Synchronous Programs

Additional examples

Esterel v7

Esterel v7 is a major evolution of the previous version Esterel v5 which makes it possible to design much richer systems and in particular hardware and software systems that combine data path and control path features.

There are two absolute constraints in the Esterel v7 language design: keeping the semantics as rigorous as in all the previous versions of Esterel, and keeping all programs synthesizable in hardware or software.

The main new features are the following ones:

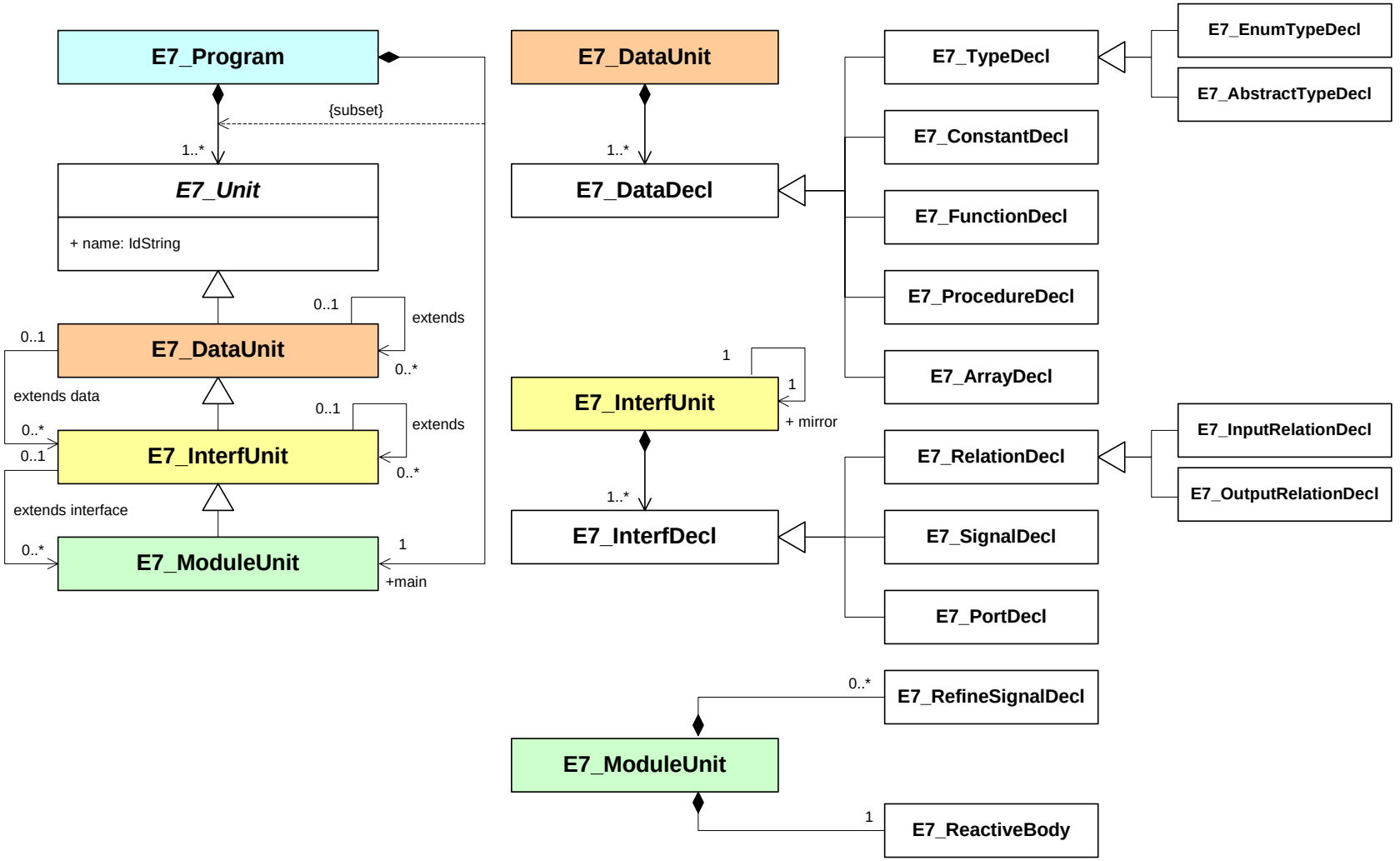
Esterel v7 (continued)

- Separation of data, interface, and module units for better program structuring and reuse.
- Data genericity of interface and module units.
- Infinite precision exact arithmetic, dealing with abstract unsigned and signed numbers.
- Multiple number encodings to switch between numbers and their representations as bitvectors.

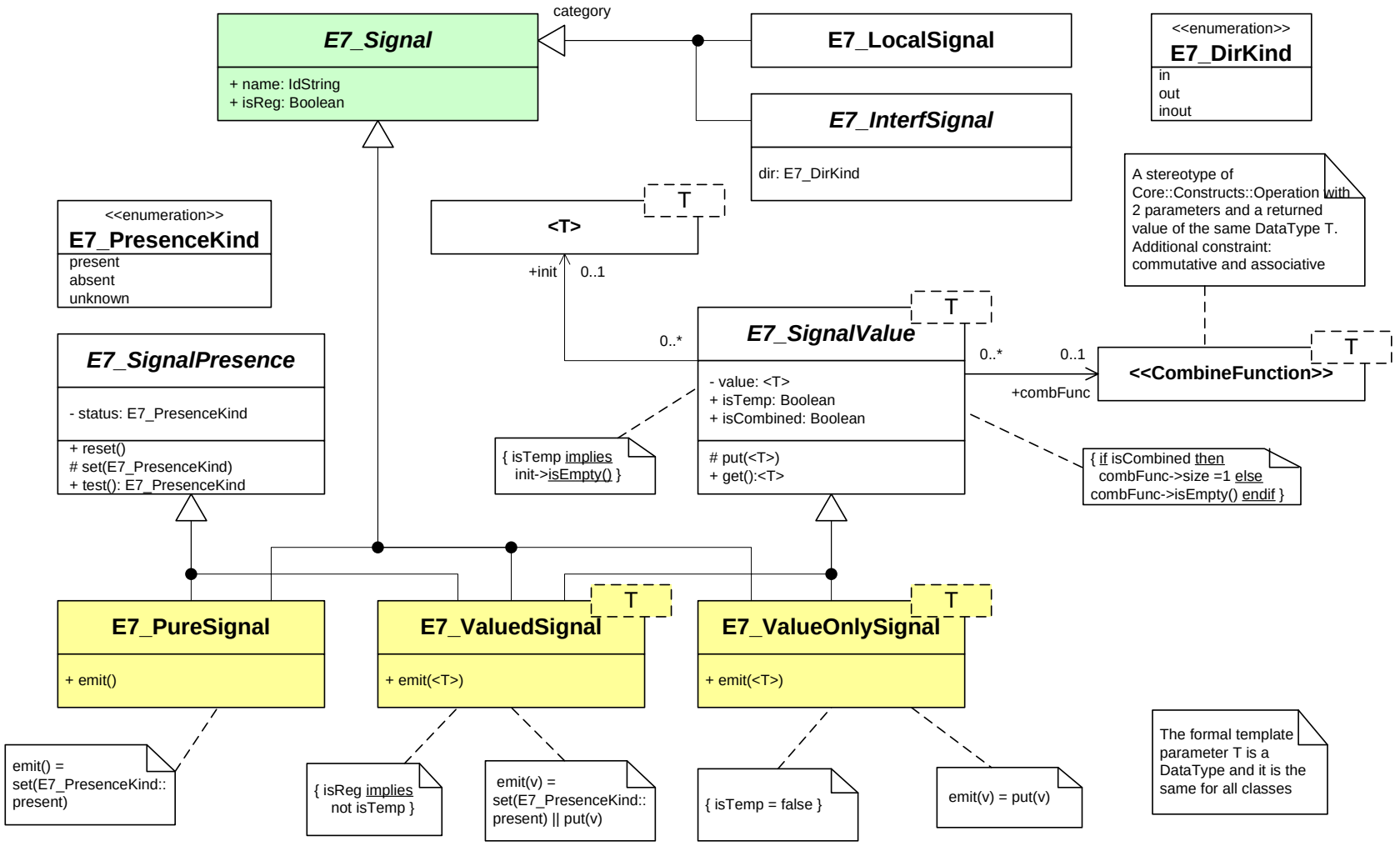
Esterel v7 (continued)

- Arrays of any dimension and arrays of arrays of any base type. Array expressions, with array slicing on any number of dimensions.
- Equational definitions of signals, which nicely complement Esterel imperative statements and are of major use in data path descriptions.
- Static replication of statements, fundamental for architectural design.
- Built-in assertions to check dynamic properties by simulation or formal verification.

Esterel v7 - Syntax



Esterel v7 - signals



Fibonacci Numbers

```
% esterel v5
module Fibonacci:

output F:integer;

signal F1:integer, F2:integer in
   emit F(1); emit F1(0); emit F2(0);
   pause;
   loop
       emit F1(pre(?F));
       emit F2(pre(?F1));
       emit F(?F1 + ?F2)
   each tick
end signal

end module
```

Fibonacci Numbers

```
// esterel v7
main module Fibonacci:
input S;
output F:unsigned;

abort
   signal F1:unsigned, F2:unsigned in
        emit {
                ?F <= 1,
                ?F1 <= 0,
                ?F2 <= 0
        };
        pause;
        sustain {
                ?F <= ?F1 + ?F2,
                ?F2 <= pre(?F1),
                ?F1 <= pre(?F)
        }
   end signal
when S
end module
```

New syntax for emission

Unordered signal equations

Signal equations in Esterel V7

equation ::=

sig-id <= *bool-exp*

sig-id if *bool-exp*

Pure signal

?*sig-id* <= *exp* if *bool-exp*

Valued signal

next *sig-id* …

Emit next

emit, sustain statements

emit [next] *equation*

emit [seq] [next] { {*equation*,} }

instantaneous

Sequential or concurrent execution of equations

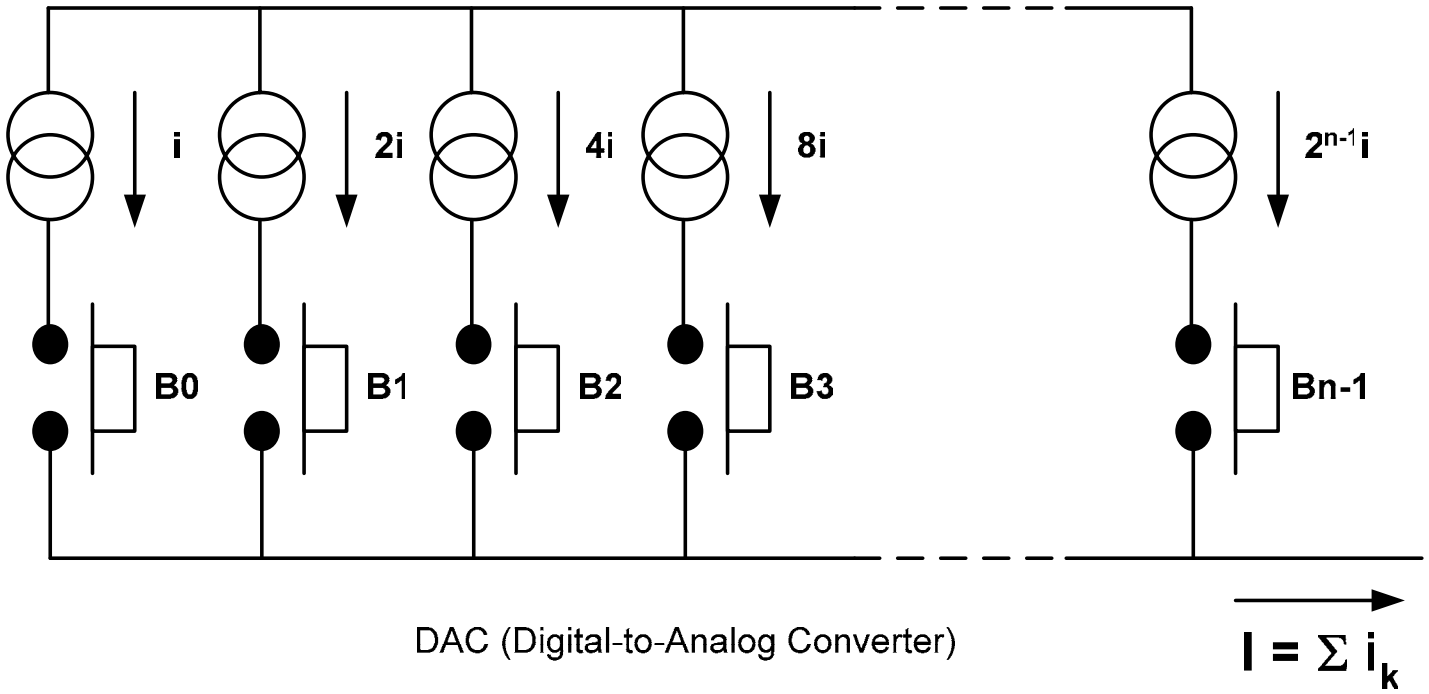
sustain [next] *equation*

sustain [seq] [next] { {*equation*,} }

Never terminates

Sequential or concurrent execution of equations

DAC



DAC (Digital-to-Analog Converter)

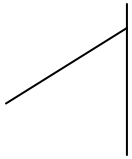
DAC - Esterel v5

```
% DAC.strl
% esterel v5
module DAC:

input B0, B1, B2, B3;
output I: combine integer with +;

Loop
   emit I(0)
||
   present B0 then emit I(1) end present
||
   present B1 then emit I(2) end present
||
   present B2 then emit I(4) end present
||
   present B3 then emit I(8) end present
each tick

end module
```



Use of combined signals

DAC - Esterel v7

```
main module DAC:
constant N: integer = 8;
input B[N], S;
output I:integer combine +;

abort
  sustain {
      ?I <= 0,
      for k<N dopar
            ?I <= 2**k if B[k],
      end for
  }
when S
end module
```

arrays

For loop

Factorial

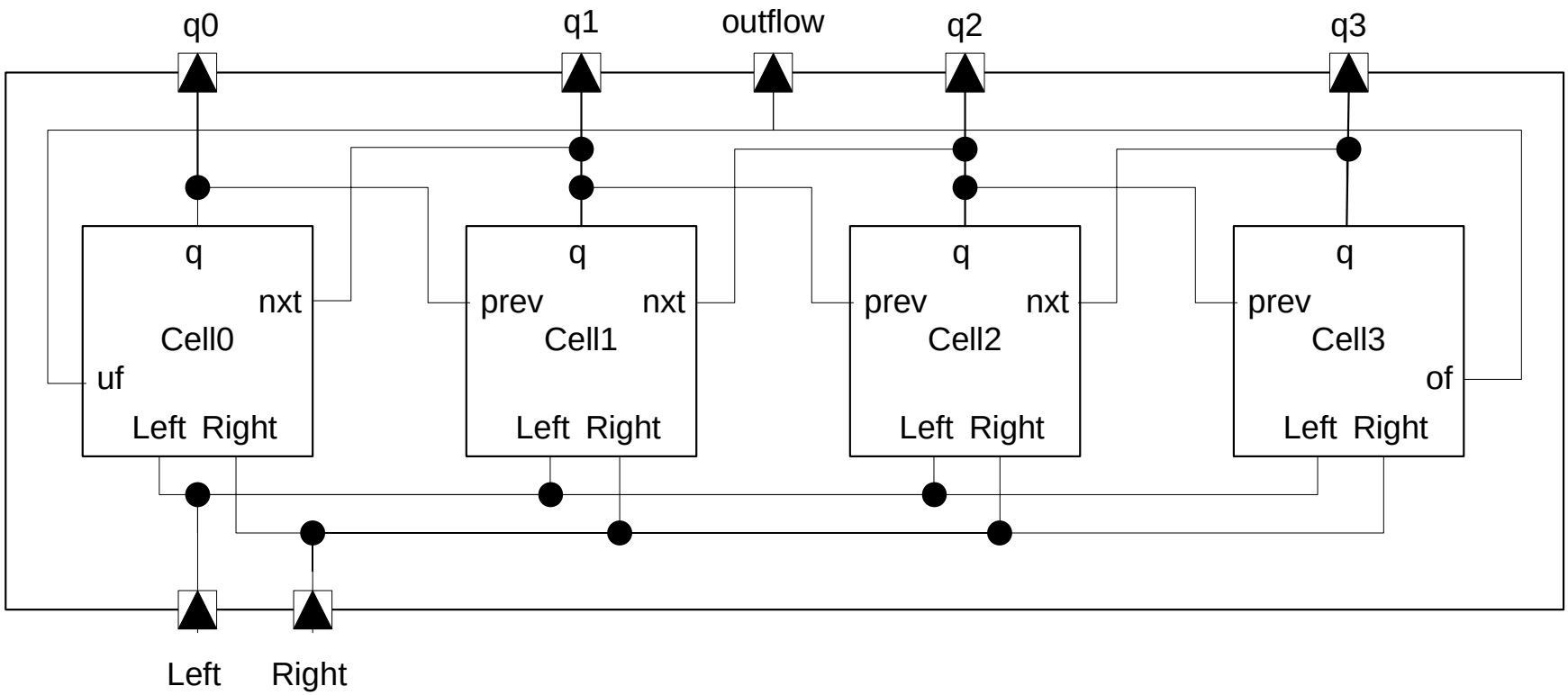
```
module SyncFactorial:
  constant N: integer;
  output Fact: unsigned combine *;
  for i < N dopar
     emit Fact(i+1);
  end for
end module
```

Registered signals

- Registered signals act with a delay of one tick for status and value transmission. Pure, full, and value-only signals can be registered. Local and output signals can be registered, but input signals cannot.
- Declaration: **`reg`** or **`reg1`**

- Emission: **`emit next S`**
- Test: as usual signals

Left/Right Shift Register



Left/Right Shift Register

```
module Cell:
input prev, nxt;
output q:reg;
input l,r;

sustain next q if
  (prev and r) or (nxt and l) or
  (q and not r and not l)

end module
```

Left/Right Shift Register

```
module Cell0:
input nxt;
output q:reg1;
input l,r;
output underflow;

  sustain {
     underflow if (q and l),
     next q if
          (nxt and l) or
          (q and not r)
  }
end module
```

Left/Right Shift Register

```
module CellLast:
input prev;
output q:reg;
input l,r;
output overflow;

  sustain {
     overflow if (q and r),
     next q if
          (prev and r) or
          (q and not l)
  }
end module
```

Left/Right Shift Register

```
main module ShiftReg:
constant N:integer = 4;
input l,r;
output outflow;

weak abort
   signal q[N+1] in
      run C0/Cell0 [q[0]/q, q[1]/nxt, outflow/underflow]
   ||
      for k in [1..N-2] dopar
        run Cell [q[k]/q, q[k+1]/nxt, q[k-1]/prev]
      end for
   ||
      run CN/CellLast [q[N]/q, q[N-1]/prev, outflow/overflow]
   end signal
when outflow

end module
```

Reincarnation

```
loop
  signal S in
     if S then emit O1 else
          emit O2 end if;
     pause;
     emit S
  end signal
end loop
```

Reincarnation

```
loop
  signal S in
     if S then emit O1 else emit O2 end;
     pause;
     emit S
  end signal ;
  signal S in
     if S then emit O1 else emit O2 end;
     pause;
     emit S
  end signal ;
end loop
```

Unfolding the loop

Fresh instance