ESE535: Electronic Design Automation

Day 14: March 14, 2011
C→RTL

Today
See how to get from a language (C)
to dataflow
- Straight-line code
- If-conversion
- Memory
- Basic Blocks
- Control Flow
- Looping
- Hyperblocks
- Common Optimizations

Behavioral
(C, MATLAB, …)
Arch. Select
RTL
FSM assign
Two-level,
Multilevel opt.
Covering
Retiming
Gate Netlist
Placement
Routing
Layout
Masks

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

Comparison Operators
- Less Than a < b
- Less Than or Equal To a <= b
- Greater Than a > b
- Greater Than or Equal To a >= b
- Not Equal To a != b
- Equal To a == b
- Logical Negation !a
- Logical AND a && b
- Logical OR a || b

Things might have an a hardware operator for…

Domain Specific
RTL
GATE
Behavioral
Behavioral

Design Productivity by Approach

GATES/WEEK
(Scalable)
8K - 12K
2K - 10K
1K - 2K
100 - 200
10 - 20

Source: Keutzer (UCB EE 244)

Bitwise Operators
- Bitwise Left Shift a << b
- Bitwise Right Shift a >> b
- Bitwise One's Complement ~a
- Bitwise AND a & b
- Bitwise OR a | b
- Bitwise XOR a ^ b

Things might have an a hardware operator for…
Expressions: combine operators

- $a \times x + b$

A connected set of operators $\rightarrow$ Graph of operators

Expressions: combine operators

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

A connected set of operators $\rightarrow$ Graph of operators

C Assignment

- Basic assignment statement is:
  Location = expression
- $f = a \times x + b$

Straight-line code

- a sequence of assignments
  - What does this mean?
    - $g = a \times x$;
    - $h = b + g$;
    - $i = h \times x$;
    - $j = i + c$;

Variable Reuse

- Variables (locations) define flow between computations
- Locations (variables) are reusable
  - $t = a \times x$;
  - $r = t \times x$;
  - $t = b \times x$;
  - $r = r + t$;
  - $r = r + c$;

Variable Reuse

- Variables (locations) define flow between computations
- Locations (variables) are reusable
- $t = a \times x$; $r = a \times x$;
  - $t = b \times x$; $t = b \times x$;
  - $r = r + t$; $r = r + t$;
  - $r = r + c$; $r = r + c$;
- Sequential assignment semantics tell us which definition goes with which use.
  - Use gets most recent preceding definition.
Dataflow

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

\[
\begin{align*}
t &= a \times x; \\
&= a \times x; \\
t &= t \times x; \\
r &= t + x; \\
r &= r + t; \\
r &= r + c; \\
r &= r + c;
\end{align*}
\]

Dataflow Height

- Height (delay) of DF graph may be less than # sequential instructions.

Simple Control Flow

- If (cond) { ... } else { ... }

- Assignments become conditional

- In simplest cases, can treat as dataflow node

Simple Conditionals

\[
\text{if } (a > b) \\
\quad \text{c = b * c}; \\
\text{else} \\
\quad \text{c = a * c};
\]

Simple Conditionals

- If not assigned, value flows from before assignment

- If not assigned, value flows from before assignment

Simple Conditionals

- May (re)define many values on each branch.
Lecture Checkpoint

• Happy with
  – Straight-line code
  – Variables
  – Conditionals

• Next topic: Memory

C Memory Model

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

C Memory Operations

Read/Use
• a=*p;
• a=p[0]
• a=p[c*10+d]

Write/Def
• *p=2*a+b;
• p[0]=23;
• p[c*10+d]=a*x+b;

Memory Operation Challenge

• Memory is just a set of location
• But memory expressions can refer to variable locations
  – Does *q and *p refer to same location?
  – *p and q[c*10+d]?
  – p[0] and p[c*10+d]?
  – p[f(a)] and p[g(b)]?

Pitfall

• P[i]=23
• r=10+P[i]
• P[j]=17
• s=P[j]*12

• Value of r and s?
  …unless i==j
  Value of r and s?

C Pointer Pitfalls

• *p=23
• r=10+*p;
• *q=17
• s=*q*12;

• Similar limit if p==q
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

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

Forcing Sequencing
•Demands we introduce some discipline for deciding when operations occur
  – Could be a FSM
  – Could be an explicit dataflow token
  – Callahan uses control register
•Other uses for timing control
  – Variable delay blocks
  – Looping
  – Complex control

Scheduled Memory Operations

Day 3

Quadratic Memory Control
1. LD_X
2. MA_SEL=x, MB_SEL [1:0]=x, LD_x*x
3. MA_SEL=x, MB_SEL [1:0]=B
4. AB_SEL=C, MA_SEL=x*x, MB_SEL=A, LD_Bx+C
5. AB_SEL=Bx+C, LD_Y

Basic Blocks
•Sequence of operations with
  – Single entry point
  – Once enter execute all operations in block
  – Set of exits at end
A=B+C
E=A*D
If(E>100)
{
  Q++; 
  t=(E>100)
  br(BB2)
}
G=F*E
BB0: A=B+C
BB1: E=A*D
Q++; if(E>100) br(BB2)
G=F*E

Basic Blocks?
Basic Blocks

- Sequence of operations with
  - Single entry point
  - Once enter execute all operations in block
  - Set of exits at end
- Can dataflow schedule operations within a basic block
  - As long as preserve memory ordering

Connecting Basic Blocks

- Connect up basic blocks by routing control flow token
  - May enter from several places
  - May leave to one of several places

Loops

```c
sum=0;
for (i=0;i<imax;i++)
    sum+=i;
r=sum<<2;
```

Beyond Basic Blocks

- Basic blocks tend to be limiting
- Runs of straight-line code are not long
- For good hardware implementation
  - Want more parallelism
Hyperblocks

- Can convert if/then/else into dataflow
  - If/mux-conversion
- Hyperblock
  - Single entry point
  - No internal branches
  - Internal control flow provided by mux conversion
  - May exit at multiple points

Hyperblock Benefits

- More code $\rightarrow$ typically more parallelism
  - Shorter critical path
- Optimization opportunities
  - Reduce work in common flow path
  - Move logic for uncommon case out of path
    - Makes smaller faster

Common-Case Flow Optimization

Optimizations

- 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 $(i=0; i<100; i++) b[i]=(a&0xff+i)$;
Flow Review

Concerns

- Parallelism in hyperblock
  - Especially if memory sequentialized
  - Disambiguate memories?
  - Allow multiple memory banks?
- Only one hyperblock active at a time
  - Share hardware between blocks?
- Data only used from one side of mux
  - Share hardware between sides?
- Most logic in hyperblock idle?
  - Couldn’t we pipeline execution?

Pipelining

for (i=0;i<MAX;i++)
o[i]=(a*x[i]+b)*x[i]+c;

- If know memory operations independent

Summary

- Language (here C) defines meaning of operations
- Dataflow connection of computations
- Sequential precedents constraints to preserve
- Create basic blocks
- Link together
- Merge into hyperblocks with if-conversion
- Result is logic and registers \(\rightarrow\) RTL

Admin

- Assignment 5 out today
- Assignments 3, 4 graded
- Reading for Wednesday online
- Office hour tomorrow (Tuesday)
  - 5:40pm-6:30pm

Big Ideas:

- Semantics
- Dataflow
- Mux-conversion
- Specialization
- Common-case optimization