Hardware/Software Partitioning for SoCs

EECE 579 - Advanced Topics in VLSI Design

Spring 2009

Brad Quinton

Goals of this Lecture

- Automatic hardware/software partitioning is **big topic**...

- In this lecture, I will try to:

  - explain the problem that we are trying to solve,
  - outline a basic strategy to attack the problem,
  - highlight the big challenges,
  - provide enough background to appreciate the assigned paper

Outline

1. Hardware/Software Partitioning in an SoC Context
2. A Procedure for Automatic Hardware/Software Partitioning
3. Control and Data Flow Graphs
4. Allocation and Scheduling
5. Algorithms
6. Summary
7. Introduction to: “Hardware-Software Cosynthesis for Microcontrollers”

Hardware/Software Partitioning in an SoC Context

SoCs

- As we have discussed previously, the problem of hardware/software partitioning is particularly relevant to Systems-on-Chip (SoCs):
  - the final SoC will likely include both **software** and **hardware** any case
  - the **software resources** are under the SoC designers control
  - the **hardware resources** are under the SoC designers control
  - the **interface** between the software and hardware is under the SoC designers control

SoCs

- As we have discussed previously, the problem of hardware/software partitioning is particularly relevant to Systems-on-Chip (SoCs):
  - the final SoC will likely include both **software** and **hardware** any case
  - the **software resources** are under the SoC designers control
  - the **hardware resources** are under the SoC designers control
  - the **interface** between the software and hardware is under the SoC designers control
- Too much flexibility....?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?
- System bus? NoC? DMA Controller?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?
- System bus? NoC? DMA Controller?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?
- System bus? NoC? DMA Controller?

Power?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?
- System bus? NoC? DMA Controller?

Power? Device Cost?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?
- System bus? NoC? DMA Controller?

Power? Device Cost? Development Cost?

Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?
- System bus? NoC? DMA Controller?

Power? Device Cost? Development Cost? Features?

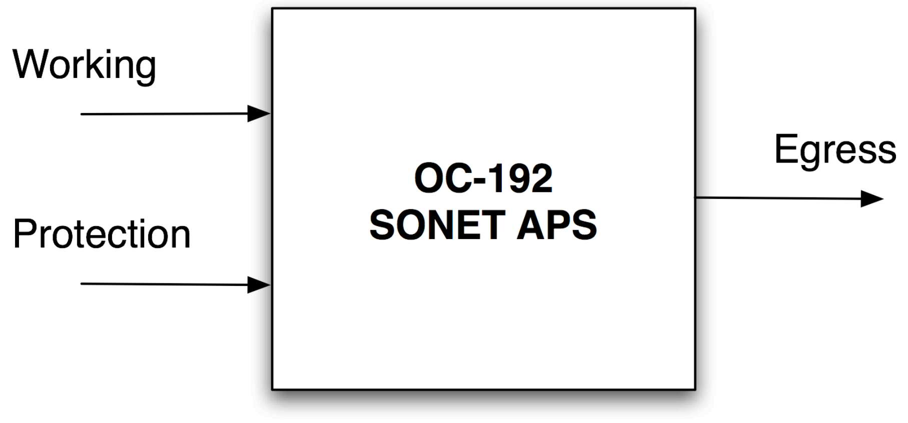
Very Large Design Space

- Given this flexibility the design space is **extremely large**

- How many processors? What kind?
- What frequency should they operate at?
- How much memory? How much bandwidth?
- How much application specific hardware?
- How frequency should it run at?
- System bus? NoC? DMA Controller?

Power? Device Cost? Development Cost? Features? Markets?

Case Study: High Level System View

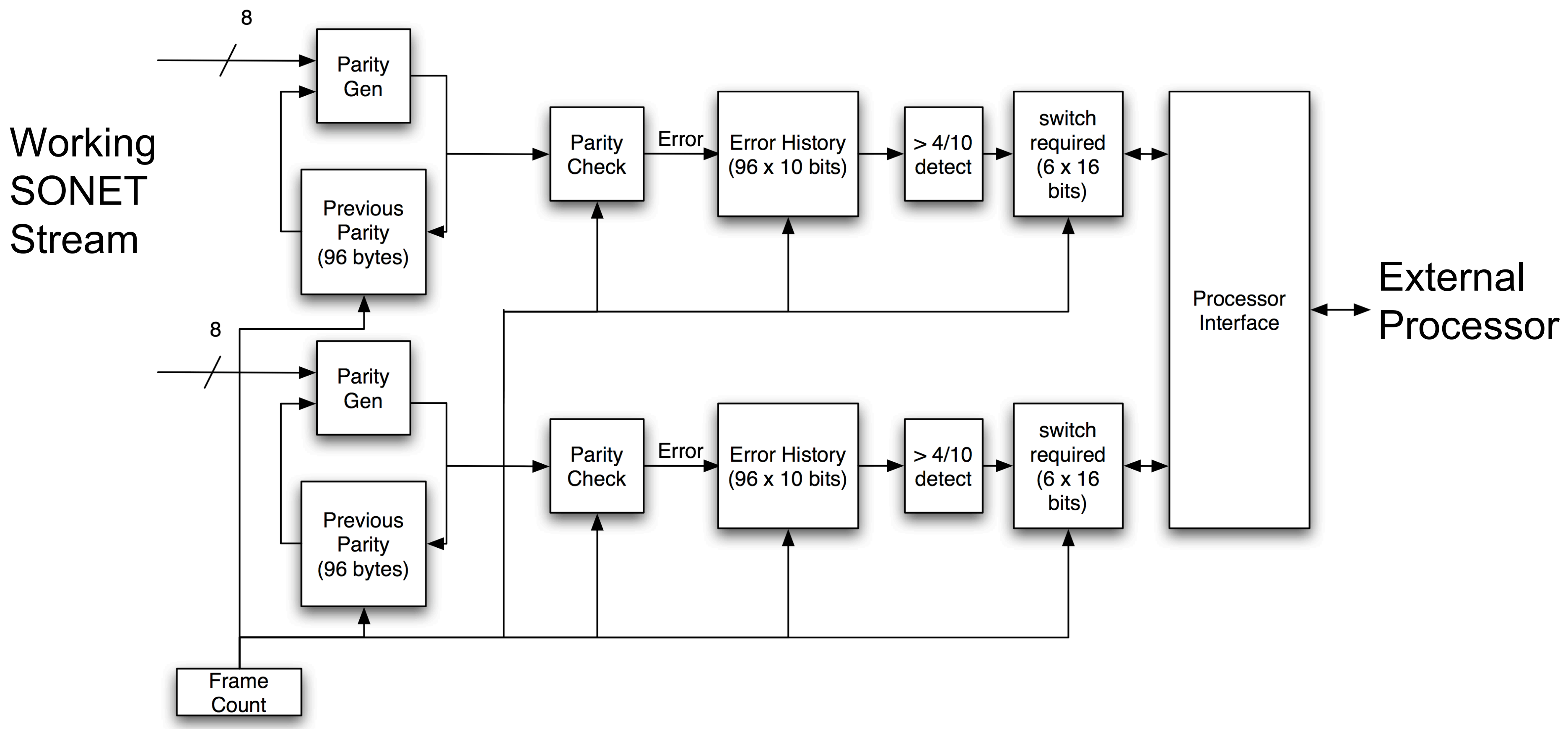


Requirement:

If the Bit Error Rate of any given *working* STS-1 channel exceeds 4 errored frames in a 10-frame sliding window, the egress channel must switch to the *protection* STS-1 within 300 μs.*

* Note: This is a simplified version of the true requirement.

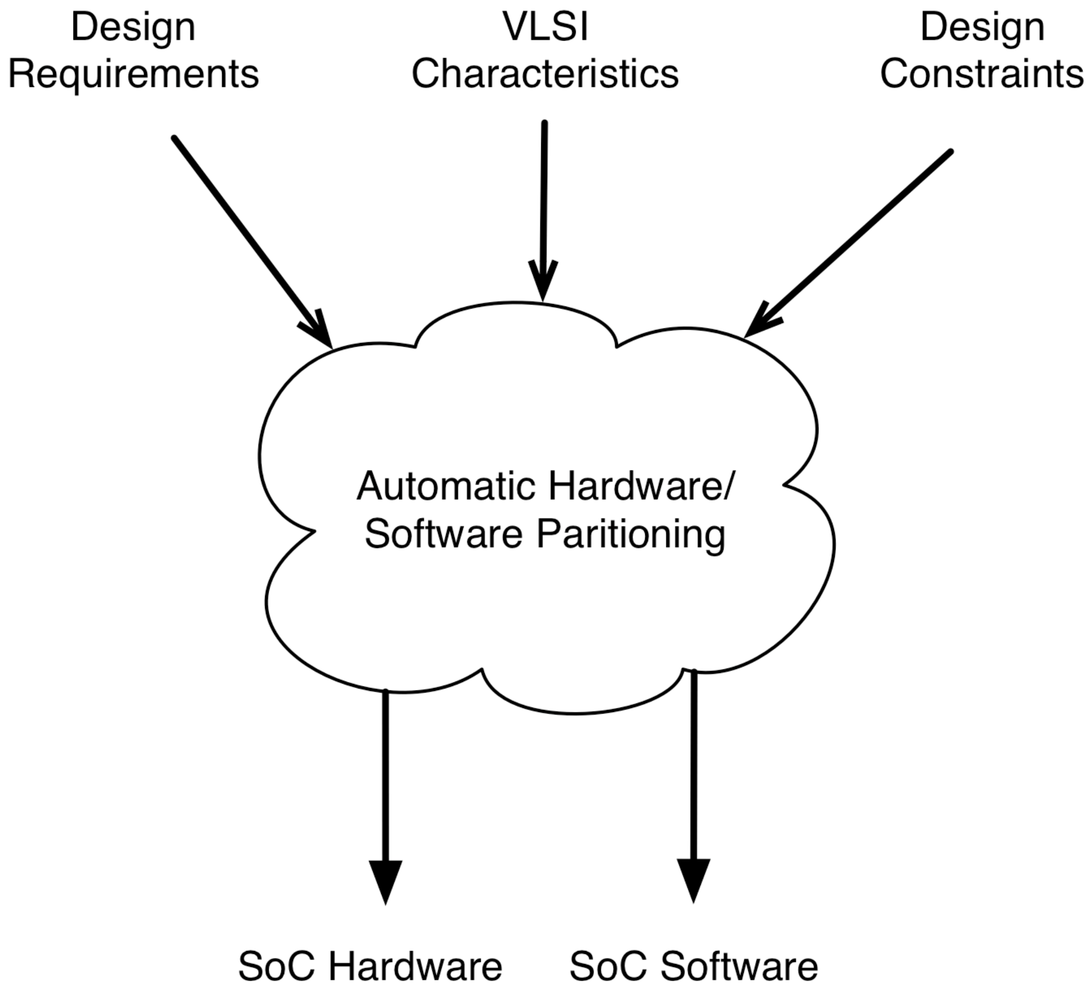
Case Study: Final Design (Hardware)


8
Parity Gen
Working SONET Stream
Previous Parity (96 bytes)
Parity Check
Error
Error History (96 x 10 bits)
> 4/10 detect
switch required (6 x 16 bits)
Processor Interface
External Processor
8
Parity Gen
Previous Parity (96 bytes)
Parity Check
Error
Error History (96 x 10 bits)
> 4/10 detect
switch required (6 x 16 bits)
Frame Count

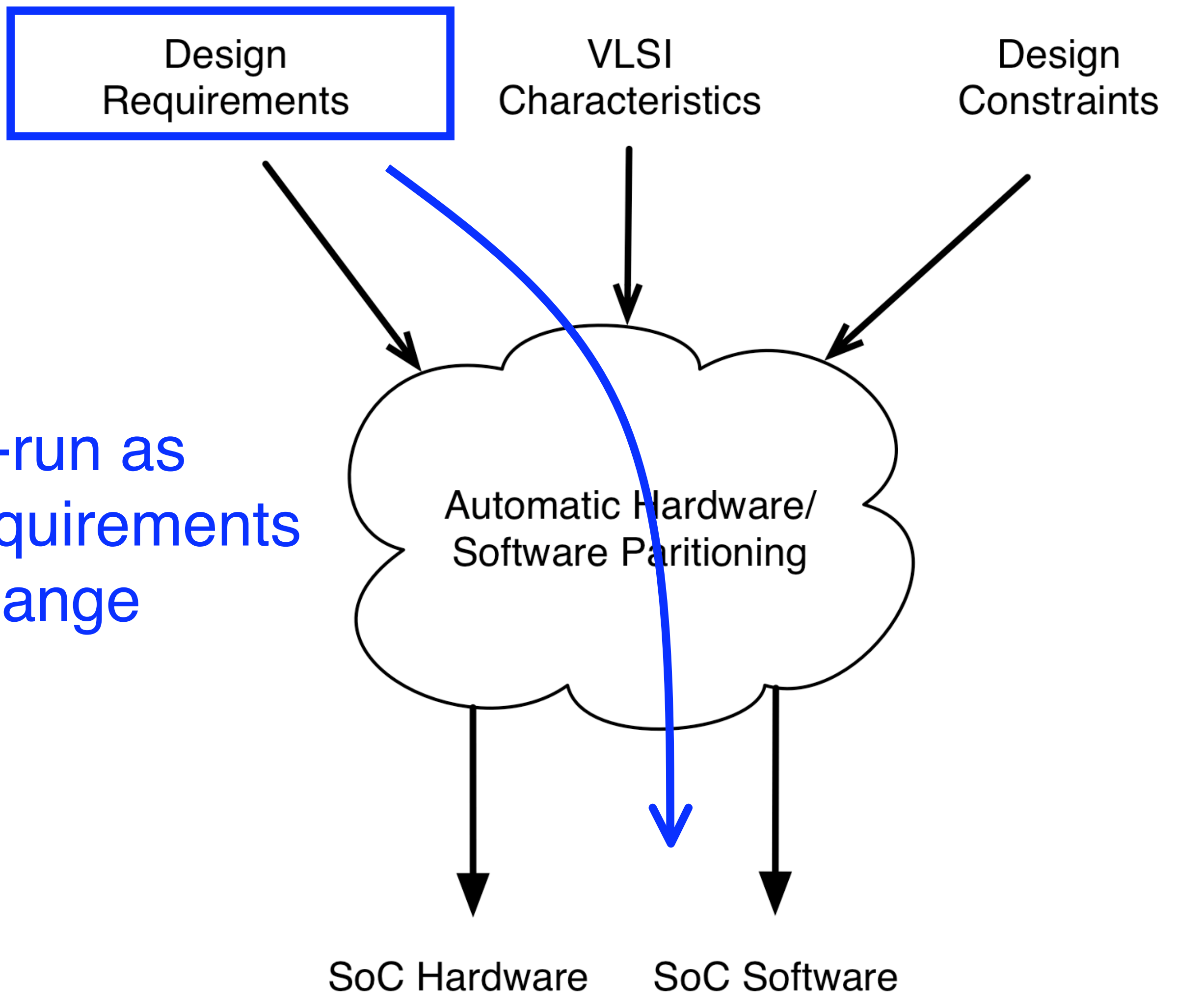
Can we automate this?

- Maybe we can get a computer to perform these trade-offs for us...
- Not only would this make our lives easier, but we might get better results!

The Dream ...


Design Requirements
VLSI Characteristics
Design Constraints
Automatic Hardware/ Software Paritioning
SoC Hardware
SoC Software

The Dream ...

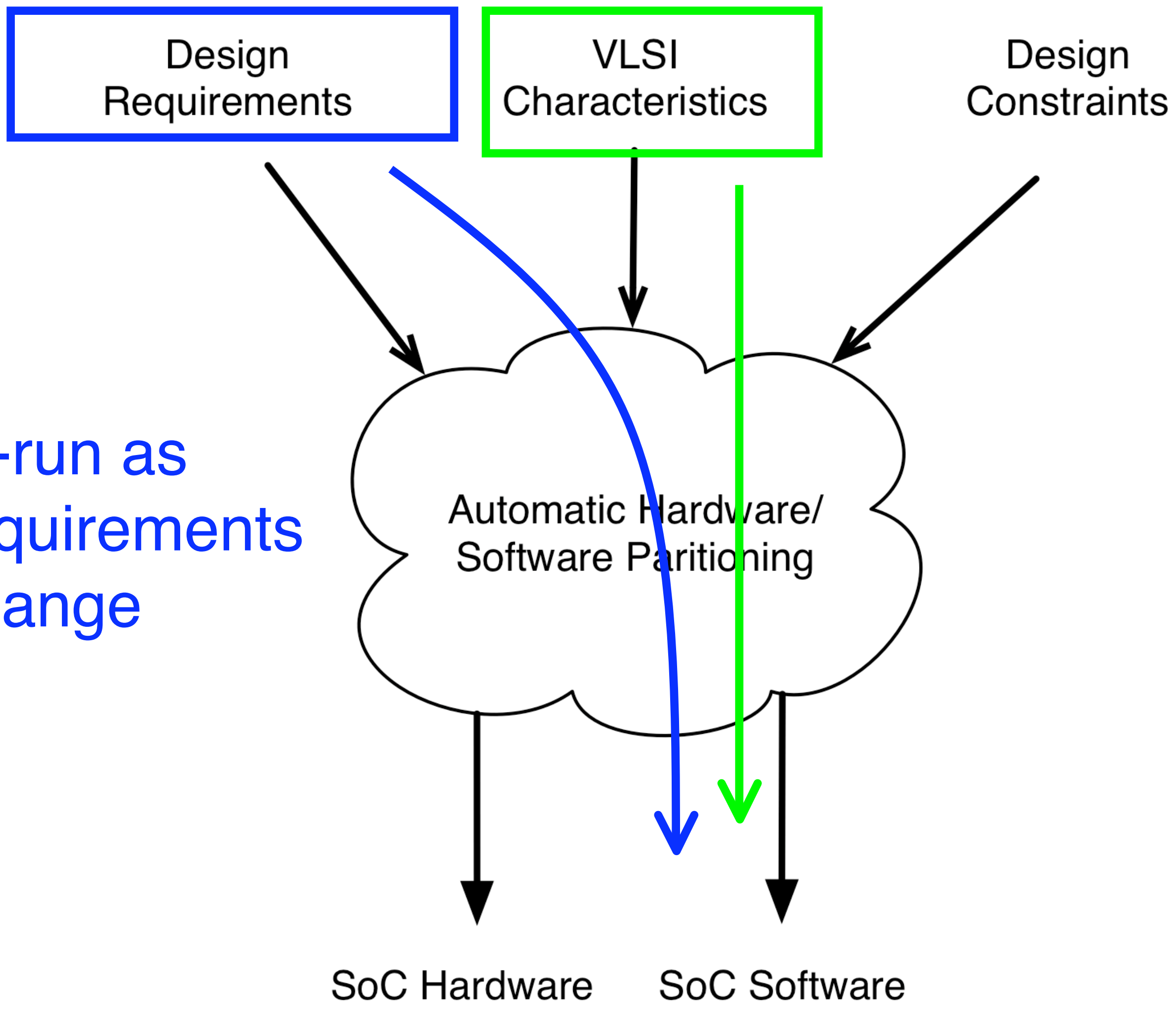


re-run as requirements change

The Dream ...

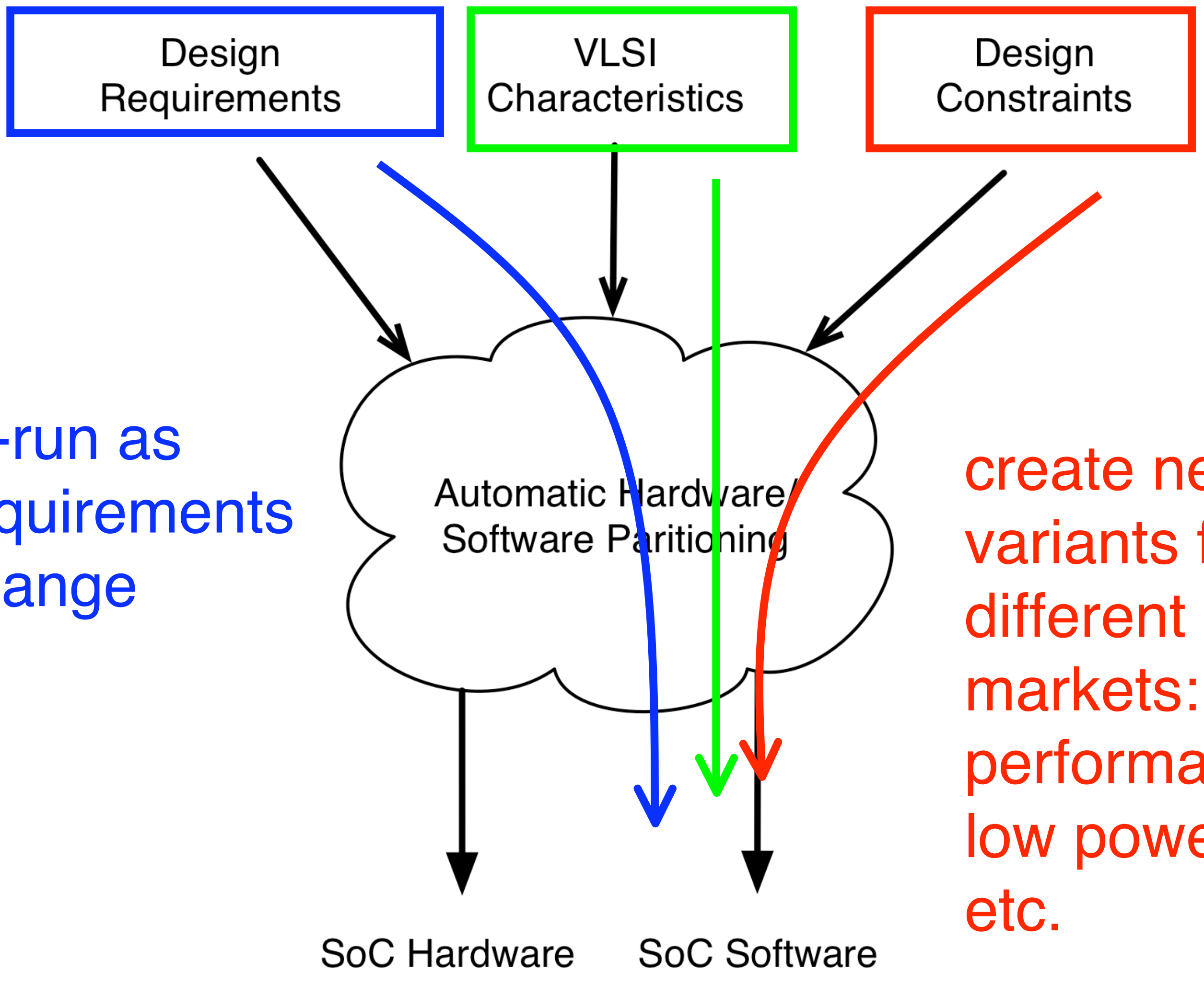
re-run for each new technology

re-run as requirements change



The Dream ...

re-run for each new technology



re-run as requirements change

create new variants for different markets: high performance, low power, etc.

Is It Possible?

- There are two highly related problems that give us hope:
  - **Software Compilation**
  - **High-level Synthesis**
- At least we have a starting point...

Software Compilers

- **Goal:** Given a high-level program (for instance C, C++, Java) generate machine code
- Long history ... earliest compiler in 1952!
- For example: gcc, Turbo C++
- Very popular, widely used, well understood

Software Compilers

- Basic idea:
  - the hardware resources are a given,
  - the problem is to **allocate** and **schedule** them efficiently

- There is lots of information about this... start with gcc webpage: http://gcc.gnu.org

- UBC Courses: CPSC 411, ...

High-level Synthesis

- **Goal:** Given a high-level hardware description (SystemVerilog, SystemC, C, VHDL) generate a RTL representation

- Much newer than software compilers: first commercial product in 1994

- For example, Synopsys Behavioral Compiler, MATLAB HDL Coder

- Still a niche product; hardware design mostly still write RTL....

High-level Synthesis

- Basic Idea:
  - Generate enough hardware to meet some timing, throughput, or other constraint
- This topic is covered in detail in EECE 583
- In fact, next weeks 583 lecture is on High-level Synthesis...

Hardware/Software Partitioning

- Hardware/Software partitioning has many of the challenges of both Software Compilation and High-level Synthesis

- ... but, it is **even harder**, as we will see!

A Procedure for Hardware/Software Partitioning

Where do we start?

Where do we start?

- Need to have a software readable representation
- We need to be able to map this to both hardware and software constructs
- Look to High-level Synthesis for the basic procedure...

Basic Procedure

1. Generate a software readable representation of the problem (for instance, a graph...)

Basic Procedure

1. Generate a software readable representation of the problem (for instance, a graph...)
2. Optimize this representation (remove redundancy, organize operations)

Basic Procedure

1. Generate a software readable representation of the problem (for instance, a graph...)
2. Optimize this representation (remove redundancy, organize operations)
3. Allocate the operations to the available resources (hardware, software)

Basic Procedure

1. Generate a software readable representation of the problem (for instance, a graph...)
2. Optimize this representation (remove redundancy, organize operations)
3. Allocate the operations to the available resources (hardware, software)
4. Schedule the utilization and interactions of the resources

Basic Procedure

1. Generate a software readable representation of the problem (for instance, a graph...)
2. Optimize this representation (remove redundancy, organize operations)
3. Allocate the operations to the available resources (hardware, software)
4. Schedule the utilization and interactions of the resources
5. Bind the operations to the resources

Basic Procedure

1. Generate a software readable representation of the problem (for instance, a graph...)
2. Optimize this representation (remove redundancy, organize operations)
3. Allocate the operations to the available resources (hardware, software)
4. Schedule the utilization and interactions of the resources
5. Bind the operations to the resources
6. Generate the hardware and software representations (C, Verilog...)

Control and Data Flow Graphs

Control and Data Flow Graphs

- We can use control and data flow graphs (CDFGs) to represent the functional behavior of our SoC in a software readable form

- CDFGs capture all of the control and data flow of the device (i.e. they are a complete representation of the behaviour)

- Usually generated manually, although there is some work on automatic generation....

Data Flow Graph

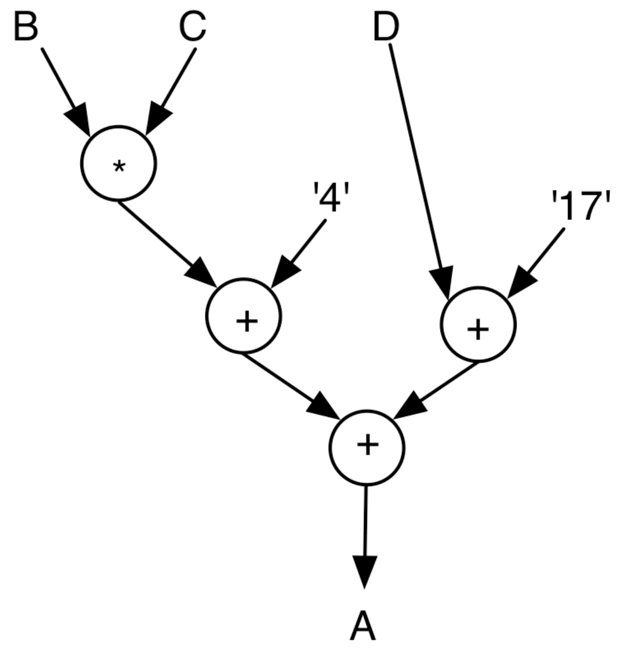
```
E := B * C + 4;
F := D + 17;
A := E + F;
```

Data Flow Graph

```
E := B * C + 4;
F := D + 17;
A := E + F;
```

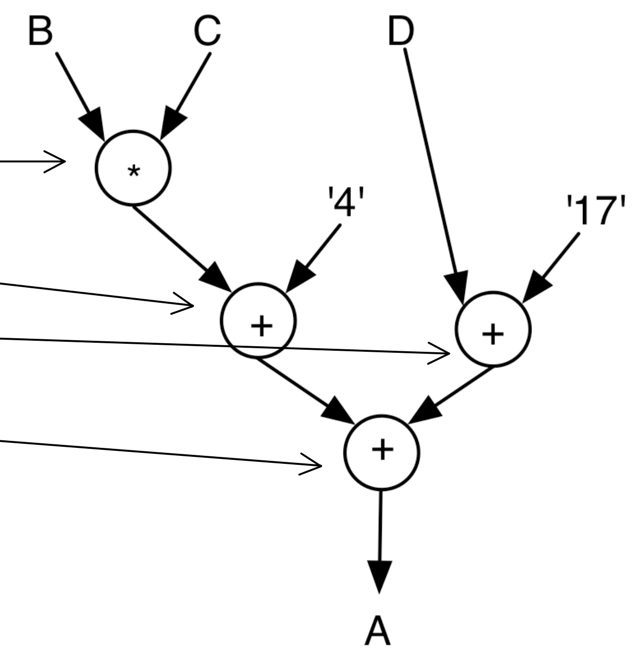
Data Flow Graph

```
E := B * C + 4;
F := D + 17;
A := E + F;
```



Data Flow Graph

```
E := B * C + 4;
F := D + 17;
A := E + F;
```



What about control info?

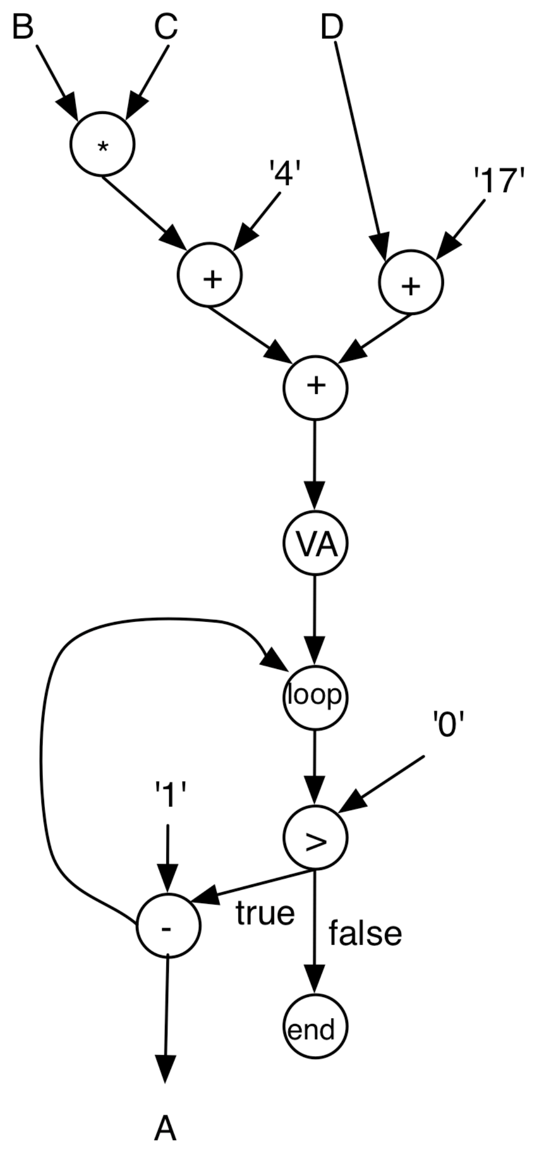
- Data flow graphs only capture part of the story...
- We need to capture control flow as well.

Control and Data Flow Graph

```
E := B * C + 4;
F := D + 17;
A := E + F;
while (A > 0) loop
     A := A - 1;
end loop;
```

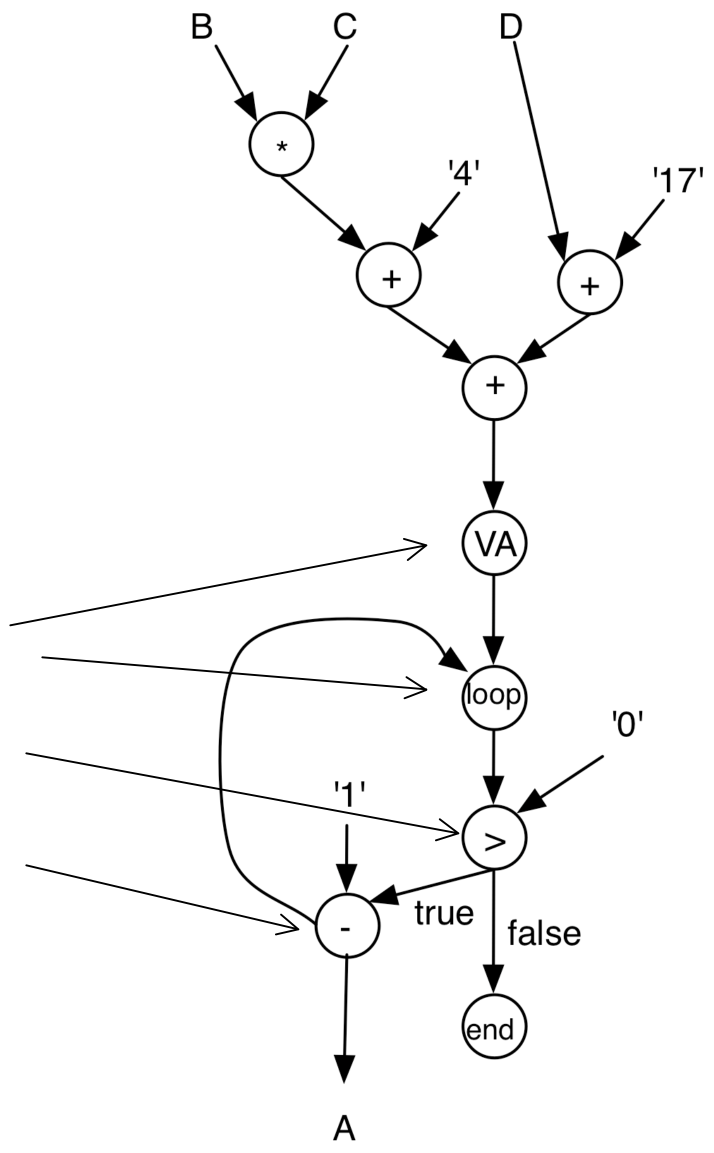
Control and Data Flow Graph

```
E := B * C + 4;
F := D + 17;
A := E + F;
while (A > 0) loop
     A := A - 1;
end loop;
```



Control and Data Flow Graph

```
E := B * C + 4;
F := D + 17;
A := E + F;
while (A > 0) loop
      A := A - 1;
end loop;
```



Control and Data Flow Graph

- directed acyclic graph: edges and nodes
- **edges:** transfer *value* or *control*
- **nodes:**
  - *Operational nodes:* Responsible for arithmetic, logical or relational operations
  - *Call nodes:* Calls to subprogram
  - *Control nodes:* Responsible for conditionals and loops
  - *Storage nodes:* Assignment operators, holding registers

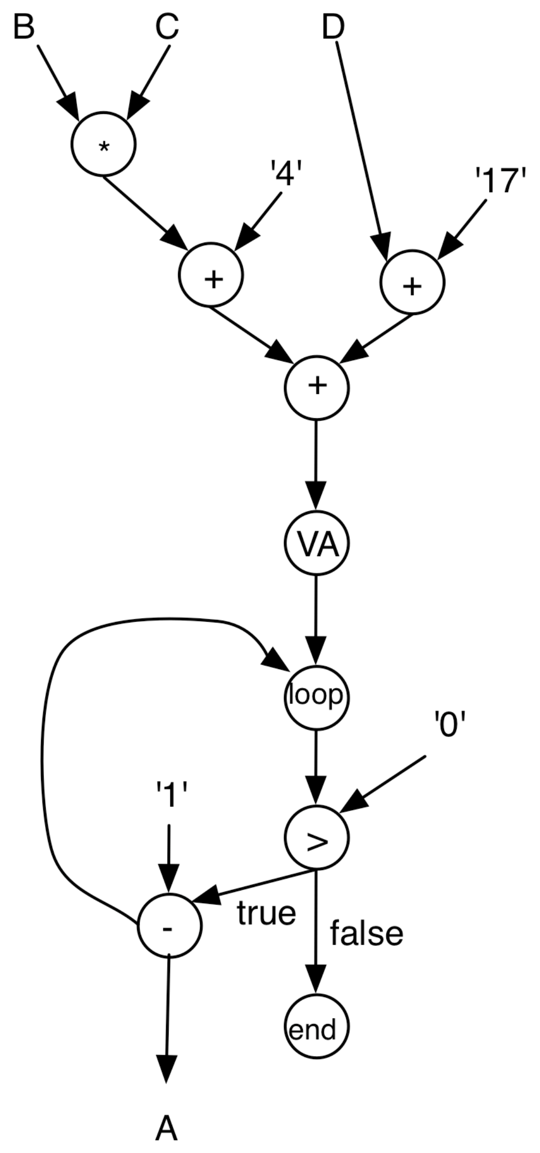
CDFG Optimization

- Once we have generated the CDFG it is possible to perform optimizations on the graph **before** it is partitioned...

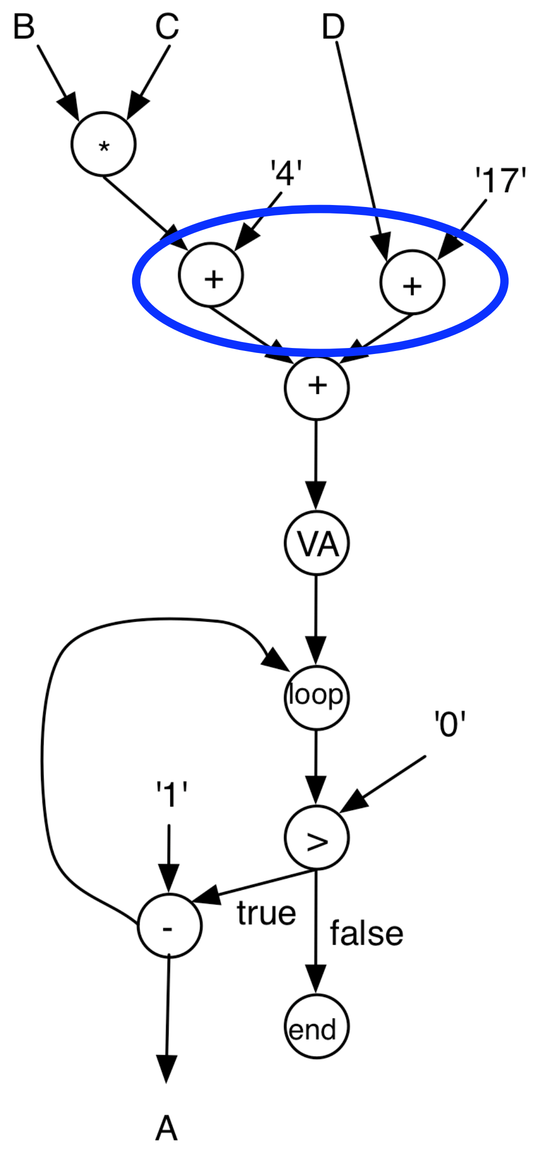
  - dead code elimination
  - loop unrolling
  - etc.

- These optimization are used extensively in software compliers

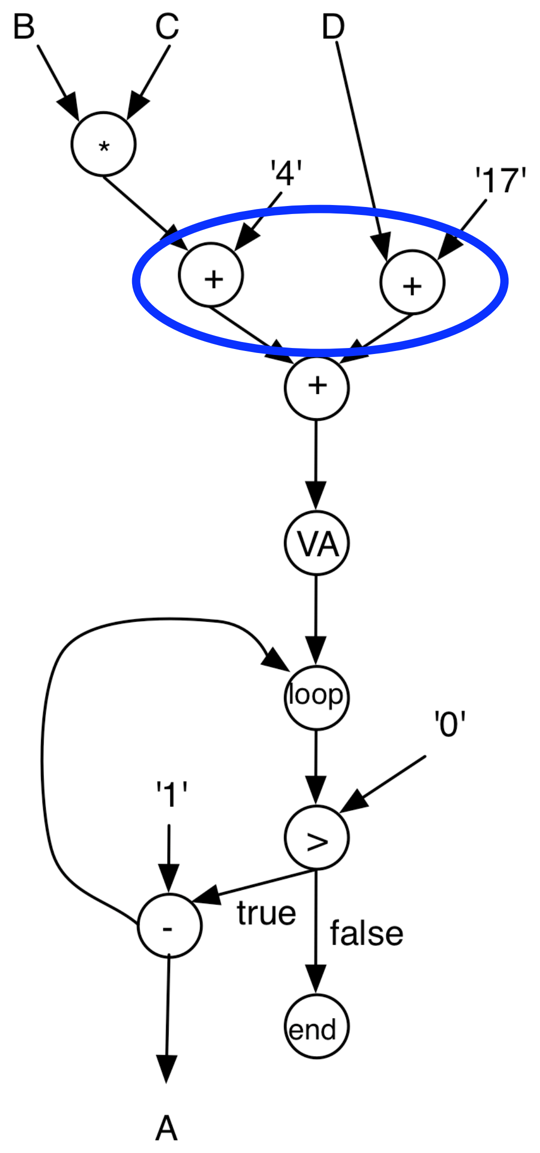
CDFG Optimization

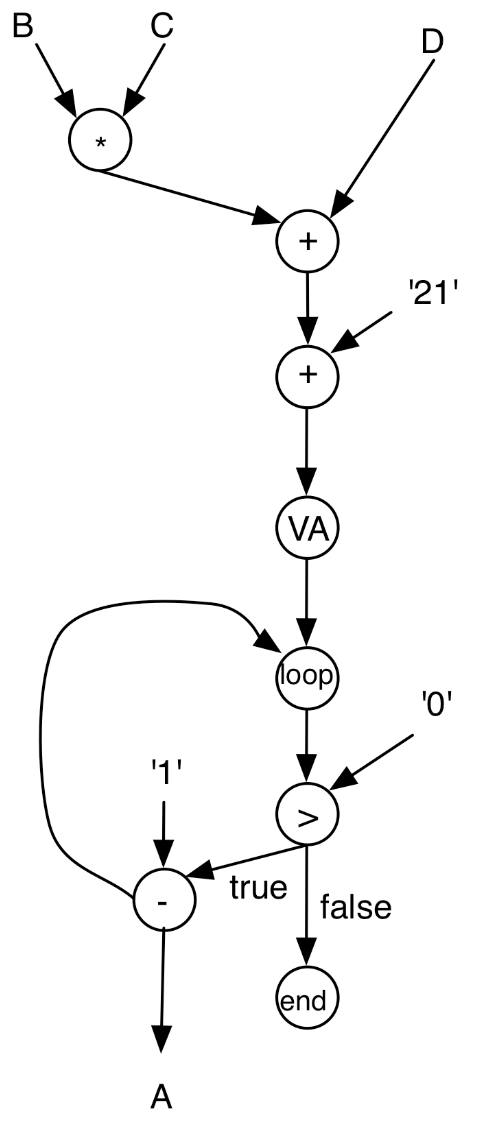


CDFG Optimization

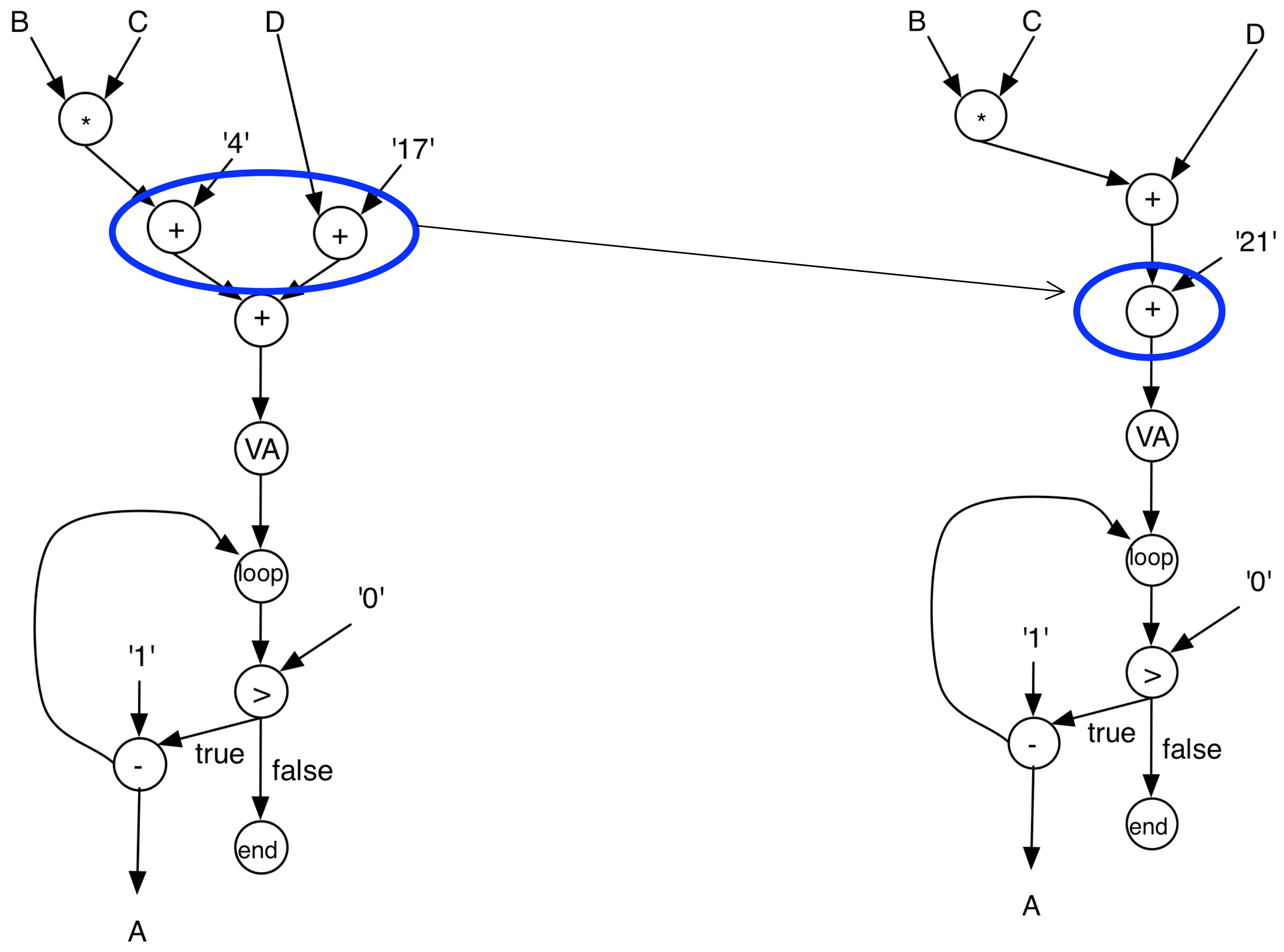


CDFG Optimization





CDFG Optimization



Key Challenges

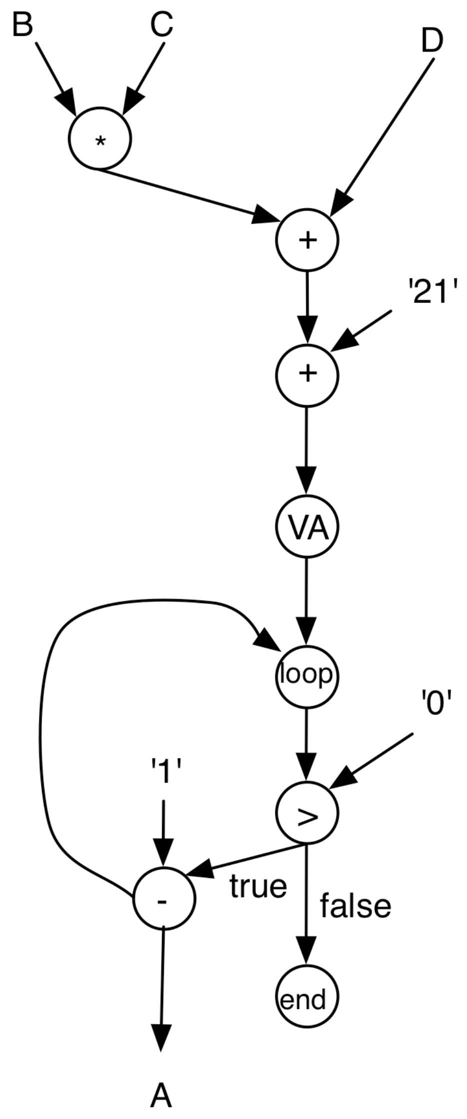
- The construction of the CDFG itself may be biased towards **hardware** or **software**
  - We want to leave the decision up to the tool, but as we decided on the structure of the graph we are influencing the decision
- Optimization at this point may also be biased towards **hardware** or **software**
  - For instance in the previous example, eliminating operators saves software execution time... but hardware instances can operate in parallel so is there value?

Allocation and Scheduling

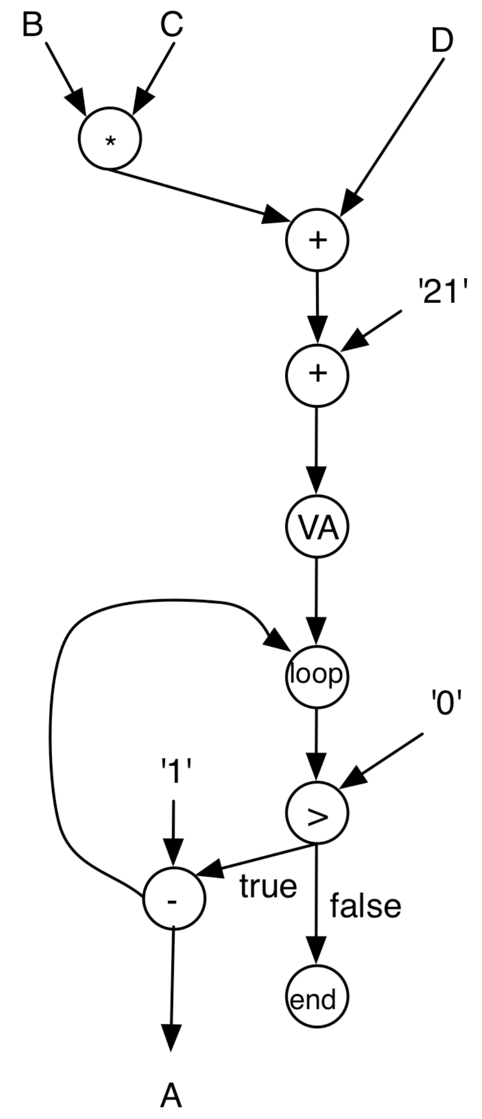
Allocation and Scheduling

- Now that we have a structure we can work with we can start to partition the problem....
- Any part of the graph may be implemented in **hardware** or **software**
- However, the target resources are not homogeneous which makes decisions hard!

Software Only....

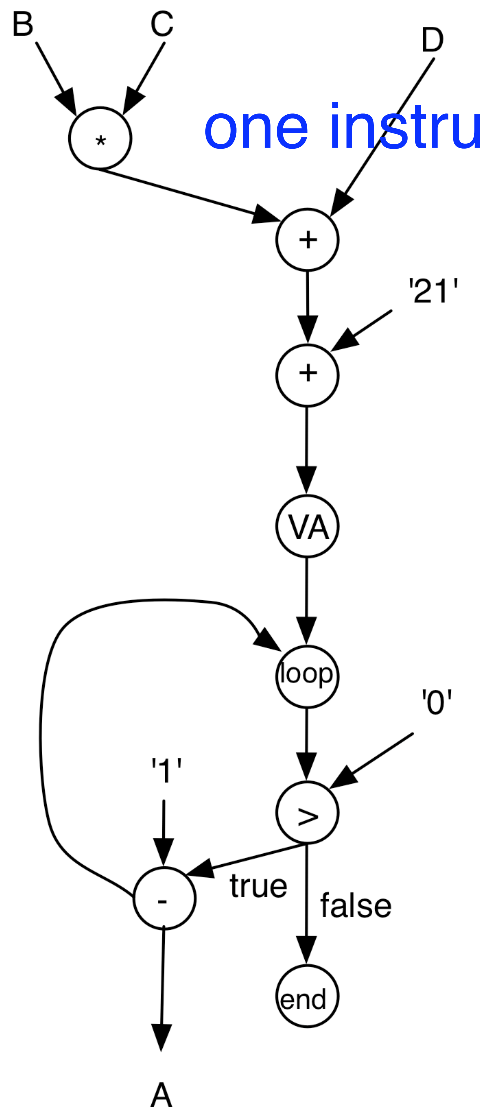


Software Only....



Resource Cost:

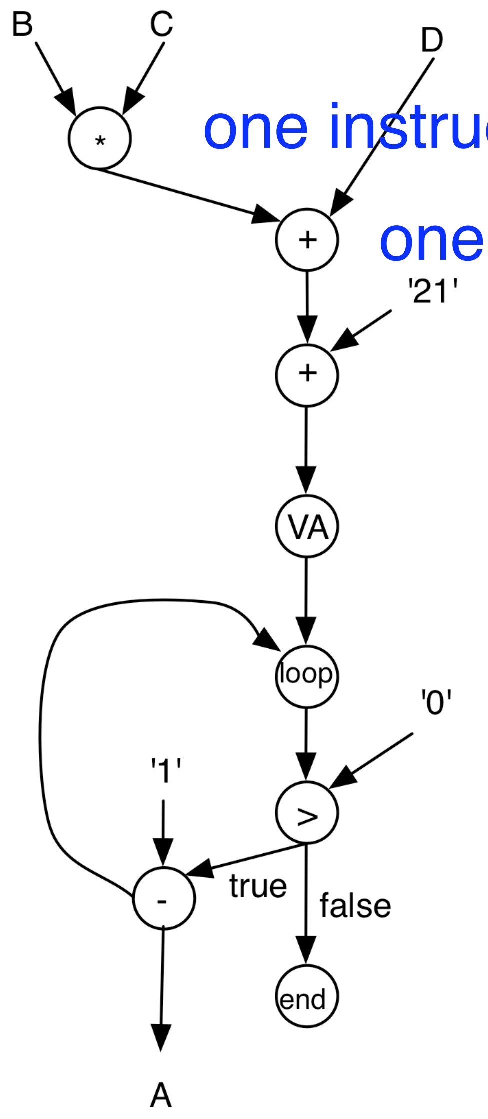
Software Only....



one instruction

Resource Cost:

Software Only....

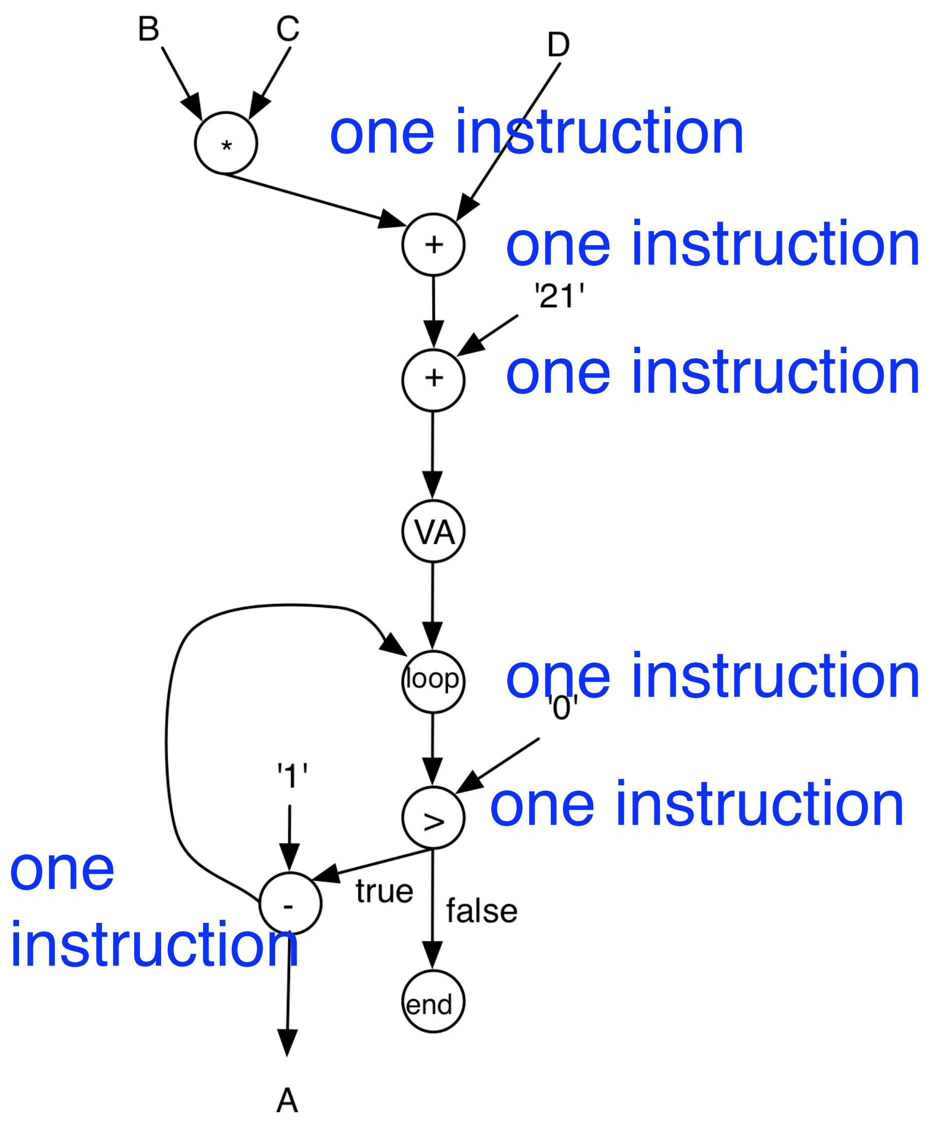


one instruction

one instruction

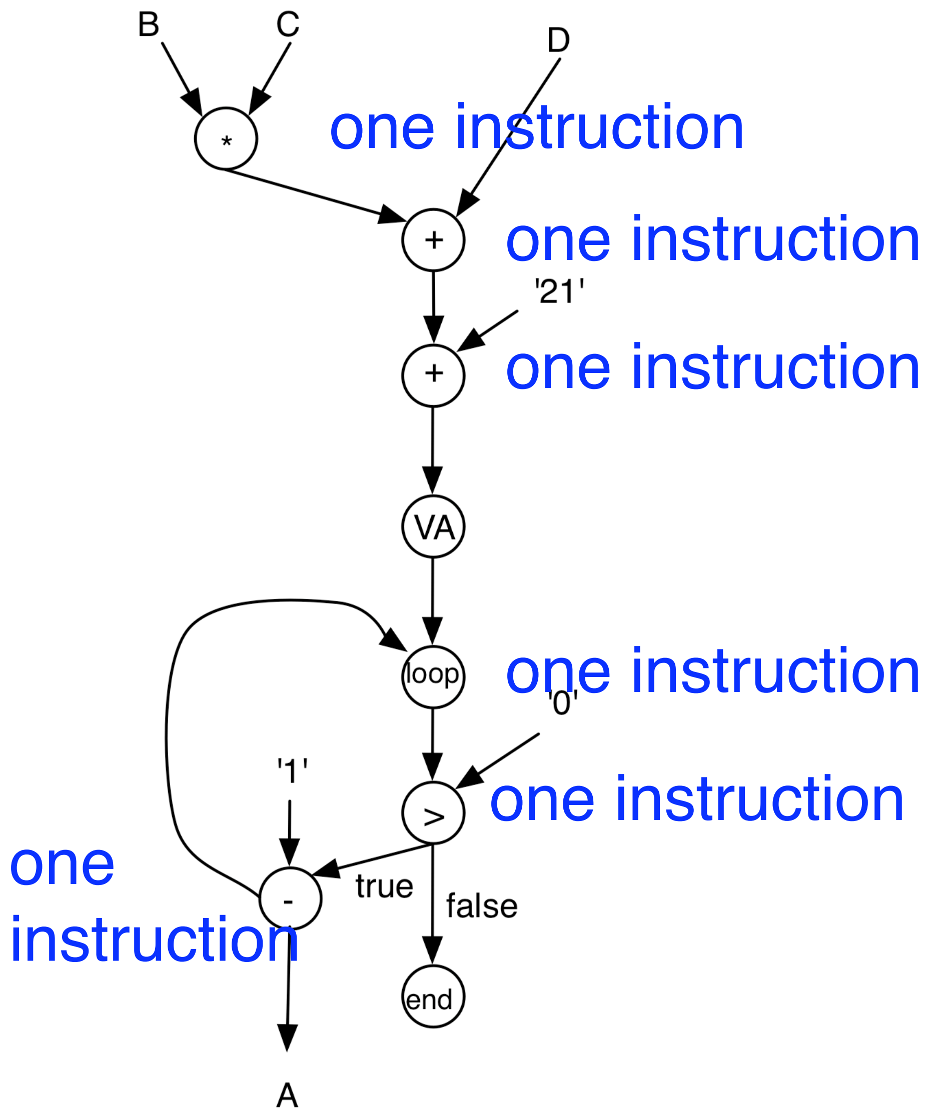
Resource Cost:

Software Only....



Resource Cost:

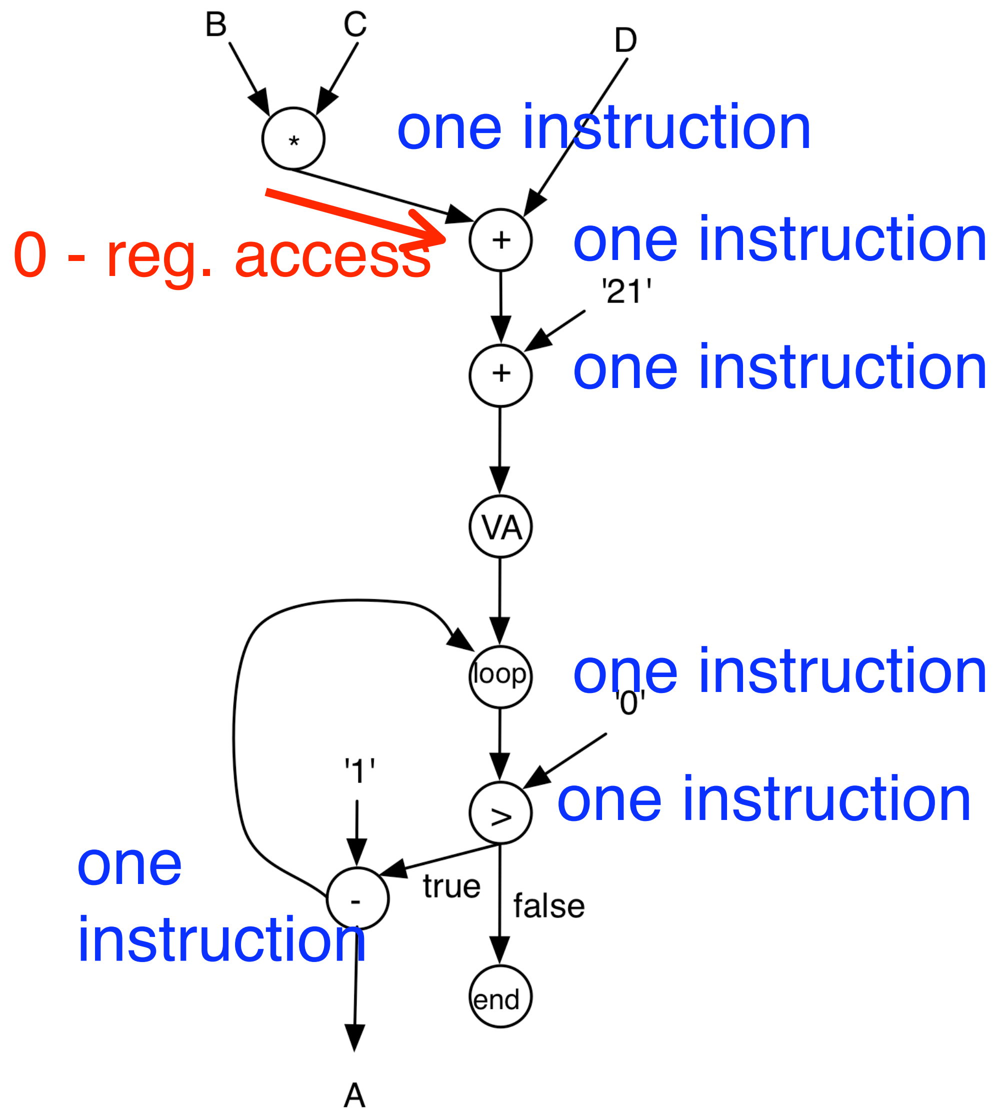
Software Only....



Resource Cost:

Communications Overhead:

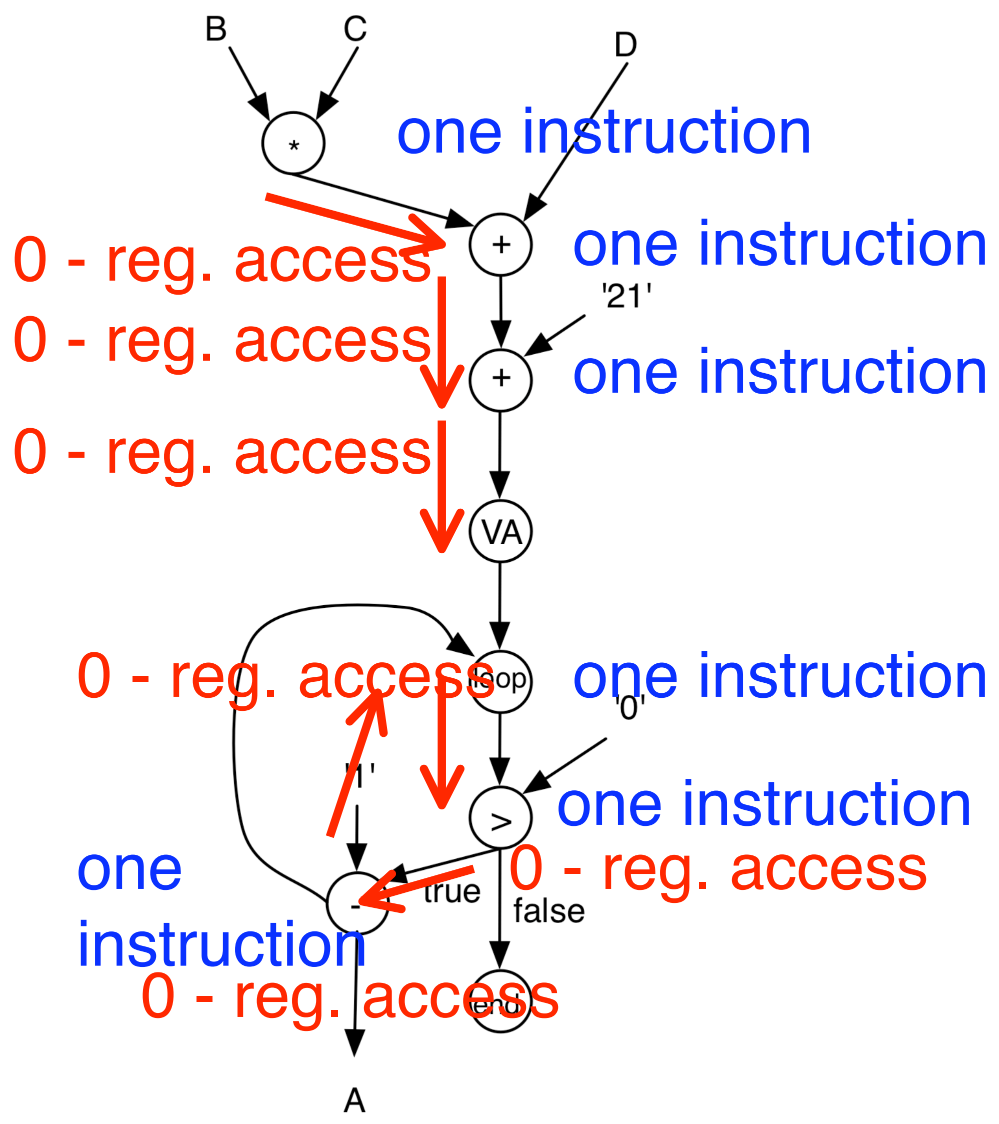
Software Only....



Resource Cost:

Communications Overhead:

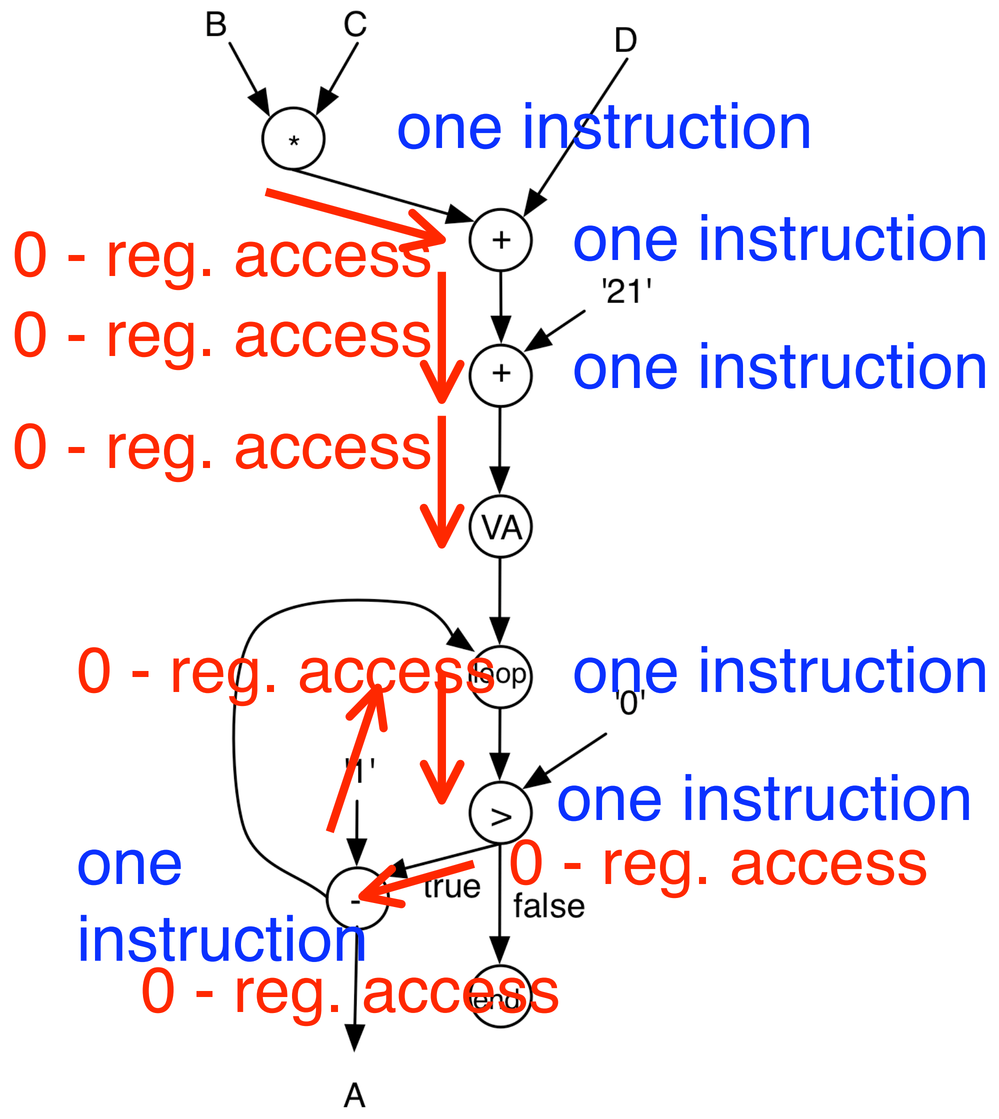
Software Only....



Resource Cost:

Communications Overhead:

Software Only....

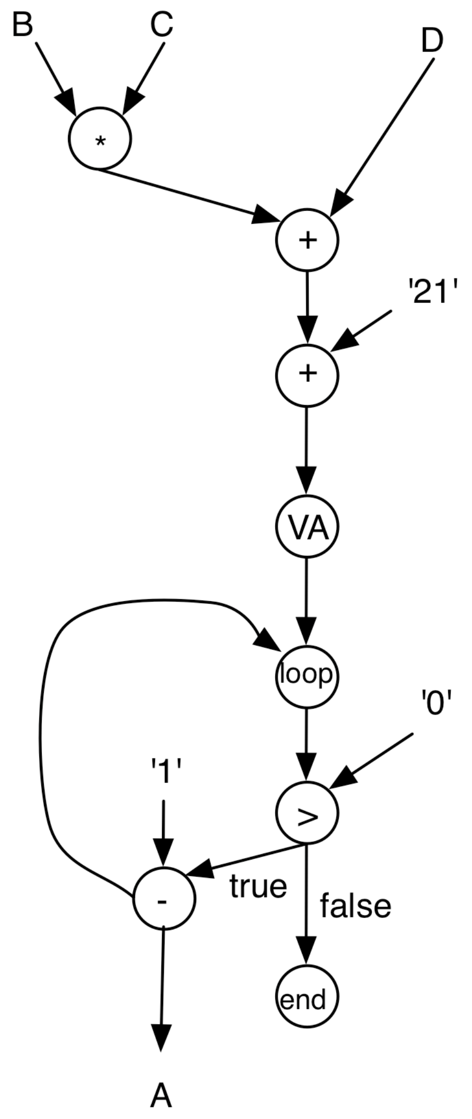


Resource Cost:

Communications Overhead:

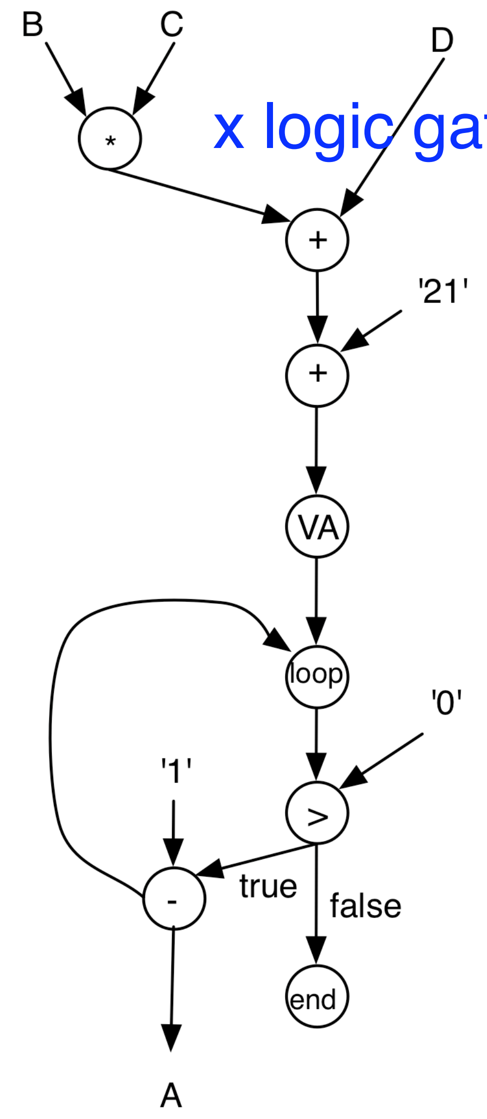
Essentially a problem of scheduling instructions...

Hardware Only...



Resource Cost:

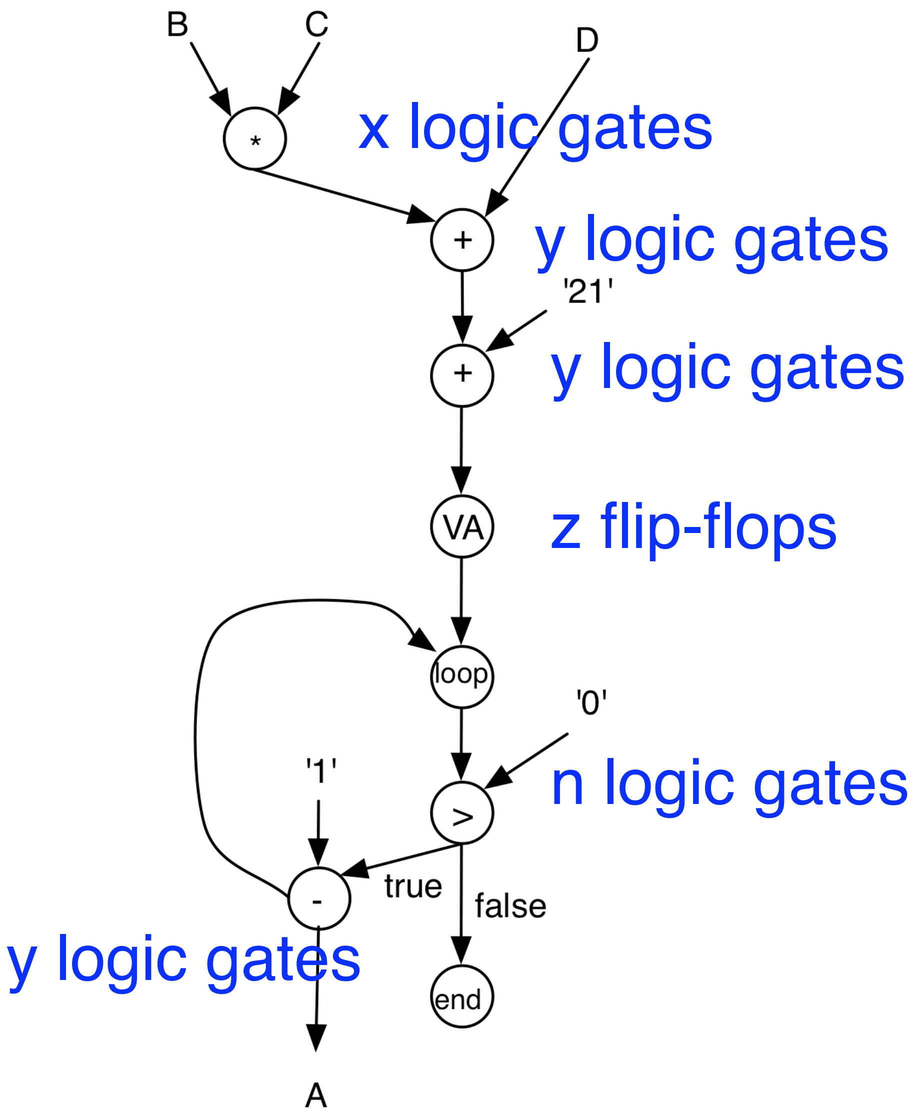
Hardware Only...



x logic gates

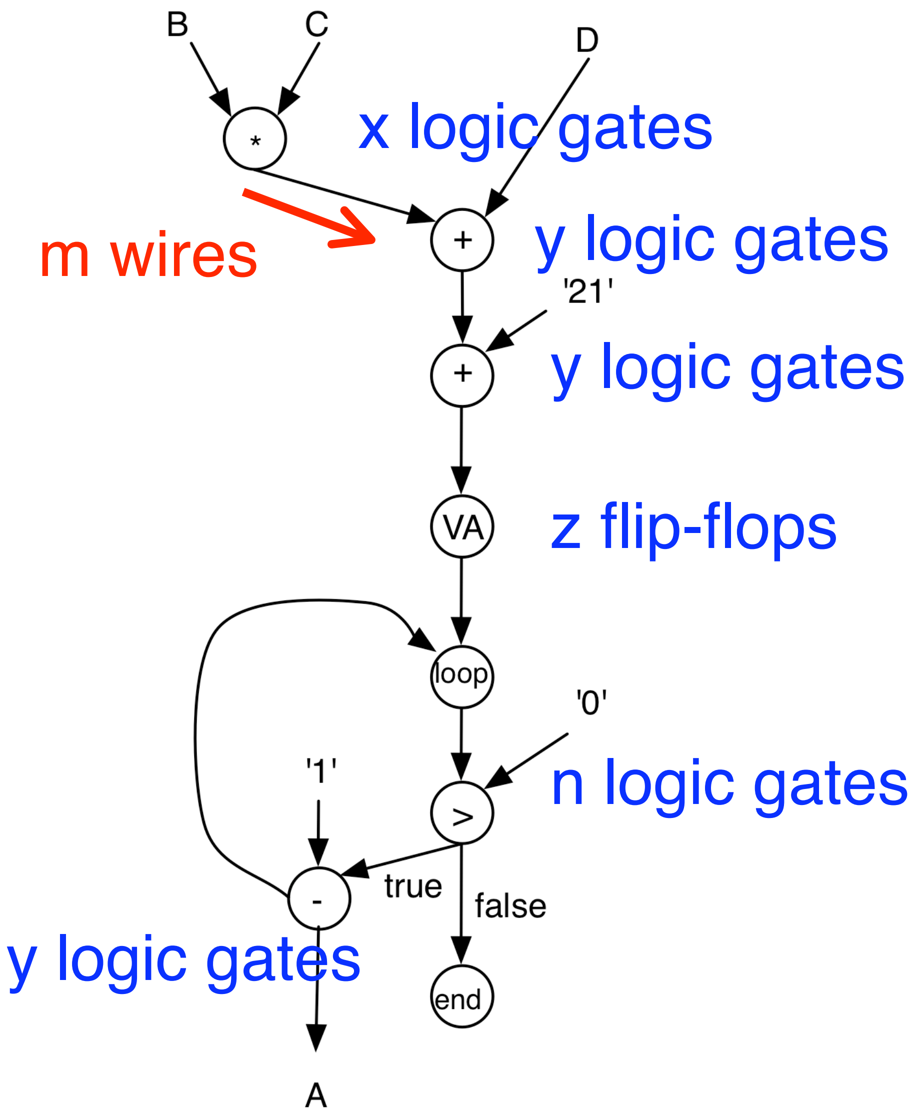
Resource Cost:

Hardware Only...



Resource Cost:

Hardware Only...

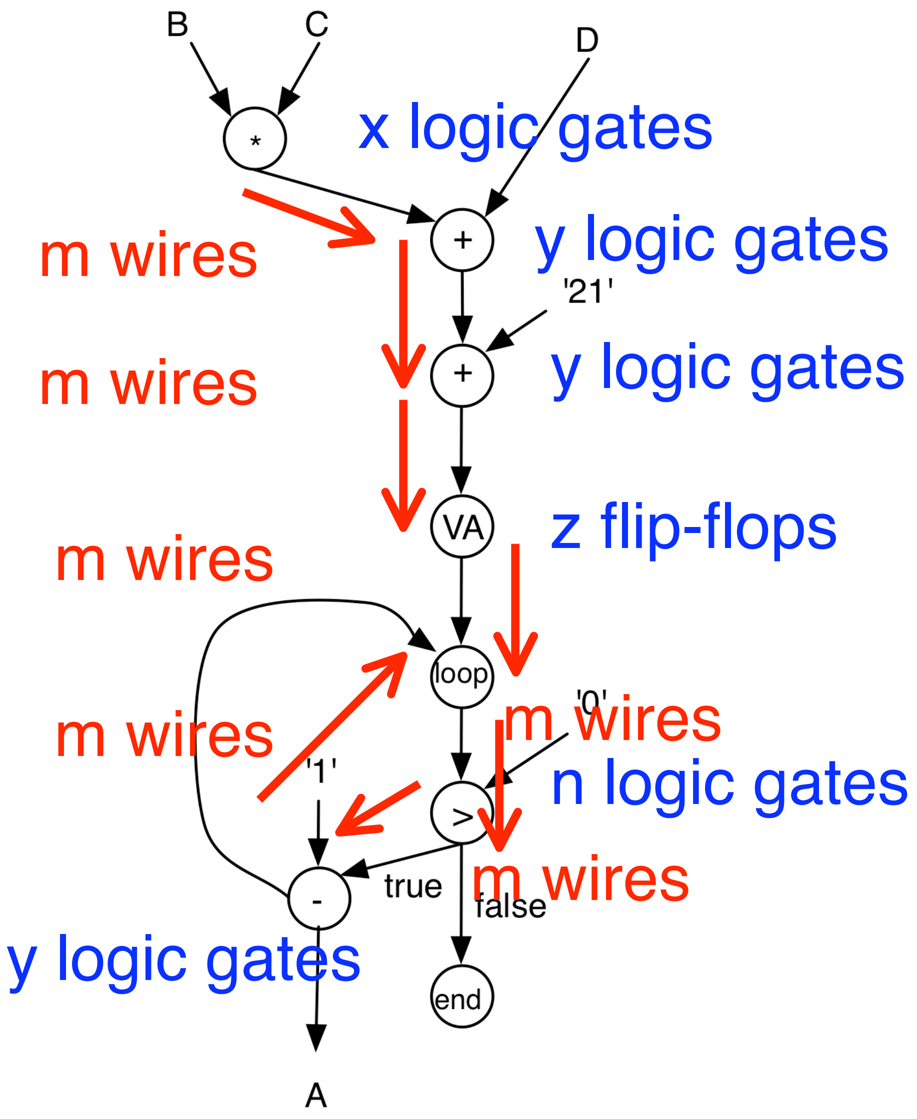


Resource Cost:

Communications

Overhead:

Hardware Only...

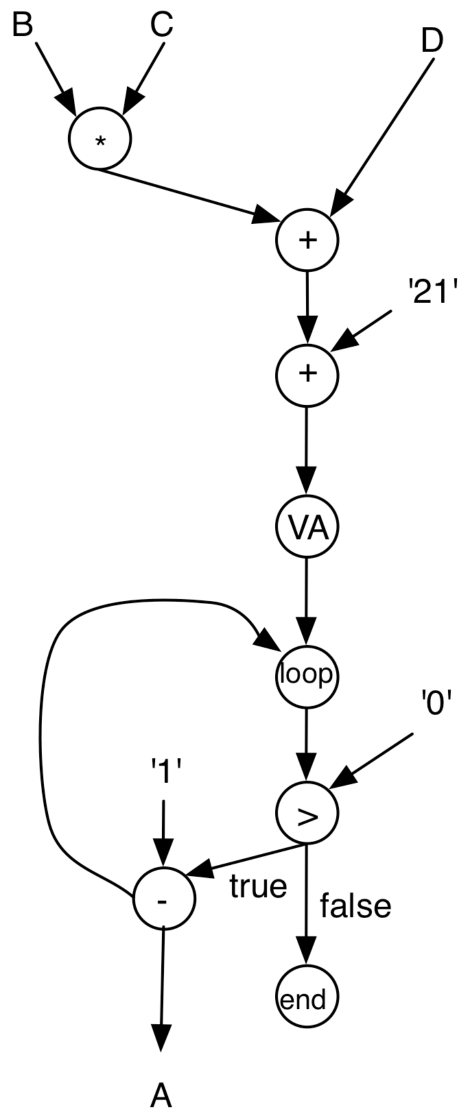


Resource Cost:

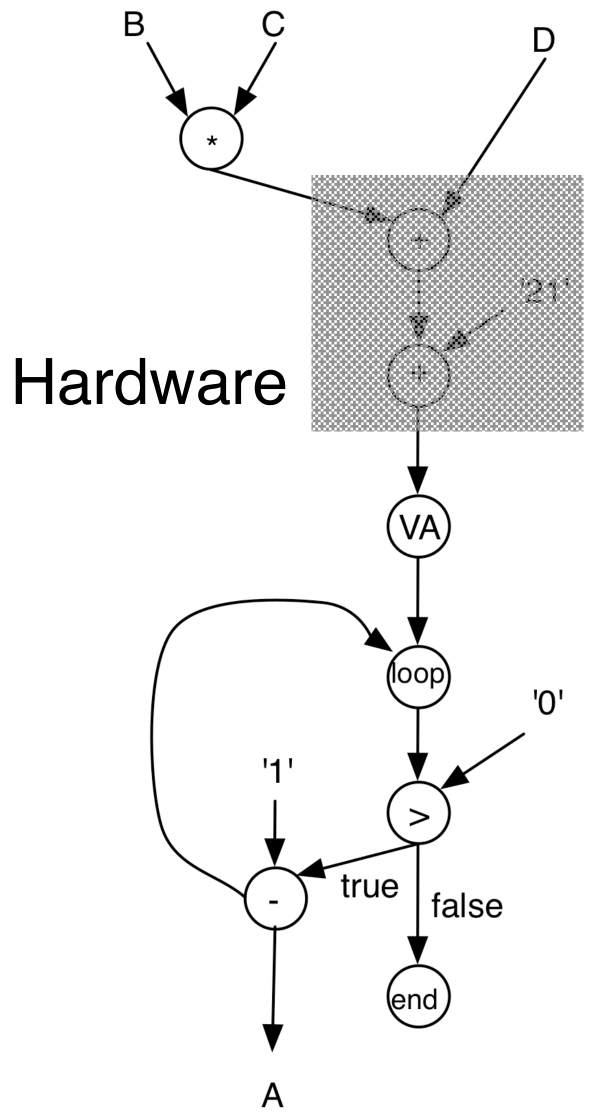
Communications Overhead:

Essentially a problem minimizing the number of gates in the circuit...

Mixed...

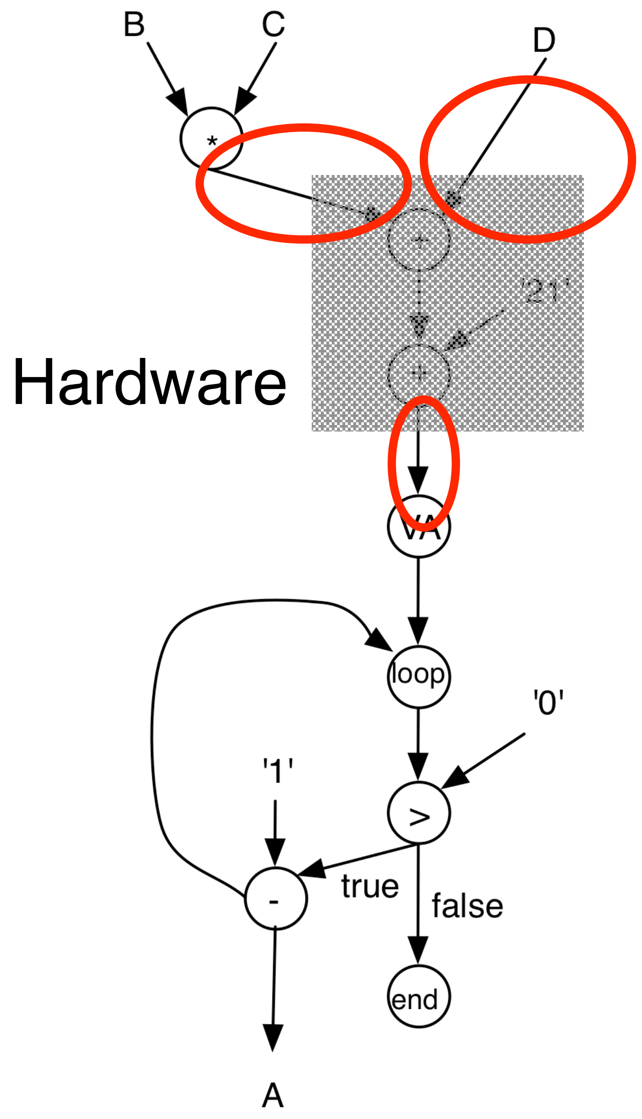


Mixed...



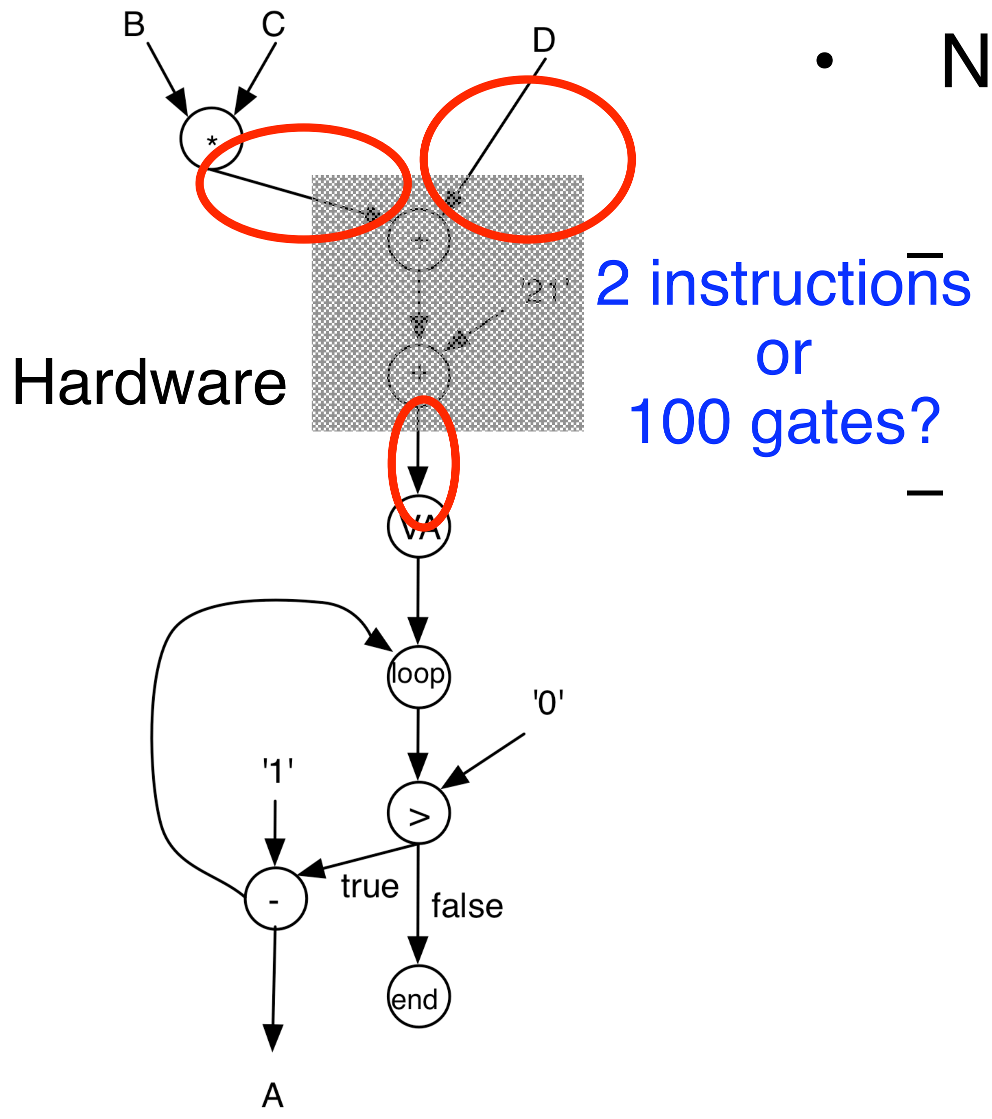
- Now things are difficult...

Mixed...



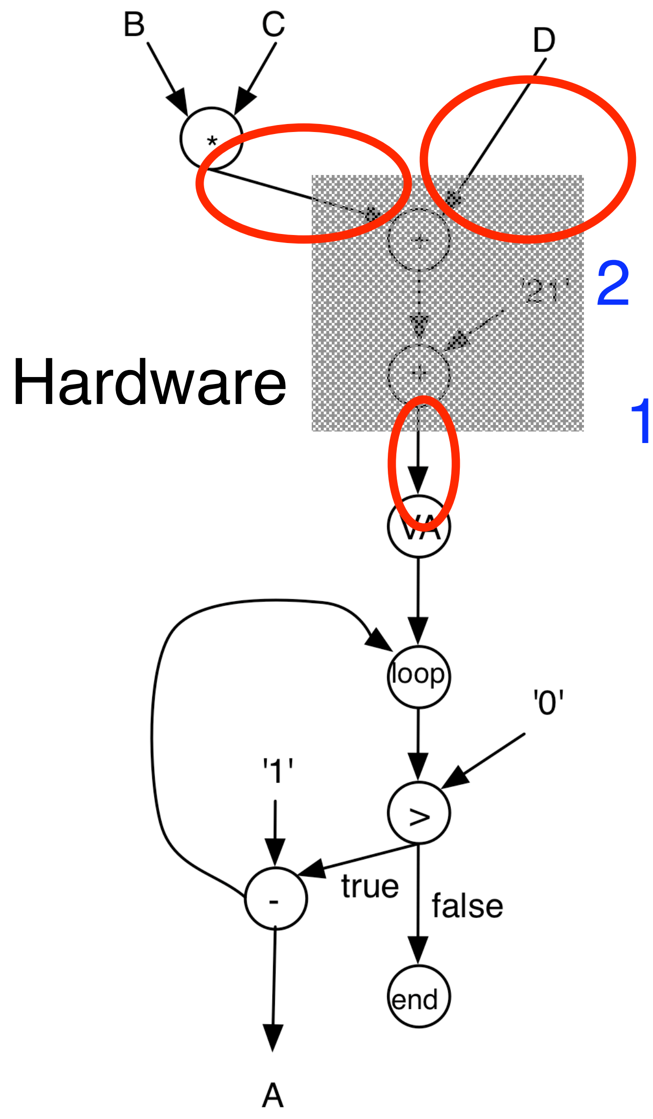
- Now things are difficult...
  - hardware/software communication overhead

Mixed...



- Now things are difficult...
  - hardware/software communication overhead
  - cost trade-off: gates versus run time?

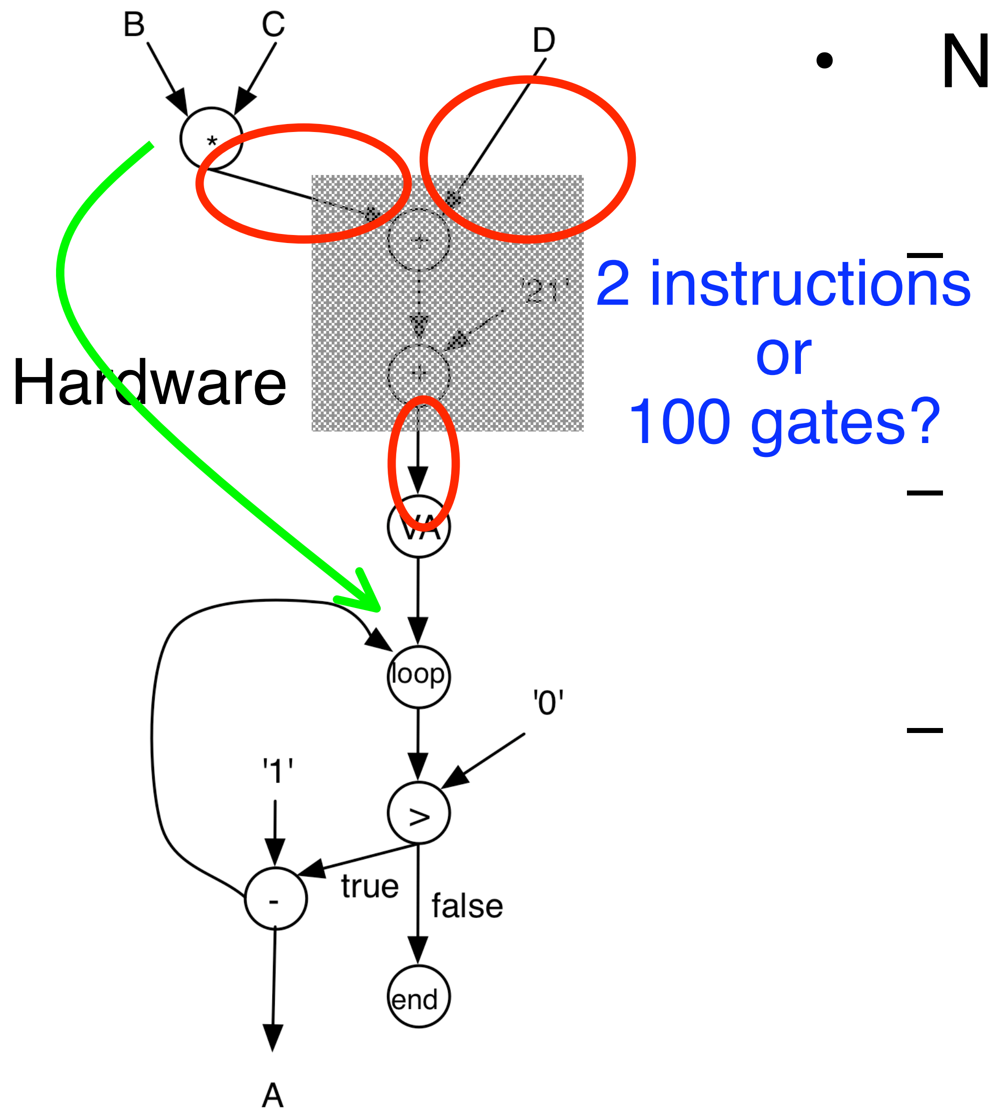
Mixed...



2 instructions
or
100 gates?

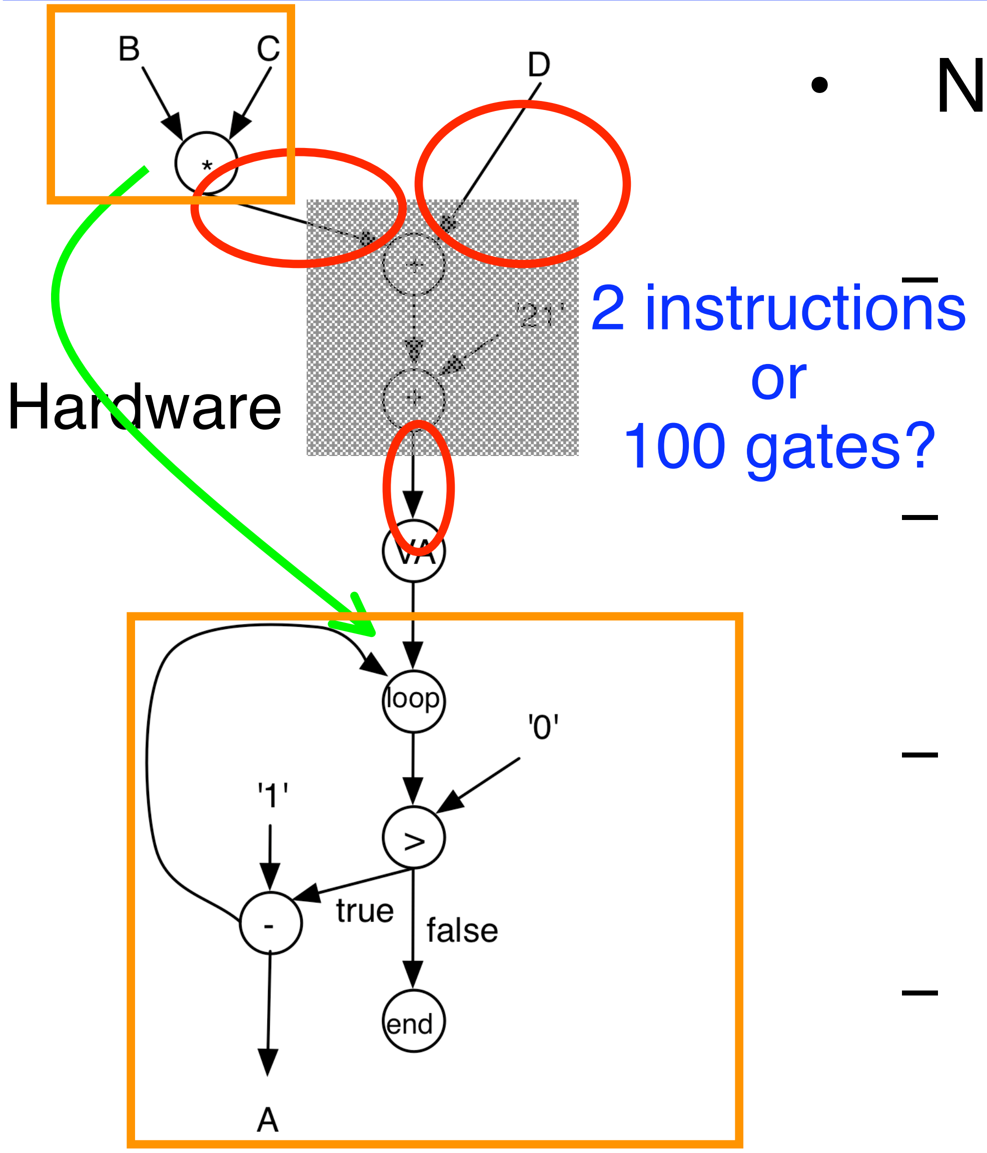
- Now things are difficult...
  - hardware/software communication overhead
  - cost trade-off: gates versus run time?

Mixed...



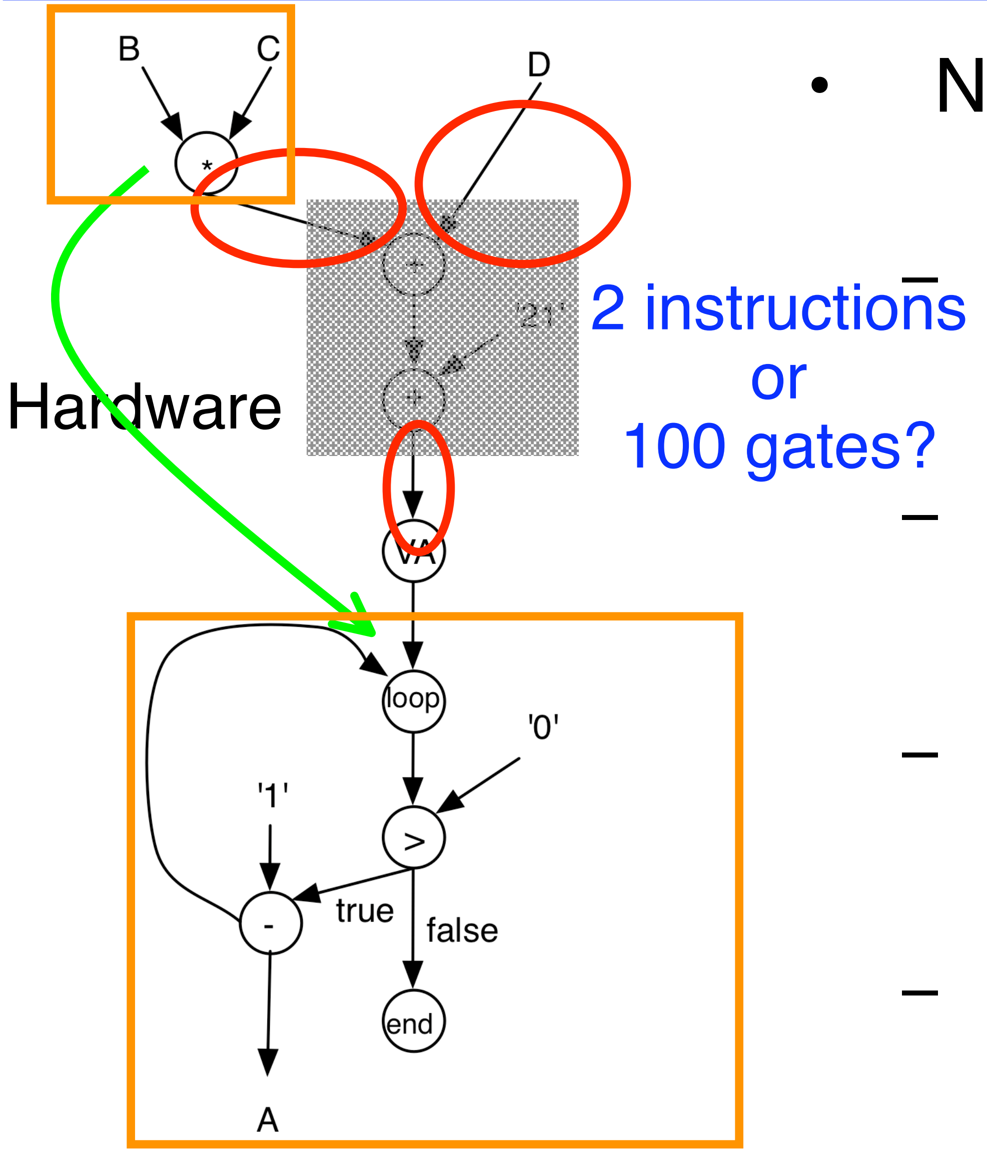
- Now things are difficult...
  - hardware/software communication overhead
  - cost trade-off: gates versus run time?
  - parallelism: is the software stalled?

Mixed...



- Now things are difficult...
  - hardware/software communication overhead
  - cost trade-off: gates versus run time?
  - parallelism: is the software stalled?
  - complier efficiency / synthesis efficiency

Mixed...



- Now things are difficult...
  - hardware/software communication overhead
  - cost trade-off: gates versus run time?
  - parallelism: is the software stalled?
  - complier efficiency / synthesis efficiency

Key Challenges

- Hardware/Software **communication** imposes a significant performance overhead
- Software and hardware have different **cost metrics**
- Often requires finding **parallelism** between hardware and software
- Breaking up the problem can often reduce the **efficiency** of both the software compiler and hardware synthesis

Algorithms

Algorithms

- Any algorithm that we use must take into account the **key issues** that we identified in the previous section...

- We will not try to cover algorithms in detail in this class:
  - EECE 583 -> High-level Synthesis
  - Research papers: look for “cosynthesis”, “hardware/software partitioning”, etc.

- Lets summarize some approaches...

Algorithms

- **Software-centric**:
  - Generate software and then try to identify sections to migrate to hardware
- **Hardware driven**:
  - Generate hardware and then try to identify circuits to migrate to software
- **Iterative**:
  - Iteratively assign some nodes to hardware and to software then evaluating results

Simulated Annealing

- At a very high-level:

1. Randomly assign each node to be hardware or software
2. Calculate the cost of the resulting design
3. Swap *one* assignment randomly
4. Re-calculate the cost of with this new design
5. If the cost is low keep the assignment, if not revert the swap
6. If cost > goal goto 3.

Simulated Annealing

- At a very high-level:

1. Randomly assign each node to be hardware or software
2. Calculate the cost of the resulting design
3. Swap *one* assignment randomly
4. Re-calculate the cost of with this new design
5. If the cost is low keep the assignment, if not revert the swap
6. If cost > goal goto 3.

Needs to be *fast* and *accurate*...
hard for this problem!

Summary

Summary

- SoCs are built with both **hardware** and **software**, so we have to make a choice...

- The partition between these two aspect of the implementation has a dramatic effect on the **cost**, **power** and **performance** of the SoC

- It is possible to automate this task, however it is difficult to get good results

“Hardware-Software Cosynthesis for Microcontrollers”

Paper

- This paper is quite old (1993)
- They are trying to solve to of building a multi-chip system, but it is very similar the SoC problem...
- Hard problem: Automated hardware-software partitioning is still not mainstream!
- Interesting to see where the challenges were and how they handled them
- Lets look at the paper....

Paper

Ernst, R.; Henkel, J.; Benner, T., "Hardware-software cosynthesis for microcontrollers," *Design & Test of Computers, IEEE* , vol.10, no.4, pp.64-75, Dec 1993

End.