Behavioral SystemC Implementation of an MP3 Decoder

F. Goldstein, G. Araujo and R. Azevedo

felipe.goldstein @ students.ic.unicamp.br
{guido, rodolfo} @ ic.unicamp.br

Speaker: Felipe Portavales Goldstein

Computer Systems Laboratory
IC-UNICAMP

Overview

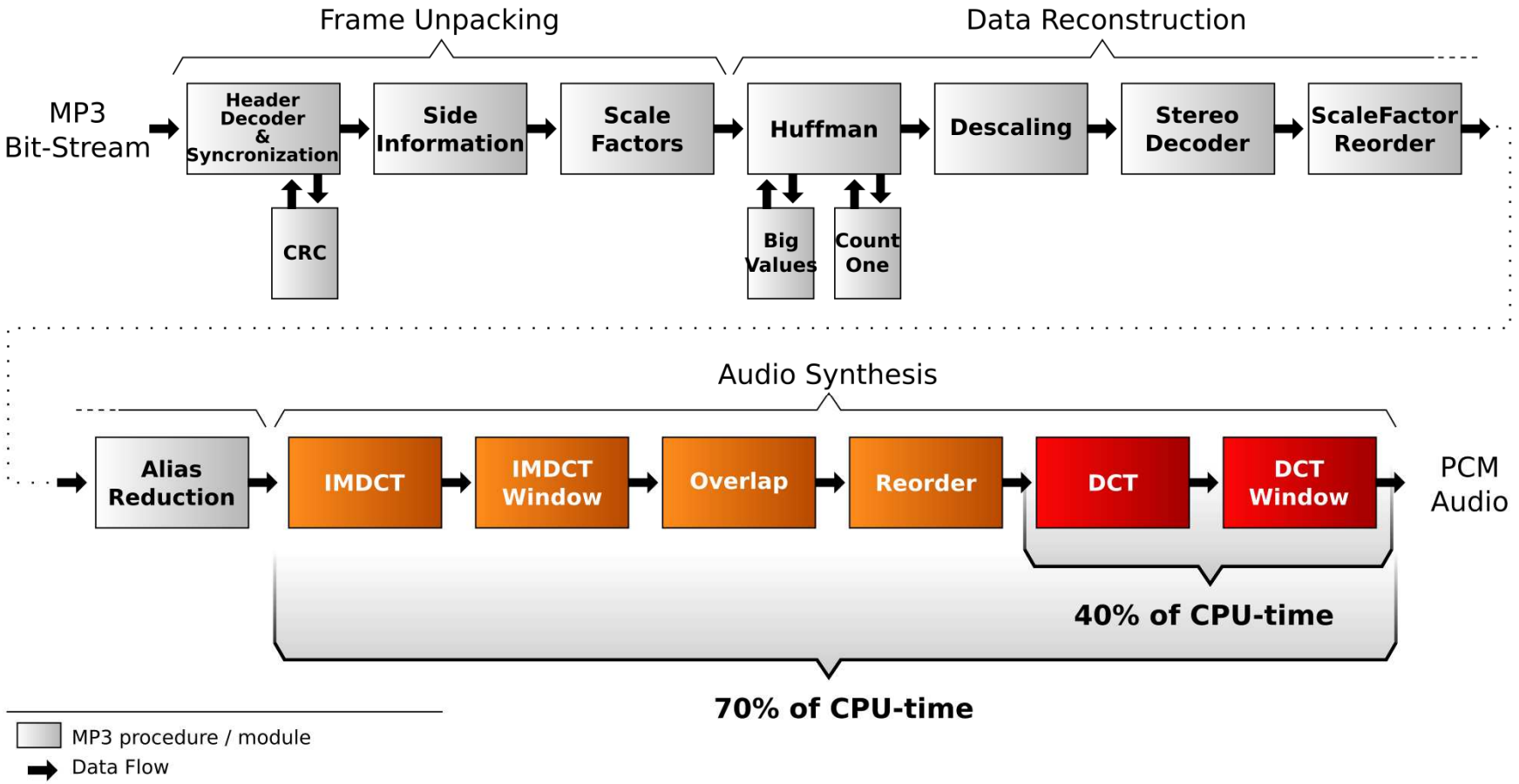
- MP3
- Behavioral Synthesis
- Software Reference Model
- Design Exploration
- Results and Comparisons

Project

- Hardware implementation of an MP3 decoder fully designed in both Behavioral and RTL SystemC
  - Independent implementations
- Goals
  - Compare Behavioral Synthesis and Classical RTL based design
  - Achieve a good design space exploration
  - Compare different software reference design implementations

MP3 : MPEG-1 Layer III *(ISO 11172-3)*

- Decoding Pipeline:



Why use Behavioral Synthesis ?

- Design space exploration
  - Different implementations can achieve different trade-offs
- Implementation closer to Software
  - Easier to code and debug
- Reduce time to market

LibMad *Vs* ISO Reference Code

- LibMad
  - A highly optimized MP3 decoding library
  - Uses fixed point math
- ISO Reference Code
  - Very well documented
  - Didactic Purpose
- We want the simplest to refine to hardware
  - Is optimized software a good starting point for a hardware design ?

Why optimized Software is not good for Hardware

- Common software optimizations
  - Store pre-calculated values in RAM
    - RAM size is bigger than size of logic to make calculations on-the-fly
    - RAM latency is longer than logic latency
  - Pointers usage
    - Breaks hardware parallelism
    - Confuse Data-Flow
    - Less modularity
  - Control Logic
    - Can lead to a penalty on performance
    - Data-flow dominant designs are easier to be scheduled
    - The hardware to perform operations is synthesized

Clean Software

- Easy to understand
  - Easy to be modified
    - Allows different design implementations
- Modularized
  - Helps on synthesis
- Well defined Data-Flow
  - Helps on operation scheduling

Design Exploration

- Concentrate most effort in the critical part
  - A profiling can show where the critical part is
  - In the case of MP3, this is the DCT module
- A small modification in the code can generate completely different synthesis results

Design Exploration

- Good code modifications
  - Unroll the inner loops of critical modules

```
for( i=0; i < 64; i++){ // Big loop
      for( k=0; k < 32; k++){ // Small inner loop
            //some small piece of code
      }
      //some big piece of code
}
```

Loop Unrolling

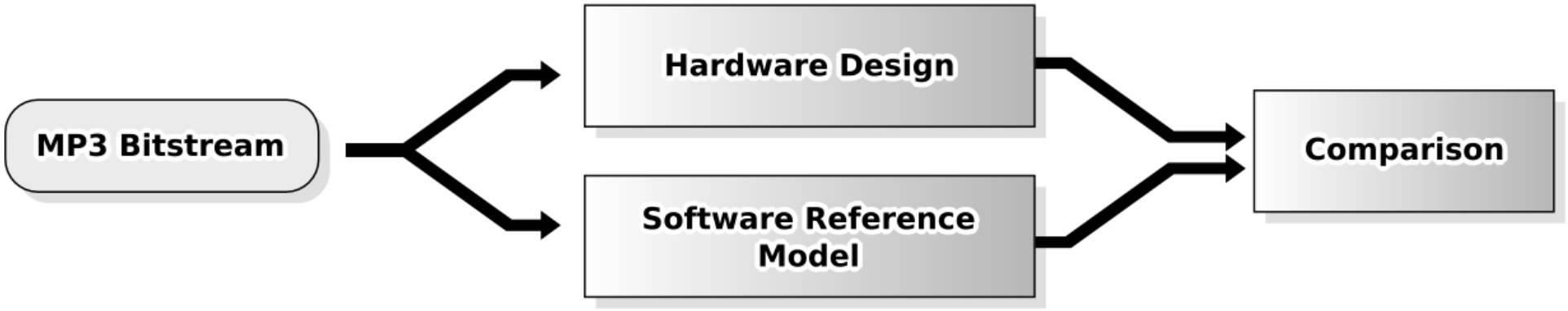
- Four small inner loops in critical modules
  - A latency reduction of **53%**
    - (From 541 to 243 cycles per sample)
  - An increase in area of only **6%**
- Same unrolling with ASAP scheduling
  - A latency reduction of 67%
  - An area increase of 40%

Loop Pipelining

- Four small inner loops in critical modules
  - A latency reduction of **42%**
    - (From 541 to 294 cycles per sample)
  - An increase in area of **34%**
- Same pipelining with ASAP scheduling
  - A latency reduction of 65%
  - An area increase of 60%

Verification Method

- Each design point was validated after synthesis
- 66 different Bitstreams to cover all corner cases
  - Including the ISO compliance test bitstreams



Behavioral vs RTL Comparison

- A single designer within a period of 3 months produced 14 design points using the Forte Cynthesizer tool

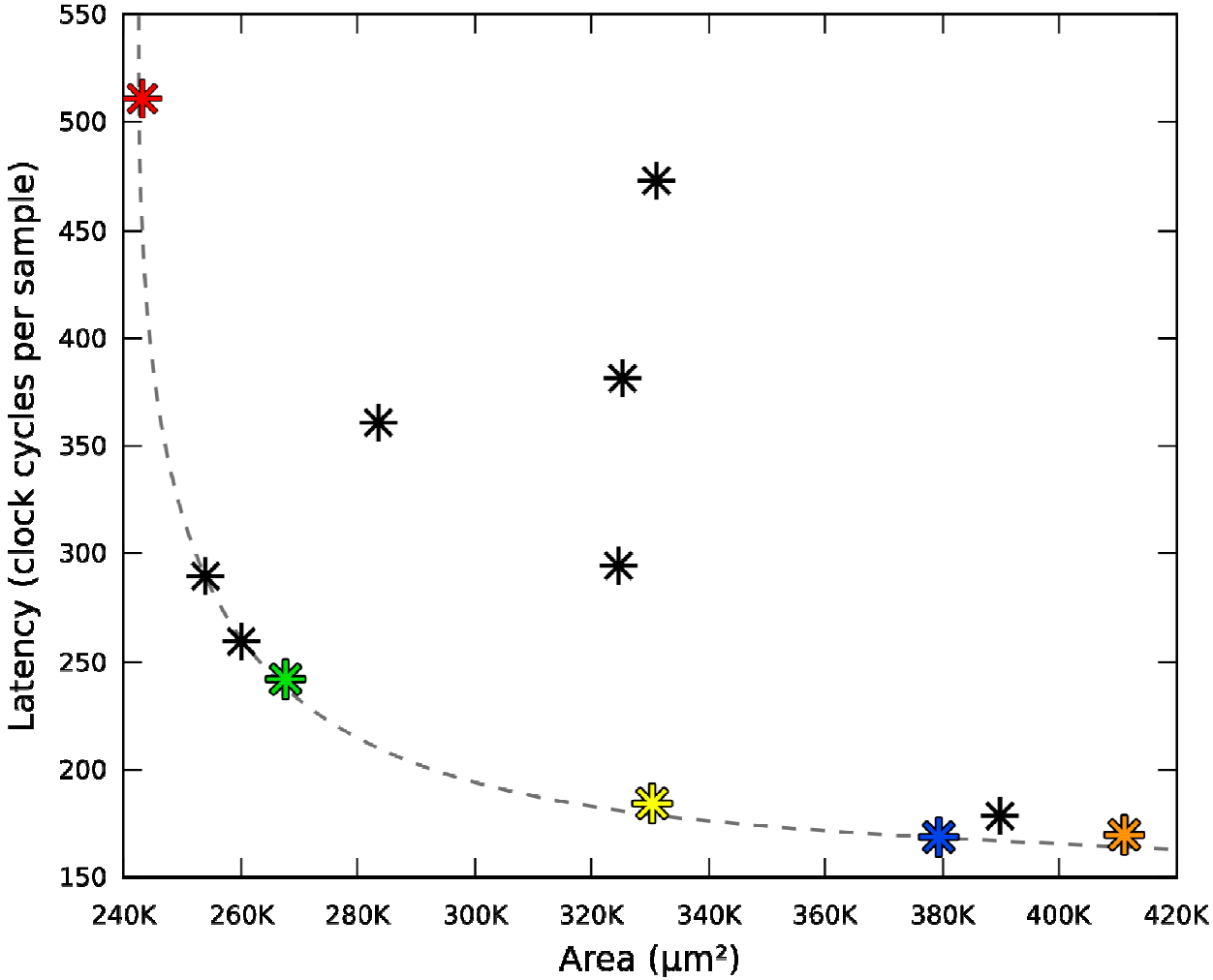
- The same application, when designed in SystemC RTL required 6 designers to produce a single design point in one year

RTL Synthesis Comparison

- Half of pipeline synthesized
  - From *ScaleFactor Reorder* through *DCT* modules
  - TSMC .13 µm technology
- RTL Design
  - 300 Cycles per sample
  - 171K µm²
- Behavioral Design
  - 196 Cycles per sample
  - 176K µm²

Behavioral Results: Area x Latency

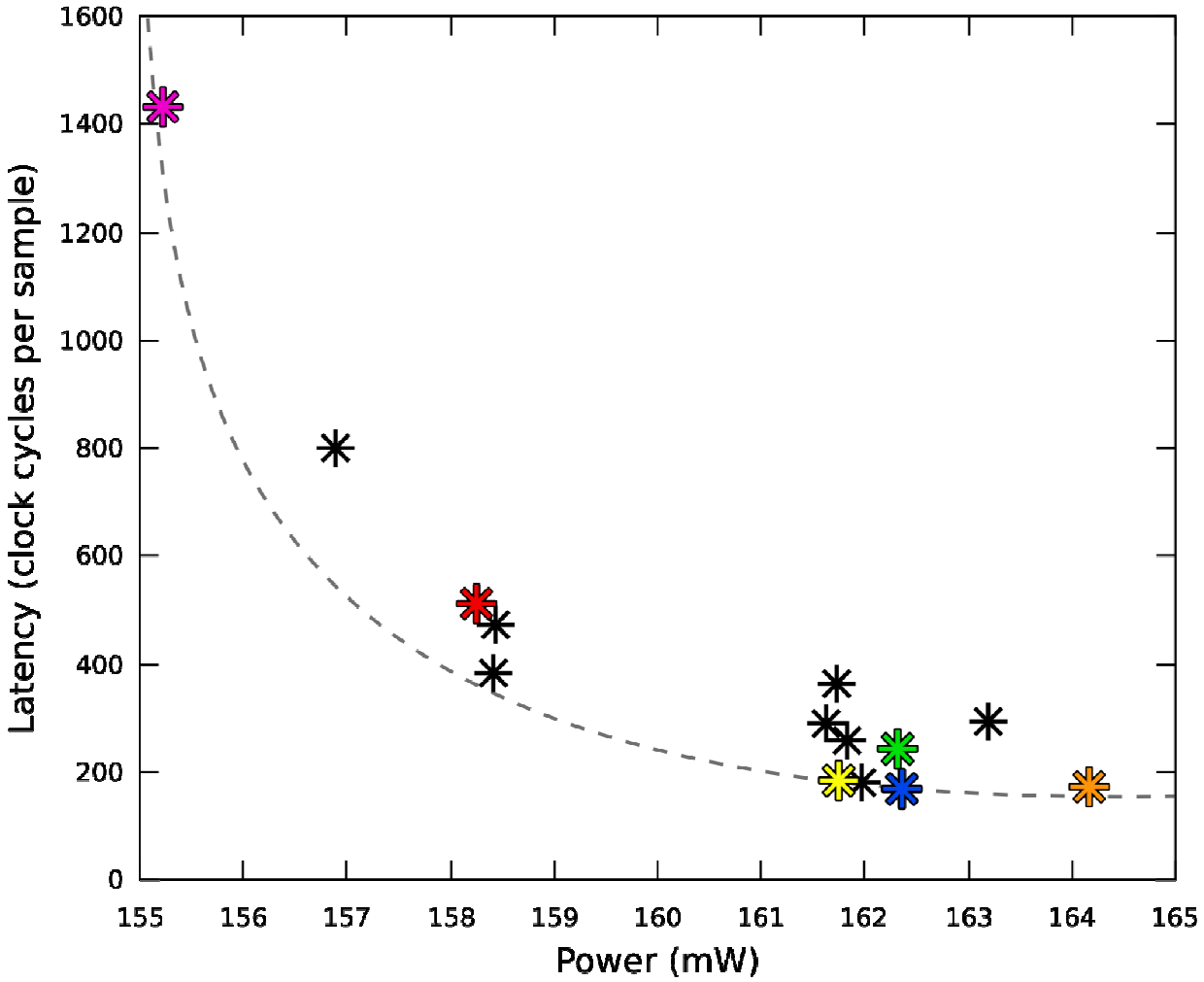
- TSMC .13 µm at 100 MHz



Latency (cycles)	Area (µm²)
169	379K
170	411K
178	390K
184	330K
242	268K
259	260K
290	254K
294	325K
361	284K
381	325K
473	331K
511	243K
801	293K
1433	292K

Behavioral Results: Power x Latency

- TSMC .13 µm at 100 MHz



Latency (cycles)	Power (mW)
169	162.4
170	164.2
178	162.0
184	161.7
242	162.3
259	161.8
290	161.6
294	163.2
361	161.7
381	158.4
473	158.4
511	158.3
801	156.9
1433	155.2

Conclusions

- Behavioral synthesis
  - Good design space exploration
    - 14 design points
  - Better results
    - 33% of reduction in Latency with the same area
  - 75% of reduction in development time
  - 1/6 of developers working on the design
  - Easier to implement and understand the code