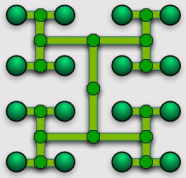
Universidade de Brasília



LAICO

Modelagem RTL SystemC

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

Módulos Combinacionais

- SC_MODULE:
  - Define os componentes, de forma possivelmente hierárquica
  - sc_in<T> e sc_out<T> definem as entradas e saídas
- SC_METHOD
  - Descreve o comportamento de cada componente
  - Sensível às entradas do componente

Estrutura do Arquivo

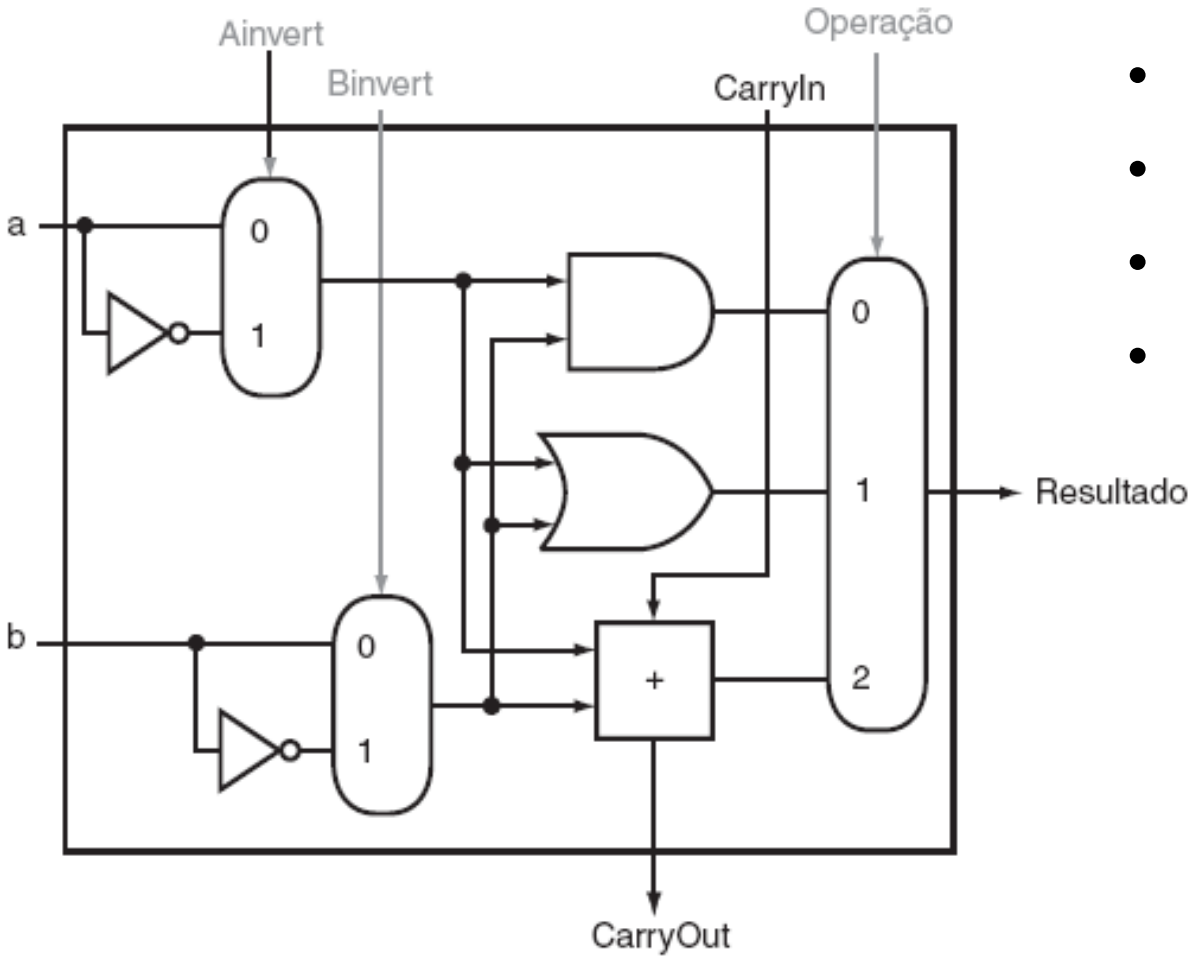
- # includes
- SC_MODULE (componente)
  - sc_in<T> entradas
  - sc_out<T> saidas
  - SC_CTOR(componente) { // construtor
    - SC_METHOD(proc1);
    - sensitive << in1 << in2 << … ;
    - SC_METHOD(proc2);
    - sensitive << in1 << in2 << … ;

    }
  - void proc1 () { … }
  - void proc2 () { … }

Descrição Estrutural...

```
SC_MODULE (mux3_1) {
    sc_in<bool> x0, x1, x2;
    sc_in<sc_lv<2> > sel;
    sc_out<bool> z;
    SC_CTOR (mux3_1) {
        SC_METHOD(proc);
        sensitive << x0 << x1 << x2 << sel;
    }
    void proc() {
        switch (sel) {
          case "00": z = x0; break;
          case "01": z = x1; break;
          case "10": z = x2; break;
          default : z =0; break;
          }
    }
};
```

Exemplo: ULA MIPS



- Inversor
- Multiplexador
- Porta E
- Porta OU
- Somador

Descrição Estrutural

```
#include "systemc.h"
SC_MODULE (p_e) {
sc_in<bool> a, b;
sc_out<bool> z;

  SC_CTOR(p_e) {
    SC_METHOD(exec);
    sensitive << a << b;
  }

  void exec() {
      z = a & b;
  }
};
```

```
#include "systemc.h"
SC_MODULE (p_ou) {
sc_in<bool> a, b;
sc_out<bool> z;

  SC_CTOR(p_ou) {
    SC_METHOD(exec);
    sensitive << a << b;
  }

  void exec() {
      z = a | b;
  }
};
```

Descrição Estrutural...

```
SC_MODULE (fa) {
    sc_in<bool> x, y, vem;
    sc_out<bool> soma, vai;

    SC_CTOR (fa) {
    SC_METHOD(proc);
      sensitive << x << y << vem;
    }

    void proc() {
      soma = x ^ y ^ vem;
      vai = x&y | x&vem | y&vem;
    }
};
```

```
SC_MODULE (mux2_1) {
    sc_in<bool> x0, x1, s;
    sc_out<bool> z;

    SC_CTOR (mux2_1) {
        SC_METHOD(proc);
        sensitive << x0 << x1 << s;
    }
    void proc() {
        if (s == true)
            z = x1;
        else
            z = x0;
    }
};
```

Descrição Estrutural...

```
SC_MODULE (ula) {
sc_in<bool> a, b, ainv, binv, vem;
sc_in<sc_uint<2> > op;
sc_out<bool> sai, vai;

sc_signal<bool> t0, t1, t2, t3, t4, ia, ib;

p_e *pe;
p_ou *pou;
fa *pfa;
mux2_1 *pmux20;
mux2_1 *pmux21;
mux3_1 *pmux3;

SC_CTOR (ula) {
      pe   = new p_e("pe");
      pou = new p_ou("pou");
      pfa = new fa("pfa");
      pmux20 = new mux2_1("pmux20");
      pmux21 = new mux2_1("pmux21");
      pmux3 = new mux3_1("pmux3");
```

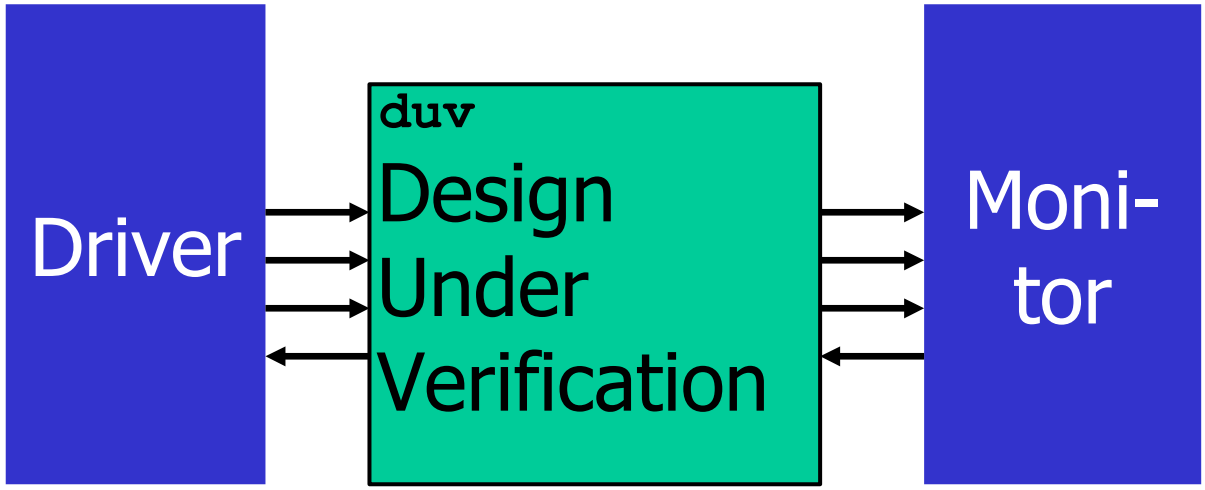
```
pmux20->x0(a); pmux20->x1(ia);
pmux20->s(ainv); pmux20->z(t0);

pmux21->x0(b); pmux21->x1(ib);
pmux21->s(binv); pmux21->z(t1);

pe->a(t0); pe->b(t1); pe->z(t2);

pou->a(t0); pou->b(t1); pou->z(t3);

pfa->x(t0); pfa->y(t1); pfa->vem(vem);
pfa->soma(t4); pfa->vai(vai);

pmux3->x0(t2); pmux3->x1(t3); pmux3->x2(t4);
pmux3->sel(op); pmux3->z(sai);

SC_METHOD(inv1);
sensitive << a;
SC_METHOD(inv2);
sensitive << b;
}
```

Descrição Comportamental

```
void ula::exec() {
    bool and_out, or_out, soma, vai;
    bool ta, tb;

    ta = a ^ ainv;
    tb = b ^ binv;

    and_out = ta & tb;
    or_out  = ta | tb;
    soma = ta ^ tb  ^  vem;
    vai = ta&tb | ta&vem | tb&vem;

    switch (op.read()) {
        case 0: sai = and_out; break;
        case 1: sai = or_out; break;
        case 2: sai = soma; break;
        default: sai = 0; break;
    }
}
```

Verificando a ULA

- Estrutura Driver => DUV => Monitor
  - Driver gera estímulos
  - DUV processa e gera as saídas
  - Monitor processa os resultados



DUV: Device Under Verification

Driver e Monitor

- SC_THREAD(Driver)
  - Acessa o DUV apenas através de sua interface
  - Saídas do Driver são as entradas do DUV
  - Gera estímulos e utiliza wait() para avançar o tempo de simulação

- SC_METHOD(Monitor)
  - Acessa o DUV apenas através de sua interface
  - Entradas do Monitor são as saídas do DUV
  - Método sensível aos eventos na saída do DUV
  - Usualmente imprime o tempo de simulação

driver.h

```
#include "systemc.h"

SC_MODULE(driver) {
    sc_out<bool> a, b, ainv, binv, vem;     // entradas da ULA
    sc_out<sc_uint<2> > op;                 // sao saidas do driver

    SC_CTOR(driver) {
        SC_THREAD(exec);
    }

    void exec();
};
```

driver.cpp

```
void driver::exec() {
        cout << "Verificando operacao AND: \n";
        a = 0; b = 0; ainv = 0; binv = 0; vem = 0; op = 0;
        wait(1, SC_NS);
        //cout << "a/b/ai/bi/ci/op :" <<a<<b<<ainv<<binv<<vem<<op << endl;
        a = 0; b = 1; ainv = 0; binv = 0; vem = 0; op = 0;
        wait(1, SC_NS);
        //cout << "a/b/ai/bi/ci/op :" <<a<<b<<ainv<<binv<<vem<<op<< endl;
        a = 1; b = 0; ainv = 0; binv = 0; vem = 0; op = 0;
        wait(1, SC_NS);
        //cout << "a/b/ai/bi/ci/op :" <<a<<b<<ainv<<binv<<vem<<op<< endl;
        a = 1; b = 1; ainv = 0; binv = 0; vem = 0; op = 0;
        wait(1, SC_NS);
        //cout << "a/b/ai/bi/ci/op :" <<a<<b<<ainv<<binv<<vem<<op<< endl;
    ...
}
```

monitor.h

```
SC_MODULE(monitor) {
sc_in<bool> sai, vai;

    SC_CTOR(monitor) {
        SC_METHOD(exec);
        sensitive << sai << vai;
    }

    void exec() {
        std::cout << sc_time_stamp() << " : sai / vai = "
        << sai << " / " << vai << endl;
    }
};
```

Rastreamento de Sinais

- SystemC permite o rastreamento de sinais visando a exibição de formas de onda
- Usa o formato VCD (*Value Change Dump*) que representa textualmente as variações dos sinais
- Visualização:
  - gtkwave
  - dinotrace
  - … vários outros

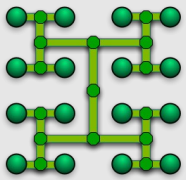
Rastreando Sinais...

```
namespace sc_core {
   class sc_trace_file {
      public:
         virtual void
               set_time_unit( double , sc_time_unit ) = 0;
               implementation-defined
   };
   sc_trace_file*
         sc_create_vcd_trace_file( const char* name );
   void sc_close_vcd_trace_file( sc_trace_file* tf );
   void sc_write_comment( sc_trace_file* tf ,
                          const std::string& comment );
   void sc_trace ...
}
```

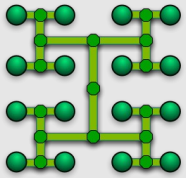
Rastreando Sinais...

- Ex:

```
// cria arquivo de rastreamento
sc_trace_file* trace_file = sc_create_vcd_trace_file("wave");

trace_file->set_time_unit(1.0, SC_NS);        // seta unidade de simulação
sc_trace(trace_file, a, "a");                 // define sinais a rastrear
sc_trace(trace_file, b, "b");
sc_trace(trace_file, ci, "ci");
sc_trace(trace_file, res, "res");
sc_trace(trace_file, vai, "vai");

sc_start();                                   // simulação

sc_close_vcd_trace_file(trace_file);          // fecha arquivo de rastreamento
cout << "Criou wave.vcd" << endl;
```



Universidade de Brasília



LAICO

Lógica Síncrona

- modelamento de flip-flops
- com set/reset assíncrono
- com set/reset sincrono
- latches

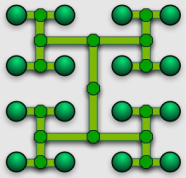
Flip-flops

- São definidos a partir da lista de sensitividade, fazendo o método sensível a borda de um sinal

```
SC_MODULE (biestavel) {
    sc_in<bool> d, clk;
    sc_out<bool> q, qb;

    void proc();

    SC_CTOR (biestavel) {
        SC_METHOD (proc);
        sensitive << clk.pos();
    }
};
```

```
void biestavel::proc() {
    q = d;
    qb = !d;
}
```

Universidade de Brasília



LAICO

Verificação

```
SC_MODULE (driver) {
    sc_out<bool> d, clk;

    void drive();

    SC_CTOR(driver) {
        SC_THREAD(drive);
    }
};

void driver::drive() {
    d = 1; clk = 0; wait(10, SC_NS);
    clk = 1; wait(10, SC_NS);
    clk = 0; wait(10, SC_NS);
    d = 1; clk = 0; wait(10, SC_NS);
}
```

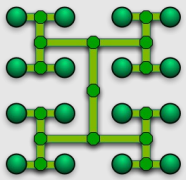
Verificação...

```
SC_MODULE (monitor) {
    sc_in<bool> q, qb, clk;

    void monitora ();

    SC_CTOR(monitor) {
        SC_METHOD (monitora);
        sensitive << q << qb << clk;
    }
};

void monitor::monitora() {
   cout << "Tempo " << sc_time_stamp() << " : ";
   cout << " (q, qb, clk) = " << q << qb << clk << endl;
}
```

Inferência

- Portas e sinais atribuidos sob controle da transição de um sinal são inferidos como flip-flops
- A mesma descrição vale para registradores, com largura de n bits
- Cada processo pode ser sensível ou a transição ou ao evento, mas não a ambos
- Apenas sinais e portas do tipo booleano podem ser sensíveis a borda

Universidade de Brasília



LAICO

Registradores

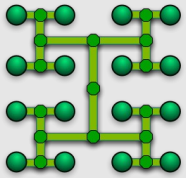
```
SC_MODULE (registrador) {
    sc_in<sc_uint<WIDTH> > regIn;
    sc_in<bool> clk;
    sc_out<sc_uint<WIDTH> > regOut;

    void proc();

    SC_CTOR (registrador) {
        SC_METHOD (proc);
        sensitive << clk.neg();
    }
};

void registrador::proc() {
    regOut = regIn;
}
```

```
void driver::drive() {
    regD = "10000001"; clk = 0;
    wait(10, SC_NS);
    clk = 1;
    wait(10, SC_NS);
    clk = 0;
    wait(10, SC_NS);
    regD = "00001111"; clk = 0;
    wait(10, SC_NS);
}
```

Banco de Registradores

- Considerando como exemplo o banco de registradores do MIPS:
  - 32 registradores de 32 bits
  - Leitura simultânea de 2 registradores
    - 2 endereços de leitura (5 bits cada)
    - 2 saídas de 32 bits
  - Escrita de um registrador
    - 1 endereço de escrita
    - 1 entrada de dados de 32 bits
    - 1 sinal de habilitação de escrita (*wr*)
  - Registrador zero é a constante zero

Universidade de Brasília

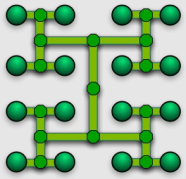


LAICO

Header BREG

```
SC_MODULE(breg_mips) {
        sc_in<sc_uint<32> > dado;
        sc_in<sc_uint<5> > ia, ib, iwr;
        sc_in<bool> clk, wr;
        sc_out<sc_uint<32> > a, b;

        SC_CTOR(breg_mips) {
            SC_METHOD(write);       // método de escrita sensível à borda do relógio
            sensitive << clk.pos();
            SC_METHOD(read);        // método de leitura sensível à eventos
            sensitive << ia << ib << clk << iwr << wr; // prever qualquer evento
        }

        void write();
        void read();

private:
        sc_uint<32> breg[32];
};
```

Universidade de Brasília



LAICO

Código BREG

```
void breg_mips::read() {
        // constante zero: basta retornar zero na leitura
        a = (ia.read()==0)?(sc_uint<32>)0:breg[ia.read()];
        b = (ib.read()==0)?(sc_uint<32>)0:breg[ib.read()];
}

void breg_mips::write() {
        if (wr == 1)
            breg[iwr.read()] = dado;
}
```