The Syndicated®
A Technical Newsletter for ASIC and FPGA Designers

FPGA Solutions

Synplify® Design Flow Using Xilinx EDK and Embedded Processor Subsystems

by *Andy Norton, Founding Partner, Comm Logic Design, Inc.*

The availability of embedded processor subsystems in FPGAs opens the door to a myriad of applications including control plane subsystems; data path assist subsystems; exception handling processors; diagnostic and test subsystems capable of generating and analyzing data flows; manufacturing diagnostics; and, error testing. Today's FPGAs integrate existing IP and interfaces with custom processing engines and now, embedded processor subsystems. These subsystems are instantiated into a top-level HDL design just as off-the-shelf IP would be integrated.

Experienced design engineers are typically not content to use GUIs without a thorough understanding of what it is doing "under the hood," what files are being created or manipulated and processed.

This article will provide a brief description of the design methodology using Xilinx embedded processors as subsystems, a ProjNav or ISE flow as the EDK project flow using Synplify Pro® software as the FPGA synthesis tool for the overall design. The subsystems, once created, are instantiated into the top-level HDL design and synthesized as in any other Synplify® project. The full application note on the Synplify / EDK design flow is available at http://appnotes.commlogicdesign.com.

Getting Started

When an EDK Base System Builder project is created, various files and directories are created:

System.xmp	project options
System.mhs	microprocessor hardware specification file
System.mss	microprocessor software specification file
/data	contains the base system builder .ucf constraints file
/pcores	user peripheral cores repository directory
/TestApp	a C source test application

EDK project options can be set through the "Project Options" menu which modifies the *system.xmp* file. Project Options should be set to:

- Specify the target device
- For Hierarchy and Flow
  - o Set to SubModule
  - o Synthesis Tool: None
  - o Implementation Tool Flow: ISE
- For HDL and Simulation
  - o If Verilog is selected, only structural simulation can be used (unisims)
  - o VHDL supports both structural and behavioral simulation
  - o The Xilinx Library Path should be indicated

By setting the project options to indicate "SubModule", the specified top instance name will be used when instantiating the subsystem in the top-level design. For submodule, the ISE tool flow must be selected. No synthesis tool is used to synthesize the overall design within EDK (since the instantiated subsystem will be included later in the Synplify project).

Building the embedded processor subsystem means crafting the MHS file and can be accomplished by means of either:

- Base system builder wizard
- GUI selection of peripheral cores
- Direct editing

"EDK" continued on page 2

"EDK" continued from page 1

Once the MHS file has been constructed, platform generation (PlatGen) constructs the "netlist," builds and interconnects indicated peripherals, runs DRC checking, produces error messages/warnings, and generates output files.

Ppc_subsystem

/hdl

system_stub.v	(subsystem with top-level IO inserted)
system.v	(embedded processor subsystem without IO primitives)
peripheral_wrappers.v	(these wrappers are instantiated in system.v)

/implementation

peripherals.ngc files	(XST generated files)
system.bmm	(BMM file without the top-level subsystem instance in path)
system_stub.bmm	(BMM file with top-level subsystem instance in path)

Top-level Embedded Processor Subsystem Files

Platgen will generate two top-level files in **/hdl**: *system_stub.v* and *system.v*.

System_stub.v instantiates *system.v* and adds I/O insertion as Xilinx primitives for all "top-level" ports. Unfortunately, this is not usually what we want since we are instantiating an embedded processor *subsystem*; clock signals could be generated by top-level instantiated DCMs, some subsystem signals may go to other modules at the same level of hierarchy instead of off-chip. Also, using Synplify software, the I/O insertion is automatic and the user does not need to explicitly instantiate BUFG, IBUF, and OBUF primitives for most I/O standards.

Choosing to instantiate *system_stub.v* as our subsystem would then require editing, removing, or modifying the I/O insertion for the ports that do not directly connect to an external pin. Once modified, rerunning PlatGen would overwrite this file once again. Another choice might be to rename *system_stub.v* to *subsystem.v* after editing the file one time with the necessary modifications; the downside to this approach is that one must remember when making port/subsystem modifications, that you have to start from the modified EDK *system_stub.v* file again.

A better approach is to instantiate *system.v* directly in the top-level HDL. The Synplify solution will take care of the necessary I/O insertion where required or, for I/O standards requiring I/O primitive instantiation (for example LVDS) this should be done directly in the top-level HDL file. *System.v* is always correct as generated by EDK PlatGen and never needs to be modified. The one additional step required is at the top-level for the case of tri-state signals. For example, take the case of a **ppc_subsystem** with DDR ddr_dq[31:0] bi-directional ports. Assuming the ports are defined in the top-level Verilog as:

```
        inout                    [31:0]  ddr_dq,
```

EDK PlatGen will generate *system.v* bringing out the tristate signals as shown below:

```
    system
      .ppc_subsystem (
      .ppc_sys_clk         ( ppc_sys_clk_BUFGP ),
      .ppc_sys_clk90       ( ppc_sys_clk90_IBUF ),
      .ppc_core_clk        ( ppc_core_clk_BUFGP ),
      .ppc_ddr_clk90       ( ppc_ddr_clk90_IBUF ),
      .ddr_clk_lock        ( ddr_clk_lock_IBUF ),
      .sys_rst             ( sys_rst_IBUF ),
      .uart_rx             ( uart_rx_IBUF ),
      .uart_tx             ( uart_tx_OBUF ),
      .ddr_casN            ( ddr_casN_OBUF ),
      .ddr_cke             ( ddr_cke_OBUF ),
      .ddr_csN             ( ddr_csN_OBUF ),
      .ddr_rasN            ( ddr_rasN_OBUF ),
      .ddr_weN             ( ddr_weN_OBUF ),
      .ddr_addr            ( ddr_addr_OBUF ),
      .ddr_bankaddr        ( ddr_bankaddr_OBUF ),
      .ddr_dm              ( ddr_dm_OBUF ),
      .ddr_dq_I            ( ddr_dq ),
```

"Incremental Debug" continued on page 3

“EDK” continued from page 2

```
      .ddr_dq_O          ( ddr_dq_o ),
      .ddr_dq_T          ( ddr_dq_t),
      .ddr_dqs_I         ( ddr_dqs ),
      .ddr_dqs_O         ( ddr_dqs_O ),
      .ddr_dqs_T         ( ddr_dqs_T ),
      .led_gpio_I        ( led_gpio ),
      .led_gpio_O        ( led_gpio_O ),
      .led_gpio_T        ( led_gpio_T )
    );
```

The EDK generated *system_stub.v*, the file we don’t want to use, added the IOBUF insertion as shown here for each bus signal:

```
    IOBUF
      iobuf_28 (
      .I ( ddr_dq_O[0] ),
      .IO ( ddr_dq[0] ),
      .O ( ddr_dq_I[0] ),
      .T ( ddr_dq_T[0] )
    );
```

Since we want to be able to instantiate *system.v* directly into our top-level, we must also add the required HDL to control the bi-directional signals:

```
        genvar i;
        generate
                for(i=0; i<=31; i=i+1)
                begin: ddrtri
                        assign ddr_dq[i] = ddr_dq_t[i] ? 1'bZ : ddr_dq_o[i];
                end
        endgenerate
```

Now, EDK generated subsystem Verilog files do not need to be modified, simply instantiated. Bi-directional signals are handled correctly. I/O insertion is either handled automatically by the Synplify tool or, explicitly instantiated as Xilinx primitives when required.

While Xilinx Platform Studio (XPS) uses the Embedded Development Kit (EDK) to generate the embedded processor subsystem, once created, it is simply added to the overall Synplify synthesis project. For example the following lines should be added to the Synplify project file (.prj):

```
        add_file -verilog "../src/project_top.v"
        add_file -verilog "../ppc_subsystem/hdl/system.v"
```

The system can then be fully synthesized as Synplify treats the processor subsystem as a black box instantiation.

About the Author

Andrew Norton is a founding partner of Comm Logic Design Inc, providing design and consulting services specializing in communications and data storage applications. Comm Logic Design is a Xilinx XPERT certified partner specializing in FPGA and embedded processor subsystems focused on architecting, building, and delivering system solutions. He can be contacted at andy@CommLogicDesign.com (www.CommLogicDesign.com).

Synplicity®
Simply Better Results

Synplicity, Inc.
600 W. California Avenue
Sunnyvale, CA 94086 USA
Phone: (US) +1 408 215-6000
Fax: (US) +1 408 222-0263
www.synplicity.com

Copyright © 2004 Synplicity, Inc. All rights reserved. Synplicity, the Synplicity logo, “Simply Better Results,” Synplify, and Synplify Pro are registered trademarks. All other names mentioned herein are trademarks or registered trademarks of their respective companies.