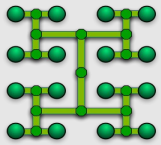
Universidade de Brasília

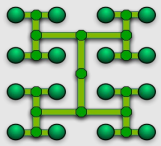


LAICO

Modelagem Dataflow

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

Universidade de Brasília

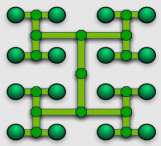


LAICO

Modelagem Funcional

- Nos estágios iniciais do projeto frequentemente estamos interessados apenas em aspectos funcionais do sistema
- Detalhes de temporização, estrutura, ou protocolos de comunicação não são considerados neste estágio
- Modelos funcionais podem ser temporais ou atemporais

Universidade de Brasília

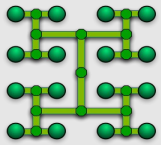


LAICO

Modelo Atemporal: *Dataflow*

- Descrições tipo **dataflow** são um exemplo de modelo atemporal
- O modelo dataflow mais genérico é a rede de processos de Kahn
- Neste modelo, o comportamento é descrito por um mapeamento de sequências de entradas em sequências de saídas

Universidade de Brasília

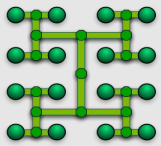


LAICO

Dataflow em SystemC

- É modelado através de processos que se comunicam através de filas acessadas por leituras e escritas bloqueantes
- Atrasos algoritmicos são representados por valores iniciais armazenados nas filas

Universidade de Brasília

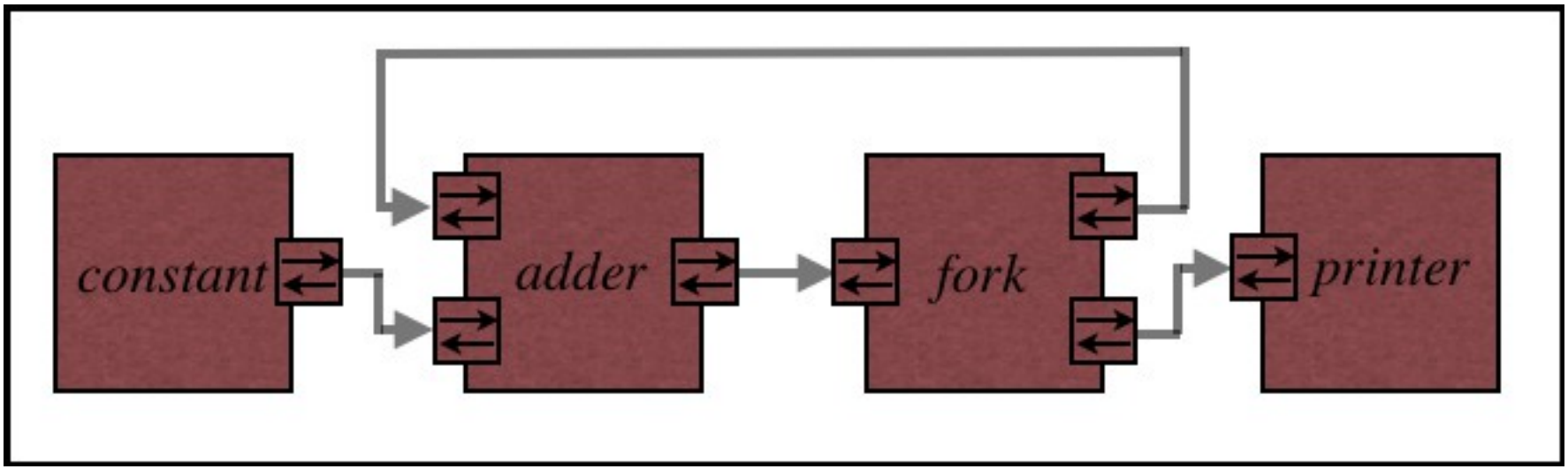


LAICO

Dataflow: dicas

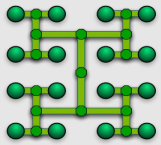
1. Usar SC_THREADS
2. Comunicar através de canais sc_fifo usando métodos read e write bloqueantes
3. Modelar atrasos algoritmicos iniciais através de:
   - 3.1 escrita na fila antes de iniciar a simulação
   - 3.2 geradores de dados que inserem dados antes que eles sejam consumidos
4. Critério de parada:
   - 4.1 simular por um tempo finito
   - 4.2 encerrar processos e interromper a simulação por falta de eventos (data backlog)
   - 4.3 chamar condicionalmente sc_stop()

Sistema *Dataflow* Simples



- constant: gerador de constantes
- adder: somador
- fork: gerador de fluxos
- printer: módulo de saída
- atraso algoritmico ?

Universidade de Brasília

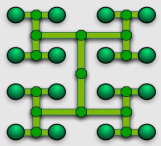


LAICO

Exemplo: DF_ADDER

- DF_Adder é um somador que lê os dados de duas filas de entradas com acessos bloqueantes
- Não necessita de sincronização explícita: a disponibilidade ou não dos dados funciona com sincronizador
- Versão com templates, para torná-la genérica

Universidade de Brasília

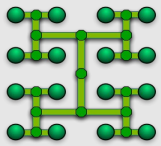


LAICO

DF_Adder.cpp

```
#include <systemc.h>
template <class T> SC_MODULE(DF_Adder) {
    sc_fifo_in<T> in1, in2;
    sc_fifo_out<T> out;

    void proc () {
      while (true) out.write(in1.read() + in2.read());
    }
    SC_CTOR(DF_Adder) {
        SC_THREAD(proc);
    }
};
```

Universidade de Brasília

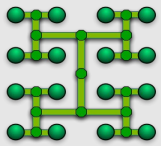


LAICO

Gerador de Constantes

```
#include <systemc.h>

template <class T> SC_MODULE(DF_Const) {
    sc_fifo_out<T> out;
    T _const;

    void proc () {
        while (true) out.write(_const);
    }

    SC_HAS_PROCESS (DF_Const);

    DF_Const(sc_module_name n, const T& c):
       sc_module(n), _const(c) {
        SC_THREAD(proc);
    }
};
```

Universidade de Brasília

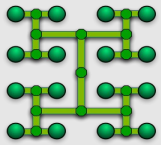


LAICO

Módulo *fork*

```
#include <systemc.h>

template <class T> SC_MODULE(DF_Fork) {
    sc_fifo_in<T> in;
    sc_fifo_out<T> out1, out2;

    void proc () {
       T val;
        while (true) {
             val = in.read();
             out1.write(val);
             out2.write(val);
    }

    SC_CTOR(DF_Fork) {
        SC_THREAD(proc);
    }
};
```

Universidade de Brasília

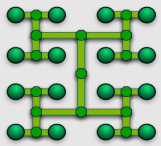


LAICO

Encerramento

- Todos os modelos são atemporais
- Tempo de simulação avança em termos de ciclos delta
- A simulação pode ser executada até que um número suficiente de dados tiver sido produzido
- Neste exemplo, printer consome um certo número de entradas e termina. A interrupção no fluxo de dados pode levar a simulação a parar por falta de eventos

Universidade de Brasília



LAICO

DF_PRINTER

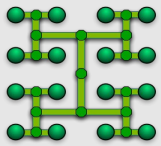
```
template <class T> SC_MODULE(DF_Printer) {
  // IO
  sc_f fo_in<T> in;
  // local
  unsigned _niter;
  bool _done;

  SC_HAS_PROCESS(DF_Printer);

  DF_Printer(sc_module_name n, unsigned niter) :
      sc_module(n), _niter(niter), _done(false) {
        SC_THREAD(proc);
      }

  void proc() {
      for (unsigned i=0; i < _niter; i++) {
          T val = in.read();
          cout << name() << " " << val << endl;
      }
      _done = true;
      return;
  }
  ~DF_Printer() {
      if (!_done) cout << name() << " not done yet." << endl;
  }
};
```

Modelo Temporal

- Pode-se introduzir a noção de tempo no exemplo anterior através de comandos wait(sc_time)
- Ex: gerador de constantes

```
void proc() {
    while (true) {
            wait(200, SC_NS);
    out.write(_const);
    }
 }
```

Universidade de Brasília



LAICO

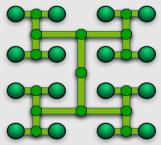
Modelo Temporal

- De forma similar, pode-se acrescentar atrasos ao somador

```
void proc () {
        while (true) {
         T data = in1.read() + in2.read();
         wait(200, SC_NS);
         out.write(data);
        }
}
```

- Como ficam os outros módulos ?

Universidade de Brasília

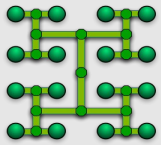


LAICO

Modelos Mistos

- É possível utilizar em conjunto modelos temporais e atemporais
- As filas podem ser escritas por um processo acionado por um relógio, enquanto que é lida por um processo atemporal

Universidade de Brasília

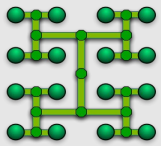


LAICO

Modelos Mistos

```
template <class T> SC_MODULE(DF_CoeffMul) {
    sc_fifo_in<T> inp;
    sc_fifo_out<T> out;
    sc_in<T> coeff;

    void proc() {
        while (true)
          out.write(inp.read() * coeff.read());
    }

    SC_CTOR (DF_CoeffMul) {
            SC_THREAD(proc);
        }
};
```

Universidade de Brasília

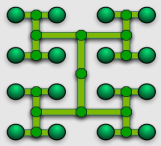


LAICO

Encerrando Simulação Dataflow

- Esperar por uma simulação encerrar por falta de eventos pode não funcionar
- Uma alternativa é indicar através de um sinal Booleano uma condição de encerramento e chamar explícitamente sc_stop()

Universidade de Brasília

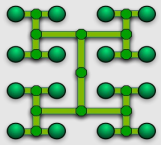


LAICO

Terminator

```
template <class T, unsigned n_iter> SC_MODULE(DF_Printer) {
    sc_f fo_in<T> in;
    sc_out<bool> done;

    SC_CTOR (DF_Printer) {
          SC_THREAD(proc);
          done.initialize(false);
       }

    void proc() {
       for (unsigned i=0; i <  n_iter; i++) {
          cout << name() << " " <<  in.read() << endl;
       done = true;
       }
       while (true) in.read();  // evita congestionamento
    }
};
```

Universidade de Brasília



LAICO

Terminator

```
SC_MODULE(Terminator) {
   sc_port<sc_signal_in_if<bool>, 0> inputs;

   SC_CTOR (Terminator) {
         SC_METHOD(arnold);
         sensitive << inputs;
      }

   void arnold() {
      for (unsigned i=0; i <  inputs.size(); i++) {
         if (inputs[i]->read() == false) return;
     sc_stop();
      }
};
```