Exercício 3

NOME	RA
Renan Camargo de Castro	147775

Introdução

Nesse exercício, vamos montar um processador mips dual-core com alguns componentes básicos, como router e periféricos de lock. Também será abordado alguns temas como o uso de programas realmente paralelos.

Montagem do processador

Para “montar” o processador, foi utilizada o esquema de um processador mips no Archc com algumas modificações.

Memória compartilhada

Para a memória compartilhada, foi utilizado um componente **ac_tlm_mem**(já fornecido) que basicamente instancia um vetor de *uint8_t* e utiliza isso em conjunto com o router para armazenar a memória de ambos os processadores.

Roteador

A primeira modificação foi a inclusão de um router que basicamente tem a função de mapear endereços de memória para periféricos específicos. O roteador foi montado de forma à rotear os seguintes endereços:

- **[0,0x6400000)** : Endereços mapeados para o bloco de memória compartilhada.
- **[0x6400000,∞)**: Endereços mapeados para o periférico de lock.

Além disso, foi feita uma modificação no roteador para ter duas portas de comunicação, visto que serão utilizados dois processadores, logo, são feito 2 *target_export* nos arquivos **ac_tlm_router**(modificações no .cpp e .h).

Periférico de Lock

Esse periférico foi codificado para funcionar como um lock global no sistema para poder separa o fluxo de execução dos dois processadores. Para isso, foi utilizada uma variável simples privada dentro da

classe/arquivo **ac_tlm_peripheral** que quando lida é setada para 1, e é passível de escrita. O endereço base desse periférico, como definido no router, é 0x6400000.

Instanciação dos processadores

Para ligar todos os componentes, foi instanciado no arquivo **main.cpp** dois processadores de classe **mips**. Conecta-se o periférico e a memória nas portas do roteador **MEM_port** e **PERIPHERAL_port** e também liga-se as duas portas do roteador nos respectivos processadores: **DM_port**.

Um detalhe importante da instanciação são os argumentos passados para os processadores, para isso foi duplicado os argumentos passados. Essa cópia é passada no método init de cada processador.

Programas

Hello World

Para testar o funcionamento inicial, foi feito um programa simples como se fosse um “hello world” para testar o funcionamento dos locks e dos processadores. Esse programa está com o nome **“hello_world.c”** no repositório. *ps: todos os programas foram compilados com a flag -O*

A saída foi a seguinte:

```bash
-bash-4.3$ ./mips-tlm/mips.x --load=hello_world

        SystemC 2.3.1-Accellera --- Apr  8 2016 08:47:14
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
ArchC: Reading ELF application file: hello_world
@@@@ SP for processor 0 : 0x63ffc00
ArchC: -------------------- Starting Simulation --------------------
ArchC: Reading ELF application file: hello_world
@@@@ SP for processor 1 : 0x63bfc00
ArchC: -------------------- Starting Simulation --------------------

proc 0: entrou na regiao critica
ArchC: -------------------- Simulation Finished --------------------
proc 1: entrou na regiao critica
ArchC: -------------------- Simulation Finished --------------------

Info: /OSCI/SystemC: Simulation stopped by user.
ArchC: Simulation statistics
    Times: 0.00 user, 0.00 system, 0.00 real
    Number of instructions executed: 4316
    Simulation speed: (too fast to be precise)

ArchC: Simulation statistics
    Times: 0.00 user, 0.00 system, 0.00 real
    Number of instructions executed: 6954
    Simulation speed: (too fast to be precise)
```

Hello World Dual Core

O programa faz um hello world recursivo utilizando lock e imprimindo os números dos dois cores. Saída:

Bash

```
-bash-4.3$ ./mips-tlm/mips.x --load=dualCore_hello_world

        SystemC 2.3.1-Accellera --- Apr  8 2016 08:47:14
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
ArchC: Reading ELF application file: dualCore_hello_world
@@@@ SP for processor 0 : 0x63ffc00
ArchC: -------------------- Starting Simulation --------------------
ArchC: Reading ELF application file: dualCore_hello_world
@@@@ SP for processor 1 : 0x63bfc00
ArchC: -------------------- Starting Simulation --------------------

Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
Hi from processor 0!
ArchC: -------------------- Simulation Finished --------------------
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
Hi from processor 1!
ArchC: -------------------- Simulation Finished --------------------

Info: /OSCI/SystemC: Simulation stopped by user.
ArchC: Simulation statistics
    Times: 0.03 user, 0.00 system, 0.03 real
    Number of instructions executed: 21613
    Simulation speed: (too fast to be precise)

ArchC: Simulation statistics
    Times: 0.03 user, 0.00 system, 0.03 real
    Number of instructions executed: 219977
    Simulation speed: (too fast to be precise)
```

Soma de números

A aplicação que foi desenvolvida para testar o desempenho da arquitetura multicore foi a soma de números até N. Para isso, foi dividida entre os processadores metade dos números, isso é, processador 1 soma de [0,n/2) e o processador 2 de [n/2,n]. No final, ambos esperam o outro ter terminado, e o processador 1 soma as duas variáveis de soma local dos processadores da memória compartilhada, e ele printa essa variável.

O valor de n utilizado é **5000000**.

Para comparar o desempenho, também foi desenvolvida uma aplicação com a mesma função, mas single core. Os arquivos são: **sum_numbers_dual_core** e **sum_numbers_one_core**.

Segue exemplo de saída:

Bash

```
-bash-4.3$ time ./mips-tlm/mips.x --load=sum_numbers_dual_core

        SystemC 2.3.1-Accellera --- Apr  8 2016 08:47:14
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
ArchC: Reading ELF application file: sum_numbers_dual_core
@@@@ SP for processor 0 : 0x63ffc00
ArchC: -------------------- Starting Simulation --------------------
ArchC: Reading ELF application file: sum_numbers_dual_core
@@@@ SP for processor 1 : 0x63bfc00
ArchC: -------------------- Starting Simulation --------------------

processador 0
processador 1
output from processor 0: 12500002500000
ArchC: -------------------- Simulation Finished --------------------
ArchC: -------------------- Simulation Finished --------------------

Info: /OSCI/SystemC: Simulation stopped by user.
ArchC: Simulation statistics
    Times: 4.02 user, 0.00 system, 4.02 real
    Number of instructions executed: 37511359
    Simulation speed: 9331.18 K instr/s

ArchC: Simulation statistics
    Times: 4.02 user, 0.00 system, 4.02 real
    Number of instructions executed: 37511376
    Simulation speed: 9331.19 K instr/s


real    0m4.026s
user    0m4.024s
sys     0m0.003s
```

```Bash
-bash-4.3$ time ./mips-tlm/mips.x --load=sum_numbers_one_core

        SystemC 2.3.1-Accellera --- Apr  8 2016 08:47:14
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
ArchC: Reading ELF application file: sum_numbers_one_core
@@@@ SP for processor 0 : 0x63ffc00
ArchC: -------------------- Starting Simulation --------------------
ArchC: Reading ELF application file: sum_numbers_one_core
@@@@ SP for processor 1 : 0x63bfc00
ArchC: -------------------- Starting Simulation --------------------

processador 0
processador 1
ArchC: -------------------- Simulation Finished --------------------
output from processor 0: 12500002500000
ArchC: -------------------- Simulation Finished --------------------

Info: /OSCI/SystemC: Simulation stopped by user.
ArchC: Simulation statistics
    Times: 4.87 user, 0.00 system, 4.87 real
    Number of instructions executed: 75009405
    Simulation speed: 15402.34 K instr/s

ArchC: Simulation statistics
    Times: 4.87 user, 0.00 system, 4.87 real
    Number of instructions executed: 5455
    Simulation speed: 1.12 K instr/s


real    0m4.878s
user    0m4.874s
sys     0m0.004s
```

Comparação de desempenho

Utilizando o último programa apresentado na última seção, **sum_numbers_dual_core** *x* **sum_numbers_one_core**, podemos construir a seguinte tabela comparativa:

Critério	Valor - Single core	Valor - Dual core
Instruções(#0)	75009405	37511359
Instruções(#1)	5455	37511376
Tempo TOTAL	0m4.864s	0m4.026s

Vemos que no dual core existe basicamente uma divisão igualitária entre os processadores no quesito instruções. O esperado seria que fosse metade do tempo de execução do single core, mas isso não acontece devido à necessidade de se usar lock em alguns lugares e também de sincronização. O ganho, nesse caso, foi de aproximadamente **20,8%**.