



ABSTRACT

This application note contains benchmarks for the AM64x and AM243x family of devices.

Table of Contents

1 Introduction	2
2 Processor Core Benchmarks	4
2.1 Dhrystone.....	4
2.2 Trigonometric Functions.....	4
3 Compute and Memory System Benchmarks	5
3.1 Memory Bandwidth and Latency.....	5
3.2 CoreMark®-Pro.....	7
3.3 Fast Fourier Transform.....	7
3.4 Cryptographic Benchmarks.....	7
4 Application Benchmarks	8
4.1 Machine Learning Inference.....	8
4.2 Field Oriented Control (FOC) Loop.....	9
4.3 PCIE to DDR Performance Using BCDMA.....	10
4.4 DDR to DDR Performance Using BCDMA.....	13
5 References	16
6 Revision History	16

List of Figures

Figure 1-1. Functional Block Diagram: AM64x.....	2
Figure 1-2. Functional Block Diagram: AM243x.....	3
Figure 4-1. Speed Closed Field Oriented Control Loop.....	9
Figure 4-2. AM64x-PCIE Benchmarking Setup.....	11
Figure 4-3. AM64x (PCIE benchmarking) Data Flow.....	11
Figure 4-4. PCIE Performance Graph.....	13
Figure 4-5. AM64x DDR to DDR Data Flow.....	14
Figure 4-6. DDR Performance Graph.....	15

List of Tables

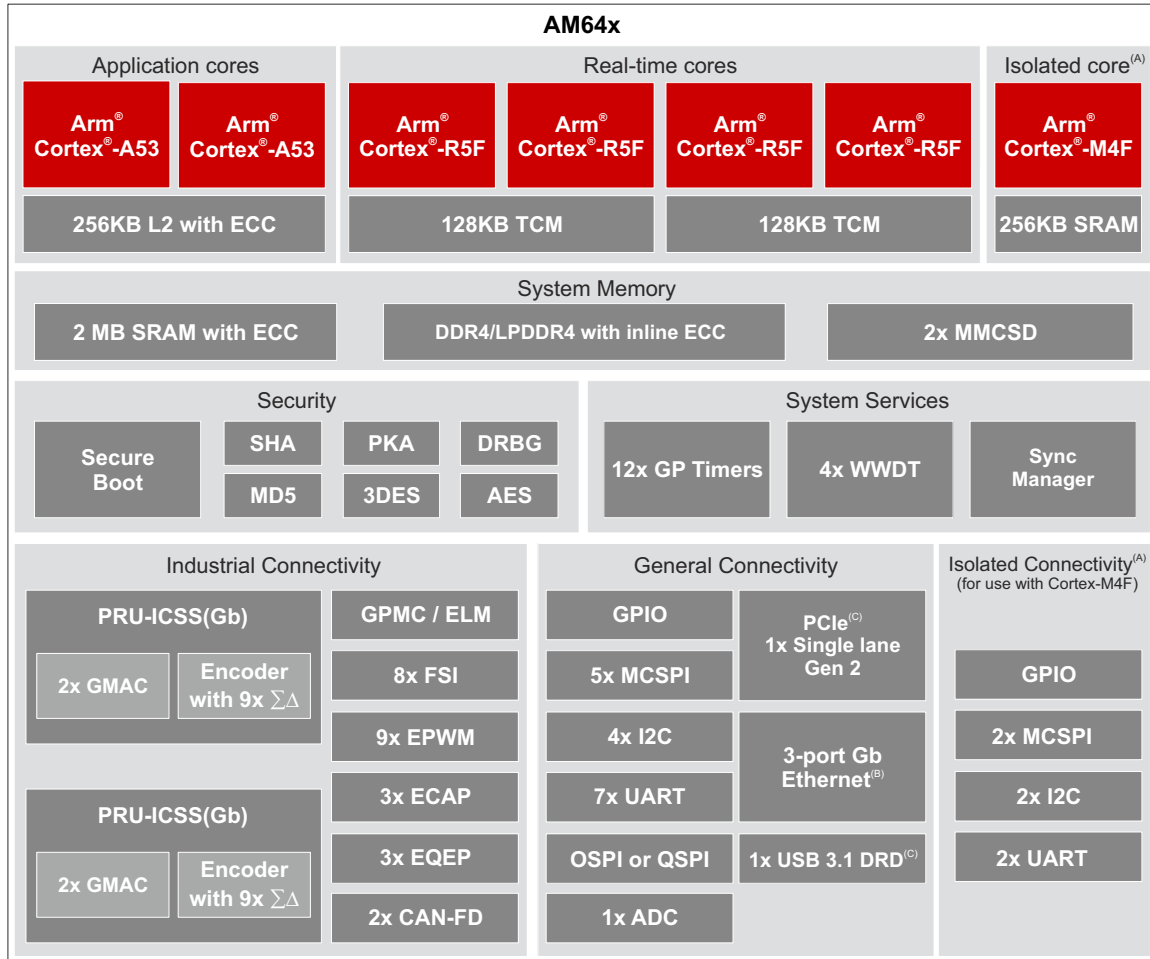
Table 2-1. Sine and Cosine on Arm Cortex-R5F (32-bit float).....	4
Table 2-2. Arctan and Arctan2 on Arm Cortex-R5F (32-bit float).....	5
Table 3-1. Memory Read Latencies.....	7
Table 3-2. Symmetric Cryptography and Secure Hash in Mbit/s.....	8
Table 3-3. Public Key Cryptography Benchmarks.....	8

Trademarks

Sitara™ is a trademark of Texas Instruments.
Cortex® and Arm® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

1 Introduction

The benchmarks were measured on the Cortex[®]-A53 and Cortex-R5F cores. For up to date results, refer to the [Performance Guide](#) and the [Benchmark Demo application](#) in the [Processor SDK for AM64x](#). The benchmarking was produced using the following: [TMDS64GPEVM](#), [SK-AM64X](#), and [LP-AM243](#). The key board parameters for the evaluation were 1GHz clock speed for the Cortex-A53 cores, 800MHz for the Cortex-R5F cores, and a 16-bit wide DDR4 or LPDDR4 at a speed of 1600MT/s. AM64x adds a dual core Cortex-A53 including a 256kB L2 cache, but otherwise the devices are identical. For reference, see [Figure 1-1](#) and [Figure 1-2](#).



- A. Isolation of peripherals and M4F core is an optional feature. MCU domain resources are shared across SoC when in non-isolated configuration.

Figure 1-1. Functional Block Diagram: AM64x

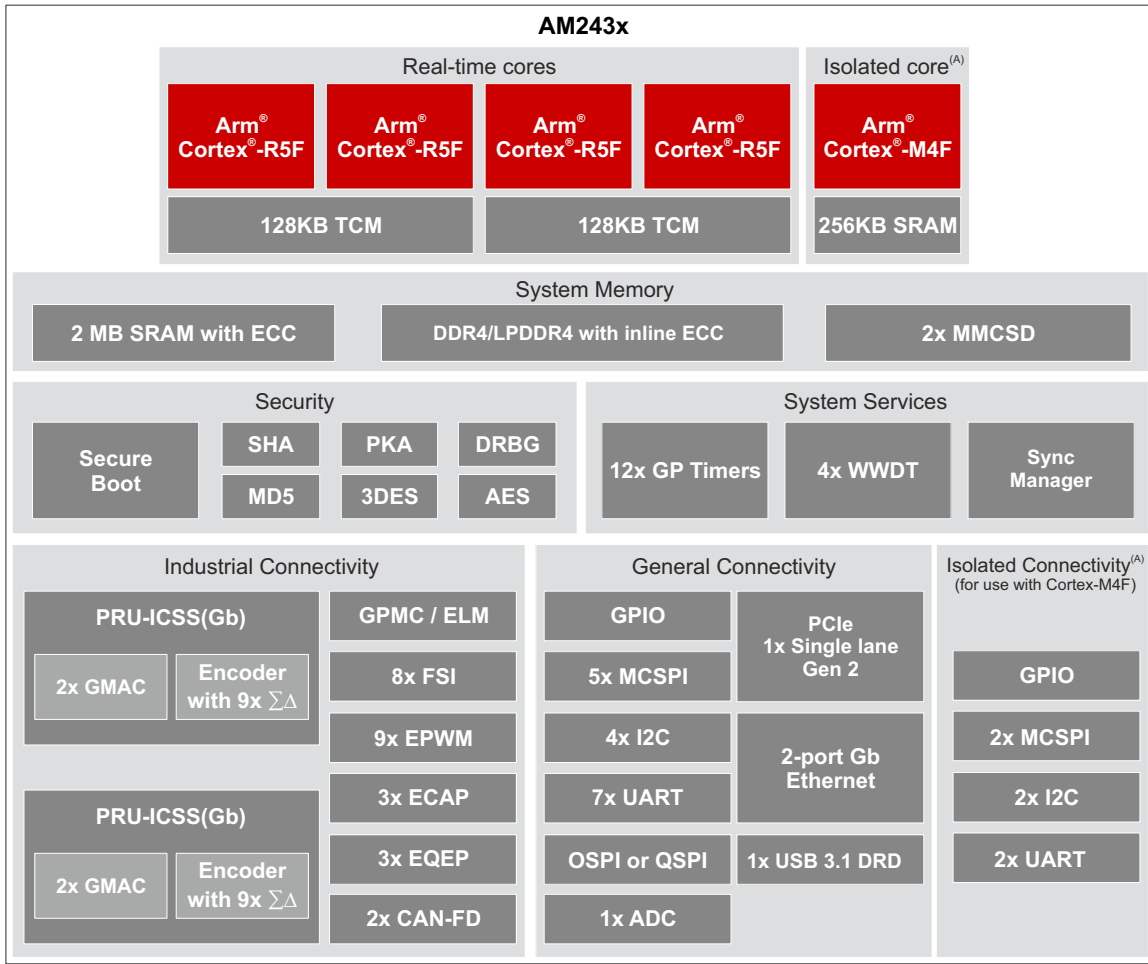


Figure 1-2. Functional Block Diagram: AM243x

2 Processor Core Benchmarks

This section contains the following benchmarks produced by the Arm® Cortex processor core: 1) Dhrystone, a synthetic benchmark, and 2) Trigonometric benchmarks, of functions defined in math.h.

2.1 Dhrystone

Dhrystone is a core only benchmark that runs from warm L1 caches in all modern processors. It scales linearly with clock speed. The score calculated by normalizing the time it takes the benchmark loop to run by the reference 1MIP machine score of 1757. Even though the benchmark was introduced in 1984 by Reinhold P. Weicker, Dhrystone still gets used in embedded processing. It is common to further normalize to DMIPS/MHz/core as the score scales linearly with clock speed. For standard Arm cores, the DMIPS/MHz will be identical with the same compiler and flags. Dhrystone is a single core benchmark, a simple sum of multiple cores running the benchmark in parallel is sometimes used. The aggregate score for AM6442 with two A53 cores at 1GHz (6000DMIPS) and four R5F cores at 800MHz (6400DMIPS) is 12400DMIPS.

	Cortex-A53 (1 GHz)	Cortex-R5F (800 MHz)
Dhrystones	5263158	2,962,962
Normalized Dhrystones (divide by 1757 reference for 1MIP)	2996	1686
DMIPS/MHz each core	3	2.1
Compiler and flags	GCC 9.2 -march=ARMv8 -O3	Arm Compiler 6.14 -mcpu=cortex-r5 -O3
Operating System	Linux 5.10 (2021 LTS)	Bare metal

2.2 Trigonometric Functions

Trigonometric functions are compute functions leveraged in for example motor control. The most commonly used ones are sine, cosine, arctan and arctan2. The standard C library *math.h* has highly accurate implementations of these functions, but the worst case execution time might not be acceptable. For Arm Cortex-R5F, the Arm CMSIS DSP library also has optimized implementations for some trigonometric functions. Sitara™ software development kit [MCU-PLUS-SDK_AM243X](#) and [PROCESSOR-SDK-AM64X](#) contains the CMSIS library (in folder <mcu-plus-install-directoty>/source/cmsis) and further optimized functions for some of the functions in a library called *ti_r5fmath_trig* (in folder <mcu-plus-install-directoty>/examples/motor_control/benchmark_demo/common). [Table 2-1](#) and [Table 2-2](#) show the performance of the most commonly used trigonometric functions using the implementations found in C runtime library (*math.h*), CMSIS, and *ti_r5fmath_trig* library. The implementations trade off accuracy, shown in column max error, and compute time and sometimes additional memory consumption (a lookup table with constants in memory) as shown in column labeled Table size. This table needs to be in TCM or warm L1D cache for the documented performance.

Table 2-1. Sine and Cosine on Arm Cortex-R5F (32-bit float)

Function	Max Cycles	Avg Cycles	Max Error (abs rad)	Table Size	Library
ti_r5fmath_sin()	34	34	7.20E-07	polynomial	TI_R5FMATHLIB
ti_r5fmath_cos()	38	38	2.90E-07	polynomial	TI_R5FMATHLIB
ti_r5fmath_sincos()	51	51	7.20E-07	polynomial	TI_R5FMATHLIB
ti_r5fmath_fast_sincosB()	57	57	1.90E-07	polynomial	TI_R5FMATHLIB
arm_cos_f32()	66	66	1.80E-05	Table 2kbytes	CMSIS
sinf()	71	71	8.40E-08	polynomial	math.h
cosf()	81	81	9.70E-08	polynomial	math.h
arm_sin_cos_f32()	114	114	6.10E-07	Table 2kbytes	CMSIS
sin() (double)	340	296	3.00E-08	polynomial	math.h

Table 2-2. Arctan and Arctan2 on Arm Cortex-R5F (32-bit float)

Function	Max Cycles	Avg Cycles	Max Error (abs rad)	Table Size	Library
ti_r5fmath_atanFast()	45	39	3.76E-03	function	TI_R5FMATHLIB
ti_r5fmath_atan2Fast()	54	54	3.76E-03	function	TI_R5FMATHLIB
ti_r5fmath_atan()	80	68	6.00E-07	poly	TI_R5FMATHLIB
atanf()	111	90	6.80E-08	poly	math.h
ti_r5fmath_atan2()	97	90	6.00E-07	poly	TI_R5FMATHLIB
atan2f()	204	171	2.00E-07	poly	math.h

3 Compute and Memory System Benchmarks

This section contains benchmarks involving the Arm Cortex processor core and the memory system of the SoC. Synthetic benchmarks included are for example LMBench and CoreMark-Pro. Math function benchmarks include functions such linear algebra and fast fourier transforms (FFT).

3.1 Memory Bandwidth and Latency

STREAM and a subset of LMBench are benchmarks to measure achieved memory bandwidth and latency from software.

3.1.1 LMBench

LMBench is a suite of microbenchmarks for processor cores and operating system primitives. The memory bandwidth and latency related tests are most relevant for modern embedded processors. The results will vary a little (< 10%) run to run.

LMBench benchmark *bw_mem* measures achieved memory copy performance. With parameter *cp* it does an array copy and *bcopy* parameter uses the runtime glibc version of *memcpy()* standard function. I practice the glibc uses a highly optimized implementation that utilizes for example SIMD resulting in higher performance. The size parameter equal to or smaller than the cache size at a given level measures the achievable memory bandwidth from software doing a typical for loop or *memcpy()* type operation. Typical use is for external memory bandwidth calculation. The bandwidth is calculated as byte read and written counts as 1, which should be roughly half of STREAM copy result. The table below shows the measured bandwidth and the efficiency compared to theoretical wire rate. The wire rate used is the DDR MT/s rate times the width divided by two (read and write making up a copy both consume the bus). The benchmark further allows creating parallel threads with *-P* parameter. To get the maximum multicore memory bandwidth create the same amount of threads as there are cores available for the operating system, which is 2 for AM64x Linux (*-P 2*).

	Arm Cortex-A53, DDR4-1600MT/s-16 Bit	DDR4 Efficiency	Arm Cortex-A53, LPDDR4-1600MT/s-16bit	LPDDR4 Efficiency
bw_mem -P 2 8M bcopy (dual core, glibc memcpy)	1226MB/s	77%	1100MB/s	69%
bw_mem 8M bcopy (single core, glibc memcpy)	1016MB/s	64%	883MB/s	55%
bw_mem -P 2 8M cp (dual core, inline copy loop)	628MB/s	39%	756MB/s	47%
bw_mem 8M cp (single core, inline copy loop)	528MB/s	33%	599MB/s	37%

LMBench benchmark *lat_mem_rd* is used to measure observed memory access latency for external memory (DDR4/LPDDR4 on AM64x) and cache hits. The two arguments are size of the transaction (64 in the screen shot below) and the stride of the read (512). These two values are selected to measure the latency to caches and external memory not the processor data prefetchers or other speculative execution. For some access patterns the prefetching will work, but this benchmark is most useful to measure the case when it does not. The left column is the size of the data access pattern in megabytes, right column is the round trip read latency in nanoseconds. As a summary for Arm Cortex-A53 read latency to:

- L1D is 3ns
- L2 latency is 14ns
- For access to DDR4-1600 latency is 196ns
- For LPDDR4-1600 the latency is 217ns

The below is a run with DDR4, for LPDDR4 the result is the same for L1D and L2 sizes but slightly higher (217ns) for the largest sizes.

```

root@am6x-evm:~# lat_mem_rd 64 512
"stride=512
0.00049 3.006
0.00098 3.006
0.00195 3.006
0.00293 3.006
0.00391 3.006
0.00586 3.006
0.00781 3.006
0.01172 3.006
0.01562 3.006
0.02344 3.009
0.03125 3.120
0.04688 9.212
0.06250 10.677
0.09375 12.269
0.12500 12.984
0.18750 13.651
0.25000 14.066
0.37500 115.226
0.50000 168.747
0.75000 189.919
1.00000 192.138
1.50000 193.431
2.00000 194.175
3.00000 194.870
4.00000 195.202
6.00000 195.463
8.00000 195.622
12.00000 195.700
16.00000 195.761
24.00000 195.876
32.00000 195.938
48.00000 196.001
64.00000 196.006
    
```

3.1.2 STREAM

STREAM is a microbenchmark for measuring data memory system performance without any data reuse. It is designed to miss on caches and exercise data prefetcher and speculative accesses. it uses double precision floating point (64 bit) but in most modern processors the memory access will be the bottleneck. The four individual scores are copy, scale as in multiply by constant, add two numbers, and triad for multiply accumulate. For bandwidth, a byte read counts as one and a byte written counts as one resulting in a score that is double the bandwidth LMBench will show. The table below shows the measured bandwidth and the efficiency compared to theoretical wire rate. The wire rate used is the DDR MT/s rate times the width. To get overall maximum achieved throughput the command used is *stream -M 16M -P 2 -N 10*, which means two parallel threads and 10 iterations.

	DDR4-1600MT/s-16-Bit Bandwidth	DDR4-1600MT/s-16-Bit Efficiency	LPDDR4-1600MT/s-16-Bit Bandwidth	LPDDR4-1600MT/s-16-Bit Efficiency
copy	2482MB/s	78%	2221MB/s	69%
scale	2516MB/s	79%	2268MB/s	71%
add	2350MB/s	73%	2130MB/s	67%
triad	2355MB/s	74%	2139MB/s	67%

3.1.3 Cortex-R5 Memory Access Latency

Table 3-1 documents measures read latencies from bare-metal software on the Cortex-R5F core to various destinations with the R5 running at 800 MHz.

Table 3-1. Memory Read Latencies

Destination	Latency in Nanoseconds
Local TCM	1.25
TCM of another Cortex-R5F	77.5
Shared on-chip memory (MSRAM)	63.75
ICSS DMEM	127.5
External DDR4	280
External OSPI	496.25
GPMC (Parallel Interface boundary)	271.25

3.2 CoreMark®-Pro

CoreMark-Pro tests the entire processor, adding comprehensive support for multicore technology, a combination of integer and floating-point workloads, and data sets for utilizing larger memory subsystems. The components of CoreMark-Pro utilizes all levels of cache with an up to 3MB data memory footprint. Many but not all of the tests also are using pthreads to allow utilization of multiple cores. The score scales with the number of cores but is always less than linear (dual core score is less than 2x single core).

CoreMark-Pro should not be confused with the smaller CoreMark which, like Dhrystone, is a microbenchmark contained in L1 caches of a modern processor.

	Arm Cortex-A53
Single core	604
Dual core	1063 (1.66 times single core)

3.3 Fast Fourier Transform

Fast Fourier Transform (FFT) is a multiply accumulate heavy building block in many applications. Below table shows a 1024-point single precision floating point complex FFT execution time. The Arm Cortex-A53 benchmark uses the implementation from Ne10 library which leverages the Advanced SIMD or NEON acceleration of Cortex A53.

	1024pt float CFFT execution time (single thread / core)
Arm Cortex-A53 (1GHz)	27 microseconds
Arm Cortex-R5 (800MHz)	134 microseconds

3.4 Cryptographic Benchmarks

The AM64x Linux SDK includes openssl cryptographic library that can be used by applications and is used by for example some HTTPS, ssh, and netconf implementations to get access to optimized implementation of cryptographic functions. For the highest performance the higher level interface provided by the EVP library should be used. Table 3-2 shows a set of selected benchmarks of software observed performance run on AM64x. Command run was `openssl speed -elapsed -evp <cryptographic mode> -multi 2`. This is utilizing both A53 cores using two threads.

Table 3-2. Symmetric Cryptography and Secure Hash in Mbit/s

	Frame Size (bytes)					
	16	64	256	1024	8192	16384
aes-128-gcm	855	2438	4671	6004	6656	6637
aes-256-gcm	811	2256	4057	5241	5624	5658
aes-128-ctr	87	190	725	2446	7513	8836
sha256	559	1672	3865	5812	6860	6866
sha512	153	614	976	1390	1584	1617
chacha20-poly1305	494	1067	2091	2380	2541	2540

Further benchmarks for public key cryptography are shown in [Table 3-3](#). Test can be run with command `openssl speed -elapsed <algorithm> -multi 2`.

Table 3-3. Public Key Cryptography Benchmarks

RSA	size	512	1024	2048	3072	4096
	sign/second	5254	1174	181	59	21
	verify/second	67996	23579	6777	3138	1523
ECDSA	curve	nistp224	nistp256	nistp521	nistk233	nistb233
	sign/second	310	1074	71	241	237
	verify/second	501	2717	103	130	128

4 Application Benchmarks

This section contains information and examples for the following application level benchmarks: 1) machine learning inference, 2) field oriented control (FOC) loop for motor drive control, 3) Peripheral Component Interconnect Express (PCIe) to DDR transfer bandwidth using BCDMA and 4) DDR to DDR transfer bandwidth using BCDMA.

4.1 Machine Learning Inference

AM64x SDK has integrated open source TensorFlow Lite for deep learning inference at the edge. AM64x is not specifically targeting real-time image processing but is can execute machine learning inference for some edge applications. As examples below are to runs of TensorFlow Lite models for image classification (224x224 pixels 3 bytes for colors) based on imagenet database and 1000 object classes. The example image of Rear Admiral Grace Hopper is installed in the file system ([available at](#)). The example `label_image` program will crop and resize the bmp image to the 224 x 224 pixels before calling the TensorFlow Lite. The inference time benchmark for the quantization aware trained Mobilenetv1 network (`mobilenet_v1_1.0_224_quant.tflite`) is 280 milliseconds. The example run is shown below can also be found in the [AM64x Linux SDK](#) (in folder `/usr/share/tensorflow-lite/examples`):

```
root@am6x-evm:/usr/share/tensorflow-lite-1.15/examples# ./label_image -i grace_hopper.bmp -l
labels.txt -m mobilenet_v1_1.0_224_quant.tflite
Loaded model mobilenet_v1_1.0_224_quant.tflite
resolved reporter
invoked
average time: 280.587 ms
0.780392: 653 military uniform
0.105882: 907 windsor tie
0.0156863: 458 bow tie
0.0117647: 466 bulletproof vest
0.00784314: 835 suit
```


The inference time for the floating point Mobilenetv2 model based inference of an image of the exact same resolution (224 x 224 x 3) is 362 milliseconds. The console command and printout is shown below:

```

root@am6x-evm:/usr/share/tensorflow-lite-1.15/examples# ./label_image -i grace_hopper.bmp -l
labels.txt -mtest/mobilenet_v2_1.0_224.tflite
Loaded model test/mobilenet_v2_1.0_224.tflite
resolved reporter
invoked
average time: 362.22 ms
0.911345: 653 military uniform
0.014466: 835 suit
0.0062473: 440 bearskin
0.00296661: 907 windsor tie
0.00269019: 753 racket
root@am6x-evm:/usr/share/tensorflow-lite-1.15/examples#
    
```

The numbers printed in the console below the inference time are the top-5 classification results as a number between 0 and 1 and the class out of the 1000 in imagenet labels.txt. The accuracy result is a benchmark of the model and input image, not the device running the inference.

All .tflite models will run on AM64x, a quantized Mobilenetv1 and floating point Mobilenetv2 were chosen as common benchmarks that can be used to interpolate the performance of an inference application. The quantized Mobilenetv1 is in the file system, the Mobilenetv2 was downloaded from Hosted models [hosted models at tensorflow.org](https://www.tensorflow.org).

4.2 Field Oriented Control (FOC) Loop

The MCU+ and Linux SDKs contain a benchmark demo application that includes an example FOC loop implementing the algorithm shown in Figure 4-1 and displaying the performance of the loop. Functions in the include Clark, Park, inverse Park, Sin, Cos, PI controllers and space vector generation.

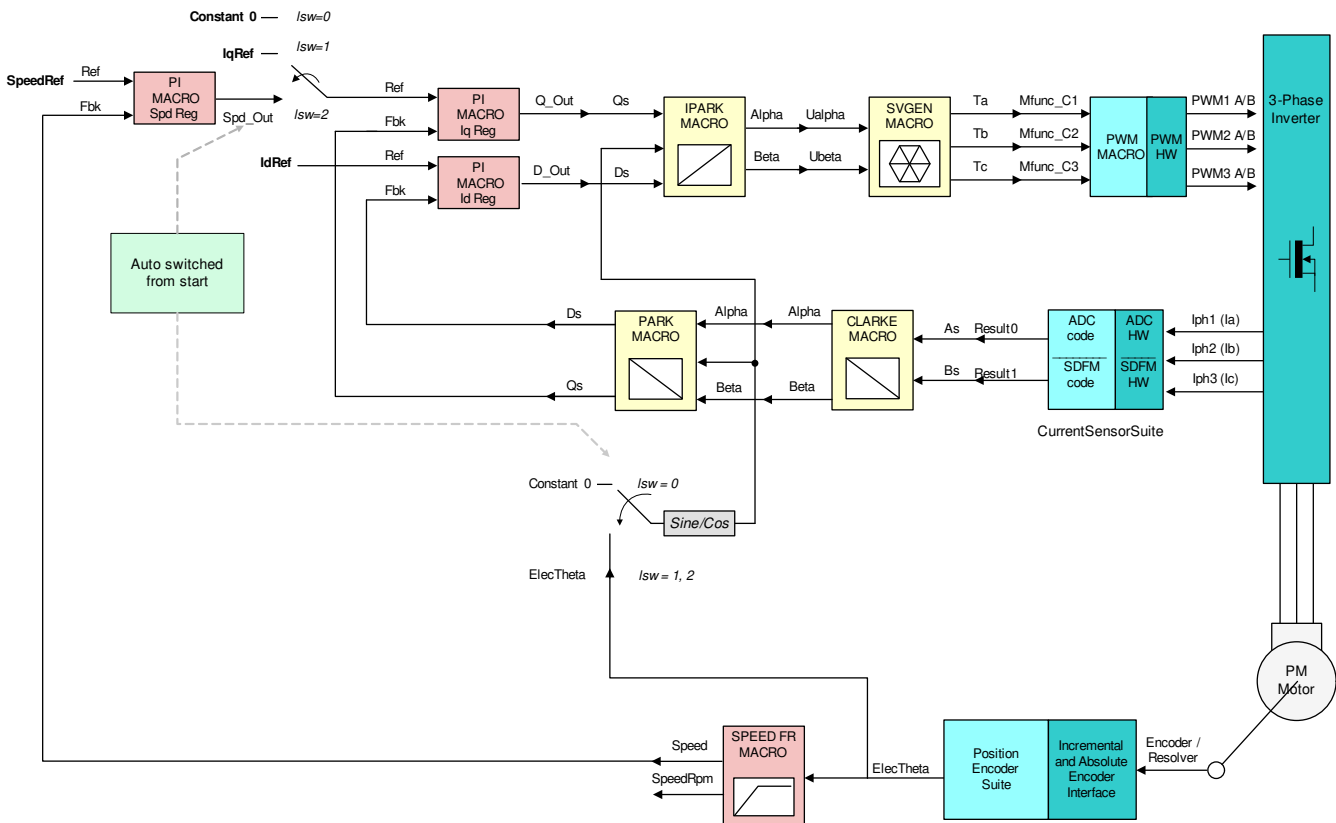


Figure 4-1. Speed Closed Field Oriented Control Loop

The performance of the FOC loop on a single Arm Cortex -R5F core of AM64x is shown in the table below. Each of the up to four Arm Cortex-R5F core can independently achieve this performance for the motor it is controlling.

	FOC Loop Execution Time (microseconds)
Average	0.4

4.3 PCIE to DDR Performance Using BCDMA

AM64x MCU+ SDK contains demo applications for PCIE enablement. It includes sample to code for PCIE benchmark for PCI-End-Point (EP) and PCIE-Root-Complex(RC) pairing of AM64x

- [PCIE buffer transfer EP](#)
- [PCIE buffer transfer RC](#)

Similar way we have DMA examples in MCU+ SDK. UDMA is the generalised name for the DMA interface, for AM64x and AM243x it uses BCDMA underneath.

- [UDMA Memcpy Polling](#)
- [UDMA Memcpy Interrupt](#)

We have used these as a baseline measurements and optimal settings to benchmark minimum read latency and bandwidth of EP-DDR location to RC-DDR location using BCDMA multi-channels over PCIE. PktDMA (used by for example Ethernet) should behave in a very similar manner. Also with AM243x/AM64x either the RC or the EP can initiate the transactions.

4.3.1 Test Setup

AM64x-PCIE Benchmarking Setup

Here is the setup we used for benchmarking PCIE-BCDMA-DDR read performance (in terms of bandwidth).

Hardware Details:

- [TMDS64EVM](#)
- PCIE cable can be obtained from [Adex Electronics](#).

For more details, see:

- [PCIE buffer transfer RC](#)

Software Details:

- Test code is available at GitHub [Link](#) which is [MCU+ SDK 8.6](#) baselined.

Note

By default in MCU+ SDK, a maximum 4 UDMA channels can be assigned to the R5F_0_0 core.

To run the test with more channels than the default configuration, you need to first use the resource management ([RM](#)) tool to manage UDMA channel allocation to the cores.



Figure 4-2. AM64x-PCIE Benchmarking Setup

Here are the details about data flow between 2-AM64x board during PCIE benchmarking.

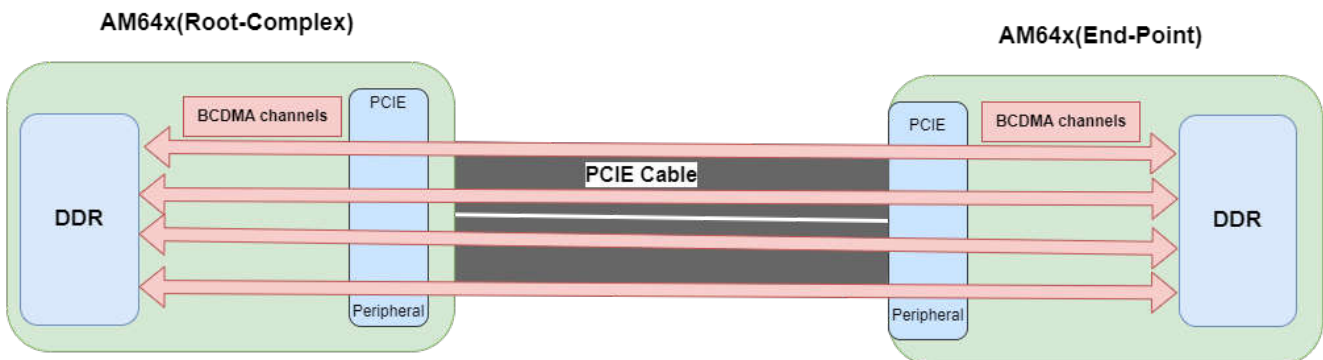


Figure 4-3. AM64x (PCIE benchmarking) Data Flow

EP sets the buffer in PCIE and RC will read and copy it into DDR

For example, on RC side:

PCIE_loc_1 = 0x68000000UL + 0x00000000U

PCIE_loc_2 = 0x68000000UL + 0x01000000U

PCIE_loc_3 = 0x68000000UL + 0x02000000U

PCIE_loc_4 = 0x68000000UL + 0x03000000U

and

DDR_loc_1 = 0xA0000000 + 0x00000000U

DDR_loc_2 = 0xA0000000 + 0x01000000U

DDR_loc_3 = 0xA0000000 + 0x02000000U

DDR_loc_4 = 0xA0000000 + 0x03000000U

And so on.

4.3.2 Result and Observation

This section summarized the results and observation of test performed.

(Bytes)	PCIE to DDR Read BW (Mbits/sec)				16 BCDMA Channels
	1 BCDMA Channel	2 BCDMA Channels	4 BCDMA Channels	8 BCDMA Channels	
1	0.89	1.6	2.29	2.67	2.61
2	3.2	4.57	7.11	6.4	6.4
4	6.4	9.14	14.2	12.8	13.13
8	12.8	21.33	23.27	26.95	26.95
16	32	42.67	56.89	51.2	53.89
32	51.2	85.33	102.4	102.4	105.02
64	102.4	17.67	204.8	204.8	210.05
128	204.8	341.33	455.11	409.6	420.10
256	292.57	512	682.67	963.76	862.31
512	455.11	744.72	1170.28	1489.45	1560.38
1024	585.14	1092.26	1489.45	1771.24	1820.44
2048	712.35	1310.72	1872.46	2048	2131.25
4096	780.19	1456.35	2048	2148.72	2166.48
8192	840.20	1618.17	2202.89	2240.55	2202.89
16384	856.68	1638.4	2259.86	2289.47	2205.21
32768	868.03	1664.40	2309.64	2314.73	2208.69

Figure 4-4 shows the PCIe to DDR read performance curve using BCDMA channels. Here, 1 BCDAM channel is being referred to as 1 TRPD. Data shown here range from 40B to 1600B for clear visualization purpose.

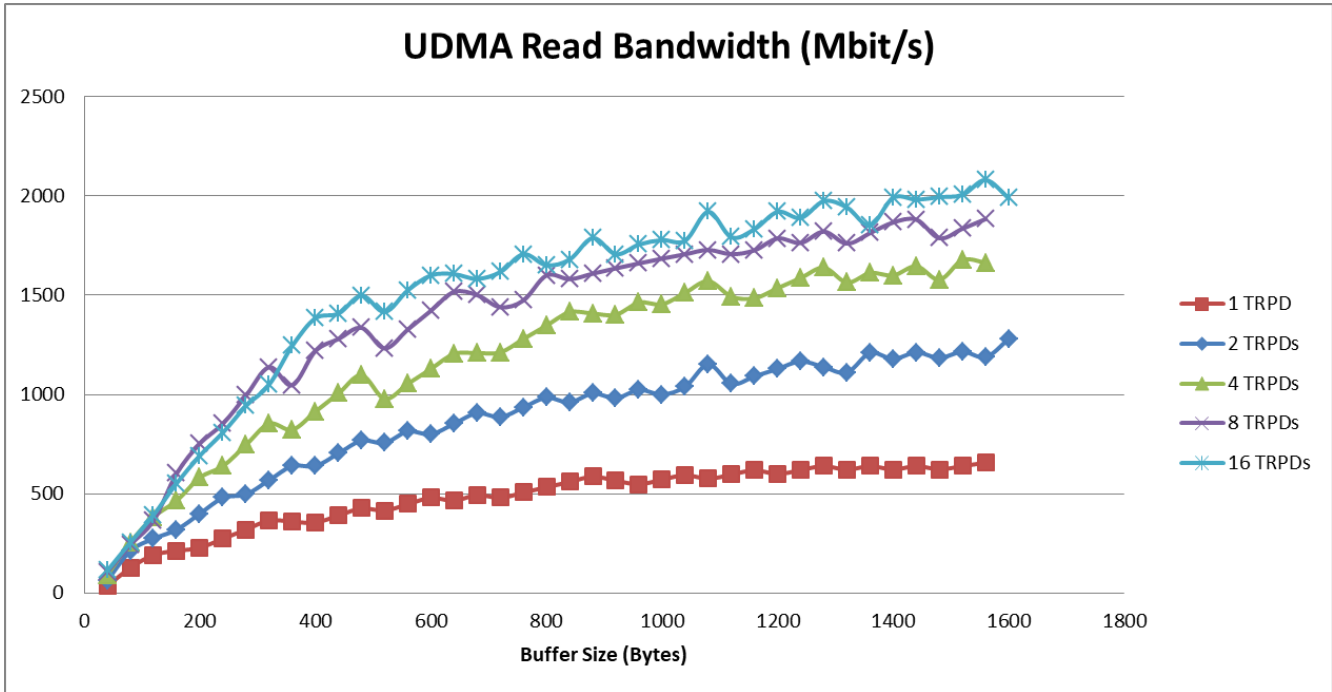


Figure 4-4. PCIe Performance Graph

Observation:

Maximum throughput is achieved with four parallel DMA transfers, further adding of channels will work but will not increase throughput.

4.4 DDR to DDR Performance Using BCDMA

AM64x MCU+ SDK contains UDMA-based examples.

- [UDMA Memcpy Polling](#)
- [UDMA Memcpy Interrupt](#)

Here we have used these to measure transfer performance (in terms of bandwidth) from DDR to DDR using BCDMA and multiple channels in parallel.

Hardware Details:

- [TMD64EVM](#)

Software Details:

- Test code used for this benchmarking is available at GitHub Link ([polling](#) and [interrupt](#)), which is [MCU+ SDK 8.6](#) baselined.

Note

By default in MCU+ SDK, maximum 4 UDMA channels can be assigned to R5F_0_0 core.

To test more channels than the default configuration:

- Use the resource management ([RM](#)) tool to manage UDMA channel allocation to the cores.
- Add number of UDMA channels, using "example.syscfg" file in the imported CCS project

4.4.1 Test Setup

This section describes test setup used for benchmarking performed for DDR to DDR copy using UDMA.

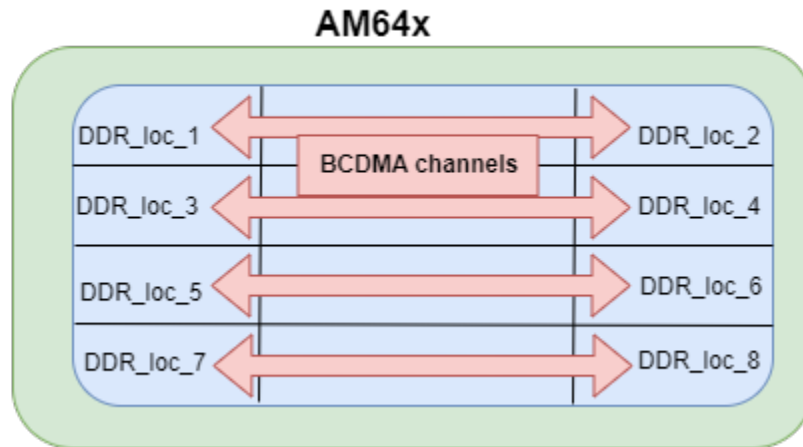


Figure 4-5. AM64x DDR to DDR Data Flow

For example:

DDR_loc_1 = 0xA0000000 + 0x00000000U

DDR_loc_2 = 0xA0000000 + 0x01000000U

DDR_loc_3 = 0xA0000000 + 0x02000000U

DDR_loc_4 = 0xA0000000 + 0x03000000U

And so on.

4.4.2 Result and Observation

This section provides test results and observation for DDR to DDR data copy using UDMA channels.

Buffer Size (Bytes)	DDR to DDR (UDMA) Transfer BW (Mbits/sec)				
	1 BCDMA Channel	2 BCDMA Channels	4 BCDMA Channels	8 BCDMA Channels	16 BCDMA Channels
1	1.143	1.6	2.13	2.06	1.80
2	5.33	6.4	5.33	4.74	3.88
4	8	12.8	10.67	9.85	7.88
8	16	21.33	23.27	18.96	15.51
16	42.67	42.67	46.54	39.38	31.51
32	85.33	85.33	93.09	75.85	63.01
64	170.67	204.8	170.67	157.54	124.12
128	256	341.33	341.33	341.33	282.48
256	512	585.14	630.15	655.36	555.39
512	1024	1170.28	1489.45	1424.69	1285.02
1024	1638.4	2048	2730.67	2520.61	2520.61
2048	2340.57	2978.91	3640.89	4369.07	3236.34
4096	3276.8	4369.07	4681.14	5041.23	3615.78
8192	4096	5698.79	4946.11	4946.11	4017.53
16384	4681.14	6898.52	5140.08	5190.97	4136.39
32768	4946.11	7710.12	5322.72	5165.40	4181.76

Figure 4-6 shows the DDR to DDR read performance curve using BCDMA channels.

The 1 BCDMA channel is being referred to as 1 TRPD.

Data shown here range from 40B to 1600B for clear visualization purpose.

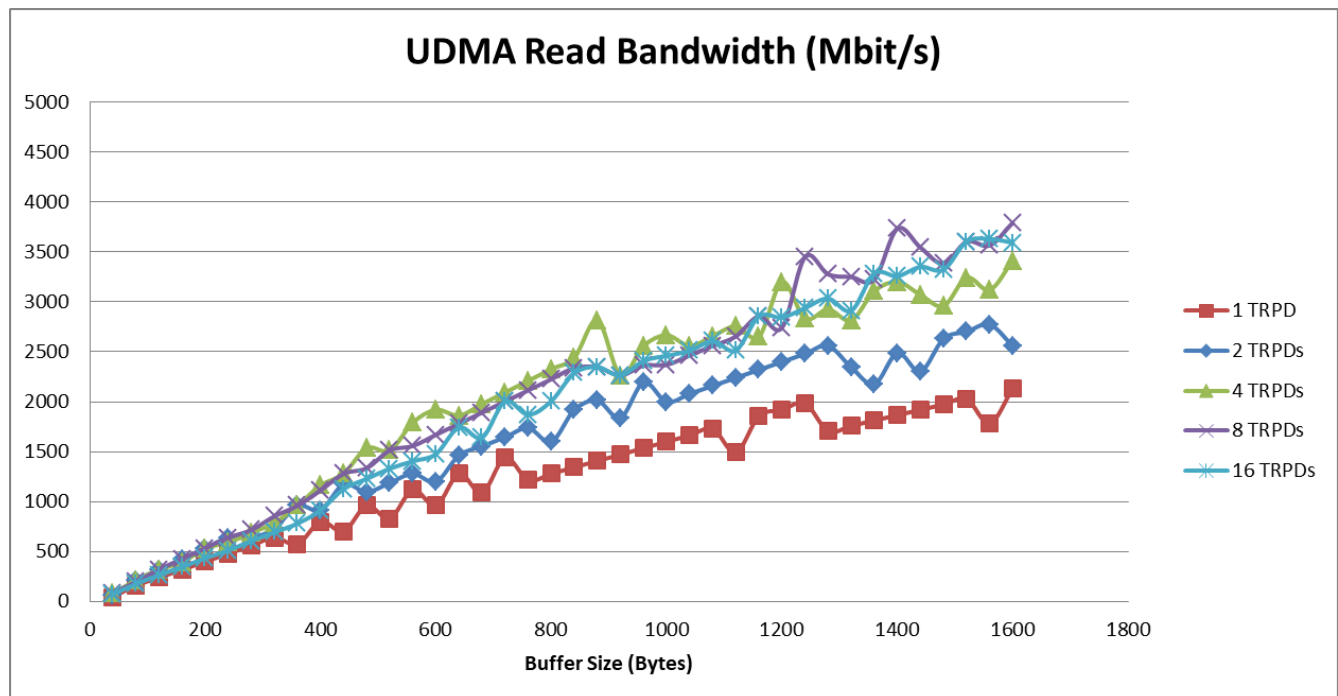


Figure 4-6. DDR Performance Graph

Observation:

Maximum throughput is achieved with 4 parallel DMA transfers, further adding of channels will work but will not increase throughput.

5 References

- [CoreMark-Pro](#)
- STREAM McCalpin, John D.: "STREAM: Sustainable Memory Bandwidth in High Performance Computers", a continually updated technical report (1991-2007), available at: <http://www.cs.virginia.edu/stream/>
- [Ne10 math library](#)
- [CMSIS library](#)
- [hosted models at tensorflow.org](#)
- [OpenSSL](#)

6 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (March 2023) to Revision B (January 2024)	Page
• Added Section 4.3	10
• Added Section 4.3.1	10
• Added Section 4.3.2	12

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2024, Texas Instruments Incorporated