Preliminaries in SV

Coding Structures: Modules

- The basic building block in SystemVerilog
  - Interfaces with outside using ports
  - Ports are input or output (for now)

```
module mymodule (a, b, c, f);
output f;
input a, b, c;
// Description goes here
endmodule
```

Concurrent Statements (CContext)

- Concurrent statements mean that the operations described in each line take place in parallel.
- The commonly used concurrent constructs are
  - gate instantiation and
  - the continuous assignment statement.

```
module ex1 (output wire y, input wire a, b, c);
    wire d, e, f;
        not g1 (d, c);
        and g2 (e, a, c);
        and g3 (f, d, b);
        or g4 (y, e, f);
endmodule
```

```
module mux2(a, b, sel, f);
    output f;
    input a, b, sel;
    logic c, d;
        assign c = a & (~sel);
        assign d = b & sel;
        assign f = c | d;
endmodule
```

Sequential Statements (SContext)

- Sequential statements are similar to statements in a normal programming language.
- A sequence of statements is regarded as describing operations that take place one after the other instead of at the same time.
  - All sequential SV statements must be inside a procedural block.
  - Sequential statements are placed inside a begin/end block and executed in sequential order within the block.
  - However, the blocks themselves are executed concurrently.
  - SV statements outside any process block are interpreted as concurrent statements and different procedural blocks execute concurrently.

Module Instantiation

```
module mymodule(a, b, c, f);
output f;
input a, b, c;
        module_name inst_name (port_connections);
endmodule
```

- You can instantiate your own modules or pre-defined gates
  - Always inside another module
- Predefined: and, nand, or, nor, xor, xnor
  - for these gates, port order is (output, input(s))

Connecting Ports (Order or Name)

- In module instantiation, can specify port connections by name or by order.
- Advice:
  - Use by-name connections (where possible)

```
module mod1 (input a, b, output f);
// ...
endmodule
// by order
module mod2 (input c, d, output g);
mod1 i0 (c, d, g);
endmodule
// by name
module mod3 (input c, d, output g);
mod1 i0 (.f(g), .b(d), .a(c));
endmodule
```

Modules and Files

- The entire module can be contained in single file.
- It is possible to have more than one module in a file,
  - but this is not advisable,
  - because any change to one module requires compilation of the entire file.

- It is recommended that you follow these guidelines when organizing your work:
  - Put each module in a separate file.
  - The file name and the module name should be the same;
    - give the file name the extension ".v" (for Verilog) or ".sv" (for SystemVerilog).
  - Do not use spaces in file names or folders/directories.
    - Some tools have difficulties, even when this is allowed by the operating system.

Identifiers, Spaces and Comments

```
module And2 (output wire z, input wire x, y);
assign z = x & y;
endmodule
```

- SystemVerilog is case-sensitive (like C).
- Keywords must be lower case.
- Identifiers (such as "And2") may be mixed-case.

Identifiers, Spaces and Comments

- It is strongly recommended that usual software engineering rules about identifiers should be applied.
  - Meaningful, non-cryptic names should be used, based on English words.
  - Use mixed-case with consistent use of case.
  - Don't use excessively long identifiers (15 characters or fewer).
  - Don't use identifiers that may be confused (e.g. two identifiers that differ by an underscore)
  - Identifiers may consist of letters, numbers and underscores ("_"), but the first character must not be a number.
  - System tasks and functions start with a dollar symbol ("$").
  - Escaped identifiers.
    - It is possible to include other symbols in identifiers by starting the identifier with a backslash ("\").
    - This is intended for transferring data between tools, so use with extreme caution !

Identifiers, Spaces and Comments

- White space (spaces, carriage returns) makes models more readable.
  - no difference between one white space character and many.
- Comments may be included in SV by putting two slashes on a line ("//").
  - All text between the slashes and the end of the line is ignored.
  - Similar to the C++ style of comment.

- There is also a C-style block comment ("/*...*/") in SV.
- It is strongly recommended that
  - comments should be included to aid in the understanding of SV code.

Files with Comments

- Each SystemVerilog file should include a header, which typically contains:
  - the name(s) of the design units in the file;
  - file name;
  - a description of the code;
  - limitations and known errors;
  - any operating system and tool dependencies;
  - the author(s), including a full address;
  - a revision number.

- For example:

```
////////////////////////////////////////////////////
// Design unit : And2
// File name : And2.v
// Description : Model of basic 2 input AND
// : gate. Inputs of type wire.
// Limitations : None
// System : IEEE 1800-2005
// Author :
// Revision : Version 1.0 04/02/09
////////////////////////////////////////////////////
```

SystemVerilog Operators

Higher

Arithmetic	Bitwise	Logical
+ – (unary) * / % + –	~ << >> & ^ ^~ |	! && ||

Lower

- The arithmetic and logical operators are given in decreasing order of precedence in Table.
- The standard arithmetic operators:
  - % a modulus operator
  - ~ a bitwise negation
  - ! a logic negation
  - << shift left
  - >> shift right
  - & AND
  - | OR
  - ^ XOR
- The bitwise operators can be used as unary reduction operators, taking all the bits in a vector as inputs and giving a single bit output.

Precedence

	Op	Meaning
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, -	PLUS, MINUS
	<<, >>	Logical Left/Right Shift
	<<<, >>>	Arithmetic Left/Right Shift
	<, <=, >, >=	Relative Comparison
	==, !=	Equality Comparison
	&, ~&	AND, NAND
Lowest	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
	?:	Conditional

- AND has precedence over OR.
- We could take advantage of this precedence to eliminate the parentheses.
- assign cout = g | p & cin;

Constants

Numbers	Bits	Base	Val	Stored
3'b101	3	2	5	101
'b11	?	2	3	000 ... 0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00 ... 0101010

- The format for constants is N'Bvalue, where
  - N is the size in bits,
  - B is a letter indicating the base, and
  - value gives the value.
- For example, 9'h25 indicates a 9-bit number with a value of $25_{16} = 37_{10} = 000100101_2$.

Numbers

Numbers	Bits	Base	Val	Stored
3'b101	3	2	5	101
'b11	?	2	3	000 ... 0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00 ... 0101010

- Numbers can be specified in binary, octal, decimal, or hexadecimal (bases 2, 8, 10, and 16, respectively).
- The size, i.e., the number of bits, may optionally be given, and leading zeros are inserted to reach this size.
- Underscores in numbers are ignored and can be helpful in breaking long numbers into more readable chunks.
- SV supports 'b for binary, 'o for octal, 'd for decimal, and 'h for hexadecimal.
- If the base is omitted, it defaults to decimal.

Numbers

Numbers	Bits	Base	Val	Stored
3'b101	3	2	5	101
'b11	?	2	3	000 ... 0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00 ... 0101010

- If the size is not given, the number is assumed to have as many bits as the expression in which it is being used.
- Zeros are automatically padded on the front of the number to bring it up to full size.
- For example, if w is a 6-bit bus,
  - assign w = 'b11 gives w the value 000011.
- It is better practice to explicitly give the size.

RTL Synthesis

- The IEEE standard 1364.1-2002
  - Defines a subset of Verilog for RTL synthesis.
- Define the minimum subset that can be accepted by any synthesis tool.
- SystemVerilog is based on the 2001 enhancements to Verilog.

SV Enhancements

- SV adds a number of significant programming capabilities over Verilog.

- The intent of these enhancements is three-fold:
  - model more functionality in fewer lines of code,
  - reduce the risk of functional errors in a design,
  - help ensure that simulation and synthesis interpret design functionality in the same way.

Procedural Blocks

- In Verilog, always blocks are used to model combinational, latch, and sequential logic.
- Synthesis and simulations tools have no way to know what type of logic an engineer intended to represent.
- Instead, these tools can only interpret the code within the procedural block, and then "guess" the engineer's intention.
- Simple coding errors in combinational logic can generate latches with no indication that latch behavior exists until the synthesis result reports are examined.
- Many re-spins have occurred due to missing unexpected latches in lengthy synthesis reports.

SV Solutions

- SV adds three new types of procedural **always** blocks that document intent and provide some synthesis rule checking.
  - always_comb, always_latch, always_ff.

- Simulators, lint checkers, and synthesis compilers can issue
  - warnings if the code modeled within these new procedural blocks does not match the designated type.
- These warnings are optional, and
  - are not required by the SV standard.

Non-synthesizable SystemVerilog

- In principle, most features of SV could be translated into hardware.

- Not synthesizable parts of SV:
  - constructs in which exact timing is specified and
  - structures whose size is not completely defined.
- Poorly written SV may result in the synthesis of unexpected hardware structures.
- All delay clauses (e.g. #10).
  - Delays are simulation models.

Non-synthesizable SV Constructs

- File operations suggest the existence of an operating system.
  - File operations cannot be synthesized and would be rejected by a synthesis tool.
- Real data types are not inherently unsynthesizable,
  - but will be rejected by synthesis tools
- Initial blocks will be ignored.
  - Hardware cannot exist for a limited period of time and then disappear!

Three Additional Specifications in Verilog

- Compiler directives:
  - A compiler directive may be used to control the compilation of a Verilog description.
- Pragmas:
  - A pragma is a generic term used to define a construct with no predefined language semantics that influences how a synthesis tool should synthesize Verilog HDL code into a circuit. Not part of language.
- Attributes: (Effort to create language constructs for Pragmas)
  - An attribute specifies special properties of a Verilog statement, for use by specific software tools, such as synthesis.
  - Attributes were added in Verilog-2001.
  - EDA did not implement all of them.

Two Categories of Objects

- Two classes of objects:
  - Variables
  - Nets

- They both have a type and a data type.

Types of Objects

- A type
  - is **a key word** to indicate the nature of the object which is either a net or a variable.
- The type for a variable:
  - **the keyword var** is explicitly specified or implicitly inferred.
- The types for a net:
  - **the keywords wire**, **tri**, etc.

Data Types of Objects

- A data type defines a value system which is either a 2-state and 4 state.
- Data types are used by software tools (simulators, synthesis compilers) to determine how to store data and process changes on the data.

- Misleading points:
  - It is not clear at all about differences between types and data types.
  - Sometimes, data types are used to include types.

SV Primitives

- 0: Clear digital 0
- 1: Clear digital 1
- X: Means either "don't know" or "don't care"
  - Useful for debugging
  - Also useful for 'don't care' bits in logic
- Z: High impedance, non-driven circuit
  - Value is not clearly 0 or 1
  - Useful for testing, debugging, and tri-state logic

Data Type: **logic**

- The **logic** keyword infers a data type.
  - logic = {0, 1, X, Z}, 4-state data type
  - It defines a value set.

- *Data type logic* **is not, in itself,** a net or a variable object.

Variables

- Variables are used as temporary storage for programming.
- This temporary storage is for simulation.

- Actual silicon **often** does not need the same temporary storage, depending on the programming context.

Type of Variables

- The keyword **var** is explicitly specified or implicitly inferred.

- The keyword **var** is seldom used in actual SV code.
- Instead, the type **var** is inferred from other keywords and context.

Data Types of Variables

- Synthesizable variable data types:
  - reg, logic, integer, bit, int, byte, shortint, longbit.

- Nonsynthesizable variable data types:
  - Real, shortreal, time, realtime, string, event, etc.

Variable Implicit Declarations by **logic**

- Variables:
  - When **logic** is used by itself,
    - a variable is inferred, with its single-source assignment restriction.
  - When the keyword pair **output logic** is used to declare a module port,
    - a variable is also inferred.

```
module fulladder(input logic a, b, cin,
                 output logic s, cout);
logic p, g;
assign p = a ^ b;
assign g = a & b;
assign s = p ^ cin;
assign cout = g | (p & cin);
endmodule
```

Uninitialized Variables

- A variable is uninitialized until a value has been assigned to the variable.

- The uninitialized value of 4-state variables (logic) is
  - 'x (all bits set to X).
- The uninitialized value of 2-state variables (bit) is
  - '0 (all bits set to 0).

Nets

- Nets are used to connect design blocks together.
- A net transfers data values
  - from a source (called a **driver**)
  - to a destination or receiver.

Type of Nets

- The net types are the following keywords:
  - **wire**, **tri**, etc.

Data Types of Nets

- The data type of Nets must be **logic**
  - Can be explicitly specified or implicitly inferred.
- Synthesizable variable data types:
  - reg, logic, integer, bit, int, byte, shortint, longbit.
- Nonsynthesizable variable data types:
  - Real, shortreal, time, realtime, string, event, etc.

Net Implicit Declarations by **logic**

- Nets:
  - When the keyword pair input logic or inout logic is used to declare a module port,
    - a wire is inferred, with its multiple driver capability.

```
module fulladder(input logic a, b, cin,
                 output logic s, cout);
logic p, g;
assign p = a ^ b;
assign g = a & b;
assign s = p ^ cin;
assign cout = g | (p & cin);
endmodule
```

Assign Values to Nets

- Nets can **only** receive a value from two types of sources:
  - As a connection to an output or inout port
  - As the left-hand side of a continuous assignment.

```
assign net = ……..
```

Nets & Variables

Data Type **logic**

Objects

- Nets: Signals with multiple drivers
  - wire, tri, input logic, inout logic
  - Used for connecting different modules
  - Treated as a physical wire
  - They can be read or assigned
  - No values get stored in them: continuous nature.
  - They need to be driven by either continuous assign statement or from a port of a module

Objects

- Variables: single driver
  - logic, out logic
  - They can be read or assigned
  - Values get stored in them

Variables: single-driver logic
Nets : multiple-driver logic

- SystemVerilog's restriction
  - variables cannot receive values from multiple sources.
  - This help prevent design errors.

- Wherever a signal should only have a single source, a variable can be used.
- The single source can be procedural block assignments, a single continuous assignment, or a single output/inout port of a module or primitive.
- Should a second source inadvertently be connected to the same signal, it will be detected as an error, because each variable can only have a single source.

Variable with Multiple Drivers

```
1.  module add_and_increment (output logic [63:0] sum, output logic carry,
2.  input logic [63:0] a, b );
3.  always @(a, b)
4.  sum = a + b; // procedural assignment to sum
5.  assign sum = sum + 1; // ERROR! sum is already being assigned a value
6.  look_ahead i1 (carry, a, b); // module instance drives carry
7.  overflow_check i2 (carry, a, b); // ERROR! 2nd driver of carry
8.  endmodule
9.  module look_ahead (output wire carry, input logic [63:0] a, b);
10. ...
11. endmodule
12. module overflow_check (output wire carry, input logic [63:0] a, b);
13. ...
14. endmodule
```

However, for the old **always**……

- Continuous assignments do not allow to have two drivers, while procedure **always** blocks allow.
- The Verilog does permit a variable to be written to by multiple **always** procedural blocks

However, for the old **always**……

- SystemVerilog does permit a variable to be written to by multiple **always** procedural blocks, which can be considered a form of multiple sources.
  - Some synthesizer like Xilinx for FPGA will complain for multiple drivers.
- This condition must be allowed for backward compatibility with the Verilog language.

- SV introduces three new types of procedural blocks:
  - always_comb, always_latch and always_ff.
- These new procedural blocks have the restriction that a variable can only be assigned from one procedural block.
- This further enforces the checking that a signal declared as a variable only has a single source.

References

- Digital System Design with SystemVerilog by Mark Zwolinski. Prentice Hall, 2009.
- Logic Design and Verification Using SystemVerilog (Revised) by Donald Thomas, 2016.
- Language Reference Manual (LRM) – IEEE 1800-2005