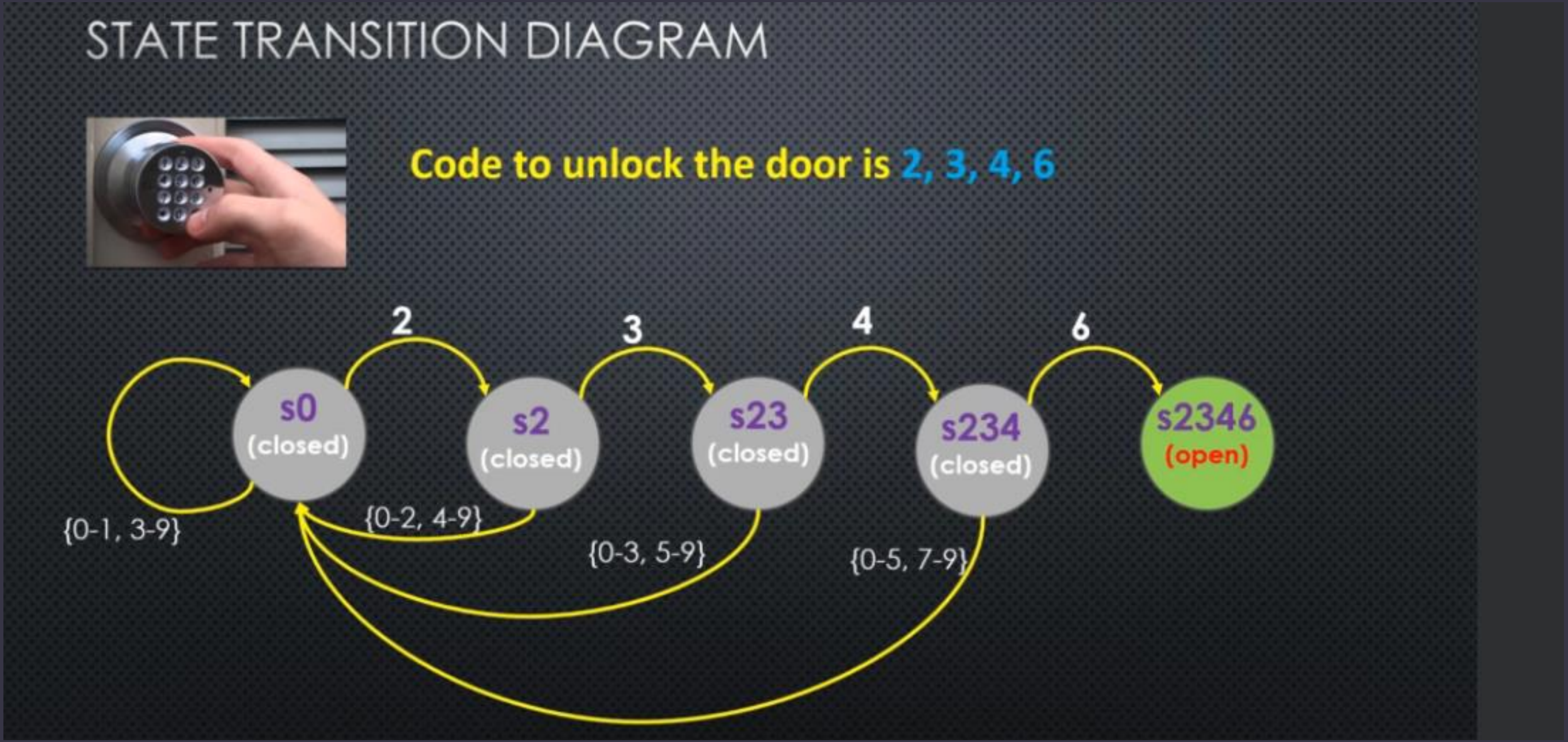
STATE MACHINES

Lecture-10

FINITE STATE MACHINE (FSM)

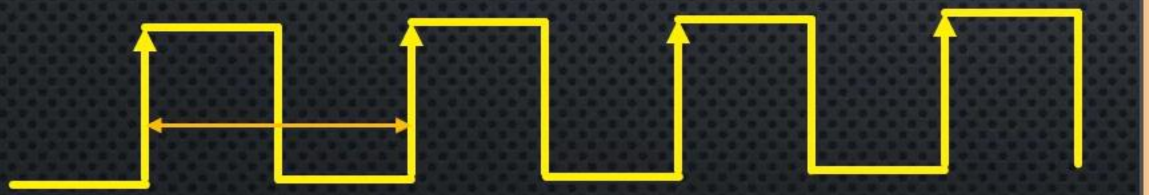


STATE TRANSITION DIAGRAM

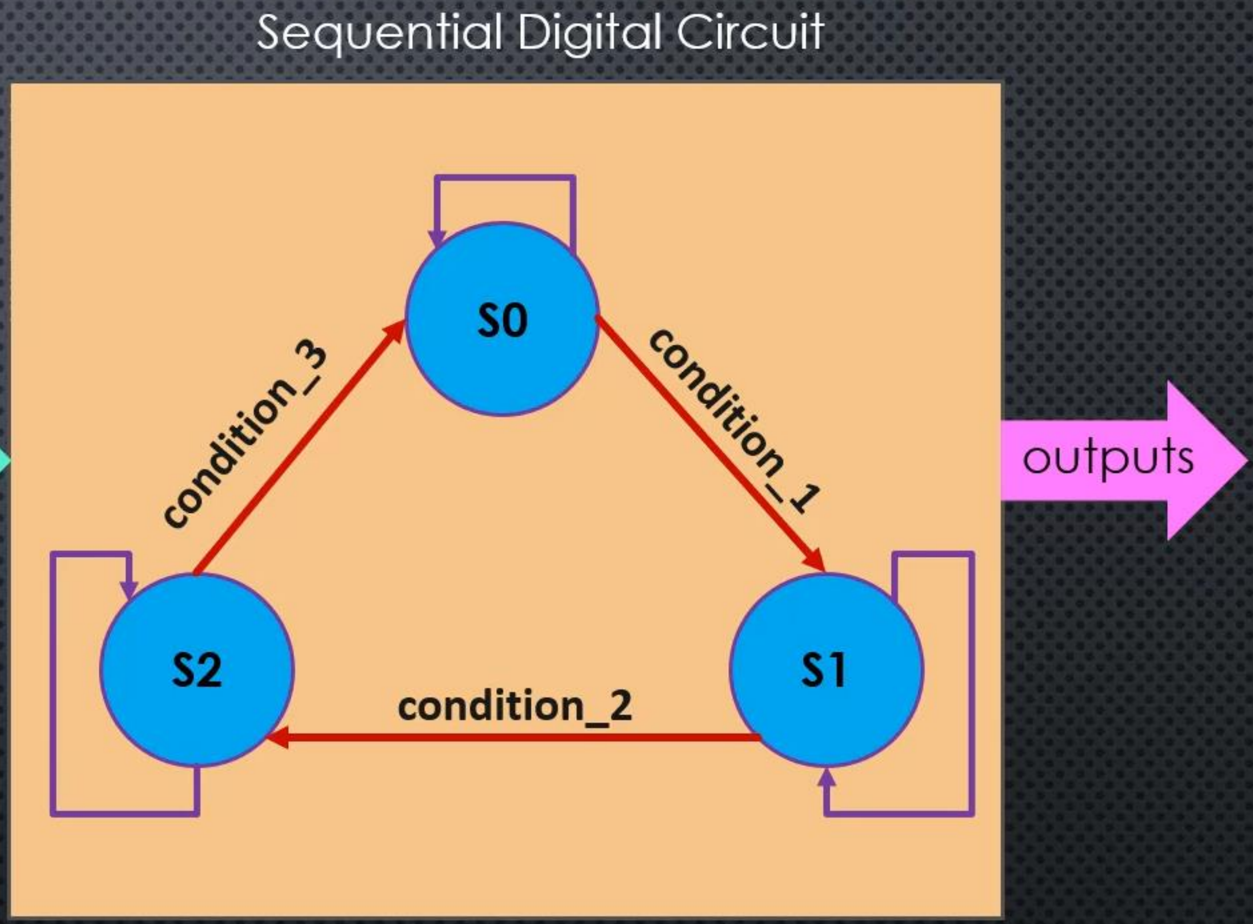
Code to unlock the door is 2, 3, 4, 6

FSM - CONCEPTS

- **States**
- **Transitions**



inputs



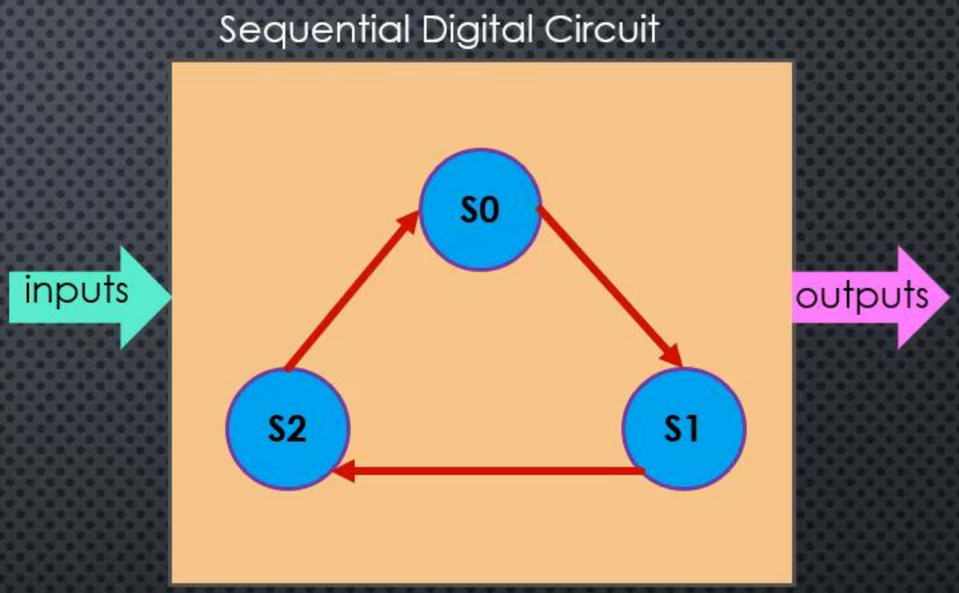
In S0 → *if* (condition_1) go to S1 else stay in S0

In S1→ *if* (condition_2) go to S2 else stay in S1

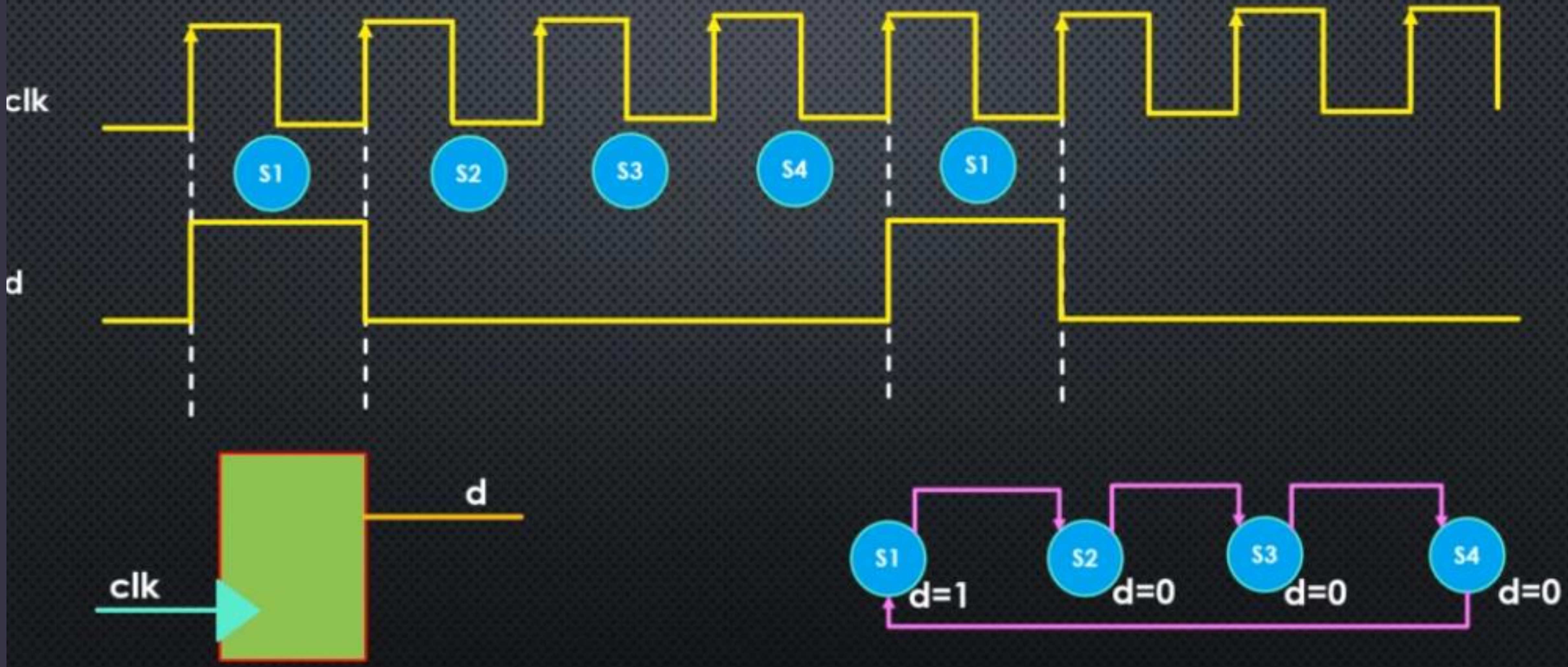
In S2→ *if* (condition_3) go to S0 else stay in S2

FSM - ELEMENTS

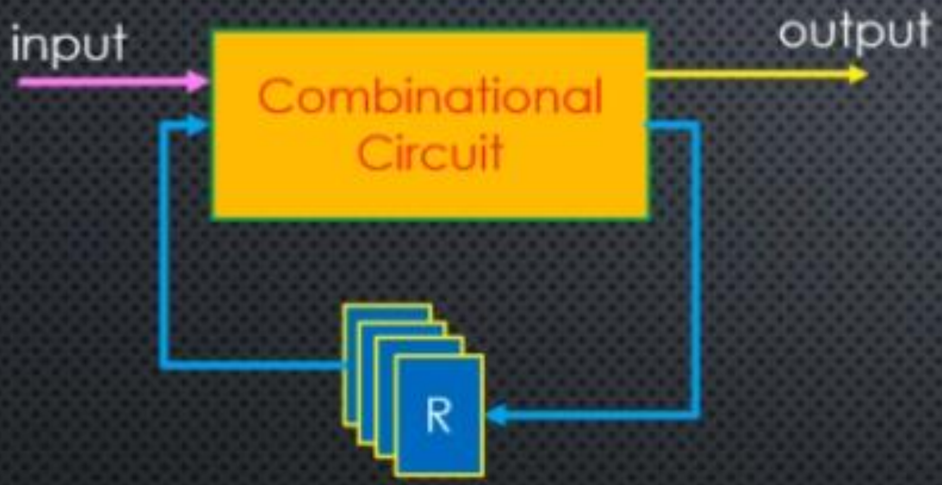
- **Registers**
- **Conditions**
- **Conditional Statement**



FSM - EXAMPLE

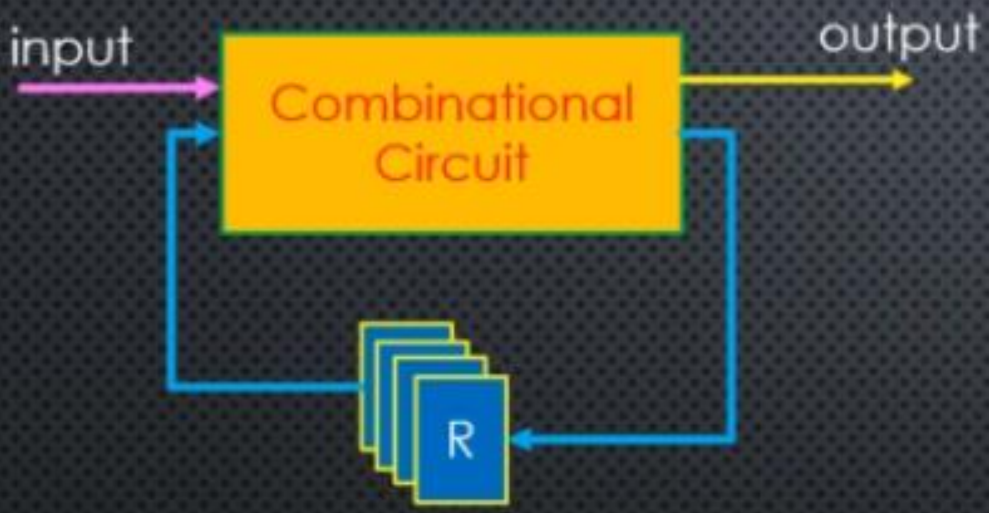


FSM TEMPLATE IN HLS



```
void function_name(arguments) {

}
```

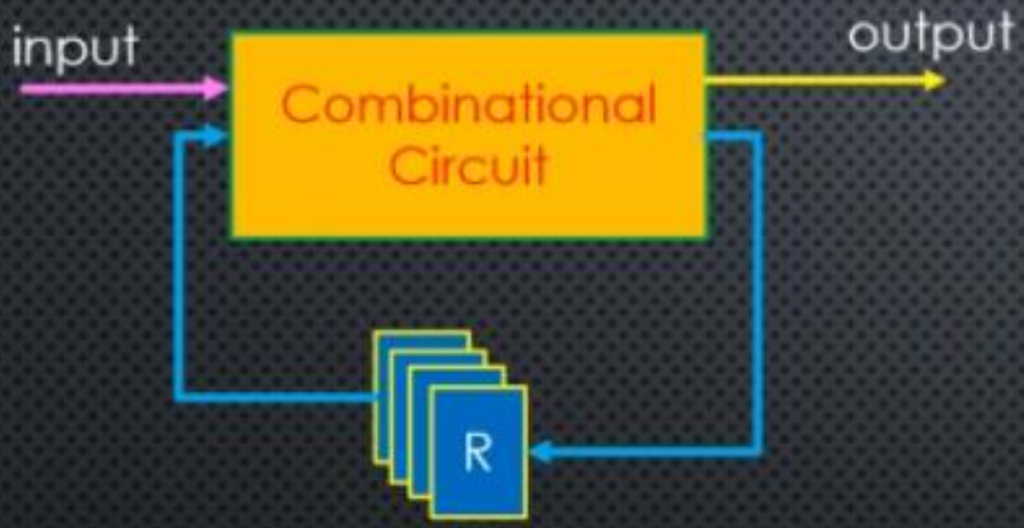
FSM TEMPLATE IN HLS



enumerate type to define list of states

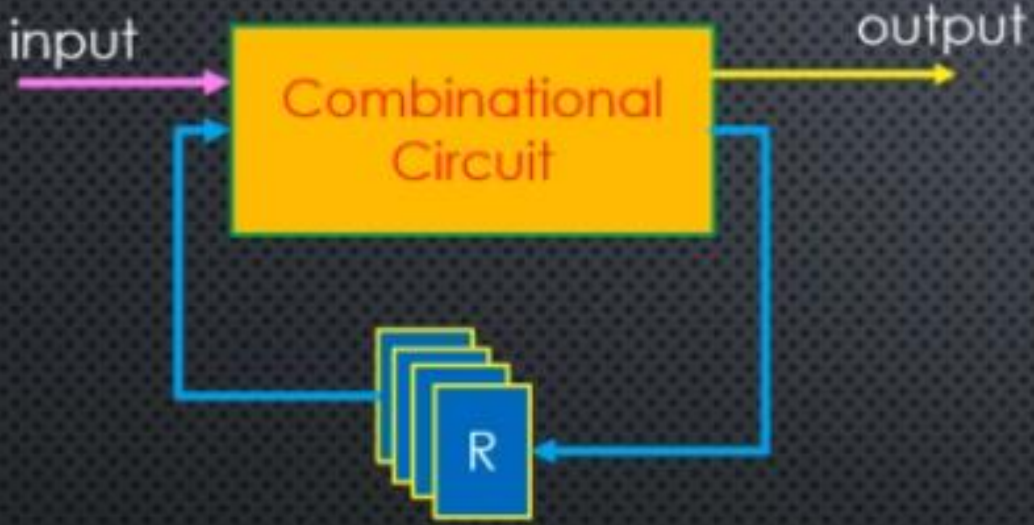
```
void function_name(arguments) {

}
```

FSM TEMPLATE IN HLS



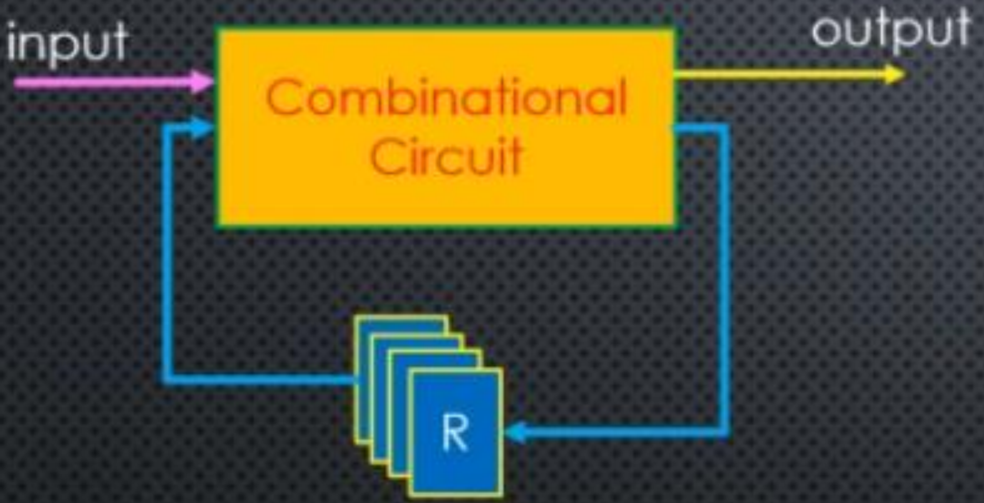
```
enumerate type to define list of states
void function_name(arguments) {
    states by defining static variables
}
```

FSM TEMPLATE IN HLS



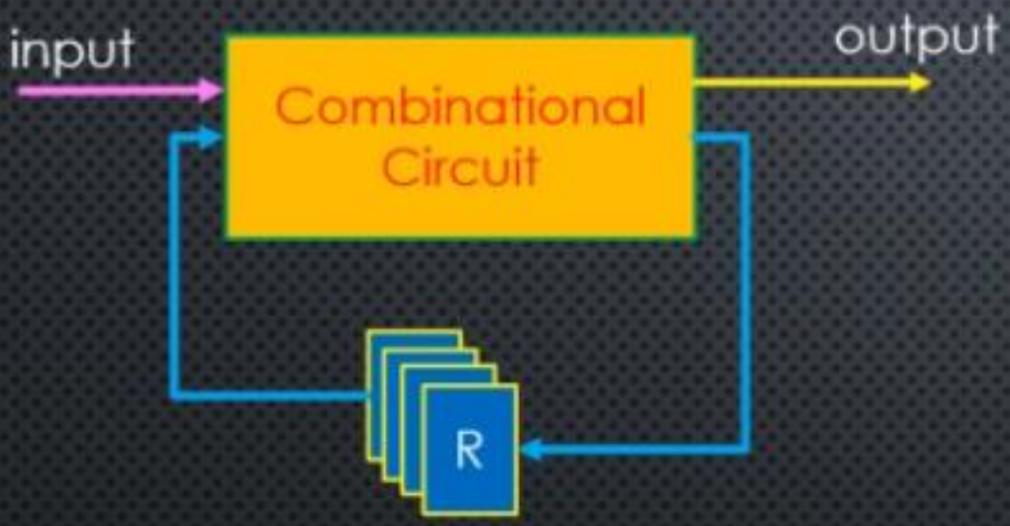
```
enumerate type to define list of states

void function_name(arguments) {
    states by defining static variables
    next-state variable definition
}
```

FSM TEMPLATE IN HLS



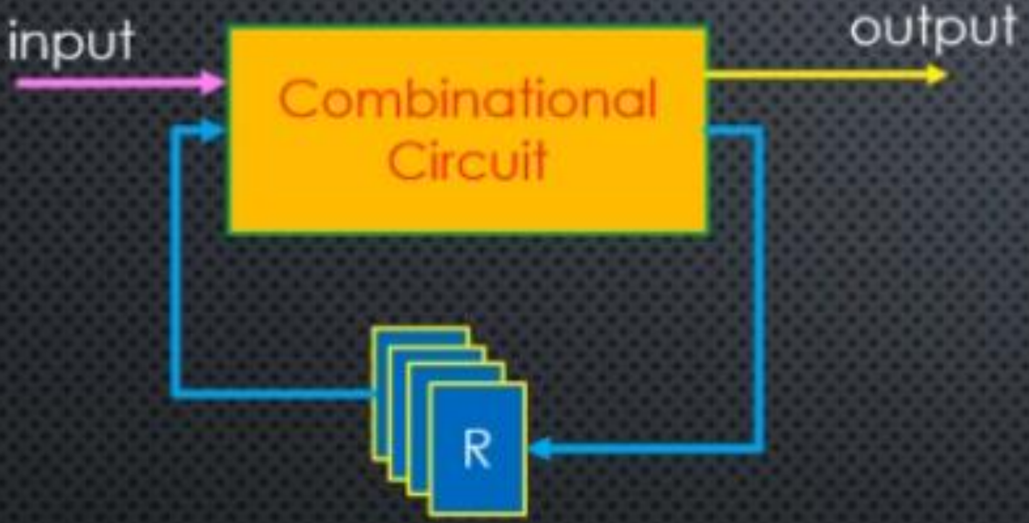
```
enumerate type to define list of states

void function_name(arguments) {
        states by defining static variables
        next-state variable definition
        output-local variable definition








}
```

FSM TEMPLATE IN HLS



```
enumerate type to define list of states

void function_name(arguments) {
    states by defining static variables
    next-state variable definition
    output-local variable definition

    switch-case statement to
    implement all transitions

}
```

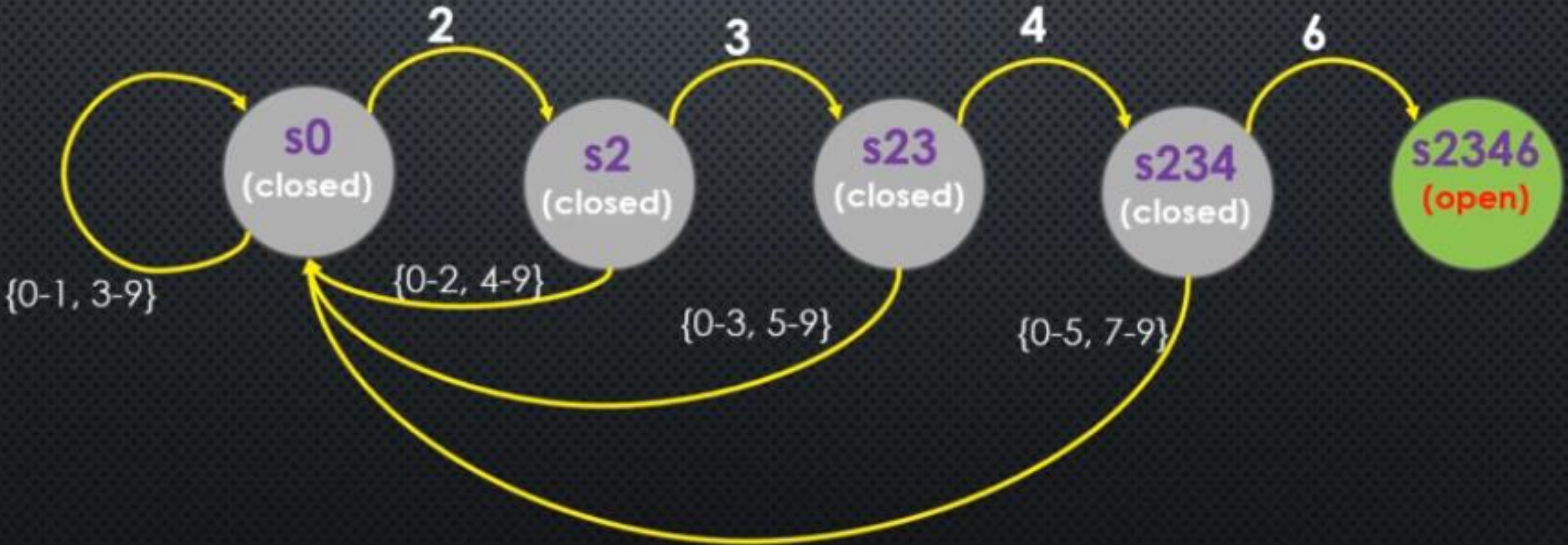
FSM TEMPLATE IN HLS



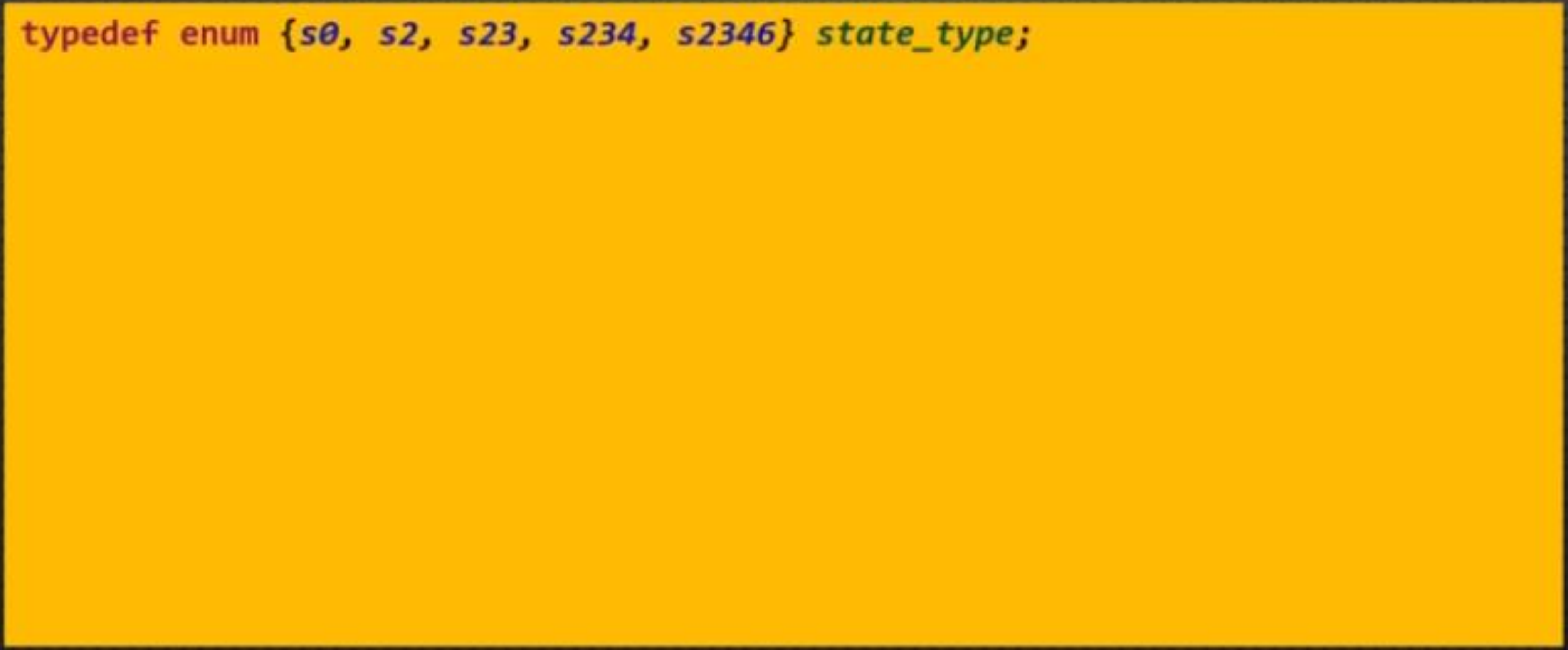
```
enumerate type to define list of states

void function_name(arguments) {
    states by defining static variables
    next-state variable definition
    output-local variable definition

    switch-case statement to
    implement all transitions

    Update states
    assign output-variables to output
    arguments
}
```

SEQUENCE FINDER

Code to unlock the door is 2, 3, 4, 6



SEQUENCE FINDER HLS CODE



```
typedef enum {s0, s2, s23, s234, s2346} state_type;
```

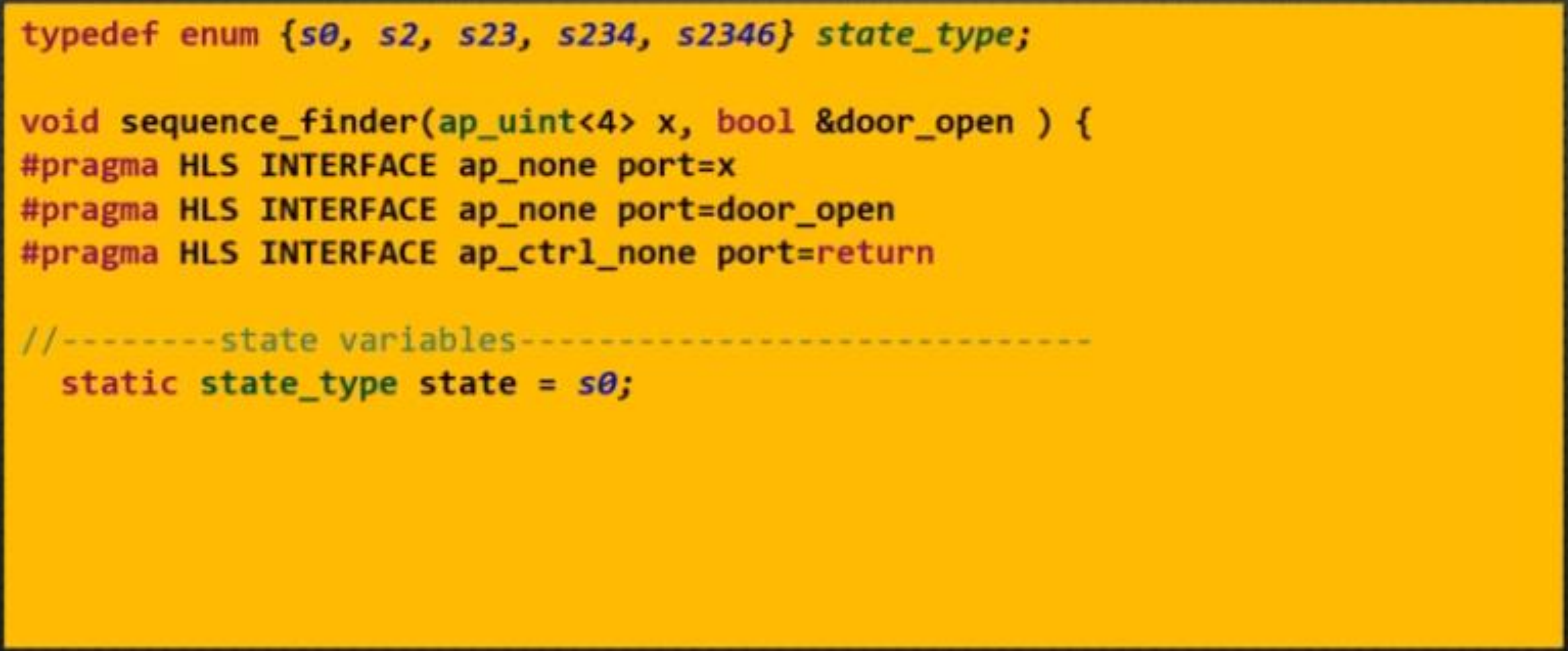
SEQUENCE FINDER HLS CODE

```
typedef enum {s0, s2, s23, s234, s2346} state_type;

void sequence_finder(ap_uint<4> x, bool &door_open ) {
```

SEQUENCE FINDER HLS CODE

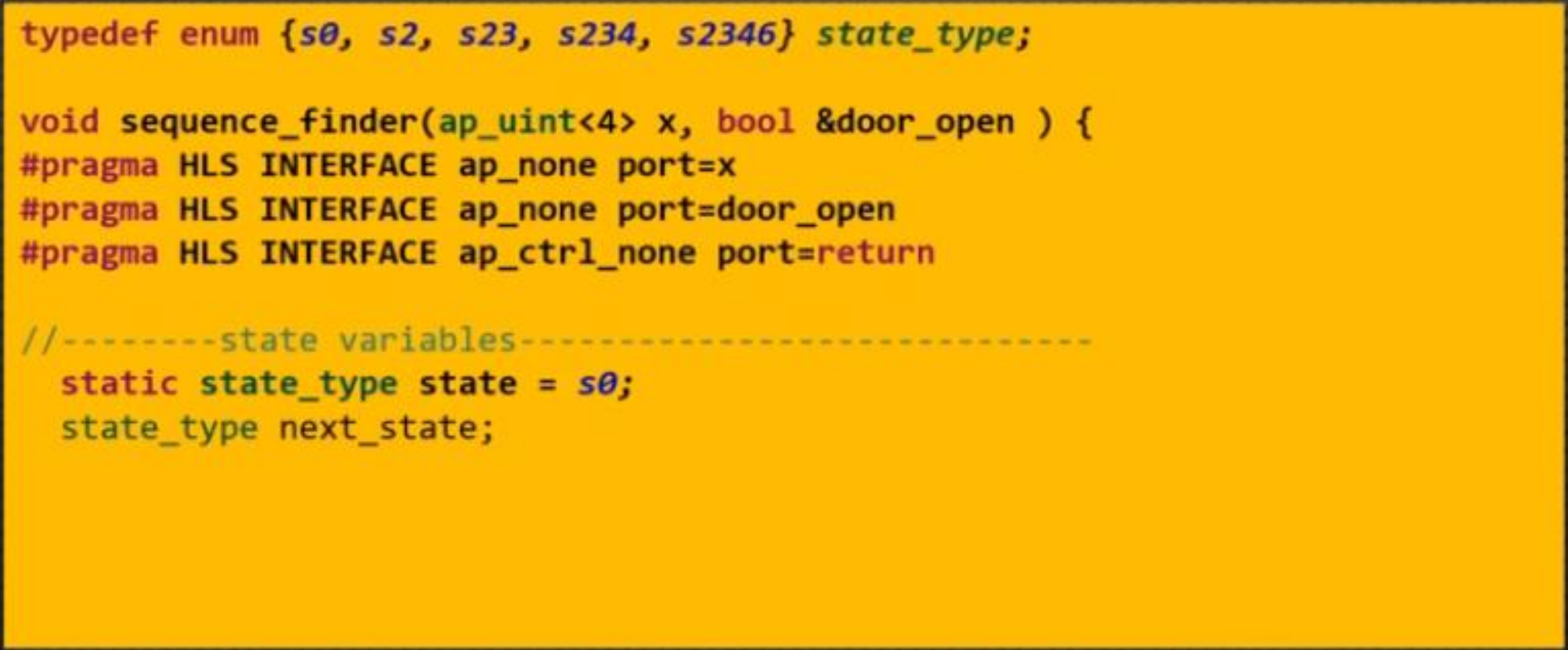
```
typedef enum {s0, s2, s23, s234, s2346} state_type;

void sequence_finder(ap_uint<4> x, bool &door_open ) {
#pragma HLS INTERFACE ap_none port=x
#pragma HLS INTERFACE ap_none port=door_open
#pragma HLS INTERFACE ap_ctrl_none port=return
```

SEQUENCE FINDER HLS CODE



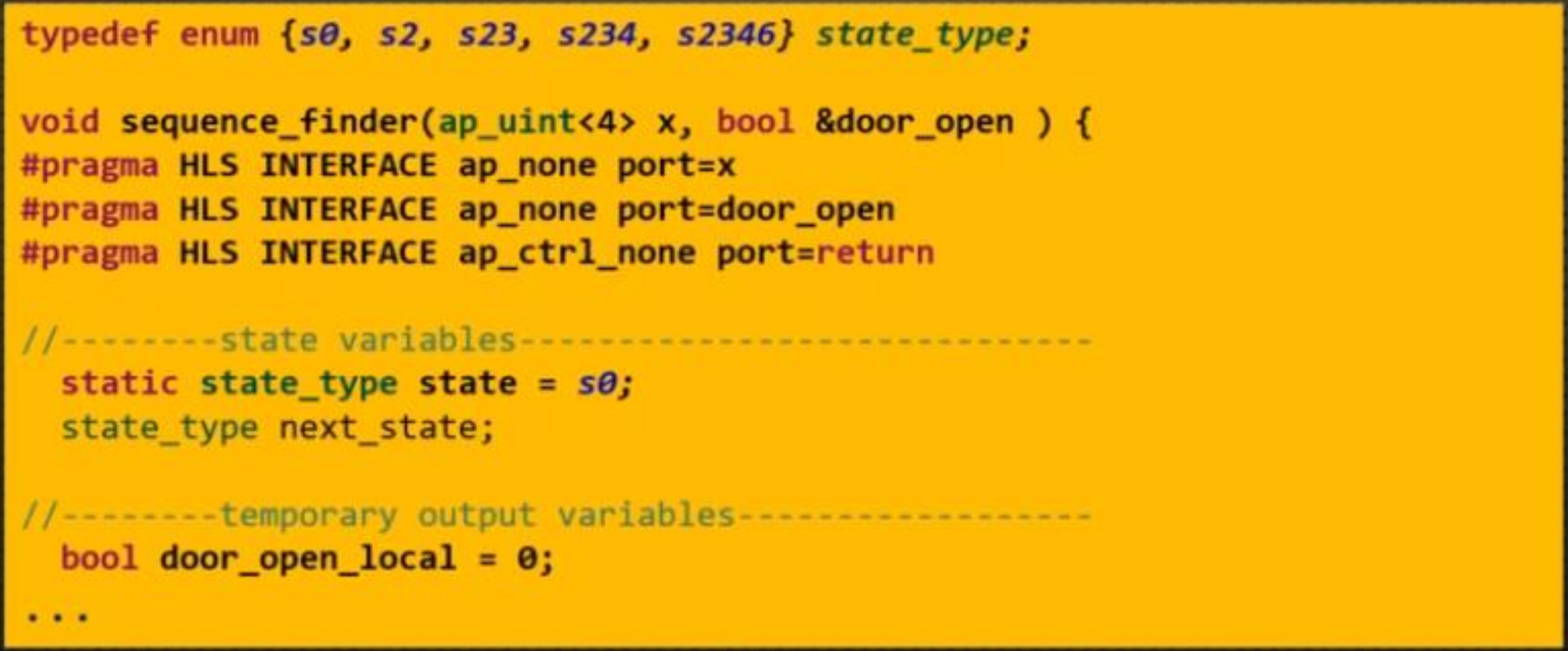
```
typedef enum {s0, s2, s23, s234, s2346} state_type;

void sequence_finder(ap_uint<4> x, bool &door_open ) {
#pragma HLS INTERFACE ap_none port=x
#pragma HLS INTERFACE ap_none port=door_open
#pragma HLS INTERFACE ap_ctrl_none port=return

//-------state variables----------------------------
  static state_type state = s0;
```

SEQUENCE FINDER HLS CODE



```
typedef enum {s0, s2, s23, s234, s2346} state_type;

void sequence_finder(ap_uint<4> x, bool &door_open ) {
#pragma HLS INTERFACE ap_none port=x
#pragma HLS INTERFACE ap_none port=door_open
#pragma HLS INTERFACE ap_ctrl_none port=return

//-------state variables-----------------------------
  static state_type state = s0;
  state_type next_state;
```

SEQUENCE FINDER HLS CODE



```
typedef enum {s0, s2, s23, s234, s2346} state_type;

void sequence_finder(ap_uint<4> x, bool &door_open ) {
#pragma HLS INTERFACE ap_none port=x
#pragma HLS INTERFACE ap_none port=door_open
#pragma HLS INTERFACE ap_ctrl_none port=return

//--------state variables----------------------------
  static state_type state = s0;
  state_type next_state;

//--------temporary output variables------------------
  bool door_open_local = 0;
...
```

SEQUENCE FINDER HLS CODE

```
//--------switch case--------------
  switch(state) {
  case s0:
    if (x == 2) {
      next_state = s2;
    } else {
      next_state = s0;
    }
    door_open_local = 0;
    break;
  case s2:




    break;
```

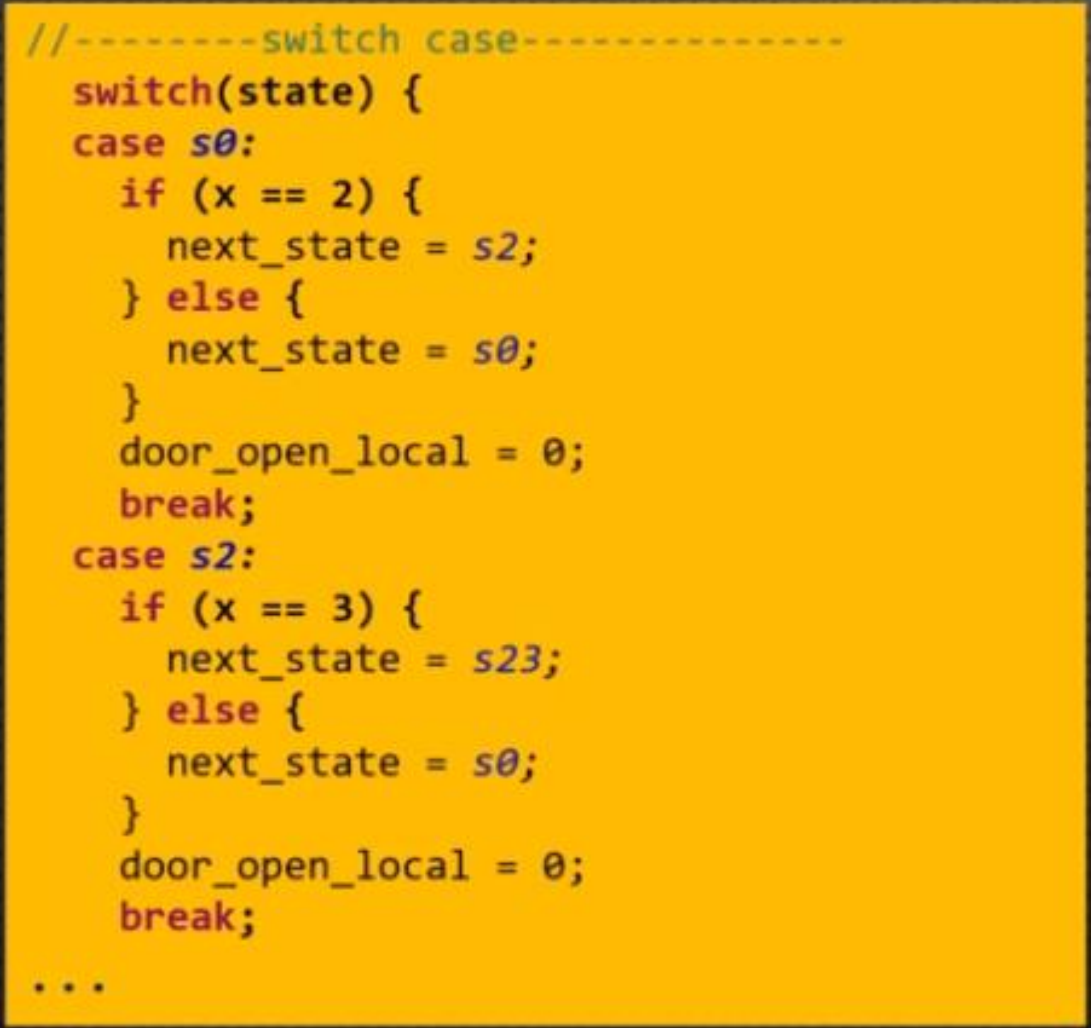
```
  case s23:




    break;
  case s234:




    break;
  case s2346:

    break;
  default:
    break;
}
```

SEQUENCE FINDER HLS CODE

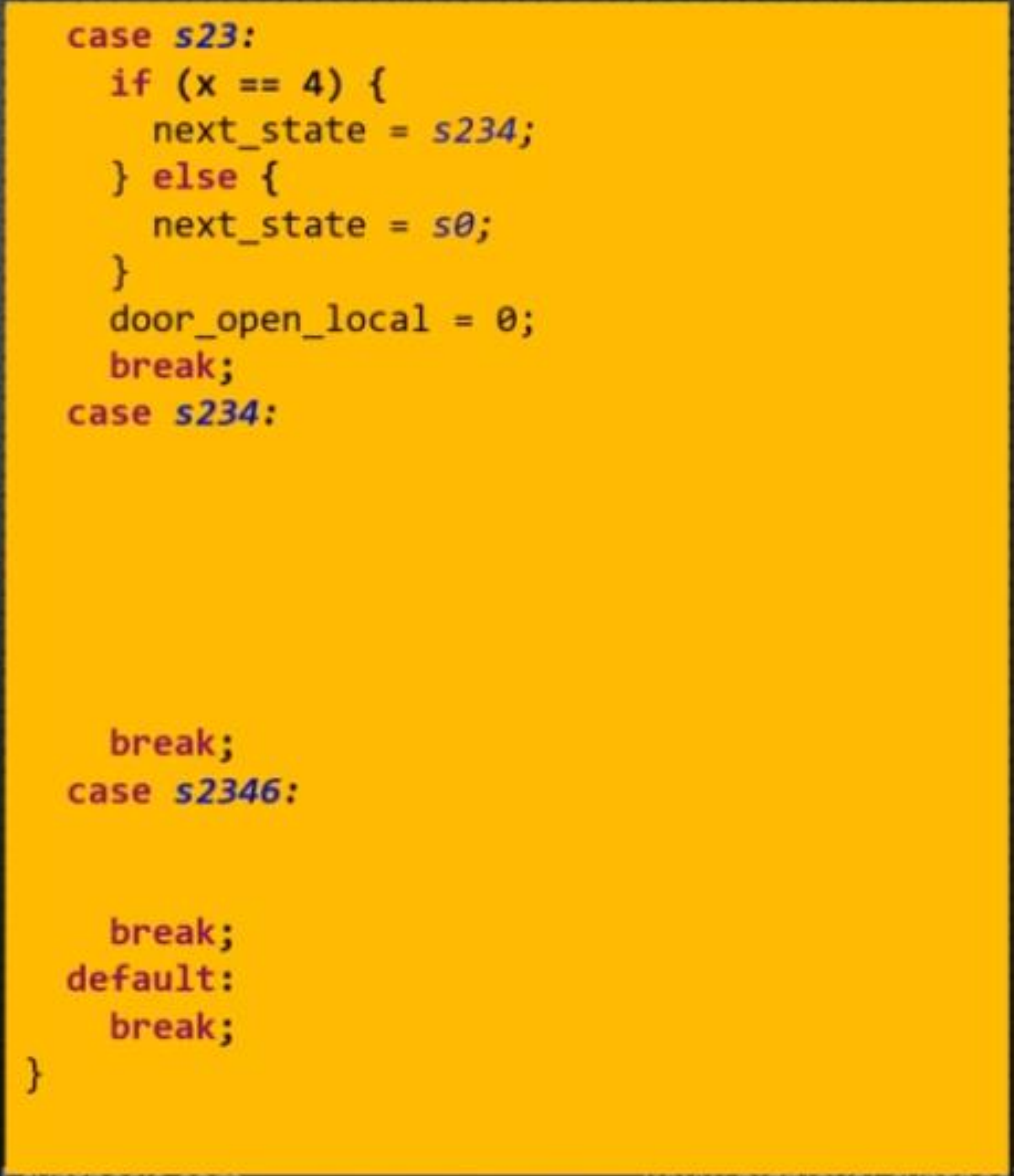
```
//--------switch case--------------
  switch(state) {
  case s0:
    if (x == 2) {
      next_state = s2;
    } else {
      next_state = s0;
    }
    door_open_local = 0;
    break;
  case s2:
    if (x == 3) {
      next_state = s23;
    } else {
      next_state = s0;
    }
    door_open_local = 0;
    break;
```

```
  case s23:







    break;
  case s234:







    break;
  case s2346:



    break;
  default:
    break;
}
```

SEQUENCE FINDER HLS CODE

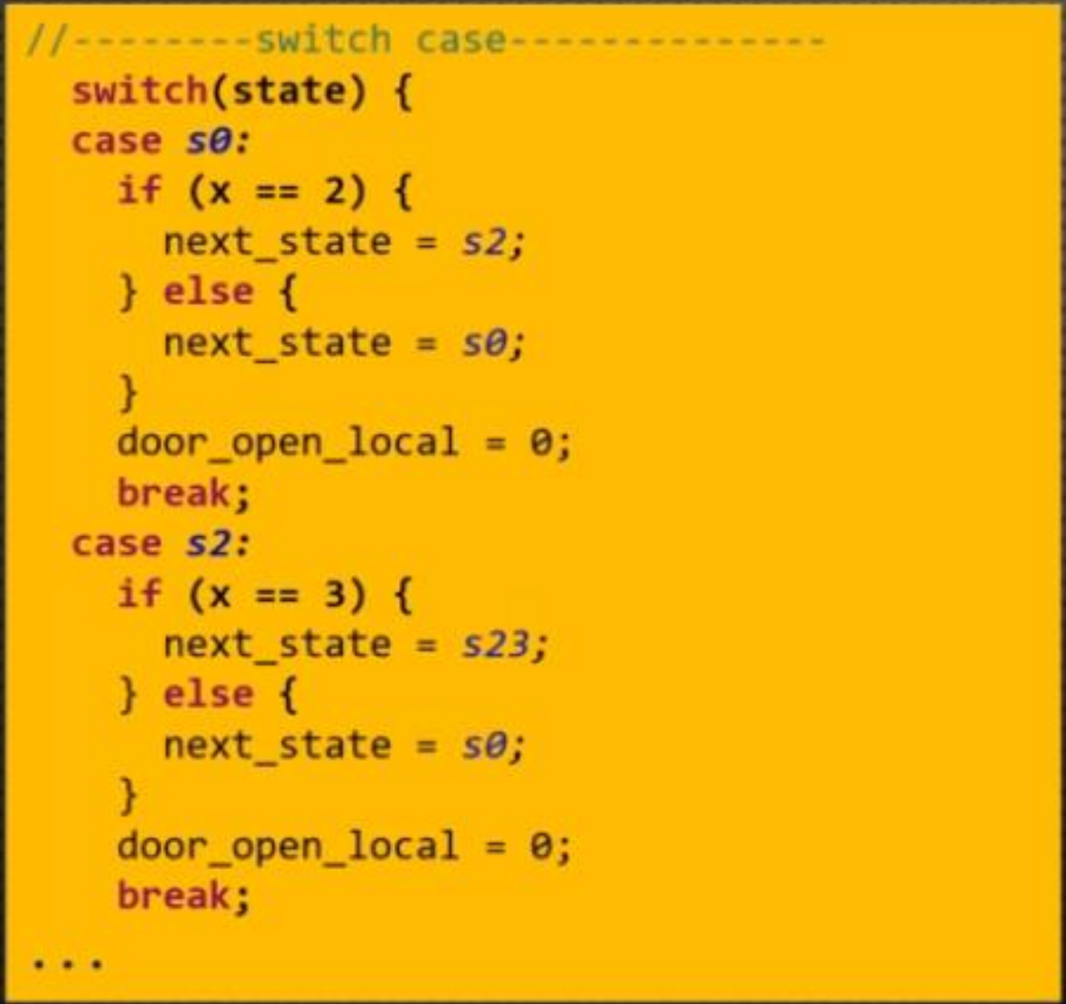


```
//--------switch case-------------
 switch(state) {
 case s0:
  if (x == 2) {
   next_state = s2;
  } else {
   next_state = s0;
  }
  door_open_local = 0;
  break;
 case s2:
  if (x == 3) {
   next_state = s23;
  } else {
   next_state = s0;
  }
  door_open_local = 0;
  break;
...
```

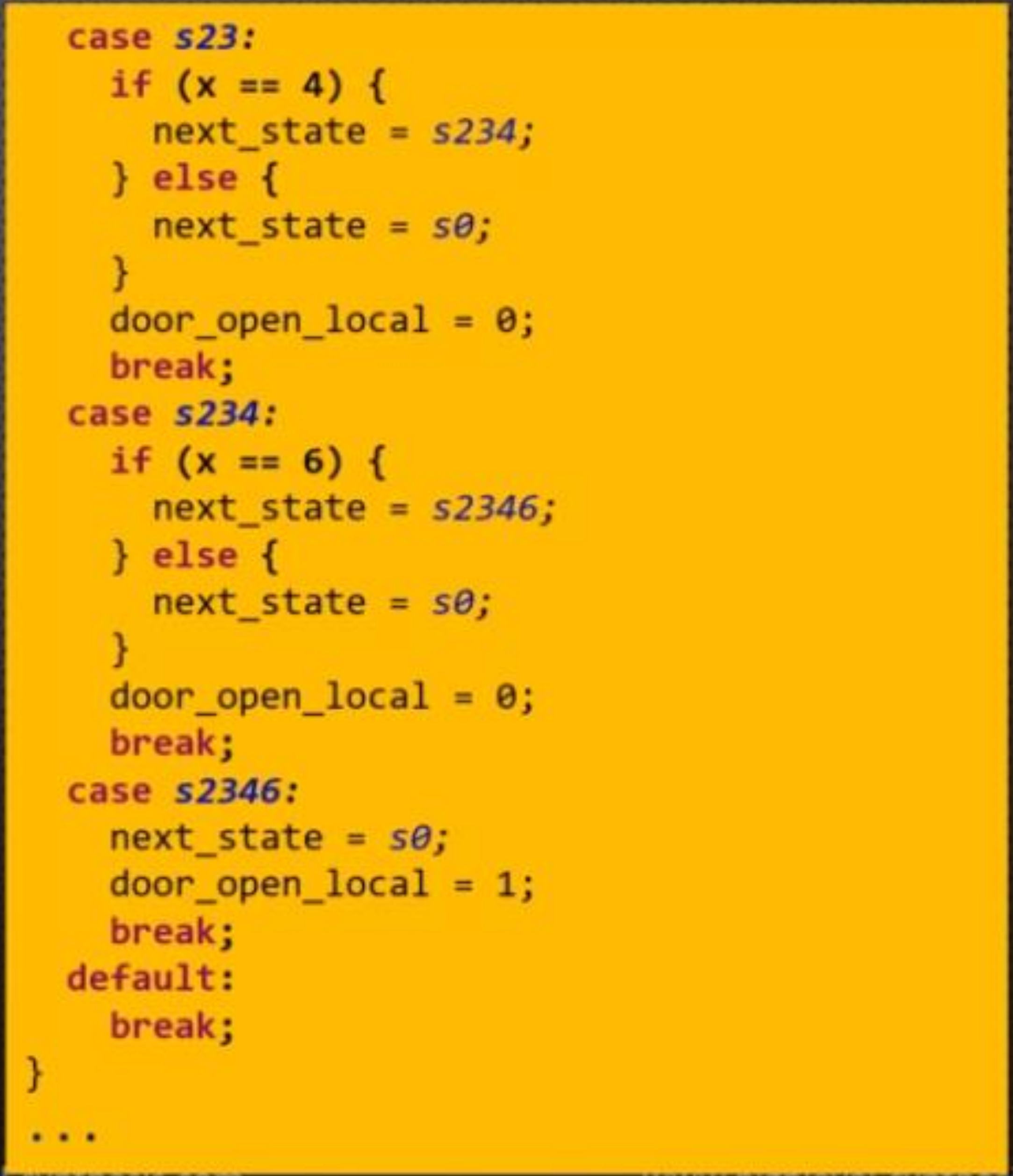


```
 case s23:
  if (x == 4) {
   next_state = s234;
  } else {
   next_state = s0;
  }
  door_open_local = 0;
  break;
 case s234:






  break;
 case s2346:


  break;
 default:
  break;
}
```

SEQUENCE FINDER HLS CODE

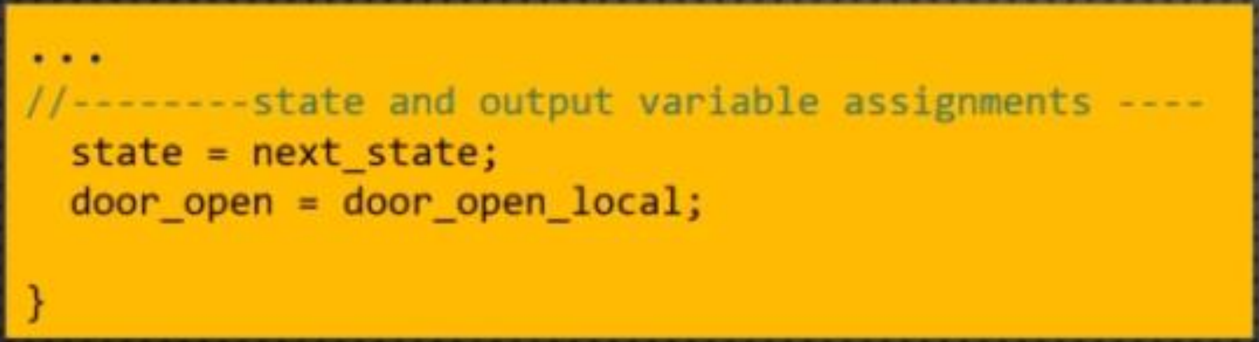


```
//--------switch case--------------
  switch(state) {
  case s0:
    if (x == 2) {
      next_state = s2;
    } else {
      next_state = s0;
    }
    door_open_local = 0;
    break;
  case s2:
    if (x == 3) {
      next_state = s23;
    } else {
      next_state = s0;
    }
    door_open_local = 0;
    break;
...
```



```
  case s23:
    if (x == 4) {
      next_state = s234;
    } else {
      next_state = s0;
    }
    door_open_local = 0;
    break;
  case s234:
    if (x == 6) {
      next_state = s2346;
    } else {
      next_state = s0;
    }
    door_open_local = 0;
    break;
  case s2346:
    next_state = s0;
    door_open_local = 1;
    break;
  default:
    break;
}
...
```

SEQUENCE FINDER HLS CODE

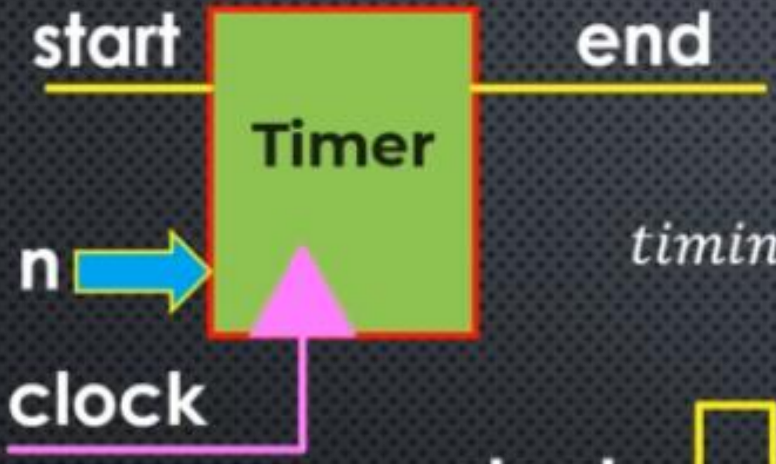


```
...
//--------state and output variable assignments ----
  state = next_state;
  door_open = door_open_local;

}
```

TIMER

Lecture-10

TIMER

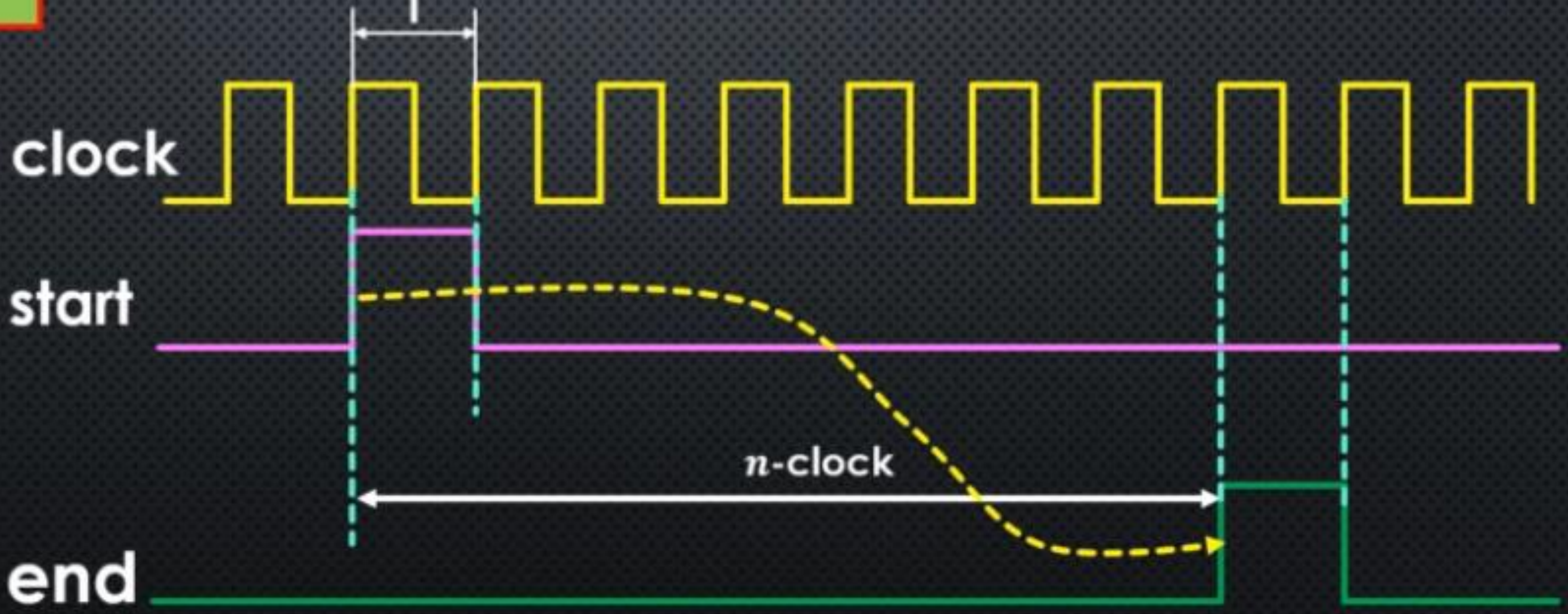


T=10 ns

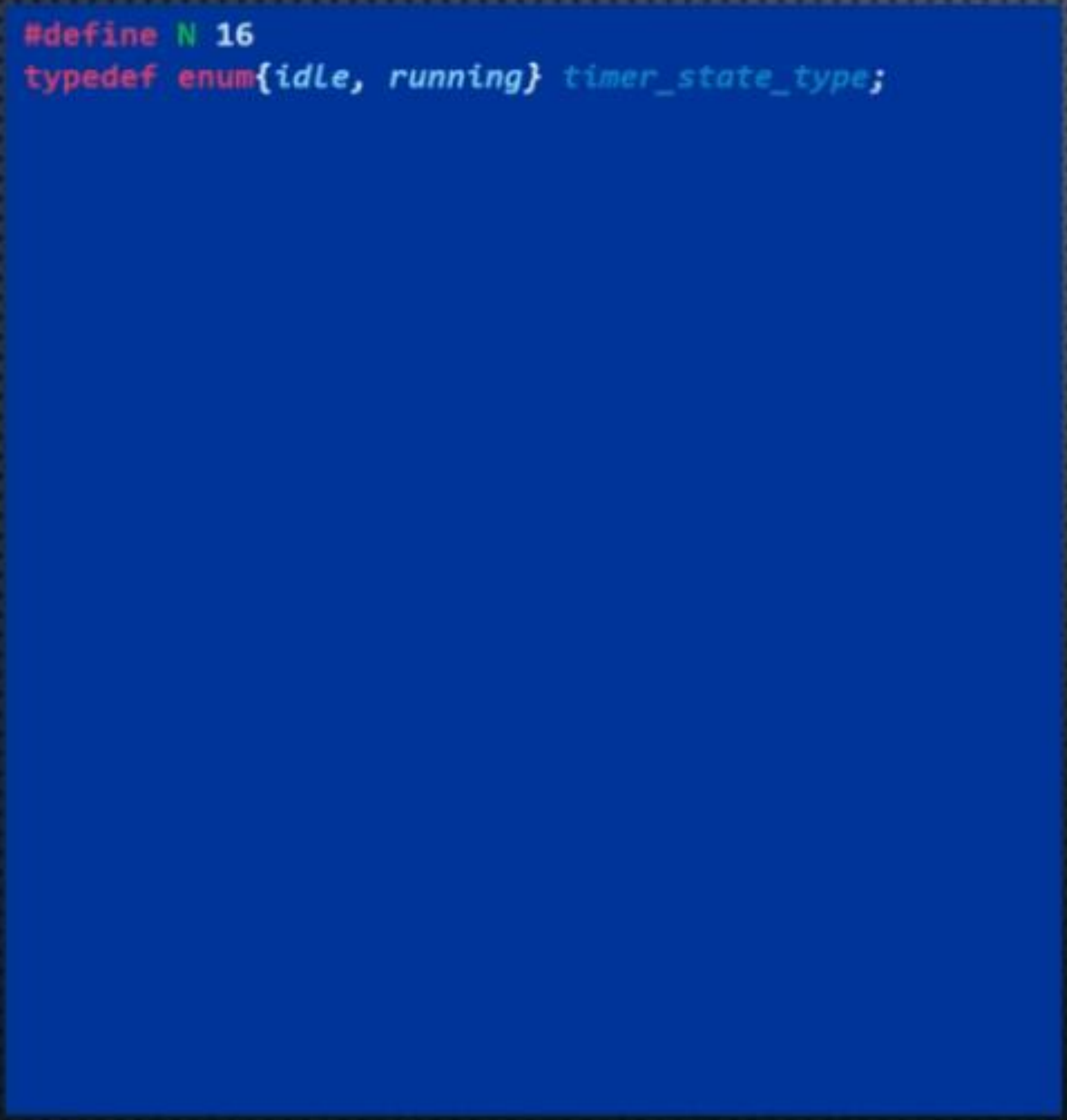
$n = 1{,}000{,}000$

$timing\ interval = 10ms$

$timing\ interval = nT$

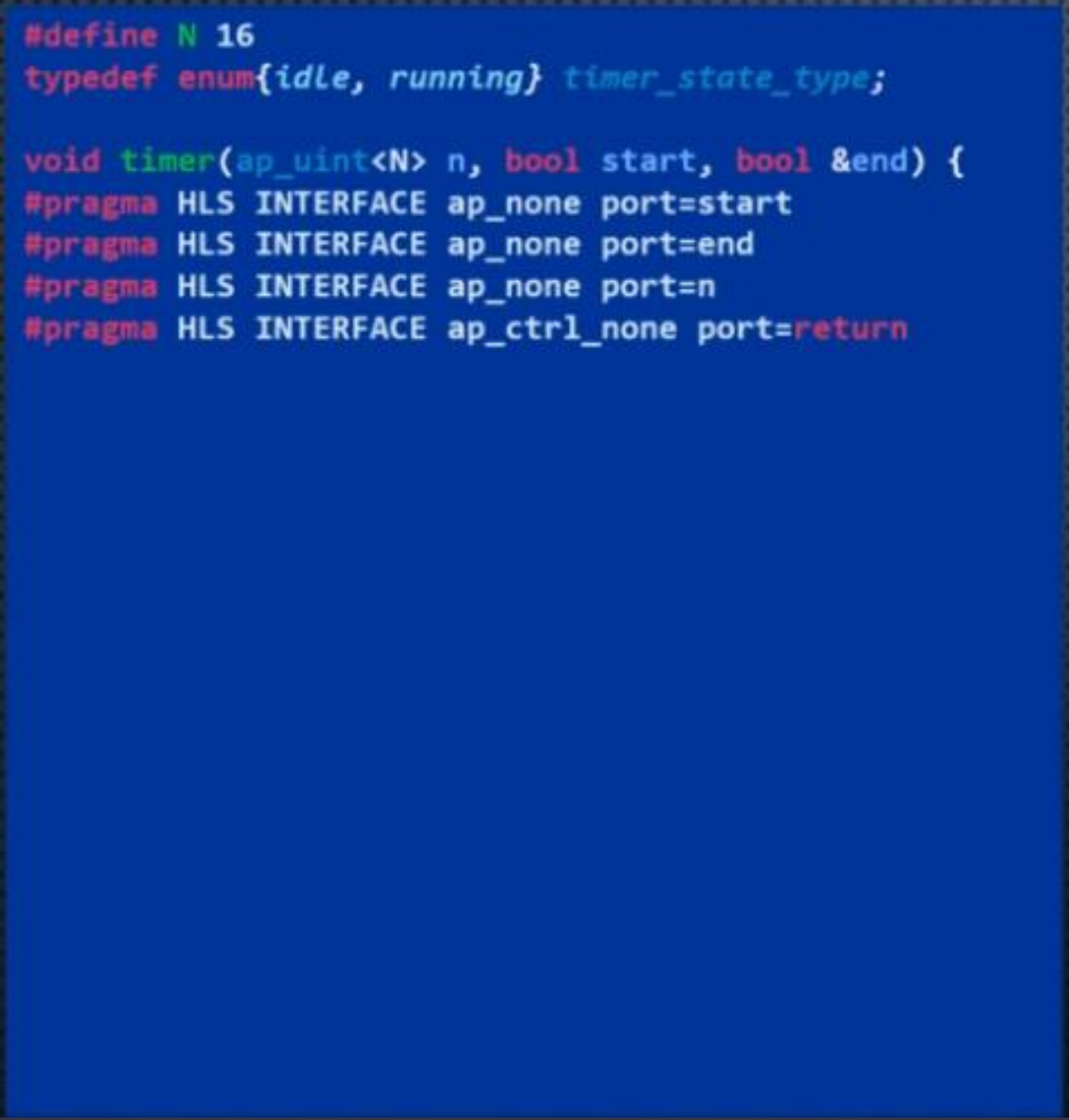


TIMER - HLS



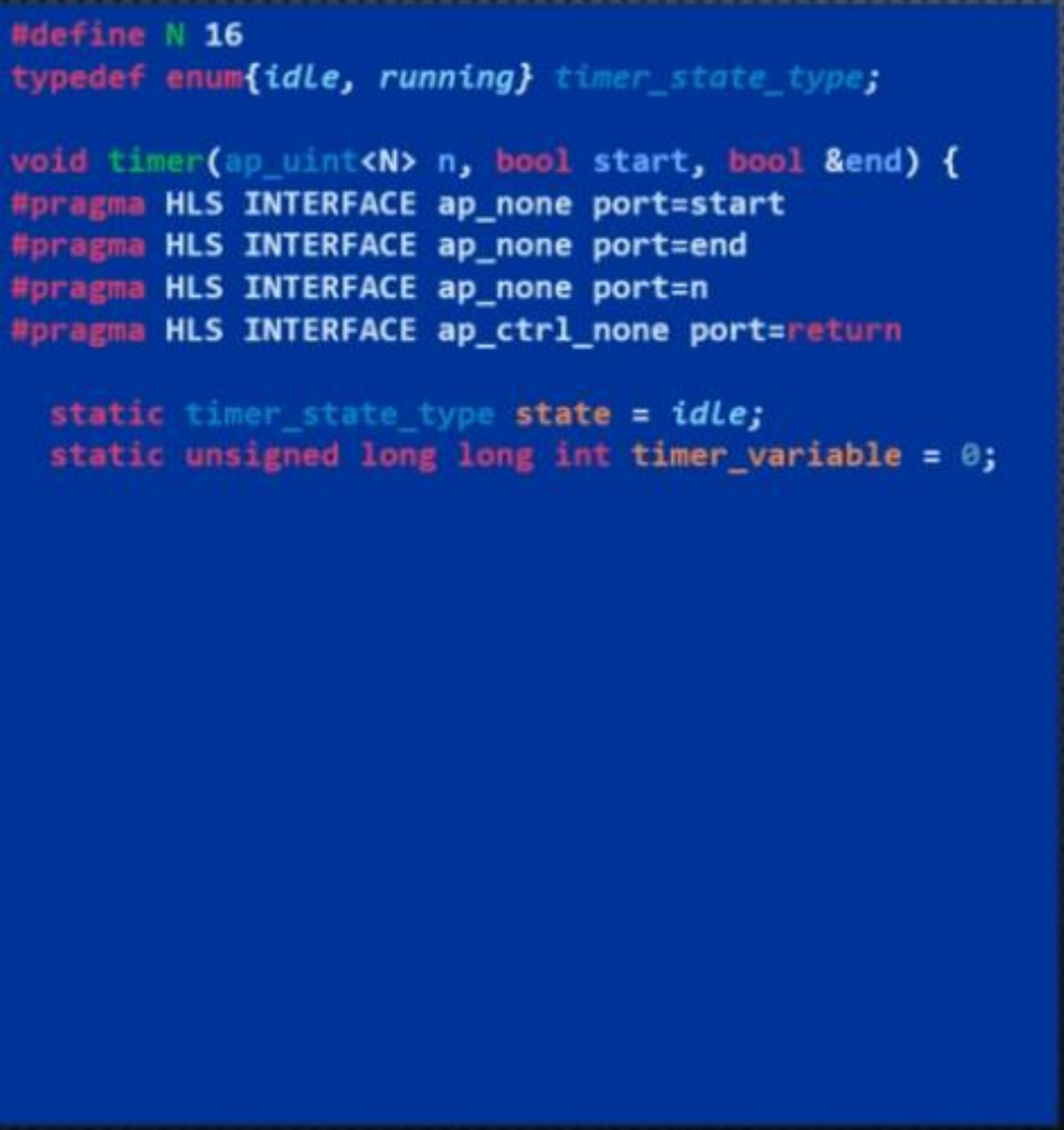
```
#define N 16
typedef enum{idle, running} timer_state_type;
```

TIMER - HLS



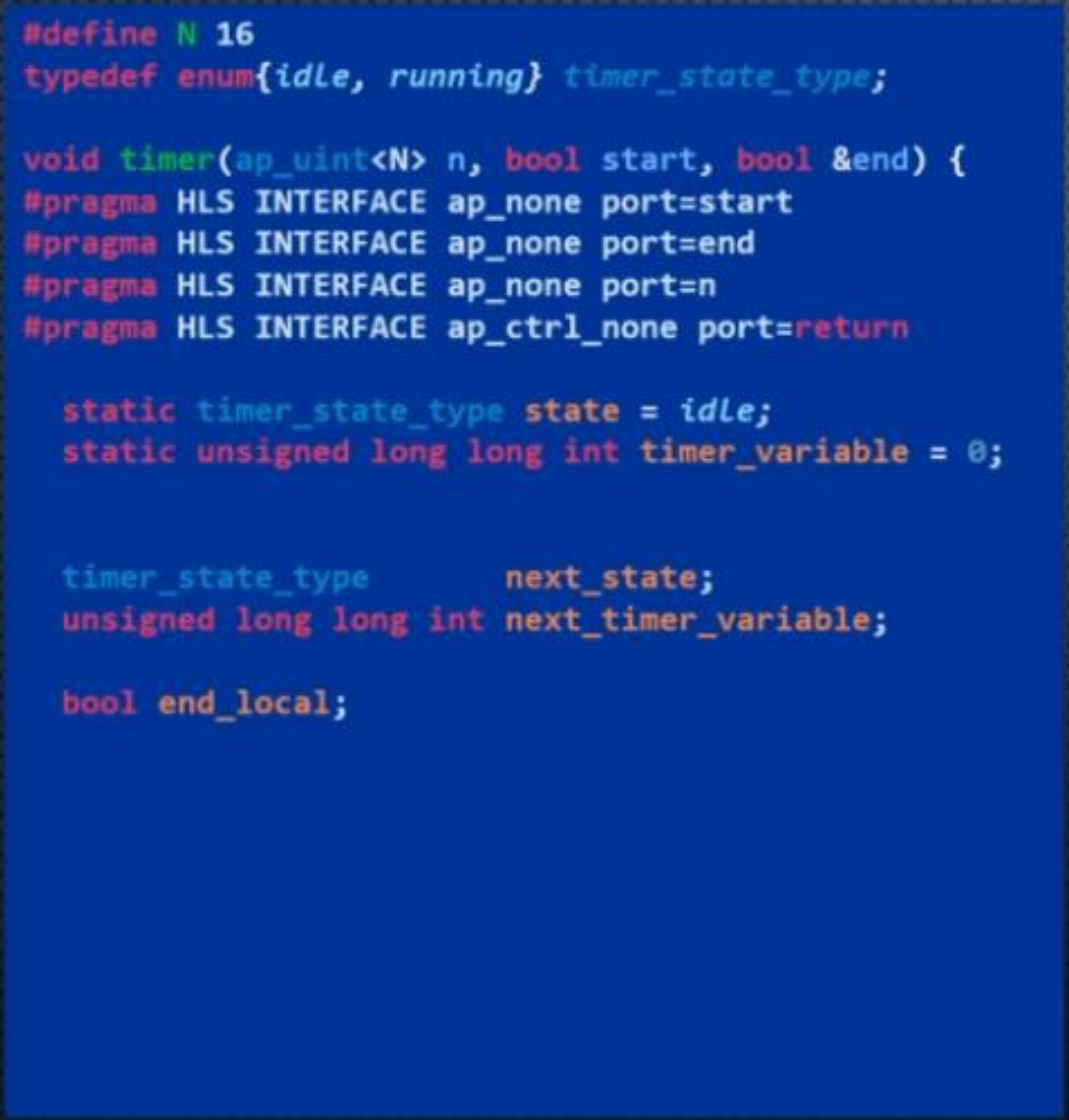
```
#define N 16
typedef enum{idle, running} timer_state_type;

void timer(ap_uint<N> n, bool start, bool &end) {
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_none port=end
#pragma HLS INTERFACE ap_none port=n
#pragma HLS INTERFACE ap_ctrl_none port=return
```

TIMER - HLS



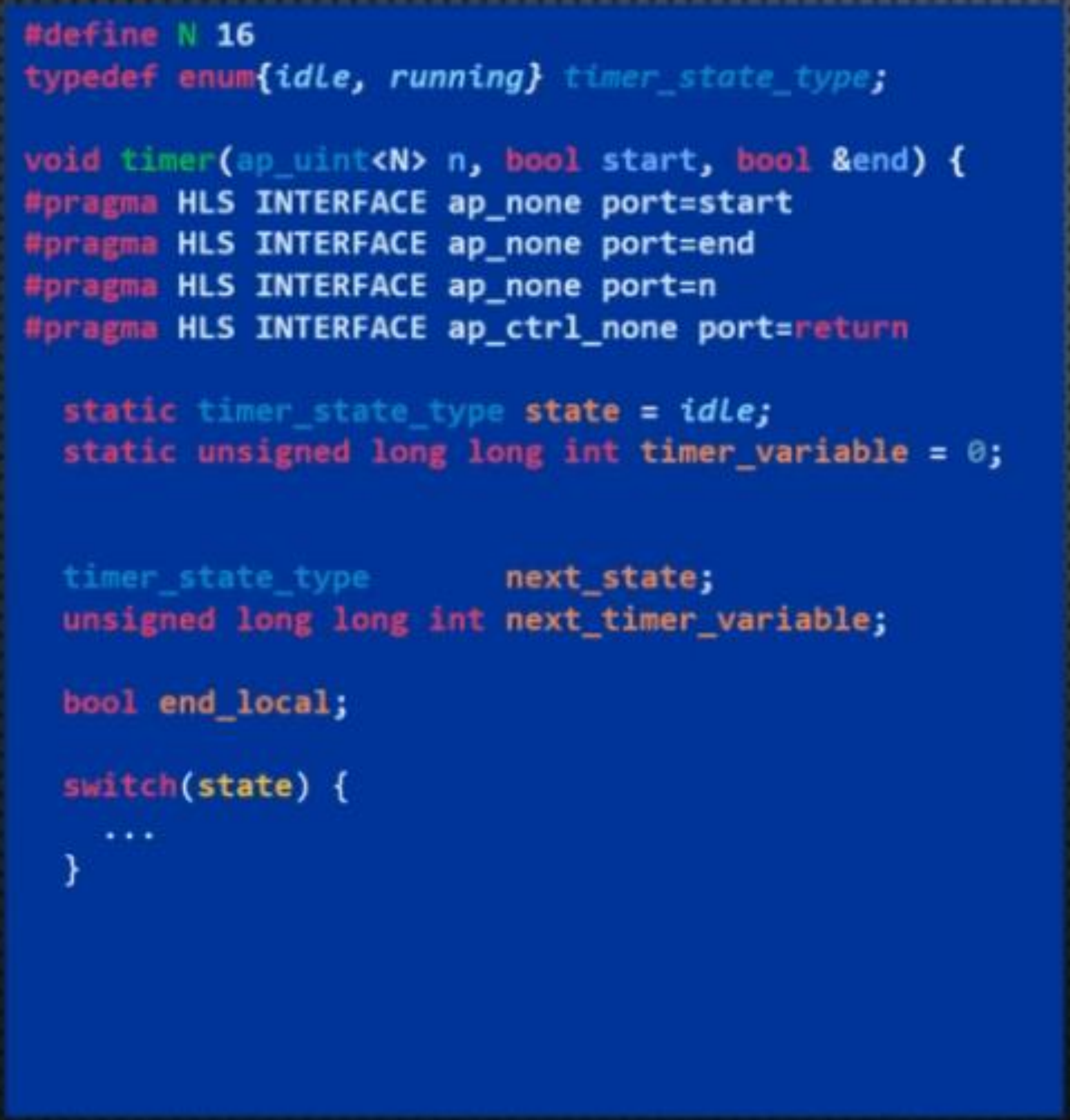
```
#define N 16
typedef enum{idle, running} timer_state_type;

void timer(ap_uint<N> n, bool start, bool &end) {
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_none port=end
#pragma HLS INTERFACE ap_none port=n
#pragma HLS INTERFACE ap_ctrl_none port=return

  static timer_state_type state = idle;
  static unsigned long long int timer_variable = 0;
```

TIMER - HLS



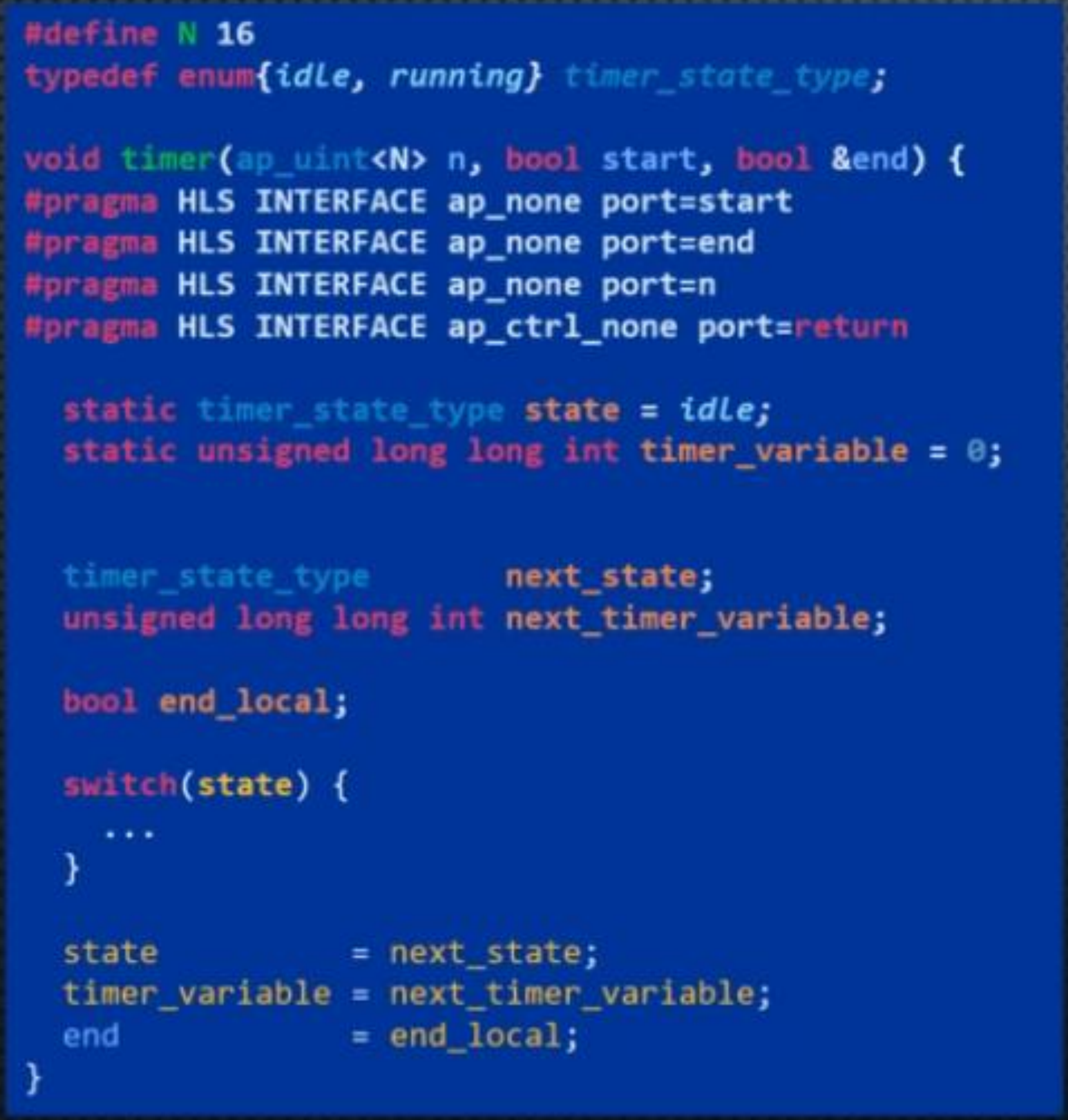
```
#define N 16
typedef enum{idle, running} timer_state_type;

void timer(ap_uint<N> n, bool start, bool &end) {
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_none port=end
#pragma HLS INTERFACE ap_none port=n
#pragma HLS INTERFACE ap_ctrl_none port=return

  static timer_state_type state = idle;
  static unsigned long long int timer_variable = 0;


  timer_state_type       next_state;
  unsigned long long int next_timer_variable;

  bool end_local;
```

TIMER - HLS



```
#define N 16
typedef enum{idle, running} timer_state_type;

void timer(ap_uint<N> n, bool start, bool &end) {
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_none port=end
#pragma HLS INTERFACE ap_none port=n
#pragma HLS INTERFACE ap_ctrl_none port=return

  static timer_state_type state = idle;
  static unsigned long long int timer_variable = 0;


  timer_state_type       next_state;
  unsigned long long int next_timer_variable;

  bool end_local;

  switch(state) {
    ...
  }
```

TIMER - HLS

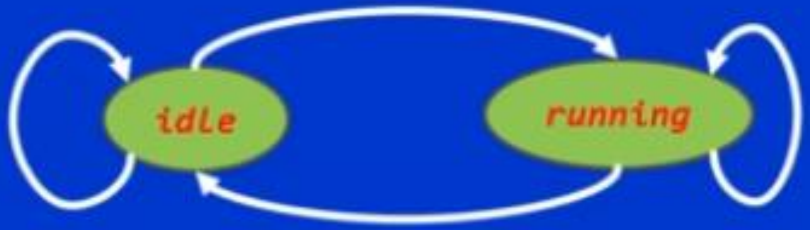


```
#define N 16
typedef enum{idle, running} timer_state_type;

void timer(ap_uint<N> n, bool start, bool &end) {
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_none port=end
#pragma HLS INTERFACE ap_none port=n
#pragma HLS INTERFACE ap_ctrl_none port=return

  static timer_state_type state = idle;
  static unsigned long long int timer_variable = 0;


  timer_state_type       next_state;
  unsigned long long int next_timer_variable;

  bool end_local;

  switch(state) {
    ...
  }

  state          = next_state;
  timer_variable = next_timer_variable;
  end            = end_local;
}
```

TIMER - HLS

```
#define N 16
typedef enum{idle, running} timer_state_type;

void timer(ap_uint<N> n, bool start, bool &end) {
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_none port=end
#pragma HLS INTERFACE ap_none port=n
#pragma HLS INTERFACE ap_ctrl_none port=return

  static timer_state_type state = idle;
  static unsigned long long int timer_variable = 0;


  timer_state_type       next_state;
  unsigned long long int next_timer_variable;

  bool end_local;

  switch(state) {
    ...
  }

  state          = next_state;
  timer_variable = next_timer_variable;
  end            = end_local;
}
```

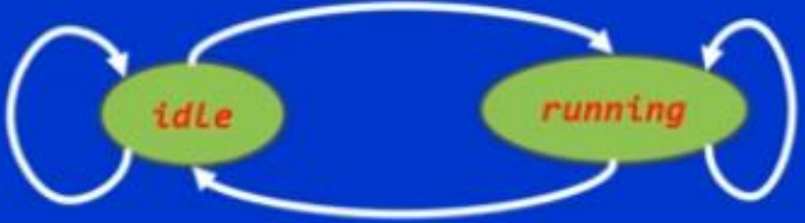
```
switch(state) {
case idle:
  if (start == 1) {
    next_state     = running;
    end_local = 0;
    next_timer_variable = 0;
  } else {
    next_state     = idle;
    end_local = 0;
    next_timer_variable = 0;
  }
  break;
case running:



  break;
default:
  break;
}
```



TIMER - HLS

```
#define N 16
typedef enum{idle, running} timer_state_type;

void timer(ap_uint<N> n, bool start, bool &end) {
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_none port=end
#pragma HLS INTERFACE ap_none port=n
#pragma HLS INTERFACE ap_ctrl_none port=return

  static timer_state_type state = idle;
  static unsigned long long int timer_variable = 0;


  timer_state_type       next_state;
  unsigned long long int next_timer_variable;

  bool end_local;

  switch(state) {
    ...
  }

  state          = next_state;
  timer_variable = next_timer_variable;
  end            = end_local;
}
```

```
  switch(state) {
  case idle:
    if (start == 1) {
      next_state     = running;
      end_local = 0;
      next_timer_variable = 0;
    } else {
      next_state     = idle;
      end_local = 0;
      next_timer_variable = 0;
    }
    break;
  case running:
    if (timer_variable == n-1) {
      next_state          = idle;
      end_local           = 1;
      next_timer_variable = 0;
    } else {
      next_timer_variable = timer_variable+1;
      next_state          = running;
      end_local           = 0;
    }
    break;
  default:
    break;
}
```



Any Question…

Thank you