

# Hardware/Software Partitioning from Application Binaries

A unique approach to hardware/software co-design empowers embedded application developers.

by Susheel Chandra, Ph.D.  
President/CEO  
Binachip, Inc.  
[schandra@binachip.com](mailto:schandra@binachip.com)

Computationally intense real-time applications such as Voice over IP, video over IP, 3G and 4G wireless communications, MP3 players, and JPEG and MPEG encoding/decoding require an integrated hardware/software platform for optimal performance. Parts of the application run in software on a general-purpose processor, while other portions must run on application-specific hardware to meet performance requirements. This methodology, commonly known as hardware/software partitioning or co-design, has caused an increasing number of software applications to be migrated to system-on-a-chip (SOC) platforms.

FPGAs have emerged as the SOC platform of choice, particularly in the fast-paced world of embedded computing. FPGAs enable rapid, cost-effective product development cycles in an environment where target markets are constantly shifting and standards continuously evolving. Several families of platform FPGAs are available from Xilinx. Most of these offer processing capabilities, a programmable fabric, memory, peripheral devices, and connectivity to bring data into and out of the FPGA. They provide embedded application developers with a basic hardware platform on which to build end applications, with minimal amounts of time and resources spent on hardware considerations.

## DSP applications add another dimension of complexity by including MATLAB source code in addition to C/C++ and assembly language. Clearly, any tool that does not support multiple source languages will fall short.

### Source-to-Hardware Tools

Several tools on the market allow you to translate high-level source descriptions in C/C++ or MATLAB into VHDL or Verilog. Most of these tools fall into the category of “behavioral synthesis” tools and are designed for hardware designers creating a hardware implementation for a complete algorithm. These tools require you to have some intrinsic knowledge of hardware design and to use only a restricted subset of the source language conducive to hardware translation. In some cases they also require you to learn new language constructs. This creates a significant learning curve, which further pushes out time to market.

More recently, some companies have introduced tools for embedded application developers. These tools promote a co-design methodology and allow you to perform hardware/software partitioning. However, they suffer from some of the same limitations of behavioral synthesis tools – you need to learn hardware-specific constructs for the tool to generate optimized hardware. This imposes a fundamental barrier for a tool targeted at embedded application developers, who in most cases are software engineers with no or limited knowledge of hardware design.

Another basic requirement for tools intended for application developers is that they support applications developed in multiple source languages. An embedded application of even moderate complexity will comprise parts written in C/C++ (calling on a suite of library functions) and other parts written in assembly language (to optimize performance). DSP applications add another dimension of complexity by including MATLAB source code in addition to C/C++ and assembly language. Clearly, any tool that does not support multiple source languages will fall short.

A new unique approach to hardware/software co-design is to start with the binary of the embedded application. It not only over-

comes all of the previously mentioned issues but also provides other significant benefits. A new commercial tool called BINACHIP-FPGA from Binachip, Inc. successfully uses this approach.

### Partitioning from Binaries

The executable code or binary compiled for a specific processor platform is the least common denominator representation of an embedded application written in multiple source languages. Combined with the instruction set architecture of the processor, it provides a basis for extremely accurate hardware/software partitioning at a very fine grain level. The close-to-hardware nature of binaries also allows for better hardware/software partitioning decisions, as illustrated in the following examples.

### Bottlenecks Inside Library Functions

One obvious consideration for hardware/software partitioning is computational complexity. Moving computationally intense functions into hardware can result in a significant performance improvement. Any embedded application typically relies on a suite of library functions for basic mathematical operations such as floating-point multiply/divide, sine, and cosine. Some of these functions are fairly compute-heavy and could be called millions of times during the normal execution of an application, resulting in a bottleneck.

Moving one or more of these functions into hardware could easily provide a 10x-50x speedup in execution time, depending on their complexity and how often they are called. Any co-design tool operating at the source-code level will completely miss this performance improvement opportunity.

### Architecture-Independent Compiler Optimizations

Most C/C++ or other source language compilers are very efficient at architecture-independent optimizations such as con-

stant folding, constant propagation, dead code elimination, and common sub-expression elimination. For example, consider this simple snippet of pseudo code:

```
int a = 30;
int b = 9 - a / 5;
int c;

c = b * 4;
if (c > 10) {
    c = c - 10;
}
return c * (60 / a);
```

Applying the compiler optimization techniques discussed earlier, this pseudo code can be reduced to:

```
return 4;
```

A co-design tool operating at the binary level can leverage these compiler optimizations to further improve the accuracy of hardware/software partitioning. Tools that work at the source level must either perform these optimizations internally or leave some area/performance on the table, resulting in a sub-optimal co-design.

### Legacy Applications

For most popular processors such as PowerPC™, ARM, and TI DSP, a host of applications are already out in the field. These applications would benefit from a port to a faster FPGA platform with a co-design implementation. In many of these situations the source code is either not available or is written in an obsolete language. If the source code is available, the original developers may no longer be associated with the project or company; therefore the legacy knowledge does not exist. In such scenarios a tool that works from the binary can prove to be an invaluable asset.

### Designing with BINACHIP-FPGA

The BINACHIP-FPGA tool is designed for embedded application developers and

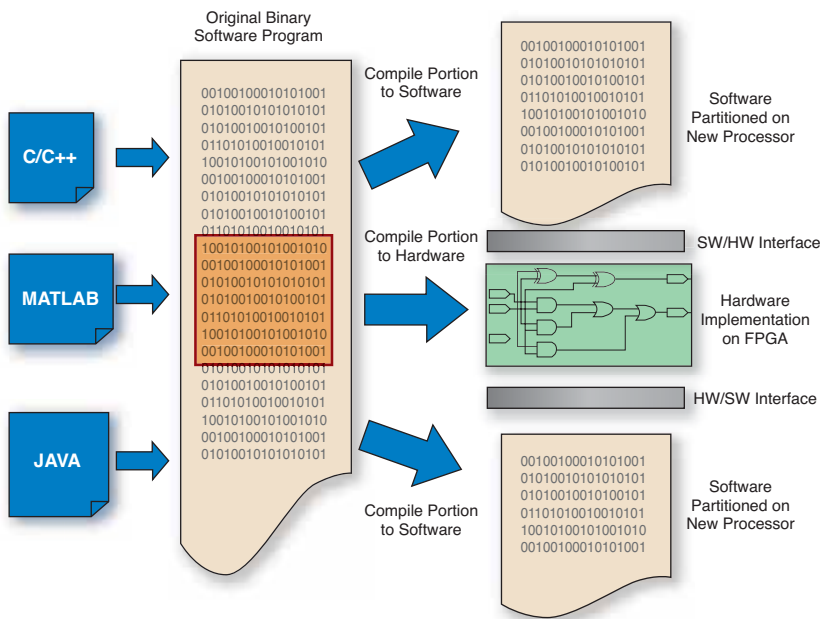


Figure 1 – Transforming a pure software implementation into a co-design

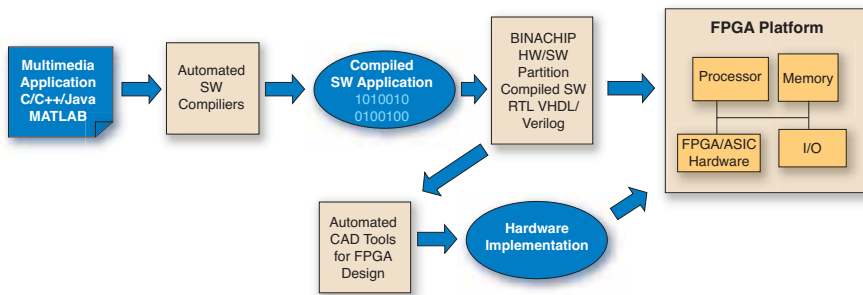


Figure 2 – Design flow using BINACHIP-FPGA

allows them to leverage all of the features of the FPGA platform, without having any intrinsic knowledge of hardware design or needing to learn a new source language. The transformation performed by the tool in going from pure software implementation to a co-design is conceptually illustrated in Figure 1.

An embedded application written in a combination of C/C++, MATLAB, and Java is compiled into a binary executable for a target platform. The bottleneck is identified based on profiling data or user input (see the orange box in Figure 1). The tool automatically generates hardware for these functions, which is mapped into the

programmable fabric of the FPGA. All hardware/software communication interfaces are also generated by the tool. Calls to these functions are now computed in hardware and control is returned to the software when the computation is complete. The tool then outputs a new executable that runs seamlessly on the target platform.

BINACHIP-FPGA is also capable of performing advanced optimizations such as loop unrolling, which allows you to make area/performance trade-offs. You can also exploit fine-grain parallelism by using advanced data scheduling techniques like as-soon-as-possible (ASAP) or as-late-as-possible (ALAP) scheduling.

## Empowering Application Developers

BINACHIP-FPGA empowers embedded application developers to create advanced applications on FPGA platforms with minimal intervention or interaction with hardware designers. Figure 2 shows a typical design flow for the tool.

After the system architect has made a decision on which platform to use, the application developers start writing software and go through the usual task of compiling, debugging, and profiling the code. BINACHIP-FPGA fits into the flow once the application developers have debugged the basic functionality. After feeding the compiled application and profiler data into the tool, you have the option to manually guide the tool and select which functions should be mapped into the hardware. The tool generates the hardware and reconstructed binary with appropriate calls to the hardware.

You can then simulate the resulting implementation in a hardware/software co-design tool using the bit-true test benches generated by BINACHIP-FPGA. The RTL portion is mapped into the FPGA fabric using Xilinx Synthesis Technology (XST) and the ISE™ design environment. The application is now ready to run on the target platform.

## Conclusion

Hardware/software co-design tools promise to empower embedded application developers and enable the large-scale deployment of platform FPGAs. To deliver on that promise, these tools must assume that users will have minimal or no hardware design knowledge. You should not be required to learn new language constructs to use the tools effectively.

BINACHIP-FPGA is the first tool in the market that uses a unique approach to enable application developers to fully leverage the FPGA platform and meet their price, performance, and time-to-market constraints.

For more information on hardware/software co-design or BINACHIP-FPGA, visit [www.binachip.com](http://www.binachip.com).