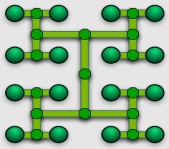
Universidade de Brasília



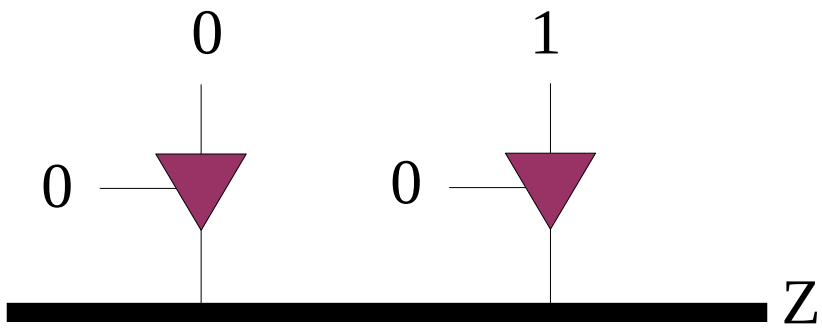
LAICO

Modelagem RTL Sequencial

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

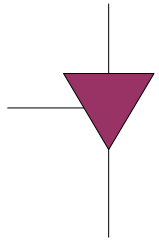
Three-state Drivers

- A lógica "*three-state*" modela sinais não acionados, elétricamente flutuando
  - Conexão a barramentos
- Em SystemC, deve-se utilizar o tipo `sc_logic` para representar sinais em tri-state
- Atribui-se o valor 'Z' aos sinais ou portas do tipo `sc_logic` ou `sc_lv<>`

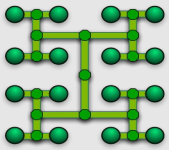


Three-state Buffer

```
SC_MODULE( tri_buffer ) {
        sc_in<bool> en;
        sc_in<sc_logic> d_in;
        sc_out<sc_logic> d_out;

        // Method for three-state driver
        void t_buffer () {
            if (en.read())
                d_out.write(d_in.read());
            else
                d_out = sc_logic_Z;
        }
        // Constructor
        SC_CTOR( tri_buffer ) {
            SC_METHOD( t_buffer );
            sensitive << en << d_in;
        }
};
```



Universidade de Brasília



LAICO

Decodificador tri-state

```
#include "systemc.h"
SC_MODULE (tri_decoder) {
    sc_in<bool> en;
    sc_in<sc_uint<3> > index;
    sc_out<sc_lv<8> > saida;

    void proc();

    SC_CTOR (tri_decoder) {
        SC_METHOD (proc);
        sensitive << en << index;
    }
};
```

Saída tipo sc_lv<>

Decodificador tri-state ...

```
void tri_decoder::proc() {
    sc_lv<8> slogic = "00000000";

    if (!en)
        slogic = "ZZZZZZZZ";  // type cast
    else
        slogic[index.read()] = '1';
    saida = slogic;
}
```

Valor default: evitar latches

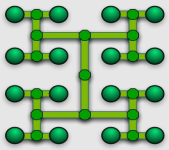
Sinal local temporário

Decodificador tri-state ...

```
int sc_main (int argc, char * argv[]) {

sc_signal<bool> habilita;
    sc_signal<sc_uint<3> > index;
    sc_signal<sc_lv<8> > saida;

    tri_decoder dec("Decoder");
    dec.en(habilita);
    dec.index(index);
    dec.saida(saida);

tri_tb tb("tb");
tb.en(habilita);
    tb.index(index);
    tb.saida(saida);
...
```

Decodificador tri-state ...

```
…
sc_trace_file* trace_file = sc_create_vcd_trace_file("wave");

trace_file->set_time_unit(1.0, SC_NS);
sc_trace(trace_file, habilita, "habilita");
sc_trace(trace_file, index, "index");
sc_trace(trace_file, saida, "saida");

sc_start();

sc_close_vcd_trace_file(trace_file);
cout << "Criou wave.vcd" << endl;

    return 0;
}
```



Decodificador tri-state ...

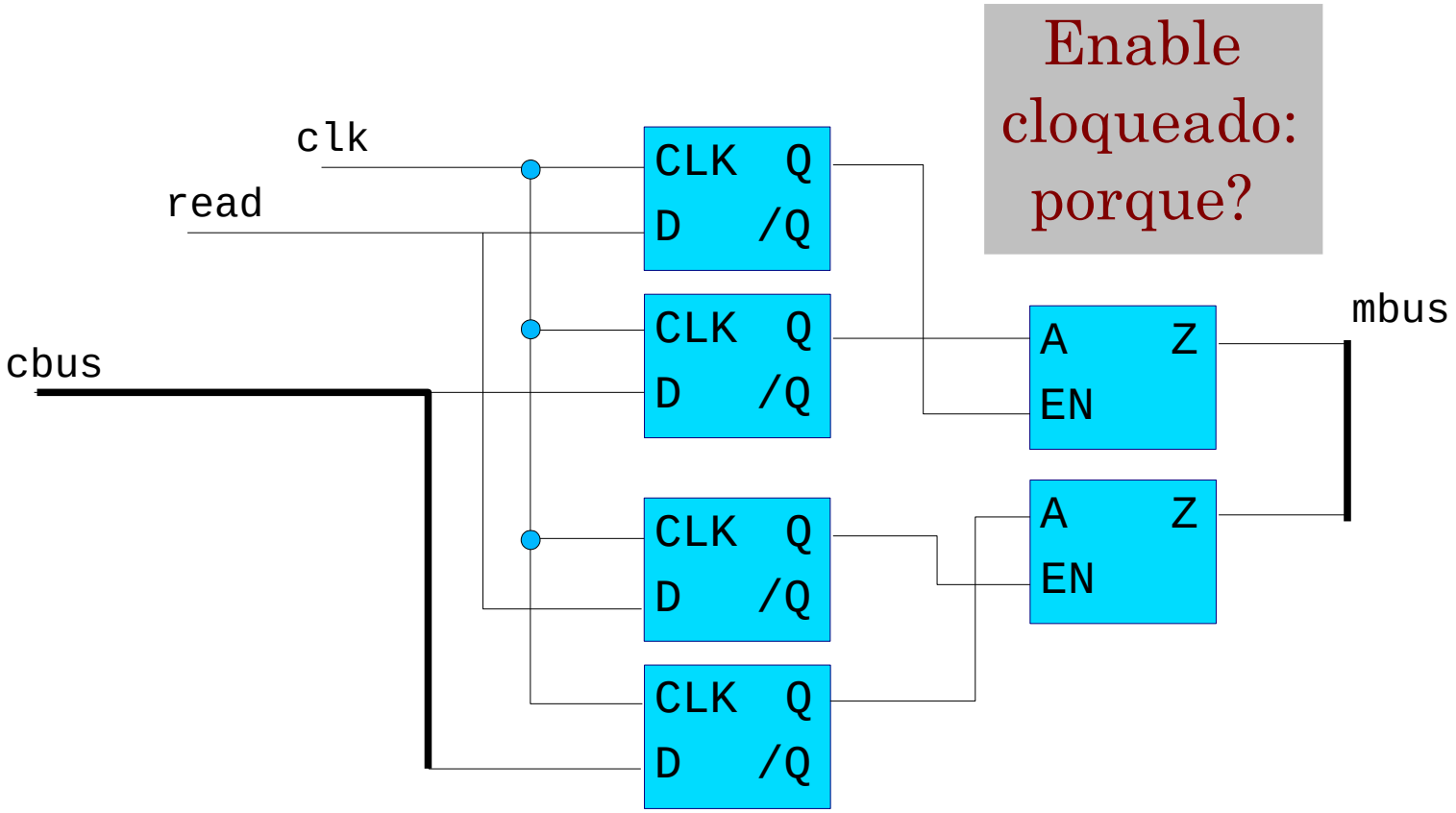
```
void tri_tb::drive() {
  en = false; index = 3;
  wait (1, SC_NS);
  cout << sc_time_stamp()<<" en: "<<en<<" index: "<<index<<" sai: "<<saida<<endl;

  en = true; index = 3;
wait (1, SC_NS);
        cout << sc_time_stamp() << " en: " << en << " index: "
              << index << " sai: " << saida << endl;
        ...
        en = true; index = 7;
        wait (1, SC_NS);
        cout << sc_time_stamp() << " en: " << en << " index: "
        << index << " sai: " << saida << endl;

        en = false; index = 5;
        wait (1, SC_NS);
        cout << sc_time_stamp() << " en: " << en << " index: "
        << index << " sai: " << saida << endl;
}
```

Barramento Tri-State com Reg

```
SC_CTOR (tri_bus) {
        SC_METHOD (proc);
        sensitive << clk.pos();
    }
void tri_bus::proc() {
    sc_lv<BUSIZE> temp(sc_logic_Z);

    if (!read)
        mbus = temp;
    else
        mbus = cbus.read();
}
```

Inicia sinal com Z's

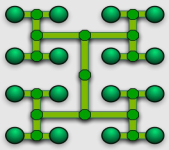
Síntese Tri-State Bus



Barramento

- O sinal `read` é amostrado na subida do relógio e a saída não muda de valor caso `read` seja alterado entre dois flancos do relógio
- Para evitar a inserção dos *flip-flops* extras pode-se criar um outro módulo sensível ao sinal read, que representa os *latches* tri-state de saída

Universidade de Brasília



LAICO

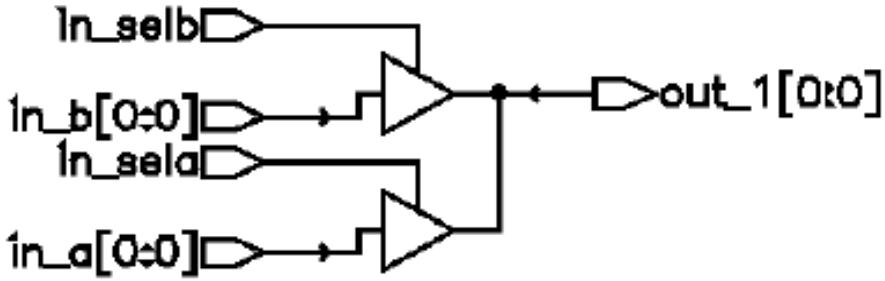
Barramento 3ST sem Reg

```
SC_MODULE( tristate_ex3 ) {
   sc_in<bool> in_sela, in_selb;
   sc_in_rv<1> in_a, in_b;
   sc_out_rv<1> out_1;
   void tristate_a(); // first three-state driver
   void tristate_b(); // second three-state driver
   SC_CTOR( tristate_ex3 ) { // Constructor
     SC_METHOD( tristate_a);
     sensitive << in_sela << in_a;
     SC_METHOD( tristate_b);
     sensitive << in_selb << in_b;
   }
};
```

Barramento 3ST sem Reg...

```
 void tristate_ex3::tristate_a() {
   if (in_sela.read()){
     out_1.write(in_a.read());
   } else {
      out_1.write("Z");
   }
}
void tristate_ex3::tristate_b() {
  if (in_selb.read()){
    out_1.write(in_b.read());
   }else{ out_1.write("Z");
   }
}
```

Síntese:



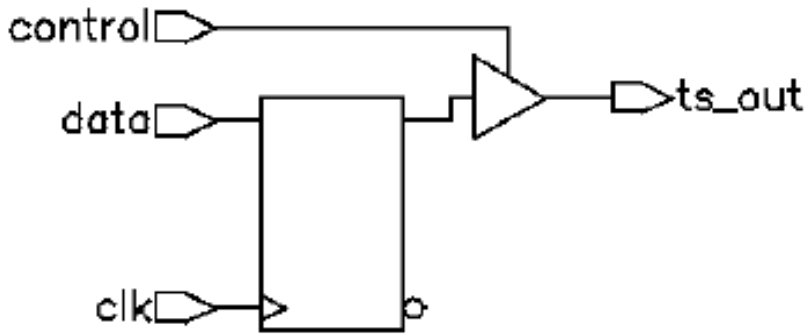
Tri-State com Reg e Enable

```
// simple three-state buffer inference
#include "systemc.h"
SC_MODULE( tristate_ex5 ) {
  sc_in<bool> control;        sc_in<sc_logic> data;
  sc_out<sc_logic> ts_out;    sc_in_clk clk;
  sc_signal<sc_logic> temp; // sinal que interliga
   processos

// Method for three-state driver
  void tristate_fcn () {
     if (control.read()){ ts_out.write(temp);
     }else{ ts_out.write( Z );
     }
}
```

Tri-State com Reg e Enable

```
// Method for sequential logic
void flop () {
  temp = data.read();
}

// Constructor
SC_CTOR( tristate_ex5 ) {
  SC_METHOD( tristate_fcn );
  sensitive << control << temp ;
  SC_METHOD( flop );
  sensitive << clk.pos();
 }
};
```

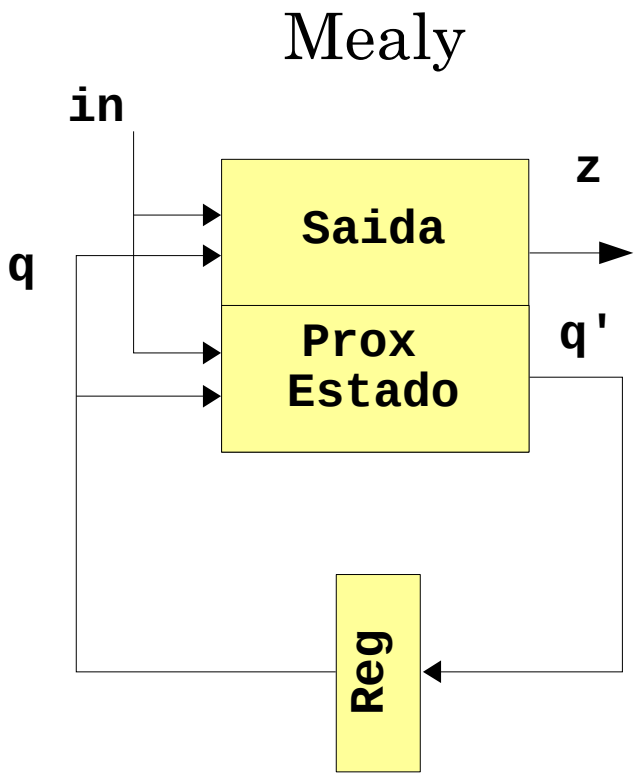
Síntese:

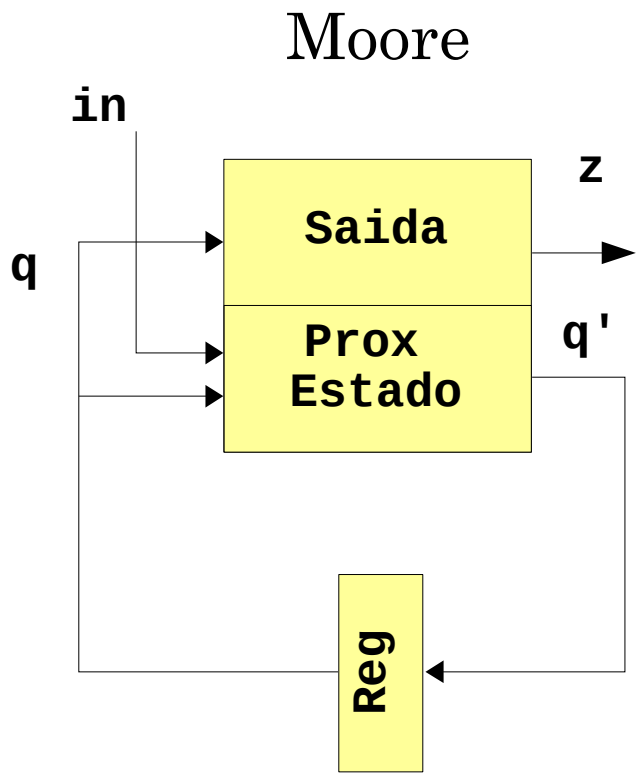


Máquinas de Estado

- Temos dois tipos de máquinas de estado, que se diferenciam na forma como as saídas são calculadas:
  - Mealy: saída depende do estado e das entradas, ou seja, pode dar respostas assíncronas
  - Moore: saída depende apenas do estado atual
  - O próximo estado em ambos os casos é calculado em função das entradas e do estado atual

Máquinas de Estado



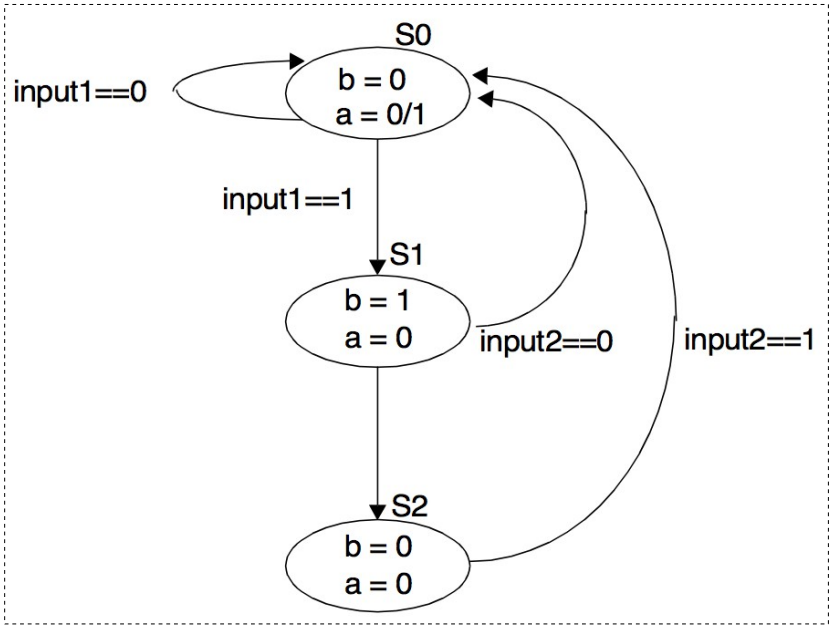


Implementação de FSM

- Alternativas:
  - Um SC_METHOD para atualizar o registrador de estado e outro SC_METHOD para calcular as saídas e próximo estado
  - Um SC_METHOD para o registrador e dois outros para as funções de saída e próximo estado
  - Um único SC_METHOD para toda a máquina

Exemplo Mealy

```
SC_MODULE(ex_fsm_a){
    sc_in_clk clk; sc_in<bool> rst, input1, input2;
    sc_out<bool> a, b;
    sc_signal<state_t> state, next_state;

    void ns_op_logic();
    void update_state();

    SC_CTOR(ex_fsm_a) {
        SC_METHOD(update_state);
        sensitive << clk.pos();
        SC_METHOD(ns_op_logic);
        sensitive << state << input1 << input2;
    }
};
```



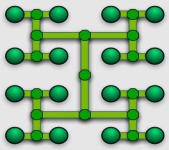
Exemplo Mealy

```
enum state_t { // enumerate states
      S0, S1, S2
};

void ex_fsm_a::update_state() {
      if (rst.read() == true)
         state = S0;
      else state = next_state;
}
```

Modela o registrador de estados, atualizado na transição do relógio

Reset síncrono

Universidade de Brasília



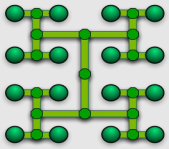
LAICO

Exemplo Mealy

```
void ex_fsm_a::ns_op_logic() {
// Determine next state and output logic
        switch(state) {
            case S0:    b.write(0); // saida depende das entradas
                        if (input1.read() || input2.read()) a.write(1);
                        else a.write(0);
                        next_state=(input1.read() == 1)?S1:S0; break;
            case S1:    a.write(0); b.write(1);
                        next_state=(input2.read() == 1)?S2:S0; break;
            case S2:    a.write(0); b.write(0);
                        next_state = (input2.read() == 1)?S0:S2; break;
            default:    a.write(0); b.write(0); next_state = S0;   break;
        }
}
```

Usando Três Processos

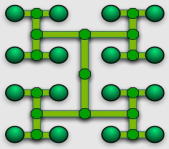
```
SC_MODULE(fsm_b){
      sc_in_clk clk; sc_in<bool> rst, input1, input2;
      sc_out<bool> a, b;
      sc_signal<state_t> state, next_state;

      void ns_logic();         // lógica de próximo estado
      void output_logic();     // lógica de saída
      void update_state();  // registrador de estado
      SC_CTOR(fsm_b){
          SC_METHOD(update_state);
          sensitive << clk.pos();
          SC_METHOD(ns_logic);
          sensitive << state << input1 << input2;
          SC_METHOD(output_logic);
          sensitive << state << input1 << input2; }
};
```



Usando Três Processos ...

```
enum state_t { // enumerate states
                S0, S1, S2 };

void fsm_b::update_state() {
        if (rst.read() == true) state = S0;
        else state = next_state;
}
void fsm_b::output_logic() { // determine outputs
        a.write(state == S0 && (input1.read() || input2.read()));
        b.write(state == S1);
}
```

Universidade de Brasília



LAICO

Uso de Três Processos ...

```
void fsm_b::ns_logic() { // Determine next state
        switch(state) {
          case S0: if (input1.read() == 1)
                       next_state = S1;
                   else next_state = S0;
                   break;
          case S1: if (input2.read() == 1)
                     next_state = S2;
                   else next_state = S0; break;
          case S2: next_state = (input2.read()==1)?S0:S2;
                   break;
        default: next_state = S0; break;
        }
}
```

Exercício

- Modelar a máquina de estados Controlador de Tráfego em SystemC
  - Estrada principal e estrada secundária com sensor
  - Verde para a principal se sensor inativo
  - Sensor ativo: sistema circula parando em tempo longo no vermelho e um tempo curto no amarelo
  - Supor um cronômetro que, depois de acionado, retorna um sinal TC (tempo curto) e um sinal TL (tempo longo).