Waveform Description Language (WDL) for Software Radios

Robert Prill / Andrew Comba
BAE SYSTEMS Communications, Navigation, Identification (CNI)
150 Parish Dr,
Wayne, N.J. 07474-0932 USA
robert.prill@baesystems.com

ABSTRACT

Waveform Description Language (WDL) was invented to ease the process of porting legacy and/ or new radio waveforms to Programmable / Software Radios. WDL has two primary requirements; 1st it is to provide a rigorous executable behavioural description of waveform signal structures that is unambiguous and yet independent of any particular end item software radio architecture. The 2nd requirement is that the behavioural specification provides a path to automatic code generation for GP's, DSP's, and FPFG's and that the Generated code be tested against the behavioural model.

KEY WORDS

UK/MOD PDR, Software Radio, ,Executable Behavioural Waveform Description Language (WDL), Canonical Representation, Automatic Compilation using COTS Tools, Initial UK Target Radio' are: SATURN and, BOWMAN VHF, JTRS Applications

INTRODUCTION

The work described herein on Waveform Description Language (WDL) was performed by BAE under a contract from UK MOD and Managed by QinetiQ. The WDL process is aimed at eliminating the ambiguities in textual specifications by rigorously defining waveforms in an executable format that is neutral to Software Radio Manufacturers, such that COTS tools can be used to automatically compile the behavioural description to run in their particular Software Radio Implementation (Usually a mixture of GP's, DSP's and FPGAs). The key to WDL process is threefold:

The notion of defining a Class Structured set of OO Radio Components

The notion of Canonical Radio Model Templates

The notion of a web-based Radio Component data base

In the following sections we shall identify exactly where in a radio model WDL was applied, we shall describe the WDL process of decomposing Radio Requirements into Components belonging to Canonical Radio Template, we shall describe the process of generating a behavioural executable abstraction of a radio, and finely the process of generating Radio Implementation Code for a variety of different Software Radio Architectures from the behavioural WDL modelling code.

TECHNICAL DISCUSSION

In this section we shall define where WDL fits into the Programmable/Software Radio generic or virtual radio architecture and exactly where in the OSI Model WDL was applied.

Fig. 1 depicts a typical virtual radio architecture and how it applies to the SRD (Software Defined Radio) architecture defined and recommended by some 15 nations under the SDR Forum.

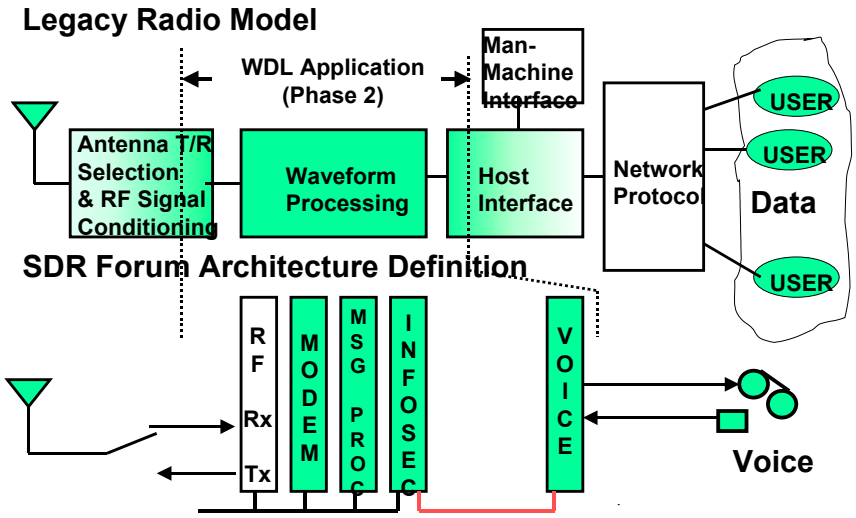

Figure 1- Virtual Radio/SDR

Fig. 1 (top) depicts a generalized virtual radio model including a number of networked users all sharing the radio's Waveform Processing, and RF & Antenna resources. Fig. 1 (bottom) depicts the functional relationship of the SDR framework to the virtual radio depiction and where WDL was applied.

Fig. 2 depicts the 7 layer OSI model, where WDL was applied to the SDR Forum's Software Defined Radio functional partitioning, and what was recommended to be "encapsulated".

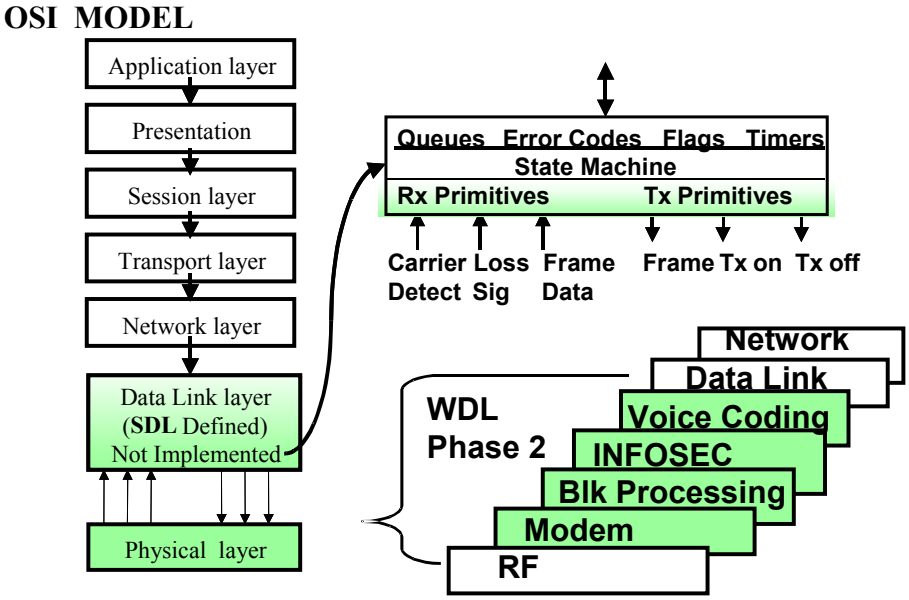

Figure 2…OSI Model and where WDL was applied

As shown WDL was recommended for RF control specifications, virtually all Modem and Black Processor functionality (except the network functions) that are assumed to be encapsulated as embedded "C" code. and non network portions of the Red Processing areas of the SDR model.

Restricting WDL to the non-networked functions is not a terrible restriction or limitation. Network Protocols are usually developed by large consortiums or corporations taking tens of man-years of development effort and are thus sometimes proprietary and need to be licensed. Furthermore, the Data Link, Network, and higher OSI layers are being addressed by JTRS JPO in the JTRS SCA. For the above reasons it makes perfect sense to encapsulate the Data Link and Network layers and interface to them with simple API's….The top portion of figure 2 depicts the functionality of a typical Data Link layer API and the listing below summarises the simplistic functionality of an API interface.

Transmit API	**Receive API**
> Request to Transmit	> Carrier Detect
> Frame of Tx Data	> Get Rx Data
> End Transmit	> Loss of Carrier

With this philosophy… interfacing the Data Link and Network Layers to the physical hardware layer via simple API… a major amount of work can go on in parallel with little or no required interaction between vastly different technology areas.

THE WDL PROCESS FROM BEGINNING TO END

A birds-eye view of the end to end process of WDL is depicted in Figure 3. As shown, conventional specifications that describe a particular Waveform are decomposed into the OSI Layered model of a radio. The physical layer requirements where WDL was applied, are forced into **Signal Path, TSEC, and Orchestration** WDL Templates. The Templates are Placeholders, into which the Executable Transformation Function, that specifies how the Component is to operate on input files to produce the desired output files is specified in a behavioural executable manner.

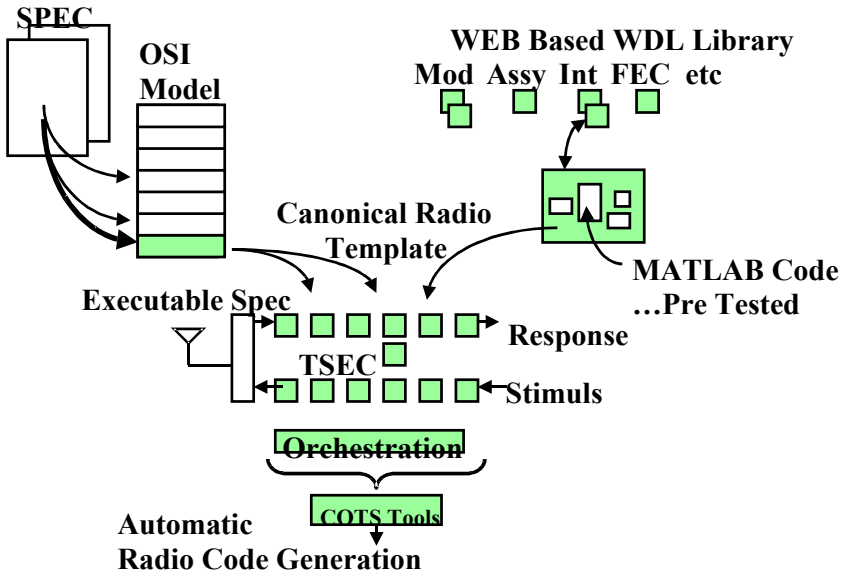

Figure 3… The WDL Process / Beginning to End

The Component depiction (magnified) is at the heart of the WDL process. Components are chosen to describe radio functions like Voice Coders, Encrypters, Message Assemblers, Interleavers, FEC devices, Sync Appenders, Line Coders (Serializers), Modulators, etc and their companion Rx counter parts. As Components are developed they are put into parametric form and entered into a Class Structured OO web-based database for reuse.

THE DETAILS OF THE WDL PROCESS

Figure 4 takes the Birds eye view of the WDL process outlined above one level lower in detail. It depicts the kinds of Radio Components that the textual specs are partitioned into. Some components bear a one to one correspondence with the original specifications, others have to be derived and created.

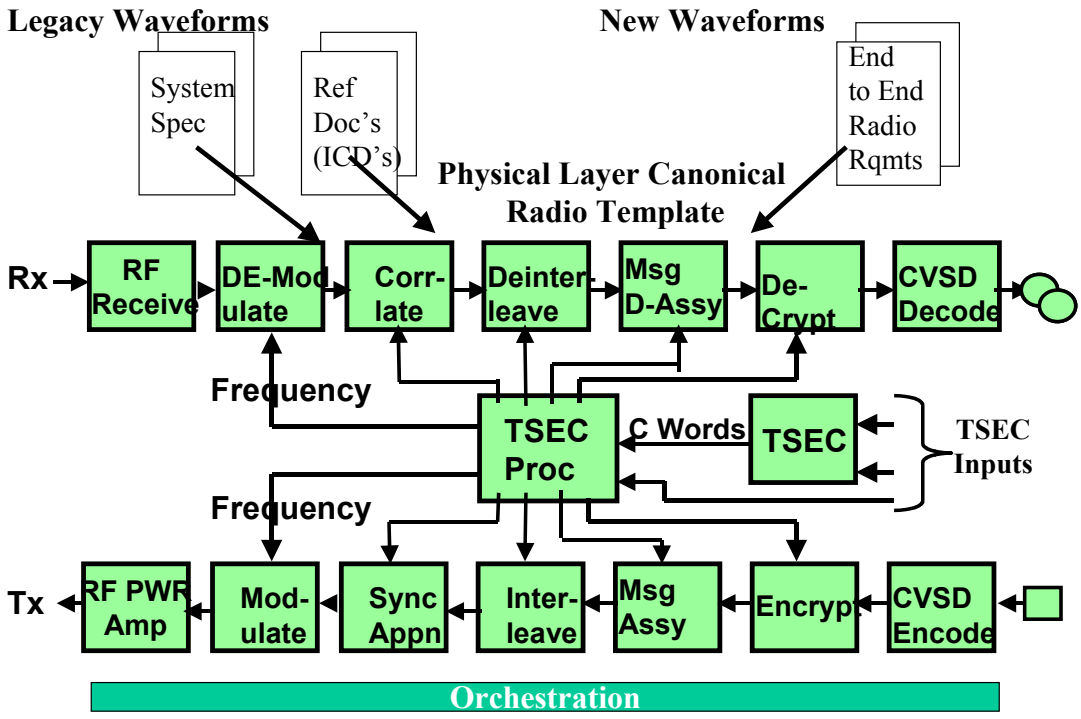

Figure 4…The WDL Templates

As implied there are three main functional Templates into which the original specifications are decomposed into, Signal Path Components, TSEC Path Components, and Orchestration Components.

The Signal Path for a particular radio might consist of the set of components identified in the referenced block diagram. Some of them are well known in the radio world, (CVSD, Encrypt, Interleaver, Modulation) others were derived (Framer, Message Assembler, Sync Appender) from the original textural specifications.

The TSEC and Orchestration Templates (Not shown in their expanded form) are also highly regularized and their structures can be reused from radio system to radio system.

COMPONENTS AND COMPONENT DATA SHEETS

At the heart of WDL is the component. This is the behavioural entity from which the physical layer of a radio system is specified. All Signal Path Components are specified in terms of Input Files, Transformation Function, and Output Files. It is assumed that Input Files are filled with a convenient amount of data (a Packet, a Frame, a Slot's worth of data prior to executing the Transformation Function that will functionally Transform the Input Files (manipulate, perform mathematical functions on, scale, etc) to the desired output and place it in the output file(s).

To document the operation of the Component, the concept of a web based data sheet, filed in a OO Class Structured Data base was developed. Figure 5 depicts the Data Sheet concept.

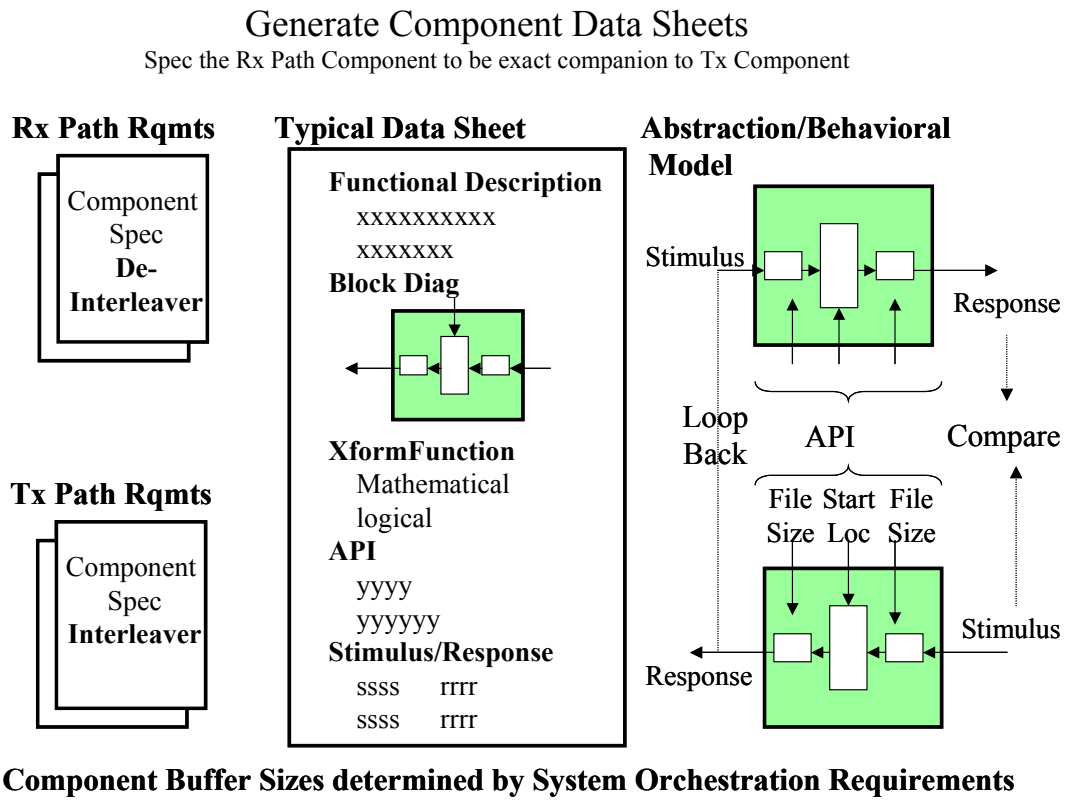

Figure 5

As shown WDL standardized on the look and feel of Component Data Sheets. All data sheets have a short functional description, a block diagram, the executable mathematical and or logical expressions that operate on input files to produce the desired output file(s), an API definition, Stimulus / Response files (Test bench), etc. In addition , as depicted in the figures on the right, the API also allows the Waveform Designer to specify the Dimensions of the Input and Output Files.

As implied in the figure, WDL forces most, if not all Signal Path components to have a mate…a Tx Component and a companion Rx Component. This duality concept allows Loop back testing at both the behavioural and Radio Implementation level of virtually every Signal Path Component.

THE EXECUTABLE BEHAVIOURAL RADIO SPEC

Once the waveform designer gathers the Components from the Web based Library or Data base, or creates the component he needs, he instantiates the Data Sheets that he needs, and develops the Orchestration Component. The Orchestration Component ties the other Components together and provides the timing to move data from Component to Component, and the timing for each Component to perform its Transformation function.

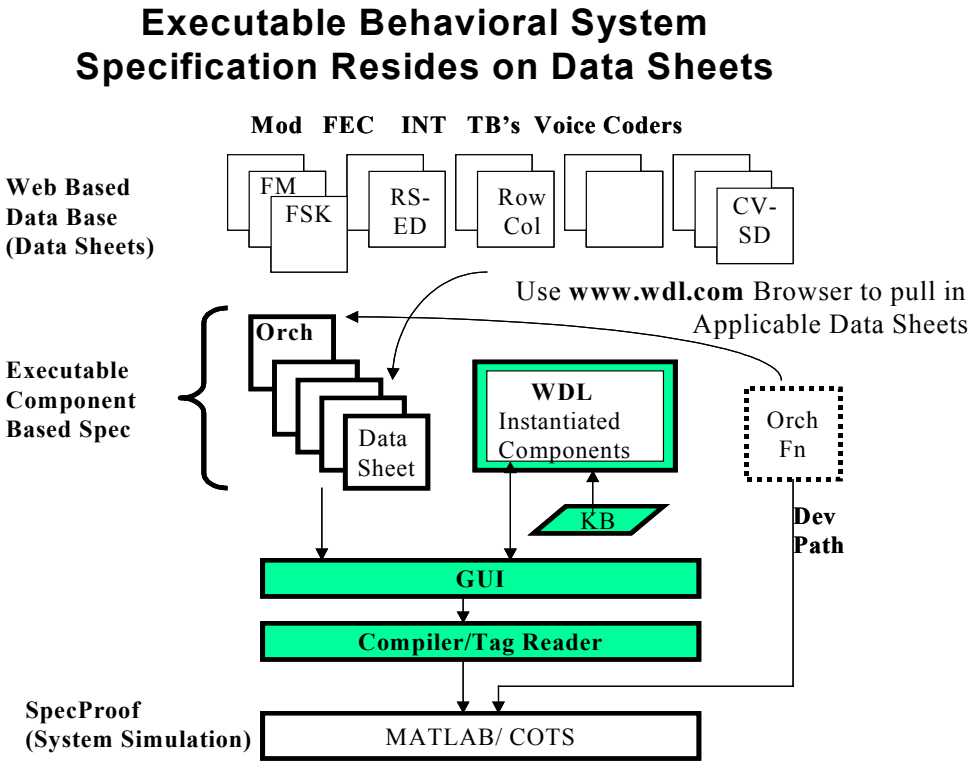

Figure 6… Building Waveform Executable Spec

Once the Orchestration Component is developed, it becomes just another Radio System Component and is placed in the Data Base along with the other instantiated components as depicted in the above referenced figure.

The Set of instantiated Data Sheets, as shown above, rigorously specifies, in an executable behavioural format, the physical layer requirements of a radio system. The ambiguity of the requirements is reduced to essentially zero, and the behavioural specifications being written in a neutral COTS language should be acceptable to all radio implementers.

AUTOMATIC RADIO CODE GENERATION

When we picked MATLAB as a radio vender neutral COTS Language for behaviourally describing the Waveform Functionality of a radio system, we had in mind that industry would develop a compiler that would transform MATLAB Code to VHDL automatically. That day has come!

GOING FROM EXECUTABLE SPEC TO RADIO IMPLEMENTATION

Figure 7 is almost a direct repeat of the previous figure, except in this one we are highlighting how one would get to Radio Code from the behavioural code. Two paths are shown, a manual path where one uses the block diagrams and MATLAB Behavioural Code to develop VHDL or C code, and an Automatic Radio Code Generation path, that uses the MATLAB behavioural Code directly with little manual intervention

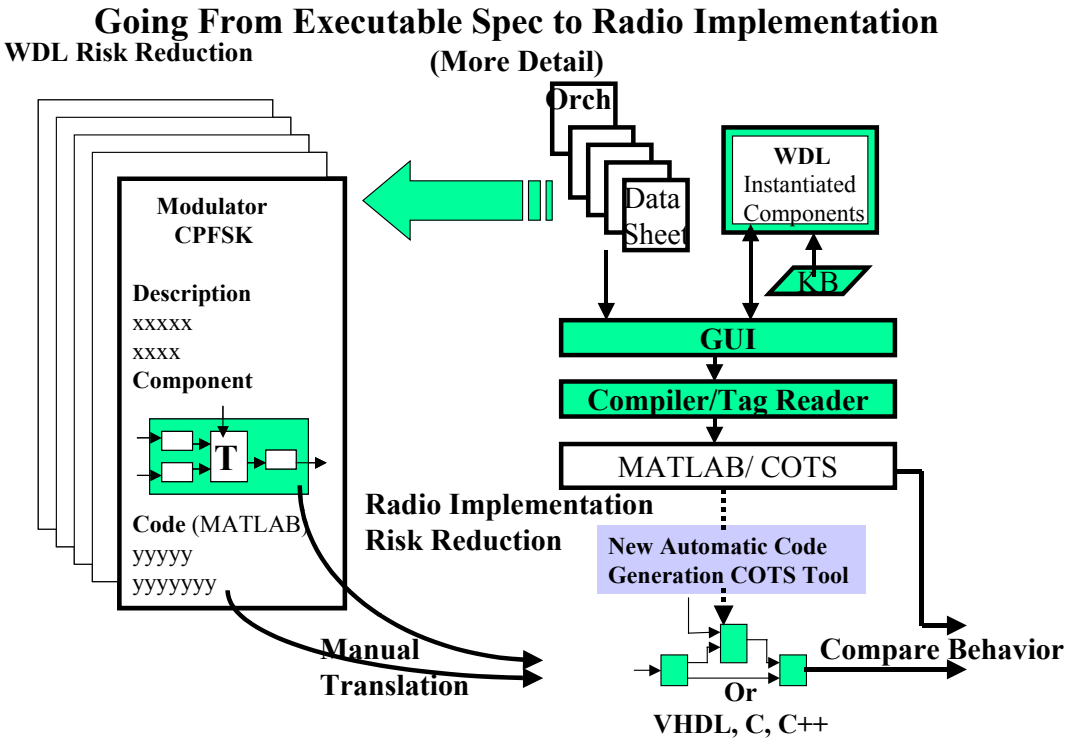

Figure 7… Radio Code Generation methods

For transitioning MATLAB Behavioural Code to C code it was found that the MATLAB Behavioural Code that was written could be transitioned to C Code rather easily. This is especially true for GP's and to a lesser extent for DSP's.

When transitioning MATLAB code to VHDL the story changes for both the manual and Automatic Code Generation paths. The behavioural code needs to be written in a "Style" that has some correspondence to the constraints imposed by FPGA memory "Dimensions", what parameters are "Registered" vs which ones are to use "Memory" structures. In modelling the Transformation Function's Behaviour, the Waveform Designer needs to "Architect" the code to be compatible with the limitations of FPGA's… all brands have the same generalised constraints. It was found that if the person has VHDL experience and training, and is instructed to write code with conversion to VHDL in mind, the code can be "Structured" to lead to an easy translation to VHDL in both the manual and Automated Code Generation paths.

Once the MATLAB code is "Structured" for the FPGA solution, it also is optimised for C code.

WDL AUTOMATIC CODE GENERATION PROCESS

Automatic code generation from "WDL Structured MATLAB Code" was generated for a few WDL Component Data Sheets. The results were very positive. Figure 8 depicts what had to be done to automatically compile the "WDL Structured code" for use in the AccelChip Compiler.

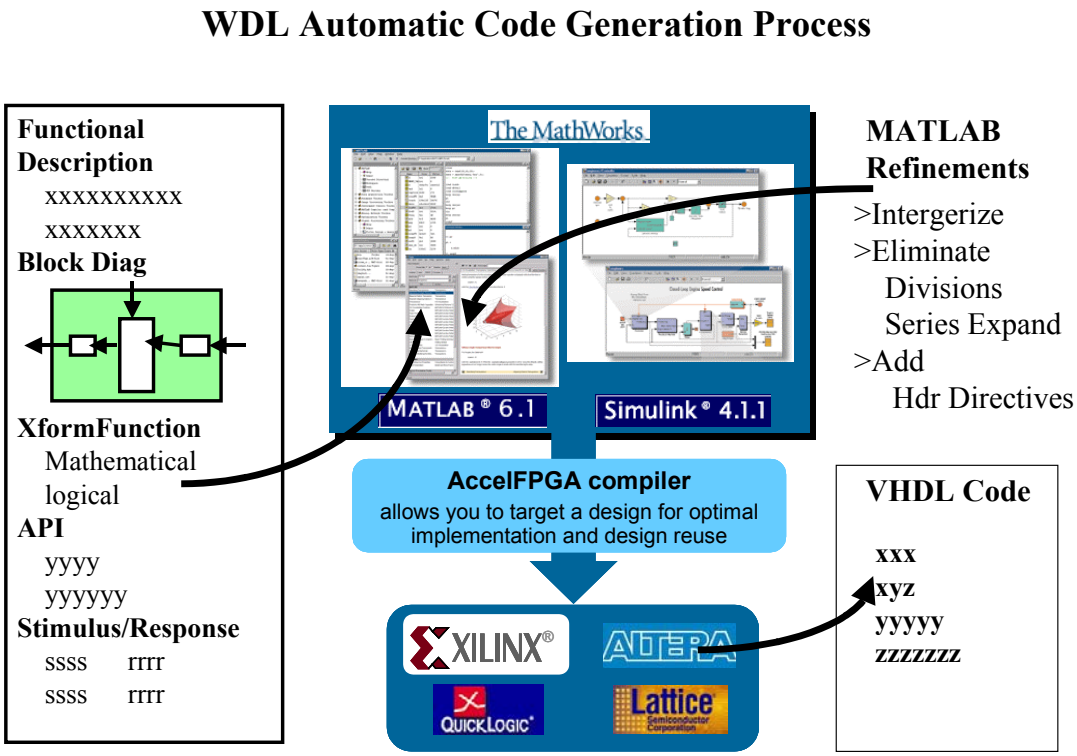

Figure 9… Automatic code generation

To evaluate the AccelChip tool BAE e-mailed a Component Data sheet describing a Complex Row / Column Interleaving Function with three mode of operation…Interleave, Interleave with a commanded Start Position and a By-pass mode to AccelChip. Within 2 days the function was automatically converted to VHDL for an Altera FPGA.

What AccelChip had to do with the e-mailed code is depicted above. As shown the MATLAB Code was put into the MATHWORKS simulator. This code was run through a MATLAB "Intergizer" to insure that all quantities are Integer and that there are no divisions, series expansions, Matrix Transpose, etc. that are not handled by the simple arithmetic, logical primitive functions that can be implemented in FPGA's.

Once the above was done, a few other "Directives" that deal with hardware allocations were added to the WDL Structured Matlab Code as illustrated in the MATLAB Refinements notes.

The compiler was then adjusted for Alteria FPGA and run… Out came VHDL Code representing the Interleaver as depicted in the bottom right of the referenced figure.

CONCLUSIONS

The WDL Process address several key needs in the Software Radio arena. These are listed below:

WDL addresses the black area of the SDR Forums / JTRS radio architecture and is intended to augment the JTRS SCA on the black side of the radio

WDL addresses the physical layer of the OSI model, and is interfaced to the higher Data Link and Network layers of the OSI model via simple API's

WDL forces the Physical Layer specifications into Component based Canonical Radio Templates…Signal Path, TSEC, and Orchestration.

WDL provides software behavioural models of parameterised components that are housed in a web based OO Class structured Library for reuse

WDL Structured MATLAB Code explicitly describes the behavioural Transformation Function that effectively moves Data from Input Files to Output Files.

WDL Provides a path to Manual and Automatic C and VHDL Code Generation that is being tested on the Decomposition of SATURN, a fast frequrncy hopped NATO Waveform which represents the complexity of radios that we want to apply WDL to.

The briefing will discuss all of the items above and include a section on lessons learned on manual and automatic radio code generation.