Применение C++ для разработки проектов ПЛИС

Дмитрий Смехов

ЛяЛямбда 2023

Дополнительные материалы

https://github.com/Xilinx/Vitis-Tutorials

https://github.com/Xilinx/Vitis-HLS-Introductory-Examples

https://github.com/dsmv/Vitis_OpenCL_Tutorials

Quokka:

https://github.com/EvgenyMuryshkin/QuokkaEvaluation

https://www.youtube.com/watch?v=_MrGRMY-6jE

Проблемы

- Большое время моделирование полного проекта
- Работа с простыми сигналами, выполнение однотипных операций

Повышение уровня разработки

Разработка программы

- C++ -> Ассемблер -> Машинный код

Разработка RTL

- C++ -> SystemVerilog -> Netlist

Vitis, VitisHLS, Vitis AI

VitisHLS: два типа компонентов:

- Одиночное срабатывание
- Обработка потока

Vitis — система разработки HOST/KERNEL

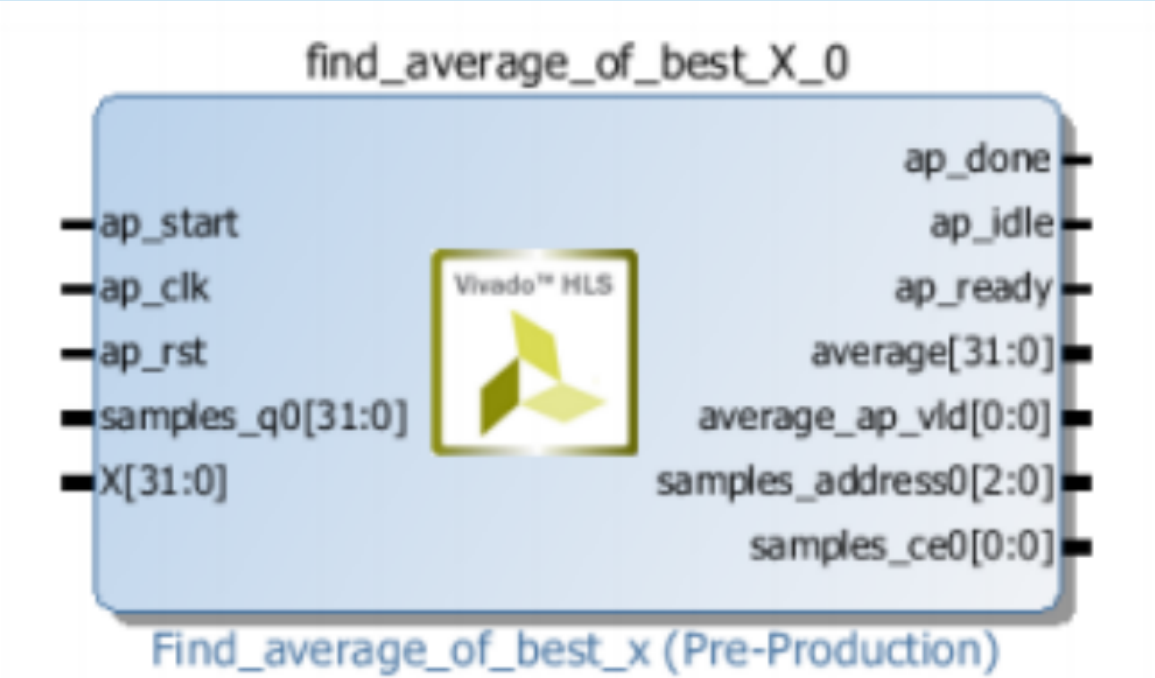
- на основе библиотеки XRT
- на основе OpenCL

Vitis AI — система для реализации нейросетей на ПЛИС XILINX

- кернел DPU

- портирования сетей из систем TensorFlow и других

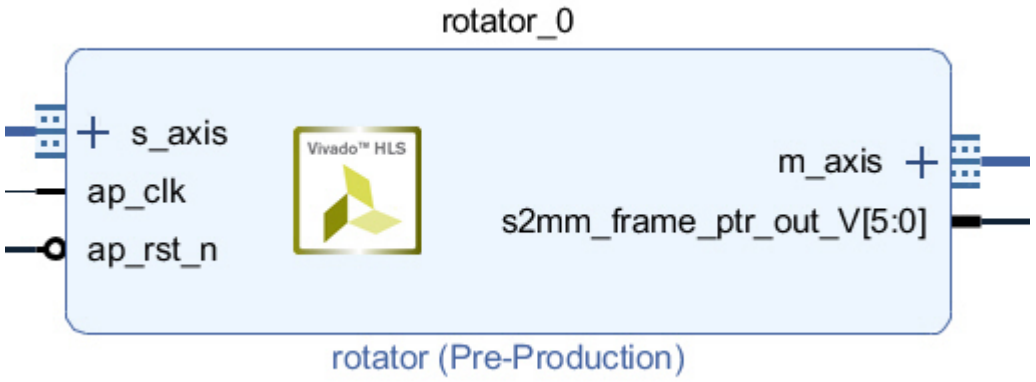
Одиночное срабатывание



ap_start — запуск процесса вычисления

ap_done — завершение процесса вычисления

Обработка потока



- На входе — поток данных
- На выходе — поток данных

```
void getinstream( hls::stream<trans_pkt >& in_stream,
                  hls::stream<data >& out_stream,
                  hls::stream<int>& out_counts)
{
    int count = 0;
    trans_pkt in_val;
    do {
#pragma HLS PIPELINE
    in_val = in_stream.read();
    data out_val = {in_val.data, in_val.last};
    out_stream.write(out_val);      count++;
    if (count >= MAX_BURST_LENGTH || in_val.last) {
      out_counts.write(count);
      count = 0;
    }
  } while(!in_val.last);
 }
```

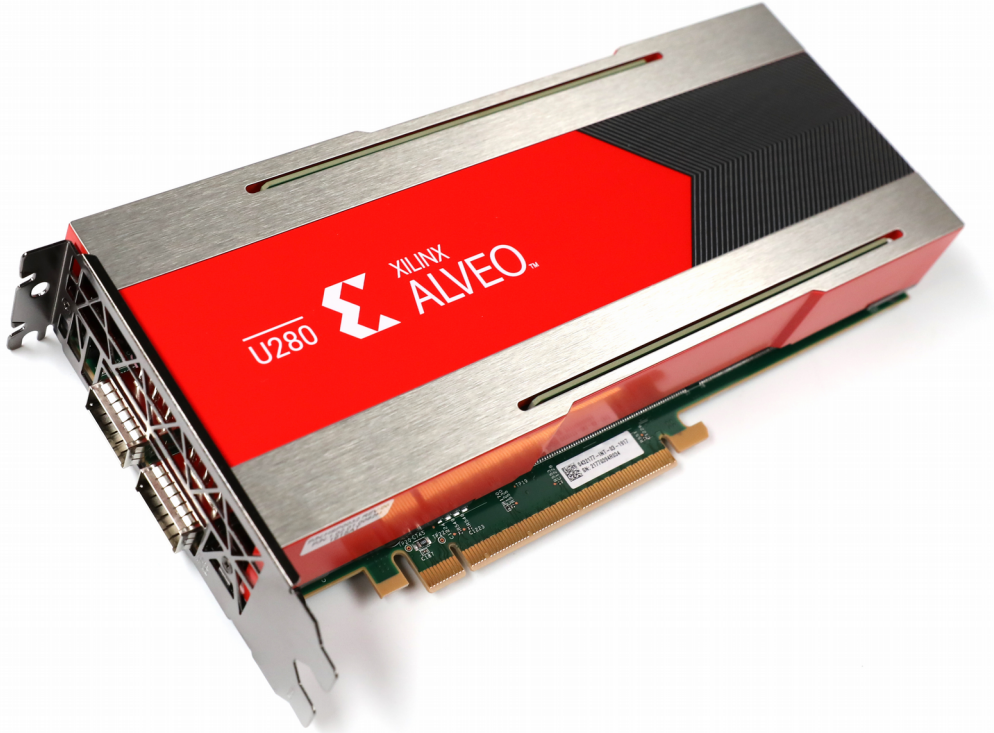
C++ для RTL

- Нельзя взять программу C++ и запустить её на ПЛИС
- C++ это ещё один инструмент для разработки RTL на ПЛИС

Особенности:

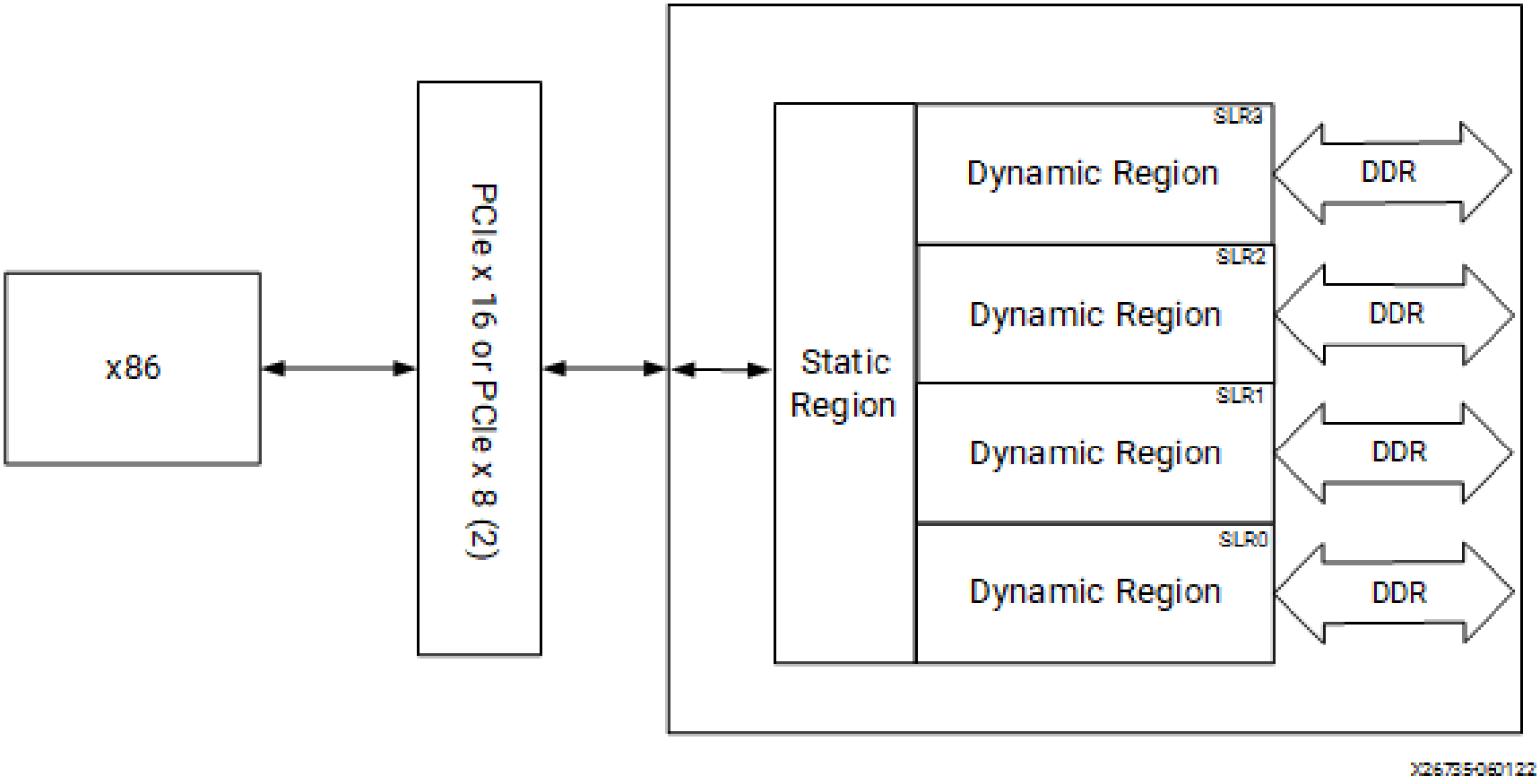
- Нет понятия тактовой частоты — это принципиально важно!
- Проект RTL на C++ можно скомпилировать и запустить как программу. Получается выигрыш во времени моделирования на несколько порядков

ALVEO U280

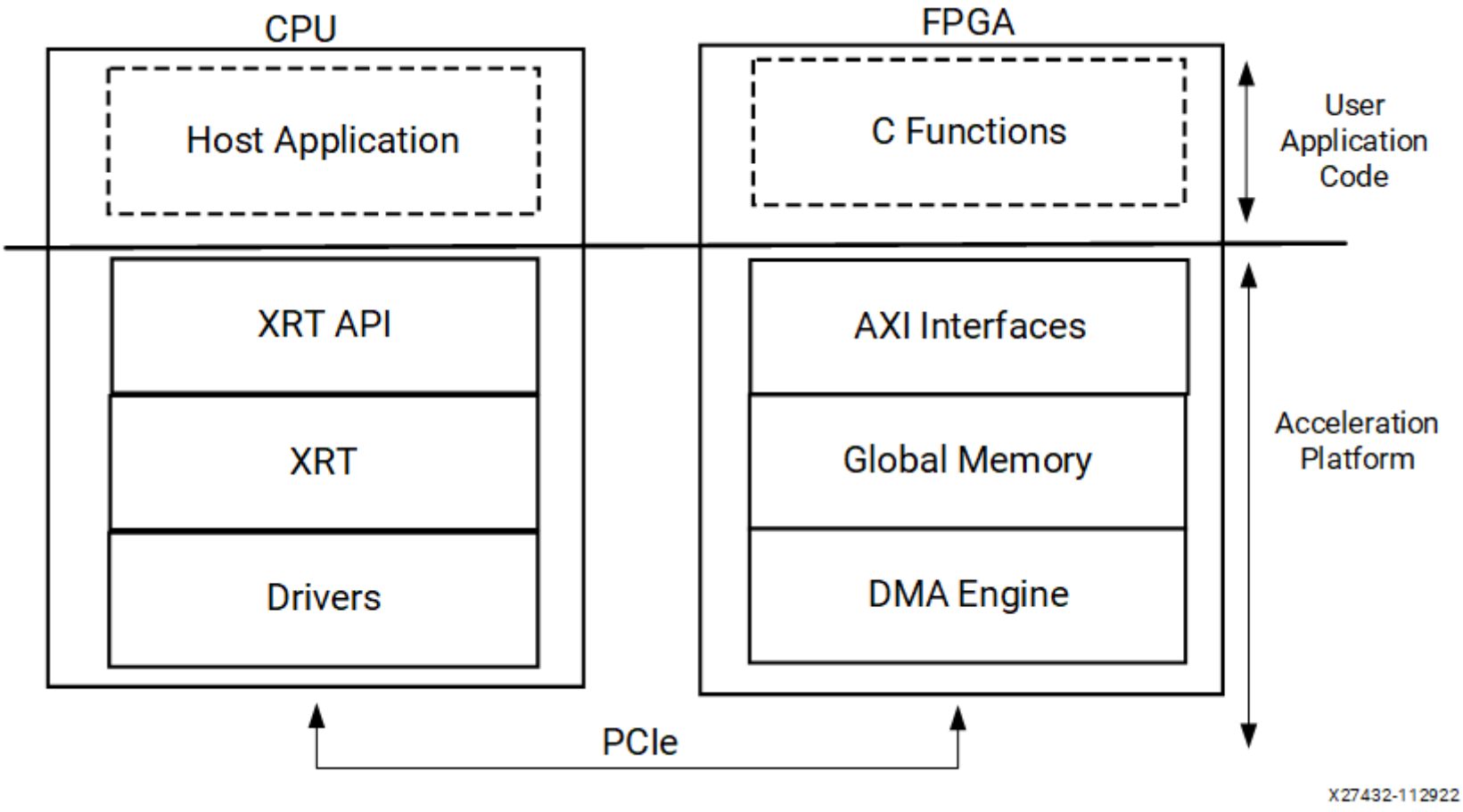


https://www.xilinx.com/products/boards-and-kits/alveo/u280.html

Структура проекта



Модель применения

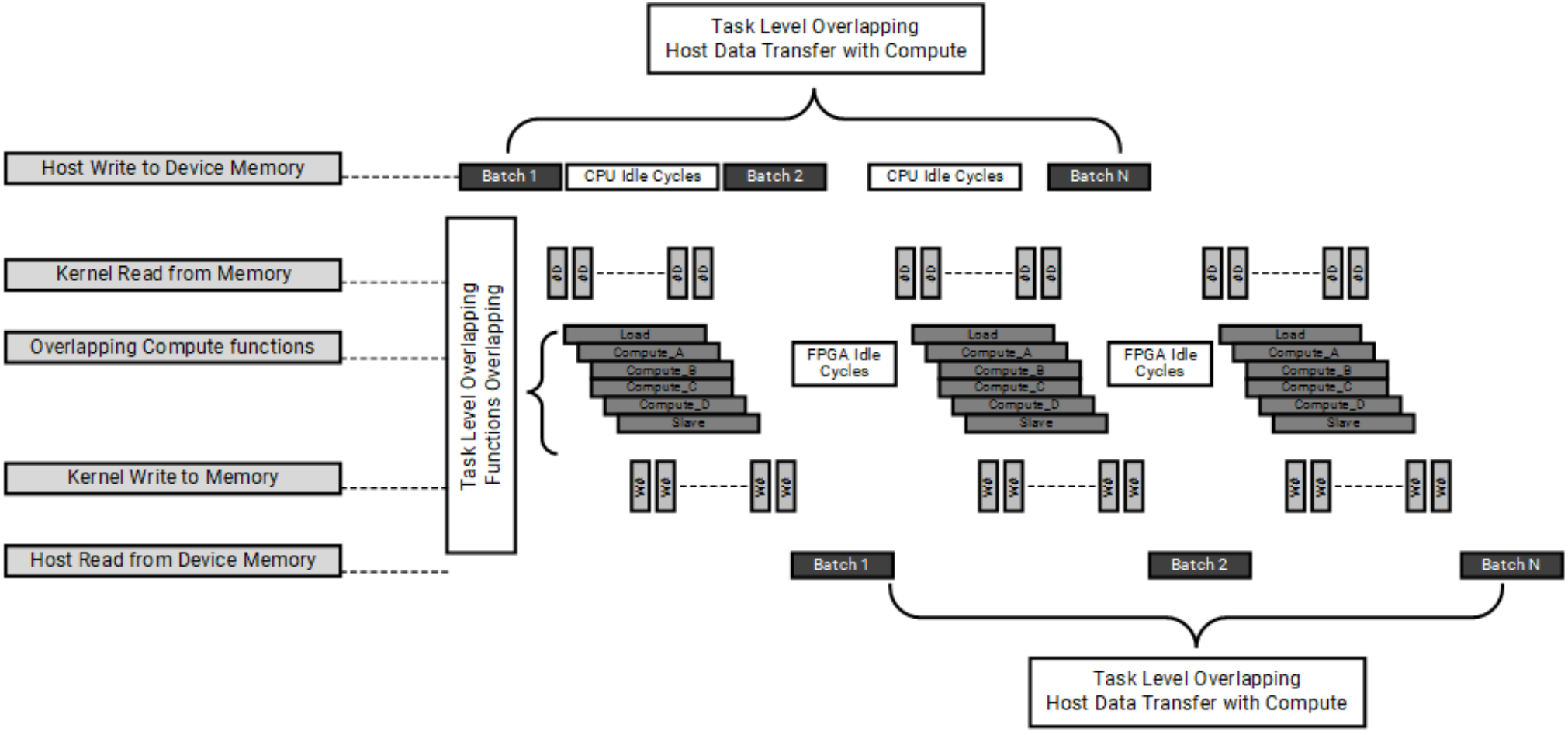


Вызов функции ускорителя

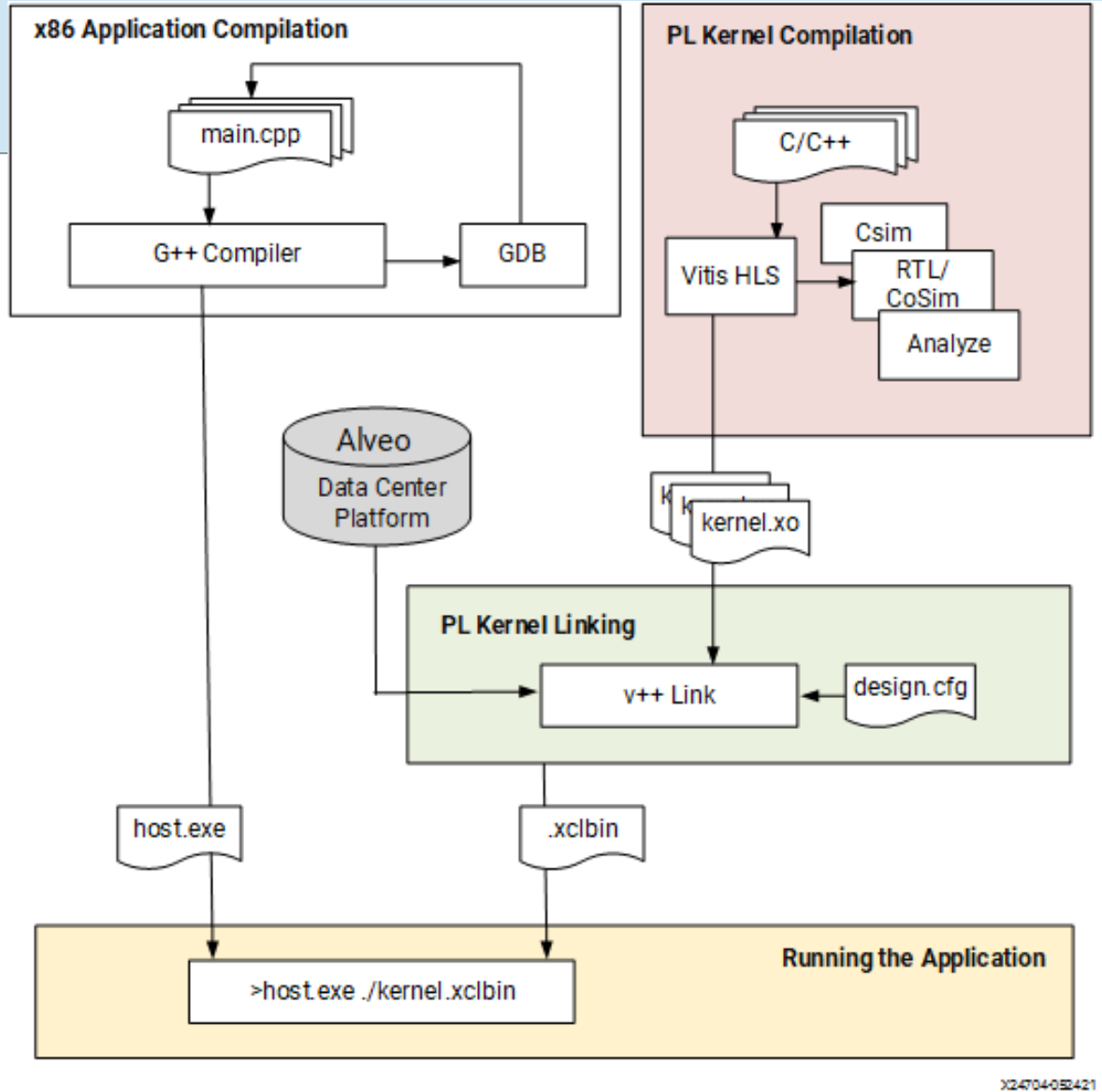
```
// Synchronize buffer content with device side
    bufIn.sync(XCL_BO_SYNC_BO_TO_DEVICE);

    std::cout << "Execution of the kernel\n";
    auto run = krnl(bufIn,bufOut,totalNumWords/16);
    run.wait();

// Get the output;
    std::cout << "Get the output data from the device" << std::endl;
    bufOut.sync(XCL_BO_SYNC_BO_FROM_DEVICE);
```

https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration/A-Sample-Application

Application timeline



X27500-120522


x86 Application Compilation
main.cpp
G++ Compiler
GDB
PL Kernel Compilation
C/C++
Csim
Vitis HLS
RTL/
CoSim
Analyze
Alveo
Data Center
Platform
kernel.xo
PL Kernel Linking
v++ Link
design.cfg
host.exe
.xclbin
Running the Application
>host.exe ./kernel.xclbin
X24704052421

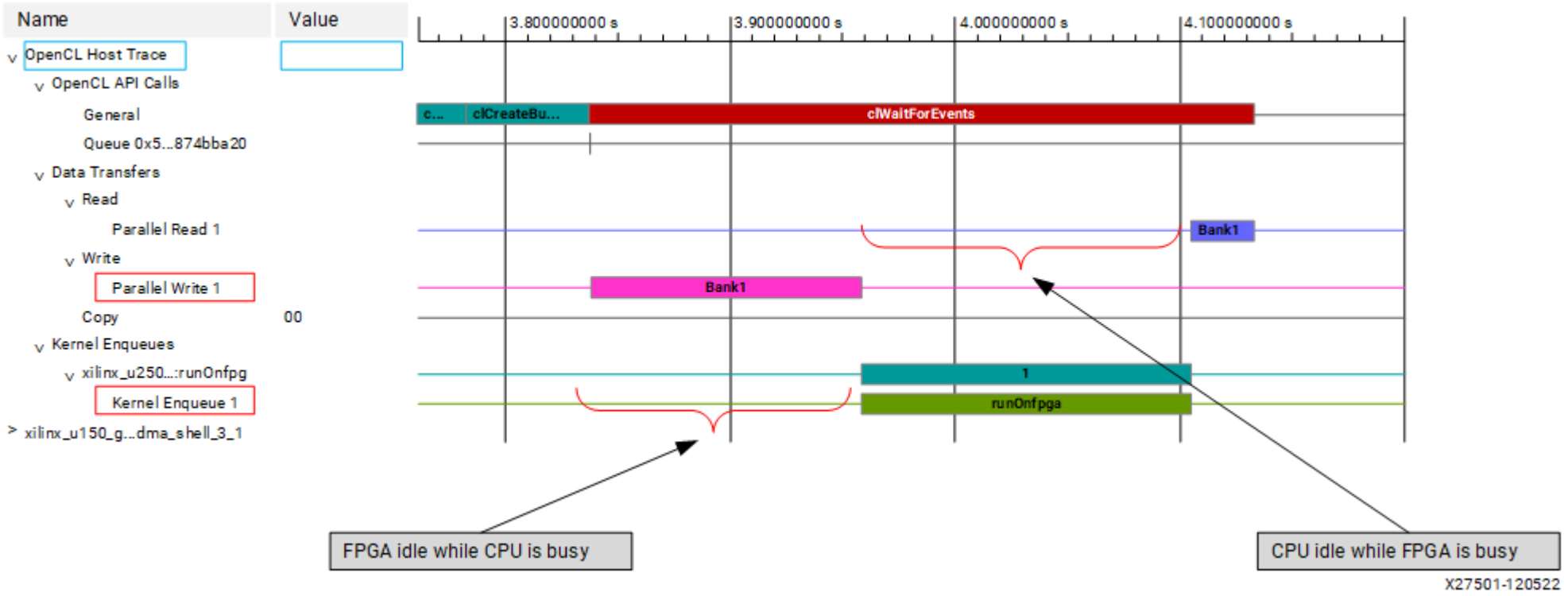
Три способа формирования kernel

- VitisHLS - Разработка на C++, 300 MHz
- OpenCL v2.0 — Разработка на C, 300 MHz
- RTL — разработка на SystemVerilog/VHDL, 300 Mhz, 500 MHz

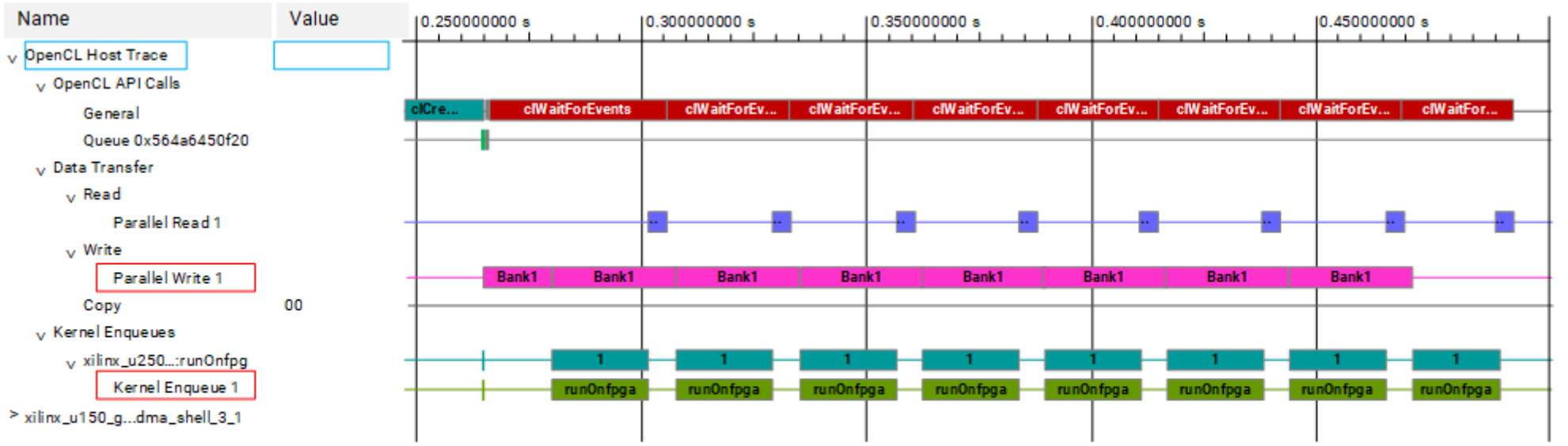
Код RTL может включать в себя отдельные компоненты на C++

https://github.com/dsmv/Vitis_OpenCL_Tutorials

Пример временной диаграммы



Улучшенный пример



Higher utilization of FPGA and CPU

X27502-120522

Quokka — разработка на C#

```
{
    [BoardConfig(Name = "Quokka")]
    public static class T00BlinkyController
    {
        public static async Task Aggregator(OutputSignal<bool> LED1)
        {
            bool internalAlive = false;
            Config.Link(internalAlive, LED1);

            Sequential aliveHandler = () =>
            {
                internalAlive = !internalAlive;
            };

            Config.OnTimer(TimeSpan.FromSeconds(1), aliveHandler);
        }
    }
}
```