CPRE 488
Embedded System Design
(VHDL Overview)

Instructor: Dr. Phillip Jones
(phjones@iastate.edu)

Reconfigurable Computing Laboratory
Iowa State University
Ames, Iowa, USA

http://class.ece.iastate.edu/cpre488/

VHDL basics

- VHDL: (V)HSIC (H)ardware (D)escription (L)anguage
  - VHSIC: (V)ery (H)igh (S)peed (I)ntegrated (C)ircuit

VHDL basics

- VHDL: (V)HSIC (H)ardware (D)escription (L)anguage
  – VHSIC: (V)ery (H)igh (S)peed (I)ntegrated (C)ircuit

- Golden Rules of Hardware Design (VHDL or Verilog)

1. VHDL is a Hardware **Description** Language (HDL)
   - VHDL is NOT a programming language
   - VHDL is conceptually VERY different than C/C++!

2. Draw your Hardware Circuit before writing ANY VHDL
   - Easier for you, and others to check for bugs at the circuit diagram.
   - A drawing gives a base from which you and other can check if the VHDL is reflecting the architecture envisioned.
   - The tools are not magic! If you cannot sketch your circuit using basic building blocks (e.g., MUXs, counters, state diagrams, etc.), then it is not reasonable to expect the tools to figure it out. Having no sketch is just asking for weird hardware behaviors to occur.

VHDL basics

- VHDL: (V)HSIC (H)ardware (D)escription (L)anguage
  - VHSIC: (V)ery (H)igh (S)peed (I)ntegrated (C)ircuit

- Golden Rules of Hardware Design (VHDL or Verilog)

1. VHDL is a Hardware **Description** Language (HDL)
   - VHDL is NOT a programming language
   - VHDL is conceptually VERY different than C/C++!

2. Draw your Hardware Circuit before writing ANY VHDL
   - Easier for you, and others to check for bugs at the circuit diagram.
   - A drawing gives a base from which you and other can check if the VHDL is reflecting the architecture envisioned.
   - The tools are not magic! If you cannot sketch your circuit using basic building blocks (e.g., MUXs, counters, state diagrams, etc.), then it is not reasonable to expect the tools to figure it out. Having no sketch is just asking for weird hardware behaviors to occur.

VHDL basics

- VHDL: (V)HSIC (H)ardware (D)escription (L)anguage
  - VHSIC: (V)ery (H)igh (S)peed (I)ntegrated (C)ircuit
- Golden Rules of Hardware Design (VHDL or Verilog)

1. VHDL is a Hardware **Description** Language (HDL)
   - VHDL is NOT a programming language
   - VHDL is conceptually VERY different than C/C++!
2. Draw your Hardware Circuit before writing ANY VHDL
   - Easier for you, and others to check for bugs at the circuit diagram.
   - A drawing gives a base from which you and other can check if the VHDL is reflecting the architecture envisioned.
   - The tools are not magic! If you cannot sketch your circuit using basic building blocks (e.g., MUXs, counters, state diagrams, etc.), then it is not reasonable to expect the tools to figure it out. Having no sketch is just asking for weird hardware behaviors to occur.

VHDL basics

- VHDL: (V)HSIC (H)ardware (D)escription (L)anguage
  - VHSIC: (V)ery (H)igh (S)peed (I)ntegrated (C)ircuit
- Golden Rules of Hardware Design (VHDL or Verilog)

1. VHDL is a Hardware **Description** Language (HDL)
   - VHDL is NOT a programming language
   - VHDL is conceptually VERY different than C/C++!
2. Draw your Hardware Circuit before writing ANY VHDL
   - Easier for you, and others to check for bugs at the circuit diagram.
   - A drawing gives a base from which you and other can check if the VHDL is reflecting the architecture envisioned.
   - The tools are not magic! If you cannot sketch your circuit using basic building blocks (e.g., MUXs, counters, state diagrams, etc.), then it is not reasonable to expect the tools to figure it out. Having no sketch is just asking for weird hardware behaviors to occur.

Some Key Differences from C

- C is inherently sequential (serial), one statement executed at a time

- VHDL is inherently concurrent (parallel), many statements "execute" at a time

Some Key Differences from C

C example

VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 1
D = 1
Ans = 1

Some Key Differences from C

C example

VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 1
D = 1
Ans = 1

Some Key Differences from C

C example

VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 1
Ans = 1

Some Key Differences from C

C example

VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 1

Some Key Differences from C

C example

Initially: A,B,C,D,Ans =1

```
  C = A + D
  D = A + B
➡ Ans = C + D
```

Current Values:

A = 1

B = 1

C = 2

D = 2

Ans = 4

VHDL example

```
C = A + D
D = A + B
Ans = C + D
```

Some Key Differences from C

C example

VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

Some Key Differences from C

<u>C example</u>

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement
is a circuit

<u>VHDL example</u>

→ C = A + D
→ D = A + B
→ Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

Some Key Differences from C

Initially: A,B,C,D,Ans =1

C example

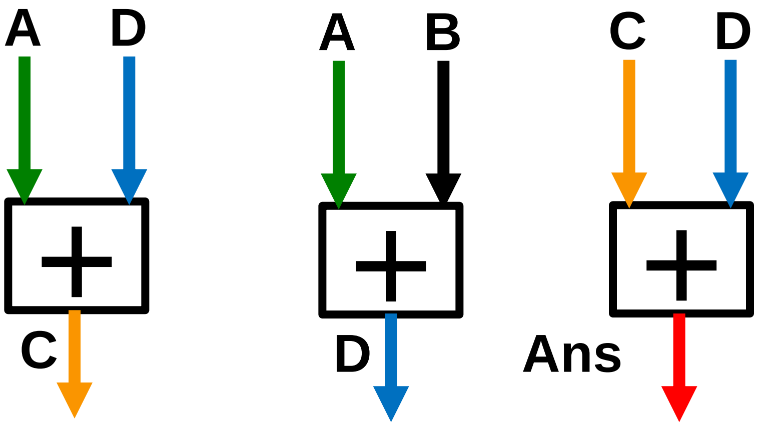
C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

VHDL example

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D



Some Key Differences from C

C example

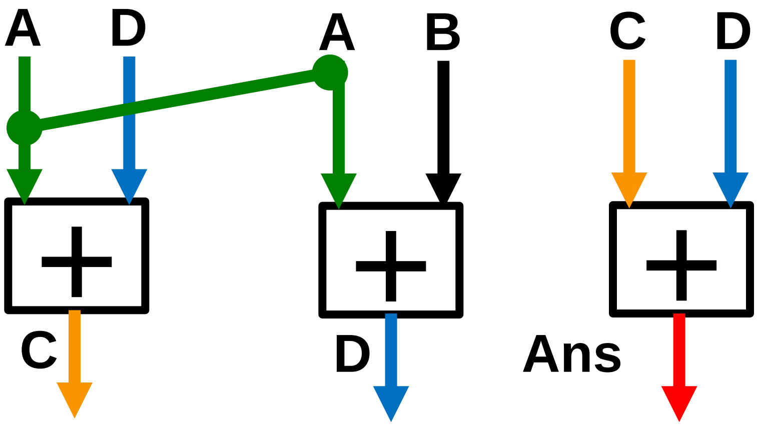
VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D



Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

Some Key Differences from C

C example

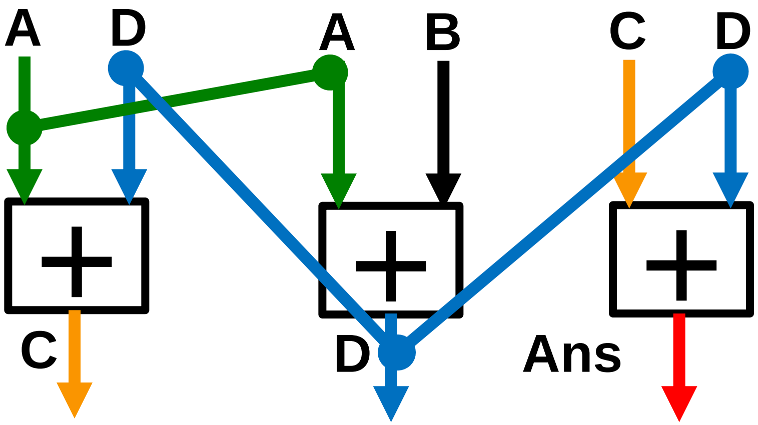
VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D



Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

Some Key Differences from C

C example

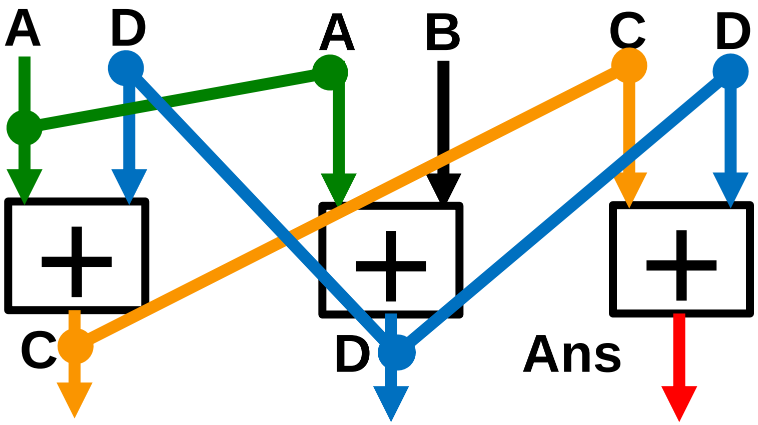
VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D



Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

Some Key Differences from C

C example

VHDL example

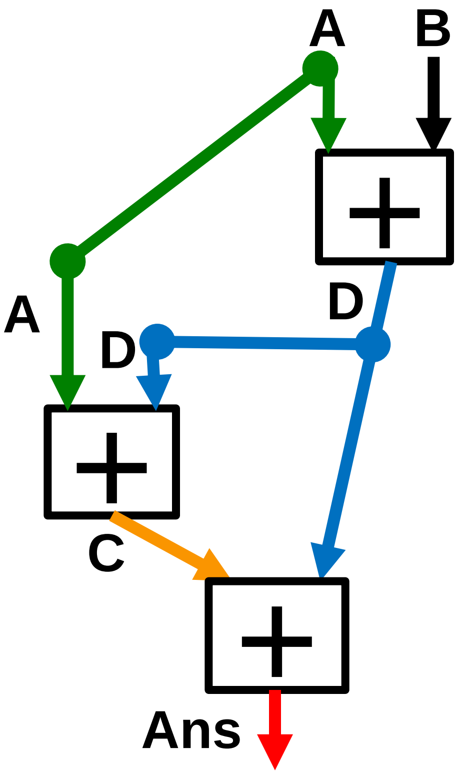
Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4



Some Key Differences from C

C example

VHDL example

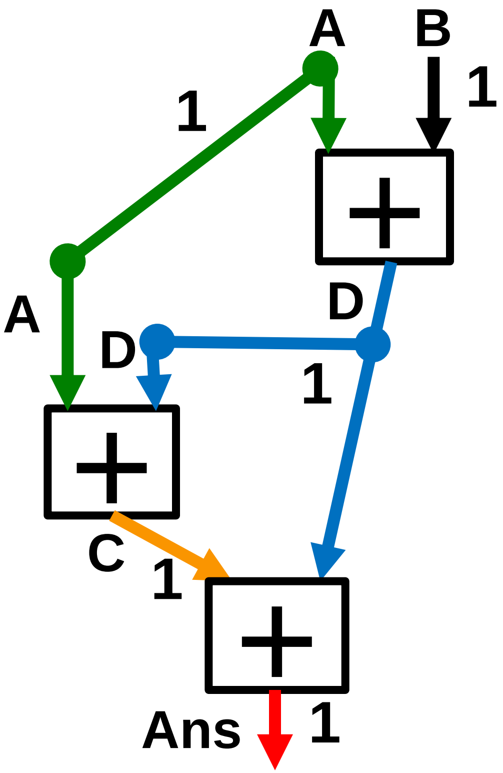
Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4



Some Key Differences from C

C example

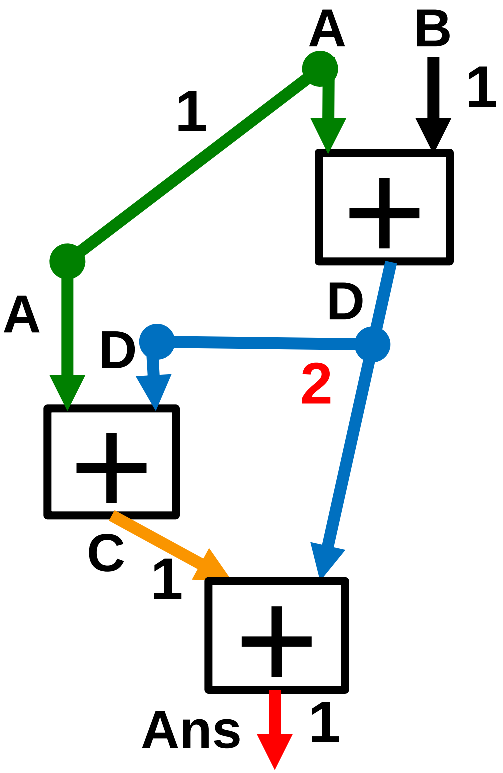
VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D



Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

Some Key Differences from C

C example

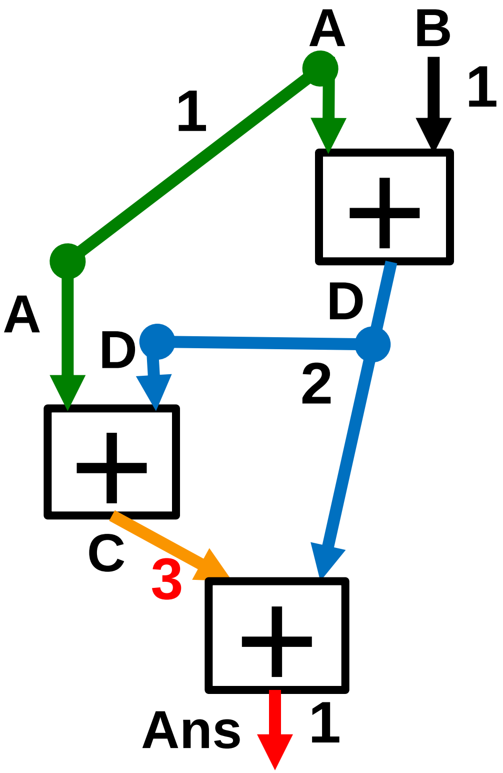
VHDL example

Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D



Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4

Some Key Differences from C

<u>C example</u>

<u>VHDL example</u>

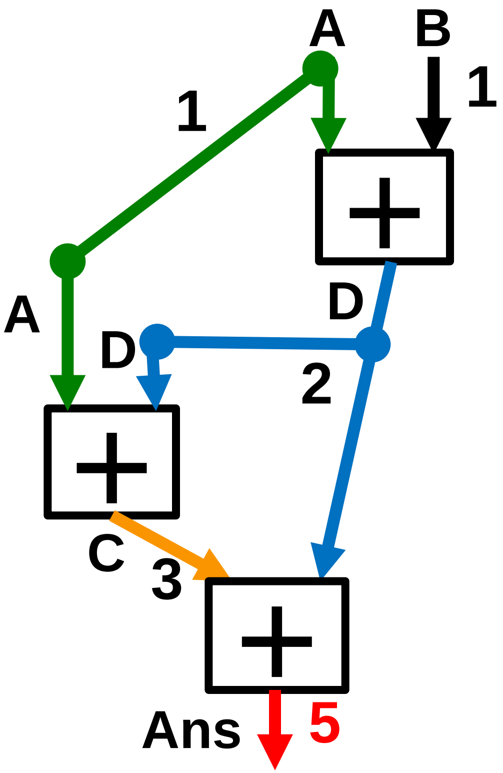
Initially: A,B,C,D,Ans =1

C = A + D
D = A + B
Ans = C + D

Each statement is a circuit

C = A + D
D = A + B
Ans = C + D

Current Values:
A = 1
B = 1
C = 2
D = 2
Ans = 4



Typical Structure of a VHDL File

```
LIBRARY ieee;
```

Include Libraries

```
ENTITY test_circuit IS
    PORT(B,C,Y,Z,Ans);
END test_circuit;
```

Define component name and Input/output ports

Declare internal signals, components

```
ARCHITECTURE structure OF test_circuit IS
  signal A       : std_logic_vector(7 downto 0);
  signal X       : std_logic_vector(7 downto 0);

BEGIN

  A <= B + C;
  X <= Y + Z;
  Ans <= A + X;

END
```

Implement components functionality

Process

- Process provide a level serialization in VHDL (e.g. variables, clocked processes)
- Help separate and add structure to VHDL design

Process Example

```
BEGIN

  My_process_1 : process (A,B,C,X,Y,Z)
  Begin
    A <= B + C;
    X <= Y + Z;
    Ans <= A + X;
  End My_process_1;

  My_process_2 : process (B,X,Y,Ans1)
  Begin
    A <= B + 1;
    X <= B + Y;
    Ans2 <= Ans1 + X;
  End My_process_2;

END;
```

Sensitivity list: specify inputs to the process. Process is updated when a **specified** input changes

Process Example (Multiple Drivers)

```
BEGIN

  My_process_1 : process (A,B,C,X,Y,Z)
  Begin
    A <= B + C;
    X <= Y + Z;
    Ans <= A + X;
  End My_process_1;

  My_process_2 : process (B,X,Y,Ans1)
  Begin
    A <= B + 1;
    X <= B + Y;
    Ans2 <= Ans1 + X;
  End My_process_2;

END;
```

A signal can only be Driven (written) by one process. But can be read by many

Compile or simulator may give a "multiple driver" Error or Warning message

Process Example (Multiple Drivers)

```
BEGIN

  My_process_1 : process (A,B,C,X,Y,Z)
  Begin
    A <= B + C;
    X <= Y + Z;
    Ans <= A + X;
  End My_process_1;

  My_process_2 : process (B,X,Y,Ans1)
  Begin
    A1 <= B + 1;
    X1 <= B + Y;
    Ans2 <= Ans1 + X;
  End My_process_2;

END;
```

Maybe A,X were suppose to be A1,X1. Cut and paste error. Or may need to rethink Hardware structure to remove multiple driver issue.

Process Example (if-statement)

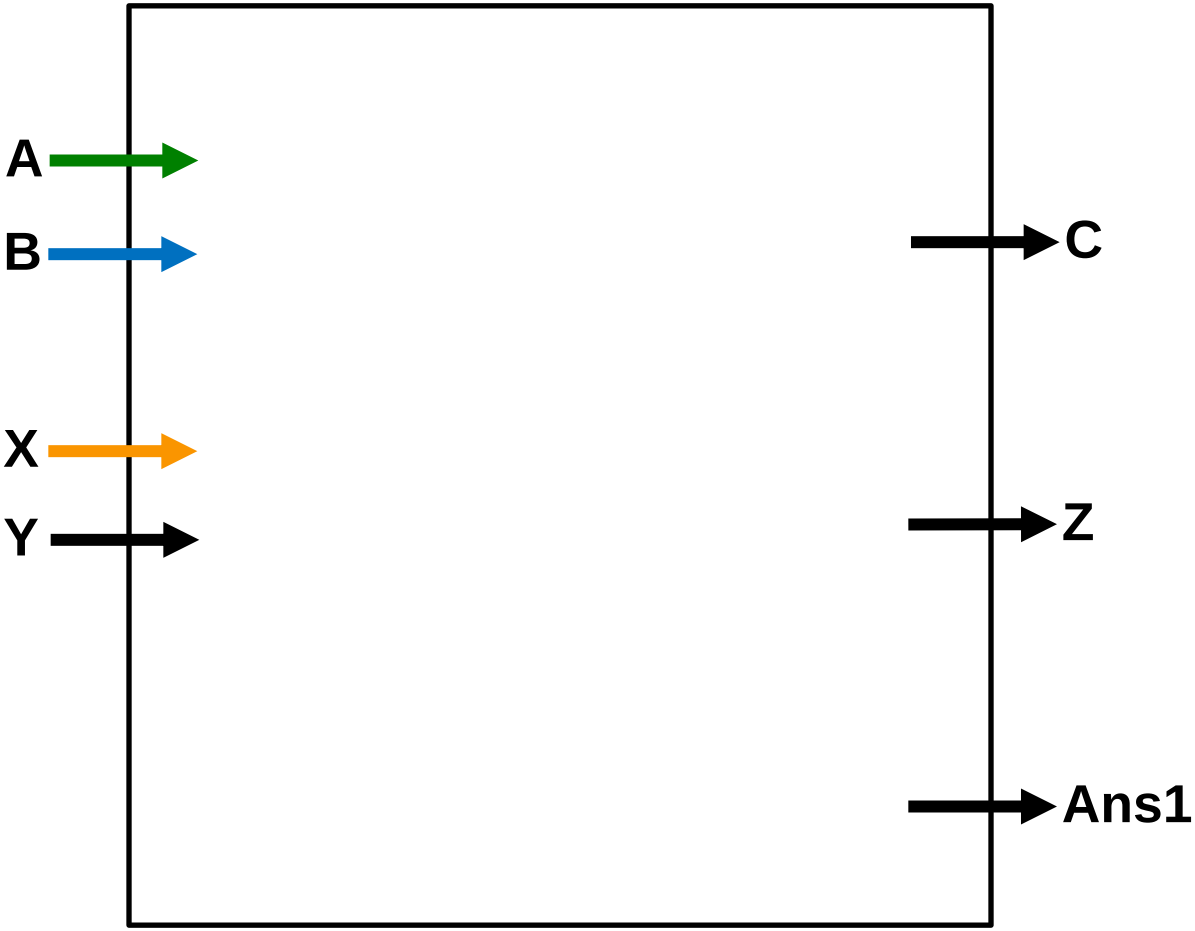
```
BEGIN
  My_process_1 : process (A,B,C,X,Y,Z)
  Begin

    if (B = 0) then
      C <= A + B;
      Z <= X + Y;
      Ans1 <= A + X;
    else
      C <= 1;
      Z <= 0;
      Ans1 <= 1;
    end if;

  End My_process_1;
END;
```

Draw circuit

Process Example (if-statement)

```
BEGIN
  My_process_1 : process (A,B,C,X,Y,Z)
  Begin

    if (B = 0) then
      C <= A + B;
      Z <= X + Y;
      Ans1 <= A + X;
    else
      C <= 1;
      Z <= 0;
      Ans1 <= 1;
    end if;

  End My_process_1;
END;
```

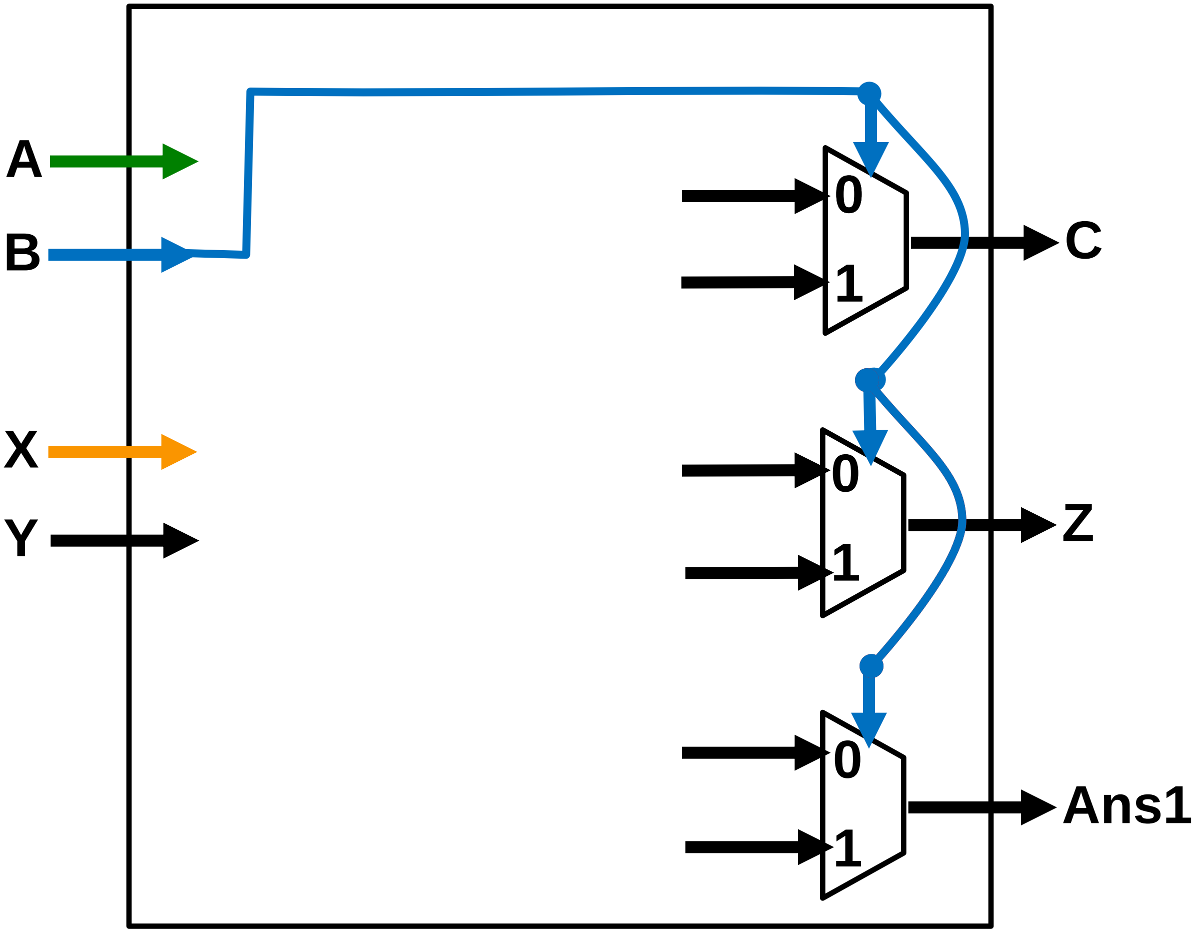
Circuit for My_process_1



Process Example (if-statement)

```
BEGIN
  My_process_1 : process (A,B,C,X,Y,Z)
  Begin

    if (B = 0) then
      C <= A + B;
      Z <= X + Y;
      Ans1 <= A + X;
    else
      C <= 1;
      Z <= 0;
      Ans1 <= 1;
    end if;

  End My_process_1;
END;
```

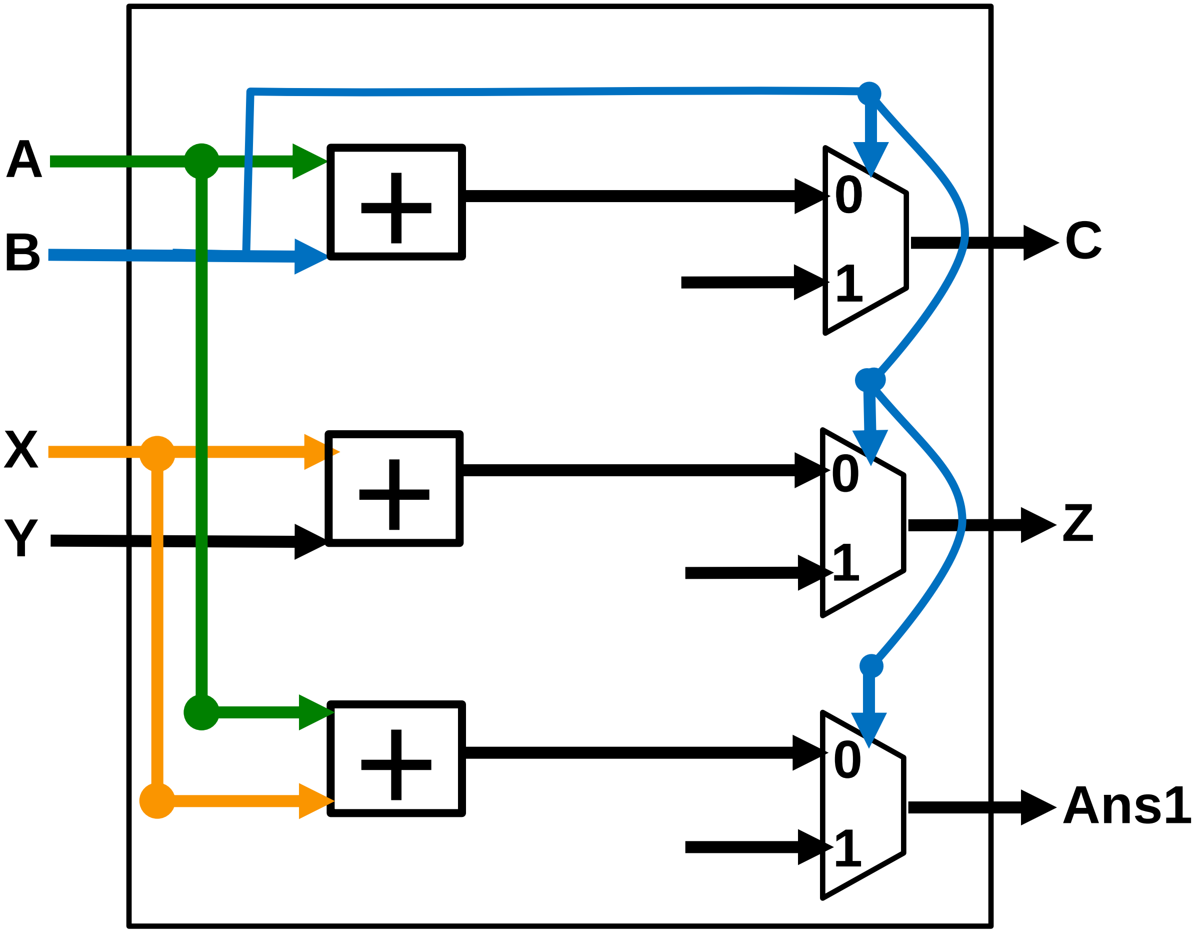
Circuit for My_process_1



Process Example (if-statement)

```
BEGIN
  My_process_1 : process (A,B,C,X,Y,Z)
  Begin

    if (B = 0) then
      C <= A + B;
      Z <= X + Y;
      Ans1 <= A + X;
    else
      C <= 1;
      Z <= 0;
      Ans1 <= 1;
    end if;

  End My_process_1;
END;
```

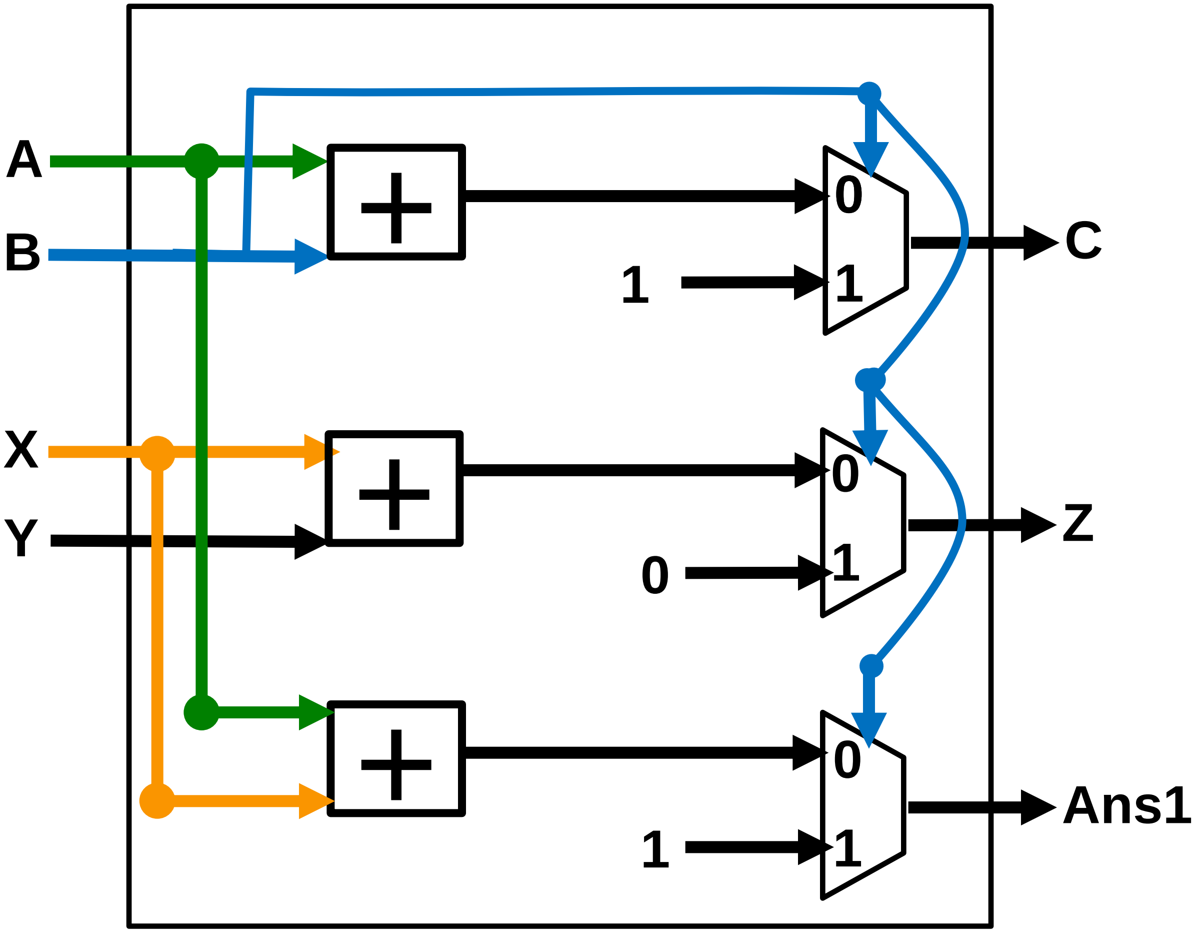
Circuit for My_process_1



Process Example (if-statement)

```
BEGIN
  My_process_1 : process (A,B,C,X,Y,Z)
  Begin

    if (B = 0) then
      C <= A + B;
      Z <= X + Y;
      Ans1 <= A + X;
    else
      C <= 1;
      Z <= 0;
      Ans1 <= 1;
    end if;

  End My_process_1;
END;
```

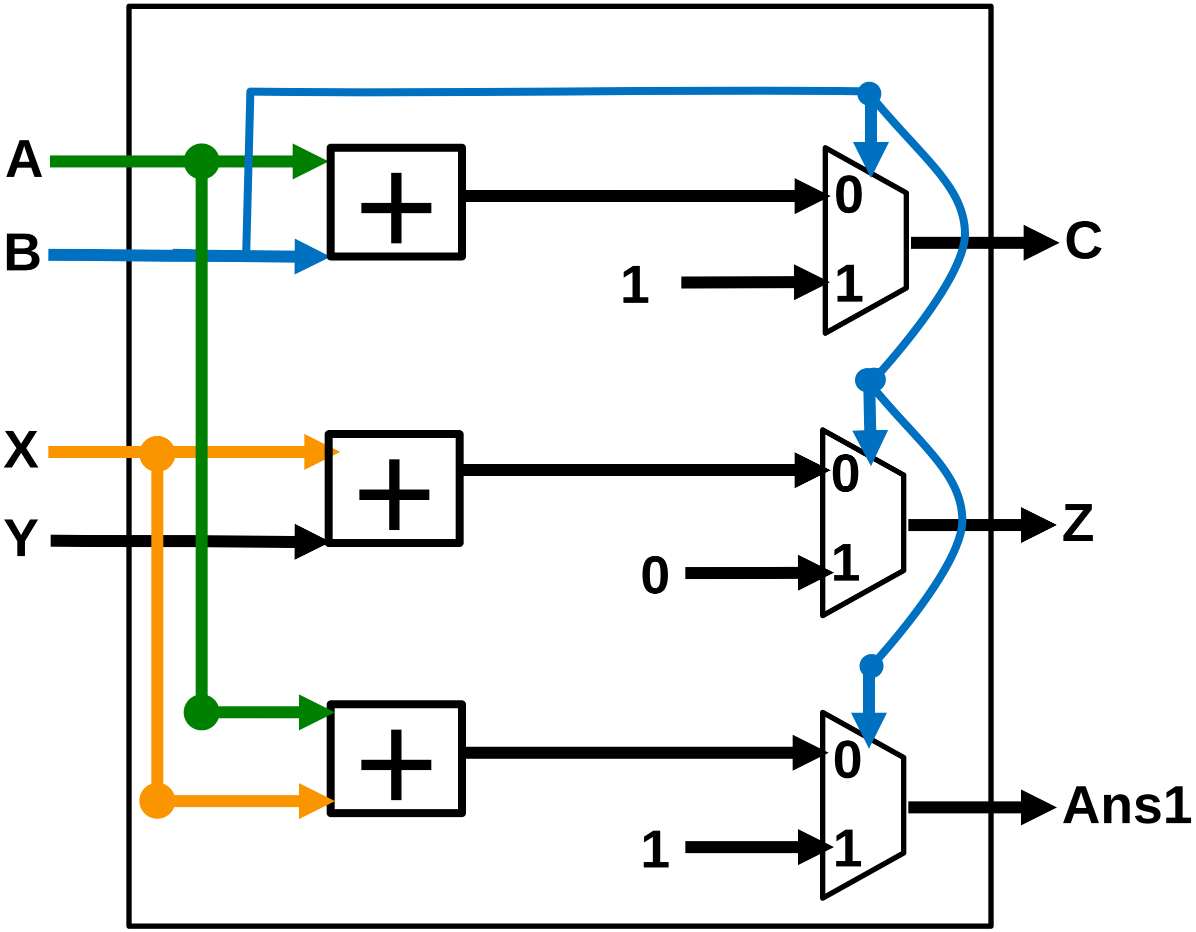
Circuit for My_process_1



Process Example (if-statement)

```
BEGIN
 My_process_1 : process (A,B,~~C~~,X,Y,~~Z~~)
 Begin

   if (B = 0) then
     C <= A + B;
     Z <= X + Y;
     Ans1 <= A + X;
   else
     C <= 1;
     Z <= 0;
     Ans1 <= 1;
   end if;

 End My_process_1;
END;
```

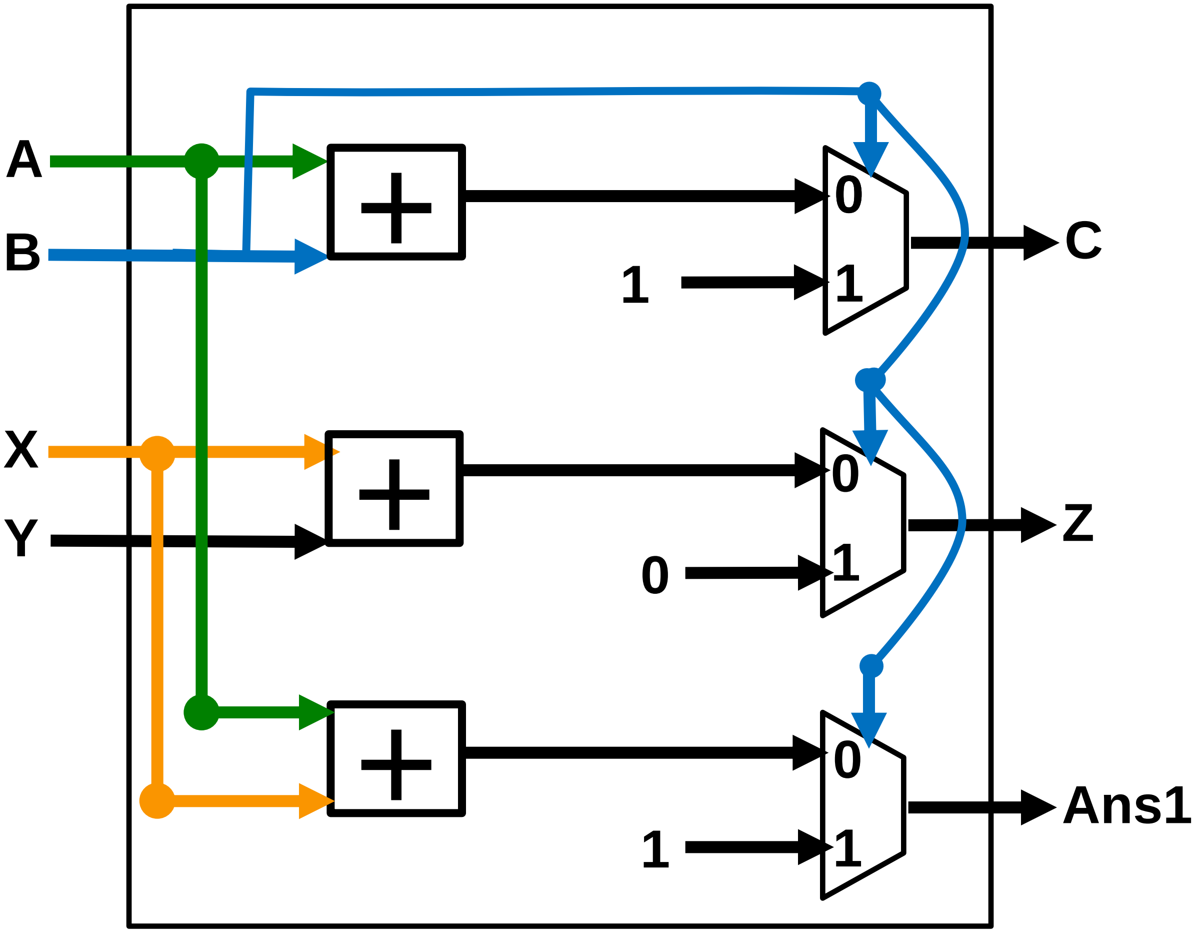
Circuit for My_process_1



Process Example (if-statement)

```
BEGIN
  My_process_1 : process (A,B,X,Y)
  Begin

    if (B = 0) then
      C <= A + B;
      Z <= X + Y;
      Ans1 <= A + X;
    else
      C <= 1;
      Z <= 0;
      Ans1 <= 1;
    end if;

  End My_process_1;
END;
```

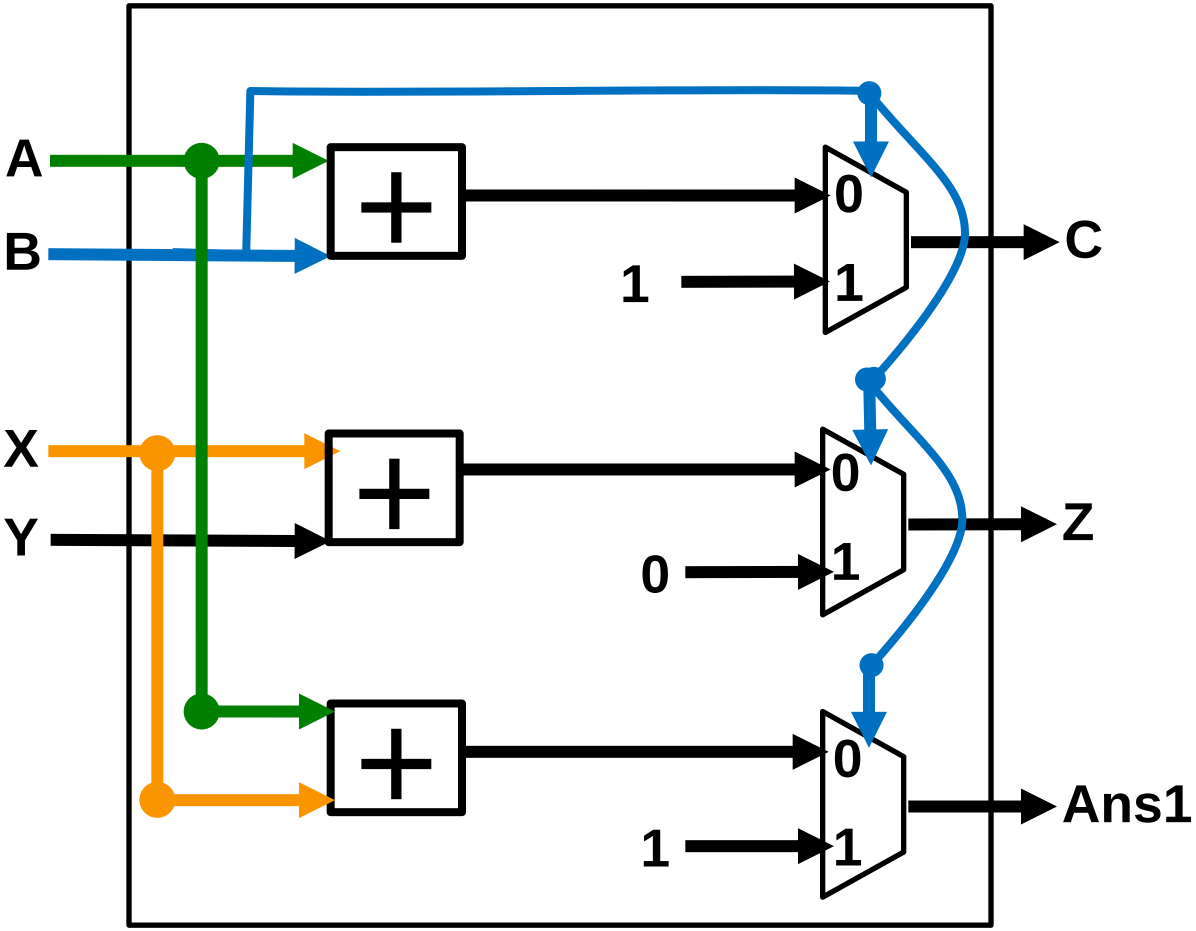
Circuit for My_process_1



Process Example (if-statement)

```
BEGIN
  My_process_1 : process (A,B,X,Y)
  Begin
      if (B = 0) then
        C <= A + B;
      else
        C <= 1;
      end if;

      if (B = 0) then
        Z <= X + Y;
      else
        Z <= 0;
      end if;

      if (B = 0) then
        Ans1 <= A + X;
      else
        Ans1 <= 1;
      end if;

  End My_process_1;
END;
```

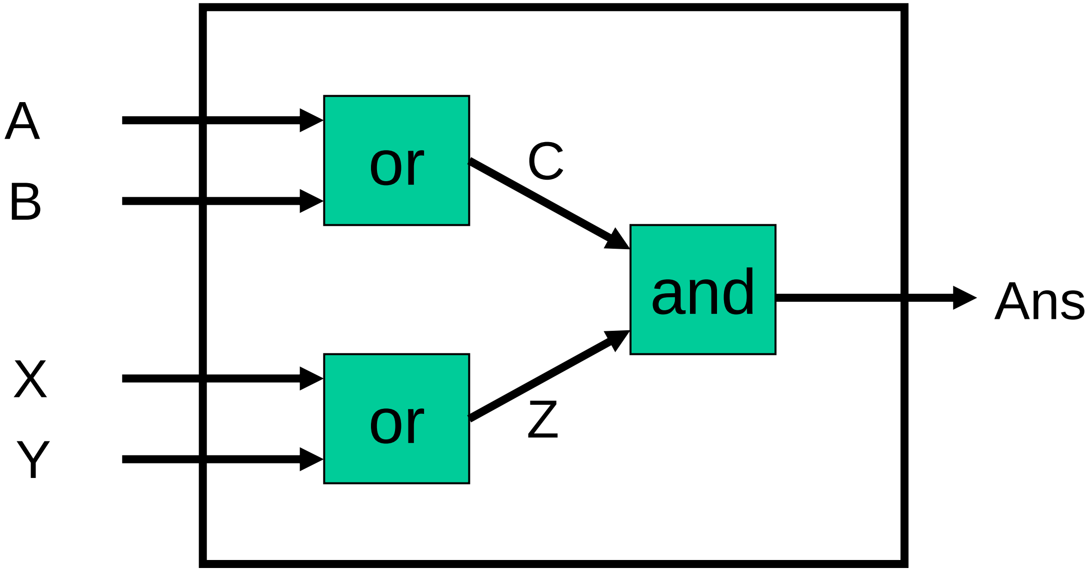
Circuit for My_process_1



Clocked Process Example

```
BEGIN

  My_process_1 : process (A, B, C, X, Y, Z)
  Begin

      C <= A or B;
      Z <= X or Y;
      Ans <= C and Z;

  End My_process_1;

END;
```

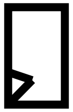
circuit **not** clocked



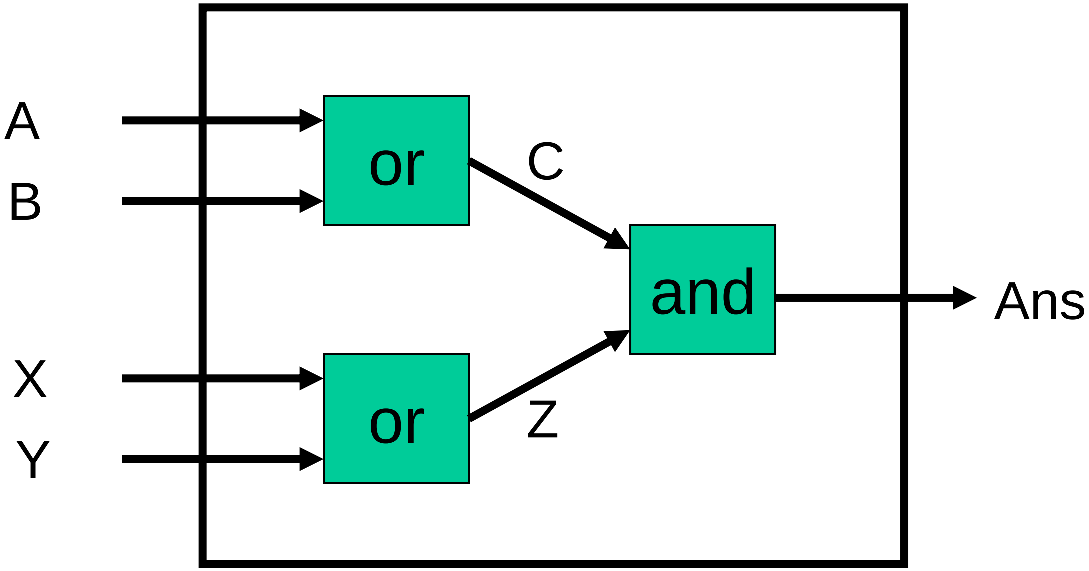
Clocked Process Example

```
BEGIN

  My_process_1 : process (A, B, C, X, Y, Z)
  Begin

      C <= A or B;
      Z <= X or Y;
      Ans <= C and Z;

  End My_process_1;

END;
```

D Flip-Flop (DFF)
Registers



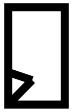
circuit **not** clocked



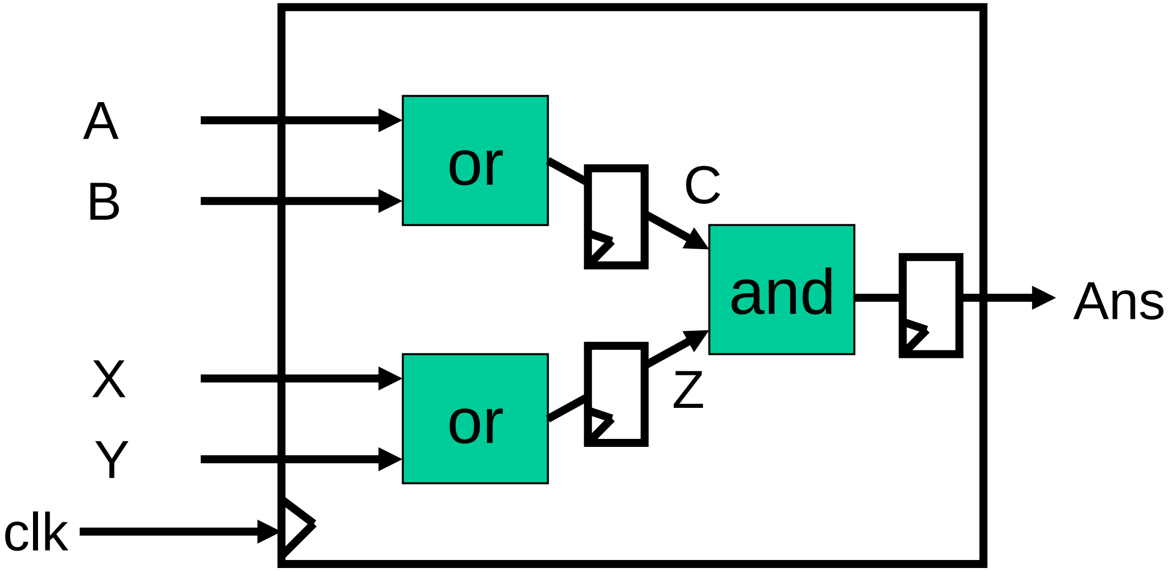
Clocked Process Example

```
BEGIN

  My_process_1 : process (A, B, C, X, Y, Z)
  Begin

      C <= A or B;
      Z <= X or Y;
      Ans <= C and Z;

  End My_process_1;

END;
```

D Flip-Flop (DFF)
Registers



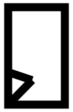
circuit with clock



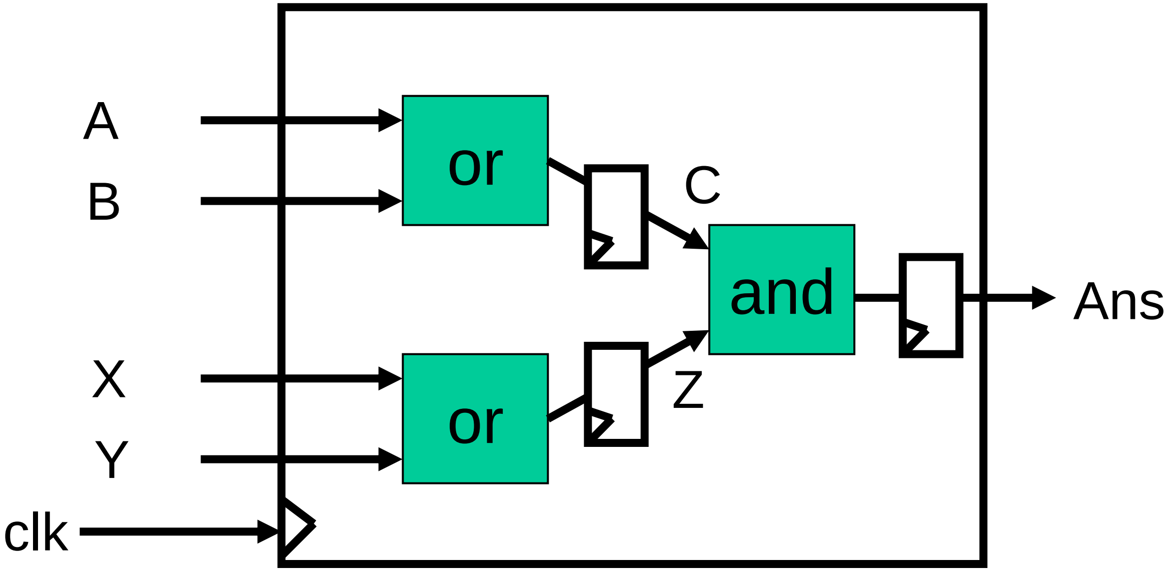
Clocked Process Example

```
BEGIN

  My_process_1 : process (clk)
  Begin
    IF (clk'event and clk = '1') THEN
      C <= A or B;
      Z <= X or Y;
      Ans <= C and Z;
    END IF;
  End My_process_1;

END;
```

D Flip-Flop (DFF)
Registers

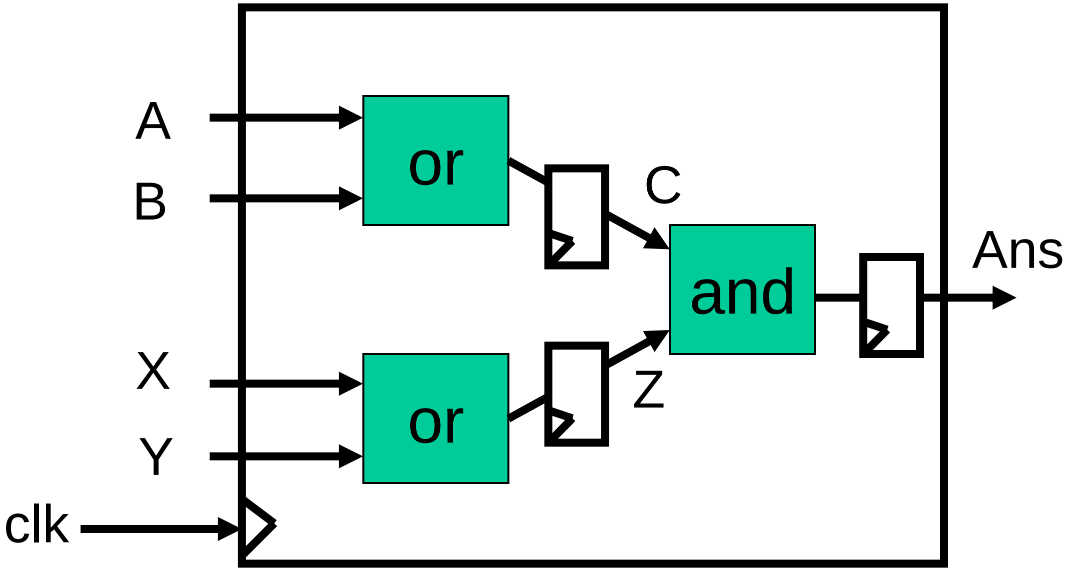


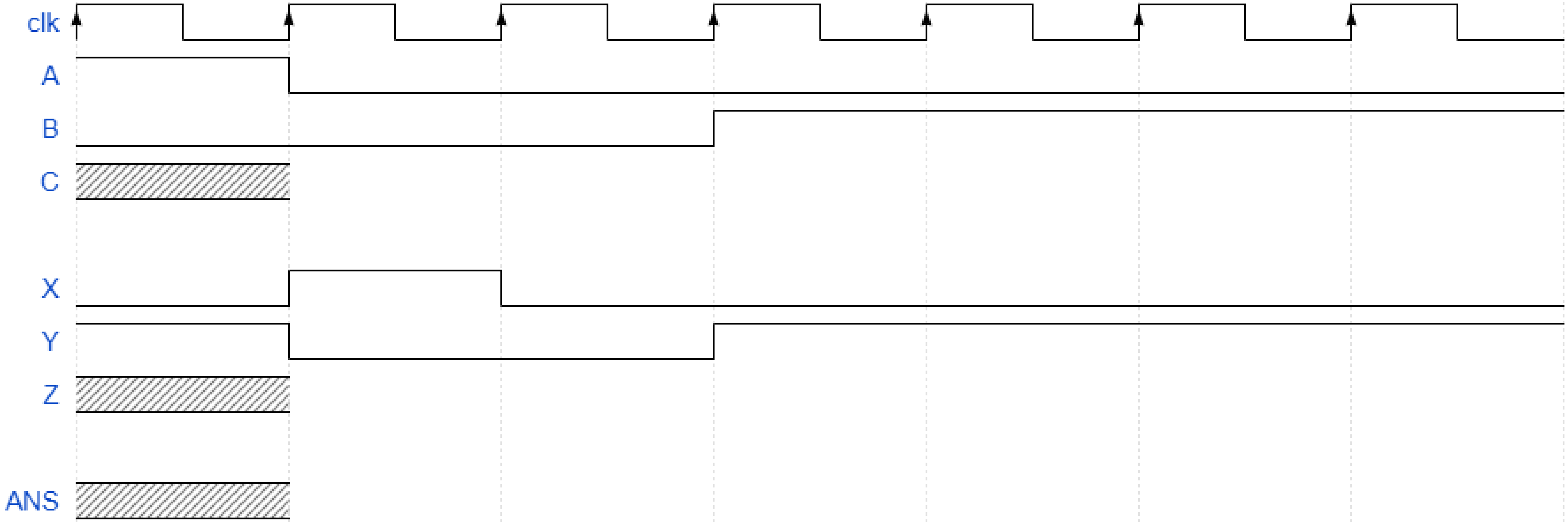
circuit with clock



Clocked Process Example

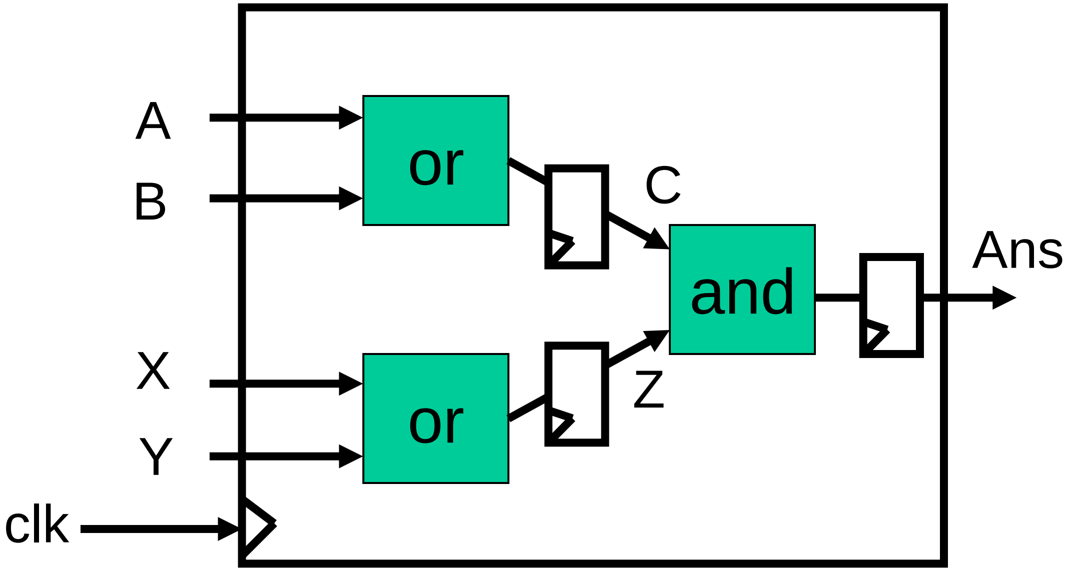
```
BEGIN

  My_process_1 : process (clk)
  Begin
    IF (clk'event and clk = '1') THEN
      C <= A or B;
      Z <= X or Y;
      Ans <= C and Z;
    END IF;
  End My_process_1;

END;
```

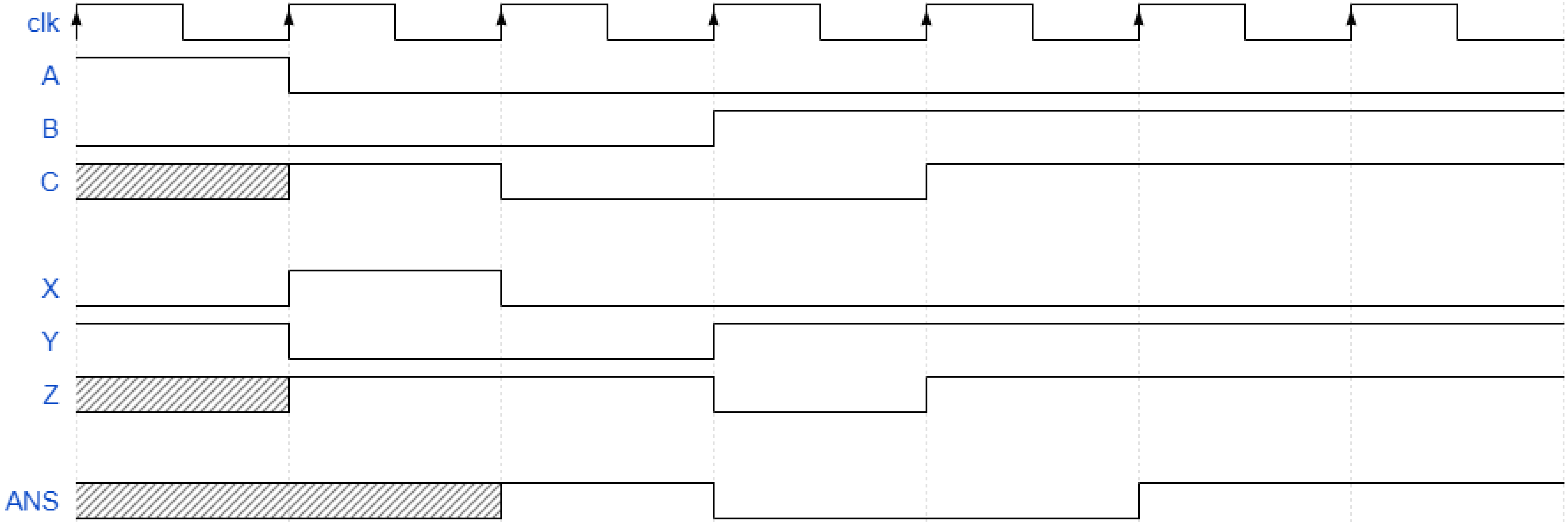




Clocked Process Example

```
BEGIN

 My_process_1 : process (clk)
 Begin
   IF (clk'event and clk = '1') THEN
     C <= A or B;
     Z <= X or Y;
     Ans <= C and Z;
   END IF;
 End My_process_1;

END;
```





VHDL Constructs

- Entity
- Process
- Signal, Variable, Constants, Integers
- Array, Record

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

Signals and Variables

- Signals
  - Updated at the end of a process
  - Have file scope
- Variables
  - Updated instantaneously
  - Have process scope

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

std_logic, std_logic_vector

- Very common data types

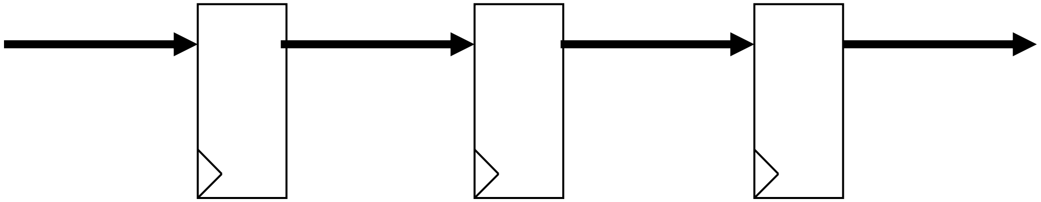
- std_logic
  - Single bit value
  - Values: U, X, 0, 1, Z, W, H, L, -
  - Example: **signal** A : std_logic;
    - A <= ‘1’;
- std_logic_vector: is an array of std_logic
  - Example: **signal** A : std_logic_vector (4 **downto** 0);
    - A <= “0Z001”

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

std_logic values

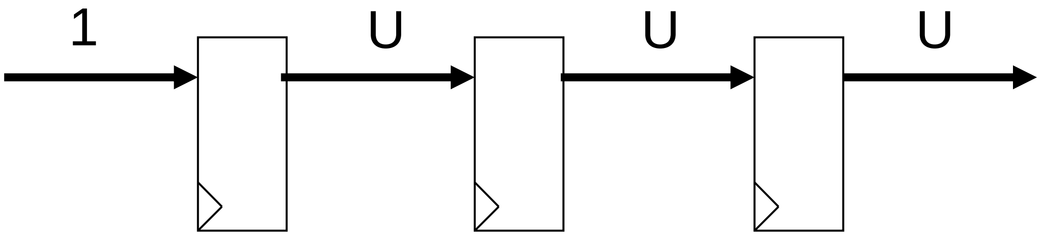
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 0

std_logic values

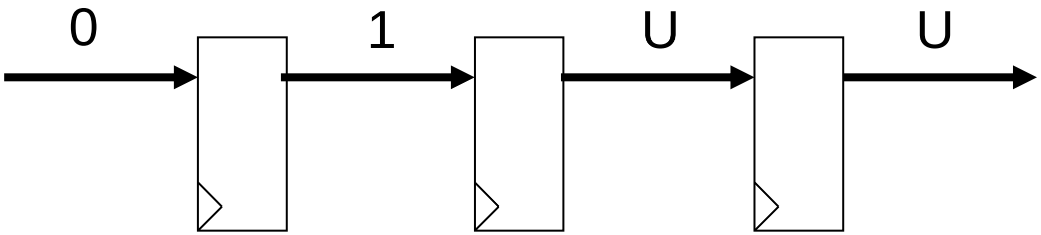
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 0

std_logic values

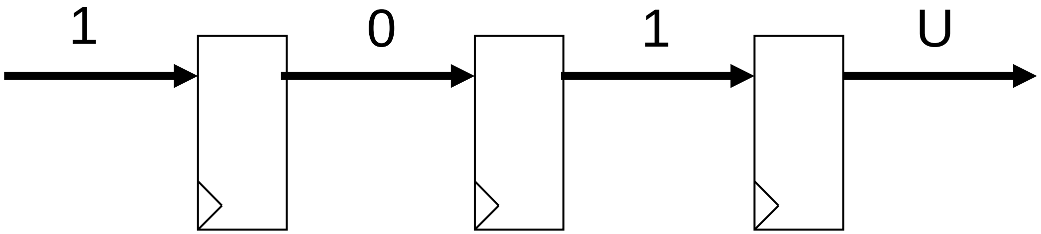
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 1

std_logic values

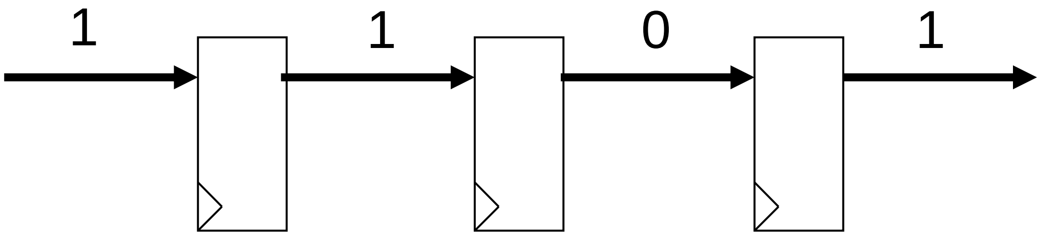
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 2

std_logic values

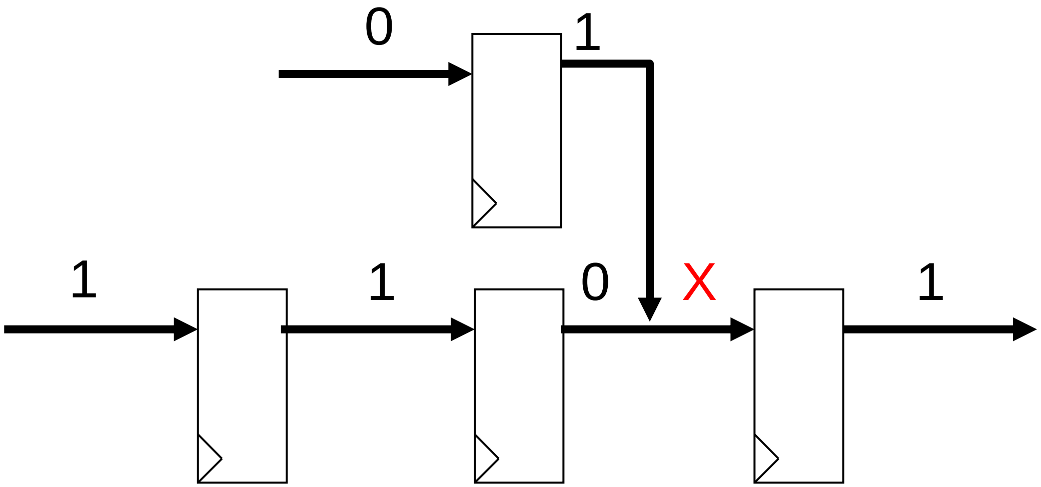
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 3

std_logic values

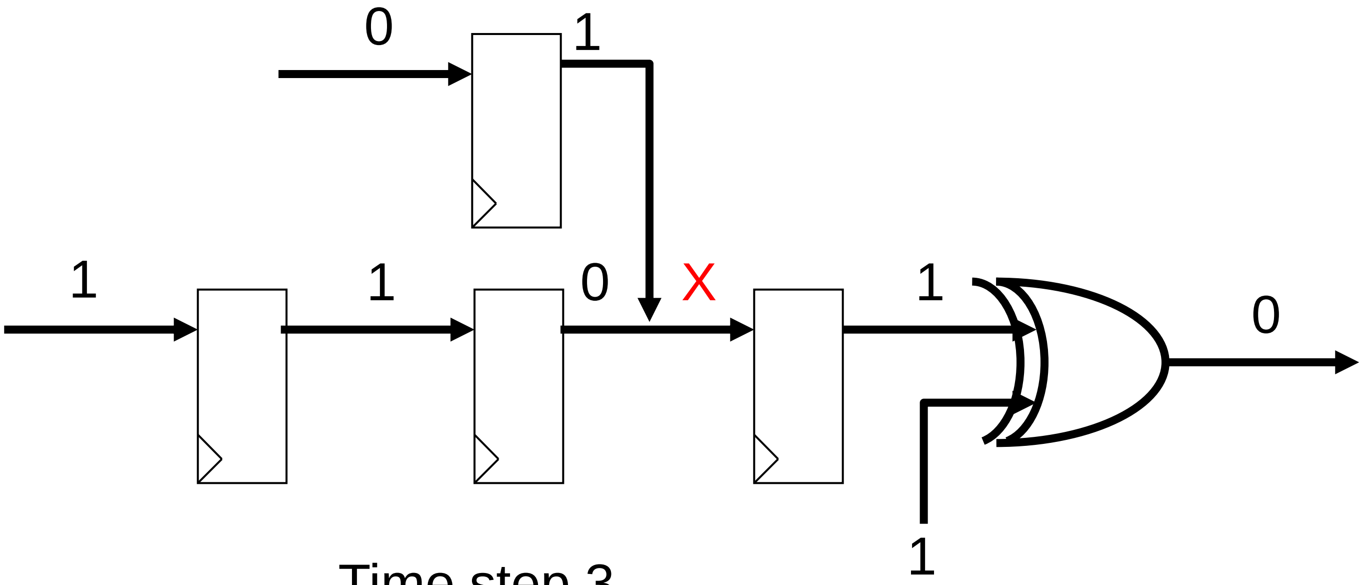
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 3

std_logic values

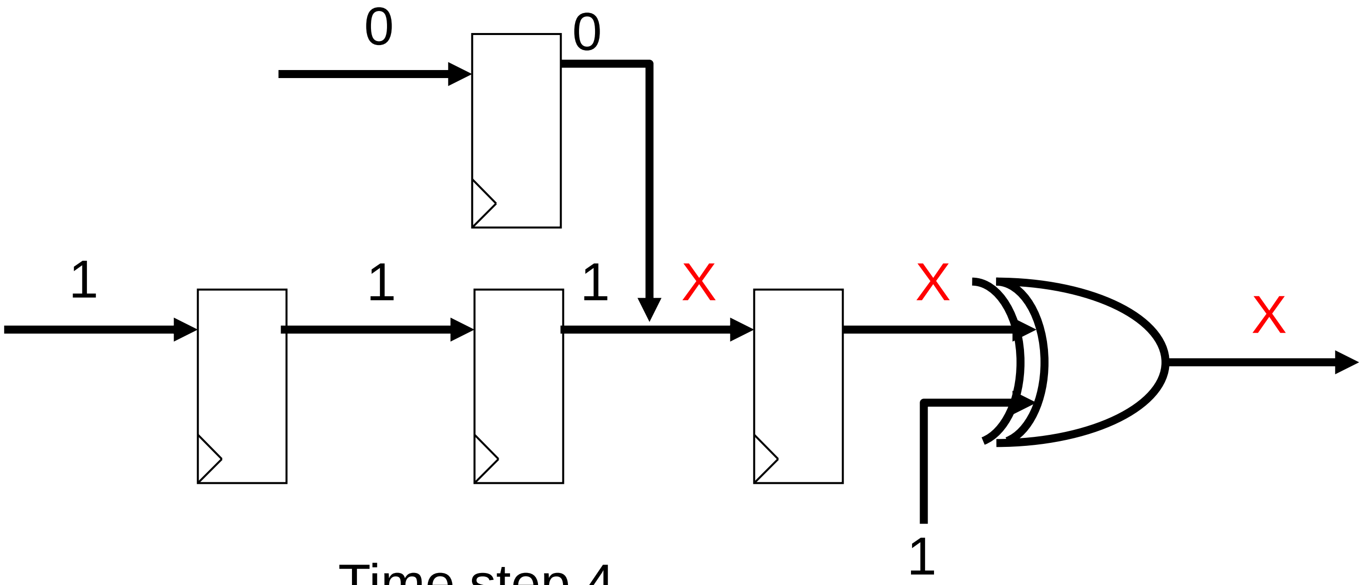
- std_logic values
  - – U : Uninitialized (signal has not been assigned a value yet)
  - – X : Unknow (2 drivers one ‘0’ one ‘1’)
  - – H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - – L : weak ‘0’



Time step 3

std_logic values

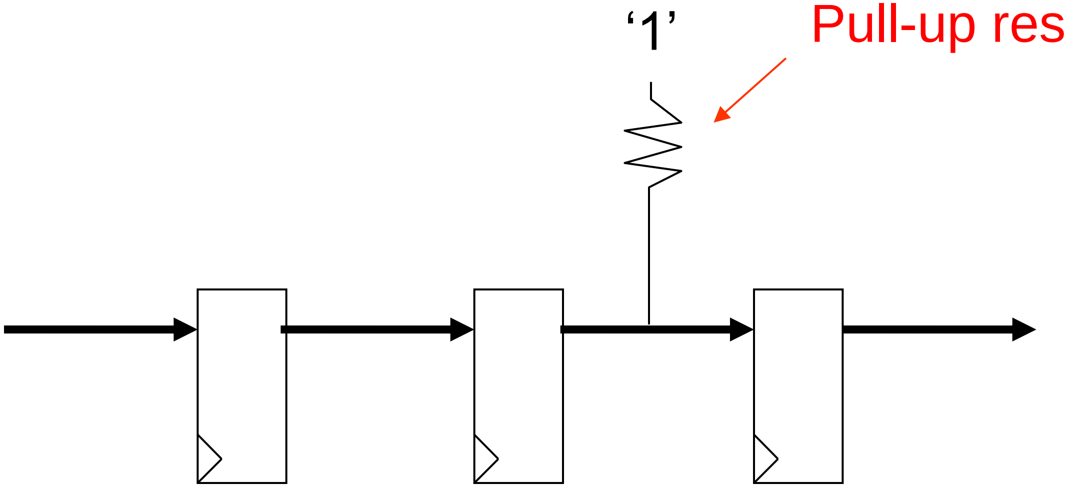
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 4

std_logic values

- std_logic values
  - – U : Uninitialized (signal has not been assigned a value yet)
  - – X : Unknow (2 drivers one ‘0’ one ‘1’)
  - – H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - – L : weak ‘0’

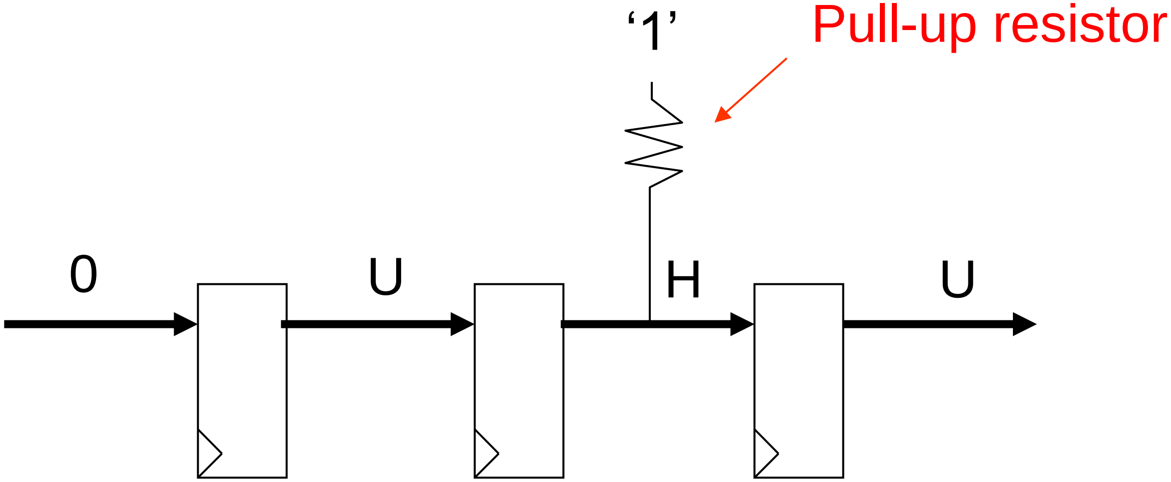


Pull-up resistor

Time step 0

std_logic values

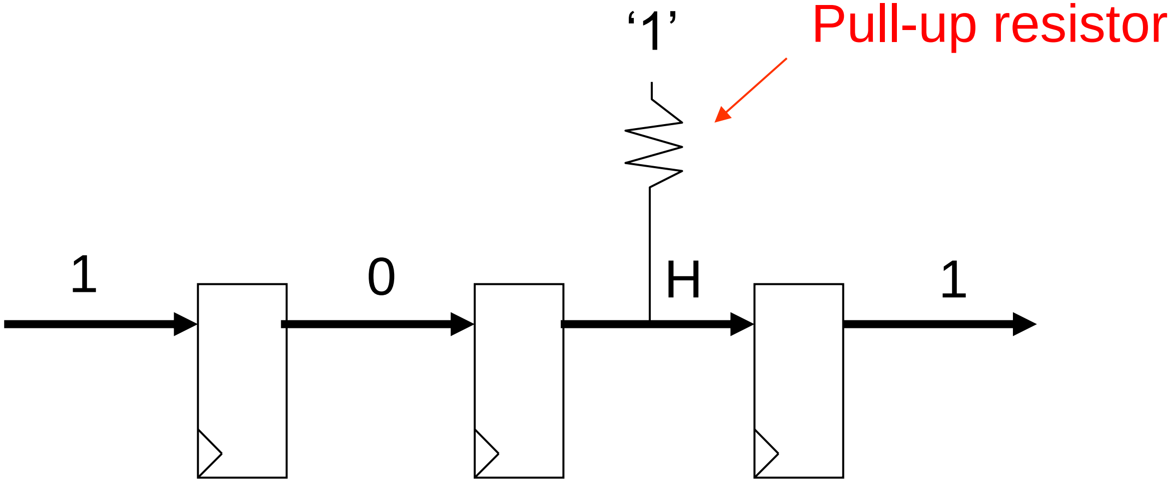
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 0

std_logic values

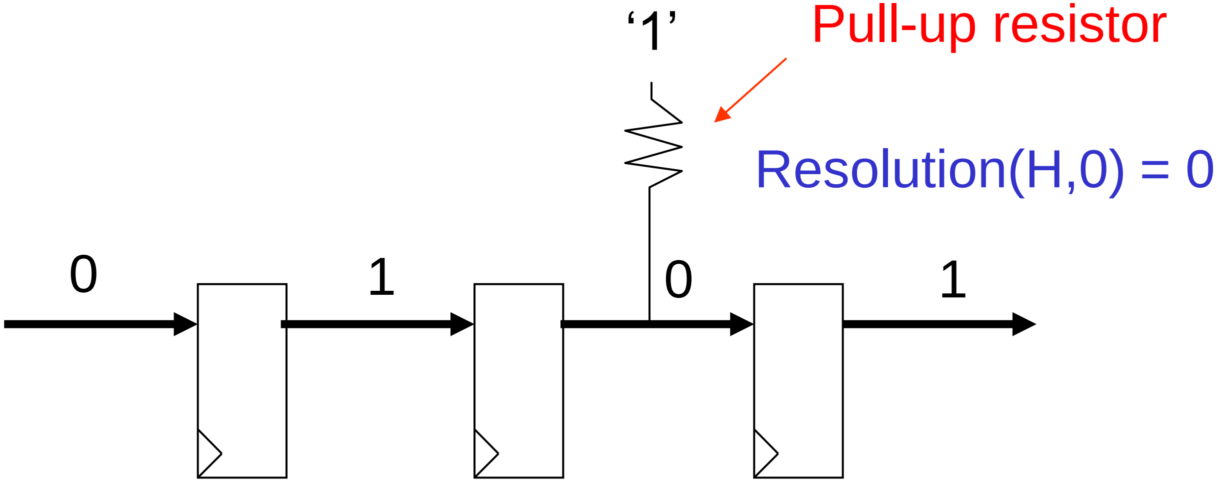
- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 1

std_logic values

- std_logic values
  - U : Uninitialized (signal has not been assigned a value yet)
  - X : Unknow (2 drivers one ‘0’ one ‘1’)
  - H : weak ‘1’ (example: model pull-up resister)
    - I have never used this value
  - L : weak ‘0’



Time step 2

https://www.vhdl-online.de/courses/system_design/vhdl_language_and_syntax/extended_data_types/standard_logic_type

Pre-defined VHDL attributes

- mysignal’event (mysignal changed value)
- mysignal’high (highest value of mysignal’s type)
- mysignal’low
- Many other attributes
  - http://www.cs.umbc.edu/help/VHDL/summary.html

Singal vs Varible scope

- Signal: global to file
- Variable: local to process

```
My_process_1 : process (B,C,Y)
Begin
  A <= B + C;
  Z <= Y + C;
End My_process_1;

My_process_2 : process (B,Y,Z)
Begin
  X <= Z + 1;
  Ans <= B + Y;
End My_process_2;
```

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

Singal vs Varible scope

- Signal: global to file
- Variable: local to process

```
My_process_1 : process (B,C,Y)
Begin
  A <= B + C;
  varZ <= Y + C;
End My_process_1;

My_process_2 : process (B,Y)
Begin
  X <= varZ + 1;
  Ans <= B + Y;
End My_process_2;
```

Each varZ are local
to their process.
Completely independent

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

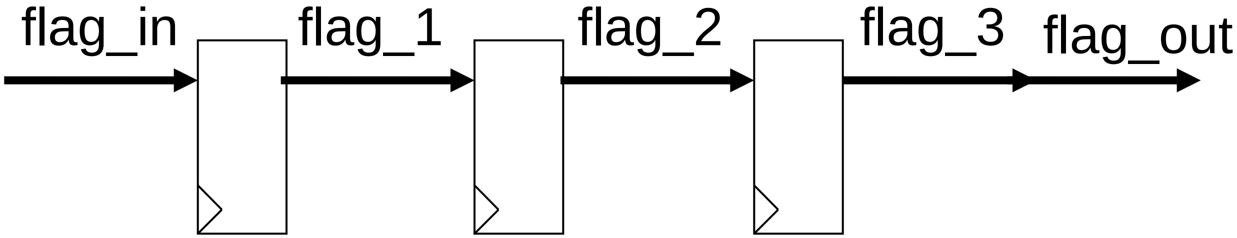
Arrays and Records

- Arrays: Group signals of the same type together
- Records: Group signal of different types together

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

Array Example (Delay Shift Register)

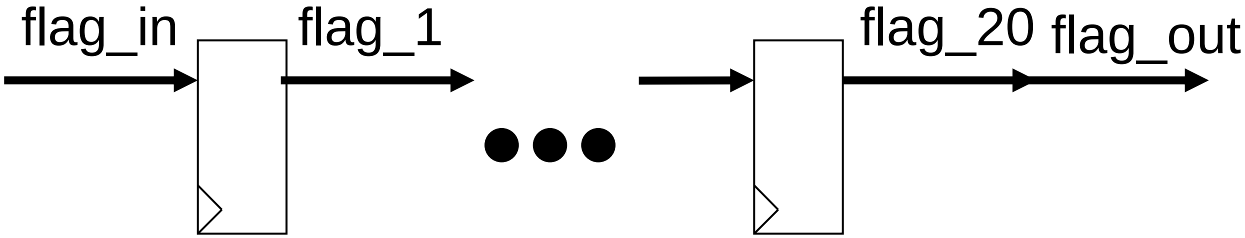


```
BEGIN
  My_process_1 : process (clk)
  Begin
    IF (clk’event and clk = ‘1’) THEN
      flag_1 <= flag_in;
      flag_2 <= flag_1;
      flag_3 <= flag_2;
    END IF;
  End My_process_1;
  flag_out <= flag_3
END;
```

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

Array Example (Delay Shift Register)

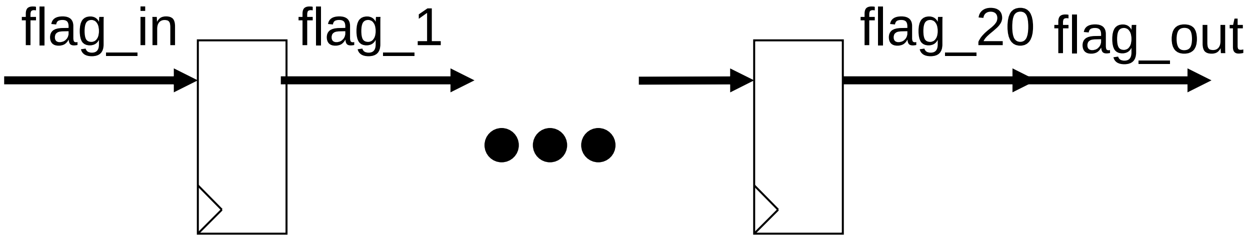


```
BEGIN
  My_process_1 : process (clk)
  Begin
    IF (clk'event and clk = '1') THEN
      flag_1 <= flag_in;
      flag_2 <= flag_1;
             ●●●
      flag_20 <= flag_19;
    END IF;
  End My_process_1;
  flag_out <= flag_20
END;
```

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

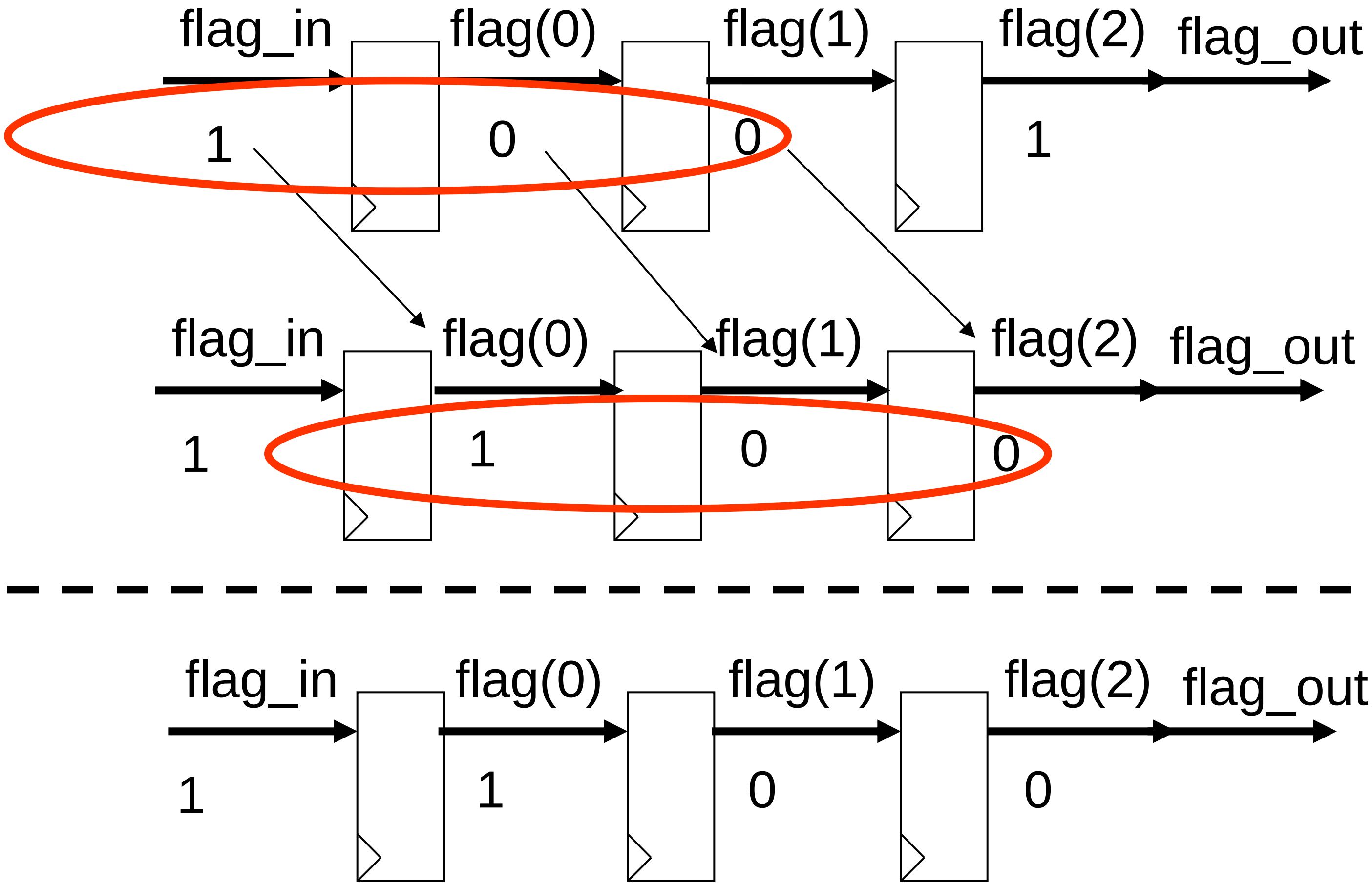
Array Example (Delay Shift Register)



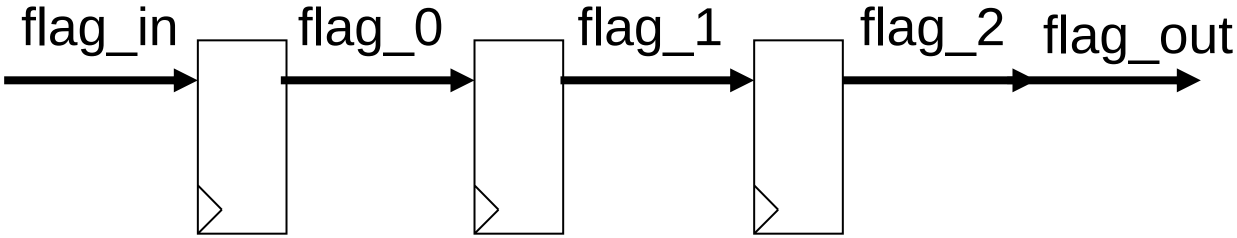
```
type flag_reg_array is array (DELAY-1 downto 0) of std_logic;
signal flag_reg : flag_reg_array;
BEGIN
  My_process_1 : process (clk)
  Begin
    IF (clk’event and clk = ‘1’) THEN
      flag_reg(flag_reg'high downto 0) <=
        flag_reg(flag_reg'high-1 downto 0) & flag_in;
    END IF;
  End My_process_1;
  flag_out <= flag_reg(flag_reg'high);
END;
```

Array Example (Delay Shift Register)

```
flag_reg(flag_reg'high downto 0)<= flag_reg(flag_reg'high-1 downto 0) & flag_in;
```



Array Example (Delay Shift Register)



```
BEGIN
  My_process_1 : process (clk)
  Begin
    IF (clk'event and clk = '1') THEN
      flag_0 <= flag_in;
      flag_1 <= flag_0;
      flag_2 <= flag_1;
    END IF;
  End My_process_1;
  flag_out <= flag_2
END;
```

VHDL on-line tutorials:

https://www.vhdl-online.de/courses/system_design/start

Finite State Machine (FSM) Design

- Model of computation
- High level application example (Networking)
- Two major types
  - Moore
  - Mealy
- Detailed view of application example

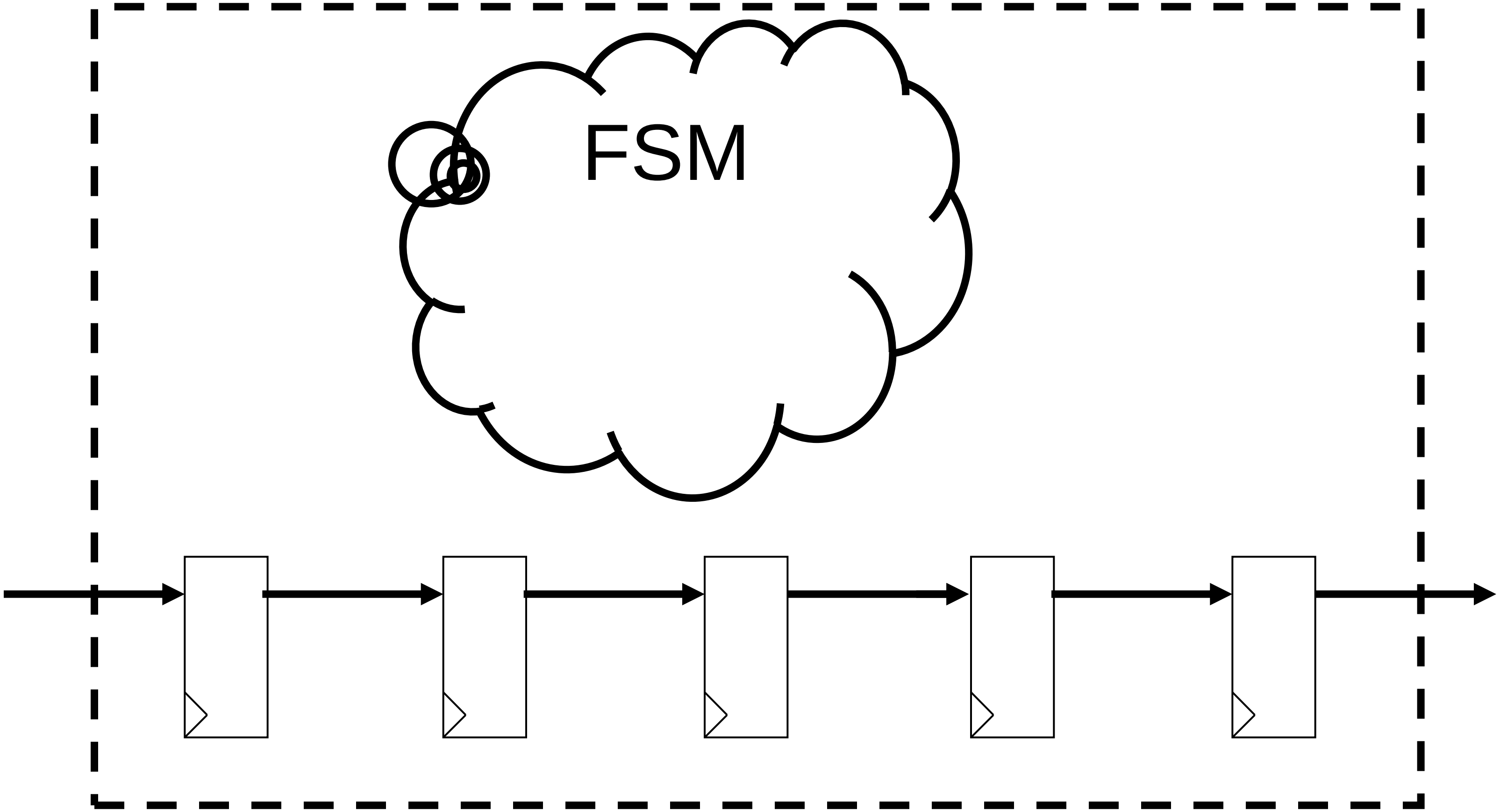
Finite State Machines

- What types of applications are they well suited
  - Streaming pattern recognition (e.g. Network Intrusion detection)
  - Sequential event based control logic (e.g. Traffic Light)

- Allows hardware designer to reason about things in small pieces

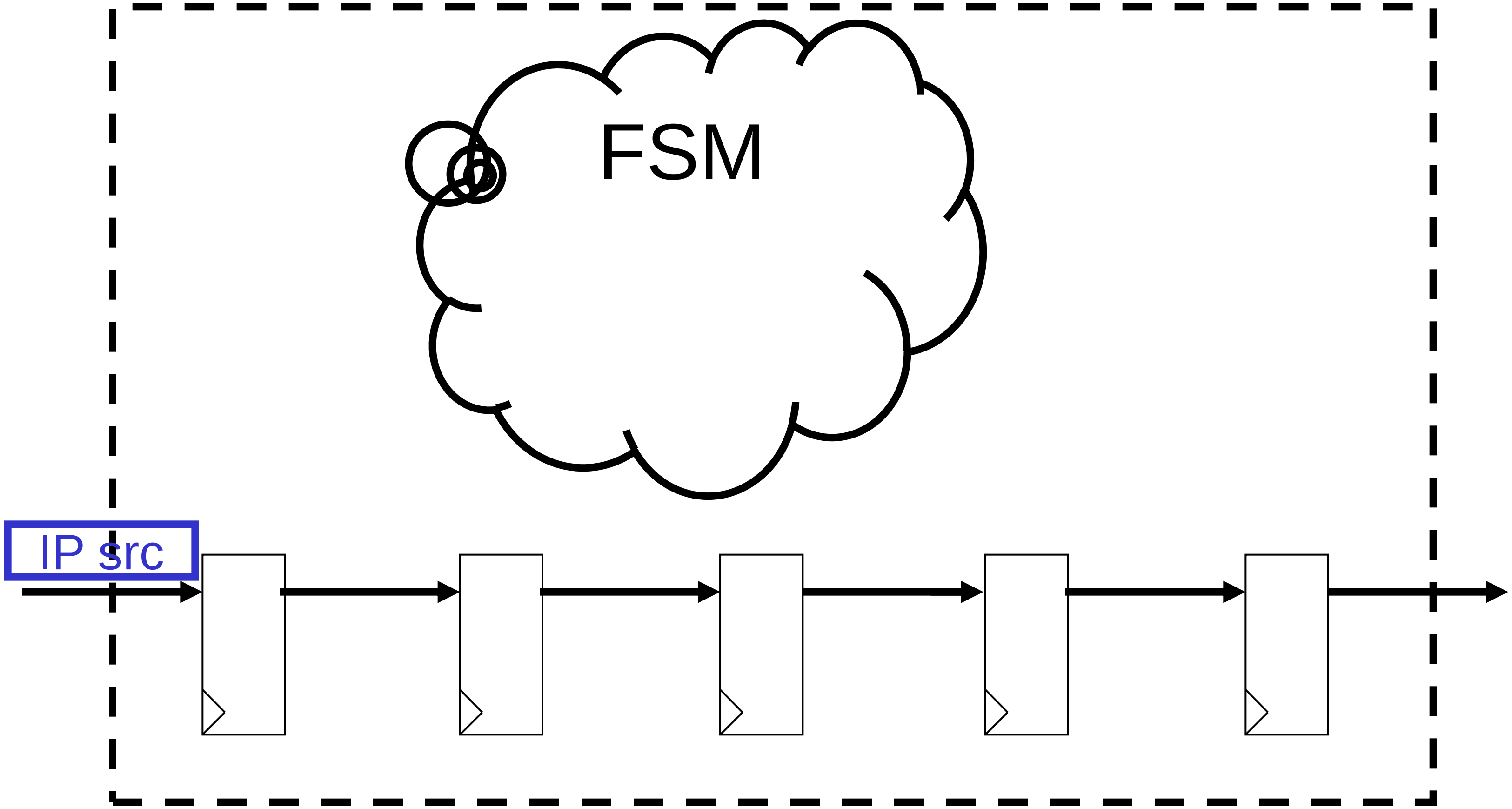
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



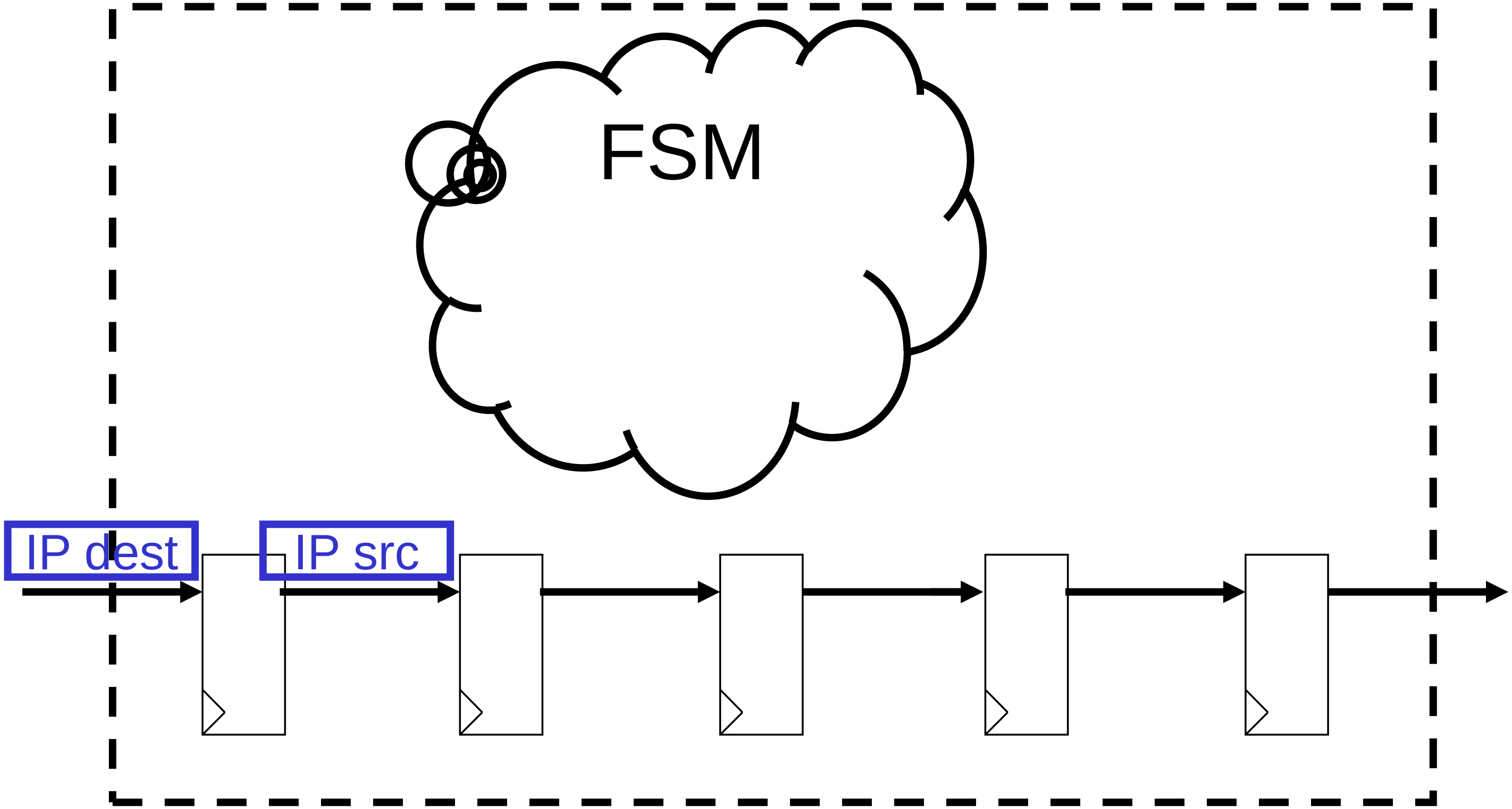
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



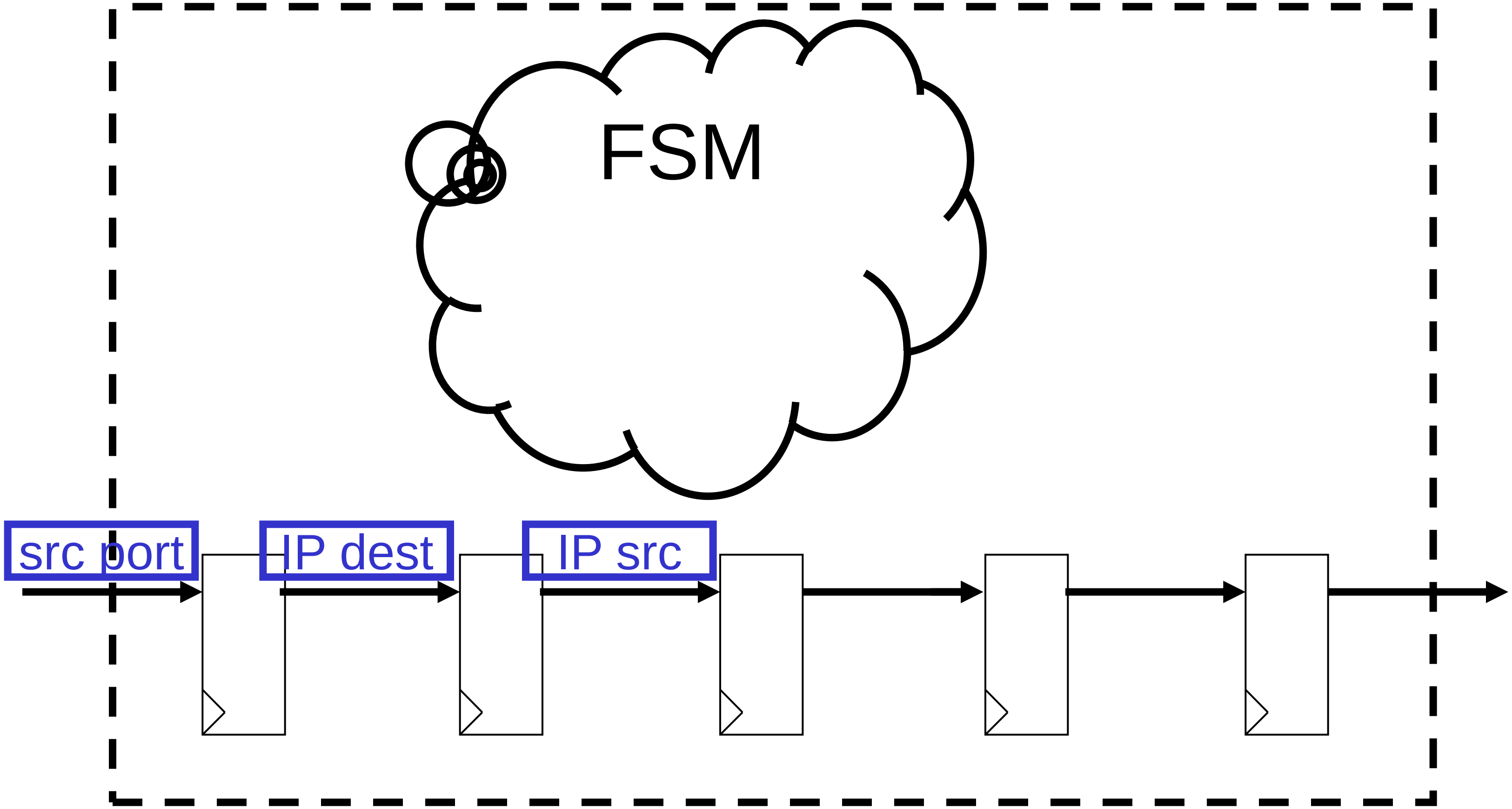
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



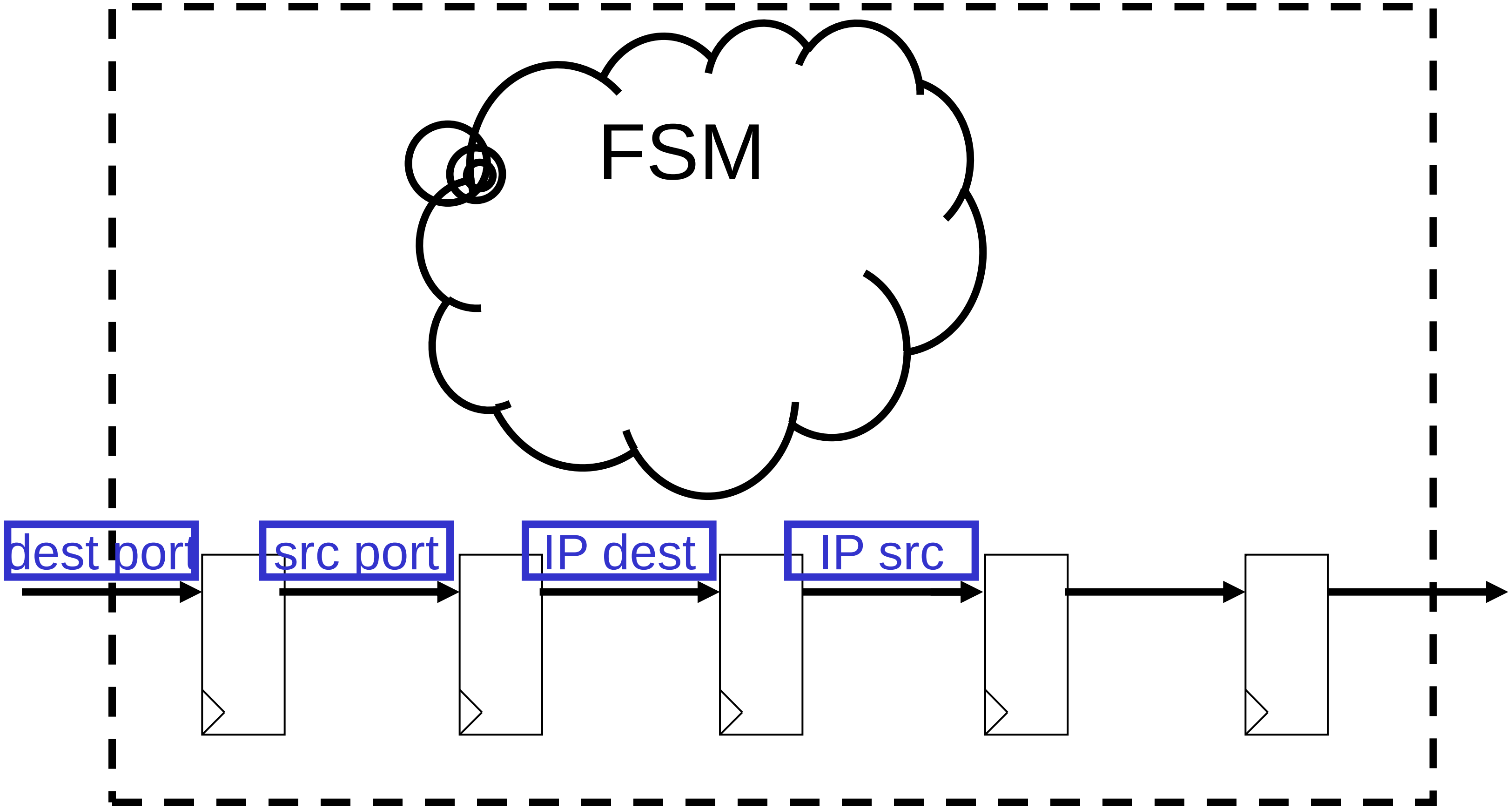
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. "Corn")
- Modify payload based on header information



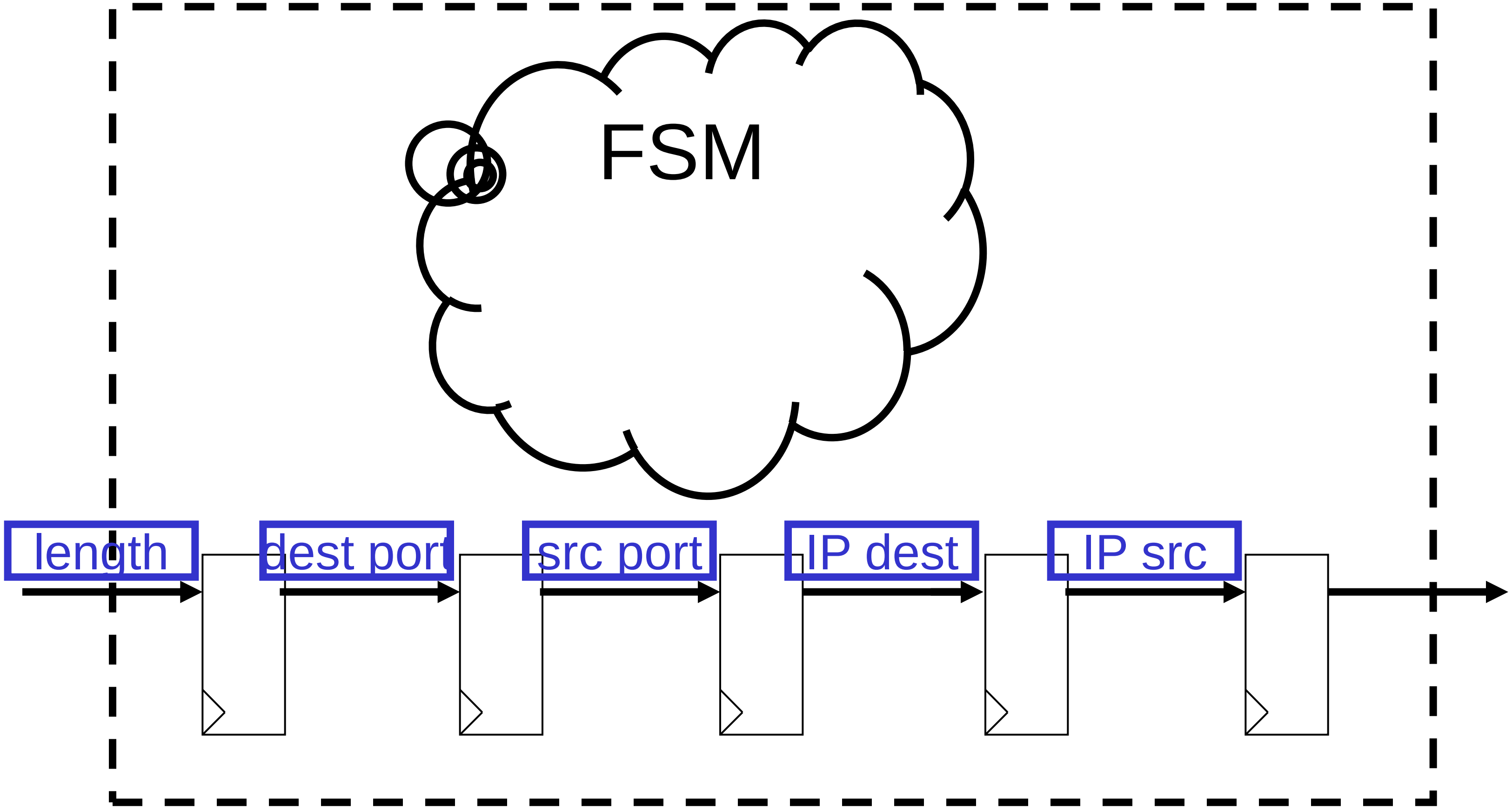
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



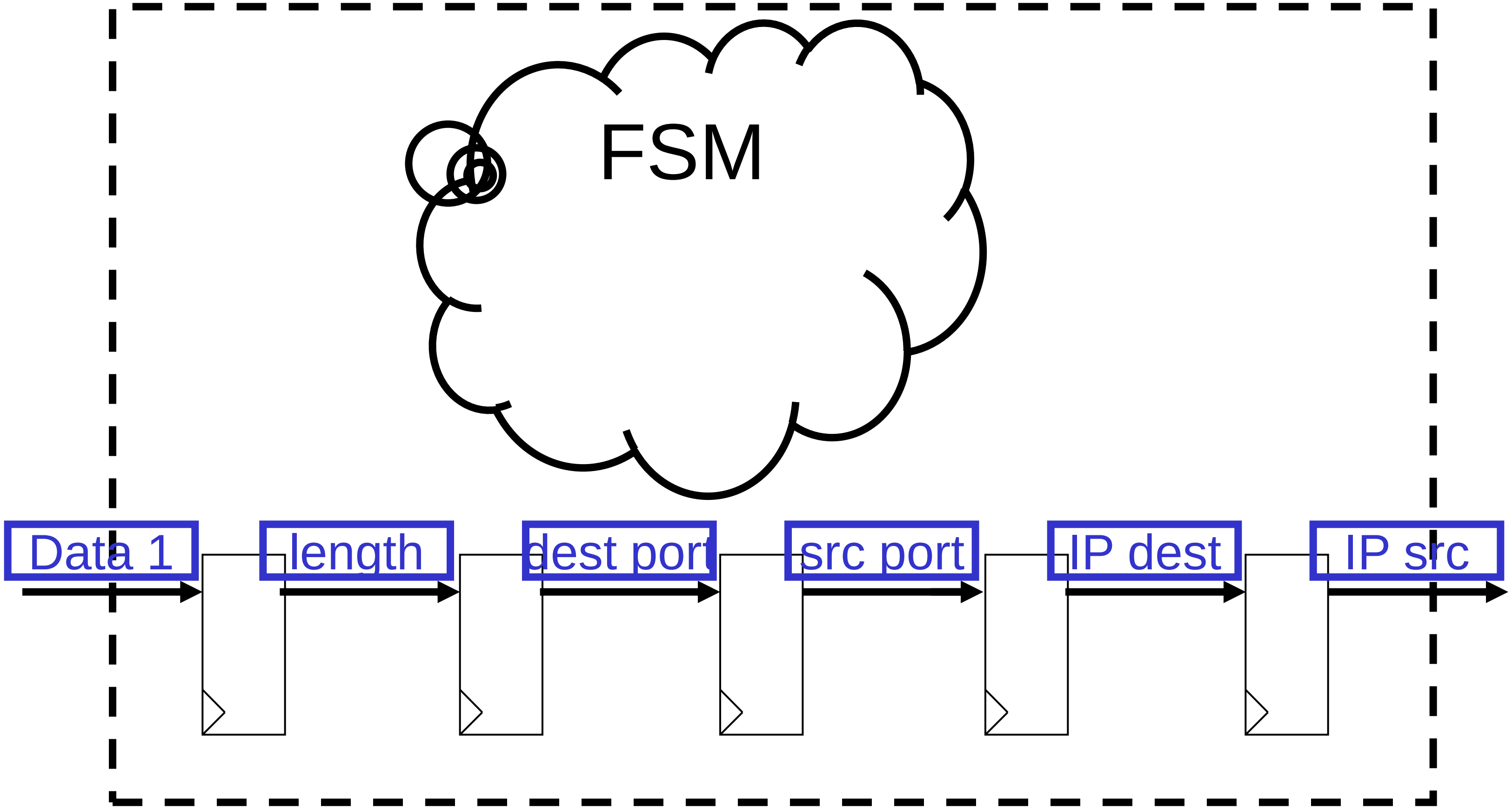
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. "Corn")
- Modify payload based on header information



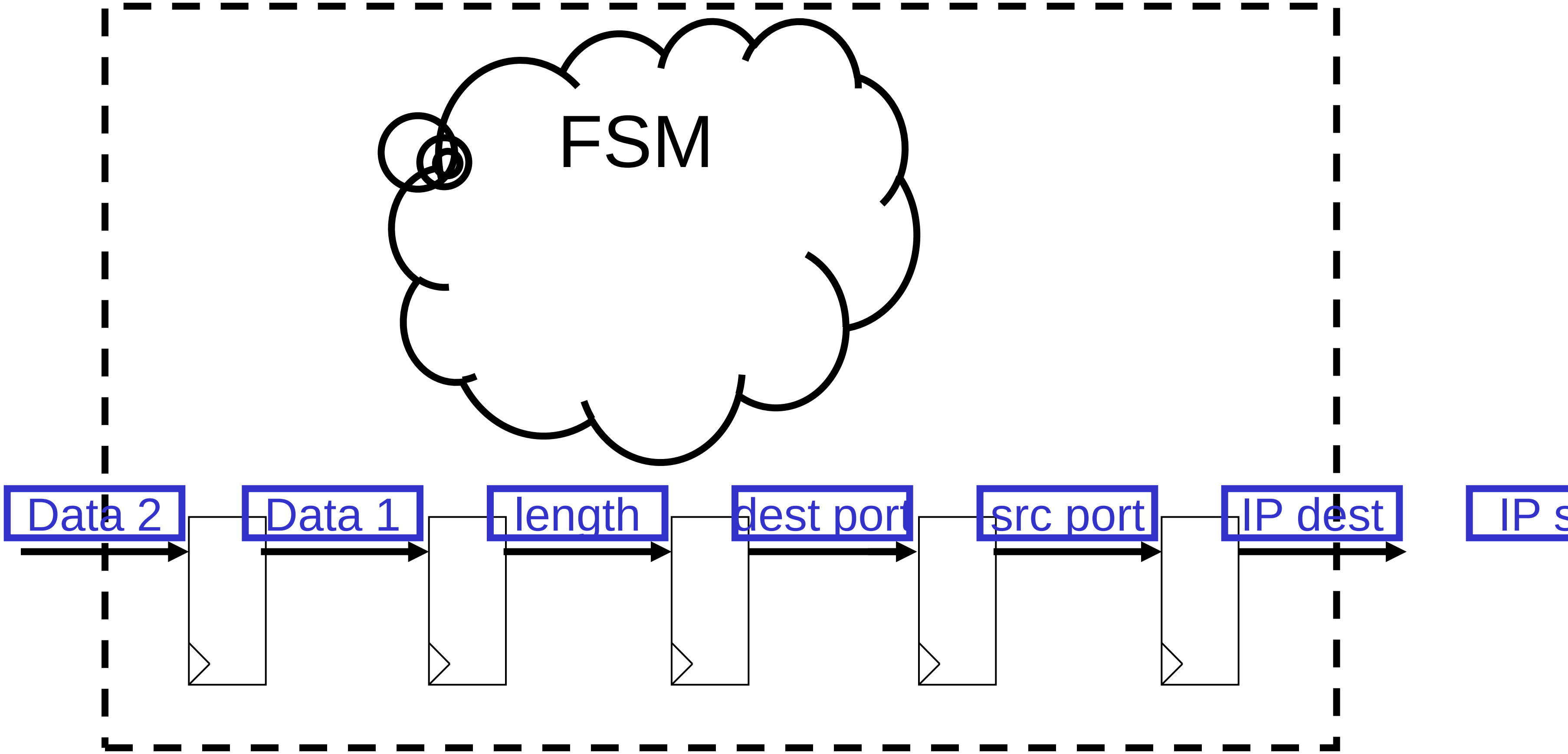
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



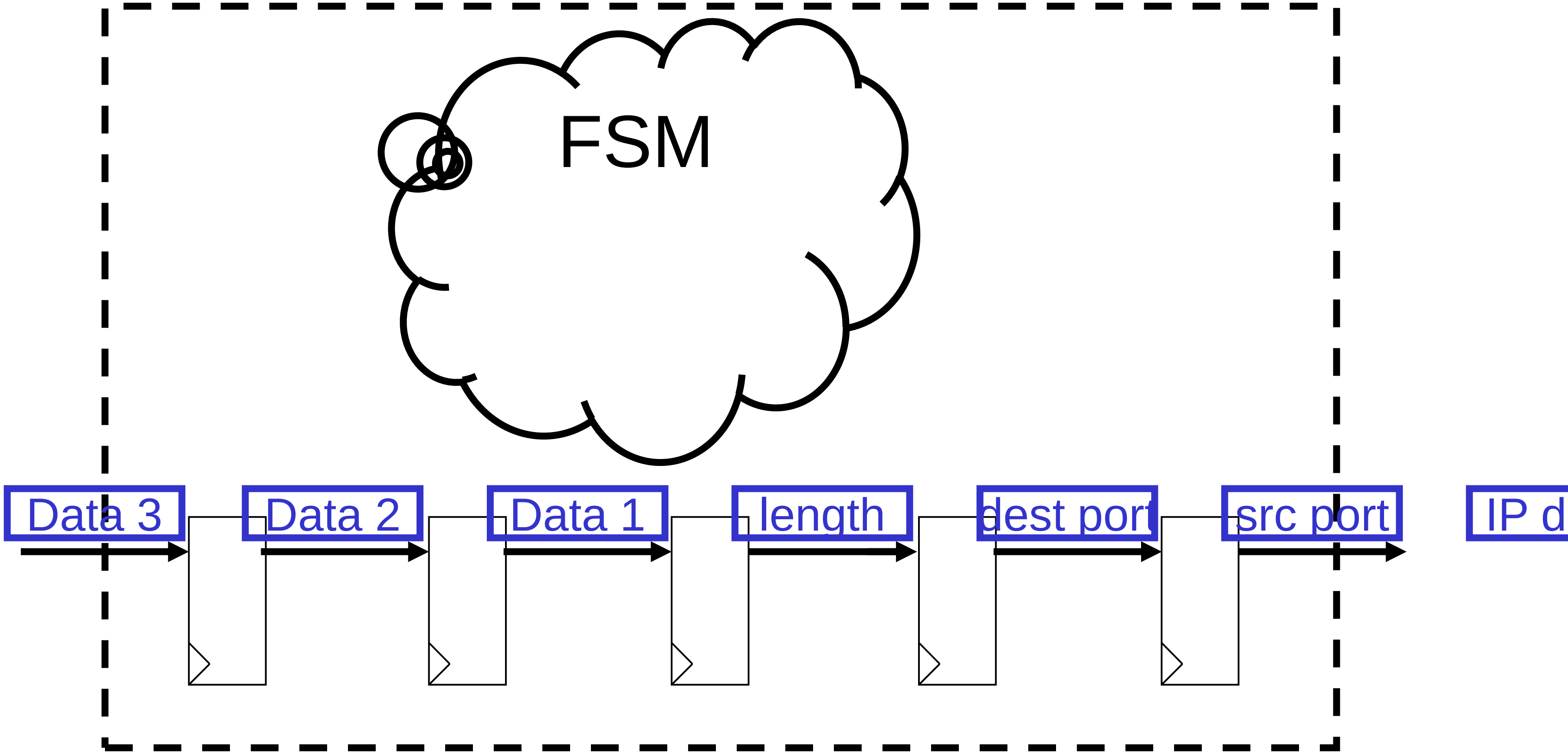
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



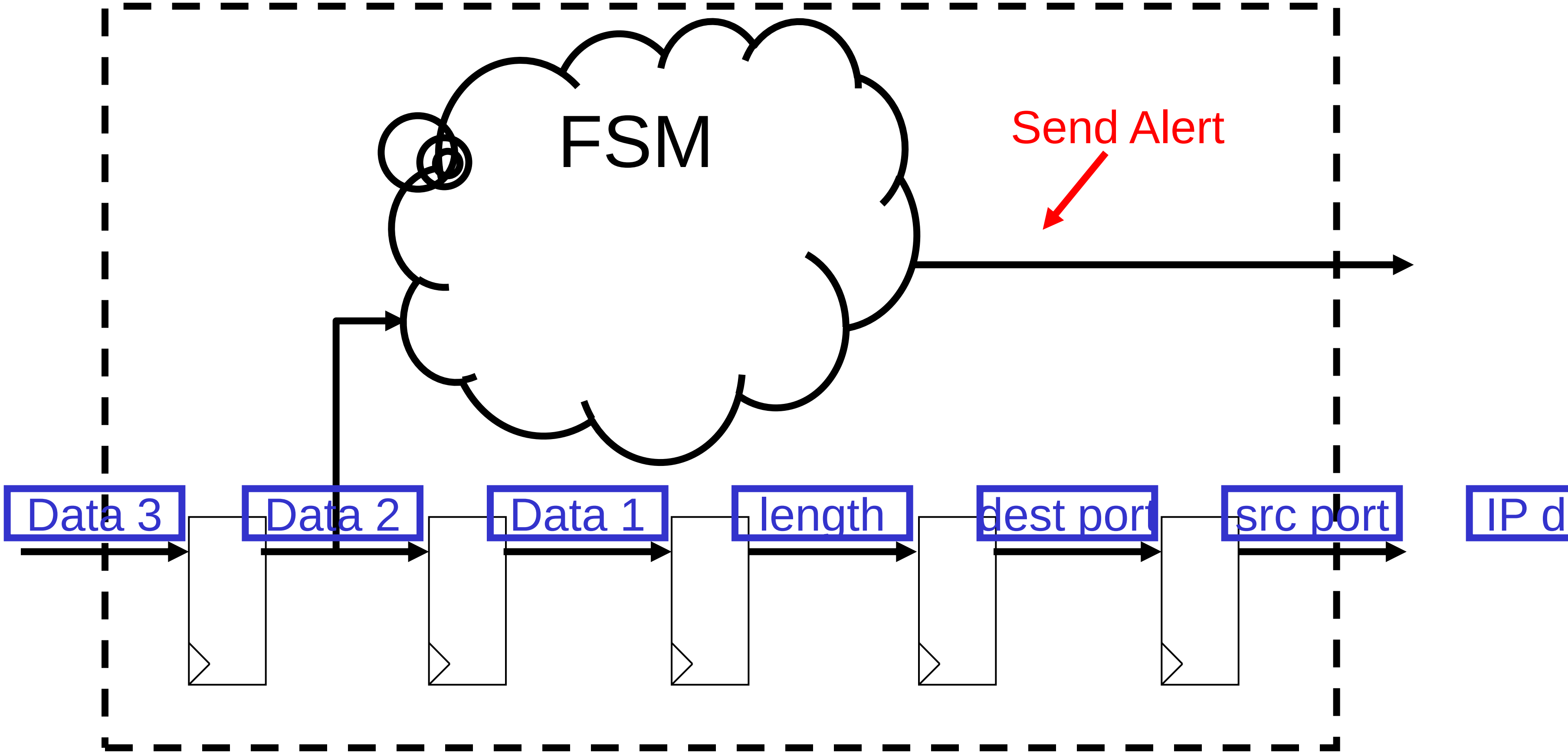
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



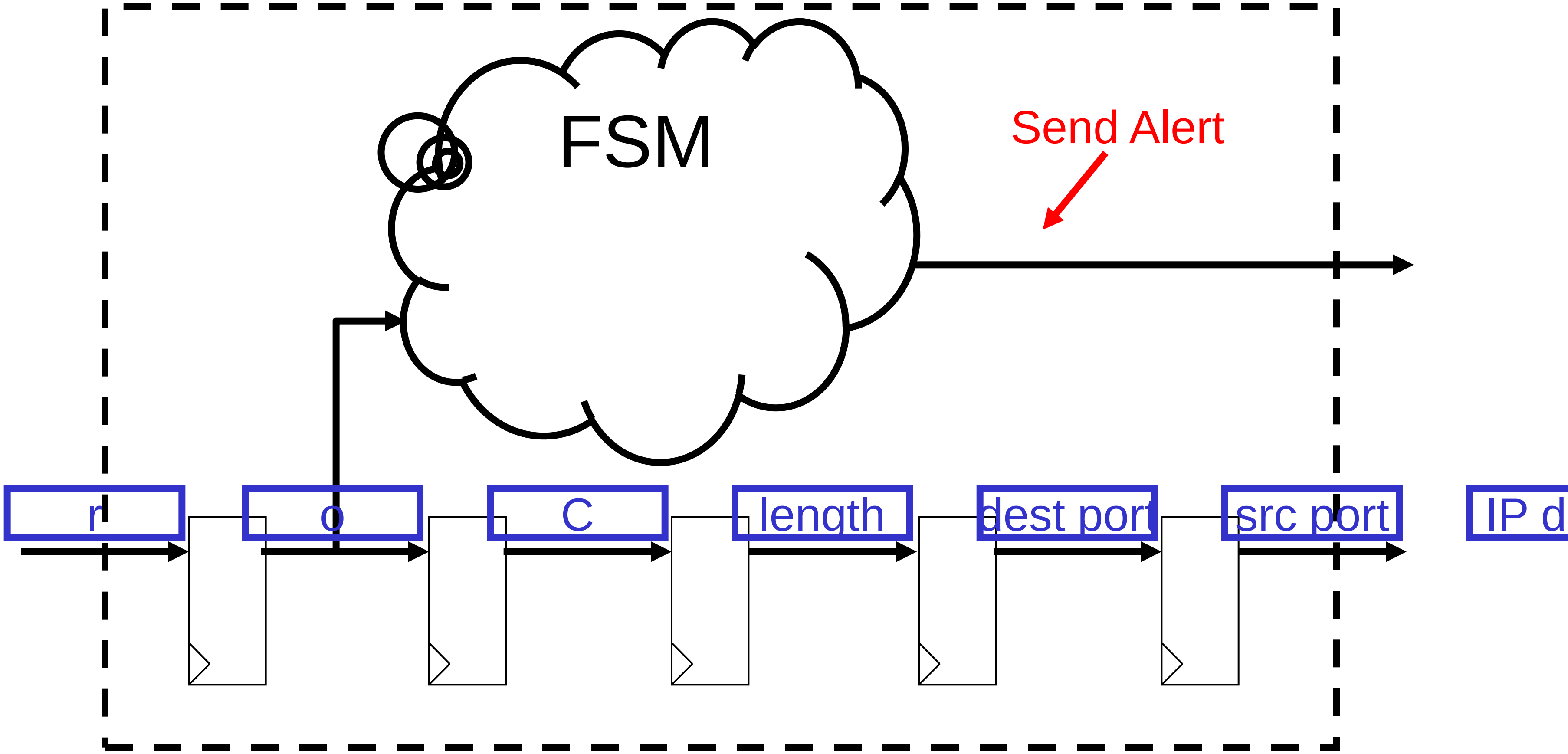
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. "Corn")
- Modify payload based on header information



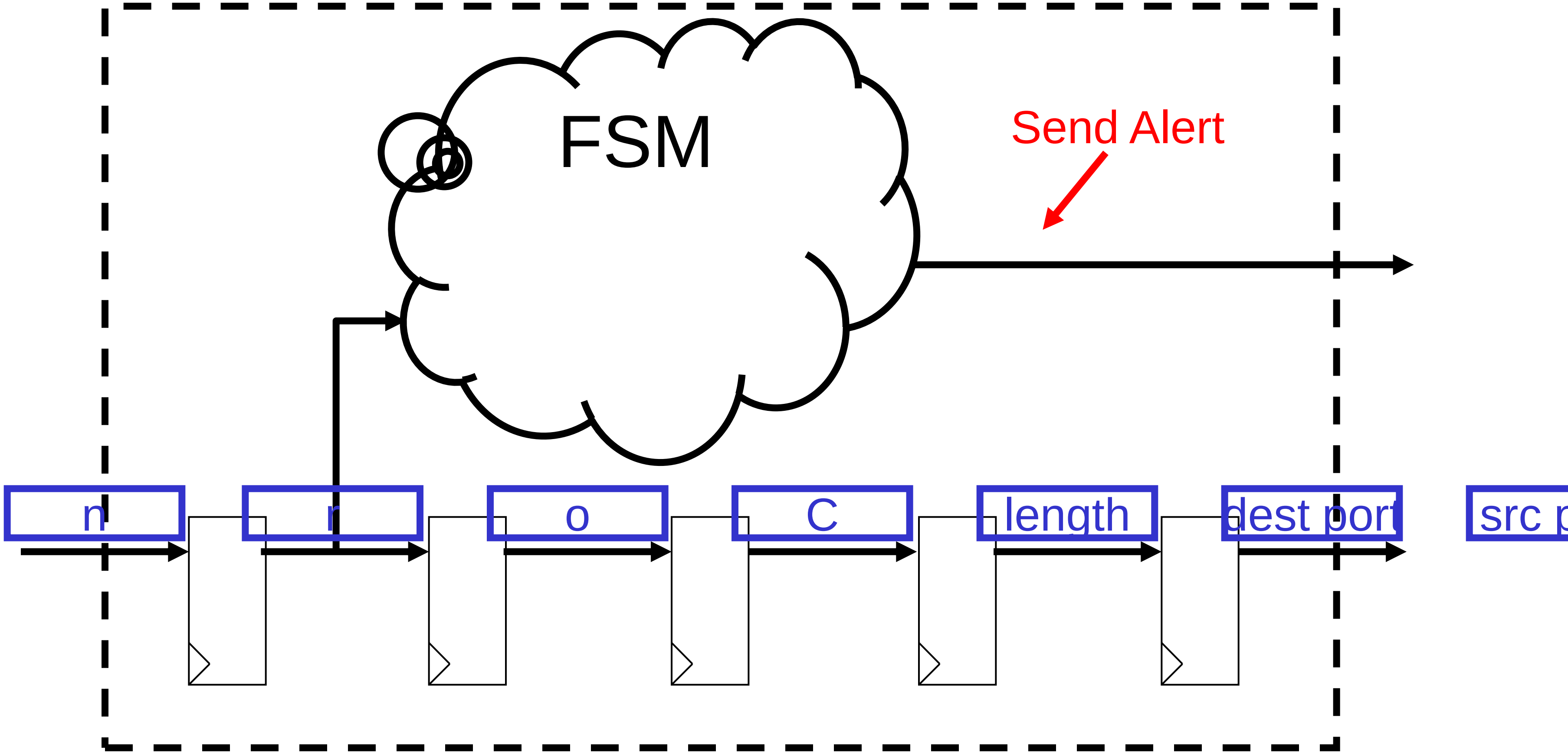
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



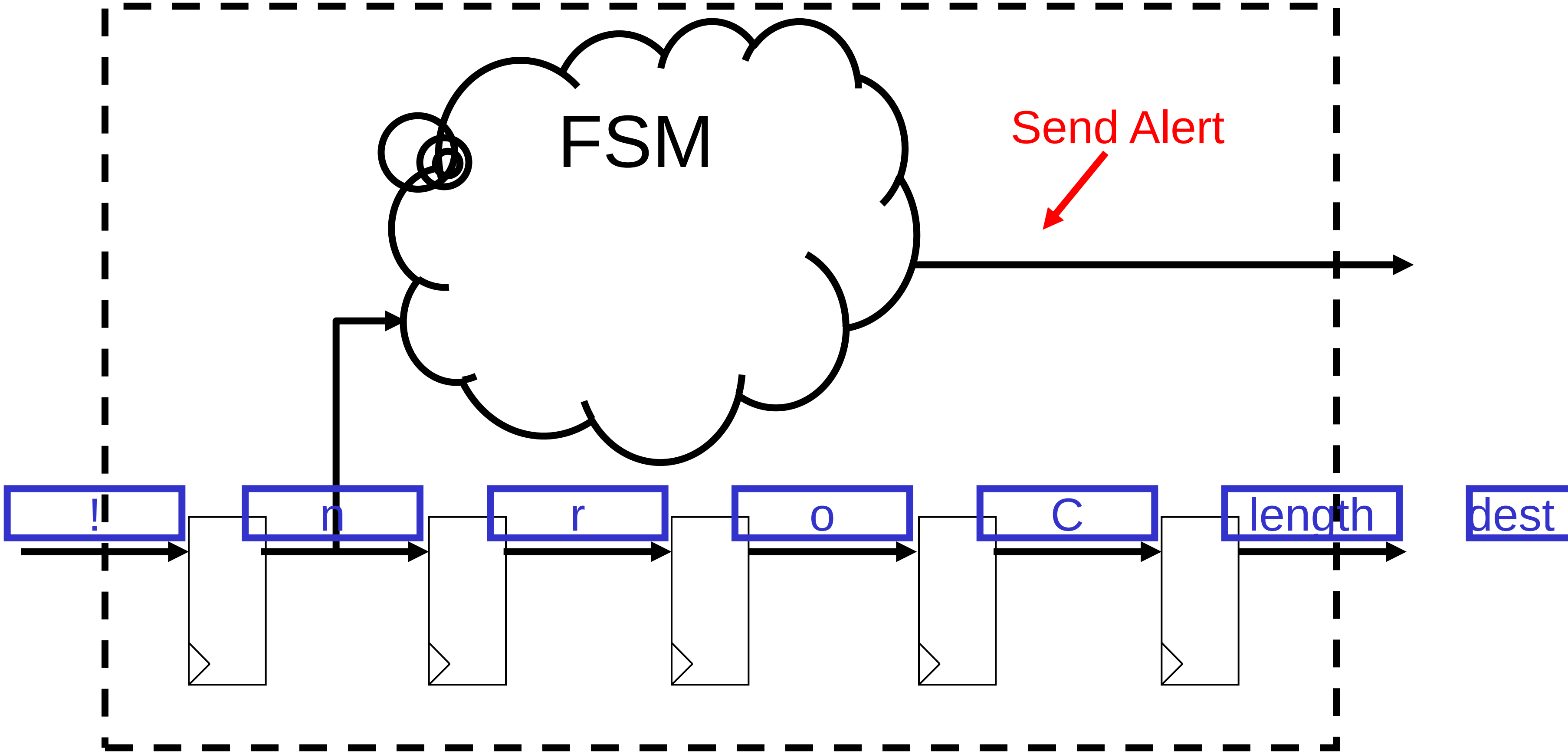
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



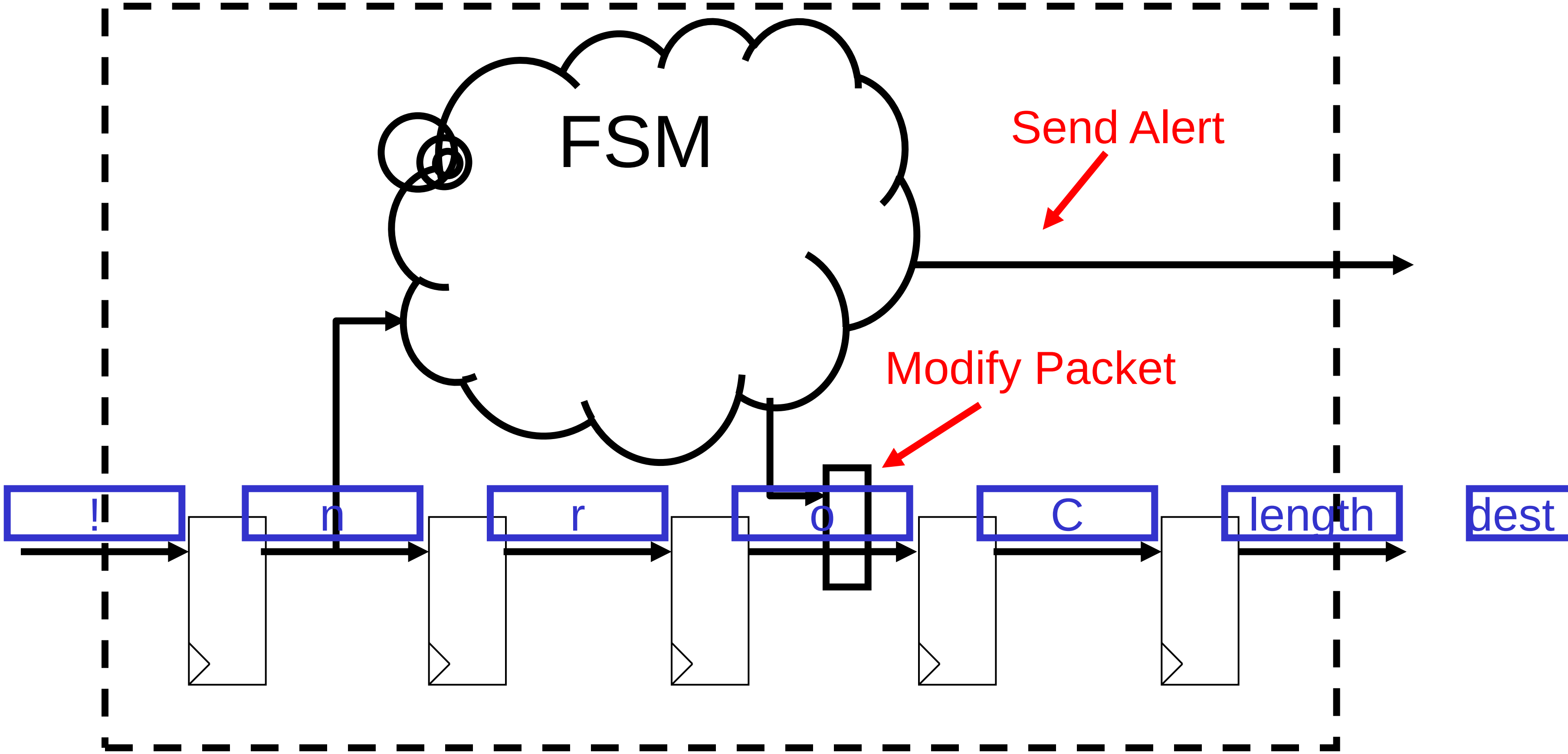
Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. “Corn”)
- Modify payload based on header information



Streaming Network application (MP1)

- Process UDP packet headers (event driven)
- Detect patterns in payload (e.g. "Corn")
- Modify payload based on header information



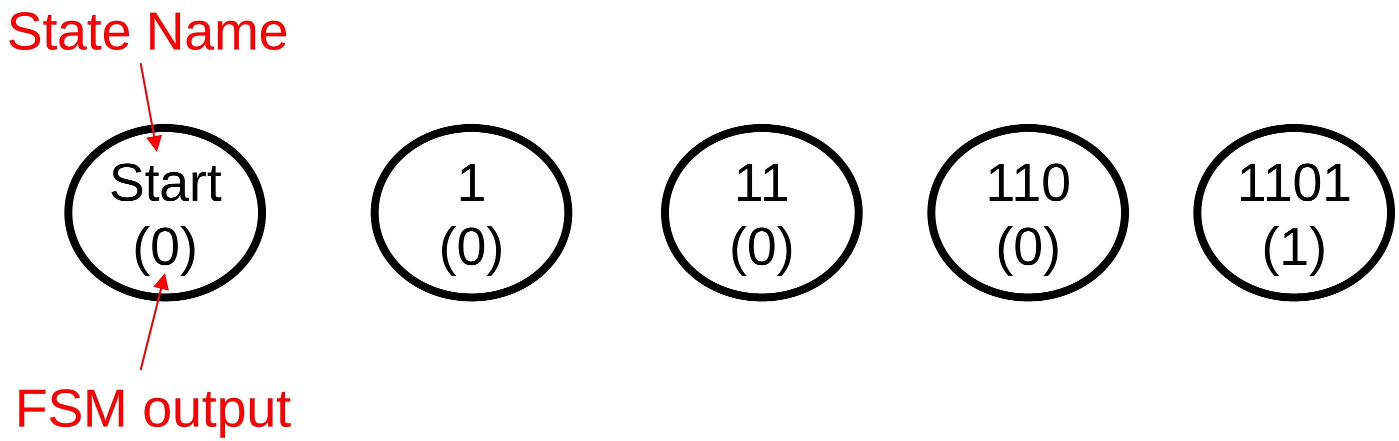
Moore and Mealy FSMs

- Moore: Output is only a function of the current state

- Mealy: Output is a function of the current state and input ("Mealy is more")

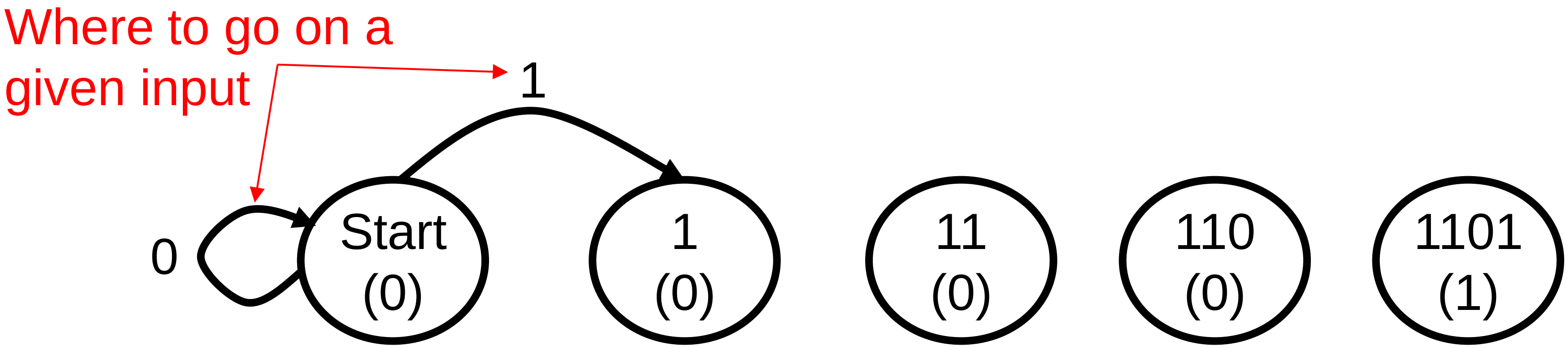
Moore FSM

- Moore: Output is only a function of the current state
- Example detect <u>every</u> occurrence of "1101"



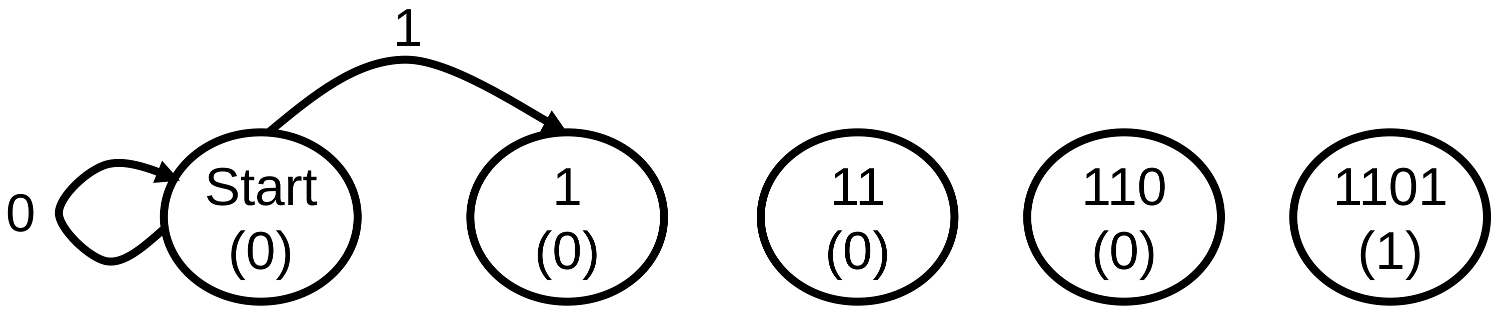
Moore FSM

- Moore: Output is only a function of the current state
- Example detect every occurrence of "1101"



Moore FSM

- Moore: Output is only a function of the current state
- Example detect every occurrence of “1101”

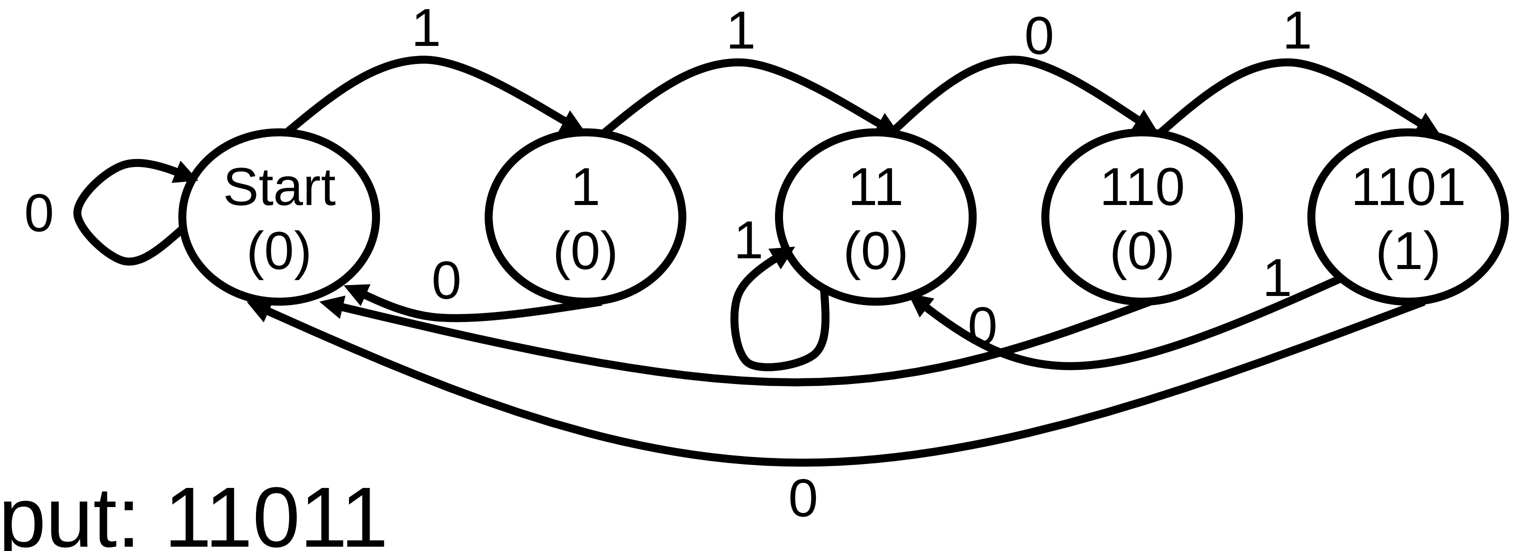


Input: 1

Output: 0

Moore FSM

- Moore: Output is only a function of the current state
- Example detect every occurrence of “1101”

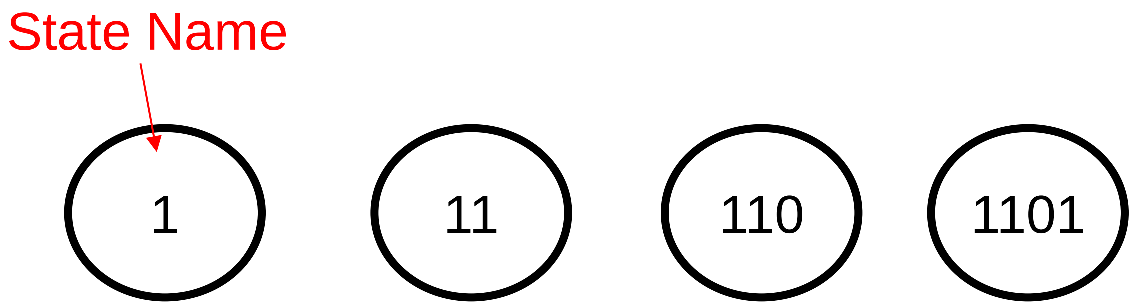


Input: 11011

Output: 00010

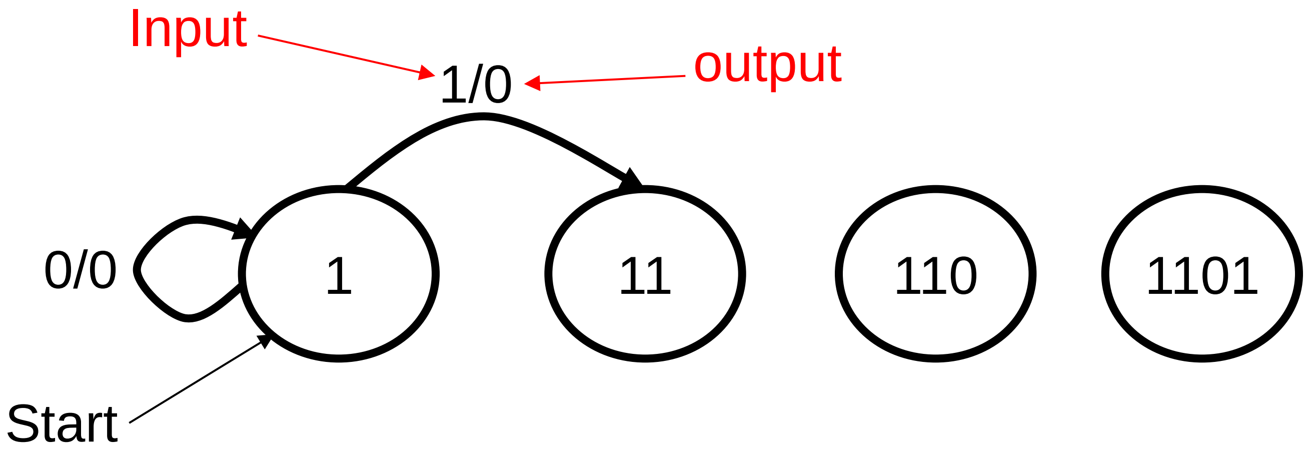
Mealy FSM

- Moore: Output a function of the current state, and input
- Example detect <u>every</u> occurrence of “1101”



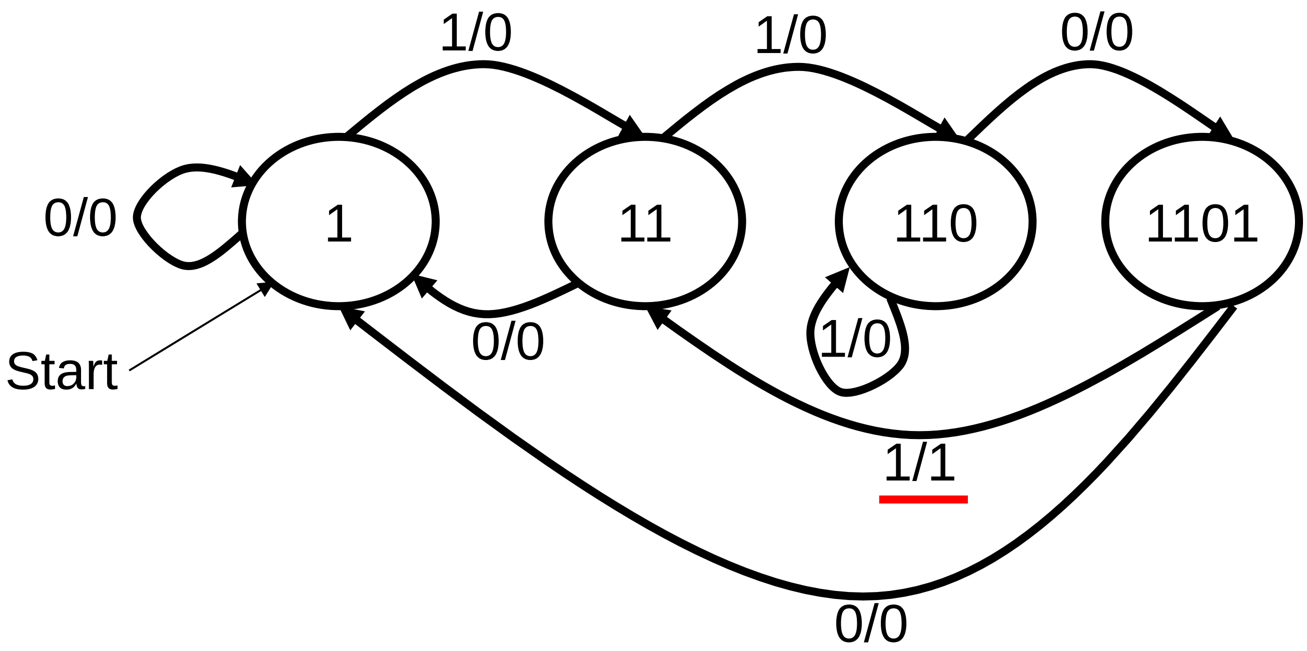
Mealy FSM

- Moore: Output a function of the current state, and input
- Example detect <u>every</u> occurrence of “1101”



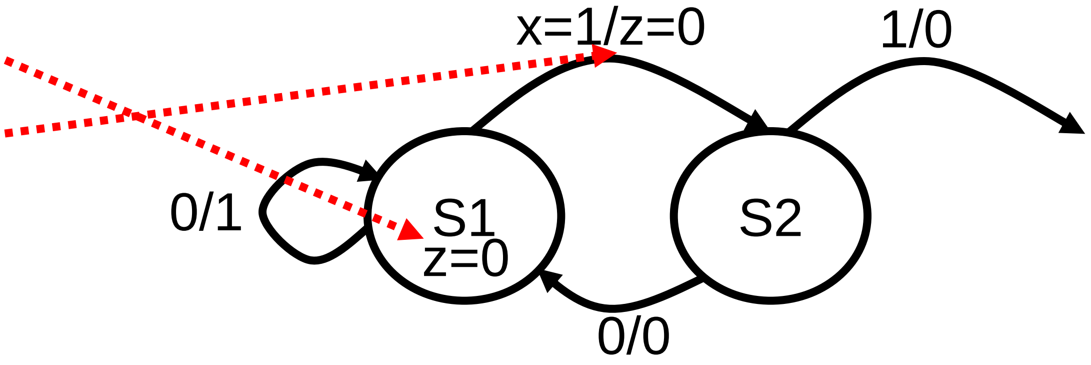
Mealy FSM

- Mealy: Output a function of the current state, and input
- Example detect every occurrence of “1101”

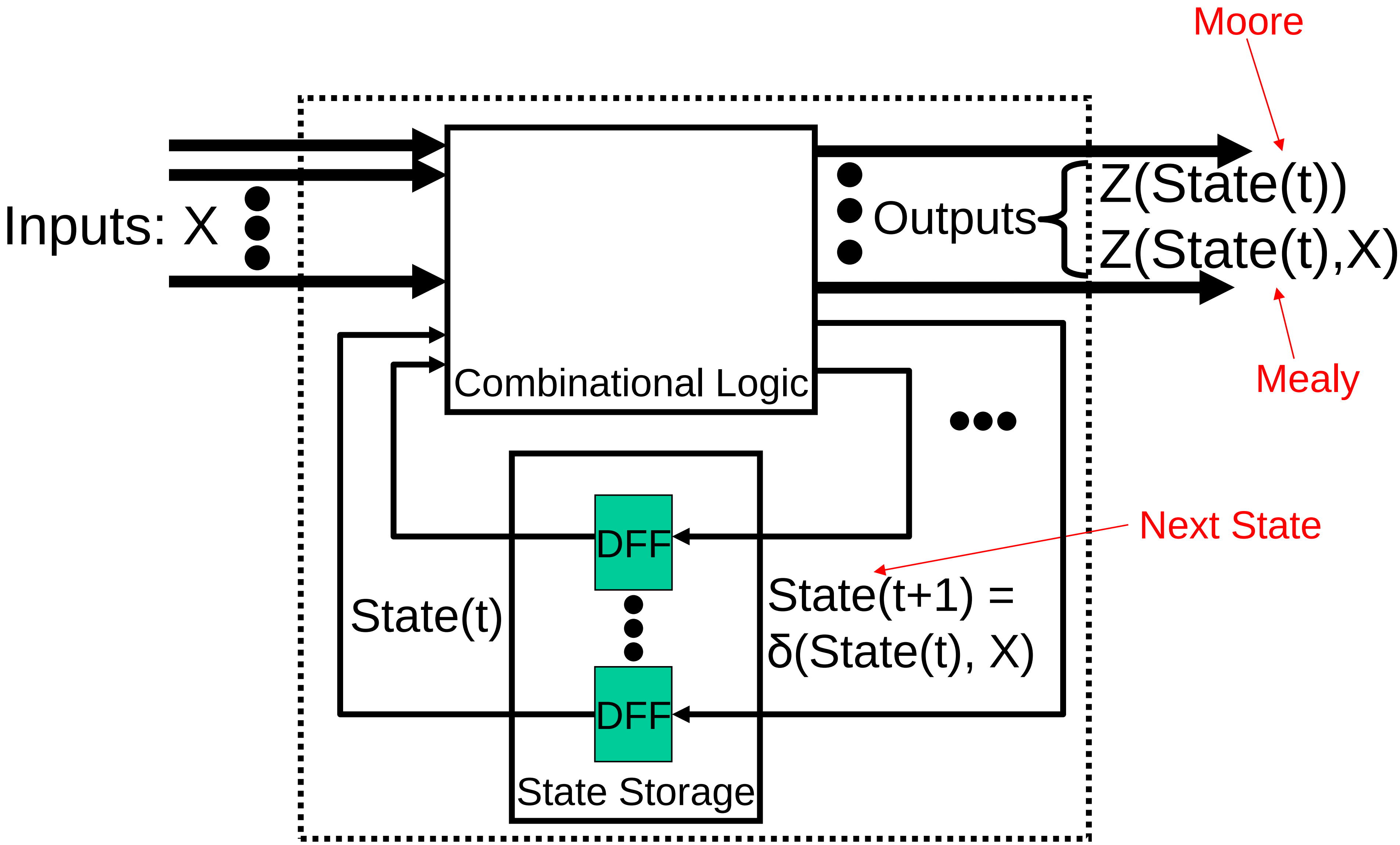


FSM: General Circuit Architecture

- Let:
  - X be inputs
  - Z be outputs
  - State(t) be the state of the FSM at the current time
  - State(t+1) be the next state of the FSM
  - δ be the transition between states
- State(t+1) = δ(State(t), X)
- Output
  - Moore: Z(State(t))
  - Mealy: Z(State(t), X)



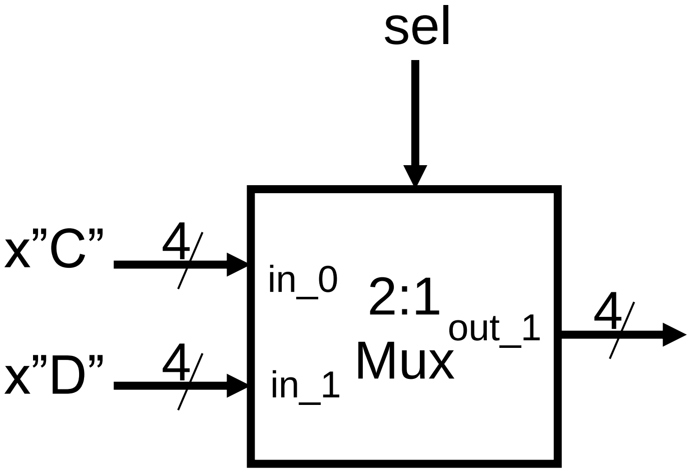
FSM: General Circuit Architecture



VHDL: IF and CASE constructs

- IF THEN ELSE can be mapped to a 2:1 Multiplexer (Mux)

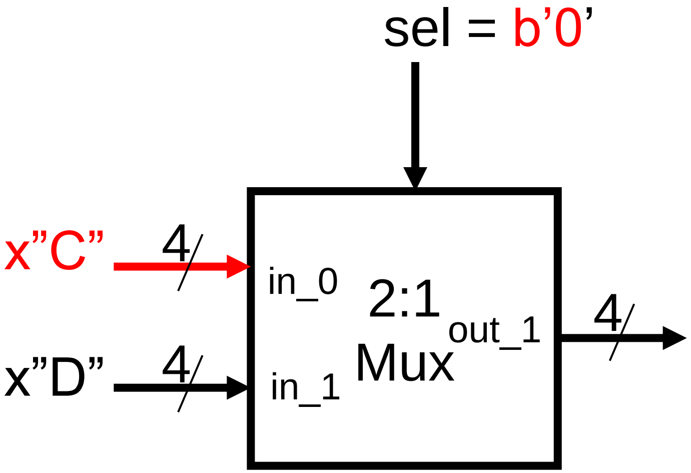
```
IF (sel = ‘0’)  THEN
  out_1 <= in_0;
ELSE
  out_1 <= in_1
END IF;
```



VHDL: IF and CASE constructs

- IF THEN ELSE can be mapped to a 2:1 Multiplexer (Mux)

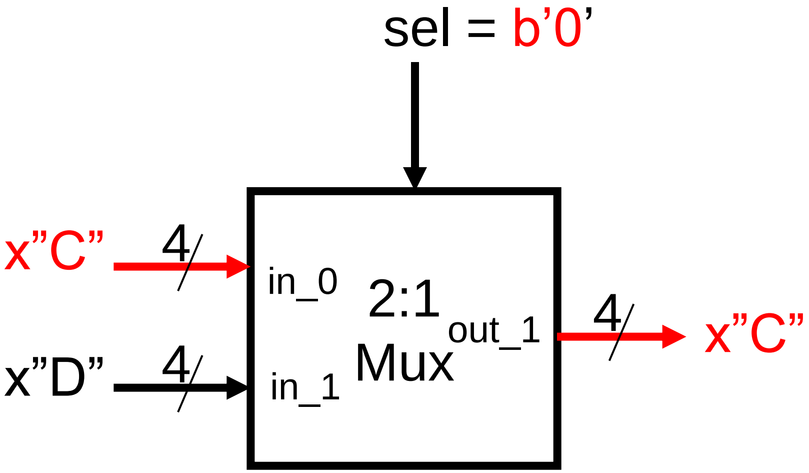
```
IF (sel = ‘0’)  THEN
  out_1 <= in_0;
ELSE
  out_1 <= in_1
END IF;
```



VHDL: IF and CASE constructs

- IF THEN ELSE can be mapped to a 2:1 Multiplexer (Mux)

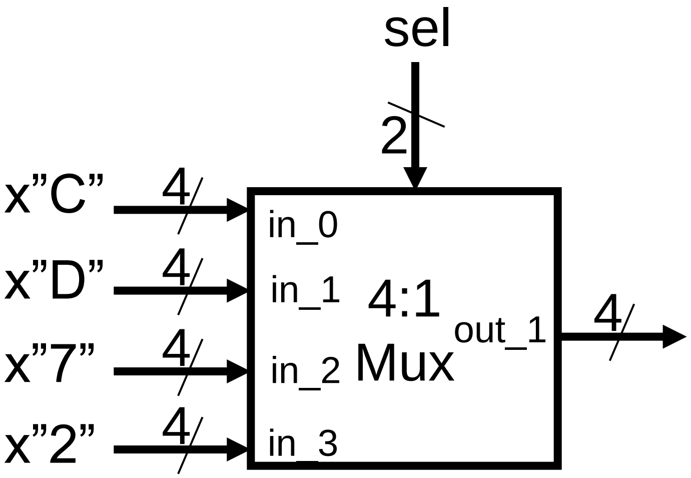
```
IF (sel = ‘0’)  THEN
  out_1 <= in_0;
ELSE
  out_1 <= in_1
END IF;
```



VHDL: IF and CASE constructs

- Mapping a CASE statement to a 4:1 Mux

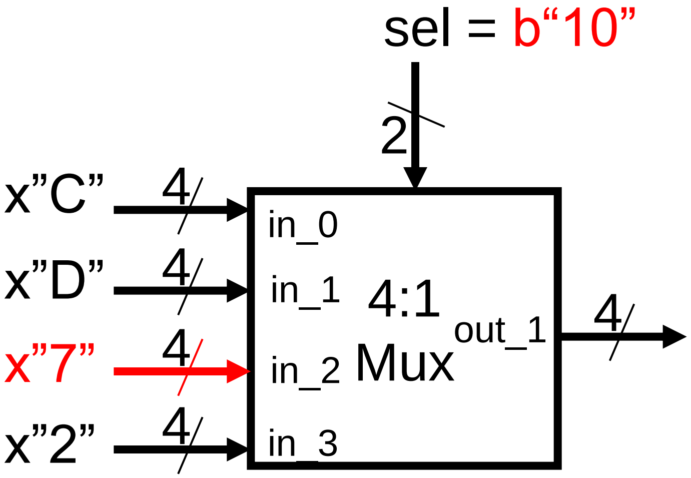
```
CASE sel is
WHEN “00” =>
   out_1 <= in_0;
WHEN “01” =>
  out_1 <= in_1;
WHEN “10” =>
  out_1 <= in_2;
WHEN “11” =>
  out_1 <= in_3
WHEN OTHERS =>
  out_1 <= in_0;
END CASE;
```



VHDL: IF and CASE constructs

- Mapping a CASE statement to a 4:1 Mux

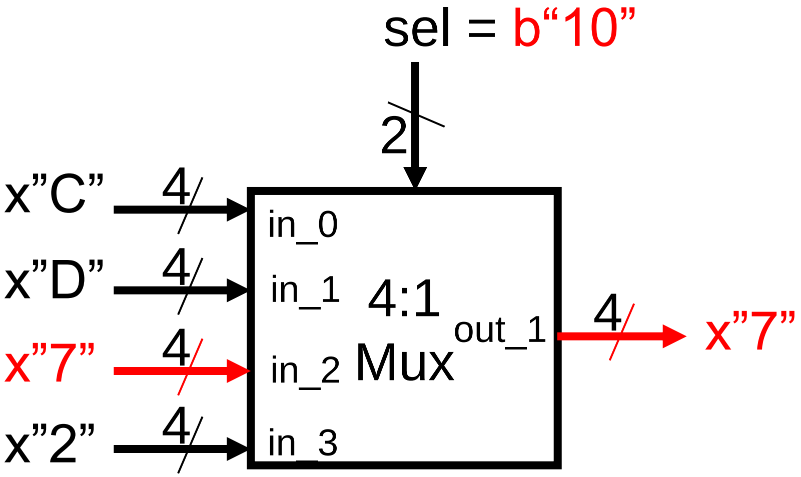
```
CASE sel is
WHEN “00” =>
   out_1 <= in_0;
WHEN “01” =>
  out_1 <= in_1;
WHEN “10” =>
  out_1 <= in_2;
WHEN “11” =>
  out_1 <= in_3
WHEN OTHERS =>
  out_1 <= in_0;
END CASE;
```



VHDL: IF and CASE constructs

- Mapping a CASE statement to a 4:1 Mux

```
CASE sel is
WHEN “00” =>
   out_1 <= in_0;
WHEN “01” =>
  out_1 <= in_1;
WHEN “10” =>
  out_1 <= in_2;
WHEN “11” =>
  out_1 <= in_3
WHEN OTHERS =>
  out_1 <= in_0;
END CASE;
```

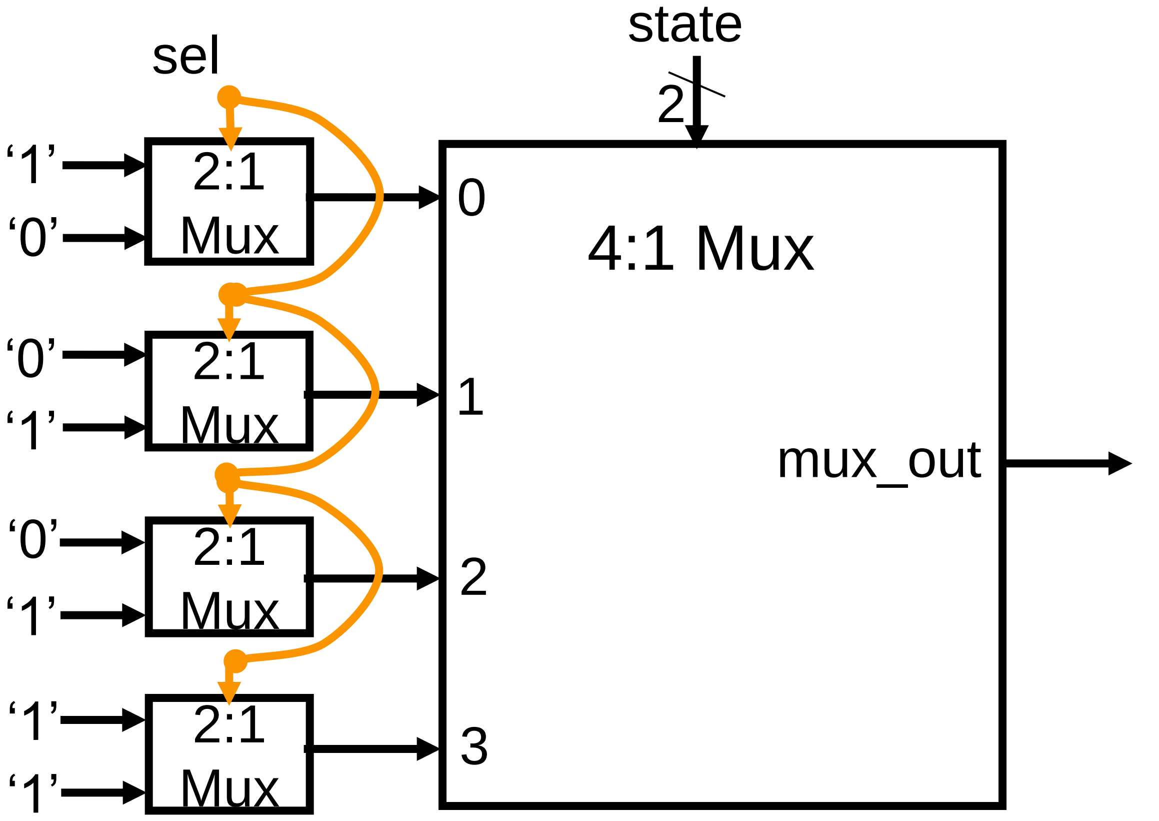


Why do we need others here?

VHDL: IF and CASE constructs

- Mapping an IF nested in CASE to hardware

```
CASE state is
WHEN state_1 =>
  IF (sel = ‘0’)  THEN
    mux_out <= ‘1’;
  ELSE
    mux_out <= ‘0’;
  END IF;
WHEN state_11 =>
  -- similar code
WHEN state_110 =>
 IF (sel = ‘0’)  THEN
    mux_out <= ‘0’;
  ELSE
    mux_out <= ‘1’;
WHEN state_1101 =>
  --similar code
END CASE;
```



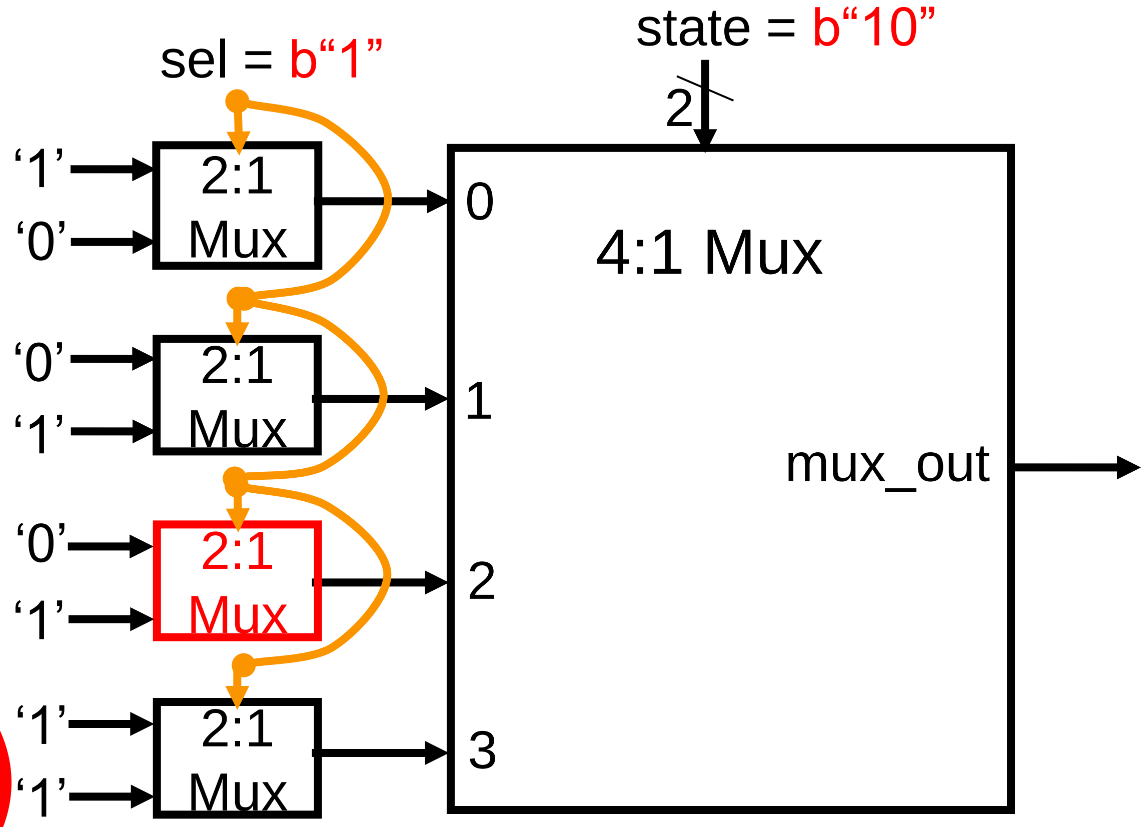
Enumerated Type

state = {state_1, state_11, state_110, state_1101} =
{“00”, ”01”, ”10”, ”11”}

VHDL: IF and CASE constructs

- Mapping an IF nested in CASE to hardware

```
CASE state is
WHEN state_1 =>
  IF (sel = ‘0’)  THEN
    mux_out <= ‘1’;
  ELSE
    mux_out <= ‘0’;
  END IF;
WHEN state_11 =>
  -- similar code
WHEN state_110 =>
 IF (sel = ‘0’)  THEN
    mux_out <= ‘0’;
  ELSE
    mux_out <= ‘1’;
WHEN state_1101 =>
  --similar code
END CASE;
```



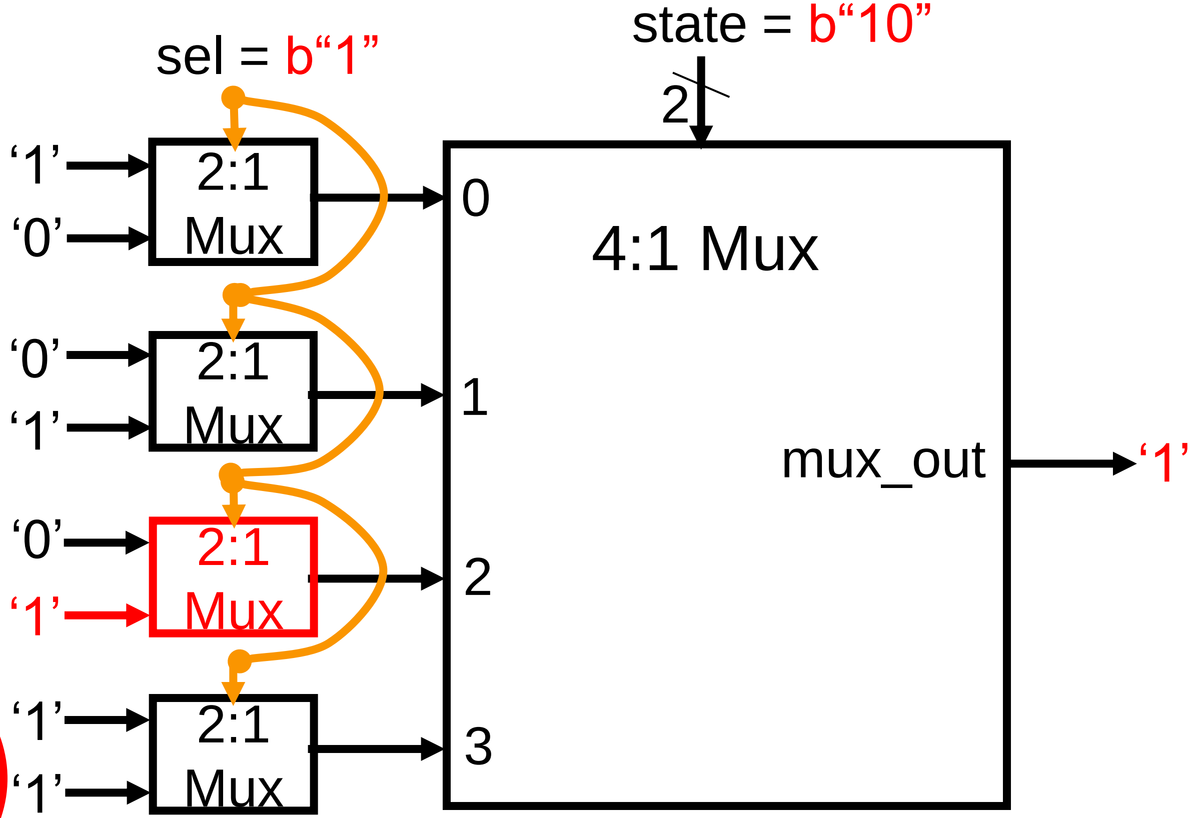
Enumerated Type

state = {state_1, state_11, state_110, state_1101} =
{“00”, ”01”, ”10”, ”11”}

VHDL: IF and CASE constructs

- Mapping an IF nested in CASE to hardware

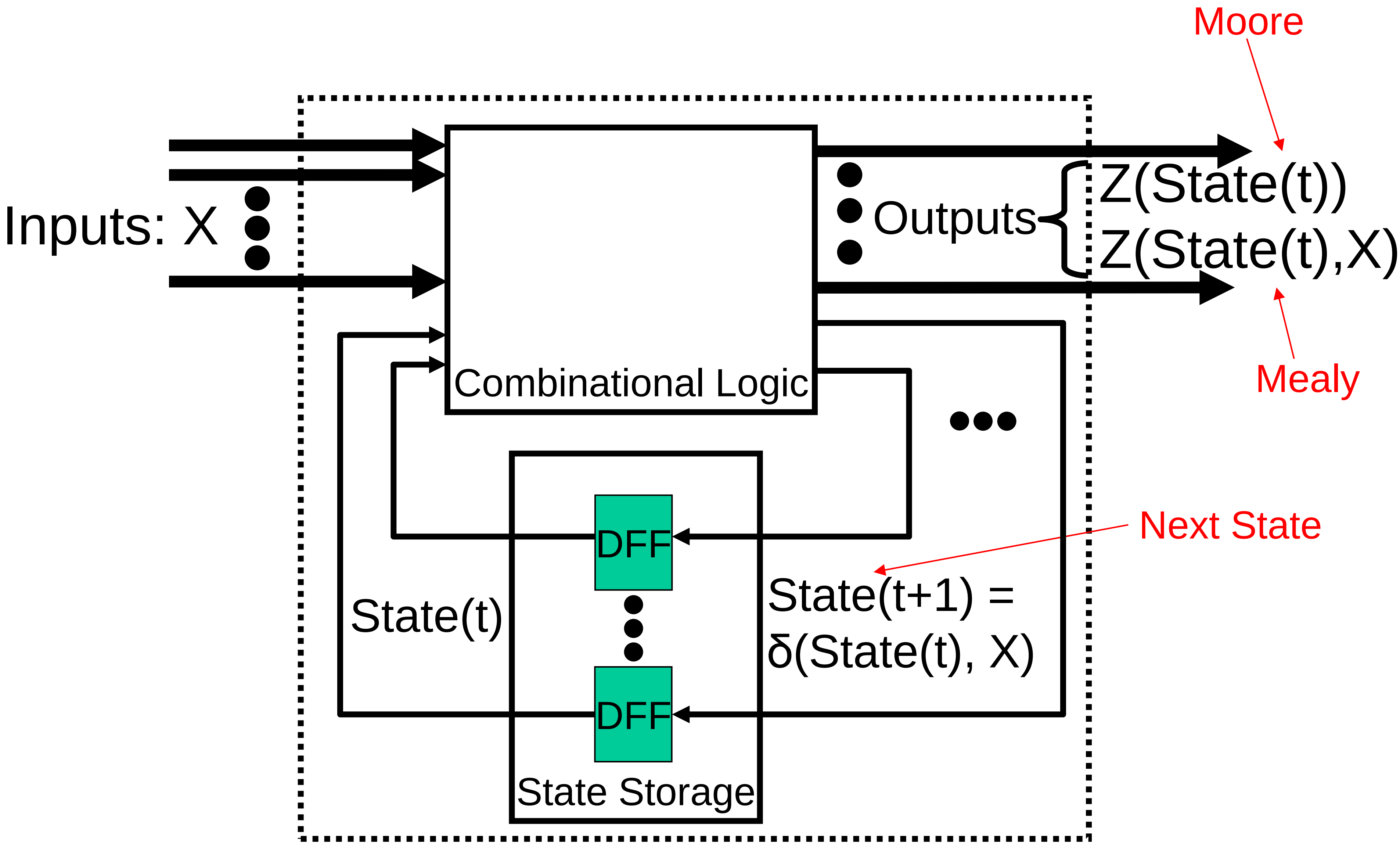
```
CASE state is
WHEN state_1 =>
  IF (sel = '0')  THEN
    mux_out <= '1';
  ELSE
    mux_out <= '0';
  END IF;
WHEN state_11 =>
  -- similar code
WHEN state_110 =>
 IF (sel = '0')  THEN
    mux_out <= '0';
  ELSE
    mux_out <= '1';
WHEN state_1101 =>
  --similar code
END CASE;
```



Enumerated Type

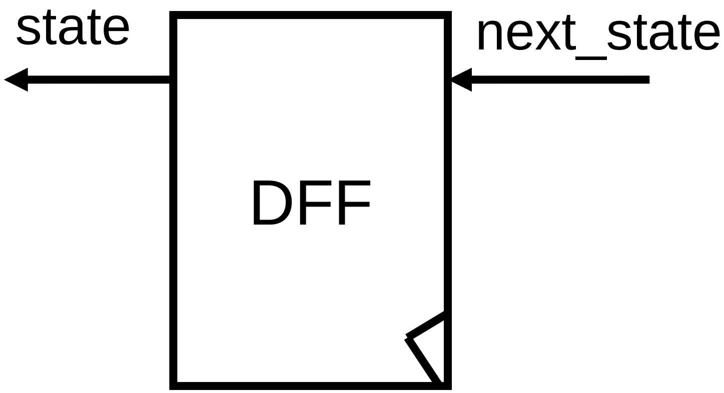
state = {state_1, state_11, state_110, state_1101} =
{"00", "01", "10", "11"}

FSM: General Circuit Architecture



VHDL for Mealy (“1101”) Example

```
-- Store the “state”
Update_State: process(clk)
begin
  if(clk’event and clk=‘1’) then
    state <= next_state;
  end if;
end process Update_State;
```

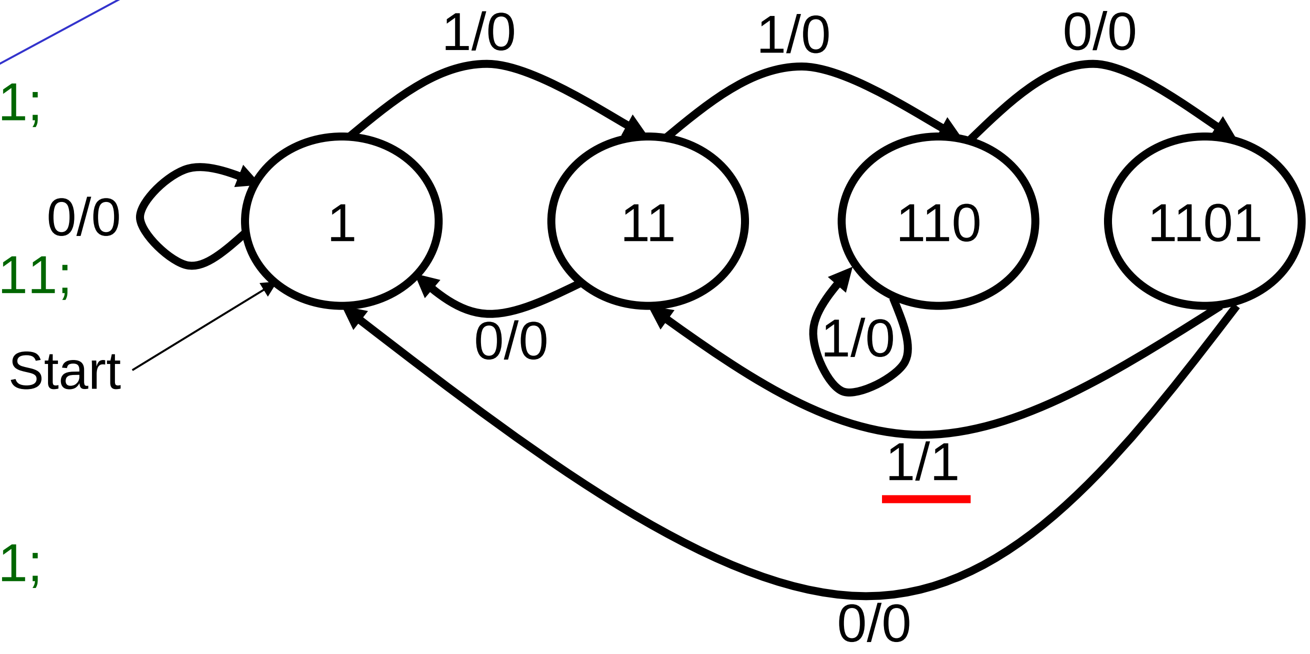


VHDL for Mealy (“1101”) Example

```
-- Compute combinational logic
Combinational: process(x, state)
begin
 case state is
 when state_1 =>
   if(x = ‘0’) then
     z <= ‘0‘;
     next_state <= state_1;
   else
     z <= ‘0‘;
     next_state <= state_11;
   end if;
 when state_11 =>
   if(x = ‘0’) then
     z <= ‘0’;
     next_state <= state_1;
   else
     z <= ‘0‘;
     next_state <= state_110 ;
   end if;
```

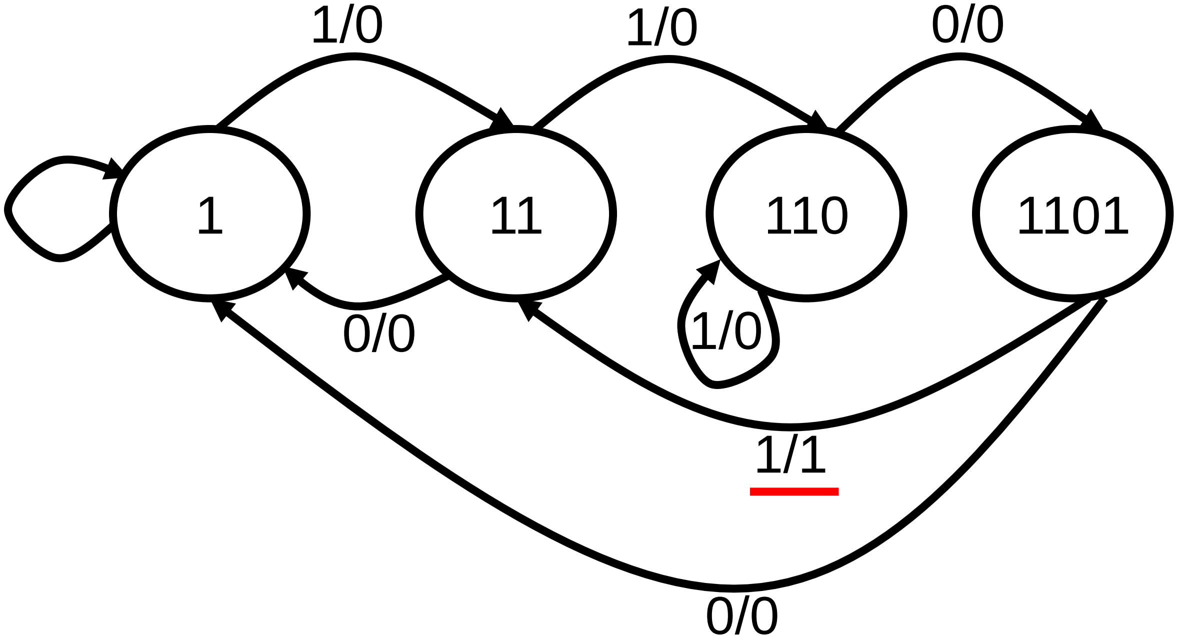
Compute output

Compute next_state



VHDL for Mealy (“1101”) Example

```
  when state_110 =>
    if(x = ‘0’) then
      z <= ‘0‘;
      next_state <= state_1101;
    else
      z <= ‘0‘;
      next_state <= state_110;
    end if;
  when state_1101 =>
    if(x = ‘0’) then
      z <= ‘0’;
      next_state <= state_1;
    else
      z <= ‘1‘;
      next_state <= state_11;
    end if;
  end case;
end process Combinational;
```



Network Processing Example: UDP

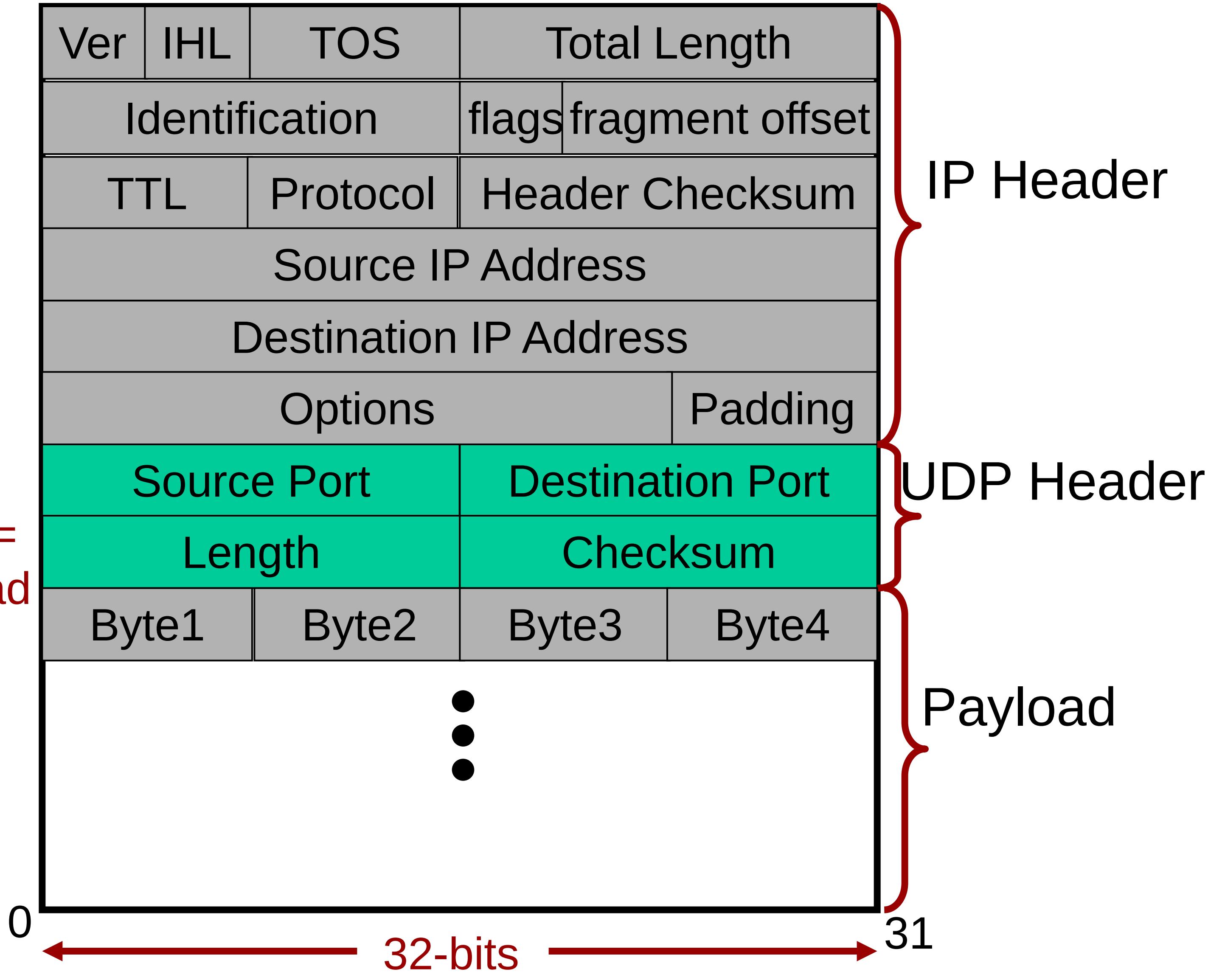
- UDP – User Datagram Protocol
  - Popular protocol for sending data over the internet (TCP is popular another protocol)
  - Typical encapsulated within IP (Internet Protocol)
    - UDP/IP
  - Gives no guarantee of delivery
    - Relies on application layer to implement reliability
    - Unlike TCP which has reliably delivery build in.
- Reference for more info on IP and UDP details
  - http://www.freesoft.org/CIE/
    - RCFs
    - Course

UDP/IP Packet Format

Note: flags 3 bits

UDP Protocol = 17

UDP length (bytes) =
UDP header+payload



Ver | IHL | TOS | Total Length

Identification | flags | fragment offset

TTL | Protocol | Header Checksum

Source IP Address

Destination IP Address

Options | Padding

Source Port | Destination Port

Length | Checksum

Byte1 | Byte2 | Byte3 | Byte4

IP Header

UDP Header

Payload

0

31

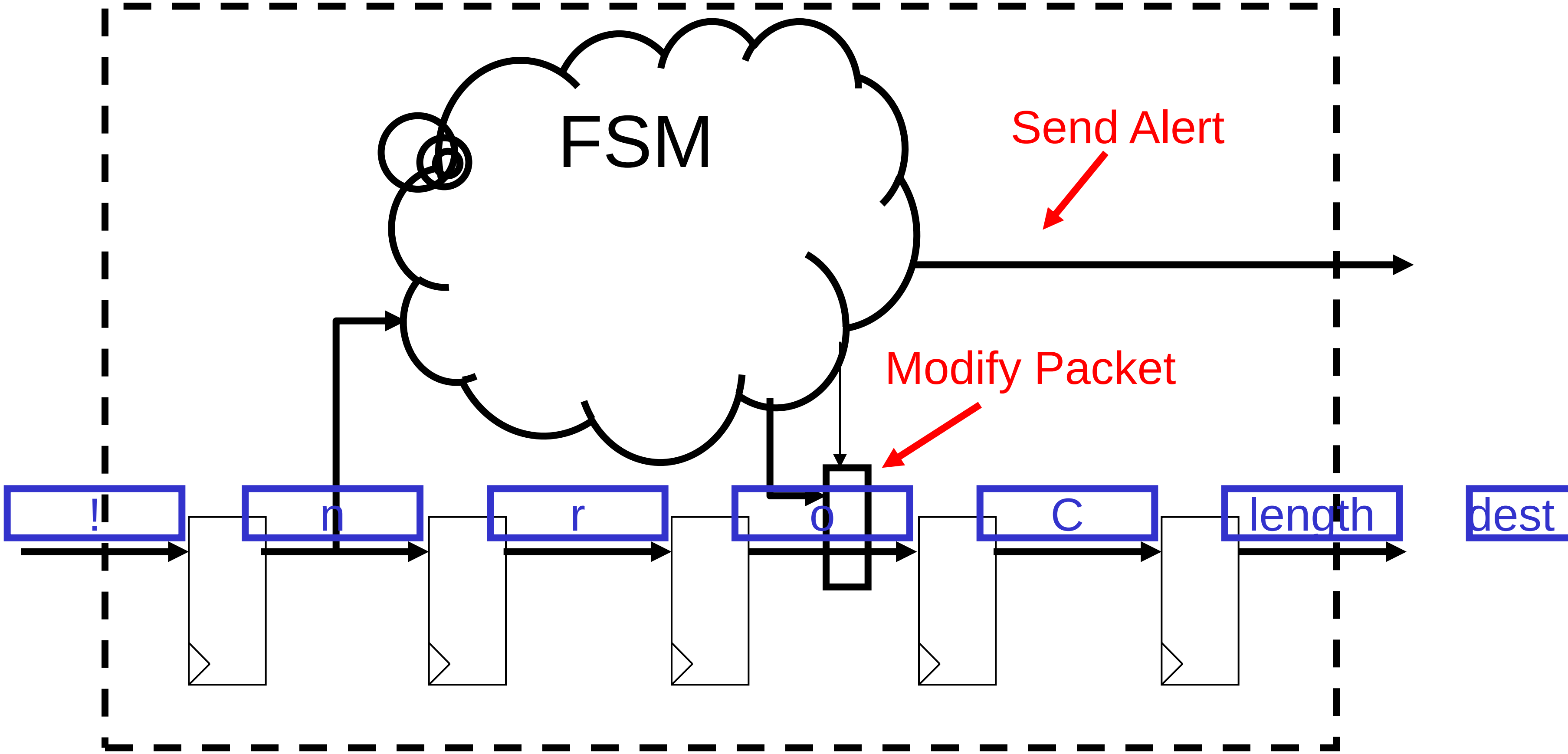
32-bits

Example: Network Processing Tasks

- Raise an alert signal when the pattern “corn!” is detected
- Return the number of times “corn!” is detected
  - Place count value as the last byte of the payload

Streaming Network application (MP1)

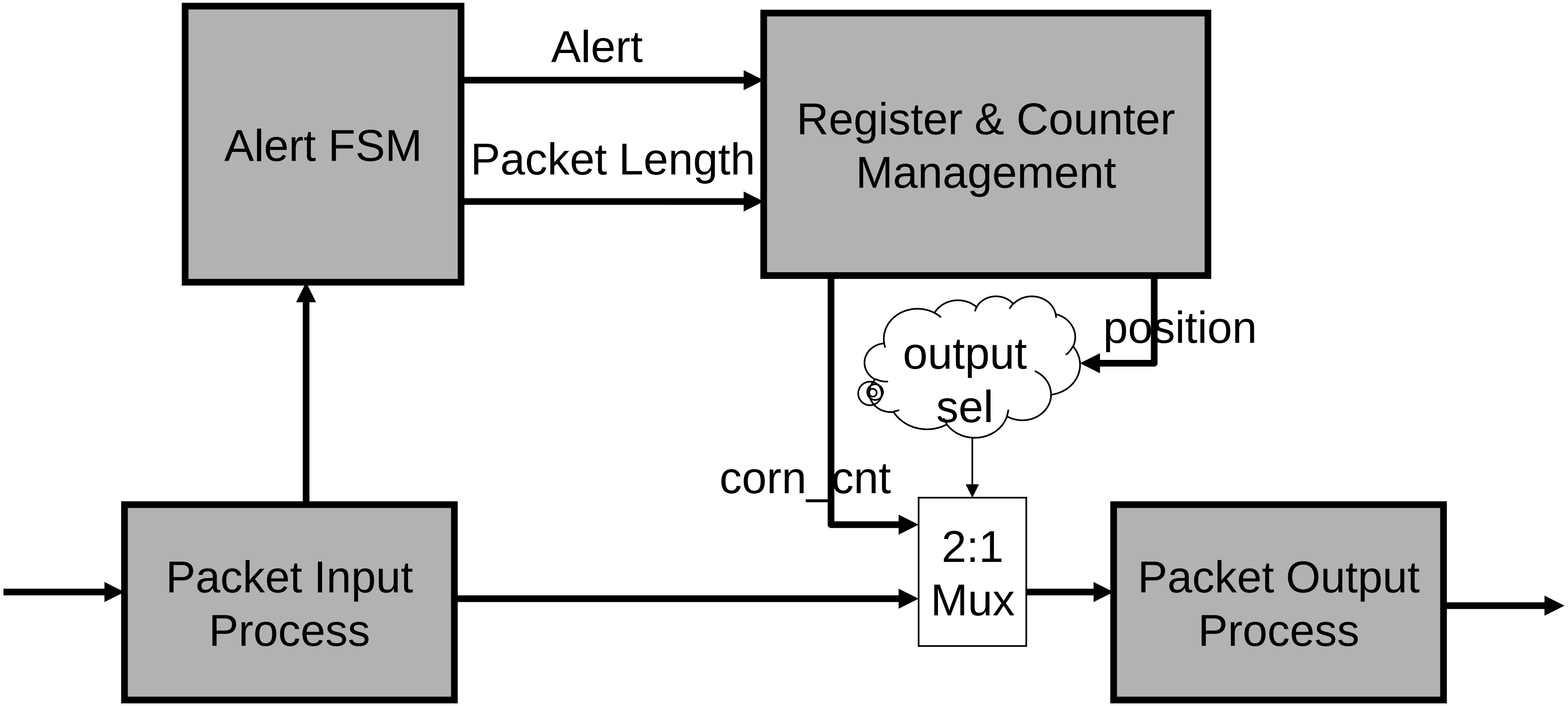
- Detect patterns in payload (e.g. “Corn!”)
- Place the number of detections in last byte of payload



Architecture

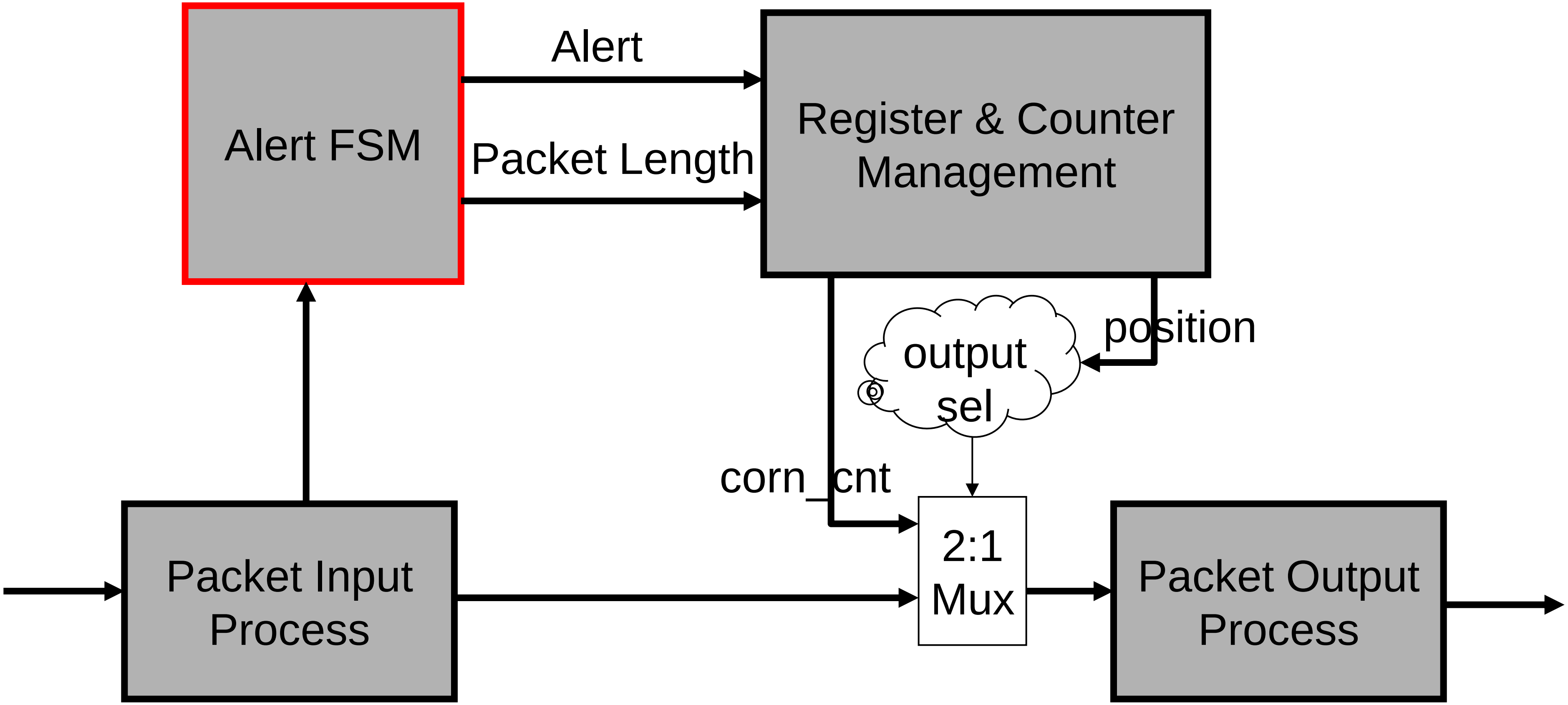
- Detect patterns in payload (e.g. "Corn!")
- Place the number of detections in last byte of payload

Draw out logic, and data flow!!!



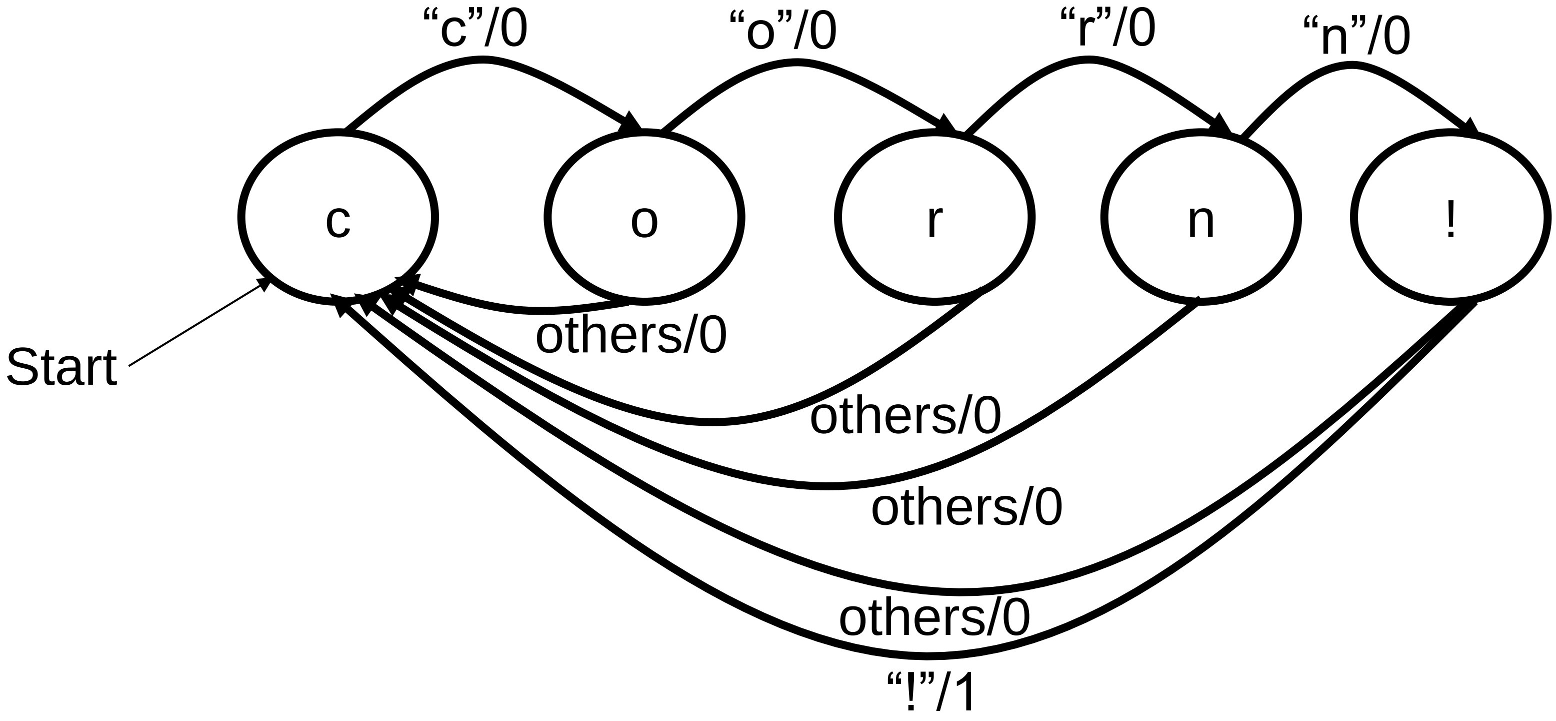
Architecture

- Detect patterns in payload (e.g. “Corn!”)
- Place the number of detections in last byte of payload



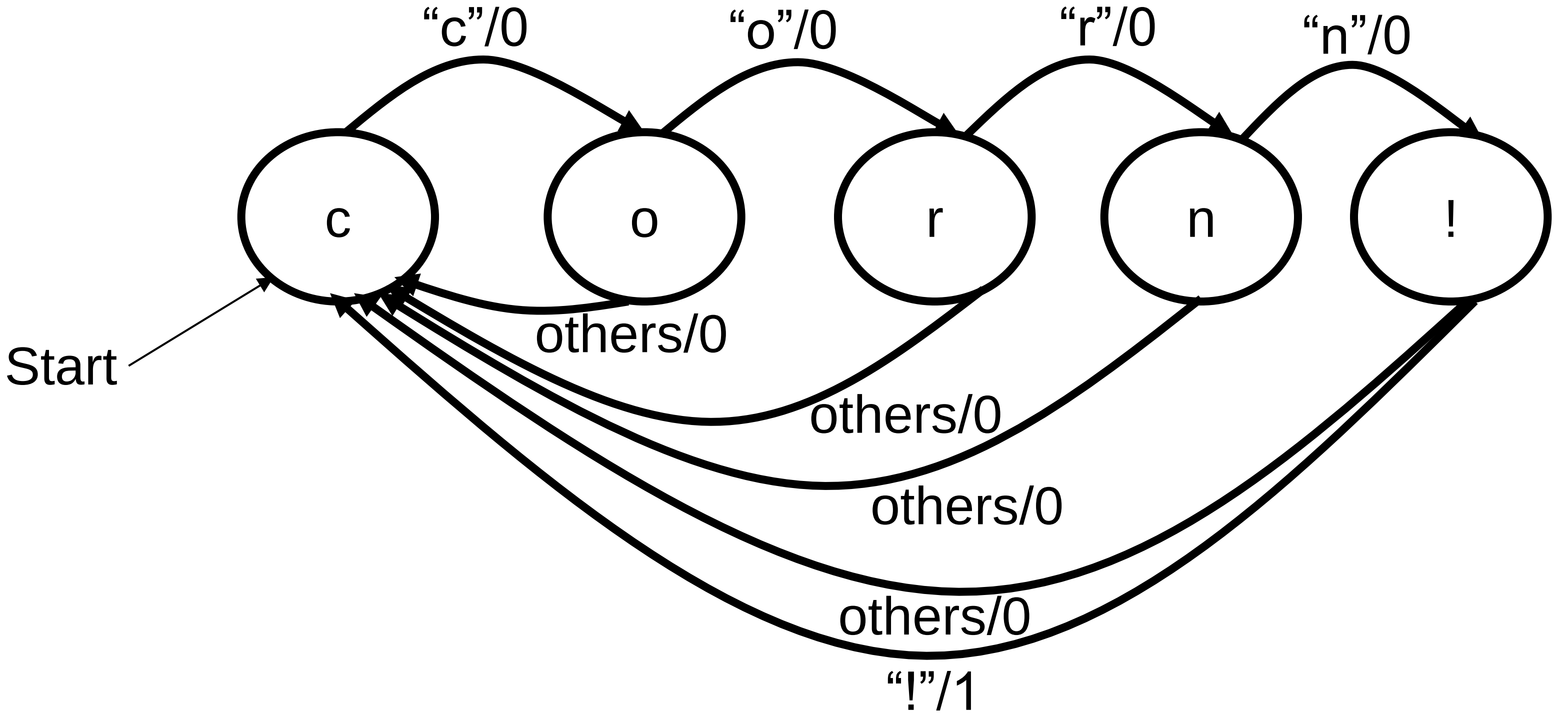
Alert FSM Design

- Alert signal when the pattern “corn!” is detected
  - Z = {Alert}



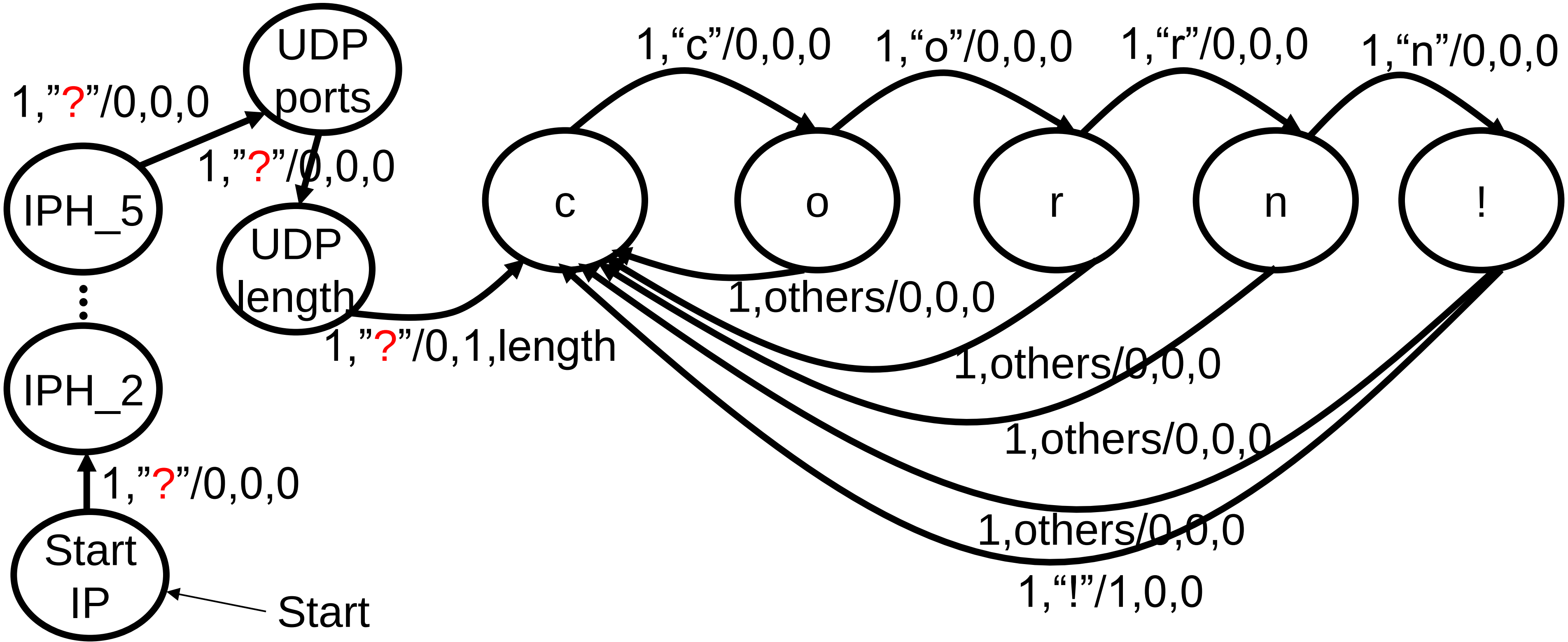
Alert FSM Design

- Alert signal when the pattern “corn!” is detected
- Output Packet’s Length
  - Z = {Alert, length_vld, pack_length}
  - X = {vld, input} : Note “?” is don’t care



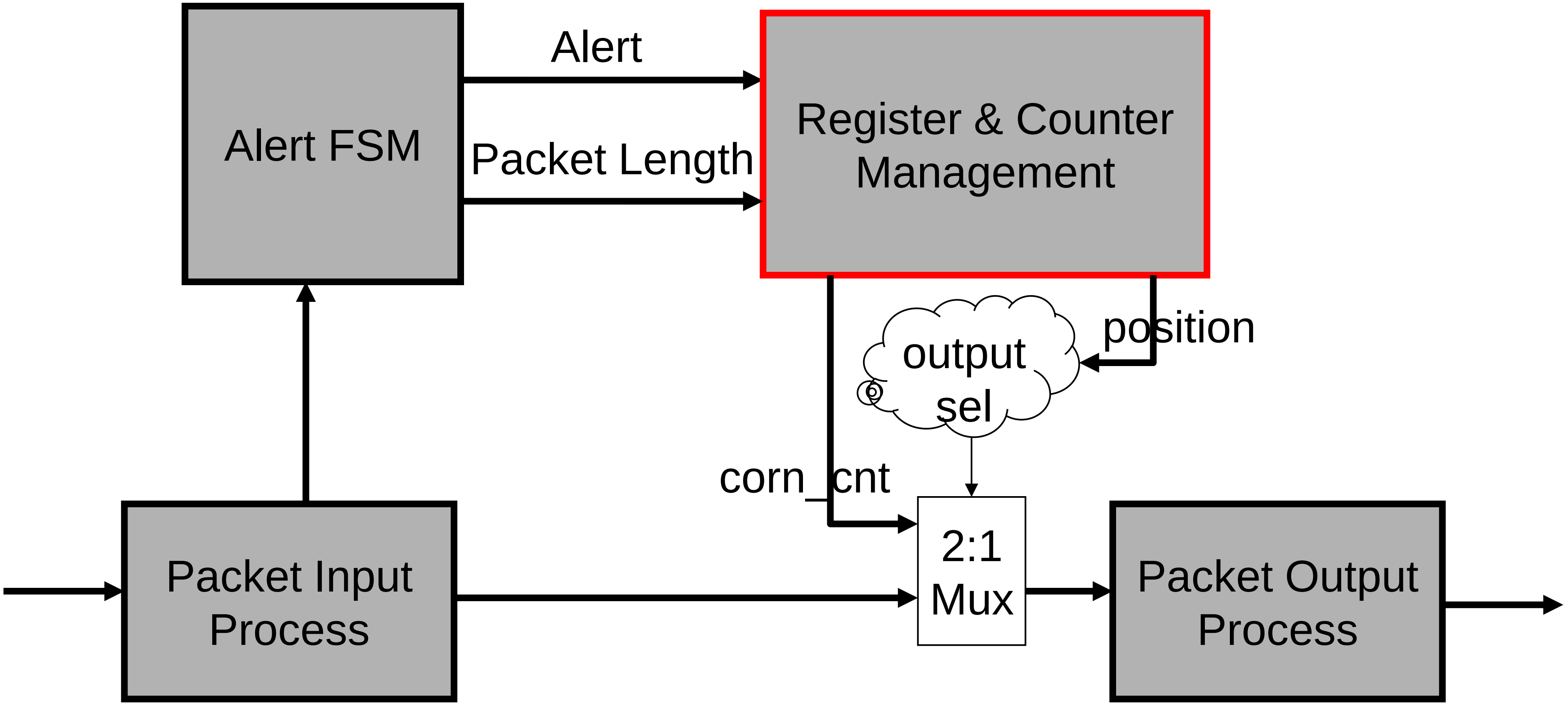
Alert FSM Design

- Alert signal when the pattern “corn!” is detected
- Output Packet’s Length
  - Z = {Alert,length_vld,pack_length}
  - X = {vld,input} : Note “?” is don’t care



Architecture

- Detect patterns in payload (e.g. “Corn!”)
- Place the number of detections in last byte of payload

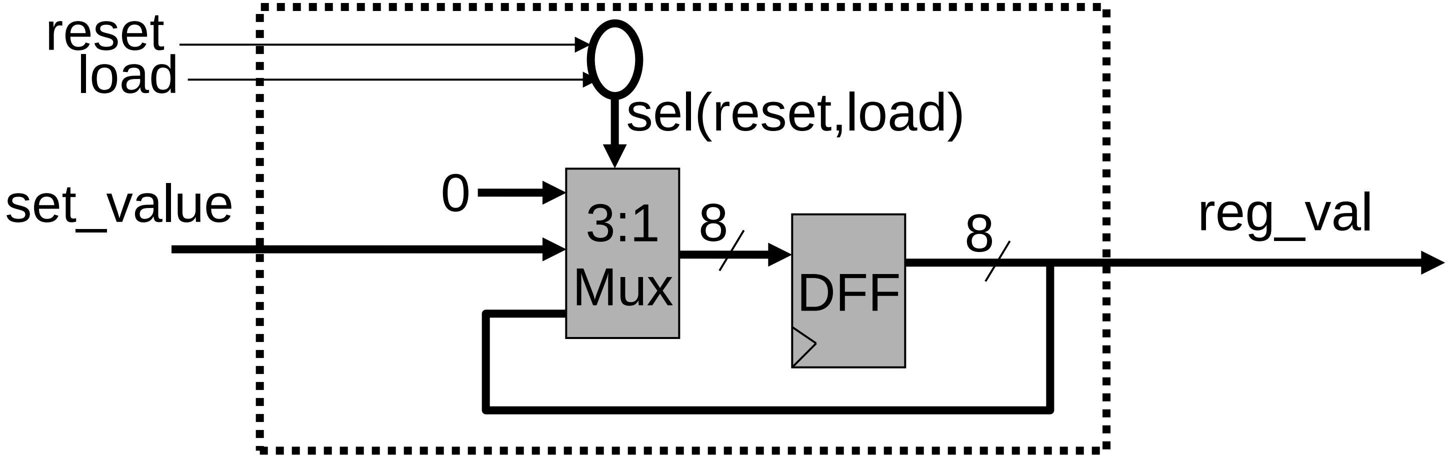


Register & Counter Manager

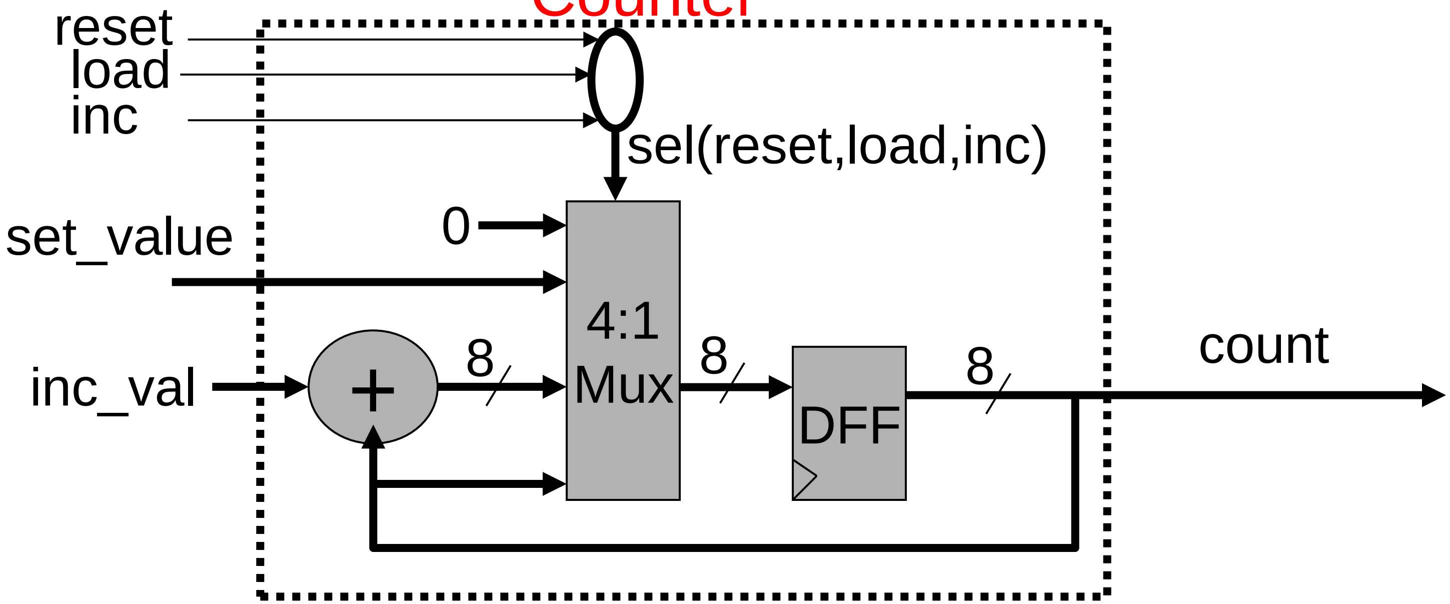
- Register & Counter Components
- Design of Manager

Register and Counter Components

Register



Counter

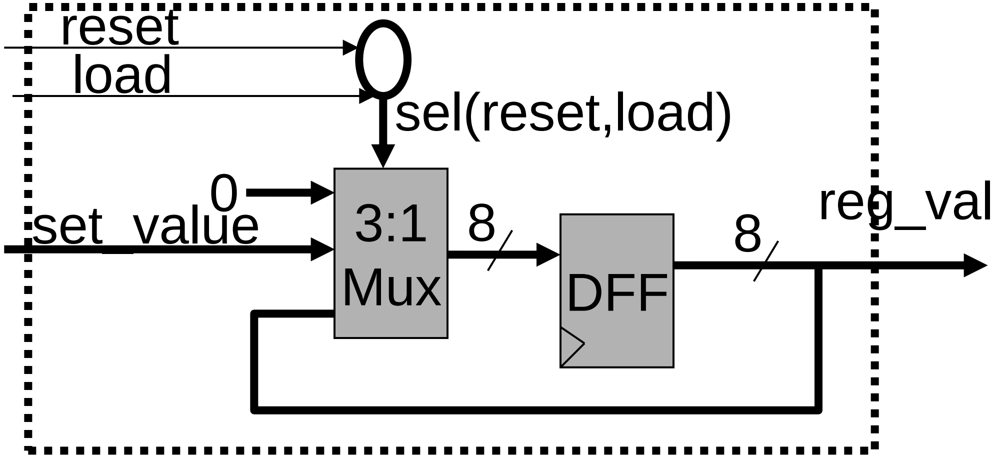


Practice: Write VHDL(process for each)

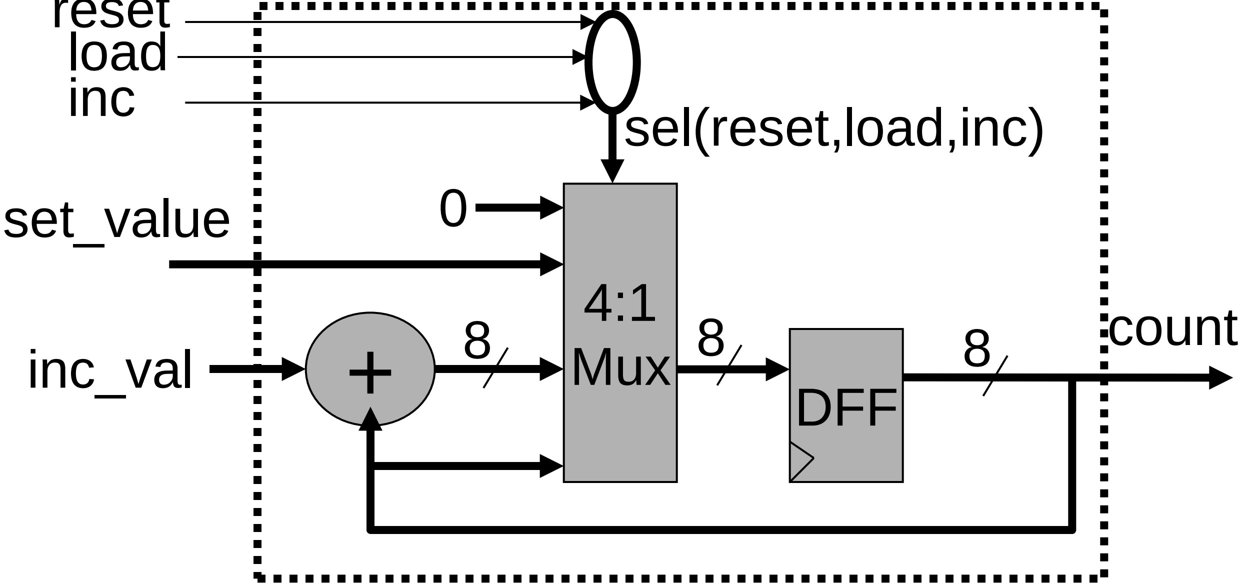
```
Name : process(clk)
begin
  if(clk'event and clk='1') then
    logic here
  end if;
end process Name
```

```
CASE sel is
WHEN "00" | "11"=>
   out_1 <= in_0;
WHEN "01" =>
  out_1 <= in_1;
       ⋮
WHEN OTHERS =>
  out_1 <= in_0;
END CASE;
```

Register

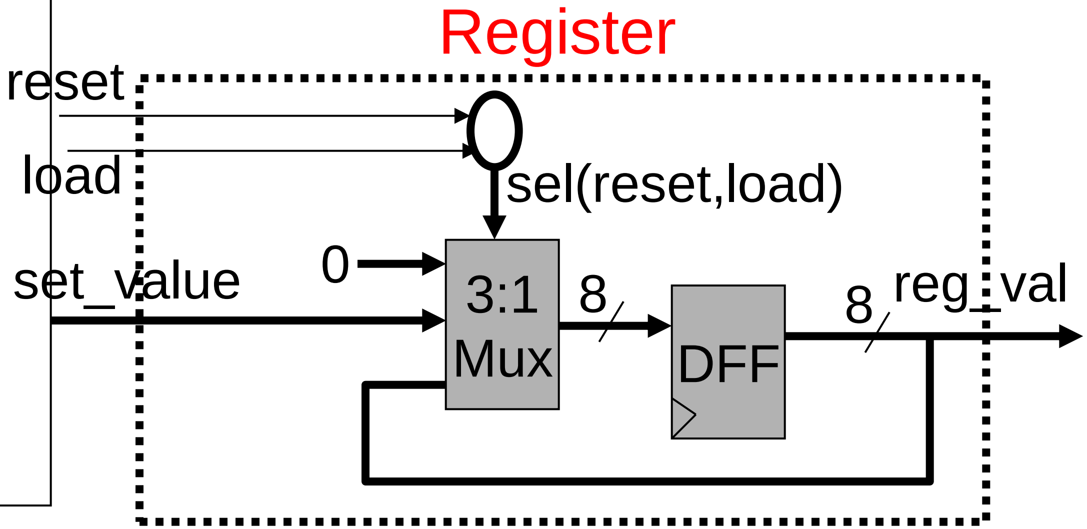


Counter



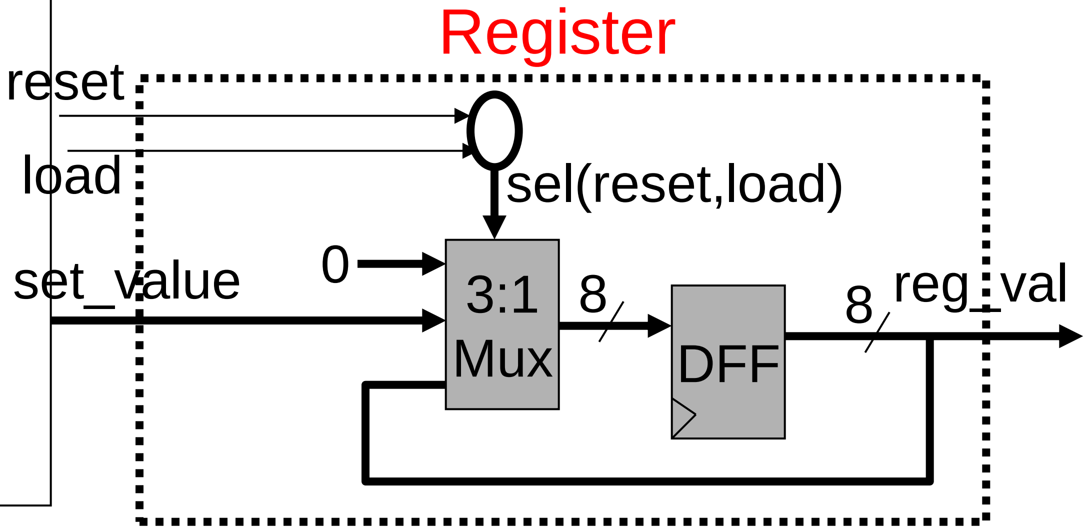
Register VHDL

```
Name : process(clk)
begin
  if(clk'event and clk='1') then
    CASE <signal> is
    WHEN <opt> | <opt> =>



    WHEN <opt> | <opt> =>



    WHEN OTHERS =>



    END CASE;
  end if;
end process Name
```



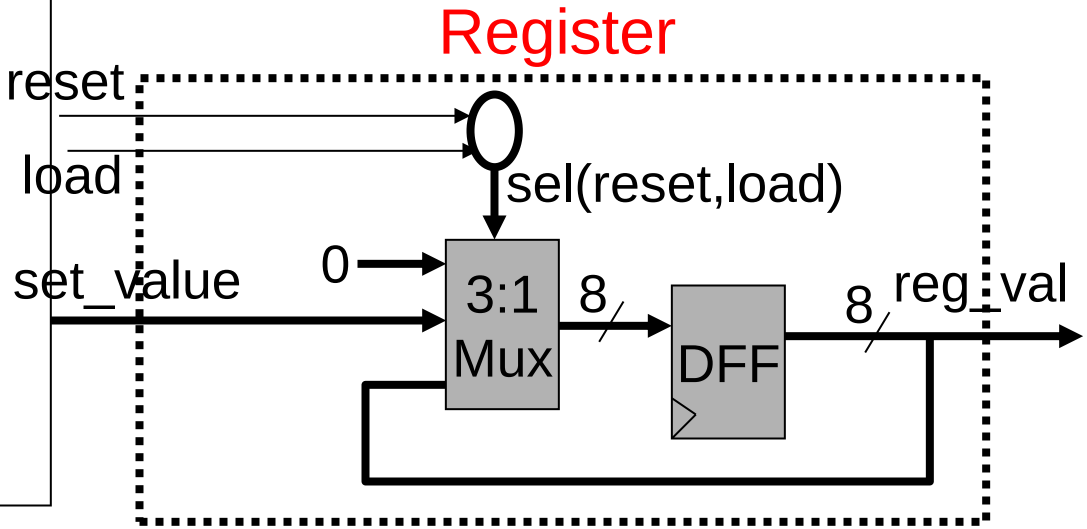
Register VHDL

```
Name : process(clk)
begin
  if(clk’event and clk=‘1’) then
    CASE reset&load is
    WHEN “10” | “11” =>
      reg_val <= 0;
    WHEN “01” =>
      reg_val <= set_value;
    WHEN OTHERS =>
      reg_val <= reg_val;
    END CASE;
  end if;
end process Name
```



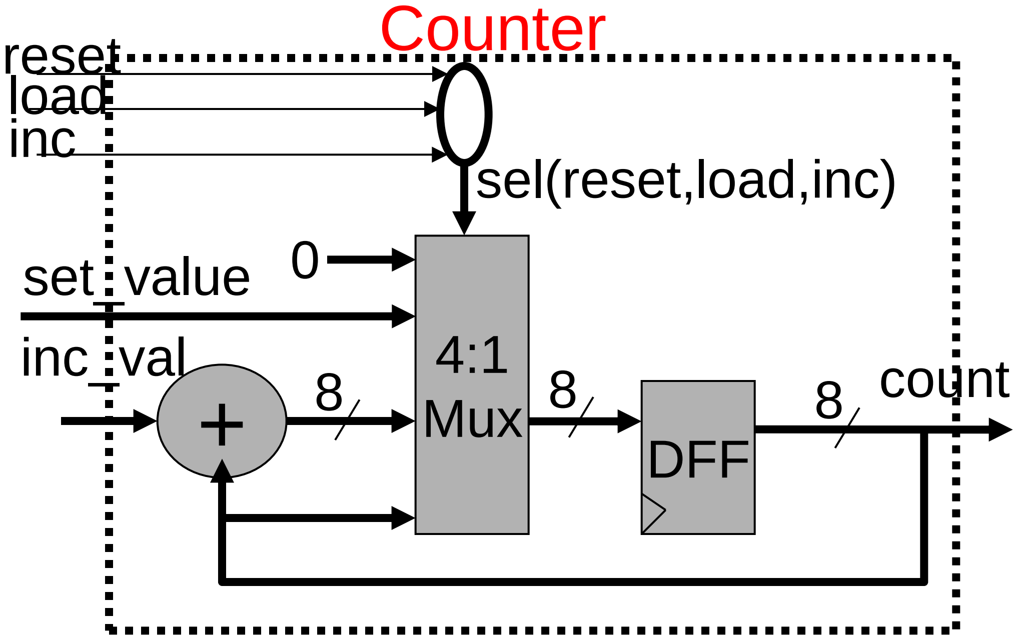
Register VHDL

```
Name : process(clk)
begin
  if(clk’event and clk=‘1’) then
    CASE sel is
    WHEN “10” | “11” =>
      reg_val <= 0;
    WHEN “01” =>
      reg_val <= set_value;
    WHEN OTHERS =>
      reg_val <= reg_val;
    END CASE;
  end if;
end process Name

sel <= reset&load;
```



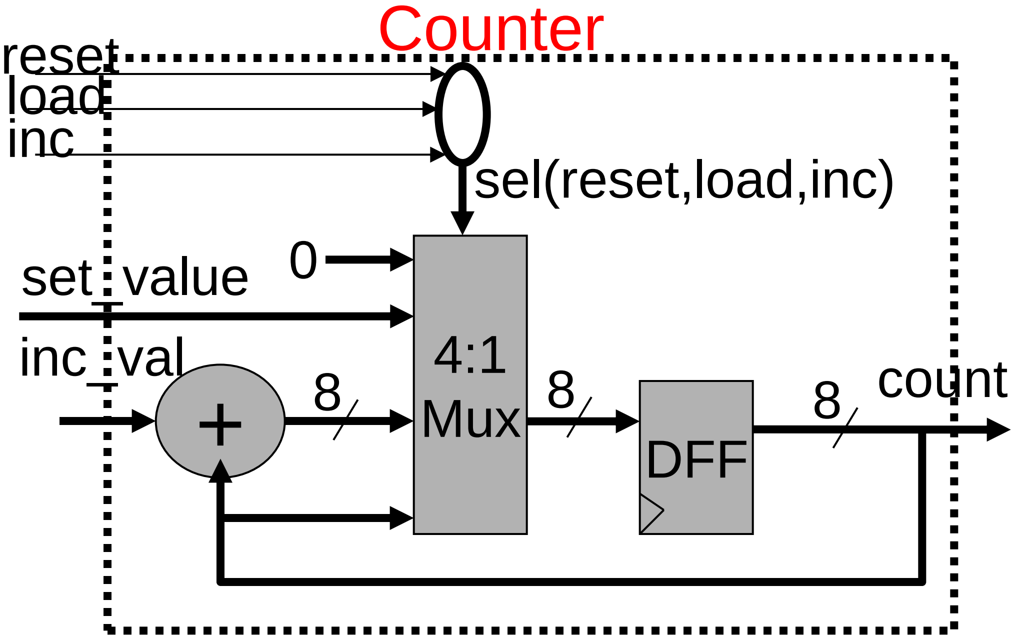
Counter VHDL

```
Name : process(clk)
begin
  if(clk’event and clk=‘1’) then
    CASE <signal> is
    WHEN <opt> | <opt> =>



    WHEN <opt> | <opt> =>



    WHEN OTHERS =>



    END CASE;
  end if;
end process Name
```



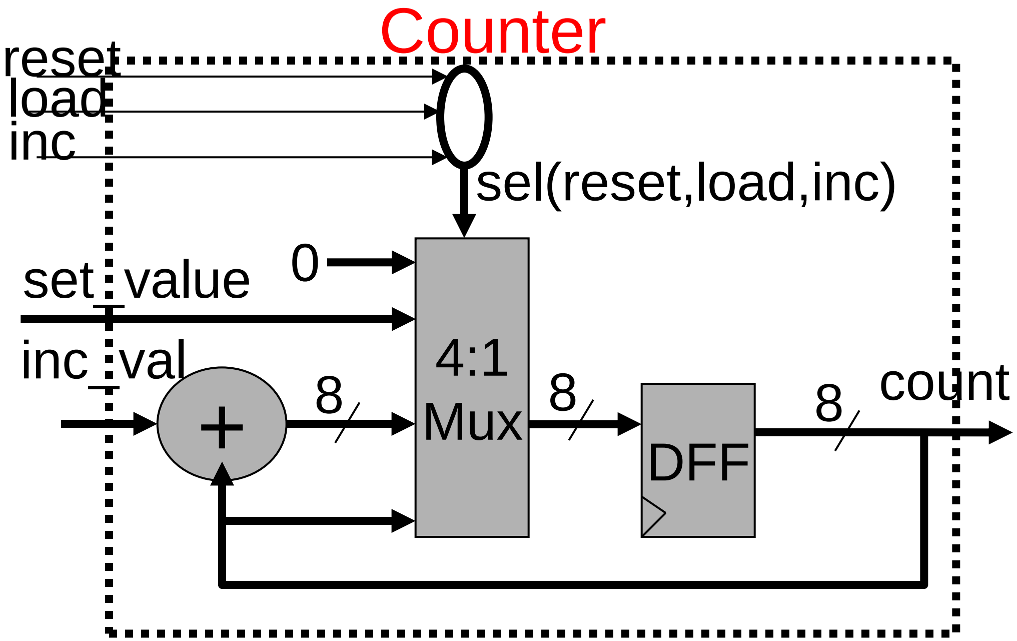
Counter VHDL

```
Name : process(clk)
begin
  if(clk’event and clk=‘1’) then
    CASE reset&load&inc is
    WHEN “100” | “101” |
                 “110”| “111”  =>
      count <= 0;
    WHEN “010” | “011” =>
      count <= set_value;
    WHEN “001” =>
      count <= count + inc_val;
    WHEN OTHERS =>
      count <= count;
    END CASE;
  end if;
end process Name
```



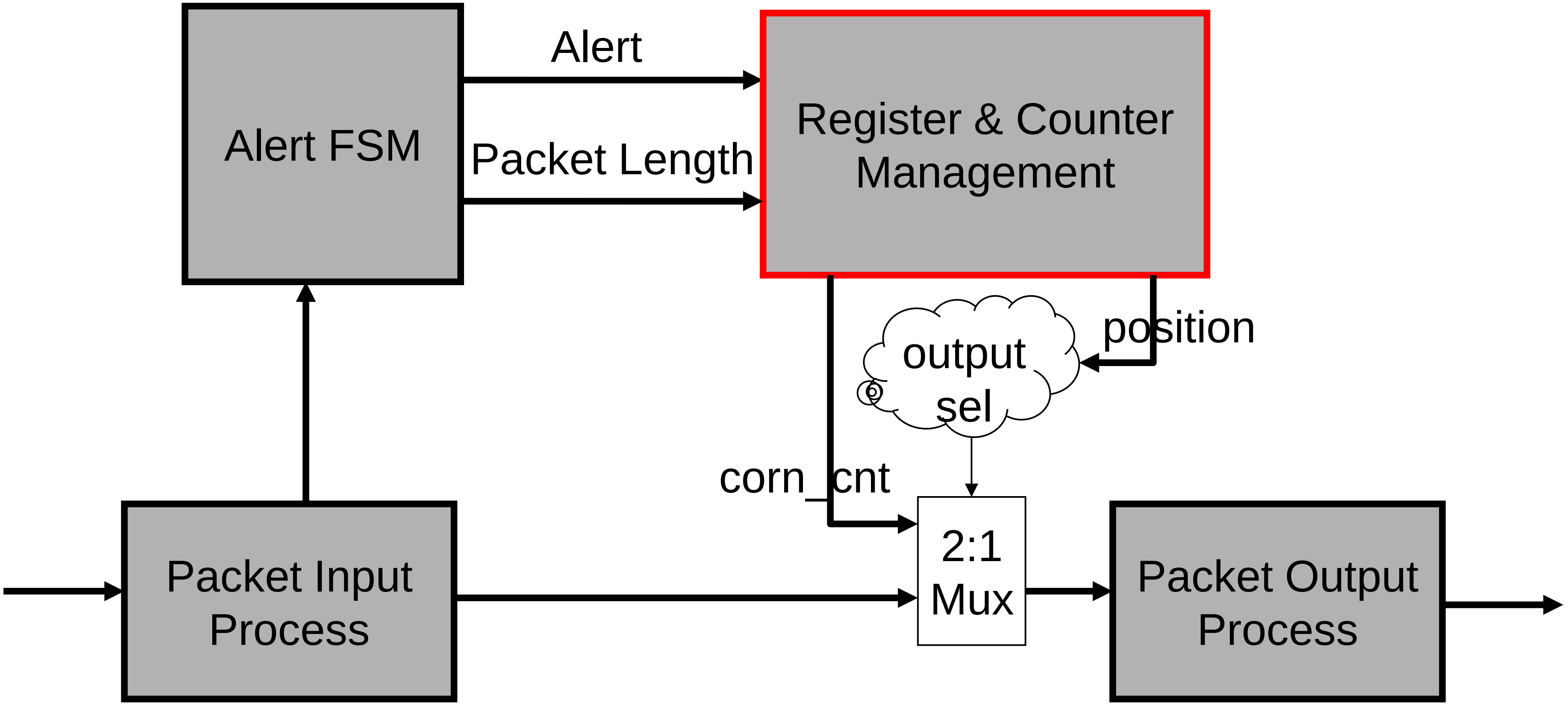
Counter VHDL

```
Name : process(clk)
begin
  if(clk'event and clk='1') then
    CASE sel is
    WHEN "100" | "101" |
                 "110"| "111"  =>
      count <= 0;
    WHEN "010" | "011" =>
      count <= set_value;
    WHEN "001" =>
      count <= count + inc_val;
    WHEN OTHERS =>
      count <= count;
    END CASE;
  end if;
end process Name
sel <= reset&load&inc;
```



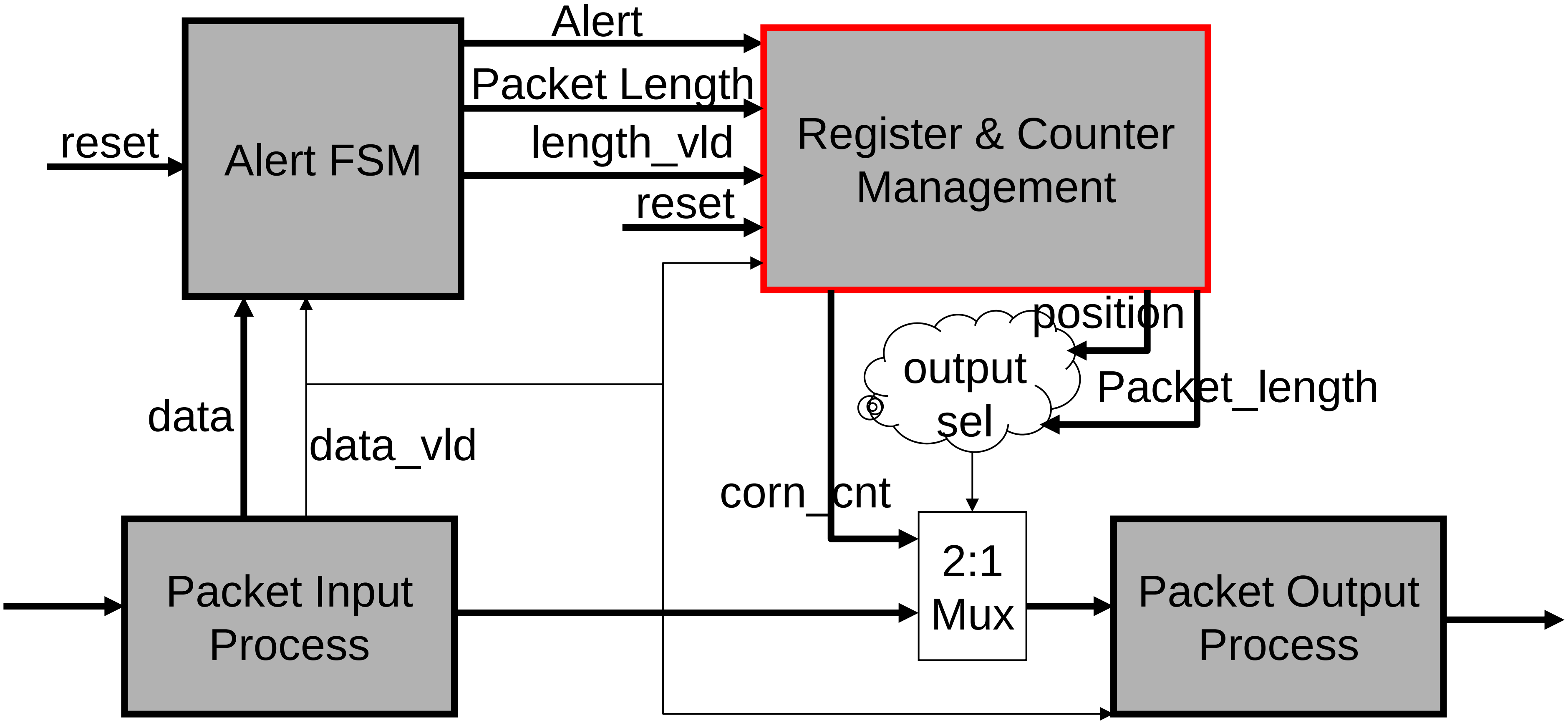
Architecture

- Detect patterns in payload (e.g. “Corn!”)
- Place the number of detections in last byte of payload

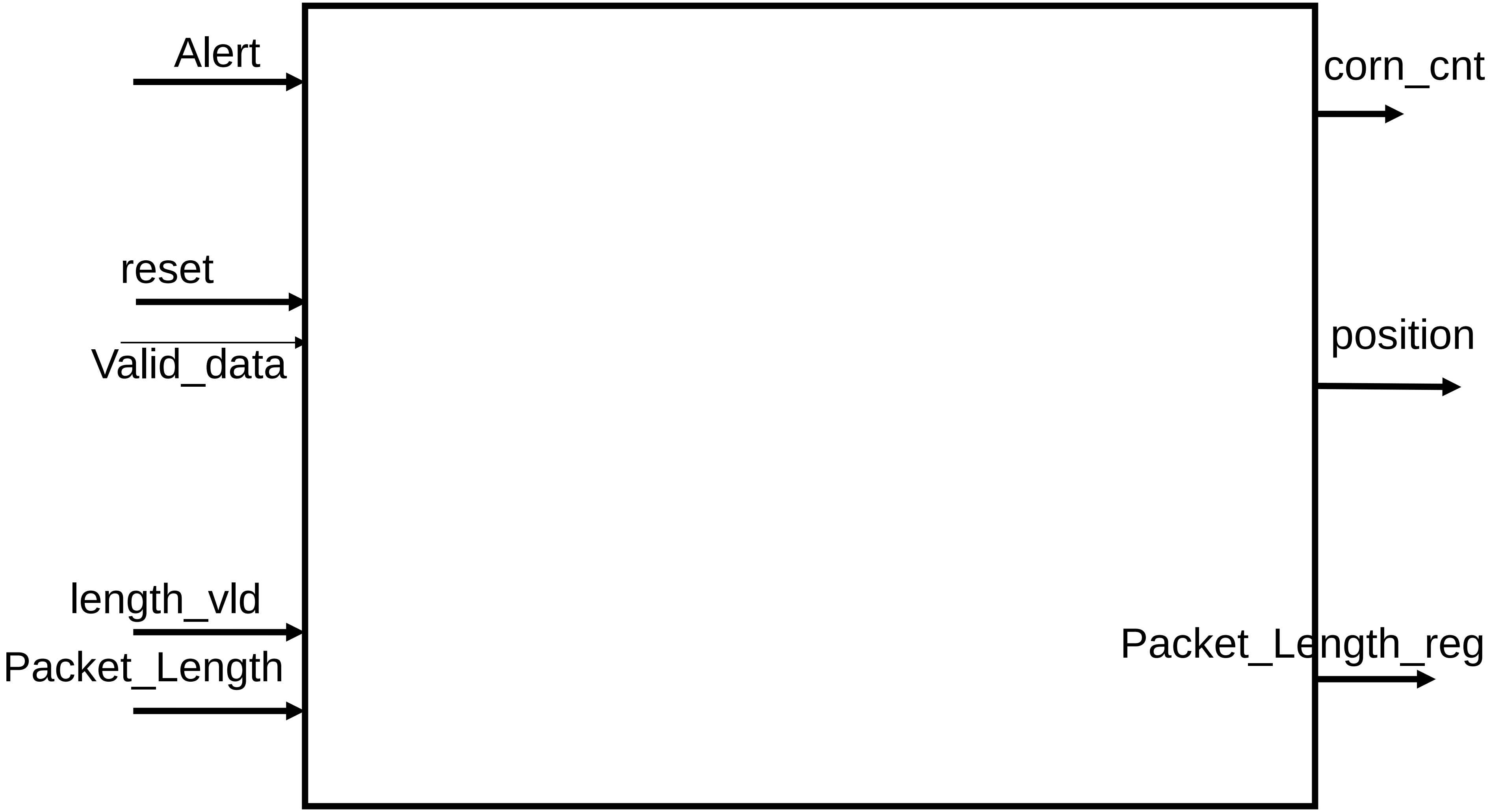


Architecture

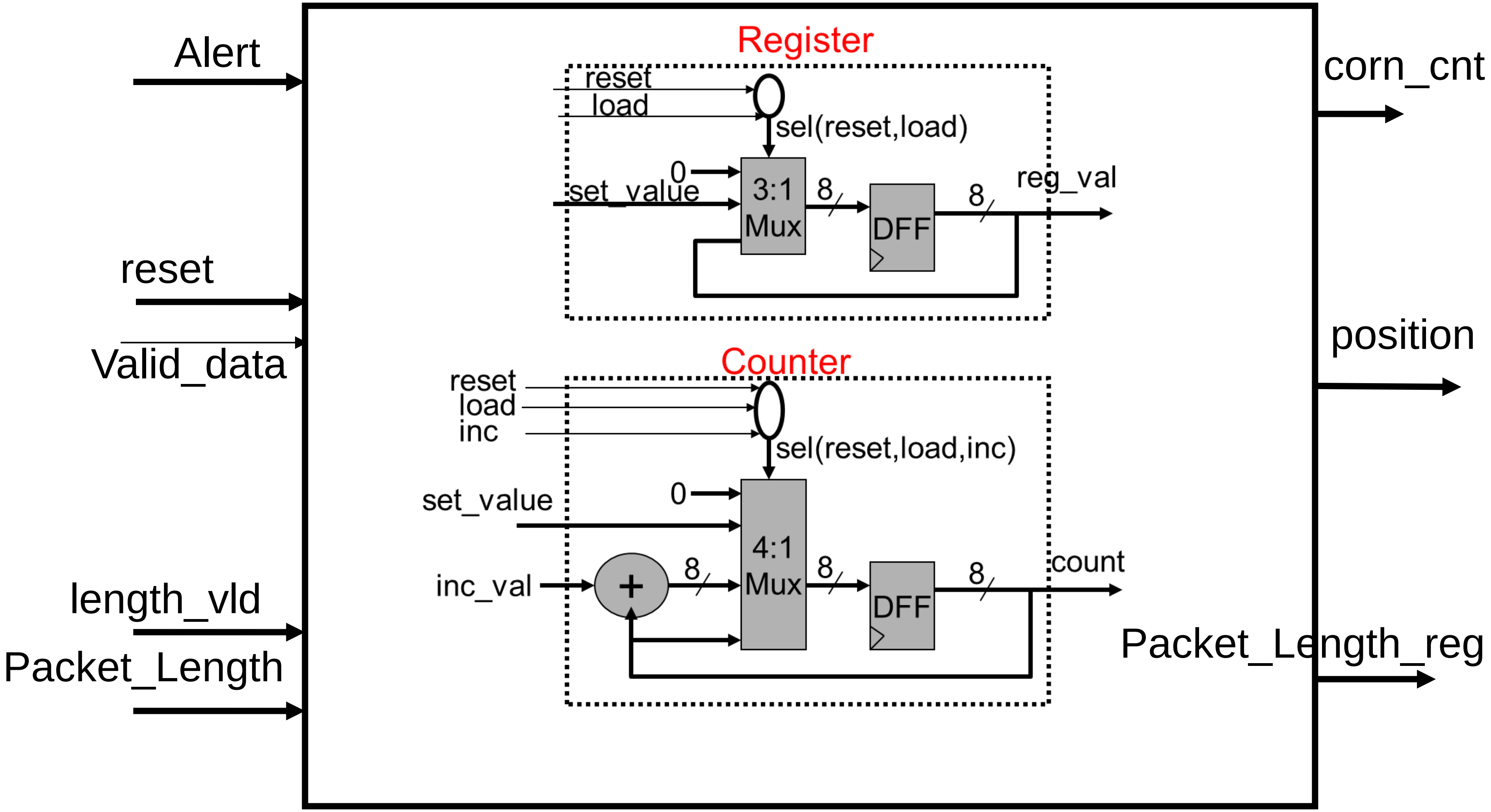
- Detect patterns in payload (e.g. “Corn!”)
- Place the number of detections in last byte of payload



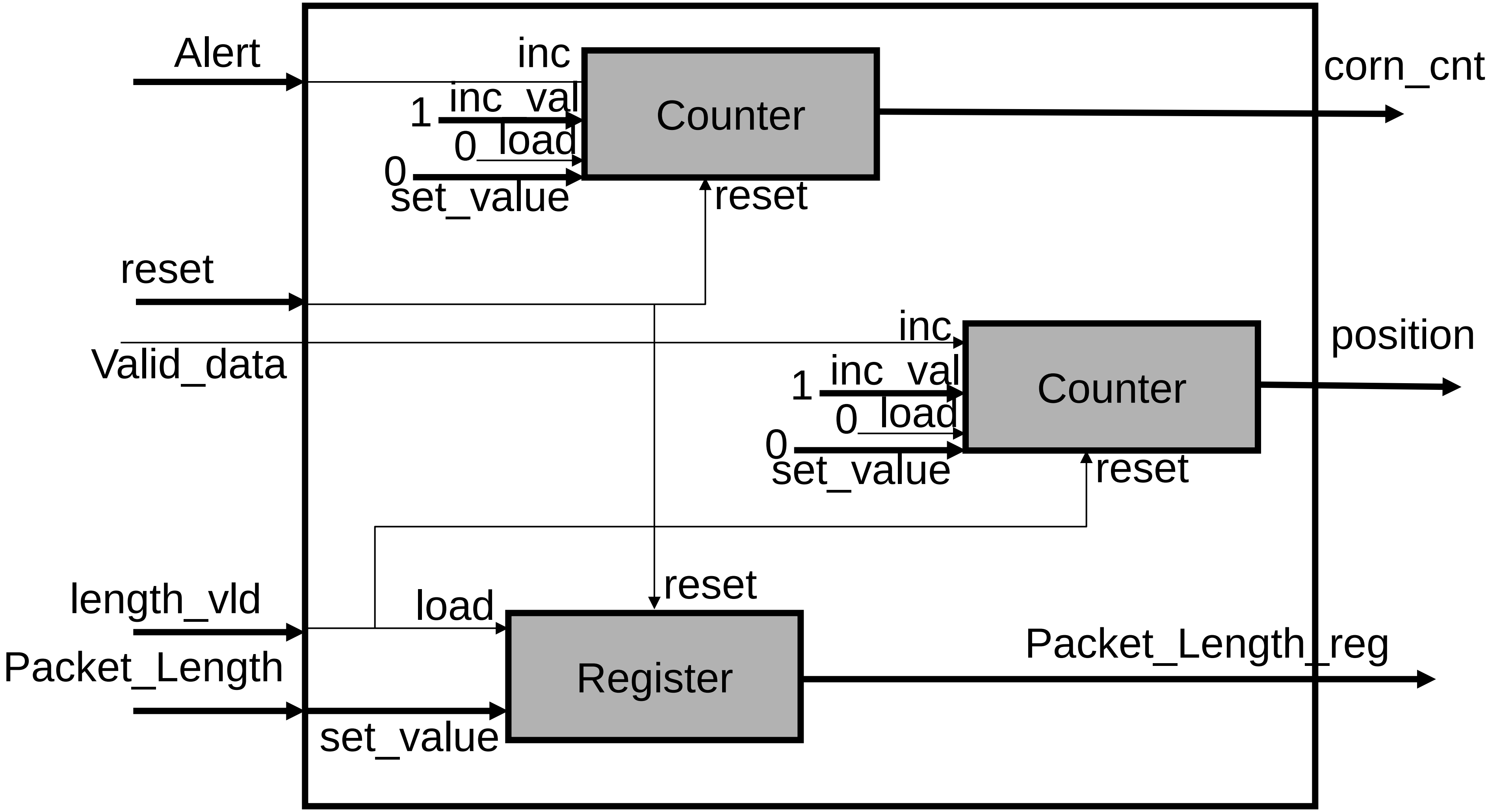
Register and Counter Manger



Register and Counter Manger

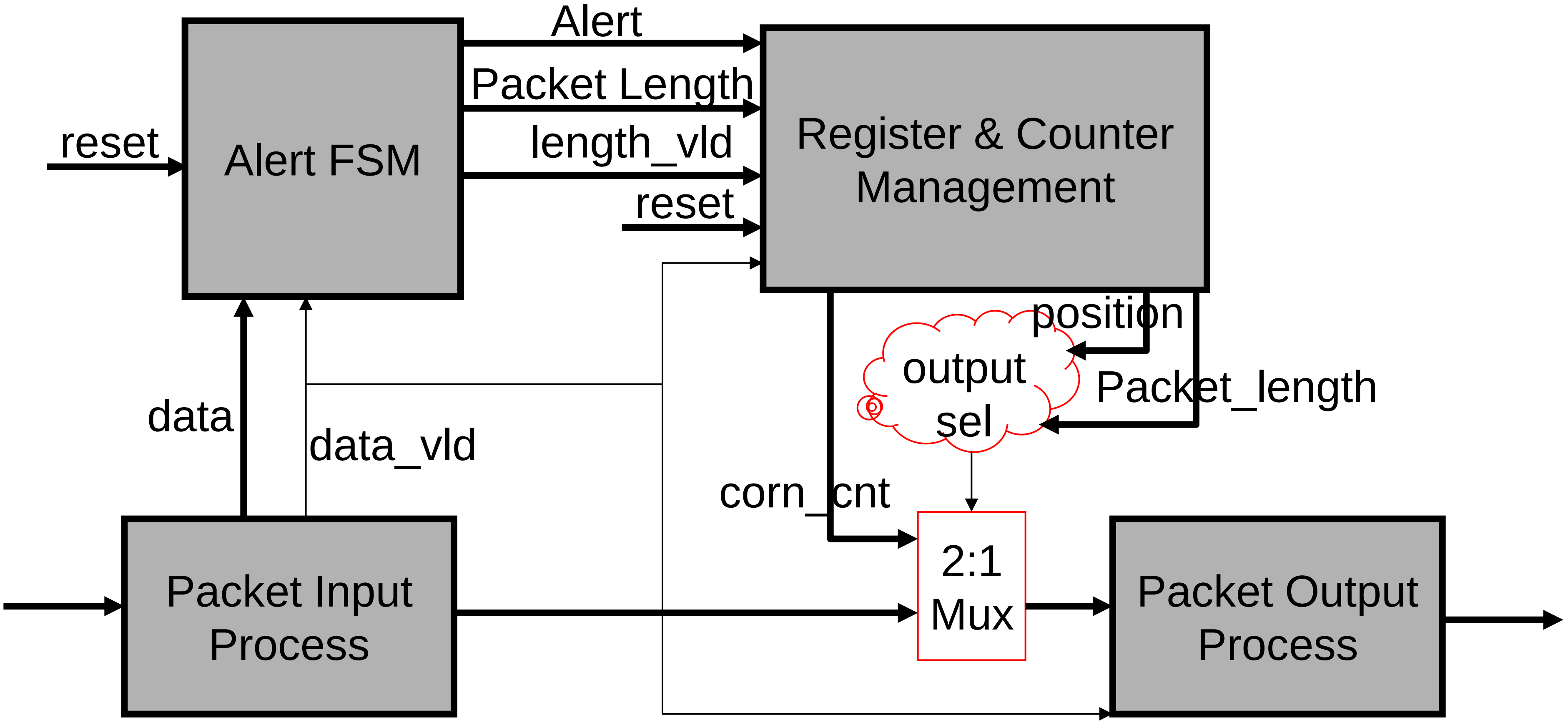


Register and Counter Manger

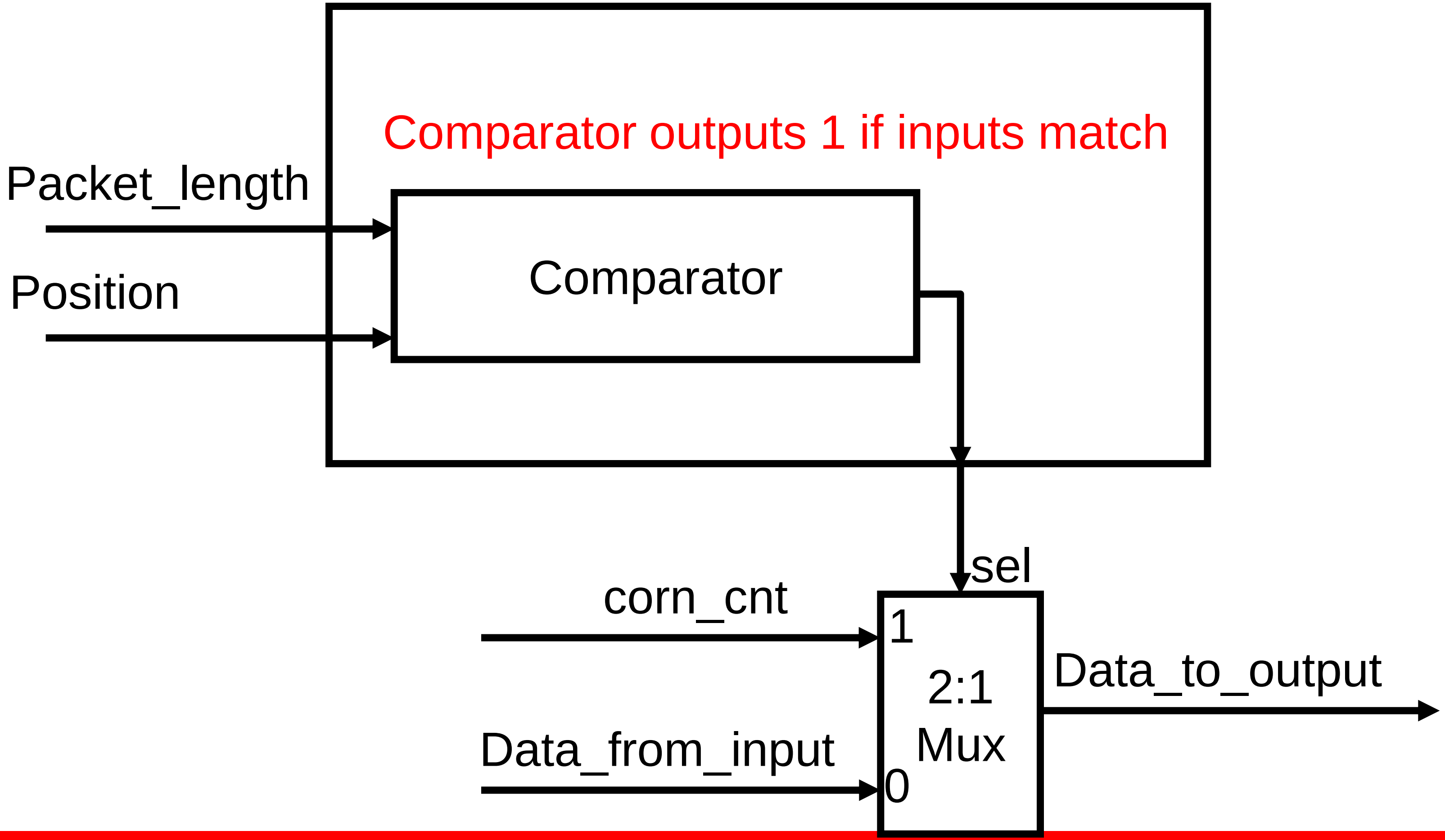


Architecture

- Detect patterns in payload (e.g. “Corn!”)
- Place the number of detections in last byte of payload



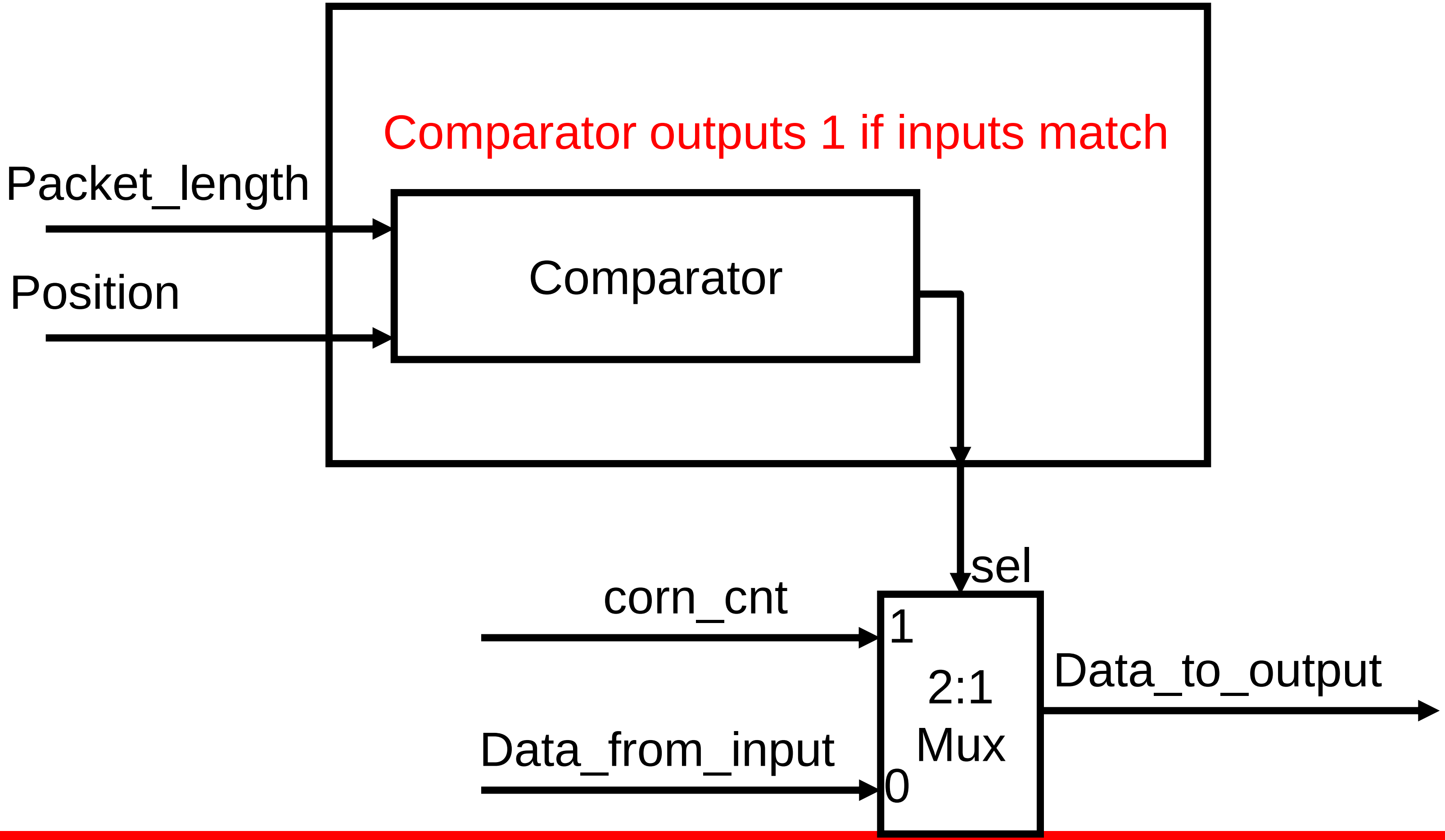
Output sel


Comparator outputs 1 if inputs match
Packet_length
Position
Comparator
sel
corn_cnt
1
2:1
Mux
Data_to_output
Data_from_input
0

Output sel: VHDL

NOT in a process!

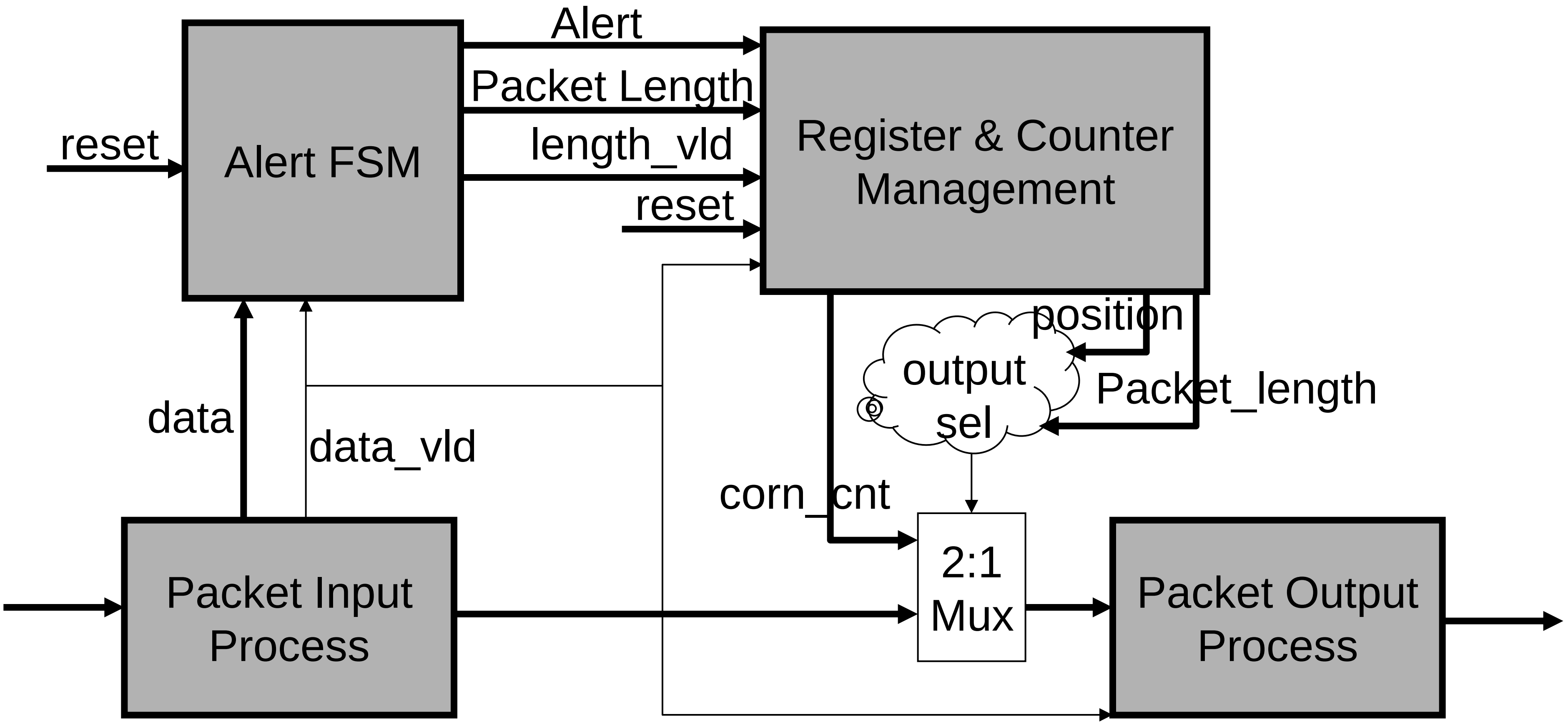
```
Data_to_output <= corn_cnt when (Packet_length = Position)
                       else Data_from_input
```



Comparator outputs 1 if inputs match

Architecture

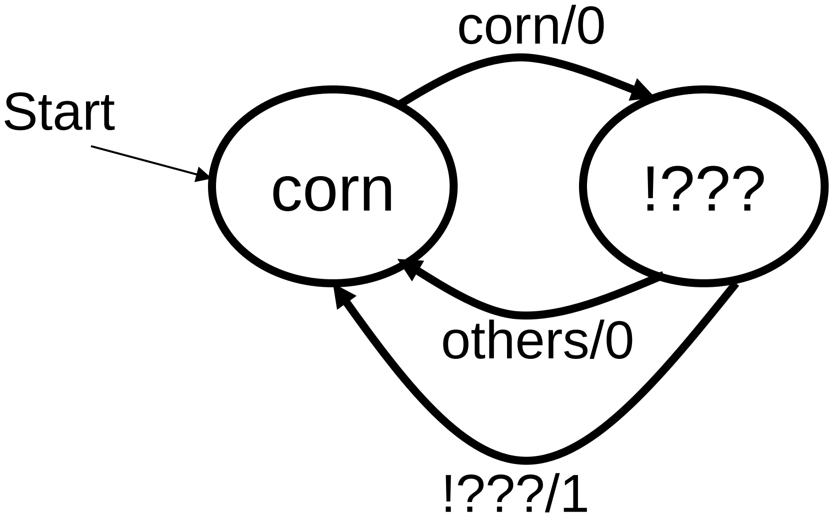
- Detect patterns in payload (e.g. “Corn!”)
- Place the number of detections in last byte of payload



Multiple Characters per Clock

- Network input stream typically 32-bit words
  - 4 8-bit characters per word.
- corn! Example

c	o	r	n	Word 1
!	?	?	?	Word 2

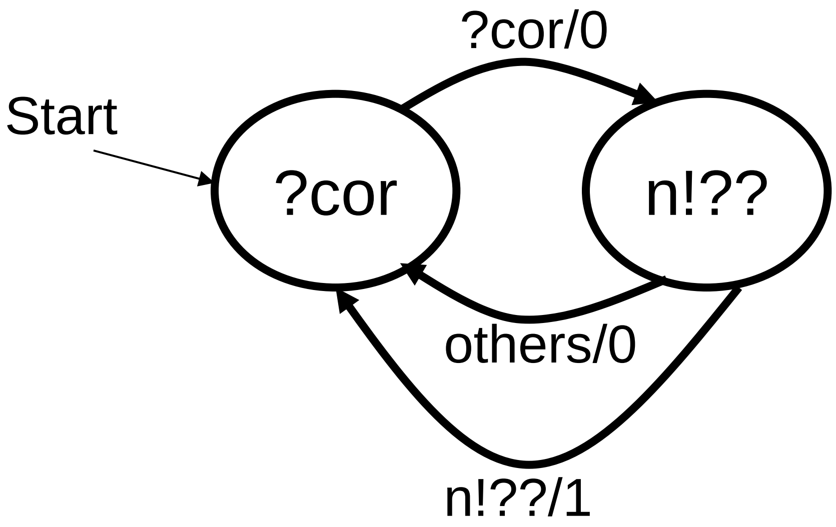


Corn! on a word boundary

Multiple Characters per Clock

- Network input stream typically 32-bit words
  - 4 8-bit characters per word.
- corn! Example

?	c	o	r	Word 1
n	!	?	?	Word 2

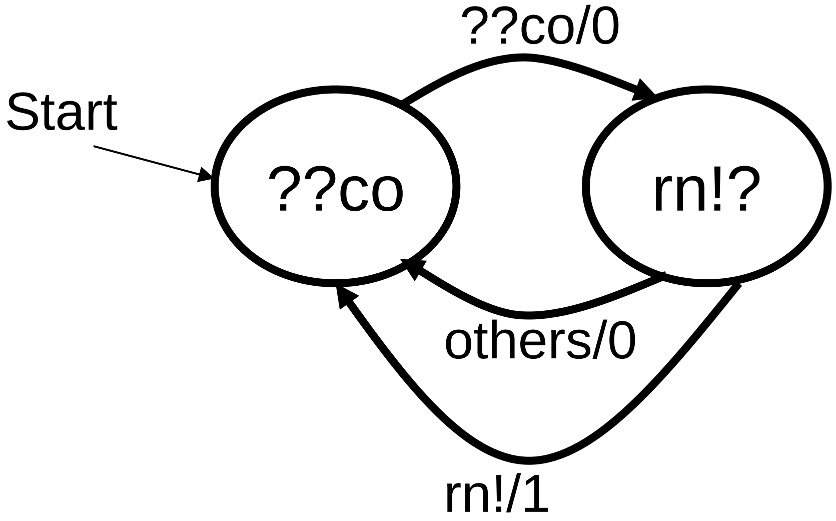


Corn! offset by 1 byte

Multiple Characters per Clock

- Network input stream typically 32-bit words
  - 4 8-bit characters per word.
- corn! Example

?	?	c	o	Word 1
r	n	!	?	Word 2

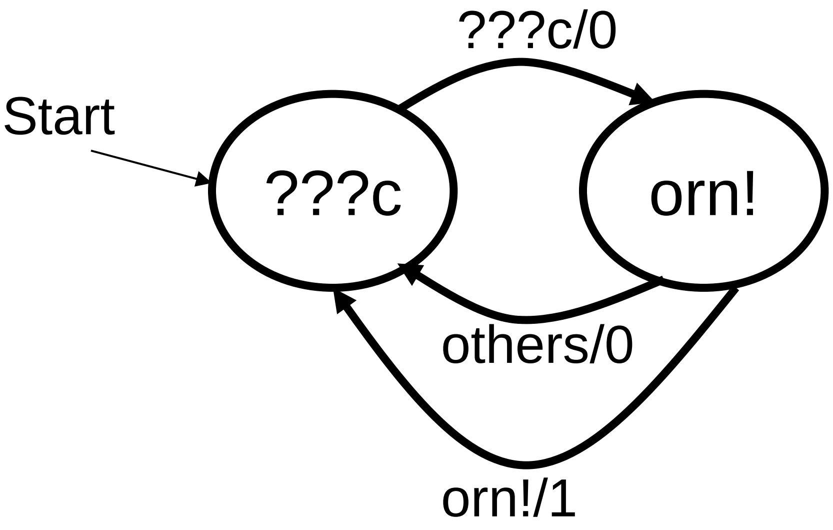


Corn! offset by 2 bytes

Multiple Characters per Clock

- Network input stream typically 32-bit words
  - 4 8-bit characters per word.
- corn! Example

?	?	?	c	Word 1
o	r	n	!	Word 2

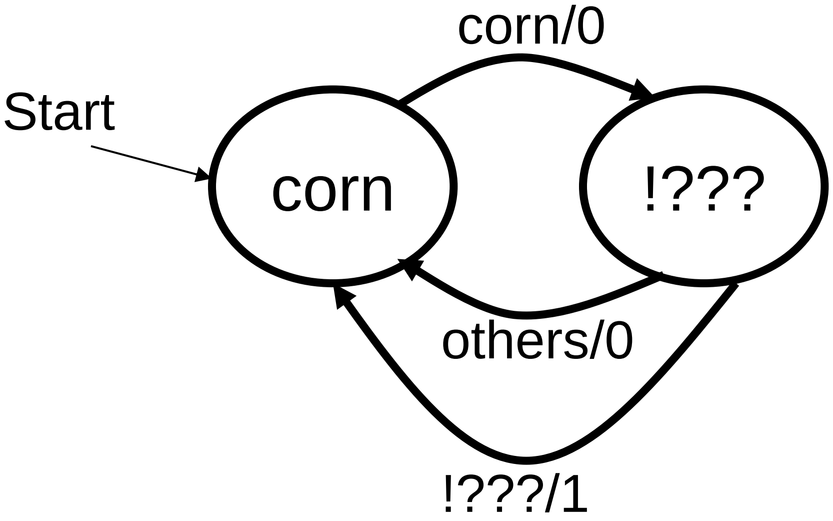


Corn! offset by 3 bytes

Multiple Characters per Clock

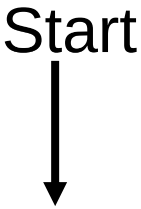
- Network input stream typically 32-bit words
  - 4 8-bit characters per word.
- corn! Example

c	o	r	n	Word 2
!	?	?	?	Word 3



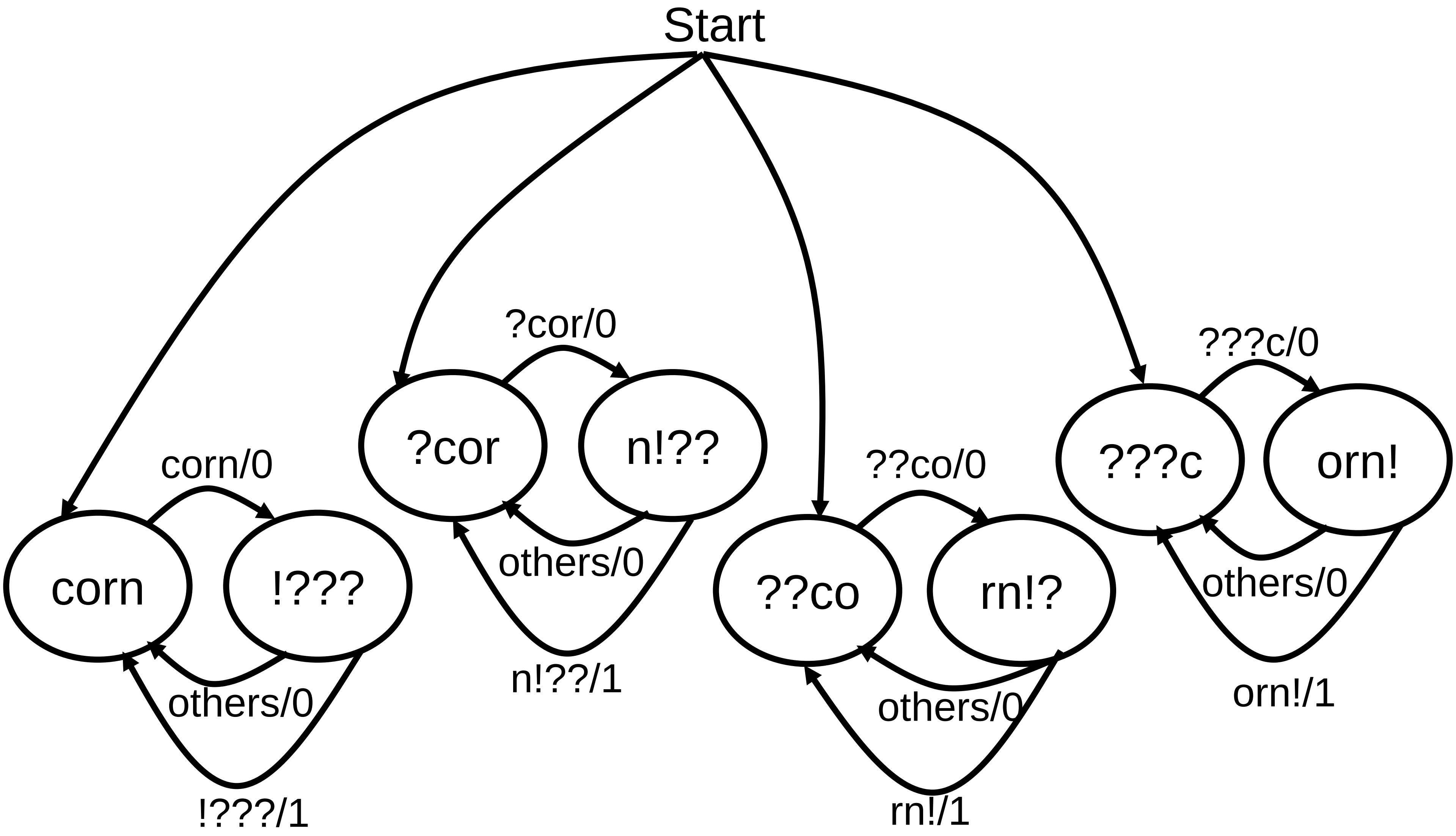
Corn! offset by 4 bytes

Modify Alert FSM for Multiple characters

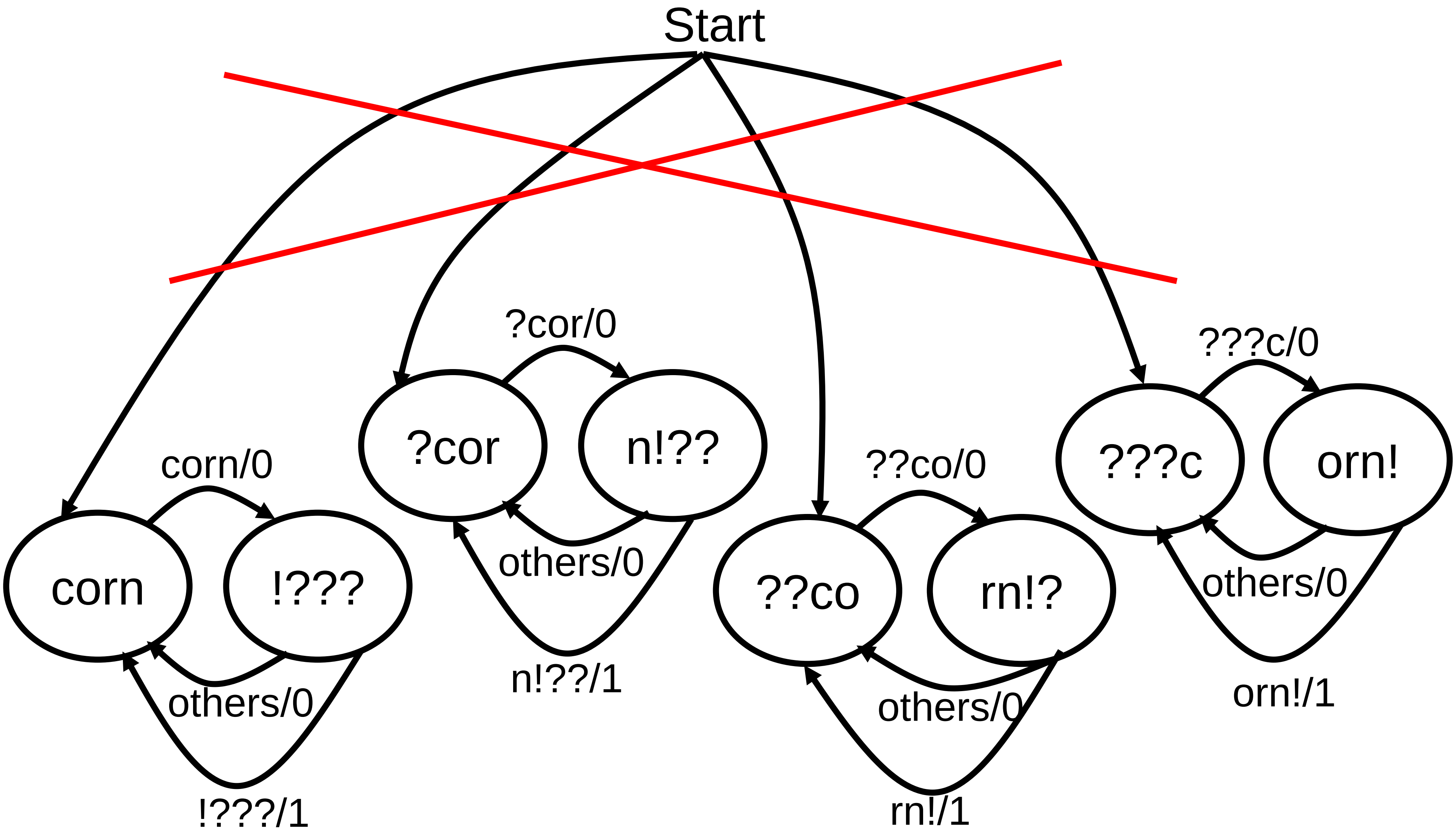


Start

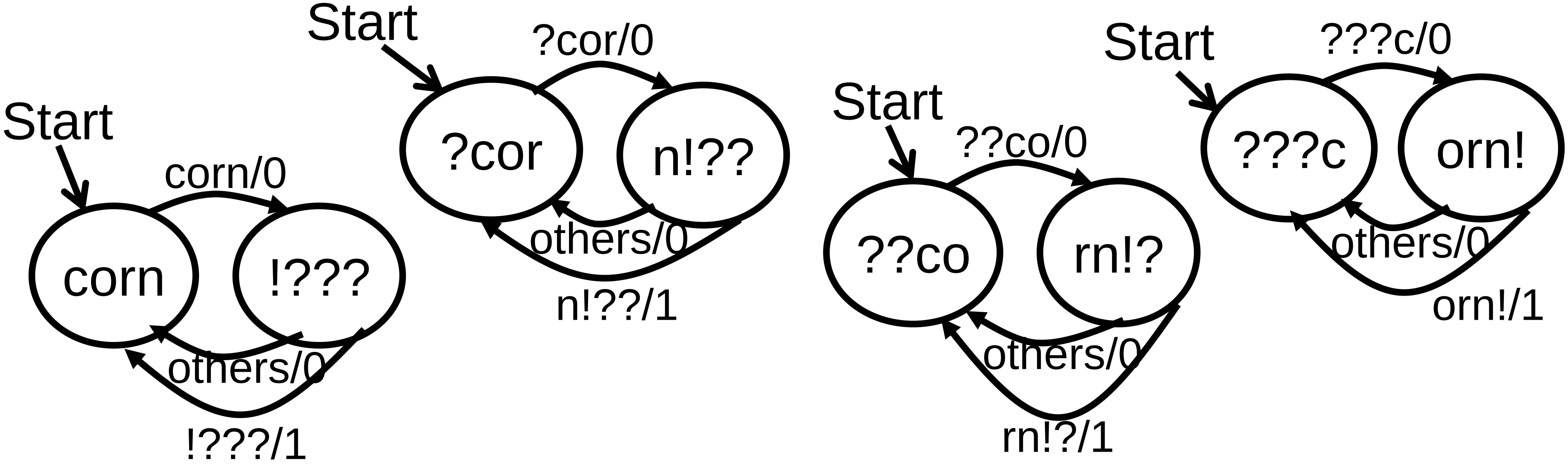
Modify Alert FSM for Multiple characters



Modify Alert FSM for Multiple characters

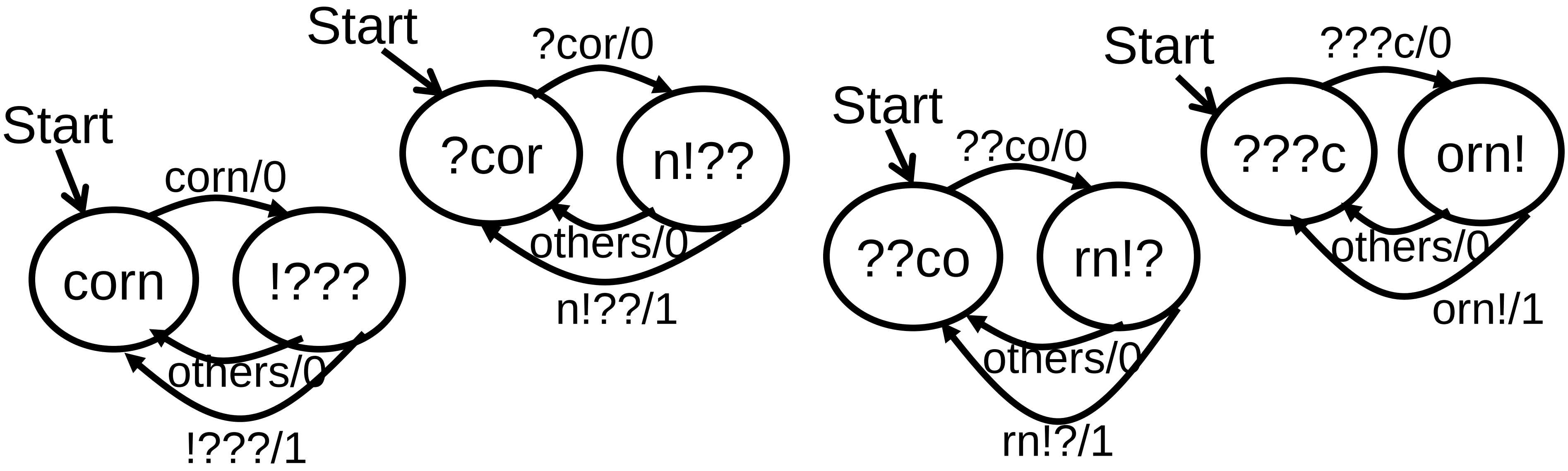


Modify Alert FSM for Multiple characters



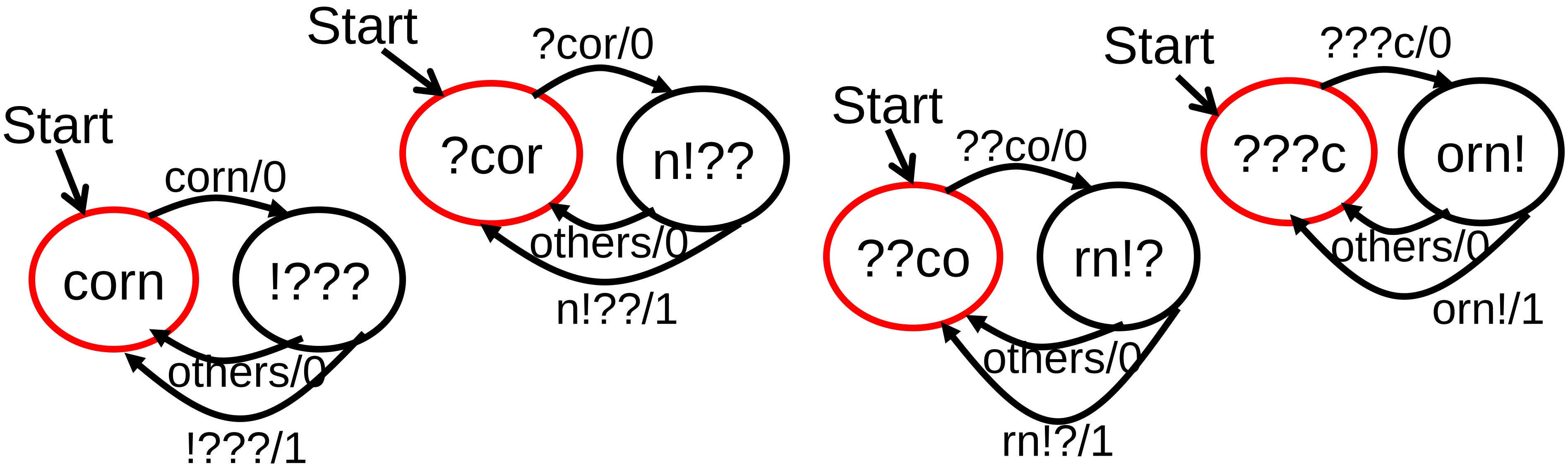
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



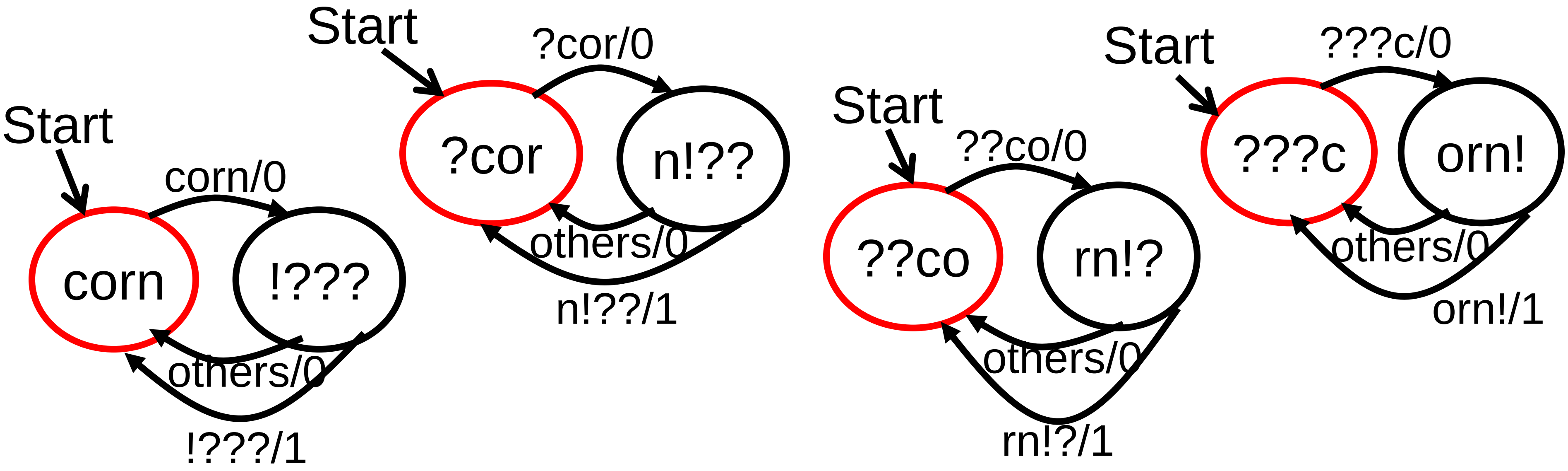
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



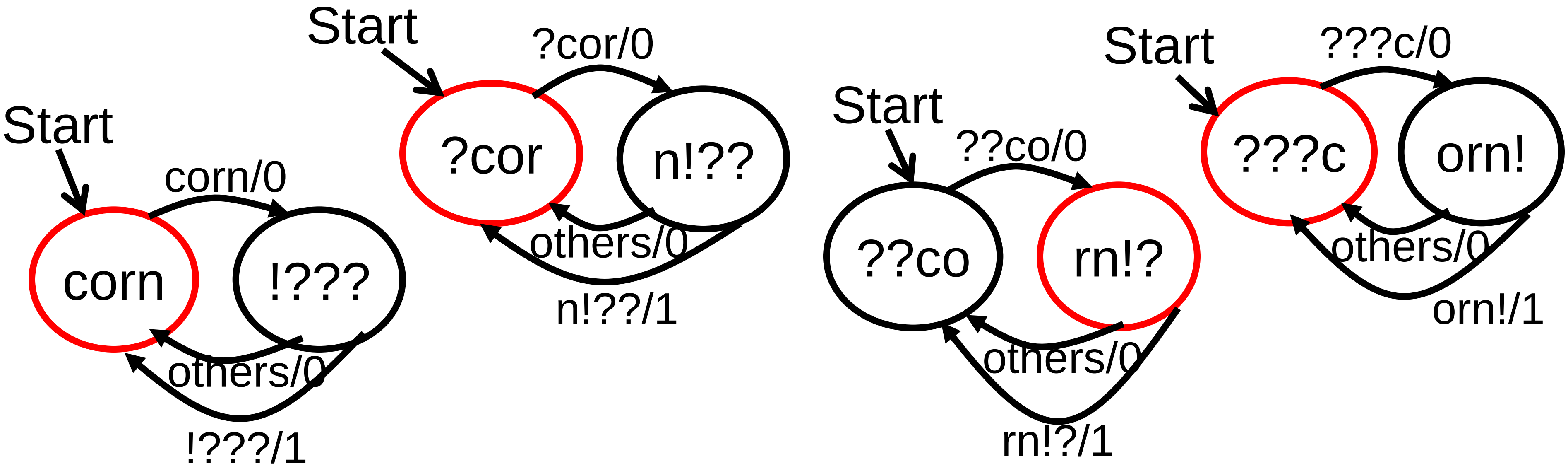
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



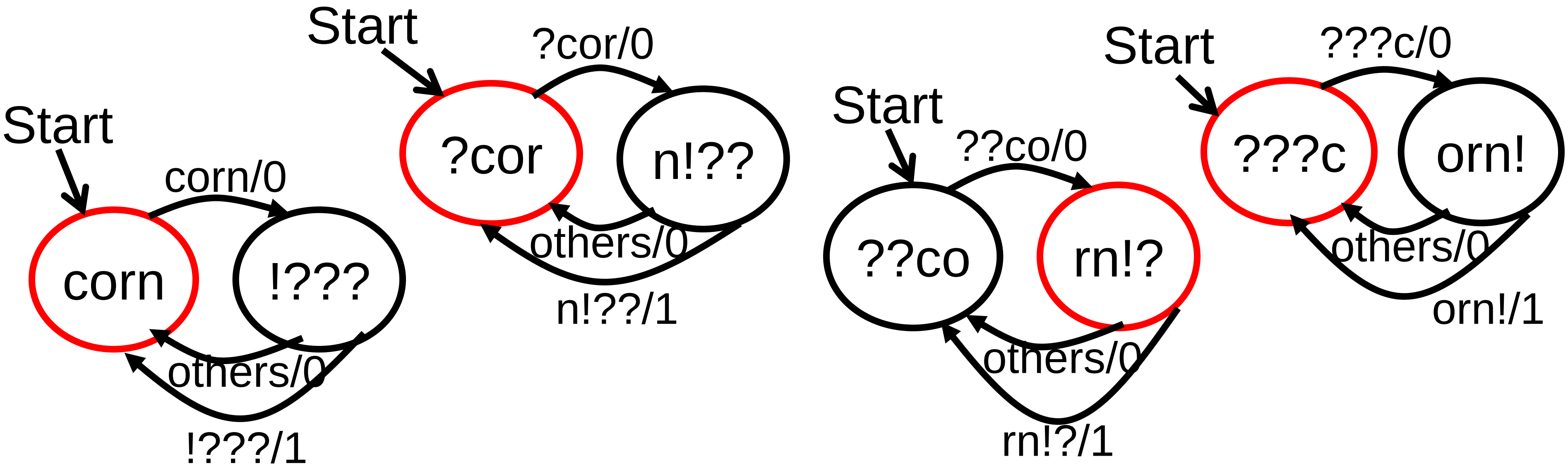
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



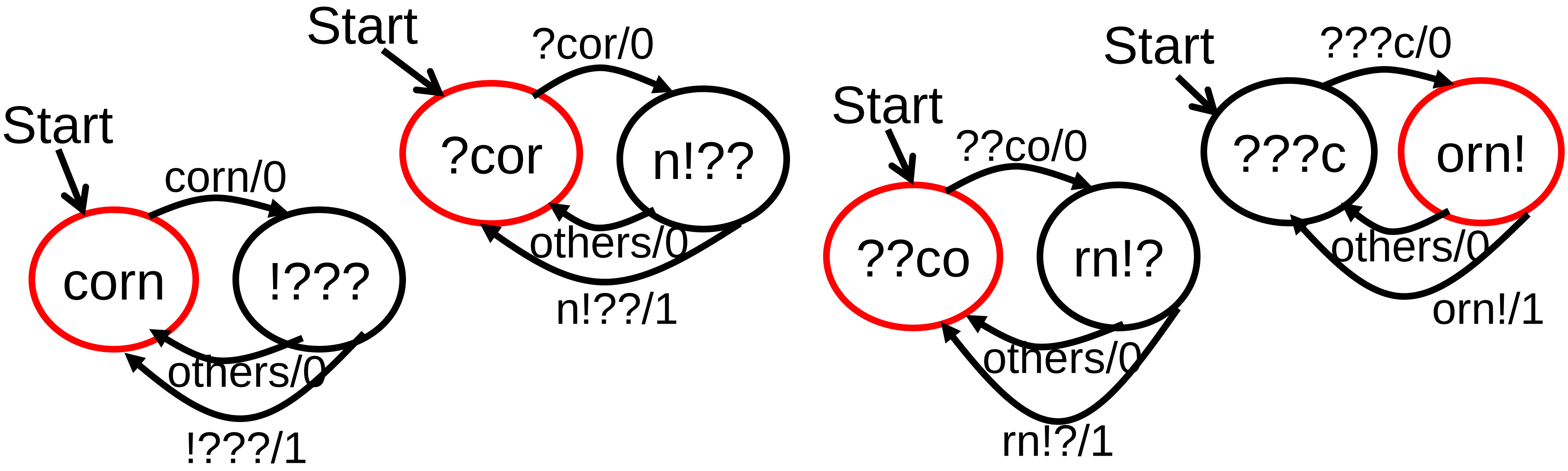
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



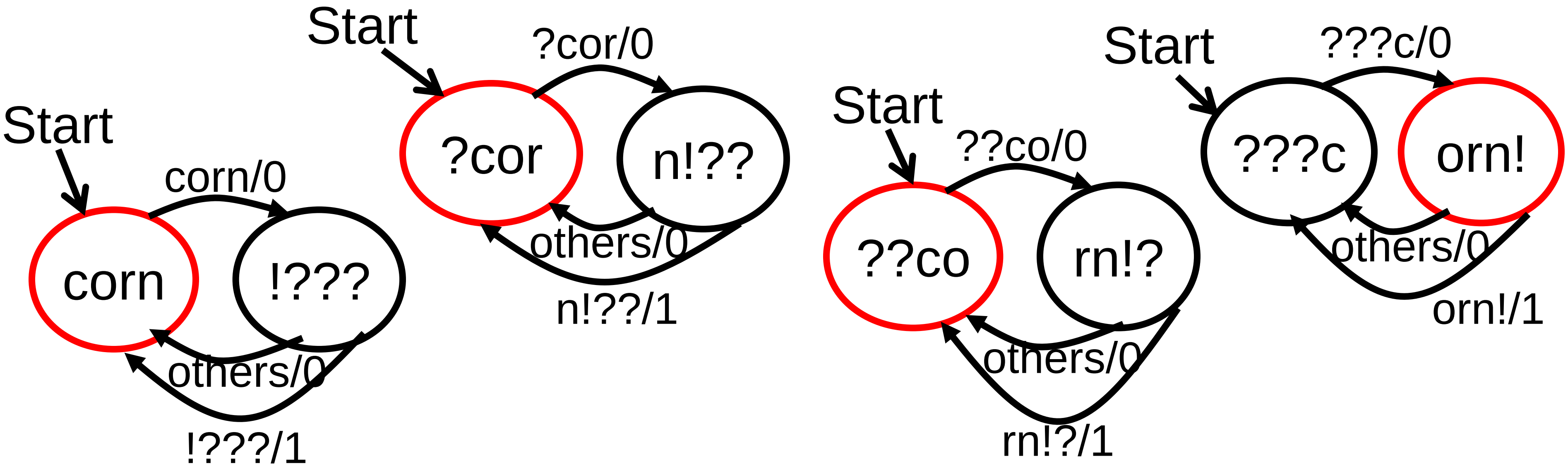
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



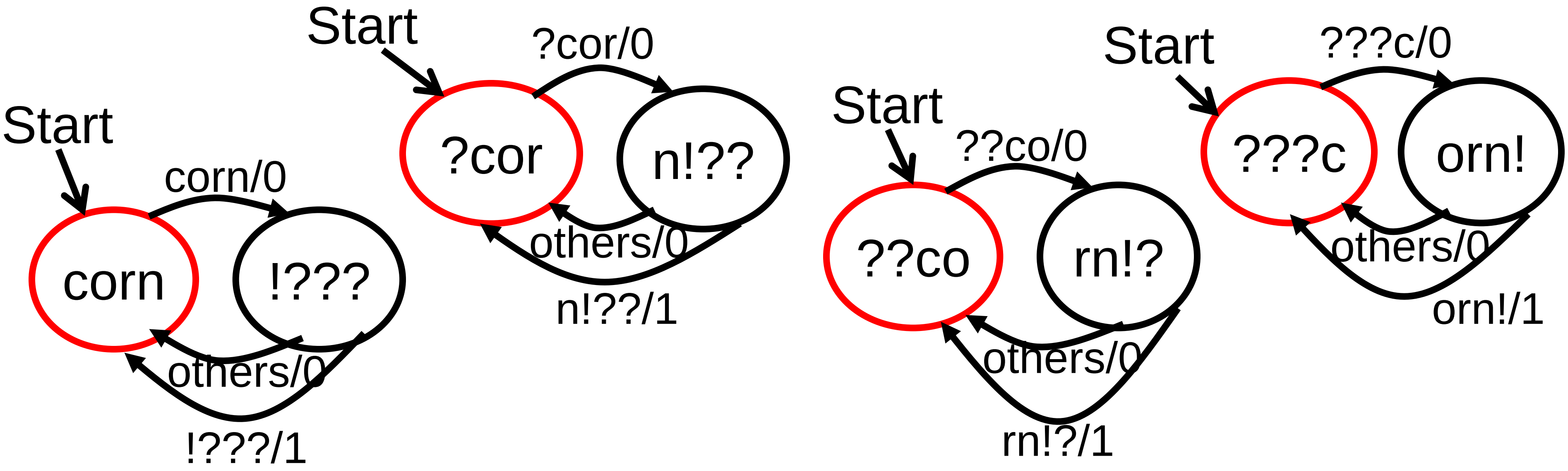
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



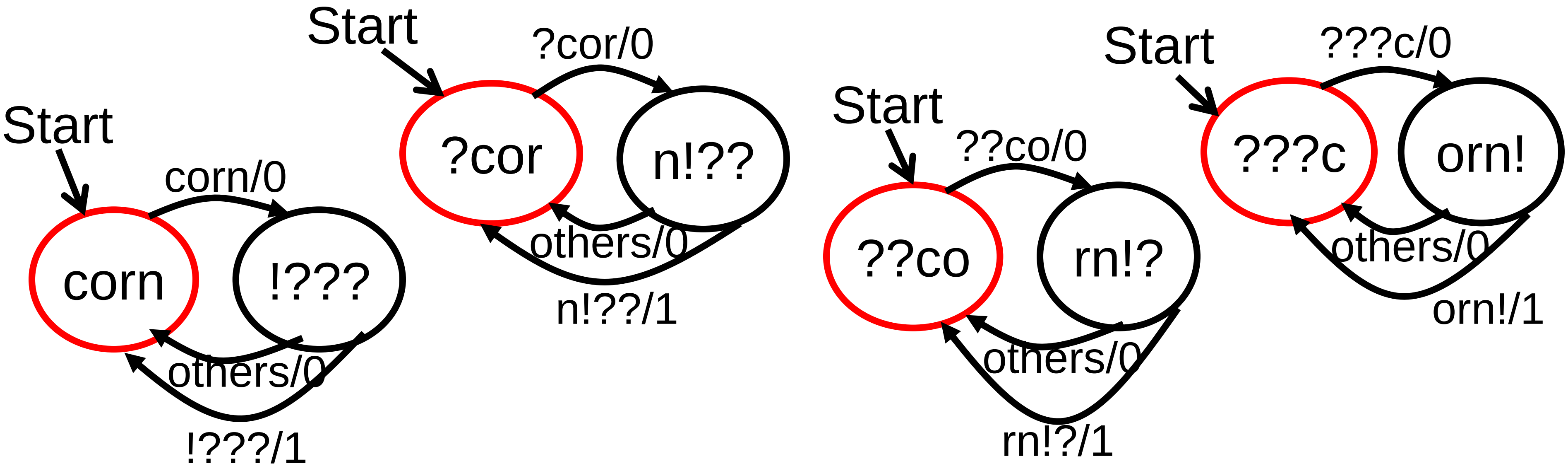
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



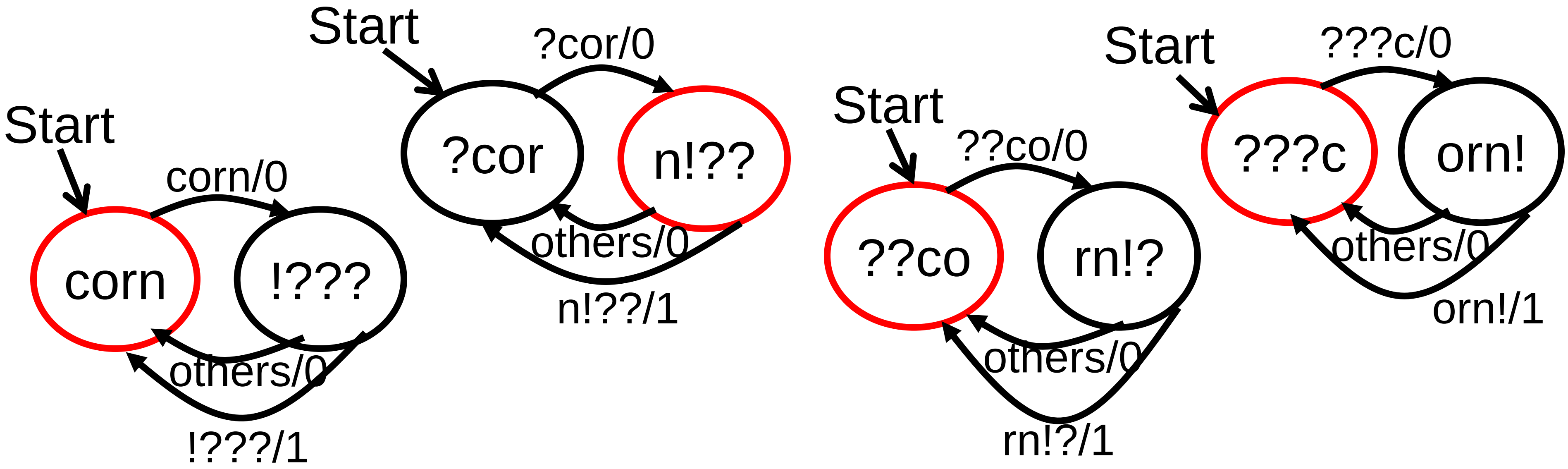
Modify Alert FSM for Multiple characters

c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r

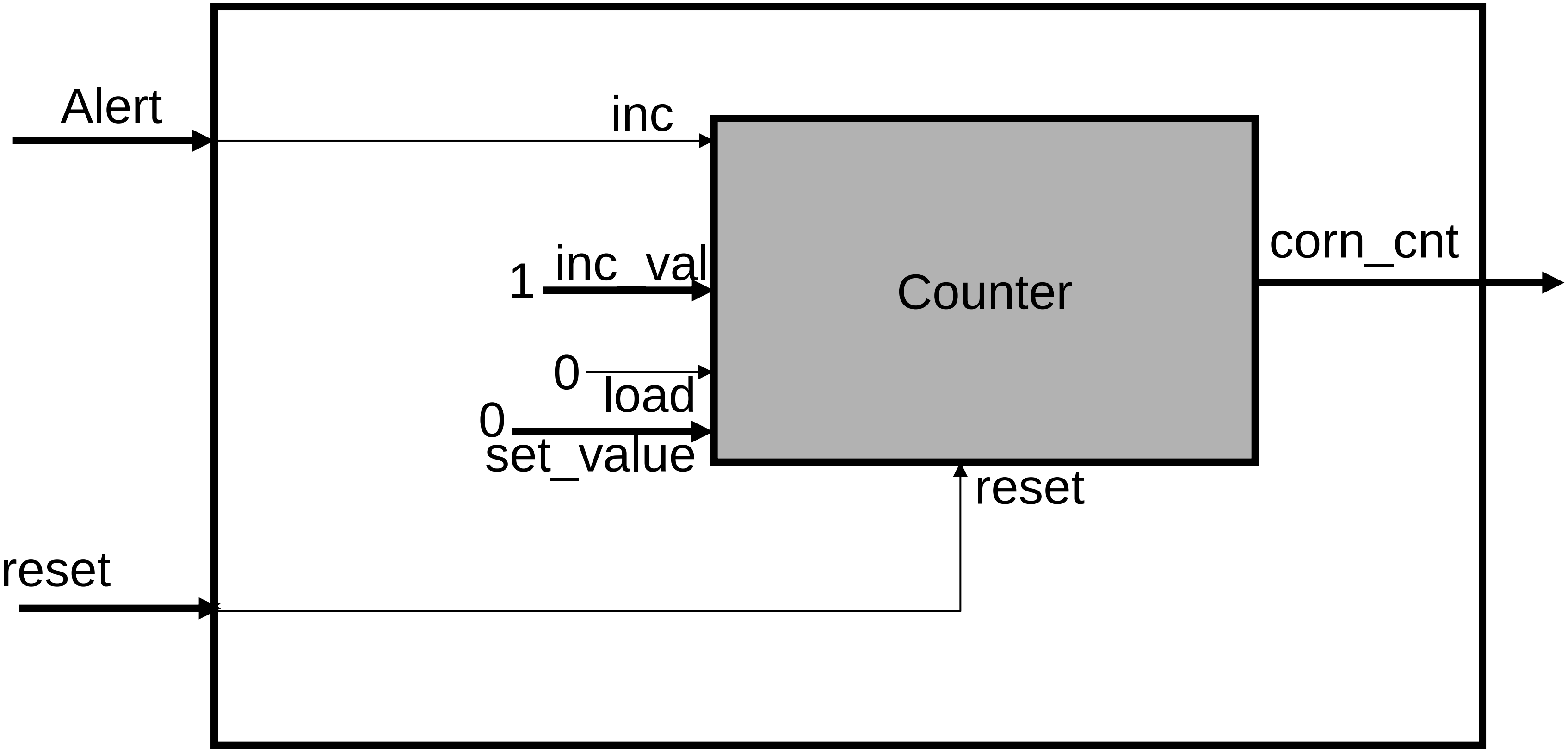


Modify Alert FSM for Multiple characters

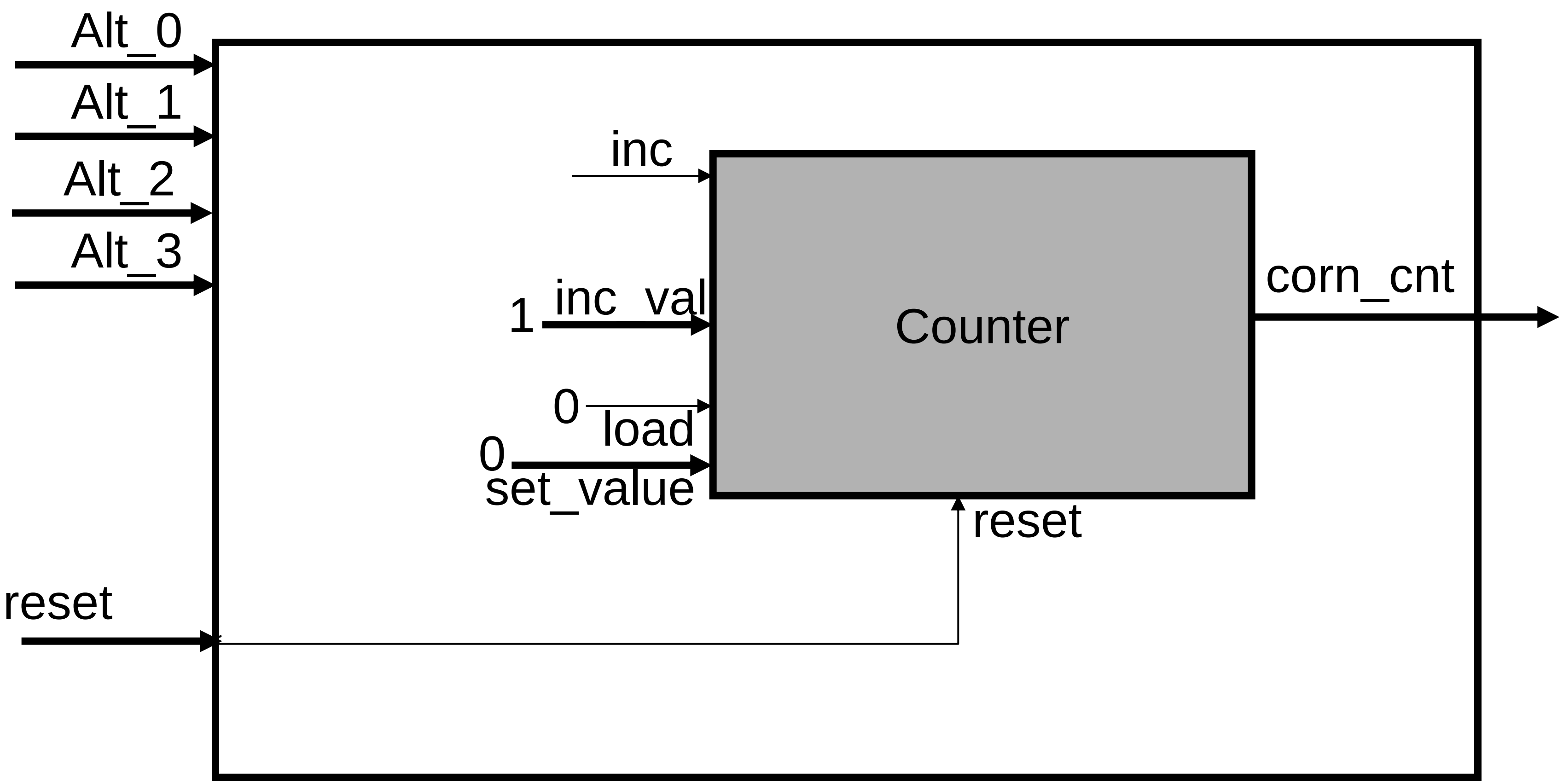
c	b	c	o
r	n	!	c
o	r	n	!
z	c	o	r



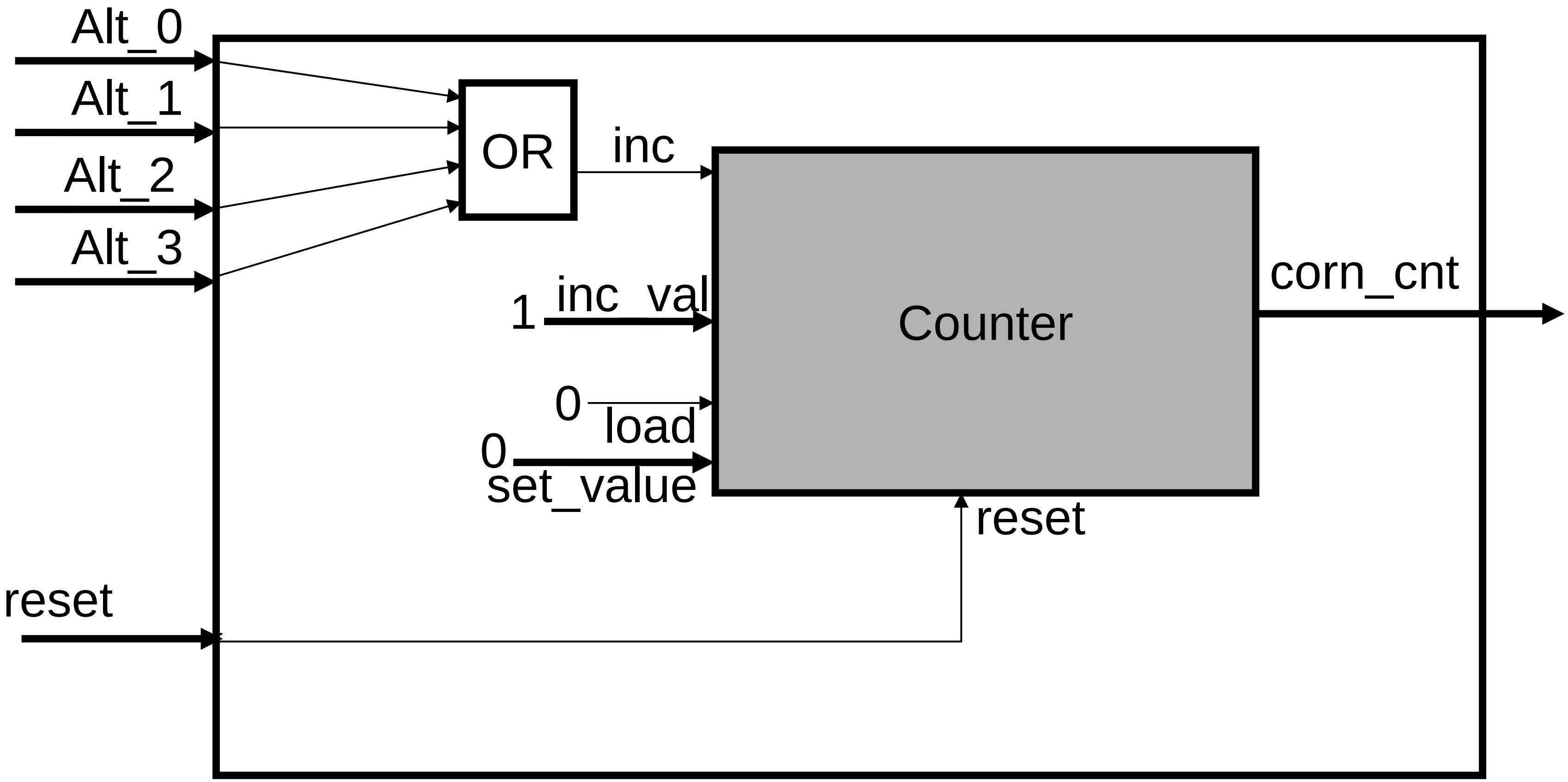
Modify corn! counter for Multiple characters



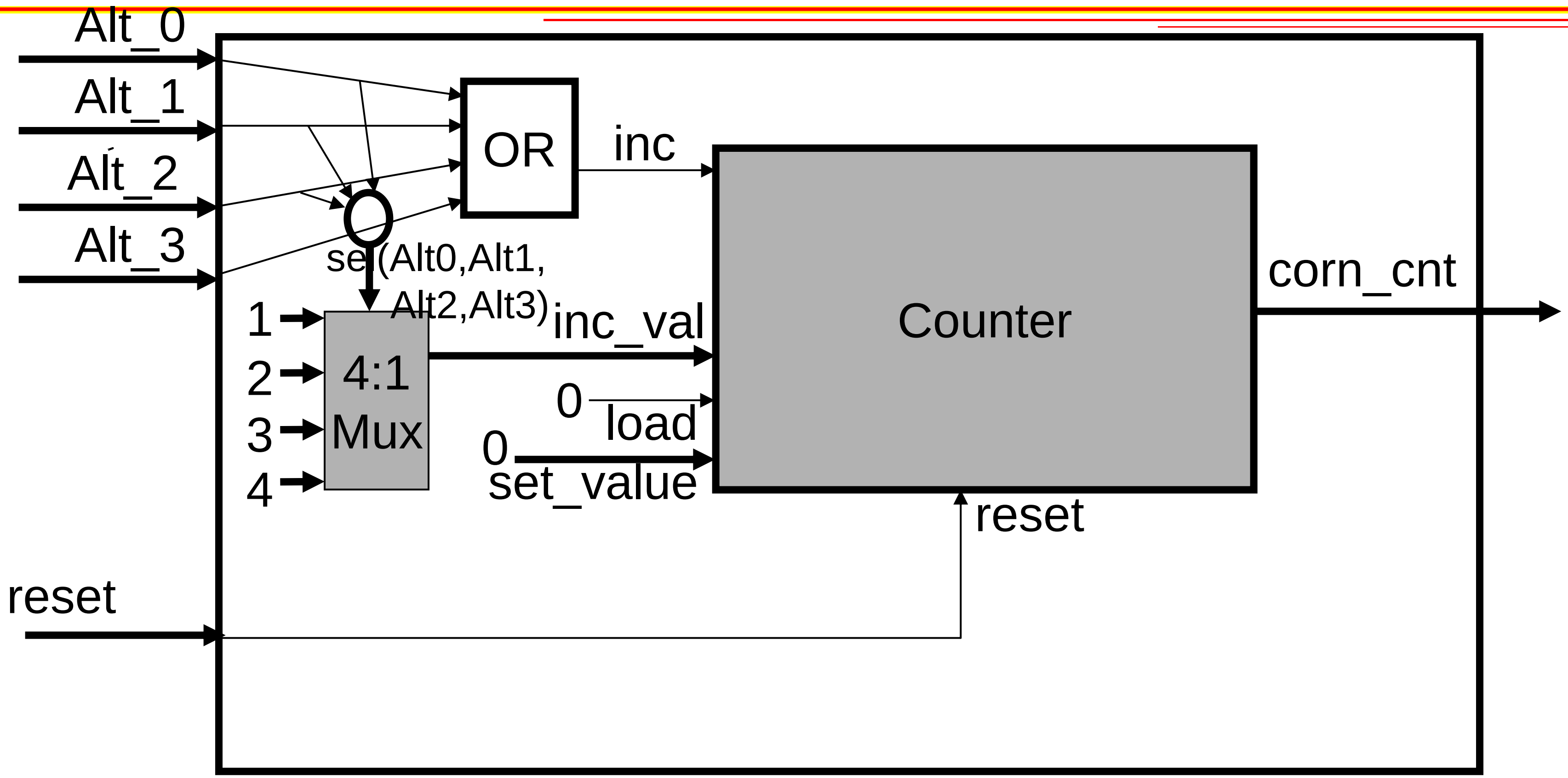
Modify corn! counter for Multiple characters



Modify corn! counter for Multiple characters

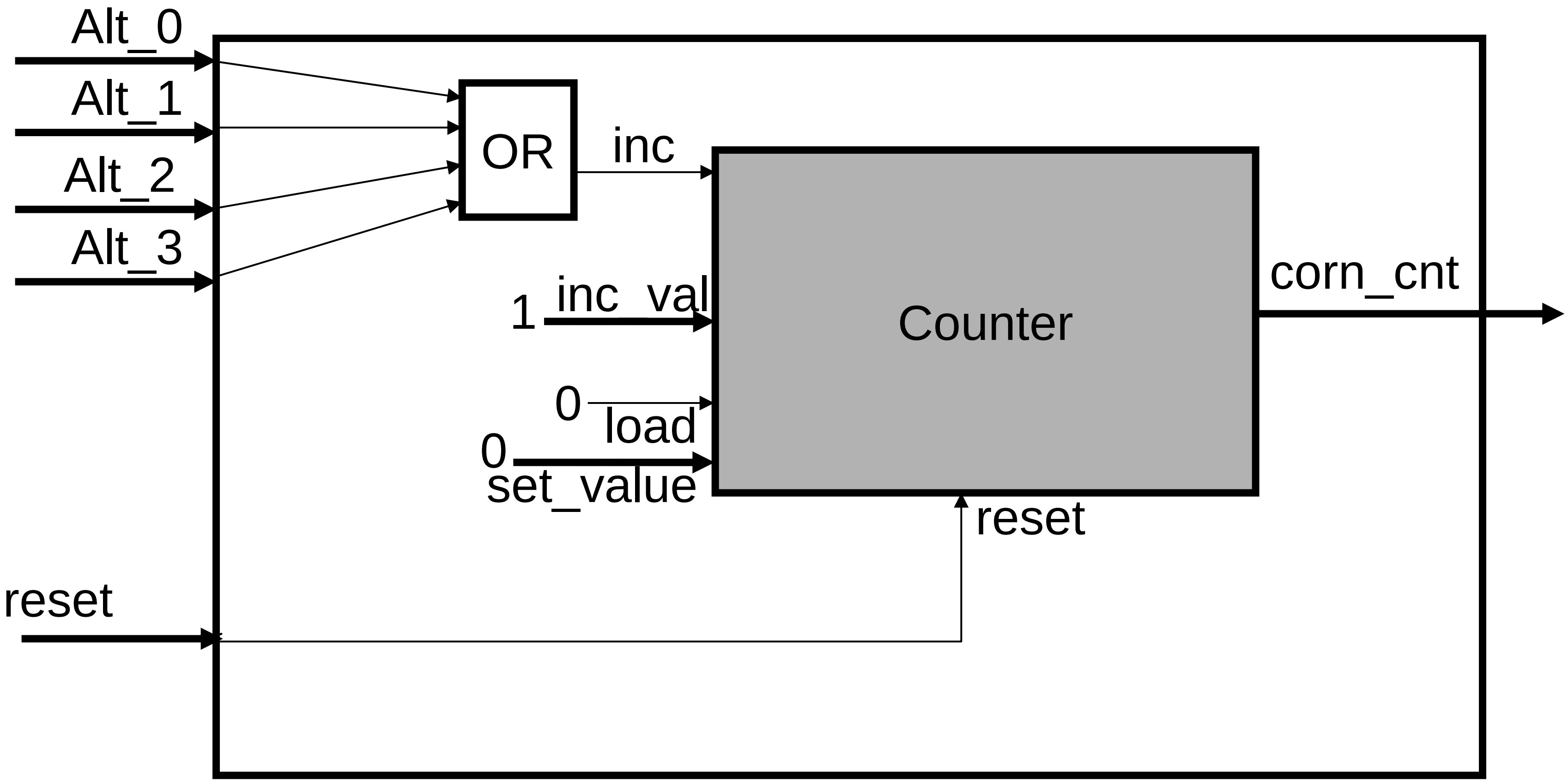


Modify corn! counter for Multiple characters



NOT in a process!
Alt_merge <= Alt0 & Alt1 & Alt2 & Alt3;
inc_val <= 4 when (Alt_merge = “1111”)
3 when (Alt_merge = “0111” or Alt_merge = “1011” ...)
2 when (Alt_merge = “0011” or Alt_merge = “0110” ...)
else 0

Modify corn! counter for Multiple characters


Alt_0
Alt_1
Alt_2
Alt_3
OR
inc
1
inc_val
0
load
0
set_value
Counter
corn_cnt
reset
reset