Hardware interfaces

Lecture 11 on Dedicated systems

Teacher: Giuseppe Scollo

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

 Copyleft © 2018 Giuseppe Scollo

Table of Contents

 Copyleft © 2018 Giuseppe Scollo

lecture topics

outline:

- the coprocessor hardware interface
- typical functions of hardware interfaces
- layout of a coprocessor hardware interface
- data addressing
- multiplexing
- masking
- control design
- hierarchical control
- programmer's model
  - address map
  - instruction set
- example: a hardware acceleration case
  - design decisions
  - Avalon interface and programming model

Copyleft © 2018 Giuseppe Scollo

functions, layout and design of hardware interfaces

seminar by Salvatore Mameli (PDF, in Italian)

Copyleft © 2018 Giuseppe Scollo

programmer's model

programmer's model = control design + data design

the *programmer's model*, that is the software view of a hardware module, includes:

- a collection of the memory locations used by the custom hardware module, and
- a definition of the commands (or instructions) understood by the module

a few considerations follow about the impact that these two kinds of design decisions have on the design of the software driver of the custom hardware module

Copyleft © 2018 Giuseppe Scollo

address map

the *address map* reflects the organization of software-readable and software-writable storage elements of the hardware module; its design should consider the viewpoint of the software designer rather than the hardware designer, thus:

- a given memory-mapped address should always affect the same hardware register, regardless of whether the operation on it is a read or a write
- by default all memory-mapped registers should be read/write; in some cases, read-only registers are justified, such as for example to implement registers that reflect hardware status information or sampled-data signals; however, there are very few cases that justify a write-only register
- the address map should respect the processor's alignment; e.g., extracting bits 5–12 out of a 32-bit word is more complicated than extracting the second byte of the same word

Copyleft © 2018 Giuseppe Scollo

instruction set

the design of a good instruction set is a hard problem, that requires the codesigner to make a proper trade-off between flexibility and efficiency

it strongly depends on the function of the custom-hardware module

here are a few generic design guidelines:

- one can distinguish three classes of instructions: one-time commands, on-off commands, and configurations; their mix affects the general behavior of the hardware module, it should be aimed at minimizing the control interaction between the software driver and the hardware module
- design the synchronization between software and hardware at multiple levels of abstraction, that is, not just at the data transfer level but also at the algorithmic level
- another synchronization problem occurs when multiple software users share a single hardware module; this may be solved either by serializing coprocessor usage or by implementing a context switch in the hardware module
- finally, reset design must be carefully considered; an example of flawed reset design is when a hardware module can only be initialized by means of full system reset—it makes sense to define one or several instructions for the hardware module to handle module initialization and reset

Copyleft © 2018 Giuseppe Scollo

example: design decisions for a hardware acceleration case

a recent lab tutorial presented a software implementation of the delay computation of a Collatz trajectory with given start point

hardware implementations of the same function were the subject of previous lab experiences

e.g. the third lab experience produces a VHDL description of it

the performance measurements carried out on the software implementation show that it consumes almost all of the program execution time

problem: accelerate the program execution by using the hardware implementation of the aforementioned function

a first alternative to evaluate: to integrate the hardware function as a custom instruction or as a memory-mapped coprocessor?

the second option seems better, for at least two reasons:

- the first option is *blocking*
- the data transfer size in each interaction is very small

other design decisions depend on this first decision, as follows

Copyleft © 2018 Giuseppe Scollo

Avalon interface and programming model for the sample case

the VHDL description of the circuit which computes the function is to be embedded into a component equipped with Avalon interfaces for the Clock, Reset, and Avalon MM Slave signals, so as to receive the initial data by a write operation and to return the result by a reply to a read operation

- *multicycle* data transfers are possible thanks to the Avalon signal `waitrequest`, set by the slave to defer the response to a read or write request by an arbitrary number of cycles

addressing of the coprocessor: since the (initial data) write and (final result) read operations take place at different times and have the same data size, a single address suffices

- for the sake of simplicity, it is convenient to use the 32-bit Avalon signals `writedata`, `readdata` in the hardware interface for this address, with internal conversion to 16-bit for the corresponding internal I/O ports of the circuit which computes the function

software driver: two macros and a function may be defined for the bus access software interface: `DC_RESET(d)`, `DC_START(d,x0)`, `unsigned int delay(d)`, where `d` is the address assigned to the coprocessor

- these project ideas will be developped in the next lab tutorial

Copyleft © 2018 Giuseppe Scollo

references

recommended readings:

- Schaumont, Ch. 12, Sect. 12.1-12.3.1, 12.4

for further consultation:

- Schaumont, Ch. 12, Sect. 12.3.2
- Avalon® Interface Specifications, Ch. 1-3, MNL-AVABUSREF, Intel Corp., 2017.05.08

Copyleft © 2018 Giuseppe Scollo