

# INSIDERS' GUIDE: FPGAs, TOOLS, AND BOARDS



## FEATURED INTERVIEW:

EXCERPTED FROM [WWW.EG3.COM](http://WWW.EG3.COM)



*Prepared by:*

[eg3.com](http://eg3.com)

Jason McDonald, Senior Editor

[eg3.com](http://eg3.com)

tel: 510.713.2150

email: [info@eg3.com](mailto:info@eg3.com)

web: <http://www.eg3.com>



## AGILITY: FPGAs FOR FAST IMAGE PROCESSING ALGORITHMS

10 October 2008: FPGAs for Fast Image Processing Algorithms

INTERVIEWEE. LARRY MELLING,  
 VP MARKETING  
 TEL. 650 846 2555  
 EMAIL. larry.melling@agilityds. com  
 COMPANY. AGILITY DESIGN SOLUTIONS INC.  
 WEB. [http://www. agilityds.com/](http://www.agilityds.com/)

**Q. First of all, tell us a little bit about yourself and your responsibilities at Agility.**

A. I am vice president of marketing for Agility Design Solutions, responsible for corporate and product marketing. I have more than 15 years of executive management experience in bringing technology to market for both hardware and software development. I joined Agility (formerly Catalytic) from Virtio Corporation, which provides system models for pre-silicon software development. Prior to Virtio (now part of Synopsys), I was vice president of marketing at IKOS Systems (now part of Mentor Graphics Corporation), a leading provider of emulation and acceleration products for SoC verification. I have Bachelor of Science degree in Electrical Engineering from Purdue University and started my career as a design engineer at Hewlett Packard.

**Q. Algorithms are one of the most common functions performed by DSPs and/or FPGAs, so can you explain to us at a basic level the how your company helps out algorithm developers?**

A. Algorithms start as abstract, implementation agnostic mathematical models. The Agility approach is to provide a development environment that starts with the algorithm in its abstract form (MATLAB or C model/description) and enable fast implementation through automation. The value is starting from the abstract model and maintaining a verification flow from abstract model to implementation.

This is especially valuable for mathematical models that are done in floating point and rely on mathematical manipulations to process incoming images or signals. The process of implementation can include floating point to fixed point conversion and modification from frame based processing to stream processing to give two examples of some of the transformation that occurs. The Agility environment helps developers with these transformations as well as the analysis of the algorithm to determine performance bottlenecks and the dynamic range of data.

The overall impact our customers realize is a time savings of 50 to 75% of schedule for their algorithm implementation.

**Q. In our pre-interview discussions, you emphasized Agility's skill in helping designers working with algorithms to take algorithms written in C or MATLAB and "accelerate" their implementation in FPGAs. Often, these algorithms are "manually" transferred to RTL. Can you explain how your products work with C or MATLAB to improve this process?**

A. Starting from a MATLAB algorithm the first step is to create C-code which is functionally equivalent to the original MATLAB. Agility's [MCS](#) (MATLAB to C Synthesis) automates this

step. Actually, translating even a simple *MATLAB* function to C is fraught with peril—once you're aware of the pitfalls. If you are interested in the details you can download our whitepaper [MATLAB to C Pitfalls](#), but I will quickly highlight a few here:

### ***MATLAB doesn't require definition of data types***

This is a mixed blessing. It makes it very easy to put together a proof of concept, but when it comes time to translate to C, a lot hinges on correctly interpreting the intent of the *MATLAB* designer— as opposed to working off a precise specification.

RESULTING MANUAL EFFORT— The types and sizes of all variables (and possibly memory allocation) must be declared

### ***MATLAB is a vector language***

Translating a vector-based description to a scalar-based description is harder than it sounds. Beyond the obvious fact that the code becomes a lot more verbose, there is the potential to miss data dependencies, which prevent you from simply turning a vector operation into a “scalarized” loop.

RESULTING MANUAL EFFORT – you may need to create several copies of sub-functions for each signature and carefully analyze data dependencies.

### ***MATLAB supports type polymorphism***

Polymorphism means that operations and functions behave differently when called with different types of arguments. For example, a multiplication can be a real multiplication or a complex one.

RESULTING MANUAL EFFORT – you may need to create several versions of the code for a given operation, and several copies of sub-functions for each possible signature

For most algorithm projects just creating a functionally equivalent C model can take weeks of effort and if you are ultimately going to FPGA this is just step 1 of the translation process. With an automation tool like Agility's MCS this time to generate a functionally equivalent C model is reduced to the time it takes to generate the C code – typically minutes.

The next step is to translate the C code into an FPGA. If you don't use high-level synthesis the process is to manually converting the abstract C code into RTL. Making use of high-level synthesis tools like Agility's [DK Design Suite](#) speeds the conversion. I have highlighted three key areas where our C-base approach adds value by saving time.

### ***Faster verification***

Contrasting C vs. RTL for verification is easy, especially abstract C code. Abstraction equates to performance. C-code that is synthesis-ready is less abstract than the original C code, but still is more abstract than the original RTL that would be hand-written or generated by the C synthesis process. The impact is faster turn-around times and more confidence that changes and improvements haven't compromised the functionality. If the tool used also supports hardware-in-the-loop execution with the test bench, designers get even higher performance and confidence that the portion of the algorithm implemented in an FPGA works.

### ***Partitioning***

Algorithm implementation is a balancing act of performance, cost and power trade-offs. Most often a combination of hardware and software is used to implement the algorithm. Using profiling on the abstract algorithm model gives immediate focus on the performance bottlenecks. This information, along with the developer's experience, makes it easier to determine whether an FPGA can help break these bottlenecks. Once identified, defining the interface for the portions of the algorithm that will run on the FPGA must be easy to handle in a good algorithm development environment. This is where using high-level synthesis to create all the necessary interface control logic delivers big time savings and correct-by-construction results over hand developed interfaces and control.

### **Parallelism**

The goal for moving an algorithm to an FPGA is to gain performance through parallelism. Again this is where developers will want a development tool to enable them to quickly direct the compiler to which functions/lines of code are to be executed sequentially and which can be run in parallel. The better the control provided the designer and the faster the turn-around time the bigger the savings in development time and the higher the quality of results.

In Agility's experience with over 1000 algorithm implementation projects, development teams can complete their implementation in less than  $\frac{1}{4}$  the time of hand-coding RTL. As algorithm complexity grows, this will only compound the development time if using conventional hand-coding techniques and create more demand for teams to use tools designed for algorithm implementation like Agility's DK Design Suite.

**Q. Are there any differences between starting with C vs. starting with MATLAB that developers should be aware of?**

- A. Mechanically there is an extra step to generate C from *MATLAB* that isn't required if you start from C. The process to get from C to the FPGA is identical whether you wrote the C or you use MCS to generate the C. The developer will have to add the parallelization and other hardware elements to the C code. This is one of the areas we are exploring adding automation, but this is just an investigation at this time.

**Q. What about your own tools, *Pixelstreams* and *DK Design Suite*, how do these compare with other "high level" design tools *MATLAB* or *Simulink*?**

- A. [\*PixelStreams\*](#) was designed as a solution for quickly implementing high-performance stream-based image processing and by developing the *PixelStreams* library of functions in the *DK Design Suite* environment they are easily parameterized and customized creating a powerful, programmable toolset for image processing.

A *PixelStreams* application is built from a network of filters connected together by streams. Filters can generate streams (for example, a sync generator), transform streams (for example, a color space converter), or absorb streams (for example, video output). Streams consist of flow control, data transport and high-level type information. The data component in turn contains:

- An active flag (indicating that this pixel is in the current region-of-interest).
- Optional pixel data (in 1-bit, 8-bit or signed 16-bit monochrome, 8-bit YCbCr or RGB color).
- An optional (x, y) coordinate.
- Optional video sync pulses (horizontal and vertical sync, blanking and field information for interlaced formats).

Combining all of these components into a single entity gives us great flexibility. Filters for example can perform purely geometric transforms by modifying only the coordinates, or create image overlays by just modifying the pixel data. This gives you a brief high-level view of *PixelStreams* focus from a technology perspective, but the value is once again saving implementation time. For more details you can download our ["Bringing Imaging to the System-Level"](#) whitepaper.

Many imaging applications need to rely on FPGA to meet the performance requirements to process live video and output the results at the 30 to 60 fps rates typically required for video. *PixelStreams* gives the developer a toolbox to build up their image-processing algorithm in FPGA quickly.

As far as comparing to *MATLAB* and *Simulink*, I think the big difference is they aren't targeting a specific vertical segment to provide a tuned solution for that segment and *PixelStreams* is focused on rapid development of streaming video solutions onto FPGA.

**Q. What about the trade-off and choices between implementations in FPGAs and implementation in DSPs? Do you or Agility have any preferences, opinions, or suggestions in this respect?**

A. The choice between FPGA and DSP is often dictated by performance requirements, IO connectivity and availability of reusable IP. For example, IO connectivity relates to architecture so if I am bringing camera input into the system for analysis and the best way to connect the camera is using an FPGA and existing IP for a camera link interface that means my video stream is on the FPGA so it could make sense to do some conversion or filtering of the video while on the FPGA. Similarly, if I start reading data from a disk and using a DSP to process the data, but can't process it fast enough to meet my output speed requirements I would look at adding an FPGA to offload data processing that could be paralleled on an FPGA. In all these cases it is a trade off between development time and functionality/performance. With Agility we provide complete solutions so you can assemble the most efficient system quickly with our FPGA platforms rich with IO and all the necessary IP to drive the IO, the tools to quickly integrate and implement your own design.

**Q. Often, developing algorithms isn't the best use of an engineer's time. He might want to purchase tools to help him do it better, or he might want to "outsource" this piece of his design. Recently your company started a focus on design services. Tell us about your services.**

A. Typically algorithms represent differentiating intellectual property for customers giving their products sharper images, faster response times, cleaner audio, etc. Getting these algorithms implemented requires a different skill set than developing the algorithms. Because Agility's products are built for getting from algorithm to implementation quickly our technical team has become expert users of the products making them real power users of the technology. This combined with their backgrounds in image and signal processing gives us a valuable team of resources that can deliver a turnkey solution to customers and leave behind the tools and training so the customer can modify and evolve their designs rapidly too.

**Q. Finally, tell us a little about your pricing models both for your products and your services. Can you give us a ballpark idea for how much things cost as well as the process of engagement?**

A. The pricing model is very broad representing the broad spectrum of solutions required for customers. For example a low-end simple prototyping system starts as low as \$1,000 and in contrast we have done programs that the solution including services was \$250,000. What is different about Agility is we have the ability to configure solutions to meet customer's needs. So it is the engagement process that is really the focus, we meet with customers to understand what they need to accomplish and can outline options from complete turnkey solutions to product and training.

**Q. Thank you for this interview.**