# BAB 4

# IMPLEMENTASI DAN EVALUASI

Dalam bab ini akan dijelaskan implementasi atau pelaksanaan dari perancangan yang telah dijelaskan pada Bab 3. Hasil implementasi tersebut selanjutnya dievaluasi untuk membuktikan bahwa prosesor RISC yang telah dibangun dapat bekerja sesuai dengan batasan yang ditetapkan.

## 4.1 Peralatan yang Digunakan

Untuk tujuan implementasi dari rancangan prosesor dan evaluasi dari hasilnya, diperlukan sistem perangkat lunak dan perangkat keras sebagai berikut:

### 4.1.1 Sistem Perangkat Lunak

- *Software* Xilinx ISE WebPack seri 5.2i

  Perangkat lunak ini digunakan untuk melakukan perancangan, sintesis, implementasi, evaluasi, dan mengkonfigurasi FPGA (*file* `.bit`) maupun CPLD (*file* `.jed`) yang digunakan.

- *Software* Xilinx Foundation 4.1i

  Perangkat lunak ini digunakan untuk melakukan simulasi dari perancangan prosesor yang telah dibangun dengan tujuan untuk mengetahui apakah hasil yang didapatkan telah sesuai dengan perancangannya.

- *Software assembler* sederhana

  *Software assembler* ini dirancang dengan tujuan untuk mempermudah pengkodean terhadap program pengujian. *Software assembler* ini akan mendekode program dalam bahasa *assembly* (*file* `.asm`) menjadi kumpulan 32 bit instruksi (*file* `.bit`) yang kemudian diletakkan pada memori instruksi (ROM).

### 4.1.2 Sistem Perangkat Keras

- PC Pentium III 600 MHz dengan memori sebesar 256 MB dan sistem operasi Windows XP Professional.
- PC Pentium 4 1,7 GHz dengan memori sebesar 256 MB dan sistem operasi Windows 98 SE.
- FPGA yang digunakan adalah modul D2 (Digilab 2) dari Digilent Inc. dengan tipe Xilinx Spartan 2 XC2S200-PQ208.
- Menggunakan modul DIO2 (Digilab Digital I/O 2) dari Digilent Inc. dengan tipe Xilinx CPLD CX95108-PC84.
- Menggunakan tombol, saklar, dan LED pada modul DIO2 sebagai *input* dan *output*.
- Menggunakan rangkaian penyangga JTAG *Parallel Download Cable* untuk mengkonfigurasi modul DIO2.
- Kabel paralel (DB-25) *male-male* untuk menghubungkan PC dengan modul D2 atau PC dengan rangkaian penyangga.
- Osiloskop Digital untuk melakukan pengukuran sinyal keluaran.

- Menggunakan adaptor yang mengubah tegangan AC 220 V menjadi DC +5 V sebagai tegangan untuk modul D2 dan D2IO.

## 4.2 Implementasi

Implementasi dilakukan menggunakan PC Pentium III 600 MHz, memori sebesar 256 MB, sistem operasi Windows XP Professional, serta *software* Xlinx ISE WebPack seri 5.2i.

### 4.2.1 Persiapan Implementasi

Langkah-langkah melakukan instalasi *software* Xilinx ISE WebPack seri 5.2i:

1. *Download software* Xilinx ISE WebPack dari situs resmi Xilinx (www.xilinx.com) dan dapatkan seri terbarunya.
2. *Double click file* `WebPACK_52_fcfull_i.exe` atau *icon* seperti Gambar 4.1 untuk instalasi seri 5.2i

.

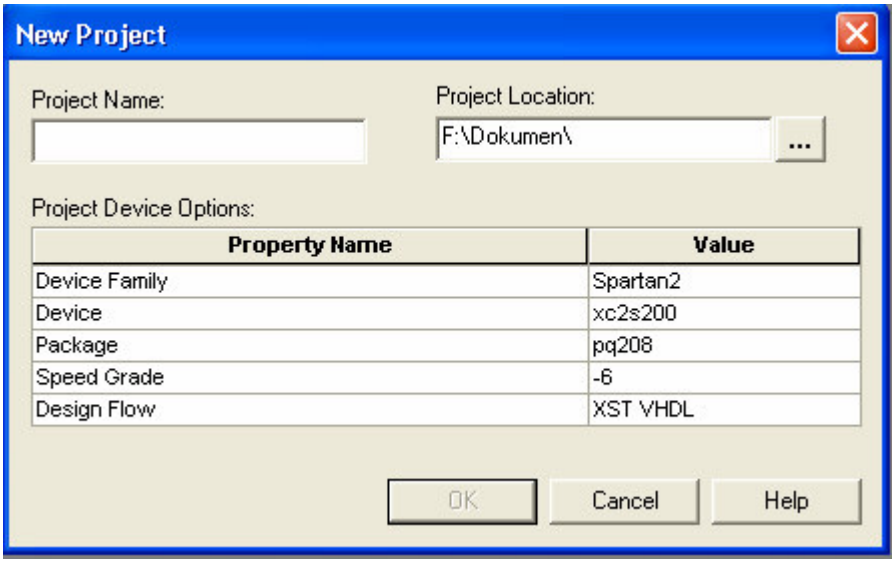

**Gambar 4.1** *Icon* Instalasi Xilinx ISE WebPack seri 5.2i

3. Tunggu hingga data selesai di *extract*.
4. Baca lisensi dan berikan tanda benar pada "*I accept the terms of this software lecense*". Dan *click* "*Next*".
5. Isikan nama *directory* dan nama *folder*-nya, kemudian *click* "*Next*".

6. Berikan tanda benar pada "*Set/Update* XILINX *variable*" dan "*Set/Update* PATH *variable*". Kemudian *click* "*Next*".
7. *Click* "*Install*".

Berikut adalah langkah-langkah membuat proyek dan *file* sumber baru:

1. Jalankan "*Project Navigator*" dengan nama *file* `ise.exe`.
2. Buat proyek baru melalui *menu bar* "*File*" kemudian pilih "*New Project...*".
3. Isi nama proyek pada "*Project Name*" dan lokasi pada "*Project Location*". Isikan informasi lainnya seperti pada Gambar 4.2. Kemudian *click* "*OK*".




**Gambar 4.2** *New Project*

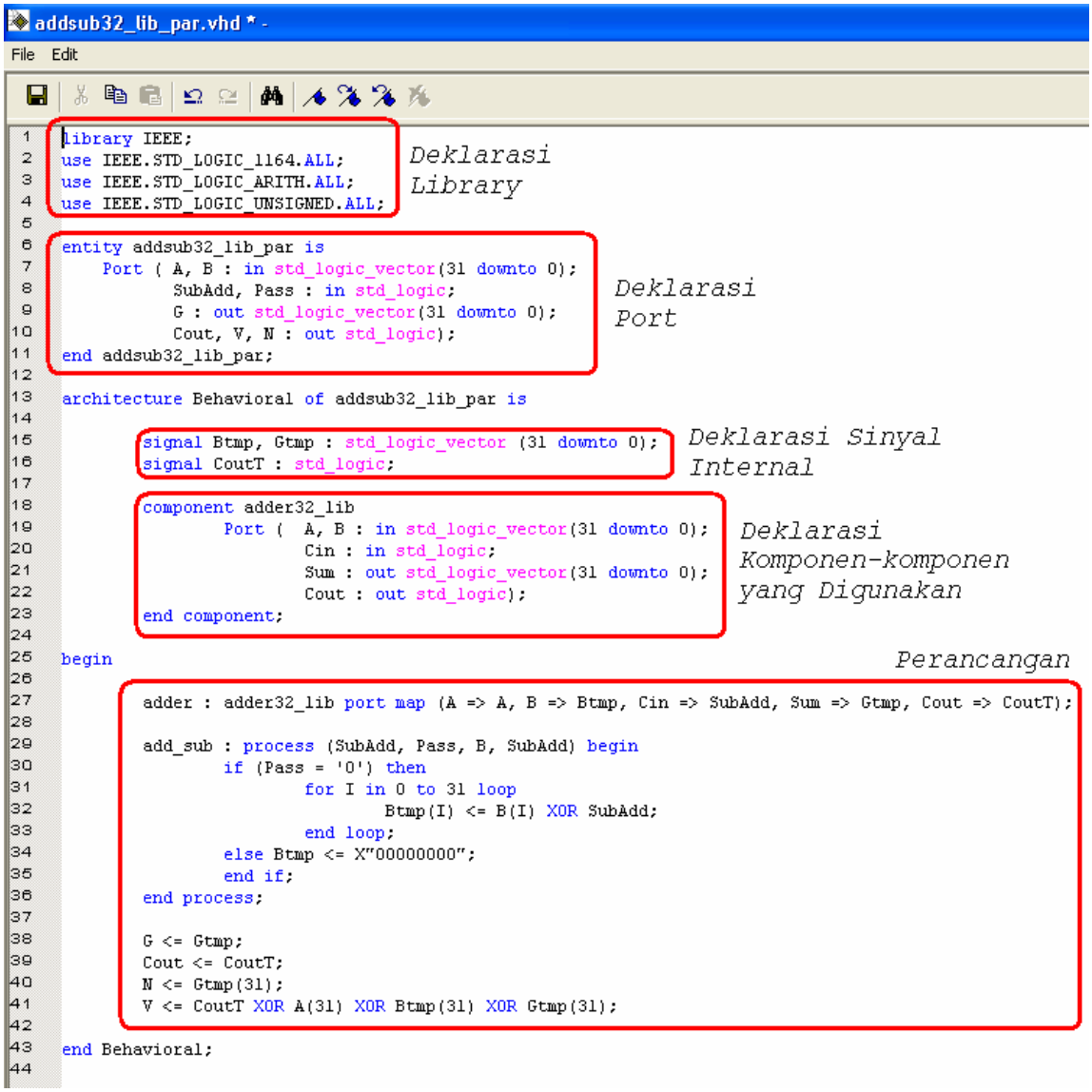
4. Masukkan *file* sumber baru dengan cara *click menu bar* "*Project*" kemudian pilih "*New Source*…".
5. Karena dalam keseluruhan perancangan digunakan VHDL maka pilih "VHDL *module*" serta berikan nama *file* dan lokasinya.
6. Berikan tanda benar pada "*Add to project*" kemudian *click* "*Next*".
7. Isikan nama *Entity* dan *Architecture*. Isi kolom *port name*, *direction*, MSB dan LSB dengan port-port yang digunakan. Kemudian *click* "*Next*".
8. *Click* "*Finish*".

### 4.2.2 Pelaksanaan Implementasi

*Double click file source* yang baru dibuat pada jendela "*Sources in Project*". Jendela *editor* akan ditampilkan seperti Gambar 4.3. Pada saat *file source* dibuat, beberapa *library* telah diberikan pada bagian "Deklarasi *Library*". *Library-library* yang disediakan sudah cukup untuk melakukan seluruh rancangan.

Gambar 4.3 menunjukkan komponen penambahan dan pengurangan 32 *bit* dari *file* `addsub32_lib_par.vhd`. Bagian "Deklarasi *Port*" merupakan tempat untuk mendeklarasi *pin-pin* masukan dan/atau keluaran dari sebuah komponen. Bagian "Deklarasi Sinyal *Internal*" berfungsi untuk mendeklarasikan beberapa label untuk sinyal yang akan digunakan dalam perancangan. Jika ingin menggabungkan komponen-komponen lain dalam perancangan maka komponen-komponen tersebut dideklarasikan pada bagian "Deklarasi Komponen-komponen yang Digunakan". Program VHDL yang diletakkan pada bagian lampiran menyertakan hirarki program tersebut yang

menunjukkan komponen-komponen yang digunakan dalam perancangannya. Bagian "Perancangan" merupakan tempat melakukan perancangan baik dengan metoda *structural*, *data flow*, maupun *behavioral*.




**Gambar 4.3** Jendela *Editor*

Jika perancangan komponen telah selesai dilakukan maka *double click* "*Synthesize*" pada jendela "*Processes for Current Source*" untuk

mendapatkan laporan mengenai kesalahan (*error*) atau peringatan (*warning*). Jika sudah tidak terjadi kesalahan dan sintesis berhasil diselesaikan maka akan didapatkan laporan mengenai jumlah *slice* (*flip-flop* dan LUT), IOB dan *three state buffer* yang digunakan serta laporan mengenai TPD yang dihasilkan. Hasil laporan sintesis akan digunakan untuk memilih komponen berdasarkan CLB dan/atau TPD terkecil.

Laporan yang diberikan pada saat sintesis merupakan kondisi komponen tanpa membatasinya dengan jenis peralatan (FPGA) yang digunakan melainkan hanya membandingkannya dengan komponen maksimum dari peralatan tersebut. Laporan hasil sintesis dapat dilihat pada *file* `.syr`. Persentase yang diberikan merupakan perbandingan jumlah komponen yang digunakan dengan jumlah komponen keseluruhan, lihat Tabel 4.1 untuk jumlah keseluruhan komponen.

**Tabel 4.1** Jumlah Keseluruhan Komponen pada Xilinx Spartan 2 XC2S200-PQ208 untuk Proses Sintesis

| Komponen | Synthesize |
|---|---|
| CLB | 1176 |
| Slice | 2352 |
| Flip-flop | 4704 |
| 4 input LUT | 4704 |
| IOB | 144 |
| TBUF | 2352 |
| GCLK | 4 |

Untuk tabel hasil laporan sintesis (seperti Tabel 4.2) dimana kolom *slice* menunjukkan jumlah *slice* yang digunakan. Dalam sebuah *slice* terdapat dua buah *flip-flop* dan dua buah LUT empat *input*. Jumlah *flip-flop* dan LUT

empat *input* yang digunakan ditunjukkan pada kolom FF dan 4 *input* LUT. Kolom IOB menunjukkan jumlah IOB (*pin* masukkan dan/atau keluaran) yang digunakan. Kolom T-BUF menunjukkan jumlah *three state buffer* yang digunakan.

Kolom *Timing Report* memberikan laporan mengenai TPD yang dihasilkan dari masing-masing komponen. Kolom Prd. menunjukkan perioda minimum yang dibutuhkan oleh sebuah komponen dan berhubungan dengan frekuensi maksimum yang ditunjukkan pada kolom Frek.. IAT (*Input Arrival Time*) merupakan waktu minimum yang dibutuhkan oleh nilai masukan sebelum terjadinya *clock*, sedangkan ORT (*Ouput Required Time*) merupakan maksimal waktu yang dibutuhkan untuk menghasilkan *output* setelah terjadinya *clock*. CPD (*Combinational Path Delay*) merupakan waktu tunda pada komponen kombinasional.

CPD dihasilkan dari komponen yang mengandung rangkaian kombinasional, IAT dan ORT dihasilkan dari komponen yang mengandung rangkaian sekuensial, sedangkan Prd. dan Freq. dihasilkan dari komponen yang menggunakan *clock*. Satuan untuk Prd., IAT, ORT, dan CPD adalah nano detik (ns), sedangkan satuan untuk Freq. adalah mega hertz (MHz).

Perancangan berdasarkan fungsi memungkinkan komponen dirancang secara terpisah kemudian baru disatukan. Berikut adalah implementasi dari masing-masing komponen.

#### 4.2.2.1 Komponen-komponen *Register File*

Ada dua jenis perancangan untuk komponen yang membangun *register file*, yaitu perancangan *register file* menggunakan *flip-flop* (`Register_File_FF.vhd`) dan DPRAM (`Register_File_DPRAM.vhd`). Pada perancangan menggunakan *flip-flop*, komponen-komponen yang dibutuhkan yaitu *register* 32 *bit* dengan *reset*, *enable*, dan *aktif low* (`reg1x32RE_1.vhd`), *decoder* 5 *to* 32 (`dec5_32.vhd`), *load* 32 *bit* (`load32.vhd`) untuk menentukan satu dari 32 *register* yang akan ditulis, dua buah *multipexer* 32 *input* 32 *bit* (`mux32x32.vhd`) untuk *bus* A dan *bus* B. Sedangkan komponen yang dibutuhkan untuk *register file* dengan DPRAM adalah 16 buah DPRAM 1 *bit aktif low* (`ram16x1D_1.vhd`) yang merupakan *library* dari Xilinx. Tabel 4.2 menunjukkan data hasil sintesis untuk perbandingan kedua *register file* beserta komponennya.

**Tabel 4.2** Perbandingan *Register File Flip-flop* dengan DPRAM

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| Register File Flip-flop | | | | | | | | | | |
| dec5_32 | 32 (1%) | --- | 32 (0%) | 37 (25%) | --- | --- | --- | --- | --- | 10.763 |
| load32 | 53 (2%) | --- | 92 (1%) | 38 (26%) | --- | --- | --- | --- | --- | 13.886 |
| reg1x32RE_1 | 18 (0%) | --- | 32 (0%) | 66 (45%) | --- | --- | --- | 5.082 | 6.788 | --- |
| register_file_ff | 1088 (46%) | 992 (21%) | 1145 (24%) | 113 (78%) | --- | --- | --- | 12.444 | 9.832 | 20.611 |
| Register File DPRAM | | | | | | | | | | |
| register_file_dpram | 230 (9%) | --- | 198 (4%) | 112 (77%) | --- | --- | --- | 3.004 | 9.315 | 17.486 |

Dari tabel 4.2 dapat dilihat bahwa *register file* dengan DPRAM dapat menghemat 858 *slice* atau sekitar 37% dari total CLB dibandingkan dengan *register file* menggunakan *flip-flop*. Dari hasil laporan mengenai waktu, TPD yang dibutuhkan oleh *register file* juga lebih kecil dibandingkan dengan *flip-flop*, oleh karena itu akan digunakan komponen *register file* dengan DPRAM pada perancangan selanjutnya.

Untuk menyederhanakan perancangan maka *register file* (`Register_File_DPRAM.vhd`), MUX A, MUX B, dan *constant unit* (`const_unit.vhd`) disatukan menjadi komponen yang disebut *register file* gabungan (`register_file_gab.vhd`). Beberapa perubahan akan terjadi pada MUX A dan MUX B jika digunakan *data forwarding* untuk menghindari *data depedency* (`register_file_gab_df.vhd`). Tabel 4.3 berikut menunjukkan laporan yang diperoleh dari hasil sintesis.

**Tabel 4.3** *Register File* Gabungan

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| const_unit | 1 (0%) | --- | 1 (0%) | 49 (34%) | --- | --- | --- | --- | --- | 9. 719 |
| register_file _gab | 265 (11%) | --- | 262 (5%) | 163 (113%) | --- | --- | --- | 3. 004 | 10. 899 | 19. 070 |
| register_file _gab_df | 294 (12%) | --- | 313 (6%) | 197 (136%) | --- | --- | --- | 3. 004 | 10. 899 | 20. 654 |

#### 4.2.2.2 Komponen-komponen *Function Unit*

Salah satu komponen yang membangun *function unit* untuk operasi aritmatika adalah komponen penjumlah (*adder*). Beberapa jenis *adder* untuk dijadikan perbandingan adalah *ripple carry adder*

(`adder32_rc.vhd`), *carry lookahead adder* (`adder32_cl.vhd`), *carry selector adder* (`adder32_cs.vhd`), dan *adder* menggunakan *library* (`adder32_lib.vhd`). Tabel 4.4 menunjukkan perbandingan keempat *adder*.

Dari tabel 4.4 dapat dilihat bahwa komponen *adder* dengan *library* menggunakan *slice* terkecil dan TPD terpendek.

**Tabel 4.4** Perbandingan Empat Buah *Adder*

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| adder32_lib | 16 (0%) | --- | 32 (0%) | 98 (68%) | --- | --- | --- | --- | --- | 10.402 |
| adder32_cl | 58 (2%) | --- | 101 (2%) | 98 (68%) | --- | --- | --- | --- | --- | 60.857 |
| adder32_cs | 63 (2%) | --- | 109 (2%) | 98 (68%) | --- | --- | --- | --- | --- | 41.642 |
| adder32_rc | 42 (1%) | --- | 73 (1%) | 98 (68%) | --- | --- | --- | --- | --- | 63.116 |

Komponen berikutnya adalah gabungan komponen *adder* dengan *subtractor*, dimana digunakan tambahan metoda *two's complement* pada keempat komponen *adder* (`addsub32_rc.vhd`, `addsub32_cs.vhd`, `addsub32_cl.vhd`, dan `addsub32_lib_par.vhd`). Selain itu digunakan juga komponen dimana *adder* yang *substractor*-nya tidak menggunakan *metoda two's complement* melainkan menggunakan *library* (`addsub32_lib_all.vhd`). Tabel 4.5 menunjukkan perbandingan kelima komponen *adder* dan *subtractor*.

Dari Tabel 4.5 dapat dilihat bahwa komponen *adder* dengan *library* dan *subtractor* dengan metoda *two's complement* menggunakan *slice* terkecil dan TPD terpendek.

**Tabel 4.5** Perbandingan Lima Buah Komponen *Adder* dan *Subtractor*

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| addsub32_lib_all | 54 (2%) | --- | 100 (2%) | 101 (70%) | --- | --- | --- | --- | --- | 17.782 |
| addsub32_lib_par | 17 (0%) | --- | 34 (0%) | 101 (70%) | --- | --- | --- | --- | --- | 16.342 |
| addsub32_cl | 81 (3%) | --- | 140 (2%) | 101 (70%) | --- | --- | --- | --- | --- | 67.985 |
| addsub32_cs | 82 (3%) | --- | 143 (2%) | 101 (70%) | --- | --- | --- | --- | --- | 45.242 |
| addsub32_rc | 61 (2%) | --- | 106 (2%) | 101 (70%) | --- | --- | --- | --- | --- | 67.706 |

Untuk menghasilkan *flag Zero* dari proses aritmatika digunakan komponen *zero detector* (`zero_detector.vhd`). Selain itu terdapat juga komponen *logic unit* (`logicunit32.vhd`) yang berfungsi membangun *function unit* untuk operasi logika. Penggabungan komponen untuk operasi aritmatika dan logika disebut *arithmetic logic unit* (`ALU.vhd`). Untuk instruksi *load upper immediate* digunakan komponen LUI (`LUI.vhd`). Tabel 4.6 menunjukkan penggunaan *slice* dan lamanya TPD untuk komponen *zero detector*, *logic unit*, ALU, dan LUI.

Komponen untuk operasi pergeseran dipilih dari tiga jenis pergeseran, yaitu menggunakan *barrel shifter* dengan 1 *bit selector* (`barrelshift32_1.vhd`), *barrel shifter* dengan 2 *bit selector*

(`barrelshift32_2.vhd`), dan dengan *shifter* biasa (`shifter_32.vhd`). Tabel 4.7 menunjukkan perbandingan ketiga *shifter*.

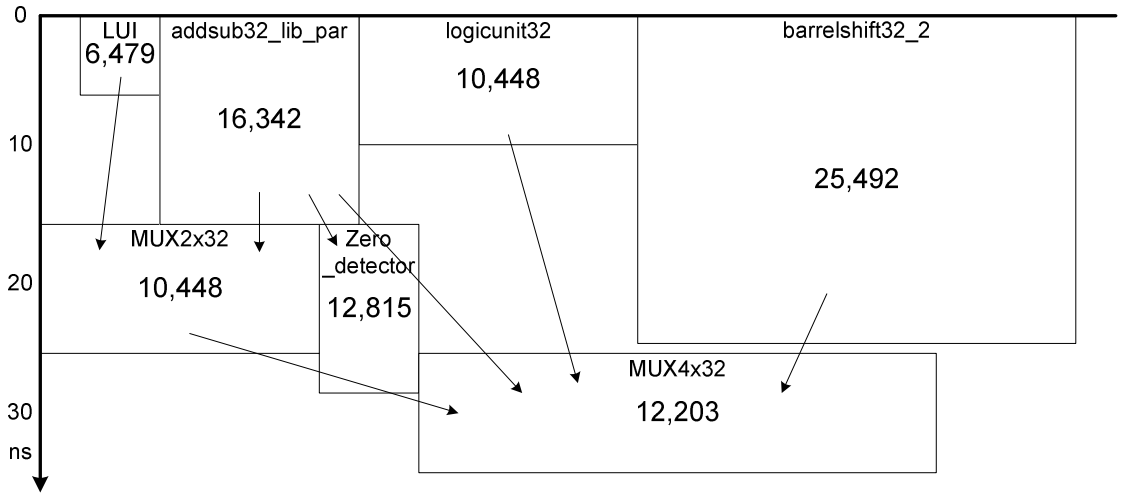
**Tabel 4.6** Penggunaan *Slice* dan Lamanya TPD untuk Komponen *Zero Detector*, *Logic Unit*, ALU, dan LUI

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| zero_ detector | 6 (0%) | --- | 11 (0%) | 33 (22%) | --- | --- | --- | --- | --- | 12. 815 |
| logicunit32 | 18 (0%) | --- | 32 (0%) | 98 (68%) | --- | --- | --- | --- | --- | 10. 448 |
| alu | 42 (1%) | --- | 78 (1%) | 135 (93%) | --- | --- | --- | --- | --- | 19. 816 |
| lui | 0 (0%) | --- | 0 (0%) | 48 (33%) | --- | --- | --- | --- | --- | 6. 479 |

**Tabel 4.7** Perbandingan Tiga Buah *Shifter*

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| barrelshift 32_1 | 129 (5%) | --- | 225 (4%) | 71 (49%) | --- | --- | --- | --- | --- | 27. 67 |
| barrelshift 32_2 | 126 (5%) | --- | 238 (5%) | 71 (49%) | --- | --- | --- | --- | --- | 25. 492 |
| shifter_32 | 261 (11%) | --- | 496 (9%) | 71 (49%) | --- | --- | --- | --- | --- | 25. 478 |

Setelah semua komponen yang membangun *function unit* dibuat dan dipilih maka tugas selanjutnya adalah menyusunnya agar didapatkan *delay path* terkecil untuk komponen *function unit* (`function_unit.vhd`). Gambar 4.4 menunjukkan hasil penyusunan yang dilakukan. Satuan untuk nilai-nilai pada Gambar 4.4 adalah nano detik (ns). Tabel 4.8 menunjukkan hasil laporan sintesis untuk *function unit*.




**Gambar 4.4** Penyusunan Komponen pada *Function Unit* untuk Mendapatkan *Delay Path* Terkecil

**Tabel 4.8** Laporan Sintesis untuk Function Unit

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| function_ unit | 200 (8%) | --- | 380 (8%) | 104 (72%) | --- | --- | --- | --- | --- | 29. 077 |

#### 4.2.2.3 Komponen-komponen *Control Unit*

Beberapa komponen pada *control unit* dirancang terpisah dengan komponen RISC, sisnya dirancang di dalam komponen RISC. Komponen yang dirancang di luar komponen RISC yaitu komponen *instruction decoder* (`id.vhd`), *branch control* (`branch_ctrl.vhd`), *data forwarding* (`data_forwarding.vhd`), dan *interrupt contorl* (`interrupt_ctrl.vhd`). Tabel 4.9 menunjukkan hasil laporan sintesis untuk komponen yang dirancang di luar komponen RISC.

**Tabel 4.9** Laporan Sintesis untuk Komponen yang Dirancang di Luar Komponen RISC

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| id | 27 (1%) | --- | 25 (0%) | 51 (35%) | --- | --- | --- | --- | --- | 10.764 |
| branch_ctrl | 3 (0%) | --- | 5 (0%) | 9 (6%) | --- | --- | --- | --- | --- | 10.367 |
| data_forwarding | 5 (0%) | --- | 10 (0%) | 20 (13%) | --- | --- | --- | --- | --- | 11.699 |
| Interrupt_ctrl | 4 (0%) | 1 (0%) | 7 (0%) | 16 (11%) | --- | --- | --- | 7.620 | 8.372 | 11.402 |

#### 4.2.2.4 Komponen-komponen Prosesor RISC

Komponen-komponen untuk prosesor RISC yaitu gabungan komponen-komponen dari *register file*, *function unit*, *control unit*, dan komponen *register* (`reg1x1.vhd`, `reg1x2.vhd`, `reg1x3.vhd`, `reg1x4.vhd`, `reg1x5.vhd`, `reg1x26.vhd`, dan `reg1x32.vhd`) untuk menampung data pada proses *pipeling* serta komponen penyangga (buffer) untuk *bus* data (`bufe32.vhd`) dan *bus* alamat (`bufe3.vhd`). Tabel 4.10 menunjukkan hasil laporan sintesis untuk komponen *register*, *buffer*, dan prosesor RISC (`risc.vhd`).

**Tabel 4.10** Laporan Sintesis untuk Komponen *Register*, *Buffer*, dan Prosesor RISC

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| Register untuk Pipeline | | | | | | | | | | |
| reg1x1 | 1 (0%) | 1 (0%) | --- | 3 (2%) | --- | --- | --- | 2.520 | 6.788 | --- |
| reg1x2 | 1 (0%) | 2 (0%) | --- | 5 (3%) | --- | --- | --- | 2.520 | 6.788 | --- |
| reg1x3 | 2 (0%) | 3 (0%) | --- | 7 (4%) | --- | --- | --- | 2.520 | 6.788 | --- |
| reg1x4 | 2 (0%) | 2 (0%) | --- | 9 (6%) | --- | --- | --- | 2.520 | 6.788 | --- |
| reg1x5 | 3 (0%) | 5 (0%) | --- | 11 (7%) | --- | --- | --- | 2.520 | 6.788 | --- |

**Tabel 4.10** Laporan Sintesis untuk Komponen *Register*, *Buffer*, dan Prosesor RISC (lanjutan)

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. | IAT | ORT | CPD |
| reg1x26 | 15 (0%) | 26 (0%) | --- | 53 (36%) | --- | --- | --- | 2. 520 | 6. 788 | --- |
| reg1x32 | 18 (0%) | 32 (0%) | --- | 65 (45%) | --- | --- | --- | 2. 520 | 6. 788 | --- |
| Buffer | | | | | | | | | | |
| bufe32 | --- | --- | --- | 65 (45%) | --- | --- | --- | --- | --- | 10. 587 |
| bufe3 | --- | --- | --- | 7 (4%) | --- | --- | --- | --- | --- | 8. 499 |
| Prosesor RISC | | | | | | | | | | |
| risc | 770 (32%) | 315 (6%) | 1200 (25%) | 140 (97%) | --- | 34. 678 | 28. 942 | 13. 707 | 27. 690 | 17. 027 |

Setelah semua komponen disatukan menjadi prosesor RISC, maka lakukan implementasi dengan *double click menu* "*Implement Design*" pada jendela "*Processes for Current Sources*". Jika tidak terjadi kesalahan, maka proses implementasi menghasilkan laporan seperti terlihat pada Tabel 4.11. Untuk komponen maksimum dari peralatan yang digunakan dapat dilihat pada Tabel 4.12. Laporan yang diberikan pada proses implementasi merupakan kondisi komponen yang dibatasi dengan jenis peralatan (FPGA) yang digunakan. Laporan hasil implementasi mengenai jumlah *slice*, LUT, dan IOB yang digunakan dapat dilihat pada *file* `.mrp`. Sedangkan laporan hasil implementasi mengenai lamanya TPD dapat dilihat pada *file* `.twr` untuk format berupa text atau *click* "*Analyze Post-Place & Route Static Timing* (*Timing Analyzer*)" untuk melakukan analisa secara interaktif dengan masukkan berupa posisi sumber dan posisi tujuan. Informasi mengenai analisa TPD secara interaktif disimpan pada file `.twx`.

**Tabel 4.11** Laporan Implementasi untuk Prosesor RISC

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | |
|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. |
| risc | 719 (30%) | 314 (6%) | 1332 (28%) | 140 (100%) | --- | 40.354 | 24.781 |

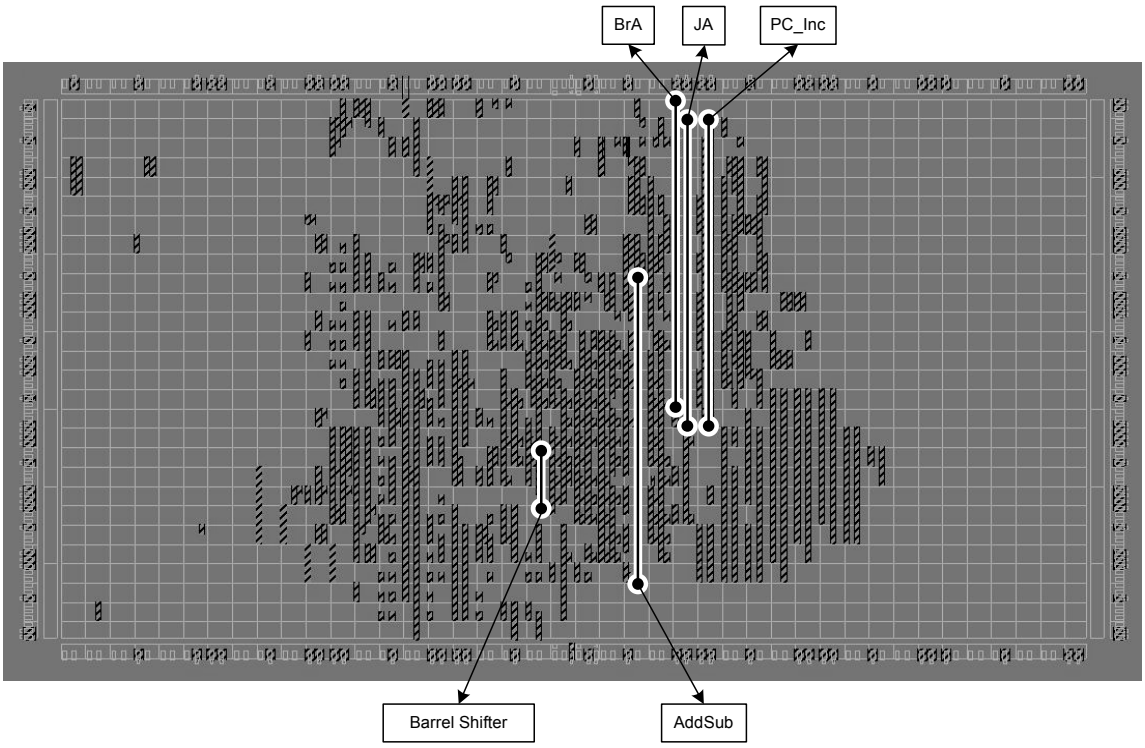
**Tabel 4.12** Jumlah Keseluruhan Komponen pada Xilinx Spartan 2 XC2S200-PQ208 untuk Proses Implementasi

| Komponen | Implementation |
|---|---|
| CLB | 1176 |
| Slice | 2352 |
| Flip-flop | 4704 |
| 4 input LUT | 4704 |
| IOB | 140 |
| TBUF | 2464 |
| GCLK | 4 |
| GCLKIOB | 4 |

Gambar 4.5 menunjukkan penempatan dan hubungan *slice* yang digunakan, dimana ditunjukkan juga *carry logic* pada operasi penjumlahan yang digunakan dalam komponen *barrel shifter*, *function unit* (AddSub), *branch* (BrA), *jump* (JA), dan PC *Increase* (PC_Inc).

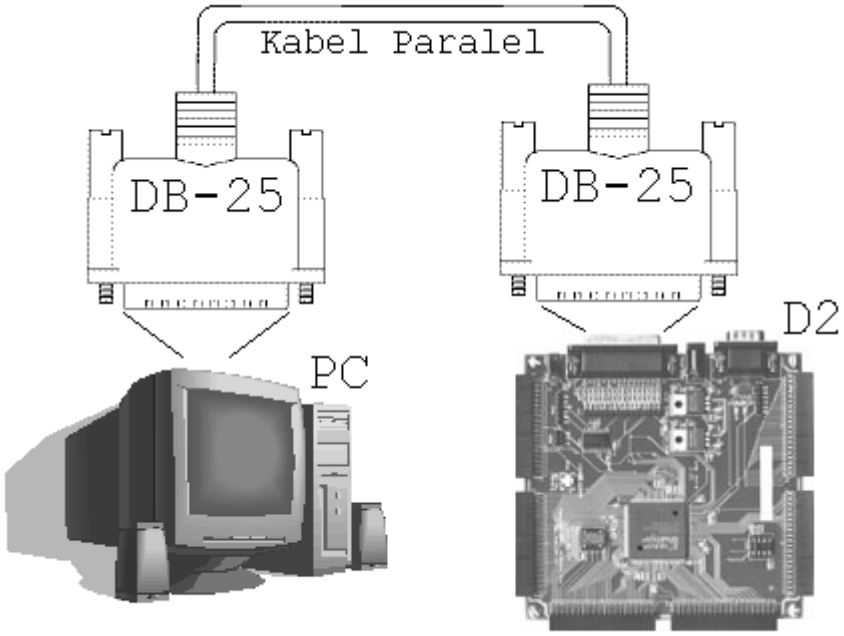
Berikut langkah-langkah untuk menghasilkan file `.bit` dan langkah-langkah untuk mengkonfigurasi FPGA:

1. Hubungkan modul D2 dengan PC seperti terlihat pada Gambar 4.6.
2. *Click kanan* pada "*Generate Programming File*" kemudian pilih "*Properties*…".
3. Pada label "*Startup options*" rubah menu "*FPGA Start-Up Clock*" dari "CCLK" menjadi "JTAG *Clock*". *Click* "*OK*".
4. *Double click* pada "*Generate Programming File*".
5. *Double click* pada "*Configure Device* (iMPACT)".




**Gambar 4.5** Penempatan dan Hubungan *Slice* yang Digunakan

6. Pilih "*Configure Devices*" untuk pertanyaan "*What do you want to do first*?". *Click* "*Next*".
7. Pilih "*Boundary-Scan Mode*" untuk pernyataan "*I want to configure device via*:". *Click* "*Next*".
8. Kemudian pilih "*Automatically connect to cable and identify Boundary-Scan chain*". *Click* "*Finish*".
9. Pilih *file* `.bit` untuk mengkonfigurasi FPGA.
10. *Click* kanan pada gambar "*device*" dan pilih "*Program*…".
11. Tunggu hingga pemrograman selesai dilakukan dan muncul tampilan "*Programming Success*".
12. Sebuah prosesor dengan arsitektur RISC telah dibangun pada FPGA.




**Gambar 4.6** Hubungan PC dengan Modul D2

Penghasil *clock* (*clock generator*) dengan frekuensi 50 MHz telah tersedia pada modul D2. Untuk menyesuaikannya dengan frekuensi maksimum rancangan, penghasil *clock* dapat diperkecil dengan menggunakan komponen pembagi *clock* (*clock divider*) atau komponen *counter*. *Clock* dari penghasil *clock* terhubung dengan pin 80 pada FPGA sedangkan *clock* menggunakan tombol terhubung dengan pin 77 pada FPGA.

### 4.2.3 Perolehan Data dari Proses Implementasi

Tabel 4.13 merupakan penundaan dari komponen kombinasional prosesor RISC pada proses implementasi untuk mendapatkan penundaan maksimum dari setiap tahap *pipeline*. Data pada Tabel 4.13 didapatkan dari

menu "*Analyze Post-Place & Route Static Timing* (*Timing Analyzer*)" dengan satuan nano detik (ns). Dari menu ini didapatkan pula bahwa penundaan pada IOBUF (*Input*/*Output Buffer*) untuk sinyal masukan sebesar 0,776ns, sedangkan untuk sinyal keluaran sebesar 4,668ns.

**Tabel 4.13** Penundaan Maksimum dari Masing-masing Tahap *Pipeline*

| Jalur | Waktu |
|---|---|
| IF | |
| Reg_PC → PC_Inc → MUX_I_PC_1 → Reg_PC_1 | **10.199** |
| Reg_PC → PC_Inc → MUX_C → MUX_I_PC → Reg_PC | 8.197 |
| Reg_PC → Pin_Inst_Add | **9.560** |
| Pin_Inst → Int_Ctrl → MUX_I_PC → Reg_PC | **12.481** |
| DO | |
| PC_1 → PC_2 | 3.728 |
| Reg_IR → Reg_JA | 5.448 |
| Reg_IR → Reg_BS | 11.315 |
| Reg_IR → Register_File → MUX_A → Reg_Bus_A | 14.557 |
| Reg_IR → ID → Data_Forwarding → MUX_A → Reg_Bus_A | 13.689 |
| Reg_IR → Register_File → MUX_B → Reg_Bus_B | **15.951** |
| Reg_IR → ID → Data_Forwarding → MUX_B → Reg_Bus_B | 13.311 |
| EX | |
| Reg_PC_2 → Adder → MUC_C → MUX_I_PC → Reg_PC | 10.348 |
| Reg_JA → Adder → MUC_C → MUX_I_PC → Reg_PC | 10.630 |
| Reg_MD_1 → Int_Ctrl → Reg_PC | 13.133 |
| Reg_DA_1 → Data_Forwarding → MUX_A → Reg_Bus_A | 12.381 |
| Reg_BS → Branch_Ctrl → MUC_C → MUX_I_PC →Reg_PC | 15.320 |
| Reg_MW → Int_Ctrl → MUX_I_PC → Reg_PC | 12.826 |
| Reg_MW → Pin_IntACK | 16.298 |
| Reg_LS → MCO → Pin_BSel | 11.957 |
| Reg_FS → Function_Unit → Reg_F | 16.832 |
| Reg_FS → Function_Unit → Branch_Ctrl → MUX_C → MUX_I_PC → Reg_PC | 22.332 |
| Reg_Bus_A → Pin_Data_Address | 9.802 |
| Reg_Bus_B → Function_Unit → Reg_F | 18.584 |
| Reg_Bus_B → Function_Unit → Branch_Ctrl → MUX_C → MUX_I_PC → Reg_PC | 22.825 |
| Reg_Bus_B → Function_Unit → Branch_Ctrl → MUX_C → Int_Ctrl → Pin_IntACK | **26.266** |
| Reg_Bus_B → Function_Unit → MUX_D_DF → MUX_B → Reg_Bus_B | **23.780** |
| Reg_Bus_B → Function_Unit → Register_File → Reg_Bus_A | 21.144 |
| Reg_Bus_B → MCO → Pin_Data | 12.861 |
| Inst → Int_Ctrl → Int → MUX_I_PC | **12.481** |
| Inst → Int_Ctrl → Int → IntACK | **15.953** |
| WB | |
| Reg_MD → Bus_D | **6.991** |
| Reg_F → Bus_D | 6.053 |
| Reg_DataReg → Bus_D | 5.264 |

*Pipeline* yang baik adalah *pipeline* dengan panjang penundaan pada masing-masing tahap adalah sama. Dari Tabel 4.13 dapat dilihat bahwa penundaan terlama pada tahap IF yaitu sebesar 12,481ns, pada tahap DO yaitu sebesar 15,951ns, pada tahap EX yaitu sebesar 26.266ns, dan pada tahap WB yaitu sebesar 6.991ns.

Data ini menunjukkan panjangnya penundaan *pipeline* pada tahap EX adalah yang terbesar, yaitu 1,647 kali tahap DO (tahap terpanjang setelah EX). Pembagian tahap EX kedalam dua tahap adalah dimungkinkan utnuk meningkatkan frekuensi prosesor keseluruhan.

Beberapa metoda untuk menghindari *data hazard* adalah dengan NOP (*No OPeration*) dari *software*, *data hazard stall*, dan *data forwarding*. Beberapa kekurangan dan kelebihan pada masing-masing metoda dapat dilihat pada Tabel 4.14.

**Tabel 4.14** Perbandingan Solusi untuk Menghindari *Data Hazard*

| Metoda | Waktu Kompilasi Program | Penambahan Jumlah Slice | Penambahan Waktu Pipeline | Penundaan Eksekusi Instruksi |
|---|---|---|---|---|
| NOP Software | Lama | Tidak ada | Tidak ada | Terjadi |
| Data Hazard Stall | Cepat | Sedikit | Sedikit | Terjadi |
| Data Forwarding | Cepat | Banyak | Banyak | Tidak terjadi |

Dari Tabel 4.13 dan Tabel 4.14 dapat dilihat bahwa penambahan waktu akibat mengintegrasikan *data forwarding* (Reg_Bus_B → Function_Unit → MUX_D_DF → MUX_B → Reg_Bus_B = 23,780ns) yang seharusnya menyebabkan penambahan waktu tunda *pipeline* tidak terjadi

karena terdapat penundaan akibat komponen lain yang lebih lama (Reg_Bus_B → Function_Unit → Branch_Ctrl → MUX_C → Int_Ctrl → Pin_IntACK = 26.266ns). Sehingga solusi terbaik diantara ketiga solusi di atas untuk menghindari *data hazard* adalah menggunakan *data forwarding*.

## 4.3 Evaluasi

Evaluasi dilakukan dengan tujuan untuk mengetahui apakah prosesor RISC yang dirancang dapat bekerja sesuai dengan yang diharapkan. Untuk itu, beberapa perancangan komponen (RAM, ROM, dan pengontrol *input*/*output*) dan program penguji dilakukan.

### 4.3.1 Persiapan Evaluasi

#### 4.3.1.1 Komponen-komponen Evaluasi

Dua komponen yang digunakan untuk evaluasi yaitu memori data (RAM) dan memori instruksi (ROM). RAM (`ram32x32s_1.vhd`) yang digunakan yaitu 32 buah 32 *bit aktif low* SRAM (*Static Random Access Memory*). ROM (`rom.vhd`) yang digunakan bergantung pada program pengujian yang digunakan. Gabungan komponen yang digunakan untuk evaluasi (prosesor RISC, RAM, dan ROM) selanjutnya akan disebut sebagai komponen *micro controler* (μC).

#### 4.3.1.2 Pembuatan Program Pengujian

Terdapat enam jenis program pengujian yang dibuat dengan tujuan untuk menunjukkan apakah prosesor yang dirancang berjalan

sesuai dengan apa yang diharapkan. Sebelum melakukan pengujian terhadap perangkat keras, pengujian terlebih dahulu disimulasikan menggunakan *software* Xilinx Foundation seri 4.1i. Setelah program yang disimulasi dinyatakan berhasil barulah program tersebut diujikan pada perangkat keras.

Pengujian dilakukan terhadap instruksi-instruksi dasar, instruksi-instruksi percabangan dan *set*, penggunaan *data forwarding* dan *branch prediction*, instruksi-instruksi *load* dan *store*, penggunaan *stack* dan prosedur, penggunaan *interrupt*, serta *register-register* pada *register file*. Gambaran umum keenam program pengujian tersebut adalah sebagai berikut:

**1. Test 1 : Pengujian Terhadap Beberapa Instruksi Dasar**

Instruksi dasar yang dimaksud adalah instruksi-instruksi aritmatika, logika, dan pergeseran serta dimasukkan juga instruksi *load address* (LA) dan *load upper immediate* (LUI). Pengujian dilakukan dengan cara mengkalkulasi dan memanipulasi data-data menggunakan instruksi yang akan di uji, kemudian hasilnya akan ditampilkan pada LED. Pengujian dinyatakan berhasil jika keluaran yang ditampilkan pada LED sama dengan hasil perhitungan yang dilakukan pada program penguji.

Tabel 4.15 merupakan langkah-langkah perhitungan yang dilakukan oleh instruksi-instruksi pada program penguji.

**Tabel 4.15** Program Pengujian Instruksi Dasar, LA, dan LUI

| No | Instruksi | Proses | Mnemonic |
|---|---|---|---|
| 01 | R01 <= R00 + 0000 1234h | R01=00001234h | ADIU |
| 02 | R04 <= R00 + FFFF 9ABCh | R04=FFFF9ABCh | ADI |
| 03 | R02 <= R01 sll 16d | R02=12340000h | SLL |
| 04 | R05 <= R04 - 0000 9AACh | R05=FFFF0010h | SBIU |
| 05 | R07 <= R00 + 0000 DEF0h | R07=0000DEF0h | ADIU |
| 06 | R06 <= R04 sll R05 | R06=9ABC0000h | SLLV |
| 07 | R03 <= R02 or 0000 5678h | R03=12345678h | ORI |
| 08 | R08 <= R06 or R07 | R08=9ABCDEF0h | OR |
| 09 | R09 <= R08 + R03 | R09=ACF13568h | ADD |
| 10 | R10 <= R08 - R03 | R10=88888878h | SUB |
| 11 | R11 <= R09 or R03 | R11=BEF57778h | OR |
| 12 | R12 <= R10 or 0000 FF00h | R12=8888FF78h | ORI |
| 13 | R13 <= R10 xor R09 | R13=2479BD10h | XOR |
| 14 | R14 <= R09 xor 0000 F0F0h | R14=ACF1C598h | XORI |
| 15 | R15 <= R13 nor R00 | R15=DB8642EFh | NOR |
| 16 | R16 <= R13 nor 0000 0000 | R16=DB8642EFh | NORI |
| 17 | R17 <= R01 slr 0000 0002h | R17=0000048Dh | SLR |
| 18 | R18 <= R11 slr R05 | R18=0000BEF5h | SLRV |
| 19 | R19 <= R15 sar 0000 0011h | R19=FFFFEDC3h | SAR |
| 20 | R20 <= R15 sar R05 | R20=FFFFDB86h | SARV |
| 21 | R21 <= PC | R21=00000015h | LA |
| 22 | R22 <= R04 - FFFF 9234h | R22=00000888h | SBI |
| 23 | R23 <= R03 and R08 | R23=12345670h | AND |
| 24 | R24 <= 0000 9876h sll 16d | R24=98760000h | LUI |
| 25 | R26 <= R20 and 0000 AAAAh | R26=00008A82h | ANDI |
| 26 | R25 <= R00 + 0000 0080h | R25=00000080h | ADIU |

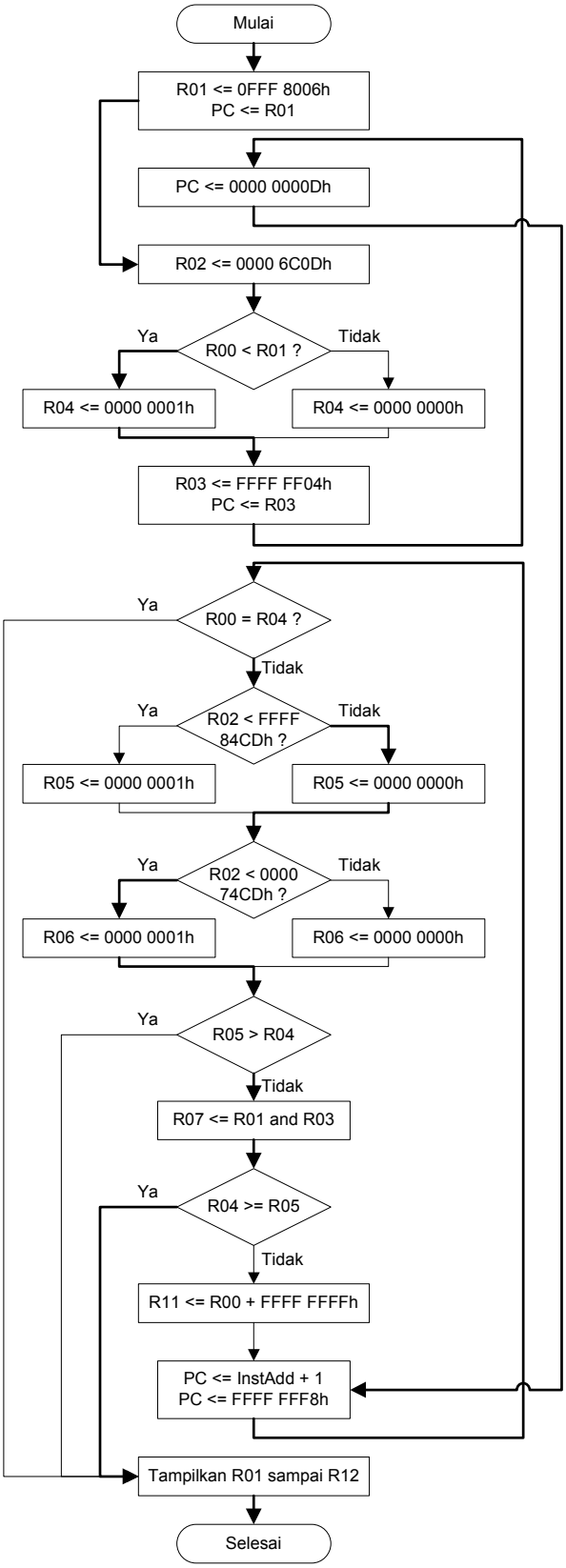
```
Menampilkan R00 sampai R25:

     R00 = 0000 0000 h          R01 = 0000 1234 h

     R02 = 1234 0000 h          R03 = 1234 5678 h

     R04 = FFFF 9ABC h          R05 = FFFF 0010 h

     R06 = 9ABC 0000 h          R07 = 0000 DEF0 h

     R08 = 9ABC DEF0 h          R09 = ACF1 3568 h

     R10 = 8888 8878 h          R11 = BEF5 7778 h

     R12 = 8888 FF78 h          R13 = 2479 BD10 h

     R14 = ACF1 C598 h          R15 = DB86 42EF h

     R16 = DB86 42EF h          R17 = 0000 048D h

     R18 = 0000 BEF5 h          R19 = FFFF EDC3 h

     R20 = FFFF DB86 h          R21 = 0000 0015 h
```

```
R22 = 0000 0888 h          R23 = 1234 5670 h
R24 = 9876 0000 h          R25 = 0000 0080 h
R26 = 0000 8A82 h
```

**2. Test 2 : Pengujian Terhadap Instruksi Percabangan dan *Set***

Gambar 4.7 menunjukkan *flowchart* untuk pengujian instruksi percabangan dan *set*. Instruksi percabangan yang terlihat pada Gambar 4.7 hanya menggunakan beberapa dari keseluruhan instruksi percabangan, terutama percabangan bersyarat. Untuk menguji percabangan bersyarat lainnya dapat dilakukan dengan mengganti instruksi percabangan yang ada pada *flowchart* dengan instruksi percabangan yang akan diuji. Jika program yang dibuat mengikuti alur seperti pada Gambar 4.7 maka *register-register* yang akan ditampilkan (R01 sampai R12) akan menunjukkan hasil sebagai berikut:

```
R01 = 0FFF 8006 h          R02 = 0000 6C0D h
R03 = FFFF FF04 h          R04 = 0000 0001 h
R05 = 0000 0000 h          R06 = 0000 0001 h
R07 = 0FFF 8004 h          R08 = 0000 0000 h
R09 = 0000 0000 h          R10 = 0000 0000 h
R11 = 0000 0000 h          R12 = 0000 0000 h
```




**Gambar 4.7** *Flowchart* Pengujian Instruksi Percabangan dan *Set*

3. **Test 3 : Pengujian Terhadap Penggunaan *Data Forwarding* dan *Branch Prediction***

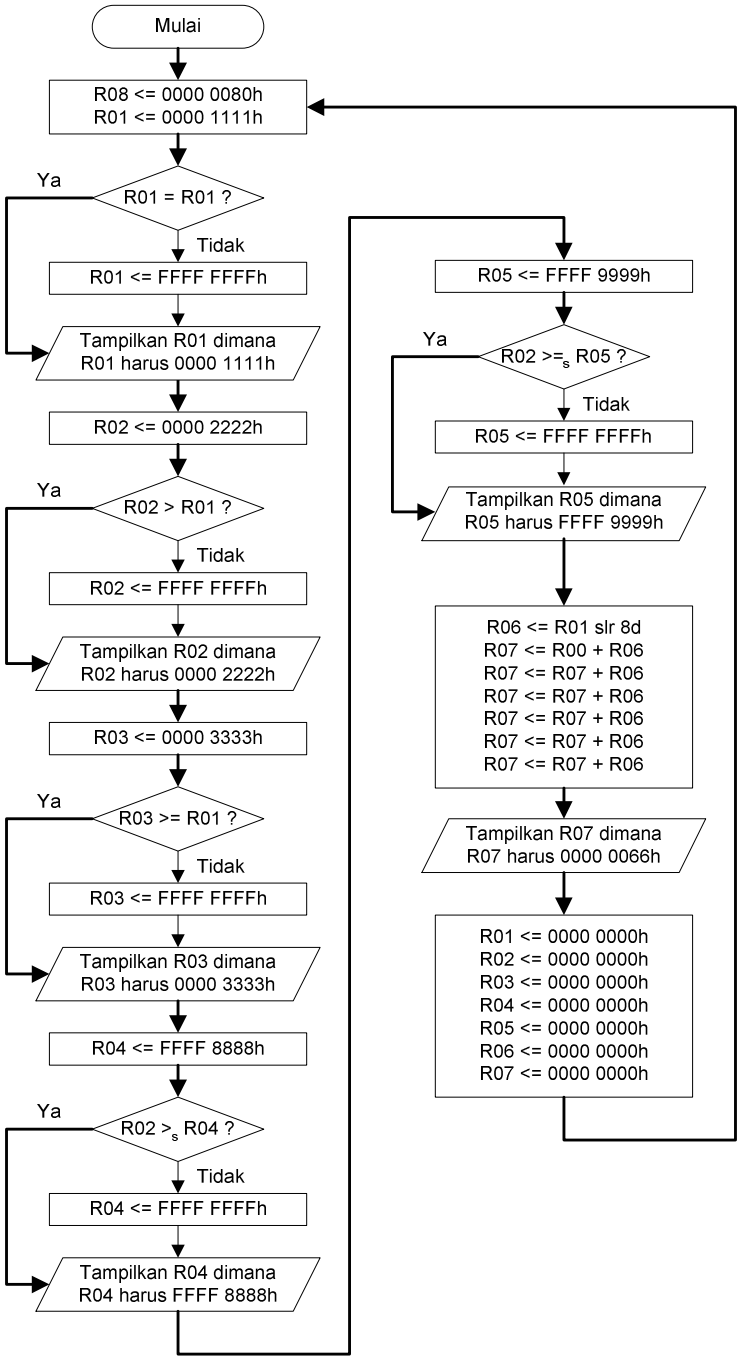
Pengujian terhadap *data forwarding* dilakukan menggunakan instruksi yang mengandung *data depedency* sedangkan pengujian terhadap *branch prediction* dilakukan dengan meletakan instruksi yang menyebabkan perubahan nilai *register* setelah instruksi percabangan seperti pada Gambar 4.8.

Operator “>” dan “>=” digunakan pada instruksi percabangan yang membandingkan bilangan tidak bertanda yaitu BH dan BHE sedangkan operator “$>_s$” dan “$>=_s$” digunakan pada instruksi percabangan yang membandingkan bilangan bertanda yaitu BG dan BGE.

4. **Test 4 : Pengujian Instruksi *Load* dan *Store***

Pengujian terhadap instruksi *store*, yaitu SB, SH, dan SW dilakukan dengan cara menyimpan isi *register* ke memori data. Untuk memeriksa kebenarannya, data yang telah disimpan pada memori data diambil dan ditampilkan ke LED.

Pengujian terhadap instruksi *load*, yaitu LB, LH, dan LW dilakukan dengan cara mengambil isi memori data dan menyimpannya ke dalam *register*. Sebelumnya memori data diisi dengan nilai tertentu. Untuk memeriksa kebenarannya, data pada *register* ditampilkan ke LED.




**Gambar 4.8** *Flowchart* Pengujian *Data Forwarding* dan *Branch Prediction*

**5. Test 5 : Pengujian Penggunaan *Stack* dan Prosedur**

Pengujian penggunaan prosedur dan *stack* dilakukan dengan cara memenuhi program yang dibuat menggunakan bahasa C, dimana program tersebut memanggil sebuah fungsi (Proc1) dengan melakukan pengiriman parameter dan mendapatkan pengembalian nilai. Operasi *stack* digunakan pada saat pemanggilan fungsi dan pada saat kembali dari fungsi ke program utama.

Program dalam bahasa C adalah sebagai berikut:

```
void main(void)
{
   int ADD1, ADD2;
   int R01 = 2;
   int R02 = 4;
   proc1(R01, &R02);
   ADD1 = R01;              // ADD1 = 2
   ADD2 = R02;              // ADD2 = 9
}
void proc1(int R01, int *R02)
{
   R01  += 5;               // R01  = 7
   *R02 += 5;               // *R02 = 9
}
```

Pengujian dinyatakan berhasil jika dengan nilai masukkan 2 dan 4 menghasilkan nilai keluaran 2 dan 9 yang ditampilkan pada LED.

**6. Test 6 : Pengujian *Interrupt***

Program untuk pengujian *interrupt* dibagi menjadi tiga bagian, yaitu program utama, program *interrupt* 1, dan program *interrupt* 2. Program utama akam menampilkan animasi LED pantul dan akan dikerjakan terus selama *interrupt* 1 dan *interrupt* 2 belum meminta layanan.

Program *interrupt* 1 akan dikerjakan jika terjadi penekanan tombol *interrupt* 1 dan layanan diberikan oleh prosesor. Program *interrupt* 1 akan mengerjakan animasi penumpukkan LED. Selama penumpukkan terjadi, tidak ada permintaan *interrupt* yang akan dilayani. Setelah penumpukkan selesai dilakukan maka program utama akan kembali dilanjutkan dan perminataan *interrupt* akan kembali dilayani.

Program *interrupt* 2 akan dikerjakan jika terjadi penekanan tombol *interrupt* 2 dan layanan diberikan oleh prosesor. Program *interrupt* 2 akan mengirim data ke prosesor sesuai dengan pengaturan kedelapan saklar. Data yang dikirim berdasarkan pengaturan kedelapan saklar akan ditampilkan pada delapan buah LED kemudian akan diberikan selang waktu sesaat sebelum keluar dari program *interrupt* 2 dan kembali ke program utama.

Tabel 4.16 menunjukkan rangkuman dari keenam program pengujian dengan instuksi-intruksi dan *register-register* yang digunakan pada masing-masing program. Dimana simbol ‘0’ menunjukkan

instruksi/*register* yang digunakan dalam masing-masing program pengujian sedangkan simbol '√ ' menunjukkan apakah instruksi/*register* sudah pernah digunakan dalam salah satu program pengujian.

**Tabel 4.16** Program Pengujian dengan Instruksi dan *Register* yang Diujinya

| Instruksi/ Register | Test1 | Test2 | Test3 | Test4 | Test5 | Test6 | Keseluruhan |
|---|---|---|---|---|---|---|---|
| ADD | 0 | | 0 | 0 | | 0 | √ |
| ADI | 0 | 0 | 0 | 0 | 0 | 0 | √ |
| ADIU | 0 | 0 | 0 | 0 | | 0 | √ |
| SUB | 0 | | | | | | √ |
| SBI | 0 | | | 0 | 0 | | √ |
| SBIU | 0 | | | 0 | | | √ |
| AND | 0 | 0 | | | | | √ |
| ANDI | 0 | | | | | | √ |
| OR | 0 | | | | | 0 | √ |
| ORI | 0 | 0 | | | | | √ |
| XOR | 0 | | | | | | √ |
| XORI | 0 | | | | | | √ |
| NOR | 0 | | | | | | √ |
| NORI | 0 | | | | | | √ |
| SLR | 0 | 0 | 0 | 0 | | 0 | √ |
| SLRV | 0 | | | | | | √ |
| SLL | 0 | | | | | 0 | √ |
| SLLV | 0 | | | | | | √ |
| SAR | 0 | | | | | | √ |
| SARV | 0 | | | | | | √ |
| LUI | 0 | 0 | 0 | 0 | | 0 | √ |
| LA | 0 | | | | | | √ |
| LB | | | | 0 | | 0 | √ |
| LH | | | | 0 | | | √ |
| LW | | | | 0 | 0 | | √ |
| SB | | | | 0 | | 0 | √ |
| SH | | | | 0 | | | √ |
| SW | 0 | 0 | 0 | 0 | 0 | | √ |
| SLT | | 0 | | | | | √ |
| SLTI | | 0 | | | | | √ |
| DI | | | | | | 0 | √ |
| EI | | | | | | 0 | √ |
| BE | | 0 | 0 | | | 0 | √ |
| BH | | 0 | 0 | | | 0 | √ |
| BHE | | 0 | 0 | | | | √ |
| BG | | | 0 | | | | √ |
| BGE | | | 0 | | | | √ |
| JMP | | 0 | 0 | | | 0 | √ |
| JL | | 0 | | | | | √ |
| JR | | 0 | | | 0 | 0 | √ |
| JRL | | 0 | | | 0 | | √ |

**Tabel 4.16** Program Pengujian dengan Instruksi dan *Register* yang Diujinya (lanjutan)

| Instruksi/ Register | Test1 | Test2 | Test3 | Test4 | Test5 | Test6 | Keseluruhan |
|---|---|---|---|---|---|---|---|
| R00 | 0 | 0 | 0 | 0 | 0 | 0 | √ |
| R01 | 0 | 0 | 0 | 0 | 0 | 0 | √ |
| R02 | 0 | 0 | 0 | 0 | 0 | 0 | √ |
| R03 | 0 | 0 | 0 | | | 0 | √ |
| R04 | 0 | 0 | 0 | | | 0 | √ |
| R05 | 0 | 0 | 0 | | 0 | 0 | √ |
| R06 | 0 | 0 | 0 | | | 0 | √ |
| R07 | 0 | 0 | 0 | | | 0 | √ |
| R08 | 0 | 0 | 0 | | | 0 | √ |
| R09 | 0 | 0 | | | | | √ |
| R10 | 0 | 0 | | | | | √ |
| R11 | 0 | 0 | | 0 | 0 | 0 | √ |
| R12 | 0 | 0 | | 0 | 0 | | √ |
| R13 | 0 | | | 0 | | 0 | √ |
| R14 | 0 | | | 0 | | 0 | √ |
| R15 | 0 | | | 0 | | 0 | √ |
| R16 | 0 | | | 0 | | | √ |
| R17 | 0 | | | 0 | | 0 | √ |
| R18 | 0 | | | 0 | | | √ |
| R19 | 0 | | | 0 | | | √ |
| R20 | 0 | | | 0 | | | √ |
| R21 | 0 | | | 0 | | | √ |
| R22 | 0 | | | 0 | | 0 | √ |
| R23 | 0 | | | | | 0 | √ |
| R24 | 0 | | | | | 0 | √ |
| R25 | 0 | 0 | | 0 | | 0 | √ |
| R26 | | | | 0 | | 0 | √ |
| R27 | | | | 0 | | 0 | √ |
| R28 | | | | 0 | | | √ |
| R29 | | | | 0 | 0 | | √ |
| R30 | | | | | | 0 | √ |
| R31 | | | | | 0 | | √ |

#### 4.3.1.3 Pembuatan *Assembler* Sederhana

Pembuatan *assembler* ditujukan untuk memudahkan mendekode program yang dibuat dengan bahasa *assembly* menjadi kumpulan 32 bit instruksi. Karena perancangan *assembler* yang sangat sederhana maka muncul beberapa keterbatasan dalam pembuatan program pengujian atau program aplikasi. Beberapa keterbatasan tersebut adalah tidak tersedianya fungsi pelabelan, penulisan instruksi tidak dapat diberikan komentar,

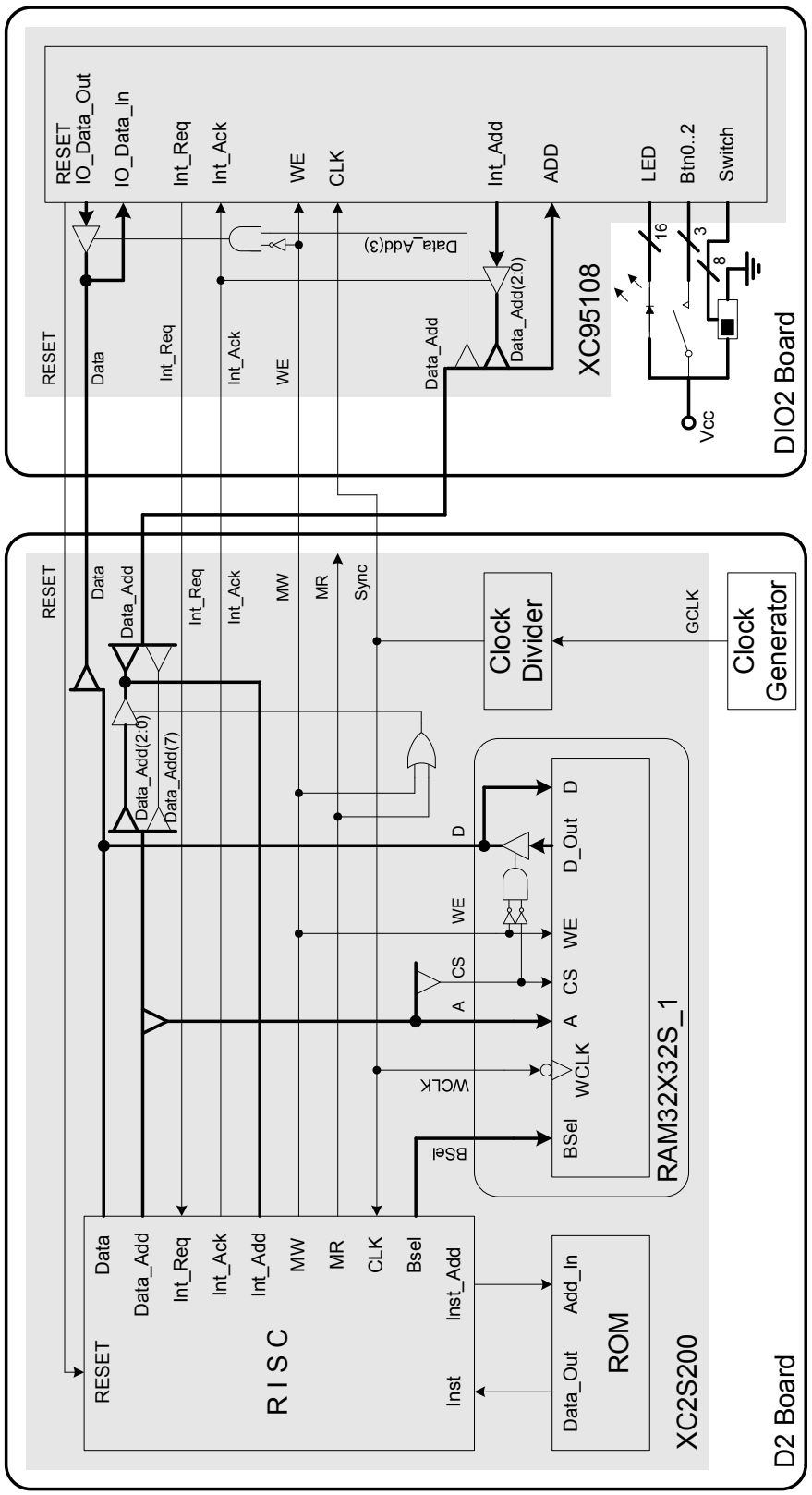
instruksi harus ditulis dengan huruf besar, jarak antar instruksi hanya satu kali *enter* (tidak boleh terdapat baris kosong), tidak diijinkan penggunaan *tabulation*, setelah koma harus diberi spasi, nilai *immediate* dan target *jump* diawali dengan tanda pagar (#) dan diakhiri dengan huruf 'h' yang menandakan bilangan heksadesimal (contoh : `#1234h` untuk nilai *immediate* dan `#1234567h` untuk target *jump*), penulisan *register* menggunakan dua *digit* (contoh : `R00`, `R07`, `R12`, dan `R31`) dan pemeriksaan kesalahan hanya pada penulisan instruksi.

Contoh instruksi yang benar:

```
ADD R01, R02, R29
SUB R03, R20, #AC34h
JMP #ADEF123h
```

#### 4.3.1.4 Perancangan Komponen *Input*/*Output*

Gambar 4.9 merupakan skematik hubungan antara modul D2 (FPGA) dengan modul DIO2 (CPLD/*Complex Programmable Logic Device*) pada saat dilakukan evaluasi dan pengambilan data secara perangkat keras. Signal-signal yang saling terhubung dapat dilihat pada Tabel 4.17, dimana Pin_A merupakan *pin* keluaran FPGA, Pin_B merupakan *pin* konektor A pada modul D2, Pin_C merupakan *pin* konektor B pada modul DIO2, dan Pin_D merupakan *pin* keluaran pada CPLD.




Gambar 4.9 Rancangan Evaluasi

**Tabel 4.17** *Pin-pin* Penghubung Antara Modul D2 dengan DIO2

| Modul D2 | | | Modul DIO2 | | |
|---|---|---|---|---|---|
| **Nama Signal** | **Pin_A** | **Pin_B** | **Pin_C** | **Pin_D** | **Nama Signal** |
| Data(0) | 41 | 23 | 17 | 11 | Data(0) |
| Data(1) | 37 | 24 | 18 | 7 | Data(1) |
| Data(2) | 43 | 21 | 19 | 6 | Data(2) |
| Data(3) | 42 | 22 | 20 | 5 | Data(3) |
| Data(4) | 45 | 19 | 21 | 4 | Data(4) |
| Data(5) | 44 | 20 | 22 | 3 | Data(5) |
| Data(6) | 47 | 17 | 23 | 2 | Data(6) |
| Data(7) | 46 | 18 | 24 | 1 | Data(7) |
| Data_Add(0) | 59 | 12 | 30 | 83 | Data_Add(0) |
| Data_Add(1) | 62 | 9 | 31 | 81 | Data_Add(1) |
| Data_Add(2) | 61 | 10 | 32 | 80 | Data_Add(2) |
| Data_Add(3) | 67 | 7 | 33 | 79 | Data_Add(3) |
| RESET | 63 | 8 | 34 | 77 | RESET |
| IntACK | 69 | 5 | 35 | 75 | IntACK |
| Sync | 49 | 15 | 25 | 13 | CLK |
| WE | 48 | 16 | 26 | 9 | WE |
| IntREQ | 58 | 13 | 27 | 76 | IntREQ |

Tabel 4.18 menunjukkan *pin-pin* keluaran CPLD yang terhubung dengan tombol, saklar, dan LED. Dimana btn0 digunakan untuk *reset*, btn1 digunakan untuk *interrupt* 1, dan btn2 digunakan untuk *interrupt* 2.

**Tabel 4.18** *Pin-pin* keluaran CPLD yang terhubung dengan tombol, saklar, dan LED

| Nama Signal | Pin Keluaran CPLD |
|---|---|
| LED(0) | 82 |
| LED(1) | 12 |
| LED(2) | 14 |
| LED(3) | 15 |
| LED(4) | 17 |
| LED(5) | 18 |
| LED(6) | 19 |
| LED(7) | 20 |

**Tabel 4.18** *Pin-pin* keluaran CPLD yang terhubung dengan tombol, saklar, dan LED (lanjutan)

| Nama Signal | Pin Keluaran CPLD |
|---|---|
| LED(8) | 62 |
| LED(9) | 69 |
| LED(10) | 67 |
| LED(11) | 68 |
| LED(12) | 70 |
| LED(13) | 71 |
| LED(14) | 72 |
| LED(15) | 74 |
| Btn0 | 84 |
| Btn1 | 47 |
| Btn2 | 66 |
| Switch(0) | 55 |
| Switch(1) | 61 |
| Switch(2) | 60 |
| Switch(3) | 62 |
| Switch(4) | 64 |
| Switch(5) | 65 |
| Switch(6) | 66 |
| Switch(7) | 67 |

Tabel 4.19 menunjukkan alamat yang digunakan untuk melewatkan data dari dan ke modul DIO2.

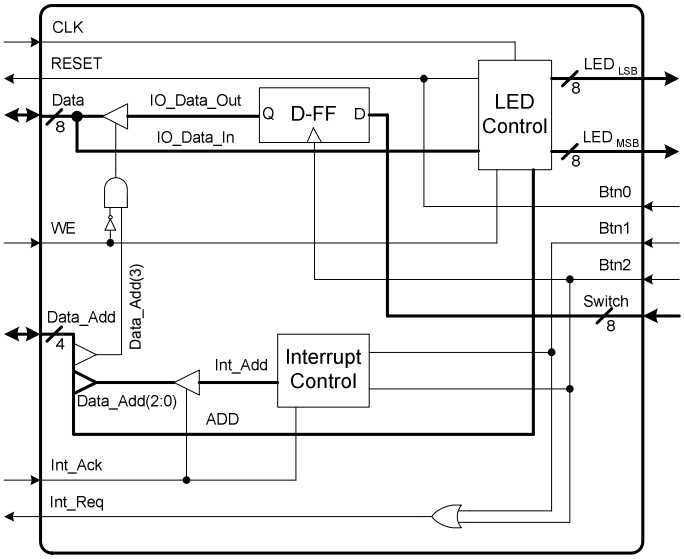
**Tabel 4.19** Alamat untuk Melewatkan Data Dari dan Ke Modul DIO2

| Data_Add(7 \|\| 2:0) | WE | Operasi |
|---|---|---|
| 1000 | 0 | Baca data 8 bit dari saklar |
| | 1 | Tulis data ke 8 bit LSB LED |
| 1001 | 1 | Tulis data ke 8 bit MSB LED |

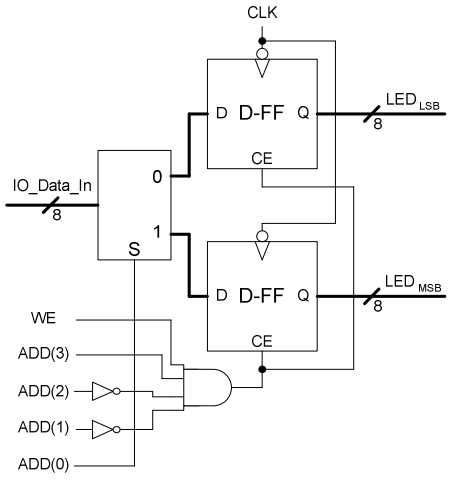
Data_Add(7) berfungsi untuk menentukan apakah data berhubungan dengan komponen memori (SRAM) atau modul D2IO.

Gambar 4.10 merupakan skematik pengontrol *input*/*ouput* pada modul DIO2. Pengontrol *input*/*output* terdiri dari LED *control* (Gambar

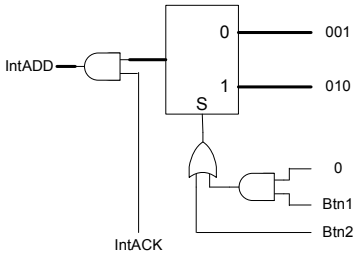
4.11) yang berfungsi menempatkan data pada LED yang dituju dan *interrupt control* (Gambar 4.12) berfungsi untuk meminta layanan *interrupt* dan memberikan alamat untuk masing-masing jenis *interrupt* jika permintaan layanan diterima.




**Gambar 4.10** Skematik Pengontrol *Input*/*Output* pada Modul DIO2




**Gambar 4.11** Skematik LED *Control* pada Pengontrol *Input*/*Output*




**Gambar 4.12** Skematik *Interrupt Control* pada Pengontrol *Input*/*Output*

Berikut langkah-langkah untuk menghasilkan *file* `.jed` dan langkah-langkah untuk mengkonfigurasi CPLD:

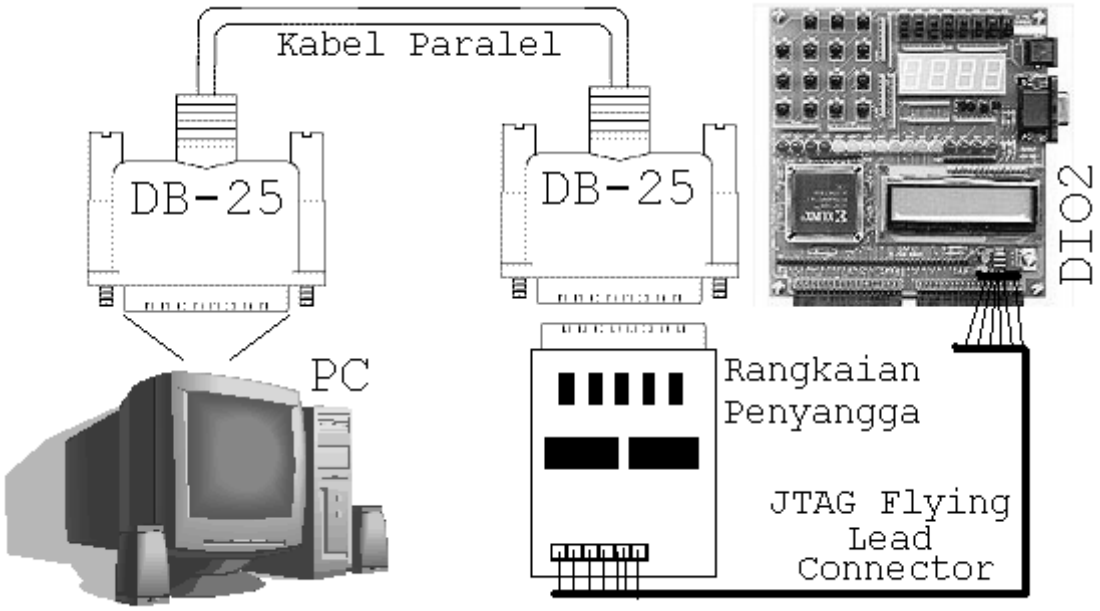
1. Hubungkan PC dengan rangkaian penyangga (Gambar 4.13) menggunakan kabel paralel (DB-25) kemudian hubungkan rangkaian penyangga dengan modul DIO2 menggunakan JTAG *Flying Lead Connector*. Hubungkan juga sumber tegangan.
2. Jalankan program "*Project Navigator*" dari Xilinx ISE WebPack 5.2i.
3. Buat proyek baru melalui *menu bar* "*File*" kemudian pilih "*New Project*...".
4. Isi nama proyek pada "*Project Name*" dan lokasi pada "*Project Location*". Isikan informasi pada Tabel 4.20 ke dalam "*Project Device Options*".
5. Masukkan *file-file* yang dibutuhkan (`Animasi_LED.vhd`, `BUFE8.vhd`, dan `BUFE3.vhd`) dengan cara *click menu bar* "*Project*" kemudian pilih "*Add Source*…".
6. *Double click* pada "*Synthesize*" untuk melakukan sintesis.

**Tabel 4.20** *Projenct Device Options* untuk CPLD

| Property Name | Value |
|---|---|
| Device Family | XC9500 CPLDs |
| Device | xc95108 |
| Package | PC84 |
| Speed Grade | -15 |
| Design Flow | XST VHDL |

7. Sebelum melakukan implementasi, harus dinyatakan terlebih dahulu hubungan antara signal-signal masukkan dan keluaran dengan *pin* dari CPLD. Hubungan sinyal dan *pin* dapat dikonfigurasi melalui *file* `.ucf` (`Animasi_LED.ucf`) atau melalui menu "*Create Timing Constraints*".
8. *Double click* pada "*Implement Design*" untuk melakukan implementasi.
9. *Double click* pada "*Generate Programming File*" untuk menghasilkan *file* `.jed`.
10. *Double click* pada "*Configure Device* (iMPACT)".
11. Pilih "*Configure Devices*" untuk pertanyaan "*What do you want to do first*?". *Click* "*Next*".
12. Pilih "*Boundary-Scan Mode*" untuk pernyataan "*I want to configure device via*:". *Click* "*Next*".
13. Kemudian pilih "*Automatically connect to cable and identify Boundary-Scan chain*". *Click* "*Finish*".
14. Pilih *file* `.jed` untuk mengkonfigurasi CPLD.
15. *Click* kanan pada gambar "*device*" dam pilih "*Program*…".

16. Berikan tanda benar pada pilihan "*Erase Before Programming*". *Click* "*OK*".
17. Tunggu hingga pemrograman selesai dilakukan dan muncul tampilan "*Programming Success*".




**Gambar 4.13** Hubungan PC, Rangkaian Penyangga, dan Modul DIO2

Tabel 4.21 merupakan rangkuman data yang diperoleh dari proses implementasi pengontrol *input*/*output* pada CPLD.

**Tabel 4.21** Rangkuman Laporan Hasil Implementasi Pengontrol *Input*/*Output*

| Macrocells | Pterms | Registers | Pins | Function Block Inputs | Perioda/ Frekunesi |
|---|---|---|---|---|---|
| 29/108 (27%) | 95/540 (18%) | 24/108 (23%) | 45/69 (66%) | 76/216 (36%) | 14 ns/ 71.429 MHz |

### 4.3.2 Evaluasi Perangkat Lunak (Simulasi)

Simulasi dilakukan menggunakan PC Pentium 4 1,7 GHz, memori sebesar 256 MB, sistem operasi Windows 98 Second Edition, *software* Xlinx Foundation seri 4.1i, serta program *assembler* sederhana.

Berikut adalah langkah-langkah untuk melakukan simulasi:

1. Pilih program penguji dari *file* `.asm` yang akan digunakan.
2. Gunakan program *assembler* untuk mendekode *file* `.asm` menjadi *file* `.bit` dengan cara:

   ```
   asm [file_sumber].asm [file_target].bit <ENTER>
   ```

3. *Copy* isi *file* `.bit` ke *file* `rom.vhd` dan berikan alamt untuk masing-masing instruksi.
4. Jalankan program "*Project Manager*" dari Xilinx Foundation 4.1i.
5. Buat proyek baru melalui *menu bar* "*File*" kemudian pilih "*New Project*…".
6. Berikan nama proyek dan lokasinya. Gunakan tipe "F4.1i" dan pilih "HDL" sebagai "*Flow*"-nya.
7. Pada *menu bar* "*Project*" pilih "*Add Source File(s)*…" kemudian masukkan semua *file* perancangan yang dibutuhkan.
8. Beberapa penyesuaian mungkin harus dilakukan. Foundation tidak mendukung *sensitivity list* yang spesifik maka gunakan hanya nama sinyalnya saja, contoh : rubah *sensitivity list* dari AA(4) menjadi AA.
9. Jika kesalahan penulisan sudah tidak ada, kemudian *click icon* "*Synthesis*".

10. Isi "*Top level*" dengan "uC", "*Family*" dengan "Spartan 2", "*Device*" dengan "2S200PQ208", dan "*Speed*" dengan "-6". "*Run*".
11. Jika tidak terjadi kesalahan pada proses sintesis, lanjutkan dengan proses simulasi (*click icon* "*Simulation*").
12. Masukkan sinyal-sinyal *input* dan *output* yang dibutuhkan melalui *menu bar* "*Signal*" kemudian pilih "*Add Signals*…".
13. Rubah frekuensi *clock* menjadi 20MHz melalui *menu bar* "*Options*" dan pilih "*Preferences*…". Isikan "20MHz" pada "B0 *Frequency*" atau "50ns" pada "B0 *Period*".
14. Isi data yang diperlukan pada sinyal masukkan. *Clock* terus menggunakan "*Simulation Step*" pada *tool bar* kemudian bandingkan hasilnya.

### 4.3.3 Evaluasi Perangkat Keras

Perangkat keras yang digunakan terdiri dari modul D2 dengan tipe Xilinx FPGA Spartan 2 XC2S200-PQ208, modul DIO2 dengan tipe Xilinx CPLD XC95108-PC84 dan satu unit komputer pribadi (PC) Pentium III 600 MHz dengan memori sebesar 256 MB dan sistem operasi Windows XP Professional..

Langkah-langkah mengkonfigurasi perangkat keras:

1. Pilih program penguji dari *file* `.asm` yang akan digunakan.
2. Gunakan program *assembler* untuk mendekode *file* `.asm` menjadi *file* `.bit` dengan cara:

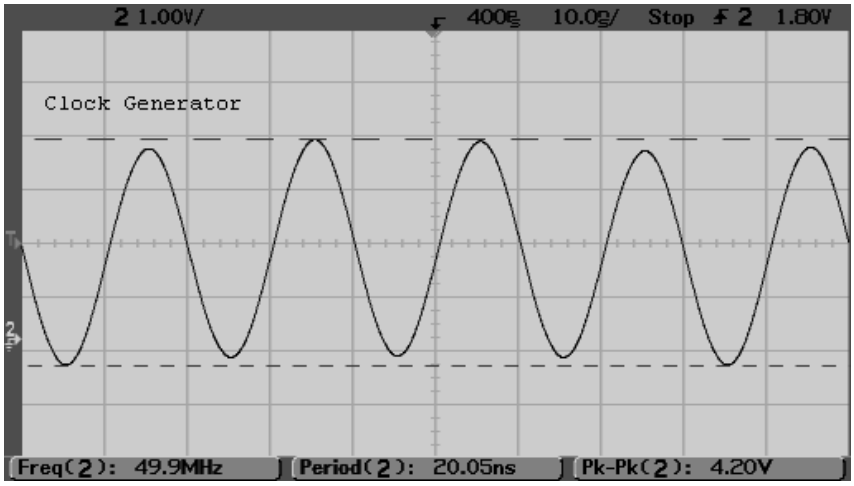
```
asm [file_sumber].asm [file_target].bit <ENTER>
```

3. *Copy* isi *file* `.bit` ke *file* `rom.vhd` dan berikan alamt untuk masing-masing instruksi.
4. Jalankan program "*Project Navigator*" dari Xilinx ISE WebPack 5.2i.
5. Buat proyek baru melalui *menu bar* "*File*" kemudian pilih "*New Project...*".
6. Isi nama proyek pada "*Project Name*" dan lokasi pada "*Project Location*". Isikan informasi pada Gambar 4.2 ke dalam "*Project Device Options*".
7. Masukkan *file-file* yang dibutuhkan dengan cara *click menu bar* "*Project*" kemudian pilih "*Add Source…*".
8. *Double click* pada "*Synthesize*" untuk melakukan sintesis.
9. Sebelum melakukan implementasi, harus dinyatakan terlebih dahulu hubungan antara signal-signal masukkan dan keluaran dengan *pin* dari FPGA. Hubungan sinyal dan *pin* dapat dikonfigurasi melalui *file* `.ucf` (`uC.ucf`) atau melalui menu "*Create Timing Constraints*".
10. *Click kanan* pada "*Generate Programming File*" kemudian pilih "*Properties…*".
11. Pada label "*Startup options*" rubah menu "*FPGA Start-Up Clock*" dari "CCLK" menjadi "JTAG *Clock*". *Click* "*OK*".
12. *Double click* pada "*Generate Programming File*".
13. Kemudian hubungkan PC dengan modul D2 menggunakan kabel paralel (DB-25). Hubungkan juga sumber tegangan.
14. *Double click* pada "*Configure Device* (iMPACT)".

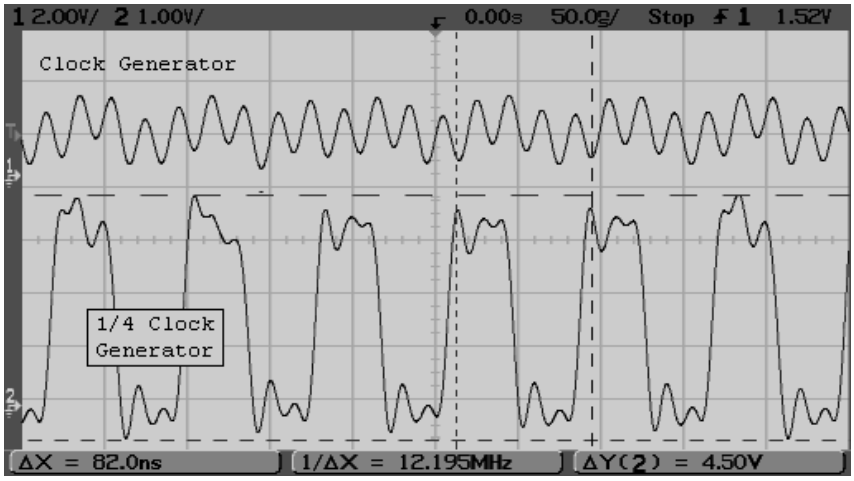
15. Pilih "*Configure Devices*" untuk pertanyaan "*What do you want to do first*?". *Click* "*Next*".
16. Pilih "*Boundary-Scan Mode*" untuk pernyataan "*I want to configure device via*:". *Click* "*Next*".
17. Kemudian pilih "*Automatically connect to cable and identify Boundary-Scan chain*". *Click* "*Finish*".
18. Pilih *file* `.bit` untuk mengkonfigurasi FPGA.
19. *Click* kanan pada gambar "*device*" dam pilih "*Program*…".
20. Tunggu hingga pemrograman selesai dilakukan dan muncul tampilan "*Programming Success*".
21. Hubungkan konektor A pada modul D2 dengan konektor B pada modul DIO2.
22. Lakukan pengujian. Beberapa program penguji menggunakan sumber *clock* dari penekanan tombol sedangkan program untuk pengujian *interrupt* menggunakan *clock* dari penghasil *clock*. Hasil dari pengujian akan ditampilkan pada LED dan tombol btn0 digunakan untuk me-*reset*. Untuk program pengujian *interrupt*, tombol btn1 dan btn2 digunakan untuk *interrupt* 1 dan *interrupt* 2 serta 8 buah saklar digunakan sebagai data yang akan dikirim ke prosesor.

Untuk menguji perioda/frekuensi perangkat keras digunakan program (`Test7.asm`) yang mengeluarkan data satu dan nol secara bergantian setiap 83.886.074 *clock*. Dalam pengujiannya digunakan frekuensi *clock generator*

(Gambar 4.14) sebesar 49,9 MHz, tetapi frekuensi yang digunakan untuk prosesor RISC (Gambar 4.15) hanya seperempatnya yaitu 12,195 MHz.



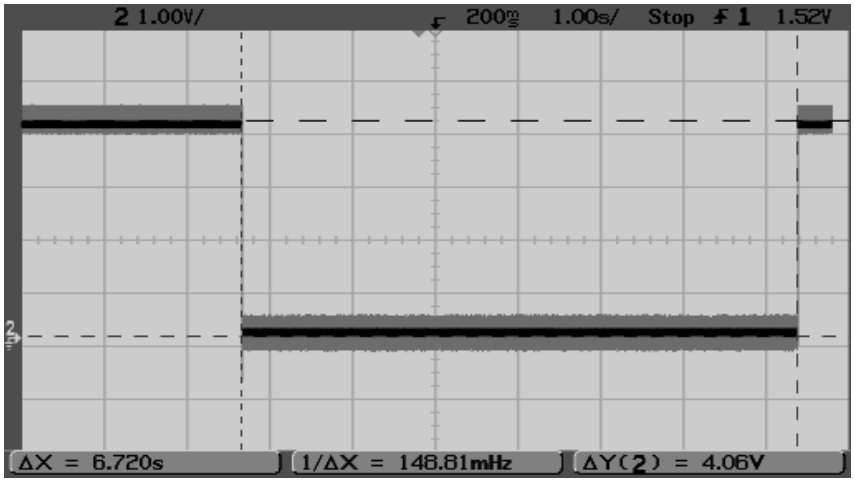

**Gambar 4.14** Frekuensi *Clock Generator*




**Gambar 4.15** Frekuensi pada Prosesor RISC

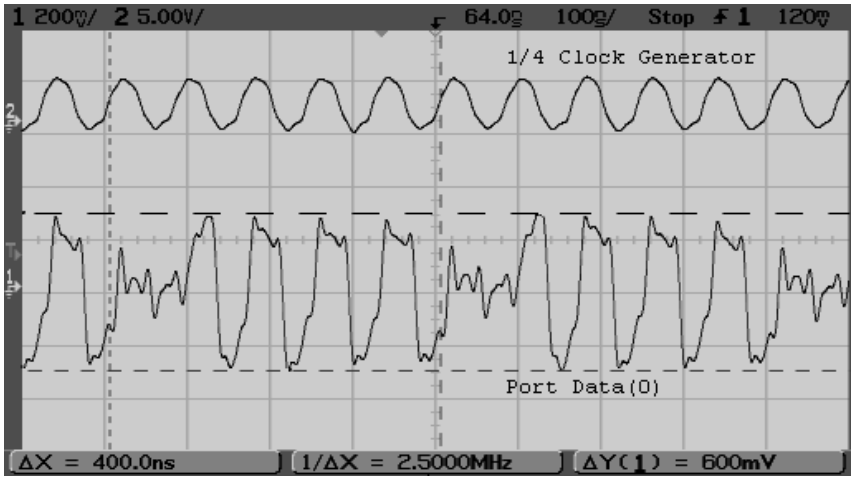
Hasil pengukuran (Gambar 4.16) menunjukkan bahwa untuk mengeksekusi instruksi 83.886.074 *clock* dibutuhkan waktu sebesar 6,720 S. Sehingga perioda untuk satu *clock* adalah 6,720 dibagi 83.886.074, yaitu

sebesar 81,109 ns. Perhitungan berdasarkan teori menunjukkan bahwa besarnya perioda untuk satu *clock* adalah 82 ns (12,5 MHz), yang diperoleh dari besarnya *clock generator* (50 MHz) dibagi empat.



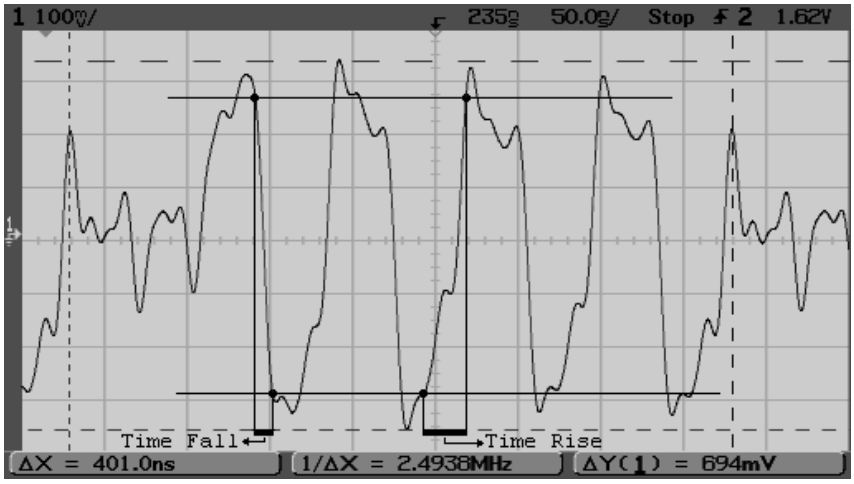

**Gambar 4.16** Waktu untuk 83.886.074 *Clock*

Gambar 4.17 merupakan sinyal keluaran dari *port* Data(0) pada saat mengeksekusi program.




**Gambar 4.17** Sinyal Keluaran dari *Port* Data(0)

Gambar 4.18 menunjukkan lamanya *time fall* (10 ns) dan *time rise* (25 ns) yang dibutuhkan oleh FPGA.




**Gambar 4.18** *Time Fall* dan *Time Rise* dari keluaran FPGA

### 4.3.4 Ringkasan Evaluasi

Hasil evaluasi perangkat lunak (simulasi) dan perangkat keras dari ketujuh program pengujian menunjukkan bahwa semua instruksi yang berjumlah 41 dan semua *register* yang berjumlah 32 bekerja dengan baik sesuai fungsinya masing-masing. Begitu juga dengan hasil pengujian perioda/frekuensi pada perangkat keras, dimana perioda/frekuensi yang dihasilkan sesuai dengan perhitungan yang dilakukan secara teori (ideal).

Tabel 4.22 menunjukkan data yang diperoleh dari proses implementasi *micro controler* (μC) dengan masing-masing program penguji menggunakan *software* Xilinx ISE WebPack seri 5.2i.

**Tabel 4.22** Rangkuman Laporan Hasil Implementasi dari Komponen *Micro Controler*

| Implementation | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Komponen** | **Slice** | **FF** | **4 input LUT** | **IOB** | **T-BUF** | **Timing Report** | |
| | | | | | | **Prd.** | **Freq.** |
| uC kosong | 34 (1%) | 34 (1%) | 2 (1%) | 66 (47%) | 32 (1%) | 6.041 | 165.536 |
| uC 1 instruksi | 204 (8%) | 146 (3%) | 378 (8%) | 66 (47%) | 32 (1%) | 9.930 | 100.705 |
| uC Test1 | 816 (34%) | 336 (7%) | 1500 (31%) | 70 (50%) | 64 (2%) | 22.360 | 44.723 |
| uC Test2 | 883 (37%) | 339 (7%) | 1611 (34%) | 70 (50%) | 64 (2%) | 40.960 | 24.414 |
| uC Test3 | 836 (35%) | 1546 (32%) | 1178 (25%) | 70 (50%) | 64 (2%) | 36.272 | 27.569 |
| uC Test4 | 887 (37%) | 348 (7%) | 1584 (33%) | 70 (50%) | 64 (2%) | 25.251 | 39.602 |
| uC Test5 | 802 (34%) | 325 (6%) | 1494 (31%) | 70 (50%) | 64 (2%) | 37.228 | 26.862 |
| uC Test6 | 908 (38%) | 349 (7%) | 1662 (35%) | 70 (50%) | 64 (2%) | 37.154 | 26.915 |
| uC Test7 | 716 (30%) | 296 (6%) | 1315 (27%) | 69 (49%) | 64 (2%) | 20.745 | 48.204 |
| uC Test6 & Ext.ROM | 899 (38%) | 340 (7%) | 1628 (34%) | 134 (95%) | 64 (2%) | 40.402 | 24.751 |

Perancangan yang keseluruhan instruksinya diletakkan pada ROM *internal* akan mengakibatkan perubahan pada hasil rancangan. Hal ini dikarenakan semua operasi yang akan dikerjakan sudah diketahui terlebih dahulu maka pada sintesis dan implementasinya, fungsi-fungsi utnuk operasi yang tidak dibutuhkan akan dibuang. Dari Tabel 4.22 dapat dilihat bahwa uC Test6 hanya menggunakan ROM *internal* sedangkan uC Test6 & Ext.ROM menggunakan gabungan ROM *internal* dan *external*.

Data mengenai hal ini juga dapat dilihat pada Tabel 4.23. Pada komponen function_unit, masukan yang diberikan belum diketahui sehingga dalam sintesis dan implementasinya, semua fungsi dari komponen function_unit akan dibangun. Sedangkan untuk komponen function_unit_input, dimana telah diberikan input berupa operasi

penjumlahan maka pada sintesis dan implementasinya, hanya fungsi untuk operasi penjumlahan yang akan dibangun. Hal ini diketahui melalui jumlah slice yang digunakan adalah sangat sedikit.

**Tabel 4.23** *Function Unit* dengan *Input*

| Implementation | | | | | | |
|---|---|---|---|---|---|---|
| **Komponen** | **Slice** | **FF** | **4 input LUT** | **IOB** | **T-BUF** | **Timing Report CPD** |
| function_ unit | 200 (8%) | --- | 380 (8%) | 104 (72%) | --- | 29.077 |
| function_ unit_input | 23 (0%) | --- | 44 (0%) | 100 (69%) | --- | 15.901 |

Karena pada perancangan implementasi untuk pengujian digunakan ROM *internal* seluruhnya maka ukuran *slice* dan frekuensi maksimum untuk masing-masing pengujian adalah berbeda. Hal ini dapat dilihat pada Tebel 4.22 yaitu pada komponen uC kosong, uC dengan 1 instruksi, uC Test 1 sampai dengan uC Test 7. Untuk mengetahui jumlah slice dan frekuensi maksimum yang dibutuhkan untuk keseluruhan perancangan dapat dilihat pada komponen risc (Tabel 4.10).

## 4.4 Perhitungan *Speed Up* untuk *Pipelining*

Dari data pada Subbab 2.2.1 dinyatakan bahwa 53% percabangan adalah diambil dan 47% percabangan tidak diambil. Pada perancangan yang telah dilakukan, percabangan yang diambil membutuhkan tiga *cycle* sedangkan percabangan tidak diambil hanya dibutuhkan satu *cycle*. Oleh karena itu maka rata-rata *cycle* yang dibutuhkan oleh operasi percabangan adalah sebagai berikut:

$$\text{BranchCycles} = (3 \times 53\%) + (1 \times 47\%) = 2.06\text{cycles}$$

Data pada Subbab 2.2.1 akan digunakan untuk membandingkan *cycle per instruction* (CPI) tanpa dan dengan *pipeline*. Tanpa *pipeline* (Tabel 4.24), dibutuhkan 1 CPI sedangkan dengan *pipeline* (Tabel 4.25), dibutuhkan 1,212 CPI.

**Tabel 4.24** CPI yang Dihasilkan Tanpa *Pipeline*

| Instuksi | | Cycle | CPI | % time |
|---|---|---|---|---|
| Aritmatika/Logika | 16% | 1 | 0.16 | 16% |
| Perpindahan Data | 33% | 1 | 0.33 | 33% |
| Percabangan Bersyarat | 20% | 1 | 0.20 | 20% |
| Lain-lain | 31% | 1 | 0.31 | 31% |
| Total | 100% | | 1 | |

**Tabel 4.25** CPI yang Dihasilkan Dengan *Pipeline*

| Instuksi | | Cycle | CPI | % time |
|---|---|---|---|---|
| Aritmatika/Logika | 16% | 1 | 0.16 | 13.2% |
| Perpindahan Data | 33% | 1 | 0.33 | 27.2% |
| Percabangan Bersyarat | 20% | 2.06 | 0.412 | 34.0% |
| Lain-lain | 31% | 1 | 0.31 | 25.6% |
| Total | 100% | | 1.212 | |

Dari Tabel 4.24, Tabel 4.25 dan rumus pada Subbab 2.11 didapatkan bahwa rata-rata CPI *stall* akibat *pipeline* adalah:

$$\text{CPI}_{\text{pipelined}} = \text{Ideal CPI} + \text{Average Pipeline Stall CPI}$$

$$\text{Average Pipeline Stall CPI} = \text{CPI}_{\text{pipelined}} - \text{Ideal CPI}$$

$$\text{Average Pipeline Stall CPI} = 1.212 - 1 = 0.212 \text{ CPI}$$

Untuk mendapatkan *speed up*, dibutuhkan parameter *cycle time unpipeline* (tanpa *pipeline*) yang dapat dilihat pada Tabel 4.26.

**Tabel 4.26** Laporan Implementasi untuk Prosesor RISC tanpa *Pipeline*

| Komponen | Slice | FF | 4 input LUT | IOB | T-BUF | Timing Report | |
|---|---|---|---|---|---|---|---|
| | | | | | | Prd. | Freq. |
| risc | 780 (33%) | 32 (1%) | 1478 (31%) | 140 (100%) | --- | 78.438 | 12.749 |

Dari Tabel 4.24, Tabel 4.25, Tabel 4.26, dan rumus pada Subbab 2.11 didapatkan *speed up* dengan *pipeline* 4 tahap adalah sebagai berikut:

$$\text{Speed Up} = \frac{\text{Ideal CPI} \times \text{Pipeline Depth}}{\text{Ideal CPI} + \text{Pipeline Stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$
$$= \frac{1 \times 4}{1 + 0.212} \times \frac{78.438}{40.354} = 3.3 \times 1.944$$
$$= 6.415 \text{ kali}$$

Idealnya, peningkatan kecepatan yang terjadi adalah 16 kali jika tidak terjadi *pipeline stall* CPI dan *cicle time pipeline* lebih kecil 4 kali dibandingkan *cycle time unpipeline*. Persentase perbandingan peningkatan yang terjadi adalah:

$$\% \text{ Speed Up} = \frac{6.415}{16} \times 100\% = 40.1\%$$