Component-Based support for FPGA and DSP

Mark Hermeling (Zeligsoft, Gatineau, QC, Canada; mark@zeligsoft.com)

ABSTRACT

Until now, Software Defined Radio (SDR) standards have focused on General Purpose Processors. Integration with DSP and FPGA processors has been done mostly manually and in a non-systematic manner. This is about to change with different standards for DSP and FPGA integration being proposed. This paper provides a high-level introduction of the various standards and highlights the differences between them. The paper focuses on the standards from a tooling and automation viewpoint.

1. INTRODUCTION

SDR using the Software Communications Architecture (SCA) has been made significantly easier through the advent of modeling and generation tools. These tools enable experts to build waveforms and platforms faster and with better quality. These tools have also lowered the amount of expertise required to build waveforms, which has allowed new projects to get started quicker.

This is commonplace with standards: the early stages are challenging, but later stages become easier as vendors adopt the standard and COTS products become available, which automate tedious tasks and offload much of the complexity from the developers.

One specific area that has changed very little so far is the Specialized Hardware Processors (SHPs), also known as FPGAs and DSPs. SHPs are used to implement the lower layers (physical layer) of a waveform. The reason for little change is that there is a lack of standardization of how software is developed and deployed to these processors. Significant work has been put into facilitating development for SHPs in past years and this is about to bear fruit. Different standards have been developed and different scalable, robust COTS technologies are available now or will be available soon.

This paper introduces the different ways that SHPs can be used in a Software Defined Radio platform. Section 4 introduces the different patterns that developers have been using and that vendors have proposed. Each of the patterns is then discussed with respect to portability, ease-of-use and performance. The inner workings of the patterns are not discussed in detail; rather, the benefits and disadvantages of the patterns are described. This provides the reader with an understanding of the relative merit of each pattern.

Special attention is paid to how automation can be used together with each pattern to lighten the load for the software engineer. The goal of automation and technology is to help the engineer to focus on writing better software faster.

This is even more so when working with SHPs, as the high-level software architecture is typically laid out by software engineers, while hardware engineers have to create the signal processing functionality. Automation of integration would make it easier for these two groups of engineers to work together, without restricting their creative capabilities.

The paper begins with a discussion on requirements that the SCA puts on software components. This is followed by a short introduction of the different technologies available. After that we take an objective look at the portability, ease-of-use, performance and automation of each of the technologies.

2. COMPONENT REQUIREMENTS

SCA-compliant systems consist of a flexible platform that can be loaded with multiple radio personalities. Each personality is a piece of software that implements a waveform such as SINCGARS, WNW, SRW and so forth. These personalities are also known as ‘waveforms’ or ‘applications’.

Each waveform contains software that needs to be loaded into the heterogeneous mix of processors that comprises the radio platform. A waveform is divided into a number of components, which are independently downloadable pieces of functionality. The SCA describes what the requirements are that these components have to meet to be considered SCA-compliant. These requirements fall in one of three categories: POSIX compliance, CORBA capable, and the ability to support the SCA Base Core Framework interfaces (CF::Resource and so forth).

These requirements are put in place to provide portability of the component. Components are typically written and tested for a particular real-time operating system (RTOS). Adherence to the requirements stated in the SCA standard makes it easier (but not trivial) to recompile and test a component for a different operating system (from VxWorks to INTEGRITY or vice-versa, for example).

The standard, however, does not refer to SHPs at all. Waveforms typically contain content that needs to be loaded towards DSP and FPGA processors. SHPs do not support POSIX and are often not CORBA capable. This provides the following main challenges for dealing with SHPs:

- Functionality written for one SHP is not easily portably to another (for example, Xylinx FPGA to Altera FPGA)
- It is difficult to control functionality on SHPs (change parameters, send start/stop messages and so forth)
- It is difficult to send and receive data to and from functionality on SHPs

The technologies that we are about to take a look at are designed to resolve all or part of these problems.

3. AUTOMATION

Automation offloads tedious responsibility from the engineer to tooling. The tool can guide the engineer through visualization, validation, and generation by using automation.

Offloading tedious responsibilities means that the engineer can focus on the actual functionality of the components, rather than the nitty gritty detail required by standards and communication busses.

This naturally begs the question as to how much of the work can be automated. For all the patterns discussed in this paper, visualization and validation are natural candidates. They provide the engineers with an easy way to express and check their design. Generation can be divided into two main categories: the domain profile and compilable source code.

Domain profile generation for all patterns is straightforward. The source code required is different for each pattern. This paper looks at that in more detail in the following sections.

4. AVAILABLE TECHNOLOGIES

As stated above, the SCA standard has so far paid little attention to SHPs. Projects have been using a workaround to access functionality running on FPGAs. Functionality—for example, a Digital Down Converter (DDC)—is downloaded to the FPGA and an DDC_Adapter (or Proxy) is run on a General Purpose Processor (GPP). The adapter is the intermediate between the content on the FPGA and the rest of the waveform. The first pattern that is covered is the use of adapters.

The second pattern is the Hardware Abstraction Layer for Connectivity (HAL-C). HAL-C also uses the adapters, but does provide a standardized way of managing connections between the functionality on the SHPs.

The third pattern is the Component Portability Standard (CPS). CPS uses adapters as well, but the adapters are standardized to the extent that the developer does not have to manually develop them anymore.

The fourth and last pattern extends the CORBA bus to the SHPs. The functionality on the SHPs is able to communicate through CORBA (GIOP) messages directly.

5. ADAPTERS

Adapters are a flexible, powerful, and efficient way of managing functional content on SHPs. An adapter is an SCA-compliant, CORBA-capable software component that runs on a GPP and provides access to functionality on an SHP.

Take, for example, a platform that consists of an FPGA and a GPP processor as shown in ***Figure 1***. The platform contains two logical devices, which implement the capability to load and execute software on the physical FPGA and GPP. Note that the platform abstracts whether the physical devices are Xylinx or Altera FPGAs, as well as whether the GPP is a PowerPC or an X86.

Figure 1: Platform Model

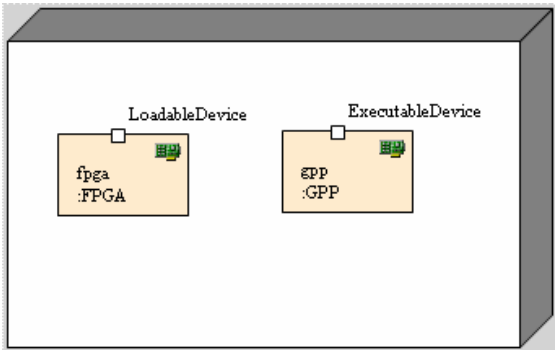


Figure 2: Waveform model for Adapter or HAL-C

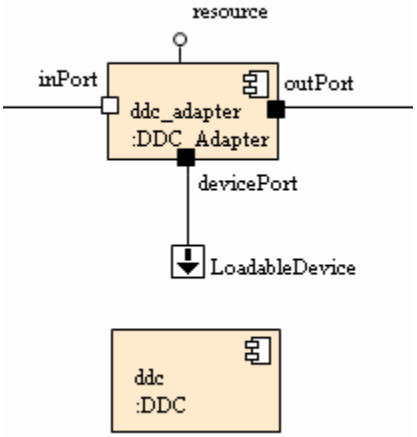


Figure 2 shows a section of a waveform. A number of observations can be made by looking at ***Figure 2***. First of all, the DDC component does not support the CF::Resource interface. This is not a CORBA capable component. Indeed, this component will execute on the FPGA and hence will not directly receive CORBA messages. Secondly, the DDC_Adapter has the Resource interface: it has an inPort to receive data, an outPort to send data, and a devicePort. This last port talks to the device that loaded the ddc component (through the SCA DeviceThatLoadedThis-ComponentRef construct). The DDC_Adapter is the front that manages all administrative tasks for the DDC component.

Figure 2 seems to indicate that the DDC_Adapter will receive data through CORBA and this is certainly one of the possibilities. The adapter could receive data through CORBA, and write it to FPGA memory. The FPGA operates on it and sends the data back. The disadvantage of this is immediately clear: the performance will not be the most optimal.

The SCA standard states that data should flow through CORBA, but other transports can be used for performance reasons. However, the connection needs to be initiated through the SCA interfaces, even if other transports are used.

Hence the connection to the inPort and from the outPort could be used by the DDC_Adapater to configure a high-speed bus that connects the FPGA to other parts of the hardware. The implementation of this depends on the hardware and the SCA BSP delivered by the hardware provider. This also implies that the implementation of the sender and receiver depends on the SCA BSP delivered by the hardware vendor. Care should be taken to limit this dependency as much as possible.

The devicePort has an interface of CF::PropertySet and is connected to the FPGA device that is executing the DDC component. This allows the adapter to discover details about the FPGA in use. These details are often needed by the adapter to make system calls to write data to the FPGA's memory to configure the DDC and to pass data to and from the DDC.

5.1 Portability

The portability of this particular solution is fairly poor. The adapter is typically not POSIX compliant; it uses system calls to communicate with the DDC. The adapter is thus tightly coupled to the platform in use; most of this dependency is towards the SCA BSP. It is also tightly coupled to the DDC.

The DDC itself is written for the particular FPGA in use and usually also has platform IP blocks integrated in it.

Hence neither the adapter nor the DDC itself are portable. However, using the adapter ensures that the other parts of the waveform are still portable.

5.2 Ease-of-Use

The use of adapters implies that the developer has a lot of extra work to do to make the DDC work in an SCA setting. This is also fairly detailed and expert work, as the developer has to be aware of a lot of the inner workings of the SCA, the platforms SCA BSP and the workings of the FPGA code.

One of the most significant pieces of work for the developer is to make it such that connections between the components can be created based on the contents of the waveform description (SAD). Connections require location transparency, which comes at a cost. Waveforms that use the straight Adapter pattern often omit this location transparency. The HAL-C pattern in the next section improves on this.

The second major piece of work is to ensure that the upper layers of the waveform are not affected by the Adapter pattern. It is important to ensure that the upper layers of the waveform are portable, even if the lower layers are not.

5.3 Performance

The performance of this solution can be very good. Typically, the adapter is not involved in data transport, but all data is transported over high-speed connections that the FPGA is connected to. This allows the developer to avoid any overhead and get the most out of the hardware.

5.4 Automation

The Adapter pattern can not be automated for the general case since there is no standard to follow that dictates what the code for the adapter and the SHP content should look like.

However, automation can be achieved for a particular platform. That is, generation of the adapter as well as the FPGA component code can be automated for a particular in-house or COTS platform. This would make that particular platform easier to use.

Automation for a platform could completely generate the source code required for the implementation of the adapter and could generate skeletons for the code that needs to execute on the SHP. All the developer would have to do is extend the skeleton with functional signal processing code.

6. HARDWARE ABSTRACTION LAYER FOR CONNECTIVITY

The Hardware Abstraction Layer for Connectivity (HAL-C) [1] pattern does not look any different than the Adapter pattern, where the application and platform design is concerned. That is, the platform and application external interfaces and connections look exactly as shown in Figure 1 and Figure 2. The difference is in the internals of the adapter and the DDC. The HAL-C standard is documented in SCA change proposal 237 [2].

HAL-C addresses several portability problems with the portability of the Adapter pattern. It describes a communication API to isolate the software from the communication mechanisms available in the hardware. The idea is that each component has an internal functional core that is wrapped in libraries (for the C/C++ language) and IP blocks (for VHDL) that interface the functional core with the transports.

HAL-C describes how to set up connections between components; it does not describe the data transport over these connections. That is left to the designer. The result of this is that HAL-C delivers greater portability of waveforms between platforms, but not portability of components within waveforms since components are tightly coupled due to the way they send data over the connections.

6.1 Portability

Portability of the waveform from one platform implementing HAL-C to another platform implementing HAL-C would increase significantly. However, components are tightly coupled; hence HAL-C does not make it easier to build a library of components that can be re-used across waveforms.

The thing to keep in mind with respect to portability at the FPGA level is that this is not really achievable according to many practitioners. The functional code running on FPGAs is very susceptible to differences in timing, for example. Hence, portability can never be guaranteed, but it can be improved.

Portability can be further improved through the use of model-driven development techniques for functional content. Tools such as MathWorks Simulink used in combination with Xilinx System Generator allow a developer to include IP blocks that manage the HAL-C implementations, together with blocks describing the functional behavior for components. This seems like an interesting approach that can further increase portability of code across platforms.

6.2 Ease-of-Use

HAL-C pushes more work to the SCA BSP, namely the implementation of the connection handling. This means that the individual developer does not need to manage this anymore. HAL-C provides location transparency and makes it easier for the developer to create connections between components, regardless of where these components are deployed. The developer can build the functionality on top of the connection handling.

The developer still needs to decide on a way to encode/decode the data that needs to be sent over the connection. HAL-C does not cover this.

The developer would also need to write the adapters. Many developers write one single adapter for an FPGA or DSP. The FPGA and DSP might contain multiple different pieces of functionality, which would normally go into multiple components. However, DSPs and FPGAs often only take one image and hence the components are combined into a monolithic load at compile time. This is then managed through a single adapter. This does limit the deployment and portability options of the waveform.

6.3 Performance

HAL-C does introduce some infrastructure code to manage the connection. However, this code is relatively light and does not typically introduce a huge performance overhead.

6.4 Automation

HAL-C defines an API. Hence, it is possible to generate implementation skeletons for components. The skeletons would contain all the code that is required to use the components in an HAL-C infrastructure. The user can add functional code into those skeletons.

Adapters can not be directly generated based on the HAL-C specification only. An adapter manages three different types of requests: properties, connections, and start/stop commands. HAL-C does not completely describe how properties and start/stop commands should be implemented; however, this could be done with a small extension to the HAL-C standard.

7. COMPONENT PORTABILITY STANDARD

The Component Portability Standard (CPS) [2] builds on top of the HAL-C standard. The CPS goes beyond HAL-C and standardizes more aspects of a component's life-cycle.

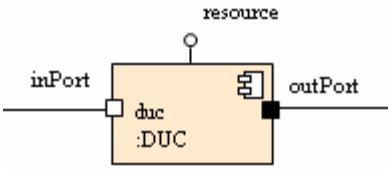
CPS standardizes not only creating connections between components, but provides components with a method-based connection on its ports, where HAL-C provides a stream-based interface. CPS further provides component life-cycle and configuration options like start, stop, configure and query.

In other words, the CPS standard treats components on SHP processors similar to components on GPP processors.

The CPS achieves this by defining a standardized adapter for every SHP component. The adapter translates CORBA messages to the SHP component. The adapter is standardized, which means the user does not have to develop a custom adapter for every component. However, the standard is flexible enough to allow users to write their own adapters if required for performance reasons.

The platform for a CPS application looks as depicted in ***Figure 1***; however, the application model looks completely different and is depicted in ***Figure 3***. The main difference is that an adapter is no longer present. As mentioned above, CPS handles adapters automatically and the user does not need to worry about them anymore.

Figure 3: Waveform model for CPS



One of the main benefits of this is that the waveform model no longer depends on whether a CPS component is used. The DUC component could have multiple implementations: one for a DSP, one for an FPGA, and one for a GPP. This type of flexibility is not achievable through the adapter or HAL-C patterns.

The CPS standard puts much more responsibility at the SCA BSP level. This responsibility has been moved from the individual component level to the infrastructure level. This also implies a higher level of overhead. CPS does try to minimize the amount of overhead as much as possible and enables optimizations such as zero-copy communication whenever possible.

CPS is the first standard that uses a componentized approach to running functional content on SHPs. An application using CPS can use multiple components in the physical layer of a radio. CPS ensures the components are deployed and connected correctly. The CPS approach can be much more granular than the adapter of HAL-C approach. This provides more options during deployment time, but also stresses the SCA CoreFramework and the hardware more to deploy the waveform correctly.

7.1 Portability

Portability with CPS is increased, though the problem with timing at this low level has not been removed. CPS abstracts communication to the operation level. In HAL-C a component would be fed a stream of data, but the component itself would have to 'know' what that data means. This implies that there is coupling between the receiver and sender of the data. In CPS, the component would be fed a stream of operations with parameters. The receiver is now decoupled from the sender: they are only related through the interface.

This means that CPS components are re-usable between waveforms, whereas HAL-C components are not. This helps engineers to build a library of components and re-use them in different waveforms.

7.2 Ease-of-Use

The fact that CPS components receive operations with parameters also makes it easier for developers to write code for these components. The amount of code that needs to be written has been reduced, since the marshalling and un-marshalling code is already included in the CPS infrastructure code.

7.3 Performance

The CPS infrastructure layer performs more work than the HAL-C infrastructure layer. The work that the CPS infrastructure performs is functionality that the developer typically would have to write himself for HAL-C. The functionality has to be included somewhere.

The implication here is that a developer has more control over the work done in the critical path of a component in HAL-C. Hence HAL-C can be optimized more than CPS. However, in the general case, these two patterns would have comparable performance.

7.4 Automation

CPS as a standard has excellent potential for automation. Every component has an interface that is expressed in the visual model. This interface completely defines what messages the component can receive and hence what the component implementation should look like. This holds for both DSP as well as FPGA processors.

A code generator can completely generate the skeleton for the component and the user can extend it.

8. CORBA

CORBA has been used extensively on GPPs in the past in systems that range from embedded control to radar to consumer electronics. Over the years CORBA has proven that it can provide excellent performance in embedded systems. However, a lot of people are still not convinced that CORBA is the ideal solution to meet the stringent real-time requirements that SDR systems have to adhere to.

Fielded SCA/SDR radios have proven that these fears can be put to rest when using high-performance, low-foot print CORBA ORBs. However, using a high-performance ORB itself is not sufficient: care must also be taken to design the system properly so as not to inadvertently introduce performance bottlenecks.

Since GPP components already use CORBA, it makes sense to try and extend the CORBA bus to the SHPs. Extending the CORBA bus to the SHPs is the easiest way to communicate from the GPPs to the SHPs. However, it might not be the best way to communicate between SHPs or within a particular SHP class (cases in which both HAL-C and CPS shine).

A DSP processor is different from a GPP, both in execution model and in programming languages. CORBA ORBs are available on DSPs and programming these DSPs is not very different from programming a GPP.

An FPGA processor, however, is completely different from a GPP, both in execution model and in programming languages. CORBA ORBs for FPGAs are just now becoming commercially available. Practitioners have a lot of questions on both performance and programmability of components that use CORBA on FPGAs.

8.1 Portability

CORBA is a standard managed by the OMG. It includes both a messaging standard and a standard programming API. This API has been defined for GPP and DSP processors, the latter through the CORBA/*e* standard. These standards ensure portability between devices that can run CORBA or CORBA/*e* compatible ORBs.

This is different for FPGA processors. There is no standardization of CORBA at all; hence it is likely that a component implemented for an FPGA ORB of vendor A will not work with the FPGA ORB of vendor B.

8.2 Ease-of-Use

Writing CORBA code for a DSP is not very different from writing CORBA code for a GPP. Once a developer has mastered CORBA he will be able to apply this knowledge to the world of DSPs effortlessly.

The same cannot be said for the world of FPGAs. A typical FPGA engineer knows how to manage streams of data and how to buffer the data and send it into the proper signal processing algorithms. CORBA messaging is significantly different compared to data streams. CORBA messages are sent to a particular object, are related to a particular operation, and contain data as payload. These concepts are often new to FPGA engineers who usually have a hardware background, as opposed to a software background.

Sending data to an FPGA that supports CORBA will be very easy; however, writing functionality for an FPGA that uses CORBA would require a shift in thinking for the FPGA engineers. The technology is too new to decide whether this shift in thinking is easily overcome.

8.3 Performance

CORBA messaging can be made very efficient. Typical CORBA messaging uses the CORBA GIOP protocol over TCP/IP. The latter adds significant overhead. However, it is possible to run pure CORBA communication directly over, for example, a RapidIO bus. SHP processors can be directly connected to that bus and hence communication can be done efficiently.

This is certainly true for communication between a GPP processor and an SHP processor. The data has to go over the bus and GIOP is as good a protocol as any. However, SHP processors frequently have to communicate together. From FPGA to FPGA, FPGA to DSP, DSP to DSP

and so forth. Even worse, communication also happens within an FPGA or within a DSP.

CORBA messaging is not the ideal communication bus in these situations. The encoding and decoding of data into the GIOP protocol is often not needed and hence slows down communication. Some other form of communication would be better suited.

8.4 Automation

Different ORB vendors provide CORBA on SHP processors. These ORB vendors specify exactly what the code should look like for a specific component; hence this pattern can be automated without any problems, to the same extent as CPS: Automation generates skeletons; the user provides the functional code.

9. CONCLUSION

This paper has presented four different standards and has discussed many advantages and disadvantages for the use of these standards. The one question on people's mind is: "Which standard will be the one used in the future?"

This question is impossible to answer. All the standards have good and bad qualities. Currently, projects are using a mixture of all four standards. The technology to extend the SCA into the SHPs is too young to declare a clear winner. The only pattern that has been extensively used in fielded systems is the Adapter pattern, with HAL-C close on its heels.

CPS is not currently available on COTS boards and CORBA on SHP is just available commercially.

The coming years will tell which standard people prefer and what standard(s) the JPEO will include in future revisions of the SCA. This process will be impacted by the available automation for the standards. COTS vendors like Zeligsoft are working hard on this.

A final conclusion is difficult to give in the context of the previous paragraph. The recommendation for now is that projects should examine their needs, examine what technology is available to them, evaluate the patterns and make a sound decision based on the criteria discovered.

If all else fails, the adapter method is tried and proven and will work on any platform. It is very well possible to update a waveform from an Adapter pattern to HAL-C, CPS or even CORBA at a later stage in the game.

11. REFERENCES

[1] "Specialized Hardware Supplement to the Software Communication Architecture (SCA) Specifications", JTRS-5000 SP, V3.0, 27 August 2004

[2] "Extension for component portability for specialized hardware to the JTRS Software Communication Architecture (SCA) Specification, V3.1x, 20 January 2005