CatapultC Synthesis

Dhruv Jain
Brief Explanation

The image shows a software interface for a design tool, likely used for digital design or hardware description language (HDL) development. The interface includes a task bar with options such as Add Input Files, Setup Design, Architecture Constraints, Resource Constraints, Schedule, and Generate RTL. There is also a section displaying project files, with a selected file named `sort.v4` marked as a Passed Extract. The design editor shows a diagram with nodes labeled `sort`, `core`, and `interface`, indicating parts of a hardware or software design. The transcript area at the bottom shows log messages about reading component libraries from specified paths.
Selection Sort

- Selection Sort code for 8 numbers was written in CatapultC.
  - Output simulated with Xilinx 12.1 ise.
  - All other parameters were kept at their default values.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board</td>
<td>Virtex 4</td>
</tr>
<tr>
<td>RAM</td>
<td>Single port Dist. RAM</td>
</tr>
<tr>
<td>Frequency</td>
<td>100Mhz</td>
</tr>
<tr>
<td>Minimum Input Arrival Time</td>
<td>7.007ns*</td>
</tr>
<tr>
<td>Maximum output time after time</td>
<td>10.999ns*</td>
</tr>
<tr>
<td>Primary Goal</td>
<td>Latency</td>
</tr>
</tbody>
</table>

*these parameters not taken care of when compiling with CatapultC so the design had to be modified again. The results remain nearly the same.

** further optimized
Simulation with ISE 12.1
Results for Selection Sort

<table>
<thead>
<tr>
<th>Parameter</th>
<th>CatapultC</th>
<th>Xilinx Ise 12.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Frequency</td>
<td>100Mhz (~)</td>
<td>149.5390Mhz**</td>
</tr>
<tr>
<td>Critical Path Delay</td>
<td>10.882ns</td>
<td>6.678ns</td>
</tr>
<tr>
<td>Total Cycles needed for appearance of output</td>
<td>138</td>
<td>138</td>
</tr>
<tr>
<td>Slices/Area</td>
<td>Post DP and FSM: 7%</td>
<td>7%</td>
</tr>
<tr>
<td></td>
<td>Post Assignment: 13%</td>
<td></td>
</tr>
</tbody>
</table>
Results for Selection Sort (CatapultC)

- Comparison of results with loop kept rolled (default) and full loop unrolling for 8 numbers

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Without unroll</th>
<th>With unroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time</td>
<td>430ns</td>
<td>1.42us</td>
</tr>
<tr>
<td>Cycles</td>
<td>43</td>
<td>138</td>
</tr>
<tr>
<td>Area Score</td>
<td>297.3 (59%, 30%, 11%)*</td>
<td>2356.0 (70%, 18%, 12%)*</td>
</tr>
</tbody>
</table>

- Results are not according to expectations
  - Possible reason: for small datapath, major time consumed by FSM

*Percentage used by (MUX, Func, Logic)
Results for Selection Sort (CatapultC)

- Comparison of results with loop kept rolled (default) and full loop unrolling for 99 numbers

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Without unroll</th>
<th>With unroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time</td>
<td>6.87us</td>
<td>5.42us</td>
</tr>
<tr>
<td>Area Score</td>
<td>365.0 (59%, 30%, 11%)</td>
<td>27228.7 (73%, 9%, 11%)</td>
</tr>
</tbody>
</table>

- These results show that our explanation was indeed correct.
  - Datapath has increased -> Less contribution from FSM
Sorting Results for different sort

- All sorting algorithms manually coded
  - Possible lack of optimization
  - All are without unroll and primary goal was latency. Numbers = 99
Sorting Results for different sorts

- Memory results were as per the expectations
Radix Sort

```c
#define MAX 99

void radixsort(int a[MAX])
{
    int i, b[MAX], m=0, exp=1;

    for(i=0; i<MAX; i++)
    {
        if(a[i]>m)
            m=a[i];
    }

    while(m/exp>0)
    {
        int bucket[10]={0};
        for(i=0; i<MAX; i++)
            bucket[a[i]/exp%10]++;
        for(i=1; i<10; i++)
            bucket[i]=bucket[i-1];
        for(i=MAX-1; i>=0; i--)
            b[--bucket[a[i]/exp%10]]=a[i];
        for(i=0; i<MAX; i++)
            a[i]=b[i];
        exp*=10;
    }
}
```
void radixsort(ac_int<8, false> a[MAX])
{
    ac_int<8, false> i;
    ac_int<8, false> m = 99;
    ac_int<9, false> exp = 1;
    ac_int<1, false> flag = 0;

    /* m can be kept as fixed */

    while (m/exp>0)
    {
        int bucket[2]={0};
        //Only two buckets for each bit (2 base)
        //Bucket is used as an index so its type is kept as int to prevent type casting again

        for (i=0;i<MAX;i++)
            bucket[a[i] || exp]++;
        //Division replaced by bitwise OR (for saving type conversions)

        //for(i=1;i<10;i++)
        //No need for this loop

        bucket[1]+=bucket[0];

        for (i=MAX-1;i>=0;i--)
            b[---bucket[a[i] || exp]]=a[i];
        //for(i=0;i<MAX;i++)
        //No need for assignment. Just reuse by copy/pasting the code

        exp = (ac_int<9, false>) exp << 1; //Multiplication replaced by left shift
if(m/exp > 0) //If MAX is under even multiple of 2, this can be removed too!
{
    bucket[1]=bucket[0] = {0};
    for(i=0;i<MAX;i++)
        bucket[b[i] || exp]++;

    //for(i=1;i<10;i++)

    bucket[1]=bucket[0];

    for(i=MAX-1;i>=0;i--)
        a[--bucket[b[i] || exp]]=b[i];
}

    //for(i=0;i<MAX;i++)
-    // a[i]=b[i];
    exp = (ac_int<9, false>) exp << 1;
}
else    flag = 1;
}

if(flag == 1) // Copy at the end if required. Wouldn't make much difference in time
for(i=0;i<MAX;i++)
a[i]=b[i];
}
Comparison with VHDL implementation of systolic sort @DHD Project

- **Xilinx Ise 9.1 results**
  - Vertix II, Single Port RAM
  - Clock frequency: 162.824Mhz
  - Critical Path delay: 6.142ns
  - Minimum input arrival time before clock: 1.802ns
  - Maximum output required time after clock: 3.473ns

- These parameters were filled for CatapultC optimized radix sort

- **Results**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Xilinx 9.1 Ise</th>
<th>CatapultC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time for appearance of the entire output</td>
<td>6.14us</td>
<td>9.21us</td>
</tr>
</tbody>
</table>
Simulation for Systolic Sort ISE 9.1
Conclusions

- CatapultC is a great tool for HLS since it supports variable bit length.
- Various possible configuration options for optimisation
  - Optimization Parameter (Area/Delay)
  - Interface (RAM/Registers)
  - Effort Level
  - Loop unrolling
  - Clock manipulation
  - Power Efficient Design (Clock Gating)
  - Input/Output Delays
  - Handshaking signals
- “Scheduling” window for verification
- But, still manual coding with HDL is most efficient.
References

  ▶ Data clustering for bioinformatics using parallel merge sort

  ▶ Available inside CatapultC Tool’s help section

Special thanks to Anand Isaac