Scalable Parallel Saturating Multioperand Adders

Article

4 authors, including:

M. Jimenez
University of Puerto Rico System
87 PUBLICATIONS **289** CITATIONS

Scalable Parallel Saturating Multioperand Adders

Cristian Medina Abkarian
Elianne Bravo
Advisor: Dr. Manuel Jimenez

Electrical and Computer Engineering Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
cmedina@sisgraf.com, elianne_bravo@hotmail.com

Abstract

Many *multimedia* applications, including *Digital Signal Processing* (DSP), use *parallel* saturating multioperand adder (PSMA) structures in their designs. Some are employed in conjunction with saturating multipliers and multiply-accumulate units. Normally this type of adder calls for a typical serial implementation, but our goal is to develop and implement (using VHDL) a parallel model that would provide for optimizations and scalability. Currently we have implemented an optimized VHDL model (based on similar research) and programmed it to accept *n bits*. We plan to have an *m-input* implementation by the end of the semester that is even further optimized than the one presented here.

1. Introduction

Saturating adders are built upon any kind of simple adder, but instead of always producing the sum of the two operands, they produce their saturated sum. Saturation is achieved when there is an "*overflow condition*" (when we run out of bits to represent the outcome of a *signed* operation), this sets the result of the procedure to a predetermined "*saturation value*". For our purposes, this value is the highest possible (*positive or negative*) number that can be represented in n bits without reaching the overflow condition.

The interesting point is that when performing a saturating addition, the operation looses its *commutative* and *associative* properties, trying to constrain us to executing the addition in a serial matter. We are looking to prove that this can be done in an optimized and scalable, parallel configuration of structures that can produce the same result in less time (i.e.: it has a pattern that enables us to construct it for any number of inputs).

According to [Glossner00], GSM (Global System for Mobile) *communication speech coders* perform millions of saturating arithmetic operations per second. Internally they must perform saturation after every arithmetic operation. There have been various optimizations developed to enhance the performance of speech coders, including software approaches for the improvement of compilers that minimize lengthy loops. But hardware support is something that can dramatically advance this process.

In [Yadav99] the focus is on multiply-and-accumulate units and saturating multipliers. Their implementation uses pipeline registers to accumulate the results. They use a single fast Carry Propagate Adder (CPA) unit in the critical path (which can be a Carry Look-ahead Adder) and special logic tells them when saturation develops. In using a pipeline [Balzola01], some type of feedback is needed: it can be presented in two's complement form (WTCF) or in carry-save format (which has a separate sum and carry vector, also known as WCSF). We gain some ground when using WCSF because you remove the CPA unit out of the critical path of every single cycle and insert it at the end. The drawbacks are: increased number in gates, fan-ins and fan-outs, among others.

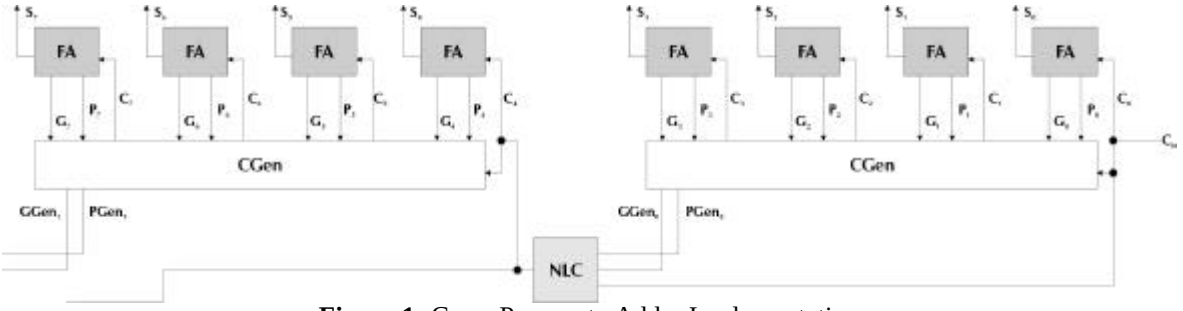

Figure 1: Carry Propagate Adder Implementation

We shall now present a brief overview of each of the components that were implemented in VHDL, their relevance in the system, and their relationships to other components.

2. Components

The letters used to symbolize the signals discussed below, have the following meanings: those that are in lowercase represent a 1-bit number; those preceded by an "s" represent the sign bit of the number (the most significant bit, since we are using a two's complement representation); and those in uppercase denote words.

2.1 Overflow detection logic (ODL)

Saturating adders rely on special logic that determines if there's an overflow condition that would saturate the result. Overflow occurs when two operands (say A and B) have the same sign and their sum (S) has the opposite sign [Glossner00]:

$$o = sa \cdot sb \cdot \overline{ss} + \overline{sa} \cdot \overline{sb} \cdot ss \qquad \text{(E-1)}$$

2.2 Saturation value generator (VGen)

When doing a saturated sum, we need to know if the result is saturated for a positive number or for a negative number, and generate the maximum allowed value for the appropriate side. For this purpose we use a simple equation based on the sign of one of the inputs:

$$V_A = sa \cdot \overline{sa} \cdot \overline{sa} \cdots \overline{sa} \cdot \overline{sa} \qquad \text{(E-2)}$$

This makes our saturation value for operand A (V_A) equal to 100...000 for negative saturation, and 0111...111 for positive saturation.

2.3 Adders

We need a set of adders that can perform the basic functions of a Full Adder (FA) or Half Adder (HA) plus a little more:

2.3.1 Carry propagate adder (CPA)

The CPA (Fig. 1) being implemented consists of a Carry Look-ahead Generator (CGen) that follows the schema of [Parhami00] (Fig. 2). It receives propagate (p) and generate (g) signals from individual adders that also function as normal FA's, and

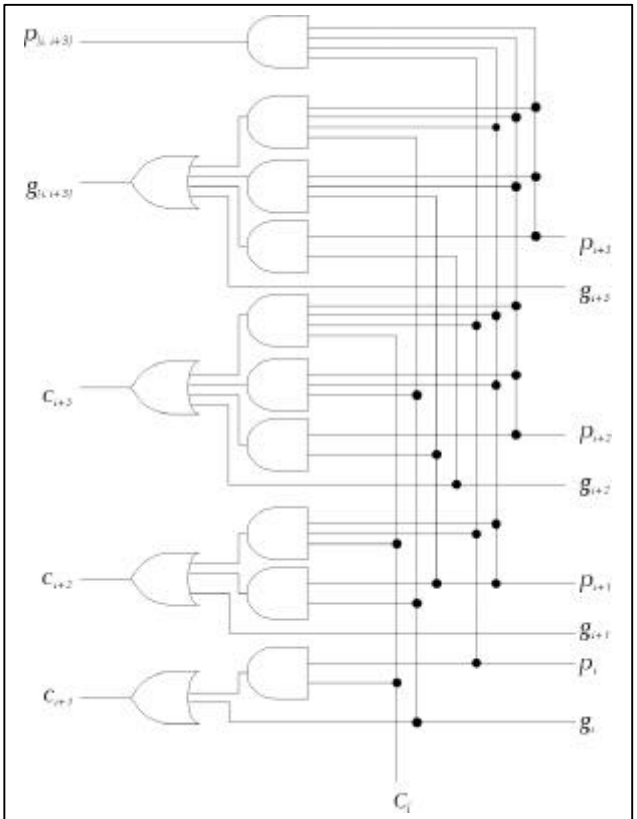

Figure 2: Structure of the CGen

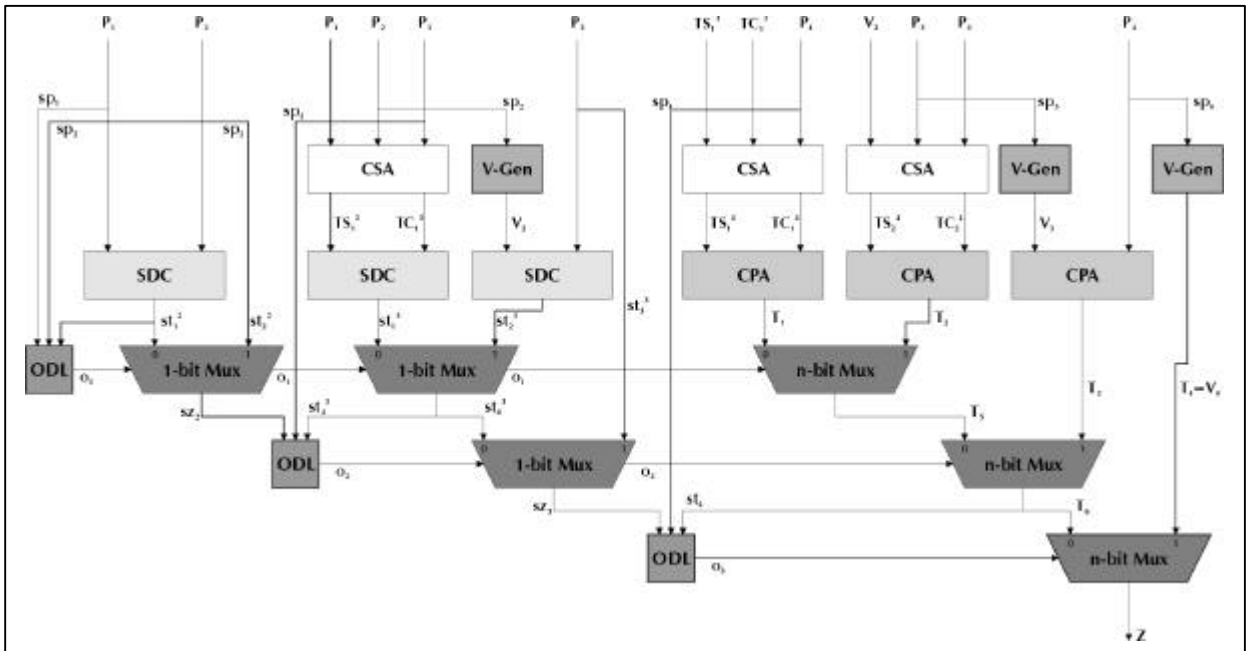

Figure 3: The Optimized SMA, where each shade of gray represents a different component as coded in VHDL.

computes the carry-in for each of those adders, plus propagate and generate signals that enable us to connect it to successive stages. The p and g signals are defined by:

$$p = a \oplus b \qquad g = a \cdot b \qquad \text{(E-3)}$$

To make it more scalable we decided to add a Next Level Carry generator (NLC), which is based on the idea of the Carry Skip adder. The equation for the NLC is:

$$c_n = g_{n-1} + (p_{n-1} \cdot c_{n-1}) \qquad \text{(E-4)}$$

A carry is generated in (E-4) if the CGen before it produces a carry (g_{n-1}) or propagates a carry-in produced by the NLC before this one ($p_{n-1}c_{n-1}$).

2.3.2 Carry save adder (CSA)

The carry save adder [Parhami00] is composed of a linear array of FA's placed in ripple carry format but not connected to each other; the carry-outs generated by each FA are given in a separate vector from the sum.

2.4 Sign detection circuit (SDC)

To reduce the amount of hardware [Glossner00], we try to share result terms between the different stages of the design, and in some cases we only need the sign of the resulting sum and not the complete sum. For this purpose we use a circuit very similar to the CPA only it outputs the 1-bit sign of the result.

3. Putting it All Together

The full Optimized Parallel Saturating Multioperand Adder (PSMA) can be seen in Fig. 3. For convenience we have used the same naming convention as in [Glossner00]. The operands are symbolized by P's, the saturated values by V's, the temporary sums by T's and the final sum by a Z.

4. VHDL Implementation and Simulation

Implementation of a 4-input, 8-bit PSMA was carried out using VHDL and Cadence Simulation tools. The code follows the optimized architecture described in [Glossner00]. The CPA, as explained earlier, was implemented using a carry-generate adder for every four bits, connected together with a carry-skip adder in order to reduce the hardware requirements. The code was simulated for several test cases and, after fixing certain coding errors, was found to provide accurate results.

We are currently working on VHDL code for a scalable version of this PSMA. Since we want an m-input, n-bit PSMA, to achieve this, the n-bit

structure is being developed first. The main difficulty was elaborating the n-bit CPA model, which is accomplished by creating an array for the CGens and the NLC structures using "generate" statements. The n-bit CSA has been fully implemented and is in the testing phase. After this has been completed, work will begin on an m-input PSMA, which is still in the research phase, but the concept is to find a pattern in the hardware to be able to generalize or abstract it. Certain patterns have already been found but these ideas need to be further developed before implementing them.

5. Conclusions & Future Work

The structures that have been presented in this paper let us create a Parallel SMA that can merge the several cycles of a serial implementation into one, while still having only one CPA in the critical delay path. Several optimizations in the amount of hardware and performance can still be made, but this hardware gives us the assistance needed to improve upon speech coders and other applications that need a logarithmic execution time for PSMA's in their cycle.

References

[Glossner00] Glossner, J., Schulte, M., Balzola, P., Ruan, J., "Parallel Saturating Multioperand Adders", in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 172-179, (San Jose, CA, Nov. 2000).

[Yadav99] Yadav, N., Schulte, M. J., and Glossner, J., "Parallel Saturating Fractional Arithmetic Units", in *Ninth Great Lakes Symposium on VLSI*, pp. 214-217, (Mar. 1999).

[Schulte00] Schulte, M. J., Balzola, P. I., Akkas, A., and Brocato R.W., "Integer Multiplication with Overflow Detection or Saturation", in *IEEE Transactions on Computers, Vol. 9, No. 7*, pp. 681-691, (July 2000).

[Balzola01] Balzola, P. I., Schulte, M. J., Ruan, J., Glossner, J., and Hokenek, E., "Design Alternatives for Parallel Saturating Multioperand Adders", in *International Conference on Computer Design*, pp. 172-177, (2001).

[Parhami00] Parhami, B., "Computer Arithmetic: Algorithms and Hardware Designs", Oxford University Press, (New York, 2000).