THALES

Fast iteration and prototyping in high-performance computing medical applications: a case study with Mentor Vista

8th INFIERI WORKSHOP 10/21/2016

Alberto Garcia-Fernandez

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement No. [317446] INFIERI "INtelligent Fast Interconnected and Efficient Devices for Frontier Exploitation in Research and Industry".

Background

Thales R&T HPC lab topics.

High-performance computing architectures.

- Finding the right architecture for a given application domain.

Tooled-up flows for high-performance cyber-physical systems.

- Efficiently programming adaptive architectures.

Mission.

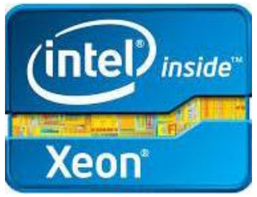
- Evaluate available vendor tools.
- Show the gap between our needs and the tools capabilities.
- Propose, develop and evaluate solutions to bridge this gap

THALES

Objectives

1. Use of tools for fast exploration on new heterogeneous architectures for high performance computing applications

- Fast performance evaluation of new chips and platforms.
- Architecture-adequacy discovery for domain representative algorithms.
  - Image processing.
  - Pixel tracking.
  - Big data.



2. Architecture design and prototyping on low-power image processing platforms

- Improve performance per watt: peak by 4x and average by 10x by the end of 2018.
  - 100x and 200x by 2023.
- Medical X-Ray Imaging.
- Reduce the level of radiation by 75%.
  - More powerful image processing.
  - The device needs to be small and mobile.
  - Processing close to the sensor to help reduce extraneous wires.

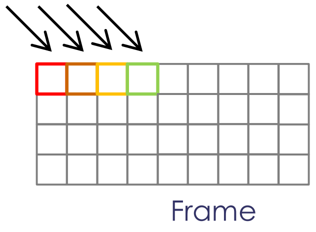
Case study

Image processing kernel chain

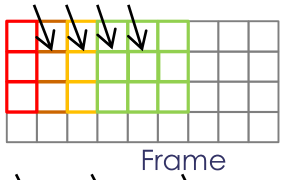
- Reference implementation in sequential C code.
- Requirements.
  - **Must:** 1024x1024 , 25 fps 420 Mbits/sec
  - **Want:** 1344x1344, 30 fps 870 Mbits/sec

Different data access patterns.

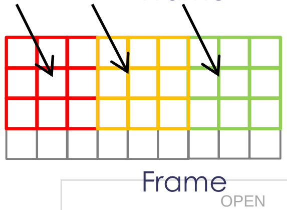
- Pixel by pixel.
  - Good spatial locality.
  - Poor temporal locality.



- Sliding window.
  - Poor spatial locality.
  - Good temporal locality.



- Chessboard.
  - Poor spatial locality.
  - Poor temporal locality.



Tools available

- HW/SW co-designers.
  - Integration of compiler and synthesizer.
  - Reduced set of architectures for exploration.
- Virtual prototyping.
  - HW simulators or emulators.
  - Design the software and test it on a VP.
- SW parallelizers.
  - Adapt SW for heterogeneous devices.
  - OpenMP, OpenCL, CUDA.
  - No custom custom HW (i.e. FPGA).

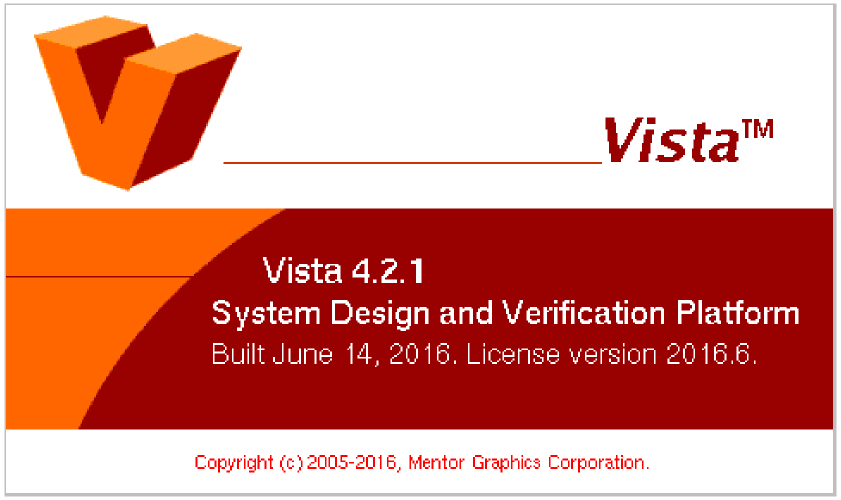
Mentor Vista and Sourcery Codebench

Mentor Vista

- SystemC TLM integrated development environment.
- Architecture design exploration.
- Virtual prototyping.
- Model the hardware before register-transfer level is available.

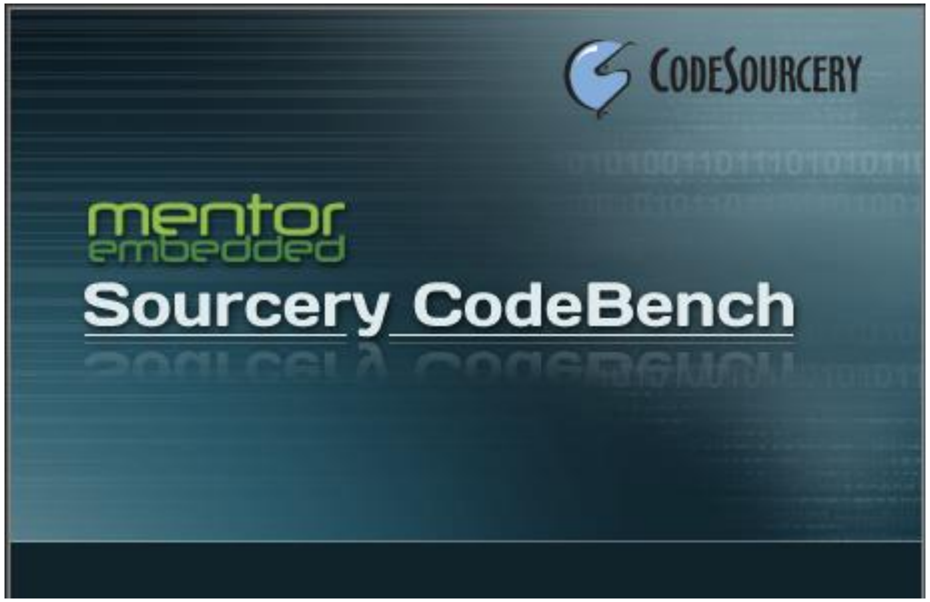
Sourcery Codebench

- C/C++ integrated development environment.
- Development for embedded devices.
  - ARM, IA32, MIPS, Power.
- Possibility to link with Mentor Vista.
  - Develop the SW before having the HW.



Vista 4.2.1
System Design and Verification Platform
Built June 14, 2016. License version 2016.6.

Copyright (c) 2005-2016, Mentor Graphics Corporation.



Development in three use cases

Model the hardware.

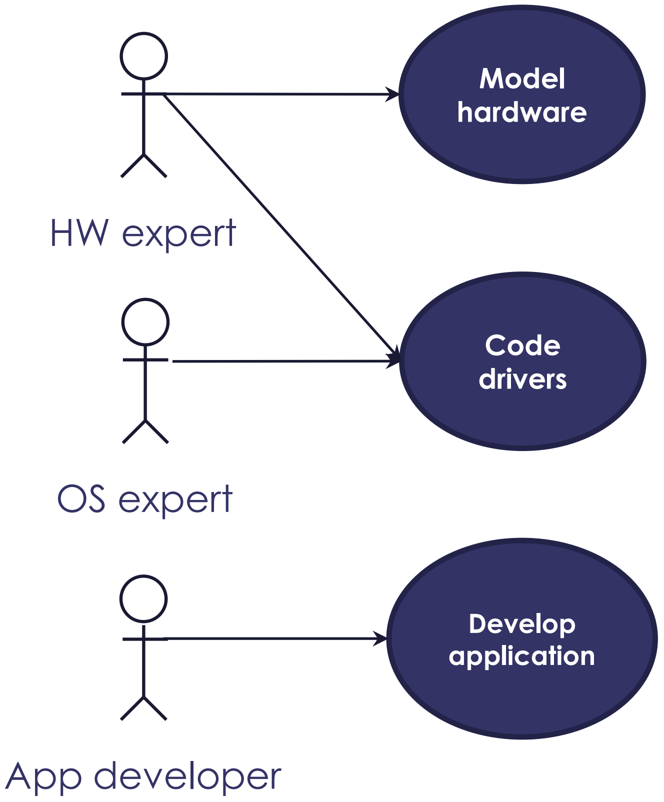
- Model IPs within FPGA fabric.
- Provide a bus mapping.
- Hardware knowledge required.

Code the kernel drivers.

- Implement the drivers of the fabric IPs in the Linux device tree.
- Hardware and OS knowledge required.

Develop the application.

- Adapt the reference application to the model of computation required.
- Offload computation to the FPGA IPs.
- Deploy the application in the prototype.
- Software development knowledge required.



Modeling the hardware

SystemC and TLM for the hardware IP models.

Easy and straightforward to develop.

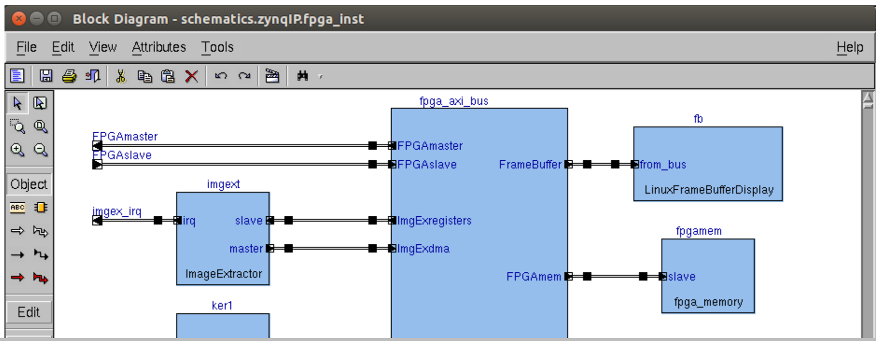
- Accuracy depending on the level of detail.

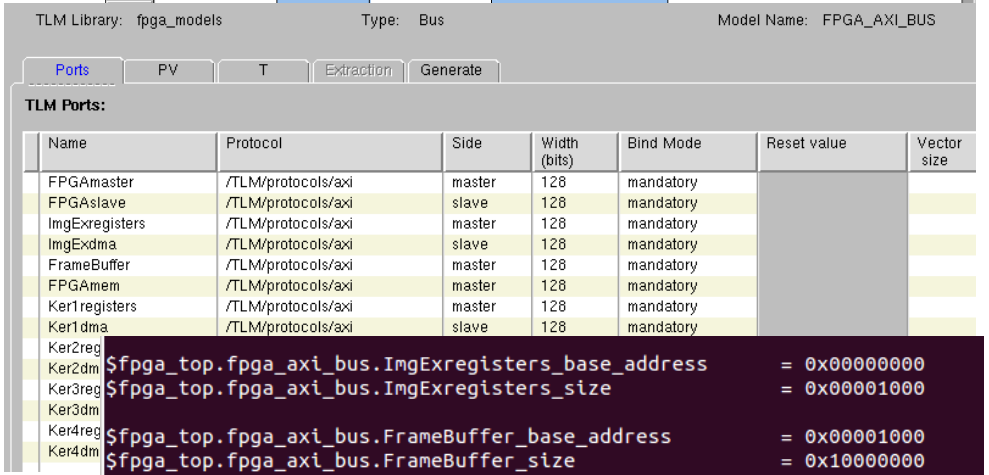
Vista for the IP modeling:

- HW registers,
- Memory addresses.
- IRQs.
- Energy consumptions.
- Physical bus connections.

Define the bus mapping.

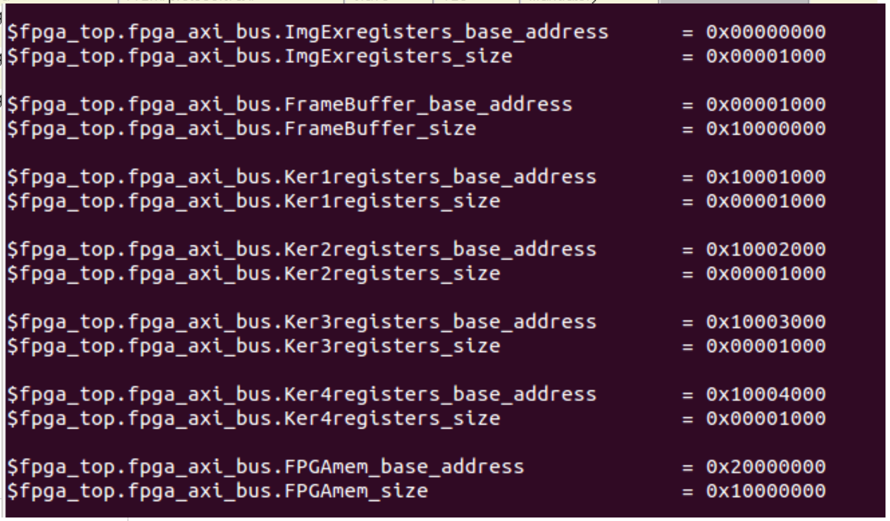
- Design the memory layout of the FPGA.




TLM Library: fpga_models Type: Bus Model Name: FPGA_AXI_BUS

TLM Ports:

Name	Protocol	Side	Width (bits)	Bind Mode	Reset value	Vector size
FPGAmaster	/TLM/protocols/axi	master	128	mandatory		
FPGAslave	/TLM/protocols/axi	slave	128	mandatory		
ImgExregisters	/TLM/protocols/axi	master	128	mandatory		
ImgExdma	/TLM/protocols/axi	slave	128	mandatory		
FrameBuffer	/TLM/protocols/axi	master	128	mandatory		
FPGAmem	/TLM/protocols/axi	master	128	mandatory		
Ker1registers	/TLM/protocols/axi	master	128	mandatory		
Ker1dma	/TLM/protocols/axi	slave	128	mandatory		
Ker2reg						
Ker2dm						
Ker3reg						
Ker3dm						
Ker4reg						
Ker4dm						


```
$fpga_top.fpga_axi_bus.ImgExregisters_base_address      = 0x00000000
$fpga_top.fpga_axi_bus.ImgExregisters_size              = 0x00001000

$fpga_top.fpga_axi_bus.FrameBuffer_base_address         = 0x00001000
$fpga_top.fpga_axi_bus.FrameBuffer_size                 = 0x10000000

$fpga_top.fpga_axi_bus.Ker1registers_base_address       = 0x10001000
$fpga_top.fpga_axi_bus.Ker1registers_size               = 0x00001000

$fpga_top.fpga_axi_bus.Ker2registers_base_address       = 0x10002000
$fpga_top.fpga_axi_bus.Ker2registers_size               = 0x00001000

$fpga_top.fpga_axi_bus.Ker3registers_base_address       = 0x10003000
$fpga_top.fpga_axi_bus.Ker3registers_size               = 0x00001000

$fpga_top.fpga_axi_bus.Ker4registers_base_address       = 0x10004000
$fpga_top.fpga_axi_bus.Ker4registers_size               = 0x00001000

$fpga_top.fpga_axi_bus.FPGAmem_base_address             = 0x20000000
$fpga_top.fpga_axi_bus.FPGAmem_size                     = 0x10000000
```

Coding the kernel drivers

UIO-based approach used to provide the interface to the FPGA fabric IPs.

- Gap between HW modelling and SW application development.

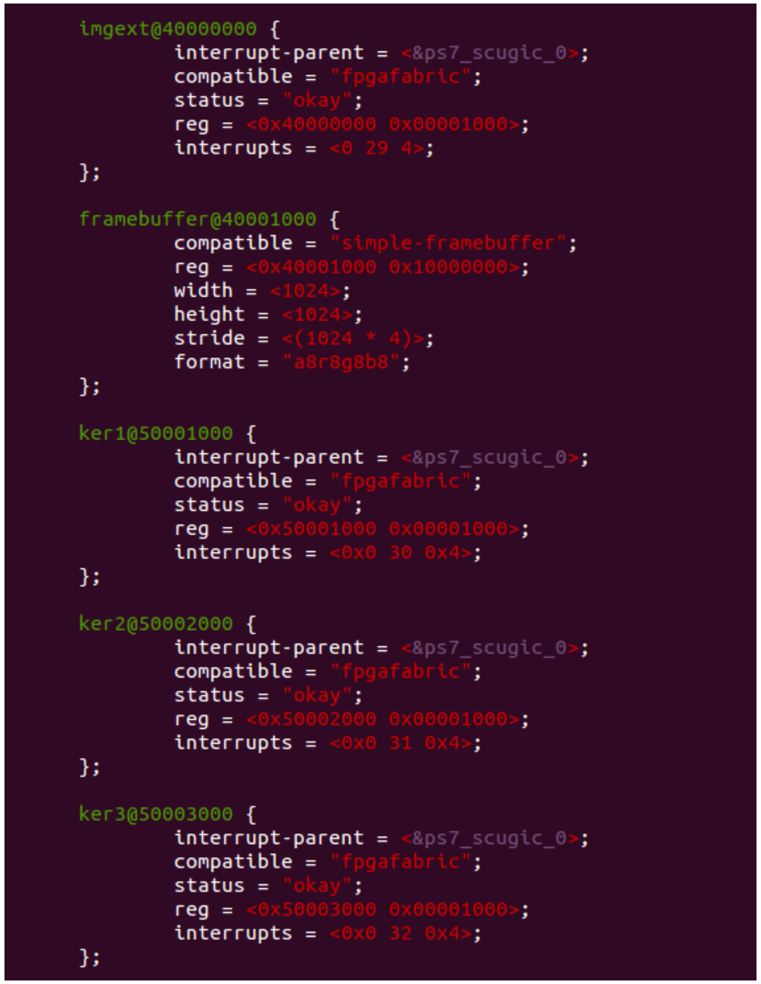
Modify the linux device tree and rebuild the root.

- Not covered by the tools.
- Not a trivial task.
- Add one node per IP.

Forward the memory, bus, and IRQ mapping defined in the previous stage.

- Calculate corresponding memory addresses and IRQs.
- Documentation provided by Mentor.

Possible UIO driver modification required.



```
imgext@40000000 {
        interrupt-parent = <&ps7_scugic_0>;
        compatible = "fpgafabric";
        status = "okay";
        reg = <0x40000000 0x00001000>;
        interrupts = <0 29 4>;
};

framebuffer@40001000 {
        compatible = "simple-framebuffer";
        reg = <0x40001000 0x10000000>;
        width = <1024>;
        height = <1024>;
        stride = <(1024 * 4)>;
        format = "a8r8g8b8";
};

ker1@50001000 {
        interrupt-parent = <&ps7_scugic_0>;
        compatible = "fpgafabric";
        status = "okay";
        reg = <0x50001000 0x00001000>;
        interrupts = <0x0 30 0x4>;
};

ker2@50002000 {
        interrupt-parent = <&ps7_scugic_0>;
        compatible = "fpgafabric";
        status = "okay";
        reg = <0x50002000 0x00001000>;
        interrupts = <0x0 31 0x4>;
};

ker3@50003000 {
        interrupt-parent = <&ps7_scugic_0>;
        compatible = "fpgafabric";
        status = "okay";
        reg = <0x50003000 0x00001000>;
        interrupts = <0x0 32 0x4>;
};
```

Developing the software

Compile and execute the reference code in the Virtual Prototype.

- Simple and easy.
- Only recompile dependent libraries (if any).

Low level programming in order to use the hardware modeled.

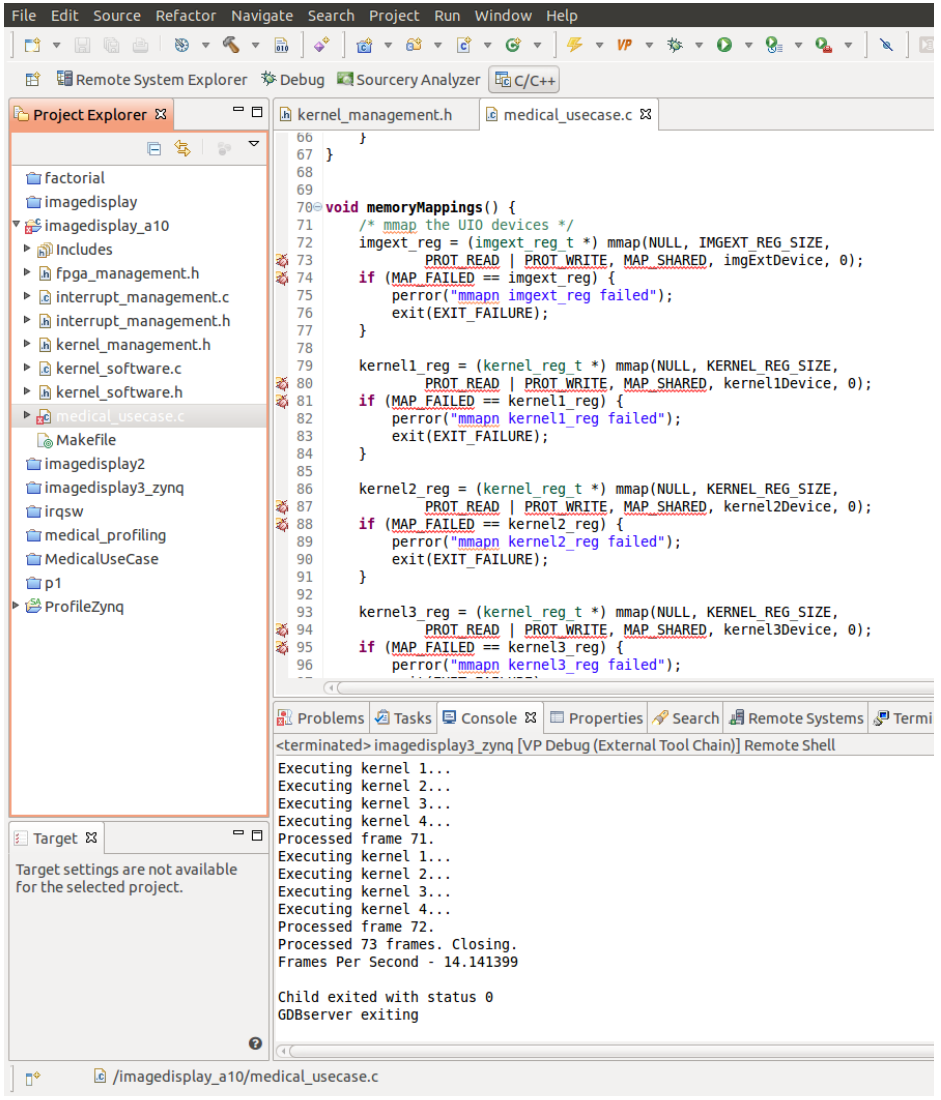
- Easy once examples are provided.

Most of the time invested adapting the reference code (sequential) to other models of computation.

- Finite state machines.
- Concurrent execution.
- Pipeline.

Mentor has several projects using GStreamer.

- Not checked in this case study, but interesting for the future.



```
    }
}


void memoryMappings() {
    /* mmap the UIO devices */
    imgext_reg = (imgext_reg_t *) mmap(NULL, IMGEXT_REG_SIZE,
            PROT_READ | PROT_WRITE, MAP_SHARED, imgExtDevice, 0);
    if (MAP_FAILED == imgext_reg) {
        perror("mmapn imgext_reg failed");
        exit(EXIT_FAILURE);
    }

    kernel1_reg = (kernel_reg_t *) mmap(NULL, KERNEL_REG_SIZE,
            PROT_READ | PROT_WRITE, MAP_SHARED, kernel1Device, 0);
    if (MAP_FAILED == kernel1_reg) {
        perror("mmapn kernel1_reg failed");
        exit(EXIT_FAILURE);
    }

    kernel2_reg = (kernel_reg_t *) mmap(NULL, KERNEL_REG_SIZE,
            PROT_READ | PROT_WRITE, MAP_SHARED, kernel2Device, 0);
    if (MAP_FAILED == kernel2_reg) {
        perror("mmapn kernel2_reg failed");
        exit(EXIT_FAILURE);
    }

    kernel3_reg = (kernel_reg_t *) mmap(NULL, KERNEL_REG_SIZE,
            PROT_READ | PROT_WRITE, MAP_SHARED, kernel3Device, 0);
    if (MAP_FAILED == kernel3_reg) {
        perror("mmapn kernel3_reg failed");
```

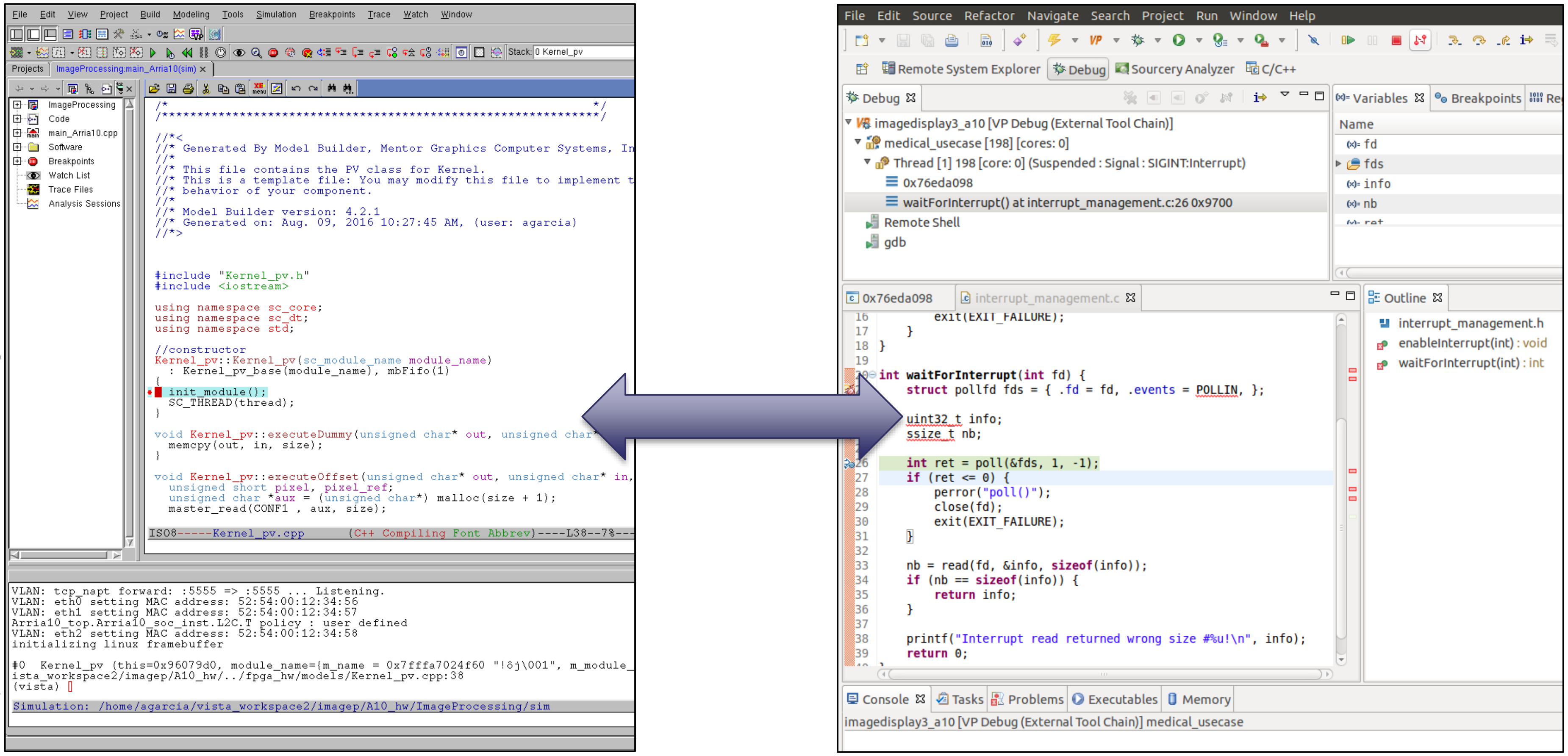
<terminated> imagedisplay3_zynq [VP Debug (External Tool Chain)] Remote Shell

```
Executing kernel 1...
Executing kernel 2...
Executing kernel 3...
Executing kernel 4...
Processed frame 71.
Executing kernel 1...
Executing kernel 2...
Executing kernel 3...
Executing kernel 4...
Processed frame 72.
Processed 73 frames. Closing.
Frames Per Second - 14.141399

Child exited with status 0
GDBserver exiting
```

Target settings are not available for the selected project.

HW/SW Co-debugging

- Dual approach: Vista for debugging the HW and Sourcery for the SW.
- Quite simple to perform co-debugging if some insights are provided.



```
/*                                                                  */
/********************************************************************/

//*<
//* Generated By Model Builder, Mentor Graphics Computer Systems, In
//*
//* This file contains the PV class for Kernel.
//* This is a template file: You may modify this file to implement t
//* behavior of your component.
//*
//* Model Builder version: 4.2.1
//* Generated on: Aug. 09, 2016 10:27:45 AM, (user: agarcia)
//*>


#include "Kernel_pv.h"
#include <iostream>

using namespace sc_core;
using namespace sc_dt;
using namespace std;

//constructor
Kernel_pv::Kernel_pv(sc_module_name module_name)
  : Kernel_pv_base(module_name), mbFifo(1)
{
  init_module();
  SC_THREAD(thread);
}

void Kernel_pv::executeDummy(unsigned char* out, unsigned char
  memcpy(out, in, size);
}

void Kernel_pv::executeOffset(unsigned char* out, unsigned char* in,
  unsigned short pixel, pixel_ref;
  unsigned char *aux = (unsigned char*) malloc(size + 1);
  master_read(CONF1 , aux, size);
ISO8-----Kernel_pv.cpp        (C++ Compiling Font Abbrev)----L38--7%---
```

```
VLAN: tcp_napt forward: :5555 => :5555 ... Listening.
VLAN: eth0 setting MAC address: 52:54:00:12:34:56
VLAN: eth1 setting MAC address: 52:54:00:12:34:57
Arria10_top.Arria10_soc_inst.L2C.T policy : user defined
VLAN: eth2 setting MAC address: 52:54:00:12:34:58
initializing linux framebuffer

#0  Kernel_pv (this=0x96079d0, module_name={m_name = 0x7fffa7024f60 "!ôj\001", m_module_
ista_workspace2/imagep/A10_hw/../fpga_hw/models/Kernel_pv.cpp:38
(vista) 

Simulation: /home/agarcia/vista_workspace2/imagep/A10_hw/ImageProcessing/sim
```

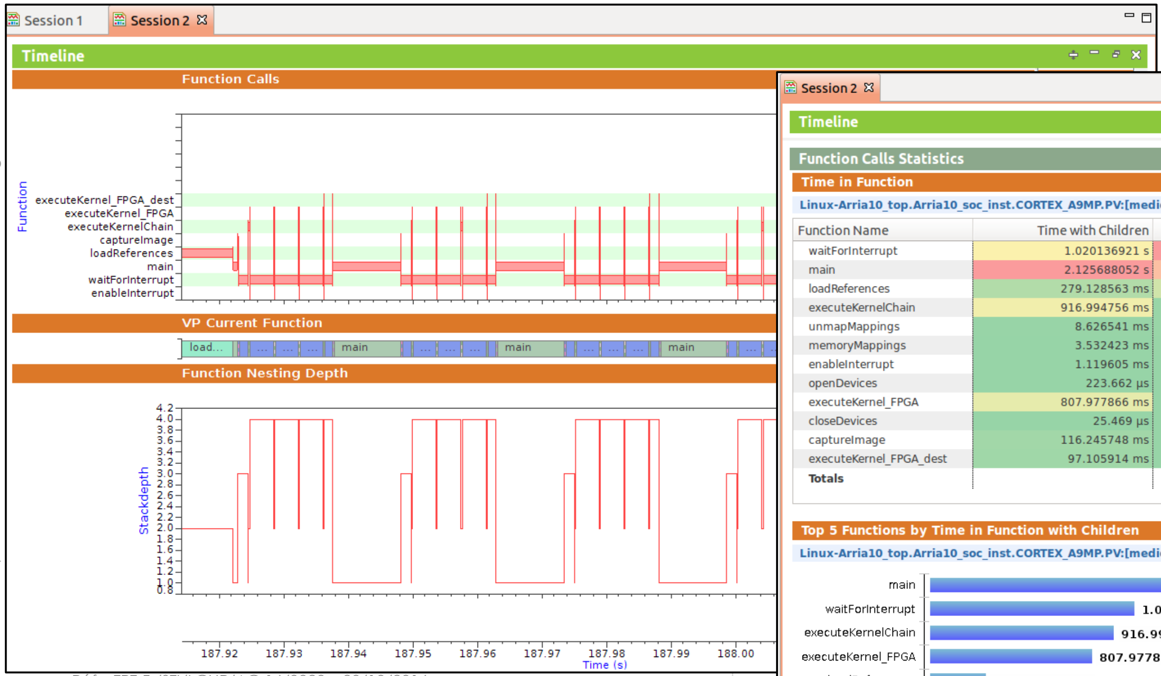
```
Debug
imagedisplay3_a10 [VP Debug (External Tool Chain)]
  medical_usecase [198] [cores: 0]
    Thread [1] 198 [core: 0] (Suspended : Signal : SIGINT:Interrupt)
      0x76eda098
      waitForInterrupt() at interrupt_management.c:26 0x9700
  Remote Shell
  gdb
```

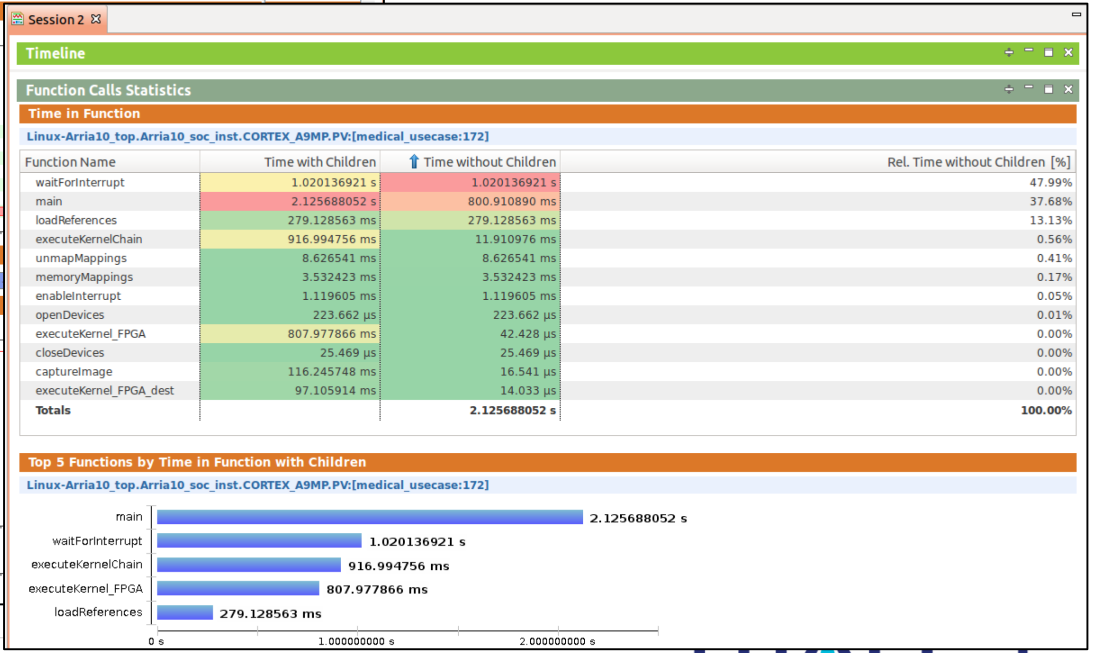
```
        exit(EXIT_FAILURE);
    }
}

int waitForInterrupt(int fd) {
    struct pollfd fds = { .fd = fd, .events = POLLIN, };

    uint32_t info;
    ssize_t nb;

    int ret = poll(&fds, 1, -1);
    if (ret <= 0) {
        perror("poll()");
        close(fd);
        exit(EXIT_FAILURE);
    }

    nb = read(fd, &info, sizeof(info));
    if (nb == sizeof(info)) {
        return info;
    }

    printf("Interrupt read returned wrong size #%u!\n", info);
    return 0;
```

```
imagedisplay3_a10 [VP Debug (External Tool Chain)] medical_usecase
```

Platform profiling

- **Entities and functions to be traced defined through a trace file.**
  - HW entities (models, buses, signals, CPUs, etc.).
  - SW entities (OS, functions, etc.).
- **Problem with very large dense traces or very long simulations.**
  - Actually it’s a problem common to all tracing tools.
- **Very good visualization of results and tracing analysis tools.**





Linux-Arria10_top.Arria10_soc_inst.CORTEX_A9MP.PV:[medical_usecase:172]

Function Name	Time with Children	Time without Children	Rel. Time without Children [%]
waitForInterrupt	1.020136921 s	1.020136921 s	47.99%
main	2.125688052 s	800.910890 ms	37.68%
loadReferences	279.128563 ms	279.128563 ms	13.13%
executeKernelChain	916.994756 ms	11.910976 ms	0.56%
unmapMappings	8.626541 ms	8.626541 ms	0.41%
memoryMappings	3.532423 ms	3.532423 ms	0.17%
enableInterrupt	1.119605 ms	1.119605 ms	0.05%
openDevices	223.662 µs	223.662 µs	0.01%
executeKernel_FPGA	807.977866 ms	42.428 µs	0.00%
closeDevices	25.469 µs	25.469 µs	0.00%
captureImage	116.245748 ms	16.541 µs	0.00%
executeKernel_FPGA_dest	97.105914 ms	14.033 µs	0.00%
Totals		**2.125688052 s**	**100.00%**

Top 5 Functions by Time in Function with Children

Linux-Arria10_top.Arria10_soc_inst.CORTEX_A9MP.PV:[medical_usecase:172]

Function	Time
main	2.125688052 s
waitForInterrupt	1.020136921 s
executeKernelChain	916.994756 ms
executeKernel_FPGA	807.977866 ms
loadReferences	279.128563 ms

Design space exploration

Migration from Arria10 to Zynq.

- HW FPGA models almost the same.
- SW application code totally portable.

Change the bus sizes of the FPGA HW models.

- Information provided by Mentor Vista's compiler.
- Change bus sizes of the fabric or use adapters.

Find the corresponding bus between FPGA fabric and SoC.

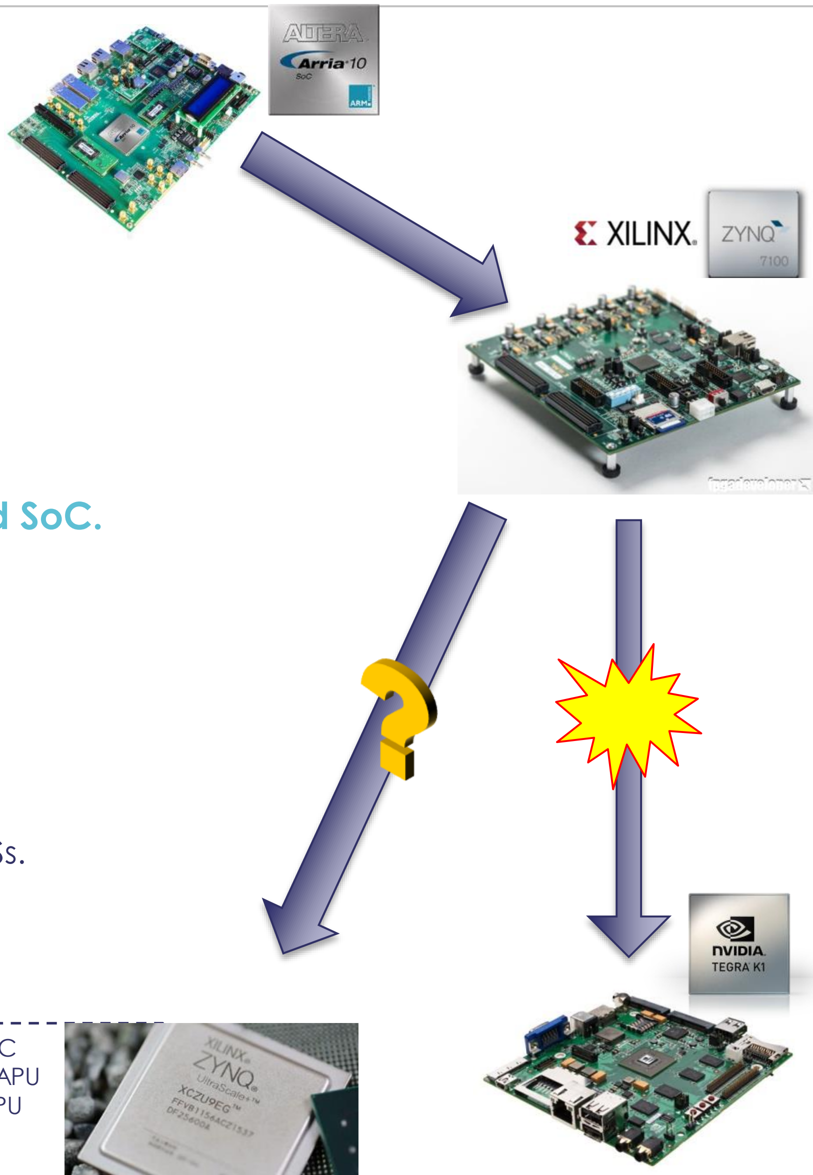
- Learn about the underlying architecture.
- Choose a new bus to connect.
- Recalculate memory mapping.

Redo the DTS file for the new platform.

- Requires deep understanding of the platforms and their DTSs.

Not very different devices (A10 vs Zynq).

- Both of them SoC.
- Waiting for the Zynq Ultrascale.
- Still untested real heterogeneity.
- GPGPUs unexplored.



Zynq Ultrascale+ MPSoC
- ARM Cortex-A53 APU
- ARM Cortex-R5 RPU
- Mali-400MP2 GPU
- H.264/H.265 VCU

THALES

Conclusions

Mentor Vista and Sourcery Codebench are very mature tools.

- Ready to be used in real developments.
- Still some level of expertise is required.

Intended to help the HW/SW co-integration using a virtual prototype.

- Not focused on fast iteration and early (real) prototyping.
- Best use case: software application on SoCs.
- No real heterogeneity (for now). No GPGPU, many-cores, or DSPs.

Fast prototyping on FPGAs: check out HW/SW co-designers.

- Reduced set of supported HW platforms.

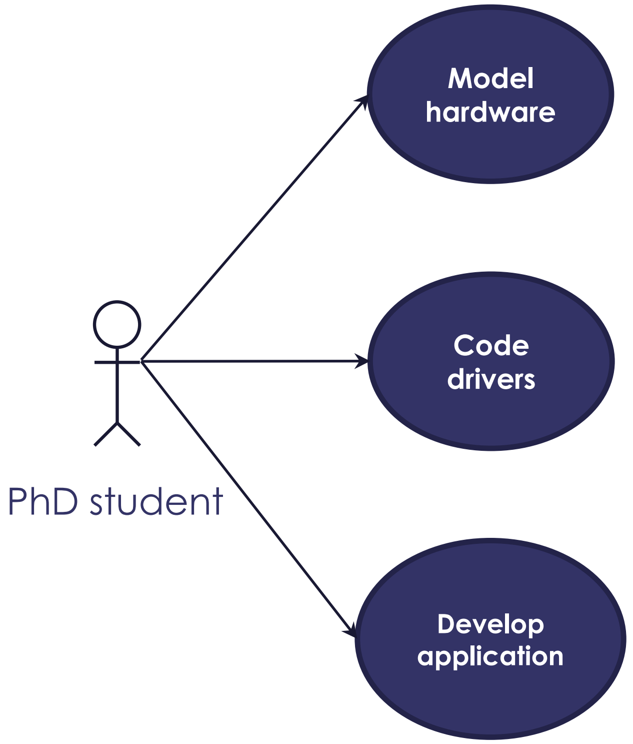
Explore real heterogeneity: check out SW parallelizers.

- High-level interfaces: OpenCL, OpenMP.
- HW design left aside. No FPGA (for now).

Future works

Share the expertise about the tool in order to speed-up the learning curve.

- Accelerate the transition between functional specification and architecture implementation.
- Reduce specific knowledge required.



Provide reference implementations ready to be used.

- Generic HW models.
  - Common kernels or data access patterns.
  - Information about timing and power consumption.
  - Bus mappings.

- Application templates implementing different models of computation.
  - Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell. *Pthreads Programming*. 1996. O'Reilly.
    - *Boss/Worker.*
    - *Peer model.*
    - *Pipeline model.*