Ternary Adder IP Cores

Martin Kumm, Jens Willkomm

October 17, 2017

1 Introduction

This IP core provides resource efficient ternary adders, i. e., adders with three inputs performing $s = x+y+z$, for the Altera and Xilinx platforms. Resource efficient means that they need exactly the same resources on modern FPGAs as two-input adders, but are slightly slower. The Xilinx core (`ternary_adder_xilinx.vhd`) is a low-level implementation, following an US patent from Xilinx [**?**]. It directly uses the Xilinx primitives (`CARRY4`, `LUT6_2` and `FDCE`). It is suitable for all FPGAs providing 6-input LUTs. Today, these are the Virtex 5-7, Spartan 6, Kintex 7 and Artix 7 families. The Altera core (`ternary_adder_altera.vhd`) is a high-level implementation using the '+' operator. However, the ternary subtract operations ($x - y + z$, $x + y - z$ and $x-y-z$) are not supported by a high-level description; so this is realized by extending the word size of the ternary adders and setting the lower bits to appropriate constant values. They can be mapped very resource efficient for all Altera FPGAs providing adaptive logic modules (ALMs), today, these are the Arria I,II,V and Stratix II-V FPGAs.

2 Interface

The generics as well as the port are identical for the Altera and Xilinx implementation and are described in Table **??** and Table **??**, respectively.

3 Implementation

Both implementations uses a carry save adder (CSA) tree with three inputs and a final ripple carry adder as vector merging adder (VMA). One stage of full adders (FAs) is used to realize a 3:2 compressor, i. e., the three input bit vectors are compressed to two bit vectors which are obtained by the sum and carry outputs. A second stage of FAs merges these two bit vectors to a single result.

For Altera, the 3:2 compressor can be directly mapped to the ALM LUT, realizing the sum $s'_i = x_i \oplus y_i \oplus z_i$ and carry $c'_i = x_i y_i + x_i z_i + y_i z_i$. The full adders of the ALM are used for the VMA. To include both stages in a single ALM stage, each ALM has to be configured to the shared arithmetic mode [**?**] in which the output of one LUT is connected to the FA input of the next higher bit. The resulting ternary adder structure is shown in Figure **??**.

Table 1: Description of the generics

Generic	Type	Default	Description
`input_word_size`	integer	10	Input word size of the inputs x,y and z. The output word size is automatically set to `input_word_size+2`
`subtract_y`	boolean	false	Input y is negated, realizing $s = x - y \pm z$
`subtract_z`	boolean	false	Input z is negated, realizing $s = x \pm y - z$
`use_output_ff`	boolean	true	If true, the adder uses flip flops at the output (without extra slice or ALM resources)

Table 2: Description of the port

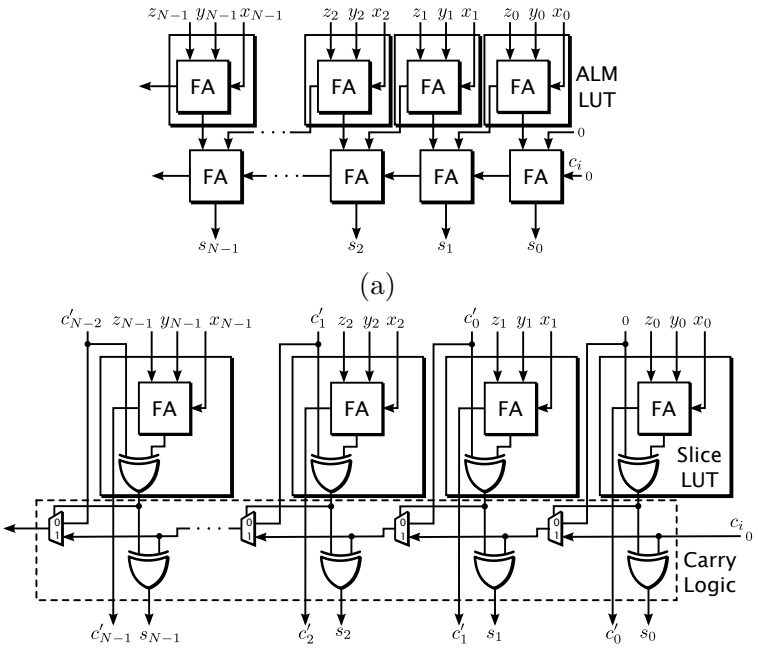
Generic	Direction	Type	Word Size	Description
`clk_i`	in	`sl`	1	Clock input (used when `use_output_ff=true`)
`rst_i`	in	`sl`	1	Reset input (used when `use_output_ff=true`)
`x_i`	in	`slv`	`input_word_size`	Input x
`y_i`	in	`slv`	`input_word_size`	Input y
`z_i`	in	`slv`	`input_word_size`	Input z
`sum_o`	out	`slv`	`input_word_size + 2`	Sum output s

For Xilinx, the FA for the 3:2 compressor is also realized in the FPGA LUT [**?**]. In addition to that, one additional XOR gate has to be realized in the same LUT to complete the fast carry chain resources to a ripple carry adder for the VMA. The carry output of the first FA (realized in the LUT) must be routed to the next higher FA input using the FPGA routing fabric. The resulting slice configuration is shown in Figure **??**.

4 Resource Consumption

For Altera, each ALM can compute two output bits. As the output word size is two bits more than the input word size, there are

$$N_{\mathrm{ALM},++} = \left\lceil \frac{\mathrm{input\_word\_size} + 2}{2} \right\rceil \tag{1}$$



(a)

(b)

Figure 1: Realization of ternary adders on (a) Altera Stratix II-V ALMs (b) Xilinx Virtex 5-7 Slices

ALMs needed for a pure addition ($s = x + y + z$). If one input is subtracted (setting one of `subtract_y` or `subtract_y` to true), the word length has to be extended by one bit leading to:

$$N_{\text{ALM},+-} = \left\lceil \frac{\text{input\_word\_size} + 3}{2} \right\rceil \tag{2}$$

Finally, if two inputs are subtracted, the word length has to be further increased leading to:

$$N_{\text{ALM},--} = \left\lceil \frac{\text{input\_word\_size} + 4}{2} \right\rceil \tag{3}$$

For Xilinx, four output bits can be computed in each slice. Thus, the number of slices is given by:

$$N_{\text{Slices}} = \left\lceil \frac{\text{input\_word\_size} + 2}{4} \right\rceil \tag{4}$$

The slice usage is independent of the operation performed. If a slice is not fully utilized, the remaining LUTs can still be used for other functionalities.

5 Performance

To estimate the performance, the maximum clock frequencies ($f_{\max}$) were obtained by synthesis experiments for Altera Stratix IV (EP4SGX230KF40C2) using Quartus-II 10.1 and Xilinx Virtex 6 (XC6VLX75T-2FF484) using ISE 13.4, both after place & route. The resulting clock frequencies with output word sizes from 16 up to 64 bit are shown in Table **??**.

Table 3: Performance of the IP Cores

output word size	$f_{\max}$ Stratix IV [MHz]	$f_{\max}$ Virtex 6 [MHz]
16	708	450
32	565	379
48	479	312
64	423	292

6 Simulation & Test

The simulation and automated tests were performed using Modelsim. For that, a testbench (`tb_ternary_adder.vhd`) was created which uses a random number generator together with assert statements to verify the designs. To automate the different FPGA targets the do-file `batch_sim.do` was created which compiles the designs and applies the tests for each target as specified in the do-file `sim_single_inst.do`. These tests include different word sizes, subtractions and the output flip flop functionality. All tests can be started from command line using `vsim -c -do 'do batch_sim.do'` (as defined in `modelsim_batch_sim.sh`).