Quick Points

<table>
<thead>
<tr>
<th>Sunday</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
<th>Saturday</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>26</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Lect-25</td>
<td></td>
<td></td>
<td></td>
<td>Lect-26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Dead Week</td>
<td></td>
<td>Project Seminars (EDE)</td>
<td></td>
<td>Project Seminars (Others)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>Finals Week</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Project Write-ups Deadline</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td></td>
<td>Electronic</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grade Due</td>
<td></td>
<td>December / November 2006</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Celoxica Handel-C

- Handel-C adds constructs to ANSI-C to enable hardware implementation
  - Synthesizable HW programming language based on C
  - Implements C algorithm direct to optimized FPGA or RTL

- Software-only ANSI-C constructs supported by DK
- Handel-C
  - Adds constructs to enable optimization of timing and area performance
  - Systems described in ANSI-C can be implemented in software and hardware using language extensions defined in Handel-C to describe hardware
  - Extensions focused towards areas of parallelism and communication

Variables

- Handel-C has one basic type - integer
- May be signed or unsigned
- Can be any width, not limited to 8, 16, 32 etc.

Variables are mapped to hardware registers

```c
void main(void)
{
    unsigned u a;
    a=45;
}
```

```plaintext
a = 1 0 1 1 1 0 1 = 0x2da
```

Timing Model

- Assignments and delay statements take 1 clock cycle
- Combinatorial Expressions computed between clock edges
  - Most complex expression determines clock period
  - Example: takes 1+n cycles (n is number of iterations)

```c
index = 0;
while (index < length){ // 1 Cycle
    if(table[index] = key)
    found = index; // 1 Cycle
    else
    index = index+1; // 1 Cycle
}
```
Parallelism

- Handel-C blocks are by default sequential
- `par{...}` executes statements in parallel
- Par block completes when all statements complete
  - Time for block is time for longest statement
  - Can nest sequential blocks in par blocks
- Parallel version takes 1 clock cycle
  - Allows trade-off between hardware size and performance

Channels

- Allow communication and synchronization between two parallel branches
- Semantics based on CSP (used by NASA and US Naval Research Laboratory)
- Unbuffered (synchronous) send and receive
- Declaration
  - Specifies data type to be communicated
  
  ```
  Chan unsigned 6 c;
  ```

Signals

- A signal behaves like a wire - takes the value assigned to it but only for that clock cycle
- The value can be read back during the same clock cycle
- The signal can also be given a default value

  ```
  // Breaking up complex expressions
  int 15 a, b;
  signal <int> sig1;
  static signal <int> sig2=0;
  a = 7;
  par
  { sig1 = (a*34)*17;
    sig2 = (a<2)+2;
    b = sig1 + sig2;
  }
  ```

Sharing Hardware for Expressions

- Functions provide a means of sharing hardware for expressions
- By default, compiler generates separate hardware for each expression
  - Hardware is idle when control flow is elsewhere in the program
  - Hardware function body is shared among call sites

```
int mult_add(int z,c1,c2){
  return z*c1 + c2;
}
```

```
x= mult_add(x,a,b);
y= mult_add(y,c,d);
```

Bit-width Analysis

- Higher Language Abstraction
  - Reconfigurable fabrics benefit from specialization
  - One opportunity is bitwidth optimization
- During C to FPGA conversion consider operand widths
  - Requires checking data dependencies
  - Must take worst case into account
  - Opportunity for significant gains for Booleans and loop indices
- Focus here is on specialization

Arithmetic Analysis

- Example

```
int a;
unsigned b;
a = random();
b = random();
a = a / 2;
b = b >> 4;
a = random() & 0xff;
```

```
a: 32 bits b: 32 bits
a: 31 bits b: 32 bits
a: 31 bits b: 28 bits
a: 8 bits b: 28 bits
```
**Loop Induction Variable Bounding**
- Applicable to for loop induction variables.
- Example
  ```
  int i;
  i: 32 bits
  for (i = 0; i < 6; i++) {
    i: 3 bits
  }
  i: 3 bits
  ```

**Clamping Optimization**
- Multimedia codes often simulate saturating instructions.
- Example
  ```
  int valpred
  valpred: 32 bits
  if (valpred > 32767)
    valpred = 32767
  else if (valpred < -32768)
    valpred = -32768
  valpred: 16 bits
  ```

**Solving the Linear Sequence**
- Sum all the contributions together, and take the data-range union with the initial value.
- Can easily find conservative range of <0,510>
  ```
  a = 0 <0,0>
  for i = 1 to 10
    a = a + 1 <1,460>
  for j = 1 to 10
    a = a + 2 <3,480>
  for k = 1 to 10
    a = a + 3 <24,510>
  ... = a + 4 <510,510>
  ```

**FPGA Area Savings**
- Graph showing area savings between bitwise and non-bitwise operations.

**Summary**
- High-level is still not well understood for reconfigurable computing.
- Difficult issue is parallel specification and verification.
- Designers efficiency in RTL specification is quite high. Do we really need better high-level compilation?

**Some Emerging Technologies**
- Several emerging technologies may make an impact:
  - Carbon nanotubes
  - Magnetoelectronic devices
- Technologies are in their infancy.
**Carbon Nanotubes**
- Extensions of carbon molecules
- Grown as long straight tubes
- "Flow" used to align nanotubes in a specific direction
- Technology still in infancy

**Bottom-Up Self-Assembly**
- We can’t make nano-circuits top-down
  - Lithography can’t get to the nano scale
- Make them bottom-up with chemical self-assembly
  - Their own physical properties keep them in regular order, much like crystals do when they grow
- Fluid flow self-assembly
  - Crossbar generated in two passes

**Nanotubes in Electronics?**
- Carbon nanotubes come in two flavors:
  - Metallic
  - Semiconducting
- Metallic nanotubes make great wires
- Semiconducting nanotubes can be made into transistors
- Depending on how nanotubes are formed, range from about 1/3 semiconducting, 2/3 metallic to 2/3 semiconducting, 1/3 metallic
- No good technology at present time for creating nanotubes of just one type

**Possible Devices**
- Diode connection formed by making connection between upper and lower nanotube
  - Nanotubes do not touch when forming a FET
    - Top nanotube covered with oxide
    - Effectively acts as a “gate” to current path

**Diode Logic**
- Arise directly from touching NW/NTs
  - Passive logic
  - Non-restoring

**PMOS-like Restoring FET Logic**
- Use FET connections to build restoring gates
  - Static load
    - Like NMOS (PMOS)
Programmed FET Arrays

- Black squares show blocked (separate) wires (no FET gating)
- \( \text{out1} = -(\text{in1} + \text{in3}) \)
- \( \text{out2} = -(\text{in1} + \text{in2}) \)
- \( \text{VPd (static load)} \)
- \( \text{Vdd} \)
- \( \text{Gnd} \)

Programmable OR-plane

- Addressing is a challenge since order of addresses can’t be predetermined
- Nanotubes can be doped to form different addresses
- Some redundancy OK
- Diode logic formed at crosspoint

Simple Nanowire-Based PLA

- NOR-NOR = AND-OR PLA Logic

Defect Tolerance

- All components (PLA, routing) interchangeable; Allows local programming around faults

Results [Deh05A]

- Pair of 60-term OR planes roughly same size as 4-LUT
- Special mapping and programming tools needed
- Fault tolerance a big issue

Magnetoelectronic Devices

- Program a cell by setting a directional magnetic field
  - Programming current sets field
  - Technique already heavily using in storage devices
- Flexible, reliable
- Advantages:
  - Non-volatile
  - Low power consumption
**HHE Devices**

- Information written as magnetization states by passing a write current through a current line
- HIGH, and LOW output Hall voltage according to direction of magnetization.
- Good remanence in the ferromagnet may lead to hysteresis loop and hence memory
- Easily integrated with rest of the CMOS circuit

**Magnetoelectronic Gates**

- Use storage cell along with a minimum of external transistors to create logic
- External circuitry induces current which can program cell

**Power Reducing**

- Logic only evaluated if the output result will change state
- If change redetected then perform reset
- Otherwise, maintain old value

**Magnetoelectronic Look-up Tables**

- SRAM storage cell used for high performance
- Initial value of SRAM cell stored in magnetoelectronic cell
- Cell is programmed following reset

**Summary**

- Difficult to explore without experts in physics and chemistry
- Initial architecture ideas based on perceptions of likely available technology
- Daunting challenges involving CAD and power reduction remain
- Not likely to have much commercial application for 10-15 years
- Active area of research