CSCE 2301, 02 Project II Report Spring 2023

Sequential 8-bit Signed Multiplier (8x8)

Omar Bahgat 900211747

Youssef Mansour 900212652

Mohammad Alashkar 900214276

Mohamed Abbas 900211252

CSCE 2303, 02

Department of Computer Science and Engineering, AUC

Table of Contents

Brief Description of the Project:

The project is focused on implementing an 8-bit signed sequential multiplier. The circuit takes two 8-bit inputs, the multiplier and the multiplicand and a "start" signal and a clock signal “clk”. The goal is to produce a 16-bit output, the product, which represents the result of multiplying the multiplier and the multiplicand. The multiplication process follows the shift and add algorithm. Initially, when the "start" signal is triggered, the multiplier and multiplicand values are loaded into separate shift registers. Simultaneously, the product is initialized to zero. This algorithm utilizes a T Flip-Flop module that acts as a toggle unit. It generates a signal to control the execution of two different sections of the shift and add algorithm. The project also incorporates other necessary modules to ensure proper functionality. A push button module is implemented to detect button presses reliably. It includes sub-modules such as a clock divider, debouncer, synchronizer, and rising edge detector to filter out noise, synchronize signals, and detect rising edges. These modules work together to provide a clean and accurate signal when a button is pressed. Additionally, the project includes a scanner module that is responsible for the complex task of controlling a seven-segment display. It includes clock division, magnitude conversion, BCD conversion, and segment display control functionalities. The shift module is also employed to handle shifting and rotating a 20-bit value based on control signals.

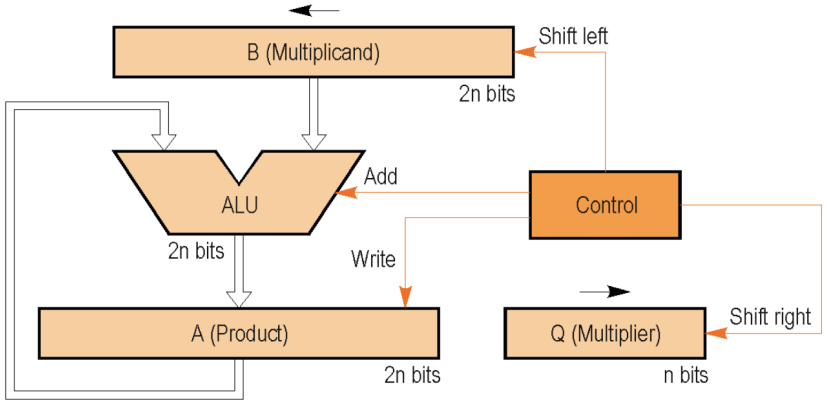
Program Design:

The program is developed into two main components: the multiplier and the display each with its own subcomponents.

1) Multiplier Part:

- **Multiplier:**

This module represents a multiplier circuit. It takes two 8-bit inputs, the **"multiplier"** and **"multiplicand"**, along with a **"start"** signal and a **"clk"** signal and produces a 16-bit output **"product"**. The "start" signal is generated by a pushbutton and once we click it, the multiplier and multiplicand are loaded into the 8-bit and 16-bit shift registers respectively while the product is initialized to 0. The multiplication process itself follows the shift and add algorithm which is the reason why the multiplier and the multiplicand are loaded into shift registers. Regarding the shift and add algorithm, we created a **TFF** (T Flip-Flop) module that is designed as a toggle unit which is responsible for generating a signal that controls the execution of the two different sections of the shift and add algorithm. The first part is to check the LSB of the multiplier and if it's equal to 1, we load the product with (product + multiplicand); otherwise, the product remains unchanged. The second part of the algorithm is shifting the multiplier one bit to the right and the multiplicand one bit to the left.



- **T Flip-Flop:**

  This module represents a T Flip-Flop circuit. It takes a single-bit input "1" and a clock signal "clk" and produces a single-bit “out”. It toggles the output signal “out” on the positive edge of the clock signal when the input is high. If the output is high, the first section of the algorithm is executed, and if it’s low, the second section is executed. By designing the shift and algorithm using a T flip-flop, we prevented the two parts of the algorithm from happening simultaneously as each part occurs on the positive edge of the clock signal where the first part happens if the signal generated by the T flip-flop is high and the second part happens if the signal is low.

- **Push Button**

  The push buttom module incorporates other sub-modules to determine if a button has been pressed or not. At the beginning of the module, a clock divider sub-module called "clockDivider" is instantiated and it takes the clock signal "clk" and the reset signal "rst" as inputs, and generates an intermediate clock signal "clk2.” which is a slower output clock signal. Then, the input signal "in" and “clk2” are connected to a debouncer module that filters out noise and bouncing effects often encountered in pushbutton signals and produces a filtered output signal “w”. Next, another sub-module the synchronizer takes the filtered signal "w," and “clk2” and generates an intermediate adjusted signal "w2." Finally, a rising edge detector sub-module takes the “clk2” and the the adjusted signal "w2," and produces the output signal "out" indicating the occurrence of a rising edge.

  In simple terms, this module represents a pushbutton circuit that debounces and detects the rising edge of a button press. It ensures a clean and reliable signal by filtering out

noise and bounces, and generates an output signal when a button press is detected. The circuit also incorporates a reset phase at the beginning to ensure proper initialization.

- **Clock Divider:**

The clock divider module is a circuit that takes as an input a clock signal and produces a new clock signal with a lower frequency. It divides the frequency of the input clock signal by a certain factor "n". The factor determines the division ratio, which specifies how many input clock cycles are required to generate a single output clock cycle. The input and output frequencies follow the following formula:

$$f_{out} = \frac{f_{in}}{2n}$$

The clock divider typically consists of a counter that keeps track of the number of input clock cycles that have occurred. As the counter increments with each rising edge of the input clock signal, it compares its value to the division ratio. When the counter reaches the desired value, it triggers the generation of an output clock pulse. Once the output clock pulse is generated, the counter is reset to its initial value, and the counting process starts again. This ensures a continuous and repetitive generation of the output clock signal. In our project, clock dividers are mainly for generating clock signals for different modules with varying frequency requirements.

- **Synchronizer:**

The synchronizer module is used to ensure that data transitions from one clock domain to another in a controlled manner. When data is transferred from one clock domain to

another, it can lead to issues like metastability, where the receiving component may misinterpret the data due to timing mismatches. Thus, the uses flip-flops to introduce controlled delays to the data signal. When the data arrives at the synchronizer, it is first captured by a flip-flop in the source clock domain. Then, the synchronized data is passed through one or more additional flip-flops in the destination clock domain to further synchronize it to the new clock. By introducing these stages, the synchronizer reduces the risk of metastability.

- **Debouncer:**

The debouncer module ensures a clean and reliable signal when a button or switch is pressed. When you press a button, it doesn't make a single clean contact right away. Instead, it bounces back and forth which can lead to glitches and a single button press might be interpreted s as multiple presses. The debouncer filters out these bounces and provides a stable signal preventing unintended multiple inputs, ensuring reliable operation.

- **Rising Edge Detector:**

The rising edge detector module detects and signals the occurrence of a rising edge in a digital signal. A rising edge specifically detects the transition from a low (0) to a high (1). It monitors the input signal and generates an output signal when it detects a rising edge and it does so by comparing the present state with the past state. The output signal from the rising edge detector can be used to trigger varous processes in the digital system such as the push button detection in our project.

2) Display Part:

- **Scanner**

  The scanner module is the most complex module in our and project and incorporates various sub-modules. It has clock input, control signals for left, right, and load, and input signals for the multiplier and the multiplicand. It also has a push button input and several output signals. The module starts with an asynchronous reset to initialize the circuit and clock divider sub-module is instantiated to generate a slower clock signal from the input clock. Then, an 8-bit two's complement module is used to convert the multiplier and the multiplicand into their magnitude. A multiplier module is then instantiated, which takes the converted signals along with the push button input and clock signal to calculate the product. The product value is then processed by a BCD converter module to obtain individual digit values (D4 to D0) to be displayed on the seven segment display. Finally, a case statement is used to select the appropriate segment value based on the generated sequence from the counter module. The selected segment value is then fed into a Seven-Segment Decoder module (SevSeg) which produces output signals for the seven-segment display and active anode selection.

- **Shift**

  The shift module is responsible for shifting and rotating a 20-bit value based on control signals. It has clock and control inputs, along with five 4-bit data inputs and a 20-bit output. If the "load" control signal is active, the module loads the input data (D0 to D4) into specific sections of the output register. If the "left" signal is active, the module rotates the contents of the output register to the left by 4 bits and if the "right" signal is active, the module rotates the contents of the output register to the right by 4 bits.

Validation Activities:

- 00000000 * 00000000
    - 0 * 0 = 0
- 00000000 * 00000010
    - 0 * 2 = 0
- 00000000 * 11111100
    - 0 * -4 = 0
- 00000001 * 00000101
    - 1 * 5 = 5
- 00000001 * 11111101
    - 1 * -3 = -3
- 00000111 * 11111011
    - 7 * -5 = -35
- 01100010 * 01001100
    - 98 * 76 = 7448
- 10000101 * 10011101
    - -128 * 127 = 12177
- 01111111 * 10000000
    - 127 * -128 = -16256

Contributions:

Due to the complexity of this project, we always worked together. Each person has done a part in every component of the project (block diagram, logisim, codes, report, readme).