An FPGA Implementation of Goertzel Algorithm

Conference Paper · August 1999

DOI: 10.1007/978-3-540-48302-1_35 · Source: DBLP

1 author:

Tomáš Dulík
Tomas Bata University in Zlín
31 PUBLICATIONS **100** CITATIONS

SEE PROFILE

An FPGA Implementation of Goertzel Algorithm

Tomas Dulik

Technical University of Brno,
Faculty of Electroengineering and Computer Science,
Department of Computer Science, Bozetechova 2, 612 66 Brno,
Czech Republic
dulik@dcse.fee.vutbr.cz
http://www.fee.vutbr.cz/~dulik

Abstract. Field Programmable Gate Arrays (FPGAs) has already proven to be the best solution in cases, where hardware flexibility and/or reprogrammability was required. Recently, with the growth of their capabilities and speed, a new kind of completely different applications has arosen. This paper describes a FPGA implementation of Goertzel algorithm and its application - a multichannel DTMF (Dual Tone Multi Frequency) decoder, that can outperform a middle-class DSP processor based system by an order of magnitude.

1 Introduction

Goertzel algorithm is used for Discrete Fourier Transformation (DFT) coefficients computation. Its core consist of bank of IIR filters, whose resonance frequencies are equal to the discrete frequencies of the transformation. For example, a two-point DFT that computes weights of frequencies f_1, f_2 can be computed by Goertzel algorithm with two IIR filters with resonance frequencies f_1 and f_2, respectively. The filters have both poles on the unit circle, so they amplify harmonics with frequency equal to the resonant frequency of the filter and attenuate all the other harmonics. The signal flow of one IIR Goertzel filter is on Fig. 1.

This core must be used repeatedly for all the analyzed frequencies. As this seems to be a disadvantage when implementing in software, this approach gives a great opportunity for parallel fashion of computation in dedicated FPGA hardware.

The two taps of the IIR filter are formed by two registers, an adder and a constant multiplier. Since all of these functions can be implemented effectively in Xilinx FPGAs that have the RAM (ROM) feature (their logic resources can be configured as distributer RAM or ROM cells), it is obvious that if only few discrete frequencies are needed, the computation can be made fully parallel by placing one Goertzel IIR core per frequency point on the chip.

The second part of the algorithm, that is triggered after every N samples are passed to the IIR filter, is the evaluation (N determines the accuracy of the

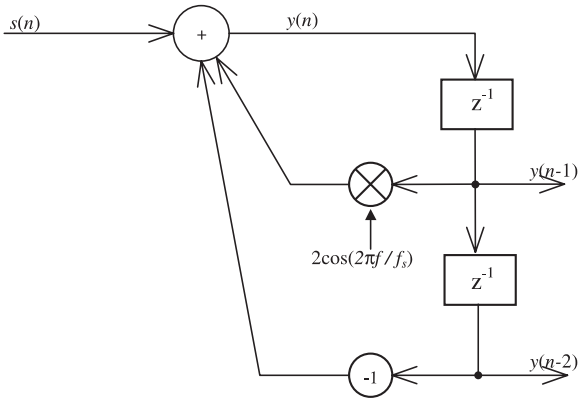

Fig. 1. The Goertzel algorithm IIR core. The $s(n)$ stands for sequence of input signal samples, while $y(n)$ forms the output. The constant $2\cos(\frac{2\pi f}{f_s})$ is dependent only on the sampling frequency f_s and analyzed frequency f.

analyze and its value depends on application demands – e.g., a DTMF decoder would have $100 < N < 300$). It must be done only if there is a need for exact DFT magnitude of the frequency and can be skipped, if the goal is just a frequency detection. Also, there is no need for implementing the magnitude computation in the FPGA - the required data-throughput of this operation is always at least 100 times lower than that one of the IIR filter. Therefore, it is better to concentrate on accelerating the IIR filter computation in the FPGA, while the magnitude computation can be done by a cheap microcontroller or DSP processor. The magnitude M can be computed as

$$M = \left| y(n) - y(n-1).e^{-j2\pi\frac{f}{f_s}} \right| . \tag{1}$$

where $y(n)$ are the output samples, f is the frequency analyzed and f_s is sampling frequency.

The function of one Goertzel IIR filter is demonstrated on Fig. 2. In the upper half of it, there are two filter responses to different harmonics - on the left, the frequency of the harmonic is equal to the resonant frequency of the filter (941 Hz) and thus the filter starts to oscillate on that frequency with the amplitude of the oscillation growing, on the right the frequency is different (1209 Hz). In the lower half of the picture, there is the frequency response of the filter, that was computed point by point using the magnitude evaluation for different input frequencies. On the left, the length of each input sequence was 200 samples, on the right it was 400 samples. The sampling frequency was 8000 Hz – the same as in the most of digital telephone systems – and the filter resonant frequency is taken from the DTMF system, which defines signals for dialing numbers in modern telephones.

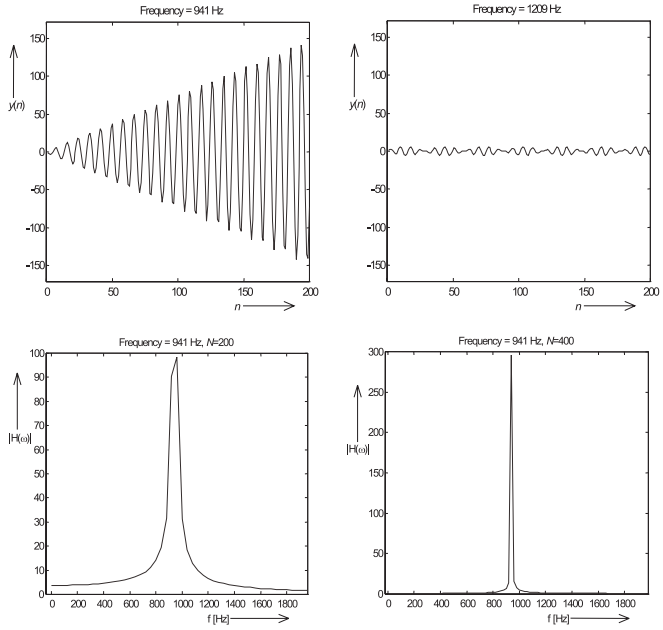

Fig. 2. Output responses of one Goertzel IIR filter

2 FPGA and Constant Coefficient Multiplier

As multipliers always create bottleneck in any FPGA design, there have always been research efforts to reduce the overhead – computation time and occupied chip area. E.g., Prof. Mintzer in [2] proposed to use distributed arithmetics for constant coefficient multipliers. His approach became standard in all modern FPGA designs because it claims the least FPGA resources of all the other methods introduced before that one, while keeping the computation time as low as 1 CLB delay.

2.1 Distributed Arithmetics

Distributed arithmetics can be used anywhere where a quick and space efficient *Multiply and Accumulate (MAC)* operation is needed. The *MAC* operation on sequence of N samples x_n and N coefficients A_n is defined as:

$$MAC(x, A) = \sum_{i=0}^{N} x_i A_i = x_0 A_0 + x_1 A_1 + \ldots + x_N A_N \ . \tag{2}$$

This can be decomposed to the bit level:

$$
\begin{aligned}
MAC(x,A) = {} & x_{00}A_0 2^1 + x_{01}A_0 2^2 + \ldots + x_{0B}A_0 2^B + \\
& x_{10}A_1 2^1 + x_{11}A_1 2^2 + \ldots + x_{1B}A_1 2^B + \\
& + \ldots + \\
& x_{N0}A_N 2^1 + x_{N1}A_N 2^2 + \ldots + x_{NB}A_N 2^B \ .
\end{aligned} \tag{3}
$$

where x_{ij} is the j-th bit of i-th sample and B is bit width of the sample (number of bits). Implementation of this schema using standard logic is straightforward, as pictured on Fig. 3.

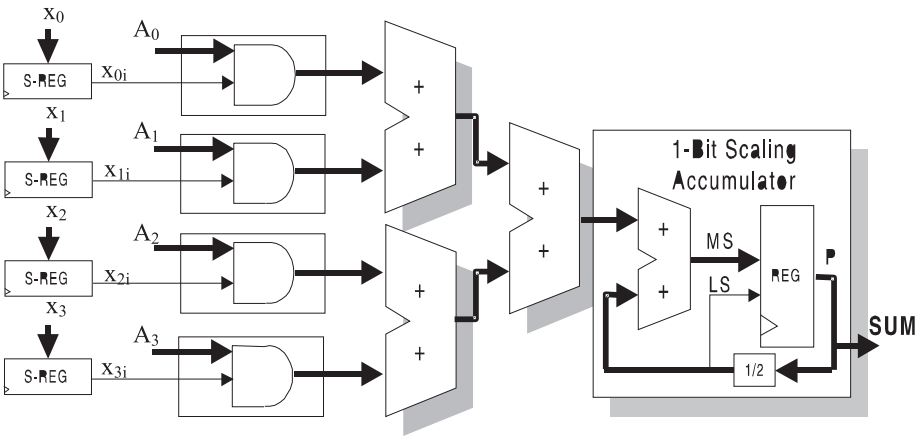

Fig. 3. The *MAC* operation using standard logic (picture taken from [4])

The computation of the *MAC* is done in mixed serial-parallel way, with parallel adders and serial multipliers (the input bits are shifted into an array of AND gates, whose other inputs are the bits of A_n. However, this can be further optimized when noticing fact, that the whole block of 4 AND arrays and following adders is in fact a 4-input logic function, that can be easily implemented in a ROM look-up table 16xB bits. When considering the possibility of using ROM feature of a FPGA, it is obvious that this operation can be done very effectively (see Fig. 4).

The 4 input look-up table easily maps to the Xilinx's XC4000 series FPGAs, taking one CLB per 2 bits of the table bit width. The operation of this circuit can be further parallelized by computing 2 or more bits a time, but there is always the trade-off with chip area occupied. The Scaling Accumulator block accumulates the partial results, scaling the them down by factor 2 each step. It also includes sign extension logic (for deeper description see [1]).

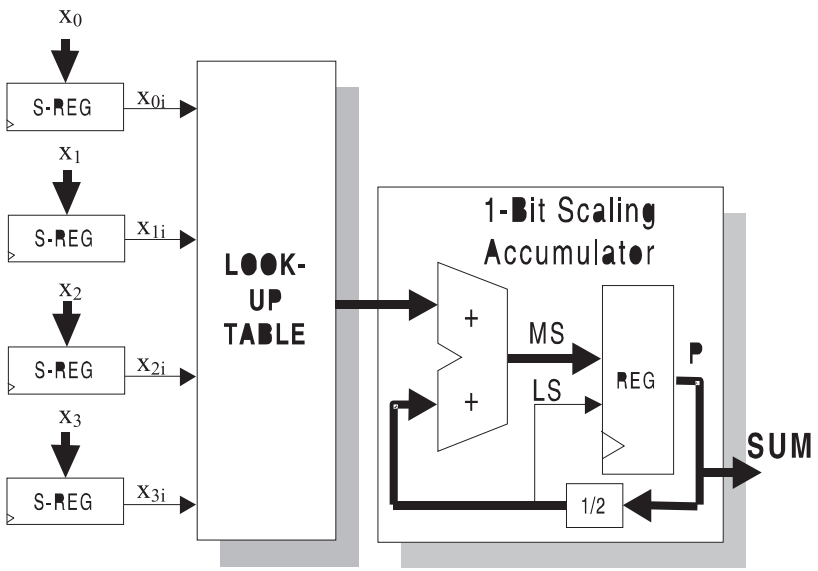

Fig. 4. The *MAC* operation using FPGA's ROM lookup table (taken from [4])

3 The IIR Core of Goertzel Algorithm and FPGA

Since the IIR core determines the overall throughput of the whole algorithm – it must be computed every sample, while the final evaluation is triggered only once per N samples, where $N > 100$ – we focus to the core algorithm only. The IIR core of Goertzel algorithm must complete following operation for getting one output sample $y(n)$:

$$y(n) = y(n-1)2\cos(2\pi f/f_s) - y(n-2) + s(n)\ . \tag{4}$$

This equation is in fact a *MAC* operation with coefficients $2\cos(\ldots)$, -1 and $+1$. However, it would be inefficient to implement it the way presented on Fig. 4 – one lookup table input would stay unused. It is more efficient to choose another solution – either computing 2 bits at once, or even 4 bits. The final choice depends on application circumstances, e.g. number of frequencies analyzed.

3.1 "Two bits at once" version

The possible solution is shown on Fig. 5:

This is typical example of distributed arithmetics – LUT (look-up table) block with the RFA (Registered Full Adder) computes the sum of products $y(n-1)2\cos(\ldots) + k.y(n-2)$, where k is constant< 0, e.g. -256. This constant makes the term $y(n-2)$ being aligned properly, because the format of

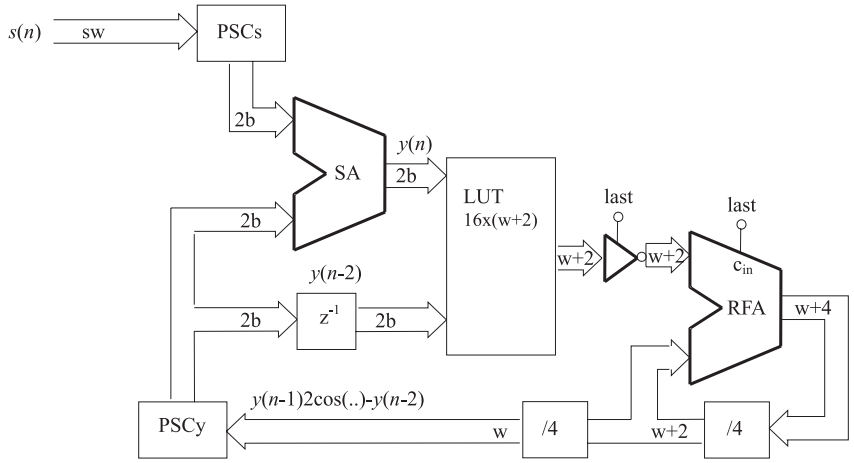

Fig. 5. "Two bits at once" version

numbers used is fractional integer with the point in the MSB position. Input samples enter the PSCs (Parallel to Serial Converter), which converts them to 2-bits-wide serial stream (from the LSB). If needed, the PSC block together with SA (serial adder) can carry out conversion to the two's-complement code too. The serial adder then adds 1 to the inverted value of every negative input samples. Otherwise, the SA only adds input samples to the intermediate result $y(n-1)2\cos(\ldots) + k.y(n-2)$. The input marked "last" is part of the sign extension logic (for closer description see [1]).After N input samples are proceeded, the result can be read from the register of RFA and all the registers can be reset to allow the next evaluation period to start.

The whole unit runs in pipelined fashion, which enables using the fastest clock rates possible. The width of output samples $y(n)$ and intermediate results $y(n-1)$ and $y(n-2)$ is dependent on the width of the input samples sw - it must be wide enough to prevent the overflow. By simulating the unit in MATLAB I found, that for 8 bit input samples and $N = 200$ is $w = 13$ enough – when applying the resonant frequency to the input of the filter, the output magnitude never exceeded $2^{12} - 1$ during the computation of the 200 samples. The throughput of this solution can be found by counting the number of cycles needed to complete computation of one output sample. This is 7 cycles in this case. Using internal clock of 32 MHz, it would be possible to use sampling frequencies of up to $32.106/7 = 4.6$ MHz. Using more expensive, quicker chips, it would be possible to achieve at least 2 times better results. The chip area occupied is summarized in following table:

3.2 "Four bits at once" version

The second solution reduces the distributed arithmetics to standard serial-parallel multiplier using LUT, as can be seen on Fig. 6.

Table 1. Nr. of CLBs–"Two bits at once"

Block	Nr. of CLBs
PSC	4
SA	2
LUT 16x15	8
INV15	8
FA	10
z^{-1}	2
SUM	34

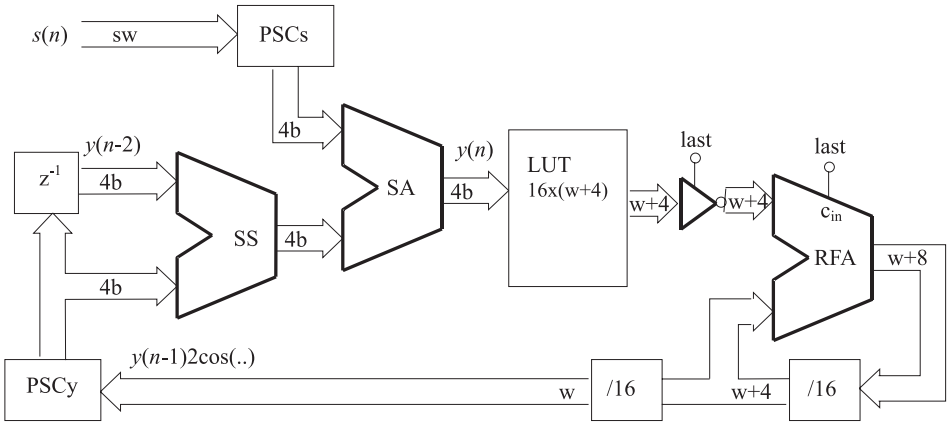

Fig. 6. "Four bits at once" version

The function of this unit is similar to the one on 5, except the SS (Serial Subtracter) block, that performs the subtraction of the terms $y(n-1)2\cos(\ldots)$ and $y(n-2)$. The number of cycles needed for processing one 8 bit input sample is 4. This is almost 2 times less than in the previous version. As for the number of occupied CLBs, it is summarized in the table:

Table 2. Nr. of CLBs–"Four bits at once"

Block	Nr. of CLBs
PSC	4
SA	4
LUT 16x15	4
INV15	9
FA	12
z^{-1}	4
SUM	46

This means, it would be possible to place 12 of the Goertzel IIR filters to the 4013 chip, and that is still quite enough when considering the throughput is almost 2 times better than with the previous version.

4 The DTMF decoder application

The application of multichannel DTMF decoder, that used the Goertzel algorithm, was built as single board with DSP320C32-60 and Xilinx XCS30-4 (576 CLBs) with external SRAM memory 128kx8, that served as input sample queue (samples from PCM are coming interleaved – channel after channel, but we need to sort them to channel queues to get sequences of N samples for each channel). Inside the Xilinx, there were 8 Goertzel IIR filters ("Two bits at once version") for each of the eight DTMF frequencies. There was also logic for interfacing serial PCM codecs and queueing the PCM samples to get normal input sequences, and interface to the DSP processor. The Xilinx was clocked by the DSP external bus clock (30 MHz), so the design had to be heavily optimized using the Foundation 1.5 Floorplanner, which allows effective manual placing of the logic.

Every 200 samples, DSP processor takes the results from all the filters and evaluates the magnitudes of the eight DTMF frequencies. If it finds a proper DTMF frequency combination, it decodes it to one of the values (1..16) and sends it to a host PC computer.

With the "two bits at once" version in this configuration, it is possible to decode as much as $30.10^6/(7.8000) = 535$ channels (Xilinx internal clock=30 MHz, nr. of clocks per sample=7 and sampling frequency is 8 kHz), so the Xilinx would have to include 16 PCM32 (32 channel PCM) interfaces. Because this would take about 60 CLBs, it was impossible to fit it in the XCS30. Therefore the number of channels was limited to 256. To use the real power of the FPGA Goertzel core, we would have to use bigger chip or codecs that support the PCM with 64 channels per frame.

5 Conclusion

The FPGA technology can bring an unexpected increase of throughput, if used in a DSP system. E.g., a multichannel Dual Tone Multi Frequency (DTMF) decoder, using FPGA configured as a Goertzel detector (with the "Four bits at once" version) could handle 1000 telephone channels. In comparison with a classical DSP processor TMS320C32-60, that can handle only 32 channels, this is 32-times increase of throughput when the price has grown only 1.5 times (for deeper analysis see [1]). Therefore, it is obvious that the FPGA chips can bring a revolution to certain areas of DSP applications.

References

1. Dulik, T.: DSP unit using FPGAs. Diploma project, TU Brno (1998)
2. Mintzer, L.: Mechanization of DSPs. Handbook of DSP (1987), 941–973
3. Texas Instruments: Modified goertzel algorithm for DTMF using teh TMS320C80. Applicatio Report SPRA066 (1996)
4. Goslin, G. R.: A guide to using FPGAs for application-specific DSP performance. Xilinx application notes (1995)