Projeto de Sistemas Digitais

Trabalho Laboratorial 3

DSP IP core for computation of real-time FIR filter

Rui Graça
Vinícius Ginja

Universidade do Porto
Faculdade de Engenharia
FEUP

1 Introduction

For this project, we set out to optimize a given digital audio IP able to compute digital FIR filters with up to 16384 coefficients, so that it meets timing requirements established by a 48 KHz input signal bandwidth. Both input and output are stereo with 18 bit per channel. To achieve this speed requirement, parallel and pipeline architectures were explored during the development. We also found different solutions that are fast enough but with unequal FPGA resource utilization. The results were verified with behavioral and post-synthesis, as well as FPGA implementation using a signal generator and an oscilloscope. Post-layout simulation failed, probably due to some error in the model of a RAM block, but tests done on the FPGA show that the behavior is correct. The input of IP, as well as the filter coefficients, are represented in two's complement with fixed point with only one non-fractional bit.

2 Design decisions

2.1 Datapath

Since we must be able to compute the response of a filter with up to 16384 coefficients in $\frac{1}{48}$ ns $= 20.8333$ μs, if we set the clock frequency to 100 MHz (clock period of 0.01 μs), we must have $\frac{16384 \cdot 0.01}{20.8333} = 7.8$ MAC (Multiply and accumulate) blocks operating in parallel, computing one product and one accumulation per clock cycle. We conclude easily that it is not feasible to use the RAM blocks given, since they have a bandwidth of 4 coefficients/data samples per clock cycle, and, to use only 4 MAC blocks in parallel, we would have to use a clock frequency of 200 MHz in order to achieve the desired timing requirements, which exceeds the maximum clock frequency in the RAM specification (147 MHz). Our choice was to extend the RAM bandwidth to 8 coefficients/data samples and to use 8 MAC blocks in parallel. The MAC block used was the one given, with small modifications, since some of the bits in the accumulator output of the original block were not used. The MAC block, that is supposed to multiply an input of 18 bits with one other of 36 bits and then to add the product to an accumulator. The block used divides input of 36 bits in an upper part of 18 bits and in a lower part and executes the multiplication and the accumulation of both parts in parallel. Because the FGPA used has Hardware optimized for the computation of MACs with signed 18 bits input (DSP48 blocks), and the lower 18 bits of the 36 bit input should be used in the MAC in an unsigned fashion, operations with the lower 18 bits were further divided, being that the lower 17 bits were multiplied with the 18 bits input, being forced to signed by introducing

0 in the MSB. The 18th bit of the 36 bit input was dealt with individually, being that its influence in the result is given by a multiplexor that outputs 16'b0 if it 0, or the 16 bit input if it is 1. This allows to take advantage of the usage of the DSP48 blocks, optimizing the performance of the MAC. This MAC block has 4 pipeline stages Since the inputs of the multiplier stage are signals of 18 and 36 bits, the output has 54 bits. The output of the MAC is the result of 16384 sums of 54 bit signals, which corresponds to a product of a 54 bit signal with a $log_2(16384) = 14$ bit signal, and so, it is a signal of 68 bits. However, because this implementations is supposed to support FIR filters without gain, and the output of the IP is a signal of 18 bits, the 15 most significant bits of the MAC output are truncated (as well as the lower 35 bits). Because of this, it is unnecessary to implement signals of 68 bits, because the occurrence of overflow in the intermediate computations will not affect the final result (unless overflow occurs in the output, due to filter coefficients that do not respect the no-gain restriction), and the internal signals of the MAC block were truncated to 53 bits. However, this change does not result in any performance improvement, since the synthesis tools successfully detect these bits as unnecessary. Other optimizations were tried in this module, aiming to make the operation the most suitable possible for the DSP48 blocks, with the least possible logic besides the DSP48 itself. However, we soon realized that the block is already quite optimized, and that the RAM blocks will introduce the bottleneck to the critical path, so that optimization in the MAC speed would not be a major benefit for the project.

As referred, both the circular buffer RAM and the RAM for the coefficients were extended to support a bandwidth of 8 coefficients/data samples per clock cycle. This was done by keeping the structure of the original blocks, but adjusting the parameters in order to change the bandwidth. In the circular buffer, it was not required to keep the two physical blocks for each logic RAM block required in the original circular buffer, causing the resulting design to be somewhat simplified in relation to the original.

The final IP is constituted by 8 MAC blocks operating in parallel, followed by the sum of the output of all the MAC blocks. We have tested three options for this sum: using only one pipeline stage, in which all the 8 signals from the MAC outputs are summed, using two pipeline stages, being that the MAC outputs are summed two by two in one stage, and the 4 signals that result from that stage are summed in the following stage, and using three pipeline stages, being that the output of the MAC blocks are summed two by two in the first stage, the 4 outputs of this stage are summed two by two in the second stage, and the two signals that result are summed in the last stage. These three options were compared relatively to time performance and resource usage. Moreover, we have implemented control mechanism for the reset of the MAC blocks in the end of the computation of one output sample and a shift register to set the output ready signal in the clock cycle that the right result outputs the last pipeline stage.

2.2 Control logic

The dataflow is controlled by a 4-state finite-state machine (FSM). There is a steady state called *IDLE* in which the controller waits for the data source (LM4550 codec controller in our case) to communicate the arrival of new data by setting a enable signal. Upon this event, the sample counter and memory addresses are reset and the state changes to *WAIT_ DATA*, which only causes a cycle delay to allow the new data written in the circular buffer be available to read by the multiply-accumulate (MAC) modules, since this is a dual-port memory. After this the FIR filter computation starts,the FSM settles in the *RUN* state throughout the 16384 MAC operations, which are complete in only 2048 cycles due to the 8-way parallel pipelined architecture implemented. The sample counter, data and coefficient memory addresses are simultaneously incremented. Execution would end when the sample counter reaches 2048 if this was a 1-cyle core, but in our case a final *TERMITATE* state is needed to drain the pipeline. It waits for the output data ready signal, set right before the last state transition, at the end of a shift register that is as long as the number of pipeline stages.

3 Simulation and Results

Simulations were done using the testbench given, comparing the output of the filter with a golden output, for a given set of coefficients. We have confirmed that the output equals the golden output all the values tested in the testbench. In the first design stages of the project, simulations were performed only at a behavioral level, whereas in later stages, post-synthesis and post-translate simulations were done. Although these resulted in the expected correct response, post-routing simulations resulted in undefined results at the output, due to some problem in the RAM. We believe that this is due to some problem in the RAM cell module, and not to the project itself, since a correct behavior was obtained in an FPGA.

As referred, several design options were compared. Using three pipeline stages for the summation of the MAC results, we obtained the following results, by post synthesis analysis, with optimization goal set as time, with high effort:

```
Maximum Clock Frequency: 125.455 MHz (7.971ns)
```

```
Logic Utilization                    Used    Available   Utilization
Number of Slice Registers            3711    54576       6%
Number of Slice LUTs                 5306    27288       19%
Number of fully used LUT-FF pairs    3631    5386        67%
Number of Block RAM/FIFO             96      116         82%
Number of BUFG/BUFGCTRLs             1       16          6%
Number of DSP48A1s                   32      58          55%
```

With the same configuration, but with area as the optimization goal, we obtained:

```
Maximum Clock Frequency:  118.99 (8.404 ns)

Logic Utilization                    Used    Available   Utilization
Number of Slice Registers            3646    54576       6%
Number of Slice LUTs                 5297    27288       19%
Number of fully used LUT-FF pairs    3616    5327        67%
Number of Block RAM/FIFO             96      116         82%
Number of BUFG/BUFGCTRLs             1       16          6%
Number of DSP48A1s                   32      58          55%
```

Using 2 pipeline stages in the MAC, with speed as the optimization goal, we obtained:

```
Maximum Clock Frequency: 121.908MHz (8.203ns)

Logic Utilization                    Used    Available   Utilization
Number of Slice Registers            3490    54576       6%
Number of Slice LUTs                 5303    27288       19%
Number of fully used LUT-FF pairs    3266    5527        59%
Number of Block RAM/FIFO             96      116         82%
Number of BUFG/BUFGCTRLs             1       16          6%
Number of DSP48A1s                   32      58          55%
```

And with area as the optimization goal:

```
Maximum Clock Frequency: 118.948 MHz (8.407ns)

Logic Utilization                    Used    Available   Utilization
```

```
Number of Slice Registers           3433    54576       6%
Number of Slice LUTs                5297    27288       19%
Number of fully used LUT-FF pairs   3404    5326        63%
Number of Block RAM/FIFO            96      116         82%
Number of BUFG/BUFGCTRLs            1       16          6%
Number of DSP48A1s                  32      58          55%
```

Using only one pipeline stage, with speed as optimization goal, we obtained:

```
Maximum Clock Frequency: 107.980 MHz (9.261ns)

Logic Utilization                   Used    Available   Utilization
Number of Slice Registers           3049    54576       5%
Number of Slice LUTs                5294    27288       19%
Number of fully used LUT-FF pairs   2593    5750        45%
Number of Block RAM/FIFO            96      116         82%
Number of BUFG/BUFGCTRLs            1       16          6%
Number of DSP48A1s                  32      58          55%
```

And using area as the optimization goal:

```
Maximum Clock Frequency: 108.036MHz (9.256ns)

Logic Utilization                   Used    Available   Utilization
Number of Slice Registers           3010    54576       5%
Number of Slice LUTs                5297    27288       19%
Number of fully used LUT-FF pairs   2980    5327        55%
Number of Block RAM/FIFO            96      116         82%
Number of BUFG/BUFGCTRLs            1       16          6%
Number of DSP48A1s                  32      58          55%
```

Based on these results, the usage of 2 stages seemed a reasonable option, and it was used for the generation of the next stages of the implementation. However, post-routing simulations suggest that the usage of three stages is a better option, when trying to accomplish the desired time performance.

Optimizing with a goal of 100 MHz for the clock frequency, we obtained:

```
Maximum Clock Frequency: 100.311MHz (9.969ns)
```

```
Slice Logic Utilization         Used    Available   Utilization
Number of Slice Registers       3590    54576       6%
Number of Slice LUTs            4492    27288       16%
Number of occupied Slices       1409    6822        20%
Number of MUXCYs used           3208    13644       23%
Number of LUT Flip Flop pairs   5054
Number of RAMB16BWERs           96      116         82%
Number of BUFG/BUFGMUXs         1       16          6%
Number of DSP48A1s              32      58          55%
```

The tools were, however, unable to optimize to a goal of 111.11 MHz, resulting in:

```
Maximum Clock Frequency: 104.178MHz (9.599ns)
```

```
Slice Logic Utilization         Used    Available   Utilization
Number of Slice Registers       3590    54576       6%
Number of Slice LUTs            4496    27288       16%
Number of occupied Slices       1444    6822        21%
Number of MUXCYs used           3208    13644       23%
Number of LUT Flip Flop pairs   4962
Number of RAMB16BWERs           96      116         82%
Number of BUFG/BUFGMUXs         1       16          6%
Number of DSP48A1s              32      58          55%
```

Using three pipeline stages, this goal was successfully achieved, with the following results:

```
Maximum Clock Frequency: 111.520MHz (8.967ns)
```

```
Slice Logic Utilization         Used    Available   Utilization
Number of Slice Registers       3712    54576       6%
Number of Slice LUTs            4612    27288       16%
Number of occupied Slices       1466    6822        21%
Number of MUXCYs used3320       13644   24%
Number of LUT Flip Flop pairs   5060
Number of RAMB16BWERs           96      116         82%
Number of BUFG/BUFGMUXs         1       16          6%
Number of DSP48A1s              32      58          55%
```

Optimization for higher frequencies could not be successfully obtained by the tools.

We see that with a small increase in the resource utilization, a higher frequency was obtained by the design with three pipeline stages. This implementation was tested in the FPGA, using a signal generator to generate the input and an oscilloscope to measure the output. We observed that the filter behavior was the expected when introducing both sine and square waves. However, we observed that overflow occurred when signals over a certain threshold were used. Analyzing the filter response, we observed that it shows gain for certain frequencies, which explains the overflow in the output.