

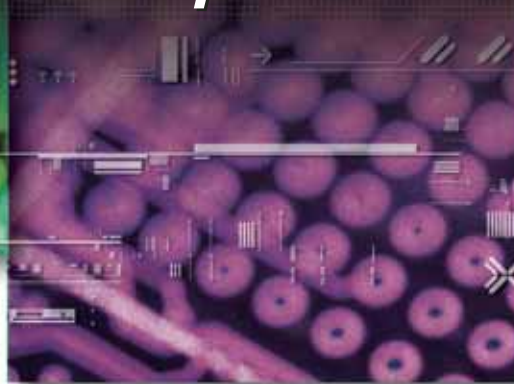
# Integrating EDK-Created Embedded Processor Subsystems

by Andy Norton  
President  
Comm Logic Design, Inc.  
andy@CommLogicDesign.com

The availability of embedded processor subsystems in FPGAs opens the door to a myriad of applications, including embedded network processors, flexible sandbox prototyping, control plane and data path subsystems, and exception handling processors. Today's FPGAs integrate existing IP cores, interfaces, custom processing engines, and now embedded processor subsystems. You can easily instantiate these subsystems into a top-level HDL design just as you would integrate off-the-shelf IP.

Xilinx® Virtex-4™ FX FPGAs integrate a higher performance IBM™ PowerPC™ core with the new Auxiliary Processor Unit interface. The direct connection to the FPGA fabric facilitates advanced coprocessor designs.

You can use Xilinx Platform Studio/EDK software to design embedded processor subsystems in FPGAs with embedded PowerPC hard processor cores or with Xilinx MicroBlaze™ soft processor cores. Although off-the-shelf peripheral cores and MicroBlaze soft cores are synthesized using XST during EDK platform generation, the overall FPGA project and custom peripheral cores are synthesized with Synplicity® Synplify Pro® 8.0, leveraging new features and superior quality of results.



Use Synplify Pro as the primary synthesis tool for complex designs containing embedded processors.



## EDK Subsystem Project Flow

All projects begin by defining an overall FPGA directory structure. The embedded subsystem should reside in its own sub-directory. 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 new EDK project in /ppc\_subsystem results in a system.xmp project file. Next, EDK Project Options must indicate that it is a subsystem by setting:

### 1. Design Hierarchy to SubModule

Specifying the top-instance name of the embedded subsystem (ppc\_subsystem). The indicated top-instance name will be used when instantiating the subsystem in the overall top-level design.

### 2. Synthesis Tool to None

This indicates that no synthesis tool is used to synthesize the overall design within EDK (the instantiated subsystem will be included later in the Synplify Pro project), although EDK will have used XST (and possibly Synplify Pro) in the platform creation of the subsystem and its peripherals.

### 3. Implementation Tool Flow to ISE™

Although Synplify Pro supports mixed languages, you can select Verilog™ or VHDL for EDK output files in Project Options/HDL and Simulation.

## Platform Generation

You can create the embedded processor subsystem by using either the Base System Builder wizard, the GUI selection of peripheral cores, or direct text editing of the microprocessor hardware specification (MHS) file.

Once the MHS file has been constructed, Generate Netlist invokes Platform Generation. PlatGen constructs the netlist, builds and interconnects indicated peripherals, runs DRC checking for errors and warnings, and generates output files.

The EDK Platform generated directories and files include:

<b>ppc_subsystem</b>	top-level instance of the subsystem
<b>/hdl</b>	
system_stub.[vhd v]	HDL subsystem with Xilinx I/O primitives inserted
system.[vhd v]	HDL subsystem without Xilinx I/O primitives
wrappers.[vhd v]	implementation netlist peripheral files with instantiated wrappers
<b>/implementation</b>	
system_stub.bmm	BMM file with top-level subsystem instance in path
system.bmm	BMM file without the top-level subsystem instance in path
peripherals.ngc files	XST-generated peripheral 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. With the processor as a subsystem, system\_stub.v is not used because there are other cores, subsystems, and logic in the design. For example, clock signals could be generated by top-level instantiated DCMs and subsystem signals could go to other modules at the same level of hierarchy instead of off-chip.

Also, using Synplify Pro, the I/O insertion is automatic; you don't need to explicitly instantiate BUFG, IBUF, or 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 not directly connected to an external pin. Once modified, rerunning

PlatGen would overwrite this file once again. Another choice might be to rename system\_stub.v after editing the file; the downside to this approach is that port/subsystem modifications would require you to recreate the modified/edited file.

A better approach is to instantiate system.v directly in the top-level HDL. Synplify 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, in the case of tri-state signals.

For example, you can define the project top-level ports as:

```
module fpga_top
(
    inout      [31:0]  ddr_dq,
);
```

PlatGen will generate system.v (in /implementation), bringing out the tri-state signals as shown in the instantiated ppc\_subsystem:

```
system ppc_subsystem (
.
.
.
.ddd_dq_I      ( ddr_dq ),
.ddd_dq_O      ( ddr_dq_o ),
.ddd_dq_T      ( ddr_dq_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] )
);
```

Because 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 – only instantiated. Bi-directional signals are handled correctly and I/O 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 block RAMs coupled with DSOCM, ISOCM (PowerPC only), LMB (MicroBlaze soft processor core only), OPB, and PLB block RAM controllers.

PlatGen will produce two BMM (block RAM memory map) files in the /implementation directory: system.bmm and system\_stub.bmm. A BMM file will be used in the ISE flow to indicate the logical data space used by the embedded subsystem and organization of the block RAM memory. In the case of our subsystem, system\_stub.bmm would be used, as it contains the complete hierarchical path (because we specified the top-level instance of our subsystem in the project options).

During the ISE bitgen phase of the flow, a system\_stub\_bd.bmm file will be created in the /implementation directory, indicating the physical location of the block RAMs.

### Synplify Project Flow

While XPS/EDK generates the embedded processor subsystem (/implementation/system.v), once created the ppc\_subsystem is instantiated exactly as any IP block by adding it to the overall Synplify synthesis project. Whether the underlying embedded processor subsystem used XST, Synplify, or both to create the peripherals and generate the subsystem is irrelevant to the overall Synplify synthesis project.

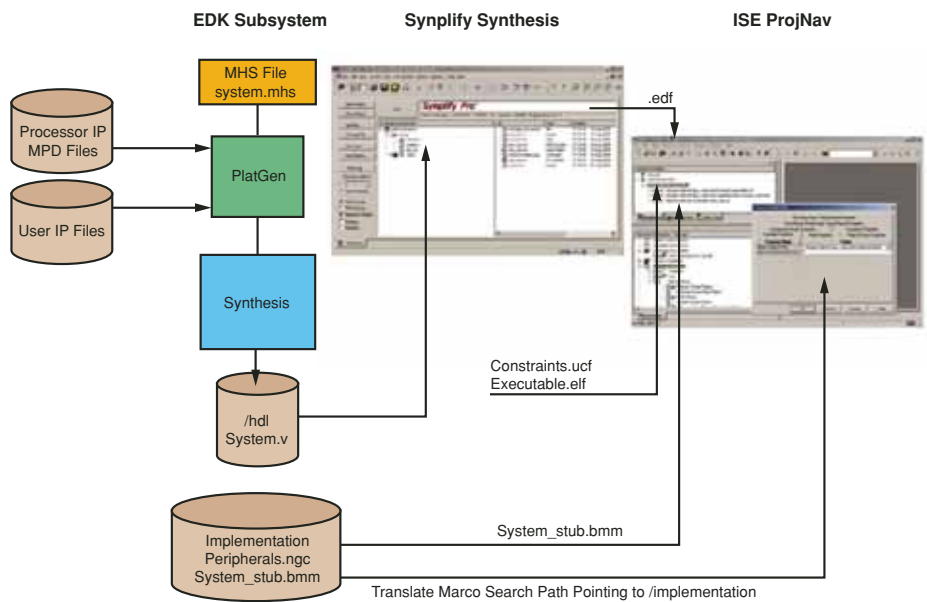


Figure 1 – Synthesis project design flow

A typical synthesis project flow, as shown in Figure 1, would follow this order:

1. Create a synthesis project
2. Add files to the synthesis project
  - project\_top.v
  - /ppc\_subsystem/hdl/system.v (EDK-generated subsystem)
3. Synthesize and review the synthesized project
4. Use the generated output files in the ISE project
  - fpga\_top.edf (top-level source file)
  - fpga\_top.ncf (sdc-translated constraints file)

System.v contains the actual embedded subsystem with the peripheral wrappers instantiated. At the end of system.v are black\_box definitions for each of the wrappers. Although Synplify doesn't recognize these XST synthesis directives, it does realize that it has to create black boxes and does so without modification.

Synplify will generate the warnings shown in Figure 2 because of the XST-generated synthesis directives and empty black box modules. Once reviewed and accounted for, these warnings can now be "hidden" using the Synplify Pro warnings filter

filter, as shown in Figure 3. The filter creates a project.prf file (Figure 4). This file can also be sourced in the Tcl window (source filename).

### ProjNav ISE Flow

The /pnr directory is used for the Xilinx ProjNav ISE flow. The fpga\_project.npl file is created by ProjNav indicating ISE project options.

The following source files are added to the ISE project:

1. fpga\_top.edf (Synplify top-level netlist with ppc\_subsystem)
- fpga\_top.ncf (not added as an explicit source file; created from the Synplify constraints [.sdc])

2. /constraints/constraints.ucf (Xilinx constraints file)

3. /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 generated by system\_stub.v (that is, the top instance name indicated in the project options).

4. /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.

[W]	[D4]	CS141	Unrecognized synthesis directive attribute	system.v [444]	test_top.in [2]	17:10:10	Compiler Report
[E]	[17]	CG146	Creating black box for empty module ppc405_0_wrapper	system.v [1041]	test_top.in [25]	17:10:10	Compiler Report

Figure 2 – Synplify Pro 8.0 compiler warnings

Type	ID	Messages	Source Location	Log Location	Time	Report
[I]	MT204	Autocustom Mode is ON	-	test_top.in [112]	17:10:10	Mapper Report
[I]	FS164	The option to pack flips in the IOB has not been specified	-	test_top.in [129]	17:10:10	Mapper Report
[I]	MT139	This timing report estimates place and route data. Please look at the place and r...	-	test_top.in [212]	17:10:10	Timing Report
[I]	MT137	Clock constraints cover only FF-to-FF paths associated with the clock.	-	test_top.in [214]	17:10:10	Timing Report
[W]	MT186	Black box <leds_16bit_wrapper> is missing a user supplied timing model. This ma...	system.v [1200]	test_top.in [183]	17:10:10	Mapper Report

Figure 3 – Synplify Pro 8.0 warnings filter

```
00001 # Message Filter
00002 log_filter -hide_matches
00003 log_filter -field type==Note -field id==MT186
00004 log_filter -field id==CS141 -field source_loc==system.v
00005 log_filter -field id==CG146 -field source_loc==system.v
00006 log_filter -field message==*translate*
00007 log_filter -field message=="Found counter*"
00008
```

Figure 4 – Synplify Pro .prf file

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

Project implementation then follows a normal ProjNav flow producing translate, map, place and route, and timing reports.

You can easily incorporate embedded processor software changes made by the EDK GNU compiler into the final .bit file without hardware recompiles by running Generate Programming File, or alternatively, the Data2Mem utility. When using Data2Mem, the BMM file specified (-bm) must use the BitGen-generated system\_stub\_bd.bmm in the /implementation directory.

### Custom Peripheral Cores

XPS provides a Create Peripheral Wizard that generates core description files and ensures that custom peripherals comply with the Xilinx implementation of the IBM CoreConnect PLB and OPB bus standard. The PLB and OPB buses will connect to an IPIF, allowing user logic to connect to the IPIF side of the interface. Unfortunately, the wizard currently supports only VHDL. Peripheral cores can also be created in Verilog, but cannot take advantage of the templates created by the wizard.

DCR and OCM bus IP cores are not currently supported through a template or wizard. DCR and OCM bus protocols are simple to understand, however, and you can easily create Pcores for these buses either in VHDL or Verilog. The current EDK-provided OCM buses now allow configurable multi-slave capabilities, providing an easy way to create low-latency slave-only peripherals.

You can integrate custom IP cores into the EDK project either as a black box

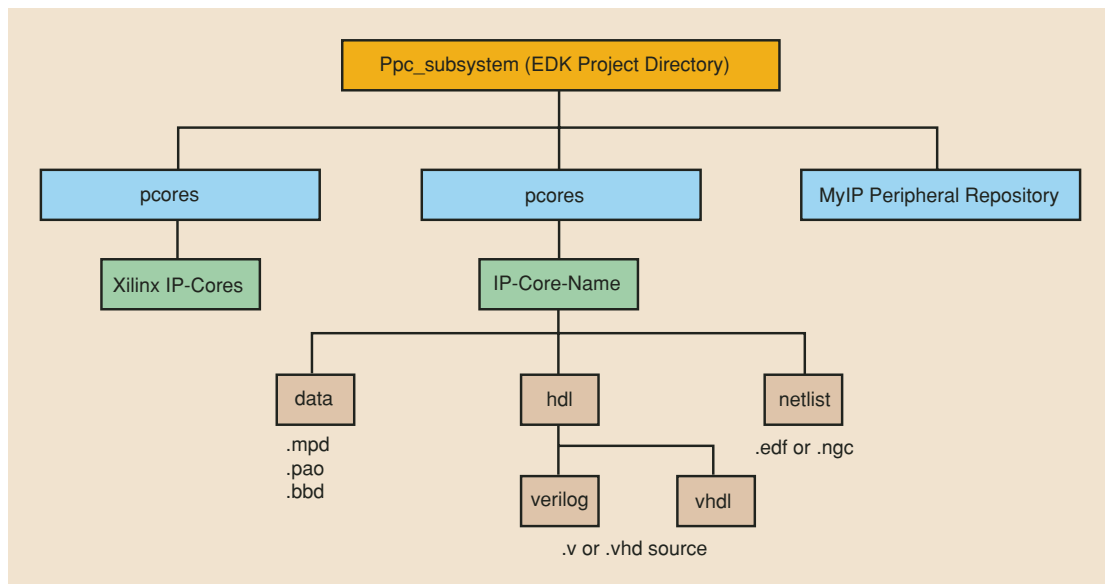


Figure 5 – Pcore directory structure

synthesized with Synplify or as an XST netlist. The Synplify-generated IP core requires associated MPD (microprocessor peripheral definition) and BBD (black box definition) files. The XST netlist is synthesized by PlatGen along with the system and requires MPD and PAO files.

### Directory Structure

Figure 5 shows the required Pcore directory structure. PlatGen searches for IP according to the following priorities:

1. /pcores directory in the project directory
2. <library\_path>/<Library Name>/pcores if -lp option set (project options/peripheral repository)
3. \$EDK/hw/XilinxProcessorIPLib/pcores

### Pcore Files

The Pcore HDL source files must be located in the /verilog or /vhdl directory if they are to be synthesized by XST with PlatGen. If the Pcore is provided as a Synplify-generated netlist, the EDIF must be located in the /netlist directory and indicate its black-box status in a BBD file. Required MPD, PAO, and BBD files for the peripheral must be placed in the /data directory.

The .mpd file specifies PORTs, PARAMETERS, BUS\_INTERFACES, and OPTIONS. For Verilog files, 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 tell 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.

### Conclusion

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

For more information, visit [www.CommLogicDesign.com](http://www.CommLogicDesign.com). Comm Logic Design is a Xilinx XPERTS partner focused on architecting, building, and delivering system solutions for wired-network, telecom, and storage applications.