OpenCores.Org

Hardware looping unit

Author: Nikolaos Kavvadias
nkavv@skiathos.physics.auth.gr

Rev. 0.1
April 13, 2004

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.1	12/04/04	Nikolaos Kavvadias	First Draft

Contents

1 Introduction

This document discusses the details for the design of the hardware looping unit (HWLU). The design is based on recent published work [1],[2],[3]. Its main purpose is to provide an enhancement to program control units found in modern microprocessors, by efficiently handling loop increments in nested loop structures. As mentioned in [3] this unit was originally used to handle loop nesting up to five levels, which suffices for the studied benchmarks in their work. The main advantage of the presented architecture is that successive last iterations of nested loops are performed in a single cycle. This architecture can be useful in the case that all data processing in context of the nested loop structure is performed in the inner loop, which is rather often in multidimensional signal processing applications as performance-critical code in image coding and video compression standards.

This implementation, which is called “hardware looping unit”, is actually a somewhat enhanced clone of the MediaBreeze looping unit. MediaBreeze is an SIMD, addressing and looping engine adapted to high-end (and very power consuming) microprocessors. As detailed in [1],[2] and given their proclaimed figures, the whole architecture is not suitable to power-sensitive embedded computing.

For the architecture, portable VHDL code that promotes design reuse is provided. The design has a simple synchronous interface with a single system clock. Also, the means are provided for reusing the VHDL code in applications with different maximum number of loops. For this reason, two software tools are provided in the distribution, that generate the parts of the architecture (both the priority encoder and top-level module as discussed later) that depend on the implied maximum number of loops.

NOTE: Compared to another looping unit proposed for use in embedded systems, currently named ZOLC (Zero Overhead Loop Controller) [4], there exists a tradeoff as for slightly better cycle performance, the HWLU necessitates redundant hardware. While with ZOLC, a complex loop structure with an arbitrary number and combination of loops can be controlled, by using a single process unit (one adder, one comparator etc), HWLU demands this hardware replicated for each loop. Also ZOLC supports loop structures with loop parameter values changing at run-time.

1.1. Files included in the distribution

The current distribution (version 0.1) contains the following files:

Directory/file	Description/usage
hwlu/syn/leonardo/bin	
hw_looping_unit_csadder.scr	LeonardoSpectrum synthesis script for HWLU. Utilizes an 8-bit carry-select adder.
hw_looping_unit_generic.scr	LeonardoSpectrum synthesis script for HWLU. Uses a generic adder.
hwlu/rtl/vhdl	
add_dw.vhd	DW-bit adder
cmpeq.vhd	Equality comparator
reg_dw.vhd	DW-bit register with synchronous reset and load enable
fa.vhd	1-bit full-adder cell
fa_nc.vhd	1-bit full-adder cell with no carry-out
mux2_1.vhd	2-to-1 DW-bit multiplexer
csa8.vhd	8-bit carry select adder (top-level of adder)
index_inc.vhd	Index incrementer (increments by 1)
prenc_loops5.vhd	Automatically generated priority encoder
hw_loops5_top.vhd	Automatically generated top-level module of the architecture
hwlu/bench/vhdl	
hw_loops5_top_tb.vhd	Testbench for hw_loops5_top.vhd
hwlu/doc	
hwlu_spec.pdf	This document (specification)
hwlu/sw	
gen_hw_looping.vhd	Parameterized software utility that can generate the top-level file of the HWLU architecture for a given number of supported loops
gen_priority_encoder.vhd	Parameterized software utility that can generated the priority encoder module for a given number of support loops

2 Architecture

The hardware looping architecture (HWLU) naturally can incorporate any number of levels of loop nesting in hardware to eliminate branch instruction overhead for loop increments. The user can re-generate the corresponding files for modules hw_looping(structural) and priority_encoder(rtl) for a different number of supported loops. Its operation is similar to control mechanisms found in recent DSPs. Figure 1 shows the block diagram of the hardware looping architecture.

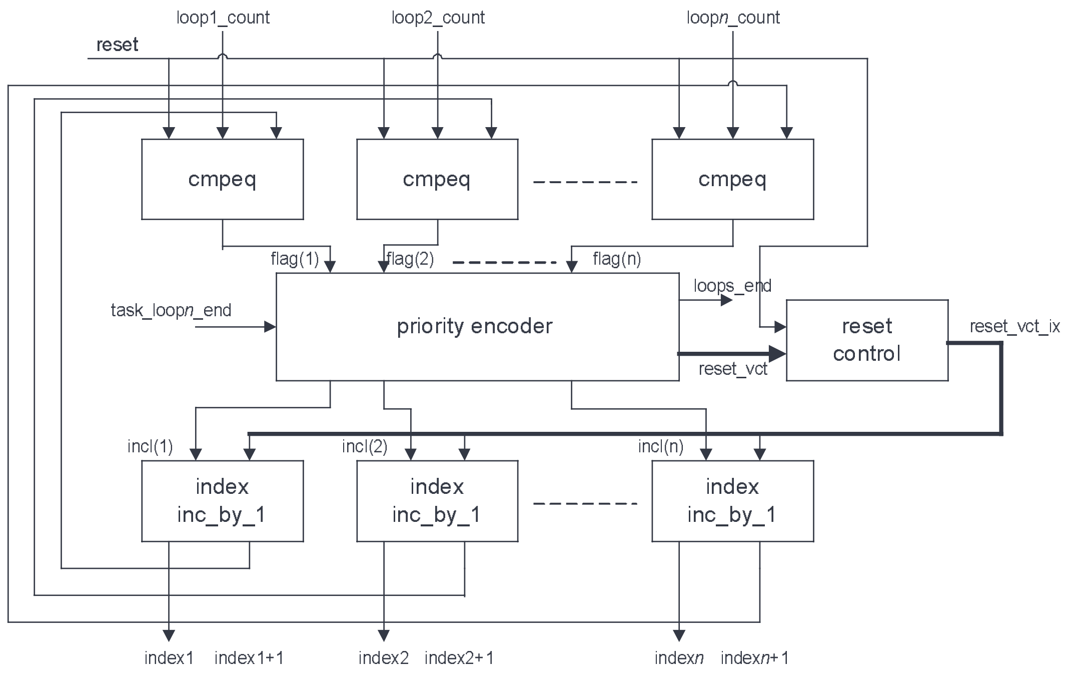


Figure 1. Block diagram of the hardware looping unit

Loop index values are produced every clock cycle based on the loop bound values (possibly read from a lookup table) for each level of nesting. The initial value for the loop indices is zero as by reset, and the maximum value is equal to the loop_bound minus one. In the following cycle of a last iteration for a specific loop, the loop index is reset to its initial value.

The priority encoder performs the actual control logic in context of the HWLU and operates asynchronously by detecting the equality comparators (cmpeq) outputs (bitwise flag signals) and an external signal from the datapath (task_loop*n*_end), where n is the enumeration of the inner loop. This signal is produced by the corresponding hardware module that performs the inner loop operations, which may be a dedicated accelerator engine.

If a specific loop is terminating, this loop as well as all its inner loops are reset in the subsequent cycle. If this loop is not the outermost one, its neighboring outer loop index is incremented. In case that none of the loops is terminating, then the inner loop is incremented. Signal task_loop*n*_end guards this increment operation.

Finally, signal loops_end designates that processing in the entire loop structure has terminated, and can be made available to the main control unit of the microprocessor.

2.1. Using the hardware looping unit within a programmable processor

Figure 2 indicates a possible design of a control unit used in a programmable instruction set processor. It is implied that the register architecture of the processor is partitioned, so that the loop index registers are stored into dedicated registers (the register bank comprised by the increment-by-1 units) and a general-purpose register file (not shown here) is used for other subroutine arguments, global variables etc.

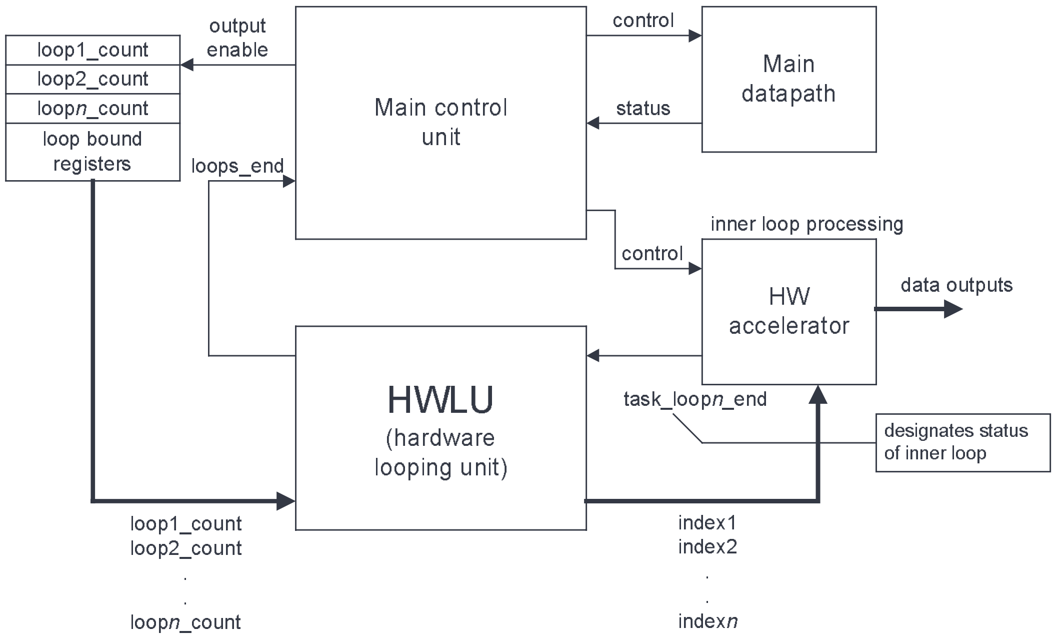


Figure 2. Usage of the hardware looping unit in a programmable processor

As can be seen, usual processing (e.g. for control-dominated segments of the user program) is implemented in the main datapath, which communicates through control and status channels with the main control unit. When appropriate, the main control unit activates the hardware acceleration datapath unit. Also, at that time, the output enable input to the loop bound register bank shown in the figure is active, so that the loop_bound value can be read by the HWLU. In our example, this unit performs all the inner loop processing. All index variables, are made available to the acceleration unit so that (high-bandwidth) data and address computation can be serviced as needed. When its operation terminates, the HWLU is acknowledged through the task_loop*n*_end asynchronous flag. On an active loops_end signal, which occurs when the loop structure is exited, the main control unit pauses the HWLU e.g. by deasserting the output enable signal to the loop bound values lookup table.

3 Operation

The operation of the core is rather simple. Input signal clk is the system clock for the design. Input signal task_loop*n*_end is the termination status flag from the computation unit that performs the operations devoted to the inner loop.

The core performs one loop increment per cycle and when a final iteration for a specific loop is reached, this loop as well as its inner loops are reset in the same cycle.

The operation of the core can be halted in case the signal task_loop*n*_end is deasserted. Then, the contents of the index registers of the hardware looping unit are not changed and any activity beyond the comparator modules is ceased.

The loop bound register outputs it is assumed that they can be tristated. This can be achieved with an appropriate design of the loop bound busses. Only when signal output enable (as shown in Figure 2) is active, the loop bound register values can be read from the hardware looping unit.

In the following figures (Figure 3 and 4) the operating modes of the hardware looping unit are indicated.

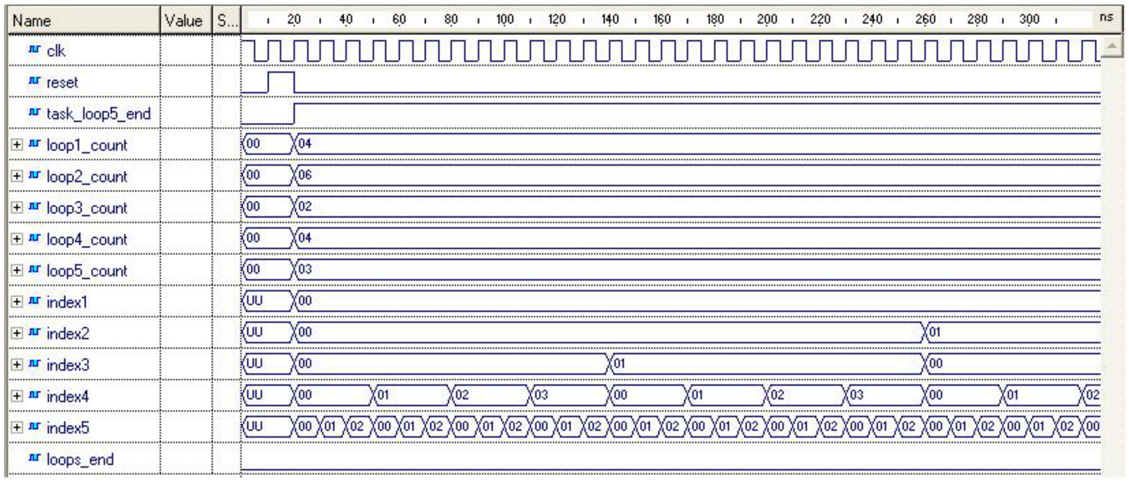
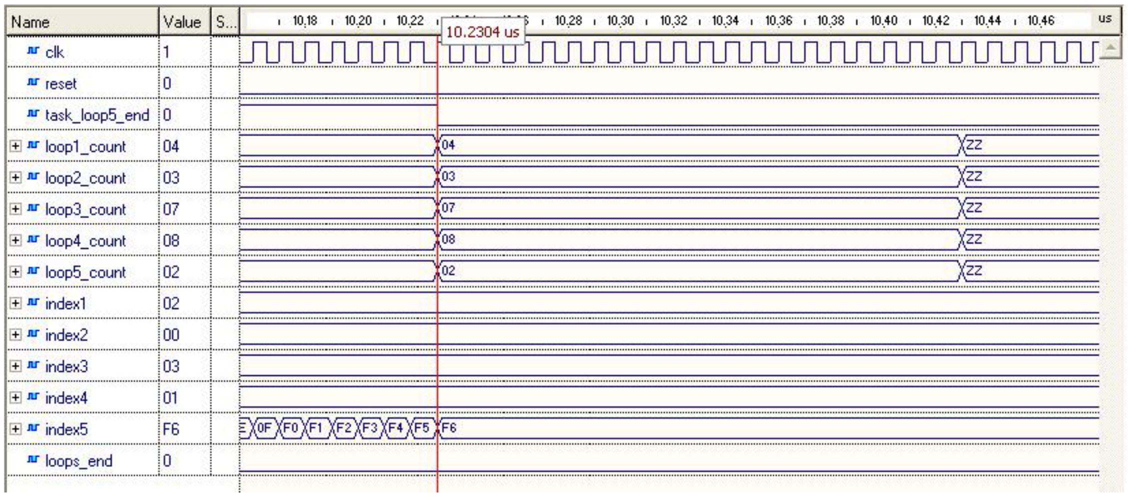


Figure 3. Normal operation of the HWLU



Name	Value	S...
clk	1	
reset	0	
task_loop5_end	0	
loop1_count	04	
loop2_count	03	
loop3_count	07	
loop4_count	08	
loop5_count	02	
index1	02	
index2	00	
index3	03	
index4	01	
index5	F6	
loops_end	0	

Figure 4. Operation of the HWLU when signal task_loopn_end is deasserted

4 Registers

The hardware looping unit contains a single register bank with non user-addressable registers. Parameter DW denotes the register bitwidth and is implemented as a generic in the VHDL sources for the design. The size of the register bank is adjustable through parameter NLP, denoting the maximum number of supported loops.

It is assumed that a possible configuration for the loop bound register bank is This section specifies all internal registers. It should completely cover the interface between the core and the host as seen from the software view.

List of Registers

Name	Address	Width	Access	Description
index1_reg	n.a.	DW	R/W	Index register for loop 1
index2_reg	n.a.	DW	R/W	Index register for loop 2
…	…	…	…	…
index*n*_reg	n.a.	DW	R/W	Index register for loop n (NLP in the VHDL sources). This is the index register for the inner loop

Table 1: List of registers

All index registers have are reset value of zero (0).

5

Clocks

The design uses a single clock which is the system clock (master clock of the processor core where the hardware looping unit) can be situated. There is no inhererent limitation for the system clock timing characteristics except as constrained by the synthesis results.

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Resolution		
clk	Input Pad	As by synth.	-	-	Must be synchronized with the main control unit clock..	System clock.

Table 2: List of clocks

6 IO Ports

This section specifies the IO ports for the hardware looping unit.

Port	Width	Direction	Description
clk	1	Input	Clock input
reset	1	Input	Reset input
task_loopn_end	1	Input	Termination flag for the data computations occurring during an iteration of the inner loop
loop1_count	DW	Input	Loop bound value for loop 1
loop2_count	DW	Input	Loop bound value for loop 2
…			
loopn_count	DW	Input	Loop bound value for loop n
loops_end	1	Output	Termination flag for the entire loop structure
index1	DW	Output	Index register output for loop 1
index2	DW	Output	Index register output for loop 2
…			
indexn	DW	Output	Index register output for loop n

Table 3: List of I/O ports

The reset input is used as synchronous reset in the index incrementer units and as an asynchronous input for the reset control operations.

Appendix A

Software tools

This appendix summarizes the usage of the delivered software tools.

A.1. Usage of the gen_priority_encoder generation tool

Usage of the gen_priority_encoder tool is summarized below:

```
Usage: gen_priority_encoder <num loops> <output base>
where:
num loops   = give number of supported loops
output base = output file base name. The generated files will be named:
              <output base>.vhd for the module
```

A.2. Usage of the gen_hw_looping generation tool

Usage of the gen_hw_looping tool is summarized below:

```
Usage: gen_hw_looping <num loops> <output base>
where:
num loops   = give number of supported loops
output base = output file base name. The generated files will be named:
              <output base>_top.vhd
```

Appendix B

To Dos

This section summarizes some additions to the hardware looping unit distribution that might appear in the future.

- A comprehensive table summarizing the modes of operation in Section 3 (Operation).
- Generation tool for simple testbench for the top-level module.
- Incorporation of the loop bound register bank in the hardware looping unit. Its initialization sequence will be determined by the main control unit.
- Synthesis results for Mentor LeonardoSpectrum, Synopsys FPGA Compiler II, Synopsys Design Compiler.
- Implementation results with Xilinx Webpack and/or Xilinx ISE.
- Flexible versions of the hw_looping(structural) and priority_encoder(rtl) entity/architectures through the use of conditional generate statements. These versions will generate the appropriate hardware supporting from 1 up to 5 loops as selected by the used at compile-time. These will be provided as an alternative method to produce the corresponding modules in case it is not intended to use our generation tools.
- Processing enable control inputs so that operation of the core is halted when requested e.g. by an external signal.

7

References

1. D. Talla, L. K. John, and D. Burger, “Bottlenecks in Multimedia Processing with SIMD Style Extensions and Architectural Enhancements,” IEEE Transactions on Computers, Vol. 52, No. 8, pp. 1015-1031, August 2003.

2. D. Talla, “Architectural Techniques to Accelerate Multimedia Applications on General-Purpose Processors,” Ph.D. thesis, University of Texas at Austin, August 2001.

3. D. Talla and L. K. John, “Cost-effective hardware acceleration of multimedia applications,” In Proceedings of the IEEE International Conference on Computer Design, pp. 415-424, September 2001, Austin, Texas.

4. N. Kavvadias and S. Nikolaidis, “Parametric architecture for implementing multimedia algorithms,” In Proceedings of the 14th International Conference on Digital Signal Processing, July 2002, Santorini, Greece.