

Hardware /Software Instruction Set Configurability for System-on-Chip Processors

2/7/2002

CS269 Hardware/Software Engineering of Embedded Systems

1

Introduction

- Configurable and extensible processor architectures offer the efficiency of tuned logic solutions with the flexibility of standard high-level programming methodology.
- Designing at the level of software and instructions set architecture significantly shortens the design cycle and reduces verification effort and risk.

2/7/2002

CS269 Hardware/Software Engineering of Embedded Systems

2

Why Configure Processors?

- Continuing growth in silicon chip capability is reducing the number of chips in a typical system, and magnifying the size, performance and power benefits of system-on-chip integration.
- Newer electronic products demand better cost, bandwidth, battery life, and software functionality.
- Application-specific processor cores promise a combination of full software flexibility with high efficiency.

2/7/2002

CS269 Hardware/Software Engineering of Embedded Systems

3

What's the Right Architecture?

- A system with a processor and a coprocessor can have its performance improved by integrating the coprocessor.
 - System architect can discard external control logic: the finite state machines and micro-sequencers.
 - Faster instruction set extension and performance testing allows for rapid prototype validation and experimentation.
 - Simplifies the use of data memory, since the processor can share a unified data memory.

2/7/2002

CS269 Hardware/Software Engineering of Embedded Systems

4

Extension Description Method

- Instruction format and encoding.
 - Ofield* : gives a name to a group of bits in the instruction word.
 - Oopcode* : assigns instruction fields with values.
 - Ooperand* : specifies how instruction operand is encoded in an instruction field.
 - Oiclass* : defines the assembly format for an instruction.

2/7/2002

CS269 Hardware/Software Engineering of Embedded Systems

5

- Example:

```
/*Define new opcodes*/
opcode A4 op2=0 CUST0
opcode S4 op2=1 CUST0
/*Define new assembly class*/
iclass RR{A4,S4}{out arr, in ars, in art}
/* Define new assembly instructions*/
A4 arr, ars, art
S4 arr, ars, art
```

2/7/2002

CS269 Hardware/Software Engineering of Embedded Systems

6

- Customized Datapath

- Computation part of an instruction is specified in a TIE *reference* block, which contains a series of assignment statements.
- Syntax is very close to Verilog.

```
reference A4 {
  assign arr = {
    arr[31:20] + art[31:24],
    arr[23:16] + art[23:16],
    arr[15:8] + art[15:8],
    arr[7:0] + art[7:0]
  }
}
reference S4 {
  assign arr = {
    arr[31:20] - art[31:24],
    arr[23:16] - art[23:16],
    arr[15:8] - art[15:8],
    arr[7:0] - art[7:0]
  }
}
```

- A semantic statement can be used to optimize the code for hardware.

```
semantic addsub {A4, S4} {
  assign arr = {
    ars[31:24] + (S4?-art[31:24],art[31:24]) + S4,
    ars[23:16] + (S4?-art[23:16],art[23:16]) + S4,
    ars[15:8] + (S4?-art[15:8],art[15:8]) + S4,
    ars[7:0] + (S4?-art[7:0],art[7:0]) + S4
  }
}
```

- Multi-cycle instructions

OTIE provides a simple *schedule* construct to capture the multi-cycle instructions requirements.

○ Example:

```
acc = acc + a * b;
schedule MAC_SCHED {MAC} {
  use a 1; use b 1; use acc 2; def acc 2;
}
```

- TIE states allow instructions to have many more sources and destinations than they are provided by the read and write ports of the core register files.
- They can be as analogous to the processor status registers or states in FSM.
- States can hold values for temporary variables during program execution.

```
state acc 40 /* a 40-bit accumulator */
opcode MAC op2=0 CUSTO
iclass RS{MAC}{in ars, in art}{inout acc}
reference MAC {
  assign acc = ars * art + acc;
}
```

• Without using the accumulator state, the MAC instruction would either take up three readports and one write port of the register file in every clock cycle, or would have to take three cycles to finish the operations if the register file only had two read ports.

