An Introduction to Vitis HLS

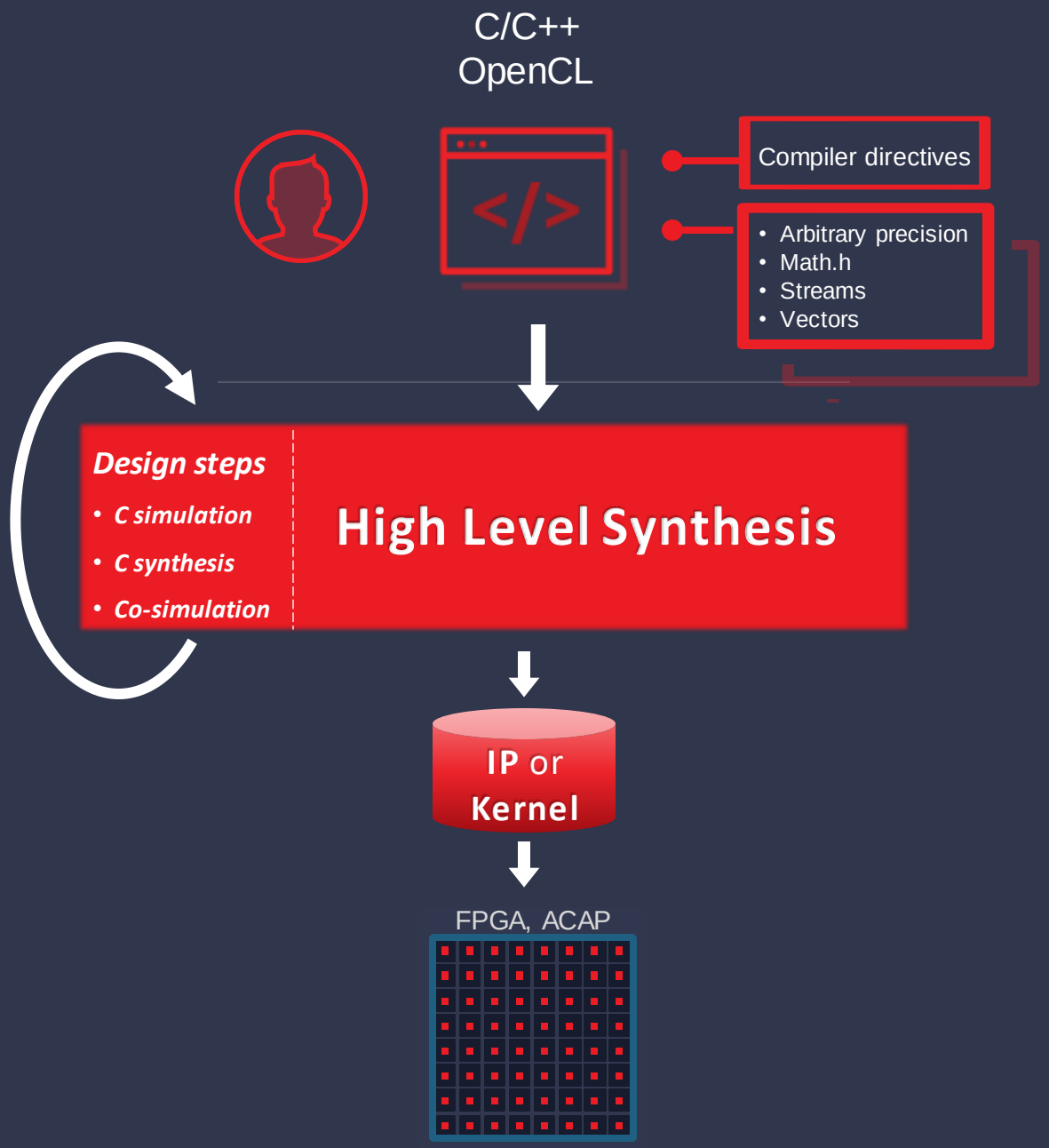
XILINX.

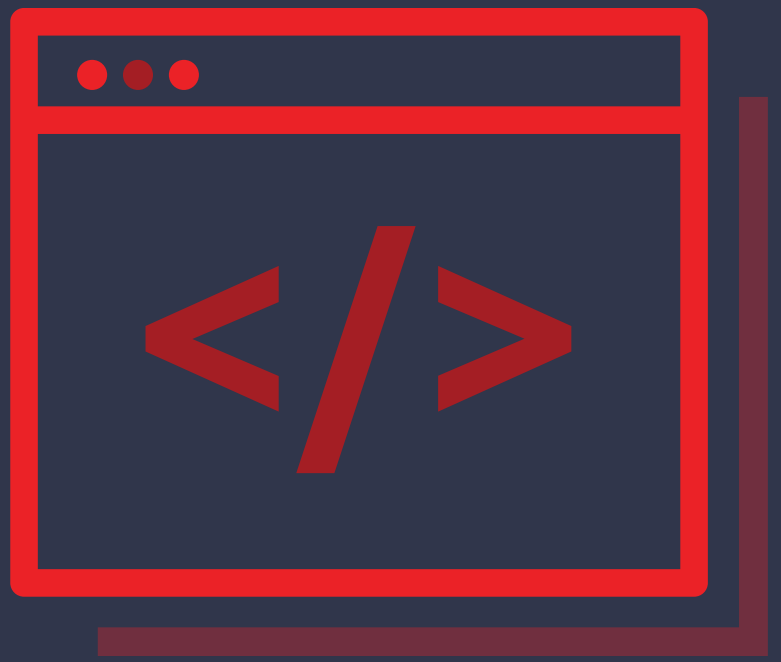
Agenda

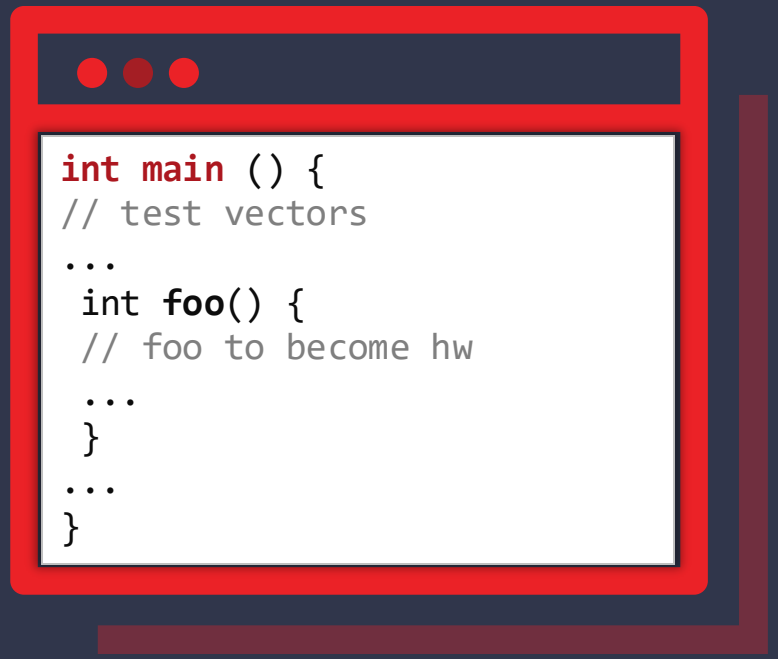
- High-Level Synthesis (HLS) Flows
- Technical Overview + Demo Examples
- Resources

Vitis HLS

- **Abstracted C-based Design Entry**
- **Higher Productivity !**
  - Concise code
  - Fast C simulation
  - Automated Simulation of Generated RTL
  - Optimized Libraries






int main () {
// test vectors
...
int foo() {
// foo to become hw
...
}
...
}

Applications for High-Level Synthesis

Surveillance, AI
- Classification
- Recognition

Robotics
- Drones
- Micro-controller, AI

Aerospace and Defense
- Radar, Sonar
- Signals Intelligence

Communications
- LTE MIMO receiver
- Advanced wireless antenna positioning

Industrial, Scientific, Medical
- Ultrasound systems
- Motor controllers

Audio, Video, Broadcast
- 3D cameras
- Video transport

Automotive
- Infotainment
- Driver assistance / AI

Consumer
- 3D television
- eReaders

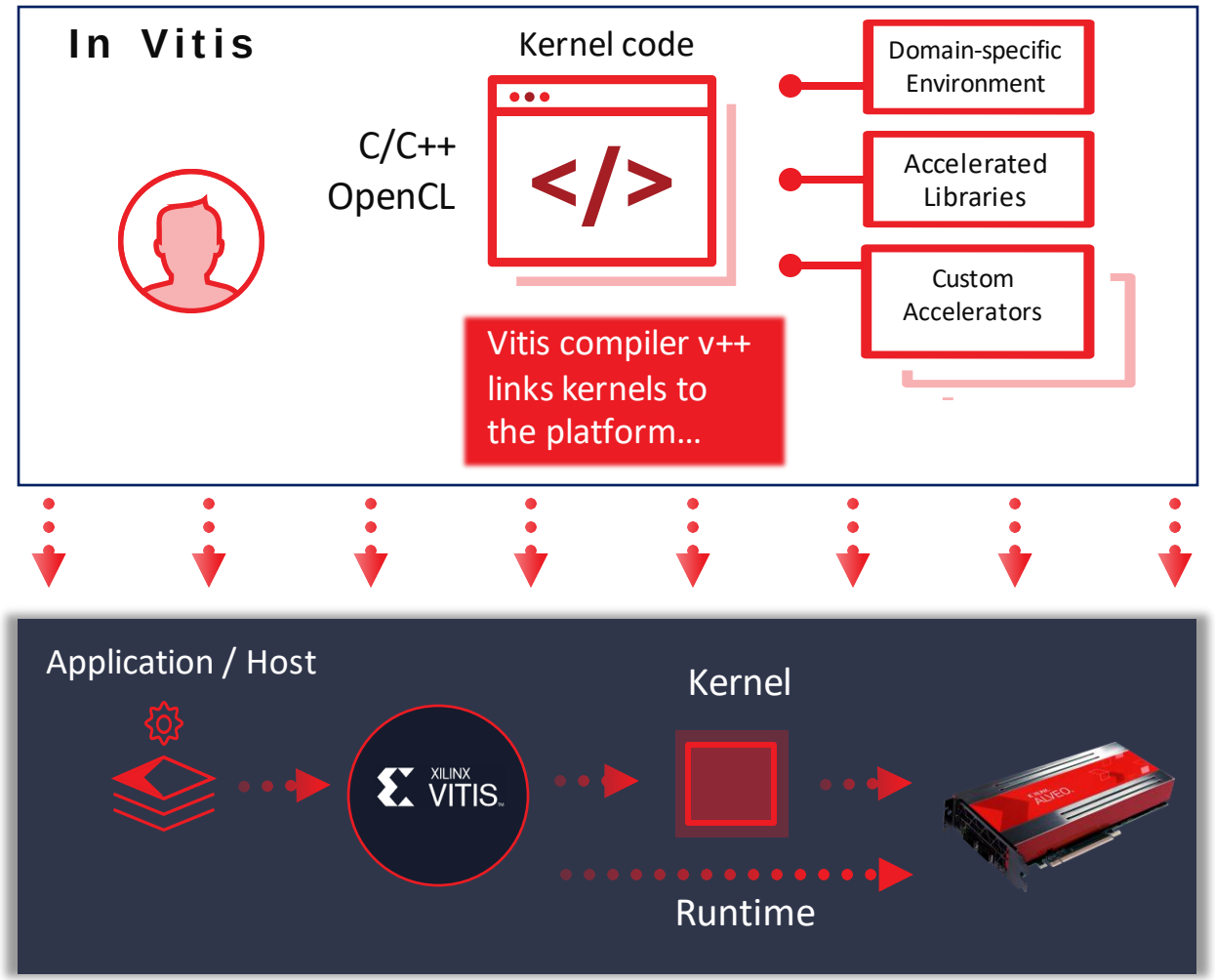
Test & Measurement
- Communications instruments
- Semiconductor ATE

Computing & Storage
- High performance computing
- Database acceleration

HLS in Vitis Flow

HLS compiles C-based Kernels

- ✓ v++ performs all the compiles and links
  - ✓ HLS is automatically invoked
- ✓ No necessary direct interaction with HLS
- ✓ HLS reports imported in Vitis Analyzer
- ✓ Full application can be C-based

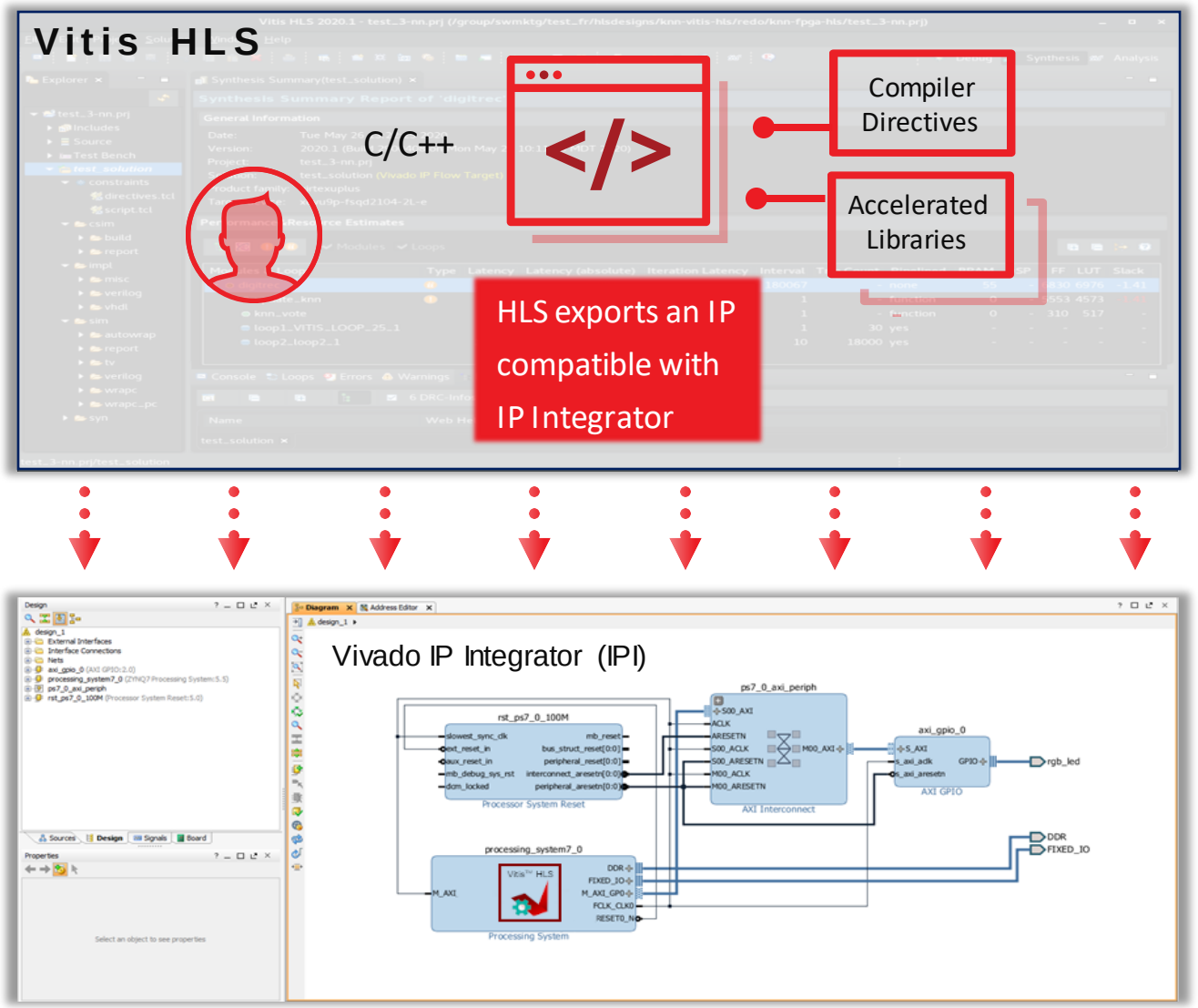


XILINX

HLS in Vivado Flow

HLS exports RTL IP…

- ✓ User runs HLS directly
- ✓ Typically block assembly done in IPI
- ✓ Design entry is C/C++
- ✓ Can invoke Vivado waveform viewer



XILINX.

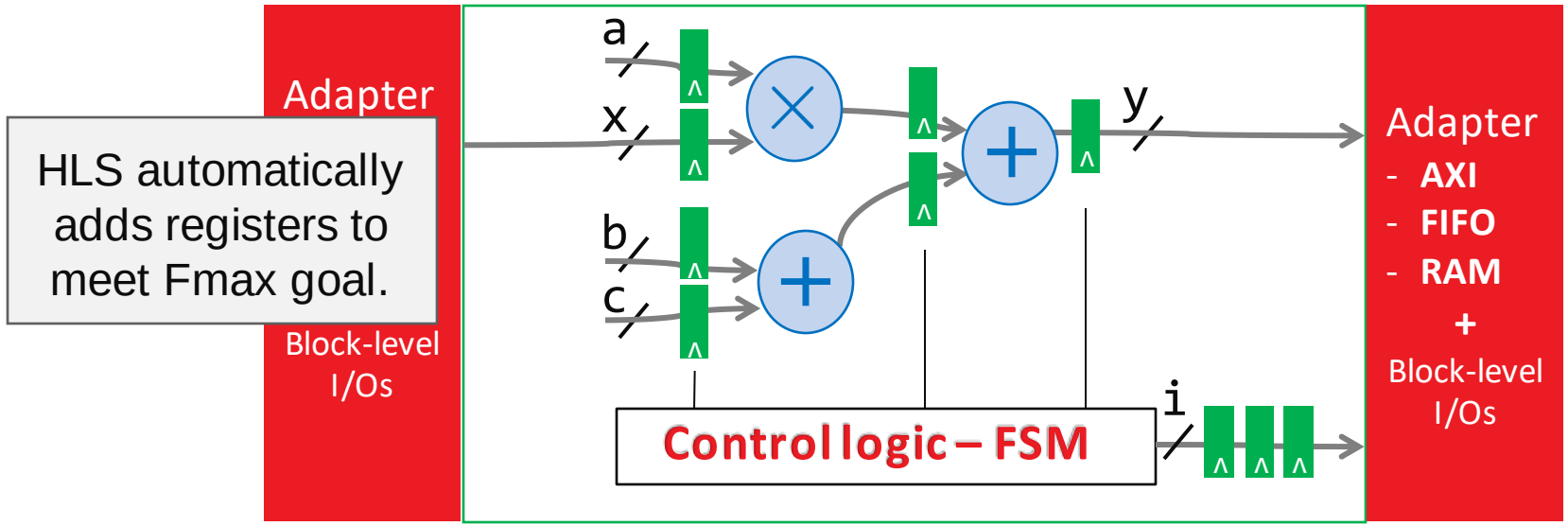
Agenda

- High-Level Synthesis (HLS) Flows
- Technical Overview + Demo Examples
- Resources

Automatic Interface and Control Logic

> **Simple C code quickly become a kernel or an IP…**

```
void f(int in[4], int out[4]) {
  int a,b,c,x,y;
  for(int i = 0; i < 4; i++) {
    x = in[i]; y = a*x + b + c; out[i] = y;
  }
```

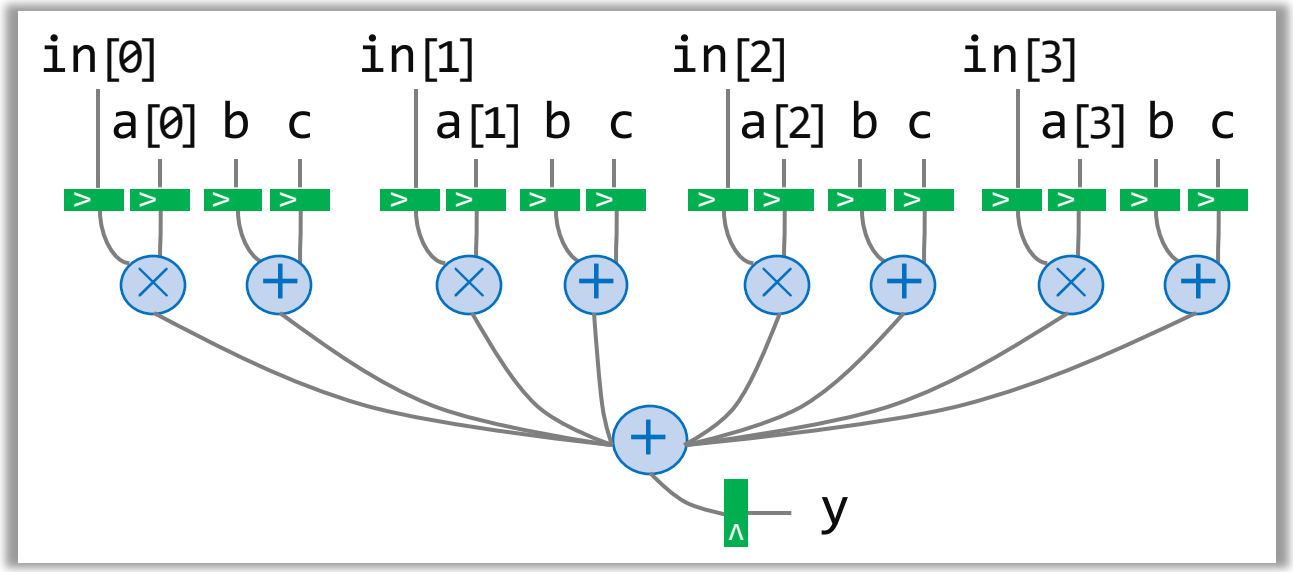


HLS automatically adds registers to meet Fmax goal.

Design Space Exploration via Pragmas

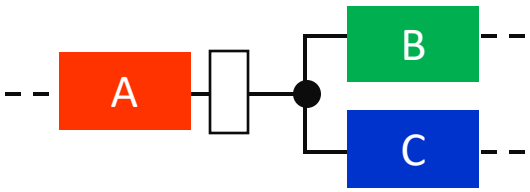
- **Pragmas change the circuit topology…**

```
void f(int in[4], int &y, int a[4], int b, int c) {
#pragma HLS ARRAY_PARTITION variable=in dim=1 complete
#pragma HLS ARRAY_PARTITION variable=a  dim=1 complete
#pragma HLS PIPELINE
   for(int i = 0; i < 4; i++)
           y += a[i] * in[i] + b + c;
}
```

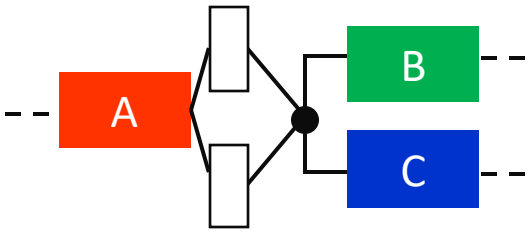


Task Parallelism with HLS

Default versus ping-pong buffers



By default…
As **A** produces its buffer, **B** and **C** wait...
Then **B** and **C** consume and **A** waits…



With **ping-pong** buffers…
A produces in one buffer, **B** and **C** read the other.
Next invocation, buffers switch roles: tasks can work continuously

The dataflow pragma in HLS automates memory expansion to enable task parallelism

Demo #1 – Task Parallelism

Vectorized Data Types…

```
// Vectorization means that the compiler detects that
// independent instructions can be executed as SIMD instructions.
// So, something like this...
for(i=0; i<N; i++){
  a[i] = a[i] + b[i];
}
// ... becomes "vectorized" as... (using vector notation)
for (i=0; i<(N-N%VF); i+=VF){
  a[i:i+VF] = a[i:i+VF] + b[i:i+VF];
}
// 1 operation that can be done on VF elements of the array
// at the same time and does this N/VF times instead of doing
// the single operation N times...
```

Vectorized Data Types in Vitis HLS

- **Vitis HLS supports the C++14 `vector_size` attribute**
  - Simply using C++…

```
// vector_size specifies size in bytes
typedef float float16 __attribute__(vector_size(64));
```

Custom vector type float16 based on C++ attribute

- **… and also supports arbitrary precision types via `hls_vector.h`**
  - Examples

```
#include "hls_vector.h"
using float16 = hls::vector<float, 16>;
```

Same as above using hls::vector

```
#include "hls_vector.h"
using quad = hls::vector<ap_int<18>, 4>;
```

Vector of four 18-bit signed variables

Vectorized Data Types – Operations

Initialization

```
hls::vector<int, 4> x;                  // uninitialized
hls::vector<int, 4> y = 10;             // scalar initialized
hls::vector<int, 4> z = {0, 1, 2, 3}; // initializer list (must have 4 elements)
```

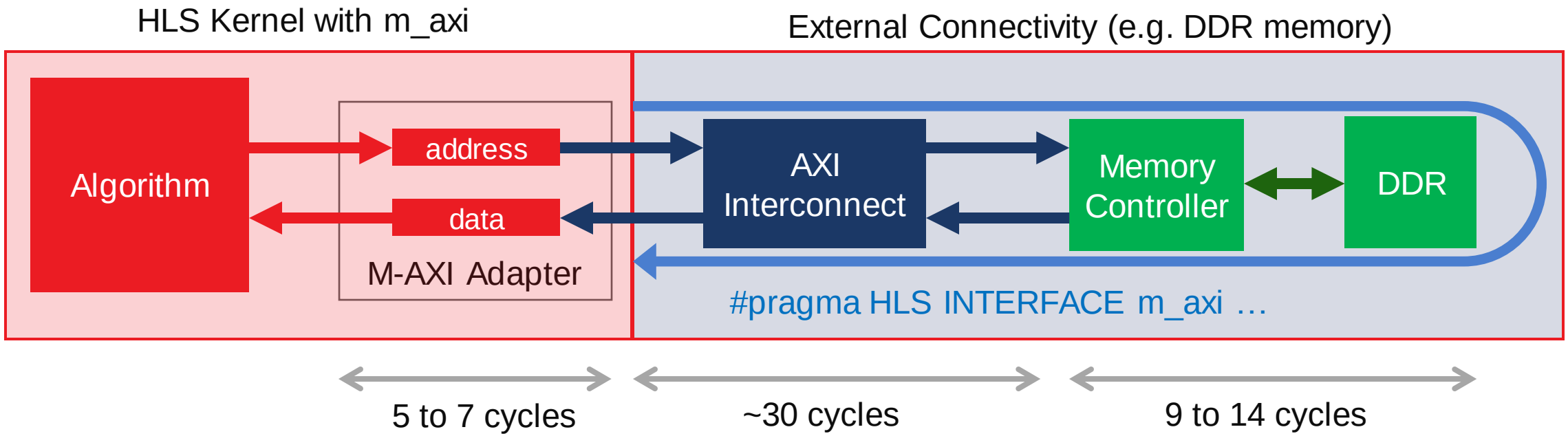
Access

```
myvec[i] = ...; // reference to an element
... = myvec[i]; // value of an element
```

Recommendations

- Use hls::vector<T, N> with N as a power of 2 for a better alignment that guarantees smaller initiation interval (II)
- Use the __attribute__((no_ctor)) for better II when using dataflow

Optimizing Interfaces



- **Efficient Pipeline**
  - Adapt the latency parameter of the interface for efficiency
- **Loop Bursts**
  - Segmented into smaller bursts by the adapter (and that's okay!)
  - Adapter will pipeline for you (independent state machine!)

Demo #2 – Vector types and AXI Interfaces

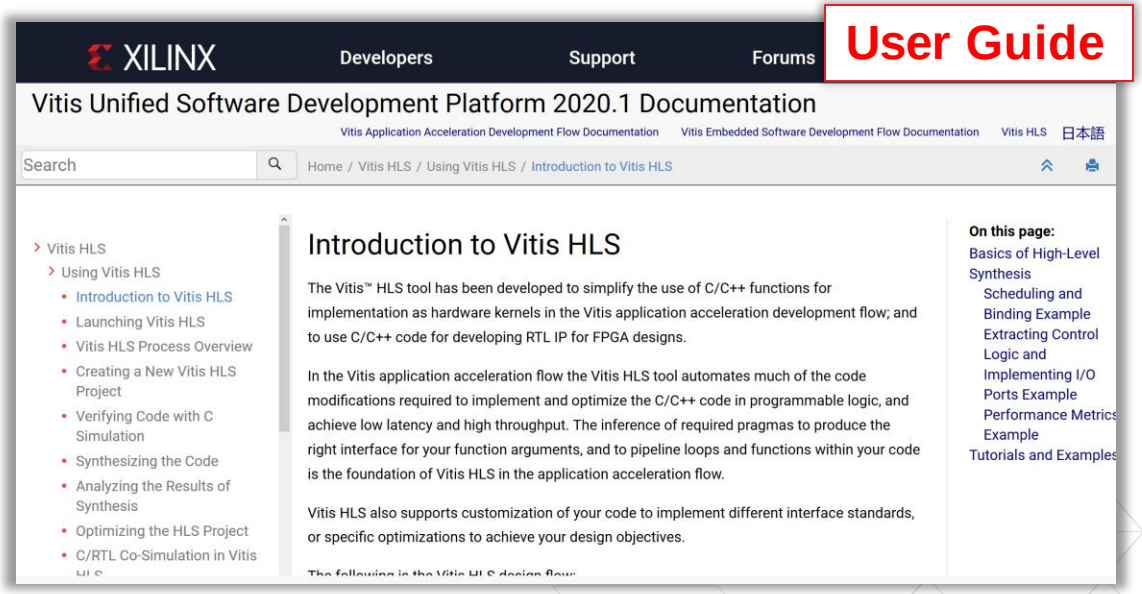
Agenda

- High-Level Synthesis (HLS) Flows
- Technical Overview + Demo Examples
- Resources

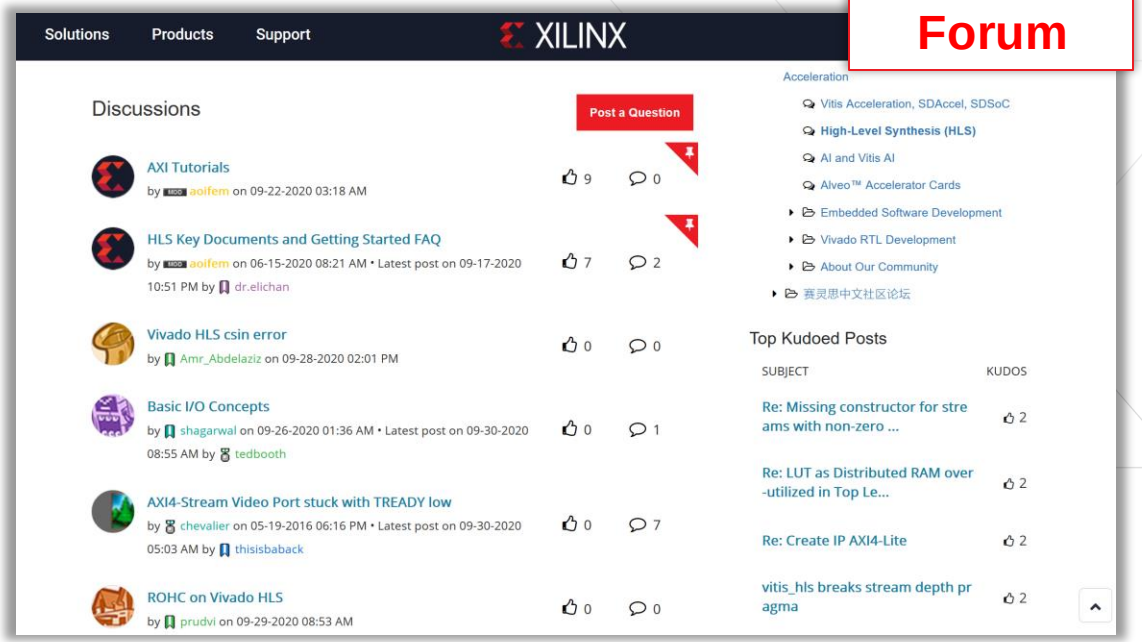
Resources – HLS

- **HTML and PDF User Guides**
- **Basic examples**
  - Github examples accessible from Vitis HLS
- **Tutorials and complete examples**
  - Github libraries: Vitis_Libraries
  - Vitis examples: Vitis_Accel_Examples
- **Forums**
  - Monitored by Xilinx support staff

User Guide



Forum



XILINX

Summary

- Vitis HLS used both in Vitis and Vivado
- C based entry boosts productivity
- Get started with examples and tutorials