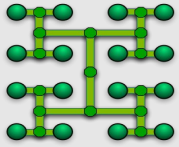
Universidade de Brasília

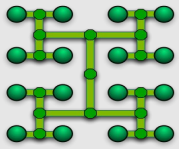
Portas

Ricardo Jacobi
Departamento de Ciência da Computação
Universidade de Brasília
jacobi@unb.br



LAICO

Universidade de Brasília



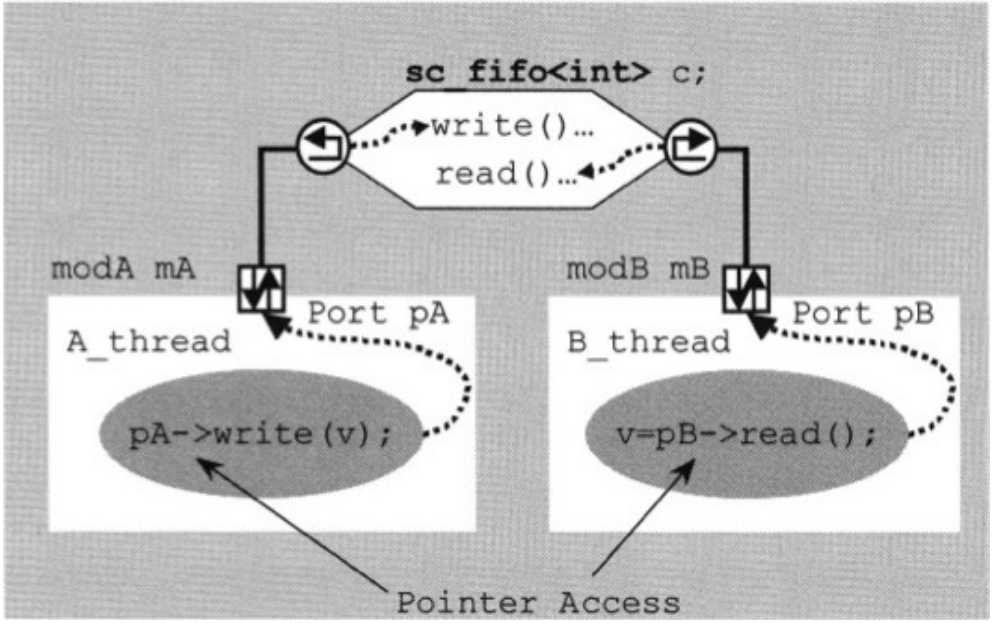
LAICO

Introdução

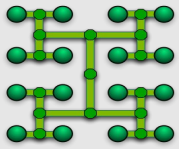
- Componentes de hardware comunicam-se com o meio externo através de portas
- Diferentemente do software, em hardware as portas tem uma interpretação física
  - envolvem aspectos elétricos de sinais
  - tem direção definida:
    - entrada, saída ou ambos
- SystemC oferece a recursos de comunicação entre módulos através de Portas e Canais
  - portas representam os pinos de entrada e saída
  - canais o meio de comunicação (“fios”)

Portas

- Em SystemC, portas pode ser consideradas basicamente como ***ponteiros*** para canais
- Através da porta, um processo chama procedimentos de comunicação implementados nos canais
- Ex:



Universidade de Brasília



LAICO

Interfaces

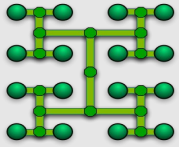
- C++ permite a definição de classes abstratas
- Uma classe abstrata não é utilizada para criar objetos, mas serve como modelo para a criação de classes derivadas
  - classes abstratas contém funções virtuais puras, que são declaradas mas não definidas na classe abstrata
  - funções virtuais puras requerem implementação nas classes derivadas para viabilizar a criação de objetos
  - nesse contexto, uma classe abstrata com funções virtuais puras define um interface a ser implementada pelas classes derivadas

Interfaces...

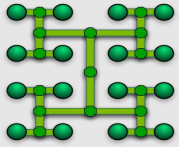
- Exemplo de classe abstrata:

```
struct base_interface {
   virtual void write (unsigned addr, int data) = 0;
   virtual int read(unsigned addr) = 0;
};
```

  - funções virtuais puras contém a palavra chave *virtual* e são atribuídas a zero: “= 0”
  - subclasses que herdam `my_interface` são obrigadas a implementar as funções virtuais puras
  - garante-se que todas as classes derivadas implementam essa interface



Universidade de Brasília



LAICO

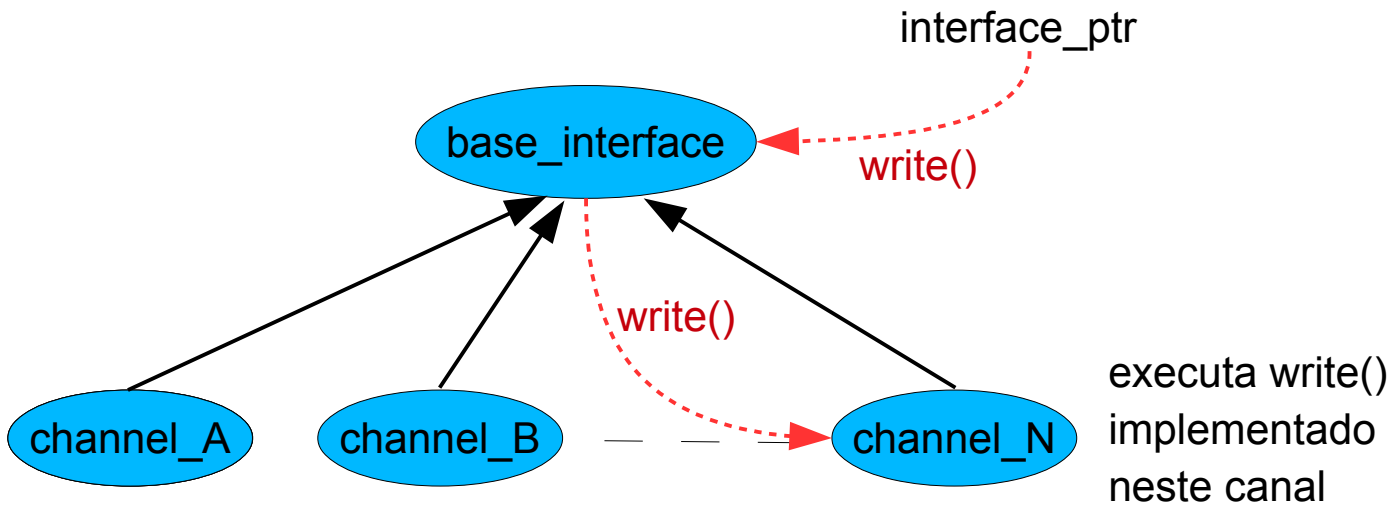
Interfaces...

- Classes derivadas definem suas próprias implementações das funções virtuais

```
class channel_A : public base_interface {
   void write (unsigned addr, int data) {
      ...
   }
   int read(unsigned addr) {
      ...
  }
}; //endclass
```

Interfaces...

- A partir de um ponteiro para a interface, é possível chamar diferentes implementações da função virtual nas classes derivadas



Interfaces...

- Uma **interface** em SystemC é uma classe abstrata que deriva de *sc_interface* e define métodos virtuais puros a serem implementados por canais e chamados através de portas
- Um **canal** em SystemC é uma classe que implementa uma ou mais interfaces e deriva tanto de *sc_channel* ou *sc_prim_channel*

Definindo Portas

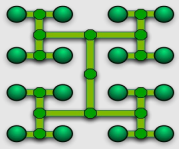
- Uma porta em SystemC é uma classe parametrizada por templates
- Declaração:

```
sc_port<interface> portname;
```

- Portas são sempre definidas dentro de módulos:

```
SC_MODULE (buffer) {
  sc_port<sc_signal_in_if<bool> > d_in;
  sc_port<sc_signal_inout_if<bool> > d_out;
...
}
```

Universidade de Brasília



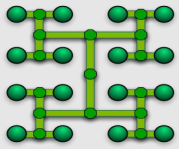
LAICO

Exemplo sc_*signal_in_if*

- sc_signal<T> é um canal primitivo que pode ser conectado a portas que implementam a interface *sc_signal_in_if*, *sc_signal_out_if* e *sc_signal_inout_if*
- Ex:

```
template <class T >  class sc_signal_in_if 
: virtual public sc_interface {
      public: // get the value changed event
        virtual const sc_event& value_changed_event() const = 0;
        // ler o valor atual
        virtual const T& read() const = 0;
}
```

Universidade de Brasília



LAICO

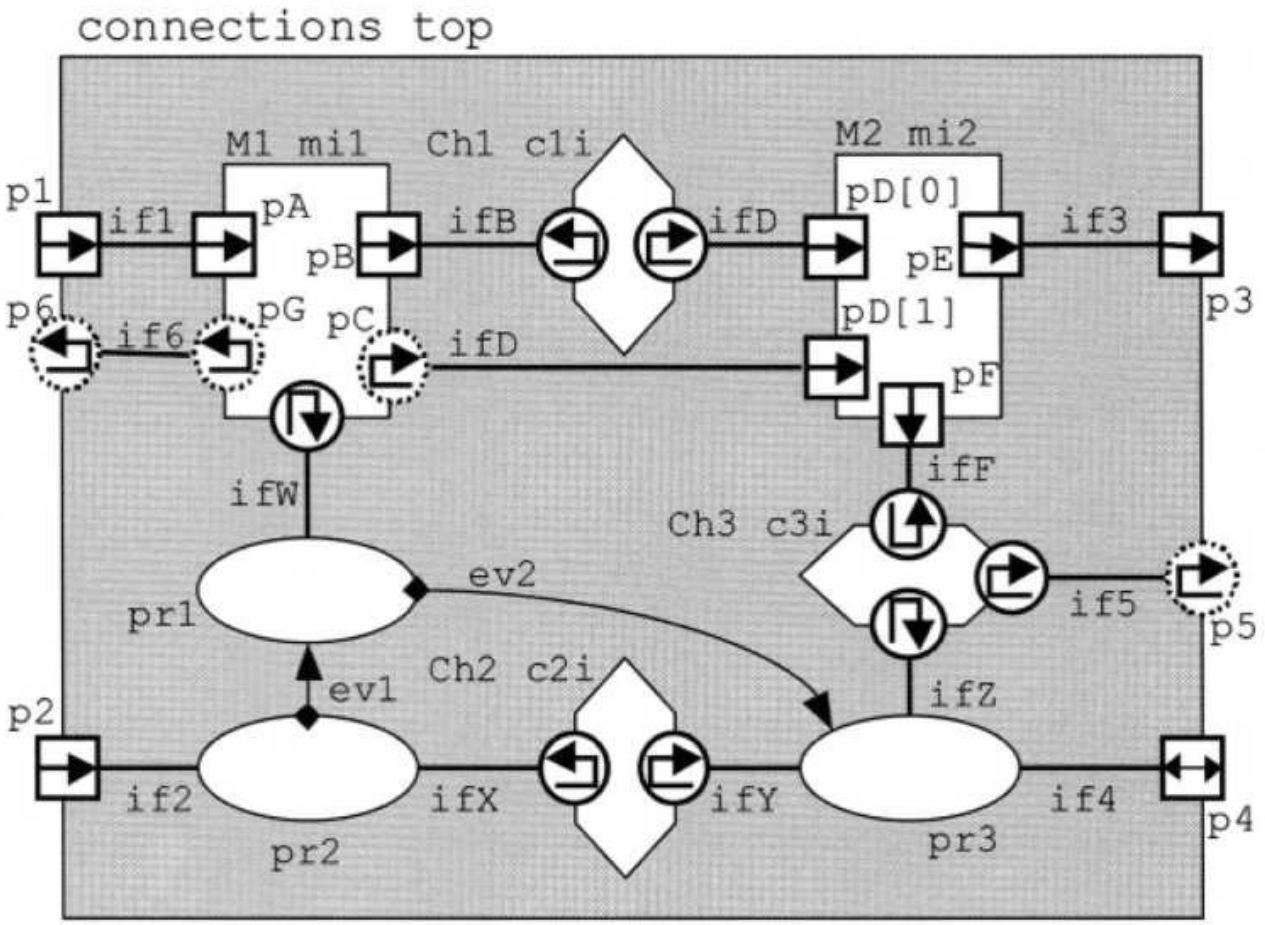
sc_signal<T>

```
template <class T> class sc_signal : public sc_signal_inout_if,  public
    sc_prim_channel {
public: // obtem o evento padrao
     virtual const sc_event& default_event() const {
          return m_valued_changed_event;
     }
     virtual const sc_event& valued_changed_event() const {
          return m_valued_changed_event;
     }
     virtual const T& read() const { return m_cur_val; }
     virtual void write(const T& value) {
          m_new_val = value;
          if ( m_new_val != m_cur_val) request_update();
      }
```

sc_signal<T>...

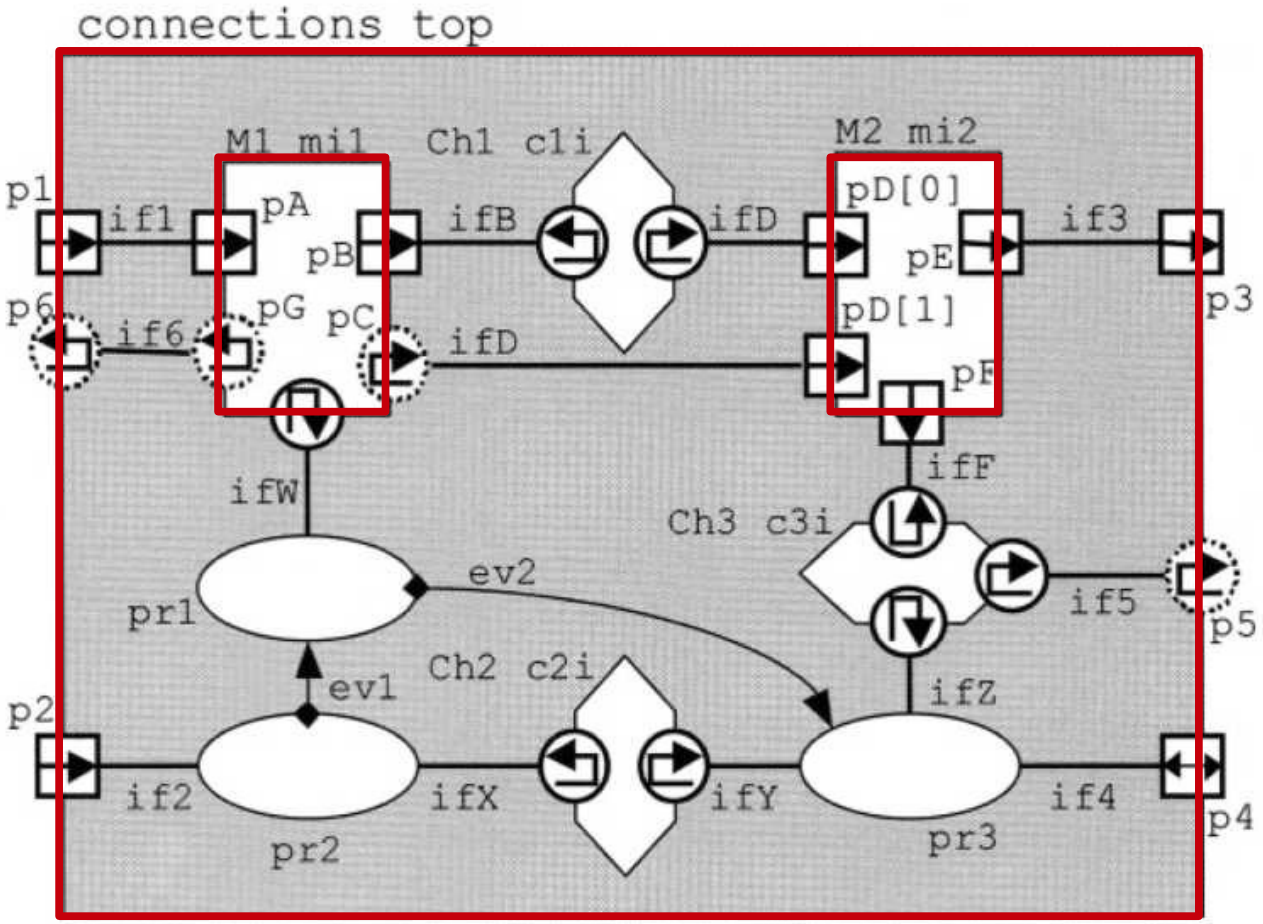
```
protected:
  virtual void update() {
     if ( m_cur_val != m_new_val ) {
          m_cur_val = m_new_val;
          m_value_changed_event.notify(SC_ZERO_TIME);
     }
  }

   T m_cur_val;
   T m_new_val;
   sc_event m_value_changed_event;
}
```

Conexões


connections top
M1 mi1
Ch1 c1i
M2 mi2
p1
if1
pA
pB
ifB
ifD
pD[0]
pE
if3
p3
p6
if6
pG
pC
ifD
pD[1]
pF
ifW
ifF
Ch3 c3i
ev2
pr1
if5
p5
ev1
Ch2 c2i
ifZ
p2
if2
ifX
ifY
if4
p4
pr2
pr3

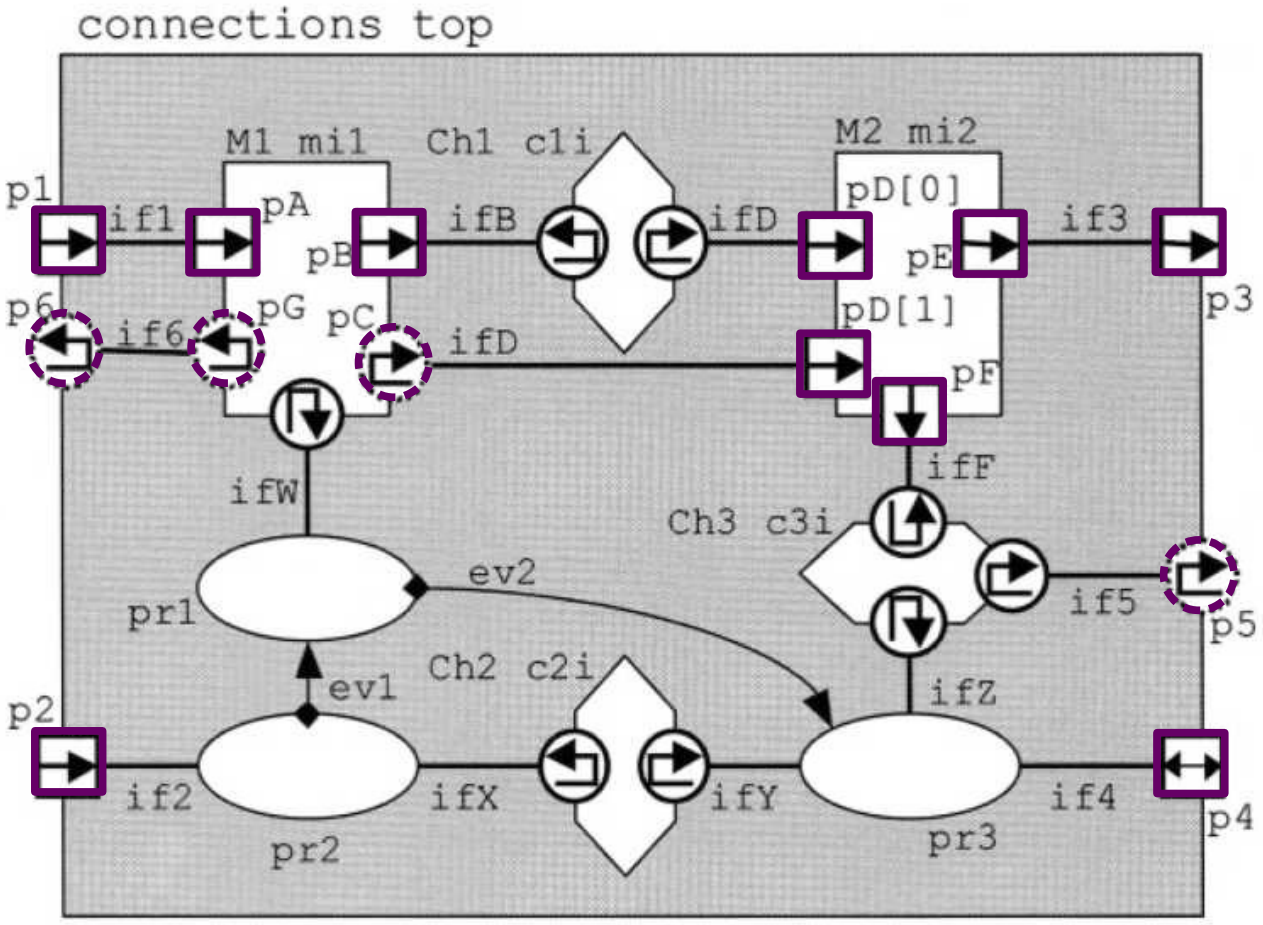
Conexões...

Módulos



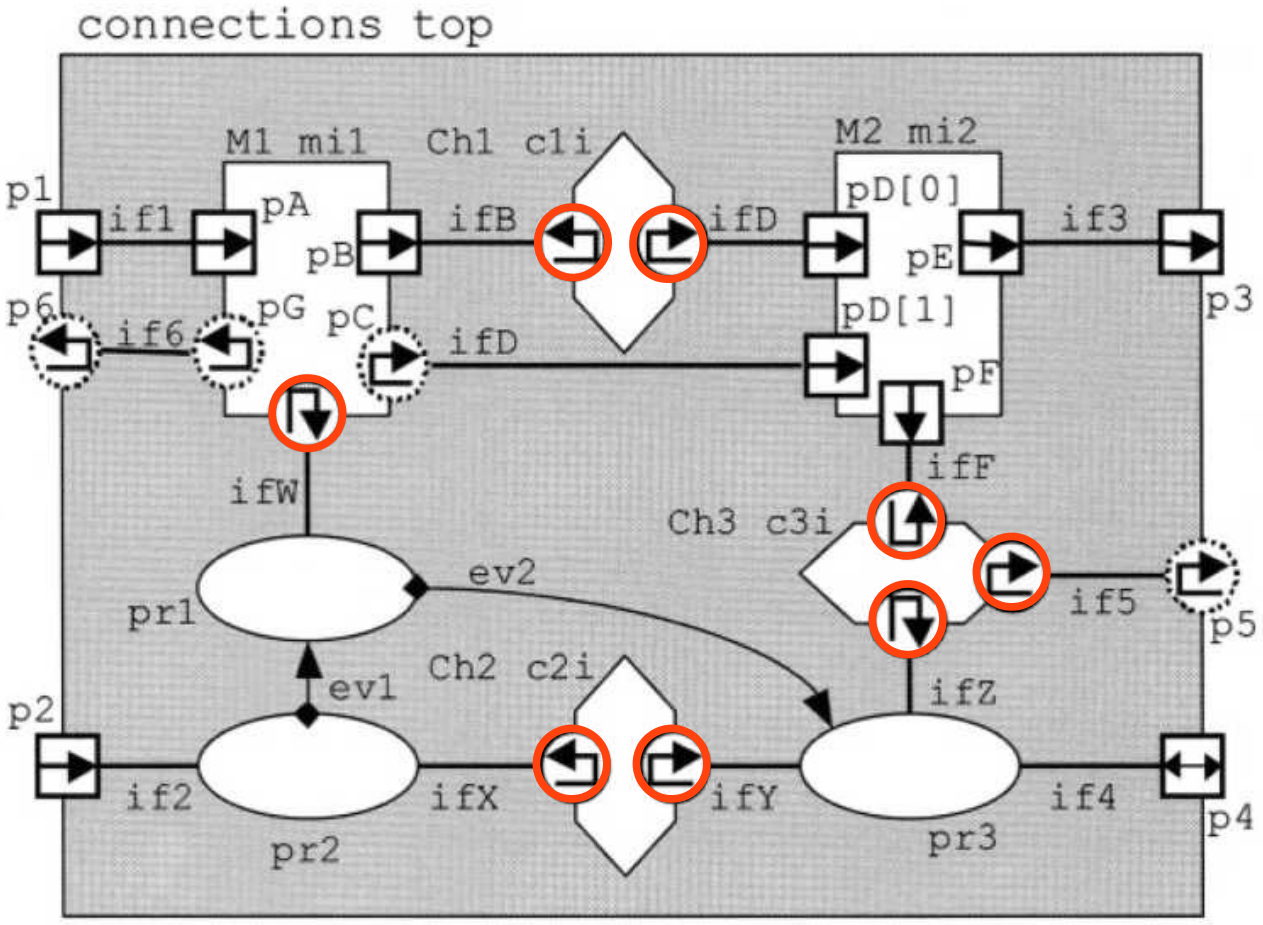
Conexões...

Portas



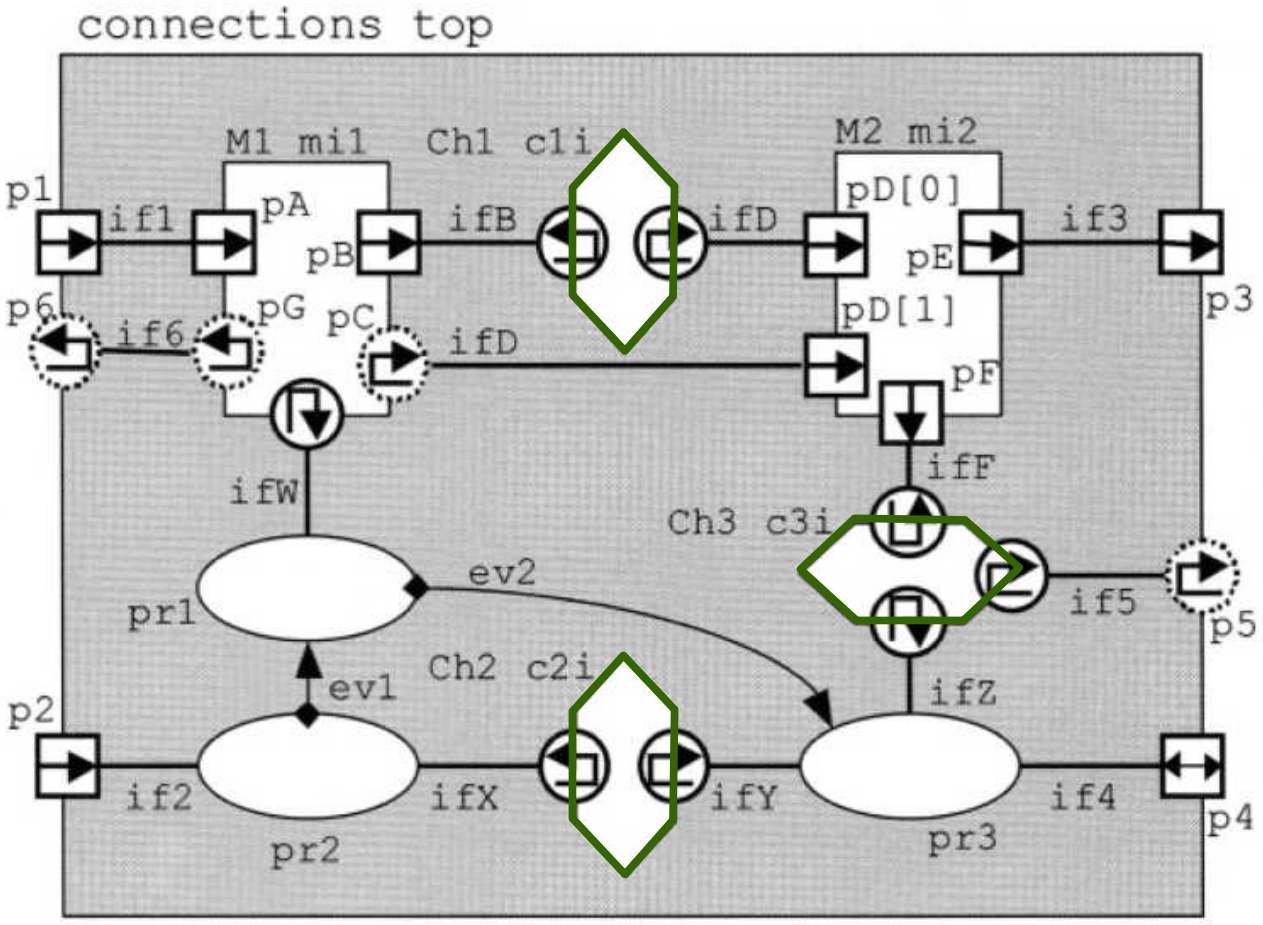
Conexões...

Interfaces

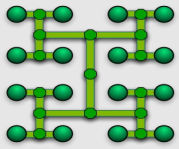


Conexões...

Canais



Universidade de Brasília

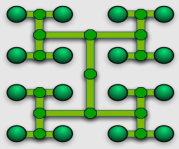


LAICO

Conexões...

- Processos no mesmo módulo podem se comunicar via canais ou sincronizar por eventos
- Processos se comunicam com outros módulos através de portas
- Cada canal pode implementar uma ou mais interface
- Módulos podem implementar interfaces
- Podemos ter arranjos de portas
- **sc_export** é um outro tipo de porta, introduzida na versão 2.1

Universidade de Brasília

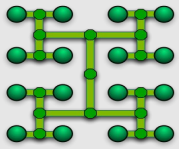


LAICO

Conexões...

Origem	Destino	Método
Porta	Submódulo	Direto via sc_port
Processo	Porta	Acesso direto à porta pelo processo. Usar o operador redefinido “->” para a porta, para lembrar que ela age como um ponteiro para interface
Submódulo	Submódulo	Através de canal local
Processo	Submódulo	- canal local - sc_export - interface implementada pelo submódulo
Processo	Processo	Eventos ou canais locais
Porta	Canal local	Conexão direta via sc_export

Universidade de Brasília



LAICO

sc_fifo_in_if

```
// Definition of sc_fifo<T> input interface
template <class T>
struct sc_fifo_in_if: virtual public sc_interface {
    virtual void read( T& ) = 0;
    virtual T read() = 0;
    virtual bool nb_read( T& ) = 0;
    virtual int  num_available() const = 0;
    virtual const sc_event&
                data_written_event() const = 0;
};
```

Vetor de Canais

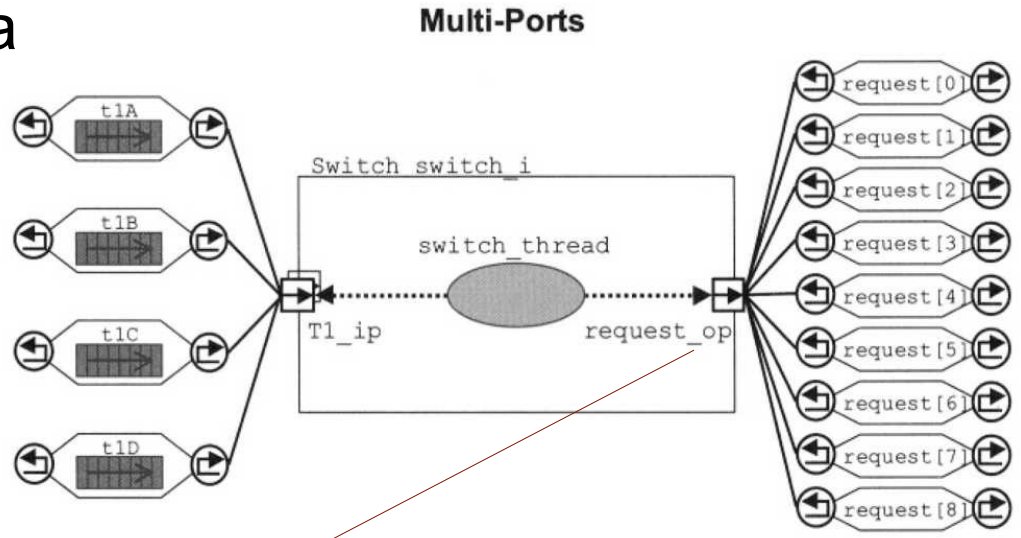
- A classe sc_port inclui dois parâmetros:

  sc_port <*interface[,N]*> *portname;*

  // N = 0 ... MAX, default = 1

  – N indica o número de canais que pode ser conectados à porta

  – Ex:



declaração

```
//FILE: Switch.h

SC_MODULE(Switch) {
   sc_port<sc_fifo_in_if<int>, 4> T1_ip;
   sc_port<sc_signal_out_if<bool>,0> request_op;
};
```

declaração

Vetor de Canais...

- Conectando canais à porta:

```
SC_MODULE (Board) {
    ...
    Switch switch_i;
    sc_fifo<int> t1A, t1B, t1C, t1D;
    sc_signal<bool> request[9];
    ...
    SC_CTOR(Board): switch_i("switch_i") {
        switch_i.T1_ip(t1A); switch_i.T1_ip(t1B);
        switch_i.T1_ip(t1C); switch_i.T1_ip(t1D);
        for (int i=0; i<9; i++)
            switch_i.request_op(request[i]);
    }
}
```

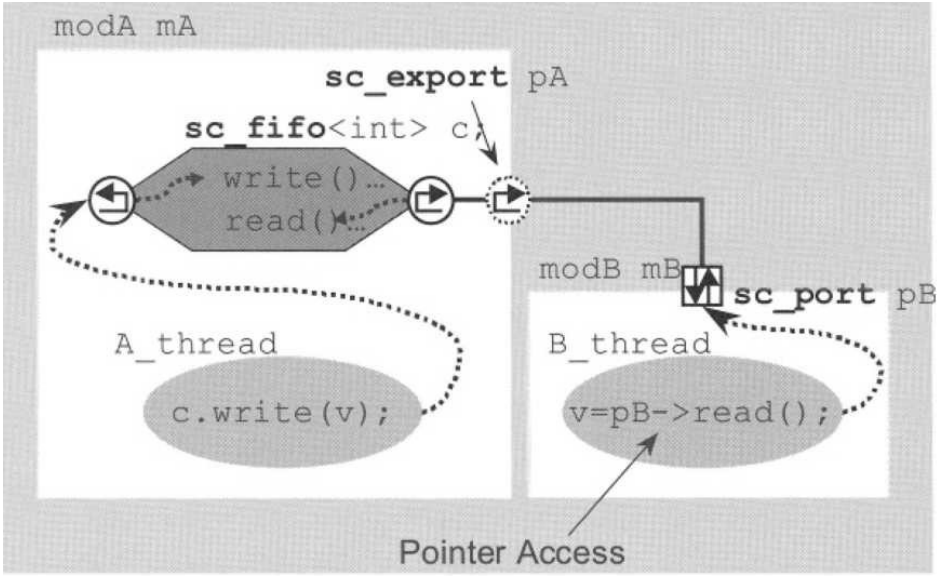
Vetor de Canais...

- Utilizando os canais:

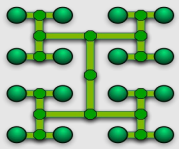
```
void Switch::switch_thread() { // Initialize requests
  for (unsigned i=0; i!=request_op.size(); i++) {
        request op[i] ->write(true);
  }//endfor - Startup after first port is activated
   wait (    T1_ip[0]->data_written_event() |
             T1_ip[1]->data_written_event() |
             T1_ip[2]->data_written_event() |
             T1_ip[3]->data_written_event());
   for(;;) {
        for (unsigned i=0;i!=T1_ip.size();i++) {
            // Process each port...
             int value = T1_ip[i]->read();
        }//endfor
    }//endforever
}//end Switch::switch_thread
```

sc_export

- É similar a uma porta, que recebe uma interface como template
- Entretanto, está associada a um canal interno ao módulo
  - É uma forma de exportar um canal interno



Universidade de Brasília



LAICO

sc_export

- Sintaxe
  - declaração:

```
sc_export<interface> portname;
```

  - definição no módulo:

```
SC_MODULE(modulename)  {
    sc_export<interface> portname;
    channel cinstance;
    SC_CTOR(modulename)  {
        portname (cinstance);
    }
} ;
```

sc_export

- Exemplo divisor de frequências

```
SC_MODULE(freq_div) {
  sc_in<bool> clk;
  sc_export<sc_signal<bool> > clk_d2, clk_d4, clk_d8, clk_d16;

SC_CTOR(freq_div) {
        // conectar os canais internos as portas sc_export
        clk_d2(clk2); clk_d4(clk4); clk_d8(clk8); clk_d16(clk16);

        SC_METHOD(div2);                    // divide por 2
          sensitive << clk.pos();
        SC_METHOD(div4);                    // divide por 4
          sensitive << clk2.posedge_event();
        SC_METHOD(div8);                    // divide por 8
          sensitive << clk4.posedge_event();
        SC_METHOD(div16);                   // divide por 16
          sensitive << clk8.posedge_event();
}
```

sc_export

```
        void div2();
        void div4();
        void div8();
        void div16();

        private:
            sc_signal<bool> clk2, clk4, clk8, clk16;
};

void freq_div::div2() { clk2 = !clk2; }

void freq_div::div4() { clk4 = !clk4; }

void freq_div::div8() { clk8 = !clk8; }

void freq_div::div16() { clk16 = !clk16; }

#endif
```

Revisando Conexões