Computer science and engineering sampler

• You’ve gotten a taste of computer science and engineering, but there’s a lot more:
  – Achieve the same functionality while using fewer resources
  – Achieve better results with the same resources
  – Achieve results that one would think impossible
  – Find that some things that appear possible are impossible

Logic minimization

• We showed one way how to express a combinational logic function (sum of products). Can we implement the same function with much less hardware?
Comparing two 2-bit numbers

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>B1</th>
<th>B0</th>
<th>A&lt;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Standard form to express this combinational logic function
- Can simplify to
Pipelining

- Remember E100 datapath
  - 15 cycles to execute one instruction
  - We can execute the same ISA about 5 times faster, without much extra hardware
- E100 datapath
  - Datapath components are used inefficiently
  - Only a few parts of the circuit are active at one time
  - Limited by bus
- Pipelining
  - Split the bus into several sections
  - Allow multiple datapath components to be used at the same time
  - Overlap the execution of multiple instructions

Caching

- SDRAM vs. E100 memory

- Caching combines slow, cheap, big memory with fast, expensive, small memory
Error correcting codes

• We’ve assumed data is stored perfectly
  – But what if memory sometimes developed errors?
  – E.g., you stored 11002, but the memory spontaneously changed this to 11012
• Can I detect and correct these errors by storing extra information?
• Does it work to store the number twice (100% overhead)?

• How many extra bits does it take to enable error correction?

Data structures

• Consider writing a program to sort a set of values

for n iterations {
    find the maximum in the remaining values [this takes about n iterations]
    append the max to the set of sorted values
    delete the max from the original set
}
• How many iterations total, e.g. for a set of $10^6$ elements?

• How fast can we do the same task?
Recursion

• Some mathematical functions are defined recursively (in terms of themselves). E.g., factorial(n) =
  \[
  \begin{align*}
  &1 \quad \text{if } n == 0 \\
  &n \times \text{factorial}(n-1) \quad \text{if } n > 0
  \end{align*}
  \]
• We can write programs in the same style

```c
int factorial(int n) {
    if (n == 0) {
        return(1);
    } else {
        return(n * factorial(n-1));
    }
}
```
• Will this work with our E100 programming style?

Parallel programming

• E100 isn’t fast enough for all applications. What if you had multiple E100 processors?
• Parallel programming is important
  – Most modern microprocessor chips have multiple “cores”
• Parallel programming is hard
• A simple program:
  ```
  add x y num1
  sub y x num0
  ```
• What are the final values of x and y?
• What if each instruction ran on a separate E100?
Halting problem

• You’ve implemented programs to accomplish various tasks, e.g. password checker, educational toy.
• What if I asked you to write a program P that analyzed another program X and tried to figure out if X would ever finish (i.e. halt)?
• This one is pretty easy to analyze:

```cpp
main()
{
    for (int i=0; i<10; i++) {
        cout << i;
    }
}
```
• So is this one:

```cpp
main() {
    for (int i=0; i<10; ) {
        cout << i;
    }
}
```
• One can imagine writing a program that could analyze the above two programs and determine that the first program halts, while the second program doesn’t halt.
• Can you write a program that will analyze ANY program and decide if it will ever halt? Can anyone write such a program, given sufficient time and expertise?
Self-reproducing program

• Can you write a program that, when run, will output its own source code?
• This one is close, but not quite

```cpp
#include <iostream>

main() {
    std::cout << "#include <iostream>
main() \n{std::cout}\n";
}
```
• What output does the above program produce?

• Is it possible to write such a program?

---

Virtualization

• We provided a single processor, small memory, and limited I/O
• How can we make it look like (to a program) that there is an arbitrary number of processors, memory, network cards (without adding more hardware)?
Many more interesting topics

- How does the Cyclone II FPGA implement arbitrary combinational and sequential logic?
- How to design and verify a system with a billion components?
- How to insert a new sound sample in our set without shifting the others down?
- How to allow multiple users to use the same computer, share the same hardware, yet prevent them from messing up each other's data?
- How to send messages securely over a communication channel that an attacker controls (can eavesdrop, modify, delete, or insert messages)?
- How to make a bunch of computers look like a single, more powerful one?