UTDSP: A VLIW DSP Processor in TSMC 0.35 CMOS

Sean Hsien-en Peng

Supervisor: Prof. Paul Chow

Computer Engineering Group

University of Toronto

speng@eecg.toronto.edu

- **Motivation**
  - **Low-Cost, Low-Power and High-Performance DSP Processors are needed for telecommunication and embedded systems**
  - **VLIW architectures are ideal targets for HLL compilers to exploite parallelism**

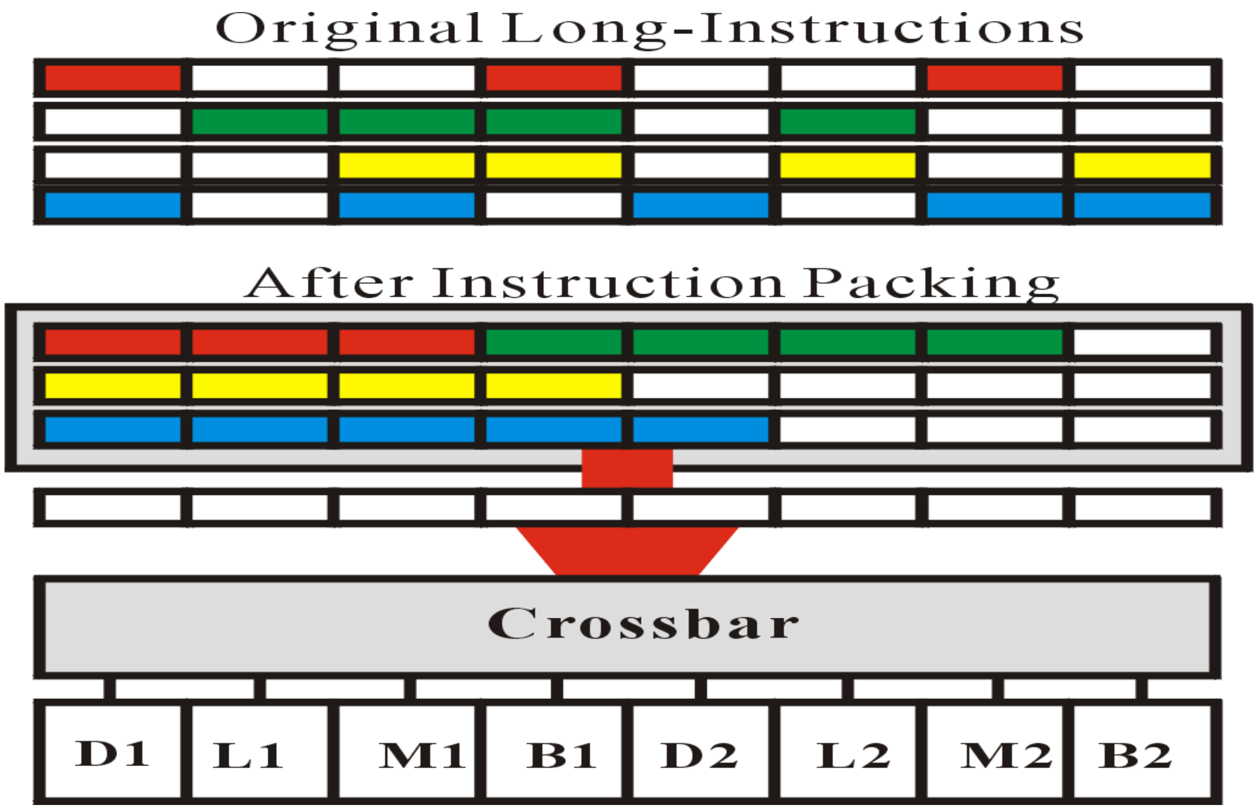
    Flexible for application-specific embedded systems

    (Current VLIWs: TI TMS320C6x, and Philips R.E.A.L. DSP)

- **Limitations of Current VLIW DSPs in Cost-Sensitive systems**
  - **Memory size is increased substantially due to the empty slots in long-instruction words**
    1. **Unable to exploite enough parallelism in applications**
    2. **Loop Unrolling Technique used in compilers**
  - **Huge memory bandwidth for long-instruction fetching**

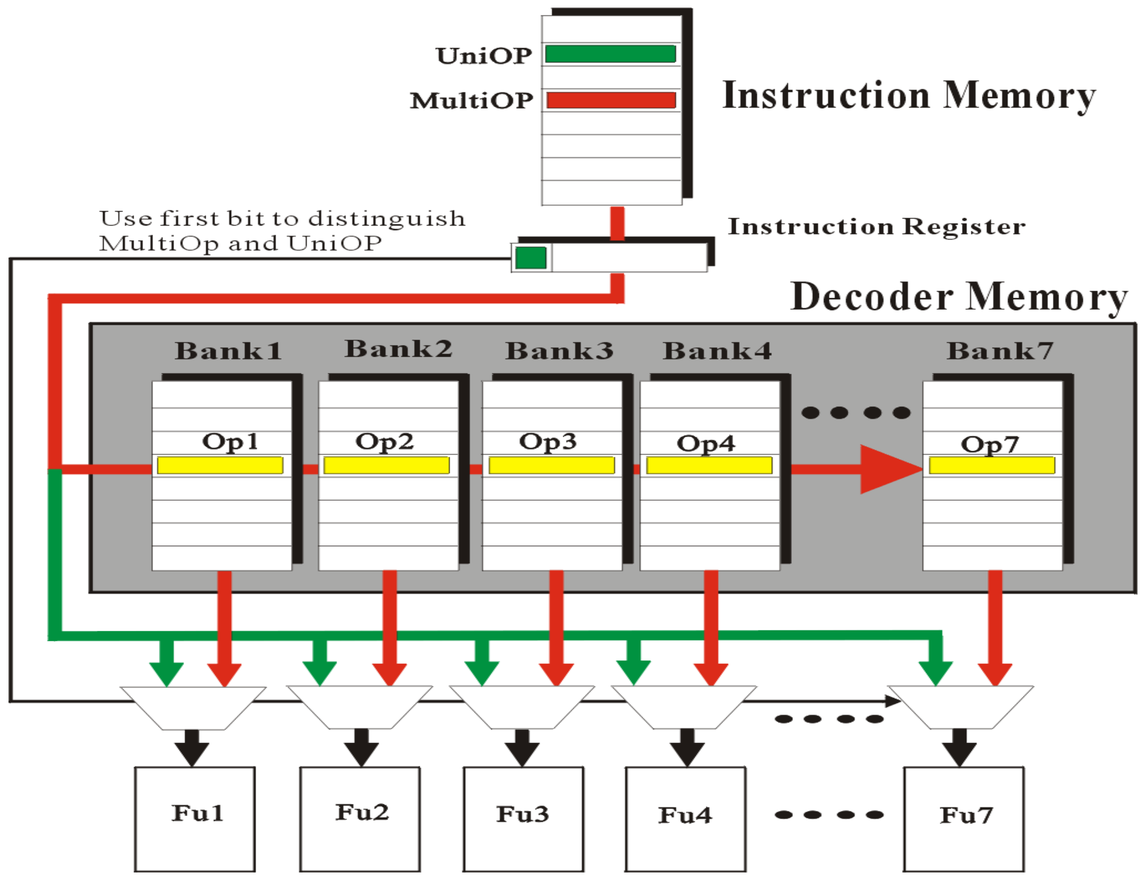
    Will be the performance bottleneck when off - chip instruction memory is used

- **The New TI VelociTI Architecture in TMS320C6x**

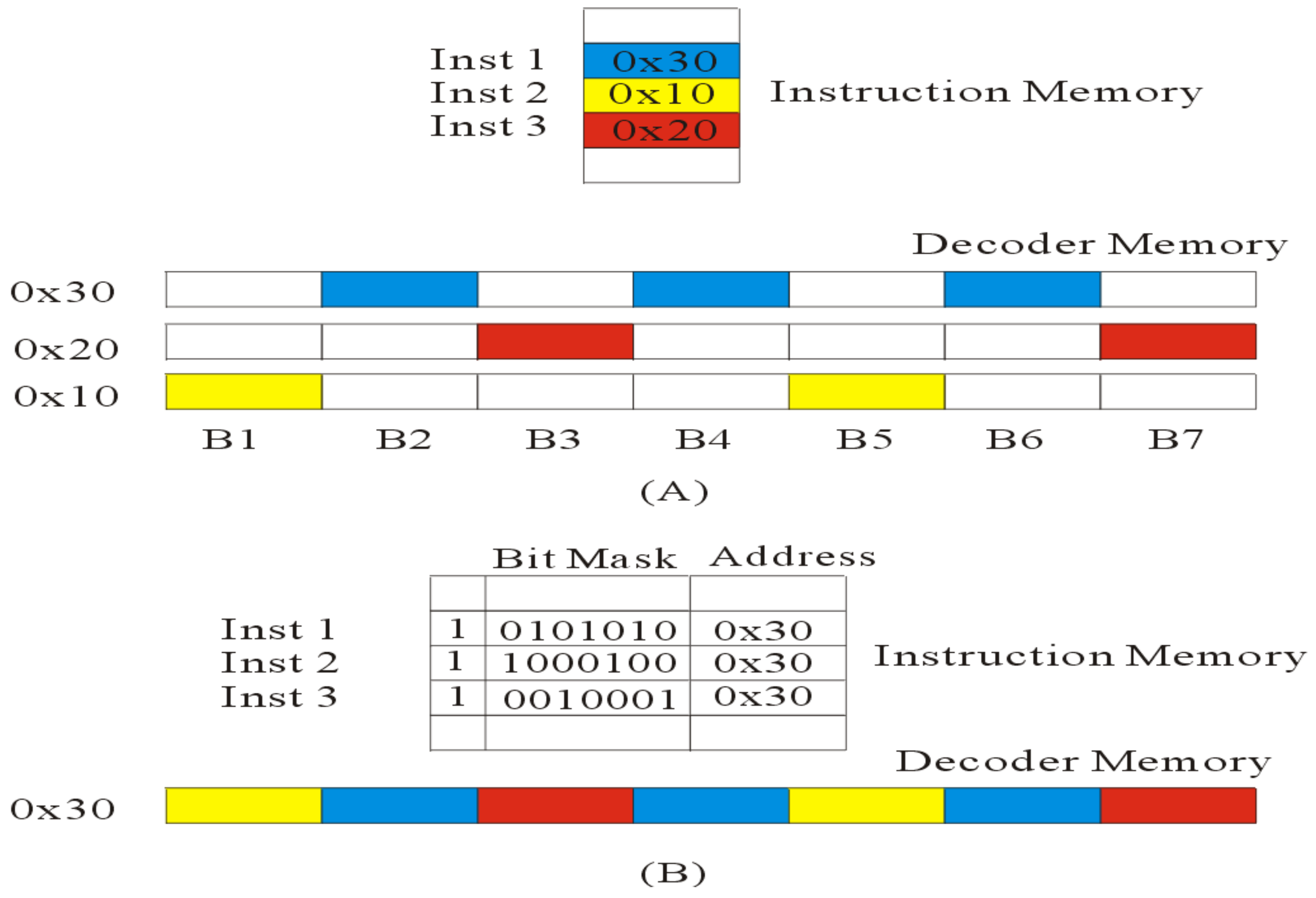


- ## A Novel Instruction Packing and Decoding Method
  - Based on a two-level horizontal microcode architecure
  - Achieve better packing results while eliminating the need of using crossbar
  - Reduce off-chip instruction memory bandwidth for cost-sensitive systems
  - Patent has been filed for the memory packing design

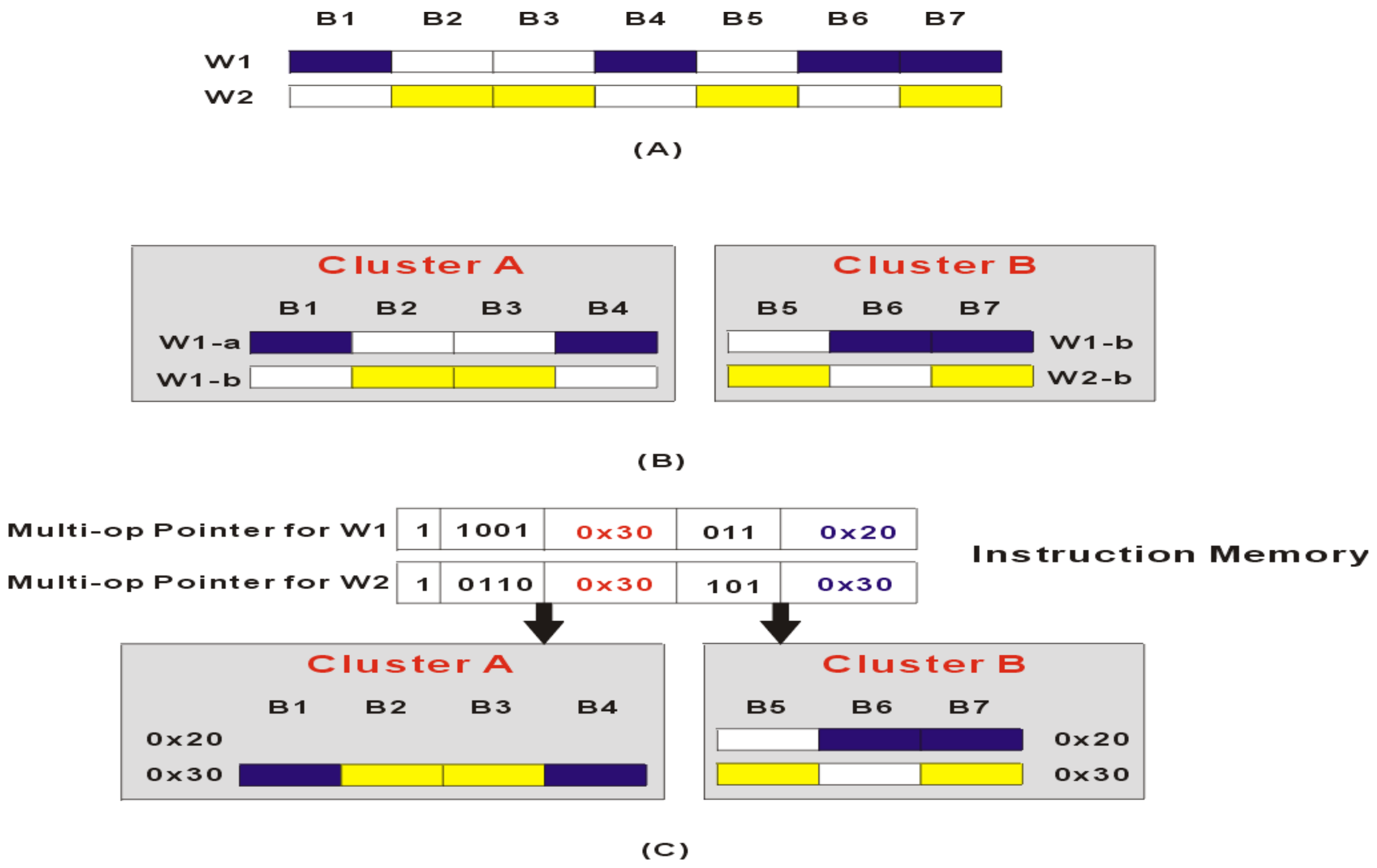
• The UTDSP Memory System



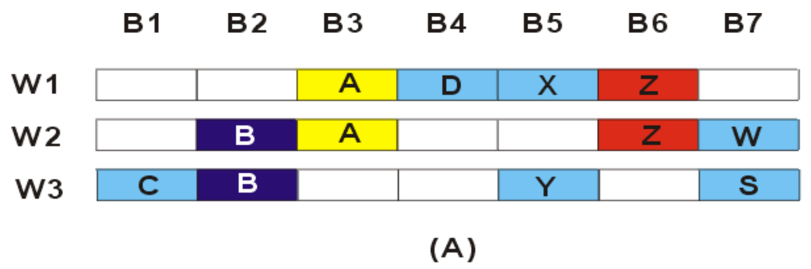
- **Reduce size of Decoder Memory**



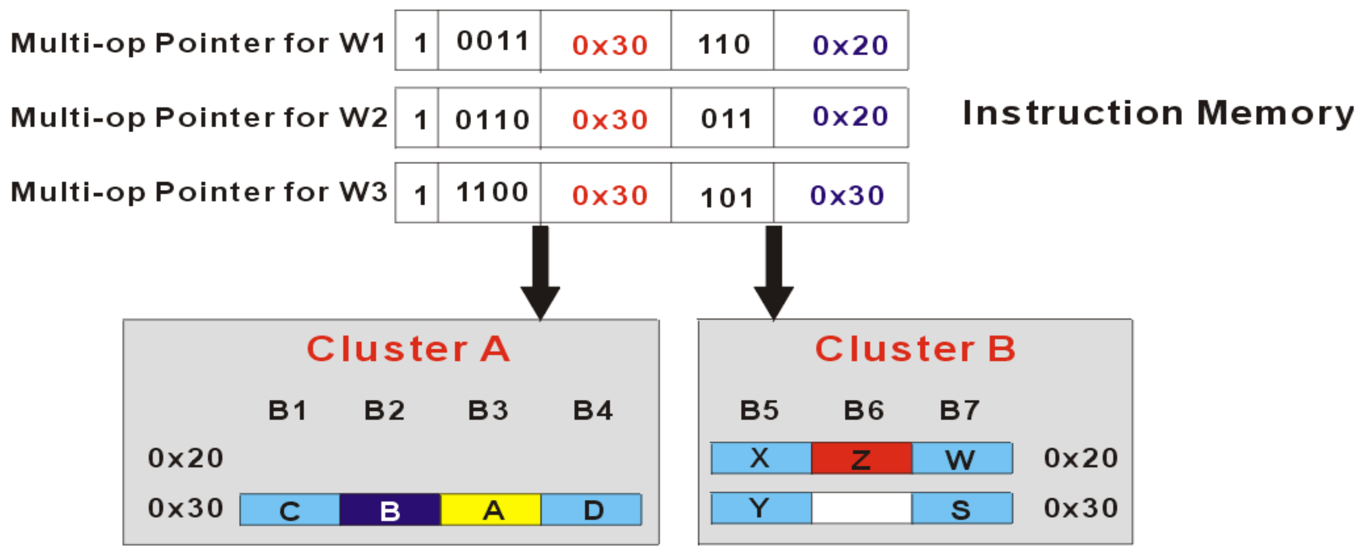
- **Denser packing using two clusters**



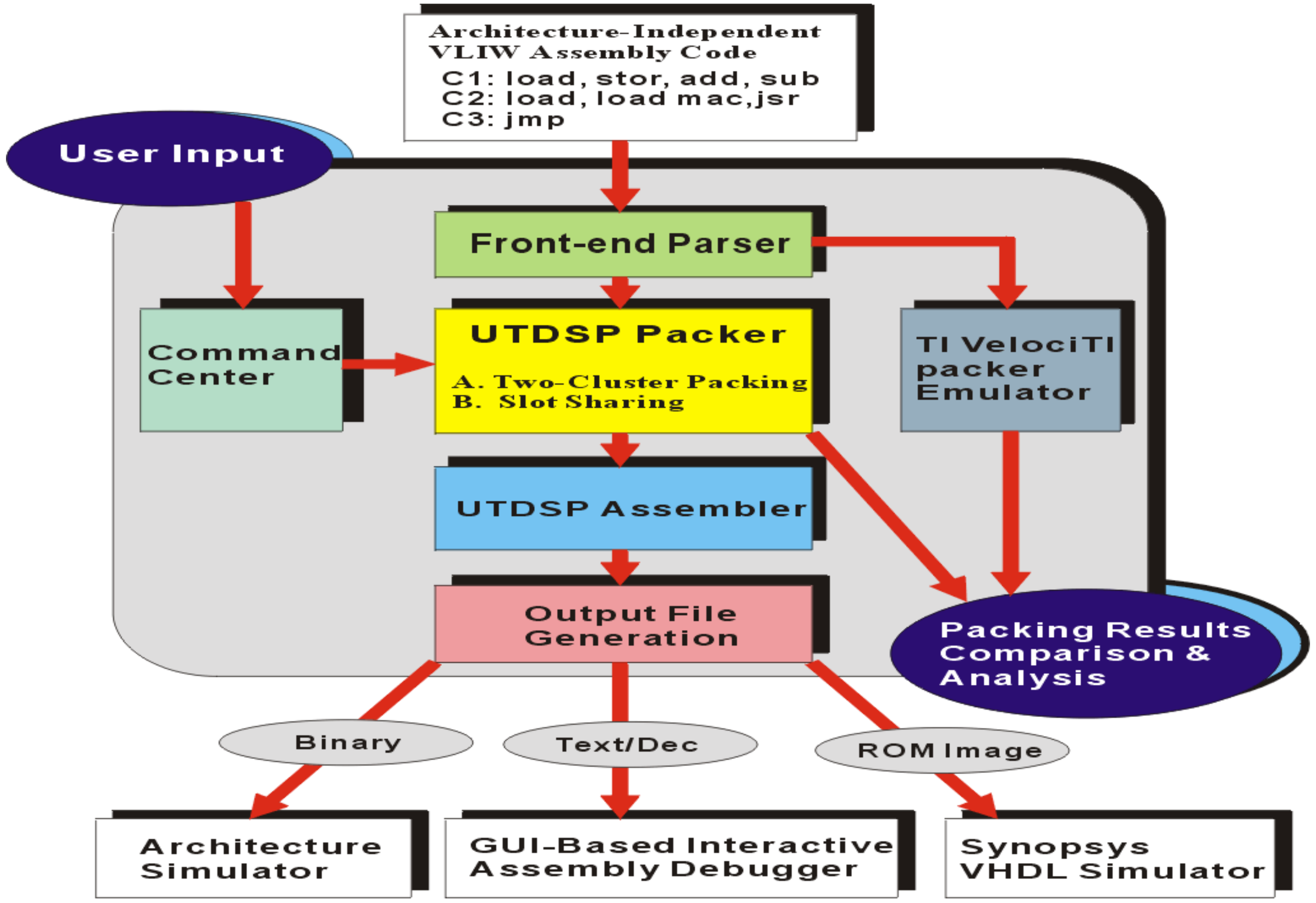
- **Combine clusters and slot sharing to have a near-optimal result**



(A)



• The Packer and Software tools



- **Need complex data structures**
  - **Design C++ template libraries to provide the container class for different objects**

```
List <Inst> A;
List <Decoder> B;
List <Token> C;
```

  - **Design associated class methods and use operator overloading to ease the packing algorithm development**

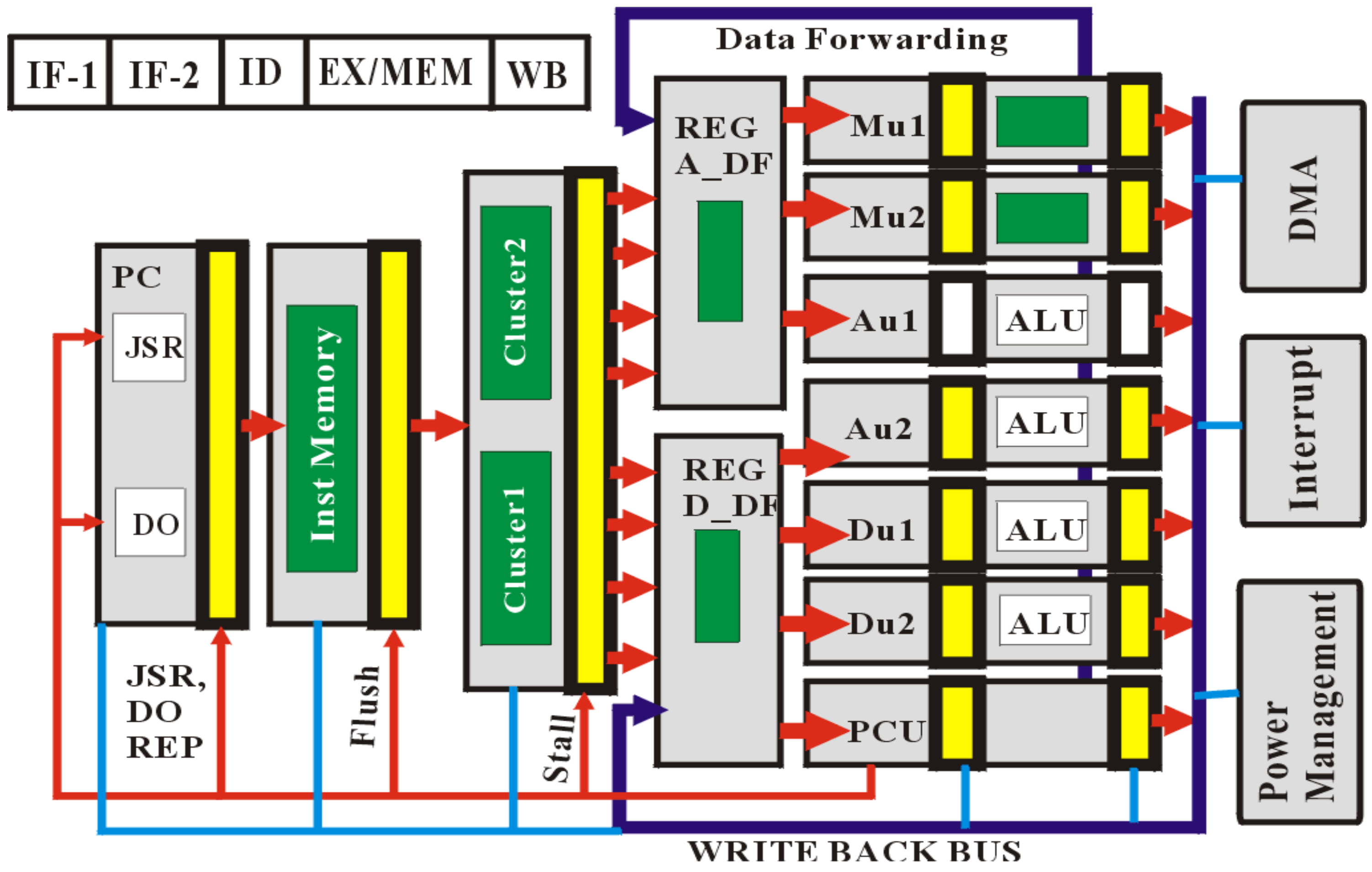
```
ListA = ListB.merge(ListC);
if ( InstA > InstB ) ....
```

- **Better VLIW solution than TI's new VelociTI architecture**
  - **Similar instruction compaction rate (65%)**
  - **Eliminate the necessity of using crossbar**
  - **Can use inexpensive off-chip instruction memory without suffering the bandwidth problem that TI has.**
  - **Minimize the size of on-chip memory**

    90% of execution time is spent in 10% of code

    => only need to store DSP kernel code on chip

• The UTDSP Architecture



Computer Engineering Group

- ## Instruction Set
  - **Highly orthogonal, RISC-like instructions**

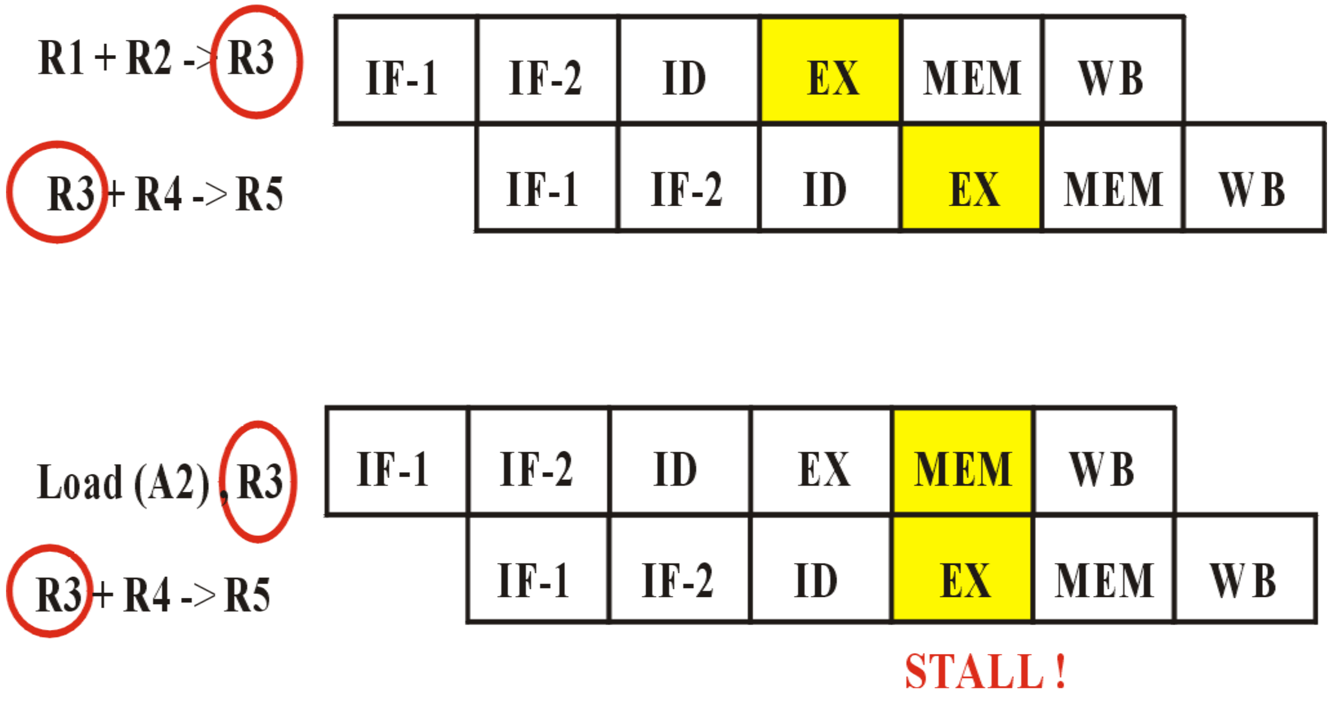
    add r1,r2,r3, mult r1,r2,r3, ld (a1), r2 st r4,(a5)
    JSR, Jmp, BEQ, BLE.
  - **Specialized DSP instructions**

    multiply-accumulator, modulo addressing,
  - **Zero-Overhead Looping Instruction**

    Achieve optimal performance in DSP kernels
    Can handle 8-level nested looping,
    interruptable, good for real-time application

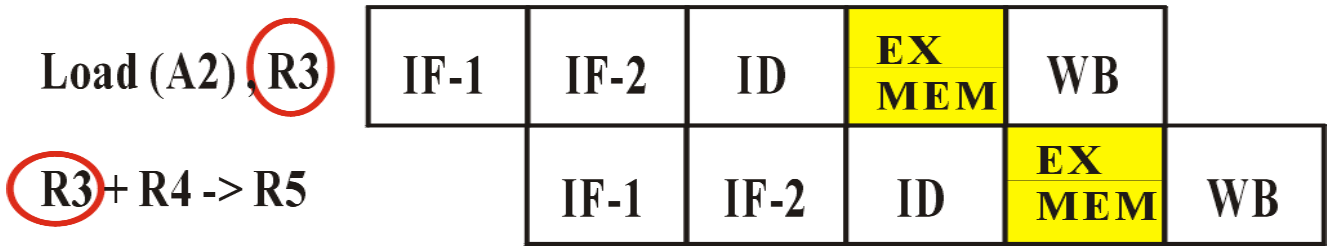
Data Hazards and Data Forward

- Traditional RISC architectures stall pipelines to solve RAW data hazards



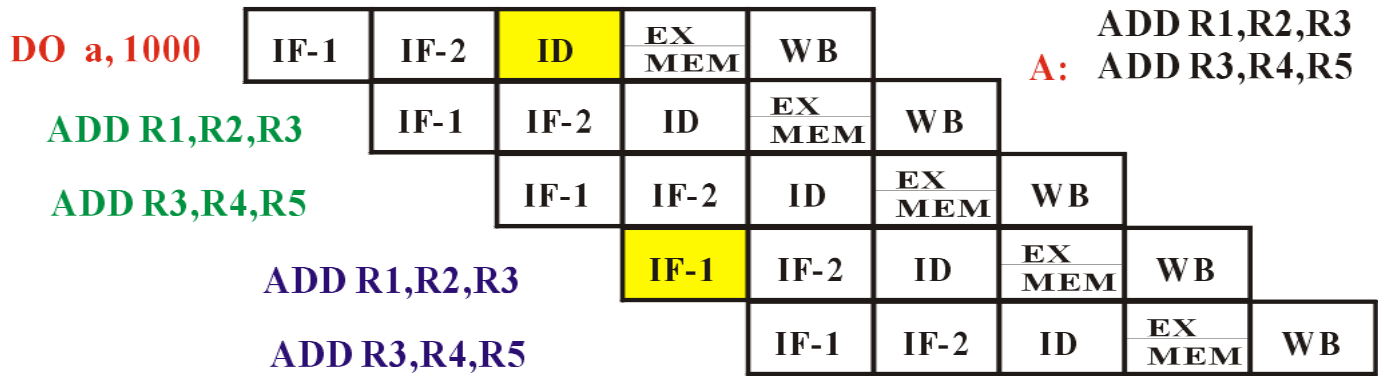
Combine EX/MEM to reduce RAWs and Bypassing Logic

- Use register indirect addressing mode instead of displacement mode (ld (A2+4), R5)
- Resolve ALL RAWs now, no need to stall
- Reduce bypassing logic by 50%
- Pipeline latency is NOT changed (parallel)



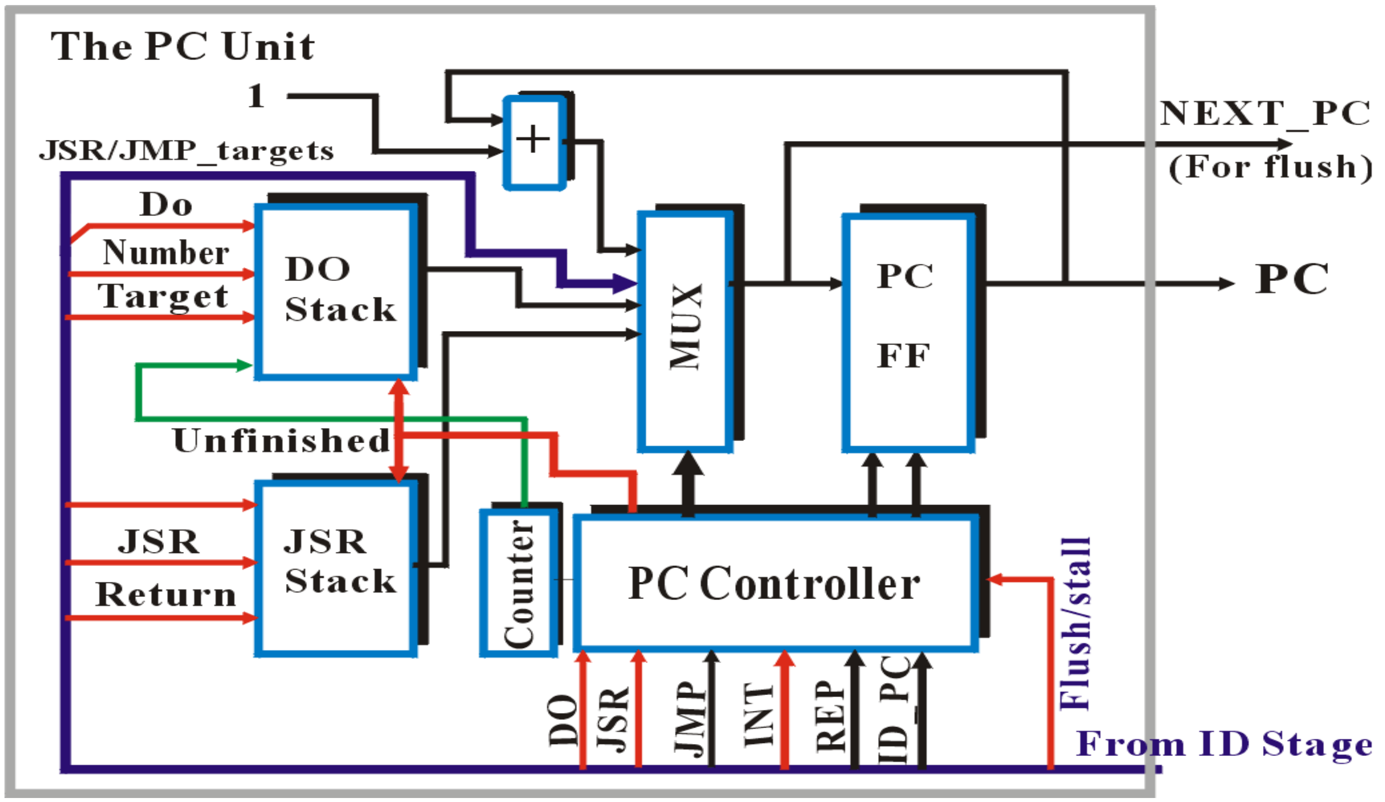
- **Zero-overhead Looping Inst. reduces branch penalty and size of instruction memory**
  - **TI suggests assembly programmers using Loop Unrolling to optimize DSP kernel code**

    1. Difficult 2. Increase code size dramatically
  - **UTDSP has a zero-overhead DO instruction**

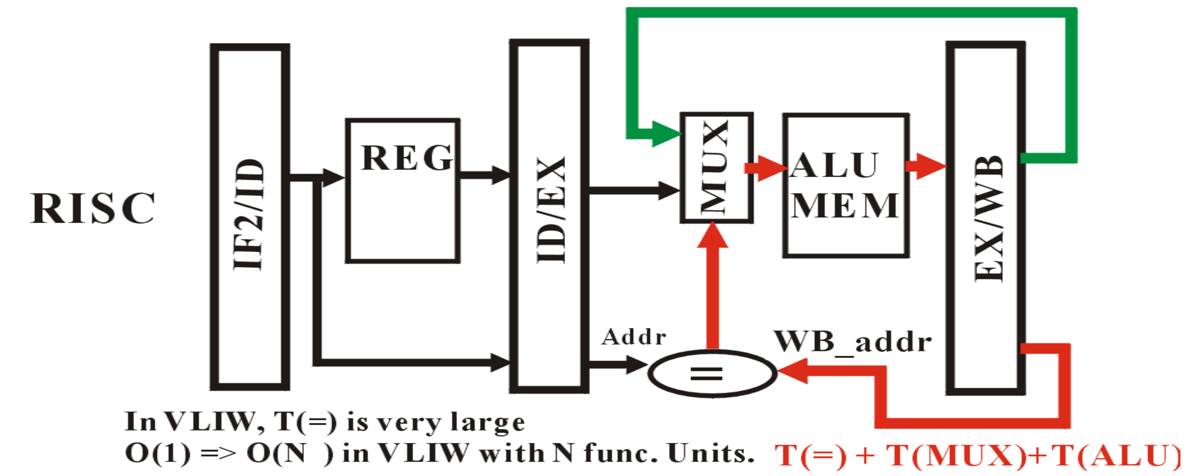


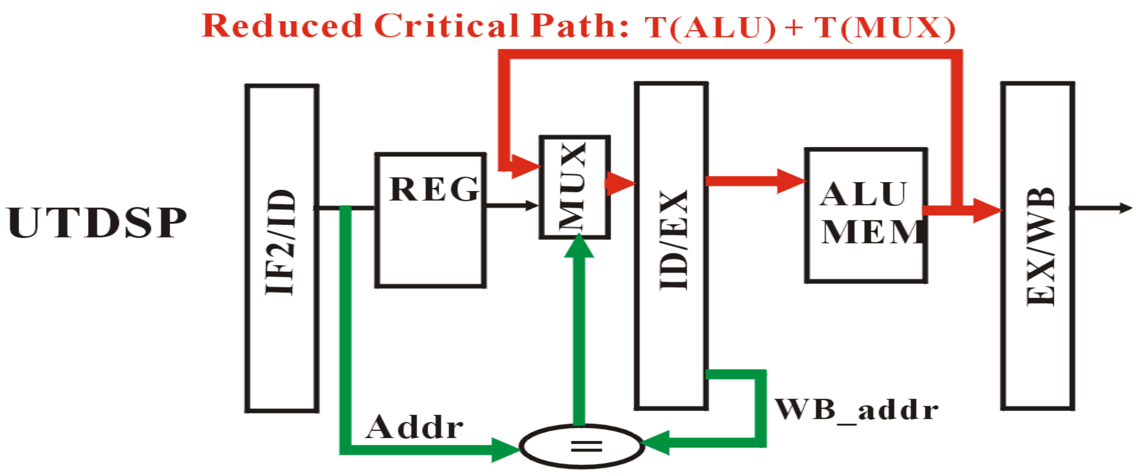
```
    DO  a, 1000
    ADD R1,R2,R3
A:  ADD R3,R4,R5
```

- **Design Challenge: The PC Unit: Handles 8-level nested DO loop. Allows JSR, Branch, and Interrupts in inner loop**



• The Forwarding Logic:


RISC
IF2/ID
REG
ID/EX
MUX
ALU
MEM
EX/WB
Addr
=
WB_addr
In VLIW, T(=) is very large
O(1) => O(N) in VLIW with N func. Units.
T(=) + T(MUX)+T(ALU)



- **VLSI Implementation Strategy**
  - **"P&R a subblock and instantiate it at top-level" methodology doesn't work in processor design**
  - **Various size of blocks => Need grouping/ merging features in floorplan tools**
  - **Memory blocks => Need hierarchical P&R**
  - **Huge interconnection bandwidth between blocks (bypassing) => Need Global Pin Optimization (GPO)**
  - **0.35u: Wiring delay dominates gate delay => Need GPO and Area-based router**

UTDSP CAD Flow

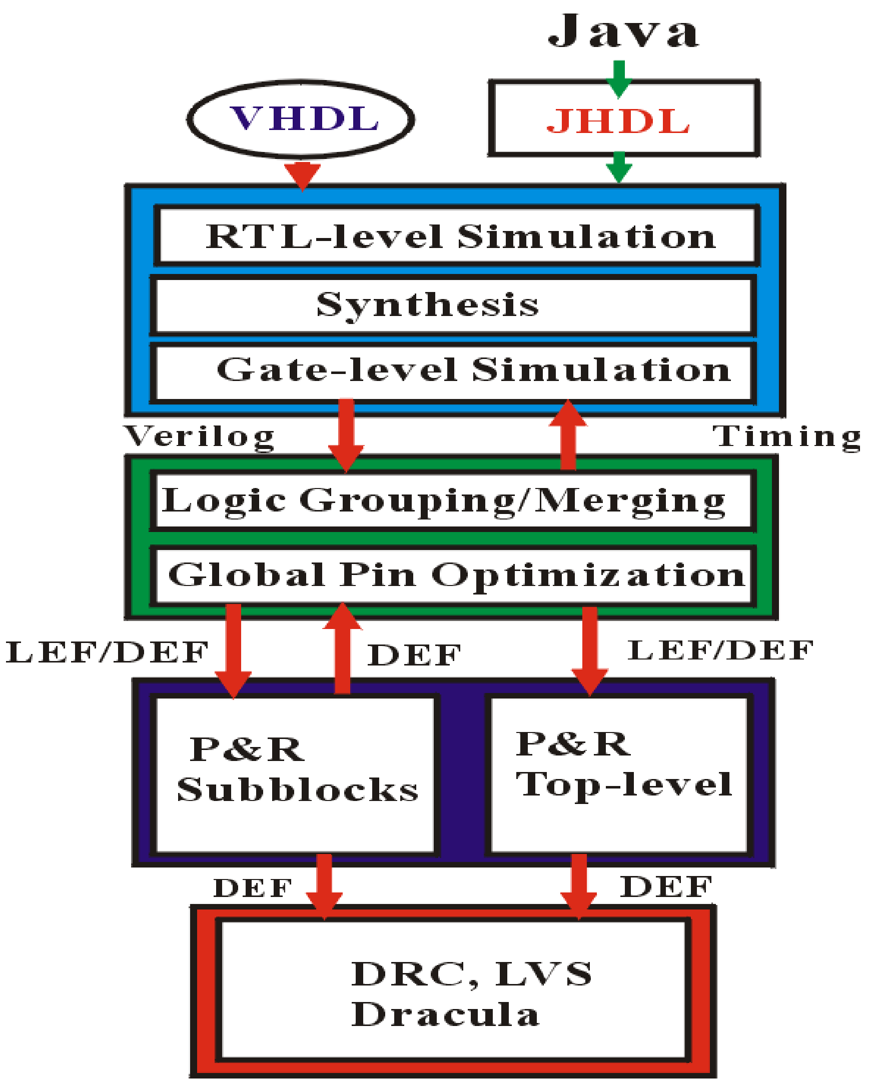
CMOSP35 V4.0 Kit
- TSMC 0.35u cells

Synopsys

Cadence PDP Physical Design Planner 3.4C

Silicon Ensemble V 5.2

Cadence 1999a



• Benchmark Results

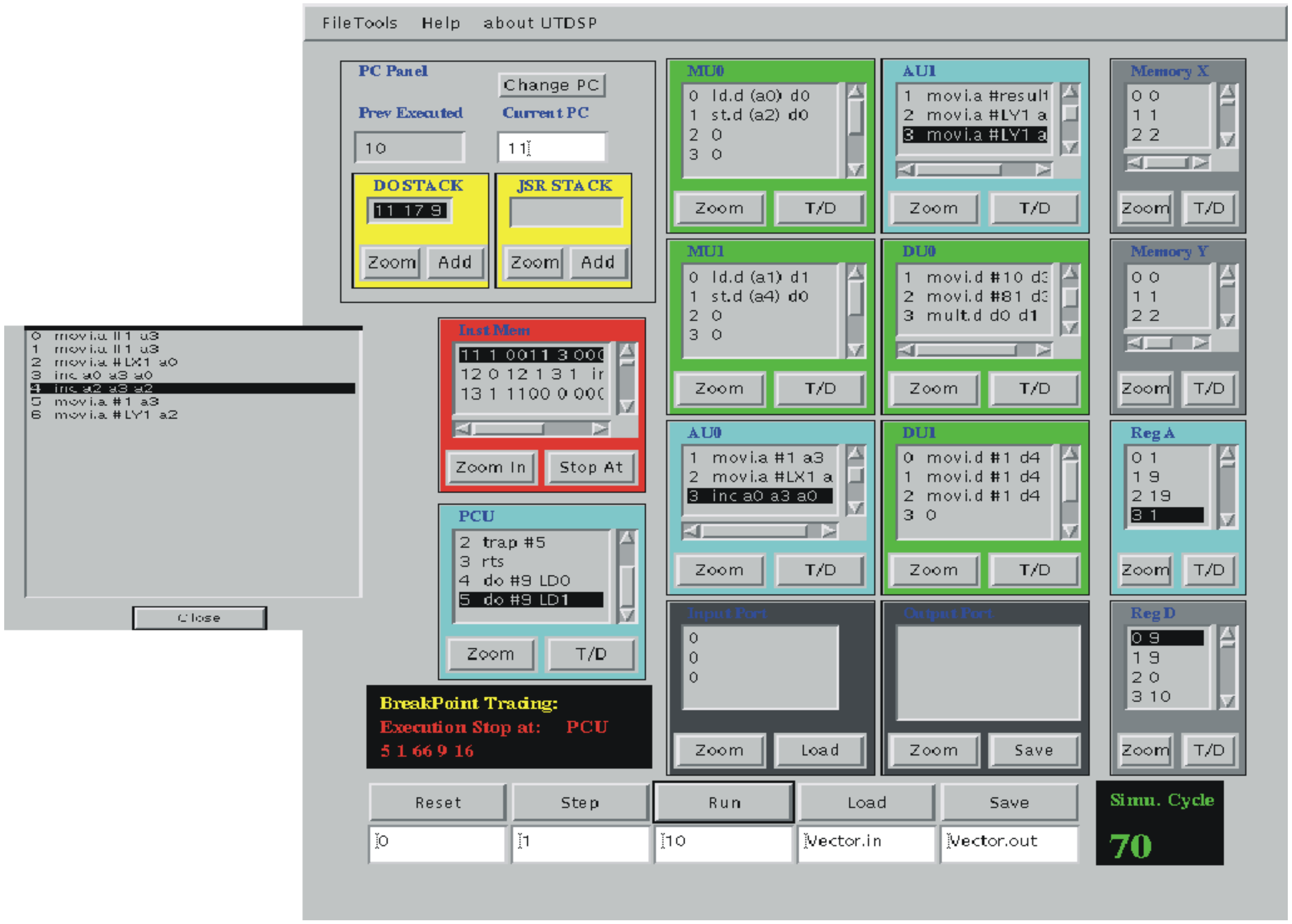
UTDSP Quick Facts: Die size: 7.2 x 7.2 mm
170,000 gates
20KByte On-chip SRAM

	Texas Instruments TMS320C62	Philips R.E.A.L DSP (Core)	The UTDSP
Clock	200 MHz	70 MHz	70 MHz
Pin count	> 400 pins		105 pins
Func. Units	8	10	7
FIR 4, N_coeff, M_output smaples	Mx(N+8)/2 + 6 cycles	~Mx(N+7)/2 + 8 clcles	M x (N+6) /2 + 6 cycles

- S.O.C Solution for Low-Cost Low-Power Telecommunication and Consumer applications
  - The UTDSP core implemented in synthesizable VHDL can be easily integrated with other system blocks as an IP core
  - Provide a GUI-based architecture simulator to help desingers to evaluate/modify the UTDSP core
  - Provide an interactive assembly debugger comparable to any commercial counterparts
    Single-step trace, set break-point, memory location probing.

- ## Conclusion: What has been done
  - RTL level VHDL for UTDSP (10,000 lines)
  - UTDSP Long-instruction packer and assembler (5,000 line C++ with template)
  - Hierarchical P&R flow using PDP + Silicon Ensemble + Cadence 1999a
  - GUI-based assembly debugger and simulator (6,000 lines in Java)
  - Potential commercialization will benefit the Canadian industry

The Assembly Debugger



PC Panel
Prev Executed: 10
Current PC: 11

DOSTACK: 11 17 9
JSR STACK:

Inst Mem
```
11 1 0011 3 000
12 0 12 1 3 1  ir
13 1 1100 0 000
```

PCU
```
2  trap #5
3  rts
4  do #9 LD0
5  do #9 LD1
```

MU0
```
0  ld.d (a0) d0
1  st.d (a2) d0
2  0
3  0
```

MU1
```
0  ld.d (a1) d1
1  st.d (a4) d0
2  0
3  0
```

AU0
```
1  movi.a #1 a3
2  movi.a #LX1 a
3  inc a0 a3 a0
```

AU1
```
1  movi.a #result
2  movi.a #LY1 a
3  movi.a #LY1 a
```

DU0
```
1  movi.d #10 d3
2  movi.d #81 d3
3  mult.d d0 d1
```

DU1
```
0  movi.d #1 d4
1  movi.d #1 d4
2  movi.d #1 d4
3  0
```

Memory X
```
0 0
1 1
2 2
```

Memory Y
```
0 0
1 1
2 2
```

Reg A
```
0 1
1 9
2 19
3 1
```

Reg D
```
0 9
1 9
2 0
3 10
```

Input Port
```
0
0
0
```

Output Port

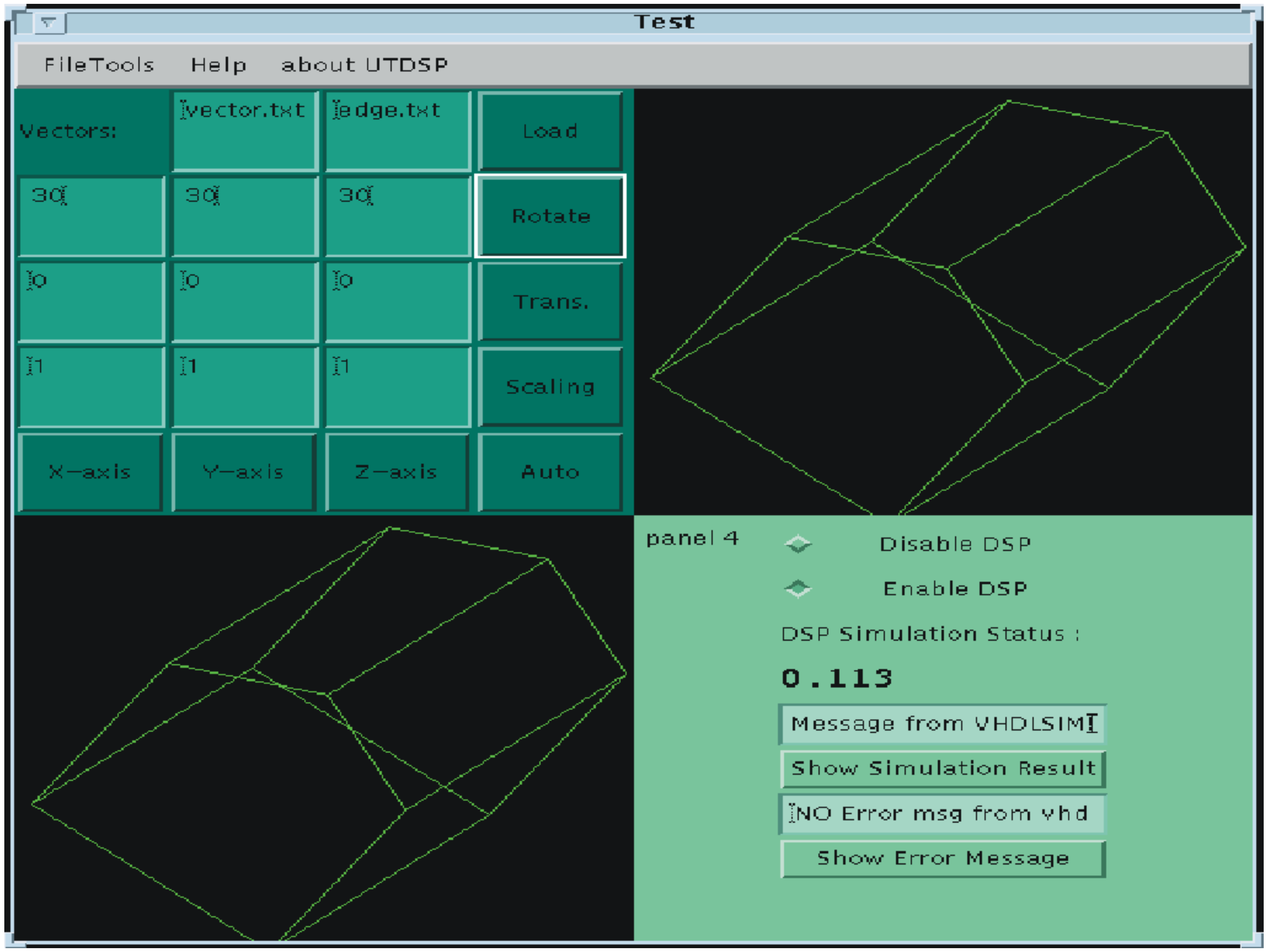
BreakPoint Trading:
Execution Stop at: PCU
5 1 66 9 16

Reset: 0 | Step: 1 | Run: 10 | Load: Vector.in | Save: Vector.out

Simu. Cycle
70

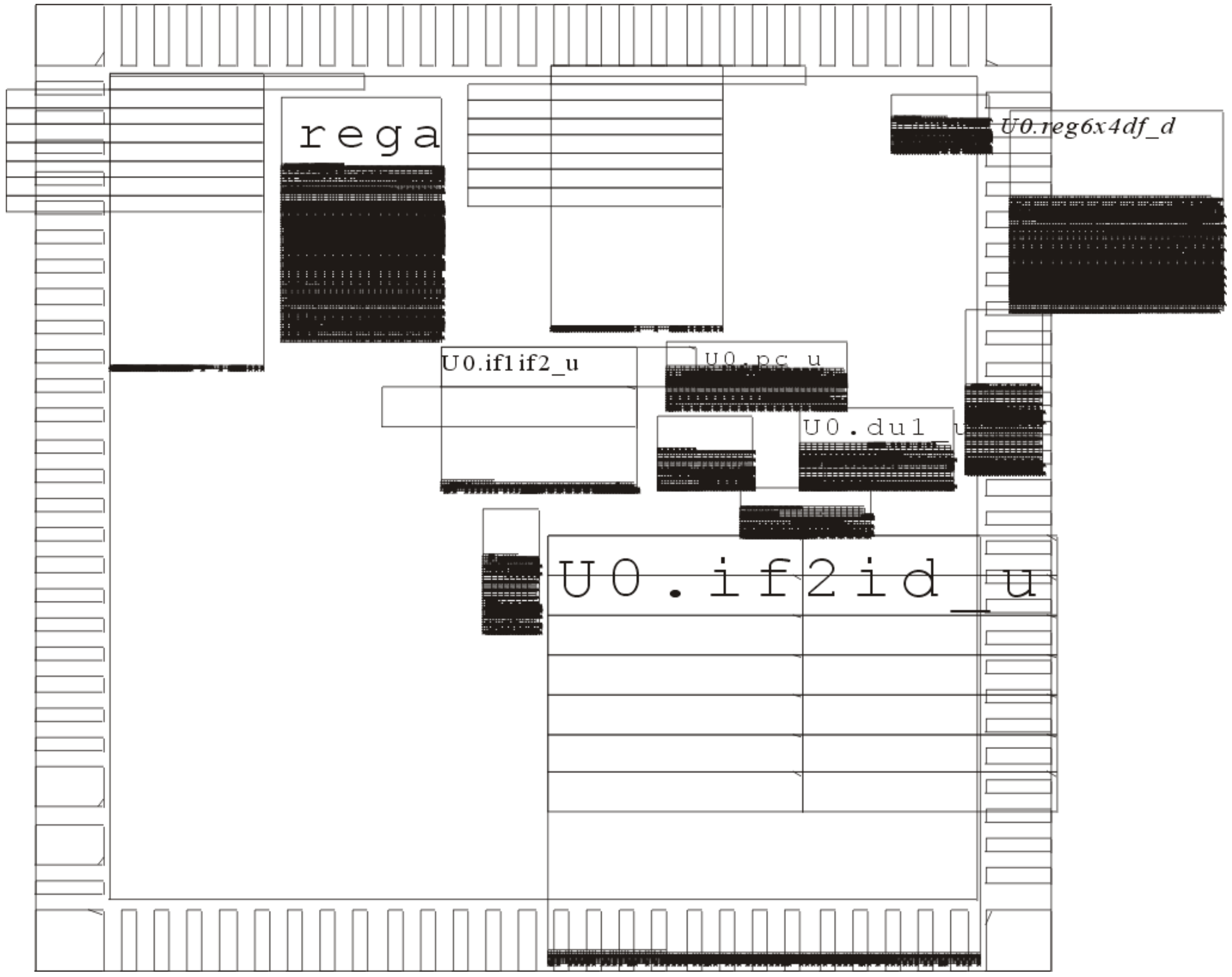
```
0  movi.a #1 a3
1  movi.a #1 a3
2  movi.a #LX1 a0
3  inc a0 a3 a0
4  inc a2 a3 a2
5  movi.a #1 a3
6  movi.a #LY1 a2
```

Computer Engineering Group

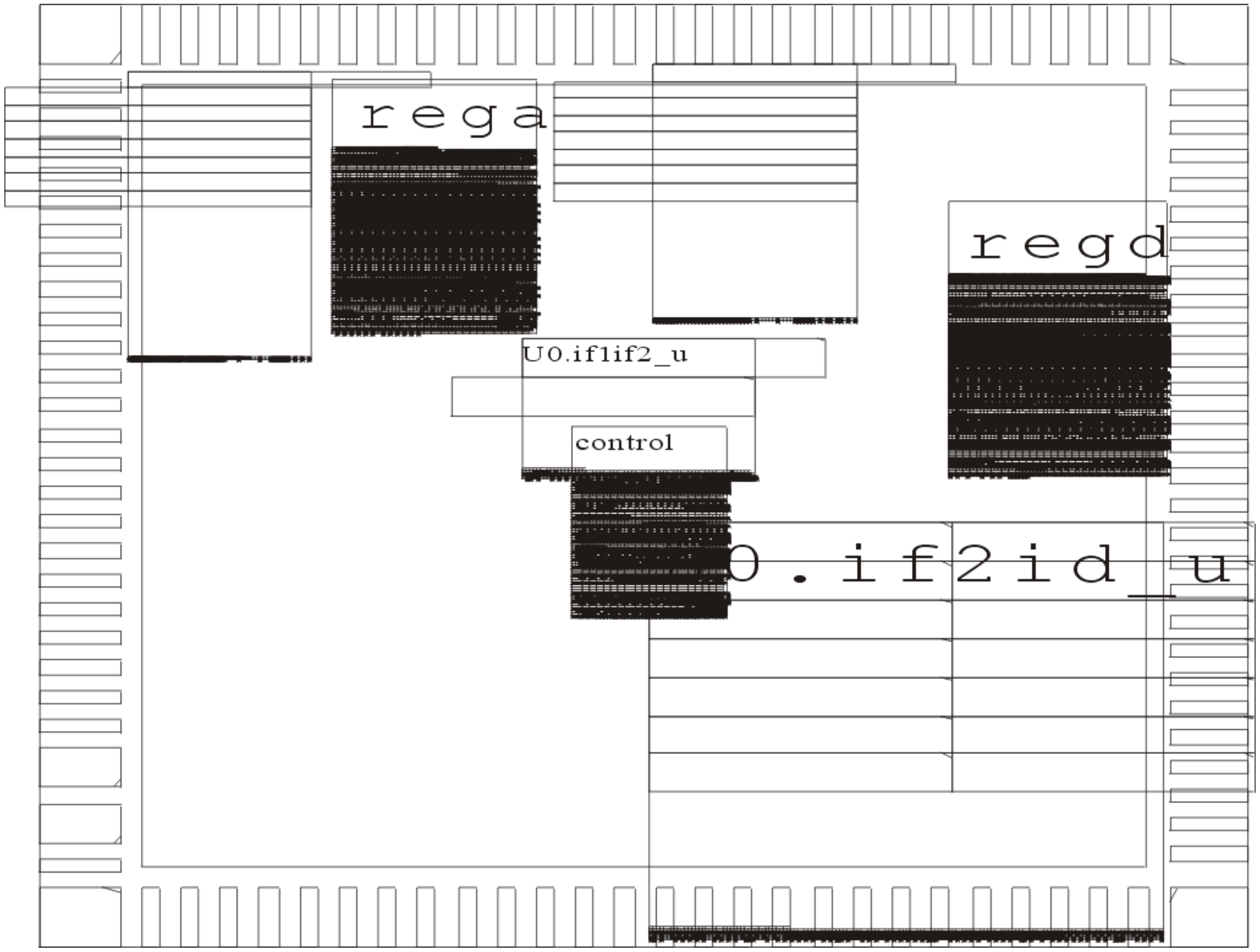


Computer Engineering Group

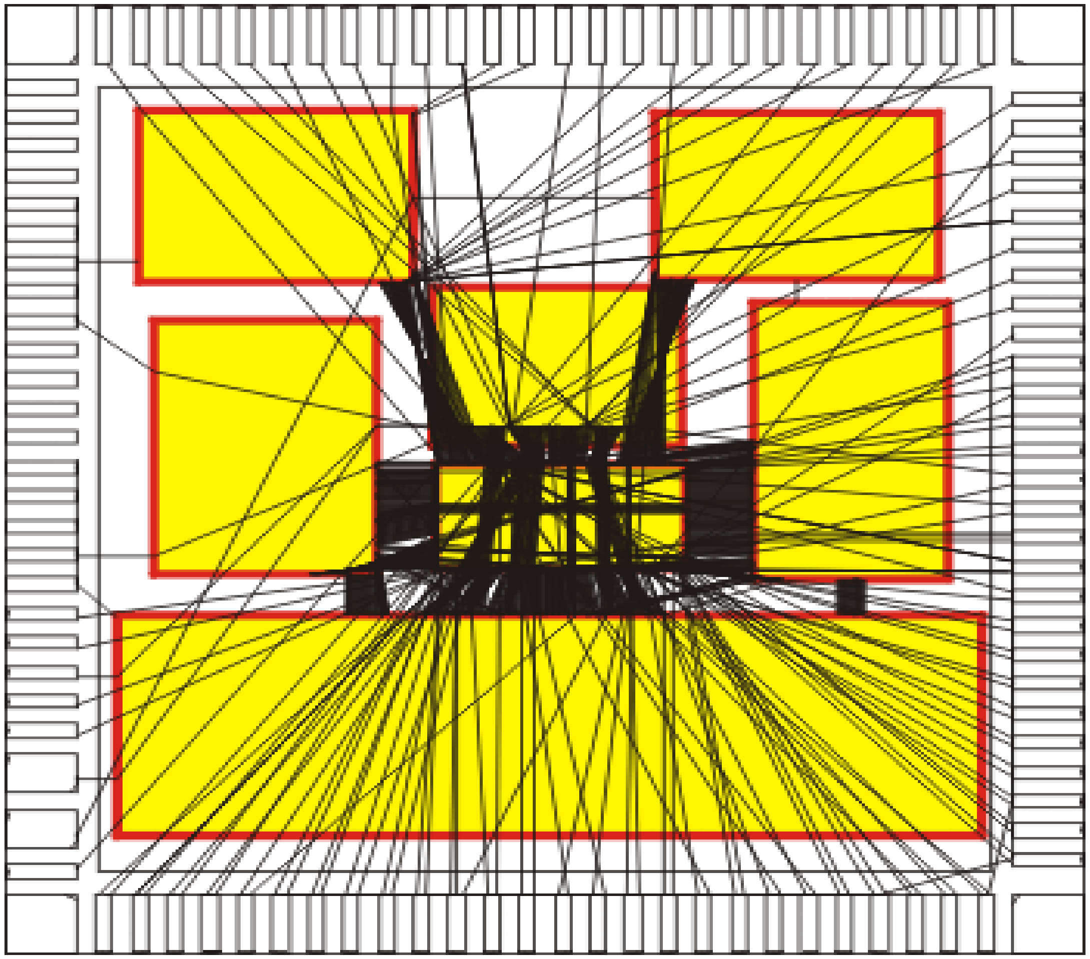
Before Logic Merging: (Cadence PDP3.4C)



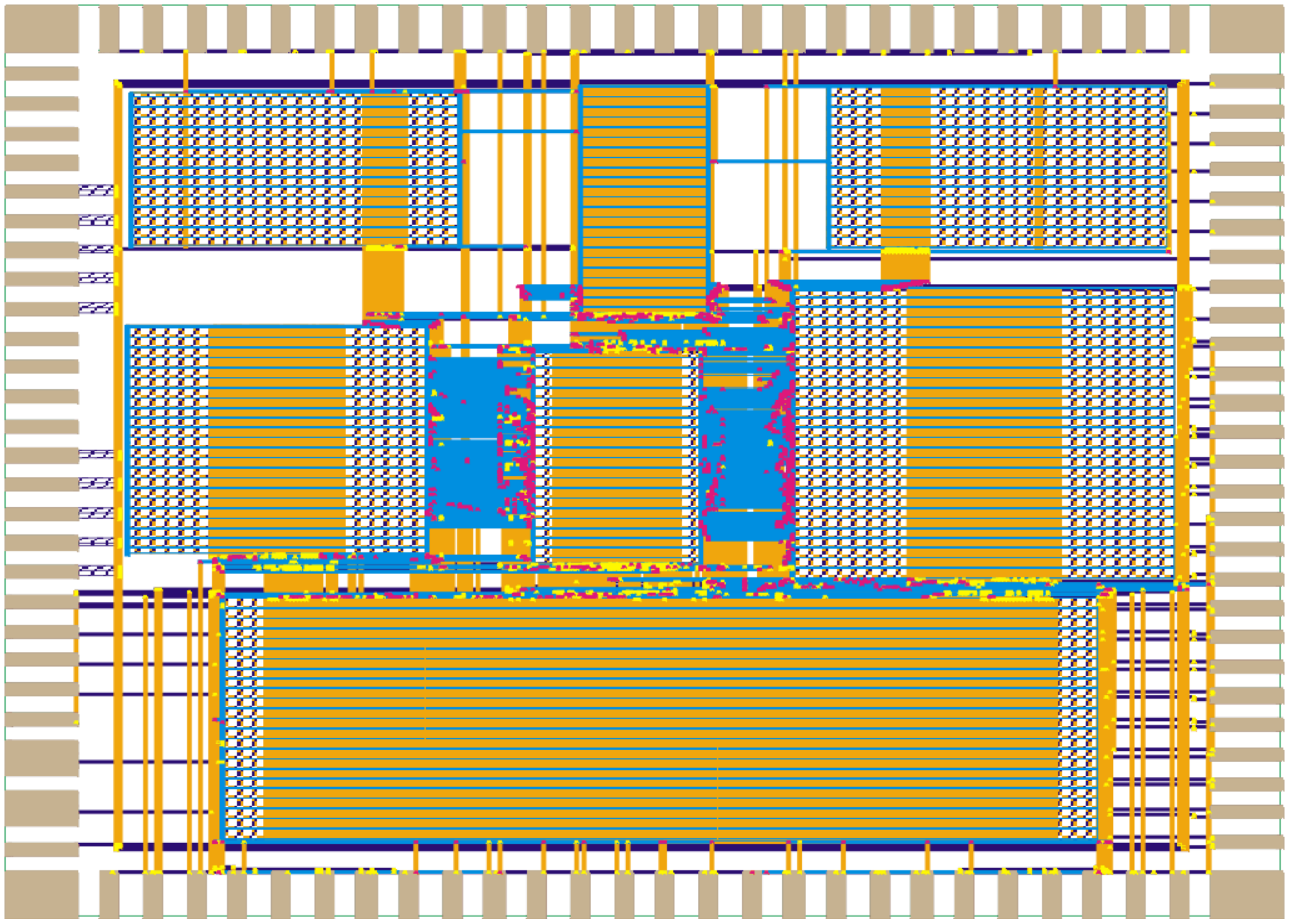
After Logic Merging: (Cadence PDP3.4C)



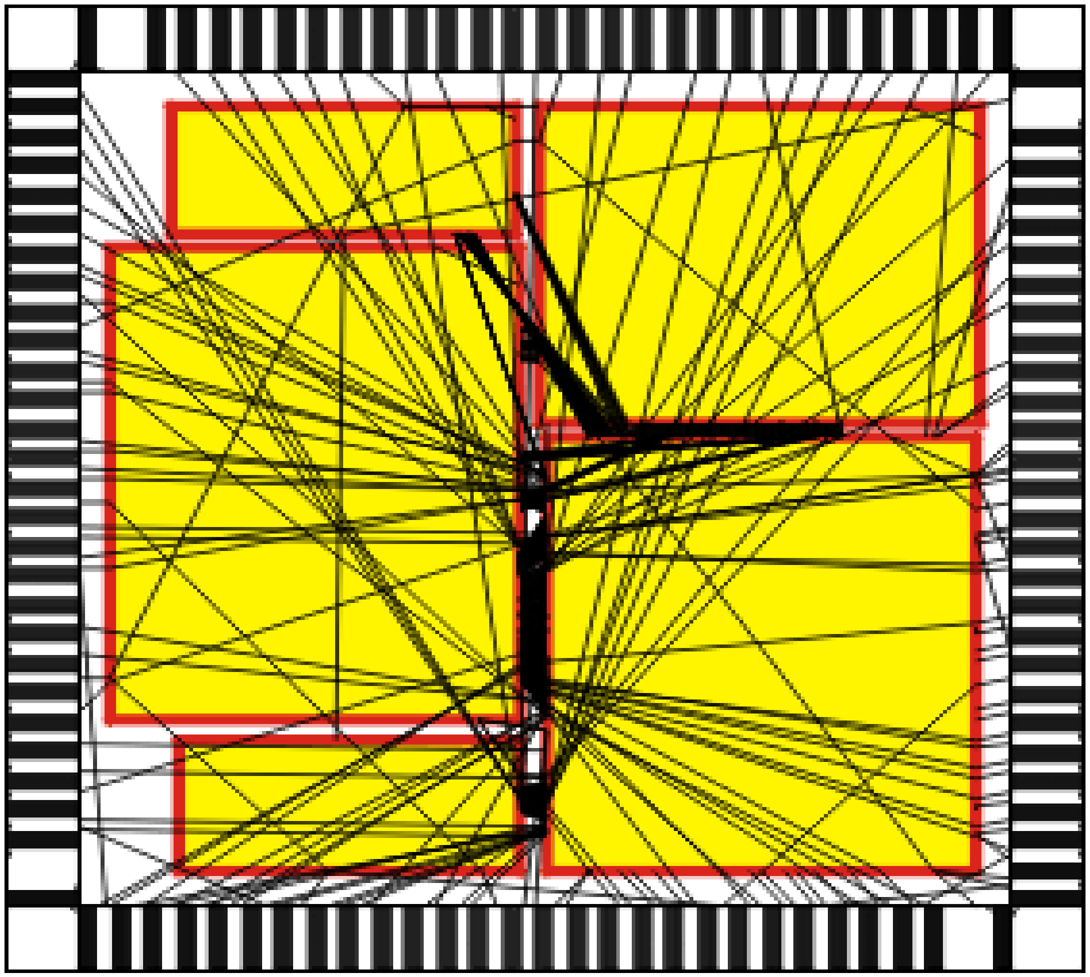
Global Pin Optimization (Cadence PDP 3.4C)



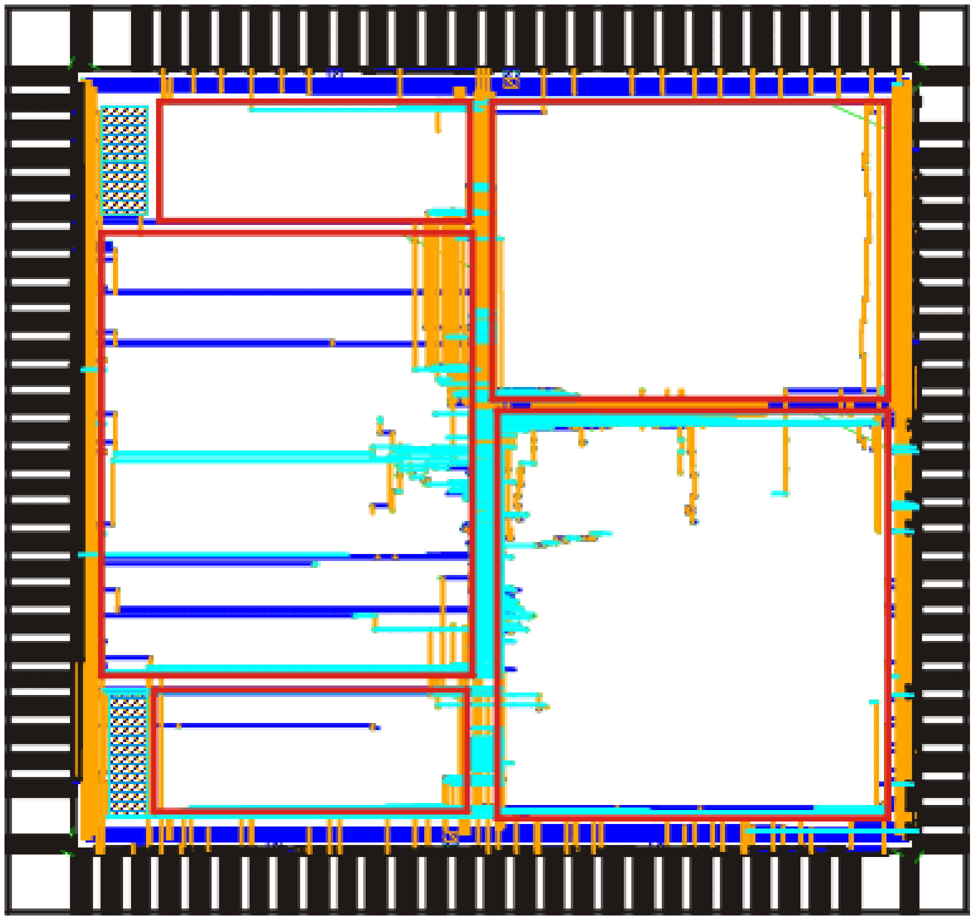
Hierarchical P&R using PDP3.4C, Silicon Ensemble 5.2



Finalized Grouping and Floorplan



Top-level Routing (Silicon Ensemble 5.2)



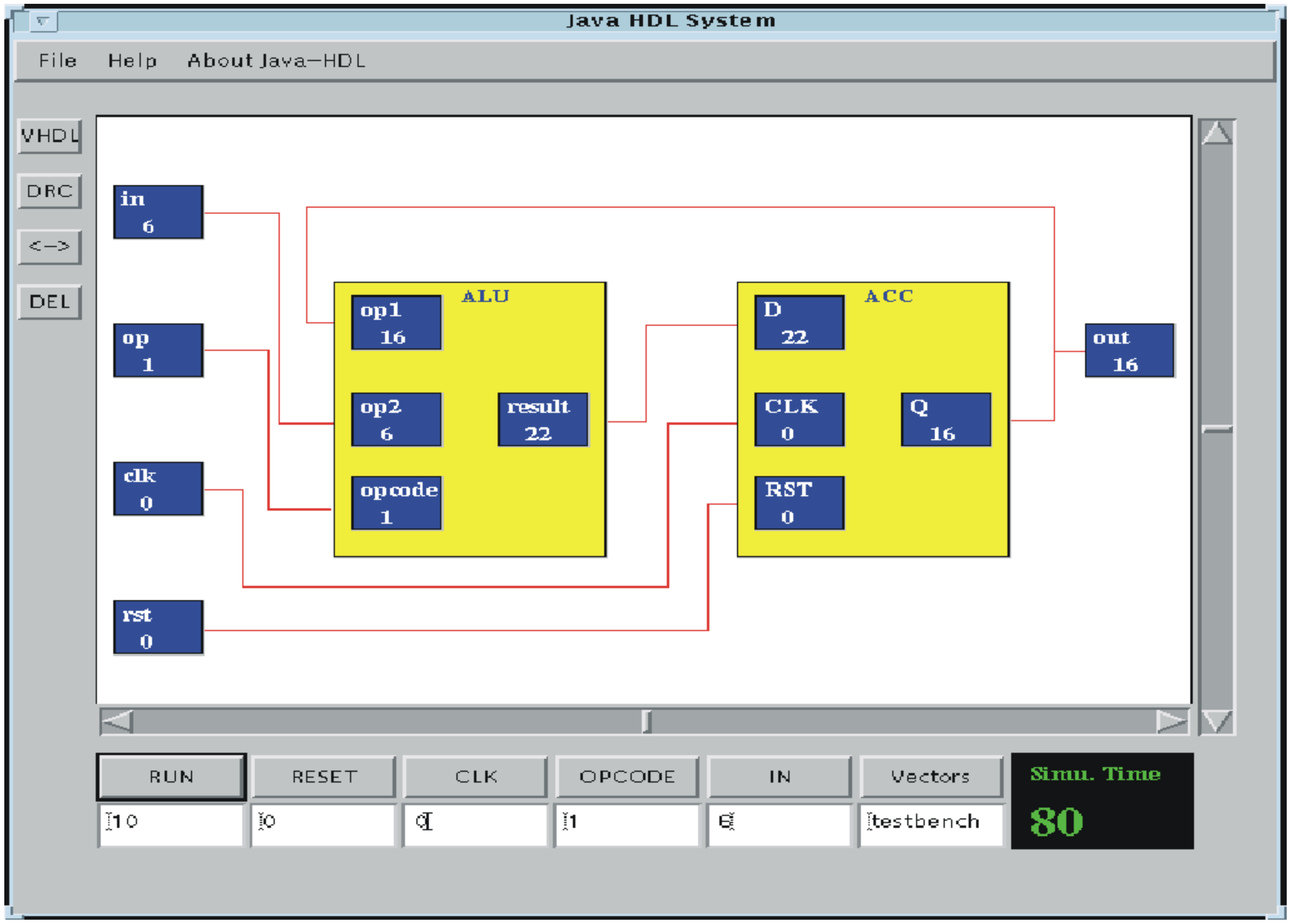
- **Java-HDL System (Experimental, Not shown in TEXPO99)**
  - **Use Java language to describe digital design**
  - **Embedd timing info into wires, and ports so that wire delay can be estimated in early design stage.**
  - **Schematic is automatically generated and is used for visualized simulation**

    => Important for DSP ASIC or pipelined design. You can observe any internal wires without using waveform simulator
  - **Synthesizable VHDL is automatically created**

- **Java-HDL code example:**

```
public class Register ...
{
    Inport p1;
    Inport p2;
      ...
    Wire w5 = new Wire(4);   // a 4-bit wide bus
    Wire w6 = new Wire(1);   // one -bit
      ..
    if ( rst.value == 1 )
        w8.drive(0);
   else if ( event1(clk)  )
        w8.drive(D);
           .
```

Java-HDL Simulation System



RUN	RESET	CLK	OPCODE	IN	Vectors
10	0	0	1	6	testbench

Computer Engineering Group