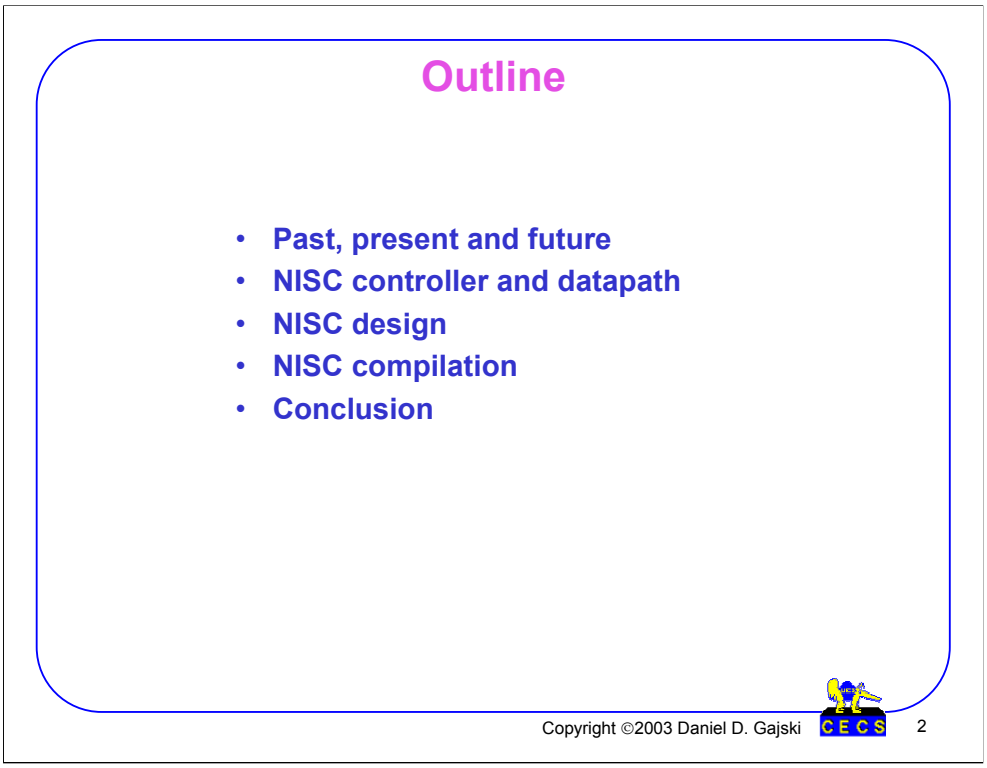
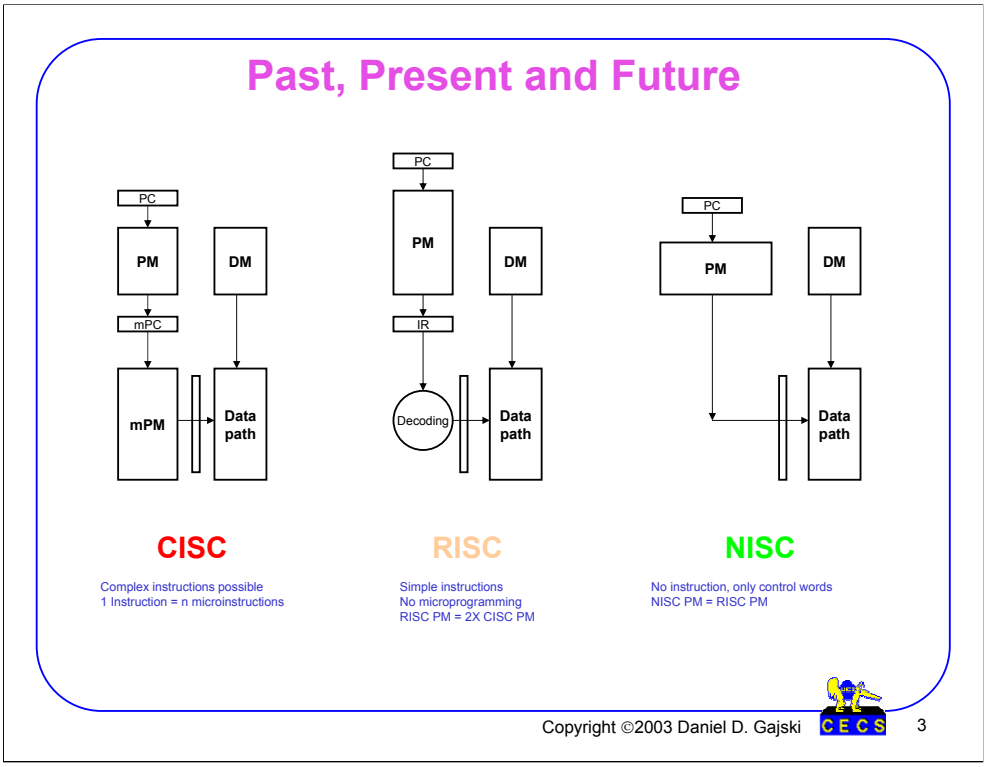
Introduction

With complexities of Systems-on-Chip rising almost daily, the design community has been searching for new methodology that can handle given complexities with increased productivity and decreased times-to-market. The obvious solution that comes to mind is increasing levels of abstraction, or in other words, increasing the size of the basic building blocks. However, it is not clear how many of these building blocks we need and what these basic blocks should be. Obviously, the necessary building blocks are processors and memories. One interesting question is: “Are they sufficient?”. The other interesting question is: “How many types of processors and memories we really need?”. In this report we try to answer both of these questions and argue that the No-instruction-set computer (NISC) is a single, necessary and sufficient processor component for design of any digital system.



Outline

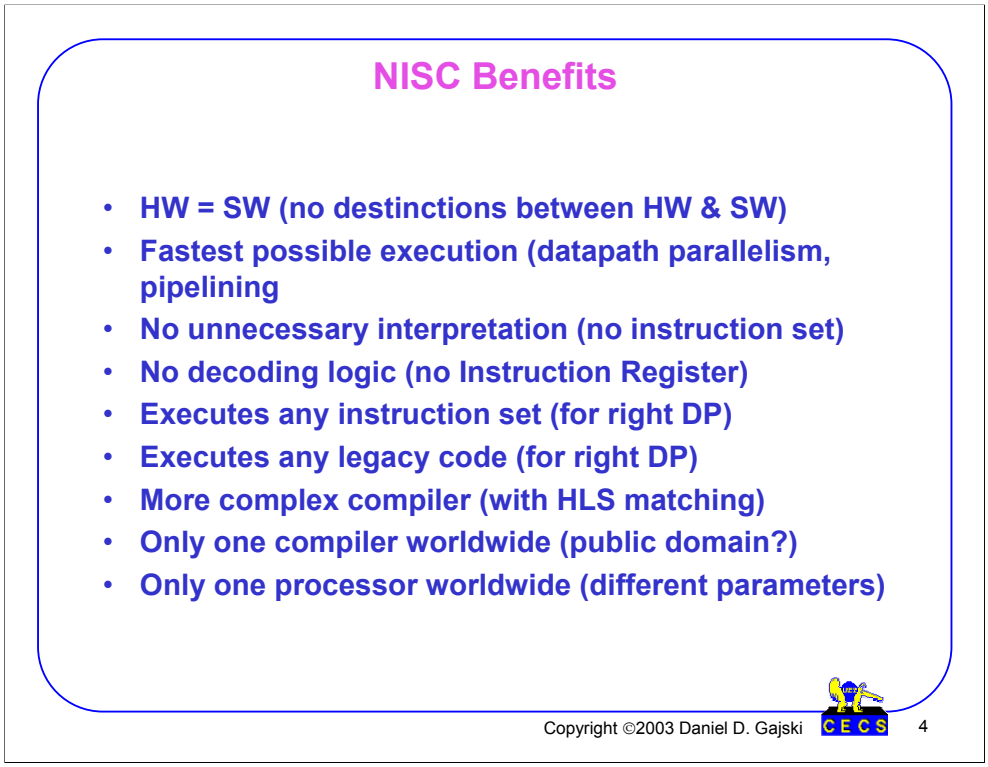
In order to introduce the concept of NISC processor and its benefits, we will first compare NISC features to those of CISC and RISC processors. Then, we will introduce the architecture of NISC controller and NISC datapath. In the second part we will demonstrate a simple methodology for design of the parametrizable and reconfigurable NISC processor and its compiler. We will conclude with the advantages of NISC processor and its capability to unite SW and HW approaches in design and education.



History of processor architecture

The evolution of the processor architecture can be divided into three phases.

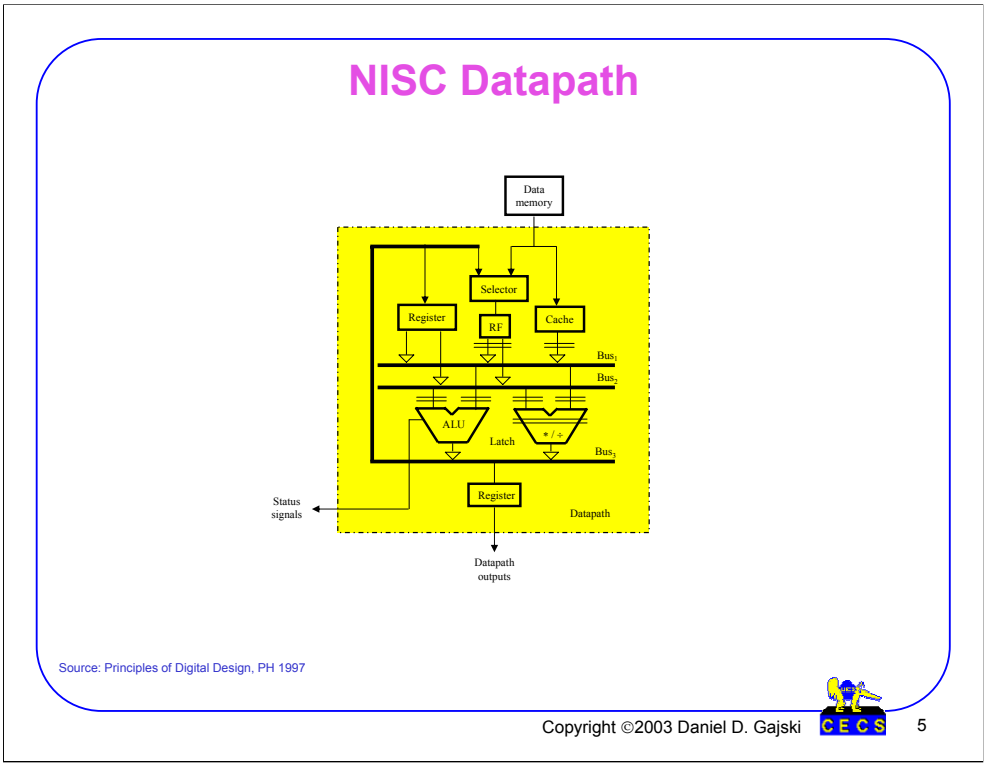
1. Complex-instruction-set computer (CISC) was popular in 1970s. Since the program memory (PM) was slow, designers tried to improve performance by constructing complex instructions. Each complex instruction took several clock cycles, with Datapath control words for each clock cycle were stored in a much faster micro program memory (mPM). The concept of micro programming allowed for emulation of any instruction set and construction of specialized instructions, while speeding up the execution. Unfortunately, micro programming did not allow for efficient pipelining of the Datapath.
2. Reduced-instruction-set computer (RISC) became popular in late 1980s by eliminating complex instructions and the mPM. All instructions in a RISC are simple and execute in one clock cycle allowing Datapath to be efficiently pipelined in 4-8 pipelined stages. The mPM was replaced with decoding stage, that followed the instruction fetch from PM. Since instructions are simpler, a RISC needs approximately two instructions for each complex instruction and, therefore, the size of the PM is doubled. However, the Fetch-Decode-Execute-Store pipeline of the whole processor improved the execution speed several times.
3. The No-instruction-set computer (NISC), introduced here, completely removes the decode stage and stores the control word in the PM. Since control words are 2-3 times wider then instructions the PM increased in width by 2-3 times. Fortunately, each control word can execute 2-3 RISC instruction. Therefore, NISC PM = RISC PM. Furthermore each NISC is parametrizable and reconfigurable, which allows for very fine tuning to any application and performance.



NISC benefits

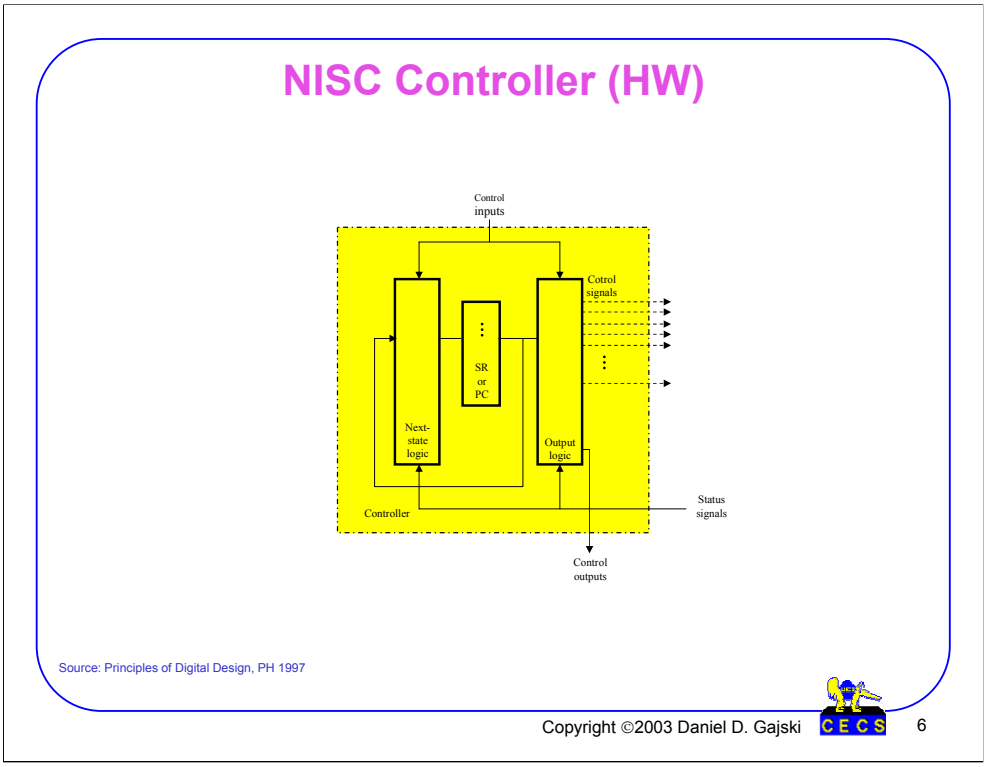
NISC architecture has several benefits.

1. The distinction between SW and HW implementation disappears. For HW implementation the control words are in ROM or gate logic, while for SW implementation they are in a RAM.
2. Since DP can be pipelined by introducing any number of stages and since DP can have any level of parallelism, it is difficult to outperform NISC.
3. Since there is no instruction set, NISC eliminates the last stage of interpretation between C code and HW. C code runs directly on HW (DP).
4. NISC can emulate any instruction set, since NISC control word can execute any operation as long as DP resources in DP are available. Therefore, any legacy code can be executed on a properly defined NISC by converting legacy instructions into NISC control words through a table look up.
5. The NISC compiler uses the High-Level Synthesis algorithms for covering parse tree with control words.
6. Since NISC is a sufficient component for any computation, only one compiler is needed world wide. Hopefully, such a compiler will be in public domain.
7. Similarly, only one NISC processor, although in different versions and with different parameters, is needed world wide. That uniqueness will simplify education, design, trade, maintenance, testing and many other aspects of system design, in similar fashion as gate libraries led to standardization of digital design.



NISC Datapath

Any C program can be executed on a NISC processor, which consists of a Controller and the Datapath. Datapath consists of a set of storage elements (registers, register files, memories), set of functional units (ALUs, multipliers, shifters, custom functions) and set of busses. All these components may be allocated in different quantities and types and connected arbitrarily through busses. Each component may take one or more clock cycles to execute, each component may be pipelined and each component may have input or output latches or registers. The entire Datapath can be pipelined in several stages in addition to components being pipelined themselves. The Controller defines the state of the processor and issues the control signals for the Datapath.

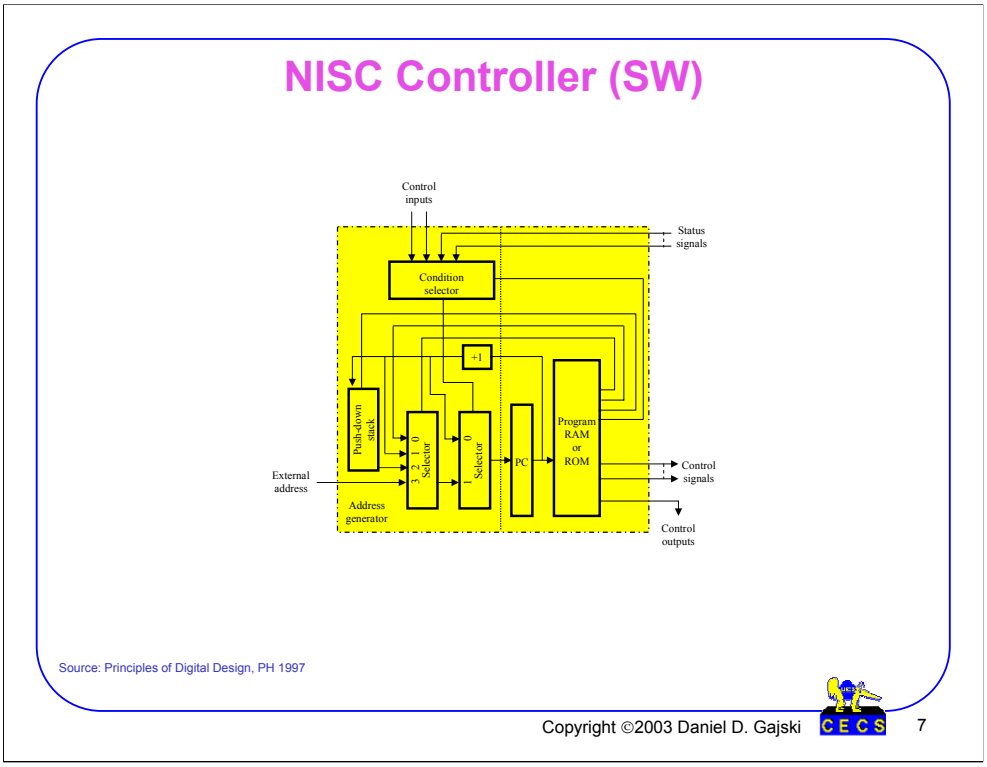


NISC Controller (Fixed implementation)

NISC controller generates a sequence of control words in order to execute a computation specified by the C program. If the sequence is short and it will not change over time, the Controller can be implemented with gates and a state register (SR). This implementation can be easily specified by a finite-state machine model (FSM).

The controller has Control inputs and outputs from the external environment and provides outputs to the external environment. It also gets the status signals from the Datapath and provides the control signals, collectively called control word to the Datapath. The Controller consists of a State register (SR), Next-state logic and Output logic. SR stores the present state of the processor which is equal to the present state of the FSM model describing the operation of the Controller. The Next-state logic computes the next state to be loaded into the SR, while the Output logic generates the Control signals and the Control outputs. The Next-state and Output logic are combinatorial circuits that are implemented with gates.

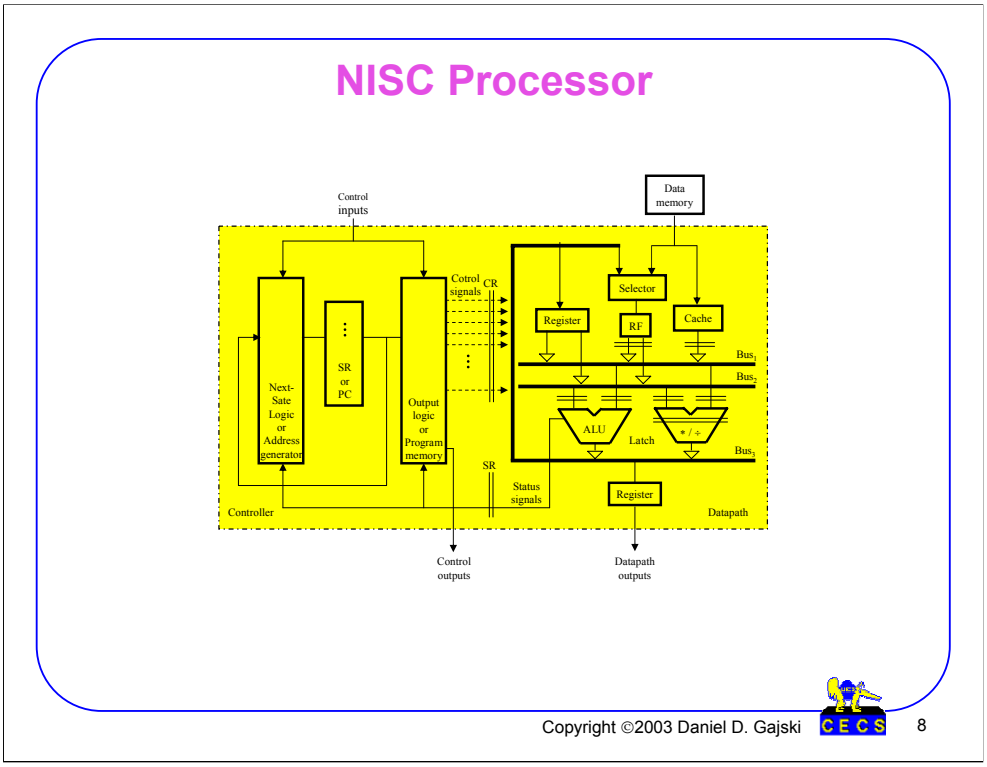
The SR, Next-state and Output logic can be redefined and reconfigured if the Controller is implemented on a FPGA.



NISC Controller (Programmable implementation)

In the programmable version of the NISC Controller the State Register is called a Program Counter (PC) while Output logic is implemented by a Program memory (PM). PM can be writable if we use a RAM or fixed if we use a ROM. The next-state logic is replaced by an Address generator, which computes the next PM address from several sources. It can just increment the present address or generate a new address from a jump address in the PM, from external sources (OS), or from the push-down stack (subroutine return). The proper address is selected by a mixture of output control signals and status signals from the Datapath. Although the Controller shown in the Figure is sufficient for any computation it can be made more sophisticated while implementing the same functionality. For example, the PM can include a cache for speed up if a large PM is required.

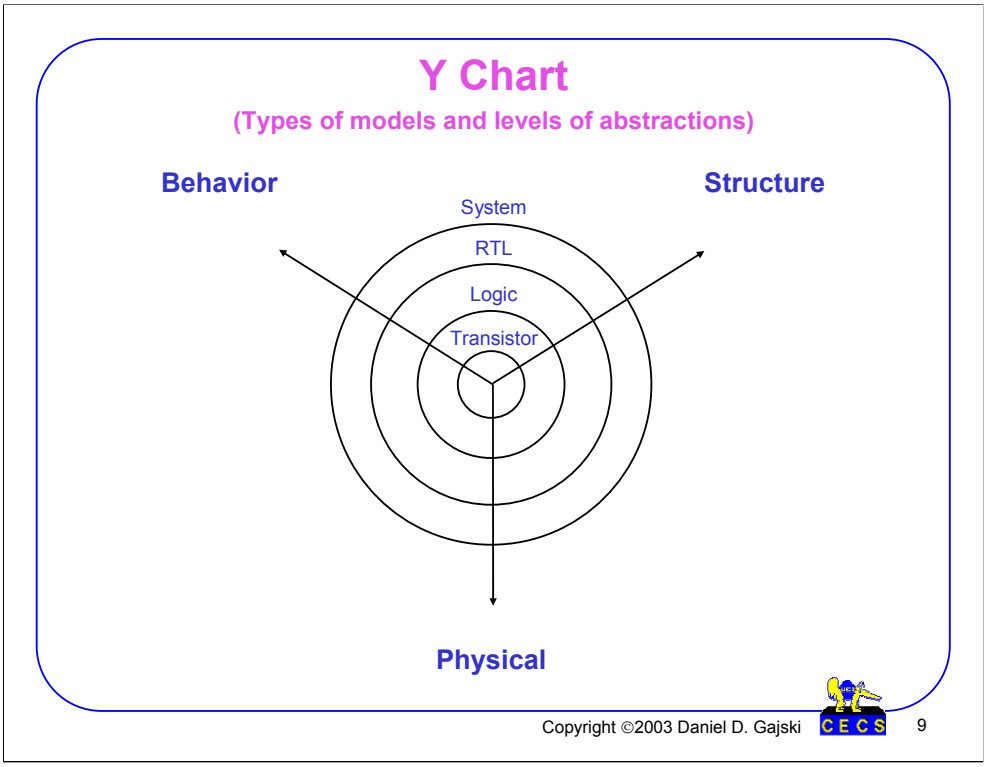
However, the main characteristic of the programmable controller is that new program can be loaded dynamically and executed, thus making NISC programmable like any other processor.



NISC Processor

NISC processor is the combination of Controller and Datapath. Controller can be fixed or programmable. Datapath can be reprogrammable and reconfigurable. Reprogrammable means that Datapath can be extended or reduced by adding or omitting some components, while reconfigurable means that Datapath can be reconnected with the same components. Either way we make changes we must recompile the original code.

In order to speed up the NISC pipelining, we may insert a Control register (CR) and the Status register (SR) between the Controller and the Datapath.



Y-Chart

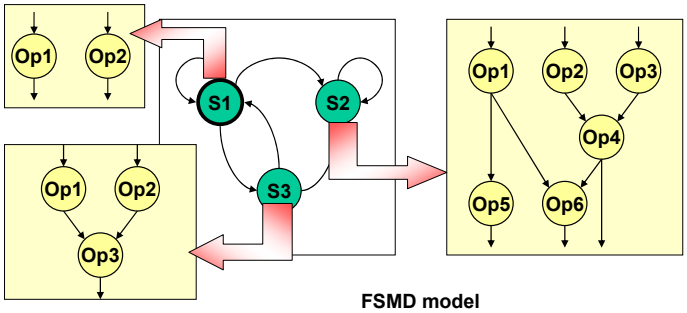
In order to explain the relationship between different abstraction levels, design models and design methodologies or design flows for a NISC processor, we will use Y-Chart. The Y-Chart makes the assumption that each design, no matter how complex, can be modeled in three basic ways emphasizing different properties of the same design.

Therefore, Y-Chart has three axes representing design behavior (function, specification), design structure (connected components, block diagram), and physical design (layout, boards, packages). Behavior represents a design as a black box and describes its outputs in terms of its inputs and time. The black-box behavior does not indicate in anyway how to implement the black box or what its structure is. That is given on the structure axis, where black box is represented as a set of components and connections. Although, behavior of the black box can be derived from its component behaviors such an obtained behavior may be difficult to understand. Physical design adds dimensionality to the structure. It specifies size (height and width) of each component, the position of each component, each port and each connection on the silicon substrate or board or any other container.

Y-Chart can also represent design on different abstraction levels identified by concentric circles around the origin. Usually, four levels are used: Transistor, Logic, Register-transfers and System levels. The name of the abstraction level is derived by the main component used in the structure on this abstraction level. Thus, the main components on Transistor level are N or P-type transistors, while on Logic level they are gates and flip-flops. On the Register-transfers level the main components are registers, register files and functional units such as ALUs, while on the System level they are processors, memories and buses.

RTL Computational Models

- **Finite State Machine with Data (FSMD)**
  - Combined model for control and computation
    - FSM + DFG = FSMD
  - Implementation: controller plus datapath

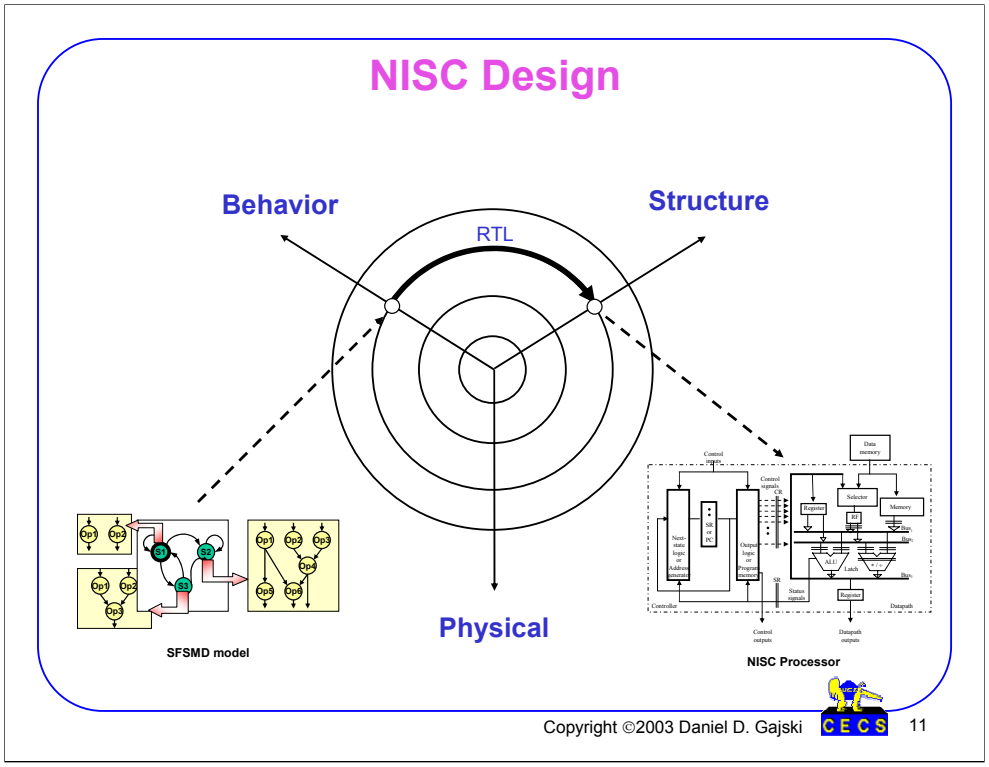


FSMD model

RTL Computational Model

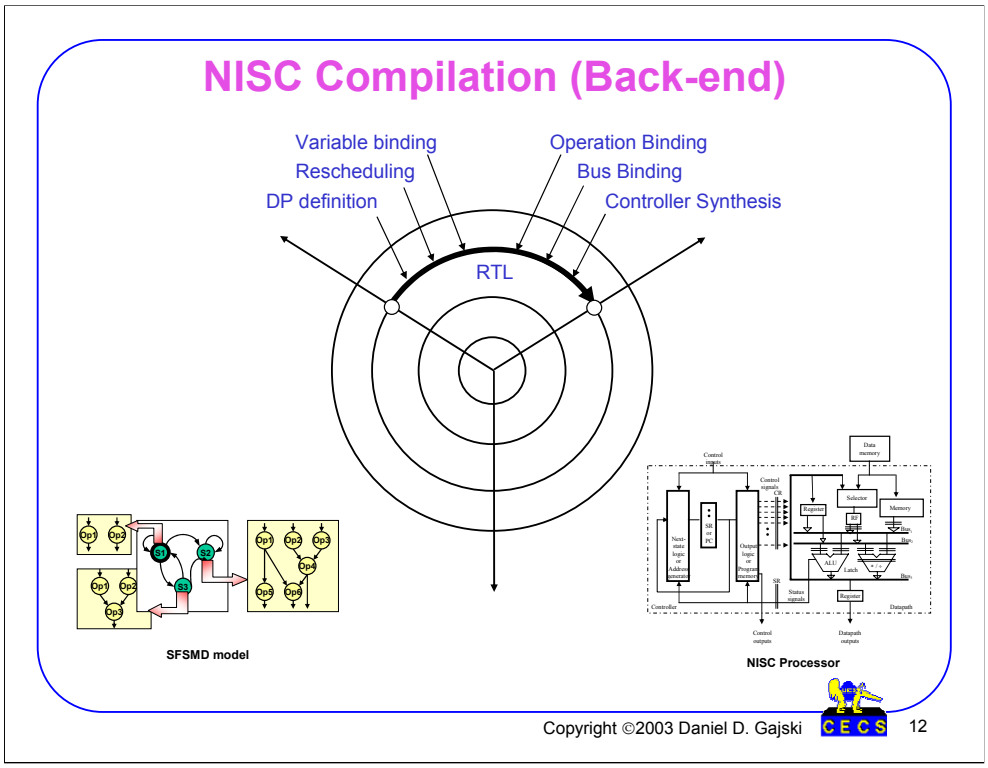
The RTL behavior or computational model is given by a Finite-state-machine with Data (FSMD). It combines finite-state-machine (FSM) model for Controller and data-flow-graph (DFG) for Datapath. FSM has a set of states and a set of transitions from one state into other depending on the value of some of the input signals. In each state FSMD executes a set of expressions represented by a DFG. FSMD model is clock-accurate if each state takes a single clock-cycle.

It should be noted that FSMD model encapsulates the definition of the state-based (Moore-type) FSM in which the output is stable during duration of each state. It also encapsulates the definition of the input-based (Mealy-type) FSM with the following interpretation: Input-based FSM transitions to a new state and outputs data conditionally on the value of some of FSM inputs. Similarly, FSMD executes set of expressions depending on the value of some FSMD inputs. However, if the inputs change just before the clock edge there may be not enough time to execute the expressions associated with that particular state. Therefore, designers should avoid this situation by making sure the input values change only early in the clock period or they must insert a state that waits for the input value change. In this case if the input changes too late in the clock cycle, FSMD will stay in the waiting state and proceed with a normal operation in the next clock cycle.



NISC Design

NISC design starts with the FSMD model in the behavior axes of the Y-chart and ends up with a custom NISC processor containing any number and type of components connected as required by the FSMD model. Note, that FSMD model can be obtained easily from a programming language code such as C by grouping all the consecutive statements into basic blocks (BB) and introducing two states for each **if** statement or **loop** statement , where each state executes a BB. Such a FSMD is sometimes called Super-state FSMD (SFSMD) since each BB may be considered to be executed in one super state. This generation is very simple. Note, that each BB will be partitioned into several states during synthesis, where the number of states depends on the resources allocated to the NISC processor.



NISC Compilation (Backend)

NISC design consists of Datapath definition, generation of control-word sequence, and design of the Controller (fixed or programmable), that executes the functionality specified by the FSMD model. It consists of several tasks;

(a) Definition of the Datapath as a set of components and connections from the RTL library,

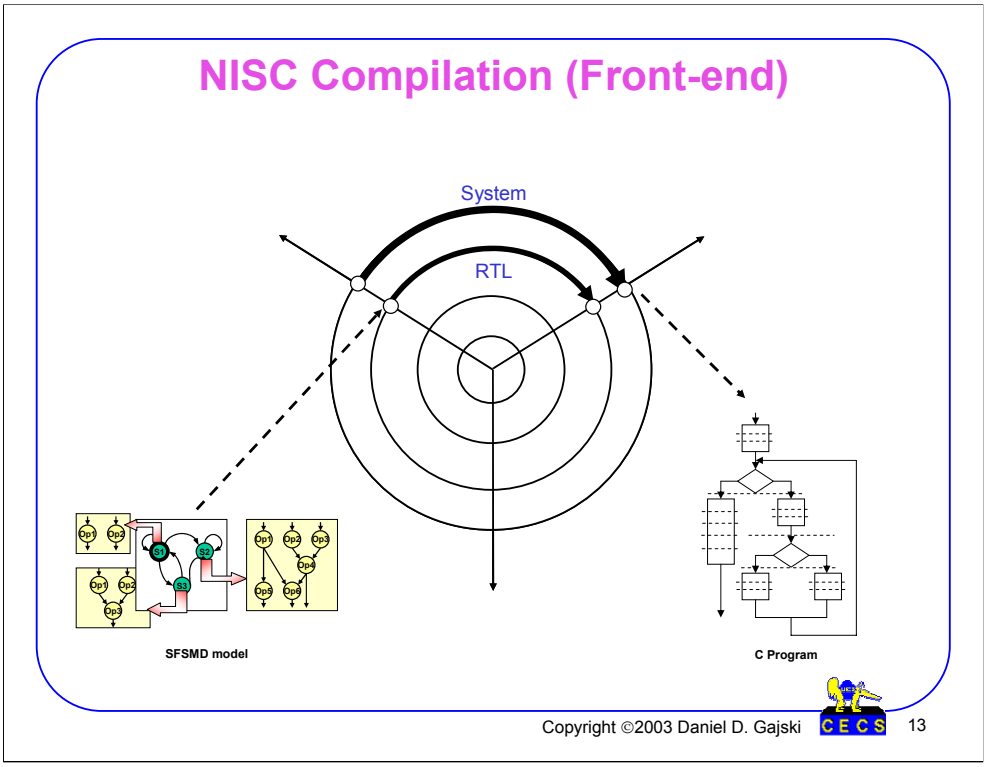
(b) Binding of variables, operations and register transfers to storage elements, functional units and busses,

(c) Rescheduling of computation in each state since some components may need more or less than one clock cycle, and computation must satisfy the Datapath pipelining constraints.

(d) Synthesis of programmable or fixed controller.

(e) Generation of control-word sequence for downloading to the Controller RAM

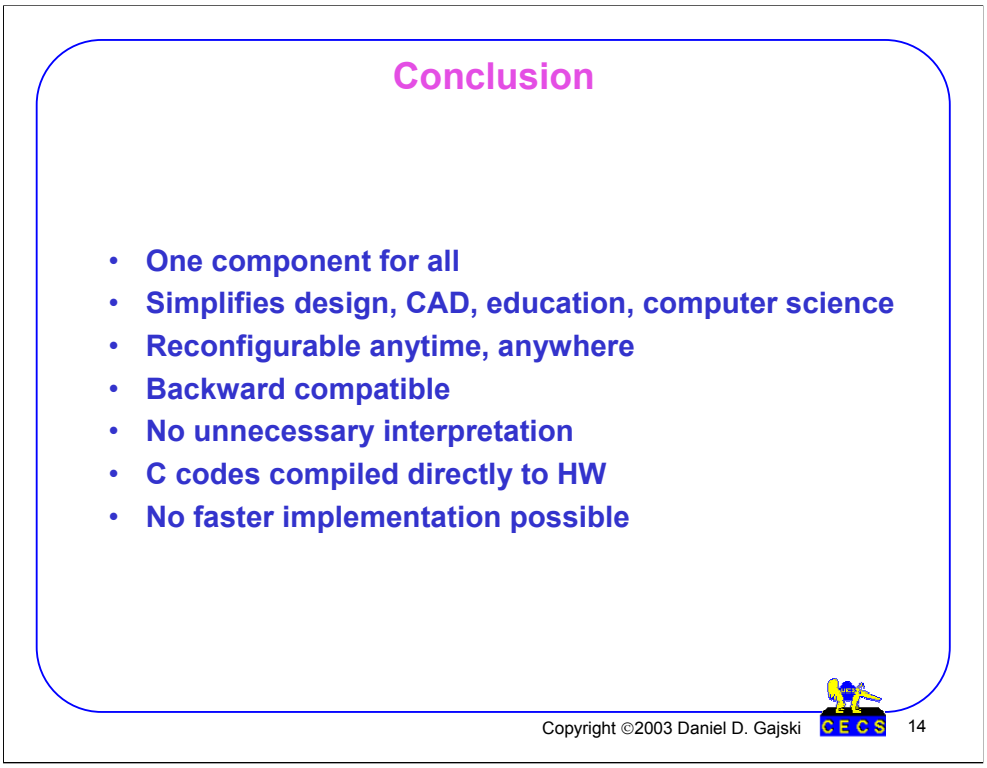
Any of the above tasks can be performed manually or automatically.



NISC Compilation (Front-end)

The NISC processor definition and compilation follows the task of system design, in which the system components and their connectivity as well as partitioning or mapping of system functionality onto different components is performed. Therefore, for each NISC component in the system the C code that executes on that component is known.

NISC compilation consists of parsing that C code and constructing from the parse tree the well known Control-Data Flow Graph (CDFG). CDFG consists of three objects: if statements, loop statements, and basic blocks (BB) of assignment statements without ifs or loops. Each if and loop statement needs two states in the FSMD while BB can be executed in one or more states depending on the availability of resources in the Datapath. Such a CDFG is equivalent to the Super-state FSMD (SFSMD) which is the starting point for the NISC Back-end.



Conclusion

The NISC processor is the single, necessary and sufficient computation component for design of systems-on-chip (memory is the other necessary and sufficient storage component). NISC is a set of components with different Datapaths and Controllers and one compiler.

NISC unifies several concepts from processor architecture, compilers and high-level synthesis into one unique concept. Therefore, it simplifies design, education, CAD, testing, IP trade and other aspects of traditional design.

NISC can be reconfigured and reprogrammed statically and dynamically to satisfy power, performance, cost, reliability and other constraints.

Such programmability allows a NISC to emulate other instruction sets.

Since the instruction set is eliminated the C code compiles directly into hardware. There is no unnecessary interpretation between C code and hardware, that allows a NISC to execute any code as fast as semiconductor technology will allow it. In other words, NISC is the fastest implementation of any computer program