STIL Language Test Vector Format (Simplified)

The Credence D10 uses the STIL (pronounced 'Style') language to define the test vectors applied to the device under test (DUT). The STIL files (also called Blocks, as the can be defined within the same physical file) describe the test pins, voltage specification, timing specification and test pattern:

- Signals Block
  - Defines the Pin Signals – NOTE: there can only be one definition for each pin.
- Timing Block and Waveform Table
  - Defines the signal timing sets and waveform formatting
- DCLevels Block
  - Defines the DC Signal Levels applied to and expected from the device under test (VIL/VIH, VOL/VOH).
- Pattern Block
  - Defines the Test Vectors, which are time-sliced on a fixed time-period (not event driven). The data within each vector then specifies the action on individual pins during that fixed time period.

These individual parameter sets are linked referenced by other STIL format blocks to define the complete set of parameters needed to execute a set of test vectors

- Signals Group Block
  - Groups together various pins, such as Busses and Control Signals
- PatternBurst Block
  - Links one or more Pattern Blocks into a sequence of patterns
- PatternExec Block
  - Ties together A Burst Block with the requisite DC levels and timing sets.

Test Vector Locations for IDD, VOL and VOH measurement.

In order to measure device pins in a given state (VOL/VOH/Tri-State) it is necessary to stop the test pattern at fixed locations and use the ATE instrumentation to measure the appropriate parameter. To facilitate this, locations in the test vectors where these measurements should be made will be marked by a program Label and an accompanying explanatory text comment. (See example at end of this document).

NOTE: For dynamic devices that require keep-alive clocks contact Micross Engineering.

Dean Cracknell

Oriel Court, Omega Park,
Alton, Hants, UK. GU34 2YT
Tel : +44 (0)1420 594180

Signals Block

Syntax

```
Signals {
            ( SigName ( In | Out | InOut | Supply | Pseudo ) ( ; |
            ( { ( ScanIn | ScanOut ) ;} ) ) )*
( WFCMap {
                        ( FROM_WFC -> TO_WFC( TO_WFC )+ ; )*
                        ( FROM_WFC1 FROM_WFC2 -> TO_WFC ; )*
                        } )
            )*)*
}
```

SigName = String representing the signal name
In = Defines the signal as input
Out = Defines the signal as output
InOut = Defines the signal as bidirectional
Supply = Defines power or ground signals
Pseudo = Used when the signal is not a device pin
ScanIn = Defines the signal as scan input
ScanOut = Defines the signal as scan output

Example

```
Signals {
          "pc4" InOut;
          irq InOut;
          scan0 In { ScanIn; }
}
```

SignalGroups Block

The **SignalGroups** block is used to create named references to zero or more signals groups. A subset of the SignalGroups block is supported in Diamond Series as shown in the following example.

Syntax

```
SignalGroups {
 ( GroupName = '( SigName | GroupName ) ( + ( SigName | GroupName ) )*' ; |
                        ( WFCMap {
                        ( FROM_WFC -> TO_WFC( TO_WFC )+ ; )*
                        ( FROM_WFC1 FROM_WFC2 -> TO_WFC ; )*
                        } )
            )*)*
}
```

`GroupName = String representing the signal group name.`
`SigName = String representing the signal name.`
`WFCMap = Block allows mapping WaveformCharacter values to other WaveformCharacters.`
`FROM_WFC = WaveformCharacter to be mapped to another WaveformCharacter.`
`FROM_WFC1 = First WaveformCharacter to be mapped to another WaveformCharacter.`
`FROM_WFC2 = Second WaveformCharacter to be mapped to another WaveformCharacter.`
`TO_WFC = WaveformCharacter or list of WaveformCharacters to use as the substitute WavefromCharacter.`

Example

```
SignalGroups {
          "porta" = ` "pa7" + "pa6" + "pa5" + "pa4" + "pa3" + "pa2" + "pa1"
          + "pa0" ` ;
          "portb" = ` "pb7" + "pb6" + "pb5" + "pb4" + "pb3" + "pb2" + "pb1"
          + "pb0" ` ;
          all = `porta + portb';
          A = `pa0+pa1+pa2+pa3' {
                    WFCMap {
                              z->x; //single-WFC mapping
                              01->x;//two-WFC mapping (requires presence of \j)
                    } // end WFCMap
          } // end A
}
```

Timing Block and WaveformTable Blocks

The **Timing** block defines the placement of the timing edges and the format of the cyclised waveforms that is referenced by applying waveform characters to signals in the Vector statements. Each vector occurs on a fixed periodic timeframe, and the placement of rising and falling edges within that time period can be defined on a pin-by-pin or pin-group basis.

A Timing block can contain one or more **WaveformTable** blocks (Wfts), each of which comprises a **Period** and a collection of **Waveforms**. Each waveform consists of a set of events, each with its edge placement.

Syntax

```
Timing TimingName {
        ( WaveformTable TableName {
                        Period TimeExpr ;
                        Waveforms {
                                (SigName {
                                        (WFC)* {
                                                ( TimeExpr (Event ( | Event)* ); )*
                                        } )*
                                } )*
                        }
        }
}
```

```
TimingName = String representing the name of the Timing block.
TableName = String representing the name of the WaveformTable block.
Period = Defines the cycle period.
SigName = String representing the signal name.
WFC = Defines the waveform character to use for pattern data. A waveform
character must be a single alphanumeric character, from the set of
characters:[0-9], [a-z], and [A-Z].
TimeExpr = ' ( integer | float ) ( engineering_prefix ) ( unit )  '
Event =
```

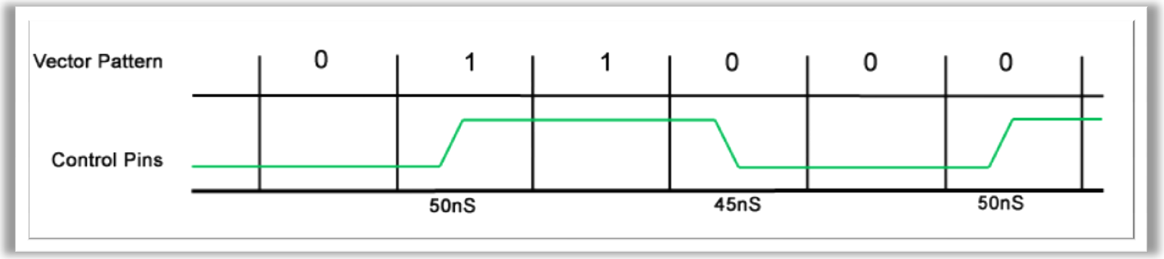
- `D | ForceDown – Forces the VIL value`
- `U | ForceUp – Forces the VIH value`
- `Z | ForceOff – Turns the driver off when the driver mode is HiZ. Forces VIHH when the driver mode is VIHH. Forces the Pin PMU voltage when the Pin PMU is connectByPattern().`
- `N | ForceUnknown – Same as the U format`
- `L | CompareLow – Compares for a value less than VOL`
- `H | CompareHigh – Compares for a value greater than VOH`
- `X | CompareUnknown – Don't care`
- `T | CompareOff – Compares for a value between VOL and VOH`

Example

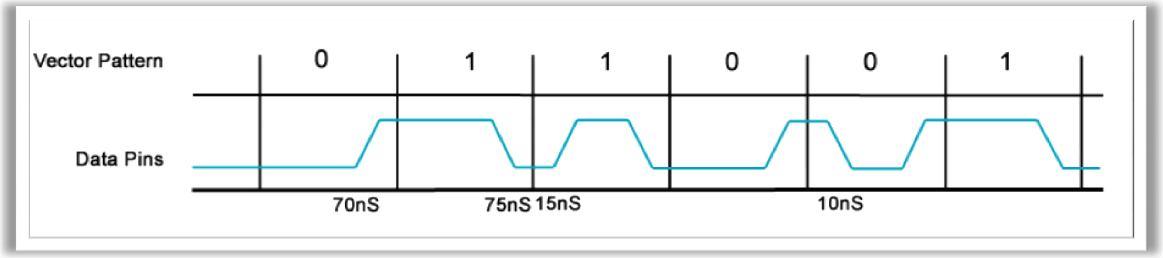
```
Timing "Param_Time_Set" {
        WaveformTable "T1" {
                Period '100nS';
                Waveforms {
                        // functional continuity timing...
                        "Control_pins" { 0 { '45nS' D; }}
                        "Control_pins" { 1 { '50nS' U; }}
                        "Data_pins" { 0 { '10ns' D; '70nS' U; }}
                        "Data_Pins" { 1 { '15nS' U; '75nS' D; }}
                        "Clock_pins" { 1 { '15ns' U; '75nS' D; }}
                        "Clock_pins" { 0 { '15ns' D; }}
                }
        }
}
```

Would give rise to the following waveforms:

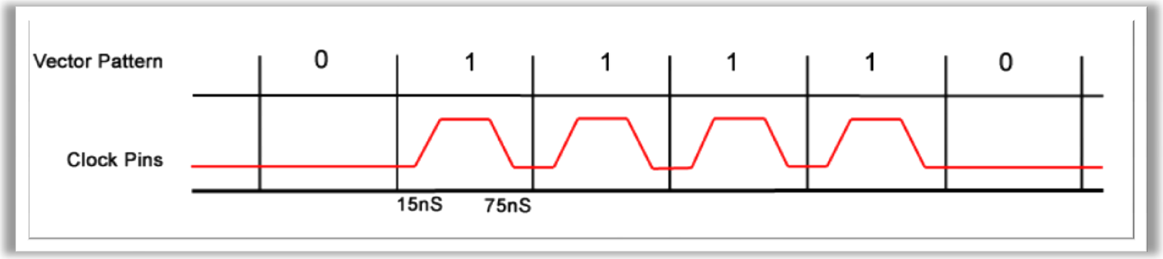
1: Pin group “Control_pins” will go high at 50nS when the vector pattern is a “1” for each of the pins in the group, if the previous vector was already high then no edge occurs. The pin will go low at 45nS when there is a “0” in the pattern for that pin, if the previous vector was already low then no edge occurs. This time format is known as Non-return To Zero (NRZ) as the pin will remain in the specified state until the next transition.



2: Pin group “Data_pins” will change state within each vector. This is a variation of Surround by Compliment or XOR format and is useful for Address and Data buses to allow set-up and hold timings.



3: Pin group “Clock_pins” will produce pulses when the pattern is “1” and will remain low when the pattern is “0” – this format is known as Return To Zero (RZ).



DCLevels Block

NOTE: Micross will create this Block from customer supplied information

The **DCLevels** block defines the DC levels to be applied to signals for each test pattern.

The DCLevels block names the DC parameters and defines specific DC data for a set of signals. Each statement in the DCLevels block specifies the characteristics for one DC parameter. Each DC parameter may be specified only once for a specific signal, in a single DCLevels block. Signals in the same block have to be unique; no overlap signal allowed. APIs can set the DCLevels by referencing the signals in the DCLevels block.

Syntax

```
(DCLevels (DCLevelsName) {   // DCLevels block
        (SignalName {
                (VIH (dc_expr)+;)
                (VIL (dc_expr)+;)
                (VIHH (dc_expr)+;)
                (VOH (dc_expr)+;)
                (VOL (dc_expr)+;)
        })*
})*
```

DCLevelsName = String representing the name of the DCLevels block.
SignalName = String representing the signal or signal group name.
VIH = Specifies the drive high voltage.
VIL = Specifies the drive low voltage.
VIHH = Specifies the alternate drive voltage.
VOH = Specifies the compare high voltage.
VOL = Specifies the compare low voltage.
dc_expr = ' (integer | float) (engineering_prefix) (unit) '

Example

```
DCLevels dc_func {
        ins {
                VIH 'vih1';
                VIL 'vil1';
        }
}
```

Pattern Block

The **Pattern** block defines the pattern vector data.

Syntax

Pattern *PatName* **{**

(*WaveformTableDeclaration* | *FunctionalVector*)*

}

PatName=ident

Wave*formTableDeclaration* = (W | WaveformTable) *ident*

FunctionalVector = (*ParallelVector* | *LoopVector* | *MatchVector* | *Macro* | *Procedure*)

Vector = (F | V) { *SigName* **=** (*WFC*)+ **;** }

Condition = C { *SigName* **=** (*WFC*)+ **;** }

WFC = (letter | digit | # | **%***)*

}

Example

```
Pattern example {
          W _default_WFT_;
          LABEL:    V { ALL = 00000000 00000000 00000000 00000000; }
                    V { ALL = 11111111 11111111 11111111 11111111; }
          Loop 10 {           // implemented as a multi-vector loop
                    V { ALL = 11111111 11111111 11111111 11111111; }
                    V { ALL = HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH; }
                    V { ALL = 00000000 00000000 00000000 00000000; }
                    V { ALL = LLLLLLLL LLLLLLLL LLLLLLLL LLLLLLLL; }
          }
          Loop 5 {                              // implemented as a repeat vector
                    V { ALL = 11111111 11111111 11111111 11111111; }
          }
          Stop;
}
```

PatternBurst Block

NOTE: Micross will create this Block from customer supplied information

The **PatternBurst** block defines the sequences of patterns to be executed during a test.

Syntax

PatternBurst *BurstName* **{**
PatList
(*PatName* **;**)*
}
BurstName = String representing the pattern burst name;
PatName = String representing the name of a Pattern block;

Example

```
PatternBurst "_burst_" {
    PatList {
        "first_pattern";
    }
}
```

PatternExec Block

NOTE: Micross will create this Block from customer supplied information

The **PatternExec** block is the "glue" that defines all of the pieces needed in order to execute patterns on a tester. It defines,

- the Category names to be used to resolve spec variables,
- the selector names to indicate which value (Min, Typ, Max, or Meas) of the spec variables to apply,
- the DCLevels block to set before executing the PatternExec,
- the APGSteering block to use when executing the PatternExec,
- the Timing block under which to find the WaveformTable references,
- and the PatternBurst to use.

If the Timing or DCLevels block being referenced contains spec variables that have multiple categories, then one or more Category statements shall be specified in the PatternExec block. If the Timing or DCLevels block references spec variables containing multiple values (i.e., Min, Typ, or Max values), the variables shall be specified in an unambiguous manner, either by resolving which value to apply via a Selector block, or by qualifying the variable name in the reference (for example, `var.Min'). The named Timing block shall resolve all WaveformTable names that are referenced in all Pattern blocks that are referenced.

Each item in the PatternExec block is optional. It can be used for binding the Spec Category and Selector with Timing and DC Levels. If the pattern burst is included, it will run each pattern in the pattern burst.

Syntax

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

CATEGORY_NAME = Selects the Category block name. The PatternExec block can have multiple categories, but the variables in each category must be unique.
SELECTOR_NAME = Selects the Selector block name. The PatternExec block can have multiple selectors, but the variables in each selector must be unique.
APG_STEERING_NAME = Selects the APGSteering block name that assigns the DUT pins to APG resources.
DC_LEVELS_NAME = Selects the DCLevels block. The PatternExec block can contain zero or one DCLevels Clocks.
TIMING_NAME = Selects the Timing block. The PatternExec block can contain zero or one Timing blocks.
PAT_BURST_NAME = Selects the PatternBurst block to execute. The PatternExec block can contain zero or one PatternBurst blocks.

Example

```
PatternExec "FuncExec" {
          DCLevels "DCLevels";
          Timing "Timing";
          PatternBurst "FuncBurst";
}
```

A Simple STIL Example

STIL 1.0;

```
Signals {
          DIR In;
          OE_ In;
          A0 In; A1 In; A2 In; A3 In;
          A4 In; A5 In; A6 In; A7 In;
          B0 Out; B1 Out; B2 Out; B3 Out;
          B4 Out; B5 Out; B6 Out; B7 Out;
}
SignalGroups {
          ABUS='A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0';
          BBUS='B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0';
          ALL ='DIR + OE_ + ABUS + BBUS';
}
Timing "basic_timing" {
          WaveformTable "one" {
                    Period '500ns';
                    Waveforms {
                              DIR { 01 { '0ns' D/U; }}
                              OE_ { 01 { '0ns' U; '200ns' D/U; '300ns' U; }}
                              ABUS { 01 { '10ns' D/U; }}
                              BBUS { HLZ { '0ns' Z; '0ns' X; '260ns' H/L/T; '280ns' X;}}
                              }
                    } // end WaveformTable one
} // end Timing "basic_timing"

PatternBurst "pat1_burst" {
          PatList { "pattern_1";
} // end PatternBurst "pat1_burst"

PatternExec {
          Timing "basic_timing";
          PatternBurst "pat1_burst";
} //end PatternExec

Pattern " pattern_1" {
          W "one";
LAB000:   V { ALL=0000000000LLLLLLLL; } // Measure all VOL here
LAB001:   V { ALL=0010000000HLLLLLLL; } // Measure B7 VOH here
          V { ALL=0001000000LHLLLLLL; }
          V { ALL=0000100000LLHLLLLL; }
          V { ALL=0000010000LLLHLLLL; }
          V { ALL=0000001000LLLLHLLL; }
          V { ALL=0000000100LLLLLHLL; }
          V { ALL=0000000010LLLLLLHL; }
LAB008:   V { ALL=0000000001LLLLLLLH; } // Measure B0 VOH here
Stop;
} // end Pattern " pattern_1"
```