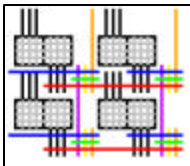


Universal Programming Organization

André DeHon
<andre@cs.caltech.edu>

Tuesday, June 18, 2002

Key Points

- Can express any function as a set of logic equations
  - With state: any Finite Automata
- Can implement any logical equation with gates
  - With only 2-input **nor** gates
- Can build substrates which can be *programmed* to perform any function
  - PLA, LUT
  - Spatial and temporal composition thereof

Logical Equations

Familiar with Basic Logic

- AND: true if **all** inputs are true
  - O=and(a,b)
  - O=a*b
  - O=a*b*c*d
- OR: true if **any** inputs are true
  - O=or(a,b)
  - O=a+b
- NOT: true if input false; false if input true
  - O=not(a)
  - O=!a

Additional Logic

- NOR = Not OR
  - O=nor(a,b)=not(or(a,b))
- NAND = Not AND
  - O=nand(a,b)=not(and(a,b))
- XOR = exactly one input true
  - O=XOR(a,b)=a*!b+!a*b
  - Multi-input defined as cascaded 2 input
    - XOR(a,b,c)=XOR(XOR(a,b),c)

Boolean Expressions

- Write expressions
  - O=a*b*c+!b*!a+a*!c
- Distribute
  - O=a*(b+c) = a*b+a*c
- DeMorgan's Equivalence
  - O=!(a+b)=!a*!b
  - O=!(a*b)=!a+!b

Any Function

- Function: (Mathematical definition)
  - each input vector
    - (set of truth values for input symbols)
  - maps to exactly one output vector
  - output is deterministic
  - output depends only on the input vector

Sketch of "Any Function"

- Consider binary output function:
  - will have either true or false (1 or 0) for each input vector
  - can pick out each input vector with an **and**
    - *e.g.* a b c d having values 1 0 1 1
      - we would pick this out with the term:
      - and(a,!b,c,d)=a*!b*c*d
  - **or** together all such terms for true outputs

Any Function (cont.)

- *e.g.* if want true when a b c d has values:
  - 1 0 1 1,
  - 0 1 0 0,
  - or 0 0 1 1
- we get:
  - O=a*!b*c*d+!a*b*!c*!d+!a*!b*c*d

Not Simplest

- Result is not necessarily the smallest way to express
- *e.g.* if want true when a b c d has values:
  - 1 0 1 1, 0 1 0 0, or 0 0 1 1
- we got:
  - O=a*!b*c*d+!a*b*!c*!d+!a*!b*c*d
- Also
  - O=!b*c*d+!a*b*!c*d

Any Function Sketch

- Previous example showed single output bit
- For multiple output bit functions
  - simply write one equation for each output bit

Logic Model

- We can express any function as a set of logic equations

Gates

Gates as Building Blocks

- Typical building block is small gates
  - Finite number of inputs (1—4)
- Gate is a small logic expression itself
  - and2(a,b)=a*b
  - xor2(a,b)=a*!b+!a*b
- To implement logic express as gates
  - Simply factor into gates

Factor into gates

- Can always factor large **and** or **or** into small gates
  - a*b*c*d*e*f
  - and(a,and(b,and(c,and(d,and(e,f)))))
  - and(and(and(a,b),and(c,d)),and(e,f))
- Sufficient to cover our expressions for any function
- Two-input gates sufficient to implement all logical expressions

Minimal Gate Set

- Only need one kind of gate
  - If it has the right properties
- Consider **nor:**
  - O=!a=nor(a,a)
  - O=and(a,b)=nor((nor(a,a),nor(b,b))
    - =nor(!a,!b)=!(!a+!b)=a*b
  - O=a+b=nor((nor(a,b),nor(a,b))
- Can implement all logical expressions using only 2-input **nor** gates

Universal Gate Model

- Can implement any logical equation with finite-input gates
  - With only 2-input **nor** gates

Programmable Building Blocks

Specific→Programmable

- Previously argued could build any particular boolean function
- Can also build circuits which can perform more than one function
  - ultimately, any function

Simple Version

- prog2gate(a,b,s)=s*(a*b)+!s*(a+b)
- If s is **true**
  - prog2gate1 performs: a*b
- Otherwise
  - prog2gate1 performs: a+b

Programmable gate

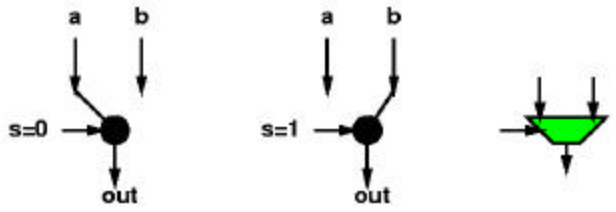
- prog2gate1
  - Is programmable
  - Can make it perform like either gate
  - By performing one operation or the other

Extended

- Prog2gate(a,b,s0,s1,s2)=
  ((s0*s1*s2)* (a*b)+
  (s0*s1*!s2)*!(a*b)+
  (s0*!s1*s2)* (a+b)+
  (s0*!s1*!s2)*!(a+b)+
  (!s0*s1*s2)* (a*!b+!a*b)+
  (!s0*s1*!s2)* (a*b+!a*!b))

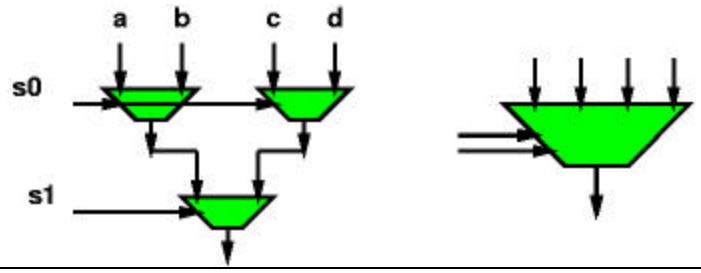
Programmable Wiring

- Mux(a,b,s)=!s*a + s*b
- Passes a or b to output based on s



Can build Larger

- mux4(a,b,c,d,s0,s1)=
  mux2 (mux2(a,b,s0),
  mux2(c,d,s0),s1)



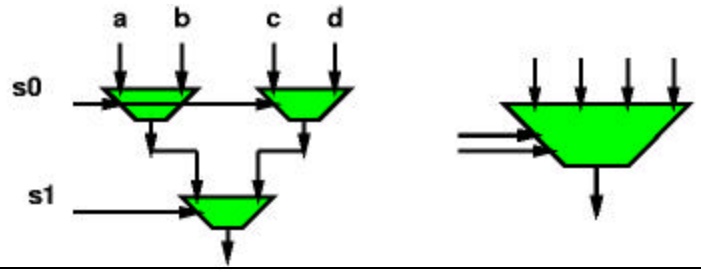
Can also use for logic

- and2(a,b)=mux4(0,0,0,1,a,b)
- or2(a,b)=mux4(0,1,1,1,a,b)
- nand2(a,b)=mux4(1,1,1,0,a,b)

- Just by routing "data" into this mux4,
  - Can select **any** two input function

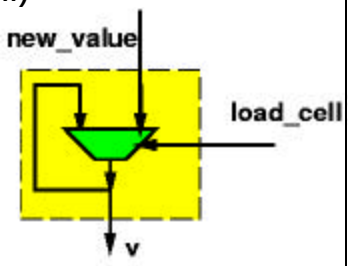
Implication

- Just based on some extra bits
  - Can "program" how wired up
  - Can "program" which gate behavior



Memory Cell

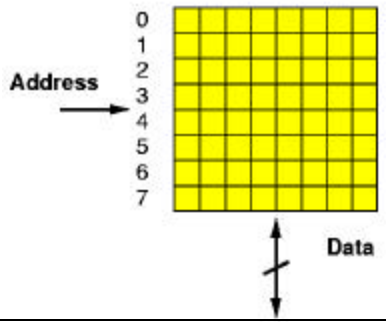
- Memory Cell
  - Gate which remembers a value
- Can build out of gates we have seen
- v=mux2(v,new_value,load_cell)



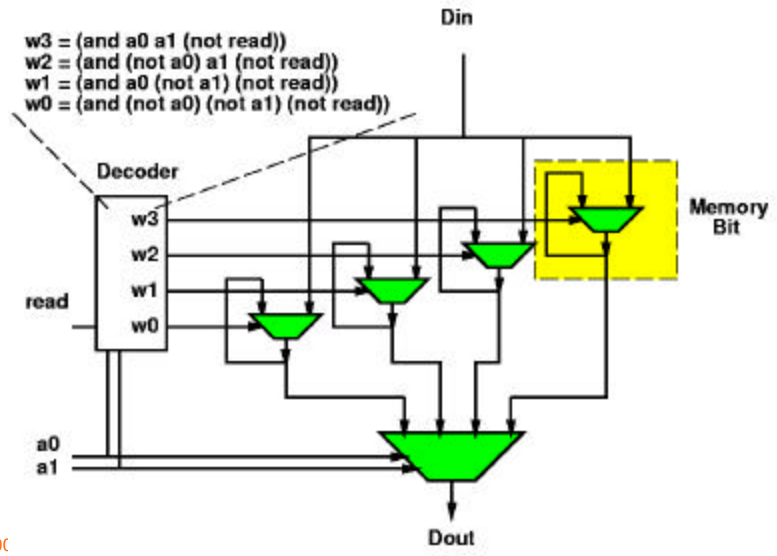
- usually a more direct/efficient implementation)

Memory Array

- Often build into array
- A Memory:
  - Series of locations
  - Can write values into
  - Read values from

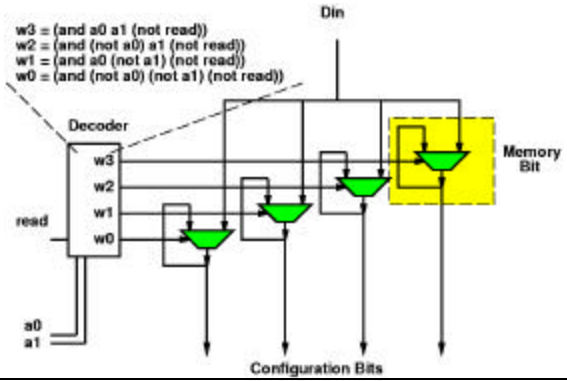


Could build Memory w/ Muxes



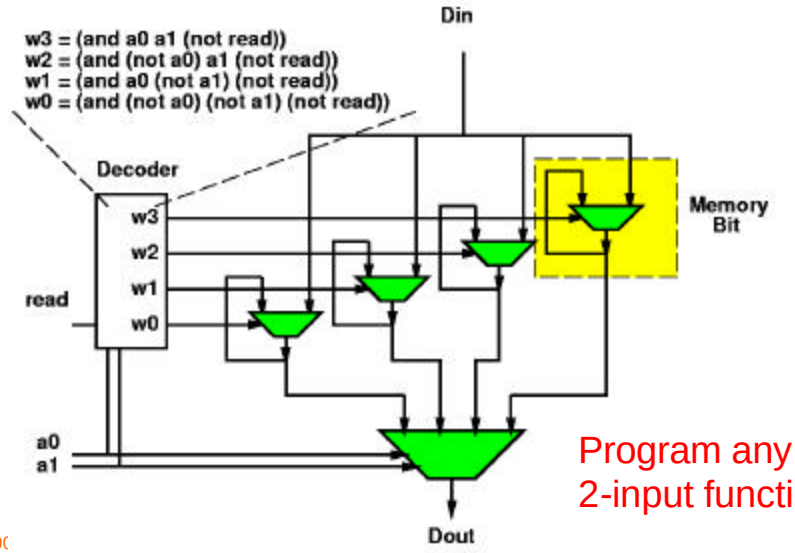
Programmable Gates

- Can use the memory cells to hold the personalization bits for programmable gates



Look-Up Table Logic (LUTs)

Lookup Table is a Programmable Gate



Program any 2-input function

K-LUT

- LUT = Look-Up Table
- K-input LUT
  - Can generalize memory to arbitrary k
  - Can be programmed to implement any function of k-inputs
- …but requires 2^K memory bits to do so

Exponential Size Problem

- LUT – will need exponentially more gates to handle more inputs
- But many functions only need linearly more gates
  - O=a*b*c*d*e*f….
  - Can be implemented with linear gates

Programmable Array Logic (PLAs)

PLA

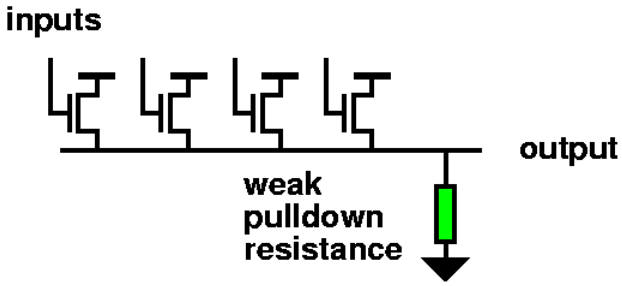
- Directly implement flat (two-level) logic
  - O=a*b*c*d + !a*b*!d + b*!c*d
- Exploit substrate properties allow wired-OR

Wired-**or**

- Connect series of inputs to wire
- Any of the inputs can drive the wire high

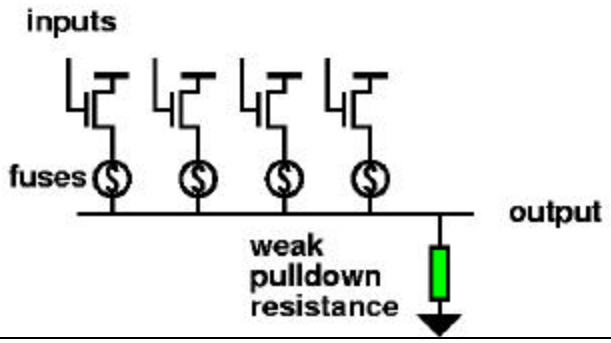
Wired-**or**

- Obvious Implementation with Transistors



Programmable Wired-**or**

- Use some memory function to programmable connect (disconnect) wires to OR
- Fuse:



Programmable Wired-**or**

- Gate-memory model

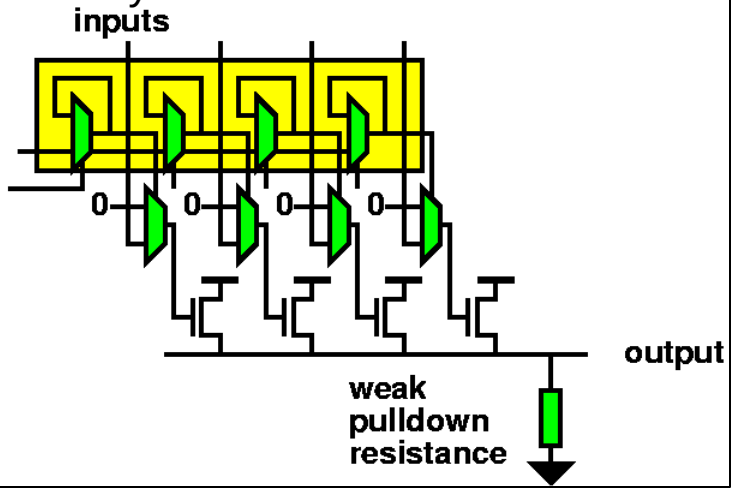
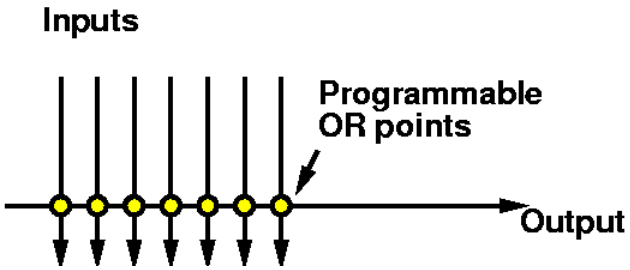
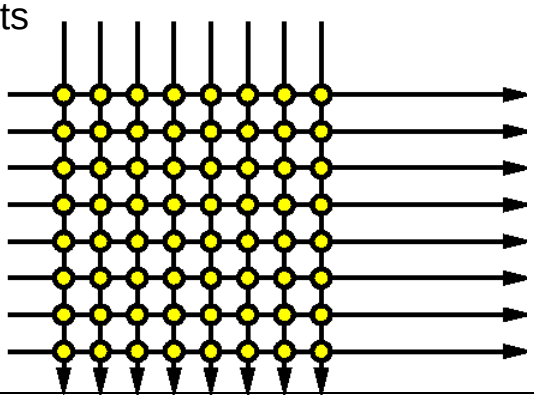


Diagram Wired-**or**



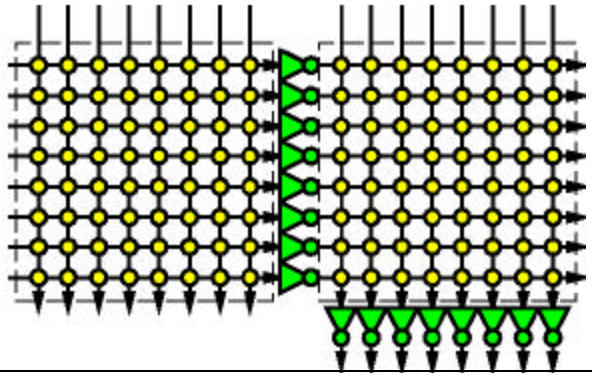
Wired-**or** array

- Build into array
  - Compute many different **or** functions from set of inputs



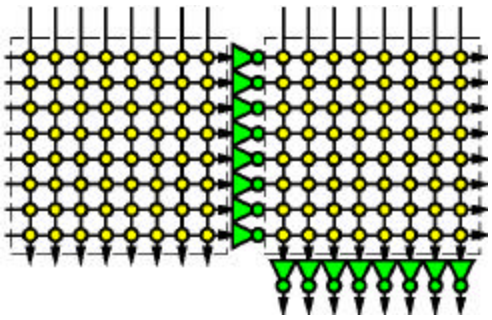
Combined **or**-arrays to PLA

- Combine two or (**nor**) arrays to produce PLA (**and**-**or** array)



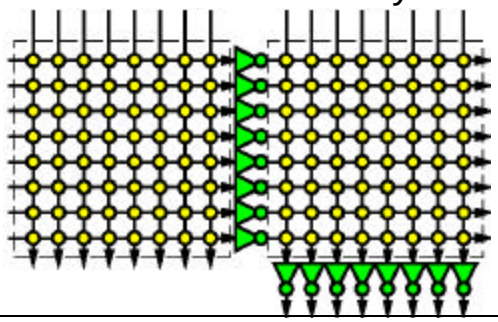
PLA

- Can implement each **and** on single line in first array
- Can implement each **or** on single line in second array



PLA

- Efficiency questions:
  - Each **and**/**or** is linear in total number of potential inputs (not actual)
  - How many product terms between arrays?



PLA Product Terms

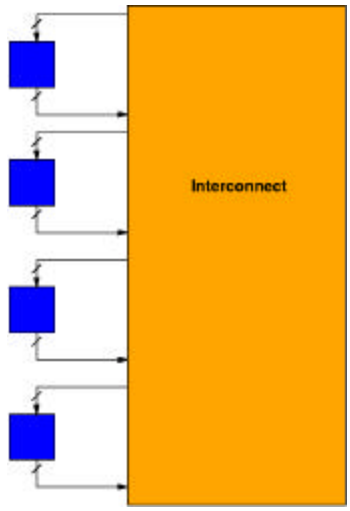
- Can be exponential in number of inputs
- *E.g.* n-input **xor** (parity function)
  - When flatten to two-level logic, requires exponential product terms
  - a*!b+!a*b
  - a*!b*!c+!a*b*!c+!a*!b*c+a*b*c
- …and shows up in important functions
  - Like addition…

Spatial Composition

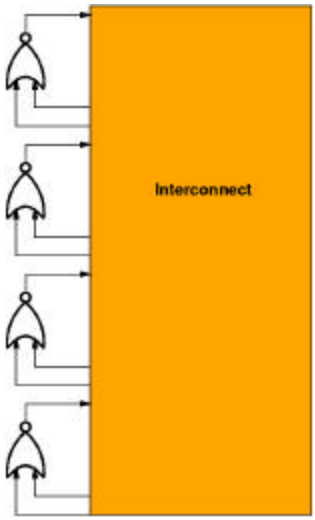
Decomposing

- Large PLAs, LUTs
  - Can be arbitrarily inefficient on some problems
- **But** gates seem efficient
- **Try**: finding a way to hook up
  - Small PLAs, LUTs, nor-gates

Logic Blocks w/ Interconnect

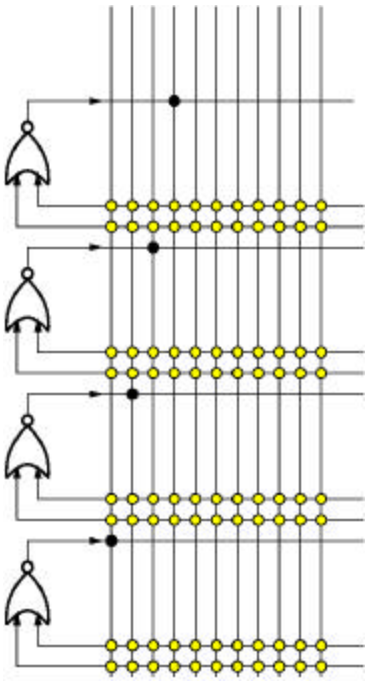


Logic Blocks w/ Interconnect



Crossbar

- Use wired-or like property like PLA array
  - Implement efficiently



Crossbar Efficiency

- Takes N^2 area
  - Better than exponential!
  - But seems expensive
- Will return
  - Talk about costs (tomorrow)
  - Talk about interconnect (following day)

Spatial Composition

- Can implement any function programmably
  - Divide into small functions
    - Use small PLAs/LUTs to implement gates
  - OR Transform into **nor** gates
    - Other fixed/universal logic functions
  - Connect up with programmable interconnect

Temporal Composition

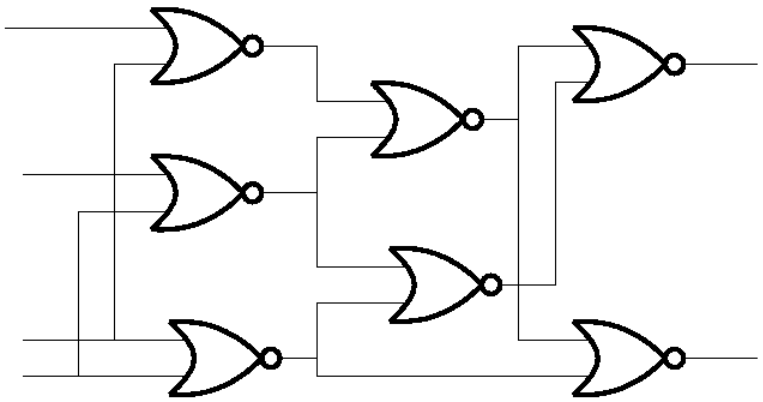
Temporal

- Don't have to implement all the gates at once
- Can use one gate over time

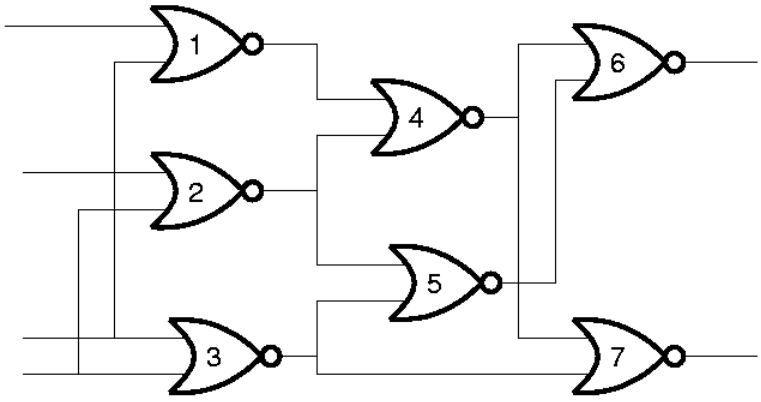
Temporal Decomposition

- Take Set of gates
- Sort topologically
  - All predecessors before successors
- Give a unique number to each gate
  - Hold value of its outputs
- Use a memory to hold the gate values
- Sequence through gates

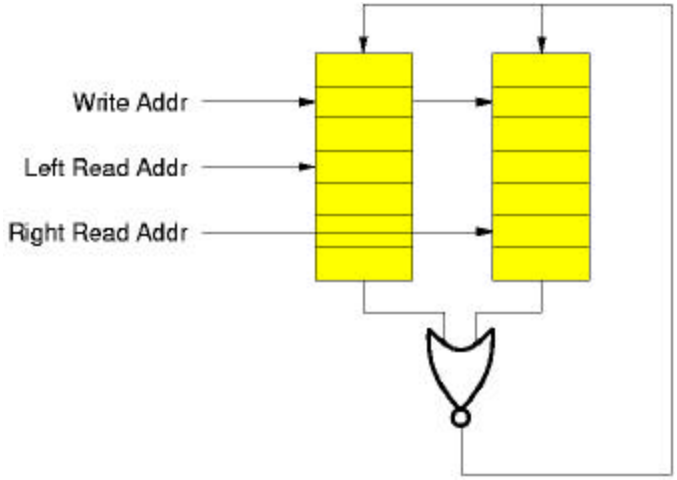
Example Logic



Numbered Gates



nor2 Memory/Datapath



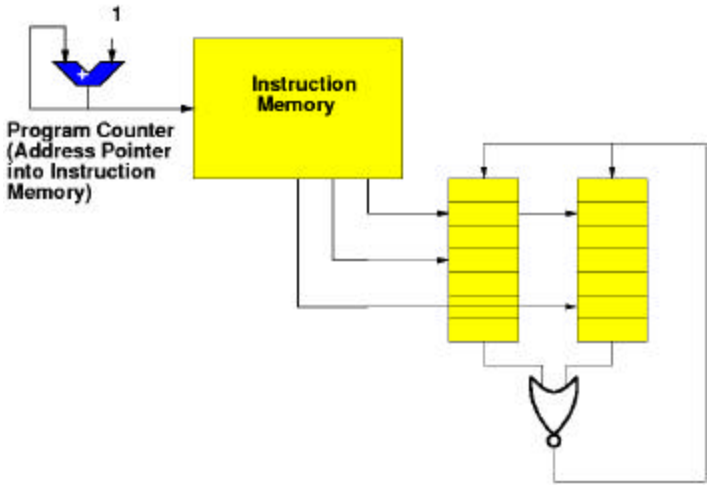
Programming?

- How do we program this network?

Programming?

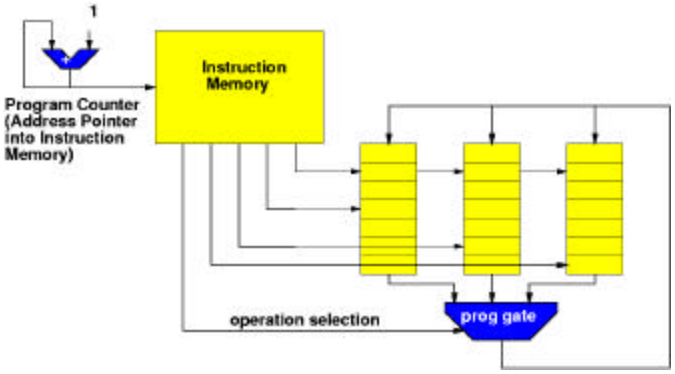
- Program gates
  - Tell each gate where to get its input
    - Tell gate n where its two inputs come from
    - Specify the memory location for the output of the associated gate
  - Each gate specified with two addresses
    - This is the *instruction* for the gate

Temporal Gate Architecture



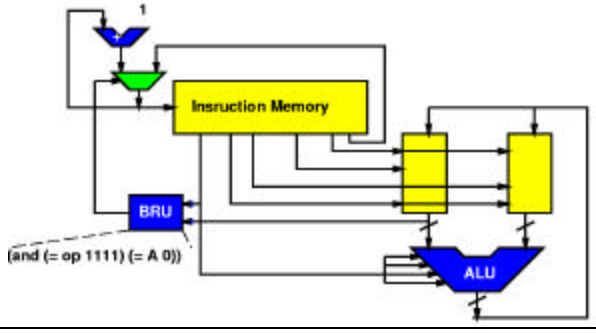
Programmable Variation

- Can use programmable gate in place of **nor** gate



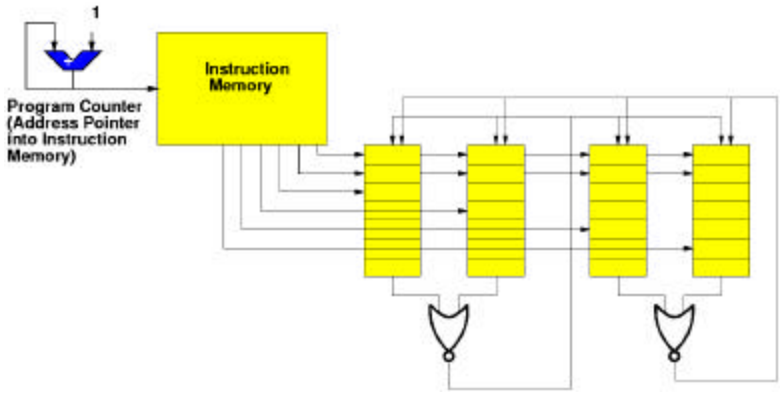
Processor

- Conventional Processor is a variant of this
  - Uses an ALU for the programmable gate



Temporal-Spatial Variation

- Can have any number of gates
  - Extreme is spatial



Temporal Composition

- Can implement any function programmably
  - Divide into small functions
    - Use small PLAs/LUTs to implement gates
  - OR Transform into **nor** gates
    - Other fixed/universal logic functions
  - Use single (or small number) such gates
  - Use memory to store function outputs
  - Connect up in time by sequencing through operations

Wrapup

Key Points

- Can express any function as a set of logic equations
  - With state: any Finite Automata
- Can implement any logical equation with gates
  - With only 2-input **nor** gates
- Can build substrates which can be *programmed* to perform any function
  - PLA, LUT
  - Spatial and temporal composition thereof

Coming Soon…

- Not clear
  - Which preferred, when?
  - Right size for PLAs, LUTs…
- Depends on substrate costs
  - Discuss next time
- Need to understand interconnect costs
  - Discuss following time