Comm Logic Design, Inc.
FPGA / HW ENGINEERING / μCODE / SW ENGINEERING

Synplicity®
Simply Better Results

Using Xilinx Embedded Processor Subsystems in a Synplify Design Flow

- Andy Norton, CommLogic Design, Inc

Introduction

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 the GUI is doing under the hood, what files are being created or manipulated and processed. In fact, many engineers would often prefer to hand-edit and modify the underlying files directly, the GUI being anathema to their preferred design methods.

Xilinx Embedded Development Kit (EDK) is used for designing embedded processor systems and subsystems in FPGAs with embedded IBM PowerPC™ hard processor cores and/or Xilinx MicroBlaze™ soft processor cores.

This application note discusses a design methodology using Xilinx embedded processors as subsystems, a ProjNav or ISE flow as the EDK project flow using Synplicity as the FPGA synthesis tool for the overall design. The flow uses EDK to create the embedded power subsystem, but once created, the subsystems are instantiated into the top-level HDL design and synthesized as in any other Synplify project.

While EDK tends to be VHDL centric, for purposes of this application note, Verilog is assumed to be the HDL of choice.

All files associated with the reference design mentioned in this application note can be found at http://appnotes.CommLogicDesign.com.

Getting Started

The Xilinx documentation repository is located in the EDK install directory, for example, XILINX_EDK/doc. Technical reference documents cover: PowerPC processor block, busses and interfaces, Embedded Systems Tools guide, IP reference guide, descriptions of software tools and drivers for existing Xilinx IP, user core templates, etc.

Familiarity with the core architecture is paramount to an understanding of how to craft an embedded processor subsystem (Power PC or MicroBlaze).

- For the PowerPC core, Data-side 32-bit On Chip Memory (DSOCM) and Instruction-side 64-bit On Chip Memory (ISOCM) interfaces are low-latency non-cacheable interfaces. The Processor Local Bus (PLB) provides a cacheable 32-bit address bus and 64-bit data bus attached to instruction and data cache units. Inserting a bus bridge, an on-chip peripheral bus (OPB) is used to connect to slow-peripherals and minimize PLB bus loading. A Device Control Register (DCR) 32-bit bus can be used for device configuration and initialization.
- The MicroBlaze is a 32-bit soft processor core with separate 32-bit Instruction on-chip peripheral and 32-bit Data on-chip peripheral busses (IOPB and DOPB) and separate on-chip instruction and data busses with direct connections to on-chip block RAM via the Local Memory Bus (LMB). In addition Fast Simplex Link Buses (FSL) provide dedicated communications channels.

For a designer not familiar with Xilinx Platform Studio (XPS) and the Embedded Development Kit (EDK), the EDK GUI provides a Base System Builder wizard that allows you to create a new project based on existing development boards. This lets you kick start your design and create a pre-built template upon which you can then dig in and start making the necessary edits and modifications. The Base System Builder allows you to

- Select target board, processor, and configuration
- Select I/O interfaces and peripherals
- Generate system and design files

Once the system has been generated, it is a simple matter to edit the files directly instead of using the GUI.

XPS (EDK) Project Flow

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

System.xmp	Project options: location of MHS, MSS, source/header files, architecture, device
System.mhs	Microprocessor hardware specification file: component instantiations, connections and paramterizations
System.mss	Microprocessor software specification file
/data	Contains the base system builder system.ucf constraints file
/pcores	User peripheral cores repository directory
/TestApp	A C source test application and linker script
/etc	XPS options and IMPACT download files

Project Options

EDK project options can be set through the Project Options menu pick which modifies the system.xmp file. Project Options should be set as follows:

- Device and Repository: specify target device
- For Hierarchy and Flow
  - Set to SubModule
  - Specify top-instance name of embedded subsystem (eg. ppc_subsystem)
  - Synthesis Tool: None
  - Implementation Tool Flow: ISE
- For HDL and Simulation
  - If Verilog is selected, only structural simulation can be used (unisims)
  - VHDL can be used for either behavioral or structural simulation
  - EDK Library path to be specified for VHDL behavioral simulation only
  - The unisims/simprims Xilinx Library Path should be indicated

By setting the project options to SubModule, the indicated 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) although EDK will have used XST (and possibly Synplify) in the platform creation of the subsystem peripherals.

Building the embedded processor subsystem means crafting the MHS file and can be accomplished by means of one of the following;

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

The makefiles used by EDK can edited by the user and are now composed of two makefiles: <*projname*>.make and <*projname*>_incl.make. More information on these can be found in the EDK est_guide.

Platform Generation

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

The EDK Platform generated output will be the embedded subsystem composed of

ppc_subsystem	Top-level instance of the subsystem
/hdl	
system_stub.v	HDL subsystem with Xilinx IO primitives inserted
system.v	HDL embedded processor subsystem without IO primitives
peripheral_wrappers.v	Implementation netlist peripheral files; 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
/synthesis	

Platgen will generate two top-level files in /hdl: system_stub.v and system.v. System_stub.v instantiates system.v and adds IO 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, the IO insertion is automatic and the user does not need to explicitly instantiate BUFG, IBUF, OBUF primitives for most IO standards.

Choosing to instantiate system_stub.v as our subsystem would then require editing, removing or modifying the IO 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. Synplify will take care of the necessary IO insertion where required or, for IO standards requiring IO 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, in the case of tristate signals. For example, take the case of a ppc_subsystem with DDR ddr_dq[31:0] bidirectional 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 ),
      .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 file, *the file we do not 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 bidirectional 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. Bidirectional signals are handled correctly and IO insertion is either handled automatically by Synplify or, explicitly instantiated as Xilinx primitives when required.

Memory Generation

PlatGen will also generate the required memory initialization files for the specified BRAMs coupled with DSOCM, ISOCM (Power PC only), LMB (MicroBlaze only), OPB and PLB BRAM controllers.

Two BMM (BlockRAM Memory Map) file will be produced in the /implementation directory by PlatGen: system.bmm and system_stub.bmm. A BMM file will be used in the ProjNav ISE flow to indicate the logical data space used by the embedded subsystem and organization of the BRAM memory. In the case of our subsystem, system_stub.bmm would be used since it contains the complete hierarchical path (since we specified the top-level instance of our subsystem in the project options).

During the bitgen phase of the ProjNav ISE flow, a system_stub_bd.bmm file will be created in the /implementation directory indicating the physical location of the BlockRAMs.

Synplify Hardware Tool Flow

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. Whether the underlying embedded processor subsystem used XST or Synplify or both to create the peripherals and generate the subsystem is irrelevant to the overall Synplify synthesis project.

A typical synthesis project is created by doing the following:

- Defining a project directory hierarchy. For example:

```
fpga_project
   /doc           spec and documentation
   /src           rtl source code files
   /constraints   .ucf, .sdc files
   /sim           simulation files
   /syn           synthesis project files
   /pnr           place and route files
   /ppc_subsystem embedded processor subsystem
```

- Creating a synthesis project

- Adding files to the synthesis project:
  `project_top.v`
  `/ppc_subsystem/hdl/system.v(EDK generated)`

  The EDK generated embedded subsystem is added to the Synplify project with this command:

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

- Synthesizing the project

  System.v contains the actual PPC subsystem with the peripheral wrappers instantiated. At the end of system.v are black box definitions for each of the wrappers. While Synplify does not recognize these XST synthesis directives, it does realize it has to create black boxes and correctly creates them without modification. Synplify will generate the following Warning due to the XST generated synthesis directives:

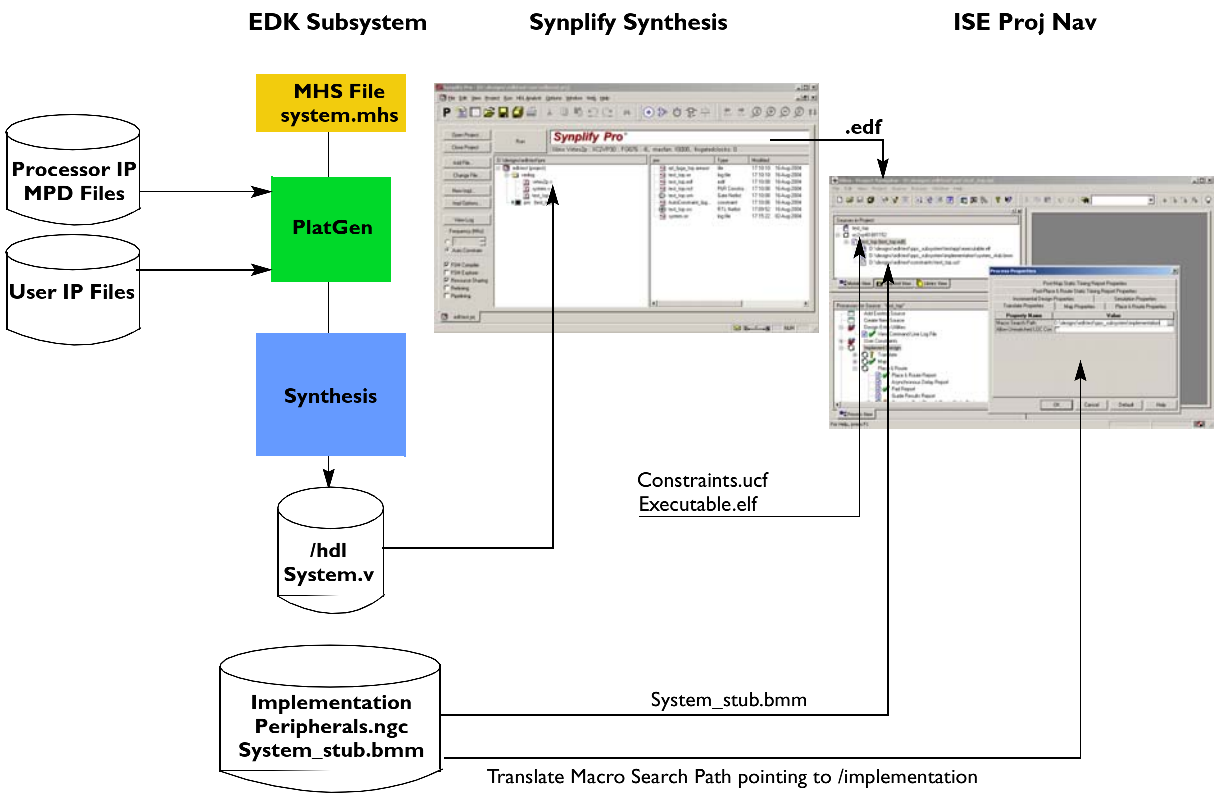
  ```
  @W: CS141: Unrecognized synthesis directive attribute Synthesizing
  module
  ```

  Synplify will then generate the following Warnings due to the black box empty modules:

  ```
  @W: CG146 : Creating black box for empty module ppc405_0_wrapper
  ```

  Synplify generates the following output files:

  – fpga_project.edf (output file)
  – fpga_project.ncf (sdc translated constraints file)



ProjNav ISE Flow

While EDK does provide mechanisms to export files for an ISE ProjNav flow, many designers prefer to build an ISE project separately and add the necessary source files manually.

The Xilinx fpga_project.npl file is the ISE project file in the fpga_project/pnr directory.

The following files are added to ProjNav following the standard ISE flow:

`fpga_project_top.edf`	This is the Synplify generated output file with instantiated ppc_subsystem.
`fpga_project_top.ncf`	This file is not added as a source file. This file is the Synplify generated constraints file translated from the .sdc file
`/constraints/constraints.ucf`	Xilinx constraints file.
`/ppc_subsystem/implementation/system_stub.bmm`	This file requires no modification, assuming that the subsystem instantiated in the top-level module uses the same instance name as was generated by system_stub.v (that is, the top instance name indicated in the project options).
`ppc_subsystem/ppc405_0/code/executable.elf`	An .elf file (pronounced *elf*) is a binary data file that contains an executable CPU code image, ready for running on a CPU. These files are produced by software compiler/linker tools. Data2BRAM uses .elf files as its basic data input form.

Project implementation then follows a normal ProjNav flow producing translate, map, place and route reports. The map report file (.mrp file) will show in the Design Summary the inclusion of the instantiated resources including the embedded processors:

```
Number of PPC405s: 2 out of 2 100%
Number of JTAGPPCs:1 out of 1 100%
```

Running BitGen produces the .bit file that is used for configuration. It also creates a system_stub_bd.bmm file in the /implementation directory; this file can be used by Impact for combining new software .elf files with the generated system.

Directory Structure

The embedded processor system is instantiated in the top-level design as a subsystem. Create an embedded subsystem project directory within your fpga project:

```
Fpga_project_directory
   /ppc_subsystem or /mb_subsystem
```

Platgen creates the following directories below the PPC project directory, ppc_subsystem:

/HDL	system.[vhd|v] file (if top-level) system_stub.[vhd|v] file (if sub-module) peripheral_wrapper.[vhd|v] files
/Implementation	Wrapper files for XST created peripherals (.ngc files) system_stub.bmm (for instantiated BRAMs used in system) system_stub_bd.bmm (created by ISE - bitgen)
/Synthesis	XST synthesis files
/ProjNav	created by EDK for projnav or ISE flow

User defined peripheral cores should be located in /Pcores or in a user-specified project options/peripheral repository directory:

/Pcores	
/data	.pao, .mpd, bbd required files
/hdl	HDL source files
/verilog or	
/vhdl	
/netlist	Precompiled netlist files (.edf or .ngc)

Simgen creates the follow directories:

/Simulation	
/structural	If structural verilog is selected

Libgen configures software libraries and device drivers, and creates the follow directories:

/Ppc405_0	Processor instance directory with instance name
/Code	Default location for .elf file
/Include	SW library.h files
/Lib	Compiler files
/Libsrc	SW library C files

Top Level Verilog Reference Design

A partial view of a reference design showing the top-level port definitions is shown here. DCMs are instantiated in the top-level as well as the embedded subsystem (ppc_subsystem). This is not shown in the following example. For the full design example, see http://appnotes.ComLogicDesign.com.

module fpga_top

```
   (
   // *** DDR SDRAM interface (  SSTL_II Class II compat  ) ***
   // All IO uses SSTL_II_DCI

   input          ddr_clkin,

   // output clock
   output               ddr_clk_out,
   output               ddr_clkN_out,

   // input feedback clock
     input              ddr_feedback_clk,

   // 32-bit external DDR interface
   inout    [31:0]ddr_dq,
   inout    [3:0] ddr_dqs,
   output         ddr_cke,
   output         ddr_rasN,
   output         ddr_casN,
   output         ddr_weN,
   output         ddr_csN,
   output   [3:0] ddr_dm,
   output   [1:0] ddr_BankAddr,
   output   [12:0]ddr_addr,

   // *** PPC based design (3.3V LVTTL) ***
   input          ppc_uart_rx,
   output         ppc_uart_tx,
   output   [15:0]led_gpio,

   // *** misc (3.3V or 2.5V TTL I/O standard) ***
   input          sys_rstN,
   output   reg   heartbeat
   );



// *******************************************************************
// ******                DCMS & CLOCKS                       ******
// *******************************************************************


//---------------------------- PPC_SYSTEM_CLK--------------------------
.
```

```
.
.
//---------------------------- PPC_DDR_CLK-----------------------------
.
.
.

// NOTE: this instantiation name must match the top-instance name
// indicated in the project options so that the system_stub.bmm
// file creates the correct hierarchical path
system ppc_subsystem (
   // reset
      .sys_rst          ( ~sys_rstN ),
   // clocks
      .ppc_sys_clk      ( ppc_sys_clk),
      .ppc_sys_clk90    ( ppc_sys_clk90 ),
      .ppc_core_clk     ( ppc_core_clk ),
      .ppc_ddr_clk90    ( ppc_ddr_clk90 ),

   //  use LAST clk to lock which is the DDR clk
      .ddr_clk_lock     ( ppc_ddr_locked ),

   // UART
      .uart_rx          ( ppc_uart_rx ),
      .uart_tx          ( ppc_uart_tx ),

   // DDR
      .ddr_casN         ( ddr_casN ),
      .ddr_cke          ( ddr_cke ),
      .ddr_csN          ( ddr_csN ),
      .ddr_rasN         ( ddr_rasN ),
      .ddr_weN          ( ddr_weN ),
      .ddr_addr         ( ddr_addr[12:0] ),
      .ddr_bankaddr     ( ddr_BankAddr[1:0] ),
      .ddr_dm           ( ddr_dm[3:0] ),
      .ddr_dq_I         ( ddr_dq[31:0] ),
      .ddr_dq_O         ( ddr_dq_o[31:0] ),
      .ddr_dq_T         ( ddr_dq_t[31:0] ),
      .ddr_dqs_I        ( ddr_dqs[3:0] ),
      .ddr_dqs_O        ( ddr_dqs_o[3:0] ),
      .ddr_dqs_T        ( ddr_dqs_t[3:0] ),

   // output clks
      .DDR_SDRAM_Clk    ( ddr_clk_out ),
      .DDR_SDRAM_Clkn   ( ddr_clkN_out ),

   // LEDs
      .led_gpio_I       (led_gpio[15:0]),
      .led_gpio_O       (led_gpio_o[15:0]),
      .led_gpio_T       (led_gpio_t[15:0])
  );

   // Create IOBUF for DDR DQ SDRAM ports
   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

   // Create IOBUF for DDR DQS ports
   assign ddr_dqs[0] = ddr_dqs_t[0] ? 1'bZ : ddr_dqs_o[0];
   assign ddr_dqs[1] = ddr_dqs_t[1] ? 1'bZ : ddr_dqs_o[1];
   assign ddr_dqs[2] = ddr_dqs_t[2] ? 1'bZ : ddr_dqs_o[2];
   assign ddr_dqs[3] = ddr_dqs_t[3] ? 1'bZ : ddr_dqs_o[3];

   // Create IOBUF for LEDs
   generate
      for(i=0; i<=15; i=i+1)
      begin: ledtri
            assign led_gpio[i] = led_gpio_t[i] ? 1'bZ : led_gpio_o[i];
      end
   endgenerate

endmodule
```

PPC Subsystem Reference Design

The Microprocessor Hardware Specification (MHS) file lives in the EDK root project directory and provides the master description of the subsystem. A system might initially constructed by the Base System Builder Wizard, after which it is easily modified by hand (or through the GUI). Just as HDL coding provides better code organization, naming and commenting, many engineers would prefer to manually edit the MHS file.

Standard and Custom peripheral cores are added to the subsystem by instantiation into this master MHS file, indicating PORTs, DIRection, BUS_INTERFACEs, PARAMETERs, and connectivity. An extensive IP core library is available with EDK.

MHS File

The MHS file defines the configuration of the embedded processor subsystem. This configuration includes selection of the cores, peripherals, processor, parameters, address space and connectivity of the subsystem. Different subsystem configurations can easily be supported by the use of different MHS files.

A partial MHS file is shown below that was initially created by the base system builder and then edited by hand. Key words are indicated in BLUE and comments in GREEN.

Originally the DCMs were created by the wizard, then commented out (bottom of the MHS file), bringing in the clocks from the top-level design where the DCMs are instantiated. The clocks entering the subsystem are

```
# Bring in clks from DCMs at top-level
PORT ppc_sys_clk = ppc_sys_clk, DIR = IN    #for PLB, DSOCM, ISOCM bus
                                             interfaces
PORT ppc_sys_clk90 = ppc_sys_clk90, DIR = IN#for DDR block
PORT ppc_core_clk = ppc_core_clk, DIR = IN  #for PPC CPU core
                                              3:1(core:busses)
PORT ppc_ddr_clk90 = ppc_ddr_clk90, DIR = IN#for DDR block
```

Bus interfaces indicate the connectivity for a common group of signals between the processor and/or peripherals. For example, this subsystem has an ISOCM bus, DSOCM bus, and both Instruction PLB and Data PLB bus interfaces are connected to a common PLB bus with a bus arbiter.

```
   BUS_INTERFACE ISOCM = iocm
   BUS_INTERFACE DSOCM = docm
   BUS_INTERFACE IPLB = plb
   BUS_INTERFACE DPLB = plb
```

Parameters are selected for each pcore/processor overriding the default parameters for the indicated module. For example, for the PLB DDR controller, some of the parameters are:

```
   PARAMETER C_INCLUDE_BURST_CACHELN_SUPPORT = 1
   PARAMETER C_DQS_PULLUPS = 1
   PARAMETER C_REG_DIMM = 1
   PARAMETER C_DDR_TMRD = 20000
   PARAMETER C_DDR_TWR = 20000
   PARAMETER C_DDR_TRAS = 60000
   PARAMETER C_DDR_TRC = 90000
   PARAMETER C_DDR_TRFC = 100000
   PARAMETER C_DDR_TRCD = 30000
   PARAMETER C_DDR_TRRD = 20000
   PARAMETER C_DDR_TRP = 30000
   PARAMETER C_DDR_TREFC = 70300000
   PARAMETER C_DDR_AWIDTH = 13
   PARAMETER C_DDR_COL_AWIDTH = 10
```

Parameters such as the timing parameters can be entered directly from the DDR manufacturers data sheet. Allowable parameters can be determined by viewing the MPD file for the parameterizable pcore. Go to the website appnotes.CommLogicDesign.com to download examples.

```
########################################################################
```

MPD File

The Microprocessor Peripheral Definition (MPD) file defines the interface of the peripheral. An MPD file has the following characteristics:

- Lists ports and default connectivity for bus interfaces
- Lists parameters and default values
- Any MPD parameter is overwritten by the equivalent MHS assignment

MPD files are required for all peripherals. The Xilinx IPLib repository is located in the Xilinx EDK install directory: XILINX_EDK\hw\XilinxProcessorIPLib\pcores. The MPD file for a peripheral is located in the /data subdirectory of the peripheral. Various OPTIONS, BUS_INTERFACEs, PARAMETERs, PORTs and PORT DIRections are specified.

When instantiating a peripheral, only PORTs and PARAMETERs specified in the peripherals MPD file are acceptable and will be cross-checked during DRC checking.

PAO File

A Peripheral Analyzer Order file is required for EDK XST synthesis specifying the list of HDL files and the analyze order. Many sample PAO files can be found in the Xilinx-ProcessorIPLib/pcore repository. The files are listed with the top-level modules last, and the sub-modules preceding them (bottom-to-top order of dependency).

PlatGen system file output

PlatGen (Platform Generation Tool) is invoked to create a netlist based on this master configuration file. PlatGen will check syntax, run DRC checking at various levels, read MPD definitions for each of the selected peripherals, create a system address map, check drivers and various properties, create hardware output directories, top-level HDL files and wrappers and if indicated in the peripherals MPD file (IMP_NETLIST=TRUE indicates that a NGC file is to be produced), synthesize the peripheral using XST synthesis.

As previously described, system.v will be EDK generated and instantiated in the project top-level HDL file. A partial view of system.v for the example MHS file is shown below. Only the PowerPC and the PLB DDR controller are shown for brevity. Note the XST black box synthesis directives produced. As previously described, Synplify will generate warnings for the unrecognized directives but create the required black boxes.

```
//-----------------------------------------------------------------------
// system.v
//-----------------------------------------------------------------------

module system
  (
    ppc_sys_clk,
    ppc_sys_clk90,
    ppc_core_clk,
    ppc_ddr_clk90,
    ddr_clk_lock,
    sys_rst,
    uart_rx,
    uart_tx,
    ddr_casN,
    ddr_cke,
    ddr_csN,
    ddr_rasN,
    ddr_weN,
    ddr_addr,
    ddr_bankaddr,
    ddr_dm,
    ddr_dq_I,
    ddr_dq_O,
    ddr_dq_T,
    ddr_dqs_I,
    ddr_dqs_O,
    ddr_dqs_T,
    led_gpio_I,
    led_gpio_O,
    led_gpio_T
  );
  input ppc_sys_clk;
  input ppc_sys_clk90;
  input ppc_core_clk;
  input ppc_ddr_clk90;
  input ddr_clk_lock;
  input sys_rst;
  input uart_rx;
  output uart_tx;
  output ddr_casN;
  output ddr_cke;
  output ddr_csN;
  output ddr_rasN;
  output ddr_weN;
  output [0:12] ddr_addr;
  output [0:1] ddr_bankaddr;
  output [0:3] ddr_dm;
  input [0:31] ddr_dq_I;
  output [0:31] ddr_dq_O;
  output [0:31] ddr_dq_T;
  input [0:3] ddr_dqs_I;
  output [0:3] ddr_dqs_O;
  output [0:3] ddr_dqs_T;
  input [0:15] led_gpio_I;
  output [0:15] led_gpio_O;
  output [0:15] led_gpio_T;
.....
ppc405_0_wrapper
    ppc405_0 (
      .C405CPMCORESLEEPREQ (  ),
      .C405CPMMSRCE (  ),
      .C405CPMMSREE (  ),
      .C405CPMTIMERIRQ (  ),
      .C405CPMTIMERRESETREQ (  ),
      .C405XXXMACHINECHECK (  ),
      .CPMC405CLOCK ( ppc_core_clk ),
      .....
  // synthesis attribute box_type of ppc405_0 is black_box;

   ddr_sdram_32mx32_wrapper
     ddr_sdram_32mx32 (
       .PLB_ABus ( plb_PLB_ABus ),
   .....
```

```
        .DDR_Clk ( DDR_SDRAM_Clk ),
        .DDR_Clkn ( DDR_SDRAM_Clkn ),
        .DDR_CKE ( DDR_SDRAM_32Mx32_DDR_CKE ),
        .DDR_CSn ( DDR_SDRAM_32Mx32_DDR_CSn ),
        .DDR_RASn ( DDR_SDRAM_32Mx32_DDR_RASn ),
        .DDR_CASn ( DDR_SDRAM_32Mx32_DDR_CASn ),
        .DDR_WEn ( DDR_SDRAM_32Mx32_DDR_WEn ),
        .DDR_DM ( DDR_SDRAM_32Mx32_DDR_DM ),
        .DDR_BankAddr ( DDR_SDRAM_32Mx32_DDR_BankAddr ),
        .DDR_Addr ( DDR_SDRAM_32Mx32_DDR_Addr ),
        .DDR_Init_done (  ),
        .PLB_Clk ( ppc_sys_clk ),
        .Clk90_in ( ppc_sys_clk90 ),
        .DDR_Clk90_in ( ppc_ddr_clk90 ),
        .PLB_Rst ( plb_PLB_Rst ),
        .DDR_DQ_I ( DDR_SDRAM_32Mx32_DDR_DQ_I ),
        .DDR_DQ_O ( DDR_SDRAM_32Mx32_DDR_DQ_O ),
        .DDR_DQ_T ( DDR_SDRAM_32Mx32_DDR_DQ_T ),
        .DDR_DQS_I ( DDR_SDRAM_32Mx32_DDR_DQS_I ),
        .DDR_DQS_O ( DDR_SDRAM_32Mx32_DDR_DQS_O ),
        .DDR_DQS_T ( DDR_SDRAM_32Mx32_DDR_DQS_T )
      );
    // synthesis attribute box_type of DDR_SDRAM_32Mx32 is black_box;
```

Synplify Synthesis Report

The Synplify project for this reference design consisted of the following source files:

```
#add_file options
add_file -verilog "$LIB/xilinx/virtex2p.v"
add_file -verilog "../ppc_subsystem/hdl/system.v"
add_file -verilog "../src/test_top.v"
```

The Synplify warnings encountered were for the following:

- Unrecognized synthesis directives for the XST instantiated Black boxes
- Creating the black boxes for the empty modules

The following excerpt shows examples:

```
$ Start of Compile
#Mon Aug 02 18:00:11 2004

Synplicity Verilog Compiler, version Compilers 2.6.0, Build 102R, built
Jan 27 2004
Copyright (C) 1994-2004, Synplicity Inc.  All Rights Reserved

@I::"C:\Program Files\synplicity\Synplify_751\lib\xilinx\virtex2p.v"
@I::"C:\DATA\edktest\ppc_subsystem\hdl\system.v"
@W: CS141 :"C:\DATA\edktest\ppc_subsystem\hdl\system.v":444:15:444:23|
Unrecognized synthesis directive attribute
....
@I::"C:\DATA\edktest\src\test_top.v" Verilog syntax check successful!
File C:\DATA\edktest\src\test_top.v changed - recompiling
```

```
Selecting top level module fpga_top
Synthesizing module IBUFG
Synthesizing module DCM
Synthesizing module BUFG
Synthesizing module ppc405_0_wrapper
@W: CG146 :"C:\DATA\edktest\ppc_subsystem\hdl\system.v":1041:7:1041:22|
Creating black box for empty module ppc405_0_wrapper
Synthesizing module jtagppc_0_wrapper
@W: CG146 :"C:\DATA\edktest\ppc_subsystem\hdl\system.v":1332:7:1332:23|
Creating black box for empty module jtagppc_0_wrapper
.....
Synthesizing module system
Synthesizing module fpga_top
@END
```

ProjNav ISE Reports

The ProjNav ISE source files were

- `test_top.edf`
- `..\ppc_subsystem\testapp\executable.elf`
- `..\constraints\test_top.ucf`
- `..\ppc_subsystem\implementation\system_stub.bmm`

ISE Translate Properties *must set the* Macro Search Path *to point to the* ppc_subsystem/implementation *directory*, in order for it to find the .ngc peripherals that were black-boxed by Synplify referenced in test_top.edf. These peripherals were created by XST during PlatGen.

Looking at a partial view of test_top.mrp (mapping report) design summary below, one can see the PowerPC and JTAGPPC were successfully created:

```
  Number of PPC405s:                    1 out of       2   50%
  Number of JTAGPPCs:                   1 out of       1  100%
  Number of Tbufs:                     16 out of   9,696    1%
  Number of Block RAMs:                16 out of     192    8%
  Number of GCLKs:                      5 out of      16   31%
  Number of DCMs:                       2 out of       8   25%
```

Xilinx IP Peripherals

Xilinx provides a wide variety of IP cores including

- Bus infrastructure cores
- Memory interface cores
- Peripherals
- User core templates
- Other IP

These cores are provided mostly as VHDL source and located in the EDK install directory at: XILINX_EDK\hw\XilinxProcessorIPLib\pcores. Many cores are free, some come with evaluation licenses. The cores are described in EDK\doc\proc_ip_ref_guide.pdf.

Adding IP Peripheral Cores

Xilinx provides a Create Peripheral Wizard which generates core description files and ensures that a custom peripheral complies with the Xilinx implementation of the IBM CoreConnect PLB and OPB bus standard. The PLB and OPB busses will connect to an IPIF allowing user logic to connect to the IPIC side of the interface.

User core templates (VHDL only) provide a starting point for attaching IP to the OPB and PLB busses. Each user core contains an IP Interface (IPIF). These cores are located in the EDK install directory at XILINX_EDK\hw\XilinxReferenceDesigns\pcores. Reference designs exist for PLB and OPB busses and different Slave Services Packages (ssp) are provided. A user_core_templates_ref_guide is located in the XILINX_EDK\doc directory.

DCR and OCM bus IP cores are not currently supported through a template or wizard. The files for these pcores are created in a similar manner however and the bus protocols are simple to understand.

There are two ways to integrate custom IP cores:

- As a black box, synthesized with Synplify
- As an XST netlist

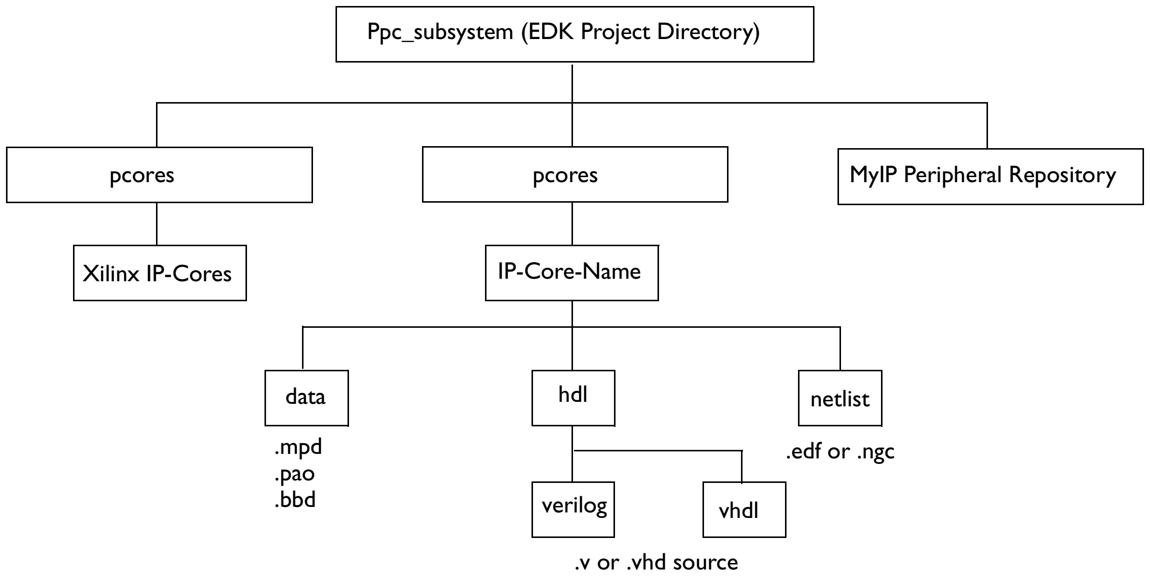
The Synplify generated IP core would then require an associated MPD and BBD (Black Box Definition) file. The Synplify output file user_logic.edf is placed in the /netlist directory. The BBD file would indicate that user_logic.edf is a black box file. An XST netlist is synthesized by PlatGen along with the system and requires an MPD and PAO file.

Pcore Directory Structure

Platgen searches for IP according to the following priority:

- /pcores directory in the project directory
- *<library_path>/<Library Name>*/pcores if -lp option set (project options/peripheral repository)
- EDK/hw/XilinxProcessorIPLib/pcores

The Pcore HDL source files must be located in the /verilog or /vhdl directory. Required MPD, PAO and BBD files for the peripheral must be placed in the /data directory.



Pcore MPD File

The .mpd file specifies PORTs, PARAMETERs, BUS_INTERFACEs and OPTIONs. For verilog, the HDL option specified is

```
OPTION HDL = VERILOG
```

If XST is used as the synthesis tool for creation of the peripheral, the netlist option is

```
OPTION IMP_NETLIST = TRUE
```

If Synplify is used for the creation of the peripheral, the netlist option is

```
OPTION IMP_NETLIST = FALSE
```

This would indicate to PlatGen to NOT run XST synthesis for this peripheral. A peripheral wrapper is still created and instantiated in system.v and the project synthesis run in Synplify would again create a black box for this peripheral. However, in ProjNav, the Translate Search Path must be modified to additionally point to the Synplify created peripheral.edf.

During the creation of an OPB or PLB IPIF (IP interface), the EDK Create Peripheral wizard generates a VHDL IPIF and a user logic block. The user logic might be written in Verilog, synthesized with the Synplify tool, and instantiated as a black box netlist with the .edf generated by the Synplify tool, placed in the /netlist directory, and marked as a black box in the BBD file. You can specify VHDL XST synthesis, although the actual user logic was created using the Synplify synthesis tool.

A separate application note and reference design will be available in the future showing the details of a Synplify synthesized custom peripheral core.

Conclusion

Xilinx embedded processor subsystems created using EDK can easily be integrated into a Synplicity synthesis flow by instantiating the EDK generated embedded subsystem into the top-level HDL design. Synplicity tools can be used not only for the overall project synthesis tool but also as the peripheral core synthesis tool in the creation of custom peripherals.

About the Author

This application note is based on material authored by Andrew Norton, who is a founding partner of Comm Logic Design Inc, which provides design and consulting services, especially for communications and data storage applications. Comm Logic Design is a Xilinx 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 West California Avenue, Sunnyvale, CA 94086 USA
Phone: (U.S.) +1 408 215-6000, Fax: (U.S.) +1 408 990-0290
www.synplicity.com