Introduction to Synchronous Programming in Control

Paul Caspi

Verimag (CNRS)

A historical perspective based on the observation of several real-world systems during the Crisys Esprit project:

- The Airbus “fly-by-wire” system.
- Schneider’s safety control and monitoring systems for nuclear plants.
- Siemens’ letter sorting machine control,

and many other distributed safety-critical control systems.

Overview

- Basic needs of the domain
- Real-time asynchronous languages
- Synchronous practices
- The formalisation of these practices

Basic Needs of the Domain

- Parallelism:
  - between the controller and the controlled device
  - between the several degrees of freedom to be controlled at the same time
- Guaranteed bounds :
  - on memory
  - on execution times
- Distribution

The Computer Science Answer:
Real-Time Kernels and Languages

Based on the concurrency tradition of operating systems:

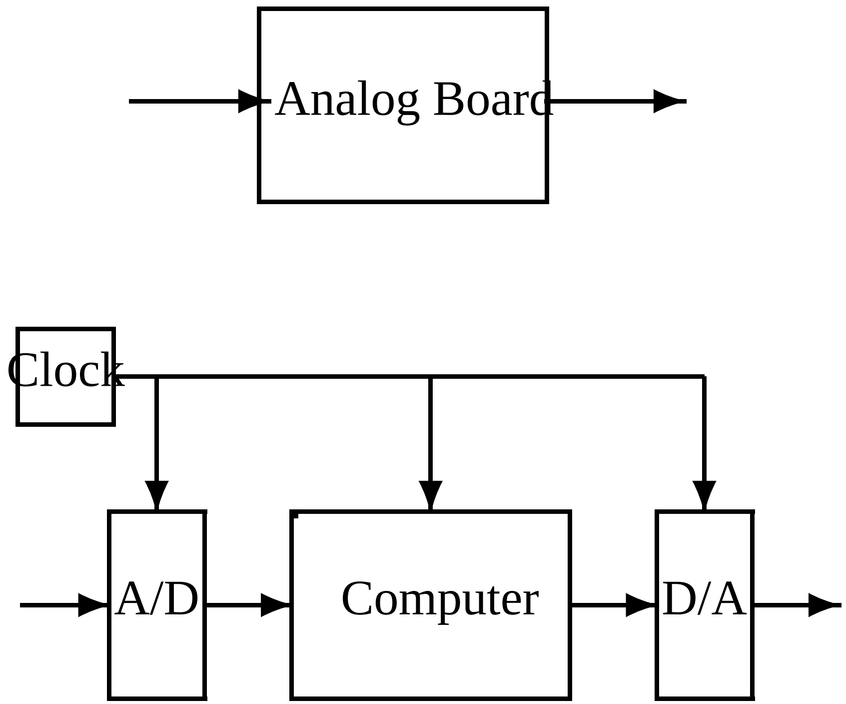
- Synchronisation: semaphores, monitors, sequential processes,
- Communication: shared memory, messages,
- Synchronisation + communication: queues, rendez-vous.

Examples:

- CSP, OCCAM,
- ADA tasking
- real-time OS

The Evolution of Practices

From analog boards to computers:



periodic clocks

synchronous programs

Periodic Synchronous Programming

```
initialize state;

loop each clock tick

   read other inputs;
   compute outputs and state;
   emit outputs

 end loop
```

Practical Interest

- Perfectly matches:
  - the need for real-time integration of differential equations: forward, fixed step methods,
  - the mathematical theory of sampled control systems,
  - the theory of switching systems.
- Safety, simplicity and efficiency:
  - almost no OS, a single interrupt (the real-time clock), no context saving (the interrupt should occur at idle time)
  - bounded memory, bounded execution time.

$\Rightarrow$ Easier validation, certification

Generalisation: Synchronous Languages

```
initialize state;
loop each input event
   read other inputs;
   compute outputs and state;
   emit outputs
 end loop
```

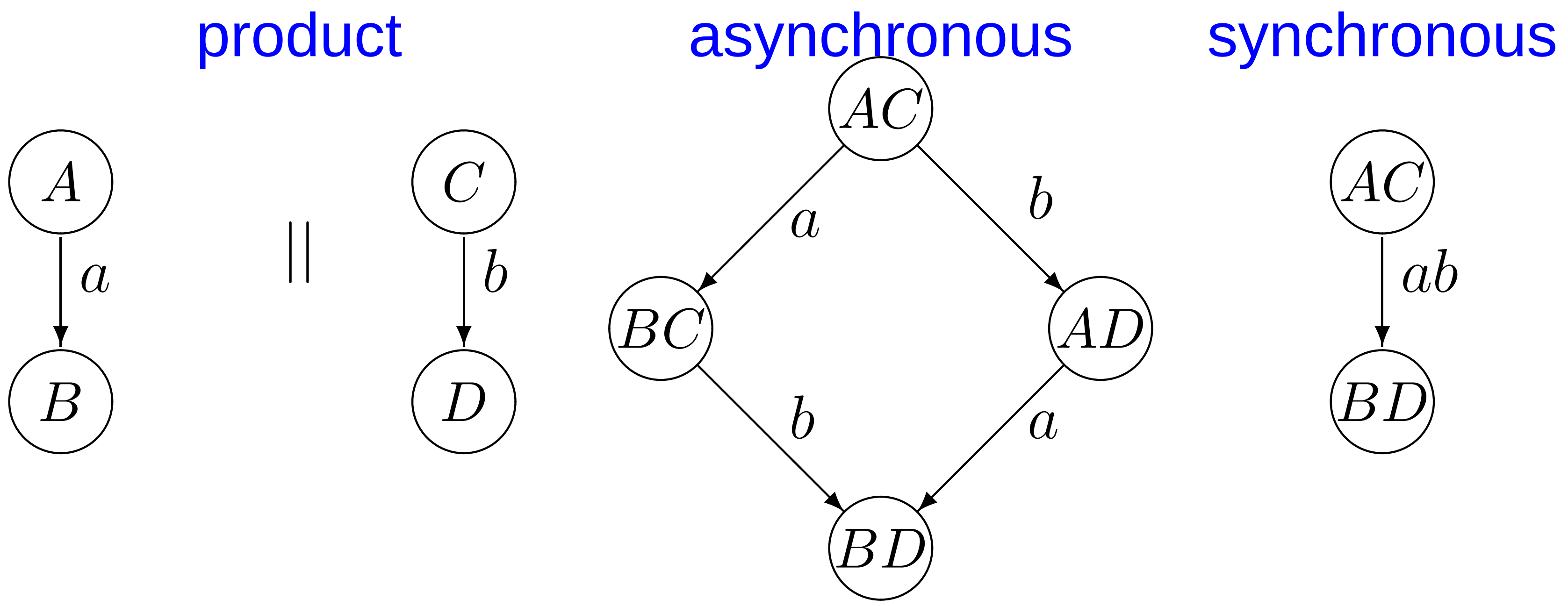
Several styles (imperative, data-flow,...)

Compiled parallelism (instead of concurrent

most applications of synchronous programming are actually periodic ones.

Theory: SCCS (Milner)

Based on the synchronous product of automata:



CCS (asynchronous) is a sub-theory of SCCS

Provides a theoretical justification of practice: Synchronous primitives are stronger, programming is easier

Further Justifications (Berry)

- No added non determinism:
  - easier debugging and test
  - less state explosion in formal verification
- Easier temporal reasoning:
  - synchronous steps provide a “natural” notion of logical time: in a concurrency framework `delay 5 seconds` means “a least 5 seconds” and is priority dependent.
  - Easier roll-back and recovery

Conclusion 1:

These advantages seem conclusive and justify the practices.

But . . .