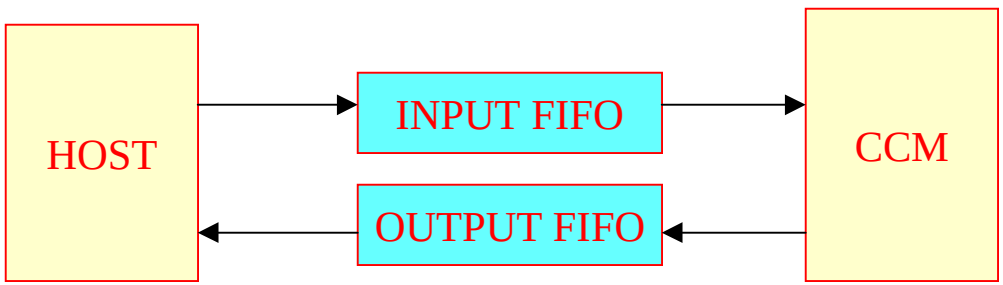
# Architecture Of Cube Calculus Machine

# Communication Between the Host and the CCM

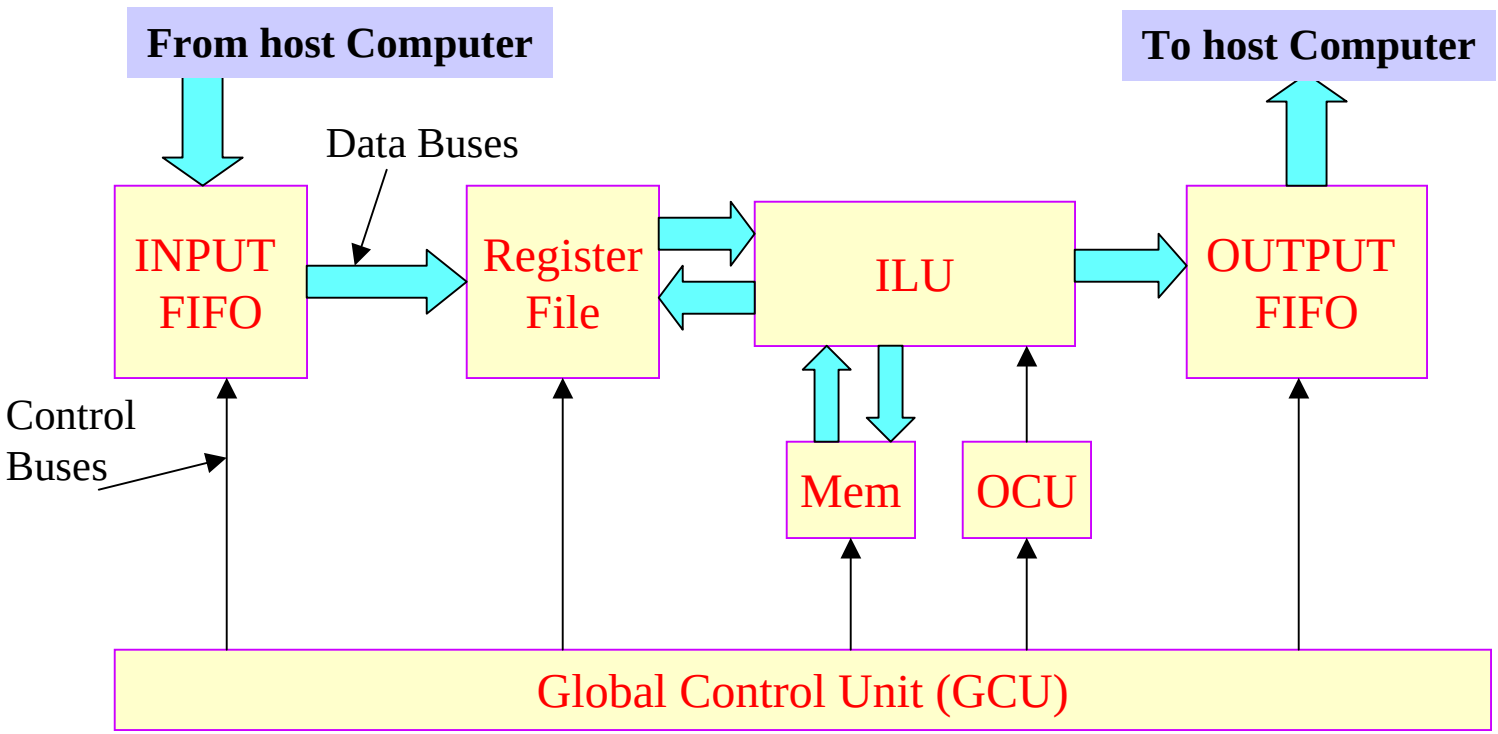
- The cube calculus machine acts as a coprocessor to the host computer.




- The host puts instructions into the input FIFO and receives the results from the output FIFO
- The CCM takes instructions from the input FIFO, executes it and puts the results into the output FIFO.

# The Architecture of CCM

- The simplified Block diagram of CCM:



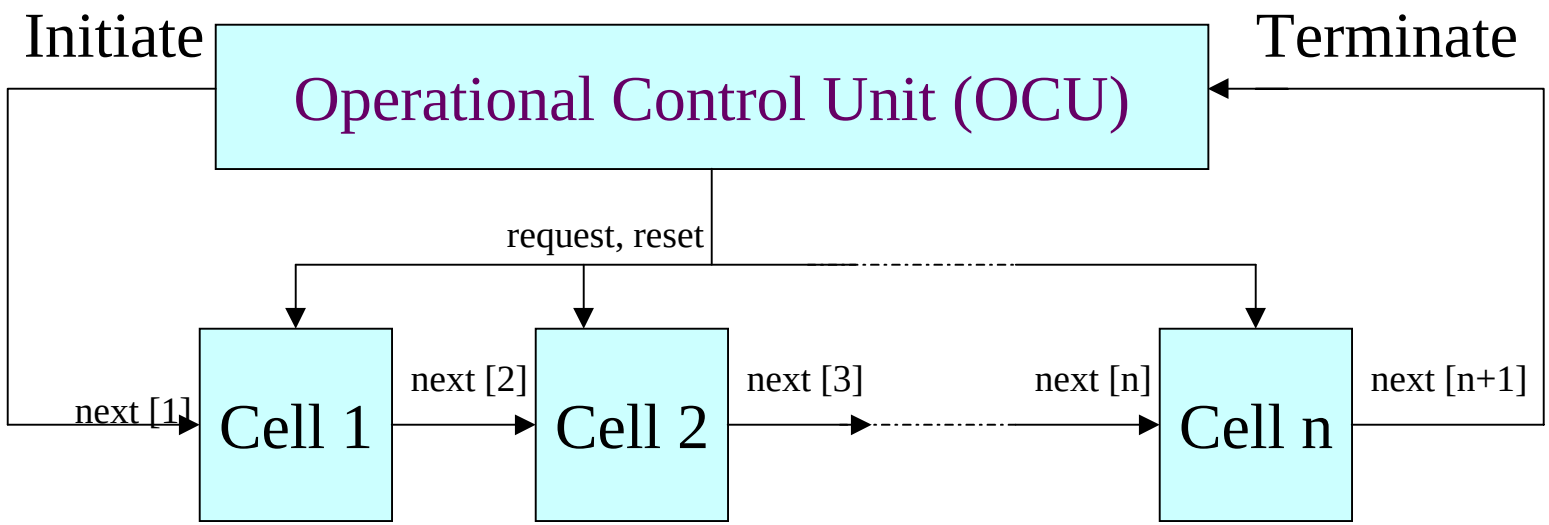
# The Architecture of CCM

- The CCM communicates with the host computer through the input and the output FIFOs.
- The ILU can take the input from register file and memory and can write output to the register file, the memory and the output FIFO.
- The ILU executes the cube operation under the control of *Operation Control Unit* ***(OCU).***
- The *Global Control Unit (**GCU**)* controls all the parts of the CCM and let them work together.
- Let's discuss the **ILU** in more detail.

# ITERATIVE LOGIC UNIT (ILU)

# ITERATIVE NETWORK OF CCM

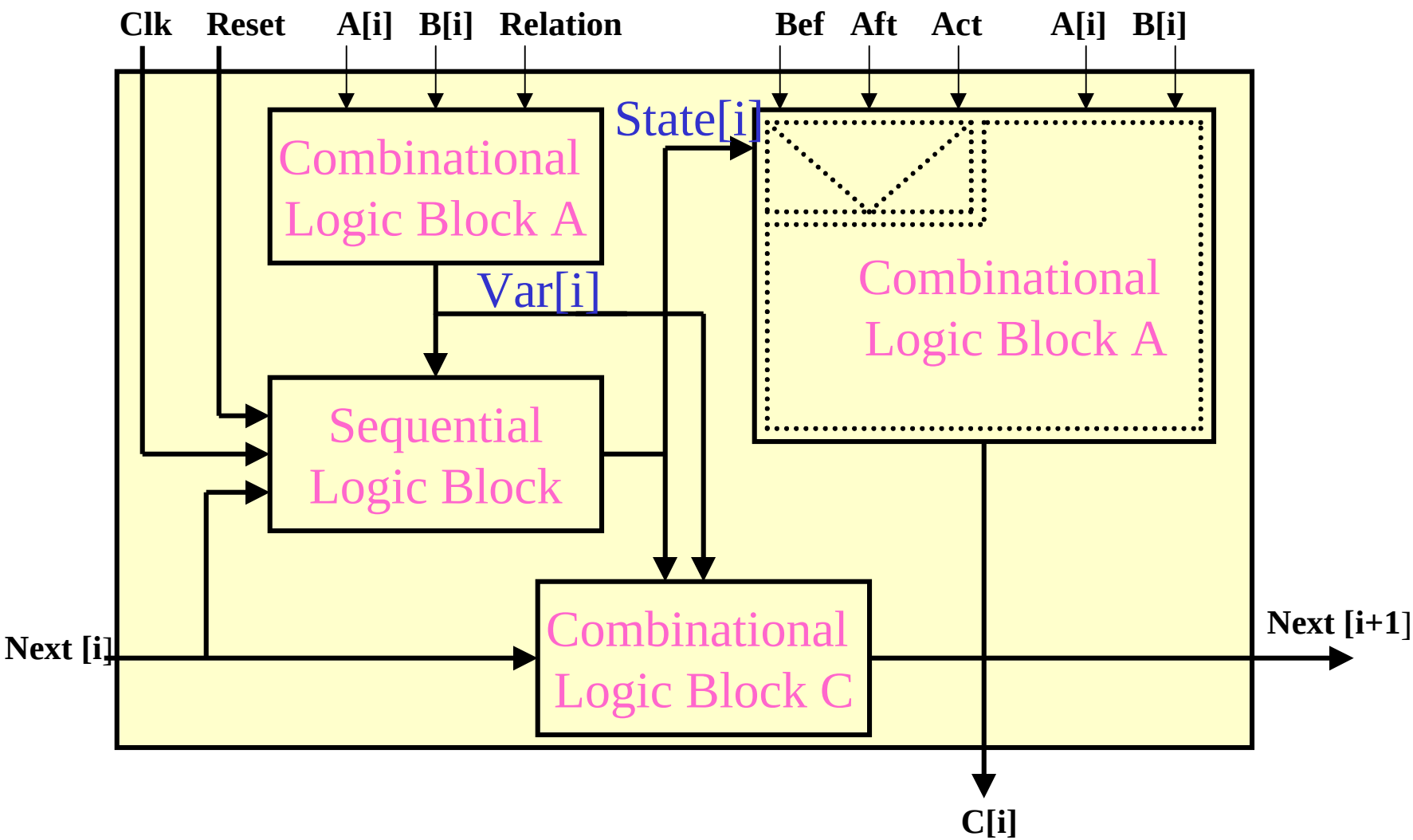
- The **ILU architecture** consists of an iterative network of cells and a control unit which controls the operation of these cells.




The iterative network of the CCM

- Here the I/P's A[i], B[i], Relation, Bef, Act, Aft come from the register file and is not shown in the figure and o/p C[i], is also not shown which goes to register file.

# The block diagram representation of the Iterative cell used in CCM



# Explanation of Iterative Cell

- The Iterative Cell consists of 3 combinational blocks and 1 sequential block.
- **Function of Combinational block A**: To take the input variables, apply a relation between the variables and provide the output **Var[i] .**
  Var[i] represent whether the variable is a special variable or not.
  Var[i] = 1 ⇒ The variable is a special variable (The relation is satisfied).
  Var[i] = 0 ⇒ The variable is not a special variable (The relation is not satisfied).

# Explanation of Iterative Cell

- The function of **combinational block A** is to identify a special variable Var[i], therefore the block is named as ***IDENTIFY*** block.
- **Var[i]** signal is fed to **sequential logic block** along with the other inputs(Clk,Next[i], Reset) and provides an **output ( State[i]),** which in turn decides which operation (Bef, Aft, Act) to be performed by the **Combinational block B.**

# Explanation of Iterative Cell

- **Combinational block B** is a block that **performs Bef, Act, Aft** operations **on input variables** depending on the input received from the sequential logic block (state[i]) and gives the output variable **C[i]** .
- It has two input bits from operand literals A[i], B[i] respectively, two bits for signal state[i],12 bits programmable inputs for functions ***before***, ***active***, ***after*** and two output bits C[i].
- The block performs different **operations** on input variable and hence named as **OPERATION** block

# Explanation of Iterative Cell

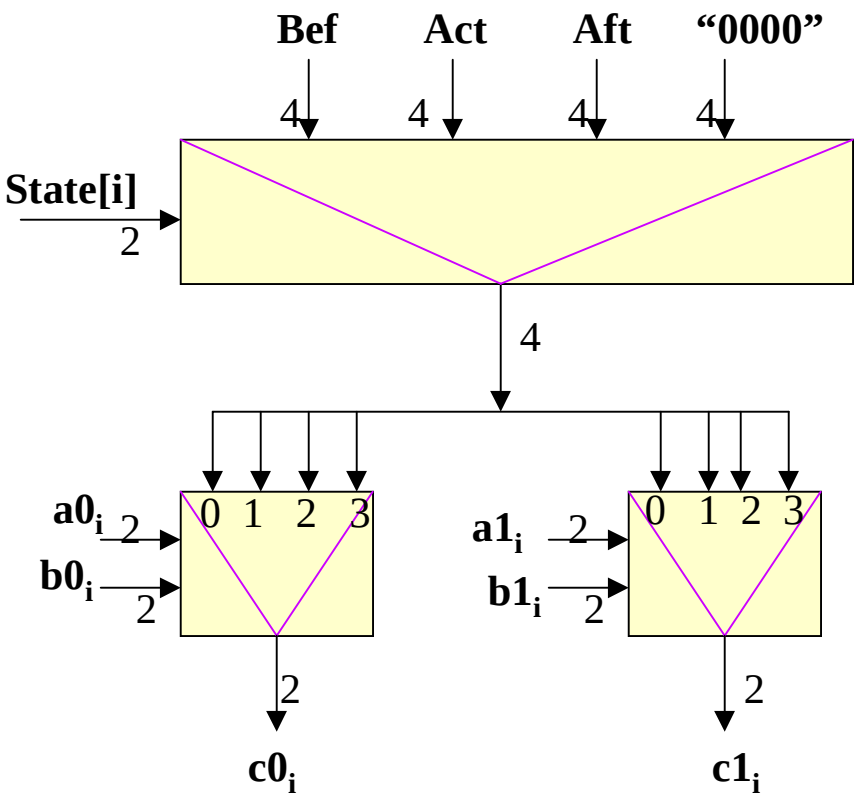
by Multiplexer.

- Here the signal state[i] selects one function from the three possible functions
- $\Rightarrow$ this realization can be done by using **one 4 to 1 multiplexer**,
- here there are only three possible functions, so the last data input is not used and is connected to a constant "0000".

- Then the function is to be applied to inputs A[i], B[i] again this is realized by the use of **two 4 to 1 multiplexer**.
- Total of three 4 to 1 multiplexers are used for the realization of combinational block B **(OPERATION block)**

# Realization of Operation block of IT

- **OPERATION** block of IT:

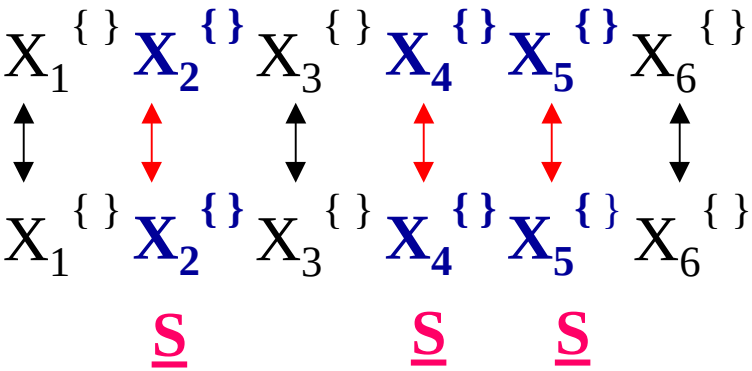


# Explanation of Iterative Cell

- Before explaining the function of **combinational logic block C**, let's explain why we require **Next[i]** and **Next[ i+1]** signals.

# Why We require Next signals in Iterative Cell

- For **Sequential cube calculus** operations the o/p result consists of as many cubes as the number of special variables.
- Let's take an Example which has 6 variables and 3 special variables in it at 2, 4, 5 position.



# Why We require Next signals in Iterative Cell

**1st Step**: Check for the special variables.

**2nd Step**: Generate 3 cubes as a result, each cube is the result of performing an Act operation on one special variable and Aft operation on variables that lie to the left of that special variable and Bef operation on the variables that lie to right of that special variable.

⇒Special variables are taken one at a time,

⇒Generating one cube at a time

⇒**3rd Step**:we need to find a way to activate special variables in series, and all other variables should know their relative position with respect to current active variable, left or right.

- This is done by Combinational logic block C.

# Explanation of Iterative Cell

- The function of **Combinational logic block C** is to produce an o/p when the variable is active variable

  ($\Rightarrow$ act[i] is true) or When the next[i] signal is 1 and Var[i] signal is 0.

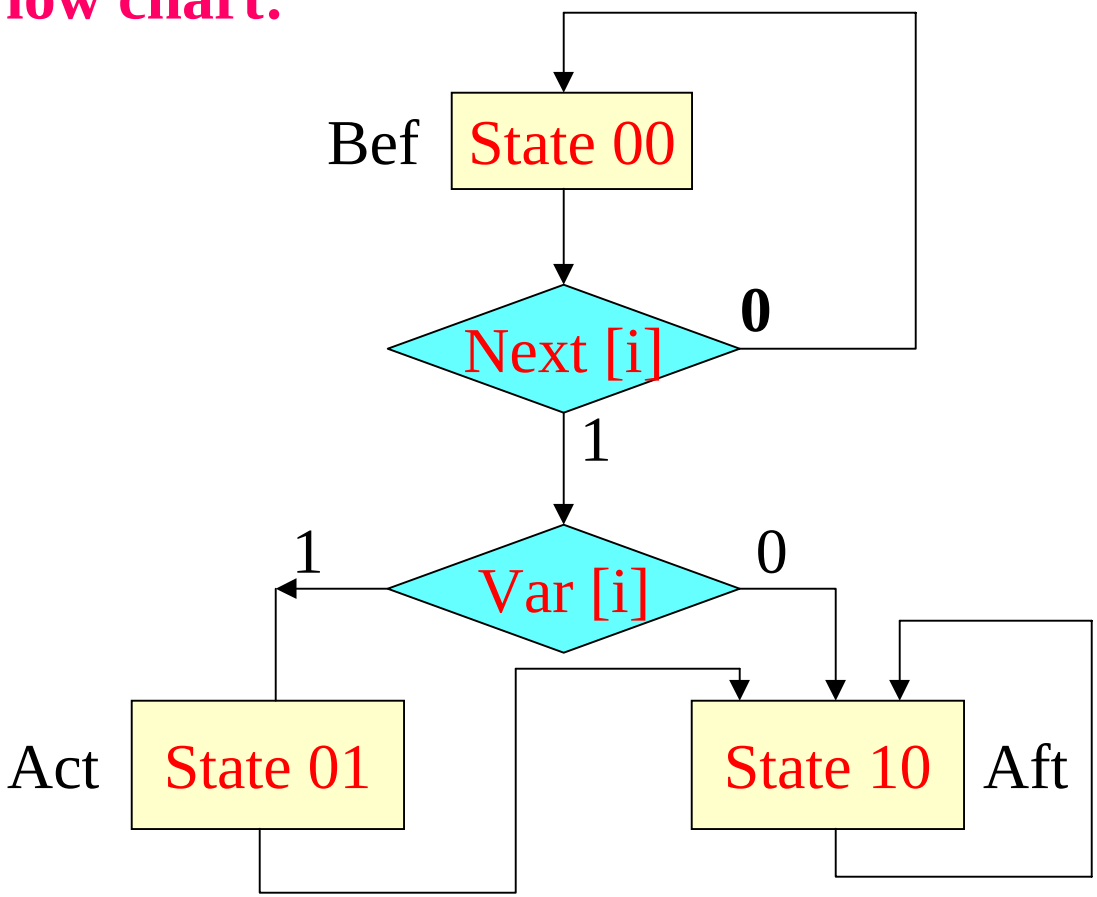
- Therefore the **logic of Combinational logic block C** is next[i+1] = act[i] + next[i]. Var[i].

# Explanation of Iterative Cell

- The **Sequential logic block** consists of:
- FSM (Finite State Machine),
  - the FSM has three states(Bef, Aft, Act)
  - the o/p of FSM is state[i] signal.
  - Here the FSM is reset to Bef state.

# Flow chart of the FSM

- **Flow chart:**



# General Expression for three states of FSM

- The **state machine** is described by:

  $bef[i]=\overline{state1[i]}*\overline{state0[i]}$

  $act[i]=\overline{state1[i]}*state0[i]$

  $aft[i]=state1[i]*\overline{state0[i]}$

- Since there are **three states** $\Rightarrow$ the machine is realized by using **two D FF.**
- The **current state** of the FSM is represented by the Q outputs of two FF, which are **state0[i] and state1[i].**
- The **next state** of the FSM are the inputs of the two FF.
- Let's denote those state as **ex0 and ex1.**

# General Expression for three states of FSM

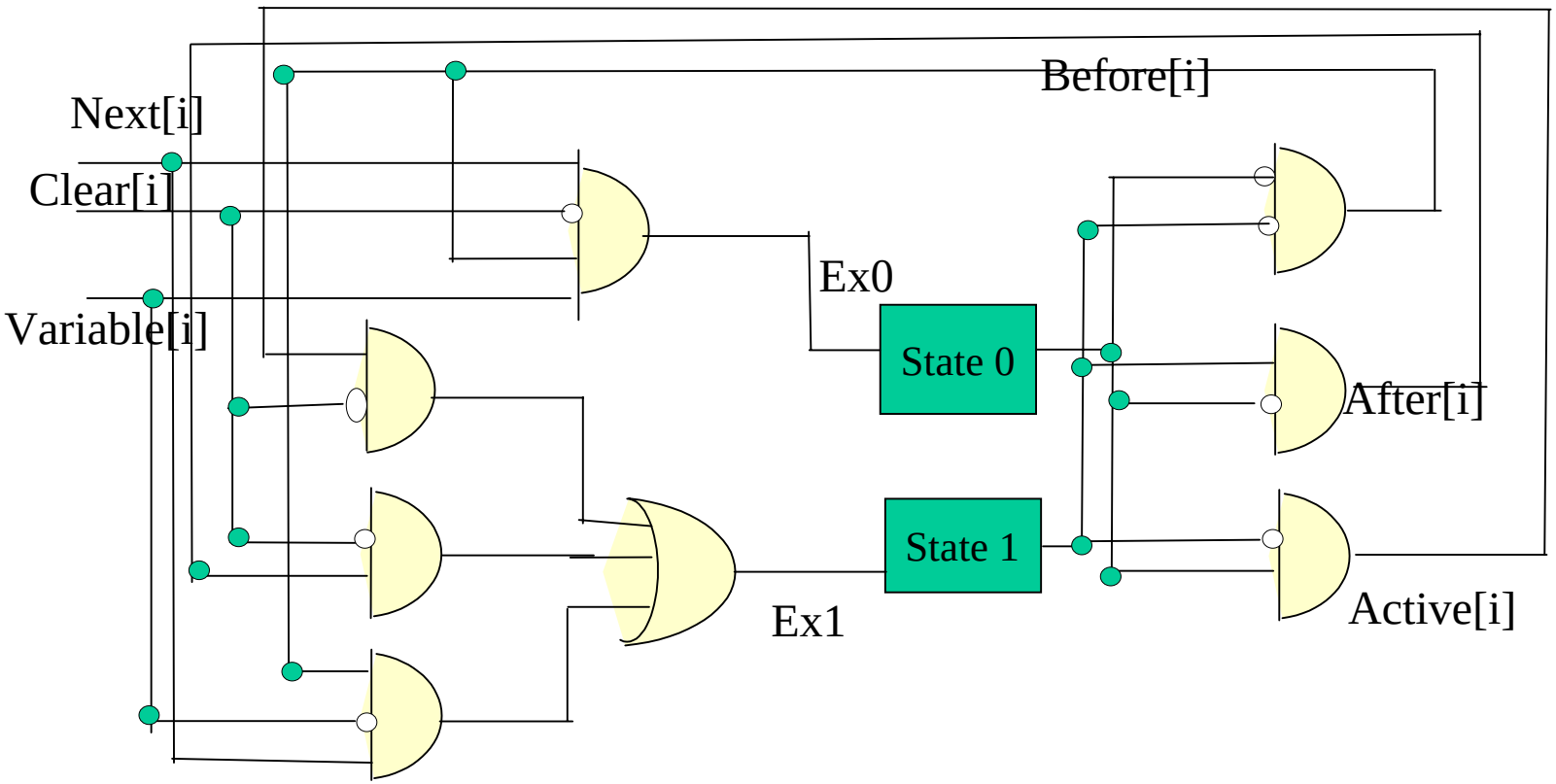
- From the flow chart we know that the ex0 and ex1 are act state and aft state.
- Looking at the flow chart we have

$$ex0[i] = bef[i] * next[i] * var[i]$$

$$ex1[i]=bef[i] * next[i] * \overline{var[i]} + act[i] + aft[i]$$

- There may an error in negation in this equation

# Gate Level realization of state[i]



# Explanation of Iterative Cell

- The function of **Sequential circuit and combinational circuit C** together is to calculate the state of the variable (State[i]) and propagate the information to next iterative cell and hence the name **STATE** block.

# Additional Inputs to Iterative cell for Handling Multi Valued Function

# Additional Inputs to Iterative Cell

- CCM is a hardware, when it is realized , it has fixed number of iterative cells.
  - When we use CCM to solve a problem we **cannot always use all its iterative cells**.
- Therefore we need **a signal vector** to tell whether a given iterative cell **is used by the operation or not**.
  - This signal is **Water[i] (w[i]),** where i=1,2-----n.
- The signal **w[i] = 1** $\Rightarrow$
  - that particular **IT is not used** and it should pass all signals running horizontally, like the next signal.
  - $\Rightarrow$ IT is transparent.

# General Expression for W[i]

- **General expression for w[i]**

W[i]=1---------- IT[i] is **not used**

W[i]=0---------- IT[i] is **used**

# Additional Inputs to Iterative Cell for Handling Multi valued Variable

- We need a **signal vector** to tell where is the **boundary** of a multi valued variable.
  - This signal vector is **right_edge[i] (re[i]),** where i =1,2----n.
- The signal **re[i] = 1** $\Rightarrow$ IT[i] is right edge of a variable
- or IT[i] is the **Last IT** of a variable .

OR

- The signal **re[i-1] = 1** $\Rightarrow$ IT[i] is **first IT** of a variable.
- Because **IT[1]** is always the first IT of a variable $\Rightarrow$ **re[0] = 1**

# General Expression for re[i]

- **General Expression of re[i]**

re[i]=1---------- IT[i] is **Last IT** of a variable
re[I-1]=1---------- IT[i] is **First IT** of a variable.

# Example for Right_edge signal and water_signal

- **Example**: Consider CCM with 6 Iterative cells. For a given cube operation, there are 3 variables with 2, 4, 6 possible values respectively.
- **Right_edge signal**: 1 - 01 - 001

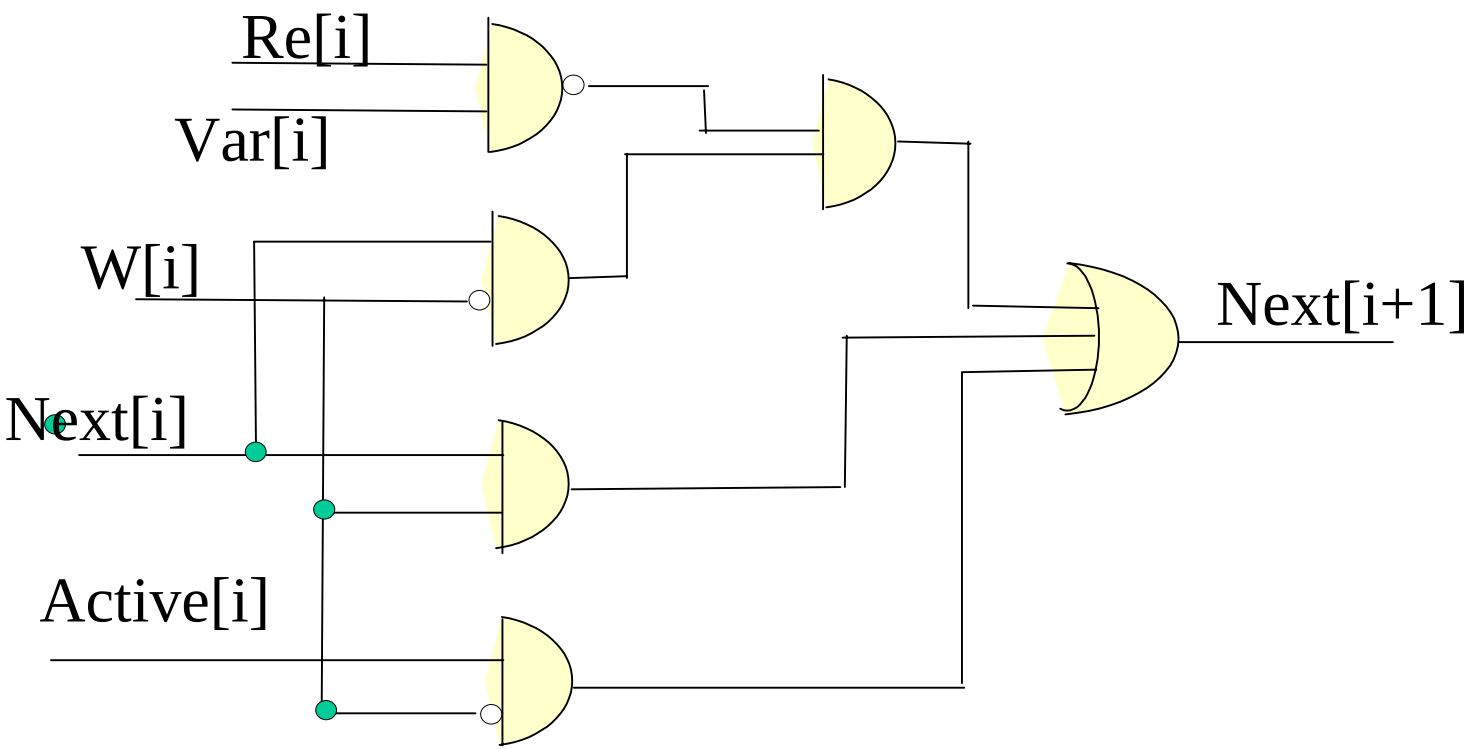
  Var1 Var 2 Var 3
- **Water_signal** : 0-00-000

- Therefore the signal **water[i]** and **re[i]** are used for **identifying** purpose and hence are the inputs to **identify block.**

# Handling Multi-valued Functions

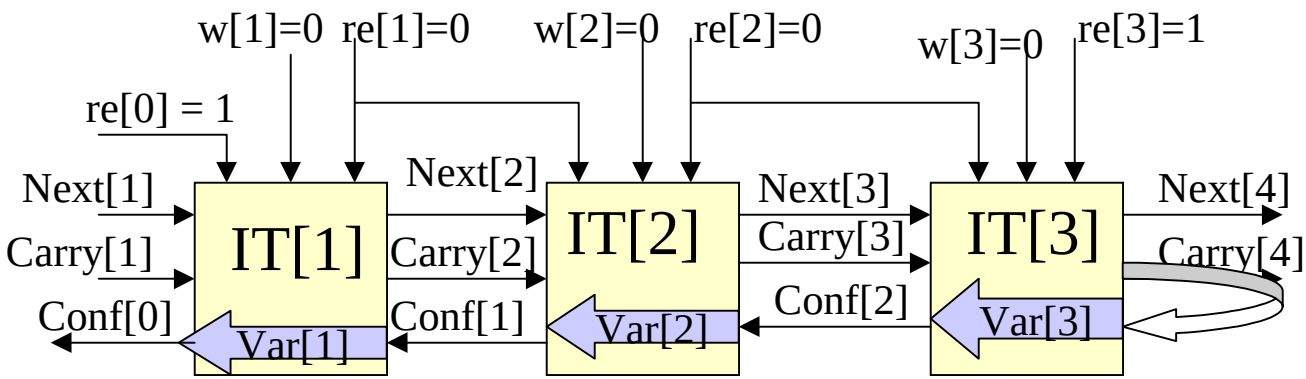
- For multi valued variable, the next signal is described as:
  When **IT[i]** is **not used (w[i]=1)** the iterative cell should be **transparent** to signal next $\Rightarrow$ **next[i+1] = w[i ].next[i].**
  otherwise
  The **next signal propogates till the right edge** of the first special variable that it will encounter **when w[i]=0.**
  $\Rightarrow$ **next[i+1] = $\overline{w[i\ ]}$(act[i] + next[i]. $\overline{Var[i].\ rel[i]}$)**
- For handling **multi valued variables** the next signal is described as :
  **next[i+1] =$\overline{w[i\ ]}$(act[i]+next[i].$\overline{Var[i].rel[i]}$)+w[i ].next[i]**

# Gate Level realization of next[i+1]



# Propagation Signals for combining multiple iterative cells

- Example:To process a pair of operand literals of a 6 valued variable ⇒ 3 Iterative Cells are combined.
- Here we assume that 3 iterative cells are used and since all the cells are used ⇒ w[1]=0, w[2]=0, w[3]=0.
- IT[1] is the first IT of a variable ⇒ re[0] = 1
- IT[3] is the last IT of a variable ⇒ re[3] = 1



# Propagation Signals for combining multiple iterative cells

- Here **three Iterative cells** are combined together therefore the signal **Var** cannot be defined by a single iterative cell because for a given **OR type cube operation**:
  Var = rel($a_1$,$b_1$)+ rel($a_2$,$b_2$)+ rel($a_3$,$b_3$)+ rel($a_4$,$b_4$)+ rel($a_5$,$b_5$)+ rel($a_6$,$b_6$)
  since a single iterative cell processes just two possible values
  carry[2]= rel($a_1$,$b_1$)+ rel($a_2$,$b_2$)
  carry[3]= carry[2]+ rel($a_3$,$b_3$)+ rel($a_4$,$b_4$)
  carry[4]= carry[3]+ rel($a_5$,$b_5$)+ rel($a_6$,$b_6$)

# Propagation Signals for combining multiple iterative cells

- $\Rightarrow$ carry[4] = rel($a_1$,$b_1$)+ rel($a_2$,$b_2$)+ rel($a_3$,$b_3$)+ rel($a_4$,$b_4$)+ rel($a_5$,$b_5$)+ rel($a_6$,$b_6$)
  $\Rightarrow$ carry[4] = Var = Var[3]
  $\Rightarrow$ the **Signal Var** is always **generated** at the **end of last cell** of a variable.
- All the three cells which process the variable should know the signal Var.
- Therefore all other cells that process the same variable **receive** the signal Var **through the propagation signal conf** from its successive cell.

# Propagation Signals for combining multiple iterative cells

- Final equations of different cell

  $\Rightarrow$ Var[3] = carry[4]

  conf[2] = Var[3]

  Var[2] = conf[2]

  conf[1] = Var[2]

  Var[1] = conf[1]

  $\Rightarrow$ The carry propagates from left to right until the right edge of the variable in order to generate signal var of the variable then the signal var is propagated back (from right to left) through conf.

# Propagation Signals for combining multiple iterative cells

- $\Rightarrow$ General Expression for **conf:**

  conf[i-1] { = carry[i+1] if IT[i] is the **last IT** of a variable.
- { = conf[i] otherwise.
- $\Rightarrow$ General Expression of **var[i]** for multi valued variable.

  var[i] = conf[i-1]

  carry[i+1]=var[i] For Last IT of a variable

# Propagation Signals for combining multiple iterative cells

- **General formula** for **signal carry[i]**

  For **OR** type relation:

  carry_or[i+1]:

  = rel0[i]+ rel1[i] if IT[i] is the **first IT** of a variable.

  = rel0[i]+ rel1[i]+carry[i] **otherwise.**

- For **AND** type relation:

  carry_and[i+1]:

  = rel0[i].rel1[i] if IT[i] is the **first IT** of a variable.

  = rel0[i].rel1[i].carry[i] **otherwise**.

# Combining the re[i] with Carry[i]

- we know for Right_edge signal:
  **re[i-1] = 1** $\Rightarrow$ IT[i] is **first IT** of a variable .
  **re[i] = 1** $\Rightarrow$ IT[i] is **last IT** of a variable.
- Taking into **consideration right edge signal** into carry_or[i+1] :
  For OR type relation:
  **carry_or[i+1]:**
  $=(rel0[i]+ rel1[i] ).re[i-1]+(rel0[i]+ rel1[i]+carry[i]). \overline{re[i-1]}$
  $=(rel0[i]+rel1[i]).\overline{re[i-1]}+ carry[i]. re[i-1]$
  $+(rel0[i]+rel1[i]).re[i-1].$
  $= (rel0[i]+rel1[i])+ carry[i]. \overline{re[i-1]}$

# Combining the re[i] with Carry[i]

- Taking into consideration the right_edge signal
  For **AND** type relation:
  carry_and[i+1]:
  = rel0[i].rel1[i].re[i-1]+ rel0[i].rel1[i].carry[i]. $\overline{re[i-1]}$
  = rel0[i].rel1[i].(re[i-1]+ carry[i]. $\overline{re[i-1]}$ )
  = rel0[i].rel1[i].re[i-1]+ rel0[i].rel1[i]. carry[i].

# Combining the carry_and[i+1] with carry_or[i+1]

- We know the **carry[i+1]**
  = carry_and[i+1] For **AND** type of cube operation.
  = carry_or[i+1] **Otherwise**
  $\Rightarrow$
  carry[i+1]=carry_and[i+1].and_or+carry_or[i+1].$\overline{\text{and\_or}}$
  Where signal **and_or represents the relation of cube** type of operation.
  **and_or =1** $\Rightarrow$ Cube operation is **AND** type .
  **Otherwise** $\Rightarrow$ Cube operation is **OR** type.

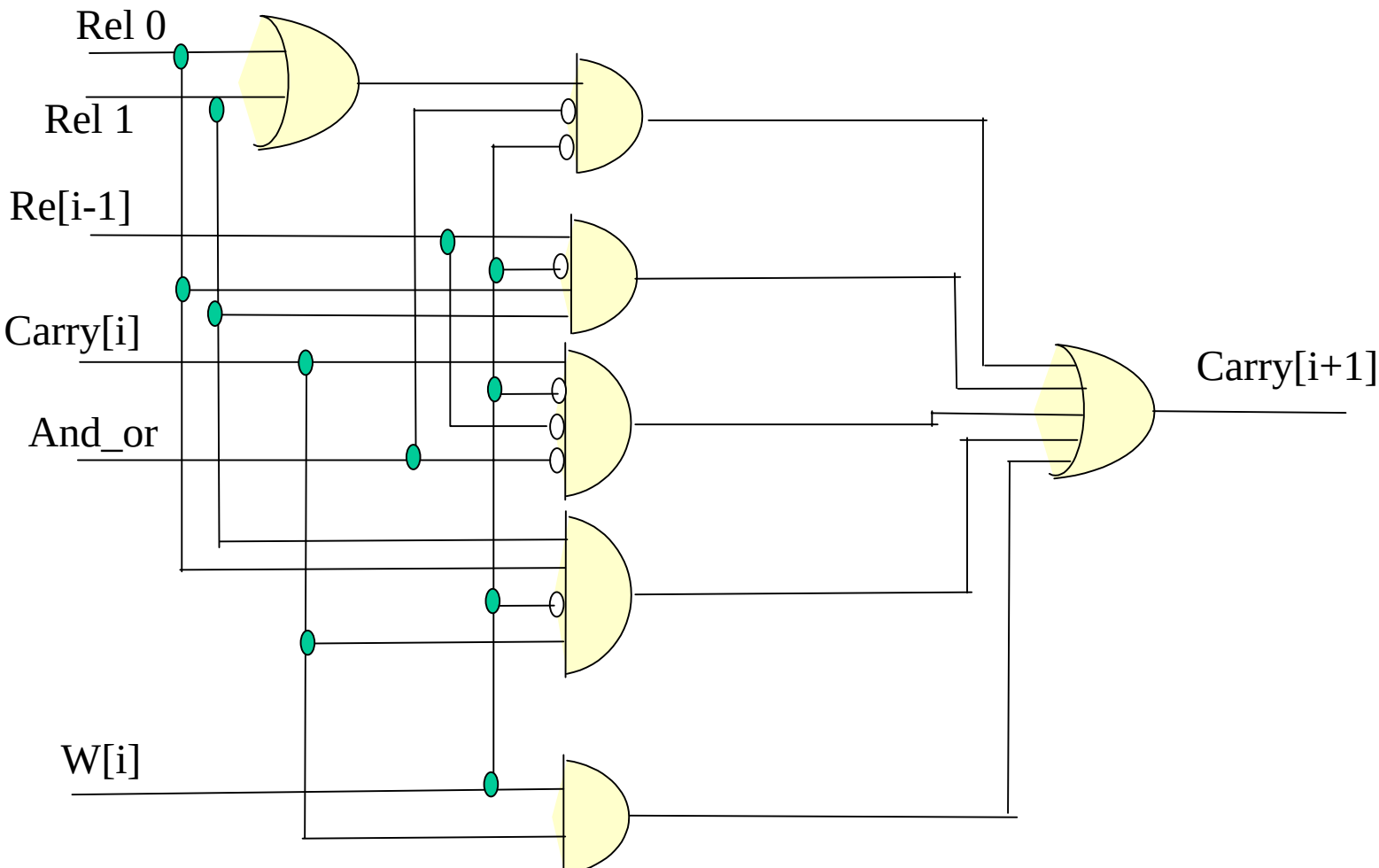
# General Expression for Carry[i+1] taking into consideration re[i]

- **carry[i+1]**
  = carry_and[i+1].and_or + carry_or[i+1] $\overline{\text{.and\_or}}$
  = carry_and[i+1] + carry_or[i+1] $\overline{\text{.and\_or}}$
  Since Carry_or always equals 1 whenever carry_and equals 1
  =rel0[i].rel1[i].re[i-1]+ rel0[i].rel1[i]. carry[i].
  + (rel0[i]+rel1[i]). $\overline{\text{and\_or}}$ +carry[i]. $\overline{\text{re[i-1]}}$ $\overline{\text{and\_or}}$

# General Expression for Carry[i+1] considering re[i] and w[i]

- **carry[i+1]=**
  **=** $\overline{\textbf{w[i]}}$**(**rel0[i].rel1[i].re[i-1]+ rel0[i].rel1[i]. carry[i]+ (rel0[i]+rel1[i]). $\overline{\text{and\_or}}$ +carry[i]. $\overline{\text{re[i-1]}}$.$\overline{\text{and\_or}}$ )
  +w[i].carry[i]

# Gate Level realization of carry[i+1]



# Combining the re[i] with Conf[i]

- For Right_edge signal **re[i]:**
  re[i-1] = 1 $\Rightarrow$ IT[i] is **first IT** of a variable .
  re[i] = 1 $\Rightarrow$ IT[i] is **last IT** of a variable General Expression for **conf:**
  conf[i-1] = carry[i+1] if IT[i] is the **last IT** of a variable.
- = conf[i] otherwise.

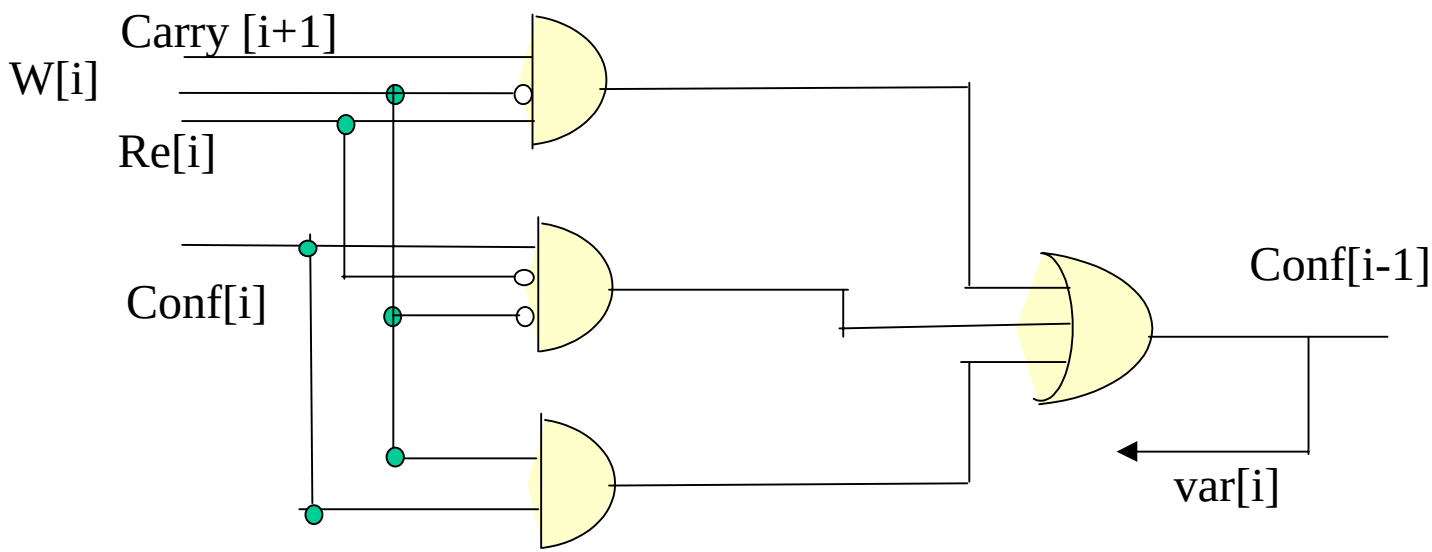
  **Combining** the above two signals:
  conf[i-1]= conf[i].rel[i]+ carry[i+1].re[i]

# General Expression for Conf[i+1] considering re[i] and w[i]

- We know that if
  W[i]=1---------- IT[i] is **not used**(IT is **Transparent**)
  W[i]=0---------- IT[i] is **used**
  conf[i-1]=[conf[i].rel[i]+carry[i+1].re[i]]$\overline{w[i]}$ + w[i].conf[i]

# Gate Level realization of conf[i-1]

Gate Level realization



# Disadvantage of Iterative Network

- The disadvantage of the iterative network is that the propagation signal must propagrate through a large number of cells
- $\Rightarrow$ The response time is longer

  $\Rightarrow$ There is a delay of propagation signal next reaching the first special variable.
- The delay is $T_{propagation}$ mathematically $T_{propagation} = t_{FF} + k.t_c$

  $\Rightarrow K\Uparrow$ delay $\Uparrow$ where

  $t_c$:The worst case delay of the combinational logic block

  C. k: Number of cells the propagation signal goes through.
- $t_{FF}$ : The time delay of FF used in sequential circuit.

# The Signal Ready

- The signal ready tells the OCU whether the ILU is ready or not then the OCU generates the request signal only when the ILU is ready.
- We introduced here a signal subready[i] which is generated at the cell that represent a special variable and recieves the next signal.

  $\Rightarrow$ subready[i]=request.next[i].var[i]

  Any of subready[i] signals become one means CCM is ready to output the result cube.

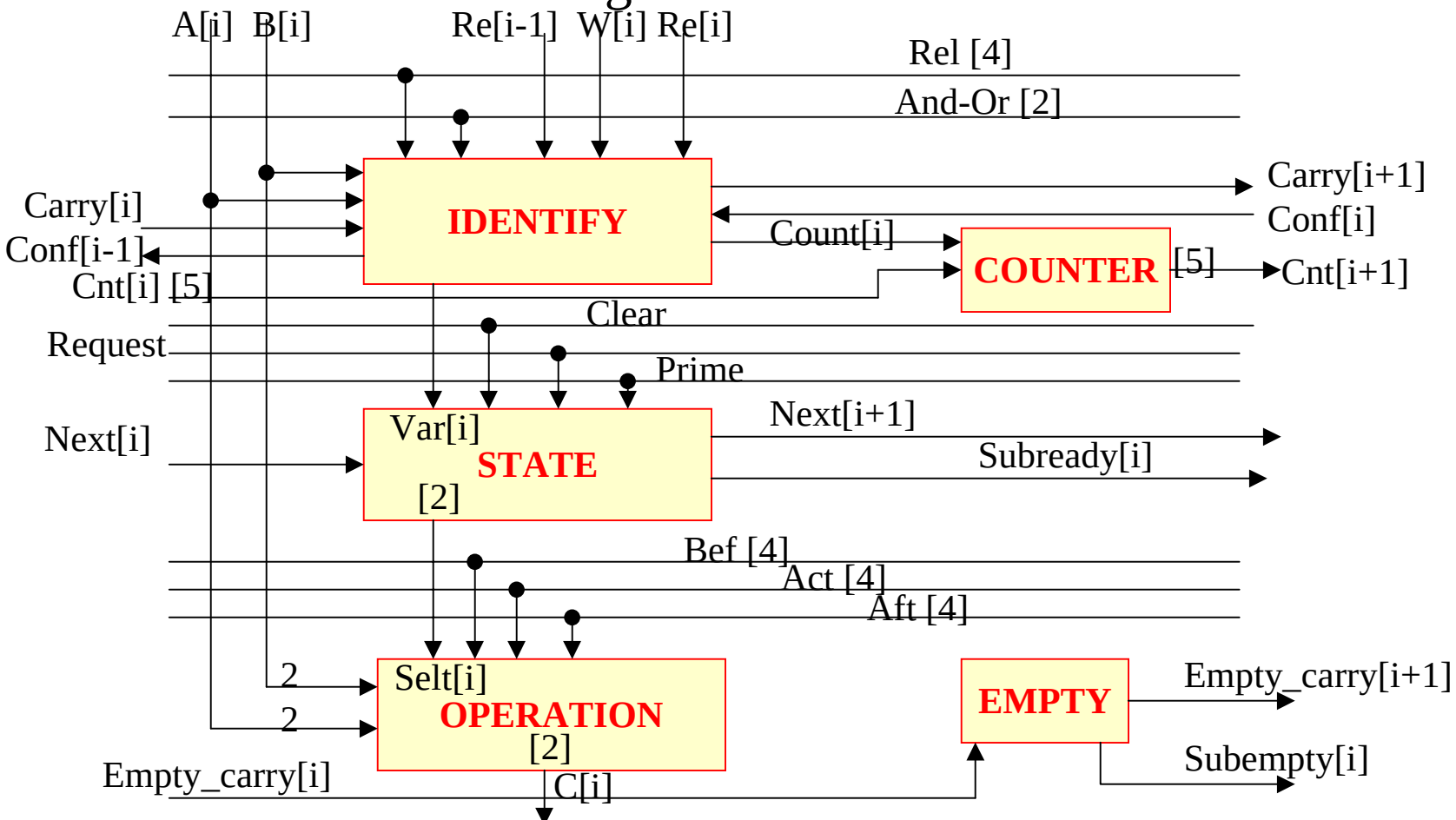
  $\Rightarrow$ ready = subready[1]+ subready[2]+------+ subready[n]

# Realization of FSM for complex combinational cube operation

- Originally the FSM was designed for sequential cube operations.
- But there are active variables in complex combinational cube operation where all the active variables are taken at a time.
- Therefore to select sequential or complex combinational cube operation we use two separate input variables (selt1[i], selt0[i])

# Detailed block diagram of a Iterative cell(IT)

- The detailed block diagram:



# Sources

- Seyda Mohsina
- Qihong Chen
- David Foote