

## STIL Reference Guide

STIL.0 - Basic STIL; approved by IEEE in 1999  
STIL.1 - Design extension; draft D19, June 17, 2003  
STIL.2 - DC Levels; approved by IEEE in 2002  
STIL.3 - TRC extension (Tester Resource Constraints); draft D8, Apr 11, 2003  
STIL.4 - Flow extension; not included  
STIL.5 - Test Method extension; not included  
STIL.6 - CTL extension (Core Test Language), draft D1.2, Aug 2004

## Lexical Environment

*Simple statement construct:*

**Keyword** (OPTIONAL\_TOKENS)\*;

*Block statement construct:*

**Keyword** (OPTIONAL\_TOKENS)\* { (OPTIONAL\_MORE\_STATEMENTS)\* }

*Whitespace:*

space  
\t tab  
\n newline character

*Comments:*

// LINE COMMENT- *line comments are terminated by newline*  
/\* BLOCK  
COMMENT \*/- *block comments may span multiple lines*

## IEEE Std. 1450-1999

STIL 1.0;

**Header** {

( **Title** "TITLE\_STRING"; )  
( **Date** "DATE\_STRING"; )  
( **Source** "SOURCE\_STRING"; )  
( **History** { } )  
}

**Include** "file\_name" (**IfNeed** <blocktype>);

**Ann** { \* ANNOTATION \* }

**UserKeywords** (USER\_KEYWORDS)+;

**UserFunctions** (USER\_KEYWORDS)+;

**Signals** {

( SIG\_NAME < **In** | **Out** | **InOut** | **Supply** | **Pseudo** > ; ) \*  
( SIG\_NAME < **In** | **Out** | **InOut** | **Supply** | **Pseudo** > {  
( **Termination**  
< **TerminateHigh** | **TerminateLow** | **TerminateOff** | **TerminateUnknown** > ; )  
( **DefaultState** < **U** | **D** | **Z** | **ForceUp** | **ForceDown** | **ForceOff** > ; )  
( **Base** < **Hex** | **Dec** > WAVEFORM\_CHARACTER\_LIST ; )  
( **Alignment** < **MSB** | **LSB** > ; )  
( **ScanIn** (DECIMAL\_INTEGER) ; )  
( **ScanOut** (DECIMAL\_INTEGER) ; )  
( **DataBitCount** DECIMAL\_INTEGER ; )  
} ) \*  
}

**SignalGroups** (DOMAIN\_NAME) {

( GROUPNAME = *sigref\_expr* ; ) \*  
( GROUPNAME = *sigref\_expr* { ( **Termination**  
< **TerminateHigh** | **TerminateLow** | **TerminateOff** | **TerminateUnknown** > ; )  
( **DefaultState** < **U** | **D** | **Z** | **ForceUp** | **ForceDown** | **ForceOff** > ; )  
( **Base** < **Hex** | **Dec** > WAVEFORM\_CHARACTER\_LIST ; )  
( **Alignment** < **MSB** | **LSB** > ; )  
( **ScanIn** (DECIMAL\_INTEGER) ; )  
( **ScanOut** (DECIMAL\_INTEGER) ; )  
( **DataBitCount** DECIMAL\_INTEGER ; )  
} ) \*  
}

```

PatternExec (PAT_EXEC_NAME) {
  ( Category CATEGORY_NAME ; ) *
  ( Selector SELECTOR_NAME ; ) *
  ( Timing TIMING_NAME ; )
  ( PatternBurst PAT_BURST_NAME ; )
}

PatternBurst PAT_BURST_NAME {
  ( SignalGroups GROUPS_DOMAIN ; ) *
  ( MacroDefs MACROS_DOMAIN ; ) *
  ( Procedures PROCEDURES_DOMAIN ; ) *
  ( ScanStructures SCAN_NAME ; ) *
  ( Start PAT_LABEL ; )
  ( Stop PAT_LABEL ; )
  ( Termination { ( sigref_expr
    < TerminateHigh | TerminateLow | TerminateOff | TerminateUnknown > ; ) *
  } ) // end Termination
  ( PatList {
    ( PAT_NAME_OR_BURST_NAME ; ) *
    ( PAT_NAME_OR_BURST_NAME {
      ( SignalGroups GROUPS_DOMAIN ; ) *
      ( MacroDefs MACROS_DOMAIN ; ) *
      ( Procedures PROCEDURES_DOMAIN ; ) *
      ( ScanStructures SCAN_NAME ; ) *
      ( Start PAT_LABEL ; )
      ( Stop PAT_LABEL ; )
      ( Termination { ( sigref_expr
        < TerminateHigh | TerminateLow | TerminateOff | TerminateUnknown > ; ) *
      } ) // end Termination
    } ) * // end of Pat_name_or_Burst_name
  } ) + // end of PatList
} // end of PatternBurst

```

```

Timing ( TIM_DOMAIN_NAME ) {
  ( SignalGroups GROUPS_DOMAIN ; ) *
  ( WaveformTable WFT {
    ( Period time_expr ; )
    ( InheritWaveformTable (TIM.)WFT ; ) *
    Waveforms {
      ( sigref_expr (WAV) {
        ( InheritWaveform ((TIM.)WFT.)WAV ; ) *
        ( WFC {
          ( InheritWaveform (((TIM.)WFT.)WAV.)WFC ; ) *
          (( EVENT_LABEL : ) ( time_expr ) ( event ) ; ) *
        } ) * // end wfc
      }
    }
  }
}

```

```

  ( WFC_LIST {
    ( InheritWaveform (((TIM.)WFT.)WAV.)WFC ; ) *
    (( EVENT_LABEL : ) ( time_expr )
      ( < event_list ( [EVENT_NUM] ) | event > ) ; ) *
    } ) * // end wfc_list
  ( < WFC | WFC_LIST > { (( EVENT_LABEL : ) ( time_expr ) (\r\n)
    < SUB_WF_LAB ;
    | SUB_WF_LAB[N] ;
    | SUB_WF_LAB[#] ; > ) * } ) *
  } ) * // end wfc | wfc_list
} // end sigref_expr
} ) * // end Waveforms
} // end Waveforms
( SubWaveforms {
  ( SUB_WF_LAB: Duration time_expr {
    (( EVENT_LABEL : ) ( time_expr )
      ( < EVENT_LIST ( [EVENT_NUM] ) | event > ) ; ) *
    } ) * // end Duration
  } ) // end SubWaveforms
} ) // end WaveformTable
} // end Timing

```

```

Spec (SPEC_NAME) { // this block statement defines variable values for a given category
  ( Category CAT_NAME {
    ( VAR_NAME = time_expr ; ) * // define only the Typ value
    ( VAR_NAME {
      ( Min time_expr ; ) ( Typ time_expr ; ) ( Max time_expr ; ) } ) *
    } ) +
  }
}

```

```

Spec (SPEC_NAME) { // this block statement defines category values for a given variable
  ( Variable VAR_NAME {
    ( CAT_NAME = time_expr ; ) * // define only the Typ value
    ( CAT_NAME {
      ( Min time_expr ; ) ( Typ time_expr ; ) ( Max time_expr ; ) } ) *
    } ) +
  }
}

```

```

Selector SELECTOR_NAME {
  ( VAR_NAME < Min | Typ | Max | Meas > ; ) *
}

```

```

ScanStructures (SCAN_NAME) {
  ( ScanChain CHAINNAME {
    ScanLength DECIMAL_INTEGER;
    ( ScanOutLength DECIMAL_INTEGER; )
    ( ScanCells CELLNAME-LIST ; )
    ( ScanIn SIGNALNAME; )
    ( ScanOut SIGNALNAME; )
    ( ScanMasterClock SIGNALNAME-LIST ; )
    ( ScanSlaveClock SIGNALNAME-LIST; )
    ( ScanInversion < 0 | 1 >; )
  })*
}

```

```

Pattern PATTERN_NAME {
  ( LABEL : )
  ( TimeUnit 'TIME_DEF'; )
  ( V(ector) { ( cyclized-data)* ( non-cyclized-data)* } )
  ( W(aveformTable) NAME ; )
  ( C(ondition) { ( cyclized-data)* ( non-cyclized-data)* } )
  ( Call PROCNAME ; )
  ( Call PROCNAME { ( scan-data)* ( cyclized-data)* ( non-cyclized-data)* } )
  ( Macro MACRONAME ; )
  ( Macro MACRONAME { ( scan-data)* ( cyclized-data)* ( non-cyclized-data)* } )
  ( Loop LOOPCNT { ( pattern-statements)* } )
  ( MatchLoop < LOOPCNT | Infinite > {
    (pattern-statements)+
    ( BreakPoint { (pattern-statements)+ } )
  } ) // end MatchLoop
  ( Goto LABELNAME ; )
  ( BreakPoint; )
  ( BreakPoint { ( PATTERN-STATEMENTS)* } )
  ( IddqTestPoint; )
  ( Stop ; )
  ( ScanChain CHAINNAME ; )
} // end Pattern

```

#### vec\_data:

```

\w - WaveformChar format (\w0100 xxxx qqQQQ).
\rN XXX - Repeat the next group of characters N times (\r60 01).
\h - Hexadecimal format (\h E4B2).
\hCHARS - Hexadecimal format with a local WaveformChar association (\h01 69BC).
\d - Decimal format ( \d 79)
\dCHARS - Decimal format with local WaveformChar association (\dHL 375).
\IN - Shift data passed in procedure or macro calls, or multiple-bit data passed into
waveforms that contain square-bracket constructs.

```

```

Procedures ( PROCEDURE_DOMAIN_NAME ) {
  ( PROCEDURE_NAME {
    ( PATTERN-STATEMENT)*
  })*
}

```

```

MacroDefs ( MACRO_DOMAIN_NAME ) {
  ( MACRO_NAME {
    ( PATTERN-STATEMENT)*
  } )*
}

```

## IEEE P1450.1, Draft 16, June 19, 2003

```
STIL 1.0 {
  ( Design D19; )
} // end STIL

Variables (VARIABLES_DOMAIN) {
  ( IntegerConstant CONST_NAME := DECIMAL_INTEGER ; ) *
  ( Integer VAR_NAME ; ) *
  ( Integer VAR_NAME {
    ( Usage Test ; )
    ( InitialValue integer_expr ; )
  } ) * // end Integer
  ( SignalVariable VAR_NAME ; ) *
  ( SignalVariable VAR_NAME {
    ( Base < Hex | Dec > WFC_LIST ; )
    ( Alignment < MSB | LSB > ; )
    ( InitialValue vec_data ; )
  } ) * // end SignalVariable
  ( WFCConstant CONST_NAME = WFC_LIST ; ) *
} // end Variables

Signals {
  ( SIG_NAME < In | Out | InOut | Supply | Pseudo > {
    ( WFCMap {
      ( FROM_WFC -> TO_WFC( TO_WFC ) + ; ) *
      ( FROM_WFC1 FROM_WFC2 -> TO_WFC ; ) *
    } ) // end WFCMap
  } ) *
}

SignalGroups {
  ( GROUPNAME = sigref_expr {
    ( WFCMap {
      ( FROM_WFC -> TO_WFC( TO_WFC ) + ; ) *
      ( FROM_WFC1 FROM_WFC2 -> TO_WFC ; ) *
    } ) // end WFCMap
  } ) *
} // end SignalGroups

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

```
( Variables VARIABLES_DOMAIN ; ) *
  } ) * // end pat_name_or_burst_name
} ) * // end PatSet
( ParallelPatList ( SyncStart | Independent | LockStep ) {
  ( PAT_NAME_OR_BURST_NAME ; ) *
  ( PAT_NAME_OR_BURST_NAME {
    ( Variables VARIABLES_DOMAIN ; ) *
    ( Extend ; )
    ( If boolean_expr ; )
    ( While boolean_expr ; )
  } ) * // end pat_name_or_burst_name
} ) * // end ParallelPatList
```

```
Timing TIMING_NAME {
  ( Variables VARIABLES_DOMAIN ; ) *
```

```
ScanStructures (SCAN_STRUCT_NAME) {
  ( InheritScanStructures SCAN_STRUCT_NAME ; )
  ( ScanChain CHAIN_NAME {
    ScanLength integer_expr ;
    ScanOutLength integer_expr ;
    ScanEnable logic_expr ;
    ScanCellType (CELL_TYPE_NAME) {
      ( (If boolean_expr) CellIn STATE-ELEMENT-LIST ; ) *
      ( (If boolean_expr) CellOut STATE-ELEMENT ; ) *
    }
    ScanCells {
      ( cell_ref (CELL_TYPE_NAME) ; ) *
    } // end ScanCells
    ( Base Hex ; ) // optional for cell groups
    ( Alignment <MSB|LSB>; ) // optional for cell groups
  } ) * // end ScanChain
  ( ScanChainGroups {
    ( GROUP_NAME { (GROUP_OR_CHAIN_NAME ; ) * } ) *
  } ) // end ScanChainGroups
} // end ScanStructures
```

```
Environment (ENV_NAME) {
  ( InheritEnvironment ENV_NAME ; ) *
  ( NameMaps (MAP_NAME) {
    InheritNameMap (ENV_NAME::) (MAP_NAME) ;
    InheritNameMap (ENV_NAME::) (MAP_NAME) {
      ( Prefix "PREFIX_STRING" ; )
      ( Separator "SEPARATOR_STRING" ; )
      ( ScanCells { (SCAN_CELL_NAME ; ) * } ) *
    } // end InheritNameMap
    ( Prefix "PREFIX_STRING" ; )
    ( Separator "SEPARATOR_STRING" ; )
    ( ScanCells { ((SCAN_CELL_NAME) "MAP_STRING" ; ) * } ) *
    ( Signals { (SIG_NAME "MAP_STRING" ; ) * } ) *
    ( SignalGroups (DOMAIN_NAME) { (GROUP_NAME "MAP_STRING" ; ) * } ) *
    ( Variable { (VAR_NAME "MAP_STRING" ; ) * } ) *
```

```

    ( AllNames { (ANY_NAME "MAP_STRING";)* } ) *
  })* // end NameMaps
( FileReference "FILE_PATH_NAME";)*
( FileReference "FILE_PATH_NAME" {
  Type FILE_TYPE ;
  Format FILE_FORMAT ;
  Version "VERSION_NUMBER" ;
})* // end FileReference
(ENVIRONMENT_DEFINED_STATEMENTS)*
} // end Environment

```

## Data content read back - \C, \D, \E, \S, \U, \W

```

\readback_function SIGNAL_NAME
\readback_function SIGNAL_GROUP_NAME
\readback_function SIGNAL_VARIABLE_NAME
\readback_function WFCCONSTANT_NAME

```

**\C** SIGNAL-OR-GROUP-NAME - return the last compare event. In the case of a group, return a string of the last compare event for each signal of the group. If no compare event has been established, then return '~' (the tilde character).

**\D** SIGNAL-OR-GROUP-NAME - return the last drive event. In the case of a group, return a string of the last compare event for each signal of the group. If no compare event has been established, then return '~' (the tilde character).

**\E** SIGNAL-OR-GROUP-NAME - return the last expect event. In the case of a group, return a string of the last compare event for each signal of the group. If no compare event has been established, then return '~' (the tilde character).

**\S** SIGNAL-OR-GROUP-NAME - return the last wfc/wfc-list that was established using a substitute (s or S) event in a waveform. In the case of a group, return a string of wfcs for each signal of the group. The actual wfc is determined by the WFCMap statement. If no substitute has been established, then return '~' (the tilde char). For further information, see CompareSubstitute (subclause 14.2) and WFCMap statement (subclause 11.1).

**\U** SIGNAL-OR-GROUP-NAME - return the last undefined event. In the case of a group, return a string of the last compare event for each signal of the group. If no compare event has been established, then return '~' (the tilde character).

**\W** SIGNAL-OR-GROUP-OR-SIGNAL-VARIABLE-OR-WFCCONSTANT-NAME - return the last wfc/wfc-list that was established using a V or a C statement. In the case of a group, return a string of wfcs for each signal of the group. If the last wfc was established by a parameter using # or %, then the substituted wfc is returned. If no wfc has been established, then return '~' (the tilde character). Note: this should never happen on the readback of a signal or group since it is required that an initial wfc be defined on the first vector of any pattern.

## Vector data mapping and joining - \m, \j

This subclause defines syntax for use within pattern data for either mapping wfcs to other wfcs or for combining wfcs into a single wfc. The syntax is defined as:

```

SIGNAL = \m wfc;
GROUP = \m wfc-list;
SIGNAL = \j wfc-1; signal = \j wfc-2;
GROUP = \j wfc-list-1; \j wfc-list-2;

```

Where:

SIGNAL, GROUP - represent the signal that are to be mapped or joined in the pattern data

wfc - represent a FROM\_WFC that is to be mapped according to a WFCMap statement for a signal

wfc-list - represent a string of FROM\_WFCs that are to be mapped according to a WFCMap for a signal group

wfc-1, wfc-2 - represent two FROM\_WFC characters that are to be joined according to a WFCMap for a signal

wfc-list-1, wfc-list-2 - represent two strings of FROM\_WFCs that are to be joined according to a WFCMap for a signal group.

### Pattern statements:

```

( LABEL : ) If boolean_expr { (PATTERN_STATEMENTS)* } ( Else {
(PATTERN_STATEMENTS)* } )
( LABEL : ) While boolean_expr { (PATTERN_STATEMENTS)* }
( LABEL : ) F(ixed) { (cyclized-data)* (non-cyclized-data)* }
( LABEL : ) E(quivalent) ( (\m) sigref_expr ) * ;
( LABEL : ) LoopData { ( PATTERN_STATEMENTS ) * }
( LABEL : ) Loop integer_expr { ( PATTERN_STATEMENTS ) * }
( LABEL : ) ActiveScanChains (GROUP_OR_CHAIN_NAME)+ ;
( LABEL : ) AllowInterleave sigref_expr;
( LABEL : ) BreakPoint { }
( LABEL : ) X < IDENTIFIER | INTEGER > ;

```

**Pragma** (NAME) { \* ... APPLICATION DEPENDANT SYNTAX ... \* }

```

PatternFailReport (REPORT_NAME) {
( DeviceID "IDENTIFYING INFORMATION"; )
( TestConditions { } )
( Pattern PAT_NAME; )
( Pattern PAT_NAME {
( Start (PAT_LABEL) (CYCLE_COUNT); )
( Stop (PAT_LABEL) (CYCLE_COUNT); )
} )
( PatternBurst PAT_BURST_NAME; )
( PatternBurst PAT_BURST_NAME {
( Start (PAT_LABEL) (CYCLE_COUNT); )
( Stop (PAT_LABEL) (CYCLE_COUNT); )
} )
( PatternExec PAT_EXEC_NAME; )

```

```
FailData {  
  ( <IDENTIFIER|INTEGER> ( . <IDENTIFIER|INTEGER> ) ) *  
    SIG_NAME  
    ( CYCLE_OFFSET ( 'observed_data' ) ) * ;  
  ) * // end fail data record  
} // end FailData  
} // end PatternFailReport
```

```
STIL 1.0 {  
  ( DCLevels 2002; ) +  
}
```

```
PatternExec (PAT_EXEC_NAME) {  
  ( DCLevels (DC_LEVELS_NAME); )  
  ( DCSets (DC_SETS_NAME); ) 15  
}
```

```
( DCLevels (DC_LEVELS_NAME) { // DCLevels block  
  ( InheritDCLevels DC_LEVELS_REF; ) *  
  ( SignalGroups GROUPS_DOMAIN; ) *  
  ( sigref_expr { // Signals of type In, Out, InOut, Supply  
    ( VIH (dc_expr); ) ( VIL (dc_expr); )  
    ( VICM dc_expr; ) ( VID dc_expr; )  
    ( VIHD dc_expr; ) ( VILD dc_expr; )  
    ( ForceHi; ) ( ForceLo; )  
    ( InitHi; ) ( InitLo; )  
    ( VIHSlew dc_expr; ) ( VILSlew dc_expr; )  
    ( VOH (dc_expr); ) ( VOL (dc_expr); )  
    ( VOCM dc_expr; ) ( VOD dc_expr; )  
    ( VOHD dc_expr; ) ( VOLD dc_expr; )  
    ( IOH dc_expr; ) ( IOL dc_expr; )  
    ( LoadVRef dc_expr; ) ( ClampHi dc_expr; )  
    ( ClampLo dc_expr; ) ( ResistiveTermination dc_expr; )  
    ( TermVRef dc_expr; ) ( VForce dc_expr; )  
    ( IClamp dc_expr; ) ( IForce dc_expr; )  
    ( VClamp dc_expr; ) } ) * // end sigref_expr  
  } ) * // end DCLevels
```

```
( DCSets (DC_SET_NAME) {  
  ( DCLevels (DC_LEVELS_REF); ) *  
  } ) *
```

```

(DCSequence (<InitialSetup | PowerRaise | PowerLower | EndOfProgram |
  User USER_DEFINED>) {
  (SignalGroups GROUPS_DOMAIN;)*
  (time_expr sigref_expr {           // Each sigref_expr is sequenced together
    (Connect (<Supply | PMU | Driver | Comparator| Load | Termination
      |Clamp>)*;)* // Physical connection (e.g., relay)
    (Disconnect (<Supply | PMU | Driver | Comparator | Load | Termination
      |Clamp>)*;)*
    (Apply (dc_expr);)*           // optional voltage or current value
    (Ramp time_expr (dc_expr);)* // ramp duration, optional voltage target
  })* // end time_expr
})* // end DCSequence

```

## IEEE P1450.3, Draft 8, Apr 11 2003

```

STIL 1.0 { TRC D08; }

```

```

Resource (TESTER_IDENTIFIER)+ ; // list of target testers
  < (RESOURCE_ID)+ >
}

```

```

Environment (ENV_NAME) {
  ( NameMaps (MAP_NAME) {
    ( Signals { (SIG_NAME "MAP_STRING"; )* } )*)
  } // end NameMaps
} // end Environment

```

```

Environment (ENV_NAME) {
  ( TRC (TRC_NAME) {
    ( Category (CAT_NAME)+ ; )
    ( Selector (SEL_NAME)+ ; )
    ( NameChecks (NAM_CHK_NAME) { } )*)
    ( PatternCharacteristics PAT_CHAR_NAME { } )*)
    ( PeriodCharacteristics PER_CHAR_NAME { } )*)
    ( SignalCharacteristics (signal_attributes)+ { } )*)
    ( SystemCharacteristics {
      ( MultipleDevices integer_expr; )
      ( MultipleSites integer_expr; )
    } )
    ( Usage < Constraints | PatternReport > ; )
    ( WaveformCharacteristics WAV_CHAR_NAME { } )*)
    ( WaveformDescriptions WAV_DESC_NAME (Explicit) { } )*)
  })* // end TestResourceConstraints
} // end Environment

```

### SignalCharacteristics

```

(<Data| Clock | Scan| In | Out| InOut | SplitIO| Asynchronous
| User USER_DEFINED>)+ {
  ( DCResourceCharacteristics DC_CHAR_NAME; )
  ( FanOut integer_expr; )
  ( InOut < WithinCycle | OnCycleBoundary | Static >; )
  ( MaxScanMemory integer_expr; )
  ( MaxCaptureMemory integer_expr (<FailOnly | Result>)+ ; )
  ( MaxScanChainLength integer_expr ; )
  ( MaxSignals integer_expr; )
  ( MaxVectorMemory integer_expr; )
  ( PatternCharacteristics PAT_CHAR_NAME ; )
  ( PeriodCharacteristics PER_CHAR_NAME (<Synchronous | Asynchronous>)+ ; )

```

```

    ( UndrivenInOut < Yes | No >;)
    ( WaveformCharacteristics WAV_CHAR_NAME ;)
    ( WaveformDescriptions WAV_DESC_NAME ;)
} // end SignalCharacteristics

DCResourceCharacteristics (DC_RESOURCE_NAME) {
    ( PerPinConfiguration
      (<Supply | PMU | Driver | Comparator | Load | Termination | Clamp>)+; )
    ( DifferentialConfiguration (<Driver | Comparator>)+; )
    ( NumberTesterChannels integer_expr; )
    ( DCLimits
      < VIH | VIL | VICM | VID | VIHD | VILD | VIHSlew | VILSlew
        | VOH | VOL | VOCM | VOD | VOHD | VOLD
        | IOH | IOL | LoadVRef | ClampHi | ClampLo
        | ResistiveTermination | TermVRef
        | VForce | IClamp | IForce | VClamp >
      { ( Min dc_expr; ) ( Max dc_expr; ) } ) *
    ( NumberLevels (< VIH | VIL | VOH | VOL >)+ integer_expr; ) *
    ( NumberDCLevels integer_expr; )
    ( NumberDCLevels integer_expr {
      SignalsPer integer_expr;
    }
  }
}

```

```

PeriodCharacteristics (PER_CHAR_NAME) {
    (Accuracy time_expr; )
    (MaxPeriods integer_expr; )
    (MaxPeriodGenerators integer_expr (< Dynamic | Static >); )
    (MaxPeriodGenerators integer_expr (< Dynamic | Static >) {
      ( SignalsPer integer_expr; )
    } // end MaxPeriodGenerators
    ( PeriodSelectMemory integer_expr; )
    ( PeriodSelectMemory integer_expr {
      ( SignalsPer integer_expr; )
    } // end PeriodSelectMemory
    ( TimeLimits boolean_expr; )
    (Resolution time_expr; )
} // end PeriodCharacteristics

```

```

WaveformCharacteristics (WAV_CHAR_NAME) {
    ( Accuracy <Edge|EdgeToEdge> time_expr; ) *
    ( CompareEvents
      (<H|L|X|V|T|h||x|v|t>/(<H|L|X|V|T|h||x|v|t>)*)+
      (integer_expr (integer_expr)); ) * // number events, number substitutes
    ( DriveEvents

```

```

      (<U|D|Z|P>/(<U|D|Z|P>)*)+
      (integer_expr (integer_expr)); ) * // number events, number substitutes
// use the following syntax to define MaxShapes
( FormatSelect < In | Out | InOut > {
    ( MaxShapes integer_expr (< Dynamic | Static >); )
    ( MaxShapes integer_expr (< Dynamic | Static >) {
      ( SignalsPer integer_expr; )
    } // end MaxShapes
  } // end FormatSelect (with MaxShapes)
// use following syntax in place MaxShapes (above)
( FormatSelect < In | Out | InOut > {
    ( MaxTimeSets integer_expr (< Dynamic | Static >); )
    ( MaxTimeSets integer_expr (< Dynamic | Static >) {
      ( SignalsPer integer_expr; )
      ( PerTimingGenerator; )
    } // end MaxTimeSets
    ( MaxTimingGenerators integer_expr (< Dynamic | Static >); )
    ( MaxTimingGenerators integer_expr (< Dynamic | Static >) {
      ( SignalsPer integer_expr; )
    } // MaxTimingGenerators
    ( MaxData <Drive|Compare|DriveCompare> integer_expr (<Dynamic|Static>); )
    ( MaxData <Drive|Compare|DriveCompare> integer_expr (<Dynamic|Static> {
      ( SignalsPer integer_expr; )
    } // end MaxData
    ( MaxIO integer_expr integer_expr (< Dynamic | Static >); )
    ( MaxIO integer_expr integer_expr (< Dynamic | Static >) {
      ( SignalsPer integer_expr; )
    } // end MaxIO
    ( MaxMask integer_expr (< Dynamic | Static >); )
    ( MaxMask integer_expr (< Dynamic | Static >) {
      ( SignalsPer integer_expr; )
    } // end MaxMask
  } ) * // end FormatSelect
    ( MaxEdgeTime time_expr; )
    ( MinCompareWindow time_expr; )
    ( MinCompareToDriveOn time_expr; )
    ( MinEdgeReTrigger time_expr (<Drive | Compare>)* ; ) *
    ( MinDriveOffTime time_expr; )
    ( MinDriveOffToCompare time_expr; )
    ( MinDriveOnTime time_expr; )
    ( MinDrivePulse time_expr; )
    ( ReplicateSubWaveform integer_expr; )
    ( Resolution time_expr; )
    ( TimeLimits boolean_expr; )
    ( WaveformSelectMemory integer_expr ; )

```

```

( WaveformSelectMemory integer_expr {
  ( SignalsPer integer_expr; )
  ( SelectWithPeriod; )
} ) // end WaveformSelectMemory
} // end WaveformCharacteristics

```

```

WaveformDescriptions (WAV_DESC_NAME) (Explicit) {
  ( < In | Out | InOut > WFNAME {
    ( NumberData integer_expr; )
    ( NumberIO integer_expr; )
    ( NumberMask integer_expr; )
    ( NumberPeriods integer_expr; )
    ( NumberShapes integer_expr; )
    ( NumberSignals integer_expr; )
    ( NumberTimeSets integer_expr; )
    Shape {
      ( ( TRC_LABEL: ) ( time_expr ) < event | event_list > ;)*
    })* // end Shape
  })* // end < In | Out | InOut >
} // end WaveformDescriptions

```

```

PatternCharacteristics (PAT_CHAR_NAME) {
  ( Base < Hex | Dec | HexDec > integer; )
  ( BreakPoint < Yes | No >; )
  ( ConditionalStatements < Yes | No >; )
  ( CoreUsageReady < Yes | No >; )
  ( DeltaChangeVectorData < Yes | No >; )
  ( GoTo < Yes | No >; )
  ( IddqTestPoints < Yes | No >; )
  ( InstructionCharacteristics {
    ( AllowedWhen boolean_expr; )
    ( AppliesTo ( < Condition | GoTo | Loop | MatchLoop | Call | Shift > )+ ; )
    ( LoopCharacteristics LOOP_NAME {
      ( Infinite; )
      ( MaxIteration integer_expr; )
      ( MaxLength integer_expr; )
      ( MaxNest integer_expr; )
      ( MinIteration integer_expr; )
      ( MinLength integer_expr; )
      ( MinTimeAfterMatch time_expr; )
      ( MinVectorsAfterMatch integer_expr; )
      ( VectorModulus integer_expr; )
    })* // end LoopCharacteristics
    ( MinVectorsAfter integer_expr; )
    ( MinVectorsBefore integer_expr; )
    ( MinVectorsBetween integer_expr; )

```

```

  })* // end InstructionCharacteristics
  ( Macro < Yes | No | WithParameters >; )
  ( MaxRunTime time_expr; )
  ( MaxVectors integer_expr; )
  ( MultiBitData ( < InWaveforms | InPatterns | No > )+ ; )
  ( NonCyclized < Yes | No >; )
  ( NumberCaptureCycles ( integer_expr ( integer_expr ) ); )
  ( NumberPatternUnits integer_expr; )
  ( NumberVectorsPerShift ( integer_expr ) ( integer_expr ); )
  ( PatternVariables ( < Integer | IntegerConstant | SignalVariable | Spec | All >* ; )
  ( ProcedureCalls < Yes | No | WithParameters >; )
  ( Shift < Yes | No >; )
  ( STILPatterns < Yes | No >; )
  ( VectorModulus integer_expr; )
} // end PatternCharacteristics

```

```

PatternInformation {
  < Pattern | PatternBurst > PAT_OR_BURST_NAME {
    ( PatternCharacteristics { } )
  }
NameChecks NAM_CHECKS_NAME {
  ( Contents ( STIL_BLOCK_NAME )+ ; )
  ( Block STIL_BLOCK_NAME ; )*
  ( CharacterContent ' regular_expr ' ; )
  ( Length integer_expr; )
  ( Scope ( STIL_BLOCK_NAME )+ ; )
}

```

## IEEE P1450.6, Draft D1.2, Aug 2004

**STIL 1.0** { **Design** D20; **CTL** D1.2; }

```
PatternBurst PAT_BURST_NAME {
  (Protocol <Macro MACRONAME (SETUP_MACRONAME)
    | Procedure PROCNAME (SETUP_PROCNAME)>;)
  (<PatList | PatSet |
    ParallelPatList (SyncStart | Independent | LockStep)>
  {
    (PAT_NAME_OR_BURST_NAME {
      (Protocol <Macro MACRONAME (SETUP_MACRONAME)
        | Procedure PROCNAME (SETUP_PROCNAME)>;)
    })*
  })+
}
```

// ONLY syntax allowed in Patterns

```
Pattern PATTERN_NAME {
  (Setup { (sigref_expr = value_variable_expr;)* })
  <((LABEL:) P;)* |
  ((LABEL:) P { (sigref_expr = value_variable_expr;)* })*>
  // Pattern-statements that are allowed from the ones
  defined in
  // IEEE Std. 1450-1999 and 1450.1
  ((LABEL:) Loop LOOPCNT { (P-statements)* })*
  ((LABEL:) Goto LABELNAME;)*
  ((LABEL:) BreakPoint;)*
  ((LABEL:) X TAG;)*
}
```

// OR

```
Pattern PATTERN_NAME {
  // Pattern-statements that are allowed from the ones
  defined in
  // IEEE Std. 1450-1999 and 1450.1
  ((LABEL:) Macro MACRONAME;)*
  ((LABEL:) Macro MACRONAME { (sigref_expr = value_variable_expr;)* })*
  ((LABEL:) Call PROCNAME;)*
  ((LABEL:) Call PROCNAME { (sigref_expr = value_variable_expr;)* })*
  ((LABEL:) Loop LOOPCNT { (Macro-or-Procedure-calls)* })*
  ((LABEL:) Goto LABELNAME;)*
  ((LABEL:) BreakPoint;)*
  ((LABEL:) X TAG;)*
}
```

```
MacroDefs (MACRO_DOMAIN_NAME) {
  ( MACRO_NAME (LockStep) {
    (PATTERN_STATEMENT)*
  })*
}
```

```
Procedures (PROCEDURE_DOMAIN_NAME) {
  ( PROCEDURE_NAME (LockStep) {
    (PATTERN_STATEMENT)*
  })*
}
```

```
(CoreType CORE_TYPE_NAME {
  ( CoreEnvironment CORE_ENV_NAME; )
  ( CoreInstance CORE_TYPE_NAME {
    (CORE_INSTANCE_NAME;)+
  })* // end Inherited CoreType- Instance definition
  ( Signals {
    core_signal_definitions
  }) // end Signals
  ( SignalGroups (DOMAIN_NAME) {
    core_signal_group_definitions
  })* // end SignalGroups
  ( ScanStructures (SCAN_STRUCT_NAME) {
    core_scan_structure_definitions
  })* // end ScanStructures
})* // end CoreType
```

```
( CoreInstance CORE_TYPE_NAME {
  (CORE_INSTANCE_NAME;)+
})*
```

```
( FileReference "FILE_PATH_NAME";)*
```

```
( FileReference "FILE_PATH_NAME" {
  Type file_type ;
  Format <pattern_file_format | design_file_format | layout_file_format |
    fault_list_file_format | script_file_format | doc_file_format>;
  Version "VERSION_NUMBER" ;
})* // end FileReference
```

file\_type =

```
< Pattern | Design
  | Layout | FaultList
  | Script | Documentation
  | User USER_DEFINED >
```

```

pattern_file_format =
  < STIL          | WGL
  | Verilog      | VHDL
  | VCD          | User USER_DEFINED >

```

```

design_file_format =
  < CTL          | EDIF
  | Verilog      | VHDL
  | User USER_DEFINED >

```

```

layout_file_format =
  < GDSII        | DEF
  | LEF          | Oasis
  | User USER_DEFINED >

```

```

fault_list_file_format =
  < DTIF          | User USER_DEFINED >

```

```

script_file_format =
  < AWK          | Perl
  | SED          | TCL
  | User USER_DEFINED >

```

```

doc_file_format =
  < HTML         | PDF
  | Postscript   | RTF
  | Text         | User USER_DEFINED >

```

```

(NameMaps (MAP_NAME) {
  (CoreType CORE_TYPE_NAME {
    (CoreInstance {(CORE_INSTANCE_NAME "map_string");+})
    (Signals { (SIGNALNAME "map_string");+})
    (ScanCells { (CELLNAME "map_string");+ }
  })*
  (CoreInstance { (CORE_INSTANCE_NAME "map_string");+ }
})

```

```

test_mode_enum =
  < Bypass          | Controller
  | Debug          | ExternalTest
  | ForInheritOnly | InternalTest
  | Isolate        | Normal
  | PowerOff       | PowerOn
  | PreLoad        | Quiet
  | Repair         | Sample
  | Transparent    | User USER_DEFINED >

```

```

exec_enum =
  < Characterization | Diagnostic
  | Production      | Verification
  | User USER_DEFINED >

```

```

pattern_or_burst_enum =
  < AtSpeed          | ChainContinuity
  | CompatibilityInformation | Endurance
  | EstablishMode    | IDDQ
  | LogicBIST        | MemoryBIST
  | Padding          | Parametric
  | Retention        | Scan
  | TerminateMode    | User USER_DEFINED >

```

```

Environment (ENV_NAME) {
  ( CTLMODE (CTLMODE_NAME) {
    ( CoreInternal { } )
    ( DomainReferences ( CORE_INSTANCE_NAME)* {
      ( Category (CATEGORY_NAME)+; )
      ( MacroDefs ( MACRO_DEF_NAME)+ ; )
      ( Procedures ( PROCEDURE_NAME)+ ; )
      ( Selector ( SELECTOR_NAME)+; )
      ( SignalGroups ( SIGNALGROUPS_NAME)+ ; )
      ( ScanStructures ( SCAN_STRUCT_NAME)+ ; )
      ( Timing ( TIMING_NAME)+ ; )
      ( Variables ( VARIABLES_NAME)+; )
    })* // end DomainReferences
    ( External { } )
    ( Family (NAME)+ ; )
    ( Focus ( Top) (CoreInstance (CORE_INSTANCE_NAME)+ ) {
      ( PatternTypes (pattern_or_burst_enum)+ ; )
      ( CTLMODE CORE_INSTANCE_NAME CTLMODE_NAME; )*
      ( TestMode (test_mode_enum)+ ; )
      ( Usage (exec_enum)+ ; )
    })* // end Focus
  }

```

```

( InheritCTLMODE CTLMODE_NAME; )
( Internal { } )
( PatternInformation { } )
( Relation { } )
( ScanInternal { } )
( ScanRelation { } )
( TestMode ( test_mode_enum )+ ; )
( TestMode ( test_mode_enum )+ {
    AlternateTestMode (CTLMODE_NAME)+ ;
} ) // end of TestMode
( TestModeForWrapper WRAPPER_TEST_MODE TEST_MODE_CODE; )
( Vendor (NAME)+ ; )
( Compliance < IEEE1500 EXT_VERSION < Wrapped | Unwrapped >
    | None | User USER_DEFINED > ; )
})* // end CTLMODE
} // end Environment

```

```

data_type_enum =
< Asynchronous | Synchronous
| In | Out
| InOut | Constant
| TestMode | Unused
| UnusedDuringTest | Functional
| TestControl ( testcontrol_subtype_enum )*
| TestData ( testdata_subtype_enum )*
| User USER_DEFINED >

```

```

testcontrol_subtype_enum =
< CaptureClock | CoreSelect
| ClockEnable | InOutControl
| Oscillator | OutDisable
| OutEnable | MasterClock
| MemoryRead | MemoryWrite
| SlaveClock | Reset
| ScanEnable | ScanMasterClock
| ScanSlaveClock | TestAlgorithm
| TestInterrupt | TestPortSelect
| TestRun | TestWrapperControl >

```

```

testdata_subtype_enum =
< MemoryAddress ( Row | Column)*
| MemoryData
| Indicator ( TestDone | TestFail | TestInvalid)*
| Regular
| ScanDataIn
| ScanDataOut >

```

```

cell_enum =
< < (WC_S | WH_S) <D|F># | WH_C | WH_CI >
( _C <<I|O|B>><I|O|U>|N > )
( _U <D|F> ) ( _O ) ( <_G0|_G1 > )
| User USER_DEFINED >

```

```

relation_enum =
< Bus | Common
| Corresponding | Differential
| Equivalent | Independent
| InOutSet | MuxSet
| NotEquivalent | OneCold
| OneHot | Port
| ZeroOneHot | ZeroOneCold
| User USER_DEFINED >

```

```

ScanDataType_enum =
< AddressGenerator | Boundary
| Bypass | Counter
| DataGenerator | Identification
| Instruction | Internal
| ResponseCompactor | User USER_DEFINED >

```

```

Environment { CTLMODE (CTLMODE_NAME) {
    Internal {
        ( sigref_expr {
            ( DataType ( data_type_enum )+ ; )
            ( DataType ( data_type_enum )+ {
                ( ActiveState
                    < ForceDown | ForceUp
                    | ForceOff | ForceValid
                    | ExpectLow | ExpectHigh
                    | ExpectOff | ExpectValid > (Weak); )
                ( DataRateForProtocol <Average|Maximum> INTEGER; )
                ( AssumedInitialState
                    < ForceDown | ForceUp
                    | ForceOff | ForceValid

```

```

    | ExpectLow          | ExpectHigh
    | ExpectOff         | ExpectValid > (Weak) ; )
( ScanDataType ( ScanDataType_enum )+ ; )
( ValueRange INTEGER INTEGER (CORE_INSTANCE_NAME)+ ; ) *
( UnusedRange INTEGER INTEGER ; ) *
} ) * // end DataType
( DisableState
  < ExpectOff          | ExpectLow
  | ExpectHigh         | ExpectValid
  > ( Weak ) ; )
( DriveRequirements {
  ( TimingNonSensitive ; )
  ( TimingSensitive {
    ( Period < Min | Max > time_expr ; ) *
    ( Pulse < High | Low > < Min | Max > time_expr ; ) *
    ( Precision time_expr ; )
    ( EarliestTime time_expr ; )
    ( LatestTime time_expr ; )
    ( Reference sigref_expr {
      ( SelfEdge
        < Leading | Trailing | LeadingTrailing > (INTEGER) ; )
      ( ReferenceEdge
        < Leading | Trailing | LeadingTrailing > (INTEGER) ; )
      ( Hold time_expr ; )
      ( Setup time_expr ; )
      ( Period real_expr ; )
      ( Precision time_expr ; )
    } ) * // end Reference
    ( Waveform ; )
  } ) // end TimingSensitive
} ) // end DriveRequirements
( ElectricalProperty
  < Digital          | Analog
  | Power            | Ground
  > (ELECTRICAL_PROPERTY_ID) ; )
( InputProperty
  ( < Edge          | GlitchFree
  | PullDown       | PullUp
  | Relation relation_enum | ScanStable
  | ScanUnstable   | SynchFF
  | SynchLatch    | Transitions (INTEGER)
  | Window         | User USER_DEFINED
  > )+ ; )
( IsConnected <In|Out> (<Direct | Transitive>) {
  ( CoreSignal (sigref_expr) ; )

```

```

  ( IsGatedBy <LogicAnd | LogicOr | LogicXor> logic_expr {
    ( LOGICSIGNAME {
      Type < Signal | StateElement Scan | StateElement NonScan |
CoreSignal > ;
      Name <SIGNAME | CELLNAME > ;
    } )+ // end logicsigname
  } ) * // end IsGatedBy
( IsGatedBy < Macro | Procedure > NAME ; ) *
( TestAccess ( <
  Control
  | Observe
  | User USER_DEFINED > )+
  < Macro | Procedure > NAME ; ) *
( < LaunchClock | CaptureClock | Reset > SIGNAME {
  ( < LeadingEdge | TrailingEdge > (LevelSensitive) ; )
  ( < Direct | Transitive > ; )
  ( StateAfterEvent <
    Connection          | ExpectLow
    | ExpectHigh         | ExpectUnknown
    | ExpectValid       | Hold
    | Invert             | ShiftState
    | User USER_DEFINED > ; )
  } ) * // end LaunchClock
( Signal (sigref_expr) ; )
( StateElement <Scan | NonScan> (cellref_expr) ; )
( Transform {
  ( WaveformTable (WFT_NAME)+ ; )
  ( Invert ; )
  ( WFCMap FROM_WFC -> TO_WFC ; ) *
  ( DelayCycles INTEGER ; )
} ) * // end Transform
( Wrapper <IEEE1500 | None | User USER_DEFINED > ( CellID
cell_enum) ; )
} ) * // end IsConnected
( IsDisabledBy <In|Out> Logic logic_expr {
  ( LOGICSIGNAME {
    Type < Signal | StateElement Scan | StateElement NonScan |
CoreSignal > ;
    Name <SIGNAME | CELLNAME > ;
  } )+
  } ) * // end IsDisabledBy
( IsDisabledBy <In|Out> < Macro | Procedure > NAME ; ) *
( < LaunchClock | CaptureClock > SIGNAME {
  ( < LeadingEdge | TrailingEdge > (LevelSensitive) ; )
  } ) * // end LaunchClock

```

```

(OutputProperty
  (< Edge                | PullDown
   | PullUp              | Relation relation_enum
   | ScanStable         | ScanUnstable
   | SynchFF            | SynchLatch
   | ThreeState        | TwoState0Z
   | TwoState1Z        | Transition INTEGER
   | Window            | User USER_DEFINED > )+ ;)
(StrobeRequirements {
  (TimingNonSensitive; )
  (TimingSensitive {
    (Precision time_expr; )
    (EarliestTimeValid time_expr; )
    (LatestTimeValid time_expr; )
    (EarliestChange time_expr; )
    (Reference sigref_expr {
      (SelfEdge < Leading | Trailing | LeadingTrailing > INTEGER; )
      (ReferenceEdge < Leading | Trailing | LeadingTrailing >
INTEGER; )
      (EarliestTimeValid time_expr; )
      (LatestTimeValid time_expr; )
      (EarliestChange time_expr; )
      (Precision time_expr; )
    })* // end Reference
    (Waveform;)
  } ) // end TimingSensitive
} ) // end StrobeRequirements
(ScanStyle < MultiplexedData | MultiplexedClock > (LevelSensitive) ; )
(Wrapper < IEEE1500 | None | User USER_DEFINED > (PinID
USER_DEFINED_PIN_ID ) ; )
} )+ // end sigref_expr
} // end Internal
}} // end Environment, CTLMode

Environment { CTLMode (CTLMODE_NAME) {
  ScanInternal {
    (cellref_expr {
      (DataType ( data_type_enum )+ ; )
      (DataType ( data_type_enum )+ {
        (ActiveState
          < ForceDown          | ForceUp
          | ForceOff           | ForceValid
          | ExpectLow          | ExpectHigh
          | ExpectOff         | ExpectValid > ; )
        (ScanDataType ( ScanDataType_enum )+ ; )
      } )
    } )
  }
}

```

```

  (ValueRange INTEGER INTEGER (CORE_INSTANCE_NAME)+; )*
  (UnusedRange INTEGER INTEGER;)*
})* // end DataType
(IsConnected < In|Out > {
  (CoreSignal sigref_expr ; )
  (StateElement < Scan | NonScan > (cellref_expr) ; )
  (IsGatedBy < LogicAnd | LogicOr | LogicXor > logic_expr {
    (LOGICSIGNAME {
      Type < Signal | StateElement Scan | StateElement NonScan |
CoreSignal > ;
      Name < SIGNAME | CELLNAME | SYMBOLNAME > ;
    } )+ // end logicsigname
  } )* // end IsGatedBy
  (IsGatedBy < Macro | Procedure > NAME ; )*
  (< LaunchClock | CaptureClock | Reset > SIGNAME {
    (< LeadingEdge | TrailingEdge > (LevelSensitive); )
    (< Direct | Transitive >; )
    (StateAfterEvent <
      Connection          | ExpectLow
      | ExpectHigh        | ExpectUnknown
      | ExpectValid       | Hold
      | Invert             | ShiftState
      | User USER_DEFINED > ; )
  } )* // end LaunchClock
  (ScanDataType (ScanDataType_enum)+);
  (TestAccess <
    (Control                | Observe
     | User USER_DEFINED )+ >
    < Macro | Procedure > NAME ; )*
  (Transform {
    (WaveformTable (WFT_NAME)+; )
    (Invert ; )
    (WFCMap FROM_WFC -> TO_WFC; )*
    (DelayCycles INTEGER; )
  } )* // end Transform
  } )* // end IsConnected
} )+ // end cellref_expr
} // end ScanInternal
}} // end Environment, CTLMode

Environment { CTLMode (CTLMODE_NAME) {
  CoreInternal {
    (sigref_expr { //format = coreinstance:signature
      (DataType ( data_type_enum )+ ; )
      (DataType ( data_type_enum )+ {

```

```

( ActiveState
  < ForceDown          | ForceUp
  | ForceOff           | ForceValid
  | ExpectLow          | ExpectHigh
  | ExpectOff         | ExpectValid >;)
( ScanDataType ( ScanDataType_enum )+ ; )
( ValueRange INTEGER INTEGER (CORE_INSTANCE_NAME)+; )*
( UnusedRange INTEGER INTEGER;)*

})* // end DataType
( IsConnected < In|Out > {
  ( CoreSignal sigref_expr ; )
  ( IsGatedBy < LogicAnd | LogicOr | LogicXor > logic_expr {
    ( LOGICSIGNAME {
      Type < Signal | StateElement Scan | StateElement NonScan |
CoreSignal >;
      Name < SIGNAME | CELLNAME | SYMBOLNAME >;
    } )+ // end logicsigname
  } ) * // end IsGatedBy
  ( IsGatedBy < Macro | Procedure > NAME ; ) *
  ( TestAccess
    ( Control
      | Observe
      | User USER_DEFINED )+
    < Macro | Procedure > NAME ; ) *
  ( Transform {
    ( WaveformTable (WFT_NAME)+ ; )
    ( Invert ; )
    ( WFCMap FROM_WFC -> TO_WFC ; ) *
    ( DelayCycles INTEGER ; )
  } ) * // end Transform
  } ) * // end IsConnected
  } )+ // end cellref_expr
} // end CoreInternal
} } // end Environment, CTLMode

Environment { CTLMode (CTLMODE_NAME) {
  Relation {
    ( Bus sigref_expr (BUSNAME);)*
    ( Common sigref_expr ; ) *
    ( Corresponding sigref_expr ; ) *
    ( Differential sigref_expr (sigref_expr) ; ) *
    ( Equivalent sigref_expr (sigref_expr) ; ) *
    ( Independent sigref_expr (sigref_expr (Set)) ; ) *
    ( InOutSet signame signame (signame) ; ) * // enable, output name, input name

```

```

    ( MuxSet sigref_expr (sigref_expr { tag } )+ ; ) *
    ( NotEquivalent sigref_expr sigref_expr; ) *
    ( OneCold sigref_expr; ) *
    ( OneHot sigref_expr; ) *
    ( Port sigref_expr (integer) ; ) * // port data, port select value
    ( ZeroOneCold sigref_expr; ) *
    ( ZeroOneHot sigref_expr; ) *
  } // end Relation
} } // end Environment, CTLMode

```

```

Environment { CTLMode (CTLMODE_NAME) {
  ScanRelation {
    ( Differential cellref_expr (cellref_expr) ; ) *
    ( Equivalent cellref_expr (cellref_expr) ; ) *
    ( OneCold cellref_expr; ) *
    ( OneHot cellref_expr; ) *
    ( ZeroOneCold cellref_expr; ) *
    ( ZeroOneHot cellref_expr; ) *
  } // end ScanRelation
} } // end Environment, CTLMode

```

```

Environment { CTLMode (CTLMODE_NAME) {
  External {
    ( sigref_expr {
      ( AllowSharedConnection {
        ( Core NAME (sigref_expr) ; ) *
        ( DataType (data_type_enum) + ; )
        ( Family (NAME) + ; )
        ( OutputFunction < And | Or | Xor > ; )
        ( Self sigref_expr ; )
        ( Vendor (NAME) + ; )
      } ) * // end AllowSharedConnection
      ( ConnectTo {
        ( Core NAME sigref_expr; )
        ( DataType (data_type_enum) + ; )
        ( Fanout INTEGER; )
        ( Instruction; )
        ( NoRebuffering; )
        ( Symbolic (SYMBOLIC_NAME) + ; )
        ( TAM; )
        ( Termination < TerminateHigh | TerminateLow | TerminateOff |
TerminateUnknown >
          (TERMINATION_VALUE);)
        ( Safe; )
        ( WBR; )

```

```

    ( Wrapper
      <IEEE1500 | None | User USER_DEFINED >
      (< CellID cell_enum | PinID USER_DEFINED_PIN_ID >); )
  ))* // end ConnectTo
  })+ // end sigref_expr
  } // end External
}} // end Environment, CTLMode

```

```

procedure_or_macro_enum =
< Capture | Control
| DoTest | DoTestOverlap
| Hold | Instruction
| MemoryPrecharge | MemoryRead
| MemoryReadModifyWrite | MemoryRefresh
| MemoryWrite | ModeControl
| Observe | Operate
| ShiftIn | ShiftOut
| Transfer | Update
| User USER_DEFINED >

```

```

identifier_event_enum=
<Capture | Control
| ControlObserve | DataFromCurrentActivity
| DataFromPriorActivity | DataFromCurrentAndPriorActivity
| Hold | Measure
| Observe | TestPatternUnit
| Reference | Transfer
| Update | User USER_DEFINED >

```

```

Environment { CTLMode (CTLMODE_NAME) {
  PatternInformation {
    (< Pattern | PatternBurst > (pat_or_burst_name) {
      (Purpose (pattern_or_burst_enum)+; )
      (CoreInstance (CORE_INSTANCE_NAME)+; )
      (CycleCount integer; )
      (Power power_expr < Average | Maximum > ;)*
      (Fault { })* // see below for Fault block syntax
      (FileName FILE_NAME ; )
      (ForeignPatterns {
        (BlockName NAME; )
        (BeginLine LINE_NUM; )
        (EndLine LINE_NUM; )
        (BeginLabel LABEL; )
        (EndLabel LABEL; )
      } ) // end ForeignPatterns
    }
  }
}

```

```

( Identifiers { } )* // see below for Identifiers block syntax
( Protocol <Macro | Procedure>
  MACRO_OR_PROC_NAME (SETUP_MACRO_OR_PROC_NAME); )
  ))* // end Pattern | PatternBurst
( PatternExec (EXEC_NAME) {
  (Purpose (exec_enum)+; )
  (PatternBurst (BURST_NAME)+; )
  (CycleCount integer; )
  (Power power_expr < Average | Maximum > ;)*
  (Fault { })* // see below for Fault block syntax
  } ))* // end PatternExec
( < Procedure | Macro > PROCEDURE_OR_MACRO_NAME {
  (Purpose ( procedure_or_macro_enum )+; )
  (ScanChain (CHAINNAME)+;)*
  (UseByPattern (pattern_or_burst_enum)+; )
  (Identifiers { } )* // see below for Identifiers block syntax
  } ))* // end Procedure | Macro
( WaveformTable (WFT)+ {
  (Purpose (< Shift | Capture | Data | User USER_DEFINED >)+; )
  (WaveformChar (WFC)+
    (< Level | Pulse | DoublePulse | Complex > )
    (< Critical | NonCritical > )
    (Measure); ))*
  } ))* // end WaveformTable
} // end PatternInformation
}} // end Environment, CTLMode

```

```

Identifiers (PATTERN_OR_BURST_ENUM)* {
  (EventType identifier_event_enum {
    (< (< Label | Xref > LABEL_ID {
      (< Prefix | Complete > ; )
      (< Begin | During | End > ; )
      (SequenceNumber INTEGER; )
      (EventValue (time_expr)+; )
    } )+ // end Label | Xref
    | ( Variable (VAR_NAME (values)*); )+ >
  } ))* // end EventType
})* // end Identifiers

```

```

Fault {
  ( Type
    < Toggle                | StuckAt
      | StuckOpen           | Transition
      | Path                 | Bridge
      | PseudoStuckAt       | User USER_DEFINED >
    (<Collapsed | UnCollapsed | Estimated>);)
  ( Boundary < Cell | Primitive >; )
  ( FaultCount integer; )
  ( FaultsDetected integer; )
  ( FaultsDetected integer {
    (Simulation integer;)
    (Implication integer;)
    (Robustly integer;)
  })
  ( FaultsPossiblyDetected integer; ) // detection credit with X
  ( FaultsPossiblyDetected integer {
    (PossibleX integer; )
    (Oscillatory integer; )
    (Hypertrophic integer; )
  })
  ( FaultsUntestable integer; )
  ( FaultsUntestable integer {
    (Dangling integer;)
    (FixedValues integer;)
    (Blocked integer;)
    (Redundant integer;)
  })
  ( FaultsNotDetected integer; )
  ( FaultsNotDetected integer {
    (Uncontrolled integer;)
    (Unobserved integer;)
    (ATPGlimitations integer; )
    (Oscillatory integer; )
    (Hypertrophic integer; )
  })
  ( MultiplyDetected integer {
    FaultsDetected ...
    FaultsPossiblyDetected ...
    FaultsUntestable ...
    FaultsNotDetected ...
  })*
} // end Fault

```

### STIL name spaces

STIL block	Type of Name	Domain restrictions
Environment	Environment domain names	Supports a single unnamed block and domain named blocks. Domain names shall be unique across all Environment blocks.
Variables	Variables domain names	Supports a single unnamed block and domain named blocks. Domain names shall be unique across all Variables blocks.
Variables, Signals, SignalGroups, Spec	Signal names SignalGroup names SignalVariable names Spec variable names Integer names IntegerConstant names WFCConstant names	Names are present in this name space are dependent on the domain reference statements in the PatternBurst (SignalGroups domains, Variables domains) and the PatternExec (Category domains). It is an error for defined Variable, or Constant names to conflict with Signals, SignalGroups, or Spec variable names accessible in the same context.
ScanStructures	ScanCell names ScanChain names (also used for scan segments and cell groups)	Names present in this space are dependent on the domain reference statements for ScanStructures in the PatternBurst. Scan cell names, scan chain names, and cell group names in this common name space are unique except if an unnamed ScanStructures block exists. Then the names present in that unnamed block are available unless hidden by a definition of the same name in a named and referenced ScanStructure block.

### Operators and functions allowed in expressions

Op	Definition	time	real	integer	logic	bool-arith	bool-sig	sigvar	sigref	user func	stmt token
<b>min ()</b>	minimum value	Y	Y	Y							
<b>max ()</b>	maximum value	Y	Y	Y							
<b>Merged-Scan ()</b>	boolean function that returns true to select BreakPoint or other option during scan operation (sub-clause 6.10).		Y	Y		Y	Y				
<b>()</b>	parenthesis	Y	Y	Y	Y	Y	Y				
<b>,</b>	comma - used as a separator in user functions									Y	
<b>table 3, table 4 of STIL.0</b>	SI units & prefixes	Y				Y					
<b>.</b>	dot - used as a domain reference operator: xxx.yyy means item yyy within object xxx (used only in Timing context of STIL.0)										Y
<b>:</b>	single colon - reserved for referencing sub-components of a design (i.e., embedded cores). When used in conjunction with double colon, the usage shall be as follows: ccc:ddd::xxx means name 'xxx' within core 'ccc' with domain 'ddd'.										

### Operators and functions allowed in expressions

Op	Definition	time	real	integer	logic	bool-arith	bool-sig	sigvar	sigref	user func	stmt token
<b>::</b>	double colon - used as a domain reference operator: ddd::yyy means name 'yyy' within domain 'ddd'										Y
<b>!</b>	negation (boolean value)			Y		Y					
<b>~</b>	bit-wise negation			Y	Y		Y				
<b>@</b>	timing event reference	Y									
<b>0x</b>	used to define hexadecimal constants - 0xFF			Y							
<b>.</b>	dot operator used as string concatenation (for long strings): xxx.yyy yields string xxx-yyy										Y
<b>/</b>	divide	Y	Y	Y		Y					
<b>*</b>	multiply	Y	Y	Y		Y					
<b>%</b>	modulus			Y		Y					
<b>+</b>	add	Y	Y	Y		Y				Y	
<b>-</b>	subtract	Y	Y	Y		Y				Y	
<b>&lt;</b>	less than (boolean value)	Y	Y	Y		Y					
<b>&gt;</b>	greater than (boolean value)	Y	Y	Y		Y					
<b>&lt;=</b>	less or equal (boolean value)	Y	Y	Y		Y					
<b>&gt;=</b>	greater or equal (boolean value)	Y	Y	Y		Y					
<b>&amp;&amp;</b>	and (boolean value)			Y		Y	Y				

### Operators and functions allowed in expressions

Op	Definition	time	real	integer	logic	bool-arith	bool-sig	sigvar	sigref	user func	stmt token
	or (boolean value)			Y		Y	Y				
==	equal (boolean, wfc)						Y				
!=	not equal (boolean, wfc)						Y				
===	equal (boolean, real and integer)					Y					
!:=	not equal (boolean, real and integer)					Y					
&	bit-wise and			Y	Y						
	bit-wise inclusive or			Y	Y						
^	bit-wise exclusive or			Y	Y						
^~, ~^	bit-wise equivalence			Y	Y						
?:	conditional expression	Y	Y	Y	Y	Y	Y				
=	assignment (wfc)							Y	Y		
:=	assignment (real and integer)	Y	Y	Y							
..	range operator (ellipsis)							Y	Y		

### Backslash pattern data operators

Operator	Result	Std	Function	Usage <sup>a</sup> (defined in bnf)
\d	wfc	STIL.0	decimal representation of WFC's	\d integer term
\e	event	STIL.0	list of events	\e event_list term
\h	wfc	STIL.0	hex representation of WFC's	\h hex_number term
\j	wfc	STIL.1	join WFC's on two signals	\j < wfc_ref > term
\l	wfc	STIL.0	length specifier	\l integer < wfc_ref   \h hex_number   \d integer   \e event_list > term
\m	wfc	STIL.1	map WFC to WFC	\m < wfc_ref > term
\r	wfc	STIL.0	repeat list of WFC's	\r integer < wfc_ref   \h hex_number   \d integer   \e event_list > term
\w	wfc	STIL.0	list of WFC's	\w wfc_list term
\C	event	STIL.1	return list of compare events	\C wfc_container term
\D	event	STIL.1	return list of compare events	\D wfc_container term
\E	event	STIL.1	return list of compare events	\E wfc_container term
\S	wfc	STIL.1	return substitute WFC's (i.e. read back)	\S wfc_container term
\U	event	STIL.1	return list of compare events	\U wfc_container term
\W	wfc	STIL.1	return list of WFC's	\W wfc_container term

a. The following syntax definitions apply:

\*\_name = as defined in STIL.0 subclause 6.10 (note: allows "name" and name[n..m] formats)  
integer = as defined in STIL.0 subclause 6.12  
hex\_number = as defined in STIL.0 subclause 6.12  
wfc\_list = as defined in STIL.0 subclause 6.15  
wfc\_container = < signal\_name | signal\_group\_name | signal\_variable\_name | wfc\_constant\_name >  
wfc\_ref = < wfc\_list | \m wfc\_list | "\W"wfc\_container > term  
drive\_event = < "D" | "U" | "Z" | "P" >  
compare\_event = < "L" | "H" | "X" | "x" | "T" | "V" | "I" | "h" | "t" | "v" | "S" | "s" >  
expect\_event = < "R" | "G" | "Q" | "M" >  
unresolved event = < "N" | "S" | "A" | "B" | "F" | "?" >  
event\_list = (< drive\_event | compare\_event | expect\_event | unresolved event >)+  
term = < " " | ";" >

### Waveform events in pattern data - using \e

	<b>D</b>	<b>Drive Action</b>	<b>C</b>	<b>Compare Action</b>
High	U	drive high at start of period	H, h	H = high edge strobe at period start; h = start high window compare at period start
Low	D	drive low at start of period	L, l	L = low edge strobe at period start; l = start low window compare at period start
Off	Z	drive off at start of period	T, t	T = tri-state edge strobe at period start; t = start tri-state window compare at period start
Don't compare	n/a	n/a	X, x	don't compare; terminate window compare; takes effect at period start
Unspecified	N	drive high or low (application can choose)	S, s	S = compare H/L/T at period start; s = start window compare h/l/t at period start; actual state unknown; also used for block read
Unknown	?	input/output and level are unknown	?	input/output and level are unknown

## Engineering Units

Pre	Descr	Mult
E	exa	10 <sup>18</sup>
P	peta	10 <sup>15</sup>
T	tera	10 <sup>12</sup>
G	giga	10 <sup>9</sup>
M	mega	10 <sup>6</sup>
k	kilo	10 <sup>3</sup>
m	milli	10 <sup>-3</sup>
u	micro	10 <sup>-6</sup>
n	nano	10 <sup>-9</sup>
p	pico	10 <sup>-12</sup>
f	femto	10 <sup>-15</sup>
a	atto	10 <sup>-18</sup>

Unit	Description
A	Amperes
Cel	Degrees Celsius
F	Farads
H	Henrys (inductance)
Hz	Hertz
m	Meter
Ohm	Ohms
s	Seconds
W	Watts
V	Volts

## Events

	Drive
D	ForceDown
U	ForceUp
Z	ForceOff
P	ForcePrior

	Compare
L	CompareLow
H	CompareHigh
X	CompareUnknown
x	
T	CompareOff
V	CompareValid
l	CompareLowWin- dow
h	CompareHighWin- dow
t	CompareOffWindow
v	CompareValidWin- dow
S	CompareSubstitute
s	CompareSubsti- tuteWindow

	Expect
R	ExpectLow
G	ExpectHigh
Q	ExpectOff
M	Marker

	Unresolved
N	ForceUnknown
A	LogicLow
B	LogicHigh
F	LogicZ
?	Unknown