# **Pizzabox FPGA Programming Workflow**

1. Change a custom core (Verilog/RTL). Open the ISE project in \enc\pquest\_1.srcs\sources\_1\edk\microblaze\pcores\*core*

*\_name\_ver*\devl\projnav\. Build in **ISE**. After running Synthesis, *core\_name\_ver*.ngc file will be created.

1. Update netlist. In **XPS**: import peripheral again, remove module wrapper **\*.ngc** files from implementation folder (\enc\pquest\_1.srcs\sources\_1\edk\microblaze\implementation\ and \enc\pquest\_1.srcs\sources\_1\edk\microblaze\ implementation\cache\) and put OPTION ARCH\_SUPPORT\_MAP = (others =DEVELOPMENT) to the **.mpd** file of the peripheral (e.g., \enc\pquest\_1.srcs\sources\_1\edk\microblaze\pcores\enc\_v1\_00\_a\data\data\enc\_v2\_1\_0.mpd), then **Hardware** 🡪 **Generate Netlist**. (Q: do we need to re-import if no ports/sub-modules are not changed?)

----------------------------------------------------------------------------------------------------------------------------------

Alternatively clean and re-compile the project in order for the changes made to custom peripherals to take effect:

1. **Project -> Rescan User Repositories**   (update custom pcores/peripherals)
2. **Project -> Clean All Generated Files**     (project netlists will be deleted and recompiled using new peripheral definitions)

In order to change the XST Options in XPS, please refer to [(Xilinx Answer 52274)](https://www.xilinx.com/support/answers/52274.html)

----------------------------------------------------------------------------------------------------------------------------------

1. Open **PlanAhead**, add sources from **Platform Studio/Implementation**, add whole directory (which directory?) with netlists (**.ngc**). Import constraints if **.ucf** file was changed. Implement design, generate bitstream. Go to **File** ->**Export bitstream** to hw folder of workspace of SDK (\enc\pquest\_1.sdk\SDK \SDK\_Export\pquest\system.bit or \enc\pquest\_1.srcs\sources\_1\

edk\microblaze\SDK\SDK\_Export \hw\system\_cclktemp.bit?).

1. Export hardware to SDK export directory of XPS if new modules were added or address space allocation was changed. In **XPS** click on Export Design, uncheck “Include bitstream and BMM file”. This will update **.xml** of hw platform in SDK export dir (\enc\pquest\_1.srcs\sources\_1\edk\microblaze\SDK\ SDK\_Export\hw\system.xml). Copy **.xml** to hw platform in EDK workspace (?).

If **mcs** will be generated: in **SDK**: open Project Explorer 🡪 pquest 🡪 **system\_bd.bmm** file and manually edit **ramb36e1** locations. Open implemented design in **PlanAhead**, search for instance \*ramb36e1\*, edit all sites placed. (How to decide the locations?).

To open the existing project in SDK, File 🡪 Import 🡪Existing Projects into Workspace. Select SDK 🡪 SDK\_Export as the root directory, and select the projects. If the workspace has been opened, simply show Project Explorer from Window 🡪 Show View

1. In **SDK** open **XMD console**, cd to hw platform directory (\enc\pquest\_1.sdk\SDK\SDK\_Export\ pquest\), execute

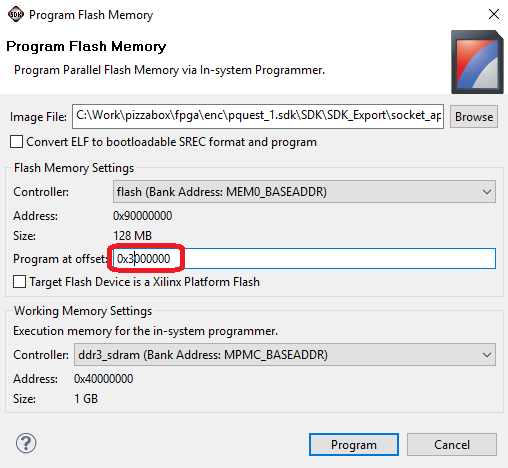
data2mem -bm "system\_bd.bmm" -p xc6vlx240tff1156-1 -bt "system.bit" -bd "boot.elf" -o b pquest.bit

* -bm: check the syntax of a BMM file
* -p: input target part
* -bt: input bitstream (BIT) file
* -bd: imput ELF or MEM file
* -o: output file

Alternatively make boot active project and download FPGA. This will create **pquest.bit** file from merging **boot.elf** and **system.bit**.

Make sure boot was built with heap and stack in BRAM, use **standalone\_bsp** with main project in DDR3 and uses **xil\_kernel\_bsp**.

1. If **mcs** will be generated: In **ImPACT** start page select create a PROM, **BPI -> virtex6 -> 128Mb -> 16bit**. Save and download **.mcs**. Use **28AF00P30** flash, RS pins 26-25. Uncheck verification after programming to save time. Program result mcs (?) file into flash.
2. Cycle the unit, check LED indication for proper firmware load.
3. In **EDK/SDK**, program flash with socket\_apps.elf file. Choose Xilinx Tools 🡪 Program Flash

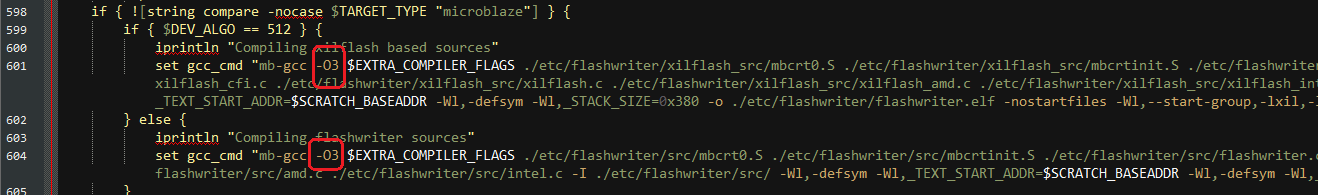


The offset is defined in BPM\_FLASH\_IMAGE\_BASEADDR in SDK\SDK\_Export\boot\src\blconfig.h.

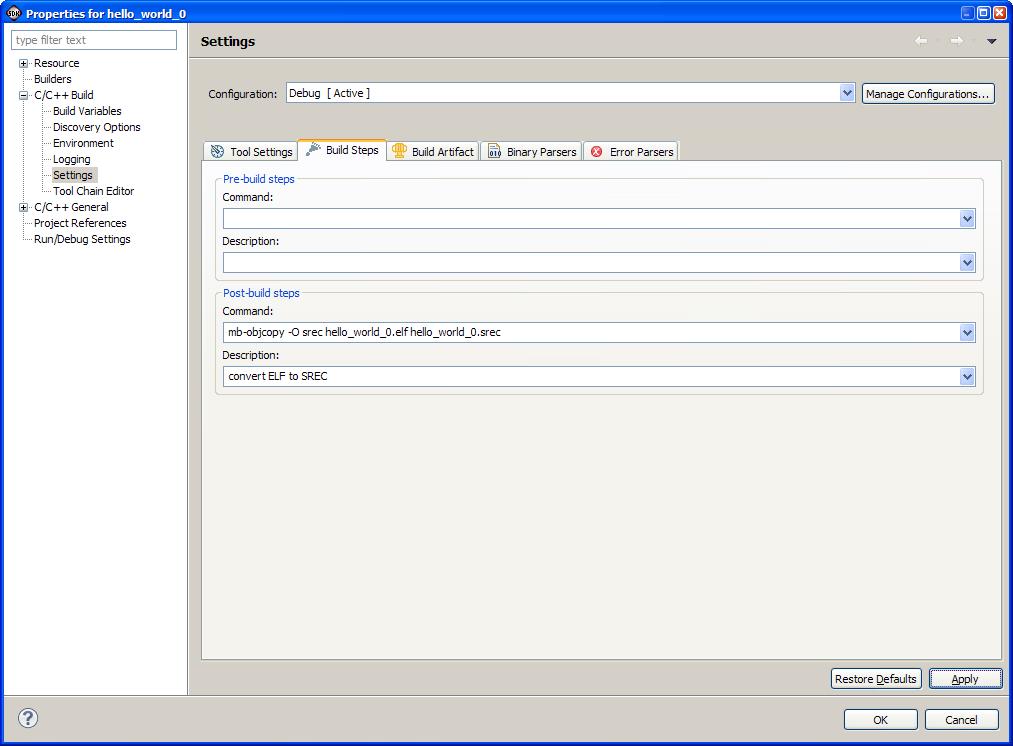
If getting error “CFI query of block/sector map returned inconsistent result!" during programming, in **Xilinx\14.7\ISE\_DS \EDK\data\xmd\**flashwriter.tcl, change all “-Os” to “-O3”

set EXTRA\_COMPILER\_FLAGS  -O3

set gcc\_cmd "powerpc-eabi-gcc -O3 $mcpu ...



SREC file can be generated automatically by executing “mb-objcopy -O srec socket\_apps.elf socket\_apps.srec" as Post-build steps in project settings:



1. Cycle the unit, check the serial console of the unit.

**Peripheral list:**

* enc
* enc\_dummy
* evr
* evr\_dummy
* io\_conf
* io\_scan
* npi\_burst
* pulse\_gen

**Name Type Produced By Description**

BIT Data BitGen Download bitstream file for

devices containing all of the

configuration information from the

NCD file

BGN ASCII BitGen Report file containing information

about a BitGen run

BLD ASCII NGDBuild Report file containing information

about an NGDBuild run, including

the subprocesses run by NGDBuild

DATA C File TRCE File created with the –stamp option

to TRCE that contains timing

model information

DC ASCII Synopsys FPGA

Compiler

Synopsys setup file containing

constraints read into the Xilinx

Development System

DLY ASCII PAR File containing delay information

for each net in a design

DRC ASCII BitGen Design Rule Check file produced

by BitGen

EDIF (various

file extensions)

ASCII CAE vendor’s EDIF 2

0 0 netlist writer.

EDIF netlist. The Xilinx

Development System accepts an

EDIF 2 0 0 Level 0 netlist file

EDN ASCII NGD2EDIF Default extension for an EDIF

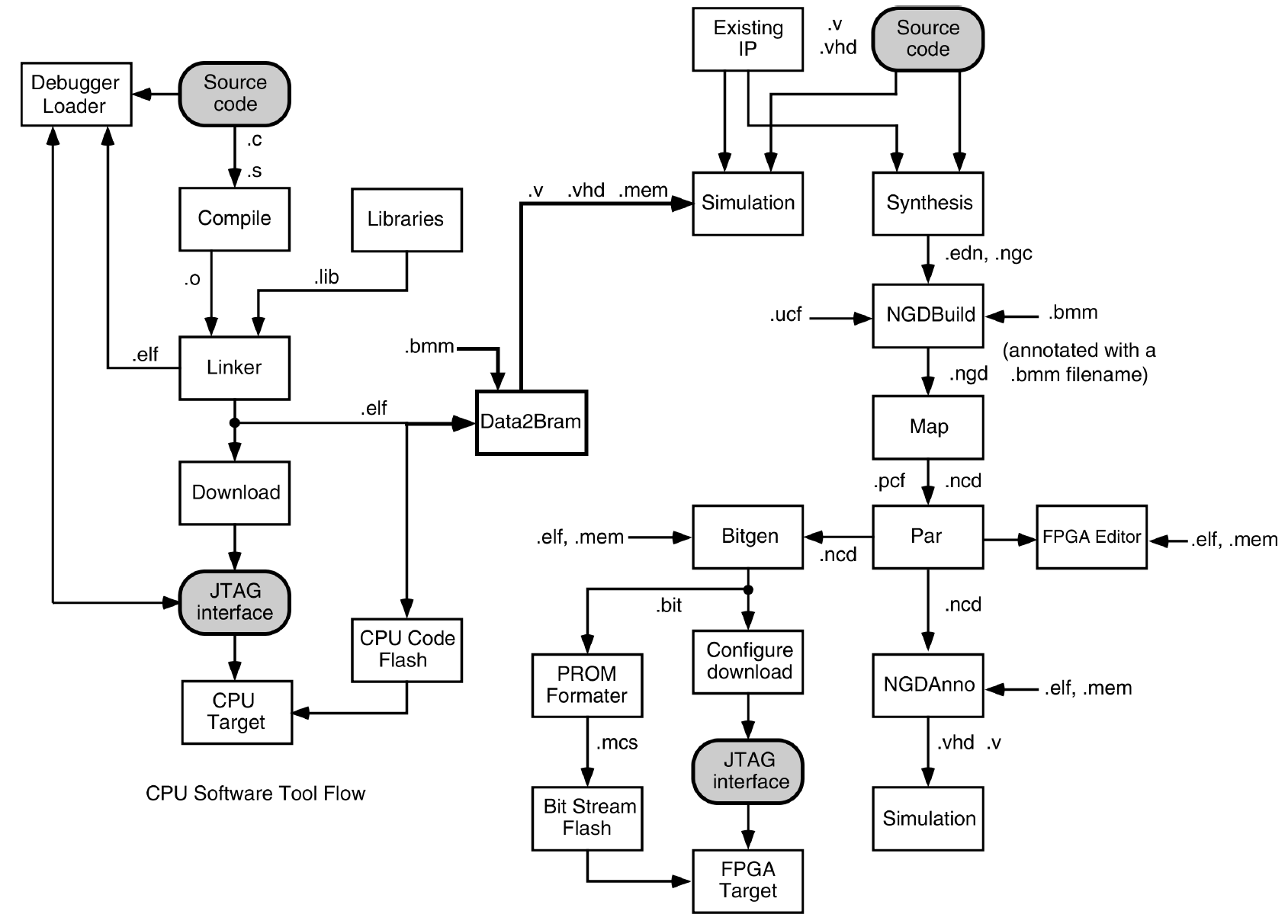
2 0 0 netlist file

ELF ASCII Used for NetGen This file populates the Block RAMs

specified in the .bmm file.

**File types (**[**external link**](https://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_r_gst_project_files.htm)**):**

|  |  |
| --- | --- |
| .bbd | The **Black Box Definition (BBD) file** manages the file locations of optimized hardware netlists for the black-box sections of your peripheral design. |
| .bmm | A **Block Memory Map (BMM) file** is a text file that has syntactic descriptions of how individual block RAMs constitute a contiguous logical data space. When updating the FPGA bitstream with memory initialization data (typically the executable program), the Data2Mem utility uses the BMM file to direct the translation of data into the proper initialization form. Although the BMM file is a text file direct editing is not recommended. This file is generated by the Platform Generator (Platgen) and updated with physical location information by the Bitstream Generator tool (Bitgen). Refer to the *Embedded System Tools Reference Manual*for more information. |
| .bsb | The **Base System Builder (BSB) file** is the playback file generated by the Base System Builder. You can use it to create new projects with pre-defined options. You cannot edit this file. |
| .elf | The **Executable and Linkable Format (ELF)** file is a common standard in computing. An executable or executable file, in computer science, is a file whose contents are meant to be interpreted as a program by a computer. Most often, they contain the binary representation of machine instructions of a specific processor, but can also contain an intermediate form that requires the services of an interpreter to be run. |
| .mcs |  |
| .mdd | A **Microprocessor Driver Description (MDD) file** contains directives for customizing software drivers. |
| .mhs | The **Microprocessor Hardware Specification (MHS) file** defines the hardware component. The MHS file serves as an input to the Platform Generator (Platgen) tool. An MHS file defines the configuration of the embedded processor system, and includes the following:   * Bus architecture * Peripherals * Processor * System Connectivity * Address space |
| .mld | The **Microprocessor Library Definition (MLD)** file. |
| .mpd | **Microprocessor Peripheral Definition (MPD) file** defines the interface for 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. |
| .mss | The **Microprocessor Software Specification (MSS)** file is used as an input file to the Library Generator (Libgen). The MSS file contains directives for customizing OSs, libraries, and drivers. |
| .mui | The **Microprocessor-IP User Interface (MUI) file**. |
| .ncd | The Native Circuit Description (NCD) file, created during Map/Place and Route, physically represents the design mapped to the components in the Xilinx FPGA. |
| .ngc | The NGC file is a netlist that contains logical design data and constraints. This file replaces both EDIF and Netlist Constraints (NCF) files. |
| .ngd | Xilinx native generic database (NGD) file, created during Translate, describes the logical design reduced to Xilinx primitives. |
| .pao | **Peripheral Analyze Order (PAO) file**. A PAO file defines the ordered list of HDL files needed for synthesis and simulation. |
| .xbd | **The Xilinx Board Description (XBD)** file. |

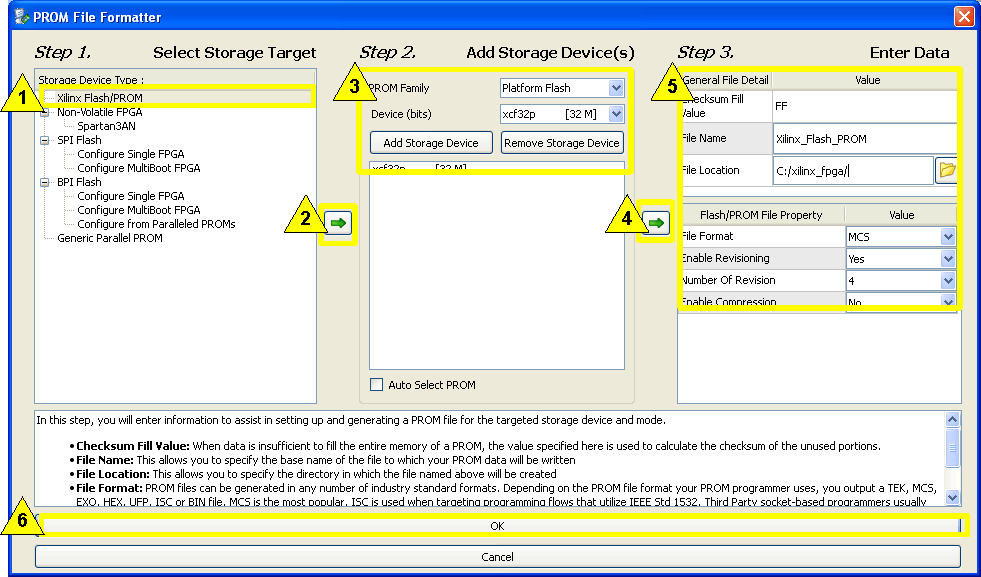


**Creating Xilinx Flash/PROM Files**

The following is the PROM File Formatter flow for generating a PROM file for a Xilinx Platform Flash or PROM device.

Before you run the procedure below, open the PROM File Formatter Wizard by double-clicking **Create PROM File (PROM File Formatter)** in the iMPACT Flows panel.

The PROM File Formatter Wizard opens (see below). A number in the figure indicates the procedure step to which the area applies.



**To Create a Xilinx Flash/PROM File**

1. In **Step 1** of the PROM File Formatter wizard ([**Select Storage Target**](https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/pim_db_pff_storage_targets.htm)), select **Xilinx FLASH/PROM**.
2. Click the arrow to the right of **Step 1**.
3. In **Step 2** of the Wizard (**Add Storage Devices**), for each applicable PROM:
   1. Specify **PROM Family** and **Device (bits)**.
   2. Click **Add Storage Device**.

For multiple PROMs (also called a PROM daisy-chain), the first PROM you enter is located at position 0; the second PROM is assigned position 1. You are limited to a maximum of 64 serial PROMs in a chain.

If you want the correct size PROM automatically chosen for you, select the **Auto Select PROM** checkbox.

**Note**If you choose to have the correct size of PROM automatically chosen for you, advanced PROM features such as revisions and compression will not be available to you.

1. Click the arrow to the right of **Step 2**.
2. In **Step 3** of the Wizard (**Enter Data**), specify the entries in the [**General File Detail**](https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/pim_db_pff_step3_gen_file_detail.htm) section and the [**Flash/PROM File Property**](https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/pim_db_pff_step3_prom_prop_xilinx.htm) section for the PROM file to be created.
3. Click **OK**.
4. In the Add Device dialog boxes, specify the device (bitstream) files to be added to the PROMs.

**Note**The same startup clock restrictions that apply for configuring devices with a cable also apply to adding bitstreams to PROM files. For PROM files, only bitstreams with startup clock settings of CCLK are allowed.

As bitstreams are added, the PROM contents are displayed in the PROM File Formatter window.

1. If you selected the **Add Non-Configuration Data Files** property in the PROM File Formatter Wizard, add data files to the PROM files in this way:
   1. In the Add Data File dialog boxes, specify the data files to add to the PROMs.

**Note**You can specify any file as a data file to be added to the PROM file. Examples are: MCS, BIN, HEX, MEM, ELF, JPG, etc.

When you have finished adding data files, the Data File Assignment dialog box opens.

* 1. In the Data File Assignment dialog box, change data files or Start Addresses as desired.
  2. In the Data File Assignment dialog box, click **OK**.

1. Right-click in the PROM File Formatter window and select **Generate File**.

iMPACT generates the PROM files you specified.

**Implementation Overview for FPGAs**

After synthesis, you run design implementation, which comprises the following steps:

1. Translate, which merges the incoming netlists and constraints into a Xilinx® design file
2. Map, which fits the design into the available resources on the target device
3. Place and Route, which places and routes the design to the timing constraints
4. Programming file generation, which creates a bitstream file that can be downloaded to the device

In the [Sources tab](https://www.xilinx.com/itp/xilinx10/isehelp/pn_r_sources_tab.htm), select **Implementation** from the Design View drop-down list, and select the top module Image. In the [Processes tab](https://www.xilinx.com/itp/xilinx10/isehelp/pn_r_processes_tab.htm), double-click **Implement Design** to run the implementation process in one step, or double-click **Translate**, **Map**, and **Place & Route** to run each of the implementation steps separately. To generate the programming file, double-click **Generate Programming File**. Alternatively, you can select **Process > Implement Top Module** to run Implement Design on the top module. For details, see [Implementing the Top Module](https://www.xilinx.com/itp/xilinx10/isehelp/ise_p_implementing_the_top_module.htm).

Default property values can be set for the Implement Design process or for each of the separate implementation processes. [Image](https://www.xilinx.com/itp/xilinx10/isehelp/ise_p_setting_process_properties.htm) These properties values can also be changed using the Design Goals & Strategies command, available from the Project menu. For details, see [Using Design Goals & Strategies.](https://www.xilinx.com/itp/xilinx10/isehelp/ise_c_design_strategies.htm)

**Note**In addition to the regular implementation flow described here, alternate implementations to save runtime and preserve results can be used. For more details, see the [Partitions Overview](https://www.xilinx.com/itp/xilinx10/isehelp/ds_c_overview.htm) and [Using SmartGuide](https://www.xilinx.com/itp/xilinx10/isehelp/ise_p_using_smartguide.htm).

**Translate**

The Translate process merges all of the input netlists and design constraints and outputs a Xilinx native generic database (NGD) file, which describes the logical design reduced to Xilinx primitives. See the following table for details.

|  |  |
| --- | --- |
| **Translate Process** | |
| Command line tool | NGDBuild |
| Tcl command | **process run "Translate"** |
| Input files | EDIF, SEDIF, EDN, EDF, NGC, UCF, NCF, URF, NMC, BMM |
| Output files | BLD (report), NGD |
| Process properties | [Translate Properties](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_translate_properties.htm) |
| Tools available after running process | Constraints Editor, Floorplan Editor, Floorplanner, PACE  **Note** Each of these tools modifies the UCF file. When you rerun Translate with the updated UCF, the NGD file is updated. |

**Map**

The Map process maps the logic defined by an NGD file into FPGA elements, such as CLBs and IOBs. The output design is a native circuit description (NCD) file that physically represents the design mapped to the components in the Xilinx FPGA. See the following table for details.

|  |  |
| --- | --- |
| **Map Process** | |
| Command line tools | MAP |
| Tcl command | **process run "Map"** |
| Input files | NGD, NMC, NCD, NGM  **Note** The NCD and NGM files are for guiding. |
| Output files | NCD, PCF, NGM, MRP (report), GRF, MAP, PSR |
| Process Properties | [Map Properties](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_map_properties.htm) |
| Tools available after running process | Floorplanner, FPGA Editor, Timing Analyzer [Image](https://www.xilinx.com/itp/xilinx10/isehelp/ise_r_imp_additional_tools.htm) |

**Place and Route**

The Place and Route process takes a mapped NCD file, places and routes the design, and produces an NCD file that is used as input for bitstream generation. See the following table for details.

|  |  |
| --- | --- |
| **Place and Route Process** | |
| Command line tools | PAR |
| Tcl command | **process run "Place & Route"** |
| Input files | NCD, PCF  **Note** In addition to the NCD file from MAP, PAR also accepts an NCD file for guiding. |
| Output files | NCD, PAR (report), PAD, CSV, TXT, GRF, DLY |
| Process Properties | [Place & Route Properties](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_place_and_route_properties.htm) |
| Tools available after running process | Floorplanner, FPGA Editor, Timing Analyzer, TRACE, XPower Analyzer |

**Programming File Generation**

The Generate Programming File process produces a bitstream for Xilinx device configuration. After the design is completely routed, you must configure the device so it can execute the desired function. See the following table for details.

|  |  |
| --- | --- |
| **Generate Programming File Process** | |
| Command line tools | BitGen |
| Tcl command | **process run "Generate Programming File"** |
| Input files | NCD, PCF, NKY |
| Output files | BGN, BIN, BIT, DRC, ISC, LL, MSD, MSK, NKY, ISC, RBA, RBB, RBD, RBT |
| Process Properties | [General Options](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_general_options.htm), [Configuration Options](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_configuration_options.htm), [Startup Options](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_startup_options.htm), [Readback Options](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_readback_options.htm), [Encryption Options](https://www.xilinx.com/itp/xilinx10/isehelp/pp_db_encryption_options.htm) |
| Tools available after running process | iMPACT |

**Additional Resources**

Additional details are available in the following Xilinx® documentation:

|  |  |
| --- | --- |
| **Documentation** | **Topics Covered** |
| [Development System Reference Guide](https://www.xilinx.com/itp/xilinx10/books/docs/dev/dev.pdf) | Command line tools, Tcl command information |

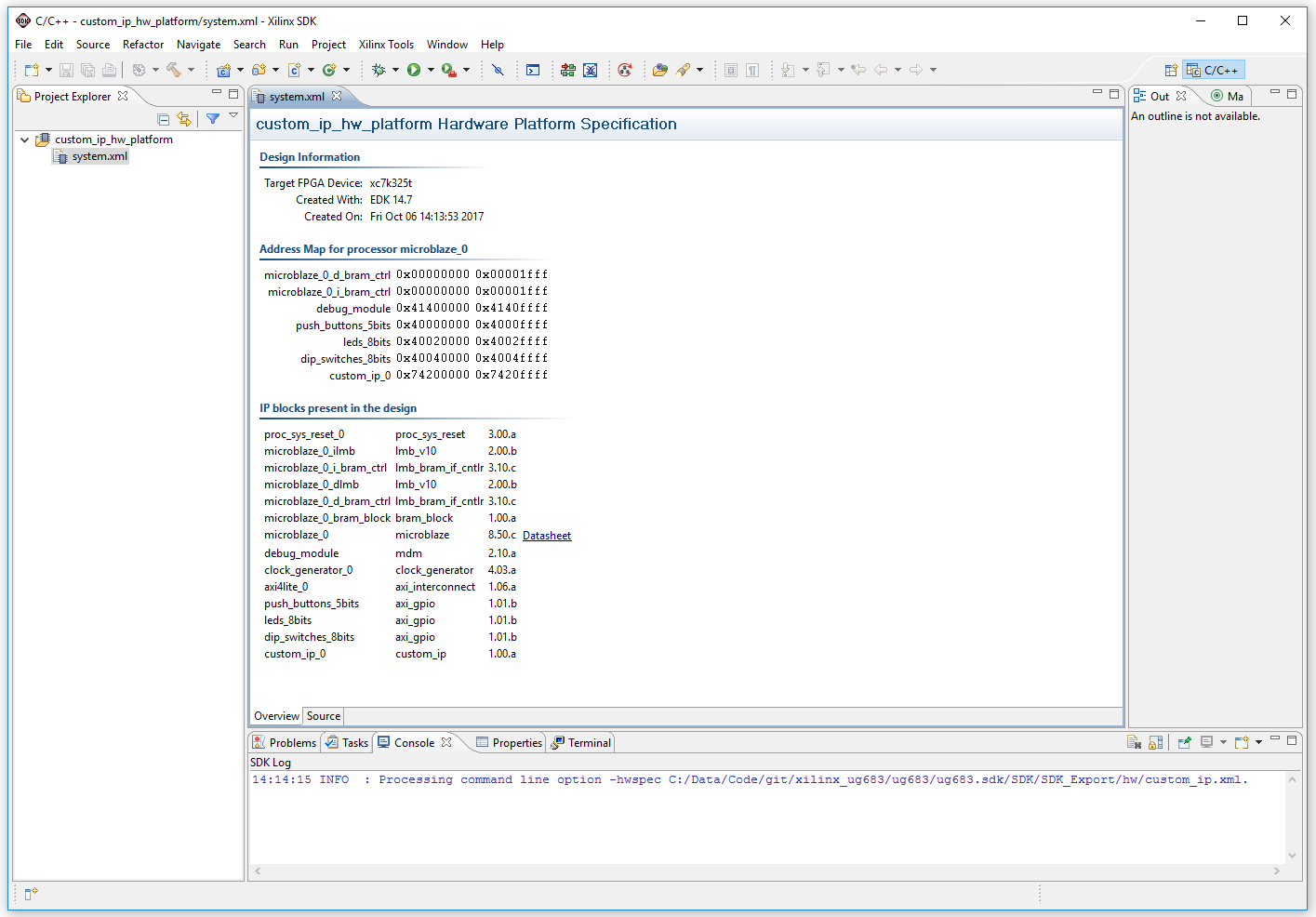
1. In PlanAhead, choose File 🡪 Add sources, 🡪 After or Create Embedded Sources. Create a sub-design named system. It will create system.xmp in <project\_directory>\<project>.srcs\sources\_1\edk\<subdesign >
2. In XPS, choose Hardwre 🡪 Create or Import Peripheral…. After creating a peripheral, a folder <peripheral\_version> is created in <project\_directory>\<project>.srcs\sources\_1\edk\<subdesign\_name>\pcores\. It creates a top-level subdesign.vhd file in hdl\vhdl\ folder, and a user\_logic.v file in hdl\verilog\. Add the ports of the subdesign in subdesign.vhd, and customized logic in user\_logic.v.

**PlanAhead Project Data Outputs**

|  |  |
| --- | --- |
| **Output** | **Description** |
| Project Directory  <*project*> | When you create a new project, the PlanAhead tool creates a project directory in which to store the project file, the project data directory, and the ISE Implementation results. The project directory has the same name as the project name entered in the New Project wizard. |
| Project File  (*project*.ppr) | The project file has the same name as the project name entered in the New Project wizard. |
| Cached Library Data  (*project*.cache) | Contains a cached version of the Xilinx CoreLib used when running simulation. |
| Project Data Directory  (*project*.data) | Contains project XML metadata files for HDL source files, design constraints, and simulation source files. |
| Project Data - Netlist  Subdirectory  (project.data/  netlist) | A /netlist subdirectory containing a copy of the netlist files for the design.  For RTL-based projects, the PlanAhead tool creates a /Synthesis subdirectory for each run for the produced netlist, which refreshes each time the run is reset.  For Netlist-based projects, the PlanAhead tool creates a single netlist directory that contains the imported netlist, including copies of all NGC core files used in the design. |
| Project RTL Directory  (*projectname*.srcs) | The project sources directory stores the HDL source files, and UCF constraint files, that are imported into a project. These folders are maintained by the PlanAhead tool and do not require your attention.  ***Caution!*** Modifying any of these files could result in project data corruption |

# Custom peripheral flow

1. Create a template peripheral in XPS
   1. Hardware 🡪 Create and import peripheral, select Create to create a peripheral named custom\_ip. You should be able to see the IP in IP Catalog under Project Local PCores 🡪 USER.
   2. Revise custom\_ip.vhd. File 🡪 Open, go to folder pcores\custom\_ip\_v1\_00\_a\hdl\vhdl, open custom\_ip.vhd.
      1. Add the top level entity port declaration
      2. Add the port map for the instantiation of the user\_logic
   3. Revise user\_logic.vhd
      1. Add the button’s port declaration
      2. Add signal declaration
      3. Remove unused example code
      4. Add user logic
   4. Update the MPD file of the custom\_ip. Right click CUSTOM\_IP in IP Catalog, choose View MPD. Add declaration of the port added to the VHDL files.
   5. Project 🡪 Rescan User Repositories to update the IP.
   6. Insert the IP core, connect busses and ports in System Assembly View 🡪 Ports
   7. Update the UCF file for pin allocation
   8. Run DRC: Project 🡪 Design Rule Check. Exit XPS when finish.
2. Write software in SDK.
   1. In PlanAhead, File 🡪 Export 🡪 Export Hardware for SDK. Select Start SDK.



* 1. In SDK, File 🡪 New 🡪 Application Project.
  2. In Templates, select Empty Application.
  3. In Project Explorer, right click <project>, select New 🡪 C Source File.
  4. Find related parameter definitions in <project\_bsp>\microblaze\_0\include\xparameters.h. Edit the new C file.

# ENC Debug

## Set TEMAC speed:

Check configure\_IEEE\_phy\_speed in xilkernel\_bsp\_0\microblaze\_0\libsrc\lwiip140\_v1\_03\_a\contrib\ports\xilinx\netif\xlltemacif\_physpeed.c