13. PatternBurst Block

Extensions to STIL.0 Clause 17

This clause defines additional statements supported within the PatternBurst block. All statements and capabilities as defined in STIL.0 Clause 17 are unchanged.

Two new pattern grouping structures are defined - the ParallelPatList and the PatSet. Also, the Fixed, Extend, and Wait statements are defined to allow the specification of how multiple patterns are to be executed.

Lastly, the If and While statements are provided to allow conditional execution of patterns within a burst.

13.1 PatternBurst Syntax

```
PatternBurst PAT_BURST_NAME {
    ( Variables VARIABLES_DOMAIN; )* (1)
    ( Fixed { (cyclized-data)* } ) (2)
    ( PatList {
        ( PAT_NAME_OR_BURST_NAME {
            ( Variables VARIABLES_DOMAIN; )* (3)
            ( Fixed { (cyclized-data)* } )
            ( If boolean_expr ; ) (5)
            ( While boolean_expr ; ) (6)
        } )* // end pat_name_or_burst_name
    } )* // end PatList
} )* // end PatSet

( ParallelPatList ( SyncStart | Independent | LockStep ) {
    ( PAT_NAME_OR_BURST_NAME ; )* (8)
    ( PAT_NAME_OR_BURST_NAME {
        ( Variables VARIABLES_DOMAIN; )* (7)
        ( Fixed { (cyclized-data)* } )
        ( Wait; ) (9)
        ( If boolean_expr ; )
        ( While boolean_expr ; )
    } )* // end pat_name_or_burst_name
} )* // end ParallelPatList
```

(1) **PatternBurst**: Refer to STIL.0 for the definition of the PatternBurst block and statements not defined in this extension.

(2) **Variables**: This statement allows reference to a named block of variables to be allowed by all patterns and pattern bursts within this block (see clause 10 for the definition of variables).

(3) **Fixed**: This statement allows the specification of signals for which the operation is to be defined outside of the Pattern or PatternBurst. The signals may define a fixed static state, or may assigned a WFC which is to be repeated. This statement may be associated with the entire pattern burst block or it may be associated with a pattern or burst within a PatList, PatSet, or ParallelPatList. It requires that the list of signals in the sigref_expr not be used in the associated patterns. If a signal is fixed to a WFC, then that WFC shall be defined within each WFT that is selected within each pattern. Any wfc on a "Fixed" signal other than that specified in the Fixed statement is an error. If a static
fixed value is desired, then this can be accomplished using the \e syntax (see subclause18.3) which requires no wfc/wft definitions.

This statement performs a similar function to the Fixed statement within a pattern. The difference is in the scope. The Fixed statement within a pattern is in effect from its occurrence to the end of the pattern, whereas the Fixed statement within a PatternBurst is in effect for the entire pattern or burst of patterns. Also, use of this statement does not require any change to the pattern.

The assertion within a pattern of a signal that is in a "fixed" state may occur, as long as the assertion is to the same state or wfc to which it has been "fixed".

(4) **PatList**: The PatList block performs exactly the same function as defined in STIL.0. It is repeated here to show the new optional statements that are allowed within a pattern list, namely the If and While attributes.

(5) **If**: This attribute defines a conditional requirement on the execution of the PAT_OR_BURST_NAME; this block will execute only if the boolean_expr is True. The expression is evaluated at the point this reference would start to execute in the sequence defined.

(6) **While**: This attribute defines an iterative option for execution of the PAT_OR_BURST_NAME; this reference will be executed repeated times as long as the boolean_expr is True. The expression is evaluated each time this reference would start to execute; pattern sequencing iterates on this reference until the expression returns False.

(7) **PatSet**: This block defines a set of PAT_OR_BURST_NAME that have no requirement on the order of execution of each reference. This construct is intended primarily for defining data in an interim format prior to being presented to a tester, to identify a set of Patterns that have no external constraints on order of execution. If this data is presented to a tester, then each Pattern shall be self initializing and capable of executing independently.

The PatSet block is similar to the PatList in that it is used to define a list of Patterns and the environment for interpreting them. The difference is that the PatSet does not imply any sequencing requirements. Thus, the system integrator is free to take that Patterns and use them in any sequence desired. All of the optional statements that are defined in STIL.0 for PatList also are available in a PatSet block.

The requirement on the Patterns in a PatSet is that they be self contained such that any initialization that is required is done within the pattern and not dependent on the execution of a prior pattern. Also, because there is no required ordering of Patterns in this block, If and While statements are not supported for these blocks because of ambiguity of expression evaluation.

The PatSet is not used to do implicit parallel patterns, even if the set of signals between patterns are non-overlapping. To do this, you need to define two bursts that contain PatSets, and reference them in a higher level with a Parallel-PatList.

(8) **ParallelPatList** (SyncStart | Independent | LockStep) : This block defines a set of PAT_OR_BURST_NAME that are to be executed in parallel. Execution of this set of Patterns is controlled by optional arguments SyncStart, Independent, or LockStep, as well as the optional statements Wait and Extend. Parallel patterns do not necessarily run synchronously or finish together. If no arguments are specified to ParallelPatList then the default operation of the Patterns is Independent. All of the optional statements that are defined in STIL.0 for PatList also are available in a ParallelPatList block.

**SyncStart**: this keyword, if present, requires that all PAT_OR_BURST_NAME present in the ParallelPatList block shall start executing at the same moment. During execution, pattern behavior may diverge if patterns contain different Vector counts or different periods in the Vectors.

**Independent**: this keyword, if present, allows each PAT_OR_BURST_NAME present in the ParallelPatList block to start as convenient. This option indicates that the set of patterns executing in parallel have little or no relationship between each other and can be executed independently.

**LockStep**: This keyword is used to specify parallel testing of sub designs that have independant patterns requiring synchronization throughout the pattern execution. Example of application where this is used are:
a) Situations where parallel sub-designs have common access constructs that require maintaining the same state on a set of signals for the cores during test (for instance, common wrapper control logic around the sub-designs).

b) Parallel testing of sub-designs that have serially connected scan chains.

c) To map patterns onto test equipment that has limited timing flexibility that prevents true independent execution.

The details of how the patterns, procedures, and macros are resolved is a function of the tools that create and consume the data. The following are the general requirements that shall be observed for LockStep:

a) All pattern shall start at the same time.

b) Each vector in each pattern shall have the same period.

c) Each macro and procedure invocation shall occur at the same time and shall be of the same vector length.

d) Signals that occur in multiple patterns shall either be set to the same wfc, or else shall be resolved in only one of the patterns.

(9) **Wait**: This statement signifies that all other patterns that are running in parallel are to wait for this pattern to terminate. **Wait** is incompatible with **LockStep**; it is an error to specify **Wait** in a PAT_OR_BURST_NAME block under a ParallelPatList block that defines **LockStep**.

(10) **Extend**: This statement signifies that the last vector of this pattern may be extended in order to wait for completion of other parallel patterns. The behavior of Signals during Extend is described in subclause 13.3. **Extend** is incompatible with **LockStep**; it is an error to specify **Extend** in a PAT_OR_BURST_NAME block under a ParallelPatList block that defines **LockStep**.

If the Extend statement is not used, then the last cycle shall "tile" exactly with any other patterns that are running in parallel.

The behavior of the Signals during the Extend period is determined by the last STIL statement in the Pattern. If the last statement in the Pattern is a "BreakPoint;" then all Signals will maintain the last asserted state indefinitely until the parallel set of Patterns is complete. If the last statement is "BreakPoint { V { ...} }", then the Vectors present in this BreakPoint block shall be executed for these Signals until all parallel patterns have completed.

If the last STIL statement in the Pattern is not a BreakPoint construct, then the last asserted state on all Signals shall be maintained indefinitely as if a "BreakPoint;" statement was present in the Pattern.

### 13.2 PatternBurst example

```
446: STIL 1.0 { Design D18; }
447: Header { 
448:   Source "IEEE P1450.1/D19, March 3 2004";
449:   Ann {* subclause 13.2 *}
450: }
451: Variables { 
452:   Integer FIRST_PAT_PASSED { InitialValue := 0; }
453:   Integer NOT_COMPLETE { InitialValue := 0; }
454: }
455: 
456: PatternBurst BURST_XX { 
457:   PatList { 
458:     FIRST_PAT;
459:     SECOND_PAT { 
460:       If (FIRST_PAT_PASSED);
461:     } 
462:     THIRD_PAT { 
```
13.3 Tiling and Synchronization of Patterns

Pattern tiling is the process of connecting patterns together end-to-end in time and side-to-side by signal. The following is a summary of the facilities provided for the purpose of tiling and the rules associated with tiling:

a) PatList and PatSet allows specification of patterns that run sequentially (i.e., one after the other). Each pattern shall run to completion before the next pattern can begin.

b) ParallelPatList allows specification of patterns that start running at the same time. All patterns in the list start execution at the same time. All patterns shall run to completion before the next pattern can begin with the length of time being determined by the longest pattern or burst in the list.

c) If a pattern ends with a Breakpoint statement, then that pattern can be extended in time as needed when run in a ParallelPatList. If a pattern ends with "BreakPoint;" then the signals can be extended by just holding them in the final state as long as required. If a pattern ends with "Breakpoint { ... }" (i.e., a block of statements) then that block can be repeated as necessary to extend the pattern.

d) The "Extend;" statement can be placed on any pattern in a ParallelPatList, effectively specifying that the last vector can be extended. This option allow for extending a pattern without having to modify the pattern file itself.

e) With regard to synchronizing patterns after a ParallelPatList execution, it is necessary that any pattern that terminates early in a ParallelPatList has one of the above extend mechanisms specified else it is an error. Also, note that all patterns must terminate or extend to the same point in time for the next pattern to execute. If the cyclized extend is being used, care must be taken that the period multiples of each pattern in parallel end up with the same composite time (usually accomplished best when periods of parallel patterns are the same).

f) There are two facilities for defining signals that are "fixed" (i.e., that don’t change for the duration of a pattern or a burst of patterns. The "Fixed" statement may be placed at the top of the PatternBurst, in which case it is fixed for all patterns within the burst. The "Fixed" statement can be placed within a Pattern, in which case it is fixed only for the duration of the pattern. If a set of signals is to be "fixed" and are not part of any pattern, then a pattern file containing only the "Fixed" statement will suffice (see example below).

g) If patterns running in parallel contain common signals, then the activity on these signal shall be coordinated in all patterns that use them. The simple method for coordinating signals (see next point for the other option) is to run the patterns in LockStep and insure that the same wft/vfc occurs in every cycle.

h) If patterns running in parallel contain common signals where the activity differs, then the AllowInterleave statement shall be used. This mechanism requires that only one of the parallel patterns defines that activity in any given cycle, and every cycle must have one pattern that defines the activity (i.e., there is no automatic repeat into the next vector).

The following is an example of Pattern Tiling using the statements available in the PatternBurst. Refer to clause 13
for definition of syntax and semantics.

![Figure 1—A Collection of Patterns to be Executed](image)

One example of STIL code which would specify the above action is as follows:

```stil
467: STIL 1.0 { Design D18; }
468: Header {
469:  Source "IEEE P1450.1/D19, March 3 2004";
470:  Ann {* subclause 13.3 *}
471: }
472: Signals { SIG1 In; SIG2 In; SIG3 Out; }
473: PatternBurst BURST_X {
474:  ParallelPatList {
475:   PAT_A {Extend;}
476:   PAT_B {Extend;}
477:   PAT_C {Extend;}
478:   PAT_FIX;
479:  }
480:  PatList {
481:   PAT_D {Extend;}
482:   PAT_E {Extend;}
483:   PAT_F;
484:  }
485: }
486: Pattern PAT_FIX {
487:  Fixed {'SIG1+SIG2'=\eUU; SIG3=\eh;}
488:  Breakpoint;
489: }
```

This code causes the following actions:

1. PAT_A, PAT_B and PAT_C are initiated at the same time. PAT_FIX defines a set of fixed signals that also begins at the same time.

2. PAT_A and PAT_B are allowed to extend, since they are expected to complete in less time than PAT_C. Extending means that the state of all signals at the end of the last vector of the pattern can be held, effec-
tively extending the period of the last vector.

3. PAT_C must run to completion before any further activity in this burst can be started.

4. PAT_C is allowed to extend its last cycle.

5. PAT_D, PAT_E, and PAT_F are run sequentially, starting immediately upon completion of the last vector of PAT_C.