VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING


# LOGIC DESIGN PROJECT

---

## DIGITAL SIGNAL PROCESSING MODULE USING VERILOG

**Advisor:** Trần Ngọc Thịnh
**Studetn:** Đỗ Quang Khanh 2053106

HO CHI MINH CITY, JUNE 2023

# Contents

# 1 Introduction

Design a simple Digital Signal Processor (DSP) using the Verilog language. This DSP module will process a discrete signal sequence, including the following functionalities:

- Receive a discrete signal sequence over time represented as a sequence of numbers with a predefined size N. It can operate in two modes: 32-bit floating-point or 32-bit integer.
- Perform Shifting operation: Shifting means shifting the signal around the X or Y axis by a unit K. In detail:
  - Time Shifting: Shift the signal along the time domain, around the Y-axis.
  - Amplitude Shifting: Shift the signal along the amplitude domain, around the X-axis.
- Perform Scaling operation: Scaling means multiplying the signal in the time or amplitude domain by a constant factor T. Specifically:
  - Time Scaling: Multiply the signal by a constant on the time axis. If $0 < T < 1$, it is called time compression; otherwise, if $T > 1$, it is time expansion.
  - Amplitude Scaling: Multiply the signal by a constant on the amplitude domain. Depending on the value of the constant, the amplitude of the signal can be increased or decreased.
- Perform Reversal operation: Reversal generates the inverse signal of the original signal. In detail:
  - Time Reversal: Reverse the signal along the time domain, creating its time-reflected signal around the Y-axis.
  - Amplitude Reversal: Reverse the signal along the amplitude domain, creating its amplitude-reflected signal around the X-axis.
- Perform Convolution of two signals in the time domain: This is equivalent to multiplying two signals in the frequency domain and is defined as follows:

$$y[n] = x_1[n] * x_2[n] = \sum_{k=-\infty}^{+\infty} x_1[k]x_2[n-k]$$

# 2 Design Implementation

The DSP (Digital Signal Processing) module in this project is designed to process discrete-time signal sequences using the Verilog hardware description language. It provides various functionalities such as shifting, scaling, reversal, and convolution for discrete signals. The module takes in an input signal, a mode selection, and a constant value for scaling or shifting. It generates the output signal based on the selected mode.

## 2.1 Input Signal

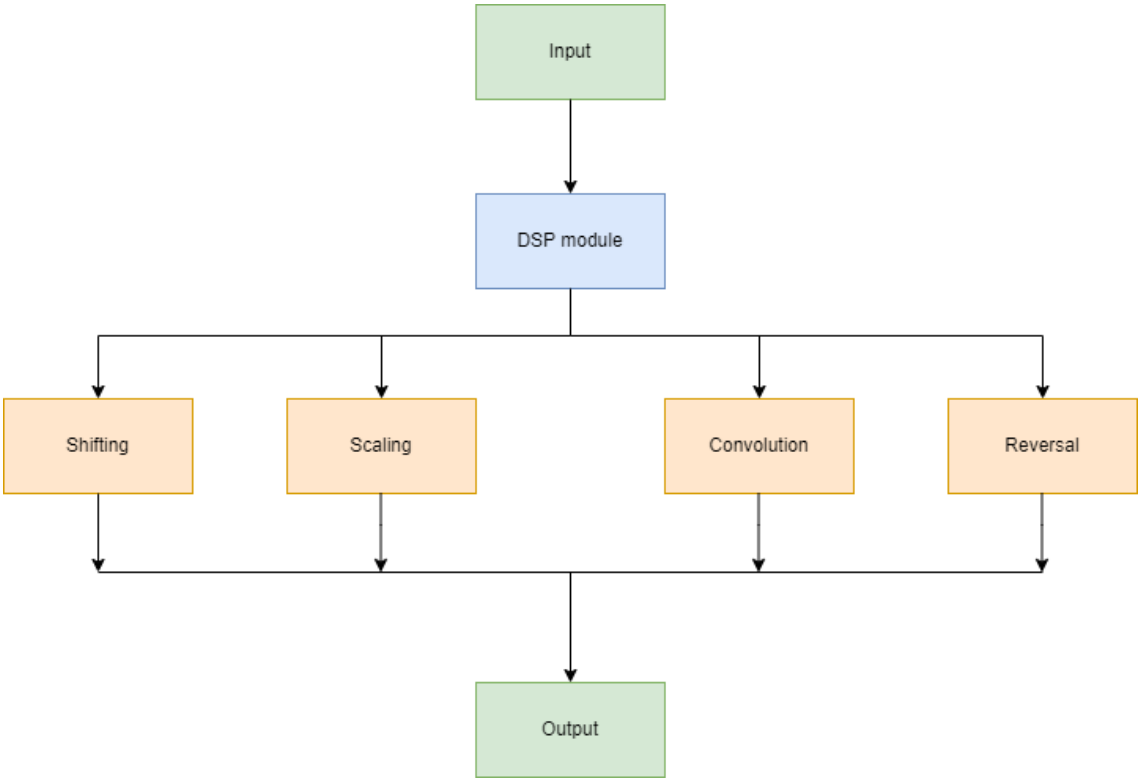
- input_signal: Two 32-bit discrete-time signals represented as a sequence of numbers.
- mode: A 3-bit mode selection that determines the type of operation to be performed.
  - 3'b000: Time Shifting

  - 3'b001: Amplitude Shifting
  - 3'b010: Time Scaling
  - 3'b011: Amplitude Scaling
  - 3'b100: Time Reversal
  - 3'b101: Amplitude Reversal
  - 3'b110: Convolution 2 different signals
- constant: A 32-bit constant value used for scaling or shifting operations.

## 2.2 Output

- output_signal: A 32-bit output signal representing the result of the chosen operation.

## 2.3 Block Diagram



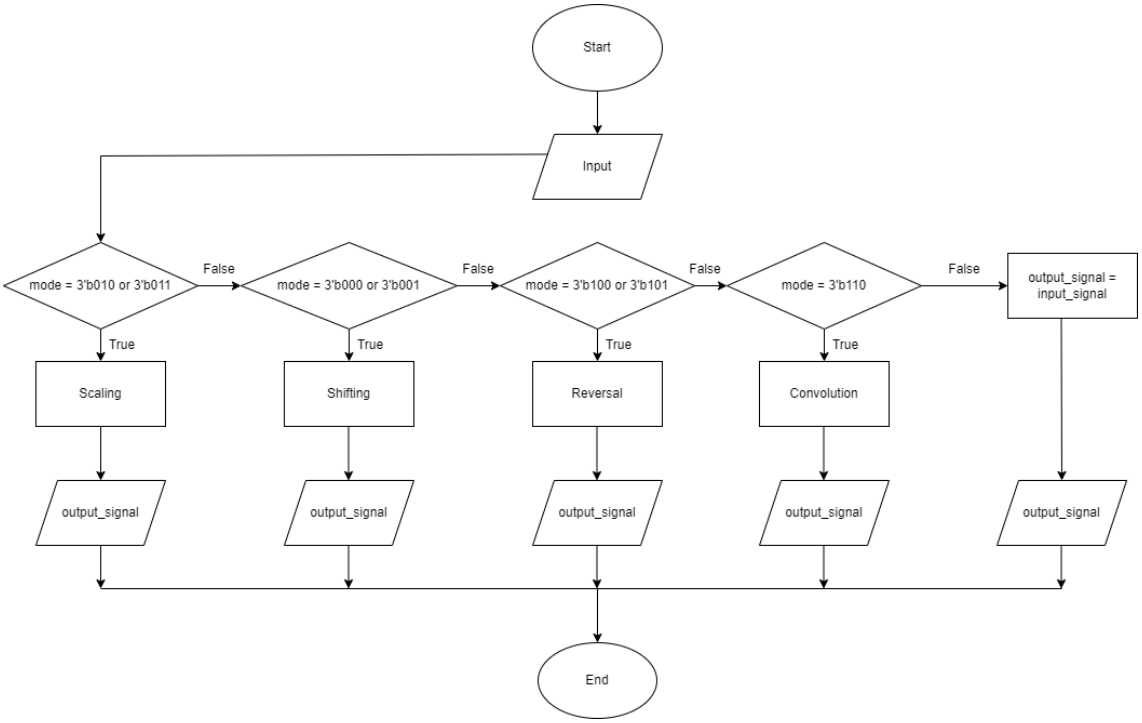

The block diagram represents the different functional blocks within the DSP module. The input signal flows through Shifting, Scaling, Reversal and Convolution block. After that, the output signal is generated at the final block. The module includes the following blocks:

1. Shifting: This block performs shifting operations on the input signal, either in the time domain or the amplitude domain.
2. Scaling: This block applies scaling operations to the input signal, adjusting the amplitude or time scale.

3. Reversal: This block generates the reversed version of the input signal, either in the time domain or the amplitude domain.
4. Convolution: This block performs the convolution operation on the input signal, combining it with another signal.
5. Output: This block represents the final output of the DSP module, which is processed signal after going through the selected operations.

The block diagram illustrates the flow of data within the DSP module, showcasing the different processing blocks that transform the input signal to generate the desired output signal.

### 2.4 Flowchart




The flowchart illustrates the sequential steps involved in the DSP module, starting from the input signal, applying the selected operations based on the mode, and generating the output signal. Each step is determined by the mode selection and performed accordingly, resulting in the desired signal processing.

## 3 Results and Reviews

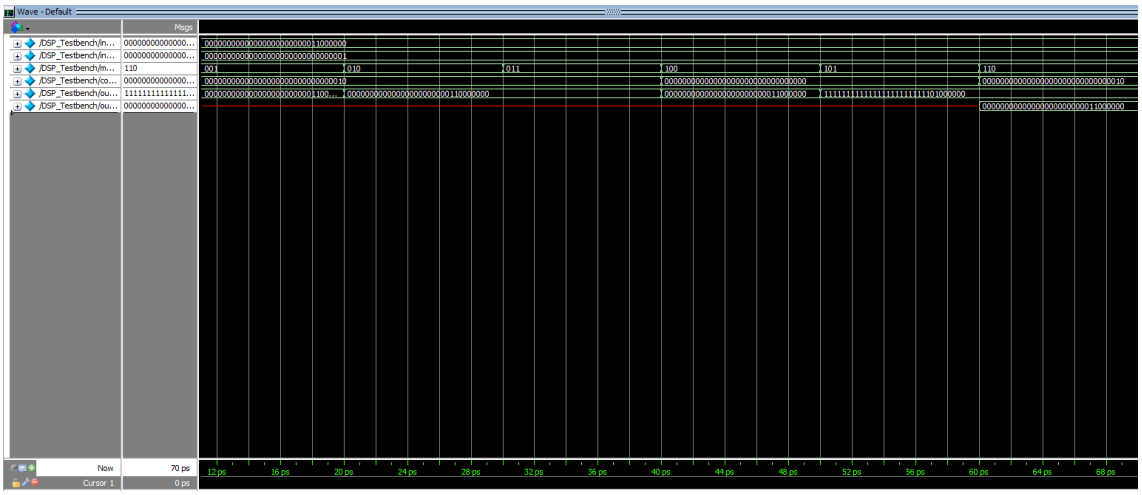
### 3.1 Testbench and Results

If we consider input_signal = 8'b11000000 (decimal 192) and constant = 2, and we want the output to be 32-bit, we need to extend the input_signal and constant to 32 bits. Let's evaluate the result for each operation in the DSP module with the updated input sizes:

1. Time Shifting:
    - Input Signal: 8'b11000000
    - Shifted signal in the time domain: input_signal « constant
    - Result: The output shifted signal will be 32'b1100000000 as the input signal is shifted left by 2 positions.
2. Amplitude Shifting:
    - Input Signal: 8'b11000000
    - Shifted signal in the amplitude domain: input_signal + constant
    - Result: The output shifted signal will be 32'b11000010 as the input signal has each element incremented by 2.
3. Time Scaling:
    - Input Signal: 8'b11000000
    - Scaled signal in the time domain: input_signal * constant
    - Result: The output scaled signal will be 32'b110000000 as the input signal is multiplied by 2.
4. Amplitude Scaling:
    - Input Signal: 8'b11000000
    - Scaled signal in the amplitude domain: input_signal * constant
    - Result: The output scaled signal will be 32'b110000000 as the input signal is multiplied by 2. The result is a signed value represented in two's complement form.
5. Time Reversal:
    - Input Signal: 8'b11000000
    - Reversed signal in the time domain: input_signal[7:0]
    - Result: The output reversed signal will be 32'b11000000 as the input signal is reversed in time.
6. Amplitude Reversal:
    - Input Signal: 8'b11000000
    - Reversed signal in the amplitude domain: -input_signal
    - Result: The output reversed signal will be 32'b11111111111111111111111101000000 as the input signal is negated.
7. Convolution:
    - Input Signal: 8'b11000000
    - Input Signal 2: 8'b00000001
    - Convolution signal: Convolution of input_signal and input_signal[constant - k]
    - Result: The output convolution signal will be 32'b11000000 as inputs signal.

Based on the given input signal and constant, the results for the different operations in the DSP module with a 32-bit output are as described above.

### 3.2 Waveform




Comparing with results above, we can see that results have the same output with the waveform.

## 4 Conclusion

- In this project, we designed a simple DSP (Digital Signal Processing) module using the Verilog hardware description language. The module is capable of processing discrete signal sequences and performs various operations such as shifting, scaling, reversal, and convolution.
- The module takes in an input signal, represented as a discrete sequence, along with a mode selection and a constant value. The mode selection determines the specific operation to be performed on the input signal. The constant value is used for scaling or shifting operations.
- The module outputs the processed signal based on the selected mode. The output signal can be the shifted signal, scaled signal, reversed signal, or convolve signal.
- Throughout the design process, we provided Verilog code for the DSP module and created a testbench to verify its functionality. We also presented a block diagram and flowchart to illustrate the module's operation.
- In conclusion, the DSP module designed in this project provides a basic set of signal processing capabilities for discrete signal sequences. It can be further expanded and optimized to handle more complex operations and larger signal sizes, based on specific application requirements.

# References

[1] https://www.fpga4student.com/

[2] https://www.fpga4fun.com/

[3] https://www.tutorialspoint.com/digital_signal_processing/index.htm

[4] https://www.hackster.io/

[5] https://www.instructables.com/circuits/howto/fpga/