Binomial Options Pricing Model: Reference Design

Table of Contents:

Prerequisites

To run the reference design, you need to

- mount Alveo U200 card in your system and
- install the following mandatory components:

SDAccel Version	**2018.3**
Target Platform	**Alveo U200 card:** • **2018.3** deployment shell: **xilinx_u200_xdma_201830_1** • **2018.3** development shell: **xilinx_u200_xdma_dev_201830_1**
XRT Version	**2018.3**

Refer to the **UG1301** and **UG1238** user guides for more information regarding Alveo and SDAccel installation.

The SDAccel development environment can be run using GUI and Makefile modes. This tutorial uses the GUI mode.

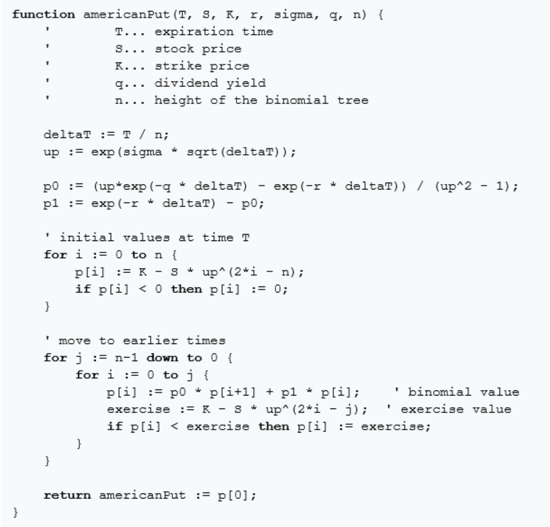
Introduction

The goal of this reference design (tutorial) is by using a Binomial Options Pricing Model (further referenced as Binomial Model) to:

- Demonstrate performance advantage of the Xilinx Alveo-based solution vs. the Intel® Xeon® CPU one
- Introduce SDAccel flow, various analysis tools and optimization techniques allowing you to improve design performance significantly

The figure below shows the original description of the Binomial Model obtained from Wikipedia (https://en.wikipedia.org/wiki/Binomial_options_pricing_model)

The original Binomial Model algorithm is rewritten in a C/C++ programming language by using single precision floating point operations and keeping the C/C++ coding style as close as possible to the original version.



```
function americanPut(T, S, K, r, sigma, q, n) {
    '        T... expiration time
    '        S... stock price
    '        K... strike price
    '        q... dividend yield
    '        n... height of the binomial tree

    deltaT := T / n;
    up := exp(sigma * sqrt(deltaT));

    p0 := (up*exp(-q * deltaT) - exp(-r * deltaT)) / (up^2 - 1);
    p1 := exp(-r * deltaT) - p0;

    ' initial values at time T
    for i := 0 to n {
        p[i] := K - S * up^(2*i - n);
        if p[i] < 0 then p[i] := 0;
    }

    ' move to earlier times
    for j := n-1 down to 0 {
        for i := 0 to j {
            p[i] := p0 * p[i+1] + p1 * p[i];    ' binomial value
            exercise := K - S * up^(2*i - j);  ' exercise value
            if p[i] < exercise then p[i] := exercise;
        }
    }

    return americanPut := p[0];
}
```

To start, implement the original C/C++ description on Xilinx Alveo card and x86 CPU and compare their performance.

- Run the original algorithm on CPU with a single thread
- Implement the same algorithm on an Alveo card as a fully sequential process for data movement and kernel execution

Then by using various optimization techniques, increase the Alveo based performance by the factor of **60x**. Compare the Alveo results with the ones obtained on CPU running in a multithread mode.

Test Environment and Important Notes

All results represented in this tutorial are generated using a **DELL 8510 Workstation** with the following characteristics:

- **CPU**: Intel® Xeon® E5-1650 v3 @ 3.50 GHz
  - Number of threads per core: 6
  - Number of cores per socket: 2
- **RAM**: 64 GB
- **OS**: CentOS 7.4

IMPORTANT: the results you will obtain on your machine and the ones presented in this document may differ.

General Flow for this reference design

- **Step 1:** Projects Setup.
- **Step 2:** Important design characteristics
- **Step 3:** Running SW Model on X86 CPU
- **Step 4:** Alveo: Original Design
- **Step 5**: Alveo: Host Optimization - Data Transfer and Kernel Execution Overlap
- **Step 6**: Alveo: Single Kernel Optimization - Data Movement
- **Step 7**: Alveo: Single Kernel Optimization - Running 4 Functions in Parallel
- **Step 8**: Alveo: Single Kernel Optimization - Additional Loops Unrolling
- **Step 9:** Alveo: Single Kernel Optimization - System Run Results
- **Step 10:** Alveo: Using Multiple Compute Units and Different DDRs

Projects Setup Step 1

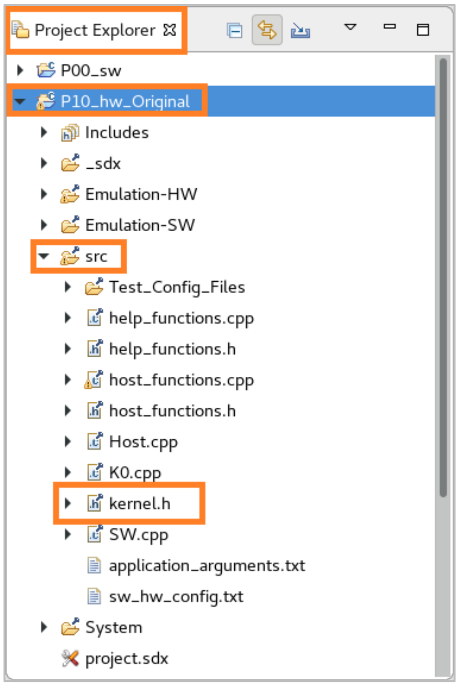
The reference design consists of several projects, and all design source code is available on GitHub (https://github.com/Xilinx/BinomialModel). You need to import them to your local machine and then create projects using the instructions listed in the **Appendix A: Projects Setup: Detailed Steps**.

Important Design Characteristics Step 2

You have successfully set up all projects, and they are ready to be run. However, before doing that, let's clarify several design aspects.

- In this tutorial, for demonstration purposes and for the sake of clarity, we were trying to keep the kernel coding style as simple as possible. For which, we assumed:
  - a Binomial tree height is limited to **1024** and
  - a Kernel can process maximum **1024** option positions
    - ***IMPORTANT****: in this document, the* ***number of option positions*** *will be further referenced as a* ***number of test vectors***

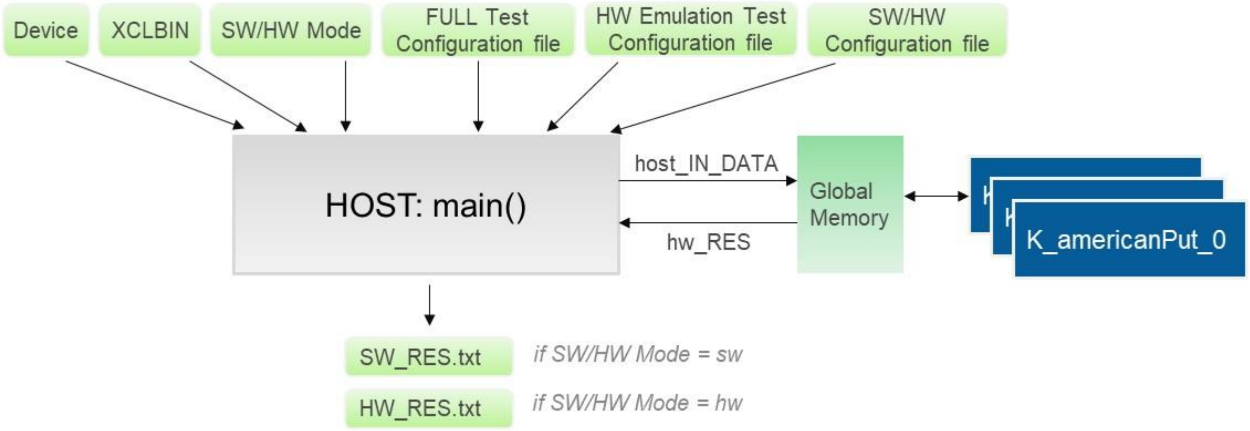
  which is defined in a **kernel.h** file in each project. For example, open the **kernel.h** file from the **P10_hw_Original** project:



```
...
#define CONST_MAX_TREE_HEIGHT 1024
#define CONST_MAX_NB_OF_TESTS 1024
...
```

- As previously mentioned before, a **single precision floating point** data type and corresponding functions were used to implement the algorithm.
- The Host code is located in the **Host.cpp** file and kernels, depending on a project are located in **K0.cpp**, **K1.cpp** and **K2.cpp** files.

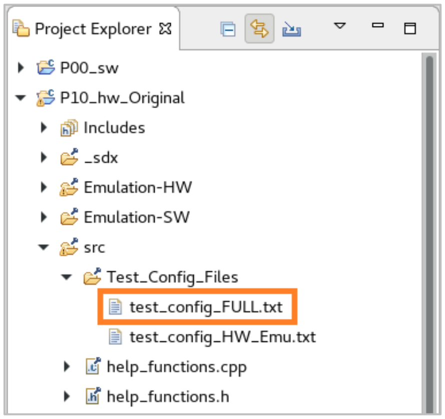
- The figure below represents the overall application structure:



The Host has 6 input arguments:

- **Device** – the target Alveo device.
  - For this design we use **xilinx_u200_xdma_201830_1**
- **XCLBIN** – the name of the Xilinx binary container where the kernel(s) are compiled
  - In our case, this is a **binary_container_1.xclbin** file
- **SW/HW Mode** - defines if the application runs as :
  - SW Model **(sw)** on an x86 CPU or
  - HW implementation **(hw)** on the Alveo card
- **FULL Test Configuration file** - defines Binomial application arguments to be used for
  - SW Model (SW/HW Mode = **sw**)
  - SW Emulation and System Run mode (SW/HW Mode = **hw**)

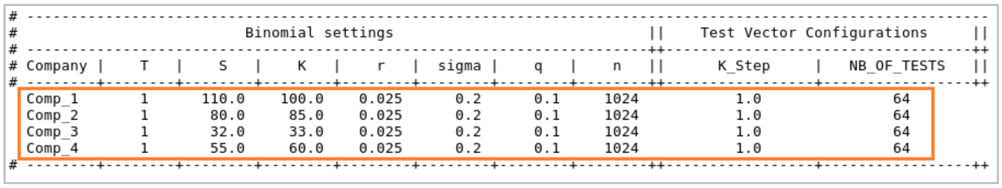
  The name of this file is **test_config_FULL.txt,** and it is available in each project.



Project Explorer

```
▸ P00_sw
▾ P10_hw_Original
  ▸ Includes
  ▸ _sdx
  ▸ Emulation-HW
  ▸ Emulation-SW
  ▾ src
    ▾ Test_Config_Files
        test_config_FULL.txt
        test_config_HW_Emu.txt
    ▸ help_functions.cpp
    ▸ help_functions.h
```

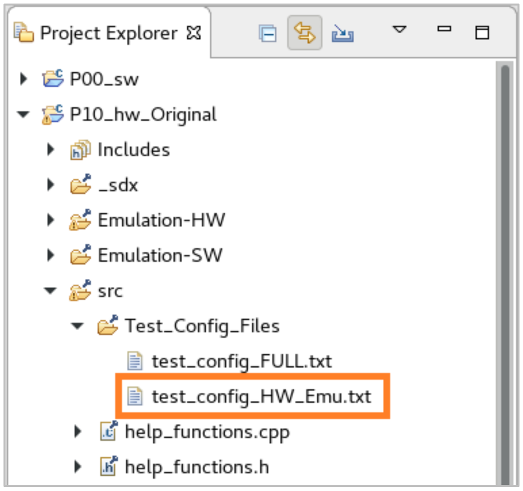
Reduced content of this file is represented in the following figure (please review this file for more information):



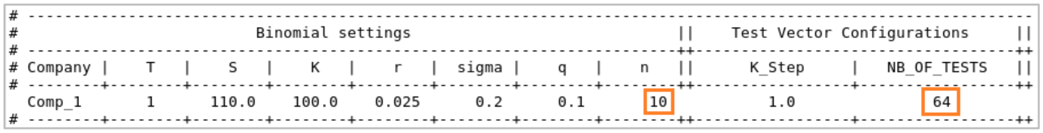
```
# ------------------------------------------------------------------------------------------------
#                         Binomial settings                               ||    Test Vector Configurations     ||
# ------------------------------------------------------------------------++-----------------------------------++
# Company |    T    |    S    |    K    |    r    |  sigma |    q    |    n   ||       K_Step       |  NB_OF_TESTS   ||
# --------+---------+---------+---------+---------+--------+---------+--------++--------------------+----------------++
  Comp_1       1      110.0     100.0    0.025      0.2       0.1     1024           1.0                   64
  Comp_2       1       80.0      85.0    0.025      0.2       0.1     1024           1.0                   64
  Comp_3       1       32.0      33.0    0.025      0.2       0.1     1024           1.0                   64
  Comp_4       1       55.0      60.0    0.025      0.2       0.1     1024           1.0                   64
# --------+---------+---------+---------+---------+--------+---------+--------++--------------------+----------------++
```

- **HW Emulation Test Configuration file** - defines Binomial application arguments (reduced complexity) to be used in HW Emulation mode (SW/HW Mode = **hw**). It ensures fast execution runtimes in HW Emulation mode.
  - This file is ignored when the SW Model is run on CPU (SW/HW Mode = **sw**).

  The name of this file is **test_config_HW_Emu.txt,** and it is available in each project:



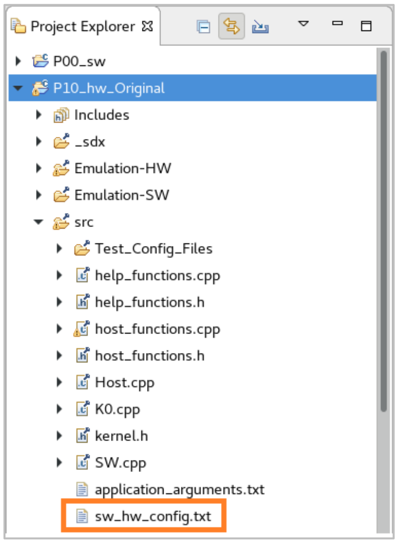
Reduced content of this file is represented on the following figure (please review this file for more information):



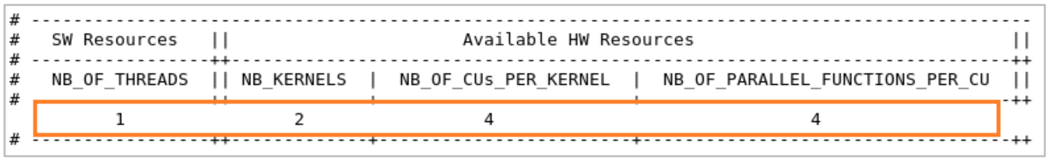
```
# ------------------------------------------------------------------------------------------------
#                         Binomial settings                               ||    Test Vector Configurations     ||
# ------------------------------------------------------------------------++-----------------------------------++
# Company |    T    |    S    |    K    |    r    |  sigma |    q    |    n   ||       K_Step       |  NB_OF_TESTS   ||
# --------+---------+---------+---------+---------+--------+---------+--------++--------------------+----------------++
  Comp_1       1      110.0     100.0    0.025      0.2       0.1       10           1.0                   64
# --------+---------+---------+---------+---------+--------+---------+--------++--------------------+----------------++
```

- **SW / HW Configuration file -** defines
  - The number of threads to be used when running the SW Model on CPU (SW/HW Mode = sw)
  - Kernel configuration implemented on the Alveo card (SW/HW Mode = hw):
    - Number of Kernels
    - Number of Compute Units (CUs) per kernel
    - Number of parallel Binomial functions per CU

  The name of the configuration file for **all** projects (except P00_sw) is **hw_sw_config.txt**:



Reduced content of this file is represented on the following figure (please review this file for more information):

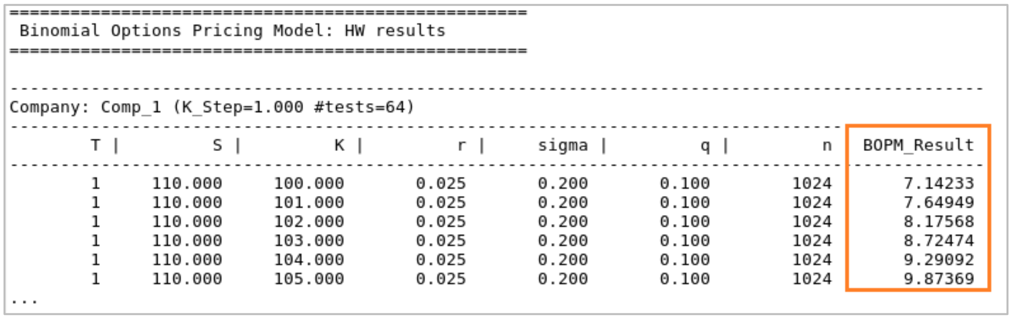


```
# -----------------------------------------------------------------------------------------------------
#   SW Resources    ||                            Available HW Resources                               ||
# ------------------++---------------------------------------------------------------------------------++
#   NB_OF_THREADS   || NB_KERNELS   |  NB_OF_CUs_PER_KERNEL   |  NB_OF_PARALLEL_FUNCTIONS_PER_CU  ||
# ------------------++--------------+--------------------------+-----------------------------------++
           1                2                   4                                 4
# ------------------++--------------+--------------------------+-----------------------------------++
```

The **P00_sw** project has two configuration files: **hw_sw_config_1_Thread.txt** and **hw_sw_config_M_Thread.txt.**

- **Result files** – the name of the result file is
  - **SW_RES.txt** if SW/WH Mode = sw
  - **HW_RES.txt** if SW/WH Mode = hw

Here is an example of a result file where the **BOPM_Result** column contains Binomial Model results:

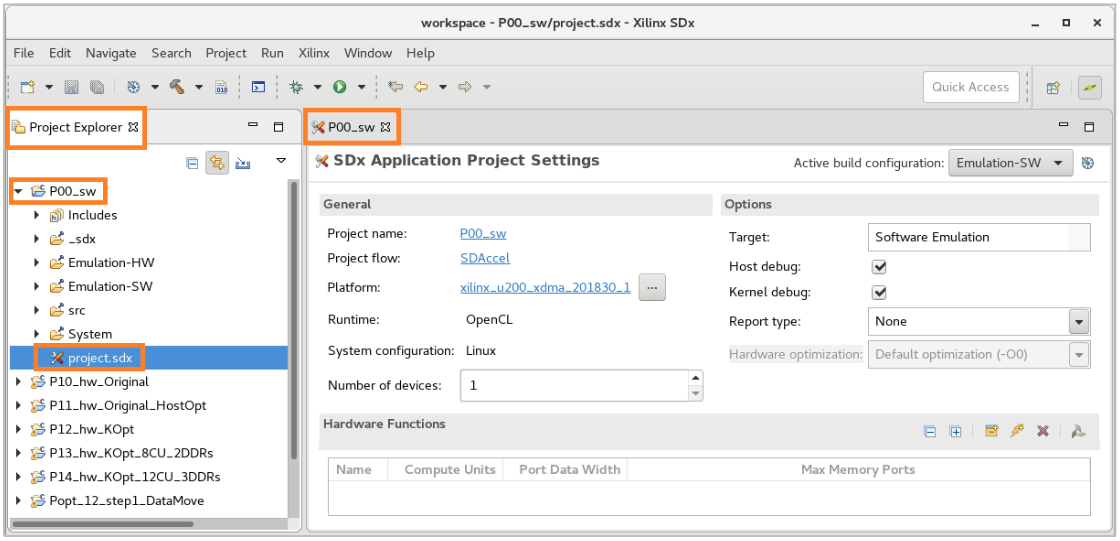


```
====================================================
 Binomial Options Pricing Model: HW results
====================================================

------------------------------------------------------------------------------------------------
Company: Comp_1 (K_Step=1.000 #tests=64)
------------------------------------------------------------------------------------------------
        T |          S |          K |          r |      sigma |          q |          n  BOPM_Result
------------------------------------------------------------------------------------------------
        1      110.000      100.000        0.025        0.200        0.100        1024      7.14233
        1      110.000      101.000        0.025        0.200        0.100        1024      7.64949
        1      110.000      102.000        0.025        0.200        0.100        1024      8.17568
        1      110.000      103.000        0.025        0.200        0.100        1024      8.72474
        1      110.000      104.000        0.025        0.200        0.100        1024      9.29092
        1      110.000      105.000        0.025        0.200        0.100        1024      9.87369
...
```

Running SW Model on X86 CPU Step 3

Projects: **P00_sw**

Open **P00_sw** project by double-clicking on the **project.sdx** under the **P00_sw** in **Project Explorer**:


workspace - P00_sw/project.sdx - Xilinx SDx
File Edit Navigate Search Project Run Xilinx Window Help
Quick Access
Project Explorer
P00_sw
Hardware Functions

Project Explorer:

- P00_sw
  - Includes
  - _sdx
  - Emulation-HW
  - Emulation-SW
  - src
  - System
  - project.sdx
- P10_hw_Original
- P11_hw_Original_HostOpt
- P12_hw_KOpt
- P13_hw_KOpt_8CU_2DDRs
- P14_hw_KOpt_12CU_3DDRs
- Popt_12_step1_DataMove

SDx Application Project Settings

Active build configuration: Emulation-SW

General

Project name: P00_sw
Project flow: SDAccel
Platform: xilinx_u200_xdma_201830_1
Runtime: OpenCL
System configuration: Linux
Number of devices: 1

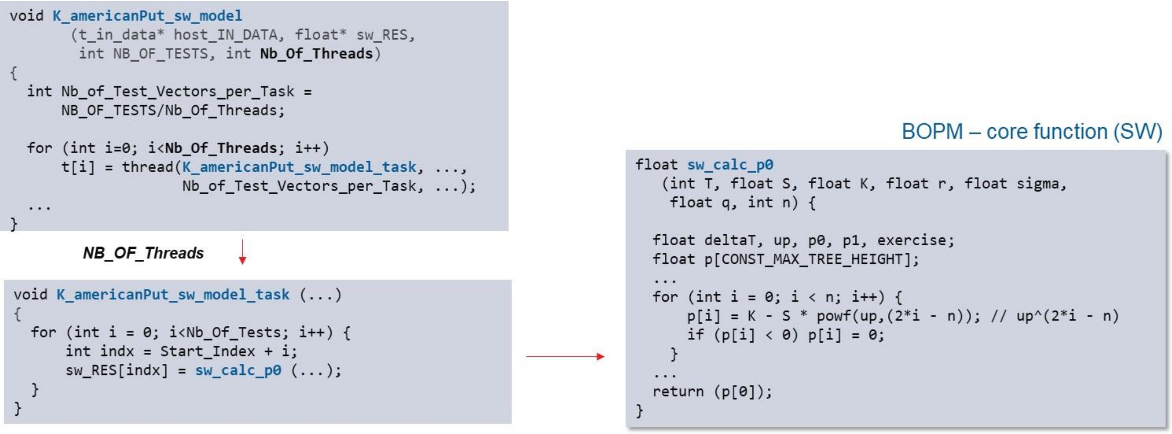
Options

Target: Software Emulation
Host debug: ☑
Kernel debug: ☑
Report type: None
Hardware optimization: Default optimization (-O0)

Name	Compute Units	Port Data Width	Max Memory Ports

This project represents a SW Model (**SW.cpp** file) of the Binomial Model to be run on CPU.

In this file, the **sw_calc_p0** function represents a Binomial Model algorithm. If you compare this description with the one published on Wikipedia, then you will see that the coding styles are identical in both cases.



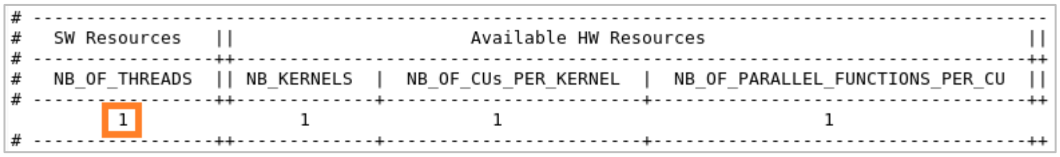
```
void K_americanPut_sw_model
       (t_in_data* host_IN_DATA, float* sw_RES,
        int NB_OF_TESTS, int Nb_Of_Threads)
{
  int Nb_of_Test_Vectors_per_Task =
      NB_OF_TESTS/Nb_Of_Threads;

  for (int i=0; i<Nb_Of_Threads; i++)
      t[i] = thread(K_americanPut_sw_model_task, ...,
                    Nb_of_Test_Vectors_per_Task, ...);
  ...
}
```

```
void K_americanPut_sw_model_task (...)
{
  for (int i = 0; i<Nb_Of_Tests; i++) {
      int indx = Start_Index + i;
      sw_RES[indx] = sw_calc_p0 (...);
  }
}
```

```
float sw_calc_p0
   (int T, float S, float K, float r, float sigma,
    float q, int n) {

  float deltaT, up, p0, p1, exercise;
  float p[CONST_MAX_TREE_HEIGHT];
  ...
  for (int i = 0; i < n; i++) {
      p[i] = K - S * powf(up,(2*i - n)); // up^(2*i - n)
      if (p[i] < 0) p[i] = 0;
   }
  ...
  return (p[0]);
}
```

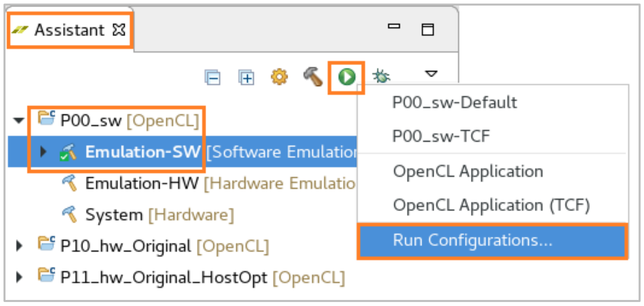
The **sw_calc_p0** is called multiple times by **K_americanPut_sw_model_task** and **K_americanPut_sw_model** functions to execute several test vectors. The **K_americanPut_sw_model** is called from a host (**Host.cpp**).

As you can also notice, the **K_americanPut_sw_model** can be configured to run multiple threads. However, at this step of the tutorial, a single thread is used, which is defined in the **sw_hw_config_1_Thread.txt** configuration file

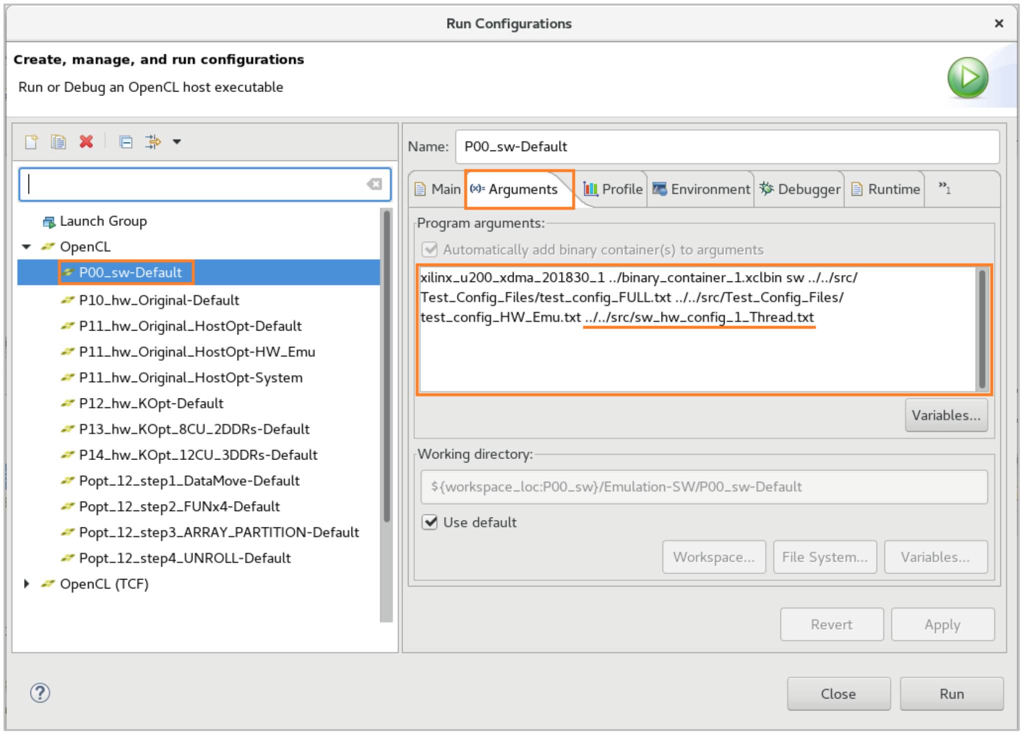


```
# ------------------------------------------------------------------------------------------------
#   SW Resources   ||                      Available HW Resources                               ||
# -----------------++-----------------------------------------------------------------------------++
#   NB_OF_THREADS  || NB_KERNELS  |  NB_OF_CUs_PER_KERNEL  |  NB_OF_PARALLEL_FUNCTIONS_PER_CU  ||
# -----------------++-------------+------------------------+-----------------------------------++
            1                1                 1                          1
# -----------------++-------------+------------------------+-----------------------------------++
```

which is passed to the host program as a command line option. To access the command line options, in the **Assistant** tab under the **P00_sw** project select **Emulation-SW**, then click on the ▶ icon and select **Run Configurations** menu:



In the opened dialog box ensure that **P00_sw-Default** run is selected in the left-side window. Select **Arguments** tab to access the command line arguments passed to the application:


Run Configurations
Launch Group
OpenCL
P00_sw-Default
P10_hw_Original-Default
P11_hw_Original_HostOpt-Default
P11_hw_Original_HostOpt-HW_Emu
P11_hw_Original_HostOpt-System
P12_hw_KOpt-Default
P13_hw_KOpt_8CU_2DDRs-Default
P14_hw_KOpt_12CU_3DDRs-Default
Popt_12_step1_DataMove-Default
Popt_12_step2_FUNx4-Default
Popt_12_step3_ARRAY_PARTITION-Default
Popt_12_step4_UNROLL-Default
OpenCL (TCF)
Main
Arguments
Profile
Environment
Debugger
Runtime
Variables...
Workspace...
File System...
Variables...
Revert
Apply
Close
Run

Create, manage, and run configurations

Run or Debug an OpenCL host executable

Name: P00_sw-Default

Program arguments:

☑ Automatically add binary container(s) to arguments

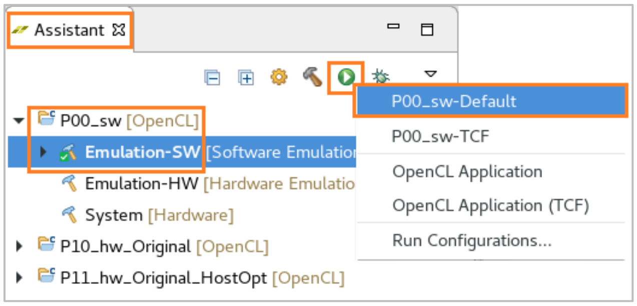
```
xilinx_u200_xdma_201830_1 ../binary_container_1.xclbin sw ../../src/
Test_Config_Files/test_config_FULL.txt ../../src/Test_Config_Files/
test_config_HW_Emu.txt ../../src/sw_hw_config_1_Thread.txt
```

Working directory:

${workspace_loc:P00_sw}/Emulation-SW/P00_sw-Default

☑ Use default

To run the SW Model on CPU with a single thread, in the **Assistant** tab under the **P00_sw** project select **Emulation-SW**, then click on the ▶ icon and select **P00_sw-Default**:



The application execution is successfully completed, and CPU execution time is reported in a Console window:

```
HOST-Info: ============================================================
HOST-Info: Step: Run SW Model
HOST-Info: ============================================================
HOST_Info: SW Model Execution
HOST_Info:     # Threads    = 1
HOST_Info:     Runtime (ms) = 9834.2
```

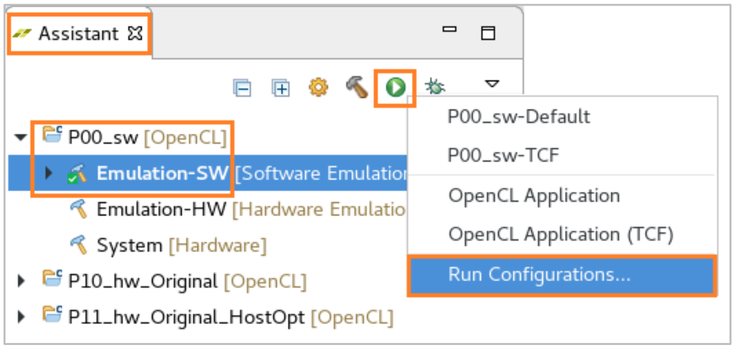
Recall: the results you obtain on your machine and the ones represented in this document may differ.

Later on (step 4), you will compare this result against the original code implementation on the Alveo card.

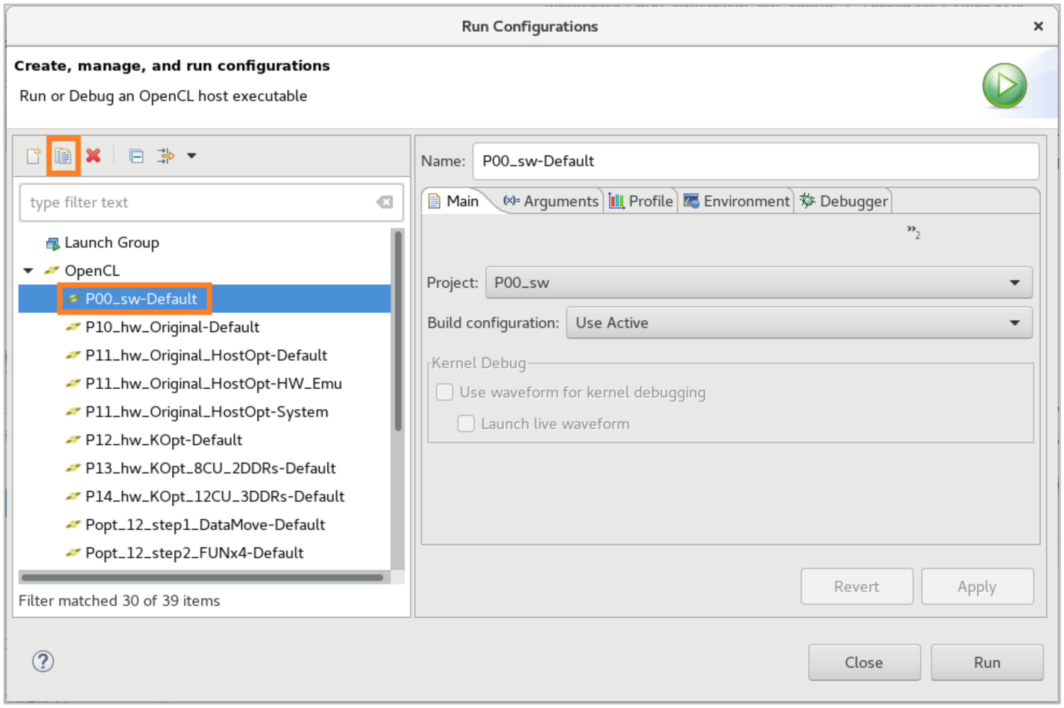
As previously mentioned, the optimized Alveo implementation will be compared against the CPU result executed in a multithreading mode. For the sake of time, let’s generate the multithreading result now and use it in later steps.

To achieve this, you need to create an additional run, called **P00_sw-M_Thread** and specify the **sw_hw_config_M_Thread.txt** configuration file in the command line options.

- In the **Assistant** tab under the **P00_sw** project select **Emulation-SW**, click on the ▶ icon and select **Run Configurations** menu



- In the opened dialog box, ensure that **P00_sw_Default** run is selected and the press on the **icon** to duplicate the default run



Run Configurations

Create, manage, and run configurations

Run or Debug an OpenCL host executable

Launch Group
OpenCL
P00_sw-Default
P10_hw_Original-Default
P11_hw_Original_HostOpt-Default
P11_hw_Original_HostOpt-HW_Emu
P11_hw_Original_HostOpt-System
P12_hw_KOpt-Default
P13_hw_KOpt_8CU_2DDRs-Default
P14_hw_KOpt_12CU_3DDRs-Default
Popt_12_step1_DataMove-Default
Popt_12_step2_FUNx4-Default

Filter matched 30 of 39 items

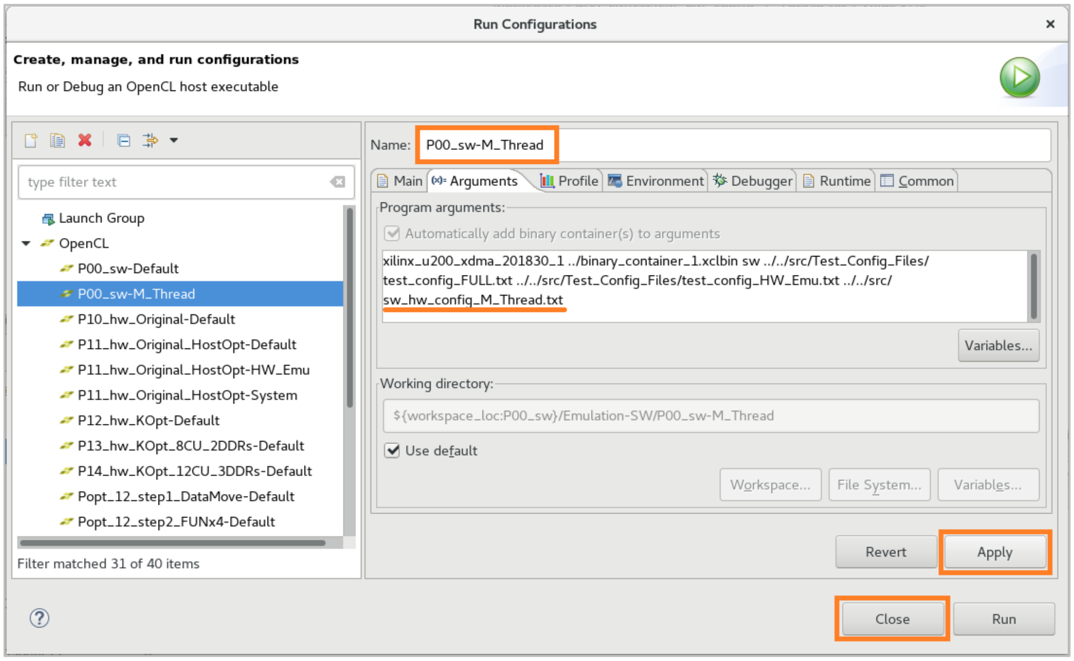
Name: P00_sw-Default

Project: P00_sw

Build configuration: Use Active

Kernel Debug
☐ Use waveform for kernel debugging
☐ Launch live waveform

- Then
  - Name the newly created run as **P00_sw-M_Thread**
  - In the **Arguments** tab, replace the **sw_hw_config_1_Thread.txt** file by the **sw_hw_config_M_Thread.txt** one:



Run Configurations

Create, manage, and run configurations

Run or Debug an OpenCL host executable

Launch Group
OpenCL
P00_sw-Default
P00_sw-M_Thread
P10_hw_Original-Default
P11_hw_Original_HostOpt-Default
P11_hw_Original_HostOpt-HW_Emu
P11_hw_Original_HostOpt-System
P12_hw_KOpt-Default
P13_hw_KOpt_8CU_2DDRs-Default
P14_hw_KOpt_12CU_3DDRs-Default
Popt_12_step1_DataMove-Default
Popt_12_step2_FUNx4-Default

Filter matched 31 of 40 items

Name: P00_sw-M_Thread

Program arguments:
☑ Automatically add binary container(s) to arguments

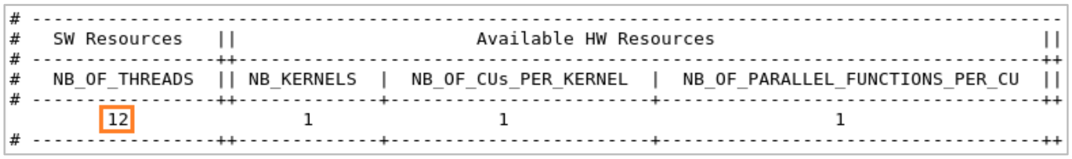
```
xilinx_u200_xdma_201830_1 ../binary_container_1.xclbin sw ../../src/Test_Config_Files/
test_config_FULL.txt ../../src/Test_Config_Files/test_config_HW_Emu.txt ../../src/
sw_hw_config_M_Thread.txt
```

Working directory:
${workspace_loc:P00_sw}/Emulation-SW/P00_sw-M_Thread

☑ Use default

Then press **Apply** and **Close**.

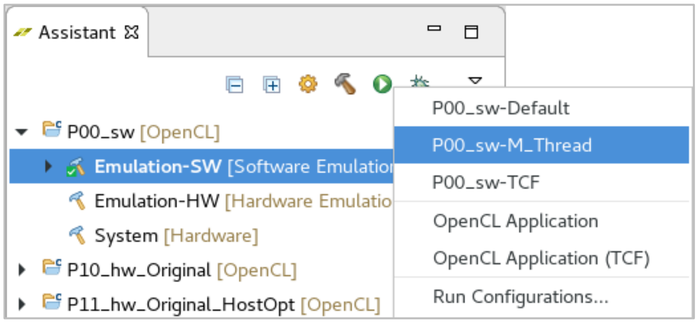
The **sw_hw_config_M_Thread.txt** file is configured to run 12 Threads on CPU:



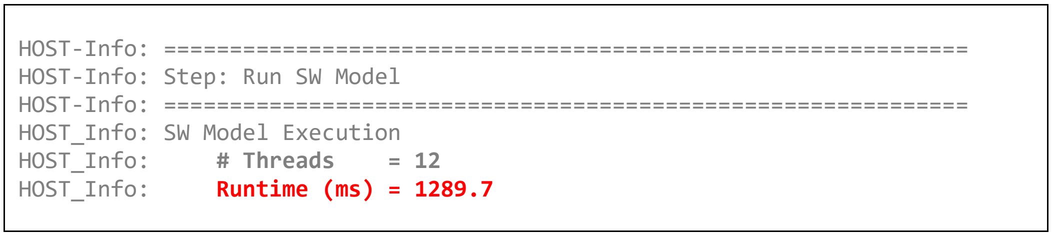
```
# -----------------------------------------------------------------------------------------------
#   SW Resources   ||                         Available HW Resources                           ||
# ----------------++--------------+-------------------------+-----------------------------------++
#   NB_OF_THREADS  || NB_KERNELS  |  NB_OF_CUs_PER_KERNEL   |  NB_OF_PARALLEL_FUNCTIONS_PER_CU  ||
# ----------------++--------------+-------------------------+-----------------------------------++
        12                 1                  1                              1
# ----------------++--------------+-------------------------+-----------------------------------++
```

You need to adjust the number of threads delivering the best results for your machine configuration.

Then launch the **P00_sw-M_Thread** run:



The application execution is successfully completed, and a CPU execution time is reported in a Console window:



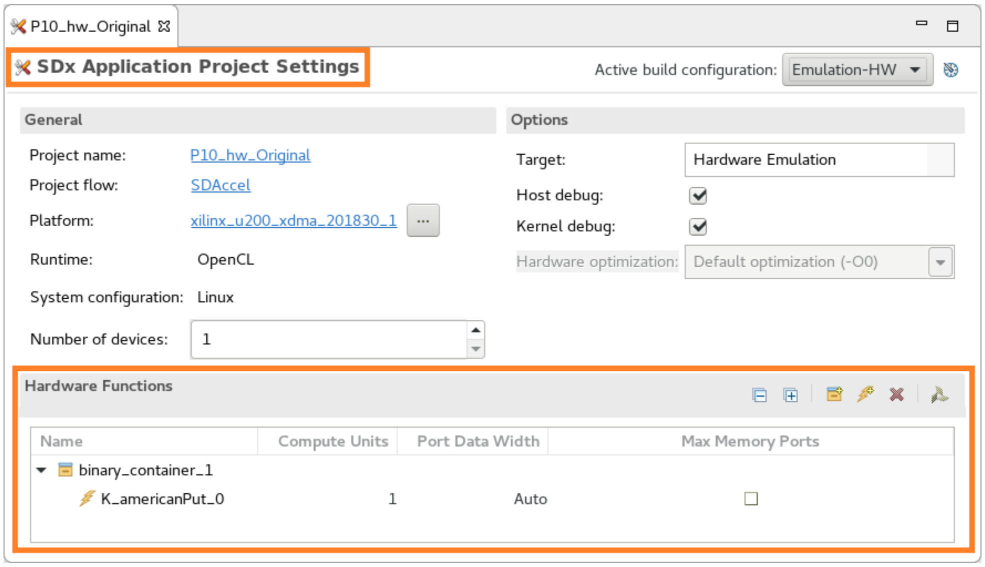
```
HOST-Info: ============================================================
HOST-Info: Step: Run SW Model
HOST-Info: ============================================================
HOST_Info: SW Model Execution
HOST_Info:     # Threads     = 12
HOST_Info:     Runtime (ms)  = 1289.7
```

Alveo: Original Design Step 4

Projects: **P10_hw_Original**

Open the **P10_hw_Original** project. This project represents the original kernel implementation on the Alveo card.

The **Hardware Functions** in the **SDx Application Project Settings** contains the list of kernels which will be implemented on the Alveo card. In this project, we have one **K_americanPut_0** kernel, which will be implemented as a single compute unit (CU) in a **binary_container_1.xclbin** file.



SDx Application Project Settings

Active build configuration: Emulation-HW

General

Project name: P10_hw_Original
Project flow: SDAccel
Platform: xilinx_u200_xdma_201830_1
Runtime: OpenCL
System configuration: Linux
Number of devices: 1

Options

Target: Hardware Emulation
Host debug: ☑
Kernel debug: ☑
Hardware optimization: Default optimization (-O0)

Hardware Functions

Name	Compute Units	Port Data Width	Max Memory Ports
binary_container_1			
K_americanPut_0	1	Auto	□

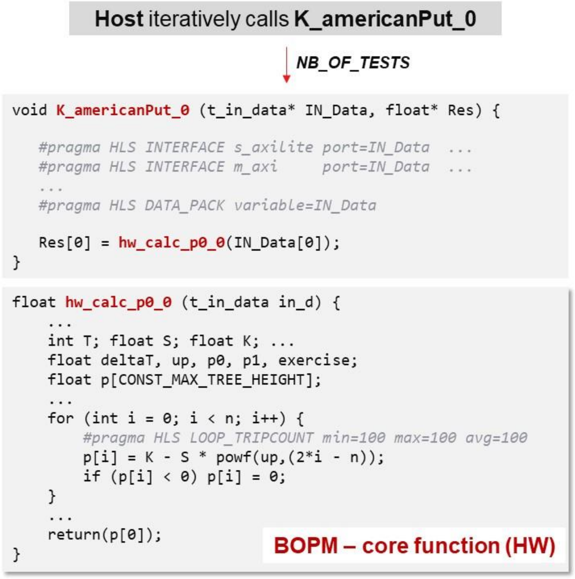
Open the **K0.cpp** file, containing the kernel description.

- In this file, the **hw_calc_p0_0** function represents the Binomial Model algorithm.
- The **hw_calc_p0_0** function is called by **K_americanPut_0**, representing a HW kernel called from the host code (**Host.cpp**).

The original application on the Alveo card is organized as a simple sequential process where input data transfer from Host to Global Memory, kernel execution and results transfer from Global to Host memory is sequentially done for each test vector.

In Alveo implementation, the **T**, **S**, **K**, and other arguments are transferred from Host to the Global memory as elements of a **t_in_data** structure defined in the **kernel.h** file.

The structure elements are decomposed on separate variables in **hw_calc_p0_0**, which represents the only difference between the **SW Model** (SW.cpp file in the P00_sw project) and the code for Alveo implementation.



```
void K_americanPut_0 (t_in_data* IN_Data, float* Res) {

   #pragma HLS INTERFACE s_axilite port=IN_Data  ...
   #pragma HLS INTERFACE m_axi     port=IN_Data  ...
   ...
   #pragma HLS DATA_PACK variable=IN_Data

   Res[0] = hw_calc_p0_0(IN_Data[0]);
}
```

```
float hw_calc_p0_0 (t_in_data in_d) {
    ...
    int T; float S; float K; ...
    float deltaT, up, p0, p1, exercise;
    float p[CONST_MAX_TREE_HEIGHT];
    ...
    for (int i = 0; i < n; i++) {
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
        p[i] = K - S * powf(up,(2*i - n));
        if (p[i] < 0) p[i] = 0;
    }
    ...
    return(p[0]);
}
```

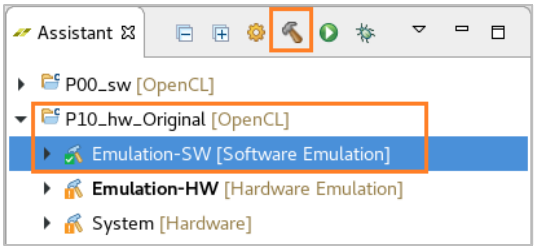
You can notice the presence of several pragmas in the **K0.cpp** file. Let's review them before continuing:

- The **INTERFACE** pragmas in **K_americanPut_0** instruct SDAccel compiler on how to connect the kernel to the Alveo shell.
- **DATA_PACK** pragma ensures that the compiler considers all fields of the t_in_data structure as a single block of data without splitting them into individual elements.
- The **LOOP_TRIPCOUNT** pragma in the **hw_calc_p0_0** function has no impact on the design optimization and are exclusively used later on for design analysis.

4-1. SW Emulation

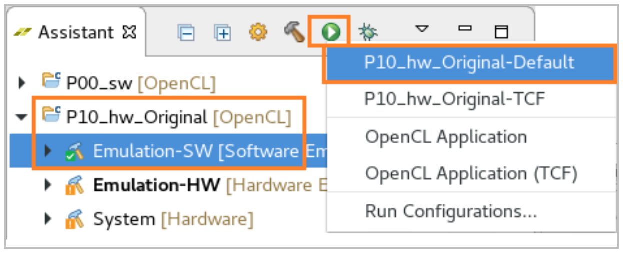
The main goal of SW Emulation is to validate the functional correctness of the design. In this mode, the runtime is very fast because both the host and the kernel code are compiled to be run on an x86 processor.

To compile the design for SW Emulation in the **Assistant tab**, under **P10_hw_Original** select **Emulation-SW**, and then press on the 🔨 icon:


Assistant
P00_sw [OpenCL]
P10_hw_Original [OpenCL]
Emulation-SW [Software Emulation]
Emulation-HW [Hardware Emulation]
System [Hardware]

The compilation process should complete successfully.

To run SW Emulation, in the **Assistant tab**, under **P10_hw_Original** select **Emulation-SW**, then click on the ▶ icon and select **P10_hw_Original-Default**:



You should see the following messages in the Console window.

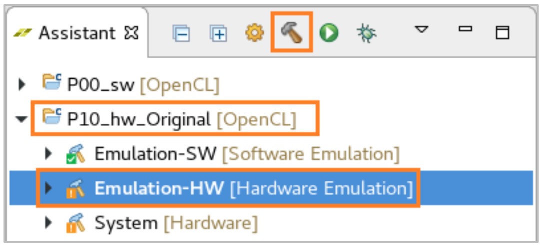
```
HOST-Info: ============================================================
HOST-Info: Step: Run Application
HOST-Info: ============================================================

HOST_Info: Waiting for application to be completed ...

HOST_Info: Test Passed
HOST-Info: Results stored in the HW_Res.txt file ...

HOST-Info: Application Completed
```

4-2. HW Emulation

While the SW Emulation flow is a good measure of functional correctness, it does not guarantee the correctness on the FPGA execution target. The Hardware Emulation flow enables checking of the correctness of the generated logic. This emulation flow invokes the hardware simulator in the SDAccel environment to test the logic functionality. As a consequence, the runtime in the Hardware Emulation flow is longer than in the SW Emulation flow.

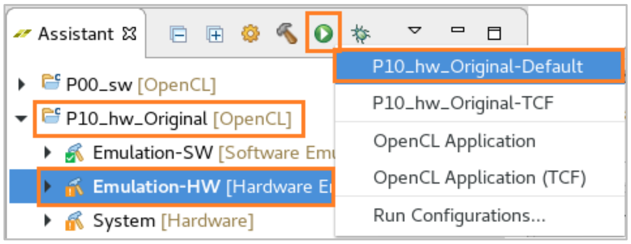
In addition, HW Emulation flow provides more comprehensive and precise profiling information allowing the user to analyze the application performance and identify the bottlenecks.

To compile the design for HW Emulation in the **Assistant tab**, under **P10_hw_Original** select **Emulation-HW**, and then press on the 🔨 icon:


Assistant
P00_sw [OpenCL]
P10_hw_Original [OpenCL]
Emulation-SW [Software Emulation]
Emulation-HW [Hardware Emulation]
System [Hardware]

The compilation process should complete successfully.

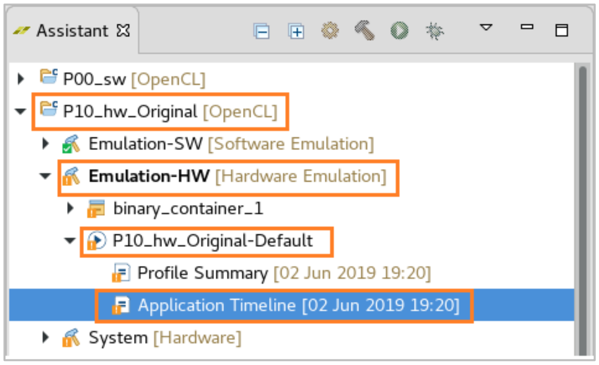
To run HW Emulation, in the **Assistant tab**, under **P10_hw_Original** select **Emulation-HW**, then click on the ▶ icon and select **P10_hw_Original-Default**:



The application is successfully run, and you should see the following messages in the Console window.

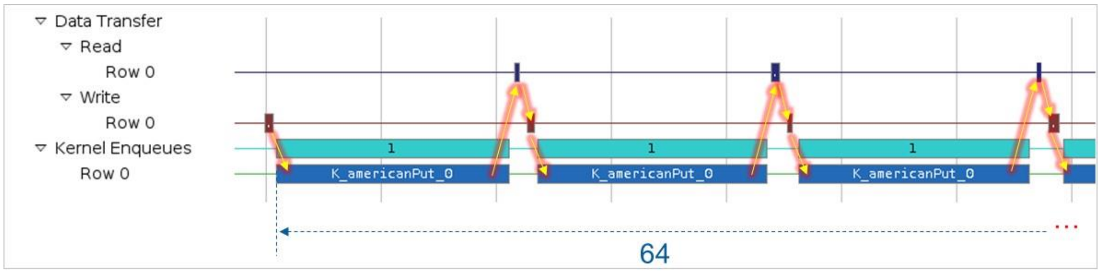
```
HOST-Info: ============================================================
HOST-Info: Step: Run Application
HOST-Info: ============================================================

HOST_Info: Waiting for application to be completed ...

HOST_Info: Test Passed
HOST-Info: Results stored in the HW_Res.txt file ...

HOST-Info: Application Completed

INFO: [SDx-EM 22] [Wall clock time: 19:20, Emulation time: 0.464845 ms] Data transfer
between kernel(s) and global memory(s)
K_americanPut_0_1:m_axi_gmem_0-DDR[1]         RD = 2.000 KB              WR = 0.000 KB
K_americanPut_0_1:m_axi_gmem_1-DDR[1]         RD = 0.000 KB              WR = 0.250 KB
```

Then open Application Timeline report: in the **Assistant** tab, under **P10_hw_Original** / **Emulation-HW / P10_hw_Original-Default** double click on the **Application Timeline** to open it:



Assistant
P00_sw [OpenCL]
P10_hw_Original [OpenCL]
Emulation-SW [Software Emulation]
Emulation-HW [Hardware Emulation]
binary_container_1
P10_hw_Original-Default
Profile Summary [02 Jun 2019 19:20]
Application Timeline [02 Jun 2019 19:20]
System [Hardware]

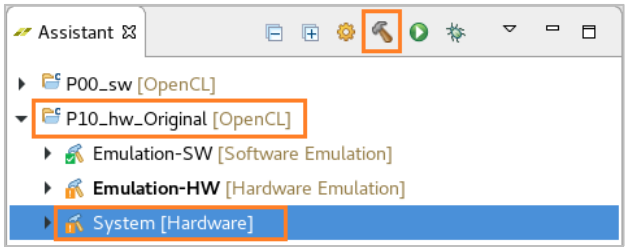
Zoom in and observe a sequential nature of the entire execution process on the Alveo card:



4-3. System Run

At this step of the flow, compile and run the design on the Alveo card. Please note that it may take ~1.5 hours to compile the application.

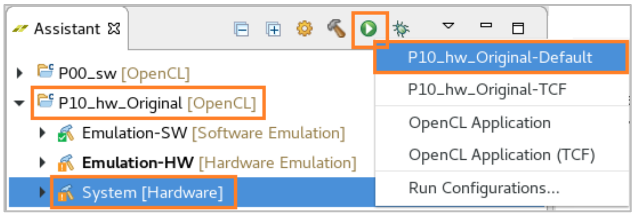
To compile the design for System Run in the **Assistant tab**, under **P10_hw_Original** select **System,** and then press on the 🔨 icon:



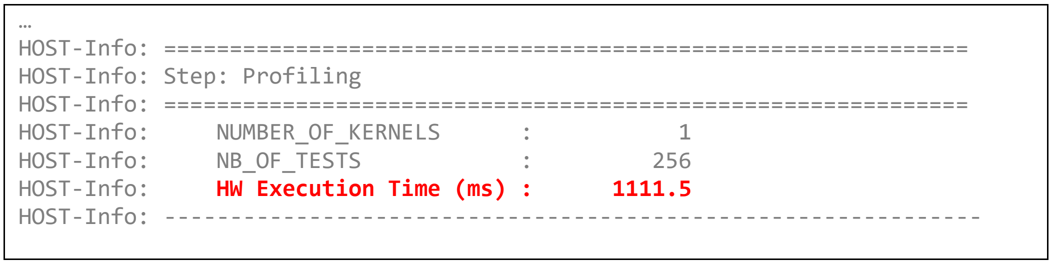
Assistant
P00_sw [OpenCL]
P10_hw_Original [OpenCL]
Emulation-SW [Software Emulation]
Emulation-HW [Hardware Emulation]
System [Hardware]

The compilation process should complete successfully.

To launch System Run, in the **Assistant tab**, under **P10_hw_Original** select **System**, then click on the ▶ icon and select **P10_hw_Original-Default**:



The application execution is successfully completed, and the Alveo execution time is reported in a Console window:



```
…
HOST-Info: ============================================================
HOST-Info: Step: Profiling
HOST-Info: ============================================================
HOST-Info:     NUMBER_OF_KERNELS      :          1
HOST-Info:     NB_OF_TESTS            :        256
HOST-Info:     HW Execution Time (ms) :      1111.5
HOST-Info: -------------------------------------------------------------
```

Comparing CPU (single thread) and the Alveo results, you can see that the Alveo solution outperforms CPU by the factor of **8.9x**:

Design	CPU (ms)	Alveo (ms)	Alveo Gain vs. CPU
Original	**9834.2** *1 Thread*	**1111.5**	**8.9 x**

Alveo: Host Optimization - Data Transfer and Kernel Execution Overlap Step 5

Projects: **P11_hw_Original_HostOpt**

Using the same (original) kernel implementation from the **P10_hw_Original** project, we can improve design performance by overlapping data transfers with Kernel execution. This is achieved by modifying the host code (Host.cpp).

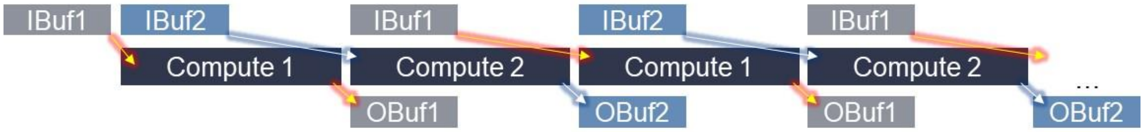
Such an approach is beneficial for processing a large amount of data.

- *Note: in our design, we deal with a small amount of data - 256 test vectors only.*
  - *Therefore, we do not expect significant performance gain by using this method.*
  - *However, we still want to demonstrate this approach when targeting Alveo cards.*

In this approach:

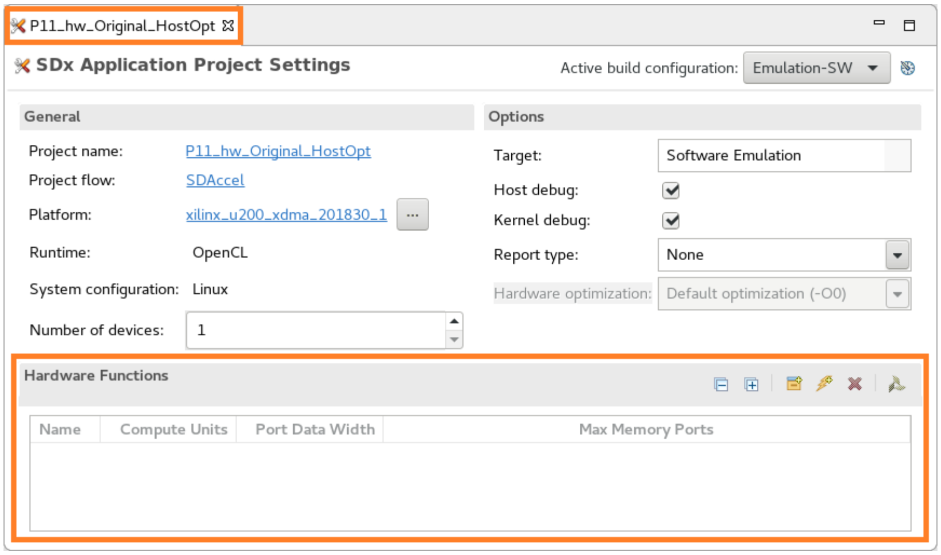
- Data is prefetched in advance for coming computations
- Ping-Pong buffers are used for Input test vectors and Results
- A single compute unit (CU) is used

This method is illustrated in the following figure:



Open the **P11_hw_Original_HostOpt** project.

Please notice that there are no kernels listed in the Hardware Functions:



SDx Application Project Settings

Active build configuration: Emulation-SW

General

Project name: P11_hw_Original_HostOpt

Project flow: SDAccel

Platform: xilinx_u200_xdma_201830_1

Runtime: OpenCL

System configuration: Linux

Number of devices: 1

Options

Target: Software Emulation

Host debug: ☑

Kernel debug: ☑

Report type: None

Hardware optimization: Default optimization (-O0)

Hardware Functions

Name	Compute Units	Port Data Width	Max Memory Ports

This is done to save time by avoiding kernel compilation in SW Emulation, HW Emulation and System Run modes. This is achieved by preconfiguring the **P11_hw_Original_HostOpt-Default** run, where the **binary_container_1.xclbin** file is referenced from the **P10_hw_Original project**.

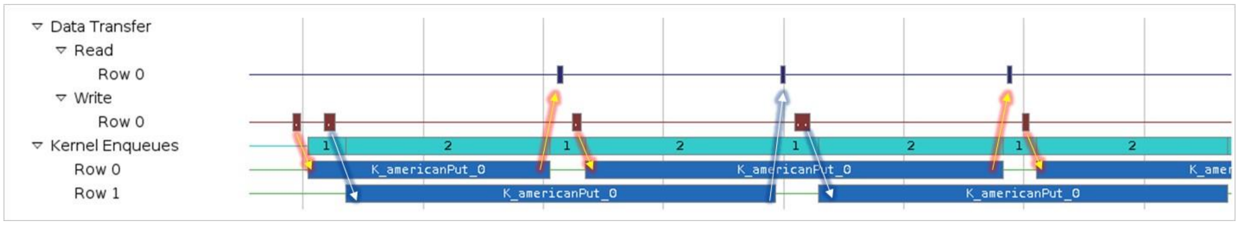
Compile (only the host will be compiled) and run the design in a **SW Emulation** mode to confirm that the design functionality is correct. You should see the following LOG messages:

```
HOST-Info: ============================================================
HOST-Info: Step: Run Application
HOST-Info: ============================================================

HOST_Info: Waiting for application to be completed ...

HOST_Info: Test Passed
...
```

Compile and run the design in a **HW Emulation** mode. You should see the following LOG messages:

```
HOST-Info: ============================================================
HOST-Info: Step: Run Application
HOST-Info: ============================================================

HOST_Info: Waiting for application to be completed ...

HOST_Info: Test Passed
...
```

Then open **Application Timeline** trace report. Zoom in. You should observe an overlap between data transfer and kernel execution:



Compile and run the design in a **System Run** mode. The Alveo execution time is reported in a Console window:

```
HOST-Info: ============================================================
HOST-Info: Step: Profiling
HOST-Info: ============================================================
HOST-Info:     NUMBER_OF_KERNELS      :          1
HOST-Info:     NB_OF_TESTS            :        256
HOST-Info:     HW Execution Time (ms) :      968.5
HOST-Info: -------------------------------------------------------------
```

Comparing CPU (multi-thread) and the Alveo results, you can see that the Alveo solution outperforms CPU by the factor of **1.3x**:

Design	CPU (ms)	Alveo (ms)	Alveo Gain vs. CPU
Original	**9834.2** *1 Thread*	**1111.5**	**8.9 x**
Original (Host Optimization)	**1289.7** *12 Threads*	**968.5**	**1.3 x**

Alveo: Single Kernel Optimization - Data Movement Step 6

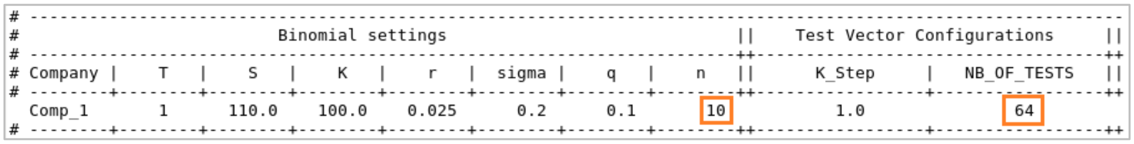
Projects: **P11_hw_Original_HostOpt**, **Popt_12_step1_DataMove**

To achieve significant performance acceleration, we need to replace the sequential execution of our application by a more efficient approach. This can be achieved by

- Changing a way data is transferred between
  - Host and Global Memory
  - Global Memory and Kernel
- Increasing the parallelization level of a computation process

Important notes:

- To achieve faster design iterations and convergence, you are going to use SW Emulation and HW Emulation modes only during the single kernel optimization process. The final acceleration gain will be of course measured on the Alveo card via System Run.
- To improve execution time at the HW Emulation step, the complexity of the test vectors was reduced. The test vectors for the HW Emulation are stored in the **test_config_HW_Emu.txt** file:



```
# --------------------------------------------------------------------------------------------------------------
#                            Binomial settings                                ||     Test Vector Configurations      ||
# ----------------------------------------------------------------------------++-------------------------------------++
# Company |    T    |    S    |    K    |    r    |  sigma  |    q    |    n   ||       K_Step       |   NB_OF_TESTS    ||
# --------+---------+---------+---------+---------+---------+---------+--------++--------------------+-----------------++
  Comp_1       1       110.0     100.0     0.025      0.2       0.1       10            1.0                   64
# --------+---------+---------+---------+---------+---------+---------+--------++--------------------+-----------------++
```

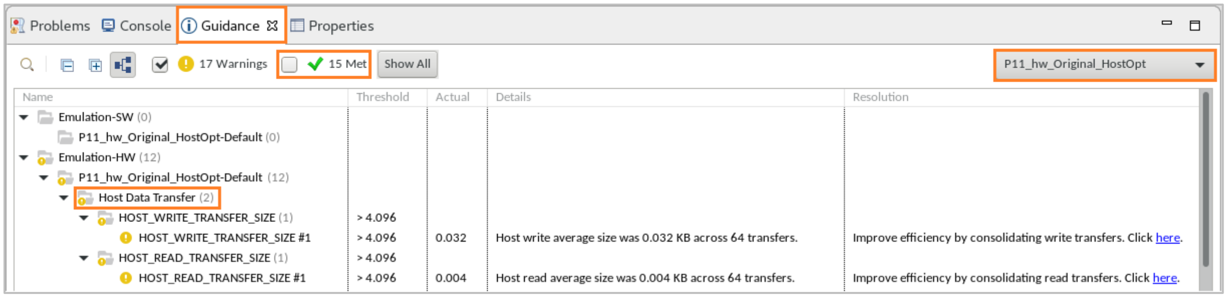
For HW Emulation:

- The number of test vectors was reduced to **64** (from of 256 used for SW Emulation and System Run)
- The height of the Binomial tree was reduced to **10** (from of 1024 used for SW Emulation and System Run)

Before optimizing the design, let's have a look at the suggestions reported by SDAccel flow at the HW Emulation step in the **P11_hw_Original_HostOpt** project. These suggestions could be found on a **Guidance** tab.

6-1. Host ↔ Global Memory Data transfer

Make the **Guidance** tab active, then select the **P11_hw_Original_HostOpt** project in the drop-down menu. Uncheck the rules which met the requirements, to focus on the suggestions only:

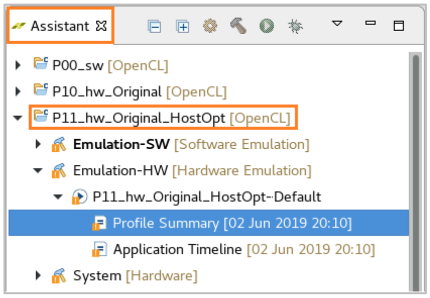


Name	Threshold	Actual	Details	Resolution
Emulation-SW (0)				
P11_hw_Original_HostOpt-Default (0)				
Emulation-HW (12)				
P11_hw_Original_HostOpt-Default (12)				
Host Data Transfer (2)				
HOST_WRITE_TRANSFER_SIZE (1)	> 4.096			
HOST_WRITE_TRANSFER_SIZE #1	> 4.096	0.032	Host write average size was 0.032 KB across 64 transfers.	Improve efficiency by consolidating write transfers. Click here.
HOST_READ_TRANSFER_SIZE (1)	> 4.096			
HOST_READ_TRANSFER_SIZE #1	> 4.096	0.004	Host read average size was 0.004 KB across 64 transfers.	Improve efficiency by consolidating read transfers. Click here.

Navigate to the **Host Data Transfer** section. You will see that SDAccel reports that data transfer between Host and Global memory is done in small chunks of data and suggest to improve efficiency by consolidating read and write transfers.

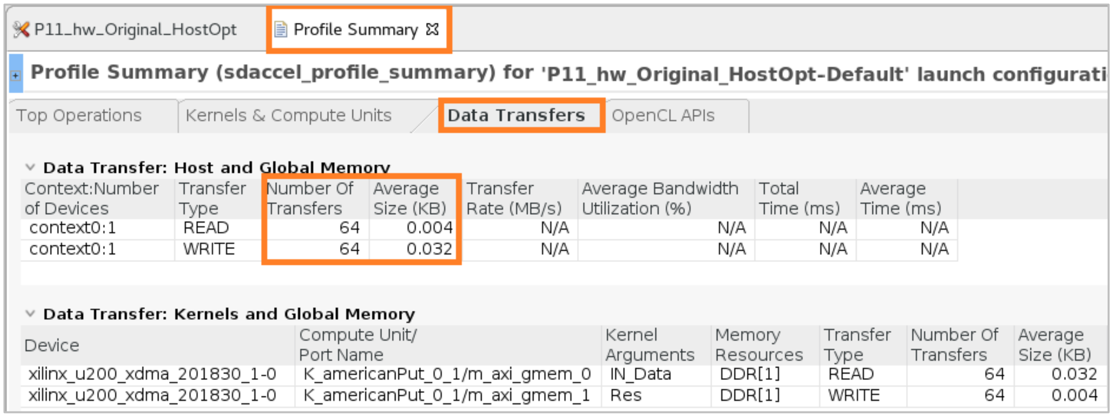
Please note that there are other Host Data Transfer suggestions, which will be reviewed later.

Information about data transfer can be found in the **profile summary** report (generated during application run). To open this report, in the Assistant, under the **P11_hw_Original_HostOpt / Emulation-HW / …** double click on the **Profile Summary**:



- P00_sw [OpenCL]
- P10_hw_Original [OpenCL]
- P11_hw_Original_HostOpt [OpenCL]
  - Emulation-SW [Software Emulation]
  - Emulation-HW [Hardware Emulation]
    - P11_hw_Original_HostOpt-Default
      - Profile Summary [02 Jun 2019 20:10]
      - Application Timeline [02 Jun 2019 20:10]
  - System [Hardware]

In the opened report, navigate to the **Data Transfer** tab. In the **Host and Global Memory** section you will find **Number of Transfers** and **Average Size** for Read and Write operations:



Profile Summary (sdaccel_profile_summary) for 'P11_hw_Original_HostOpt-Default' launch configurati

Data Transfer: Host and Global Memory

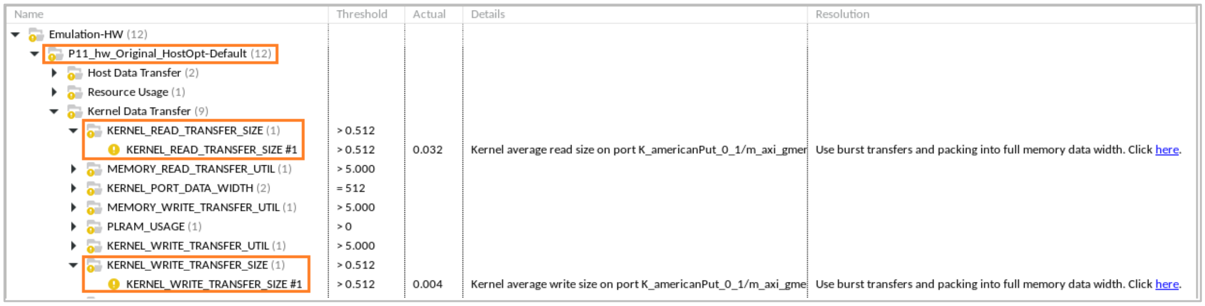
Context:Number of Devices	Transfer Type	Number Of Transfers	Average Size (KB)	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Total Time (ms)	Average Time (ms)
context0:1	READ	64	0.004	N/A	N/A	N/A	N/A
context0:1	WRITE	64	0.032	N/A	N/A	N/A	N/A

Data Transfer: Kernels and Global Memory

Device	Compute Unit/ Port Name	Kernel Arguments	Memory Resources	Transfer Type	Number Of Transfers	Average Size (KB)
xilinx_u200_xdma_201830_1-0	K_americanPut_0_1/m_axi_gmem_0	IN_Data	DDR[1]	READ	64	0.032
xilinx_u200_xdma_201830_1-0	K_americanPut_0_1/m_axi_gmem_1	Res	DDR[1]	WRITE	64	0.004

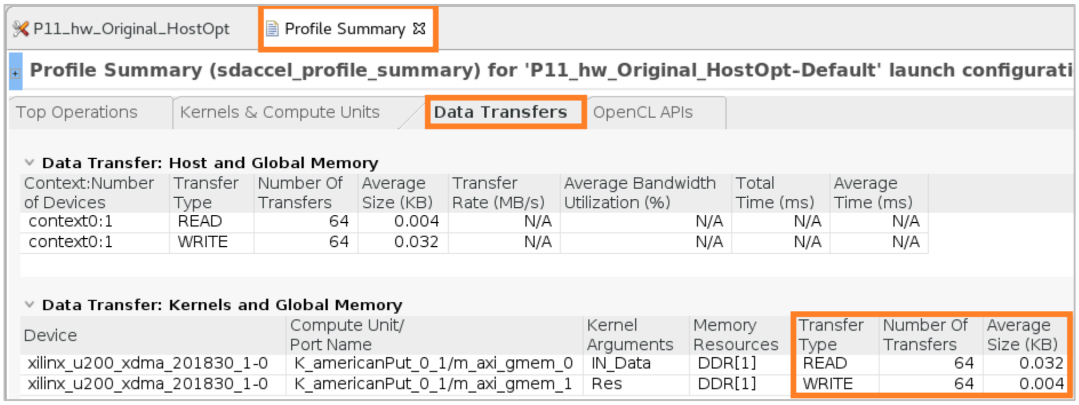
6-2. Global Memory ↔ Kernel Data transfer

If you navigate to the **Kernel Data Transfer** section in the **Guidance** report, then you will find suggestions representing data transfer between Kernel and Global memory. SDAccel suggests to improve efficiency by using Read/Write **Burst** transfers:



Name	Threshold	Actual	Details	Resolution
Emulation-HW (12)				
P11_hw_Original_HostOpt-Default (12)				
Host Data Transfer (2)				
Resource Usage (1)				
Kernel Data Transfer (9)				
KERNEL_READ_TRANSFER_SIZE (1)	> 0.512			
KERNEL_READ_TRANSFER_SIZE #1	> 0.512	0.032	Kernel average read size on port K_americanPut_0_1/m_axi_gmer	Use burst transfers and packing into full memory data width. Click here.
MEMORY_READ_TRANSFER_UTIL (1)	> 5.000			
KERNEL_PORT_DATA_WIDTH (2)	= 512			
MEMORY_WRITE_TRANSFER_UTIL (1)	> 5.000			
PLRAM_USAGE (1)	> 0			
KERNEL_WRITE_TRANSFER_UTIL (1)	> 5.000			
KERNEL_WRITE_TRANSFER_SIZE (1)	> 0.512			
KERNEL_WRITE_TRANSFER_SIZE #1	> 0.512	0.004	Kernel average write size on port K_americanPut_0_1/m_axi_gme	Use burst transfers and packing into full memory data width. Click here.

Information about data transfer between **Kernel and Global memory** can be found in the **profile summary** report as well:



Profile Summary (sdaccel_profile_summary) for 'P11_hw_Original_HostOpt-Default' launch configurati

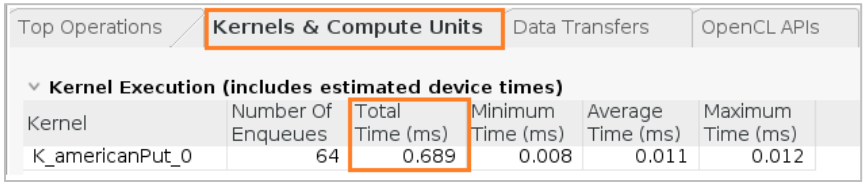
Data Transfer: Host and Global Memory

Context:Number of Devices	Transfer Type	Number Of Transfers	Average Size (KB)	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Total Time (ms)	Average Time (ms)
context0:1	READ	64	0.004	N/A	N/A	N/A	N/A
context0:1	WRITE	64	0.032	N/A	N/A	N/A	N/A

Data Transfer: Kernels and Global Memory

Device	Compute Unit/ Port Name	Kernel Arguments	Memory Resources	Transfer Type	Number Of Transfers	Average Size (KB)
xilinx_u200_xdma_201830_1-0	K_americanPut_0_1/m_axi_gmem_0	IN_Data	DDR[1]	READ	64	0.032
xilinx_u200_xdma_201830_1-0	K_americanPut_0_1/m_axi_gmem_1	Res	DDR[1]	WRITE	64	0.004

Then open **Kernel & Compute Units** tab in the profile summary report and record total estimated time to run a kernel:



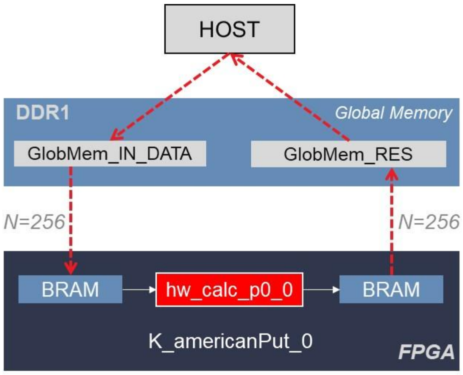
Kernel Execution (includes estimated device times)

Kernel	Number Of Enqueues	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
K_americanPut_0	64	0.689	0.008	0.011	0.012

In order to improve data transfer and therefore, design performance:

- The host should transfer ALL test vectors to Global Memory as a single block of data
- Then Kernel
  - Prefetches ALL test vectors to Local Memory (implemented usually using BRAM resources) in a burst mode
  - Processes ALL test vectors and stores ALL results in Local Memory
  - Transfers all results to Global Memory in Burst Mode
- Finally, Host transfers ALL results from to Host Memory

This can be represented using the following block diagram:



6-3. Open the Popt_12_step1_DataMove project.

To implement this solution, we need to make changes in:

- **Host** code to transfer all data to/from Global Memory using a single API call per write and read operations (see **Host.cpp**):

```
…
// .....................................................................
// Copy ALL test vectors: host_IBuf -> GlobMem_IBuf
// .....................................................................
errCode = clEnqueueMigrateMemObjects(Command_Queue, 1, &(HW_Kernels[k_index].GlobMem_IBuf),
                                     0, 0, NULL,&Mem_wr_event[0]);
...

// .................................................................
// Submit Kernel for execution
// .................................................................
...
errCode = clEnqueueNDRangeKernel(Command_Queue, HW_Kernels[k_index].kernel, 1, NULL,
                                 globalSize, localSize, 1,
                                 &Mem_wr_event[0], &K_exe_event[0]);
...


// .................................................................
// Copy ALL results: GlobMem_OBuf -> host_OBuf
// .................................................................
errCode = clEnqueueMigrateMemObjects(Command_Queue, 1, &(HW_Kernels[k_index].GlobMem_OBuf),
                           CL_MIGRATE_MEM_OBJECT_HOST,
                           1, &K_exe_event[0], &Mem_rd_event[0]);
...
```

- In **K_americanPut_0** kernel function (no changes are required in **hw_calc_p0_0**) (see **K0.cpp**)

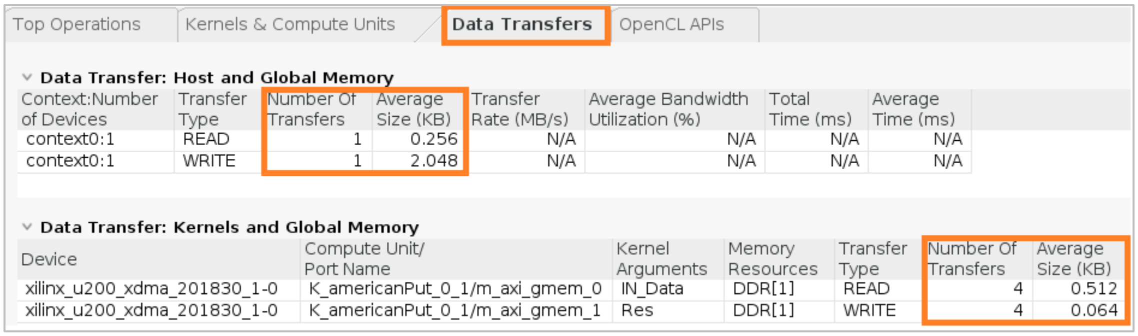
```
…
void K_americanPut_0(t_in_data* IN_Data, float* Res,
                     int Nb_of_Tests, int Start_Index ) {

// ---------------------------------------------------------------------- //
       #pragma HLS INTERFACE s_axilite port=IN_Data        bundle=control
       #pragma HLS INTERFACE s_axilite port=Res            bundle=control
       ...
// ---------------------------------------------------------------------- //

    t_in_data  tmp_IN_Data[CONST_MAX_NB_OF_TESTS];
    #pragma HLS DATA_PACK variable=tmp_IN_Data

    float      tmp_Res[CONST_MAX_NB_OF_TESTS];

    // -------------------------------------
    // Transfer data: Global Memory -> BRAM
    // -------------------------------------
    read_in_data_loop: for (int i = 0; i < Nb_of_Tests; i++)
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
        tmp_IN_Data[i] = IN_Data[Start_Index+i];

    // -------------------------------------
    // Calculate
    // -------------------------------------
    calcualte: for (int i = 0; i < Nb_of_Tests; i++) {
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
        tmp_Res[i] = hw_calc_p0_0(tmp_IN_Data[i]);
    }

    // -------------------------------------
    // Transfer data: BRAM -> Global Memory
    // -------------------------------------
    write_out_data_loop: for (int i = 0; i < Nb_of_Tests; i++)
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
        Res[Start_Index+i] = tmp_Res[i];
}
```

In the above code:

- We added two input arguments: **Nb_of_Test** and **Start_Index**. The first one defines the number of test vectors kernel need to process. The last one will be used at later steps of the flow when multiple CUs will be implemented on the Alveo card.
- We introduced local memory: **tmp_IN_Data**, **tmp_Res**.
- In the **read_in_data_loop** loop, we transfer input data from Global to Local memory in a burst mode.
- The **calculate** loop processes all test vectors and stores them in local memory.
- And finally, the **write_out_data_loop** loop transfers all results to the Global memory in a burst mode.

Compile and run the project in **SW Emulation** mode to check its functionality.
Then compile and run the project in **HW Emulation** mode and then open Profile Summary report.

You will see that the number of transfers was significantly reduced:



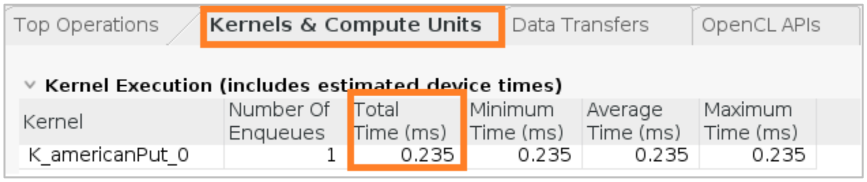
Data Transfer: Host and Global Memory

Context:Number of Devices	Transfer Type	Number Of Transfers	Average Size (KB)	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Total Time (ms)	Average Time (ms)
context0:1	READ	1	0.256	N/A	N/A	N/A	N/A
context0:1	WRITE	1	2.048	N/A	N/A	N/A	N/A

Data Transfer: Kernels and Global Memory

Device	Compute Unit/ Port Name	Kernel Arguments	Memory Resources	Transfer Type	Number Of Transfers	Average Size (KB)
xilinx_u200_xdma_201830_1-0	K_americanPut_0_1/m_axi_gmem_0	IN_Data	DDR[1]	READ	4	0.512
xilinx_u200_xdma_201830_1-0	K_americanPut_0_1/m_axi_gmem_1	Res	DDR[1]	WRITE	4	0.064

Then check Kernel execution time (estimated):

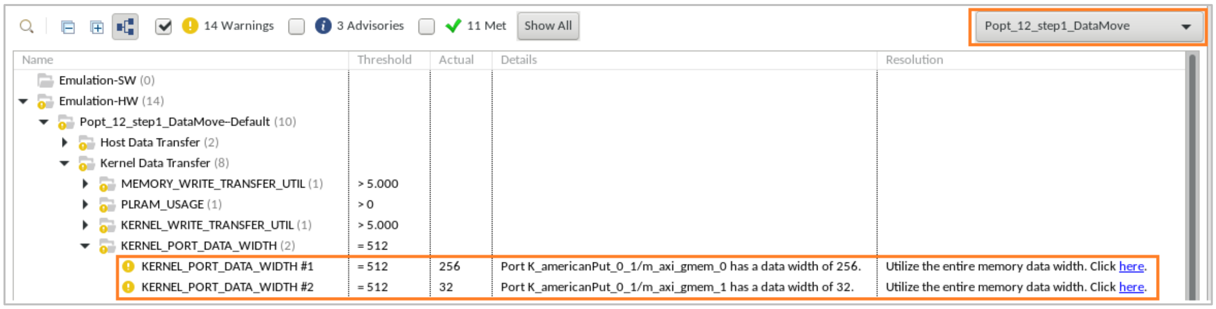


Kernel Execution (includes estimated device times)

Kernel	Number Of Enqueues	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
K_americanPut_0	1	0.235	0.235	0.235	0.235

You can see that it was reduced by the factor of **3x** compared to the **0.689 ms** obtained in the **P11_hw_Original_HostOpt** project.

Now let's open **Guidance** report and review a couple of additional suggestions from the SDAccel flow. Please ensure that the **Popt_12_step1_DataMove** project is selected:



Name	Threshold	Actual	Details	Resolution
Emulation-SW (0)				
Emulation-HW (14)				
Popt_12_step1_DataMove-Default (10)				
Host Data Transfer (2)				
Kernel Data Transfer (8)				
MEMORY_WRITE_TRANSFER_UTIL (1)	> 5.000			
PLRAM_USAGE (1)	> 0			
KERNEL_WRITE_TRANSFER_UTIL (1)	> 5.000			
KERNEL_PORT_DATA_WIDTH (2)	= 512			
KERNEL_PORT_DATA_WIDTH #1	= 512	256	Port K_americanPut_0_1/m_axi_gmem_0 has a data width of 256.	Utilize the entire memory data width. Click here.
KERNEL_PORT_DATA_WIDTH #2	= 512	32	Port K_americanPut_0_1/m_axi_gmem_1 has a data width of 32.	Utilize the entire memory data width. Click here.

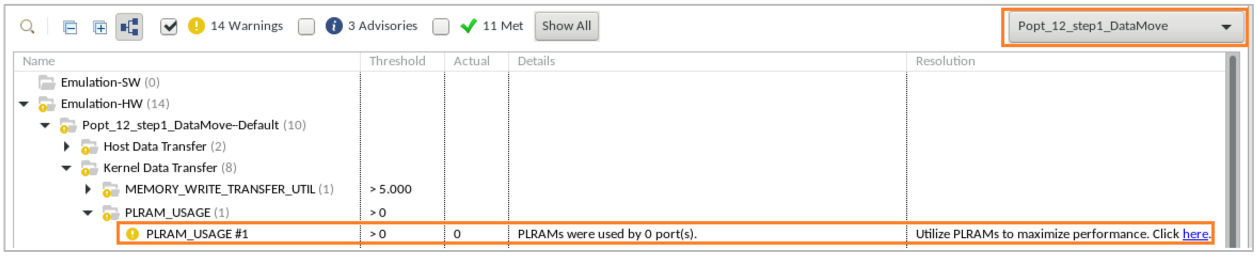
DDR data bus enables the transfer of **512** bits of data simultaneously. The guidance points out that for

- READ operation we use **256** bits (one test vector transferred)
- WRITE operation we use **32** bits (one result transferred)

and suggests using **512** bits to reduce the number of transfers further.

This can be achieved by using vectorization approach (ex: to transfer 16 floats (results) simultaneously). This technique is not considered in this example. Please refer to **UG1207** Optimization Guide for more information.

In addition, we have the following suggestion:



Name	Threshold	Actual	Details	Resolution
Emulation-SW (0)				
Emulation-HW (14)				
Popt_12_step1_DataMove-Default (10)				
Host Data Transfer (2)				
Kernel Data Transfer (8)				
MEMORY_WRITE_TRANSFER_UTIL (1)	> 5.000			
PLRAM_USAGE (1)	> 0			
PLRAM_USAGE #1	> 0	0	PLRAMs were used by 0 port(s).	Utilize PLRAMs to maximize performance. Click here.

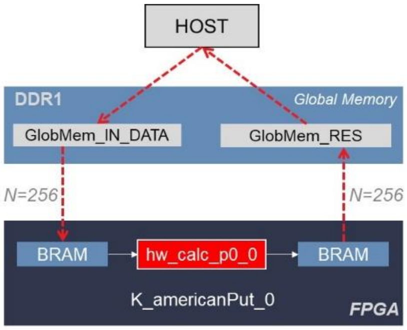
Global Memory can be implemented using **BRAM** resources instead of DDR (called **PLRAM**), providing faster data access, which is suggested here. This technique is not considered in this example. Please refer to UG1207 Optimization Guide for more information.

Alveo: Single Kernel Optimization - Running 4 Functions in Parallel Step 7

Projects: **Popt_12_step2_FUNx4**, **Popt_12_step3_ARRAY_PARTITION**

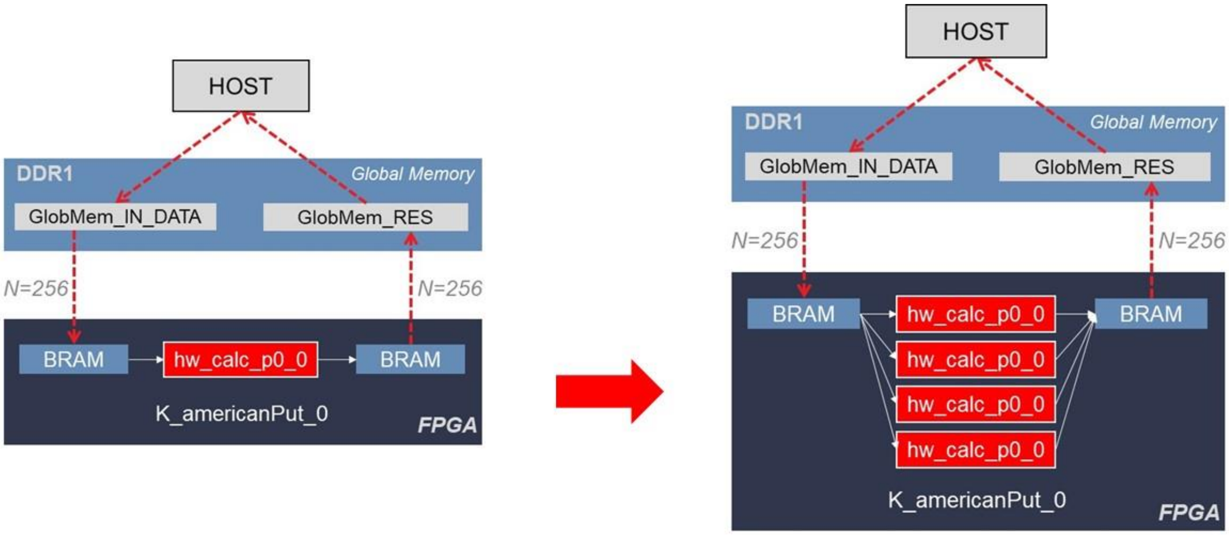
In the previous **Popt_12_step2_DataMove** project we processed a single test vector per iteration:

```
void K_americanPut_0(t_in_data* IN_Data, float* Res,
                     int Nb_of_Tests, int Start_Index ) {
...
    calcualte: for (int i = 0; i < Nb_of_Tests; i++) {
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
        tmp_Res[i] = hw_calc_p0_0(tmp_IN_Data[i]);
    }
...
```



7-1. Open the Popt_12_step2_FUNx4 project.

To further boost performance, we can process several test vectors simultaneously. In our design, we chose to process four of them, which should provide an additional **4x** acceleration factor. This transformation is presented in the following figure:

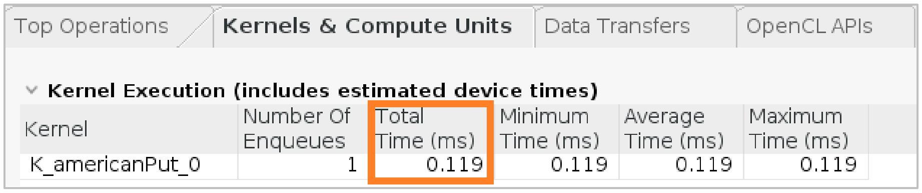


For that we need to slightly modify the Kernel code by introducing an additional loop (**calculate_sub_i**) with 4 iterations and **unroll** it:

```
void K_americanPut_0(t_in_data* IN_Data, float* Res,
                     int Nb_of_Tests, int Start_Index ) {
...

    calculate_i: for (int i = 0; i < Nb_of_Tests/4; i++) {
        #pragma HLS LOOP_TRIPCOUNT min=25 max=25 avg=25
        calculate_sub_i: for (int sub_i = 0; sub_i < 4; sub_i++) {
            #pragma HLS UNROLL
            tmp_Res[i*4+sub_i] = hw_calc_p0_0(tmp_IN_Data[i*4+sub_i]);
        }
    }
...
```

Compile and run the project in **SW Emulation** mode to check its functionality.

After they compile and run the project in **HW Emulation** mode and then open Profile Summary report.

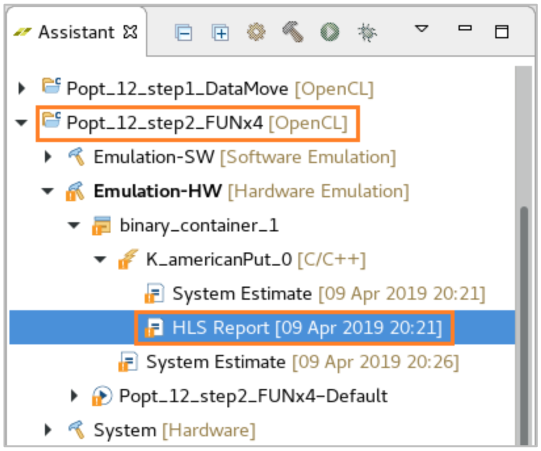


Kernel Execution (includes estimated device times)

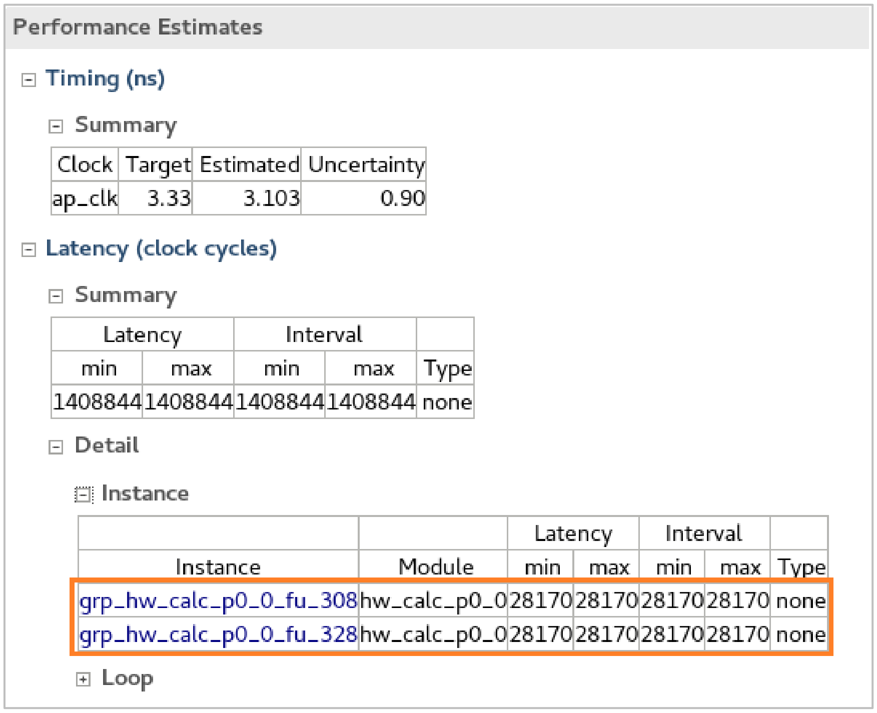
Kernel	Number Of Enqueues	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
K_americanPut_0	1	0.119	0.119	0.119	0.119

Comparing kernel execution time (**0.119** ms) with the one obtained in the **Popt_12_step2_DataMove** project (**0.235** ms), we can observe that **2x** acceleration was only achieved instead of expected **4x**.

Using Assistant, open **HLS synthesis** report which provides more detailed information regarding Kernel implementation:


Assistant
Popt_12_step1_DataMove [OpenCL]
Popt_12_step2_FUNx4 [OpenCL]
Emulation-SW [Software Emulation]
Emulation-HW [Hardware Emulation]
binary_container_1
K_americanPut_0 [C/C++]
System Estimate [09 Apr 2019 20:21]
HLS Report [09 Apr 2019 20:21]
System Estimate [09 Apr 2019 20:26]
Popt_12_step2_FUNx4-Default
System [Hardware]

In the HLS report, navigate to the **Performance Estimates / Instance** section:



Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.33	3.103	0.90

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
1408844	1408844	1408844	1408844	none

⊟ Detail

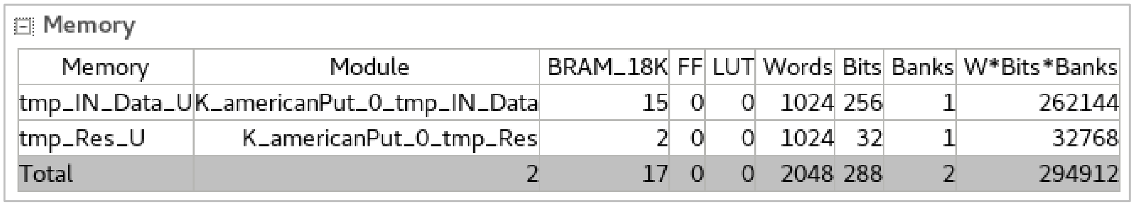
⊟ Instance

		Latency		Interval		
Instance	Module	min	max	min	max	Type
grp_hw_calc_p0_0_fu_308	hw_calc_p0_0	28170	28170	28170	28170	none
grp_hw_calc_p0_0_fu_328	hw_calc_p0_0	28170	28170	28170	28170	none

⊞ Loop

In the kernel code, we asked to fully unroll a 4-iteration loop, expecting that **hw_calc_p0_0** will be called four times simultaneously. However, as you can see from the HLS report, only two function calls were implemented.

The performance bottleneck is in accessing local memory: **tmp_IN_Data** and **tmp_Res** arrays. These arrays are implemented as BRAMs, which can be seen from the **Utilization Estimates** section of the HLS report:



⊟ Memory

Memory	Module	BRAM_18K	FF	LUT	Words	Bits	Banks	W*Bits*Banks
tmp_IN_Data_U	K_americanPut_0_tmp_IN_Data	15	0	0	1024	256	1	262144
tmp_Res_U	K_americanPut_0_tmp_Res	2	0	0	1024	32	1	32768
Total	2	17	0	0	2048	288	2	294912

7-2. Open the Popt_12_step3_ARRAY_PARTITION project.

Architecturally, a BRAM has 2 ports and therefore enables to simultaneously read/write 2 pieces of information (maximum). Therefore, SDAccel compiler by default can implement only two **hw_calc_p0_0** function calls instead of four. In order to implement four function calls, the algorithm should have the possibility to read/write 4 pieces of data from/to **tmp_IN_Data** and **tmp_Res** simultaneously. This can be achieved via restructuring how data is stored in **tmp_IN_Data** and **tmp_Res**, by using the **ARRAY_PARTITION** pragma (please refer to the UG1207 and UG1253 users guide for more information).

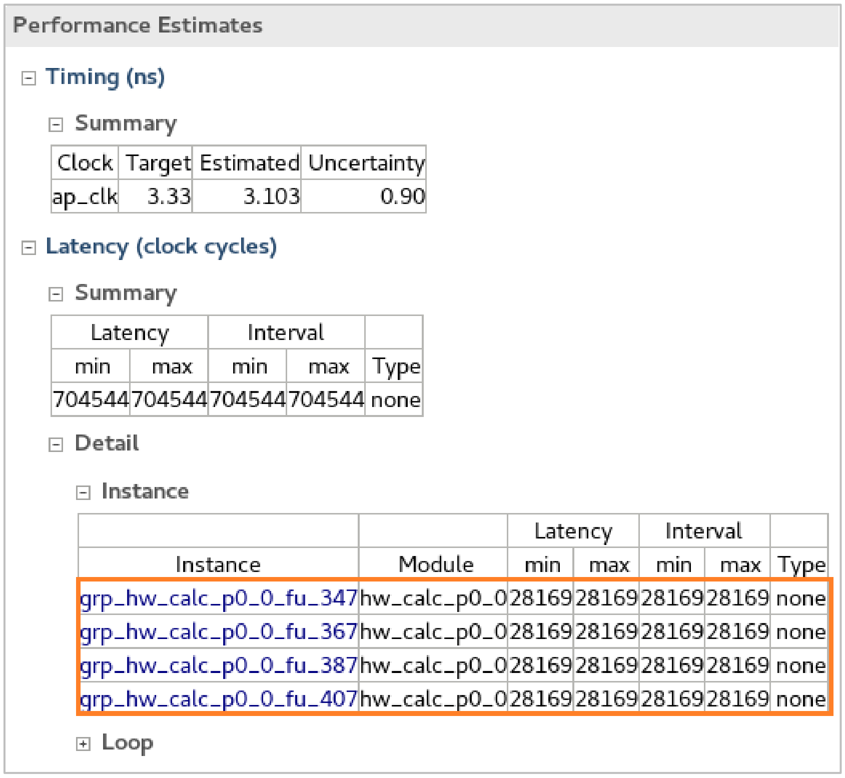
According to the Binomial algorithm, the **tmp_IN_Data** and **tmp_Res** arrays should be partitioned using a **cyclic** mode with the factor of **2** (K0.cpp):

```
void K_americanPut_0(t_in_data* IN_Data, float* Res,
                     int Nb_of_Tests, int Start_Index ) {
...
    t_in_data  tmp_IN_Data[CONST_MAX_NB_OF_TESTS];
    #pragma HLS DATA_PACK variable=tmp_IN_Data
    #pragma HLS ARRAY_PARTITION variable=tmp_IN_Data cyclic factor=2 dim=1

    float      tmp_Res[CONST_MAX_NB_OF_TESTS];
    #pragma HLS ARRAY_PARTITION variable=tmp_Res     cyclic factor=2 dim=1
...

    calculate_i: for (int i = 0; i < Nb_of_Tests/4; i++) {
        #pragma HLS LOOP_TRIPCOUNT min=25 max=25 avg=25
        calculate_sub_i: for (int sub_i = 0; sub_i < 4; sub_i++) {
            #pragma HLS UNROLL
            tmp_Res[i*4+sub_i] = hw_calc_p0_0(tmp_IN_Data[i*4+sub_i]);
        }
    }
...
```

Compile and run the project in **SW Emulation** mode to check its functionality.

Then compile and run the project in **HW Emulation** mode.

Open **HLS report**:



Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.33	3.103	0.90

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
704544	704544	704544	704544	none

⊟ Detail

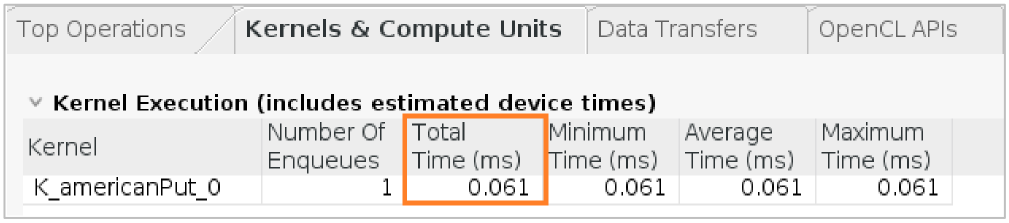
⊟ Instance

		Latency		Interval		
Instance	Module	min	max	min	max	Type
grp_hw_calc_p0_0_fu_347	hw_calc_p0_0	28169	28169	28169	28169	none
grp_hw_calc_p0_0_fu_367	hw_calc_p0_0	28169	28169	28169	28169	none
grp_hw_calc_p0_0_fu_387	hw_calc_p0_0	28169	28169	28169	28169	none
grp_hw_calc_p0_0_fu_407	hw_calc_p0_0	28169	28169	28169	28169	none

⊞ Loop

In this version, four **hw_calc_p0_0** function calls were implemented, and as a consequence, kernel latency was reduced from 1408884 to 704544.

Open **Profile summary** report:



Kernel Execution (includes estimated device times)

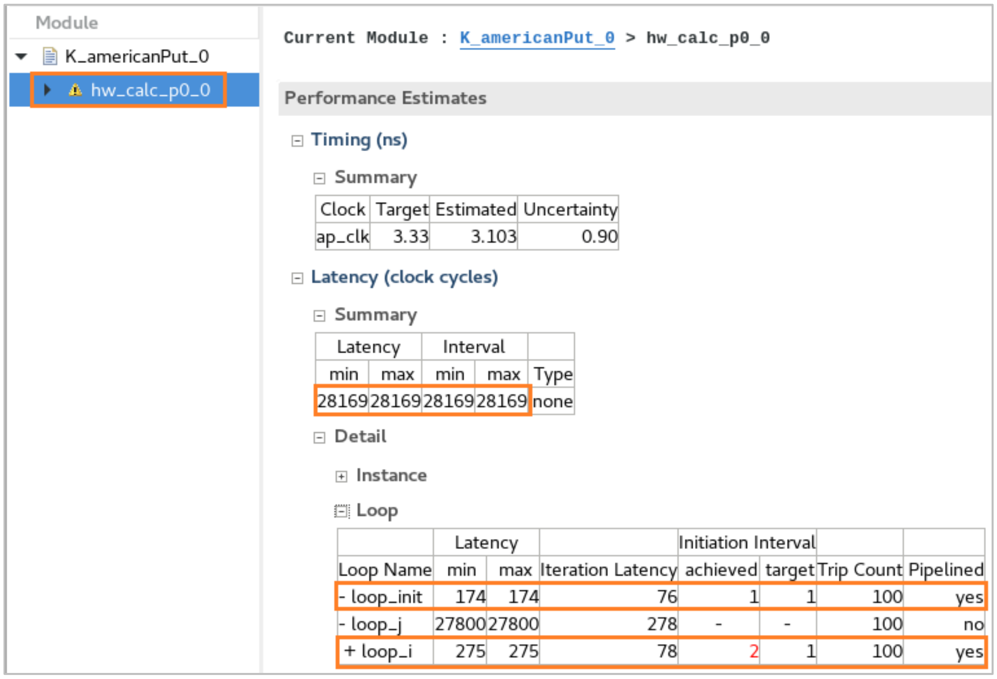
Kernel	Number Of Enqueues	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
K_americanPut_0	1	0.061	0.061	0.061	0.061

Comparing kernel execution time (**0.061** ms) with the one obtained in the **Popt_12_step2_DataMove** project (**0.235** ms), we can observe that we were able to achieve **4x** of acceleration.

Alveo: Single Kernel Optimization - Additional Loops Unrolling Step 8

Projects: **Popt_12_step4_UNROLL**

Open HLS report generated for the **Popt_12_step3_ARRAY_PARTITION** project, navigate to the **hw_calc_p0_0** function and then expand the **Loop** subsection in **Performance Estimates**:



Current Module : K_americanPut_0 > hw_calc_p0_0

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.33	3.103	0.90

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
28169	28169	28169	28169	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- loop_init	174	174	76	1	1	100	yes
- loop_j	27800	27800	278	-	-	100	no
+ loop_i	275	275	78	2	1	100	yes

The **loop_init** and **loop_i** loops are described in the following code (K0.cpp in Popt_12_step3_ARRAY_PARTITION):

```
float hw_calc_p0_0 (t_in_data in_d) {

    float p[CONST_MAX_TREE_HEIGHT];
    ...
    loop_init: for (int i = 0; i < n; i++) {
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
        p[i] = K - S * powf(up,(2*i - n));
        if (p[i] < 0) p[i] = 0;
    }

    loop_j: for (int j = n-1; j > 0; j--) {
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100

        loop_i: for (int i = 0; i < j; i++) {
            #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
            p[i] = p0 * p[i+1] + p1 * p[i];
            exercise = K - S * powf(up,(2*i - j));
            if (p[i] < exercise) p[i] = exercise;
        }
    }
...
```

According to the HLS report, the **loop_init** and **loop_i** loops have a trip count of **100**. This means that they deal with a single set of **p** values per iteration, where **p** is a Local memory implemented as BRAM with two RW ports.

We can partially unroll these loops with a factor of **two** to take advantage of the two RW BRAM ports and as a consequence, increase performance (limiting the increase of FPGA resources).

Open the **Popt_12_step4_UNROLL** project.

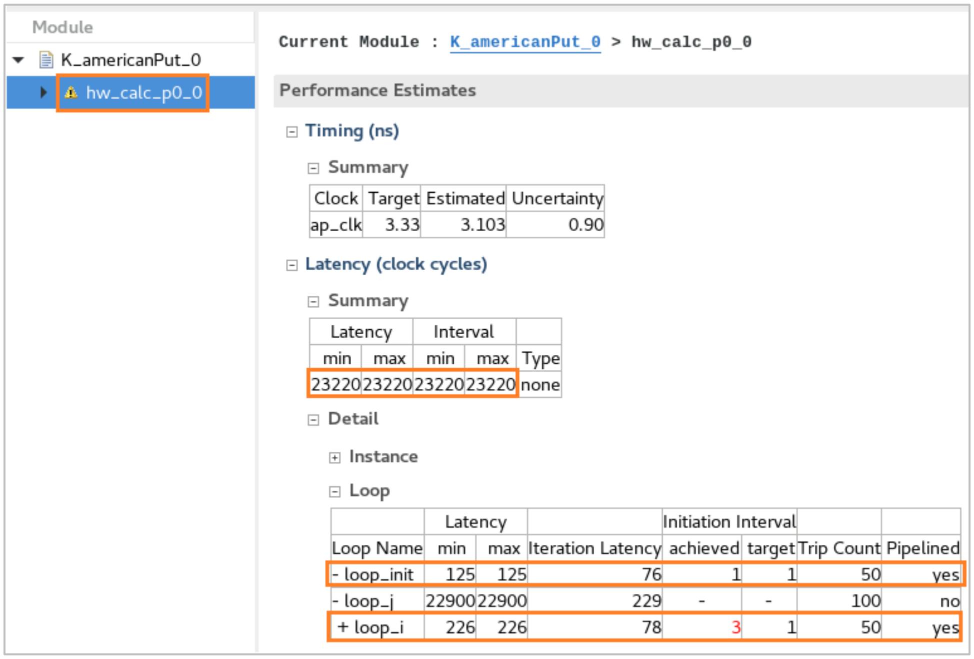
Then open the **K0.cpp** file, and observe how we applied the **UNROLL** pragmas :

```
float hw_calc_p0_0 (t_in_data in_d) {

    float p[CONST_MAX_TREE_HEIGHT];
    ...
    loop_init: for (int i = 0; i < n; i++) {
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
        #pragma HLS UNROLL factor=2
        p[i] = K - S * powf(up,(2*i - n));
        if (p[i] < 0) p[i] = 0;
    }

    loop_j: for (int j = n-1; j > 0; j--) {
        #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100

        loop_i: for (int i = 0; i < j; i++) {
            #pragma HLS LOOP_TRIPCOUNT min=100 max=100 avg=100
            #pragma HLS UNROLL factor=2
            p[i] = p0 * p[i+1] + p1 * p[i];
            exercise = K - S * powf(up,(2*i - j));
            if (p[i] < exercise) p[i] = exercise;
        }
    }
...
```

Compile and run the project in **SW Emulation** mode to check its functionality.

Then compile and run the project in **HW Emulation** mode.

Open **HLS report**:



Current Module : K_americanPut_0 > hw_calc_p0_0

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.33	3.103	0.90

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
23220	23220	23220	23220	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- loop_init	125	125	76	1	1	50	yes
- loop_j	22900	22900	229	-	-	100	no
+ loop_i	226	226	78	3	1	50	yes

As a result of the applied optimization, the latency of the **hw_calc_p0_0** function was reduced by **17%** (from 28169 to 23220).

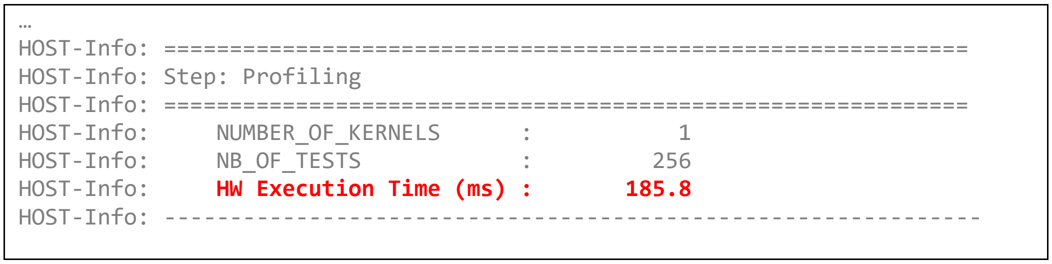
Alveo: Single Kernel Optimization - System Run Results Step 9

Projects: **P12_hw_KOpt**

Source files obtained in the **Popt_12_step4_UNROLL** project were propagated to the **P12_hw_KOpt** one to compile the design for System run and get Alveo performance results.

Open the **P12_hw_KOpt** project. Compile and run it for **System Run**. Please note that it may take about 2 hours to complete the compilation process.

The Alveo execution time is reported in a Console window:



```
…
HOST-Info: ==============================================================
HOST-Info: Step: Profiling
HOST-Info: ==============================================================
HOST-Info:     NUMBER_OF_KERNELS      :          1
HOST-Info:     NB_OF_TESTS            :        256
HOST-Info:     HW Execution Time (ms) :      185.8
HOST-Info: --------------------------------------------------------------
```

Comparing CPU (multi-thread) and the Alveo results, you can see that the optimized Alveo solution outperforms CPU by the factor of **6.9x**:

Design	CPU (ms)	Alveo (ms)	Alveo Gain vs. CPU
Original	**9834.2** *1 Thread*	**1111.5**	**8.9 x**
Original (Host Optimization)	**1289.7** *12 Threads*	**968.5**	**1.3 x**
Kernel Optimization		**185.8**	**6.9x**

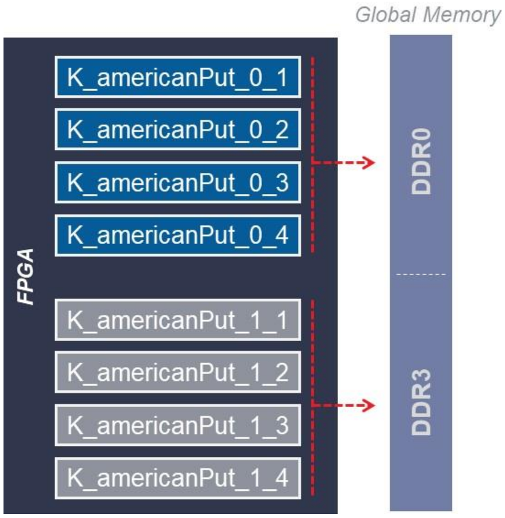
Alveo: Using Multiple Compute Units and Different DDRs Step 10

Projects: **P13_hw_KOpt_8CU_2DDRs, P14_hw_KOpt_12CU_3DDRs**

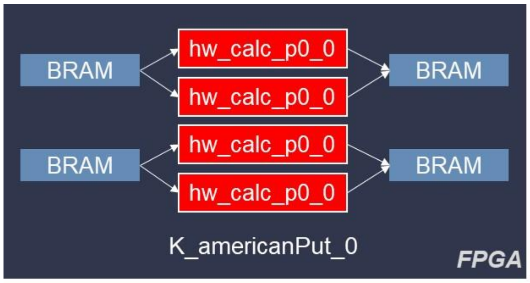
To further increase performance, we can use another technique and instantiate the optimized Kernel multiple times. In addition, we can use different DDR channels to accelerate further data movement, which is very beneficial for large data transfers.

10-1. Open the P13_hw_KOpt_8CU_2DDRs

In this project, we will implement **8 CUs**. Four of the CUs will be connected to the **DDR0** channel and the other ones to **DDR3,** as shown in the following figure:



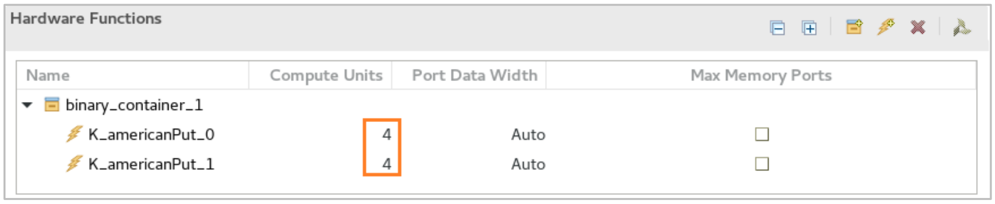
Taking into account that each CU is capable of processing 4 test vectors simultaneously:



the entire solution will be able to deal with 32 test vectors in parallel.

To implement such solution we have done the following changes with regards to a Kernel:

- We Created two identical kernels: **K_americanPut_0** (K0.cpp) and **K_americanPut_1** (K1.cpp) and added them to the Hardware Functions
- For each kernel we created 4 CUs:



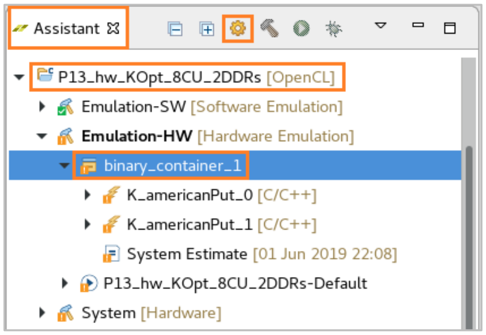
Hardware Functions

Name	Compute Units	Port Data Width	Max Memory Ports
binary_container_1			
K_americanPut_0	4	Auto	☐
K_americanPut_1	4	Auto	☐

- Then we instructed SDAccel compiler how to connect the kernels to the DDR0 and DDR3 banks. This is done by specifying the following linker options:

```
--sp K_americanPut_0_1.IN_Data:DDR[0] -sp K_americanPut_0_1.Res:DDR[0]
--sp K_americanPut_0_2.IN_Data:DDR[0] -sp K_americanPut_0_2.Res:DDR[0]
--sp K_americanPut_0_3.IN_Data:DDR[0] -sp K_americanPut_0_3.Res:DDR[0]
--sp K_americanPut_0_4.IN_Data:DDR[0] -sp K_americanPut_0_4.Res:DDR[0]
--sp K_americanPut_1_1.IN_Data:DDR[3] -sp K_americanPut_1_1.Res:DDR[3]
--sp K_americanPut_1_2.IN_Data:DDR[3] -sp K_americanPut_1_2.Res:DDR[3]
--sp K_americanPut_1_3.IN_Data:DDR[3] -sp K_americanPut_1_3.Res:DDR[3]
--sp K_americanPut_1_4.IN_Data:DDR[3] -sp K_americanPut_1_4.Res:DDR[3]
```

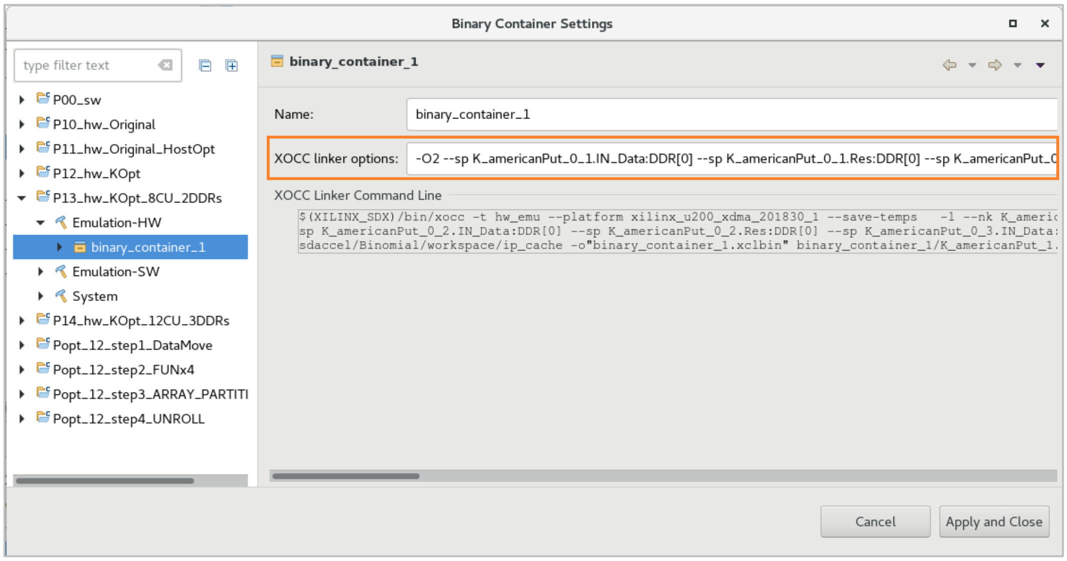
To access these options, in the **Assistant** select **binary_container_1** under **Emulation-HW** or **System** and the press the ⚙ icon:



Assistant

P13_hw_KOpt_8CU_2DDRs [OpenCL]
Emulation-SW [Software Emulation]
Emulation-HW [Hardware Emulation]
binary_container_1
K_americanPut_0 [C/C++]
K_americanPut_1 [C/C++]
System Estimate [01 Jun 2019 22:08]
P13_hw_KOpt_8CU_2DDRs-Default
System [Hardware]

You will see:



Name: binary_container_1

XOCC linker options: -O2 --sp K_americanPut_0_1.IN_Data:DDR[0] --sp K_americanPut_0_1.Res:DDR[0] --sp K_americanPut_0

XOCC Linker Command Line

```
$(XILINX_SDX)/bin/xocc -t hw_emu --platform xilinx_u200_xdma_201830_1 --save-temps  -l --nk K_americ
sp K_americanPut_0_2.IN_Data:DDR[0] --sp K_americanPut_0_2.Res:DDR[0] --sp K_americanPut_0_3.IN_Data:
sdaccel/Binomial/workspace/ip_cache -o"binary_container_1.xclbin" binary_container_1/K_americanPut_1.
```

In addition, we need to instruct the host code to which DDR banks the input data must be transferred and from which DDR banks the results should be obtained. This is done by using a Xilinx **cl_ext.h** extension (see **Host.cpp**):

- Please refer to the **UG1207** user guide for more information.

```
...
// ...................................................................
// Configure DDRs (using Xilinx Extension)
// Note: DDR Banks should be set for each implementation strategy
// ...................................................................
for (int i=0; i<(SW_HW_Config).NB_OF_KERNELS; i++) {
        HW_Kernels[i].GlobMem_IBuf_EXT.obj   = HW_Kernels[i].host_IBuf;
        HW_Kernels[i].GlobMem_IBuf_EXT.param = 0;
        HW_Kernels[i].GlobMem_OBuf_EXT.obj   = HW_Kernels[i].host_OBuf;
        HW_Kernels[i].GlobMem_OBuf_EXT.param = 0;
}

HW_Kernels[0].GlobMem_IBuf_EXT.flags  = XCL_MEM_DDR_BANK0;
HW_Kernels[0].GlobMem_OBuf_EXT.flags  = XCL_MEM_DDR_BANK0;
HW_Kernels[1].GlobMem_IBuf_EXT.flags  = XCL_MEM_DDR_BANK3;
HW_Kernels[1].GlobMem_OBuf_EXT.flags  = XCL_MEM_DDR_BANK3;
...
```

Run **SW Emulation** and **HW Emulation** to validate the design.

Then compile and run the design for **System Run** (please note that it may take about 4 hours to implement the design). The Alveo execution time is reported in a Console window:

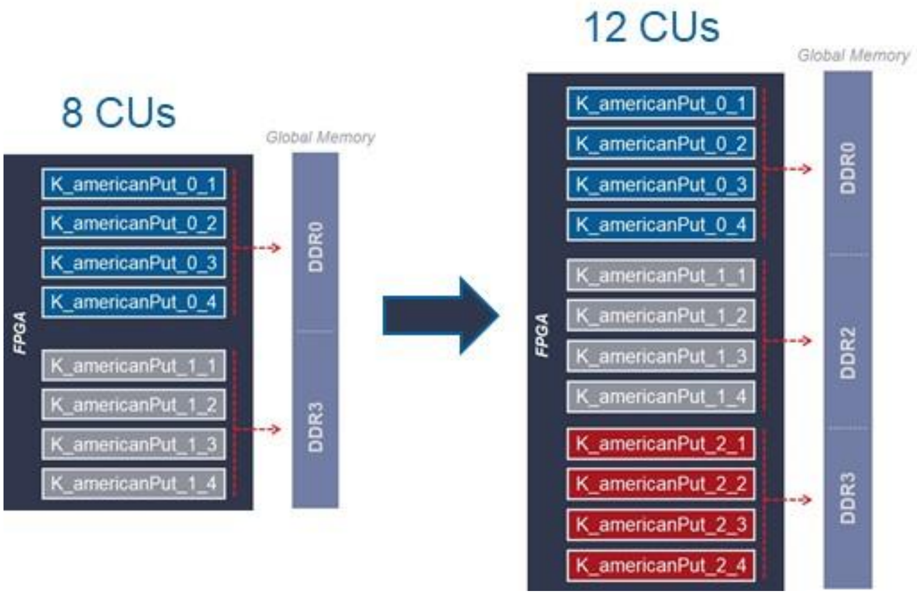
```
…
HOST-Info: ==============================================================
HOST-Info: Step: Profiling
HOST-Info: ==============================================================
HOST-Info:     NUMBER_OF_KERNELS      :          1
HOST-Info:     NB_OF_TESTS            :        256
HOST-Info:     HW Execution Time (ms) :       24.0
HOST-Info: ---------------------------------------------------------------
```

Comparing CPU (multi-thread) and the Alveo results, you can see that the optimized Alveo solution outperforms CPU by the factor of **53.7x**:

Design	CPU (ms)	Alveo (ms)	Alveo Gain vs. CPU
Original	**9834.2** *1 Thread*	**1111.5**	**8.9 x**
Original (Host Optimization)	**1289.7** *12 Threads*	**968.5**	**1.3 x**
Kernel Optimization		**185.8**	**6.9x**
8 CUs and 2 DDRs		**24.0**	**53.7x**

10-2. Open the P14_hw_KOpt_12CU_3DDRs project

In this project we further increase the number of CUs up to 12 and connected them to them three DDRs as shown in the following figure:



Alveo-base implementationd shows additional performance improvements:

```
…
HOST-Info: ============================================================
HOST-Info: Step: Profiling
HOST-Info: ============================================================
HOST-Info:     NUMBER_OF_KERNELS      :          1
HOST-Info:     NB_OF_TESTS            :        256
HOST-Info:     HW Execution Time (ms) :       18.5
HOST-Info: -------------------------------------------------------------
```

Comparing CPU (multi-thread) and the Alveo results, you can see that the optimized Alveo solution outperforms CPU by the factor of **69.7x**.

Comparing original Alveo performance (1111.5 ms) vs. the latest one (18.5 ms), we can see a **60.1x** improvement in the overall performance

Design	CPU (ms)	Alveo (ms)	Alveo Gain vs CPU
Original	**9834.2** *1 Thread*	**1111.5**	**8.9 x**
Original (Host Optimization)	**1289.7** *12 Threads*	**968.5**	**1.3 x**
Kernel Optimization		**185.8**	**6.9x**
8 CUs and 2 DDRs **12 CUs and 3 DDRs**		**24.0** **18.5**	**53.7x** **69.7x**

Please note that this design can be further optimized to achieve additional performance improvements.

Conclusion:

In this tutorial by using a Binomial Options Pricing Model we:

- Introduced SDAccel flow and various analysis tools and optimization techniques allowing you to improve design performance significantly
- Demonstrated performance advantage of the Xilinx Alveo-based solution vs. the CPU one

Projects Setup: Detailed Steps Appendix A

The reference design consists of several projects, and all design source code is available on GitHub (https://github.com/Xilinx/BinomialModel). You need to import them to your local machine and then create projects using the following steps.

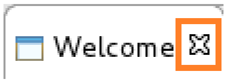
- Open **Terminal**
- Create a directory where you are going to place and run the reference design. Let's assume that this is a **/home/<*user*>/Binomial** folder.
- Go to the **/home/<*user*>/Binomial** folder
- Setup SDAccel environment by running the following command:

```
<installation_path>/settings64.sh (bash shell)
<installation_path>/settings64.csh (c shell)
```

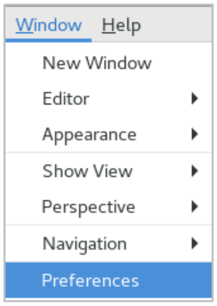
- Launch SDAccel by specifying `workspace` as a workspace:

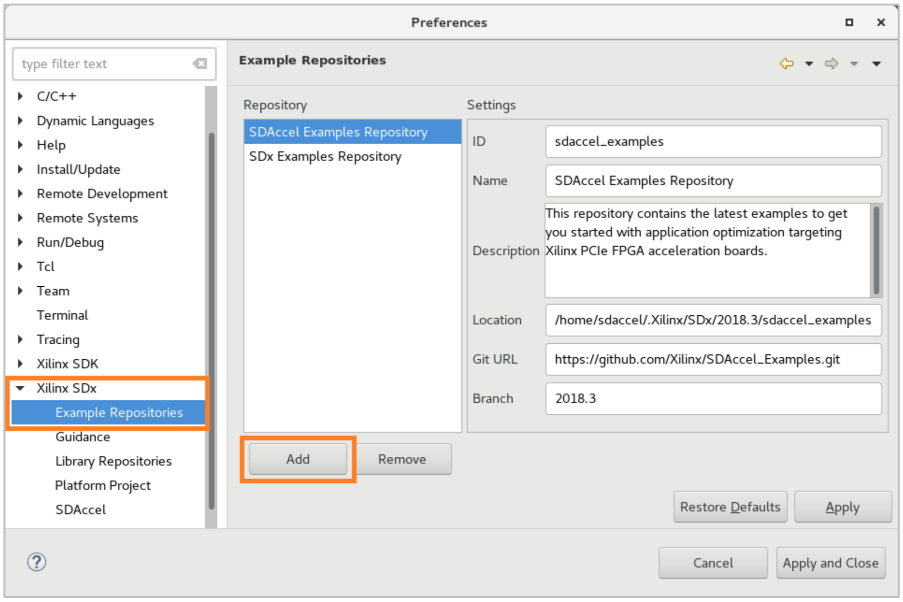
```
sdx -workspace workspace
```

- The Welcome window appears in the opened SDx environment. Press on the **x** button to Close this window before continue:



- In the opened SDAccel GUI, go to the **Window/Preferences** menu and then in the opened dialogue box navigate to the **Xilinx SDx/Example Repositories**:





ID: sdaccel_examples

Name: SDAccel Examples Repository

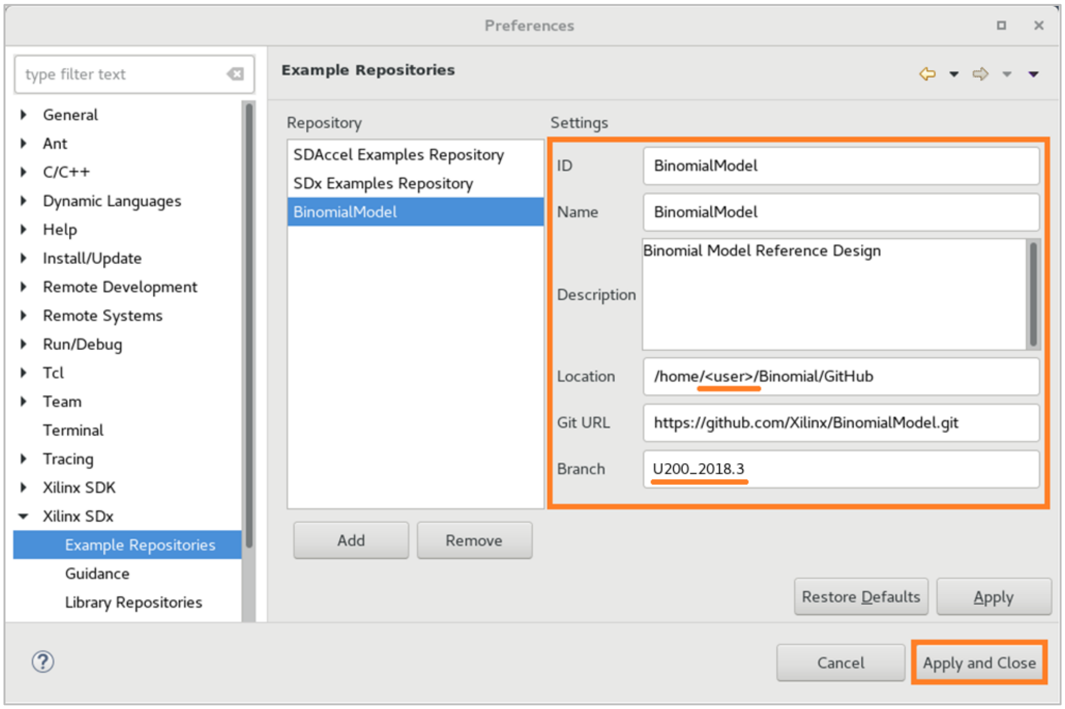
Description: This repository contains the latest examples to get you started with application optimization targeting Xilinx PCIe FPGA acceleration boards.

Location: /home/sdaccel/.Xilinx/SDx/2018.3/sdaccel_examples

Git URL: https://github.com/Xilinx/SDAccel_Examples.git

Branch: 2018.3

Press **Add** to add a new repository and then fill in a Settings part of the dialog box as shown below (note: replace `<user>` with your user name).

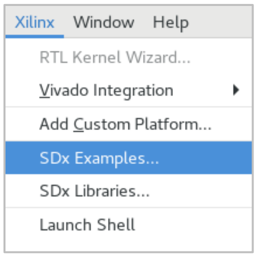


ID: BinomialModel
Name: BinomialModel
Description: Binomial Model Reference Design
Location: /home/<user>/Binomial/GitHub
Git URL: https://github.com/Xilinx/BinomialModel.git
Branch: U200_2018.3

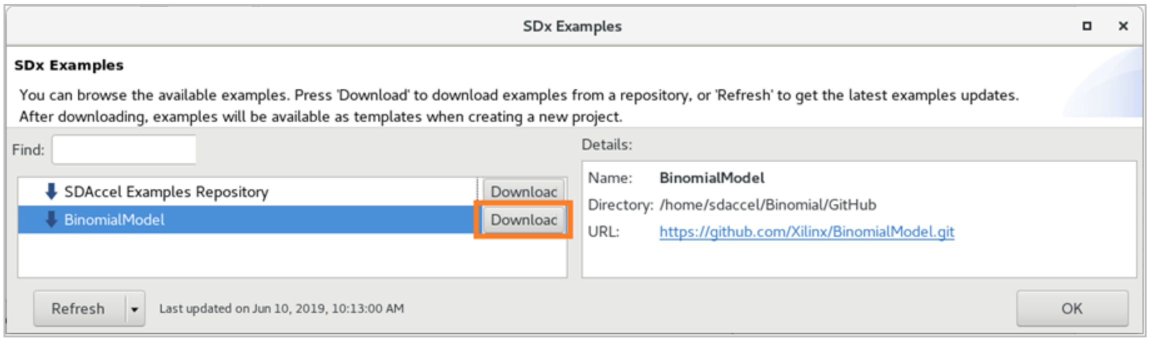
Imported project sources are placed in the **Binomial/GitHub** directory.

Then press **Apply and Close** to continue.

○ To export the reference design, go to the **Xilinx/SDx Examples** menu



where you should see a **BinomialModel** repository. Press **Download** to import the project sources:

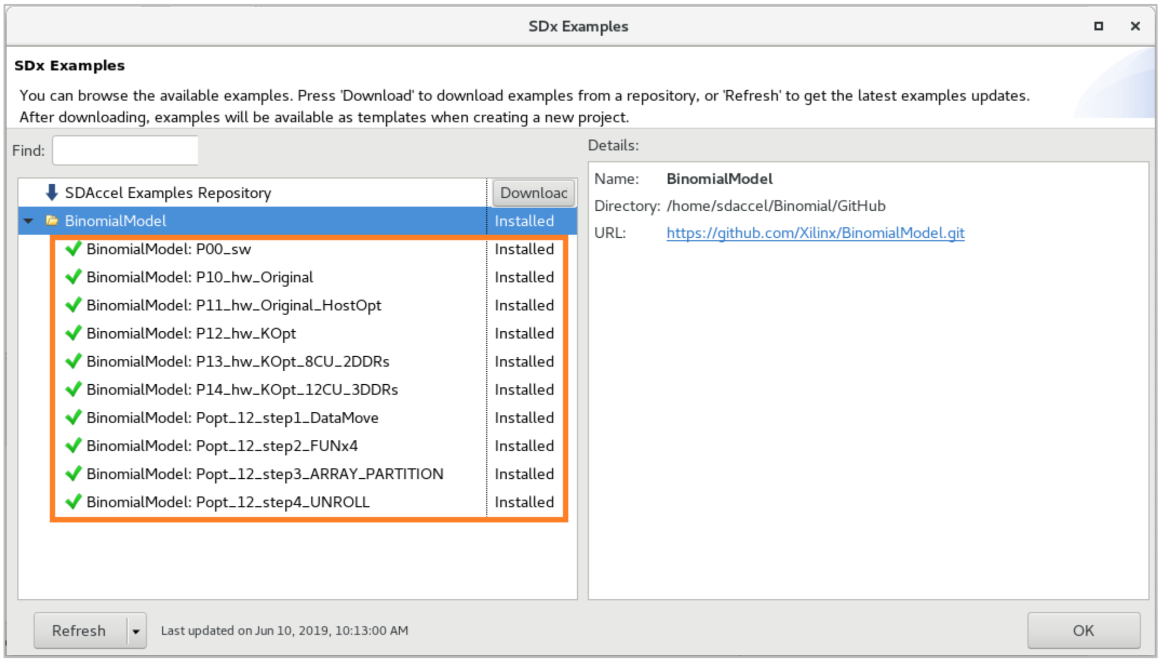


You can browse the available examples. Press 'Download' to download examples from a repository, or 'Refresh' to get the latest examples updates. After downloading, examples will be available as templates when creating a new project.

Name: BinomialModel
Directory: /home/sdaccel/Binomial/GitHub
URL: https://github.com/Xilinx/BinomialModel.git

Last updated on Jun 10, 2019, 10:13:00 AM

Once the import has completed, you should see the following list of project sources:



SDx Examples

You can browse the available examples. Press 'Download' to download examples from a repository, or 'Refresh' to get the latest examples updates.
After downloading, examples will be available as templates when creating a new project.

Find:

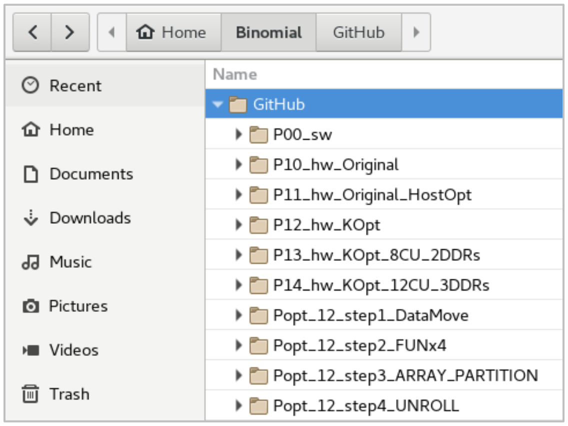
SDAccel Examples Repository	Downloac
BinomialModel	Installed
BinomialModel: P00_sw	Installed
BinomialModel: P10_hw_Original	Installed
BinomialModel: P11_hw_Original_HostOpt	Installed
BinomialModel: P12_hw_KOpt	Installed
BinomialModel: P13_hw_KOpt_8CU_2DDRs	Installed
BinomialModel: P14_hw_KOpt_12CU_3DDRs	Installed
BinomialModel: Popt_12_step1_DataMove	Installed
BinomialModel: Popt_12_step2_FUNx4	Installed
BinomialModel: Popt_12_step3_ARRAY_PARTITION	Installed
BinomialModel: Popt_12_step4_UNROLL	Installed

Details:

Name: **BinomialModel**
Directory: /home/sdaccel/Binomial/GitHub
URL: https://github.com/Xilinx/BinomialModel.git

Last updated on Jun 10, 2019, 10:13:00 AM

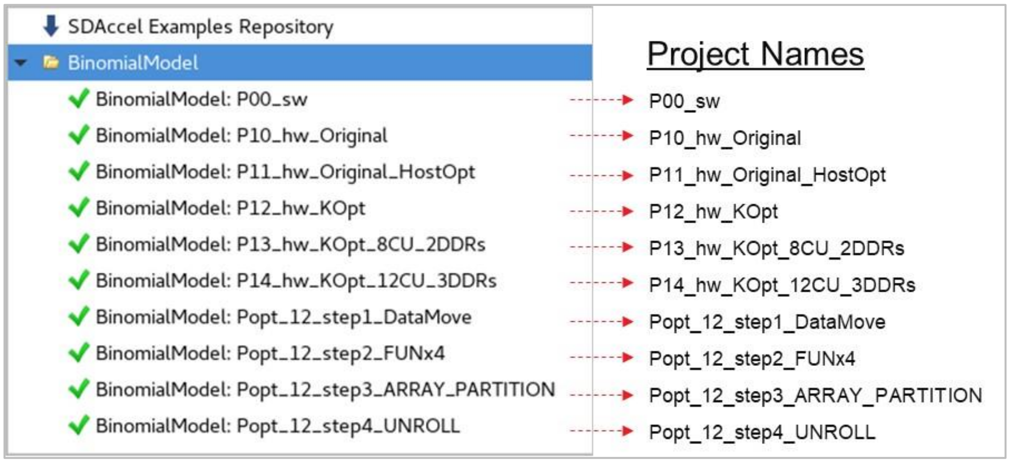
They are placed in the **/home/*<user>*/Binomial/GitHub** directory:



Name

- GitHub
  - P00_sw
  - P10_hw_Original
  - P11_hw_Original_HostOpt
  - P12_hw_KOpt
  - P13_hw_KOpt_8CU_2DDRs
  - P14_hw_KOpt_12CU_3DDRs
  - Popt_12_step1_DataMove
  - Popt_12_step2_FUNx4
  - Popt_12_step3_ARRAY_PARTITION
  - Popt_12_step4_UNROLL

- Finally, you need to create projects using the following project names:

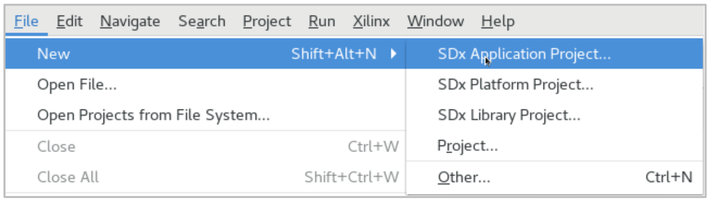


SDAccel Examples Repository
BinomialModel
✓ BinomialModel: P00_sw
✓ BinomialModel: P10_hw_Original
✓ BinomialModel: P11_hw_Original_HostOpt
✓ BinomialModel: P12_hw_KOpt
✓ BinomialModel: P13_hw_KOpt_8CU_2DDRs
✓ BinomialModel: P14_hw_KOpt_12CU_3DDRs
✓ BinomialModel: Popt_12_step1_DataMove
✓ BinomialModel: Popt_12_step2_FUNx4
✓ BinomialModel: Popt_12_step3_ARRAY_PARTITION
✓ BinomialModel: Popt_12_step4_UNROLL

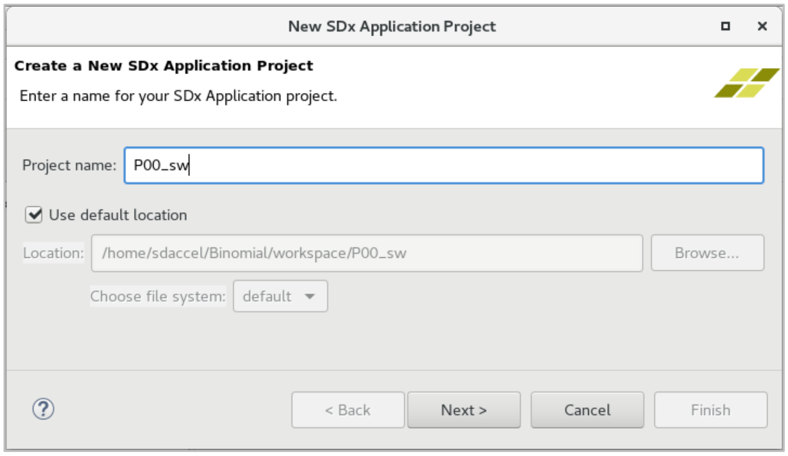
Project Names
P00_sw
P10_hw_Original
P11_hw_Original_HostOpt
P12_hw_KOpt
P13_hw_KOpt_8CU_2DDRs
P14_hw_KOpt_12CU_3DDRs
Popt_12_step1_DataMove
Popt_12_step2_FUNx4
Popt_12_step3_ARRAY_PARTITION
Popt_12_step4_UNROLL

using the following procedure (shown for a **P00_sw** project):

- Select the "**SDx Application Project**" from the **File / New Menu**:



- In the "**Create a New SDx Project**" dialog box specify the following project name: **P00_sw**



New SDx Application Project

Create a New SDx Application Project

Enter a name for your SDx Application project.

Project name: P00_sw

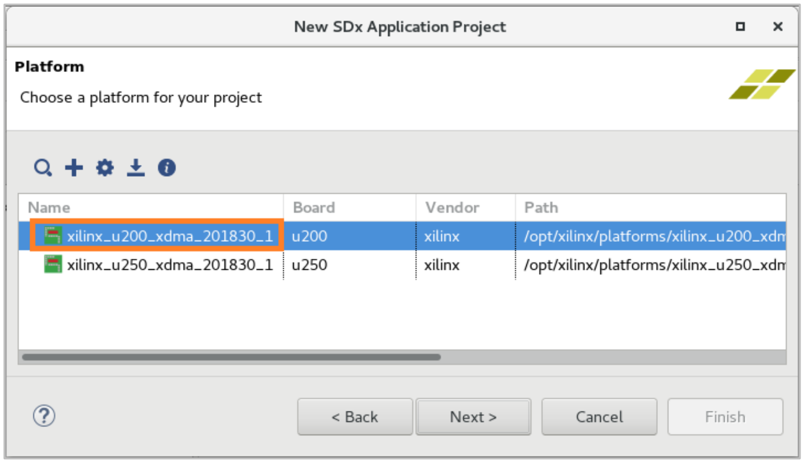
☑ Use default location

Location: /home/sdaccel/Binomial/workspace/P00_sw

Choose file system: default

Then press **Next**

- In the "Choose Hardware Platform" dialog box select the "**xilinx_u200_xdma_201830_1**" platform:



New SDx Application Project

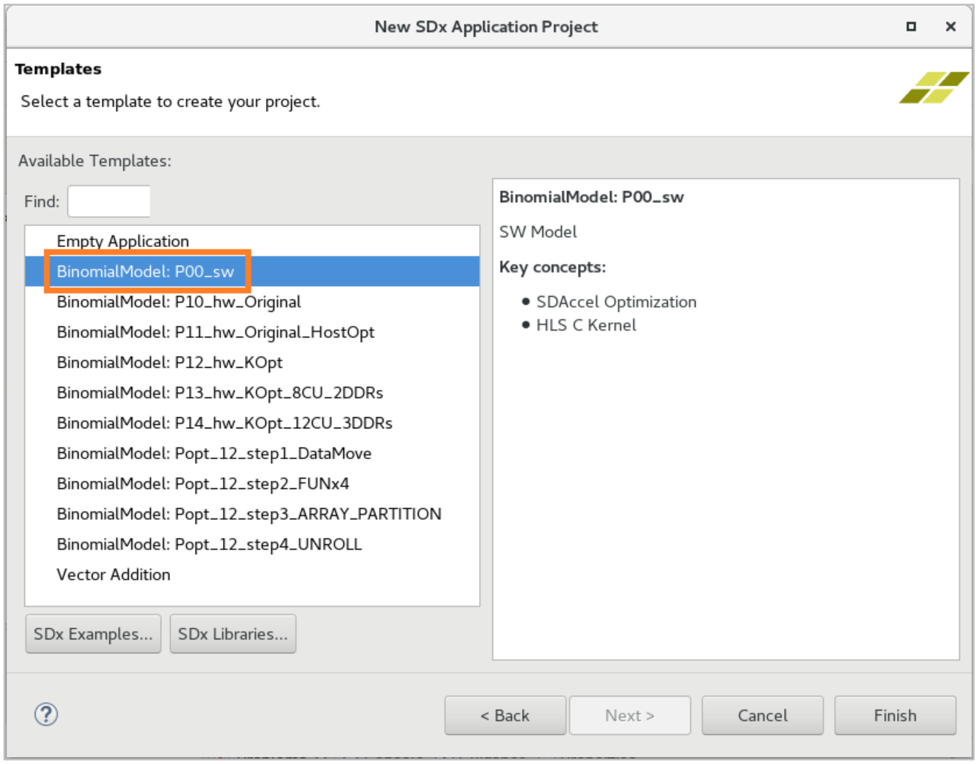
Platform

Choose a platform for your project

Name	Board	Vendor	Path
xilinx_u200_xdma_201830_1	u200	xilinx	/opt/xilinx/platforms/xilinx_u200_xdm
xilinx_u250_xdma_201830_1	u250	xilinx	/opt/xilinx/platforms/xilinx_u250_xdm

Then press **Next**

- In the "Templates" dialog box select **BinomialModel: P00_sw**



New SDx Application Project

Templates

Select a template to create your project.

Available Templates:

Find:

Empty Application
BinomialModel: P00_sw
BinomialModel: P10_hw_Original
BinomialModel: P11_hw_Original_HostOpt
BinomialModel: P12_hw_KOpt
BinomialModel: P13_hw_KOpt_8CU_2DDRs
BinomialModel: P14_hw_KOpt_12CU_3DDRs
BinomialModel: Popt_12_step1_DataMove
BinomialModel: Popt_12_step2_FUNx4
BinomialModel: Popt_12_step3_ARRAY_PARTITION
BinomialModel: Popt_12_step4_UNROLL
Vector Addition

BinomialModel: P00_sw

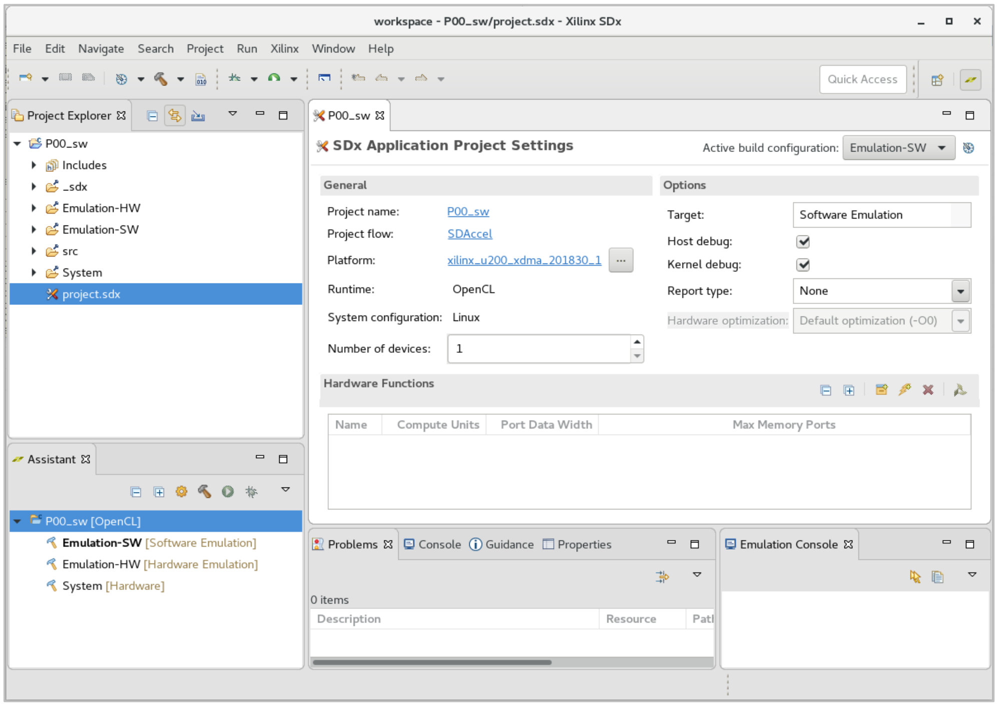
SW Model

Key concepts:

- SDAccel Optimization
- HLS C Kernel

Then press **Finish** to finalize the project creation.

You will obtain the following environment:


workspace - P00_sw/project.sdx - Xilinx SDx
File Edit Navigate Search Project Run Xilinx Window Help
Quick Access
Project Explorer
P00_sw
Assistant
Problems
Console
Guidance
Properties
Emulation Console

P00_sw
- Includes
- _sdx
- Emulation-HW
- Emulation-SW
- src
- System
- project.sdx

SDx Application Project Settings

Active build configuration: Emulation-SW

General

Project name: P00_sw

Project flow: SDAccel

Platform: xilinx_u200_xdma_201830_1

Runtime: OpenCL

System configuration: Linux

Number of devices: 1

Options

Target: Software Emulation

Host debug: ☑

Kernel debug: ☑

Report type: None

Hardware optimization: Default optimization (-O0)

Hardware Functions

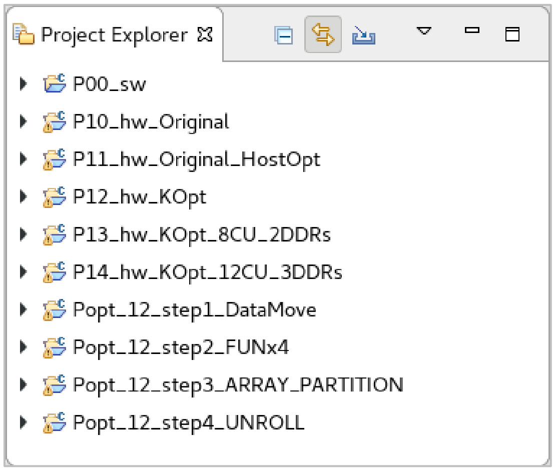
Name	Compute Units	Port Data Width	Max Memory Ports

P00_sw [OpenCL]
- Emulation-SW [Software Emulation]
- Emulation-HW [Hardware Emulation]
- System [Hardware]

0 items

Description	Resource	Pat

○ Repeat the same procedure for the remaining projects. After that you should see all the projects in the SDAccel GUI:



- P00_sw
- P10_hw_Original
- P11_hw_Original_HostOpt
- P12_hw_KOpt
- P13_hw_KOpt_8CU_2DDRs
- P14_hw_KOpt_12CU_3DDRs
- Popt_12_step1_DataMove
- Popt_12_step2_FUNx4
- Popt_12_step3_ARRAY_PARTITION
- Popt_12_step4_UNROLL

Please Read: Important Legal Notice Appendix B

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright