PSAC-Gen User's Manual
(Version 1.0)

Ketan Padalia (padalia@eecg.toronto.edu)

July 15, 1998

Table of Contents

PSAC-Gen User's Manual
(Version 1.0)

Ketan Padalia (padalia@eecg.toronto.edu)
July 15, 1998

1 - Overview

1.1 - Running PSAC-Gen

PSAC-Gen (Parameterized Serial Arithmetic Core Generator) is an FPGA core generation tool. PSAC-Gen can be used to easily generate a wide variety of arithmetic circuits involving addition, subtraction, and multiplication. PSAC-Gen takes as input an arithmetic circuit description and creates a set of VHDL files that describe the circuit. PSAC-Gen is invoked by typing:

psacgen [output_dir] [output_name] < [psac_script]

Output_dir is the name of the directory in which the VHDL files generated will be stored. Alternatively, if "-debug" is specified as the output directory, PSAC-Gen operates in **debug mode**. In debug mode, PSAC-Gen does not produce any VHDL output. It simple outputs the various information, warning, and error messages that it would output had it been producing the VHDL. This feature can be used to get information about a generated circuit before actually generating it.

Output_name is the name of the VHDL entity that represents the circuit generated by the "serial level" of PSAC-Gen (described below). Psac_script is the name of the file in which the desired circuit description is provided according to the syntax of "PSAC-Script" (see Section 4). The PSAC-Script file is redirected as the input to PSAC-Gen. Alternatively, you may type out a PSAC-Script program after running PSAC-Gen without the psac_script argument or its associated '<' symbol.

All the output messages that PSAC-Gen produces are sent to the standard output (usually the video terminal). Often, the messages are too long to fit on one screen. The easiest way to store these messages for full viewing is to redirect the output by adding a "> dump_filename" argument at the end of the psacgen invocation. This will send all output into a file named "dump_filename".

1.2 - PSAC-Gen Operation

There are two basic levels of operation when using PSAC-Gen (referred to in this manual as the "**serial level**" and the "**parallel level**"). The serial level allows you to generate an arithmetic circuit that takes inputs serially and generates outputs serially as well (i.e., the input numbers are sent to the circuit a certain number of bits at a time and the output numbers are generated that same number of bits at a time… this "number of bits at a time" is referred to in this manual as the "serial_data_width" (**SDW**)).

The parallel level of operation is optional. This level allows you to specify the overall throughput requirement for the circuit (referred to in this manual as the "parallel_throughput" (**PT**)). A PT requirement represents the rate "clock cycles per operation". For instance, if a circuit to average numbers is being designed and one average must be computed every 4 clock cycles, then the PT requirement is 4. If a PT requirement is specified, PSAC-Gen automatically replicates the modules described in the serial level to meet this requirement. PSAC-Gen also creates the control circuitry required to connect these modules. In addition to this, the parallel level generates a wrapper circuit that abstracts away the serial operation of the circuit by making the inputs and outputs parallel and serializing them internally.

The concept of these two levels of operation is illustrated below in ***Figure 1.1*** and ***Figure 1.2***.

Figure 1.1

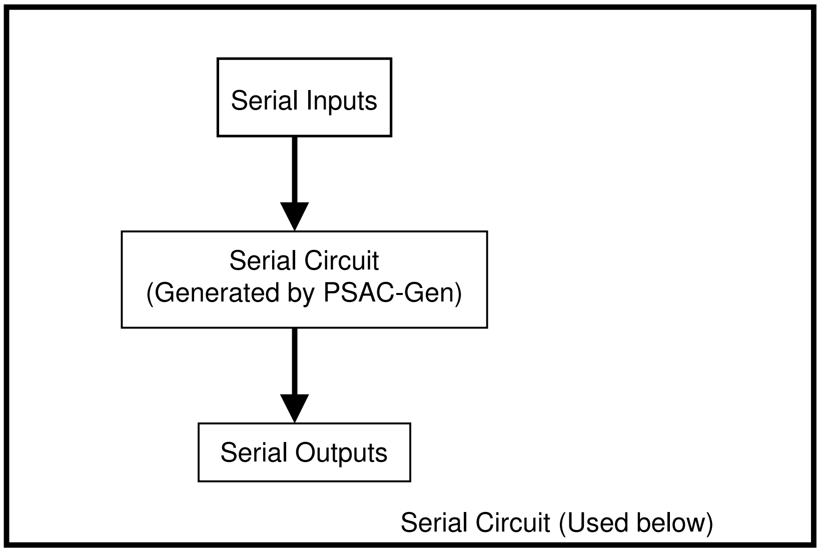
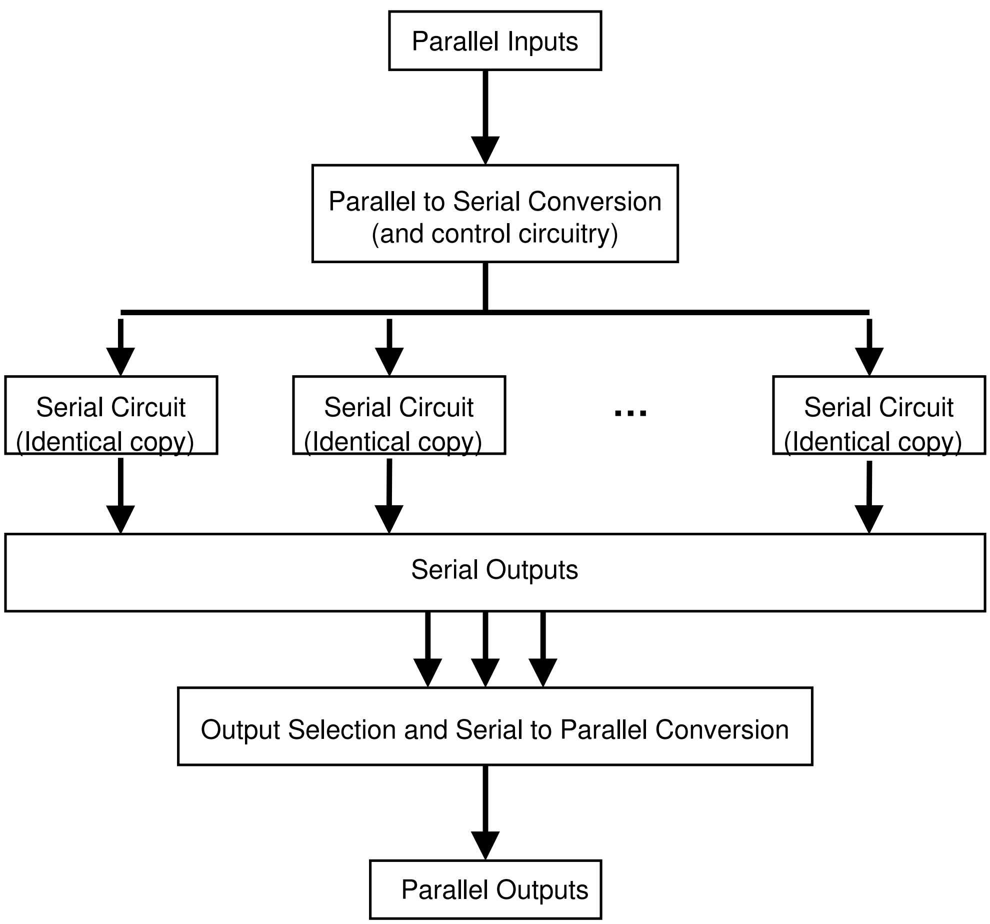


Figure 1.2



2 - PSAC-Gen Source

PSAC-Gen is comprised of five files: psacgen.y, psacgen.l, psacgen_hdr.h, Makefile and psacgen.

"psacgen.y" is the "Yacc" source code. Within this file, the rules for parsing PSAC-Script are specified. The majority of this file, however, is a series of C routines which interpret the PSAC-Script commands and generate VHDL files which describe the required circuit. This single file essentially contains all of PSAC-Gen's source code. "psacgen.l" is the "Lex" code which, as a companion to psacgen.y, helps parse PSAC-Script. "psacgen_hdr.h" contains constant definitions, function prototypes, and structure declarations that are used by other files. "Makefile" holds the directions for compiling PSAC-Gen. Finally, "psacgen" is the compiled and executable version of the tool.

In order to compile PSAC-Gen yourself, you must have the Yacc and Lex utilities. Compilation is done by either typing "make" in the directory where the files exist or by typing the following commands:

```
yacc -d psacgen.y
lex psacgen.l
gcc -o psacgen y.tab.c -ly -ll
```

3 - PSAC-Script

PSAC-Script is a simple, easy-to-use syntax for specifying arithmetic circuits that PSAC-Gen can generate. A PSAC-Script program consists of four major parts: parameter assignment, variable declaration, arithmetic description, and parallel level specification (optional). In addition, PSAC-Script provides commenting functionality. Comments can be inserted by using a '#' character. Everything that follows this character until the end of the line is considered to be part of the comment.

3.1 - Parameter Assignment

PSAC-Gen uses two parameters to specify the essential characteristics of an arithmetic circuit. These are the "operation_duration" (**OD**) and the SDW.

The OD represents the number of clock cycles that are devoted to a single operation. An "operation" is defined as the generation of outputs for one particular set of inputs. Thus, the duration is a measure of how many clock cycles you will wait after starting to send a set of inputs before another set of inputs will be sent. For example, a duration of six implies that if the first set of inputs begins to arrive at clock cycle 1, the next set of inputs will arrive at cycle 7, and then 13, 19, and so forth. If a duration is not specified, the minimum possible value will be chosen. If the duration specified is too small, it will be raised to the minimum value. Either of these generates a message which informs you of what actions PSAC-Gen is taking. For certain PT requirements, the OD must be divisible by a certain value. If this is not the case, PSAC-Gen will increase it to meet this requirement.

The SDW, discussed in Section 1, represents the number of bits of an input/output value that are sent/generated per clock cycle. These parameters are assigned with the following statements:

operation_duration = <value> and
serial_data_width = <value>

3.2 - Variable Declaration

There are three types of variables that need to be declared in PSAC-Script: input, output, and internal. For users familiar with VHDL, these correspond to input, output, and signal declarations in VHDL. All variables are declared in the form:

[variable type] [comma-separated list of names] : [data type] : [variable size]

Of these, the only partly optional parameter is the variable size. It is required for inputs. However, for internal and output variables the size is optional. If not provided, the size will automatically be calculated and PSAC-Gen will report the size it is using.

Variable type represents the three types indicated above (i.e., input, output and internal). Internal variables are needed because PSAC-Script only provides simple binary expressions as valid arithmetic descriptions (this is discussed further in Section 4.3). The comma-separated list of names allows you to declare more than one variable if those variables all have identical parameters. There are two types of data types available in PSAC-Script: unsigned and signed. The unsigned data type is meant for numbers which are known to be positive. The signed data type is meant for numbers which will be sent in as “two’s complement” signed numbers. The variable size represents the total number of bits that make up the variable. This value is not equivalent to the SDW.

3.3 - Arithmetic Description

This section is the heart of any PSAC-Script program. It provides a description of the circuit that is to be generated by using simple unary and binary expressions. Because only simple binary expressions are supported, all operations must be interfaced using internal variables. There are seven types of operations that can be used in the arithmetic description section: addition, subtraction, unsigned multiplication, unsigned one-input-constant (**OIC**) multiplication, signed multiplication, signed OIC multiplication, and the delay operation.

3.31 - Addition

The addition operation allows you to add two numbers together. This operation is used in PSAC-Script by typing:

[sink] = [source] + [source]

3.32 - Subtraction

The subtraction operation allows you to subtract two numbers. This operation is used in PSAC-Script by typing:

[sink] = [source] - [source]

3.33 - Multiplication

Multiplication allows two numbers to be multiplied together. This operation is used in PSAC-Script by typing:

[sink] = [source] * [source]

The two multiplication operations (signed and unsigned) are selected from automatically using the data types of the source variables. If one or both of the source variables is signed, the signed multiplier is used. It is important to note that signed multipliers are larger in area than their unsigned counterparts, and should be used only if signed multiplication is actually required.

3.34 - OIC Multiplication

OIC Multiplication allows a number to be multiplied by a constant. The presence of a constant input allows certain optimizations to be made to the multiplier, resulting in a smaller size. This operation is used in PSAC-Script by typing:

[sink] = [constant] * [source] or
[sink] = [source] * [constant]

Just as in regular multiplication, the type of multiplier (unsigned OIC or signed OIC) is decided automatically based on the data type of the single source variable.

3.35 - The Delay Operation

The delay operation allows you to specify a number of clock cycles by which a variable will be delayed. This operation is used in PSAC-Script by typing:

[sink] = [source]^N

N represents the number of clock cycles of delay that is required. This construct is particularly useful in order to latch the outputs of a circuit. Because the entire circuit is pipelined, PSAC-Gen latches only inputs of arithmetic units by convention. This is done to avoid extra pipeline stages when connecting different units together. If a latched output is desired, this must be done in the PSAC-Script program itself (Section 4 provides examples where this has been done).

You should note that PSAC-Gen does not require you to use delay operations in order to ensure information arrives at certain modules at the correct times. PSAC-Gen automatically determines where such delays are required and inserts them. However, if the circuit would benefit from a *particular* arrangement of delays, you must describe them yourself via the delay operation.

3.36 - The Left Shift Operation

The left shift operation allows you to specify a leftward shift of a variable. The resulting number is made bigger by the number of bits shifted. This operation is used in PSAC-Script by typing:

[sink] = [source] << N

N represents the number of bits that the variable is to be shifted.

3.37 - The Right Shift Operation

The right shift operation allows you to specify a rightward shift of a variable. The resulting number is made smaller by the number of bits shifted. This operation is used in PSAC-Script by typing:

[sink] = [source] >> N

N represents the number of bits that the variable is to be shifted. Note that if the number is to be shifted right without losing its size, a combination of right shift followed by left shift can be used.

IMPORTANT NOTE: The left and right shifting functionality of PSAC-Gen has not been tested extensively.

3.4 - Parallel Level Specification

This final section of a PSAC-Script program allows you to decide whether the parallel level of PSAC-Gen will be used or not. As discussed in Section 1, the parallel level is enabled by specifying a PT requirement. This is done by a single statement typed in the form:

parallel_throughput ([output filename], [PT requirement])

The output filename represents the name of the VHDL file that will be the highest possible file in the design hierarchy. This filename <u>must</u> be different from the filename specified as the second parameter to PSAC-Gen on the command line (i.e., the output_name).

The PT requirement is a number that represents the required throughput in terms of clock cycles per operation. For example, if one set of parallel inputs needs to be processed every two clock cycles, the PT requirement is 2. PSAC-Gen assumes that the inputs will be held steady for the number of cycles given as the PT requirement and the inputs can be loaded by PSAC-Gen on any of these clock cycles.

If the parallel level is enabled in this way, an additional set of statements must be provided to complete the process. In the parallel level, each output signal must have a "select" statement specifying which bits of the output signal are to be used in generating the final output. This is done by typing the following for each output variable:

select [variable name] ([MSB] downto [LSB])

This syntax is similar to VHDL's use of "downto" to select a certain range of bits. The variable name is self-explanatory. MSB represents the "most significant bit" or the upper bound of the bits wanted. LSB represents the "least significant bit" or the lower bound of the bits wanted. This range, as in VHDL, is inclusive. For example, "select final_output (7 downto 0)" means that the user requires only the lowest 8 bits of final_output to be considered as the output of the circuit.

4 - PSAC-Script Examples

4.1 - An 8-input Adder Tree

```
# 8-input Adder Tree - a fully serial circuit (i.e., one bit at a time)
#
# By: Ketan Padalia
# PSAC-Gen User's Manual (Version 1.0)

serial_data_width = 1            # Fully bit-serial

# duration is not set here and PSAC-Gen is allowed to automatically assign a value

input a,b,c,d,e,f,g,h : unsigned : 16    # The actual input values - 16 bit numbers

# Note that the sizes need not be given for the internal and output variables below

internal ab,cd,ef,gh : unsigned : 17
internal abcd,efgh : unsigned : 18
internal abcdefgh : unsigned : 19
output sum : unsigned : 19

ab = a + b        # These add the input values in pairs
cd = c + d
ef = e + f
gh = g + h

abcd = ab + cd            # These generate the next level of the adder tree
efgh = ef + gh

abcdefgh = abcd + efgh          # This is the final adder in the tree

# Outputs are never latched by PSAC-Gen.  If desired, you
#         may latch them as has been done below:

sum = 1^abcdefgh

# No parallel throughput requirement is specified.  Thus,
#         PSAC-Gen operates only at the serial level in this case.
```

4.2 - Averaging Four Numbers

```
# Four-input Averaging with Parallel Throughput Requirement
#
# By: Ketan Padalia
# PSAC-Gen User's Manual (Version 1.0)

serial_data_width = 2

input a,b,c,d : unsigned : 8

#Sizes for the internal and output variables are calculated automatically

internal ab,cd : unsigned
internal abcd : unsigned
output sum : unsigned

ab = a + b
cd = c + d

abcd = ab + cd

sum = 1^abcd

# PT Requirement: One average should be calculated every 2 clock cycles:

parallel_throughput (average, 2)

# The select statement below discards the two lowest bits
#	which is equivalent to dividing by 4 to get the final average

select sum (9 downto 2)
```

4.3 - 1x3 by 3x1 Matrix Multiply

```
# Matrix Multiplication
#
# By: Ketan Padalia
# PSAC-Gen User's Manual (Version 1.0)

serial_data_width = 4

input first_a, first_b, first_c : signed : 16
input second_a, second_b, second_c : signed : 16
internal aa, bb, cc : signed
internal aabb : signed
internal result : signed
output product : signed

aa = first_a * second_a
bb = first_b * second_b
cc = first_c * second_c

aabb = aa + bb

# Note that since aa,bb, and cc are all produced at the same time,
# cc cannot be added to aabb until aa and bb have been added.
# PSAC-Gen automatically handles the need for this scheduling

result = aabb + cc
product = 1^result

parallel_throughput (matrix, 6)
select product (15 downto 8)          # Low 8 bits discarded
```