VECTOR

VectorCAST RSP GHS Integrity

Version 1.0

2024-05-24

Application Note AN-ACT-1-111

Author Craig Pedersen

Restrictions Public Document

Abstract Information on the VectorCAST RSP for the GreenHills Integrity OS

Table of Contents

1 Overview

A VectorCAST Runtime Support Package (RSP) is an extension of the VectorCAST product. It provides an interface layer that allows you to use the VectorCAST testing techniques and methods on an embedded target processor. The VectorCAST RSP will always run on your host platform (the same platform as the compiler). Only the VectorCAST generated test harness will be downloaded to, and executed on, the embedded target.

A VectorCAST RSP is always customized to a particular Target CPU, Cross Compiler, and Run-Time environment (or kernel). As such, the information presented here contains sections for this specific RSP.

This application note provides details on configuring VectorCAST to run tests on applications that use the GreenHills Integrity Operating System running on the Integrity Simulator or on a target board. The GreenHills compiler is used to build the VectorCAST test harness.

VectorCAST offers off-the-shelf support for certain variants of Integrity running on a PPC or ARM target. However, there are several configuration options that change the behavior of the RSP. The intent of this paper is to explain those options and how they affect the RSP. Some of those options are as follows:

- Integrity Build type (Kernel, Monolith, Dynamic)
- Integrity Version
- GreenHills Multi & Compiler version
- CPU (ARM, PPC, X86, etc.)
- I/O Type (e.g., Standard I/O vs. File I/O)
- Target Connection (e.g., Simulator vs. JTAG probe)

2 Setup

2.1 Environment Setup

Before launching VectorCAST for use with the GreenHills Integrity Operating System, 3 things need to be done:

1) Set the `VC_GHS_OS_DIR` environment variable to point to the location of Integrity
2) Set a path to the Multi Installation
3) Set a path to the Compiler installation

It is suggested to create a DOS batch file or Linux script to set this up and then launch VectorCAST from that script. For Windows it would look something like this, depending on the version of the tools:

```
set VECTORCAST_DIR=c:\vcast
set VC_GHS_OS_DIR=c:\GHS\int1144
set GHS_MultiPath=C:\GHS\multi_716d
set GHS_CompilerPath=C:\GHS\comp_201416b
set path=%path%;% GHS_MultiPath%;% GHS_CompilerPath%;%VECTORCAST_DIR%
vcastqt.exe
```

Note that The GreenHills Integrity RSP is very consistent across different CPUs (PPC, ARM, etc.). For this application note, any reference to a CPU type will be PPC. Any change to the CPU type should be obvious and the RSP should work in the same manner. In addition, the Integrity Simulator works in the same manner as a real target, except for the connection method. Differences in the connection method will be noted, but the rest of the configuration should mostly be the same.

2.2 Selecting the Proper Configuration

There are 3 build types for building Integrity Applications:

- Static
  - This is for applications that run in Kernel Space
- Dynamic
  - This is for applications that run in an Address Space and are downloaded separately from the Kernel
- Monolith
  - This is for applications that also run in an Address Space but are packaged with the Kernel and both the Kernel and application are downloaded as one monolithic image.

Typically, you would want to select the build type that the code under test would use in the system environment (outside of VectorCAST). However, most application code can run either in the kernel or in an address space, so there may be reasons to use a different build type (to improve execution time or for other reasons). Some experimentation may be needed before selecting the most expedient build type.

In addition to the Build Type, you will also need to select one of 3 different Integrity Versions:

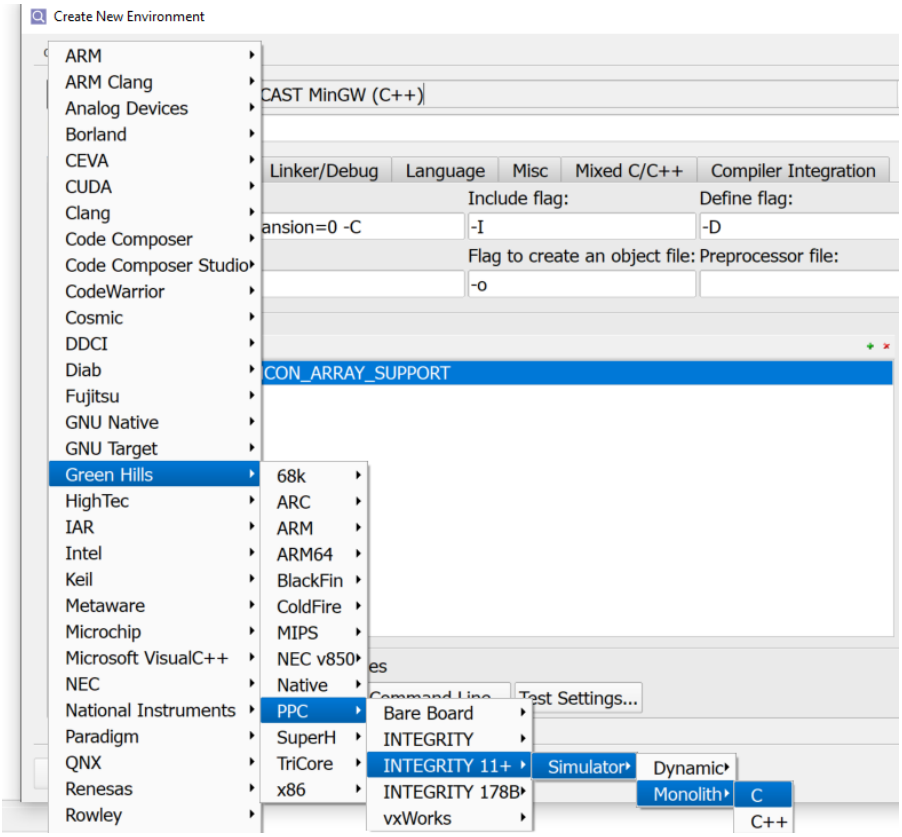
- Integrity 178b
  - This is a special version of Integrity usually used for Safety Certified applications. This version may encounter problems when building the environment or executing tests and you may need to work with a VectorCAST Field Applications Engineer to correct any issues. Some customers may want to use the commercial version of Integrity to get started with their testing efforts because it may save development time, then switch to Integrity 178b at a later point. The test development should easily port between versions.
- Integrity 11
  - This is “commercial” Integrity and is the most common and recent version of Integrity. This version should work out-of-the-box.
- Generic Integrity

- This option is for all other generic versions of Integrity, other than Integrity 11. This may or may not work out of the box. There are suggestions below in section 2.4 on how to diagnose any issues.

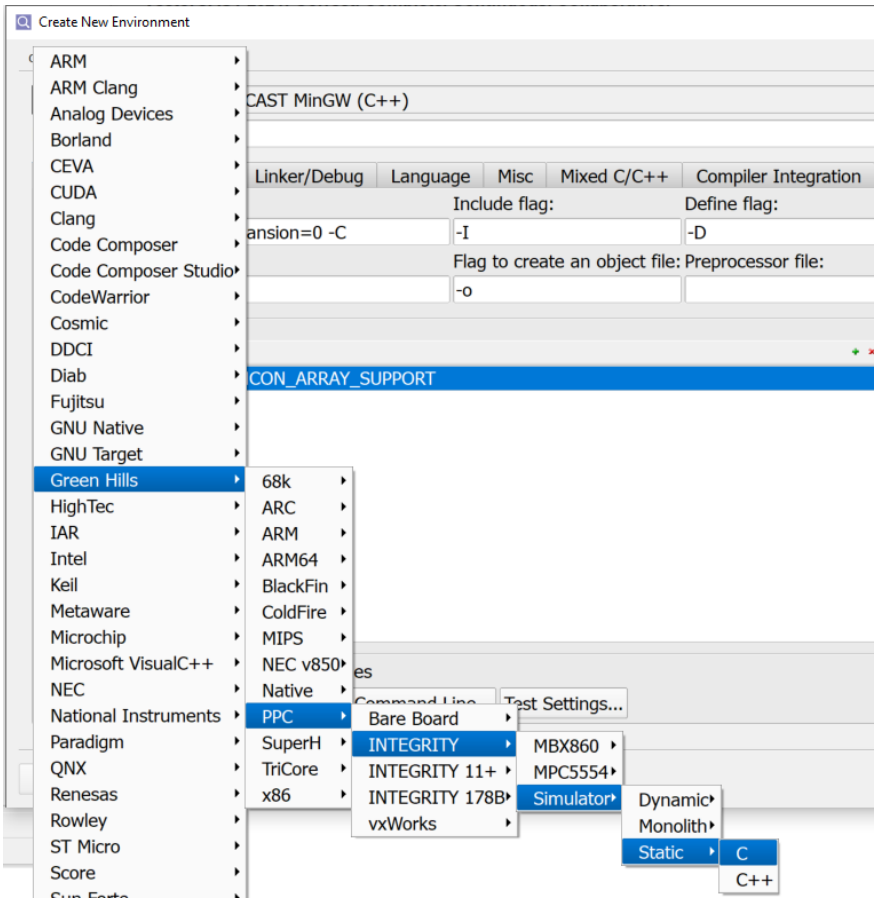
2.3 Building an Environment

First, select an empty directory for your work area. You can use the same working directory that have the same configuration. If the configuration needs to change, it is suggested to create a new working directory, so it doesn't affect the original work area. Before setting the work directory, make sure all current environments are closed, then do File->SetWorkingDirectory. Once the working directory is set, you can create a new test environment based on the Integrity Build Type and Integrity version, as discussed above. To create a new environment, do File->New C/C++ Unit Test Environment, and then select the appropriate compiler. Here are a couple example screenshots:

> Monolith Build Type for Integrity 11:



- Static Build Type for generic Integrity:



2.4 Testing Options

There are a lot of variables involved when building a VectorCAST test harness and running tests on GreenHills Integrity, such as:

- Integrity Version
  - Commercial vs. 178b
  - Commercial version 5.x vs. 11.x, etc.…
- Build Type
  - Static vs. Monolith vs. Dynamic
- Compiler Version & Location
  - In older versions of Multi, the compiler was in the Multi directory
  - In new versions, the compiler is in its own directory
- Debugger (Multi) Version
  - The execution script can be different for each version of the Multi Debugger
- Connection Method
  - Simulator vs. MPServ vs. RTServ
- Output Mechanism
  - Host File I/O vs. Standard Output

Because of these variables, it's hard to test all combinations. VectorCAST offers a few example combinations when you create a new environment, but those selections may or may not work because of some of the complexities listed above. This section will suggest a progression of test environments, starting with the simplest case and building up to more complicated scenarios. In this manner, you can build confidence in the integration between VectorCAST and Greenhills and also become familiar with how to diagnose any problems.

2.4.1 Greenhills Multi New Project Wizard

First, it is suggested to become familiar with the Greenhills Multi New Project Wizard, to build both standalone and Integrity builds. Also, since VectorCAST typically uses the debugger to control execution of the Test Harness, it is important to understand how to debug those projects outside of VectorCAST. For starters, build a "hello, world" standalone application in Multi that you can build and debug using the simulator. Then, add a simple routine like the following (i.e. "add2") which you can later test in VectorCAST:

```
#include <stdio.h>
int add2(int x, int y)
{
    return(x + y);
}
int main(int argc, char *argv[])
{
    printf("Hello world.  Add2: %d\n", add2(3,4));
    return 0;
}
```

After successfully debugging the above code in the Multi Debugger, repeat this process for the various Integrity Build Types (Static, Monolith, & Dynamic). Once familiar with these processes in Multi, you can proceed to VectorCAST to start building a test harness against this code.

2.4.2 Testing Strategy

While the ultimate goal may be to test Integrity application code running on a live target, it is best to take an incremental approach and start with some simple code (like the above "add2()" function) running on the simulator. The sequence of steps should be as follows:

1) Test simple code on the standalone simulator (simppc)
2) Test simple code on the Integrity simulator (isimppc)
3) Test proprietary application code on the Integrity simulator (isimppc)
4) Test simple code on an Integrity target
5) Test proprietary application code on an Integrity target

In addition, most Integrity application code can run in one of 3 modes:

1) Static: Inside the Integrity kernel
2) Monolith: In its own address space, but packaged with the Kernel
3) Dynamic Download: In its own address space, but downloaded separately from the Kernel

You may want to experiment with each build type to see what is most conducive to your testing requirements. The Test Scenarios below will give examples of each. Once you've selected the build type you desire, using the other 2 build types is probably not necessary.

When selecting a compiler option, you typically will have a C & C++ option. C++ will work for both C & C++ code, so select C++ unless you know for sure you will only be using C code.

2.4.3 Test Scenarios

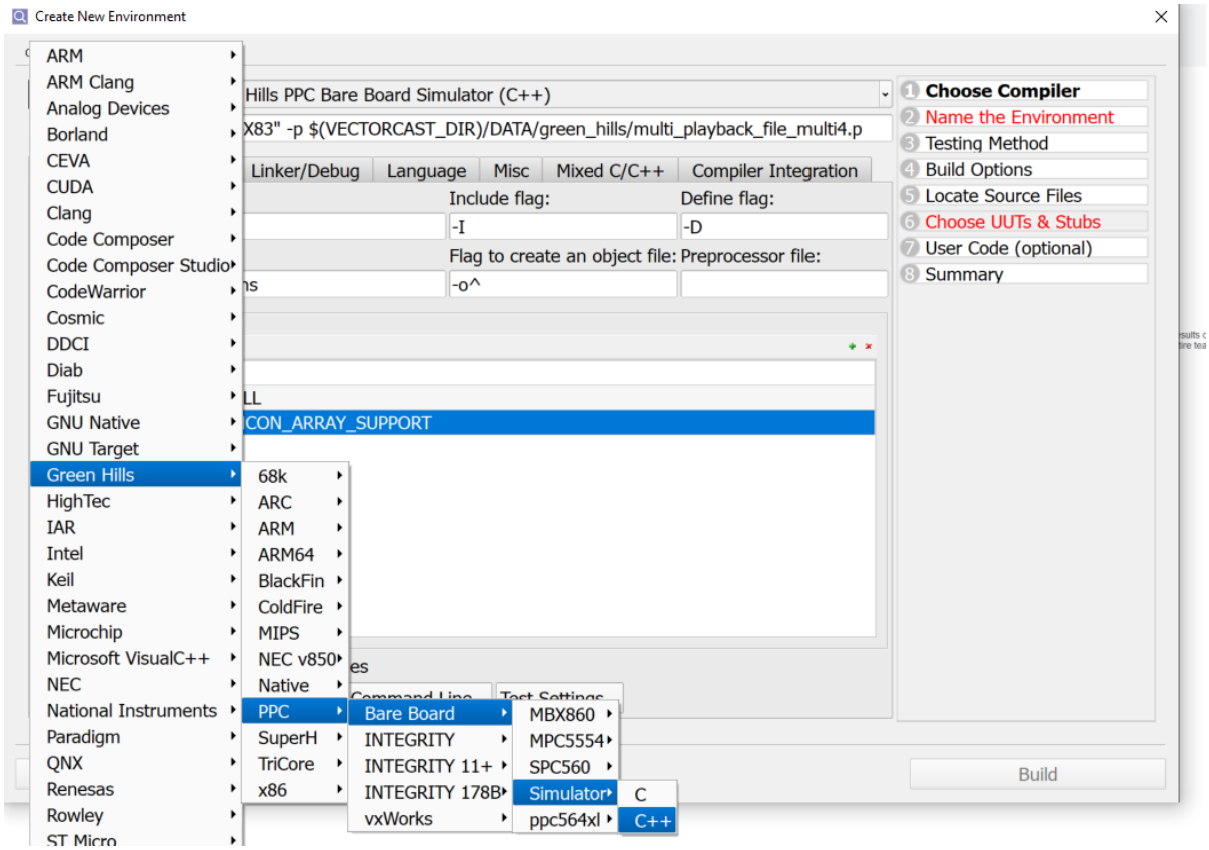
This section will describe how to build a VectorCAST test harness for both standalone and Integrity build types (note: it is assumed that the reader has experience with VectorCAST and knows how to build test environments in general).

In each of these scenarios, create a new working directory so you can keep your configuration separate. When selecting a new working directory, all environments need to be closed, then do File->SetWorkingDirectory and select a blank directory with an appropriate name. Then do File->New->C/++UnitTestEnvironment, and chose the compiler as depicted in the following screen shots.

If any of these test scenarios fail, it's suggested to fix that problem before proceeding to other scenarios because the same problem may persist.

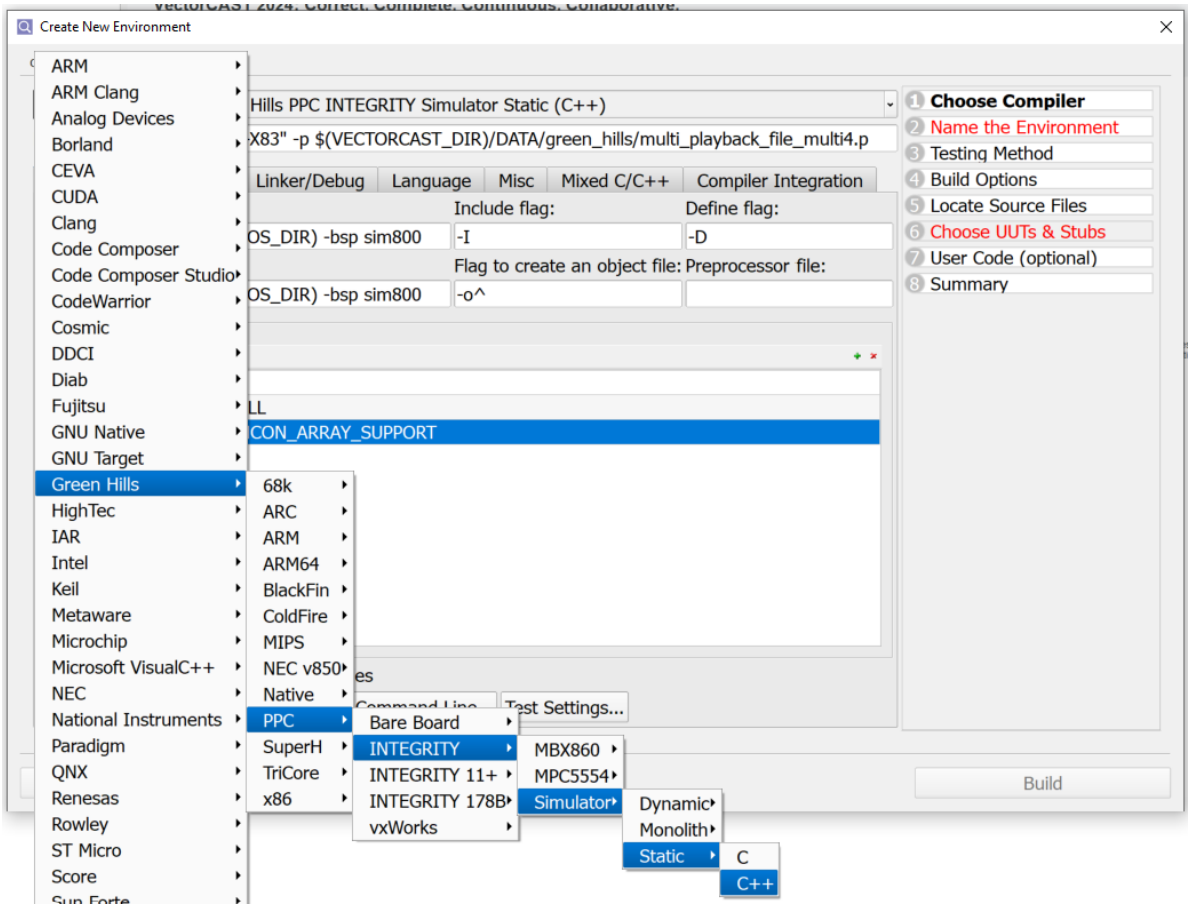
2.4.3.1 Standalone Code running on the Simulator (simppc)

It's a good idea to start with a standalone project to verify the basic integration between VectorCAST and Greenhills. Once the environment is built, ensure that test cases run successfully on the standalone simulator. Here is a screenshot that shows the compiler selection:



2.4.3.2 Static Integrity Running on the Integrity Simulator (isimppc)

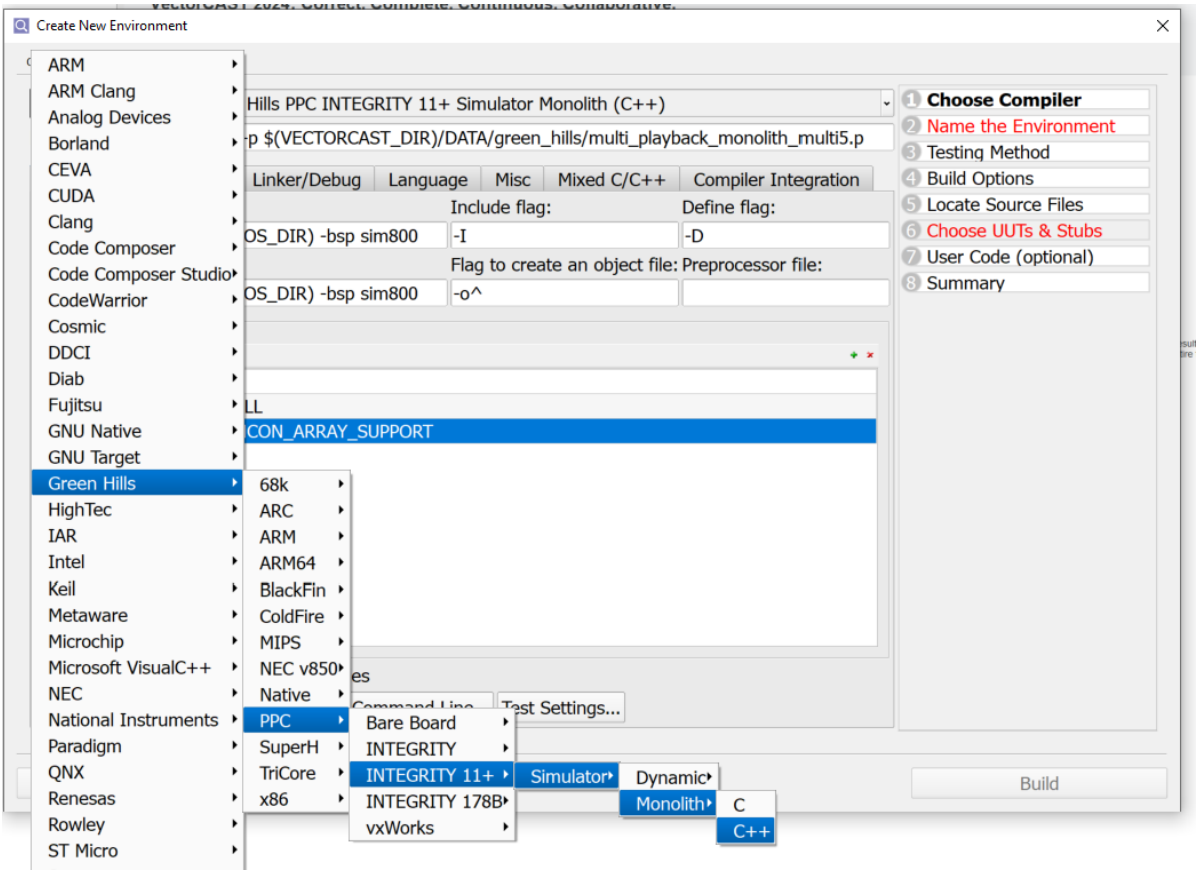
Even if you've decided to use an Integrity Monolith or Dynamic Download for your testing purposes, it's a good idea to start with a Static Kernel. This is typically the simplest Integrity build type and it's good to understand how that works. Once the environment is built, ensure that test cases run successfully on the Integrity Simulator (isimppc). Here is a screenshot that shows the compiler selection:



2.4.3.3 Monolith Integrity Running on the Integrity Simulator (isimppc)

Using a Monolithic image means that the test harness runs in its own address space but is combined into one file with an Integrity Kernel. The build process for a Monolith is similar to a Dynamic Download but executing the test harness is simpler because there is only 1 file that needs to be considered (i.e. the monolithic image). The utility that controls the linkage between the application address space and the Kernel is called “intex” and you’ll see that command on the linker options in VectorCAST. It uses a specification file with a “.int” extension to control the linkage. “intex” is used by both Monolithic & Dynamic Download build types, but it is not used with Static builds.

Here is a screenshot that shows the compiler selection for a Monolith:



2.4.3.4 Dynamic Download Image Running on Integrity

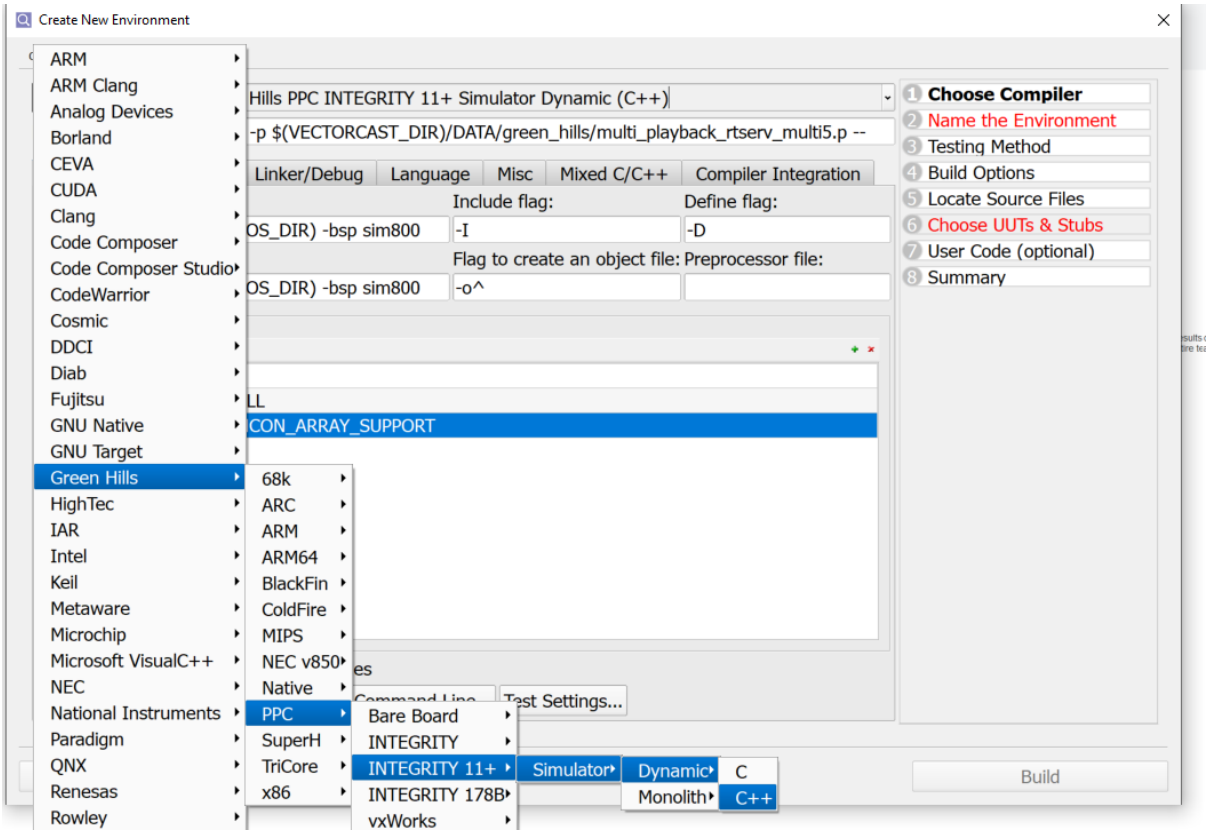
As mentioned above, a Dynamic Download image is similar to the image used with a Monolith. The only difference is that it is not combined with a Kernel image in a monolithic file. To run a Dynamic Download image, it needs to be downloaded to a target that already has a Kernel running. This is done thru a target server utility called RTSERV (or RTSERV2). On the execute line, you'll see something like this: "multi -nosplash -remote "rtserv2 -port udp@localhost …" , where "udp@localhost" is the IP Address of the Integrity Target. This IP address could be the host computer (running isimppc with a prebuilt Kernel image) or an IP address of a physical target running an Integrity image for that target.

Note that when using the Integrity simulator (isimppc), the Kernel needs to be built with the RTSERV debug agent. The default Kernel for the simppc BSP should be adequate. Here is an example of how to run isimppc with that image.

```
isimppc C:\GHS\int1144\bin\sim800\kernel
```

Alternatively, the kernel can be started up in the Multi Debugger. Regardless of how the Kernel is started, VectorCAST should be able to connect to the image and download the Test Harness.

Here is a screenshot that shows the compiler selection for a Dynamic Download:



2.4.4 Advanced Test Scenarios

At this point, you've taken some simple code and have run it on the Standalone simulator (simppc) and with at least one of the Integrity Build Types (Static, Monolith, and/or Dynamic) using the Integrity simulator (isimppc). You should now be ready to take similar steps with the following scenarios:

- Proprietary code running on the Integrity Simulator with one (or all) of the Integrity build types
- Simple code running on an actual Integrity target
- Proprietary code running on an actual Integrity target

2.5 Use Case: Debugging

VectorCAST typically uses the Multi Debugger to download and control the execution of the Test Harness. To do this, it invokes Multi and passes in a debug script (*.p file) to control the execution. Here is an example execution command:

```
multi -remote "isimppc -cpu=ppc860" -p multi_playback_monolith_multi5.p
```

There are a series of debug scripts (*.p files) located in the VectorCAST installation directory for THE various build types. You can find those scripts here:

```
%VECTORCAST_DIR%\DATA\green_hills
```

If for some reason the Test Harness doesn't execute properly, the debug script can be modified to help diagnose the problem (you will see some examples with a "_debug" suffix). With your environment open, go to Tools->Options->C/C++->LinkerDebug and note the "debugger command". This will be very similar to the execute command, but with a different script. If that command is not present, then copy the execute command to the debug command and replace the debug script with one of your own (see one of the "_debug" scripts for an example, such as

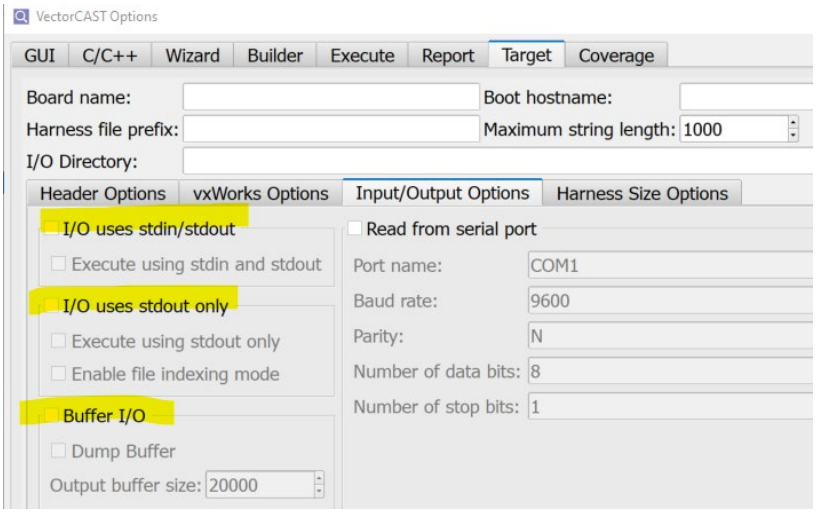
"multi_playback_monolith_multi5_debug.p"). This will allow you to debug the Harness using the Multi Debugger.

2.6 Miscellaneous Options

2.6.1 Host I/O

The Multi Debugger supports a mode called "Host I/O", where I/O calls are captured on the target and executed on the Host. When Host I/O is used, the Test Harness data gets written directly to files in the host's environment directory, instead of being written to standard output. Host I/O is typically slower than Standard Output because the debugger needs to handle an exception on each I/O call. However, it has the advantage that the test harness does not need to get rebuilt for each test case. If rebuild time is excessive, Host I/O may be an option. To enable Host I/O, do the following:

- Add "-lhostio" as a Linker option
- Disable Standard Output, as follows:



2.7 Speed Improvement: Using the Integrity Simulator Directly

When test cases are run in VectorCAST, the execution command will invoke Multi and connect to the Integrity Simulator (isimppc) and then use the debugger to control execution. A breakpoint is typically send at the end of the Test Harness and when the break is hit, the debugger will exit and VectorCAST will be triggered to process the output data. However, this is somewhat slow because of the interaction with the debugger. To improve this process, it's possible to run isimppc directly without the debugger. For example, the execution command would look like this:

- Isimppc uut_inst.exe

When you do that, you should see the Test Harness output in the command window. However, the problem is that VectorCAST doesn't know when the harness is finished. To solve this problem, there is a utility that can monitor the output and detect when the "end" string is written. Here is what that command would look like:

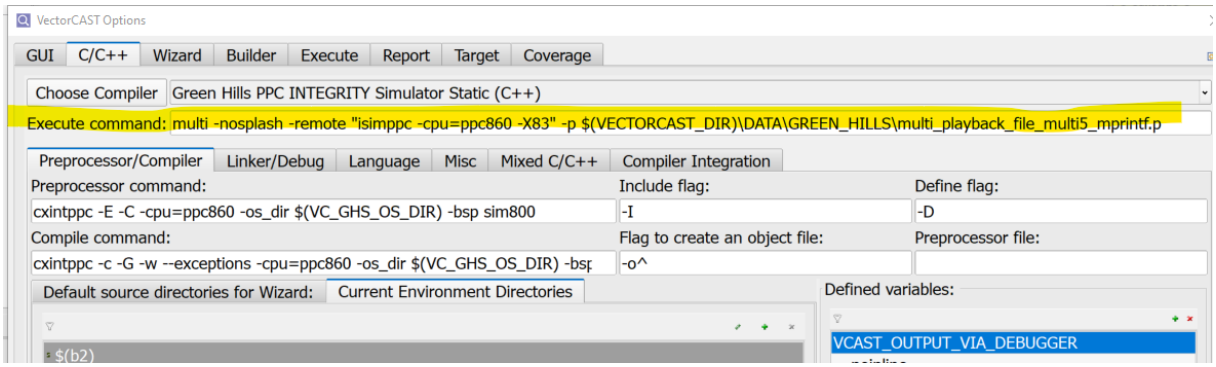
- pty monitor_stdout isimppc uut_inst.exe

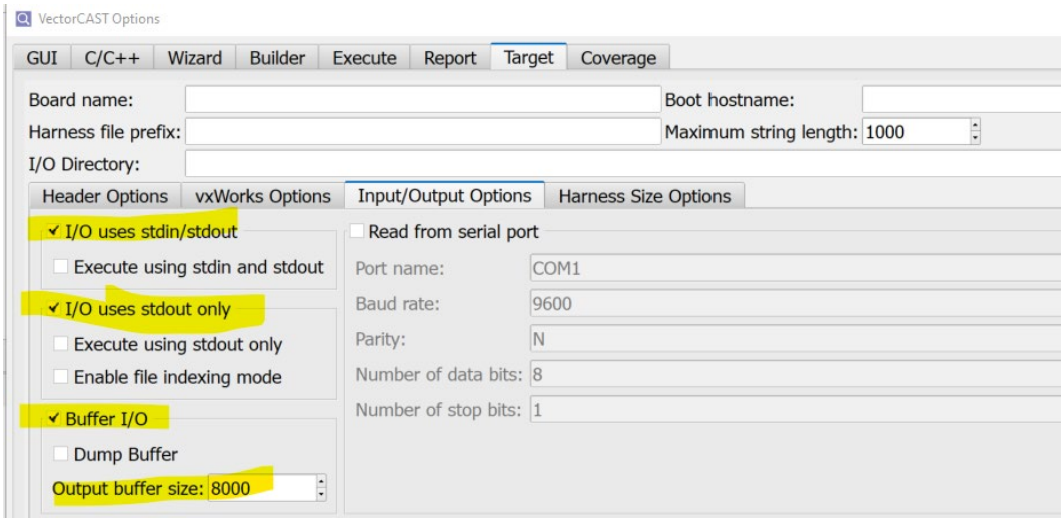
This utility (i.e. "pty monitor") is not fully documented, but it should work, and you should see improved execution time.

Note this only works for Static & Monolithic builds. Also, standard output needs to be enabled.

2.8 Speed Improvement: Using "`mprintf`" instead of "`printf`"

As an alternative to the method described in the previous paragraph, the GreenHills debugger has a feature that can speed up standard output. This is called "`mprintf`" and it is functionally equivalent to "`printf`" except that the output is buffered and reduces the number of i/o calls. To enable this, you need to define the symbol `VCAST_OUTPUT_VIA_DEBUGGER` when building the harness. Also, "dump buffer" needs to be disabled and the buffer size must be limited to 8k (8192) due to a limitation in mprintf. The advantage of doing "mprintf" over "printf" is that it is usually much faster. Here are 2 screen shots of the required configuration items:





3 Additional Resources

4 Trademarks

All mentioned names are either registered or unregistered trademarks of their respective owners.

5 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.