STUDI PERANCANGAN *PARALLEL CRC 32 BIT* MENGGUNAKAN BAHASA VHDL

SKRIPSI

JURUSAN TEKNIK ELEKTRO

Diajukan untuk memenuhi sebagian persyaratan

Memperoleh gelar Sarjana Teknik

Disusun oleh :

PRIMA WIDYAWATI WARDANINGSIH

NIM 0210630098

DEPARTEMEN PENDIDIKAN NASIONAL

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

JURUSAN TEKNIK ELEKTRO

2009

LEMBAR PERSETUJUAN

STUDI PERANCANGAN *PARALLEL CRC 32 BIT* MENGGUNAKAN BAHASA VHDL

SKRIPSI

JURUSAN TEKNIK ELEKTRO

Diajukan untuk memenuhi sebagian persyaratan

Memperoleh gelar Sarjana Teknik

Disusun oleh :

PRIMA WIDYAWATI WARDANINGSIH

NIM 0210630098

DOSEN PEMBIMBING :

Suprapto, ST. MT	Panca Mudjirahardjo., ST. MT.
NIP. 19710727 199603 1 001	NIP. 19700329 200012 1 001

LEMBAR PENGESAHAN

STUDI PERANCANGAN *PARALLEL CRC 32 BIT* MENGGUNAKAN BAHASA VHDL

SKRIPSI

JURUSAN TEKNIK ELEKTRO

Diajukan untuk memenuhi sebagian persyaratan

memperoleh gelar Sarjana Teknik

Disusun oleh :

PRIMA WIDYAWATI WARDANINGSIH

NIM 0210630098

Skripsi ini telah diuji dan dinyatakan lulus pada

tanggal 4 Agustus 2009

MAJELIS PENGUJI

Adharul Muttaqin, ST., MT	Ir. Ponco Siwindarto, M.Eng.,Sc.
NIP. 19760121 200501 1 001	NIP. 19590304 198903 1 001

Dr. Agung Darmawansyah, ST., MT.
NIP. 19721218 199903 1 002

Mengetahui,
Ketua Jurusan Teknik Elektro FT UB

Ir. Heru Nurwarsito, M. Kom.
NIP. 131 837 966

PENGANTAR

Alhamdulillah, segala puji syukur kehadirat Allah SWT, Tuhan semesta alam, atas segala bimbingan, kasih sayang, dan hidayah-Nya. Dan atas ijin_Nya pula , akhirnya penulis dapat menyelesaikan skripsi ini yang berjudul **"Studi Perancangan** *Parallel CRC 32 Bit* **Menggunakan Bahasa VHDL"** dengan baik.

Tidak banyak yang bisa penulis sampikan kecuali ungkapan terima kasih kepada berbagai pihak yang telah berjasa memberikan dukungan, bimbingan, dan arahan hingga penulisan tugas akhir ini dapat selesai. Semoga segala ketulusan tersebut dapat dibalas dengan balasan yang terbaik di sisi Allah SWT.

Terima kasih kepada Bapak Ir. Heru Nurwarsito, M.Kom selaku Ketua Jurusan Teknik Elektro Brawijaya. Kepada Bapak Suprapto, ST. MT dan Bapak Panca Mudjirahardjo, ST. MT. selaku pembimbing skripsi ini atas bimbingan dan pengarahan yang diberikan.

Terima kasih yang tak terkira untuk bapak ibu tercinta atas doa, perhatian dan kepercayaan yang diberikan. Kepercayaan bahwa suatu saat nanti aku bisa memenuhi setiap janji dengan jalan pilihanku sendiri. Untuk *my two little brothers* yang selalu punya cara-cara "ajaib" untuk menghiburku. Terima kasih. Kepada semua teman-teman ,orang-orang terdekat yang baik hati, soleh dan solehah. Yang tak pernah mengharapkan balasan atas semua kebaikan yang telah dilakukan dan tak ingin namanya disebut-sebut atas kebajikan-kebajikan yang telah dilakukannya. Semoga mendapatkan balasan terindah di sisi Allah SWT.

Atas kekurangan-kekurangan yang mungkin ada dalam tugas akhir ini, penulis sangat mengharapkan adanya kritik dan saran yang dapat lebih menyempurnakan karya ini. Semoga apa yang dituliskan dalam tugas akhir ini memberikan banyak manfaat bagi penulis terutama dan bagi semua pihak apabila memungkinkan.

Malang

Awal September 2009

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

ABSTRAKSI

PRIMA WIDYAWATI W., Jurusan Teknik Elektro, Fakultas Teknik, Universitas Brawijaya, Juli 2009, *Studi Perancangan Parallel CRC 32 Bit Menggunakan Bahasa VHDL,* **Dosen Pembimbing : Suprapto, ST. MT. dan Panca Mudjirahardjo, ST. MT.**

Pada proses transmisi data sering terjadi kerusakan atau kesalahan data yang menyebabakan data hasil tranmisi berkurang keakuratannya. Padahal pada era sekarang ini kebutuhan akan data dan infUntuk mencegah hal itu maka diperlukan sebuah metode yang dapat mendeteksi kesalahan pada saat proses transmisi data berlangsung.

Ethernet sebagai salah satu teknologi protokol transportasi data menggunakan *Cyclic Redudancy Codes* (CRC) 32 bit sebagai metode pendeteksian kesalahan. Metode pendeteksian kesalahan ini akan diimplemetasikan secara hardware dengan menggunakan VHDL. Hasil penelitian menunjukkan pengimplemetasian sebuah algoritma pada sebuah hardware akan menghasilakan kecepatan eksekusi lebih cepat dibandingkan dengan pengimplementasian secara software. Agar hasil pengimplementasian ini dapat memenuhi kebutuhan *high-speed aplication*, pengimplentasian dilakukan secara paralel.

Perancangan *Parallel CRC 32 bit* ini terdiri atas 47 pin utama, 14 pin untuk masukan dan 33 pin untuk keluaran, serta pin untuk catu daya dan *ground.* Dengan 8 buah pin untuk data masukan, *Parallel CRC 32 bit* ini mampu menghitung nilai *checksum* dari 8 bit data masukan secara bersamaan.

Hasil pengimplemntasian *Parallel CRC 32 bit* ini menunjukkan frekuensi maksimum yang dihasilkan sebesar 128,535 MHz Gbps dan menghabiskan 1.805 logic gate dengan *maxsimum delay* sebesar 3,117 ns. Hasil pengujian *validitas checksum* menunjukkan *Parallel CRC 32 bit* ini memiliki kevalidatan checksum sesuai dengan standart CRC untuk aplikasi Ethernet dengan throughput yang dihasilkan sebesar 1,03. Sehingga dengan throughput sebesar itu *Parallel CRC 32 bit* ini dapat diimplementasikan pada Gigabit Ethernet.

Kata kunci : *error detection, parallel CRC,* **Gigabit Ethernet, VHDL, FPGA**

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi, dewasa ini berkembang sangat pesat memungkinkan kita dapat mengakses informasi dari belahan bumi manapun dengan sangat cepat dan murah.

Pada saat proses transmisi data sering terjadi kesalahan atau kerusakan data. Hal ini akan menyebabkan informasi yang kita terima tidak lagi akurat. Untuk mencegah hal itu maka diperlukan sebuah metode yang dapat mendeteksi kesalahan pada saat proses tranmisi data berlangsung.

Ethernet sebagai teknologi protokol transportasi data untuk jaringan komputer merupakan salah satu produk dari teknologi informasi yang memiliki kelebihan dari segi biayanya yang rendah, dan kemudahan dalam instalasi serta perawatannya. Untuk mendeteksi kesalahan, Ethernet menggunakan *error detection* yang dinamakan *Cyclic Redudancy Codes* (CRC) 32 bit sesuai dengan standart IEEE 802.3.

Cyclic Redudancy Codes (CRC) merupakan *error detection* yang digunakan secara luas dalam teknologi komunikasi baik komunikasi data ataupun *networking*. CRC mempunyai kemampuan untuk mendeteksi kesalahan secara *attractive* dengan tingkat keakuratan dan kecepatan yang tinggi, terutama dalam mendeteksi *burst errors*.

Pengimplementasian CRC dapat dilakukan baik secara software maupun hardware. Pada pengimplentasian CRC secara software memiliki kelemahan terutama dalam hal *troughput* yang sangat rendah. Dalam implementasi CRC ke dalam hardware pada awalnya dilakukan secara serial dengan menggunakan *simple shift register* dan gerbang XOR. Namun pada perkembangannya pengimplementasian ini tidak dapat memenuhi apabila diterapkan pada *high speed aplication*. Solusi yang ditawarkan untuk mengatasi hal ini dengan cara mengimplentasikan CRC ke dalam hardware secara paralel.

Dalam tugas akhir ini akan dilakukan desain dan implementasi *Parallel CRC-32 bit* ke dalam sebuah chip sebagai skema pendeteksian kesalahan yang

dipakai dalam Gigabit Ethernet. Pengimplementasian dilakukan dengan menggunakan bahasa VHDL (*Very High Speed Integrated Circuit (VHSIC) Hardware Description Language*) memanfaatkan *Xilinx ISE WebPACK 7.1i*.

1.2 Rumusan Masalah

Mengacu pada permasalahan yang telah dipaparkan pada latar belakang di atas, maka kita dapat merumuskan masalah sebagai berikut :

a. Bagaiamana mendesain dan mengimplementasikan *Parallel CRC-32 bit* dengan menggunakan VHDL *(Very High Speed Integrated Circuit (VHSIC) Hardware Description Language)* dengan memanfaatkan *Xilinx ISE WebPACK 7.1i.*
b. Bagaimana menguji hasil implementasi dengan menggunakan *ModelSim Simulator.*

1.3 Batasan Masalah

Pada perancangan kali ini batasan masalah mencakup :

a. Pembahasan *error detection* secara terperinci hanya dilakukan pada Ethernet.
b. Pembahasan algoritma *CRC* secara terperinci hanya dilakukan pada *parallel* CRC yang akan digunakan pada perancangan *Parallel CRC 32 bit* untuk aplikasi Ethernet.
c. Desain dan implementasi algoritma CRC menggunakan VHDL *(Very High Speed Integrated Circuit (VHSIC) Hardware Description Language)* dengan memanfaatkan *Xilinx ISE WebPACK 7.1i.*

1.4 Tujuan

Tujuan perancangan ini adalah untuk merancang *Parallel CRC-32 bit* yang dapat diimplementasikan untuk aplikasi *error detection* pada Ethernet menggunakan bahasa pemrograman VHDL dengan memanfaatkan *Xilinx ISE WebPACK 7.1i.*

1.5 Manfaat

Manfaat dari pengimplementasian *Parallel CRC-32 bit* ini adalah untuk menghasilkan *CRC core* yang dapat dimanfaatkan untuk aplikasi *error detection* pada Gigabit Ethernet.

1.6 Sistematika Penulisan

Sistematika penulisan untuk Laporan Tugas Akhir ini adalah sebagai berikut :

BAB I : Pendahuluan

Meliputi latar belakang, rumusan masalah, batasan masalah, tujuan, dan sistematika penulisan.

BAB II : Dasar Teori

Berisi seluruh teori yang menunjang dalam perancangan dan pemngimpletasian.

BAB III : Metodologi perancangan

Menjelaskan tentang sistematika (alur) berfikir dalam proses perancangan dan pengimplementasian.

BAB IV : Perancangan dan Implementasi

Membahas tentang desain IC dan implementasiannya menggunakan program VHDL.

BAB V : Pengujian dan Analisis

Memuat hasil pengujian terhadap desain dan analisa terhadap hasil pengujian tadi.

BAB VI : Penutup

Meliputi kesimpulan dan saran.

BAB II
DASAR TEORI

2.1 Ethernet

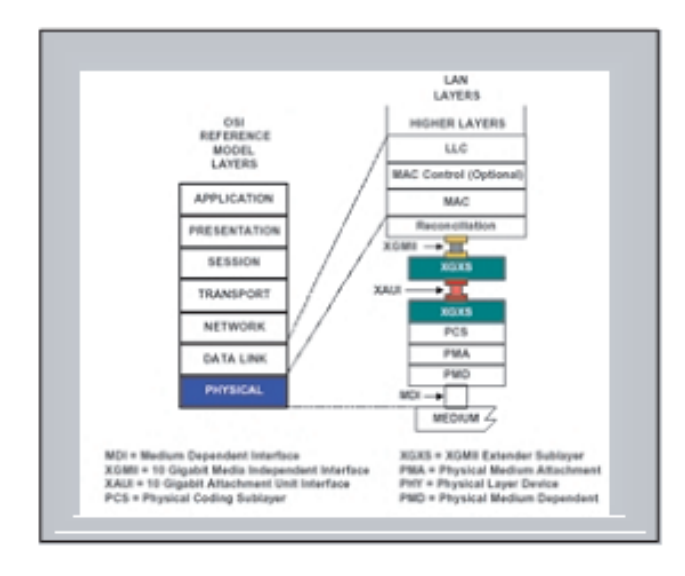
Ethernet adalah sistem jaringan yang dibuat dan dipatenkan perusahaan Xerox. Kecepatan Ethernet pada awal dikembangkan hanya 3 Mbps dan dikenali sebagai Experimental Ethernet. Ethernet adalah implementasi metoda CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) yang dikembangkan tahun 1960 pada proyek wireless ALOHA di Hawaii University diatas kabel coaxial. Standarisasi sistem ethernet dilakukan sejak tahun 1978 oleh IEEE. Kecepatan transmisi data di ethernet sampai saat ini adalah 10 sampai 100 Mbps. Saat ini yang umum ada dipasaran adalah ethernet berkecepatan 10 Mbps – 100 Mbps yang biasa disebut seri 10Base / 100Base. Ada bermacam-macam jenis 10Base diantaranya adalah: 10Base5, 10Base2, 10BaseT, dan 10BaseF. Teknologi ini semakin berkembang menjadi Gigabit Ethernet (GbE) dengan kecepatan 1000 Mbps atau 1 Gbps (*Gigabit per second*), hingga saat ini dengan adanya 10 Gigabit Ethernet (10GbE) dengan kecepatan 10.000 Mbps atau 10 Gbps

Pada metoda CSMA/CD, sebuah host komputer yang akan mengirim data ke jaringan pertama-tama memastikan bahwa jaringan sedang tidak dipakai untuk *transfer* dari dan oleh host komputer lainnya. Jika pada tahap pengecekan ditemukan transmisi data lain dan terjadi tabrakan (*collision*), maka host komputer tersebut diharuskan mengulang permohonan (*request*) pengiriman pada selang waktu berikutnya yang dilakukan secara acak (*random*). Teknik ini disebut dengan *backoff algorithm* Dengan demikian maka jaringan efektif bisa digunakan secara bergantian.

Untuk menentukan pada posisi mana sebuah host komputer berada, maka tiap-tiap perangkat ethernet diberikan alamat (*address*) sepanjang 48 bit yang unik (hanya satu di dunia). Informasi alamat disimpan dalam chip yang biasanya nampak pada saat kompiuter di start dalam urutan angka berbasis 16. 48 bit angka agar mudah dimengerti dikelompokkan masing-masing 8 bit untuk menyatakan bilangan berbasis 16 seperti contoh di atas (00 40 05 61 20 e6), 3 angka di depan

adalah kode perusahaan pembuat chip tersebut. Chip di atas dibuat oleh ANI Communications Inc.

Hubungan antara arsitekur Ethernet dengan layer standart OSI diperlihatkan dalam Gambar 2.1. Dalam Gambar 2.1 diperlihatkan, dari 7 layer yang ada (Application Layer, Presentation Layer, Session Layer, Transport Layer, Network Layer, Data Link Layer dan Physical Layer), arsitektur Ethernet berbasis 2 layer terbawah OSI. Ethernet Physical Layer (PHY) sesuai dengan Physical Layer pada OSI, dan Ethernet Media Access Control (MAC) sesuai dengan Data Link Layer pada OSI.



Gambar 2.1. Arsitektur Gigabit Ethernet dengan referensi layer OSI
Sumber : Agilent, 2003:4

2.1.1 Arsitektur Ethernet

Dalam arsitektur ethernet, terdapat beberapa layer antara lain :

a. *Medium Access Control* (MAC)

Sublayer MAC menyediakan koneksi antara MAC *client* dan *peer station*. Sublayer ini bertanggung jawab untuk inisialisasi, kontrol serta mengatur koneksi dengan *peer station.*

b. Sublayer *Reconciliation*

Sublayer *reconciliation* bertindak sebagai penerjemah perintah. Sublayer ini memetakan terminologi dan perintah yang digunakan dalam layer MAC ke dalam format elektrik yang sesuai untuk *physical layer*.

c. *Media Independent Interface* (MII)

MII menyediakan antarmuka standar antara layer MAC dan *physical layer*. MII mengisolasi layer MAC dan *physical* layer sehingga memungkinkan layer MAC untuk digunakan dengan berbagai tipe *physical layer*.

d. *Physical Coding Sublayer* (PCS)

PCS bertanggung jawab untuk pengkodean *data stream* dari dan menuju layer MAC.

e. *Physical Medium Attachment* (PMA)

PMA bertugas untuk serialisasi grup kode kedalam bit *stream* serta untuk sinkronisasi data yang telah dikodekan.

f. *Physical Medium Dependent* (PMD)

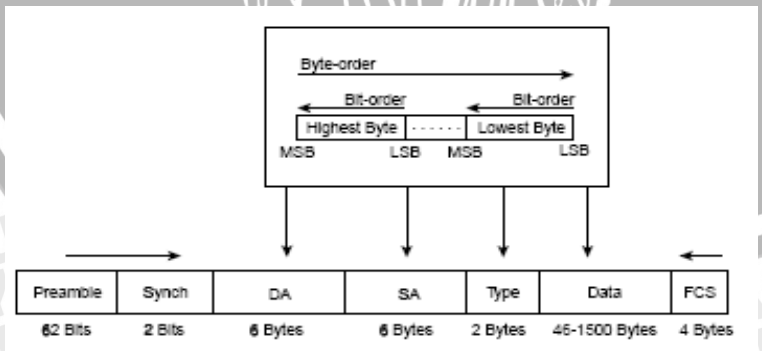
PMD bertanggung jawab untuk transmisi sinyal. PMD yang berbeda dapat digunakan untuk jarak jangkau yang berbeda pula.

g. *Medium Dependent Interface* (MDI)

MDI berhubungan dengan konektor. MDI memiliki tipe konektor yang berbeda untuk tipe media dan PMD yang berbeda.

2.1.2 Ethernet Frame

Isi frame data dari Ethernet dapat dilihat dalam Gambar 2.2. Dalam Gambar 2.2 juga diperlihatkan bagaimana tranmisi tiap-tip bagian frame.. Frame data ini dibangkitkan oleh *Media Access Control* (MAC) layer sekaligus juga mengatur pengiriman frame data tersebut dari transmitter ke receiver.



Gambar 2.2 Frame data Ethernet

Sumber : Prasimax, 2002:19

Penjelasan tiap-tiap bagian frame adalah sebagai berikut

a. Preamble

Setiap station pada Ethernet network memiliki pembangkitan clock sendiri-sendiri, yang berbeda antara satu dengan yang lain. Padahal agar bisa terjadi komunikasi antar station pembangkitan clocknya harus serempak. Dan preamble memfasilitasi ini. Preamble terdiri dari 8 byte dengan 2 angka terakhir adalah 11. Dua bit terakhir ini merupakan sinyal yang menandakan bahwa preamble sudah berakhir dan *destination address* dimulai.

b. *Destination Address* (DA)

Dalam destination address ini berisi alamat tujuan, kemana frame data akan dikirim. Besarnya 6 byte. Bagian ini dipakai oleh MAC station penerima untuk menentukan apakah data yang masuk dialamatkan pada node tersebut atau tidak. Apabila station penerima mendeteksi kesamaan antara DA dengan alamat nodenya maka data tersebut diterima apabila tidak sama data tersebut ditolak.

c. *Source Address* (SA)

Source address berisi alamat asal dari frame data. Bagian digunakan oleh station penerima untuk mengetahui darimana frame data berasal. Besarnya sama seperti destination addres yaitu sebesar 6 bytes .

d. Type

Bagian ini berisi type dari frame data. Besarnya 2 bytes

e. Data

Bagian ini berisi data yang sebenarnya yang akan dikirimkan. Lebar minimal dari data field sebesar 46 bytes dan maksimum 1500 bytes. Hanya bagian data field ini yang sampai ke *application* layer. Sedang header dan trailer yang lain akan ditanggalkan pada layer-layer sebelumnya.

f. *Frame Check Sequence* (FCS)

FCS ini besarnya 4 bytes yang terdiri atas *checksum* yang digunakan untuk *error detection*. Pada Ethernet memakai standart CRC-32 untuk mendeteksi error. CRC dibangkitkan pada station pengirim yang kemudian ditambahkan ke bagian akhir dari frame data. Standart yang dipakai antara station pengirim dan penerima harus sama agar terjadi komunikasi. Kesalahan yang dapat

dideteksi bagian FCS ini hanya kesalahan yang disebabkan oleh network bukan *human error*. Apabila dideteksi adaya kerusakan data maka station penerima akan meminta pengiriman ulang data kepada station pengirim. *Frame* data yang akan dihitung nilai CRCnya adalah *Destination Address* (DA) sampai data.

2.2 *Error Detection*

Pada proses komunikasi data tidak akan pernah lepas dari kesalahan, baik itu dalam proses pengiriman maupun pada waktu penerimaan data. Kerusakan data ini disebabkan oleh beberapa faktor. Bisa dikarenakan pengaruh medium tranmisi, jarak tranmisi, atau format data itu sendiri yang rawan mengalami kerusakan. Kerusakan data akan menyebabkan data yang dikirim dengan data yang diterima bisa berbeda sama sekali sehingga akan terjadi kesalahan penangkapan informasi pada bagian penerima. Apalagi untuk data biner, yang hanya mengenal nilai ’0’ dan ’1’, kerusakan pada salah satu bitnya saja yang awalnya ’1’ berubah menjadi ’0’ dapat menyebabkan pesan yang diterima berbeda.

Error control berfungsi untuk mengontrol terjadinya kesalahan yang terjadi pada waktu data dikirimkan. *Error control* biasanya dilakukan dengan cara melakukan *error detection* atau *error corection*. Untuk *error detection* apabila pada bagian *receiver* mendeteksi adanya kerusakan data akan meminta bagian transmitter untuk mengirmkan data kembali. Sedang untuk *error corection* begitu dideteksi ada kerusakan data, data yang rusak akan segera diperbaiki menggunakan algoritma tertentu. Pada arsitektur protokol OSI, *error control* merupakan tugas dari lapidan data link. Pada lapisan tersebut disediakan space sekian byte sebagai *error control.*

Keefektifan sebuah teknik *error detection* dapat dipengaruhi oleh beberapa faktor di bawah ini :

a. Lebar minimum dari *error detection code,* artinya jumlah terkecil kesalahan yang dapat dideteksi dari satu *codeword.*
b. Kemampuan mendeteksi deretan kesalahan (*burst error*), panjang minimum *burst error* yang dapat dideteksi

c. Probability dari pola kesalahan *random bit* sebagai kesalahan bebas.

Beberapa jenis kesalahan yang biasanya terjadi pada *tranmisted data* antara lain :

a. Kesalahan bit tunggal (*single bit error*)
b. Kesalahan bit ganda (*two-bit error*)
c. Kesalahan dengan jumlah bit ganjil
d. Deretan kesalahan *(burst error)*

2.3 CRC (*Cyclic Redudandy Check*)

2.3.1 Algoritma Dasar CRC

CRC digunakan dalam berbagai aplikasi jaringan computer dan data storage devices. CRC sering digunakan karena CRC bisa menyediakan *error detection* yang murah dan efektif. Algoritma CRC ini sering digunakan pada system komunikasi data yang mengirim blok data dalam jumlah yang besar..

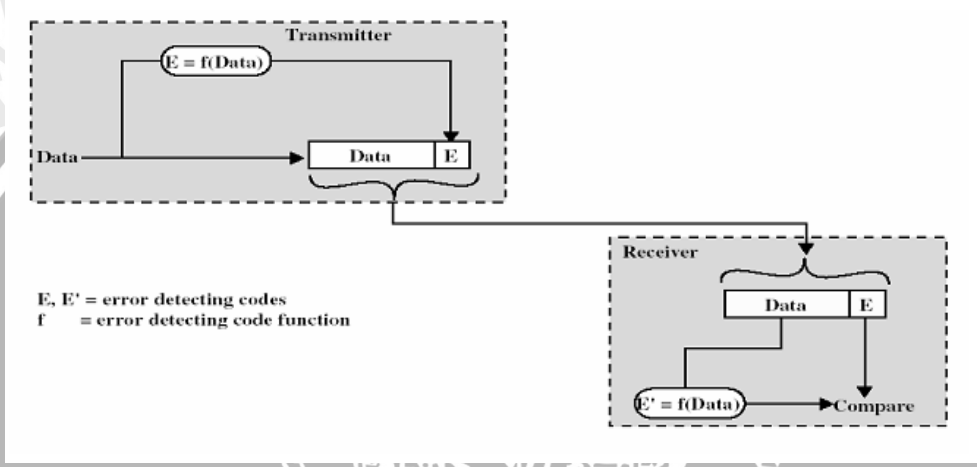
Beberapa keunggulan CRC apabila dibandingkan dengan algortima yang lain antara lain, dapat mendeteksi :

a. Semua kesalahan bit tunggal.
b. Semua kesalahan bit ganda, selama C(x) memiliki min 3 suku.
c. Semua kesalahan dgn jumlah bit ganjil, selama C(x) mengandung faktor (x+1).
d. Semua deretan kesalahan (burst error) dgn panjang < k, pangkat tertinggi C(x).
e. Sebagian besar burst error yg lebih panjang (99,9 %).

Ide dasar CRC adalah bagaimana menampilkan sebuah pesan (data) ke dalam bentuk angka biner, dan kemudian membaginya dengan sebuah angka tertentu. Angka pembagi ini biasa disebut dengan *generator polynomial*. *Generator polynomial* yang dipakai antara *transmitter* dan *receiver* haruslah sama agar proses komunikasi data dapat berjalan. Sisa hasil bagi pembagian tersebut akan menjadi *checksum*. *Checksum* ini akan ditambahkan ke dalam pesan tadi, baru setelah itu data ditransmisikan. Setelah pesan diterima, dilakukan pembagian

lagi. Apabila hasil baginya,biasa disebut *remainder*, sama dengan 0 berarti data tidak mengalami kerusakan. Proses penghitungan CRC pada bagian transmitter dan receiver dapat dilihat dalam Gambar 2.3.

Yang perlu digarisbawahi disini adalah CRC hanya bisa digunakan untuk mendeteksi apakah ada kerusakan atau tidak. CRC tidak bisa menetukan dimana letak kesalahannya dan memperbaikinya. Untuk dapat melakukan hal itu diperlukan diimplementasikan algoritma yang lain.



Gambar 2.3 Proses CRC
Sumber : William Staling

Seperti yang telah disebutkan diatas algoritma CRC menggunakan konsep matematis aritmatika modulo-2 dan polynomial untuk melakukan perhitungannya. Contohnya untuk data = {1010001101} perhitungan nilai checksum CRCnya sebagai berikut :

1. Diketahui : message M = 1010001101 (10 bit)
   pattern P = 110101 (6 bit)
   FCS R = dikalkulasi (5 bit)
2. Message M dikalikan dengan 2^5 , maka : 101000110100000
3. Kemudian dibagi dengan P :

```
            110101011  à Q
P ? 110101 ⌈101000110100000  à M
           110101 ⊕
           111011
           110101 ⊕
           111010
```

```
110101
------ ⊕
111110

110101
------ ⊕
101100

110101
------ ⊕
110010

110101
------ ⊕
1110  à  R
```

Keterangan : ⊕ adalah gerbang XOR

4. Remainder (R = 01110) ditambahkan ke 2^n M untuk mendapatkan T = 10100011010110, yang ditransmisi [$T = 2^n M + R$].

   Jika tidak ada error, maka receiver menerima T secara utuh. Frame yang diterima dibagi dengan P :

```
          110101010
       ----------
110101 | 10100011010110
       110101
       ------ ⊕
         111011
         110101
         ------ ⊕
           111010
           110101
           ------ ⊕
             111110
             110101
             ------ ⊕
               101111
               110101
               ------ ⊕
                 110101
                 110101
                 ------ ⊕
                   000000 à R
```

Karena tidak ada remainder maka dianggap tidak ada error.

2.3.2 Standart Generator Polynomial CRC

Pemilihan generator polynomial (G) merupakan parameter kunci yang menentukan keberhasikan dari CRC code. Dengan kata lain akan menentukan keberhasilan kita dalam mendeteksi sebuah kesalahan. Terdapat beberapa standart generator polynomial sekarang ini. Dan beberapa genarator polynomial memiliki kelebihan dibandingkan yang lain.

Kekuatan sebuah polynomial CRC dalam mendeteksi sebuag kesalahan, tergantung pada panjang dari generator polynomial tersebut. Semakin panjang generator polynomial yang digunakan maka semakin hebat kekuatan dari CRC untuk mendeteksi sebuah kesalahan. Sebagai contoh kita bandingkan kekuatan dari 16 bit CRC dengan 32 bit CRC. Dengan 16 bit CRC kekuatan pendeteksian kesalahan menjadi :

$$\frac{2^{16}-1}{2^{16}} = \frac{65535}{65536} = 99.998\%$$

Sedang dengan 32 bit CRC kekuatan pendeteksian kesalan menjadi :

$$\frac{2^{32}-1}{2^{32}} = \frac{4294967295}{4294967296} = 99.99999998\%$$

Berdasarkan standart IEEE 802.3 ethernet menggunakan CRC-32 bit sebagai *error detection*. Persamaan dari CRC 32 bit adalah sebagai berikut :

$$x = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + x^{0}$$

Daftar standart model CRC beserta parametr dan jenis aplikasi implementasinya untuk CRC 32 bit selengkapnya dapat dilihat dalam Tabel 2.1. Sedangkan untuk keterangan dari masing-masing parameternya dapat dilihat dalam Tabel 2.2.

Tabel 2.1 Daftar Standart CRC dan Parameternya

No.	Nama Model CRC	Parameter	Implementasi
1.	CRC-32/ADCCP/ PKZIP	Poly : 04C11DB7 Init : FFFFFFFF RefIn : True RefOut : True XorOut : FFFFFFFF Check : CBF43926	PKZIP, Ethernet, FDDI, WinZIP
2.	CRC-32/BZIP2	Poly : 04C11DB7 Init : FFFFFFFF RefIn : False RefOut : False XorOut : FFFFFFFF Check : FC891918	BZIP2

3.	CRC-32/MPEG-2	Poly : 04C11DB7 Init : FFFFFFFF RefIn : False RefOut : False XorOut : 00000000 Check : 0376E6E7	MPEG-2
4.	CRC-32/POSIX	Poly : 04C11DB7 Init : 00000000 RefIn : False RefOut : False XorOut : FFFFFFFF Check : 765E7680 LCheck : 377A6011	POSIX
5.	JAMCRC	Poly : 04C11DB7 Init : FFFFFFFF RefIn : True RefOut : True XorOut : 00000000 Check : 340BC6D9	JAMCRC
6.	CRC-32C /ISCSI /CASTAGNOLI	Poly : 1EDC6F41 Init : FFFFFFFF RefIn : True RefOut : True XorOut : FFFFFFFF Check : E3069283	ISCSI
7.	XFER	Poly : 000000AF Init : 00000000 RefIn : False RefOut : False XorOut : 00000000 Check : BD0BE338	XFER

Sumber : Cook, Greg, 1993

Tabel 2.2 Keterangan Parameter

No.	Parameter	Keterangan
1.	Poly	Merupakan generator polynomial dari model CRC
2.	init	Merupakan init value pada register ketika algpritma akan

		dijalankan, nilainya antara ”FFFFFFFF” H atau ”00000000” H
3.	Refin	Merupakan parameter boolean. Jika bernilai ”False” maka byte input akan diproses dengan bit ke-7 menjadi MSB dan bit ke-0 menjadi LSB, tetapi jika bernilai ”True” maka masing-masing byte input akan dicerminkan (*reflected*) sebelum diproses.
4.	Refout	Merupakan parameter boolean. Jika bernilai ”False” maka nilai terakhir register langsung dimasukkan untuk tahap selanjutnya, yaitu tahap xorout, tetapi jika bernilai ”True” maka nilai terakhir register tersebut harus dicerminkan (*reflected*) terlebih dahulu.
5.	XorOut	Merupakan nilai yang akan di-xor-kan dengan nilai terakhir register sebelum nilai tersebut dikeluarkan menjadi nilai checksum
6.	Check	Merupakan nilai keluaran CRC yang digunakan untuk mengecek validitas output ketika diberi masukan ”123456789” ASCII atau dalam hexsa ”313233343536373838” H
7.	LCheck	Merupakan nilai keluaran CRC yang digunakan untuk mengecek validitas output ketika diberi masukan ”313233343536373839” H *plus* biner ”9”

Sumber : William, Ross N., 1993

2.4 *Parallel CRC*

Seperti sudah dijelaskan sebelumnya bahwa pengimplementasian CRC pada hardware secara serial tidak dapat lagi memenuhi apabila diterapkan pada *high-speed aplication*. Keuntungan melakukan implementasi secara pararel apabila dibandingkan pengimplementasian secara serial adalah kita dapat menghemat jumlah clock yang diperlukan. Sehingga implementasi CRC secara parallel akan menghasilkan operasi logic yang lebih cepat dibandingkan implementasi konvensional secara serial.

Dengan pengimplementasian CRC secara paralel kita dapat memperoleh *m* bit FCS untuk *k + m message* atau data *plus* bilangan 0 dalam setiap *w* bit *parallel CRC*. Periode yang dibutuhkan untuk menghitung *m* bit FCS untuk *w* bt parallel CRC sebesar $\frac{k+m}{w}$ *clcok period.*

Dari teori sistem linear kita ketahui bahwa *discrete time, time invariant* dapat diekspresikan sebagai berikut :

$$\begin{cases} X(i+1) &= FX(i) + GU(i) \\ Y(i) &= HX(i) + JU(i) \end{cases} \quad (1)$$

Dimana X merupakan state dari sistem, U sebagai input dan Y sebagai output. F, G, H, J untuk menunjukkan matrik dan X, Y, U digunakan juga untuk menunjukkan *colom vector* dari matrik. Solusi penyelesain pertama dari persamaan (1) adalah

$$X(i) = F^i X(0) + [F^{i-1}G \cdots FG\ G][U(0) \cdots U(i-1)]^T \quad (2)$$

Ketika *i* sama dengan *w*, penyelesaian dari persamaan (2) adalah sebagai berikut :

$$X(w) = F^w \otimes X(0) \oplus [0 \cdots 0 | d(0) \cdots d(w-1)]^T, \quad (3)$$

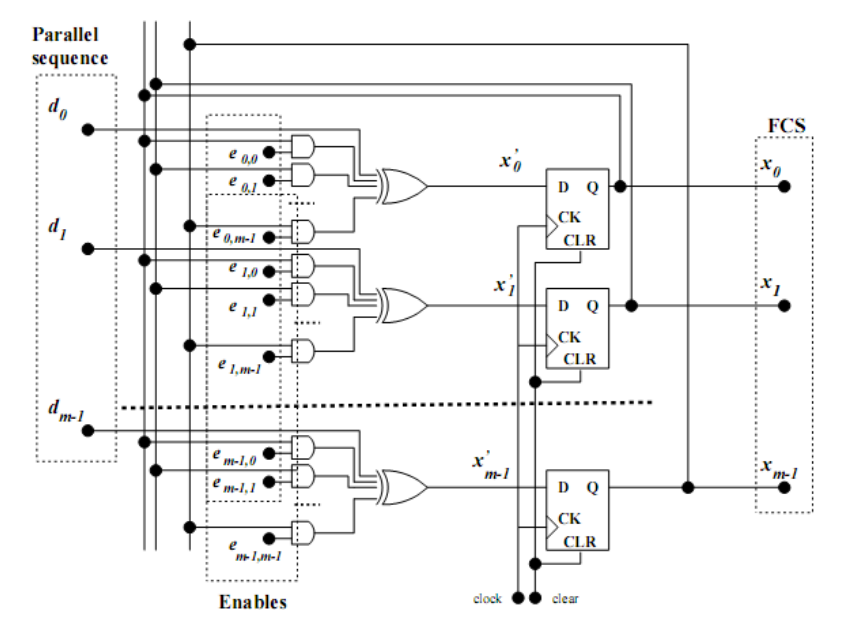
Dimana X(0) merupakan *initial state* dari FFs. Dengan menganggap sistem merupakan *time-invariant*, dapat diperoleh persamaan sebagai berikut :

$$X' = F^w \otimes X \oplus D \quad (4)$$

X' merupakan *next state* dan *X* merupakan *present state*-nya. Sedangkan D merupakan pesan atau data masukan yang akan dihitumg nilai CRCnya. F^w matrik perkalian yang merupakan komponen penting dalam perhitungan CRC secara paralel. Dengan matrik ini kita dapat menghitung *w* bit data secara bersamaan. Untuk CRC 32 bit nilia dari matrik F^w adalah sebagai berikut :

$$F^{32}_{\text{CRC-32}} = [\text{FB808B20}\ \ \text{7DC04590}\ \ \text{BEE022C8}\ \ \text{5F701164}\ \ \text{2FB808B2}\ \ \text{97DC0459}\ \ \text{B06E890C}$$
$$\text{58374486}\ \ \text{AC1BA243}\ \ \text{AD8D5A01}\ \ \text{AD462620}\ \ \text{56A31310}\ \ \text{2B518988}\ \ \text{95A8C4C4}$$
$$\text{CAD46262}\ \ \text{656A3131}\ \ \text{493593B8}\ \ \text{249AC9DC}\ \ \text{924D64EE}\ \ \text{C926B277}\ \ \text{9F13D21B}$$
$$\text{B409622D}\ \ \text{21843A36}\ \ \text{90C21D1B}\ \ \text{33E185AD}\ \ \text{627049F6}\ \ \text{313824FB}\ \ \text{E31C995D}$$
$$\text{8A0EC78E}\ \ \text{C50763C7}\ \ \text{19033AC3}\ \ \text{F7011641}\ ]^T$$

Desain implementasi *parallel CRC* secara *hardware* dapat dilihat dalam Gambar 2.4.



Gambar 2.4 Implementasi *Paralel CRC* secara *hardware*

2.5 *Galois Field*

Galois Field (GF) atau yang lebih sering disebut *field* yang mempunyai jumlah elemen terbatas. Biasa dinotasikan dengan *GF (q)*, dengan *q* adalah jumlah elemennya. Contoh sederhana dari *GF* ini adalah biner *field* yang mempunyai elemen 0 dan 1, sehingga disebut GF(2). Kita dapat membuat GF yang lebih lebar lagi dengan cara

Dua operasi dasar dari GF adalah penambahan dan perkalian. Penambahan dapat dengan mudah direalisasikan menggunakan gerbang XOR. Operasi penambahan dan perkalian dapat dilakukan apabila memenuhi syarat sebagai berikut :

a. Memiliki identitas, yaitu 0 dan 1
b. Memiliki invers, yaitu – dan /
c. Memenuhi hukum asosiatif dan distributif

2.6 Aritmetik Modulo-2

Aritmetik modulo-2 adalah aritmatik yang bekerja pada ruang biner, yaitu 0 dan 1. Secara keseluruhan aritmatik modulo-2 sama seperti operasi aritmatik pada umunya. Baik pada operasi pembagian, perkalian, penjumlahan

maupun pengurangannya tetapi pada aritmetik modulo-2 tidak mengenal sistem carry. Misalnya,

Penjumlahan :

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0

Pengurangan :

0 - 0 = 0
0 - 1 = 1
1 – 0 = 1
1 – 1 = 0

Operasi biner tanpa *carry* ini mirip dengan operasi XOR (exclusive-OR). Hal ini dapat dibuktikan dengan melihat table kebenaran dari gerbang XOR (Tabel 2.3). Sehingga operasi penambahan dan pengurangan adalah sama. Misalnya, penjumlahan dan pengurangan pada bilangan 1010 dan 001 :

1010 + 0011 = 1001
1010 – 0011 = 1001

Tabel 2.3 Tabel Kebenaran gerbang logika XOR

A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

2.7 *Polynomial*

Algoritma CRC menggunakan persamaan *polynomial* dalam hampir semua perhitungannya. *Polynomial* merupakan persamaan variable tingkat tinggi. Konsep *polynomial* pada CRC mempunyai konsep yang sama dengan konsep pada aritmatik *polynomial* pada umumnya. Secara umum persamaan sebuah *polynomial* adalah sebagai berikut :

$$M(x) = bnXn + bn\text{-}1Xn\text{-}1 + bn\text{-}2Xn\text{-}2 + \ldots + b2X2 + b1X + b01$$

Keterangan : b : nilai dari koefisien variabel x

n : tingkat derajat *polynomial*

Hanya saja pada operasi polynomial CRC menggunakan aritmatik modulo-2. Sehingga nilai koefisien dari persamaan *polynomial*-nya adalah antara 0 atau 1. Apabila pesan yang akan ditranmisikan mempunyai lebar n bit, maka derajat *polynomial* tertingginya adalah (n-1). Misal pesan yang akan dikirim adalah 101001101 yang mempunyai lebar 9 bit . Sehingga persamaan *polynomial*-nya adalah

$$M(x) = 1*x^8 + 0*x^7 + 1*x^6 + 0*x^5 + 0*x^4 + 1*x^3 + 1*x^2 + 0*x + 1*x^0$$

Dan apabila disederhanakan persamaannya menjadi :

$$M(x) = x^8 + x^6 + x^3 + x^2 + 1$$

2.8 VHDL (*VHSIC Hardware Description Language*)

2.8.1 Sejarah VHDL

VHDL pertama kali dikembangkan oleh Departemen Pertahanan Amerika Serikat pada tahun 1981. Perkembangan selanjutnya sejak tahun 19983-1985 dikembangkan baseline dari bahasa VHDL oleh intermedia, IBM dan TI. VHDL ini dapat mendeskripsikan funsi, perilaku masukan dan keluaran rangkaian digital. Tahun 1986 baseline bahasa VHDL tadi ditransfer ke pihak IEEE. Baru pada tahun 1987 diumumkan oleh IEEE dan sejak itu VHDL menjadi standar HDL oleh IEEE yang dikenal dengan nama VHDL 1076-1993. Kemudian ditinjau kembali dan disempurnakan pada tahun 1994.

VHDL adalah salah satu bahasa deskripsi perangkat keras, yang menyerupai bahasa C / C++. Dengan tools ini kita cukup mendeskripsikan *behaviour* perangkat keras yang kita rancang dalam sebuah bahasa pemrograman, dan kemudian melakukan simulasi output perangkat keras tersebut. Setelah rancangan selesai, deskripsi perangkat keras tersebut secara otomatis di *synthesis* oleh tools khusus menjadi rangkaian gerbang-gerbang logika (AND/OR, XOR.. dsb). Rancangan dalam bentuk gerbang logika ini kemudian di petakan ke rangkaian transistor untuk dilakukan proses pengaturan penempatan (*placement*) dan interkoneksi (*routing*) diatas silicon

Tetapi berbeda dengan bahasa pemrograman algoritma seperti bahasa C atau Java, dimana proses eksekusi diselesaikan satu persatu, maka pada bahasa pemrograman VHDL proses eksekusi berlangsung secara bersamaan. Hal ini dikarenakan VHDL mendesain *hardware* yang bekerja secara paralel atau bersamaan.

2.8.2 Keunggulan VHDL

Beberapa keunggulan dari VHDL antara lain :

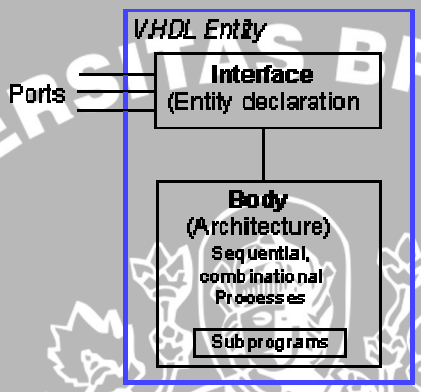
a. Mempunyai *execuetable spefication*
b. Simulasinya cepat
c. Memberi desain alternatif apabila desain kita tidak dideskripsikan dan dimplementasikan
d. Sintesis dan test generation dilakukan secara otomatis
e. Dapat meningkatkan produktifitas desain
f. Memberi umpan balik
g. Tipe data baru dapat disebutkan.

2.8.3 Sintak VHDL

Secara umum sintak VHDL terdiri dari *library*, *entity*, dan *architecture*. Blo diagram dari sintak VHDL dapat dilihat dalam Gambar 2.5. Bagian library digunakan untuk menjelaskan *library* yang diperlukan oleh bagian *architecture* dari sebuah desain. Pada bagian *entity* dijelaskan pin-pin yang digunakan sebagai input dan output. Sedangkan bagian *architecture* mendefinisikan secara detail arsitektur program yang dibuat.

```
library <library_name>;
use <library_name>;
use <library_name>;
....
entity <entity_name> is
    port (      <port_name>: <port_type>;
                <port_name>: <port_type>;
                ....
                );
```

```
end <entity_name>;

architecture <architecture_name> of <entity_name> is
    local_declaration;
    ....
begin
    program_statment;
    ....
end <architecture_name>;
```



Gambar 2.5 Blok Diagram Sintak VHDL
Sumber : Spiegel, 2001 : 4

Ada tiga pendekatan dalam merancang *architecture* dalam VHDL, antara lain :

1. *Dataflow architecture*

Pada *dataflow architecture*, rangkaian dijelaskan dengan menunjukkan hubungan input output diantara berbagai macam komponen *built-in* pada VHDL, seperti AND, OR, XOR, dan sebagainya. Sehingga pada pendekatan ini kita dapat menjelaskan secara detail bagaimana rancangan suatu rangkaian akan diimplementasikan dalam hardware. *Dataflow architecture* memiliki kelebihan yaitu proses aliran data dalam rangkaian dapat dilihat dengan menganalisis kode VHDLnya. Akan tetapi, *dataflow* bekerja dengan baik untuk rangkaian yang kecil dan sederhana. Sehingga *dataflow architecture* tidak efisien apabila diterapkan pada rangkaian yang semakin rumit dan kompleks.

2. *Behavioral architecture*

Sebagai perbandingan dengan *dataflow architecture*, *behavioral architecture* tidak menjelaskan secara detail bagaimana rancangan suatu rangkaian akan diimplementasikan dalam hardware. VHDL yang ditulis dengan

behavioral architecture tidak menunjukkan bagaimana rangkaian diimplementasikan setelah disintesis. *Behavioral architecture* menggambarkan bagaimana output rangkaian bekerja atau menunjukkan reaksi pada input rangkaian.

3. *Structural architecture*

Pada *structural architecture*, suatu rangkaian digambarkan sebagai interkoneksi sinyal sub bagian dalam rangkaian tersebut. Tiap sub bagian dapat dideskripsikan sebagai interkoneksi sinyal sub sub bagian, dan seterusnya hingga mencapai komponen yang paling sederhana yang hanya bisa dideskripsikan dengan *behavior*-nya saja.

Tipe deskripsi pernyataan pada VHDL adalah *concurrent statements*. Tipe *concurrent statements* ini ada beberapa macam, yaitu:

1. *Concurrent signal assignment statements*

Deskripsi sintaks dari bahasa pemrograman VHDL menggunakan konsep paralel atau bersamaan. Pernyataan dalam VHDL hampir serupa dengan pernyataan dalam pemrograman algoritma, akan tetapi dalam VHDL terdapat eksekusi pernyataan secara bersamaan. Pada sintaks VHDL terdapat operator "<=" yang merupakan operator *signal assignment*.

```
architecture <Architecture_name> of <Entity_name_name> is
begin
target <= statement;
target <= statement;
end <Architecture_name>;
```

2. *Conditional signal assignment statements*

Pada deskripsi pernyataan sebelumnya yaitu *concurrent signal assignment statements*, satu target hanya diasosiasikan dengan satu pernyataan. Pada *conditional signal assignment statements*, satu target dapat diasosiasikan dengan lebih dari satu pernyataan. Tiap pernyataan akan diasosiasikan dengan suatu kondisi tertentu. Sintaks untuk *conditional signal assignment statements* ditunjukkan sebagai berikut:

```
architecture <Architecture_name> of <Entity_name_name> is
begin
target <= statement when condition else
   statement when condition else statement;
end <Architecture_name>;
```

3. *Selected signal assignment statements*

Selected signal assignment statements adalah tipe ketiga dari *concurrent statements*. Deskripsi ini memiliki satu target yang diasosiasikan dengan satu pernyataan yang menentukan pilihan operasi yang akan dieksekusi. Sintaks untuk *selected signal assignment statements* ditunjukkan sebagai berikut:

```
architecture <Architecture_name> of <Entity_name_name> is
begin
with <choose_statement> select
target <= statement when choices
  statement when choices;
end <Architecture_name>;
```

4. *Process statements*

Pada dasarnya penyataan dengan *process statements* ini hampir sama dengan *concurrent signal assignment statements*. Yang membedakan adalah pada *process statement* terdapat *sequential statement*. Sintaks untuk *process statement* ini ditunjukkan sebagai berikut:

```
architecture <Architecture_name> of <Entity_name_name> is
label : process <sensitivity_list>

begin
target <= sequential_statement;
target <= sequential_statement;
end <Architecture_name>;
```

2.9 FPGA (*Field Programmable Gate Array*)

Field Programmable Device (FPD) adalah jenis IC yang digunakan untuk mengimplementasikan *hardware* digital, dimana penggunanya dapat merancang

bermacam desain dengan IC ini. FPD memiliki banyak jenis, yaitu PAL, PLD, CPLD dan salah satunya adalah FPGA. PAL, PLD dan CPLD biasanya memiliki kapasitas lebih kecil daripada FPGA dan dapat diimplementasikan dengan *Sum of Products*, dan *Produts of Sum*. FPGA biasanya berbasis Flash, SRAM, EEPROM, dan konektivitas *Anti-Fuse*. Dengan FPGA kita dapat melakukan perancangan system digital yang selanjutnya menjadi *prototype* rancangan system untuk dapat dimanufaktur. FPGA menggunakan *Hardware Description Language* (HDL) seperti VHDL atau Verilog.

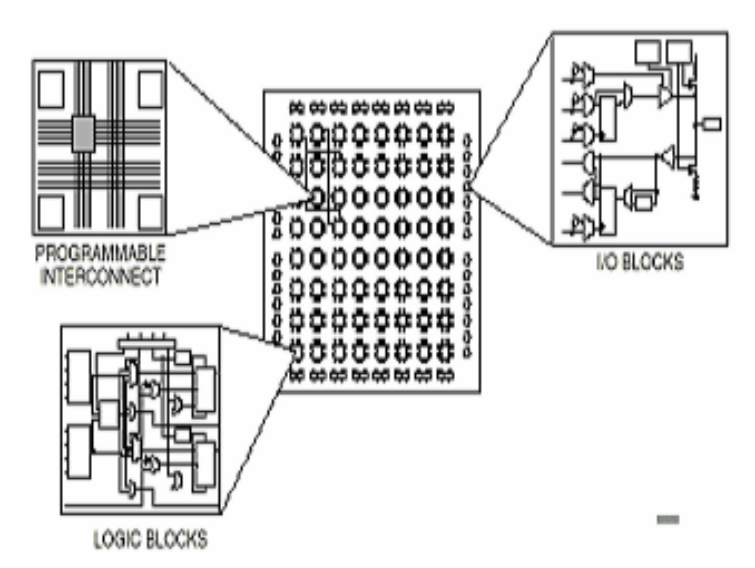
Beberapa keunggukan dari FPGA antara lain :

1. Tidak terikat dengan ketersediaan IC
2. Perancangan IC tanpa soldering
3. Desain sepenuhnya *software-based*
4. *Time to market* lebih singkat
5. Dapat menggantikan seperangkat rak penyimpan IC dan kabel jumper.
6. Ideal untuk untuk aplikasi pembuatan *prototype*
7. Bisa diimplementasikan untuk hardware yang bias dikonfigurasi ulang secara cepat.

2.9.1 Arsitektur FPGA

Seperti yang terlihat dalam Gambar 2.6 secara umum arsitektur bagian dalam dari IC FPGA terdiri atas tiga elemen utama yaitu :

1. Configurable Logic Blocks :
   – Look up table based complex structure
   – Implement the sequential circuit
2. Programmable Interconnect :
   – Berisi wire segments dan programmable switches
   – Menghubungkan anata Configurable Logic Blocks yang berbeda
3. Input/Output Block (IOB) :
   – Sebagai interface antara external package pin dari device dan internal user logic



Gambar 2.6 Arsitektur bagian dalam FPGA

2.9.2 Pemrograman IC FPGA

Seperti terlihat dalam Gambar 2.7 proses pemrograman IC FPGA adalah sebagai berikut :

1. *Synthesis*

   Pada tahap ini *VHDL Source code* dikonversi menjadi gerbang-gerbang logika menghasilkan file yang disebut *netlist.*

2. *Map, Place and Route*

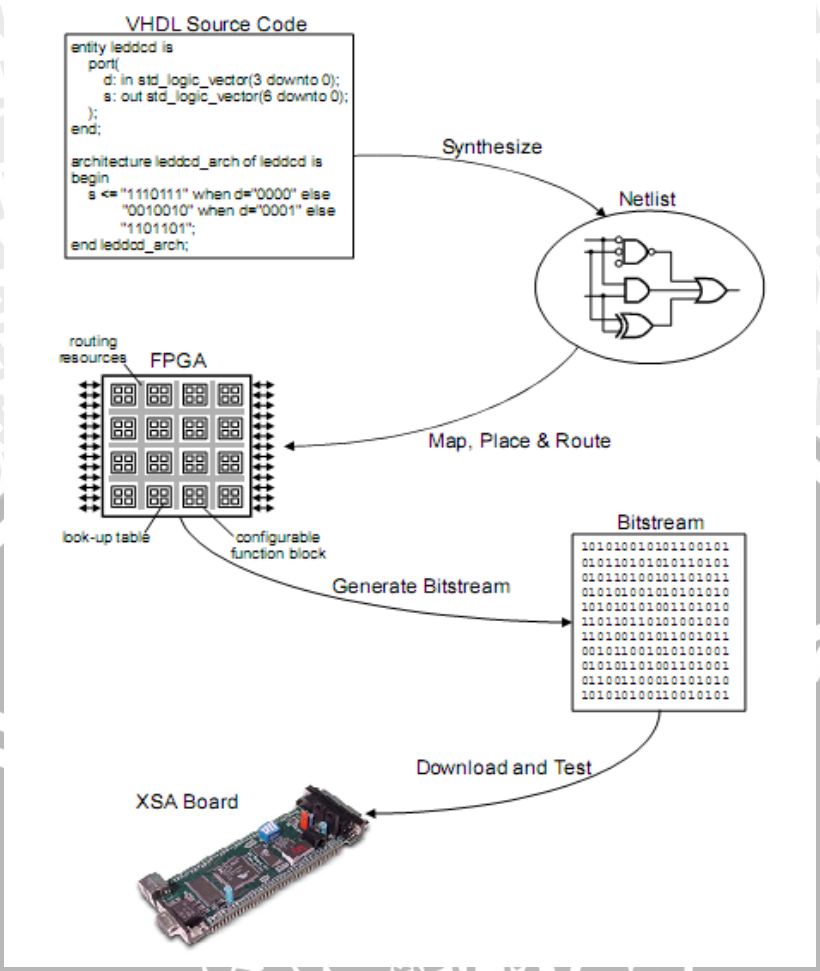
   Setelah disintesis, dilakukan proses implementasi yang meliputi proses *map, place* dan *route.* Pada tahap ini proses disesuaikan dengan spesifikasi yang disediakan *vendor*, seperti Xilinx atau Altera.

3. *Generate Bitstream*

   Setelah dimplementasi, dibuat file bitstream yang merupakan file yang akan di-*download* ke hardware (FPGA chip).

4. *Download and Test*

   Pada tahap ini file bitstream yang dihasilkan akan di-*download* ke hardware (FPGA chip). *Download* dapat dilakukan dengan koneksi serial, kabel JTAG atau diprogram dalam ROM



Gambar 2.7 Proses Pemrograman IC FPGA

Sumber : XESS Corporation, 2001: 4

BAB III
METODE PENELITIAN

Langkah-langkah yang perlu dilakukan untuk merealisasikan perancangan IC yang akan dibuat adalah sebagai berikut:

3.1 Studi Literatur

Studi literatur yang dilakukan bertujuan untuk mengkaji hal-hal yang berhubungan dengan teori-teori yang mendukung dalam perencanaan dan pengimplentasian desain ini. Teori-teori yang dikaji adalah sebagai berikut:

a. Kajian pustaka tentang Ethernet, arsitektur dan frame datanya.
b. Kajian pustaka mengenai *error detection* macam-macam error, dan *error detection* pada ethernet.
c. Kajian pustaka mengenai CRC, diantaranya tentang analisa matematis algoritma CRC, yaitu: modulo-2, polynomial, *galois field*, dan *LFSR*.
d. Kajian pustaka mengenai metode *Parallel CRC*
e. Kajian pustaka mengenai VHDL *(Very Hgh Speed Integreted Circuit (VHSIC) Hardware Description Language)*
f. Kajian pustaka mengenai *Field Programmable Gate Array* (FPGA)

3.2 Perancangan dan Implementasi

Perancangan *Parallel CRC 32 bit* ini menggunakan bahasa pemrograman VHDL dengan memanfaatkan Xillinx *ISE WebPACK 7.1i*. Perancangan *Parallel CRC 32 bit* secara umum terdiri atas 47 pin utama dan 2 pin untuk catu daya, yaitu pin vcc dan pin ground. Pin-pin utama dapat dilihat dalam Tabel 3.1.

Tabel 3.1 Konfigurasi pin-pin utama

No.	Pin	I/O	Fungsi
1.	*clk*	input	Pin untuk mengatur clock
2.	*rst*	input	Pin untuk mengatur reset
3.	*data_valid*	input	Pin untuk mengenali data paralel
4.	*crc_done*	input	Pin untuk mengatifkan proses penghitungan checksum
5.	*ref_dat*	input	Pin untuk memilih data yang pertama kali

			masuk MSB atu LSB pada input dan output data
6.	*xor_done*	input	Pin untuk melakukan fungsi xorout sebelum data crc dikeluarkan
7.	*data_in[7]*	input	Pin untuk memasukkan data masukan bit ke-7
8.	*data_in[6]*	input	Pin untuk memasukkan data masukan bit ke-6
9.	*data_in[5]*	input	Pin untuk memasukkan data masukan bit ke-5
10.	*data_in[4]*	input	Pin untuk memasukkan data masukan bit ke-4
11.	*data_in[3]*	input	Pin untuk memasukkan data masukan bit ke-3
12.	*data_in[2]*	input	Pin untuk memasukkan data masukan bit ke-2
13.	*data_in[1]*	input	Pin untuk memasukkan data masukan bit ke-1
14.	*data_in[0]*	input	Pin untuk memasukkan data masukan bit ke-0
15.	*data_out[31]*	output	Pin untuk mengeluarkan output bit ke-31
16.	*data_out[30]*	output	Pin untuk mengeluarkan output bit ke-30
17.	*data_out[29]*	output	Pin untuk mengeluarkan output bit ke-29
18.	*data_out[28]*	output	Pin untuk mengeluarkan output bit ke-28
19.	*data_out[27]*	output	Pin untuk mengeluarkan output bit ke-27
20.	*data_out[26]*	output	Pin untuk mengeluarkan output bit ke-26
21.	*data_out[25]*	output	Pin untuk mengeluarkan output bit ke-25
22.	*data_out[24]*	output	Pin untuk mengeluarkan output bit ke-24
23.	*data_out[23]*	output	Pin untuk mengeluarkan output bit ke-23
24.	*data_out[22]*	output	Pin untuk mengeluarkan output bit ke-22
25.	*data_out[21]*	output	Pin untuk mengeluarkan output bit ke-21
26.	*data_out[20]*	output	Pin untuk mengeluarkan output bit ke-20
27.	*data_out[19]*	output	Pin untuk mengeluarkan output bit ke-19
28.	*data_out[18]*	output	Pin untuk mengeluarkan output bit ke-18
29.	*data_out[17]*	output	Pin untuk mengeluarkan output bit ke-17
30.	*data_out[16]*	output	Pin untuk mengeluarkan output bit ke-16
31.	*data_out[15]*	output	Pin untuk mengeluarkan output bit ke-15
32.	*data_out[14]*	output	Pin untuk mengeluarkan output bit ke-14
33.	*data_out[13]*	output	Pin untuk mengeluarkan output bit ke-13
34.	*data_out[12]*	output	Pin untuk mengeluarkan output bit ke-12
35.	*data_out[11]*	output	Pin untuk mengeluarkan output bit ke-11
36.	*data_out[10]*	output	Pin untuk mengeluarkan output bit ke-10
37.	*data_out[9]*	output	Pin untuk mengeluarkan output bit ke-9
38.	*data_out[8]*	output	Pin untuk mengeluarkan output bit ke-8
39.	*data_out[7]*	output	Pin untuk mengeluarkan output bit ke-7
40.	*data_out[6]*	output	Pin untuk mengeluarkan output bit ke-6

41.	*data_out[5]*	output	Pin untuk mengeluarkan output bit ke-5
42.	*data_out[4]*	output	Pin untuk mengeluarkan output bit ke-4
43.	*data_out[3]*	output	Pin untuk mengeluarkan output bit ke-3
44.	*data_out[2]*	output	Pin untuk mengeluarkan output bit ke-2
45.	*data_out[1]*	output	Pin untuk mengeluarkan output bit ke-1
46.	*data_out[0]*	output	Pin untuk mengeluarkan output bit ke-0
47.	*crc_valid*	output	Pin untuk mengeluarkan data paralel

Tahap perancangan dan implementasi algoritma CRC ini ke dalam IC mengacu pada tutorial yang dikeluarkan oleh xilinx Inc antara lain :

a. Spesifikasi

Pada tahap ini *behavioral* desain rangkaian digital dideskripsikan dalam sebuah *file text*

b. Sintesis

Pada proses ini *behavioral* dan *structural* desain rangkaian dikonversi menjadi gerbang-gerbang logika dasar berdasarkan file text yang telah dideskripsikan tadi. Hasil dari proses ini berupa netlist yang sangat bergantung kepada vendor dan spesifikasi device-family, sehingga penggunaaan library harus sesuai dengan vendor.

c. Simulasi

Desain rangkaian *programmable logic* diverifikasi dengan menggunakan simulator, yaitu sebuah software yang mengkonfirmasi fungsi dan timing sebuah rangkain. Pada tahap ini simulasi ini dilakukan pengecekan kombinasi hasil dari desain rancangan digital.

d. Implementasi

Netlist sebuah desain menjelaskan gerbang logika yang digunakan dalam desain untuk vendor ataupun device-family tertentu. Setelah tahap simulasi, tahap berikutnya adalah pengimplementasikan desain ke sebuah chip pada FPGA. Tahap ini disebut *place and route.* Place adalah sebuah proses untuk memilih modul yang spesifik, sedangkan route adalah proses routing fisik yang menghubungkan antar *logic stream.*

Gambaran umum *design cycle* dalam pemngimplemntasian algoritma CRC ini adalah :

a. Spesifikasi device yang digunakan adalah *FPGA Xilinx Virtex2 XC2S40* dengan *package FG256 Speed Grade* -6.
b. Implementasi algoritma CRC menggunakan bahasa pemrograman VHDL. Pengimplementasian memakai pendekatan *behavioral* dan *structural architecture*.
c. *Logic design tool* digunakan software *Xilinx ISE WebPACK* 7.1i
d. *Synthesis Tool* yang digunakan XST.
e. Simulator yang digunakan adalah *ModelSim* dengan menggunakan *file Test Bench Wave Form*

3.3 Pengujian dan Analisa

Pada perancangan ini dilakukan dua macam pengujian terhadap 8 bit paralel CRC 32 bit yaitu :

a. Pengujian validitas checksum
b. Pengujian perfomansi IC

3.3.1 Pengujian validitas checksum

Pengujian validitas checksum, yang merupakan output dari CRC, dilakukan untuk menguji apakah keluaran dari 32-bit CRC ini sudah sesuai dengan standart *32-bit CRC Calculation* atau tidak, baik keluaran modul utama maupun modul yang lainnya. Metode pengujian dilakukan dengan simulasi menggunakan program *testbench* yaitu *test vector* yang terdapat dalam *Xilinx ISE WebPACK 7.1i* untuk masing-masing modul.

Dari simulasi tersebut dapat dilihat apakah output yang dihasilkan sesuai dengan output yang kita inginkan setelah kita menberi input sesuai dengan desain. Input yang digunakan dan output yang diharapkan dalam perancangan ini mengacu kepada standart parameter CRC-32/PKZIP untuk aplikasi Ethernet. Sehingga IC ini dapat terbukti tingkat kevaliditasan outputnya. Standart parameter tersebut dapat dilihat dalam Tabel 3.2.

Tabel 3.2 Parameter Pengujian CRC 32 bit

No.	Parameter	Nilai
1.	Width	32 bit
2.	Poly	04C11DB7
3.	Init	FFFFFFFF
4.	RefIn	FFFFFFFF
5.	RefOut	FFFFFFFF
6.	XorOut	FFFFFFFF
4.	Checksum	CBF43926

3.3.2 Pengujian perfomansi IC

Pengujian perfomansi ini terkait dengan kecepatan eksekusi IC, frekuensi, dan troughput yang dihasilkan. Nilai tersebut dapat diperoleh dengan melakukan pengujian *timing constraint. Tools* yang digunakan untuk melakukan pengujian *timing constraint* adalah *Timing Analyzer* yang juga merupakan salah satu fasilitas di *Xilinx iSE WebPACK 7.1i. Timing Analysis* yang dilakukan meliputi :

1. *Maksimum Clock Frequencies*
2. *Slack Time*
3. *Critical Path*
4. *Delay*

3.4 Penutup

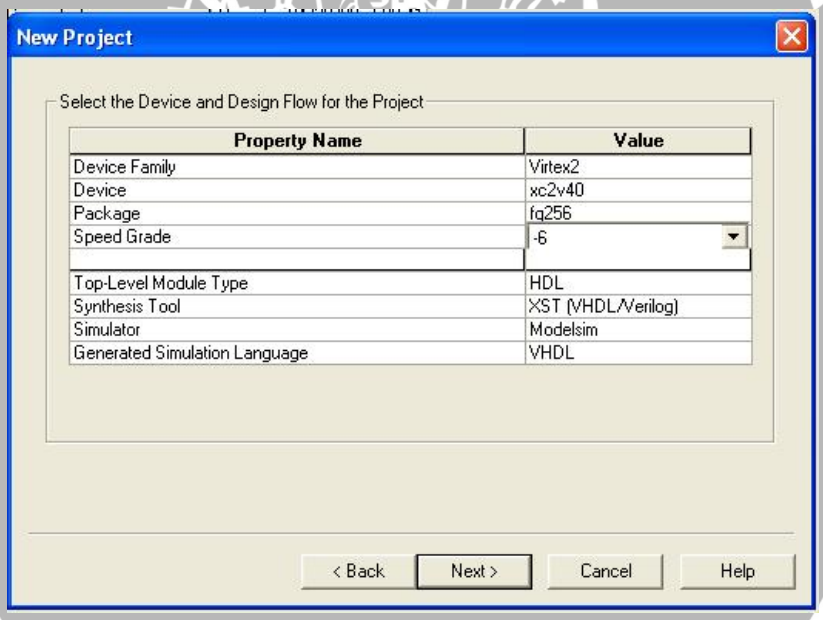
BAB ini berisi kesimpulan dan saran. Tahap pertama, pengambilan kesimpulan berdasarkan hasil perancangan dan pengujian. Kita bandingkan kesesuainnya antara data yang kita peroleh dan data menurut teori. Tahap selanjutnya adalah penulisan saran yang bertujuan untuk memperbaiki kesalahan dan menyempurnakan penelitian ini.

BAB IV
PERANCANGAN DAN IMPLEMENTASI

Pada BAB ini akan dibahas mengenai perancangan dan implementasi dari *Parallel CRC 32 bit* menggunakan bahasa VHDL. *Parallel CRC 32 bit* ini dapat digunakan untuk aplikasi *error detection* pada Ethernet.

4.1 Tahap Spesifikasi

Dalam perancangan dan implementasi *Parallel CRC 32 bit* ini digunakan bahasa pemrograman VHDL. *Logic design tool* yang digunakan adalah *Xilinx ISE WebPACK 7.1i*. Untuk pembuatan *Parallel CRC 32 bit* ini digunakan *FPGA Xilinx family Virtex2, device XC2V40* dengan *package* FG256 *speed grade -6. Project properties* dari perancangan ini dapat dilihat dalam Gambar 4.1.

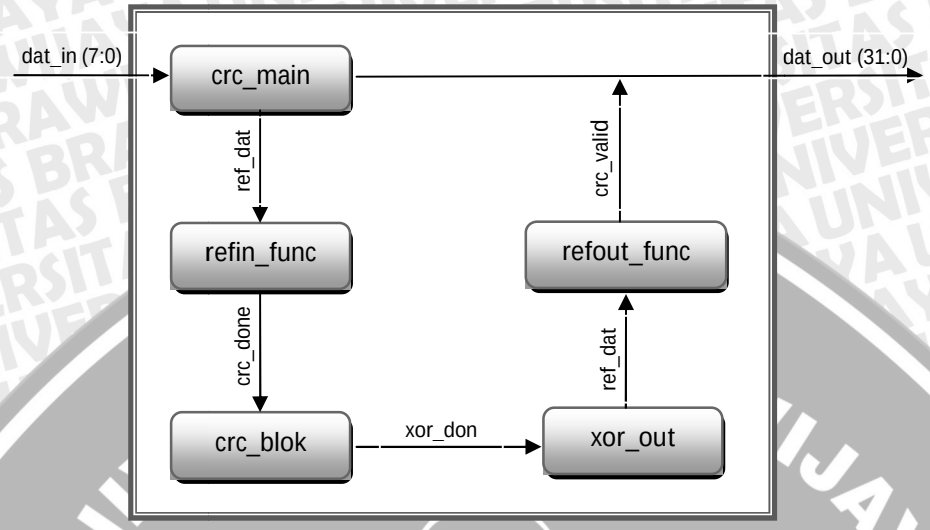


Gambar 4.1 *Project properties Parallel CRC 32 bit*

4.2 Perancangan *Parallel CRC 32 bit*

Perancangan *Parallel CRC 32 bit* ini mengacu pada blok diagram yang ditunjukkan dalam Gambar 4.2. Berdasarkan blok diagram tersebut, perancangan *Parallel CRC 32 bit* ini terdiri dari 4 buah modul. Antara lain modul `crc_main`

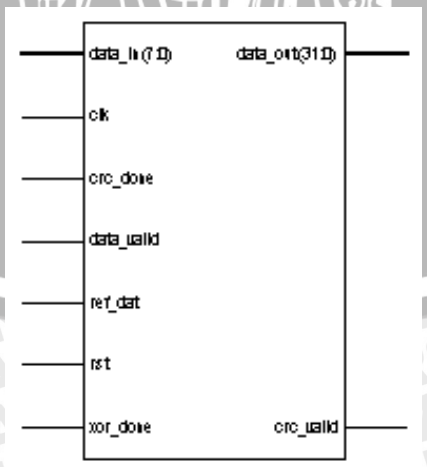
sebagai modul utama, modul `crc_blok`, modul `refin_func` dan modul `refout_func`.



Gambar 4.2 Blok Diagram *Parallel CRC 32 Bit*

4.2.1 Modul *crc_main*

Tahap pertama dalam perancangan *Parallel 32 bit* ini adalah mendesain modul `crc_main`. Modul `crc_main` ini merupakan modul utama yang mengatur seluruh kontrol logika untuk proses penghitungan *checksum* pada *Parallel CRC 32 bit*. Desain modul `crc_main` ini terdiri dari 47 pin utama dan 2 pin untuk catu daya, yaitu pin Vcc dan ground. Pin-pin utama yang digunakan dapat dilihat dalam Gambar 4.3 dan Tabel 4.1.



Gambar 4.3 Perancangan crc_main

Tabel 4.1 Konfigurasi pin-pin utama Modul `crc_main`

No.	Pin	I/O	Fungsi
1.	*clk*	input	Pin untuk mengatur clock
2.	*rst*	input	Pin untuk mengatur reset
3.	*data_valid*	input	Pin untuk mengenali data paralel
4.	*crc_done*	input	Pin untuk mengatifkan proses penghitungan checksum
5.	*ref_dat*	input	Pin untuk memilih data yang pertama kali masuk MSB atu LSB pada input dan output data
6.	*xor_done*	input	Pin untuk melakukan fungsi xorout sebelum data crc dikeluarkan
7.	*data_in[7]*	input	Pin untuk memasukkan data masukan bit ke-7
8.	*data_in[6]*	input	Pin untuk memasukkan data masukan bit ke-6
9.	*data_in[5]*	input	Pin untuk memasukkan data masukan bit ke-5
10.	*data_in[4]*	input	Pin untuk memasukkan data masukan bit ke-4
11.	*data_in[3]*	input	Pin untuk memasukkan data masukan bit ke-3
12.	*data_in[2]*	input	Pin untuk memasukkan data masukan bit ke-2
13.	*data_in[1]*	input	Pin untuk memasukkan data masukan bit ke-1
14.	*data_in[0]*	input	Pin untuk memasukkan data masukan bit ke-0
15.	*data_out[31]*	output	Pin untuk mengeluarkan output bit ke-31
16.	*data_out[30]*	output	Pin untuk mengeluarkan output bit ke-30
17.	*data_out[29]*	output	Pin untuk mengeluarkan output bit ke-29
18.	*data_out[28]*	output	Pin untuk mengeluarkan output bit ke-28
19.	*data_out[27]*	output	Pin untuk mengeluarkan output bit ke-27
20.	*data_out[26]*	output	Pin untuk mengeluarkan output bit ke-26
21.	*data_out[25]*	output	Pin untuk mengeluarkan output bit ke-25
22.	*data_out[24]*	output	Pin untuk mengeluarkan output bit ke-24
23.	*data_out[23]*	output	Pin untuk mengeluarkan output bit ke-23
24.	*data_out[22]*	output	Pin untuk mengeluarkan output bit ke-22
25.	*data_out[21]*	output	Pin untuk mengeluarkan output bit ke-21
26.	*data_out[20]*	output	Pin untuk mengeluarkan output bit ke-20
27.	*data_out[19]*	output	Pin untuk mengeluarkan output bit ke-19
28.	*data_out[18]*	output	Pin untuk mengeluarkan output bit ke-18
29.	*data_out[17]*	output	Pin untuk mengeluarkan output bit ke-17
30.	*data_out[16]*	output	Pin untuk mengeluarkan output bit ke-16
31.	*data_out[15]*	output	Pin untuk mengeluarkan output bit ke-15
32.	*data_out[14]*	output	Pin untuk mengeluarkan output bit ke-14

33.	*data_out[13]*	output	Pin untuk mengeluarkan output bit ke-13
34.	*data_out[12]*	output	Pin untuk mengeluarkan output bit ke-12
35.	*data_out[11]*	output	Pin untuk mengeluarkan output bit ke-11
36.	*data_out[10]*	output	Pin untuk mengeluarkan output bit ke-10
37.	*data_out[9]*	output	Pin untuk mengeluarkan output bit ke-9
38.	*data_out[8]*	output	Pin untuk mengeluarkan output bit ke-8
39.	*data_out[7]*	output	Pin untuk mengeluarkan output bit ke-7
40.	*data_out[6]*	output	Pin untuk mengeluarkan output bit ke-6
41.	*data_out[5]*	output	Pin untuk mengeluarkan output bit ke-5
42.	*data_out[4]*	output	Pin untuk mengeluarkan output bit ke-4
43.	*data_out[3]*	output	Pin untuk mengeluarkan output bit ke-3
44.	*data_out[2]*	output	Pin untuk mengeluarkan output bit ke-2
45.	*data_out[1]*	output	Pin untuk mengeluarkan output bit ke-1
46.	*data_out[0]*	output	Pin untuk mengeluarkan output bit ke-0
47.	*crc_valid*	output	Pin untuk mengeluarkan data paralel

Sumber : Perancangan

Pada modul `crc_modul` ini menggunakan pendekatan gabungan antara *behavorial* dan *structural architecture.* Deskripsi *structural* untuk modul ini terdiri dari beberapa bagian, antara lain deklarasi sinyal dan deklarasi komponen, serta proses sequential `crc_code`. Pada proses sequential, `present_state` ditentukan oleh keadaan pin `rst` atau `clk`.

```
if rst = '0' then
      present_state <= idle;
elsif clk'event and (clk = '1') then
      present_state <= next_state;
end if;
```

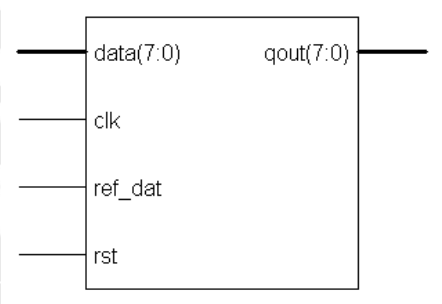
Proses `crc_code` bertanggung jawab untuk sinkronisasi kontrol logika proses penghitungan nilai *checksum.* Pin `crc_valid` aktif ketika pengkodean telah selesai dilakukan dan `data_out` telah siap untuk dikeluarkan. Untuk proses `crc_code` ini ada beberapa macam kondisi inisialisasi sinyal, yaitu `idle`, `assign` dan `done`. Pada kondisi `idle`, variabel data masukan akan direset pada logika 0. Untuk kondisi `assign` proses harus menunggu fungsi penghitungan

nilai *checksum* selesai terlebih dahulu dan untuk kondisi done, nilai crc_valid akan dikeluarkan.

```
crc_code : process (present_state, data_valid, crc_done, data_in)
   begin
      next_state <= present_state;
            din  <= data_in;
      crc_valid <= '1';

      case present_state is
            when idle =>
            din <= x"00";
                  if data_valid = '1' then
                        next_state <= assign;
                  end if;

            when assign =>
                  if crc_done = '1' then
                        next_state <= done;
                  end if;
            when done =>
            crc_valid <= '1';
                  if data_valid = '0' then
                        next_state <= idle;
                  end if;

            when others =>
                  next_state <= idle;

   end case;
   end process crc_code;
```

4.2.2 Modul refin_func

Modul ini berfungsi untuk proses *reflected* data masukan. Proses *reflected* berlangsung saat pin ref_dat pada modul crc_main berlogika '1'. Dengan melakukan *reflected* maka data yang pertama kali masuk adalah MSB-nya terlebih dahulu. Konfigurasi pin-pin utama modul refin_func dapat dilihat dalam Gambar 4.4 dan Tabel 4.2. antara lain terdiri dari 19 pin utama dan 2 pin catu daya yaitu pin Vcc dan ground.



Gambar 4.4 Perancangan modul refin_func

Tabel 4.2 Konfigurasi Pin Utama Modul `refin_func`

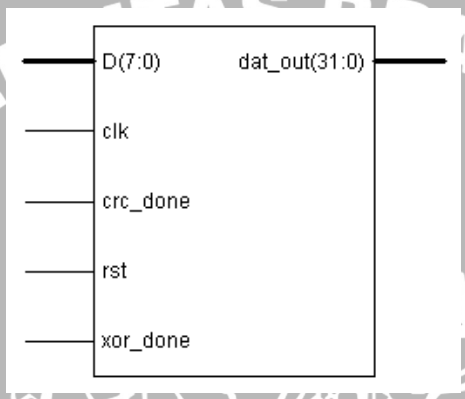
No.	Pin	I/O	Fungsi
1.	*clk*	input	Pin untuk mengatur clock
2.	*rst*	input	Pin untuk mengatur reset
3.	*init*	input	Pin untuk memulai proses inisialisasi
4.	*data[7]*	input	Pin untuk memasukkan data masukan bit ke-7
5.	*data[6]*	input	Pin untuk memasukkan data masukan bit ke-6
6.	*data[5]*	input	Pin untuk memasukkan data masukan bit ke-5
7.	*data[4]*	input	Pin untuk memasukkan data masukan bit ke-4
8.	*data[3]*	input	Pin untuk memasukkan data masukan bit ke-3
9.	*data[2]*	input	Pin untuk memasukkan data masukan bit ke-2
10.	*data[1]*	input	Pin untuk memasukkan data masukan bit ke-1
11.	*data[0]*	input	Pin untuk memasukkan data masukan bit ke-0
12.	*qout[7]*	output	Pin untuk mengeluarkan output bit ke-7
13.	*qout[6]*	output	Pin untuk mengeluarkan output bit ke-6
14.	*qout[5]*	output	Pin untuk mengeluarkan output bit ke-5
15.	*qout[4]*	output	Pin untuk mengeluarkan output bit ke-4
16.	*qout[3]*	output	Pin untuk mengeluarkan output bit ke-3
17.	*qout[2]*	output	Pin untuk mengeluarkan output bit ke-2
18.	*qout[1]*	output	Pin untuk mengeluarkan output bit ke-1
19.	*qout[0]*	output	Pin untuk mengeluarkan output bit ke-0

Sumber : Perancangan

Pemodelan yang digunakan pada modul ini adalah *behavorial architecture.* Proses berlangsung saat *rising edge* yaitu saat clock transisi dari ‘0’ ke ‘1’ dan saat nilai `rst` = ‘1’.

4.2.3 Modul *crc_blok*

Modul ini berfungsi untuk mengatur fungsi penghitungan nilai checksum dari data masukan pada *Parallel CRC 32 bit*. Dari modul ini 8 bit data masukan akan dihitung nilai *checksum*-nya sehingga menghasilkan keluaran sebesar 32 bit. Data masukan pada modul ini merupakan keluaran dari modul `refin_func`. Desain modul `crc_blok` ini terdiri dari 44 pin utama dan 2 pin untuk catu daya, yaitu pin Vcc dan ground. Pin-pin utama yang digunakan dapat dilihat dalam Gambar 4.5 dan Tabel 4.3.



Gambar 4.5 Perancangan modul crc_blok

Tabel 4.3 Konfigurasi Pin Utama Modul `crc_blok`

No.	Pin	I/O	Fungsi
1.	*clk*	input	Pin untuk mengatur clock
2.	*rst*	input	Pin untuk mengatur reset
3.	*crc_done*	input	Pin untuk mengatifkan proses penghitungan checksum
4.	*xor_done*	input	Pin untuk mengaktifkan fungsi modul xor_out
5.	*D[7]*	input	Pin untuk memasukkan data masukan bit ke-7
6.	*D[6]*	input	Pin untuk memasukkan data masukan bit ke-6
7.	*D[5]*	input	Pin untuk memasukkan data masukan bit ke-5
8.	*D[4]*	input	Pin untuk memasukkan data masukan bit ke-4
9.	*D[3]*	input	Pin untuk memasukkan data masukan bit ke-3
10.	*D[2]*	input	Pin untuk memasukkan data masukan bit ke-2
11.	*D[1]*	input	Pin untuk memasukkan data masukan bit ke-1
12.	*D[0]*	input	Pin untuk memasukkan data masukan bit ke-0
13.	*dat_out[31]*	output	Pin untuk mengeluarkan output bit ke-31
14.	*dat_out[30]*	output	Pin untuk mengeluarkan output bit ke-30

15.	*dat_out[29]*	output	Pin untuk mengeluarkan output bit ke-29
16.	*dat_out[28]*	output	Pin untuk mengeluarkan output bit ke-28
17.	*dat_out[27]*	output	Pin untuk mengeluarkan output bit ke-27
18.	*dat_out[26]*	output	Pin untuk mengeluarkan output bit ke-26
19.	*dat_out[25]*	output	Pin untuk mengeluarkan output bit ke-25
20.	*dat_out[24]*	output	Pin untuk mengeluarkan output bit ke-24
21.	*dat_out[23]*	output	Pin untuk mengeluarkan output bit ke-23
22.	*dat_out[22]*	output	Pin untuk mengeluarkan output bit ke-22
23.	*dat_out[21]*	output	Pin untuk mengeluarkan output bit ke-21
24.	*dat_out[20]*	output	Pin untuk mengeluarkan output bit ke-20
25.	*dat_out[19]*	output	Pin untuk mengeluarkan output bit ke-19
26.	*dat_out[18]*	output	Pin untuk mengeluarkan output bit ke-18
27.	*dat_out[17]*	output	Pin untuk mengeluarkan output bit ke-17
28.	*dat_out[16]*	output	Pin untuk mengeluarkan output bit ke-16
29.	*dat_out[15]*	output	Pin untuk mengeluarkan output bit ke-15
30.	*dat_out[14]*	output	Pin untuk mengeluarkan output bit ke-14
31.	*dat_out[13]*	output	Pin untuk mengeluarkan output bit ke-13
32.	*dat_out[12]*	output	Pin untuk mengeluarkan output bit ke-12
33.	*dat_out[11]*	output	Pin untuk mengeluarkan output bit ke-11
34.	*dat_out[10]*	output	Pin untuk mengeluarkan output bit ke-10
35.	*dat_out[9]*	output	Pin untuk mengeluarkan output bit ke-9
36.	*dat_out[8]*	output	Pin untuk mengeluarkan output bit ke-8
37.	*dat_out[7]*	output	Pin untuk mengeluarkan output bit ke-7
38.	*dat_out[6]*	output	Pin untuk mengeluarkan output bit ke-6
39.	*dat_out[5]*	output	Pin untuk mengeluarkan output bit ke-5
40.	*dat_out[4]*	output	Pin untuk mengeluarkan output bit ke-4
41.	*dat_out[3]*	output	Pin untuk mengeluarkan output bit ke-3
42.	*dat_out[2]*	output	Pin untuk mengeluarkan output bit ke-2
43.	*dat_out[1]*	output	Pin untuk mengeluarkan output bit ke-1
44.	*dat_out[0]*	output	Pin untuk mengeluarkan output bit ke-0

Sumber : Perancangan

Di dalam modul crc_blok ini terdapat modul `xor_out` . Dengan modul ini data hasil perhitungan crc akan di-xor dengan nilai FFFFFFFF sebelum dikeluarkan dan menjadi masukan modul `refout_func`. Fungsi ini akan aktif ketika pin `xor_done` pada modul `crc_main` berlogika ‘1’.

Pendekatan yang digunakan pada modul ini gabungan antara *behavorial* dan *structural*. *Structural architecture* pada modul ini terdiri pendeklarasian komponen dan pendeklarasian sinyal serta proses xor_func sedangkan *behavorial architecture* digunakan pada proses crc_logic. Proses xor_func bertanggung jawab melakukan proses *xorout* pada data. Sinyal yang dipakai pada proses ini antara lain, `xor_done, C`, dan `xor1`.

```
xor_func : process (xor_done, C, xor1)
begin
      if xor_done = '0' then
                  dat_out <= C;
      else
                  dat_out <= xor1;
      end if;
 end process;
```

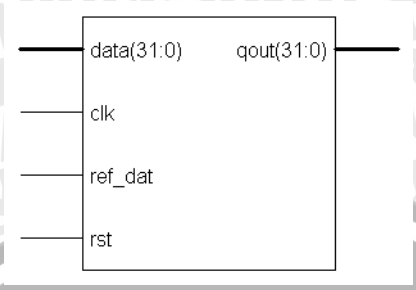
Sedang untuk proses crc_logic bertanggunjawab melakukan control logika pada proses perhitungan nilai CRC berdasarkan data masukan yang ada. Proses crc_logic ditentukan oleh pin `clk` dan `rst`. Saat pin `rst` berlogika '0' maka register akan diinisialisasi dengan nilai "FFFFFFFF". Proses perhitungan CRC akan dimulai saat pin `crc_done` berlogika '1'.

```
crc_logic : process (clk,rst)
begin
      if (rst = '0') then
            C <= x"ffffffff";
      elsif (rising_edge(clk)) then
            if (crc_done = '1') then
                  C <= NewCRC;
            end if;
      end if;
end process;
```

4.2.4 Modul `refout_func`

Modul ini berfungsi untuk proses *reflected* pada data hasil perhitungan CRC pada `crc_blok` sebelum dikeluarkan menjadi output saat pin `ref_dat` berlogika '1'. Seperti halnya pada modul `refin_func`, dengan modul ini maka bit-bit data keluaran akan dibalik. bit MSB menjadi bit LSB saat data akan dilekuarkan. Konfigurasi pin-pin utama modul `ini` dapat dilihat dalam Gambar

4.6 dan Tabel 4.4, antara lain terdiri dari pin utama dan 2 pin catu daya yaitu pin Vcc dan ground.



Gambar 4.6 Perancangan modul refout_func

Tabel 4.4 Konfigurasi Pin Utama Modul `refout_func`

No.	Pin	I/O	Fungsi
1.	*clk*	input	Pin untuk mengatur clock
2.	*rst*	input	Pin untuk mengatur reset
3.	*dat_in[31]*	input	Pin untuk memasukkan data masukan bit ke-31
4.	*dat_in[30]*	input	Pin untuk memasukkan data masukan bit ke-30
5.	*dat_in[29]*	input	Pin untuk memasukkan data masukan bit ke-29
6.	*dat_in[28]*	input	Pin untuk memasukkan data masukan bit ke-28
7.	*dat_in[27]*	input	Pin untuk memasukkan data masukan bit ke-27
8.	*dat_in[26]*	input	Pin untuk memasukkan data masukan bit ke-26
9.	*dat_in[25]*	input	Pin untuk memasukkan data masukan bit ke-25
10.	*dat_in[24]*	input	Pin untuk memasukkan data masukan bit ke-24
11.	*dat_in[23]*	input	Pin untuk memasukkan data masukan bit ke-23
12.	*dat_in[22]*	input	Pin untuk memasukkan data masukan bit ke-22
13.	*dat_in[21]*	input	Pin untuk memasukkan data masukan bit ke-21
14.	*dat_in[20]*	input	Pin untuk memasukkan data masukan bit ke-20
15.	*dat_in[19]*	input	Pin untuk memasukkan data masukan bit ke-19
16.	*dat_in[18]*	input	Pin untuk memasukkan data masukan bit ke-18
17.	*dat_in[17]*	input	Pin untuk memasukkan data masukan bit ke-17
18.	*dat_in[16]*	input	Pin untuk memasukkan data masukan bit ke-16
19.	*dat_in[15]*	input	Pin untuk memasukkan data masukan bit ke-15
20.	*dat_in[14]*	input	Pin untuk memasukkan data masukan bit ke-14
21.	*dat_in[13]*	input	Pin untuk memasukkan data masukan bit ke-13
22.	*dat_in[12]*	input	Pin untuk memasukkan data masukan bit ke-12
23.	*dat_in[11]*	input	Pin untuk memasukkan data masukan bit ke-11
24.	*dat_in[10]*	input	Pin untuk memasukkan data masukan bit ke-10
25.	*dat_in[9]*	input	Pin untuk memasukkan data masukan bit ke-9

26.	*dat_in[8]*	input	Pin untuk memasukkan data masukan bit ke-8
27.	*dat_in[7]*	input	Pin untuk memasukkan data masukan bit ke-7
28.	*dat_in[6]*	input	Pin untuk memasukkan data masukan bit ke-6
29.	*dat_in[5]*	input	Pin untuk memasukkan data masukan bit ke-5
30.	*dat_in[4]*	input	Pin untuk memasukkan data masukan bit ke-4
31.	*dat_in[3]*	input	Pin untuk memasukkan data masukan bit ke-3
32.	*dat_in[2]*	input	Pin untuk memasukkan data masukan bit ke-2
33.	*dat_in[1]*	input	Pin untuk memasukkan data masukan bit ke-1
34.	*dat_in[0]*	input	Pin untuk memasukkan data masukan bit ke-0
35.	*qout[31]*	output	Pin untuk mengeluarkan output bit ke-31
36.	*qout[30]*	output	Pin untuk mengeluarkan output bit ke-30
37.	*qout[29]*	output	Pin untuk mengeluarkan output bit ke-29
38.	*qout[28]*	output	Pin untuk mengeluarkan output bit ke-28
39.	*qout[27]*	output	Pin untuk mengeluarkan output bit ke-27
40.	*qout[26]*	output	Pin untuk mengeluarkan output bit ke-26
41.	*qout[25]*	output	Pin untuk mengeluarkan output bit ke-25
42.	*qout[24]*	output	Pin untuk mengeluarkan output bit ke-24
43.	*qout[23]*	output	Pin untuk mengeluarkan output bit ke-23
44.	*qout[22]*	output	Pin untuk mengeluarkan output bit ke-22
45.	*qout[21]*	output	Pin untuk mengeluarkan output bit ke-21
46.	*qout[20]*	output	Pin untuk mengeluarkan output bit ke-20
47.	*qout[19]*	output	Pin untuk mengeluarkan output bit ke-19
48.	*qout[18]*	output	Pin untuk mengeluarkan output bit ke-18
49.	*qout[17]*	output	Pin untuk mengeluarkan output bit ke-17
50.	*qout[16]*	output	Pin untuk mengeluarkan output bit ke-16
51.	*qout[15]*	output	Pin untuk mengeluarkan output bit ke-15
52.	*qout[14]*	output	Pin untuk mengeluarkan output bit ke-14
53.	*qout[13]*	output	Pin untuk mengeluarkan output bit ke-13
54.	*qout[12]*	output	Pin untuk mengeluarkan output bit ke-12
55.	*qout[11]*	output	Pin untuk mengeluarkan output bit ke-11
56.	*qout[10]*	output	Pin untuk mengeluarkan output bit ke-10
57.	*qout[9]*	output	Pin untuk mengeluarkan output bit ke-9
58.	*qout[8]*	output	Pin untuk mengeluarkan output bit ke-8
59.	*qout[7]*	output	Pin untuk mengeluarkan output bit ke-7
60.	*qout[6]*	output	Pin untuk mengeluarkan output bit ke-6
61.	*qout[5]*	output	Pin untuk mengeluarkan output bit ke-5
62.	*qout[4]*	output	Pin untuk mengeluarkan output bit ke-4

63.	*qout[3]*	output	Pin untuk mengeluarkan output bit ke-3
64.	*qout[2]*	output	Pin untuk mengeluarkan output bit ke-2
65.	*qout[1]*	output	Pin untuk mengeluarkan output bit ke-1
66.	*qout[0]*	output	Pin untuk mengeluarkan output bit ke-0

Sumber : Perancnagan

Pada modul `refout_func` ini menggunkan pemodelan *behavioral architecture* seperti yang digunakan pada modul refin_func. Proses juga berlangsung saat *rising edge* yaitu saat clolk transisi dari '0' ke '1' dan saat nilai `rst` = '1'.

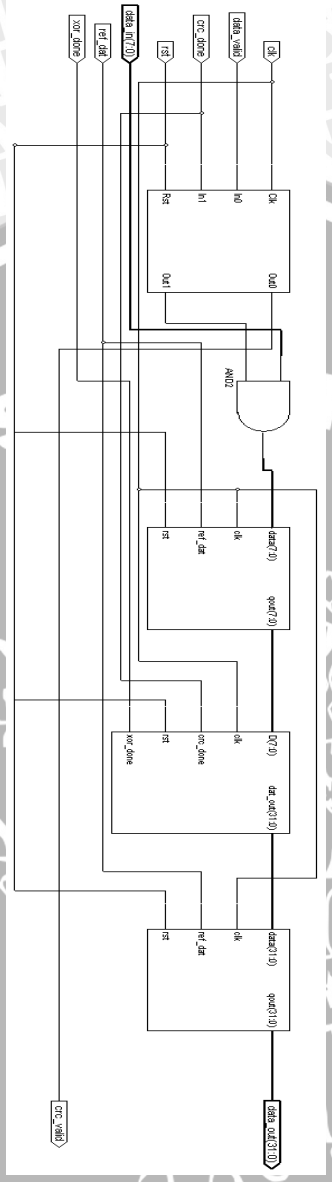
4.3 Tahap Sintesis

Pada proses ini *behavioral* desain rangkaian dikonversi menjadi gerbang-gerbang logika berdasarkan file text yang telah dideskripsikan tadi. Hasil dari proses ini berupa netlist yang sangat bergantung kepada vendor dan spesifikasi device-family, sehingga penggunaaan library harus sesuai dengan vendor

Proses sintesis pada *Xilinx ISE WebPACK 7.1i* disediakan pada menu *Synthesis/Implementation* pada *Source Window* dan *Synthesis – XST* pada *Process Window*. Langkah pertama adalah dengan melakukan proses *check syntax* untuk mengecek apakah ada kesalahan penulisan *syntax* VHDL-nya atau tidak sebelum disintesis. Apabila pengecekan ini berhasil dan tidak ada kesalahan sintaks, maka akan muncul *consule tab* yang menyatakan bahwa tidak ada penulisan sintaks dan akan muncul tanda centang (v) pada icon *check syntax*. Tetapi apabila terdapat kesalahan apabila ada kesalahan akan diperlihatkan kesalahanya seperti apa dan pada baris ke berapa. Serta tanda yang muncul pada icon *chechk syntax* berupa tanda silang (x).

Langkah berikutnya, file .vhd yang telah dicek sintaksnya siap untuk disintesis menggunakan fasilitas *Synthesis –XST*. Pada tahap ini akan dihasilkan *synthesis report* yang meliputi hasil Simulasi dan Analisis HDL, *timing report* serta gerbang-gerbang logika yang digunakan. Hasil *synthesis report* secara lengkap dapat dilihat dalam Lampiran 2. Pada tahap sintesis ini juga dapat diketahui hasil *RTL* (*Register Transfer Logic*) *Schematic Diagram* yang memperlihatkan gambaran hirarki dari perancangan dan komponen yang

digunakan. *Schematic Diagram* dari *parallel CRC 32 bit* dapat dilihat dalam Gambar 4.7.



Gambar 4.7 RTL *Schematic Diagram Parallel CRC 32 Bi*

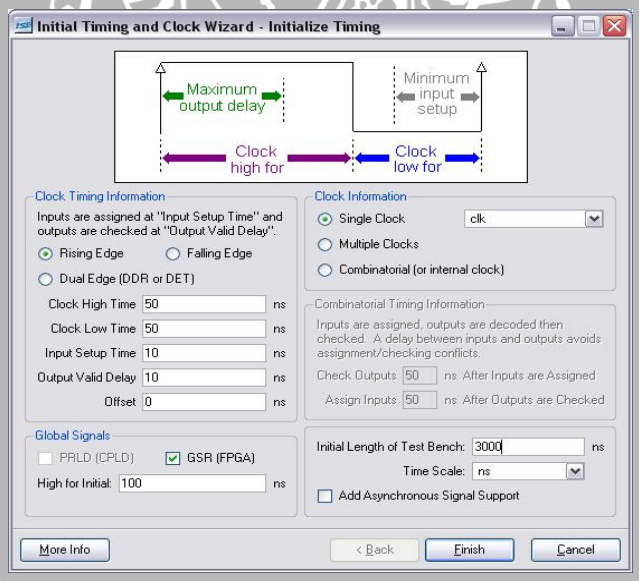
4.4 Tahap Simulasi

Untuk mensimulasikan desain rangkain *programmable logic* diperlukan sebuah simulator. Simulator merupakan sebuah software yang mengkonfirmasi fungsi dan timing sebuah rangkain sehingga kita bisa melakukan pengecekan terhadap kombinasi hasil dari desain rancangan digital.

Pada tahap simulai ini digunakan *ModelSim* sebagai software simulatornya. Pengecekan dilakukan dengan menggunakan *file Test Bench WaveForm. Test bench* adalah *netlist* HDL (*Hardware Description Language*) yang terdiri dari *test vector* untuk menjalankan simulasi. Dengan menggunakan *file Test Bench Wave Form* ini kita bisa mensimulasikan *structural* dan *behavioral* dari *Parallel CRC 32 bit* dengan memasukkan input-input sesuai dengan desain yang kita inginkan.

Langkah pertama yang harus dilakukan adalah melakuakan *initial timing set up* yang digunakan untuk simulasi. *Initial timing set up* yang diperlukan meliputi *periode*, input *set-up time* dan *valid-output* serta panjang *test bench* yang diinginkan. *Initial timing set up* untuk peranacangan CRC ini lebih lengkapnya dapat dilihat dalam Gambar.

Setelah kita memasukkan input sesuai dengan desain, dari simulasi tersebut kita dapat melihat apakah output yang dihasilkan sesuai dengan yang kita inginkan. Hasil dari simulasi pengimplementasian *parallel 32 bit* dapat dilihat dalam Gambar 4.8.

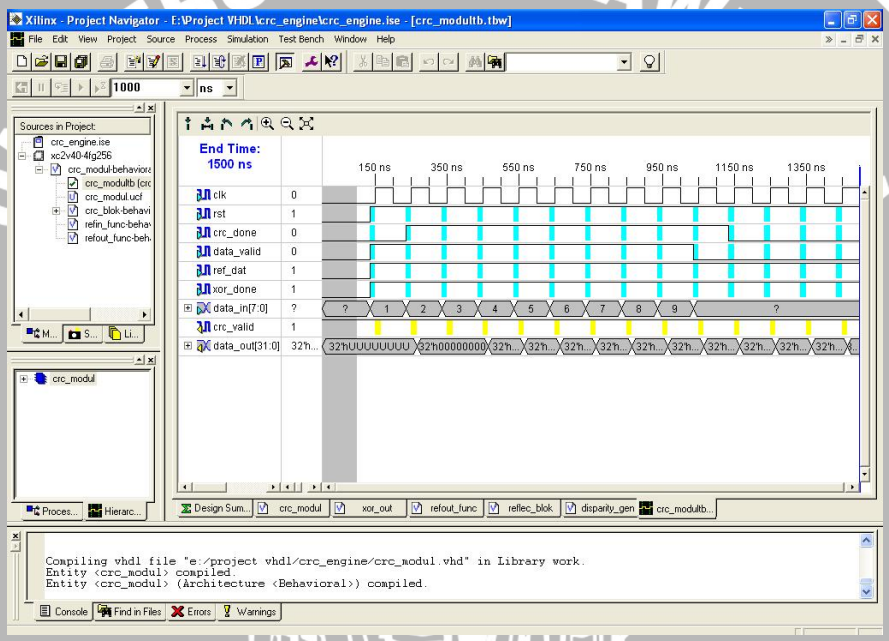


Gambar 4.8. *Initial timing set up*

Pada simulasi ini, *Clock High Time* dan *Clock Low Time* diset 50 ns, sehingga setiap periode *clock* adalah 100 ns. C*lock* aktif pada saat *rising edge* dan pin *reset* berlogika 1. Proses inisialisai register ke nilai “FFFFFFFF” terjadi saat pin `rst` berlogika ‘0’. Setelah itu pin `crc_done` diberi logika ‘1’ untuk

melakukan proses perhitungan nilai crc dari data yang masuk. Pin data_valid menandai data masukan yang akan dihitung nilai crcnya, dengan aktif logika tinggi. Pin `xor_done` diberi logika '1' apabila akan melakukan fungsi `xor_out` pada *checksum* hasil perhitungan crc. Untuk pin `ref_dat` diberi logika '1' agar data masukan dan data keluaran final dicerminkan terlebih dahulu sebelum dihitung nilai crcnya atau sebelum data tersebut dikeluarkan. Sehingga bit MSB menjadi LSB, begitu juga sebaliknya.

Hasil data simulasi parallel CRC 32 bit ini dapat dilihat dalam Gambar 4.9 Untuk simulasi ini data masukannya yang dipakai adalah bilangan "123456789" ASCII.



Gambar 4.9 Hasil simulasi data "123456789" ASCII

4.5 Implementasi

Langkah awal yang harus dilakukan sebelum tahap implementasi ini adalah dengan melakukan pengesetan spesifikasi *constraint.* Spesifikasi *constraint* yang diperlukan meliputi spesifikasi *constraint* waktu dan *constraint* pin. *Timing constraint* digunakan sebagai pedoman *place and route* pada desain sehingga implementasi desain dapat dioptimalkan.

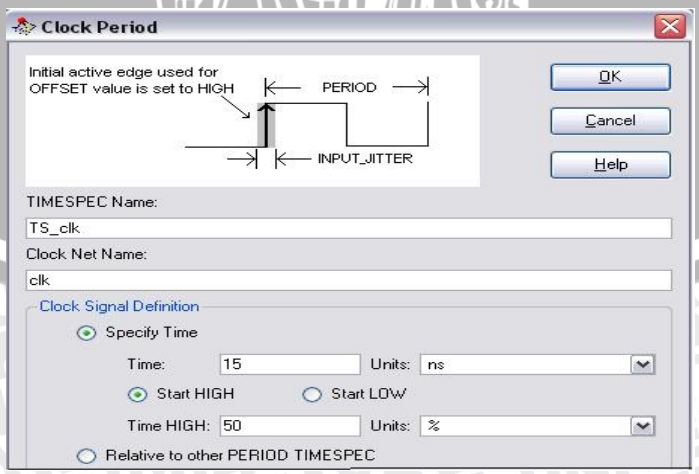
Clock period pada *Timing constraint* digunakan untuk spesifikasi frekuensi clock yang digunakan dalam pengoperasian FPGA. Sedang *Offset constraint* digunakan untuk spesifikasi data valid pada input FPGA maupun

output FPGA. Fasilitas yang digunakan untuk penentuan spesifikasi *constraint* ini adalah *Synthesis/Implementation* yang terdapat pada *Source Window* dan *User Constraint* pada *Process Window*.

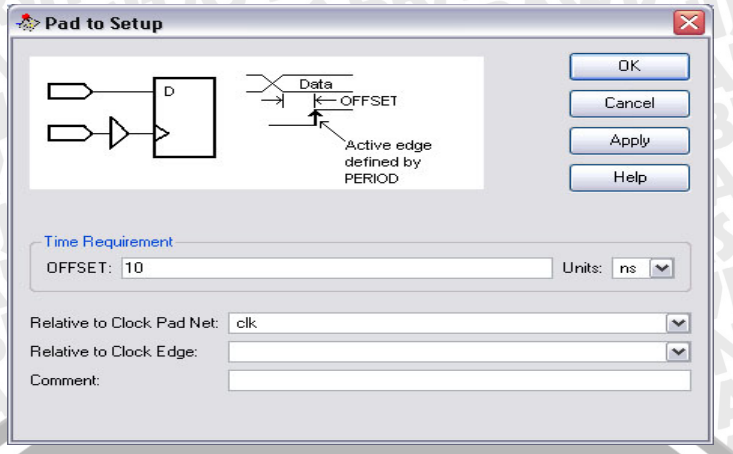
Proses implementasi dimulai dengan melakukan proses translation. Proses translation ini digunakan untuk mengimplementasikan *constraint* yang telah didefinisikan sebelumnya. Pada proses ini menghasilkan file .ngd dan .bld. Proses selanjutnya adalah proses *mapping*. Pada proses *mapping* hasil dari translation akan dipetakan pada device yang digunakan sehingga kita bisa mengetahui pemetaan dari gerbang logika yang digunakan.

Proses selanjutnya adalah proses *Place and Route* yang merupakan proses terakhir pada tahap implemntai ini. Pada tahap inilah desain akan diimplemntasikan ke dalam sebuah chip. *Place* adalah sebuah proses untuk memilih modul yang spesifik atau *logic stream* pada FPGA tempat gerbang logika akan ditempatkan. *Route* adalah proses *routing* fisik yang menghubungkan antar *logic stream*.

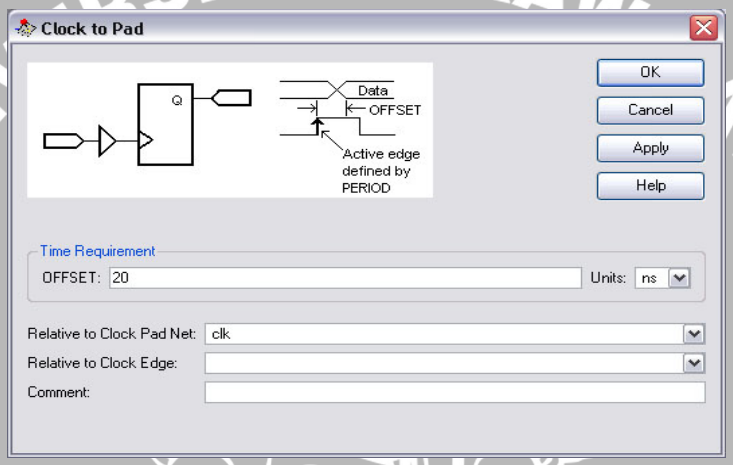
Pada perancangan ini kotak dialog *time* di-*set* pada 15 ns. Gambar 4.10. menunjukkan pengesetan *clock period*. Kotak dialog *Pad to Setup* menunjukkan *input offset constraint* dengan nilai *offset* 10 ns, seperti yang terlihat dalam Gambar 4.11. Kotak dialog *Clock to Pad* menunjukkan *output delay constraint* dengan nilai *offset* 20 ns, seperti ditunjukkan dalam Gambar 4.12. Pengesetan seperti ini mengacu pada hasil *synthesis report*.



Gambar 4.10. *Clock Period Set up* dari *Parallel CRC 32 Bit*

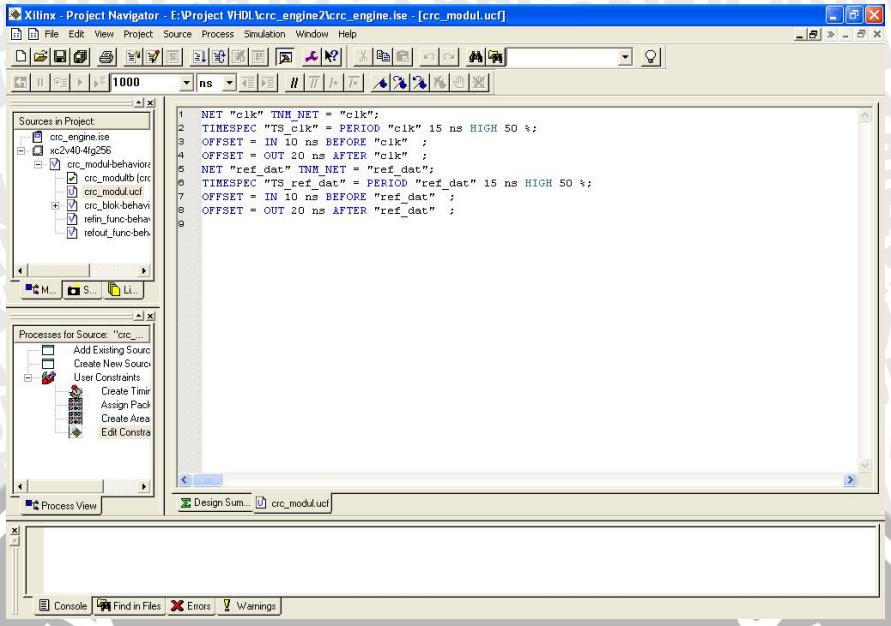


Gambar 4.11. *Pad to Set up* dari *Parallel CRC 32 Bit*



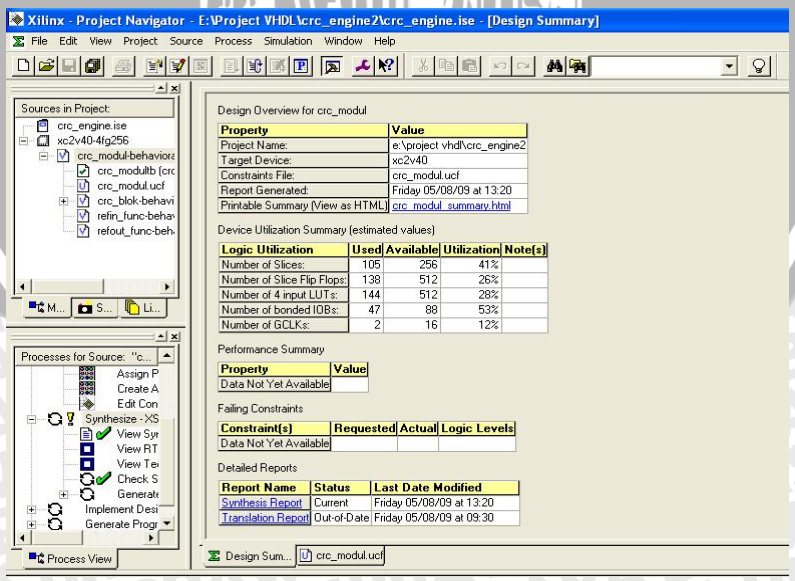
Gambar 4.12 *Clock to Pad Set up* dari *Parallel CRC 32 Bit*

Jika semua *time constraint* sukses diimplementasikan pada FPGA, maka akan muncul pesan tidak ada kesalahan dan laporan hasil *constraint* tersebut dapat dilihat dalam Gambar 4.13.



Gambar 4.13 Hasil *Timing Constrain*

Dari tahapan-tahapan yang sudah dilaksanakan di atas akan dihasilkan *report* yang menunjukkan bahwa *parallel CRC 32 bit* dalam implementasinya membutuhkan 105 *slices*, menggunakan 47 atau 53% IOB (*Input/Output Block*), menempati 138 atau 26% *slices flip-flops*, membutuhkan 144 LUT dan 2 *General Clock device* Xilinx Virtex2 XC2V40 *package* FG256 dengan frekuensi maksimum 128,535 MHz dan periode minimum 7,780 ns pada *speed grade* -6. *Design Sumary* dari peranncangan ini dapat dilihat dalam Gambar 4.14.

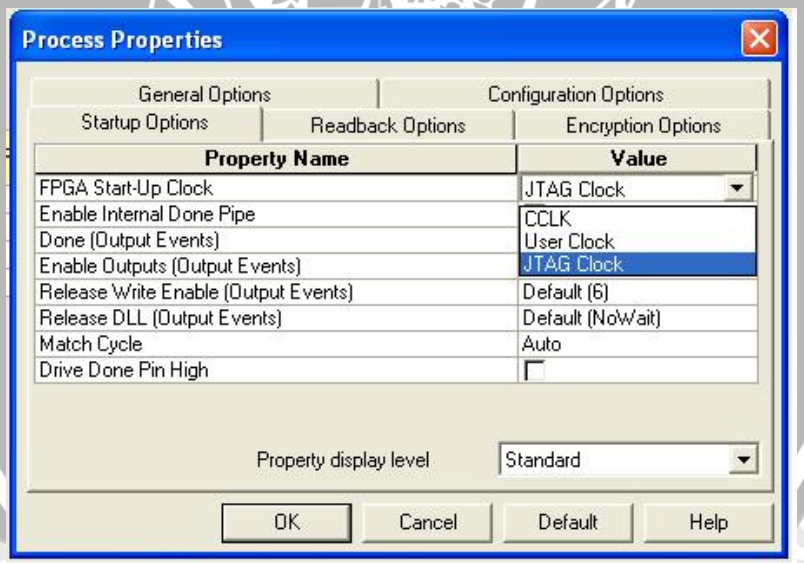


Gambar 4.14 *Design Summary Parallel CRC 32 bit*

4.6 Tahap Programming

Pada tahap programming ini hasil dari implemtasi akan di-*download* pada FPGA sesuai spesifikasi yang telah ditentukan sebelumnya. Proses programming dilakukan dengan memanfaatkan fasilitas *Synthesis/Implementation* pada *Source Window* dan *Generate Programming File* serta *Configure Device* (iMPACT) pada *Process Window*.

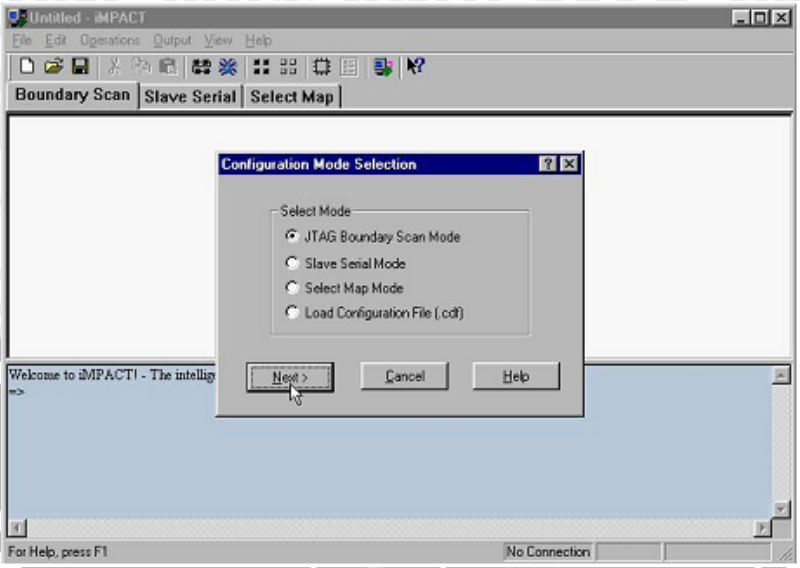
Langkah awal pada tahap programming ini adalah dengan membuat file *bitstream* dengan ekstensi *.bit dari rancnagan IC yang dibuat. Dengan cara melakukan *double-clik* pada *Generate Programming File* yang ada pada *process windows*. Proses ini akan menampilkan kotak dialog IMPACT. File ini yang nantinya akan di *upload* ke FPGA. Sebelumnya kita bisa mengatur *startup options* dengan cara meng-klik kanan pada *Generate Programming File*, pilih *Properties* kemudian *Startup Options*. Pilih CCLK apabila kita akan menggunakan *parallel port cable* atau pilih *JTAG Clock* apabila kita akan menggunakan *USB cable*. Dalam Gambar 4.15. diperlihatkan kotak dialog *Process Propeties* untuk pengaturan *Startup Options*.



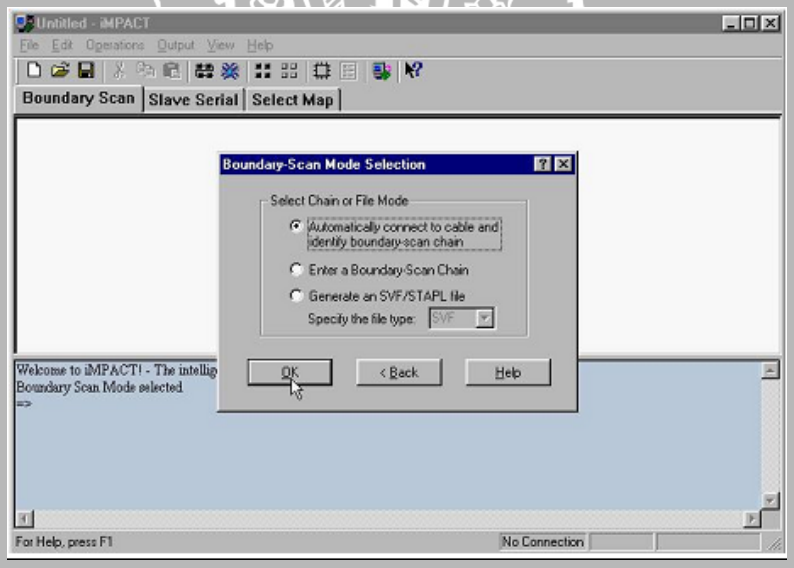
Gambar 4.15. kotak dialog *Process Propeties* untuk pengaturan *Startup Options*.

Pada kotak dialog iMPACT pastikan menu yang dipilih adalah *Configure devices using Boundary-Scan* (JTAG) kemudian klik *Next* dan pilih *Automatically*

connect to a cable and identify Bounday-Scan chain, seperti dalam Gambar 4.16 dan 4.17.

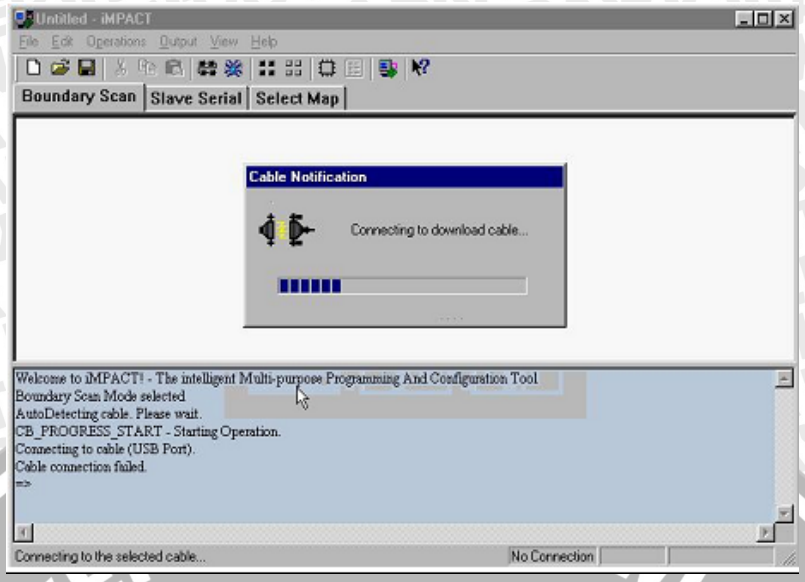


Gambar 4.16 Kotak dialog untuk *Configuration Mode Selections*



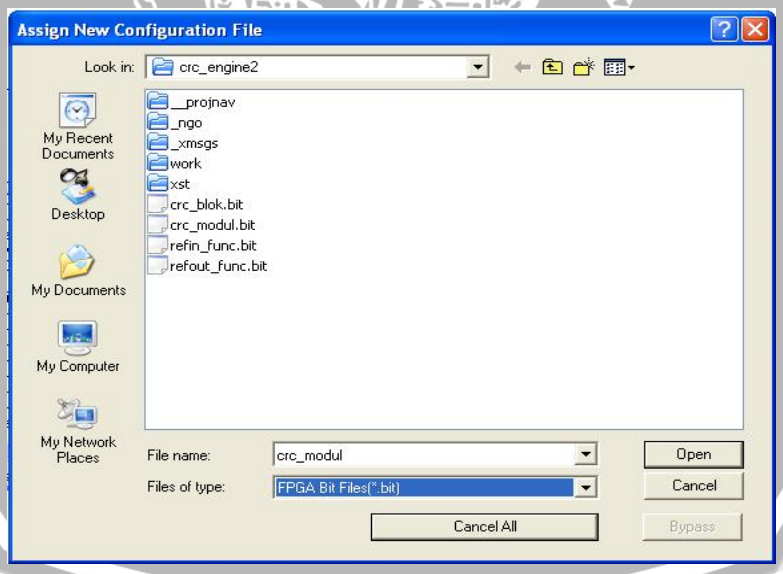
Gambar 4.17 Kotak dialog untuk *Boundary Scan Mode Selections*

Setelah kita memilih *Automatically connect to a cable and identify Bounday-Scan chain*, kita klik *OK* maka iMPACT akan mendeteksi FPGA. Seperti yang diperlihatkan dalam Gambar 4.18.



Gambar 4.18 Kotak dialog untuk *Cable Notification*

Berikutnya akan muncul kotak dialog *Assign New Configuration File*, dan bila berhasil maka pesan akan muncul dan siap dijalankan pada FPGA. Kotak dialog *Assign New Configuration File* dapat dilihat dalam Gambar 4.19.



Gambar 4.19 Kotak dialog untuk *Assign New Configuration File*

BAB V

PENGUJIAN

Setelah tahap implementasi, untuk mengetahui perfomansi dan validitas dari *Parallel CRC 32 bit* dilakukan tahap pengujian. Proses pengujian yang dilakukan terdiri dari dua macam, pengujian performansi IC dan validitas output dari *Parallel CRC 32 bit*. Pengujian validitas IC dilakukan dengan melakukan *test vector* pada masing-masing modul baik pada modul utama, yaitu `crc_main` maupun modul lainnya antara lain modul `crc_blok`, `refin_func`, dan `refout_func` .

Sedangkan untuk pengujian perfomansi terutama untuk melihat *troughput* atau kecepatan dari IC dan *timing analysis*. Dengan melihat kesesuaiaannya dengan standart untuk Gigabit Ethernet.

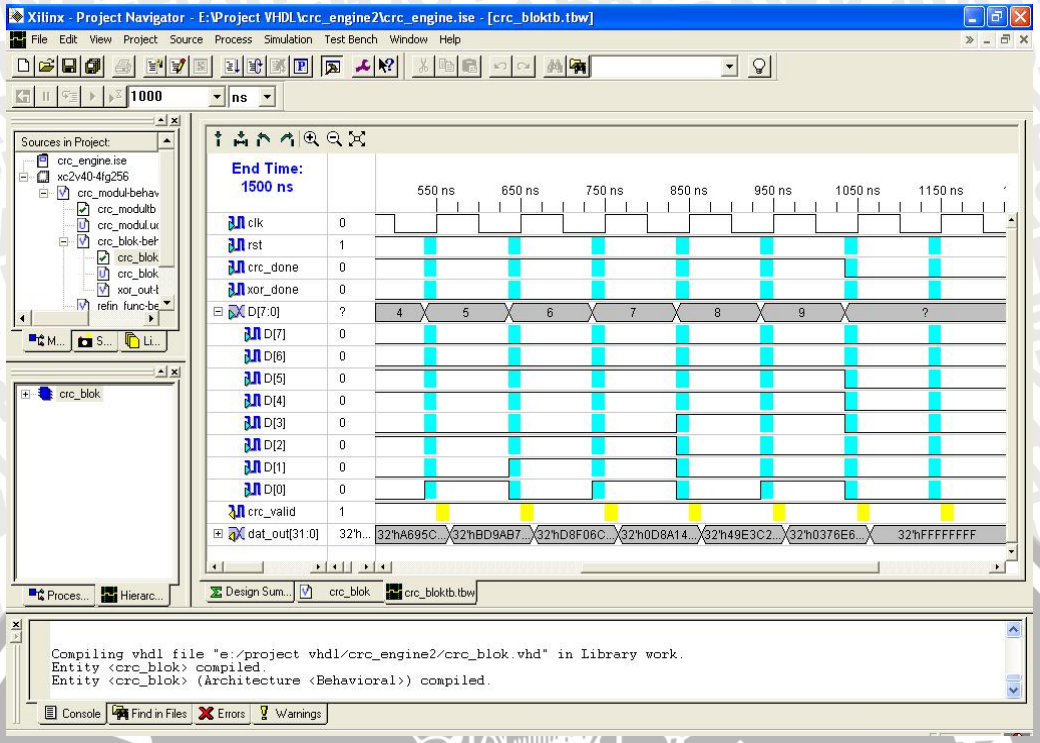
5.1 Pengujian Validitas *Checksum*

5.1.1 Modul `crc_blok`

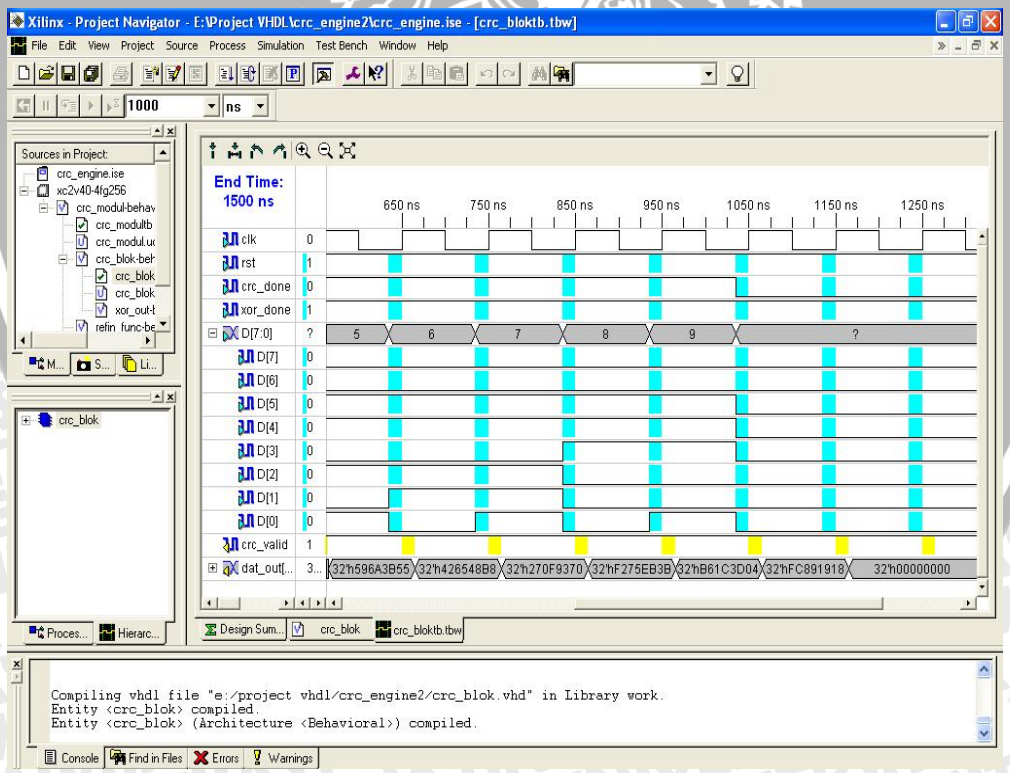
5.1.1.1 Pengujian Modul `crc_blok`

Pengujian modul ini melakukan simulasi menggunakan file *test bench* yang merupakan salah satu fasilitas yang ada pada simulator, yaitu *ModelSim*. Pengujian dilakukan dengan melakukan *test vector*. Data yang digunakan untuk pengujian seperti yang digunakan sesuai dengan standart data yang digunakan untuk melakukan pengujian pada CRC yaitu bilangan “123456789” ASCII . Data ini juga yang digunakan pada proses simulasi bab sebelumnya.

Pengujian untuk dua data masukan tersebut dilakukan dengan dua kondisi yaitu saat pin `xor_done` berlogika ‘0’ dan ‘1’ .Gambar hasil simulasi dapat dilihat dalam Gambar 5.1 dan 5.2. Dalam kedua gambar hasil simulasi ini, data keluaran yang ditampilkankan belum merupakan nilai checksum CRC dari data masukan.



Gambar 5.1 Hasil simulasi modul crc_blok saat xor_done =’0’



Gambar 5.2 Hasil simulasi modul crc_blok saat xor_done =’1’

Gambar 5.1 merupakan hasil pengujian untuk *sample* data masukan "123456789" ASCII saat pin `xor_done` berlogika '0' sedangkan Gambar 5.2. merupakan hasil pengujian untuk data masukan yang sama tetapi saat pin `xor_done` berlogika '1'. Dari Gambar 5.1 diketahui data keluaran yang ditampilkan `dat_out` (31 :0) adalah "0376E6E7" H. Sedang dari Gambar 5.2 diketahui data keluaran yang ditampilkan `dat_out` (31 :0) adalah "FC891918" H.

5.1.1.2 Analisa Hasil Pengujian Modul `crc_blok`

Untuk pengujian modul `crc_blok`, pin `D (7:0)` yang digunakan untuk input data masukan yang akan dihitung nilai CRCnya. Sedangkan outputnya akan dikeluarkan melalui pin `dat_out` (31:0). Seperti yang sudah disebutkan di atas input data yang dimasukkan adalah bilangan "123456789" ASCII.

Pin `rst` diberi logika tinggi, begitu pula untuk pin `crc_done` diberi logika tinggi apabila kita akan melakukan perhitungan nilai CRC pada data masukan. Sedang pin `xor_done` diberi logika tinggi apabila kita ingin melakukan fungsi *xorout* terhadap data hasil perhitungan CRC data masukan sebelum dikeluarkan pada pin `dat_out`. Tetapi apabila kita tidak ingin malakukan fungsi *xorout* maka pin `xor_done` diberi logika rendah. Penentuan perlu tidaknya melakukan fungsi *xorout* disesuaikan dengan standart parameter dari CRC 32 bit yang sudah ada. Pada pengujian ini pin xor_done diuji dengan dua kondisi tersebut, yaitu diberi logika tinggi dan rendah. Hal ini dilakukan selain untuk mengetahui data keluaran untuk dua kondisi tersebut..

Untuk mengetahui validitas keluaran modul ini, maka dilakukan perbandingan antara data keluaran berdasarkan teori dan simulasi untuk data masukan "123456789" ASCII. Data keluaran berdasarkan simulasi mengacu pada hasil pengujian dalam Gambar 5.1 dan 5.2, sedangkan data keluaran berdasarkan teori mengacu pada standart CRC dalam Tabel 2.1. Tabel perbandingan data keluaran berdasarkan teori dan simulasi dapat dilihat dalam Tabel 5.1. Berdasarkan tabel tersebut terlihat bahwa data keluaran antara berdasarkan teori dan simulasi adalah sama. Sehingga dapat disimpulkan bahwa fungsi pengkodeaan CRC untuk modul ini telah sesuai dengan standart yang ada.

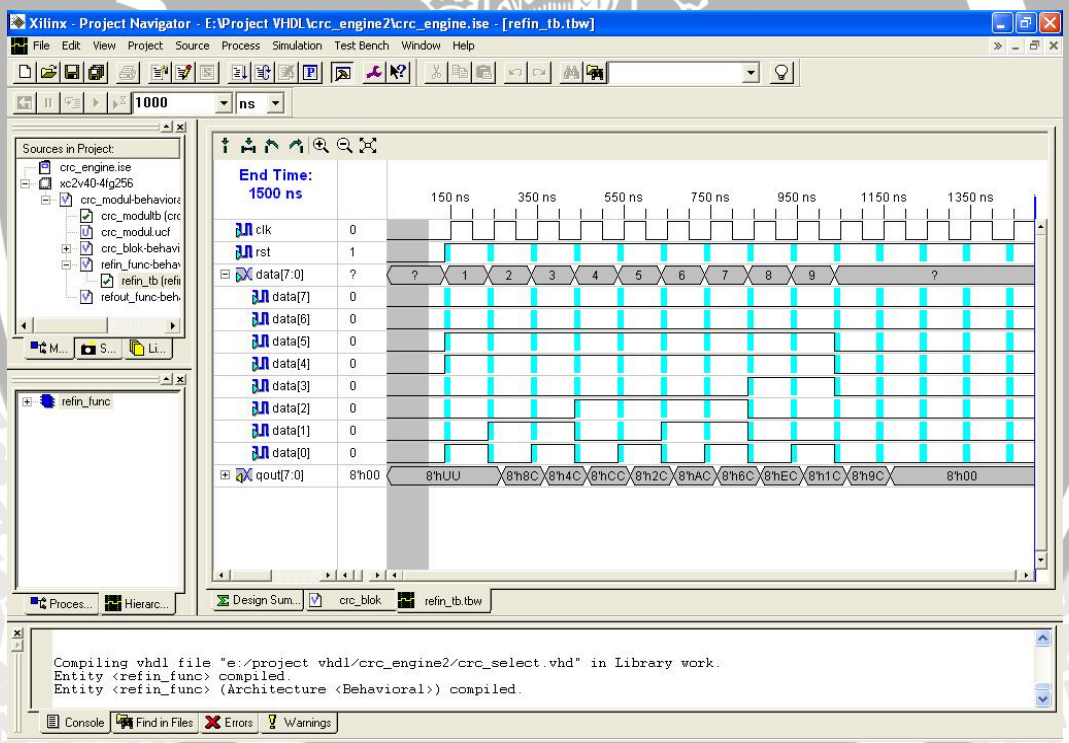
Tabel 5.1 Perbandingan Data Keluaran Untuk Modul `crc_blok`

Masukan	XorOut	Keluaran	
		Teori	Simulasi
“123456789” ASCII	False	“0376E6E7” H	“0376E6E7” H
	True	“FC891918” H	“FC891918” H

5.1.2 Modul `refin_func`

5.1.2.1 Pengujian Modul `refin_func`

Seperti pada pengujian sebelumnya, pengujian untuk modul ini dilakukan dengan simulasi menggunakan program *test bench* yaitu dengan menggunakan *test vector* data masukan. *Sample* data masukan yang digunakan antara lain adalah bilangan “123456789” ASCII. Hasil pengujian dapat dilihat dalam Gambar 5.3.



Gambar 5.3 Hasil simulasi modul refin_func

Dalam Gambar 5.3 data keluaran adalah yang ditampilkan oleh pin `qout` (7:0). Untuk bilangan ‘1’ ASCII keluarannya “8C” H, bilangan ‘2’ ASCII keluarannya “4C” H, bilangan ‘3’ ASCII keluarannya “CC” H, bilangan ‘4’ ASCII keluarannya “2C” H, bilangan ‘5’ ASCII keluarannya “AC” H, bilangan

‘6’ ASCII keluarannya “6C” H, bilangan ‘7’ ASCII keluarannya “EC” H, bilangan ‘8’ ASCII keluarannya “1C” H, dan untuk bilangan ‘9’ ASCII keluarannya “9C” H.

5.1.2.2 Analisa Hasil Pengujian Modul `refin_func`

Untuk pengujian modul `refin_func` ini, input data dimasukan melalui pin `data (7:0)`. Sedang data keluaran ditampilkan oleh pin qout (7:0). Seperti pengujian modul sebelumnya untuk data masukannya digunakan data yang sama, yaitu bilangan “123456789” ASCII dan. Dari hasil pengujian terlihat dalam Gambar 5.3 bahwa bit MSB masukan berubah menjadi LSB, begitu juga sebaliknya. Sehingga dapat disimpulkan bahwa modul ini telah berfungsi dengan baik.

Fungsi *reflected* terhadap data masukan pada modul ini aktif apabila pin rst diberi logika tinggi. Dalam sekali periode clock dapat dilakukan fungsi *reflected* untuk 8 bit data. Sehingga pada pengujian ini dibutuhkan 9 clock untuk 9 byte data masukan. Penentuan perlu tidaknya menggunakan fungsi *reflected* ini juga disesuaikan dengan standart parameter dari CRC 32 bit yang ada.

Untuk mengetahui validitas keluaran, maka dilakukan untuk modul ini juga dilakukan perbandingan antara data keluaran berdasarkan perhitungan dan simulasi. Data keluaran berdasarkan simulasi mengacu pada gambar hasil simulasi yaitu Gambar 5.3. Tabel perbandingan data keluaran berdasarkan perhitungan dan simulasi untuk dapat dilihat dalam Tabel 5.2. Berdasarkan tabel tersebut terlihat bahwa data keluaran antara berdasarkan teori dan simulasi adalah sama. Sehingga dapat disimpulkan bahwa fungsi *reflected* untuk modul ini telah berfungsi dengan baik.

Tabel 5.2 Perbandingan Data Keluaran Untuk Modul `refin_func`

No.	Masukan		Keluaran (Perhitungan)		Keluaran (Simulasi)	
	ASCII/Hexsa	Biner	Biner	Hexsa	Hexsa	Biner
1.	1 ASCII /31	00110001	10001100	8C	8C	10001100
2.	2 ASCII / 32	00110010	01001100	4C	4C	01001100
3.	3 ASCII /33	00110011	11001100	CC	CC	11001100
4.	4 ASCII / 34	00110100	00101100	2C	2C	00101100
5.	5 ASCII /35	00110101	10101100	AC	AC	10101100

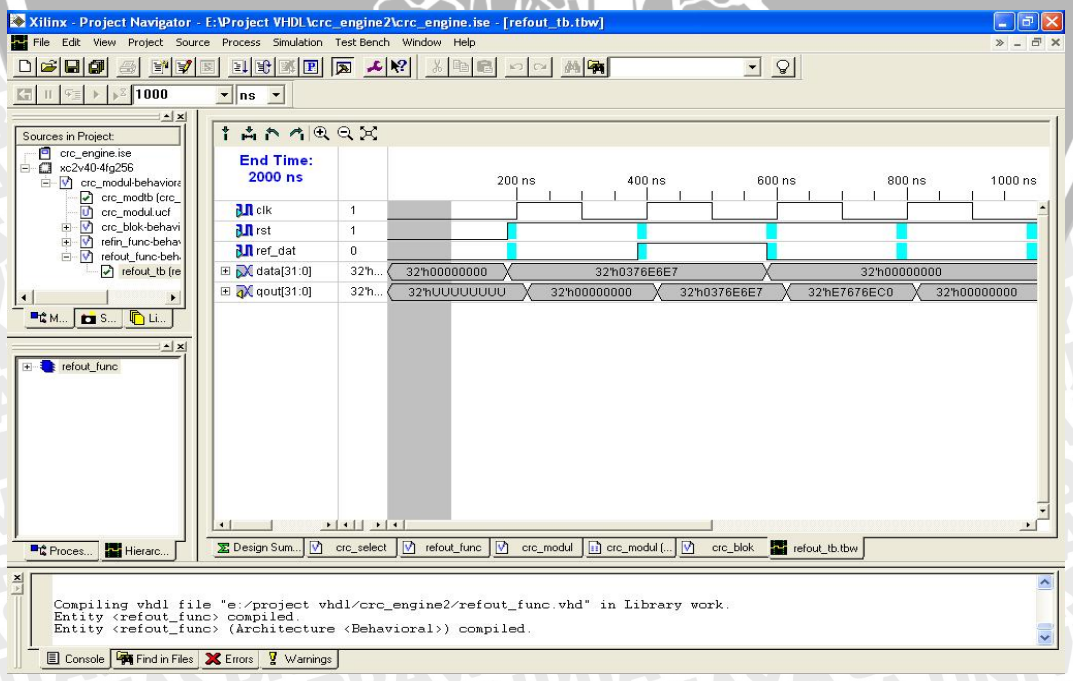
6.	6 ASCII / 36	00110110	01101100	6C	6C	01101100
7.	7 ASCII / 37	00110111	11101100	EC	EC	11101100
8.	8 ASCII / 38	00111000	00011100	1C	1C	00011100
9.	9 ASCII / 39	00111001	10011100	9C	9C	10011100

5.1.3 Modul `refout_func`

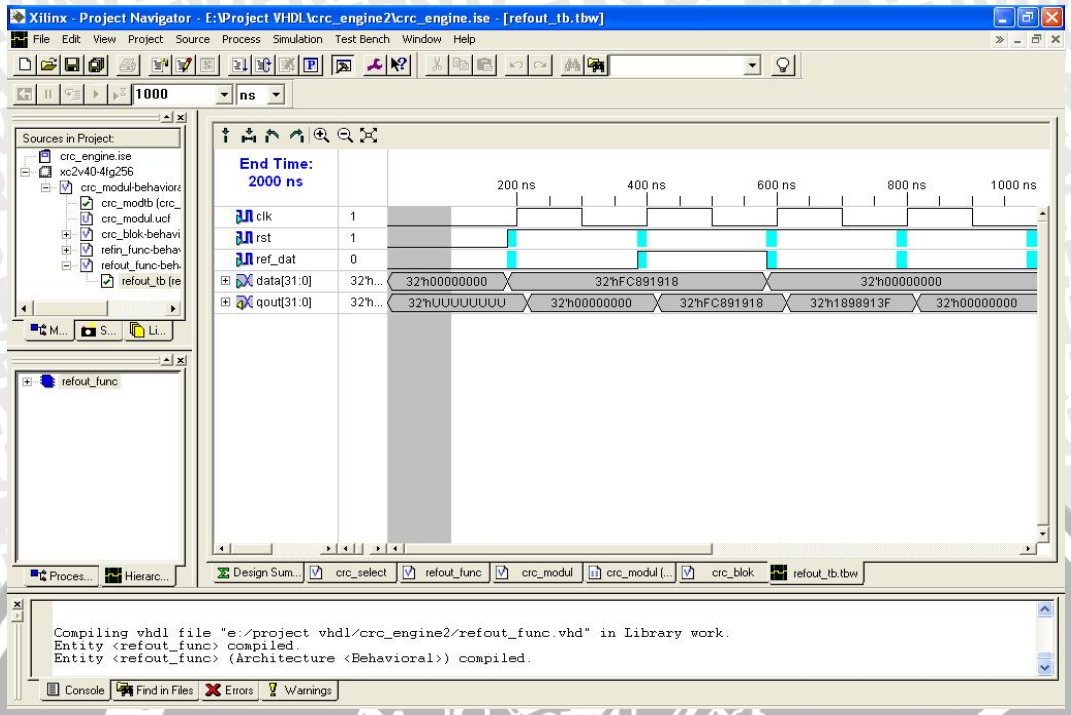
5.1.3.1 Pengujian Modul `refout_func`

Pada pengujian modul ini juga dengan simulasi menggunakan *test bench*, yaitu dengan melakukan *test vector*. Berbeda dengan pengujian-pengujian modul sebelumnya, pengujian pada modul ini menggunakan data masukan output dari modul `crc_blok`. Hal ini dilakukan karena data masukan dari modul refout_func ini berasal dari modul `crc_blok` yang merupakan nilai crc dari data masukan sebelum data tersebut dikeluarkan oleh pin dat_out.

Dengan melakukan pengujian pada modul ini kita bisa mengetahui apakah modul ini bisa melakukan fungs *reflected* atau tidak pada nilai *checksum* hasil perhitungan CRC. Hasil pengujian modul `refout_func` ini dapat dilihat dalam Gambar 5.4. dan 5.5. Dalam Gambar 5.4 data keluaran yang ditampilkan oleh pin qout adalah “E7676ECO” H. Sedang dalam Gambar 5.5 diketahui nilai keluaran modul ini adalah “1898913F” H.



Gambar 5.4 Hasil simulasi modul `refout_func` untuk data “0376E6E7” H



Gambar 5.5 Hasil simulasi modul `refout_func` untuk data “FC891918” H

5.1.3.2 Analisa Hasil Pengujian Modul `refout_func`

Pada pengujian modul ini, pin `data` (31:0) digunakan untuk input data masukan. Sedang pin `qout` (31:0) untuk output data keluaran. Seperti yang sudah dijelaskan di atas, data yang dimasukkan pada pin `data` adalah output dari modul `crc_blok`. Pada pengujian ini hanya dimabil untuk output dari bilangan “123456789” ASCII saja baik saat kondisi pin `xor_done` berlogika ‘1’ maupun ‘0’.

Fungsi dari modul refout_func ini hampir sama dengan modul `refin_func`, yaitu untuk mengubah bit MSB data masukan menjadi bit LSB dan bit LSB menjadi bit MSB serta diikuti bit-bit selanjutnya. Perbedaanya terletak pada jumlah bit data masukan dan data keluaran. Seperti pada modul `refin_func`, penentuan perlu tidaknya menggunakan fungsi *reflected* ini juga disesuaikan dengan standart parameter dari CRC 32 bit yang ada.

Fungsi *reflected* pada modul ini juga akan aktif apabila pin `rst` diberi logika tinggi. Pada modul ini dalam sekali periode clock dapat dilakukan fungsi

reflected untuk 32 bit data. Sehingga pada pengujian ini dibutuhkan 1 clock baik untuk data masukan pertama maupun data masukan kedua.

Untuk mengetahui validitas keluaran modul ini, maka dilakukan perbandingan antara data keluaran berdasarkan perhitungan dan simulasi. Data keluaran hasil simulasi mengacu pada Gambar 5.4 dan 5.5. Tabel perbandingan data keluaran berdasarkan perhitungan dan simulasi untuk dapat dilihat dalam Tabel 5.3. Berdasarkan tabel tersebut terlihat bahwa data keluaran antara berdasarkan teori dan simulasi adalah sama. Sehingga dapat disimpulkan bahwa fungsi pengkodeaan CRC untuk modul ini telah sesuai dengan standart yang ada.

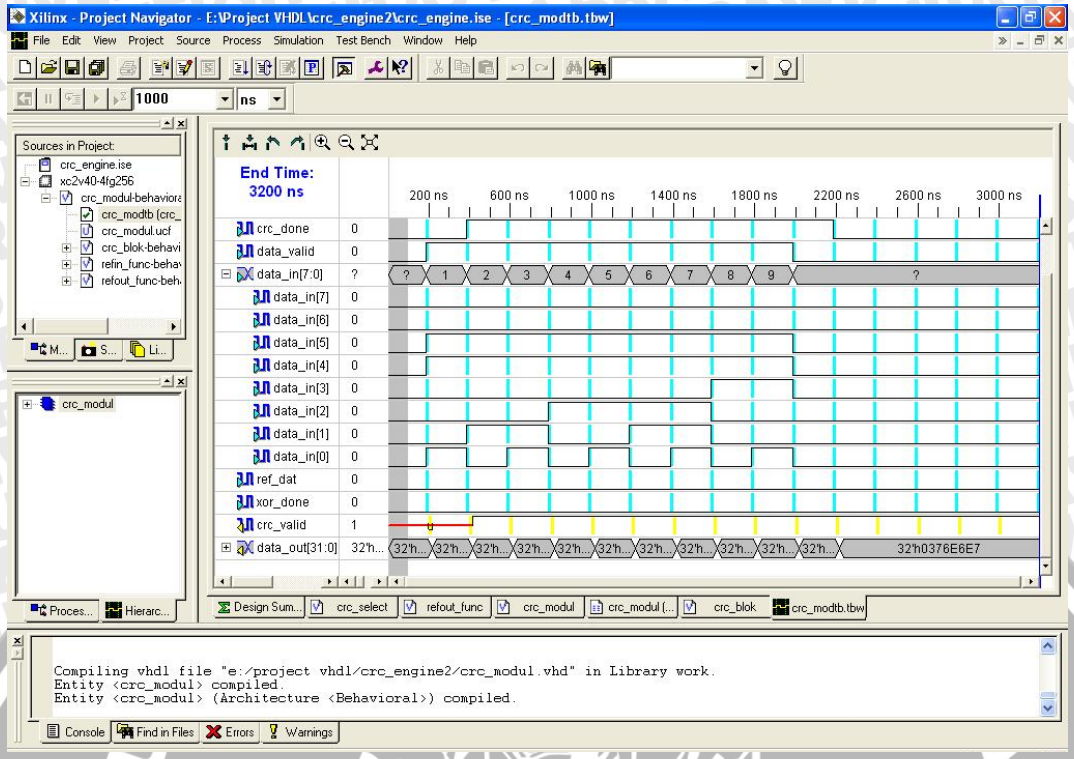
Tabel 5.3 Perbandingan Data Keluaran Untuk Modul `refout_func`

No.	Masukan		Keluaran (Perhitungan)		Keluaran (Simulasi)	
	Hexsa	Biner	Biner	Hexsa	Hexsa	Biner
1.	0376E6E7	0000 0011 0111 0110 1110 0110 1110 0111	1110 0111 0110 0111 0110 1110 1100 0000	E7676EC0	E7676EC0	1110 0111 0110 0111 0110 1110 1100 0000
2.	FC891918	1111 1100 1000 1001 0001 1001 0001 1000	0001 1000 1001 1000 1001 0001 0011 1111	1898913F	1898913F	0001 1000 1001 1000 1001 0001 0011 1111

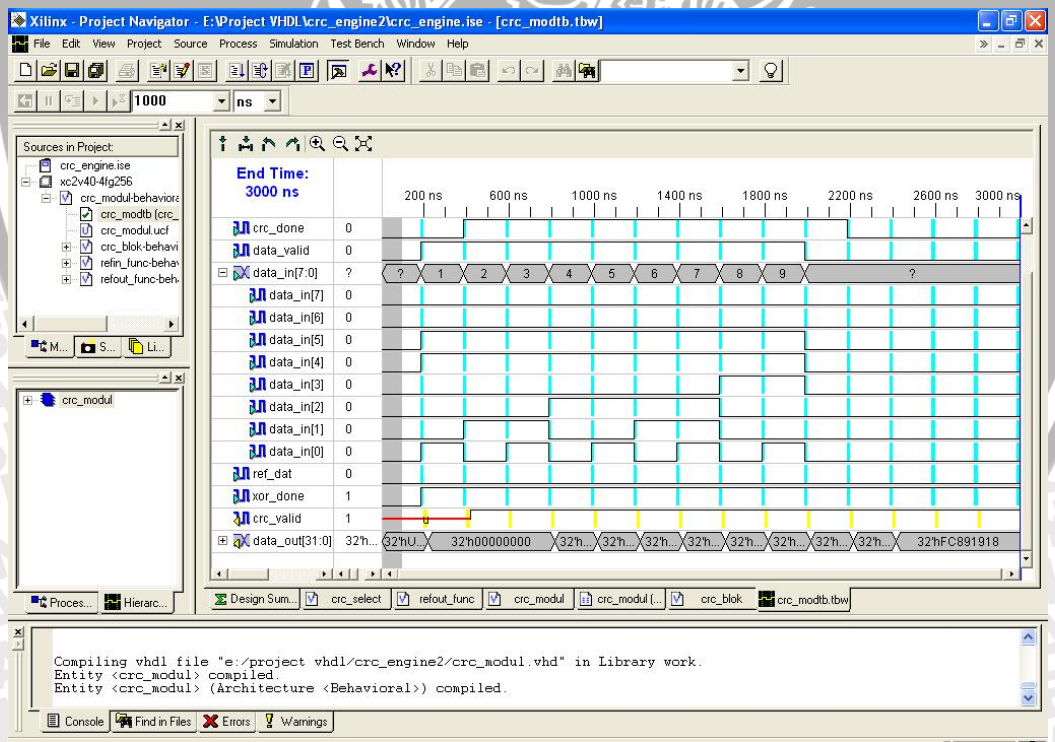
5.1.4 Modul `crc_main`

5.1.4.1 Pengujian Modul `crc_main`

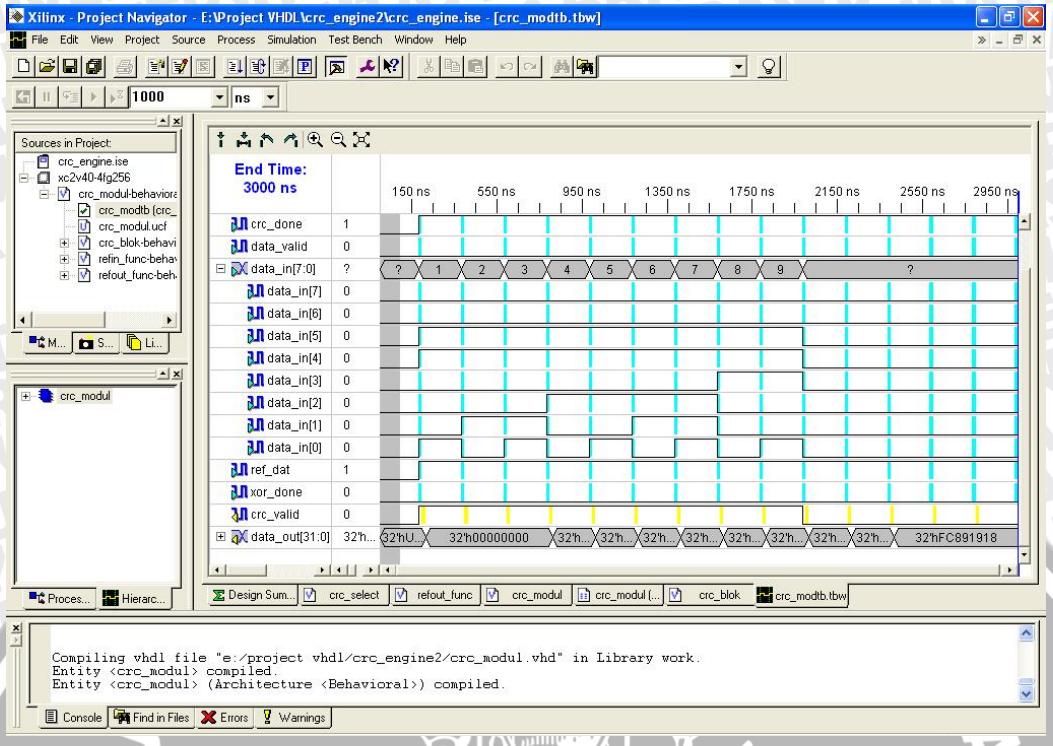
Pengujian modul utama `crc_main` dilakukan dengan simulasi *test bench* menggunakan *test vector* data masukan dan parameter sesuai dengan standart CRC 32 bit. Tabel standart parameter dan data masukan dapat dilihat dalam Tabel 2.1 dalam BAB II. Pengujian modul utama ini juga untuk mengetahui sinkronisasi kinerja antar modul-modul dalam modul utama. Hasil pengujian untuk modul ini dapat dilihat dalam Gambar 5.6 sampai.5.10 dan Tabel 5.4.



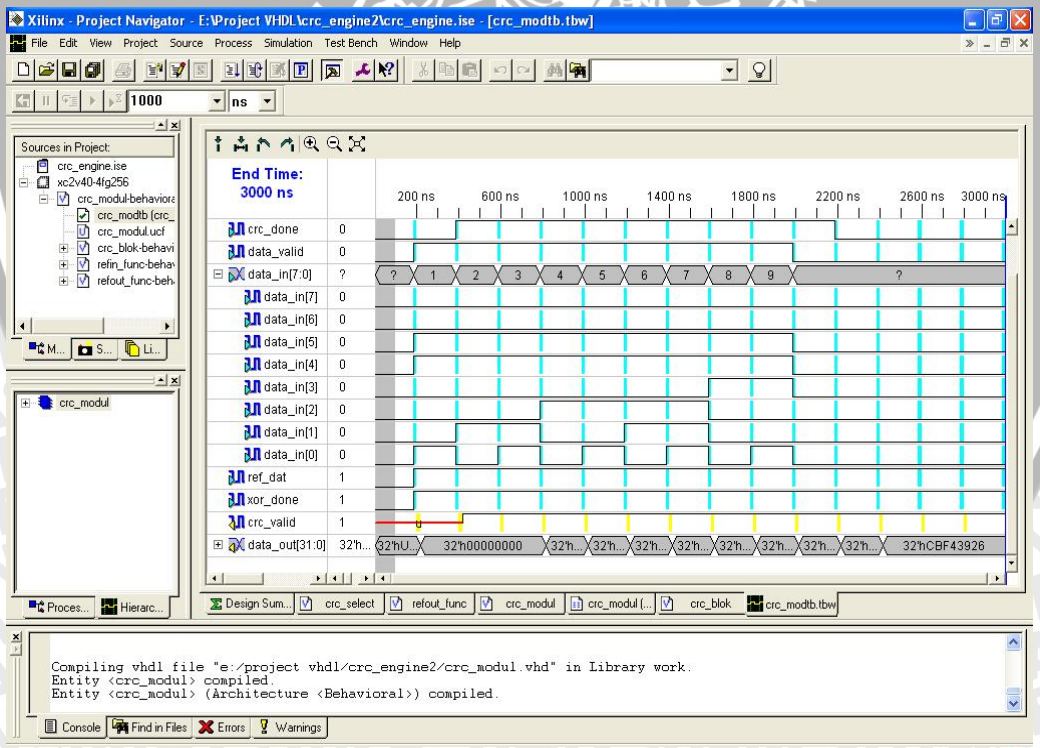
Gambar 5.6 Hasil simulasi modul `crc_main` untuk `ref_dat` = '0' dan `xor_done` = '0



Gambar 5.7 Hasil simulasi modul `crc_main` untuk `ref_dat` = '0' dan `xor_done` = '1'



Gambar 5.8 Hasil simulasi modul `crc_main` untuk `ref_dat` = '1' dan `xor_done` = '0'



Gambar 5.9 Hasil simulasi modul `crc_main` untuk `ref_dat` = '1' dan `xor_done` = '1'

Tabel 5.4 Tabel hasil pengujian modul crc_main

data_in	rst	data_valid	ref_dat	xor_done	crc_done	data_out
“123456789” ASCII	1	1	0	0	1	“0376E6E7” H
	1	1	0	1	1	“FC891918” H
	1	1	1	0	1	“340BC6D9” H
	1	1	1	1	1	“CBF43926” H

5.1.4.2 Analisa Hasil Pengujian Modul `crc_main`

Untuk pengujian pada modul utama ini, data masukan yang dipakai merupakan standart parameter dari CRC 32 bit. Pin `rst` diberi logika ‘1’, sedang pin `data_valid` akan diberi logika ‘1’ pada data-data yang akan dihitung nilai CRCnya. Dengan kata lain pin `data_valid` ini menandai data mana saja yang akan dihitung nilai CRCnya. Pin `crc_done` akan diberi logika ‘1’ saat akan dilakukan proses perhitungan nilai CRC terhadap data masukan. Sedangkan pin `ref_dat` dan `xor_done` pemberian logika berdasarkan standart parameter yang ada untuk maisng-masing jenis algoritma CRC 32 bit yang dipilih. Pin `ref_dat` untuk memenuhi parameter *RefIn* dan *RefOut* . Sedang pin `xor_done` untuk memenuhi parameter *XorOut.*

Pada pengujian crc_main ini dilakukan pengujian dengan menggunakan semua kemungkinan kombinasi parameter yang ada, dengan menggunakan dua data masukan yaitu “123456789” ASCII. Seperti yang sudah dijelaskan di atas, hal ini dilakukan untuk menguji sinkroninasi kinerja antar modul di dalam modul utama. Apakah dapat berfungsi dengan baik atau tidak. Untuk standart CRC 32 bit yang digunakan pada aplikasi Ethernet semua parameter mempunyai nilai *True* sehingga pada simulasi pengujiannya semua pin diberi logika ‘1’. Data hasil pengujian selengkapnya juga dapat dilihat dalam Tabel 5.4 di atas.

Untuk mengetahui validitas keluaran modul ini, juga dilakukan perbandingan antara data keluaran berdasarkan standart dan simulasi. Data keluaran hasil simulasi mengacu pada Gambar 5.6 sampai 5.10 dan Tabel 5.4. Tabel perbandingan data keluaran berdasarkan perhitungan dan simulasi untuk dapat dilihat dalam Tabel 5.5.

Tabel 5.5 Tabel Perbandingan Data Keluaran Untuk Modul `crc_main`

Masukan	Parameter				Keluaran	
	Init	RefIn	RefOut	XorOut	Teori	Simulasi
"123456789" ASCII	FFFFFFFF	False	False	00000000	"0376E6E7" H	"0376E6E7" H
	FFFFFFFF	False	False	FFFFFFFF	"FC891918" H	"FC891918" H
	FFFFFFFF	True	True	00000000	"340BC6D9" H	"340BC6D9" H
	FFFFFFFF	True	True	FFFFFFFF	"CBF43926" H	"CBF43926" H

Berdasarkan tabel 5.5 di atas terlihat bahwa data keluaran antara berdasarkan teori dan simulasi adalah sama. Sehingga dapat disimpulkan bahwa fungsi pengkodeaan CRC untuk modul ini telah sesuai dengan standart yang ada.

Dalam satu periode clock IC *parallel 32 bit* CRC ini mampu menghitung nilai CRC untuk 8 bit atau 1 byte data secara lansung. Seperti terlihat dalam Gambar 5.6 sampai 5.10 dibutuhkan 11 clock untuk dapat menghasilkan nilai checksum dari data masukan.

Dari data hasil pengujian dalam Tabel 5.4 dan 5.5 juga dapat disimpulkan bahwa desain IC *parallel 32 bit* CRC ini juga bisa diterapkan untuk tiga standart model CRC 32 bit yang lain selain untuk aplikasi Ethernet, yaitu untuk aplikasi MPEG-2. JAMCRC, dan BZIPS.

5.2 Pengujian Perfomansi IC

Untuk mengetahui perfomansi dari IC maka dilakukan pengujian *Timimg Constraints.* Pengujian *Time Constraints* yang dilakukan meliputi pengujian *Period Constraints, OFFSET IN Constraints,* dan *OFFSET OUT Constraints.*

Metode *timing analysis* yang digunakan pada perancangan ini adalah *Static Timing Analysis.* Pada Xilinx *Static Timing Analysis* dapat dilakukan dengan menggunakan *tools Timing Analyzer*. Untuk memulai proses *timing analysis* menggunakan *Timing Analyzer* pertama yang dilakukan memilih modul yang akan dianalisa timing-nya pada *source windows.* Setelah itu *double* klik bagian *Analyze Post-Place & Route Static Timing (Timing Analyzer).* Bagian ini merupakan salah satu proses dari *implementation design* pada *Processes windows.* Pada pengujian ini akan dilakukan analisa terhadap :

5. *Maksimum Clock Frequencies*
6. *Slack Time*
7. *Critical Path*
8. *Delay*

Pada perancangan ini tidak dilakukan analisa untuk *pad to pad* dikarenakan pada perancangan ini tidak ditemukan *Combinational path delay*. *Timing Analysis* selain berdasarkan *Timing Analyzer Report* analisa juga dilakukan berdasarkan *Place and Route Report* dan *Asynchronous Delay Report*.

5.2.1 Analisa *Maksimum Clock Frequencies*

Tujuan dari perancangan IC ini adalah selain untuk merancang IC yang mampu melakukan perhitungan nilai checksum data masukan untuk aplikasi Ethernet juga untuk merancang IC CRC 32 bit yang *compatible* dengan Gigabit Ethernet dari segi kecepatan throughputnya. Sehingga variabel kecepatan throughput dari IC perlu diperhatikan. Untuk mengetahui throughput dari IC hasil perancangan ini dilakukan analisa terhadap *Maksimum Clock Frequencies.*

Dari *synthesis report* diketahui frekuensi maksimum dari IC ini adalah 128,535 MHz, ini berarti besar *troughput* untuk 1 jalur data adalah 128,535 Mbps. Sehingga untuk misalkan 9 byte data dengan input parallel 8 bit nilai troughput dari IC ini adalah :

$$\text{Throughput} = \frac{\text{? data}}{\text{? cycle}} \text{ x throughput/s x 8 bit}$$

$$\text{Throughput} = \frac{9}{9} \text{ x 128,535 MHz / s x 8 bit} = 1028{,}28 \text{ Mb/s}$$

$$= 1{,}03 \text{ Gb/s}$$

Dari hasil perhitungan troughput di atas, dapat disimpulkan bahwa hasil rancnagan dari IC *parallel CRC 32 bit* ini *compatible* untuk Gigabit Ethernet sehingga bisa diimplementasikan untuk aplikasi tersebut.

5.2.4 Analisa *Slack Time*

Slack merupakan selisih antara waktu yang dibutuhkan *(constrain)* dengan jumlah waktu yang datang (input dan delay). Penghitungan nilai *slack* dilakukan untuk mengetahui apakah *timing constraints* yang dilakukan berhasil atau tidak. Nilai *slack* yang positif maka ini menunjukkan bahwa *timing constraints* yang dilakukan berhasil sedangkan apabila *slack* bernilai negative berarti *timing constraints* yang dilakukan belum berhasil.

Hasil pengujian nilai *slack* pada perancangan ini dapat dilihat dalam Tabel 5.6 sampai 5.8. Tabel 5.6 menunjukan hasil pengujian nilai *slack* untuk *Period Constraints.* Sedang Tabel 5.7 menunjukkan hasil pengujian nilai *slack* untuk *OFFSET IN Constraints* dan Tabel 5.8 menunjukkan hasil pengujian nilai *slack* untuk *OFFSET OUT Constraints*.

Tabel 5.6 Hasil Analisa *Slack Time* untuk *Period Constraints*

No.	*Data Path*	*Slack Time* **(ns)**
1.	*xor_crc/C_19 to refout/tmp_12*	3,610
2.	*xor_crc/C_1 to refout/tmp_1*	3,653
3.	*xor_crc/xor_func/out_xor_30 to refout/tmp_30*	3,677
4.	*xor_crc/xor_func/out_xor_30 to refout/tmp_1*	3,715
5.	*xor_crc/C_5 to refout/tmp_5*	3,773

Tabel 5.7 Hasil Analisa *Slack Time* untuk *OFFSET IN Constraints*

No.	*Path Name*		*Slack Time* **(ns)**
	Data Path	*Clock Path*	
1.	*rst to xor_crc/xor_func/out_xor_22*	*clock to xor_crc/xor_func/out_xor_22*	8,055
2.	*rst to xor_crc/xor_func/out_xor_23*	*clock to xor_crc/xor_func/out_xor_23*	8,055
3.	*rst to xor_crc/xor_func/out_xor_29*	*clock to xor_crc/xor_func/out_xor_29*	8,057
4.	*rst to xor_crc/xor_func/out_xor_28*	*clock to xor_crc/xor_func/out_xor_28*	8,057
5.	*rst to xor_crc/xor_func/out_xor_6*	*clock to xor_crc/xor_func/out_xor_6*	8,061

Tabel 5.8 Hasil Analisa *Slack Time* untuk *OFFSET OUT Constraints*

No.	*Path Name*		*Slack Time* **(ns)**
	Clock Path	*Data Path*	
1.	*clock to refout/temp_17*	*refout/temp_21 to data_out<17>*	1,778
2.	*clock to refout/temp_21*	*refout/temp_21 to data_out<21>*	1,778

3.	*clock to refout/temp_3*	*refout/temp_21 to data_out<3>*	1,778
4.	*clock to refout/temp_4*	*refout/temp_21 to data_out<4>*	1,778
5.	*clock to refout/temp_26*	*refout/temp_21 to data_out<26>*	1,778

Dari hasil pengujian yang ditunjukkan Tabel 5.6 sampai 5.8 terlihat bahwa nilai *slack* yang dihasilkan bernilai positif semua. Sehingga dapat disimpulkan bahwa *Timing Constraints* berhasil dilakukan. Sehingga tidak terjadi *Hold Time Violation* dan *Setup Time Violation* yang harus dihindari agar rangkaian dapat beroperasi dengan baik.

5.2.5 Analisa *Critical Path*

Critical Path merupakan jalur pada rangkaian yang mengakibatkan waktu tunda yang paling lama, dapat ditentukan dengan melihat waktu delay per level rangkaian. Berdasarkan hasil *Asynchronous Delay Report* diketahui 20 daerah *Critical Path*, yang dapat dilihat dalam Tabel 5.9. Hasil *Net Delay* selengkapnya dapat dilihat dalam *Asynchronous Delay Report* pada lampiran.

Tabel 5.9 Hasil Pengujian 20 *Critical Path*

No.	*Max Delay (ns)*	*Net Nama*
1.	3,117	rst_IBUF
2.	2,560	ref_dat_IBUF
3.	2,180	xor_done_IBUF
4.	2,114	crc_done_IBUF
5.	1,508	refout/_n0037
6.	1,501	present_state_FFd2
7.	1,465	refin/temp<7>
8.	1,414	xor_crc/C<30>
9.	1,408	xor_crc/C<18>
10.	1,408	refin/temp<6>
11.	1,375	xor_crc/C<14>
12.	1,319	refout/_n0033
13.	1,283	refout/_n0045
14.	1,283	xor_crc/C<10>
15.	1,275	xor_crc/C<8>
16.	1,268	xor_crc/C<5>

17.	1,251	data_in_6_IBUF
18.	1,242	refout/_n0058
19.	1,238	refin/temp<1>
20.	1,207	xor_crc/C<31>

Dari Tabel 5.9 diketahui bahwa *Worst case delay path* adalah jalur rangkaian rst_IBUF dengan *maxsimum delay* sebesar 3,117 ns. Agar sebuah rangkaian dapat beroperasi secara baik, batas dari *Worst case delay path* adalah harus lebih kecil dibandingkan dengan nilai maksimum kecepatan clock atau dengan kata lain dapat diartikan *maksimum delay* yang terjadi harus lebih kecil dengan *minimum clock period* dari rancangan ini. *Minimum Clock period* pada perancangan ini adalah 7,780 ns sedang *maksimum delay* yang tejadi adalah 3,117 ns. Sehingga dapat disimpulkan bahwa dengan *maksimum delay* sebesar 3,117 ns rangkaian ini bias berjalan dengan baik.

5.2.3 Analisa *Delay*

Delay atau waktu tunda terjadi salah satunya karena masalah *logic* yaitu dimana tegangan pada rangkaian tidak dapat langsung berubah drastis dari 0 ke 5 volt misalnya atupun sebaliknya, melainkan perubahan terjadi secara kontinyu. Selain itu delay juga bias terjadi karena masalah *route* atau yang biasa disebut *sequential delay.*

Berdasarkan hasil *Place and Route Report* diketahui nilai rata-rata *connection delay* sebesar 0.928 ns dan maksimum *pin delay* sebesar 3,117 ns. Sedangkan nilai rata-rata untuk *pin delay* untuk 10 daerah *Critical Pad* adalah 1,876 ns. Rincian untuk jumlah delay pada masing-masing pin dapat dilihat dalam Tabel 5.10.

Tabel 5.10 Rincian *Pin Delay*

No.	Range Delay (d)	Jumlah Pin
1.	d < 1,00 ns	527
2.	1,00 ns < d < 2,00 ns	193
3.	2,00 ns < d < 3,00 ns	88
4.	3,00 ns < d < 4,00 ns	1
5.	4,00 ns < d < 5,00 ns	0

Seperti dijelaskan pada analisa di atas bahwa nilai maksimum delay tidak boleh lebih besar dibandingkan nilai minimum dari *clock period*. Sehingga dengan nilai *pin delay* maksimum sebesar 3,117 ns dan nilai *clock period* sebesar 7,780 ns, maka rancnagan ini dapat beroperasi secara baik.

BAB VI
PENUTUP

6.1 Kesimpulan

Dari pengujian dan analisis hasil perancangan dan implementasi *Parallel CRC 32 bit* ini diperoleh kesimpulan sebagai berikut :

1. Implementasi *Parallel CRC 32 bit* ini 105 *slices*, menggunakan 47 atau 53% IOB (*Input/Output Block*), menempati 138 atau 26% *slices flip-flops*, membutuhkan 144 LUT dan 2 *General Clock device* Xilinx Virtex2 XC2V40 *package* FG256 dengan frekuensi maksimum 128,535 MHz, periode minimum 7,780 ns pada *speed grade* -6 dan *maximum delay* sebesar 3,117 ns.
2. Test vector data masukan pada *Parallel CRC 32 bit* ini menghasilkan output yang sesuai dengan standar parameter CRC-32 untuk Ethernet.
3. Throughput dari *Parallel 32 bit CRC Computation* ini sebesar 1.03 Gb/s. Dengan troughput ini maka *Parallel CRC 32 bit* ini dapat diimplementasikan pada Gigabit Ethernet.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan perancangan *Parallel CRC 32 bit* ini antara lain :

1. Hasil throughput dari perancangan ini dapat ditingkatkan dengan menambah jumlah bit *parallelizem*-nya sehingga throughput yang dihasilkan bisa lebih besar.
2. Hasil perancangan *Parallel CRC 32 bit* ini dapat dikembangkan dan diterapkan untuk *error detection* pada aplikasi-aplikasi yang lain, antara lain untuk aplikasi MPEG-2. JAMCRC, dan BZIPS yang mempunyai standart polynomial yang sama dengan Ethernet.

DAFTAR PUSTAKA

Ashenden, P.J. 1990. *The VHDL Cookbook. First Edition*. Adelaide: Dept. Computer Science, University of Adelaide.

Agilent Tech. 2002. *10 Gigabit Ethernet and XAUI Interface*. Agilent Tech, USA.

Beach, Dr. David P. 2004. *A Mini Thesis : Design of CRC (Cyclic Redudancy Check) Circuit.*

Department of Electronics and Computer Technolog, Indiana State University. Terre Haute, Indiana.

Campobello, Giuseppe dkk. 2003. *Parallel Realization.* Department of Physics Uniersity of Messina. Italy.

Cook, Greg. 2009. *Catalogue of Parameterized CRC Algorithms.* http ://homepages.tesco.net/rainstorm/crc-catalogue.htm (didownload pada kamis, 19 Maret 2009, 7:48:58 PM)

Fujitsu Network Communication Inc. 2006. *Ethernet Tutorial*. Fujitsu Network Communication Inc.

Held, Gilbert. 2001. *Data Comunications Networking Device : Operation, Utilization and LAN and WAN Internetworking Fourth Edition.* John Wiley and Sons Ltd. USA.

Henriksson, Thomas dkk. 1993. *Implementation of Fast CRC Calculation.* Dept. of Engineering, Linkopings universitet. Linkoping, Swedia.

Koopman, Philip. 2002. *32-Bit Cyclic Redudancy Codes for Internet Applications.* ECE Departement & ICES Carnegie Mellon University. Piitsburgh, PA, USA.

Kopman, Philip. 2004. *Cyclic Redudancy Code (CRC) Polynomial Selection For Embedded Netsworks.* ECE Departement & ICES Carnegie Mellon University. Piitsburgh, PA, USA.

M.D.Shiehetal. 2001. *A Systematic Approach for Parallel CRC Computations.* Journal of Information Scienceand Engineering.

Norris, Mark. 2003. *Gigabit Ethernet : Technology and Aplication.* Artech House Telecommunications Library.

Pearson Education. 2004. *Error Detection and Correction.* Pearson Education.

Spiegel, Jan V. 2001. *Tutorial VHDL*. University of Pennsylvania Departement of Electrical Engineering.

Wlliams, Ross N. 1993. *A Painless Guide To CRC Error Detection Algorithms*. Australia

Wireless Network Spring. 2005. *Coding and Error Control*. Wireless Network Spring.

XESS Corporation. 2001. FPGA. XESS Corporation

Xilinx Inc. 2006. *Progammable Logic Design Quick Start Handbook*. Xilinx Inc.

Xilinx Inc. 2006. *Synthesis and Simulation Design Guide*. Xilinx Inc.

Xilinx Inc. 2006. *Synthesis and Verification Design Guide*. Xilinx Inc.

Xilinx Inc. 2004. *Virtex II FPGA Family : Complete Data Sheet*. Xilinx Inc.

Xilinx Inc. 2008. *Xilinx Timing Constrain User Guide*. Xilinx Inc.

LAMPIRAN

LAMPIRAN I
LISTING **PROGRAM** *PARALLEL CRC 32 BIT*

crc_main.vhd

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

entity crc_main is
    port ( clk            :  in  std_logic;
           rst            : in  std_logic;
           data_valid     : in std_logic;
           data_in        : in  std_logic_vector (7 downto 0);
           ref_dat        : in std_logic;
           xor_done       : in std_logic;
           crc_done       : in std_logic;
           crc_valid      : out std_logic;
           data_out       : out std_logic_vector (31 downto 0));
end crc_main;

architecture Behavioral of crc_main is

----declaration signal----
type state is (idle, assign, done);
signal present_state, next_state : state;
signal r_in    : std_logic_vector (7 downto 0);
signal din     : std_logic_vector (7 downto 0);
signal fcs     : std_logic_vector (31 downto 0);
signal fcs_out : std_logic_vector (31 downto 0);

---declaration component---
  component crc_blok
       port ( clk         : in std_logic;
              rst         : in std_logic;
              crc_done    : in std_logic;
              xor_done    : in std_logic;
              D           : in std_logic_vector (7 downto 0);
              dat_out     : out std_logic_vector (31 downto 0));
  end component;

  component refin_func
       port ( clk, rst   : std_logic;
              data       : in std_logic_vector (7 downto 0);
              qout       : out std_logic_vector (7 downto 0));
  end component;

  component refout_func
       port ( clk, rst   : std_logic;
              data       : in std_logic_vector (31 downto 0);
              qout       : out std_logic_vector (31 downto 0));
       end component;

begin

  refin : refin_func
       port map ( clk  => clk,
                  rst  => rst,
                  data => din,
                  qout => r_in);

  xor_crc : crc_blok
       port map ( clk       => clk,
                  rst       => rst,
                  crc_done  => crc_done,
                  xor_done  => xor_done,
                  D         => r_in,
                  dat_out   => fcs );
```

```
    refout : refout_func
        port map ( clk  => clk,
                   rst   => rst,
                   data  => fcs ,
                   qout  => fcs_out);

        sequential: process (rst, clk)
        begin
                if rst = '0' then
                        present_state <= idle;
            elsif clk'event and (clk = '1') then
                        present_state <= next_state;
            end if;
        end process sequential;

        crc_code : process (present_state, data_valid,data_in, crc_done, crc_valid)
        begin
        next_state      <= present_state;
                din     <= data_in;
        crc_valid       <= '0';

       case present_state is
                when idle =>
                        din <= x"00";
                        if data_valid = '1' then
                                next_state <= assign;
                        end if;
                when assign =>
                        if crc_done = '1' then
                                next_state <= done;
                        end if;
                when done =>
                crc_valid <= '1';
                        if data_valid = '0' then
                                next_state <= idle;
                        end if;
                when others =>
                        next_state <= idle;
        end case;

        end process crc_code;

end Behavioral;
```

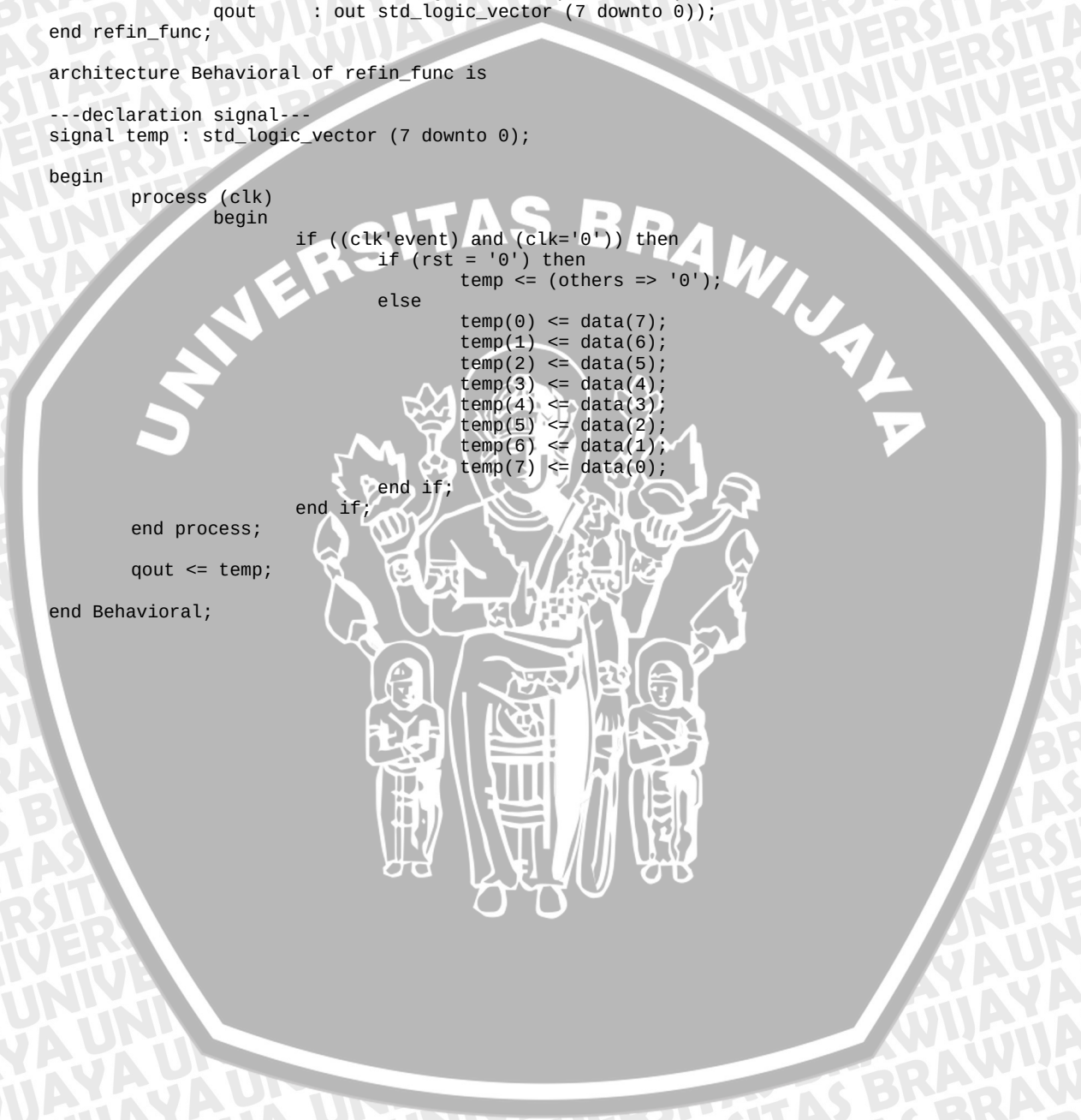
crc_blok.vhd

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

entity crc_blok is
        port ( clk        : in std_logic;
               rst        : in std_logic;
               crc_done   : in std_logic;
               xor_done   : in std_logic;
               D          : in std_logic_vector (7 downto 0);
               dat_out    : out std_logic_vector (31 downto 0));
end crc_blok;

architecture Behavioral of crc_blok is

---declaration signal---
signal C       : std_logic_vector (31 downto 0);
signal NewCRC  : std_logic_vector (31 downto 0);
signal xor1    : std_logic_vector (31 downto 0);
signal d_out   : std_logic_vector (31 downto 0);

---declaration component---
component xor_out
        port ( clk     : in std_logic;
               rst     : in std_logic;
              in_xor   : in std_logic_vector (31 downto 0);
              out_xor  : out std_logic_vector (31 downto 0));
end component;

begin
 xor_func : xor_out
        port map ( clk      => clk,
                   rst      => rst,
                   in_xor   => C,
                   out_xor  => xor1);

---CRC control logic---
crc_logic : process (clk,rst)
begin
        if (rst = '0') then
                C <= x"ffffffff";
        elsif (rising_edge(clk)) then
        if (crc_done = '0') then
                        C <= x"ffffffff";
                 else
                        C <= NewCRC;
                end if;
        end if;
end process;

---CRC Computation----
    NewCRC(0) <= D(6) xor D(0) xor C(24) xor C(30);
    NewCRC(1) <= D(7) xor D(6) xor D(1) xor D(0) xor C(24) xor C(25) xor
                 C(30) xor C(31);
    NewCRC(2) <= D(7) xor D(6) xor D(2) xor D(1) xor D(0) xor C(24) xor
                 C(25) xor C(26) xor C(30) xor C(31);
    NewCRC(3) <= D(7) xor D(3) xor D(2) xor D(1) xor C(25) xor C(26) xor
                 C(27) xor C(31);
    NewCRC(4) <= D(6) xor D(4) xor D(3) xor D(2) xor D(0) xor C(24) xor
                 C(26) xor C(27) xor C(28) xor C(30);
    NewCRC(5) <= D(7) xor D(6) xor D(5) xor D(4) xor D(3) xor D(1) xor
                 D(0) xor C(24) xor C(25) xor C(27) xor C(28) xor C(29) xor
                 C(30) xor C(31);
    NewCRC(6) <= D(7) xor D(6) xor D(5) xor D(4) xor D(2) xor D(1) xor
                 C(25) xor C(26) xor C(28) xor C(29) xor C(30) xor C(31);
    NewCRC(7) <= D(7) xor D(5) xor D(3) xor D(2) xor D(0) xor C(24) xor
                 C(26) xor C(27) xor C(29) xor C(31);
    NewCRC(8) <= D(4) xor D(3) xor D(1) xor D(0) xor C(0) xor C(24) xor
                 C(25) xor C(27) xor C(28);
    NewCRC(9) <= D(5) xor D(4) xor D(2) xor D(1) xor C(1) xor C(25) xor
                 C(26) xor C(28) xor C(29);
```

```
    NewCRC(10) <= D(5) xor D(3) xor D(2) xor D(0) xor C(2) xor C(24) xor
                  C(26) xor C(27) xor C(29);
    NewCRC(11) <= D(4) xor D(3) xor D(1) xor D(0) xor C(3) xor C(24) xor
                  C(25) xor C(27) xor C(28);
    NewCRC(12) <= D(6) xor D(5) xor D(4) xor D(2) xor D(1) xor D(0) xor
                  C(4) xor C(24) xor C(25) xor C(26) xor C(28) xor C(29) xor
                  C(30);
    NewCRC(13) <= D(7) xor D(6) xor D(5) xor D(3) xor D(2) xor D(1) xor
                  C(5) xor C(25) xor C(26) xor C(27) xor C(29) xor C(30) xor
                  C(31);
    NewCRC(14) <= D(7) xor D(6) xor D(4) xor D(3) xor D(2) xor C(6) xor
                  C(26) xor C(27) xor C(28) xor C(30) xor C(31);
    NewCRC(15) <= D(7) xor D(5) xor D(4) xor D(3) xor C(7) xor C(27) xor
                  C(28) xor C(29) xor C(31);
    NewCRC(16) <= D(5) xor D(4) xor D(0) xor C(8) xor C(24) xor C(28) xor
                  C(29);
    NewCRC(17) <= D(6) xor D(5) xor D(1) xor C(9) xor C(25) xor C(29) xor
                  C(30);
    NewCRC(18) <= D(7) xor D(6) xor D(2) xor C(10) xor C(26) xor C(30) xor
                  C(31);
    NewCRC(19) <= D(7) xor D(3) xor C(11) xor C(27) xor C(31);
    NewCRC(20) <= D(4) xor C(12) xor C(28);
    NewCRC(21) <= D(5) xor C(13) xor C(29);
    NewCRC(22) <= D(0) xor C(14) xor C(24);
    NewCRC(23) <= D(6) xor D(1) xor D(0) xor C(15) xor C(24) xor C(25) xor
                  C(30);
    NewCRC(24) <= D(7) xor D(2) xor D(1) xor C(16) xor C(25) xor C(26) xor
                  C(31);
    NewCRC(25) <= D(3) xor D(2) xor C(17) xor C(26) xor C(27);
    NewCRC(26) <= D(6) xor D(4) xor D(3) xor D(0) xor C(18) xor C(24) xor
                  C(27) xor C(28) xor C(30);
    NewCRC(27) <= D(7) xor D(5) xor D(4) xor D(1) xor C(19) xor C(25) xor
                  C(28) xor C(29) xor C(31);
    NewCRC(28) <= D(6) xor D(5) xor D(2) xor C(20) xor C(26) xor C(29) xor
                  C(30);
    NewCRC(29) <= D(7) xor D(6) xor D(3) xor C(21) xor C(27) xor C(30) xor
                  C(31);
    NewCRC(30) <= D(7) xor D(4) xor C(22) xor C(28) xor C(31);
    NewCRC(31) <= D(5) xor C(23) xor C(29);

---xor_func process---
       process (xor_done, C, xor1)
       begin
             if xor_done = '0' then
                    d_out <= C;
             else
                    d_out <= xor1;
             end if;
        end process;

 dat_out <= d_out;

end Behavioral;
```

refin_func.vhd

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

entity refin_func is
	port ( clk, rst : std_logic;
		data	: in std_logic_vector (7 downto 0);
		qout	: out std_logic_vector (7 downto 0));
end refin_func;

architecture Behavioral of refin_func is

---declaration signal---
signal temp : std_logic_vector (7 downto 0);

begin
	process (clk)
		begin
			if ((clk'event) and (clk='0')) then
				if (rst = '0') then
					temp <= (others => '0');
				else
					temp(0) <= data(7);
					temp(1) <= data(6);
					temp(2) <= data(5);
					temp(3) <= data(4);
					temp(4) <= data(3);
					temp(5) <= data(2);
					temp(6) <= data(1);
					temp(7) <= data(0);
				end if;
			end if;
	end process;

	qout <= temp;

end Behavioral;
```



refout_func.vhd

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

entity refout_func is
	port ( clk, rst : std_logic;
		data	: in std_logic_vector (31 downto 0);
		qout	: out std_logic_vector (31 downto 0));
end refout_func;

architecture Behavioral of refout_func is

---declaration signal---
 signal tmp : std_logic_vector (31 downto 0);

begin
	process (clk)
		begin
			if ((clk'event) and (clk='0')) then
				if (rst = '0') then
					tmp <= (others => '0');
				else
					tmp(0) <= data(31);
					tmp(1) <= data(30);
					tmp(2) <= data(29);
					tmp(3) <= data(28);
					tmp(4) <= data(27);
					tmp(5) <= data(26);
					tmp(6) <= data(25);
					tmp(7) <= data(24);
					tmp(8) <= data(23);
					tmp(9) <= data(22);
					tmp(10) <= data(21);
					tmp(11) <= data(20);
					tmp(12) <= data(19);
					tmp(13) <= data(18);
					tmp(14) <= data(17);
					tmp(15) <= data(16);
					tmp(16) <= data(15);
					tmp(17) <= data(14);
					tmp(18) <= data(13);
					tmp(19) <= data(12);
					tmp(20) <= data(11);
					tmp(21) <= data(10);
					tmp(22) <= data(9);
					tmp(23) <= data(8);
					tmp(24) <= data(7);
					tmp(25) <= data(6);
					tmp(26) <= data(5);
					tmp(27) <= data(4);
					tmp(28) <= data(3);
					tmp(29) <= data(2);
					tmp(30) <= data(1);
					tmp(31) <= data(0);
			end if;
		end if;
	end process;

	qout <= tmp;

end Behavioral;
```

xor_out.vhd

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

entity xor_out is
        port ( clk     : in std_logic;
               rst     : in std_logic;
               in_xor  : in std_logic_vector (31 downto 0);
               out_xor : out std_logic_vector (31 downto 0));
    end xor_out;

architecture Behavioral of xor_out is

begin
        xor_func:process (clk)
        begin
                if (rising_edge(clk)) then
                        if (rst= '1') then
                                out_xor <= in_xor xor x"FFFFFFFF";
                        end if;
                end if;
        end process;

end Behavioral;
```

LAMPIRAN 2
SYNTHESIS REPORT

```
Release 7.1i - xst H.38
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to __projnav
CPU : 0.00 / 2.16 s | Elapsed : 0.00 / 1.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 2.16 s | Elapsed : 0.00 / 1.00 s

--> Reading design: crc_modul.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) HDL Analysis
  4) HDL Synthesis
  5) Advanced HDL Synthesis
     5.1) HDL Synthesis Report
  6) Low Level Synthesis
  7) Final Report
     7.1) Device utilization summary
     7.2) TIMING REPORT


=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "crc_modul.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "crc_modul"
Output Format                      : NGC
Target Device                      : xc2v40-6-fg256

---- Source Options
Top Module Name                    : crc_modul
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
ROM Style                          : Auto
Mux Extraction                     : YES
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
Resource Sharing                   : YES
Multiplier Style                   : auto
Automatic Register Balancing       : No

---- Target Options
Add IO Buffers                     : YES
Global Maximum Fanout              : 500
Add Generic Clock Buffer(BUFG)     : 16
Register Duplication               : YES
Equivalent register Removal        : YES
Slice Packing                      : YES
Pack IO Registers into IOBs        : auto


---- General Options
Optimization Goal                  : Speed
```

```
Optimization Effort                 : 1
Keep Hierarchy                      : NO
Global Optimization                 : AllClockNets
RTL Output                          : Yes
Write Timing Constraints            : NO
Hierarchy Separator                 : /
Bus Delimiter                       : <>
Case Specifier                      : maintain
Slice Utilization Ratio             : 100
Slice Utilization Ratio Delta       : 5

---- Other Options
lso                                 : crc_modul.lso
Read Cores                          : YES
cross_clock_analysis                : NO
verilog2001                         : YES
safe_implementation                 : No
Optimize Instantiated Primitives    : NO
tristate2logic                      : Yes
use_clock_enable                    : Yes
use_sync_set                        : Yes
use_sync_reset                      : Yes
enable_auto_floorplanning           : No

=========================================================================


=========================================================================
*                          HDL Compilation                              *
=========================================================================
Compiling vhdl file "E:/Project VHDL/crc32/xor_out.vhd" in Library work.
Entity <xor_out> compiled.
Entity <xor_out> (Architecture <Behavioral>) compiled.
Compiling vhdl file "E:/Project VHDL/crc32/crc_select.vhd" in Library work.
Entity <refin_func> compiled.
Entity <refin_func> (Architecture <Behavioral>) compiled.
Compiling vhdl file "E:/Project VHDL/crc32/crc_blok.vhd" in Library work.
Entity <crc_blok> compiled.
Entity <crc_blok> (Architecture <Behavioral>) compiled.
Compiling vhdl file "E:/Project VHDL/crc32/refout_func.vhd" in Library work.
Entity <refout_func> compiled.
Entity <refout_func> (Architecture <Behavioral>) compiled.
Compiling vhdl file "E:/Project VHDL/crc32/crc_modul.vhd" in Library work.
Entity <crc_modul> compiled.
Entity <crc_modul> (Architecture <Behavioral>) compiled.

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing Entity <crc_modul> (Architecture <Behavioral>).
Entity <crc_modul> analyzed. Unit <crc_modul> generated.

Analyzing Entity <refin_func> (Architecture <behavioral>).
Entity <refin_func> analyzed. Unit <refin_func> generated.

Analyzing Entity <crc_blok> (Architecture <behavioral>).
Entity <crc_blok> analyzed. Unit <crc_blok> generated.

Analyzing Entity <xor_out> (Architecture <behavioral>).
Entity <xor_out> analyzed. Unit <xor_out> generated.

Analyzing Entity <refout_func> (Architecture <behavioral>).
Entity <refout_func> analyzed. Unit <refout_func> generated.


=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <xor_out>.
    Related source file is "E:/Project VHDL/crc32/xor_out.vhd".
    Found 32-bit register for signal <out_xor>.
    Summary:
        inferred  32 D-type flip-flop(s).
```

```
Unit <xor_out> synthesized.


Synthesizing Unit <refout_func>.
    Related source file is "E:/Project VHDL/crc32/refout_func.vhd".
    Found 32-bit register for signal <tmp>.
    Summary:
        inferred  32 D-type flip-flop(s).
Unit <refout_func> synthesized.


Synthesizing Unit <crc_blok>.
    Related source file is "E:/Project VHDL/crc32/crc_blok.vhd".
    Found 1-bit xor2 for signal <$n0001> created at line 86.
    Found 1-bit xor2 for signal <$n0002> created at line 86.
    Found 1-bit xor2 for signal <$n0003> created at line 88.
    Found 1-bit xor2 for signal <$n0004> created at line 90.
    Found 1-bit xor2 for signal <$n0005> created at line 90.
    Found 1-bit xor3 for signal <$n0006> created at line 93.
    Found 1-bit xor2 for signal <$n0007> created at line 93.
    Found 1-bit xor2 for signal <$n0008> created at line 93.
    Found 1-bit xor2 for signal <$n0009> created at line 96.
    Found 1-bit xor2 for signal <$n0012> created at line 98.
    Found 1-bit xor2 for signal <$n0013> created at line 98.
    Found 1-bit xor2 for signal <$n0019> created at line 104.
    Found 1-bit xor2 for signal <$n0024> created at line 67.
    Found 1-bit xor2 for signal <$n0030> created at line 71.
    Found 1-bit xor2 for signal <$n0035> created at line 75.
    Found 1-bit xor2 for signal <$n0036> created at line 75.
    Found 1-bit xor2 for signal <$n0038> created at line 80.
    Found 1-bit xor2 for signal <$n0039> created at line 80.
    Found 1-bit xor2 for signal <$n0041> created at line 82.
    Found 32-bit register for signal <C>.
    Found 1-bit xor2 for signal <NewCRC<31>>.
    Found 2-bit xor4 for signal <NewCRC<30:29>>.
    Found 2-bit xor4 for signal <NewCRC<28>>.
    Found 1-bit xor5 for signal <NewCRC<27>>.
    Found 1-bit xor4 for signal <NewCRC<26>>.
    Found 1-bit xor3 for signal <NewCRC<25>>.
    Found 2-bit xor2 for signal <NewCRC<24:23>>.
    Found 2-bit xor2 for signal <NewCRC<22>>.
    Found 2-bit xor3 for signal <NewCRC<21:20>>.
    Found 2-bit xor3 for signal <NewCRC<19>>.
    Found 3-bit xor5 for signal <NewCRC<18:16>>.
    Found 3-bit xor5 for signal <NewCRC<15>>.
    Found 1-bit xor4 for signal <NewCRC<14>>.
    Found 2-bit xor3 for signal <NewCRC<13:12>>.
    Found 2-bit xor3 for signal <NewCRC<11>>.
    Found 4-bit xor4 for signal <NewCRC<10:7>>.
    Found 4-bit xor4 for signal <NewCRC<6>>.
    Found 1-bit xor4 for signal <NewCRC<5>>.
    Found 3-bit xor2 for signal <NewCRC<4:2>>.
    Found 3-bit xor2 for signal <NewCRC<1>>.
    Found 1-bit xor2 for signal <NewCRC<0>>.
    Summary:
        inferred  32 D-type flip-flop(s).
        inferred  27 Xor(s).
Unit <crc_blok> synthesized.


Synthesizing Unit <refin_func>.
    Related source file is "E:/Project VHDL/crc32/crc_select.vhd".
    Found 8-bit register for signal <temp>.
    Summary:
        inferred   8 D-type flip-flop(s).
Unit <refin_func> synthesized.


Synthesizing Unit <crc_modul>.
    Related source file is "E:/Project VHDL/crc32/crc_modul.vhd".
    Found finite state machine <FSM_0> for signal <present_state>.
    -----------------------------------------------------------------------
    | States             | 3                                              |
    | Transitions        | 6                                              |
```

```
    | Inputs             | 2                                          |
    | Outputs            | 2                                          |
    | Clock              | clk (rising_edge)                          |
    | Reset              | rst (negative)                             |
    | Reset type         | asynchronous                               |
    | Reset State        | idle                                       |
    | Power Up State     | idle                                       |
    | Encoding           | automatic                                  |
    | Implementation     | LUT                                        |
    -----------------------------------------------------------------------
    Summary:
        inferred   1 Finite State Machine(s).
Unit <crc_modul> synthesized.


=========================================================================
*                            Advanced HDL Synthesis                     *
=========================================================================

Advanced RAM inference ...
Advanced multiplier inference ...
Advanced Registered AddSub inference ...
Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <FSM_0> on signal <present_state[1:2]> with gray encoding.
--------------------
 State  | Encoding
--------------------
 idle   | 00
 assign | 01
 done   | 11
--------------------
Dynamic shift register inference ...

=========================================================================
HDL Synthesis Report

Macro Statistics
# FSMs                          : 1
# Registers                     : 44
 1-bit register                 : 42
 32-bit register                : 2
# Xors                          : 51
 1-bit xor2                     : 24
 1-bit xor3                     : 13
 1-bit xor4                     : 10
 1-bit xor5                     : 4

=========================================================================

=========================================================================
*                            Low Level Synthesis                        *
=========================================================================
WARNING:Xst:1989 - Unit <crc_blok>: instances <Mxor__n0041>, <Mxor__n0003> of unit
<LPM_XOR2_1> are equivalent, second instance is removed
WARNING:Xst:1989 - Unit <crc_blok>: instances <Mxor__n0038>, <Mxor__n0001> of unit
<LPM_XOR2_1> are equivalent, second instance is removed

Optimizing unit <crc_modul> ...

Optimizing unit <crc_blok> ...

Optimizing unit <refin_func> ...

Optimizing unit <refout_func> ...

Optimizing unit <xor_out> ...
Loading device for application Rf_Device from file '2v40.nph' in environment
C:/Program Files/Xilinx.

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block crc_modul, actual ratio is 35.
=========================================================================
*                              Final Report                             *
```

```
=========================================================================
Final Results
RTL Top Level Output File Name     : crc_modul.ngr
Top Level Output File Name         : crc_modul
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 47

Macro Statistics :
# Registers                        : 42
#      1-bit register              : 40
#      32-bit register             : 2
# Xors                             : 27
#      1-bit xor3                  : 13
#      1-bit xor4                  : 10
#      1-bit xor5                  : 4

Cell Usage :
# BELS                             : 208
#      INV                         : 33
#      LUT2                        : 4
#      LUT2_L                      : 2
#      LUT3                        : 10
#      LUT3_L                      : 71
#      LUT4                        : 15
#      LUT4_D                      : 4
#      LUT4_L                      : 37
#      MUXF5                       : 32
# FlipFlops/Latches                : 106
#      FDC                         : 2
#      FDC_1                       : 40
#      FDE                         : 32
#      FDPE                        : 32
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 46
#      IBUF                        : 13
#      OBUF                        : 33
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v40fg256-6

 Number of Slices:                       80  out of    256    31%
 Number of Slice Flip Flops:            106  out of    512    20%
 Number of 4 input LUTs:                143  out of    512    27%
 Number of bonded IOBs:                  47  out of     88    53%
 Number of GCLKs:                         1  out of     16     6%


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 106   |
-----------------------------------+------------------------+-------+


Timing Summary:
---------------
```

```
Speed Grade: -6

   Minimum period: 5.680ns (Maximum Frequency: 176.056MHz)
   Minimum input arrival time before clock: 2.668ns
   Maximum output required time after clock: 4.727ns
   Maximum combinational path delay: No path found

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

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 5.680ns (frequency: 176.056MHz)
  Total number of paths / destination ports: 424 / 106
-------------------------------------------------------------------------
Delay:               2.840ns (Levels of Logic = 2)
  Source:            refin/temp_1 (FF)
  Destination:       xor_crc/C_11 (FF)
  Source Clock:      clk falling
  Destination Clock: clk rising

  Data Path: refin/temp_1 to xor_crc/C_11
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC_1:C->Q           10   0.449   0.686  refin/temp_1 (refin/temp_1)
     LUT4_D:I3->O          4   0.347   0.718  xor_crc/Mxor_NewCRC<27>_Xo<1>1
(xor_crc/Mxor_NewCRC<27>_Xo<1>)
     LUT3_L:I1->LO         1   0.347   0.000  xor_crc/Mxor_NewCRC<11>_Xo<1>1
(xor_crc/NewCRC<11>)
     FDPE:D                    0.293          xor_crc/C_11
    ----------------------------------------
    Total                      2.840ns (1.436ns logic, 1.404ns route)
                               (50.6% logic, 49.4% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 187 / 106
-------------------------------------------------------------------------
Offset:              2.668ns (Levels of Logic = 3)
  Source:            xor_done (PAD)
  Destination:       refout/tmp_29 (FF)
  Destination Clock: clk falling

  Data Path: xor_done to refout/tmp_29
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            64   0.653   1.030  xor_done_IBUF (xor_done_IBUF)
     LUT3_L:I1->LO         1   0.347   0.000  refout/_n005911_F (N253)
     MUXF5:I0->O           1   0.345   0.000  refout/_n005911 (refout/_n0059)
     FDC_1:D                   0.293          refout/tmp_26
    ----------------------------------------
    Total                      2.668ns (1.638ns logic, 1.030ns route)
                               (61.4% logic, 38.6% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 33 / 33
-------------------------------------------------------------------------
Offset:              4.727ns (Levels of Logic = 1)
  Source:            present_state_FFd1 (FF)
  Destination:       crc_valid (PAD)
  Source Clock:      clk rising

  Data Path: present_state_FFd1 to crc_valid
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q              3   0.449   0.536  present_state_FFd1
(present_state_FFd1)
     OBUF:I->O                 3.743          crc_valid_OBUF (crc_valid)
    ----------------------------------------
```

```
    Total                       4.727ns (4.192ns logic, 0.536ns route)
                                        (88.7% logic, 11.3% route)

=====================================================================
CPU : 9.73 / 12.02 s | Elapsed : 10.00 / 11.00 s

-->

Total memory usage is 117332 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    2 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

LAMPIRAN 3

TRANSLATION REPORT

```
Release 7.1i ngdbuild H.38
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

Command Line: ngdbuild -intstyle ise -dd "e:\project vhdl\crc_engine2/_ngo" -uc
crc_modul.ucf -p xc2v40-fg256-4 crc_modul.ngc crc_modul.ngd

Reading NGO file 'E:/Project VHDL/crc_engine2/crc_modul.ngc' ...

Applying constraints in "crc_modul.ucf" to the design...

Checking timing specifications ...
Checking expanded design ...

NGDBUILD Design Results Summary:
  Number of errors:     0
  Number of warnings:   0

Total memory usage is 83416 kilobytes

Writing NGD file "crc_modul.ngd" ...

Writing NGDBUILD log file "crc_modul.bld"...
```

LAMPIRAN 4
MAPPING REPORT

```
Release 7.1i Map H.38

Xilinx Mapping Report File for Design 'crc_modul'
Design Information
------------------
Command Line   : C:/Program Files/Xilinx/bin/nt/map.exe -ise e:\project
vhdl\crc32\crc32.ise -intstyle ise -p xc2v40-fg256-6 -cm area -pr b -k 4 -c 100
-tx off -o crc_modul_map.ncd crc_modul.ngd crc_modul.pcf
Target Device  : xc2v40
Target Package : fg256
Target Speed   : -6
Mapper Version : virtex2 -- $Revision: 1.26.6.3 $
Mapped Date    : Wed Jun 10 06:22:09 2009

Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:          74 out of     512   14%
  Number of 4 input LUTs:             143 out of     512   27%
Logic Distribution:
  Number of occupied Slices:           89 out of     256   34%
  Number of Slices containing only related logic:      89 out of      89  100%
  Number of Slices containing unrelated logic:          0 out of      89    0%
       *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:           143 out of     512   27%

  Number of bonded IOBs:              47 out of      88   53%
    IOB Flip Flops:                   32
  Number of GCLKs:                     1 out of      16    6%

Total equivalent gate count for design:  1,805
Additional JTAG gate count for IOBs:  2,256
Peak Memory Usage:  122 MB

NOTES:

   Related logic is defined as being logic that shares connectivity - e.g. two
   LUTs are "related" if they share common inputs.  When assembling slices,
   Map gives priority to combine logic that is related.  Doing so results in
   the best timing performance.

   Unrelated logic shares no connectivity.  Map will only begin packing
   unrelated logic into a slice once 99% of the slices are occupied through
   related logic packing.

   Note that once logic distribution reaches the 99% level through related
   logic packing, this does not mean the device is completely utilized.
   Unrelated logic packing will then begin, continuing until all usable LUTs
   and FFs are occupied.  Depending on your timing budget, increased levels of
   unrelated logic packing may adversely affect the overall timing performance
   of your design.

Table of Contents
-----------------
Section 1 - Errors
Section 2 - Warnings
Section 3 - Informational
Section 4 - Removed Logic Summary
Section 5 - Removed Logic
Section 6 - IOB Properties
Section 7 - RPMs
Section 8 - Guide Report
Section 9 - Area Group Summary
Section 10 - Modular Design Summary
Section 11 - Timing Report
Section 12 - Configuration String Information
Section 13 - Additional Device Resource Counts

Section 1 - Errors

------------------

Section 2 - Warnings
--------------------
```

Section 3 - Informational

INFO:MapLib:562 - No environment variables are currently set.
INFO:MapLib:535 - The following Virtex BUFG(s) is/are being retargetted to Virtex2 BUFGMUX(s) with input tied to I0 and Select pin tied to constant 0: BUFGP symbol "clk_BUFGP" (output signal=clk_BUFGP)
INFO:LIT:244 - All of the single ended outputs in this design are using slew rate limited output drivers. The delay on speed critical single ended outputs can be dramatically reduced by designating them as fast outputs in the schematic.

Section 4 - Removed Logic Summary

Section 5 - Removed Logic

To enable printing of redundant blocks removed and signals merged, set the detailed map report option and rerun map.

Section 6 - IOB Properties

IOB Name	Type	Direction	IO Standard	Drive Strength	Slew Rate	Reg (s)	Resistor	IOB Delay
clk	IOB	INPUT	LVTTL					
crc_done	IOB	INPUT	LVTTL					
crc_valid	IOB	OUTPUT	LVTTL	12	SLOW			
data_in<0>	IOB	INPUT	LVTTL					
data_in<1>	IOB	INPUT	LVTTL					
data_in<2>	IOB	INPUT	LVTTL					
data_in<3>	IOB	INPUT	LVTTL					
data_in<4>	IOB	INPUT	LVTTL					
data_in<5>	IOB	INPUT	LVTTL					
data_in<6>	IOB	INPUT	LVTTL					
data_in<7>	IOB	INPUT	LVTTL					
data_out<0>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<1>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<2>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<3>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<4>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<5>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<6>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<7>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<8>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<9>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<10>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<11>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<12>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<13>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<14>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<15>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<16>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<17>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<18>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<19>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<20>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<21>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<22>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<23>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<24>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<25>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<26>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<27>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<28>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<29>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<30>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_out<31>	IOB	OUTPUT	LVTTL	12	SLOW	OFF1		
data_valid	IOB	INPUT	LVTTL					
ref_dat	IOB	INPUT	LVTTL					
rst	IOB	INPUT	LVTTL					
xor_done	IOB	INPUT	LVTTL					

Section 7 - RPMs

Section 8 - Guide Report

Guide not run on this design.

Section 9 - Area Group Summary

```
----------------------------
No area groups were found in this design.

Section 10 - Modular Design Summary
-----------------------------------
Modular Design not used for this design.

Section 11 - Timing Report
--------------------------
This design was not run using timing mode.

Section 12 - Configuration String Details
-------------------------
Use the "-detail" map option to print out Configuration Strings

Section 13 - Additional Device Resource Counts
---------------------------------------------
Number of JTAG Gates for IOBs = 47
Number of Equivalent Gates for Design = 1,805
Number of RPM Macros = 0
Number of Hard Macros = 0
CAPTUREs = 0
BSCANs = 0
STARTUPs = 0
PCILOGICs = 0
DCMs = 0
GCLKs = 1
ICAPs = 0
18X18 Multipliers = 0
Block RAMs = 0
TBUFs = 0
Total Registers (Flops & Latches in Slices & IOBs) not driven by LUTs = 64
IOB Dual-Rate Flops not driven by LUTs = 0
IOB Dual-Rate Flops = 0
IOB Slave Pads = 0
IOB Master Pads = 0
IOB Latches not driven by LUTs = 0
IOB Latches = 0
IOB Flip Flops not driven by LUTs = 32
IOB Flip Flops = 32
Unbonded IOBs = 0
Bonded IOBs = 47
ORCYs = 0
XORs = 0
CARRY_INITs = 0
CARRY_SKIPs = 0
CARRY_MUXes = 0
Total Shift Registers = 0
Static Shift Registers = 0
Dynamic Shift Registers = 0
16x1 ROMs = 0
16x1 RAMs = 0
32x1 RAMs = 0
Dual Port RAMs = 0
MUXFs = 32
MULT_ANDs = 0
4 input LUTs used as Route-Thrus = 0
4 input LUTs = 143
Slice Latches not driven by LUTs = 0
Slice Latches = 0
Slice Flip Flops not driven by LUTs = 32
Slice Flip Flops = 74
Slices = 89
F6 Muxes = 0
F5 Muxes = 32
F8 Muxes = 0
F7 Muxes = 0
Number of LUT signals with 4 loads = 1
Number of LUT signals with 3 loads = 0
```

LAMPIRAN 5

PALCE AND ROUTE REPORT

```
Release 7.1i par H.38
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

IMA :  Wed Jun 10 06:22:12 2009

par -w -intstyle ise -ol std -t 1 crc_modul_map.ncd crc_modul.ncd crc_modul.pcf
```

```
Constraints file: crc_modul.pcf.
Loading device for application Rf_Device from file '2v40.nph' in environment
C:/Program Files/Xilinx.
   "crc_modul" is an NCD, version 3.1, device xc2v40, package fg256, speed -6

Initializing temperature to 85.000 Celsius. (default - Range: 0.000 to 85.000
Celsius)
Initializing voltage to 1.425 Volts. (default - Range: 1.425 to 1.575 Volts)


Device speed data version:  "PRODUCTION 1.120 2005-01-22".


Device Utilization Summary:

   Number of BUFGMUXs                    1 out of 16      6%
   Number of External IOBs              47 out of 88     53%
      Number of LOCed IOBs               0 out of 47      0%

   Number of SLICEs                     89 out of 256    34%


Overall effort level (-ol):   Standard (set by user)
Placer effort level (-pl):    Standard (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    Standard (set by user)

Starting initial Timing Analysis.  REAL time: 1 secs
Finished initial Timing Analysis.  REAL time: 1 secs


Starting Placer

Phase 1.1
Phase 1.1 (Checksum:98981a) REAL time: 1 secs

Phase 2.31
Phase 2.31 (Checksum:1312cfe) REAL time: 1 secs

Phase 3.2
.


Phase 3.2 (Checksum:1c9c37d) REAL time: 5 secs

Phase 4.30
Phase 4.30 (Checksum:26259fc) REAL time: 5 secs

Phase 5.3
Phase 5.3 (Checksum:2faf07b) REAL time: 5 secs

Phase 6.5
Phase 6.5 (Checksum:39386fa) REAL time: 5 secs

Phase 7.8
.....
.....
Phase 7.8 (Checksum:9c07ec) REAL time: 5 secs

Phase 8.5
Phase 8.5 (Checksum:4c4b3f8) REAL time: 5 secs

Phase 9.18
Phase 9.18 (Checksum:55d4a77) REAL time: 5 secs

Phase 10.5
Phase 10.5 (Checksum:5f5e0f6) REAL time: 5 secs

Phase 11.24
Phase 11.24 (Checksum:68e7775) REAL time: 5 secs

Phase 12.27
Phase 12.27 (Checksum:7270df4) REAL time: 5 secs
```

```
Writing design to file crc_modul.ncd


Total REAL time to Placer completion: 5 secs
Total CPU time to Placer completion: 5 secs

Starting Router

Phase 1: 810 unrouted;       REAL time: 6 secs

Phase 2: 728 unrouted;       REAL time: 6 secs

Phase 3: 157 unrouted;       REAL time: 6 secs

Phase 4: 157 unrouted; (0)      REAL time: 6 secs

Phase 5: 157 unrouted; (0)      REAL time: 6 secs

Phase 6: 157 unrouted; (0)      REAL time: 6 secs

Phase 7: 0 unrouted; (0)      REAL time: 6 secs

Phase 8: 0 unrouted; (0)      REAL time: 6 secs


Total REAL time to Router completion: 6 secs
Total CPU time to Router completion: 6 secs

Generating "PAR" statistics.

**************************
Generating Clock Report
**************************

+---------------------+--------------+------+------+------------+-------------+
|        Clock Net    |   Resource   |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+---------------------+--------------+------+------+------------+-------------+
|           clk_BUFGP |     BUFGMUX0S| No   |   81 |  0.141     |  0.520      |
+---------------------+--------------+------+------+------------+-------------+


   The Delay Summary Report


The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is: 0

   The AVERAGE CONNECTION DELAY for this design is:        0.928
   The MAXIMUM PIN DELAY IS:                               3.117
   The AVERAGE CONNECTION DELAY on the 10 WORST NETS is:   1.876

   Listing Pin Delays by value: (nsec)

    d < 1.00   < d < 2.00  < d < 3.00  < d < 4.00  < d < 5.00  d >= 5.00
    ---------  ---------   ---------   ---------   ---------   ---------
          527         193          88           1           0           0

Timing Score: 0

Asterisk (*) preceding a constraint indicates it was not met.
   This may be due to a setup or hold violation.

--------------------------------------------------------------------------------
  Constraint                                | Requested  | Actual     | Logic
                                            |            |            | Levels
--------------------------------------------------------------------------------
  TS_clk = PERIOD TIMEGRP "clk" 15 ns HIGH  | 15.000ns   | 7.780ns    | 1
  50%                                       |            |            |
--------------------------------------------------------------------------------
  OFFSET = IN 10 ns BEFORE COMP "clk"       | 15.000ns   | 1.945ns    | 2
--------------------------------------------------------------------------------
  OFFSET = OUT 20 ns AFTER COMP "clk"       | 20.000ns   | 13.222ns   | 0
--------------------------------------------------------------------------------
```

```
All constraints were met.
Generating Pad Report.

All signals are completely routed.

Total REAL time to PAR completion: 7 secs
Total CPU time to PAR completion: 6 secs

Peak Memory Usage:  75 MB

Placement: Completed - No errors found.
Routing: Completed - No errors found.
Timing: Completed - No errors found.

Number of error messages: 0
Number of warning messages: 0
Number of info messages: 0

Writing design to file crc_modul.ncd


PAR done!
```

LAMPIRAN 6
STATIC TIMING REPORT

```
Release 7.1i Trace H.38
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

C:/Program Files/Xilinx/bin/nt/trce.exe -ise e:\project vhdl\crc32\crc32.ise
-intstyle ise -e 3 -l 3 -s 6 -xml crc_modul crc_modul.ncd -o crc_modul.twr
crc_modul.pcf


Design file:              crc_modul.ncd
Physical constraint file: crc_modul.pcf
Device,speed:             xc2v40,-6 (PRODUCTION 1.120 2005-01-22, STEPPING level
1)
Report level:             error report

Environment Variable      Effect
--------------------      ------
NONE                      No environment variables were set
--------------------------------------------------------------------------------

INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.

================================================================================
Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 15 ns HIGH 50%;

 424 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum period is   7.780ns.
--------------------------------------------------------------------------------

================================================================================
Timing constraint: OFFSET = IN 10 ns BEFORE COMP "clk";

 187 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum allowable offset is   1.945ns.
--------------------------------------------------------------------------------

================================================================================
Timing constraint: OFFSET = OUT 20 ns AFTER COMP "clk";

 33 items analyzed, 0 timing errors detected.
 Minimum allowable offset is  13.222ns.
--------------------------------------------------------------------------------


All constraints were met.


Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
------------+------------+------------+------------------+--------+
            |  Setup to  |  Hold to   |                  |  Clock |
Source      | clk (edge) | clk (edge) |Internal Clock(s) |  Phase |
------------+------------+------------+------------------+--------+
crc_done    |    1.186(R)|   -0.015(R)|clk_BUFGP         |   0.000|
data_in<0>  |   -7.017(F)|    7.475(F)|clk_BUFGP         |   7.500|
data_in<1>  |   -7.128(F)|    7.505(F)|clk_BUFGP         |   7.500|
data_in<2>  |   -7.299(F)|    7.538(F)|clk_BUFGP         |   7.500|
data_in<3>  |   -6.780(F)|    7.024(F)|clk_BUFGP         |   7.500|
data_in<4>  |   -6.922(F)|    7.160(F)|clk_BUFGP         |   7.500|
data_in<5>  |   -6.862(F)|    7.106(F)|clk_BUFGP         |   7.500|
data_in<6>  |   -6.726(F)|    7.105(F)|clk_BUFGP         |   7.500|
data_in<7>  |   -7.005(F)|    7.444(F)|clk_BUFGP         |   7.500|
```

```
data_valid  |   -0.218(R)|    0.464(R)|clk_BUFGP         |   0.000|
ref_dat     |   -3.801(F)|    7.194(F)|clk_BUFGP         |   7.500|
rst         |    1.945(R)|   -0.553(R)|clk_BUFGP         |   0.000|
xor_done    |   -4.109(F)|    6.610(F)|clk_BUFGP         |   7.500|
------------+------------+------------+------------------+--------+

Clock clk to Pad
------------+------------+------------------+--------+
            | clk (edge) |                  | Clock  |
Destination | to PAD     |Internal Clock(s) | Phase  |
------------+------------+------------------+--------+
crc_valid   |    6.597(R)|clk_BUFGP         |   0.000|
data_out<0> |   13.217(F)|clk_BUFGP         |   7.500|
data_out<10>|   13.217(F)|clk_BUFGP         |   7.500|
data_out<11>|   13.208(F)|clk_BUFGP         |   7.500|
data_out<12>|   13.207(F)|clk_BUFGP         |   7.500|
data_out<13>|   13.208(F)|clk_BUFGP         |   7.500|
data_out<14>|   13.220(F)|clk_BUFGP         |   7.500|
data_out<15>|   13.217(F)|clk_BUFGP         |   7.500|
data_out<16>|   13.207(F)|clk_BUFGP         |   7.500|
data_out<17>|   13.222(F)|clk_BUFGP         |   7.500|
data_out<18>|   13.221(F)|clk_BUFGP         |   7.500|
data_out<19>|   13.221(F)|clk_BUFGP         |   7.500|
data_out<1> |   13.217(F)|clk_BUFGP         |   7.500|
data_out<20>|   13.216(F)|clk_BUFGP         |   7.500|
data_out<21>|   13.222(F)|clk_BUFGP         |   7.500|
data_out<22>|   13.208(F)|clk_BUFGP         |   7.500|
data_out<23>|   13.221(F)|clk_BUFGP         |   7.500|
data_out<24>|   13.218(F)|clk_BUFGP         |   7.500|
data_out<25>|   13.220(F)|clk_BUFGP         |   7.500|
data_out<26>|   13.222(F)|clk_BUFGP         |   7.500|
data_out<27>|   13.216(F)|clk_BUFGP         |   7.500|
data_out<28>|   13.216(F)|clk_BUFGP         |   7.500|
data_out<29>|   13.217(F)|clk_BUFGP         |   7.500|
data_out<2> |   13.220(F)|clk_BUFGP         |   7.500|
data_out<30>|   13.217(F)|clk_BUFGP         |   7.500|
data_out<31>|   13.222(F)|clk_BUFGP         |   7.500|
data_out<3> |   13.222(F)|clk_BUFGP         |   7.500|
data_out<4> |   13.222(F)|clk_BUFGP         |   7.500|
data_out<5> |   13.220(F)|clk_BUFGP         |   7.500|
data_out<6> |   13.208(F)|clk_BUFGP         |   7.500|
data_out<7> |   13.207(F)|clk_BUFGP         |   7.500|
data_out<8> |   13.216(F)|clk_BUFGP         |   7.500|
data_out<9> |   13.207(F)|clk_BUFGP         |   7.500|
------------+------------+------------------+--------+

Clock to Setup on destination clock clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clk            |    3.381|    3.279|    3.890|         |
---------------+---------+---------+---------+---------+


Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 644 paths, 0 nets, and 744 connections

Design statistics:
   Minimum period:   7.780ns (Maximum frequency: 128.535MHz)
   Minimum input required time before clock:   1.945ns
   Minimum output required time after clock:  13.222ns

Analysis completed Wed Jun 10 06:22:21 2009
--------------------------------------------------------------------------------


Peak Memory Usage: 77 MB
```

LAMPIRAN 7
TIMING ANALYZER REPORT

```
Release 7.1i - Timing Analyzer H.38
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

Design file:              E:\Project VHDL\crc32\crc_modul.ncd
Physical constraint file: E:\Project VHDL\crc32\crc_modul.pcf
Device,speed:             xc2v40,-6 (PRODUCTION 1.120 2005-01-22, STEPPING level
1)
Report level:             verbose report, limited to 5 items per constraint

Environment Variable      Effect
--------------------      ------
NONE                      No environment variables were set
--------------------------------------------------------------------------------

INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.

================================================================================
Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 15 ns HIGH 50%;

 424 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum period is   7.780ns.
--------------------------------------------------------------------------------
Slack:                  3.610ns (requirement - (data path - clock path skew +
uncertainty))
  Source:               xor_crc/C_19 (FF)
  Destination:          refout/tmp_12 (FF)
  Requirement:          7.500ns
  Data Path Delay:      3.759ns (Levels of Logic = 1)
  Clock Path Skew:      -0.131ns
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP falling at 7.500ns
  Clock Uncertainty:    0.000ns

  Data Path: xor_crc/C_19 to refout/tmp_12
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tcko                  0.449   xor_crc/C_19
    net (fanout=4)        1.112   xor_crc/C<19>
    Tif5x                 0.692   refout/_n004411_G
                                  refout/_n004411
    net (fanout=1)        1.199   refout/_n0044
    Tioock                0.307   refout/tmp_12
    -------------------------  -------------------------
    Total                 3.759ns (1.448ns logic, 2.311ns route)
                                  (38.5% logic, 61.5% route)

--------------------------------------------------------------------------------
Slack:                  3.653ns (requirement - (data path - clock path skew +
uncertainty))
  Source:               xor_crc/C_1 (FF)
  Destination:          refout/tmp_1 (FF)
  Requirement:          7.500ns
  Data Path Delay:      3.726ns (Levels of Logic = 1)
  Clock Path Skew:      -0.121ns
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP falling at 7.500ns
  Clock Uncertainty:    0.000ns

  Data Path: xor_crc/C_1 to refout/tmp_1
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tcko                  0.449   xor_crc/C_1
    net (fanout=4)        0.959   xor_crc/C<1>
    Tif5x                 0.692   refout/_n003311_F
```

```
                                        refout/_n003311
    net (fanout=1)        1.319   refout/_n0033
    Tioock                0.307   refout/tmp_1
    ----------------------------  ------------------------
    Total                 3.726ns (1.448ns logic, 2.278ns route)
                                  (38.9% logic, 61.1% route)

--------------------------------------------------------------------------------
Slack:                  3.677ns (requirement - (data path - clock path skew +
uncertainty))
  Source:               xor_crc/xor_func/out_xor_30 (FF)
  Destination:          refout/tmp_30 (FF)
  Requirement:          7.500ns
  Data Path Delay:      3.692ns (Levels of Logic = 1)
  Clock Path Skew:      -0.131ns
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP falling at 7.500ns
  Clock Uncertainty:    0.000ns

  Data Path: xor_crc/xor_func/out_xor_30 to refout/tmp_30
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  ------------------
    Tcko                  0.449   xor_crc/xor_func/out_xor_30
    net (fanout=2)        1.002   xor_crc/xor_func/out_xor<30>
    Tif5x                 0.692   refout/_n005811_F
                                  refout/_n005811
    net (fanout=1)        1.242   refout/_n0058
    Tioock                0.307   refout/tmp_30
    ----------------------------  ------------------------
    Total                 3.692ns (1.448ns logic, 2.244ns route)
                                  (39.2% logic, 60.8% route)

--------------------------------------------------------------------------------
Slack:                  3.715ns (requirement - (data path - clock path skew +
uncertainty))
  Source:               xor_crc/xor_func/out_xor_30 (FF)
  Destination:          refout/tmp_1 (FF)
  Requirement:          7.500ns
  Data Path Delay:      3.654ns (Levels of Logic = 1)
  Clock Path Skew:      -0.131ns
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP falling at 7.500ns
  Clock Uncertainty:    0.000ns

  Data Path: xor_crc/xor_func/out_xor_30 to refout/tmp_1
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  ------------------
    Tcko                  0.449   xor_crc/xor_func/out_xor_30
    net (fanout=2)        0.887   xor_crc/xor_func/out_xor<30>
    Tif5x                 0.692   refout/_n003311_G
                                  refout/_n003311
    net (fanout=1)        1.319   refout/_n0033
    Tioock                0.307   refout/tmp_1
    ----------------------------  ------------------------
    Total                 3.654ns (1.448ns logic, 2.206ns route)
                                  (39.6% logic, 60.4% route)

--------------------------------------------------------------------------------
Slack:                  3.773ns (requirement - (data path - clock path skew +
uncertainty))
  Source:               xor_crc/C_5 (FF)
  Destination:          refout/tmp_5 (FF)
  Requirement:          7.500ns
  Data Path Delay:      3.610ns (Levels of Logic = 1)
  Clock Path Skew:      -0.117ns
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP falling at 7.500ns
  Clock Uncertainty:    0.000ns

  Data Path: xor_crc/C_5 to refout/tmp_5
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  ------------------
    Tcko                  0.449   xor_crc/C_5
    net (fanout=4)        0.654   xor_crc/C<5>
```

```
    Tif5x                  0.692   refout/_n003711_F
                                   refout/_n003711
    net (fanout=1)         1.508   refout/_n0037
    Tioock                 0.307   refout/tmp_5
    ----------------------------  ----------------------------
    Total                  3.610ns (1.448ns logic, 2.162ns route)
                                   (40.1% logic, 59.9% route)

--------------------------------------------------------------------------------

==============================================================================
Timing constraint: OFFSET = IN 10 ns BEFORE COMP "clk";

 187 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum allowable offset is   1.945ns.
--------------------------------------------------------------------------------
Slack:                 8.055ns (requirement - (data path - clock path - clock
arrival + uncertainty))
  Source:              rst (PAD)
  Destination:         xor_crc/xor_func/out_xor_22 (FF)
  Destination Clock:   clk_BUFGP rising at 0.000ns
  Requirement:         10.000ns
  Data Path Delay:     3.717ns (Levels of Logic = 1)
  Clock Path Delay:    1.772ns (Levels of Logic = 2)
  Clock Uncertainty:   0.000ns

  Data Path: rst to xor_crc/xor_func/out_xor_22
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
    Tiopi                0.653   rst
                                 rst_IBUF
    net (fanout=81)      2.874   rst_IBUF
    Tceck                0.190   xor_crc/xor_func/out_xor_22
    ----------------------------  ----------------------------
    Total                3.717ns (0.843ns logic, 2.874ns route)
                                 (22.7% logic, 77.3% route)


  Clock Path: clk to xor_crc/xor_func/out_xor_22
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
    Tiopi                0.653   clk
                                 clk_BUFGP/IBUFG
    net (fanout=1)       0.143   clk_BUFGP/IBUFG
    Tgi0o                0.465   clk_BUFGP/BUFG
    net (fanout=81)      0.511   clk_BUFGP
    ----------------------------  ----------------------------
    Total                1.772ns (1.118ns logic, 0.654ns route)
                                 (63.1% logic, 36.9% route)

--------------------------------------------------------------------------------
Slack:                 8.055ns (requirement - (data path - clock path - clock
arrival + uncertainty))
  Source:              rst (PAD)
  Destination:         xor_crc/xor_func/out_xor_23 (FF)
  Destination Clock:   clk_BUFGP rising at 0.000ns
  Requirement:         10.000ns
  Data Path Delay:     3.717ns (Levels of Logic = 1)
  Clock Path Delay:    1.772ns (Levels of Logic = 2)
  Clock Uncertainty:   0.000ns

  Data Path: rst to xor_crc/xor_func/out_xor_23
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
    Tiopi                0.653   rst
                                 rst_IBUF
    net (fanout=81)      2.874   rst_IBUF
    Tceck                0.190   xor_crc/xor_func/out_xor_23
    ----------------------------  ----------------------------
    Total                3.717ns (0.843ns logic, 2.874ns route)
                                 (22.7% logic, 77.3% route)

  Clock Path: clk to xor_crc/xor_func/out_xor_23
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
```

```
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.511   clk_BUFGP
    --------------------------  -------------------------
    Total                 1.772ns (1.118ns logic, 0.654ns route)
                                  (63.1% logic, 36.9% route)

--------------------------------------------------------------------------------
Slack:                  8.057ns (requirement - (data path - clock path - clock
arrival + uncertainty))
  Source:               rst (PAD)
  Destination:          xor_crc/xor_func/out_xor_29 (FF)
  Destination Clock:    clk_BUFGP rising at 0.000ns
  Requirement:          10.000ns
  Data Path Delay:      3.724ns (Levels of Logic = 1)
  Clock Path Delay:     1.781ns (Levels of Logic = 2)
  Clock Uncertainty:    0.000ns

  Data Path: rst to xor_crc/xor_func/out_xor_29
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiopi                 0.653   rst
                                  rst_IBUF
    net (fanout=81)       2.881   rst_IBUF
    Tceck                 0.190   xor_crc/xor_func/out_xor_29
    -------------------------  -------------------------
    Total                 3.724ns (0.843ns logic, 2.881ns route)
                                  (22.6% logic, 77.4% route)

  Clock Path: clk to xor_crc/xor_func/out_xor_29
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.520   clk_BUFGP
    --------------------------  -------------------------
    Total                 1.781ns (1.118ns logic, 0.663ns route)
                                  (62.8% logic, 37.2% route)

--------------------------------------------------------------------------------
Slack:                  8.057ns (requirement - (data path - clock path - clock
arrival + uncertainty))
  Source:               rst (PAD)
  Destination:          xor_crc/xor_func/out_xor_28 (FF)
  Destination Clock:    clk_BUFGP rising at 0.000ns
  Requirement:          10.000ns
  Data Path Delay:      3.724ns (Levels of Logic = 1)
  Clock Path Delay:     1.781ns (Levels of Logic = 2)
  Clock Uncertainty:    0.000ns

  Data Path: rst to xor_crc/xor_func/out_xor_28
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiopi                 0.653   rst
                                  rst_IBUF
    net (fanout=81)       2.881   rst_IBUF
    Tceck                 0.190   xor_crc/xor_func/out_xor_28
    -------------------------  -------------------------
    Total                 3.724ns (0.843ns logic, 2.881ns route)
                                  (22.6% logic, 77.4% route)

  Clock Path: clk to xor_crc/xor_func/out_xor_28
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.520   clk_BUFGP
    --------------------------  -------------------------
```

```
    Total                 1.781ns (1.118ns logic, 0.663ns route)
                                  (62.8% logic, 37.2% route)

--------------------------------------------------------------------------------
Slack:                  8.061ns (requirement - (data path - clock path - clock
arrival + uncertainty))
  Source:               rst (PAD)
  Destination:          xor_crc/xor_func/out_xor_6 (FF)
  Destination Clock:    clk_BUFGP rising at 0.000ns
  Requirement:          10.000ns
  Data Path Delay:      3.710ns (Levels of Logic = 1)
  Clock Path Delay:     1.771ns (Levels of Logic = 2)
  Clock Uncertainty:    0.000ns

  Data Path: rst to xor_crc/xor_func/out_xor_6
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
    Tiopi                 0.653   rst
                                  rst_IBUF
    net (fanout=81)       2.867   rst_IBUF
    Tceck                 0.190   xor_crc/xor_func/out_xor_6
    ----------------------------  -------------------------
    Total                 3.710ns (0.843ns logic, 2.867ns route)
                                  (22.7% logic, 77.3% route)

  Clock Path: clk to xor_crc/xor_func/out_xor_6
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.510   clk_BUFGP
    ----------------------------  -------------------------
    Total                 1.771ns (1.118ns logic, 0.653ns route)
                                  (63.1% logic, 36.9% route)

--------------------------------------------------------------------------------

==============================================================================
Timing constraint: OFFSET = OUT 20 ns AFTER COMP "clk";

 33 items analyzed, 0 timing errors detected.
 Minimum allowable offset is  13.222ns.
--------------------------------------------------------------------------------
Slack:                  1.778ns (requirement - (clock arrival + clock path + data
path + uncertainty))
  Source:               refout/tmp_26 (FF)
  Destination:          data_out<26> (PAD)
  Source Clock:         clk_BUFGP falling at 7.500ns
  Requirement:          15.000ns
  Data Path Delay:      4.067ns (Levels of Logic = 0)
  Clock Path Delay:     1.655ns (Levels of Logic = 2)
  Clock Uncertainty:    0.000ns

  Clock Path: clk to refout/tmp_26
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.394   clk_BUFGP
    ----------------------------  -------------------------
    Total                 1.655ns (1.118ns logic, 0.537ns route)
                                  (67.6% logic, 32.4% route)

  Data Path: refout/tmp_26 to data_out<26>
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------  -------------------
    Tiockp                4.067   refout/tmp_26
                                  data_out_26_OBUF
                                  data_out<26>
    ----------------------------  -------------------------
```

```
    Total                     4.067ns (4.067ns logic, 0.000ns route)
                                      (100.0% logic, 0.0% route)

--------------------------------------------------------------------------------
Slack:                    1.778ns (requirement - (clock arrival + clock path + data
path + uncertainty))
  Source:                 refout/tmp_4 (FF)
  Destination:            data_out<4> (PAD)
  Source Clock:           clk_BUFGP falling at 7.500ns
  Requirement:            15.000ns
  Data Path Delay:        4.067ns (Levels of Logic = 0)
  Clock Path Delay:       1.655ns (Levels of Logic = 2)
  Clock Uncertainty:      0.000ns

  Clock Path: clk to refout/tmp_4
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.394   clk_BUFGP
    -------------------------  -------------------------
    Total                 1.655ns (1.118ns logic, 0.537ns route)
                                  (67.6% logic, 32.4% route)

  Data Path: refout/tmp_4 to data_out<4>
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiockp                4.067   refout/tmp_4
                                  data_out_4_OBUF
                                  data_out<4>
    -------------------------  -------------------------
    Total                 4.067ns (4.067ns logic, 0.000ns route)
                                  (100.0% logic, 0.0% route)

--------------------------------------------------------------------------------
Slack:                    1.778ns (requirement - (clock arrival + clock path + data
path + uncertainty))
  Source:                 refout/tmp_3 (FF)
  Destination:            data_out<3> (PAD)
  Source Clock:           clk_BUFGP falling at 7.500ns
  Requirement:            15.000ns
  Data Path Delay:        4.067ns (Levels of Logic = 0)
  Clock Path Delay:       1.655ns (Levels of Logic = 2)
  Clock Uncertainty:      0.000ns

  Clock Path: clk to refout/tmp_3
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.394   clk_BUFGP
    -------------------------  -------------------------
    Total                 1.655ns (1.118ns logic, 0.537ns route)
                                  (67.6% logic, 32.4% route)

  Data Path: refout/tmp_3 to data_out<3>
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  -------------------
    Tiockp                4.067   refout/tmp_3
                                  data_out_3_OBUF
                                  data_out<3>
    -------------------------  -------------------------
    Total                 4.067ns (4.067ns logic, 0.000ns route)
                                  (100.0% logic, 0.0% route)

--------------------------------------------------------------------------------
Slack:                    1.778ns (requirement - (clock arrival + clock path + data
path + uncertainty))
  Source:                 refout/tmp_21 (FF)
  Destination:            data_out<21> (PAD)
```

```
  Source Clock:          clk_BUFGP falling at 7.500ns
  Requirement:           15.000ns
  Data Path Delay:       4.067ns (Levels of Logic = 0)
  Clock Path Delay:      1.655ns (Levels of Logic = 2)
  Clock Uncertainty:     0.000ns

  Clock Path: clk to refout/tmp_21
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  --------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.394   clk_BUFGP
    -------------------------  --------------------------
    Total                 1.655ns (1.118ns logic, 0.537ns route)
                                  (67.6% logic, 32.4% route)

  Data Path: refout/tmp_21 to data_out<21>
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  --------------------
    Tiockp                4.067   refout/tmp_21
                                  data_out_21_OBUF
                                  data_out<21>
    -------------------------  --------------------------
    Total                 4.067ns (4.067ns logic, 0.000ns route)
                                  (100.0% logic, 0.0% route)

--------------------------------------------------------------------------------
Slack:                 1.778ns (requirement - (clock arrival + clock path + data
path + uncertainty))
  Source:              refout/tmp_17 (FF)
  Destination:         data_out<17> (PAD)
  Source Clock:        clk_BUFGP falling at 7.500ns
  Requirement:         15.000ns
  Data Path Delay:     4.067ns (Levels of Logic = 0)
  Clock Path Delay:    1.655ns (Levels of Logic = 2)
  Clock Uncertainty:   0.000ns

  Clock Path: clk to refout/tmp_17
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  --------------------
    Tiopi                 0.653   clk
                                  clk_BUFGP/IBUFG
    net (fanout=1)        0.143   clk_BUFGP/IBUFG
    Tgi0o                 0.465   clk_BUFGP/BUFG
    net (fanout=81)       0.394   clk_BUFGP
    -------------------------  --------------------------
    Total                 1.655ns (1.118ns logic, 0.537ns route)
                                  (67.6% logic, 32.4% route)

  Data Path: refout/tmp_17 to data_out<17>
    Delay type         Delay(ns)  Logical Resource(s)
    -------------------------  --------------------
    Tiockp                4.067   refout/tmp_17
                                  data_out_17_OBUF
                                  data_out<17>
    -------------------------  --------------------------
    Total                 4.067ns (4.067ns logic, 0.000ns route)
                                  (100.0% logic, 0.0% route)

--------------------------------------------------------------------------------

All constraints were met.


Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
------------+------------+------------+-----------------+--------+
            |  Setup to  |  Hold to   |                 |  Clock |
```

Source	clk (edge)	clk (edge)	Internal Clock(s)	Phase
crc_done	1.186(R)	-0.015(R)	clk_BUFGP	0.000
data_in<0>	-7.017(F)	7.475(F)	clk_BUFGP	7.500
data_in<1>	-7.128(F)	7.505(F)	clk_BUFGP	7.500
data_in<2>	-7.299(F)	7.538(F)	clk_BUFGP	7.500
data_in<3>	-6.780(F)	7.024(F)	clk_BUFGP	7.500
data_in<4>	-6.922(F)	7.160(F)	clk_BUFGP	7.500
data_in<5>	-6.862(F)	7.106(F)	clk_BUFGP	7.500
data_in<6>	-6.726(F)	7.105(F)	clk_BUFGP	7.500
data_in<7>	-7.005(F)	7.444(F)	clk_BUFGP	7.500
data_valid	-0.218(R)	0.464(R)	clk_BUFGP	0.000
ref_dat	-3.790(F)	7.184(F)	clk_BUFGP	7.500
rst	1.945(R)	-0.553(R)	clk_BUFGP	0.000
xor_done	-4.109(F)	6.610(F)	clk_BUFGP	7.500

Clock clk to Pad

Destination	clk (edge) to PAD	Internal Clock(s)	Clock Phase
crc_valid	6.597(R)	clk_BUFGP	0.000
data_out<0>	13.217(F)	clk_BUFGP	7.500
data_out<10>	13.217(F)	clk_BUFGP	7.500
data_out<11>	13.208(F)	clk_BUFGP	7.500
data_out<12>	13.207(F)	clk_BUFGP	7.500
data_out<13>	13.208(F)	clk_BUFGP	7.500
data_out<14>	13.220(F)	clk_BUFGP	7.500
data_out<15>	13.217(F)	clk_BUFGP	7.500
data_out<16>	13.207(F)	clk_BUFGP	7.500
data_out<17>	13.222(F)	clk_BUFGP	7.500
data_out<18>	13.221(F)	clk_BUFGP	7.500
data_out<19>	13.221(F)	clk_BUFGP	7.500
data_out<1>	13.217(F)	clk_BUFGP	7.500
data_out<20>	13.216(F)	clk_BUFGP	7.500
data_out<21>	13.222(F)	clk_BUFGP	7.500
data_out<22>	13.208(F)	clk_BUFGP	7.500
data_out<23>	13.221(F)	clk_BUFGP	7.500
data_out<24>	13.218(F)	clk_BUFGP	7.500
data_out<25>	13.220(F)	clk_BUFGP	7.500
data_out<26>	13.222(F)	clk_BUFGP	7.500
data_out<27>	13.216(F)	clk_BUFGP	7.500
data_out<28>	13.216(F)	clk_BUFGP	7.500
data_out<29>	13.217(F)	clk_BUFGP	7.500
data_out<2>	13.220(F)	clk_BUFGP	7.500
data_out<30>	13.217(F)	clk_BUFGP	7.500
data_out<31>	13.222(F)	clk_BUFGP	7.500
data_out<3>	13.222(F)	clk_BUFGP	7.500
data_out<4>	13.222(F)	clk_BUFGP	7.500
data_out<5>	13.220(F)	clk_BUFGP	7.500
data_out<6>	13.208(F)	clk_BUFGP	7.500
data_out<7>	13.207(F)	clk_BUFGP	7.500
data_out<8>	13.216(F)	clk_BUFGP	7.500
data_out<9>	13.207(F)	clk_BUFGP	7.500

Clock to Setup on destination clock clk

Source Clock	Src:Rise Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
clk	3.382	3.279	3.890	

Timing summary:

Timing errors: 0 Score: 0

Constraints cover 644 paths, 0 nets, and 744 connections

----------Footnotes----------
1) The minimum period statistic assumes all single cycle delays.

Analysis completed Thu Jun 18 10:32:16 2009

Timing Analyzer Settings:

OpenPCF E:\Project VHDL\crc32\crc_modul.pcf
Speed -6
IncludeNets
ExcludeNets
SelectFailingTimingConstraint False
IncludeNoTimingConstraint False
Report normal
MaxPathsPerTimingConstraint 5
ReportFastestPaths False
GenerateDataSheet True
GenerateTimeGroup False
DefineEndpoints ToAll
DefineEndpoints FromAll
OmitUserConstraints False
DropTimingConstraint
SetForce Off
ProratingOptions

Peak Memory Usage: 148 MB

LAMPIRAN 8
DELAY REPORT

```
Release 7.1i - reportgen H.38
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

Thu Jun 18 09:09:41 2009

File: crc_modul.dly

 The 20 worst nets by delay are:
+-----------------+-----------+
| Max Delay       | Netname   |
+-----------------+-----------+
   3.117            rst_IBUF
   2.560            ref_dat_IBUF
   2.180            xor_done_IBUF
   2.114            crc_done_IBUF
   1.508            refout/_n0037
   1.501            present_state_FFd2
   1.465            refin/temp<7>
   1.414            xor_crc/C<30>
   1.408            xor_crc/C<18>
   1.408            refin/temp<6>
   1.375            xor_crc/C<14>
   1.319            refout/_n0033
   1.283            refout/_n0045
   1.283            xor_crc/C<10>
   1.275            xor_crc/C<8>
   1.268            xor_crc/C<5>
   1.251            data_in_6_IBUF
   1.242            refout/_n0058
   1.238            refin/temp<1>
   1.207            xor_crc/C<31>
--------------------------------

----------------------------------------------------------------------------
                                Net Delays
----------------------------------------------------------------------------
N33
   N33.X
         0.002  xor_crc/C<5>.F3

N37
   N37.X
         0.228  xor_crc/C<6>.G1

N39
   N39.X
         0.231  xor_crc/C<13>.G1

N47
   N47.X
         0.364  xor_crc/C<6>.F4

N51
   xor_crc/C<25>.Y
         0.028  xor_crc/C<25>.F4

N53
   xor_crc/C<30>.Y
         0.002  xor_crc/C<30>.F4

N55
   xor_crc/C<19>.Y
         0.226  xor_crc/C<19>.F4

N57
   xor_crc/C<29>.Y
```

```
            0.390   xor_crc/C<29>.F4

N59
   xor_crc/C<28>.Y
            0.390   xor_crc/C<28>.F4

N61
   xor_crc/C<24>.Y
            0.002   xor_crc/C<24>.F4

N63
   xor_crc/C<18>.Y
            0.028   xor_crc/C<18>.F4

N65
   xor_crc/C<23>.Y
            0.002   xor_crc/C<23>.F4

N67
   xor_crc/C<17>.Y
            0.028   xor_crc/C<17>.F4

N69
   xor_crc/C<16>.Y
            0.390   xor_crc/C<16>.F4

N71
   N71.X
            0.265   xor_crc/C<13>.F4

N73
   N39.Y
            0.486   xor_crc/C<3>.F3

N75
   N47.Y
            0.493   xor_crc/C<7>.F4

N77
   N91.Y
            0.424   xor_crc/C<15>.G4

N79
   xor_crc/C<31>.X
            0.033   xor_crc/C<9>.G1

N81
   N81.X
            0.004   xor_crc/C<27>.G1

N83
   N83.X
            0.442   xor_crc/C<27>.F1

N85
   N37.Y
            0.033   xor_crc/C<14>.G1

N87
   N83.Y
            0.367   xor_crc/C<11>.G1

N91
   N91.X
            0.341   xor_crc/C<1>.G1

clk_BUFGP
   clk_BUFGP/BUFG.O
            0.393   data_out<19>.OTCLK1
            0.388   data_out<27>.OTCLK1
            0.380   data_out<6>.OTCLK1
            0.379   data_out<7>.OTCLK1
            0.388   data_out<28>.OTCLK1
            0.388   data_out<8>.OTCLK1
```

```
0.389  data_out<29>.OTCLK1
0.379  data_out<9>.OTCLK1
0.509  xor_crc/C<6>.CLK
0.509  xor_crc/C<7>.CLK
0.511  xor_crc/C<12>.CLK
0.511  xor_crc/C<8>.CLK
0.512  xor_crc/C<4>.CLK
0.511  xor_crc/C<9>.CLK
0.510  refin/temp<1>.CLK
0.512  refin/temp<3>.CLK
0.512  refin/temp<5>.CLK
0.510  refin/temp<7>.CLK
0.510  xor_crc/C<11>.CLK
0.511  xor_crc/C<21>.CLK
0.509  xor_crc/C<13>.CLK
0.509  xor_crc/C<14>.CLK
0.510  xor_crc/C<15>.CLK
0.511  xor_crc/C<31>.CLK
0.510  xor_crc/C<27>.CLK
0.511  xor_crc/C<3>.CLK
0.509  xor_crc/C<5>.CLK
0.510  xor_crc/C<1>.CLK
0.510  xor_crc/C<2>.CLK
0.520  present_state_FFd2.CLK
0.511  xor_crc/C<22>.CLK
0.512  xor_crc/C<17>.CLK
0.520  xor_crc/xor_func/out_xor<11>.CLK
0.511  xor_crc/C<25>.CLK
0.520  xor_crc/xor_func/out_xor<21>.CLK
0.519  xor_crc/xor_func/out_xor<13>.CLK
0.511  xor_crc/xor_func/out_xor<23>.CLK
0.512  xor_crc/xor_func/out_xor<15>.CLK
0.520  xor_crc/xor_func/out_xor<31>.CLK
0.511  xor_crc/xor_func/out_xor<25>.CLK
0.512  xor_crc/xor_func/out_xor<17>.CLK
0.511  xor_crc/xor_func/out_xor<27>.CLK
0.519  xor_crc/xor_func/out_xor<19>.CLK
0.511  xor_crc/C<16>.CLK
0.511  xor_crc/C<24>.CLK
0.520  xor_crc/xor_func/out_xor<29>.CLK
0.510  xor_crc/C<19>.CLK
0.511  xor_crc/C<18>.CLK
0.512  xor_crc/C<29>.CLK
0.511  xor_crc/C<28>.CLK
0.511  xor_crc/C<30>.CLK
0.511  xor_crc/xor_func/out_xor<1>.CLK
0.520  xor_crc/xor_func/out_xor<3>.CLK
0.512  xor_crc/xor_func/out_xor<5>.CLK
0.510  xor_crc/xor_func/out_xor<7>.CLK
0.512  xor_crc/xor_func/out_xor<9>.CLK
0.512  xor_crc/C<23>.CLK
0.389  data_out<10>.OTCLK1
0.380  data_out<11>.OTCLK1
0.388  data_out<20>.OTCLK1
0.379  data_out<12>.OTCLK1
0.389  data_out<0>.OTCLK1
0.380  data_out<13>.OTCLK1
0.394  data_out<21>.OTCLK1
0.389  data_out<1>.OTCLK1
0.389  data_out<30>.OTCLK1
0.380  data_out<22>.OTCLK1
0.392  data_out<14>.OTCLK1
0.392  data_out<2>.OTCLK1
0.389  data_out<15>.OTCLK1
0.393  data_out<23>.OTCLK1
0.394  data_out<31>.OTCLK1
0.394  data_out<3>.OTCLK1
0.390  data_out<24>.OTCLK1
0.379  data_out<16>.OTCLK1
0.394  data_out<17>.OTCLK1
0.392  data_out<25>.OTCLK1
0.394  data_out<4>.OTCLK1
0.392  data_out<5>.OTCLK1
0.394  data_out<26>.OTCLK1
```

```
            0.393  data_out<18>.OTCLK1

clk_BUFGP/IBUFG
   clk.I
            0.143  clk_BUFGP/BUFG.I0

crc_done_IBUF
   crc_done.I
            1.620  xor_crc/C<6>.CE
            1.620  xor_crc/C<7>.CE
            1.621  xor_crc/C<12>.CE
            1.621  xor_crc/C<8>.CE
            1.882  xor_crc/C<4>.CE
            2.114  xor_crc/C<9>.CE
            2.114  xor_crc/C<11>.CE
            1.637  xor_crc/C<21>.CE
            1.620  xor_crc/C<13>.CE
            1.620  xor_crc/C<14>.CE
            2.109  xor_crc/C<15>.CE
            2.114  xor_crc/C<31>.CE
            2.114  xor_crc/C<27>.CE
            1.637  xor_crc/C<3>.CE
            1.620  xor_crc/C<5>.CE
            1.143  xor_crc/C<1>.CE
            1.143  xor_crc/C<2>.CE
            0.717  present_state_FFd2.G1
            1.615  xor_crc/C<22>.CE
            1.615  xor_crc/C<17>.CE
            1.637  xor_crc/C<25>.CE
            2.109  xor_crc/C<16>.CE
            1.637  xor_crc/C<24>.CE
            2.109  xor_crc/C<19>.CE
            1.122  xor_crc/C<18>.CE
            1.621  xor_crc/C<29>.CE
            1.620  xor_crc/C<28>.CE
            1.620  xor_crc/C<30>.CE
            1.622  xor_crc/C<23>.CE

data_in_0_IBUF
   data_in<0>.I
            0.717  refin/temp<1>.G1
            0.960  refin/temp<7>.F4

data_in_1_IBUF
   data_in<1>.I
            0.849  refin/temp<1>.F1
            0.687  refin/temp<7>.G4

data_in_2_IBUF
   data_in<2>.I
            0.656  refin/temp<3>.G1
            0.680  refin/temp<5>.F4

data_in_3_IBUF
   data_in<3>.I
            1.170  refin/temp<3>.F1
            1.199  refin/temp<5>.G4

data_in_4_IBUF
   data_in<4>.I
            1.057  refin/temp<3>.F4
            1.034  refin/temp<5>.G1

data_in_5_IBUF
   data_in<5>.I
            1.117  refin/temp<3>.G4
            1.088  refin/temp<5>.F1

data_in_6_IBUF
   data_in<6>.I
            1.251  refin/temp<1>.F4
            1.087  refin/temp<7>.G1

data_in_7_IBUF
```

```
    data_in<7>.I
          0.972  refin/temp<1>.G4
          0.748  refin/temp<7>.F1

data_valid_IBUF
   data_valid.I
          0.238  present_state_FFd2.F3
          0.269  present_state_FFd2.G4

present_state_FFd1
   present_state_FFd2.YQ
          0.624  crc_valid.O1
          0.391  present_state_FFd2.F1
          0.368  present_state_FFd2.G2

present_state_FFd1-In
   present_state_FFd2.Y
          0.001  present_state_FFd2.DY

present_state_FFd2
   present_state_FFd2.XQ
          0.753  refin/temp<1>.F2
          1.132  refin/temp<1>.G2
          1.501  refin/temp<3>.F2
          1.268  refin/temp<3>.G2
          1.237  refin/temp<5>.F2
          1.268  refin/temp<5>.G2
          0.948  refin/temp<7>.F2
          0.977  refin/temp<7>.G2
          0.422  present_state_FFd2.F2
          0.451  present_state_FFd2.G3

present_state_FFd2-In
   present_state_FFd2.X
          0.001  present_state_FFd2.DX

ref_dat_IBUF
   ref_dat.I
          1.006  refout/_n0050.BX
          2.066  refout/_n0034.BX
          1.492  refout/_n0042.BX
          2.548  refout/_n0051.BX
          2.295  refout/_n0035.BX
          1.254  refout/_n0043.BX
          1.796  refout/_n0052.BX
          1.764  refout/_n0036.BX
          2.295  refout/_n0044.BX
          2.548  refout/_n0060.BX
          2.560  refout/_n0053.BX
          1.535  refout/_n0061.BX
          1.769  refout/_n0037.BX
          2.555  refout/_n0045.BX
          2.273  refout/_n0054.BX
          2.024  refout/_n0062.BX
          2.273  refout/_n0038.BX
          2.548  refout/_n0046.BX
          2.555  refout/_n0055.BX
          2.039  refout/_n0047.BX
          2.066  refout/_n0063.BX
          2.277  refout/_n0039.BX
          2.277  refout/_n0056.BX
          1.254  refout/_n0048.BX
          2.560  refout/_n0057.BX
          2.039  refout/_n0049.BX
          1.764  refout/_n0058.BX
          2.250  refout/_n0059.BX
          2.548  refout/_n0032.BX
          1.806  refout/_n0040.BX
          1.764  refout/_n0033.BX
          1.783  refout/_n0041.BX
          1.511  refin/temp<1>.F3
          1.487  refin/temp<1>.G3
          1.246  refin/temp<3>.F3
          1.635  refin/temp<3>.G3
```

```
        1.658  refin/temp<5>.F3
        1.010  refin/temp<5>.G3
        1.649  refin/temp<7>.F3
        1.626  refin/temp<7>.G3

refin/_n00081/O
   refin/temp<1>.Y
        0.001  refin/temp<1>.DY

refin/_n00091/O
   refin/temp<1>.X
        0.001  refin/temp<1>.DX

refin/_n00101/O
   refin/temp<3>.Y
        0.001  refin/temp<3>.DY

refin/_n00111/O
   refin/temp<3>.X
        0.001  refin/temp<3>.DX

refin/_n00121/O
   refin/temp<5>.Y
        0.001  refin/temp<5>.DY

refin/_n00131/O
   refin/temp<5>.X
        0.001  refin/temp<5>.DX

refin/_n00141/O
   refin/temp<7>.Y
        0.001  refin/temp<7>.DY

refin/_n00151/O
   refin/temp<7>.X
        0.001  refin/temp<7>.DX

refin/temp<0>
   refin/temp<1>.YQ
        0.919  xor_crc/C<7>.G4
        0.929  xor_crc/C<12>.G4
        0.657  xor_crc/C<1>.F4
        0.686  xor_crc/C<22>.G3
        0.423  xor_crc/C<16>.F3
        0.969  xor_crc/C<23>.F3

refin/temp<1>
   refin/temp<1>.XQ
        0.866  xor_crc/C<6>.G2
        0.493  xor_crc/C<8>.G4
        1.132  N33.F2
        0.632  xor_crc/C<3>.G2
        1.238  N71.F3
        1.201  xor_crc/C<1>.G2
        0.632  xor_crc/C<2>.G3
        0.678  xor_crc/C<17>.F3
        0.921  xor_crc/C<24>.F3
        0.678  xor_crc/C<23>.F2

refin/temp<2>
   refin/temp<3>.YQ
        0.265  xor_crc/C<4>.G3
        0.437  xor_crc/C<9>.F4
        0.466  xor_crc/C<9>.G2
        0.716  xor_crc/C<11>.G2
        0.933  xor_crc/C<13>.G4
        0.878  xor_crc/C<2>.F1
        0.740  N39.G3
        0.491  xor_crc/C<25>.F2
        0.973  N47.F3
        0.949  N47.G3
        0.755  xor_crc/C<24>.F2
        0.467  xor_crc/C<18>.F3
        0.242  xor_crc/C<28>.F3
```

```
refin/temp<3>
   refin/temp<3>.XQ
         0.760  xor_crc/C<7>.G3
         0.369  xor_crc/C<4>.F1
         0.784  N33.F4
         0.656  xor_crc/C<13>.G2
         0.760  xor_crc/C<15>.G3
         0.656  xor_crc/C<27>.G2
         0.684  xor_crc/C<3>.F1
         0.689  xor_crc/C<25>.F1
         0.922  xor_crc/C<19>.F2
         0.268  xor_crc/C<29>.F3

refin/temp<4>
   refin/temp<5>.YQ
         0.960  xor_crc/C<6>.G4
         0.533  xor_crc/C<8>.G3
         0.290  xor_crc/C<4>.F2
         0.942  N33.F3
         0.266  xor_crc/C<21>.G3
         0.981  xor_crc/C<14>.G2
         1.144  xor_crc/C<15>.F3
         0.767  xor_crc/C<27>.G4
         0.474  xor_crc/C<16>.F2
         0.670  xor_crc/C<30>.F2

refin/temp<5>
   refin/temp<5>.XQ
         1.164  xor_crc/C<6>.F2
         1.138  xor_crc/C<7>.F1
         0.427  xor_crc/C<9>.F3
         0.746  xor_crc/C<9>.G4
         0.959  xor_crc/C<11>.G4
         0.523  xor_crc/C<21>.F3
         0.895  xor_crc/C<13>.F2
         1.184  xor_crc/C<15>.F2
         0.265  xor_crc/C<31>.G3
         0.735  xor_crc/C<27>.F2
         0.976  xor_crc/C<5>.G3
         1.148  xor_crc/C<17>.F2
         0.533  xor_crc/C<16>.F1
         0.289  xor_crc/C<28>.F2

refin/temp<6>
   refin/temp<7>.YQ
         0.751  xor_crc/C<6>.F1
         0.761  xor_crc/C<12>.G3
         0.963  xor_crc/C<13>.F1
         1.187  xor_crc/C<14>.G4
         0.977  xor_crc/C<5>.F1
         0.730  xor_crc/C<17>.F1
         0.906  xor_crc/C<18>.F2
         1.408  xor_crc/C<29>.F2
         0.452  xor_crc/C<28>.F1
         0.656  xor_crc/C<23>.F1

refin/temp<7>
   refin/temp<7>.XQ
         0.771  xor_crc/C<6>.G3
         1.427  xor_crc/C<7>.F2
         0.980  xor_crc/C<13>.G3
         1.253  xor_crc/C<15>.F1
         1.004  xor_crc/C<27>.F4
         1.144  xor_crc/C<3>.F2
         1.465  xor_crc/C<1>.F3
         0.703  xor_crc/C<1>.G4
         1.004  xor_crc/C<2>.F2
         0.875  xor_crc/C<24>.F1
         0.658  xor_crc/C<19>.F1
         1.180  xor_crc/C<18>.F1
         0.737  xor_crc/C<29>.F1
         0.770  xor_crc/C<30>.F1
refout/_n0032
```

```
   refout/_n0032.X
         0.621  data_out<0>.O1

refout/_n0033
   refout/_n0033.X
         1.319  data_out<1>.O1

refout/_n0034
   refout/_n0034.X
         0.867  data_out<2>.O1

refout/_n0035
   refout/_n0035.X
         0.876  data_out<3>.O1

refout/_n0036
   refout/_n0036.X
         0.640  data_out<4>.O1

refout/_n0037
   refout/_n0037.X
         1.508  data_out<5>.O1

refout/_n0038
   refout/_n0038.X
         0.621  data_out<6>.O1

refout/_n0039
   refout/_n0039.X
         0.941  data_out<7>.O1

refout/_n0040
   refout/_n0040.X
         1.140  data_out<8>.O1

refout/_n0041
   refout/_n0041.X
         0.829  data_out<9>.O1

refout/_n0042
   refout/_n0042.X
         1.102  data_out<10>.O1

refout/_n0043
   refout/_n0043.X
         0.897  data_out<11>.O1

refout/_n0044
   refout/_n0044.X
         1.199  data_out<12>.O1

refout/_n0045
   refout/_n0045.X
         1.283  data_out<13>.O1

refout/_n0046
   refout/_n0046.X
         0.869  data_out<14>.O1

refout/_n0047
   refout/_n0047.X
         1.186  data_out<15>.O1

refout/_n0048
   refout/_n0048.X
         0.859  data_out<20>.O1

refout/_n0049
   refout/_n0049.X
         0.874  data_out<16>.O1

refout/_n0050
   refout/_n0050.X
         0.621  data_out<21>.O1
```

```
refout/_n0051
   refout/_n0051.X
         0.375  data_out<17>.O1

refout/_n0052
   refout/_n0052.X
         1.088  data_out<22>.O1

refout/_n0053
   refout/_n0053.X
         0.640  data_out<18>.O1

refout/_n0054
   refout/_n0054.X
         1.113  data_out<23>.O1

refout/_n0055
   refout/_n0055.X
         0.876  data_out<19>.O1

refout/_n0056
   refout/_n0056.X
         1.113  data_out<24>.O1

refout/_n0057
   refout/_n0057.X
         0.870  data_out<25>.O1

refout/_n0058
   refout/_n0058.X
         1.242  data_out<30>.O1

refout/_n0059
   refout/_n0059.X
         0.883  data_out<26>.O1

refout/_n0060
   refout/_n0060.X
         0.375  data_out<31>.O1

refout/_n0061
   refout/_n0061.X
         0.871  data_out<27>.O1

refout/_n0062
   refout/_n0062.X
         1.070  data_out<28>.O1

refout/_n0063
   refout/_n0063.X
         1.088  data_out<29>.O1

rst_IBUF
   rst.I
         2.385  data_out<19>.SR
         1.167  data_out<27>.SR
         1.667  data_out<6>.SR
         1.421  data_out<7>.SR
         1.882  data_out<28>.SR
         1.882  data_out<8>.SR
         2.153  data_out<29>.SR
         2.369  data_out<9>.SR
         2.336  xor_crc/C<6>.SR
         2.336  xor_crc/C<7>.SR
         2.113  xor_crc/C<12>.SR
         2.387  xor_crc/C<8>.SR
         2.367  xor_crc/C<4>.SR
         2.101  xor_crc/C<9>.SR
         2.137  refin/temp<1>.SR
         2.614  refin/temp<3>.SR
         2.614  refin/temp<5>.SR
         1.868  refin/temp<7>.SR
         2.863  xor_crc/C<11>.SR
```

```
        2.387  xor_crc/C<21>.SR
        2.336  xor_crc/C<13>.SR
        2.599  xor_crc/C<14>.SR
        2.858  xor_crc/C<15>.SR
        2.614  xor_crc/C<31>.SR
        2.589  xor_crc/C<27>.SR
        2.113  xor_crc/C<3>.SR
        2.599  xor_crc/C<5>.SR
        2.584  xor_crc/C<1>.SR
        2.589  xor_crc/C<2>.SR
        2.596  present_state_FFd2.SR
        2.113  xor_crc/C<22>.SR
        2.636  xor_crc/C<17>.SR
        1.614  xor_crc/xor_func/out_xor<11>.CE
        2.113  xor_crc/C<25>.SR
        1.864  xor_crc/xor_func/out_xor<21>.CE
        2.874  xor_crc/xor_func/out_xor<13>.CE
        2.874  xor_crc/xor_func/out_xor<23>.CE
        2.865  xor_crc/xor_func/out_xor<15>.CE
        2.834  xor_crc/xor_func/out_xor<31>.CE
        2.631  xor_crc/xor_func/out_xor<25>.CE
        1.874  xor_crc/xor_func/out_xor<17>.CE
        1.631  xor_crc/xor_func/out_xor<27>.CE
        2.363  xor_crc/xor_func/out_xor<19>.CE
        2.101  xor_crc/C<16>.SR
        2.387  xor_crc/C<24>.SR
        2.881  xor_crc/xor_func/out_xor<29>.CE
        2.584  xor_crc/C<19>.SR
        2.863  xor_crc/C<18>.SR
        2.357  xor_crc/C<29>.SR
        2.614  xor_crc/C<28>.SR
        2.101  xor_crc/C<30>.SR
        2.346  xor_crc/xor_func/out_xor<1>.CE
        2.834  xor_crc/xor_func/out_xor<3>.CE
        1.859  xor_crc/xor_func/out_xor<5>.CE
        2.867  xor_crc/xor_func/out_xor<7>.CE
        1.874  xor_crc/xor_func/out_xor<9>.CE
        2.636  xor_crc/C<23>.SR
        1.245  data_out<10>.SR
        2.612  data_out<11>.SR
        1.167  data_out<20>.SR
        2.369  data_out<12>.SR
        2.153  data_out<0>.SR
        2.612  data_out<13>.SR
        1.576  data_out<21>.SR
        1.245  data_out<1>.SR
        1.245  data_out<30>.SR
        1.667  data_out<22>.SR
        2.359  data_out<14>.SR
        2.359  data_out<2>.SR
        1.882  data_out<15>.SR
        2.834  data_out<23>.SR
        2.600  data_out<31>.SR
        2.600  data_out<3>.SR
        3.117  data_out<24>.SR
        2.369  data_out<16>.SR
        2.834  data_out<17>.SR
        2.823  data_out<25>.SR
        1.850  data_out<4>.SR
        2.823  data_out<5>.SR
        1.850  data_out<26>.SR
        2.385  data_out<18>.SR

xor_crc/C<0>
  xor_crc/C<12>.YQ
        0.950  refout/_n0060.F1
        0.979  refout/_n0032.G1
        0.391  xor_crc/C<8>.F1
        1.045  xor_crc/xor_func/out_xor<1>.BY

xor_crc/C<10>
  xor_crc/C<11>.YQ
        0.769  refout/_n0050.F1
        0.884  refout/_n0042.G1
```

```
        1.283  xor_crc/xor_func/out_xor<11>.BY
        0.229  xor_crc/C<18>.G4

xor_crc/C<11>
   xor_crc/C<11>.XQ
        0.763  refout/_n0043.G1
        0.927  refout/_n0048.F1
        1.011  xor_crc/xor_func/out_xor<11>.BX
        0.251  xor_crc/C<19>.G2

xor_crc/C<12>
   xor_crc/C<12>.XQ
        0.753  refout/_n0044.G1
        0.915  refout/_n0055.F1
        0.258  xor_crc/C<21>.G1
        1.048  xor_crc/xor_func/out_xor<13>.BY

xor_crc/C<13>
   xor_crc/C<13>.XQ
        0.663  refout/_n0053.F1
        0.501  refout/_n0045.G1
        0.482  xor_crc/C<21>.F1
        1.028  xor_crc/xor_func/out_xor<13>.BX

xor_crc/C<14>
   xor_crc/C<14>.YQ
        1.097  refout/_n0051.F1
        1.073  refout/_n0046.G1
        0.500  xor_crc/C<22>.G1
        1.375  xor_crc/xor_func/out_xor<15>.BY

xor_crc/C<15>
   xor_crc/C<15>.XQ
        0.630  refout/_n0047.G1
        0.515  refout/_n0049.F1
        0.778  xor_crc/xor_func/out_xor<15>.BX
        0.397  xor_crc/C<23>.G4

xor_crc/C<16>
   xor_crc/C<16>.XQ
        0.458  refout/_n0047.F1
        0.294  refout/_n0049.G1
        0.555  xor_crc/xor_func/out_xor<17>.BY
        0.345  xor_crc/C<24>.G4

xor_crc/C<17>
   xor_crc/C<17>.XQ
        0.509  refout/_n0051.G1
        0.478  refout/_n0046.F1
        0.235  xor_crc/C<25>.G2
        0.746  xor_crc/xor_func/out_xor<17>.BX

xor_crc/C<18>
   xor_crc/C<18>.XQ
        0.640  refout/_n0053.G1
        0.525  refout/_n0045.F1
        1.408  xor_crc/xor_func/out_xor<19>.BY
        0.282  N81.F3

xor_crc/C<19>
   xor_crc/C<19>.XQ
        1.112  refout/_n0044.F1
        0.950  refout/_n0055.G1
        0.989  xor_crc/xor_func/out_xor<19>.BX
        0.226  N83.F3

xor_crc/C<1>
   xor_crc/C<1>.YQ
        0.735  refout/_n0058.F1
        0.959  refout/_n0033.G1
        0.661  xor_crc/C<31>.F3
        0.543  xor_crc/xor_func/out_xor<1>.BX

xor_crc/C<20>
```

```
   xor_crc/C<21>.YQ
         0.528  refout/_n0043.F1
         0.643  refout/_n0048.G1
         1.033  xor_crc/xor_func/out_xor<21>.BY
         0.497  xor_crc/C<28>.G4

xor_crc/C<21>
   xor_crc/C<21>.XQ
         0.860  refout/_n0050.G1
         0.745  refout/_n0042.F1
         1.002  xor_crc/xor_func/out_xor<21>.BX
         0.367  xor_crc/C<29>.G4

xor_crc/C<22>
   xor_crc/C<22>.YQ
         0.346  refout/_n0052.G1
         0.231  refout/_n0041.F1
         0.557  xor_crc/xor_func/out_xor<23>.BY
         0.268  xor_crc/C<30>.G2

xor_crc/C<23>
   xor_crc/C<23>.XQ
         0.235  refout/_n0054.G1
         0.399  refout/_n0040.F1
         0.656  xor_crc/C<31>.G1
         0.551  xor_crc/xor_func/out_xor<23>.BX

xor_crc/C<24>
   xor_crc/C<24>.XQ
         0.543  refout/_n0039.F1
         0.519  refout/_n0056.G1
         0.658  xor_crc/C<7>.G1
         0.385  xor_crc/C<12>.G1
         0.448  xor_crc/C<1>.F1
         0.385  xor_crc/C<22>.G2
         0.576  xor_crc/xor_func/out_xor<25>.BY
         0.239  xor_crc/C<16>.G4
         0.484  xor_crc/C<23>.G3

xor_crc/C<25>
   xor_crc/C<25>.XQ
         0.280  refout/_n0038.F1
         0.478  refout/_n0057.G1
         0.699  xor_crc/C<8>.G1
         0.689  N33.G3
         0.234  xor_crc/C<3>.G1
         0.739  N71.G2
         0.467  xor_crc/C<2>.G1
         0.455  xor_crc/C<17>.G4
         0.708  N37.F3
         0.533  xor_crc/xor_func/out_xor<25>.BX
         0.455  xor_crc/C<24>.G3
         0.465  N91.F2
         0.455  xor_crc/C<23>.G2

xor_crc/C<26>
   xor_crc/C<27>.YQ
         0.479  refout/_n0037.F1
         0.315  refout/_n0059.G1
         0.658  xor_crc/C<4>.G4
         0.530  xor_crc/C<9>.F1
         0.725  xor_crc/C<31>.F2
         0.290  xor_crc/C<2>.G2
         0.482  N39.F3
         1.187  N39.G2
         0.682  xor_crc/C<25>.G1
         0.889  N47.F2
         0.866  N47.G1
         0.839  xor_crc/xor_func/out_xor<27>.BY
         0.543  xor_crc/C<24>.G2
         0.483  N83.G2
         0.315  xor_crc/C<18>.G3
         0.754  xor_crc/C<28>.G3
```

```
xor_crc/C<27>
   xor_crc/C<27>.XQ
         0.910  refout/_n0036.F1
         0.955  refout/_n0061.G1
         0.540  xor_crc/C<7>.G2
         0.955  xor_crc/C<4>.G1
         0.412  N33.G2
         0.530  N39.F2
         0.506  N39.G1
         1.154  xor_crc/C<25>.F3
         0.753  xor_crc/xor_func/out_xor<27>.BX
         0.455  xor_crc/C<19>.G1
         0.733  N91.G3
         0.395  N81.F2
         0.939  xor_crc/C<29>.G3

xor_crc/C<28>
   xor_crc/C<28>.XQ
         0.881  refout/_n0035.F1
         0.996  refout/_n0062.G1
         0.378  xor_crc/C<8>.G2
         0.485  xor_crc/C<4>.G2
         0.551  N33.G1
         0.378  xor_crc/C<21>.G2
         0.744  xor_crc/C<15>.G1
         0.958  N37.F2
         0.987  N37.G3
         0.289  xor_crc/C<16>.G3
         1.147  xor_crc/xor_func/out_xor<29>.BY
         0.470  N81.F1
         0.276  xor_crc/C<30>.G1

xor_crc/C<29>
   xor_crc/C<29>.XQ
         0.922  refout/_n0034.F1
         0.898  refout/_n0063.G1
         0.301  xor_crc/C<9>.F2
         0.421  xor_crc/C<21>.F2
         0.890  xor_crc/C<15>.G2
         0.513  xor_crc/C<31>.F1
         0.737  xor_crc/C<31>.G2
         0.968  xor_crc/C<5>.G2
         0.486  N71.G1
         0.642  xor_crc/C<17>.G3
         0.970  N47.F4
         1.001  N47.G2
         0.747  xor_crc/C<16>.G2
         1.052  xor_crc/xor_func/out_xor<29>.BX
         0.706  N83.F2
         0.890  N83.G1
         0.416  xor_crc/C<28>.G2

xor_crc/C<2>
   xor_crc/C<2>.XQ
         0.835  refout/_n0034.G1
         0.859  refout/_n0063.F1
         0.251  N83.G3
         1.050  xor_crc/xor_func/out_xor<3>.BY

xor_crc/C<30>
   xor_crc/C<30>.XQ
         0.445  refout/_n0058.G1
         0.221  refout/_n0033.F1
         0.982  xor_crc/C<12>.G2
         0.758  xor_crc/C<5>.G1
         1.388  N71.F2
         0.734  xor_crc/C<17>.G2
         1.226  N37.G2
         1.414  N47.F1
         1.275  xor_crc/xor_func/out_xor<31>.BY
         0.659  xor_crc/C<18>.G2
         0.520  xor_crc/C<29>.G2
         0.291  xor_crc/C<28>.G1
         0.763  xor_crc/C<23>.G1
```

```
xor_crc/C<31>
   xor_crc/C<31>.YQ
         0.778  refout/_n0060.G1
         0.802  refout/_n0032.F1
         0.537  xor_crc/C<1>.F2
         0.974  xor_crc/C<2>.G4
         0.909  N39.F1
         0.886  N39.G4
         0.538  N37.F1
         1.207  N47.G4
         1.060  xor_crc/xor_func/out_xor<31>.BX
         0.800  xor_crc/C<24>.G1
         0.537  xor_crc/C<19>.F3
         0.675  N83.F1
         0.675  N91.F1
         0.652  N91.G2
         0.710  xor_crc/C<18>.G1
         0.517  xor_crc/C<29>.G1
         0.392  xor_crc/C<30>.F3

xor_crc/C<3>
   xor_crc/C<3>.XQ
         1.114  refout/_n0035.G1
         0.999  refout/_n0062.F1
         0.368  xor_crc/C<11>.F1
         1.014  xor_crc/xor_func/out_xor<3>.BX

xor_crc/C<4>
   xor_crc/C<4>.XQ
         0.273  refout/_n0036.G1
         0.398  refout/_n0061.F1
         0.506  xor_crc/C<12>.F1
         1.002  xor_crc/xor_func/out_xor<5>.BY

xor_crc/C<5>
   xor_crc/C<5>.XQ
         0.654  refout/_n0037.G1
         0.539  refout/_n0059.F1
         0.702  N71.F1
         1.268  xor_crc/xor_func/out_xor<5>.BX

xor_crc/C<6>
   xor_crc/C<6>.XQ
         0.857  refout/_n0038.G1
         0.509  refout/_n0057.F1
         0.273  N37.G1
         1.008  xor_crc/xor_func/out_xor<7>.BY

xor_crc/C<7>
   xor_crc/C<7>.XQ
         0.253  refout/_n0039.G1
         0.277  refout/_n0056.F1
         0.258  N91.G1
         0.773  xor_crc/xor_func/out_xor<7>.BX

xor_crc/C<8>
   xor_crc/C<8>.XQ
         0.674  refout/_n0054.F1
         0.512  refout/_n0040.G1
         0.894  xor_crc/C<16>.G1
         1.275  xor_crc/xor_func/out_xor<9>.BY

xor_crc/C<9>
   xor_crc/C<9>.YQ
         0.501  refout/_n0052.F1
         0.725  refout/_n0041.G1
         0.771  xor_crc/C<17>.G1
         1.019  xor_crc/xor_func/out_xor<9>.BX

xor_crc/Mxor_NewCRC<10>_Xo<1>1/O
   xor_crc/C<11>.Y
         0.001  xor_crc/C<11>.DY
xor_crc/Mxor_NewCRC<11>_Xo<1>1/O
```

```
   xor_crc/C<11>.X
         0.001  xor_crc/C<11>.DX

xor_crc/Mxor_NewCRC<12>_Xo<3>1/O
   xor_crc/C<12>.X
         0.001  xor_crc/C<12>.DX

xor_crc/Mxor_NewCRC<13>_Xo<2>1_SW0/O
   N71.Y
         0.002  N71.F4

xor_crc/Mxor_NewCRC<13>_Xo<3>1/O
   xor_crc/C<13>.X
         0.001  xor_crc/C<13>.DX

xor_crc/Mxor_NewCRC<14>_Xo<2>1/O
   xor_crc/C<14>.Y
         0.001  xor_crc/C<14>.DY

xor_crc/Mxor_NewCRC<15>_Xo<3>1/O
   xor_crc/C<15>.X
         0.001  xor_crc/C<15>.DX

xor_crc/Mxor_NewCRC<16>_Xo<0>1_SW0/O
   xor_crc/C<15>.Y
         0.002  xor_crc/C<15>.F4

xor_crc/Mxor_NewCRC<16>_Xo<2>2/O
   xor_crc/C<16>.X
         0.001  xor_crc/C<16>.DX

xor_crc/Mxor_NewCRC<17>_Xo<2>2/O
   xor_crc/C<17>.X
         0.001  xor_crc/C<17>.DX

xor_crc/Mxor_NewCRC<18>_Xo<2>2/O
   xor_crc/C<18>.X
         0.001  xor_crc/C<18>.DX

xor_crc/Mxor_NewCRC<19>_Xo<1>2/O
   xor_crc/C<19>.X
         0.001  xor_crc/C<19>.DX

xor_crc/Mxor_NewCRC<1>_Result1/O
   xor_crc/C<1>.Y
         0.001  xor_crc/C<1>.DY

xor_crc/Mxor_NewCRC<20>_Result1/O
   xor_crc/C<21>.Y
         0.001  xor_crc/C<21>.DY

xor_crc/Mxor_NewCRC<21>_Result1/O
   xor_crc/C<21>.X
         0.001  xor_crc/C<21>.DX

xor_crc/Mxor_NewCRC<23>_Xo<2>2/O
   xor_crc/C<23>.X
         0.001  xor_crc/C<23>.DX

xor_crc/Mxor_NewCRC<24>_Xo<2>2/O
   xor_crc/C<24>.X
         0.001  xor_crc/C<24>.DX

xor_crc/Mxor_NewCRC<25>_Xo<1>2/O
   xor_crc/C<25>.X
         0.001  xor_crc/C<25>.DX

xor_crc/Mxor_NewCRC<26>_Xo<2>1/O
   xor_crc/C<27>.Y
         0.001  xor_crc/C<27>.DY

xor_crc/Mxor_NewCRC<27>_Xo<1>
   xor_crc/C<8>.Y
         0.199  xor_crc/C<12>.F3
```

```
        0.061  xor_crc/C<8>.F2
        0.361  xor_crc/C<9>.G3
        0.294  xor_crc/C<11>.F2
        0.432  xor_crc/C<27>.F3

xor_crc/Mxor_NewCRC<27>_Xo<3>1/O
   xor_crc/C<27>.X
        0.001  xor_crc/C<27>.DX

xor_crc/Mxor_NewCRC<28>_Xo<0>
   xor_crc/C<9>.X
        0.398  xor_crc/C<12>.F2

xor_crc/Mxor_NewCRC<28>_Xo<2>2/O
   xor_crc/C<28>.X
        0.001  xor_crc/C<28>.DX

xor_crc/Mxor_NewCRC<29>_Xo<1>2/O
   xor_crc/C<29>.X
        0.001  xor_crc/C<29>.DX

xor_crc/Mxor_NewCRC<2>_Xo<1>1/O
   xor_crc/C<2>.X
        0.001  xor_crc/C<2>.DX

xor_crc/Mxor_NewCRC<2>_Xo<1>1_SW0/O
   xor_crc/C<2>.Y
        0.002  xor_crc/C<2>.F4

xor_crc/Mxor_NewCRC<30>_Xo<1>2/O
   xor_crc/C<30>.X
        0.001  xor_crc/C<30>.DX

xor_crc/Mxor_NewCRC<31>_Result1/O
   xor_crc/C<31>.Y
        0.001  xor_crc/C<31>.DY

xor_crc/Mxor_NewCRC<3>_Xo<1>1/O
   xor_crc/C<3>.X
        0.001  xor_crc/C<3>.DX

xor_crc/Mxor_NewCRC<4>_Xo<1>1/O
   xor_crc/C<4>.X
        0.001  xor_crc/C<4>.DX

xor_crc/Mxor_NewCRC<4>_Xo<1>1_SW0/O
   xor_crc/C<4>.Y
        0.002  xor_crc/C<4>.F4

xor_crc/Mxor_NewCRC<5>_Xo<1>
   xor_crc/C<1>.X
        0.364  xor_crc/C<5>.F4

xor_crc/Mxor_NewCRC<5>_Xo<2>1/O
   xor_crc/C<5>.X
        0.001  xor_crc/C<5>.DX

xor_crc/Mxor_NewCRC<5>_Xo<2>1_SW0/O
   xor_crc/C<5>.Y
        0.252  xor_crc/C<5>.F2

xor_crc/Mxor_NewCRC<6>_Xo<2>1/O
   xor_crc/C<6>.X
        0.001  xor_crc/C<6>.DX

xor_crc/Mxor_NewCRC<7>_Xo<1>1/O
   xor_crc/C<7>.X
        0.001  xor_crc/C<7>.DX

xor_crc/Mxor_NewCRC<8>_Xo<1>1/O
   xor_crc/C<8>.X
        0.001  xor_crc/C<8>.DX

xor_crc/Mxor_NewCRC<9>_Xo<1>1/O
```

```
   xor_crc/C<9>.Y
         0.001  xor_crc/C<9>.DY

xor_crc/Mxor__n0008_Result1/O
   xor_crc/C<3>.Y
         0.028  xor_crc/C<3>.F4

xor_crc/Mxor__n0036_Result1_SW0_SW0/O
   N33.Y
         0.002  N33.F1

xor_crc/Mxor__n0036_Result1_SW1/O
   xor_crc/C<6>.Y
         0.002  xor_crc/C<6>.F3

xor_crc/NewCRC<0>
   xor_crc/C<12>.Y
         0.021  xor_crc/C<12>.F4
         0.020  xor_crc/C<12>.DY
         0.453  xor_crc/C<4>.F3
         0.427  xor_crc/C<27>.G3
         0.296  xor_crc/C<1>.G3
         0.398  xor_crc/C<2>.F3

xor_crc/NewCRC<22>
   xor_crc/C<22>.Y
         0.001  xor_crc/C<22>.DY

xor_crc/_n0052<2>
   xor_crc/C<13>.Y
         0.456  xor_crc/C<13>.F3
         0.158  xor_crc/C<14>.G3

xor_crc/_n0104<1>
   xor_crc/C<7>.Y
         0.029  xor_crc/C<7>.F3
         0.923  xor_crc/C<8>.F3
         0.256  xor_crc/C<11>.F3
         0.287  xor_crc/C<11>.G3

xor_crc/xor_func/out_xor<0>
   xor_crc/xor_func/out_xor<1>.YQ
         0.835  refout/_n0060.F3
         0.811  refout/_n0032.G3

xor_crc/xor_func/out_xor<10>
   xor_crc/xor_func/out_xor<11>.YQ
         0.390  refout/_n0050.F3
         0.228  refout/_n0042.G3

xor_crc/xor_func/out_xor<11>
   xor_crc/xor_func/out_xor<11>.XQ
         0.344  refout/_n0043.G3
         0.229  refout/_n0048.F3

xor_crc/xor_func/out_xor<12>
   xor_crc/xor_func/out_xor<13>.YQ
         0.450  refout/_n0044.G3
         0.226  refout/_n0055.F3

xor_crc/xor_func/out_xor<13>
   xor_crc/xor_func/out_xor<13>.XQ
         0.690  refout/_n0053.F3
         0.526  refout/_n0045.G3

xor_crc/xor_func/out_xor<14>
   xor_crc/xor_func/out_xor<15>.YQ
         0.656  refout/_n0051.F3
         0.633  refout/_n0046.G3

xor_crc/xor_func/out_xor<15>
   xor_crc/xor_func/out_xor<15>.XQ
         0.494  refout/_n0047.G3
         0.658  refout/_n0049.F3
```

```
xor_crc/xor_func/out_xor<16>
   xor_crc/xor_func/out_xor<17>.YQ
         0.200  refout/_n0047.F3
         0.424  refout/_n0049.G3

xor_crc/xor_func/out_xor<17>
   xor_crc/xor_func/out_xor<17>.XQ
         0.494  refout/_n0051.G3
         0.518  refout/_n0046.F3

xor_crc/xor_func/out_xor<18>
   xor_crc/xor_func/out_xor<19>.YQ
         0.466  refout/_n0053.G3
         0.628  refout/_n0045.F3

xor_crc/xor_func/out_xor<19>
   xor_crc/xor_func/out_xor<19>.XQ
         0.226  refout/_n0044.F3
         0.450  refout/_n0055.G3

xor_crc/xor_func/out_xor<1>
   xor_crc/xor_func/out_xor<1>.XQ
         0.654  refout/_n0058.F3
         0.492  refout/_n0033.G3

xor_crc/xor_func/out_xor<20>
   xor_crc/xor_func/out_xor<21>.YQ
         0.203  refout/_n0043.F3
         0.427  refout/_n0048.G3

xor_crc/xor_func/out_xor<21>
   xor_crc/xor_func/out_xor<21>.XQ
         0.271  refout/_n0050.G3
         0.435  refout/_n0042.F3

xor_crc/xor_func/out_xor<22>
   xor_crc/xor_func/out_xor<23>.YQ
         0.257  refout/_n0052.G3
         0.421  refout/_n0041.F3

xor_crc/xor_func/out_xor<23>
   xor_crc/xor_func/out_xor<23>.XQ
         0.242  refout/_n0054.G3
         0.404  refout/_n0040.F3

xor_crc/xor_func/out_xor<24>
   xor_crc/xor_func/out_xor<25>.YQ
         0.364  refout/_n0039.F3
         0.341  refout/_n0056.G3

xor_crc/xor_func/out_xor<25>
   xor_crc/xor_func/out_xor<25>.XQ
         0.204  refout/_n0038.F3
         0.273  refout/_n0057.G3

xor_crc/xor_func/out_xor<26>
   xor_crc/xor_func/out_xor<27>.YQ
         0.226  refout/_n0037.F3
         0.450  refout/_n0059.G3

xor_crc/xor_func/out_xor<27>
   xor_crc/xor_func/out_xor<27>.XQ
         0.228  refout/_n0036.F3
         0.492  refout/_n0061.G3

xor_crc/xor_func/out_xor<28>
   xor_crc/xor_func/out_xor<29>.YQ
         0.398  refout/_n0035.F3
         0.234  refout/_n0062.G3

xor_crc/xor_func/out_xor<29>
   xor_crc/xor_func/out_xor<29>.XQ
         0.421  refout/_n0034.F3
```

```
        0.450  refout/_n0063.G3

xor_crc/xor_func/out_xor<2>
   xor_crc/xor_func/out_xor<3>.YQ
        0.344  refout/_n0034.G3
        0.367  refout/_n0063.F3

xor_crc/xor_func/out_xor<30>
   xor_crc/xor_func/out_xor<31>.YQ
        1.002  refout/_n0058.G3
        0.887  refout/_n0033.F3

xor_crc/xor_func/out_xor<31>
   xor_crc/xor_func/out_xor<31>.XQ
        0.381  refout/_n0060.G3
        0.404  refout/_n0032.F3

xor_crc/xor_func/out_xor<3>
   xor_crc/xor_func/out_xor<3>.XQ
        0.257  refout/_n0035.G3
        0.421  refout/_n0062.F3

xor_crc/xor_func/out_xor<4>
   xor_crc/xor_func/out_xor<5>.YQ
        0.466  refout/_n0036.G3
        0.206  refout/_n0061.F3

xor_crc/xor_func/out_xor<5>
   xor_crc/xor_func/out_xor<5>.XQ
        0.424  refout/_n0037.G3
        0.200  refout/_n0059.F3

xor_crc/xor_func/out_xor<6>
   xor_crc/xor_func/out_xor<7>.YQ
        0.508  refout/_n0038.G3
        0.411  refout/_n0057.F3

xor_crc/xor_func/out_xor<7>
   xor_crc/xor_func/out_xor<7>.XQ
        0.257  refout/_n0039.G3
        0.226  refout/_n0056.F3

xor_crc/xor_func/out_xor<8>
   xor_crc/xor_func/out_xor<9>.YQ
        0.421  refout/_n0054.F3
        0.257  refout/_n0040.G3

xor_crc/xor_func/out_xor<9>
   xor_crc/xor_func/out_xor<9>.XQ
        0.390  refout/_n0052.F3
        0.228  refout/_n0041.G3

xor_done_IBUF
   xor_done.I
        0.922  refout/_n0050.F2
        0.491  refout/_n0050.G2
        2.009  refout/_n0034.F2
        1.752  refout/_n0034.G2
        0.653  refout/_n0042.F2
        0.951  refout/_n0042.G2
        1.961  refout/_n0051.F2
        1.990  refout/_n0051.G2
        1.766  refout/_n0035.F2
        1.985  refout/_n0035.G2
        1.170  refout/_n0043.F2
        0.738  refout/_n0043.G2
        1.277  refout/_n0052.F2
        1.491  refout/_n0052.G2
        1.266  refout/_n0036.F2
        1.242  refout/_n0036.G2
        2.180  refout/_n0044.F2
        1.792  refout/_n0044.G2
        1.766  refout/_n0060.F2
        1.797  refout/_n0060.G2
```

```
1.501  refout/_n0053.F2
1.532  refout/_n0053.G2
1.033  refout/_n0061.F2
1.469  refout/_n0061.G2
1.266  refout/_n0037.F2
1.242  refout/_n0037.G2
1.696  refout/_n0045.F2
1.725  refout/_n0045.G2
1.277  refout/_n0054.F2
1.491  refout/_n0054.G2
1.761  refout/_n0062.F2
1.990  refout/_n0062.G2
1.696  refout/_n0038.F2
1.725  refout/_n0038.G2
1.961  refout/_n0046.F2
1.990  refout/_n0046.G2
1.956  refout/_n0055.F2
2.016  refout/_n0055.G2
1.765  refout/_n0047.F2
1.725  refout/_n0047.G2
1.723  refout/_n0063.F2
2.040  refout/_n0063.G2
1.734  refout/_n0039.F2
1.958  refout/_n0039.G2
1.929  refout/_n0056.F2
1.765  refout/_n0056.G2
0.900  refout/_n0048.F2
1.006  refout/_n0048.G2
1.501  refout/_n0057.F2
1.037  refout/_n0057.G2
1.501  refout/_n0049.F2
1.880  refout/_n0049.G2
1.679  refout/_n0058.F2
1.724  refout/_n0058.G2
1.404  refout/_n0059.F2
1.381  refout/_n0059.G2
1.766  refout/_n0032.F2
1.797  refout/_n0032.G2
1.267  refout/_n0040.F2
1.392  refout/_n0040.G2
1.886  refout/_n0033.F2
1.515  refout/_n0033.G2
1.267  refout/_n0041.F2
1.298  refout/_n0041.G2
```