Extending Embedded System into PL

Zynq
14.2 Version
After completing this module, you will be able to:

- Identify the IP supplied as part of EDK
- Describe how to add hardware to an existing XPS project
- Explain how IP is added to extend processing system functionality
- Discuss the function of PlatGen and bitstream generation
Outline

- IP Catalog
- Pcore directory
- IP device files
- GP Interfaces
- Adding IP to extend PS into PL
- PlatGen and bitstream generation
- Summary
The Zynq-7000 AP SoC architecture consists of two major sections

- **PS**: Processing system
  - Dual ARM Cortex-A9 processor based
  - Multiple peripherals
  - Hard silicon core
- **PL**: Programmable logic
  - Shares the same 7 series programmable logic as
    - Artix™-based devices: Z-7010 and Z-7020 (high-range I/O banks only)
    - Kintex™-based devices: Z-7030 and Z-7045 (mix of high-range and high-performance I/O banks)
Communicating with PL

- **Processing system master**
  - Two ports from the processing system to programmable logic
  - Connects the CPU block to common peripherals through the central interconnect

- **Processing system slave**
  - Two ports from programmable logic to the processing system

- **Slave PL peripherals address range**
  - 4000_0000 and 7FFF_FFFF (connected to GP0) and
  - 8000_0000 and BFFF_FFFF (connected to GP1)
The IP catalog contains all of the components needed to assemble an embedded system.

- Listed in a tree structure by functional category
- Facilitates quick system construction
- Each IP core has its own configuration parameters
- Most of the peripherals are free, few require licenses
- Stored as source code in the install directory
  - Always synthesized with the latest tools
  - Some proprietary source code is encrypted
  - Source directory is typically $C:\Xilinx\14.x\ISE_DS\EDK\hw\XilinxProcessor\PLib\pcores$

Peripherals in the PS are always present and can be dynamically enabled or disabled through PS Configuration wizard.
IP Peripherals
Included as VHDL Source (Free)

- **Bus and bridge controllers**
  - PLBv46 to AXI and AXI to PLBv 46 bridge
  - AXI to AXI connector
  - Local Memory Bus (LMB)
  - Fast Simplex Link (FSL)
  - AXI Chip to Chip
  - AHB-Lite to AXI
  - AXI4-Lite to APB
  - AXI4 to AHB-Lite

- **Debug cores**
  - ChipScope™ Pro tool

- **DMA and Timers**
  - Watchdog, fixed interval

- **Inter-processor communication**

- **External peripheral controller Memory and memory controller**

- **High-speed and low-speed communication peripherals**
  - AXI 10/100 Ethernet MAC controller
  - Hard-core tri-mode Ethernet MAC
  - AXI IIC
  - AXI SPI
  - AXI UART

- **Other cores**
  - System monitor
  - Xilinx Analog-to-Digital Converter (XADC)
  - Clock generator
  - System reset module; interrupt controller
IP Cores Included as Evaluation

- AXI CAN controller
- AXI USB2 device

Xilinx developed, delivered, and supported
Evaluation IP installs with a 90-day evaluation license
IP Core Information

- The size of each core is available in the data sheet
- For example, the axi_timer_v1_03_a data sheet contains the following table:

<table>
<thead>
<tr>
<th>Parameter Values</th>
<th>Device Resources</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C_COUNT_WIDTH</td>
<td>Slices</td>
</tr>
<tr>
<td></td>
<td>C_ONE_TIMER_ONLY</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>95</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>108</td>
</tr>
</tbody>
</table>
Outline

- IP Catalog
- *Pcore directory*
- IP device files
- GP Interfaces
- Adding IP to extend PS into PL
- PlatGen and bitstream generation
- Summary
Peripheral Storage

User peripherals can be located in the project directory or a peripheral repository

Platform Generator searches the following directories for IP:

- pcores directory (located in the project directory)
- MyProcessorIPLib directory (user defined)
  - Repository Directory listed using Project → Project Options → Device and Repository Search tab
  - \$XILINX_EDK/EDK/hw/XilinxProcessorIPLib/pcores (UNIX)
  - %XILINX_EDK%\EDK\hw\XilinxProcessorIPLib\pcores (PC)
Outline

- IP Catalog
- Pcore directory
- IP device files
- GP Interfaces
- Adding IP to extend PS into PL
- PlatGen and bitstream generation
- Summary
Peripheral Device Files

Microprocessor Peripheral Definition (MPD)
– Provides default parameters and options for peripheral device

Peripheral Analysis Order (PAO)
– Contains the list of HDL files that are needed for synthesis and defines the analyze order for compilation

Black-Box Definition (BBD)
– Manages the file locations of optimized hardware netlists for the black-box sections of your peripheral design
Peripheral Options

BEGIN axi_gpio

# Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION DESC = AXI General Purpose IO
OPTION LONG_DESC = General Purpose Input/Output (GPIO) core for the AXI bus.
OPTION IP_GROUP = General Purpose IO:MICROBLAZE
OPTION ARCH_SUPPORT_MAP = (spartan6t=PRODUCTION, spartan6=PRODUCTION, spartan61
OPTION RUN_NGCBUILD = FALSE
OPTION STYLE = HDL

IO_INTERFACE IO_IF = gpio_0, IO_TYPE = XIL_AXI_GPIO_V1

- Select various options
  - HDL Language
  - Supported device architectures
    - Supported processors
  - Provide description
**Bus Interfaces and Parameters**

Specify possible bus interfaces

```vhdl
## Bus Interfaces
BUS_INTERFACE BUS = S_AXI, BUS_STD = AXI, BUS_TYPE = SLAVE
```

Over-rides HDL generics

```vhdl
## Generics for VHDL or Parameters for Verilog
PARAMETER C_FAMILY = virtex6, DI = STRING
PARAMETER C_INSTANCE = axi_gpio_inst, DI = STRING
PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector(31 downto 0),
PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector(31 downto 0),
PARAMETER C_S_AXI_ADDR_WIDTH = 9, DI = INTEGER, BUS = S_AXI, ASSIGNMENT,
PARAMETER C_S_AXI_DATA_WIDTH = 32, DI = INTEGER, BUS = S_AXI, ASSIGNMENT,
PARAMETER C_GPIO_WIDTH = 32, DI = INTEGER, RANGE = (1:32), PERMIT = BA,
PARAMETER C_GPIO2_WIDTH = 32, DI = INTEGER, RANGE = (1:32), PERMIT = B,
PARAMETER C_ALL_INPUTS = 0, DI = INTEGER
PARAMETER C_INTERRUPT_PRESENT = 0, DI = INTEGER, RANGE = (0,1)
```

```
C_FAMILY : string := "virtex6";
C_INSTANCE : string := "axi_gpio_inst";
```

```
-- -- AXI Parameters
C_S_AXI_ADDR_WIDTH : integer range 9 to 9 := 9;
C_S_AXI_DATA_WIDTH : integer range 32 to 128 := 32;
```

```
-- -- GPIO Parameters
C_GPIO_WIDTH : integer range 1 to 32 := 32;
C_GPIO2_WIDTH : integer range 1 to 32 := 32;
C_ALL_INPUTS : integer range 0 to 1 := 0;
```
Lists peripheral signal ports that are accessible in XPS

```plaintext
PORT S_AXI_ARADDR = ARADDR, DIR = I, VEC = [8:0], ENDIAN =
PORT S_AXI_ARVALID = ARVALID, DIR = I, BUS = S_AXI
PORT S_AXI_ARREADY = ARREADY, DIR = O, BUS = S_AXI
PORT S_AXI_RDATA = RDATA, DIR = O, VEC = [(C_S_AXI_DATA_WI
PORT S_AXI_RRESP = RRESP, DIR = O, VEC = [1:0], BUS = S_AXI
PORT S_AXI_RVALID = RVALID, DIR = O, BUS = S_AXI
PORT S_AXI_RREADY = RREADY, DIR = I, BUS = S_AXI
PORT IP2INTC_IRQt = "", DIR = O, SIGIS = INTERRUPT, SENSITI
PORT GPIO_IO_I = "", DIR = I, VEC = [(C_GPIO_WIDTH-1):0], E
PORT GPIO_IO_O = "", DIR = O, VEC = [(C_GPIO_WIDTH-1):0], E
PORT GPIO_IO_T = "", DIR = O, VEC = [(C_GPIO_WIDTH-1):0], E
PORT GPIO2_IO_I = "", DIR = O, VEC = [(C_GPIO2_WIDTH-1):0],
PORT GPIO2_IO_O = "", DIR = O, VEC = [(C_GPIO2_WIDTH-1):0],
PORT GPIO2_IO_T = "", DIR = O, VEC = [(C_GPIO2_WIDTH-1):0],
PORT GPIO_IO = "", TRI_O = GPIO IO_I O, TRI_T = GPIO IO T, D
PORT GPIO2_IO = "", TRI_O = GPIO2 IO O, TRI_T = GPIO2 IO T,
```

Bus Signals

User data and control signals
PAO File

Contains a list of HDL files required for synthesis, and defines the analyze order for compilation

```plaintext
# Peripheral Analyze Order File
#
#
#--------------------------------------

#------Proc Common Library Files--------
lib proc_common_v3_00_a all

#------ PLB IPIF Library Files---------
lib axi_lite_ipif_v1_01_a all

#------ INTERRUPT CONTROL Library File --
lib interrupt_control_v2_01_a all

#------GPIO Files ----------------------
lib axi_gpio_v1_01_b gpio_core.vhd vhdl
lib axi_gpio_v1_01_b axi_gpio.vhd vhdl
```

Order of dependency

Add lower level modules entry here
BBD File

Manages file locations of optimized hardware netlists for black-box sections of the peripheral design

- The NGC netlists are copied into the project/implementation directory
  - The MPD file should have `OPTION STYLE = MIX` for the tools to copy the files

- A BBD file content defining single netlist file without options:
  FILES
  Blackbox.ngc

- A BBD file content defining multiple file selections without options:
  FILES
  blackbox1.ngc, blackbox2.ngc, blackbox3.edn
Example of a BBD File with multiple file selections

- A BBD file content defines multiple file selections
- C_FAMILY, C_BUS_CONFIG are options
- FILES should be the last columns in the first line
- Actual file paths may be placed

```
<table>
<thead>
<tr>
<th>C_FAMILY</th>
<th>C_BUS_CONFIG</th>
<th>FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtue</td>
<td>1</td>
<td>virtex/ip1.edf</td>
</tr>
<tr>
<td>virtue</td>
<td>2</td>
<td>virtex/ip2.edf</td>
</tr>
<tr>
<td>spartan2</td>
<td>1</td>
<td>virtex/ip1.edf</td>
</tr>
<tr>
<td>spartan2</td>
<td>2</td>
<td>virtex/ip2.edf</td>
</tr>
<tr>
<td>virtexe</td>
<td>1</td>
<td>virtex/ip1.edf</td>
</tr>
<tr>
<td>virtexe</td>
<td>2</td>
<td>virtex/ip2.edf</td>
</tr>
<tr>
<td>spartan2e</td>
<td>1</td>
<td>virtex/ip1.edf</td>
</tr>
<tr>
<td>spartan2e</td>
<td>2</td>
<td>virtex/ip2.edf</td>
</tr>
<tr>
<td>virtex2</td>
<td>1</td>
<td>virtex2/ip1.edf</td>
</tr>
<tr>
<td>virtex2</td>
<td>2</td>
<td>virtex2/ip2.edf</td>
</tr>
<tr>
<td>virtex2p</td>
<td>1</td>
<td>virtex2/ip1.edf</td>
</tr>
<tr>
<td>virtex2p</td>
<td>2</td>
<td>virtex2/ip2.edf</td>
</tr>
</tbody>
</table>
```
Three ways to integrate your own IP:

– As a black box
  • Synthesized with XST or a third-party synthesis tool
  • Requires MPD and BBD files
    □ MPD file should have `OPTION STYLE = MIX`

– As a source
  • Synthesized with the rest of the processor system
  • Uses XST
  • Requires MPD and PAO files

– Mix
  • Uses netlist and source files
  • Requires MPD, PAO, and BBD files
    □ MPD file should have `OPTION STYLE = MIX`
Outline

- IP Catalog
- Pcore directory
- IP device files
- **GP Interfaces**
- Adding IP to extend PS into PL
- PlatGen and bitstream generation
- Summary
GP Ports

- By default, GP Slave and Master ports are disabled (indicated by gray arrows)

- Enable GP Master and/or Slave ports depending on whether a slave or a master peripheral is going to be added in PL

- If GP Master/Slave are enabled but are of different protocol (AXI Lite vs Full) then enable another Master/Slave port since two slave devices or two master devices using different protocols can not be combined to the same GP interface

- If more IPs of the same kind and protocol are added then an axi_interconnect block with appropriate configuration and functionality is automatically added
Enabling GP Ports

➢ Click on the 32b GP AXI Master Ports or 32b GP AXI Slave Ports configuration boxes in Zynq tab

➢ Click on the check box of the corresponding ports to enable them
Outline

- IP Catalog
- Pcore directory
- IP device files
- GP Interfaces
- *Adding IP to extend PS into PL*
- PlatGen and bitstream generation
- Summary
Adding IP to the Design

- Enable Slave or Master GP port(s) of the PS
- Add IP of the desired functionality using IP Catalog tab
- Connect the added IP to the appropriate GP port
- Assign address to the added IP, if unmapped
- Configure the IP, if necessary
- Make ports connections, if needed
  - Make ports external if the added IP is interacting with external devices
Add IP in the PL

- Make sure that M_AXI_GP0 or M_AXI_GP1 is enabled
  - Select Make New Connection for the enabled port
- Select the IP Catalog tab in XPS
- Expand the group(s) of IP in the left window
- Select an IP and double-click or drag it to add to the System Assembly View window
- Click Yes and followed by OK twice to add and connect to the system
  - This automatically updates the system MHS file
Expand the IP instance, click under the Bus Connection column, and select a bus instance to which it needs to connect

– Typically peripherals are added after the bus structures
– Clicking the hollow square (master) or circle (slave) next to the peripheral or processor attaches it to the selected bus
– When peripherals are added after the bus(es), the tools will ask which connection to make
Peripherals in the Zynq™ AP SoC PS have fixed addresses and do not appear in the address map when an IP is added to the system.

Click on the Generate Address button.

The address will be generated and show the generated addresses of the added IP and the fixed addresses of the configured peripherals of the PS.
Double-click or right click the instance and select **Configure IP** to open the configurable parameters dialog box (refer to the datasheet if needed)

- Default values are shown
  - Customize the parameters that you want
Select the Ports tab

Click the + sign to see available ports

Click under the Connected Port column and select the appropriate signal

- Drop-down menu organizes signals by source peripheral/processor and an appropriate signal from that peripheral/processor
- "External ports" are considered sources even if they are outputs
If the signal needs to be available outside the embedded portion of the design, make it external
  – Right-click the signal and select Make External

For global ports, click Add External Port and assign a name
  – Usually this does not need to be done if above step is performed
    • "External ports" are considered sources even if they are outputs
Connecting Ports: Port Filters

- Port filters remove undesired congestion in the net column by sorting on ports of interest.
- When the Defaults box is selected, all possible port connections, including bus members, are displayed.
Outline

- IP Catalog
- Pcore directory
- IP device files
- GP Interfaces
- Adding IP to extend PS into PL
  - PlatGen and bitstream generation
- Summary
After defining the system hardware, the next step is to create hardware netlists with the Platform Generator (PlatGen) if the system hardware has logic in PL.

PlatGen requires the following files as inputs:
- Microprocessor Peripheral Definitions (MPD) file (one per peripheral)
  - MPD parameters are the defaults
- Microprocessor Hardware Specification (MHS) file (one for the whole system)
  - MHS file parameters override MPD parameters

PlatGen constructs the embedded processing system in the form of hardware netlists for each component (HDL wrappers and implementation netlist files).

If the system contains hardware in PL, bitstream must be generated.
If bitstream is generated, the FPGA must be programmed before application can be downloaded and executed.
PlatGen Generated Directories

**hdऀ directory**
- `system.[vhd|v]` file (if top level)
- `system_stub.[vhd|v]` file (if submodule)
- `peripheral_wrapper.[vhd|v]` files

**implementation directory**
- `peripheral_wrapper.ngc` files
- `system.ngc` file
- `system.bmm` file

**synthesis directory**
- `peripheral_wrapper.[prj|scr]` files
- `system.[prj|scr]` files
Memory generation
- One of the tasks Platform Generator perform is to allocate block RAM based on the MHS and allocate memory space to the various memory controllers
- A BMM file is generated
  - Describes how block RAMs are used in embedded design
  - Placement information is later added to the BMM file by PAR
  - Used by Data2MEM to load the ELF software object into block RAMs (for MicroBlaze-based system)
  - ELF application merged into bitstream (loaded into block RAM) using Data2Mem (MicroBlaze-based system)
  - If no block RAMs are used, the BMM file contains no information

Current block RAM controllers include the following
- AXI block RAM controller (axi_bram_cntlr)
- LMB block RAM Controller (lmb_bram_if_cntlr)

Zynq AP SoC boots from external memory
- Data2Mem not required
ADDRESS_MAP processing_system7_0 ARM 100
ADDRESS_SPACE axi_bram_ctrl_0_bram_block_1_combined COMBINED [0x81210000:0x8121ffff]
ADDRESS_RANGE RMB32
BUS_BLOCK
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_0 [31:30] INPUT = axi_bram_ctrl_0_bram_block_1_combined_0.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_1 [29:28] INPUT = axi_bram_ctrl_0_bram_block_1_combined_1.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_2 [27:26] INPUT = axi_bram_ctrl_0_bram_block_1_combined_2.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_3 [25:24] INPUT = axi_bram_ctrl_0_bram_block_1_combined_3.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_4 [23:22] INPUT = axi_bram_ctrl_0_bram_block_1_combined_4.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_5 [21:20] INPUT = axi_bram_ctrl_0_bram_block_1_combined_5.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_6 [19:18] INPUT = axi_bram_ctrl_0_bram_block_1_combined_6.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_7 [17:16] INPUT = axi_bram_ctrl_0_bram_block_1_combined_7.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_8 [15:14] INPUT = axi_bram_ctrl_0_bram_block_1_combined_8.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_9 [13:12] INPUT = axi_bram_ctrl_0_bram_block_1_combined_9.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_10 [11:10] INPUT = axi_bram_ctrl_0_bram_block_1_combined_10.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_11 [9:8] INPUT = axi_bram_ctrl_0_bram_block_1_combined_11.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_12 [7:6] INPUT = axi_bram_ctrl_0_bram_block_1_combined_12.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_13 [5:4] INPUT = axi_bram_ctrl_0_bram_block_1_combined_13.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_14 [3:2] INPUT = axi_bram_ctrl_0_bram_block_1_combined_14.mem;
  axi_bram_ctrl_0_bram_block_1/axi_bram_ctrl_0_bram_block_1/rmb36e1_15 [1:0] INPUT = axi_bram_ctrl_0_bram_block_1_combined_15.mem;
END_BUS_BLOCK;
END_ADDRESS_RANGE;
END_ADDRESS_SPACE;
END_ADDRESS_MAP;
Outline

- IP Catalog
- Pcore directory
- IP device files
- GP Interfaces
- Adding IP to extend PS into PL
- PlatGen and bitstream generation
- Summary
PS functionality can be extended by instantiating peripherals in PL

Adding IP in PL involves
- Enabling Master/Slave GP interface(s)
- Selecting an IP from IP Catalog
- Connecting it to Master GP or Slave GP ports of PS
- Assigning address
- Configuring IP for a desired functionality
- Connecting IP ports to ports of other peripherals and/or to external pins

PlatGen must be run to generate netlists when an IP is added in PL

PlatGen output is a netlist for each instance in your embedded hardware platform
- The implementation directory holds netlist files
- The synthesis and HDL directories contain all the necessary scripts and wrappers to synthesize your design
PlatGen automatically generates a memory array structure
- Only for block RAM controllers: axi_bram_cntlr and lmb_bram_if_cntlr
- Size is determined by the address space allocated to the memory controller component
- BMM file defines the memory subsystem structure

Bitstream must be generated when PL has any IP
The FPGA must be programmed with the generated hardware bitstream before an application can be run