ACCELERATE COMPUTER VISION DESIGN USING HIGH-LEVEL SYNTHESIS

MIKE FINGEROFF, HLS TECHNOLOGIST, MENTOR, A SIEMENS BUSINESS

Mentor®
A Siemens Business

WHITEPAPER

HIGH-LEVEL SYNTHESIS

www.mentor.com

INTRODUCTION

Digital processing of images and video with complex algorithms in order to interpret meaning defines the world of computer vision. This over $6 billion market is expected to grow 7x by 2022 (*Source: Tractica*) and computer vision solutions are all around us, in cars, consumer products, security, retail, and agriculture. But, designing these solutions is not easy, mainly because of constant algorithm upgrades and related requirements changes (Figure 1). In some cases, the team might be directed to change the technology target (FPGA or ASIC for example). This means that wherever the team is in the RTL creation and verification flow, they might have to start over, which can cause an unacceptable delay in the production schedule. This also means that algorithm designers cannot experiment in real time to determine the impact on hardware implementations.

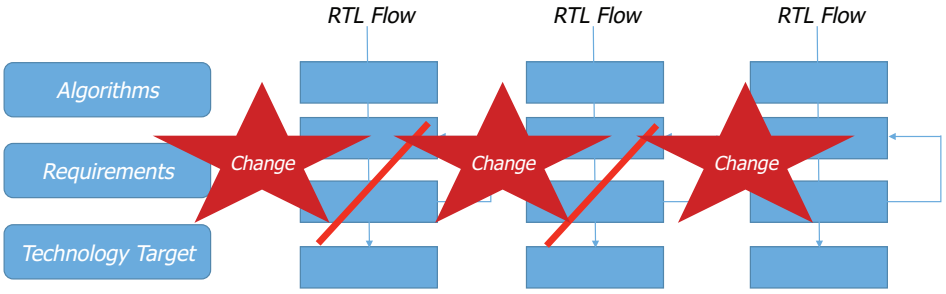

Figure 1: Change causes the RTL flow to start over

If managing change is not enough of a challenge, there are technical challenges as well. Many computer vision systems need real-time speed. For example, a collision avoidance system in a car cannot have delays in decision making. Real-time speed can mean billions of calculations per second, yet designers are also trying to keep power consumption to a minimum. And to make matters even more challenging, camera resolutions and frame rates are expected to double every few years.

THE ROLE OF CONVOLUTIONAL NEURAL NETWORKS

Computer vision systems must understand what they are "seeing" in order to make correct decisions. This understanding comes from deep learning which is commonly implemented using convolutional neural networks (CNN), inspired by the way brain neurons interact to recognize images. CNNs are employed in two distinct ways (Figure 2):

- Training: is typically performed on floating-point GPU farms that can spend weeks processing massive amounts of training data to tune the weights of the network. This iterative process provides many images of the same types of objects with different attributes like orientations, aspect ratios, and clarity, to allow the network to "learn" how to detect and classify these objects.
- Inferencing: the CNN runs convolutions using the weights from the training in order to detect and classify objects in any image. Inferencing can be implemented in fixed-point hardware to be deployed in computer vision systems.

Because many computer vision systems are deployed in cars, equipment, or consumer products, which cannot depend on a lab full of computers, inferencing solutions are the focus of hardware designers.

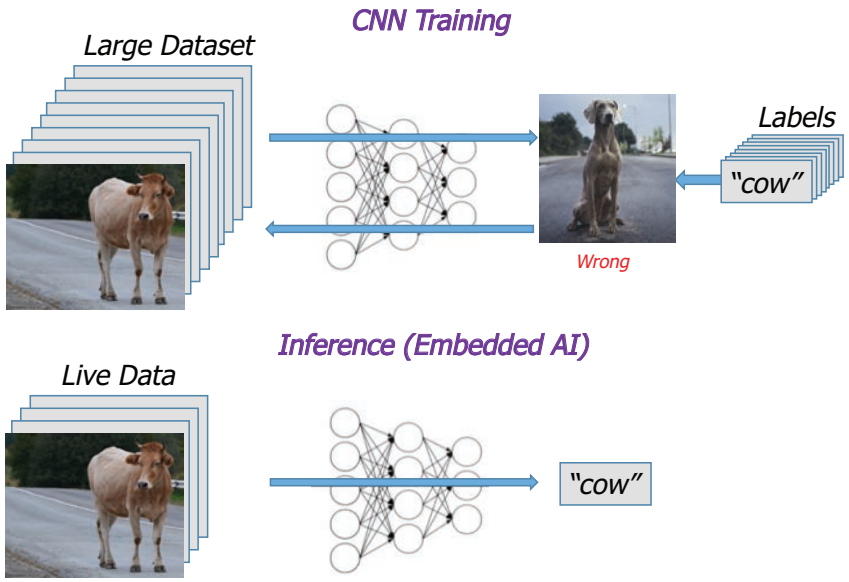

Figure 2: Comparing CNN training and inferencing

HIGH-LEVEL SYNTHESIS TO THE RESCUE

Many years ago, Mentor recognized that computer vision design and verification teams needed to move up from the RTL to the high-level synthesis (HLS) level to address the challenges of computer vision system design. Working with customers over all those years has resulted in today's Catapult® HLS Platform that provides a complete flow from C++/SystemC to optimized RTL (Figure 3).

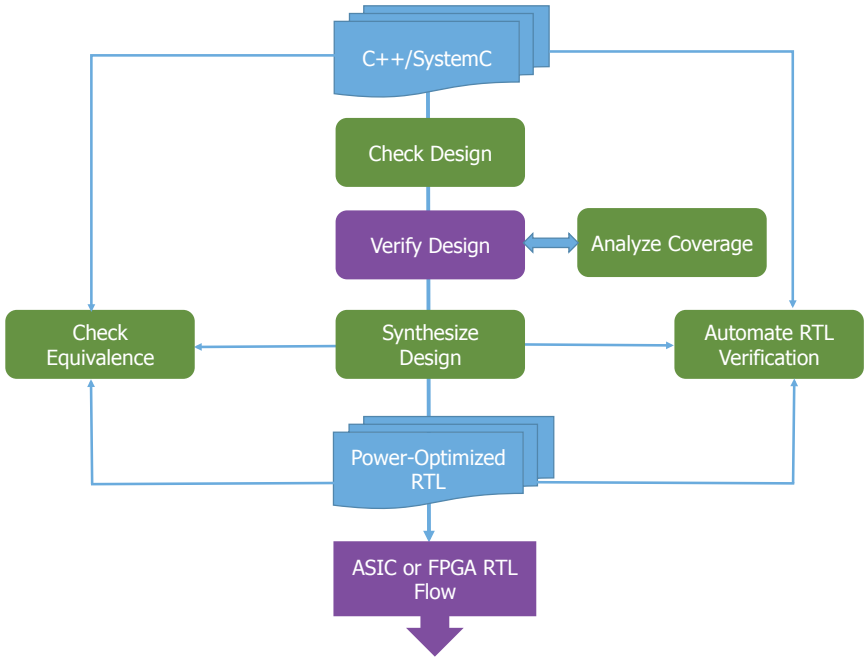

Figure 3: The Catapult HLS Platform flow

The Catapult HLS Platform provides a hardware design solution for algorithm designers that generates high-quality RTL from C++ and/or SystemC descriptions that target ASIC, FPGA, and embedded FPGA solutions. The platform delivers the ability to check the design for errors before simulation, provides a seamless and reusable testing environment, and supports formal equivalence checking between the generated RTL and the original source. The platform flow ensures fast design and verification and delivers power-optimized RTL ready for simulation and RTL synthesis.

By employing these elements of the Catapult HLS Platform solution, teams take their computer vision products to market faster and at a lower cost:

- Enable late-stage changes: easily change C++ algorithms at any time and regenerate RTL code or target a new technology.
- Evaluate hardware: rapidly explore options for power, performance and area without changing source code.
- Accelerate schedules: reduce design and verification time from one year to a few months and add new features in days not weeks, all using C/C++ code that contains 5x fewer lines of code than RTL.

CODING FOR SYNTHESIS

A typical CNN architecture (Figure 4) consists of layers of convolutional filters, pooling, and fully connected layers whose lowest levels of implementation are matrix multiplication and dot-product functions, which are a very good fit for HLS.

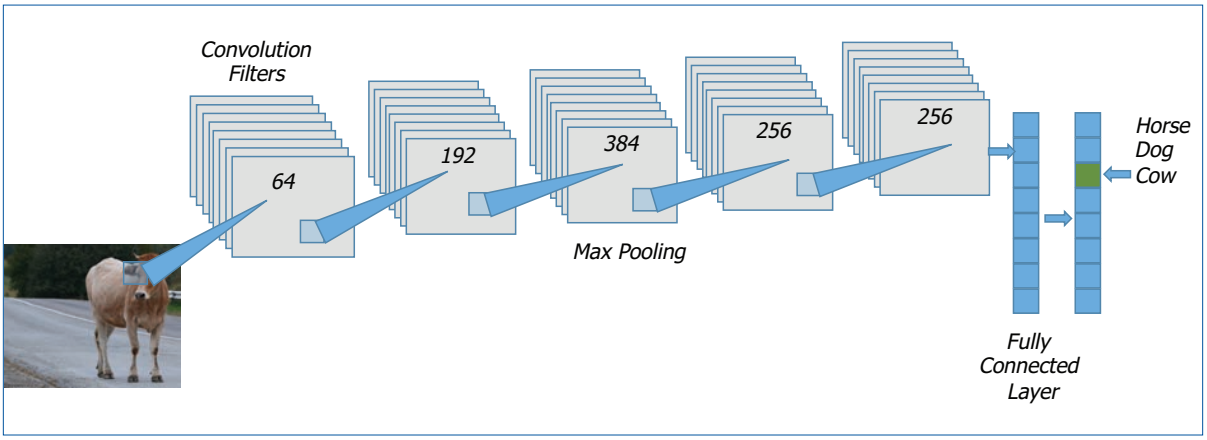

Figure 4: A typical CNN architecture

Not all C++ code is suitable for synthesis. Generally, code is made synthesizable by converting dynamic memory allocation to bounded arrays, replacing floating-point with fixed-point data types, and changing *math.h* function calls to use the appropriate hardware math libraries that are optimized for synthesis. However, the most important aspect of coding for HLS is describing the general hardware architecture in C++ to achieve the best quality of results (Figure 5). This includes concepts such as describing a shift-register versus a memory-based circular buffer implementation for storing image data.

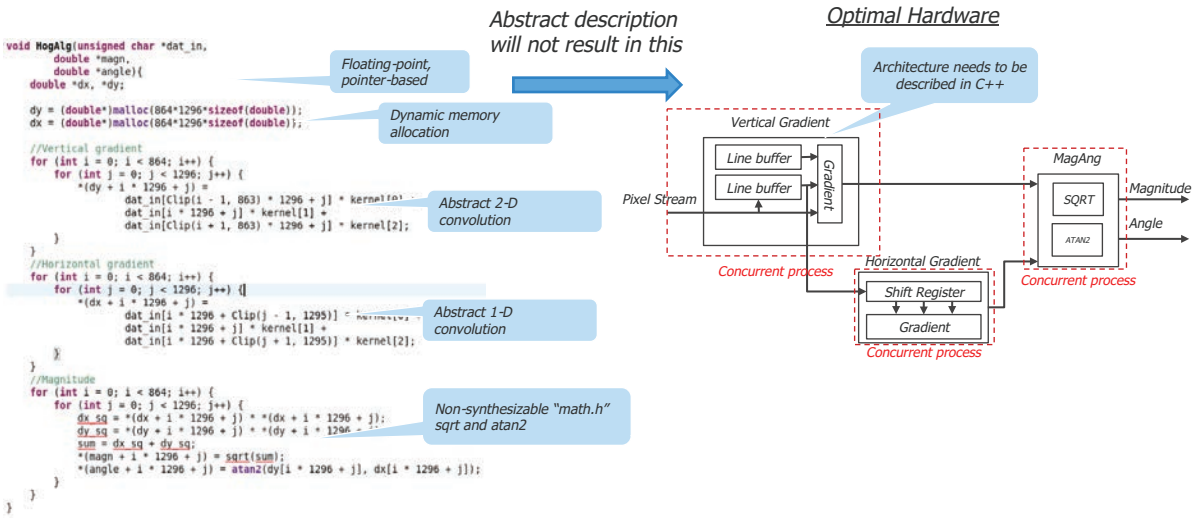

Figure 5: C++ code must be written with hardware in mind

While it is beyond the scope of this paper to exhaustively discuss coding computer vision algorithms for synthesis to hardware, a few key points are:

- Catapult HLS supports bit-accurate, algorithmic C data types that work like any other native C++ data type. Using these data types for fixed, integer, float, and complex numbers allows the code to synthesize into hardware with the same bit-for-bit behavior as the C++ model.
- Catapult HLS needs to know array bounds for internal arrays and for block interface variables (Figure 6).

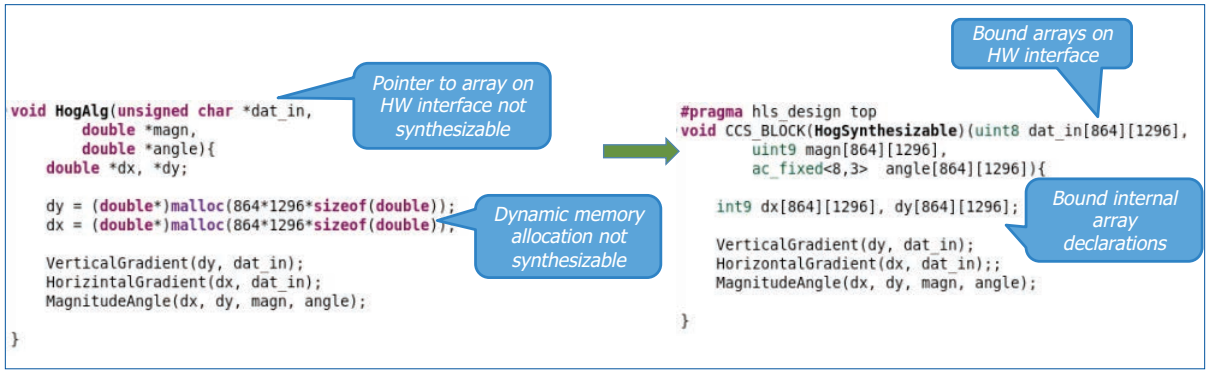

Figure 6: Using array bounds in code

- Computer vision algorithms often send and receive sample data in a streaming fashion. Using pointers is not compatible with HLS, but using the ac_channel class allows efficient representation of interfaces that support streaming producer/consumer behavior and results in correct-by-construction RTL generation.
- Array access patterns in C++ can impact the performance of the resulting memory architecture. Memory accesses tend to be the performance bottleneck in a design, so the designer must be careful to avoid coding too many accesses to an array that is mapped to memory. Catapult HLS provides designers the flexibility of easily choosing memory architectures such as FIFOs, circular buffers, and multi-bank memories by changing a few lines of C++ code. These architectural tradeoffs are not feasible in an RTL flow.

After understanding how to code for successful hardware implementations, the Catapult HLS Platform feature set can help designers address the challenges of computer vision design.

UNDERSTANDING WHY CATAPULT ADDRESSES THE CHALLENGES

Perhaps computer vision design teams are initially drawn to HLS because the Catapult HLS Platform generates correct-by-construction RTL from C++ descriptions. The tool builds concurrent RTL modules from C++ classes and functions that are synthesized into concurrent, clocked processes (no need for multi-threaded design and debugging). But, a closer look at the tool demonstrates why it is so useful for computer vision design.

AUTOMATICALLY ADDING INTERFACE PROTOCOLS

Instead of coding interface protocols in source code, designers can use Interface Synthesis to add a timed protocol to untimed C++ function interface variables. This is done by setting architectural constraints for the protocol in the Catapult GUI (Figure 7) or as a TCL command. The tool supports typical protocols such as AXI4 video stream, request/acknowledge handshaking, and memory interfaces. This allows designers to explore interface protocols without changing the C++ source.

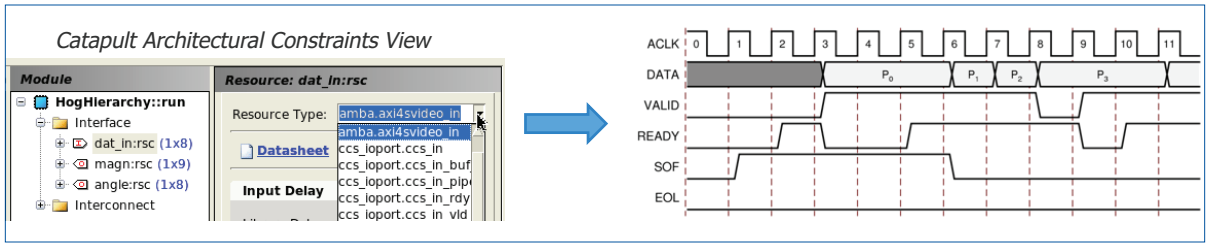

Figure 7: Selecting interface protocols

CLOSING TIMING

HLS adds time to the C++ algorithm using a process called scheduling (Figure 8). Based on the selected technology and clock frequency, scheduling examines the operations that the algorithm describes and decides which clock cycle they execute. Scheduling inserts registers between operations (pipelining) to meet clock constraints and to minimize combinational path delays. This insures that timing closure is automatic, however the designer has control over optimizing for latency or area.

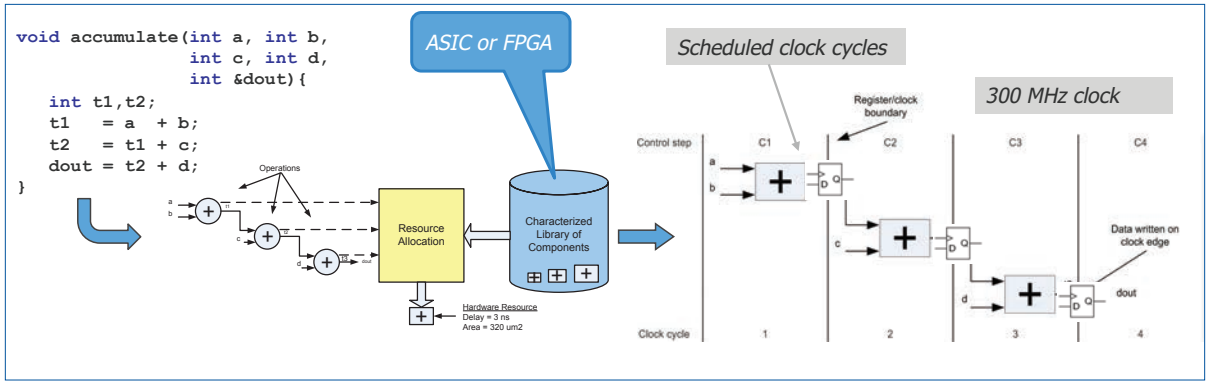

Figure 8: Closing timing based on technology and clock frequency

ANALYZING PERFORMANCE BOTTLENECKS

One of the key questions that C++ algorithm developers have is, "What happened during synthesis to RTL?" In particular, they need to determine the performance of the design and identify any bottlenecks in the synthesized design. Catapult provides a Schedule view of the design for analysis (Figure 9). This view quickly shows what loops are consuming the most time with green bars in the Runtime Profile.

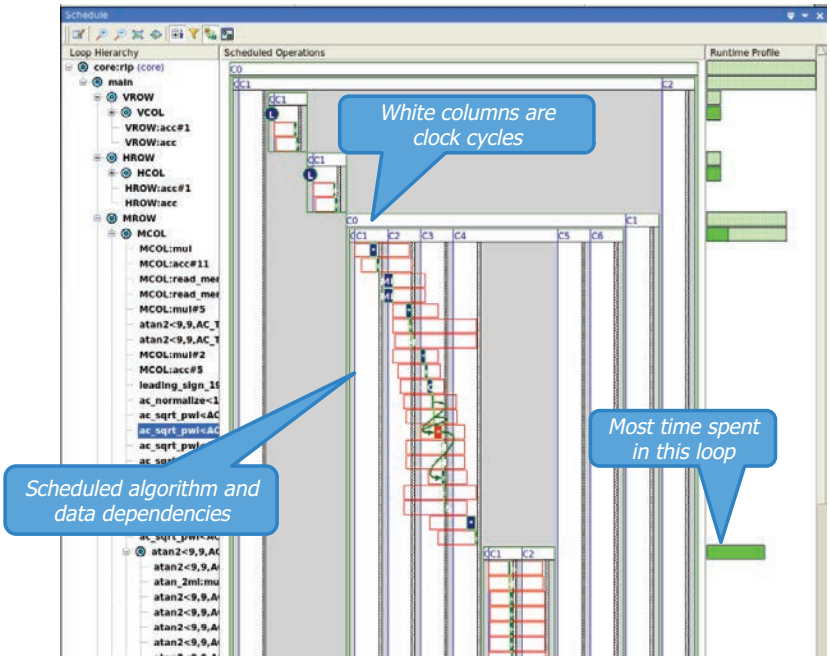

Figure 9: A graphical view of the scheduled design

Scheduling could fail if the constraints are impossible to meet with the existing C++ description. The tool provides advanced feedback that shows designers critical timing paths relating to a failed scheduling run (Figure 10). This visualization allows designers to trace dataflow timing issues down to the source code in order to understand and to correct the problem.

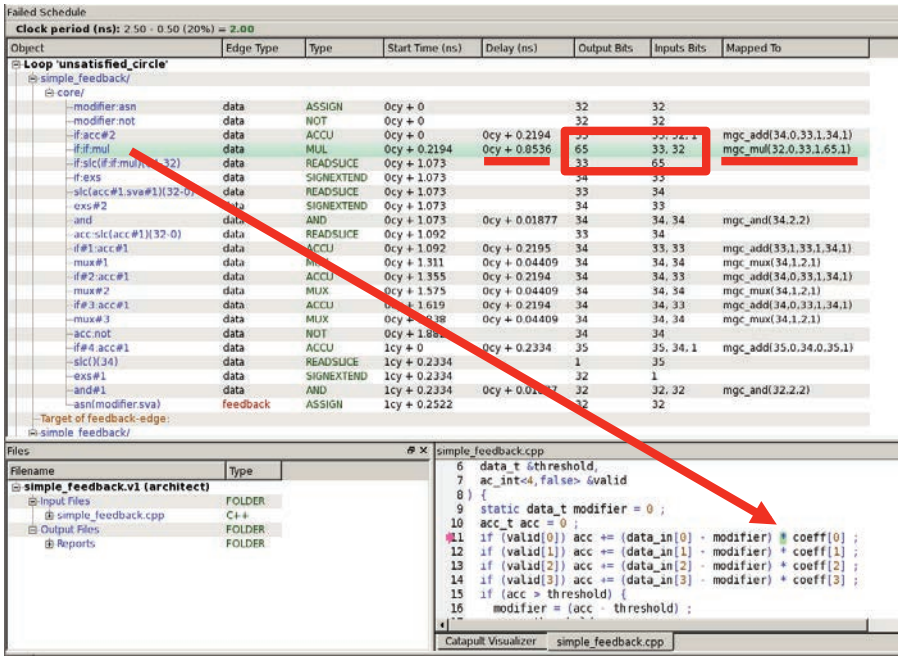

Figure 10: Advanced violation analysis when failing to schedule the design

Typical violations are caused by competition between resources, data feedback dependencies, or memory read/write dependencies. Sometimes, changing the design constraints can solve the problem. Other times, architectural changes to the C++ are required (similar to tradeoffs made in RTL flows).

EXPLORING PARALLELISM

Designers can explore parallelism by performing loop transformations within the tool. Unrolling loops drives parallelism while pipelining influences throughput and the maximum frequency. For a selected loop in the design, the designer can perform what-if analysis by changing architectural constraints and then use the Visualizer to see the RTL schematic that results from synthesis (Figure 11).

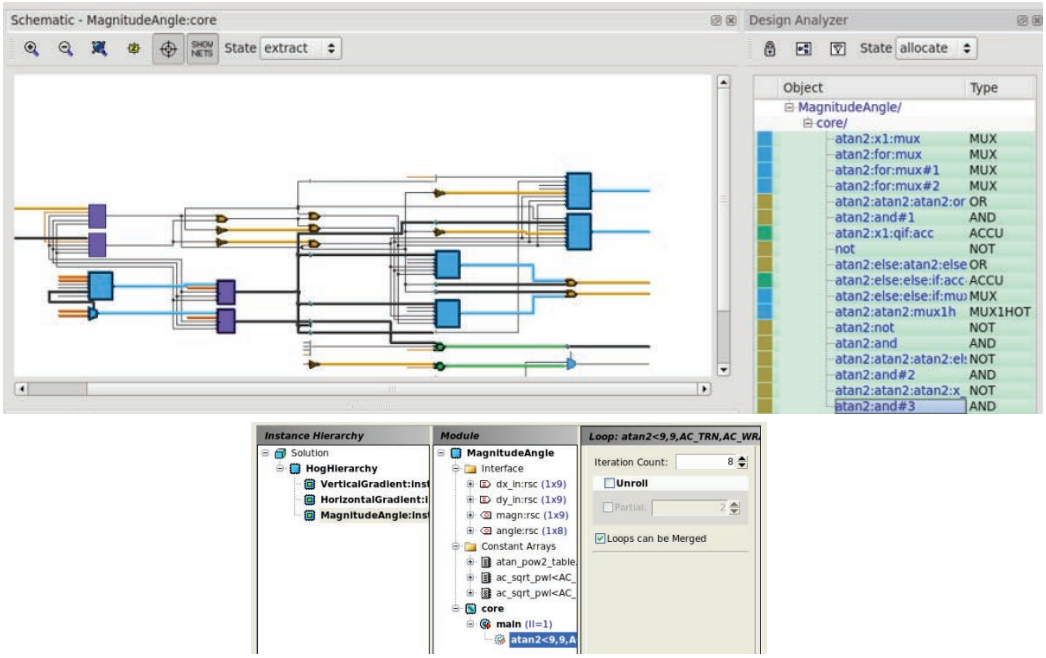

Figure 11: Performing what-if analysis of parallelism constraints

INFERRING AND OPTIMIZING MEMORY

Catapult automatically infers memory from arrays and maps to ASIC or FPGA memories/registers (Figure 12). Arrays on the design interface can be synthesized as memory interfaces and internal arrays can be synthesized to instantiated (black-boxed) memories.

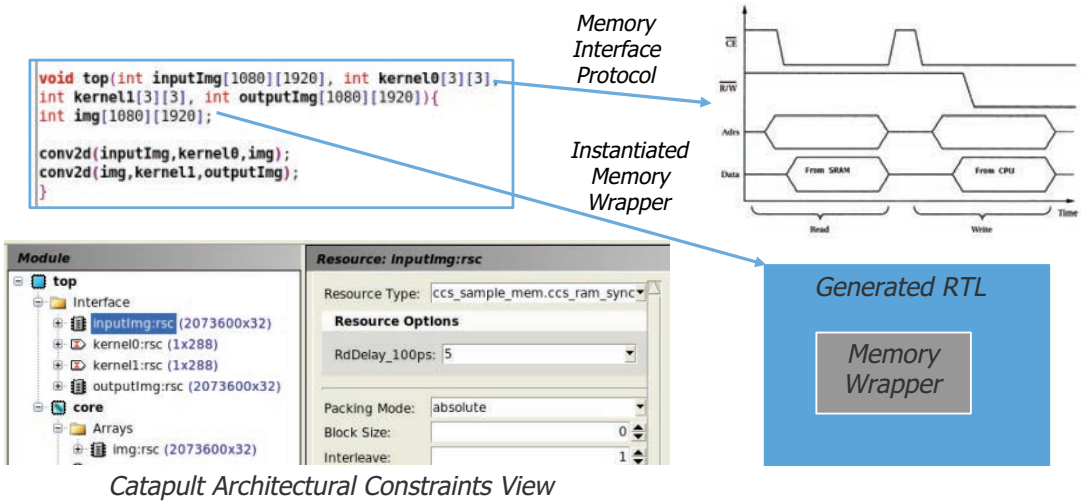

Figure 12: Inferring memory from arrays

The designer has control over memory bandwidth and partitioning, including word width, packing mode, and controlling the number of memories by specifying Block or Interleave methods. In combination with loop unrolling, the designer can easily increase memory bandwidth.

SHARING RESOURCES

To keep area to a minimum, Catapult automatically shares resources by looking for explicit mutual exclusivity in the code and by applying design constraints. It performs register sharing by performing lifetime analysis, guided by design constraints (Figure 13).

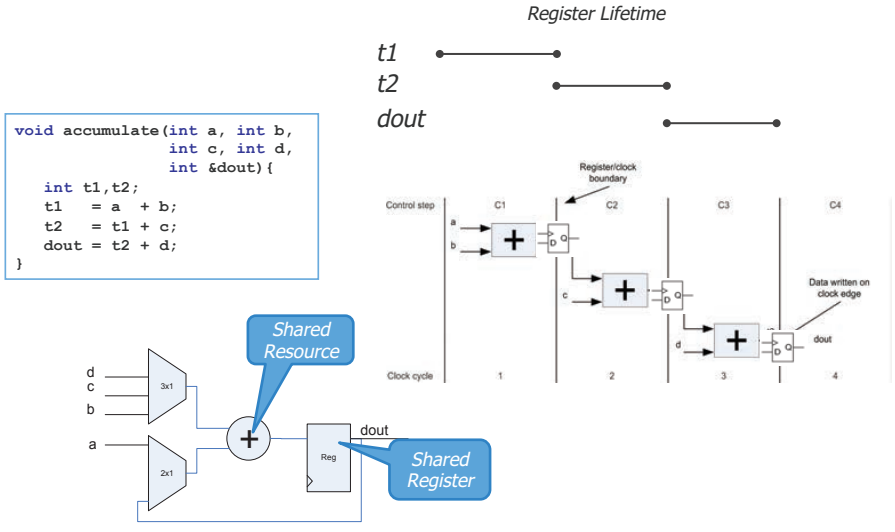

Figure 13: Resource sharing directed by constraints

EXPLORING AND OPTIMIZING POWER

Using PowerPro® technology, Catapult generates power-optimized RTL to converge rapidly on an ideal design solution. Designers can interactively explore micro-architectures using constraints to evaluate power, performance, and area tradeoffs. They can explore design choices and determine the effect on power using techniques such as numerical refinement, frequency exploration, memory access minimization (banking and interleaving), and resource sharing.

LEVERAGING LIBRARIES

When designing computer vision systems, it not hard to notice that there are low-level building blocks instantiated many times within the same design, such as convolutional filters, interfaces, sliding window memory architectures, pooling, line buffers, and image flag classes. The Catapult HLS Platform provides parameterized C++ libraries for these low-level, basic imaging operations and Mentor is continually adding to high-level libraries of complete IP imaging classes to use as a starting point for the design team's next project.

THE TAKE-AWAY

The world of computer vision design is constantly changing, which means traditional RTL flows cannot keep up with schedules. In order to quickly respond to changes and to accelerate design and verification, high-level synthesis offers a path to success. With its myriad of solutions, the Catapult HLS Platform provides a unique methodology that addresses the challenges of computer vision design.

To learn more about how HLS is revolutionizing computer vision innovation and to experience customer success stories, view the free video seminar series here.

For the latest product information, call us or visit: www.mentor.com

Corporate Headquarters
Mentor Graphics Corporation
8005 SW Boeckman Road
Wilsonville, OR 97070-7777
Phone: 503.685.7000
Fax: 503.685.1204

Sales and Product Information
Phone: 800.547.3000
sales_info@mentor.com

Silicon Valley
Mentor Graphics Corporation
46871 Bayside Parkway
Fremont, CA 94538 USA
Phone: 510.354.7400
Fax: 510.354.7467

North American Support Center
Phone: 800.547.4303

Europe
Mentor Graphics
Deutschland GmbH
Arnulfstrasse 201
80634 Munich
Germany
Phone: +49.89.57096.0
Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics (Taiwan)
11F, No. 120, Section 2,
Gongdao 5th Road
HsinChu City 300,
Taiwan, ROC
Phone: 886.3.513.1000
Fax: 886.3.573.4734

Japan
Mentor Graphics Japan Co., Ltd.
Gotenyama Trust Tower
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140-0001
Japan
Phone: +81.3.5488.3033
Fax: +81.3.5488.3004

Mentor® A Siemens Business

MGC 07-18 TECH17570-w