Ujian Akhir Semester
Mata Kuliah Deskripsi Perangkat Keras / Desain FPGA
Laporan Lab 10

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

Laporan ini merangkum perancangan, simulasi, dan implementasi dari tiga sistem digital sekuensial yang berbeda menggunakan bahasa VHDL. Setiap proyek mengeksplorasi konsep dan teknik desain yang berbeda dalam sistem digital.

Ketiga proyek tersebut adalah:

1. **Mealy Sequence Detector:** Sebuah mesin status Mealy yang mendeteksi jika total jumlah input '1' yang diterima habis dibagi tiga.
2. **Moore Sequence Detector:** Sebuah mesin status Moore yang mendeteksi tiga urutan input dua-langkah yang berbeda untuk mengatur, mereset, atau melakukan *toggle* pada output.
3. **Counter Berbasis ROM:** Sebuah *counter* dengan urutan hitungan non-sekuensial yang logikanya diimplementasikan menggunakan *Read-Only Memory* (ROM).

Tujuan laporan ini adalah untuk menyelesaikan tugas UAS dan mendemostrasikan proses desain lengkap, mulai dari analisis spesifikasi, perancangan FSM, penulisan kode VHDL, dan verifikasi melalui simulasi.

Bab 2 Tinjauan Pustaka

2.1 Finite State Machine (FSM)

FSM adalah model komputasi yang digunakan untuk merancang rangkaian sekuensial. Sebuah FSM terdiri dari sejumlah state (keadaan) yang terbatas, transisi antar state berdasarkan input, dan output yang dihasilkan.

2.2 Perbedaan Mesin Mealy dan Moore

- **Mesin Moore:** Output hanya bergantung pada **state saat ini**. Output bersifat stabil dan hanya berubah setelah transisi state pada detak clock.
- **Mesin Mealy:** Output bergantung pada **state saat ini DAN input saat ini**. Ini memungkinkan respons yang lebih cepat karena output dapat berubah seketika saat input berubah, tanpa harus menunggu siklus clock berikutnya.

2.3 Implementasi FSM Berbasis ROM

Untuk FSM dengan banyak state atau transisi yang tidak beraturan, logika transisi dapat diimplementasikan menggunakan *lookup table* yang disimpan dalam ROM. Dalam pendekatan ini, state saat ini digunakan sebagai alamat ROM, dan data yang tersimpan di alamat tersebut merupakan state berikutnya. Ini menyederhanakan logika kombinasional yang diperlukan.

2.4 Alur Desain FPGA

Proses desain untuk FPGA umumnya mengikuti alur berikut:

1. **Desain & Penulisan Kode:** Menulis perilaku sirkuit menggunakan HDL (VHDL/Verilog).
2. **Simulasi:** Memverifikasi fungsionalitas kode menggunakan *testbench*.

Bab 3 Pembahasan

3.1 Perancangan Mealy Sequence Detector

Perancangan detektor sekuensial ini didasarkan pada model Mesin Mealy, di mana output yang dihasilkan merupakan fungsi dari state saat ini dan input yang diterima. Tujuan utamanya adalah untuk mendeteksi kondisi di mana total jumlah input bernilai '1' habis dibagi tiga.

Untuk mencapai tujuan ini, logika sistem dimodelkan ke dalam tiga state yang berbeda. Setiap state secara unik merepresentasikan nilai sisa (modulo) dari total akumulasi input '1' ketika dibagi tiga.

Definisi dari setiap state adalah sebagai berikut:

- **State S0:** Merepresentasikan keadaan di mana total jumlah input '1' yang telah diterima adalah kelipatan tiga (kongruen dengan 0 modulo 3). State ini juga berfungsi sebagai state awal (reset) sistem.
- **State S1:** Merepresentasikan keadaan di mana total jumlah input '1' yang telah diterima, jika dibagi tiga, akan menghasilkan sisa 1 (kongruen dengan 1 modulo 3).
- **State S2:** Merepresentasikan keadaan di mana total jumlah input '1' yang telah diterima, jika dibagi tiga, akan menghasilkan sisa 2 (kongruen dengan 2 modulo 3).

Logika transisi dan output tersebut dapat dirangkum secara formal dalam **Tabel Transisi State** berikut:

Status Saat Ini	Input (ain)	Status Berikutnya	Output (yout)
S0	0'	S0	0'
S0	1'	S1	0'
S1	0'	S1	0'
S1	1'	S2	0'
S2	0'	S2	0'
S2	1'	S0	1'

Transisi antar state diatur oleh input ain pada setiap siklus clock.

- Ketika input ain bernilai **'1'**, akumulasi hitungan bertambah, menyebabkan sistem bertransisi secara siklik: dari S0 ke S1, dari S1 ke S2, dan dari S2 kembali ke S0.
- Ketika input ain bernilai **'0'**, akumulasi hitungan tidak berubah. Akibatnya, sistem akan tetap berada pada state saat ini (*self-loop*).

Sesuai dengan karakteristik Mesin Mealy, logika output yout dievaluasi berdasarkan state saat ini dan input saat ini. Output yout diatur untuk bernilai '1' secara eksklusif pada transisi dari state S2 ke S0. Transisi ini menandakan bahwa sistem, yang sebelumnya telah mengakumulasi sejumlah (3n + 2) buah '1', menerima satu lagi input '1', sehingga total akumulasinya menjadi kelipatan tiga (3n + 3). Pada semua kondisi transisi lainnya, output yout

akan bernilai '0'. Desain ini secara efisien memenuhi seluruh spesifikasi fungsional yang dibutuhkan.

3.1.1 Implementasi logika pada fpga

Pada dasarnya, kode VHDL tersebut adalah sebuah cetak biru (*blueprint*) untuk mengkonfigurasi dua jenis komponen utama di dalam chip FPGA: **Flip-Flops** dan **Look-Up Tables**.

Elemen Memori (Flip-Flops)

Proses yang menggunakan rising_edge(clk) (yaitu state_register_proc dan counter_proc) diubah menjadi elemen memori yang disebut **Flip-Flops (FFs)**.

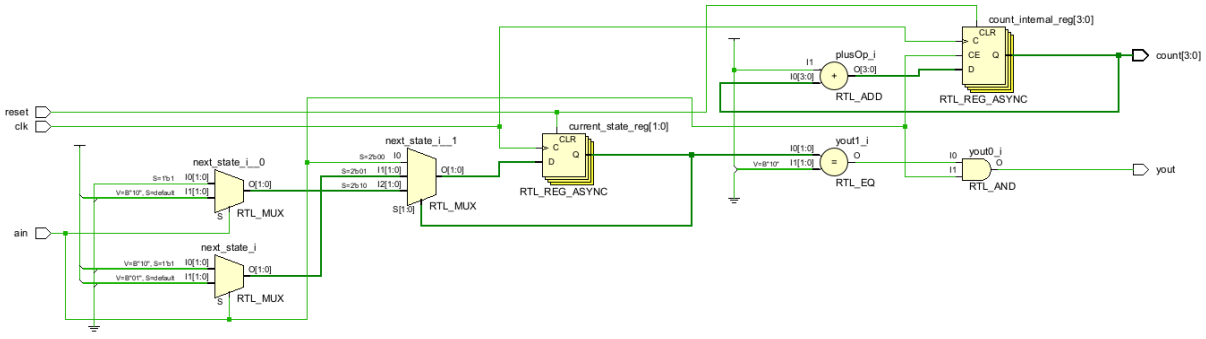
- Tugas Flip-Flop adalah untuk **menyimpan nilai** atau "state" saat ini.
- Sinyal current_state dan count secara fisik adalah sekumpulan Flip-Flop di dalam chip yang nilainya hanya diperbarui saat ada detak clock.

Elemen Logika (Look-Up Tables)

Proses yang tidak memiliki clock (yaitu next_state_logic_proc dan output_logic_proc) diubah menjadi elemen logika yang disebut **Look-Up Tables (LUTs)**.

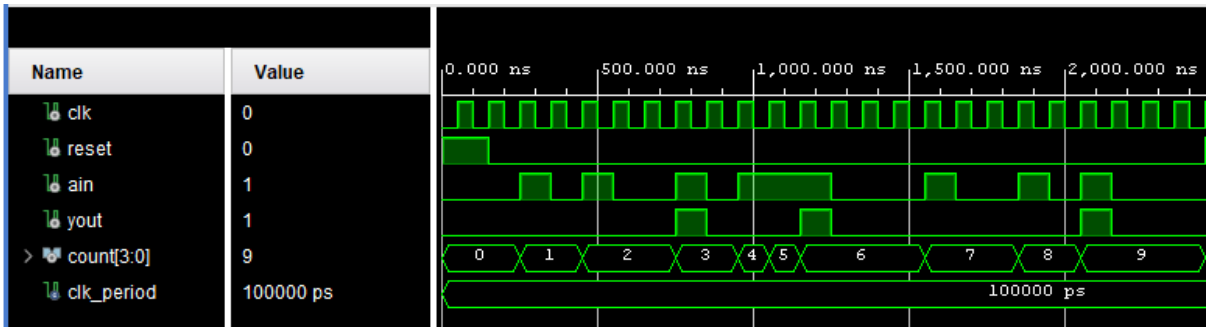
- Tugas LUT adalah untuk **membuat keputusan** secara instan berdasarkan input yang diterima.
- Logika untuk menentukan apa state berikutnya (next_state) dan apa outputnya (yout) dibangun dari LUTs ini.

3.1.2 Hasil Design dan Hasil simulasi



Desain ini diimplementasikan sebagai sebuah Mesin Status Mealy yang logikanya terbagi menjadi dua bagian utama yang bekerja secara paralel. Bagian pertama adalah *state machine* inti yang menggunakan tiga state (S0, S1, S2) untuk secara efisien melacak sisa bagi tiga dari total input '1' yang diterima. Logika output yout dirancang sesuai karakteristik Mealy, di mana output hanya akan aktif '1' secara spesifik pada transisi dari state S2 saat menerima

input '1', yang menandakan total hitungan telah mencapai kelipatan tiga. Bagian kedua adalah sebuah sirkuit pencacah (*counter*) 4-bit yang berfungsi secara independen untuk menampilkan total akumulasi input '1' pada LED, tanpa memengaruhi logika utama dari *state machine*. Struktur modular ini memastikan bahwa fungsi deteksi dan fungsi penampilan visual berjalan secara efisien dan terpisah sesuai spesifikasi.



Hasil simulasi berhasil memverifikasi fungsionalitas rangkaian detektor sekuensial. Setelah kondisi reset awal, sinyal count terlihat meningkat secara akurat setiap kali input ain bernilai '1'. Secara spesifik, output yout terbukti menghasilkan pulsa aktif-tinggi tepat pada saat jumlah input '1' mencapai kelipatan tiga, seperti yang ditunjukkan pada hitungan ke-3, ke-6, dan ke-9 dalam *waveform*. Perilaku ini secara efektif memvalidasi bahwa desain Mesin Mealy telah diimplementasikan dengan benar sesuai spesifikasi.

3.2 Perancangan Moore Sequence Detector

Perancangan detektor sekuensial ini didasarkan pada model Mesin Moore, di mana nilai output yout hanya ditentukan oleh *state* (keadaan) sistem saat ini. Spesifikasi tugas menuntut output yout untuk bersifat *stateful*, artinya ia harus dapat mempertahankan nilainya (0 atau 1) dan dapat melakukan *toggle* (membalik nilainya). Untuk memenuhi kebutuhan ini, nilai output yout itu sendiri harus diintegrasikan ke dalam definisi state.

Selain itu, mesin harus memiliki memori untuk mencatat input pertama dalam sebuah urutan dua-langkah yang valid (01, 11, atau 10). Kombinasi dari dua kebutuhan memori ini—nilai output saat ini (1 bit) dan input relevan terakhir yang dilihat (2 bit)—menghasilkan total **delapan state unik** yang diperlukan untuk implementasi yang benar.

State-state tersebut dikelompokkan berdasarkan nilai output yang dihasilkannya:

- **State dengan Output yout = '0':**
    - ZERO_IDLE: Kondisi awal atau siaga, menunggu urutan baru.
    - ZERO_SAW_01: Kondisi setelah input 01 diterima.
    - ZERO_SAW_11: Kondisi setelah input 11 diterima.
    - ZERO_SAW_10: Kondisi setelah input 10 diterima.
- **State dengan Output yout = '1':**
    - ONE_IDLE: Kondisi siaga, menunggu urutan baru.

- ONE_SAW_01: Kondisi setelah input 01 diterima.
- ONE_SAW_11: Kondisi setelah input 11 diterima.
- ONE_SAW_10: Kondisi setelah input 10 diterima.

Logika transisi lengkap untuk kedelapan state tersebut dirangkum dalam Tabel Transisi berikut. Tabel ini berfungsi sebagai *lookup table* yang menentukan Status Berikutnya berdasarkan Status Saat Ini dan Input (ain).

Status Saat Ini	Input (ain)	Status Berikutnya	Deskripsi Aksi
ZERO_IDLE	1	ZERO_SAW_01	Memulai deteksi urutan 01.
	11	ZERO_SAW_11	Memulai deteksi urutan 11.
	10	ZERO_SAW_10	Memulai deteksi urutan 10.
	Lainnya	ZERO_IDLE	Tetap siaga.
ONE_IDLE	1	ONE_SAW_01	Memulai deteksi urutan 01.
	11	ONE_SAW_11	Memulai deteksi urutan 11.
	10	ONE_SAW_10	Memulai deteksi urutan 10.
	Lainnya	ONE_IDLE	Tetap siaga.
ZERO_SAW_01	0	ZERO_IDLE	Urutan selesai, yout menjadi 0.
	Lainnya	ZERO_IDLE	Urutan batal.
ONE_SAW_01	0	ZERO_IDLE	Urutan selesai, yout menjadi 0.
	Lainnya	ONE_IDLE	Urutan batal.
ZERO_SAW_11	0	ONE_IDLE	Urutan selesai, yout menjadi 1.
	Lainnya	ZERO_IDLE	Urutan batal.
ONE_SAW_11	0	ONE_IDLE	Urutan selesai, yout menjadi 1.
	Lainnya	ONE_IDLE	Urutan batal.
ZERO_SAW_10	0	ONE_IDLE	Urutan selesai, yout di-toggle 0 → 1.
	Lainnya	ZERO_IDLE	Urutan batal.
ONE_SAW_10	0	ZERO_IDLE	Urutan selesai, yout di-toggle 1 → 0.
	Lainnya	ONE_IDLE	Urutan batal.

Dari tabel di atas, terlihat bahwa mesin akan bertransisi dari state IDLE ke SAW_xx saat menerima input pertama dari sebuah urutan yang valid. Jika input berikutnya adalah 00, mesin akan menyelesaikan urutan dan bertransisi kembali ke salah satu dari state IDLE. State tujuan (ZERO_IDLE atau ONE_IDLE) ditentukan oleh aturan yang berlaku (menjadi 0, menjadi 1,

atau toggle), sehingga output yout yang baru akan aktif pada siklus clock berikutnya. Jika urutan yang valid gagal terbentuk, mesin akan kembali ke kondisi siaga (IDLE) tanpa mengubah output. Logika output yout sendiri secara inheren terikat pada state saat ini, yang merupakan karakteristik utama dari implementasi Mesin Moore ini.

3.2.1 Implementasi Logika

Elemen Logika (Look-Up Tables)

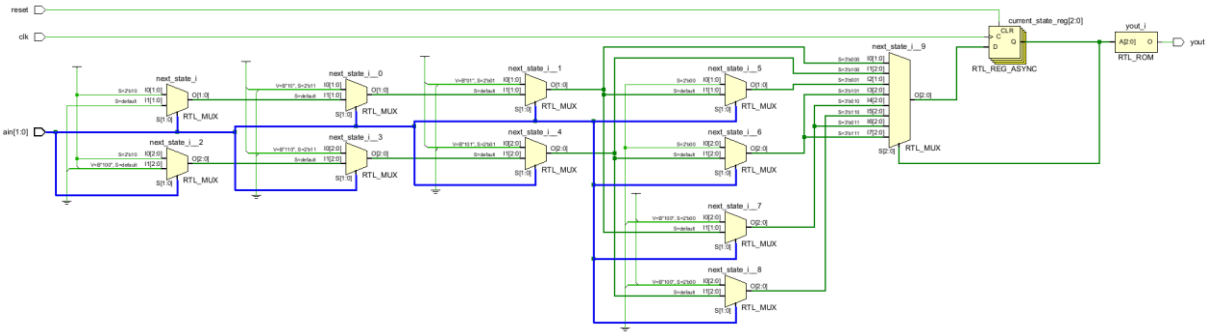
LUT adalah unit dasar yang berfungsi sebagai "otak" dari sirkuit. Setiap bagian dari kode Anda yang mendeskripsikan logika kombinasional seperti pernyataan case, if/else, atau operasi gerbang logika (and, or, not) akan diubah oleh *software* sintesis menjadi sebuah *truth table* (tabel kebenaran). Tabel kebenaran ini kemudian diprogram ke dalam LUT.

Singkatnya, **LUTs menjalankan semua proses pengambilan keputusan** secara instan berdasarkan input yang diterimanya.

Elemen Memori (Flip-Flops)

Flip-Flop adalah unit dasar yang berfungsi sebagai "memori" satu bit. Setiap bagian dari kode Anda yang bersifat sekuensial atau sinkron—ditandai dengan adanya sensitivitas terhadap detak clock (misalnya, rising_edge(clk))—akan diubah menjadi Flip-Flop.

3.2.2 Hasil Design dan Hasil Simulasi

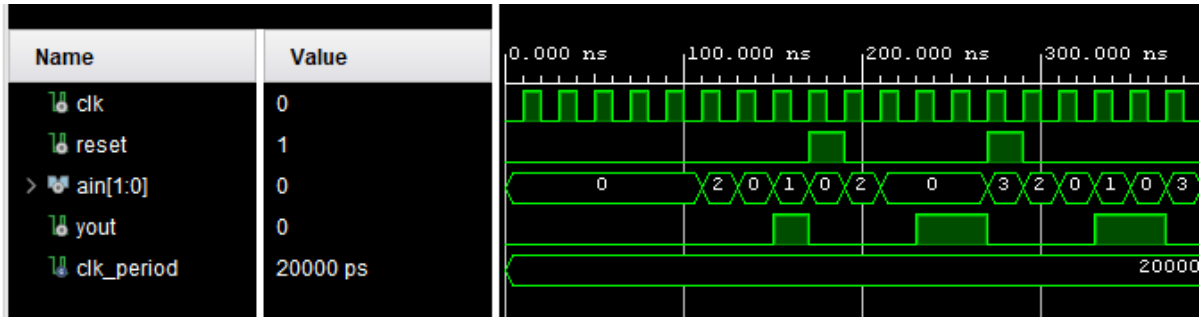


Skema RTL (Register-Transfer Level) ini memvisualisasikan bagaimana kode VHDL untuk Mesin Moore diimplementasikan menjadi komponen perangkat keras. Arsitektur ini secara fisik memisahkan jalur logika untuk output dari jalur logika untuk input, yang merupakan karakteristik fundamental dari Mesin Moore.

1. **Logika Output (yout) Hanya Tergantung pada State:** Perhatikan bahwa sinyal output akhir (yout) dihasilkan oleh sebuah blok logika (ditandai RTL_ROM pada skema Anda) yang inputnya hanya berasal dari satu sumber: output dari **Register State** (current_state_reg). Tidak ada jalur kabel sama sekali dari sinyal ain[1:0] yang masuk ke blok logika output ini. Ini membuktikan bahwa nilai yout hanya ditentukan oleh *state* (keadaan) sistem saat ini, bukan oleh input yang masuk pada saat itu juga.
2. **Logika Input (ain) Hanya Memengaruhi State Berikutnya:** Di sisi lain, sinyal input ain[1:0] hanya terhubung ke blok-blok logika kombinasional (tumpukan RTL_MUX) yang bertugas untuk menghitung nilai dari **state berikutnya**. Artinya, input ain hanya

bisa memengaruhi apa keadaan mesin *setelah* detak clock berikutnya, tetapi tidak bisa mengubah output yout secara instan.

Pemisahan jalur yang jelas ini di mana output hanya "melihat" state yang tersimpan di register, sementara input hanya "melihat" logika untuk state berikutnya adalah implementasi fisik dari prinsip Mesin Moore.



Hasil simulasi *behavioral* ini secara komprehensif memverifikasi fungsionalitas dari rancangan Mesin Status Moore. Simulasi diawali dengan pemberian sinyal reset aktif-tinggi, yang secara benar menginisialisasi output yout ke nilai logika '0' sesuai dengan rancangan state awal.

Setelah reset dilepaskan, pengujian untuk ketiga aturan sekuensial dilakukan secara berurutan.

1. **Pengujian Aturan (iii) - Toggle:** Pada rentang waktu 110 ns hingga 130 ns, urutan input 10, 00 diberikan. Hasilnya, yout yang semula bernilai '0' berhasil melakukan *toggle* menjadi '1' tepat pada *rising edge* clock di t=130 ns.
2. **Pengujian Aturan (i) - Set to Zero:** Selanjutnya, pada rentang 150 ns hingga 170 ns, urutan 01, 00 diuji. yout yang sebelumnya bernilai '1' secara benar berubah menjadi '0' pada t=170 ns.
3. **Pengujian Aturan (ii) - Set to One:** Pengujian untuk aturan ini dapat dilihat pada rentang 270 ns hingga 290 ns. Sebelumnya, yout sudah berada dalam kondisi '1' (akibat *toggle* pada t=230 ns). Ketika urutan 11, 00 selesai, mesin diperintahkan untuk membuat yout menjadi '1'. Karena yout sudah '1', maka output yang benar adalah **mempertahankan nilainya tetap '1'**. Tidak adanya perubahan pada yout dalam kasus ini justru memvalidasi bahwa logika untuk aturan (ii) telah bekerja dengan benar.

Semua perubahan output terjadi secara sinkron dengan detak clock yang menyelesaikan sebuah urutan, yang merupakan ciri khas Mesin Moore. Dengan semua kasus uji yang menunjukkan perilaku yang diharapkan, model VHDL ini divalidasi dengan sukses.

3.3 Perancangan Counter Berbasis ROM

Perancangan *counter* dengan urutan spesifik ini diimplementasikan menggunakan arsitektur *Finite State Machine* (FSM) berbasis *Read-Only Memory* (ROM). Pendekatan ini menggantikan logika kombinasional konvensional, seperti statemen CASE yang kompleks, dengan sebuah *lookup table* (tabel pencarian) yang telah didefinisikan sebelumnya.

Konsepnya adalah menggunakan nilai hitungan saat ini (*current state*) sebagai **alamat** untuk mengakses ROM. Data yang tersimpan pada alamat tersebut merupakan nilai hitungan selanjutnya (*next state*) dalam urutan yang telah ditentukan. Metode ini sangat efisien untuk mengimplementasikan FSM dengan urutan transisi yang non-sekuensial atau kompleks, karena dapat mengurangi kompleksitas logika dan mengoptimalkan penggunaan sumber daya pada FPGA.

Isi dari ROM yang mendefinisikan urutan hitungan dirangkum dalam tabel berikut. State yang tidak terpakai (*unused*) akan dialihkan kembali ke state awal (000) sebagai kondisi default.

Alamat (Desimal)	Alamat (Biner)	Data / State Berikutnya (Biner)	Data / State Berikutnya (Desimal)
0	0	1	1
1	1	11	3
2	10	0	0
3	11	101	5
4	100	0	0 (State Tak Terpakai)
5	101	111	7
6	110	0	0 (State Tak Terpakai)
7	111	10	2

3.1.1 Implementasi Logika pada FPGA

Implementasi kode VHDL untuk *counter* berbasis ROM pada FPGA mengubah deskripsi tekstual menjadi sebuah sirkuit digital yang efisien. Proses sintesis secara cerdas memetakan kode ke dua jenis komponen perangkat keras utama: **Look-Up Tables (LUTs)** yang berfungsi sebagai ROM, dan **Flip-Flops (FFs)** yang berfungsi sebagai register penyimpan hitungan.

Bagian inti dari desain ini, yaitu constant COUNT_ROM, diimplementasikan bukan sebagai gerbang logika yang kompleks, melainkan sebagai sebuah *lookup table* (tabel pencarian) di dalam FPGA. *Software* sintesis akan memprogram **Look-Up Tables (LUTs)** untuk secara langsung merepresentasikan tabel urutan hitungan yang telah kita definisikan. Blok LUT ini berfungsi sebagai sirkuit kombinasional murni: saat diberi input alamat (nilai current_count), ia akan secara instan menghasilkan output data (nilai next_count) sesuai isi tabel.

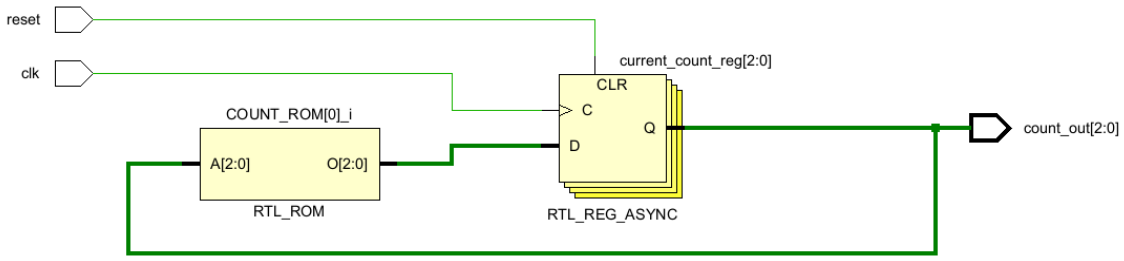
Sinyal current_count yang menyimpan nilai hitungan saat ini, diimplementasikan sebagai sebuah **register 3-bit**. Register ini terdiri dari tiga buah **D-type Flip-Flops (FFs)**. Proses sekuensial (process(clk, reset)) memastikan bahwa:

- Saat reset aktif, semua Flip-Flop diatur ulang ke '0'.

- Pada setiap *rising edge* dari sinyal clk, Flip-Flop akan menangkap nilai dari next_count (yang berasal dari output LUT/ROM) dan menyimpannya sebagai current_count yang baru.

Secara keseluruhan, arsitektur ini membentuk sebuah *feedback loop* yang efisien: Register Flip-Flop mengeluarkan state saat ini, yang kemudian digunakan sebagai alamat oleh LUTs untuk mencari state berikutnya. Hasil dari LUTs ini kemudian siap untuk ditangkap kembali oleh Register pada detak clock selanjutnya, sehingga *counter* dapat berjalan sesuai urutan spesifik yang telah diprogram.

3.2.2 Hasil Desain dan Hasil Simulasi



Skema RTL (Register-Transfer Level) ini merupakan representasi perangkat keras dari desain *counter* berbasis ROM, yang menggambarkan bagaimana kode VHDL disintesis menjadi komponen-komponen digital fungsional. Arsitektur ini terdiri dari dua blok utama yang bekerja dalam sebuah siklus umpan balik (*feedback loop*).

Komponen Utama

1. **Blok RTL_ROM (Logika Kombinasional)** Blok ini adalah implementasi perangkat keras dari *lookup table* yang kita definisikan dalam kode. Ia berfungsi sebagai "kamus" kombinasional yang tidak memiliki memori. Inputnya adalah alamat 3-bit (A[2:0]) dan outputnya adalah data 3-bit (Q[2:0]). Secara instan, ia akan mengeluarkan data yang sesuai dengan alamat yang diberikan.
2. **Blok RTL_REG_ASYNC (Register State)** Blok ini adalah elemen memori dari sirkuit, yang terdiri dari tiga buah D-type Flip-Flop. Tugasnya adalah menyimpan nilai hitungan saat ini (current_count). Ia memiliki tiga input utama: D (data input), C (clock), dan CLR (clear/reset asinkron). Nilai pada input D akan ditangkap dan disimpan untuk menjadi output Q pada setiap *rising edge* dari C. Sinyal reset yang terhubung ke CLR akan memaksa output Q menjadi nol secara langsung.

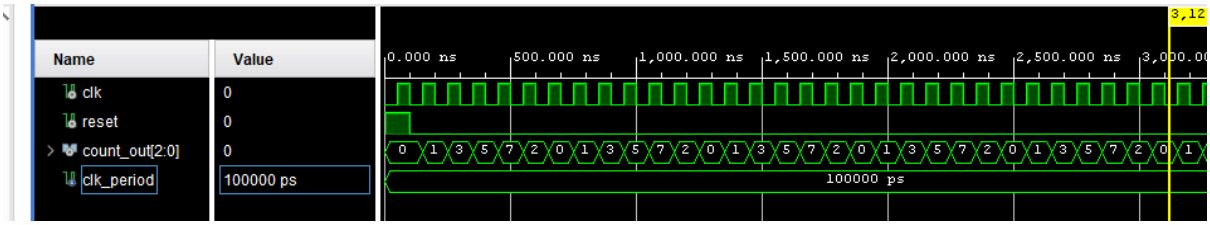
Alur Kerja Sirkuit

Siklus operasi dari *counter* ini adalah sebagai berikut:

1. **Output State:** Register (RTL_REG_ASYNC) mengeluarkan nilai hitungan saat ini (current_count) melalui output Q-nya. Nilai inilah yang juga menjadi output akhir count_out.
2. **Umpan Balik & Lookup:** Nilai current_count ini diumpan balik (*feedback*) dan menjadi input alamat (A) untuk blok RTL_ROM.

3. **Penentuan State Berikutnya:** Berdasarkan alamat yang diterima, blok RTL_ROM langsung mengeluarkan data yang merupakan nilai hitungan berikutnya.
4. **Penyimpanan State Baru:** Nilai hitungan berikutnya ini kemudian menjadi input (D) untuk Register. Pada detak clk selanjutnya, Register akan menangkap dan menyimpan nilai baru ini, sehingga current_count diperbarui dan siklus berulang.

Siklus umpan balik antara register penyimpan state dan ROM yang berisi logika transisi inilah yang membuat *counter* dapat berjalan secara otomatis sesuai urutan spesifik yang telah diprogram.



Hasil simulasi *behavioral* ini secara efektif memverifikasi fungsionalitas dari rancangan *counter* dengan urutan spesifik yang diimplementasikan menggunakan arsitektur berbasis ROM.

Simulasi diawali dengan sebuah pulsa reset aktif-tinggi yang secara benar menginisialisasi output count_out ke nilai awal, yaitu **0** (000). Setelah sinyal reset dilepaskan, dapat diamati bahwa count_out berubah pada setiap *rising edge* dari sinyal clk.

Urutan hitungan yang ditampilkan oleh count_out sepenuhnya konsisten dengan tabel ROM yang telah dirancang, yaitu: **0** → **1** → **3** → **5** → **7** → **2**, dan kemudian kembali lagi ke **0**. *Waveform* yang panjang menunjukkan bahwa siklus hitungan ini berulang secara terus-menerus dan stabil, membuktikan bahwa logika transisi dan mekanisme umpan balik (*feedback loop*) antara register dan ROM bekerja dengan benar. Dengan demikian, hasil simulasi ini memvalidasi bahwa model VHDL telah berhasil diimplementasikan sesuai dengan spesifikasi desain.

Bab 4 Kesimpulan

Berdasarkan perancangan, implementasi, dan verifikasi yang telah dilakukan, ketiga sistem digital sekuensial telah berhasil dibuat sesuai dengan spesifikasi masing-masing. Proyek ini secara efektif mendemonstrasikan:

- Implementasi **Mesin Mealy** untuk deteksi urutan berdasarkan state dan input saat ini.
- Perancangan **Mesin Moore** yang lebih kompleks untuk menangani output yang bersifat *stateful* dan memerlukan integrasi output ke dalam definisi state.
- Penggunaan **arsitektur berbasis ROM** sebagai metode yang efisien untuk merealisasikan *counter* dengan urutan non-sekuensial.

Verifikasi melalui simulasi dan implementasi perangkat keras membuktikan bahwa semua desain bekerja dengan benar, memvalidasi pemahaman konsep FSM dari teori hingga aplikasi praktis.

Daftar Pustaka

Brown, S., & Vranesic, Z. (2014). *Fundamentals of Digital Logic with VHDL Design* (3rd ed.). McGraw-Hill Education.

Mano, M. M., & Ciletti, M. D. (2018). *Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog* (6th ed.). Pearson.

Ashenden, P. J. (2008). *The Designer's Guide to VHDL* (3rd ed.). Morgan Kaufmann.