RAPID PROTOTYPING OF COMMUNICATION WAVEFORMS FROM A HIGH-LEVEL DESIGN LANGUAGE

Mark Beardslee PhD (Coherent Logix Inc, Milpitas CA, USA; beards@coherentlogix.com); Matthew Hall (Coherent Logix Inc, Steamboat Springs CO, USA; hall@coherentlogix.com); Zhong Shang (Coherent Logix Inc, Milpitas CA, USA; shang@coherentlogix.com)

ABSTRACT

This paper presents a rapid prototyping system for realizing communication waveforms onto a real-time multi-processor hardware platform which runs a standard programming language such as ANSI C. The waveform is described in a high-level design language, such as The MathWorks Simulink, and is transformed to an executable multi-processor program using C-code generation tools from The MathWorks and a software tool flow called the HyperX Integrated System Development Environment (hxISDE). The transformation is done automatically, and is guided by user constraints on the performance of the resulting real-time design. The verification tests from the user's high-level design environment are employed to verify the resulting C-code implementation on both software and hardware platforms. Using this tool flow and transformation procedure, one can rapidly create a verified C-code design which runs in real-time on a multi-processor platform. In this paper we present the details of the transformation process and demonstrate its operation on a commonly used communication waveform.

1. INTRODUCTION

With the proliferation of communication waveforms and the continuing evolution of waveform standards, a rapid methodology for prototyping and developing waveforms on a real-time platform has become increasingly important. At the same time, high-level graphical model-based design systems [1][2] are now commonly used to define and optimize electronic systems without writing any low-level source code. The combination of these two trends promises to create a rapid prototyping system that is a major improvement in communications waveform development methodology.

Previous work includes existing commercial tools that attempt to accomplish rapid prototyping to hardware [3][4] [5][6]. These systems primarily target hardware implementations that incorporate Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). Typically, these systems start with a high-level design system such as Simulink and create a design description in a hardware description language (HDL) such as VHDL[7]. From there, tools exist to compile the HDL code for operation on existing FPGA development boards. There are two ways the HDL code gets created: first, generic HDL code can be generated by tools such as Simulink. Second, the tool vendors provide libraries of code to implement Simulink blocks. The reason the second approach is taken is that the generic HDL code may not produce efficient structures for implementation on FPGAs or ASICs. The vendor specific libraries ensure that the generated hardware is efficient on the particular platform. The downside of this approach is that often the block libraries provided by the vendors do not exactly match the behavior of the high-level blocks so there will be functional differences between the high-level design and the design as implemented on the FPGA development board. And consequently, the user can not simply take an existing design through the flow without modification. Another drawback with this flow is that the user does not get an understanding of the performance of their design until it is run completely through the implementation flow – this process could take hours for a reasonable sized design.

Other work in this area is aimed at implementation on ASIC/FPGA platforms using VHDL and custom compute structures [8] and implementing Simulink designs using a combination of hardware (on ASIC/FPGA) and software (on a processor) [9][10].

In this paper we propose a rapid prototype-to-hardware model-based design flow (MBDF) that avoids the aforementioned issues by directly using the C-language code that is generated by the Simulink tool for each of its design entities (called blocks) and by employing a hardware platform that is well suited to implementing Simulink communication and can natively execute the code for each block. In this flow, the performance of the overall system can be calculated from the characterized performances of its constituent blocks. In addition, the verification environment used to verify the Simulink design is employed to verify both the translations of the individual blocks and of the

overall system. We also propose an associated debug and performance measurement environment that enables real-time design visibility and aids in design optimization.

In the following sections, we first describe the hardware platform and how it executes Simulink generated code. Then we detail the complete MBDF high-level language to hardware flow. Finally we show and example of a waveform implemented using this system.

2. THE REAL-TIME HARDWARE PLATFORM

The center of the hardware platform is the HyperX™ processor from Coherent Logix Inc.™ [11]. This chip has 100 core processors arrayed in a 10 by 10 grid. Each core processor has a locally stored program and can execute integer and floating point operations. Interspersed with the core processors are a 11 by 11 grid of Data, Memory, and Router (DMR) blocks which can store and move data across the chip and form a memory embedded in a network architecture. DMRs work in concert to efficiently move blocks of data from one DMR to another, which may be on the other side of the chip – once a connection is established between two DMRs, one word of data is transferred every clock cycle. The DMR-to-DMR connections can be created and destroyed dynamically.

The HyperX processor is employed in the HyperX Hardware Application Development System hxHADS™ which is a configurable hardware environment which can host numerous HyperX processors, General Purpose Processor (GPP) chips, and I/O interface cards. The smallest configuration of an hxHADS includes a HyperX processor, an ARM-based general purpose processor and a PCIe communication card. The development system also includes the software tool suite called the HyperX Integrated System Development Environment (hxISDE™) which is a complete development environment for compilation, simulation, real-time debugging, and optimization of C-language based designs on the HyperX processor.

High-level systems, such as Simulink, that model designs as networks of communicating computation blocks are implemented on the HyperX processor by compiling each block to run on one or more core processors. The data communication is implemented using the DMR-to-DMR mechanism for moving data. This style is similar to a Message Passing model of computation [12]. Thus, at a simplified level, the translation of a Simulink design to HyperX consists of compiling the Simulink Generated C-code for each block and implementing the block-to-block communication using DMR-to-DMR transfers.

3. MBDF - HIGH-LEVEL LANGUAGE TO HARDWARE FLOW

The objective of the MBDF is to provide a well defined starting point for the initial system design from which a designer can rapidly get an accurate understanding of system performance and resource requirements without writing any low-level implementation code. The MBDF enables rapid characterization of architectural tradeoffs so that the designer can explore the impact of different uses of parallelism, communication schemes, and resource allocations on the system performance.

The MBDF has the capability to measure and optimize the performance and resource usage of the design by partitioning the design. Often the number of blocks in the high-level design is larger than the number of core processors it will be run on. In this case, the partitioning algorithm decides which blocks to place on the same core processor (i.e. it serializes those blocks on one core processor) while meeting the given performance and resource constraints.

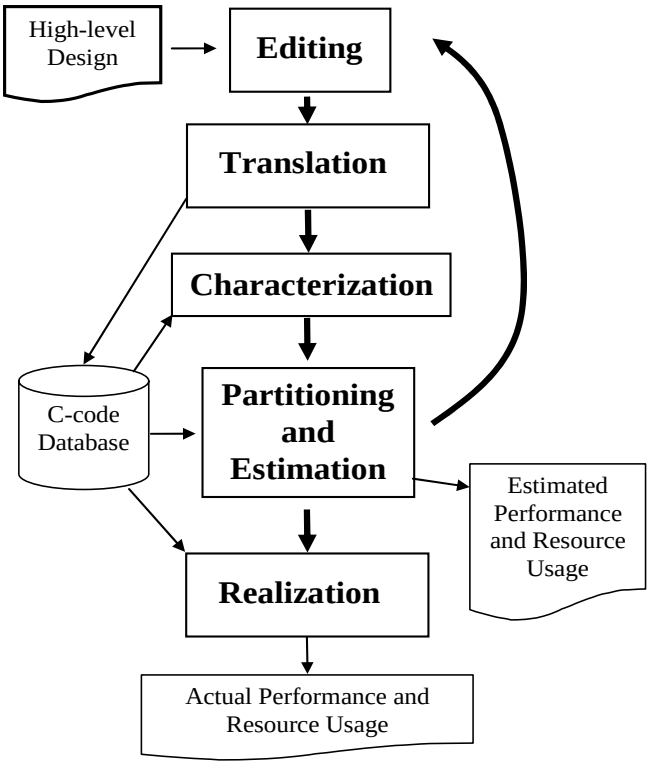

Figure 1 – MBDF: high-level design to hardware flow.

The user can control this partitioning by setting performance and resource constraints on the partitioning optimization

algorithm. The user may also control the partition by manually directing that some set of blocks should all reside on the same core processor.

Additionally, the code generated by Simulink is serial C-code that can only be run on a single core processor. The MBDF does not automatically parallelize serial C-code. So, for some performance critical blocks, the user may elect to replace the Simulink generated code with a custom implementation that has better resource or performance properties. For example, an FFT generated by Simulink will run on a single core processor but may be replaced by a multiple core processor implementation to create better overall system performance. Another option is to break the Simulink block into multiple parallel executing blocks in the Simulink editor. This may not be easy to do for many compute intensive blocks.

The major steps in MBDF (see Figure 1) are described here:

- The Editing step is done in the high-level design environment. Here the user can modify the design as needed and add system-level performance constraints such as system latency, throughput, and resource limits, direct a block to be replaced with a custom implementation, or manually select parts of the design to reside entirely on a single core processor (manual partitioning).
- The translation step utilizes the built-in code generation capability of the high-level design environment (in this case the Real-Time Workshop Embedded Coder™ from The MathWorks) to create code for each block. The translator wraps each block with a communication layer which creates the block-to-block communication and creates block-level and system-level testbenches.
- The Characterization step determines the expected performance and resource requirements of each block on the HyperX by compiling the block with its block-level test bench. The stimulus for the test bench is derived from the simulation done in the high-level design environment, in this case from the Simulink simulation of the Simulink design. From this step we can, at a minimum, get the block latency and the required data and instruction storage for this block.
- The partitioning algorithm starts with a representation of the topology of the system, the block-level performance and resource requirements, a hardware performance model for computation and communication, and user given constraints. Then the partitioning algorithm employs an accurate estimation algorithm to determine the system level performance, system-level resources, and constraint satisfaction as it optimizes the partitioning of the design. At this point a detailed performance report is emitted, and the user has the option to further modify the design and constraints in the editor and re-run the flow.
- Finally, once the user is satisfied with the estimated performance, a complete realization of the design can be created. The difference in this implementation of the design is that any partitions which contain multiple blocks are implemented by combining the code for the blocks into a single executable entity. Additionally, the C-code for custom blocks that the user selected is incorporated into this version of the design.

One of the more time consuming parts of the MBDF is the individual block characterization. A good sized design may have hundreds or thousands of blocks. Importantly, the flow has been designed so that the characterization of a block does not depend on its context. So to reduce runtime, any identical blocks need only be characterized once, and any commonly used blocks can be pre-characterized and the characterization data utilized by the tool flow.

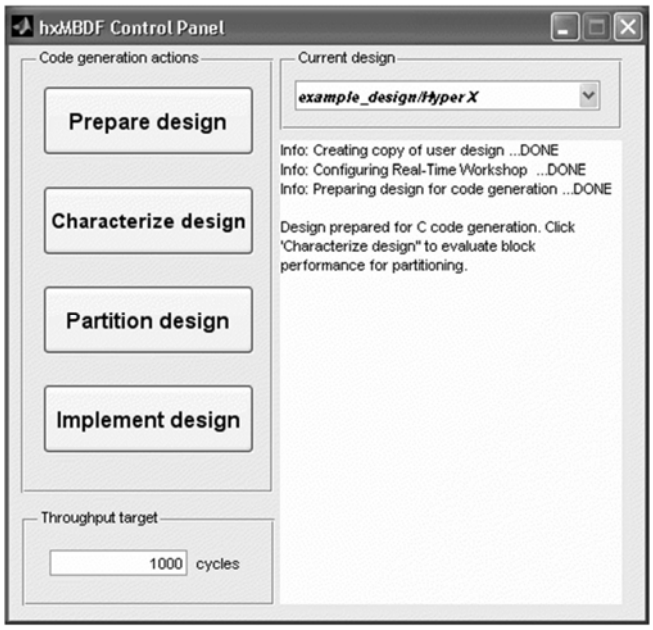

Figure 2 - The MBDF control panel.

3.1 Graphical User Interface

The Simulink design environment is a graphical user interface (GUI) for editing and simulating model-based designs. The MBDF flow adds a window to the Simulink GUI to control and configure the flow (see **Figure 2**). In the "Current Design" entry field the user can declare what part of the design to translate. The "Throughput target" entry field sets the performance constraint on the partitioning algorithm. The 4 buttons on the left side are activated one at a time in descending order. The first ("Prepare Design") configures the user's design for translation. The next three

buttons correspond to the flow steps as described in the previous section and in **Figure 1**. Using these controls, the flow may be run many times to explore different performance targets, design structures, and manual partitions.

The hxISDE is also a graphical design environment (see **Figure 3**) and is integrated with Simulink GUI so that the flow between the tools is consistent and cooperative and the user experience is that of a single unified tool.

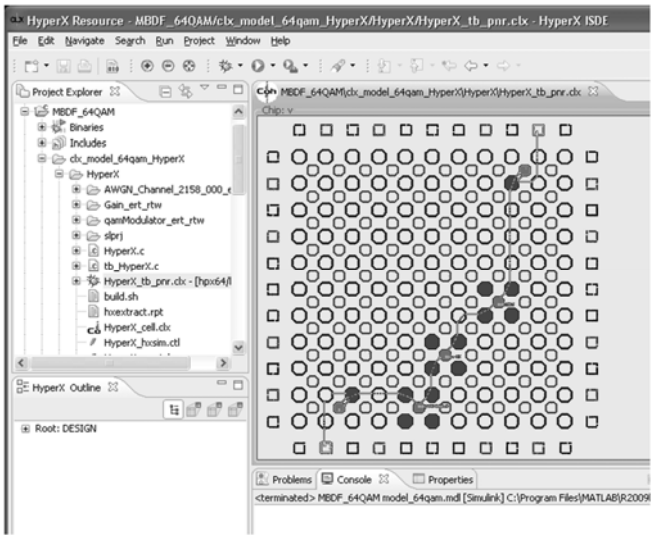

Figure 3 - The hxISDE graphical user interface

3.2 Real-time debugging and performance analysis

This MBDF also allows the user to monitor, in real time, design signals, events, and clocks as well as modify key system parameters on-the-fly. This capability gives the user a powerful debug and optimization capability for designs running on live hardware. In addition, Logic analyzer-like triggering and sampling capabilities can be provided so that the user can get specific data from their design and so that the volume of data being transmitted from the design is minimized. This capability is controlled through the graphical user interface in Simulink. The translator will automatically create the design structures and add them to the final implementation of the design in a way that does not affect the performance of the design.

In addition, this capability can be extended to support real-time performance monitoring capabilities such as power performance, constellation diagrams, and BER/PER characterizations. These diagrams would be displaying live data streamed from the design running live in hardware.

4. RESULTS

The MBDF was applied to a Simulink model of a Scalable OFDM (SC-OFDM) PHY layer. The goal of this example was to validate the capability to automatically implement a real-world DSP design, and additionally, to assess the ability to meet real-world timing requirements.

4.1. The Design

SC-OFDM is derived from the multiple access scheme used in the 3GPP LTE downlink. Scalability is supported by adjusting the FFT size while fixing the sub-carrier frequency spacing to 15 kHz. It supports channel bandwidths ranging from 1.44 to 20 MHz. With bandwidth scalability, LTE can comply with various frequency regulations worldwide.

The scalable OFDM design example is built from the SC-OFDM Simulink model which includes both the transmission and reception algorithms and is complete with error analysis and signal display blocks (see Figure 7). The model can be simulated within the Simulink environment using automatic stimulus generation and post-decode stage packet comparison. The model used for this example was created using "single" floating-point datatype within Simulink but has also been implemented using fixed point datatypes.

As mentioned in the description of SC-OFDM above, the design can be scaled to various available channel bandwidths. For this example the signal bandwidth was set to 1.44MHz bandwidth corresponding to a 128-pt I/FFT signal dimension.

As required by the 3GPP LTE PHY specification, both the Transmitter and Receiver algorithm must meet a throughput requirement of 83.33 µs for the useful symbol period. This constraint was used by the optimization algorithm to produce a design with the lowest resource usage while meeting this throughput constraint.

4.2. The Process

In order to initiate the model-based translation process, the portion of the design to be implemented was isolated in the Simulink model. In this case, both the Transmitter portion and Receiver portion of the design are isolated into their own Simulink subsystems. These subsystems represent two independent designs to be translated.

A C implementation for both the Receiver and Transmitter subsystems were created using the MBDF. During the translation, the blocks which make-up each design were

characterized for performance, memory usage and instruction usage. Using these values, the partitioning algorithm produced an optimized partition of the design.

During characterization, the C-code implementation of each block was functionally verified using vectors generated from Simulink simulations. After partitioning, the C-code generated for each partition was verified using Simulink simulation vectors. Finally, both the complete transmitter and the complete receiver were verified using Simulink simulation vectors.

4.3. Summary of Results

4.3.1. The Transmitter

Initial leaf characterization: *54 Simulink intrinsic blocks*
Post partitioning resource usage: *12 core processors*
Post partitioning throughput achieved: *60.7 µs*

Figure 4 - The final performance of the transmitter blocks

Block Name	*Latency*
Encoder	7.8 µs
Puncture	2.0 µs
Matrix Interleaver	4.3 µs
Block Interleaver	4.3 µs
QAM	9.6 µs
Mod	2.2 µs
Group	12.5 µs
Pilots	0.3 µs
Assemble	15.0 µs
Pad	3.3 µs
Transform	60.7 µs
Cyclic_Multiplex	6.2 µs

Based on the data and instruction space required for each intrinsic block, the partitioning algorithm was able to implement the design using 12 core processors. Additionally, all 12 partition implementations meet the timing constraint of 83.3 µs.

The throughput of this implementation is limited by the generated implementation of the FFT block (labeled Transform in Figure 2). The throughput could be further improved by replacing the FFT with a custom high-performance multi-core-processor implementation.

4.3.2 The Receiver

Initial leaf characterization: *55 Simulink intrinsic blocks*
Post partitioning resource usage: *12 core processors*
Post partitioning throughput achieved: *1224.3 µs*
Post partitioning throughput achieved (after library cell replacement): *56.8 µs*

Figure 5 - The initial performance of the receiver blocks

Block Name	*Latency*
Remove_Demulti	3.4 µs
Transform	56.8 µs
Select	5.1 µs
Equalizer_Sub1	6.7 µs
Equalizer_Sub2	31.5 µs
Dissassemble_Sub1	29.7 µs
Dissassemble_Sub2	9.4 µs
QAM_demod	19.8 µs
blockDeinterleaver	4.3 µs
matrixDeinterleaver	8.2 µs
Viterbi	1224.3 µs
Demod_1	27.1 µs

Based on the data and instruction space required for each intrinsic block, the partitioning algorithm was able to implement the design using 12 core processors. However, in the case of the Receiver block, the generated Viterbi implementation does not meet the throughput performance constraint. Since this block is a Simulink intrinsic block, it cannot be further optimized by the partitioning algorithm.

The Simulink generated Viterbi decoder block is inefficient because it is generic parameterized code that can implement many varieties of decoders. Consequently, a custom hand-coded Viterbi implementation was substituted to meet the required design performance. The custom version of the Viterbi decoder uses one core processor and has been hand-optimized specifically for the HyperX architecture. The performance of the final implementation with the library cell replacement is shown in **Figure 6.**

Figure 6 - The final performance of the receiver blocks with a custom implementation of the Viterbi block.

Block Name	*Latency*
Remove_Demulti	3.4 µs
Transform	56.8 µs
Select	5.1 µs
Equalizer_Sub1	6.7 µs
Equalizer_Sub2	31.5 µs
Dissasseble_Sub1	29.7 µs
Dissassemble_Sub2	9.4 µs
QAM_demod	19.8 µs
blockDeinterleaver	4.3 µs
matrixDeinterleaver	8.2 µs
Viterbi	45.0 µs
Demod_1	27.1 µs

The throughput of this final Receiver implementation is now limited by the generated implementation of the IFFT block (labeled Transform).

5. CONCLUSIONS

In this paper, we have presented a complete development and optimization environment for rapidly transforming high-level designs to hardware. For efficient execution of the design, we employ the HyperX processor which is well suited to executing such designs. This environment has been used to automatically implement, optimize, and verify a current communication algorithm which meets the throughput requirements in the specification.

6. REFERENCES

[1] "Simulink", Version 7, The MathWorks Inc, USA.
[2] "LabVIEW", National Instruments Inc, USA.
[3] "Synphony Model Compiler", Synopsys Inc, USA.
[4] "System Generator", Xilinx Inc, USA.
[5] "DSP Builder", Altera Inc, USA.
[6] D. Haessig, J. Hwang, S. Gallagher, M. Uhm, "Case-Study of a Xilinx System Generator Design Flow for Rapid Development of SDR Waveforms", *Proceedings of the SDR 05 Technical Conference and Product Exposition*, 2005
[7] "IEEE Standard VHDL Language Reference Manual". ANSI/IEEE Std 1076-1993
[8] I. Grout, Configuration and debug of field programmable gate arrays using MATLAB®/SIMULINK®, Journal of Physics: Conference Series 15 (2005) 244–249, Sensors & their Applications XIII
[9] L. M. Reyneri , F. Cucinotta , A. Serra , L. Lavagno, A hardware/software co-design flow and IP library based on simulink, Proceedings of the 38th conference on Design automation, p.593-598, June 2001
[10] Huang, Yan, Han, Chae, Jerraya, Popovici, Guerin, Brisolara and Carro, "Gradual refinement for application-specific MPSoC design from Simulink model to RTL implementation", Journal of Zhejiang University - Science A Volume 10, Number 2, 151-164.
[11] D. Dalio, K. Shelby. "The Implementation of OFDM Waveforms on an SRD Development Platform Supporting a Massively Parallel Processor". *Proceedings of the SDR 09 Technical Conference and Product Exposition*, 2009
[12] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, MPI: The Complete Reference, MIT Press, Cambridge, MA, Sept. 1998.

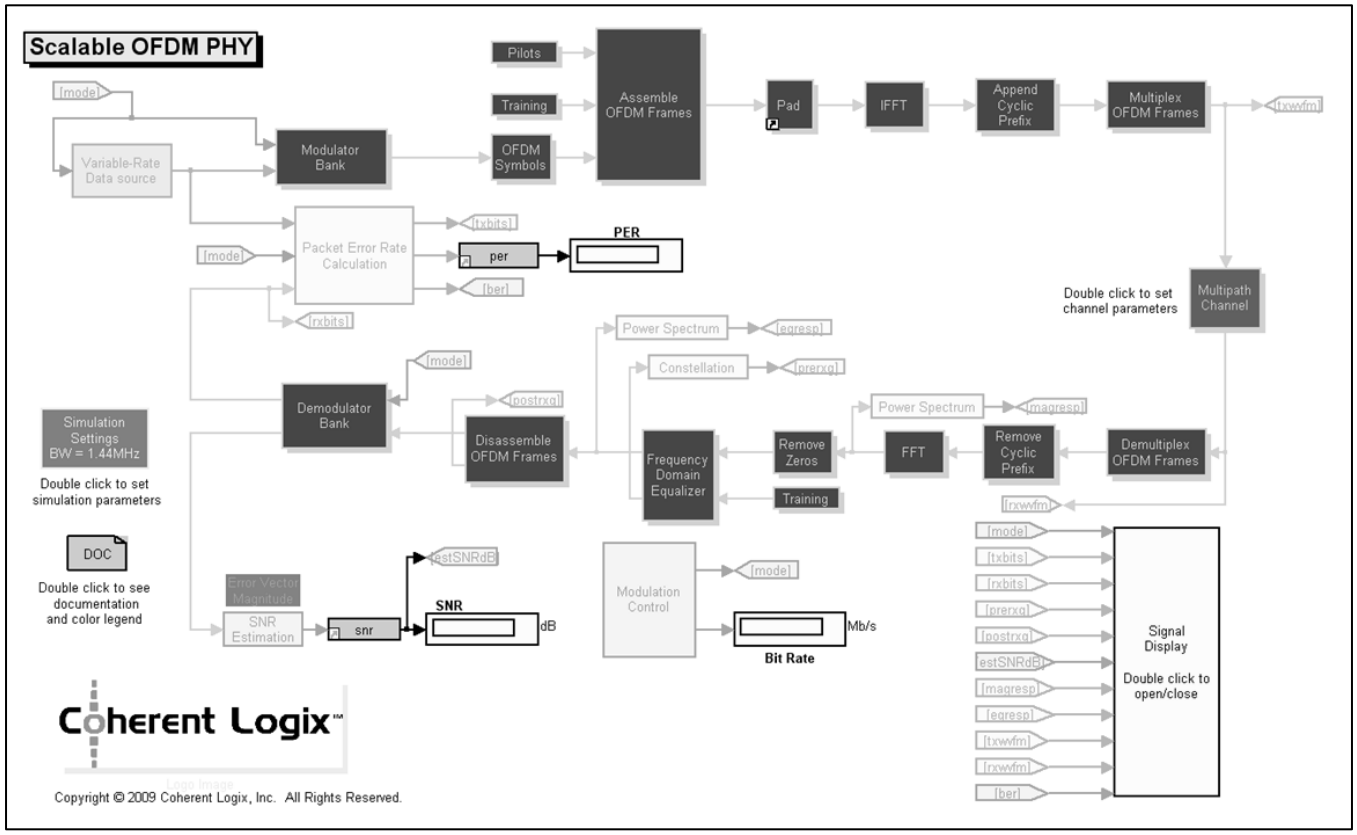

Figure 7 - the Simulink design of the scalable OFDM physical layer