c_1(0);
s_2(2 downto 0) <= s_1(3 downto 1) xor c_1(3 downto 1);
s_2(3) <= s_1(4) xor c_1(4) xor '1';
s_2(5 downto 4) <= s_1(6 downto 5) xor c_1(6 downto 5);
s_2(14 downto 6) <= s_1(15 downto 7) xor c_1(15 downto 7) xor a_inv(8 downto
0);
s_2(15) <= s_1(15) xor c_1(16) xor a_inv(9);
s_2(16) <= s_1(15) xor c_1(16) xor a_inv(10);
s_2(17) <= s_1(15) xor c_1(16) xor a_inv(11);

v_2 <= '0';
c_2(3 downto 0) <= s_1(3 downto 0) and c_1(3 downto 0);
c_2(4) <= (s_1(4) and c_1(4)) or (s_1(4) xor c_1(4));
c_2(6 downto 5) <= s_1(6 downto 5) and c_1(6 downto 5);
c_2(15 downto 7) <= (s_1(15 downto 7) and c_1(15 downto 7)) or (a_inv(8 downto
0) and (s_1(15 downto 7) xor c_1(15 downto 7)));
c_2(16) <= (s_1(15) and c_1(16)) or (a_inv(9) and (s_1(15) xor c_1(16)));
c_2(17) <= (s_1(15) and c_1(16)) or (a_inv(10) and (s_1(15) xor c_1(16)));
c_2(18) <= (s_1(15) and c_1(16)) or (a_inv(11) and (s_1(15) xor c_1(16)));

--stage3
z_3(0) <= z_2 xor v_2;
z_3(1) <= s_2(0) xor c_2(0);
s_3(4 downto 0) <= s_2(5 downto 1) xor c_2(5 downto 1);
s_3(5) <= s_2(6) xor c_2(6) xor '1';
s_3(6) <= s_2(7) xor c_2(7);

s_3(16 downto 7) <= s_2(17 downto 8) xor c_2(17 downto 8) xor a(9 downto 0);
s_3(17) <= s_2(17) xor c_2(18) xor a(10);
s_3(18) <= s_2(17) xor c_2(18) xor a(11);

v_3(0) <= '0';
v_3(1) <= z_2 and v_2;
c_3(5 downto 0) <= s_2(5 downto 0) and c_2(5 downto 0);
```

```
c_3(6) <= (s_2(6) and c_2(6)) or (s_2(6) xor c_2(6));
c_3(7) <= s_2(7) and c_2(7);
c_3(17 downto 8) <= (s_2(17 downto 8) and c_2(17 downto 8)) or (a(9 downto 0)
and (s_2(17 downto 8) xor c_2(17 downto 8)));
c_3(18) <= (s_2(17) and c_2(18)) or (a(10) and (s_2(17) xor c_2(18)));

c_4 <= (z_3(1) and v_3(1)) or ((z_3(1) xor v_3(1)) and (z_3(0) and v_3(0)));
p(18 downto 0) <= s_3(18 downto 0) xor c_3(18 downto 0);
g(17 downto 0) <= s_3(17 downto 0) and c_3(17 downto 0);
c_m(0) <= c_4;

process (p,g,c_m,c_4)
begin
c_m(0) <= c_4;
                    c_m(1) <= g(0) or (c_m(0) and p(0));
                    inst : for i in 1 to 17 loop
                    c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                    end loop;
end process;
temp_1 <= p(18 downto 0) xor c_m(18 downto 0);
det : process (temp_1)
begin
      if (temp_1 = "111111111111111111") then
            m_1 <= not temp_1(18 downto 7);
            else
            m_1 <= temp_1(18 downto 7);
            end if;
            end process;

end Behavioral;
```

L.15 Program pengali 7 (*pengali_7.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_7 is
     port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                    m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end pengali_7;

architecture Behavioral of pengali_7 is
signal temp_1 : STD_LOGIC_VECTOR (15 DOWNTO 0);
signal a_inv : std_logic_vector (11 downto 0);
signal s_1 : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal s_2 : STD_LOGIC_VECTOR (17 DOWNTO 0);

signal c_1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal c_2 : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal p,c_m : STD_LOGIC_VECTOR (15 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal c_3 : STD_LOGIC;

begin
a_inv <= not(a(11 downto 0));
```

```
--stage1
s_1(0) <= a_inv(1);
s_1(1) <= a_inv(2) xor a(0);
s_1(10 downto 2) <= a_inv(11 downto 3) xor a(9 downto 1) xor a(8 downto 0);
s_1(11) <= a_inv(11) xor a(10) xor a(9);
s_1(12) <= a_inv(11) xor a(11) xor a(10);
s_1(13) <= a_inv(11) xor a(11) xor a(11);

c_1(0) <= a_inv(0);
c_1(1) <= '0';
c_1(2) <= a_inv(2) and a(0);
c_1(11 downto 3) <= (a_inv(11 downto 3) and a(9 downto 1)) or (a(8 downto 0)
and (a_inv(11 downto 3) xor a(9 downto 1)));
c_1(12) <= (a_inv(11) and a(10)) or (a(9) and (a_inv(11) xor a(10)));
c_1(13) <= (a_inv(11) and a(11)) or (a(10) and (a_inv(11) xor a(11)));
c_1(14) <= (a_inv(11) and a(11)) or (a(11) and (a_inv(11) xor a(11)));

--stage2
s_2(5 downto 0) <= s_1(5 downto 0) xor c_1(5 downto 0);
s_2(13 downto 6) <= s_1(13 downto 6) xor c_1(13 downto 6) xor a(7 downto 0);
s_2(14) <= s_1(13) xor c_1(14) xor a(8);
s_2(15) <= s_1(13) xor c_1(14) xor a(9);
s_2(16) <= s_1(13) xor c_1(14) xor a(10);
s_2(17) <= s_1(13) xor c_1(14) xor a(11);

c_2(0) <= '0';
c_2(6 downto 1) <= s_1(5 downto 0) and c_1(5 downto 0);
c_2(14 downto 7) <= (s_1(13 downto 6) and c_1(13 downto 6)) or (a(7 downto 0)
and (s_1(13 downto 6) xor c_1(13 downto 6)));
c_2(15) <= (s_1(13) and c_1(14)) or (a(8) and (s_1(13) xor c_1(14)));
c_2(16) <= (s_1(13) and c_1(14)) or (a(9) and (s_1(13) xor c_1(14)));
c_2(17) <= (s_1(13) and c_1(14)) or (a(10) and (s_1(13) xor c_1(14)));

c_3 <= (s_2(1) and c_2(1)) or ((s_2(1) xor c_2(1)) and (s_2(0) and c_2(0)));
p(15 downto 0) <= s_2(17 downto 2) xor c_2(17 downto 2);
g(14 downto 0) <= s_2(16 downto 2) xor c_2(16 downto 2);
c_m(0) <= c_3;

process (p,g,c_m,c_3)
begin
c_m(0) <= c_3;
            c_m(1) <= g(0) or (c_m(0) and p(0));
            inst : for i in 1 to 14 loop
            c_m(i + 1) <= g(i) or (p(i) and c_m(i));
            end loop;
end process;
temp_1 <= p(15 downto 0) xor c_m(15 downto 0);
det : process (temp_1)
begin
      if (temp_1 = "1111111111111111") then
              m_1 <= not temp_1(15 downto 4);
              else
              m_1 <= temp_1(15 downto 4);
              end if;
              end process;

end Behavioral;
```

L.16 Program pengali 8 (*pengali_8.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_8 is
      port ( a : in STD_LOGIC_VECTOR (11 downto 0);
                  m_1 : out STD_LOGIC_VECTOR (11 downto 0));
end pengali_8;

architecture Behavioral of pengali_8 is
signal a_inv : STD_LOGIC_VECTOR (11 DOWNTO 0);
signal temp_1 : std_logic_vector (14 downto 0);
signal s_1 : STD_LOGIC_VECTOR (15 DOWNTO 0);
signal s_2 : STD_LOGIC_VECTOR (16 DOWNTO 0);


signal c_1 : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal c_2 : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal p,c_m : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal c_3 : STD_LOGIC;

begin
a_inv <= not(a);
--stage1
s_1(0) <= a_inv(0) xor '1';
s_1(3 downto 1) <= a_inv(3 downto 1) xor a(2 downto 0);
s_1(11 downto 4) <= a_inv(11 downto 4) xor a(10 downto 3) xor a_inv(7 downto
0);
s_1(12) <= a_inv(11) xor a(11) xor a_inv(8);
s_1(13) <= a_inv(11) xor a(11) xor a_inv(9);
s_1(14) <= a_inv(11) xor a(11) xor a_inv(10);
s_1(15) <= a_inv(11) xor a(11) xor a_inv(11);

c_1(0) <= '0';
c_1(1) <= a_inv(0);
c_1(4 downto 2) <= a_inv(3 downto 1) and a(2 downto 0);
c_1(12 downto 5) <= (a_inv(11 downto 4) and a(10 downto 3)) or (a_inv(7 downto
0) and (a_inv(11 downto 4) xor a(10 downto 3)));
c_1(13) <= (a_inv(11) and a(11)) or (a_inv(8) and (a_inv(11) xor a(11)));
c_1(14) <= (a_inv(11) and a(11)) or (a_inv(9) and (a_inv(11) xor a(11)));
c_1(15) <= (a_inv(11) and a(11)) or (a_inv(10) and (a_inv(11) xor a(11)));
c_1(16) <= (a_inv(11) and a(11)) or (a_inv(11) and (a_inv(11) xor a(11)));


--stage2
s_2(3 downto 0) <= s_1(3 downto 0) xor c_1(3 downto 0);
s_2(4) <= s_1(4) xor c_1(4) xor '1';
s_2(15 downto 5) <= s_1(15 downto 5) xor c_1(15 downto 5) xor a(10 downto 0);
s_2(16) <= s_1(15) xor c_1(16) xor a(11);

c_2(0) <= '0';
c_2(4 downto 1) <= s_1(3 downto 0) and c_1(3 downto 0);
c_2(5) <= (s_1(4) and c_1(4)) or (s_1(4) xor c_1(4));
c_2(16 downto 6) <= (s_1(15 downto 5) and c_1(15 downto 5)) or (a(10 downto 0)
and (s_1(15 downto 5) xor c_1(15 downto 5)));

c_3 <= (s_2(1) and c_2(1)) or ((s_2(1) xor c_2(1)) and (s_2(0) and c_2(0)));
p(14 downto 0) <= s_2(16 downto 2) xor c_2(16 downto 2);
g(13 downto 0) <= s_2(15 downto 2) and c_2(15 downto 2);
c_m(0) <= c_3;

process (p,g,c_m,c_3)
begin
c_m(0) <= c_3;
            c_m(1) <= g(0) or (c_m(0) and p(0));
```

```
            inst : for i in 1 to 13 loop
            c_m(i + 1) <= g(i) or (p(i) and c_m(i));
            end loop;
end process;
temp_1 <= p(14 downto 0) xor c_m(14 downto 0);
det : process (temp_1)
begin
      if (temp_1 = "111111111111111111") then
            m_1 <= not temp_1(14 downto 3);
            else
            m_1 <= temp_1(14 downto 3);
            end if;
            end process;

end Behavioral;
```

L.17 Program pengali 9 (*pengali_9.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_9 is
      port ( a : in STD_LOGIC_VECTOR (11 DOWNTO 0);
                  m_1 : out STD_LOGIC_VECTOR (11 DOWNTO 0));
end pengali_9;

architecture Behavioral of pengali_9 is
signal temp_1 : STD_LOGIC_VECTOR (18 DOWNTO 0);
signal a_inv : std_logic_vector ( 11 downto 0);
signal s_1 : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal s_2 : STD_LOGIC_VECTOR (18 DOWNTO 0);

signal c_1 : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal c_2 : STD_LOGIC_VECTOR (19 DOWNTO 0);
signal p,c_m : STD_LOGIC_VECTOR (18 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal c_3 : STD_LOGIC;

begin
a_inv <= not(a(11 downto 0));
--stage1
s_1(0) <= a_inv(0) xor '1';
s_1(1) <= a_inv(1);
s_1(4 downto 2) <= a_inv(4 downto 2) xor a(2 downto 0);
s_1(11 downto 5) <= a_inv(11 downto 5) xor a(9 downto 3) xor a(6 downto 0);
s_1(12) <= a_inv(11) xor a(10) xor a(7);
s_1(13) <= a_inv(11) xor a(11) xor a(8);
s_1(14) <= a_inv(11) xor a(11) xor a(9);
s_1(15) <= a_inv(11) xor a(11) xor a(10);
s_1(16) <= a_inv(11) xor a(11) xor a(11);

c_1(0) <= '0';
c_1(1) <= a_inv(0);
c_1(2) <= '0';
c_1(5 downto 3) <= a_inv(4 downto 2) and a(2 downto 0);
```

```
c_1(12 downto 6) <= (a_inv(11 downto 5) and a(9 downto 3)) or (a(6 downto 0)
and (a_inv(11 downto 5) xor a(9 downto 3)));
c_1(13) <= (a_inv(11) and a(10)) or (a(7) and (a_inv(11) xor a(10)));
c_1(14) <= (a_inv(11) and a(11)) or (a(8) and (a_inv(11) xor a(11)));
c_1(15) <= (a_inv(11) and a(11)) or (a(9) and (a_inv(11) xor a(11)));
c_1(16) <= (a_inv(11) and a(11)) or (a(10) and (a_inv(11) xor a(11)));
c_1(17) <= (a_inv(11) and a(11)) or (a(11) and (a_inv(11) xor a(11)));

--stage2
s_2(6 downto 0) <= s_1(6 downto 0) xor c_1(6 downto 0);
s_2(16 downto 7) <= s_1(16 downto 7) xor c_1(16 downto 7) xor a(9 downto 0);
s_2(17) <= s_1(16) xor c_1(17) xor a(10);
s_2(18) <= s_1(16) xor c_1(17) xor a(11);

c_2(0) <= '0';
c_2(7 downto 1) <= s_1(6 downto 0) and c_1(6 downto 0);
c_2(17 downto 8) <= (s_1(16 downto 7) and c_1(16 downto 7)) or (a(9 downto 0)
and (s_1(16 downto 7) xor c_1(16 downto 7)));
c_2(18) <= (s_1(16) and c_1(17)) or (a(10) and (s_1(16) xor c_1(17)));
c_2(19) <= (s_1(16) and c_1(17)) or (a(11) and (s_1(16) xor c_1(17)));

c_3 <= (s_2(1) and c_2(1)) or ((s_2(1) xor c_2(1)) and (s_2(0) and c_2(0)));
p(16 downto 0) <= s_2(18 downto 2) xor c_2(18 downto 2);
p(17) <= s_2(18) xor c_2(19);
p(18) <= s_2(18) xor c_2(19);
g(16 downto 0) <= s_2(18 downto 2) and c_2(18 downto 2);
g(17) <= s_2(18) and c_2(19);
c_m(0) <= c_3;

process (p,g,c_m,c_3)
begin
c_m(0) <= c_3;
                    c_m(1) <= g(0) or (c_m(0) and p(0));
                    inst : for i in 1 to 17 loop
                    c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                    end loop;
end process;
temp_1 <= p(18 downto 0) xor c_m(18 downto 0);
det : process (temp_1)
begin
if (temp_1 = "11111111111111111") then
             m_1 <= not temp_1(18 downto 7);
             else
             m_1 <= not temp_1(18 downto 7) +1 ;
             end if;
             end process;

end Behavioral;
```

L.18 Program pengali 10 (*pengali_10.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_10 is
```

```
      port ( a : in STD_LOGIC_VECTOR (11 DOWNTO 0);
                   m_1 : out STD_LOGIC_VECTOR (11 DOWNTO 0));
end pengali_10;

architecture Behavioral of pengali_10 is
signal temp_1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal a_inv : std_logic_vector (11 downto 0);
signal s_1 : STD_LOGIC_VECTOR (15 DOWNTO 0);

signal c_1 : STD_LOGIC_VECTOR  (15 DOWNTO 0);
signal p,c_m : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal c_2,z,v : STD_LOGIC;

begin
a_inv <= not(a(11 downto 0));
--stage1
z <= a_inv(0) xor '1';
s_1(0) <= a_inv(1) xor a_inv(0) xor '1';
s_1(3 downto 1) <= a_inv(4 downto 2) xor a_inv(3 downto 1);
s_1(10 downto 4) <= a_inv(11 downto 5) xor a_inv(10 downto 4) xor a(6 downto
0);
s_1(11) <= a_inv(11) xor a_inv(11) xor a(7);
s_1(12) <= a_inv(11) xor a_inv(11) xor a(8);
s_1(13) <= a_inv(11) xor a_inv(11) xor a(9);
s_1(14) <= a_inv(11) xor a_inv(11) xor a(10);
s_1(15) <= a_inv(11) xor a_inv(11) xor a(11);

v <= '0';
c_1(0) <= a_inv(0);
c_1(1) <= (a_inv(1) and a_inv(0)) or (a_inv(1) xor a_inv(0));
c_1(4 downto 2) <= a_inv(4 downto 2) and a_inv(3 downto 1);
c_1(11 downto 5) <= (a_inv(11 downto 5) and a_inv(10 downto 4)) or (a(6 downto
0) and (a_inv(11 downto 5) xor a_inv(10 downto 4)));
c_1(12) <= (a_inv(11) and a_inv(11)) or (a(7) and (a_inv(11) xor a_inv(11)));
c_1(13) <= (a_inv(11) and a_inv(11)) or (a(8) and (a_inv(11) xor a_inv(11)));
c_1(14) <= (a_inv(11) and a_inv(11)) or (a(9) and (a_inv(11) xor a_inv(11)));
c_1(15) <= (a_inv(11) and a_inv(11)) or (a(10) and (a_inv(11) xor a_inv(11)));

c_2 <= (s_1(0) and c_1(0)) or ((s_1(0) xor c_1(0)) and (z and v));
p(14 downto 0) <= s_1(15 downto 1) xor c_1(15 downto 1);
g(13 downto 0) <= s_1(14 downto 1) and c_1(14 downto 1);
c_m(0) <= c_2;

process (p,g,c_m,c_2)
begin
c_m(0) <= c_2;
                   c_m(1) <= g(0) or (c_m(0) and p(0));
                   inst : for i in 1 to 13 loop
                   c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                   end loop;
end process;
temp_1 <= p(14 downto 0) xor c_m(14 downto 0);
det : process (temp_1)
begin
if (temp_1 = "111111111111111111") then
            m_1 <= not temp_1(14 downto 3);
            else
            m_1 <= not temp_1(14 downto 3) +1 ;
            end if;
            end process;

end Behavioral;
```

L.19 Program pengali 11 (*pengali_11.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_11 is
      port ( a : in STD_LOGIC_VECTOR (11 DOWNTO 0);
                  m_1 : out STD_LOGIC_VECTOR (11 DOWNTO 0));
end pengali_11;

architecture Behavioral of pengali_11 is
signal a_inv : STD_LOGIC_VECTOR (11 DOWNTO 0);
signal temp_1 : std_logic_vector (18 downto 0);
signal s_1 : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal s_2 : STD_LOGIC_VECTOR (18 DOWNTO 0);
signal s_3 : STD_LOGIC_VECTOR (20 DOWNTO 0);

signal c_1 : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal c_2 : STD_LOGIC_VECTOR (19 DOWNTO 0);
signal c_3 : STD_LOGIC_VECTOR (20 DOWNTO 0);
signal p,c_m : STD_LOGIC_VECTOR (18 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal c_4 : STD_LOGIC;

begin
a_inv <= not(a);
--stage1
s_1(0) <= a_inv(0) xor '1';
s_1(1) <= a_inv(1);
s_1(4 downto 2) <= a_inv(4 downto 2) xor a(2 downto 0);
s_1(11 downto 5) <= a_inv(11 downto 5) xor a(9 downto 3) xor a_inv(6 downto
0);
s_1(12) <= a_inv(11) xor a(10) xor a_inv(7);
s_1(13) <= a_inv(11) xor a(11) xor a_inv(8);
s_1(14) <= a_inv(11) xor a(11) xor a_inv(9);
s_1(15) <= a_inv(11) xor a(11) xor a_inv(10);
s_1(16) <= a_inv(11) xor a(11) xor a_inv(11);

c_1(0) <= '0';
c_1(1) <= a_inv(0);
c_1(2) <= '0';
c_1(5 downto 3) <= a_inv(4 downto 2) and a(2 downto 0);
c_1(12 downto 6) <= (a_inv(11 downto 5) and a(9 downto 3)) or (a_inv(6 downto
0) and (a_inv(11 downto 5) xor a(9 downto 3)));
c_1(13) <= (a_inv(11) and a(10)) or (a_inv(7) and (a_inv(11) xor a(10)));
c_1(14) <= (a_inv(11) and a(11)) or (a_inv(8) and (a_inv(11) xor a(11)));
c_1(15) <= (a_inv(11) and a(11)) or (a_inv(9) and (a_inv(11) xor a(11)));
c_1(16) <= (a_inv(11) and a(11)) or (a_inv(10) and (a_inv(11) xor a(11)));
c_1(17) <= (a_inv(11) and a(11)) or (a_inv(11) and (a_inv(11) xor a(11)));

--stage2
s_2(4 downto 0) <= s_1(4 downto 0) xor c_1(4 downto 0);
s_2(5) <= s_1(5) xor c_1(5) xor '1';
s_2(6) <= s_1(6) xor c_1(6);
s_2(16 downto 7) <= s_1(16 downto 7) xor c_1(16 downto 7) xor a_inv(9 downto
0);
s_2(17) <= s_1(16) xor c_1(17) xor a_inv(10);
```

```
s_2(18) <= s_1(16) xor c_1(17) xor a_inv(11);

c_2(0) <= '0';
c_2(5 downto 1) <= s_1(4 downto 0) and c_1(4 downto 0);
c_2(6) <= (s_1(5) and c_1(5)) or (s_1(5) xor c_1(5));
c_2(7) <= s_1(6) and c_1(6);
c_2(17 downto 8) <= (s_1(16 downto 7) and c_1(16 downto 7)) or (a_inv(9 downto
0) and (s_1(16 downto 7) xor c_1(16 downto 7)));
c_2(18) <= (s_1(16) and c_1(17)) or (a_inv(10) and (s_1(16) xor c_1(17)));
c_2(19) <= (s_1(16) and c_1(17)) or (a_inv(11) and (s_1(16) xor c_1(17)));

--stage3
s_3(6 downto 0) <= s_2(6 downto 0) xor c_2(6 downto 0);
s_3(7) <= s_2(7) xor c_2(7) xor '1';
s_3(8) <= s_2(8) xor c_2(8);
s_3(18 downto 9) <= s_2(18 downto 9) xor c_2(18 downto 9) xor a(9 downto 0);
s_3(19) <= s_2(18) xor c_2(19) xor a(10);
s_3(20) <= s_2(18) xor c_2(19) xor a(11);

c_3(0) <= '0';
c_3(7 downto 1) <= s_2(6 downto 0) and c_2(6 downto 0);
c_3(8) <= (s_2(7) and c_2(7)) or (s_2(7) xor c_2(7));
c_3(9) <= s_2(8) and c_2(8);
c_3(19 downto 10) <= (s_2(18 downto 9) and c_2(18 downto 9)) or (a(9 downto 0)
and (s_2(18 downto 9) xor c_2(18 downto 9)));
c_3(20) <= (s_2(18) and c_2(19)) or (a(10) and (s_2(18) xor c_2(19)));

c_4 <= (s_3(1) and c_3(1)) or ((s_3(1) xor c_3(1)) and (s_3(0) and c_3(0)));
p(18 downto 0) <= s_3(20 downto 2) xor c_3(20 downto 2);
g(17 downto 0) <= s_3(19 downto 2) xor c_3(19 downto 2);
c_m(0) <= c_4;

process (p,g,c_m,c_4)
begin
c_m(0) <= c_4;
                    c_m(1) <= g(0) or (c_m(0) and p(0));
                    inst : for i in 1 to 17 loop
                    c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                    end loop;
end process;
temp_1 <= p(18 downto 0) xor c_m(18 downto 0);
det : process (temp_1)
begin
if (temp_1 = "11111111111111111") then
             m_1 <= not temp_1(18 downto 7);
             else
             m_1 <= not temp_1(18 downto 7) +1 ;
             end if;
             end process;

end Behavioral;
```

L.20 Program pengali 12 (*pengali_12.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

entity pengali_12 is
            port ( a : in STD_LOGIC_VECTOR (11 DOWNTO 0);
                   m_1 : out STD_LOGIC_VECTOR (11 DOWNTO 0));
end pengali_12;

architecture Behavioral of pengali_12 is
signal a_inv : STD_LOGIC_VECTOR (11 DOWNTO 0);
signal s_1 : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal s_2 : STD_LOGIC_VECTOR (12 DOWNTO 0);
signal s_3 : STD_LOGIC_VECTOR (12 DOWNTO 0);

signal c_1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal c_2 : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal c_3 : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal p,c_m,temp_1 : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal c_4 : STD_LOGIC;
signal z_2,v_2 : std_logic_vector (3 downto 0);
signal z_3,v_3 : std_logic_vector (4 downto 0);

begin
a_inv <= not(a);
--stage1
s_1(0) <= a(2) xor a_inv(0) xor '1';
s_1(1) <= a(3) xor a_inv(1);
s_1(9 downto 2) <= a(11 downto 4) xor a_inv(9 downto 2) xor a(7 downto 0);
s_1(10) <= a(11) xor a_inv(10) xor a(8);
s_1(11) <= a(11) xor a_inv(11) xor a(9);
s_1(12) <= a(11) xor a_inv(11) xor a(10);
s_1(13) <= a(11) xor a_inv(11) xor a(11);

c_1(0) <= a(1);
c_1(1) <= (a(2) and a_inv(0)) or (a(2) xor a_inv(0));
c_1(2) <= a(3) and a_inv(1);
c_1(10 downto 3) <= (a(11 downto 4) and a_inv(9 downto 2)) or (a(7 downto 0)
and (a(11 downto 4) xor a_inv(9 downto 2)));
c_1(11) <= (a(11) and a_inv(10)) or (a(8) and (a(11) xor a_inv(10)));
c_1(12) <= (a(11) and a_inv(11)) or (a(9) and (a(11) xor a_inv(11)));
c_1(13) <= (a(11) and a_inv(11)) or (a(10) and (a(11) xor a_inv(11)));
c_1(14) <= (a(11) and a_inv(11)) or (a(11) and (a(11) xor a_inv(11)));

--stage2
z_2(3 downto 0) <= s_1(3 downto 0) xor c_1(3 downto 0);
s_2(0) <= s_1(4) xor c_1(4);
s_2(9 downto 1) <= s_1(13 downto 5) xor c_1(13 downto 5) xor a_inv(8 downto
0);
s_2(10) <= s_1(13) xor c_1(14) xor a_inv(9);
s_2(11) <= s_1(13) xor c_1(14) xor a_inv(10);
s_2(12) <= s_1(13) xor c_1(14) xor a_inv(11);

v_2(0) <= '0';
v_2(3 downto 1) <= s_1(2 downto 0) and c_1(2 downto 0);
c_2(1 downto 0) <= s_1(4 downto 3) and c_1(4 downto 3);
c_2(10 downto 2) <= (s_1(13 downto 5) and c_1(13 downto 5)) or (a_inv(8 downto
0) and (s_1(13 downto 5) xor c_1(13 downto 5)));
c_2(11) <= (s_1(13) and c_1(14)) or (a_inv(9) and (s_1(13) xor c_1(14)));
c_2(12) <= (s_1(13) and c_1(14)) or (a_inv(10) and (s_1(13) xor c_1(14)));
c_2(13) <= (s_1(13) and c_1(14)) or (a_inv(11) and (s_1(13) xor c_1(14)));

--stage3
z_3(3 downto 0) <= z_2(3 downto 0) and v_2(3 downto 0);
z_3(4) <= s_2(0) xor c_2(0);
s_3(0) <= s_2(1) xor c_2(1) xor '1';
s_3(11 downto 1) <= s_2(12 downto 2) xor c_2(12 downto 2) xor a(10 downto 0);
s_3(12) <= s_2(12) xor c_2(13) xor a(11);
```

```
v_3(0) <= '0';
v_3(4 downto 1) <= z_2(3 downto 0) and v_2(3 downto 0);
c_3(0) <= s_2(0) and c_2(0);
c_3(1) <= (s_2(1) and c_2(1)) or (s_2(1) xor c_2(1));
c_3(12 downto 2) <= (s_2(12 downto 2) and c_2(12 downto 2)) or (a(10 downto 0)
and (s_2(12 downto 2) xor c_2(12 downto 2)));
c_3(13) <= (s_2(12) and c_2(13)) or (a(11) and (s_2(12) xor c_2(13)));

c_4 <= (z_3(1) and z_3(1)) or ((z_3(1) xor z_3(1)) and (z_3(0) and z_3(0)));
p(2 downto 0) <= z_3(4 downto 2) xor v_3(4 downto 2);
p(15 downto 3) <= s_3(12 downto 0) xor c_3(12 downto 0);
p(16) <= s_3(12) xor c_3(13);
p(17) <= s_3(12) xor c_3(13);
g(2 downto 0) <= z_3(4 downto 2) and v_3(4 downto 2);
g(15 downto 3) <= s_3(12 downto 0) and c_3(12 downto 0);
g(16) <= s_3(12) and c_3(13);
c_m(0) <= c_4;

process (p,g,c_m,c_4)
begin
c_m(0) <= c_4;
              c_m(1) <= g(0) or (c_m(0) and p(0));
              inst : for i in 1 to 16 loop
                     c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                     end loop;
end process;
temp_1 <= p(17 downto 0) xor c_m(17 downto 0);
det : process (temp_1)
begin
if (temp_1 = "11111111111111111") then
              m_1 <= not temp_1(17 downto 6);
              else
              m_1 <= not temp_1(17 downto 6) +1 ;
              end if;
              end process;

end Behavioral;
```

L.21 Program pengali 13 (*pengali_13.vhd*)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pengali_13 is
              port ( a : in STD_LOGIC_VECTOR (11 DOWNTO 0);
                     m_1 : out STD_LOGIC_VECTOR (11 DOWNTO 0));
end pengali_13;

architecture Behavioral of pengali_13 is
signal a_inv : STD_LOGIC_VECTOR (11 DOWNTO 0);
signal s_1 : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal s_2 : STD_LOGIC_VECTOR (13 DOWNTO 0);
signal s_3 : STD_LOGIC_VECTOR (14 DOWNTO 0);

signal c_1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
```

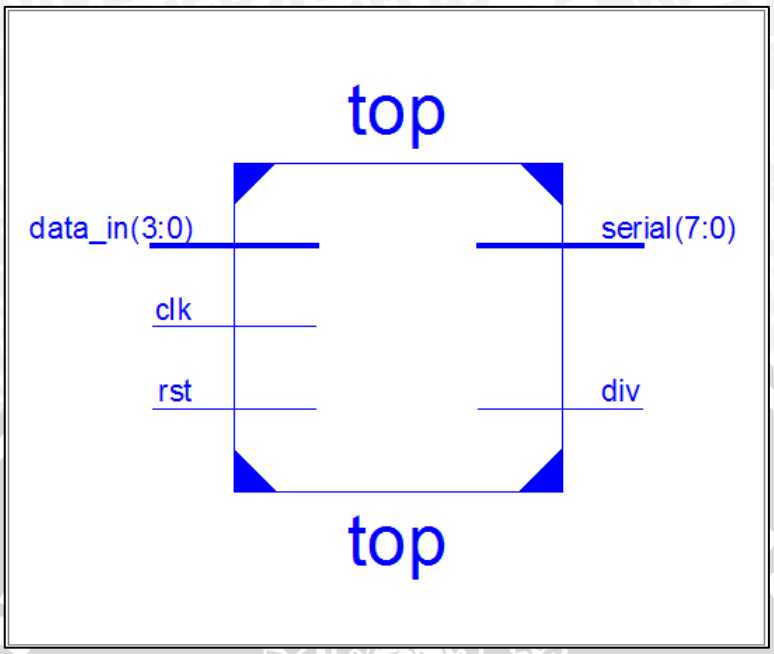
```
signal c_2 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal c_3 : STD_LOGIC_VECTOR (15 DOWNTO 0);
signal p,c_m,temp_1 : STD_LOGIC_VECTOR (17 DOWNTO 0);
signal g : STD_LOGIC_VECTOR (16 DOWNTO 0);
signal c_4 : STD_LOGIC;
signal z_2,v_2 : std_logic_vector (1 downto 0);
signal z_3,v_3 : std_logic_vector (3 downto 0);

begin
a_inv <= not(a);
--stage1
s_1(0) <= a_inv(1) xor a_inv(0) xor '1';
s_1(1) <= a_inv(2) xor a_inv(1);
s_1(10 downto 2) <= a_inv(11 downto 3) xor a_inv(10 downto 2) xor a_inv(8
downto 0);
s_1(11) <= a_inv(11) xor a_inv(11) xor a_inv(9);
s_1(12) <= a_inv(11) xor a_inv(11) xor a_inv(10);
s_1(13) <= a_inv(11) xor a_inv(11) xor a_inv(11);

c_1(0) <= a_inv(0);
c_1(1) <= (a_inv(1) and a_inv(0)) or (a_inv(1) xor a_inv(0));
c_1(2) <= a_inv(2) and a_inv(1);
c_1(11 downto 3) <= (a_inv(11 downto 3) and a_inv(10 downto 2)) or (a_inv(8
downto 0) and (a_inv(11 downto 3) xor a_inv(10 downto 2)));
c_1(12) <= (a_inv(11) and a_inv(11)) or (a_inv(9) and (a_inv(11) xor
a_inv(11)));
c_1(13) <= (a_inv(11) and a_inv(11)) or (a_inv(10) and (a_inv(11) xor
a_inv(11)));
c_1(14) <= (a_inv(11) and a_inv(11)) or (a_inv(11) and (a_inv(11) xor
a_inv(11)));

--stage2
z_2(1 downto 0) <= s_1(1 downto 0) xor c_1(1 downto 0);
s_2(0) <= s_1(2) xor c_1(2) xor '1';
s_2(1) <= s_1(3) xor c_1(3);
s_2(11 downto 2) <= s_1(13 downto 4) xor c_1(13 downto 4) xor a_inv(9 downto
0);
s_2(12) <= s_1(13) xor c_1(14) xor a_inv(10);
s_2(13) <= s_1(13) xor c_1(14) xor a_inv(11);

v_2(0) <= '0';
v_2(1) <= s_1(0) and c_1(0);
c_2(0) <= s_1(1) and c_1(1);
c_2(1) <= (s_1(2) and c_1(2)) or (s_1(2) xor c_1(2));
c_2(2) <= s_1(3) and c_1(3);
c_2(12 downto 3) <= (s_1(13 downto 4) and c_1(13 downto 4)) or (c_1(13 downto
4) and a_inv(9 downto 0)) or (s_1(13 downto 4) and a_inv(9 downto 0));
c_2(13) <= (s_1(13) and c_1(14)) or (a_inv(10) and (s_1(13) xor c_1(14)));
c_2(14) <= (s_1(13) and c_1(14)) or (a_inv(11) and (s_1(13) xor c_1(14)));

--stage3
z_3(1 downto 0) <= z_2(1 downto 0) xor v_2(1 downto 0);
z_3(3 downto 2) <= s_2(1 downto 0) xor c_2(1 downto 0);
s_3(0) <= s_2(2) xor c_2(2) xor '1';
s_3(2 downto 1) <= s_2(4 downto 3) xor c_2(4 downto 3);
s_3(11 downto 3) <= s_2(13 downto 5) xor c_2(13 downto 5) xor a(8 downto 0);
s_3(12) <= s_2(13) xor c_2(14) xor a(9);
s_3(13) <= s_2(13) xor c_2(14) xor a(10);
s_3(14) <= s_2(13) xor c_2(14) xor a(11);

v_3(0) <= '0';
v_3(2 downto 1) <= z_2(1 downto 0) and v_2(1 downto 0);
v_3(3) <= s_2(0) and c_2(0);
c_3(0) <= s_2(1) and c_2(1);
c_3(1) <= (s_2(2) and c_2(2)) or (s_2(2) xor c_2(2));
c_3(3 downto 2) <= s_2(4 downto 3) and c_2(4 downto 3);
c_3(12 downto 4) <= (s_2(13 downto 5) and c_2(13 downto 5)) or (a(8 downto 0)
and (s_2(13 downto 5) xor c_2(13 downto 5)));
```

```
c_3(13) <= (s_2(13) and c_2(14)) or (a(9) and (s_2(13) xor c_2(14)));
c_3(14) <= (s_2(13) and c_2(14)) or (a(10) and (s_2(13) xor c_2(14)));
c_3(15) <= (s_2(13) and c_2(14)) or (a(11) and (s_2(13) xor c_2(14)));

c_4 <= (z_3(1) and v_3(1)) or ((z_3(1) xor v_3(1)) and (z_3(0) and v_3(0)));
p(1 downto 0) <= z_3(3 downto 2) xor v_3(3 downto 2);
p(16 downto 2) <= s_3(14 downto 0) xor c_3(14 downto 0);
p(17) <= s_3(14) xor c_3(15);
g(1 downto 0) <= z_3(3 downto 2) and v_3(3 downto 2);
g(16 downto 2) <= s_3(14 downto 0) and c_3(14 downto 0);
c_m(0) <= c_4;

process (p,g,c_m,c_4)
begin
c_m(0) <= c_4;
            c_m(1) <= g(0) or (c_m(0) and p(0));
            inst : for i in 1 to 16 loop
                   c_m(i + 1) <= g(i) or (p(i) and c_m(i));
                   end loop;
end process;
temp_1 <= p(17 downto 0) xor c_m(17 downto 0);
det : process (temp_1)
begin
if (temp_1 = "11111111111111111") then
            m_1 <= not temp_1(17 downto 6);
            else
            m_1 <= temp_1(17 downto 6);
            end if;
            end process;

end Behavioral;
```

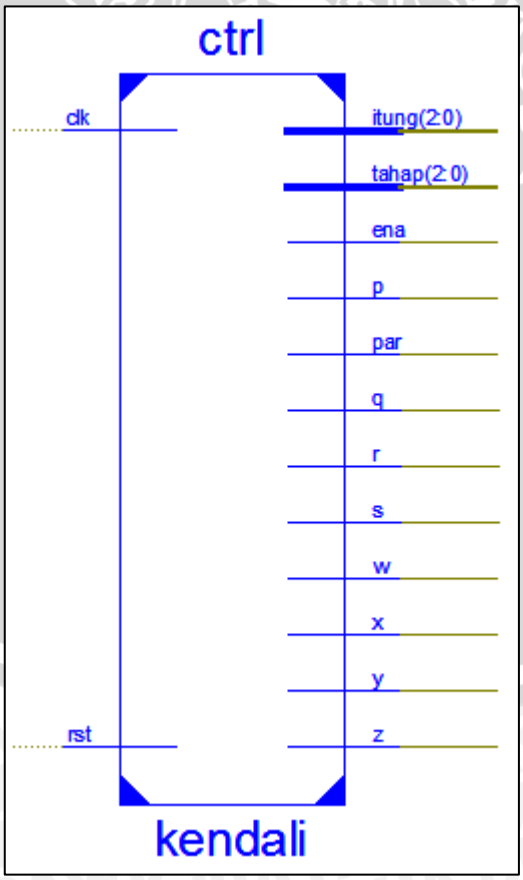
L.22 *User Constraint Files* (*pins.ucf*)

```
NET "clk" LOC = B8;
NET "div" LOC = M13;
NET "rst" LOC = B18;
NET "data_in<0>" LOC = L15; #| IOSTANDARD = LVCMOS33;
NET "data_in<1>" LOC = K12; #| IOSTANDARD = LVCMOS33;
NET "data_in<2>" LOC = L17; #| IOSTANDARD = LVCMOS33;
NET "data_in<3>" LOC = M15; #| IOSTANDARD = LVCMOS33;

NET "serial<0>" LOC = J14; #| IOSTANDARD = LVCMOS33;
NET "serial<1>" LOC = J15; #| IOSTANDARD = LVCMOS33;
NET "serial<2>" LOC = K15; #| IOSTANDARD = LVCMOS33;
NET "serial<3>" LOC = K14; #| IOSTANDARD = LVCMOS33;
NET "serial<4>" LOC = E17; #| IOSTANDARD = LVCMOS33;
NET "serial<5>" LOC = P15; #| IOSTANDARD = LVCMOS33;
NET "serial<6>" LOC = F4; #| IOSTANDARD = LVCMOS33;
NET "serial<7>" LOC = R4; #| IOSTANDARD = LVCMOS33;
```

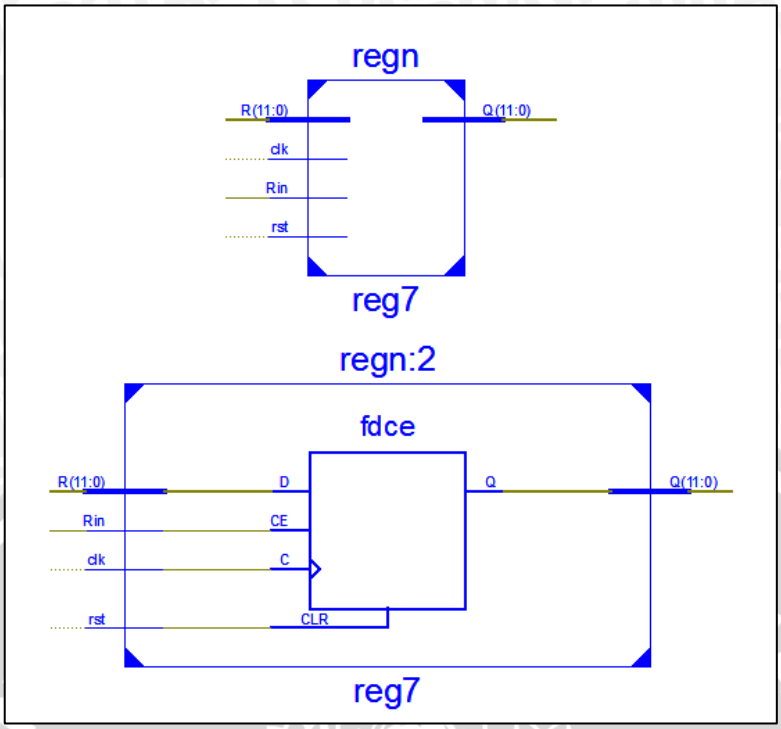
L.23 Gambar RTL (*Register-transfer Level*) hasil perancangan



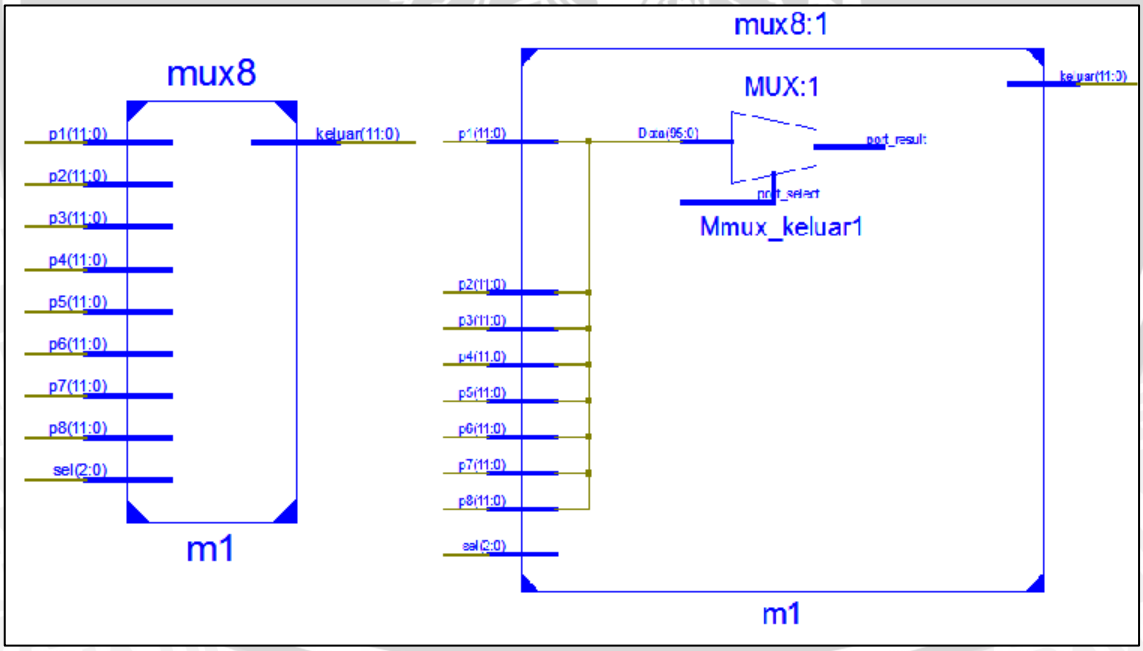
Top Level



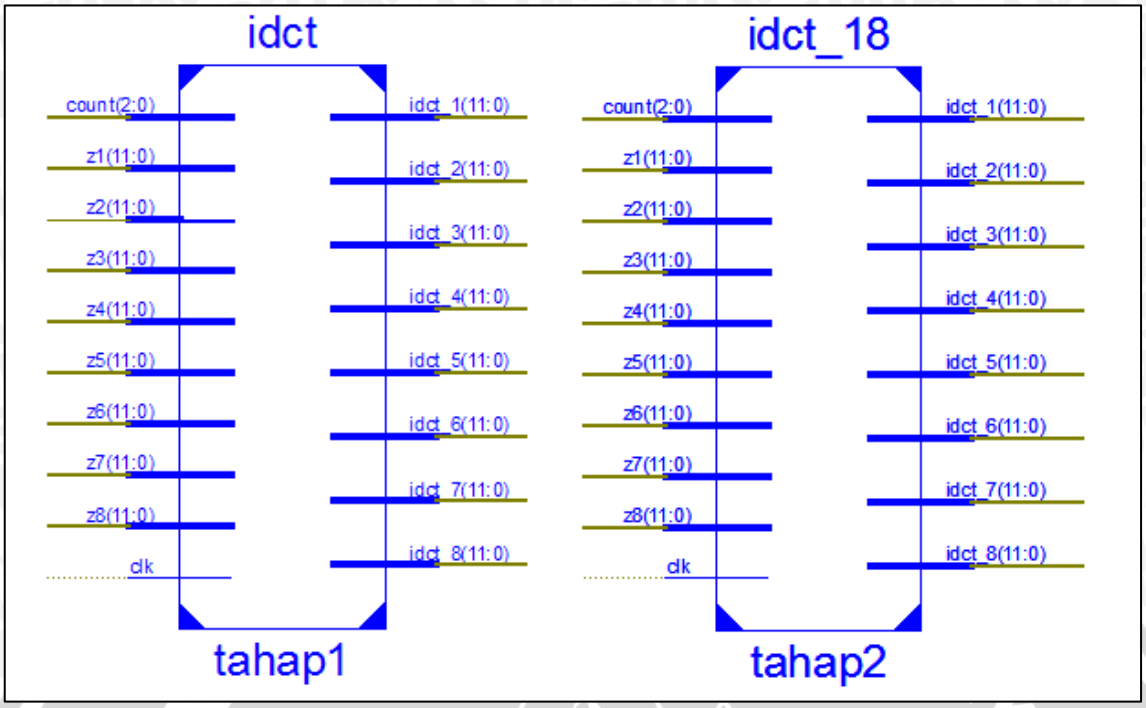
Control Signal



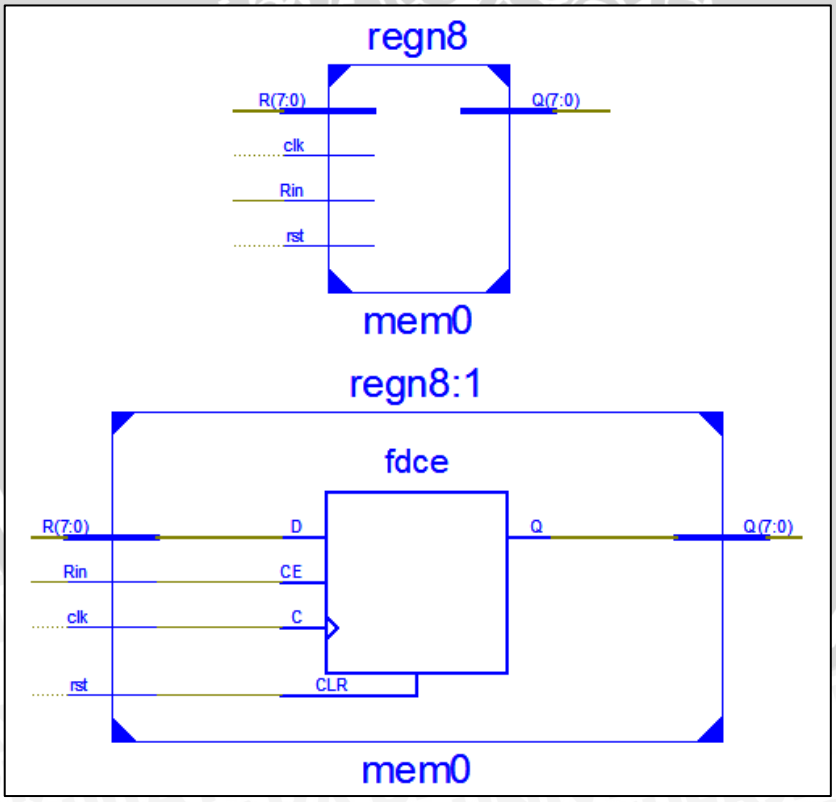
Register 12 bit



Multiplekser 8 ke 1



IDCT Blok 1 dan 2



Register 8 bit