Politecnico di Torino

1859

Corso di Laurea Magistrale in
INGEGNERIA ELETTRONICA

INTEGRATED SYSTEMS ARCHITECTURE

Lab Report 1

Professors:
Prof. Guido Masera
Prof. Maurizio Martina

Students:
Christian Vespo 292674
Claudio Raccomandato 290190
Pietro Romeo 269010

Anno Accademico 2021/2022

Contents

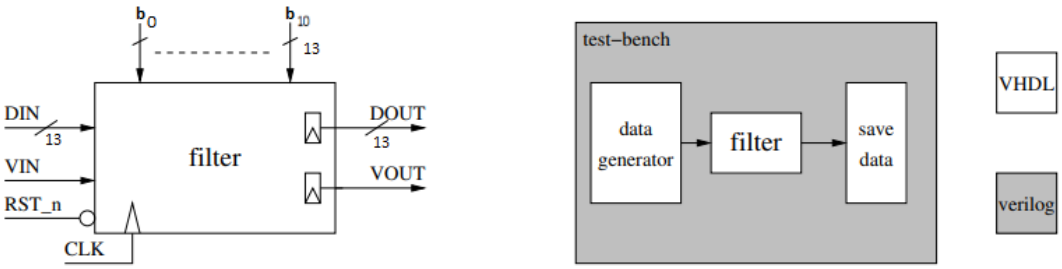
Developement of a Digital Filter

The goal of this Lab is to design a Digital Filter with a cut-off frequency of 2 kHz and a sampling frequency of 10 kHz. The designed filter is a Finite Impulse Response (FIR) filter, chosen according to the parameter p, that is equal to 1, because the group number is odd. To compute the order of the filter (N) and the number of bits n_b, the following formulas were used

$$N = 2^p \cdot [(x mod 2) + 1] + 6 \cdot p$$

$$n_b = (y mod 7) + 8$$

with x = 5 (number of characters in the first surname), y = 5 (number of characters in the second surname), getting N = 10 and n_b = 13 .



(a)

(b)

Figure 1: High level block diagram of FIR filter (direct form)

1 Testing the filter and fixed point implementation

1.1 Matlab pseudo-fixed-point

By repurposing the provided Matlab code, a fixed-point representation of our filter was obtained.

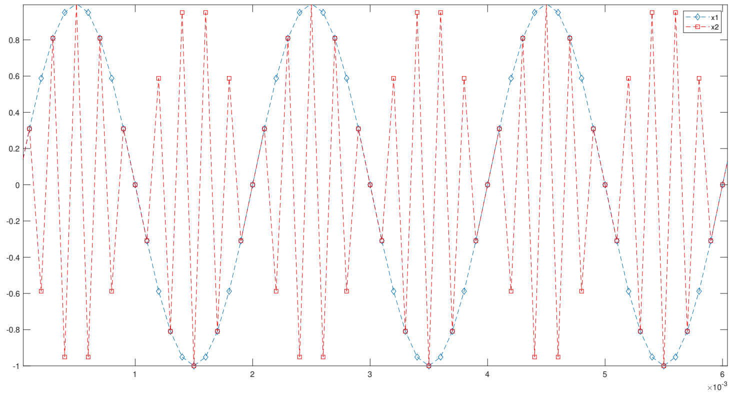


Figure 2: Inputs of the Matlab filter

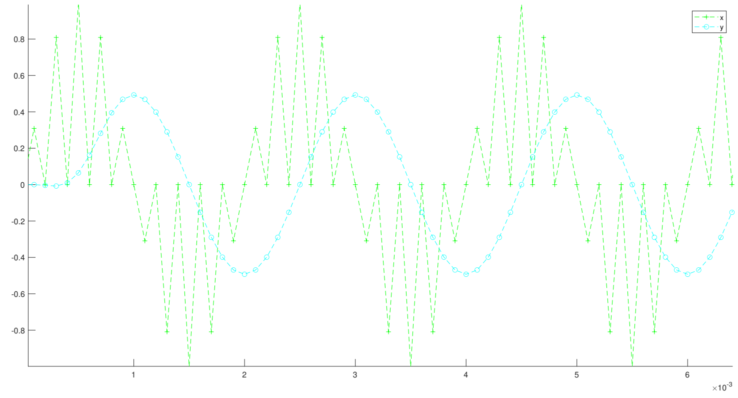


Figure 3: Matlab output vs sum of the two inputs

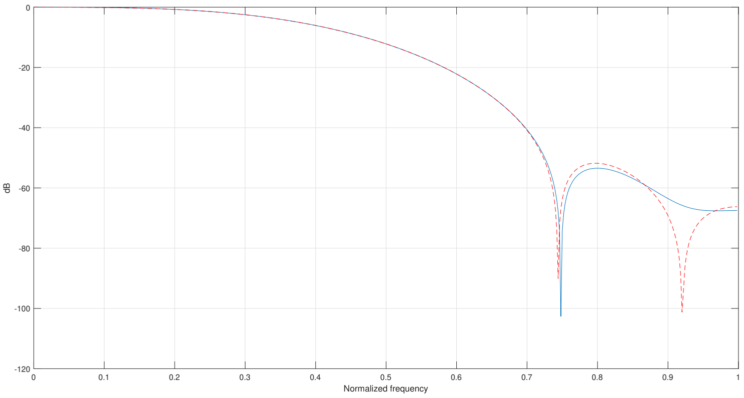


Figure 4: Matlab filter response

Figure 2 represents the filter output for the given inputs: a sinusoidal wave in band and a sinusoidal wave out of band. Figure 3 shows the output signal; comparing it with the sum of the input signals previously shown, we can observe that at the beginning of the time axis, the output is not yet a sinusoidal wave. The reason for this behavior is due to the fact that at the beginning of the filtering process the filter does not have enough samples to calculate the right output. Figure 4 represents the filter response.

1.2 Fixed point C model

A fixed point implementation of the filtering operation, represented with the following expression:

$$y_i = \sum_j x_{i-j} \cdot b_j$$

has been made in C and is shown in Code 1.

```
y = 0;
for (i=0; i<NT; i++) {
    y += ((sx[i] >> NB_APPROXIMATION)*(b[i] >> NB_APPROXIMATION));
}
y = y << (2*NB_APPROXIMATION-NB+1);
```

Code 1: C algorithm

In the code, the computation of y_i is made with x_{i-j} and b_j on 6 bits, because they have been truncated by a number of bits (indicated in the code with NB_APPROXIMATION) equal to 7.
Working with operands that have a lower number of bits, it is possible to use multipliers and adders with a lower parallelism, and therefore the area of the filter is reduced. At the end of the computation, each y_i is reported on 13 bits. This particular implementation in C is valid for any value of NB_APPROXIMATION bigger than six.
Plotting the values of y obtained in C, stored in ”RESULTS.txt”, together with the values of y obtained in Matlab, stored in ”resultsm.txt”, become clear that this implementation cannot overflow, as shown in Figure 5.

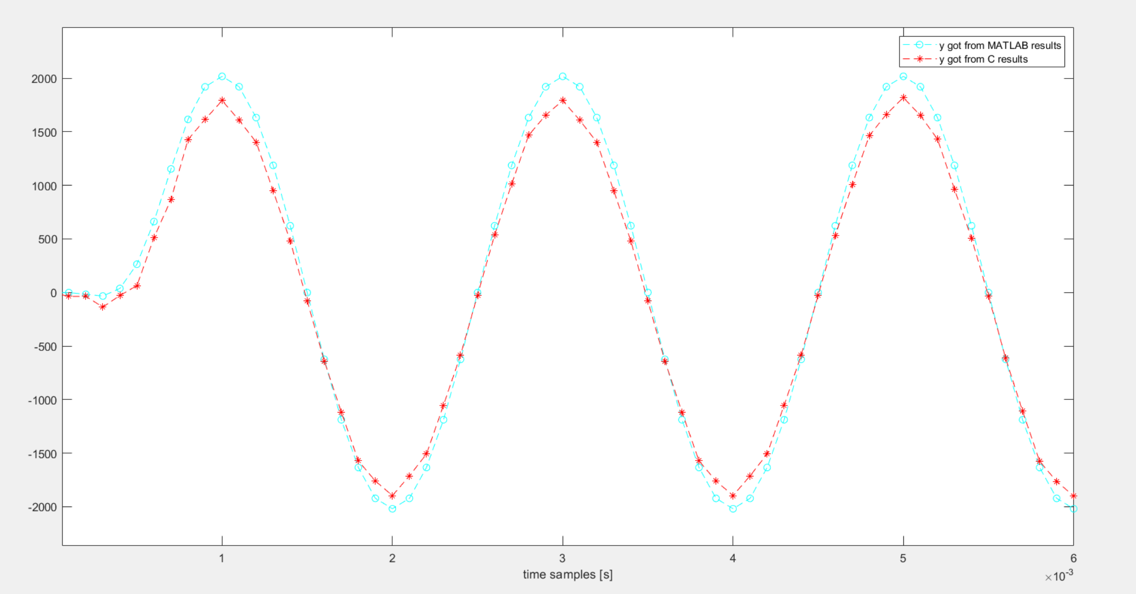


Figure 5: Comparison between Matlab Output and C output after approximation

Finally, the Total Harmonic Distortion (THD) of the output y of the fixed point implementation has been evaluated: using the values from the Matlab output file (resultsm.txt), the THD obtained is equal to -67.37 dBc (-67.37dB considering a value of the carrier of 0dB) as shown in Figure 6.

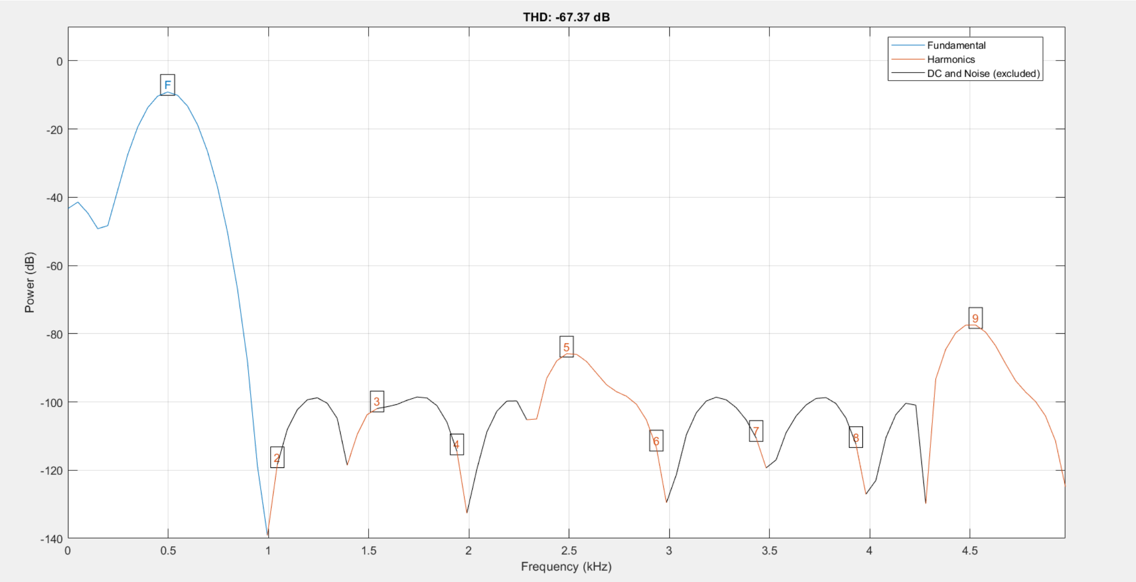


Figure 6: THD computed with Matlab results

On the other hand, using the values of the C output file (RESULTS.txt), the THD obtained is equal to -32.99 dBc, as shown in Figure 7, which is close enough to the maximum value of -30dB. Both THD evaluations have been done considering harmonics up to the ninth, which is also the upper limit imposed by the Matlab function used to generate the THDs.

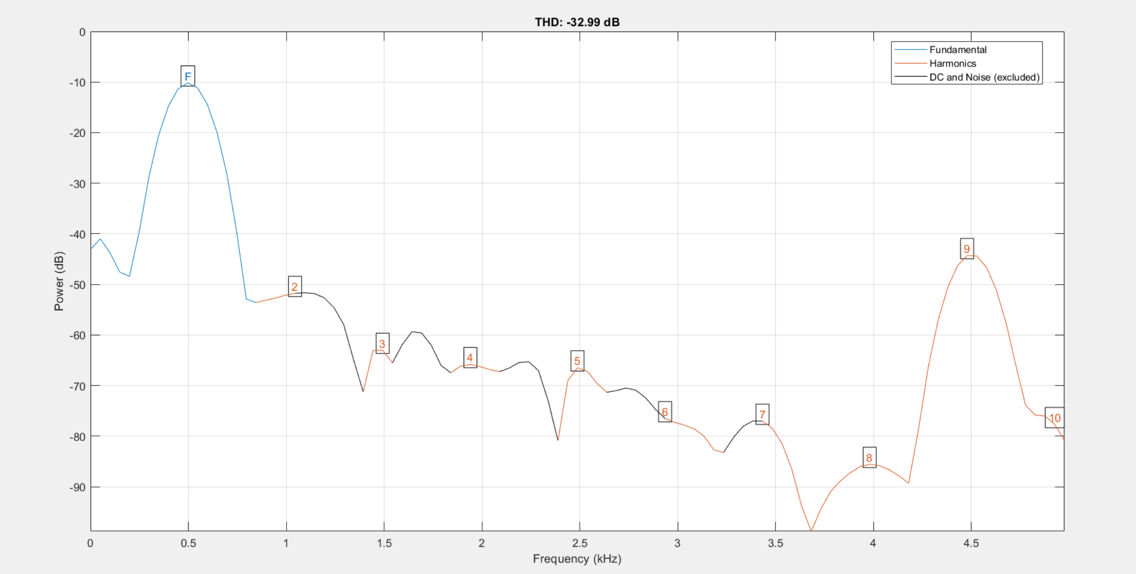


Figure 7: THD computed with C results

2 VLSI implementation

2.1 Starting Architecture Development

After working on the C and Matlab models, it was time to develop a behavioral VHDL model. In order to make it as similar as possible to the C model, it was implemented with a loop as shown in the extracted piece of code below:

```
y_var := 0;
fir_algorithm :
        for k in 0 to 10 loop
            y_var := y_var +
                (to_integer(signed(x_out(k)))*to_integer(signed(bn_r_out(bn_index(k)))));
        end loop fir_algorithm;
y <= std_logic_vector(to_signed(y_var, y'length));
```

Code 2: VHDL behavioral algorithm

A variable was used to make all the calculations, finally the result was converted back to std_logic_vector. All the input data was stored in a 6-bit buffer to get access to the last eleven data received. To store the eleven b coefficients, only six 6-bit register were used because values were symmetrical. In particular these registers sample the coefficients only once on reset front, because we assume that the values of the b coefficients does not need to change until the next reset. Another possibility could be to connect the coefficient pins directly to the internal components of the circuit, but this solution was discarded because noises on the external pins could lead to incorrect output results.

2.2 Simulation

The output text produced by the C model exactly matches the output of the VHDL simulation. Using Modelsim it is also possible to visualize the timing diagram generated by the testbench. From the image below it is possible to see that only 2 clock cycles pass between the rising edge of VIN and that of VOUT. This happens because the algorithm runs instantaneously, but both the input and output storage require 1 clock cycle.

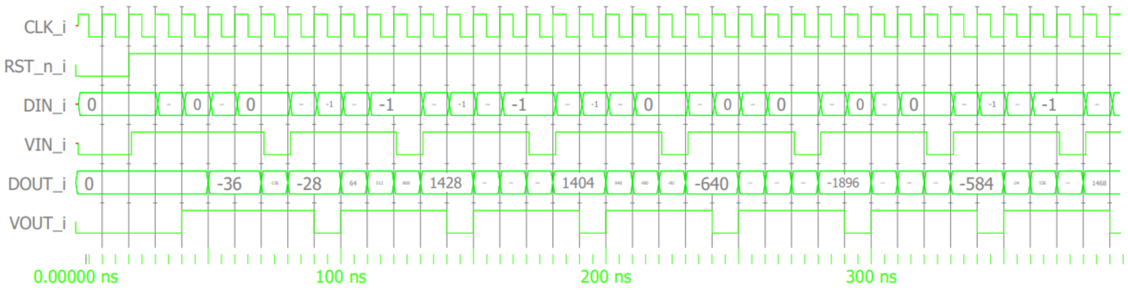


Figure 8: Testbench result of FIR (direct form)

2.3 Logic Synthesis

In order to get information on area, frequency and power consumption of the model, we proceeded to import our architecture in Synopsys Design compiler. The critical path achievable by the design was found through a number of iterations, by varying the input clock frequency between $10ns$ and $0ns$ with the goal of reducing the Slack time to zero. The value found this way is $T_{cp} = 1.85ns$. Therefore the maximum frequency allowed for the design is $f_M = 540.5MHz$. The reports have been generated, as requested, by applying a clock with $f_{ck} = f_M/4 = 135MHz$, and the most relevant results obtained from them are summed up in the following table.

Power Consumption	Total Area	Slack
$287.85\mu W$	$2892.48\mu m^2$	$5.34ns$

2.4 Place and Route

After the optimization and the routing done by Innovus, the above parameters have now changed to:

Power Consumption	Total Area	Slack
$0.9257mW$	$2795.4\mu m^2$	$5.718ns$

While the area has been reduced, and the slack has increased, the most remarkable change from Synopsys to Innovus is the Power consumption, which has roughly tripled. We assume that this was due to the fact that Innovus added interconnections and supply rails, that were not taken into account by Synopsys.

3 Advanced Architecture Development

3.1 Architecture Development

After having evaluated the various characteristic of the behavioral model, the group proceeded to develop a more advanced architecture, by using the Unfolding Technique. Due to the high order of the filter, the first experiments on paper were done on filters of order two and three. Such experiments mostly regarded the comparison between the architecture obtained by unfolding the FIR in its direct form (first approach), and that obtained by unfolding the filter in its transpose form (second approach).

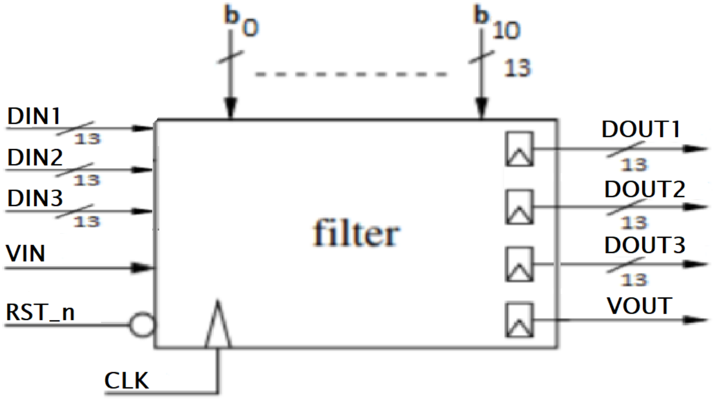


Figure 9: High level block diagram of FIR filter (unfolded form)

The first approach leads to an architecture that can be conveniently divided into a register bank and a part with only the arithmetical operators. The operators, therefore, are never separated by a register. The second approach leads to an architecture with the same number of registers and operators as the first one, however the registers are placed between operators. While the first approach leads to a more regular structure, easier to describe in VHDL and thus may seem like the right choice, the second one has shorter critical paths and therefore it was chosen.

	Adders	Multipliers	Registers	Critical Path Delay
I Approach	30	33	30	$1 \cdot T_{MULT} + 10 \cdot T_{ADD}$
II Approach	30	33	30	$1 \cdot T_{MULT} + 3 \cdot T_{ADD}$

It is worth mentioning at this point that the transpose form seems to retain its advantages over the direct form even after the unfolding. Namely, its critical path delay is independent of the order of the filter (as long as $N \geq 3$, N being the FIR order), and such improvement comes for free, not requiring additional blocks.

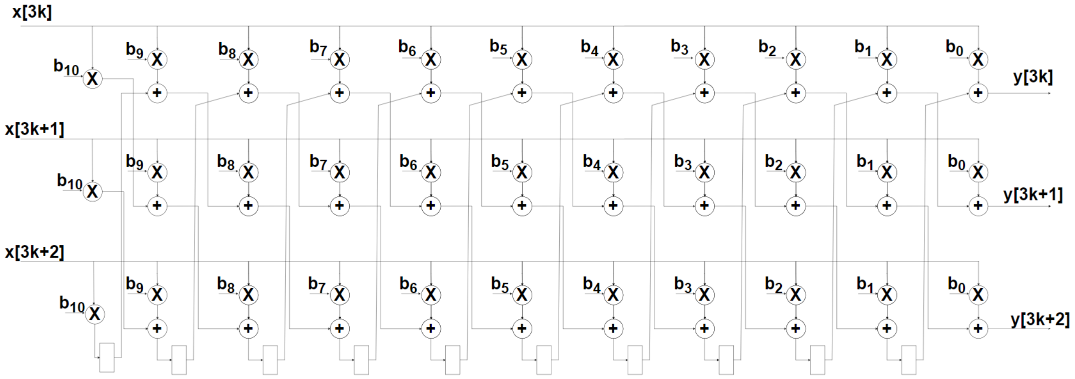


Figure 10: II Approach, non pipelined Architecture

Figure 10 only represents the structure of the filter, meaning the input and output registers have not been drawn here but will still be included in the VHDL implementation of the final architecture. Also, it must be noted that while the unfolding algorithm also added a register at the output y[3k+2], such register was not necessary to calculate the correct output and therefore was not included.

While the structure represented in Figure 10 already displays short critical paths, pipeline stages were added in order to bring the critical path delay to a minimum.

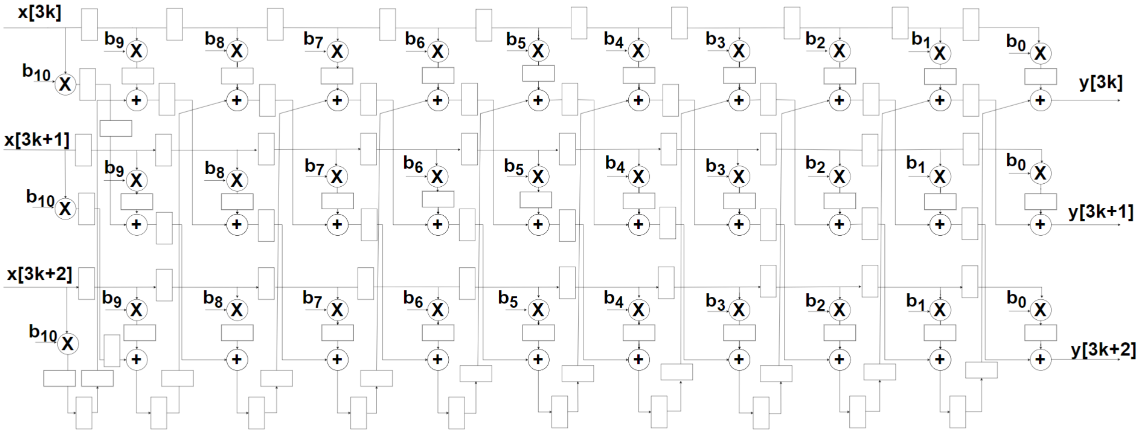


Figure 11: II Approach, pipelined Architecture

With such a choice, the critical path delay is equal to the delay of one arithmetic block only (usually Multiplier takes more time than Adder), and the number of registers has grown to 103 (excluding input and output registers).

For the VHDL code we used the for generate statement to create multiple instances of the block that identifies a ”column” in Figure 10; only the first block, formed by three multipliers and one register, was defined outside the for generate.
Between each of these blocks we then placed a level of pipes and also another level, for

each row, between multiplier and adder.
To generate the VOUT signal we connected VIN to a row of flip-flips.

3.2 Simulation

Also in this case the output text produced by the C model exactly matches the output of the VHDL simulation.
From Figure 12 it is possible to see that, unlike the behavioral case, 14 clock cycles are required to pass from the rising edge of VIN to that of VOUT. An increase in latency was to be expected because in this case we are describing the architecture by components and also because there are many levels of pipes.
In fact, of the 14 clock cycles required, 11 are due to the pipeline inserted between the blocks, 1 is due to the pipeline placed between multiplier and adder and 2 are due to the input and output registers.

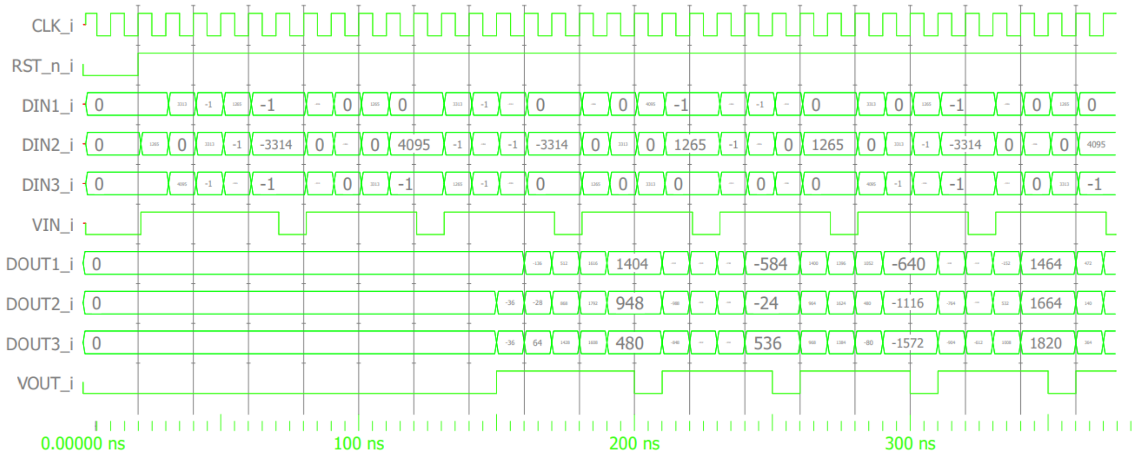


Figure 12: Testbench result of FIR (unfolded form + pipeline)

3.3 Logic Synthesis

Following the same procedures as in Section 2.3, the critical path delay was found to be equal to $1.13ns$; therefore, $f_M = 885MHz$ and $f_M/4 = 221MHz$. By applying a clock equals to $f_M/4$, the following results were obtained.

Power Consumption	Total Area	Slack
$1.5314mW$	$14633\mu m^2$	$3.04ns$

3.4 Place and Route

After the optimization and the routing done by Innovus, the above parameters have now changed to:

Power Consumption	Total Area	Slack
$7.455mW$	$12839\mu m^2$	$2.010ns$

Once again, power consumption has increased dramatically and the area was reduced. This time though, the slack time was reduced.

3.5 Final Considerations

Let us now compare the two architectures developed.

	Operating Frequency	Total Power	Total Area	Slack	Throughput
Behavioral (I)	135 MHz	$0.9257mW$	$2795.4\mu m^2$	$5.718ns$	135 MHz
Unfolded, pipelined (II)	221 MHz	$7.455mW$	$12839\mu m^2$	$2.010ns$	663 MHz

The above table shows the main characteristics of the two structures. In terms of throughput, the version II of the FIR produces outputs roughly five times faster. Such improvement is paid dearly, the Area being 4.5 times larger and the power consumption being eight times higher. While the area increase is almost proportional to the throughput increase, the power consumption of II is so high compared to the structure I that it would deter engineers to choose structure II in most applications.

While Latency was not included in the table, it must be kept in mind that the structure II has a latency of fourteen clock cycles (as it was discussed in section 3.2) against the two clock cycles of structure I. This is not detrimental to filter II, since in its normal operations it works with fairly long streams of data.

Furthermore, in the Behavioral structure the slack time of the critical path would give a lot of room to increase operating frequency, whereas in the Unfolded, Pipelined structure the slack is close to its limit.

Lastly, we believe that by experimenting more with the pipeline stages in the II structure, or with the operating frequency in the I structure, the spectrum of solutions would increase significantly. This goes beyond the scopes of this lab though, in which our goal was simply that of maximizing throughput and minimizing critical path, thus comparing the simplest possible FIR filter architecture with a more complex one.