Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.202296755
Vol. 9, No. 7, Desember 2022, hlm. 1617-1622 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

IMPLEMENTASI KOMPUTASI AKAR KUADRAT RESOLUSI TINGGI PADA FIELD PROGRAMMABLE GATE ARRAY (FPGA)

Muhammad Irfan[1], Hendra Setiawan*[2]

[1,2]Universitas Islam Indonesia, Daerah Istimewa Yogyakarta
Email: [1]16524100@students.ac.id, [2]hendra.setiawan@uii.ac.id
*Penulis Korespondensi

(Naskah masuk: 07 Desember 2022, diterima untuk diterbitkan: 26 Desember 2022)

Abstrak

Komputasi akar kuadrat diperlukan pada beberapa proses pengendalian, diantaranya untuk Direct Torque Control (DTC) pada sistem penggerak motor yang membutuhkan proses perhitungan yang sangat cepat. Field Programmable Gate Array (FPGA) merupakan salah satu perangkat yang dapat digunakan untuk implementasi komputasi yang memerlukan kecepatan dan presisi tinggi. Penerapan komputasi akar kuadrat pada FPGA menggunakan metode digit by digit non-restoring dengan beberapa modifikasi agar memiliki hasil perhitungan dengan nilai error yang kecil. Sistem tersebut diimplementasikan menggunakan 32-bit input dan 16-bit output. Proses perhitungan melibatkan Finite State machine (FSM) untuk menghemat resource yang diperlukan. Proses verifikasi sistem dilakukan dalam dua tahap, yaitu verifikasi fungsional dengan aplikasi ModelSim-Altera dan verifikasi hardware menggunakan modul FPGA Cylcone IV EP4CE6E228N. Hasil verifikasi menunjukkan bahwa hasil perhitungan akar kuadrat memiliki resolusi sampai dengan 0,0039. Selain itu, sistem ini membutuhkan 157 Logic Elements dan 120 register dengan kecepatan clock tertinggi yang dicapainya adalah 205 MHz untuk input 32 bit

Kata kunci: *Akar kuadrat, FPGA, FSM, Logic Element*

IMPLEMENTATION OF HIGH RESOLUTION SQUARE ROOT COMPUTING ON FIELD PROGRAMMABLE GATE ARRAY (FPGA)

Abstract

Square root computing is required in several control processes, such as for Direct Torque Control (DTC) on motor drive systems that require a very fast calculation process. Field Programmable Gate Array (FPGA) is one of the devices that recommended for high speed and precision computation. The implementation of the square root on the FPGA uses the digit-by-digit non-restoring method with some modifications to get a high precision of computation result. The system is implemented using 32-bit input and 16-bit output. The calculation process involves a Finite State machine (FSM) to minimize computation resources. The system verification process is carried out in two stages, i.e. functional verification using the ModelSim-Altera and hardware verification using the FPGA Cylcone IV EP4CE6E228N. The verification shows that the result of the square root calculation has a resolution of up to 0.0039. In addition, the system requires 157 Logic Elements and 120 registers with the highest clock speed can achieves 205 MHz for 32-bit input.

Keywords: *FPGA; FSM; Logic Element; Square root*

1. PENDAHULUAN

Sistem kendali selalu melibatkan proses komputasi mulai dari yang sederhana sampai ke perhitungan komputasi kompleks. Proses komputasi sederhana mungkin hanya melibatkan sebuah proses penjumlahan/pengurangan antara set point dengan actual value yang dengan mudah dapat diimplementasikan ke dalam perangkat keras seperti *operational amplifier* atau mikrokontroler. Namun ada beberapa proses komputasi yang memerlukan suatu perhitungan yang cukup kompleks apabila diimplementasikan ke dalam perangkat keras. Salah satunya adalah perhitungan akar kuadrat.

Perhitungan akar kuadrat digunakan dalam beberapa proses komputasi pengendalian diantaranya perhitungan fluks stator pada *Direct Torque Control* (DTC) (SUTIKNO dkk, 2012), proses PID dengan kompensasi akar kuadrat (REYES dkk, 2013), dan

pengendali ketinggian pada *drone/quadcopter* (XUAN-MUNG dan HONG, 2019). Selain itu perhitungan akar kuadrat juga dilibatkan dalam beberapa perhitungan modern misalkan untuk modulasi digital (CHEN dkk, 2012), *channel estimation* (SALMELA dkk, 2006, SALMELA dkk., 2011) *singular-value decomposition* (MARKOVIC dkk, 2007) dan *matrix inversion* (MAHAPATRA, 2012).

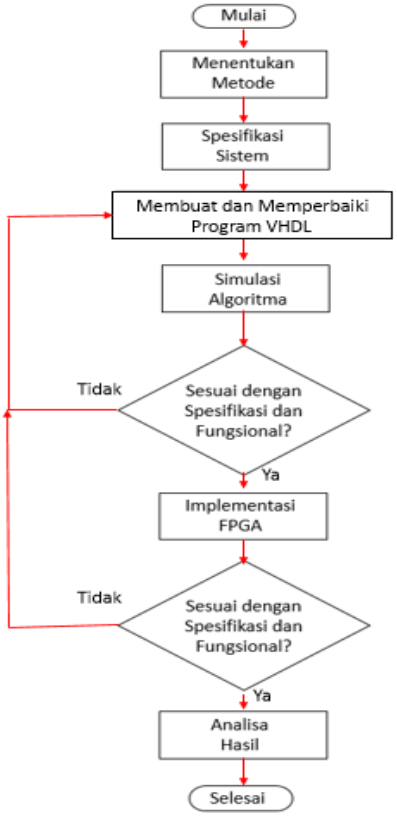
Dalam tataran implementasi, perhitungan akar kuadrat dapat dilakukan dengan perangkat mikrokontroler, *Digital Signal Processor* (DSP), maupun *Field Programmable Gate Array* (FPGA). Namun kendala utama dalam implementasi adalah proses perhitungan harus dilakukan dengan sangat cepat dan mampu bekerja pada frekuensi yang tinggi (SUTIKNO dkk, 2011a). Dengan mempertimbangkan beberapa keunggulan yang dimilikinya (FRANCES-VILLORA dkk, 2016), implementasi sebuah algoritme dengan menggunakan FPGA menjadi pilihan yang tepat.

Implementasi perhitungan akar kuadrat di FPGA yang telah dilakukan pada penelitian sebelumnya hanya berlaku pada bilangan bulat dan tanpa melibatkan bilangan di belakang koma. Hal ini menyebabkan nilai kesalahan yang cukup besar terutama pada bilangan di bawah sepuluh. Sebagai contoh perhitungan akar kuadrat dari bilangan desimal tiga (‘3’) akan diperoleh hasil satu (‘1’) yang seharusnya 1,732…, sehingga nilai kesalahan mencapai 42%. Sedangkan hasil komputasi dengan presisi yang tinggi diperlukan dalam proses pengendalian untuk meningkatkan stabilitas sistem yang dikendalikan. Sehingga, tujuan utama dalam penelitian ini adalah mendapatkan implementasi perhitungan akar kuadrat dengan kecepatan proses dan presisi yang tinggi.

2. METODE PENELITIAN

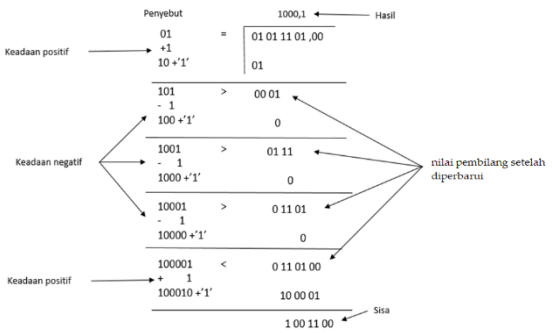
Proses perancangan dalam penelitian ini ditunjukkan pada Gambar 1. Adapun spesifikasi rancangan adalah: (1) input terdiri dari 32-bit *fixed point* dengan 16 bit di belakang koma, (2) output terdiri dari 16 bit *fixed point* dengan 8bit di belakang koma, dan (3) kecepatan *clock* minimal 20MHz (rekomendasi JIDIN dkk, 2016).

Metode yang digunakan dala perhitungan akar kuadrat adalah modified *digit by digit non-restoring*. Tahapan yang dilakukan pada metode ini dijelaskan sebagai berikut: pertama angka yang akan dihitung akarnya (pembilang) dikelompokkan dalam beberapa kelompok yang masing-masing berisi 2 bit. Selanjutkan dilakukan proses pembandingan antara nilai pembilang (dimulai dari yang paling kiri) dengan nilai penyebut yang awalnya bernilai “01”. Jika nilai pembilang sama atau lebih besar dari penyebut maka angka ‘1’ akan tercatat di atas sebagai hasil sementara.



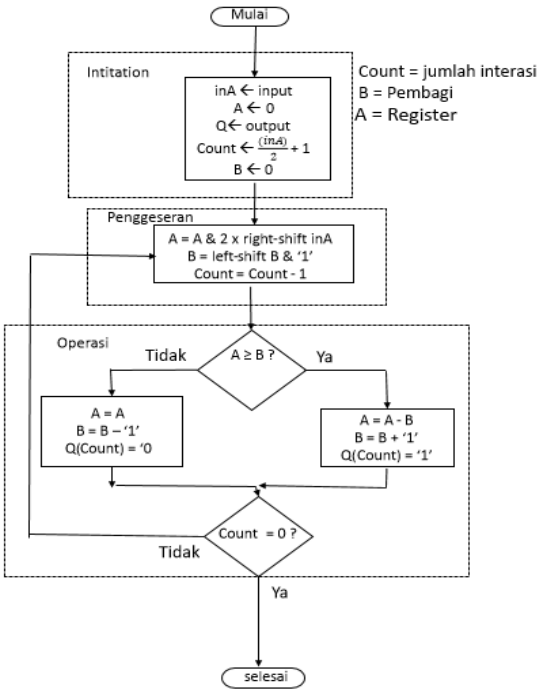
Gambar 1. Proses implementasi akar kuadrat pada FPGA

Sebaliknya, jika nilai pembilang lebih kecil dari penyebut maka angka ‘0’ akan tercatat di atas sebagai hasil sementara. Selanjutnya dilakukan proses pembaruan nilai pembilang dan penyebut. Nilai pembilang yang baru adalah hasil pengurangan antara pembilang yang lama dengan nilai perkalian hasil sementara (di bagian atas) dengan penyebut. Sedangkan pembaruan nilai penyebut dilakukan dengan menambahkan ‘1’ pada bit paling kanan atau menyisipkan bit ‘0’ sesuai hasil perbandingan yang dilakukan sebelumnya. Ilustrasi perhitungan ini ditunjukkan pada Gambar 2. Pada Gambar 2 tersebut perhitungan akar kuadrat dilakukan pada bilangan 93 dengan nilai biner 01011101,0. Hasil perhitungan diperoleh hasil 9,64 atau nilai biner 1001,10… Panjang bit setelah koma disesuaikan dengan jumlah bit yang diinginkan.

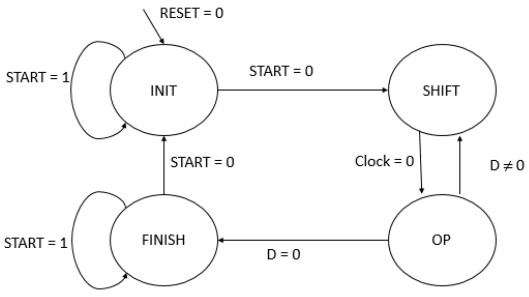


Gambar 2. Ilustrasi perhitungan akar kuadrat dengan *modified digit by digit non-restoring*

Untuk implementasi proses perhitungan yang telah dijelaskan sebelumnya, dibuat sebuah diagram alir sebagaimana ditunjukkan pada Gambar 4. Secara umum terdapat tiga (3) proses utama yaitu: inisialisasi, pergeseran, dan operasi perbandingan. Berdasarkan diagram alir tersebut dibuat sebuah *Finite State Machine* (FSM) yang melibatkan beberapa variable input dan terdiri dari empat (4) state, yaitu INIT, SHIFT, OP, dan FINISH. Rincian diagram FSM ditunjukkan pada Gambar 3. Berdasarkan FSM inilah, dilakukan pemrograman FPGA dengan menggunakan Bahasa VHSIC *Hardware Description Language* (VHDL).



Gambar 3. Diagram alir poses perhitungan akar kuadrat



Gambar 4. Diagram FSM

Struktur rancangan dibuat dalam dua kelompok besar, yaitu bagian kontrol (berbasis pada *state machine*) dan bagian pengolahan data. Gambar 4 memperlihatkan potongan *coding* VHDL untuk implementasi bagian kontrol. Sedangkan, pengolahan data dipecah dalam beberapa process tergantung pada masing-masing *state*.

Proses selanjutnya adalah melakukan verifikasi terhadap rancangan yang telah diimplementasikan dalam bahasa VHDL. Verifikasi dilakukan melalui dua tahap, yaitu verifikasi fungsional dan verifikasi di level perangkat keras (*hardware*). Verifikasi fungsional dilakukan dengan perangkat *software* ModelSIM, sedangkan verifikasi *hardware* menggunakan FPGA Cylcone IV EP4CE6E228N.

```
21      type states is (INIT, SHIFT, OP, FINISH1);
52      process (currentstate, start, D)
53      begin
54          case currentstate is
55          when INIT =>
56              if (start = '1') then
57                  nextstate <= INIT;
58              else
59                  nextstate <= SHIFT;
60              end if;
61          when SHIFT =>
62              nextstate <= OP;
63          when OP =>
64              if (D = 0) then
65                  nextstate <= FINISH1;
66              else
67                  nextstate <= SHIFT;
68              end if;
69          when FINISH1 =>
70              if (start = '0') then
71                  nextstate <= INIT;
72              else
73                  nextstate <= FINISH1;
74              end if;
75          when others =>
76              nextstate <= INIT;
77          end case;
78      end process;
79
```

Gambar 5. Implementasi FSM di VHDL

3. TINJAUAN PUSTAKA

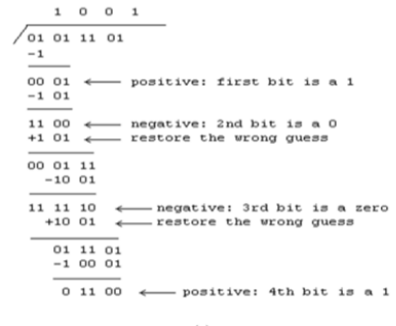
3.1. Hasil Penelitian Sebelumnya

Perhitungan akar kuadrat dengan menggunakan kalkulator dapat dengan mudah dilakukan. Namun bagaimana implementasi perhitungan tersebut dalam perangkat digital. Ada beberapa algoritme yang dapat digunakan untuk implementasi akar kuadrat, diantaranya algoritme *Rough estimation*, metode *Babylonian*, *exponential identity*, algoritme *Taylor-series expansion*, metode *Newton-Raphson*, metode *Sweeney Robertson Tocher redundant*, metode *non redundant*, dan algoritme *sequential (digit-by-digit method)* (SUTIKNO dkk, 2011a). Pada penelitian ini digunakan metode digit by digit *non restoring* yang dimodifikasi karena metode tersebut memiliki struktur yang sederhana sehingga sangat dimungkinkan menghasilkan komputasi yang lebih cepat dengan menggunakan sumber daya *hardware* yang lebih kecil. Metode yang sejenis juga pernah dilakukan oleh (SAMAVI dkk, 2003; SUTIKNO, 2011b; JIDIN dkk, 2016). Penelitian SAMAVI dkk, 2003, mengimplementasikan perhitungan akar kuadrat untuk masukan 4, 8 dan 16 bit dengan diperoleh maksimal delay process 6,485ns, 10,023ns dan 32,016ns. Sedangkan penelitian SUTIKNO, 2011b, mengimplementasikan proses perhitungan 32 bit dan 64 bit dengan hasil secara berurutan memerlukan 256 *logic elements* (LEs) dan 1023 *logic elements* (LEs). Implementasi yang dilakukan oleh JIDIN dkk, 2016, untuk perhitungan 62 bit dengan kebutuhan *resource* sebesar 281 LEs dan mampu bekerja sampai dengan frekuensi clock 81MHz.

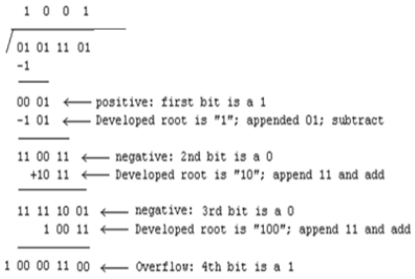
3.2. Metode Digit by Digit

Salah satu metode yang biasa digunakan adalah digit by digit. Pada metode ini, digit yang akan dihitung nilai akarnya dikelompokkan dalam beberapa kelompok yang masing-masing terdiri dari dua digit. Selanjutnya dilakukan proses pengurangan dengan nilai '1'. Jika hasil pengurangan adalah nilai negatif maka angka dikembalikan (*restore*) seperti sebelum proses pengurangan dan hasil digit '0' dituliskan di bagian atas. Sedangkan apabila hasil pengurangan masih bernilai positif maka masil digit '1' dituliskan di bagian atas dan lanjut ke kelompok digit selanjutnya. Proses tersebut berulang sampai ke digit terkecil yang diinginkan.

Dalam perkembangannya metode *digit by digit* dapat dibagi menjadi dua yaitu dengan restoring dan *non-restoring*. Gambar 1 menunjukkan contoh proses perhitungan akar kuadrat metode *digit by digit* dengan *restoring* (a) dan *non-restoring* (b).



(a)



(b)

Gambar 6. Contoh perhitungan akar kuadrat *digit by digit* (a) algoritme *restoring* (b) algoritme *non-restoring* (SUTIKNO, 2011b)

4. HASIL DAN PEMBAHASAN

4.1. Hasil Verifikasi Fungsional

Verifikasi fungsional dilakukan untuk memastikan proses perhitungan dapat berjalan dengan benar dalam kondisi ideal. *Software* ModelSIM digunakan dalam proses verifikasi fungsional ini. Verifikasi dilakukan dengan memberi nilai masukan desimal 1, 2, 3, 4, 5, 6, 7, 8, 9, 22, 39, 72, 137, 265, 534, 1063, 2120, 4233, 8201, 16406, 32807, dan 65535,99609375. *Screenshot* hasil verifikasi fungsional ditunjukkan pada Gambar 7 dengan output hasil perhitungan ditunjukkan pada sinyal paling bawah (sinyal "O").

Selanjutnya dilakukan perhitungan kesalahan hasil simulasi yang diperoleh dibandingkan dengan nilai yang seharusnya sebagaimana ditunjukkan dalam Tabel 1. Berdasarkan Tabel 1, diperoleh nilai rata-rata selisih sebesar 0,001545 dan nilai *Mean Squared Error* (MSE) sebesar $3{,}91\times10^{-6}$. Nilai selisih terbesar diperoleh 0,00389862 yang terjadi saat input bernilai '1' untuk semua bit-nya. Nilai ini sesuai dengan nilai kesalahan karena lebar bit yang terbatas. Untuk 8-bit dibelakang koma, maka nilai terkecil setara dengan $1/2^8 = 0{,}00390625$.

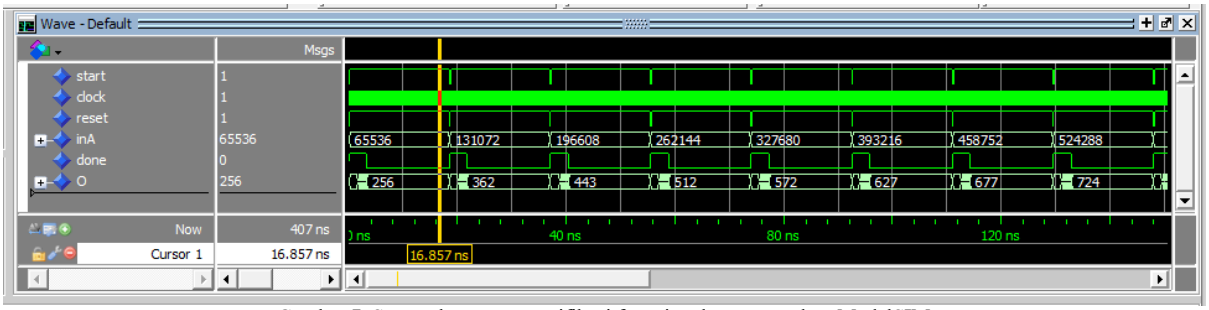
Tabel 1. Perbandingan hasil simulasi fungsional dengan nilai seharusnya

Nilai Input Simulasi	Output Simulasi	Output Seharusnya	Hasil Selisih
1,0	1,00000000	1,00000000	-
2,0	1,41406250	1,41421356	0,00015106
3,0	1,73046875	1,73205081	0,00158206
4,0	2,00000000	2,00000000	-
5,0	2,23437500	2,23606798	0,00169298
6,0	2,44921875	2,44948974	0,00027099
7,0	2,64453125	2,64575131	0,00122006
8,0	2,82812500	2,82842712	0,00030212
9,0	3,00000000	3,00000000	-
22	4,68750000	4,69041576	0,00291576
39	6,24218750	6,24499800	0,00281050
72	8,48437500	8,48528137	0,00090637
137	11,70312500	11,70469991	0,00157491
265	16,27734375	16,27882060	0,00147685
534	23,10546875	23,10844002	0,00297127
1063	32,60156250	32,60368077	0,00211827
2120	46,04296875	46,04345773	0,00048898
4233	65,05859375	65,06150936	0,00291561
8201	90,55859375	90,55937279	0,00077904
16406	128,08203125	128,08590867	0,00387742
32807	181,12500000	181,12702725	0,00202725
65535.99609375	255,99609375	255,99999237	0,00389862

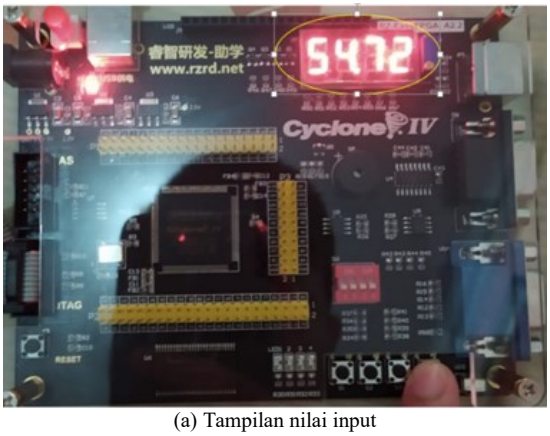
4.2. Implementasi FPGA

Setelah verifikasi fungsional dilakukan dan tidak ditemukan permasalahan, selanjutnya dilakukan proses impelemntasi ke FPGA. FPGA yang digunakan adalah Cylcone IV EP4CE6E228N dengan jumlah *Logic Elements* (LEs) 6272, 91 *port* I/O, dan frekuensi *clock* yang tersedia 200MHz. Hasil kompilasi dan sintesis di FPGA, diperoleh bahwa rancangan yang dibuat memerlukan 157 LEs, 120 register, dan 52 pin I/O. Selain itu, hasil *timing analysis* menunjukkan bahwa sistem yang dibangun mampu bekerja sampai dengan frekuensi 205MHz

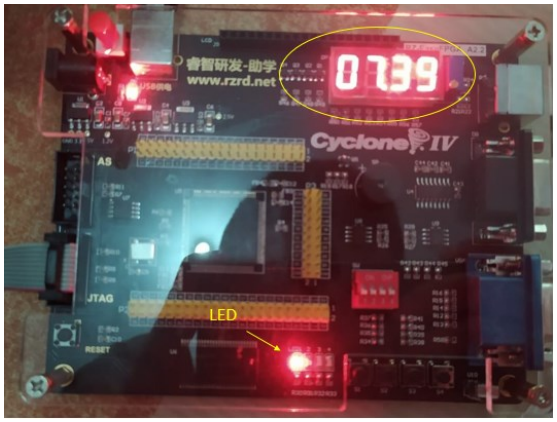
Verifikasi di level perangkat keras dilakukan untuk semakin menyakinkan bahwa tidak ada masalah dalam desain yang telah dibuat. Sebagai perangkat input digunakan push button yang berfungsi untuk mengatur nilai masukan yang diberikan ke penghitung akar kuadrat. Nilai input dan output hasil perhitungan ditampilkan ke penampil 7-segmen. Karena jumlah 7-segmen hanya empat buah, maka dilakukan *multiplexing* menggunakan *push button*. Ilustrasi verifikasi di level perangkat keras ditunjukkan pada Gambar 8 dengan (a) menunjukkan nilai masukan yang akan dihitung, dan (b) menunjukkan nilai hasil perhitungan akar kuadrat.



Gambar 7. Screenshot proses verifikasi fungsional menggunakan ModelSIM



(a) Tampilan nilai input



(b) Tampilan nilai hasil

Gambar 8. Verifikasi sistem di board FPGA

4.3. Unjuk Kerja

Selanjutnya dilakukan perbandingan hasil implementasi FPGA yang telah dilakukan dengan penelitian sejenis yang sudah ada. Modifikasi jumlah bit ke 32bit dan 64bit dilakukan agar setara dengan desain yang dilakukan oleh peneliti lainnya. Hasil perbandingan dari sisi jumlah LEs ditunjukkan di Tabel 2, sedangkan perbandingan *critical path* disajikan dalam Tabel 3. Dari Tabel 2, nampak bahwa rancangan yang dibuat dalam penelitian ini memerlukan LEs yang lebih sedikit dibandingkan dengan dua penelitian sebelumnya. Rancangan ini juga memerlukan register sebanyak 120 buah.

Sedangkan dari sisi kecepatan, hasil rancangan penelitian ini memberikan nilai *critical path* sebesar 8,35ns untuk input 64-bit dan 4,86ns untuk input 32-bit. Nilai tersebut lebih kecil dibandingkan dengan penelitian sebelumnya. Dengan nilai *critical path* tersebut, hasil rancangan yang dibuat dapat diberi *clock* dengan kecepatan maksimal 119MHz untuk input 64-bit dan 205MHz untuk input 32-bit. Hasil tersebut lebih baik dibandingkan penelitian serupa sebagaimana ditunjukkan dalam Tabel 3.

Tabel 2. Perbandingan hasil simulasi fungsional dengan nilai seharusnya

No	Penelitan oleh	Penggunaan LEs		
		64-bit	62-bit	32-bit
1	SUTIKNO, 2010	1023	-	256
2	JIDIN, 2016	-	281	-
3	Penelitian ini	258	-	157

Tabel 3. Perbandingan critical path pada implementasi di FPGA

No	Metode yang digunakan	*Critical path*	
		64-bit	32-bit
1	SAMAVI, 2003	N/A	32,02ns
2	Simple X Module (SAMAVI, 2008)	214ns	100ns
3	SAJID, 2012	N/A	3.5ns
4	ANANTHALAKSHMI, 2017	N/A	8ns
5	JIDIN, 2016	12,34ns	N/A
6	Penelitian ini	8,35ns	4,86ns

5. KESIMPULAN

Dalam penelitian ini berhasil dibuat rancangan sistem penghitung akar kuadrat yang melibatkan bilangan di belakang koma. Untuk input 32-bit, resolusi output diperoleh sampai dengan 8-bit di belakang koma dan nilai MSE $3{,}91\times10^{-6}$. Hasil verifikasi fungsional menunjukkan tidak ada masalah yang ditemukan dalam hasil perhitungan yang diperoleh. Rancangan yang dibuat juga berhasil diimplementasikan dalam perangkat FPGA Cylcone IV EP4CE6E228N dengan LEs sebanyak 258 untuk input 64-bit dan 157 untuk input 32-bit. Nilai tersebut lebih kecil dari hasil penelitian sejenis yang telah dilakukan sebelumnya. Ditinjau dari critical path, rancangan yang dibuat memiliki nilai critical path 8,35ns untuk input 64-bit dan 4,86ns untuk input 32-bit. Dengan nilai critical path tersebut, clock maksimal yang dapat diterapkan untuk rancangan ini adalah 205MHz untuk input 32-bit.

DAFTAR PUSTAKA

ANANTHALAKSHMI, A.V. & SUDHA, G.F., 2017. Design of a reversible floating-point square root using modified non-restoring

algorithm. Microprocessors and Microsystems, 50, pp.39-53.

CHEN, Y.L., ZHAN, C.Z., JHENG, T.J. & WU, A.Y., 2012. Reconfigurable adaptive singular value decomposition engine design for high-throughput MIMO-OFDM systems. IEEE transactions on very large scale integration (VLSI) systems, 21(4), pp.747-760.

FRANCES-VILLORA, J.V., ROSADO-MUÑOZ, A., MARTÍNEZ-VILLENA, J. M., BATALLER-MOMPEAN, M., GUERRERO, J.F. and WEGRZYN, M., 2016. Hardware implementation of real-time Extreme Learning Machine in FPGA: analysis of precision, resource occupation and performance. Computers & Electrical Engineering, 51, pp.139-156.

JIDIN, A.Z., MOHAMAD, S.H., AHMAD, S., YAKUB, M.F., AZLAN, N.A.N. & JIDIN, A., 2016, November. Optimizing the flux and torque estimator of DTC in FPGA by using low-area square root calculator. In 2016 IEEE International Conference on Power and Energy (PECon) (pp. 517-521). IEEE.

MAHAPATRA, C., MAHBOOB, S., LEUNG, V.C. & STOURAITIS, T., 2012, December. Fast inverse square root-based matrix inverse for MIMO-LTE systems. In 2012 International Conference on Control Engineering and Communication Technology (pp. 321-324). IEEE.

MARKOVIC, D., NIKOLIC, B. & BRODERSEN, R.W., 2007. Power and area minimization for multidimensional signal processing. IEEE Journal of Solid-State Circuits, 42(4), pp.922-934.

REYES, F., CID, J., LIMON, M.A. & CERVANTES, M., 2013. Square root-type control for robot manipulators. International Journal of Advanced Robotic Systems, 10(1), p.39.

SAJID, I., AHMED, M.M. & ZIAVRAS, S.G., 2012. Novel pipelined architecture for efficient evaluation of the square root using a modified non-restoring algorithm. Journal of Signal Processing Systems, 67(2), pp.157-166.

SALMELA, P., HAPPONEN, A., JARVINEN, T., BURIAN, A. & TAKALA, J., 2006, June. DSP implementation of Cholesky decomposition. In Joint IST Workshop on Mobile Future, 2006 and the Symposium on Trends in Communications. SympoTIC'06. (pp. 6-9). IEEE.

SALMELA, P., BURIAN, A., JÄRVINEN, T., HAPPONEN, A. & TAKALA, J.H., 2011. Low-complexity inverse square root approximation for baseband matrix operations. International Scholarly Research Notices, 2011.

SAMAVI, S., SADRABADI, A. & KAKLAR, R.Z., 2003. Improving the Array Structure of a Non-Restoring Square Root Circuit. International Journal Of Engineering Science (English), 14(1), pp.1-15.

SAMAVI, S., SADRABADI, A. & FANIAN, A., 2008. Modular array structure for non-restoring square root circuit. Journal of Systems Architecture, 54(10), pp.957-966.

SUTIKNO, T., 2010. An optimized square root algorithm for implementation in FPGA hardware. TELKOMNIKA (Telecommunication Computing Electronics and Control), 8(1), pp.1-8.

SUTIKNO, T., IDRIS, N.R.N., JIDIN, A.Z. & DAUD, M.Z., 2011a, April. FPGA based high precision torque and flux estimator of direct torque control drives. In 2011 IEEE Applied Power Electronics Colloquium (IAPEC) (pp. 122-127). IEEE.

SUTIKNO, T., 2011b. An efficient implementation of the non-restoring square root algorithm in gate level. International journal of computer theory and engineering, 3(1), p.46.

SUTIKNO, T., IDRIS, N.R.N., JIDIN, A. & CIRSTEA, M.N., 2012. An improved FPGA implementation of direct torque control for induction machines. IEEE Transactions on Industrial Informatics, 9(3), pp.1280-1290.

XUAN-MUNG, N. & HONG, S.K., 2019. Improved altitude control algorithm for quadcopter unmanned aerial vehicles. Applied sciences, 9(10), p.2122.