# Automatically Identifying and Creating Accelerators Directly from C Code

## The Mimosys Clarity tool exploits the power of the Virtex-4 FX PowerPC APU interface.

by Jason Brown, Ph.D.
CEO
Mimosys
*jason.brown@mimosys.com*

Marc Epalza, Ph.D.
Senior Hardware Engineer
Mimosys
*marc.epalza@mimosys.com*

Let's say that you have been tasked to ensure that your company has an H.264 solution that supports high-definition video decoding at 30 frames per second. You are not a video expert. What do you do? You could get on the Internet and perform a Web search for H.264; before you know it, you'll have the source code and be on your way.

You managed to compile the code and get it running on the target, but it decodes at a whopping two frames per second. Now what? After sifting through pages and pages of profiling data, you find some hotspots, but you are not sure which parts to focus on to maximize the acceleration and you do not have enough time to try to optimize them all.

Many of us have found ourselves in this situation at one time or another. Maybe you have even delivered a solution, but not without a lot of sweat and tears.

### Mimosys Clarity

The Mimosys Clarity software tool automatically identifies candidate hardware accelerators directly from application C source code, guided by the execution profile of the application. It also takes a set of constraints on the number of I/O parameters available to the accelerator and a model of the execution costs for operations on the PowerPC™ and in Xilinx® FPGA fabric.

Clarity's approach to profiling is unique in that the execution profiling information is visually presented in the tool as annotations to basic blocks of control flow graphs (CFGs), as shown in Figure 1. The nodes of a control flow graph represent the basic blocks of the C source code, where a basic block is any sequence of C expressions without conditions. The edges of a CFG represent an execution path between basic blocks, where a specific path is followed (or taken) if a particular condition is true or false.

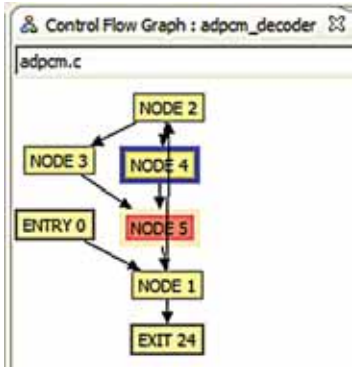


*Figure 1 – Control flow graph (CFG); each node represents a basic block (DFG).*

Because each basic block is a sequence of expressions, it can be represented by a data flow graph (DFG), the nodes of which are operations (+, -, *, /) and the edges values (Figure 2). In the vast majority of cases, this information is not automatically created or visualized. Furthermore, this visualization, if done at all, is typically static and thus unable to retain one of the most important aspects: the correspondence between the graphs and the source code from which they came.

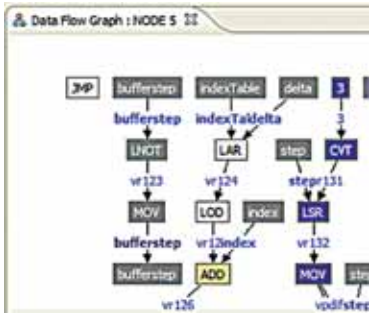


*Figure 2 – Data flow graph (DFG); each node represents a single operation.*

This unique approach of visually representing profiling information in CFGs and DFGs, along with the aforementioned correspondence, helps you quickly hone in on and understand the most computationally intensive parts of the application, as all of the views are always synchronized.

Once you have gathered the application profiling information, you can invoke the automatic accelerator identification search algorithm. This algorithm identifies a set of optimal hardware accelerators that provide the best execution speedup of the application given the constraints.

An important constraint on accelerators is the number of available data inputs and the number of data outputs, which in the current Virtex™-4 FX device is fixed by the PowerPC to two inputs and one output. Figure 3 shows the results from two invocations of the accelerator search algorithm. The upper table uses a constraint of two inputs and one output, with the lower using a constraint of four inputs and one output for the application ADPCM (adaptive differential pulse code modulation).

| Name | Inputs | Outputs | Cycles Saved | Est. SW Latency | Est. HW Latency |
|---|---|---|---|---|---|
| ISE | 2 | 1 | 38.10 % | | |
| Instr_0 | 2 | 1 | 1.52 % | 2 | 1 (0.16) |
| Instr_1 | 2 | 1 | 27.43 % | 10 | 1 (0.90) |
| Instr_2 | 2 | 1 | 6.10 % | 3 | 1 (0.75) |
| Instr_3 | 1 | 1 | 3.05 % | 2 | 1 (0.25) |

| Name | Inputs | Outputs | Cycles Saved | Est. SW Latency | Est. HW Latency |
|---|---|---|---|---|---|
| ISE | 4 | 1 | 47.24 % | | |
| Instr_0 | 2 | 1 | 1.52 % | 2 | 1 (0.16) |
| Instr_1 | 3 | 1 | 39.62 % | 15 | 2 (1.40) |
| Instr_2 | 2 | 1 | 6.10 % | 3 | 1 (0.75) |

*Figure 3 – Instructions found with a constraint of 2-1 (top) and 4-1 (bottom), with corresponding performance increases.*

Clearly, the number of I/Os has a significant impact on the acceleration achieved. In order to realize the higher acceleration while overcoming the hard constraints imposed by the PowerPC, you can use pipelining techniques at the cost of increasing design complexity. Clarity automatically pipelines accelerators, giving access to the higher performance with no extra work for you.

The identified hardware accelerators are optimal; no DFG of the application considered by Clarity contains a sub-graph with better estimated performance. As a consequence, if the set of accelerators discovered provides little or no benefit, it indicates that you need to rework the application source code to expose improved acceleration opportunities. Furthermore, the automatic identification algorithm takes both local (DFG) and global (execution profile) information into account, thus providing unique insight into where to focus your efforts in optimizing an application.

In the iterative process of optimizing an application, the identification algorithm will discover a set of accelerators that will satisfy your requirements. However, the task remains to realize them on the Virtex-4 FX platform. You must perform two important steps to achieve this: first, create an HDL definition of the accelerators that includes the necessary logic to interface the PowerPC APU port with the accelerator itself. You must assign the new accelerators unique identifiers (instructions) so that the PowerPC can be instructed to activate the accelerator by the software application.

The second step is to modify the original application source code to use the new instructions, thus achieving the execution speedup offered by the new hardware accelerators. In a normal design flow both of these steps are manual, although in some ESL design flows you can describe the hardware accelerators using C/C++/SystemC or a derivative programming language. But the software integration step is always manual.

## Realizing the Accelerators

Clarity automates the entire process of creating accelerators for Virtex-4 FX FPGAs, including software integration. This is achieved when you commit the newly dis-

covered accelerators to the Xilinx platform. The commit stage is fully automatic and comprises three parts:

- Modifying the original application source code to use the newly generated accelerators. The source code of the application is modified so that the parts to be implemented in the FPGA are removed and replaced by calls corresponding to the new hardware accelerators. Figure 4 shows a snippet of code from the ADPCM application, in which the removed source code is commented out and replaced with a call to the macro Instr_1. This macro contains the special PowerPC instruction used to activate the hardware accelerator corresponding to the Instr_1 that Clarity discovered and you selected.
- Generating RTL for each accelerator along with the necessary logic to interface the Xilinx PowerPC APU and the accelerator data path. Each accelerator is implemented in VHDL as an execution data path along with the required PowerPC interfacing logic. The data path is translated directly from the C source code and is thus correct by construction. This translation is possible because the data path implements a pure data flow with limited control, avoiding the issues of high-level synthesis. As described above, if the I/O constraints specified for the search exceed those of the PowerPC architecture, the data path will automatically be split into stages and pipelined to fit these constraints.
- Creating a Xilinx Platform Studio (XPS) project containing the RTL implementations and modified application source code. To provide confidence in the correctness and synthesized result of the HDL accelerators, you can have a test bench created automatically for each accelerator along with the necessary simulation and synthesis scripts. You can invoke an HDL simulator such as ModelSim from within Clarity to compare the functionality of the original C code with the HDL code of the accelerator replacing it. The generated synthesis script enables you to perform synthesis on each accelerator in the target technology and obtain information on the critical path and area. Furthermore, this verification test bench provides a framework to ensure the correctness of identified accelerators that may be subsequently modified by a design engineer.

## Hardware Design for Software Engineers

The challenge of identifying hardware accelerators for an application is formidable, especially if expert domain knowledge is required but unavailable. This challenge is made more acute by the difficulties in realizing these accelerators as coprocessors, requiring new interfaces and software integration to manipulate them. By providing a standardized interface, the Virtex-4 FX PowerPC enables new automation technologies that address both identification and realization of accelerators. Furthermore, Clarity can automatically circumvent the apparent I/O limitations of the PowerPC APU.

In particular, Clarity offers a unique solution to this challenge through an innovative combination of automation and visualization techniques (see Figure 5). Working at the level of the C programming language, you can visualize all aspects of the control flow, data flow, and execution behavior of an application. A unique hardware accelerator identification technology automatically discovers and creates application-specific hardware accelerators targeted to the Xilinx Virtex-4 FX device. Fully automatic HDL generation, application software integration, and test bench generation mean that you are freed from any concerns about how to realize application acceleration in hardware, thus empowering you to focus on your product differentiation.

So finding the H.264 source code on the Web was not such a bad idea. You created some useful accelerators and implemented them on the Virtex-4 FX device before lunch time, leaving the afternoon free to explore some different solutions just for fun.

For more information about Clarity, please e-mail *enquiries@mimosys.com* or visit *www.mimosys.com/xilinx*.

```
adpcm.c
    /* Step 2 - Find new index value (for later) */
    index += indexTable[delta];
    if ( index < 0 ) index = 0;
    if ( index > 88 ) index = 88;

    /* Step 3 - Separate sign and magnitude */
    sign = delta & 8;
    /* MIMOSYS  delta = delta & 7; */
    Instr_1(delta, step, &vpdiff);

    /* Step 4 - Compute difference and new predicted value */
    /*
    ** Computes 'vpdiff = (delta+0.5)*step/4', but see comment
    ** in adpcm_coder.
    */
    /* MIMOSYS  vpdiff = step >> 3; */
    /* MIMOSYS  if ( delta & 4 ) vpdiff += step; */
    /* MIMOSYS  if ( delta & 2 ) vpdiff += step>>1; */
    /* MIMOSYS  if ( delta & 1 ) vpdiff += step>>2; */

    if ( sign )
      valpred -= vpdiff;
    else
      valpred += vpdiff;
```

*Figure 4 – Modified application; the code moved into the hardware accelerator has been commented out and replaced with a call to the accelerator. The code is from ADPCM.*

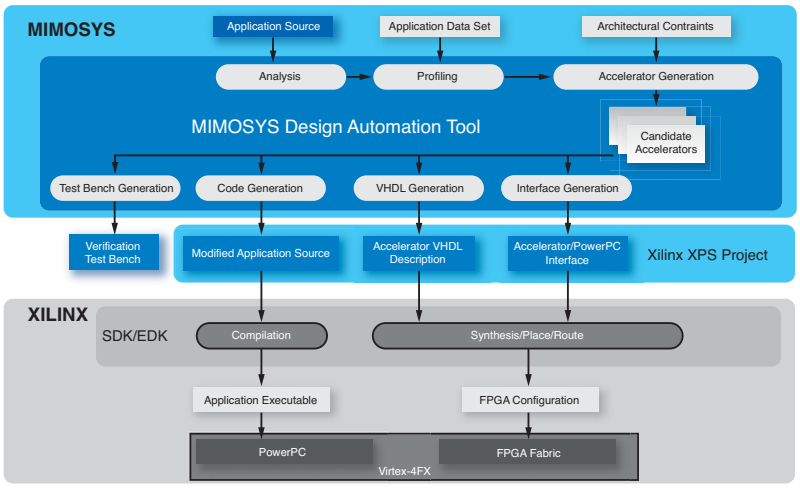


*Figure 5 – Mimosys Clarity flow; from the application source code and some constraints, you can generate a complete Xilinx Platform Studio project with accelerators.*