Application oriented embedded systems

ZEMRIS, 17.10.2008.

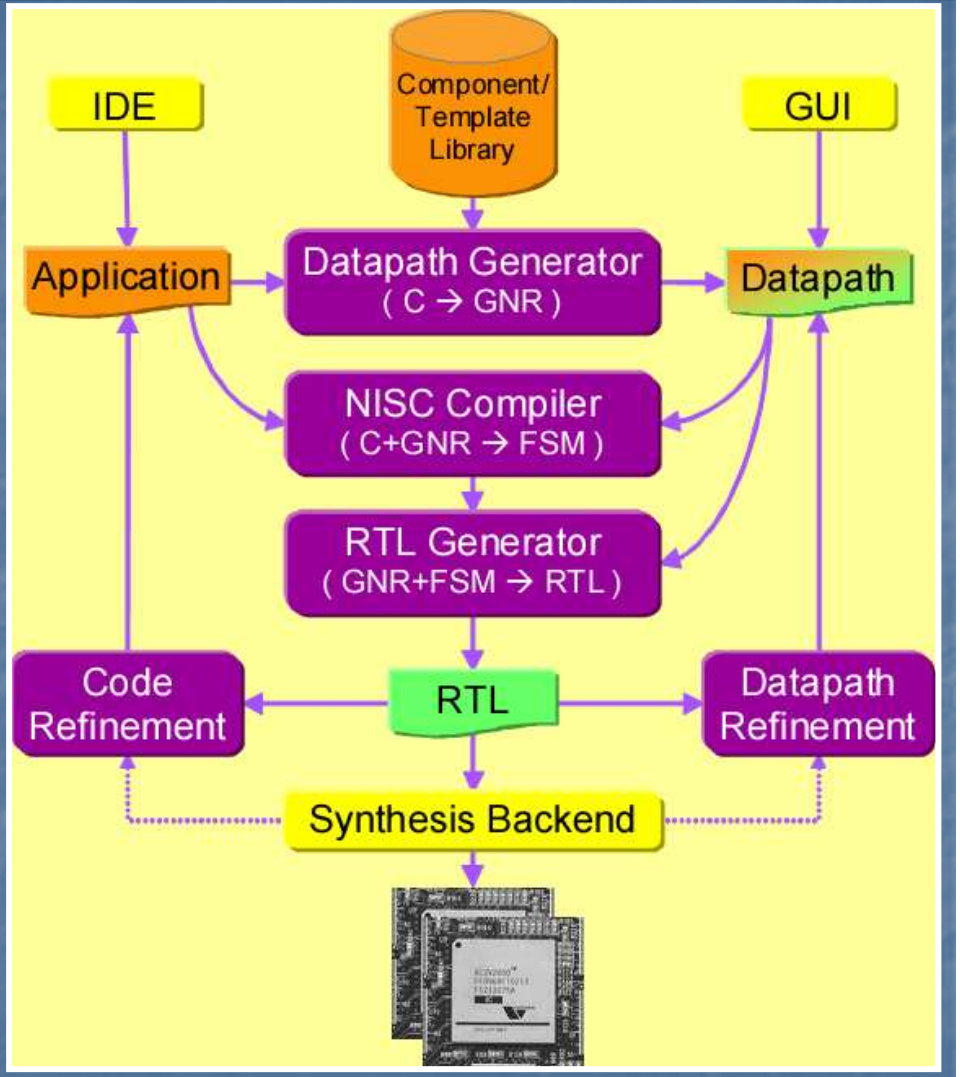
University of Zagreb
Faculty of Electrical Engineering and Computing

Agenda

- Demo
  - NISC Design Flow
  - IP Cores Integration in NISC environment
  - Complex Example
  - NISC as Co-Processor
  - ESE: Embedded System Environment

NISC Technology Toolset (C-to-RTL)

- NISC Technology
  - C-to-RTL Synthesis
  - Embedded Custom-Processor Design
  - Design Space Exploration
- Inputs:
  - C Code
  - Architecture Selection
- Outputs:
  - Synthesizable Verilog

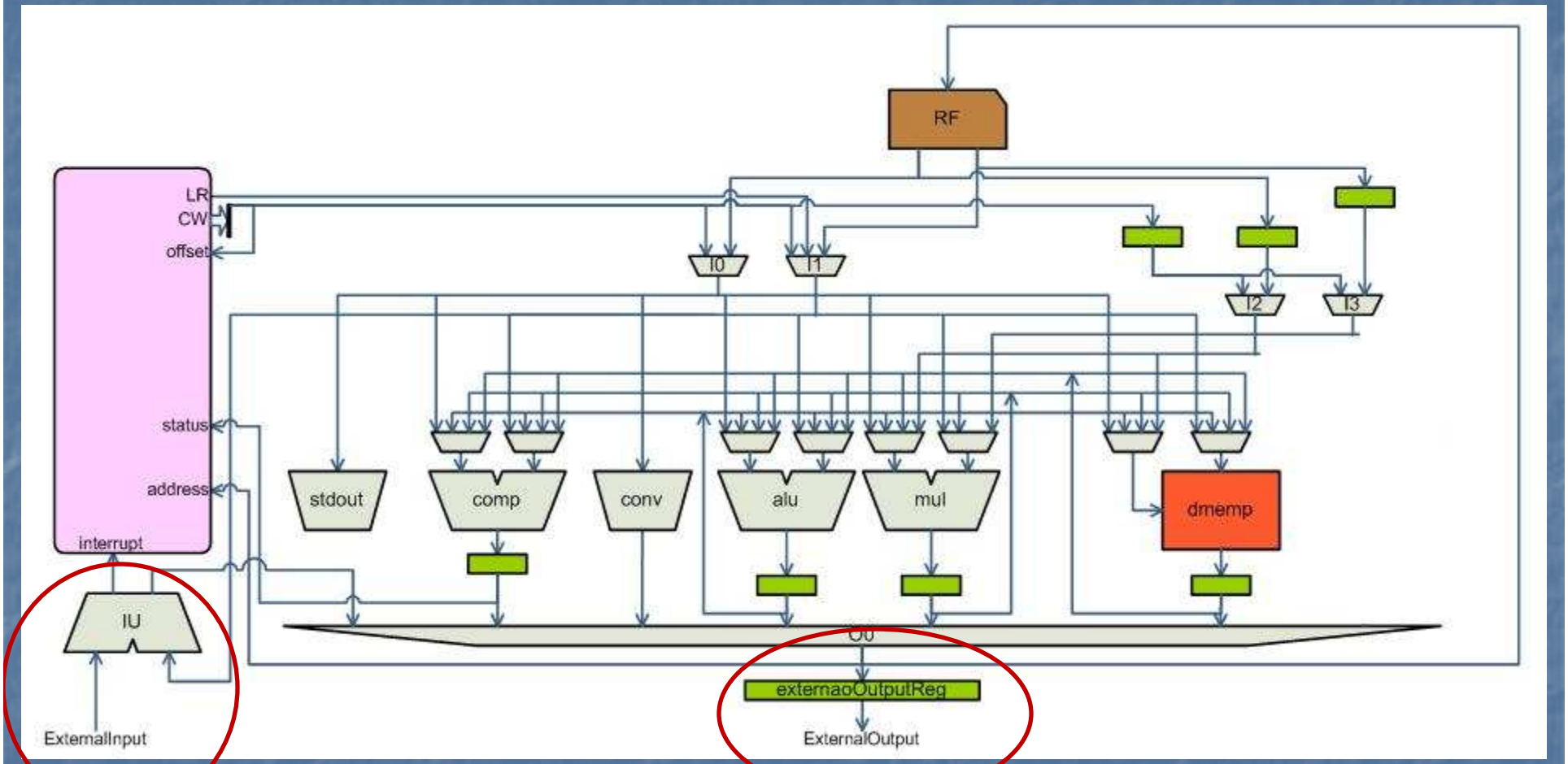


Source: NISC Technology & Toolset (www. .ics.uci.edu/~nisc/)

NISC Design Flow

1. Write application in C
2. Select a NISC architecture
3. Compile C using NISC Toolset
4. Simulate/debug/evaluate
5. If not satisfactory Change program/architecture/both (repeat steps 1-4)
6. Generate RTL for FPGAs/ASICs

NISC Demo



Source: NISC Technology & Toolset

Inputs -> User Pushbuttons

Output -> LCD

NISC Demo C Code

```
#include "processor.h"
/*Please refer to http://www.cecs.uci.edu/~nisc/toolset/release-notes.html to see limitations on
    C code*/

/* The main routine for the Nisc. Your project must have a "void NiscMain()" function. */
#pragma optimize("", off)
void NiscMain()
{
    /*
    * Add your code here.
    */
}
#pragma optimize("", on)


/*The main routine for interrupt handling. Your project must have a "void NiscInterrupt()"
    function. */
void NiscInterrupt()
{
    /*
    * Add your interrupt service routine code here.
    * You can process all interrupts and determine priority here. e.g.
      switch(interrupt_number){case 0: ...}
    * Remember, you need to disable, clear, and enable interrupts using the PreBound functions.
    */
}
```

NISC Demo – C Code

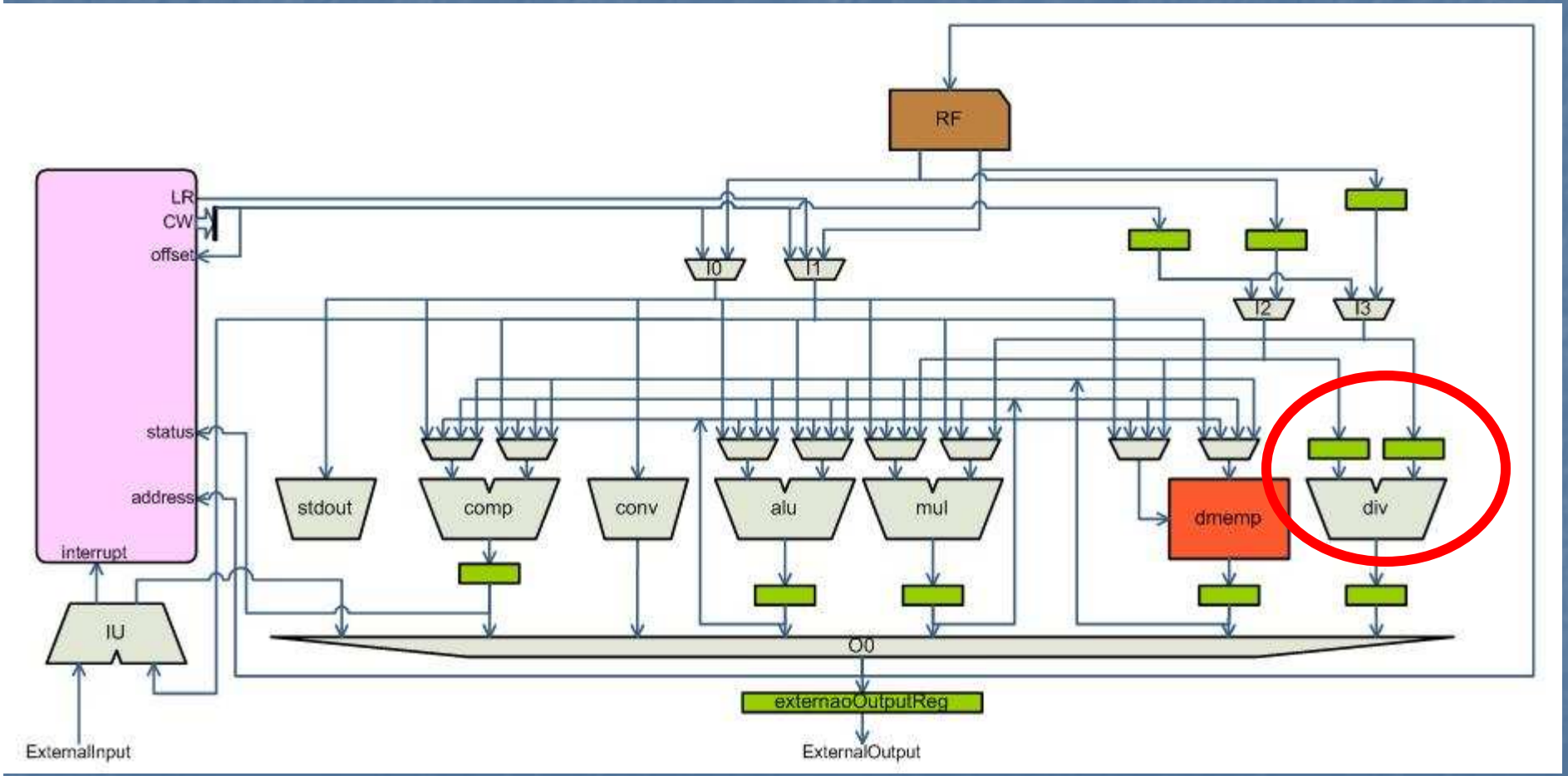
```
#include "processor.h“

#pragma optimize("", off)
void NiscMain()
{
    ...
    switch(iNum)
    {
          case 0: LCDPrintString("EAST"); break;
          case 1: LCDPrintString("CENTER"); break;
          case 2: LCDPrintString("WEST"); break;
          case 3: LCDPrintString("SOUTH"); break;
    }
    ...
}
#pragma optimize("", on)

//The main routine for interrupt handling.
void NiscInterrupt()
{
    iNum = __$IU_interruptNumber(); //Get the current interrupt number
    __$IU_clearInterrupt(iNum); //Clear the current interrupt
}
```

IP Cores Integration in NISC environment

- Simple usage of third-party IP Cores
- Example: Xilinx CoreGenerator

- Divider problem
  - Software divider
    - Slow operation
  - Hardware divider core
    - can help to speed-up the execution
    - Pipelined divisions

IP Cores Integration



Xilinx divider core

- Radix-2 Feature Summary
- Provides quotient with integer or fractional remainder
- Pipelined architecture for increased throughput
- Pipeline reduction for size versus throughput selections
- Dividend width from 2 to 32 bits
- Divisor width from 2 to 32 bits
- Independent dividend, divisor and fractional bit widths
- Fully synchronous design using a single clock
- Supports unsigned or two's complement signed numbers
- Can implement 1/X (reciprocal) function
- Fully registered outputs

Figure 1: **Core Pinout Diagram**

Source: Xilinx, Pipelined ***divider v2.0***, LogiCore Product Specification

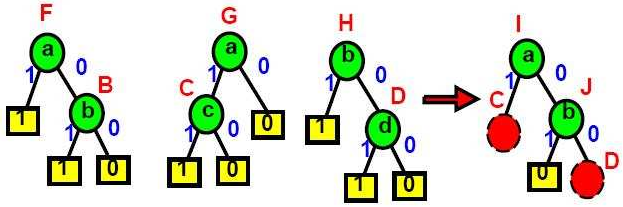
NISC benefit

- Based on IP technology
- Allows perfect design tuning to application
- Produces RTL for custom implementation
- Single compiler/simulator for all NISC designs
- C codes compiled directly to HW
- Unifies SW and HW concepts
- Simplifies design, tools, education, design science
- Fast Prototype

- Design Goal Verification?
  - Behavioral Simulation
    - Does not show the actual timing relations of the final implementation
  - Post Place and Route Simulation
    - Takes too long (over a day!)
  - FPGA board implementation

Complex Example

- Binary Decision Diagram trees
  - Used in Formal Verification techniques



F,G,H,I,J,B,C,D
are pointers

$$
\begin{aligned}
I &= ite(F,G,H) \\
&= (a, ite(F_a, G_a, H_a), ite(F_{\overline{a}}, G_{\overline{a}}, H_{\overline{a}})) \\
&= (a, ite(1, C, H), ite(B, 0, H)) \\
&= (a, C, (b, ite(B_b, 0_b, H_b), ite(B_{\overline{b}}, 0_{\overline{b}}, H_{\overline{b}}))) \\
&= (a, C, (b, ite(1, 0, 1), ite(0, 0, D))) \\
&= (a, C, (b, 0, D)) \\
&= (a, C, J)
\end{aligned}
$$

Check:

$$
\begin{aligned}
F &= a + b \\
G &= ac \\
H &= b + d
\end{aligned}
$$

$$
\begin{aligned}
ite(F,G,H) &= (a+b)(ac) + \overline{a}\overline{b}(b+d) \\
&= ac + \overline{a}\overline{b}d
\end{aligned}
$$

Complex Example Evaluation

- The goals:
  - Measuring run-time
  - The times of specific application sections
  - The time of operations

  - PC Intel Core2 1.86GHz 6μs
  - Virtex-5 NISC 33MHz 387μs
  - Virtex-5 NISC 33MHz 127μs
  - No optimization!!!
    - < 60 μs expected

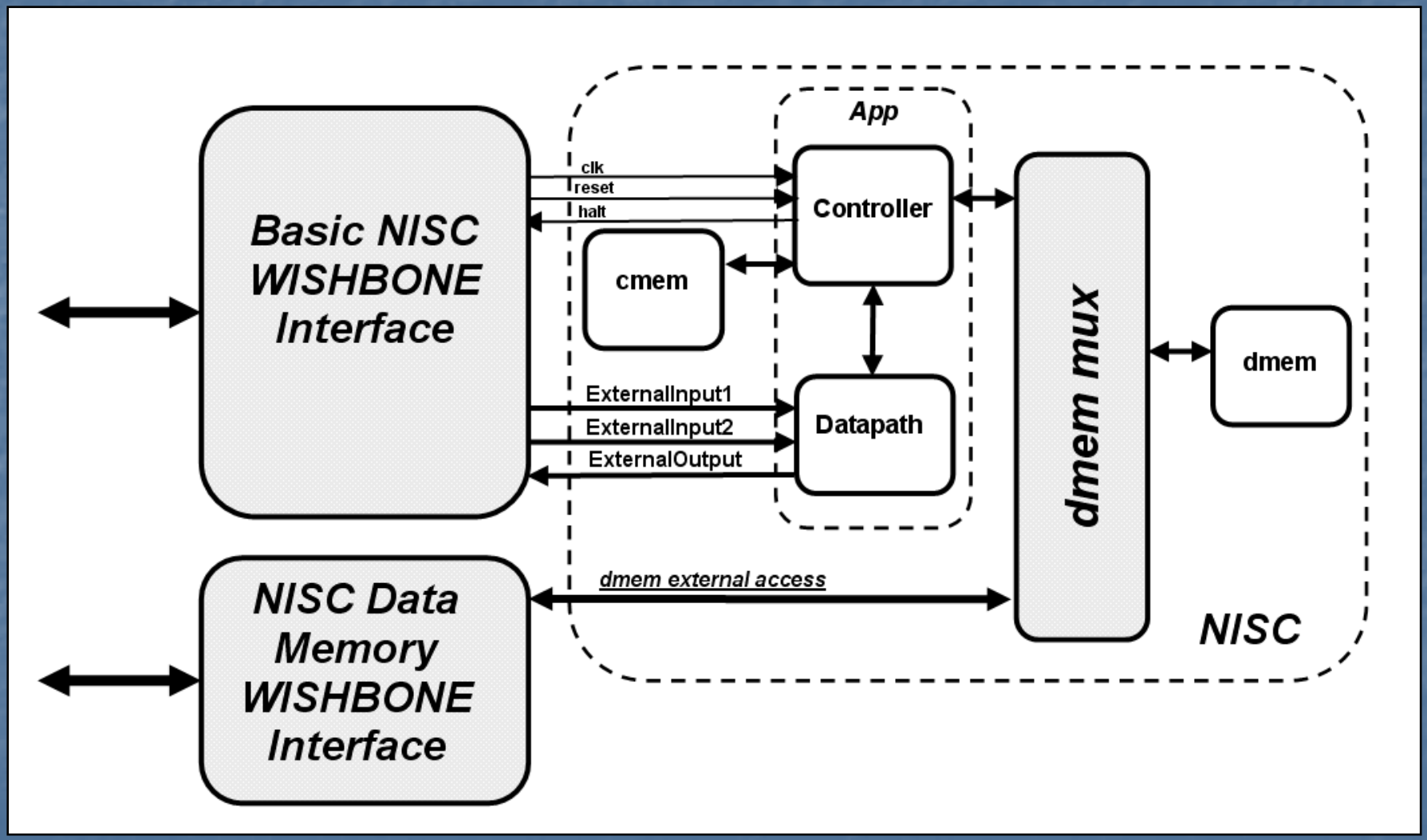
Achieved Results

- Case study on DCT algorithm

	No. of cycles	Clock Freq	DCT exec. time(us)	Power (mW)	Enegy (uJ)	Normalized area
NMIPS	10772	78.3	137.57	177.33	24.40	1.00
CDCT1	3080	85.7	35.94	120.52	4.33	0.81
CDCT2	2952	90.0	32.80	111.27	3.65	0.71
CDCT3	2952	114.4	25.80	82.82	2.14	0.40
CDCT4	3080	147.0	20.95	125.00	2.62	0.46
CDCT5	3208	169.5	18.93	106.00	2.01	0.43
CDCT6	3208	171.5	18.71	104.00	1.95	0.34
CDCT7	3460	250.0	13.84	137.00	1.90	0.35

Source: B. Gorjiara, M. Reshadi, D. Gajski, "Designing a Custom Architecture for DCT Using NISC Technology"

NISC as a Coprocessor



NISC as a Coprocessor

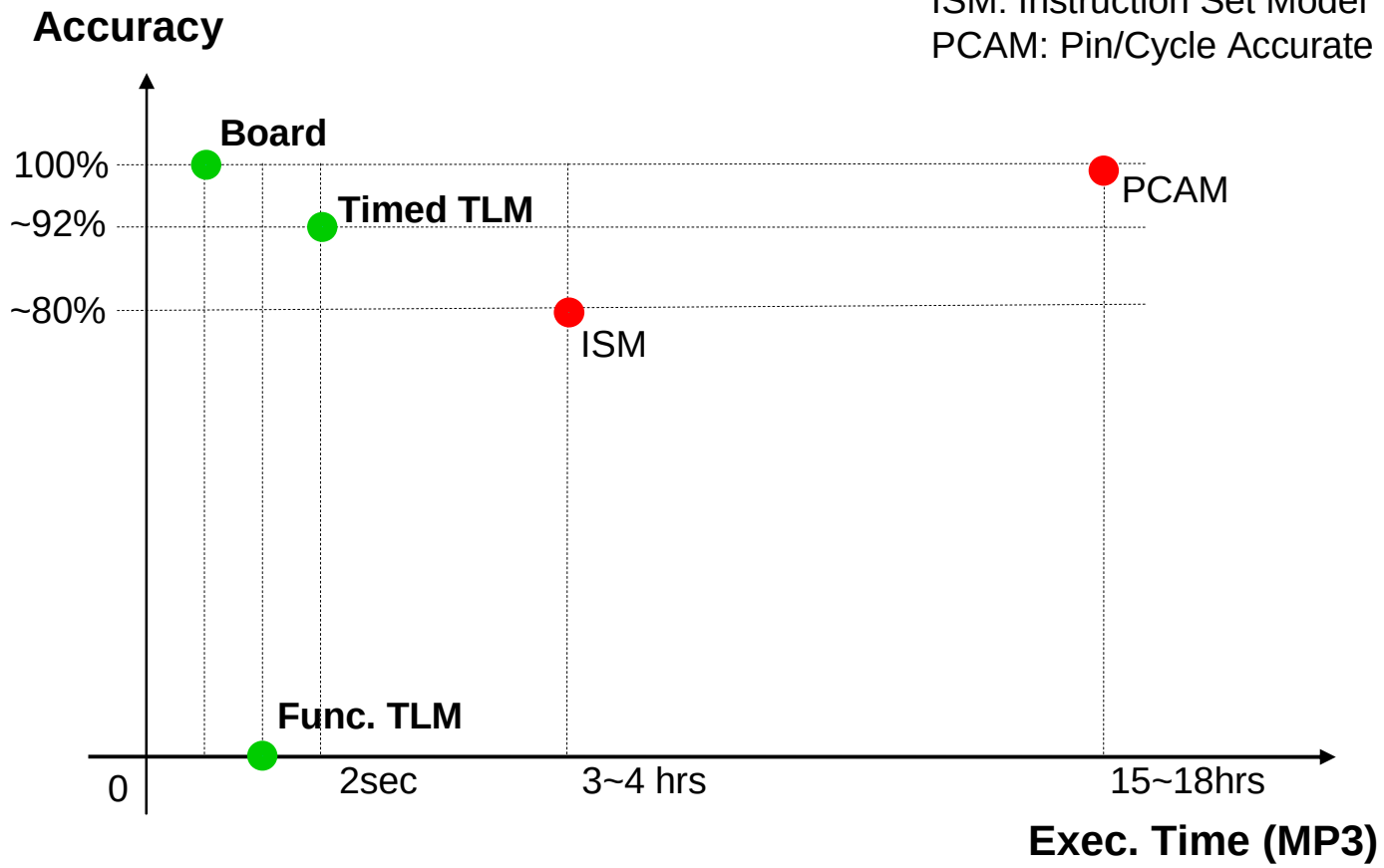
- Simple functions
  - `result = function(arg1, arg2, …, argN);`
  - Using arguments as direct inputs to datapath's functional units
  - Direct output of results
- Data transforms
  - `result[i:j] = function(inputs[k:l]);`
  - Direct access to NISC's data memory
  - Transfer
- Completion detection through pooling or interrupts

ESE: Embedded System Environment

- **Technology advantages**
- **No basic change in design methodology required**
  - ES methodology follows present manual design process
- **Productivity gain of more than 1000X demonstrated**
  - Designers do not write models
- **Simple change management: 1-day change**
  - No rework for new design decisions
- **High error-reduction: Automation + verification**
  - Error-prone tasks are automated
- **Simplified globally-distributed design**
  - Fast exchange of design decisions and easy impact estimates
- **Benefit through derivatives designs**
  - No need for complete redesign
- **Better market penetration through customization**
- **Shorter Time-to-Market through automation**

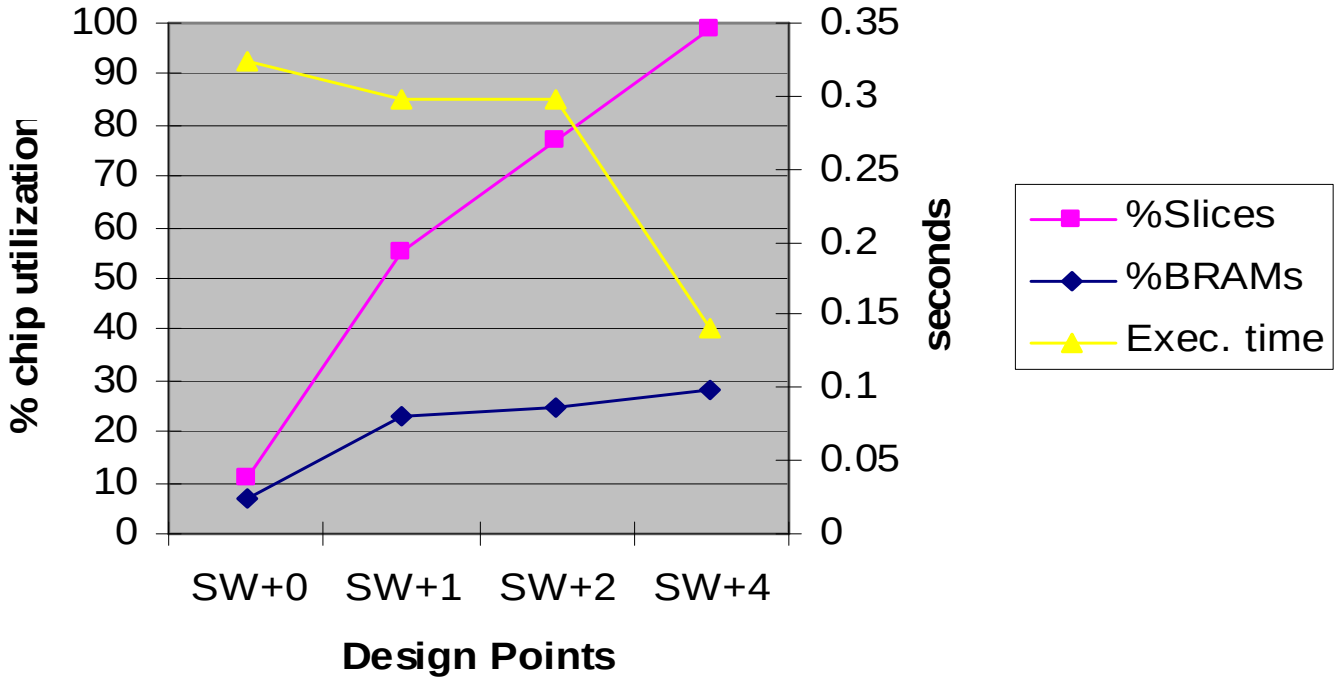
Model Accuracy vs. Execution Time

TLM: Transaction Level Model
ISM: Instruction Set Model
PCAM: Pin/Cycle Accurate Model



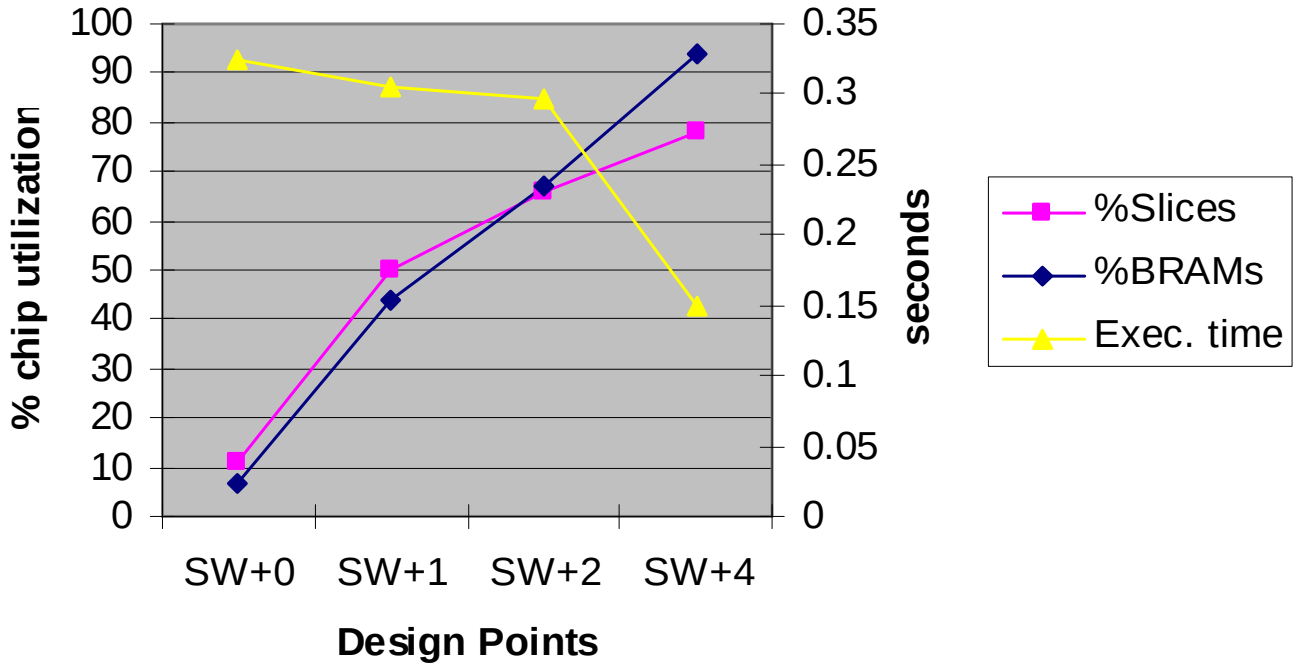
Time and accuracy trade off among different models

Design Quality: Manual



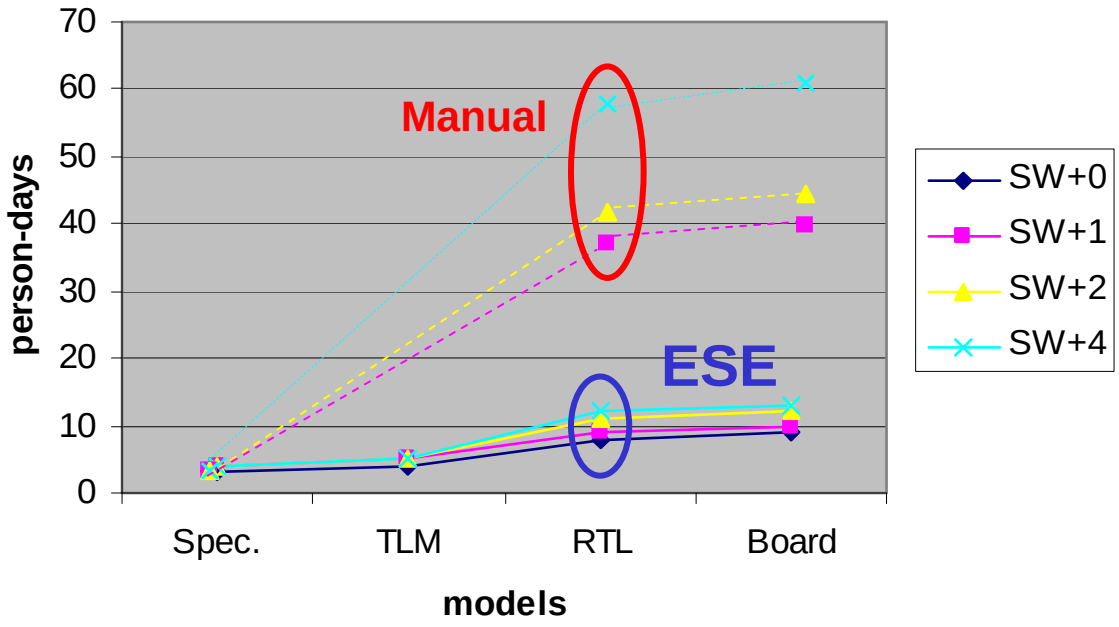
- **Area**
  - % of FPGA slices and BRAMS
- **Performance**
  - Time to decode 1 frame of MP3 data

Design Quality: ESE



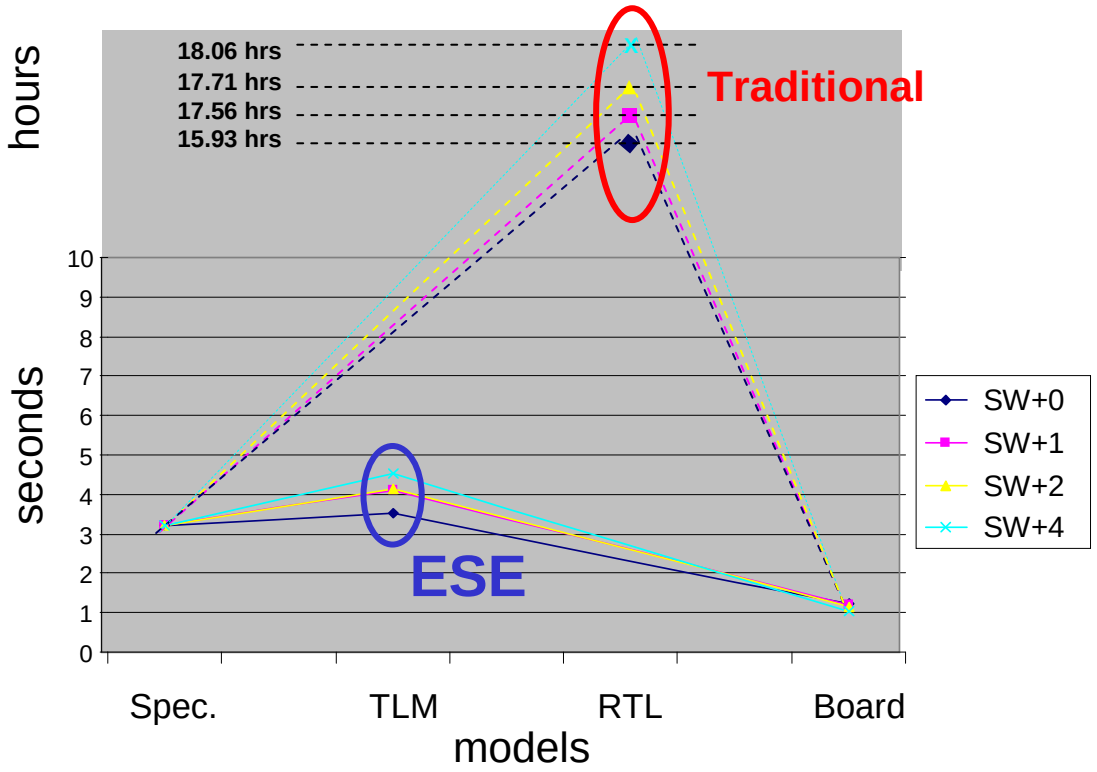
- **Area**
  - ESE designs use fewer FPGA slices and more BRAMs than manual HW
- **Performance**
  - ESE designs execute at same speed as manual designs

Development Time: ESE vs. Manual



- **ESE drastically cuts RTL and Board development time**
  - Manual development includes months of RTL coding
  - Models can be developed at Spec level with ESE
  - TLM, RTL and Board models are generated automatically by ESE

Validation Time: ESE vs. Traditional



- **ESE cuts validation time from hours to seconds**
  - No need to verify RTL models for every design change
  - Designers can perform high speed validation with TLM and board

ESE Back-end Advantages

- HW synthesis in ESE removes the need to code and debug large RTL HDL models
- Transducer and interface synthesis allows flexibility to include heterogeneous IP in the design
- SW driver synthesis removes the need for SW developers to understand HW details
- SW and HW application can be easily upgraded at TL and validated on board
- C and graphical input of TL model allows even non-experts to develop and test HW/SW systems with ESE

References

- Daniel D. Gajski: “NISC: The Ultimate Reconfigurable Component", *Center for Embedded Computer Systems, TR 03-28*
- NISC Home: http://www.ics.uci.edu/~nisc
- ESE Home: http://www.cecs.uci.edu/~ese