Combining HDLs

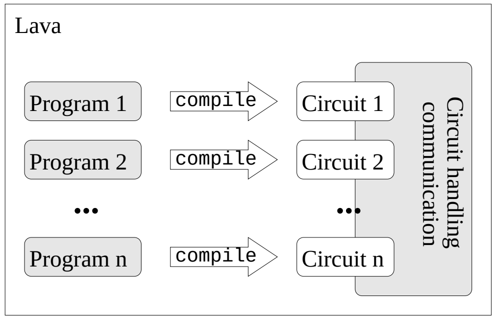
Use different languages to solve different parts of the problem.

- Easier to express
- Less bugs
- Express properties and circuit together

... but ...

- Different language constructs
- Different encodings/protocols

Combining languages explicitly



Intensive one week course in

Hardware Description Languages for Synchronous Hardware Design

Combining HDLs

Gordon J. Pace

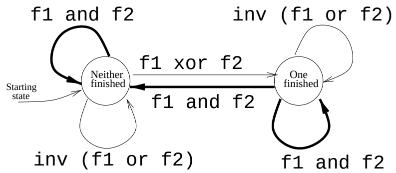
September 2000

Combining languages explicitly: An example

The synchroniser example revisited:

```
(f_p, emit_p) = compileFlash start p
(f_q, emit_q) = compileFlash start q
emit          = or2 (emit_p, emit_q)
f             = compileFSM fsm f_p f_q
```

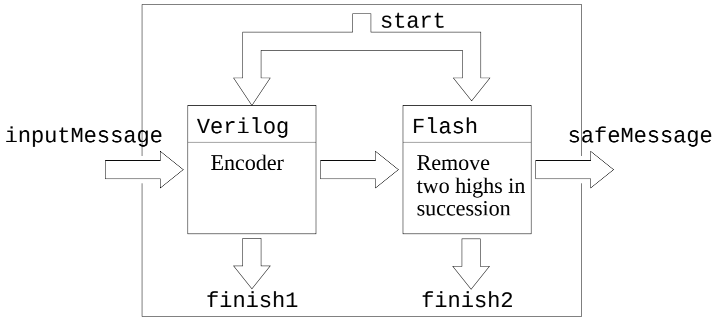
```
fsm f1 f2     = (
```




```
)
```

Combining languages explicitly: Another example (1)

Sending an encoded message without burning the sensitive output wire:



Combining languages explicitly: Another example (2)

```
safeEncoder inMessage = safeMessage
  where
    start               = delay high low
    (_, encodedMessage) = compileVerilog start verilogEncoder
    (_, safeMessage   ) = compileFlash start
                              ( forever
                                  ( IfThenElse (encodedMessage)
                                      ( emit1 :>>  Delay
                                      , Delay
                                      )
                                  )
                              )
```


Combining circuits explicitly

- Need common language to interface.
- Need to design (low-level) interface.
- Needs explicit translation of data between different languages.

Combining circuits implicitly: An example

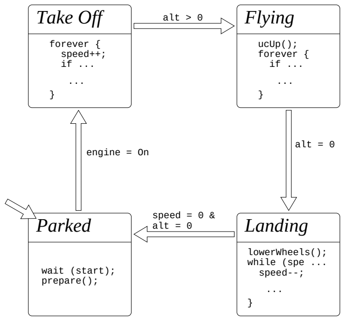
Circuits can be included as parts of other programs.

Example: Write a program which repeatedly executes non-deterministically one of the Flash programs `p` and `q`.

$$\begin{array}{lll} \texttt{cspProgram =} & & \mu X.\ (req?\ \texttt{useFlash(p)}.\ ack!\ X)+ \\ & & \quad (req?\ \texttt{useFlash(q)}.\ ack!\ X) \\ & \mid & \mu Y.\ req!\ ack?\ Y \end{array}$$

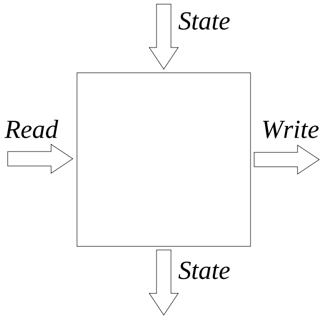
where `useFlash = useCircuit.compileFlash`.

Combining circuits implicitly: Another example

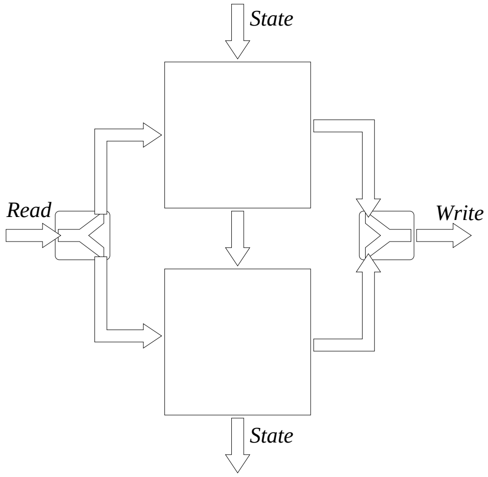


Combining circuits implicitly: The problem

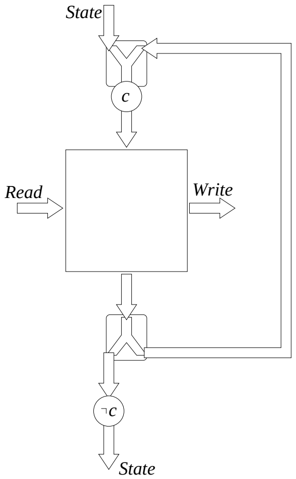
- We need a common circuit shape.
- Shape depends on the language features we intend to use.



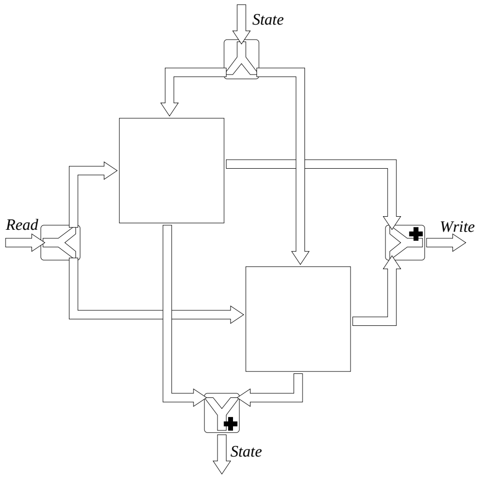
General sequential composition



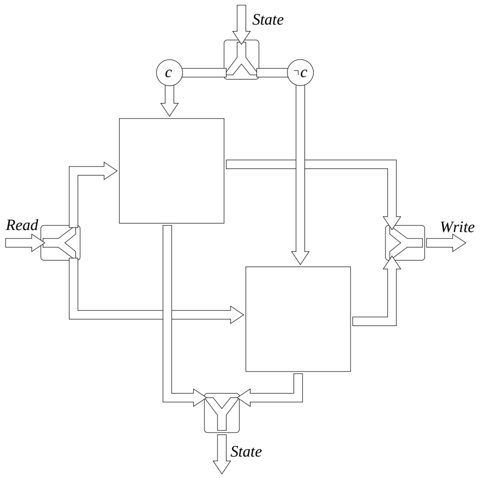
General iteration



General fork-join



General conditional



Circuit combinators

- The types and the Y-components depend on the data-flow path of the circuits.
- The `start` and `finish` signal are now simply part of a more general design.
- Rather than talking about *languages* we should be talking about *language features*.

Conclusions

- Different languages have different strengths and weaknesses.
- Combining languages enables more effective descriptions of hardware, and ...
- more efficient compiled circuits.