Cairo University
Faculty of Engineering
Department of Electronics and Electrical Communications Engineering

4th Year
Digital Design
Project – Spring 2023

# FFE EQUALIZER

## Team Report:

| Name | Sec | B.N |
|---|---|---|
| أسامة علي محمد رشوان | 1 | 29 |
| عمر ايمن عبد المتعال عبد الحليم | 3 | 13 |
| محمد عادل كامل سيد | 3 | 42 |
| محمد احمد محمد محمد ابوسعدة | 3 | 32 |
| محمد ايمن عبدالعليم احمد | 3 | 34 |
| محمد حسام عثمان يسن | 3 | 35 |
| محمد سيد حسين سيد | 3 | 37 |

# Contents

Table of Figures ..... 2
Abstract ..... 3
Introduction ..... 3
The FFE Algorithm ..... 3
Concept ..... 3
Equation ..... 4
Implementation ..... 4
Block Diagram ..... 4
Synchronizer ..... 5
Shift Register ..... 5
Multiplier ..... 5
Adder ..... 6
Control Unit ..... 6
MATLAB Code simulating FFE operation ..... 6
Pseudo Code ..... 7
Verilog Simulation ..... 8

# Table of Figures

FIGURE 1. THE ABSTRACT BLOCK DIAGRAM ..... 4
FIGURE 2. DETAILED BLOCK DIAGRAM ..... 5
FIGURE 3. VERILOG SIMULATION ..... 8

## Abstract

In this paper we present the Feed-Forward Equalizer FFE. The feed-forward section (FFE) is a linear transposed filter that removes the linear precursor inter-symbol interference (ISI) by a sufficient length to delay the channel response to make it causal.

## Introduction

The FFE is an important technique in the field of channel equalization. The term equalization can be used to describe any circuit or signal processing operation that minimizes ISI. The purpose of an equalizer is to reduce ISI as much as possible to minimize the probability of wrong decisions.The equalization target is to reduce distortion and BER to acceptable levels. The equalizer can be made of discrete components on PCB, integrated using silicon technologies like CMOS or BiCMOS or built into cables or connectors.

The equalizers can be classified as:

1. Linear and non-linear types. Linear equalizers do not have a feedback path, while non-linear equalizers like Decision Feedback Equalizer (DFE) and Maximum Likelihood Sequence Estimator (MLSE) have a feedback path. Linear equalization is easy to implement but amplifies noise power due to the inversion of channel frequency response. On the other hand, non-linear equalization is more complex but has less noise enlargement than linear equalizers. While linear equalization is faster than non-linear equalization, the latter achieves better equalization at the cost of complexity and speed loss.
2. Pre-equalizers and post-equalizers: the equalizer may be placed anywhere in the channel, at the transmitter side (pre-equalizer), receiver side (post-equalizer), or both sides. Pre-equalization adjusts the magnitude of the transmitter output by adding certain peaking based on the channel response. This reduces the maximum swing, but produces an open eye.

## The FFE Algorithm

### Concept

To implement an RX FFE in the analog domain, a series of delay lines must be utilized to buffer and combine or subtract the incoming signal based on the FFE coefficients. Alternatively, FFE can be implemented at the bit or symbol level by convolving the FFE coefficient with the sampled input data stream for equalization. Typically, an adaptation scheme is paired with RX FFE to derive FFE coefficients from channel characteristics.

## Equation

The FFE filter taps values (hi) are given as follows: $h0 = 0.5$, $h1 = -0.25$, $h2 = 0.15625$, $h3 = -0.0625$. The input data ($Di$) is signed and 12-bit wide, and its range is given as follows: $Di \in (-2^{11}, 2^{11})$. The output data ($Yi$) has the following relation with input data and the FIR filter taps: $\boldsymbol{Yi} = \sum \boldsymbol{hjDi}$.

## Implementation

Assuming an input data frequency of 1MHZ and an FFE frequency of 4MHZ, the implementation employs a resource sharing algorithm due to the design frequency being four times greater than the minimum. To optimize area and resources, a single multiplier and adder were utilized, along with memory and storage optimizations.

Below we put the Full Design Block Diagram, then we explain each distinctive block in details.

## Block Diagram

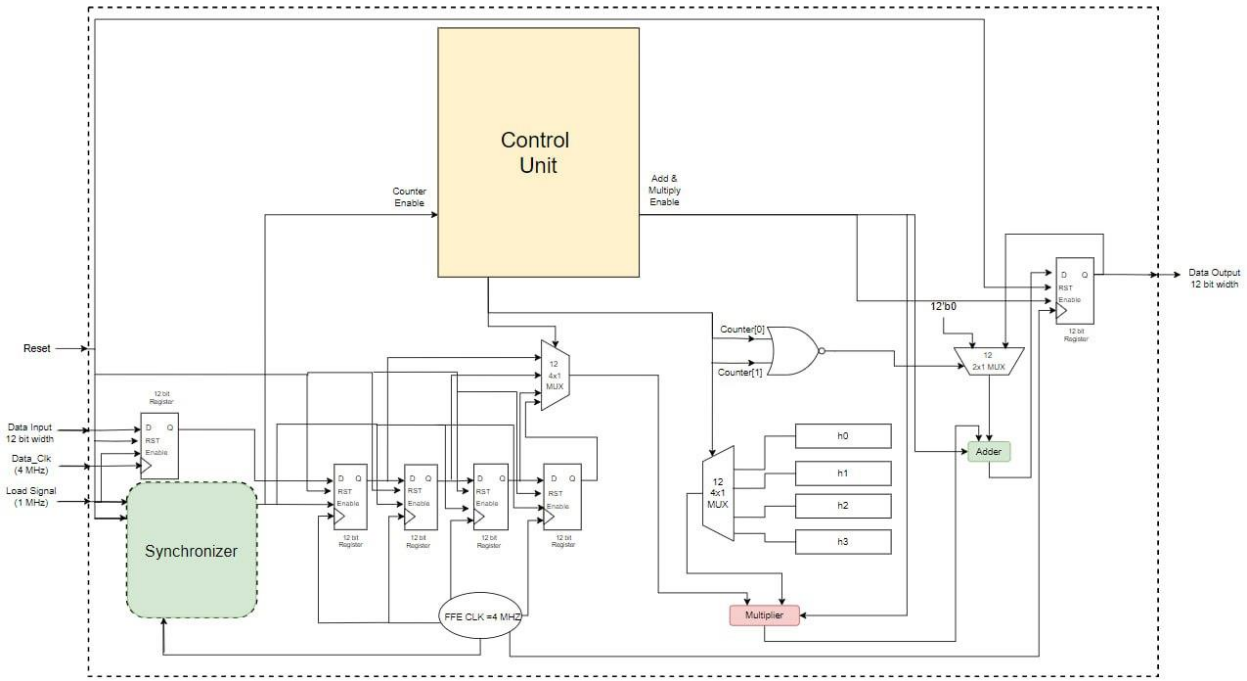


*Figure 1. The Abstract Block Diagram*

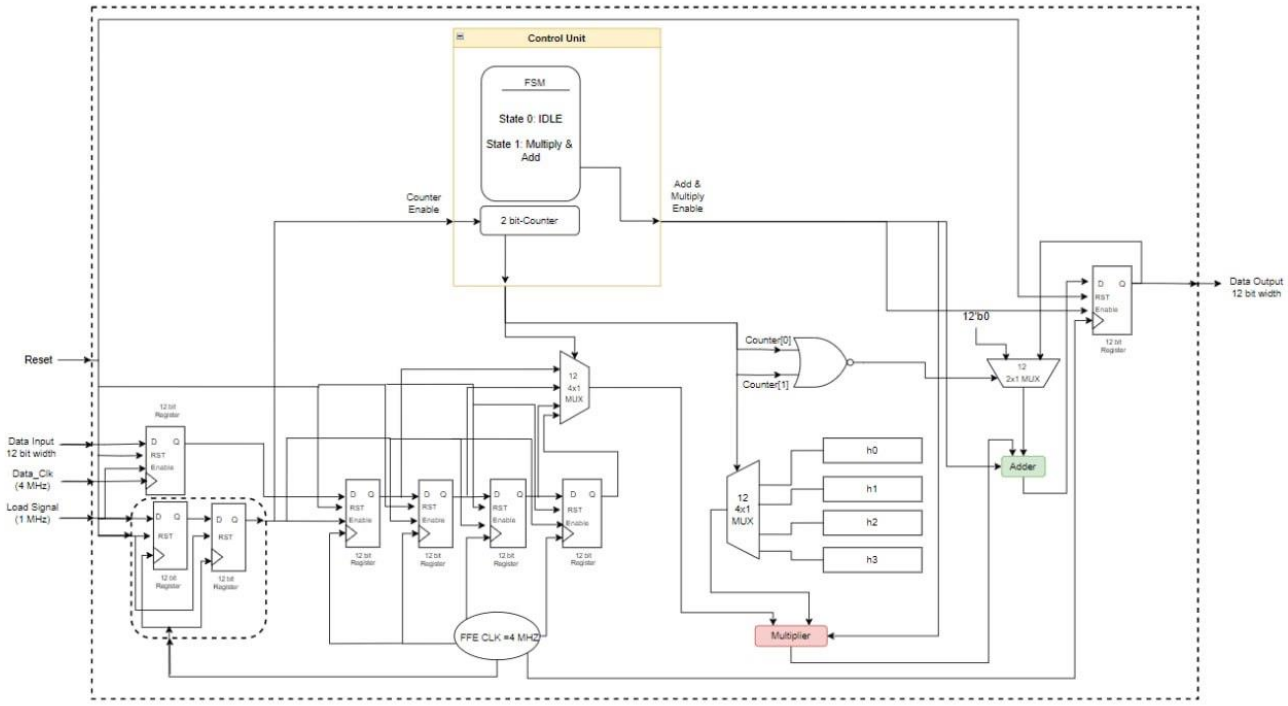


*Figure 2. Detailed Block Diagram*

## Synchronizer

In this design we have two different clock domains. The input data clock frequency is 1MHZ while the FFE clock frequency is 4MHZ and they both come from two different sources. For this reason, we put a synchronizer for the load signal from the input clock domain before entering the FFE to prevent any timing violations. The synchronizer is simply two flip flops connected back-to-back and their clock is that of the destination clock domain.

## Shift Register.

We have a multiple-bits shift register. The word length is 12-bits and we have 4 registers each is 12-bits. Every time a new data enters we do a serial shift to the right, the oldest data sample is thrown out and the new data sample is stored in the leftmost register. We then use a multiplexer to choose between the 4 words stored currently in the shift register to be multiplied by the equalizer coefficients.

## Multiplier

We use a 12-bit multiplier that will multiply at each clock cycle of the FFE one of the data words by one of the FFE tabs. After doing Fixed Point Analysis, we chose to divide those 12-bit operands to 6-bits for the integer part and 6-bits for fractions. The output would be 24-bits, and we will take only 12-bits from bit (6) to bit (17) to represent the output, 6-bits of them from bit(6) to bit(11) represent the fractional part, and the rest from bit(12) to bit(17) represent the integer part of the output.

## Adder

We have a 12-bit adder, the output each time would be 13-bits but we truncate the Least Significant Bit. After receiving the load signal, the adder starts by adding the multiplier output to 0 in the first clock cycle and in the next cycles it begins accumulating the multiplier outputs to each other.

## Control Unit

The control unit is composed of a Finite State Machine (FSM) and a two-bit counter, with the FSM having two states: IDLE and Multiply & Add. The IDLE state is entered upon reset, while the Multiply & Add state controls the primary function of the FFE by issuing control signals such as the selection of multiplexers, which correspond to the count of the 2-bit counter. It also produces the control signal "Add & Multiply Enable" which is the enable signal of the adder and multiplier blocks.

After passing through the synchronizer, the 2-bit counter begins counting upon receiving the Load signal. The counter counts from 0 to 3 and each count represents the index of data word from the shift register that would be multiplied by the FFE tabs at this clock cycle. The counter value is updated at each clock edge of the FFE and is reset after receiving the load signal. The counter 2-bits are input to a NOR gate to produce the selection of the multiplexer that chooses the Adder operands.

## MATLAB Code simulating FFE operation

```
%%%%%%%%%%

h=[0.5 -0.25 0.15625 -0.0625];

x=rand(40,1)*4-1

y=[];

 y = zeros(size(x));

%  Apply FIR filter

 for n = 1:numel(x)

    for k = 1:numel(h)

       if (n-k+1) > 0

          y(n) = y(n) + h(k) * x(n-k+1);

       end

    end

 end
```

## Pseudo Code

```
if (load_Signal)
{
 synchronizer_input=load_Signal; //Load signal first enters the synchronizer since it comes from a different clock domain
 synchronizer_output=load_Signal_synch; //Synchronized load signal
 for (j=1;j<=3;j++) {
   shift_reg[j]=shift_reg[j-1]; }
 shift_reg[0]=input_data;
 counter_enable=1;
 //shift_reg={input_data,shift_reg[1:3]};  }
if(counter_enable)
{
 shift_reg_mux_select=count;    // The mux would select the data word with index equal to the counter count.
 FFE_tabs_mux_select=count;   // The mux would select the FFE tab with index equal to the counter count.
 adder_mux_select=(count[0] nor count[1]);  // One operand of the adder is always the current multiplier output and this
mux   selects the other operand which might be 0 for the first time operation or it might be the previous adder result
 FSM_Current_State=Add_and_Multiply_state; // This state gives the control signals that enable the multiplier and addition
to   work.
 Add_and_Multiply_enable=1;  // The enable signals of both the multiplier and adder
}
if(Add_and_Multiply_enable)
{
  for(i=0;i<=3;i++) {
    multiplier_output=shift_reg[i] * h[i]; }
    if(i==0) {
      adder_output=multiplier_output + 0;
      adder_accumulate=adder_output;   }
    else {
      adder_output=multiplier_output+adder_accumulate;
      adder_accumulate=adder_output;
           } }
```

## Verilog Simulation

We wrote a Verilog code that executes the FFE function and the following waveform from simulations that shows a sample of an Input and the FFE output.

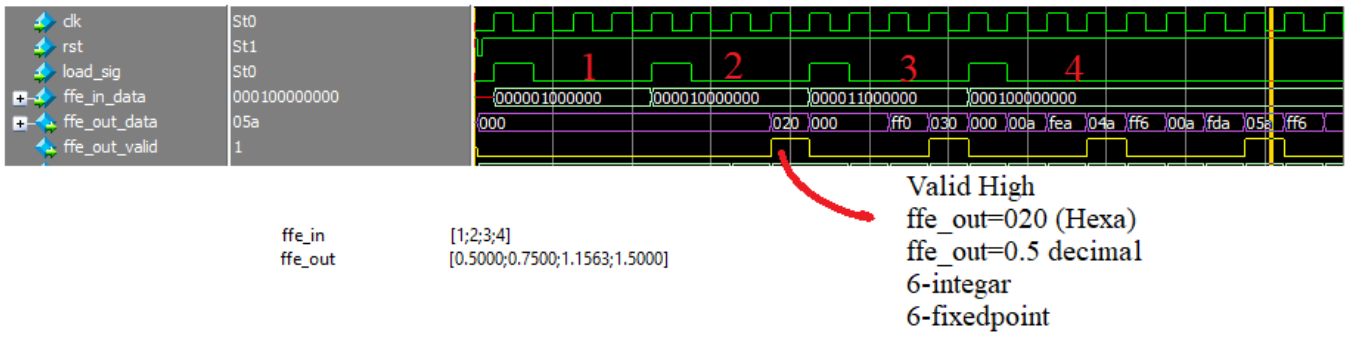


*Figure 3. Verilog Simulation*