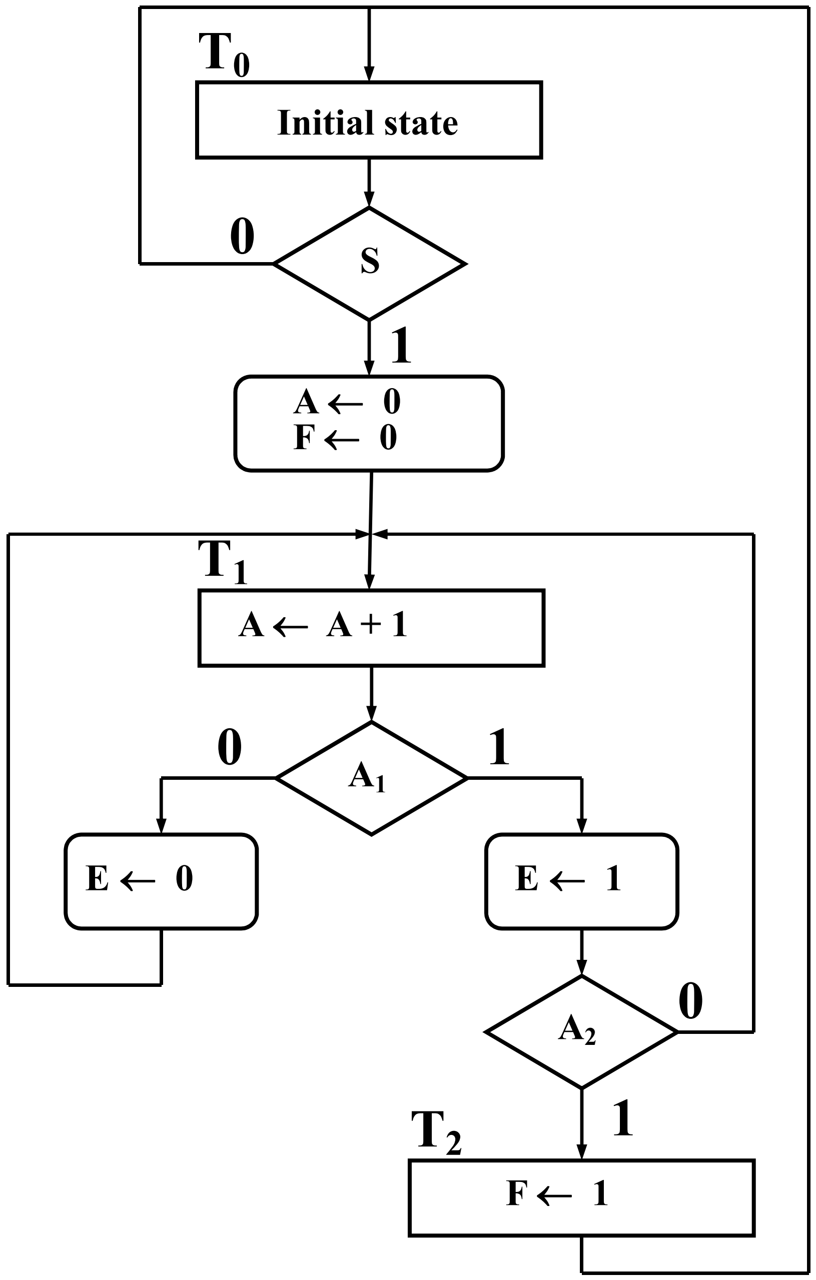
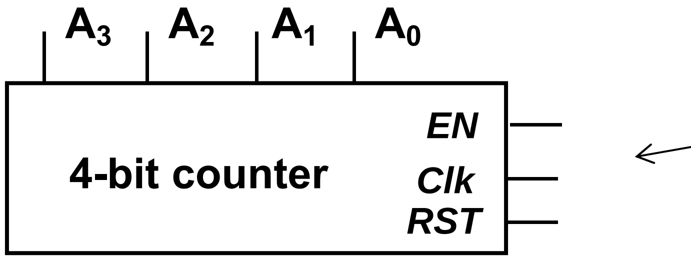
Algorithmic State Machines (II)

- ASMD Circuit Solution
- Verilog Model of ASMD


T_0
Initial state
S
0
1
A ← 0
F ← 0
T_1
A ← A + 1
A_1
0
1
E ← 0
E ← 1
A_2
0
1
T_2
F ← 1

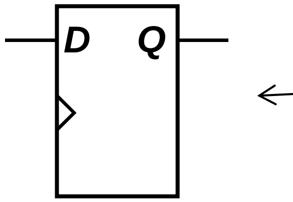
Data Path Components

$(A \leftarrow A + 1)$ when state = T_1
$(A \leftarrow 0)$ when state = T_0 and S = 1



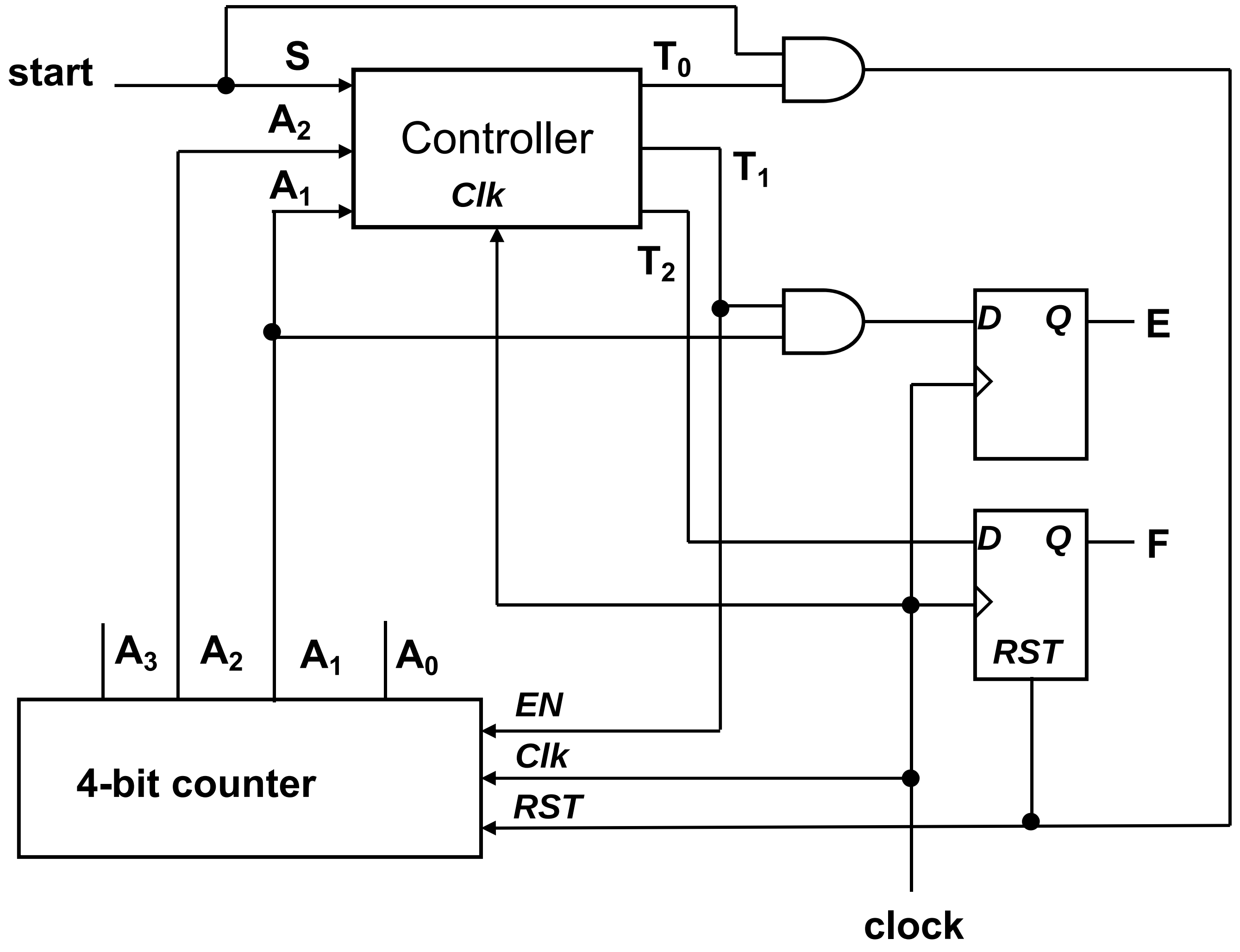
4-bit counter with synchronous reset and chip enable

$(E \leftarrow 1)$ when state = T_1 and $A_1 = 1$
$(E \leftarrow 0)$ when state = T_1 and $A_1 = 0$
$(F \leftarrow 1)$ when state = T_2



D-Type flip-flop

Controller & Data Path



Design Problem:

A sequential circuit is required for a counting application.

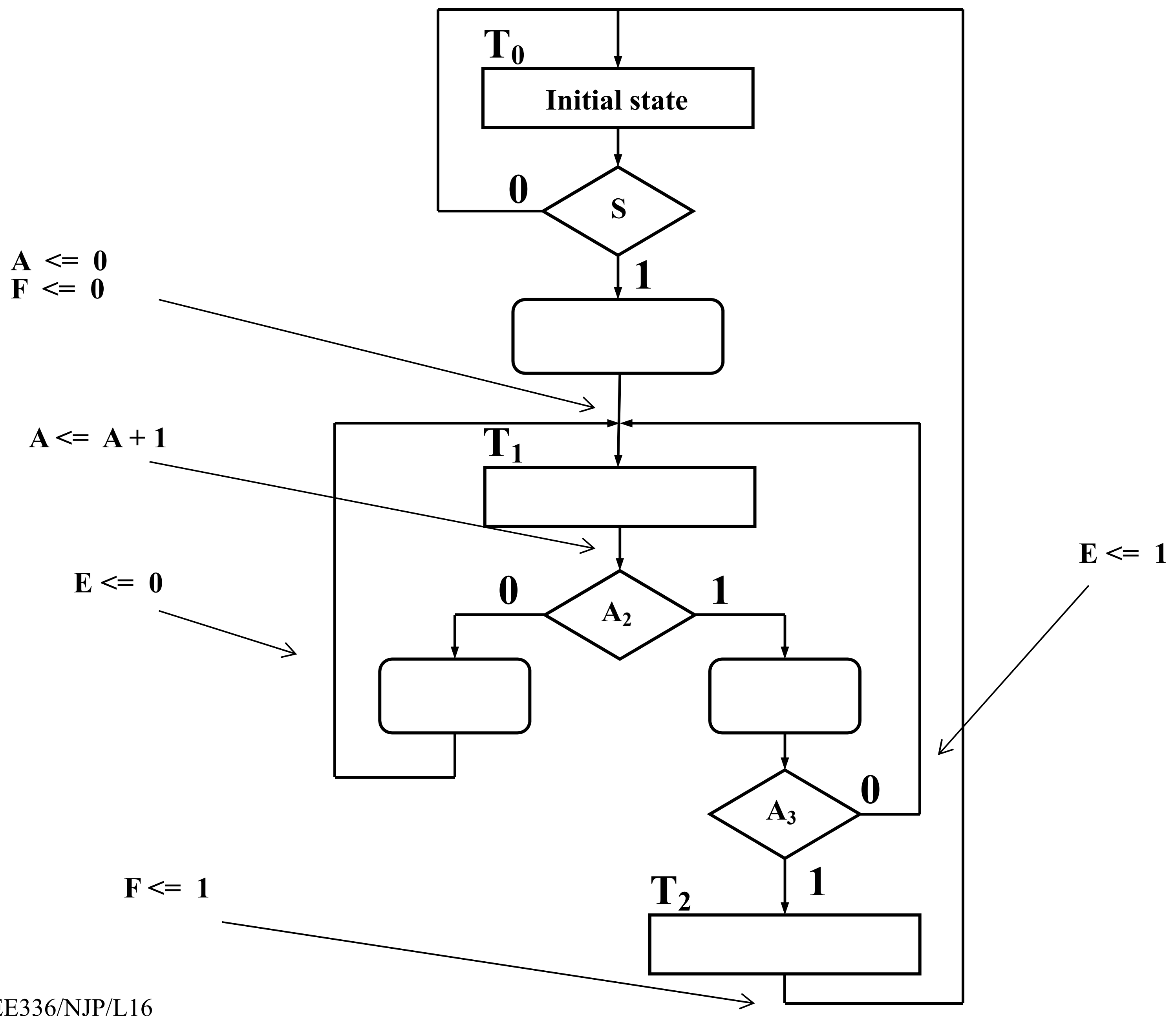
The system outputs are a four bit register A[3:0] to hold the count and two flip-flops E and F.

A signal, *Start*, initiates the system's operation by clearing the counter and status bit F.

At subsequent clock pulses the counter is incremented by 1 until the operation stops.

Counter bits A_2 and A_3 determine the sequence of operations.

Reference: Digital Design, 4e, Mano & Ciletti



T_0
Initial state
0
S
1
A <= 0
F <= 0
A <= A + 1
T_1
0
A_2
1
E <= 0
E <= 1
0
A_3
1
T_2
F <= 1

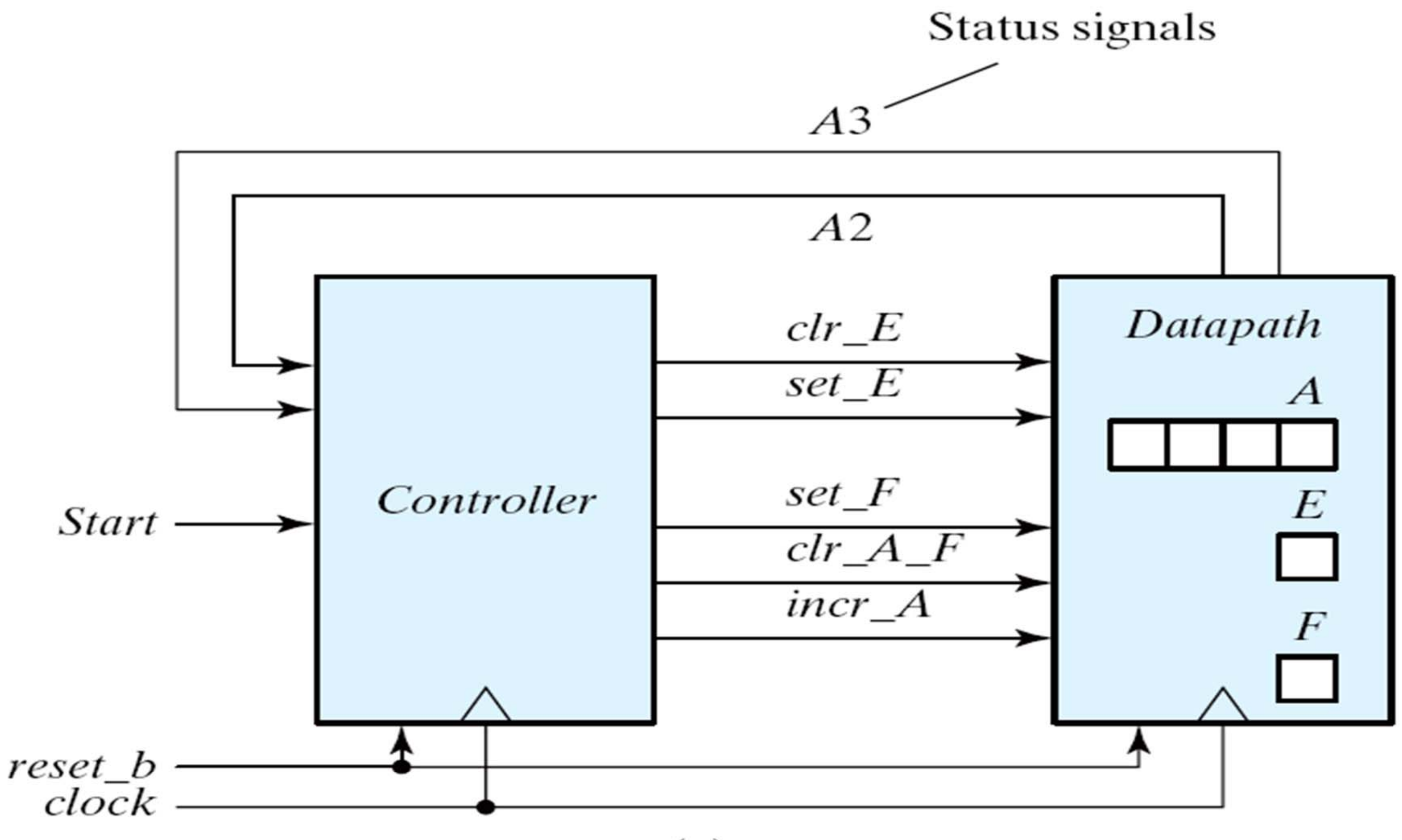
Sequence of operations:

If $A_2 = 0$, E is cleared to 0 and the count continues.

If $A_2 = 1$, E is set to 1; then, if $A_3 = 0$, the count continues, but if $A_3 = 1$, F is set to 1 on the next clock pulse and the system stops counting.

Then, if Start = 0, the system remains in the initial state, but if Start = 1, the operation cycle repeats.

(Slightly different from previous example)



(a)

Note: A3 denotes A[3],
A2 denotes A[2],
<= denotes nonblocking assignment
reset_b denotes active-low reset condition

Top Level RTL Description of Example ASMD

```
module ASMD_RTL (A, E, F, Start, clock, reset_b);

   output [3:0] A;
   output E, F;
   input Start, clock, reset_b;

   Controller_RTL   M0 (set_E, clr_E, set_F, clr_F, incr_A, A[2], A[3], Start, clock, reset_b);
   Datapath_RTL     M1 (A, E, F, set_E, clr_E, set_F, clr_A_F, incr_A, clock);

endmodule
```

Top Level RTL Description of Example ASMD

```
module ASMD_RTL (A, E, F, Start, clock, reset_b);

  output [3:0] A;
  output E, F;
  input Start, clock, reset_b;

  Controller_RTL   M0 (set_E, clr_E, set_F, clr_F, incr_A, A[2], A[3], Start, clock, reset_b);
  Datapath_RTL     M1 (A, E, F, set_E, clr_E, set_F, clr_A_F, incr_A, clock);

endmodule
```

structural top level

RTL Description of Controller

```
module Controller_RTL (set_E, clr_E, set_F, clr_A_F, incr_A, A2, A3, Start, clock, reset_b);

  output reg       set_E, clr_E, set_F, clr_A_F, incr_A;
  input            Start, A2, A3, clock, reset_b;
  reg [1:0]        state, next_state;
  parameter        S_idle = 2'b00, S_1 = 2'b01, S_2 = 2'b11;

  always @ (posedge clock or negedge reset_b) begin
    if (reset_b == 0) state <= S_idle;
    else state <= next_state;
  end
```

RTL Description of Controller

```
module Controller_RTL (set_E, clr_E, set_F, clr_A_F, incr_A, A2, A3, Start, clock, reset_b);

  output reg         set_E, clr_E, set_F, clr_A_F, incr_A;
  input              Start, A2, A3, clock, reset_b;
  reg [1:0]          state, next_state;
  parameter          S_idle = 2'b00, S_1 = 2'b01, S_2 = 2'b11;

  always @ (posedge clock or negedge reset_b) begin
    if (reset_b == 0) state <= S_idle;
    else state <= next_state;
  end
```

register for state

define states

Synchronous update

Next State Logic

```
always @ (state, Start, A2, A3) begin
   next_state = S_idle;
   case (state)
      S_idle:        if (Start) next_state = S_1; else next_state = S_idle;
      S_1:           if (A2 & A3) next_state = S_2; else next_state = S_1;
      S_2:           next_state = S_idle;
      default:       next_state = S_idle;
   endcase
end
```

Next State Logic

```
always @ (state, Start, A2, A3) begin
  next_state = S_idle;
  case (state)
    S_idle:       if (Start) next_state = S_1; else next_state = S_idle;
    S_1:          if (A2 & A3) next_state = S_2; else next_state = S_1;
    S_2:          next_state = S_idle;
    default:      next_state = S_idle;
  endcase
end
```

combinational logic using blocking assignments

Output Logic

```
  always @ (state, Start, A2) begin
    set_E     = 0;
    clr_E     = 0;
    set_F     = 0;
    clr_A_F  = 0;
    incr_A    = 0;
    case (state)
      S_idle:          if (Start) clr_A_F = 1;
      S_1:             begin incr_A = 1; if (A2) set_E = 1; else clr_E = 1; end
      S_2:             set_F = 1;
    endcase
  end
endmodule
```

Datapath Logic

```
module Datapath_RTL (A, E, F, set_E, clr_E, set_F, clr_A_F, incr_A, clock);

  output reg [3:0]  A;
  output reg        E, F;
  input             set_E, clr_E, set_F, clr_A_F, incr_A, clock;

  always @ (posedge clock) begin
    if (set_E)      E <= 1;
    if (clr_E)      E <= 0;
    if (set_F)      F <= 1;
    if (clr_A_F)    begin A <= 0; F <= 0; end
    if (incr_A)     A <= A + 1;
  end
endmodule
```

Datapath Logic

```
module Datapath_RTL (A, E, F, set_E, clr_E, set_F, clr_A_F, incr_A, clock);

  output reg [3:0]  A;
  output reg        E, F;
  input             set_E, clr_E, set_F, clr_A_F, incr_A, clock;

  always @ (posedge clock) begin
    if (set_E)      E <= 1;
    if (clr_E)      E <= 0;
    if (set_F)      F <= 1;
    if (clr_A_F)    begin A <= 0; F <= 0; end
    if (incr_A)     A <= A + 1;
  end
endmodule
```

sequential logic using nonblocking assignments