EESTEC-ZG Workshop

Lecture 4:
Introduction to HW/SW Co-Design using the NISC Technology

Daniel Gajski, Vlado Sruk, Roko Grubišić

University of Zagreb
Faculty of Electrical Engineering and Computing
vlado.sruk@fer.hr

Lecture overview

- Recap
- UKF project
- HW/SW Co-Design
- No Instruction Set Computer

Unity Through Knowledge Fund

- *Project: “Application-Oriented Embedded System Technology”*
- Pricipal investigator: Prof. Daniel Gajski
  - University of California Irvine
  - Center of Embedded Computer Systems
- www.ukf.hr

UNITYTHROUGH
KNOWLEDGEFUND

The complexity of embedded systems

- Embedded systems are no longer used as simple control devices
- Complex tasks and algorithms and high performance requirements require complex hardware platforms
  - Multiprocessor System-On-Chip (MPSoC)
  - System on Programmable Chips (SoPC)
- Heterogeneous multiprocessor systems
  - General purpose processors
  - Application specific coprocessors
  - Bus architectures
- Complex software
- Complex hardware/software interface

Constraints

- The need for rapid development of embedded systems
  - Short time-to-market (TTM)
  - Reliability?
- Stringent constraints
  - Speed
  - Area
  - Power
- There is an evident need for new approaches to the design of embedded systems

HW/SW Co-Design

- Hardware/software co-design means meeting system-level objectives by exploiting the synergism of hardware and software through their concurrent design
- Unlike traditional design approaches, co-design views the process of developing hardware and software as a whole
- Tasks (parts of an algorithm) can migrate between software and hardware
- We evaluate the benefits and draw-backs of software and hardware implementations of a particular function
- Automatic generation of hardware?

Different implementations of an algorithm

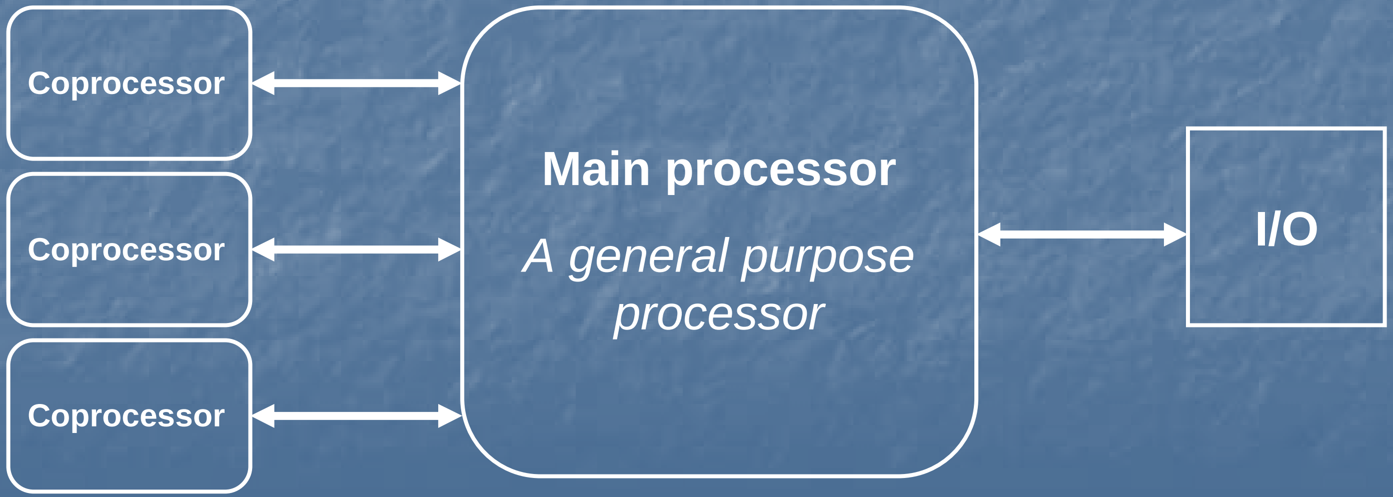
- Software implementation
  - Programming
  - Higher designer productivity
  - Shorter Time To Market
  - Lower performance
- Hardware implementation
  - RTL modeling
  - Lower designer productivity
  - Longer Time To Market
  - Higher performance

The ideal system?

- We're looking for an optimal system to perform a given task
- What is optimal depends on the constraints
- Design Space Exploration
  - Try to determine which part of the system's task should be implemented in hardware and which in software
- The key for enhancing the performance is migrating the right part of an algorithm to hardware

System design

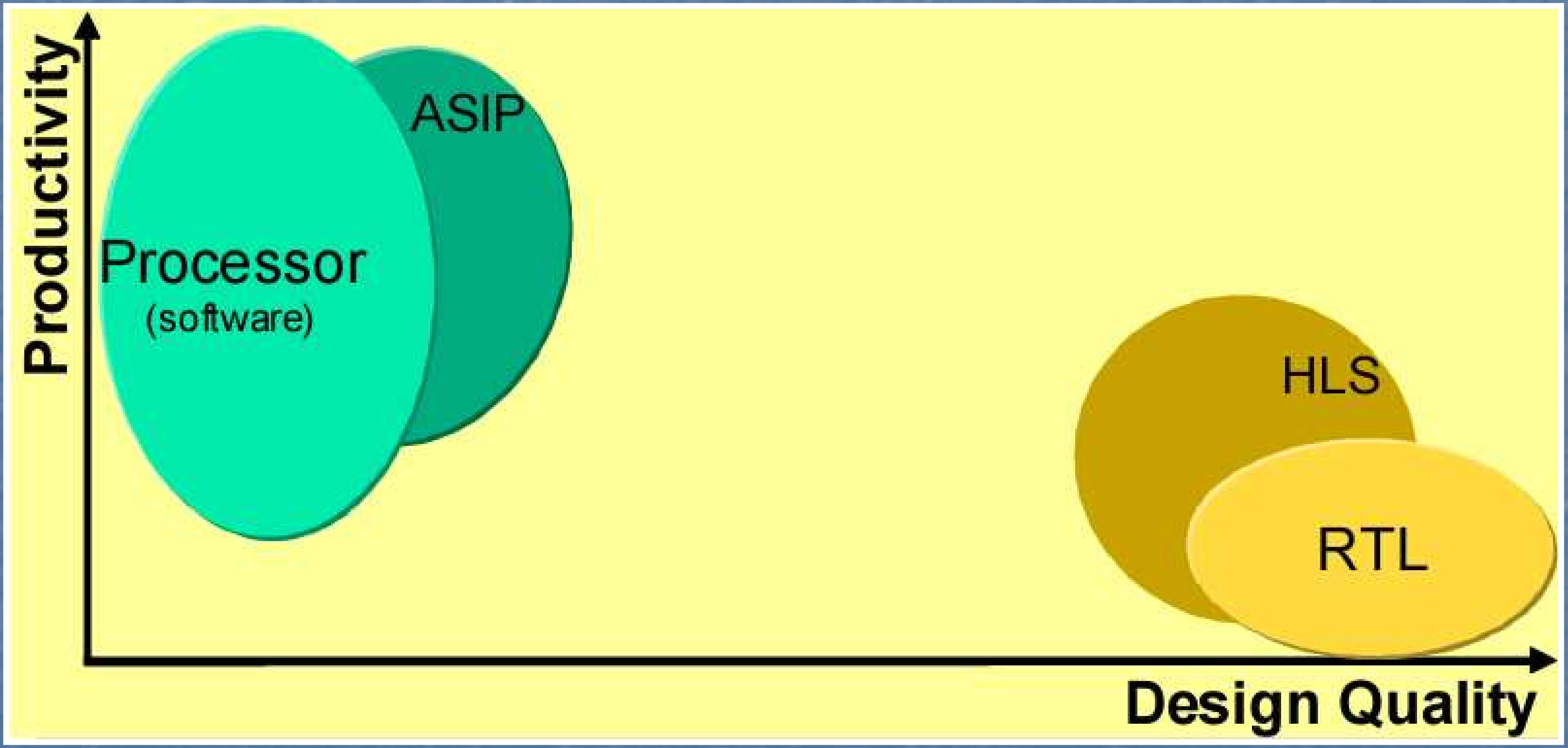
- General purpose processors
- Application specific coprocessors
  - Automatic generation?
- System integration
  - Standard bus architectures?



Automatic generation of hardware

- Raising the level of abstraction
  - Synthesize hardware from a high-level description and not from a RTL description
  - High-Level Synthesis (HLS)
  - C-to-hardware tools
- Expanding existing processors with custom instructions
  - Application-Specific Instruction set Processors (ASIPs)

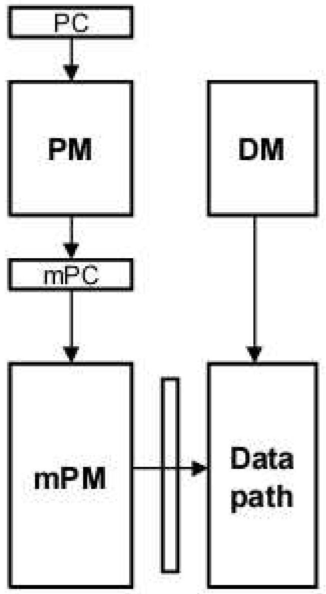
Design productivity

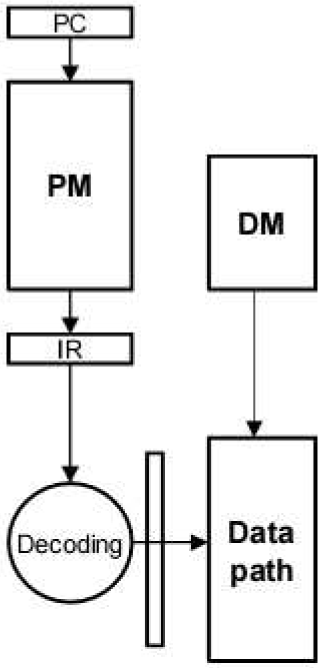


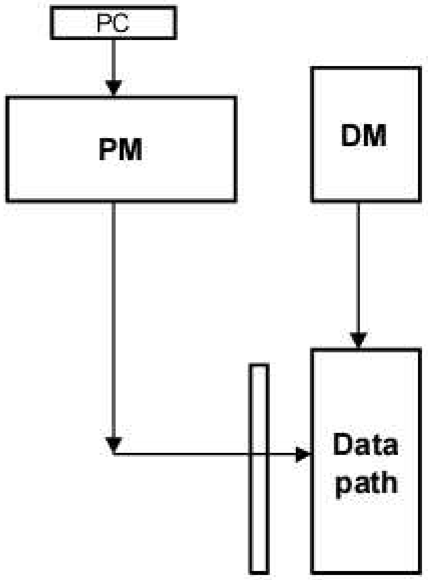
NISC – No Instruction Set Computer

- A novel concept developed by Prof. Daniel D. Gajski and his team at University of California at Irvine (UCI)
- The idea is to provide automatic generation of custom processors (IPs) from high-level description while retaining the ability to predictably control the quality of the final outcome
- The main characteristic of the NISC approach is the elimination of the instruction abstraction
- The high-level programming language code is complied directly to a datapath
- Horizontally microprogrammed control unit

CISC Vs. RISC Vs. NISC







CISC

Complex instructions possible
1 Instruction = n microinstructions

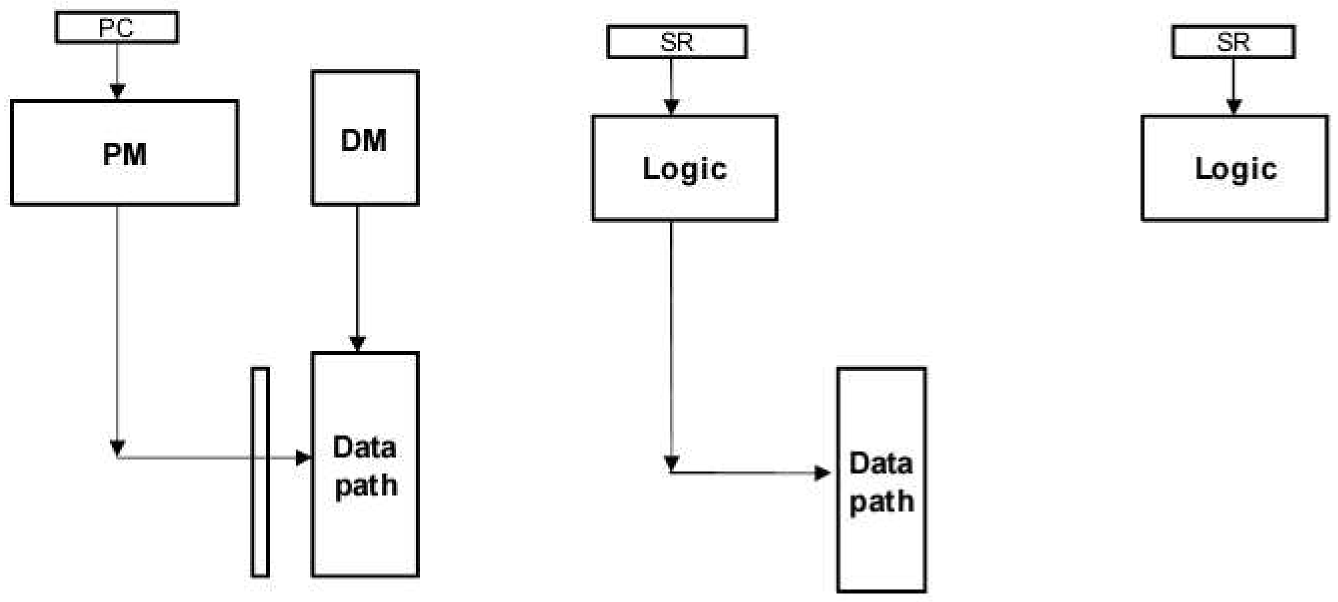
RISC

Simple instructions
No microprogramming
RISC PM = 2X CISC PM

NISC

No instruction, only control words
NISC PM = RISC PM

NISC Styles



NISC

No instruction, only control words

RTL Processor

PM is implemented with gates
PC is equal to SR (State register)
Datapath is simple

FSM

No Datapath, no Data memory
Simple controller

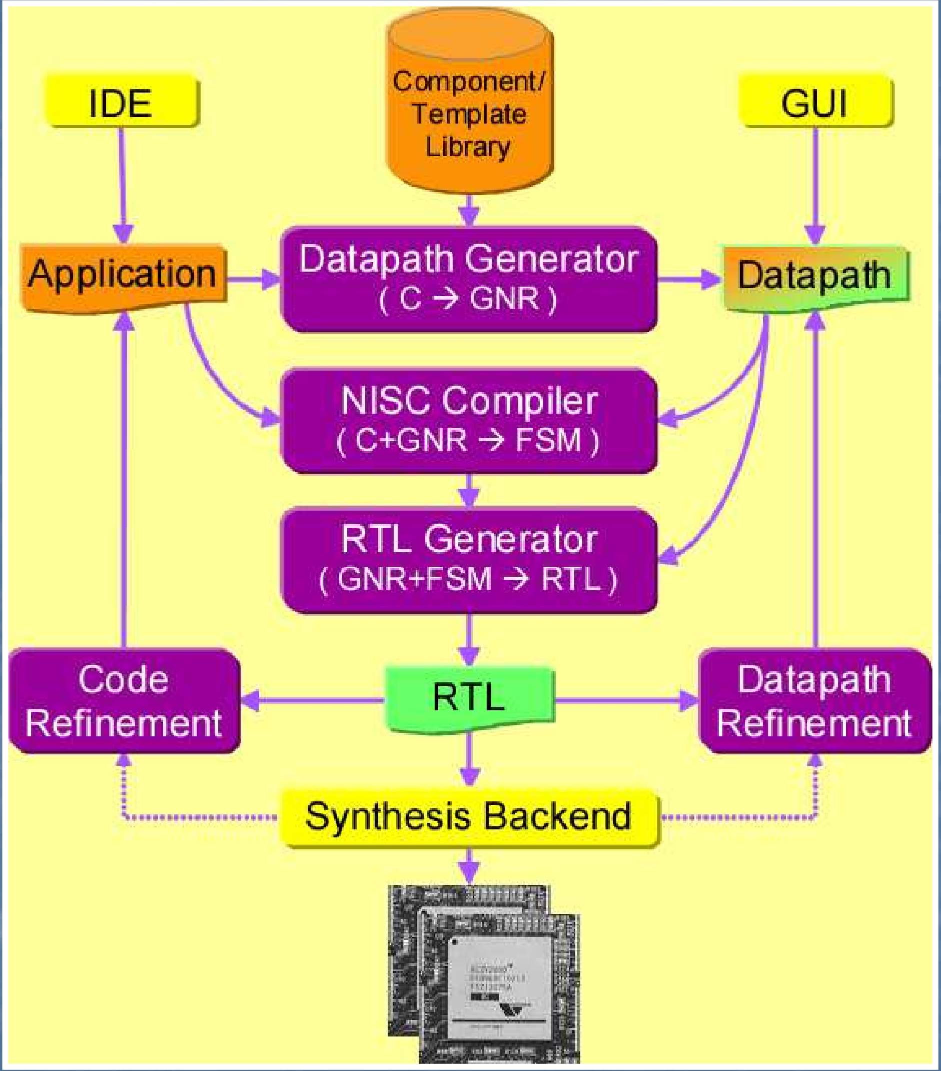
NISC methodology

1. Write an application in C
2. Select a NISC architecture (manually or automatically)
3. Compile C for the selected architecture
4. Simulate/debug/evaluate the result
5. Repeat 1-4 if not satisfactory
6. Generate RTL for FPGA/ASIC

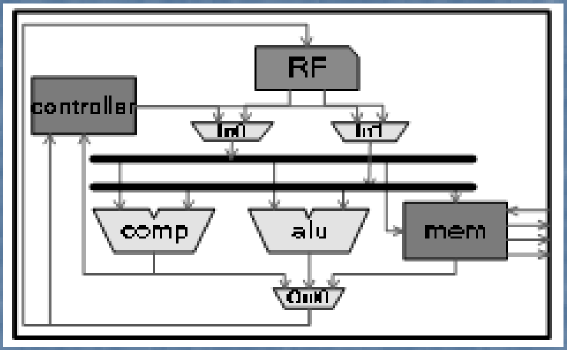
NISC Technology Toolset (C-to-RTL)

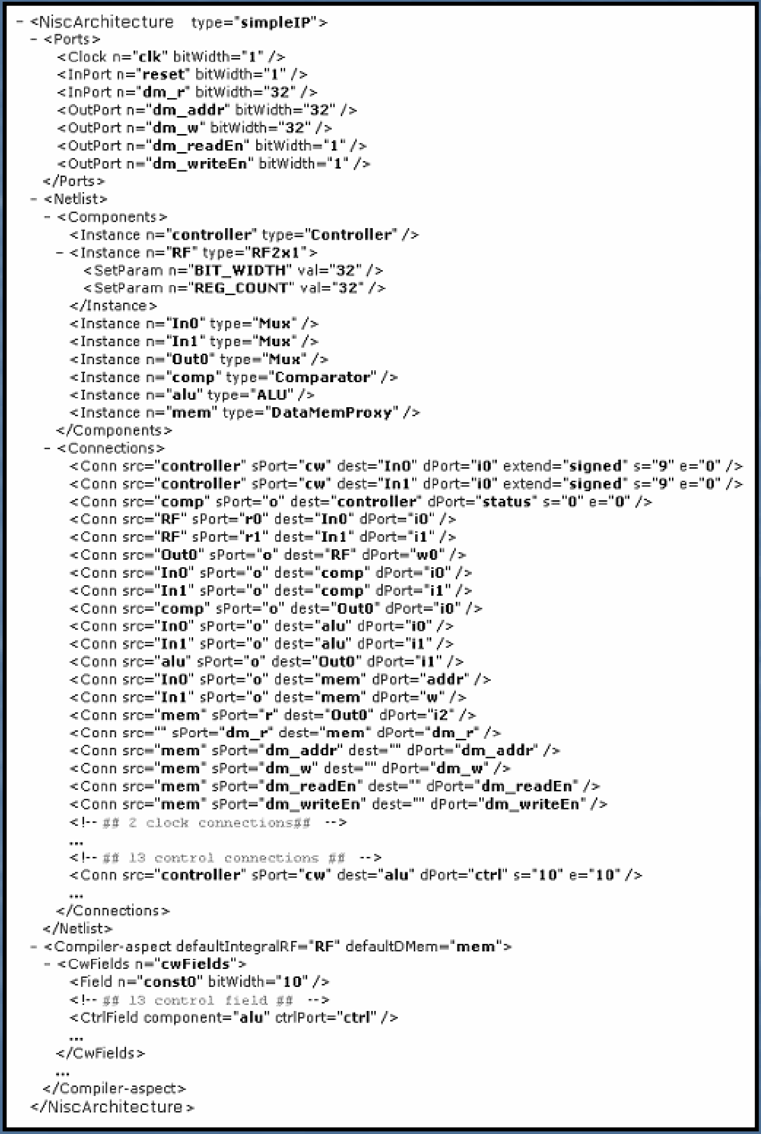
- Datapath Generator / Selector
  - Generates/Selects datapath for a given application
  - Converts C => GNR (Generic Netlist Representation of datapath)
- NISC Cycle-accurate Compiler
  - Compiles the application for a given datapath
  - Converts C => CW (Control Words stream controlling components in each clock cycle)
- RTL Generator
  - Generates the RTL for input to FPGA or ASIC
  - Converts GNR+CW => RTL
- Datapath Refinement
  - Refine datapath to improve quality
- Code Refinement
  - Refine application to improve quality

NISC Technology Toolset (C-to-RTL)


IDE
Component/ Template Library
GUI
Application
Datapath Generator (C → GNR)
Datapath
NISC Compiler (C+GNR → FSM)
RTL Generator (GNR+FSM → RTL)
Code Refinement
RTL
Datapath Refinement
Synthesis Backend

Generic Netlist Representation





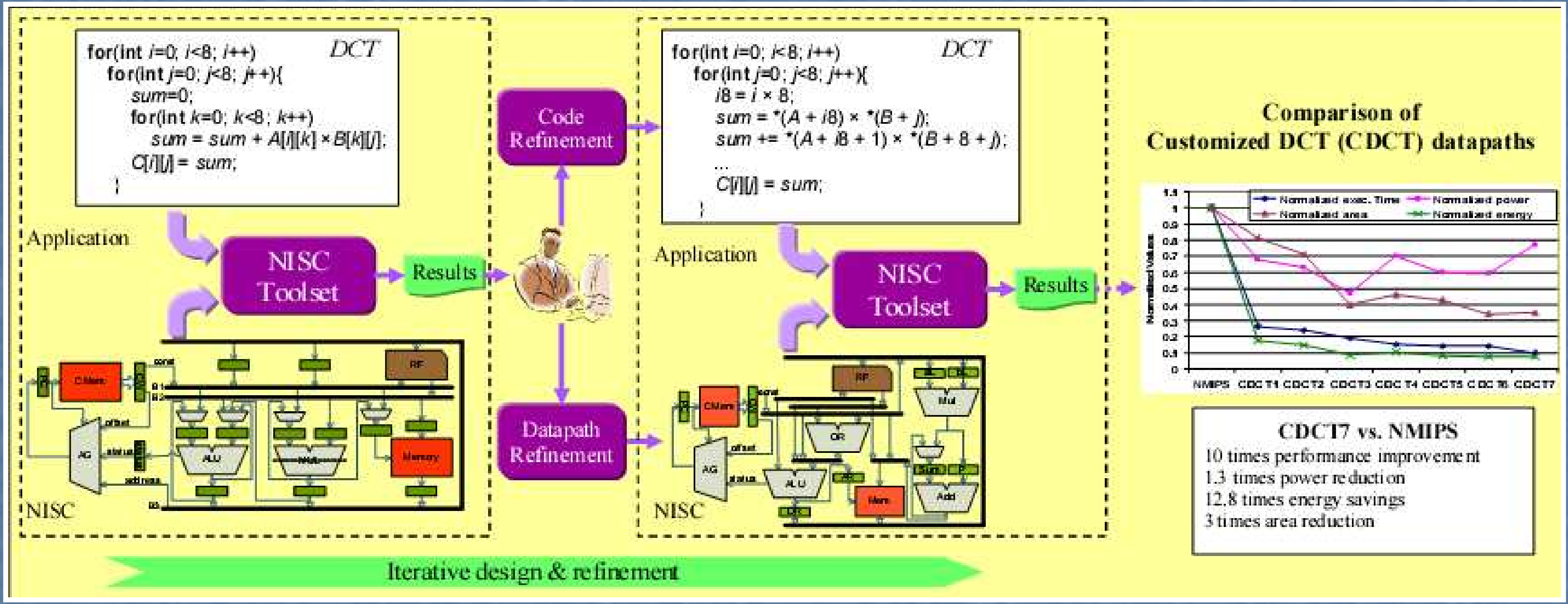
```
- <NiscArchitecture   type="simpleIP">
  - <Ports>
      <Clock n="clk" bitWidth="1" />
      <InPort n="reset" bitWidth="1" />
      <InPort n="dm_r" bitWidth="32" />
      <OutPort n="dm_addr" bitWidth="32" />
      <OutPort n="dm_w" bitWidth="32" />
      <OutPort n="dm_readEn" bitWidth="1" />
      <OutPort n="dm_writeEn" bitWidth="1" />
    </Ports>
  - <Netlist>
    - <Components>
        <Instance n="controller" type="Controller" />
      - <Instance n="RF" type="RF2x1">
          <SetParam n="BIT_WIDTH" val="32" />
          <SetParam n="REG_COUNT" val="32" />
        </Instance>
        <Instance n="In0" type="Mux" />
        <Instance n="In1" type="Mux" />
        <Instance n="Out0" type="Mux" />
        <Instance n="comp" type="Comparator" />
        <Instance n="alu" type="ALU" />
        <Instance n="mem" type="DataMemProxy" />
      </Components>
    - <Connections>
        <Conn src="controller" sPort="cw" dest="In0" dPort="i0" extend="signed" s="9" e="0" />
        <Conn src="controller" sPort="cw" dest="In1" dPort="i0" extend="signed" s="9" e="0" />
        <Conn src="comp" sPort="o" dest="controller" dPort="status" s="0" e="0" />
        <Conn src="RF" sPort="r0" dest="In0" dPort="i0" />
        <Conn src="RF" sPort="r1" dest="In1" dPort="i1" />
        <Conn src="Out0" sPort="o" dest="RF" dPort="w0" />
        <Conn src="In0" sPort="o" dest="comp" dPort="i0" />
        <Conn src="In1" sPort="o" dest="comp" dPort="i1" />
        <Conn src="comp" sPort="o" dest="Out0" dPort="i0" />
        <Conn src="In0" sPort="o" dest="alu" dPort="i0" />
        <Conn src="In1" sPort="o" dest="alu" dPort="i1" />
        <Conn src="alu" sPort="o" dest="Out0" dPort="i1" />
        <Conn src="In0" sPort="o" dest="mem" dPort="addr" />
        <Conn src="In1" sPort="o" dest="mem" dPort="w" />
        <Conn src="mem" sPort="r" dest="Out0" dPort="i2" />
        <Conn src="" sPort="dm_r" dest="mem" dPort="dm_r" />
        <Conn src="mem" sPort="dm_addr" dest="" dPort="dm_addr" />
        <Conn src="mem" sPort="dm_w" dest="" dPort="dm_w" />
        <Conn src="mem" sPort="dm_readEn" dest="" dPort="dm_readEn" />
        <Conn src="mem" sPort="dm_writeEn" dest="" dPort="dm_writeEn" />
        <!-- ## 2 clock connections##  -->
        ...
        <!-- ## 13 control connections ##  -->
        <Conn src="controller" sPort="cw" dest="alu" dPort="ctrl" s="10" e="10" />
        ...
      </Connections>
    </Netlist>
  - <Compiler-aspect defaultIntegralRF="RF" defaultDMem="mem">
    - <CwFields n="cwFields">
        <Field n="const0" bitWidth="10" />
        <!-- ## 13 control field ##  -->
        <CtrlField component="alu" ctrlPort="ctrl" />
        ...
      </CwFields>
      ...
    </Compiler-aspect>
  </NiscArchitecture>
```

- XML-based Architecture Description Language (ADL)

NISC Technology Design Flow

- In NISC Technology, the designer can iteratively refine to get satisfactory results.
- Architecture selection:
  - It can be selected from a set of pre-designed templates optimized for a specific application domain.
  - A customized architecture can be generated for a specific application by specifying netlist of components after profiling and analyzing the application.
- Compilation: The compiler maps the input C code on the given architecture.
- RTL generation: RTL generator translates the NISC netlist and the output of compiler into synthesizable RTL.
- Synthesis: The final result can be simulated and synthesized for final implementation on FPGA or ASIC.
- Refinement: After analysis, application and/or architecture can be redefined until the desired results are achieved.

Code and datapath refinement



```
for(int i=0; i<8; i++)                    DCT
   for(int j=0; j<8; j++){
      sum=0;
      for(int k=0; k<8; k++)
         sum = sum + A[i][k] × B[k][j];
      C[i][j] = sum;
   }
```

```
for(int i=0; i<8; i++)                    DCT
   for(int j=0; j<8; j++){
      i8 = i × 8;
      sum = *(A + i8) × *(B + j);
      sum += *(A + i8 + 1) × *(B + 8 + j);

      ...
      C[i][j] = sum;
   }
```

CDCT7 vs. NMIPS

10 times performance improvement
1.3 times power reduction
12.8 times energy savings
3 times area reduction

NISC Technology Strength

- Both top-down and bottom-up design flow
  - Standard behavioral synthesis (a.k.a. high-level synthesis or HLS) tools provide only a top-down design flow (from high-level C to low-level RTL)
  - ASIP approaches provide only a bottom-up design flow (from custom-instructions up to using them in the application)
  - NISC provides both top-down flow (by generating architecture from C) and bottom-up flow (by providing datapath as input)

NISC Technology Strength

- Fast & Predictable path to implementation
  - When using behavioral synthesis, a change in the application will result in many changes in the generated RTL
    - There is no way to predict or correlate the application changes to the changes in the results
    - Therefore the only way to improve unsatisfactory results is by try-and-error and guess-work
  - When using ASIP, after adding a new custom instruction, the designer should either use HLS to synthesis the datapath and controller (unpredictable results), or must do it manually (very time consuming)
  - NISC enables the designers to control every aspect of the design
    - Designer can select the exact points for improvement and then do it quickly
    - For example, by directly changing the GNR description of architecture, the designer can reduce a critical path delay or fix complex multiplexers and connections that costume too much power or make the layout unroutable

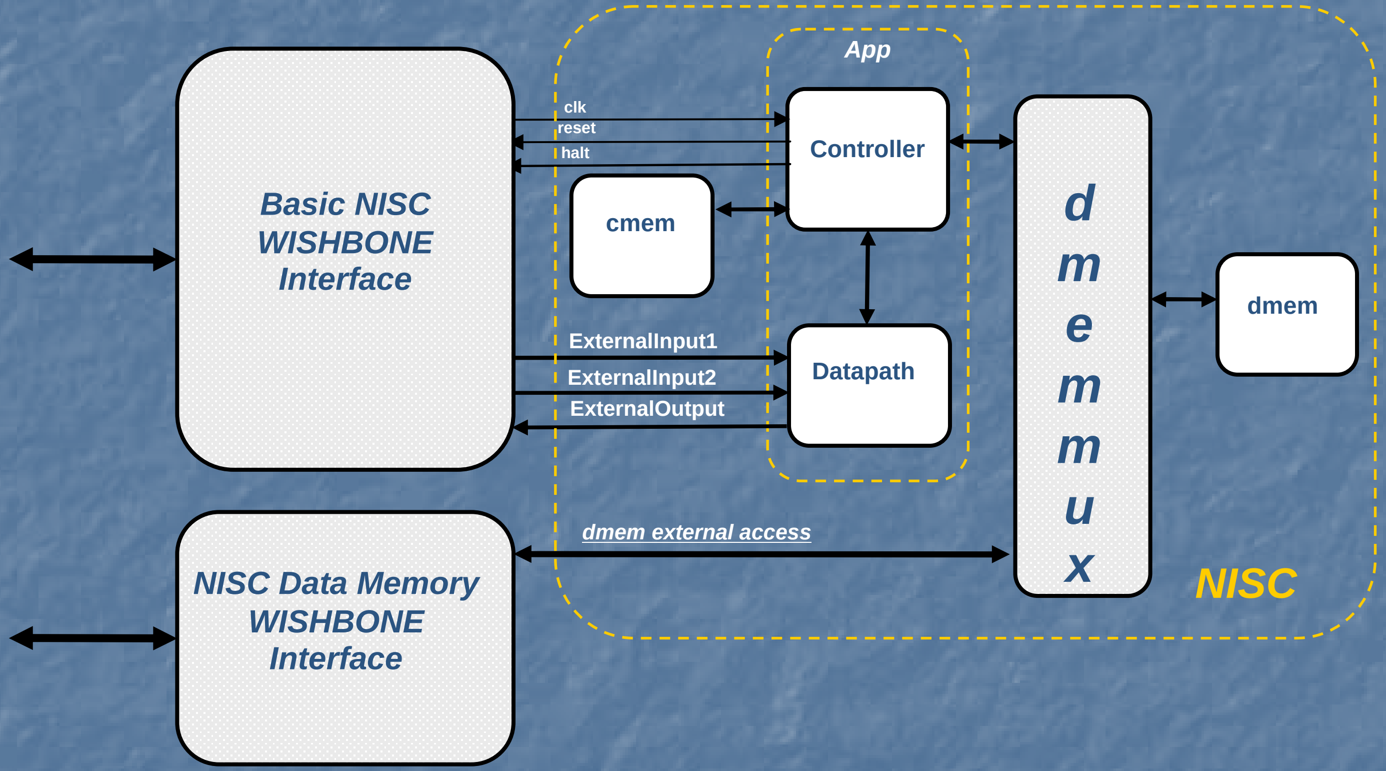
NISC Technology Strength

- Designer-controled design space exploration
  - Since other tools and techniques do not provide fast & predictable path to implementation, the designer can at best explore the design aspects that the tool provides but not the ones the designer wants!
  - Sine datapath can be an input in NISC Technology, the designer can explore options for quality metrics selectively
  - For example, designer can focus on dynamic power minimization by modifying the connections or gating or latching them in the datapath description and quickly see the effect on the final results

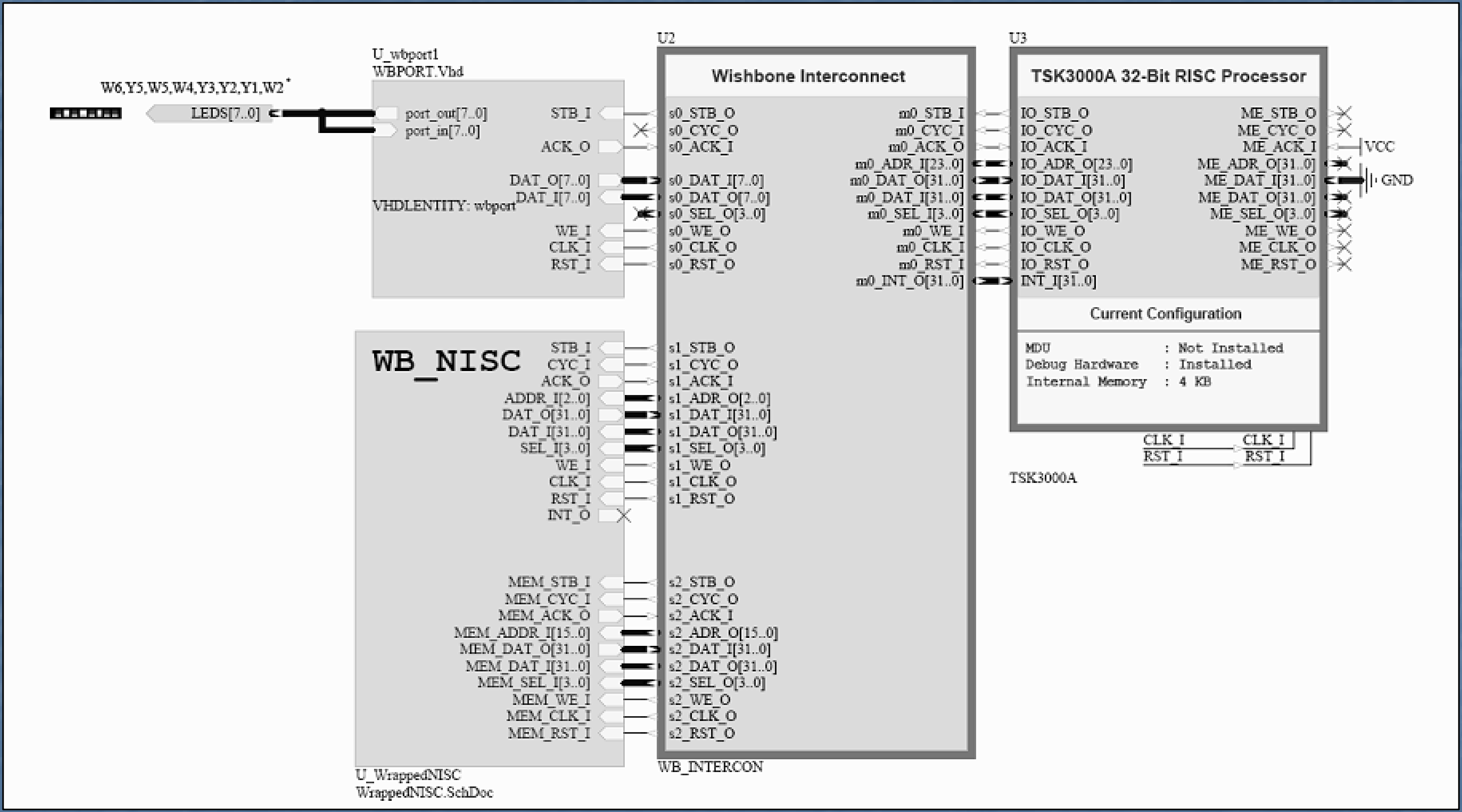
NISC Technology Strength

- Right mixture of standard languages
  - Since ANSI C alone is not enough for designing a complete system, all HLS tools provide their own extension of C as input language
  - Such extensions require extra effort for learning. Additionally, they prevent the use of unmodified C code and lock the designer into a proprietary tool by making the developed algorithms not-portable to other tools.
  - NISC Technology only uses standard languages
  - Designers do not need extra training and also can use their favorite development tools such as editors and debuggers
  - NISC provides a seamless way of combining standard C and Verilog through pre-bound functions enable the inclusion of low-level Verilog code inside a C program

NISC on WISHBONE



NISC as coprocessor



References and further reading

- D. Gajski, "NISC: The Ultimate Reconfigurable Component", *Center for Embedded Computer Systems, TR 03-28*, October 2003.
- NISC Technology & Toolset, URL: http://www.ics.uci.edu/~nisc/
- M. Reshadi, D. Gajski, "A Cycle-Accurate Compilation Algorithm for Custom Pipelined Datapaths", *International Symposium on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp 21-26, September 2005.
- B. Gorjiara, M. Reshadi, D. Gajski, "Generic Architecture Description for Retargetable Compilation and Synthesis of Application-Specific Pipelined IPs", *International Conference on Computer Design (ICCD)*, October 2006.