10.4 Variable Synchronizing

When variables are used,

Synchronizing within a Condition or Vector Statement - Consider the following case involving two integer variables, FOO and BAR:

\[
\begin{align*}
C &\{ \text{FOO = 1}; \} \\
C &\{ \text{FOO = 6; BAR = 'FOO+1';} \} \quad // \text{BAR result is 2}
\end{align*}
\]

After the second C statement what is the value of BAR? The answer is 2. All expressions that are placed in a Pattern statement are considered to be executed in zero time and all results are based on the value of variables at the beginning of the statement. There is no circular or order dependency. If it is desired to force an order dependency then multiple statements are used:

\[
\begin{align*}
C &\{ \text{FOO = 1}; \} \\
C &\{ \text{FOO = 6;} \} C &\{ \text{BAR = 'FOO+1';} \} \quad // \text{BAR result is 7}
\end{align*}
\]

Synchronizing across parallel patterns - Now consider the case of two patterns P1 and P2 that are executing in lock step and share the common variable FOO:

\[
\begin{align*}
\text{Variables} &\{ \text{Integer FOO } \{ \text{InitialV alue 0;} \} \} \quad // \text{globally available} \\
\text{PatternBurst B} &\{ \\
&\quad \text{ParallelPatList LockStep } \{ \text{P1; P2;} \} \\
\text{Pattern P1} &\{ \\
&\quad \text{V }\{ \text{SIGS_P1 = 000111;} \} \\
&\quad \text{C }\{ \text{FOO = 16;} \} \\
&\quad \text{Loop FOO }\{ \text{CLK1 = P;} \} \quad // \text{loop 16 times}
\text{Pattern P2} &\{ \\
&\quad \text{V }\{ \text{SIGS_P2 = 111000;} \} \\
&\quad \text{Loop FOO }\{ \text{CLK2 = P;} \} \quad // \text{loop 16 times}
\end{align*}
\]

How many times does the loop in P2 execute and why? The answer is 16. The variable FOO is in scope across both patterns because it is global in scope in this case. The C/Condition statements are considered to execute in zero time, so the sequence of execution is: 1. all consecutive C statements are executed in each pattern independantly, 2. any global variables are exported to other patterns, 3. the next non-C statement uses the values established from other patterns for all global variables.

15.5 ScanStructures Syntax

\[
\begin{align*}
\text{ScanStructures} &\{ \text{SCAN _ STRUCT _ NAME } \} \quad (1) \\
&\quad \text{(ScanChain} \quad \text{CHAIN _ NAME} \quad \{ \\
&\quad \quad \text{ScanLength} \quad \text{integer_expr} \quad ; \quad (2) \\
&\quad \quad \text{ScanOutLength} \quad \text{integer_expr} \quad ; \quad (3) \\
&\quad \quad \text{ScanEnable} \quad \text{logic_expr} \quad ; \quad (4) \\
&\quad \quad \text{ScanCells} \quad \{ \quad (5) \\
&\quad \quad \quad \text{scan_cell_name} \quad \text{(CELL _ TYPE _ NAME )} \quad ; \quad )* \quad (6) \\
&\quad \quad \quad \text{((If } \quad \text{boolean_expr} \quad \text{CellIn} \quad \text{STATE - ELEMENT - LIST } ; \quad ) \quad )* \\
&\quad \quad \quad \text{((If } \quad \text{boolean_expr} \quad \text{CellOut} \quad \text{STATE - ELEMENT } ; \quad ) \quad )* \\
&\quad \quad \quad \} \quad >)* \quad // \text{end scan_cell_name} \\
&\quad \quad \} \quad // \text{end ScanCells}
\end{align*}
\]
ScanCells: This statement shall appear at most once in a ScanChain block. It shall contain either: a) a space separated list of scan cell names as defined in IEEE Std. 1450-1999, or b) a brace delimited block containing a semi-colon separated list of scan cell names. The new block form of this statement allows for additional attributes to be attached to the scan cells to define complex scan cells. See “ScanStructures using complex scan cells” on page 92 for examples of this construct.

CELL_TYPE_NAME - In the new brace delimited form, an optional cell type name may be defined. Once this name has been established, all subsequent cells of the same type may then reference this name and be semi-colon delimited. Cell type names shall be defined (i.e., brace statement) before used (i.e., semicolon statement). Cell type names are in-scope for the ScanStructure block in which they are defined.