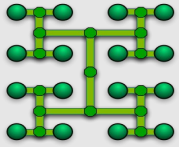
Universidade de Brasília

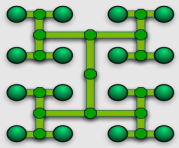
Refinando Canais

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



LAICO

Universidade de Brasília



LAICO

Projeto de Interfaces

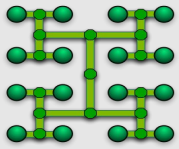
- Objetivo: minimizar o esforço no projeto do sistema, simplificando os passos seguintes de refinamento dos canais de comunicação
- Princípios:
  - Minimizar o número de interfaces distintas simplifica o reuso do projeto
  - Ex: fifos com políticas distintas
    - sc_fifo implementa métodos read() e write()
    - implementa acesso bloqueante
    - uma política alternativa seria descartar escritas quando a fila estiver cheia: sc_fifo_descarta
    - implementando a mesma interface, a substituição é direta

Projeto de Interfaces...

- Organizar as interfaces em camadas, começando com interfaces mais genéricas e derivando-se interfaces mais específicas
  - aumenta as chances de reuso de canais
  - interfaces mais especializadas herdam das mais gerais
- Exemplo:
  - interface de barramento

```
class simple_rw_if : virtual public sc_inteface  {
public:
    virtual void read (unsigned addr, char *data) = 0;
    virtual void write(unsigned addr, char *data) = 0;
};
```

Universidade de Brasília



LAICO

Projeto de Interfaces...

- Supondo a existência de transações em rajadas (*burst*):

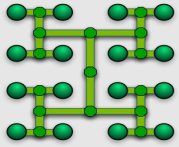
```
class burst_rw_if : public simple_rw_if  {
public:
  virtual void burst_read(unsigned addr, char *data, unsigned n) = 0;
  virtual void burst_write(unsigned addr, char *data, unsigned n) = 0;
};
```

- Um módulo que não necessita rajadas:

```
class simple_module : public sc_module  {
            public:    sc_port<simple_rw_if> rw_port;
};
```

  - C++ garante que métodos com rajada não podem ser chamados

Universidade de Brasília



LAICO

Projeto de Interfaces...

- Fatorar métodos comuns através de herança para reduzir duplicação de código
- Ex:
  - supor que ***simple_rw_if*** não seja usado diretamente
  - supor outra interface ***complex_rw_if*** usa mesmos métodos de acesso
  - interessa neste contexto manter os métodos *read()* e *write()* básicos em ***simple_rw_if*** e herdá-los de ***complex_rw_if*** e ***burst_rw_if***
  - métodos são definidos apenas uma vez

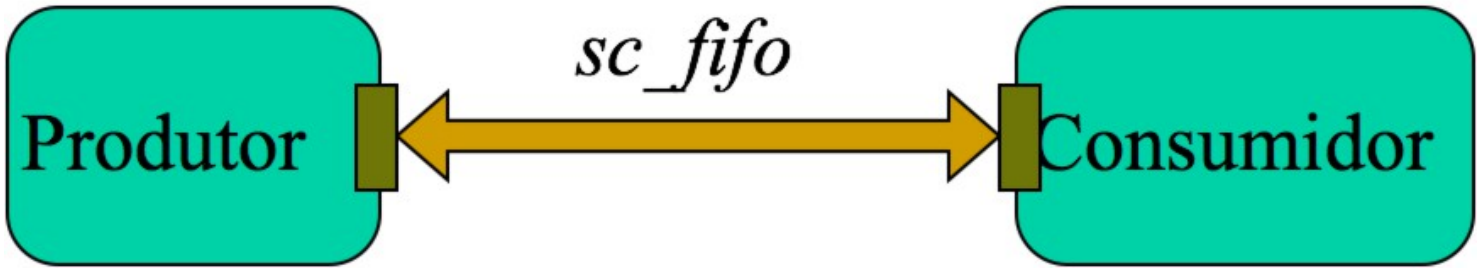
Projeto de Interfaces...

- Criar interfaces unificadas com multiplas funcionalidades a partir de interfaces mais simples através de múltipla herança:
- Ex:

```
class unified_if
        : public first_if,
        : public second_if  {
}
```

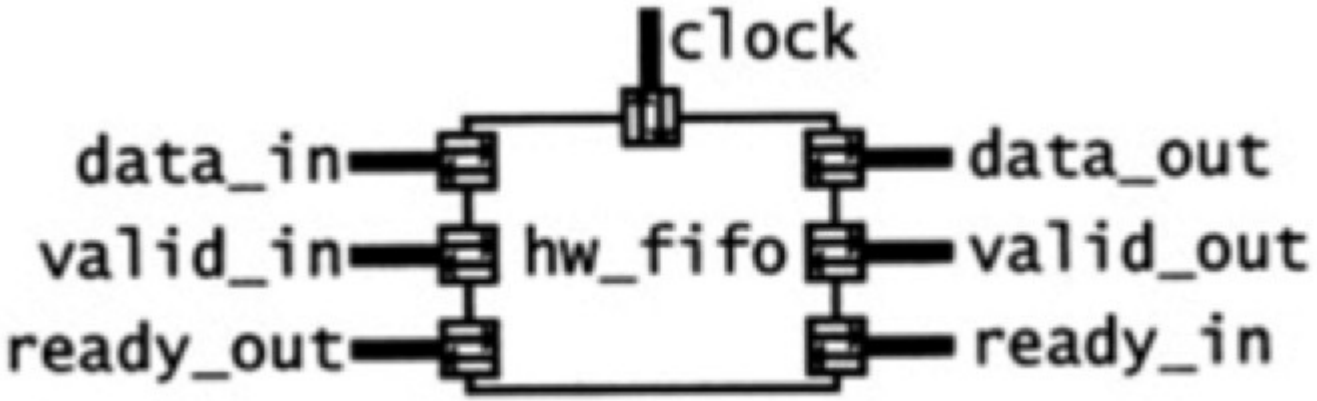
Refinamento de Canais

- Sistema Produtor – Consumidor
  - Interconectado por uma canal tipo *sc_fifo*



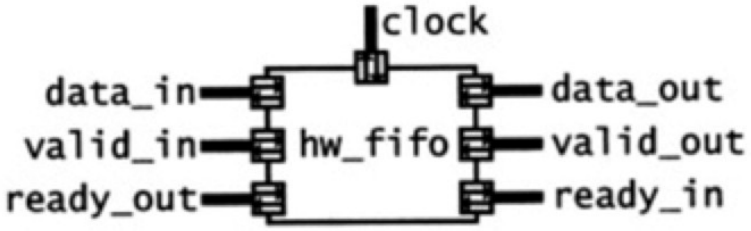
Refinando sc_fifo: hw_fifo

- Suponha que se deseje refinar a implementação da fila para o nível RTL
- *hw_fifo* é uma fila implementada com um arranjo circular, sinais de entrada e saída e clock



hw_fifo...

- escrita:
  - sempre que o sinal *ready_out* for *true*, a fila lê um dado colocado em *data_in* se *valid_in* for *true* na transição de subida do relógio
- leitura:
  - sempre que o sinal *valid_out* for true, um dado foi lido da fila se *ready_in* for true na subida do relógio

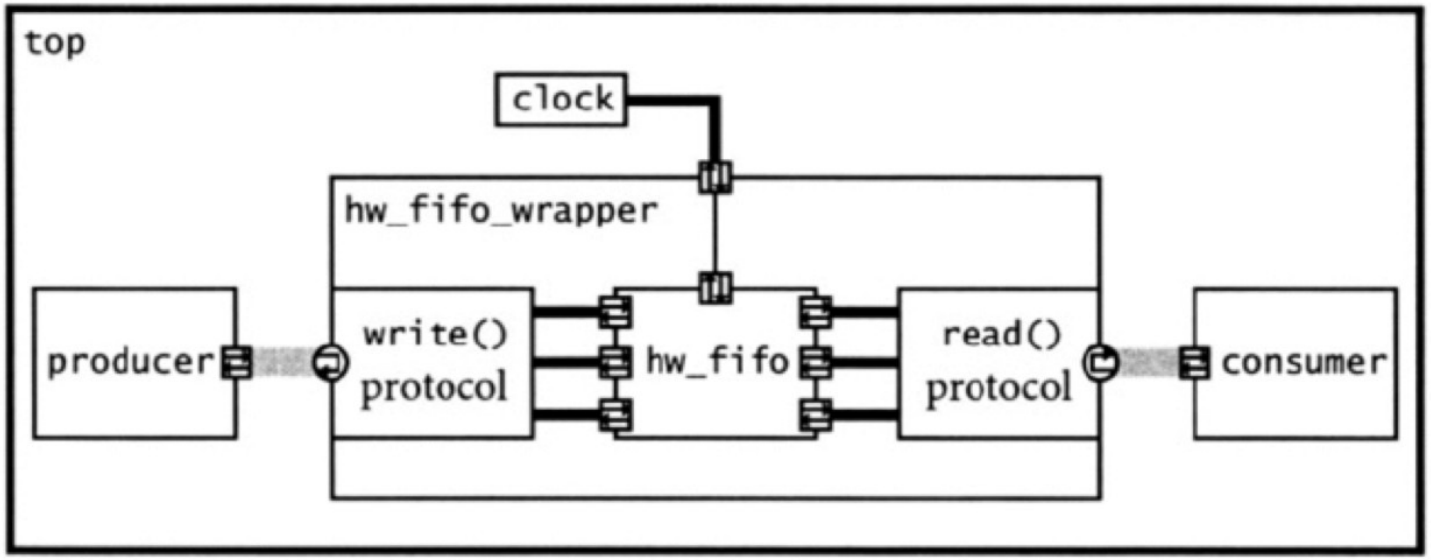


hw_fifo...

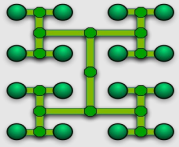
- *hw_fifo* não pode ser utilizada diretamente para interconectar o produtor com o consumidor, visto que não implementa a interface requerida pelas portas dos módulos
- para utilizar *hw_fifo* é necessário desenvolver um módulo de casamento de interface, que implemente os métodos de comunicação com produtor e consumidor de um lado, e a interface de handshake de sinais do outro
  - estes módulos são chamados wrappers (embrulhos)

hw_fifo_wrapper

- *hw_fifo_wrapper* é um canal hierárquico que contém o módulo *hw_fifo* e implementa as interfaces *sc_fifo_in_if* e *sc_fifo_out_if*



Universidade de Brasília

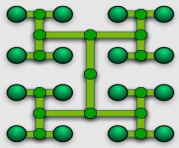


LAICO

hw_fifo_wrapper...

```
#include "systemc.h"
#include "hw_fifo.h"
template <class T> class hw_fifo_wrapper
:       public sc_module,
        public sc_fifo_in_if<T>,
        public sc_fifo_out_if<T> {
 public:
  sc_in<bool> clk;
 protected:
  // canais embutidos
  sc_signal<char>         write_data;
  sc_signal<bool>         write_valid;
  sc_signal<bool>         write_ready;

  sc_signal<char>         read_data;
  sc_signal<bool>         read_valid;
  sc_signal<bool>         read_ready;
  // componente embutido
  hw_fifo<T>              hw_fifo;
```

Universidade de Brasília

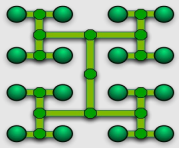


LAICO

hw_fifo_wrapper...

```
public:
  hw_fifo_wrapper(sc_module_name n, unsigned size)
          // elaboração do canal
  : sc_module(n), hw_fifo("hw_fifo", size) {

      hw_fifo.clk(clk);

      hw_fifo.data_in  (write_data);
      hw_fifo.valid_in (write_valid);
      hw_fifo.ready_out(write_ready);

      hw_fifo.data_out (read_data);
      hw_fifo.valid_out(read_valid);
      hw_fifo.ready_in (read_ready);
  }
```

Universidade de Brasília

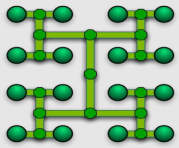


LAICO

hw_fifo_wrapper...

```
// metodo que escreve um dado na fila de hardware

virtual void write(const char& data) {

        write_data = data;
        write_valid = true;

        do {
                wait(clk->posedge_event());
        } while (write_ready != true);

        write_valid = false;
}
```

Universidade de Brasília

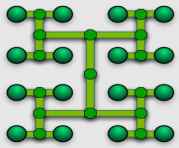


LAICO

hw_fifo_wrapper...

```
// metodo que le um dado da fila de hardware

 virtual char read() {

        read_ready = true;
        do {
                wait(clk->posedge_event());
        } while (read_valid != true);
        read_ready = false;
        return read_data.read();
  }
```

Universidade de Brasília



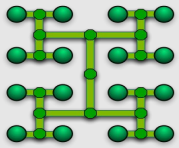
LAICO

hw_fifo_wrapper...

- Métodos virtuais puros a serem redefinidos:

```
virtual void read(T& d) { d = read(); }

virtual bool nb_read(T&) { assert(0); return false; }

virtual bool nb_write(const T&) { assert(0); return false; }

virtual int num_available() const { assert(0); return 0; }

virtual int num_free() const { assert(0); return 0; }

virtual const sc_event& data_written_event() const {
      assert (0); return e;
}
virtual const sc_event& data_read_event() const {
      assert (0); return e;
}
```

Universidade de Brasília



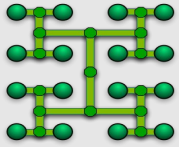
LAICO

hw_fifo_wrapper...

- Métodos virtuais puros a serem redefinidos:

```
virtual void read(T& d) { d = read(); }

virtual bool nb_read(T&) { assert(0); return false; }

virtual bool nb_write(const T&) { assert(0); return false; }

virtual int num_available() const { assert(0); return 0; }

virtual int num_free() const { assert(0); return 0; }

virtual const sc_event& data_written_event() const {
      assert (0); return e;
}
virtual const sc_event& data_read_event() const {
      assert (0); return e;
}
```

Universidade de Brasília



LAICO

hw_fifo_wrapper...

- Elaboração:

```
class top_wrap : public sc_module {
public:
        hw_fifo_wrapper<char> fifo_inst;
        producer prod_inst;
        consumer cons_inst;
        sc_clock clk;


        top_wrap(sc_module_name name, int size) :
                sc_module(name),
                fifo_inst("Fifo1", size),
                prod_inst("Prod_inst"),
                cons_inst("Cons_inst"),
                clk("clk", 1 , SC_NS)
        {
                prod_inst.out(fifo_inst);
                cons_inst.in(fifo_inst);
                fifo_inst.clk(clk);
        }
};
```