Orange Digital Center

Digital Design and FPGA Flow

Assignment 3
Week 3

Created By: Ahmed Mostafa Gomaa
Submitted to :
Dr. Tamer Saleh
Eng. Asmaa El-sayed
Eng. Nourhan

Table of Contents

1. Introduction **3**

2. Sequence Detector **4**

2.1. Introduction 4
2.2. Overlapping Sequence Detector For (1010110) 4
2.3. Moore 4
2.4. FSM Diagram 5
2.5. Code Verilog 5
2.6. RTL Viewer 6
2.7. Timing Diagram 6
2.8. Mealy 6
2.9. FSM Diagram 6
2.10. Code Verilog 7
2.11. RTL Viewer 7
2.12. Timing Diagram 7

3. Testbench **8**

3.1. Introduction 8
3.2. Testbench for Carry Lock ahead 8
3.3. Timimg Diagram 8

1. Introduction

Finite State Machines (FSMs)

A **Finite State Machine (FSM)** is a computational model used to design sequential logic circuits and systems. It consists of a finite number of states, transitions between those states, and actions associated with transitions or states. FSMs are widely used in digital design to model systems that respond to inputs over time, such as control units, communication protocols, and user interfaces.

An FSM is defined by:

1. **States**: The system's possible conditions or modes.
2. **Transitions**: Rules that dictate how the system moves from one state to another based on inputs.
3. **Actions**: Operations performed during transitions or while in a specific state.

FSMs can be classified into two types:

- **Moore Machine**: Outputs depend only on the current state.
- **Mealy Machine**: Outputs depend on both the current state and inputs.

Testbenches

A **testbench** is a virtual environment used to verify the functionality of a digital design, such as an FSM, through simulation. It provides input stimuli to the design and monitors its outputs to ensure it behaves as expected. Testbenches are essential for debugging and validating designs before they are implemented in hardware.

Key components of a testbench include:

1. **Input Stimuli**: Simulated inputs to drive the design.
2. **Design Under Test (DUT)**: The FSM or circuit being tested.
3. **Output Monitoring**: Observing and comparing outputs against expected results.
4. **Clock and Reset Signals**: Essential for synchronizing and initializing the design.

Why Combine FSMs and Testbenches?

FSMs are often used in complex systems where correct behavior is critical. A testbench allows designers to:

- Verify state transitions and outputs under various conditions.
- Detect and fix errors early in the design process.
- Ensure the FSM meets design specifications before implementation.

By simulating an FSM with a testbench, designers can confidently proceed to hardware implementation, knowing the design has been thoroughly tested. This combination is a cornerstone of reliable digital system design.

2. Sequence Detector

2.1. Introduction

A **sequence detector** is a fundamental component in digital systems used to identify a specific pattern of bits in a stream of input data. It plays a crucial role in communication systems, digital signal processing, and error detection mechanisms. Sequence detectors can be implemented using **finite state machines (FSMs)**, which transition between different states based on the incoming bit sequence.
There are two main types of sequence detectors:

1. **Overlapping Sequence Detector** – Detects a sequence even if part of it overlaps with the previous sequence.
2. **Non-Overlapping Sequence Detector** – Requires the sequence to restart after a successful detection.

These detectors are commonly built using flip-flops and logic gates in hardware or programmed using Verilog or VHDL in FPGA-based designs. Their applications include **data synchronization, protocol validation, and security systems** where specific bit patterns need to be identified efficiently.

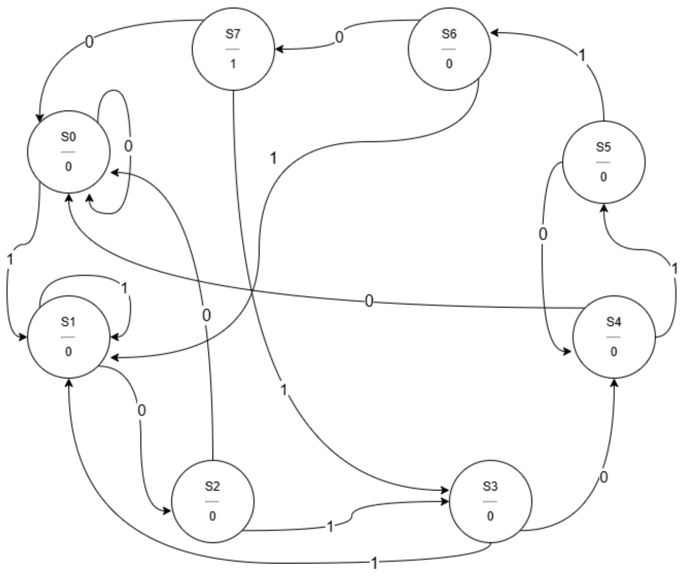
2.2. Overlapping Sequence Detector For (1010110)

2.3. Moore

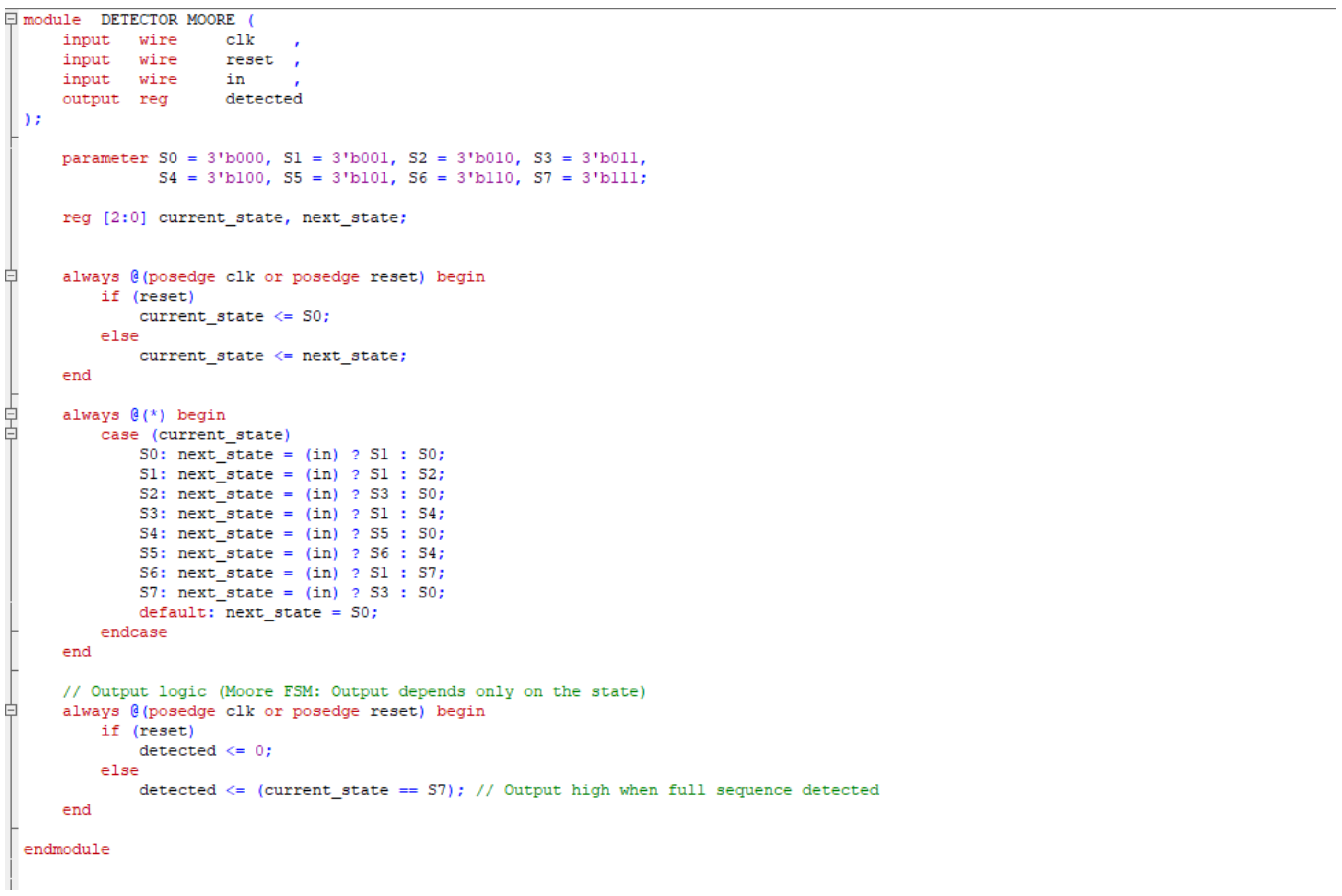
A **sequence detector** identifies a specific pattern in a stream of input bits. In this case, we are designing a **Moore finite state machine (FSM)** to detect the **"1010110"** sequence with overlapping allowed.
Moore Machine Overview
A **Moore FSM** determines its output based **only on the current state**, unlike a **Mealy FSM**, which depends on both the state and input. This means the output changes **only when transitioning to a new state** and is not directly affected by the current input.
State Diagram and Explanation

To detect the pattern **1010110** in an **overlapping manner**, we need to track partial matches and move through states accordingly. Each state represents the progress in matching the sequence.

2.4. FSM Diagram

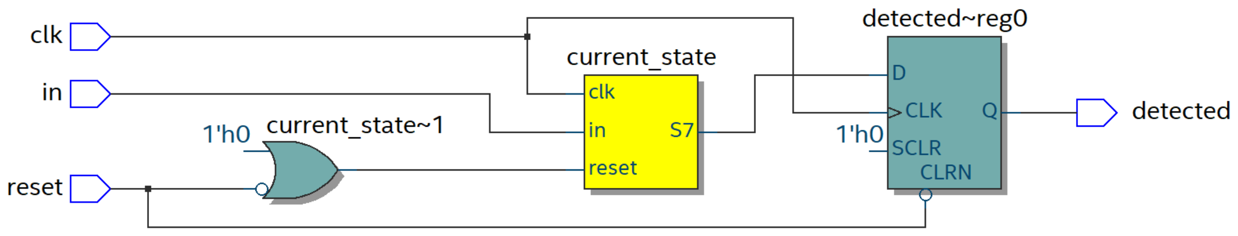


2.5. Code Verilog

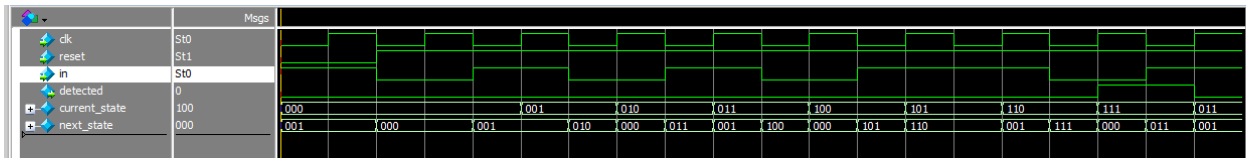


```
module  DETECTOR_MOORE (
    input   wire     clk    ,
    input   wire     reset  ,
    input   wire     in     ,
    output  reg      detected
);

    parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011,
              S4 = 3'b100, S5 = 3'b101, S6 = 3'b110, S7 = 3'b111;

    reg [2:0] current_state, next_state;


    always @(posedge clk or posedge reset) begin
        if (reset)
            current_state <= S0;
        else
            current_state <= next_state;
    end

    always @(*) begin
        case (current_state)
            S0: next_state = (in) ? S1 : S0;
            S1: next_state = (in) ? S1 : S2;
            S2: next_state = (in) ? S3 : S0;
            S3: next_state = (in) ? S1 : S4;
            S4: next_state = (in) ? S5 : S0;
            S5: next_state = (in) ? S6 : S4;
            S6: next_state = (in) ? S1 : S7;
            S7: next_state = (in) ? S3 : S0;
            default: next_state = S0;
        endcase
    end

    // Output logic (Moore FSM: Output depends only on the state)
    always @(posedge clk or posedge reset) begin
        if (reset)
            detected <= 0;
        else
            detected <= (current_state == S7); // Output high when full sequence detected
    end

endmodule
```

2.6. RTL Viewer



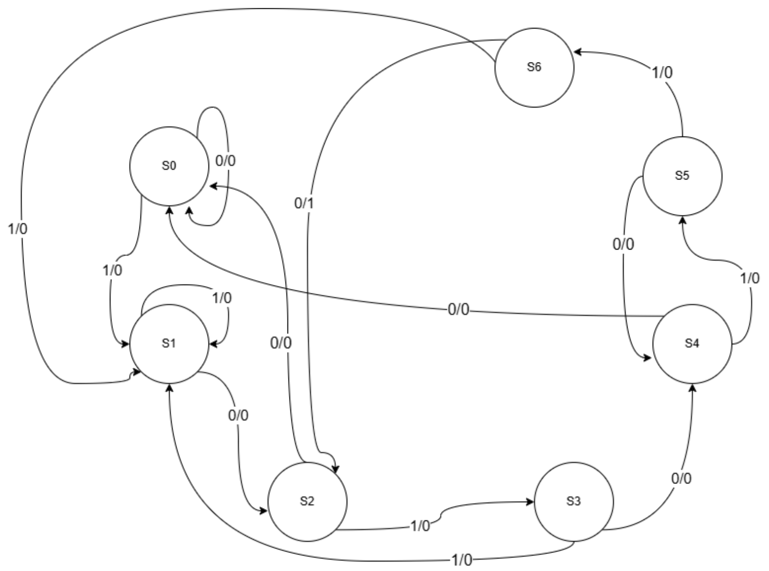
2.7. Timing Diagram



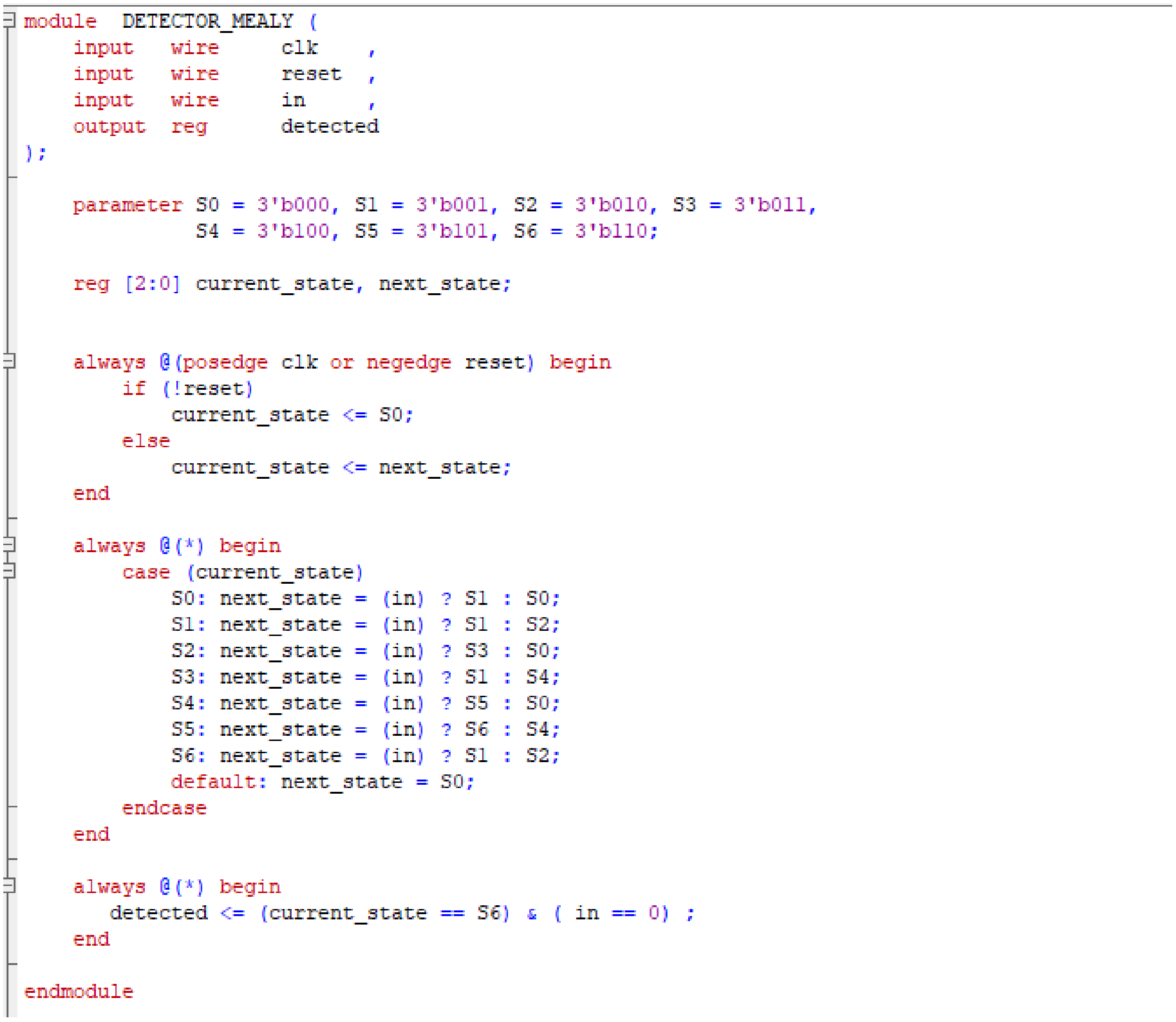
2.8. Mealy

An **overlapping sequence detector** is a digital system designed to identify a specific sequence of bits in a continuous stream of input data. Unlike non-overlapping detectors, an overlapping detector allows the detected sequence to share bits with subsequent sequences, making it more efficient for continuous data processing.
In this context, we focus on designing a **Mealy Machine** to detect the sequence **"1010110"**. A Mealy Machine is a type of Finite State Machine (FSM) where the outputs depend on both the current state and the inputs. This makes it particularly suitable for sequence detection, as the output can immediately reflect whether the desired sequence has been detected.

2.9. FSM Diagram

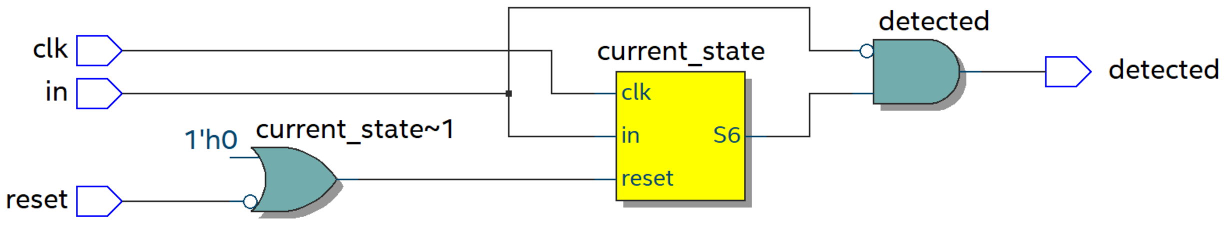


2.10. Code Verilog

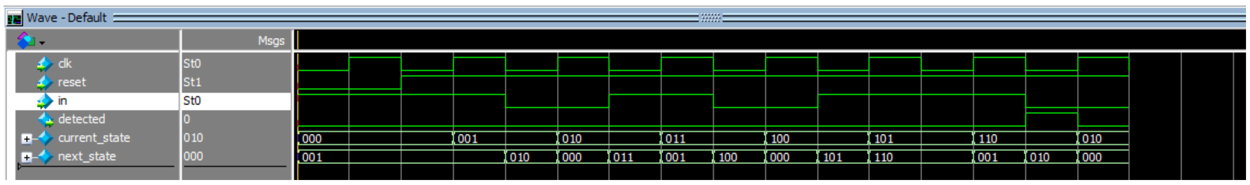


```
module  DETECTOR_MEALY (
    input   wire     clk    ,
    input   wire     reset  ,
    input   wire     in     ,
    output  reg      detected
);

    parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011,
              S4 = 3'b100, S5 = 3'b101, S6 = 3'b110;

    reg [2:0] current_state, next_state;


    always @(posedge clk or negedge reset) begin
        if (!reset)
            current_state <= S0;
        else
            current_state <= next_state;
    end

    always @(*) begin
        case (current_state)
            S0: next_state = (in) ? S1 : S0;
            S1: next_state = (in) ? S1 : S2;
            S2: next_state = (in) ? S3 : S0;
            S3: next_state = (in) ? S1 : S4;
            S4: next_state = (in) ? S5 : S0;
            S5: next_state = (in) ? S6 : S4;
            S6: next_state = (in) ? S1 : S2;
            default: next_state = S0;
        endcase
    end

    always @(*) begin
       detected <= (current_state == S6) & ( in == 0) ;
    end

endmodule
```

2.11. RTL Viewer



2.12. Timing Diagram



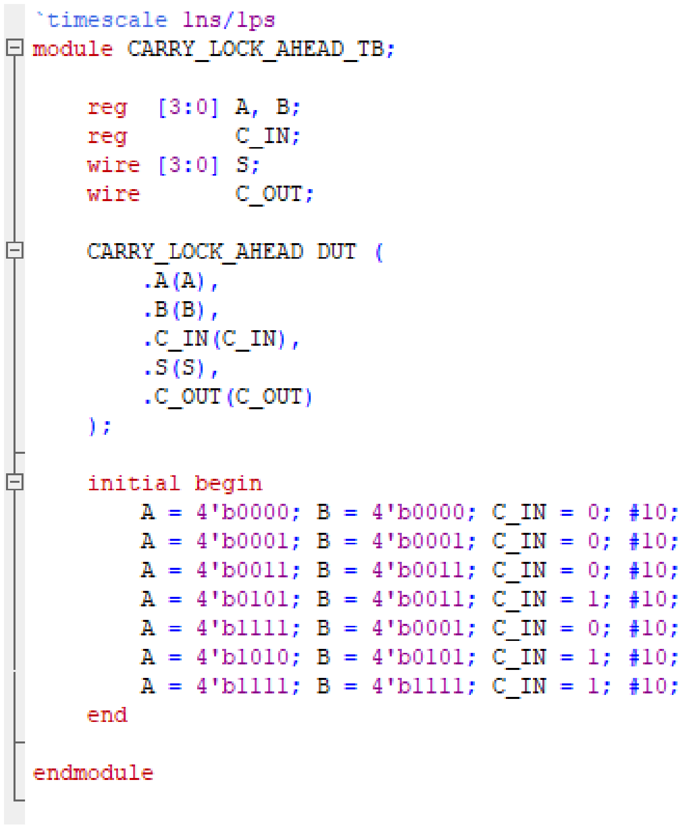
```
Wave - Default
                 Msgs
clk              St0
reset            St1
in               St0
detected         0
current_state    010   000  001  010  011  100  101  110  010
next_state       000   001  010  000  011  001  100  000  101  110  001  010  000
```

3. Testbench

3.1. Introduction

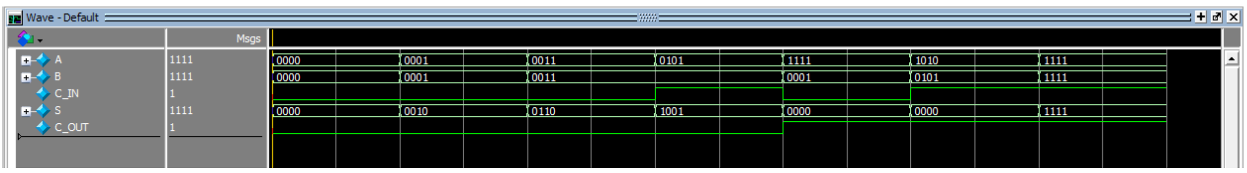
A **testbench** is a critical tool in digital design and verification, used to simulate and validate the functionality of a hardware design before it is implemented in physical hardware. It acts as a virtual environment where designers can apply input stimuli to a **Design Under Test (DUT)** and observe its outputs to ensure it behaves as expected. Testbenches are essential for debugging, verifying correctness, and ensuring that the design meets its specifications.

3.2. Testbench for Carry Lock ahead



```
`timescale 1ns/1ps
module CARRY_LOCK_AHEAD_TB;

    reg  [3:0] A, B;
    reg        C_IN;
    wire [3:0] S;
    wire       C_OUT;

    CARRY_LOCK_AHEAD DUT (
        .A(A),
        .B(B),
        .C_IN(C_IN),
        .S(S),
        .C_OUT(C_OUT)
    );

    initial begin
        A = 4'b0000; B = 4'b0000; C_IN = 0; #10;
        A = 4'b0001; B = 4'b0001; C_IN = 0; #10;
        A = 4'b0011; B = 4'b0011; C_IN = 0; #10;
        A = 4'b0101; B = 4'b0011; C_IN = 1; #10;
        A = 4'b1111; B = 4'b0001; C_IN = 0; #10;
        A = 4'b1010; B = 4'b0101; C_IN = 1; #10;
        A = 4'b1111; B = 4'b1111; C_IN = 1; #10;
    end

endmodule
```

3.3. Timimg Diagram


Wave - Default
Msgs
A 1111 0000 0001 0011 0101 1111 1010 1111
B 1111 0000 0001 0011 0001 0101 1111
C_IN 1
S 1111 0000 0010 0110 1001 0000 0000 1111
C_OUT 1