PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA

PLATAFORMAS VIRTUAIS DE DESENVOLVIMENTO DE SOFTWARE

IAÇANÃ WEBER - FERNANDO GEHM MORAES

JUNHO/2023

Sumário

- Introdução
- Plataformas Virtuais de Desenvolvimento de SW
- OVP
- Conceitos
- Modelos
- API
- Ferramentas
- Exemplos

Introdução

Plataformas virtuais (VPs) são mais um método para simulação do seu sistema

Existem 2 tipos de VPs:

A) Voltadas para o desenvolvimento de HW

B) Voltadas para o desenvolvimento de SW

Introdução

A) Voltadas para o desenvolvimento de HW

- São direcionados ao desenvolvedor de software
- Complexas – programação de baixo nível de abstração
- Precisão de ciclo de clock
  - Simulação lenta, bem abaixo de 1 MIPS
- Plataforma de desenvolvimento de SW somente após a conclusão do projeto do HW

Introdução

B) Voltadas para o desenvolvimento de SW

- São direcionados ao desenvolvedor de software
- Precisão em nível de instrução (do processador)
  - Simulação completa do sistema em “tempo real” (acima de 1000 MIPS)
- Análise antecipada de problemas de projeto
  - Consumo de energia (*se instrumentalizado*)
  - Tráfego de barramento
  - Uso de memória
  - Desempenho do sistema

Introdução

Projetar e testar software no início do processo de design está se tornando imprescindível

- Hardware pode ser extremamente complexo ou ainda não estar disponível

Então, propõem-se o uso de Plataformas Virtuais de Desenvolvimento de SW

- Reduzir o *time to market* e os custos no desenvolvimento do software

Usa-se modelagem em nível de sistema

- Para descrever aspectos de hardware em alto nível de abstração

Plataformas Virtuais de Desenvolvimento de Software

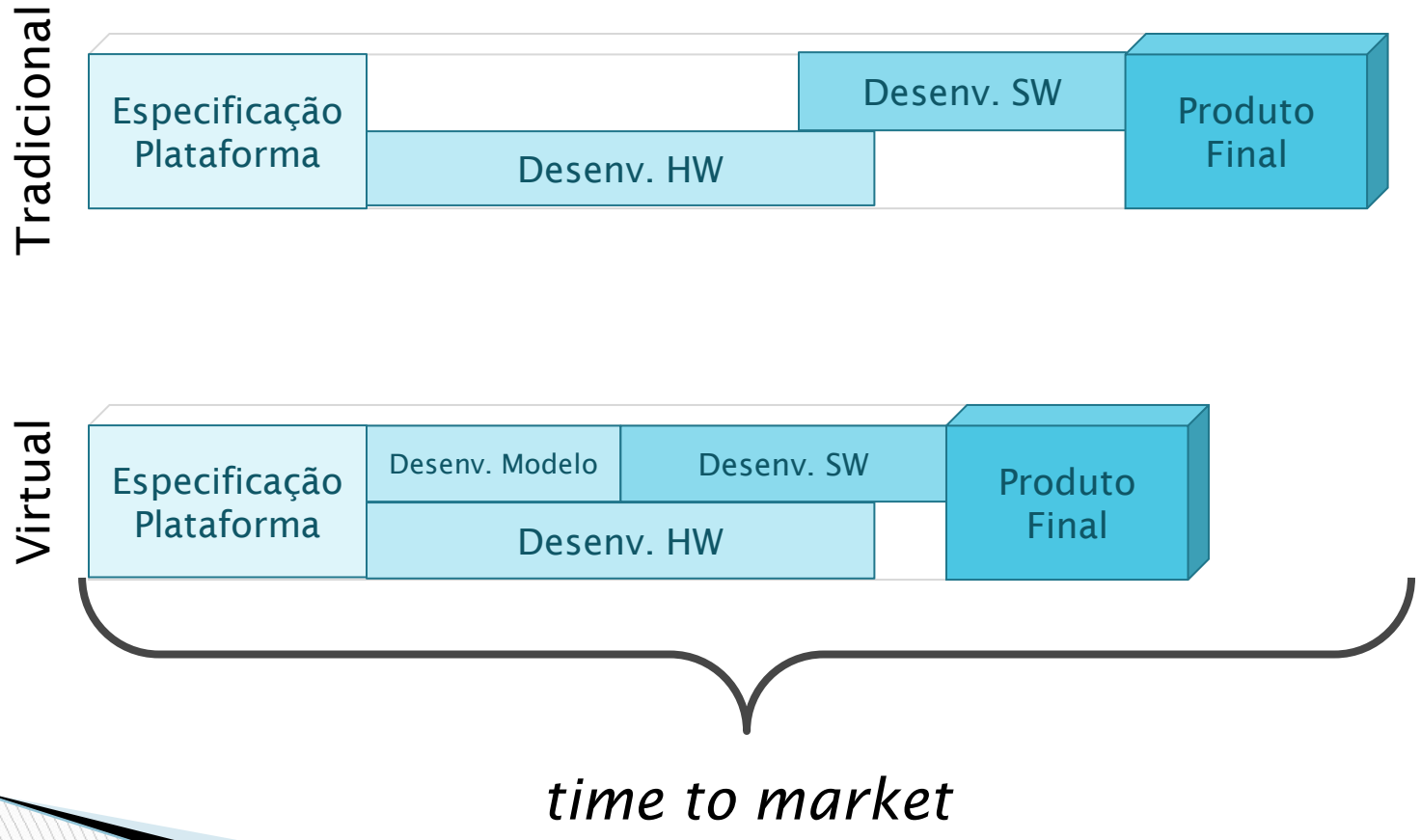
Plataformas Virtuais de Desenvolvimento SW

- Descreve-se, em software, um conjunto de modelos de hardware, representando um sistema completo

- Modelos são parametrizáveis:
  - Tamanho da memória
  - Profundidade de *buffer*
  - Tipos de processadores
  - Periféricos associados

- Projetistas de Hardware modelam e configuram a plataforma virtual e a disponibilizam aos desenvolvedores de software

Plataformas Virtuais de Desenvolvimento SW



Plataformas Virtuais de Desenvolvimento SW

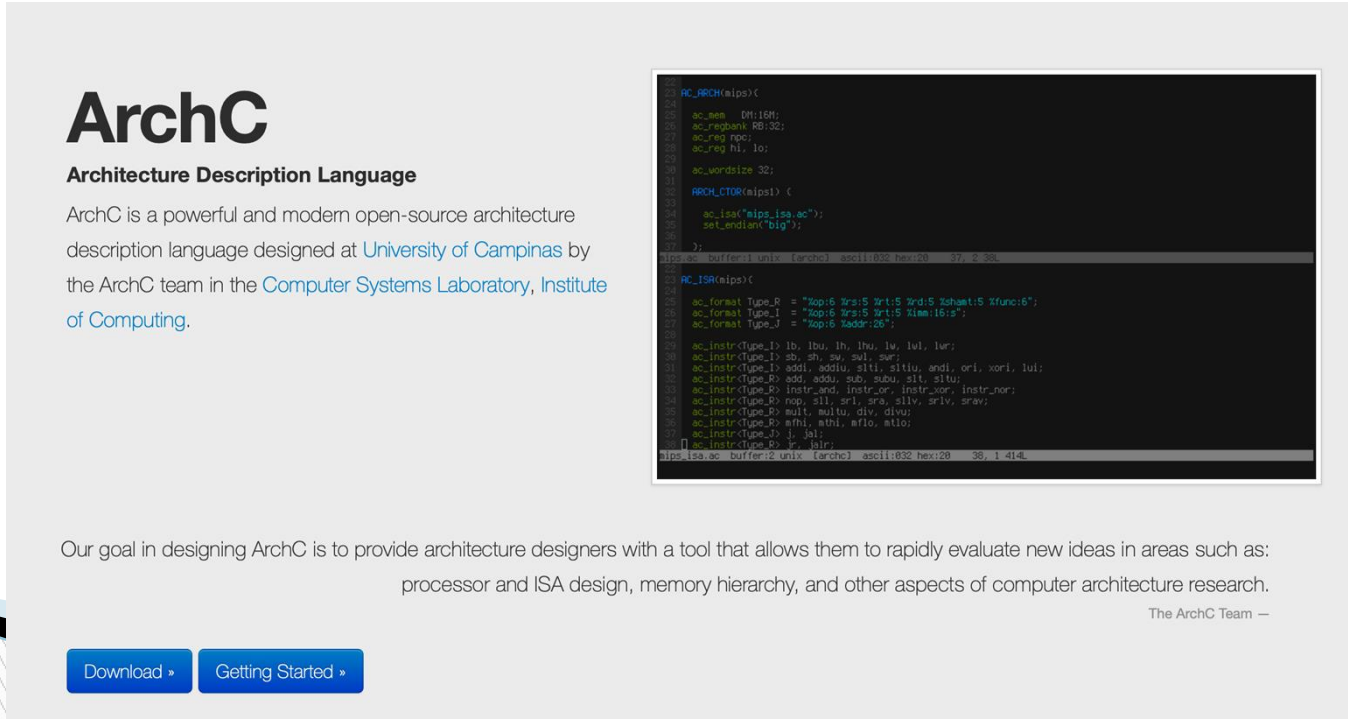
- Requisito:
  - Deve ser possível executar os binários sem necessidade do hardware real

- Vantagem:
  - Não é necessário fornecer todas as funcionalidades de todos os componentes de HW
  - A maioria dos modelos de HW é simples de criar
  - Permitem a construção de uma simulação de todo o sistema, incluindo o ambiente com o qual o sistema embarcado interage

Plataformas Virtuais de Desenvolvimento SW

Exemplos de Plataforma Virtual de Desenv. de SW:

- OVP (Open Virtual Platform)
- ArchC (última atualização 2014) - http://www.archc.org



Open Virtual Platform

OVP

Conceitos

Iniciou em 2008

Ferramenta *open source*, flexível e com licença para 90 dias

Simulação rápida

Configuração de plataformas multiprocessadas homogêneas e heterogêneas é simples

OVP

Conceitos

Três componentes básicos

- **Modelos**
  - Consiste em um conjunto de modelos *open-source* de processadores, memórias, periféricos e plataformas que são disponibilizados gratuitamente para serem usados.
- **Ferramentas**
  - OVPSim, iGen, ISS, harness
- **APIs**
  - Possibilitam a descrição do comportamento de modelos para gerar seu próprio modelo de processadores, memórias, periféricos e plataformas virtuais.
  - As APIs são escritas em linguagem C/C++

OVP - Modelos

Modelos

- OVP disponibiliza várias categorias de modelos:
  - **Processadores** (+/-150 variantes das mais diversas famílias)
    - ARC, ARM, MIPS, PowerPC, Renesas, Altera, Xilinx e OpenRisc.
  - **Memória**
    - ram, rom, cache, etc.
  - **Periféricos**
    - dma, uart, fifo, ethernet, usb, etc.
  - **Plataformas pré-construídas que executam sistemas operacionais**
    - Linux, Android, MQX, Micrium, FreeRTOS e outros
- Esses modelos são fornecidos como código objeto pré-compilado e geralmente com o código fonte.

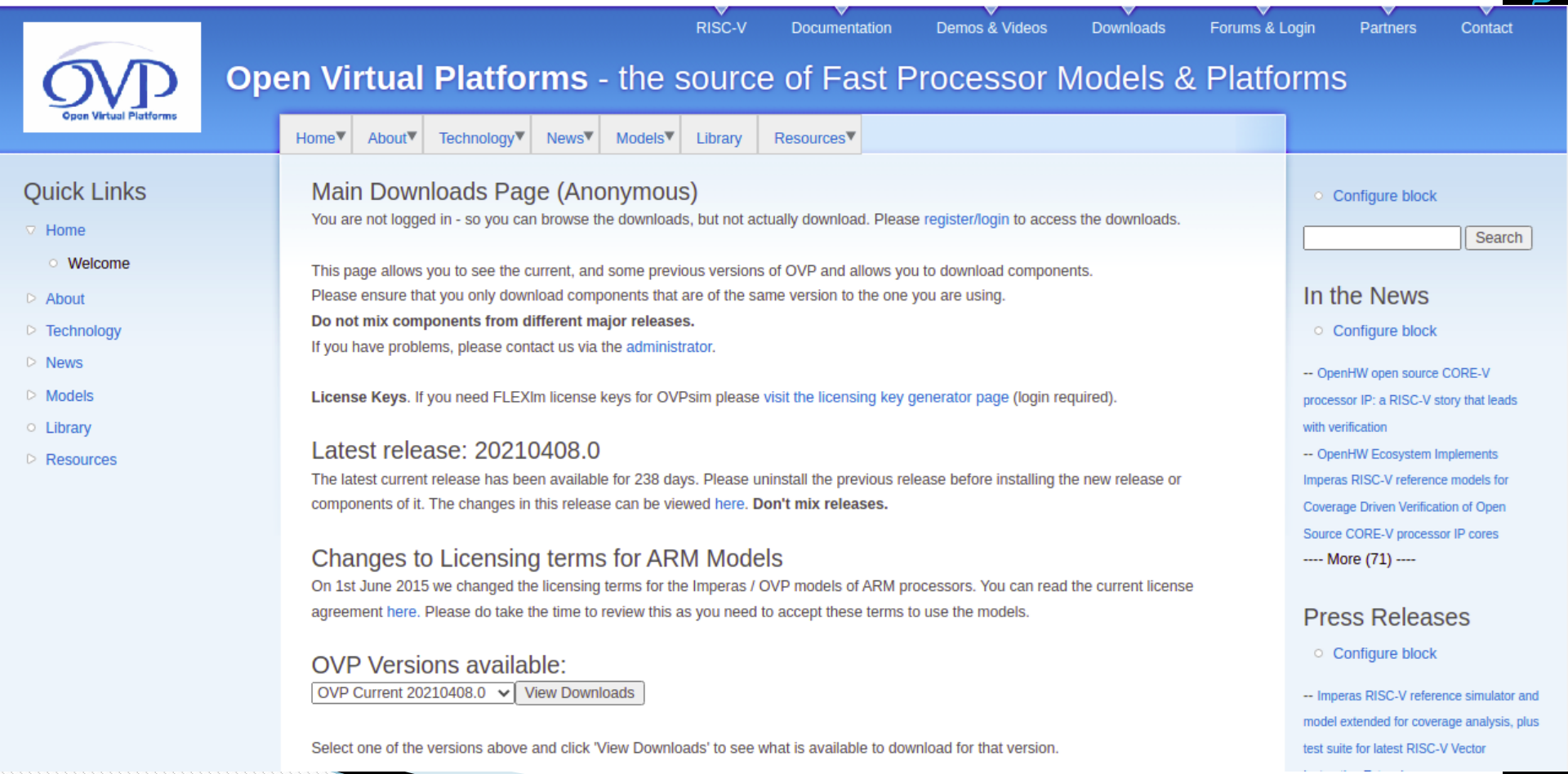
Modelos

Todos modelos estão disponíveis na página do OVP para download

- Cada modelo de processador contém seu *crosscompiler* (deve-se baixar o *toolchain* do modelo requerido)
  - *Toolchain* está disponível para download na mesma seção do modelo, no site do OVP
- Também é possível criar seu próprio modelo de processador, periférico, etc.
  - usando C/C++ e chamadas para as funções da API OVP

Download

https://www.ovpworld.org/dlp/



Documentação

https://www.ovpworld.org/documentation

OVP Documentation

Installation, Getting Started with OVP, and Cross-Compiling Applications
Writing C Platforms and Modules using the OVP OP API
Simulation Control of Platforms and Modules User Guide
Advanced Simulation Control of Platforms and Modules User Guide
iGen Model Generator Introduction
iGen Platform and Module Creation User Guide
iGen Peripheral Generator User Guide
Using OVP models with OSCI SystemC TLM2.0 platforms to gain 200-500 MIPS performance
Using OVP Fast Processor Models with OVPsim and other simulators
Debugging Applications with GDB running on OVP platforms
Debugging with Imperas eGui running on OVP platforms
Debugging Applications with Eclipse running on OVP platforms
Debugging Applications with INSIGHT running on OVP platforms
Control File User Guide
Creating Behavioral (Peripheral) components using BHM/PPM APIs and adding them to Platforms
Function by function Reference Guide for BHM / PPM APIs.
Creating Instruction Accurate Processor models using the VMI API
VMI Morph Time (VMI MT) API Reference Guide
VMI Run Time (VMI RT) API Reference Guide
VMI Memory Modeled Component (VMI MMC) API Reference Guide

Como Instalar um Modelo

- Modelos e seus *toolchains* são encontrados na página de Download no site do OVP

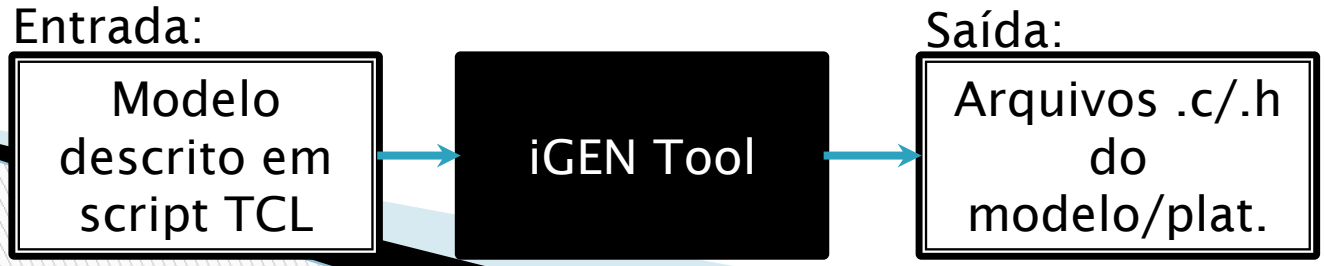
	Cortex CPUs		
ARM Models			
armm.model	ARM OVP Cortex-M Profile Model	Windows(1.46MB)	Linux(2.12MB)
arm.model	ARM Classic, Cortex-A, and Cortex-R Profile OVP Models	Windows(19.1MB)	Linux(23.56MB)
ARM Toolchains			
armv8-aarch64.toolchain	Linaro GCC and GDB tools	Windows(48.46MB)	Linux(40.84MB)
armv8-aarch32.toolchain	Linaro GCC and GDB tools	Windows(83.99MB)	Linux(91.41MB)
armv7.toolchain	ARM Embedded GCC and GDB tools	Windows(50.1MB)	Linux(64.51MB)

OVP - Ferramentas

iGen

- Plataformas e modelos podem ser criados usando C/C++ e chamadas para as funções da API OVP
  - Uso de C e API pode ser entediante e propenso a erros.

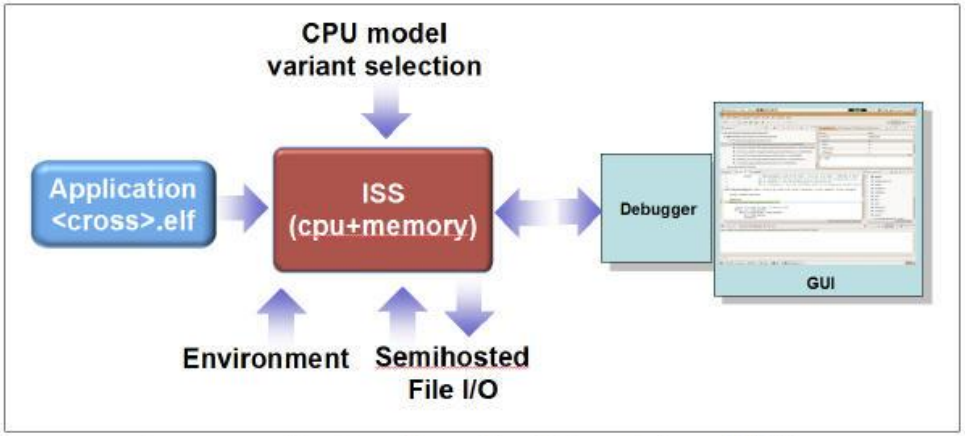
- Para facilitar a criação de plataformas e modelos, a Imperas criou o iGen
  - Usa como entrada a descrição do modelo em um script TCL e gera arquivos C e chamadas de API que definem a plataforma/modelo.
  - Maior parte da descrição pode ser escrita em script de entrada iGen
  - Para modelos próprios, o iGen cria a estrutura e fornece as funções para adição dos comportamentos necessários



ISS – Instruction Set Simulator

- Permite a execução de arquivos binários de software embarcado “cross-compilados” em qualquer uma das mais de 150 variantes de processadores da biblioteca OVP

- É possível simular ou ainda anexar um depurador para fins de verificação/validação do software embarcado



Imperas Instruction Set Simulator (ISS)

OVPSim

- Permite criar plataformas complexas (multiprocessado, memória compartilhada, periféricos)
- Mecanismo simulador que traduz dinamicamente o código binário para instruções de host x86
- Código Proprietário da Imperas

OVP - APIs

OVP

APIs

- As funções das APIs do OVP são responsáveis por instanciar todos os componentes do sistema
  - Processadores
  - Memórias
  - Periféricos

- São divididas em 3 categorias:
  - Modelagem da plataforma
    - ~~ICM~~ e OP - Open Platforms
  - Modelagem do processador
    - VMI - Virtual Machine Interface
  - Modelagem de periféricos
    - BHM - Behavioral Hardware Modeling
    - PPM - Peripheral Programming Model

OVP

APIs

- **Modelagem da plataforma:**
  - **OP** (Open Platform)
    - Permite criar plataformas de forma hierarquia, definindo as conexões entre seus componentes
    - Controlar plataformas/módulos com “testbench” (*harness*)
    - Criação e instanciação de módulos
    - Processadores, barramentos, memórias e periféricos podem ser interconectados em topologias arbitrárias
    - Permite tanto a criação de um *harness* pelo designer ou a simulação com o harness.exe padrão
    - Não é próprio para simulação em nível de clock o objetivo principal é o desenvolvimento rápido de software embarcado

OVP

APIs

- **Modelagem dos processadores**
  - **VMI (*Virtual Machine Interface*)**
    - Utilizada para realizar a descrição do processador
    - Pode-se criar novos modelos de processadores
    - Suporta qualquer formato de instruções
    - O VMI pode ser usado para modelar arquiteturas de 8, 16, 32 e 64 bits e possui extensões para RISC, CISC, VLIW, DSP, etc.
    - Um modelo de processador escrito em C usando o VMI basicamente:
      - decodifica a instrução de destino a ser simulada
      - traduz ela em instruções x86
      - executa essas instruções no PC host

OVP

APIs

- Modelagem dos periféricos
  - **PPM** (*Peripherals Models*)
  - **BHM** (*Behavioral Models*)
  - São utilizados para descrição de modelos de comportamento em hardware e software que sejam periféricos ao processador
  - **Estes modelos executam em um ambiente protegido, sem comprometer a simulação**
  - Junto a estas APIs, há um **mecanismo de escalonamento** baseado em eventos para habilitar modelagem de tempo, eventos e concorrência

OVP

APIs

- **BHM**
  - Modelagem de comportamento
  - Processos, eventos, delays
  - Inicializa processos
  - Aguarda por evento ou tempo
  - Debug através de output
- **PPM**
  - Modelagem de periféricos
  - Interface com a plataforma
  - Conexão com barramento
  - Conexão com a rede
  - Provê callbacks que são chamadas quanto o sw acessa posições de memória onde o periférico está habilitado.

Exemplo 1

Single Processor - OR1K

```
$ cd applications
$ make all
$ cd ..
$ ./RUN_fibonacci.sh
```

```
Info ------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                    : or1k (generic)
Info   Nominal MIPS            : 100
Info   Final program counter   : 0x1948
Info   Simulated instructions: 10,716,781,497
Info   Simulated MIPS          : 1715.0
Info ------------------------------------------------
Info
Info ------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time          : 107.17 seconds
Info   User time               : 6.23 seconds
Info   System time             : 0.02 seconds
Info   Elapsed time            : 6.25 seconds
Info   Real time ratio         : 17.15x faster
Info ------------------------------------------------
```

Exemplo 2

Single Processor - ARM

```
$ cd applications
$ make all
$ cd ..
$ ./RUN_fibonacci.sh
```

```
Info ------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                    : arm (ARM920T)
Info   Nominal MIPS            : 100
Info   Final program counter   : 0x8048
Info   Simulated instructions: 6,185,195,050
Info   Simulated MIPS          : 2612.9
Info ------------------------------------------------
Info
Info ------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time          : 61.85 seconds
Info   User time               : 2.37 seconds
Info   System time             : 0.01 seconds
Info   Elapsed time            : 2.37 seconds
Info   Real time ratio         : 26.13x faster
Info ------------------------------------------------
```

OR1K

```
nfo ------------------------------------------------
nfo CPU 'iss/cpu0' STATISTICS
nfo   Type                  : or1k (generic)
nfo   Nominal MIPS          : 100
nfo   Final program counter : 0x1948
nfo   Simulated instructions: 10,716,781,497
nfo   Simulated MIPS        : 1715.0
nfo ------------------------------------------------
nfo
nfo ------------------------------------------------
nfo SIMULATION TIME STATISTICS
nfo   Simulated time        : 107.17 seconds
nfo   User time             : 6.23 seconds
nfo   System time           : 0.02 seconds
nfo   Elapsed time          : 6.25 seconds
nfo   Real time ratio       : 17.15x faster
nfo ------------------------------------------------
```

ARM

```
Info ------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                  : arm (ARM920T)
Info   Nominal MIPS          : 100
Info   Final program counter : 0x8048
Info   Simulated instructions: 6,185,195,050
Info   Simulated MIPS        : 2612.9
Info ------------------------------------------------
Info
Info ------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 61.85 seconds
Info   User time             : 2.37 seconds
Info   System time           : 0.01 seconds
Info   Elapsed time          : 2.37 seconds
Info   Real time ratio       : 26.13x faster
Info ------------------------------------------------
```

Exemplo 3

MultiProcessor – ARM

```
$ cd application
$ make all
$ cd ..
$ ./Run_MultiCore2.sh
```

Exemplo 2: SingleCore

```
Info ------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                    : arm (ARM920T)
Info   Nominal MIPS            : 100
Info   Final program counter   : 0x8048
Info   Simulated instructions: 6,185,195,050
Info   Simulated MIPS          : 2612.9
```

```
147  %.ARM9T-O3-g.o: %.c
148      @    echo "# Compiling ARM9T $<"
149      $(V) $(ARM9T_CC) -c -o $@ $< -DARM9T $(DEF) -O3 -g
```

```
Info   Simulated time          : 61.85 seconds
Info   User time               : 2.37 seconds
Info   System time             : 0.01 seconds
Info   Elapsed time            : 2.37 seconds
Info   Real time ratio         : 26.13x faster
Info ------------------------------------------------
```

```
Info ------------------------------------------------
Info CPU 'iss/cpu1' STATI
Info   Type
Info   Nominal MIPS
Info   Final program counter
Info   Simulated instructio
Info   Simulated MIPS
Info ------------------------------------------------
Info
Info ------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                    : arm (ARM920T)
Info   Nominal MIPS            : 100
Info   Final program counter   : 0x8048
                               : 4,005,500,292
                               : 462.0
                               ------------------------
```

```
119  %.ARM9T-O1-g.o: %.c
120      @    echo "# Compiling ARM9T $<"
121      $(V) $(ARM9T_CC) -c -o $@ $< -DARM9T $(DEF) -O1 -g
```

Porque diminuiu a quantidade de instruções?

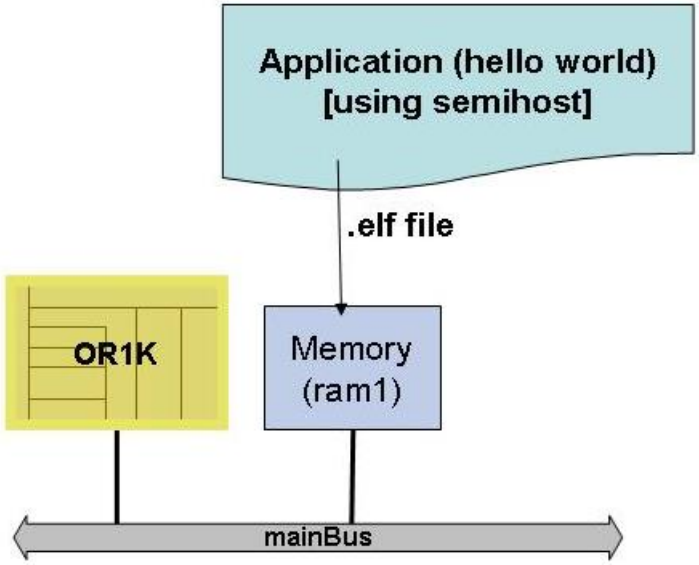
- Ambos calculam fib(0) .. fib(39)
- O código do fibonacci não foi paralelizado

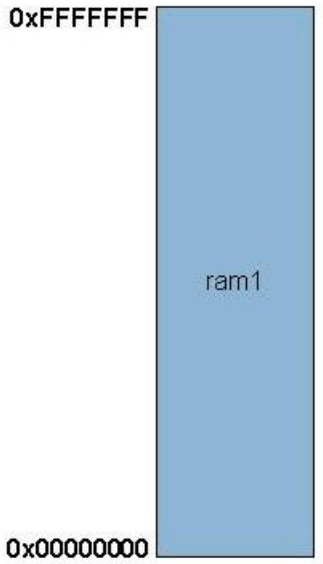
Exemplo 4

Gerar Plataforma usando *iGen*

module.op.tcl

Simple CPU and Memory





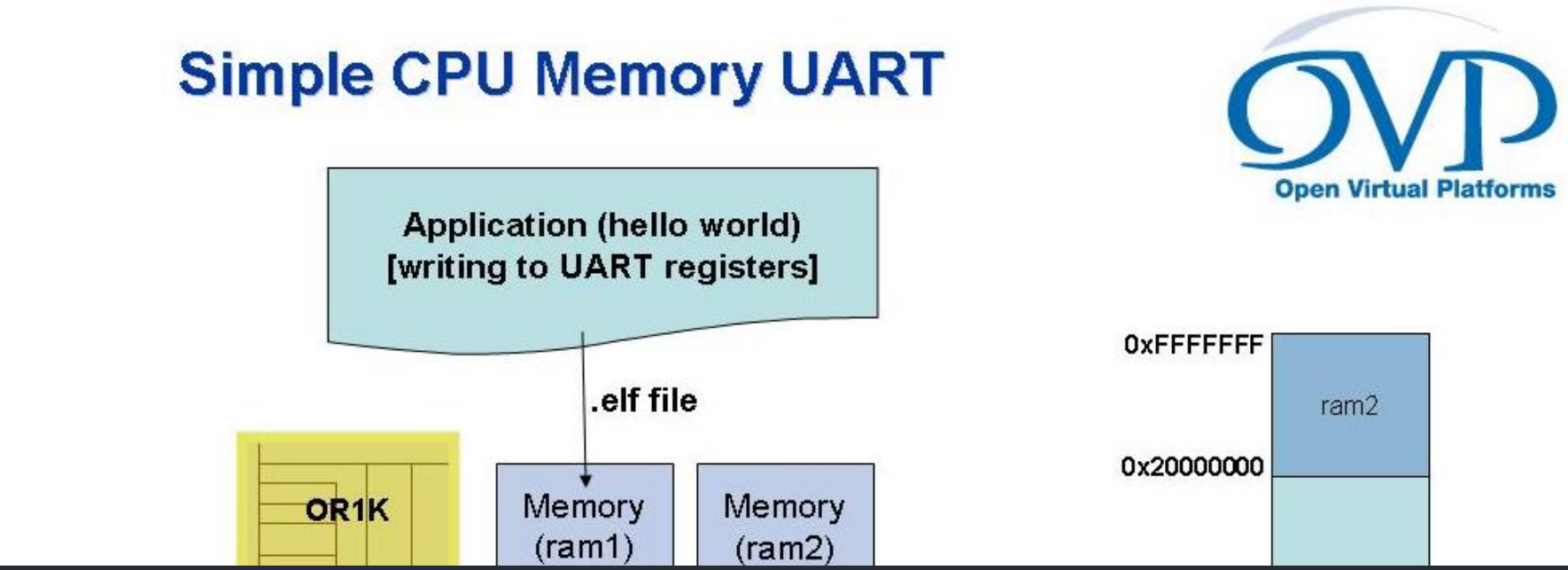
Platform Memory Map

```
19  ihwaddbus -instancename mainBus -addresswidth 32
20
21  ihwaddprocessor -instancename cpu1 \
22                  -vendor ovpworld.org -library processor -type or1k -version 1.0 \
23                  -variant generic \
24                  -semihostname or1kNewlib
25  ihwconnect -bus mainBus -instancename cpu1 -busmasterport INSTRUCTION
26  ihwconnect -bus mainBus -instancename cpu1 -busmasterport DATA
27
28  ihwaddmemory -instancename ram1 -type ram
29  ihwconnect -bus mainBus -instancename ram1 -busslaveport sp1 -loaddress 0x0 -hiaddress 0xffffffff
```

Exemplo 5

Gerar Plataforma usando *iGen* + periférico (UART)

module.op.tcl



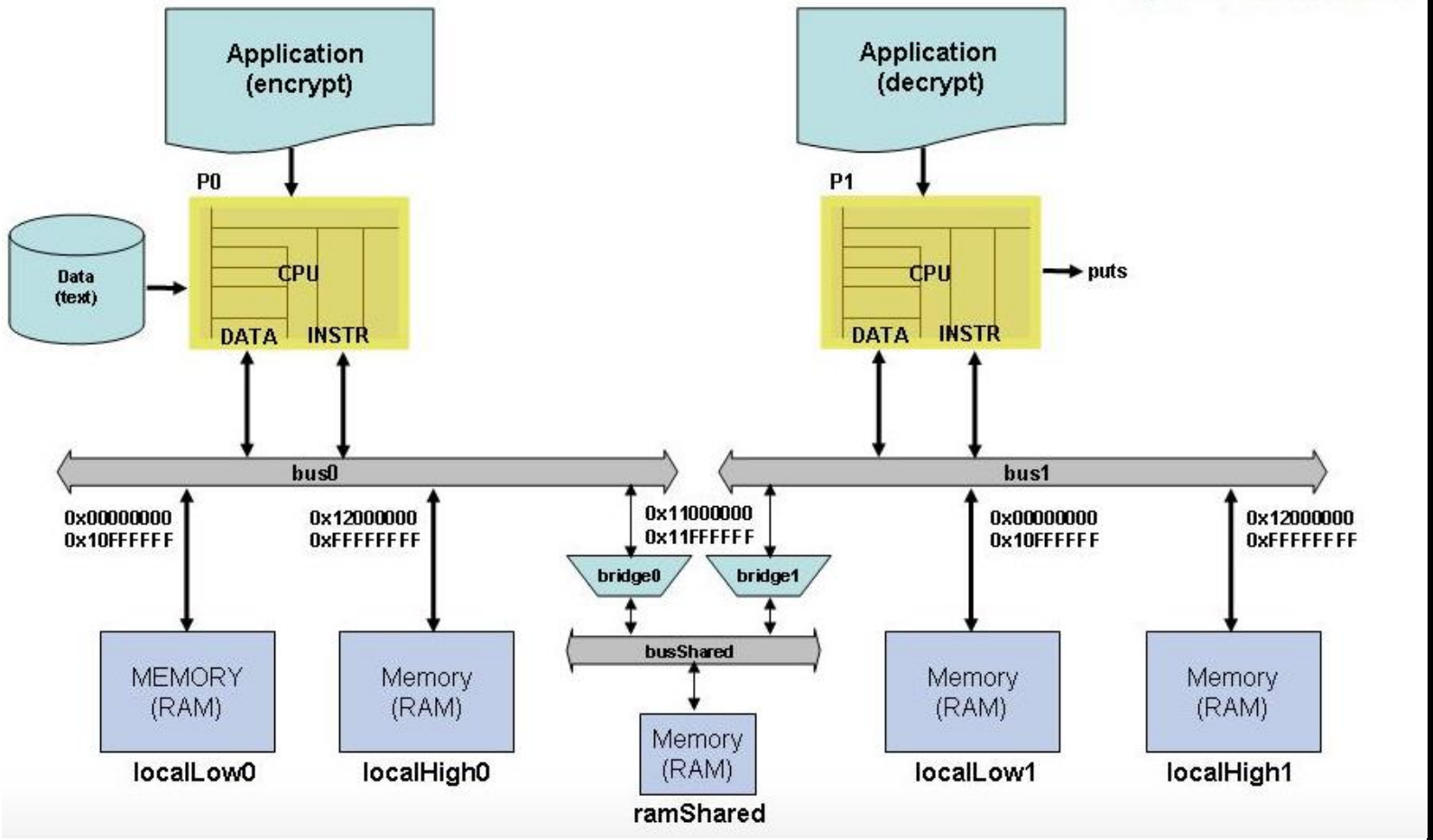

```
23  ihwaddbus -instancename mainBus -addresswidth 32
24
25  ihwaddprocessor -instancename cpu1 \
26                  -vendor ovpworld.org -library processor -type or1k -version 1.0 \
27                  -variant generic \
28                  -semihostname or1kNewlib
29  ihwconnect -bus mainBus -instancename cpu1 -busmasterport INSTRUCTION
30  ihwconnect -bus mainBus -instancename cpu1 -busmasterport DATA
31
32  ihwaddmemory -instancename ram1 -type ram
33  ihwconnect -bus mainBus -instancename ram1 -busslaveport sp1 -loaddress 0x0 -hiaddress 0x0fffffff
34
35  ihwaddmemory -instancename ram2 -type ram
36  ihwconnect -bus mainBus -instancename ram2 -busslaveport sp1 -loaddress 0x20000000 -hiaddress 0xffffffff
37
38  ihwaddperipheral -instancename periph0 -vendor freescale.ovpworld.org -library peripheral -version 1.0 -type KinetisUART
39  ihwsetparameter    -handle periph0 -name         outfile     -value uartTTY0.log    -type string
40  ihwconnect -instancename periph0 -busslaveport bport1      -bus mainBus         -loaddress 0x100003f8 -hiaddress 0x100013f7
41
```

Exemplo 6

Gerar Plataforma usando *iGen* - Multicore + Shared Memory

module.op.tcl

Two processors with local and shared memory



module.op.tcl

```
# Sub-System Zero
#
# add the processor
ihwaddprocessor -type or1k -instancename P0 -semihostname  or1kNewlib -variant generic
# add local memory
ihwaddmemory -type ram -instancename localLow0
ihwaddmemory -type ram -instancename localHigh0
# add the local bus
ihwaddbus -instancename bus0 -addresswidth "32"
# add the bus bridge
ihwaddbridge -instancename bridge0
# add connections to bus0
ihwconnect -bus bus0 -instancename P0         -busmasterport "INSTRUCTION"
ihwconnect -bus bus0 -instancename P0         -busmasterport "DATA"
ihwconnect -bus bus0 -instancename localLow0  -busslaveport  "sp0" -loaddress "0x00000000" -hiaddress "0x10ffffff"
ihwconnect -bus bus0 -instancename bridge0    -busslaveport  "sp0" -loaddress "0x11000000" -hiaddress "0x11ffffff"
ihwconnect -bus bus0 -instancename localHigh0 -busslaveport  "sp0" -loaddress "0x12000000" -hiaddress "0xffffffff"
#
# Sub-System One
#
# add the processor
ihwaddprocessor -type or1k -instancename P1 -semihostname  or1kNewlib -variant generic
# add the local memory
ihwaddmemory -type ram -instancename localLow1
ihwaddmemory -type ram -instancename localHigh1
# add the bus
ihwaddbus -instancename bus1 -addresswidth "32"
# add the bus bridge
ihwaddbridge -instancename bridge1
# add connections to bus1
ihwconnect -bus bus1 -instancename P1         -busmasterport "INSTRUCTION"
ihwconnect -bus bus1 -instancename P1         -busmasterport "DATA"
ihwconnect -bus bus1 -instancename localLow1  -busslaveport  "sp1" -loaddress "0x00000000" -hiaddress "0x10ffffff"
ihwconnect -bus bus1 -instancename bridge1    -busslaveport  "sp1" -loaddress "0x11000000" -hiaddress "0x11ffffff"
ihwconnect -bus bus1 -instancename localHigh1 -busslaveport  "sp1" -loaddress "0x12000000" -hiaddress "0xffffffff"
#
# Shared Resources
#
# add the shared memory
ihwaddmemory -type ram -instancename ramShared
# add the shared bus
ihwaddbus -instancename busShare -addresswidth "32"
# add connections to busShare
# bridge master connections mapping from processor sub-systems
ihwconnect -bus busShare -instancename bridge0   -busmasterport "mp0" -loaddress "0x00000000" -hiaddress "0x00ffffff"
ihwconnect -bus busShare -instancename bridge1   -busmasterport "mp1" -loaddress "0x00000000" -hiaddress "0x00ffffff"
# connection of shared memory
ihwconnect -bus busShare -instancename ramShared -busslaveport  "sp0" -loaddress "0x00000000" -hiaddress "0x00ffffff"
```

Exemplo 7

Custom Harness

harness.c

```
85  int main(int argc, const char *argv[]) {
86      opSessionInit(OP_VERSION);
87      optCmdParserP parser = opCmdParserNew(MODULE_NAME, OP_AC_ALL);
88      cmdParser(parser);
89      opCmdParseArgs(parser, argc, argv);
90      optModuleP mi = opRootModuleNew(&modelAttrs, MODULE_NAME, 0);
91      opRootModuleSimulate(mi);
92      opSessionTerminate();
93
94      return (opErrors() ? 1 : 0);    // set exit based upon any errors
95  }
```

Porque usar o harness customizado???

application.c

```
26  unsigned int trace_fib = 0;
27
28  static int fib(int i) {
29      trace_fib++;
30      return (i>1) ? fib(i-1) + fib(i-2) : i;
31  }
32
33  int main(int argc, char *argv[]) {
34
35      int i;
36      int num = FIB_ITERATIONS;
37
38      printf("starting fib(%d)...\n", num);
39
40      for(i=0; i<num; i++) {
41          printf("fib(%d) = %d\n", i, fib(i));
42      }
43
44      printf("finishing...\n");
45      printf("fib(int i) was called %d times", trace_fib);
46      return 0;
```

Quantas vezes esse programa chama a função fib()?

```
fib(int i) was called 535828550 times
```

Instrumentalizar o código nem sempre pode ser suficiente, além de impactar no desempenho do programa...

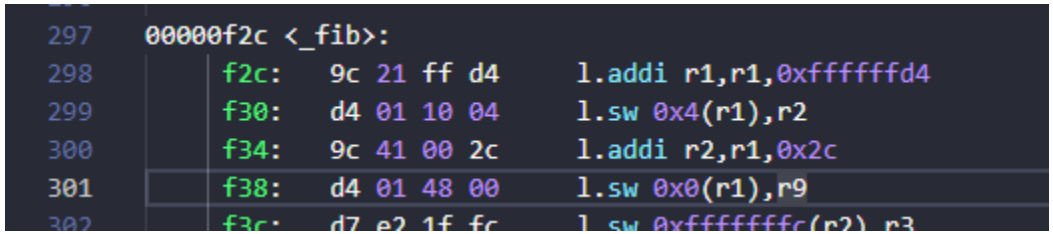
harness.c

Podemos instrumentalizar o harness!

- **Ele é o módulo topo da simulação OVP, tendo acesso a todos os recursos da plataforma (processadores, memória, periférico)**

Como fazer?

- Uma ideia é observar quantas vezes o processador faz fetch no endereço onde começa a função fib()
- Para descobrir o endereço de fib() vamos fazer o dump do código objeto do programa..
  - **`$ or32-elf-objdump -D application.OR1K.elf >> application.S`**



- Vendo o assembly, facilmente descobrimos que o endereço de fib() é 0x0F2C
- Agora vamos instrumentalizar o HARNESS!!!

harness.c

Podemos instrumentalizar o harness!

- **Ele é o módulo topo da simulação OVP, tendo acesso a todos os recursos da plataforma (processadores, memória, periférico)**

```
85  static OP_MONITOR_FN(fib_fetch_CB) {
86      trace_fib++;
87      return;
88  }
89
90  int main(int argc, const char *argv[]) {
91      opSessionInit(OP_VERSION);
92      optCmdParserP parser = opCmdParserNew(MODULE_NAME, OP_AC_ALL);
93      cmdParser(parser);
94      opCmdParseArgs(parser, argc, argv);
95      optModuleP  mi      = opRootModuleNew(0, 0, 0);
96      optModuleP  ui      = opModuleNew(mi, MODULE_DIR, MODULE_INSTANCE, 0, 0);
97
98      // gets the processor
99      optProcessorP proc = opProcessorNext(ui, NULL);
100
101     // pre simulate
102     opRootModulePreSimulate(mi);
103
104     // add a fetch monitor to the processor
105     opProcessorFetchMonitorAdd(proc, 0x0f2c, 0x0f2c, fib_fetch_CB, "fib_trace");
106
107     // Simulate the loaded module
108     opRootModuleSimulate(mi);
109     opMessage("I", "Harness", "fib(int i) was called: %d times", trace_fib);
110     Finishing...
111     Info (Harness) fib(int i) was called: 535828550 times
112
113     return (opErrors() ? 1 : 0);    // set exit based upon any errors
114 }
```

Trabalho OVP

OBJETIVO: exercitar a modelagem de um periférico utilizando a plataforma OVP.

- Deve ser desenvolvido um periférico que irá trabalhar junto com processador, fazendo aceleração de computação.
- Portanto, além de modelar o periférico, será desenvolvida uma aplicação capaz de se comunicar com este periférico
- A aplicação deve realizar uma convolução de matrizes
- A operação de convolução deve ser executada pelo periférico desenvolvido pelos alunos

Trabalho OVP - Peripheral

Criar o **Makefile** no diretório do periférico

```
IMPERAS_HOME := $(shell getpath.exe "$(IMPERAS_HOME)")

include $(IMPERAS_HOME)/ImperasLib/buildutils/Makefile.pse
```

Modelar o periférico utilizando como base o arquivo "**`checkSum.c.igen.stubs`**" que deve ser renomeado para "**`checkSum.c`**"

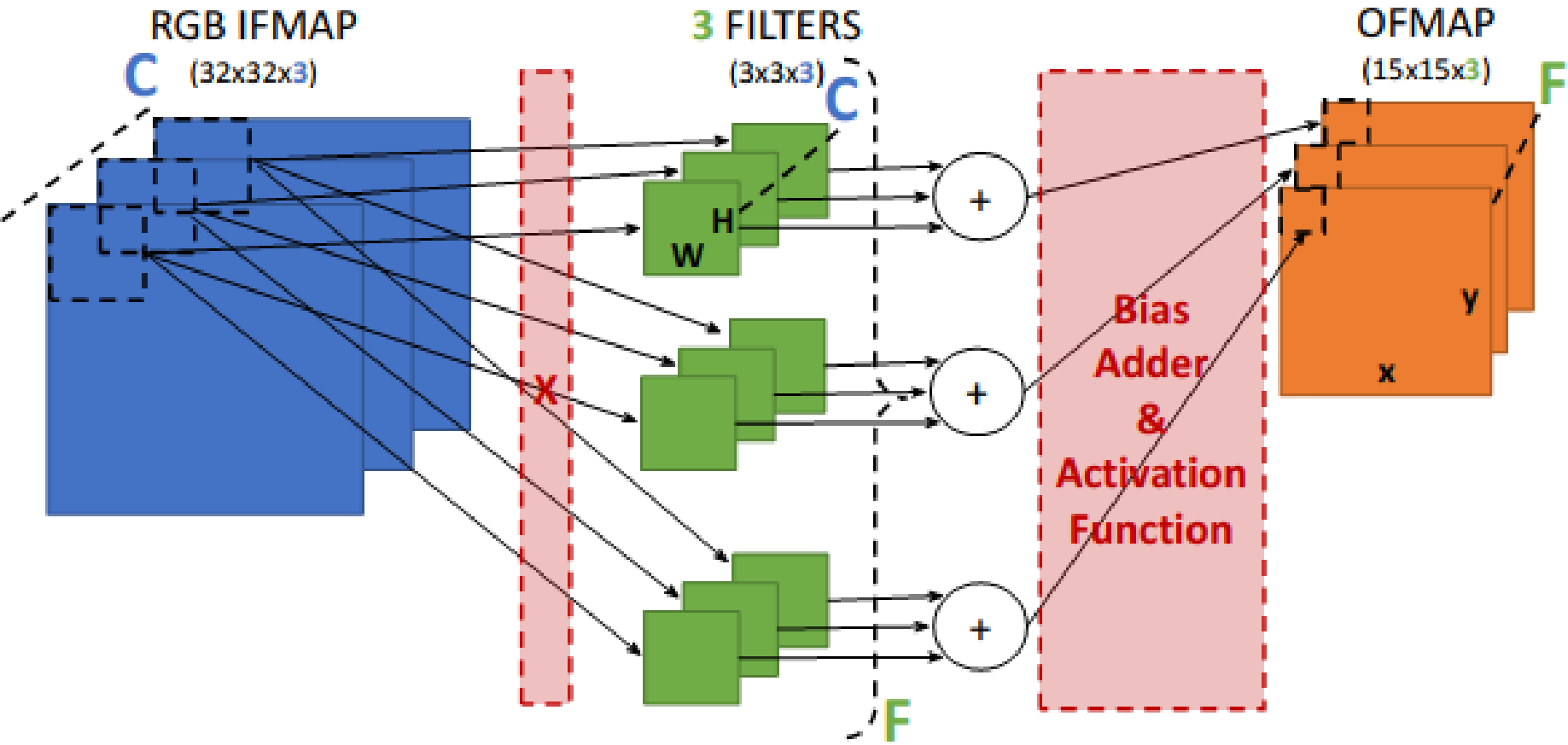
Trabalho OVP - Peripheral

Instanciar o periférico no **`module.tcl`**

```
21  # add the peripheral
22  ihwaddperipheral -instancename checkSum0 -modelfile peripheral/pse.pse
23  ihwconnect -instancename checkSum0 -busslaveport DMAC -bus bus0 -loaddress 0x80000000 -hiaddress 0x8000000B
24  ihwconnect -instancename checkSum0 -busmasterport MREAD  -bus bus0
25
```

Adicionar os registradores mapeados em memória na aplicação...

```
// defines the peripheral base address
#define CHECKSUM_BASE ((unsigned int *)0x80000000)

// defines the memory mapped registers
volatile unsigned int *cs_address   = CHECKSUM_BASE + 0x0;
volatile unsigned int *cs_status    = CHECKSUM_BASE + 0x1;
volatile unsigned int *cs_checkSum  = CHECKSUM_BASE + 0x2;
```

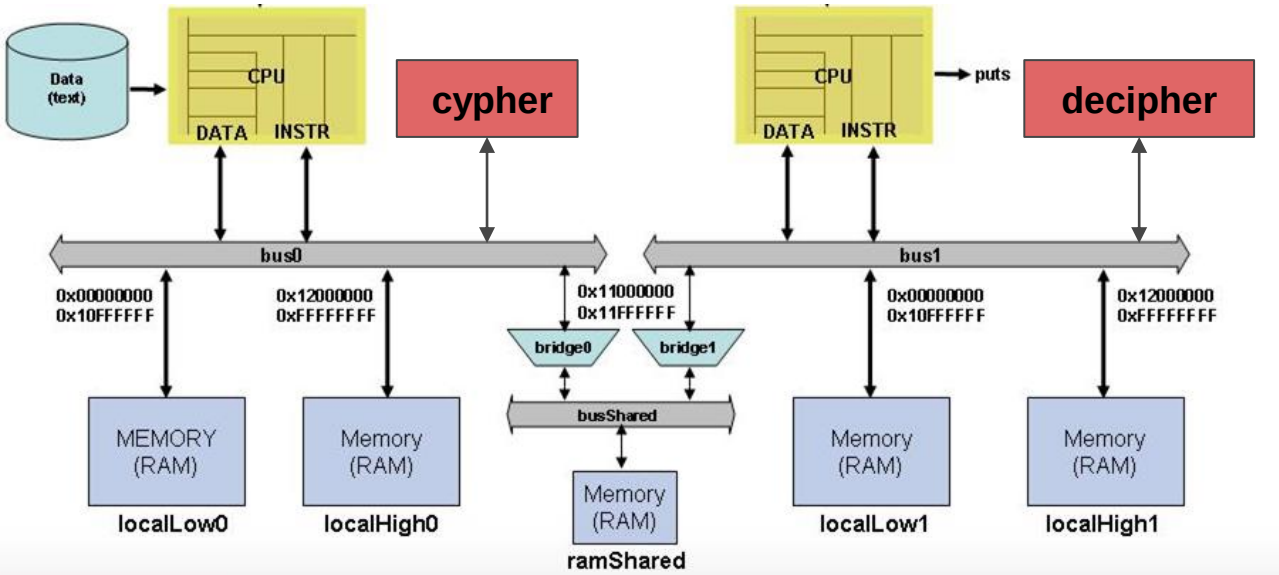


WF: Set of filters **w(f,c)**

Trabalho OVP - Peripheral

Para maiores detalhes da utilização do periférico, veja o exemplo checkSum disponibilizado.

1. Aplicação de que codifique e decodifique uma mensagem em um sistema multiprocessado com memória compartilhada utilizando um periférico para executar a codificação e a decodificação.
2. **A aplicação deve ler a mensagem de um arquivo**. A aplicação **DEVE** ser capaz de tratar mensagens de qualquer tamanho.
3. Fazer um pequeno relatório comentando o código e providenciar um .zip contendo um cenário pronto para execução.



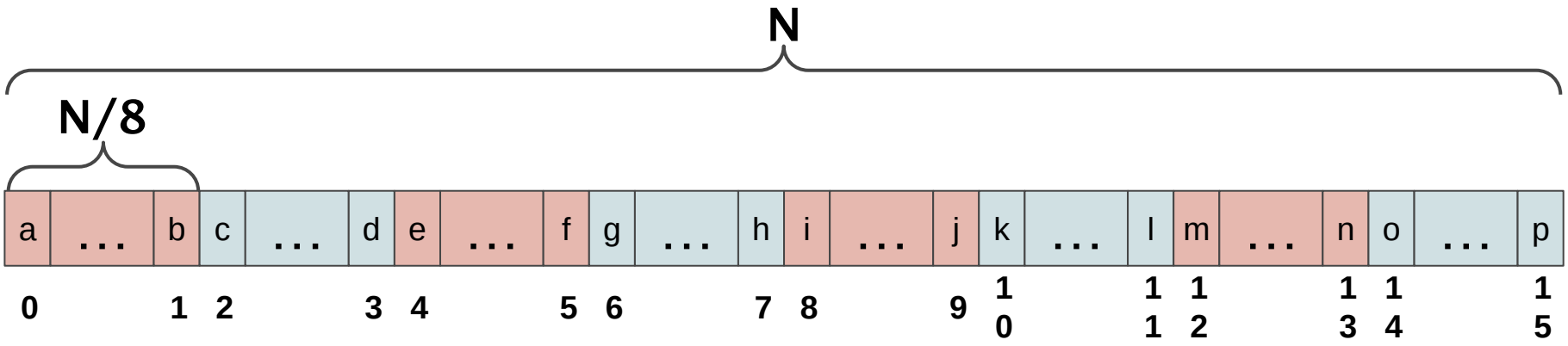
Trabalho OVP - Peripheral

Como gerar um periférico?

```
imodelnewperipheral -name checkSum \
                    -constructor constructor \
                    -destructor  destructor \
                    -vendor gaph \
                    -library peripheral \
                    -version 1.0

iadddocumentation -name Description \
                  -text "A OVP checkSum peripheral"

########################################
# Master read and write ports
########################################
imodeladdbusmasterport -name "MREAD"   -addresswidth 32

########################################
## A slave port on the processor bus
########################################
imodeladdbusslaveport -name DMAC -size 12 -mustbeconnected

# Address block
imodeladdaddressblock -name ab -port DMAC -offset 0x0 -width 32 -size 12

# 8 bit control registers
imodeladdmmregister -addressblock DMAC/ab -name address  -readfunction address_R  -writefunction address_W  -offset 0
imodeladdmmregister -addressblock DMAC/ab -name statusCS -readfunction statusCS_R -writefunction statusCS_W -offset 4
imodeladdmmregister -addressblock DMAC/ab -name checkSum -readfunction checkSum_R -writefunction checkSum_W -offset 8

```

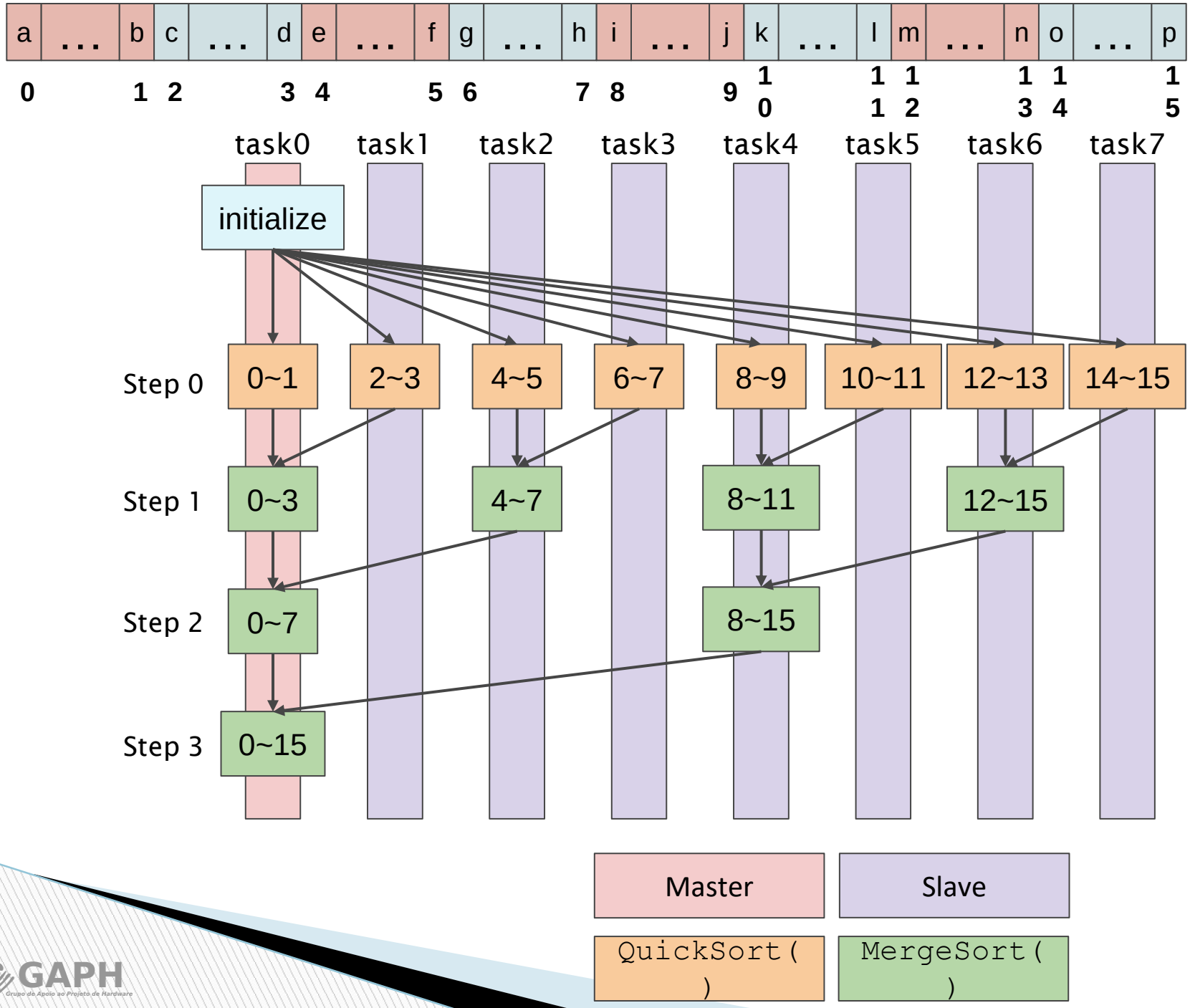
```
igen.exe --batch checkSum.tcl --writec checkSum
```

Trabalho OVP - Sort Shared Memory



1. Aplicação de *sorting* em sistema multiprocessado com memória compartilhada utilizando o paradigma de mestre-escravo e *Binary Tree Reduction*.
2. **A aplicação deve aceitar qualquer tamanho de vetor**. A aplicação **DEVE** ser capaz de tratar vetores de tamanho que **não** seja divisível pelo número de tarefas. *Por exemplo*, a aplicação deve ser capaz de ordenar um vetor de 103 posições em um sistema de 8 processadores.
3. Fazer um pequeno relatório comentando o código e providenciar um .zip contendo um cenário pronto para execução.

Organizar vetor de 1E6 elementos usando QuickSort() levou 162 minutos. Usando divisão e conquista com 8 processadores levou 21 min.


a ... b c ... d e ... f g ... h i ... j k ... l m ... n o ... p
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
task0 task1 task2 task3 task4 task5 task6 task7
initialize
Step 0
0~1 2~3 4~5 6~7 8~9 10~11 12~13 14~15
Step 1
0~3 4~7 8~11 12~15
Step 2
0~7 8~15
Step 3
0~15
Master
Slave
QuickSort( )
MergeSort( )

Trabalho OVP - Instruction Counting

Este trabalho tem por objetivo a instrumentalização do harness.c para gerar um relatório da quantidade de cada tipo de instrução que o processador utilizou para a execução de um determinado programa.

1. Modificações no harness de forma a criar uma instrumentalização capaz de contar as instruções (separadas em categorias) que foram executadas pela aplicação.
2. Ao final da execução o harness deve imprimir um relatório em formato de texto contendo as informações relativas à execução do programa.
3. Fazer um pequeno relatório comentando o código e providenciar um .zip contendo um cenário pronto para execução.