Estimation

- Estimates allow
  - Evaluation of design quality
  - Design space exploration

- Design model
  - Represents degree of design detail computed
  - Simple vs. complex models

- Issues for estimation
  - Accuracy
  - Speed
  - Fidelity
Outline

- Accuracy versus speed
- Fidelity
- Quality metrics
  - Performance metrics
  - Hardware and software cost metrics
Accuracy vs. Speed

- Accuracy: difference between estimated and actual value
  \[ \mathcal{A} = 1 - \frac{|E(D) - M(D)|}{M(D)} \]

- Speed: computation time for obtaining estimate

![Graph showing estimation error vs. computation time for simple model and actual design.](image)
Estimates must predict quality metrics for different design alternatives

Fidelity: % of correct predictions for pairs of design implementations

Higher fidelity $\rightarrow$ correct decisions based on estimates

\[
\begin{align*}
(A, B) &= E(A) > E(B), \ M(A) < M(B) \\
(B, C) &= E(B) < E(C), \ M(B) > M(C) \\
(A, C) &= E(A) < E(C), \ M(A) < M(C)
\end{align*}
\]

Fidelity = 33 %
Quality metrics

- **Performance Metrics**
  - Clock cycle, control steps, execution time, communication rates

- **Cost Metrics**
  - **Hardware**: manufacturing cost (area), packaging cost (pin)
  - **Software**: program size, data memory size

- **Other metrics**
  - Power, testability, design time, time to market
Hardware design model

Control Logic

State Reg.

Next-State Logic

Control Register

RF

R1

R2

Memory

DR

AR

MUXES

Datapath

Status bits

FU

Status Register

Control Unit

Memory

Registers/Register Files

Muxes

Functional Units
Clock cycle estimation

- Clock cycle determines:
  - Resources, execution time

- Determining clock cycle
  - Designer specified [PK89, MK90]
  - Maximum delay of any functional unit [PPM86, JMP88]
  - Clock utilization [NG92]
Clock slack and utilization

- **Slack**: portion of clock cycle for which FU is idle

\[
slack(clk, t_i) = (\left\lfloor \frac{delay(t_i)}{clk} \right\rfloor \times clk) - delay(t_i)
\]

- **Average slack**: FU slack averaged over all operations

\[
ave_{\text{slack}}(clk) = \frac{\sum_{i=1}^{T} \left[ \text{occur}(t_i) \times slack(clk, t_i) \right]}{\sum_{i=1}^{T} \text{occur}(t_i)}
\]

- **Clock utilization**: % of clock cycle utilized for computations

\[
\text{utilization}(clk) = 1 - \frac{ave_{\text{slack}}(clk)}{clk}
\]
Clock utilization

Clock = 65 ns

ave_slack(65 ns) = $\frac{6 \times 32}{6} + \frac{2 \times 9}{2} + \frac{2 \times 17}{2} = 24.4$ ns

utilization(65 ns) = $1 - \frac{24.4}{65.0} = 62\%$
Slack minimization algorithm

Clock Slack Minimization [NG92]

Compute range: $clk_{max}, clk_{min}$
Compute occurrences: $occur(t_i)$

$max_{utilization} = 0$
/* Examine each clock cycle in range */ for $clk_{min} \leq clk \leq clk_{max}$

loop

for all operation types $t_i \in T$ loop

Compute slack $slack(clk, t_i)$
end loop

Compute average slack: $ave_{slack}(clk)$
Compute utilization: $utilization(clk)$
/* If highest utilization */ if $utilization(clk) > max_{utilization}$

then

$max_{utilization} = utilization(clk)$
$max_{utilization, clk} = clk$

end if
end loop

$clk(SM) = max_{utilization, clk}$
Execution time vs. clock utilization

Second order differential equation example

- Clock with highest utilization results in better execution times

Clock cycle vs. Utilization

Execution time vs. utilization
Control steps estimation

- Operations in the specification assigned to control step

- Number of **control steps** determines:
  - Execution time of design
  - Complexity of control unit

- Scheduling
  - Granularity is operations in a dataflow graph
  - Computationally expensive
Operator-use method

- Granularity is statements in specification
- Faster than scheduling, average error 13%

<table>
<thead>
<tr>
<th>$t_i$</th>
<th>$\text{num}(t_i)$</th>
<th>$\text{clocks}(t_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>mult</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```
add: (1/1)*1= 1
mult: (4/2)*4= 4
```

```
max (1 ) = 1
```

Estimated total control steps = 14
Branching in behaviors

- Control steps maybe shared across exclusive branches
  - sharing schedule: fewer states, status register
  - non-sharing schedule: more states, no status registers
Execution time estimation

• Average start to finish time of behavior

• **Straight-line code behaviors**
  
  \[
  \text{exec}_{time}(B) = \text{csteps}(B) \times \text{clk}
  \]

• **Behavior with branching**
  
  - Estimate execution time for each basic block
  - Create control flow graph from basic blocks
  - Determine branching probabilities
  - Formulate equations for node frequencies
  - Solve set of equations
  
  \[
  \text{exec}_{time}(B) = \sum_{b_i \in B} \text{exec}_{time}(b_i) \times \text{freq}(b_i)
  \]
Probability-based flow analysis

A := A + 1;
for I in 1 to 10 loop
B := B + 1;
C := C - A;
if (D > A) then
  D := D + 2;
else
  D := D + 3;
end if;
E := D * 2;
end loop;
B := B * A;
C := 3;

\begin{align*}
A &:= A + 1; \\
\text{for } I \text{ in } 1 \text{ to } 10 \text{ loop} \\
B &:= B + 1; \\
C &:= C - A; \\
\text{if } (D > A) \text{ then} \\
D &:= D + 2; \\
\text{else} \\
D &:= D + 3; \\
\text{end if}; \\
E &:= D * 2; \\
\text{end loop}; \\
B &:= B * A; \\
C &:= 3;
\end{align*}
Probability-based flow analysis

- Flow equations:
  \[ \text{freq}(S) = 1.0 \]
  \[ \text{freq}(v_1) = 1.0 \times \text{freq}(S) \]
  \[ \text{freq}(v_2) = 1.0 \times \text{freq}(v_1) + 0.9 \times \text{freq}(v_5) \]
  \[ \text{freq}(v_3) = 0.5 \times \text{freq}(v_2) \]
  \[ \text{freq}(v_4) = 0.5 \times \text{freq}(v_2) \]
  \[ \text{freq}(v_5) = 1.0 \times \text{freq}(v_3) + 1.0 \times \text{freq}(v_4) \]
  \[ \text{freq}(v_6) = 0.1 \times \text{freq}(v_5) \]

- Node execution frequencies:
  \[ \text{freq}(v_1) = 1.0 \quad \text{freq}(v_2) = 10.0 \]
  \[ \text{freq}(v_3) = 5.0 \quad \text{freq}(v_4) = 5.0 \]
  \[ \text{freq}(v_5) = 10.0 \quad \text{freq}(v_6) = 1.0 \]

- Can be used to estimate number of accesses to variables, channels or procedures
Communication rates

- **Average channel rate**
  
  rate of data transfer over lifetime of behavior
  
  \[ \text{ averate}(C') = \frac{56 \text{ bits}}{1000 \text{ ns}} = 56 \text{ Mb/s} \]

- **Peak channel rate**
  
  rate of data transfer of single message
  
  \[ \text{ peakrate}(C') = \frac{8 \text{ bits}}{100 \text{ ns}} = 80 \text{ Mb/s} \]
Communication rate estimation

- Total behavior execution time consists of
  - Computation time, $\text{comptime}(P)$, obtained from ow-analysis
  - Communication time, $\text{commt}\text{ime}(P, C) = \text{access}(P, C) \times \text{delay}(C)$

- Total bits transferred by the channel,
  \[ \text{total bits}(P, C) = \text{access}(P, C) \times \text{bits}(C) \]

- Channel average rate
  \[ \text{averate}(C) = \frac{\text{total bits}(B, C)}{\text{comptime}(B) + \text{commt}\text{ime}(B, C)} \]

- Channel peak rate
  \[ \text{peakrate}(C) = \frac{\text{bits}(C)}{\text{protocol delay}(C)} \]
Area estimation

- Two tasks:
  - Determining number and type of components required
  - Estimating component size for a specific technology (FSMD, gate arrays etc.)

- Behavior implemented as a FSMD (finite state machine with datapath)
  - Datapath components: registers, functional units, multiplexers/buses
  - Control unit: state register, control logic, next-state logic

- We will discuss
  - Datapath component estimation
  - Control unit estimation
  - Layout area for a custom implementation
Clique-partitioning

- Commonly used for determining datapath components

- Let $G = (V, E)$ be a graph, $V$ and $E$ are set of vertices and edges

- Clique is a complete subgraph of $G$

- Clique-partitioning
  - divides the vertices into a minimal number of cliques
  - each vertex in exactly one clique

- One heuristic: maximum number of common neighbors [CS86]
  - Two nodes with maximum number of common neighbors are merged
  - Edges to two nodes replaced by edges to merged node
  - Process repeated till no more nodes can be merged
Clique-partitioning

Clique 1: \( \{v_1, v_3, v_4\} \)
Clique 2: \( \{v_2, v_5\} \)

Common neighbors:
- \( e'_{1,3} \): 1
- \( e'_{1,4} \): 1
- \( e'_{2,3} \): 0
- \( e'_{2,5} \): 0
- \( e'_{3,4} \): 1
- \( e'_{4,5} \): 0

Edges:
- \( s_{134} \)
- \( s_{25} \)
Storage-unit estimation

- Variables not used concurrently maybe mapped same storage-unit

- To use clique-partitioning, construct a graph where
  - Each variable represented by a vertex
  - Variables with non-overlapping lifetimes have an edge between their vertices

\[
\begin{align*}
\{v_2, v_3\} &= R_1 \\
\{v_6, v_7, v_9\} &= R_2 \\
\{v_4, v_5, v_8\} &= R_3 \\
\{v_{10}, v_{11}\} &= R_4 \\
\{v_1\} &= R_5
\end{align*}
\]
Functional-unit and interconnect-unit estimation

- Clique-partitioning can be applied

- For determining the number of FU’s required, construct a graph where
  - Each operation in behavior represented by a vertex
  - Edge connects two vertices if
    - Corresponding operations assigned different control steps
    - There exists an FU that can implement both operations

- For determining the number of interconnect units, construct a graph where
  - Each connection between two units is represented by a vertex
  - Edge connects two vertices if corresponding connections not used
    in same control step
Computing datapath area

- Bit-sliced datapath

\[ L_{bit} = \alpha \times tr(DP) \]

\[ H_{rt} = \frac{nets}{nets\_per\_track} \times \beta \]

\[ area(bit) = L_{bit} \times (H_{cell} + H_{rt}) \]

\[ area(DP) = \text{bitwidth}(DP) \times area(bit) \]
Pin estimation

- Number of wires at behavior’s boundary depends on
  - Global data
  - Port accessed
  - Communication channels used
  - Procedure calls

```plaintext
variable N : integer;
variable X : bit_vector(15 downto 0);

procedure SUM(A, B, OUT) is
begin
  SUM;
end procedure SUM;

process Main (ch1, ch2)
  out channel ch1;
  in  channel ch2;
  
  send (ch1, N);
  portF <= portG + 4;
  ...
  receive (ch2, Result);

process Factorial (ch1, ch2)
  in  channel ch1;
  out channel ch2;
  
  receive (ch1, M);
  /* compute factorial */
  ...
  send (ch2, result);
```

```plaintext
channel ch1

channel ch2
```

```plaintext
portF

portG
```
Software estimation models

Processor specific model

Generic model

Specification

Compile to 8086

Compile to 68000

Compile to MIPS

8086 instructions

68000 instructions

MIPS instructions

8086 Estimator

68000 Estimator

MIPS Estimator

Software Metrics

8086 instruction timing & size information

68000 instruction timing & size information

MIPS instruction timing & size information

Estimator

Generic instructions

technology files for target processors

Software Metrics

8086 instruction timing & size information

68000 instruction timing & size information

MIPS instruction timing & size information
### Deriving processor technology files

#### Generic instruction

\[ \text{dmem3} = \text{dmem1} + \text{dmem2} \]

#### 8086 instructions

<table>
<thead>
<tr>
<th>instruction</th>
<th>clocks</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov ax, word ptr[bp+offset1]</td>
<td>(10)</td>
<td>3</td>
</tr>
<tr>
<td>add ax, word ptr[bp+offset2]</td>
<td>(9 + EA1)</td>
<td>4</td>
</tr>
<tr>
<td>mov word ptr[bp+offset3], ax</td>
<td>(10)</td>
<td>3</td>
</tr>
</tbody>
</table>

#### 68020 instructions

<table>
<thead>
<tr>
<th>instruction</th>
<th>clocks</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov a6@(offset1), d0</td>
<td>(7)</td>
<td>2</td>
</tr>
<tr>
<td>add a6@(offset2), d0</td>
<td>(2 + EA2)</td>
<td>2</td>
</tr>
<tr>
<td>mov d0, a6@(offset3)</td>
<td>(5)</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Technology file for 8086

<table>
<thead>
<tr>
<th>generic instruction</th>
<th>execution time</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ldots</td>
<td>35 clocks</td>
<td>10 bytes</td>
</tr>
<tr>
<td>dmem3 = dmem1 + dmem2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Technology file for 68020

<table>
<thead>
<tr>
<th>generic instruction</th>
<th>execution time</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ldots</td>
<td>22 clocks</td>
<td>6 bytes</td>
</tr>
<tr>
<td>dmem3 = dmem1 + dmem2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Software estimation

- **Program execution time**
  - Create basic blocks and compile into generic instructions
  - Estimate execution time of basic blocks
  - Perform probability-based flow analysis
  - Compute execution time of the entire behavior:
    \[
    \text{exectime}(B) = \delta \times \left( \sum_{b_i \in B} \text{exectime}(b_i) \times f_{req}(b_i) \right)
    \]
    \[
    \delta \text{ accounts for compiler optimizations}
    \]

- **Program memory size**
  \[
  \text{progsize}(B) = \sum_{g \in G} \text{instr\_size}(g)
  \]

- **Data memory size**
  \[
  \text{datasize}(B) = \sum_{d \in D} \text{datasize}(d)
  \]
Summary and future directions

- We described methods for estimating:
  - Performance metrics: clock, control steps, execution time, communication rates
  - Cost metrics: design area, pins, program and data memory size

- Future directions:
  - Incorporating synthesis/compilation optimizations
  - New metrics for testability, power, integration cost, etc.
  - New architectural features for the estimation model