Projeto Modelagem Sistemas em Silício

Prof. Ricardo Jacobi

Semestre: 2017/01

Apresentação

O projeto da disciplina consiste em desenvolver um modelo SystemC de um sistema em silício baseado em uma NoC como arquitetura de comunicação. A estrutura geral do sistema é apresentada na figura 1.

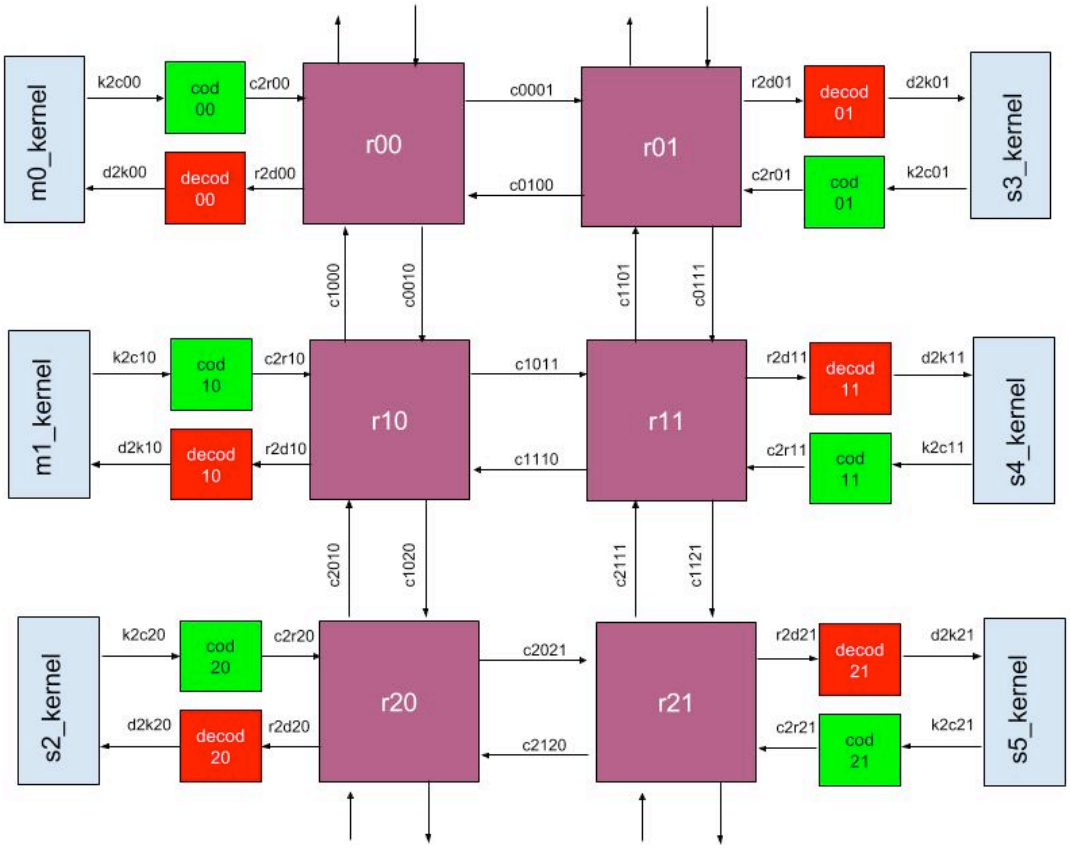


Figura 1. Estrutura do sistema.

A rede *intrachip* proposta consistem de 6 roteadores interconectados de forma matricial, aos quais são acoplados os elementos da rede (mestres e escravos).

Os módulos a serem desenvolvidos neste trabalho são:

1) NoC
    - Jessé, Javier, João e Pedro
2) MIPS + Cache
    - Felipe, Renan e Fabrícia

3) RISC V + Cache
    - Davi, Iuri, Marcos
4) Memória + DMA
    - Anderson e Arthur
5) IDEA
    - Lucas Avelino e Lucas Nascimento
6) Convolução + Interface Gráfica
    - Gildo e Manoel

Mapeamento em Memória

Os processadores incluídos no sistema acessam dispositivos de entrada e saída através de endereçamento mapeado em memória. Operações de *load* e *store* são utilizadas para escrever valores em dispositivos na rede e ler valores fornecidos por estes, respectivamente. O espaço de endereçamento é dividido segundo o modelo padrão do MIPS, que pode ser consultado no MARS, ambiente de programação *assembler* do MIPS:

- endereço inicial para mapeamento em memória: 0xFFFF0000
- endereço final para mapeamento em memória: 0xFFFFFFFF

Descrição dos Módulos

NoC

A rede *intrachip* deve prover a capacidade de comunicação entre quaisquer nós por ela interligados.

O funcionamento geral de uma NoC é apresentado nas transparências da disciplina.

A documentação e código da NoC de referência estão incluídas em arquivo no Moodle.

Os módulos da NoC devem ser endereçados a partir de um identificador. A troca de mensagens entre módulos da NoC deve ocorrer utilizando-se o identificador e comandos que serão interpretados por suas interfaces de comunicação.

Tarefas:

- estudar a NoC original
- propor alterações para flexibilizar e aumentar sua capacidade
- dar suporte aos desenvolvedores de módulos no interfaceamento com a NoC

MIPS + Cache

Implementação de uma versão pipeline do MIPS com *cache* de dados e instruções. Existe abundante documentação relativa ao processador MIPS. A versão a ser implementada é o R2000, cuja versão didática é apresentada nas disciplinas de arquitetura de computadores. O conjunto de instruções a ser implementado é:

Log – arith: AND, ANDi, NOP, NOR, NOT, OR, ORi, XOR, XORi, ADD, ADDi, LUI, SUB

Shift : SLL, SLLV, SRA, SRAV, SRL, SRLV

Teste: SLT, SLTi, SLTIU, SLTU

Branches: BEQ, BNE, BGTZ, BLTZ, J, JAL, JR

Load/Store: LB, LBU, LH, LHU, LW, SB, SH, SW

A ISA detalhada do processador pode ser encontrada no Moodle.

A *cache* a ser implementada pode ser baseada na FastMATH, que utiliza 256 blocos de 16 palavras, mapeamento direto (figura 2).

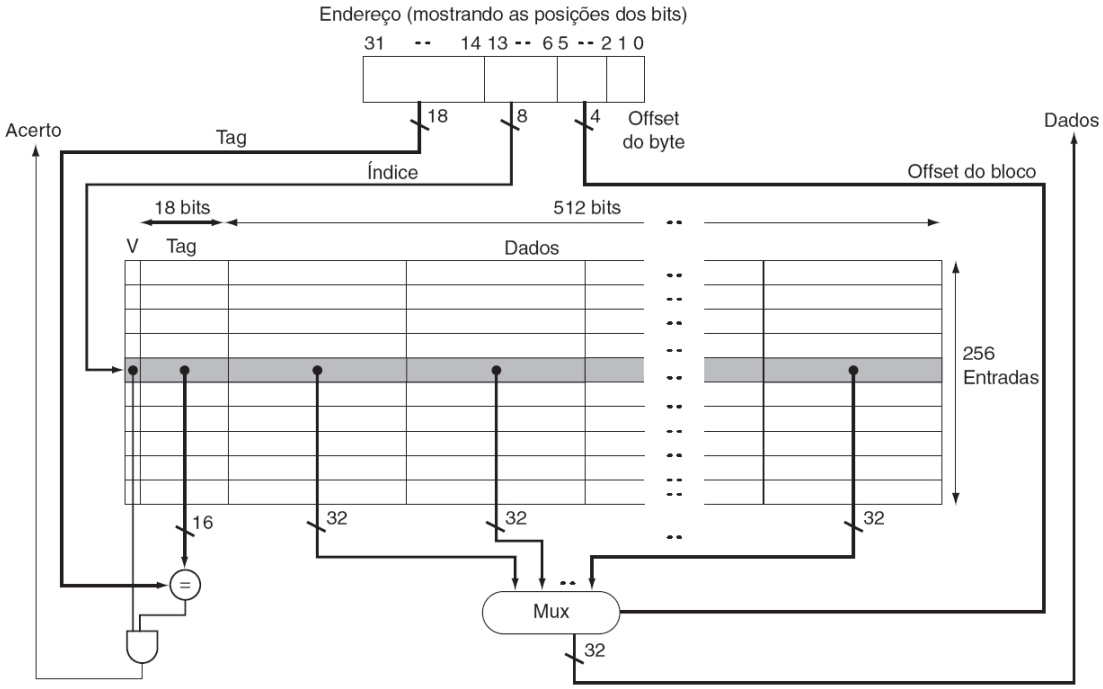


Figura 2. Cache FastMATH, 16 KiB, 256 blocos de 16 palavras.

RISC V + Cache

Implementação de uma versão pipeline do RISC V com *cache* de dados e instruções. O conjunto de instruções a ser implementado é o conjunto básico de instruções inteiras.

Excluindo as instruções de sincronização, temos:

Log – arith: ADDI, XORI, ORI, ANDI, ADD, SUB, XOR, OR, AND, LUI

Shift : SLLI, SLL, SRL, SRA, SRLI SRAI

Teste: SLTI, SLTIU, SLT, SLTU

Branches: JAL, JALR, BEQ, BNE, BLT, BGE, BLTU, BGEU, AUIPC

Load/Store: LB, LH, LW, LBU, LHU, SB, SH, SW

Em termos de cache, pode-se implementar a mesma cache indicada para o MIPS, o FastMATH.

Memória + DMA

Memória

Desenvolver um módulo de memória que suporte operações de leitura e escrita, e leitura e escrita em rajada, com tamanho de bloco configurável.

Os comandos de acesso à memória:

- READ_MEM
- WRITE_MEM
- BURST_READ_MEM
- BURST_WRITE_MEM.

DMA (*direct memory access)*
Módulo que permite a transferência de blocos de dados da memória para o disco do computador, disco para memória e memória para memória. Por exemplo, os dados para serem cifrados pelo IDEA podem ser carregados de disco. Para acionamento do DMA, é necessário indicar o endereço de memória para a tranferência dos dados e o número de palavras a serem transferidas. O nome do arquivo em disco pode ser indicado passando-se um ponteiro para um *string*, que é armazenado em um registrador interno do DMA.

O módulo de DMA é configurado pelo processador e realiza transferências de blocos de dados entre:

- memória e disco
- disco e memória
- memória e memória

Comandos do DMA:

- *DMA_LOAD:* carrega um arquivo de disco para a memória. Informações necessárias:
    - *nome do arquivo*: ponteiro para string
    - *endereço inicial da memória:* i_address
    - *comando*: deve ser escrito no endereço correspondente para iniciar a transferência do arquivo. Ao encerrar a transferência, o controlador DMA escreve no endereço correspondente o comando DMA_IDLE e salva o número de

bytes do arquivo na posição *size*, para acesso pelo processador.

- *DMA_STORE:* escreve um bloco de dados em arquivo. Tipo de arquivo criado é binário, escreve-se diretamente o conteúdo da memória. Informações*:*
  - *nome do arquivo:* ponteiro para string (será criado um arquivo. Se existir, deve sobrescrever)
  - *endereço inicial da memória:* i_address
  - *tamanho do bloco:* número de bytes a ser transferido para o arquivo. Esta informação corresponde ao dado *size*.
- *DMA_IDLE:* indica que o controlador está disponível para nova transferência.
- *DMA_MOVE:* movimentação de bloco de dados na memória. Informações necessárias:
  - *endereço inicial da memória: i_address*
  - *endereço destino da memória*: *d_address*
  - *tamanho do bloco: size*
  - *comando: DMA_MOVE* deve ser escrito pelo processador no endereço correspondente. Após a transferência do bloco o controlador coloca o DMA em modo DMA_IDLE, para indicar o final da operação.

Endereços para acesso ao DMA:

0xFFFF0000 - *size* # número de palavras a serem transferidas
0xFFFF0004 – i_address # endereço inicial de memória (*file*)
0xFFFF0008 – d_address # endereço destino (blocos de memória)
0xFFFF000C - file_name # ponteiro para *string*
0xFFFF0010 – cmd/status # comando ou estado do controlador,
conforme explicado acima.

IDEA:

Módulo para cifrar mensagens. É iniciado com uma código de 128 bits, que gera internamente 52 chaves de 16 bits, utilizadas na cifragem de blocos de 64 bits. Os blocos são subdivididos em 4 palavras de 16 bits e processados conforme o fluxo de dados indicado na figura 2.

Endereços para acesso ao IDEA:

0xFFFF0040 - cifra_out_h (32 a 63) # bloco cifrado, 32 hsb
0xFFFF0044 - cifra_out_l (0 a 31) # bloco cifrado, 32 lsb
0xFFFF0048 - cifra_in_h (32 a 63) # input, 32 hsb
0xFFFF004C - cifra_in_l (0 a 31) # input, 32 lsb
0xFFFF0050 - chave_128_3 (96 a 127) # chave - 4a palavra
0xFFFF0054 - chave_128_2 (64 a 95) # chave - 3a palavra
0xFFFF0058 - chave_128_1 (32 a 63) # chave - 2a palavra
0xFFFF005C - chave_128_0 (0 a 31) # chave - 1a palavra
0xFFFF0060 – DMA_CMD # comando

Comandos para IDEA:

- IDEA_IDLE: cifrador disponível
- IDEA_BKEYS: gerar as chaves a partir da semente de 128 bits. O módulo deve gerar as chaves diretas e inversas (decifragem).
- IDEA_CYPHER: cifrar uma palavra de 64 bits com os parâmetros fornecidos, colocando o resultado em cifra_out_h e cifra_out_l
- IDEA_DECYPHER: decifrar uma palavra de 64 bits com os parâmetros fornecidos, colocando o resultado em cifra_out_h e cifra_out_l

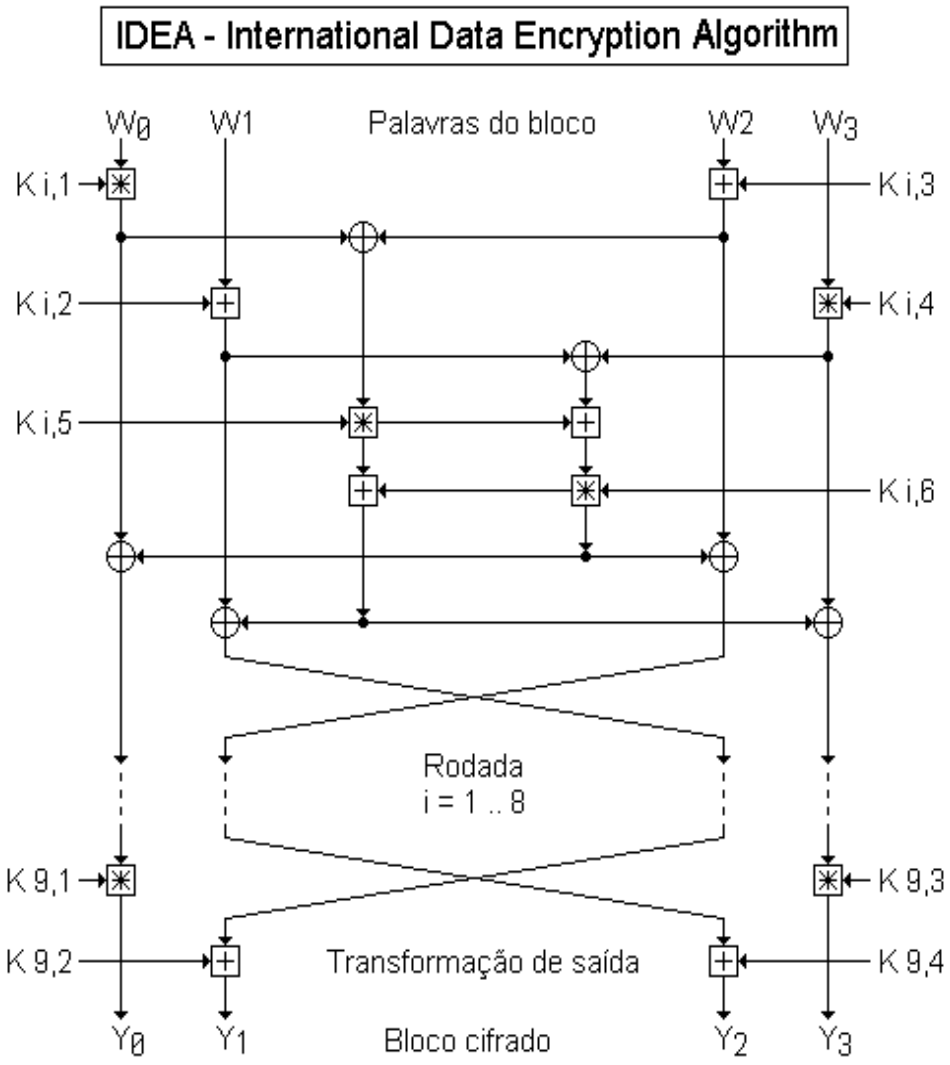


Figura 2. Fluxo de dados do IDEA.

INTERFACE GRÁFICA

A interface gráfica é um módulo que permite a execução de operações gráficas sobre uma tela criada no ambiente de simulação.

As operações serão implementadas utilizando a biblioteca OpenCV. A tela padrão sobre a qual as operações são realizadas tem resolução de 640x480 pontos.

Cada *pixel* é representado por uma palavra na memória. Imagens monocromáticas tem sua intensidade representada pelo *byte* menos significativo da palavra, representando 256 níveis de intensidade. Imagens coloridas utilizam o sistema RGB, com os 3 bytes menos significativos representando as três componentes de cor.

Cada operação recebe uma lista de parâmetros. Os parâmetros são armazenados em memória e um ponteiro para a estrutura contendo os dados é fornecido nas chamadas de operações.

A interface trabalha com coordenadas de tela, ou seja, o ponto 0,0 de um bitmap corresponde ao vértice superior esquerdo. A coordenada *x* cresce para a direita e a coordenada *y* cresce para baixo.

As operações que podem ser realizadas são:

- DRAW_IMG: exibe um bloco de pixels na tela, a partir de uma posição inicial (x, y). As dimensões *dx* e *dy* da imagem são passadas como parâmetro.
- DRAW_POINT: ativa um pixel endereçado pelas suas coordenadas (*x*, *y)* com a cor / intensidade especificada.
- DRAW_LINE: desenha uma linha entre dois pontos (*xi*, *yi*) e (*xf*, *yf*) com a cor / intensidade especificada.
- DRAW_RECT: desenha um retângulo entre dois pontos (*xi*, *yi*) e (*xf*, *yf*) com a cor / intensidade especificada.
- DRAW_CIRC: desenha um círculo centrado no ponto (*xc*, *yc*) e raio r, com a cor / intensidade especificada.
- FILL_RECT: desenha um retângulo preenchido entre dois pontos (*xi*, *yi*) e (*xf*, *yf*) com a cor / intensidade especificada.
- FILL_CIRC: desenha um círculo preenchido centrado no ponto (*xc*, *yc*) e raio r, com a cor / intensidade especificada.
- GI_IDLE: interface disponível para operações.

Endereços para acesso ao módulo de Convolução:

0xFFFF0200 – DSPL_CMD	# comando/estado da interface
0xFFFF0204 – parâmetro	# endereço dos parâmetros

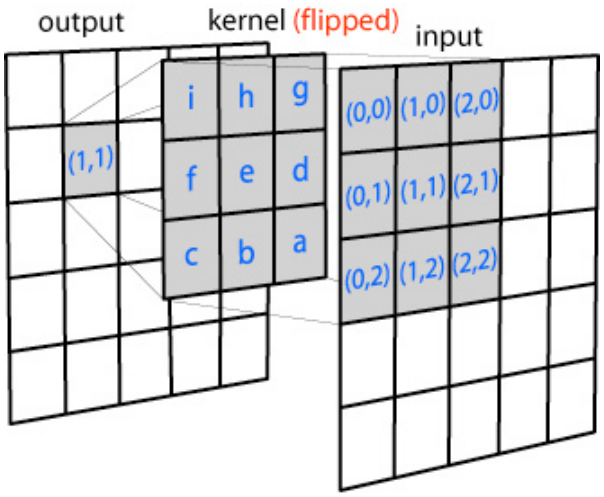
CONVOLUÇÃO DE IMAGEM

O módulo de convolução de imagens aplica um elemento estruturante (*kernel*) de 3x3 coeficientes sobre uma imagem armazenada em memória. Para executar sua função, é necessário definir o elemento estruturante, as dimensões vertical e horizontal da imagem e o endereço do pixel superior esquerdo.

Cada *pixel* é representado por um *byte*, definindo-se 256 níveis de cinza.

Para fins de processamento das bordas, se supõe que a imagem já contenha uma moldura extra de um *pixel*.

A figura a seguir ilustra o procedimento de convolução.



Endereços para acesso ao módulo de Convolução:

0xFFFF0100 – kernel address	# end kernel na memória
0xFFFF0104 – resolução horizontal	# DX da imagem
0xFFFF0108 – resolução vertical	# DY da imagem
0xFFFF010C – endereço imagem	# end pixel(0,0) na memória
0xFFFF0110 – endereço destino img	# end pixel(0,0) na memória
0xFFFF0114 – CONV_CMD	# comando

Comandos para Convolução:

- CNV_SET_KERNEL: instrui o módulo a carregar um novo conjunto de coeficientes para o *kernel*
- CNV_IDLE: módulo disponível, resultado pronto
- CNV_START: inicia a convolução. Quando encerrar, o estado do módulo é automaticamente configurado como CNV_IDLE