TU Delft

ET4171 Processor Design Project

LEON3 processor optimization

Luca Feltrin, Henrique Dantas

13

Introduction

For the Processor Design Project course we have been asked to improve the performance of the LEON3, a 32-bit SPARC V8 processor designed for embedded application.

Our main target is to decrease the computation time for certain benchmarks keeping the power consumption as low as possible, so the main compound metric we are going to care of is the power*benchmark score (P*BS).

The SPARC V8 architecture contemplates the use of instruction and special hardware for integer multiplications and divisions, but with the original configuration the multiplier takes 5 clock cycles to calculate the result and the divider 36, so one of the first things we decided to do is to improve this arithmetic cores, some easy algorithms can be implemented to have a real improvement.

Some other changes are described at the end of this document.

Improved Arithmetic Cores

Multiplier

Divider

The algorithm implemented in the original version of the processor is one of the simplest but the slowest available.
Several other algorithms can compute the division faster but all of them present disadvantages that must be taken into account according to the target application.

Algorithms like repeated multiplication or reciprocation are fast but require a significant amount of area, similarly an array divider would have been very fast only If we had control on the place&routing process in order to create a regular. In the end we decided to implement a simple radix-4 division algorithm for simplicity of implementation and of the circuit itself.
Using an higher radix could have improved performance but the size of the look-up table required by the algorithm would have increased again the area consumption.

The divider consist in a state machine (its diagram is shown in Fig. 1) which check if the inputs will generate an overflow and performs a preliminary shift to put the divisor in the appropriate range to be computed correctly.

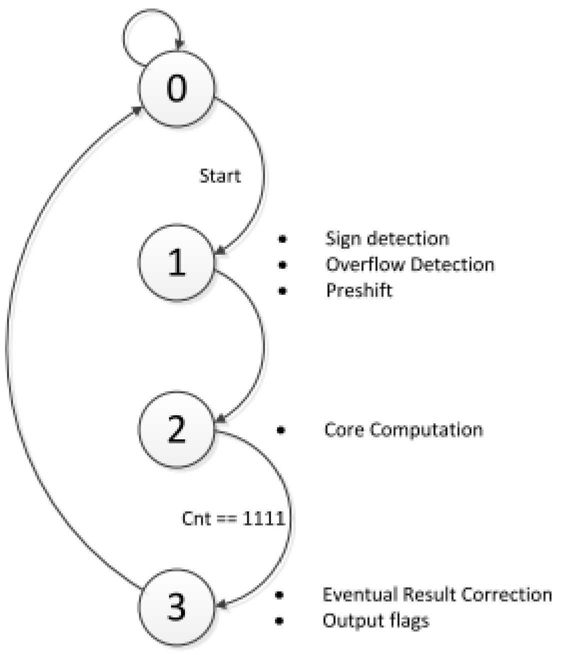

FIG. 1 DIVIDER STATE DIAGRAM

After that, the real computation begins and lasts 16 clock cycles. The block diagram of the divider while it's in this state is shown in Fig. 2.

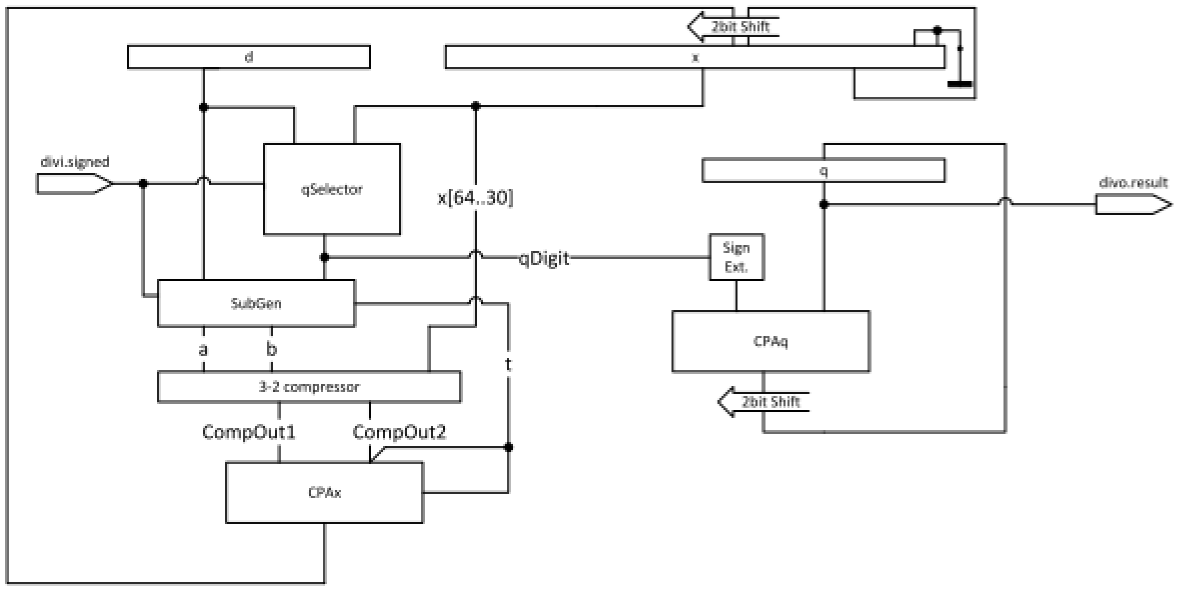

FIG. 2 DIVIDER BLOCK DIAGRAM (DURING CORE COMPUTATION, STATE=1)

The algorithm is very similar to the original radix-2 version but in this case the partial reminder (x) is shifted by 2 bits every cycle and the circuit has to guess the quotient digit from the range [-3,3]. "qSelector" is the look-up table which perform the quotient digit guessing and it's based on the p-d plot of the radix-4 SRT division shown in Fig. 3, in case of unsigned division only the right half of the p-d plot is being used.

The quotient digits are in a radix-4 redundant format so a conversion in binary format is needed, the conversion is performed gradually every cycle by the 32-bit adder "CPAq" which shift and sum each generated digit with the already calculated quotient.

One could think that it would be better keep the quotient in a radix-4 redundant format and avoid the addition in order not to slow down the execution every cycle so that the clock frequency could be higher, but also the original divider executes a 32-bit addition every cycle so from this point of view our divider is not worse than the original one, moreover a conversion from radix-4 redundant format to binary is quite complicated, doing this it consists in a simple addition.

The same concept has been used also for the computation of the partial reminder.

An addition/subtraction in Carry-Save format would have been much faster and also easier, but the selection of the quotient digit would have required the analysis of the most significant bits of both the sum and the carry making the lookup table several orders of magnitude bigger.
In our divider "SubGen" generates the multiple of d to sum with the current partial reminder in a carry save format, all this operands are been compressed by a 3-2 compressor (1 full adder of delay) and finally the new partial reminder is calculated with a 35-bit adder, "CPAx".

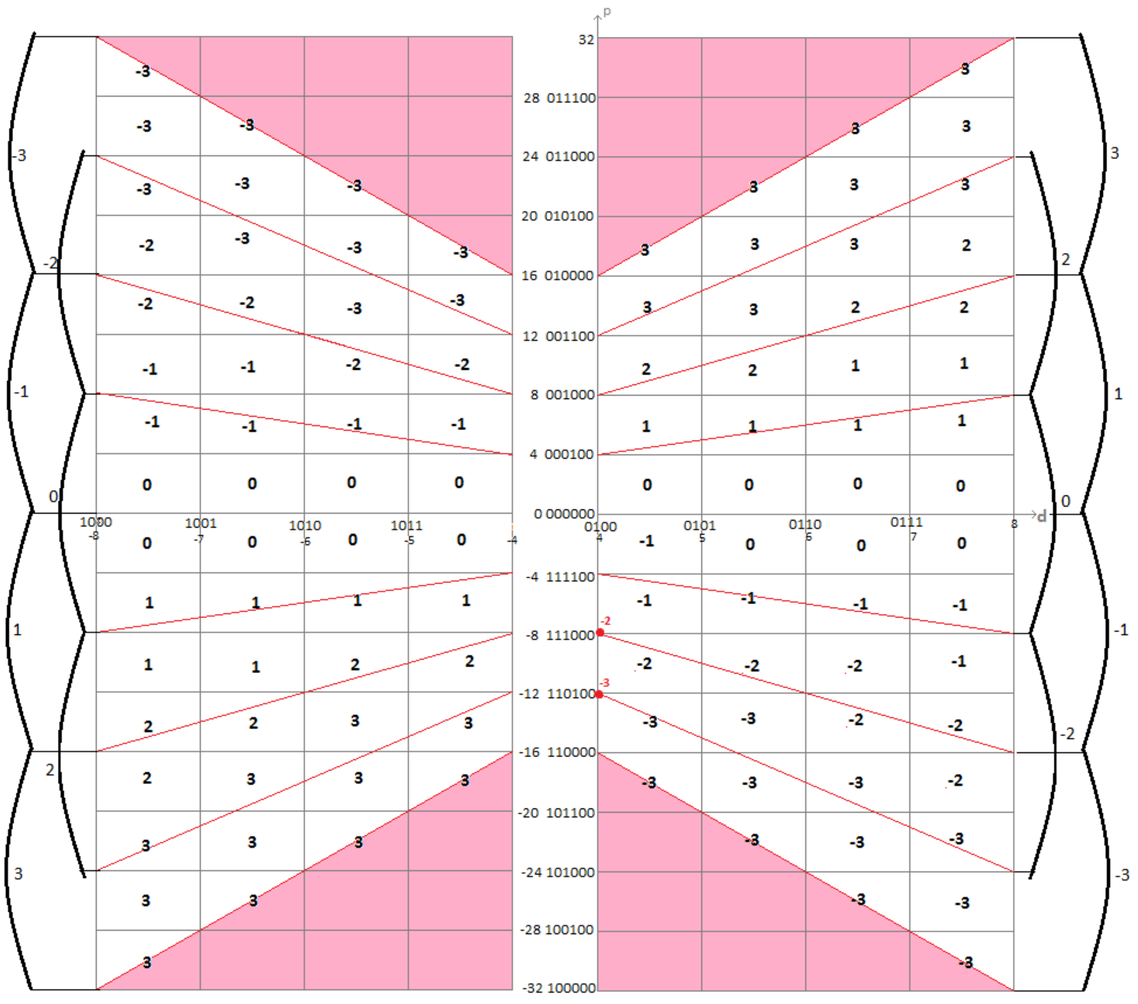

FIG. 3 RADIX-4 P-D PLOT (RED DOTS ARE EXCEPTIONS)

Other solutions have been analyzed, such as having a look-up table only for unsigned division, half of the size of the final version, and handle the sign separately but the synthesis has shown that the resources utilization would have not changed a lot while one more cycle would have been needed so we decided to keep using this divider.

In the end in order to check the compatibility of the radix-4 divider with the original one we performed a simulation of the two versions with the same output, the only difference is in the case of overflows where the flag is set up correctly but the result is different or in some cases undefined, but since there is an overflow the result has no meaning so this is acceptable.

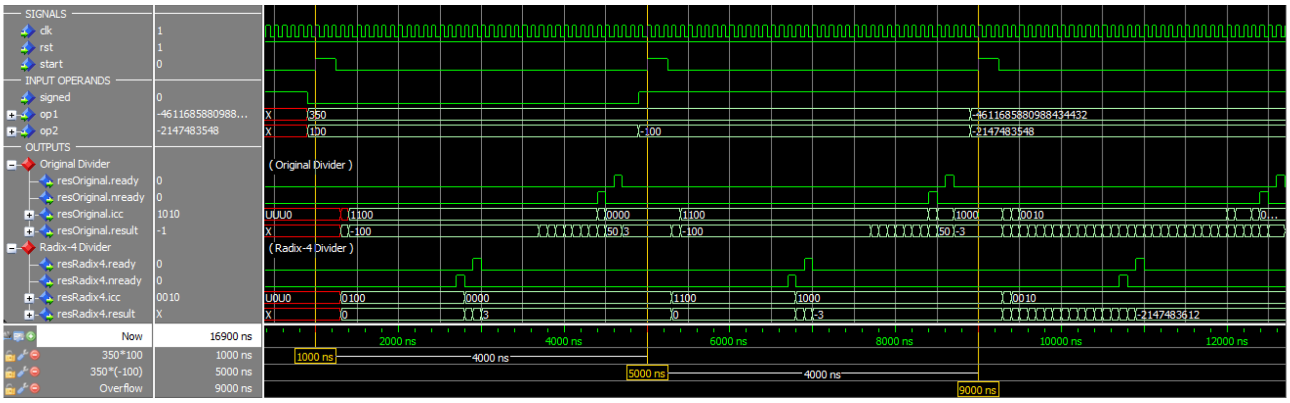

FIG. 4 SIGNAL DUMP OF RADIX-4 DIVIDER VS. ORIGINAL DIVIDER

Results

Synthesis

In order to evaluate the performance of our improved processor we need first to evaluate the performance of the baseline version of the processor.

The synthesis tool reported the values in the Table 1 for the resources utilization.

Timing Summary (max clock freq. [MHz])	# of Occupied Slices	Total # of 4-input LUTs	Quiescent power [W]	Dynamic power [W]	Total power [W]	P/f [W/MHz]
80.522	9904	16889	2,467	0,721	3,188	0,03959

TABLE 1: RESOURCE UTILIZATION BASELINE

Notice that the value "P/f" indicate the energetic efficiency of the processor, since the power consumption is almost proportional to the clock frequency we can use this value to estimate the power consumption at different clock frequencies.

From the synthesis report we can also see the slowest path which determines the max clock frequency.

This path is from "ddrsp0.ddrc0/ddr_phy0/ddr_phy0/xc4v.ddr_phy0/ddgen[24].gi/FF1" to "ddrsp0.ddrc0/ddr32.ddrc/read_buff/xc2v.x0/a0.x0/Mram_rfd1"

?????????????????????????????????? ANY DETAILED INFORMATION??!?!!

Benchmark Scores

Metrics comparison

In order to get a fair comparison between the baseline and the improved processor some standard metrics have to be calculated and studied.

Usually these metrics are **A**, the area consumption here calculated as the weighted sum of the number of occupied slices and the number of 4-input LUTs used where the weight is the reciprocal of the number of available resources, **D** is the delay or the reciprocal of the max clock frequency and indicate the delay of the slowest path, **P** is the power and it's simply calculated as the total power consumed by the Dhrystone benchmark used for the simulation and **BS** is the benchmark score which indicate how fast a program can be executed, it's calculated as the total execution time of the benchmarks on the FPGA board. (DO YOU AGREE WITH THIS?????)

Moreover some composite metrics can be observed: these metrics consider two or more primitive metrics and often are more interesting than these ones because when a modification is done on the processor usually we obtain two opposite effects, one primitive metrics increases while another one decreases, but what we want is that all in all we have an improvement. Composite metrics show the overall performance.

Since we want to speed-up the execution of the software while keeping the power consumption low because this is a processor designed for embedded applications, the composite metric we are interested the most is **P*BS** which show how much the processor is able to execute the software fast with the same amount of energy.

Other composite metrics are A*D, A*BS and P*D. Since we focused on the improvement of the execution time and power consumption one can notice that these metrics are worse in our version compared to the baseline, because the area consumption has increased notably and the delay hadn't had any significant changes.

Both the baseline version's and our improved version's synthesis and benchmark results have be condensed in this metrics and their values are in the Table 2.

Version	**Primitive metrics**				**Composite metrics**			
	A	D	P	BS	A*D	A*BS	P*D	P*BS
Baseline	3,63E-01	1,24E-02	3,19E+00	2,19E+02	4,51E-03	7,96E+01	3,96E-02	6,99E+02
Modified (ONLY DIVISION!!!!)	3,84E-01	1,24E-02	3,21E+00	2,13E+02	4,77E-03	8,19E+01	3,99E-02	6,85E+02

TABLE 2: FINAL METRICS FOR BASELINE AND IMPROVED VERSIONS

Conclusions and further improvements

(FURTHER IMPROVEMENTS)

Because of lack of time we didn't do any other changes, but of course there are many things to change in the architecture to improve further the performance.

The size and structure of the cache memory can be changed to decrease the probability of misses and so the benchmarks execution time, but this can be done using the configuration tool and so it would have not been an our real achievement, moreover increasing the cache size probably would have increased also the power consumption making things worse.

The LEON3 uses a static branch prediction in the integer unit, which is a good compromise between power consumption, because no difficult computation is needed, and gain in terms of execution time. To improve performance further a 1 bit or a 2 bit branch prediction buffer algorithm. The calculation needed is more complicated and it has to be done very often (30% of the instructions are branches) so the power consumption probably would increase, but the gain in terms of execution time could be worth it.

Attached documents

Baseline

Dhrystone report

```
# Xilinx ML410 Development board
# GRLIB Version 1.1.0, build 4104
# Target technology: virtex4    ,  memory library: virtex4
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area at 0xfff00000, 1 Mbyte
# ahbctrl: AHB masters: 4, AHB slaves: 8
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Gaisler Research        Leon3 SPARC V8 Processor
# ahbctrl: mst1: Gaisler Research        AHB Debug UART
# ahbctrl: mst2: Gaisler Research        JTAG Debug Link
# ahbctrl: mst3: Gaisler Research        GR Ethernet MAC
# ahbctrl: slv0: Gaisler Research        Single-port DDR266 controller
# ahbctrl:       memory at 0x40000000, size 256 Mbyte, cacheable, prefetch
# ahbctrl:       I/O port at 0xfff00100, size 256 byte
# ahbctrl: slv1: Gaisler Research        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 1 Mbyte
# ahbctrl: slv2: Gaisler Research        Leon3 Debug Support Unit
# ahbctrl:       memory at 0x90000000, size 256 Mbyte
# ahbctrl: slv3: European Space Agency   Leon2 Memory Controller
# ahbctrl:       memory at 0x00000000, size 512 Mbyte, cacheable, prefetch
# ahbctrl:       memory at 0x20000000, size 512 Mbyte
# ahbctrl:       memory at 0xc0000000, size 16 Mbyte, cacheable, prefetch
# ahbctrl: slv5: Gaisler Research        System ACE I/F Controller
# ahbctrl:       I/O port at 0xfff00000, size 256 byte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv0: European Space Agency   Leon2 Memory Controller
# apbctrl:       I/O ports at 0x80000000, size 256 byte
# apbctrl: slv1: Gaisler Research        Generic UART
# apbctrl:       I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Gaisler Research        Multi-processor Interrupt Ctrl.
# apbctrl:       I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Gaisler Research        Modular Timer Unit
# apbctrl:       I/O ports at 0x80000300, size 256 byte
# apbctrl: slv7: Gaisler Research        AHB Debug UART
# apbctrl:       I/O ports at 0x80000700, size 256 byte
# apbctrl: slv8: Gaisler Research        General Purpose I/O port
# apbctrl:       I/O ports at 0x80000800, size 256 byte
# apbctrl: slv11: Gaisler Research        GR Ethernet MAC
# apbctrl:       I/O ports at 0x80000b00, size 256 byte
# apbctrl: slv15: Gaisler Research        AHB Status Register
# apbctrl:       I/O ports at 0x80000f00, size 256 byte
# ahbstat15: AHB status unit rev 0, irq 7
# grgpio8: 14-bit GPIO Unit rev 1
# gptimer3: GR Timer Unit rev 0, 8-bit scaler, 2 32-bit timers, irq 8
# irqmp: Multi-processor Interrupt Controller rev 3, #cpu 1, eirq 0
# apbuart1: Generic UART rev 1, fifo 4, irq 2
# gracectrl5: System ACE I/F Controller, rev 0, irq 13
# greth3: 10/100 Mbit Ethernet MAC rev 03, EDCL 1, buffer 2 kbyte 128 txfifo, irq 4
# ddrsp0: 32-bit DDR266 controller rev 0, 64 Mbyte, 100 MHz DDR clock
# ahbjtag AHB Debug JTAG rev 1
# ahbuart7: AHB Debug UART rev 0
# dsu3_2: LEON3 Debug support unit + AHB Trace Buffer, 2 kbytes
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 2*8 kbyte, dcache 2*4 kbyte
# clkgen_virtex2: virtex-2 sdram/pci clock generator, version 1
# clkgen_virtex2: Frequency 100000 KHz, DCM divisor 16/20
# Execution starts, 1 runs through Dhrystone
#
# Execution ends
#
#
#
# Final values of the variables used in the benchmark:
#
#
#
# Int_Glob:            5
#
#        should be:   5
#
# Bool_Glob:           1
#
#        should be:   1
#
```

```
# Ch_1_Glob:          A
#
#         should be:  A
#
# Ch_2_Glob:          B
#
#         should be:  B
#
# Arr_1_Glob[8]:      7
#
#         should be:  7
#
# Arr_2_Glob[8][7]:   11
#
#         should be:  Number_Of_Runs + 10
#
# Ptr_Glob->
#
#   Ptr_Comp:         1073811648
#
#         should be:  (implementation-dependent)
#
#   Discr:            0
#
#         should be:  0
#
#   Enum_Comp:        2
#
#         should be:  2
#
#   Int_Comp:         17
#
#         should be:  17
#
#   Str_Comp:         DHRYSTONE PROGRAM, SOME STRING
#
#         should be:  DHRYSTONE PROGRAM, SOME STRING
#
# Next_Ptr_Glob->
#
#   Ptr_Comp:         1073811648
#
#         should be:  (implementation-dependent), same as above
#
```

```
#   Discr:             0
#
#         should be:   0
#
#   Enum_Comp:         1
#
#         should be:   1
#
#   Int_Comp:          18
#
#         should be:   18
#
#   Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
#
#         should be:   DHRYSTONE PROGRAM, SOME STRING
#
# Int_1_Loc:           5
#
#         should be:   5
#
# Int_2_Loc:           13
#
#         should be:   13
#
# Int_3_Loc:           7
#
#         should be:   7
#
# Enum_Loc:            1
#
#         should be:   1
#
# Str_1_Loc:           DHRYSTONE PROGRAM, 1'ST STRING
#
#         should be:   DHRYSTONE PROGRAM, 1'ST STRING
#
# Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
#
#         should be:   DHRYSTONE PROGRAM, 2'ND STRING
#
#
#
# Begin time is:           399496
#
```

```
# End time is:             399496
#
# user time is:            0
#
# Measured time too small to obtain meaningful results
#
# Please increase number of runs
```

Upgraded version