# ECE 385

Spring 2024

Experiment #

# Lab5 : An 8-bit Multiplier in SV

Name:Jie Wang, Shitian Yang

Student ID: 3200112404, 3200112415

Prof. Chushan Li, Prof. Zuofu Cheng

ZJU-UIUC Institute

March 15, 2024, Friday D-225

TA: Jiebang Xia

**Demo Point: 5/5**

## 1.Introduction

## 2.Prelab Question

### 1. Extension from 4-bit to 8-bit:

We carefully modified the SystemVerilog code for key modules including ***`Control.sv`***, ***`Register\_unit.sv `,`Reg\_4.sv` and `Processor.sv `***. We then integrated an existing ***`testbench\_8.sv`*** file, ensuring it was fully compatible with our enhanced 8-bit processor. Our modifications were compiled and subjected to simulation, which we completed successfully with no errors, confirming the functional correctness of our updates.

**Fig-1:** RTL Viewer of Our Circuit

### 2.Adders Implementation:

Most of our time are spent on the document reading and environment configuration. We finished the adder code modification very soon, but it is time-consuming to understand the path setting and usage of testbench. Thankfully, we finished the lab in time with the assistance of our fellow classmates and [the detailed blog written by a formal ECE385 TA.](https://kttechnology.wordpress.com/)

Detail information please read Operation of the Adder Circuit.

### 3. Performance analysis(Design Analysis Comparison Results)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Carry-Ripple | Carry-Lookahead | Carry-Select |
| Memory (BRAM) | **1** | **1** | **1** |
| Frequency | **1** | **1.284** | **1.1** |
| Total Power | **1** | **1.02** | **1** |

**Table-1 : Memory, Frequency and Power Comparison**

**Table-2 : Memory, Frequency and Power Comparison**

## 3. Bit-Serial Logic Processor

**Fig 2:** FSM Viewer Output, including 4 more state G, H, I, J

**Fig-3: 0-1000ns Waveform, ErrorCnt == 0**

**Fig-4: Passed all the testcase in testbench\_8.sv**

## 4.Operation of the Adder Circuit

### Overview:

As demonstrated in Fig-3, The data flow in our adders can be considered as below:

**Fig-5:** RTL Viewer of Adder Circuit

### Ripple adder:

1.**Description:**

The Ripple Carry Adder is a basic digital adder design that implements addition of multi-bit binary numbers. It is composed of a series of one-bit full adders connected in a chain.

2.**Description of the Operation:**

In an RCA, each full adder takes in two corresponding bits from the input numbers and the carry-out from the previous adder as its input. The least significant bit adder receives the initial carry-in (often set to 0). The sum output from each adder forms one bit of the final result, and the carry-out is passed to the next most significant bit's full adder. This process "ripples" from the least significant to the most significant bit, hence the name.

3.**Features:**

Simple and straightforward design.

Easy to implement and understand.

Scalable to any bit-size by adding more full adders.

The delay is proportional to the number of bits due to the ripple effect, leading to slower operation for large bit sizes.

**4.** **RTL View**

**Fig-6:** RTL Viewer of Ripple Adder Circuit

5.**Schematic Block Diagrams**

**Fig-7:** Schematic Block Diagrams of Ripple Adder Circuit

**6.Purpose and Operation of Each Module**

**1) Module: ripple\_adder**

**Purpose:**

The ripple\_adder module is designed to perform a 16-bit addition by employing a series of 4-bit adders, utilizing the ripple carry technique.

**Operation:**

The module instantiates four 4-bit full adders (full\_adder\_4bit) to compute the sum of two 16-bit inputs A and B. Each 4-bit adder receives a carry-in from the previous adder's carry-out, starting with an initial carry-in of 1'b0 for the least significant bits.

**Inputs/Outputs:**

**Inputs**:

A[15:0]: First 16-bit operand

B[15:0]: Second 16-bit operand

**Outputs**:

Sum[15:0]: 16-bit sum of A and B

CO: Carry-out of the most significant bit addition

1. **Module: full\_adder\_4bit**

**Purpose:**

The full\_adder\_4bit module is a component used in the ripple\_adder to add 4-bit slices of the input operands along with a carry-in bit.

**Operation:**

This module contains four instances of the full\_adder module, each performing a 1-bit addition. The carry-out of each bit addition is passed as the carry-in to the next significant bit.

**Inputs/Outputs:**

**Inputs:**

x[3:0]: First 4-bit operand slice

y[3:0]: Second 4-bit operand slice

cin: Carry-in from the previous less significant slice

**Outputs:**

s[3:0]: 4-bit sum of x, y, and cin

cout: Carry-out to the next more significant slice

1. **Module: full\_adder**

**Purpose:**

The full\_adder module is the fundamental building block that adds two single bits along with a carry-in bit to produce a sum bit and a carry-out bit.

**Operation:**

The module performs bit-wise addition using XOR gates to compute the sum and AND, OR gates to determine the carry-out.

**Inputs/Outputs:**

**Inputs:**

x: First operand bit

y: Second operand bit

cin: Carry-in bit

**Outputs:**

s: Sum bit of x, y, and cin

cout: Carry-out bit indicating if there is an overflow out of the bit addition

### Lookahead adder:

1**.Description:**

The Carry Lookahead Adder is an advanced type of adder that improves upon the RCA by reducing the carry propagation delay.

**Fig-8: A 4x4-bit Hierarchical Carry-Lookahead Adder Block Diagram**

2.**Description of the Operation:**

The CLA uses the concepts of 'generate' and 'propagate' to predict the carry-out of each bit without waiting for the previous bit's carry-out. The generate function determines if a carry will be generated by a pair of bit additions, and the propagate function determines if a carry will be passed through. These functions are used to quickly calculate carries for each bit, allowing for simultaneous sum calculation across all bits.

3.**Features:**

Faster than the RCA due to reduced carry propagation delay.

More complex design as it involves additional logic for generate and propagate functions.

Better suited for high-speed operations and large bit-size additions.

Typically consumes more area on a chip due to the additional logic required.

**4.** **RTL View**

**Fig-9:** RTL Viewer of Lookahead Adder Circuit

5.**Schematic Block Diagrams**

**Fig-10:** Schematic Block Diagrams of Lookahead Adder Circuit

1. **Purpose and Operation of Each Module**
2. **Module: carry\_lookahead\_adder**

**Description:**

The carry\_lookahead\_adder module implements a 16-bit adder using the carry-lookahead logic to improve the speed of binary addition by reducing the carry propagation delay between consecutive full adders.

**Operation:**

The module consists of four instances of compute\_4bit\_PG\_GG that compute the propagate and generate signals for each 4-bit block of inputs A and B. The carry\_lookahead\_adder\_4bit\_helper\_compute\_carry computes the carry-out signals for each block, which are then used by instances of carry\_lookahead\_adder\_4bit to calculate the final sum.

**Inputs/Outputs:**

**Inputs:**

A[15:0]: First 16-bit operand.

B[15:0]: Second 16-bit operand.

**Outputs:**

Sum[15:0]: 16-bit sum of A and B.

CO: Carry-out signal representing an overflow out of the most significant bit.

1. **Module: compute\_4bit\_PG\_GG**

**Purpose:**

The compute\_4bit\_PG\_GG module computes the propagate and generate signals for a 4-bit block of the adder. These signals are essential for the carry-lookahead logic to determine the carry for each bit without waiting for the previous bits.

**Operation:**

It calculates the block-level propagate signal PG by ANDing all the propagate signals, and the block-level generate signal GG by ORing all the individual generate signals conditioned on the propagate signals.

**Inputs/Outputs:**

**Inputs:**

P[3:0]: Propagate signals for a 4-bit block.

G[3:0]: Generate signals for a 4-bit block.

**Outputs:**

PG: Block-level propagate signal.

GG: Block-level generate signal.

1. **Module: carry\_lookahead\_adder\_4bit**

**Purpose:**

This module performs 4-bit binary addition using carry-lookahead logic to compute the sum and carry-out signals for the given inputs and carry-in.

**Operation:**

The module generates individual generate G and propagate P signals, which are used by carry\_lookahead\_adder\_4bit\_helper\_compute\_carry to compute the carry signals C. The sum is then computed using the propagate signals and carry signals.

**Inputs/Outputs:**

**Inputs:**

A[3:0]: First 4-bit operand.

B[3:0]: Second 4-bit operand.

C\_in: Carry-in signal.

**Outputs:**

Sum[3:0]: 4-bit sum of A and B.

CO: Carry-out signal.

1. **Module: carry\_lookahead\_adder\_4bit\_helper\_compute\_carry**

**Purpose:**

The purpose of this module is to compute the carry signals for a 4-bit block of the carry-lookahead adder.

**Operation:**

It calculates carry signals based on individual generate G, propagate P, and the input carry-in C\_in.

**Inputs/Outputs:**

**Inputs:**

P[3:0]: Propagate signals for the 4-bit block.

G[3:0]: Generate signals for the 4-bit block.

C\_in: Input carry signal.

Outputs:

C[3:0]: Carry signals for the 4-bit block.

### C. Select adder:

1.**Description:**

The Carry Select Adder is designed to improve the speed of addition by speculatively calculating two possible results for each bit addition, based on the assumption of the carry-in being either 0 or

**Fig-11: 16-bit Carry-Select Adder Block Diagram**

2.**Description of the Operation:**

The CSA divides the input numbers into blocks and computes two sums for each block, one assuming the carry-in is 0, and the other assuming it is 1. Once the actual carry-in is known, the correct sum is selected using multiplexers. This allows the CSA to begin computing sums for the next block without waiting for the carry-out of the previous block.

3.**Features:**

Provides a good trade-off between speed and complexity.

Utilizes extra hardware to speculate and select correct sums, leading to an increase in power consumption and chip area.

Well-suited for medium-sized bit-width operations where speed is a concern but area and power are not critical constraints.

**4.** **RTL View**

**Fig-12:** RTL Viewer of Select Adder Circuit

5.**Schematic Block Diagrams**

**Fig-13:** Schematic Block Diagrams of Select Adder Circuit

1. **Purpose and Operation of Each Module**
2. **Module: carry\_select\_adder**

**Description:**

The carry\_select\_adder module is an optimized adder design that improves upon the traditional ripple carry adder's performance by speculatively computing sums and carry-outs for both possible incoming carry values.

**Operation:**

The module is made of a series of 4-bit carry-select adder units (carry\_select\_adder\_4bit\_unit). The first 4-bit adder is a regular full adder that starts with an initial carry-in of 1'b0. For subsequent 4-bit blocks, carry-select adder units are used which compute two possible results for each block based on the assumption that the carry-in is either 1'b0 or 1'b1. The correct result is selected once the carry-in is known.

**Inputs/Outputs:**

**Inputs:**

A[15:0]: First 16-bit operand.

B[15:0]: Second 16-bit operand.

Outputs:

Sum[15:0]: 16-bit sum of A and B.

CO: Carry-out of the most significant bit addition.

1. **Module: carry\_select\_adder\_4bit\_unit**

**Purpose:**

Each carry\_select\_adder\_4bit\_unit serves as a building block of the carry\_select\_adder and computes two sets of sums and carry-outs based on both possible carry-in values.

**Operation:**

The module instantiates two 4-bit full adders (full\_adder\_4bit), one assuming carry-in is 1'b0 and another assuming 1'b1. After both adders compute their respective sums and carry-outs, a multiplexer selects the correct sum and carry-out based on the actual carry-in, C\_in.

**Inputs/Outputs:**

**Inputs:**

A[3:0]: First 4-bit operand slice.

B[3:0]: Second 4-bit operand slice.

C\_in: Carry-in from the previous less significant slice.

**Outputs:**

Sum[3:0]: 4-bit sum of A and B.

CO: Carry-out to the next more significant slice.

## 5.Post-lab Questions

1. **Compare the usage of LUT, Memory, and Flip-Flop of your bit-serial logic processor exercise in the IQT with your TTL design in Lab 3. Make an educated guess of the usage of these resources for TTL assuming the processor is extended to 8-bit. Which design is better, and why?**

**Bit-Serial Logic Processor (SystemVerilog):**

LUT Usage: Bit-serial processors typically require a moderate number of LUTs because each operation is performed serially.

Memory Usage: Memory requirements are generally lower for bit-serial processors as they often use registers to store intermediate values.

Flip-Flop Usage: Flip-flops are used extensively in a bit-serial processor to hold the state of each bit being processed, as well as control signals for each cycle of the serial operation.

**TTL Design:**

LUT Usage: In a TTL implementation, LUTs are not a resource because TTL technology does not use configurable logic blocks.

Memory Usage: TTL designs might use separate memory chips if needed, but typically they would be built using discrete logic without large-scale on-chip memory.

Flip-Flop Usage: Flip-flops would still be used in a TTL design, but they would be discrete components rather than integrated parts of configurable logic blocks

The SystemVerilog implementation would be better. It would take up less physical space, consume less power, and likely operate at higher speeds due to the reduced propagation delay of integrated circuits compared to discrete TTL components.

1. **For the adders, refer to the Design Resources and Statistics in IQT.16-18 and complete**

**the following design statistics table for each adder. This is more comprehensive than the above design analysis and is required for every SystemVerilog circuit.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Ripple Adder** | **Lookahead Adder** | **Select Adder** |
| **LUT** | 117 | 123 | 129 |
| **DSP** | 0 | 0 | 0 |
| **Memory** | 0 | 0 | 0 |
| **Flip-Flop** | 105 | 105 | 105 |
| **Frequency / MHz** | 184.23 | 236.57 | 204.16 |
| **Static Power / mW** | 98.5 | 98.5 | 98.5 |
| **Dynamic Power / mW** | 0 | 3.1 | 0 |
| **Total Power / mW** | 139.11 | 142.2 | 139.11 |

1. **Observe the data plot and provide explanation to the data, i.e., does each resource breakdown comparison from the plot makes sense? Are they complying with the theoretical design expectations, e.g., the maximum operating frequency of the carry-lookahead adder is higher than the carry-ripple adder? Which design consumes more power than the other as you expected, why?**

**Expected:**The power consumption and operating frequency are correlated with the complexity of the design and the efficiency of logic implementation. The CLA should have the highest frequency but also the highest power consumption, while the CRA should show the opposite. The CSA's performance and power usage should fall between the two. If the plot reflects these relationships, it aligns with theoretical expectations. Any deviations would warrant a closer examination of the design implementation and potential optimizations.

**Real**: The Lookahead Adder seems to offer the best performance in terms of speed, with a slight increase in total power consumption that can be justified by its significant improvement in operating frequency. The Ripple Adder, while being the least resource-intensive and the most power-efficient, offers the slowest performance. The Select Adder provides a middle ground, with better performance than the Ripple Adder without a significant power penalty. This analysis demonstrates that the trade-offs between speed, resource usage, and power consumption are consistent with the theoretical expectations for these adder designs.

## 6. Bug Log

* **Description of all bugs encountered, and corrective measures taken:**

## 7.Conclusion

Through lab5, we practiced on the overall deployment of System Verilog together with the

performance analysis as below:

# 8. References

[1] KTTECH. (2017, January 31). ECE 385 Lab 5: An 8-bit Multiplier in SV. Retrieved from <https://kttechnology.wordpress.com/2017/02/10/ece-385lab5-an-8-bit-multiplier-in-sv/> Teaching Assistant Blog

[2] ECE385 Faculty. (n.d.). [Lab 5 description](https://learn.intl.zju.edu.cn/bbcswebdav/pid-101276-dt-content-rid-1361038_1/xid-1361038_1)

[3] ECE385 Faculty. (n.d.). [Introduction to SystemVerilog (pdf)](https://learn.intl.zju.edu.cn/bbcswebdav/pid-101280-dt-content-rid-1361044_1/xid-1361044_1)

[4] ECE385 Faculty. (n.d.). [Introduction to Quartus Prime in the lab manual.](https://learn.intl.zju.edu.cn/bbcswebdav/pid-101280-dt-content-rid-1361046_1/xid-1361046_1)