

PETUNJUK PRAKTIKUM

Praktikum Sistem Digital



**Laboratorium Dasar
Teknik Elektro**

Sekolah Teknik Elektro Dan Informatika

Institut Teknologi Bandung

2023

Buku Petunjuk Praktikum Sistem Digital EL 2102

Mervin T. Hutabarat

Arif Sasongko

Eric Agustian

Harry Septanto

M. Zakiyullah R.

Ardimas Andi Purwita

Nina Lestari

**Sekolah Teknik Elektro Dan Informatika
Institut Teknologi Bandung
2023**

DAFTAR ISI

| | |
|--|------|
| DAFTAR ISI | i |
| ATURAN UMUM LABORATORIUM | v |
| KELENGKAPAN | v |
| PERSIAPAN | v |
| PERGANTIAN JADWAL | vi |
| SANKSI | vi |
| PANDUAN UMUM KESELAMATAN DAN PENGGUNAAN PERALATAN LABORATORIUM | vii |
| KESELAMATAN | vii |
| PENGGUNAAN PERALATAN PRAKTIKUM | viii |
| SANKSI | ix |
| TABEL SANKSI PRAKTIKUM | x |
| PLAGIARISME DAN KECURANGAN AKADEMIK | xi |
| BENTUK-BENTUK PLAGIARISME | xi |
| PRINSIP MENGHINDARI PLAGIARISME | xi |
| PERCOBAAN I | 1 |
| PARAMETER GERBANG LOGIKA | 1 |
| 1.1 TUJUAN | 1 |
| 1.2 PERSIAPAN | 1 |
| 1.3 DASAR TEORI | 1 |
| 1.4 TUGAS PENDAHULUAN | 4 |
| 1.5 PERCOBAAN | 5 |
| 1.6 MENGAKHIRI PERCOBAAN | 10 |
| PERCOBAAN II | 11 |
| Pengenalan Desain Menggunakan FPGA | 11 |
| 1.1 TUJUAN | 11 |
| 1.2 PERSIAPAN | 11 |
| 1.3 DASAR TEORI | 11 |
| 1.4 TUGAS PENDAHULUAN | 13 |
| 1.5 PERCOBAAN | 13 |

| | |
|--|----|
| 1.6 MENGAKHIRI PERCOBAAN | 36 |
| PERCOBAAN III..... | 37 |
| RANGKAIAN LOGIKA KOMBINASIONAL | 37 |
| 1.1 TUJUAN..... | 37 |
| 1.2 PERSIAPAN..... | 37 |
| 1.3 DASAR TEORI..... | 37 |
| 1.4 TUGAS PENDAHULUAN..... | 39 |
| 1.5 PERCOBAAN..... | 40 |
| 1.6 MENGAKHIRI PERCOBAAN | 52 |
| PERCOBAAN IV..... | 56 |
| RANGKAIAN LOGIKA SEKUENSIAL | 56 |
| 1.1 TUJUAN..... | 56 |
| 1.2 PERSIAPAN..... | 56 |
| 1.3 DASAR TEORI..... | 56 |
| 1.4 TUGAS PENDAHULUAN..... | 58 |
| 1.5 PERCOBAAN..... | 59 |
| 1.6 MENGAKHIRI PERCOBAAN | 66 |
| PERCOBAAN V..... | 68 |
| PERANCANGAN DAN IMPLEMENTASI DISPLAY LCD MENGGUNAKAN MODUL VGA PADA FPGA | 68 |
| 1.1 TUJUAN..... | 68 |
| 1.2 PERSIAPAN..... | 68 |
| 1.3 DASAR TEORI..... | 68 |
| 1.4 TUGAS PENDAHULUAN..... | 71 |
| 1.5 PERCOBAAN..... | 73 |
| 1.6 MENGAKHIRI PERCOBAAN | 75 |
| APENDIKS A..... | 1 |
| PENJELASAN KAKI GERBANG LOGIKA | 1 |
| 74LS00 2 INPUT NAND GATE 74LS02 2 INPUT NOR GATE | 1 |
| 74LS08 2-INPUT AND GATE 74LS04 INVERTER GATE | 1 |
| 74LS10 3-INPUT NAND GATE 74LS11 3-INPUT AND GATE | 2 |
| 74LS27 3-INPUT NOR GATE | 2 |
| APENDIX B..... | 1 |

| | |
|---|----------|
| <i>VHDL REFERENCE.....</i> | <i>1</i> |
| <i>STRUKTUR KODE VHDL.....</i> | <i>1</i> |
| <i>LIBRARY DAN PACKAGE.....</i> | <i>1</i> |
| <i>ENTITY.....</i> | <i>2</i> |
| <i>ARCHITECTURE</i> | <i>2</i> |
| <i>INSTANSIASI COMPONENT.....</i> | <i>2</i> |
| <i>GENERATE STATEMENT.....</i> | <i>3</i> |
| <i>KONKUREN ASSIGNMENT STATEMENT.....</i> | <i>4</i> |
| <i>SEKUENSIAL ASSIGNMENT STATEMENT.....</i> | <i>5</i> |
| <i>OBJEK DAN TIPE DATA.....</i> | <i>6</i> |
| <i>NOTASI ANGKA</i> | <i>8</i> |
| <i>OPERATOR</i> | <i>9</i> |
| <i>TYPE CONVERSION.....</i> | <i>9</i> |

ATURAN UMUM LABORATORIUM

KELENGKAPAN

Setiap praktikan wajib berpakaian lengkap, mengenakan **masker, celana panjang/ rok, kemeja** dan mengenakan **sepatu**. Untuk memasuki ruang laboratorium Praktikan wajib membawa kelengkapan berikut:

- Modul praktikum
- Buku Catatan Laboratorium (BCL)
- Alat tulis (dan kalkulator, jika diperlukan)
- *Name tag*
- Kartu Praktikum

PERSIAPAN

SEBELUM PRAKTIKUM

Sebelum mengikuti percobaan sesuai jadwalnya, sebelum memasuki laboratorium praktikan harus mempersiapkan diri dengan melakukan hal-hal berikut:

- Membaca dan memahami isi modul praktikum,
- Mengerjakan Tugas Pendahuluan
- Mengerjakan hal-hal yang harus dikerjakan sebelum praktikum dilaksanakan, misalnya mengerjakan perhitungan-perhitungan, menyalin source code, mengisi Kartu Praktikum dlsb.,
- Mengisi daftar hadir di Tata Usaha Laboratorium,
- Mengambil kunci loker dan melengkapi administrasi peminjaman kunci loker dengan meninggalkan kartu identitas (KTM/ SIM/ KTP).

SELAMA PRAKTIKUM

Setelah dipersilahkan masuk dan menempati bangku dan meja kerja, praktikan haruslah:

- Menuliskan identitas diri pada Berita Acara Praktikum yang diedarkan oleh asisten
- Memperhatikan dan mengerjakan setiap percobaan dengan waktu sebaik-baiknya, diawali dengan kehadiran praktikan secara tepat waktu,
- Mengumpulkan dan meminta paraf asisten pada Kartu Praktikum,
- Mendokumentasikan dalam Buku Catatan Laboratorium. (lihat Petunjuk Penggunaan BCL) tentang hal-hal penting terkait percobaan yang sedang dilakukan.

SETELAH PRAKTIKUM

- Memastikan BCL telah ditandatangani oleh asisten,
- Mengembalikan kunci loker dan melengkapi administrasi pengembalian kunci loker (pastikan kartu identitas KTM/ SIM/ KTP diperoleh kembali),
- Mengerjakan laporan dalam bentuk SoftCopy (lihat Panduan Penyusunan Laporan di laman <http://ldte.stei.itb.ac.id/>),
- Mengirimkan file laporan dengan cara mengunggah di laman <http://praktikum.stei.itb.ac.id>. Waktu pengiriman paling lambat jam 11.00 WIB, dua hari kerja berikutnya setelah praktikum, kecuali ada kesepakatan lain antara Dosen Pengajar dan/ atau Asisten.
- Mengumpulkan BCL secara fisik ke TU LDTE. Waktu pengumpulan paling lambat jam 11.00 WIB, dua hari kerja berikutnya setelah praktikum, kecuali ada kesepakatan lain antara Dosen Pengajar dan/ atau Asisten. (Kartu Praktikum wajib dibawa pada saat mengumpulkan BCL).

PERGANTIAN JADWAL

KASUS BIASA

Pertukaran jadwal hanya dapat dilakukan per orang dengan modul yang sama. Langkah untuk menukar jadwal adalah sebagai berikut:

- Lihatlah format Pertukaran Jadwal di <http://ldte.stei.itb.ac.id/> pada halaman Panduan
- Salah satu praktikan yang bertukar jadwal harus mengirimkan format pertukaran jadwal via OA Line LDTE : @kiy3574q atau e-mail ke labdasar@stei.itb.ac.id . Waktu pengiriman paling lambat jam 16.30, sehari sebelum praktikum yang dipertukarkan.
- Pertukaran diperbolehkan setelah ada konfirmasi dari Lab. Dasar

KASUS SAKIT ATAU URUSAN MENDESAK PRIBADI LAINNYA

Jadwal pengganti dapat diberikan kepada praktikan yang sakit atau memiliki urusan mendesak pribadi.

- Praktikan yang hendak mengubah jadwal untuk urusan pribadi mendesak harus memberitahu staf tata usaha laboratorium sebelum jadwal praktikumnya melalui Line / email.
- Segera setelah praktikan memungkinkan mengikuti kegiatan akademik, praktikan dapat mengikuti praktikum pengganti setelah mendapatkan konfirmasi dari staf tata usaha laboratorium dengan melampirkan surat keterangan dokter bagi yang sakit atau surat terkait untuk yang memiliki urusan pribadi.

KASUS "KEPENTINGAN MASSAL"

"Kepentingan massal" terjadi jika ada lebih dari 1/3 rombongan praktikan yang tidak dapat melaksanakan praktikum pada satu hari yang sama karena alasan yang terkait kegiatan akademis

SANKSI

Pengabaian aturan-aturan di atas dapat dikenakan sanksi pengurangan nilai praktikum terkait.

PANDUAN UMUM KESELAMATAN DAN PENGGUNAAN PERALATAN LABORATORIUM

KESELAMATAN

Pada prinsipnya, untuk mewujudkan praktikum yang aman diperlukan partisipasi seluruh praktikan dan asisten pada praktikum yang bersangkutan. Dengan demikian, kepatuhan setiap praktikan terhadap uraian panduan pada bagian ini akan sangat membantu mewujudkan praktikum yang aman.

BAHAYA LISTRIK

- Perhatikan dan pelajari tempat-tempat sumber listrik (*stop-kontak* dan *circuit breaker*) dan cara menyala-matikannya. Jika melihat ada kerusakan yang berpotensi menimbulkan bahaya, laporkan pada asisten
- Hindari daerah atau benda yang berpotensi menimbulkan bahaya listrik (sengatan listrik/ *strum*) secara tidak disengaja, misalnya kabel jala-jala yang terkelupas dll.
- Tidak melakukan sesuatu yang dapat menimbulkan bahaya listrik pada diri sendiri atau orang lain
- Keringkan bagian tubuh yang basah karena, misalnya, keringat atau sisa air wudhu
- Selalu waspada terhadap bahaya listrik pada setiap aktivitas praktikum

Kecelakaan akibat bahaya listrik yang sering terjadi adalah tersengat arus listrik. Berikut ini adalah hal-hal yang harus diikuti praktikan jika hal itu terjadi:

- Jangan panik
- Matikan semua peralatan elektronik dan sumber listrik di meja masing-masing dan di meja praktikan yang tersengat arus listrik
- Bantu praktikan yang tersengat arus listrik untuk melepaskan diri dari sumber listrik
- Beritahukan dan minta bantuan asisten, praktikan lain dan orang di sekitar anda tentang terjadinya kecelakaan akibat bahaya listrik

BAHAYA API ATAU PANAS BERLEBIH

- Jangan membawa benda-benda mudah terbakar (korek api, gas dll.) ke dalam ruang praktikum bila tidak disyaratkan dalam modul praktikum
- Jangan melakukan sesuatu yang dapat menimbulkan api, percikan api atau panas yang berlebihan

- Jangan melakukan sesuatu yang dapat menimbulkan bahaya api atau panas berlebih pada diri sendiri atau orang lain
- Selalu waspada terhadap bahaya api atau panas berlebih pada setiap aktivitas praktikum

Berikut ini adalah hal-hal yang harus diikuti praktikan jika menghadapi bahaya api atau panas berlebih:

- Jangan panik
- Beritahukan dan minta bantuan asisten, praktikan lain dan orang di sekitar anda tentang terjadinya bahaya api atau panas berlebih
- Matikan semua peralatan elektronik dan sumber listrik di meja masing-masing
- Menjauh dari ruang praktikum

BAHAYA BENDA TAJAM DAN LOGAM

- Dilarang membawa benda tajam (pisau, gunting dan sejenisnya) ke ruang praktikum bila tidak diperlukan untuk pelaksanaan percobaan
- Dilarang memakai perhiasan dari logam misalnya cincin, kalung, gelang dll.
- Hindari daerah, benda atau logam yang memiliki bagian tajam dan dapat melukai
- Tidak melakukan sesuatu yang dapat menimbulkan luka pada diri sendiri atau orang lain

LAIN-LAIN

- Dilarang membawa makanan dan minuman ke dalam ruang praktikum

PENGUNAAN PERALATAN PRAKTIKUM

Berikut ini adalah panduan yang harus dipatuhi ketika menggunakan alat-alat praktikum:

- Sebelum menggunakan alat-alat praktikum, pahami petunjuk penggunaan alat itu. Petunjuk penggunaan beberapa alat dapat didownload di <http://labdasar.ee.itb.ac.id>
- Perhatikan dan patuhi peringatan (*warning*) yang biasa tertera pada badan alat
- Pahami fungsi atau peruntukan alat-alat praktikum dan gunakanlah alat-alat tersebut hanya untuk aktivitas yang sesuai fungsi atau peruntukannya. Menggunakan alat praktikum di luar fungsi atau peruntukannya dapat menimbulkan kerusakan pada alat tersebut dan bahaya keselamatan praktikan
- Pahami *rating* dan jangkauan kerja alat-alat praktikum dan gunakanlah alat-alat tersebut sesuai *rating* dan jangkauan kerjanya. Menggunakan alat praktikum di luar *rating* dan jangkauan kerjanya dapat menimbulkan kerusakan pada alat tersebut dan bahaya keselamatan praktikan

- Pastikan seluruh peralatan praktikum yang digunakan aman dari benda/ logam tajam, api/ panas berlebih atau lainnya yang dapat mengakibatkan kerusakan pada alat tersebut
- Tidak melakukan aktifitas yang dapat menyebabkan kotor, coretan, goresan atau sejenisnya pada badan alat-alat praktikum yang digunakan

SANKSI

Pengabaian uraian panduan di atas dapat dikenakan sanksi tidak lulus mata kuliah praktikum yang bersangkutan

TABEL SANKSI PRAKTIKUM

Berlaku mulai: 14 Agustus 2017

| Level | | Kasus | Sanksi | Pengurangan nilai per modul |
|----------|----------------------------|---|----------------------------|------------------------------|
| Akademik | Saat dan setelah praktikum | Semua kegiatan plagiasi (mencontek): tugas pendahuluan, test dalam praktikum, laporan praktikum | Gugur praktikum | |
| | | Sengaja tidak mengikuti praktikum | | |
| Berat | Saat praktikum | Terlambat hadir praktikum | Gugur modul | |
| | | Pakaian tidak sesuai: kemeja, sepatu | | |
| | | Tugas pendahuluan tidak dikerjakan/hilang/tertinggal | | |
| Ringan | Saat Praktikum | Tidak mempelajari modul sebelum praktikum/tidak mengerti isi modul | Dikeluarkan dari praktikum | |
| | | Pertukaran jadwal tidak sesuai ketentuan | | -25 nilai akhir |
| | | BCL tertinggal/hilang | | -100% nilai BCL |
| | | Name Tag tertinggal/hilang | | -10 nilai akhir |
| | | Kartu praktikum tertinggal/hilang | | -25 nilai akhir |
| | | Kartu praktikum tidak lengkap data dan foto | | -10 nilai akhir |
| | | Loker tidak dikunci/kunci tertinggal | | -10 nilai akhir |
| | Setelah Praktikum | Tidak minta paraf asisten di BCL/kartu praktikum | | -25 nilai akhir |
| | | Terlambat mengumpulkan laporan | | -1/min nilai akhir, maks -50 |
| | | Terlambat mengumpulkan BCL | | -1/min nilai BCL, maks -50 |
| | | Tidak bawa kartu praktikum saat pengumpulan BCL | | -50 nilai BCL |
| | | Tidak minta paraf admin saat pengumpulan BCL | | -50 nilai BCL |

Catatan:

- Pelanggaran akademik menyebabkan gugur praktikum, nilai praktikum E
- Dalam satu praktikum, praktikan maksimal boleh melakukan
 - 1 pelanggaran berat dan 1 pelanggaran ringan; atau
 - 3 pelanggaran ringan
- Jika jumlah pelanggaran melewati point 2, praktikan dianggap gugur praktikum.
- Praktikan yang terkena sanksi gugur modul wajib mengganti praktikum pada hari lain dengan nilai modul tetap 0. Waktu pengganti praktikum ditetapkan bersama asisten. Jika praktikan tidak mengikuti ketentuan praktikum (pengganti) dengan baik, akan dikenakan sanksi gugur praktikum.
- Setiap pelanggaran berat dan ringan dicatat/diberikan tanda di kartu praktikum
- Waktu acuan adalah waktu sinkron dengan NIST
- Sanksi yang tercantum di tabel adalah sanksi minimum.
- Sanksi yang belum tercantum akan ditentukan kemudian.

PLAGIARISME DAN KECURANGAN AKADEMIK

Plagiarisme merupakan salah satu bentuk kecurangan akademik. Definisi **plagiarisme** sesuai Peraturan Akademik ITB adalah **menggunakan kata-kata atau karya orang lain sebagai kata-kata atau karya sendiri dalam suatu kegiatan akademik tanpa menyebutkan acuan yang dipakai**. Plagiarisme bisa dilakukan secara sengaja, akibat kecerobohan, maupun tidak sengaja. Plagiarisme merupakan pelanggaran integritas akademik. **Prinsip kejujuran intelektual** menyiratkan bahwa semua anggota komunitas akademik harus mengakui peran pemilik gagasan awal dalam hal kata-kata dan data yang membentuk dasar untuk pekerjaan mereka sendiri. Mengakui karya orang lain sebagai milik anda memberi makna bahwa anda telah gagal menyelesaikan proses pembelajaran. Plagiarisme adalah sangat tidak etis dan memiliki konsekuensi serius bagi karir masa depan Anda sekaligus merusak reputasi institusi.

BENTUK-BENTUK PLAGIARISME

1. Mengutip kata demi kata (Verbatim)
2. Parafrase: menuliskan kembali karya hasil orang lain dengan mengubah kata atau mengubah urutan kalimat, dengan mengikuti struktur argumen orang lain tersebut tanpa menyebutkan acuan.
3. Kolusi: kolaborasi tidak sah antar mahasiswa tanpa atribusi terhadap bantuan dari luar yang diterima, atau tidak mengikuti sebenarnya pada peraturan kerja berkelompok
4. Kutipan tidak akurat: salah kutip atau mencantumkan referensi yang tidak pernah dikutip.
5. Apresiasi (*acknowledgement*) tidak akurat: tidak menyebutkan kontribusi pihak yang berkontribusi atau sebaliknya memberi apresiasi pada pihak yang tidak berkontribusi.
6. Menggunakan jasa pihak ketiga, profesional maupun tidak.

PRINSIP MENGHINDARI PLAGIARISME

1. Semua karya ilmiah harus dilandasi latar belakang, motivasi, dan lain sebagainya yang bisa dipertanggungjawabkan secara ilmiah. Adalah wajib untuk menggunakan referensi untuk mendukung ide-ide yang telah Anda kembangkan.
2. Dalam karya ilmiah, Anda harus menunjukkan bahwa Anda memiliki pemahaman yang jelas dan benar tentang materi yang telah Anda dapatkan dari referensi.
3. Berikan kejelasan antara analisa (ide) original Anda dengan apa yang telah diambil dari referensi:
 - Berikan penanda bagian mana suatu paragraf adalah berasal dari referensi.
 - Kutipan harus selalu diidentifikasi dengan menggunakan tanda kutip atau indentasi, dan dengan referensi penuh dari sumber yang dikutip.
 - Untuk menghindari parafrase, lebih baik menuliskan kembali ringkasan singkat dari keseluruhan sumber dengan kata-kata sendiri, dan dengan jelas menunjukkan bahwa itu yang dilakukan sehingga jelas bagian mana yang merupakan ide original Anda, mana yang diambil dari referensi.
 - Untuk menghindari kolusi, adalah tanggung jawab Anda untuk memastikan bahwa Anda sepenuhnya jelas tentang sejauh mana kolaborasi/kerja kelompok diizinkan, dan bagian mana dari pekerjaan itu harus Anda kerjakan sendiri.

- Tidak boleh memasukkan apa pun dalam referensi atau bibliografi yang sebenarnya tidak direferensikan.
- Jika akses ke sumber utama tidak diperoleh, boleh menggunakan teks sekunder.
- Sitasi (menyebutkan) referensi harus diikuti dengan identifikasi pengutipannya dalam paragraf.

Kecurangan akademik dalam pelaksanaan praktikum

Tugas pendahuluan harus dikerjakan sendiri dalam setiap aspeknya, baik apabila tugas berupa analisis, perhitungan, atau simulasi. Kegiatan mencontoh atau meniru tugas pendahuluan tidak diperkenankan, dan apabila terbukti/bisa dibuktikan dapat dianggap melakukan kecurangan akademik seperti halnya mencontek. Apabila tugas yang diberikan membutuhkan referensi dari buku, internet dan sejenisnya, berlaku aturan plagiarisme. Untuk menghindari plagiarisme dalam mengerjakan tugas pendahuluan yang membutuhkan referensi, gunakan minimal 3 referensi dengan melakukan elaborasi dari referensi-referensi tersebut. Hindari dalam menggunakan hanya satu referensi meskipun dengan melakukan parafrase.

Tes awal termasuk dalam kategori yang sama dengan kuis atau **ujian**, dimana segala bentuk upaya mendapatkan bantuan dari pihak luar (mencontek pekerjaan peserta lain dengan bekerjasama atau tidak, menerima bantuan melalui alat komunikasi, memakai joki, dsb) dan menggunakan metode diluar yang diperkenankan (memakai contekan: melalui catatan, smartphone, dsb) adalah terlarang dan merupakan pelanggaran akademik.

Laporan praktikum sebagaimana laporan teknis, makalah, dan buku TA termasuk dalam kategori karya ilmiah, sehingga definisi dan aturan mengenai plagiarisme berlaku. Kecurangan yang biasa dilakukan diantaranya menggunakan data dari peserta lain, menggunakan template laporan peserta lain dan hanya mengganti datanya dan melakukan parafrase isi laporan yang lain.

PERCOBAAN I

PARAMETER GERBANG LOGIKA

1.1 TUJUAN

Mengenal dan memahami beberapa karakteristik dari gerbang logika diantaranya *voltage transfer*, *noise margin*, dan *propagation delay*.

Mengenal dan memahami parameter dari gerbang logika yaitu *operating point* yang merepresentasikan *range* logika HIGH dan LOW.

Dapat membuat rangkaian kombinasional sederhana menggunakan IC logika CMOS.

1.2 PERSIAPAN

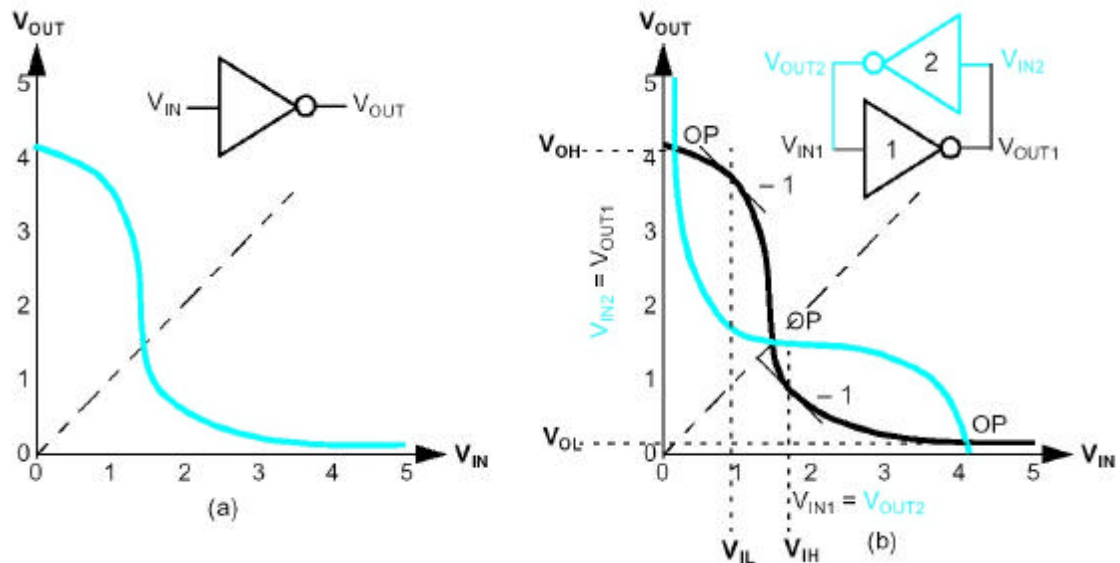
Bacalah appendix yang ada pada buku petunjuk praktikum ini dan bahan kuliah yang berkaitan, bagi yang mendapatkan Praktikum Rangkaian Elektrik baca kembali Percobaan 1 tentang Instrumentasi Laboratorium. Kerjakan **Tugas Pendahuluan** dan kumpulkan sesuai ketentuan yang berlaku.

1.3 DASAR TEORI

KARAKTERISTIK VOLTAGE TRANSFER

Karakteristik ***static voltage transfer*** dari sebuah gerbang logika adalah plot dari tegangan keluaran gerbang logika V_{OUT} dibandingkan dengan tegangan masukan gerbang logika V_{IN} .

Secara matematis kita bisa mendeskripsikan karakteristik *voltage transfer* sebagai $V_{OUT} = f(V_{IN})$. Istilah statik digunakan disini karena kita tidak memperhitungkan faktor waktu yang diantaranya adalah waktu tunda pada gerbang logika. Gambar 1(a) memperlihatkan ***static voltage transfer*** dari gerbang *inverter* dengan tegangan catu daya sebesar $V_{CC}=5V$.



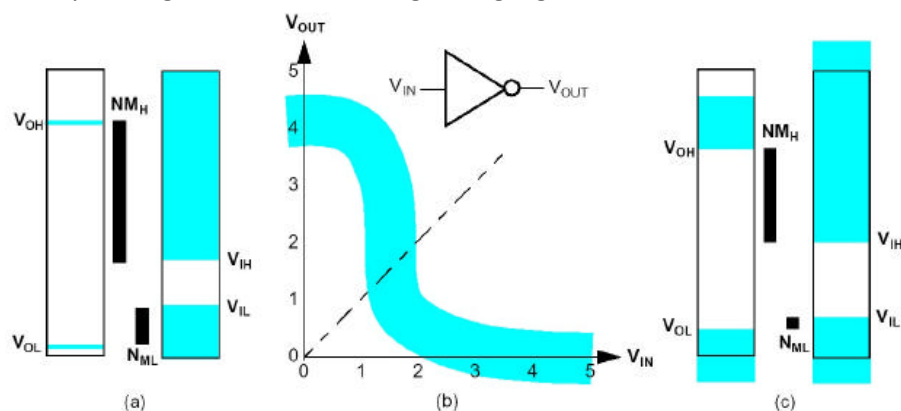
Gambar 1: (a) Karakteristik voltage transfer dan (b) operating points

Dari karakteristik *voltage transfer* kita bisa mendapatkan beberapa hal, yang pertama adalah **operating point**.

Operating point merupakan nilai tegangan keluaran yang dihasilkan oleh gerbang logika yang bisa diidentifikasi sebagai keluaran bernilai LOW atau bernilai HIGH. Karena tegangan keluaran bergantung pada tegangan masukan maka untuk mendapatkan nilai HIGH *operating point* secara utuh untuk keluaran inverter, nilai LOW *operating point* harus menjadi masukan inverter. Begitu pula sebaliknya, sehingga diperlukan konfigurasi umpan balik atau yang menyerupai.

Kemudian yang kedua adalah kita bisa mendapatkan nilai **noise margin**. **Noise/derau** didefinisikan sebagai tegangan efektif dari satu atau lebih masukan gerbang logika yang ditambahkan atau dikurangi terhadap tegangan normal. Tegangan normal adalah tegangan titik operasi yang stabil.

Noise margin didefinisikan sebagai jumlah dari tegangan derau efektif yang bisa ditoleransi oleh input tanpa mengubah nilai keluaran gerbang logika.



Gambar 2: Noise margin karakteristik transfer voltage gerbang logika

Untuk mendapatkan nilai **noise margin**, kita memerlukan dua nilai tegangan yang didapatkan dari grafik karakteristik transfer yaitu dua tegangan input yang memiliki **gradient = -1** seperti

yang ditandai pada **Gambar 1**. Tegangan yang lebih rendah dari kedua tegangan ini disebut **V input LOW** yang dituliskan V_{IL} dan yang lebih tinggi disebut **V input HIGH** yang dituliskan V_{IH} . Kedua tegangan ini merupakan tegangan perkiraan yang dianggap sebagai tegangan batas yang masih dikenali sebagai jenis masukan logika HIGH atau LOW.

Dengan menggunakan tegangan ini beserta tegangan V_{OH} dan V_{OL} kita bisa mendapatkan *static voltage noise margin* untuk gerbang logika. Untuk *LOW noise margin* dirumuskan:

$$NM_L = V_{IL} - V_{OL}$$

sedangkan *HIGH noise margin* dirumuskan:

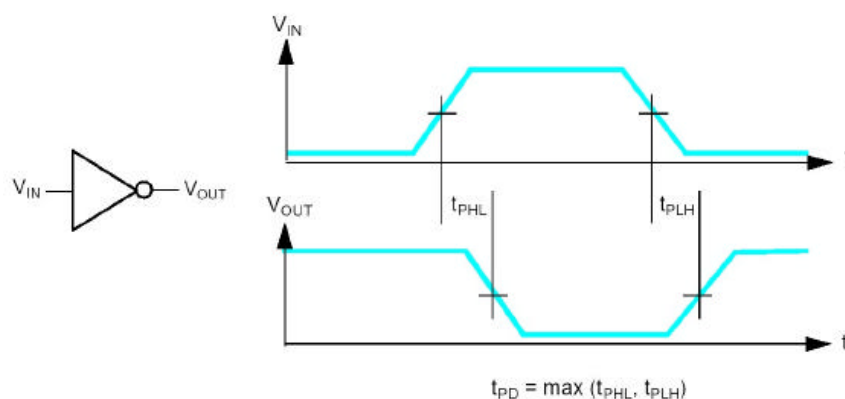
$$NM_H = V_{OH} - V_{IH}$$

Dari semua hal diatas, kita akan bisa menyimpulkan apakah yang disebut dengan nilai logika LOW dan logika HIGH baik untuk masukan maupun keluaran.

GATE DELAY

Dalam penjelasan berikut akan dibahas dua parameter gate delay yang penting. Untuk mendefinisikan parameter ini, kita akan menggunakan *inverter* sebagai contoh. Kita akan mengasumsikan sebuah pulsa diberikan kepada masukan *inverter* V_{IN} seperti pada Gambar 3. Respon terhadap pulsa ini pada keluaran *inverter* adalah V_{OUT} yang bisa dilihat pula pada Gambar 3.

Dua parameter yang akan dijelaskan tersebut dinamakan **high to low propagation time** (t_{PHL}) dan **low to high propagation time** (t_{PLH}). Pengukuran kedua parameter ini dilakukan pada **posisi 50% tegangan maksimal** dari bentuk gelombang V_{IN} dan V_{OUT} seperti yang terlihat pada Gambar 3.



Gambar 3: Definisi parameter *gate delay*

Pada kasus rangkaian dimana bentuk gelombang keluaran sama dengan gelombang masukan t_{PHL} adalah waktu yang diukur dari level tegangan ini ketika *falling input waveform* hingga *falling output waveform*, sedangkan t_{PLH} diukur dari level tegangan ini ketika *rising input waveform* hingga *rising output waveform*.

Perhatikan bahwa *subscript* pada parameter ini mencerminkan arah perubahan tegangan dari sinyal keluaran. Sebagai tambahan kita akan mendefinisikan parameter kedua yaitu **worst case propagation delay** yang dirumuskan:

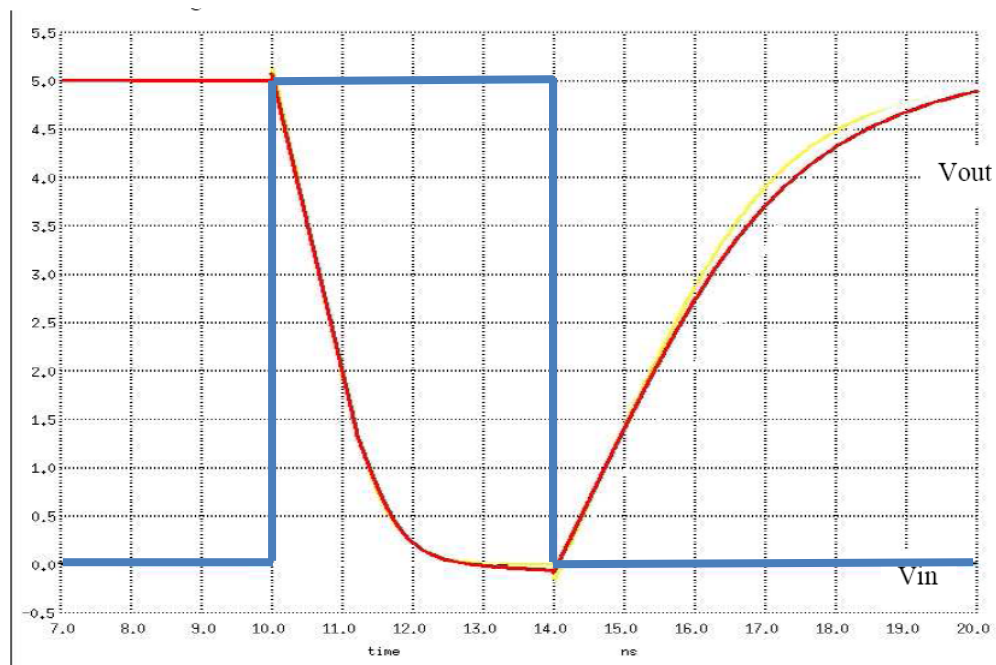
$$t_{PD} = \text{maximum}(t_{PHL}, t_{PLH}).$$

Patut diperhatikan bahwa tingkat 50% yang kita gunakan disini bukan sesuatu yang umum dalam pengukuran *delay*. Untuk $t_{PD(\text{average})}$ kita akan merumuskannya sebagai **nilai rata-rata** dari t_{PHL} dan t_{PLH} yang dirumuskan:

$$t_{PD(\text{average})} = (t_{PHL} + t_{PLH})/2.$$

1.4 TUGAS PENDAHULUAN

1. Cari dan bacalah datasheet dari semua IC yang digunakan pada percobaan ini terutama posisi kaki dan karakteristiknya. Sebutkan perbedaan yang mendasar dari IC rangkaian logika, antara yang berbasis TTL dan CMOS.
2. a. Desain dan susunlah gerbang AND, OR, NAND, dan NOR menggunakan transistor PMOS!!
b. Jelaskan mengapa pada perancangan digital gerbang NOR dan NAND lebih disukai dibandingkan menggunakan gerbang lainnya?
3. Untuk rangkaian logika, sering dibuat hubungan langsung output suatu gerbang dengan input gerbang lain (feeding/driving). Sebutkan dan jelaskan batasan-batasan dalam melakukan hal ini!
4. Analisis gambar berikut:



- a. Hubungan input-output grafik diatas mensimulasikan rangkaian apa? Bagaimana penjelasan anda?
- b. Berapa nilai t_{PLH} , t_{PHL} , *rise time*, dan *fall time*? Tunjukkan pada gambar diatas posisi anda mendapatkan nilai tersebut!

1.5 PERCOBAAN

PERALATAN YANG DIGUNAKAN

- Kit praktikum Gerbang Logika NOR TTL dan Parameter Gerbang Logika
- 1 buah project board
- Power Supply, Osiloskop dan Generator Sinyal
- Komponen IC gerbang logika 7400
- Osiloskop dan Generator Sinyal
- Kabel jumper secukupnya
- 1 buah Kabel BNC-BNC, 2 buah kabel BNC-Probe Kait / BNC-Jepit Buaya / BNC-Banana
- 2 buah kabel Banana-Banana / Banana-Jepit Buaya merah dan hitam.

PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan semua alat dan bahan sudah disiapkan
2. Perhatikan datasheet tiap-tiap IC yang digunakan pada modul ini, amati setiap pin pada IC tersebut (letak VCC, GND, dan kaki *input/output* **Bisa dilihat di Appendix A**).
3. Periksa catu daya sebelum diberikan terhadap rangkaian, sesuaikan dengan TTL yang dibutuhkan yaitu **+5VDC**. **Kerusakan komponen akibat tegangan yang tidak sesuai atau akibat kesalahan letak input/output menjadi tanggung jawab praktikan!!!**
4. Periksa pemasangan IC pada rangkaian dengan mengukur kaki tegangan catu daya(+5V dan GND)
5. Periksa kabel-kabel dan konektor, gunakan multimeter untuk melakukannya

Pada saat praktikum berlangsung, praktikan hendaknya memperhatikan hal-hal berikut ini:

1. **Matikan catu daya** pada saat merangkai atau mengubah rangkaian dan mengganti IC
2. Periksa nilai **VCC** dan **GROUND** yang akan diberikan ke pin IC.

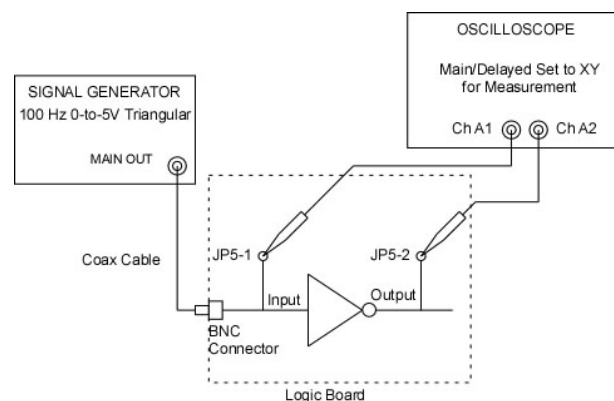
PERCOBAAN 1A: VOLTAGE TRANSFER CHARACTERISTIC DAN NOISE MARGINS DARI IC 74LS04

Pada percobaan ini kita akan mencari karakteristik *transfer voltage* dari sebuah inverter 74LS04 dan inverter CMOS 4007.

PROSEDUR PERCOBAAN:

1. Gunakan kit praktikum Parameter Gerbang Logika Percobaan 1A, 1B

- Setting keluaran generator sinyal menjadi sinyal segitiga dengan frekuensi maksimal **1KHz** dan tegangan *peak-to-peak* **5V**, gunakan **offset DC dengan menarik knop OFFSET keluar** terlebih dahulu dan memutarnya sehingga dihasilkan tegangan minimum keluaran adalah **0V**. Gunakan port **OUTPUT** sebagai keluaran **bukan port TTL/CMOS**. Cek keluaran sinyal generator menggunakan **osiloskop dengan mode coupling DC sebelum menyambungkannya dengan inverter karena dapat merusak IC**.
- Sambungkan output generator sinyal ke input gerbang logika (IN).
- Sambungkan kanal 1 osiloskop dengan input gerbang logika (IN).
- Sambungkan kanal 2 osiloskop dengan output gerbang logika (OUT)
- Setting power supply pada tegangan 5V dan sambungkan dengan VCC dan GND.
- Catat hasil pengukuran input dan output dalam mode *dual-trace* BCL
- Setting osiloskop dengan mode X-Y. Sebelum melakukan pengamatan atur posisi sinyal pada mode X-Y dengan menekan tombol GND pada kedua kanal masukan hingga terlihat 1 titik kecil, tempatkan titik yang terlihat pada tengah osiloskop/sumbu koordinat (**Jangan terlalu lama pada bentuk titik ini!!**). Setelah itu tekan tombol GND kembali untuk pengamatan bentuk sinyal.
- Lihat keluaran osiloskop, apakah bentuknya mirip dengan gambar referensi ataukah ada perbedaan. Tulis hasil dan langkah yang anda kerjakan pada BCL anda. Cantumkan gambar yang didapat pada laporan anda dan jelaskan yang bisa anda analisa dari gambar tersebut.
- Catat hasil percobaan pada BCL anda.



Gambar 5 : Bentuk rangkaian untuk percobaan 1a (*nilai sinyal ikuti petunjuk praktikum*)

PERCOBAAN 1B : Mencari Nilai NM_L dan NM_H

Pada percobaan ini kita akan mencari karakteristik *static noise margin* dari sebuah IC-74LS04 dan inverter CMOS 4007

PROSEDUR PERCOBAAN:

1. Gunakan kit praktikum Parameter Gerbang Logika Percobaan 1A, 1B
2. Gambarkan kembali pada BCL anda keluaran mode XY dari percobaan sebelumnya pada tempat yang terpisah.
3. Lakukan langkah berikut untuk inverter TTL 74LS04
4. Tukarkan posisi probe osiloskop kanal 1 dengan kanal 2 sehingga posisinya bertukar dari percobaan 1 (kanal 1 terhubung dengan output IC dan kanal 2 dengan input IC).
5. Sama seperti percobaan 1 dapatkan sinyal keluaran inverter dalam mode XY.
6. Kemudian gambarkan pula sinyal tersebut secara manual pada bidang gambar yang sama pada langkah 1 sehingga kedua gambar akan saling bertumpukan dan membentuk seperti pada gambar 1.
7. Pada laporan anda cantumkan gambar yang didapat dan tunjukkan pada gambar serta hitung nilai-nilai berikut berdasarkan hasil pengamatan anda:
8. Nilai dan posisi V_{OL} , V_{OH} , V_{IL} , dan V_{IH} dengan ketelitian 1 desimal (**lihat referensi gambar 1**)
9. Nilai NM_H dan NM_L yang anda dapatkan dari percobaan berdasarkan rumus yang sudah diberikan dan bandingkan dengan nilai yang tertera pada datasheet.
10. Catat hasil percobaan pada BCL anda. Apa yang dapat anda simpulkan pada percobaan ini?
11. Ulangi percobaan 1A Langkah 2-9 dan percobaan 1B langkah 4-10 untuk inverter CMOS 4007.

PERCOBAAN 1C : DELAY PROPAGASI

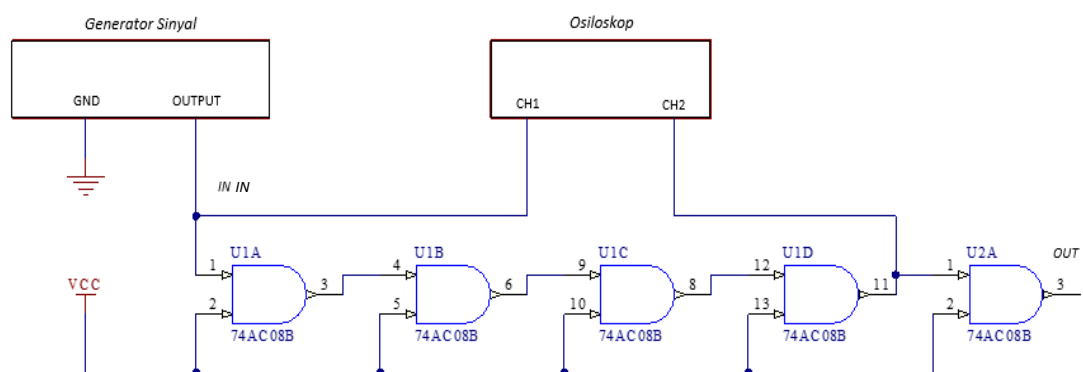
Dalam percobaan delay propagasi ini, kita akan menggunakan gerbang logika AND 2 masukan (IC 7408). Karena keterbatasan kemampuan osiloskop maka kita akan menggunakan konfigurasi 4 buah gerbang yang disajikan.

Dengan konfigurasi ini hasil delay propagasi yang didapatkan **harus dibagi empat** terlebih dahulu untuk mendapatkan nilai sebenarnya.

PROSEDUR PERCOBAAN:

1. Gunakan kit praktikum Parameter Gerbang Logika Percobaan 1C
2. Susunlah rangkaian seperti pada gambar 6 dibawah dengan kondisi seluruh alat dimatikan
3. Kemudian sambungkan power supply dengan VCC dan GND kit praktikum

4. Nyalakan power supply
5. Ubah setting *triggering* menggunakan tombol slope menjadi *positive edge*.
6. Setting setiap kanal input menjadi 1V/DIV. Sambungkan *ground channel* 1 dan *channel* 2 dan setting TIME/DIV ke posisi terendah osiloskop yaitu 0.2 us.
7. Setting keluaran generator sinyal menjadi sinyal kotak dengan frekuensi 600KHz jika menggunakan osiloskop jenis 622G. Untuk osiloskop jenis GOS 6050 atau osiloskop digital gunakan frekuensi 300KHz. Gunakan port **OUTPUT** sebagai keluaran. **Cek keluaran sinyal generator menggunakan osiloskop sebelum menyambungkannya dengan Gerbang logika karena dapat merusak IC apabila salah!!!**
8. Tampilkan keluaran dari kedua kanal sehingga bentuk pulsa pada saat naik pada kanal 1 dan kanal 2 bisa diamati secara utuh.
9. Apabila sinyal output yang dihasilkan terlalu kecil (rentang mV), gunakan tombol X1/MAG untuk memperbesar hasil yang didapatkan, kemudian tekan tombol x5-x10x20 dan perbesar hingga 10x agar lebih terlihat jelas.
10. Atur posisi vertikal kedua sinyal sehingga posisi 50% berada di sumbu X (Nilai sinyal diatas dan dibawah sumbu X pada masing-masing kanal sama).
11. Gambarkan atau foto hasil yang didapatkan.
12. Ubah setting triggering menjadi *negative edge* dan ulangi semua langkah diatas.
13. Gunakan nilai t_{PLH} dan t_{PHL} yang didapatkan untuk mencari t_{PD} dan $t_{PD(average)}$ menggunakan rumus yang telah diberikan sebelumnya.
14. Baca datasheet dari 74LS08, kemudian bandingkan t_{PD} dan $t_{PD(average)}$ yang didapatkan pada percobaan dengan rentang nilai yang tertulis pada datasheet dan jelaskan alasannya apabila ada perbedaan hasil yang didapat.



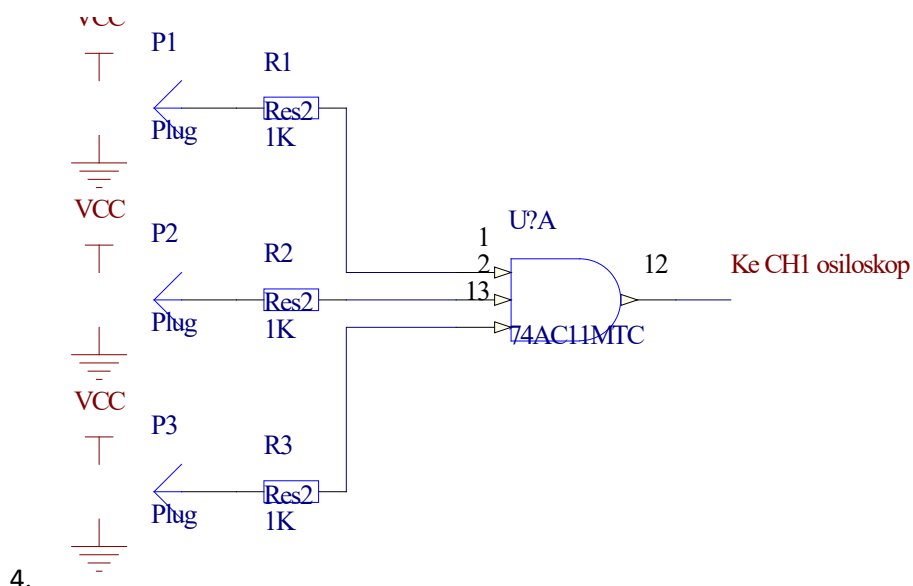
Gambar 6 : Bentuk rangkaian untuk percobaan1c

PERCOBAAN 1D : VERIFIKASI FUNGSI LOGIKA

Sebelumnya anda akan diberikan sebuah IC logika yang nomor serinya sudah disamarkan. Tujuan dari percobaan ini adalah untuk mencari jenis IC logika yang digunakan berdasarkan hubungan input-output yang terukur. IC yang digunakan memiliki 3 input, lihatlah datasheet IC logika CMOS 3 input apa saja untuk verifikasi posisi pin karena semuanya memiliki posisi pin yang sama.

PROSEDUR PERCOBAAN:

1. Gunakan kit praktikum Parameter Gerbang Logika Percobaan 1D.
2. Gunakan salah satu kanal masukan osiloskop atau multimeter untuk mengukur tegangan keluaran dari gerbang logika yang akan diukur pada pin OUT.
3. Buatlah tabel logika dari gerbang yang dipakai dengan bervariasikan ketiga masukan gerbang logika menggunakan tegangan dari power supply. Untuk logika High gunakan Vcc power supply yang diset bernilai **5V**, sedangkan untuk logika LOW gunakan ground power supply.



Gambar 7 : Bentuk rangkaian untuk percobaan 1d

OBSERVASI:

Jawab pertanyaan berikut:

1. Apakah fungsi logika dari gerbang $[Y=f(A,B,C)]$? Jelaskan bagaimana anda mendapatkannya dari bentuk pulsa yang terlihat.
2. Catat semua hasil percobaan pada BCL anda.

PERCOBAAN 1E : RANGKAIAN KOMBINASIONAL SEDERHANA

Dalam percobaan ini anda akan mengkonversikan suatu persamaan logika ke bentuk lainnya

PROSEDUR PERCOBAAN:

1. Buatlah persamaan logika : $Q = A + B$, menjadi persamaan yang hanya memuat operasi **NAND** atau **NOR** saja.
2. Rancang dan gambarkan rangkaiannya pada BCL anda, kemudian buat rangkaiannya dari IC CMOS 7400 yang tersedia pada kit breadboard (pinout ada di Apendiks A).
3. Verifikasi fungsionalitas rangkaian anda dengan memberikan kombinasi berbagai input yang mungkin, catat dan bandingkan hasilnya dengan tabel kebenaran yang anda harapkan.
4. Dari percobaan ini apa yang dapat anda simpulkan?
5. Catat semua hasil percobaan pada BCL anda.

PERCOBAAN 1F : GERBANG LOGIKA NOR TTL

1. Gunakan kit praktikum Gerbang Logika NOR TTL
2. Hubungkan VCC dan GND ke power supply 5 V, hubungkan multimeter pada terminal OUT untuk mengukur tegangan.
3. Berikan input IN A, IN B, IN C logika 0 (tegangan 0V), baca tegangan pada OUT. Nilai logika apakah yang terbaca? Baca dan catat nilai tegangan di seluruh simpul rangkaian (tidak termasuk input dan power supply)
4. Ubah salah satu nilai input menjadi logika 1 (tegangan 5V), baca tegangan pada OUT. Nilai logika apakah yang terbaca? Baca dan catat nilai tegangan di seluruh simpul rangkaian (tidak termasuk input dan power supply). Ulangi kembali untuk seluruh susunan yang mungkin.
5. Ubah dua nilai input menjadi logika 1 (tegangan 5V), baca tegangan pada OUT. Nilai logika apakah yang terbaca?. Catat nilai tegangan seperti sebelumnya dan ulangi kembali untuk seluruh susunan yang mungkin.
6. Ubah semua nilai input menjadi logika 1 (tegangan 5V), baca tegangan pada OUT. Nilai logika apakah yang terbaca?. Catat nilai tegangan seperti sebelumnya.

1.6 MENAKHIRI PERCOBAAN

1. Sebelum keluar dari ruang praktikum, rapikan meja praktikum. **Rapikan kabel dan matikan komputer, osiloskop, generator sinyal, dan power supply DC.** Cabut daya dari jala-jala ke kit FPGA dan letakkan kembali pada tempat semula.
2. Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau merapikan meja ketika praktikum berakhir akan mendapatkan potongan nilai sebesar minimal 10.
3. **Pastikan asisten telah menandatangani catatan percobaan kali ini** pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

PERCOBAAN II

Pengenalan Desain Menggunakan FPGA

1.1 Tujuan

1. Mempelajari teknik perancangan rangkaian digital dengan target FPGA.
2. Dapat melakukan perancangan rangkaian digital dengan target FPGA baik menggunakan pendekatan skematik maupun bahasa VHDL.

1.2 Persiapan

Sebelum praktikum ini dilaksanakan praktikan wajib membaca referensi berikut(bisa didapat di web lab dasar:<http://labdasar.ee.itb.ac.id>):

- Buku manual board ALTERA DE1 yang bisa diambil di web labdasar.
- Buku pegangan mata kuliah Sistem Digital Anda mengenai persamaan Boolean dan rangkaian aritmatika khususnya Full Adder beserta bentuk-bentuk implementasinya.
- Teori bahasa VHDL tentang **entity, architecture, component, signal** (lihat **Appendix G**).

Pelajari sekilas mengenai software Quartus® dan Modelsim®!

1.3 Dasar Teori

FPGA

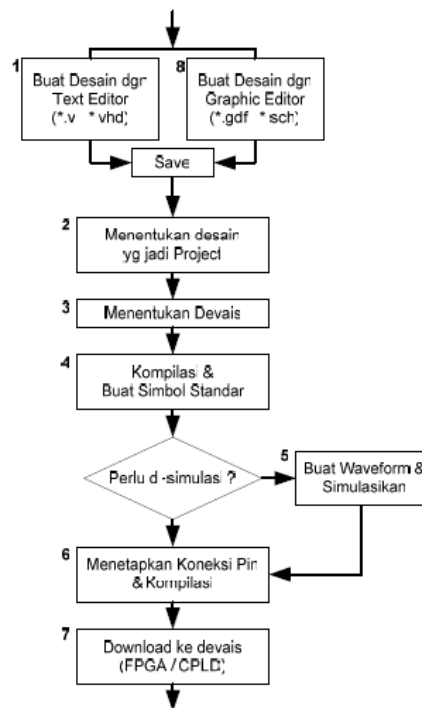
Secara umum alur perancangan rangkaian digital dengan menggunakan FPGA dari ALTERA dapat digambarkan seperti flowchart pada Gambar 1.

Full Adder

Keunggulan FULL-ADDER bila dibandingkan dengan HALF-ADDER adalah kemampuan-nya menampung dan menjumlahkan bit CARRY-in (Cin) yang berasal dari CARRY-out (Cout) dari tahapan sebelumnya. Oleh karenanya fungsi FULL ADDER itu sendiri adalah menjumlahkan ketiga bit input yaitu bit A, bit B dan Cin untuk menghasilkan dua bit output yaitu S dan Cout.

Dengan menginterpretasikan fungsi dan melihat format operasi rangkaian FULLADDER, tabel kebenaran dapat disusun untuk setiap kemungkinan kombinasi ketiga bit input. Diasumsikan input berasal dari sumber logika positif dan output berupa ACTIVE HIGH.

Langkah selanjutnya adalah membuat K-Map orde 2 dari tabel kebenaran tersebut. KMap ini akan membantu merumuskan fungsi logika dari S dan Cout.



Gambar 1. Flowchart umum proses perancangan

Tabel 1. Tabel kebenaran dan K-map dari FULL ADDER

$$S = C_{in} (A \oplus B) + C_{in} (A \oplus B)$$

$$= A \oplus B \oplus C_{in}$$

$$C_{out} = C_{in} (A \oplus B) + (A \cdot B)$$

| A | B | C_{in} | S | C_{out} |
|---|---|----------|---|-----------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

(a)

B

| | | |
|-------|---------------------|---------------------|
| A \ B | 0 | 1 |
| 0 | C_{in} | $\overline{C_{in}}$ |
| 1 | $\overline{C_{in}}$ | C_{in} |

S

B

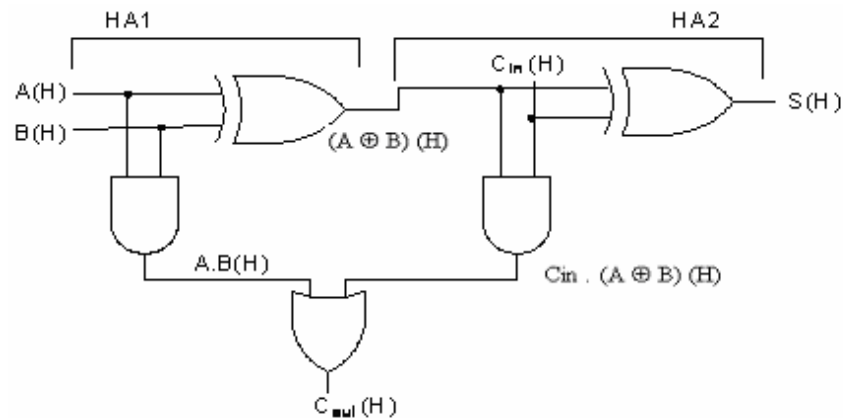
| | | |
|-------|----------|----------|
| A \ B | 0 | 1 |
| 0 | 0 | C_{in} |
| 1 | C_{in} | 1 |

C_{out}

(b)

Implementasikan rangkaian FULL-ADDER dibuat berdasarkan persamaan ekspresi logika di atas. Rangkaian ini dapat tersusun dari dua buah HALF-ADDER (HA1 dan HA2), seperti terlihat pada **Gambar 2**.

Untuk penjumlahan dengan jumlah bit yang lebih banyak, dapat dilakukan dengan menambahkan rangkaian HALF ADDER, sesuai dengan jumlah bit input. Di pasaran, rangkaian FULL ADDER sudah ada yang berbentuk IC, seperti 74xx83 (4-bit FULL ADDER).



Gambar 2. Salah satu bentuk rangkaian Full Adder

Terdapat beberapa jenis rangkaian FULL ADDER, yaitu PARALLEL ADDER, LOOK AHEAD CARRY ADDER, dan CARRY SAVE ADDER dimana masing-masing memiliki kelebihan dan kekurangannya.

1.4 TUGAS PENDAHULUAN

1. Jelaskan hubungan dari transistor, BJT, MOSFET, IC, ASIC, PLA, PAL, CPLD, FPGA, prosesor, mikrokontroller, DSP prosesor, dan SoC (System on Chip)! Buatlah dalam bentuk 1 halaman A4 essay!
2. Tuliskan spesifikasi FPGA **ALTERA DE1** yang akan digunakan dalam praktikum!
3. Siapkan skematik atau kode VHDL (sesuai arahan percobaan) untuk percobaan 2A hingga 2E, serta lakukan simulasi fungsional untuk seluruh rangkaian digital bersangkutan. Tambahkan *screenshot* hasil simulasi fungsional pada tugas pendahuluan anda!

1.5 PERCOBAAN

PERALATAN YANG DIGUNAKAN

- Komputer(PC) yang telah terinstal program Quartus II
- FPGA development board, tipe ALTERA DE1 beserta perlengkapannya yang meliputi:
 - Board FPGA tipe DE1.
 - Catu daya+ kabel dan konektor tambahan
 - Kabel *downloader* USB-Blaster.

PROSEDUR PRAKTIKUM

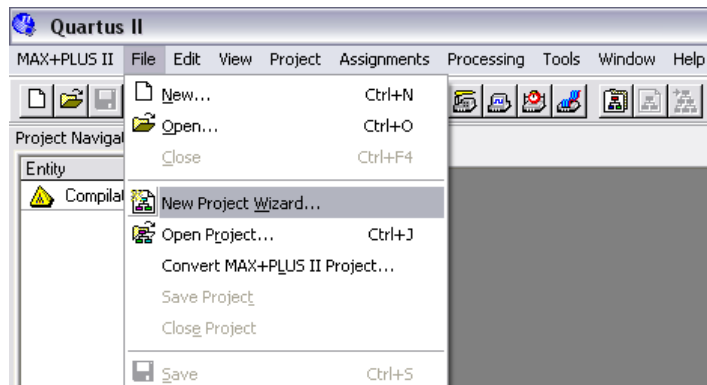
PERCOBAAN 2A: MENDESAIN *FULL ADDER* DENGAN SKEMATIK

Dalam percobaan ini kita akan mendesain full adder menggunakan FPGA dengan pendekatan skematik

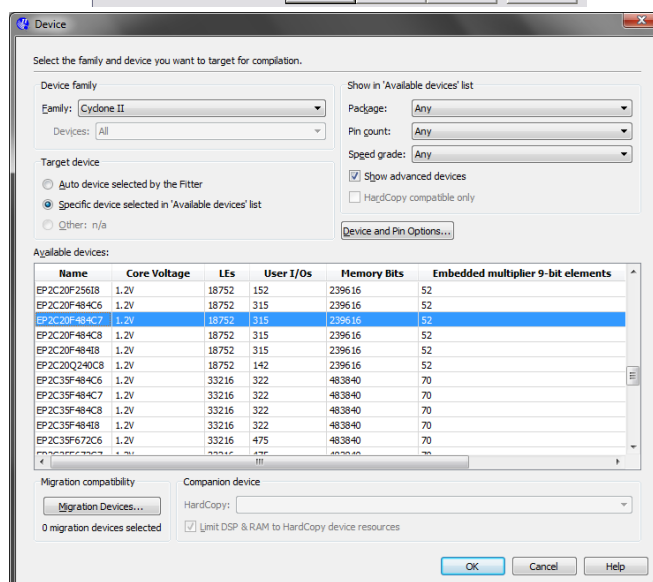
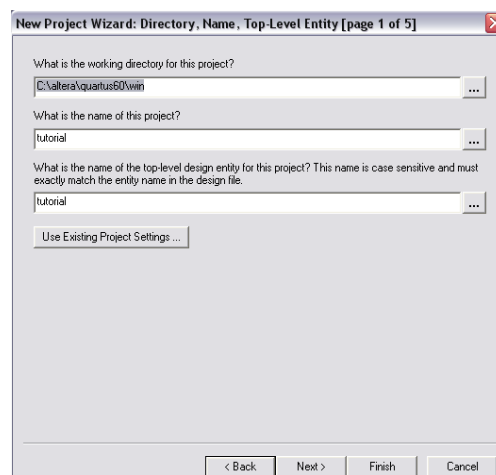
PROSEDUR PERCOBAAN:

a. Membuat Proyek Baru Menggunakan Quartus II 9.0 sp2 Web Edition

1. Buatlah folder baru di dalam folder PraktikumSisDig (jika belum ada buatlah folder tersebut), misalnya untuk kelompok2 folder yang dibuat **"D:\PraktikumSisDig\Kelompok2\Modul2\..."**
2. Kemudian pada folder tsb buatlah dua folder baru yang bernama **Tutorial1** dan **Tutorial2**.
3. Jalankan **Quartus II 9.0 sp2 Web Edition**.
4. Lihat Gambar 3 untuk melihat ilustrasi langkah-langkah berikutnya pada prosedur(a) ini.
5. Klik **File → New Project Wizard** seperti yang terlihat pada **Gambar3(a)**. Setelah ini akan tampil jendela **Introduction**, Klik **Next**.
6. Pada langkah ini akan terlihat jendela seperti **Gambar 3(b)**. Pada kolom paling atas (terkait direktori untuk project yang sedang Anda buat), tekan tombol **"..."** yang ada di sebelah kanan kemudian carilah folder **Tutorial1** yang sudah Anda buat sebelumnya. Akhiri dengan tekan tombol **Open**.
7. Kemudian pada kolom berikutnya (terkait nama project) ketikkan **"Tutorial1"**. Pastikan pada kolom ketiga (terkait top level entity) terisi nama yang sama.
8. Klik **Next** untuk sampai ke jendela **"Add Files"**, lewati jendela ini dengan klik **Next** kembali
9. Pada langkah ini akan terlihat jendela seperti **Gambar 3(c)**, pada daftar **"Family"** untuk yang mendapatkan **board DE1** untuk **"Family"** pilih **CycloneII**, kemudian dalam bagian device pilih **EP2C20F484C7**. Setelah itu klik **Finish** karena untuk langkah berikutnya kita hanya menggunakan setting default.



(a)

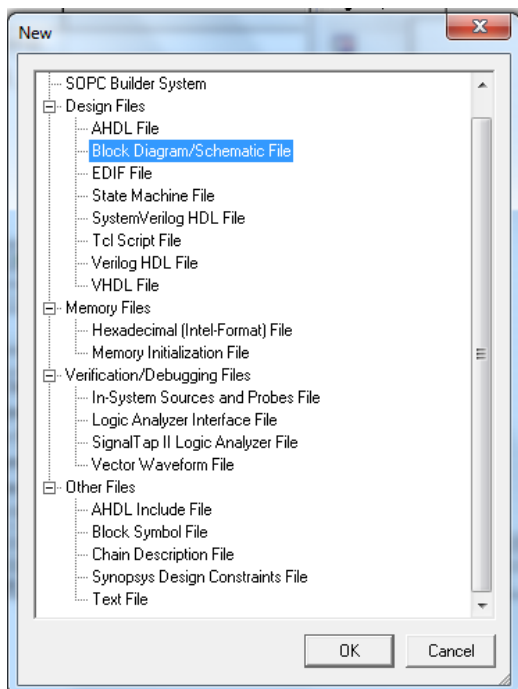


(b)

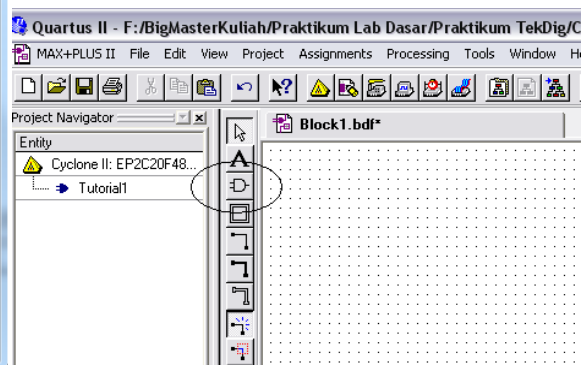
Gambar 3. Tampilan langkah petunjuk a

b. Memilih dan Menempatkan komponen

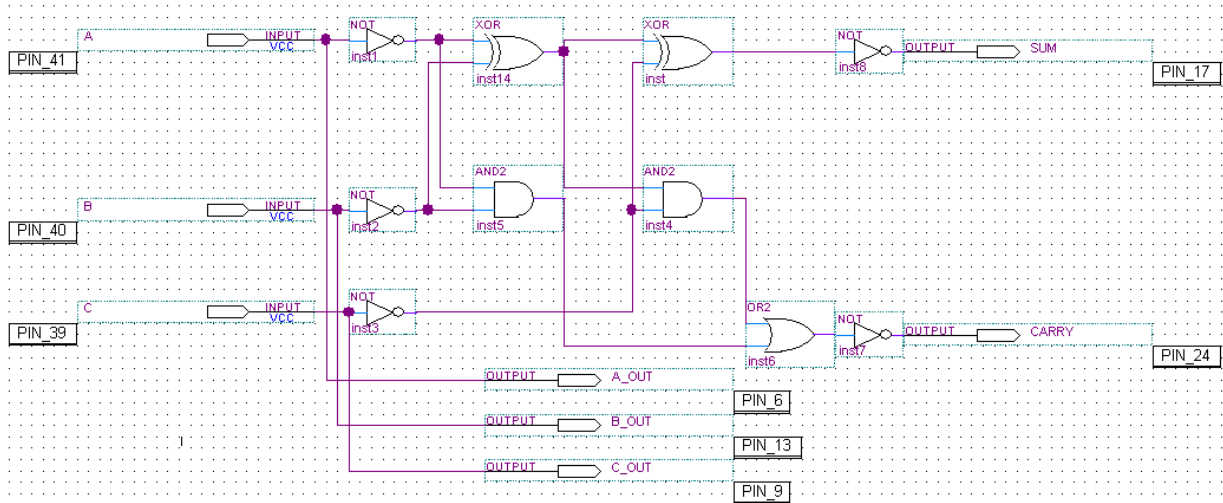
1. Klik **File** → **New**, pada jendela yang tampil pilih **Block Diagram/Schematic File** sebagai pilihan desain dan klik **OK**. Simpan file tersebut sebagai **Tutorial1.bdf** seperti pada **Gambar 4(a)**.
2. Pilih **File** → **Page Setup** dan pilih **Letter** sebagai ukuran kertas, klik **OK**.
3. Buka jendela **Symbol Tools** dengan mengklik tombol dengan ikon gerbang AND pada bagian kiri jendela schematic editor seperti bagian yang dilingkari pada **Gambar 4(b)**.
4. Cari komponen **XOR** pada folder **..\primitives\logic** dan klik dua kali nama komponen tsb atau klik **OK**. Di ujung panah mouse akan muncul gambar komponen **XOR** dengan 2 masukan. Cari posisi yang tepat pada skematik dan klik 1 kali pada posisi itu untuk menempatkan gerbang **XOR**. Untuk menyudahi tekan tombol **Esc** atau klik kanan dan pilih **cancel**.
5. Ulangi langkah diatas untuk menempatkan dua buah gerbang **AND** dengan 2 masukan dan sebuah gerbang **OR** dengan 2 masukan serta lima buah gerbang **NOT**.
6. Buka kembali jendela **Symbol Tools**, kali ini buka folder **..\primitives\Pin**.
7. Pilih jenis **Input Pin** dan tempatkan 3 buah pada skematik. Ulangi langkah ini untuk menempatkan 5 buah **Output pin** pada skematik. Posisikan (belum dihubungkan) sesuai dengan **Gambar 4(c)**.



(a)




(b)



(c)

Gambar 4. Skematik yang digunakan pada Percobaan 2a

c. Menambahkan hubungan untuk membentuk net

1. Pilih **Orthogonal Node Tool** pada bagian toolbar bagian kiri yang memiliki simbol , untuk menggambarkan kabel.
2. Arahkan ujung pointer mouse ke salah satu sisi yang akan dihubungkan lalu klik kiri dan tahan kemudian tarik garis hingga ujung lain yang diinginkan kemudian lepaskan tombol mouse Anda.
3. Lihat kembali **Gambar 4(c)** sebagai referensi penempatan kabel yang dibutuhkan.

d. Pelabelan Net dan pin I/O

1. Klik dua kali pada port input/output yang akan diubah namanya kemudian ubah nama dari pin sesuai dengan yang pada **Gambar 4(c)** ("A", "B", "C" untuk input dan "SUM", "CARRY", "A_OUT", "B_OUT", "C_OUT" untuk output).
2. Untuk **port** masukan biarkan *default value* sebagai VCC.

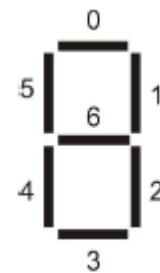
e. Menetapkan I/O pin pada kaki FPGA

1. Simpan skematik Anda kemudian pilih **Processing → Start → Start Analysis & Synthesis** atau **Ctrl+K** (Pastikan tidak ada error).
2. Pilih **Assignment → Pin Planner**.
3. Akan terbuka sebuah jendela baru dimana sebelah atas akan ada gambar FPGA dengan posisi kaki-kakinya dan di bawah ada daftar yang sudah berisi port input-output skematik kita seperti yang terlihat pada **Gambar 5**.

4. Klik **Direction** untuk mengurutkan pin.
5. Pada kolom **Location** double-klik kiri kolom yang sebaris dengan port yang ditinjau. Akan muncul suatu daftar kaki FPGA yang bisa dipakai.
6. Untuk percobaan ini, kita akan menggunakan switch untuk masukan dan LED pada 7-segment untuk keluaran. LED pada DE1 bersifat **active low**. Ketika **terbuka/tidak ditekan switch akan berlogika 1** karena ada rangkaian **pull-up** dan jika **tertutup/ditekan akan berlogika 0**, sedangkan LED akan **menyala** ketika mendapatkan **input LOW VOLTAGE** dan **mati** ketika mendapatkan **input HIGH VOLTAGE**.
7. Kita hanya memanfaatkan LED pada bagian a,g,dan d dari 7-segment dimana menyala berarti '1' dan mati berarti '0' (dalam bentuk biner bukan desimal!). Adapun nama pin yang terhubung dengan switch atau LED pada DE1 dapat dilihat pada table 2 di bawah ini: **(Untuk referensi lengkap lihat datasheet!)**

Tabel 2. Posisi kaki yang terhubung 7 segment dan switch untuk DE1

| Switch | Cyclone II Pin |
|-----------|----------------|
| Switch[0] | PIN_L22 |
| Switch[1] | PIN_L21 |
| Switch[2] | PIN_M22 |
| Switch[3] | PIN_V12 |
| Switch[4] | PIN_W12 |
| Switch[5] | PIN_U12 |
| Switch[6] | PIN_U11 |
| Switch[7] | PIN_M2 |
| Switch[8] | PIN_M1 |
| Switch[9] | PIN_L2 |

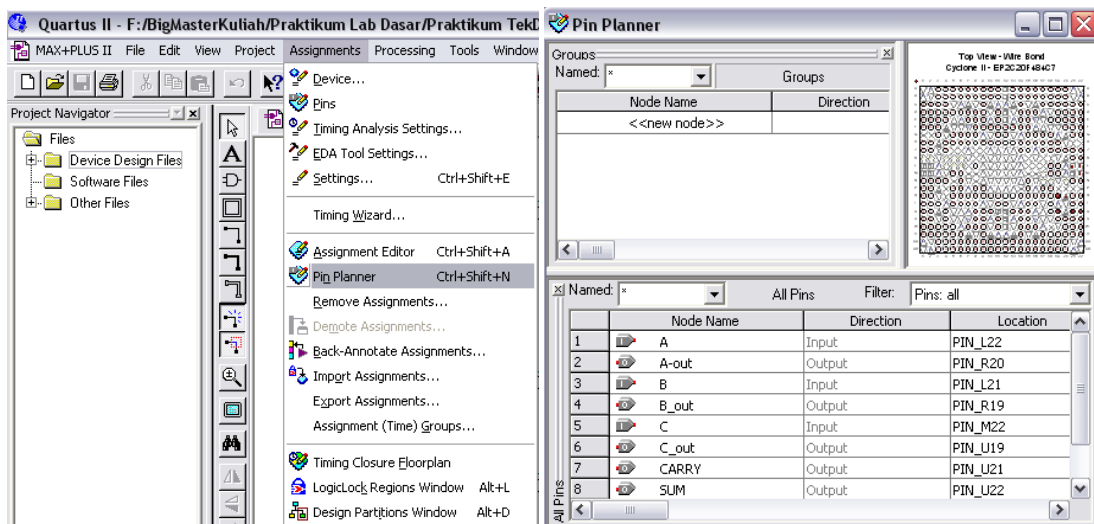


| Signal Name | FPGA Pin No. | Description |
|-------------|--------------|--------------------------|
| HEX0[0] | PIN_J2 | Seven Segment Digit 0[0] |
| HEX0[1] | PIN_J1 | Seven Segment Digit 0[1] |
| HEX0[2] | PIN_H2 | Seven Segment Digit 0[2] |
| HEX0[3] | PIN_H1 | Seven Segment Digit 0[3] |
| HEX0[4] | PIN_F2 | Seven Segment Digit 0[4] |
| HEX0[5] | PIN_F1 | Seven Segment Digit 0[5] |
| HEX0[6] | PIN_E2 | Seven Segment Digit 0[6] |
| HEX1[0] | PIN_E1 | Seven Segment Digit 1[0] |
| HEX1[1] | PIN_H6 | Seven Segment Digit 1[1] |
| HEX1[2] | PIN_H5 | Seven Segment Digit 1[2] |
| HEX1[3] | PIN_H4 | Seven Segment Digit 1[3] |
| HEX1[4] | PIN_G3 | Seven Segment Digit 1[4] |
| HEX1[5] | PIN_D2 | Seven Segment Digit 1[5] |
| HEX1[6] | PIN_D1 | Seven Segment Digit 1[6] |

8. Untuk pemasangan kaki komponen pada **Pin Planner** bisa dilihat pada referensi tabel 3 di bawah ini:

Tabel 3. Referensi kaki komponen

| Nama Pin I/O | Kaki yang digunakan DE1 |
|---------------------|--------------------------------|
| A | PIN_L22 |
| B | PIN_L21 |
| C | PIN_M22 |
| A_OUT | PIN_J2 |
| B_OUT | PIN_E2 |
| C_OUT | PIN_H1 |
| CARRY | PIN_D1 |
| SUM | PIN_E1 |




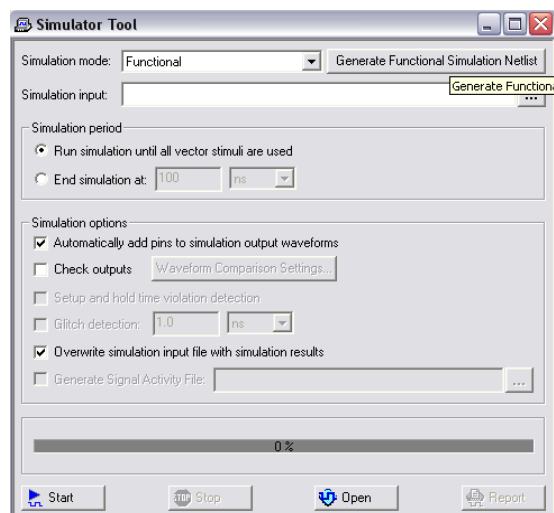
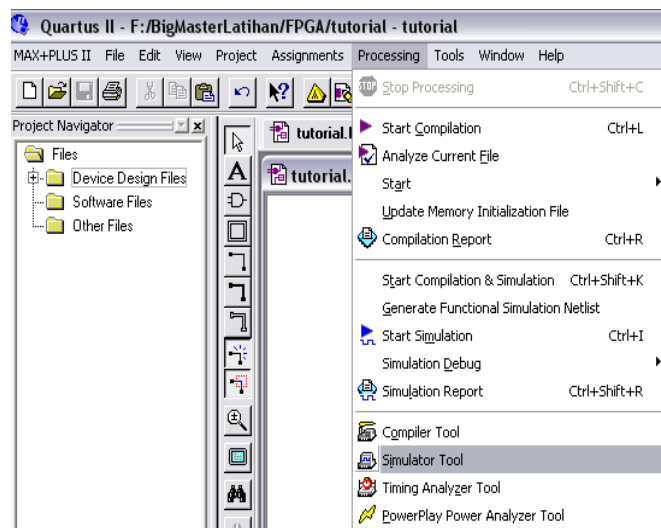
Gambar 5. Tampilan langkah petunjuk e

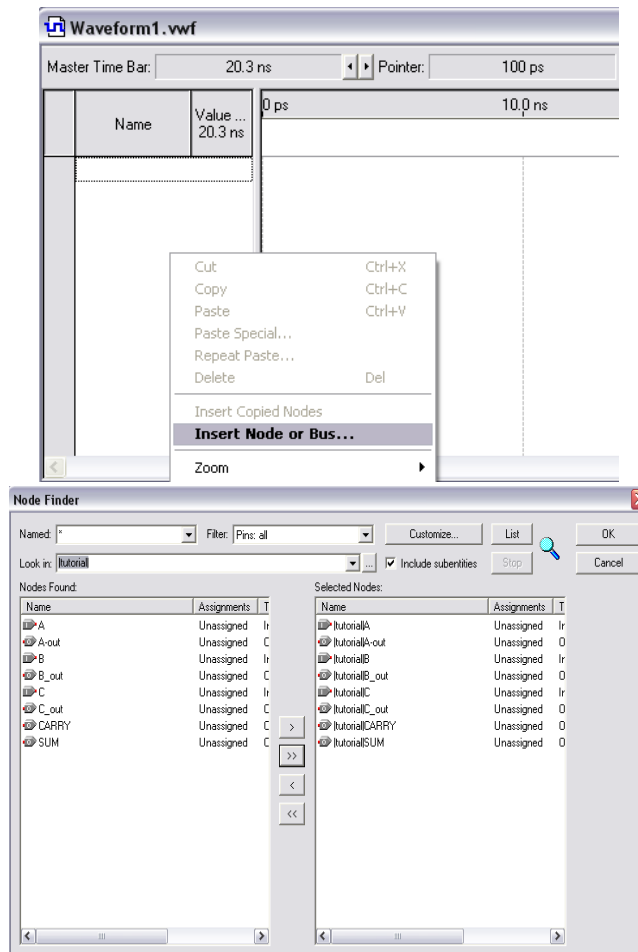
f. Pembuatan Netlist untuk simulasi

Untuk melaksanakan simulasi secara fungsional pada program ini diperlukan sebuah deskripsi netlist dari rangkaian. Langkah untuk membuatnya adalah sebagai berikut: (**Lihat Gambar 6 untuk petunjuk secara visual**)

1. Pilih **Processing** → **Simulator Tool**.
2. Pilih **Simulation Mode** menjadi **Functional**.
3. Klik pada tombol **Generate Functional Simulation Netlist** (Pastikan tidak ada error).
4. Klik pada check box di sebelah kiri **“Overwrite Simulation input file with simulation result”** agar setiap kita melakukan simulasi hasilnya langsung ditampilkan pada file simulasi kita.

5. Sekarang kita perlu membuat sebuah file yang akan digunakan oleh simulator sebagai sumber dari masukan vektor simulasi. Untuk membuatnya, klik pada tombol **Open** pada bagian bawah jendela **Simulator Tool**. Anda akan mendapatkan jendela baru yang memiliki nama default waveform1.vwf.
6. Klik kanan pada bagian kolom **Name** jendela tersebut dan pilih **Insert → Insert Node or Bus → Node Finder**. Anda bisa pilih pada bagian **Filter → Pins: all** kemudian klik kiri pada tombol **List** untuk mengeluarkan semua *port input/output* yang kita pakai. Klik kanan pada tombol dengan tanda >> untuk mensimulasikan seluruh port.
7. Klik  (**Detach Windows**), lalu Simpan file simulasi ini dengan nama **Tutorial1.vwf**.
8. Kemudian pada kolom **Simulation Input** di **Simulator Tool**, pilih file **Tutorial1.vwf** sebagai input simulasi.






Gambar 6. Tampilan Langkah Petunjuk f

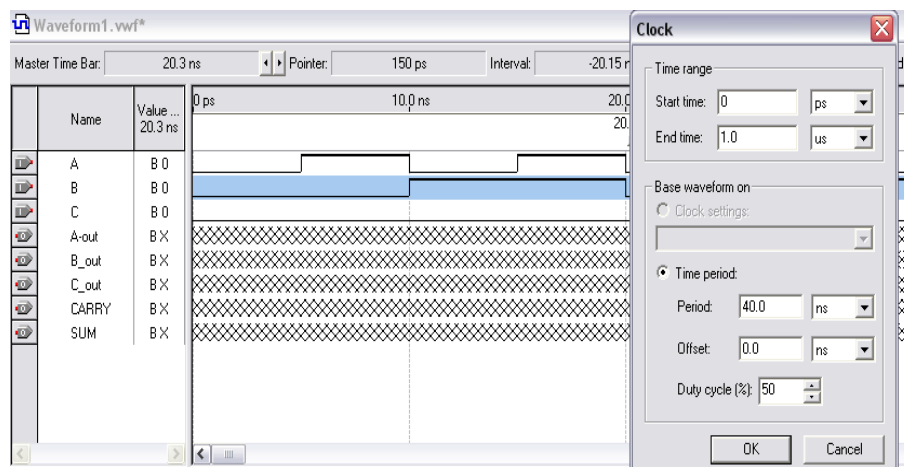
g. Membuat waveform masukan

Apabila pada akhir tahapan sebelumnya pada **Simulator Tool** kita klik tombol **Start**, maka simulasi bisa terjadi dengan bentuk input default yang biasanya tidak sesuai dengan keperluan kita, oleh karena itu kita perlu mendefinisikan bentuk sinyal masukan melalui langkah berikut ini:

1. Buka kembali file **Tutorial1.vwf** dengan menggunakan **File→Open** ataupun **SimulatorTool → Open**
2. Klik kiri pada port masukan **A** pada kolom paling kiri file tersebut.
3. Perhatikan pada jendela utama dibagian kiri setelah bagian **Project Navigator**. Setelah melakukan langkah 2 beberapa toolbar di bagian itu yang semula abu-abu (tidak aktif) berubah menjadi biru (aktif).
4. Pilih salah satu kotak tombol  yang bernama **Overwrite Clock** (berada di dalam *toolbar* dari jendela waveform). Anda dapat melihat nama tersebut dengan mengarahkan mouse Anda keatas tombol tersebut selama beberapa saat. Overwrite Clock akan menghasilkan pulsa segiempat yang berulang

terus menerus dengan periode tertentu. Anda bisa juga melakukan klik kanan pada nama pin dan pilih **Value→...** untuk menentukan bentuk sinyal input.

5. Pada jendela **Clock** seperti pada **Gambar 7** bagian kanan pilih **Time Period→Period** dan isi periode sebesar 10 ns
6. Ulangi langkah 2-5 untuk port masukan **B** tetapi nilai periode sekarang sebesar 20 ns
7. Ulangi langkah 2-5 untuk port masukan **C** tetapi nilai periode sekarang sebesar 40 ns
8. Semua langkah diatas akan menghasilkan seluruh kombinasi sinyal masukan yang mungkin untuk percobaan ini.
9. Setelah itu pada jendela **Simulator Tool** pilih tombol **Start** untuk memulai simulasi.
10. Amati hasil simulasi pada jendela tutorial.vwf dan cek apakah hasilnya sudah sesuai dengan yang diharapkan.



Gambar 7. Tampilan langkah petunjuk g

h. Mengimplementasikan desain

Setelah memastikan rancangan kita sudah benar melalui simulasi secara fungsional, waktunya untuk mengimplementasikannya pada alat sebenarnya melalui langkah-langkah berikut:

1. Lakukan kompilasi terhadap program dengan memilih **Processing → Start Compilation**.
2. Siapkan board FPGA Anda, pasang kabel catu daya dan kabel programmer pada tempatnya masing-masing dan nyalakan board tersebut.

3. Untuk konfigurasi, klik **Tools → Programmer**. Klik pada tombol **Hardware setup**. Klik pada **Add Hardware**, untuk **DE1** klik 2 kali pada **USB-Blaster (Jika tidak ada minta bantuan asisten untuk menginstall)**.
4. Kemudian pada bagian **Mode** pilih **JTAG**.
5. Jika file **Tutorial1.sof** tidak terlihat pada jendela utama programmer, klik **Add File** dan carilah file **Tutorial1.sof** kemudian klik **Open**.
6. Sorot nama file, lakukan **checklist** pada kolom **“Program/Configure”**, kemudian klik tombol **Start** untuk memprogram FPGA.
7. Sekarang coba mainkan switch 1-3 yang merepresentasikan masukan A,B, dan C. Lihat apa yang terjadi, *apakah full adder yang kita buat sudah bekerja dengan benar? Jelaskan alasan Anda!*
8. Catat hasil percobaan pada BCL Anda.

PERCOBAAN 2B: MENDESAIN FULL ADDER DENGAN PENDEKATAN BAHASA VHDL

Pada percobaan ini kita akan mendesain full adder dengan pendekatan yang berbeda yaitu dengan memanfaatkan bahasa VHDL. Sebelumnya praktikan disarankan membaca kembali bahan-bahan materi kuliah mengenai bahasa VHDL karena dalam praktikum kebanyakan materi ini tidak akan diulang kembali.

PROSEDUR PERCOBAAN:

a. Membuat Projek Baru Kembali

1. Buat project baru untuk percobaan ini seperti yang telah dilakukan pada percobaan sebelumnya dengan memperhatikan langkah-langkah di bawah ini.
2. Klik **File → New Project Wizard**
3. Buka directory dan cari folder **Tutorial2** untuk menyimpan file-file pada percobaan ini.
4. Beri nama project dan top level entity: **“modul2vhdl”**.
5. Klik Next untuk sampai ke jendela yang dapat digunakan untuk menambahkan file pendukung, lewatkan jendela ini dengan klik Next kembali
6. untuk yang mendapatkan **board DE1** untuk **“Family”** pilih **CycloneII**, kemudian dalam bagian device pilih **EP2C20F484C7**. Setelah itu klik **Finish** karena untuk langkah berikutnya kita hanya menggunakan setting default.

b. Memasukkan Desain VHDL

1. Klik **File** → **New**, pada jendela yang tampil pilih **VHDL File** sebagai pilihan desain dan klik **OK**. Klik **Detach Windows**, lalu simpan file tersebut sebagai **modul2vhdl.vhd**
2. Anda akan mendapatkan jendela kosong tempat untuk menuliskan kode VHDL Anda, pada praktikum ini Anda akan diberikan kode sumber VHDL yang akan dipakai yang ada pada Gambar 8, untuk praktikum selanjutnya hal ini tidak akan dilakukan untuk melatih Anda .

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY modul2vhdl IS
    PORT( A,B,Cin : IN STD_LOGIC;
          S,Cout  : OUT STD_LOGIC);
END modul2vhdl;

ARCHITECTURE behavioral OF modul2vhdl IS
BEGIN
    S <= A XOR B XOR Cin;
    Cout <= (Cin AND (A XOR B)) OR (A AND B);
END behavioral;

```

Gambar 8. Kode VHDL untuk Percobaan 2b

3. Seperti yang telah Anda pelajari, kode VHDL memiliki banyak bentuk arsitektur dan kode diatas hanyalah salah satunya. Setelah selesai simpan file tersebut (CTRL+S).

*Untuk langkah-langkah berikutnya akan mirip dengan **Percobaan 2a**, oleh karena itu tidak akan dituliskan kembali. Silahkan ikuti petunjuk **Percobaan 2a** mulai dari bagian **Percobaan e** hingga terakhir, tentukan posisi switch masukan ataupun posisi led 7-segment keluaran sesuai dengan keinginan Anda. Setelah itu, kerjakan tugas berikut:*

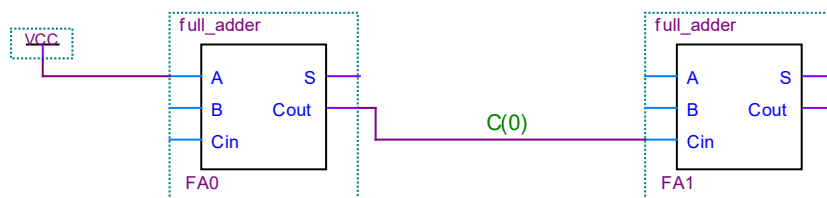
1. Pada saat simulasi dan implementasi alat apakah ada perbedaan bentuk keluaran antara menggunakan skematik dan vhd, jelaskan.
2. Jelaskan pada laporan menurut Anda, apa kelebihan dan kekurangan menggunakan vhd ataupun skematik.
3. Catat hasil percobaan pada BCL Anda.

PERCOBAAN 2C: MENDESAIN 4-BIT RIPPLE CARRY ADDER DENGAN VHDL

Kita dapat membangun n-bit adder dengan memanfaatkan kode vhdl sebelumnya melalui penggunaan kata kunci **component**. Di bawah ini Anda akan diberikan contoh **4-bit full adder** dengan arsitektur *Ripple Carry Adder*.

PROSEDUR PERCOBAAN

1. Buatlah folder dan project baru dengan nama project dan top-level entity **adder4bit**.
2. Tambahkan file vhdl pada project tersebut dan tuliskan kode yang ada pada Gambar 9 (b).
3. Lakukan simulasi secara fungsional seperlunya dan lihat apakah adder4bit kita bekerja seperti yang diharapkan. Catat hasil percobaan pada BCL Anda.



(a)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY fulladder IS
    PORT( A,B,Cin : IN STD_LOGIC;
          S,Cout  : OUT STD_LOGIC);
END fulladder;

ARCHITECTURE behavioral OF fulladder IS
BEGIN
    S <= A XOR B XOR Cin;
    Cout <= (Cin AND (A XOR B)) OR (A AND B);
END behavioral;
```

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY adder4bit IS
    PORT (
        A, B      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Cin       : IN STD_LOGIC;
        S         : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        Cout      : OUT STD_LOGIC
    );
END adder4bit;

ARCHITECTURE behavioral OF adder4bit IS
    SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0);
    COMPONENT fulladder IS
```

```

        PORT(  A,B,Cin : IN STD_LOGIC;
              S,Cout   : OUT STD_LOGIC);
    END COMPONENT;

BEGIN

FA0: fulladder PORT MAP (A=>A(0), B=>B(0), Cin=>Cin , S=>S(0), Cout=>C(0));
FA1: fulladder PORT MAP (A=>A(1), B=>B(1), Cin=>C(0), S=>S(1), Cout=>C(1));
FA2: fulladder PORT MAP (A=>A(2), B=>B(2), Cin=>C(1), S=>S(2), Cout=>C(2));
FA3: fulladder PORT MAP (A=>A(3), B=>B(3), Cin=>C(2), S=>S(3), Cout=>Cout);

END behavioral;

```

(b)

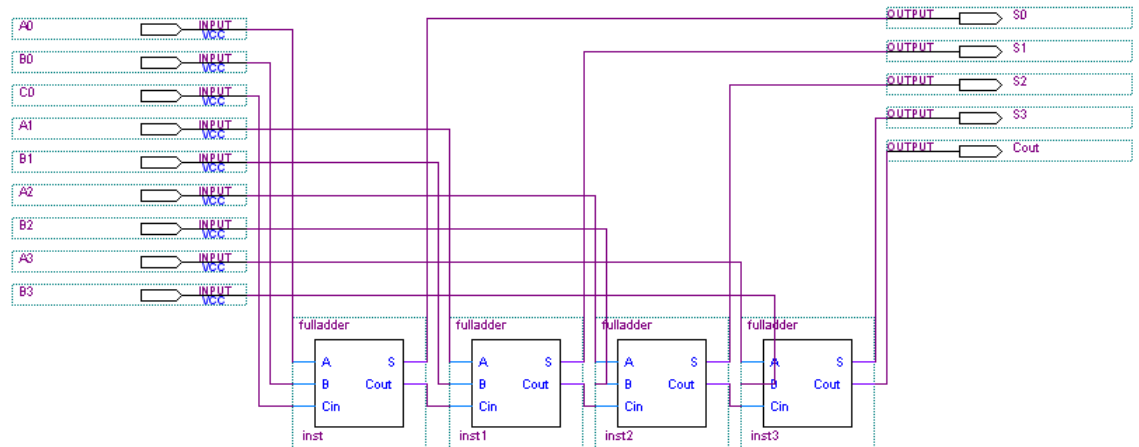
Gambar 9. (a) Ilustrasi fisis program VHDL; (b) Kode vhdI untuk Percobaan 2c

PERCOBAAN 2D: MENDESAIN 4-BIT ADDER DENGAN SKEMATIK

Terkadang membangun rangkaian digital menggunakan skematik bisa lebih mudah dibandingkan menggunakan vhdI, hal ini karena dengan menggunakan skematik kita mendapatkan visualisasi dari rangkaian yang kita bangun.

PROSEDUR PERCOBAAN

1. Buatlah project baru dengan nama project dan top-level entity **adder4bit2**
2. Kopi file pendukung yang bernama **FullAdder.bsf** dan **FullAdder.bdf** dari website labdasar ke dalam folder proyek Anda.
3. Tambahkan file skematik kosong ke dalam project Anda. Ketika Anda membuka **Symbol Tool**, Anda akan mendapati direktori baru yang bernama **project**, di dalamnya terdapat blok yang bernama **FullAdder** dan merupakan representasi skematik dari file pendukung yang kita gunakan.
4. Gunakan blok tersebut dan symbol lainnya untuk membuat rangkaian seperti pada **Gambar 10**
5. Lakukan simulasi secara fungsional dan lihat apakah hasilnya sama dengan ketika kita menggunakan vhdI. Catat hasil percobaan pada BCL Anda.



Gambar 10. Gambar skematik untuk Percobaan2D

PERCOBAAN 2E: SIMULASI SEDERHANA MENGGUNAKAN MODELSIM

Pada percobaan ini kita akan melakukan simulasi dengan software yang berbeda, yaitu Modelsim®. Modelsim yang digunakan adalah bawaan dari software Altera Quartus® versi starter edition (free license). Penggunaan simulator dengan modelsim ini penting karena:

- Software Altera Quartus® yang terbaru tidak terdapat simulator tool yang lama. Oleh karena itu digunakan software modelsim® ini sebagai solusinya.
- Software simulasi ini memiliki lebih banyak fasilitas untuk debugging
- Software ini sama dengan software yang digunakan pada design digital yang sesungguhnya terutama untuk keperluan design IC.

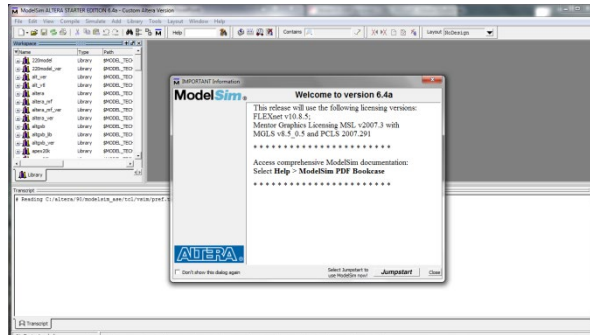
Pada percobaan ini praktikan diminta melakukan simulasi dengan desain yang sederhana dan masih menggunakan interface GUI software.

PROSEDUR PERCOBAAN

a. Memulai Simulasi dengan Menggunakan Modelsim ALTERA STARTER EDITION 6.4a

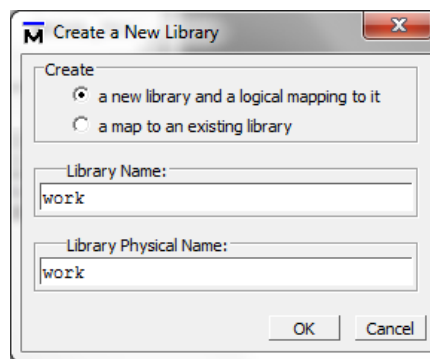
Pada tahap ini, praktikan akan membuka software/tool modelsim untuk keperluan simulasi, dan melakukan konfigurasi mengenai library/directory apa yang akan digunakan.

1. Bukalah program **Modelsim ALTERA STARTER EDITION 6.4a**. Setelah muncul tampilan seperti pada gambar di bawah ini, pilih **Close** untuk menutup jendela awal yang muncul di program ini.
2. Pilih folder sebagai direktori kerja dengan cara, pilih **File -> Change Directory** pada baris menu yang terdapat di bagian atas jendela program, lalu masukan path direktori folder kerja.

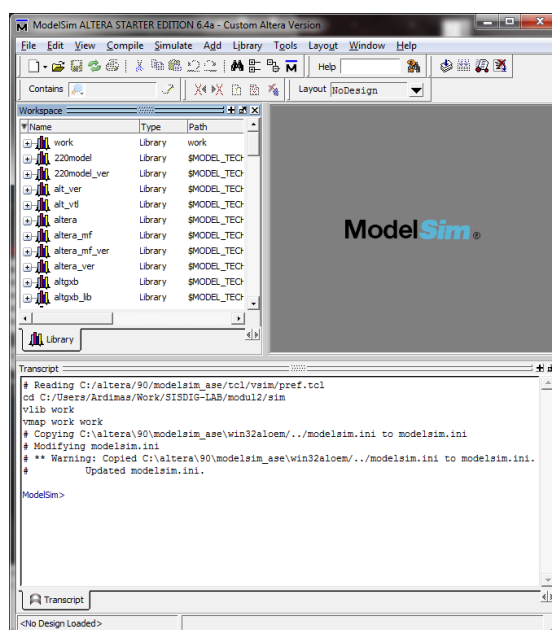


Gambar 11. Tampilan awal program Modelsim ALTERA STARTER EDITION 6.4a

3. Setelah menentukan folder yang akan digunakan, buatlah *library* baru dengan cara memilih **File -> New -> Library**. Atur dan isilah jendela yang kemudian muncul sesuai dengan gambar di bawah ini. Langkah ini dilakukan untuk memberi tahu simulator mengenai (library) directory yang akan digunakan untuk meng-*compile*, dan menggunakan hasil *compile* yang ada di direktory ini sebagai model untuk simulasi. *Library name* adalah nama library yang diberikan pada directory ini, sedangkan *Library physical name* adalah nama directory yang digunakan.



Gambar 12. Jendela untuk membuat library baru

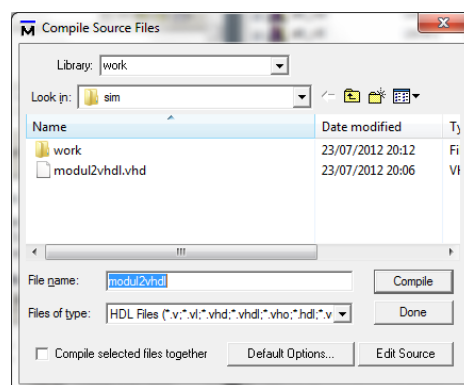


Gambar 13. Jendela tampilan library "work"

b. Menjalankan Simulasi dengan Menggunakan Modelsim ALTERA STARTER EDITION 6.4a

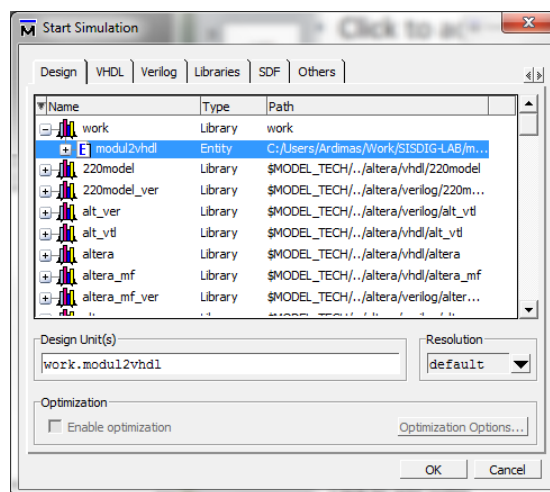
Pada tahap ini, praktikan akan akan meng-*compile* file VHDL untuk menghasilkan model simulasi, dan menggunakan model simulasi ini untuk melakukan simulasi. Proses simulasi dilakukan dengan member input, menjalankan simulasi, dan mengamati output-nya.

1. *Compile* desain yang ingin disimulasikan dengan cara memilih **Compile** -> **Compile** pada baris menu yang terdapat di bagian atas jendela Modelsim. Tentukan file yang akan di compile pada jendela yang muncul, lalu klik **Compile**. Apabila file telah selesai di compile, pilih **Done**. Langkah ini dilakukan untuk membuat model simulasi dari file VHDL ("**modul2vhdl.vhd**") yang telah dibuat. Model simulasi ini akan disimpan di dalam directory/library work.



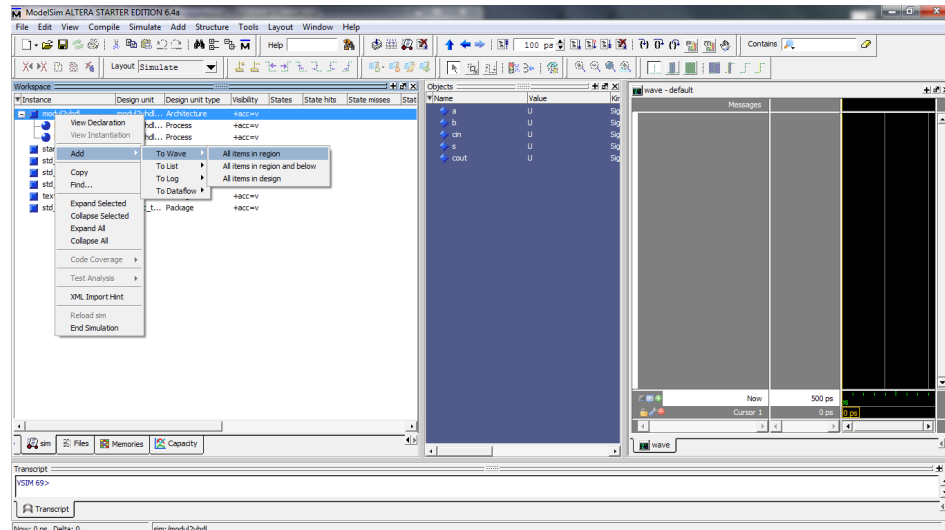
Gambar 14. Jendela tampilan untuk menu Compile

2. Setelah melakukan compile, langkah selanjutnya adalah menjalankan simulasi. Pilih menu **Simulate** -> **Start Simulation**. Pada jendela yang muncul seperti gambar di bawah ini, pilih file yang akan disimulasikan (file yang akan disimulasi merupakan file yang telah dcompile sebelumnya), kemudian pilih **OK**. Dalam langkah ini, kita memilih modul yang ada dalam *library* kita (work) untuk disimulasikan.



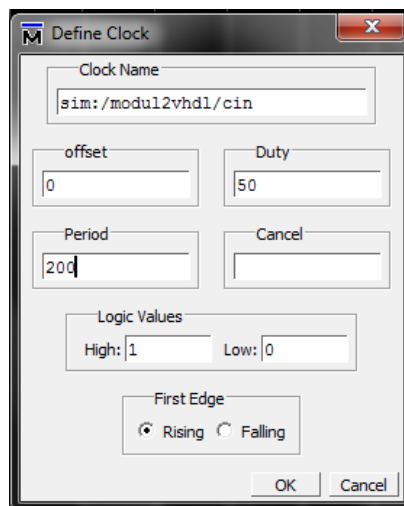
Gambar 15. Jendela untuk memilih file yang akan disimulasi

3. Tambahkan *wave* yang akan kita lihat hasil simulasi di window simulator dengancara melakukan **Right Click -> Add -> To Wave -> All item in region** seperti yang ditunjukkan pada gambar di bawah ini. Pada langkah ini kita memilih signal/port yang akan diberi input dan dilihat outputnya.



Gambar 16. Langkah penambahan *wave* pada simulasi

4. Kemudian buatlah *stimulus (sequence signal)* pada simulasi dengan cara memilih **Right Click -> Clock**. Pada jendela yang muncul seperti gambar di bawah ini, masukkan nilai periode *clock* simulasi sebesar **50ps** untuk **sinyal A** dan biarkan parameter lain disetting dalam keadaan default (tidak diubah), setelah itu pilih **OK**. Tambahkan 2 sinyal lain dengan cara yang sama namun dengan periode yang berbeda, **100ps** untuk **sinyal B** dan **200ps** untuk **sinyal C**. Stimulus ini merupakan signal yang diberikan kepada rangkaian. Selanjutnya akan dilihat output yang dihasilkan.



Gambar 17. Jendela untuk mendefinisikan *clock*

5. Jalankan simulasi dengan memilih menu **Simulate -> Run**. Ambil gambar sinyal hasil simulasi yang muncul kemudian sertakan dalam laporan! **Lakukan analisis dari hasil yang didapatkan!**

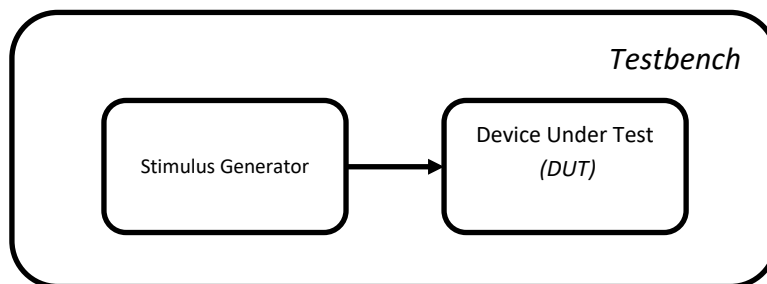
6. Setelah didapatkan hasil simulasi dari langkah sebelumnya, **Right Click -> No Force** dan jalankan kembali simulasi dengan memilih menu **Simulate -> Run**. Ambil gambar sinyal hasil simulasi yang muncul kemudian sertakan dalam laporan! **Lakukan analisis dari hasil yang didapatkan!**
7. Kemudian lakukan **Right Click -> Force** dan jalankan kembali simulasi dengan memilih menu **Simulate -> Run**. Ambil gambar sinyal hasil simulasi yang muncul kemudian sertakan dalam laporan! Lakukan analisis dari hasil yang didapatkan! Bandingkan sinyal hasil simulasi dari langkah ini dengan sinyal hasil simulasi dari kedua langkah sebelumnya. Apa yang dapat disimpulkan?

PERCOBAAN 2F: MEMBUAT TESTBENCH

Ada cara lain untuk mensimulasikan suatu sistem/rangkaian digital selain memasukan inputnya satu-persatu. Cara ini adalah dengan membuat suatu file VHDL yang berfungsi untuk memberikan input pada rangkaian yang akan diuji (disebut DUT: design under test). File ini disebut stimulus generator.

Setelah itu dibuat satu modul yang menggabungkan stimulus generator dan DUT tadi.

Testbench digunakan untuk menguji desain (DUT) dengan cara memberi sinyal stimulus masukan dan memverifikasi keluaran desain. Gambar di bawah mengilustrasikan hierarki modul testbench dan DUT.



Gambar 18. Diagram blok *Testbench*

PROSEDUR PERCOBAAN

1. Buatlah direktori (folder) baru pada direktori yang telah dibuat sebelumnya. Tambahkan file **testbench** dan file **DUT (Device Under Test)** dengan cara mengetikkan kode di bawah ini kemudian menyimpannya pada direktori tersebut untuk file **testbench** dan meng-copy dari percobaan sebelumnya untuk file **DUT**. Tb_modul2VHDL adalah entity yang mendeskripsikan testbench. Didalamnya ada dua buah block: instance modul, dan stimulus generator. Instance modul adalah bagian yang akan diuji, sedangkan stimulus generator merupakan bagian yang memberikan stimulus/input.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
  
```

```

ENTITY tb_modul2vhdl IS
END tb_modul2vhdl;

ARCHITECTURE behavioral OF tb_modul2vhdl IS

COMPONENT modul2vhdl IS
    PORT(
        A,B,Cin : IN STD_LOGIC;
        S,Cout   : OUT STD_LOGIC);
END COMPONENT;

SIGNAL A      : STD_LOGIC := '0';
SIGNAL B      : STD_LOGIC := '0';
SIGNAL Cin    : STD_LOGIC := '0';
SIGNAL S      : STD_LOGIC;
SIGNAL Cout   : STD_LOGIC;

BEGIN

dut : modul2vhdl
PORT MAP (
    A    => A    ,
    B    => B    ,
    Cin  => Cin  ,
    S    => S    ,
    Cout => Cout );

clock_A : PROCESS
BEGIN
WAIT FOR 50 ps; A <= NOT A;
end PROCESS clock_A;

clock_B : PROCESS
BEGIN
WAIT FOR 100 ps; B <= NOT B;
end PROCESS clock_B;

clock_Cin : PROCESS
BEGIN
WAIT FOR 200 ps; Cin <= NOT Cin;
end PROCESS clock_Cin;

END behavioral;

```

2. Lakukan proses compile file tersebut dengan cara seperti pada percobaan **2E-a** menggunakan **Modelsim ALTERA STARTER EDITION 6.4a**. Kemudian lakukan simulasi seperti pada percobaan **2E-b** apabila proses compile telah selesai.
3. Jalankan simulasi dengan memilih menu **Simulate -> Run**. Ambil gambar sinyal hasil simulasi tersebut, kemudian sertakan dalam laporan! Analisis sinyal hasil simulasi tersebut!
4. Kemudian lakukan modifikasi pada *stimulus generator*, dengan cara mengubah *script* yang telah disimpan sebelumnya. Ubah bagian *script* yang mendefinisikan periode *clock* dengan *script* di bawah ini

```

clock_A : PROCESS
BEGIN
WAIT FOR 50 ps; A <= NOT A;
end PROCESS clock_A;

clock_B : PROCESS
BEGIN
WAIT FOR 100 ps; B <= NOT B;
end PROCESS clock_B;

clock_Cin : PROCESS
BEGIN
WAIT FOR 200 ps; Cin <= NOT Cin;
end PROCESS clock_Cin;

```



```

stimulus : PROCESS
BEGIN
WAIT FOR 50 ps; A <= '1';
WAIT FOR 100 ps; B <= '1';
WAIT FOR 200 ps; Cin <= '1';
WAIT;
END PROCESS stimulus;

```

5. Simpan perubahan yang terjadi pada file tersebut. Lakukan kembali *compile* dan simulasi pada file yang telah dimodifikasi seperti pada langkah 2. Setelah proses simulasi berjalan, hentikan proses tersebut dengan memilih menu **Simulate -> End Simulation**. Ambil gambar sinyal hasil simulasi tersebut, kemudian sertakan dalam laporan ! Bandingkan sinyal tersebut dengan sinyal hasil simulasi pada langkah 2, sertakan analisis dan kesimpulan dalam laporan !

PERCOBAAN 2G: MELAKUKAN PROSES TAPPING SINYAL DARI SEBUAH DESAIN

Pada praktikum kali ini, kita akan melakukan tapping sinyal pada sebuah desain untuk dimunculkan pada waveform simulator. Tapping signal adalah mengambil nilai sinyal yang sebenarnya bukan merupakan output sistem/rangkaian digital yang sedang diuji. Proses ini dilakukan untuk mencari, jika ada kesalahan/bug. Dengan menggunakan tapping sinyal ini, kita dapat mengetahui lebih detail bagian yang salah.

PROSEDUR PERCOBAAN

1. Lakukan modifikasi pada file **DUT** dengan cara mengubahnya menjadi seperti *scripts* yang tertera di bawah ini

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

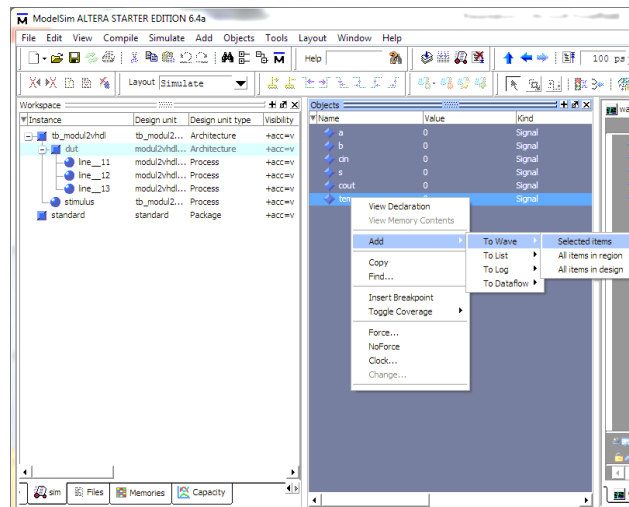
ENTITY modul2vhdl IS
    PORT(
        A,B,Cin : IN STD_LOGIC;
        S,Cout   : OUT STD_LOGIC);
END modul2vhdl;

ARCHITECTURE behavioral OF modul2vhdl IS
    SIGNAL TEMP : STD_LOGIC;
BEGIN
    TEMP <= A XOR B;
    S <= TEMP XOR Cin;
    Cout <= (Cin AND TEMP) OR (A AND B);
END behavioral;

```

2. Simpan perubahan yang terjadi pada file tersebut.
3. Lakukan kembali *compile* dan simulasi pada file yang telah dimodifikasi dengan menambahkan *wave* pada jendela simulasi yang kemudian dimunculkan oleh

program setelah langkah sebelumnya selesai dilakukan. Caranya dengan **Right Click -> Add -> To Wave -> All Items in Region** seperti yang ditunjukkan pada gambar di bawah ini. Berikan nilai untuk clock seperti pada **2E-b Langkah 4 pada instance dut**.



Gambar 19. Menambahkan wave pada simulasi

4. Jalankan simulasi dengan memilih menu **Simulate -> Run**. Ambil gambar sinyal hasil simulasi tersebut, kemudian sertakan dalam laporan! Analisis sinyal hasil simulasi tersebut! **Catatan : bedakan dengan simulator bawaan dari Quartus !**

TUGAS BONUS

PERCOBAAN 2H: MEMBUAT SCRIPT UNTUK MELAKUKAN SIMULASI

Salah satu keunggulan dari Modelsim® adalah dapat menggunakan script untuk melakukan semua prosedur simulasi sehingga kita tidak perlu melakukan proses yang sama berulang kali tiap ingin melakukan simulasi. *Script* adalah file yang berisi sekumpulan instruksi untuk melakukan sesuatu. Dalam kasus ini yang dilakukan adalah membuat *directory*, meng-compile, dan mensimulasikan rangkaian.

PROSEDUR PERCOBAAN

1. Pada direktori sebelumnya, lakukan modifikasi pada file bernama **sim.do** dengan mengetikkan *script* di bawah ini. Simpan file yang telah dimodifikasi tersebut.

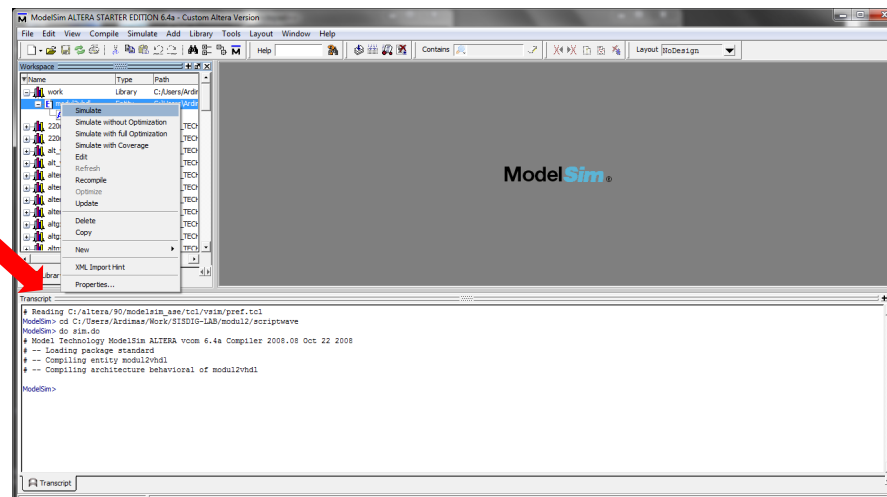
```
# Resume macro file
onbreak {resume}

# Menghapus library yang telah dibuat jika ada
if [file exists work] {
    vdel -all
}

# Membuat library
vlib work

# Compile
vcom modul2vhdl.vhd
```

2. Pada jendela *transcript* yang terdapat pada jendela program **Modelsim ALTERA STARTER EDITOR 6.4a**, ketik “do sim.do”. Setelah itu, klik kanan file **modul2vhdl** pada jendela library dan pilih **Simulate** seperti pada gambar di bawah ini.



Gambar 19. Jendela untuk melakukan simulasi pada file sim.do

3. Tambahkan *wave* pada jendela simulasi dengan melakukan **Right Click -> Add -> To Wave -> All item in region**. Kemudian buat sinyal stimulus secara manual seperti pada langkah 4 di percobaan **2A-b**
4. Pada jendela *transcript* akan muncul *script* seperti di bawah ini. Tambahkan *script* ini sebagai modifikasi pada baris paling bawah file **sim.do**, lalu simpan perubahan file tersebut

```
# Stimulus generator
force -freeze sim:/modul2vhdl/a 1 0, 0 {50 ps} -r 100
force -freeze sim:/modul2vhdl/b 1 0, 0 {100 ps} -r 200
force -freeze sim:/modul2vhdl/cin 1 0
```

5. Langkah selanjutnya adalah kembali mengetikkan “do sim.do” pada jendela *transcript*, kemudian jalankan simulasi dengan memilih menu **Simulate->Run**. Ambil gambar sinyal hasil simulasi tersebut, kemudian sertakan dalam laporan! Apa kesimpulan yang didapatkan dari percobaan ini?

6. Kemudian buatlah **file.do** seperti di bawah ini.

```
onbreak {resume}

if [file exists work] {
    vdel -all
}
vlib work

vcom modul2vhdl.vhd tb_modul2vhdl.vhd

vsim -novopt tb_modul2vhdl

add wave sim:/tb_modul2vhdl/dut/*

run 500
```

7. Lakukan simulasi **file.do** menggunakan testbench pada percobaan **2F** dengan mengetikkan “do file.do” pada jendela *transcript*. Catat hasilnya, sertakan dalam laporan, dan lakukan analisis dari hasil yang diperoleh.

1.6 MENGAKHIRI PERCOBAAN

Prosedur untuk mengakhiri percobaan:

1. Sebelum keluar dari ruang praktikum, rapikan meja praktikum. **Rapikan kabel dan matikan komputer, osiloskop, generator sinyal, dan power supply DC.** Cabut daya dari jala-jala ke kit FPGA dan letakkan kembali pada tempat semula.
2. Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau merapikan meja ketika praktikum berakhir akan mendapatkan **potongan nilai sebesar minimal 10.**
3. **Pastikan asisten telah menandatangani catatan percobaan kali ini** pada Buku Catatan Laboratorium Anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai

PERCOBAAN III

RANGKAIAN LOGIKA KOMBINASIONAL

1.1 TUJUAN

1. Mendesain rangkaian sederhana untuk melihat pengaruh waktu tunda
2. Mendesain rangkaian kombinasional berupa decoder BCD-to-7-segment untuk diimplementasikan di dalam FPGA
3. Menggunakan simulasi fungsional untuk memverifikasi fungsi rangkaian
4. Menggunakan analisis dan simulasi waktu untuk mengidentifikasi *worst case delay path*
5. Melakukan pengukuran waktu tunda propagasi pada level rangkaian
6. Mengenal level abstraksi dalam perancangan digital.

CATATAN

Untuk seluruh percobaan 3 ini, jika Anda menggunakan design skematik sesuai yang tertera di modul, maka nilai maksimal yang bisa Anda dapatkan adalah 75. Jika Anda menggunakan VHDL, maka nilai maksimal yang Anda dapatkan adalah 100. Code VHDL telah dipersiapkan praktikan sebelum praktikum. Jika ada pelanggaran berupa copy paste sebagian atau seluruh code VHDL, praktikan dikenai sanksi nilai nol dan/atau tidak lulus praktikum. Nilai tambah akan diberikan jika praktikan menggunakan simulator Modelsim®.

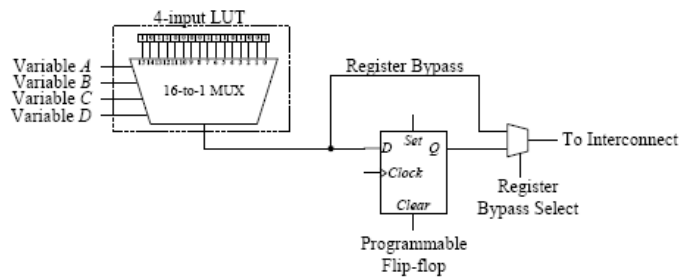
1.2 PERSIAPAN

*Pelajari kembali bahan kuliah Anda mengenai rangkaian logika kombinasional. Pelajari juga keseluruhan petunjuk praktikum untuk modul rangkaian logika kombinasional ini. Kerjakan **Tugas Pendahuluan** dan kumpulkan sesuai ketentuan yang berlaku.*

1.3 DASAR TEORI

IMPLEMENTASI FPGA DAN WAKTU TUNDA

Dalam teknologi Altera Cyclone yang kita gunakan, fungsi logika diuraikan oleh software implementasi kedalam bentuk subfungsi 4-masukan. Setiap subfungsi kemudian diimplementasikan oleh tabel kebenaran yang bekerja seperti multiplexer dan dibuat dengan memprogram SRAM yang mendefinisikan fungsionalitas dari FPGA. Setiap tabel kebenaran memiliki waktu tunda yang berkontribusi ke waktu tunda keseluruhan. Sedangkan untuk membedakan antara rangkaian kombinasional dan sekuensial, dalam subfungsi juga diberikan sebuah D flip-flop seperti yang terlihat pada Gambar 1 .



Gambar 1. Bentuk subfungsi yang merepresentasikan logika pada FPGA

Penguraian kedalam subfungsi yang dikombinasikan dengan routing interkoneksi menghasilkan ketidakpastian dalam delay propagasi dari masukan ke keluaran dalam implementasi rangkaian. Suatu persamaan logika dengan 2 variabel mungkin saja memiliki waktu tunda yang sama dengan yang menggunakan 4 variabel karena bentuk subfungsi FPGA.

Perancang yang berpengalaman mungkin bisa menggunakan pengaturan tertentu untuk menspesifikasikan waktu tunda maksimum yang dapat diterima. Apapun masalahnya, sangat berguna bagi kita untuk mengetahui berapa waktu tunda dari rangkaian kita. Karena hampir semua rangkaian kombinasional ditempatkan pada kondisi sekuensial, biasanya kita tertarik pada *worst case delay* yang bisa terjadi dalam operasi rangkaian dari masukan rangkaian kombinasional ke setiap keluaran rangkaian kombinasional.

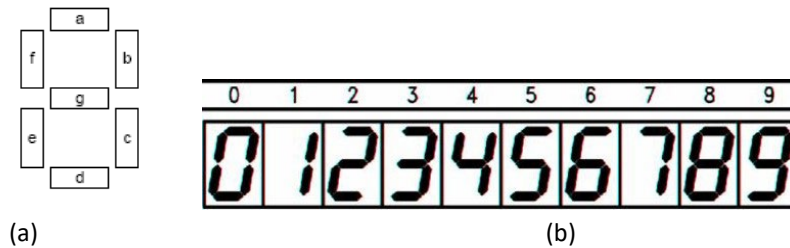
Estimasi *worst case delay* ditentukan dengan menambahkan *delay* perkiraan maksimum kedalam rangkaian kombinasional termasuk logika dan interkoneksi. Karena ketidakpastian ini, *worst case delay* hanya bisa ditentukan setelah proses implementasi selesai termasuk penguraian menjadi subfungsi dan routing interkoneksi.

Dalam percobaan ini, kita akan membangun dua rangkaian. Dengan rangkaian pertama kita akan melihat beberapa tipe dari simulasi yang dapat kita gunakan dan melihat kemungkinan efek dari proses penguraian yang mengimplementasikan rangkaian sebenarnya secara fisik. Kemudian dengan rangkaian kedua, selain memverifikasi fungsionalitasnya, kita juga akan mencari *worst case delay* dari setiap masukan ke setiap keluaran dan akan menggunakan metode simulasi yang hanya dapat diaplikasikan pada rangkaian sederhana untuk mencari jalur sebenarnya yang ditempuh dimana delay ditemukan. Dengan Mengetahui jalur dari *worst case delay* kita kemudian bisa mengukur delay pada setiap titik jalur tersebut di lab. Delay yang terukur ini bukanlah *worst case delay* tetapi lebih kepada waktu tunda rata-rata.

BCD-TO-7-SEGMENT CODE CONVERTER

Rangkaian ini digunakan untuk mengkonversikan suatu nilai desimal terkode biner(BCD) ke pola segmen yang sesuai pada display 7-segmen. Karena nilai BCD adalah angka 4-bit pada jangkauan 0-9, bagaimana kita memperlakukan nilai 10-15(don't care atau tidak) akan berpengaruh pada desain kita.

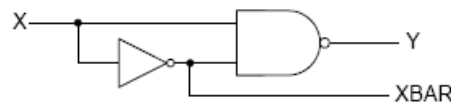
7-segmen biasanya diidentifikasi dalam industri menggunakan huruf a-g seperti pada Gambar 2 berikut ini:



Gambar 2. (a) Konvensi penomoran 7-segmen, (b) Pola Display 7-segmen

1.4 TUGAS PENDAHULUAN

1. Jelaskan apa yang dimaksud dengan rangkaian kombinasional, berikan satu contoh rangkaian kombinasional sederhana selain Adder atau materi percobaan ini, turunkan K-maps dan tabel kebenarannya!!
2. Perhatikan **Gambar 3** dibawah ini, carilah tabel kebenaran dari rangkaian tersebut!
3. Asumsikan kita akan membangun rangkaian pada Gambar 3 dengan komponen gerbang logika, gambarkan dan jelaskan perkiraan bentuk dari keluaran Y jika kita mengaplikasikan sinyal kotak pada masukan X!!



Gambar 3. Bentuk rangkaian dalam project **seederhana**

4. Pelajari rangkaian BCD-to-7-segment, buatlah tabel kebenaran dan K-maps pada tabel yang tersedia pada bagian akhir modul ini dimana kita menggunakan 4 masukan yang bernama D3(MSB)...D0(LSB) dan 7 keluaran yang bernama A..G, kemudian buatlah persamaan boolean berbentuk Sum Of Product (SOP)/POS yang minimal!!! (Anggap untuk masukan diluar 0..9 sebagai don't care dan sinyal masukan/keluaran adalah *active HIGH* ('1'=aktif/menyal).)
5. Siapkan skematik atau kode VHDL (sesuai arahan percobaan) untuk percobaan 3A hingga 3C, serta lakukan simulasi fungsional untuk seluruh rangkaian digital bersangkutan. Tambahkan *screenshot* hasil simulasi fungsional pada tugas pendahuluan anda!

Tugas pendahuluan di bawah ini merupakan tugas pendahuluan tambahan dengan materi yang berkaitan dengan percobaan modul 3D – Perancangan Kalkulator Sederhana.

6. Buatlah sebuah blok *subtractor* menggunakan bahasa VHDL dengan menggunakan pendekatan *high level abstraction* (menggunakan library *unsigned*).
7. Buatlah sebuah blok *comparator* menggunakan bahasa VHDL dengan menggunakan pendekatan *high level abstraction*.
8. Lakukan kompilasi *Analysis & Synthesis* dari project kalkulator (blok komponen lain seperti *top-level* dapat dilihat pada modul tambahan percobaan 3D).
9. Lakukan simulasi baik secara *timing* maupun *functional*.

1.5 PERCOBAAN

PERALATAN YANG DIGUNAKAN

- Board FPGA tipe DE1
- Catu daya + kabel dan konektor tambahan serta kabel *downloader*
- Osiloskop

PROSEDUR PERCOBAAN

PERCOBAAN 3A: MEMBUAT RANGKAIAN SEDERHANA

Dalam percobaan ini, Anda akan membuat 2 project, yang pertama diberi nama **sederhana** dan hanya terdiri dari satu skematik, yang kedua akan diberi nama **bcd** dan memerlukan 2 skematik.

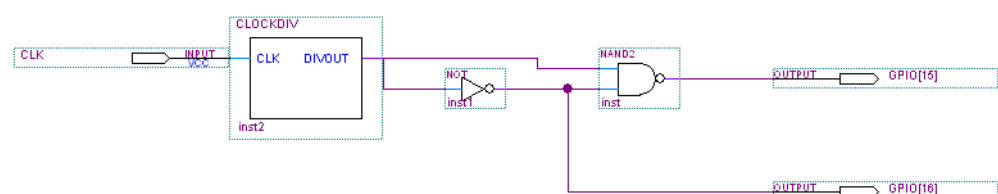
PROSEDUR PERCOBAAN:

a. Manajemen File

1. Buatlah direktori baru dengan nama **Modul3** pada direktori praktikum digital, kemudian di dalamnya buatlah dua direktori baru kembali dengan nama **sederhana** dan **bcd**.

b. Pembuatan Project Sederhana

1. Buat proyek Quartus baru dengan nama **sederhana** pada direktori **sederhana**
2. Download file **clockdiv.vhd** dan **clockdiv.bsf** dari web labdasar. Rangkaian ini akan digunakan untuk memperlambat clock masukan rangkaian sederhana.
3. Buatlah sebuah file diagram skematik baru bernama **sederhana.bdf**, tambahkan file tersebut ke dalam project dan implementasikan rangkaian pada **Gambar 4**. Berikan nama kaki masukan sebagai **CLK[0]**. Berikan nama kaki keluaran yang tersambung ke gerbang NAND (keluaran Y pada Gambar 3) sebagai **GPIO[15]** dan untuk kaki keluaran yang tersambung dengan inverter dengan nama **GPIO[16]**.



Gambar 4. Rangkaian Gambar 3 dengan modifikasi

c. Kompilasi project dan Simulasi

1. Untuk keperluan I/O pin lakukan seperti pada modul 2.
2. Lakukan compile pada project Anda, jika ada error perbaiki skematik Anda kemudian ulangi langkah sebelumnya. Pada tahap ini mungkin akan terdapat banyak warning karena banyak port yang tidak kita gunakan tetapi hal ini tidak akan menjadi masalah pada percobaan ini.
3. Pertama kita akan menggunakan simulasi **Functional** seperti pada percobaan 2. Ikuti langkah-langkah yang telah Anda pelajari pada percobaan 2 untuk melaksanakannya, atur simulasi sehingga sinyal yang dipakai harus dapat merepresentasikan setiap kemungkinan logika!
4. Simpan hasil simulasi Anda untuk dilampirkan pada laporan Anda.
5. Sekarang, ubahlah **Simulation Mode** menjadi **Timing** dan jalankan simulasi kembali.
6. Catat hasil percobaan pada BCL Anda.
7. Jawab beberapa pertanyaan berikut pada laporan Anda:
 - i. Apa perbedaan dari kedua mode simulasi tersebut?
 - ii. Menurut Anda mode simulasi mana yang akan lebih memodelkan secara akurat kondisi nyata rangkaian yang Anda rancang?
 - iii. Apakah Anda mengharapkan hasil sebenarnya lebih baik, buruk, atau sama saja dibandingkan simulasi yang Anda coba dan mengapa demikian?

PERCOBAAN 3B: MEMBUAT RANGKAIAN BCD

a. Pembuatan project BCD.

1. Buatlah project Quartus baru bernama **bcd** pada direktori **bcd**
2. Import pin assignment seperti pada percobaan sebelumnya.
3. Buatlah dua file diagram skematik, yang satu bernama **bcd_test.bdf** dan satunya lagi bernama **bcd_7seg.bdf** (file yang terakhir ini tidak ditambahkan dalam project).

b. Pembuatan skematik

1. Desainlah sebuah rangkaian decoder BCD-to-7-segment seperti yang dispesifikasikan diatas dengan menggunakan persamaan Boolean berbentuk

Sum of Product (SOP)/ POS minimal yang sudah Anda kerjakan pada tugas pendahuluan.

2. **Bcd_7seg.bdf** : Anda akan mengimplementasikan rangkaian decoder BCD-to-7-segment pada file skematik ini. Kemudian dalam beberapa kasus untuk penyederhanaan rangkaian gunakan gerbang NAND gate (BANDx pada Quartus) misalnya untuk mengimplementasikan logika $\overline{X2.X1.X0}$ tanpa harus menggunakan 3 inverter. Gunakan gerbang logika dan pin input/output sesuai keperluan. Setelah selesai pilih **File→Create/Update→Create Symbol for Current File**. Langkah ini akan membuat skematik kita bisa digunakan pada skematik lain sebagai blok fungsi.
3. **Bcd_test.bdf** : dalam skematik ini Anda akan memasukkan rangkaian BCD-to-7-segment pada skematik lainnya sebagai blok fungsi dan menghubungkan input kepada switch dan output dengan 7-segment display. Masukkan blok bcd_7seg(terdapat di **Symbol Toolbox→Project**) kemudian sambungkan kaki-kaki pada blok bcd_7seg dengan pin input dan output yang masing-masing dinamakan seperti pada tabel dibawah ini.

Tabel 1. Penamaan Pin Input/Output

| <i>Nama Pin Pada kaki bcd_7seg</i> | <i>Nama Pin Input/Output</i> |
|------------------------------------|------------------------------|
| <i>D3</i> | <i>SW1[3]</i> |
| <i>D2</i> | <i>SW1[2]</i> |
| <i>D1</i> | <i>SW1[1]</i> |
| <i>D0</i> | <i>SW1[0]</i> |
| <i>A</i> | <i>HEX1[0]</i> |
| <i>B</i> | <i>HEX1[1]</i> |
| <i>C</i> | <i>HEX1[2]</i> |
| <i>D</i> | <i>HEX1[3]</i> |
| <i>E</i> | <i>HEX1[4]</i> |
| <i>F</i> | <i>HEX1[5]</i> |
| <i>G</i> | <i>HEX1[6]</i> |

c. Pembuatan Netlist dan Simulasi Fungsional

1. Set skematik **bcd_test** sebagai Top Level entity pada hierarki program. Hal ini bisa dilakukan dengan memilih **Project→Set as Top-Level Entity**.
2. Simulasikan rangkaian untuk setiap kombinasi masukan yang mungkin dengan menggunakan jenis masukan **Overwrite Clock** seperti yang dilakukan pada percobaan sebelumnya.
3. Simpan hasil simulasi Anda untuk dilampirkan pada laporan dan jelaskan apakah decoder Anda sudah berfungsi dengan benar?

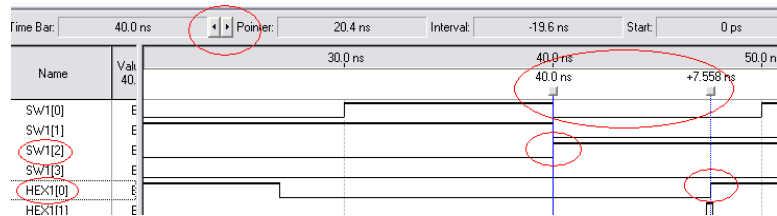
d. Simulasi Timing

1. Lakukan simulasi timing pada rangkaian menggunakan bentuk sinyal masukan yang sama seperti pada simulasi fungsional. Pastikan simulasi sudah diset sebagai **Timing** bukan **Fungsional**.
2. **Compile** dahulu project Anda apabila belum dilakukan.
3. Jalankan simulasi dan lihatlah apakah keluaran identik dengan simulasi secara fungsional (kecuali beberapa delay dan glitch).
4. Jangan tutup jendela simulasi Timing karena akan digunakan untuk analisa selanjutnya

e. Simulasi Worst Case Delay

1. Periksa bagian **Timing Analyzer Summary** dan **tpd** dari **Processing→Compilation Report**, kemudian cari pasangan kaki keluaran-masukan yang memiliki delay maksimal/paling besar. Selanjutnya kaki masukan dari delay maksimum ini kita beri nama sebagai **MasukanDelay** dengan simbol **Xi**(misalkan $X_i = SW1[3]$) dan keluarannya akan kita beri nama **KeluaranDelay** dengan simbol **Yj** (misalkan $Y_j = HEX1[0]$).
2. Dengan melihat tabel kebenaran dari keluaran **Yj** carilah semua **nilai set** dari **Xi** dimana ketika **Xi** berubah dari **'0'→'1'** atau **'1'→'0'** maka **Yj** akan berubah pula nilainya. Misalkan $X_i = SW1[3]$ dan berdasarkan tabel kebenaran saat masukan $SW1[3]=1, SW1[2]=1, SW1[1]=0, SW1[0]=0$, **Yj** bernilai **0**, kemudian saat masukan $SW1[3]=0, SW1[2]=1, SW1[1]=0, SW1[0]=0$, **Yj** bernilai **1** maka $SW1[2], SW1[1], SW1[0] = (1,0,0)$ adalah nilai set. Ulangi untuk kombinasi lain hingga Anda mendapatkan seluruh nilai set yang ada.
3. Jawab beberapa pertanyaan berikut pada laporan Anda:
 - Berapa delay maksimum dari decoder?
 - Apakah nama input dari MasukanDelay yang diberi kode **Xi**?
 - Apakah nama output dari KeluaranDelay yang diberi kode **Yj**?
4. Apa saja nilai masukan yang Anda dapatkan sebagai nilai set pada point nomer 2?
5. Laksanakan kembali *timing simulation*, kali ini Anda hanya memakai kombinasi nilai input yang mengakibatkan **nilai set**. Disini kita akan mencari kombinasi input yang mengakibatkan worst case delay.
6. Buka hasil simulasi pada **Simulation Report**. Arahkan mouse pada bagian gambar pulsa, klik kanan dan pilih insert time bar hingga terdapat 2 time bar pada gambar pulsa.

7. Geserlah **time bar** hingga yang satunya berada pada posisi ketika input yang dianggap **Xi** berubah dan yang satunya pada posisi ketika input yang dianggap **Yj** ikut berubah. Geser-geser menggunakan panah di sebelah kanan tulisan master time bar untuk memposisikan time bar dengan tepat. Lihat angka yang terdapat diatas **time bar** dan **catat delay** dari masing-masing kombinasi nilai set tersebut!



Gambar 5. Contoh menghitung delay jika **Xi = SW1[2]** dan **Yj = HEX1[0]**

8. Jawab pertanyaan berikut pada laporan Anda:
 - Berapa delay maksimum yang terukur pada simulasi kali ini, apakah sama dengan yang didapatkan pada langkah 1?
 - Untuk kombinasi masukan bagaimana delay maksimum tersebut didapatkan?

f. Memprogram kedalam FPGA

1. Coba Anda download program BCDto-7-segmen Anda kedalam board FPGA yang tersedia, lihat kembali modul 2 untuk cara pemrograman.
2. Mainkan 4 switch yang kita pakai pada percobaan ini dan lihat apakah program kita sudah berjalan dengan benar.
3. Catat Hasil percobaan pada BCL Anda.

Setelah menyelesaikan ini, simpan seluruh file percobaan3B karena **akan digunakan kembali** pada modul 4 dan modul 5. Jika belum selesai maka selesaikan di rumah.

PERCOBAAN 3C: MERANCANG BCD 7SEG DENGAN LEVEL ABSTRAKSI BEHAVIORAL

Pada percobaan kali ini kita akan mengimplementasikan desain dengan level abstraksi yang lebih tinggi. Level abstraksi yang tinggi artinya lebih dekat dengan cara manusia berpikir. Pada percobaan ini ditunjukkan bahwa kita sering kali tidak perlu melakukan/mencari persamaan logika untuk setiap signal/variable. Pada contoh ini, praktikan cukup menentukan bentuk keluaran, untuk setiap jenis input yang diinginkan. Proses merubah menjadi persamaan Boolean, meminimisasi, dan membuat rangkaian gerbang logikanya dikerjakan oleh tool/software. Dengan cara ini manusia/engineer dapat membuat rangkaian yang lebih besar/kompleks karena tidak perlu memikirkan detailnya.

PROSEDUR PERCOBAAN:

1. Buatlah folder baru untuk melakukan percobaan pada praktikum ini. Folder ini nantinya digunakan sebagai direktori kerja, untuk menyimpan file-file yang berhubungan dengan praktikum ini.
2. Buatlah file **DUT (Device Under Test)** dengan cara mengetikkan *script* di bawah ini menggunakan *text editor*, kemudian simpan file tersebut di folder yang telah dibuat pada langkah sebelumnya.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY bcd IS PORT (
    SW      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    HEX1    : OUT STD_LOGIC_VECTOR (1 TO 7));
END bcd;

ARCHITECTURE behavioral OF bcd IS

    CONSTANT NOL      : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
    CONSTANT SATU     : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0001";
    CONSTANT DUA      : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0010";
    CONSTANT TIGA     : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011";
    CONSTANT EMPAT    : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0100";
    CONSTANT LIMA     : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0101";
    CONSTANT ENAM     : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0110";
    CONSTANT TUJUH    : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0111";
    CONSTANT DELAPAN  : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1000";
    CONSTANT SEMBILAN : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1001";

BEGIN

    PROCESS (SW)
    BEGIN
        CASE SW IS
            WHEN NOL      => HEX1 <= "1111110";
            WHEN SATU     => HEX1 <= "0110000";
            WHEN DUA      => HEX1 <= "1101101";
            WHEN TIGA     => HEX1 <= "1111001";
            WHEN EMPAT    => HEX1 <= "0110011";
            WHEN LIMA     => HEX1 <= "1011011";
            WHEN ENAM     => HEX1 <= "1011111";
            WHEN TUJUH    => HEX1 <= "1110000";
            WHEN DELAPAN  => HEX1 <= "1111111";
            WHEN SEMBILAN => HEX1 <= "1110011";
            WHEN OTHERS   => HEX1 <= "0000000";
        END CASE;
    END PROCESS;

END behavioral;

```

3. Buatlah file **Testbench** dengan cara mengetikkan *script* di bawah ini menggunakan *text editor*, kemudian simpan file tersebut di folder yang telah dibuat pada langkah sebelumnya.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

```

```

ENTITY tb_bcd IS
END tb_bcd;

ARCHITECTURE behavioral OF tb_bcd IS

    SIGNAL clk : STD_LOGIC := '0';
    SIGNAL SW   : STD_LOGIC_VECTOR (3 DOWNT0 0) := "0000";
    SIGNAL HEX1 : STD_LOGIC_VECTOR (1 TO 7);

    COMPONENT bcd IS
        PORT (
            SW : IN STD_LOGIC_VECTOR (3 DOWNT0 0);
            HEX1 : OUT STD_LOGIC_VECTOR (1 TO 7));
    END COMPONENT;

BEGIN

    dut : bcd
    PORT MAP (
        SW => SW ,
        HEX1 => HEX1 );

    clock : PROCESS
    BEGIN
        WAIT FOR 50 ps; clk <= not clk;
    end PROCESS clock;

    increment: PROCESS (clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            SW <= SW + "0001";
        END IF;
    END PROCESS;

END behavioral;

```

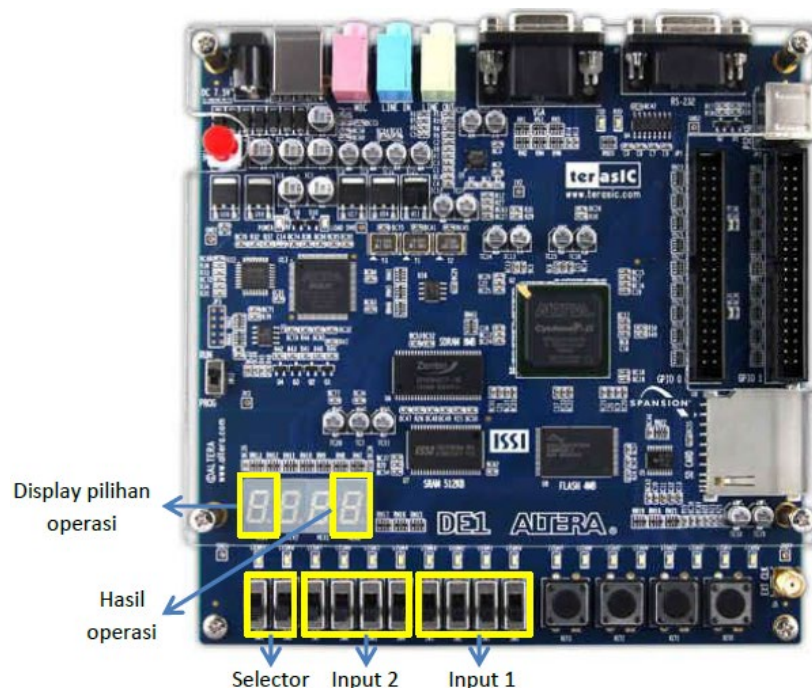
4. Buatlah file **file.do** dengan cara mengetikkan script seperti yang telah dilakukan pada percobaan **2H**. Lakukan beberapa modifikasi penyesuaian pada *script* tersebut.
5. Pada jendela *transcript* ketikkan “do file.do” untuk menjalankan simulasi. Ambil gambar sinyal hasil simulasi tersebut, kemudian sertakan dalam laporan! Analisis sinyal hasil simulasi tersebut!
6. Implementasikan desain ini ke dalam FPGA dengan cara yang telah dijelaskan pada percobaan sebelumnya. Apakah hasilnya menunjukkan behavioral yang sama dengan langkah **3-B**? Analisis kelebihan dan kekurangan masing-masing level abstraksi!

PERCOBAAN 3D : MERANCANG SEBUAH KALKULATOR SEDERHANA

Operasi yang telah dipelajari hingga percobaan sebelum ini adalah menggunakan pendekatan abstraksi *gate-level*, dimana operasi dilakukan dalam bit per bit. Sekarang banyak pihak sudah menggunakan pendekatan *high level abstraction*, dimana dalam implementasinya tentu lebih praktis. Di percobaan ini terdapat operasi penjumlahan dan pengurangan yang memanfaatkan library *unsigned* (IEEE.STD_LOGIC_UNSIGNED.ALL), sehingga tidak perlu lagi menggunakan

rancangan operasi penjumlahan/pengurangan dengan abstraksi *gate-level*. Dengan *library* tersebut, operasi matematika dapat dilakukan dengan menggunakan operatornya langsung.

Pada percobaan ini, akan dibuat suatu “kalkulator sederhana”, yang dapat digunakan untuk melakukan 3 operasi, yaitu : penjumlahan, pengurangan, dan komparasi. Kalkulator ini memiliki dua input bilangan bulat 4-bit yang akan dioperasikan. Input dapat diatur dari switch SW0 hingga SW3 (input pertama) dan switch SW4 hingga SW7 (input kedua). Hasil operasi akan ditampilkan pada 7-segment HEX0 pada FPGA. Hasil operasi disesuaikan dengan batas representasi data sebuah 7-segment, yaitu dari 0-9, di luar itu tidak akan ditampilkan pada 7-segment. Pemilihan operasi dapat dilakukan dengan sebuah selector 2-bit yang diatur dengan menggunakan SW8 dan SW9. Pilihan operasi (selector) ditampilkan pada 7-segment HEX3 pada FPGA, tampilan angka 0,1, dan 2 masing-masing menunjukkan pemilihan operasi komparasi, pengurangan, dan penjumlahan. Jika selector diatur bernilai 3, maka hasil operasi sama dengan nol. Gambar 6 menunjukkan input dan output percobaan ini sesuai dengan deskripsi di atas.



Gambar 6. Input dan output percobaan 3D pada FPGA

Operasi komparasi memberikan keluaran 1 jika input kedua lebih besar dari input pertama, jika input pertama lebih besar, maka keluarannya 2, jika input pertama sama dengan input kedua maka keluarannya 3.

Arsitektur program ini dibagi dalam 6 blok, yaitu blok *adder* untuk operasi penjumlahan, blok *subtractor* untuk operasi pengurangan, blok *comparator* untuk operasi komparasi, blok *multiplexer* sebagai selector, blok *display* untuk tampilan pada 7-segment, serta blok kalkulator sebagai top-level. Blok kalkulator, *display*, *multiplexer*, dan *adder* diberikan lengkap dalam VHDL, sementara blok *subtractor* dan *comparator* dibuat oleh praktikan.

PROSEDUR PERCOBAAN:

1. Buatlah project baru pada folder yang berbeda dengan percobaan-percobaan sebelumnya.
2. Buatlah file-file VHDL untuk masing-masing blok dengan mengetikkan script di bawah ini, dan simpan dalam folder project yang telah dibuat. Perhatikan bahwa nama masing-masing file VHDL harus sama dengan nama *entity* pada program.

A. Blok kalkulator / *top-level* (simpan dengan nama file kalkulator.vhd)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity kalkulator is
    port(
        input1, input2: in std_logic_vector(3 downto 0); --
        input angka
        in_sel: in std_logic_vector(1 downto 0); -- selektor
        operasi
        result: out std_logic_vector(3 downto 0); -- output
        hasil operasi
        a_7segment, b_7segment, c_7segment, d_7segment: out
        std_logic_vector(6 downto 0) -- display 7-segment
    );
end kalkulator;

architecture kalkulator_arc of kalkulator is

    -- komponen selektor
    component mux is
        port(
            sel: in std_logic_vector(1 downto 0);
            com, sub, add: in std_logic_vector(3 downto 0);
            output: out std_logic_vector(3 downto 0)
        );
    end component;

    -- komponen operasi komparasi
    component comparator is
        port(
            x, y: in std_logic_vector(3 downto 0);
            output: out std_logic_vector(3 downto 0)
        );
    end component;

    -- komponen operasi pengurangan
    component subtractor is
        port(
            x, y: in std_logic_vector(3 downto 0);
            output: out std_logic_vector(3 downto 0)
        );
    end component;

    -- komponen operasi penjumlahan
    component adder is
        port(
            x, y: in std_logic_vector(3 downto 0);
```

```

        output: out std_logic_vector(3 downto 0)
    );
end component;

-- komponen display 7-segment
component display7S is
    port(
        sel: in std_logic_vector(1 downto 0);
        r_operation: in std_logic_vector(3 downto 0);
        a_7S, b_7S, c_7S, d_7S: out std_logic_vector(6
            downto 0)
    );
end component;

signal r_com, r_sub, r_add, r_mux: std_logic_vector(3 downto 0);

begin

    -- alur data
    X_COM: comparator port map(input1, input2, r_com);
    X_SUB: subtractor port map(input1, input2, r_sub );
    X_ADD: adder port map(input1, input2, r_add );
    X_MUX: mux port map(in_sel, r_com, r_sub, r_add, r_mux);
    X_DIS: display7S port map (in_sel, r_mux, a_7segment,
        b_7segment, c_7segment, d_7segment);

    -- hasil
    result <= r_mux;

end kalkulator_arc;

```

B. Blok *display* (simpan dengan nama file display7S.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity display7S is
    port(
        sel: in std_logic_vector(1 downto 0);
        -- selektor operasi
        r_operation: in std_logic_vector(3 downto 0);
        -- hasil operasi
        a_7S, b_7S, c_7S, d_7S: out std_logic_vector(6
            downto 0) -- display keseluruhan operasi
    );
end display7S;

architecture display7S_arc of display7S is
begin
    process(sel, r_operation)
    begin

        case r_operation is
            -- display hasil operasi
            when "0000" => a_7S <= "0000001"; -- '0'
            when "0001" => a_7S <= "1001111"; -- '1'
            when "0010" => a_7S <= "0010010"; -- '2'
            when "0011" => a_7S <= "0000110"; -- '3'
            when "0100" => a_7S <= "1001100"; -- '4'

```

```

        when "0101"=> a_7S <="0100100"; -- '5'
        when "0110"=> a_7S <="0100000"; -- '6'
        when "0111"=> a_7S <="0001111"; -- '7'
        when "1000"=> a_7S <="0000000"; -- '8'
        when "1001"=> a_7S <="0000100"; -- '9'
        when others=> a_7S <="1111111";
    end case;

    b_7S <= "1110110"; -- display karakter "="
    c_7S <= "1111010"; -- display karakter "r"

    case sel is
        -- display pilihan selektor
        when "00"=> d_7S <="0000001"; -- '0'
        when "01"=> d_7S <="1001111"; -- '1'
        when "10"=> d_7S <="0010010"; -- '2'
        when "11"=> d_7S <="0000110"; -- '3'
    end case;

    end process;
end display7S_arc;

```

C. Blok *multiplexer* (simpan dengan nama file mux.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mux is
    port(
        sel: in std_logic_vector (1 downto 0); -- selektor operasi
        com, sub, add: in std_logic_vector(3 downto 0); -- hasil operasi komparasi, pengurangan, penjumlahan
        output: out std_logic_vector(3 downto 0) -- output selektor
    );
end mux;

architecture mux_arc of mux is
begin
    process(sel, com, sub, add )
    begin
        if sel = "00" then
            output <= com; -- ketika selektor = 0, memilih hasil operasi komparasi
        elsif sel = "01" then
            output <= sub; -- ketika selektor = 1, memilih hasil operasi pengurangan
        elsif sel = "10" then
            output <= add; -- ketika selektor = 2, memilih hasil operasi penjumlahan
        else
            output <= "0000";
        end if;
    end process;
end mux_arc;

```

D. Blok *adder* (simpan dengan nama file *adder.vhd*)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity adder is
    port(
        x, y: in std_logic_vector(3 downto 0); -- input
        angka
        output: out std_logic_vector(3 downto 0) -- output
        hasil operasi
    );
end adder;

architecture adder_arc of adder is
begin
    process(x, y)
    begin -- algoritma utama operasi penjumlahan
        output <= unsigned(x)+unsigned(y);
    end process;
end adder_arc;
```

E. Blok *subtractor* (simpan dengan nama file *subtractor.vhd*)

Blok ini dibuat oleh praktikan.

F. Blok *comparator* (simpan dengan nama file *comparator.vhd*)

Blok ini dibuat oleh praktikan.

3. Lakukan kompilasi Analysis & Synthesis.
4. Lakukan pengaturan pin FPGA (Pin Planner). Pin-pin yang digunakan adalah SW0 hingga SW9, serta HEX0 hingga HEX3. Pin diatur sedemikian sesuai tabel 2.

Tabel 2. Pengaturan pin percobaan 3D

| To | Direction | Location |
|---------------|-----------|----------|
| a_7segment[6] | Output | PIN_J2 |
| a_7segment[5] | Output | PIN_J1 |
| a_7segment[4] | Output | PIN_H2 |
| a_7segment[3] | Output | PIN_H1 |
| a_7segment[2] | Output | PIN_F2 |
| a_7segment[1] | Output | PIN_F1 |
| a_7segment[0] | Output | PIN_E2 |
| b_7segment[6] | Output | PIN_E1 |
| b_7segment[5] | Output | PIN_H6 |
| b_7segment[4] | Output | PIN_H5 |
| b_7segment[3] | Output | PIN_H4 |
| b_7segment[2] | Output | PIN_G3 |
| b_7segment[1] | Output | PIN_D2 |
| b_7segment[0] | Output | PIN_D1 |
| c_7segment[6] | Output | PIN_G5 |
| c_7segment[5] | Output | PIN_G6 |
| c_7segment[4] | Output | PIN_C2 |
| c_7segment[3] | Output | PIN_C1 |

| | | |
|---------------|--------|---------|
| c_7segment[2] | Output | PIN_E3 |
| c_7segment[1] | Output | PIN_E4 |
| c_7segment[0] | Output | PIN_D3 |
| d_7segment[6] | Output | PIN_F4 |
| d_7segment[5] | Output | PIN_D5 |
| d_7segment[4] | Output | PIN_D6 |
| d_7segment[3] | Output | PIN_J4 |
| d_7segment[2] | Output | PIN_L8 |
| d_7segment[1] | Output | PIN_F3 |
| d_7segment[0] | Output | PIN_D4 |
| in_sel[1] | Input | PIN_L2 |
| in_sel[0] | Input | PIN_M1 |
| input1[3] | Input | PIN_V12 |
| input1[2] | Input | PIN_M22 |
| input1[1] | Input | PIN_L21 |
| input1[0] | Input | PIN_L22 |
| input2[3] | Input | PIN_M2 |
| input2[2] | Input | PIN_U11 |
| input2[1] | Input | PIN_U12 |
| input2[0] | Input | PIN_W12 |

5. Lakukan kompilasi rancangan secara menyeluruh.
6. Download program ini kedalam board FPGA. Lalu lihat apakah program sudah berjalan dengan benar.
7. Catat hasil percobaan pada BCL Anda (catat minimal tiga input berbeda untuk setiap mode operasi kalkulator).

1.6 MENGAKHIRI PERCOBAAN

Prosedur untuk mengakhiri percobaan:

1. Sebelum keluar dari ruang praktikum, rapikan meja praktikum. **Rapikan kabel dan matikan komputer, osiloskop, generator sinyal, dan power supply DC.** Cabut daya dari jala-jala ke kit FPGA dan letakkan kembali pada tempat semula.
2. Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau merapikan meja ketika praktikum berakhir akan mendapatkan **potongan nilai sebesar minimal 10.**
3. **Pastikan asisten telah menandatangani catatan percobaan kali ini** pada Buku Catatan Laboratorium Anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

[illegible]

K-map for segment ____

| D1 D0 | | 00 | 01 | 11 | 10 |
|-------|----|----|----|----|----|
| D3 D2 | 00 | | | | |
| | 01 | | | | |
| | 11 | | | | |
| | 10 | | | | |

K-map for segment ____

| D1 D0 | | 00 | 01 | 11 | 10 |
|-------|----|----|----|----|----|
| D3 D2 | 00 | | | | |
| | 01 | | | | |
| | 11 | | | | |
| | 10 | | | | |

K-map for segment ____

| D1 D0 | | 00 | 01 | 11 | 10 |
|-------|----|----|----|----|----|
| D3 D2 | 00 | | | | |
| | 01 | | | | |
| | 11 | | | | |
| | 10 | | | | |

K-map for segment ____

| D1 D0 | | 00 | 01 | 11 | 10 |
|-------|----|----|----|----|----|
| D3 D2 | 00 | | | | |
| | 01 | | | | |
| | 11 | | | | |
| | 10 | | | | |

K-map for segment ____

| D1 D0 D3 D2 | | | | | |
|----------------|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| 00 | 01 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| 11 | 10 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

K-map for segment ____

| D1 D0 D3 D2 | | | | | |
|----------------|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| 00 | 01 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| 11 | 10 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

K-map for segment ____

| D1 D0 D3 D2 | | | | | |
|----------------|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| 00 | 01 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| 11 | 10 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

K-map for segment ____

| D1 D0 D3 D2 | | | | | |
|----------------|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| 00 | 01 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| 11 | 10 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

PERCOBAAN IV

RANGKAIAN LOGIKA SEKUENSIAL

1.1 TUJUAN

1. Mendesain sekuensial rangkaian untuk implementasi didalam FPGA.
2. Mengenal dan memahami cara menggunakan hierarki dalam desain rangkaian
3. Mengenal dan memahami cara menggunakan FPGA sebagai prototype system untuk memverifikasi fungsi rangkaian.

1.2 PERSIAPAN

Pelajari kembali bahan kuliah Anda mengenai rangkaian logika sekuensial. Pelajari juga keseluruhan petunjuk praktikum untuk modul rangkaian logika sekuensial ini. Kerjakan **tugas pendahuluan** dan kumpulkan sesuai ketentuan yang berlaku.

CATATAN

Untuk seluruh percobaan 4 ini, jika Anda menggunakan design skematik sesuai yang tertera di modul, maka nilai maksimal yang bisa Anda dapatkan adalah 75. Jika Anda menggunakan VHDL, maka nilai maksimal yang Anda dapatkan adalah 100. Code VHDL telah dipersiapkan praktikan sebelum praktikum. Jika ada pelanggaran berupa copy paste sebagian atau seluruh code VHDL, praktikan dikenai sanksi nilai nol dan/atau tidak lulus praktikum. Nilai tambah akan diberikan jika praktikan menggunakan simulator Modelsim®.

1.3 DASAR TEORI

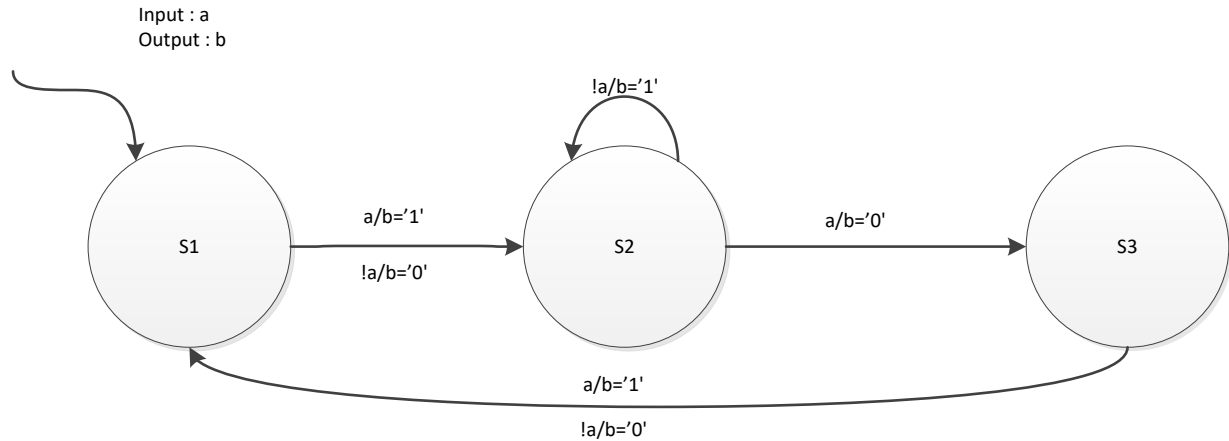
Pada praktikum sebelumnya praktikan telah merancang rangkaian kombinasional. Pada praktikum kali ini praktikan akan mencoba merancang rangkaian sekuensial. Perbedaan mendasar rangkaian kombinasional dengan rangkaian sekuensial adalah ada tidaknya memori statenya. Keluaran rangkaian sekuensial bergantung pada state dan bergantung pada masukannya (rangkaiannya Mealy) atau hanya bergantung pada statenya (rangkaiannya Moore).

Terdapat beberapa model yang digunakan untuk membantu merancang rangkaian sekuensial. Salah satunya yang paling banyak digunakan adalah Finite State Machine (FSM). Dinamakan FSM karena jumlah state yang mungkin terbatas dan rangkaian sekuensial bekerja mirip dengan mesin yang beroperasi dengan urutan state.

Level abstraksi perancangan FSM pun bertingkat-tingkat. Pada praktikum kali ini disarankan menggunakan level abstraksi behavioral. Pada perancangan dengan level ini, sebelum mengimplementasikan menggunakan VHDL, praktikan cukup membuat state diagram atau flow chart transisi statenya. Pada praktikum kali ini akan dicontohkan cara membuat FSM

dengan menggunakan state diagram. Komponen-komponen yang harus ada pada state diagram adalah deklarasi input dan output, definisi state, transisi, dan keluarannya.

Gambar di bawah adalah contoh gambar state diagram FSM Mealy dan implementasinya dalam VHDL.



```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY FSM IS
PORT (
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    a : IN STD_LOGIC;
    b : OUT STD_LOGIC
);
END FSM;

ARCHITECTURE behavioral OF FSM IS

    TYPE executionStage IS (s1,s2,s3);
    SIGNAL currentstate, nextstate: executionStage;

BEGIN

    PROCESS
    BEGIN
        WAIT UNTIL( clk'EVENT ) AND ( clk = '1' );
        IF ( rst = '0' ) THEN
            currentstate <= s1;
        ELSE
            currentstate <= nextstate;
        END IF;
    END PROCESS;

    PROCESS(a,currentstate)
    BEGIN
        CASE currentstate IS
            WHEN s1 =>
                IF (a = '1') THEN
                    b <= '1';
                ELSE
                    b <= '0';

```

```

        END IF;

        nextstate <= s2;

    WHEN s2 =>
        IF (a = '1') THEN
            b <= '1';
            nextstate <= s3;
        ELSE
            b <= '0';
            nextstate <= currentstate;
        END IF;

    WHEN s3 =>
        IF (a = '1') THEN
            b <= '1';
        ELSE
            b <= '0';
        END IF;

        nextstate <= s1;
    END CASE;
END PROCESS;
END behavioral;

```

1.4 TUGAS PENDAHULUAN

Buatlah FSM dari studi kasus di bawah ini dan lakukan simulasinya di rumah masing-masing (kerjakan dengan partner praktikum anda)!

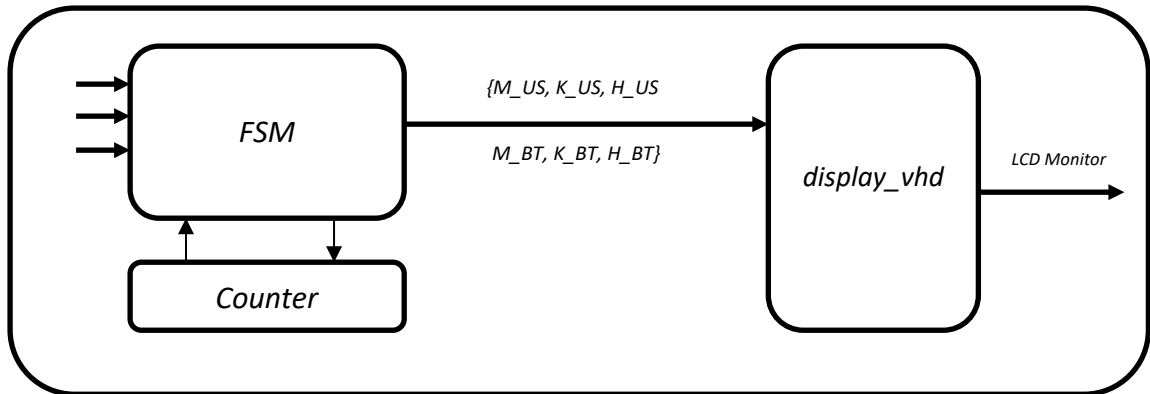
Bawalah bukti *script*, gambar FSM (dalam bentuk state diagram), dan hasil simulasi (dalam bentuk file vwf simulator quartus atau wlf modelsim) yang telah dibuat pada saat praktikum.

Sebuah perempatan jalan raya mempunyai 4 buah lampu lalu lintas. Lampu lalu lintas pada arah utara dan selatan menyala bersamaan. Lampu lalu lintas dari arah barat dan timur juga menyala bersamaan. Karena itu hanya diperlukan dua buah kontrol : (1) untuk lampu lalu lintas utara-selatan, dan (2) untuk lampu lalu lintas barat-timur.

Ketika siang hari lampu merah menyala selama 10 detik lalu lampu hijau menyala selama 8 detik, serta lampu kuning menyala selama 2 detik ketika perpindahan lampu hijau ke merah. Ketiga lampu menyala sendiri-sendiri (tidak ada lampu yang menyala bersamaan antara merah-kuning-hijau). Ketika malam hari lampu merah menyala selama 5 detik lalu lampu hijau menyala selama 4 detik serta lampu kuning menyala selama 1 detik ketika perpindahan lampu hijau ke merah.

Sistem lampu lalu lintas dilengkapi dengan tombol darurat. Ketika tombol darurat ditekan lampu kuning akan berkedip selama 4 detik.

Pada praktikum ini, praktikan harus mendesain FSM yang mengatur lampu lalu lintas tersebut. FSM yang didesain harus memiliki 6 buah output yang lebarnya masing-masing 1 bit. Output tersebut menunjukkan kondisi setiap lampu Utara Selatan dan lampu Barat Timur. Misalkan M_US (Merah Utara Selatan), M_BT (Merah Barat Timur), K_US, K_BT, H_US, dan H_BT.



Gambar di atas adalah gambaran blok-blok yang harus dibuat kecuali untuk blok display_vhd karena blok ini sudah disediakan sebagai modul display ke LCD via VGA (Blok ini akan dipelajari lebih lanjut pada praktikum berikutnya). Input dari FSM yang dibuat adalah mode siang, mode malam hari, dan mode darurat. Blok counter sendiri untuk menentukan lama waktu lampu lalu lintas nyala atau transisi.

Tugas pendahuluan di bawah ini merupakan tugas pendahuluan tambahan dengan materi yang berkaitan dengan percobaan tambahan modul 4D – Perancangan Kalkulator dengan Operasi Spesifik.

1. Ubahlah blok komponen FSM yang terdapat pada modul percobaan 4D untuk dapat melakukan operasi modulo seperti yang dijelaskan pada modul.
2. Lakukan kompilasi *Analysis & Synthesis* dari project kalkulator sederhana (blok komponen lain seperti *top-level* dapat dilihat pada modul tambahan percobaan 4D).
3. Lakukan simulasi *functional* untuk beberapa nilai input yang berbeda.

PERHATIKAN ! Segala bentuk plagiarisme dalam pengerjaan tugas pendahuluan ini akan diberikan sanksi yang tegas.

1.5 PERCOBAAN

PERALATAN YANG DIGUNAKAN

- Komputer/PC yang telah terinstal program Quartus II 9.0
- Monitor LCD
- FPGA development board, tipe ALTERA DE1 beserta perlengkapannya yang meliputi:
 - a. Board FPGA tipe DE1
 - b. Catu daya+ kabel dan konektor tambahan

- c. Kabel USB-Blaster

PROSEDUR PERCOBAAN

Untuk tahapan percobaan Anda akan mendesain dan menguji (dalam simulasi) sebuah BCD counter yang dapat di-cascade dan sebuah divide-by-N counter.

PERCOBAAN 4A: IMPLEMENTASI DESAIN FSM PADA FPGA

Percobaan ini, praktikan diminta untuk mengimplementasikan FSM ke FPGA dengan **keluaran ke LED FPGA**.

PROSEDUR PERCOBAAN:

1. Buatlah folder sebagai direktori kerja baru untuk praktikum kali ini kemudian *copy script* desain FSM yang telah dibuat sebagai tugas pendahuluan sebelumnya ke dalam folder tersebut.
2. Jalankan program **ALTERA QUARTUS®**, kemudian bukalah file yang merupakan *script* desain FSM yang telah dibuat sebagai tugas pendahuluan sebelumnya.
3. Implementasikan desain FSM tersebut dengan keluaran pada LED FPGA (lihat kembali implementasi desain pada FPGA dalam praktikum-praktikum sebelumnya).
4. Catat keluaran LED FPGA untuk mode siang, malam, dan darurat pada BCL anda!

PERCOBAAN 4B: IMPLEMENTASI MODUL VGA DRIVER

Praktikum kali ini, praktikan diminta untuk mengimplementasikan modul VGA Drive dengan masukan dari FPGA (Switch) selebar 6 bit. Masukan modul ini akan dihubungkan dengan keluaran modul FSM yang telah dibuat.

PROSEDUR PERCOBAAN:

1. Buatlah folder sebagai direktori kerja baru untuk percobaan ini, kemudian buatlah project baru dengan modul-modul yang disediakan untuk praktikum ("**display_DE1.rar**"). **Catatan : download di web labdasar!**
2. Implementasikan desain di atas pada FPGA dengan *pin planner* yang sudah didownload (deklarasi pin dapat dilihat di file "qsf").
3. Lakukan beberapa kali perubahan posisi *switch* pada board FPGA untuk melihat efek dan perubahannya pada layar LCD! Pelajari input dan keluaran dari desain di atas untuk selanjutnya digabungkan dengan modul FSM yang telah dibuat. **Catatan : jangan lupa untuk menghubungkan port VGA FPGA board dengan VGA LCD menggunakan kabel VGA DB15.**
4. Catat tampilan pada layar LCD untuk berbagai variasi input pada BCL anda!

PERCOBAAN 4C: MENGGABUNGKAN DESAIN FSM DENGAN VGA DRIVER

Pada praktikum kali ini, praktikan diminta untuk menggabungkan modul FSM dengan modul VGA.

PROSEDUR PERCOBAAN:

1. Hubungkan keluaran FSM dengan masukan modul VGA. Lakukan *compile* dan download gabungan desain FSM dan modul VGA tersebut ke dalam board FPGA. Amati hasil yang didapatkan !
2. Catat tampilan pada layar LCD untuk mode siang, malam, dan darurat!

PERCOBAAN 4D: MERANCANG SEBUAH KALKULATOR UNTUK MELAKUKAN OPERASI SPESIFIK

Pada percobaan ini, akan dibuat suatu kalkulator yang dapat melakukan suatu operasi spesifik dengan menggunakan operasi-operasi dasar. Operasi spesifik yang akan dibuat ada dua jenis, yaitu :

1. Perhitungan GCD (great common divisor) atau FPB (faktor persekutuan terbesar)

Algoritma perhitungan GCD yang digunakan adalah membandingkan kedua input bilangan, apakah input pertama (x) sama dengan input kedua (y). Jika iya, maka GCD telah ditemukan (bisa x atau y). Jika $x > y$, maka $x = x - y$, jika $y > x$, maka $y = y - x$. Lalu kedua bilangan tersebut dikomparasi berulang-ulang hingga ditemukan GCD (kondisi $x = y$). Catatan : kedua input tidak boleh 0.

2. Perhitungan modulo (sisanya pembagian)

Algoritma perhitungan modulo ($x \bmod y$) yang digunakan adalah dengan melakukan komparasi apakah $x < y$, jika iya, maka modulo telah didapatkan, yaitu x. Jika $x > y$, maka $x = x - y$. Kemudian dilakukan komparasi kembali hingga modulo telah didapatkan. Catatan : input kedua tidak boleh 0.

Arsitektur program ini disusun dalam 6 block, yaitu subtractor (untuk melakukan operasi pengurangan), comparator (untuk melakukan operasi komparasi), mux (sebagai selector), regis (sebagai register), fsm (sebagai kontroler), dan kalkulator (sebagai top-level). Block-block tersebut akan diberikan secara lengkap untuk jenis operasi GCD..

PROSEDUR PERCOBAAN:

1. Buatlah project baru pada folder yang berbeda dengan percobaan-percobaan sebelumnya.
2. Buatlah file-file VHDL untuk masing-masing blok dengan mengetikkan script di bawah ini, dan simpan dalam folder project yang telah dibuat. Perhatikan bahwa nama masing-masing file VHDL harus sama dengan nama *entity* pada program.
 - A. Blok *kalkulator / top-level* (simpan dengan nama file kalkulator.vhd)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity kalkulator is
    port( rst, clk, go_i: in std_logic;
```

```

        x_i, y_i: in std_logic_vector( 3 downto 0 );
        d_o: out std_logic_vector( 3 downto 0 )
    );
end kalkulator;

architecture kalkulator_arc of kalkulator is

    component fsm is
        port( rst, clk, proceed: in std_logic;
              comparison: in std_logic_vector( 1 downto 0 );
              enable, xsel, ysel, xld, yld: out std_logic
            );
    end component;

    component mux is
        port( rst, sLine: in std_logic;
              load, result: in std_logic_vector( 3 downto 0 );
              output: out std_logic_vector( 3 downto 0 )
            );
    end component;

    component comparator is
        port( rst: in std_logic;
              x, y: in std_logic_vector( 3 downto 0 );
              output: out std_logic_vector( 1 downto 0 )
            );
    end component;

    component subtractor is
        port( rst: in std_logic;
              cmd: in std_logic_vector( 1 downto 0 );
              x, y: in std_logic_vector( 3 downto 0 );
              xout, yout: out std_logic_vector( 3 downto 0 )
            );
    end component;

    component regis is
        port( rst, clk, load: in std_logic;
              input: in std_logic_vector( 3 downto 0 );
              output: out std_logic_vector( 3 downto 0 )
            );
    end component;

    signal xld, yld, xsel, ysel, enable: std_logic;
    signal comparison: std_logic_vector( 1 downto 0 );
    signal result: std_logic_vector( 3 downto 0 );

    signal xsub, ysub, xmux, ymux, xreg, yreg: std_logic_vector( 3
downto 0 );

begin
    -- FSM controller
    TOFSM: fsm port map( rst, clk, go_i, comparison,
enable, xsel, ysel, xld, yld );
    -- Datapath
    X_MUX: mux port map( rst, xsel, x_i, xsub, xmux );
    Y_MUX: mux port map( rst, ysel, y_i, ysub, ymux );
    X_REG: regis port map( rst, clk, xld, xmux, xreg );
    Y_REG: regis port map( rst, clk, yld, ymux, yreg );

```



```

        U_COMP: comparator port map( rst, xreg, yreg, comparison
        );
        X_SUB: subtractor port map( rst, comparison, xreg, yreg,
xsub, ysub );
        OUT_REG: regis port map( rst, clk, enable, xsub, result
        );

        d_o <= result;

    end kalkulator_arc;

```

B. Blok *mux* (simpan dengan nama file mux.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mux is
    port( rst, sLine: in std_logic;
        load, result: in std_logic_vector( 3 downto 0 );
        output: out std_logic_vector( 3 downto 0 )
    );
end mux;

architecture mux_arc of mux is
begin
    process( rst, sLine, load, result )
    begin
        if( rst = '1' ) then
            output <= "0000"; -- do nothing
        elsif sLine = '0' then
            output <= load; -- load inputs
        else
            output <= result; -- load results
        end if;
    end process;
end mux_arc;

```

C. Blok *comparator* (simpan dengan nama file comparator.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity comparator is
    port( rst: in std_logic;
        x, y: in std_logic_vector( 3 downto 0 );
        output: out std_logic_vector( 1 downto 0 )
    );
end comparator;

architecture comparator_arc of comparator is
begin
    process( x, y, rst )
    begin
        if( rst = '1' ) then
            output <= "00"; -- do nothing
        elsif( x > y ) then
            output <= "10"; -- if x greater

```

```

        elsif( x < y ) then
            output <= "01"; -- if y greater
        else
            output <= "11"; -- if equivalence.
        end if;
    end process;
end comparator_arc;

```

D. Blok *register* (simpan dengan nama regis.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity regis is
    port( rst, clk, load: in std_logic;
          input: in std_logic_vector( 3 downto 0 );
          output: out std_logic_vector( 3 downto 0 )
    );
end regis;

architecture regis_arc of regis is
begin
    process( rst, clk, load, input )
    begin
        if( rst = '1' ) then
            output <= "0000";
        elsif( clk'event and clk = '1' ) then
            if( load = '1' ) then
                output <= input;
            end if;
        end if;
    end process;
end regis_arc;

```

E. Blok *subtractor* (simpan dengan nama file subtractor.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity subtractor is
    port( rst: in std_logic;
          cmd: in std_logic_vector( 1 downto 0 );
          x, y: in std_logic_vector( 3 downto 0 );
          xout, yout: out std_logic_vector( 3 downto 0 )
    );
end subtractor;

architecture subtractor_arc of subtractor is
begin
    process( rst, cmd, x, y )
    begin
        if( rst = '1' or cmd = "00" ) then -- not active.
            xout <= "0000";
            yout <= "0000";
        elsif( cmd = "10" ) then -- x is greater
            xout <= ( x - y );
            yout <= y;
        elsif( cmd = "01" ) then -- y is greater

```

```

        xout <= x;
        yout <= ( y - x );
    else
        xout <= x; -- x and y are equal
        yout <= y;
    end if;
end process;
end subtractor_arc;

```

F. Blok *fsm* (simpan dengan nama file fsm.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fsm is
    port( rst, clk, proceed: in std_logic;
          comparison: in std_logic_vector( 1 downto 0 );
          enable, xsel, ysel, xld, yld: out std_logic
        );
end fsm;

architecture fsm_arc of fsm is

    type states is ( init, s0, s1, s2, s3, s4, s5 );
    signal nState, cState: states;

begin
    process( rst, clk )
    begin
        if( rst = '1' ) then
            cState <= init;
        elsif( clk'event and clk = '1' ) then
            cState <= nState;
        end if;
    end process;

    process( proceed, comparison, cState )
    begin
        case cState is

            when init => if( proceed = '0' ) then
                nState <= init;
            else
                nState <= s0;
            end if;

            when s0 => enable <= '0';
                xsel <= '0';
                ysel <= '0';
                xld <= '0';
                yld <= '0';
                nState <= s1;

            when s1 => enable <= '0';
                xsel <= '0';
                ysel <= '0';
                xld <= '1';
                yld <= '1';
                nState <= s2;

```

```

when s2 => xld <= '0';
    yld <= '0';
    if( comparison = "10" ) then
        nState <= s3;
    elsif( comparison = "01" ) then
        nState <= s4;
    elsif( comparison = "11" ) then
        nState <= s5;
    end if;

when s3 => enable <= '0';
    xsel <= '1';
    ysel <= '0';
    xld <= '1';
    yld <= '0';
    nState <= s2;

when s4 => enable <= '0';
    xsel <= '0';
    ysel <= '1';
    xld <= '0';
    yld <= '1';
    nState <= s2;

when s5 => enable <= '1';
    xsel <= '1';
    ysel <= '1';
    xld <= '1';
    yld <= '1';
    nState <= s0;

when others => nState <= s0;
end case;

end process;

end fsm_arc;

```

3. Lakukan kompilasi Analysis & Synthesis.
4. Lakukan kompilasi rancangan secara menyeluruh.
5. Lakukan simulasi timing untuk beberapa nilai input. (Gunakanlah tipe data unsigned untuk kedua sinyal input dan output agar mempermudah pembacaan)
6. Catat hasil percobaan pada BCL Anda (minimal tiga input berbeda untuk satu mode operasi).
7. Lakukan kembali langkah-langkah sebelumnya untuk membuat kalkulator modulo, namun modifikasilah program pada block fsm untuk operasi modulo.

1.6 MENGAKHIRI PERCOBAAN

1. Sebelum keluar dari ruang praktikum, rapikan meja praktikum. **Rapikan kabel dan matikan komputer, osiloskop, generator sinyal, dan power supply DC.** Cabut daya dari jala-jala ke kit FPGA dan letakkan kembali pada tempat semula.

2. Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau merapikan meja ketika praktikum berakhir akan mendapatkan **potongan nilai sebesar minimal 10**.
3. **Pastikan asisten telah menandatangani catatan percobaan kali ini** pada Buku Catatan Laboratorium Anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

PERCOBAAN V

PERANCANGAN DAN IMPLEMENTASI DISPLAY LCD MENGGUNAKAN MODUL VGA PADA FPGA

1.1 TUJUAN

1. Mendapatkan pengetahuan dan pengalaman menggunakan interface pada board evaluasi FPGA.
2. Memahami cara kerja VGA pada umumnya

1.2 PERSIAPAN

Pelajari secara rinci spesifikasi VGA dan cara kerjanya. Pelajari juga petunjuk praktikum kelima ini.

Kerjakan Tugas Pendahuluan dan kumpulkan sesuai ketentuan yang berlaku.

CATATAN

Jika ada pelanggaran berupa copy paste sebagian atau seluruh code VHDL, praktikan dikenai sanksi nilai nol dan/atau tidak lulus praktikum.

1.3 DASAR TEORI

Video Graphics Array (VGA) masih menjadi interface yang populer untuk sebuah tampilan. VGA interface ini masih banyak ditemukan di beberapa device sekarang, misalnya layar LCD dan proyektor. VGA interface ini terdapat juga di board altera yang kita gunakan saat ini. Pada percobaan kali ini tampilan VGA digunakan agar tampilan hasil desain yang kita rancang menjadi lebih menarik, tidak terbatas hanya pada LED atau 7-Segment. Tujuan percobaan kali ini juga adalah memberikan ilustrasi penggunaan interface I/O yang ada pada FPGA, misalnya GPIO, komunikasi serial menggunakan RS232, Audio CODEC, LCD karakter 16x2, dll.

Interface ke VGA menggunakan 2 jenis sinyal, yaitu : sinyal warna (Merah, Hijau, dan Biru) dan sinyal sinkron (horizontal dan vertical). Berikut adalah penjelasan beberapa sinyal yang digunakan :

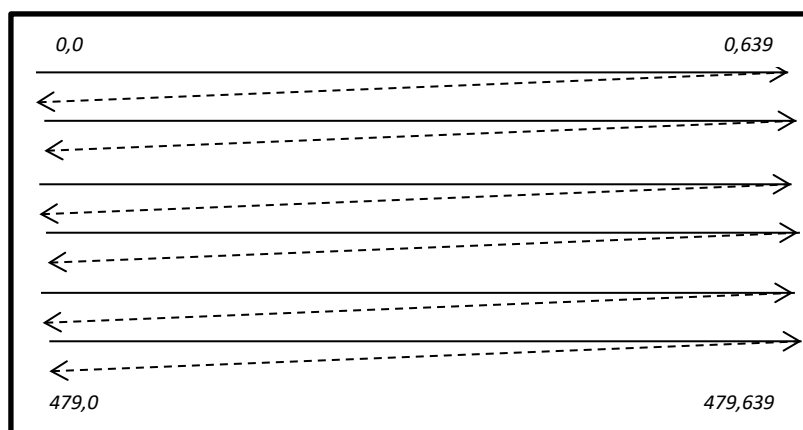
- a. Horizontal Sync (TTL level)
Sinyal ini akan aktif pada range piksel kolom 0 sampai dengan 639.
Sehingga kalau sinyal ini tidak aktif, yang terjadi adalah pergantian baris.
- b. Vertical Sync (TTL level)
Sinyal ini akan aktif pada range piksel baris 0 sampai dengan 479.

Sehingga kalau sinyal ini tidak aktif, yang terjadi adalah pergantian layar. Atau kembali ke baris pertama.

c. Sinyal RGB (Analog 3 pin: 0,7 – 1 V)

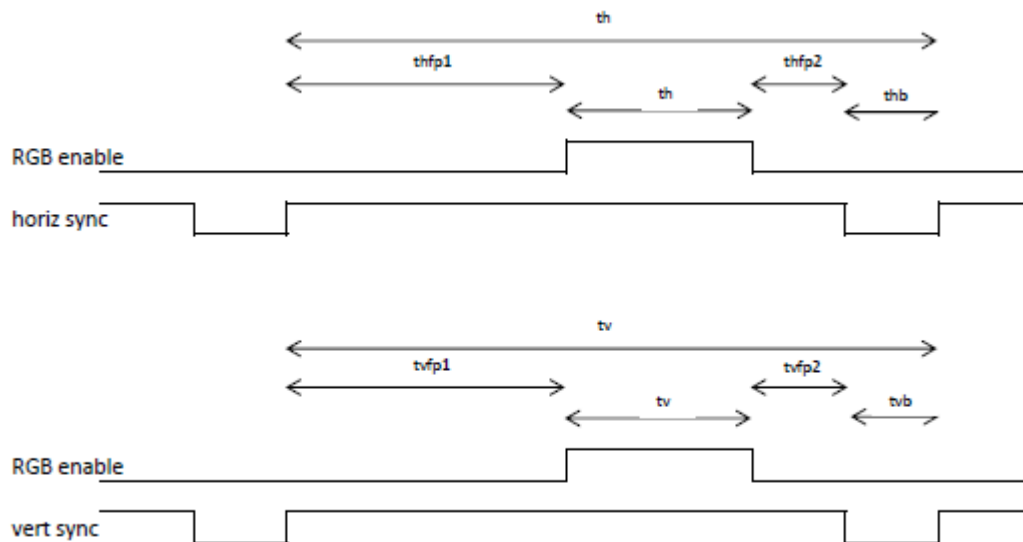
Sinyal ini merepresentasikan intensitas untuk masing2 komponen warna merah, hijau, dan biru untuk setiap pixel yang saat itu aktif. Sehingga yang terjadi ketiga sinyal ini berubah-ubah sesuai pixel yang sedang aktif dalam proses scanning (dari kiri ke kanan untuk setiap baris, selanjutnya dari baris paling atas sampai baris paling bawah).

Pada percobaan kali ini kita menggunakan resolusi 640x480 pixel dan menggunakan refresh rate lebih dari 60 Hz. Refresh rate ini digunakan karena pada range kurang dari 30-60 Hz manusia dapat melihat adanya flicker. Selain itu refresh rate ini juga umum digunakan pada monitor LCD. LCD modern memiliki fitur multirate, sehingga kita tidak harus tepat membuat refresh ratenya 60 Hz. Proses scanning berawal dari kiri atas ke kanan lalu ke kiri bawah dan kembali ke kiri atas ketika sudah mencapai pixel terakhir.



Gambar 1. Razor Scan pada Layar LCD

Gambar 2 dan Tabel 1 menunjukkan spesifikasi timing dari sinkronisasi VGA. Sebuah sinyal aktif low menunjukkan akhir dari sebuah sinkronisasi. Misalkan sinyal aktif low untuk *horiz sync* menandakan akhir dari scanning satu baris dan awal untuk baris berikutnya. Data RGB harus didrive 0 untuk beberapa waktu tertentu *thfp* dan *tvfp*.

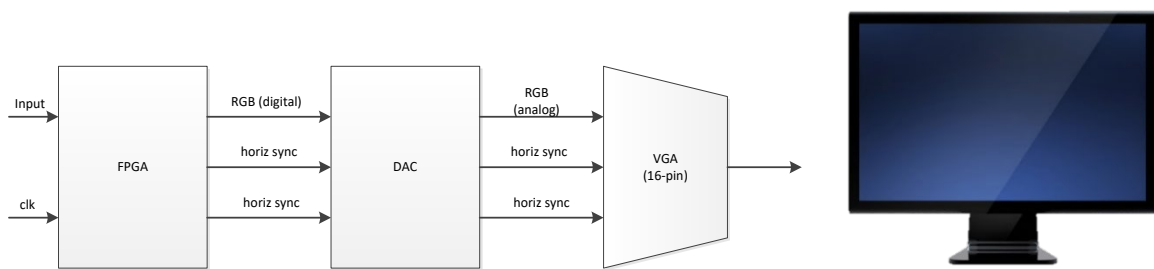


Gambar 2. Timing Sinyal untuk VGA 640x480 piksel

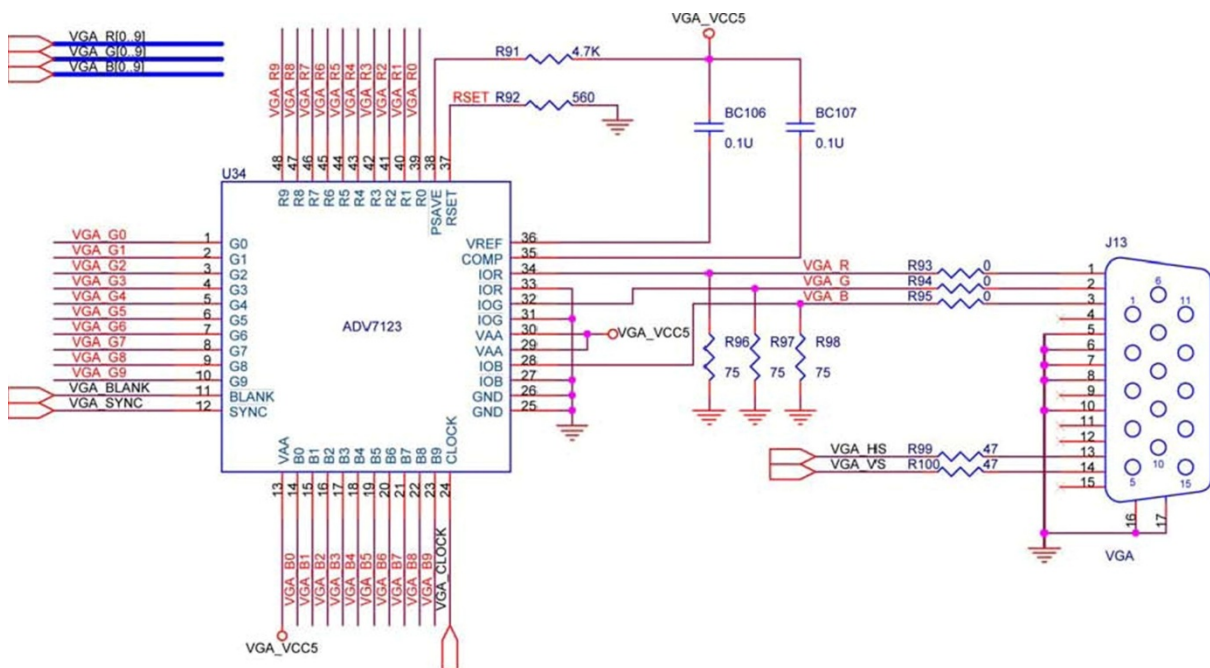
| SYMBOL | Min | Typ | Max | Unit |
|--------|-----|-----|-----|------|
| thd | | 640 | | DCLK |
| fclk | | 24 | 50 | MHz |
| th | | 760 | | DCLK |
| thpw | 1 | 48 | 255 | DCLK |
| thb | | 88 | | DCLK |
| thfp | 1 | 32 | 255 | DCLK |
| th-thd | 85 | 120 | 512 | DCLK |
| tdv | | 480 | | th |
| tv | 513 | 525 | 767 | th |
| tvpw | 3 | 3 | 255 | th |
| tvb | | 32 | | th |
| tvfp | 1 | 13 | 255 | th |
| tv-tvd | 4 | 45 | 255 | th |

Tabel 1. Nilai-nilai parameter pada Gambar 1

Gambar 3 menunjukkan blok diagram dari FPGA hingga ke LCD monitor. Chip DAC mengubah sinyal digital ke analog. Dalam kasus ini, data RGB digital diubah ke data RGB analog, begitu juga untuk sinyal sinkronisasinya. Sedangkan gambar 4 menunjukkan skematik dari display VGA yang ada pada board DE1. Untuk board lainnya dapat dibaca di datasheet masing-masing board. Board DE1 menyediakan 16-pin konektor untuk output VGA dan Analog Devices ADV7123 10-bit high speed video DAC. DAC ini mendapatkan sinyal sinkronisasi dari FPGA.



Gambar 3. Diagram Blok VGA Display



Gambar 4. Skematik VGA Display

1.4 TUGAS PENDAHULUAN

1. Hitunglah nilai-nilai parameter (dalam satuan waktu) yang ditunjukkan pada gambar timing signal untuk VGA, dengan menggunakan clock sebesar 25Mhz ! Gunakan file **vga.vhd** sebagai acuan untuk menghitung nilai tersebut !
2. Jelaskan mengenai isi dan cara kerja file **vga.vhd**!

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY vga IS
PORT (
    i_clk      : IN STD_LOGIC;
    i_red      : IN STD_LOGIC;
    i_green    : IN STD_LOGIC;

```

```

        i_blue           : IN STD_LOGIC;
        o_red            : OUT STD_LOGIC;
        o_green          : OUT STD_LOGIC;
        o_blue           : OUT STD_LOGIC;
        o_horiz_sync     : OUT STD_LOGIC;
        o_vert_sync      : OUT STD_LOGIC;
        o_pixel_row       : OUT STD_LOGIC_VECTOR( 9 DOWNTO 0 );
        o_pixel_column    : OUT STD_LOGIC_VECTOR( 9 DOWNTO 0 ));
END vga;

```

ARCHITECTURE behavioral **OF** vga **IS**

```

    CONSTANT TH      : INTEGER := 800;
    CONSTANT THB1    : INTEGER := 660;
    CONSTANT THB2    : INTEGER := 756;
    CONSTANT THD      : INTEGER := 640;

```

```

    CONSTANT TV      : INTEGER := 525;
    CONSTANT TVB1    : INTEGER := 494;
    CONSTANT TVB2    : INTEGER := 495;
    CONSTANT TVD      : INTEGER := 480;

```

```

    SIGNAL clock_25MHz : STD_LOGIC;
    SIGNAL horiz_sync  : STD_LOGIC;
    SIGNAL vert_sync   : STD_LOGIC;
    SIGNAL video_on     : STD_LOGIC;
    SIGNAL video_on_v   : STD_LOGIC;
    SIGNAL video_on_h   : STD_LOGIC;
    SIGNAL h_count      : STD_LOGIC_VECTOR( 9 DOWNTO 0 );
    SIGNAL v_count      : STD_LOGIC_VECTOR( 9 DOWNTO 0 );

```

BEGIN

```

    video_on    <= video_on_h AND video_on_v;

```

```

    o_red       <= i_red AND video_on;
    o_green     <= i_green AND video_on;
    o_blue      <= i_blue AND video_on;

```

```

    o_horiz_sync <= horiz_sync;
    o_vert_sync  <= vert_sync;

```

PROCESS (i_clk)

BEGIN

```

        IF i_clk'EVENT AND i_clk='1'THEN
            IF (clock_25MHz = '0') THEN
                clock_25MHz <= '1';
            ELSE
                clock_25MHz <= '0';
            END IF;
        END IF;

```

END PROCESS;

PROCESS

BEGIN

```

    WAIT UNTIL( clock_25MHz'EVENT ) AND ( clock_25MHz = '1' );

```

```

    IF ( h_count = TH-1 ) THEN
        h_count <= (others=>'0');
    ELSE

```

```

        h_count <= h_count + 1;
    END IF;

```

```

    IF ( h_count <= THB2-1 ) AND (h_count >= THB1-1 ) THEN
        horiz_sync <= '0';
    ELSE

```

```

        horiz_sync <= '1';
    END IF;

```

```

IF ( v_count >= TV-1 ) AND ( h_count >= 699 ) THEN
    v_count <= (others=>'0');
ELSE IF ( h_count = 699 ) THEN
    v_count <= v_count + 1;
END IF;

END IF;

IF ( v_count <= TVB2-1 ) AND ( v_count >= TVB1-1 ) THEN
    vert_sync <= '0';
ELSE
    vert_sync <= '1';
END IF;

IF ( h_count <= THD-1 ) THEN
    video_on_h <= '1';
    o_pixel_column <= h_count;
ELSE
    video_on_h <= '0';
END IF;

IF ( v_count <= TVD-1 ) THEN
    video_on_v <= '1';
    o_pixel_row <= v_count;
ELSE
    video_on_v <= '0';
END IF;

END PROCESS;

END behavioral;

```

1.5 PERCOBAAN

PERALATAN YANG DIGUNAKAN

- Board FPGA tipe DE1
- Catu daya + kabel dan konektor tambahan serta kabel *downloader*
- Monitor LCD

PROSEDUR PERCOBAAN

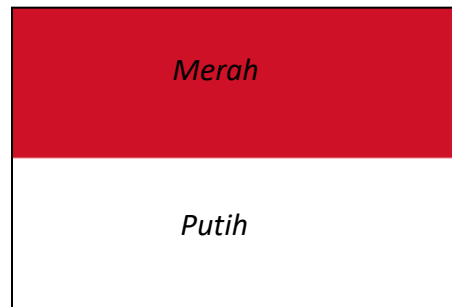
PERCOBAAN 5A : IMPLEMENTASI DESAIN PADA BOARD FPGA

Pada percobaan pertama ini, praktikan diminta membuat controller VGA sederhana yang mengeluarkan sinyal-sinyal digital untuk mengendalikan VGA. Sinyal digital ini sebagian akan diubah menjadi sinyal analog (untuk yang warna-warna RGB). Masing-masing warna akan diwakili 6 bit. Praktikan dapat menggunakan file **vga.vhd** yang telah ada sebagai template.

PROSEDUR PERCOBAAN 1

Gambarlah bendera RI dilayar (atas merah, bawah putih) pada layar VGA. Untuk mendapatkan warna merah R = 111111, G = B = 000000, sedangkan putih R = G = B =

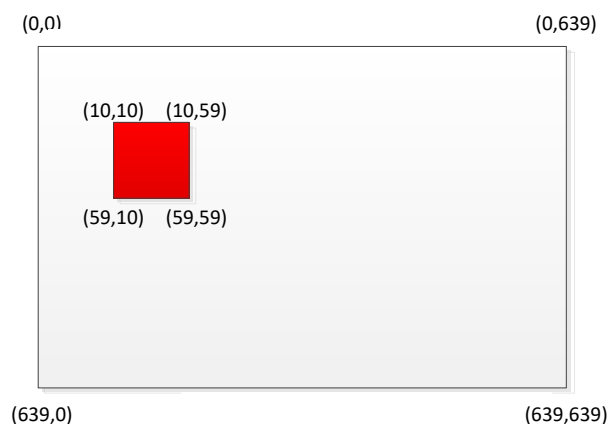
111111. Cara yang dapat dilakukan adalah dengan cara mengeluarkan warna merah untuk pixel-pixel pada baris atas (nomor baris < 241), dan warna putih untuk baris bawah.



Gambar 4. Ilustrasi Tampilan Prosedur Percobaan 1

PROSEDUR PERCOBAAN 2

Gambarlah bendera sebuah kotak/bujur sangkar (solid) berukuran 50 pixel x 50 pixel pada layar VGA. Caranya adalah dengan member warna tertentu pada pixel-pixel tertentu. Misalkan, jika ujung kiri atas kotak tadi ingin diletakan pada baris 10 kolom 10, maka pixel yang harus diwarnai berbeda dengan lainnya adalah semua pixel yang ada baris 10 sampai 59 dan kolom 10 sampai 59.



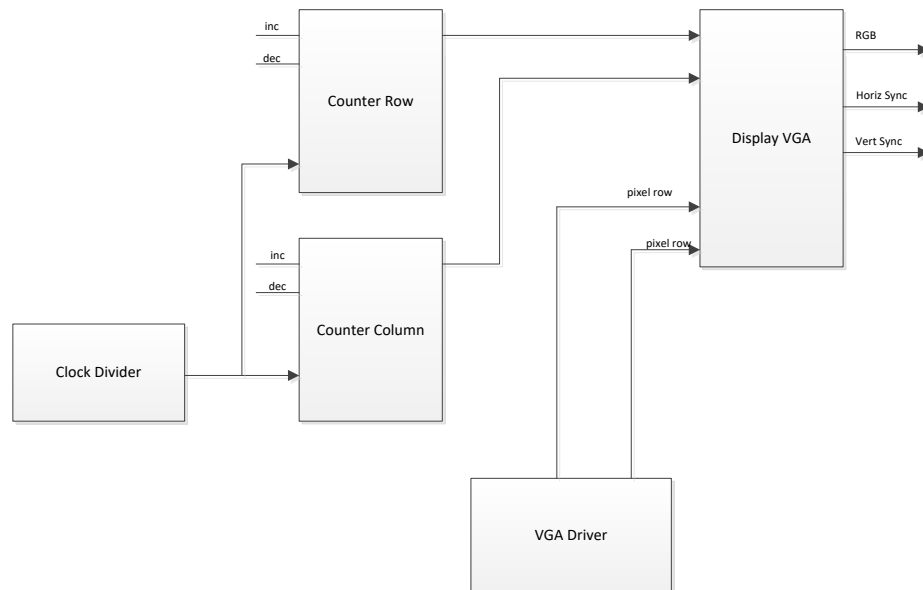
Gambar 5. Ilustrasi Tampilan Prosedur Percobaan 2

PROSEDUR PERCOBAAN 3

Buatlah agar gambar kotak yang telah anda buat agar dapat bergerak berdasar kan input dari *push-button* yang ada di board. Kotak ini harus dapat digerakan ke empat arah: kiri, kanan, atas, bawah dengan empat button yang berbeda-beda. Caranya:

1. Ujung kiri atas dari gambar tersebut harus dibuat agar dapat diubah-ubah (menjadi input)

2. Membuat dua buah FSM/counter: satu FSM untuk menghasilkan posisi batas atas (baris), satu FSM untuk menghasilkan posisi batas kiri. Tentu saja counter ini harus dibatasi maksimum dan minimumnya sesuai jumlah baris dan kolom yang ada di layar. Untuk setiap FSM harus dapat menghitung maju (up-counting: ... \rightarrow 100 \rightarrow 101 \rightarrow 102 \rightarrow ...) dan mundur (down counting: ... \rightarrow 87 \rightarrow 86 \rightarrow 85 \rightarrow ...). FSM ini harus diclock, namun tidak boleh terlalu cepat agar gerakan kotak tadi juga tidak terlalu cepat. Misalnya 20 Hz – 50 Hz.
3. Membuat input untuk perintah up/down counting pada kedua FSM menggunakan push-button.



Gambar 6. Ilustrasi Blok Diagram Prosedur Percobaan 3

Gambar 6 merupakan diagram blok kasar yang mungkin untuk mengimplementasikan prosedur percobaan 3. Clock Divider di sini berguna agar masukan oleh user tidak terlalu cepat dan efeknya hasil pergerakan objek gambar dapat ditangkap oleh mata. Dengan informasi dari posisi objek dan posisi alamat piksel dari VGA driver cukup untuk mengimplementasikan prosedur percobaan 3 ini.

TUGAS BONUS: membuat agar kecepatan bergerak kotak tersebut dapat diubah-ubah

1.6 MENGAKHIRI PERCOBAAN

1. Sebelum keluar dari ruang praktikum, rapikan meja praktikum. **Rapikan kabel dan matikan komputer, osiloskop, generator sinyal, dan power supply DC.** Cabut daya dari jala-jala ke kit FPGA dan letakkan kembali pada tempat semula.
2. Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau merapikan meja ketika praktikum berakhir akan mendapatkan potongan nilai sebesar minimal 10.
3. **Pastikan asisten telah menandatangani catatan percobaan kali ini** pada Buku Catatan Laboratorium Anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai

SURAT PERNYATAAN

EL-2102 : PRAKTIKUM SISTEM DIGITAL

Surat pernyataan ini dibuat oleh :

Nama : _____

NIM : _____

Tulislah persentasi usaha dimana jumlah persentase antara Anda dengan rekan tim Anda adalah 100%. Tulislah bagian pekerjaan yang dikerjakan oleh anggota tim dan komentar pribadi Anda pada kolom dibawah nama masing-masing.

Nama Anggota Tim(Anda)

Persentase usaha

| | |
|-------|-------|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

Nama Anggota Tim(Rekan Anda)

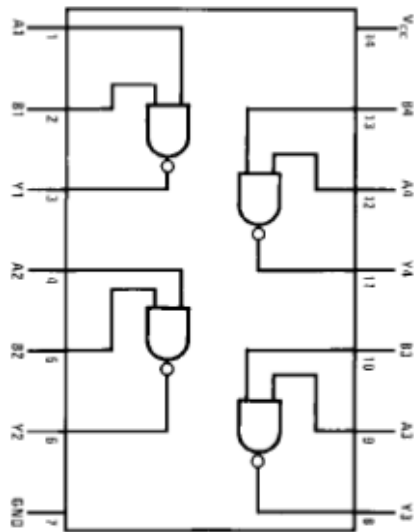
Persentase usaha

| | |
|-------|-------|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

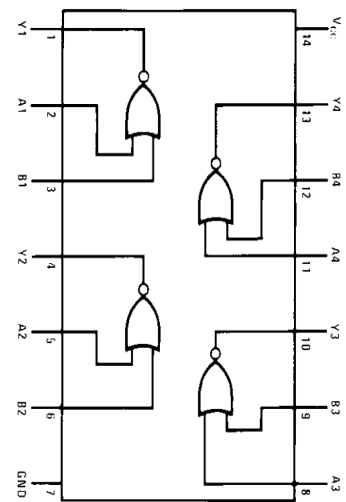
APENDIKS A

PENJELASAN KAKI GERBANG LOGIKA

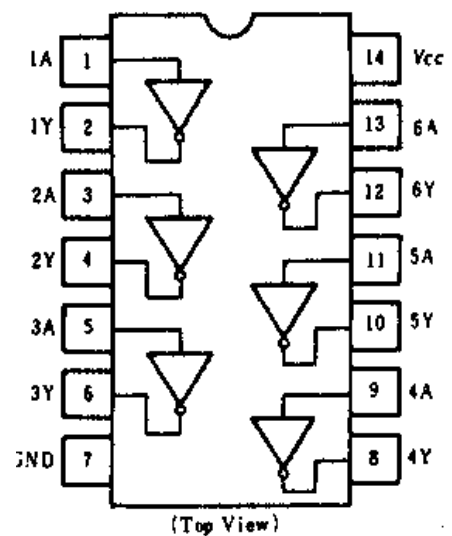
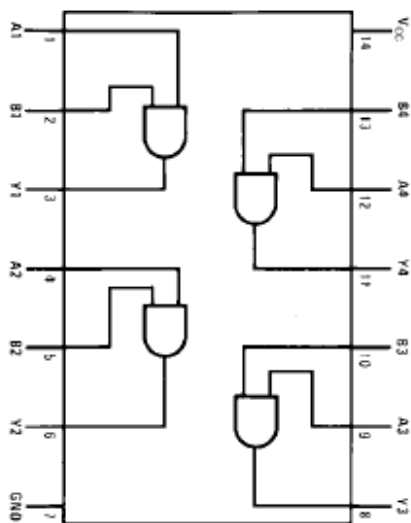
74LS00 2 INPUT NAND GATE



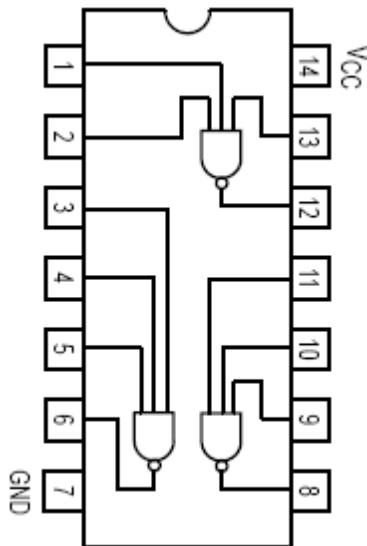
74LS02 2 INPUT NOR GATE



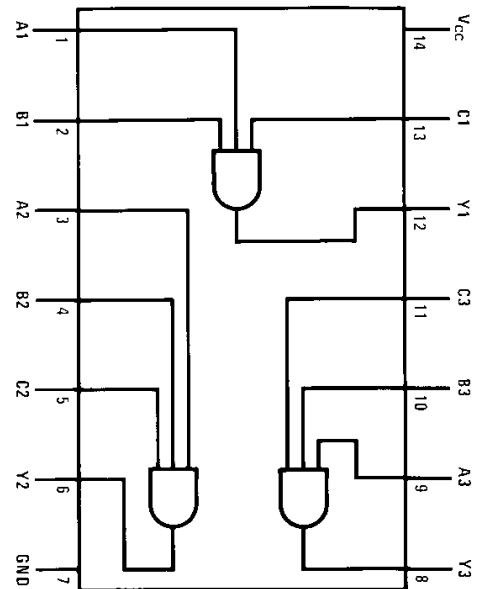
74LS08 2-INPUT AND GATE 74LS04 INVERTER GATE



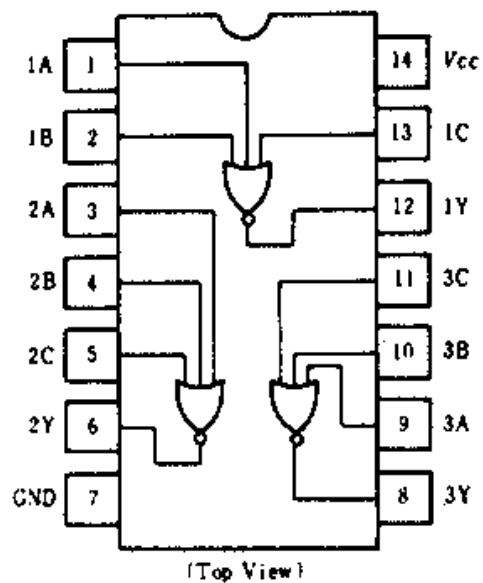
74LS10 3-INPUT NAND GATE



74LS11 3-INPUT AND GATE



74LS27 3-INPUT NOR GATE



APENDIKS B

VHDL REFERENCE

STRUKTUR KODE VHDL

Berikut adalah struktur umum yang akan sering digunakan selama praktikum. **Ingat VHDL merupakan bahasa case-insensitive dan strongly-typed!**

```
-----  
--  
-- Header  
-- Nama File : nama_entity.vhd  
-----  
--  
  
LIBRARY nama_library;  
USE nama_library.nama_package.all;  
  
ENTITY nama_entity IS  
    GENERIC (  
        nama_signal1 : nama_type := nilai;  
        nama_signal2 : nama_type := nilai;  
        ...  
        nama_signal3 : nama_type := nilai  
    );  
    PORT (  
        nama_port1 : [mode] nama_type;  
        nama_port2 : [mode] nama_type;  
        ...  
        nama_port3 : [mode] nama_type  
    );  
END nama_entity;  
  
ARCHITECTURE nama_architecture OF nama_entity IS  
    [SIGNAL deklarasi];  
    [CONSTANT      deklarasi];  
    [TYPE          deklarasi];  
    [COMPONENT     deklarasi];  
    [ATTRIBUTE     deklarasi];  
  
BEGIN  
    { COMPONENT                instantiasi statement; }  
    { CONCURRENT ASSIGNMENT    statement; }  
    { PROCESS                  statement; }  
    { GENERATE                  statement; }  
END nama_architecture;
```

LIBRARY DAN PACKAGE

Implementasi library di VHDL menggunakan “PACKAGE”. Package digunakan agar kode VHDL dapat diakses oleh semua kode VHDL lain baik dalam satu project kerja atau berbeda. Pada umumnya digunakan untuk mendefinisikan sebuah tipe data sendiri (*user-defined type*) atau konstanta. Pada praktikum ini tidak sampai membutuhkan untuk

membuat sebuah package sendiri. Praktikan lebih disarankan agar dapat memahami struktur file pada saat kompilasi, sehingga dapat menggunakan package secara tepat dan efisien. Berikut adalah package yang akan sering digunakan selama praktikum. **Catatan :** package-package ini merupakan sub-directory dari folder “ieee” baik di program Quartus maupun Modelsim. Keyword “all” memiliki arti bahwa semua type, constant, function, procedure, atau lainnya yang terdefinisi di dalam package tersebut dapat digunakan.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

ENTITY

Kata “ENTITY” di sini mengacu pada sebuah keyword yang disediakan oleh VHDL. Pada bagian ini perlu dideklarasikan nama port input output. Pada bagian “ENTITY” dapat juga tidak dideklarasikan port sama sekali, hal ini biasa dilakukan untuk file testbench. Mode yang didukung oleh VHDL tampak pada table di bawah.

| Mode | Keterangan |
|--------|---|
| IN | Signal input. |
| OUT | Signal output. Signal ini tidak dapat digunakan di dalam entity. |
| INOUT | Signal input sekaligus output. |
| BUFFER | Signal output. Signal ini dapat digunakan di dalam entity. |

ARCHITECTURE

Pada bagian ini menggambarkan secara rinci cara kerja dari sebuah entity. Bagian “ARCHITECTURE” terdiri menjadi dua bagian, yaitu bagian declarative dan body. Pemisah antara dua bagian tersebut adalah keyword “BEGIN” (lihat struktur kode VHDL).

INSTANSIASI COMPONENT

“COMPONENT” adalah istilah dari VHDL untuk menyatakan entity lain yang akan di-instantiasi. “COMPONENT” tidak harus dideklarasikan pada bagian declarative architecture. Sebagai alternatifnya, deklarasi “COMPONENT” dapat dilakukan di bagian “PACKAGE”, lalu package ini digunakan sebagai library entity tersebut.

Berikut adalah contoh deklarasi instantiasi.

```

COMPONENT nama_entity_instantiated IS
GENERIC (
nama_signal1_instantiated : nama_type := nilai_default;
nama_signal2_instantiated : nama_type := nilai_default;
...
nama_signal3_instantiated : nama_type := nilai_default
);
PORT (
nama_port1_instantiated : [mode] nama_type;
nama_port2_instantiated : [mode] nama_type;
...
nama_port3_instantiated : [mode] nama_type
);
END COMPONENT;

```

Sedangkan berikut adalah cara instantiasi di bagian body “*ARCHITECTURE*”.

```

label_instantiasi : nama_entity_instantiated
    GENERIC MAP(
        nama_signal1_instantiated => nilai;
        nama_signal2_instantiated => nilai;
        ...
        nama_signal3_instantiated => nilai
    )
    PORT MAP(
        nama_port1_instantiated => nama_port1_entity;
        nama_port2_instantiated => nama_port2_entity;
        ...
        nama_port3_instantiated => nama_port3_entity
    );

```

Atau dengan cara di bawah. **Catatan : urutan sangat diperhatikan.**

```

label_instantiasi : nama_entity_instantiated
    GENERIC MAP(
        nama_signal1_instantiated => nilai;
        nama_signal2_instantiated => nilai;
        ...
        nama_signal3_instantiated => nilai
    )
    PORT MAP(
        nama_port1_entity;
        nama_port2_entity;
        ...
        nama_port3_entity
    );

```

GENERATE STATEMENT

Terdapat dua jenis “*GENERATE*” statement, yaitu looping “*GENERATE*” dan conditional “*GENERATE*”. “*GENERATE*” statement sangat menguntungkan jika digunakan untuk mendefinisak sebuah stament yang berulang. Berikut adalah format sintaksnya.

```
nama_label:
FOR nama_variable IN {range} GENERATE
    statement;
ENDGENERATE nama_label ;
```

```
nama_label:
IF ekspresi GENERATE
    statement;
ENDGENERATE nama_label ;
```

KONKUREN ASSIGNMENT STATEMENT

Pada dasarnya ekspresi-ekspresi dalam VHDL dijalankan secara konkuren. Biasanya bagian kombinasional diimplementasikan secara konkuren. Terdapat empat jenis konkuren assignment yang dapat dilakukan, yaitu simple signal, selected signal, conditional signal, dan "GENERATE".

- Simple Signal Assignment

VHDL memiliki dua jenis operator assignment, yaitu "<=" dan ":=". **Catatan : operator "<=" dapat digunakan baik di konkuren assignment atau sekuensial. Sedangkan operator ":" hanya dapat digunakan di sekuensial assignment. Pada bagian sekuensial assignment operator "<=" disebut operator non-blocking, sedangkan operator ":" disebut operator blocking.**

Contoh :

```
f <= (x1 AND x2) OR x3;
s <= res(3 DOWNTO 0);
temp <= (OTHERS => '0');
```

- Selected Signal Assignment

Format :

```
nama_label :
WITH ekspresi SELECT
    nama_signal <= ekspresi WHEN nilai_constant,
    ekspresi WHENOTHERS;
```

- Conditional Signal Assignment

Format :

```
nama_label :
nama_signal <= ekspresi WHEN ekspresi_logic ELSE
    ekspresi WHEN ekspresi_logic ELSE
    ekspresi ;
```

SEKUENSIAL ASSIGNMENT STATEMENT

Tidak semua jenis rangkaian dapat diekspresikan dengan konkuren statement. Dalam sekuensial assignment statement, urutan penulisan berpengaruh terhadap kerja entity tersebut. Salah satu cara implementasi sekuensial statement di VHDL menggunakan "PROCESS". Bagian "PROCESS" dengan bagian lain di body "ARCHITECTURE" dijalankan secara konkuren. Pada bagian sensitivity_list dideklarasikan signal-signal referensi yang mana jika nilai signal tersebut berubah maka statement yang didefinisikan di dalam "PROCESS" tersebut dijalankan. **Catatan : "WAIT" dan "LOOP" pada umumnya digunakan untuk modul testbench.**

```
nama_label : PROCESS (sensitivity_list)
              [VARIABLE declarations];
    BEGIN
        [Simple assignment statements]
        [VARIABLE assignment statements]
        [Conditional assignment statements]
        [WAIT assignment statements]
        [LOOP assignment statements]
    END PROCESS nama_label;
```

- Conditional assignment statement

```
nama_label :
    IF ekspresi THEN
        statement;
    ELS IF ekspresi THEN
        statement;
    ELSE THEN
        statement;
    END IF nama_label;
```

```
nama_label :
    CASE nama_signal IS
        when ekspresi =>
            statement;
        when ekspresi =>
            statement;
        ...
        when others =>
            statement;
    ENDCASE nama_label ;
```

- Looping assingment statement

Statemen ini akan banyak digunakan di modul testbench.

```
nama_label :
    FOR nama_variable IN {range} LOOP
        statement;
    END LOOP nama_label ;
```

```
nama_label :
    WHILE ekspresi LOOP
        statement;
    END LOOP nama_label ;
```

- "WAIT" statement

Statement ini akan banyak digunakan di modul testbench. `time_unit` yang didukung oleh VHDL diantaranya adalah `"PS"` (picosecond), `"NS"` (nanosecond), dst.

```
WAIT FOR time_value time_unit; statement;
```

OBJEK DAN TIPE DATA

Segala bentuk informasi di VHDL dinyatakan dalam sebuah data objek. VHDL menyediakan 3 jenis objek data, yaitu `"SIGNAL"`, `"CONSTANT"`, `"VARIABLE"`.

- `"SIGNAL"`

Objek `"SIGNAL"` merupakan representasi wire dalam sebuah rangkaian. Objek ini dapat dideklarasikan di bagian deklarasi entity, architecture, dan package.

- `"CONSTANT"`

Objek ini memiliki nilai yang tidak dapat diubah dan dapat direpresentasikan di bagian deklarasi architecture dan package.

- `"VARIABLE"`

Pada umumnya objek ini digunakan sebagai objek data sementara untuk menyimpan nilai yang akan digunakan dalam jangka waktu tertentu atau sebagai parameter looping atau kondisional statement. Objek ini hanya dapat dideklarasikan di bagian deklarasi `"PROCESS"`.

Setiap objek memiliki tipe data. Berikut adalah tipe data yang akan banyak digunakan :

- `"BIT"`

Nama package : *predefined*

Penjelasan :

Objek yang memiliki tipe data ini hanya dapat bernilai '0' atau '1' dan memiliki lebar bitnya adalah 1.

Contoh deklarasi objek :

```
SIGNAL A : BIT;
```

- `"BIT_VECTOR"`

Nama package : *predefined*

Penjelasan :

Merupakan array dari tipe data `"BIT"`. Tipe data ini pada umumnya digunakan untuk menyatakan multibit objek data. Cara deklarasi jumlah bit dan tipe endiannya digunakan sintaks `"TO"` (Little Endian) atau `"DOWNTO"` (Big Endian).

Contoh deklarasi objek :

```
SIGNAL LE : BIT_VECTOR(0 TO 7); -- Little Endian
SIGNAL BE : BIT_VECTOR(7 DOWNTO 0); -- Big Endian
```

- **"STD_LOGIC"**

Nama package : ieee.std_logic_1164

Penjelasan :

Perbedaan utama dari tipe data **"BIT"** adalah representasi nilai yang didukung, diantaranya adalah : '0', '1', 'Z' (high impedance), '-' (don't care). Lebar bit tipe data ini adalah 1.

Contoh deklarasi objek :

```
SIGNAL A: STD_LOGIC;
```

- **"STD_LOGIC_VECTOR"**

Nama package : ieee.std_logic_1164

Penjelasan :

Merupakan array dari tipe data **"STD_LOGIC"**. Sama halnya dengan tipe data **"BIT_VECTOR"** sintaks **"TO"** dan **"DOWNTO"** berlaku untuk deklarasi lebar bit dan tipe endian. **Catatan : jika ingin menggunakan objek data dengan tipe "STD_LOGIC" atau "STD_LOGIC_VECTOR" sebagai operan dalam ekspresi arithmetic perlu ditambahkan package "ieee.std_logic_signed" atau "ieee.std_logic_unsigned". Format signed data menggunakan 2's complement.**

Contoh deklarasi objek :

```
SIGNAL A: STD_LOGIC_VECTOR(7 DOWNTO 0);
```

- **"SIGNED" dan "UNSIGNED"**

Nama package : ieee.std_logic_arith atau ieee.numeric_std

Penjelasan :

Mirip dengan tipe data **"STD_LOGIC_VECTOR"** yang ditambahkan dengan package **"ieee.std_logic_signed"** atau **"ieee.std_logic_unsigned"**.

- **Enumeration**

Nama package : *user-specified*

Penjelasan :

VHDL menyediakan fasilitas agar user dapat mendefinisikan sendiri tipe data sesuai kebutuhan. Pada umumnya digunakan untuk mendefinisikan nama state FSM. Pada implementasi FSM, user difasilitasi juga dengan metode encoding untuk keperluan optimasi menggunakan sintaks **"ATTRIBUTE"**.

Contoh kode :

```
TYPE State_Type IS (S_A, S_B, S_C);
SIGNAL s_present : State_Type;
ATTRIBUTE ENUM_ENCODING : STRING;
ATTRIBUTE ENUM_ENCODING OF State_Type : TYPE IS "00 01 11";
```

- **"INTEGER"**

Nama package : *predefined*

Penjelasan :

Pada umumnya tipe data ini digunakan untuk keperluan operasi arithmetic. Tetapi berbeda dengan tipe data *"STD_LOGIC_VECTOR"*, secara default tipe data *"INTEGER"* memiliki lebar data sepanjang 32 bit. Untuk mendeklarasikan lebar datanya digunakan sintaks *"RANGE"*.

Contoh deklarasi objek :

```
SIGNAL A : INTEGERRANGE 0 TO 255;-- 8 bits
```

- **"BOOLEAN"**

Nama package : *predefined*

Penjelasan :

Nilai legal dari tipe data ini hanyalah *"TRUE"* (ekuivalen dengan '1') atau *"FALSE"* (ekuivalen dengan '0').

Contoh deklarasi objek :

```
SIGNAL A : BOOLEAN;
```

NOTASI ANGKA

VHDL memiliki 2 cara dalam menotasikan sebuah nilai.

1. Cara 1

Format : [-][radix][#][nilai][#]

| Elemen | Keterangan |
|--------|---|
| - | Menyatakan bilangan negatif |
| radix | Bernilai 2, 8, 10, atau 16 |
| nilai | Angka numerik sesuai radix |
| # | Tanda # setelah radix harus selalu ditulis, sedangkan # di akhir tidak harus selalu ditulis |

Contoh :

- $16\#F011\# = (F011)_{16}$
- $2\#1011 = (1011)_2$
- Dst.

2. Cara 2 (lebih sering digunakan)

Format : [basis][“][nilai][“]

| Elemen | Keterangan |
|--------|--------------------------------|
| Basis | B : binary, O : octal, X : hex |
| “ | Harus selalu ditulis |
| Nilai | Angka numerik sesuai radix |

Contoh :

- B"1011" = (1011)₂
- X"F101" = (F101)₁₆
- Dst.

Catatan : perhatikan baik-baik tipe data objek yang akan di-assign sebuah nilai!
Notasi angka di atas tidak berlaku untuk semua tipe data.

OPERATOR

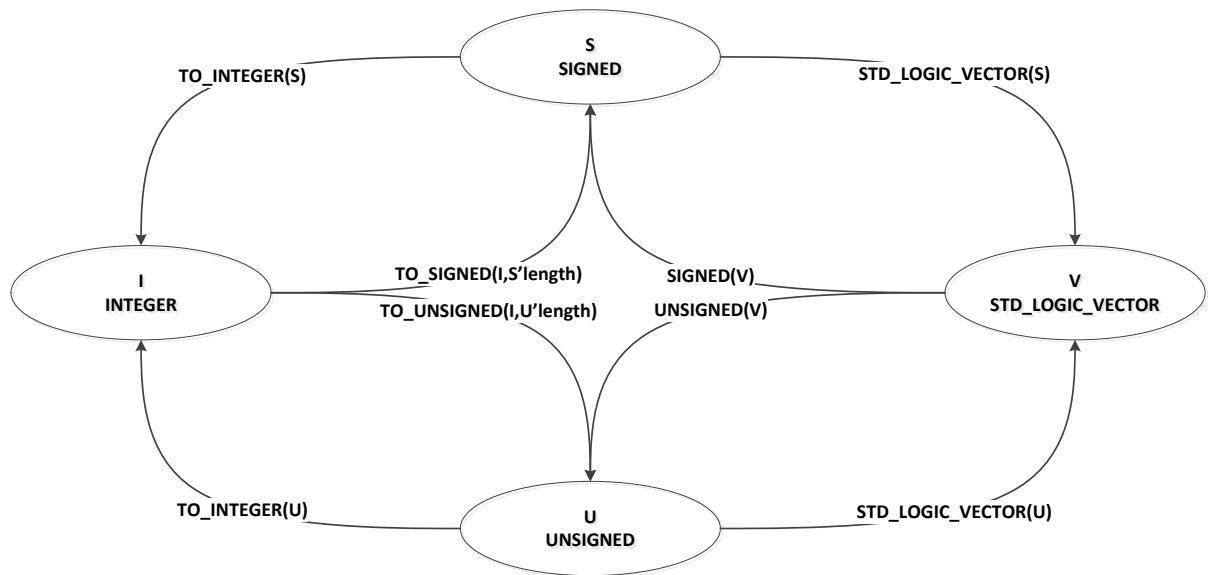
VHDL memiliki 3 jenis operator, yaitu boolean, arithmetic, dan relational operator. Berikut adalah jenis operator yang akan banyak digunakan selama praktikum.

| | Operator Class | Operator |
|--------------------|----------------|-------------------------------|
| Highest precedence | Misc | **, ABS, NOT |
| | Multiplying | *,/,MOD,REM |
| | Sign | +, - |
| | Adding | +, -, & |
| | Relational | =, /, <, <=, >, >= |
| Lowest precedence | Logical | AND, OR, NAND, NOR, XOR, XNOR |

Catatan : dalam satu ekspresi dengan operator class yang sama level precedencenya sama, oleh karena itu digunakan tanda kurung “(ekspresi)”.
Operator “&” merupakan operator concatenation.

TYPE CONVERSION

Bagian ini sangat perlu diperhatikan dalam membuat kode VHDL. Objek yang memiliki tipe data berbeda tidak dapat berada dalam satu statement yang sama, kecuali terdapat hubungan pewarisan tipe data. Gambar di bawah menyederhanakan fungsi-fungsi konversi yang akan banyak digunakan.



CATATAN : konversi dari tipe data INTEGER ke STD_LOGIC_VECTOR dapat menggunakan fungsi CONV_STD_LOGIC_VECTOR(I,V'length). Untuk sebaliknya dapat dilakukan dengan cara konversi ke SIGNED atau UNSIGNED terlebih dahulu.