ITS
Institut
Teknologi
Sepuluh Nopember

KERJA PRAKTIK - EC184601

Laboratorium Mikroelektronika dan Sistem Tertanam
Departemen Teknik Elektro
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
(17 Desember 2021 s/d 17 Januari 2022)

Synthesizable Single-Cycle FPGA Soft Processor Core

Aaron Elson Phangestu **NRP 0721 18 4000 0041**

Dosen Pembimbing
Ahmad Zaini, S.T., M.T.

DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2022

KERJA PRAKTIK - EC184601

Laboratorium Mikroelektronika dan Sistem Tertanam
Departemen Teknik Elektro
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
(17 Desember 2021 s/d 17 Januari 2022)

Synthesizable Single-Cycle FPGA Soft Processor Core

Aaron Elson Phangestu **NRP 0721 18 4000 0041**

Dosen Pembimbing
Ahmad Zaini, S.T., M.T.

DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2022

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

Synthesizable Single-Cycle FPGA Soft Processor Core

Laporan Kerja Praktik ini disusun untuk memenuhi persyaratan akademik Departemen Teknik Komputer – Fakultas Teknologi Elektro dan Informatika Cerdas – Institut Teknologi Sepuluh Nopember

Tempat Pengesahan di: Surabaya
Tanggal: 5 Juni 2022

Menyetujui,
Dosen Pembimbing,

Ahmad Zaini, S.T., M.T.
NIP. 19750419 200212 1 003

Mengetahui,
Kepala Departemen Teknik Komputer FTEIC - ITS,

Dr. Supeno Mardi Susiki Nugroho ST., M.T.
NIP. 19700313 199512 1 001

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

Synthesizable Single-Cycle FPGA Soft Processor Core

Laporan Kerja Praktik ini disusun untuk memenuhi persyaratan akademik Departemen Teknik Komputer – Fakultas Teknologi Elektro dan Informatika Cerdas – Institut Teknologi Sepuluh Nopember

Tempat Pengesahan di: Surabaya
Tanggal: 5 Juni 2022

Mengetahui,
Pembimbing Laboratorium

Dr. Ir. Totok Mujiono, M.Ikom

Mengetahui,
Kepala Laboratorium Mikroelektronika dan Sistem Tertanam
Departemen Teknik Elektro – FTEIC – ITS

Dr. Ir. Hendra Kusuma, M.Eng.Sc.

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur kami ucapkan kepada Tuhan Yang Maha Esa karena atas petunjuk dan rahmat-Nya, penulis telah dapat menyeleasikan Kerja Praktik di Departemen Teknik Elektro, FTEIC, ITS yang dilaksanakan tanggal 17 Desember 2021 sampai dengan 17 Januari 2022.

Dalam penyelesaian Laporan Kerja Praktik ini, kami mengucapkan terima kasih kepada semua pihak yang telah membantu terselesaikannya laporan ini hingga akhir:

1. Bapak Dr. Ir. Totok Mujiono, M.Ikom. Selaku pembimbing Laboratorium Mikroelektronika dan Sistem Tertanam Departemen Teknik Elektro FTEIC-ITS.
2. Ibu Dr. Diah Puspito Wulandari, S.T., M.Sc. selaku Koordinator Kerja Praktik Departemen Teknik Komputer FTEIC-ITS.
3. Pak Ahmad Zaini, S.T, M.T. selaku pembimbing pelaksanaan kerja praktik dan penulisan laporan Depertemen Teknik Komputer FTEIC-ITS.
4. Semua pihak terkait yang tidak dapat kami sebutkan satu per satu.

Penulis menyampaikan permohonan maaf jika selama pelaksanaan kerja praktik ini terdapat hal yang kurang berkenan dan apabila ada salah kata dalam penulisan laporan ini.

Surabaya, Juni 2022

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

1.1 Latar Belakang

Selaras dengan perkembangan teknologi pada era revolusi industri 4.0 ini, peran Perguruan Tinggi sangat penting sebagai wadah untuk menghasilkan Sumber Daya Manusia yang memiliki kepribadian yang mandiri, berkualitas serta memiliki kemampuan intelektual yang baik, merasa terpanggil untuk meningkatkan mutu setiap lulusannya, baik dari segi kuantitasnya maupun segi kualitasnya. Institut Teknologi Sepuluh Nopember Surabaya sebagai salah satu Institut Teknologi di Indonesia berupaya mengembangkan sumber daya manusia serta ilmu pengetahuan dan teknologi (IPTEK), guna menunjang perkembangan pada revolusi industri 4.0

Wawasan mahasiswa mengenai dunia kerja yang berkaitan dengan bidang industri sangat diperlukan, berhubung dengan kondisi Indonesia yang merupakan negara berkembang dan memerlukan pembangunan yang sempurna dalam berbagai bidang, yang dimana teknologi adalah suatu hal yang berperan penting dalam memajukan bidang industri. Oleh karena itu dapat diharapkan bahwa pada masa yang akan datang mahasiswa sebagai calon lulusan dari Perguruan Tinggi akan lebih mengenal perkembangan pada Industri. Pada umumnya, dunia kerja dan dunia pendidikan akan memiliki perbedaan yang sangat signifikan. Hal tersebut menyebabkan lulusan dari dunia pendidikan tersebut yang ingin langsung terjun ke dunia kerja belum memiliki kepastian akan kesiapan yang matang untuk menghadapi segala sesuatu yang akan terjadi dalam dunia kerja.

Oleh karena itu, dalam rangka mengurangi kesenjangan yang sudah disebutkan sebelumnya, maka dibentuklah kegiatan praktek di lapangan yang merupakan hal yang sangat membantu bagi mahasiswa dalam merasakan dunia kerja. Salah satu cara yang efektif adalah dengan melakukan proses magang pada sebuah perusaha-

an atau yang lebih dikenal dengan Kerja Praktek. ITS diharapkan mampu menghasilkan tenaga tenaga profesional yang siap dan sesuai dengan bidang keahliannya Untuk mewujudkan itu maka ITS, pada khususnya Departemen Teknik Komputer, mempunyai program kegiatan Kerja Praktek bagi Mahasiswa, sehingga mahasiswa lulusan dari departemen ini akan mendapatkan pengalaman dan wawasan kerja yang menambah kesiapan dalam menghadapi dunia kerja yang sesungguhnya.

1.2 Rumusan Permasalahan

Soft Microprocessor merupakan desain prosesor yang dideskripsikan menggunakan suatu Logic Description Language. Umumnya, digunakan Hardware Description Language Verilog atau VHDL mendeskripsikan desain suatu soft processor core. Desain ini dapat disintesis ke dalam suatu programmable logic device (PLD). Terdapat beberapa perangkat semikonduktor yang dapat di-program untuk sintesis soft processor. Misalnya, ASIC, FPGA, dan CPLD.

Perangkat terprogram yang paling banyak digunakan untuk keperluan prototyping yakni Field-Programable Gate Array (FPGA). FPGA merupakan integrated circuit yang dapat diatur ulang yang terdiri dari blok logika kombinasional dan flip-flop. FPGA dapat dikonfigurasi menggunakan Hardware Description Language, mirip dengan yang digunakan untuk Application Specific Integrated Circuit (ASIC). Bedanya, ASIC tidak dapat dikonfigurasi ulang jika pengguna ingin mengubah desain arsitektur.

Dalam merancang suatu arsitektur prosesor, digunakan Instruction Set Architecture (ISA) sebagai abstraksi. ISA merupakan bagian abstrak dari model komputer yang mendefinisikan bagaimana CPU dikontrol oleh software. ISA berfungsi sebagai antarmuka antara hardware dan software. Terdapat dua approach desain ISA: Reduced Instruction Set Computer (RISC) dan Complex Instruction Set Computer (CISC). Keduanya memiliki keunggulan dan kelemahannya masing-masing.

RISC-V merupakan salah satu Instruction Set Architecture yang sedang berkembang pesat. RISC-V adalah Instruction Set Architecture open-standard berdasarkan prinsip RISC. Karena tersedia

dengan lisensi open source, RISC-V dapat digunakan dan diubah sesuai keinginan pengguna, tanpa harus membeli lisensi seperti kebanyakan Instruction Set Architecture. Prosesor dengan RISC-V ISA dapat menjalankan program tingkat tinggi yang dikompilasi menggunakan RISC-V toolchain. Sebagai pendukung pembelajaran teoritis pada mata kuliah yang sudah dijalankan, dibutuhkan suatu program Kerja Praktek. Arsitektur dan Organisasi komputer merupakan wawasan dasar yang perlu dieksplorasi mahasiswa Teknik Komputer. Oleh karena itu, dipilih topik Kerja Praktek mengenai implementasi soft microprocessor yang dapat disintesis ke dalam hardware aktual. Selain itu, Laboratorium Mikroelektronika dan Sistem Tertanam dianggap menjadi tempat yang cocok untuk mengeksplorasi topik ini karena topik ini berhubungan dengan fokus utama laboratorium tersebut.

Kegiatan Kerja Praktek yang direncanakan akan dijalankan secara remote. Hal ini dikarenakan situasi pandemi yang sedang berlangsung sehingga work-from-home dianggap menjadi solusi paling efektif. Tidak akan terjadi perbedaan signifikan antara hasil Kerja Praktek offline maupun online, karena semua alat dan materi yang dibutuhkan dalam Kerja Praktek sudah tersedia, dan bimbingan dapat dilakukan secara online melalui online meeting.

1.3 Tujuan

Tujuan utama dari topik Kerja Praktek yang diangkat yakni untuk merealisasikan suatu soft microprocessor core yang dapat disintesis ke dalam sebuah Field Programable Gate Array. Processor core tersebut berbasiskan RISC-V Instruction Set Architecture dan memiliki arsitektur Single-cycle sederhana. Akan dilakukan analisa terhadap resource usage dan verifikasi terhadap eksekusi instruksi. Selain itu, FPGA direncanakan dapat berinteraksi dengan input melalui suatu interface. Interaksi tersebut dapat berupa kemampuan mengoutputkan nilai melalui serial.

1.4 Waktu dan Tempat Pelaksanaan

Kerja praktek akan dilaksanakan pada bulan Desember 2021 hingga Januari 2022 di Departemen Teknik Elektro, Fakultas Tek-

nologiElektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya, secara *online*.

1.5 Metodologi Kerja Praktik

Metode yang digunakan dalam pelaksanaan kerja praktek iniadalah sebagai berikut:

1. Metode *literature review* dengan mempelajari literatur-literatur mengenai perangkat yang akan dirancang.
2. Metode pengumpulan data dengan mengamati hasil yang dicapai dan membandingkannya dengan prediksi saat perancangan alat.
3. Metode diskusi yang dilakukan dengan pembahasan hasil dengan pembimbing di lapangan.

Laporan kerja praktik akan terbagi menjadi Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. yaitu:

1. **Bab I Pendahuluan**
   Bab pertama membahas tentang latar belakang, rumusan masalah, tujuan, waktu dan tempat pelaksanaan, metodologi, dan sistematika penulisan.
2. **Bab II Profil Perusahaan**
   Bab kedua berisi profil singkat tentang Departemen Teknik Elektro ITS.
3. **Bab III Tinjauan Pustaka**
   Pada bab ini dibahas mengenai teori penunjang yang berkaitan dengan perancangan *soft processor* berbasis RISC-V.
4. **Bab IV Desain dan Implementasi**
   Bab keempat membahas mengenai proses dan hasil dari perancaan *soft processor single-cycle* berbasis RISC-V.
5. **Bab V Pengujian dan Evaluasi**
   Bab kelima membahas mengenai hasil dan cara menguji dan memverifikasi desain dari *soft processor single-cycle* berbasis RISC-V.
6. **Bab VI Kesimpulan dan Saran**
   Bab terakhir membahas tentang kesimpulan dan saran mengenai kerja praktek yang sudah dijalankan.

BAB II
PROFIL PERUSAHAAN

2.1 Sejarah Departemen Teknik Elektro FTEIC - ITS

Depatemen Teknik Elektro secara struktural berada dibawah Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember (ITS). Struktur dan nama Departemen Teknik Elektro ini telah berjalan sejak januari tahun 2017 berdasarkan Peraturan Pemerintah No. 54 tahun 2015 dan Peraturan Rektor No. 10 tahun 2016.

Departemen Teknik Elektro berdiri pada tahun 1960, bersamaan dengan peresmian Perguruan Tinggi Negeri: Institut Teknologi Sepuluh Nopember di Surabaya, oleh Presiden pertama RI Ir. Soekarno. ITS yang semula memiliki 2 (dua) departemen yaitu Teknik Sipil dan Teknik Mesin, berkembang menjadi lima yaitu: Teknik Elektro, Teknik Sipil, Teknik Mesin, Teknik Perkapalan, dan Teknik Kimia. Departmen-departemen tersebut kemudian berubah menjadi fakultas.

Secara singkat sejarah perjalanan Departemen Teknik Elektro dapat diringkas sebagai berikut:

2.1.1 Periode 1960-1961

Bernama Departemen Teknik Elektro, dipimpin oleh Ketua Departemen yang merupakan Presidium Institut, terdiri dari:

1. Presiden ITS : Dr. Angka Nitisastro

2. Wakil Presiden ITS : Prof. Ir. R. Soemardjo

3. Sekretaris ITS : R. Soemani

2.1.2 Periode 1963-1965

Departemen Teknik Elektro berubah nama menjadi Fakultas Teknik Elektro, dengan pimpinan:

1. Dekan : Ir. Moerhadi
2. Pembantu Dekan I : Ir. Arief Owen
3. Pembantu Dekan II : Ir. Ong Sing Han
4. Pembantu Dekan III : Ir. Joe Boen Kiat

2.1.3 Periode 1965 – 1983

Fakultas Teknik Elektro terus berkembang dengan berbagai dinamika seperti:

- Mula-mula memiliki 2 program studi: arus kuat dan arus lemah
- Berkembang menjadi 11 program studi
- Kemudian berubah lagi menjadi 5 program studi

2.1.4 Tahun 1983

Merupakan periode transisi akibat terbitnya Keputusan Presiden No. 58 tahun 1982 tentang perubahan struktur fakultas di ITS. Fakultas Teknik Elektro berubah nama menjadi Jurusan Teknik Elektro dan berada dibawah Fakultas Teknologi Industri bersama beberapa jurusan lain.

2.1.5 Periode 1984 – 1999

Jurusan Teknik Elektro terus berbenah dan berkembang dengan komposisi 5 bidang studi sebagai berikut:

- Teknik Sistem Tenaga
- Telekomunikasi
- Elektronika

- Teknik Sistem Pengaturan
- Teknik Komputer

Pada tahun 1986, Bidang Studi Teknik Komputer berubah menjadi program studi Teknik Komputer. Program Studi Teknik Komputer tersebut merupakan cikal bakal berdirinya Fakultas Teknologi Informasi, yang kemudian diresmikan berdasarkan SK Rektor tanggal 14 Juni 2001. Sementara itu di Jurusan Teknik Elektro masih terdapat bidang studi Teknik Komputer yang menekankan kompetensi bidang komputer terkait perangkat keras.

2.1.6 Periode 1999 – 2011

Jurusan Teknik Elektro terus berkembang mengikuti perkembangan keilmuan dengan perubahan komposisi di bidang teknik komputer:

- Teknik Sistem Tenaga
- Telekomunikasi Multimedia
- Elektronika
- Teknik Sistem Pengaturan
- Teknik Komputer dan telematika

2.1.7 Periode 2012 – 2016

Jurusan Teknik Elektro terus berbenah dan berkembang dengan komposisi yang berubah, terdiri dari :

- Teknik Sistem Tenaga
- Telekomunikasi Multimedia
- Elektronika
- Teknik Sistem Pengaturan
- Program Studi Teknik Multimedia dan Jaringan
- Program Studi Teknik Biomedik

Pada tahun 2012, Bidang Studi Teknik Komputer dan Telematika berubah menjadi program studi Teknik Multimedia dan Jaringan, berdasarkan Keputusan Menteri Pendidikan dan Kebudayaan Republik Indonesia No. 382/E/O/2012.

Pada tahun 2013, berdasarkan penugasan Direktorat Jenderal Pendidikan Tinggi No. 221/E/DT/2013, tanggal 20 Maret 2013 Jurusan Teknik Elektro mendapatkan amanah untuk mempersiapkan Program Studi Teknik Biomedik. Program studi ini mulai beroperasi pada tahun 2015 berdasarkan Keputusan Menteri Riset, Teknologi, dan Pendidikan Tinggi No. 102/M/Kep/2015, tangal 30 Maret 2015. Program Studi Teknik Biomedik ITS telah menerima mahasiswa baru untuk tahun ajaran 2015/2016 melalui seleksi masuk mahasiswa baru ITS program kemitraan dan mandiri (PKM). Program studi ini beranggotakan 10 orang dosen yang komposisinya terdiri dari 1 (satu) orang guru besar, 2 (dua) orang Doktor, dan 7 (tujuh) orang magister.

Pada januari tahun 2017 berdasarkan berdasarkan Peraturan Pemerintah No. 54 tahun 2015 dan Peraturan Rektor No. 10 tahun 2016, program studi D3 Teknik Elektro menjadi Departemen Teknik Elektro Otomasi di bawah Fakultas Vokasi.

2.1.8 Periode 2017 – sekarang

Departemen Teknik Elektro terus berkembang mengikuti perkembangan keilmuan dengan perubahan komposisi bidang keahlian sebagai berikut :

- Teknik Sistem Tenaga
- Telekomunikasi Multimedia
- Elektronika
- Teknik Sistem Pengaturan

Selama beberapa dasawarsa, sejak Teknik Elektro berbentuk fakultas hingga berubah menjadi Departemen Teknik Elektro (DTE) di bawah payung Fakultas Teknologi Elektro (FTE), program utama yang dijalankan adalah

- Program Sarjana (S1) yang memiliki 4 bidang keahlian

- Program Magister Teknik (S2) mulai didirikan pada tahun 1993

- Program Doktoral (S3) dimulai pada tahun 2005

Disisi jumlah SKS, jenjang Sarjana Teknik Elektro yang semula ditempuh dalam 200 SKS, berubah menjadi 160 SKS, mengecil menjadi 152 SKS dan sekarang menjadi 144 SKS. Hal ini memaksa adaptasi kurikulum yang cukup ketat diantara 4 bidang keahlian yang ada di Departemen Teknik Elektro.

2.2 Laboratorium Mikroelektronika dan Sistem Tertanam

2.2.1 Deskripsi Laboratorium

Laboratorium Mikroelektronika dan Sistem Tertanam merupakan salah satu laboratorium di bawah naungan Departemen Elektro FTEIC ITS. Laboratorium Mikroelektronika dan Sistem Tertanam memberi sarana kepada para mahasiswa dalam mempelajari dasar sistem elektronika dan hukum-hukum rangkaian listrik dan aplikasinya pada sistem elektronika umum khususnya pada sistem elektronika industri. Laboratorium ini melayani praktikum mahasiswa, pengerjaan tugas akhir dan tesis mahasiswa, penelitian mahasiswa bersama dosen, perkuliahan, dan kerja praktek mahasiswa.

2.2.2 Topik Penelitian

Penelitian yang terkait dengan laboratorium ini adalah

- Perancangan dan pembuatan modul/*board* elektronika Analog dan Digital

- Desain dan *Implementasi Sensor Conditioning Circuits*

- Perancangan dan pembuatan rangkaian akuisisi data

- Pengolahan sinyal analog

- Pengembangan dan perancangan alat cetak *Braille*

2.2.3 Layanan

Laboratorium ini melayani beberapa kegiatan antara lain

- Praktikum Rangkaian Listrik
- Praktikum Elektronika Dasar
- Kerja Praktek Mahasiswa
- Pengerjaan Tugas Akhir Mahasiswa yang sesuai dengan topik-topik penelitian Laboratorium
- *Internship*/Magang dalam mempelajari Elektronika Dasar
- Pelatihan Ketrampilan Elektronika Dasar untuk masyarakat umum/industri

2.2.4 Kepala Laboratorium

Dr. Ir. Hendra Kusuma, M.Eng.Sc.

2.2.5 Anggota Laboratorium

- Dr. Ir. Totok Mujiono, M.I.Kom.
- Astria Nur Irfansyah, S.T., M.Eng., Ph.D.
- Ir. Harris Pirngadi, M.T.

2.3 Visi dan Misi

2.3.1 Visi Departemen Teknik Elektro FTEIC - ITS

Visi Departemen Teknik Elektro adalah menjadi lembaga pendidikan tinggi dan penelitian yang unggul berkelas dunia di tahun 2020 serta menjadi ujung tombak dalam proses pengembangan dan pengalihan Ilmu Pengetahuan dan Teknologi (IPTEK) dalam bidang Teknik Elektro. Untuk menjadi berkelas dunia, maka Departemen Teknik Elektro ditargetkan mempunyai 100 kegitan penelitian, 100 prestasi mahasiswa dan 200 publikasi pertahun.

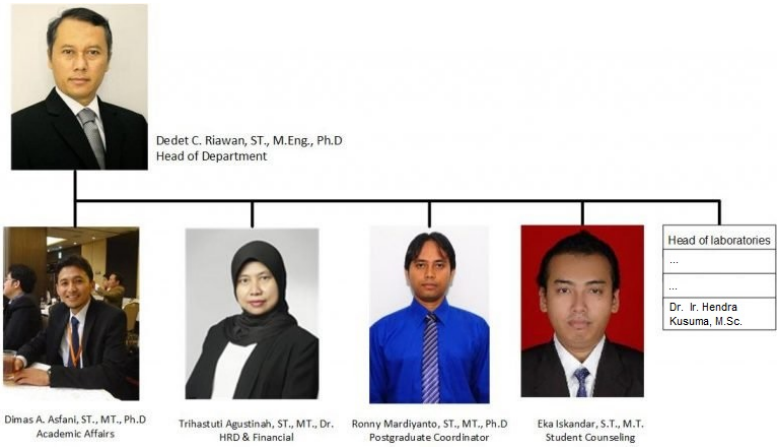
2.3.2 Misi Departemen Teknik Elektro FTEIC - ITS

Misi Departemen Teknik Elektro adalah

- Untuk melakukan program pendidikan yang didukung oleh sumber daya yang berkualitas.
- Untuk melaksanakan pengembangan dan penerapan teknik listrik yang berkontribusi terhadap perkembangan ilmu pengetahuan dan teknologi, khususnya di bidang elektronik, energi, telekomunikasi, sistem kontrol, komputasi dan teknologi informasi. Untuk meningkatkan kompetensi staf akademik dan dukungan untuk menjadi profesional dalam tugas-tugas mereka.
- Untuk membangun suasana akademik yang kondusif dalam rangka mengembangkan kreativitas dan inovasi dari staf dan siswa
- Untuk memberikan proses pembelajaran yang berbasis pada kompetensi listrik; untuk melakukan inovatif, berkualitas tinggi, dan penelitian yang berlaku yang relevan dengan kebutuhan nasional dan global.
- Memainkan peran penting dalam meningkatkan kualitas hidup dan keunggulan kompetitif bangsa, dan mendorong kerja sama dengan lembaga domestik dan internasional.

2.4 Struktur Organisasi

Gambar 2.1 merupakan struktur organisasi Departemen Teknik Elektro ITS. Dalam menjalankan kegiatan operasionalnya Departemen Teknik Elektro ITS telah membentuk struktur organisasi yang kompatibel dengan struktur organisasi ITS. Dalam pelaksanaan program Departemen Teknik Elekto ITS, Kepala Departemen dibantu oleh Sekretaris Departemen, Kepala program pascasarjana, kepala laboratorium yang bertanggung jawab atas Kepemimpinan Operasional di Laboratorium dan Kepala staf non akademik (Kasubag TU) yang membantu dalam Sumber Daya Keuangan.



Gambar 2.1: Struktur organisasi Departemen Teknik Elektro FTE-IC -ITS

BAB III
TINJAUAN PUSTAKA

3.1 RISC-V

Awalnya dirancang untuk mendukung penelitian dan pendidikan di bidang Arsitektur Komputer, arsitektur set instruksi RISC-V (ISA) sekarang ditetapkan untuk menjadi arsitektur standar bebas dan terbuka untuk aplikasi akademik dan industri [3]. Untuk keberhasilan dan adopsi RISC-V, telah dirancang untuk mendukung ruang alamat 32-bit, 64-bit dan 128-bit. ISA dipisahkan menjadi ISA integer dasar kecil, yang merupakan kumpulan instruksi minimal yang memadai untuk memberikan target yang masuk akal bagi assembler, linker, compiler, dan sistem operasi. Pondasi RISC-V menyediakan rangkaian rantai alat yang kompatibel yang mencakup setelan di atas.

Arsitektur berbasis RISC telah digunakan baik dalam aplikasi tingkat rendah dan sistem seluler pada awal abad ke-21. Daya rendah dan pasar tertanam biaya rendah didominasi oleh arsitektur ARM berbasis RISC. Sebagian besar perangkat berbasis android, Apple iPhone iPad dan sebagian besar perangkat genggam menggunakan arsitektur ARM. Garis MIPS saat ini dapat ditemukan di game seperti konsol game PlayStation Portable, Nintendo 64 dan gateway perumahan pribadi seperti seri Linksys WRT54G. SuperH (SH) adalah ISA RISC 32-bit lainnya yang dikembangkan oleh Hitachi. Karena banyak paten untuk SuperH yang kedaluwarsa, SuperH2 sedang diimplementasikan kembali sebagai perangkat keras open source dengan nama J2.

3.2 RV32I

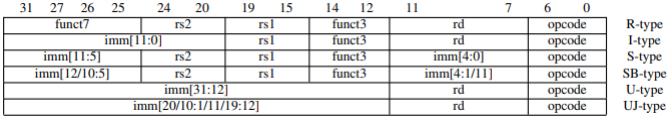
RV32I merupakan salah satu varian dari kumpulan RISC-V *Instruction Set Architecture*. RV32I dirancang agar cukup untuk membentuk target kompiler dan mendukung lingkungan sistem operasi modern. Itu juga dirancang untuk mengurangi perangkat keras

Instruction Category (Type)	RV32I Instruction	Example
Integer Register-Register Instructions (R-Type)	add, sub, sll, slt, sltu, xor, srl, sra, or, and	*add* rd,rs1,rs2 The content of register rs1 is added with that of rs2 and the result is stored in register rd.
Integer Register-Immediate Instructions (I-Type)	addi, slti, sltiu, ori, xori, andi, slli, srli, srai	*xori* rd,rs1,imm XOR operation is performed on the content of register rs1 and immediate value and the result is stored in register rd.
Integer Computational Instructions (U-Type)	lui, auipc	*lui* rd,imm The immediate value is placed in the top 20 bits of the register rd, filling in the lowest 12 bits with zeros.
Control Transfer Instructions -Unconditional Jumps (UJ, I-Type)	jal, jalr	*jal* rd,imm The immediate offset is sign -extended and added to PC to form the jump target address. The address of the instruction following the jump (PC+4) is stored into register rd.
Control Transfer Instructions -Conditional Branches (SB-Type)	beq, bne, blt, bltu, bge, bgeu	*beq* rs1,rs2,imm The 12-bit immediate value is added to the current PC to give the target address. Branch is taken if the contents of register rs1 and rs2 are equal.
Load Instructions (I-Type)	lw, lh, lb, lhu, lbu	*lw* rd,rs1,imm A 32-bit value is copied from memory to register rd. The effective byte address is obtained by adding register rs1 to the sign-extended 12-bit offset.
Store Instructions (S-Type)	sw, sh, sb	*sw* rs1,rs2,imm The 32-bit value in register rs2 is copied to memory. The effective byte address is obtained by adding register rs1 to the sign-extended 12-bit offset.

Gambar 3.1: Instruksi dalam RV32I *ISA*.

yang diperlukan untuk implementasi minimal. RV32I berisi 40 instruksi unik. Gambar 3.1 merupakan intruksi dalam RV32I *ISA* dan kategori-kategorinya. Implementasi yang lebih terintegrasi dan kompleks mungkin mencakup delapan instruksi SCALL/SBREAK/CSRR dengan satu instruksi perangkat keras SISTEM yang selalu menjebak dan mungkin dapat mengimplementasikan instruksi FENCE dan FENCE.I sebagai NOP yang akan mengurangi jumlah instruksi perangkat keras menjadi total 38. Implementasi ini mencakup pengurangan 38 instruksi perangkat keras.

RISC-V memiliki 32 register dalam file register yang diindeks dari 0 hingga 31 [2], di antaranya 31 adalah register tujuan umum (diindeks 1 hingga 31) dan register yang diindeks 0 diprogram ke konstan 0. RISC-V tidak menentukan konvensi pemanggilan perangkat lunak apa pun, jadi konvensi pemanggilan MIPS diikuti un-



Gambar 3.2: Tipe instruksi dalam RV32I *ISA*.

tuk pekerjaan ini. Penghitung program adalah register lain yang terlihat oleh pengguna di RISC-V.

Ada empat format instruksi inti dalam set instruksi bilangan bulat dasar RV32I: R, I, S dan U. Gambar 3.1 merangkum instruksi dengan contoh dan Gambar 3.2 mewakili pengkodean instruksi ini di RV32I. Setiap instruksi memiliki panjang 32-bit tetap dan harus disejajarkan dengan memori hingga batas empat-byte. Jika ini tidak diikuti, pengecualian alamat instruksi yang tidak selaras akan dihasilkan. ISA dasar RISC-V tidak memiliki instruksi untuk memeriksa luapan aritmatika bilangan bulat dan karenanya tidak ada pemeriksaan tingkat perangkat keras untuk luapan dalam operasi aritmatika bilangan bulat yang diimplementasikan di RV32I karena dapat diimplementasikan menggunakan cabang RISC-V.

3.3 FPGA (*Field Programmable Gate Array*)

FPGA (*Field Programmable Gate Array*) terdiri dari kelompok *Logic Cell* (LC), yang saling berhubungan melalui sumber daya perutean yang dapat diprogram[4]. Konfigurasi FPGA disimpan pada RAM statis tertanam di dalam chip, ini mengontrol isi LC dan multiplexer yang melakukan perutean. Arsitektur dasar ini tidak berubah secara dramatis sejak diperkenalkan pada 1980-an. FPGA awal menggunakan sel logika yang terdiri dari 4-input lookup table (LUT) dan register. Perangkat saat ini menggunakan input dalam jumlah yang lebih besar (6 input untuk Virtex-5 dan 7 input untuk Stratix III) dan memiliki sirkuit terkait lainnya. Karena area meningkat dengan jumlah input tetapi kedalaman logika berkurang, tren untuk LUT yang lebih besar mencerminkan peningkatan interkoneksi ke penundaan logika dalam teknologi sirkuit terintegrasi (IC) modern. Cluster menghubungkan beberapa LC dan berfungsi

untuk menyediakan perutean lokal di dalam cluster. Jumlah lapisan logam yang lebih banyak tersedia dalam proses IC (12 lapisan dalam kasus Virtex-5) tercermin dalam kepadatan interkoneksi yang dapat diprogram lebih baik. Sementara perangkat awal memiliki masalah dengan rutabilitas, hal ini jarang terjadi pada perangkat kontemporer.

Blok tertanam banyak digunakan dalam FPGA, berfungsi untuk meningkatkan penundaan, daya, dan area jika digunakan oleh aplikasi, tetapi membuang area dan daya jika tidak digunakan. Blok tertanam awal termasuk rantai pembawa cepat, memori, loop terkunci fase, loop terkunci tunda, pengujian pemindaian batas, dan pengganda. Baru-baru ini, pengganda telah digantikan oleh blok pemrosesan sinyal digital (DSP) yang menambahkan dukungan untuk operasi logis, pergeseran, penambahan, perkalian-tambah, perkalian kompleks, dll. Sifatnya yang umum telah ditingkatkan melalui fitur-fitur seperti dukungan panjang kata berganda dan kemampuan kaskade. Hasilnya, mereka dapat mengimplementasikan primitif seperti filter, transformasi, dan floating point dengan lebih efisien. Dukungan langsung untuk sebagian besar standar memori eksternal berkecepatan tinggi modern, jaringan, bus komunikasi, dan IO juga disertakan. Baik prosesor tertanam keras (misalnya PowerPC) dan lunak (misalnya NIOS II) untuk FPGA tersedia; prosesor keras tampaknya menurun, mungkin karena peningkatan biaya saat tidak diperlukan. Prosesor, keras atau lunak, mendukung perangkat periferal yang dapat dikonfigurasi pengguna yang diimplementasikan pada FPGA dan menjalankan sistem operasi umum seperti Linux.

3.4 *Soft Processor*

Prosesor inti lunak adalah model bahasa deskripsi perangkat keras (HDL) dari prosesor tertentu (CPU) yang dapat disesuaikan untuk aplikasi tertentu dan disintesis untuk target ASIC atau FPGA [5]. Dalam banyak aplikasi, prosesor inti lunak memberikan beberapa keunggulan dibandingkan prosesor yang dirancang khusus seperti pengurangan biaya, fleksibilitas, kemandirian platform, dan kekebalan yang lebih besar terhadap keusangan. Sistem tertanam adalah komponen perangkat keras dan perangkat lunak yang

bekerja bersama untuk melakukan fungsi tertentu. Biasanya mereka berisi prosesor tertanam yang sering dalam bentuk prosesor inti lunak yang mengeksekusi kode perangkat lunak. Makalah ini menyajikan survei prosesor soft-core yang digunakan dalam sistem tertanam. Beberapa prosesor inti lunak yang tersedia dari vendor komersial dan komunitas sumber terbuka ditinjau dan dibandingkan berdasarkan fitur arsitektur utama. Selain itu, beberapa contoh dunia nyata dari sistem tertanam yang menggunakan prosesor inti lunak dirangkum. Karena kompleksitas sistem tertanam terus meningkat, diharapkan penggunaan prosesor soft-core yang dapat disesuaikan akan menjadi lebih luas.

3.5 Quartus Prime

Intel Quartus Prime adalah perangkat lunak desain perangkat logika yang dapat diprogram yang diproduksi oleh Intel [6]. Quartus Prime memungkinkan analisis dan sintesis desain HDL, yang memungkinkan pengembang untuk mengkompilasi desain mereka, melakukan analisis waktu, memeriksa diagram RTL, mensimulasikan reaksi desain terhadap rangsangan yang berbeda, dan mengkonfigurasi perangkat target dengan programmer. Quartus Prime mencakup implementasi VHDL dan Verilog untuk deskripsi perangkat keras, pengeditan visual sirkuit logika, dan simulasi bentuk gelombang vektor.

3.6 ModelSim

ModelSim adalah alat pengembangan yang disesuaikan dengan insinyur desain digital [7]. Ini adalah mesin simulasi yang dioptimalkan untuk deskripsi SystemC, Verilog, dan VHDL dari sistem perangkat keras tetapi juga berisi tampilan bentuk gelombang yang kuat dan fitur ekspor data simulasi yang penting untuk debugging dan verifikasi desain perangkat keras digital.

3.7 VHDL (*VHSIC Hardware Description Language*)

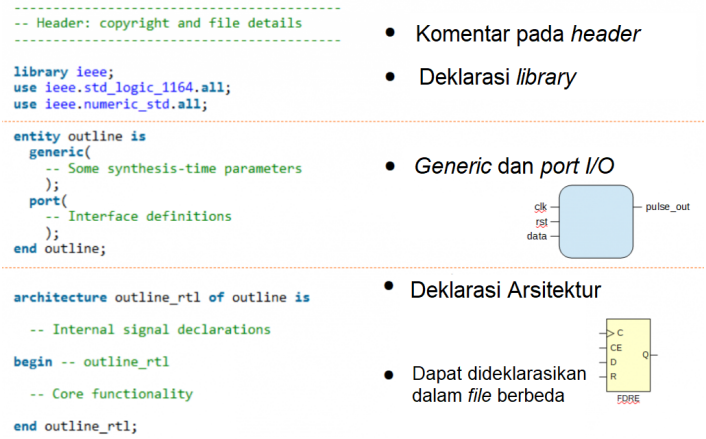
VHDL (*VHSIC Hardware Description Language*) merupakan bahasa pemodelan sistem digital untuk sirkuit terintegrasi berke-

cepatan tinggi. Pada bulan Maret 1980, Departemen Pertahanan meluncurkan program *Very High Speed Integrated Circuit* untuk memajukan teknologi sirkuit terpadu berkecepatan tinggi, khususnya untuk sistem pertahanan [8]. Pada awal program VHSIC, menjadi jelas bahwa bahasa deskripsi perangkat keras standar diperlukan untuk mengomunikasikan data desain, dan pada musim panas 1981, Institut Analisis Pertahanan mengatur lokakarya untuk menentukan persyaratan untuk standar semacam itu.

Departemen Pertahanan menggunakan laporan akhir dari lokakarya IDA sebagai dasar untuk menentukan seperangkat persyaratan bahasa untuk Bahasa Deskripsi Perangkat Keras VHSIC, mengeluarkan permintaan proposal untuk pengadaan dua fase VHDL dan lingkungan pendukungnya. Tim Intermetrics, IBM, dan Texas Instruments dianugerahi kontrak untuk merancang dan mengimplementasikan VHDL dan lingkungan pendukungnya. Program, yang dimulai pada 31 Juli 1983, membutuhkan fase desain 12 bulan, diikuti dengan dua bulan tinjauan desain dan periode implementasi 14 bulan. Salah satu persyaratan utama adalah bahwa VHDL menggunakan konstruksi Ada4 sedapat mungkin.

VHDL mendukung desain, dokumentasi, dan simulasi perangkat keras yang efisien dari level sistem digital hingga level gerbang. Meskipun dirancang untuk tidak bergantung pada teknologi, metodologi desain, atau alat lingkungan yang mendasarinya, bahasa ini juga dapat diperluas ke berbagai teknologi perangkat keras, metodologi desain, dan berbagai kebutuhan informasi alat otomasi desain.

Tujuan utama dari VHDL adalah untuk mendukung penyisipan teknologi-istilah yang diberikan untuk penggunaan teknologi terbaru dalam pengembangan sistem baru dan peningkatan yang sudah ada untuk bersantai persyaratan lingkungan atau untuk meningkatkan kinerja. VHDL diharapkan dapat mengurangi jeda waktu dan biaya yang terlibat dalam penyisipan teknologi.



Gambar 3.3: Bagian-bagian dari kode VHDL. [1]

Gambar 3.3 merupakan struktur dasar dari kode dalam bahasa pemodelan VHDL. *File* VHDL dimulai dengan beberapa pernyataan *import* perpustakaan, diikuti oleh 2 bagian utama: entitas dan arsitektur. Bagian entitas adalah tempat penentuan pin I/O modul (atau pin chip, jika itu menggambarkan level teratas). Ini terdiri dari satu atau dua bagian: generik dan port. Bagian umum adalah tempat penjelasan parameter yang terkunci setelah desain disintesis. Jika ingin dideskripsikan bus data yang lebarnya 'n' bit, maka dapat didefinisikan apa itu 'n' di bagian ini, lalu rujuk ke bagian port di bawah ini. Jika modul digunakan dalam desain hierarkis, maka setiap instance modul ini dapat memiliki nilai yang berbeda. Bagian port adalah tempat dijelaskan kabel dan bus yang masuk dan keluar dari modul. Bagian bawah file adalah arsitektur dan di situlah dapat digambarkan fungsionalitas desain. Ini memiliki 2 bagian: deklarasi dan fungsionalitas inti. Di antara arsitektur dan pernyataan awal, dapat ditentukan sinyal, konstanta, tipe, catatan, dan semua hal lain yang akan digunakan di bagian selanjutnya. Terakhir, di antara pernyataan awal dan akhir, dijelaskan apa yang sebenarnya dilakukan oleh fungsionalitas inti.

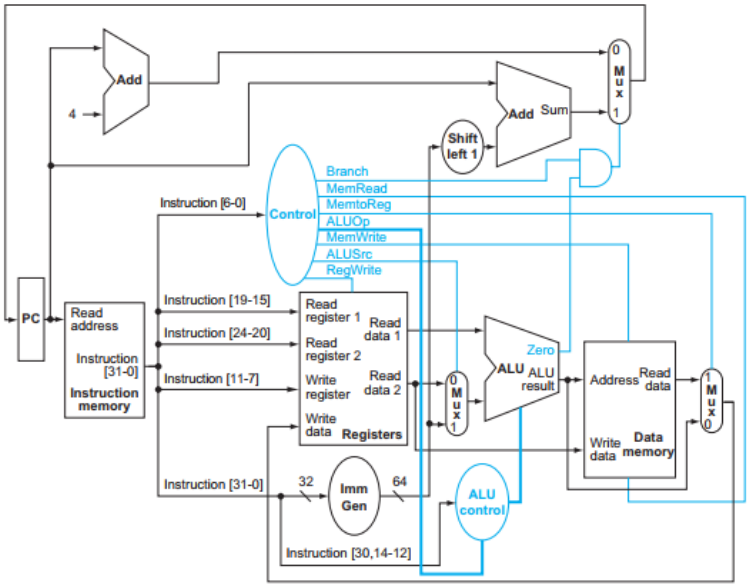
3.8 UART (*UART atau Universal Asynchronous Receiver-Transmitter*)

UART atau *Universal Asynchronous Receiver-Transmitter* merupakan protokol komunikasi *serial* pada sistem komputer. Motivasi utama di balik implementasi UART pada FPGA adalah: UART tersedia langsung di mikrokontroler untuk komunikasi serial tetapi ini tidak tersedia dalam kasus FPGA. Ini terbukti bermanfaat untuk menyediakan antarmuka antara FPGA dan beberapa modul atau perangkat komunikasi seperti modul GSM, dll. UART pada dasarnya digunakan di antara perangkat periferal yang lambat dan cepat misalnya: komputer dan printer atau di antara pengontrol dan LCD. Karena alasan ini, UART sebagian besar digunakan untuk jarak pendek, kecepatan rendah, dan biaya rendah. Blok utama untuk perancangan UART adalah clock generator, transmitter, dan receiver. Diagram blok pemancar dan penerima diimplementasikan menggunakan mesin keadaan terbatas.

Penerima/pemancar asinkron universal disingkat UART. UART pada dasarnya digunakan di antara perangkat periferal yang lambat dan cepat misalnya: komputer dan printer atau di antara pengontrol dan LCD. Karena alasan ini, UART sebagian besar digunakan untuk jarak pendek, kecepatan rendah, dan biaya rendah [9]. Makalah ini menggunakan bahasa deskripsi Verilog untuk mengimplementasikan fungsi inti UART dan mengintegrasikannya ke dalam chip FPGA. UART memainkan peran penting dalam sistem kontrol kompleks modern, desain IC kontrol gerak, Sistem Akuisisi Data Berkecepatan Tinggi yang Andal, algoritma AES untuk berkomunikasi dengan cepat dan efektif. Ini memiliki tiga komponen utama yaitu pemancar, penerima dan generator baud rate. Karena kami menggunakan mesin negara untuk pemancar dan penerima karena desain kami menjadi kurang kompleks dan UART yang diusulkan menjadi lebih stabil, andal dan kompak untuk komunikasi data serial. Karena itu, konsumsi LUT, sandal jepit atau singkatnya konsumsi area chip menjadi lebih sedikit. Kami juga telah menguji desain kami untuk kesalahan yang muncul selama transmisi data untuk menganalisis bahwa keluaran penerima kami bebas dari kesalahan atau tidak.

BAB IV
DESAIN DAN IMPLEMENTASI

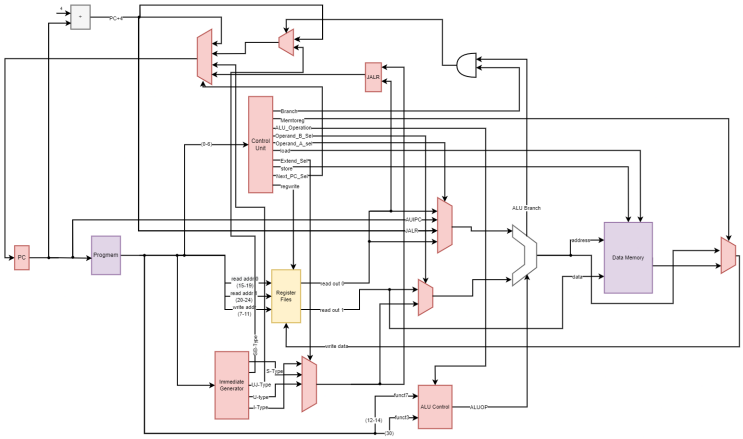
4.1 Deskripsi Sistem



Gambar 4.1: Arsitektur dari *soft processor single-cycle* RV32I sederhana [2].

Gambar 4.1 merupakan representasi sederhana dari *soft processor* RISC-V yang mendukung sebagian instruksi [2]. *Datapath* dalam desain ini mendukung sebagian kecil instruksi dari *Instruction Set Architecture* RISC-V RV32I. Arsitektur ini mampu menjalankan suatu instruksi dalam satu siklus *clock*. Instruksi yang belum didukung misalnya JAL, JALR, dan AUIPC. Dalam proyek ini, desain

arsitektur sederhana ini akan dikembangkan menjadi *soft processor* yang mampu menjalankan semua instruksi RV32I *ISA* kecuali ECALL, FENCE, dan CSSR.



Gambar 4.2: Arsitektur dari *soft processor single-cycle* RV32I.

Gambar 4.2 merupakan arsitektur *top-level* dari *soft processor* yang telah dirancang. Prosesor ini dapat menjalankan sebuah instruksi dalam satu *clock cycle*. Instruksi mula-mula berasal dari *Program Memory* yang mendapatkan alamat instruksi dari *Program Counter*. Instruksi ini kemudian akan didekodekan dan didapatkan sinyal kontrolnya. Sinyal kontrol ini akan memilih *operand* dan tipe operasi *ALU* yang akan dijalankan. Sinyal *immediate* akan dihasilkan oleh *immediate generator*. Tipe dari *immediate* yang digunakan tergantung dengan tipe instruksi. Penggunaan *register files* dan *data memory* juga akan ditentukan sesuai tipe instruksi.

4.1.1 Spesifikasi Perangkat

Dalam proyek ini, telah digunakan sejumlah perangkat baik perangkat keras maupun perangkat lunak, diantarnya:

1. *Personal Computer*

   Komputer akan digunakan untuk melakukan pemodelan *soft*

processor dan pemrograman *Field Programmable Gate Array*. Komputer digunakan untuk menjalankan perangkat lunak pemodelan dan simulasi, misalnya: Logisim, ModelSim, Ripes, dan Quartus Prime.

- Prosesor: AMD Ryzen 5 1600
- *RAM*: 16 GB
- *GPU*: GTX 1080
- *Storage*: 256 GB

2. *Field Programmable Gate Array*

   FPGA merupakan perangkat logika yang dapat diprogram. *FPGA* akan digunakan untuk menjalankan *soft processor* hasil sintesis.

   - Tipe: EP4CE6E22C8N
   - *Logic Elements*: 6272
   - *Memory bits*: 276480
   - *Logic Array Blocks*: 392
   - *SDRAM*: 64Mbit

3. *USB Blaster*

   USB Blaster merupakan perangkat yang digunakan untuk mengunggah *bitstream* hasil kompilasi *soft processor* dari komputer ke *FPGA*. Dapat digunakan koneksi JTAG maupun *Active Serial* sesuai dengan kebutuhan. Pada mode JTAG, konfigurasi *bitstream* yang diunggah hanya bertahan hingga *FPGA* dimatikan. Sedangkan, pada mode *Active Serial*, *bitstream* akan disimpan dalam memori *flash* yang akan dituliskan kembali pada saat *FPGA* dinyalakan.

4. Kabel RS-232

   Merupakan kabel yang digunakan oleh *FPGA* untuk berkomunikasi melalui protokol *UART* dengan komputer.

5. Quartus Prime *design software*

   Merupakan perangkat lunak yang digunakan untuk melakukan pemodelan, kompilasi, dan analisa desain digital dalam bentuk *Hardware Description Language*.

6. ModelSim *simulation software*

   Merupakan perangkat lunak yang digunakan untuk mensimulasikan gelombang-gelombang logika pada *file* pemodelan perangkat keras yang dikompilasi dari berkas *Hardware Description Language*.

7. Ripes *simulator*

   Merupakan perangkat lunak simulasi khusus prosesor dengan *Instruction Set Architecture* RISC-V. Memiliki antarmuka pengguna yang interaktif untuk pembelajaran mikroarsitektur berbasiskan RISC-V *ISA*. Perangkat lunak ini digunakan untuk membandingkan hasil penjalanan instruksi dari mikroarsitektur yang dirancang dan mikroarsitektur yang hasilnya sudah tepat.

8. Logisim Merupakan perangkat lunak yang digunakan untuk mensimulasikan desain *gate-level* dari desain prosesor secara interaktif. Digunakan dalam salah satu referensi yang menjadi acuan proyek.

4.2 Implementasi Alat

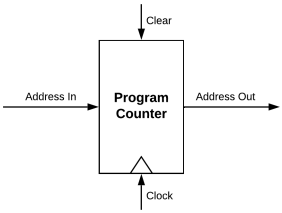
4.2.1 Pemodelan arsitektur *soft processor* menggunakan bahasa pemodelan VHDL

Pada tahap ini, desain *gate level* dari komponen-komponen desain *top-level* akan dimodelkan secara *behavioral* menggunakan *Hardware Description Language* VHDL. Proses pemodelan sistem dapat dilakukan dalam Quartus Prime, ModelSim, maupun *text editor* biasa. Bahasa pemodelan perangkat keras yang digunakan yakni VHDL. Gambar 4.3 merupakan contoh penulisan kode VHDL menggunakan Quartus Prime.

Gambar 4.3: Contoh penulisan program VHDL dalam Quartus Prime

Program Counter

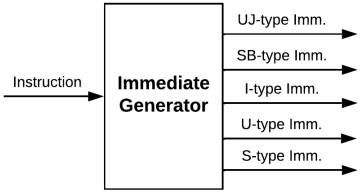
Program Counter adalah register dalam prosesor komputer yang berisi alamat (lokasi) instruksi yang sedang dieksekusi pada saat ini. Saat setiap instruksi diambil, penghitung program meningkatkan nilai simpanannya sebesar 1. Setelah setiap instruksi diambil, penghitung program menunjuk ke instruksi berikutnya dalam urutan. Saat komputer dihidupkan ulang atau direset, *Program Counter* kembali ke alamat 0. Secara *RTL-Level*, *Program Counter* bekerja seperti sebuah *register* biasa. *Program Counter* dimodelkan secara *behavioral* seperti pada *Listing* 7.1. Gambar 4.4 merupakan diagram blok *RTL* dari *Program Counter*.



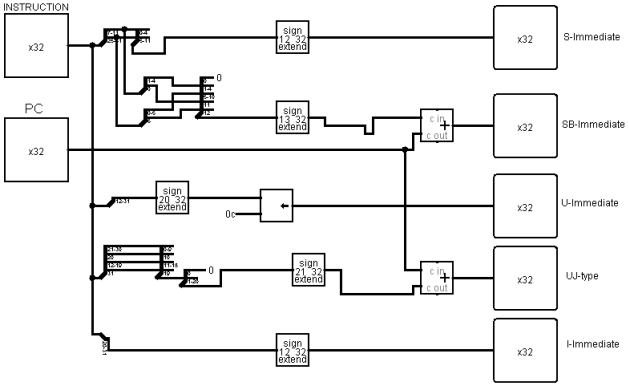
Gambar 4.4: *Program Counter* dari *soft processor*.

Immediate Generator

Immediate Generator akan menghasilkan nilai *immediate* dari instruksi sesuai dengan tipe instruksi. *Immediate* ini akan digunakan sebagai *offset* atau *operand* untuk proses kalkulasi. *Immediate Generator* dimodelkan secara *behavioral* seperti pada *Listing* 7.3. Gambar 4.5 merupakan diagram blok *RTL* dari *immediate generator*.



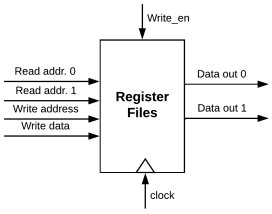
Gambar 4.5: *Immediate Generator* dari *soft processor*.



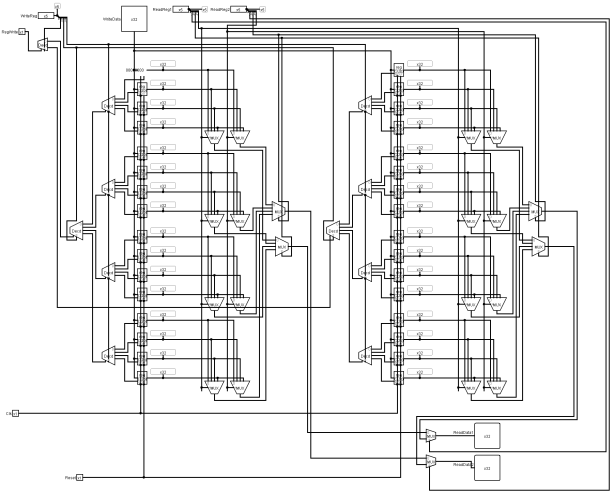
Gambar 4.6: Implementasi *RTL-Level* dari *Immediate Generator*.

Register Files

Register Files menyimpan nilai register x0 hingga x31 yang dapat diakses oleh pemrogram. Nilai x0 selalu bernilai 0 sesuai dengan spesifikasi RISC-V. *Register Files* dimodelkan secara *behavioral* seperti pada *Listing* 7.4. Gambar 4.7 merupakan diagram blok *RTL* dari *register files*.



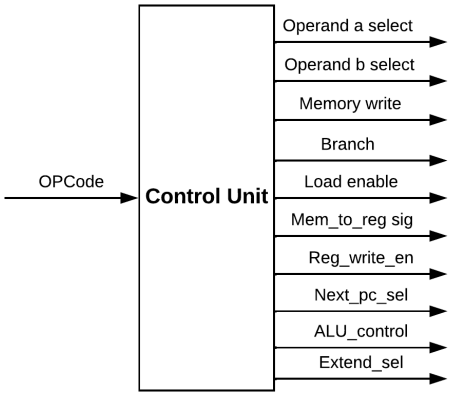
Gambar 4.7: *Register Files* dari *soft processor*.



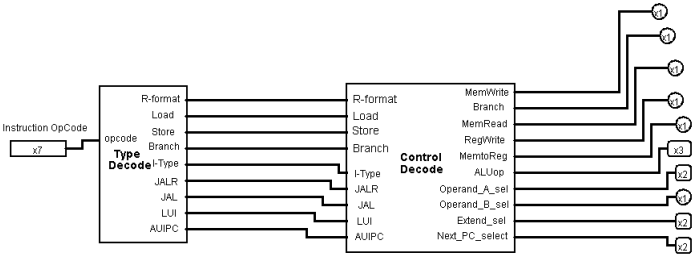
Gambar 4.8: Implementasi *RTL-Level* dari *Register Files*.

Control Unit

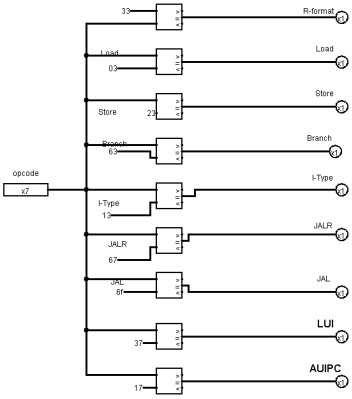
Control unit akan mengendalikan sistem secara keseluruhan sesuai dengan tipe instruksi. Tipe instruksi didapatkan dari sejumlah bits pertama pada instruksi dalam bentuk *OP code*. Desain *Control Unit* dibagi menjadi dua tahap: *type-decode* dan *control-decode*. Tahap pertama akan menentukan tipe dari instruksi, sedangkan tahap kedua akan menghasilkan sinyal kontrol dari tipe instruksi tersebut. *Control Unit* dimodelkan secara *behavioral* seperti pada *Listing* 7.8. Gambar 4.9 merupakan diagram blok *RTL* dari *control unit*. Sedangkan, Gambar 4.10 merupakan ilustrasi tahap-tahap dalam *control unit*. Gambar 4.10 merupakan diagram blok *RTL-level* dari tahap *type decode* dan Gambar 4.12 merupakan diagram blok *RTL-level* dari tahap *control decode* dalam *control unit*.



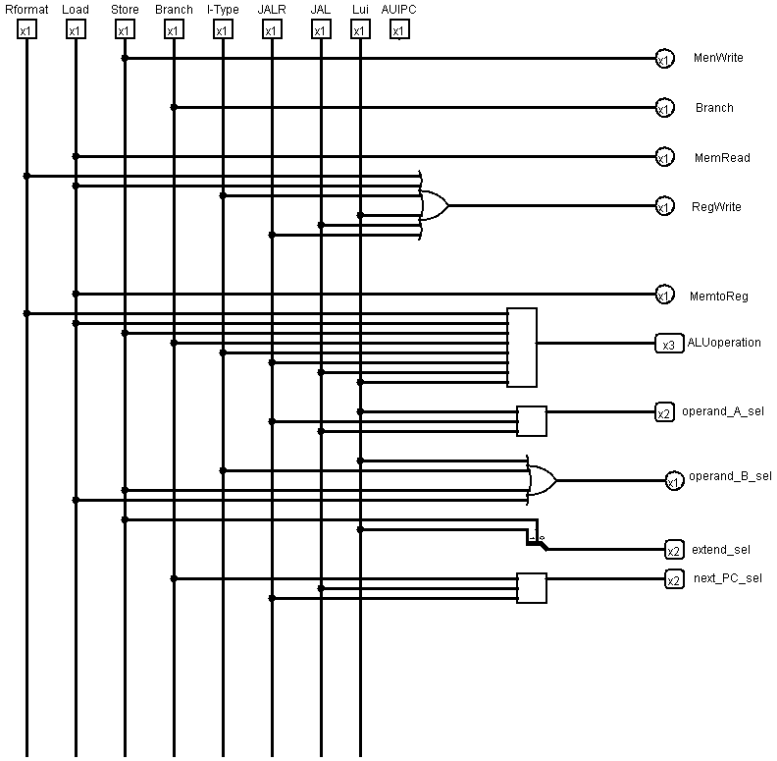
Gambar 4.9: *Control Unit* dari *soft processor*.



Gambar 4.10: Implementasi *RTL-Level* dari *Control Unit*.



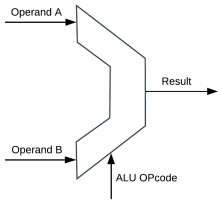
Gambar 4.11: Implementasi *RTL-Level* tahap *Type Decode* dari *Control Unit*.



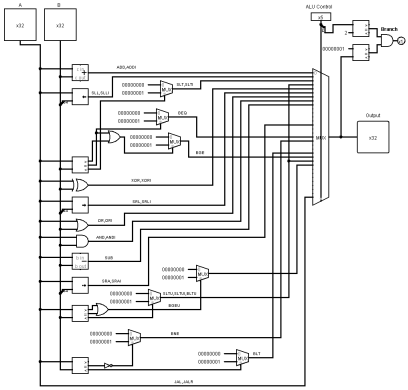
Gambar 4.12: Implementasi *RTL-Level* tahap *Control Decode* dari *Control Unit*.

ALU (Arithmetic Logic Unit)

ALU (Arithmetic Logic Unit) merupakan otak kalkulasi pada prosesor. Dalam ALU terdapat sejumlah logika operasi seperti adisi, substraksi, *shifting*, *and*, *or*, dan lainnya. Output dari *ALU* ditentukan oleh *ALU Controller*. *ALU* dimodelkan secara *behavioral* seperti pada *Listing* 7.6. Gambar 4.13 merupakan diagram blok *RTL* dari *Arithmetic Logic Unit*.



Gambar 4.13: *ALU* dari *soft processor*.

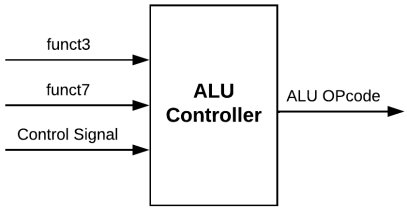


Gambar 4.14: Implementasi *Gate-Level* dari *ALU*.

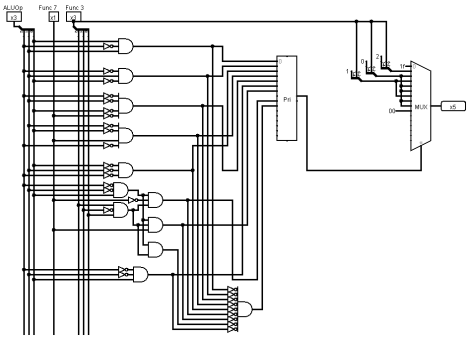
ALU Controller

ALU Controller akan memberikan sinyal kendali sesuai dengan operasi yang diperlukan suatu instruksi. *ALU Controller* dimodelk-

an secara *behavioral* seperti pada *Listing* 7.7. Gambar 4.15 merupakan diagram blok *RTL* dari *Arithmetic Logic Unit controller*.



Gambar 4.15: *ALU Controller* dari *soft processor*.



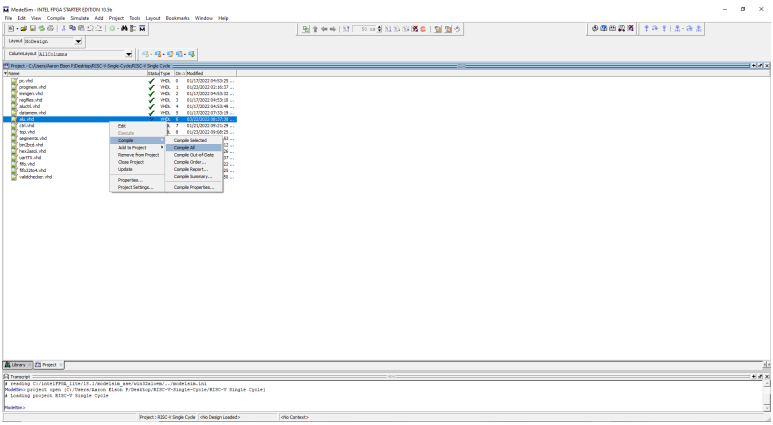
Gambar 4.16: Implementasi *Gate-Level* dari *ALU Controller*.

4.2.2 Pengujian logika *soft processor* menggunakan ModelSim

Untuk memverifikasi validitas dari model yang dibuat, akan digunakan *software* simulasi ModelSim. Untuk menggunakan ModelSim, akan dilakukan sejumlah tahap:

1. *Import* semua *file* VHDL yang sudah ditulis ke *Project* ModelSim.

2. Lakukan kompilasi terhadap semua file yang sudah di-*import*.



Gambar 4.17: Melakukan kompilasi terhadap semua file desain dalam ModelSim.

3. Pastikan semua *file* dari desain sudah valid.

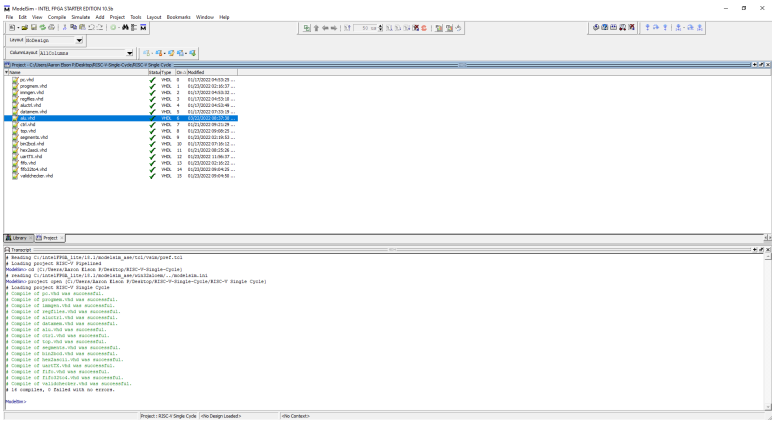
4. Lakukan simulasi logika terhadap desain *Top-Level*. Tahap ini akan dijelaskan pada bab berikutnya.

4.2.3 Sintesis *processor core* dalam Quartus Prime

Setelah *soft processor* diverifikasi menggunakan ModelSim, selanjutnya *core* akan disintesis menggunakan *tool* dari *vendor FPGA*. Karena *FPGA* yang digunakan diproduksi oleh Altera dan Intel, maka akan digunakan *design software* Quartus Prime. Untuk mensintesis *soft processor*, dilakukan sejumlah tahap:

1. Lakukan *Analysis & Synthesis*

2. Masukkan informasi mengenai *Pinout FPGA* yang digunakan pada *menu Pin Planner*.



Gambar 4.18: Semua *file* sudah valid, ditunjukkan dengan tanda centang hijau.

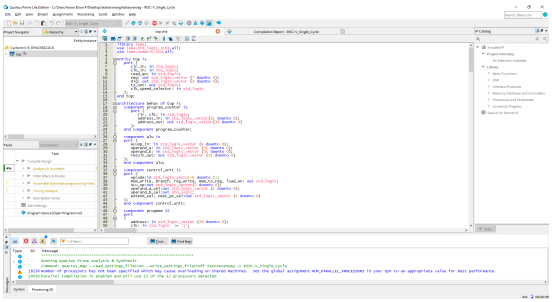
3. Melanjutkan proses kompilasi mulai dari *Place & Route* hingga selesai.

4. Dapat dilakukan *Timing Analysis* untuk menganalisa performa dari *soft processor*. Tahap ini akan dibahas di bab selanjutnya.

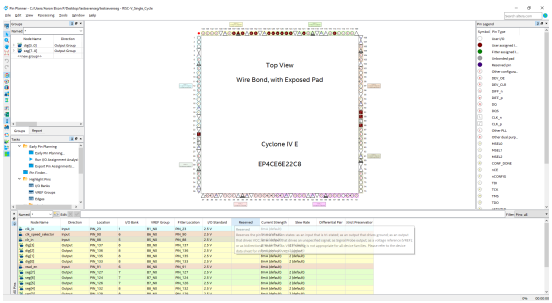
4.2.4 Pengunggahan *processor core* ke dalam *FPGA* menggunakan *USB Blaster*

Soft Processor dapat diunggah ke *FPGA* setelah proses kompilasi selesai. Untuk melakukan proses pengunggahan, dilakukan tahap-tahap berikut:

1. Ubah hasil kompilasi dari bentuk ".sof" ke bentuk ".pof". Hasil kompilasi berbentuk ".sof" digunakan untuk pengunggahan dalam mode *JTAG*, sedangkan ".pof" digunakan dalam mode *Active Serial*. *Mode JTAG* hanya bertahan selama daya menyala dan akan terhapus setelah *FPGA* di-*restart*. Sebaliknya, dalam mode *Active Serial*, *bitstream* yang diunggah akan terus tersimpan. Proses ini dapat dilakukan pada *menu Convert Programming Files*.



Gambar 4.19: Proses *analysis & synthesis*.

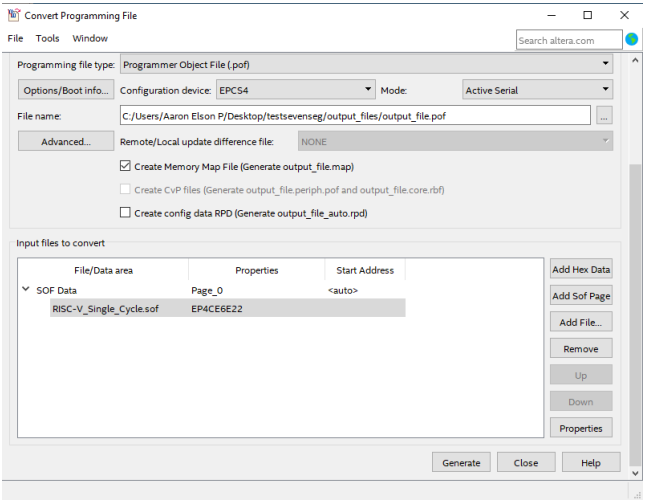


Gambar 4.20: *Pin Planner* pada Quartus Prime.

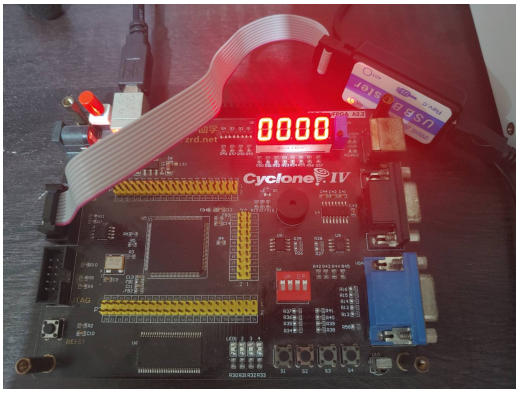
2. Hubungkan *FPGA board*, *USB Blaster*, dan komputer.

3. Unggah *file ".pof"* melalui *menu Programmer* Quartus Prime. *Mode* yang digunakan yakni *Active Serial*.

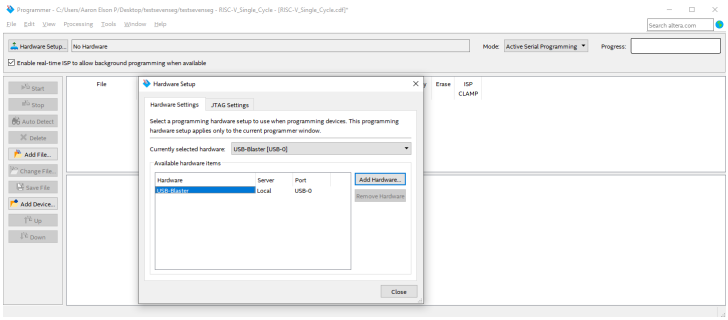
4. Setelah berhasil, akan dilakukan pengujian terhadap penjalanan program pada *soft processor* yang telah diunggah ke *FPGA*. Tahap ini akan dibahas dalam bab berikutnya.



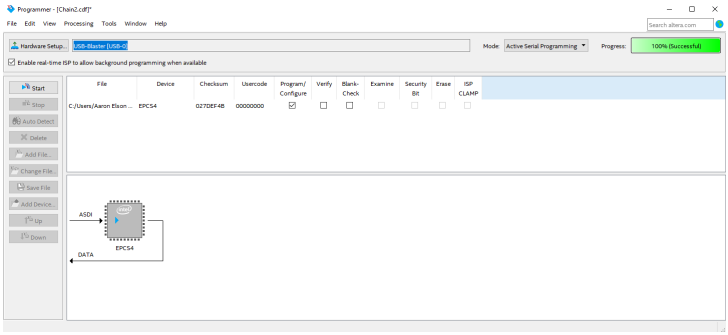
Gambar 4.21: *Menu Convert Programming Files* pada Quartus Prime.



Gambar 4.22: Koneksi *FPGA*, *USB Blaster*, daya, dan komputer.



Gambar 4.23: *USB Blaster* digunakan untuk mengunggah *soft processor* dalam mode *Active Serial*.



Gambar 4.24: Proses pengunggahan ke *FPGA* telah berhasil.

Halaman ini sengaja dikosongkan

BAB V
PENGUJIAN DAN EVALUASI

5.1 Program pengujian

Untuk menguji *soft processor*, telah digunakan program deret Fibonacci. Program dituliskan dalam bahasa Assembly RISC-V dan akan di-kompilasi ke dalam bentuk biner. Program ini akan menghitung deret Fibonacci dimulai dari yang pertama hingga bilangan Fibonacci dibawah 9999. Setelah nilai melampaui 9999, program akan berhenti dengan cara *Branch Greater Equal*. Listing 5.1 merupakan algoritma program dalam bahasa Assembly.

Listing 5.1: Deret Fibonacci dibawah 9999 dalam RISC-V Assembly.

```
1      lui x15 0x2
2      addi x10 x15 1807
3  start:
4      addi x6,x0,1
5      addi x5,x0, 0
6  up:
7      add x7,x5,x6
8      bge x7,x10,stop
9      sw x7,0x34(x0)
10     sw x6,0x30(x0)
11     lw x5,0x30(x0)
12     sw x7,0x30(x0)
13     lw x6,0x30(x0)
14     jal up
15 stop:
16     nop
```

5.2 Kompilasi Program Pengujian

Agar dapat digunakan dalam *soft processor*, program Assembly ini perlu dikompilasi terlebih dahulu menjadi bentuk biner. Untuk melakukan kompilasi, dapat digunakan GCC *toolchain* untuk RISC-V. Supaya lebih mudah, dapat digunakan menu kompilasi bawaan Ripes *simulator*, yang sudah terintegrasi dengan RISC-V GCC *to-*

olchain. Program yang sudah di-*disassembly* akan terlihat setelah proses kompilasi berhasil. Setelah kode biner didapatkan, program biner akan dimasukkan ke dalam *Program Memory* seperti pada Listing 7.2.

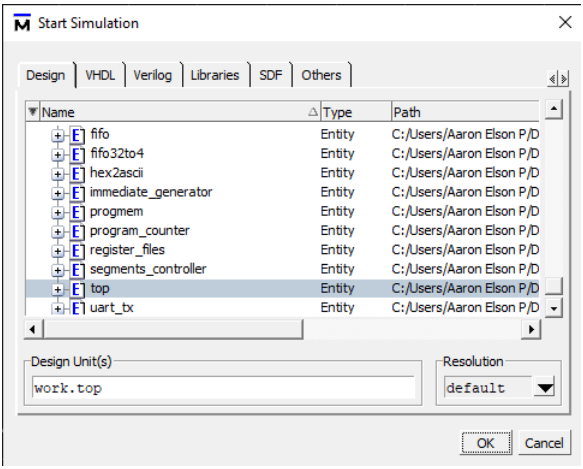
```
    0:      000027b7        lui x15 0x2
    4:      70f78513        addi x10 x15 1807

00000008 <start>:
    8:      00100313        addi x6 x0 1
    c:      00000293        addi x5 x0 0

00000010 <up>:
    10:     006283b3        add x7 x5 x6
    14:     00a3de63        bge x7 x10 28 <stop>
    18:     02702a23        sw x7 52 x0
    1c:     02602823        sw x6 48 x0
    20:     03002283        lw x5 48 x0
    24:     02702823        sw x7 48 x0
    28:     03002303        lw x6 48 x0
    2c:     fe5ff0ef        jal x1 -28 <up>

00000030 <stop>:
    30:     00000013        addi x0 x0 0
```

Gambar 5.1: Hasil *deassembly* menggunakan GCC *toolchain* bawaan Ripes *simulator*.

5.3 Simulasi pada ModelSim

Setelah proses kompilasi desain menggunakan ModelSim seperti pada sub-bab 4.2.2, desain dapat disimulasikan untuk memastikan validitas logika. Untuk memulai simulasi, dapat menggunakan *menu simulation*, kemudian memilih hasil kompilasi *top-level* dari *soft processor*. Pada saat masuk ke dalam *mode* simulasi, semua sinyal yang ingin diamati perlu diletakkan ke dalam *window waveform* seperti pada gambar 5.3. Register `x5`, `x6`, dan `x7` pada Gambar 5.4 sedang dilakukan proses kalkulasi bilangan Fibonacci. Bilangan yang ditunjukkan berbasis heksadesimal. Dalam *waveform* simulasi, *soft processor* telah mampu melakukan kalkulasi bilangan Fibonacci dibawah 9.999. Pada Gambar 5.5, telah dilakukan *branching* ke akhir program pada saat register `x7` bernilai lebih dari 9.999. Bilangan

yang ditunjukkan berbasis heksadesimal.



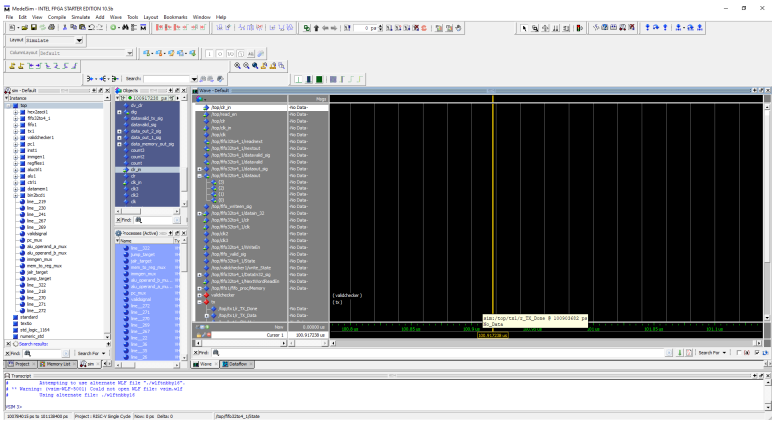
Gambar 5.2: Memulai proses simulasi pada ModelSim.

5.4 *Resource Usage* pada Sintesis *Processor Core*

Resource Usage dari *soft processor* dapat dilihat dalam *report* tahap *Analysis & Synthesis* proses kompilasi. Dari *report* yang didapatkan, dapat dilihat bahwa *soft processor* membutuhkan 3712 *logic elements*, 1473 *register*, dan 16384 *memory bits*. *Memory bits* yang dibutuhkan dapat ditambah atau dikurangi dengan mengubah total *address* memori pada *data memory* dan *program memory*. Gambar 5.6 merupakan laporan penggunaan sumber daya hasil kompilasi *soft processor* yang dirancang.

5.5 Performa pada *Processor Core*

Performa dari *processor core* dapat diukur melalui *frekuensi* maksimal. Frekuensi maksimal dari *soft processor* dapat diamati dalam jendela *Timing Analysis* pada proses kompilasi. Frekuensi maksimal dari *soft processor* yang telah dirancang yakni 46.6 MHz. Ini terlihat pada ringkasan analisa *timing* pada Gambar 5.7.



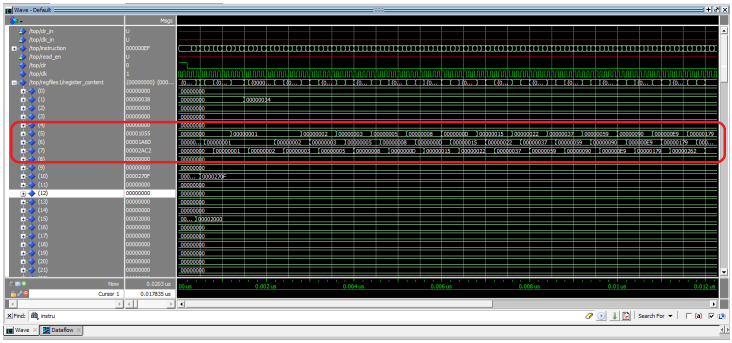
Gambar 5.3: Memasukkan sinyal yang ingin diamati ke jendela *waveform*.

5.6 *Critical Path* pada *Processor Core*

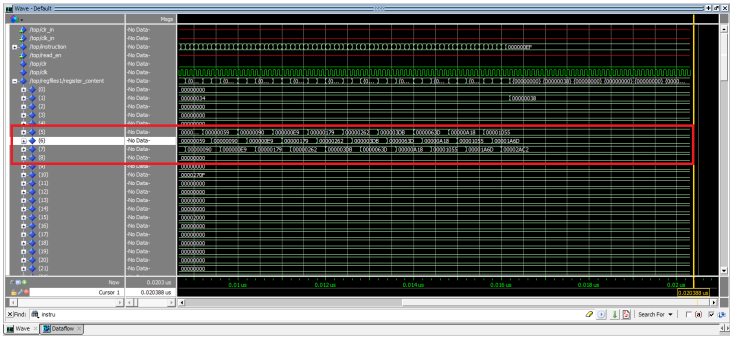
Critical Path merupakan jalur terpanjang dari *soft processor* yang akan menentukan frekuensi maksimal. Dalam *Timing Analysis*, dapat dilakukan analisa lebih lanjut mengenai *critical path* dari desain. Untuk memulai analisa *critical path*, dipilih menu *report critical path*. Dalam jendela *Timing Analyzer*, dapat dipilih opsi *report timing*. Jalur terpanjang dari desain yakni jalur yang melalui masukkan *Program Counter* hingga masuk kembali ke multiplekser masukkan *Program Counter*.

5.7 Pembacaan Data Hasil Pembacaan Program Melalui Koneksi *UART*

UART merupakan sebuah protokol *interfacing* yang dapat digunakan untuk meng-*outputkan* sinyal dari *soft processor* secara digital. Dalam *protokol UART*, terdapat komponen TX dan RX. Dari desain yang ada, telah diimplementasikan suatu modul TX untuk komunikasi satu arah antara *FPGA* dan komputer. Pada komputer, data yang ditransmisikan melalui *UART TX* akan dibaca menggunakan *serial monitor*. Pertama-tama, kalkulasi dari program *Fi-*

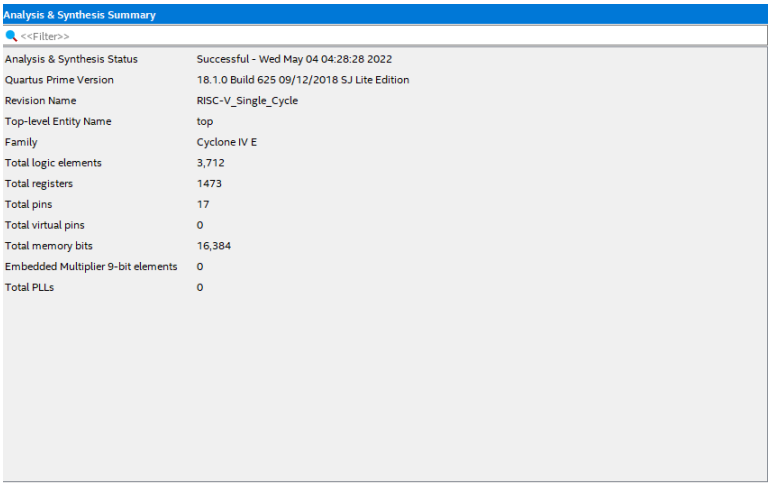


Gambar 5.4: Menjalankan simulasi ModelSim.

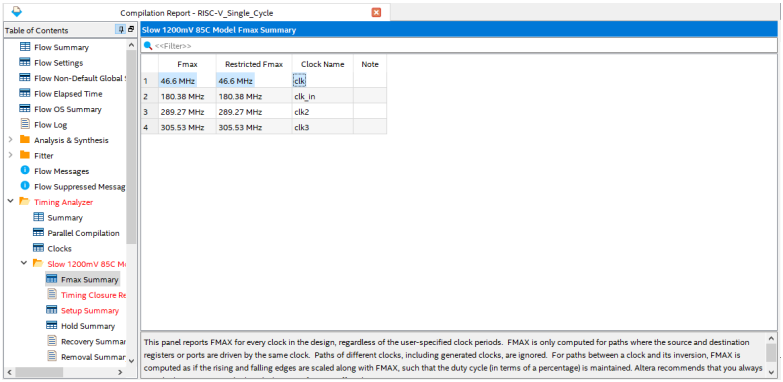


Gambar 5.5: Simulasi ModelSim sudah berhasil.

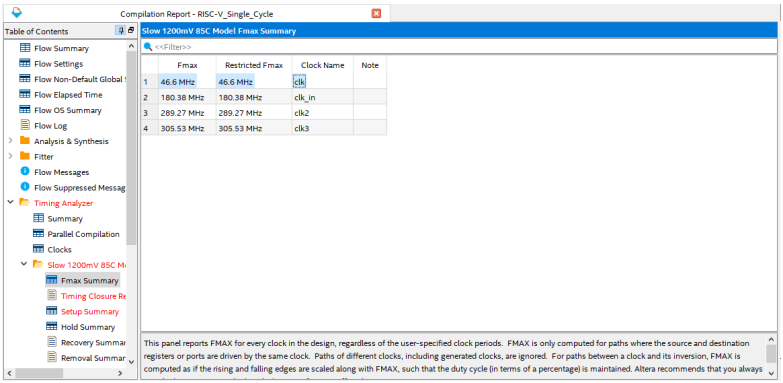
bonacci akan diletakkan dalam sebuah memori *First In First Out*. Nilai yang dimasukkan ke dalam *FIFO memory* adalah nilai dari *register* x6 yang terhubung dengan suatu *probe*. Setelah program selesai, isi dari memori akan dikeluarkan satu-satu pada saat *input* diaktifkan. Hasilnya, semua bilangan Fibonacci sudah berhasil dikalkulasi dan ditampilkan dalam *serial monitor*. Untuk menghubungkan *FPGA board* dan komputer menggunakan protokol *UART*, digunakan kabel RS-232.



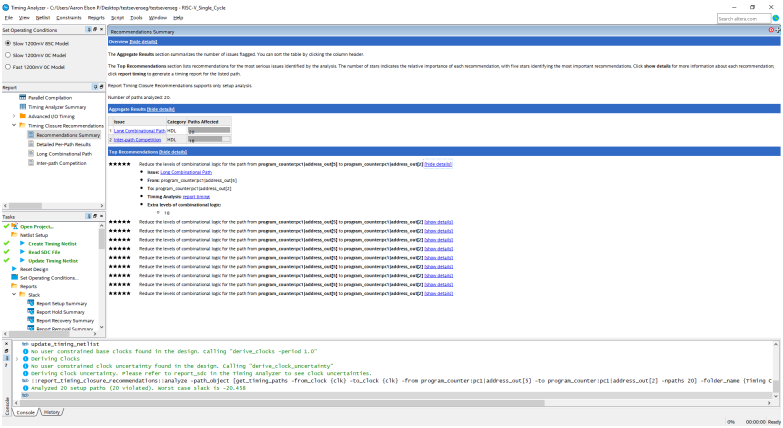
Gambar 5.6: *Resource Usage* pada laporan tahap kompilasi *Analysis& Synthesis* dalam Quartus Prime.



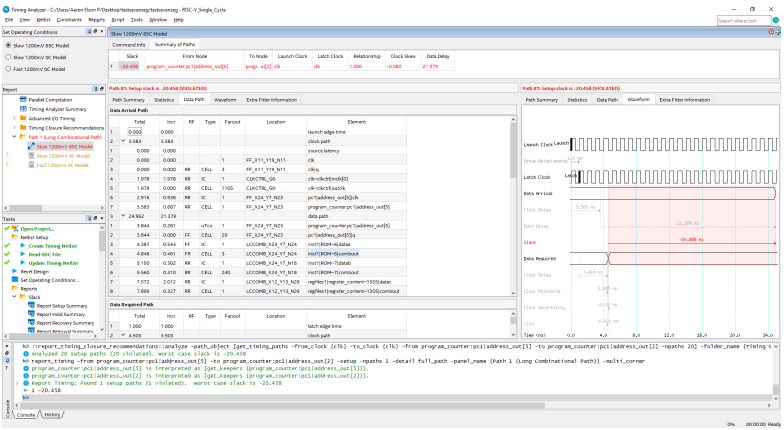
Gambar 5.7: Frekuensi maksimal dari desain.



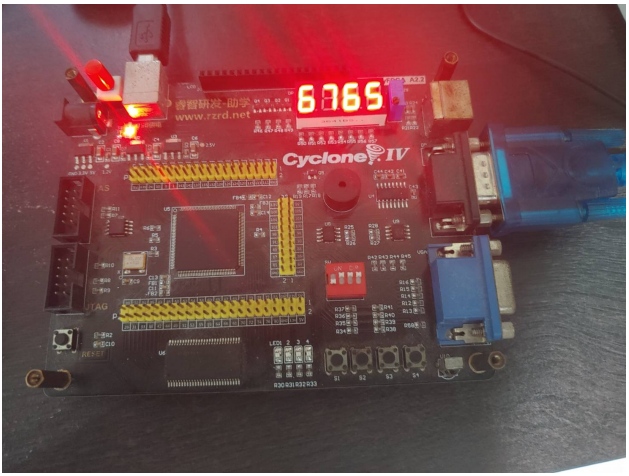
Gambar 5.8: Frekuensi maksimal dari *clock* desain.



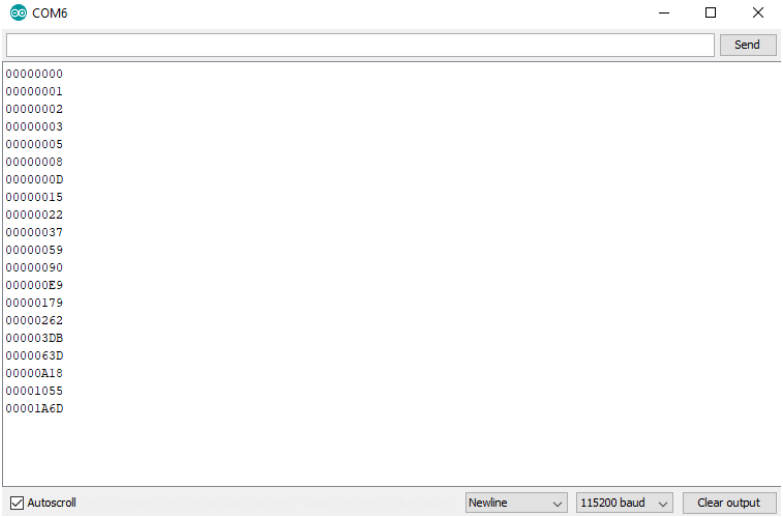
Gambar 5.9: *Report* dan rekomendasi mengenai *timing closure soft processor*.



Gambar 5.10: Jalur terpanjang dari *datapath* yang merupakan *critical path* desain.



Gambar 5.11: Sambungan antara *FPGA board* dan komputer menggunakan kabel RS-232. *Display* menampilkan bilangan Fibonaci terakhir di bawah 9.999.



Gambar 5.12: Nilai dari *FIFO memory* yang dibaca satu per satu melalui transmisi *UART*.

Halaman ini sengaja dikosongkan

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Dari Kerja Praktek yang telah dilaksanakan, dapat ditarik beberapa kesimpulan:

1. *Soft Processor* berbasis RV32I *Instruction Set Architecture* dapat dikembangkan untuk sebuah *FPGA* dalam bentuk *processor core*.

2. Untuk memastikan validitas dari logika prosesor, dapat dilakukan *functional simulation* menggunakan ModelSim.

3. *Resource Usage* dan performa dari *soft processor* dapat diujikan menggunakan fitur yang terdapat dalam Quartus Prime.

4. *Soft processor* yang dirancang memerlukan 3712 *logic elements*, *1473 registers*, dan 16384 *memory bits*.

5. Prosesor memiliki frekuensi maksimal 46.6 MHz. Ini merupakan 93.2% kecepatan maksimum yang didukung FPGA, dimana kristal *on-board* pada *FPGA* yang digunakan mendukung frekuensi hingga 50MHz.

6. Frekuensi ini dipengaruhi oleh jalur paling panjang dari desain, yang disebut sebagai *critical path*.

7. Prosesor dapat menggunakan protokol *UART* untuk melakukan komunikasi antara *input* dan *output*.

6.2 Saran

1. Penggunaan *resource* dapat dioptimasi lebih lanjut dengan mengembangkan desain menjadi lebih efisien.

2. Frekuensi maksimal dapat ditingkatkan dengan menggunakan metode *pipelining*.

3. Fungsi dari prosesor dapat ditingkatkan lebih lanjut dengan mengimplementasikan *extension* RISC-V lainnya sesuai kebutuhan.

4. *Interface UART* dapat ditingkatkan lagi sehingga dapat mendukung komunikasi *transmit-receive* secara dua arah.

5. Verifikasi dapat dijalankan lebih lanjut menggunakan metode *formal verification*.

DAFTAR PUSTAKA

[1] Iain Waugh. How to read vhdl code, Jul 1970. URL `https://cadhut.com/2020/07/26/how-to-read-vhdl-code/`.

[2] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovi. The risc-v instruction set manual. volume 1: User-level isa, version 2.0. 2014.

[3] Don Kurian Dennis, Ayushi Priyam, Sukhpreet Singh Virk, Sajal Agrawal, Tanuj Sharma, Arijit Mondal, and Kailash Chandra Ray. Single cycle risc-v micro architecture processor and its fpga prototype. In *2017 7th International Symposium on Embedded Computing and System Design (ISED)*, pages 1–5, 2017. doi: 10.1109/ISED.2017.8303926.

[4] Philip H. W. Leong. Recent trends in fpga architectures and applications. In *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*, pages 137–141, 2008. doi: 10.1109/DELTA.2008.14.

[5] Jason G. Tong, Ian D. L. Anderson, and Mohammed A. S. Khalid. Soft-core processors for embedded systems. In *2006 International Conference on Microelectronics*, pages 170–173, 2006. doi: 10.1109/ICM.2006.373294.

[6] Fpga design software - intel® quartus® prime. URL `https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html`.

[7] Brian Gestner and David V. Anderson. Automatic generation of modelsim-matlab interface for rtl debugging and verification. In *2007 50th Midwest Symposium on Circuits and Systems*, pages 1497–1500, 2007. doi: 10.1109/MWSCAS.2007.4488824.

[8] M. Shahdad, R. Lipsett, E. Marschner, K. Sheehan, and H. Cohen. Vhsic hardware description language. *Computer*, 18(2): 94–103, 1985. doi: 10.1109/MC.1985.1662802.

[9] Jayesh More, Rushank Suryavanshi, Gaurav Dasarwar, S Sivanantham, and K Sivasankaran. Fpga implementation of universal asynchronous transmitter and receiver. In *2015 Online International Conference on Green Engineering and Technologies (IC-GET)*, pages 1–3, 2015. doi: 10.1109/GET.2015.7453796.

BAB VII
Lampiran

Listing 7.1: *Program Counter*

```
library ieee;
use ieee.std_logic_1164.all;

entity program_counter is
  port (
    clr, clk: in std_logic;
    address_in: in std_logic_vector(31 downto 0);
    address_out: out std_logic_vector(31 downto 0)
  );
end program_counter;

architecture behav of program_counter is
begin
  process (clk, clr)
  begin
    if (clr = '1') then
      address_out <= (others => '0');
    elsif rising_edge(clk) then
      address_out <= address_in;
    end if;
  end process;
end behav;
```

Listing 7.2: *Program Memory*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity progmem IS
  port
  (
    address: in std_logic_vector (29 downto 0);
    clk: in std_logic  := '1';
    instruction_out: out std_logic_vector (31 downto 0)
  );
end progmem;
```

```
13
14
15 architecture behav of progmem is
16   type rom_type is array (0 to 255) of std_logic_vector↩
         (31 downto 0);
17   signal ROM : rom_type:=(
18     -- Example program
19     x"00000000",
20     x"000027b7",
21     x"70f78513",
22     x"00100313",
23     x"00000293",
24     x"006283b3",
25     x"00a3de63",
26     x"02702a23",
27     x"02602823",
28     x"03002283",
29     x"02702823",
30     x"03002303",
31     x"fe5ff0ef",
32     x"000000ef",
33     others => x"00000000"
34   );
35 begin
36   instruction_out <= ROM(to_integer(unsigned(address)))↩
       ;
37 end behav;
```

Listing 7.3: *Immediate Generator*

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity immediate_generator is
6    port (
7      inst: in std_logic_vector(31 downto 0);
8      s_type_out: out std_logic_vector(31 downto 0);
9      sb_type_out: out std_logic_vector(31 downto 0);
10     u_type_out: out std_logic_vector(31 downto 0);
11     uj_type_out: out std_logic_vector(31 downto 0);
12     i_type_out: out std_logic_vector(31 downto 0)
13   );
14 end immediate_generator;
15
16 architecture behav of immediate_generator is
17 begin
```

```
18   s_type_out <= std_logic_vector(resize(signed(inst(31 ↩
         downto 25)&inst(11 downto 7)),32));
19   sb_type_out <= std_logic_vector(resize(signed(inst↩
         (31)&inst(7)&inst(30 downto 25)&inst(11 downto 8)↩
         &'0'),32));
20   u_type_out <= std_logic_vector(inst(31)&inst(30 ↩
         downto 20)&inst(19 downto 12))&x"000";
21   uj_type_out <= std_logic_vector(resize(signed(inst↩
         (31)&inst(19 downto 12)&inst(20)&inst(30 downto ↩
         25)&inst(24 downto 21)&'0'),32));
22   i_type_out <= std_logic_vector(resize(signed(inst(31 ↩
         downto 20)),32));
23 end behav;
```

Listing 7.4: *Register Files*

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity register_files is
6    port (
7      write_en, clk: in std_logic;
8      read_address_1: in std_logic_vector(4 downto 0);
9      read_address_2: in std_logic_vector(4 downto 0);
10     write_address: in std_logic_vector(4 downto 0);
11     write_data: in std_logic_vector(31 downto 0);
12     data_out_1: out std_logic_vector(31 downto 0);
13     data_out_2: out std_logic_vector(31 downto 0)
14   );
15 end register_files;
16
17 architecture behav of register_files is
18   type regfiles is array (0 to 31) of std_logic_vector ↩
         (31 downto 0);
19   signal register_content:regfiles := (0 => x↩
         "00000000", others => x"00000000");
20
21 begin
22   process (clk, write_en, write_data) is
23   begin
24     if rising_edge(clk) then
25       if  (write_en = '1') and (not(write_address↩
             ="00000")) then
26         register_content(to_integer(unsigned(↩
               write_address))) <= write_data;
27       end if;
```

```
28      end if;
29    end process;
30    data_out_1 <= register_content(to_integer(unsigned(↩
          read_address_1)));
31    data_out_2 <= register_content(to_integer(unsigned(↩
          read_address_2)));
32  end behav;
```

Listing 7.5: *Data Memory*

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity data_memory IS
6    port (
7      clk: in std_logic;
8      data: in std_logic_vector (31 downto 0);
9      address: in std_logic_vector (7 downto 0);
10     write_en: in std_logic;
11     load_en: in std_logic;
12     q: out std_logic_vector (31 downto 0):= x↩
           "00000000";
13     probe_out: out std_logic_vector (31 downto 0)
14   );
15 end data_memory;
16
17 architecture behav OF data_memory IS
18    type mem is array(0 to 255) of std_logic_vector(31 ↩
          downto 0);
19    signal ram_block : mem:=(
20     others => x"00000000"
21   );
22   signal probe_out_sig: std_logic_vector (31 downto 0);
23   signal addr:integer:=0;
24 begin
25
26   process (clk)
27     begin
28     if rising_edge(clk) then
29         addr <= to_integer(unsigned(address));
30       if (write_en = '1') then
31         ram_block(to_integer(unsigned(address))) <= ↩
               data;
32       end if;
33     end if;
34   end process;
```

```
35   q <= ram_block(addr) when load_en = '1' else (others ↩
         => 'X');
36   probe_out <= ram_block(13);
37 end behav;
```

Listing 7.6: *ALU*

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity alu is
6    port (
7      ALUop_in: in std_logic_vector (4 downto 0);
8      operand_a: in std_logic_vector (31 downto 0);
9      operand_b: in std_logic_vector (31 downto 0);
10     result_out: out std_logic_vector (31 downto 0)
11   );
12 end alu;
13
14 architecture behav of alu is
15   signal lt, eq, neq, gt, lt_u, gt_u: std_logic_vector ↩
         (31 downto 0);
16 begin
17
18   process (operand_a, operand_b, ALUop_in, lt, eq, neq,↩
          gt, lt_u, gt_u)
19   begin
20     case ALUop_in is
21       when "00000" => result_out <= std_logic_vector(↩
             unsigned(operand_a) + unsigned(operand_b));
22       when "00001" => result_out <= std_logic_vector(↩
             shift_left((unsigned(operand_a)), to_integer(↩
             unsigned(operand_b(4 downto 0)))));
23       when "00010" => result_out <= lt;
24       when "00011" => result_out <= lt_u;
25       when "00100" => result_out <= operand_a xor ↩
             operand_b;
26       when "00101" => result_out <= std_logic_vector(↩
             shift_right((unsigned(operand_a)), to_integer↩
             (unsigned(operand_b(4 downto 0)))));
27       when "00110" => result_out <= operand_a or ↩
             operand_b;
28       when "00111" => result_out <= operand_a and ↩
             operand_b;
29       when "01000" => result_out <= std_logic_vector(↩
             unsigned(operand_a) - unsigned(operand_b));
```

```
30        when "01101" => result_out <= std_logic_vector(↩
              shift_right((signed(operand_a)), to_integer(↩
              unsigned(operand_b(4 downto 0)))));
31        when "10000" => result_out <= eq;
32        when "10001" => result_out <= neq;
33        when "10100" => result_out <= lt;
34        when "10101" => result_out <= eq or gt;
35        when "10110" => result_out <= lt_u;
36        when "10111" => result_out <= eq or gt_u;
37        when "11111" => result_out <= operand_a;
38        when others => result_out <= (others => 'X');
39      end case;
40    end process;
41
42    lt <= x"00000001" when (signed(operand_a) < signed (↩
          operand_b)) else x"00000000";
43    eq <= x"00000001" when (signed(operand_a) = signed (↩
          operand_b)) else x"00000000";
44    neq <= x"00000000" when (signed(operand_a) /= signed ↩
          (operand_b)) else x"00000001";
45    gt <= x"00000001" when (signed(operand_a) > signed (↩
          operand_b)) else x"00000000";
46    lt_u <= x"00000001" when (unsigned(operand_a) < ↩
          unsigned (operand_b)) else x"00000000";
47    gt_u <= x"00000001" when (unsigned(operand_a) > ↩
          unsigned (operand_b)) else x"00000000";
48  end behav;
```

Listing 7.7: *ALU Controller*

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity alu_controller is
6    port (
7      ALUop_in: in std_logic_vector (2 downto 0);
8      funct7: in std_logic;
9      funct3: in std_logic_vector (2 downto 0);
10     ALUop_out: out std_logic_vector (4 downto 0)
11   );
12 end alu_controller;
13
14 architecture behav of alu_controller is
15 begin
16   process (ALUop_in, funct7, funct3)
17   begin
```

```
18      if ALUop_in = "011" then
19        ALUop_out <= '1'&x"f";
20      elsif ALUop_in = "010" then
21        ALUop_out <= "10"&funct3;
22      elsif ALUop_in = "000" and funct7 = '0' then
23        ALUop_out <= "00"&funct3;
24      elsif ALUop_in = "000" and funct7 = '1' then
25        ALUop_out <= "01"&funct3;
26      elsif ALUop_in = "000" then
27        ALUop_out <= "00"&funct3;
28      elsif ALUop_in = "001" and funct7 = '0' and funct3 ↩
            = "101" then
29        ALUop_out <= "00"&funct3;
30      elsif ALUop_in = "001" and funct7 = '1' and funct3 ↩
            = "101" then
31        ALUop_out <= "01"&funct3;
32      elsif ALUop_in = "001" and funct3 = "101" then
33        ALUop_out <= "00"&funct3;
34      elsif ALUop_in = "001" then
35        ALUop_out <= "00"&funct3;
36      else
37        ALUop_out <= "00000";
38      end if;
39    end process;
40  end behav;
```

Listing 7.8: *Control Unit*

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity control_unit is
5    port (
6      opcode:in std_logic_vector(6 downto 0);
7      mem_write, branch, reg_write, mem_to_reg, load_en: ↩
           out std_logic;
8      ALU_op:out std_logic_vector(2 downto 0);
9      operand_a_sel:out std_logic_vector (1 downto 0);
10     operand_b_sel:out std_logic;
11     extend_sel, next_pc_sel:out std_logic_vector (1 ↩
           downto 0)
12   );
13 end control_unit;
14
15 architecture behav of control_unit is
16   signal r_type, load_type, store_type, branch_type, ↩
       i_type, jalr_type, jal_type, lui_type: std_logic;
```

```
begin
  type_decode:
    r_type <= '1' when opcode = "0110011" else '0';
    load_type <= '1' when opcode = "0000011" else '0';
    store_type <= '1' when opcode = "0100011" else '0';
    branch_type <= '1' when opcode = "1100011" else '0';
    i_type <= '1' when opcode = "0010011" else '0';
    jalr_type <= '1' when opcode = "1100111" else '0';
    jal_type <= '1' when opcode = "1101111" else '0';
    lui_type <= '1' when opcode = "0110111" else '0';

  control_decode:
    mem_write <= store_type;
    branch <= branch_type;
    reg_write <= r_type or load_type or i_type or lui_type or jal_type or jalr_type;
    mem_to_reg <= load_type;
    ALU_op <= not (r_type or branch_type or i_type or jal_type or jalr_type) & not(r_type or load_type or store_type or i_type) & not(r_type or load_type or branch_type or lui_type);
    operand_a_sel <= "11" when lui_type = '1' else "10" when jalr_type = '1' else "10" when jal_type = '1' else "00";
    operand_b_sel <= lui_type or store_type or load_type or i_type;
    extend_sel <= store_type&lui_type;
    load_en <= load_type;
    next_pc_sel <= "01" when branch_type = '1' else "10" when jal_type = '1' else "11" when jalr_type = '1' else "00";
end behav;
```

Listing 7.9: *Top-Level*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top is
  port (
    clr_in: in std_logic;
    clk_in: in std_logic;
    read_en: in std_logic;
    seg: out std_logic_vector (7 downto 0);
    dig: out std_logic_vector (3 downto 0);
```

```
12      tx_out: out std_logic;
13      clk_speed_selector: in std_logic
14    );
15  end top;
16
17  architecture behav of top is
18    component program_counter is
19      port (
20        clr, clk: in std_logic;
21        address_in: in std_logic_vector(31 downto 0);
22        address_out: out std_logic_vector(31 downto 0)
23      );
24    end component program_counter;
25
26    component alu is
27    port (
28      ALUop_in: in std_logic_vector (4 downto 0);
29      operand_a: in std_logic_vector (31 downto 0);
30      operand_b: in std_logic_vector (31 downto 0);
31      result_out: out std_logic_vector (31 downto 0)
32    );
33    end component alu;
34
35    component control_unit is
36    port (
37      opcode:in std_logic_vector(6 downto 0);
38      mem_write, branch, reg_write, mem_to_reg, load_en: ↩
            out std_logic;
39      ALU_op:out std_logic_vector(2 downto 0);
40      operand_a_sel:out std_logic_vector (1 downto 0);
41      operand_b_sel:out std_logic;
42      extend_sel, next_pc_sel:out std_logic_vector (1 ↩
            downto 0)
43    );
44    end component control_unit;
45
46    component progmem IS
47    port
48    (
49      address: in std_logic_vector (29 downto 0);
50      clk: in std_logic  := '1';
51      instruction_out: out std_logic_vector (31 downto 0)
52    );
53    end component progmem;
54
55    component immediate_generator is
56    port (
```

```
57      inst: in std_logic_vector(31 downto 0);
58      s_type_out: out std_logic_vector(31 downto 0);
59      sb_type_out: out std_logic_vector(31 downto 0);
60      u_type_out: out std_logic_vector(31 downto 0);
61      uj_type_out: out std_logic_vector(31 downto 0);
62      i_type_out: out std_logic_vector(31 downto 0)
63    );
64    end component immediate_generator;
65
66    component register_files is
67    port (
68      write_en, clk: in std_logic;
69      read_address_1: in std_logic_vector(4 downto 0);
70      read_address_2: in std_logic_vector(4 downto 0);
71      write_address: in std_logic_vector(4 downto 0);
72      write_data: in std_logic_vector(31 downto 0);
73      data_out_1: out std_logic_vector(31 downto 0);
74      data_out_2: out std_logic_vector(31 downto 0)
75    );
76    end component register_files;
77
78    component alu_controller is
79    port (
80      ALUop_in: in std_logic_vector (2 downto 0);
81      funct7: in std_logic;
82      funct3: in std_logic_vector (2 downto 0);
83      ALUop_out: out std_logic_vector (4 downto 0)
84    );
85    end component alu_controller;
86
87    component data_memory IS
88    port (
89      clk: in std_logic;
90      data: in std_logic_vector (31 downto 0);
91      address: in std_logic_vector (7 downto 0);
92      write_en: in std_logic;
93      load_en: in std_logic;
94      q: out std_logic_vector (31 downto 0);
95      probe_out: out std_logic_vector(31 downto 0)
96    );
97    end component data_memory;
98
99    component sevenseg is
100   port (
101     clk: in std_logic;
102     data_in: in std_logic_vector (31 downto 0);
103     seg: out std_logic_vector (7 downto 0);
```

```
104     dig: out std_logic_vector (3 downto 0)
105   );
106   end component sevenseg;
107
108   component bin2bcd is
109    port (
110       input:      in   std_logic_vector (15 downto 0);
111       ones:       out  std_logic_vector (3 downto 0);
112       tens:       out  std_logic_vector (3 downto 0);
113       hundreds:   out  std_logic_vector (3 downto 0);
114       thousands:  out  std_logic_vector (3 downto 0)
115    );
116   end component bin2bcd;
117
118   component hex2ascii is
119   port(
120     input: in std_logic_vector (3 downto 0);
121     output: out std_logic_vector (7 downto 0);
122     next_out: in std_logic
123   );
124   end component hex2ascii;
125
126   component UART_TX is
127     generic (
128      g_CLKS_PER_BIT : integer := 217     -- Needs to be↩
           set correctly
129      );
130     port (
131      i_Clk       : in  std_logic;
132      i_TX_DV     : in  std_logic;
133      i_TX_Byte   : in  std_logic_vector(7 downto 0);
134      o_TX_Active : out std_logic;
135      o_TX_Serial : out std_logic;
136      o_TX_Done   : out std_logic
137      );
138   end component UART_TX;
139
140   component fifo is
141   Generic (
142     constant DATA_WIDTH  : positive := 32;
143     constant FIFO_DEPTH : positive := 256
144   );
145   Port (
146     CLK   : in  STD_LOGIC;
147     RST   : in  STD_LOGIC;
148     WriteEn : in  STD_LOGIC;
149     DataIn  : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 ↩
```

```
        downto 0);
    ReadEn   : in  STD_LOGIC;
    DataOut : out STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    Empty : out STD_LOGIC;
    Full  : out STD_LOGIC
  );
  end component fifo;

  component fifo32to4 is
    Port (
      clk   : in  std_logic;
      clr   : in  std_logic;
      datain_32: in  std_logic_vector (31 downto 0);
      NextWordReadEn  : in  std_logic;
      WriteEn: in std_logic;
      dataout : out std_logic_vector (3 downto 0);
      nextout: out std_logic;
      readnext: out std_logic;
      datavalid: out std_logic
    );
  end component fifo32to4;

  component validchecker is
  Port (
    clk   : in  std_logic;
    clr   : in  std_logic;
    datavalidin: in std_logic;
    txactive: in std_logic;
    validout: out std_logic;
    nextwordreaden_sig: out std_logic;
    tx_done_sig: in std_logic;
    tx_active_sig: in std_logic;
    read_next_sig: in std_logic;
    fifo_valid_sig: in std_logic;
    fifo32_4_writeen_sig: out std_logic;
    fifo_read_next: out std_logic;
    fifo_readen_sig: in std_logic
  );
  end component validchecker;


  component Debounce_Switch is
    port (
     i_Clk    : in  std_logic;
     i_Switch : in  std_logic;
     o_Switch : out std_logic
```

```
195     );
196 end component Debounce_Switch;
197 
198 
199 signal clr: std_logic := '1';
200 signal clk, clk2, clk3, clk4, clk_fifo: std_logic := ↩
        '0';
201 signal count, count2, count3, count4: integer := 1;
202 signal pc_in_sig: std_logic_vector (31 downto 0) := x↩
        "00000000"; --pc
203 signal pc_out_sig: std_logic_vector (31 downto 0) := ↩
        x"00000000"; --pc
204 signal instruction: std_logic_vector (31 downto 0); ↩
        --pc
205 signal s_type_sig, sb_type_sig, u_type_sig, ↩
        uj_type_sig, i_type_sig, immgen_mux_sig: ↩
        std_logic_vector (31 downto 0); --immgen
206 signal write_data_sig, data_out_1_sig, data_out_2_sig↩
        : std_logic_vector (31 downto 0); --regfiles
207 signal reg_write_en_sig, branch_sig, mem_write_sig, ↩
        mem_to_reg_sig, operand_b_sel_sig, load_en_sig: ↩
        std_logic; --ctrl unit
208 signal ALUop_sig: std_logic_vector (2 downto 0); --↩
        ctrl unit
209 signal operand_a_sel_sig, extend_sel_sig, ↩
        next_pc_sel_sig: std_logic_vector (1 downto 0); ↩
        --ctrl unit
210 signal ALUop_in_sig: std_logic_vector (4 downto 0); ↩
        --alu
211 signal ALU_branch_sig: std_logic; --alu
212 signal operand_a_sig, operand_b_sig, result_sig: ↩
        std_logic_vector (31 downto 0); --alu
213 signal data_memory_out_sig, probe_out_sig: ↩
        std_logic_vector (31 downto 0); --datamem
214 signal jump_sig, jalr_sig: std_logic_vector (31 ↩
        downto 0); -- jump target
215 signal bcd_sig: std_logic_vector (31 downto 0); -- ↩
        binary to bcd for 7-segments
216 signal ones_sig, tens_sig, hundreds_sig, ↩
        thousands_sig: std_logic_vector(3 downto 0); --↩
        binary to bcd for 7-segments
217 signal fifo_writeen_sig, fifo_readen_sig, ↩
        fifo_empty_sig, fifo_empty_sig_r, fifo_full_sig, ↩
        fifo_valid_sig, fifo_read_next: std_logic; --fifo
218 signal fifo_data_out_sig: std_logic_vector(31 downto ↩
        0); -- fifo
219 signal nextwordreaden_sig, fifo32_4_writeen_sig, ↩
```

```
        next_out_sig: std_logic; --fifo 32 to 4
  signal read_next_sig: std_logic; -- fifo 32 to 4
  signal fifo_4_out_sig: std_logic_vector (3 downto 0);
       --fifo 32 to 4
  signal asciiout_sig: std_logic_vector (7 downto 0);
      -- hex2ascii
  signal dv_clr, datavalid_sig: std_logic; -- datavalid
       checker
  signal datavalid_tx_sig, tx_dv_sig, tx_active_sig,
      tx_done_sig: std_logic; -- uart tx8
  signal read_en_debounced : std_logic; -- debounce

begin
  clr <= not clr_in;
  process (clk_in)
    begin
    if rising_edge(clk_in) then
      count <= count+1;
      if (count = 250000/1024) then
        clk <= not clk;
        count <= 1;
      end if;
    end if;
  end process;

  process (clk_in)
    begin
    if rising_edge(clk_in) then
      count2 <= count2+1;
      if (count2 = 2) then
        clk2 <= not clk2;
        count2 <= 1;
      end if;
    end if;
  end process;

  process (clk_in)
    begin
    if rising_edge(clk_in) then
      count3 <= count3+1;
      if (count3 = 1) then
        clk3 <= not clk3;
        count3 <= 1;
      end if;
    end if;
  end process;

```

```
262   process (clk_in)
263     begin
264     if rising_edge(clk_in) then
265       count4 <= count4+1;
266       if (count4 = 25000000/1024) then
267         clk4 <= not clk4;
268         count4 <= 1;
269       end if;
270     end if;
271   end process;
272
273   Debounce_Inst: Debounce_Switch port map (clk_in,↩
          read_en,read_en_debounced);
274   hex2ascii1: hex2ascii port map (fifo_4_out_sig, ↩
          asciiout_sig, next_out_sig);
275   fifo32to4_1: fifo32to4 port map (clk2, dv_clr, ↩
          fifo_data_out_sig, nextwordreaden_sig, ↩
          fifo32_4_writeen_sig, fifo_4_out_sig, ↩
          next_out_sig, read_next_sig, datavalid_sig);
276   fifo1: fifo port map (clk_fifo, clr, fifo_writeen_sig↩
          , data_out_2_sig, fifo_readen_sig, ↩
          fifo_data_out_sig, fifo_empty_sig, fifo_full_sig)↩
          ;
277   tx1: uart_tx port map (clk3, tx_dv_sig, asciiout_sig,↩
           tx_active_sig, tx_out, tx_done_sig);
278   validchecker1: validchecker port map (clk3, dv_clr, ↩
          datavalid_sig, tx_active_sig, datavalid_tx_sig, ↩
          nextwordreaden_sig, tx_done_sig, tx_active_sig, ↩
          read_next_sig, fifo_valid_sig, ↩
          fifo32_4_writeen_sig, fifo_read_next, ↩
          fifo_readen_sig);
279   sevenseg1: sevenseg port map (clk_in, bcd_sig, seg, ↩
          dig);
280   pc1: program_counter port map (clr, clk, pc_in_sig, ↩
          pc_out_sig);
281   inst1: progmem port map (pc_out_sig(31 downto 2), clk↩
          , instruction);
282   immgen1: immediate_generator port map (instruction, ↩
          s_type_sig, sb_type_sig, u_type_sig, uj_type_sig,↩
           i_type_sig);
283   regfiles1: register_files port map (reg_write_en_sig,↩
           clk, instruction(19 downto 15), instruction(24 ↩
          downto 20), instruction(11 downto 7), ↩
          write_data_sig, data_out_1_sig, data_out_2_sig);
284   aluctrl1: alu_controller port map (ALUop_sig, ↩
          instruction(30), instruction(14 downto 12), ↩
          ALUop_in_sig);
```

```
285   alu1: alu port map (ALUop_in_sig, operand_a_sig, ↩
          operand_b_sig, result_sig);
286   ctrl1: control_unit port map (instruction(6 downto 0)↩
          , mem_write_sig, branch_sig, reg_write_en_sig, ↩
          mem_to_reg_sig, load_en_sig, ALUop_sig, ↩
          operand_a_sel_sig, operand_b_sel_sig, ↩
          extend_sel_sig, next_pc_sel_sig);
287   datamem1: data_memory port map (clk, data_out_2_sig, ↩
          result_sig(9 downto 2), mem_write_sig, ↩
          load_en_sig, data_memory_out_sig, probe_out_sig);
288   bin2bcd1: bin2bcd port map (probe_out_sig(15 downto ↩
          0), ones_sig, tens_sig, hundreds_sig, ↩
          thousands_sig);
289
290   bcd_sig <= x"0000"&thousands_sig&hundreds_sig&↩
          tens_sig&ones_sig;
291
292   fifo_writeen_sig <= '1' when (mem_write_sig and (not ↩
          fifo_full_sig or fifo_empty_sig)) = '1' and ↩
          result_sig(9 downto 2) = x"0d" else '0';
293
294   clk_fifo <= clk4 when clk_speed_selector = '1' else ↩
          clk;
295
296   fiforeaden:process(clk)
297   begin
298     if rising_edge (clk) then
299       fifo_readen_sig <= fifo_read_next and ↩
              read_en_debounced and (not fifo_empty_sig_r);
300     end if;
301   end process;
302   tx_dv_sig <= datavalid_tx_sig and read_en_debounced
303   ;
304   dv_clr <= clr or not read_en_debounced;
305
306   fifoemptyregister:process(clk)
307   begin
308     if rising_edge (clk) then
309       fifo_empty_sig_r <= fifo_empty_sig;
310     end if;
311   end process;
312
313
314   fifo_valid_sig <= read_en_debounced and ↩
          fifo_readen_sig;
315
316
```

```
317   pc_mux:
318     with next_pc_sel_sig select pc_in_sig <=
319       std_logic_vector(unsigned(pc_out_sig) + ↩
              to_unsigned(4, 32)) when "00",
320       jump_sig when "01",
321       std_logic_vector(unsigned(uj_type_sig) + unsigned↩
              (pc_out_sig)) when "10",
322       jalr_sig when "11",
323       (others => 'X') when others;
324
325   alu_operand_a_mux:
326     with operand_a_sel_sig select operand_a_sig <=
327       data_out_1_sig when "00",
328       pc_out_sig when "01",
329       std_logic_vector(unsigned(pc_out_sig) + ↩
              to_unsigned(4, 32)) when "10",
330       data_out_1_sig when "11",
331       (others => 'X') when others;
332
333   alu_operand_b_mux:
334     with operand_b_sel_sig select operand_b_sig <=
335       data_out_2_sig when '0',
336       immgen_mux_sig when '1',
337       (others => 'X') when others;
338
339   immgen_mux:
340     with extend_sel_sig select immgen_mux_sig <=
341       i_type_sig when "00",
342       u_type_sig when "01",
343       s_type_sig when "10",
344       (others => 'X') when others;
345
346   mem_to_reg_mux:
347     with mem_to_reg_sig select write_data_sig <=
348       result_sig when '0',
349       data_memory_out_sig when '1',
350       (others => 'X') when others;
351
352   jalr_target:
353     jalr_sig <= std_logic_vector(unsigned(↩
            data_out_1_sig) + unsigned(immgen_mux_sig)) and↩
             x"fffffffc";
354
355   jump_target:
356     ALU_branch_sig <= '1' when (ALUop_in_sig(4 downto ↩
            3) = "10") and (result_sig = x"00000001") else ↩
            '0';
```

```
357     with ALU_branch_sig and branch_sig select jump_sig ↩
            <=
358       std_logic_vector(unsigned(pc_out_sig) + ↩
              to_unsigned(4, 32)) when '0',
359       std_logic_vector(unsigned(sb_type_sig) + unsigned↩
              (pc_out_sig)) when '1',
360       (others => 'X') when others;
361 end behav;
```