Lab B

Ethash

陳炫均

Introduction to Ethash

- Ethash is the name of the hashing algorithm that is at the core of all Ethereum transactions.
- This algorithm is used as a proof-of-work (PoW) that a substantial amount of distributed effort when towards the creation of a transaction (i.e. a new block in the chain).

Application Project on Vitis

- pow.cpp: the main application file and represents the host code. It's responsible for initializing the OpenCL framework and loading/executing the acceleration kernel.
- pow.h: a header file for the application. Includes the required OpenCL header.
- krnl_ethash.cl: actual kernel which will get run on the targeted FPGA.
- xcl.*: helper functions.

libethash/internal.c

Where the Ethash library and function would be called upon:

```
static void ethash_hash(
        ethash_return_value *ret,
        node const *full_nodes,
        ethash_cache const *cache,
        ethash_params const *params,
        const uint8_t header_hash[32],
        const uint64_t nonce) {
...
}
```

libethash/internal.c

Simply copy this code and use it to replace the kernel code in our krnl_ethash.cl file would cause a flood of compilation errors including:

- The code makes extensive use of macros and helper functions that are defined in separate libethash files.
- Some standard C coding practices aren't necessarily acceptable to OpenCL (e.g. memcpy).

libethash/internal.c

Make the following adaptations:

- Add all dependent macros and helper functions to the kernel *.cl file.
- Replace all instances of memcpy as well as any datatypes not natively supported by OpenCL.
- Update the host *.cpp code to accommodate appropriate arguments.

PoW Kernel File

- ethash_return_value: returns two hash values so the original code used a special structure to hold these. In the OpenCL these are simply split out into two 32-byte hash values ret_mix & ret_hash.
- full_nodes: the core data that the PoW function works with. It is a large dataset (>1GB) of hashes that can be reproduced consistently by any mining computer. The PoW creates hashes based on pseudo-random chunks of data from this dataset. This dataset is also referred to as the DAG, after Dagger Hashimoto (a hashing algorithm that contributed to how the DAG is generated).

PoW Kernel File

- ethash_params: contained information about the size of some data, including the DAG. To simplify things for now we discard this argument and test on a fixed size DAG.
- header_hash: a 32-byte hash representing the previous block in a block-chain. For the sake of testing, this is just an arbitrary value since we not actually interfacing with the live blockchain.
- nonce: a random integer-valued seed. When the PoW is used to mine, it iterates over changing seed values until is produces a hash that meets certain criteria. For the sake of testing, we'll just fix this to zero.

krnl_ethash.cl

ethash_system | ethash | pow.cpp | pow.h | krnl_ethash.cl

```
1  //------------------------------------------------------------------------------
2  //
3  // kernel:  ethash
4  //
5  // Purpose: Demonstrate Ethereum Ethash in OpenCL for FPGA
6  //
7
8  /*
9   * BEGIN code from all headers
10  */
11
12 #define MIX_BYTES 128
13 #define HASH_BYTES 64
14 #define DATASET_PARENTS 256
15 #define CACHE_ROUNDS 3
16 #define ACCESSES 64
17
18 /*
19  * BEGIN from fnv.h
20  */
21
22 #define FNV_PRIME 0x01000193
23
24 static inline uint fnv_hash(const uint x, const uint y) {
25     return x*FNV_PRIME ^ y;
26 }
27
28 /*
29  * END from fnv.h
30  */
31
32 /*
33  * BEGIN from sha3.h
34  */
35
36 #define decsha3(bits) \
37         int sha3_##bits(uchar*, size_t, const uchar*, size_t);
38
39 decsha3(256)
40 decsha3(512)
41
42 static inline void SHA3_256(uchar * const ret, uchar const *data, const size_t size) {
43     sha3_256(ret, 32, data, size);
44 }
```

pow.cpp

ethash_system | ethash | pow.cpp | pow.h | krnl_ethash.cl

```
35 THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS PART OF THIS FILE AT
36 ALL TIMES.
37 ********************************************************************************/
38 #include <stdlib.h>
39 #include <fstream>
40 #include <iostream>
41 #include "pow.h"
42
43 static const int DATA_SIZE = 4096;
44
45 static const std::string error_message =
46     "Error: Result mismatch:\n"
47     "i = %d CPU result = %d Device result = %d\n";
48
49 static char nibbleToChar(unsigned nibble)
50 {
51     return (char) ((nibble >= 10 ? 'a'-10 : '0') + nibble);
52 }
53
54 static uint8_t charToNibble(char chr)
55 {
56     if (chr >= '0' && chr <= '9')
57     {
58         return (uint8_t) (chr - '0');
59     }
60     if (chr >= 'a' && chr <= 'z')
61     {
62         return (uint8_t) (chr - 'a' + 10);
63     }
64     if (chr >= 'A' && chr <= 'Z')
65     {
66         return (uint8_t) (chr - 'A' + 10);
67     }
68     return 0;
69 }
70
71 static std::vector<uint8_t> hexStringToBytes(char const* str)
72 {
73     std::vector<uint8_t> bytes(strlen(str) >> 1);
74     for (unsigned i = 0; i != bytes.size(); ++i)
75     {
76         bytes[i] = charToNibble(str[i*2 | 0]) << 4;
77         bytes[i] |= charToNibble(str[i*2 | 1]);
78     }
```

pow.h

ethash_system | ethash | pow.cpp | pow.h | krnl_ethash.cl

```
23 profits, goodwill, or any type of loss or damage suffered as a result of any
24 action brought by a third party) even if such damage or loss was reasonably
25 foreseeable or Xilinx had been advised of the possibility of the same.
26 CRITICAL APPLICATIONS
27 Xilinx products are not designed or intended to be fail-safe, or for use in any
28 application requiring fail-safe performance, such as life-support or safety
29 devices or systems, Class III medical devices, nuclear facilities, applications
30 related to the deployment of airbags, or any other applications that could lead
31 to death, personal injury, or severe property or environmental damage
32 (individually and collectively, "Critical Applications"). Customer assumes the
33 sole risk and liability of any use of Xilinx products in Critical Applications,
34 subject only to applicable laws and regulations governing limitations on product
35 liability.
36 THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS PART OF THIS FILE AT
37 ALL TIMES.
38 ********************************************************************************/
39
40 #pragma once
41
42 #define CL_HPP_CL_1_2_DEFAULT_BUILD
43 #define CL_HPP_TARGET_OPENCL_VERSION 120
44 #define CL_HPP_MINIMUM_OPENCL_VERSION 120
45 #define CL_HPP_ENABLE_PROGRAM_CONSTRUCTION_FROM_ARRAY_COMPATIBILITY 1
46
47 #include <CL/cl2.hpp>
48
49 //Customized buffer allocation for 4K boundary alignment
50 template <typename T>
51 struct aligned_allocator
52 {
53   using value_type = T;
54   T* allocate(std::size_t num)
55   {
56     void* ptr = nullptr;
57     if (posix_memalign(&ptr,4096,num*sizeof(T)))
58       throw std::bad_alloc();
59     return reinterpret_cast<T*>(ptr);
60   }
61   void deallocate(T* p, std::size_t num)
62   {
63     free(p);
64   }
65 };
66
```

Builds

Since we do not have the Alveo U200 actual board, we can only build the Emulation-SW and Emulation-HW stages.

Emulation-SW Builds

Console | Problems | Vitis Log | Guidance

Build Console [ethash_system, Debug]

```
01:09:02 **** Build of configuration Debug for project ethash_system ****
make all
Skipping SD card image generation. Reason: System sdcard generation is not supported for Datacenter applic

01:09:03 Build Finished (took 481ms)
```

Compile Summary — krnl_ethash (Hardware Emulation) | Kernel Estimate — krnl_ethash (Hardware Emulation)

krnl_ethash — /home/msoc/Downloads/ethash/Emulation-HW/binary_container_1.build/krnl_ethash.xo.compile_summary — Hardware Emulation

STATUS

Completed

Logs

STARTED May 10, 2021 01:13 **COMPLETED** May 10, 2021 01:14 **ELAPSED** 55s

PLATFORM xilinx_u200_xdma_201830_2

COMMAND LINE

```
--target hw_emu
--compile
-I../src
--config common-config.ini
--config binary_container_1-krnl_ethash-compile.ini
-obinary_container_1.build/krnl_ethash.xo ../src/krnl_ethash.cl
```

Emulation-HW Builds

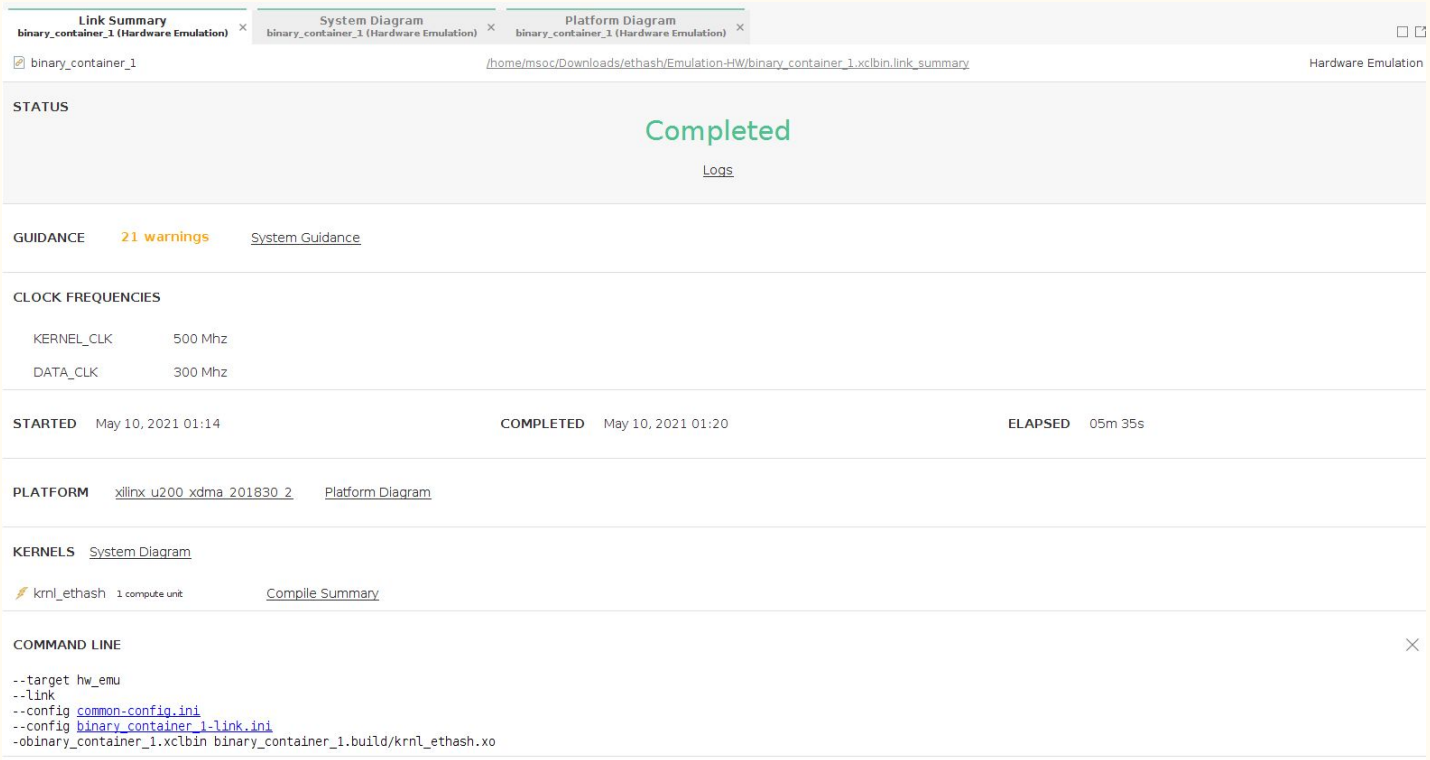
Console | Problems | Vitis Log | Guidance

Build Console [ethash, Emulation-HW]

```
****** configutil v2019.2 (64-bit)
  **** SW Build 2708876 on Wed Nov  6 21:39:14 MST 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

INFO: [ConfigUtil 60-895]   Target platform: /opt/xilinx/platforms/xilinx_u200_xdma_201830_2/xilinx_u200_x
INFO: [ConfigUtil 60-1578]   This platform contains Device Support Archive '/opt/xilinx/platforms/xilinx_u
emulation configuration file `emconfig.json` is created in . directory

01:20:16 Build Finished (took 6m:45s.561ms)
```



Emulation-HW Kernel Estimates

Compile Summary
krnl_ethash (Hardware Emulation)

Kernel Estimate
krnl_ethash (Hardware Emulation)

/home/msoc/Downloads/ethash/Emulation-HW/binary_container_1.build/reports/krnl_ethash/system_estimate_krnl_ethash.xtxt

```
================================================================================
Version:    v++ v2019.2 (64-bit)
Build:      SW Build 2708876 on Wed Nov  6 21:39:14 MST 2019
Copyright:  Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
Created:    Mon May 10 01:14:28 2021
================================================================================

--------------------------------------------------------------------------------
Design Name:             krnl_ethash
Target Device:           xilinx:u200:xdma:201830.2
Target Clock:            300.000000MHz
Total number of kernels: 1
--------------------------------------------------------------------------------

Kernel Summary
Kernel Name  Type  Target              OpenCL Library  Compute Units
-----------  ----  ------------------  --------------  -------------
krnl_ethash  clc   fpga0:OCL_REGION_0  krnl_ethash     1


--------------------------------------------------------------------------------
OpenCL Binary:     krnl_ethash
Kernels mapped to: clc_region

Timing Information (MHz)
Compute Unit   Kernel Name  Module Name  Target Frequency  Estimated Frequency
-------------  -----------  -----------  ----------------  -------------------
krnl_ethash_1  krnl_ethash  xorin_1      300.300293        424.088196
krnl_ethash_1  krnl_ethash  keccakf_1    300.300293        692.041504
krnl_ethash_1  krnl_ethash  setout_2     300.300293        714.796265
krnl_ethash_1  krnl_ethash  krnl_ethash  300.300293        411.015198

Latency Information
Compute Unit   Kernel Name  Module Name  Start Interval  Best (cycles)  Avg (cycles)  Worst (cycles)  Best (absolute)  Avg (absolute)  Worst (absolute)
-------------  -----------  -----------  --------------  -------------  ------------  --------------  ---------------  --------------  ----------------
krnl_ethash_1  krnl_ethash  xorin_1      122 ~ 290       122            undef         290             0.407 us         undef           0.967 us
krnl_ethash_1  krnl_ethash  keccakf_1    51              51             51            51              0.170 us         0.170 us        0.170 us
krnl_ethash_1  krnl_ethash  setout_2     35              35             35            35              0.117 us         0.117 us        0.117 us
krnl_ethash_1  krnl_ethash  krnl_ethash  11311 ~ 11647   11310          undef         11646           37.696 us        undef           38.816 us

Area Information
Compute Unit   Kernel Name  Module Name  FF     LUT    DSP  BRAM  URAM
-------------  -----------  -----------  -----  -----  ---  ----  ----
krnl_ethash_1  krnl_ethash  xorin_1      105    1428   0    0     0
krnl_ethash_1  krnl_ethash  keccakf_1    3553   8879   0    0     0
krnl_ethash_1  krnl_ethash  setout_2     346    1564   0    0     0
krnl_ethash_1  krnl_ethash  krnl_ethash  22057  27906  144  49    0
--------------------------------------------------------------------------------
```

Emulation-HW Performance Estimates

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.33 ns	2.433 ns	0.90 ns

Latency

Summary

min	max	min	max	min	max	Type
11310	11646	37.696 us	38.816 us	11310	11646	none

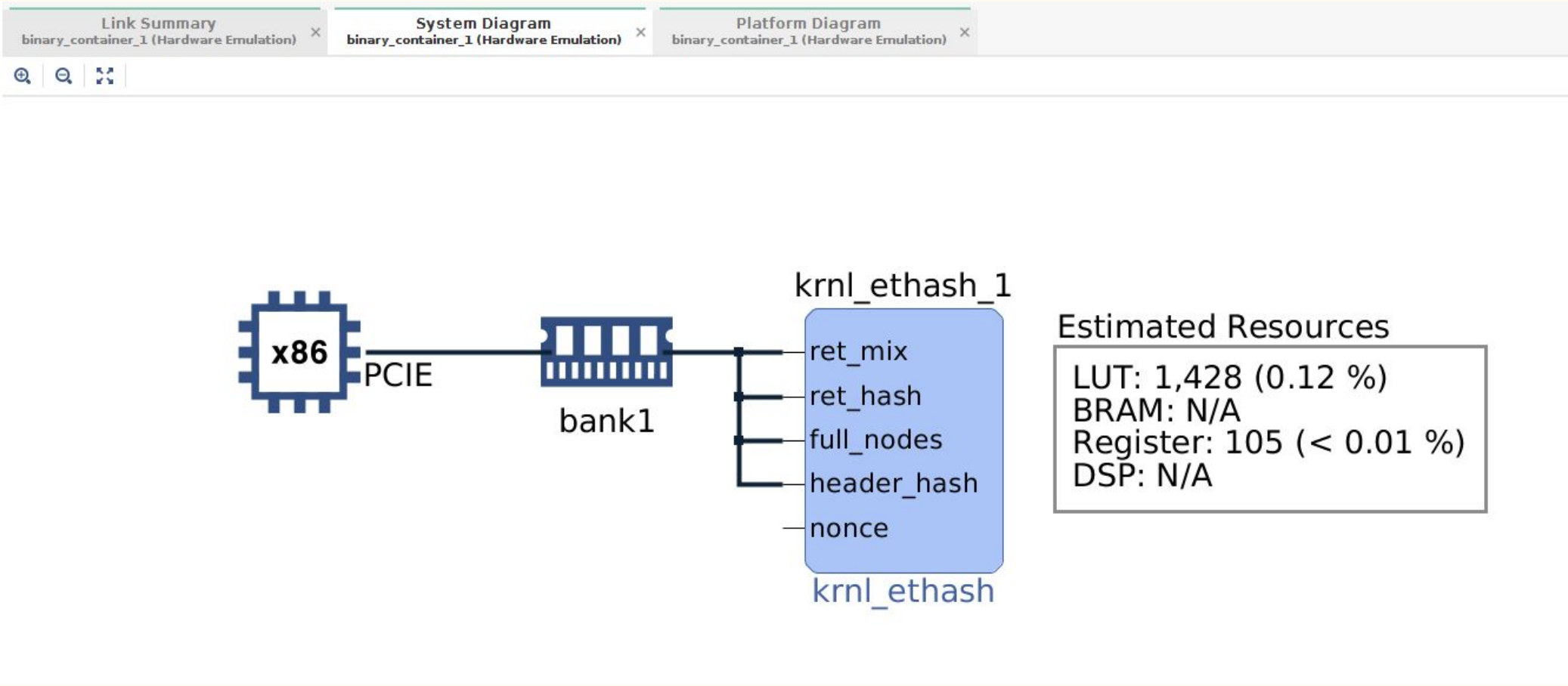
Emulation-HW Utilization Estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	7619	-
FIFO	-	-	-	-	-
Instance	30	144	16372	17574	-
Memory	19	-	0	0	0
Multiplexer	-	-	-	2713	-
Register	-	-	5685	-	-
Total	49	144	22057	27906	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	1	2	~0	2	0
Utilization SLR (%)	3	6	2	7	0

Emulation-HW System Diagram



Key Message 1

- The report indicates a relatively long latency estimate for the kernel at ~0.038ms and the memory utilization is very low at just ~1%.
- This is because we took C code designed to run in a fairly serial manner on CPU clocked at a very high frequency and applied it to an FPGA, which is very good at leveraging parallelism, and ran it in a still serial manner at a much lower clock rate (300MHz).
- For the sake of acceleration, we need toward additional optimizations that can yield higher levels of memory utilization and parallelism.

Key Message 2

- Lots of "科普" articles talk about and compare FPGA blockchain mining with GPU and ASIC, but none of them really tells you how to actually mine with FPGA.
- If the goal is the whole end-to-end FPGA blockchain mining (i.e., we can actually mine with the FPGA), this is just the very beginning.

Github Link

https://github.com/agenda425/hls_ethash