How to Run Industry-Standard Benchmarks on the Cyclone V SoC and Arria V SoC FPGAs

1 Overview

1.1 Introduction

This how-to guide discusses how to obtain, compile and run industry-standard processor performance benchmarks on the Cyclone V SoC Development Kit. Using the steps outlined in this document, you can understand how to obtain and run benchmarks on your system and compare to the stated results. Once you have validated your system setup and measurement techniques using this document, you can choose additional benchmarks or create your own benchmarks which best represent your end application for performance analysis purposes. Because the dual-core ARM® Cortex™-A9 processor subsystem is the same architecture between the Cyclone V SoC and the Arria V SoC, the techniques described in this application note can be readily applied to the Arria V SoC Development Kit, although the Arria V SoC results will be faster because it has a maximum processor frequency of up to 1.05 GHz and a higher performance main memory subsystem.

1.2 Benchmark Overview

CoreMark, Dhrystone, and Whetstone are processor core related benchmarks. Please refer to the table below to understand the goals of and hardware elements used by each benchmark.

Benchmark	Emphasis	Hardware Elements Utilized
CoreMark	Processor core, cache memory read/write	CPU Pipeline, L1 cache
Dhrystone	Integer and branch operations	CPU Pipeline, Integer ALU, L1 cache
Whetstone	Floating point operations	CPU Pipeline, NEON/FPU, L1 cache

STREAM and LMbench are memory intensive benchmarks. They are commonly used and well respected benchmarks for measuring embedded system memory performance. The table below summarizes the differences between these two benchmarks.

Benchmark	Emphasis
STREAM	Mostly sequential data stream of memory writes and reads
LMBench	Random memory writes and reads

Using different tool chains to compile the benchmarks may result in different benchmark results. The results are also affected by compile flags such as optimization level, single core or dual cores, whether or not neon is used, or whether or not hardware floating point is used.

1.3 Setup: Device, Hardware and Software

- Device and Hardware
  - Cyclone V SoC Development Kit Rev C
  - CPU Core frequency: 925 MHz
  - DDR frequency: 400 MHz, 32bit physical interface width
- Software
  - SD pre-build image stored at <QUARTUS_INSTALL>\embedded\embeddedsw\socfpga\prebuilt_images folder after install SoC EDS tool, pre-build image used in this guide is included in SoC EDS 15.0 tool.
  - Arm-linux-gnueabihf-gcc, Linux GCC compiler tool chain from Linaro, version 4.8.3 shipped with the SoC EDS 15.0, after ds-5 is installed on the machine, this GCC compiler is located at <QUARTUS_INSTALL>\embedded\ds-5\sw\gcc, the default configuration of this toolchain is:
    - Runs on all Cortex-A profile devices
    - Tuned for the Cortex-A9
    - Thumb-2

        - 'hard float' calling convention
        - Uses the VFPv3-D16 FPU
        - Multiarch and multilib enabled
    - Linux OS with kernel version 3.10.31-ltsi

1.4 Compile Options

The following options can be used during binary building process:

- -lm (Math Library)
- -mcpu=cortex-a9 (Gears compilation to the cortex-A9)
- -mfloat-abi=hard (Makes use of the floating-point unit)
- -O2, O3 or Ofast (Optimization level 2, 3 and fast)
- -fopenmp (Makes use of multiple cores if the benchmark uses it)
- -mfpu=neon (Uses neon)
- -lrt and -lpthread (helpful for boosting speed by using rt and pthread library)

For the arm-linux-gnueabihf-gcc toolchain used in this guide, the –mcpu, -mfloat-abi, and -mfpu options are not needed during the build process because the toolchain uses these options by default. If you use an alternative toolchain, such as Code Sourcery Pro, you must use these three options explicitly. The examples shown in the Compile sections below uses all the needed options explicitly in the command line. You can refer to this command line to determine the correct flags and add them into the command line of the toolchain you choose as needed.

1.5 Moving Compiled Binaries onto an SD Card

This section describes how to copy a binary file onto an SD card. Your SD card with a Linux install should have a Linux filesystem.

1. Plug your SD card into the SD card reader attached to your PC.
2. Choose a directory to be your working directory and copy your compiled binaries to that directory on the SD card.

2 Coremark

2.1 Introduction

CoreMark 1.0 is capable of testing a processor's basic pipeline structure, as well as its ability to efficiently complete basic read/write operations, integer operations, and control operations. Coremark was designed to replace Dhrystone, like Dhrystone benchmark, it entirely fits in the L1 cache of the Cortex-A9 also.

To obtain a copy of the benchmark, go to http://www.eembc.org/coremark/download.php

1. Click on the register link.
2. Fill out the registration form.
3. Wait for a response e-mail to arrive.
4. Follow the e-mail link to a login page, and use your login information.
5. Click the download link.
6. Download the top four links.

2.2 Compile

Unpack the software tarball to the default directory
Move to that directory, and go into /coremark_v1.0/linux
Edit the core_portme.mak file to define the toolchain and options:

1. Change the line that says "CC" to "CC = arm-none-linux-gnueabihf-gcc"
2. Change the line that says "EXE = exe" to "EXE= ", otherwise the tool generates binary file with postfix .exe.
3. Comment out this line: #FLAGS_STR = "$(PORT_CFLAGS) $(XCFLAGS) $(XLFLAGS) $(LFLAGS_END)", use command line arguments to main during compiling process, and save the file

Edit the core_portme.h to execute both threads in parallel on dual core HPS devices. If you don't make this change, only one thread is running and only one core is used in the process, greatly reducing the performance on the benchmark.

1. Change "#define USE_PTHREAD 0" to "#define USE_PTHREAD 1"
2. Change "#define MULTITHREAD 1" to "#define MULTITHREAD 2"
3. Save the file.

To compile the benchmark, move up one level and navigate to the /coremark_v1.0 folder, then type:

$make compile PORT_DIR=linux XCFLAGS="-O3 -Ofast -mcpu=cortex-a9 -mfpu=neon -lrt -lpthread "

```
bash-4.1$ make compile PORT_DIR=linux XCFLAGS="-O3 -Ofast -mcpu=cortex-a9 -mfpu=neon -lrt -lpthread"
arm-linux-gnueabihf-gcc  -Ilinux -I. -DFLAGS_STR=\"\" -DITERATIONS=0 -O3 -Ofast -mcpu=cortex-a9 -mfpu=neon -lrt -lpthread core_l
atrix.c core_state.c core_util.c linux/core_portme.c -o ./coremark -lrt
```

- -lrt and -lpthread gives a speed boost.
- -O3 and –Ofast gives the best performace

Move the coremark binary file onto the SD Card.

2.3 Execute

1. On your development board, insert the SD card and power up the board.
2. Navigate to the directory where the coremark binary is located.
3. Execute the binary in the embedded command shell:

4. $./coremark

2.4 Result

```
root@socfpga:~# ./coremark
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 12163
Total time (secs): 12.163000
Iterations/Sec   : 4932.993505
Iterations       : 60000
Compiler version : GCC4.8.3 20131202 (prerelease)
Compiler flags   :
Parallel PThreads : 2
Memory location  : Please put data memory location here
                        (e.g. code in flash, data on heap etc)
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[1]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[1]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[1]crcstate      : 0x8e3a
[0]crcfinal      : 0x5275
[1]crcfinal      : 0x5275
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 4932.993505 / GCC4.8.3 20131202 (prerelease)  / Heap / 2:PThreads
```

The Coremark score for the Cyclone V SoC is 4932.

Cyclone V SoC Coremark/MHz
4932/925= 5.33 Coremark/MHz

For the Arria V SoC running at 1.05 GHz, the EEMBC-certified CoreMark score is 5654.

Arria V SoC CoreMark/MHz
5654/1050 = 5.38 CoreMark/MHz

It is strongly recommended that you run the CoreMark benchmark on your test system first. This way you can validate your system setup and measurement techniques before moving to more advanced system level benchmarks such as LMBench, STREAM or other EEMBC system application benchmark suites (e.g. TeleBench, AutoBench, etc.).

3 Whetstone

3.1 Introduction

This section describes the steps of running the Whetstone benchmark on the Altera Cyclone V SoC FPGA.

The Whetstone benchmark 1.2 is designed to measure the processor performance for double precision floating point numerical applications. The provided benchmark is normalized by the CPU clock frequency.

3.2 Compile

whetstone.c can be downloaded from www.netlib.org/benchmark/whetstone.c

Compile the benchmark using the following optimizations:

$ arm-linux-gnueabihf-gcc –o whetstone_arm ./whetstone.c -lm -O3 -mcpu=cortex-a9 -mfloat-abi=hard -fopenmp -mfpu=neon -static

```
arm-linux-gnueabihf-gcc -o whetstone_arm ./whetstone.c -lm -O3 -mcpu=cortex-a9 -mfloat-abi=hard -fopenmp -mfpu=neon -static
```

- -O3 gives the best performace
- -mfloat-abi = hard is to use hardware float-point unit
- -fopenmp makes using multicores

Copy the whetstone_arm file onto the SD Card.

3.3 Execute

1. On your development board, insert the SD card and power up the board.
2. Move to the directory with the whetstone_arm binary.
3. Execute the binary in the embedded command shell:
4. $./whetstone 5000000

```
root@socfpga:~# ./whetstone_arm 5000000

Loops: 5000000, Iterations: 1, Duration: 327 sec.
C Converted Double Precision Whetstones: 1529.1 MIPS
```

3.4 Result

The Whetstone result is 1529.1 MIPS.

Whetstone result – Normalized per Core Frequency
1529.1/925 = 1.65 MIPS/MHz

4 Dhrystone

4.1 Introduction

The Dhrystone 2.1 benchmark contains no floating point operations, so it provides an indicator of general-purpose ("integer") performance of new computers. Its small size allows it to easily fit inside most L1 caches; as a result, the results demonstrate CPU performance and not system performance.

Download dhry-c from www.netlib.org/benchmark/dhry-c.

4.2 Compile

To compile the Dhrystone benchmark, type the following in your terminal:

$ sh dhry-c

$arm-linux-gnueabihf-gcc -O2 -mcpu=cortex-a9 -mfloat-abi=hard -fopenmp -static -mfpu=neon -o dhrystone_arm -DTIME dhry_1.c dhry_2.c

```
bash-4.1$ arm-linux-gnueabihf-gcc -O2 -mcpu=cortex-a9 -mfloat-abi=hard -mfpu=neon -o dhrystone_arm -static -DTIME dhry_1.c dhry_2.c
dhry_1.c:31:18: warning: conflicting types for built-in function ‘malloc’ [enabled by default]
 extern char     *malloc ();
                  ^
dhry_1.c: In function ‘main’:
dhry_1.c:94:3: warning: incompatible implicit declaration of built-in function ‘strcpy’ [enabled by default]
   strcpy (Ptr_Glob->variant.var_1.Str_Comp,
   ^
```

- -O2 gives the best performace
- -fopenmp makes using multicores

Move the dhrystone_arm file onto the SD Card.

4.3 Execute

1. On your development board, insert the SD card and power up the board.
2. Navigate to the directory where the whetstone_arm binary file is stored.
3. Execute the binary in the embedded command shell:
4. $./dhrystone_arm
5. When you input the number of runs, you are prompted to input a bigger number if your chosen number is not big enough.

4.4 Result

DMIPS = 2941176.5/1757 = 1674

DMIPS/MHz = 1674/925= 1.81

The execution process and result are shown in the figure below.

```
root@socfpga:~# ./dhrystone_arm

Dhrystone Benchmark, Version 2.1 (Language: C)

Program compiled without 'register' attribute

Please give the number of runs through the benchmark: 100000000

Execution starts, 100000000 runs through Dhrystone
Execution ends

Final values of the variables used in the benchmark:

Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    100000010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          472584
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          472584
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
        should be:   18
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:           5
        should be:   5
Int_2_Loc:           13
        should be:   13
Int_3_Loc:           7
        should be:   7
Enum_Loc:            1
        should be:   1
Str_1_Loc:           DHRYSTONE PROGRAM, 1'ST STRING
        should be:   DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
        should be:   DHRYSTONE PROGRAM, 2'ND STRING

Microseconds for one run through Dhrystone:    0.3
Dhrystones per Second:                      2941176.5
```

5 LMBench

5.1 Introduction

The LMbench 3.0 benchmark assesses random data access performance. LMBench testing items shown in this guide includes memory read, memory write, partial memory read, partial memory write, and partial memory read/write performance. The benchmarks included in LMBench 3.0 include the following:

- Bandwidth benchmarks
    - Cached file read
    - Memory copy (bcopy)
    - Memory read
    - Memory write
    - Pipe
    - TCP
- Latency benchmarks
    - Context switching
    - Networking: connection establishment, pipe, TCP, UDP, and RPC hot potato
    - File system creates and deletes
    - Process creation
    - Signal handling
    - System call overhead
    - Memory read latency
- Miscellaneous
    - Processor clock rate calculation

To obtain a copy of the benchmark,

1. Go to www.bitmover.com/lmbench/get_lmbench.html
2. Click the link to download LMBench
3. Unpack the software tarball to the default directory
4. Navigate to the extracted lmbench3 directory

5.2 Compile

When compiling this benchmark, you may encounter this error message:

make[2]: * No rule to make target `../SCCS/s.ChangeSet', needed by `bk.ver'. Stop.**

To avoid this error, type the following into your terminal:

$ mkdir ./SCCS

$ touch ./SCCS/s.ChangeSet

To compile the LMBench benchmark, type:

$ make CC=arm-linux-gnueabihf-gcc OS=armv7l-linux-gnu CFLAGS=“ -mcpu=cortex-a9 -Ofast -lrt”

```
make CC=arm-linux-gnueabihf-gcc OS=armv71-linux-gnu CFLAGS="-mcpu=cortex-a9 -Ofast -lrt"
```

- -lrt gives a speed boost.
- -Ofast gives the best performace

Navigate up one directory.
Copy the entire lmbench3 directory to the SD card.

5.3 Execute

1. On your development board, insert the SD card and power up the board.
2. In your terminal, navigate to the lmbench3/scripts directory
3. Type:
   $ ./config-run
4. Follow the benchmarking wizard based on the test you wish to run. For example, the following configuration runs the memory read and write bandwidth benchmark:

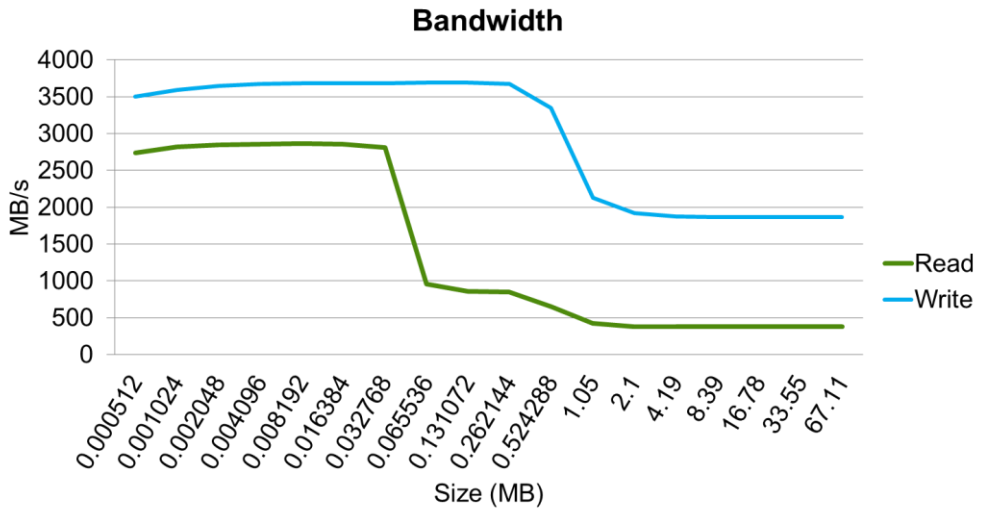
   MULTIPLE COPIES [default 1]: 1
   Job placement selection [default 1]: 1
   MB [default 350]: 64
   SUBSET (ALL|HARWARE|OS|DEVELOPMENT) [default all]: Enter
   FASTMEM [default no]: Enter
   SLOWFS [default no]:Enter
   DISKS [default none]: Enter
   REMOTE [default none]: Enter
   Processor mhz [default 498 MHz, 2.0080 nanosec clock] 925
   FSDIR [default /tmp] :Enter
   Status output file [default /dev/tty] :Enter
   Mail results [default yes] no

5. After the configuration process is finished, type the command below in the embedded command shell:
   $ ./results
6. Wait for the test to finish. This test could take some time; for example the configuration above requires about 40 minutes to complete. If you choose 32 MB instead of 64MB, the test runs in about half the time. After the benchmark testing is finished, the results are written to the result/armv7l-linux-gnu folder, with the name socfpga.X, where X is the run number. This protocol prevents previous result files from being overwritten.
7. Next, navigate to the lmbench3/results/armv7l-linux-gnu/ directory
8. Read the socpfpga.X file to see the benchmark results, this socfpga.X file can be opened by a text editor after you copy it onto the PC.
   You may find following items in the result file.
   - Memory read bandwidth
   - Memory partial read bandwidth
   - Memory write bandwidth
   - Memory partial write bandwidth
   - Memory partial read/write bandwidth

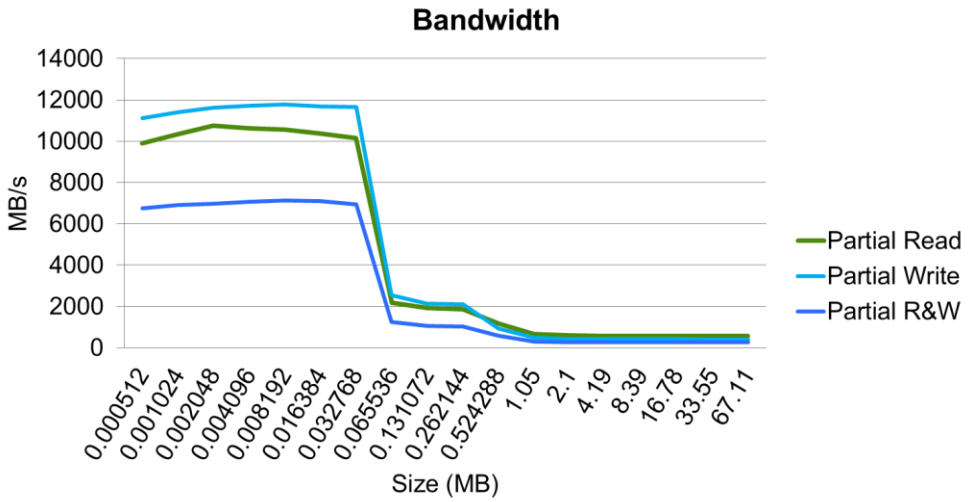
5.4 Result

The vertical axis shows the memory bandwidth vs. the data transfer size along the horizontal axis. (Higher is better for the memory bandwidth.) The curve can be grouped into three stages as the data size grows from requiring only the L1 cache (32KB data + 32KB instruction) to requiring only the L1 and the L2 caches (512KB shared) to requiring the use of external memory. So you can see that read throughput begins to decrease when the data size is bigger than 32KB and write throughput begins to decrease when the data size is bigger than 512KB.

Please refer to diagram below for information about memory read and write bandwidth.



Please refer to diagram below for memory partial read, partial write and partial read/write bandwidth results.



6 STREAM

6.1 Introduction

The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. STREAM measures performance using a sequential data stream of memory writes and reads.

To obtain the benchmark, go to www.cs.virginia.edu/stream/FTP/Code/, and download the files in the link.

6.2 Compile

To compile the benchmark, open your terminal, navigate to the directory where the stream source code is storedand type:

$ arm-linux-gnueabihf-gcc –o stream_arm ./stream.c -Ofast -mcpu=cortex-a9 -fopenmp -static

```
arm-linux-gnueabihf-gcc -o stream_arm ./stream.c -Ofast -mcpu=cortex-a9 -fopenmp -static
```

- -Ofast gives the best performace
- -fopenmp makes using multicores

Move the stream_arm binary file onto the SD Card.

6.3 Execute

1. On the development board, insert the SD card and power up the board.
2. In your terminal, navigate to the directory where the stream_arm binary file is stored and execute the binary in the embedded command shell:
   ./stream_arm

6.4 Result

The figure below details the execution process and results.

```
root@socfpga:~# ./stream_arm
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 2
Number of Threads counted = 2
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 139294 microseconds.
   (= 139294 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            994.6     0.161289     0.160874     0.161809
Scale:          1270.0     0.126422     0.125988     0.127042
Add:            1149.3     0.209479     0.208818     0.210546
Triad:          1116.7     0.215441     0.214919     0.215914
-------------------------------------------------------------
```

For normalized STREAM memory bandwidth (MB/s) results per memory bus frequency, please refer to the table below.

Function	Rate (MB/s)
Copy	2.48
Scale	3.17
Add	2.87
Triad	2.79

7 Conclusion

This document describes the process of how to obtain, compile and execute benchmark results for the CoreMark, Dhrystone, Whetstone, LMBench and STREAM processer performance benchmarks on the Cyclone V SoC Development Kit. Because the processor subsystems are the same, these binary files can also be used on the Arria V SoC Dev kit. It is recommended that you run the CoreMark benchmark on your test system first to validate your system setup and measurement techniques before moving on to more advanced system-level benchmarks. Using these techniques, you can select the benchmarks that best represent major elements of your end application or create your own benchmarks for further performance analysis. All benchmark results are dependent on the system configuration and the benchmark numbers stated in this report are only valid for the system configurations shown.