

Architectures and design process of dedicated systems

Lecture 02 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2016-17

1 di 14

Table of Contents

1. Architectures and design process of dedicated systems
2. lecture topics
3. hardware vs software design paradigms
4. software program development
5. software program analysis
6. codesign models
7. example: functions on Collatz trajectories
8. Collatz delay datapath, v. 1
9. a Collatz delay codesign model
10. Collatz delay datapath, v. 2
11. concurrency and parallelism
12. example: parallel addition
13. references

2 di 14

outline:

- dualism of hardware design vs software design paradigms
- software program development and analysis
- codesign models
- example: a Collatz delay component for codesign
- concurrency and parallelism
- proposed problems (in the reserved area)

hardware vs software design paradigms

key professional challenge in hardware-software codesign:

capability to combine two radically different design paradigms

hardware and software are the dual of one another in many respects

here is a comparative synopsis of their fundamental differences (Schaumont, Table 1.1)

	Hardware	Software
Design Paradigm	Decomposition in space	Decomposition in time
Resource cost	Area (# of gates)	Time (# of instructions)
Flexibility	Must be designed-in	Implicit
Parallelism	Implicit	Must be designed-in
Modeling	Model \neq implementation	Model \sim implementation
Reuse	Uncommon	Common

software models are programs, therefore implementations, yet they are developed at very different abstraction levels

```

1  int max;
2
3  int findmax(int a[10]) {
4      unsigned i;
5      max = a[0];
6      for (i=1; i<10; i++)
7          if (a[i] > max) max = a[i];
8  }

```

Schaumont, Listing 1.1 - C example

this fairly high-level model gives little information about the cost of its execution, in terms of machine instructions

to this purpose one needs to analyse a low-level model, such as the assembly program produced by a compiler

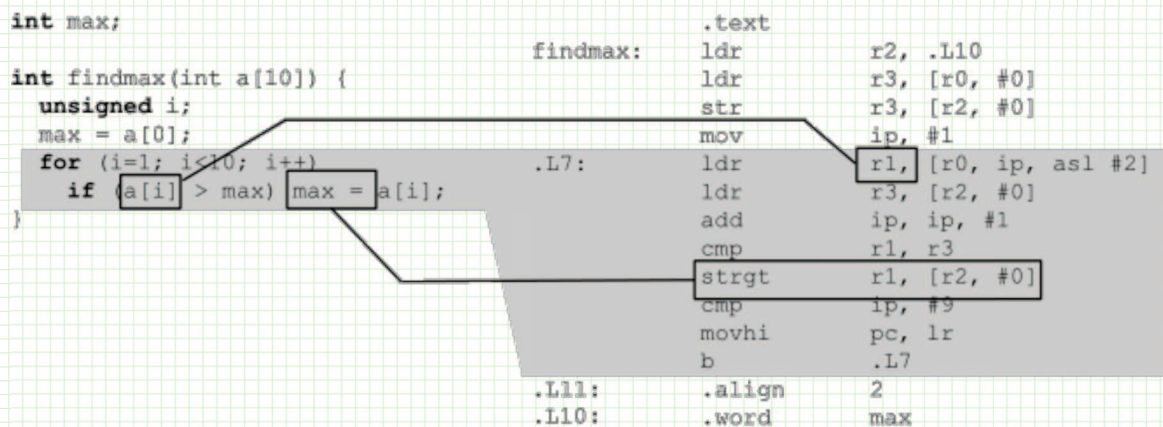
```

        .text
findmax:
        ldr    r2, .L10
        ldr    r3, [r0, #0]
        str    r3, [r2, #0]
        mov    ip, #1
.L7:
        ldr    r1, [r0, ip, asl #2]
        ldr    r3, [r2, #0]
        add    ip, ip, #1
        cmp    r1, r3
        strgt   r1, [r2, #0]
        cmp    ip, #9
        movhi  pc, lr
        b      .L7
.L11:
        .align 2
.L10:
        .word  max

```

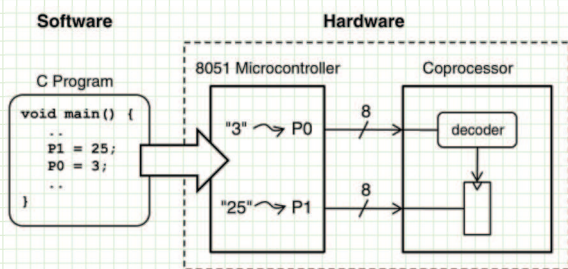
Schaumont, Listing 1.2 - ARM assembly example

even if one is not familiar with all details of a particular assembly language, finding the correspondence between high-level source instructions and assembly instructions is not so difficult as it may seem



Schaumont, Fig. 1.2 (edited) - Mapping C to assembly

a simple example highlights the variety of models which come into play in hardware-software codesign:



Schaumont, Fig. 1.3 - A codesign model

- software models: the C program, its microprocessor machine-language executable
- hardware models: microprocessor, coprocessor, hardware interface between them
- a model of the hardware-software interface: which instructions determine which interactions between microprocessor and coprocessor

the details of the formalization of this example in Gezel are deferred to a later lab tutorial

example: functions on Collatz trajectories

the hardware datapath presented in the first lecture could hardly serve as a coprocessor to accelerate the visualization of a Collatz trajectory

why?

however, it may be embedded in a coprocessor that is designed to accelerate the computation of functions on a Collatz trajectory

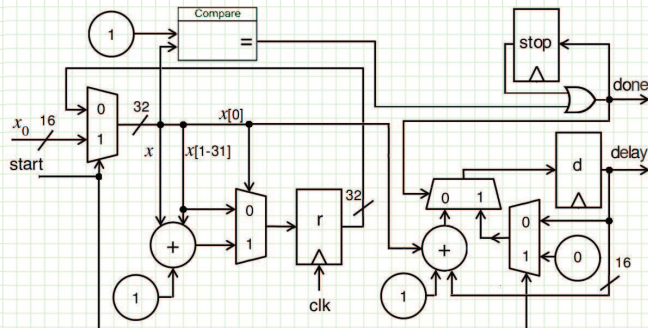
for example: the *delay* of the trajectory, its (highest) peak value, etc.

to this purpose a redefinition of the circuit interface is needed, as well as its extension with some control logic, e.g. to stop the computation and output the result upon the first '1' occurrence in the trajectory

N.B. with respect to the Collatz delay, take it into account that:

- the delay grows by 2 at every iteration from an odd value
- '1' is a legal initial value, in which case the delay is 0

an extension of the circuit seen in the first lecture that does not output the trajectory, rather its *delay*.



Hardware datapath for the *delay* of a Collatz trajectory

Gezel representation:

```
dp delay_collatz (
  in start : ns(1) ; in x0 : ns(16) ;
  out done : ns(1) ; out delay : ns(16))
{
  reg r : ns(32) ;
  reg d : ns(16) ;
  reg stop : ns(1) ;
  sig x : ns(32) ;
  always { x = start ? x0 : r ;
    r = x[0] ? x + (x >> 1) + 1 : x >> 1 ;
    done = ( x == 1 ) | stop ;
    stop = done ;
    d = done ? ( start ? 0 : d ) : d + 1 + x[0] ;
    delay = d ;
  } }
```

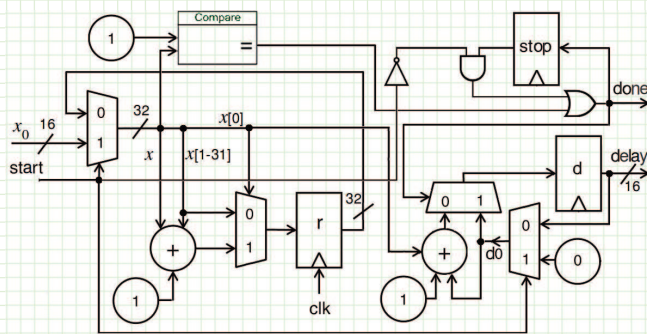
a Collatz delay codesign model

the interface of the datapath just seen suggests an easy implementation of the coprocessor as a memory-mapped I/O device, for example equipped with:

- control register, including the *start* bit
- state register, including the *done* bit
- a bidirectional data register, which may be made use of both for the initial input and for the output of the result

but... is the aforementioned datapath adequate to perform the required computation for subsequent interactions with the software?

revised circuit for the delay of Collatz trajectories:



Hardware datapath for the delay of Collatz trajectories

Gezel representation:

```

dp delay_collatz_rev (
  in start : ns(1) ; in x0 : ns(16) ;
  out done : ns(1) ; out delay : ns(16))
{
  reg r : ns(32) ;
  reg d : ns(16) ;
  reg stop : ns(1) ;
  sig x : ns(32) ;
  sig d0, dd : ns(16) ;
  always { x = start ? x0 : r ;
    r = x[0] ? x + (x >> 1) + 1 : x >> 1 ;
    done = ( x == 1 ) | ( stop & ~start ) ;
    stop = done ;
    dd = 1 + x[0] ;
    d0 = start ? 0 : d ;
    d = done ? d0 : d0 + dd ;
    delay = d ;
  }
}

```

concurrency and parallelism

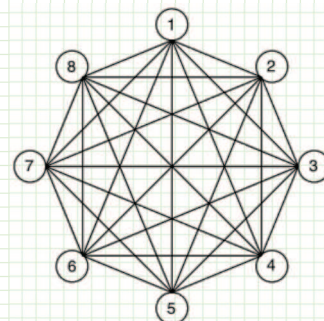
concurrency and parallelism are not synonyms:

- concurrent processes: mutual independence of their computations
- parallel processes: simultaneity of their executions on different processors or circuits

concurrency is a feature of the application,
parallelism is a feature of its implementation, that requires:

- concurrency in the application, and
 - a parallel hardware architecture
- e.g. the Connection Machine (CM), see figure

Amdahl's law sets at $1/s$ the maximum speed-up that may be achieved by parallel execution of an application that has a fraction s of sequential execution



Schaumont, Fig. 1.9 - Eight node connection machine

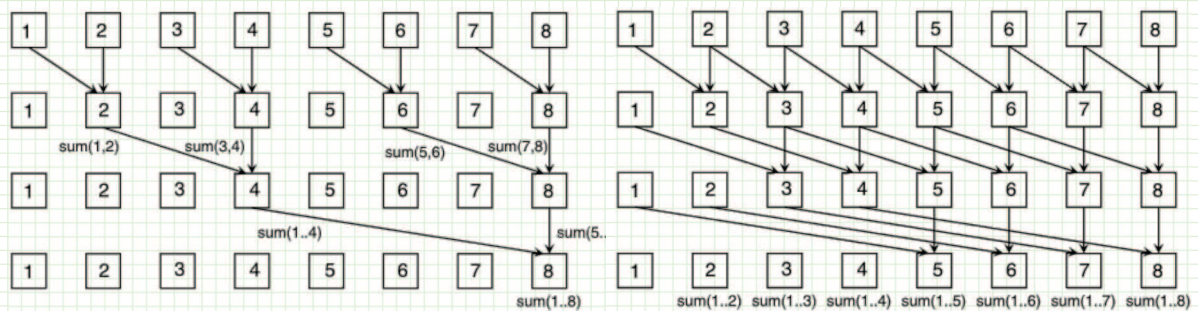
example: parallel addition

is it difficult to devise concurrent algorithms for parallel architectures?

not necessarily, it depends on programming education and habits

for example, consider the sum of n numbers on the CM, say with $n = 8$, by assigning one of the summands to each processor initially

the algorithms illustrated next compute the sum in $\lceil \log_2(n) \rceil$ steps



Schaumont, Fig. 1.10 - Parallel sum

Schaumont, Fig. 1.11 - Parallel partial sum

DMI - Graduate Course in Computer Science

Copyright © 2016-2017 Giuseppe Scillo

13 di 14

references

recommended readings:

Schaumont (2012) Cap. 1, Sez. 1.1.2-1.1.3, 1.5, 1.7

for further consultation:

F. Vahid & T. Givargis (2002) Cap. 1, Sez. 1.5-1.6

C. Brandolese, W. Fornaciari (2007) Cap. 1

DMI - Graduate Course in Computer Science

Copyright © 2016-2017 Giuseppe Scillo

14 di 14