 ISSN: 1411-3201

IMPLEMENTASI FIELD PROGRAMMABLE GATE ARRAY DALAM PERANCANGAN ARITHMETIC-LOGIC UNIT DAN SHIFTER

Ferry Wahyu Wibowo

STMIK AMIKOM Yogyakarta
e-mail : ferrywahyuwibowo@scientist.com

Abstraksi

Paper ini membahas mengenai implementasi Field Programmable Gate Array (FPGA) untuk membuat Arithmetic-Logic Unit (ALU) merupakan core dari central processing unit (CPU). ALU terdiri dari dua fungsi, yaitu unit aritmetik dan unit logik. Pada paper ini unit arimetik berupa penjumlahan, pengurangan, perkalian, dan pembagian. Sedangkan unit logik berupa AND, OR, NOT, dan XOR. Rangkaian penggeser dalam paper ini juga diaplikasikan untuk melengkapi instruksi kode pemrograman secara perangkat lunak. Jalur pemilih unit aritmetik dan unit logik pada ALU menggunakan rangkaian multiplekser. Paper ini menggunakan teknik perancangan top-down yang dilakukan dengan cara menentukan komponen-komponen yang akan digunakan dalam implementasi dengan mengetahui fungsi dan kerja komponen tersebut, kemudian menentukan kode konfigurasinya dalam bentuk bahasa deskripsi perangkat keras, Very High Speed Integrated Circuit Hardware Description Language (VHDL). Implementasi dari kode VHDL dikonfigurasi ke dalam Field Programmable Gate Array (FPGA) Xilinx Spartan-3E untuk menentukan konsumsi komponen dan waktu tunda yang dihasilkan. Hasil yang didapatkan dari konfigurasi tersebut mengkonsumsi 26 slices, 50 4-masukan LUT, 17 I/O, dan 17 Bonded IO; sedangkan waktu tunda komponen yang dihasilkan dalam FPGA sebesar 11,413 ns.

Kata kunci :
Aritmetik, CPU, FPGA, Logik, Penggeser, VHDL

Pendahuluan

Perkembangan teknologi dalam perangkat keras (*hardware*) dan perangkat lunak (*software*) dalam bidang komputing harus mempunyai aspek ekonomis dan berdaya guna yang tinggi. Banyak inovasi yang dituangkan dalam berbagai aspek disiplin ilmu, namun dalam hal ini tidak terlepas dari mengembangkan dan menemukan suatu algoritma. Algoritma yang semakin kompleks dan efektif kinerjanya dalam dunia ilmu komputer, akan berdampak pada implementasi perancangan yang lebih baik. Gambaran sifat algoritma khususnya dalam perancangan teknologi *very large scale integration* (VLSI) menjadi hal yang penting dalam pengembangan dan inovasi bentuk *integrated circuit* (IC) untuk mengimplementasikan mikroprosesor sebagai *core* dalam *central processing unit* (CPU) ataupun mikrokontroler, terutama ketika digunakan untuk menentukan teknik kompile yang mengkonfigurasikan komponen penyimpan instruksi data, *random-access memory* (RAM), yang berbeda-beda sebagai penyimpan data pemrograman.

Komunikasi data yang dilakukan antar komponen mempunyai peranan penting dalam pengolahan instruksi dan pensinyalan. Penentuan jalur/*bus* dan aktivasi sinyal suatu komponen harus dilakukan dengan tepat, karena jika tidak tepat atau tidak sesuai maka kondisi yang dihasilkan akan mengakibatkan *error* termasuk penentuan tunda pewaktuannya, sehingga derau data digital (*glitch*) bisa diantisipasi dalam perancangan konfigurasinya [1]. Desain *arithmetic logic unit* (ALU) yang diimplementasikan dalam *field programmable gate array* (FPGA) mempunyai andil yang cukup besar dalam penentuan gerbang dan register yang dikonsumsinya. Pemilihan konfigurasi dan kode pemrograman mempunyai dampak yang cukup signifikan dalam menentukan konsumsi komponen yang digunakan, dalam hal ini mengabaikan teknik kompile dan penentuan penjalaran dan penempatan (*routing-and-placement*). Konfigurasi kode *very high speed integrated circuit hardware description language* (VHDL) dalam perancangan FPGA dalam beberapa perancangan tertentu mempunyai penggunaan dan penempatan komponen yang sama, seperti perancangan *read-only memory* (ROM) [2]. Metode perancangan *top-down* dan *bottom-up* dalam implementasi FPGA akan berpengaruh cukup signifikan dalam kemudahan dan kecepatan perancangan. Teknik perancangan *top-down* dilakukan dengan cara menentukan komponen-komponen yang akan digunakan dalam implementasi dengan mengetahui fungsi dan kerja komponen tersebut terlebih dahulu, kemudian menentukan kode konfigurasinya dalam bentuk bahasa deskripsi perangkat keras, seperti VHDL dan Verilog. Hal tersebut berlawanan dengan teknik perancangan *bottom-up* yang menggunakan bahasa deskripsi perangkat keras untuk membuat komponen-komponen yang diperlukan dalam perancangan, kemudian komponen-komponen tersebut diintegrasikan untuk konfigurasi FPGA.

 ISSN: 1411-3201

Banyak perancang implementasi FPGA lebih cenderung menggunakan teknik perancangan *top-down* daripada *bottom-up*, karena teknik ini relatif mudah dipahami dan dikonstruksikan [3]. Paper ini menggunakan VHDL sebagai bahasa deskripsi perangkat keras untuk konfigurasi FPGA. Bagi perancang FPGA, kemampuan mengembangkan kasus dan logika sinyal menjadi pokok penting dalam penentuan komponen rancangan dan penggunaan kode VHDL. Penerapan yang tepat implementasi kode VHDL untuk konfigurasi FPGA yang akan dihasilkan pun cukup erat terkait dengan pemakaian CPU pada FPGA itu sendiri [4]. Kode VHDL akan dikompile dengan berbagai tahapan, sehingga akan menghasilkan konsumsi komponen-komponen beserta waktu tundanya (*delay*), konsumsi gerbang-gerbang dasar dan registernya; maka hal tersebut relatif berkontribusi dalam perancangan [5,6].

FPGA dapat dengan mudah ditanamkan sistem mikrokontroler dengan mengkonsumsi komponen *logic block*. Mikrokontroler yang dibuat menggunakan FPGA lebih sedikit efisien daripada menggunakan IC mikrokontroler, namun juga lebih memudahkan memahami perancangan sistem mikrokontroler [7]. Penerapan ALU, pada dasarnya bervariasi tergantung *bus* / banyaknya jalur bit, fungsi aritmetika dan logik yang ingin diimplementasikan. Paper ini menggunakan 2x4-bit masukan utama, yang terdiri *bus* A dan B, ditambah 1-bit *carry* dan 5-bit pemilih (*selector*). *Bus* pemilih digunakan untuk memilih konfigurasi komponen yang akan diimplementasikan sebagai keluaran fungsi.

Metode Penelitian

Arithmetic Logic Unit (ALU) merupakan unit inti (*core*) dari *central processing unit* (CPU) yang terdiri dari rangkaian logik kombinasional yang dibangun dari komponen fungsi operasi aritmetika dan logik menggunakan operand dua masukan dalam perintah instruksi komputer, yaitu masukan A dan B. ALU menampilkan operasi penjumlahan, pengurangan, perkalian integer, logika AND, OR, NOT, XOR, dan operasi Boolean yang lain, selain itu ALU mempunyai masukan pemilih dari instruksi CPU yang dienkode untuk memilih komponen-komponen yang hendak dioperasikan dan ditampilkan data keluarannya. Jalur pemilih didekode menggunakan ALU yang menyesuaikan dengan operasinya sejumlah 2^n, dimana n merupakan masukan terenkode untuk pemilihan operasi yang diinginkan. Pemahaman mengenai bilangan digital diperlukan dalam perancangan ALU, misalkan jika suatu operasi aritmetik dituliskan sebagai Y<= A+/B+1, maka komplemen satu B dikurangi A dan ditambah 1. Hal ini sama dengan pengurangan B terhadap A, kemudian dijumlahkan dengan 1, atau dengan kata lain A-B+1.

Ada tiga cara merepresentasikan bilangan bertanda (*signed*), yaitu:

(1) Magnitudo *signed* merupakan cara representasi dengan menegasikan bilangan dengan menambahkan bit tanda di depan bilangan, dimana tanda bit 0 merepresentasikan bilangan positif, sedangkan tanda bit 1 merepresentasikan bilangan negatif. Misalkan bilangan $+11_{10}$ direpresentasikan sebagai 01011_2, sedangkan bilangan -11_{10} direpresentasikan sebagai 11011_2.

(2) Komplemen 1 (satu) merupakan cara representasi bilangan dengan mengkomplemen setiap bit bilangan. Misalkan bilangan $+11_{10}$ direpresentasikan sebagai 01011_2, sedangkan bilangan -11_{10} direpresentasikan sebagai 10100_2.

(3) Komplemen 2 (dua) merupakan cara representasi bilangan dengan menegasikan bilangan tersebut kemudian mengkomplemen setiap bit (mirip dengan komplemen 1) kemudian ditambahkan 1. Misalkan bilangan $+11_{10}$ direpresentasikan sebagai 01011_2, sedangkan bilangan -11_{10} direpresentasikan sebagai 10101_2.

Lebar bit ALU pada instruksi yang ditanganinya, biasanya sama dengan nama banyaknya bit prosesor dimana *bus* eksternal bisa jadi lebih sedikit, misalkan prosesor 4-bit, prosesor, 8-bit, prosesor 16-bit, prosesor 32-bit, prosesor 64-bit, dan seterusnya. Operasi *floating point* dilakukan oleh unit *floating-point*. Beberapa prosesor juga menggunakan ALU untuk mencacah alamat, seperti *increment* / *decrement* pada *program counter*, sedangkan beberapa prosesor juga mempunyai *logic* yang terpisah. Beberapa prosesor membagi ALU menjadi dua unit, yaitu unit aritmetik dan unit logik, selain itu beberapa prosesor juga membagi unit aritmetik menjadi operasi *fixed-point* dan operasi *floating-point*. Pada *personal computer* (PC) menggunakan operasi *floating point* kadang kala dilakukan oleh unit *floating-point* pada *chip* terpisah yang disebut *numeric coprocessor*. Biasanya ALU mempunyai masukan langsung dan akses keluaran pada pengendali prosesor, memori utama (random access memory (RAM) pada PC), dan piranti *input*/*output* (I/O). Masukan terdiri dari perintah instruksi (terkadang disebut sebagai *machine instruction word*) yang terdiri dari kode operasi (*operation code* (op-code)), operand, dan kode format (*format code*). Kode operasi menginstruksikan ALU untuk menampilkan operasi sedangkan *operand* digunakan dalam operasi tersebut, misalkan perintah ADD A,B mengindikasikan bahwa operasi yang dilakukan adalah operasi aritmetik penjumlahan, dimana A ditambah dengan B hasilnya disimpan pada *register* penyimpan, biasanya akumulator (*accumulator*) berupa register dimana fungsi biasanya sebagai *operand* A. Namun operand tersebut harus didefinisikan terlebih dahulu, apakah nilai operand tersebut bersifat *fixed-point* atau *floating-point*. Jika hasilnya sudah sesuai dengan inisialisasi datanya maka operasi tersebut akan berhasil dioperasikan, jika tidak maka status akan menunjukkan bahwa

 ISSN: 1411-3201

data akan disimpan dalam tempat permanen yang sering disebut sebagai *machine status word*. Aliran bit dan operasi yang ditampilkan pada ALU dikendalikan oleh rangkaian gerbang yang juga dikendalikan oleh urutan unit logik yang menggunakan algoritma untuk masing-masing kode operasi. Pada unit aritmetik, perkalian dan pembagian dilakukan oleh kombinasi dari operasi penjumlahan, pengurangan, dan pergeseran. Dari berbagai hal yang telah dipaparkan tersebut, desain ALU merupakan bagian yang penting dari prosesor dan pendekatan untuk mempercepat proses instruksi yang digunakan secara *software* selalu dikembangkan secara berkesinambungan.

Unit Logik

Unit logik terdiri dari logika NOT, AND, OR, NAND, NOR, XOR dan XNOR. Contoh format instruksi dari logika AND secara *software* dituliskan sebagai "AND, operand1, operand2". Sedangkan kode VHDL untuk membentuk komponen gerbang AND yang akan diimplementasikan pada FPGA dituliskan sebagai :

```
  LIBRARY ieee;
USE ieee.std_logic_1164.all;
  ENTITY gerbang_and IS
        PORT (a,b : IN BIT;
                x : OUT BIT);
END;
  ARCHITECTURE arsitektur OF gerbang_and IS
  BEGIN
        x <= a AND b;
END;
```

Variabel a, b, dan x pada contoh kode VHDL di atas merupakan jalur 1-bit. Sedangkan kalau menggunakan bus / jalur 4-bit maka penulisan kode VHDL dituliskan sebagai BIT_VECTOR(1 TO 4) atau BIT_VECTOR(4 DOWNTO 1). Kedua kode VHDL tersebut bedanya terletak pada penempatan urutan jalur-jalurnya saja.

Unit Aritmetik

Unit *aritmetic* terdiri dari penjumlahan, pengurangan, perkalian, dan pembagian. Tipe data yang dapat digunakan dalam operasi aritmetika dalam kode VHDL adalah *integer*, *signed*, *unsigned*, dan *real*. Sehingga jika paket std_logic_signed atau std_logic_unsigned dari library ieee, maka STD_LOGIC_VECTOR dapat digunakan dalam operasi penjumlahan dan pengurangan [8]. Operasi penjumlahan (*full adder*) dapat dibentuk dari kombinasi gerbang logik, sebagaimana dituliskan dalam kode VHDL sebagai berikut:

```
  LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  ENTITY penjumlah IS
  PORT ( a, b, cin: IN BIT;
          s,cout: OUT BIT);
  END;
  ARCHITECTURE aritmetik OF penjumlah IS
  BEGIN
    s <= a XOR b XOR cin;
    cout <= (a AND b) OR (a AND cin) OR (b
        AND cin);
  END;
```

Unit penjumlah selain mengkombinasikan logik juga dapat menggunakan *operator* aritmetik sebagaimana dituliskan dalam kode VHDL berikut:

```
  LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.std_logic_unsigned.all;
  ENTITY penjumlah IS
  PORT ( a, b : IN STD_LOGIC;
          c, d : IN INTEGER RANGE 0 TO 15;
          x : OUT STD_LOGIC;
          y : OUT INTEGER RANGE 0 TO 30);
  END;
  ARCHITECTURE tugas OF penjumlah IS
  BEGIN
        x <= a + b;
        y <= c + d;
END;
```

Penulisan ieee.std_logic_unsigned.all diperlukan untuk operasi penjumlahan (+) berbeda jika menggunakan gerbang logik. Tipe data *operand* yang digunakan juga harus sesuai antara masukan dan keluarannya, sebagaimana STD_LOGIC dan INTEGER, jika tidak sesuai maka dapat menggunakan kode VHDL konversi, CONV.

Unit Shifter

Register merupakan salah satu komponen yang penting, karena digunakan dalam implementasi komponen yang lebih kompleks, seperti *shift-register*. Komponen *shift-register* digunakan untuk menggeser data ketika dipicu oleh *clock*. *Register* dengan masukan data d, *clock* dan *reset* asinkron dituliskan dalam bentuk kode VHDL sebagai berikut:

```
  LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  ENTITY register_d IS
  PORT ( d, clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
  END;
  ARCHITECTURE arsitektur OF register_d IS
  BEGIN
        PROCESS (rst, clk)
        BEGIN
        IF (rst='1') THEN
                q <= '0';
        ELSIF (clk'EVENT AND clk='1') THEN
                q <= d;
        END IF;
```

 ISSN: 1411-3201

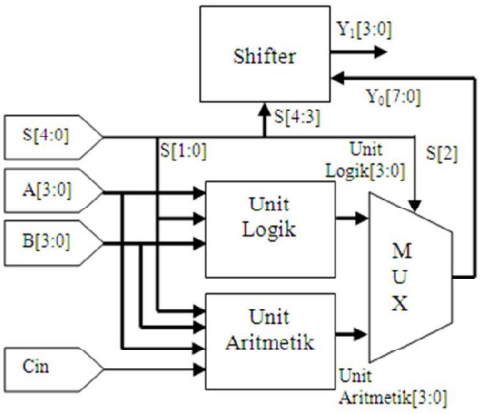
```
        END PROCESS;
    END;
```

Operator pergeseran (*shifter*) digunakan untuk menggeser bit data. Sintaksnya dituliskan sebagai *3left operand4* 3*shift operation*4 3*right operand*4. *Operand* kiri harus bertipe data BIT_VECTOR, sementara *operand* kanan harus berupa INTEGER yang tanda + atau – dapat ditempatkan didepannya. Operator geser dapat berupa SLL (*shift left logic*) dan SRL (*shift right logic*). Operator SLL mempunyai nilai ’0’ pada posisi yang berada di kanannya, sedangkan operator SRL mempunyai nilai ’0’ pada posisi yang berada di kirinya [8].

Unit Multiplekser

Unit multiplekser digunakan sebagai rangkaian pemilih data masukan yang akan dioperasikan sebagai data keluaran. Kode VHDL untuk rangkaian multiplekser dapat dituliskan sebagai berikut :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY muks IS
PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
   y: OUT STD_LOGIC);
END;
ARCHITECTURE tugas OF muks IS
BEGIN
```



Gambar1. Bagan Desain ALU

```
    y <=  a WHEN sel="00" ELSE
          b WHEN sel="01" ELSE
          c WHEN sel="10" ELSE
          d;
END;
```

Desain *arithmetic logic unit* (ALU) dan rangkaian penggeser (*shifter*) yang akan diimplementasikan dalam *field programmable gate array* (FPGA) Xilinx Spartan-3E mempunyai masukan 8-bit yang terdiri dari 4-bit masukan A dan 4-bit masukan B, 4-bit masukan *selection*, S, dan masukan *carry*, C_{in}. Masukan A dan B difungsikan sebagai *operand*, sedangkan masukan S difungsikan sebagai pemilih operasi pada komponen-komponen ALU. Komponen-komponen yang diimplementasikan dalam desain ALU ini terdiri dari unit logik, unit aritmetik, rangkaian multiplekser (MUX), dan rangkaian penggeser (*shifter*). Bagan desain ALU yang dibuat digambarkan menurut gambar 1.

Sebagaimana pada gambar 1 terdapat dua masukan 4-bit A dan B digunakan sebagai masukan untuk komponen unit logik dan unit aritmetik, sedangkan pemilih keluaran untuk komponen unit logik dan unit aritmetik digunakan multiplekser (MUX) yang mengimplementasikan 2-bit masukan pemilih yaitu S[1:0]. Unit aritmetik mempunyai jalur *carry* 1-bit, C_{in}. Keluaran unit aritmetik dan logik dirancang untuk menghasilkan 4-bit. Pemilih bit pada rangkaian multiplekser menggunakan jalur pemilih, S[2], dimana jika nilai S[2] = ‘0’ maka rangkaian multiplekser akan memilih keluaran dari unit aritmetik, sedangkan jika nilai S[2] = ‘1’ maka rangkaian multiplekser akan memilih keluaran dari unit logik. Sebagai masukan rangkaian penggeser (*shifter*) digunakan data dari *operand* A pada unit aritmetik pada fungsi pelewat (*transfer*), Nilai penggeser bit dari *bus* yang dibentuk terdiri dari dua kombinasi pemilih (*selector*) dan masukan *carry*, C_{in}. Nilai yang dikeluarkan dari multiplekser untuk fungsi *transfer* terdiri dari 4-bit masukan, sedangkan fungsi penggeser baik geser kiri (SHL) maupun kanan (SHR) ditentukan secara konfigurasi VHDL dengan nilai geser kiri, maka bit paling kanan bernilai referensi ‘0’ dan konfigurasi VHDL dengan nilai geser kanan, maka bit paling kiri bernilai referensi ‘0’. Rangkaian penggeser (*shifter*) dilengkapi dengan dua keadaan khusus, yaitu nilai transfer A dan nilai transfer ‘0’, selain menggeser data bit ke kiri maupun ke kanan. Jika nilai *selector* S[4:0] dan *carry* masukan, C_{in}, sama dengan “00000” maka unit rangkaian penggeser hanya melewatkan data bit dari *operand* A saja, sedangkan jika *selector*, S[4:3] bernilai “11” dan semua S[2:0] dan C_{in} bernilai ‘0’ semuanya maka data yang akan dikeluarkan bernilai “0000”. Fungsi dan kerja dari konfigurasi bit dapat dituliskan menurut tabel I.

 ISSN: 1411-3201

Tabel 1. Desain Operasi ALU dan Shifter

Unit Aritmetik							
S_4	S_3	S_2	S_1	S_0	C_{in}	Operasi	Fungsi
0	0	0	0	0	0	Y<=A	*Transfer* A
0	0	0	0	0	1	Y<=A+1	*Increment* A
0	0	0	0	1	0	Y<=A+ B	Penjumlahan
0	0	0	0	1	1	Y<=A+ B+1	Penjumlahan dengan *carry*
0	0	0	1	0	0	Y<=A+/ B	Penjumla-han A dengan komplemen B
0	0	0	1	0	1	Y<=A+/ B+1	Pengurangan
0	0	0	1	1	0	Y<=A-1	*Decrement* A
0	0	0	1	1	1	Y<=A	*Transfer* A
Unit Logik							
0	0	1	0	0	0	Y<=A and B	Fungsi AND
0	0	1	0	1	0	Y<=A or B	Fungsi OR
0	0	1	1	0	0	Y<=A xor B	Fungsi XOR
0	0	1	1	1	0	Y<=/A	Komplemen A
Unit Shifter							
0	0	0	0	0	0	Y<=A	*Transfer* A
0	1	0	0	0	0	Y<=shl A	Geser kiri A
1	0	0	0	0	0	Y<=shr A	Geser kanan A
1	1	0	0	0	0	Y<=0	*Transfer* 0

Tabel 1 dapat ditambahkan beberapa fungsi lagi, namun fungsi pemilih juga harus bertambah karena jalur pemilih sangat berperan penting untuk menentukan keluaran data.

Hasil Penelitian dan Pembahasan

Pada pendekatan desain *top-down*, empat modul aritmetik – penjumlahan, pengurangan, perkalian dan pembagian – telah diimplementasikan oleh [9] dengan mengkombinasikan fungsi aritmetik tersebut untuk membentuk unit ALU *floating point*. Setiap modul dibagi menjadi sub-modul. Dua pemilih bit dikombinasikan untuk memilih operand dalam desain ALU yang direalisasikan menggunakan VHDL. Fungsionalitas desain divalidasi melalui simulasi VHDL. Prinsip yang telah digunakan dalam mendesain ALU tersebut diterapkan juga di dalam paper ini. Namun yang membedakan dalam penerapannya terletak pada algoritma dan pemilihan fungsi perancangannya yang lebih kompleks. *Pipelining* tidak diaplikasikan dalam perancangan karena tidak mengindikasikan pemecahan blok ALU.

Logik balik diterapkan dalam ALU untuk mendapatkan optimisasi daya yang tinggi dengan menentukan logik kontrol yang sesuai pada salah satu penjumlah paralel, sehingga berbagai operasi aritmetik dapat direalisasikan [10]. Penggunaan komponen yang tepat mampu mempercepat waktu tunda yang dihasilkan oleh tunda logik dan tunda rute untuk membentuk modul, unit aritmetik yang diaplikasikan dalam paper ini menggunakan penjumlahan (+) bukan kombinasi dari logik yang ada. Namun pemakaian komponen FPGA Xilinx Spartan-3E kurang begitu optimal karena tidak mempunyai fitur *digital signal processing* (DSP). Bagan *register transfer logic* (RTL) yang dihasilkan dari FPGA digambarkan menurut gambar 2.

Tabel 2. Komponen ALU Dari FPGA SPARTAN-3E

Komponen	Jumlah	Total
Slices	26	4656
4-masukan *Look-Up-Table*	50	9312
Input/Output	17	
Bonded IOB	17	232

Konsumsi blok logik yang dikonsumsi oleh konfigurasi FPGA Xilinx Spartan-3E dituliskan sebagaimana pada tabel 2.

Waktu tunda yang dihasilkan oleh komponen logik sebesar 8,830 ns dan tunda rute sebesar 2,583 ns.

 ISSN: 1411-3201

Sehingga total waktu tunda yang dihasilkan sebesar 11,413 ns.

Daftar Pustaka

[1] F.W. Wibowo, "Interoperability of Reconfiguring System on FPGA Using a Design Entry of Hardware Description Language," *Proc. of International Conference on Advances in Computing, Control, and Telecommunication Technologies 2011*, pp. 79-83, 2011.

[2] F.W. Wibowo, "The Conservative Structure of Synthesizing Read Only Memory Using VHDL on FPGA," *Proc. of 4th International Seminar on Industrial Engineering and Management (4th ISIEM)*, pp. 231-235, 2010.

[3] F.W. Wibowo, "System on Chip untuk Mesin Pengepakan Barang Berbasis FPGA," *Pros. Seminar Teknik Informatika 2011*, pp. B-52-B-59, 2011.

[4] F.W. Wibowo, "Desain dan Implementasi Convolutional Encoder (2, 1, 8) dalam Field Programmable Gate Array (FPGA), *Jurnal Rekayasa Elektrika*, vol. 9, no. 4, pp. 166-170, Oktober 2011.

[5] F.W. Wibowo, "Mergesort pada tingkat Register Transfer Logic Berbasis FPGA," *Journal DASI*, vol. 11, no. 4, pp. 33-47, Desember 2010.

[6] F.W. Wibowo, "Finite State Machine untuk Pengendali Elevator Berbasis Field Programmable Gate Array," *Journal DASI*, vol. 12, no. 1, pp. 6-8, Maret 2011.

[7] S.R. Ball, *Embedded Microprocessor Systems*: *Real World Design*, Third Edition, USA: Newnes, 2002.

[8] V.A. Pedroni, *Circuit Design with VHDL*, London: MIT Press, 2004.

[9] S. Purnima, T. Mukesh, S. Jaikaran, R. Sanjay, "VHDL Environment for Floating Point Arithmetic Logic Unit – ALU Design and Simulation," *Research Journal of Engineering Sciences*, vol. 1(2), pp. 1-6, Agustus 2012.

[10] A. Dixit, V. Kapse, "Arithmetic & Logic Unit (ALU) Design Using Reversible Control Unit," *International Journal of Engineering and Innovative Technology* (IJEIT), Vol. 1, Issues 6, Juni 2012.