Politecnico di Torino

Politecnico di Torino

Laboratory 2: Floating Point Multiplier
Integrated Systems Architecture

Master degree in Embedded Systems

Muccioli Geremia 287590
Goachem Farah 282871

December 19, 2021

Contents

CHAPTER 1

Floating Point Multiplier Simulations

1.1 Verifying model correctness

1.1.1 Single Cycle Floating Point Multiplier

The first operation that has been performed was the simulation of the multiplier without the pipeline. This case has been taken into consideration just to verify the correctness of its results, because, due to its unoptimized architecture it's not particularly interesting, and therefore it has not been synthetized. Anyway in Figure 1.1 are shown the results of the simulation.

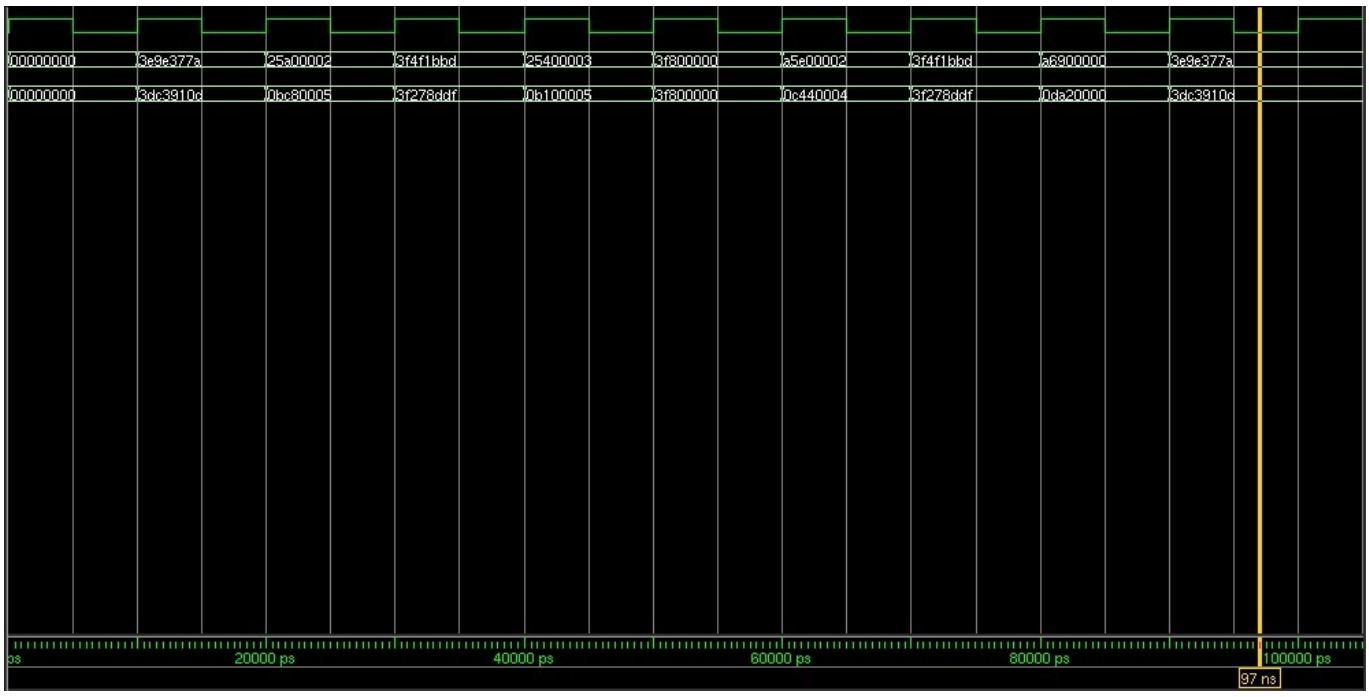

Figure 1.1: Single cycle floating point multiplier simulation

1.1.2 Pipelined Floating Point Multiplier

The next simulation,instead, has been performed on the pipelined version of the multiplier, which still behaves correctly, adding the latency of the pipeline on the output. The simulation is shown below in Figure 1.2

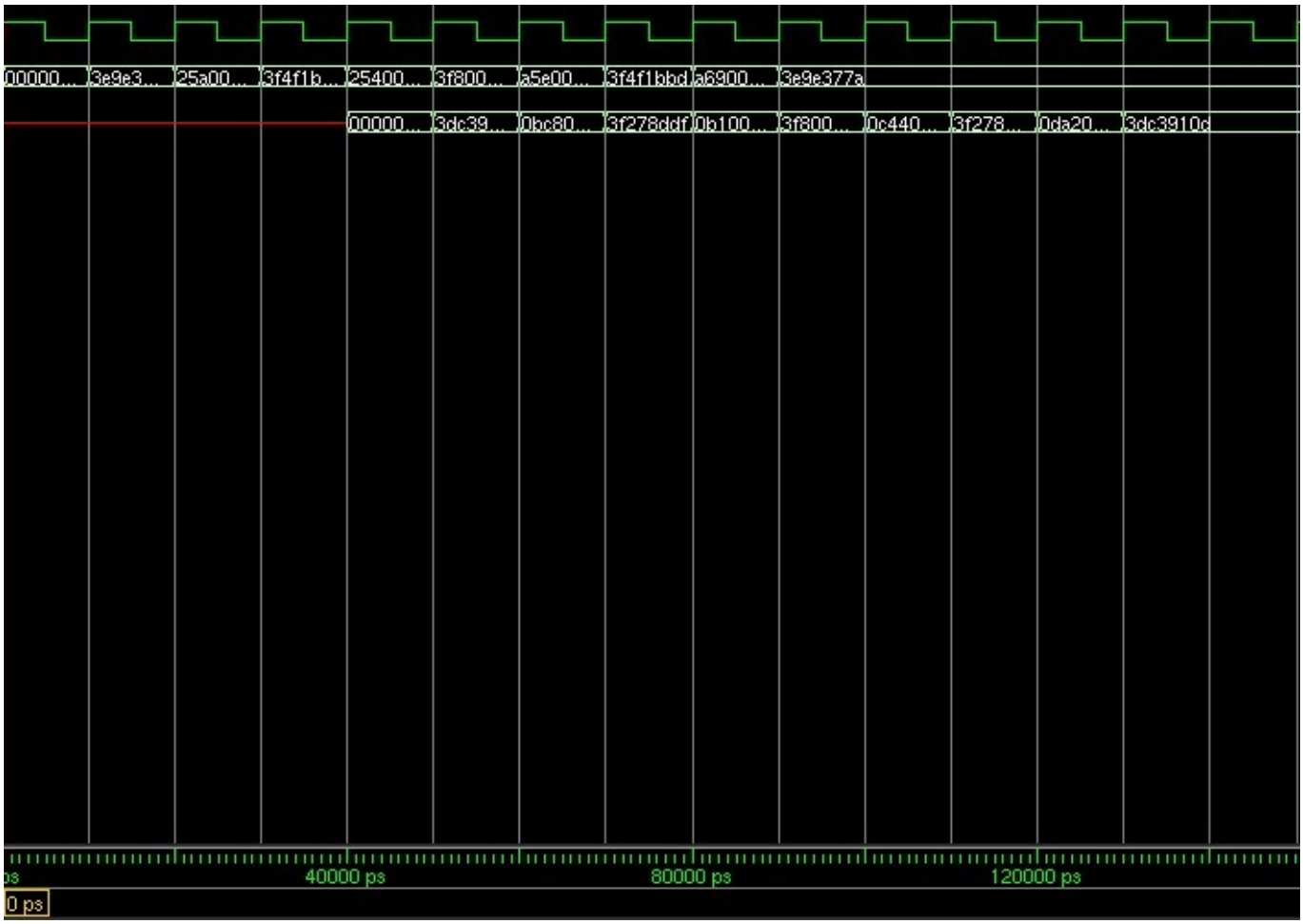

Figure 1.2: Pipelined floating point multiplier simulation

This simulation is useful to check the validity of the output results, but a part from that, also this case is not much interesting, because it still has margins of improvement, starting from the synchronization of the data at the input, which is indeed described in the following section.

1.1.3 Multiplier with input registers

Once we have verified that the model was behaving correctly, we have inserted input registers in order to synchronize data arriving from the external environment. Finally, we simulated again the multiplier in order to check if the correctness of the component was preserved. As expected, observing the results, it can be seen that the values are still the correct ones, and the latency incremented by 1 clock cycle w.r.t the previous case. The simulation is shown in Figure 1.3.

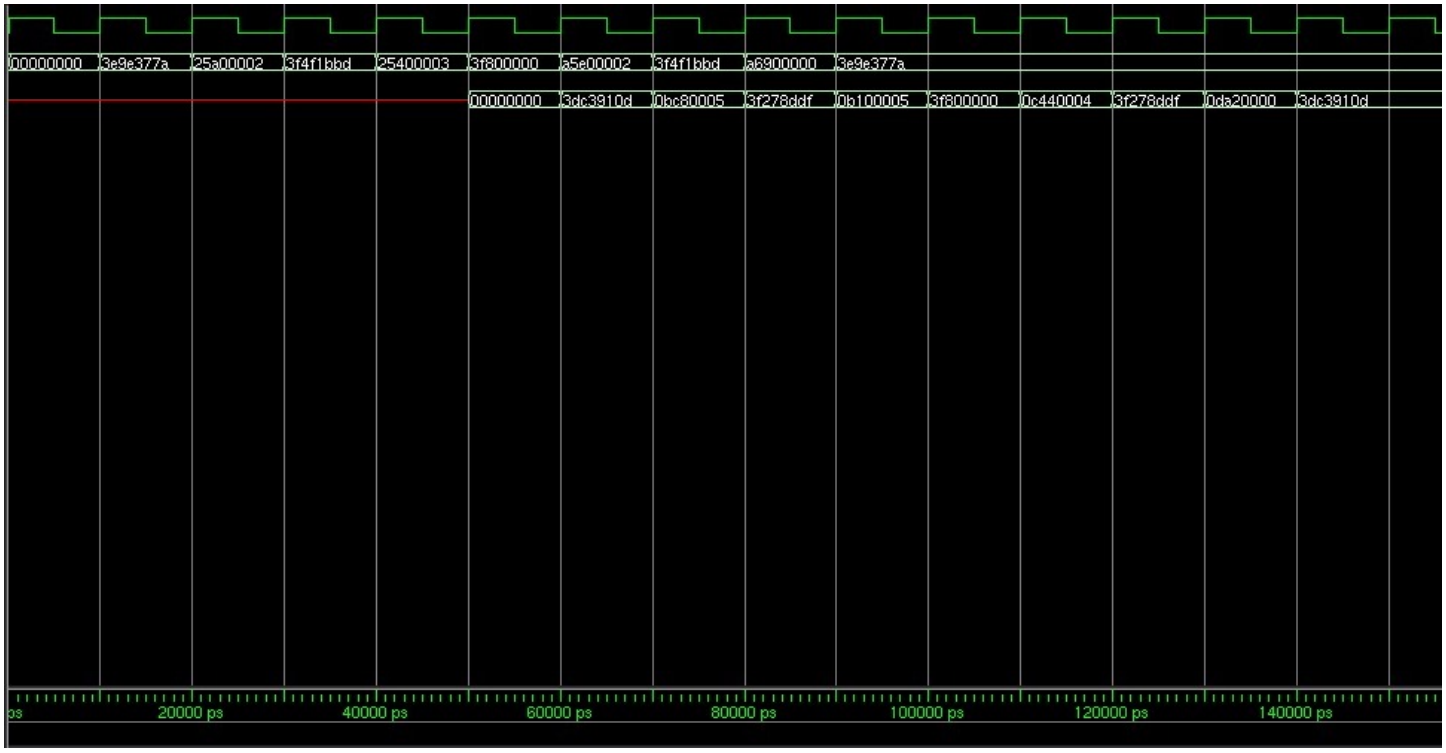

Figure 1.3: Pipelined floating point multiplier simulation with input registers

CHAPTER 2

Floating Point Multiplier Synthesis

2.1 Synthesis

After the simulation step, we went straight on the synthesis of the multiplier, in order to explore different solutions for the choice of its internal components, in particular we focused on the implementation of the unsigned multiplier in stage 2, which performs the calculations on the significands.

2.1.1 Unconstrained Synthesis

The first synthesis has been executed leaving to the synthesis designer the choice of the various internal component of the multiplier, obtaining a maximum period for the clock equal to $T_{clk} = 1.40$ns and its related area equal to Area $= 4122.99\ um^2$. Once the multiplier has been synthetized, we issued the report resource command in order to verify which components had been inserted, and, as can be seen in Figure 2.1, the Synopsis Compiler used a PPARCH for the unsigned multiplier in stage 2.

```
****************************************
Report : resources
Design : FPmul_pipeline
Version: O-2018.06-SP4
Date   : Sat Dec 18 16:30:24 2021
****************************************

Resource Sharing Report for design FPmul_pipeline in file
        ../src/fpmul_stage1_struct.vhd

===============================================================================
|          |               |            | Contained     |                      |
| Resource | Module        | Parameters | Resources     | Contained Operations |
===============================================================================
| r343     | DW01_add      | width=8    |               | add_1_root_I2/add_126_2 |
| r347     | DW01_add      | width=8    |               | I3/I9/add_41_aco     |
| r349     | DW01_inc      | width=25   |               | I3/I11/add_45        |
| r351     | DW01_add      | width=8    |               | I4/I1/add_41_aco     |
| r931     | DW_mult_uns   | a_width=32 |               | I2/mult_134          |
|          |               | b_width=32 |               |                      |
===============================================================================


Implementation Report
===============================================================================
|                    |                   | Current            | Set            |
| Cell               | Module            | Implementation     | Implementation |
===============================================================================
| I2/mult_134        | DW_mult_uns       | pparch (area,speed)                 |
|                    |                   | mult_arch: benc_radix4              |
| I3/I11/add_45      | DW01_inc          | pparch (area,speed)                 |
| add_1_root_I2/add_126_2                |                    |                |
|                    | DW01_add          | rpl                |                |
===============================================================================

1
```

Figure 2.1: Report resources for the unconstrained synthesis

2.1.2 Synthesis forcing CSA in stage 2

This synthesis, instead, has been perfomed by forcing the design compiler to use a CSA multiplier in the second stage of the floating point unit. We expected to have a clock period longer than the unconstrained synthesis because of how the compiler implemented the second stage, and indeed the clock period in this case is equal to $T_{clk} = 3.89$ns, and the related area equal to Area $= 4906.64um^2$. Coherently to what expected, the maximum achievable frequency is less than half w.r.t the case using the PPARCH multiplier.

2.1.3 Synthesis forcing PPARCH in stage 2

Since the unconstrained synthesis implemented a PPARCH multiplier in the stage 2, the results of this synthesis are very close the the other case. The minimum clock period is indeed equal to $T_{clk} = 1.41$ns, and its related area equal to Area $= 4145.08\ um^2$. We'd also like to underline the fact that, both in this case and in the unconstrained one, the critical path is found by the compiler between the first input of the second stage (A_SIG) and the significand output. In Figure 2.2 we can look at the resources used in this case, and in fact is very similar to Figure 2.1.

```
****************************************
Report : resources
Design : FPmul_pipeline
Version: O-2018.06-SP4
Date   : Sat Dec 18 16:33:32 2021
****************************************

Resource Sharing Report for design FPmul_pipeline in file
        ../src/fpmul_stage1_struct.vhd

===============================================================================
|          |              |            | Contained     |                      |
| Resource | Module       | Parameters | Resources     | Contained Operations |
===============================================================================
| r120     | DW01_add     | width=8    |               | add_1_root_I2/add_126_2 |
| r122     | DW02_mult    | A_width=32 |               | I2/mult_134          |
|          |              | B_width=32 |               |                      |
| r124     | DW01_add     | width=8    |               | I3/I9/add_41_aco     |
| r126     | DW01_inc     | width=25   |               | I3/I11/add_45        |
| r128     | DW01_add     | width=8    |               | I4/I1/add_41_aco     |
===============================================================================


Implementation Report
===============================================================================
|                    |                    | Current           | Set           |
| Cell               | Module             | Implementation    | Implementation |
===============================================================================
| I2/mult_134        | DW02_mult          | pparch (area,speed) | pparch        |
|                    |                    | mult_arch: benc_radix4               |
| I3/I11/add_45      | DW01_inc           | pparch (area,speed)                  |
| add_1_root_I2/add_126_2                 |                   |                |
|                    | DW01_add           | rpl               |                |
===============================================================================

1
```

Figure 2.2: Report resources for the PPARCH constrain synthesis

2.2 Results Table

In green are underlined the best cases, while in red the worsts.

	Unconstrained	CSA	PPARCH
Max Frequency(MHz)	714,29	257,07	709,22
Area(µm^2)	4122,99	4906,64	4145,08

Figure 2.3: Results from the three different synthesis

CHAPTER 3

Fine-grain Pipelining and MBE

3.1 Optimize_registers case

3.1.1 Simulation

In this case, we added a register at the output of the multiplier in stage 2 which is in charge to evaluate the significand, and to preserve the correctness of the data flow, some other registers have been added to input signals. The simulation reported in Figure 3.1 shows that the results are correct and the latency is again increased by one clock cycle w.r.t the previous cases (Figure 1.3).

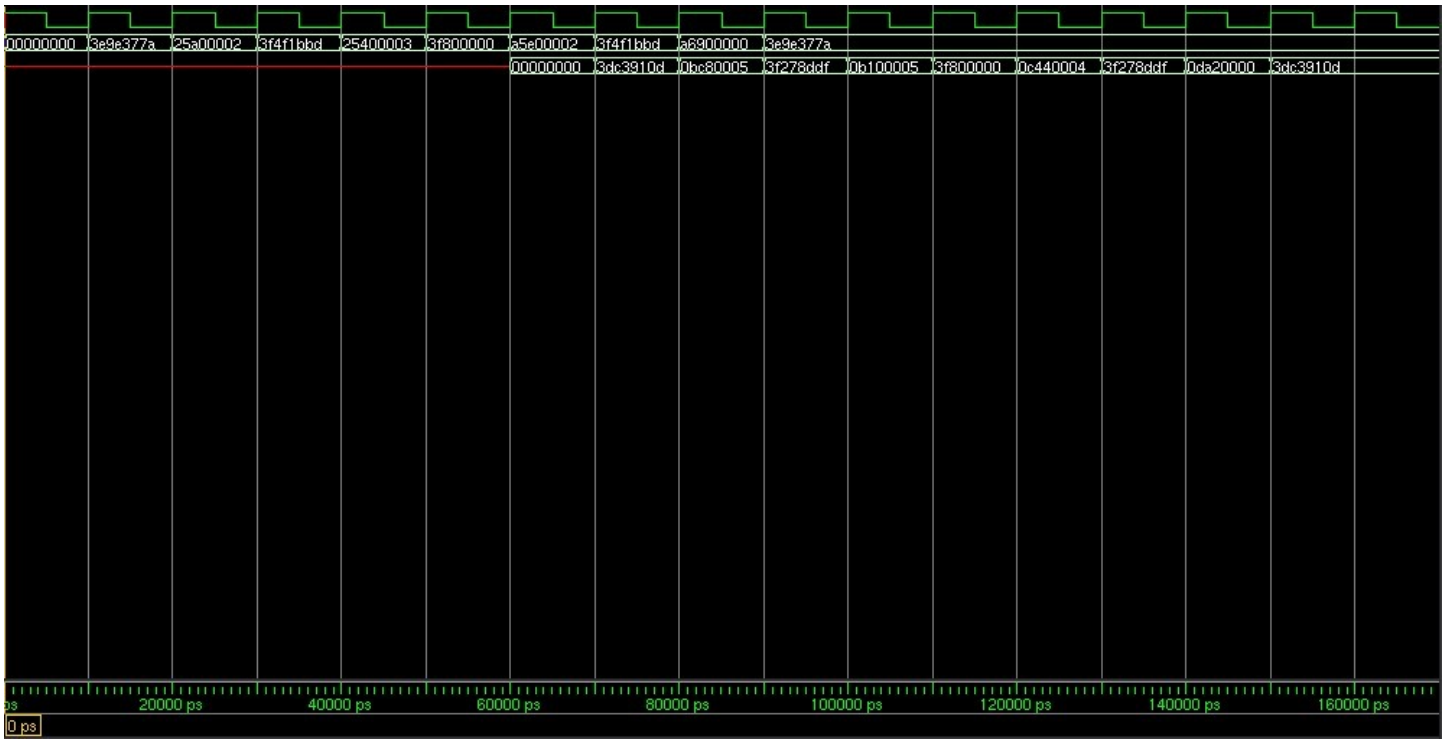

Figure 3.1: *optimize_registers simulation results*

3.1.2 Synthesis

For the synthesis we raised the command *optimize_registers* after the *compile* command. Doing this way we got the best frequency performance, w.r.t all the other cases. The clock period found is indeed equal to $T_{clk} = 0.94$ns, achieving a maximum frequency greater than 1GHz. The drawback is that the related area is Area = 5341,01 um^2, the biggest among all the architectures. Also here, the design compiler choose the PPARCH multiplier for the second stage, as can be seen in Figure 3.2.

```
****************************************
Report : resources
Design : FPmul_pipeline
Version: O-2018.06-SP4
Date   : Sat Dec 18 17:17:23 2021
****************************************

Resource Sharing Report for design FPmul_pipeline in file
        ../src/fpmul_stage1_struct.vhd

===============================================================================
|          |              |            | Contained     |                      |
| Resource | Module       | Parameters | Resources     | Contained Operations |
===============================================================================
| r343     | DW01_add     | width=8    |               | add_1_root_I2/add_154_2 |
| r347     | DW01_add     | width=8    |               | I3/I9/add_41_aco     |
| r349     | DW01_inc     | width=25   |               | I3/I11/add_45        |
| r351     | DW01_add     | width=8    |               | I4/I1/add_41_aco     |
| r931     | DW_mult_uns  | a_width=32 |               | I2/mult_162          |
|          |              | b_width=32 |               |                      |
===============================================================================


Implementation Report
===============================================================================
|                    |                  | Current            | Set            |
| Cell               | Module           | Implementation     | Implementation |
===============================================================================
| I2/mult_162        | DW_mult_uns      | pparch (area,speed)                 |
|                    |                  | mult_arch: benc_radix4              |
| I3/I11/add_45      | DW01_inc         | pparch (area,speed)                 |
===============================================================================

1
```

Figure 3.2: Report resources for *optimize_registers* case

3.2 Compile_ultra case

3.2.1 Synthesis

This synthesis has been performed issuing the command *compile_ultra* instead of *compile+optimize_registers* commands. We were expecting the results to be very similar the the previous case, but instead, they're almost equal to the PPARCH case, and in fact the lowest achievable period for the clock is $T_{clk} = 1.48$ns and the area is Area = 4417,99 um^2.

3.3 MBE

This paragraph is about the MBE for unsigned data that has been designed in order to be inserted in the second stage of the floating point multiplier for the significand evaluation.

3.3.1 MBE description

The MBE is designed based on a Booth Encoder multiplier, i.e. we have a first step in which the booth encoder generates some partial products (17 in out case) to be added together, and this adding operation relies on the Dadda Tree network architecture. Therefore, the sources files of the MBE could be divided in three parts: *MBE.vhd* which is the higher entity, *Booth_encoder.vhd* which takes care of generating the partial products starting from the input data, and *Dadda_tree.vhd* which is implementing the Dadda network (and so it instantiates all the required FAs and HAs). The partial products are sign extended according to the optimization found in the file *sign_extension_booth_multiplier_Stanford.pdf*. Another implementation choice to be mentioned is that the Dadda Tree network layers have been implemented separately, leaving the instantiation of the adders of each column to the *adder_gen.vhd* module, that receives the number of HA and FA, the length of inputs and outputs vectors and the length of the carry out. Therefore, all the *Dadda_lx.vhd* source files are different from each other only in the mapping of the *adder_gen* modules.

3.3.2 MBE simulation

The simulation of the floating point multiplier that uses the MBE in stage 2, as expected, is the same as the previous case, because, in fact, the overall architecture does not change, just its internal components. The Figure 3.3 shows the result of the simulation, and indeed it's the same as Figure 3.1.

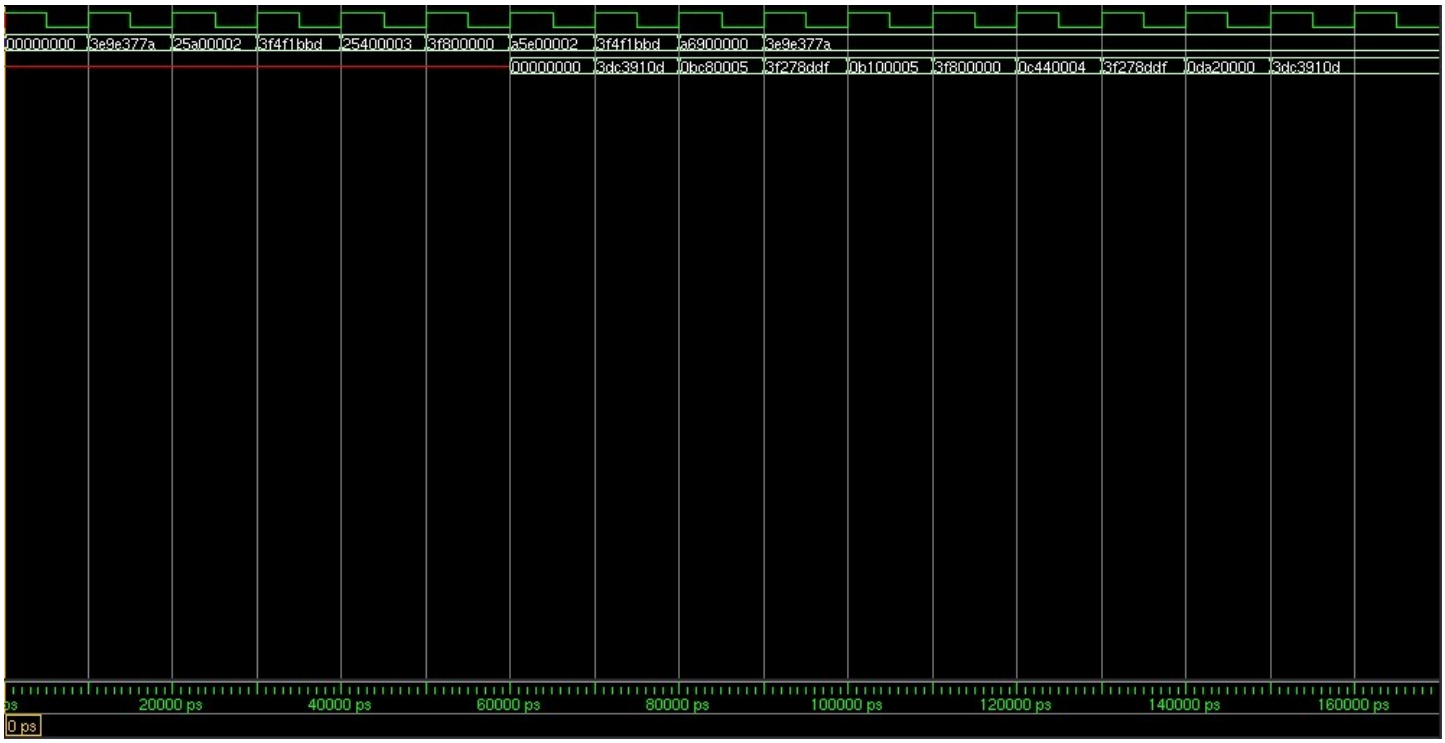

Figure 3.3: *MBE* simulation results

3.3.3 MBE synthesis

Regarding the synthesis of the FP multiplier using the MBE in stage 2, we can observe that is very similar to the PPARCH case, with a slightly bigger area. The values are T_{clk} = 1.39ns and Area = 4327,82 um^2. For this synthesis the report of the resources is shown in Figure 3.2

```
****************************************
Report : resources
Design : FPmul_pipeline
Version: O-2018.06-SP4
Date   : Sat Dec 18 17:47:26 2021
****************************************

Resource Sharing Report for design FPmul_pipeline in file
        ../src/fpmul_stage1_struct.vhd

===============================================================================
|          |              |            | Contained     |                       |
| Resource | Module       | Parameters | Resources     | Contained Operations  |
===============================================================================
| r343     | DW01_add     | width=8    |               | add_1_root_I2/add_154_2 |
| r347     | DW01_add     | width=8    |               | I3/I9/add_41_aco      |
| r349     | DW01_inc     | width=25   |               | I3/I11/add_45         |
| r351     | DW01_add     | width=8    |               | I4/I1/add_41_aco      |
| r931     | DW_mult_uns  | a_width=32 |               | I2/mult_162           |
|          |              | b_width=32 |               |                       |
===============================================================================


Implementation Report
===============================================================================
|                    |                  | Current            | Set            |
| Cell               | Module           | Implementation     | Implementation |
===============================================================================
| I2/mult_162        | DW_mult_uns      | pparch (area,speed)                 |
|                    |                  | mult_arch: benc_radix4              |
| I3/I11/add_45      | DW01_inc         | pparch (area,speed)                 |
| add_1_root_I2/add_154_2               |                    |                |
|                    | DW01_add         | rpl                |                |
===============================================================================

1
```

Figure 3.4: Report resources for *MBE* case

CHAPTER 4

Comparison Table

This is the table with all the results obtained by the synthesis of the different cases, in green are underlined the best cases, while in red the worsts.

	Unconstrained	CSA	PPARCH	Optimized Registers	Compile Utra	MBE
Max Frequency(MHz)	714,29	257,07	709,22	1063,83	675,68	719,42
Area(µm^2)	4122,99	4906,64	4145,08	5341,01	4417,99	4327,82
Latency(clock cycles)	6	6	6	7	7	7

Figure 4.1: Comparison among results