

Generating SVF Files in EDK

The following documentation explains how to generate SVF files for use by an SVF player. This could be a hardware or software player.

This document contains the following sections:

- Using `genace.tcl` to generate an SVF file
- Defining A Custom Board or Changing XMD Options
- Multi-Processor SVF files
- Multi-Device SVF files
- Using XMD to create SVF Files
- Playing SVF files via Parallel Cable 3 or Parallel Cable 4

Using genace.tcl to generate an SVF file

EDK provides a Tcl script, genace.tcl, which is used to generate SVF files. In the process of generating the ACE files, genace.tcl creates a SVF file. The genace.tcl file, delivered with EDK 6.1, can be used to generate five basic types of SVF files.

The flow diagram show in figure 1, explains each of the steps which genace.tcl runs.

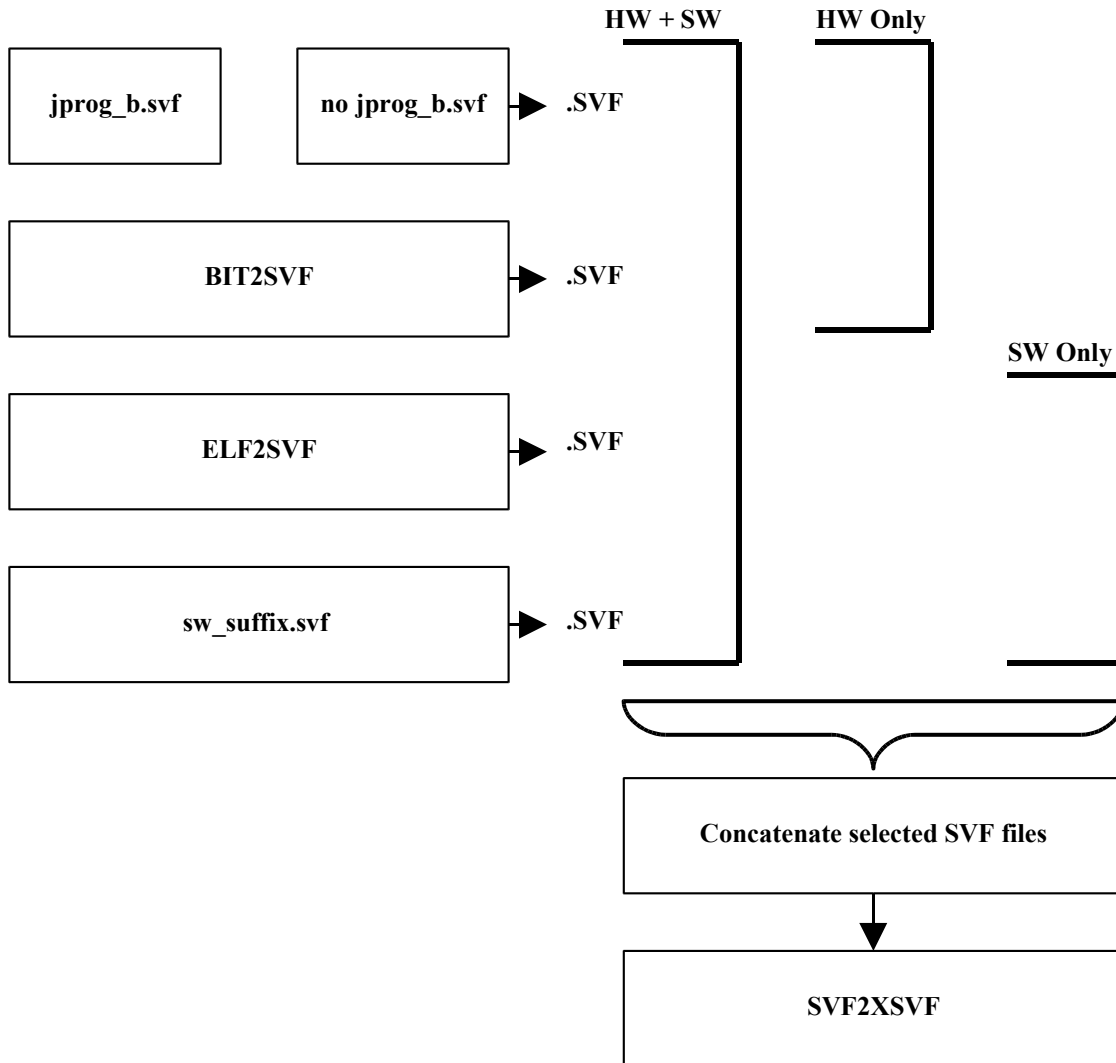


Figure 1. genace.tcl flow diagram

As illustrated by the flow diagram, the genace.tcl file can be used to generate five basic types of SVF files. A description for each of the SVF files created and an example of the command line used are shown below.

Hardware Only SVF

The first step in generating a hardware only SVF file is the inclusion of JTAG commands to reset the FPGA. This functionality is included in the jprog_b.svf file.

A SVF file representing the BIT file can be created using BIT2SVF. The resulting SVF file is concatenated to the jprog_b.svf file. It should be noted that the bitstream used to create the SVF file has either a bootloop.elf in BRAM or some small program located in BRAM.

For additional information on adding bootloop.elf refer to the bootloop section.

Command Line:

```
xmd -tcl genace.tcl -jprog -hw <bitstream> -board <target board> -ace <output ACE/SVF file>
```

Hardware + Software SVF

The first step in generating a hardware + software SVF file is the inclusion of JTAG commands to reset the FPGA. This functionality is included in the jprog_b.svf file.

A SVF file representing the BIT file can be created using BIT2SVF. The resulting SVF file is concatenated to the jprog_b.svf file.

It should be noted that the bitstream used to create the SVF file has either bootloop.elf in BRAM or some small program in BRAM.

The second step is conversion of the software ELF file to an SVF file. This consists of two steps:

- a. A SVF file which downloads the ELF file
- b. A SVF file which runs the ELF file

ELF2SVF creates an SVF file which downloads the ELF file.

The sw_suffix.svf file is the SVF file, which begins execution of the downloaded ELF image.

Each of these SVF files is then merged. This SVF file will reset and configure the FPGA, then download the ELF image and begin execution of that image.

Command Line:

```
xmd -tcl genace.tcl -jprog -hw <bitstream> -elf <elf files> -board <target board> -ace <output ACE/SVF file>
```

Software Only SVF

This SVF file requires the FPGA to be configured and running.

The first step is conversion of the software ELF file to an SVF file. This consists of two steps:

- a. A SVF file which downloads the ELF file
- b. A SVF file which runs the ELF file

ELF2SVF creates an SVF file, which downloads the ELF file.

The sw_suffix.svf file is the SVF file, which begins execution of the downloaded ELF image.

Each of these SVF files is then merged and converted into a single SVF file. This SVF file will then download the ELF image and begin execution of that image.

Command Line:

```
xmd -tcl genace.tcl -elf <elf files> -board <target board> -ace <output ACE/SVF file>
```

Hardware Only SVF with No jprog_b reset

The first step in generating a hardware + software SVF file is generation of a SVF file from the bit file.

BIT2SVF is the utility used to do this conversion. It should be noted that jprog_b.svf is not included, hence the program pin will not be toggled. This allows for partial reconfiguration.

It should be noted that the bitstream used to create the SVF file has either bootloop.elf in BRAM or some small program in BRAM.

Command Line:

```
xmd -tcl genace.tcl -hw <bitstream> -board <target board> -ace <output ACE/SVF file>
```

Hardware and Software SVF with No jprog_b reset

The first step in generating a hardware + software SVF file is generation of a SVF file from the bit file.

BIT2SVF is the utility used to do this conversion. It should be noted that jprog_b.svf is not included, hence the program pin will not be toggled. This allows for partial reconfiguration.

It should be noted that the bitstream used to create the SVF file has either bootloop.elf in BRAM or some small program in BRAM.

The second step is conversion of the software ELF file to an SVF file. This consists of two steps:

- a. A SVF file which downloads the ELF file
- b. A SVF file which runs the ELF file

ELF2SVF creates an SVF file which downloads the ELF file.

The sw_suffix.svf file is the SVF file, which begins execution of the downloaded ELF image.

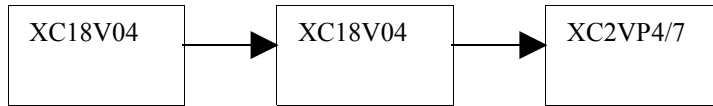
Command Line:

```
xmd -tcl genace.tcl -hw <bitstream> -elf <elf files> -board <target board> -ace <output ACE/SVF file>
```

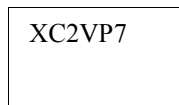
Supported Boards

The genace.tcl script supports two boards:

- 1) Memec 2VP4/7 FG456: This board has the following devices in the JTAG chain



- 2) ML300: This board has the following devices in the JTAG chain



Note: If your software is placed in OCM memory or Cache, the default XMD option in the genace.tcl script will need to be updated.

Additional information on defining a custom board can be found in the Defining A Custom Board or Changing XMD Options section.

Defining A Custom Board or Changing XMD Options

The default XMD options do not include the use of OCM memory or the use of Cache. Applications using either OCM memory or Cache memory require changes to the XMD options. You can define your own board or add additional XMD options by making the modifications listed below.

- 1) Copy the `genace.tcl` from the `<EDK>\data` directory to your project directory. Do not modify the installed script.
- 2) Open `genace.tcl` in the project directory.
- 3) Search for “# Setting some board specific options”
- 4) Copy and paste the `memec` board, as it can be used as an example. There are three lines associated with the definition. Each line is explained below.

Line 1

```
set memec(xmd_powerpc_options) "-configdevice devicenr 1 irlength 8
partname xc18v04 -configdevice devicenr 2 irlength 8 partname
xc18v04 -configdevice devicenr 3 idcode 0x123E093 irLength 10
partname xc2vp4 -debugdevice devicenr 3 cpunr 1"
```

Modify this line to define the JTAG chain on the custom board or add additional XMD options.

Table 2, contains the `irLength` for each device used in defining the JTAG chain.

Device	# PPC Cores	PPC405 IR Length
XC2VP2	0	6
XC2VP4	1	10
XC2VP7	1	10
XC2VP20	2	14
XC2VP30	2	14
XC2VP40	2	14
XC2VP50	2	14
XC2VP70	2	14
XC2VP100	2	14
XC2VP125	4	22

Table 2. `irLength` for PPC405 Cores

The following is an explanation of these switches and their associated options:

-configdevice : used to define a specific device on the JTAG chain. A separate “-configdevice” switch is required for each device in the chain. The “-configdevice” switch supports the following options:

- **partname** <devicename>
 - Name of the device
- **devicenr** <device position>
 - Position of the device in the JTAG chain
- **irlength** <length of the JTAG Instruction Register>
 - Length of the IR register of the device. This information can be found in the device BSDL file.
- **idcode** <device idcode>
 - JTAG Idcode of the device

-debugdevice : used for configuration and debug specifications of a specific CPU. Each “-debugdevice” switch defines the device containing the CPU as well as the CPU number being debugged. The “-debugdevice” switch supports the following options:

- **devicenr** <PowerPC device position>
 - Position of the VirtexIIPro device containing the PowerPC, in the JTAG chain
- **cpunr** <CPU Number>
 - ID of the specific PowerPC to be debugged in a VirtexIIPro containing multiple

PowerPC processors

The following options allow users to map special PowerPC features like ISOCM, Caches, TLB, DCR registers, etc. to unused memory addresses. The debugger can then access these special functions using these unused memory addresses. This is helpful for reading and writing to these registers/memory from GDB or XMD. These addresses are also used to preload the OCM memory or PPC caches.

Note that, these options **do not** create any real memory mapping in hardware.

- **icachestartadr** <I-Cache start address>
 - Start address for reading or writing the instruction cache contents
- **dcachestartadr** <D-Cache start address>
 - Start address for reading or writing the data cache contents
- **itagstartadr** <I-Cache start address>
 - Start address for reading or writing the instruction cache tags
- **dtagstartadr** <D-Cache start address>
 - Start address for reading or writing the data cache tags
- **isocmstartadr** <ISOCM start address>
 - Start address for the ISOCM
- **isocmsize** <ISOCM size>
 - Size of the ISBRAM memory connected to the ISOCM interface
- **isocmdcrstartadr** <ISOCM DCR address>
 - DCR address corresponding to the ISOCM interface specified using the TIEISOCMDCRADDR signals on PowerPC
- **tlbstartadr** <TLB start address>
 - Start address for reading and writing the Translation Look-aside Buffer
- **dcrstartadr** <DCR start address>
 - Start address for reading and writing the Device Control Registers. Using this option, the entire DCR address space (210 addresses) can be mapped to addresses starting from <dcrstartadr> for debugging purposes from XMD and GDB

One example of these special option is the use of ISOCM memory. If the application uses Instruction Side OCM Memory, at address 0xFC000000, additional XMD options must be included. The additional XMD options are listed in **bold**. The following is an example configuration:

```
set memec(xmd_powerpc_options) "-configdevice devicenr 1 irlength 8
partname xc18v04 -configdevice devicenr 2 irlength 8 partname
xc18v04 -configdevice devicenr 3 idcode 0x123E093 irLength 10
partname xc2vp4 -debugdevice devicenr 3 cpunr 1 isocmstartadr
0xFC000000 isocmsize 16384 isocmdcrstartadr 0x18 dcrstartadr
0x78000000"
```

For additional information regarding each of these options, refer to the Embedded Systems Tool Guide chapter 13 included in the EDK Installation.

Line 2

```
set memec(jtag_v2p_position) 3
```

Modify this line to specify the JTAG position of the device containing the PPC405 cores.

Line 3

```
set memec(jtag_devices) "xc18v04 xc18v04 xc2vp4"
```

Modify this line to specify the JTAG devices in the chain.

- 5) If you are using the jprog_b option, the following must be modified.
- a) Search for the following string, "Send into bypass and idle" as shown below:

```
// Send into bypass and idle until program latency is over  
// 4 us per frame. XC2VP7 is 1320 frames = 5280 us  
// @maximum TCK freq 33 MHz = 174240 cycles  
SIR 10 TDI (3ff) SMASK (3ff) ;  
RUNTEST 200000 TCK ;
```

- b) Change the Shift Instruction Register (SIR) register length to match your device. Refer to table2 for the IR length for each device.
- c) Change the RUNTEST delay to match your device requirements using the following equation:

$$\text{Delay(us)} = \# \text{ of frames} * 4\text{us/frame} * 33\text{MHz (max TCK frequency)}$$

- 6) Table 3, contains the # of frames for each device.

Device	# of frames
XC2VP2	884
XC2VP4	884
XC2VP7	1320
XC2VP20	1756
XC2VP30	1756
XC2VP40	2192
XC2VP50	2628
XC2VP70	3064
XC2VP100	3500
XC2VP125	3936

Table 3. # of frames in each Virtex-II Pro

Multi-Processor SVF files

Many of the Virtex-II Pro devices contain two or more processors. In general, each of these processors requires a separate ELF file. This section describes the use of `genace.tcl` in creating these SVF files. Figure 2 is the example system for which an SVF file will be built.

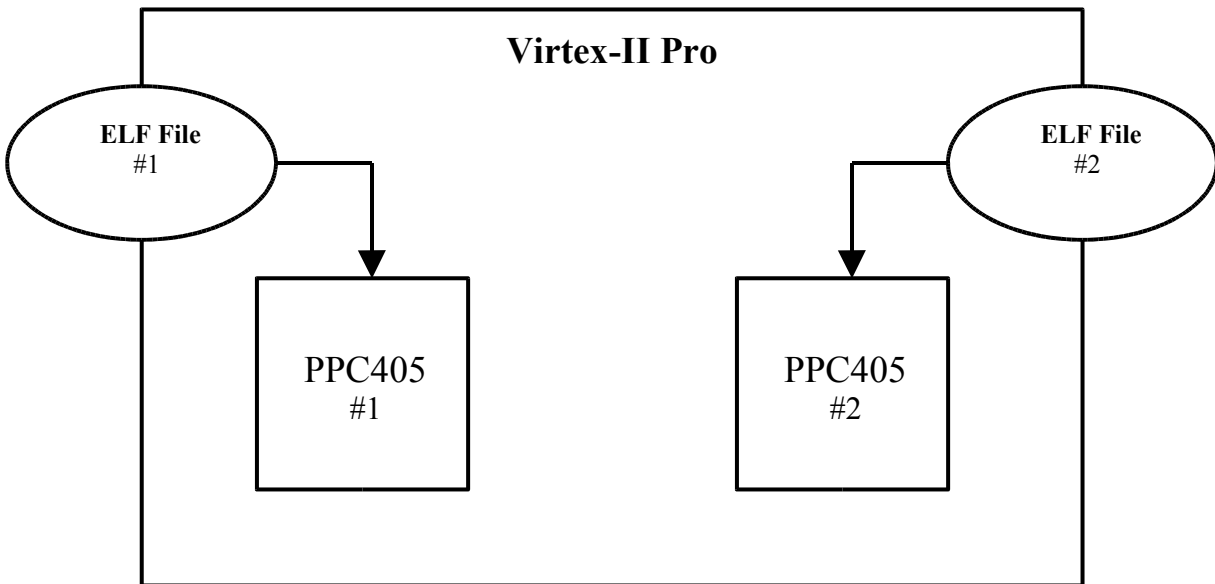


Figure 2.

Figure 3, illustrates the flow diagram used to create an SVF file for the example system.

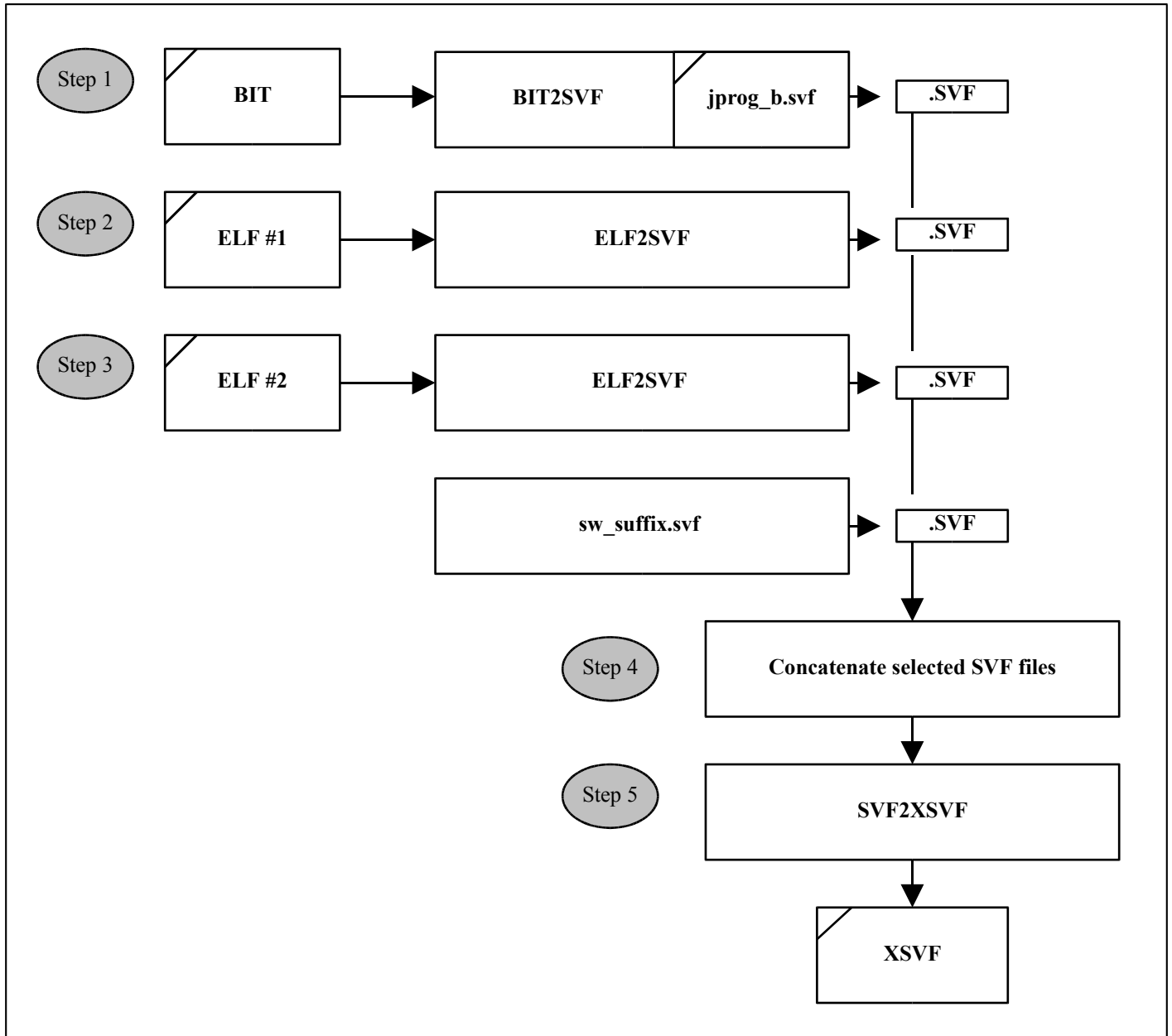


Figure 3. Flow Diagram

Each of the steps, with the associated commands, illustrated in figure 3 is outlined below.

Step 0 – Defining a Custom Board

- Refer to the designing a custom board section to update the script to match the hardware platform being used.

Step 1 – Generate a SVF file from the BIT file

- Run the following genace.tcl command in xygwin shell:

```
xmd -tcl genace.tcl -jprog -hw <bitstream> -board <target
board> -ace <output ACE/SVF file>
```

- Two files will be generated, a SVF and ACE file. The ACE file can be deleted since it will not be utilized.

Note: The jprog_b.svf file is included in the generated SVF file.

Step 2 – Generate a SVF file from the first ELF file

- Open the genace.tcl script.
- Go to the *Board Specific Options* section.
- Update the debug device number to specify the first processor:

```
-debugdevice devicenr 1 cpunr 1
```

- Run the following genace.tcl command in xygwin shell:

```
xmd -tcl genace.tcl -jprog -elf <elf files> -board <target
board> -ace <output ACE/SVF file>
```

- Two files will be generated, a SVF and ACE file. The ACE file can be deleted since it will not be utilized.

Step 3 – Generate a SVF file from the second ELF file

- Open the genace.tcl script.
- Go to the *Board Specific Options* section.
- Update the debug device number to specify the second processor:

```
-debugdevice devicenr 1 cpunr 2
```

- Run the following genace.tcl command in xygwin shell:

```
xmd -tcl genace.tcl -jprog -elf <elf files> -board <target
board> -ace <output ACE/SVF file>
```

- Two files will be generated, a SVF and ACE file. The ACE file can be deleted since it will not be utilized.

Step 4 – Concatenate the generated SVF files

- Concatenate the three generated SVF file into one SVF file. Maintain the order defined by the flow diagram shown in figure 3.

Multi-Device SVF files

The creation of Multi-Device SVF file flow follows the same process as the Multi-Processor SVF files flow. This section describes the use of genace.tcl in creating these SVF files. Figure 4 is the example system for which a SVF file will be built.

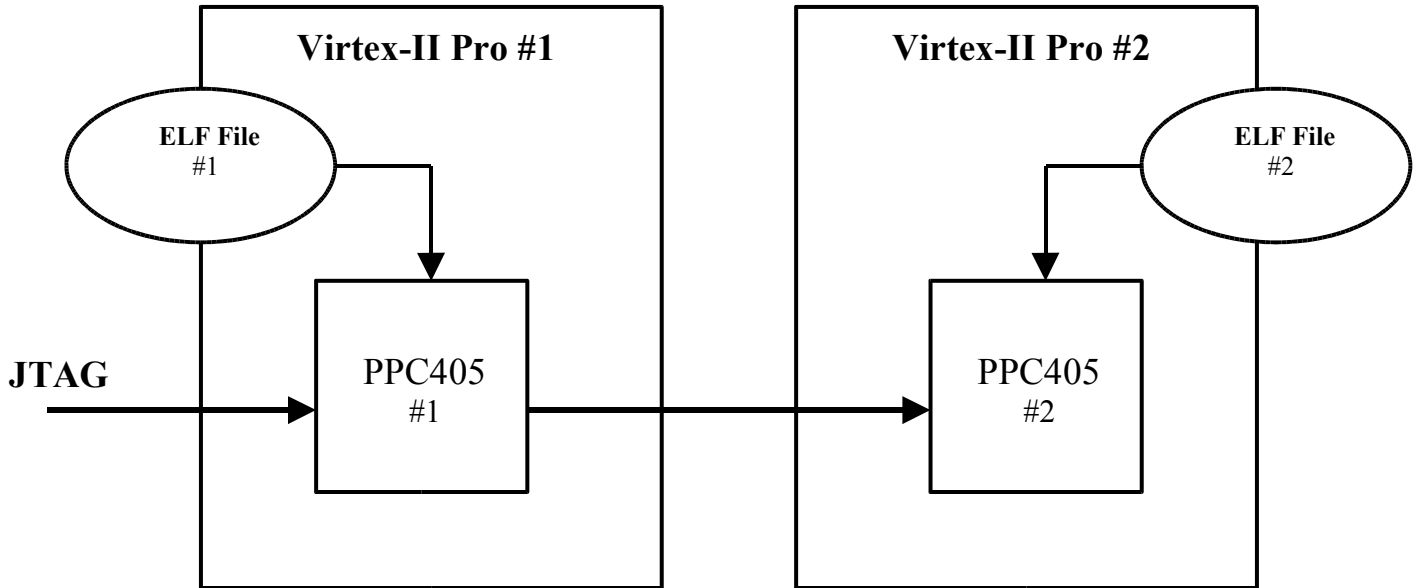


Figure 4.

Figure 5, illustrates the flow diagram used to create a SVF file for the example system.

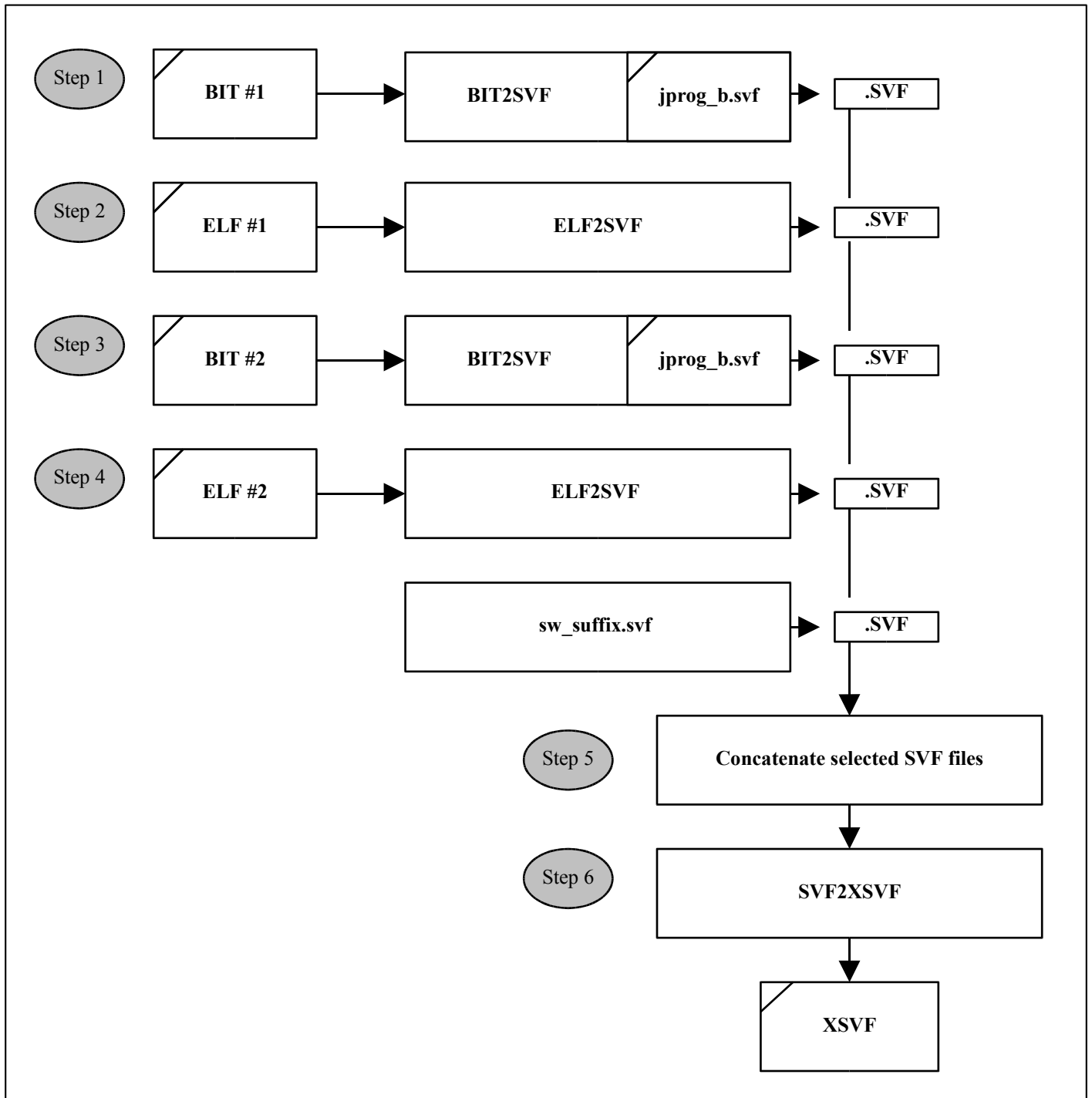


Figure 5. Flow Diagram

Each of the steps, with the associated commands, illustrated in figure 5 is outlined below.

Step 0 – Defining a Custom Board

- Refer to the designing a custom board section to update the script to match the hardware platform being used.

Step 1 – Generate a SVF file from the first BIT file

- Run the following genace.tcl command in xygwin shell:

```
xmd -tcl genace.tcl -jprog -hw <bitstream> -board <target board> -ace <output ACE file>
```
- Two files will be generated, a SVF and ACE file. The ACE file can be deleted since it will not be utilized.

Note: The jprog_b.svf file is included in the generated SVF file.

Step 2 – Generate a SVF file from the first ELF file

- Open the genace.tcl script.
- Go to the *Board Specific Options* section.
- Update the debug device number to specify the first device and first processor:

```
-debugdevice devicenr 1 cpunr 1
```
- Run the following genace.tcl command in xygwin shell:

```
xmd -tcl genace.tcl -jprog -elf <elf files> -board <target board> -ace <output ACE file>
```
- Two files will be generated, a SVF and ACE file. The ACE file can be deleted since it will not be utilized.

Step 3 – Generate a SVF file from the second BIT file

- Run the following genace.tcl command in xygwin shell:

```
xmd -tcl genace.tcl -jprog -hw <bitstream> -board <target board> -ace <output ACE file>
```
- Two files will be generated, a SVF and ACE file. The ACE file can be deleted since it will not be utilized.

Note: The jprog_b.svf file is included in the generated SVF file.

Step 4 – Generate a SVF file from the second ELF file

- Open the genace.tcl script.
- Go to the *Board Specific Options* section.
- Update the debug device number to specify the second device and the first processor:

```
-debugdevice devicenr 2 cpunr 1
```
- Run the following genace.tcl command in xygwin shell:

```
xmd -tcl genace.tcl -jprog -elf <elf files> -board <target board> -ace <output ACE file>
```
- Two files will be generated, a SVF and ACE file. The ACE file can be deleted since it will not be utilized.

Step 5 – Concatenate the generated SVF files

- Concatenate the four generated SVF file into one SVF file. Maintain the order defined by the flow diagram shown in figure 5.

Using XMD to create SVF Files

A SVF file containing only the software ELF file can be generated using the XMD command line. The following XMD command line session, with comments included, illustrates the required commands to generate and SVF file:

```
puts " "  
puts "**** Generates an SVF file from the ELF file ****"  
puts " "  
  
connect ppc hw -cable type xilinx_svffile fname my.svf \  
-configdevice devicenr 1 partname XC2VP7 irlength 10 idcode 0x0124a093 \  
-debugdevice devicenr 1 isocmstartadr 0xFC000000 isocmsize 16384 \  
isocmdcrstartadr 0x18 dcrstartadr 0x78000000  
  
puts " "  
puts " *** Reset the PPC ***"  
puts " "  
rst  
  
puts " "  
puts " *** Download the ELF file ***"  
puts " "  
dow ppc405_i/code/executable.elf  
  
puts " "  
puts " *** Run the code *** "  
puts " "  
run  
  
exit
```

Playing SVF files via Parallel Cable 3 or Parallel Cable 4

One important feature of SVF file generation is the ability to verify the SVF file. This can be done using a software SVF player. In Application Note 58 (XAPP058), a software SVF player is made available and can be used to test the SVF file generation process. The following steps outline how this verification is done.

- 1) Download the conversion tool (svf2xsvf502.exe) and player (playxsvf501.exe) from the ftp site.

ftp://ftp.xilinx.com/pub/swhelp/cpld/eisp_pc.zip

Note: For additional information regarding svf2xsvf502.exe and playxsvf501.exe executables refer to XAPP058.

- 2) Unzip the files to a temporary directory.
- 3) Copy svf2xsvf502.exe and playxsvf501.exe from the <temp_dir>\xapp058_v5.01\bin\nt to the project directory.
- 4) In XPS, select **Xygwin shell**
- 5) To convert the SVF file to a XSVF file, enter the following command:

```
svf2xsvf502 -fpga -extensions -i <input_file>.svf -o <output_file>.xsvf
```

- 6) To download to the target, connect the Parallel Cable 3 or Parallel Cable 4.
- 7) Turn the power to the board on.
- 8) Verify that the parallel port is not in use by any other applications.
- 9) If the SVF file contains software only, configure the FPGA.
- 10) Play the XSVF file by entering the following command in the xygwin shell:

```
playxsvf501 <output_file>.xsvf
```

