Ujian Akhir Semester
Mata Kuliah Deskripsi Perangkat Keras / Desain FPGA
Laporan Lab 11- Perancangan Multiplier 3-bit x 3-bit dengan Algoritma Geser dan Jumlah (*Shift-and-Add*)

Dosen Pengampu:

Adharul Muttaqin, S.T., M.T.

Disusun Oleh:

Reza Shah Akhtar 225060301111005

Departemen Teknik Elektro

Fakultas Teknik

Universitas Brawijaya

Malang

2025

Daftar Isi

Bab 1 Pendahuluan

1.1 Latar Belakang

Dalam dunia komputasi dan sistem digital, operasi aritmetika merupakan fondasi utama dari hampir semua proses pengolahan data. Di antara operasi dasar seperti penjumlahan dan pengurangan, operasi perkalian memegang peranan krusial dalam aplikasi yang lebih kompleks, seperti pada unit pemrosesan sinyal digital (DSP), grafika komputer, dan berbagai aplikasi ilmiah. Implementasi operasi perkalian secara efisien pada level perangkat keras sangat penting untuk mencapai kinerja sistem yang tinggi.

Laporan ini berfokus pada perancangan sebuah rangkaian digital spesifik, yaitu pengali biner (*binary multiplier*). Rangkaian ini dirancang untuk mengalikan dua buah bilangan biner berukuran 3-bit dan menghasilkan produk 6-bit, menggunakan salah satu metode yang paling fundamental, yaitu algoritma *shift-and-add*.

1.2 Tujuan

Tujuan dari penyusunan laporan ini adalah sebagai berikut:

1. Merancang arsitektur unit pemroses data (*datapath*) dan unit kontrol (*control unit*) untuk sebuah pengali biner 3-bit x 3-bit.
2. Mengembangkan diagram *Algorithmic State Machine* (ASM) untuk logika kontrol.
3. Mengimplementasikan keseluruhan desain perangkat keras menggunakan bahasa deskripsi VHDL.

Bab 2 tinjauan pustakah

2.1 Algorithmic State Machine (ASM)

Diagram **Algorithmic State Machine (ASM)** adalah sebuah **metode *flowchart*** yang digunakan secara khusus untuk merancang unit kontrol dalam sistem digital. Tujuannya adalah untuk menggambarkan secara visual urutan langkah-langkah (algoritma) yang harus dilakukan oleh perangkat keras.

Diagram ini memiliki tiga komponen dasar:

- **Kotak Keadaan (State Box):** Berbentuk persegi panjang, berisi nama *state*. Sirkuit akan "diam" dalam *state* ini selama satu siklus *clock*.
- **Kotak Keputusan (Decision Box):** Berbentuk belah ketupat, digunakan untuk memeriksa sinyal input (misalnya, sebuah tombol atau bit data). Hasilnya akan menentukan jalur mana yang akan diambil selanjutnya.
- **Kotak Output Kondisional (Conditional Output Box):** Berbentuk lonjong, berisi output yang hanya aktif jika kondisi tertentu pada kotak keputusan terpenuhi.

Secara sederhana, diagram ASM adalah peta jalan untuk "otak" sirkuit, yang memberitahukan apa yang harus dilakukan, kondisi apa yang harus diperiksa, dan kapan harus pindah ke langkah berikutnya

2.2 Perkalian Biner: Algoritma Geser dan Jumlah (*Shift-and-Add*)

Perkalian pada dasarnya adalah proses penjumlahan berulang. Pada sistem digital, operasi perkalian biner dapat diimplementasikan secara efisien menggunakan algoritma *shift-and-add*. Metode ini meniru cara perkalian panjang yang dilakukan secara manual, tetapi dioptimalkan untuk perangkat keras dengan operasi dasar berupa penjumlahan dan pergeseran bit.

Proses ini melibatkan tiga register utama:

- **Register M (Multiplicand):** Menyimpan bilangan pengali.
- **Register Q (Multiplier):** Menyimpan bilangan yang akan dikalikan. Bit terendah (LSB) dari register ini digunakan untuk mengambil keputusan pada setiap langkah.
- **Register A (Accumulator):** Awalnya diatur ke nol, register ini berfungsi untuk mengakumulasi hasil penjumlahan parsial.

Algoritma *shift-and-add* untuk perkalian N-bit berjalan dalam N siklus. Setiap siklus terdiri dari langkah-langkah berikut:

1. **Pengecekan:** Periksa nilai bit terendah (LSB) dari register Q.
2. **Penjumlahan (Jika Perlu):** Jika LSB dari Q adalah '1', maka nilai pada register M ditambahkan ke register A. Jika '0', langkah ini dilewati.
3. **Pergeseran:** Lakukan operasi geser-kanan-aritmetik (*arithmetic right shift*) sepanjang 1 bit pada register gabungan A dan Q. Bit carry-out dari penjumlahan (jika ada) juga ikut digeser masuk ke bit tertinggi (MSB) dari A.
4. **Ulangi:** Proses diulangi sebanyak N kali hingga semua bit pada multiplier telah diproses.

Setelah N siklus selesai, hasil perkalian 2N-bit akan tersimpan di dalam register gabungan A dan Q.

Bab 3 Pembahasan

3.1 Perancangan

Perancangan pengali biner ini bertujuan untuk membuat sebuah sirkuit digital yang mampu mengalikan dua buah bilangan biner 3-bit untuk menghasilkan sebuah produk 6-bit. Arsitektur yang digunakan memisahkan antara **Unit Pemroses Data (Datapath)** dan **Unit Kontrol (Control Unit)**, di mana logikanya didasarkan pada **Algoritma Geser dan Jumlah (*Shift-and-Add*)**.

- **Datapath** dirancang untuk melakukan semua operasi aritmetika dan penyimpanan data. Komponen utamanya meliputi register 3-bit untuk *Multiplicand* (M), *Multiplier* (Q), dan Akumulator (A), serta sebuah unit penjumlah (*adder*) dan pencacah (*counter*) untuk mengontrol jumlah iterasi.
- **Control Unit** berfungsi sebagai otak dari sistem. Bagian ini diimplementasikan sebagai sebuah **Algorithmic State Machine (ASM)** yang menghasilkan urutan sinyal kontrol (load, add_op, shift_op, done) untuk mengarahkan operasi pada Datapath.

Logika transisi dan output dari Unit Kontrol (ASM) dapat dirangkum secara formal dalam sebuah *lookup table* atau Tabel Transisi Keadaan berikut. Tabel ini mendefinisikan *state* berikutnya dan sinyal output yang aktif berdasarkan *state* saat ini dan input status (start, Q_lsb, cnt_done).

Tabel 1. *Lookup table*

Siklus Clock	State FSM	Operasi / Keputusan	M	C	A	Q
1	S_LOAD	load <= '1'	110	0	000	101
2	S_CHECK	Cek Q[0] -> (1)	110	0	000	101
3	S_ADD	add_op <= '1'	110	0	110	101
4	S_SHIFT	shift_op <= '1'	110	0	011	010
5	S_CHECK	Cek Q[0] -> (0)	110	0	011	010
6	S_SHIFT	shift_op <= '1'	110	0	001	101
7	S_CHECK	Cek Q[0] -> (1)	110	0	001	101
8	S_ADD	add_op <= '1'	110	0	111	101
9	S_SHIFT	shift_op <= '1'	110	0	011	110
10	S_CHECK	Cek cnt_done -> (1)	110	0	011	110
11	S_DONE	done <= '1'	110	0	011	110

Proses perkalian diinisiasi oleh Unit Kontrol saat sinyal start diaktifkan. Unit Kontrol pertama-tama mengeluarkan sinyal load untuk menginstruksikan Datapath agar memuat nilai *multiplicand* ke register M, *multiplier* ke register Q, serta mereset akumulator A dan carry C. Setelah inisialisasi, FSM pada Unit Kontrol masuk ke dalam sebuah *loop* iteratif. Dalam setiap iterasi, FSM akan memeriksa bit LSB dari register Q (Q_lsb). Jika Q_lsb bernilai '1', FSM akan

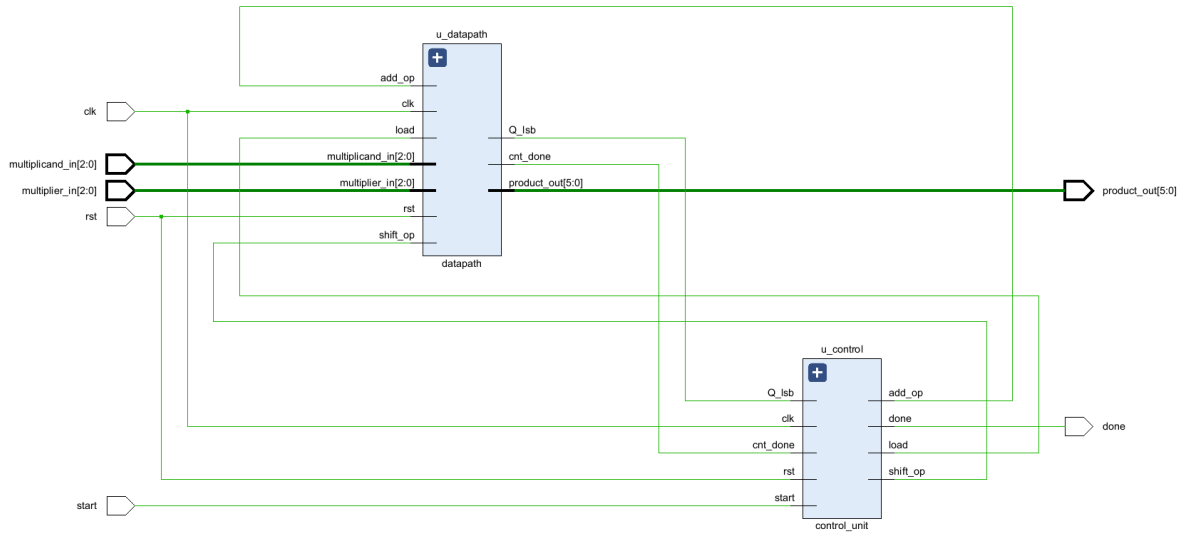
memerintahkan Datapath untuk melakukan operasi penjumlahan (A = A + M) yang kemudian selalu diikuti oleh operasi pergeseran (shift). Jika Q_lsb bernilai '0', operasi penjumlahan akan dilewati dan FSM langsung memerintahkan operasi pergeseran.

```
case current_state is
    when S_IDLE =>
        if (start = '1') then
            next_state <= S_LOAD; -- Pindah ke state LOAD, jangan buat keputusan dulu
        end if;

    when S_LOAD => -- STATE BARU
        load <= '1'; -- Aktifkan sinyal load di sini
        next_state <= S_CHECK; -- Setelah load, pindah untuk mengecek LSB

    when S_CHECK => -- STATE BARU UNTUK MEMBUAT KEPUTUSAN
        if (cnt_done = '1') then
            next_state <= S_DONE; -- Jika sudah selesai, ke DONE
        else
            if (Q_lsb = '1') then -- Sekarang Q_lsb sudah valid
                next_state <= S_ADD;
            else
                next_state <= S_SHIFT;
            end if;
        end if;

    when S_ADD =>
        add_op <= '1';
        next_state <= S_SHIFT; -- Setelah ADD, selalu SHIFT

    when S_SHIFT =>
        shift_op <= '1';
        next_state <= S_CHECK; -- Setelah SHIFT, kembali mengecek kondisi

    when S_DONE =>
        done <= '1';
        if (start = '0') then
            next_state <= S_IDLE;
        end if;

end case;
```

Gambar 3.1 Bagian Kode Unit Control

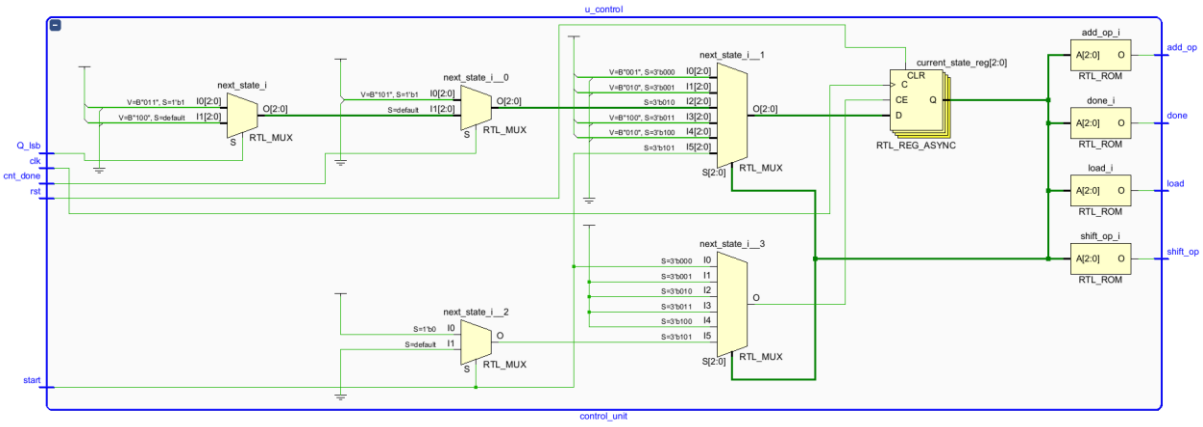
Unit Kontrol dirancang sebagai sebuah Finite State Machine (FSM) yang kokoh dengan state-state yang terdefinisi secara eksplisit: S_IDLE, S_LOAD, S_CHECK, S_ADD, S_SHIFT, dan S_DONE. Penggunaan state khusus seperti S_LOAD memastikan bahwa Datapath memiliki satu siklus *clock* penuh untuk proses inisialisasi yang stabil. Keputusan utama dibuat pada state S_CHECK, di mana kondisi dari Q_lsb dan cnt_done dievaluasi untuk menentukan transisi state berikutnya. Desain ini secara efektif memisahkan antara pengambilan keputusan dengan eksekusi aksi (add_op dan shift_op), sehingga mencegah potensi *race condition* dan menjamin operasi yang sinkron dan dapat diprediksi pada setiap langkahnya. Setelah tiga siklus pergeseran, FSM akan transisi ke state S_DONE dan mengeluarkan sinyal done untuk menandakan bahwa hasil perkalian 6-bit yang valid telah tersedia pada gabungan register {A, Q}.

3.2 RTL Desainn



Gambar 3.2 RTL

Diagram pada Gambar 3.2 mengilustrasikan interaksi dinamis antara Unit Kontrol dan Unit Data yang membentuk sebuah sistem loop tertutup (closed-loop system). Alur kerja dimulai ketika sinyal start menginisiasi **u_control**. Unit Kontrol kemudian secara sekuensial mengeluarkan sinyal-sinyal perintah seperti load, add_op, dan shift_op ke **u_datapath**. Setelah mengeksekusi setiap perintah, u_datapath mengirimkan sinyal status umpan balik (Q_lsb dan cnt_done) kembali ke u_control. Umpan balik inilah yang digunakan oleh FSM untuk membuat keputusan dan menentukan langkah selanjutnya dalam algoritma. Siklus 'perintah-eksekusi-umpan balik' ini menjamin eksekusi yang tersinkronisasi dan berulang hingga proses perkalian selesai, yang pada akhirnya ditandai dengan aktifnya sinyal done dan tersedianya hasil akhir pada port product_out.



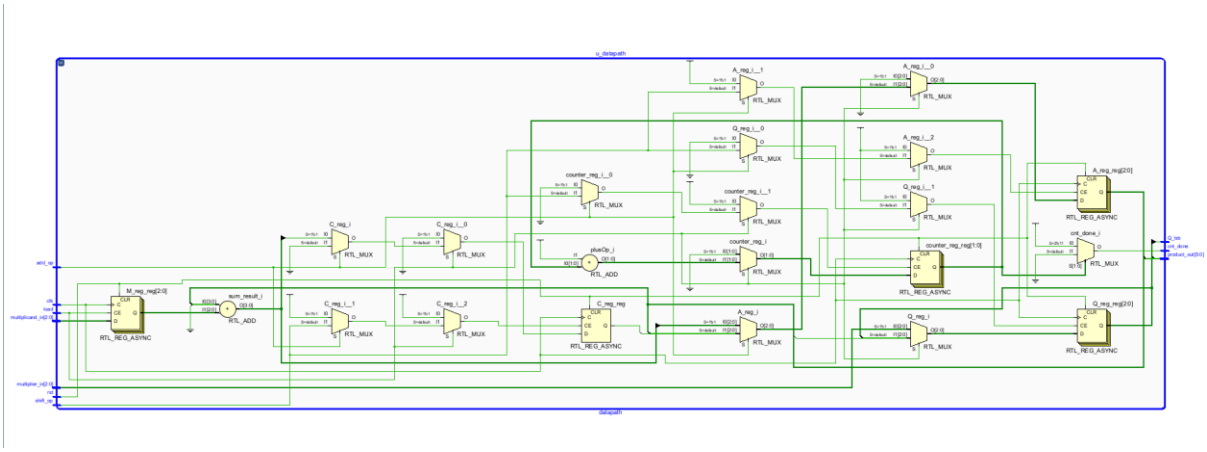
Gambar 3.3 Isi Blok unit control

Diagram skematik RTL pada Gambar [Nomor Gambar] memberikan visualisasi detail dari implementasi perangkat keras untuk **Unit Kontrol**. Inti dari desain ini adalah sebuah

Finite State Machine (FSM) yang dapat diuraikan menjadi komponen-komponen fungsional berikut.

Pusat dari FSM adalah **Register State (RTL_REG_ASYNC)**, sebuah register 3-bit yang menyimpan *state* sistem saat ini (current_state_reg[2:0]). Register ini diperbarui pada setiap tepi naik dari sinyal clk dan memiliki reset asinkron untuk mengembalikan FSM ke keadaan awal. Logika untuk menentukan *state* selanjutnya diimplementasikan oleh jaringan kombinasional yang terdiri dari beberapa **Multiplexer (RTL_MUX)**. Jaringan ini secara efektif merepresentasikan struktur CASE atau IF-ELSE dari kode VHDL, di mana ia memilih nilai *state* berikutnya berdasarkan masukan FSM (start, Q_lsb, cnt_done) dan nilai *state* yang sekarang.

Implementasi logika output menjadi sangat menarik, di mana sinyal-sinyal kontrol (add_op, done, load, shift_op) dihasilkan oleh empat blok **RTL_ROM** terpisah. Setiap ROM ini berfungsi sebagai Look-Up Table (LUT) yang menggunakan current_state_reg sebagai alamatnya. Berdasarkan alamat *state* saat ini, ROM akan langsung mengeluarkan nilai '1' atau '0' yang telah diprogram sebelumnya. Struktur ini secara jelas mengklasifikasikan FSM ini sebagai **tipe Moore**, karena sinyal output hanya merupakan fungsi dari *state* saat ini, yang menjamin kestabilan sinyal output selama satu siklus *clock* penuh dan bebas dari *glitch*.



Gambar 3.4 Isi Blok Datapath

Diagram RTL pada Gambar 3.4 memvisualisasikan implementasi perangkat keras dari **Unit Data (u_datapath)**, yang dapat dianalisis berdasarkan fungsinya:

1. **Elemen Penyimpanan Data:** Unit ini memiliki lima set register utama (RTL_REG_ASYNC) untuk menyimpan keadaan sistem: M_reg_reg untuk multiplicand yang nilainya tetap setelah di-load; serta A_reg_reg, Q_reg_reg, C_reg_reg, dan counter_reg_reg yang nilainya diperbarui secara dinamis selama proses iterasi. Setiap register ini memiliki input Clock Enable (CE) yang dikendalikan oleh sinyal dari Unit Kontrol, memastikan data hanya ditulis saat operasi yang benar sedang aktif.
2. **Unit Aritmetika:** Operasi penjumlahan inti A + M diimplementasikan oleh blok plus0_1 (RTL_ADD). Blok ini menerima input 3-bit dari register A dan M, dan

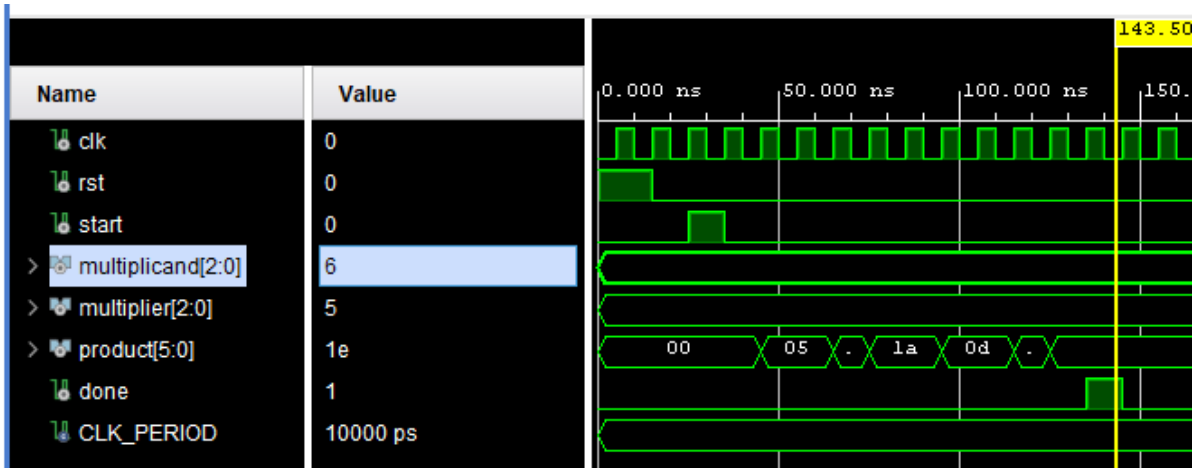
menghasilkan output 4-bit (sum_result_i), di mana 3 bit LSB merupakan hasil penjumlahan dan bit MSB adalah nilai *carry-out* yang akan disimpan di register C.

3. **Logika Aliran Data:** Jaringan **Multiplexer (RTL_MUX)** yang kompleks memainkan peran krusial dalam mengimplementasikan logika kondisional dari kode VHDL (IF-ELSIF-ELSE). Sebagai contoh, multiplexer yang terhubung ke input D dari register A_reg_reg berfungsi untuk memilih sumber data. Ia akan memilih nilai nol saat load aktif, hasil dari blok penjumlah saat add_op aktif, atau nilai A yang telah digeser saat shift_op aktif. Jaringan MUX ini memastikan bahwa data yang benar diarahkan ke register yang tepat pada waktu yang tepat.
4. **Logika Output:** Sinyal output dari Datapath dihasilkan secara langsung dari register. product_out adalah hasil konkatenasi (penggabungan) dari output A_reg_reg dan Q_reg_reg. Sinyal status seperti Q_lsb dan cnt_done juga diambil langsung dari bit register Q dan output pembanding pada register counter, untuk kemudian dikirim sebagai umpan balik ke Unit Kontrol.

Arsitektur yang disajikan pada level RTL ini merupakan contoh klasik dari sebuah **Algorithmic State Machine (ASM)**, sebuah metodologi untuk merancang sistem digital yang menjalankan algoritma. Secara spesifik, **Unit Kontrol (`u_control`)** diimplementasikan sebagai **FSM tipe Moore** yang murni. Hal ini terbukti karena sinyal-sinyal outputnya (`add_op`, `load`, `shift_op`, `done`) hanya merupakan fungsi dari *state* sistem saat ini dan tidak bergantung pada input seketika. Pendekatan Moore ini dipilih untuk menjamin sinyal kontrol yang stabil dan bebas *glitch* selama satu siklus *clock* penuh.

Di sisi lain, interaksi antara Unit Kontrol dengan **Unit Data (`u_datapath`)** menunjukkan **karakteristik dari mesin Mealy**. Meskipun `u_datapath` sendiri bukanlah FSM, ia menghasilkan sinyal status (`Q_lsb`, `cnt_done`) yang menjadi input langsung untuk logika transisi state pada Unit Kontrol. Ketergantungan transisi *state* pada input-input yang berasal dari datapath inilah yang mencerminkan perilaku Mealy, di mana *state* berikutnya ditentukan oleh *state* saat ini dan input saat ini. Kombinasi dari **kontroler Moore** yang stabil dengan **umpan balik status (feedback) bergaya Mealy** dari datapath ini membentuk sebuah sistem ASM yang lengkap, efisien, dan andal untuk eksekusi algoritma di level perangkat keras.

3.3 Hasil Simulasi



Gambar 3.5 Hasil Simulasi

Gambar 3.5 menampilkan hasil simulasi fungsional dari rangkaian pengali biner 3x3 dalam bentuk *waveform*. Simulasi ini bertujuan untuk memverifikasi kebenaran logika dari desain yang telah dibuat dengan menggunakan kasus uji spesifik. Pada pengujian ini, input **multiplicand** diatur ke nilai **6** (biner 110) dan input **multiplier** diatur ke nilai **5** (biner 101). Proses diawali dengan pemberian sinyal **rst** (reset) sesaat untuk memastikan sistem berada dalam kondisi awal yang terdefinisi. Selanjutnya, sebuah pulsa tunggal pada sinyal **start** diaktifkan untuk memicu Unit Kontrol memulai sekuens perkalian.

Setelah sinyal **start** aktif, dapat diamati bahwa sistem mulai beroperasi, yang ditunjukkan oleh perubahan nilai pada sinyal **product**. Nilai-nilai antara yang muncul pada sinyal product ini merepresentasikan keadaan gabungan register {A, Q} pada setiap siklus algoritma "Add-and-Shift". Setelah melalui beberapa siklus *clock*, sinyal **done** menjadi aktif ('1'), yang menandakan bahwa FSM pada Unit Kontrol telah menyelesaikan seluruh iterasinya. Pada saat yang sama, sinyal **product** stabil pada nilai akhirnya yaitu **1E** heksadesimal. Nilai 1E ini setara dengan **30** dalam sistem desimal, yang merupakan hasil perkalian yang benar dari 6 dikali 5. Dengan demikian, hasil simulasi ini berhasil memvalidasi bahwa rangkaian pengali bekerja secara fungsional sesuai dengan rancangan.

Bab 4 Kesimpulan

Berdasarkan perancangan, implementasi, dan verifikasi yang telah dilakukan, dapat ditarik beberapa kesimpulan utama sebagai berikut:

1. **Keberhasilan Perancangan**: Rangkaian pengali biner 3-bit x 3-bit yang menghasilkan produk 6-bit telah berhasil dirancang dan diimplementasikan secara fungsional. Desain ini secara efektif mengadopsi arsitektur modular yang memisahkan antara **Unit Data (Datapath)** dan **Unit Kontrol (Control Unit)** , dengan landasan algoritma **Geser dan Jumlah (*Shift-and-Add*)**.

2. **Implementasi FSM yang Robust**: Unit Kontrol berhasil diwujudkan sebagai sebuah **Finite State Machine (FSM) tipe Moore** yang kokoh, di mana logikanya didefinisikan secara formal menggunakan diagram **ASM (*Algorithmic State Machine*)**. Penggunaan *state* yang terdefinisi secara eksplisit seperti S_LOAD dan S_CHECK terbukti mampu menciptakan proses yang stabil dan dapat diprediksi, serta bebas dari potensi *race condition*. Kombinasi kontroler Moore dengan umpan balik dari datapath membentuk sistem ASM bergaya Mealy yang lengkap dan efisien.

3. **Validasi Fungsional**: Verifikasi melalui simulasi *waveform* telah membuktikan bahwa desain bekerja sesuai dengan spesifikasi yang diharapkan. Pada pengujian kasus perkalian 6 dengan 5 , rangkaian secara akurat menghasilkan output akhir **1E heksadesimal (30 desimal)**. Keberhasilan simulasi ini, yang ditandai dengan aktifnya sinyal done setelah semua iterasi selesai, mengonfirmasi bahwa seluruh komponen perangkat keras pada level RTL, mulai dari register, unit aritmetika, hingga logika aliran data, telah terintegrasi dan berfungsi dengan benar.

Daftar Pustaka

Patterson, D. A., & Hennessy, J. L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th ed.). Morgan Kaufmann.

Xilinx, Inc. (2013). *Sequential System Design Using ASM Charts*. Xilinx University Program.