

ESE532: System-on-a-Chip Architecture

Day 9: October 5, 2020
High-Level Synthesis (HLS)
C-to-gates
More accurate: C-for-gates



Penn ESE532 Fall 2020 -- DeHon

Today

- Motivation
- Spatial Computations from C specification
 - Variables and expression (pre-lecture)
 - Simple Conditionals (Part 1)
 - Functions (part 2)
 - Globals
 - Arrays (Part 3)
 - Memory (Part 4, time permitting)
- Complexities from C semantics (Part 5)

Penn ESE532 Fall 2020 -- DeHon

2

Message

- C (or any programming language) specifies a computation
- Can describe spatial computation
 - A dataflow graph with physical operators for each operation
- Underlying semantics is sequential
 - Watch for unintended sequentialization
 - Write C for spatial differently than you write C for processors

Penn ESE532 Fall 2020 -- DeHon

3

Coding Accelerators

- Want to exploit FPGA logic on F1, Zynq to accelerate computations
- Traditionally has meant develop accelerators in
 - Hardware Description Language (HDL)
 - E.g. Verilog → see in CIS371, CIS501
 - Directly in schematics

Penn ESE532 Fall 2020 -- DeHon

4

Course “Hypothesis”

- C-to-gates synthesis mature enough to use to specify hardware
 - Leverage fact everyone knows C
 - (must, at least, know C to develop embedded code)
 - Avoid taking time to teach Verilog or VHDL
 - Or making Verilog a pre-req.
 - Focus on teaching how to craft hardware
 - Using the C already know
 - ...may require thinking about the C differently

Penn ESE532 Fall 2020 -- DeHon

5

Discussion [open]

- Is it obvious we can write C to describe hardware?
- What parts of C translate naturally to hardware?
- What parts of C might be problematic?
- What parts of hardware design might be hard to describe in C?

Penn ESE532 Fall 2020 -- DeHon

6

Three Perspectives

1. How express spatial/hardware computations in C
 - May want to avoid some constructs in C
2. How express computations
 - Hopefully, equally accessible to spatial and sequential implementations
3. Given C code: how could we implement in spatial hardware
 - Some corner cases and technicalities make tricky

Penn ESE532 Fall 2020 -- DeHon

[copy to board]

7

Advantage

- Use C for hardware and software
 - Test out functionality entirely in software
 - Debug code before put on hardware
 - where harder to observe what's happening
 - ...without spending time in place and route
 - ...which you soon see is slow...
 - Explore hardware/software tradeoffs by targeting same code to either hardware or software

Penn ESE532 Fall 2020 -- DeHon

8

Context

- C most useful for describing behavior of operators



- C alone doesn't naturally capture task parallelism

Penn ESE532 Fall 2020 -- DeHon

9

Preclass F

- Ready for preclass f?
- [Skip to preclass f](#)

Penn ESE532 Fall 2020 -- DeHon

10

C Primitives Arithmetic Operators

- Unary Minus (Negation) $-a$
- Addition (Sum) $a + b$
- Subtraction (Difference) $a - b$
- Multiplication (Product) $a * b$
- Division (Quotient) a / b
- Modulus (Remainder) $a \% b$

Things might have a hardware operator for...

Penn ESE532 Fall 2020 -- DeHon

11

C Primitives Bitwise Operators

- Bitwise Left Shift $a \ll b$
- Bitwise Right Shift $a \gg b$
- Bitwise One's Complement $\sim a$
- Bitwise AND $a \& b$
- Bitwise OR $a | b$
- Bitwise XOR $a \wedge b$

Things might have a hardware operator for...

Penn ESE532 Fall 2020 -- DeHon

12

C Primitives Comparison Operators

- Less Than $a < b$
- Less Than or Equal To $a \leq b$
- Greater Than $a > b$
- Greater Than or Equal To $a \geq b$
- Not Equal To $a \neq b$
- Equal To $a == b$
- Logical Negation $!a$
- Logical AND $a \&\& b$
- Logical OR $a \|\| b$

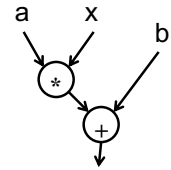
Things might have a hardware operator for...

Penn ESE532 Fall 2020 -- DeHon

13

Expressions: combine operators

- $a*x+b$



A connected set of operators
→ Graph of operators

Penn ESE532 Fall 2020 -- DeHon

14

Expressions: combine operators

- $a*x+b$
- $a*x*x+b*x+c$
- $a*(x+b)*x+c$
- $((a+10)*b < 100)$

A connected set of operators
→ Graph of operators

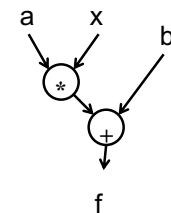
Penn ESE532 Fall 2020 -- DeHon

15

C Assignment

- Basic assignment statement is:
Location = expression

- $f=a*x+b$



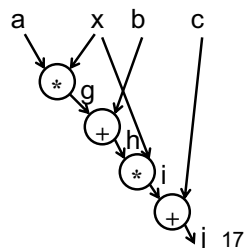
Penn ESE532 Fall 2020 -- DeHon

16

Straight-line code

- a sequence of assignments
- What does this mean?

```
g=a*x;
h=b+g;
i=h*x;
j=i+c;
```



Penn ESE532 Fall 2020 -- DeHon

17

Variable Reuse

- Variables (locations) define flow between computations
- Locations (variables) are reusable

```
t=a*x;
r=t*x;
t=b*x;
r=r+t;
r=r+c;
```

Penn ESE532 Fall 2020 -- DeHon

18

Variable Reuse

- Variables (locations) define flow between computations
- Locations (variables) are reusable


```
t=a*x; t=a*x;
r=t*x; r=t*x;
t=b*x; t=b*x;
r=r+t; r=r+t;
r=r+c; r=r+c;
```
- Sequential assignment semantics tell us which definition goes with which use.
 - Use gets most recent preceding **definition**.

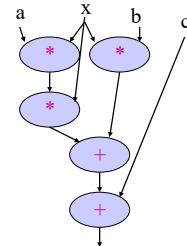
Penn ESE532 Fall 2020 -- DeHon

19

Dataflow

- Can turn sequential assignments into dataflow graph through def→use connections

```
t=a*x; t=a*x;
r=t*x; r=t*x;
t=b*x; t=b*x;
r=r+t; r=r+t;
r=r+c; r=r+c;
```

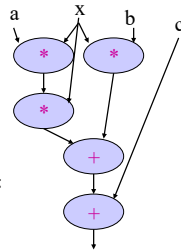


Penn ESE532 Fall 2020 -- DeHon

20

Dataflow Height

- $t=a*x$; $t=a*x$;
 $r=t*x$; $r=t*x$;
 $t=b*x$; $t=b*x$;
 $r=r+t$; $r=r+t$;
 $r=r+c$; $r=r+c$;
- Height (delay) of DF graph may be less than # sequential instructions.



Penn ESE532 Fall 2020 -- DeHon

21

Lecture Checkpoint

- Happy with ?
 - Straight-line code
 - Variables
- Graph for preclass f

```
int f(int a, int b)
{
  int t, c, d;
  a=a&(0xf);
  b=b&(0xf);
  t=b+3;
  c=a^t;
  t=a-2;
  d=b^t;
  return(d);
}
```

Penn ESE532 Fall 2020 -- DeHon

Straight Line Code

- C is fine for expressing straight-line code and variables
 - Has limited data types
 - Address with tricks like masking
 - Address with user-defined types

Penn ESE532 Fall 2020 -- DeHon

23

Optimizations can probably expect compiler to do

- Constant propagation: $a=10$; $b=c[a]$;
- Copy propagation: $a=b$; $c=a+d$; $\rightarrow c=b+d$;
- Constant folding: $c[10*10+4]$; $\rightarrow c[104]$;
- Identity Simplification: $c=1*a+0$; $\rightarrow c=a$;
- Strength Reduction: $c=b*2$; $\rightarrow c=b<<1$;
- Dead code elimination
- Common Subexpression Elimination:
 - $C[x*100+y]=A[x*100+y]+B[x*100+y]$
 - $t=x*100+y$; $C[t]=A[t]+B[t]$;
- Operator sizing: `for (i=0; i<100; i++) b[i]=(a&0xff+i);`

Penn ESE532 Fall 2020 -- DeHon

24

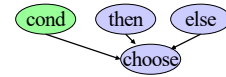
Conditionals

- What can we do for simple conditionals?

```
if (a<b)
  res=b-a
Else
  res=a-b
```

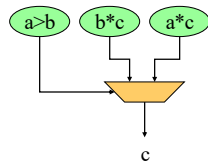
Simple Control Flow

- If (cond) { ... } else { ... }
- Assignments become conditional
- In simplest cases (no memory ops), can treat as dataflow node



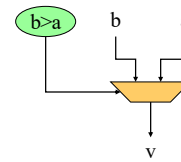
Simple Conditionals

```
if (a>b)
  c=b*c;
else
  c=a*c;
```



Simple Conditionals

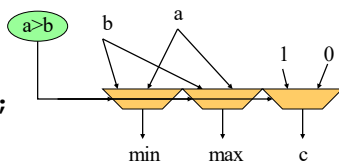
```
v=a;
if (b>a)
  v=b;
```



- If not assigned, value flows from before assignment

Simple Conditionals

```
max=a;
min=a;
if (a>b)
  {min=b;
  c=1;}
else
  {max=b;
  c=0;}
• May (re)define many values on each branch.
```



Preclass G

- Graph for preclass g as mux-conversion?

```
int g(int a, int b)
{
  int t, c, d;
  // same as above
  a=a&(0x0f);
  b=b&(0x0f);
  t=b+3;
  c=a^t;
  t=a-2;
  d=b^t;
  //added (not in f)
  if (a<b)
    d=c;
  // end added
  return(d);
}
```

Part 2

Functions and Globals

Function Call

- What computation is this describing?

```
int f(int a, int b)
  return(sqrt(a*a+b*b));
```

```
for(i=0;i<MAX;i++)
  D[i]=f(A[i],B[i]);
```

- What role does the function call play?

Inline Transformation

- Inline a function
 - Copy the body of function
 - Into the point of call
 - Replacing the function arguments
 - With the arguments supplied in the call

```
int f(int a, int b)
  return(sqrt(a*a+b*b));
for(i=0;i<MAX;i++)
  D[i]=f(A[i],B[i]);
```

→

```
for(i=0;i<MAX;i++)
  D[i]=sqrt(A[i]*A[i]
            +B[i]*B[i]);
```

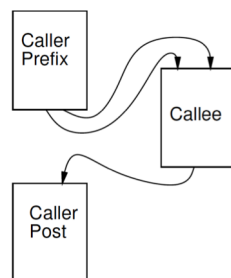
Inline

```
int p(int a)
  return(a*a+2*a-1);
for(i=0;i<MAX;i++)
  D[i]=A[i]*A[i]+2*A[i]-1
        - (B[i]*B[i]+2*B[i]-1);
```

Functions provide descriptive convenience and compactness.
...but don't need to force implementation.

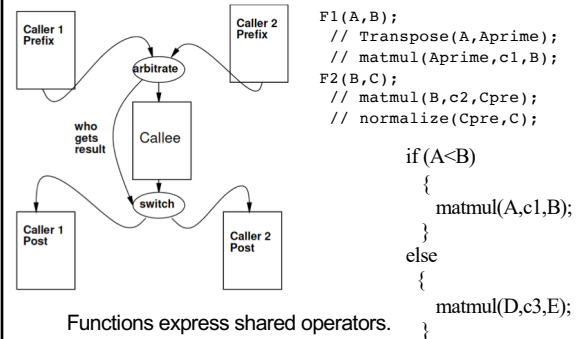
Treat as data flow

- Implement function as an operation
- Send arguments as input tokens
- Get result back as token



Functions provide potential division between substrates? Assign different functions to different substrate (proc, fpga)

Shared Function



Functions express shared operators.

Recursion?

```
int fib(int x) {
  if ((x==0) ||
      (x==1))
    return(1);
  else
    return(
      fib(x-1) +
      fib(x-2));
}
```

- In general won't work.
 - Problem?
- Smart compiler might be able to turn some cases into iterative loop.
- ...but don't count on it.
 - VivadoHLS will not

Penn ESE532 Fall 2020 -- DeHon

37

Global Variables

- Variables not declared in a function resolve to outer context

```
int a=0;
int f1(int *A) {
  for (int i=0;i<a;i++)
    sum+=A[i];
  return(sum); }
void f2(int *A) {
  while (A[a]!=0);
  a++;
}
f2(input);
isum=f1(input);
```

Penn ESE532 Fall 2020 -- DeHon

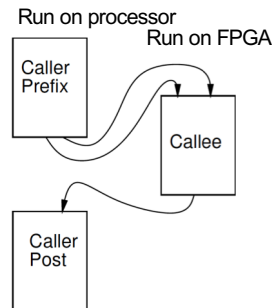
38

Treat as data flow

Functions provide potential division between substrates.

- Impact on global variables?

```
int a=0;
int f1(int *A) {
  for (int i=0;i<a;i++)
    sum+=A[i];
  return(sum); }
void f2(int *A) {
  while (A[a]!=0);
  a++; }
f2(input);
isum=f1(input);
```



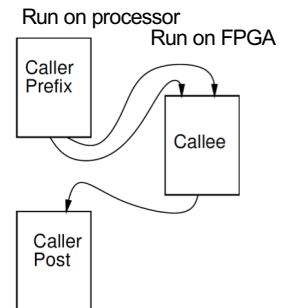
Penn ESE532 Fall 2020 -- DeHon

39

Treat as data flow

Functions provide potential division between substrates.

- Impact on global variables?
- Correct thing
 - Reflect change in variable between substrates
- Evidence Vivado HLS
 - Not synchronized with host C on globals



Penn ESE532 Fall 2020 -- DeHon

40

Global Variables

- Globals generally considered **bad coding practice**
 - Obfuscate flow of data even for human
- Avoid Globals
- With hardware, have extra reason avoid

```
int a=0;
int f1(int *A) {
  for (int i=0;i<a;i++)
    sum+=A[i];
  return(sum); }
void f2(int *A) {
  while (A[a]!=0);
  a++;
}
f2(input);
isum=f1(input);
```

Penn ESE532 Fall 2020 -- DeHon

41

Global Variables

Bad

Better

```
int a=0;
int f1(int *A) {
  for (int i=0;i<a;i++)
    sum+=A[i];
  return(sum); }
void f2(int *A) {
  while (A[a]!=0);
  a++;
}
f2(input);
isum=f1(input);
```

```
int f1(int *A, int len) {
  for (int i=0;i<len;i++)
    sum+=A[i];
  return(sum); }
int f2(int *A) {
  while (A[a]!=0);
  len++;
  return(len)
}
len=f2(input);
isum=f1(input, len);
```

Penn ESE532 Fall 2020 -- DeHon

42

Part 3

Loops and Arrays

Penn ESE532 Fall 2020 -- DeHon

43

Loops...

- From an *express computation* standpoint, have several roles
 - Compact code
 - Unbounded computation
- From describe hardware
 - Compact expression of parallel hardware
 - Express pipelines
 - Express area-time tradeoff

Penn ESE532 Fall 2020 -- DeHon

44

Loop Compact Expression

- What express?
 - Sequential, fully unrolled, partially unrolled?

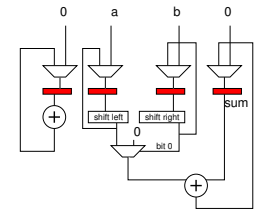
```
sum=0;
for (i=0;i<32;i++) {
    sum+=(0-(b%2)) & a;
    b=b>>1;
    a=a<<1;
}
```

Penn ESE532 Fall 2020 -- De

45

Sequential

```
sum=0;
for (i=0;i<32;i++) {
    sum+=(0-(b%2)) & a;
    b=b>>1;
    a=a<<1;
}
```

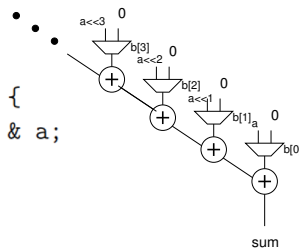


Penn ESE532 Fall 2020 -- DeHon

46

Spatial = fully unrolled

```
sum=0;
for (i=0;i<32;i++) {
    sum+=(0-(b%2)) & a;
    b=b>>1;
    a=a<<1;
}
```



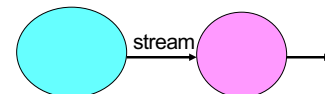
Penn ESE532 Fall 2020 -- DeHon

47

Stream

Day 5

- Logical abstraction of a persistent point-to-point communication link between operators
 - Has a (single) source and sink
 - Carries data presence / flow control
 - Provides in-order (FIFO) delivery of data from source to sink

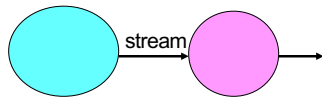


Penn ESE532 Fall 2020 -- DeHon

48

Stream

- For the moment assume way to read and write to streams:
 - stream.read() – return next value on stream
 - stream.write(val); put val onto stream

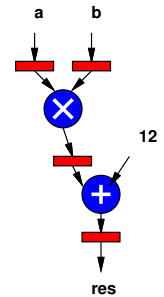


Penn ESE532 Fall 2020 -- DeHon

49

Unbounded, Pipelined Operator

What C code describe?



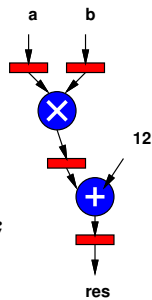
Penn ESE532 Fall 2020 -- DeHon

50

Unbounded, Pipelined Operator

What describe?

```
int c=12;
while(true)
{
    int aval=astream.read();
    int bval=bstream.read();
    int res=a*b+c;
    resstream.write(res);
}
```



Penn ESE532 Fall 2020 -- DeHon

51

With function call, loop in function

```
int c=12;
while(true)
{
    int aval=astream.read();
    int bval=bstream.read();
    int res=multiply(a,b)+c;
    resstream.write(res);
}
```

```
sum=0;
for (i=0;i<32;i++) {
    sum+=(0-(b%2)) & a;
    b=b>>1;
    a=a<<1;
}
```

Penn ESE532 Fall 2020 -- DeHon

52

Compact Expression: Arrays

- Useful to be able to refer to different values (a large number of values) with the same code.
- Arrays + Loops: give us a way to do that
- Useful:
 - general expression
 - hardware description

Penn ESE532 Fall 2020 -- DeHon

53

Compact Expression: Arrays+Logic

- Vector sum:
 - c3=a3+b3; c2=a2+b2; c1=a1+b1; c0=a0+b0;
 - for(i=0;i<3;i++) c[i]=a[i]+b[i];
- Chose small length to fit non-array on slide
 - #define K 16
 - for(i=0;i<K;i++) c[i]=a[i]+b[i];

Penn ESE532 Fall 2020 -- DeHon

54

Compact Expression: Arrays+Logic

- Dot Product:
 - $Y = a_3 \cdot b_3 + a_2 \cdot b_2 + a_1 \cdot b_1 + a_0 \cdot b_0$;
 - $Y = 0$; for($i=0; i < 3; i++$) $Y += a[i] \cdot b[i]$;

Penn ESE532 Fall 2020 -- DeHon

55

Compact Expression: Arrays+Logic

- Vector sum:
 - $c_3 = a_3 + b_3$; $c_2 = a_2 + b_2$; $c_1 = a_1 + b_1$; $c_0 = a_0 + b_0$;
 - for($i=0; i < 3; i++$) $c[i] = a[i] + b[i]$;
- These array elements may be nodes in dataflow graph, just like the variables we saw for function f
 - Express large dataflow graphs
 - Make area-time choices for implementation

Penn ESE532 Fall 2020 -- DeHon

56

Foreshadowing: C Array Challenge

- C programmers think of arrays as memory (or memory as arrays)
 - ...and sometimes we will want to
- Be careful understanding (and expressing) arrays that don't have to be memories
 - ...and treated with memory semantics

Penn ESE532 Fall 2020 -- DeHon

57

Loop Interpretations

- What does a loop describe?
 1. Sequential behavior [when to execute]
 2. Spatial construction [when create HW]
 3. Data Parallelism [sameness of compute]
- We will want to use for all 3
- Sometimes need to help the compiler understand which we want

Penn ESE532 Fall 2020 -- DeHon

58

Loop Bounds

- Loops without constant bounds

```
while (sum+a[i]<100) {
  sum+=a[i];
  b[i]=a[i]>>2;
  i++; }
```
- How many times loop execute?
- Typically forces sequentialization
 - Cannot unroll into hardware

Penn ESE532 Fall 2020 -- DeHon

59

Loop Increment

- Loops with variable increment also force sequentialization

```
for (i=0; i<100; i+=f(i))
  { b[i]=a[i]; sum+=a[i]; }
```
- What are values of i for which evaluate body?

Penn ESE532 Fall 2020 -- DeHon

60

Loop Interpretations

- What does a loop describe?
 - Sequential behavior [when execute]
 - Spatial construction [when create HW]
 - Data Parallelism [sameness of compute]
- We will want to use for all 3
- C allows expressive loops
 - Some expressiveness
 - Not compatible with spatial hardware construction

Penn ESE532 Fall 2020 -- DeHon

61

Unroll

- Vivado HLS has pragmas for unrolling
- UG901: Vivado HLS User's Guide
 - P180—229 for optimization and directives
- **#pragma HLS UNROLL factor=...**
- Use to control area-time points
 - Use of loop for spatial vs. temporal description

Penn ESE532 Fall 2020 -- DeHon

62

Part 4

Arrays: Memory and I/O
(time permitting)
Skip to Wrapup

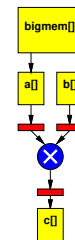
Penn ESE532 Fall 2020 -- DeHon

63

Arrays as Memory Banks

- Hardware expression: Sometimes we will want to describe computations with separate memory banks

```
int a[1024], b[1024],
    c[1024];
for (i=0; i<1024; i++)
    a[i]=bigmem[offset+i];
for (i=0; i<1024; i++)
    c[i]=a[i]*b[i];
```



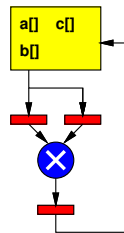
Penn ESE532 Fall 2020 -- DeHon

64

Arrays as Memory Banks

- If single memory has only one port
 - Can perform only one memory operation per cycle
 - What if if a, b, c all in bigmem?

```
for (i=0; i<1024; i++)
    c[i]=a[i]*b[i];
```



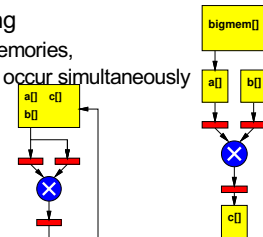
Penn ESE532 Fall 2020 -- DeHon

65

Physical Memory Port as Limited Shared Resource

- Typically single memory port
 - Must sequentialize on use of memory port
 - Reason for banking
 - Put in separate memories, so operations can occur simultaneously

DRAM 1 port
BRAM 2 ports



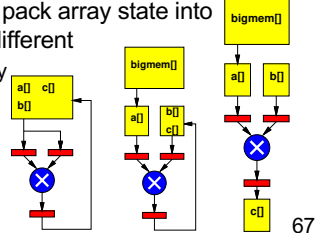
Penn ESE532 Fall 2020 -- DeHon

66

Arrays as things to put in Memory Banks

- Computational expression:
 - sometimes it is useful to express computation
 - **then** decide how to pack array state into memory banks for different

- Hardware availability
- Area-Time tradeoffs



Penn ESE532 Fall 2020 -- DeHon

67

Arrays as Inputs and Outputs

- Computational Expression: arrays are often a natural way of expression set of inputs and outputs

```
int c=12;
while(true)
{
    int aval=astream.read();
    int bval=bstream.read();
    int res=a*b+c;
    resstream.write(res);
}

void op(int a[BLOCK], int b[BLOCK], int out[BLOCK]) {
    for (i=0;i<BLOCK;i++)
    {
        out[i]=a[i]*b[i]+c;
    }
}
```

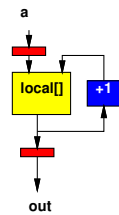
Penn ESE532 Fall 2020 -- DeHon

68

Arrays as Local Memory

- Hardware/Computational expression: natural way of describing local state

```
hist(int a[SIZE], out[EVENTS]) {
    int local[EVENTS];
    for(i=0;i<EVENTS;i++)
        local[i]=0;
    for(i=0;i<SIZE;i++)
        local[a[i]]++;
    for(i=0;i<EVENTS;i++)
        out[i]=local[i];
}
```



Penn ESE532 Fall 2020 -- DeHon

69

Array Interpretations

- What does an array describe?
 1. Compact expression [write less code]
 2. Memory banks [where place data]
 3. Things put in separate memory banks
 4. Local memory [not need to be shared]
 5. I/O [source and sink of data]
- We will want to use for all 5
- C allows expressive use of arrays/memories
 - Some expressiveness will inhibit efficient hardware

Penn ESE532 Fall 2020 -- DeHon

70

Part 5

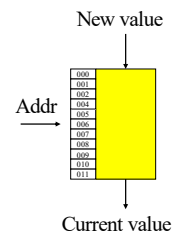
C Memory Model
(time permitting)
Skip to wrapup

Penn ESE532 Fall 2020 -- DeHon

71

C Memory Model

- One big linear address space of locations
- Most recent definition to location is value
- Sequential flow of statements

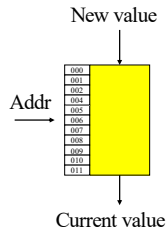


Penn ESE532 Fall 2020 -- DeHon

72

Challenge: C Memory Model

- One big linear address space of locations
- Assumes all arrays live in same memory
- Assumes arrays may overlap?

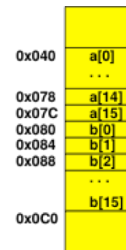


Penn ESE532 Fall 2020 -- DeHon

73

Example

- Assume a, b live in same memory
- Placed in sequence as shown
- What happens when
 - Read from a[17]
 - Read from b[-2]
- Can inhibit separation into memory banks, parallelism



Penn ESE532 Fall 2020 -- DeHon

74

Memory Operation Challenge

- Memory is just a set of location
- But **memory expressions** in C can refer to variable locations
 - Does A[i], B[j] refer to same location?
 - A[f(i)], B[g(j)] ?
- Can inhibit banking, parallelism
 - Or add expensive interconnect

Penn ESE532 Fall 2020 -- DeHon

75

C Memory/Pointer Sequentialization

- Must preserve ordering of memory operations
 - A read cannot be moved before write to memory which may redefine the location of the read
 - Conservative: any write to memory
 - Sophisticated analysis may allow us to prove independence of read and write
 - Writes which may redefine the same location cannot be reordered

Penn ESE532 Fall 2020 -- DeHon

76

C Memory/Pointer Sequentialization

- Must preserve ordering of memory operations
 - A read cannot be moved before write to memory which may redefine the location of the read
 - Writes which may redefine the same location cannot be reordered
- True for read/write to single array even if know arrays isolated
 - Does A[B[i]] refer to same location as A[C[i]]?
 - So expression issue broader than C

Penn ESE532 Fall 2020 -- DeHon

77

Consequence

- **Expressions and operations** through variables (whose address is never taken) can be executed at any time
 - Just preserve the dataflow
- **Memory assignments** must execute in strict order
 - Ideally: partial order
 - Conservatively: strict sequential order of C

Penn ESE532 Fall 2020 -- DeHon

78

More on Wednesday

- If time permits on Wednesday, more on Sequentialization and Dependencies

Penn ESE532 Fall 2020 -- DeHon

79

Big Ideas:

- C (any prog lang) specifies a computation
- Can describe spatial computation
 - Has some capabilities that don't make sense in hardware
 - Shared memory pool, globals, recursion
 - Watch for unintended sequentialization
- C for spatial is coded differently from C for processor
 - ...but can still run on processor
- Good for leaf functions (operations)
 - Limiting for full task

Penn ESE532 Fall 2020 -- DeHon

80

Admin

- Midterm on Wednesday
 - No lecture
 - Can take any 2 hour block in 24 hour period
 - See details on web
 - Previous midterms on web
 - Parts 1—3 today are relevant to exam
- HW5 due Friday 10/16
 - Several long compiles
 - Get started early

Penn ESE532 Fall 2020 -- DeHon

81