Politecnico di Torino
III Facoltà di Ingegneria

Lab 1 Report
Integrated Systems Architecture

Master degree in Computer Engineering

Authors: ISA36

Nicole Dai Prà, Leonardo Izzi

November 15, 2020

Many thanks to Prof. Mariagrazia Graziano for providing us with this template.

Contents

CHAPTER 1

Introduction

Our assignment was to implement a 2nd order IIR filter on 12 bits. For this architecture we have implemented, as required, first a Matlab model, then a C model and finally the actual VHDL model. A note on the filter order: to obtain its value we have considered the space character in "Dai Prà".

As required, we have also created a GitHub repository, available at the following link: `https://github.com/leoizzi/isa_labs/tree/main/lab1`.

The folders are organized as follows: in `isa_labs/lab1` there are `iir` and `iir_opt`. The former contains the Matlab model, C model and VHDL model of the standard IIR, while the latter contains the C model and the VHDL model of the improved filter.

CHAPTER 2

IIR

In this chapter the IIR filter is analyzed from its Matlab specification to the physical design.

2.1 Matlab model

We have updated `my_iir_filter.m` to match the value of the given N and n_b values. This script produces the filter's coefficients and saves the quantized filter's input in a file called `samples.txt`, which is then used by the C model (section 2.2) to test the goodness of the implementation. The coefficients we have obtained are:

$$b_0 = 423 \quad b_1 = 846 \quad b_2 = 423 \quad a_0 = 1 \quad a_1 = -757 \quad a_2 = 401$$

After the C model's simulation, its results are read back to measure the *Total Harmonic Distortion* (THD). By keeping the internal representation on 12 bits we have measured $THD \approx -65.46\ dB$, which is better than $THD_{target} = -30\ dB$, hence the whole filter has been implemented on 12 bits.

2.2 C model

In the C model we have only modified the coefficients to match the ones given by Matlab and the shift amount, because a shift of `NB-1` in our case resulted in an internal representation of 13 bits, while as said earlier 12 bits were enough. Therefore, we have added a macro called `INT_REPR` set to 1 which is added to the previous term to obtain the target shift value. The output values have been stored in a file called `results_c_12bit.txt`.

2.3 VHDL model

The filter has been implemented by deriving its structure from the C model and by using behavioral, generic components for the arithmetic operations. The top module, instead, is not generic and its implemented in a structural way. It is contained in the file `iir.vhd`. In figure 2.1 is shown the block diagram of the IIR.

The model entirely reflects the one described in C as it can be seen, the only peculiarity is in the organization of the register's enable signals, not shown in picture 2.1 (along with the input, coefficients and output registers) to reduce the clutter. Since the structure is so small no control unit has been developed. Instead, we have decided to add 1 bit registers to store the delayed value of vin to correctly drive the filter. The `x[n]` register, along with the ones shown in the figure (that correspond to the

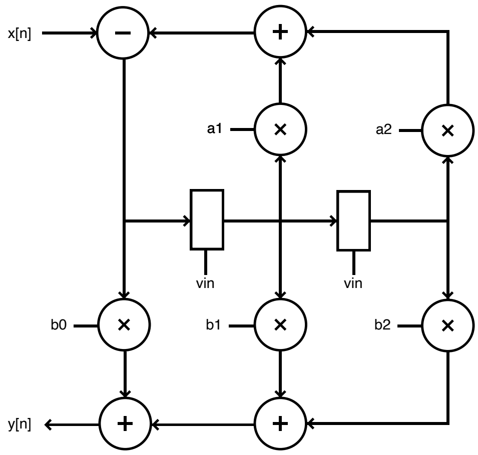

Figure 2.1: IIR direct form II block diagram

C model's sw[i] array) and the coefficients' registers are enabled by `vin` itself. `y[n]` register, on the other hand, takes as enable `vin` delayed by 1 clock cycle since it has to latch the value computed the cycle before. This means that the value of y[n] is available to the filter's interface only the cycle after the register have been enabled, so the signal `vout` corresponds to `vin` delayed by 2 clock cycles.

2.3.1 Simulation

This circuit has been simulated with *ModelSim*, that has written the simulation result in a file called `results_vhdl_12bit.txt`. The `diff` tool has been run on the C model's output and this file to check that the VHDL model was compliant with it. To reproduce the simulation there is a script in the `sim` folder called `sim_script.tcl`.

2.3.2 Logic synthesis

The model described above has been synthesized with *Synopsys Design Compiler*. The synthesis has been run three times:

- **Run 1**: to find the maximum clock frequency we have created a clock with period 0 with the command `create_clock -name my_clk -period 0 clk`. With `report_timing` we have found out that $t_{ck_{min}} = 2.81\ ns$.
- **Run 2**: to verify that $t_{ck_{min}}$ was actually a valid value we run a synthesis with a clock period $t_{ck} = t_{ck_{min}}$. The synthesis met the constraint and the area for this implementation is $A_{ck_{min}} \approx 4258\ \mu m^2$.
- **Run 3**: the synthesis with a clock period $t_{ck} = 4 * t_{ck_{min}} = 11.24\ ns$ was executed. In this case the circuit is able to compute a value in 5.14 ns with a slack of 5.99 ns, with an area $A \approx 3701\ \mu m^2$.

To speed up the synthesis process two scripts are provided: `setup.sh` and `syn_script.tcl`. The former is used to create a clean synthesis environment, the latter to run the synthesis. In this script is possible to choose among the settings of **Run 2**, by setting the variable `use_tx4` to 0, and the settings of **Run 3**, by setting `use_tx4` to 1.

2.3.3 Power consumption estimation

The resulting Verilog file produced in **Run 3** has been simulated again to check that the results were the same as the VHDL model and to obtain the circuit's switching activity. The ModelSim script to calculate the switching activity is ???, that writes the file `iir.sdf` in the `netlist` folder.

This file, then, is read back in Synopsys by using the script `syn_script_pwr.tcl` contained in the `syn` folder. This script outputs `report_switching_activity.txt`, a file containing the estimated power consumption of the IIR based on the switching activity. Extracts of its content are reported below.

```
Cell Internal Power  = 278.3994 uW   (61%)
Net Switching Power  = 177.0917 uW   (39%)
                         ---------
Total Dynamic Power    = 455.4911 uW  (100%)

Cell Leakage Power     =  76.5232 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000             0.0000            0.0000  (   0.00%)
memory             0.0000           0.0000             0.0000            0.0000  (   0.00%)
black_box          0.0000           0.0000             0.0000            0.0000  (   0.00%)
clock_network      0.0000           0.0000             0.0000            0.0000  (   0.00%)
register          83.7543          26.6556         9.8137e+03          120.2236  (  22.60%)
sequential         0.0000           0.0000             0.0000            0.0000  (   0.00%)
combinational    194.6452         150.4362         6.6709e+04          411.7907  (  77.40%)
--------------------------------------------------------------------------------------------------
Total            278.3995 uW      177.0918 uW      7.6523e+04 nW       532.0143 uW
```

2.3.4 Place and route

The Verilog file obtained in the previous step has been used to perform the *Place and route* on *Innovus*. We have rigorously followed all the steps reported in the file `documents.pdf` to complete this part, hence we will not repeat them here.

We have checked that

CHAPTER 3

IIR Lookahead

In this chapter we will explain how we have improved the original IIR design by applying the techniques studied in the course, specifically the lookahead, the pipelining and retiming. We derived the new architecture from the block diagram shown in figure 2.1, then we have modified the C model to obtain a reference implementation and, finally, we have derived the VHDL implementation to reproduce all the steps done in chapter 2. Hence, for this last part, we will outline only the results and compare them to the IIR's one since the procedures we have followed are the same.

3.1 Architecture derivation

First, as mandated by exercise 2.4, we have applied the J-lookahead method with $J = 1$. This is a non-universal technique, that is normally applied when other techniques such as pipelining, retiming, unfolding, ecc. cannot be applied. In our baseline architecture there is actually space for universal techniques, as demonstrated below, although we have applied the lookahead first nevertheless to comply with the exercise's request.

3.1.1 Baseline IIR analysis

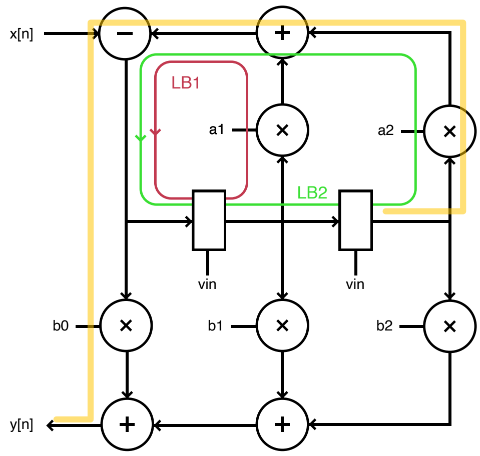

Figure 3.1: Baseline IIR critical path

In figure 3.1 it is outlined in yellow the critical path of the non-optimized filter. If we define with

T_a the delay of the adder and with with T_m the delay of the multiplier, we obtain that

$$CP_{iir} = 2T_m + 3T_a \tag{3.1}$$

It can be also seen in the picture that there are two loops, namely LB_1 and LB_2. It is possible to calculate the loop bound as follows:

$$LB_{iir_1} = \frac{T_m + 2T_a}{1} \qquad LB_{iir_2} = \frac{T_m + 2T_a}{2}$$

Hence, we obtain

$$T_{iir_\infty} = \max\{LB_{iir_1}, LB_{iir_2}\} = LB_{iir_1} < CP_{iir}$$

This means that a universal technique to improve the architecture exists; indeed pipelining could speed-up the architecture, as there is a feed-forward cutset which divides the feed-forward multipliers from the upper part of the circuit. By placing pipeline register there we obtain

$$CP_{iir_{pipeline}} = T_{iir_\infty} = T_m + 2T_a \tag{3.2}$$

This demonstrate that multiple optimization paths do exists for our architecture.

3.1.2 Lookahead application

The lookahead method works by recursively expanding the base equation. As the exercise stated that only one expansion was required, we have proceeded in this way:

The equation of our filter is

$$\begin{aligned} y[n] &= \sum_{k=0}^{2} a_k y[n-k] + \sum_{i=0}^{2} b_i x[n-i] = \\ &= a_0 y[n] + a_1 y[n-1] + a_2 y[n-2] + b_0 x[n] + b_1 x[n-1] + b_2[x-2] \end{aligned} \tag{3.3}$$

We have calculated the formula of $y[n-1]$, which is

$$y[n-1] = a_0 y[n-1] + a_1 y[n-2] + a_2 y[n-3] + b_0 x[n-1] + b_1 x[n-2] + b_2[x-3] \tag{3.4}$$

If we insert 3.4 in 3.3 the result is

$$\begin{aligned} y[n] = a_0 y[n] + a_1 a_0 y[n-1] + a_1^2 y[n-2] + a_1 a_2 y[n-3] + a_1 b_0 x[n-1] + a_1 b_1 x[n-2] + \\ + a_1 b_2 x[n-3] + a_2 y[n-2] + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] \end{aligned} \tag{3.5}$$

Which is our new filter's equation. Its block diagram is shown in figure 3.2.

In equation 3.5 it can be noticed that new coefficients appear in the form $a_1 * a_j$ and $a_1 * b_j$. Their result does not fit in 12 bits, therefore we have decided to truncate them (by taking the 12 MSBs) to respect the original interface's bit width.

Starting from the structure shown in figure 3.2 we have derived a new C and VHDL model: the latter has been used as basis for the optimizations described in section 3.2, the former as golden model to check the results since, due to the new coefficients, the filter has a different THD and produces slightly different values.

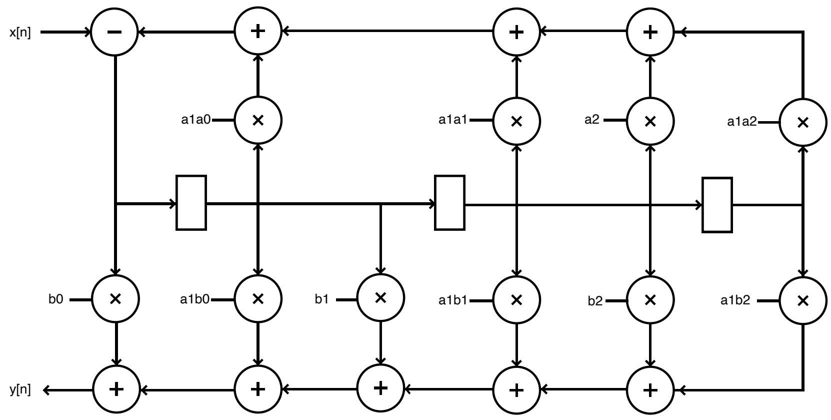

Figure 3.2: IIR lookahead

3.2 Lookahead optimizations

Before applying any universal optimization technique, as always we needed to check if

$$T_{lookahead_\infty} < CP_{lookahead}$$

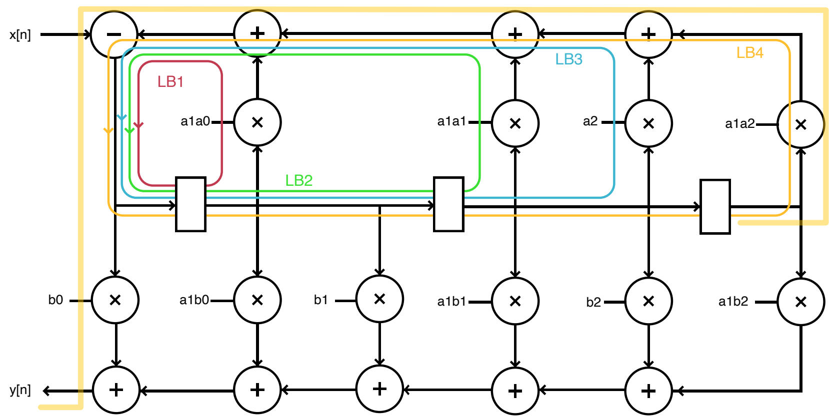

Figure 3.3: IIR lookahead critical path and loop bounds

In yellow it is highlighted the new critical path, which is equal to

$$CP_{lookahead} = 2T_m + 5T_a$$

While in red, green, blue and orange are highlighted all the design's loops, which are respectively LB_1, LB_2, LB_3, LB_4 and are equal to

$$LB_{lookahead_1} = \frac{T_m + 2T_a}{1} \qquad LB_{lookahead_2} = \frac{T_m + 3T_a}{2}$$
$$LB_{lookahead_3} = \frac{T_m + 4T_a}{2} \qquad LB_{lookahead_4} = \frac{T_m + 4T_a}{3}$$

Again, we have calculated $T_{lookahead_\infty}$ as

$$T_{lookahead_\infty} = \max\{LB_{lookahead_1}, LB_{lookahead_2}, LB_{lookahead_3}, LB_{lookahead_4}\} = LB_{lookahead_1}$$

Which is smaller than $CP_{lookahead}$. To have $T_{lookahead_\infty} = CP_{lookahead}$ we have applied a combination of retiming and pipelining, as shown in figure 3.4.

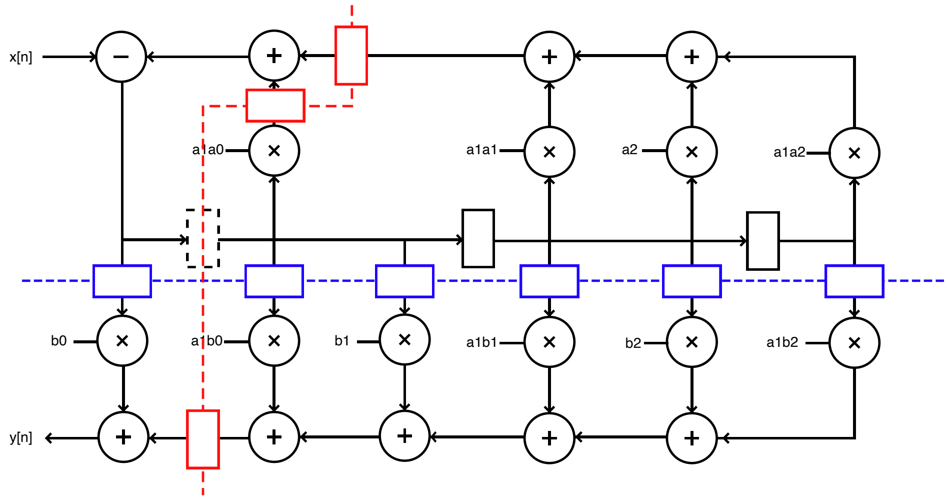

Figure 3.4: Optimized IIR lookahead

First, we applied retiming on the leftmost register, which is the dashed black one, by using the cutset shown in red. With the same color are shown the retiming registers. Furthermore, in blue it is shown the forward cut-set used for pipelining and the related registers. Pipelining, in respect to retiming, changes the behavior of the circuit. Indeed it had consequences on the delay of the *vin* signal, which has been adjusted accordingly. The new critical path has now a delay of

$$CP_{lookahead} = T_m + 2T_a \tag{3.6}$$

Which is better than the one of the baseline IIR described in equation 3.1.

3.3 Results comparison