loopgen user manual

1. Introduction

The loopgen IP collection provides fast hardware architectures for implementing nested loop structures. The collection comprises of three different architectures (variants) adhering to a common I/O interface, namely hwlu, a mixed-level structural/RTL architecture, ixgenb, a behavioral-level and ixgenr, a high-performance, pure RTL description of a more generalized form of the architecture.

Hardware looping architectures have potential uses for data-intensive processing in embedded systems. The implemented architectures are able to provide all necessary control means for executing perfect loop nests without any cycle overhead for updating the iteration vector. Actually, successive last iterations of nested loops are performed in a single cycle. Such architectures can be used for implementing zero-cycle overhead loop controllers for perfect loop nests operating on multi-dimensional data.

The following sections provide details on the contents of the delivered IP cores, which include all necessary materials such as source files and scripts for RTL simulation and logic synthesis.

Reference documentation for LOOPGEN can be found in the /doc subdirectory in plain text, HTML and PDF form.
2. Functional description

LOOPGEN is implemented as three distinct variants following the exact same interface (hwlu, ixgenb, ixgenr). All variants support the following two generics for hardware configuration:

- NLP: number of supported hardware loops.
- DW: datapath bitwidth.

The following table summarizes the LOOPGEN default interface.

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>1</td>
<td>I</td>
<td>Clock input</td>
</tr>
<tr>
<td>RESET</td>
<td>1</td>
<td>I</td>
<td>Reset input</td>
</tr>
<tr>
<td>INNERLOOP-_END</td>
<td>1</td>
<td>I</td>
<td>Termination flag for the data computations occurring during an iteration of the innermost loop.</td>
</tr>
<tr>
<td>LOOP_COUNT</td>
<td>NLP*DW</td>
<td>I</td>
<td>Loop bound values for each loop</td>
</tr>
<tr>
<td>STRIDE</td>
<td>NLP*DW</td>
<td>I</td>
<td>Stride values for each loop</td>
</tr>
<tr>
<td>INDEX</td>
<td>NLP*DW</td>
<td>O</td>
<td>Index register output for each loop</td>
</tr>
<tr>
<td>DONE</td>
<td>1</td>
<td>O</td>
<td>Termination flag for the entire loop structure</td>
</tr>
</tbody>
</table>

The corresponding interface block diagram is shown below. Each core uses a single external clock source, connected to signal CLK. It can be reset with the active high signal RESET. The signal INNERLOOP_END indicates that the innermost loop computations for the current iteration vector have completed and a new iteration vector must be computed. Data inputs LOOP_COUNT and STRIDE provide the loop bound value and stride (step), correspondingly for each loop, in the form of long, concatenated vectors. To address the LOOP_COUNT entry for the i-th loop, the following field selection should be used: loop_count(i*DW-1 downto (i-1)*DW), assuming that the loops are enumerated from 1 to NLP, with the NLP-th loop being the innermost one. The iteration vector is represented by the INDEX output vector, which again follows the conventions of LOOP_COUNT and STRIDE. DONE signifies the end of the computation for the entire loop structure.

![Loopgen I/O Interface Diagram](image)

Figure 1: loopgen I/O interface.

The default interface is supported by the parametric testbench for the purpose of simulations. It is possible to use the provided software tool (specifically gen_hwlu) to generate a LOOPGEN core with an interface where LOOP_COUNT, STRIDE and
INDEX are split into corresponding vectors of size $D_W$. However, this version of $hwlu$ is not supported by the testbench.

It is useful to mention that the LOOPGEN IP can be viewed as implementing the following C-like looping structure in hardware:

```c
for (i1 = 0; i1 < loop_count1; i1 += stride1) {
    for (i2 = 0; i2 < loop_count2; i2 += stride2) {
        ...
        for (inlp = 0; inlp < loop_countnl; inlp += striden) {
            // innermost loop computations
        }
    }
}
```

All variants implement the same cycle timing. A new iteration vector can be calculated every single clock cycle.

### 3. Architecture

#### 3.1 HWLU

The HWLU architecture can be configured for any number of loop nesting levels to eliminate branch instruction overhead for loop increments. The user can regenerate the corresponding files for the top-level module ($hwlu$) and the priority encoder used internally ($prenc$) for a different number of supported loops.

The following figure shows the block diagram of the hardware looping architecture.

![Block diagram of the HWLU](image)

Figure 2: Block diagram of the HWLU.

Loop index values are produced every clock cycle based on the corresponding loop bound and stride values for each level of nesting. The initial value for the loop indices is provided by a reset mechanism, 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 magnitude comparator (cmpge) outputs (bit-wise flag signals) and an external signal presumably from the datapath (innerloop_end). 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 during the subsequent cycle. For a non-outermost loop, its immediate parent loop index is incremented. In case that none of the loops is terminating, then the inner loop is incremented. Signal innerloop_end guards this increment operation.

Finally, signal done designates that processing in the entire loop structure has terminated.

3.2 IXGENB

The ixgenb architecture variant is based on a hardware algorithm for zero-overhead looping on perfect nests. This algorithm automates the design of compact and efficient hardware looping units that can be implemented as fully synchronous hardware. The looping units of this type are hereafter termed as index generators’, and abbreviated to i[IXGEN] which is also used when referring to the algorithm. These units can be also viewed as tuple generators, covering the entire space of “d”-tuples for d-dimensional data processing.

The first form of the algorithm, named IXGENB, is directly applicable in context of a behavioral HDL model for any number of loops. The pseudocode semantics for implementing these mechanisms can be found below.

```plaintext
local temp_index: temporary copy of index.
parameter NLP: number of supported loops.
begin
  if innerloop_end == 1 then
    for i in 1 to NLP loop
      if temp_index[i] + stride[i] < loop_count[i] then
        if i greater than 1 then
          for j in 0 to i-1 loop
            temp_index[j] = zero;
          endfor;
        endif
        temp_index[i] = temp_index[i] + stride[i];
        break;
      endif;
    endfor;
  endif;
  if temp_index[] + stride[] >= loop_count[] then
    temp_index = zero;
    done = 1;
  endif;
endfor;
index = temp_index;
end;
```

When the data processing in the inner loop is completed, innerloop_end is asserted and a cascaded set of comparisons between index registers to their corresponding loop bound values is activated. The flow of comparisons is directed from outermost to
their immediately innermost loops. If the index value is less than the loop bound for a
given loop \( i \), the index is incremented by a stride value, while all its outer loops are set
to the initial index values. After the first successful comparison, the cascaded structure
is prematurely exited in a form similar to the \texttt{break} statement of the C programming
language. Given that the cascaded comparisons fail, an index value which is lexicograp-hically larger or equal to \texttt{loop\_count} signifies the end of processing in the loop
nest.

### 3.3 IXGENR

The second form of the algorithm, named \textit{IXGENR}, describes an HDL code generator
of an equivalent index generation unit. Its main difference lies in the fact that it uses
a priority encoded scheme that cannot be specified in a parameterized manner using
natural HDL semantics. The pseudocode semantics of algorithm \textit{IXGENR} can be found
below.

Here, the temporary signals \texttt{temp\<n\>\_index} and \texttt{loop\_count\<n\>} are used
where \( n \) is the current loop enumeration. In the generated HDL code, these signals
are defined as aliased names of elements of the \texttt{index} and \texttt{loop\_count} vectors,
respectively. It should be noted than all lines featuring a call to the \texttt{PRINT()} routine
illustrate emitted code.

```plaintext
local    temp\_index: temporary copy of index.
alias    temp\_index\[i\]: i-th segment of temp\_index.
alias    loop\_count\[i\]: i-th segment of loop\_count.
parameter NLP: number of supported loops.
begin
    PRINT(if innerloop\_end = 1 then);
    for i in n downto 1 loop
        if i == n then
            PRINT(if temp\_index\[i\] + stride\[i\] < loop\_count\[i\] then)
        else
            PRINT(elsif temp\_index\[i\] + stride\[i\] < loop\_count\[i\] then)
        endif;
        for j in n downto i+1 loop
            PRINT(temp\_index\[i\] = zero;)
        endfor;
        PRINT(temp\_index\[i\] = temp\_index\[i\] + stride\[i\];)
    endfor;
    PRINT(else)
    for i in 1 to NLP loop
        PRINT(temp\_index\[i\] = zero;)
    endfor;
    done = 1;
    PRINT(endif)
    PRINT(endif)
    PRINT(endif)
    PRINT(index = temp\_index;)
end
```
4. File listing
The LOOPGEN distribution includes the following files.

<table>
<thead>
<tr>
<th>File Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/loopgen</td>
<td>Top-level directory</td>
</tr>
<tr>
<td>/bench/vhdl</td>
<td>Benchmarks VHDL directory</td>
</tr>
<tr>
<td>loopgen_tb_tmpl.vhd</td>
<td>Testbench template for the loopgen IP.</td>
</tr>
<tr>
<td>/doc</td>
<td>Documentation directory</td>
</tr>
<tr>
<td>AUTHORS</td>
<td>List of authors.</td>
</tr>
<tr>
<td>LICENSE</td>
<td>End-user license agreement for using xmodz.</td>
</tr>
<tr>
<td>loopgen-if.png</td>
<td>PNG image illustrating the loopgen IP I/O interface.</td>
</tr>
<tr>
<td>hwlu-arch.png</td>
<td>Block diagram of the hwlu architecture.</td>
</tr>
<tr>
<td>loopgen-pb.pdf</td>
<td>Product brief (brochure) for the LOOPGEN IP cores.</td>
</tr>
<tr>
<td>README</td>
<td>This file.</td>
</tr>
<tr>
<td>README.html</td>
<td>HTML version of README.</td>
</tr>
<tr>
<td>README.pdf</td>
<td>PDF version of README.</td>
</tr>
<tr>
<td>VERSION</td>
<td>Current version of the LOOPGEN IP cores.</td>
</tr>
<tr>
<td>/rtl/vhdl</td>
<td>RTL source code directory for the IP core</td>
</tr>
<tr>
<td>add_dw.vhd</td>
<td>Generic, 2’s complement binary adder.</td>
</tr>
<tr>
<td>cmpge.vhd</td>
<td>Greater than-or-equal comparator.</td>
</tr>
<tr>
<td>hwlu.vhd</td>
<td>Sample hwlu architecture generated by the gen_hwlu tool for NLP=3.</td>
</tr>
<tr>
<td>index_inc.vhd</td>
<td>Index incremerter unit for the hwlu architecture.</td>
</tr>
<tr>
<td>ixgenb.vhd</td>
<td>Behavioral implementation of loopgen as the ixgenb architecture.</td>
</tr>
<tr>
<td>ixgenr.vhd</td>
<td>Sample ixgenr architecture generated by the gen_ixgen tool for NLP=3.</td>
</tr>
<tr>
<td>prenc.vhd</td>
<td>Priority encoder for the hwlu architecture generated by the gen_prenc tool for NLP=3.</td>
</tr>
<tr>
<td>reg_b.vhd</td>
<td>Single-bit D-type register with load enable.</td>
</tr>
<tr>
<td>reg_dw.vhd</td>
<td>Generic D-type register with load enable.</td>
</tr>
<tr>
<td>/sim/rtl_sim</td>
<td>RTL simulation files directory</td>
</tr>
<tr>
<td>/sim/rtl_sim/bin</td>
<td>RTL simulation scripts directory</td>
</tr>
<tr>
<td>hwlu.do</td>
<td>do script for simulating the hwlu architecture with Modelsim.</td>
</tr>
<tr>
<td>hwlu.mk</td>
<td>GNU Makefile for simulating the hwlu architecture with GHDL.</td>
</tr>
<tr>
<td>ixgenb.do</td>
<td>do script for simulating the ixgenb architecture with Modelsim.</td>
</tr>
<tr>
<td>ixgenb.mk</td>
<td>GNU Makefile for simulating the ixgenb architecture with GHDL.</td>
</tr>
<tr>
<td>ixgenr.do</td>
<td>do script for simulating the ixgenr architecture with Modelsim.</td>
</tr>
</tbody>
</table>
5. Simulation

The LOOPGEN IP cores distribution supports both GHDL and Mentor Modelsim simulation. In this section, it is assumed that the required software applications located in the /sw subdirectory have been already build (more on this in Section 7).
5.1. GHDL

For running the GHDL simulation, change directory to the `/sim/rtl_sim/run` subdirectory:

```
$ cd $LOOPGEN_HOME/sim/rtl_sim/run
```

assuming `LOOPGEN_HOME` is the directory where the top-level `/loopgen` is found.

Then, the `sim.sh` shell script is executed, with appropriate command-line arguments for e.g. the `hwlu` architecture:

```
$ ./sim.sh hwlu ghdl
```

The simulation produces a VCD (waveform) dump named `hwlu.vcd` (correspondingly `ixgenb.vcd` or `ixgenr.vcd` for the other two architectural variants) which is automatically copied to the `/sim/rtl_sim/out` subdirectory. In order to distinguish between different values for generic parameters, the VCD files are renamed to reflect the corresponding settings, e.g. `hwlu_3_8.vcd` for `NLP=3` and `DW=8`.

The `ixgenb` and `ixgenr` designs can be simulated in the same way, if you replace appropriately `hwlu` in the instructions above.

5.2. Modelsim

For running the Modelsim simulation, e.g. for the `hwlu` case the corresponding shell script is executed from the `/sim/rtl_sim/run` subdirectory:

```
$ ./sim.sh hwlu mti
```

As in the GHDL case, the VCD dump is produced.

Again, the `ixgenb` and `ixgenr` designs can be simulated in the same way, if you replace appropriately `hwlu` in the aforementioned instructions.

5.3 Example operation

The operation of the core is rather simple. Input signal `clk` is the system clock for the design. Input signal `innerloop_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 `innerloop_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 following figure illustrates the timing diagram of a `loopgen` unit, for `loop_bound` values equal to `X"4"`, `X"7"`, `X"4"` and `stride` values equal to `X"3"`, `X"1"`, `X"2"` for each loop (`NLP=3,DW=4`).

```
8
```
Figure 3: Timing diagram from sample operation of the loopgen unit.
6. Synthesis

The LOOPGEN IP cores distribution includes scripts for logic synthesis automation supporting Xilinx ISE. The corresponding synthesis script can be edited in order to specify the following for adapting to the user’s setup:

- **XDIR**: the path to the /bin subdirectory of the Xilinx ISE/XST installation where the xst.exe executable is placed
- **arch**: specific FPGA architecture (device family) to be used for synthesis
- **part**: specific FPGA part (device) to be used for synthesis

6.1. Running the synthesis script

For running the Xilinx ISE synthesis tool, change directory to the /syn/xise/run subdirectory:

```
$ cd $XMODZ_HOME/syn/xise/run
```

and execute the corresponding script (for synthesizing hwlu):

```
$ ./syn.sh hwlu
```

The synthesis procedure invokes several Xilinx ISE command-line tools for logic synthesis as described in the corresponding Makefile, found in the the /syn/xise/bin subdirectory.

Typically, this process includes the following:

- Generation of the *.xst synthesis script file.
- Generation of the *.ngc gate-level netlist file in NGC format.
- Building the corresponding *.ngd file.
- Performing mapping using map which generates the corresponding *.ncd file.
- Place-and-routing using par which updates the corresponding *.ncd file.
- Tracing critical paths using trce for reoptimizing the *.ncd file.
- Bitstream generation (*.bit) using bitgen, however with unused pins.

Finally, the hwlu.bit bitstream file is produced.

The same process can be applied for synthesizing the ixgenb and ixgenr designs as well. Notably, ixgenb albeit written in behavioral VHDL is well-accepted and supported for synthesis by the Xilinx XST/ISE logic synthesis tool.

7. Reference software application

Three C applications for generating architectures or components of LOOPGEN are located in the /sw subdirectory:

- **gen_prenc.c**: a priority encoder generator for hwlu
• `gen_hwlu.c`: generator for the top-level design file of `hwlu`

• `gen_ixgen.c`: generator for the `ixgenr` architecture.

To build all executables, the supplied GNU Makefile can be used:

```
make clean ; make
```

All three generators have similar options; use `-h` as an argument to emit usage information. For instance, in order to generate the `ixgenr` architecture, use the following, presumably from within the `sw` subdirectory:

```
./gen_ixgen.exe -nlp 3 ixgenr
```

This prompt will generate a design file with the name `ixgenr3.vhd` that is an index generator (looping unit) for three nested loops.

Again, the corresponding simulation script (`sim.sh`) automatically takes care of generating the executables, running them to produce RTL design files, and execute all the requested simulations.

### 8. Prerequisites

- Standard UNIX-based tools (tested with gcc-4.6.2 on MinGW/x86).
  - `make`
  - `bash` (shell)
  - `perl`

For this reason, MinGW (http://www.mingw.org) or Cygwin (http://sources.redhat.com/cygwin) are suggested, since POSIX emulation environments of sufficient completeness.

- GHDL simulator (http://ghdl.free.fr) or Modelsim (http://www.model.com). The latest GHDL distribution (0.29.1, Windows version) also installs GTKwave on Windows.

- Xilinx ISE (free ISE webpack is available from the Xilinx website: http://www.xilinx.com)

### 9. Contact

You may contact me at:

Nikolaos Kavvadias <nikos@nkavvadias.com>

http://www.nkavvadias.com
http://www.perfeda.gr
Perfeda Technologies headquarters
35100 Lamia, Fthiotis
Greece