Digital Design with the Verilog HDL

Chapter 8 Datapath & Controller

Binh Tran-Thanh

Department of Computer Engineering
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology

May 26, 2023

Digital Systems Classification

Control-dominated

- Reactive systems responding to external events

Data-dominated

- High throughput data computation and transport

$\Rightarrow$ Sequential machines are classified and portioned into datapath units and control units

What are controller, Datapath, and its example

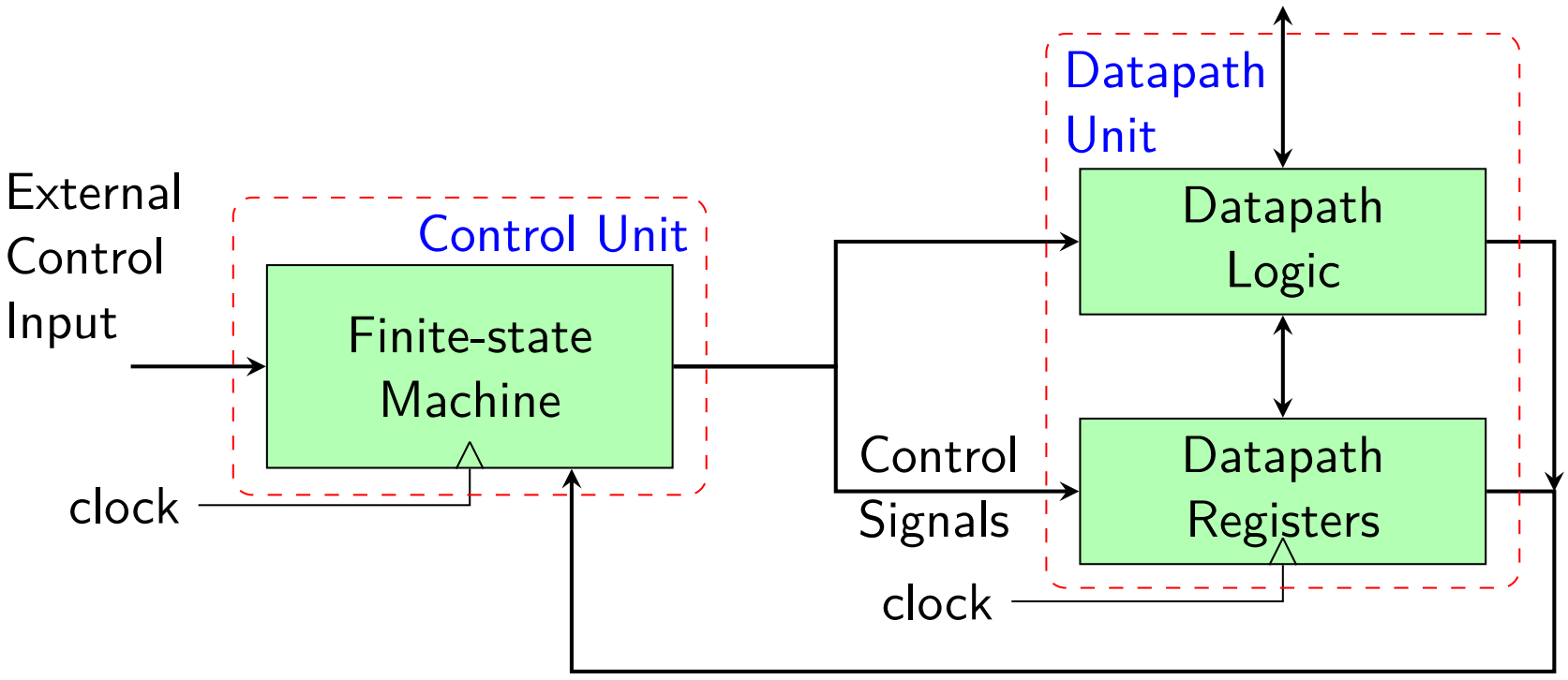
Control units

- FSM coordinating the execution of instructions that perform operation on the datapath

Datapath units

- Computational resources (ALU, register,...)
- Adder, multiplier
- DSP
- ...

State-machine Controller and Datapath



Controller and Datapath Modelling

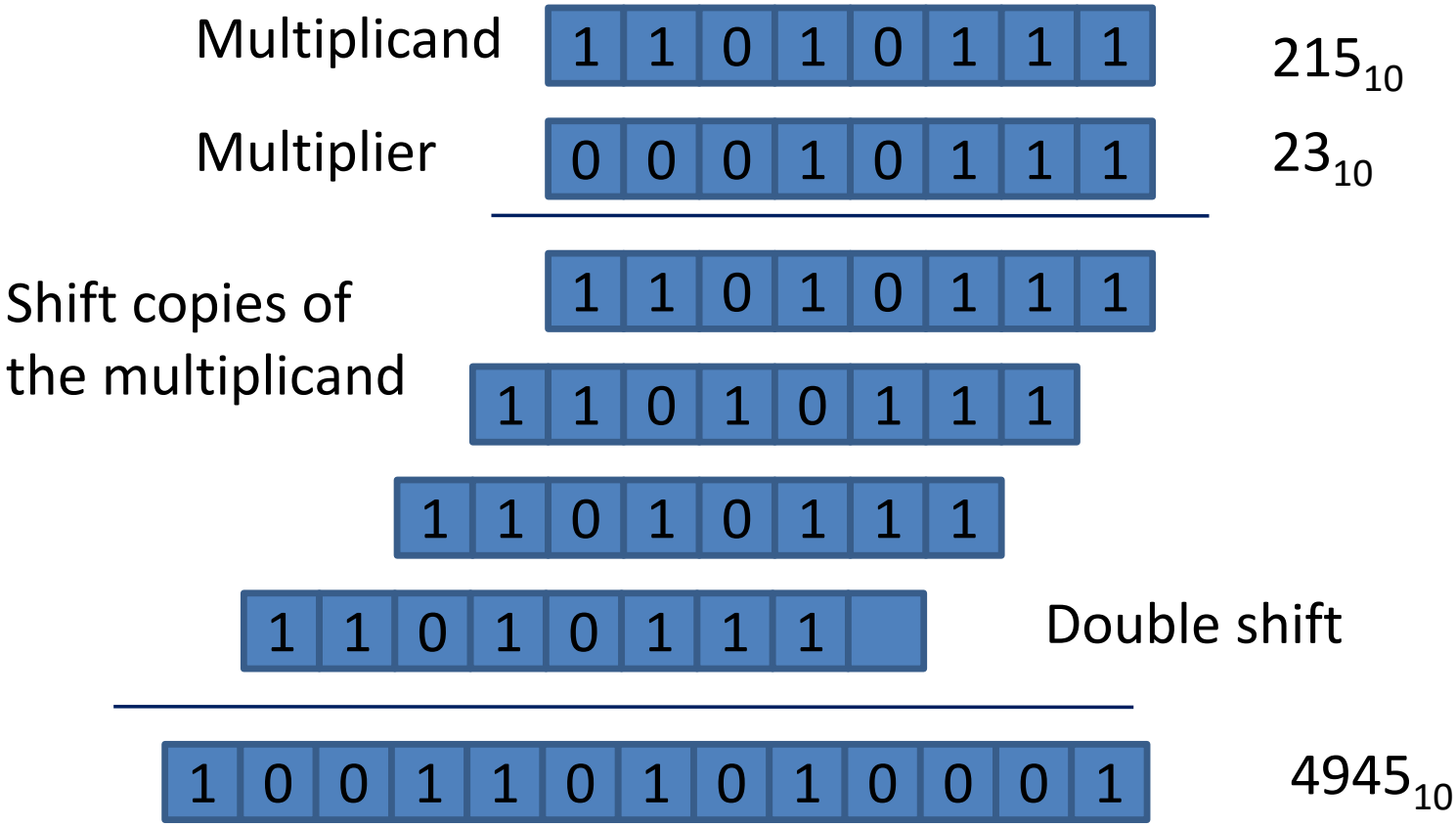
Modelling

- Controller:
  - State transaction graphs
  - Algorithmic-state machine (ASM)
- Datapath:
  - Data flow graph

Functional

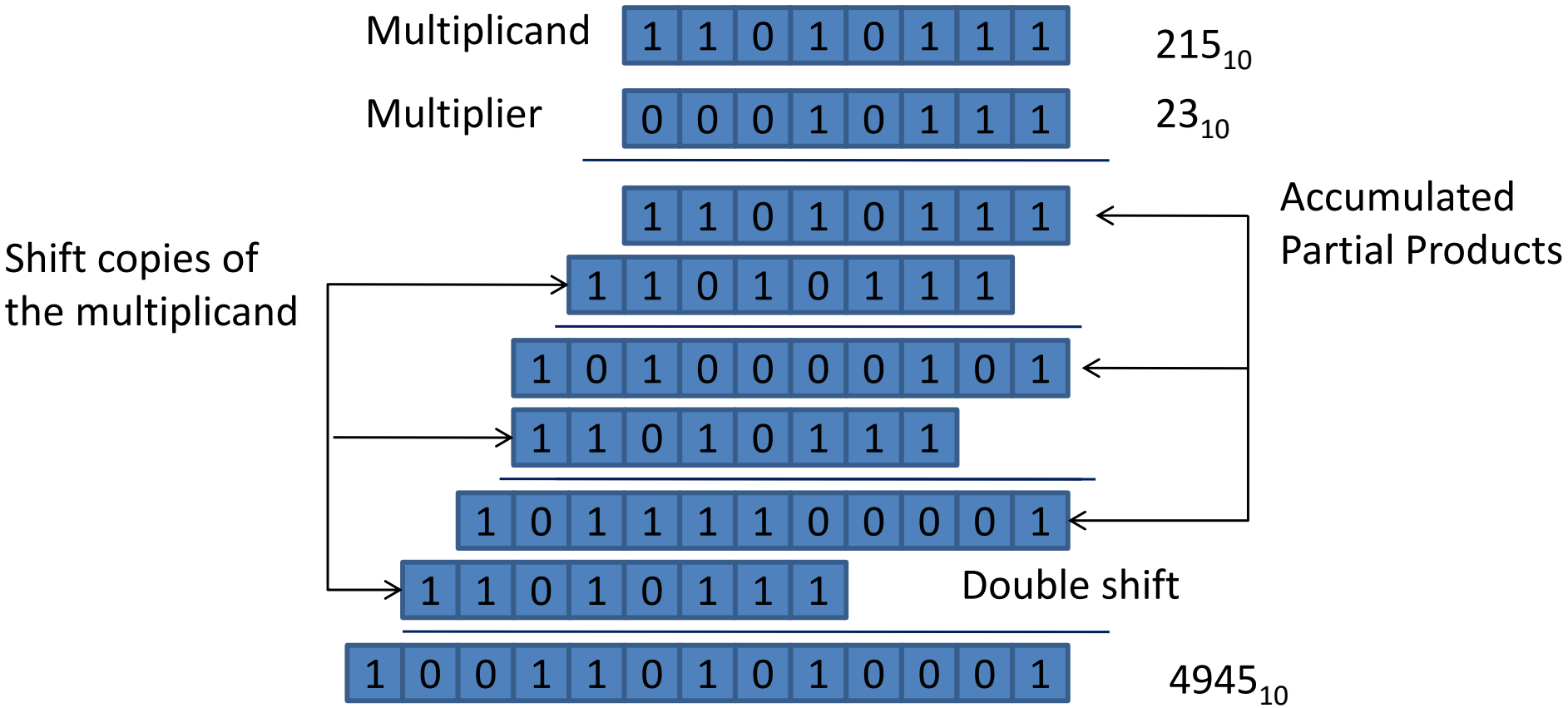
- Controller:
  - Generate signals: load, read, clear, and shift storage registers
  - Control the operations: ALU, complex datapath units
- Datapath:
  - Perform operations

Combinational Binary Multiplier



- A combinational Circuit can be developed to implement the product
- Require hardware with multiple adders for each column
- Ordinary adder operates on only two words at a time

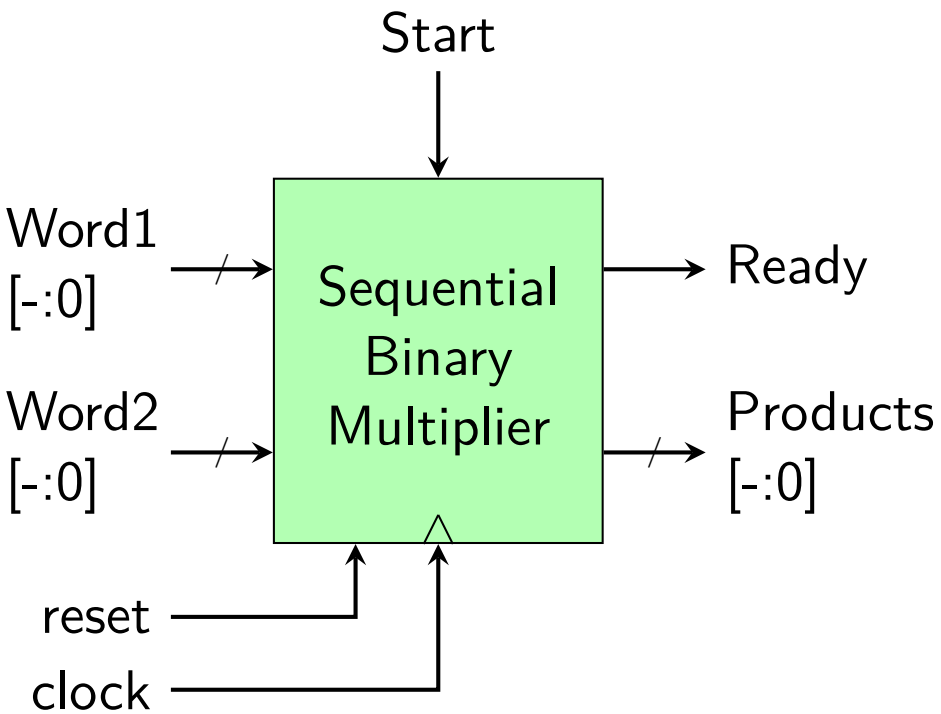
Combinational Binary Multiplier

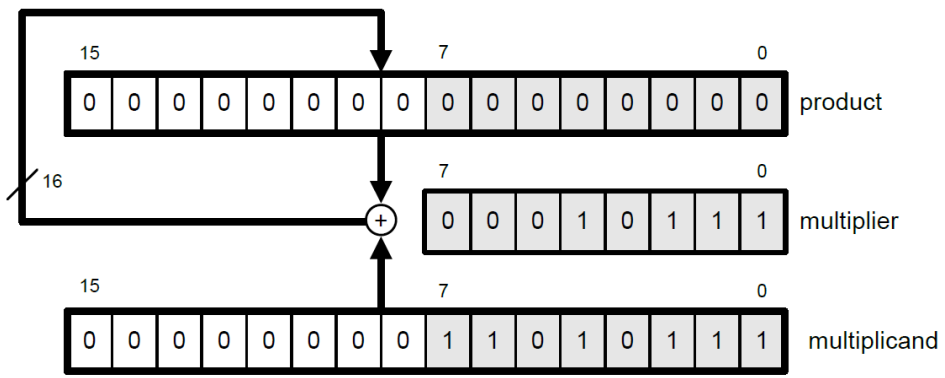


- Combinational Binary Multiplier operate fast, but require a significant amount of silicon area

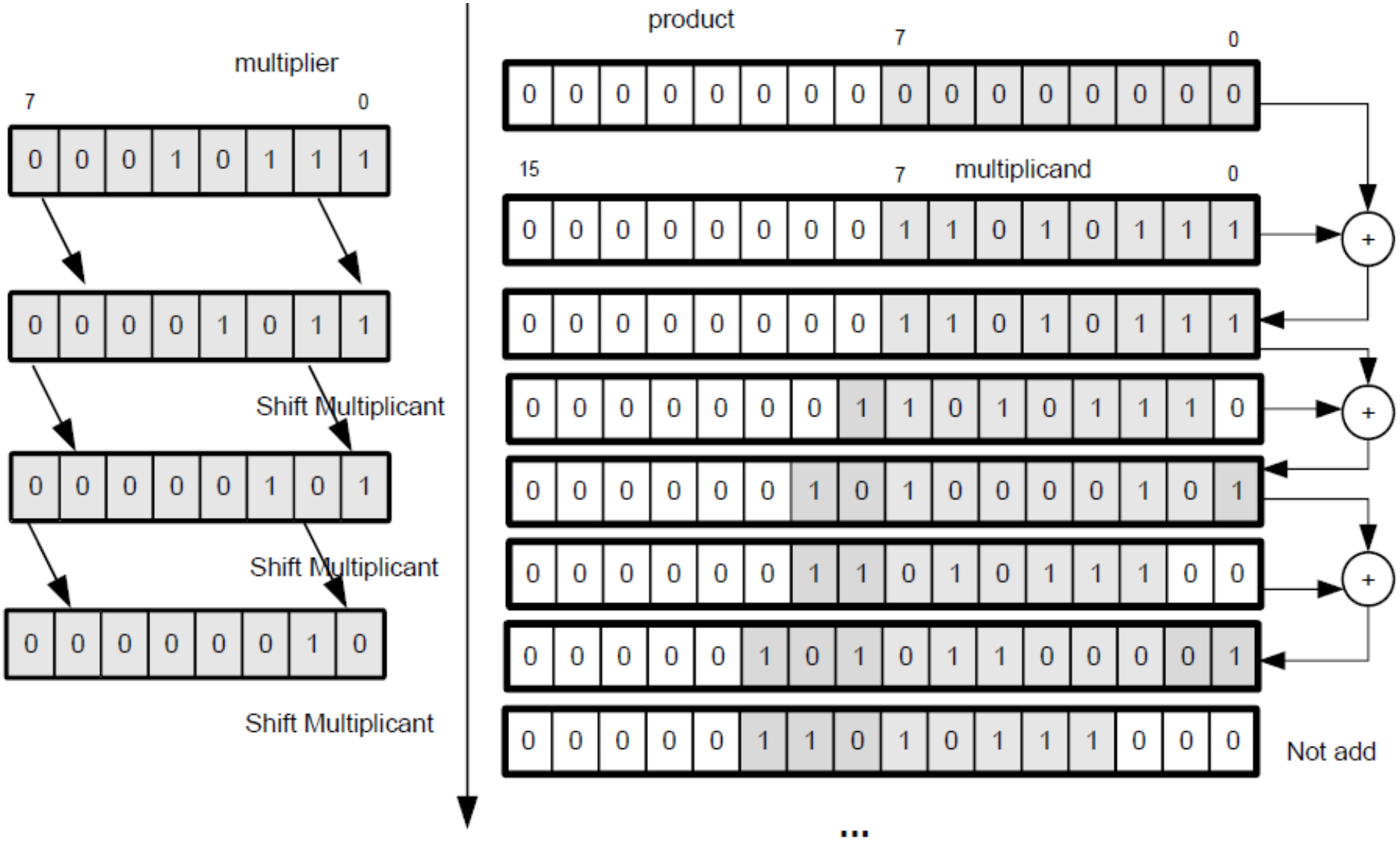
Sequential Binary Multiplier

- Choose a datapath architecture
- Design state machine for controller



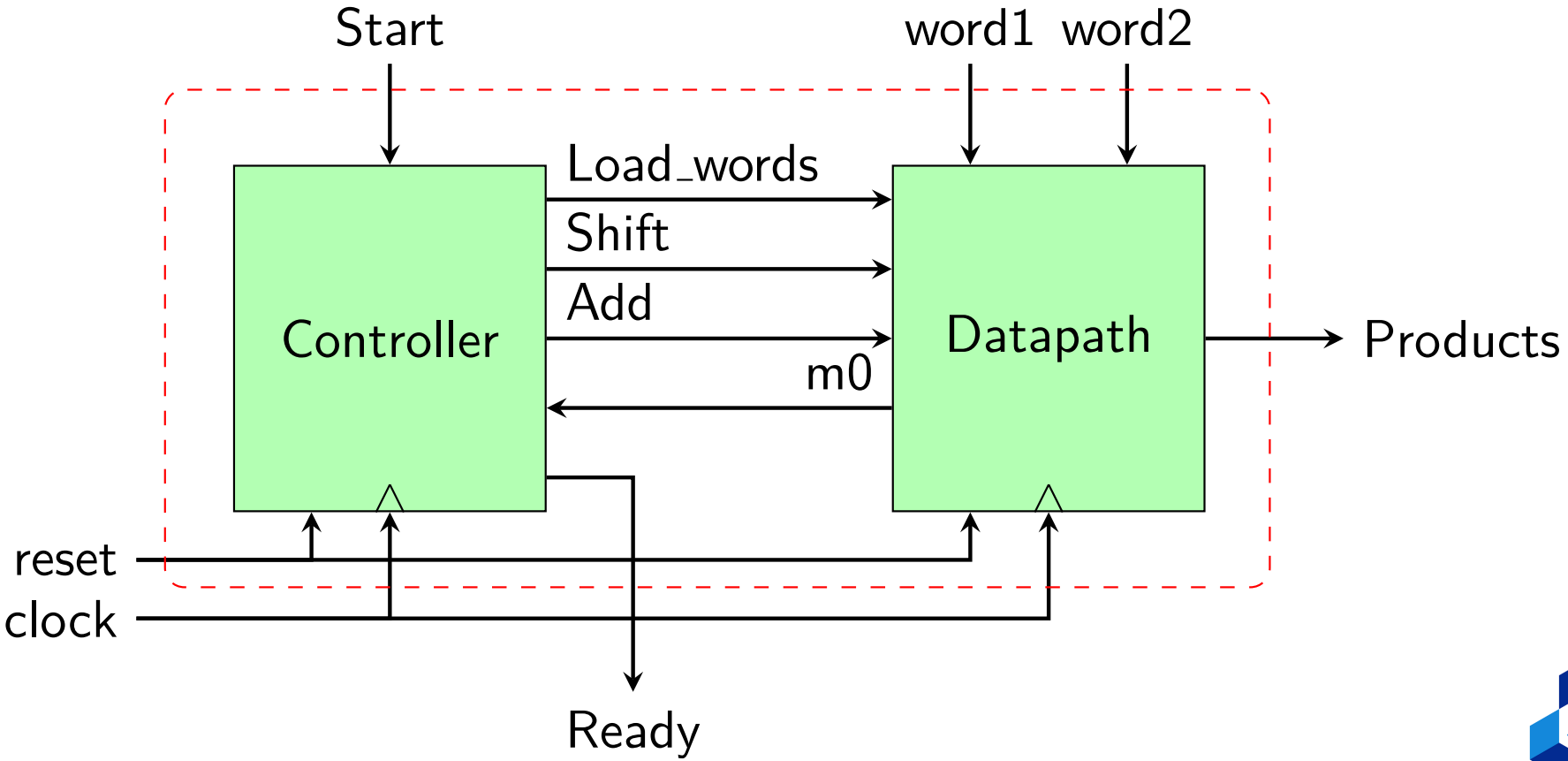


Register transfers

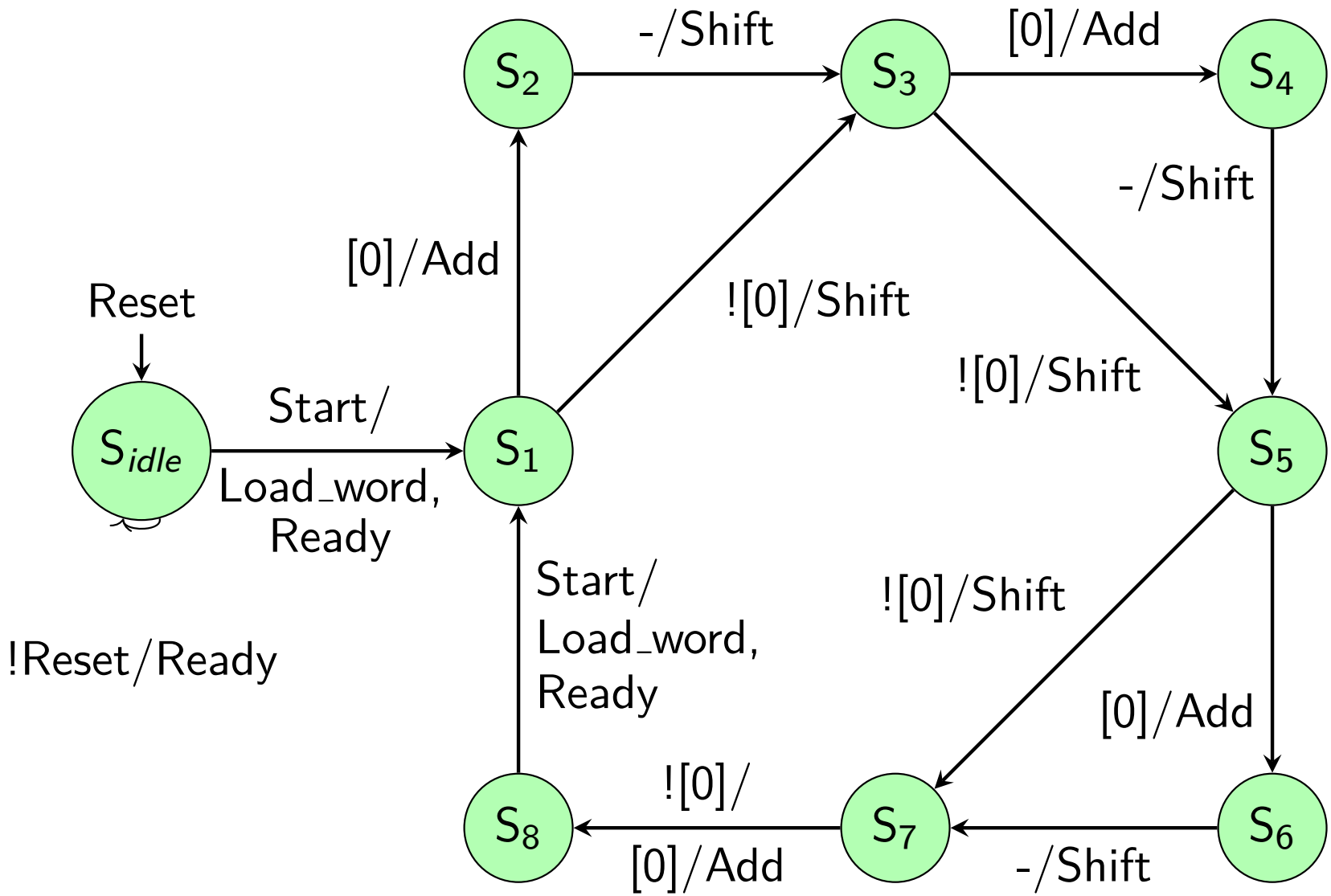


Structural Units

- Top-level module: Multiplier_STG_0
- m0: LSB of Multiplier, used to control state transaction



STGs for 4-bit Seq. Multiplier



Module Decleration

```
module Multiplier_STG_0 (product, Ready, word1, word2,
    Start, clk, rst);
  parameter L_WORD= 4;// Datapathsize
  output[2*L_WORD-1: 0] product;
  output Ready;
  input [L_WORD -1: 0] word1, word2;
  input Start, clk, rst;
  wire m0, Load_words, Shift;

  Datapath M1(product, m0, word1, word2, Load_words, Shift,
     Add, clk, rst);
  Controller M2(Load_words, Shift, Add, Ready, m0, Start,
    clk, rst);
endmodule
```

Controller (1/2)

```
module Controller (Load_words, Shift, Add, Ready, m0, Start
    , clk, rst);
  output Load_words, Shift, Add, Ready;
  input m0, Start, clk, rst;

  parameter L_WORD= 4;// Datapath size
  parameter L_STATE= 4;// State size
  parameter S_idle= 0, S_1 = 1, S_2 = 2, S_3 = 3;
  parameter S_4 = 4, S_5 = 5, S_6 = 6, S_7 = 7, S_8 = 8;

  reg Load_words, Shift, Add;
  reg[L_STATE-1: 0] state, next_state;

  wire Ready = ((state == S_idle) && !rst) ||
               (state == S_8);

  /* State transitions */
  always @ (posedge clk or posedge rst)
__if(rst) state <= S_idle;
__else    state <= next_state;
```

Controller(2/2)

```
/*Next state and control logic */
always @ (state or Start or m0) begin
  Load_words= 0; Shift = 0; Add = 0;
  case(state)
   S_idle: if(Start) begin Load_words=1; next_state=S_1;
    end
            else next_state= S_idle;
   S_1:if(m0) begin Add = 1; next_state= S_2; end
       else begin Shift = 1; next_state= S_3; end
   S_2: begin Shift = 1; next_state= S_3; end
   S_3: if(m0) begin Add = 1; next_state= S_4; end
        else begin Shift = 1; next_state= S_5; end
   S_4: beginShift = 1; next_state= S_5; end
   S_5: if(m0) begin Add = 1; next_state= S_6; end
        else begin Shift = 1; next_state= S_7; end
   S_6: beginShift = 1; next_state= S_7; end
   S_7: if(m0) begin Add = 1; next_state= S_8; end
        else begin next_state= S_8; end
   S_8: if(Start) beginLoad_words= 1; next_state= S_1; end
        else next_state= S_8;
   default:next_state= S_idle;
  endcase
end
endmodule
```

Datapath

```
module Datapath(product, m0, word1, word2, Load_words,
    Shift, Add, clk, rst);
  parameter L_WORD= 4;
  output [2*L_WORD-1: 0] product;
  output m0;
  input [L_WORD-1: 0] word1, word2;
  input Load_words, Shift, Add, clk, rst;
  reg [2*L_WORD-1: 0] product, multiplicand;
  reg [L_WORD-1: 0] multiplier;
  wire m0 = multiplier[0];

  always @ (posedge clk or posedge rst) begin
    if (rst) begin multiplier <= 0;
            multiplicand <= 0; product <= 0; end
    else if (Load_words) begin multiplicand <= word1;
            multiplier <= word2; product <= 0; end
    else if (Shift) begin multiplier <= multiplier >> 1;
            multiplicand <= multiplicand << 1; end
    else if (Add) begin product <= product+ multiplicand;
    end
  end
endmodule
```