Firmware development for hybrid processors (ARM and FPGA) computing

Mirko Mariotti [1,2] Giulio Bianchini [1] Loriano Storchi [3,2] Giacomo Surace [1]
Daniele Spiga [2]

[1]Dipartimento di Fisica e Geologia, Universitá degli Studi di Perugia

[2]INFN sezione di Perugia

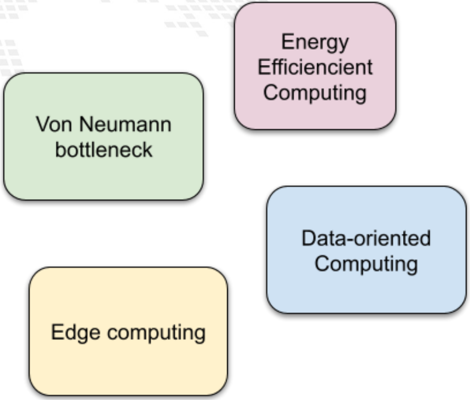
[3]Dipartimento di Farmacia, Universitá degli Studi G. D'Annunzio

Outline

Introduction

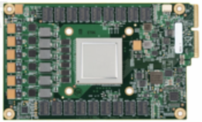
Evolution of computing

- New challenges



Evolution of computing

- New challenges
- New architectures



Evolution of computing

- New challenges
- New architectures
- From the software point of view: Heterogeneity

```python
with ipu.scopes.ipu_scope('/device:IPU:0'):
    training_loop_body_on_ipu = ipu.ipu_compiler.compile(computation=training_loop_body, inputs=[x, y])

ipu_configuration = ipu.config.IPUConfig()
ipu_configuration.auto_select_ipus = 1
ipu_configuration.configure_ipu_system()
```

```c
e_load_group("e_math_test.srec", &dev, 0, 0, platform.rows,  platform.cols, E_FALSE);
e_load_group("e_math_test1.srec", &dev, 0, 0, platform.rows, platform.cols, E_FALSE);
```

```c
const char *kernel =
"__kernel                                    \n"
"void kernel(float alpha,     \n"
"                  __global float *A,        \n"
"                  __global float *B,        \n"
"                  __global float *C)        \n"
"{                                           \n"
"    int index = get_global_id(0);           \n"
"    C[index] = alpha* A[index] + B[index]; \n"
"}                                           \n";

cl_program program = clCreateProgramWithSource(context, 1,(const char **)&kernel, NULL, &clStatus);
```

Evolution of computing

- New challenges
- New architectures
- From the software point of view: Heterogeneity
- From the hardware point of view: Accelerators

Accelerators

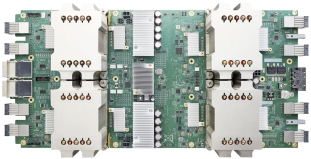
Hardware device or software program designed to improve the performance of certain workload.

Graphics Processing Unit (GPU)

- High Data Throughput
- Massive Parallel Computing

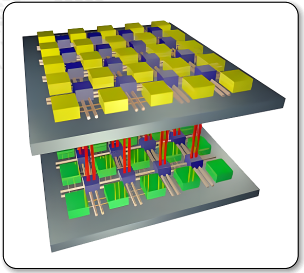
Application-Specific Integrated Circuit (ASIC)

- Highly specialized for a task
- Energy efficient (due to specialization)

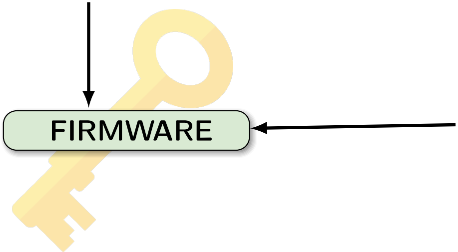


FPGA accelerator

A field programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable.



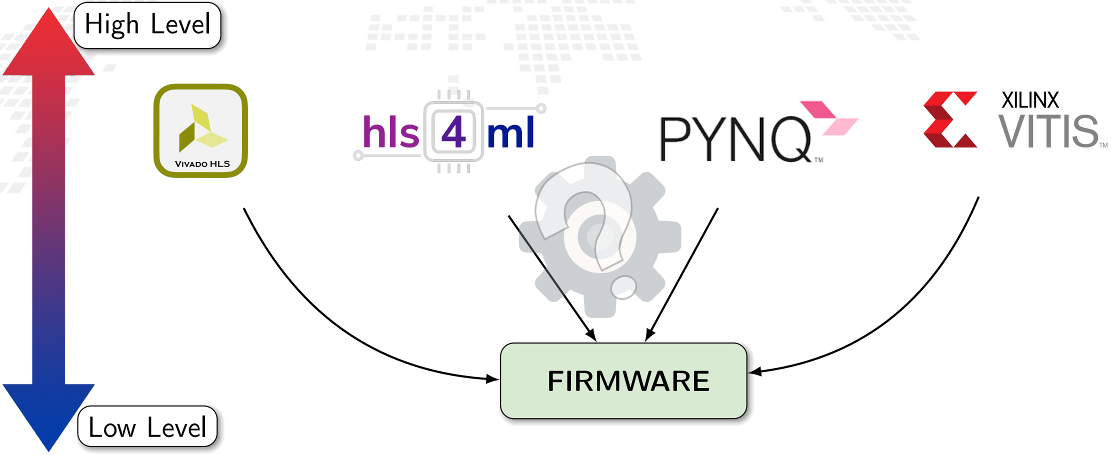
- Parallel computing
- Highly specialized
- Energy efficient



- Array of programmable logic blocks
- Logic blocks configurable to perform complex functions
- The configuration is specified with the hardware description language

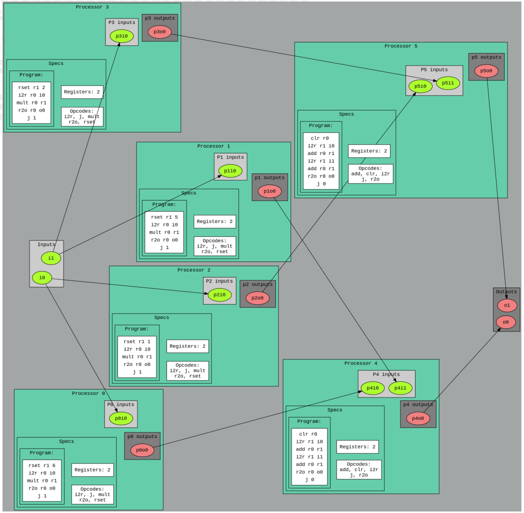
Firmware generation

Many projects have the goal of abstracting the firmware generation process.



BondMachine

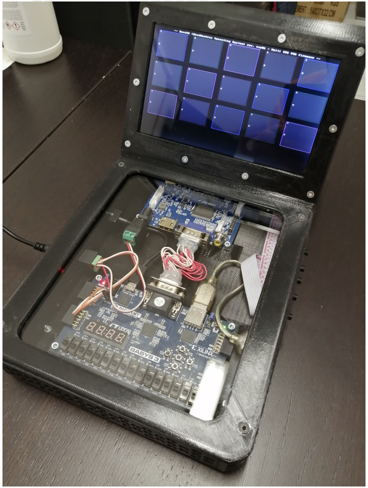
- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized of FPGA and



BondMachine

- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized of FPGA and

- used as standalone devices,

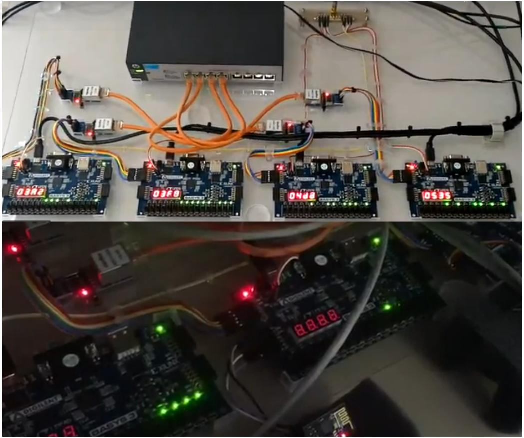


BondMachine

- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized of FPGA and

- used as standalone devices,

- as clustered devices,

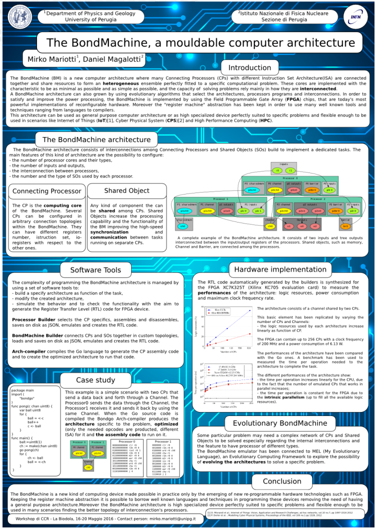


BondMachine

- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized of FPGA and
- used as standalone devices,
- as clustered devices,
- and, as in this talk, as firmware for computing accelerators.

BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk



BondMachine

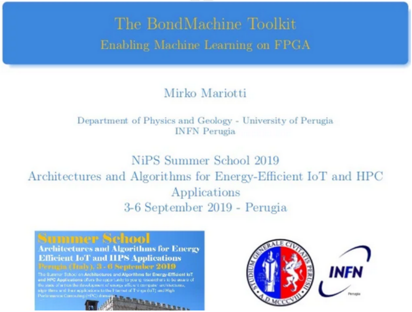
- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA

BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019

BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"



BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS

BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022

ELSEVIER
Parallel Computing
Volume 109, March 2022, 102873

The BondMachine, a moldable computer architecture

Mirko Mariotti [a,b], Daniel Magalotti [b], Daniele Spiga [b], Loriano Storchi [c,b]

Show more

+ Add to Mendeley Share Cite

https://doi.org/10.1016/j.parco.2021.102873 Get rights and content

Highlights

- Co-design HW/SW of domain specific architectures via the modern GO language.
- Design of essential processors where only needed components are implemented.
- Creation of heterogeneous processor systems distributed over multiple fabrics.

BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022
- PON PHD program

An accelerated system from ground up

Specs

FPGA

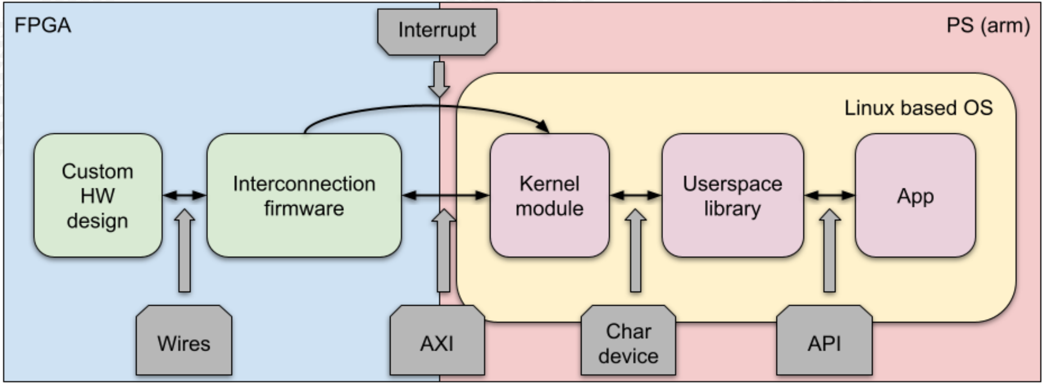
- Digilent Zedboard
- Soc: Zynq XC7Z020-CLG484-1
- 512 MB DDR3
- Vivado 2020.2

Workstations

- Dell Precision Tower 3620
- Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz
- 16GB Ram
- Golang 1.18.1

- Intel(R) CPU I5-8500 v5 @ 3GHz
- 16GB Ram
- GCC with -O0

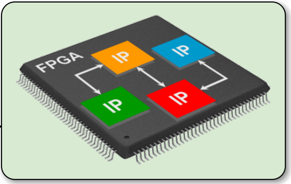
The whole system overview

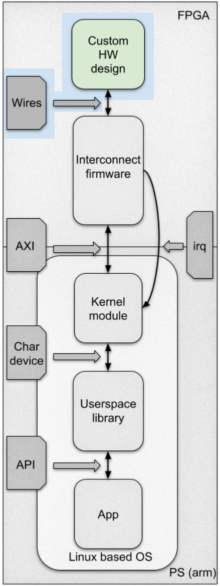


The Accelerator IP

Hardware Description Language

```
module Mat_mult(A,B,Res);
    input [31:0] A;
    input [31:0] B;
    output [31:0] Res;
    //internal variables
    reg [31:0] Res;
    reg [7:0] A1 [0:1][0:1];
    reg [7:0] B1 [0:1][0:1];
    reg [7:0] Res1 [0:1][0:1];
    integer i,j,k;

    always@ (A or B)
    begin
        {A1[0][0],A1[0][1],A1[1][0],A1[1][1]} = A;
        {B1[0][0],B1[0][1],B1[1][0],B1[1][1]} = B;
        i = 0;
        j = 0;
        k = 0;
        {Res1[0][0],Res1[0][1],Res1[1][0],Res1[1][1]} = 32'd0;
        for(i=0;i < 2;i=i+1)
            for(j=0;j < 2;j=j+1)
                for(k=0;k < 2;k=k+1)
                    Res1[i][j] = Res1[i][j] + (A1[i][k] * B1[k][j]);
        Res = {Res1[0][0],Res1[0][1],Res1[1][0],Res1[1][1]};
    end
endmodule
```



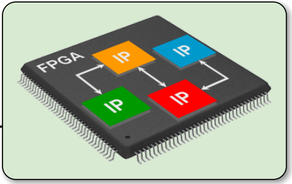


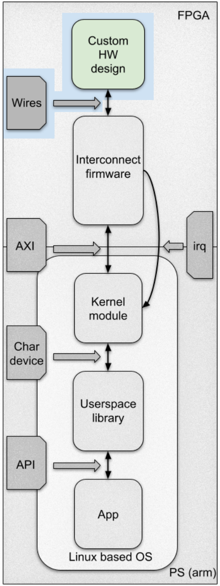
The Accelerator IP

Hardware Description Language

High Level Synthesis

```
template <typename T, int DIM>
void mmult_hw(T A[DIM][DIM], T B[DIM][DIM], T C[DIM][DIM])
{
    // matrix multiplication of a A*B matrix
    L1:for (int ia = 0; ia < DIM; ++ia)
    {
        L2:for (int ib = 0; ib < DIM; ++ib)
        {
            T sum = 0;
            L3:for (int id = 0; id < DIM; ++id)
            {
                sum += A[ia][id] * B[id][ib];
            }
            C[ia][ib] = sum;
        }
    }
}
```



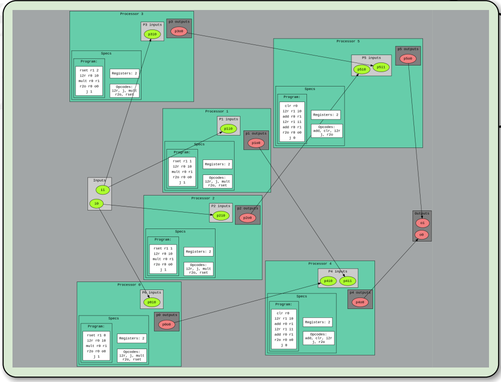


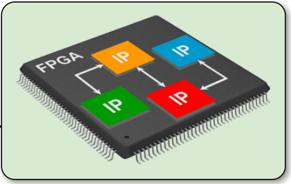
The Accelerator IP

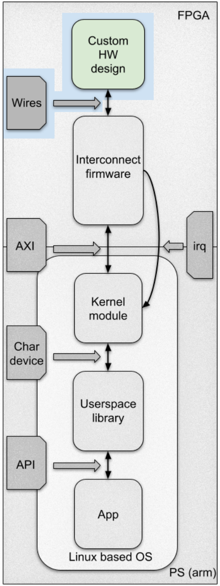
Hardware Description Language

High Level Synthesis

BondMachine





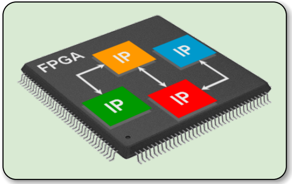


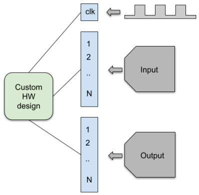
The Accelerator IP

Hardware Description Language

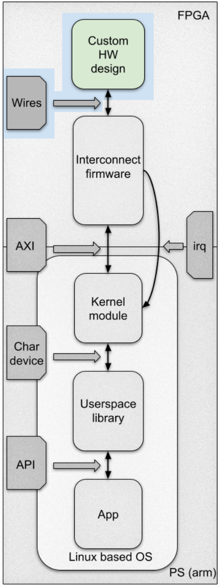
High Level Synthesis

BondMachine



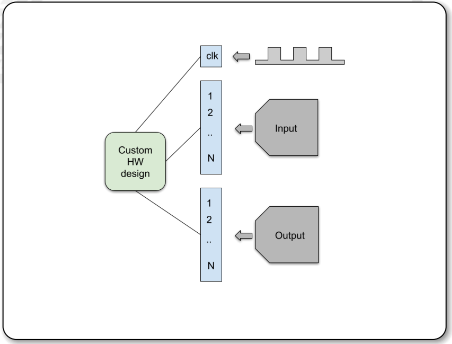


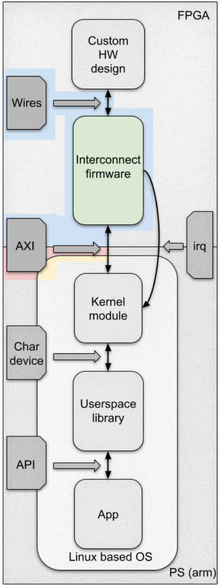
Wires:
- a clock signal,
- an input bus,
- an output bus for the result



Interconnection firmware

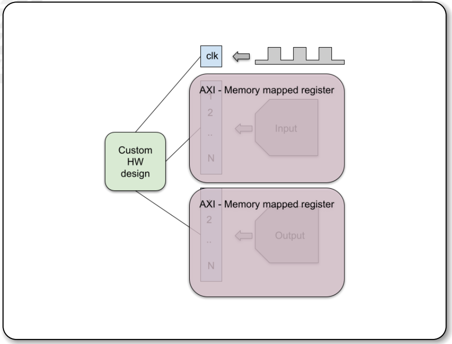
The input and output buses are the endpoints that we would like to have on the linux system.

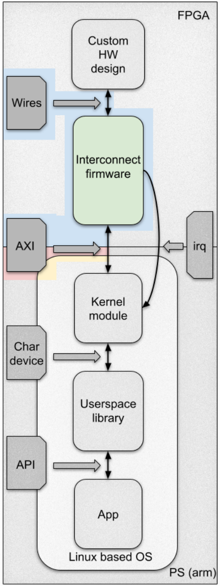




Interconnection firmware

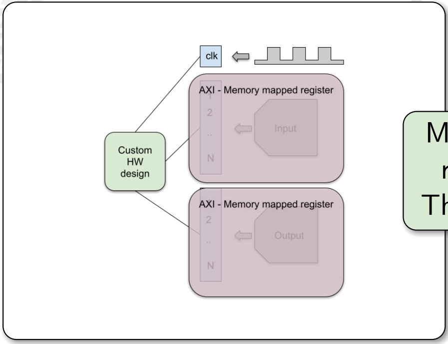
The input and output buses are the endpoints that we would like to have on the linux system.





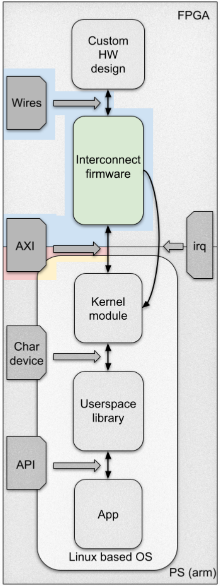
Interconnection firmware

The input and output buses are the endpoints that we would like to have on the linux system.



Memory mapped registers using The AXI protocol

```
wire [31:0] states;
wire [31:0] changes;
wire [31:0] DVDR_PS2PL;
wire [31:0] DVDR_PL2PS;
wire [31:0] port_o0;
wire [31:0] port_o1;
wire [31:0] port_o2;
wire [31:0] port_i0;
wire [31:0] port_i1;
wire [31:0] port_i2;

bondmachine_main bondmachine_inst(
    .clk(S_AXI_ACLK),
    .btnC(btnC),
    .A_DVDR_PS2PL(DVDR_PS2PL),
    .A_DVDR_PL2PS(DVDR_PL2PS),
    .A_changes(changes),
    .A_states(states),
    .A_port_o0(port_o0),
    .A_port_o1(port_o1),
    .A_port_o2(port_o2),
    .A_port_i0(port_i0),
    .A_port_i1(port_i1),
    .A_port_i2(port_i2),
    .interrupt(interrupt)
);
assign port_i0 = slv_reg0[31:0];
assign port_i1 = slv_reg1[31:0];
assign port_i2 = slv_reg2[31:0];
assign DVDR_PS2PL = slv_reg3[31:0];
assign states = slv_reg4[31:0];

always @( posedge S_AXI_ACLK )
begin
    slv_reg5 <= port_o0[31:0];
    slv_reg6 <= port_o1[31:0];
    slv_reg7 <= port_o2[31:0];
    sv_reg8 <= DVDR_PL2PS[31:0];
    slv_reg9 <= changes[31:0];
end
```

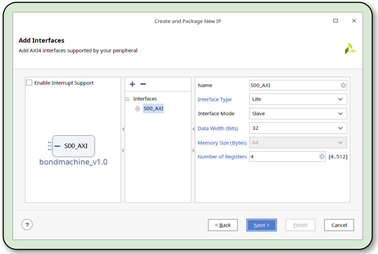


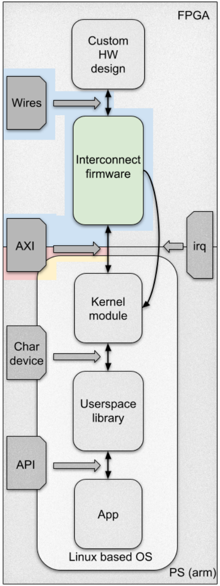
The Advanced eXtensible Interface Protocol

AXI is a communication bus protocol defined by ARM as part of the Advanced Microcontroller Bus Architecture (AMBA) standard.

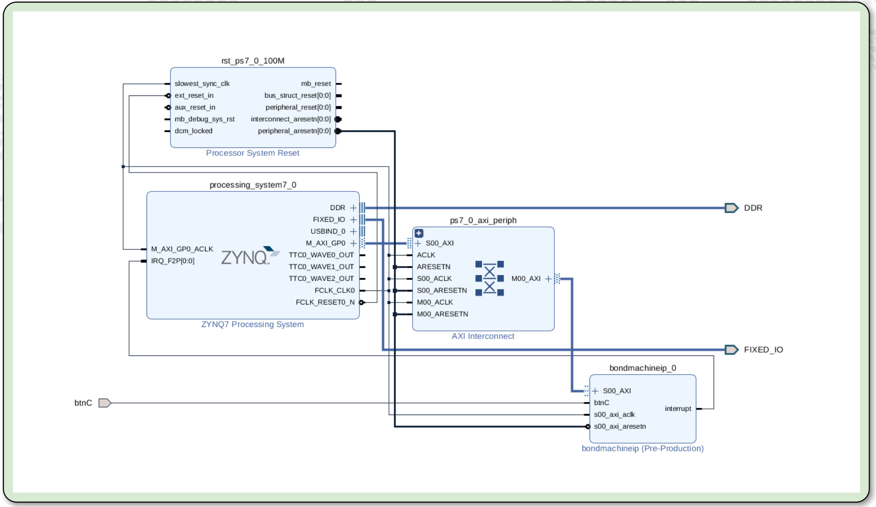
There are 3 types of AXI Interfaces:

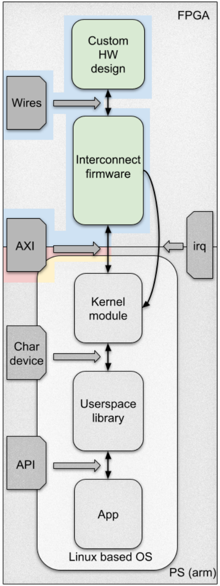
- AXI Full: for high-performance memory-mapped requirements.
- AXI Lite: for low-throughput memory-mapped communication.
- AXI Stream: for high-speed streaming data.





Block Design





Linux

Now that we have a custom accelerated hardware, we need a Linux distro to run on it.

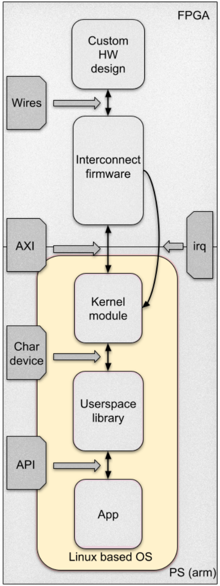
Common Features

Complete system build from source
Allow choice of kernel and bootloader
Support for modifying packages with patches or custom configuration files
Can build cross-toolchains for development
Convenient support for read-only root filesystems
Support offline builds
The build configuration files integrate well with SCM tools

- **Yocto**
  Convenient sharing of build configuration among similar projects (meta-layers)
  Larger community (Linux Foundation project)
  Can build a toolchain that runs on the target
  A package management system

- **Buildroot**
  Simple Makefile approach, easier to understand how the build system works
  Reduced resource requirements on the build machine
  Very easy to customize the final root filesystem (overlays)

Credits: https://jumpnowtek.com/linux/Choosing-an-embedded-linux-build-system.html



Ingredients to build the distro

Kernel config

Linux kernel configuration

Boot Loader config

U-boot Boot loader configuration

Device tree

Data structure describing hardware components

Used by: Kernel Boot loader

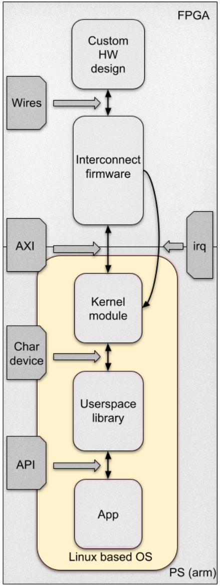
Created by: Vivado + GCC

Bitstream (firmware)

File describing Cells and routing of the FPGA

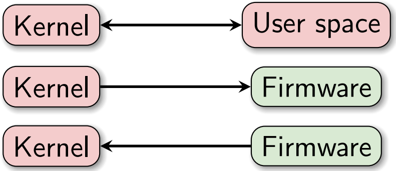
Used by Boot loader to programm FPGA

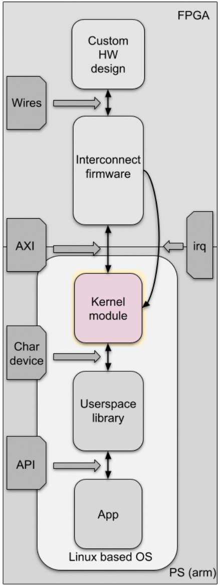
Created by: Vivado



kernel module

- The accelerator endpoints are exposed via AXI memory-mapped as memory location of the arm processor running Linux.
- To properly use the accelerator from user space, the kernel has to handle the accelerator endpoints and make them available to user space.
- We developed a kernel module for our accelerators. It manages 3 data flows:





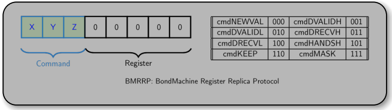
Kernel from an to user space: char device

The communication are through the standard read and write system call on a kernel generated char device

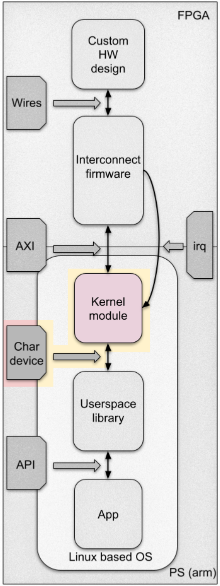
A language has been implemented for the desired operations

```c
static ssize_t bm_read(struct file *filp, char __user *buf, size_t len, loff_t *off)
{
        struct work_data *writework;

        wait_event_interruptible(wait_queue_bm, wait_queue_flag != 0);

        switch (wait_queue_flag)
        {
        case 1:
                switch (bmacc_state)
                {
                case stateHSRECV:
                        if (copy_to_user(buf, &hsmask, 1))
                        {
                                pr_err("Data Read : Err!\n");
                        }
                        //pr_info("Sent MASK");
                        bmacc_state = stateMASKSENT;
                        wait_queue_flag = 0;
                        return 1;
                        brealk
// ....
```

```c
static ssize_t bm_write(struct file *filp, const char __user *buf, size_t len, loff_t *off)
{
        struct work_data *writework;

        if (copy_from_user(write_buffer, buf, len))
        {
                pr_err("data write error\n");
        }
        else
        {
                for (i = 0; i < len; i++)
                {
                        switch (bmacc_state)
                        {
                        case stateWAIT:
                                switch (write_buffer[i] & cmdMASK)
                                {
                                case cmdHANDSH:
                                        //pr_info("HandShake");
                                        hs = write_buffer[i];
                                        bmacc_state = stateHSRECV;
                                        wait_queue_flag = 0;
                                        writework = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                                        INIT_WORK(&writework->work, work_handler);
                                        writework->cs = 1;
                                        queue_work(wq, &writework->work);
                                        break;
```

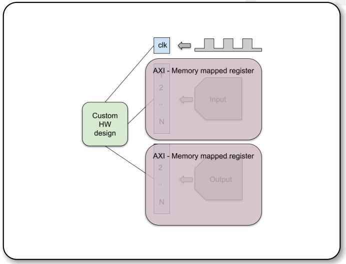


cmdNEWVAL	000	cmdDVALIDH	001
cmdDVALIDL	010	cmdDRECVH	011
cmdDRECVL	100	cmdHANDSH	101
cmdKEEP	110	cmdMASK	111



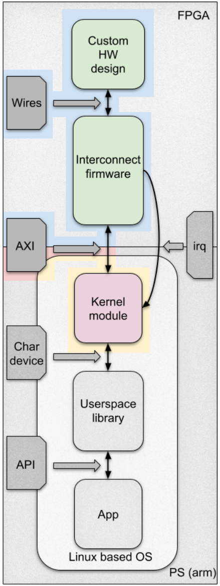
Kernel to firmware

Once the kernel has correctly decoded the data from the char device, it can directly write on AXI registers.



AXI registers are directly written by the kernel

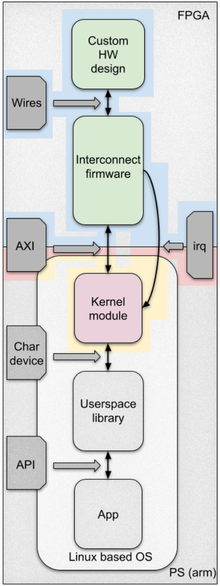
AXI guarantees consistency and transfer to the firmware input ports. Moreover the data flow from kernel cannot saturate the PL part.



Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

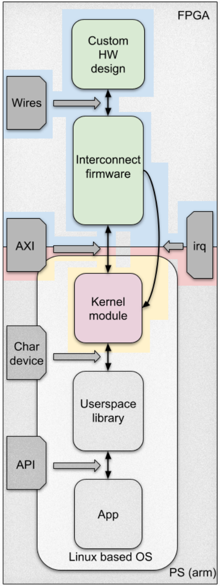


Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Stop accepting new changes from the IP



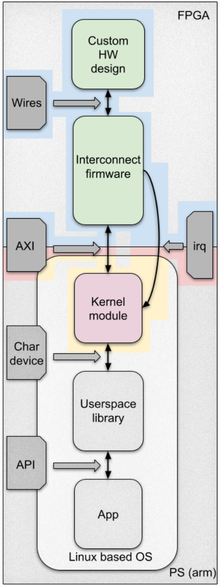
Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Send an interrupt request to the kernel

Stop accepting new changes from the IP



Firmware to kernel: IRQ

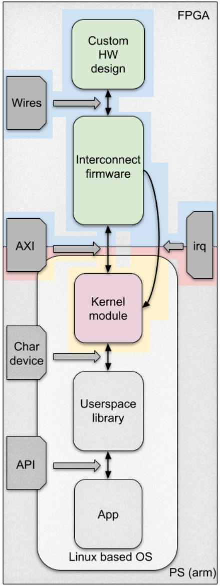
Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Send an interrupt request to the kernel

```c
// Interrupt Request number
#define IRQ_NO 46

// Interrupt handler for BM IRQ 46.
static irqreturn_t irq_handler(int irq, void *dev_id)
{
        struct work_data *irqwork;
        if (bmacc_state == stateCONNECT)
        {
                irqwork = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                INIT_WORK(&irqwork->work, work_handler);
                irqwork->cs = 3;
                queue_work(wq, &irqwork->work);
        }
        return IRQ_HANDLED;
}
```

Stop accepting new changes from the IP



Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

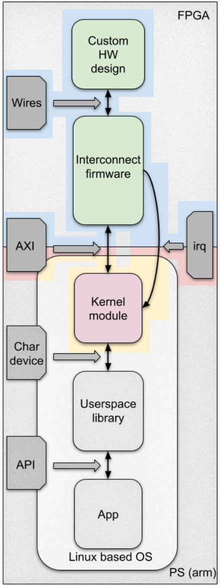
The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Stop accepting new changes from the IP

Send an interrupt request to the kernel

```c
// Interrupt Request number
#define IRQ_NO 46

// Interrupt handler for BM IRQ 46.
static irqreturn_t irq_handler(int irq, void *dev_id)
{
        struct work_data *irqwork;
        if (bmacc_state == stateCONNECT)
        {
                irqwork = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                INIT_WORK(&irqwork->work, work_handler);
                irqwork->cs = 3;
                queue_work(wq, &irqwork->work);
        }
        return IRQ_HANDLED;
}
```

The kernel get the IRQ, read the list of changes and send each of they through the char dev



Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

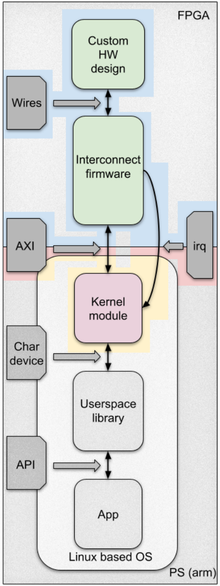
Send an interrupt request to the kernel

```c
// Interrupt Request number
#define IRQ_NO 46

// Interrupt handler for BM IRQ 46.
static irqreturn_t irq_handler(int irq, void *dev_id)
{
        struct work_data *irqwork;
        if (bmacc_state == stateCONNECT)
        {
                irqwork = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                INIT_WORK(&irqwork->work, work_handler);
                irqwork->cs = 3;
                queue_work(wq, &irqwork->work);
        }
        return IRQ_HANDLED;
}
```

The kernel get the IRQ, read the list of changes and send each of they through the char dev

The kernel notify the firmware when done

Stop accepting new changes from the IP



Library

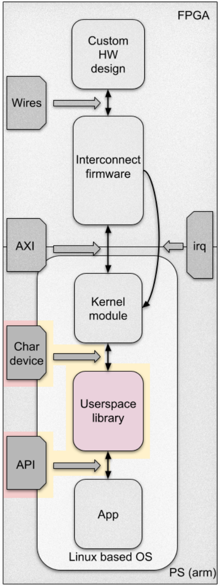
The char device created by the kernel is opened by the BMAPI user space library that implements the BMMRP.

/dev/bm

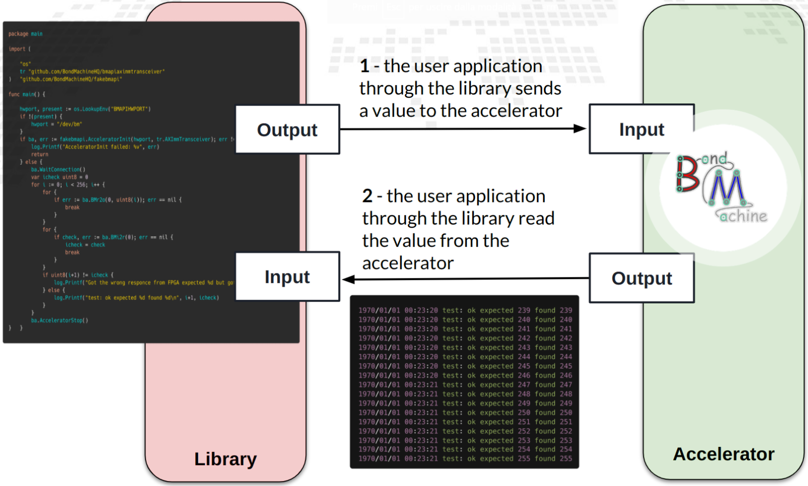
BMAPI Library

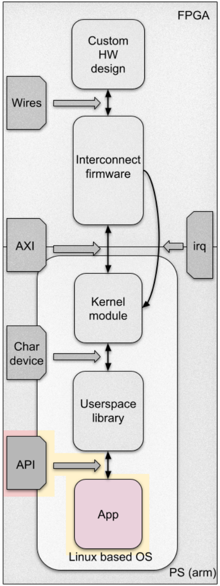
- (*BMAPI) BMr2owa
- (*BMAPI) BMr2ow
- (*BMAPI) BMr2o
- (*BMAPI) BMi2rw
- (*BMAPI) BMi2r

The library functions can be used by the application

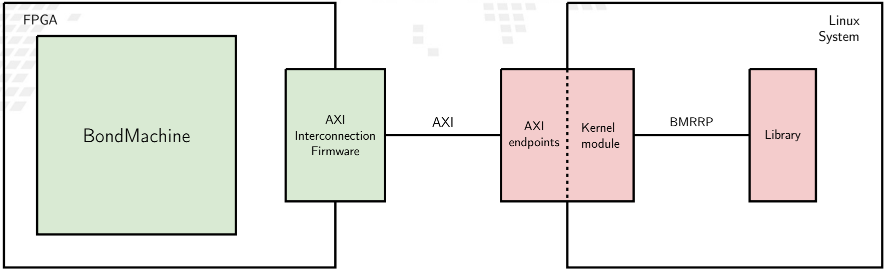


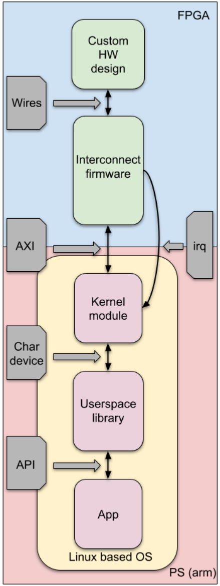
Accelerated application: an example





Accelerated Application





Tests and Benchmarks

An example

- Definition of an example
- Check of the correctness of the accelerator results
- Benchmark of the execution

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \left[c_i\right]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \left[c_i\right]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$


```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = [c_i]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$


```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

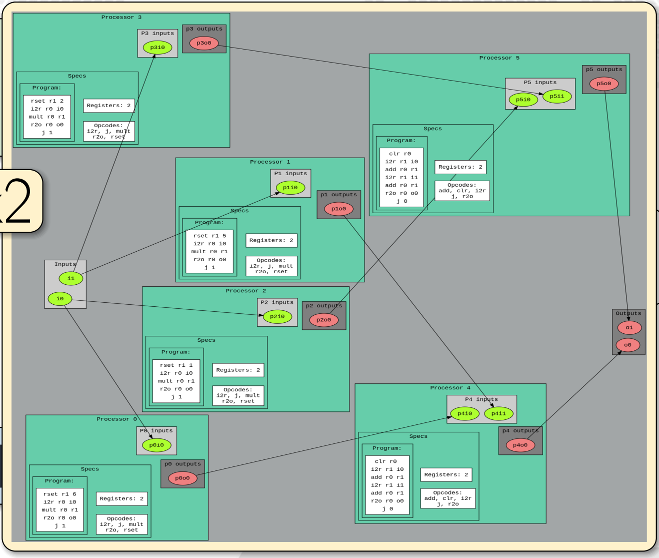

```
matrixwork -constants constants.json -constant-matrix A -numerical-type uint8 ...
```

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{bmatrix}$$

2x2

matrixwork

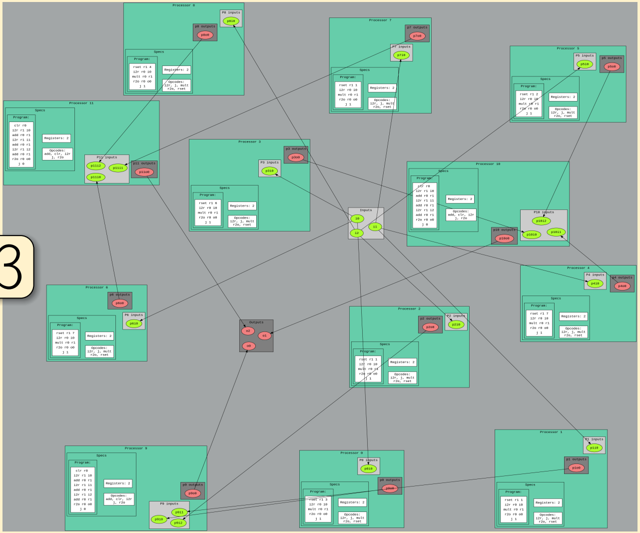



```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{bmatrix}$$

3x3

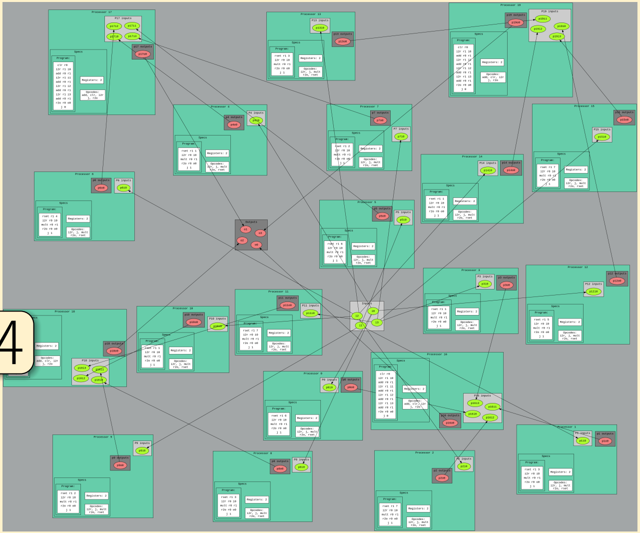


```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

matrixwork

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{bmatrix}$$



4x4

matrixwork

```
"A": [
    [6,5],
    [1,2]
],
"B": [
    [3,1,1],
    [6,7,2],
    [7,1,4]
],
"C": [
    [6,3,7,1],
    [1,6,4,2],
    [3,2,1,7],
    [5,3,1,7]
],
```

Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory

```
# ./monitor -g 0x43c00000 -n 8
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL:  00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0:     00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1:     00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2:     00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3:     00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4:     00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11100010 (0x43c0003c)
o6:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11110100 (0x43c00040)
o7:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench:  00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS:  00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory
- write directly to AXI memory mapped input addresses (through devmem)

```
# ./monitor -g 0x43c00000 -n 8
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL:  00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0:     00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1:     00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2:     00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3:     00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4:     00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11100010 (0x43c0003c)
o6:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11110100 (0x43c00040)
o7:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench:  00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS:  00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

```
devmem 0x43c00000 b 1
```

Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory

```
# ./monitor -g 0x43c00000 -n 8
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL:  00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0:     00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1:     00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2:     00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3:     00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4:     00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11100010 (0x43c0003c)
o6:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11110100 (0x43c00040)
o7:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench:  00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS:  00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

- write directly to AXI memory mapped input addresses (through devmem)

```
devmem 0x43c00000 b 1
```

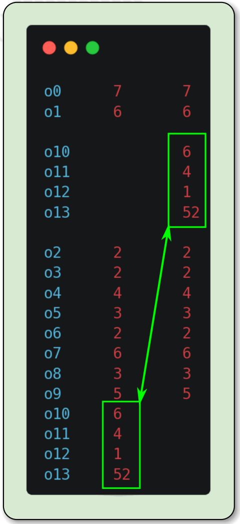
- check the AXI memory mapped output addresses

An example of error

```
#  ./monitor -g 0x43c00000 -n 13
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 00000001 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
i8:     00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
i9:     00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
i10:    00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 00000000 (0x43c00028)
i11:    00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 00000000 (0x43c0002c)
i12:    00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 00000000 (0x43c00030)
PS2PL:  00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 00000000 (0x43c00034)
STATES: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 00000000 (0x43c00038)
o0:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 00000111 (0x43c0003c)
o1:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 00000110 (0x43c00040)
o2:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 00000110 (0x43c00044)
o3:     00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00000100 (0x43c00048)
o4:     00000000 (0x43c0004f) 00000000 (0x43c0004e) 00000000 (0x43c0004d) 00000001 (0x43c0004c)
o5:     00000000 (0x43c00053) 00000000 (0x43c00052) 00000000 (0x43c00051) 00110100 (0x43c00050)
o6:     00000000 (0x43c00057) 00000000 (0x43c00056) 00000000 (0x43c00055) 00000010 (0x43c00054)
o7:     00000000 (0x43c0005b) 00000000 (0x43c0005a) 00000000 (0x43c00059) 00000010 (0x43c00058)
o8:     00000000 (0x43c0005f) 00000000 (0x43c0005e) 00000000 (0x43c0005d) 00000100 (0x43c0005c)
o9:     00000000 (0x43c00063) 00000000 (0x43c00062) 00000000 (0x43c00061) 00000011 (0x43c00060)
o10:    00000000 (0x43c00067) 00000000 (0x43c00066) 00000000 (0x43c00065) 00000010 (0x43c00064)
o11:    00000000 (0x43c0006b) 00000000 (0x43c0006a) 00000000 (0x43c00069) 00000110 (0x43c00068)
o12:    00000000 (0x43c0006f) 00000000 (0x43c0006e) 00000000 (0x43c0006d) 00000011 (0x43c0006c)
o13 bcm 00000000 (0x43c00073) 00000000 (0x43c00072) 00000000 (0x43c00071) 00000101 (0x43c00070)
PL2PS:  00000000 (0x43c00077) 00000111 (0x43c00076) 11111111 (0x43c00075) 11100000 (0x43c00074)
CHANGE: 00000000 (0x43c0007b) 00000111 (0x43c0007a) 11111111 (0x43c00079) 11111111 (0x43c00078)
```

An example of error

```
#  ./monitor -g 0x43c00000 -n 13
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 00000001 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
i8:     00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
i9:     00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
i10:    00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 00000000 (0x43c00028)
i11:    00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 00000000 (0x43c0002c)
i12:    00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 00000000 (0x43c00030)
PS2PL:  00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 00000000 (0x43c00034)
STATES: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 00000000 (0x43c00038)
o0:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 00000111 (0x43c0003c)
o1:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 00000110 (0x43c00040)
o2:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 00000110 (0x43c00044)
o3:     00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00000100 (0x43c00048)
o4:     00000000 (0x43c0004f) 00000000 (0x43c0004e) 00000000 (0x43c0004d) 00000001 (0x43c0004c)
o5:     00000000 (0x43c00053) 00000000 (0x43c00052) 00000000 (0x43c00051) 00110100 (0x43c00050)
o6:     00000000 (0x43c00057) 00000000 (0x43c00056) 00000000 (0x43c00055) 00000010 (0x43c00054)
o7:     00000000 (0x43c0005b) 00000000 (0x43c0005a) 00000000 (0x43c00059) 00000010 (0x43c00058)
o8:     00000000 (0x43c0005f) 00000000 (0x43c0005e) 00000000 (0x43c0005d) 00000100 (0x43c0005c)
o9:     00000000 (0x43c00063) 00000000 (0x43c00062) 00000000 (0x43c00061) 00000011 (0x43c00060)
o10:    00000000 (0x43c00067) 00000000 (0x43c00066) 00000000 (0x43c00065) 00000010 (0x43c00064)
o11:    00000000 (0x43c0006b) 00000000 (0x43c0006a) 00000000 (0x43c00069) 00000110 (0x43c00068)
o12:    00000000 (0x43c0006f) 00000000 (0x43c0006e) 00000000 (0x43c0006d) 00000011 (0x43c0006c)
o13 bcm 00000000 (0x43c00073) 00000000 (0x43c00072) 00000000 (0x43c00071) 00000101 (0x43c00070)
PL2PS:  00000000 (0x43c00077) 00000111 (0x43c00076) 11111111 (0x43c00075) 11100000 (0x43c00074)
CHANGE: 00000000 (0x43c0007b) 00000111 (0x43c0007a) 11111111 (0x43c00079) 11111111 (0x43c00078)
```



Benchmark: caveats

This is a preliminary work.

We trust some tools:

- Vivado reports
- perf

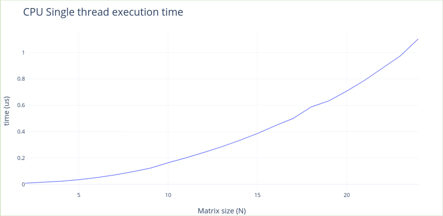
The FPGA benchmarks do not include the PS part overhead (the comparisons are not really fair)

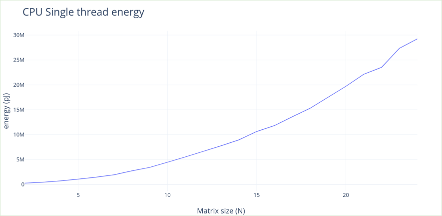
Benchmark: the CPU (Golang)

```go
func matrixtest(n int, iter int64) float32 {
//...
    start := time.Now()

    for k := 0; int64(k) < iter; k++ {
        for i := 0; i < n; i++ {
            output[i] = uint8(0)
        }

        for i := 0; i < n; i++ {
            for j := 0; j < n; j++ {
                output[i] += input[j] * matrix[i+j*n]
            }
        }
    }
    return float32(time.Since(start).Microseconds()) / float32(iter)
}
func main() {
    for i := 2; i <= 32; i++ {
        fmt.Println(i, ",", matrixtest(i, 100000000))
    }
}
```

- Time measures: built-in golang facilities
- Energy measures: perf
- Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz
- Go 1.18.2

	N	single op time (us)	single op energy (pJ)	Energy eff
1	2	0.00940269	259200	3.85802E-06
2	3	0.01631968	454200	2.20167E-06
3	4	0.02399984	722200	1.38466E-06
4	5	0.03632906	1070400	9.3423E-07
5	6	0.05176583	1471400	6.79625E-07
6	7	0.07183811	1935800	5.16582E-07
7	8	0.09597738	2737000	3.65364E-07
8	9	0.12337612	3429200	2.91613E-07
9	10	0.16450378	4465900	2.23919E-07
10	11	0.20173332	5530300	1.80822E-07
11	12	0.24265632	6643300	1.50528E-07
12	13	0.28554472	7762600	1.28823E-07
13	14	0.33400026	8954000	1.11682E-07
14	15	0.3861176	10633500	9.40424E-08
15	16	0.44550508	11832200	8.45151E-08
16	17	0.5004054	13604700	7.3504E-08
17	18	0.5867083	15324500	6.5255E-08
18	19	0.63376665	17524400	5.70633E-08
19	20	0.708154	19716100	5.072E-08
20	21	0.7893206	22133800	4.51798E-08
21	22	0.8830365	23525100	4.25078E-08
22	23	0.97467226	27346700	3.65675E-08
23	24	1.1031791	29235100	3.42055E-08

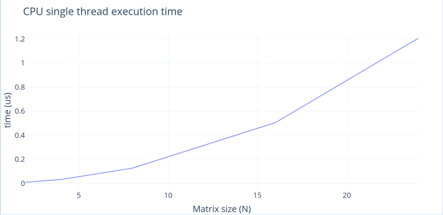


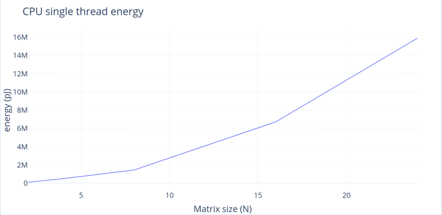


Benchmark: the CPU (C)

- Time measures: time
- Energy measures: perf
- Intel(R) CPU I5-8500 v5 @ 3GHz
- gcc with -O0

	n	single op energy (pJ)	single op time (us)	energy eff
1	2	100000	0.01	0.000008333333333
2	4	500000	0.033	0.000002702702703
3	8	1450000	0.127	0.0000005524861878
4	16	6720000	0.505	0.0000001326259947
5	24	15880000	1.205	0.00000006854009596



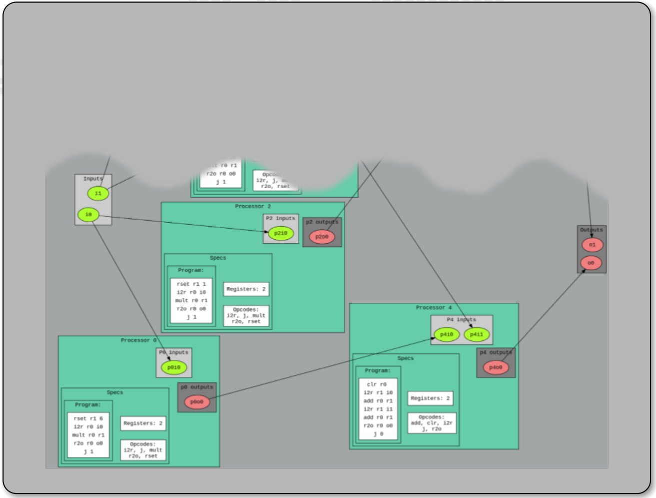


Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

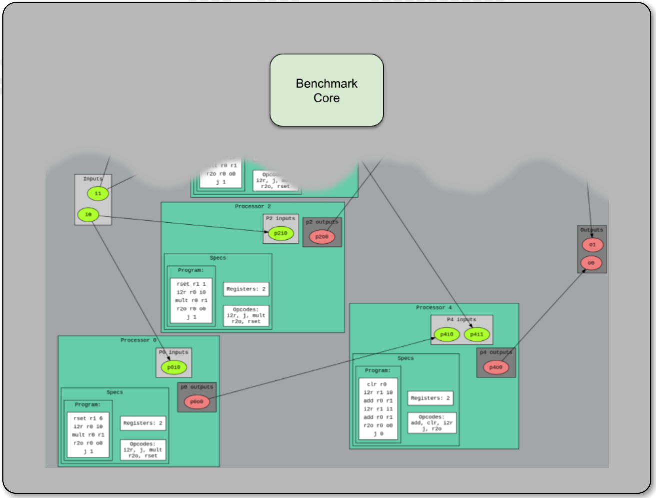


Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

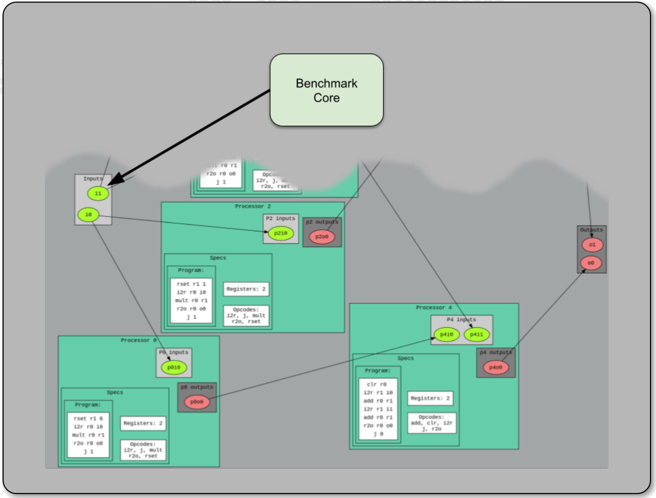


Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

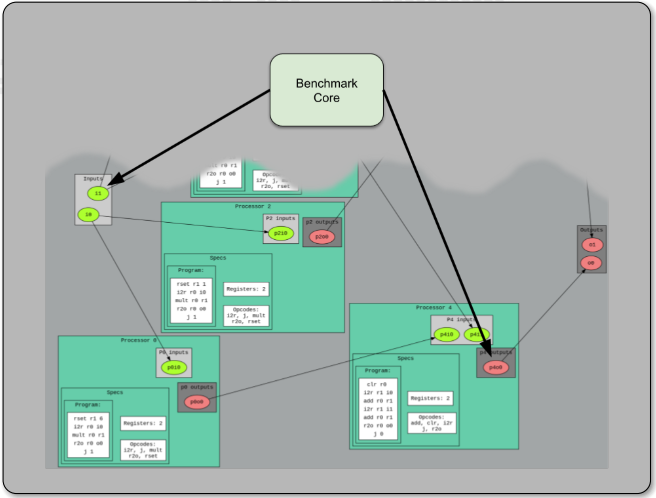


Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

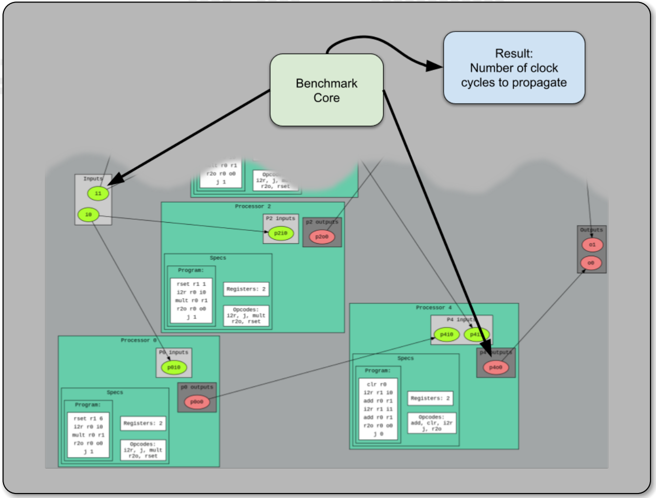


Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

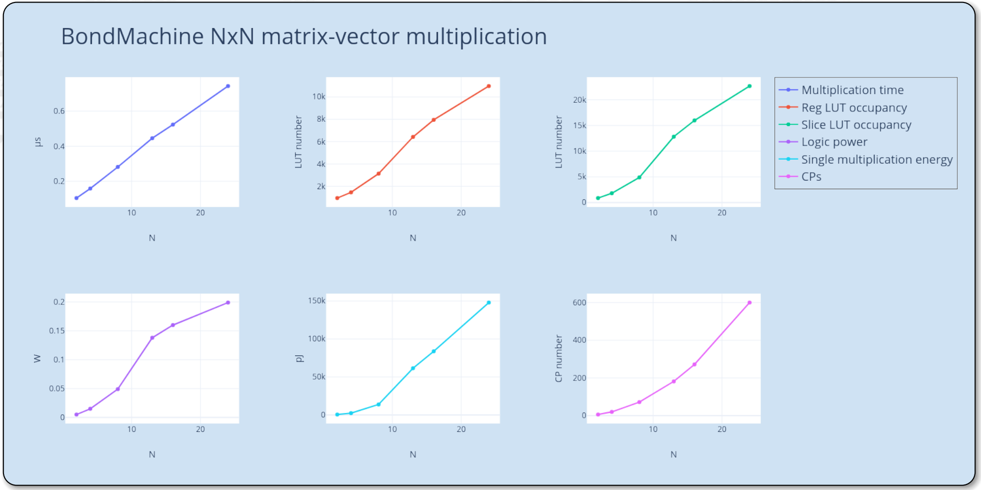


Benchmark core clock cycles distributions

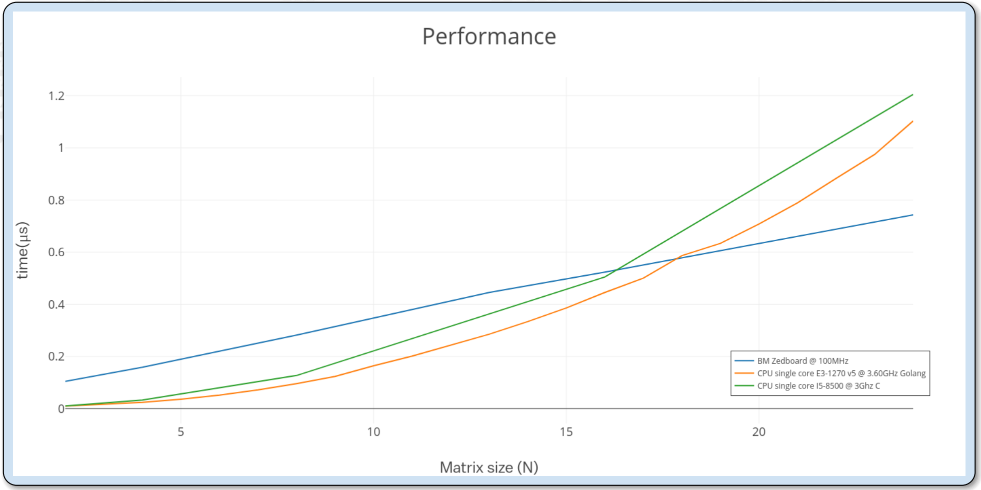
FPGA benchmark summary

	N	single op time (us)	Register LUTs	Slice LUTs	Power	single op energy (pJ)	CPs
1	2	0.1044	947	875	0.005	522	6
2	4	0.1587	1457	1813	0.015	2380.5	20
3	8	0.2819	3131	4897	0.049	13813.1	72
4	13	0.4456	6422	12819	0.138	61492.8	182
5	16	0.5234	7950	15979	0.160	83744	272
6	24	0.7432	10974	22669	0.199	147896.8	600

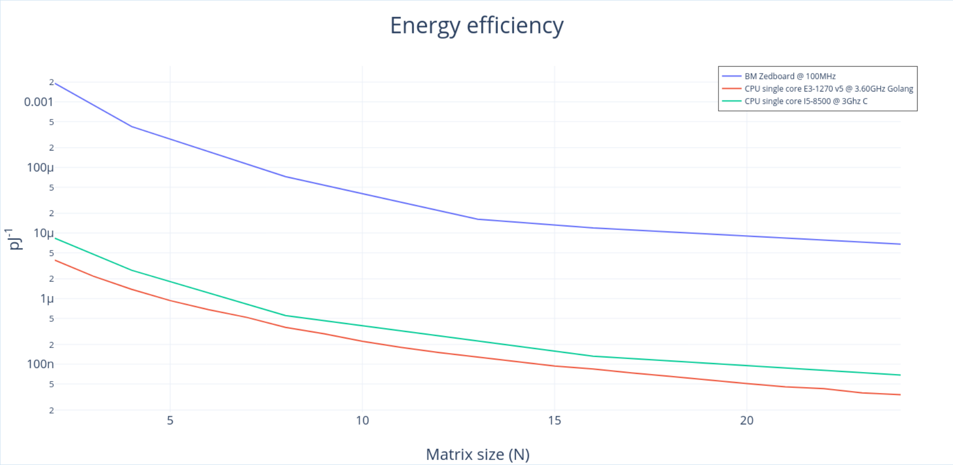
Benchmark core



Comparisons: Performace



Comparisons: Energy



Conclusions and Future directions

Conclusions

- The creation of a firmware from ground up is not a mere exercise. It gives perspective on how heterogeneous system really works and what really is an FPGA accelerator
- Even if the methodology and the tools were specifically created for the BondMachine project, the are sufficiently general to be appliable to other FPGA accelerators as well
- FPGA is a groundbreaking technology but require a change of perspective in how we develop software

Future directions

We plan to extend the benchmarks to:

- different data types
- different boards
- compare with GPUs
- include some real power consumption measures

For the project:

- First DAQ use case
- Complete the inclusion of Intel and Lattice FPGAs and try a more performant Zynq based board
- Accelerator in a cloud workflow

Thank you

website: `http://bondmachine.fisica.unipg.it`
code: `https://github.com/BondMachineHQ`
parallel computing paper: link
contact email: mirko.mariotti@unipg.it