Nallatech

White Paper

OpenCL

ALTERA®

# 40Gbit AES Encryption Using OpenCL and FPGAs

## Introduction

Altera's launch of OpenCL support for FPGA systems has ushered in a new era in high performance computing using CPUs and FPGAs in a hybrid computing model. Altera's OpenCL Compiler (ACL) support for FPGA cards:

- Gives programmers easy access to the power of FPGA computing.
- Offers significantly higher performance at much lower power than is available using other technologies.
- Provides significant time-to-market advantage compared to traditional FPGA development using a hardware description languages.
- Automatically abstract details of hardware design for designers.

Implementing FPGA designs with the OpenCL compiler allows a designer to easily offload parts of their algorithm to the FPGA to increase performance, lower power and improve productivity.

This parallel programming methodology uses a kernel approach where data is passed to the specified kernel or processing. The kernel code uses C language with a minimal set of extensions that allows parts of the application code or sub routines to take advantage of parallel performance by processing via the FPGA.

This application note illustrates how to perform AES encryption on FPGAs using the OpenCL tool flow.

## Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is a symmetric-key encryption standard that has been adopted by the U.S. government. AES can have block ciphers of 128, 192 and 256 bits in width, all of which require data in 128 bit blocks.

The AES algorithm consists of multiple bit shifts and Exclusive Or (XOR) operations that make it an ideal candidate for acceleration on FPGAs.

## AES Operations

AES operates on a 4×4 array of bytes, termed the state (different versions of AES with a larger block size have additional columns in the state). AES consists of four distinct processing stages, as listed below:

1. **Key Expansion** - The round keys are derived from the cipher key using the Rijndael's key schedule.
2. **Initial Round** :
    a. Add Round Key: Each byte of the state is combined with the round key using a bitwise XOR.
3. **Rounds**
    a. Sub Bytes: A non-linear substitution step where each byte is replaced with another using a lookup table.
    b. Shift Rows: A transposition step where each row of the state is shifted cyclically a certain number of steps.

c. Mix Columns: A mixing operator which operators on the columns of the state, combining the four bytes in each column.
d. Add Round Key

4. Final Round
   a. Sub Bytes
   b. Shift Rows
   c. Add Round Key

In this implementation the host processor performs the key expansion and the results are passed to the AES encryption algorithm on the FPGA. The key schedule process varies infrequently, depending on session key changes, so there is not significant performance impact with this approach.

## ECB and CTR Ciphers

Electronic Codebook (ECB) is the simplest cipher mode to program on an FPGA. It is easily replicated multiple times and can be pipelined as the output has no effect on the next result. ECB can be seen in Figure 1.

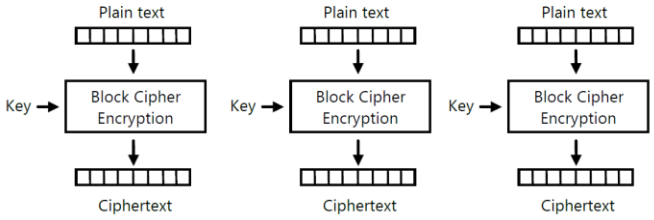


Figure 1. Electronic Codebook (ECB) mode encryption

The downside of ECB is identical plaintext blocks are encrypted in the same way which does not hide patterns in the data.

A better approach is to use a Counter (CTR) approach, as seen in Figure 2. Here a counter is encrypted, incremented and XORed with the plaintext to create the output ciphertext.

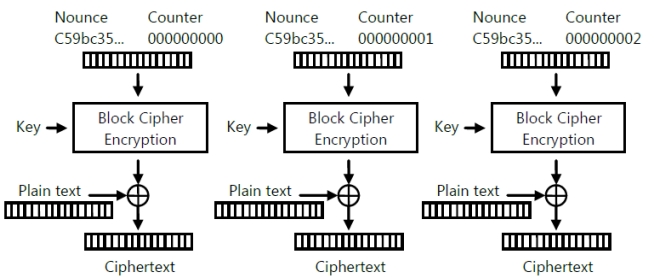


Figure 2. Counter (CTR) mode encryption

The increment of the counter is arbitrary with the most common being a simple count. An added bonus of the CTR method is encryption and decryption logic are identical (Figure 3). The counter is simply reset before decrypting.

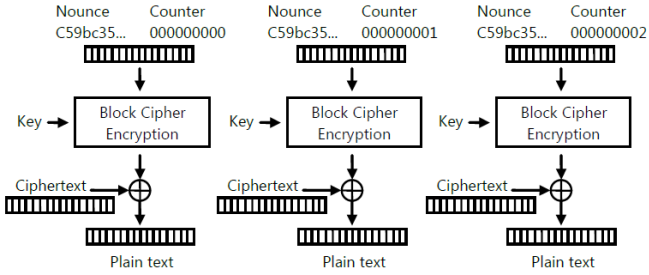


Figure 3. Counter (CTR) mode decryption

CTR encryption also allows consecutive blocks of data to be encrypted in parallel by including a stride pattern into the counter.

## OpenCL for FPGAs

The FPGA is effectively a 'blank canvas' on which a user can design an architecture fit for purpose.

When an OpenCL kernel is compiled using the ACL compiler, a processing architecture is designed around the needs of the algorithm. This includes integer and floating point logic built to the required depth and accuracy. The memory architecture is designed to meet the needs of the algorithm by utilizing the many hundreds of individually accessible memories available on the FPGA fabric.

Altera's OpenCL compiler compiles OpenCL kernels, which in turn are compiled via Altera's Quartus tools to create a SRAM object file (SOF). This SOF file can then be downloaded onto the FPGA. The OpenCL API for the host CPU allows OpenCL commands for controlling the compiled kernel source.

## Describing AES encryption using OpenCL

The following pseudo code in Figure 4 describes the processing stages required for AES encryption.

The nature of the AES encryption algorithm allows entire code to be unrolled into a single very deep pipeline containing thousands of integer operations. The ACL compiler has #pragma directives that can be added to a users OpenCL code to instruct nested loops to be unrolled, allowing the full AES code to be flattened. Only a few small number of changes to the original OpenCL source code are required.

Targeting a GPU with this modified source code is still possible as the #pragma directives are simply ignored. This allows the OpenCL code to be functionally verified quickly using a CPU or GPU prior to compilation to the required SOF file.

```
//Add the First round key to the state
//before starting the rounds

    AddRoundKey(0);

//Next there will be Nr rounds. The
//first Nr-1 rounds are identical and
//are executed in the loop below

    for(round=1;round<Nr;round++)
    {
        SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(round);
    }

//The last round is given below. The
//MixColumns function is not
//implemented in the last round

    SubBytes();
    ShiftRows();
    AddRoundKey(Nr);
```

Figure 4. Counter (CTR) mode decryption

Once the SOF file is built the OpenCL kernel is downloaded onto the FPGA using the Altera Quartus tools. Just-in-time compilation of the kernel is not permitted due to the FPGA compile times, therefore kernels are loaded using the clCreateProgramWithBinary method.

Altera also provides an OpenCL API library for the host to allow communication to support the FPGA accelerator card.

## Target Technology


Figure 5. PCIe-385N Stratix V FPGA Accelerator

Nallatech's PCIe-385N accelerator card is supported by the ACL compiler and host OpenCL API. Figure 5 shows a top down view of the PCIe-385N. To target the Nallatech card the OpenCL kernel is compiled using the relevant compiler switch to target the 385.

The PCIe-385N features an 8-lane PCI Express Gen 3 capable interface for high speed host communications. The card also has 2 independent banks of DDR3 memory, totaling 16GBytes, coupled to the Stratix V.

## AES Performance

The OpenCL FPGA program methodology allows a programmer to pick the number of "work-groups" that best fit the desired performance, whether this be as much as possible or tailored for a particular throughput. In this case the goal was to encrypt 40 Gbit Ethernet data which equates to a throughput of 5 GBytes per second. Note that although the PCIe-385N has a 10Gbit connection, the data is to be generated internally for testing purposes.

```
Kernel throughput analysis for        : AESEncrypt
. vector lanes                                : 1
. num copies                                  : 1
. kernel work-group limit                     : 2
. local work-group limits                     : 2
. throughput due to control flow analysis : 250.00
  M workitems/second
. best case throughput FLOP analysis      : 0.00
  MFLOPS/s
. kernel global memory bandwidth analysis: 8421.05
  MB/second
. kernel number of local memory banks     : 4
. total # of RAMs used (local mem) / copy : 28
. kernel number of local memory masters   : 16
. kernel local avg port fanin             : 1.0
.. reducing fmax because of connectivity : 240.9
   MHz
.. local stalls derate throughput by      : 1.00
   for [ AESEncrypt.wii_blk ]
.. local stalls derate throughput by      : 1.00
   for [ entry ]
.... because of local stalls mflops =  : 0.00
     MFLOPS/s
.... because of local stalls throughput: 240.91 M
     workitems/second
```

Figure 6. ACL Output for single kernel

Compilation of the AES algorithm targeting a single work group yields a predicted throughput of 240 million work items/second. Each work item is a 16 Byte word giving a throughput of 3.8 GBytes/second. This would be insufficient to encrypt 40 Gbit data. Figure 6 shows the ACL output for a single AES kernel on the FPGA.

Fortunately FPGAs are particularly efficient at integer arithmetic allowing more than one work-group to fit within the FPGA. Targeting

multiple work-groups is done using a simple kernel attribute "__attribute((num_copies(n)))". To achieve the desired 40 Gbit data rate only 2 copies of the AES kernel were required. Figure 7 shows the ACL output for a single AES kernel on the FPGA.

```
Kernel throughput analysis for        : AESEncrypt
. vector lanes                          : 1
. num copies                            : 2
. kernel work-group limit               : 2
. local work-group limits               : 2
. throughput due to control flow analysis : 500.00
  M workitems/second
. best case throughput FLOP analysis    : 0.00
  MFLOPS/s
. kernel global memory bandwidth analysis :
  16842.11 MB/second
. kernel number of local memory banks   : 4
. total # of RAMs used (local mem) / copy : 28
. kernel number of local memory masters : 16
. kernel local avg port fanin           : 1.0
.. reducing fmax because of connectivity : 240.9
   MHz
.. local stalls derate throughput by    : 1.00
   for [ AESEncrypt.wii_blk ]
.. local stalls derate throughput by    : 1.00
   for [ entry ]
.... because of local stalls mflops =   : 0.00
   MFLOPS/s
.... because of local stalls throughput: 481.82
   M workitems/second
```

Figure 7. ACL Output for two AES kernels

## Device Utilisation

How much an algorithm uses of the FPGA is an important aspect of FPGA programming. An algorithm only utilizes the logic it requires within the FPGA, leaving the remaining logic unused and consuming minimal power. Therefore large power savings can be achieved by designing a kernel to meet only the needs of the target system, something that is not possible on CPU and GPU technologies.

There is no point designing an FPGA OpenCL kernel to run 100x faster, using say 100% of the device, when Amdahl's law suggests only a maximum system increase of 10x is possible.

## Performance

To measure the performance improvement the same OpenCL source code was compiled and ran on an AMD Radeon HD 7970 GPU card. This device has 2048 stream processors and an engine clock speed of 925 MHz. The FPGA design has 2 dedicated AES streams and a clock speed of only 170 MHz. The complexity of the AES encryption and the interdependency of the data results in a modest peak performance of ~0.33 GBytes/Sec throughput on this GPU. The FPGA AES streams are able to encrypt a full 16 Byte block every clock cycle to achieve 5.2 GBytes/Sec throughput. All performance figures reflect the kernel processing time only.

The power consumption of the FPGA accelerator is also significantly lower, requiring approximately 25 Watts compared to several hundred Watts on the GPU.

The AES source code was also compiled onto a 2GHz Intel Xeon E5503 processor achieving a performance of ~0.01 GBytes/sec per thread. The low throughput reflects upon the thousands of operators required for each 16 Byte output of the AES calculation and the limited parallel processing available to the CPU.

| Technology | Throughput (GBytes/Sec) |
|---|---|
| E5503 Xeon Processor | 0.01 (Single core) |
| AMD Radeon HD 7970 | 0.33 |
| PCIe385 FPGA Accelerator | 5.20 |

Figure 8. Performance results for various technologies

## Conclusion

To achieve 40 Gbits/second throughput for the AES encryption described here, only 42 % of the Stratix A7 FPGA device was utilized. The remainder could be left unused for power savings or extra kernels could be placed in parallel to the encryption core.

Altera's ACL compiler allows easy access to FPGA accelerator technology. For the first time, utilizing OpenCL, code is truly portable between CPU, GPU and FPGA technologies. AES encryption is a new class of algorithm that can now be tackled using the OpenCL language which was not possible to perform efficiently on traditional compute platforms.