Project 5 Scripts Help

18-765: Digital Systems Testing and Testable Design

For any questions or further information about how the items discussed in this document fit into Project 5, please reference *Project 5 Tutorial: Benchmark Circuit s27*, and the help manuals for Mentor Fastscan.

Table of Contents

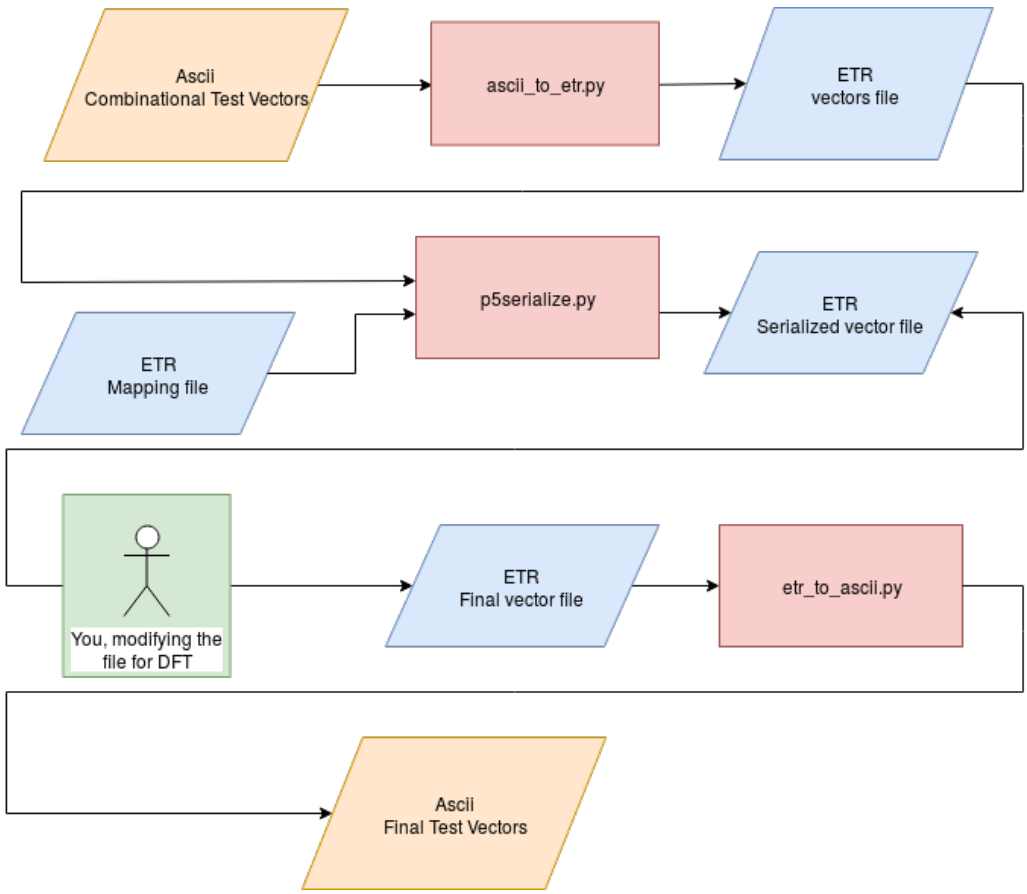
Introduction

Purpose

One of the most important portions of this course's Project 5 comes when you must create the test vectors for your newly-designed DFT circuit. The test vectors will likely be applied in a manner that is different from the original circuit, which requires manipulation of the original test vectors to conform to your design. These scripts are designed to aid in that manipulation.

Process

These scripts were designed to help in one primary process. When working with a scan-based design you will need to perform combinational ATPG using Mentor Fastscan. After this is complete, you must change the application or ordering of the bits in the vectors to match your DFT design. The first script, *p5convert.py* will convert the Mentor Fastscan test vector format to one that is more-easily workable. Then, *p5serialize.py* will serialize those test vectors according to your specifications. Then, after the serialized vectors have been further modified to match your design, they are converted back to the format which is accepted by Mentor Fastscan. This general flow for Project 5's use of these scripts is described graphically below.



Convert Between File Types: *etr_to_ascii.py, ascii_to_etr.py, and etr_to_verilogTB.*

When working with Fastscan for most applications, all test vectors and testing sequences are handled internally. For this course's Project 5, this is not the case. As a student completing this project, you will need to read, analyze, and manipulate these test vectors. Thus, the vectors must be represented in a way that allows this to occur.
Mentor is capable of writing the test vectors it creates to a file, but the formats are not as easily understandable and workable as one would like.

For this purpose, we have provided you with two conversion scripts.

ascii_to_etr.py - Usage

This script converts a combinational ascii file to the easy-to-read (etr) format which is easier to modify.

Input: combinational ascii file
Output: etr file

Syntax:

```
python ascii_to_etr.py <input combinational ascii file> <output
                                                    etr file>
```

Further information and details about any of the files discussed in this section can be found toward the end of this document.

etr_to_ascii.py - Usage

This script converts an (easy-to-read) etr file to an ascii test file which can be read by fastscan.

Input: etr file
Output: ascii file

Syntax:

```
python etr_to_ascii.py  <input etr file> <output ascii file>
```

Further information and details about any of the files discussed in this section can be found toward the end of this document.

etr_to_verilogTB.py - Usage

This script converts an (easy-to-read) etr file to a verilog test bench which can be simulated. The purpose of the test bench is to debug your input sequences and visualize the functionality.

Input: etr file
Output: verilog file

Syntax:

```
python etr_to_verilogTB.py  <input etr file> <output verilog file>
                                    <module name>
```

Further information and details about any of the files discussed in this section can be found toward the end of this document.

Serialize Test Vectors: *p5serialize.py*

Project 5 will require you to implement scan in some form. However, the test vectors created by Fastscan will not be generated for this type of circuit. ATPG must be performed on the purely combinational circuit at the core of your updated DFT design. In order to properly apply these test vectors they must be serialized and placed in correct order to ensure that the correct nets within the circuit are obtaining the correct values placed on the input scan pins.

The purpose of the *p5serialize.py* script is to perform this serialization. This is accomplished by accepting two separate files. The first containing the test vectors to be serialized, and the second containing information about the manner in which the serialization should occur.

NOTE: In order to perform serialization, the script must make an assumption about the manner in which bits are applied to your design. This assumption applies to the most common use of scan in a design. Your circuit may be different. In this case the resulting serialized *Easy-to-Read* file will not match your circuit.
In this instance there are two possible options. You may use the *p5serialize.py* script as it functions now, and make manual modifications to the obtained serialized file. Instead of this, you may also copy the script to your own workspace and alter it to print the test vectors in accordance with your circuit.

Usage

This script accepts an *Easy-to-Read* file (not serialized), and an *Easy-to-Read* mapping file, and creates a serialized version of the file according to the defined mapping. Note that the inputs must be in the correct order for the script to function as desired.

Inputs: etr file, mapping file
Output: serialized etr file

Syntax:

```
./p5serialize.py <inputetrfile> <inputmapfile> <outfilename>
```

File Type Descriptions

ASCII

ASCII files are present in this project to allow the user to work with Mentor Fastscan in the writing, reading, and simulation of test vectors. The *ASCII* files are capable of being both written out and read in by Mentor Fastscan. This makes *ASCII* files the obvious choice for test vector communication with Mentor Fastscan.
While this aspect of the file type is very appealing, there are a couple of drawbacks. It is hard to modify the file type as is for the purpose of this project so we turn to the file types listed below to make this easier.

Easy-to-Read

The *Easy-to-Read* vector format is was created to represent a file of test vectors in a way that is more easily understandable and workable. This is likely the format in which most of your time viewing and manipulating test vectors will be spent.

Easy-to-Read Unserialized Vector File

Information
The top of the *Easy-to-Read* file created by the *p5convert.py* script contains information about the circuit, about the test vectors, and about the conversion process. At the very top of the file is a string which should not be removed. This is used in the *p5convert.py* script when no input file type has been designated. The script will look for this string and will determine that it should be read as an *Easy-to-Read* vector file.

```
EASY-TO-READ ** do not remove, file type detection
-- s27_comb.etr --
Initial input: p5example/final/STIL.FULLSCAN.exp1.logic.ex1.ts1.stil
Converted from stil to easy-to-read
10 vectors in file
0 clock signals present
7 input signals present (not including clocks)
4 output signals preset
```

Input/Output signal definitions
The input and output signals are then given in lists separated by commas. The ordering of these signals corresponds to the ordering of the bits in the test vectors printed below, so be careful when making any changes to these lists.

```
INPUTS: G0, G1, G2, G3, G5, G6, G7
OUTPUTS: G10, G11, G13, G17
```

Vectors

The rest of the file contains the test vectors. As stated above, the ordering of the bits is such to match the ordering of the signals listed above. The test vectors are represented in this fashion to make for a neat display and to increase the number of test vectors that can be viewed on the screen.

```
#       IN      OUT
1:      1000100 HLLH
2:      0011000 LHLL
3:      0000011 LHHL
4:      0100000 LLHH
5:      0000000 LLLH
6:      0111001 LLLH
7:      1000010 HLLH
8:      0000110 LLLH
9:      1001000 LHLL
10:     1001000 XXXX
```

Easy-to-Read Mapping File Template

When converting to the *Easy-to-Read* file format the ----map option exists, which creates a template for the *Easy-to-Read* mapping file used with *p5serialize.py*. This template file is meant to make the creation of the *Easy-to-Read* mapping file easier and more straightforward.

Information and Input/Output Signals

The top of this file is a little bit of information about the file. The inputs and outputs from the original test vector file are then printed.

```
 -- s27_comb.Tp5map --

Project 5 mapping file template
File used to define scan chains for use with 'p5serialize.py' script


** Input/output definitions do not need modification
INPUTS: G0, G1, G2, G3, G5, G6, G7
OUTPUTS: G10, G11, G13, G17
```

Signal Assignment

In this portion of the mapping file the user must decide how the serialization should treat each of the input and output signals printed above. For this project, the serialization will be performed on a set of test vectors which have been created for a purely combinational circuit.

This combinational circuit is a modified version of the original sequential circuit in which the inputs
and outputs of internal flip flops become primary outputs and inputs respectively.
In the DFT circuit that will be built in this project, many of the signals corresponding to the primary inputs and outputs of the combinational circuit are no longer input or output pins. Some may now be boundary scan cells, and others may be internal scan flip flops. In this section of the mapping file, the user will define how the signals are treated so that the serialization can remain consistent with their DFT circuit. For each signal group, every signal must be listed on the same line.
A completed example of this section can be seen here.

```
** EACH OF THE ABOVE SIGNALS MUST BE ASSIGNED TO ONE OF THE FOLLOWING GROUPS **

** Which of the inputs/outputs are to remain primary I/Os?
PRIMARY_INPUTS:
PRIMARY_OUTPUTS:

** Which of the inputs/outputs are being used in boundary scan cells?
INPUT_BS_SIGNALS:
OUTPUT_BS_SIGNALS:

** Which of the inputs/outputs correspond to internal scan flip flops?
** If 'A_in' (PO of comb circuit) is the serial input to the scan flip flop and
**    'A_out' (PI of comb circuit) is the serial output of the scan flip flop
** Use form:    'A_in+A_out, ...'
SCAN_PAIRS:
```

Controls and Clocks

The user's DFT circuit will contain clocks and controls which do not relate to any of the test vectors created for the combinational circuit. In the case of boundary scan, *TMS* and *TCK* will likely be required. In this section of the mapping file the user can list an clock or control signals that are present in their circuit. When the test vectors are serialized, values for the clock and control signals will be included at the beginning of each test vector. For clock signals, a 0 will be printed as the first line and will be pulsed for every subsequent vector.

Control signals can be given a value which will be printed in the serialized vector file. This can be either a singular value which will be printed for every test vector, or a sequence of bits which will be repeated throughout the test vectors. The boundary scan *TMS* pin or a scan enable pin are examples of situations when this sequencing may be useful.

A completed example of this section can be seen here.

```
** Define clock or control signals if desired.
** Clock signals will contain an initial 0 followed by pulses.
** Control signals can be assigned a cycle using the form:
**      'CNTLA=1, CNTLB=1110, CNTLC=ABC'
** If value is not assigned for control signal the value '%' will be given
CLOCKS:
CONTROLS:
```

Padding Vectors

Depending on the design of your circuit, you may wish to apply filler vectors to the circuit. In these filler vectors there are no bits applied to inputs or scan in pins, and there will only be expects on scan out pins if the vector is the last in the cycle. There will still be values applied to the control signals in these vectors.

Filler vectors will primarily be used when you wish to use the control signals to set up the circuit for scanning. For example, when working with boundary scan, the signal *TMS* controls the TAP controller. A specific sequence of bits will need to be applied to *TMS* at the end of each cycle to move the TAP controller through its states and update the circuit. In this case, adding the required number of filler vectors at the end of each cycle will allow the sequence you have defied for *TMS* to operate as desired.

```
** How many filler vectors should print before stimulating PIs or reading POs?
** Use digit, or leave blank if no added vectors are desired
ADD_VECTORS_BEFORE_IO:

** How many filler vectors should print at the end of each cycle?
** Use digit, or leave blank if no added vectors are desired
END_OF_CYCLE_VECTORS:
```

Scan Chains

This section allows the user to define the scan chains of their circuit. Each chain is given a name followed by an ordered list of signals corresponding to boundary scan cells or internal scan cells.

A completed example of this section can be seen here.

```
** User defines scan chains to produce corresponding serialized output
** Scan in port and scan out port will be created automatically
** Definition should be in following form
**  Use form:
**      SCAN_CHAIN <name>: <signal 1>, <signal 2>,  ... , <signal n>
**  For scan chain with order:
**      >|SI_<name>| >> signal 1 > signal 2 >  ... > signal n >> |SO_name|>
** Multiple scan chains can be defined
** For internal scan FFs give only the serial input signal (comb circuit output)
SCAN_CHAIN A:
```

Easy-to-Read Mapping File

This section of the report gives an example of what a completed version of the *Easy-to-Read* Mapping file might look like. Note that this example was used for very small circuit without boundary scan, so the file for your circuit should look very different.
The results of this completed mapping file can be found here.

Signal Assignment

In the signal assignment, you'll note that five of the signals which are primary inputs and outputs remain as such in serialization. There is no boundary scan implemented in this circuit, so none
of the original inputs or outputs correspond to boundary scan cells. Finally, there are three internal flip flops in the original sequential version of the circuit which are to become internal scan flip flops in the new version of the circuit. These scan pairs are defined as shown, with the input signal of the scan flip flop and the output signal connected with a "+".
The most important thing to note in this section is that every signal defined as a primary input or output is assigned to one of the three areas. Forgetting to perform this assignment for all of the signals will result in problematic results.

```
** Input/output definitions do not need modification
INPUTS: G0, G1, G2, G3, G5, G6, G7
OUTPUTS: G10, G11, G13, G17


** EACH OF THE ABOVE SIGNALS MUST BE ASSIGNED TO ONE OF THE FOLLOWING GROUPS **

** Which of the inputs/outputs are to remain primary I/Os?
PRIMARY_INPUTS: G0, G1, G2, G3
PRIMARY_OUTPUTS: G17

** Which of the inputs/outputs are being used in boundary scan cells?
INPUT_BS_SIGNALS:
OUTPUT_BS_SIGNALS:

** Which of the inputs/outputs correspond to internal scan flip flops?
** If 'A_in' (PO of comb circuit) is the serial input to the scan flip flop and
**    'A_out' (PI of comb circuit) is the serial output of the scan flip flop
** Use form:    'A_in+A_out, ...'
SCAN_PAIRS: G10+G5, G11+G6, G13+G7
```

Clocks/Controls

In this section the mapping file the clock and control signals are defined. In this example, the value “1” will be applied to the `reset` signal for every vector in the resulting file. The `scan_en` signal will cycle through the defined sequence “1, 1, 1, 0, 0, 0” in order throughout the test vectors. In this instance, the length of the sequence matches the length of the one serialized test vector, as can be examined in the serialized *Easy-to-Read* file. Because it is only possible
to print a repeated value or sequence of values, there may need to be some modifications made at the beginning of the test vectors in order to provide the correct setup for the circuit. For the design of most circuits, these additions should not be relatively easy.

```
** Define clock or control signals if desired.
** Clock signals will contain an initial 0 followed by pulses.
** Control signals can be assigned a cycle using the form:
**      'CNTLA=1, CNTLB=1110, CNTLC=ABC'
** If value is not assigned for control signal the value '%' will be given
CLOCKS: CK
CONTROLS: reset=1, scan_en=111000
```

Padding Vectors

In order to add padding vectors to the serialized *Easy-to-Read* file you must specify how many vectors should be added. In the following example, there will be no padding vectors printed before the inputs and outputs are tested. After the scan chain values have finished, the primary input and outputs values will be printed on the next line.
In this example, there will be two additional filler vectors printed after the primary input and output values have been applied. As mentioned earlier, this effect is generally used when control signals are acting upon the circuit. You can observe the result in the serialized *Easy-to- Read* file in the following section.

```
** How many filler vectors should print before stimulating PIs or reading POs?
** Use digit, or leave blank if no added vectors are desired
ADD_VECTORS_BEFORE_IO: 0

** How many filler vectors should print at the end of each cycle?
** Use digit, or leave blank if no added vectors are desired
END_OF_CYCLE_VECTORS: 2
```

Scan Chain Definitions

In this portion of the file the scan chains are defined. For this circuit there is only one scan chain given the name “chain1” and containing three signals. If you look to the signal assignment, you’ll find that these three signals all correspond to internal scan flip flops. Take note that only the input signal to the scan flip flop is defined in the scan chain.

```
** User defines scan chains to produce corresponding serialized output
** Scan in port and scan out port will be created automatically
** Definition should be in following form
**  Use form:
**      SCAN_CHAIN <name>: <signal 1>, <signal 2>,  ... , <signal n>
**  For scan chain with order:
**      >|SI_<name>| >> signal 1 > signal 2 >  ... > signal n >> |SO_name|>
** Multiple scan chains can be defined
** For internal scan FFs give only the serial input signal (comb circuit output)
** Reference help document 'Project 5 Scripts Help' for details on serialization
SCAN_CHAIN chain1: G10, G11, G13
```

Easy-to-Read Serialized File

Below is an example of what the final *Easy-to-Read* Serialized file might look like.
p5serialize.py
does most of the work required work with test vectors, but some small changes have been made to the test vectors to accomplish the desired function with the DFT circuit.

Information

At the top of the file is the “EASY-TO-READ” string printed for automatic file type detection in *p5convert.py* when no input file type is designated. The remainder of this section contains information about the circuit and test vectors.

```
EASY-TO-READ ** do not remove, file type detection
-- TEST.Setr --
Initial etr input: extra/s27_comb.etr
Mapped using: TEST.p5map

11 vectors in file
1 clock signals present
2 control signals present
7 total input signals present (with controls & scanins, not including clocks)
2 total output signals preset (including scanouts)
3 internal scan flip flops
0 boundary scan cells
1 scan chains present
```

Signal Definitions

This is where the signals in the circuit are defined and printed. Look at this portion in conjunction with the test vectors and take note of two important aspects of the file. Correct ordering of both the signal group definitions and the individual signal definitions is essential. You'll see that the clock signal group is the first to be defined here, and its test bits are the first to be listed in the test vectors. The control signals are defined next and their test bits are placed next in the test vector definition. The scan in signal group is then defined, and its test bits are placed next. Additionally, the ordering within the signal groups is also important to keep in mind. As with the unserialized *Easy-to-Read* vector file, the order of the signal definition within their groups corresponds to the bits in the test vectors.

Be careful when making changes to the ordering of these definitions. Any incorrect modifications will result in incorrect bit application to signals.

```
CLOCKS: CK
CONTROLS: reset, scan_en

SCAN_INS: scan_in
SCAN_OUTS: scan_out

PRIMARY_INPUTS: G0, G1, G2, G3
PRIMARY_OUTPUTS: G17
```

Test Vectors

In the serialized *Easy-to-Read* vector file the test vectors are split into signal groups. When adding a vector be sure to include a digit or "X" designation at the beginning of the line containing the test vector, as shown in this example. In *p5convert.py*, the digit (or "X") denotes that the line is a test vector, but the number of the test vector is not read in. The order of test vector application is performed solely by the order of vector definition within the file.

```
#       CK      CTL     SI      SO      PIN     POUT    FIN     FOUT
1:      0       ZZ      Z       X       ZZZZ    X
X:      P       00      Z       X       ZZZZ    X
2:      P       11      0       X       ZZZZ    X
3:      P       11      0       X       ZZZZ    X
4:      P       11      1       X       ZZZZ    X
5:      P       10      Z       X       1000    H
6:      P       10      Z       X       ZZZZ    X
7:      P       10      Z       L       ZZZZ    X
** END OF VECTOR 1 **
8:      P       11      0       L       ZZZZ    X
9:      P       11      0       H       ZZZZ    X
10:     P       11      0       X       ZZZZ    X
11:     P       10      Z       X       0011    L
12:     P       10      Z       X       ZZZZ    X
13:     P       10      Z       L       ZZZZ    X
** END OF VECTOR 2 **
14:     P       11      1       H       ZZZZ    X
15:     P       11      1       L       ZZZZ    X
16:     P       11      0       X       ZZZZ    X
17:     P       10      Z       X       0000    L
18:     P       10      Z       X       ZZZZ    X
19:     P       10      Z       H       ZZZZ    X
** END OF VECTOR 3 **
20:     P       11      0       H       ZZZZ    X
```