Microprocessor interfaces

Lecture 10 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2017-18

Table of Contents

1. Microprocessor interfaces
2. lecture topics
3. memory-mapped interfaces
4. coprocessor interfaces
5. custom instruction interfaces
6. ASIP design flow
7. example: the Nios-II custom-instruction interface
8. register files for Nios-II custom instructions
9. references

lecture topics

outline:

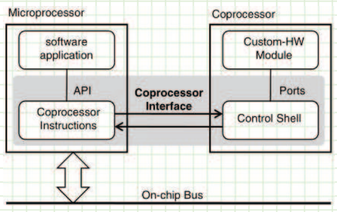
- memory-mapped interfaces
  - memory-mapped register
  - mailbox with handshake
  - FIFO queue
  - handshake protocols
  - shared memory
- coprocessor interfaces
- ASIP design flow
- custom-instruction interfaces

  example: the Nios-II custom-instruction interface

memory-mapped interfaces

seminar by Grazia Pagano (PDF, in Italian)

coprocessor interfaces



Schaumont, Figure 11.7 - Coprocessor interface

when high data-throughput between the software and the custom hardware is needed, a dedicated processor interface outperforms memory-mapped interfaces

a coprocessor interface does not make use of the on-chip bus, it uses a dedicated port on the processor, driven by coprocessor instructions

both the coprocessor instruction set and the specific coprocessor interface depend on the type of processor—not all processors have a coprocessor interface

the decision of using a specific coprocessor interface locks the custom hardware module into a particular processor, thus it limits the reusability of that custom hardware module to systems that also use the same processor

main advantages of a coprocessor interface over an on-chip bus:

- *higher throughput*: because not constrained by the wordlength of the bus on chip, nor by the load/store transfer mechanism
- *fixed latency*: a coprocessor bus is a dedicated, point-to-point connection with stable, predictable timing

custom instruction interfaces

the integration of hardware and software can be considerably accelerated as follows:

1. reserve a portion of the opcodes from a microprocessor for new instructions
2. integrate the custom-hardware modules directly into the micro-architecture of the micro-processor
3. control the custom-hardware modules using new instructions derived from the reserved opcodes

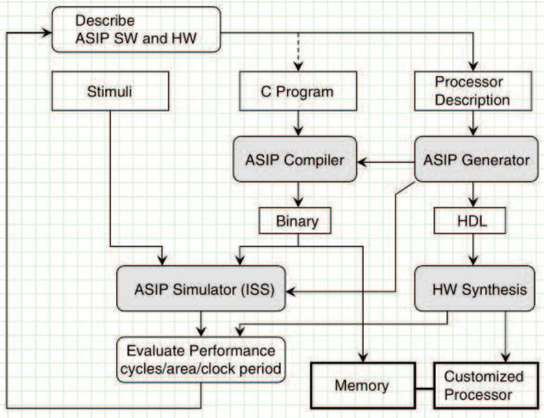
the resulting design is called an *Application-Specific Instruction-set Processor* (ASIP)

while a coprocessor instruction set is part of a microprocessor, an ASIP instruction set is defined by the application

ASIP design automates some of the more difficult aspects of HW/SW codesign:

- the instruction-fetch and dispatch mechanism in the micro-processor ensures that custom-hardware and software remain synchronized
- design of an ASIP proceeds in an incremental fashion, that helps the designer to avoid drastic changes to the system architecture while exploring different options

ASIP design flow



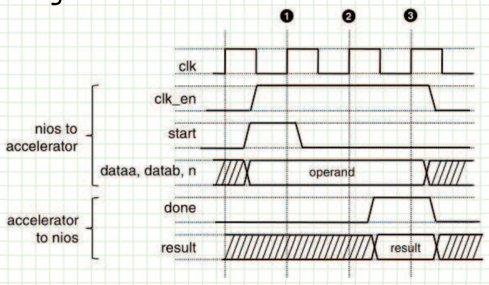
Schaumont, Figure 11.12 - ASIP design flow

sequential ASIP design does not generally deliver better performance than SoC design based on custom hardware modules, yet it does deliver less error-prone results

significant progress has been made on design tools that support the ASIP design flow—all of the shaded boxes in the figure can be obtained as commercial tools

example: the Nios-II custom-instruction interface

the Nios-II softcore processor has a coprocessor interface whereby custom instructions may be defined and hardware modules may be attached to



Schaumont, Figure 11.15 - Nios-II custom-instruction interface timing

the interface supports variable-length execution of custom instructions through a two-way handshake

the clk_en input is used to mask off the clock to the custom hardware when the instruction is inactive

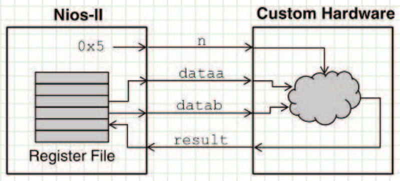
the software instruction custom commands its execution, e.g.:

```
custom 0x5, r2, r3, r5
```

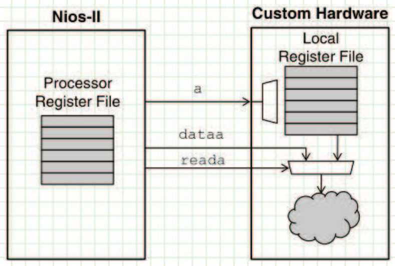
assigns the value 0x5 to n and associates the dataa, datab, result ports with registers r2, r3, r5, respectively—use of n: to multiplex different custom instructions in the hardware module

register files for Nios-II custom instructions

the use of a local register file in the custom hardware module is also supported



Schaumont, Figure 11.16a - Nios-II custom-instruction integration with processor register file



Schaumont, Figure 11.16b - Nios-II custom-instruction integration with local register file

a custom instruction may take operands from either register file: registers prefixed with r are located in the processor, while registers prefixed with c are located in the custom hardware

instructions that use both are allowed, such as custom 0x5, c2, c3, r5

figure 11.16b shows the case for the first input operand only: the control signal reada selects either the processor's or the local register file

- in the former case, the operand is provided through the dataa port, that is associated with a processor's register
- in the latter case, the input a selects the local register to use as operand

references

recommended readings:

Schaumont, Ch. 11, Sect. 11.1.1-11.1.5, 11.2.0, 11.3.0-11.3.1, 11.3.3

for further consultation:

Schaumont, Ch. 11, Sect. 11.1.6, 11.2.1-11.2.2, 11.3.2, 11.3.4