

HeMPS 7 Tutorial

Por: Marcelo Ruaro, Guilherme Madalozzo, Fernando Gehm Moraes
Contato: marcelo.ruaro@acad.pucrs.br

Versão atual: **7.2.**

Data da liberação: 31/08/2015

Recursos suportados:

- *Reclustering*
- Repositório de aplicações unificado
- Contagem de instruções
- Simulação em SystemC
- DMA com endereço duplo para leitura de memória e escrita na rede
- Simulação VHDL
- ~~Simulação OVP~~
- Hold
- Mapeamento Load
- ~~Mapeamento Estático~~
- Migração de Tarefas

Existem dois modos de executar e simular aplicações no MPSoC HeMPS:

- Fazendo download da HeMPS e executando localmente;
- Via servidor Kriti (GAPH/PUC-RS)

➤ Executando Localmente

1. Fazer o *download* do repositório: https://corfu.pucrs.br/svn/hemps_exe/tags/7.2

Descrição da árvore de diretórios:

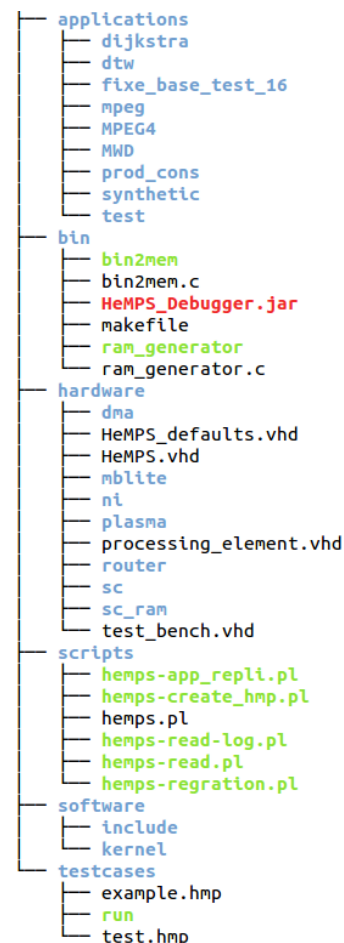
A HeMPS é composta por 6 diretórios principais:

- applications:** contém o código fonte (.c) das aplicações;
- bin:** contém arquivos executáveis úteis para a configuração da memória e depuração;
- hardware:** contém a descrição dos componentes hardware em VHDL e SystemC RTL;
- scripts:** contém os scripts úteis e o script *hemps.pl*, que é executado para configurar a plataforma através de parâmetros informados em tempo de projeto. Estes parâmetros se encontram nos arquivos de testecase (.hmp);
- software:** contém os códigos fontes da camada de software;
- testcase:** contém os testcases que serão utilizados pelo script *hemps.pl*.

Após o download do repositório é necessário compilar o executáveis do diretório **bin**:

2. Criar uma variável de ambiente chamada HEMPS_PATH com o caminho do diretório raiz da HeMPS. Ex:

```
export HEMPS_PATH=/home/user/Documents/hemps
```



3. Definir na variável path o ponteiro para as pastas /bin e /scripts, do diretório raiz da HeMPS. Ex:

```
export PATH=$PATH:$HEMPS_PATH/bin:$HEMPS_PATH/scripts
```

4. Para execução da HeMPS é necessário que alguns compiladores estejam instalados. Estes compiladores são o *MIPS Cross Compiler* (que compila a parte de *software*), e o *SystemC* (que compila a parte de *hardware*). Computadores do GAPH podem fazer acesso direto a esses módulos através dos comandos:

```
source /soft64/source_gaph
module load mips
module load systemc
```

Outros usuários deverão fazer manualmente o download e a instalação desses compiladores em seus computadores.

5. Após realizar os passos anteriores deve-se entrar no diretório raiz da hempo: /hempo\$

6. Entrar no diretório bin: /hempo/bin\$

7. Fazer um *make all*: /hempo/bin\$ make all

A próxima etapa consiste em criar um arquivo de *testcase* para ser executado. A criação do arquivo deve ser feita diretamente na pasta *testcases*: /hempo6.0/testcases. A estrutura básica de um *testcase* é descrita como segue:

[project name]

Este nome deve ser o mesmo do arquivo de testcase, ex: nome_do_projeto.hmp.

[tasks per pe]

Número de tarefas por PE, um valor típico configurado pela equipe do GAPH é uma faixa entre 1 e 6 tarefas(1-6).

[processor description]

Descrição da infraestrutura de hardware. *sc* - simulação em SystemC (GCC), *scmod* - simulação em SystemC utilizando a ferramenta ModelSim, *rtl* - simulação em VHDL utilizando a ferramenta ModelSim

[noc buffer size]

Parâmetro com número inteiro que permite configurar a profundidade dos buffers dos roteadores da NoC, um valor típico é 8.

[noc routing algorithm]

Parâmetro que permite configurar o algoritmo de roteamento utilizado na NoC, os parâmetros suportados são: *xy*, *wf*

[dimensions]

Dois parâmetros numéricos inteiros que permitem definir a dimensão do MPSoC. Não é suportado uma dimensão menor que 2x2. Estes parâmetros devem ser informados utilizando duas linhas do scripts. Ex:

```
6
6
```

[cluster size]

Dois parâmetros numéricos inteiros que permitem definir a dimensão de cada cluster do MPSoC. Não é suportado uma dimensão menor que a dimensão do próprio MPSoC, além disso o número de processadores do cluster deve ser divisível (com resto igual a 0) pelo número de processadores do MPSoC. Estes parâmetros devem ser informados utilizando duas linhas do scripts. Ex:

```
3
3
```

[masters location]

Localização onde o processador mestre global irá ser gerado. Atualmente a única posição suportada é LB (Left Bottom).

[global master]

Endereço de rede do processador mestre global, esta variável numérica inteira está relacionada em função da posição do mestre no MPSoC. O valor padrão é 0.

Os próximos parâmetros estão relacionados as aplicações que executarão no sistema, a mesma estrutura deve ser utilizada para cada aplicação que irá executar.

[application]

Nome da aplicação que será executada na plataforma, este nome deve ser o mesmo das aplicações presentes no diretório *applications*.

[start time]

Tempo de início de execução da aplicação descrita acima, este valor numérico inteiro é representado em **um** (microsegundos - μm) ou **ms** (milissegundos). Esse parâmetro é opcional, caso não estiver presente a aplicação irá iniciar assim que o sistema puder admiti-la.

Exemplo de instanciação da aplicação mpeg, que inicia em 1ms.

```
[application]
mpeg
[start time]
1 ms
```

Um testcase de exemplo, chamado *example.hmp*, encontra-se no diretório *testcases*. Para executá-lo deve-se considerar se o *testcase* será executado pelo GCC ou Modelsim. Para fazer esta verificação, deve-se analisar se a *tag* [processor description] está com *sc* (GCC) ou *scmod* (modelsim).

Execução somente pelo GCC

8. Voltar para o diretório raiz da hems: `/hems$`
9. Entrar no diretório testcases: `cd testcases`
10. Executar o script: `hems <nome_testase>.hmp <tempo_ms>`. Ex: `hems example.hmp 50`

Execução pelo ModelSim

8. Voltar para o diretório raiz da hems: `/hems$`
9. Entrar no diretório scripts: `cd scripts`
10. Alterar a permissão para o scripts *hems.pl*: `chmod 777 hems.pl`
11. Voltar para o diretório raiz da hems: `/hems$`
12. Entrar no diretório testcases: `cd testcases`
13. Executar o scripts: `hems <nome_testcase>.hmp`
14. Entrar no diretório do *testcase*: `cd <nome_testcase>`
15. Executar o comando: `make all`
16. Abrir o ModelSim: `vsim`
17. No shell do ModelSim executar o comando: `do sim.do`

Ferramenta de depuração: HeMPS Debugger

A ferramenta de depuração foi desenvolvida em Java e o seu executável (.jar) encontra-se no diretório */bin* da plataforma. A ferramenta é totalmente gráfica e possui diversas funções que permitem avaliar a simulação verificando o comportamento das tarefas, serviços e comunicação. Entre suas principais funções destacam-se:

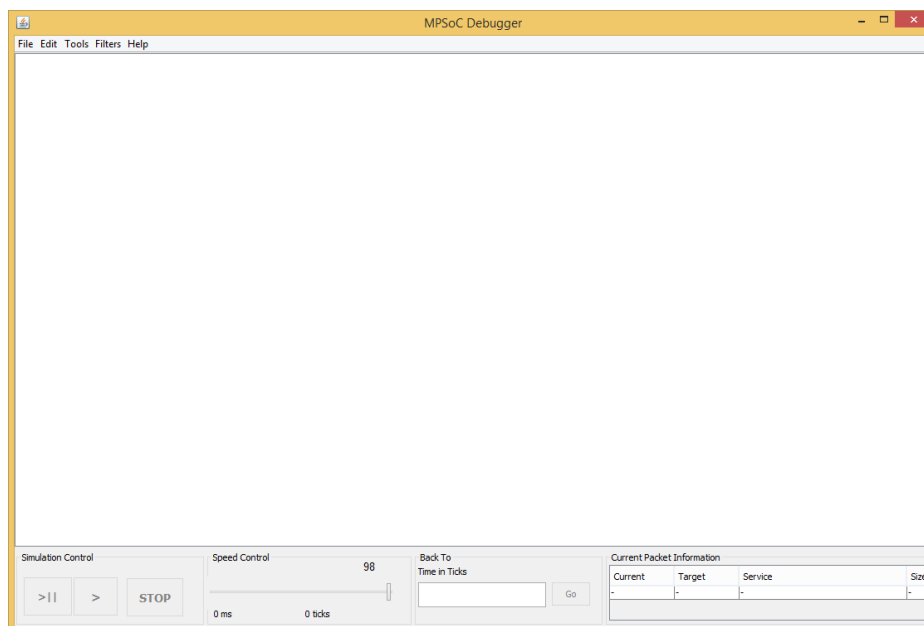
- Acompanhamento da execução em tempo de simulação;
- Visualização do mapeamento das tarefas;
- Verificação dos protocolos, tanto de comunicação quanto de gerenciamento;
- Verificação dos percentual de utilização de cada link entre um par de roteadores;

- Verificação do percentual de comunicação em cada roteador, tanto no nível de serviço quanto no nível de flits;
- Medição do tempo de cada protocolo;
- Visão geral da distribuição de serviços executados por cada PE;
- Verificação do sincronismo de comunicação entre tarefas baseado no protocolo *read-request*;
- Suporte multiplataforma (testado em Windows, Linux e MAC OS);
- Visualização dos logs das tarefas (Deloream).

O HeMPS Debugger é executado automaticamente na simulação por SystemC (GCC). Quando configurar um testcase com simulação por ModelSim, deve-se dar um clique duplo sobre o ícone HeMPS_Debugger.jar presente no diretório */bin* da plataforma.

OBS: É necessário ter a JVM (*Java Virtual Machine*) instalada na versão 6 ou acima.

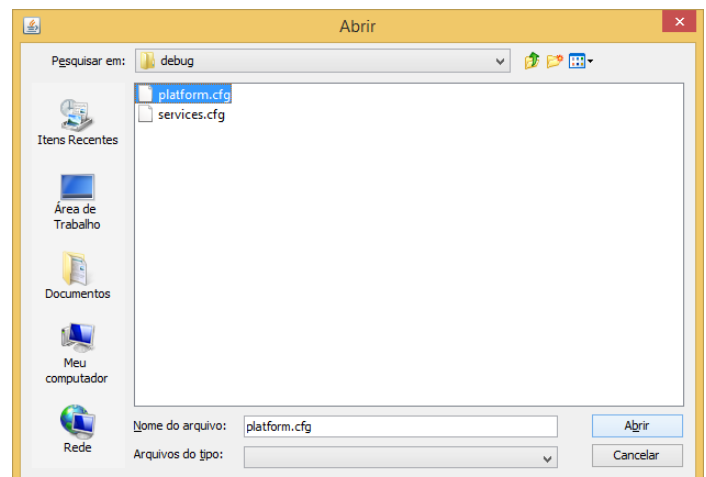
Ao abrir a ferramenta a seguinte tela é exibida:



Nesta tela além do menu superior, há os seguintes painéis de controle de uso:

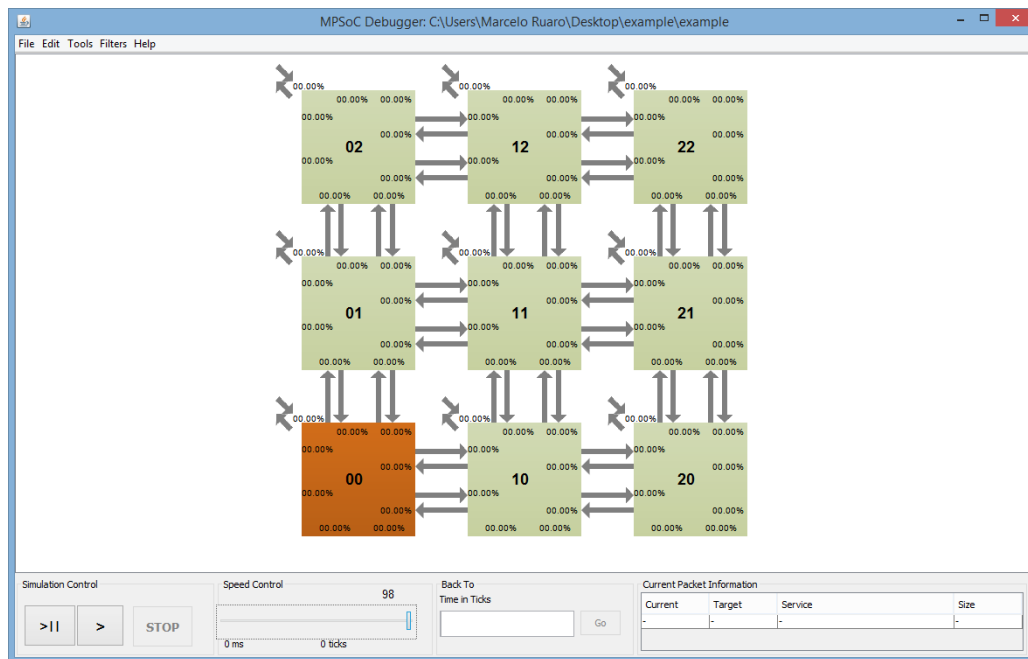
- Controle de simulação (botões passo-a-passo, executar e parar);
- Controle da velocidade de simulação (em porcentagem de 0 a 100%);
- Campo para voltar a um determinado tempo de simulação;
- Informações sobre o pacote atual: roteador atual, roteador alvo, serviço e tamanho do pacote em flits e largura de banda alocada.

Para abrir uma nova depuração é necessário ir em Menu->New Debugging (Ctrl + N) e informar o arquivo *platform.cfg* do *testcase* pelo qual se deseja depurar. Esse arquivo encontra-se no diretório *<nome_testcase>/debug*, e é criado somente após a simulação ter iniciado (GCC ou ModelSim).



Ao realizar essa etapa a ferramenta irá carregar e gerar uma visualização gráfica do MPSoC, como representado pela figura abaixo. O mestre global está representado em laranja, em azul os mestres locais e em verde os escravos.

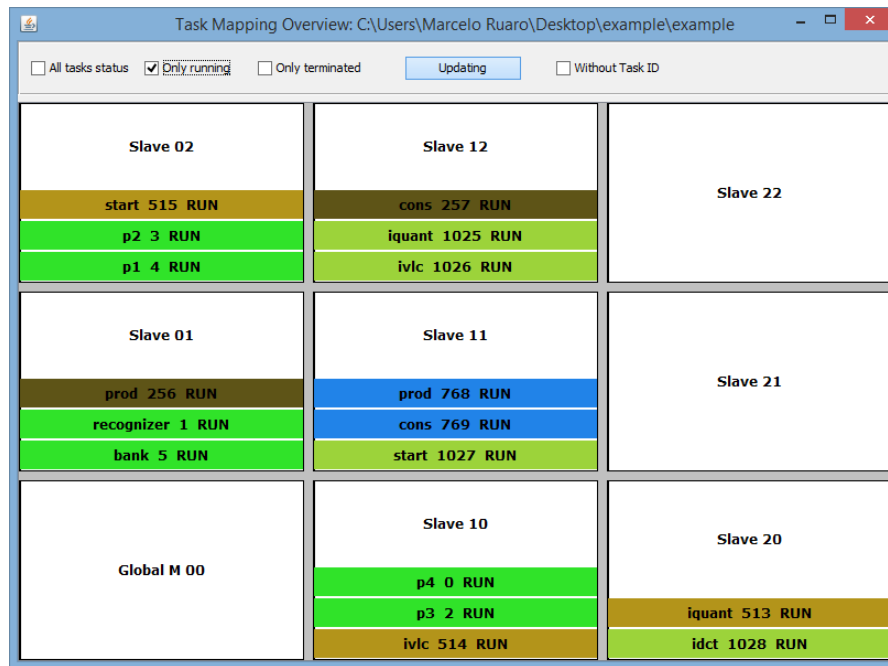
As setas que compõe a representação gráfica indica um enlace entre dois roteadores (setas maiores) ou entre um roteador e o elemento de processamento (setas menores nos cantos superior esquerdo). Como essa ferramenta foi inicialmente projetada para um MPSoC com canais duplicados, a comunicação entre roteadores é duplicada. Porém, em um MPSoC que utiliza uma NoC com canal simples, a ferramenta colorirá apenas uma seta (canal).



Com o MPSoC carregado, pode-se verificar o andamento da simulação configurando a velocidade de representação das comunicação, através da barra *speed control* e clicando no botão *play* do painel *Simulation Control*. Um valor abaixo ou igual a 98%, além de avançar a simulação, irá colorir em vermelho a seta (enlace) pelo qual o pacote está sendo enviado. Além disso, os valores representados com o símbolo de %, em cada enlace, representa a porcentagem de largura de banda utilizada para aquele enlace. Além da execução automática, pode-se optar por verificar passo-a-passo o envio de pacotes. Para isso, primeiramente a simulação deve ser parada e depois deve-se utilizar o botão “passo-a-passo” para analisar cada pacote transmitido de PE por PE.

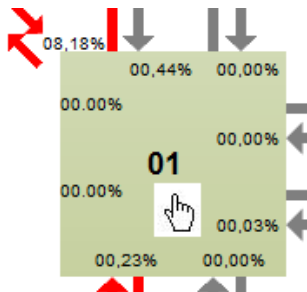
Mapeamento e Estado das Tarefas

A aba de mapeamento e estado das tarefas pode ser aberta clicando em Tools->Task Mapping Overview (Ctrl + t). A figura abaixo apresenta a tela exibida para o *testcase example*. Nesta tela, as aplicações são representadas por cores, no caso da figura pode-se perceber que há 5 aplicações em execução, o PE slave 10 executa 3 tarefas simultaneamente, o PE 20 executa 2 tarefas, e assim por diante.

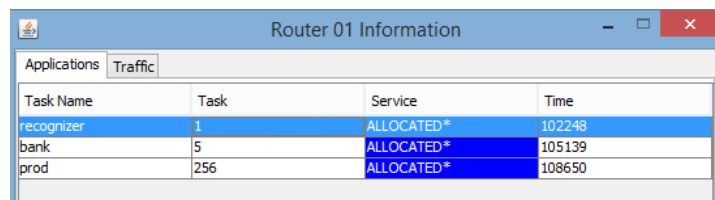


Sincronismo de Comunicação entre Tarefas

Outra funcionalidade da HeMPS Debugger é o mecanismo de verificação da sincronização entre MESSAGE_REQUEST e MESSAGE_DELIVERY, que ocorre entre um par de tarefas comunicantes. Esse recurso pode ser acessado através de um clique simples sobre um PE, na tela principal.



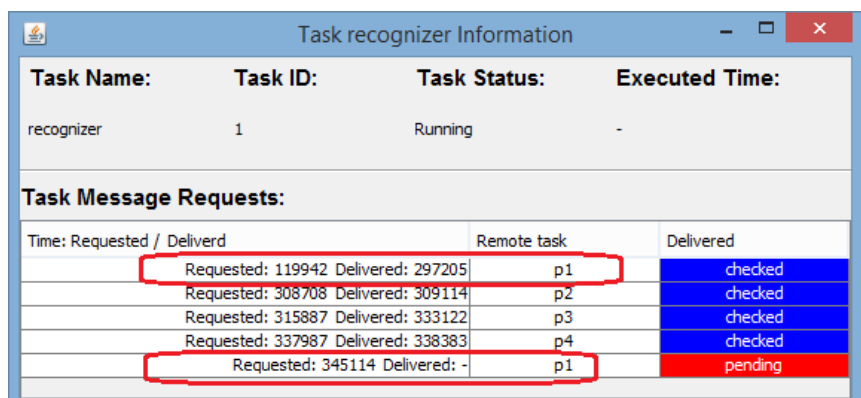
Essa ação irá abrir uma tela onde pode-se verificar em detalhes o estado de cada tarefa, bem como o volume de comunicação do PE.



Router 01 Information

Task Name	Task	Service	Time
recognizer	1	ALLOCATED*	102248
bank	5	ALLOCATED*	105139
prod	256	ALLOCATED*	108650

Ao dar duplo clique em uma tarefa que contenha o *Service* ALLOCATED, a tela de verificação de sincronismo de comunicação de tarefa (no caso acima, *recognizer*) será exibida. Nesta tela, representada pela figura abaixo, pode-se perceber que, por exemplo, a tarefa *recognizer* enviou um MESSAGE_REQUEST para a tarefa *p1* no tempo de 119.942 (medido em ciclos de *clock* ou *ticks*), e obteve o respectivo MESSAGE_DELIVERY da tarefa *p1* aos 297.205 *ticks*. Ou seja, ficou bloqueada por 177.263 *ticks*. Além disso, é possível perceber que a última linha da tabela representa o envio de um MESSAGE_REQUEST para a tarefa *p1*, mas que ainda não foi recebido. Portanto, a tarefa está em estado de WAITING aguardando a mensagem.



Task recognizer Information

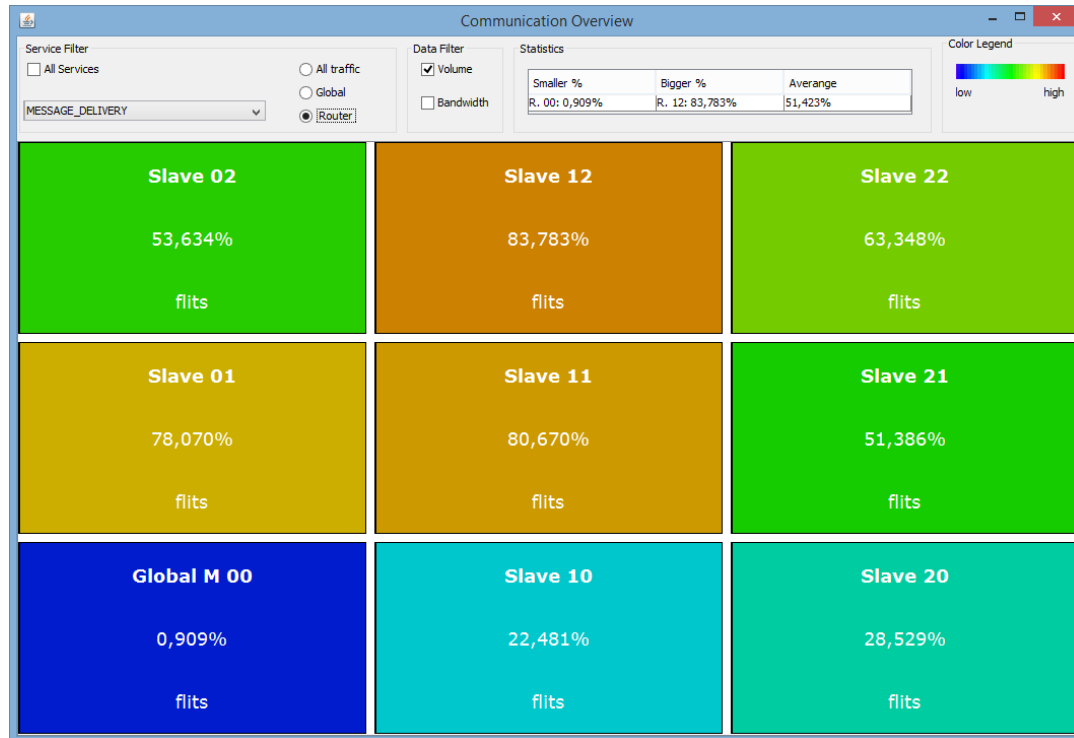
Task Name:	Task ID:	Task Status:	Executed Time:
recognizer	1	Running	-

Task Message Requests:

Time: Requested / Delivered	Remote task	Delivered
Requested: 119942 Delivered: 297205	p1	checked
Requested: 308708 Delivered: 309114	p2	checked
Requested: 315887 Delivered: 333122	p3	checked
Requested: 337987 Delivered: 338383	p4	checked
Requested: 345114 Delivered: -	p1	pending

Distribuição da Comunicação

A tela de distribuição de comunicação é exibida através da opção Tools->Communication Overview. Nesta tela é possível avaliar a distribuição de comunicação no nível de serviço entre os PEs. Diferentes cores, de acordo com a frequência de cor, representa o volume de comunicação do PE. É possível selecionar o serviço pelo qual se deseja verificar a distribuição ou também considerar todos os serviços (*All Services*). Além disso, é possível considerar três pontos de vista: (i) *All traffic*, que relaciona o serviço escolhido em relação a todo tráfego de comunicação contendo todos os serviços trafegados na rede; (ii) *Global*, que relaciona o serviço selecionado com todo o tráfego na NoC, que contenham aquele serviço; e (iii) *Router*, que relaciona o serviço selecionado com todo o tráfego de comunicação do roteador, contendo todos os serviços que foram recebidos por ele.



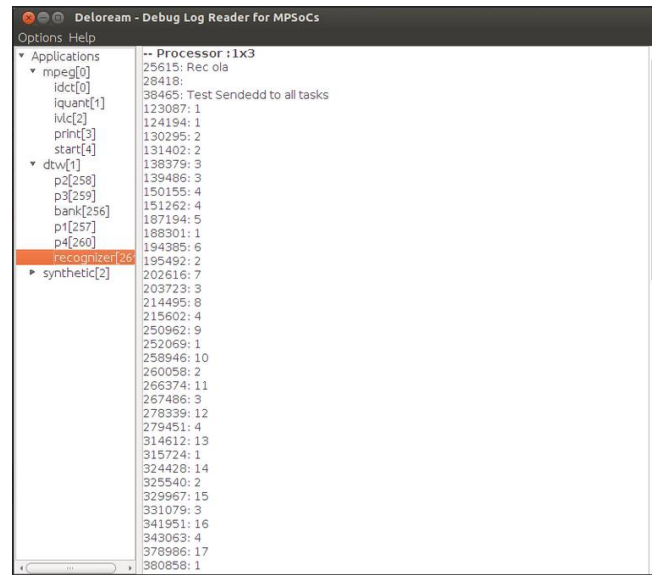
Novo Padrão de Serviços

O padrão de serviços suportado pelo MPSoC HeMPS unifica a camada de transporte da NoC. Desta maneira, os dados a serem enviados na rede são encapsulados em uma estrutura que é capaz de representar qualquer serviço da HeMPS. Essa estrutura é enviada para a DMNI que implementa a camada de rede e converte a estrutura em um pacote. Todos os serviços com seus respectivos campos estão representados na tabela abaixo.

SERVICE NAME	PACKET DIRECTION				SOURCE PE	TIMESTAMP	TRANSACTION	GENERIC FLIT 0	GENERIC FLIT 1	GENERIC FLIT 2	GENERIC FLIT 3	GENERIC FLIT 4	GENERIC FLIT 5	GENERIC FLIT 6	PAYLOAD ...
	S->S	M->S	M->M	S->M											
MESSAGE_REQUEST	X				source PE			producer task	consumer task						
MESSAGE_DELIVERY	X				source PE			producer task	consumer task	msg length					
MONITOR_RESOLUTION				X	source PE			producer task	consumer task	resolution (tick)					
UPDATE_CS_CONTROL		X			source PE			producer task	consumer task	priority					
QOS_REQUEST_SERVICE				X	source PE			producer task	consumer task	latency deadline	throughput deadline				
MONITORING_PACKET				X	source PE			producer task	consumer task	pkt latency	pkt size				
MULTICAST_MESSAGE_DELIVERY	X				source PE			producer task	consumer task (list (binary code))	msg length					
OPEN_CONNECTION_SERVICE	X				source PE			producer task	consumer task						
CLOSE_CONNECTION_SERVICE	X				source PE			NULL	consumer task						
CHANGE_FLOW_QOS		X			source PE			producer task	consumer task	priority					
TASK_MIGRATION		X			source PE			task id	NULL	allocated processor					
TIME_SLICE_INCREASE		X			source PE			task id	NULL						
MIGRATION_CODE	X				source PE			task id	NULL						
MIGRATION_DATA	X				source PE			task id	NULL	stack size	data size	code size	bus size		
UPDATE_TASK_LOCATION	X				source PE			task id	NULL	allocated processor					
TASK_DEALLOCATED (DEPRECATED)		X			source PE			task id	NULL						
LOCATION_ANSWER (DEPRECATED)		X			source PE			task id	NULL						
TASK_ALLOCATION		X			source PE			task id	master ID		data size	code size	bus size		
TASK_ALLOCATED		X			source PE			task id	NULL	allocated processor					
TASK_REQUEST			X		source PE			task id	NULL	requesting task					
TASK_TERMINATED				X	source PE			task id	master ID						
LOAN_PROCESSOR_RELEASE		X			source PE			task id	master ID	released proc					
NEW_TASK			X		source PE			task id	master ID	allocated processor	data size	code size	bus size	initial address	
APP_TERMINATED			X		source PE			app ID	cluster id	app task number					
NEW_APP			X		source PE			app ID	NULL	app descriptor size					
INITIALIZE_CLUSTER			X		source PE			NULL	cluster id						
INITIALIZE_SLAVE		X			source PE			NULL	NULL						
TASK_TERMINATED_OTHER_CLUSTER			X		source PE			task id	master ID						
LOAN_PROCESSOR_REQUEST			X		source PE			task id		p.allocated processor					
LOAN_PROCESSOR_DELIVERY			X		source PE			task id	hops	allocated processor	max free procs				

Debugger Log Reader for MPSoC (Deloream)

O Deloream é uma ferramenta de usuário que visa simplificar, em uma interface gráfica, a análise de log de cada tarefa simulada no MPSoC. A figura abaixo apresenta tal ferramenta com 3 aplicações. 2 aplicações (mpeg e dtw) estão expandidas para poder verificar o log de suas tarefas. Na figura, com duplo clique em cima da reconhecer, pode-se verificar as informações gravadas no log.



➤ Executando Via Servidor Kriti

1. Inicializar o ambiente com as variáveis do ambiente: `module load hemps`
2. Criar os diretórios de trabalho:

```
mkdir hemps_teste
cd hemps_teste
mkdir applications
```
3. No diretório `hemps_teste` deve-se criar o arquivo de configuração de plataforma (`teste.hmp`), adicionando 2 aplicações para serem executadas. Para configurar o hmp, adicione o seguinte texto ao arquivo:

```
vi teste.hmp
[project name]           WithoutLoad
teste                   [masters location]
[tasks per pe]          LB
2                       [global master]
[processor description]  0
sc                      [application]
[dimensions]            pc1
6                       [start time]
6                       1 ms
[cluster size]          [application]
3                       pc2
3                       [start time]
[mapping]               1 ms
```

Pode-se notar que foram adicionadas duas aplicações que iniciam no mesmo instante: 1ms
 Não devem ter linhas em branco neste arquivo.

4. Agora, devemos criar as duas aplicações adicionadas no arquivo de configuração.

A. No diretório /applications vamos criar 2 outros diretórios: "pc1" e "pc2"

```
cd applications
mkdir pc1
mkdir pc2
```

B. Criando a aplicação pc1 (produtor-consumido 1)

```
cd pc1
vi taskA.c
#include <task.h>
#include <stdlib.h>

Message msg;

int main(){

    int i, j,t;

    Echo("task A started.");
    Echo(itoa(GetTick()));

    for(i=0;i<10;i++){
        msg.length = 30;
        for (j=0;j<30;j++) msg.msg[j]=i;
        Send(&msg,taskB);
    }

    Echo(itoa(GetTick()));
    Echo("task A finished.");
    exit();
}

vi taskB.c
#include <task.h>
#include <stdlib.h>

Message msg;

int main(){
    int i;

    Echo("task B started.");
    Echo(itoa(GetTick()));

    for(i=0;i<10;i++){
        Receive(&msg,taskA);
    }

    Echo(itoa(GetTick()));
    Echo("task B finished.");
    exit();
}
```

C. Criando a aplicação pc2 (produtor-consumido 2): basta copiar os arquivos .c de pc1 para pc2

```
cd ../pc2/
cp ../pc1/* .
```

D. Crie também um arquivo de configuração (<nome_app>.cfg) para ambas as aplicações.

```
cd ../pc1
```

```
vi pcl.cfg
```

Dentro deste arquivo deve-se informar as tarefas iniciais, usando a tag <initialTasks>:

```
<initialTasks>  
taskA
```

O mesmo deve ser feito com a aplicação pc2:

```
cd ../pc2  
vi pc2.cfg  
<initialTasks>  
taskA
```

5. Agora que o ambiente está configurado, devemos simular o MPSoC e suas aplicações.

A. Voltar para raiz do projeto: `hemps_teste` (local onde está nosso teste.hmp)

B. Executar: `hemps teste.hmp 100`

Diversas mensagens de debug são criadas

Por fim a simulação irá começar, carregando o HeMPS Debugger para análise gráfica da simulação.