FINITE IMPULSE RESPONSE FILTER POWER REDUCTION THROUGH ARCHITECTURE OPTIMIZATION

C. M. ALBINĂ

In this paper is presented a digital finite impulse response (FIR) filter using the standard digital design workflow. The advantages and disadvantages of several architectures and of the circuit modeling were discussed using a standard toggle-based method for the circuit power estimation, gate-level simulations and synthesis. The main idea was to show that we can achieve up to 60% power reduction from the beginning by carefully selecting the right architecture and optimizing the VHDL code description of the module. The analysis was made based on the unity delay model and not on the physical extracted layout for a 150 nm CMOS technology and $V_{DD}$ 1.5 volts.

Keywords: FIR filter, synthesis, RTL, architecture, power

1. Introduction

Over the years, the state-of-the-art technologies pushed the VLSI chips to higher clock speed and packing density. The trends for the coming years are defined already by the International Technology Roadmap for Semiconductors ITRS. Applications using Digital Signal Processing (DSP) continue to expand, driven by trends such as the increased use of video, audio and still images and the demand for increasingly reconfigurable systems such as Software Defined Radio (SDR). Many of these applications combine the need for significant DSP processing with cost sensitivity, creating demand for high-performance, low-cost DSP solutions.

1 Dipl.-Ing., Gesellschaft für Mikroelektronik-Entwicklungen mbH, Unterhaching, Germany, IEEE Member, E-mail: cristian.albina@gmail.com
As indicated in Table 1, for most aggressively scaled DRAM, the integration scale will reach 720 Millions of transistors by the year 2012 [1].

<table>
<thead>
<tr>
<th>Year</th>
<th>1997</th>
<th>1999</th>
<th>2001</th>
<th>2003</th>
<th>2006</th>
<th>2009</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology (μm)</td>
<td>0.25</td>
<td>0.18</td>
<td>0.15</td>
<td>0.13</td>
<td>0.10</td>
<td>0.07</td>
<td>0.05</td>
</tr>
<tr>
<td>Nr. Of Transistors (Mil.)</td>
<td>11</td>
<td>21</td>
<td>40</td>
<td>76</td>
<td>200</td>
<td>520</td>
<td>720</td>
</tr>
<tr>
<td>On Chip CLK (MHz)</td>
<td>750</td>
<td>1200</td>
<td>1400</td>
<td>1600</td>
<td>2000</td>
<td>2500</td>
<td>2800</td>
</tr>
<tr>
<td>Wafer Area (mm²)</td>
<td>300</td>
<td>340</td>
<td>385</td>
<td>430</td>
<td>520</td>
<td>620</td>
<td>650</td>
</tr>
<tr>
<td>Wiring levels</td>
<td>6</td>
<td>6-7</td>
<td>7</td>
<td>7</td>
<td>7-8</td>
<td>8-9</td>
<td>9-10</td>
</tr>
<tr>
<td>V_{DD,logic} [V]</td>
<td>2.5-1.8</td>
<td>1.8-1.5</td>
<td>1.5-1.2</td>
<td>1.5-1.2</td>
<td>1.2-0.9</td>
<td>0.9-0.6</td>
<td>0.6-0.5</td>
</tr>
<tr>
<td>T_{eq,equivalent}</td>
<td>4-5nm</td>
<td>3-4nm</td>
<td>2-3nm</td>
<td>2-3nm</td>
<td>1.5-2nm</td>
<td>1.5nm</td>
<td>1nm</td>
</tr>
</tbody>
</table>

While a vast array of digital signal processing functions are implemented by designers, Finite Impulse Response (FIR) filters, Infinite Impulse Response (IIR) filters, Fast Fourier Transforms (FFTs) and mixers are common to many applications. Fast Fourier Transforms are used for a variety of applications, ranging from image compression to determining the spectral content of a data sample. Each of these functions requires a combination of multiply elements along with addition, subtraction and accumulation. Because the power consumption has become an important factor in the design process of a chip due to the limited lifetime of the battery fast and accurate power estimation tools are needed for each level of the circuit in order to ensure that the energy and the design constraints are met. There are already several power estimation tools [2], [3], [4] which can help the designer to this task but the more detailed the simulation is the longer the time and the bigger the amount of data to be simulated. In this paper I’ll use the standard power estimation for the CMOS circuits using gate-level and RTL (Register Transfer Level) simulations combined with Synopsys [5] synthesis tool to show the effect of the RTL architecture and of the VHDL description on the final area of the module as well as on the total power, solution which offers the advantage of small designs with lowest power consumption and which are highly correlated to the final layout.

2. Digital finite impulse response filter concept

The finite impulse response filter stores a series of n data elements, each delayed by an additional cycle. These data elements are commonly referred to as taps. Each tap is multiplied by a coefficient and the results summed to produce the output. Some implementations perform all the multiplications in parallel. More generally, the implementation is broken down into N stages, with an accumulator passing the partial result from one stage to the next. This implementation trades
speed for functional resources, taking N computation stages and requiring n/N multipliers. Depending upon whether the coefficients are static or dynamic and the design of the coefficient values, there are a number of other design optimizations commonly used that are beyond the scope of this article.

A finite impulse response (FIR) digital filter is called 'finite' because its response to an impulse ultimately settles to zero. This is in contrast to infinite impulse response filters which have internal feedback and may continue to respond indefinitely. The FIR transfer function contains M poles for \( z = 0 \). Since all poles are at the origin, all poles are located within the unit circle of the \( z \)-plane; therefore all FIR filters are stable. This useful property together to the fact that they don’t require feedback which can cause rounding errors in the summed iterations and because they’re having a linear and minimum phase make sometimes this type of filter preferable to an IIR filter. The direct-form (Fig. 1) and transpose-form (Fig. 2) structures are most commonly used to implement FIR filters.

3. Digital FIR implementations using different architectures

To design a filter means to select the coefficients such that the system has specific characteristics. The required characteristics are stated in filter specifications. Most of the time filter specifications refer to the frequency response of the filter. I decided to used in this paper a 29th order single channel FIR (30 coefficients) with a system clock frequency of 100 MHz and for the evaluation I used the input and output sample rate equal to 100MHz/32 instead of 30 due to the easy implementation with 5 bits. The filter pass-band is 1 MHz and the stop-band is 1.2 MHz at which the filter attenuation is -60 dB. The whole module was implemented using a 150nm CMOS technology models and \( V_{DD} \) 1.5
volts. In each case the synthesis parameters were: clock period 10ns, critical path and area analysis, use enclosed wire load model, and I used clock gating option for the synthesis.

3.1 Direct implementation based on the C++ description

In most cases the digital designer starts his implementation based on the direct behavioral description delivered by the system concept engineer. This description is usually written in C++ language and has no hardware related physical parameters included. The filter architecture is illustrated in Fig. 3 and contains a shift register, a coefficient ROM and a multiply and accumulate unit (MAU).

![Fig. 3. Digital FIR architecture based on direct C++ code implementation](image)

![Fig. 4. Digital FIR architecture simulation results](image)

After running the gate-level and the RTL simulations like in Fig. 4 we obtain the following power consumption summary for this straightforward implementation:
Table 2

<table>
<thead>
<tr>
<th>Hierarchy Level</th>
<th>Switch Power (mW)</th>
<th>Internal Power (mW)</th>
<th>Leak Power (µW)</th>
<th>Total Power (mW)</th>
<th>Power %</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td>6.649</td>
<td>10.61</td>
<td>15.34</td>
<td>17.279</td>
<td>100</td>
</tr>
<tr>
<td>Top level FIR logic and ROM</td>
<td>2.810</td>
<td>4.273</td>
<td>2.869</td>
<td>7.594</td>
<td>42.4</td>
</tr>
<tr>
<td>Adder inside MAU</td>
<td>0.641</td>
<td>0.309</td>
<td>0.147</td>
<td>1.239</td>
<td>12.5</td>
</tr>
<tr>
<td>Multiplier inside MAU</td>
<td>3.186</td>
<td>4.596</td>
<td>3.452</td>
<td>7.785</td>
<td>45.1</td>
</tr>
</tbody>
</table>

3.2 Better FIR implementation using a Read Decoder

In order to reduce the module power consumption we’ll have to try to minimize the number of cycles necessary to read the filter coefficients out of the ROM memory. One way to achieve this is by carefully choosing the coefficients and their position inside the memory in such a way that the whole process takes only half of the time like in the architecture illustrated in Fig. 5 and by using a read decoder register. The decoder uses the fact that the filter is symmetric, and that two samples can be added before multiplication with the same coefficient. Reducing the number of multiplication by 2 we can reduce the dynamic power consumption significantly.

![Fig. 5. Improved Digital FIR architecture using a Read Decoder](image-url)

Redoing the RTL simulations, synthesis and gate-level simulations like in Fig. 6 we obtain the following results for the module power consumption:
Fig. 6. Improved Digital FIR architecture simulation results

We can notice the reduction of the filter convolution activity on the data bus, second signal in the diagram underneath the clock signal. Furthermore there is an additional switching and internal power reduction inside the MAU (multiplier and adder) and on the top level due to the fact that the implementation needs less digital logic on the top level, most of the operations being already done by the read-decoder.

Table 3

<table>
<thead>
<tr>
<th>Hierarchy Level</th>
<th>Switch Power (mW)</th>
<th>Internal Power (mW)</th>
<th>Leak Power (µW)</th>
<th>Total Power (mW)</th>
<th>Power %</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td>3.495</td>
<td>4.543</td>
<td>15.72</td>
<td>8.054</td>
<td>100</td>
</tr>
<tr>
<td>Top level FIR logic and ROM</td>
<td>0.63</td>
<td>0.753</td>
<td>0.695</td>
<td>1.368</td>
<td>17.4</td>
</tr>
<tr>
<td>Adder inside MAU</td>
<td>6.47 x 10^{-2}</td>
<td>0.242</td>
<td>0.338</td>
<td>0.307</td>
<td>3.8</td>
</tr>
<tr>
<td>Partial adder</td>
<td>6.01 x 10^{-2}</td>
<td>0.176</td>
<td>0.414</td>
<td>0.264</td>
<td>2.9</td>
</tr>
<tr>
<td>Multiplier inside MAU</td>
<td>2.683</td>
<td>3.365</td>
<td>3.277</td>
<td>6.051</td>
<td>75.1</td>
</tr>
<tr>
<td>Read decoder</td>
<td>5.71 x 10^{-2}</td>
<td>6.48 x 10^{-3}</td>
<td>1.88 x 10^{-2}</td>
<td>6.36 x 10^{-3}</td>
<td>0.8</td>
</tr>
</tbody>
</table>

3.3 Optimal FIR implementation using a Read/Write decoder

A lot of parallel architectures and additional algorithm improvements were discussed in the past years in the [7]-[10]. For the further power reduction we can use an additional write decoder combined with a special read/write pointer (Circular Buffer) which reduces the toggle activity. Such architecture is proposed in the Fig. 7. In contrast to the previous implementations, the additional Write Decoder selects only one address location of the Circular Buffer and writes the new sample into it. This mechanism avoids the complete rotation of the registers like it happens in the previous shift register implementations and therefore the amount of toggles gets reduced. Once the new incoming data is stored, the write
Finite impulse response filter power reduction through architecture optimization

pointer is incremented and points now to the next address location of the Circular Buffer.

![Diagram](image)

Fig. 7. Optimal Digital FIR architecture with additional Write Decoder

The VHDL code was implemented using the diagram from Fig. 8.

![Diagram](image)

Fig. 8. Optimal Digital FIR implementation flowchart
To implement the shift register and the circular buffer with read/write pointers I used the following VHDL source code:

```
process (clk, res)
begin
if (res = '0') then -- asynchronous reset
    ShiftReg <= (others => (others => '0'));
elsif (clk'event and clk = '1') then
    if (syncres = '1') then -- synchronous reset
        ShiftReg <= (others => (others => '0'));
    else
        for i in 29 downto 1 loop
            ShiftReg(i) <= ShiftReg(i-1);
        end loop;
        ShiftReg(0) <= data_in;
    end if;
end if;
end if;
end process;
```

```
process (clk, res)
begin
if (res = '0') then -- asynchronous reset
    CircBuffer   <= (others => (others => '0'));
    writepointer <= (others => '0');
elsif (clk'event and clk = '1') then
    if (syncres = '1') then -- synchronous reset
        CircBuffer   <= (others => (others => '0'));
        writepointer <= (others => '0');
    else
        if (valid_in='1') then -- write data, clock gating enable
            CircBuffer(to_integer(writepointer)) <= data_in;
            writepointer <= writepointer + 1;
        end if;
    end if;
end if;
end process;
```

Based on this flowchart I decided to use the following architecture presented in Fig. 9 for the final implementation of a dual channel digital FIR (I and Q) to insure that I can achieve the minimum power consumption required for the implementation of the module inside a mobile device and also the optimal area inside the chip.
Based on the simulation results from the Fig. 10 we’ve got the following results for the module power consumption:
Fig. 10. Optimal Digital FIR simulation results

Table 4

Optimal FIR Architecture Power Consumption

<table>
<thead>
<tr>
<th>Hierarchy Level</th>
<th>Switch Power (mW)</th>
<th>Internal Power (mW)</th>
<th>Leak Power (µW)</th>
<th>Total Power (mW)</th>
<th>Power %</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td>3.007</td>
<td>4.189</td>
<td>17.69</td>
<td>7.214</td>
<td>100</td>
</tr>
<tr>
<td>Top level FIR logic and ROM</td>
<td>0.538</td>
<td>0.673</td>
<td>1.358</td>
<td>1.225</td>
<td>17.9</td>
</tr>
<tr>
<td>Adder inside MAU</td>
<td>4.17 x 10^{-2}</td>
<td>0.153</td>
<td>0.253</td>
<td>0.195</td>
<td>2.7</td>
</tr>
<tr>
<td>Multiplier inside MAU</td>
<td>2.293</td>
<td>3.143</td>
<td>3.277</td>
<td>5.44</td>
<td>75.4</td>
</tr>
<tr>
<td>Circular Buffer read decoder</td>
<td>2.03 x 10^{-2}</td>
<td>4.06 x 10^{-3}</td>
<td>9.78 x 10^{-3}</td>
<td>2.43 x 10^{-2}</td>
<td>0.3</td>
</tr>
<tr>
<td>Circular Buffer write decoder</td>
<td>1.49 x 10^{-3}</td>
<td>2.13 x 10^{-3}</td>
<td>9.78 x 10^{-3}</td>
<td>3.63 x 10^{-3}</td>
<td>0.1</td>
</tr>
<tr>
<td>Circular Buffer 5 bit inc/dec</td>
<td>6.83 x 10^{-3}</td>
<td>2.59 x 10^{-2}</td>
<td>5.25 x 10^{-2}</td>
<td>3.28 x 10^{-2}</td>
<td>0.5</td>
</tr>
<tr>
<td>Read decoder</td>
<td>4.64 x 10^{-2}</td>
<td>5.48 x 10^{-3}</td>
<td>1.41 x 10^{-2}</td>
<td>5.19 x 10^{-2}</td>
<td>0.7</td>
</tr>
<tr>
<td>Partial adder</td>
<td>4.87 x 10^{-2}</td>
<td>0.126</td>
<td>0.194</td>
<td>0.175</td>
<td>2.4</td>
</tr>
</tbody>
</table>

4. Conclusions

Summarizing the three implementation methods we can see that by careful architecture and code implementation we can significantly reduce the total module power consumption up to 60% without a big increase of the circuit area, just by reducing the amount of toggles. For the presented architectures I used SR (shift register) fully (32bit) or half (16 bit) addressed, CG (clock gating), CB (circular buffer) and parallel processing of the coefficients using pipe-line structures. This filter was already implemented using a 150 nm CMOS technology and the measurements done on the final silicon confirmed the simulation results for the main filter parameters.
Finite impulse response filter power reduction through architecture optimization

Table 5

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Area (mm²)</th>
<th>Toggle Count (Mils Tc)</th>
<th>Used Registers</th>
<th>Clock gate efficiency</th>
<th>Power (mW)</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR1</td>
<td>0.050</td>
<td>64.2</td>
<td>568</td>
<td>0.0%</td>
<td>17.279</td>
<td>SR, full</td>
</tr>
<tr>
<td>FIR2</td>
<td>0.054</td>
<td>33.7</td>
<td>589</td>
<td>90.3%</td>
<td>8.054</td>
<td>SR, half, pipe, CG</td>
</tr>
<tr>
<td>FIR3</td>
<td>0.057</td>
<td>30.2</td>
<td>604</td>
<td>99.5%</td>
<td>7.214</td>
<td>CB, half, pipe, CG</td>
</tr>
</tbody>
</table>

If we introduced the simulated values of the VLSI schematic of the implemented filters obtained during synthesis and gate-level simulations in the Matlab waveform viewer we can visualize the filter frequency response to an in-band signal as well as the rejection of an off-band signal as presented in Fig. 11.

Fig. 11. Digital FIR frequency domain analysis of an in-band and off-band signal

There is a lot of work going around trying to find even better architectures and better coefficient quantization methods to improve furthermore the overall power consumption combined in parallel with the usage of modern submicron technologies. The general tendency is towards low power modules combined with intelligent algorithms for easy VLSI implementation.
REFERENCES