

Introduction to OmpSs@FPGA

Daniel Jiménez-Gonzalez (UPC/BSC)
Xavier Martorell (UPC/BSC)

xavier.martorell@bsc.es

Programming Models and OS for FPGA-based Architectures team

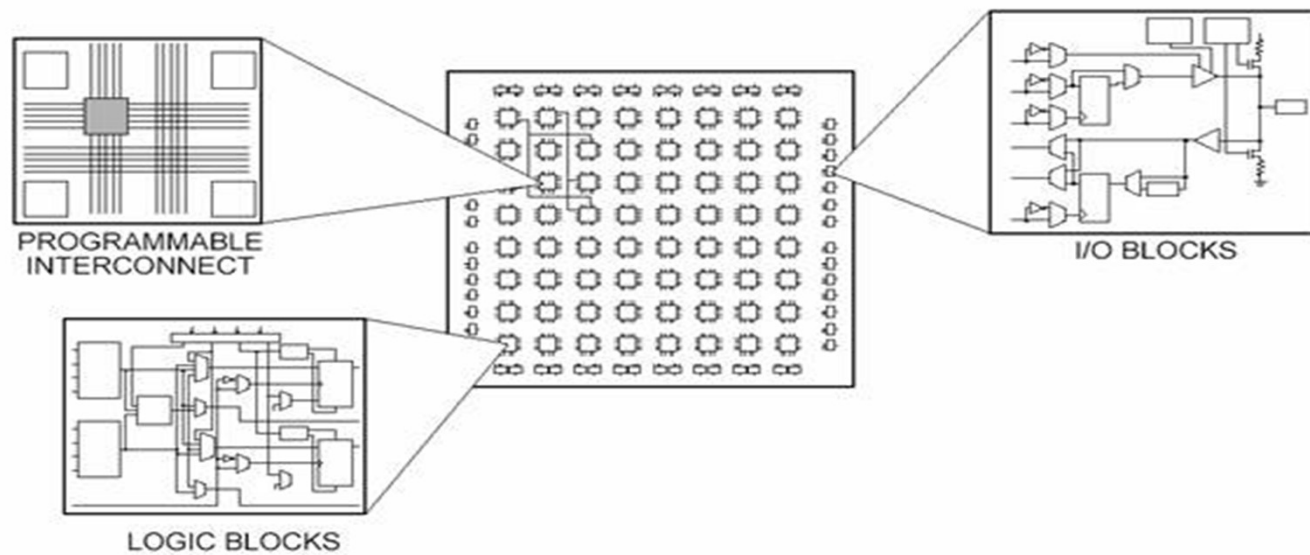
Agenda

- [OmpSs@FPGA](#) Overview and Vivado/Vitis HLS usage
 - Basic Programming with OmpSs@FPGA
 - Advanced Programming
 - Compilation and Execution
 - Matrix Multiplication example

FPGAs and Zynq

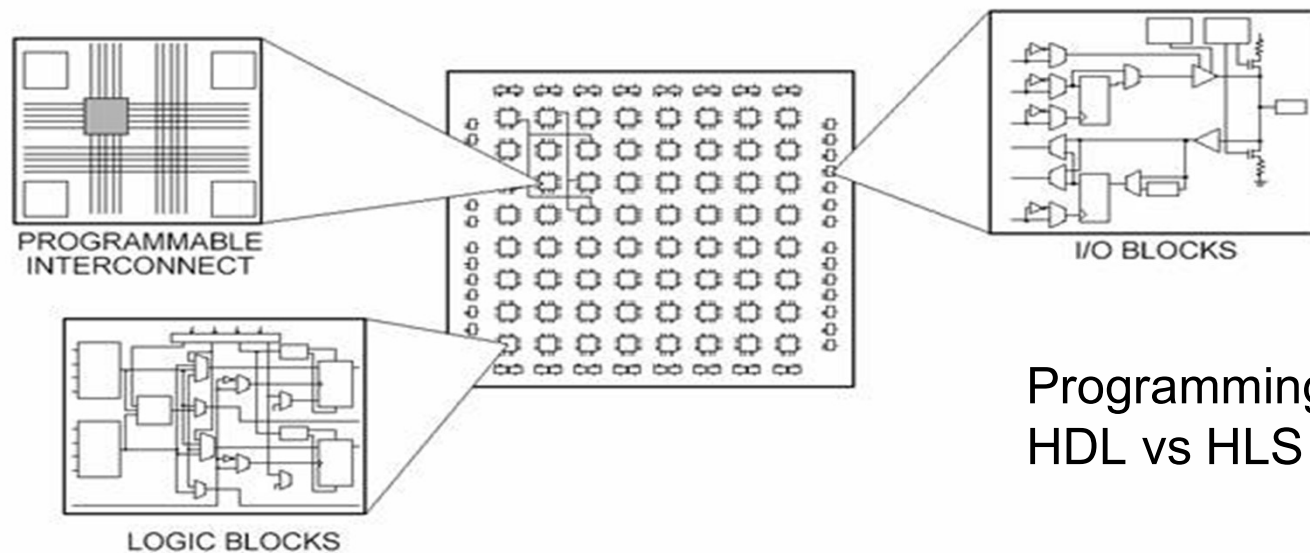
What is a FPGA?

Field Programmable Gate Array



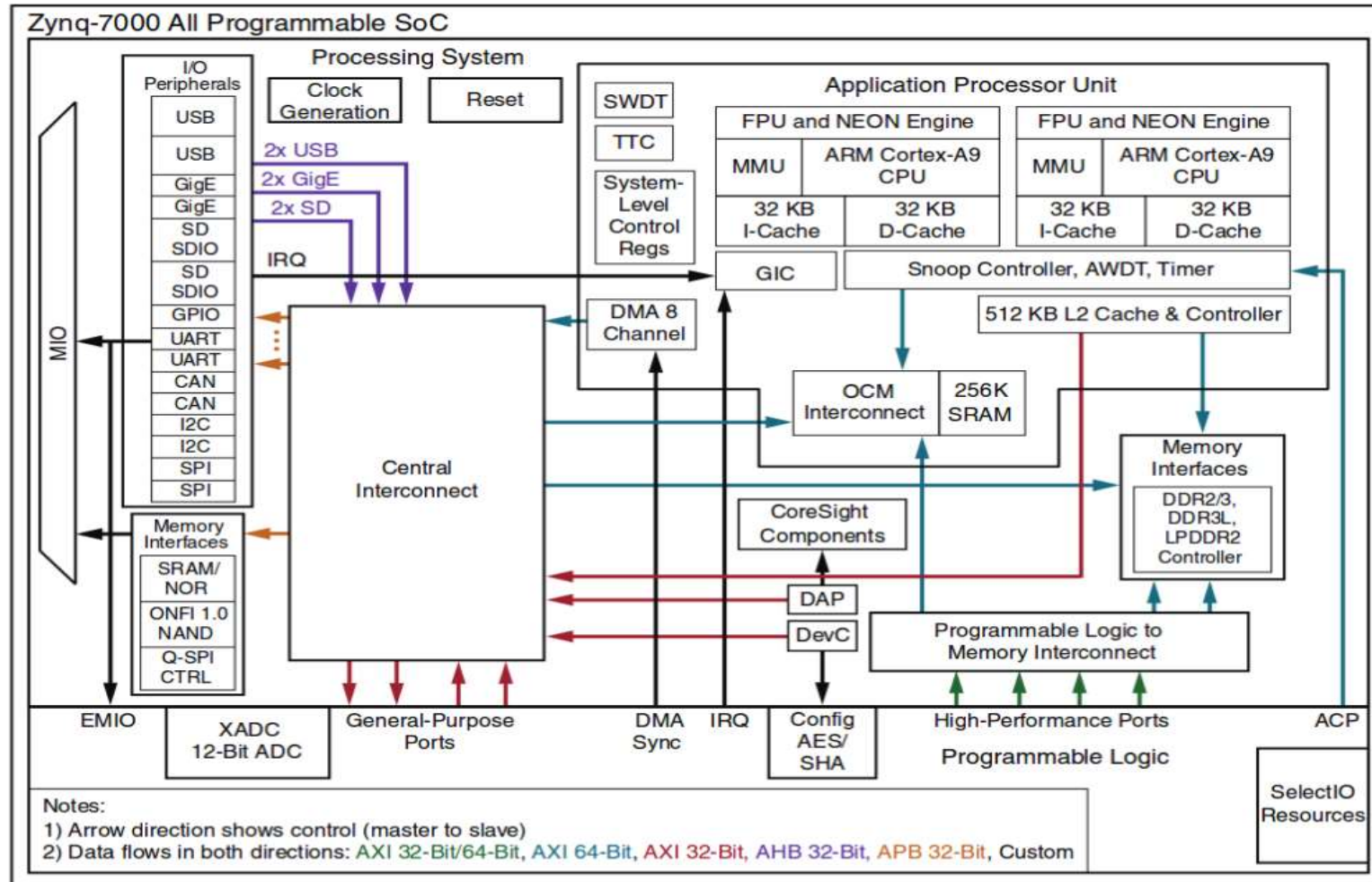
What is a FPGA?

「 FPGA stands for Field Programmable Gate Array



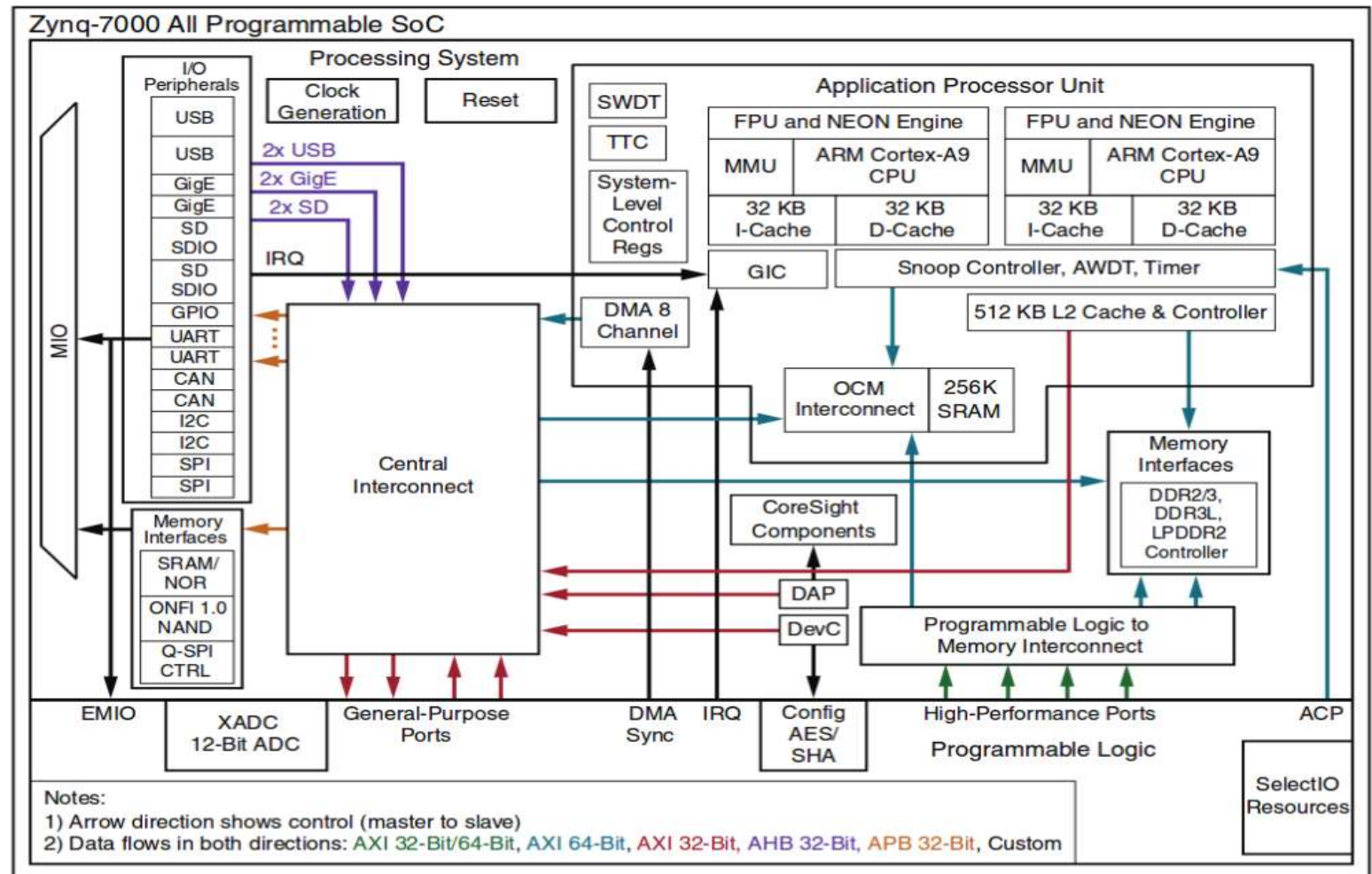
Programming:
HDL vs HLS

What is Zynq? Zynq-7000 – 32-bits

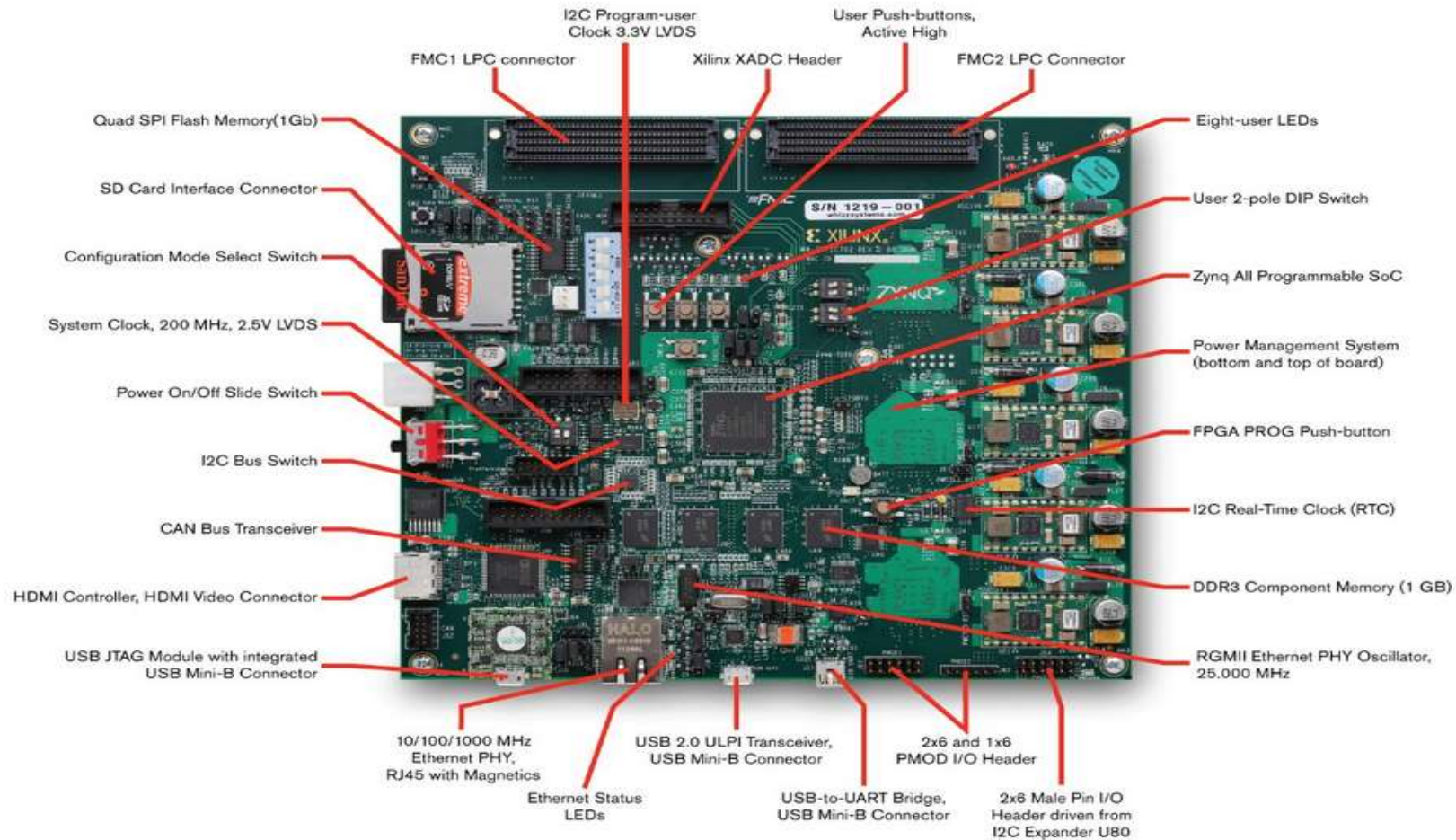


What is Zynq? Zynq-7000 – 32-bits

Programming:
HDL vs HLS
And
Heterogeneous?



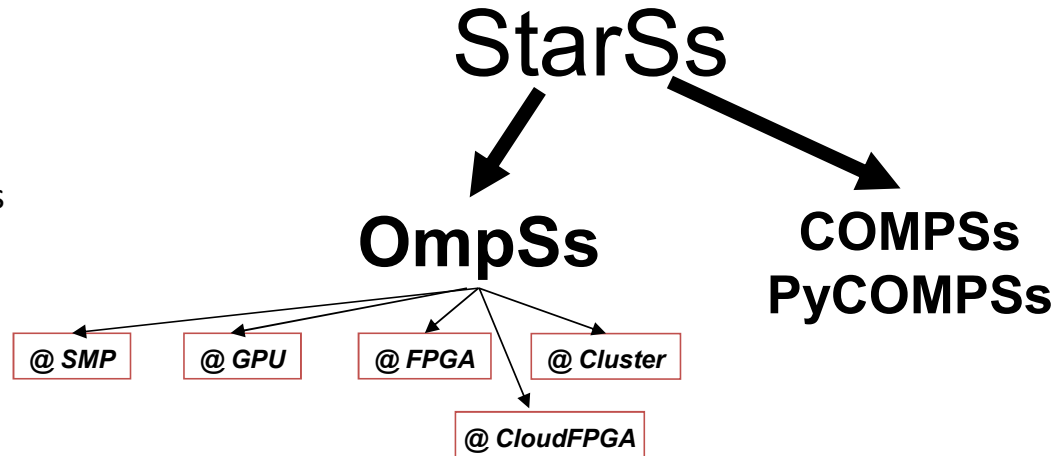
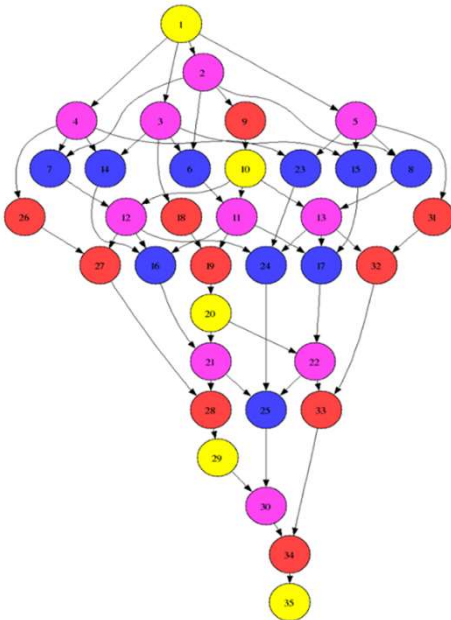
Zynq 702-board



OmpSs@FPGA Programming Model

Task-based programming

- StarSs family key concepts
 - Sequential task-based program
 - View of a single address/name space
 - Execution in parallel: automatic
 - runtime computation of dependencies
- Productivity and portability

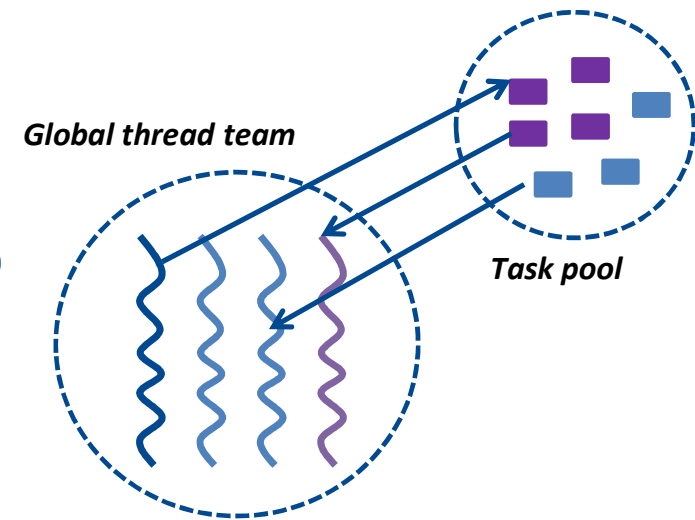


- OmpSs is used for prototyping extensions to OpenMP
 - Tasking
 - data-dependences
 - Heterogeneity
 - Multiple address spaces
 - ...
- LLVM/Clang compiler
- Nanos6 runtime system

Execution Model (OmpSs + accelerators)

Global thread team created on startup

- Master starts main task (also executes)
- N workers execute tasks
- One representative per device (accelerator)



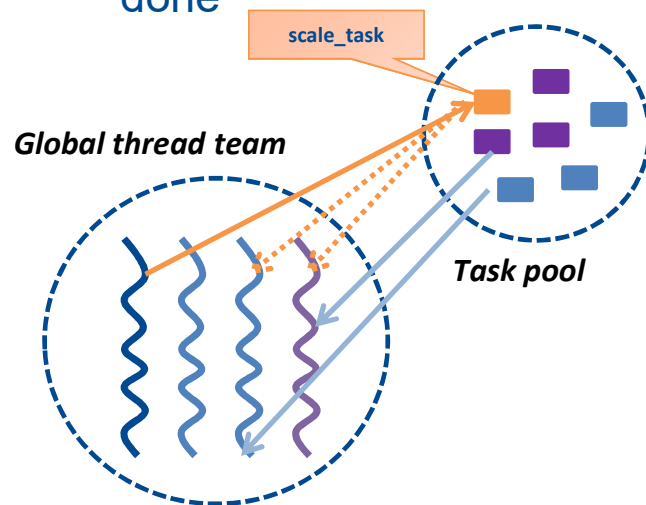
All threads get work from a task pool

- Accelerator kernels become tasks
- Tasks are labeled with a target device
 - » smp, **fpga**
- Scheduler decides which task to execute, and we can use **affinity to select an accelerator**
- FPGA tasks may be created by other FPGA tasks (**task creation**) with task dependencies (**Picos runtime**)
- Smp tasks can be created by FPGA tasks (**reverse offloading**)

OmpSs@FPGA Model Details

Very Simple Code: Execution Model

- There are a pool of threads to run tasks
- Runtime decides
 - » When a thread executes a task (it should be ready at task pool)
 - » Where to execute a task based on user annotations
 - » Which copies has to be done



```
const int SIZE=1024;
#pragma oss task in([SIZE]c,a) out([SIZE]b) device(fpga) copy_deps
    num_instances(1)
void scale_task(double *b, double *c, double *a);
```

scale.fpga.h

```
#include "scale.fpga.h"

void scale_task(double *b, double *c, double *a) {
    #pragma HLS ARRAY_PARTITION variable=c complete dim=1
    #pragma HLS ARRAY_PARTITION variable=b complete dim=1
    double alpha=*a;
    for (int j=0; j < SIZE; j++) b[j] = alpha*c[j]; }

int main (int argc, char *argv[])
{
    for (. . . ) {
        scale_task(B,C,&alpha);
    } . . .
    #pragma oss taskwait
}
```

main.cpp

Very Simple Code: Execution Model

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(1)

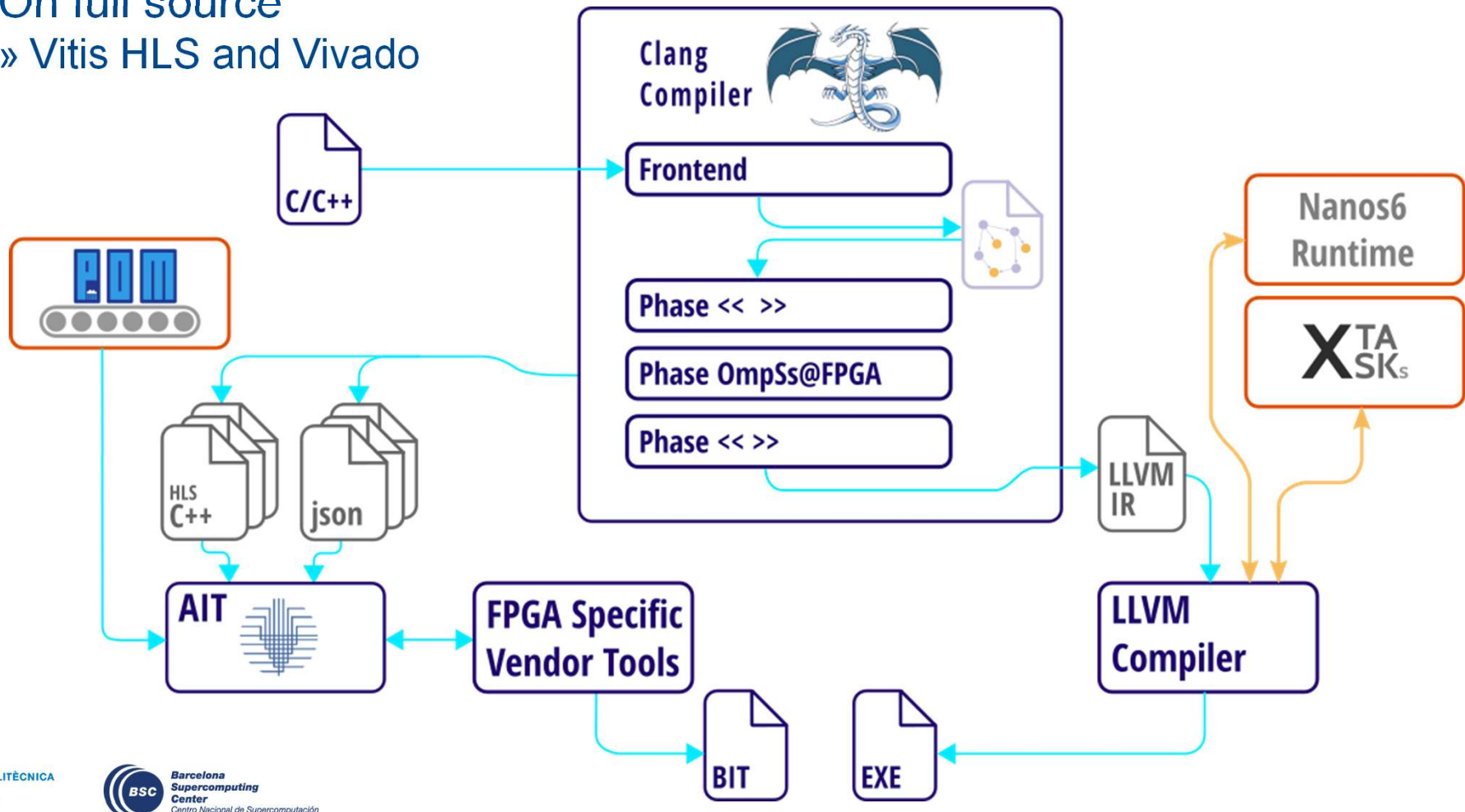
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE]) {
    unsigned int i, j, k;
    const unsigned int FACTOR=BSIZE/2;
    ...
    for (i = 0; i < BSIZE; i++) {
        for (j = 0; j < BSIZE; j++) {
            elem_t sum = c[i*BSIZE + j];
            for (k = 0; k < BSIZE; k++) {
                sum += a[i*BSIZE + k] * b[k*BSIZE + j];
            }
            c[i*BSIZE + j] = sum;
        }
    }
}
```

OmpSs@FPGA Ecosystem: Integrating FPGA compilation into OmpSs

Ecosystem: LLVM/Clang + AIT + Nanos6 + Xtasks

OmpSs transformations

- On full source
 - » Vitis HLS and Vivado



Compilation

Few details on the steps to compile

- Compiler: use clang or clang++ (C++)
 - » `compiler -fompss-2 -o program program.c`
 - » `compiler -fompss-2 -fompss-fpga-instrumentation -o program program.c`
- Details...
 - » AIT options:
 - `--board=$(BOARD_NAME),--clock=100`
 - `--name=vivado_project_name,--dir=$(VIVADO_WORKSPACE)”`

Execution

Few details on the steps to run the application

— Environment:

- Load the bitstream:
 - **fpgautil -b program.bit.bin**

— Run Program: **OMP_NUM_THREADS=NT ./program <input arguments>**

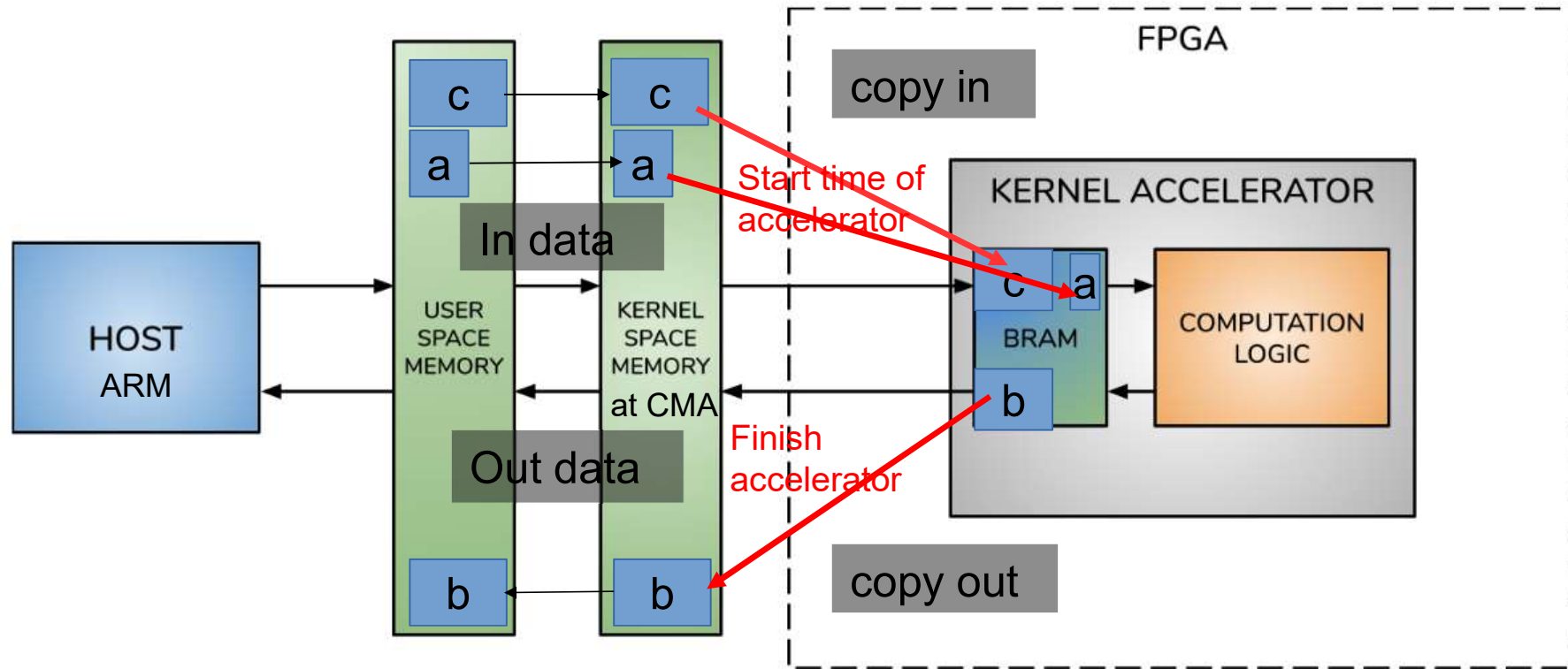
» Runtime Application Configuration:

- **program.xtasks.config** or **xtasks.config**

» With instrumentation:

- **NANOS6_CONFIG_OVERRIDE="version.instrument=ovni"**
OMP_NUM_THREADS=NT ./program <input>

Execution Model (Data Transfer): scale



- Runtime takes care of data transfer from user to kernel
- Advanced: **Wide Port and Interleave/Priorities** (No Zynq)

Code transformations

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(1)

void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE]) {
    unsigned int i, j, k;
    const unsigned int FACTOR=BSIZE/2;
    ...
    for (i = 0; i < BSIZE; i++) {
        for (j = 0; j < BSIZE; j++) {
            elem_t sum = c[i*BSIZE + j];
            for (k = 0; k < BSIZE; k++) {
                sum += a[i*BSIZE + k] * b[k*BSIZE + j];
            }
            c[i*BSIZE + j] = sum;
        }
    }
}
```

- “ Wrapper for communication is **automatically generated**
- **Local declarations**
 - **Copy_deps**, local copy of TILE
 - Call to user kernel

```

void matmulBlock_hw(elem_t a[1024], elem_t b[1024], elem_t c[1024], hls::stream<__mcxx_instrData_t> &mcxx_instr) {
    unsigned int i, j, k;
    loop_i_matmul:
    for (i = 0; i < 32; i++) {
        loop_j_matmul:
        for (j = 0; j < 32; j++) {
            elem_t sum = c[i * 32 + j];
            loop_k_matmul:
            for (k = 0; k < 32; k++) {
                sum += a[i * 32 + k] * b[k * 32 + j];
            }
            c[i * 32 + j] = sum;
        }
    }
}

const unsigned int CONST_BSIZE = 32;
void mcxx_write_out_port(const ap_uint<64> data, const ap_uint<2> dest, const ap_uint<1> last, hls::stream<mcxx_outaxis>& mcxx_outPort) {
    #pragma HLS inline
    mcxx_outaxis axis_word;
    axis_word.data = data;
    ./.fpga-naive-zedboard/HLS-32x32-naive/zedboard/matmul_ait/xilinx/HLS/matmulBlock_hw/matmulBlock_hw_hls_automatic_clang.cpp
    ap_wait();
}
mcxx_instrument_event(78, __mcxx_taskId, mcxx_instr);
if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
}
if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
}
if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
}
}
mcxx_instrument_event(206, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(80, __mcxx_taskId, mcxx_instr);
matmulBlock_hw(a, b, c, mcxx_instr);
mcxx_instrument_event(208, __mcxx_taskId, mcxx_instr);

```

- Kernel is the same
- Automatic instrumentation
- Connection through AXI ports

Execution Model: Compiler Support

```
void matmulBlock_hw_wrapper(hls::stream<ap_uint<
rData_t>& mcxx_instr) {
#pragma HLS interface ap_ctrl_none port=return
#pragma HLS interface axis port=mcxx_inPort
#pragma HLS interface axis port=mcxx_outPort
#pragma HLS interface m_axi port=mcxx_a
#pragma HLS interface m_axi port=mcxx_b
#pragma HLS interface m_axi port=mcxx_c
#pragma HLS interface axis port=mcxx_instr

    static elem_t a[1024];
    static elem_t b[1024];
    static elem_t c[1024];
```

Local
declaration for
local copies

Execution Model: Compiler Support

```
    ap_wait();
}
mcxx_instrument_event(78, __mcxx_taskId, mcxx_instr);
if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
}
if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
}
if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
}
mcxx_instrument_event(206, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(80, __mcxx_taskId, mcxx_instr);
    matmulBlock_hw(a, b, c, mcxx_instr);
mcxx_instrument_event(208, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(79, __mcxx_taskId, mcxx_instr);
if (mcxx_flags_2[5]) {
    memcpy(mcxx_c + mcxx_offset_2/sizeof(elem_t), c, (4096));
}
}
```

- Local copies and management in the wrapper
- Local copies using ports

Execution Model: Compiler Support

```
void matmulBlock_hw(elem_t a[1024], elem_t b[1024], elem_t c[1024], hls::stream<__mcxx_instrData_t> &mcxx_instr) {
    unsigned int i, j, k;
    loop_i_matmul:
    for (i = 0; i < 32; i++) {
        loop_j_matmul:
        for (j = 0; j < 32; j++) {
            elem_t sum = c[i * 32 + j];
            loop_k_matmul:
            for (k = 0; k < 32; k++) {
                sum += a[i * 32 + k] * b[k * 32 + j];
            }
            c[i * 32 + j] = sum;
        }
    }
}
```

- Kernel is the same
- Local copies and management in the wrapper

```
const unsigned int CONST_BSIZE = 32;
void mcxx_write_out_port(const ap_uint<64> data, const ap_uint<2> dest, const ap_uint<1> last, hls::stream<mcxx_outaxis>& mcxx_outPort) {
    #pragma HLS inline
    mcxx_outaxis axis_word;
    axis_word.data = data;
    ap_wait();
}
```

./fpga-naive-zedboard/HLS-32x32-naive/zedboard/matmul_ait/xilinx/HLS/matmulBlock_hw/matmulBlock_hw_hls_automatic_clang.cpp

39,1

25%

```
ap_wait();
}
mcxx_instrument_event(78, __mcxx_taskId, mcxx_instr);
if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
}
if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
}
if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
}
mcxx_instrument_event(206, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(80, __mcxx_taskId, mcxx_instr);
matmulBlock_hw(a, b, c, mcxx_instr);
mcxx_instrument_event(208, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(79, __mcxx_taskId, mcxx_instr);
if (mcxx_flags_2[5]) {
    memcpy(mcxx_c + mcxx_offset_2/sizeof(elem_t), c, (4096));
}
}
```

- Local copies using ports

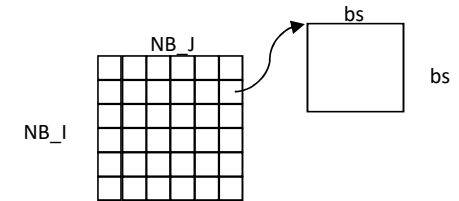
```
void matmulBlock_hw_wrapper(hls::stream<ap_uint<64>>& rData_t & mcxx_instr) {
    #pragma HLS interface ap_ctrl_none port=return
    #pragma HLS interface axis port=mcxx_inPort
    #pragma HLS interface axis port=mcxx_outPort
    #pragma HLS interface m_axi port=mcxx_a
    #pragma HLS interface m_axi port=mcxx_b
    #pragma HLS interface m_axi port=mcxx_c
    #pragma HLS interface axis port=mcxx_instr
    static elem_t a[1024];
    static elem_t b[1024];
    static elem_t c[1024];
}
```

Local declaration for local copies

OmpSs@FPGA MxM Example: num_instances and HLS kernel

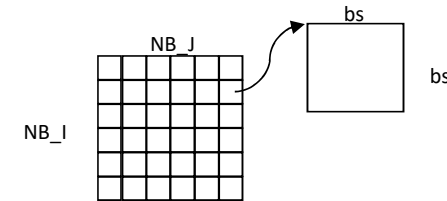
Sequential Example: Tiled MxM (2D structure)

```
// FPGA
void matrix_multiply(T a[BS][BS], T b[BS][BS], T c[BS][BS]);
...
// HOST
for (i_b=0; i_b<NB_I; i_b++)
    for (j_b=0; j_b<NB_J; j_b++)
        for (k_b=0; k_b<NB_K; k_b++)
            matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
...
```



OmpSs Example: Tiled MxM

```
#pragma oss task device(fpga) in([BS]a,[BS]b) inout([BS]c) \
copy_deps num_instances(1)
void matrix_multiply(T a[BS][BS],T b[BS][BS],T c[BS][BS]);
...
for (i_b=0; i_b<NB_I; i_b++)
    for (j_b=0; j_b<NB_J; j_b++)
        for (k_b=0; k_b<NB_K; k_b++)
            matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
...
#pragma oss taskwait
// Or
// other tasks depending on input output c of matrix multiply task
```

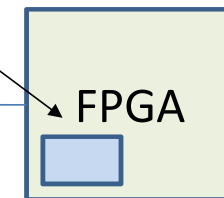
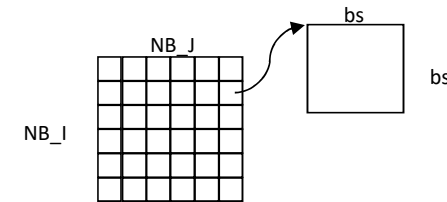


OmpSs Example: Tiled MxM (1 acc)

LLVM/Clang (OmpSs-2 compiler) with AIT* tool

- Few extensions (num_instances)
- Triggers the bitstream generation automatically (Stub function generated)

```
#pragma oss task device(fpga) in([BS]a,[BS]b) inout([BS]c) \  
copy_deps num_instances(1)  
  
void matrix_multiply(T a[BS][BS], T b[BS][BS], T c[BS][BS]);  
  
...  
for (i_b=0; i_b<NB_I; i_b++)  
    for (j_b=0; j_b<NB_J; j_b++)  
        for (k_b=0; k_b<NB_K; k_b++)  
            matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);  
  
...  
#pragma oss taskwait  
// Or  
// other tasks depending on input output c of matrix multiply task
```



OmpSs Example: Tiled MxM (1 acc)

LLVM/Clang (OmpSs-2 compiler) with AIT tool

- Few extensions (num_instances)
- Triggers the bitstream generation automatically (Stub function generated)

```
#pragma omp taskwait (5) \
copy_deps num_instances(2)
```

```
void matrix_multiply(T a[BS][BS], T b[BS][BS], T c[BS][BS]);
```

```
...
```

```
for (i_b=0; i_b<NB_I; i_b++)
```

```
    for (j_b=0; j_b<NB_J; j_b++)
```

```
        for (k_b=0; k_b<NB_K; k_b++)
```

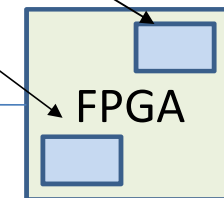
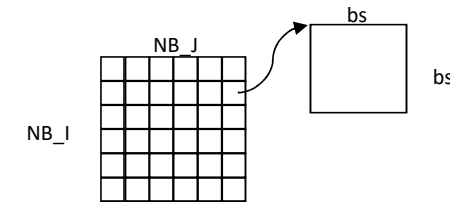
```
            matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
```

```
...
```

```
#pragma omp taskwait
```

```
// Or
```

```
// other tasks depending on input output c of matrix multiply task
```



OmpSs Example: MxM kernel + Vitis HLS

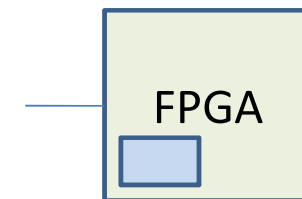
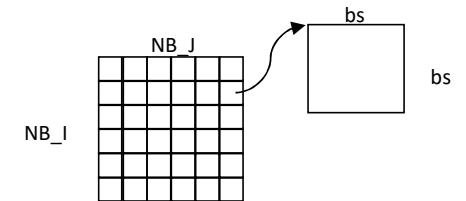
LLVM/Clang (Ompss-2 compiler) with AIT tool

- Few extensions (num_instances)
- Triggers the bitstream generation automatically (Stub function generated)

```
#pragma oss task device(fpga) in([BS]a,[BS]b) inout([BS]c) \  
copy_deps num_instances(1)
```

```
void matrix_multiply(T a[BS][BS],T b[BS][BS],T c[BS][BS]);
```

```
void matrix_multiply(float a[BS][BS], float b[BS][BS],float c[BS][BS])  
{  
    #pragma HLS inline  
    int const FACTOR = BS/2;  
    #pragma HLS array_partition variable=a block factor=FACTOR dim=2  
    #pragma HLS array_partition variable=b block factor=FACTOR dim=1  
    // matrix multiplication of a A*B matrix  
    for (int ia = 0; ia < BS; ++ia)  
        for (int ib = 0; ib < BS; ++ib) {  
            #pragma HLS PIPELINE II=1  
            float sum = 0;  
            for (int id = 0; id < BS; ++id)  
                sum += a[ia][id] * b[id][ib];  
            c[ia][ib] += sum;  
        }  
}
```



AMD-Xilinx Zynq Support ++

OmpSs FPGA Support

```
void matrix_multiply(float a[BS][BS], float b[BS][BS], float c[BS][BS])
{
    #pragma HLS inline
    int const FACTOR = BS/2;
    #pragma HLS array_partition variable=a block factor=FACTOR dim=2
    #pragma HLS array_partition variable=b block factor=FACTOR dim=1
    // matrix multiplication of a A*B matrix
    for (int ia = 0; ia < BS; ++ia)
        for (int ib = 0; ib < BS; ++ib) {
            #pragma HLS PIPELINE II=1
            float sum = 0;
            for (int id = 0; id < BS; ++id)
                sum += a[ia][id] * b[id][ib];
            c[ia][ib] += sum;
        }
}
```

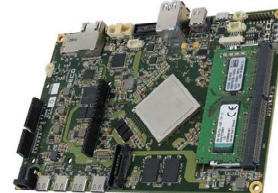
Discrete FPGAs like
Alveo and Cloud (IMP)



Kria (v260som)* &
ZCU102 Board
XCZU9EG-2FFVB1156

Trenz Electronics Zynq U+
TE0808 XCZU9EG-ES1

SECO AXIOM Board
Zynq U+ XCZU9EG-ES2



4x Cortex-A53 cores + FPGA (64-bit platforms)



Zynq-7000 Family
2x Cortex-A9 cores + FPGA (32-bit platforms)



Introduction to OmpSs@FPGA

Thank you!!