



# Velocity<sup>®</sup> Program Generator

For Simulation to ATE and  
ATE to ATE Conversion

Release 522

# Usage Guide

---

# Velocity Program Generator

## Usage Guide

---

### **Copyright Notice**

Copyright © 2007 Alliance ATE Consulting Group

All rights reserved

Documentation version 1.0

Any technical documentation that is made available by Alliance ATE Consulting Group is the copyrighted work of Alliance ATE Consulting Group and is owned by Alliance ATE Consulting Group.

**NO WARRANTY.** The technical documentation is being delivered to you AS-IS and Alliance ATE Consulting Group makes no warranty as to its accuracy or use. Any use of the technical documentation or the information contained therein is at the risk of the user. Documentation may contain technical or other inaccuracies or typographical errors. Alliance ATE Consulting Group reserves the right to make change without prior notice.

No part of this publication may be copied without the express written permission of Alliance ATE Consulting Group, 1095 E. Duane Ave. Suite 100, Sunnyvale, CA 94085.

### **Trademarks**

Velocity Program Generator, D10Shell, and ShellConstructor are trademarks of Alliance ATE Consulting Group. Diamond, D10, and ITE are trademarks of Credence Systems Corporation.

**Contents**

**COPYRIGHT NOTICE.....I**

**TRADEMARKS .....I**

**OVERVIEW..... 3**

**VELOCITY ARCHITECTURE ..... 4**

**COMMAND LINE VERSION USAGE ..... 5**

**COMMAND LINE OPTIONS ..... 6**

**USAGE FROM THE SHELLCONSTRUCTOR GUI ..... 12**

**DIAMOND OUTPUT FILES..... 14**

**STIL FILES: ..... 14**

**SOURCE AND HEADER FILES:..... 14**

**GENERIC TEST FUNCTIONS ..... 15**

**DIAMOND COMMAND LINE FEATURES ..... 17**

### *Overview*

The purpose of this document is to describe the purpose and usage of the Velocity Toolkit. This documentation is included as part of the release of the tool kit and includes detailed descriptions of the following subjects

- The architecture of the tool as it integrates with other existing Credence or 3<sup>rd</sup> Party Diamond tools
- General Usage definition
- The layout of contents of the Velocity Configuration tool
  - Pointers and Environment Variables
  - Pin map
  - Test definitions
  - Flow setup
  - Custom pattern manipulation features
  - Timing and levels setup
- Generic Test functions and external libraries
- Command line execution
- GUI Usage
- Automation and reuse of configurations.
- Customization of the tool
- Examples
- Debugger Commands and Usage

## ***Velocity Architecture***

Velocity is a tool that is designed to facilitate quick creation of compilable, ready-to-use test programs for the Sapphire D10 digital test system. The generated programs will have a minimum set of features that will provide the user with quicker access to methods for executing pre-defined test sequences, copying reusable source files, and packaging local versions of STIL files as part of the test program so that these files can be manipulated without fear of destroying information that exists in the production or database versions of these files.

Tester files and program source files will be created and each pattern will be created as stand alone STIL representation so that incremental compile can be used for multiple pattern test programs

## Velocity Reference

### Command Line Version Usage

There are multiple command line translation paths available. Each contains a similar set of options, but some have options that are specific to the platform being extracted. For example WGL and EVCD have no concept of time sets and spec sets. Therefore these programs do not have options to set these. By default, the usage is:

**velocity -translationTypeSwitch [options] configFile [sourceFile [sourceFile2 sourceFile3 ...]]**

where,

*options* are listed and described in detail in the Command Line Options section below

*translationTypeSwitch* is the directive that tells velocity which translation engine to use. The allowable values for this are as follows in the Translation Types section

*configFile* is the velocity configuration file. See the Velocity Configuration Guide for a detailed description of the contents and usage of this file

*sourceFile* is a list of source files that will depend on the value of the translationTypeSwitch. The section Source Files will completely define the requirements and limitations of this field of arguments

### Translation Types

- **-D10Shell,-d10shell**: Creates Credence D10 program with no patterns. Requires the VCD and Verigy licenses
- **-WGLtoD10,-wgltoavc**: Translates WGL to Credence D10. Requires the WGL and Credence licenses
- **-VCTtoD10,-vcttod10**: Translates Verilog VCD/EVCD to Credence D10. Requires the VCT and Credence licenses
- **-VCDtoD10,-vcdtod10**: Translates VCD to Credence D10. Requires the VCD and Credence licenses
- **-J750toD10,-j750tod10**: Translates Terdyne J750 to Credence D10. Requires the J750 and Credence licenses
- **-FLEXtoD10,-flextod10**: Translates Terdyne UltraFlex to Credence D10. Requires the Flex and Credence licenses
- **-J973toD10,-j973tod10**: Translates J973 to Credence D10. Requires the J973 and Credence licenses
- **-AVCtoD10,-avctod10**: Translates AVC to Credence D10. Requires the AVC and Credence licenses

- **-STILtoD10,-stiltod10**: Translates STIL to Credence D10. Requires the Credence licenses

## *Command Line Options*

A set of command line directives is available that will allow for various optimizations to the output. These options are defined below

### **Universally Available options**

#### **+/-o**

Turns on and off optimization of output. Specifically, the timing and levels blocks that are included in the source files but are not used by the listed patterns will be removed from the output program. This facilitates smaller test programs that will compile and load much faster

#### **+/-s**

Turns on and off the output of new STIL files. If **-s** is used, the previously created output files will be reloaded and left untouched in the current Shell creation. Therefore, these STIL files will need no recompilation.. **+s** will ensure that the STIL files are recreated

#### **+/-t**

Turns on and off the output of all tester files for the D10 program. This includes all non-STIL and non-test program files (\*.cpp \*.h). **-t** will mask the generation of these files. **+t** will ensure recreation or new JOB, SIG, MAKE, etc.

#### **+/-p**

Turns on and off the generation of D10 test program files. **-p** will prevent existing \*.h and \*.cpp files from being overwritten. **+p** will create new source files.

#### **+/-c**

Turns on and off the automatic binary compilation of the target program.

#### **+/-n**

Turns on and off auto normalization of timing. When used a spec set will be used that will allow global and waveform table specific scaling of timing values. This is an easier method of providing period scaled timing than the Custom Timing blocks

#### **+a**

Turns on “append mode”. When in use, this feature will automatically add new patterns, timing, etc. to the existing program without overwriting and/or removing existing data.

#### **+q**

Turns on “quiet mode”. When in use, this feature will automatically block the progress meter GUI so that the transitions can be run remotely without needing to export a display

## Velocity Reference

### **+h**

Prints out the complete usage man page to screen

### **J750 and J973 specific options**

#### **+ac=/+dc=**

Allows a global override for the spec categories that are used for the AC and DC tests respectively. Useful if a specific timing is need for all tests regardless of what is used in the source program

#### **+ts=/+ls=**

Allows manual override to redefine globally the timing or leves set/sheet that is used for all tests regardless of what is predefined by the source program

### **VCD/EVCD Specific options**

#### **+xN**

Define the maximum number of databits to be defined in a single data cycle. This will automatically adjust the waveform tables to allow more than one transition with in a single cycle. N=1 by default is option is not used

#### **+eN**

Adjust the resolution for edge snapping. Drive edges will snap forward, Receive edges will snap back. N will determine how many regions are used. By default, N=1, which means that there will be a single Drive edge and a single receive edge. N=4 will divide the period into 4 equal regions. Edges will snap into based on the region within the period the raw edge falls into.

### **WGL Specific Options**

#### **+/-m**

Enable or Disable the usage of scan macros for representation of scan instances. If enabled, STIL syntax which uses macros will be implemented so that scan instances are defined as serial scan. If didabled, the scan information will be converted to standard parallel vectors.

## Source Files

Source files listed on the command line will depend on the value of the translationTypeSwitch.

Translation Type	File List	Description
D10Shell		No source files will be used. Only the CFG is required
STILtoD10	*.STIL and *.stil files	
AVCtoD10	*.dvc files to define the timing; *.avc to define the patterns	
J750toD10	*.xml files to define program, levels, and timing; *.atp files to define the patterns	
J973toD10	*.pinadr to define pin lists; *.waveadr to define waveform tables; *.specadr files to define timing spec values; *.lvmaadr files to define patterns; optional *.svmaadr to define subroutines; optional *.ts to redefine timset to waveform table mappings	
WGLtoD10	*.wgl files to define pins, specs, timing, and patterns	
VCDtoD10	*.evcd files to define timestamp based events	

If used properly, each program will create the D10 program files in the directory defined in the configuration. The following figure shows a successful usage of the velocity command line program.

```

Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
[field@change-hostname moca]$ pwd
/mnt/hgfs/Data/CustomApps/Freescale/WGL/moca
[field@change-hostname moca]$ ls
bist_pll.log  bist_pll.wgl  moca.cfg
[field@change-hostname moca]$ velocity -WGLtoD10 moca.cfg bist_pll.wgl

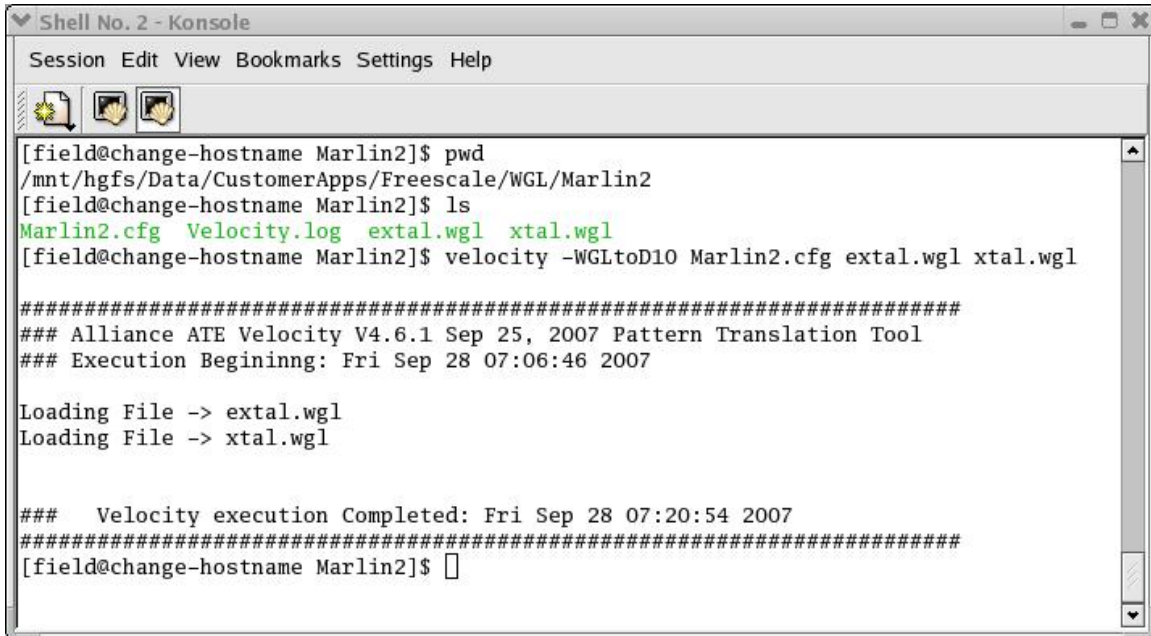
#####
### Alliance ATE Velocity V4.6.1 Sep 25, 2007 Pattern Translation Tool
### Execution Beginning: Fri Sep 28 06:52:13 2007

Loading File -> bist_pll.wgl

### Velocity execution Completed: Fri Sep 28 06:53:08 2007
#####
[field@change-hostname moca]$
    
```

## Velocity Reference

The previous example translates a program that uses only a single pattern. You can include multiple patterns by simply including all relevant patterns on the command line. The next example shows the results when multiple valid patterns are included in a single translation.



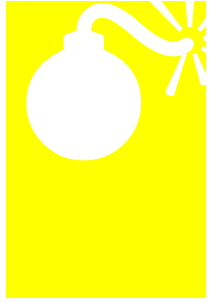
```
Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
[field@change-hostname Marlin2]$ pwd
/mnt/hgfs/Data/CustomApps/Freescale/WGL/Marlin2
[field@change-hostname Marlin2]$ ls
Marlin2.cfg Velocity.log extal.wgl xtal.wgl
[field@change-hostname Marlin2]$ velocity -WGLtoD10 Marlin2.cfg extal.wgl xtal.wgl

#####
### Alliance ATE Velocity V4.6.1 Sep 25, 2007 Pattern Translation Tool
### Execution Begining: Fri Sep 28 07:06:46 2007

Loading File -> extal.wgl
Loading File -> xtal.wgl

### Velocity execution Completed: Fri Sep 28 07:20:54 2007
#####
[field@change-hostname Marlin2]$
```

## Velocity Reference



There are a number of usage examples that will result in a variety of errors. The following are examples of common errors

- Non-Existent or misspelled configuration file name
- Non-existent or misspelled source files
- Missing references caused by include statements in source files that can't be resolved.
- Having no write privileges to the directories defined by configuration file

## Velocity Reference



Once completed successfully, the following files and directories will reside in the target directory

### **Job File:**

The job file is the master file that packages everything for the diamond software. This file will allow ITE to determine which files need to get loaded and which files need to be compiled when using the Diamond BuildTool.

### **Make file:**

This file is used to compile both the source and STIL files. This file instructs Linux which compiler is to be used, which options should be used, and where output files should go.

### **Sig File:**

This file defines the pin and group names that are used for a given program and attaches the pins to the proper resource and channel in hardware.

### **Res File:**

This file determines which resources are used. It is needed so that the calibration information can be attached properly and so that the proper libraries for each slot are used.

### **Binmap and BinDef Files:**

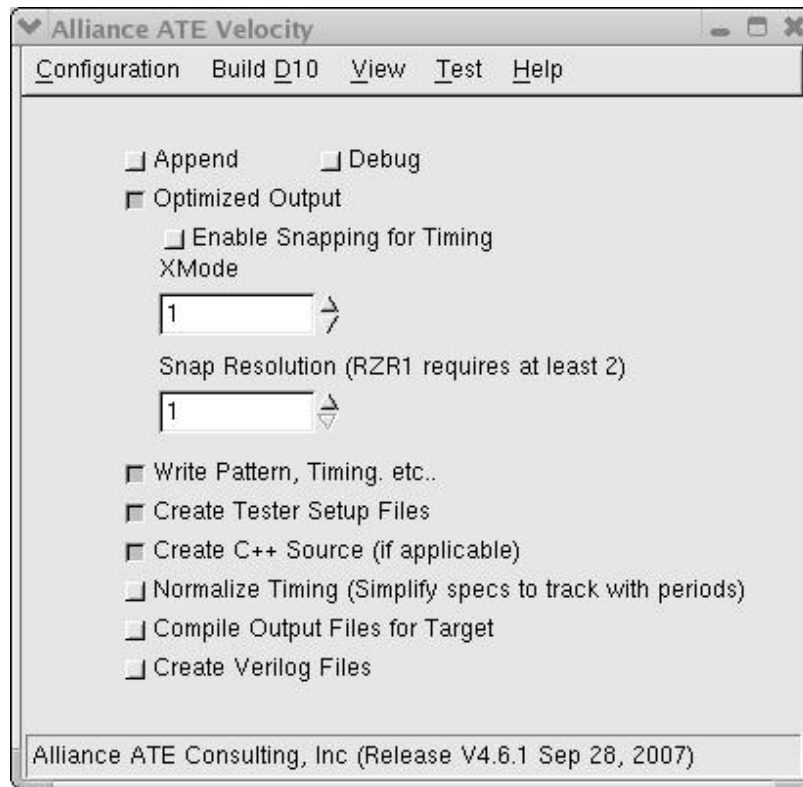
These files are automatically generated so that graceful exits for various types of failures can be defined explicitly. By default, PASS, FUNCTIONAL, DC, and EXCEPTION are the only bins defined. For engineering use, there will be little need to ever touch these files once they have been created.

### **STIL pattern directories:**

Each pattern translated will result in its own sub directory containing the STIL representation of the source file. Each pattern is converted into its own directory so that 1) timing can be setup using different formats from pattern to pattern. The D10 can't have the same pattern defined with different formats. Splitting into different directories prevents this error from occurring. And 2) so that compilation can be done incrementally. Edits to one pattern won't require ALL pattern to be recompiled, only the pattern that is touched.

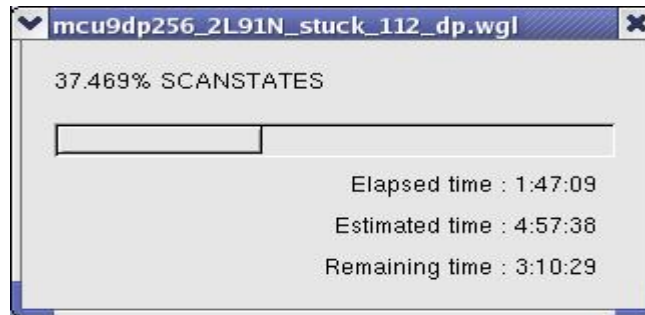
## Usage from the Velocity GUI

The GUI simplifies some aspects of usage for those that do not require the scripted control of the command line programs. All of the options available to the command line are available from the GUI. In addition, some other features are available to speed up the program creation process.



1. The ability to generate new configuration files from any of the defined input sources is available. These CFG files can also be edited from within the GUI as well.
2. Output files can be viewed easily from the tool to allow simple edits to STIL, source or tester setup files
3. ITE can be started from the tool and the default directory will automatically be set to match the target location of the active program.
4. A link is made available to this documentation

## Velocity Reference



5. The progress indicator displays in terms of percentage complete to give the user a more accurate idea of the time remaining in a given translation



**For a step by step recipe on using the GUI, please refer to the Velocity Quick Start manual.**

## **Diamond Output Files**

### ***STIL files:***

Files used as input will be reformatted and reorganized so that pin, timing, level, and pattern information is well commented and easily organized. In addition, requested modifications may result in multiple patterns that are created from a common base pattern. These files will all be created and copied into the local directory so that they can be edited, modified, and used without fear of harming the production database .

For a more detailed definition of STIL, documentation can be obtained from <http://standards.ieee.org/> (search for STIL)

STIL files will generally be ported exactly as they have been defined by the source STIL files. But, some minor modifications might be requested that will require loops, repeats, or other sequencing changes. These will be directed by the contents of the Pattern Blocks of the configuration file. This optional section of the configuration is discussed later in this document.

### ***SOURCE and HEADER Files:***

Source and header files will be created that will define the test flow that is used, create stand alone test functions that can be accessed through the command line, and power up and power down sequences that can also be used as part of the flow and/or from the command line interface. These source and header files can be used with the SlickEdit debugger to provide another access point for device level debug.

### **ShellExample.cpp**

The “main” program will be directly defined by the contents of the Flow block of the configuration which is used to describe later in this document. If no flow is defined, the main program will still be created but it will have no function calls in it.

### **TestFunctions.cpp and TestFunctions.h**

The test functions are accessible from both the main program above as well as the command line scripts are defined and implemented in these files. They make use of the GenericFunctions object which is accessed through the included user\_commands shared library.

### ***Generic Test Functions***

The generic test functions are referenced through inclusion of an external library. The contents of this file are required by the Shell so that the resulting D10 test program will have proper function definitions for the various digital tests that might be referred to within the Test Block section of the configuration. The names and argument lists of these functions should not be changed as this will break the link between the construction process and the end use of the resulting test code. In particular, the names of the generic functions are mapped to the test blocks by the “type” keyword and described in the Test Block Definitions section above. Arguments of the generic functions are all mapped to specific keywords. The function and argument mappings for the included generic functions are defined below:

- **func** = **GenericFunction::Functional**(*testName, testNumber, patternName*)
- **cont** = **GenericFunction::Continuity**(*testName, int testNumber, pinOrGroup, ForceCurrentValue, hiLimit, lowLimit*)
- **shrt** = **GenericFunction::Shorts**(*testName, testNumber, pinOrGroup, ForceVoltageValue, hiLimit, lowLimit*)
- **leak** = **GenericFunction::Leakage**(*testName, testNumber, pinOrGroup, powerLevel, hiLimit, lowLimit*)
- **iddq** = **GenericFunction::StaticIDD**(*testName, testNumber, patternName, stopPoints, supplyName, highLimit, lowLimit*)

Customization is permitted within the body of these functions assuming that valid code results and that this code does not violate and compilation rules. If customized versions of test functions is desired. It is recommended that the files be copied to a new location and manipulated there rather than modifying the contents of the versions of this file that are installed by default.

## Velocity Reference

### General Test Functions

The following are default definitions for the various AC and DC functions that are available for use with the installed version of the Diamond User Procedures. These are implemented using the Diamond API's as described in the online documentation.

```

/*****
PowerUp/PowerDown Functions
*****/
static void PowerUp(std::string boardType, std::string supplyName,
                    double forceVoltage, double currentClamp, double delay);
static void PowerDown(std::string boardType, std::string supplyName);
static void DisconnectPower(std::string boardType, std::string supplyName);
static void SetVoltage(const std::string &mySigGrp, double voltageVal,
                       double currentLim);
static void SetVoltageOff(const std::string &mySigGrp );
static void SetDCLevels(const std::string& SigGroup, double Vil, double Vih,
                       double Vol, double Voh );

/*****
Generic Tests
*****/
static void Functional(std::string testName, int testNumber, std::string
                       patternName, int runStandAlone=0);
static void Continuity(std::string testName, int testNumber,
                       std::string boardType, std::string pinOrGroup,
                       double ForceCurrentValue, double ClampValue,
                       double hiLimit, double lowLimit, int runStandAlone=0);
static void Leakage(std::string testName, int testNumber, std::string pinOrGroup,
                    double powerLevel, double hiLimit, double lowLimit);
static void StaticIDD(std::string testName, int testNumber, std::string patternName,
                      std::string stopPoints, std::string supplyName,
                      double highLimit, double lowLimit);

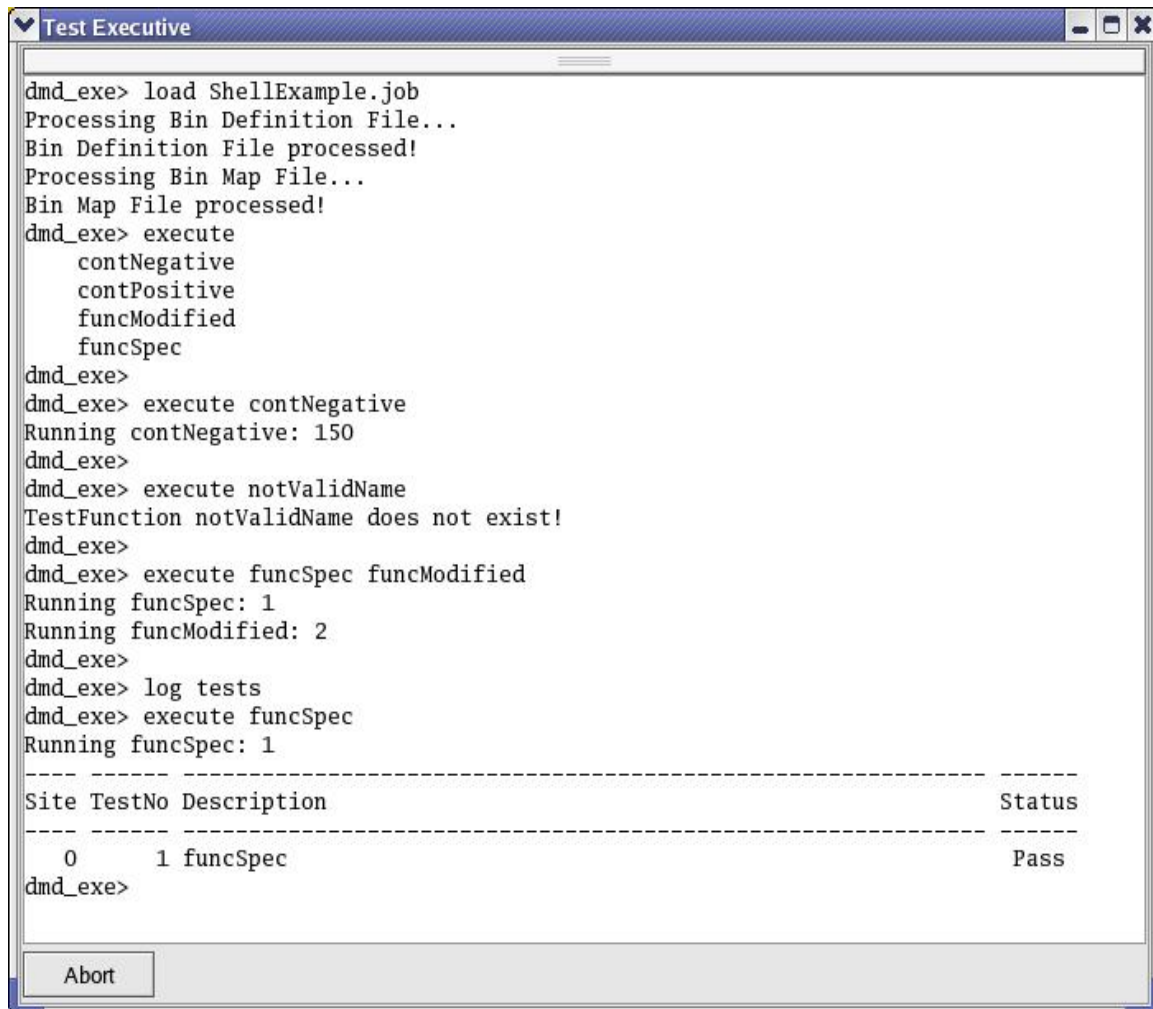
```

## *Diamond Command Line Features*

This section describes that template code that is installed that is used to automatically generate test scripts that can be executed from the Diamond Command Line tool. The code that is installed can be moved but it highly recommended that these files' contents not be tampered with as this may result in unspecified behavior. If needed additional features can be added and installed as part of custom patches

First refer to the Test Block and Power sequence blocks above. Each of these block names will result in a corresponding execution block. The code in user\_commands.cpp will automatically create and register a command line function named "execute". Velocity will add functionality to this script by allowing each power sequence and test block to be executed using this registered command.

The following figure shows a sample session using the scripts assuming 4 test blocks have been created



```

Test Executive
-----
dmd_exe> load ShellExample.job
Processing Bin Definition File...
Bin Definition File processed!
Processing Bin Map File...
Bin Map File processed!
dmd_exe> execute
    contNegative
    contPositive
    funcModified
    funcSpec
dmd_exe>
dmd_exe> execute contNegative
Running contNegative: 150
dmd_exe>
dmd_exe> execute notValidName
TestFunction notValidName does not exist!
dmd_exe>
dmd_exe> execute funcSpec funcModified
Running funcSpec: 1
Running funcModified: 2
dmd_exe>
dmd_exe> log tests
dmd_exe> execute funcSpec
Running funcSpec: 1
-----
Site TestNo Description                               Status
-----
    0      1 funcSpec                                   Pass
dmd_exe>
-----
Abort
  
```

## Velocity Reference

## Iddq

This function also measures the DUT power supply current, but unlike the *ivpow* routine, *iddq* varies the stop vector of the functional test, not the supply voltage. This routine is particularly helpful in finding failure modes in the pattern that cause the DUT to draw unusually high amounts of current and/or latch-ups. An example illustrating this situation will be shown.

The `iddq()` API is called for a pattern that contains multiple scan bursts. The stop vectors are chosen to correspond to the vectors immediately following each scan burst. The intent is to identify scan bursts that put the DUT in a high-power state.

```
iddq( "vdd",           // supply pins
      "ScanExec",     // pattern name
      "64,128,192,256,320,384,448", // stop vectors
      10e-6,          // delay
      "iddqPlot.txt" ); // output file name
```

Invoking the *iddq* routine interactively with the same parameters looks like the following. Note that you must first pause on the functional test that you will be running.

```
dmd_exe> pause fcall ←be sure to pause on the functional test before calling iddq
dmd_exe> run
-----
Site TestNo Description                               Status
-----
    0      1 ScanExec                                   Pass
Pause on test: 1 (ScanExec)

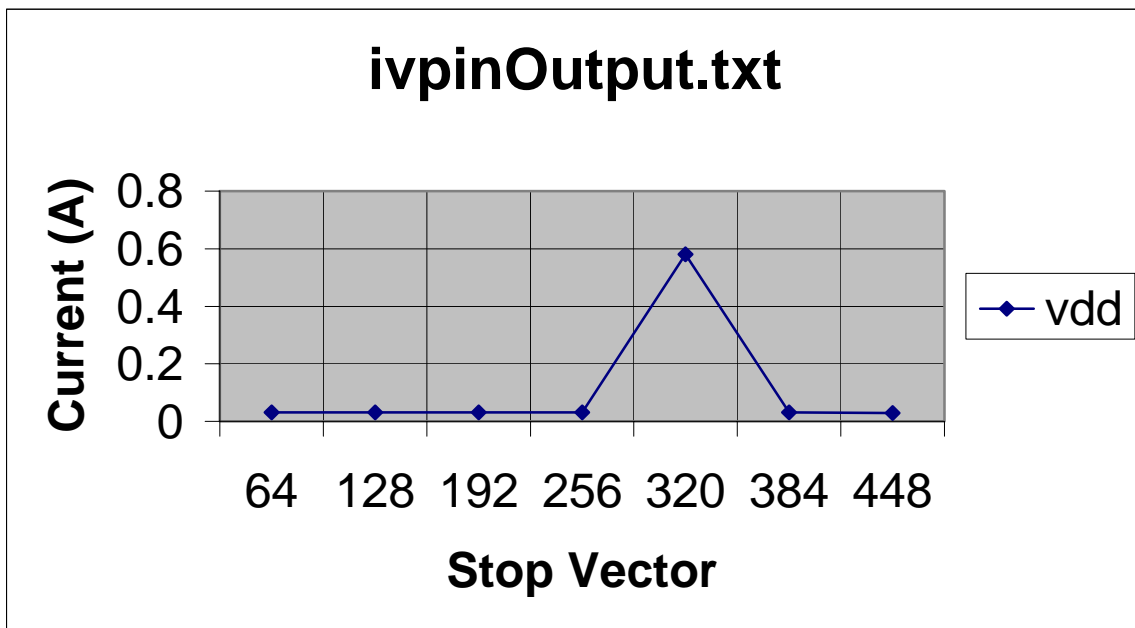
dmd_paused> iddq set all ←set up the iddq parameters
pins (allSupplies): vdd
stop (0) [vec,vec-vec]: 64,128,192,256,320,384,448 ←vectors after scan chains
delay ( 0.010mS):
filename (iddqOutput.txt):
verbose (off) [on/off/pause]:

dmd_paused> iddq ←run the iddq routine
Opening output file iddqOutput.txt.....
Done!
```

## Velocity Reference

Calling this function yields the following data and plot. The data indicates that one of the scan bursts does indeed drive the DUT into a high-power state (vector 320). Note that the plot axes for the *iddq* routine are stop vector (as opposed to force voltage in the other routines) and current.

Stop Vector	vdd
64	0.031784
128	0.031818
192	0.030851
256	0.031755
320	0.580799
384	0.030868
448	0.028956



## Debugger Command Syntax

### Ivpin Usage Statement

<code>ivpin</code>	- runs the <code>ivpin</code> routine
<code>ivpin set</code>	- shows current argument values
<code>ivpin help</code>	- shows the usage statement
<code>ivpin set all</code>	- prompts for all arguments
<code>ivpin set pins &lt;pin/group&gt;</code>	- specifies the pins to be tested
<code>ivpin set vmin &lt;value&gt;</code>	- sets low limit of the voltage sweep
<code>ivpin set vmax &lt;value&gt;</code>	- sets high limit of the voltage sweep
<code>ivpin set vreso &lt;value&gt;</code>	- sets step size of the voltage sweep
<code>ivpin set delay &lt;time&gt;</code>	- sets delay for the measurement
<code>ivpin set file &lt;name&gt;</code>	- specifies the output file name
<code>ivpin set verbose on/off/pause</code>	- enable/disable logging and pausing

Note: for single measurements, specify one signal pin and set `vmin` and `vmax` to the same value.

### Ivpin Parameter Descriptions

- *ivpin* This command will call the `ivpin()` API, using the current argument values. The current argument values are either the default values or the values used the last time `ivpin()` was called, whether from the test program or the command line debugger. When this command is called, it will automatically turn off DC tests datalogging if it was enabled, so that no datalogging will appear on the screen. Pausing on DC tests also will not occur inside the *ivpin* command. The *verbose* flag can be set to control pausing and logging inside the command.
- *ivpin set* This command shows the current settings of each of the arguments. The following script shows the default settings.
 

```
dmd_paused> ivpin set
pins = allpins
vmin = -1.000 V
vmax = 1.000 V
vreso = 0.100 V
delay = 0.000 S
filename = ivpinOutput.txt
verbose = off
```
- *ivpin help* This command displays the usage statement as seen in the previous section of this document.
- *ivpin set all* This command provides a simple way to change one or more arguments. It prompts for each argument, showing the current value in parenthesis. Entering a carriage return at the prompt will retain the current value, or the user can enter a new value. An error message will be displayed for incorrect values and the argument will retain its initial value. Floating point, exponential or scientific notation can be used for numeric values.
- *ivpin set pins <pin/group>* This command is used to specify the pins that will be measured by the *ivpin* command. Only one pin or pingroup name can be entered.

## Velocity Reference

That is, a list of multiple pins and pingroups will cause an error. The specified pin(s) must be digital pins of type In, Out or InOut.

- *ivpin set vmin <value>* This command sets the lower limit of the voltage sweep. The force voltage will be incremented in *vreso* steps from *vmin* to *vmax*, while measuring and logging the current at each step. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The valid range is -1V to 6V.
- *ivpin set vmax <value>* This command sets the upper limit of the voltage sweep. The force voltage will be incremented in *vreso* steps from *vmin* to *vmax*, while measuring and logging the current at each step. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The valid range is -1V to 6V.
- *ivpin set vreso <value>* This command sets the resolution of the voltage sweep. The force voltage will be incremented in *vreso* steps from *vmin* to *vmax*, while measuring and logging the current at each step. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The value must be greater than 0V and less than or equal to 6V.
- *ivpin set delay <time>* This command sets the amount of delay to occur between applying the force voltage and measuring the current. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The value must be greater than 0s.
- *ivpin set file <name>* This command specifies the name of the file in which the data will be saved. The D10\_DATA environment variable will be concatenated with the file name specified by this command. For example, if D10\_DATA is equal to *./* and the file name is specified to be *out.txt*, then the file *./out.txt* will be created. If the environment variable doesn't exist, the file will be created in the current directory.
- *ivpin set verbose on/off/pause* This command enables additional debug features. When *verbose on* is set, the datalog for each measurement is displayed on the screen. Auto-ranging of the current ranges can be seen in this datalog, as well. When *verbose pause* is set, the datalog is enabled and the program pauses after each measurement is complete. That is, it will not pause at each step of the auto-ranging, but will pause after the auto-ranging is complete and the current has been successfully measured. It pauses for each pin on each voltage step. This is not the same pause state as the *dmd\_pause* debugger state, though. In the case if the *verbose pause*, the GUI tools will not be active and the command line debugger will not be active. The *verbose pause* is strictly for looking at the DUT state with a scope or meter. The command *verbose off* will turn off the logging and pausing.

## Ivpow Usage Statement

<code>ivpow</code>	- runs the <code>ivpow</code> routine
<code>ivpow set</code>	- shows current argument values
<code>ivpow help</code>	- shows the usage statement
<code>ivpow set all</code>	- prompts for all arguments
<code>ivpow set pins &lt;pin/group&gt;</code>	- sets the supply pins to be tested
<code>ivpow set vmin &lt;value&gt;</code>	- sets low limit of the voltage sweep
<code>ivpow set vmax &lt;value&gt;</code>	- sets high limit of the voltage sweep
<code>ivpow set vreso &lt;value&gt;</code>	- sets step size of the voltage sweep
<code>ivpow set delay &lt;time&gt;</code>	- sets delay for the measurement
<code>ivpow set file &lt;name&gt;</code>	- specifies the output file name
<code>ivpow set verbose on/off/pause</code>	- enable/disable logging and pausing

Note: for single measurements, specify one supply pin and set `vmin` and `vmax` to the same value.\n"

## Ivpow Parameter Descriptions

- *ivpow* This command will call the `ivpow()` API, using the current argument values. The current argument values are either the default values or the values used the last time `ivpow()` was called, whether from the test program or the command line debugger. When this command is called, it will automatically turn off DC tests datalogging if it was enabled, so that no datalogging will appear on the screen. Pausing on DC tests also will not occur inside the *ivpow* command. The *verbose* flag can be set to control pausing and logging inside the command.
- *ivpow set* This command shows the current settings of each of the arguments. The following script shows the default settings.
 

```
dmd_exe> ivpow set
pins = allSupplies
vmin = 0.000 V
vmax = 1.000 V
vreso = 0.100 V
delay = 0.010mS
filename = ivpowOutput.txt
verbose = off
```
- *ivpow help* This command displays the usage statement as seen in the previous section of this document.
- *ivpow set all* This command provides a simple way to change one or more arguments. It prompts for each argument, showing the current value in parenthesis. Entering a carriage return at the prompt will retain the current value, or the user can enter a new value. An error message will be displayed for incorrect values and the argument will retain its initial value. Floating point, exponential or scientific notation can be used for numeric values.
- *ivpow set pins <pin/group>* This command is used to specify the pins that will be measured by the *ivpow* command. Only one pin or pingroup name can be entered. That is, a list of multiple pins and pingroups will cause an error. The specified pin(s) must be pins of type Supply. An error message will occur if they are any other type.

## Velocity Reference

- *ivpow set vmin <value>* This command sets the lower limit of the voltage sweep. The supply voltage will be incremented in *vreso* steps from *vmin* to *vmax*, while measuring and logging the current at each step. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The valid range is 0V to 6V.
- *ivpow set vmax <value>* This command sets the upper limit of the voltage sweep. The supply voltage will be incremented in *vreso* steps from *vmin* to *vmax*, while measuring and logging the current at each step. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The valid range is 0V to 6V.
- *ivpow set vreso <value>* This command sets the resolution of the voltage sweep. The supply voltage will be incremented in *vreso* steps from *vmin* to *vmax*, while measuring and logging the current at each step. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The value must be greater than 0V and less than or equal to 6V.
- *ivpow set delay <time>* This command sets the amount of delay to occur between applying the force voltage and measuring the current. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The value must be greater than 0s.
- *ivpow set file <name>* This command specifies the name of the file in which the data will be saved. The D10\_DATA environment variable will be concatenated with the file name specified by this command. For example, if D10\_DATA is equal to *../* and the file name is specified to be *out.txt*, then the file *../out.txt* will be created. If the environment variable doesn't exist, the file will be created in the current directory.
- *ivpow set verbose on/off/pause* This command enables additional debug features. When *verbose on* is set, the datalog for each measurement is displayed on the screen. Auto-ranging of the current ranges can be seen in this datalog, as well. When *verbose pause* is set, the datalog is enabled and the program pauses after each measurement is complete. That is, it will not pause at each step of the auto-ranging, but will pause after the auto-ranging is complete and the current has been successfully measured. It pauses for each pin on each voltage step. This is not the same pause state as the *dmd\_pause* debugger state, though. In the case if the *verbose pause*, the GUI tools will not be active and the command line debugger will not be active. The *verbose pause* is strictly for looking at the DUT state with a scope or meter. The command *verbose off* will turn off the logging and pausing.

## Iddq Usage Statement

```

iddq                - runs the iddq routine
iddq set            - shows current argument values
iddq help          - shows the usage statement
iddq set all       - prompts for all arguments
iddq set pins <pin/group> - sets the supply pins to be tested
iddq set stop <stop vec string> - sets stop vectors for measurements
iddq set delay <time> - sets delay for the measurement
iddq set file <name> - specifies the output file name
iddq set verbose on/off/pause - enable/disable logging and/or pausing

```

## Iddq Parameter Descriptions

- *iddq* This command will call the `iddq()` API, using the current argument values. The current argument values are either the default values or the values used the last time `iddq()` was called, whether from the test program or the command line debugger. When this command is called, it will automatically turn off DC tests datalogging if it was enabled, so that no datalogging will appear on the screen. Pausing on DC tests also will not occur inside the *iddq* command. The *verbose* flag can be set to control pausing and logging inside the command.
- *iddq set* This command shows the current settings of each of the arguments. Note that the patternExec name is displayed by this command, however the user cannot change the patternExec through the *iddq set* commands. The patternExec is changed by calling the `iddq()` API directly or by pausing on the functional test API call for the desired patternExec. The following script shows the default settings.

```

dmd_exe> iddq set
pins = allSupplies
pattern =
stop = 0
delay = 0.010mS
filename = iddqOutput.txt
verbose = off

```

- *iddq help* This command displays the usage statement as seen in the previous section of this document.
- *iddq set all* This command provides a simple way to change one or more arguments. It prompts for each argument, showing the current value in parenthesis. Entering a carriage return at the prompt will retain the current value, or the user can enter a new value. An error message will be displayed for incorrect values and the argument will retain its initial value. Floating point, exponential or scientific notation can be used for numeric values.
- *iddq set pins <pin/group>* This command is used to specify the pins that will be measured by the *iddq* command. Only one pin or pingroup name can be entered. That is, a list of multiple pins and pingroups will cause an error. The specified pin(s) must be pins of type Supply. An error message will occur if they are any other type.
- *iddq set stop <stop vector string>* This command specifies the list of stop vectors on which measurements will be made. The syntax for the stop vector

## Velocity Reference

string is a comma-separated list of vector addresses. In addition to individual addresses, a range of addresses is also supported, for example *27-35* or *100-105*. Other examples of valid stop vector strings are *200,312-320,457* and *12,25,26*.

- *iddq set delay <time>* This command sets the amount of delay to occur between applying the force voltage and measuring the current. Floating point, exponential or scientific notation can be used for the numeric value. An error message will be given if the value is out of range, and the previous value will be retained. The value must be greater than 0s.
- *iddq set file <name>* This command specifies the name of the file in which the data will be saved. The D10\_DATA environment variable will be concatenated with the file name specified by this command. For example, if D10\_DATA is equal to *./* and the file name is specified to be *out.txt*, then the file *./out.txt* will be created. If the environment variable doesn't exist, the file will be created in the current directory.
- *iddq set verbose on/off/pause* This command enables additional debug features. When *verbose on* is set, the datalog for each measurement is displayed on the screen. Auto-ranging of the current ranges can be seen in this datalog, as well. When *verbose pause* is set, the datalog is enabled and the program pauses after each measurement is complete. That is, it will not pause at each step of the auto-ranging, but will pause after the auto-ranging is complete and the current has been successfully measured. It pauses for each pin on each stop vector in the list. This is not the same pause state as the *dmd\_pause* debugger state, though. In the case if the *verbose pause*, the GUI tools will not be active and the command line debugger will not be active. The *verbose pause* is strictly for looking at the DUT state with a scope or meter. The command *verbose off* will turn off the logging and pausing.