The VIPER project has so far produced a formal specification of a 32 bit RISC microprocessor, an implementation of that chip in radiation-hard SOS technology, a partial proof of correctness of the implementation which is still being extended, and a large body of supporting software. The time has now come to consider what has been achieved and what directions should be pursued in future.

The most obvious lesson from the VIPER project has been the time and effort needed to use formal methods properly. Most of the problems arose in the interfaces between different formalisms e.g. between the (informal) English description and the HOL spec, between the block-level spec in HOL and the equivalent in ELLA needed by the low-level CAD tools. These interfaces need to be made rigorous or (better) eliminated.

VIPER 1A (the latest chip) is designed to operate in pairs, to give protection against breakdowns in service as well as design faults. We have come to regard redundancy and formal design methods as complementary, the one to guard against normal component failures and the other to provide insurance against the risk of the common-cause failures which bedevil reliability predictions.

Any future VIPER chips will certainly need improved performance to keep up with increasingly demanding applications. We have a prototype design (not yet specified formally) which includes 32 and 64 bit multiply, instruction pre-fetch, more efficient interface timing, and a new instruction to allow a quick response to peripheral requests. Work is under way to specify this device in MIRANDA, and then to refine the spec into a block-level design by top-down transformations. When the refinement is complete, a relatively simple proof checker should be able to demonstrate its correctness.
Example of NODEN output

The NODEN analysis suite provides automatic comparison between the specification and design of moderately complex blocks of logic. The following example is taken from the VIPER design. MINOR is the simplest block in the chip, essentially consisting of a three bit counter. Following this paragraph is its specification in NODEN-HDL, whilst on the following pages are a correct and incorrect implementation. The final page shows the output of the comparison program when presented with the erroneous circuit.

```
\ ** MINOR STATE LOGIC in NODEN ** \\

FN INCWORD3 = (word3: minor) -> word3:
   IF (VAL3 minor) = 7
    THEN WORD3 0
   ELSE WORD3((VAL3 minor)+1)
   FI.

BLOCK MINOR = (bool: nextmainbar advance reset intresetbar)
   -> (^word3: minor):
IF reset OR (NOT intresetbar) OR
  (advance AND (NOT nextmainbar))
  THEN WORD3 0
ELIF advance
  THEN INCWORD3 minor
ELSE minor
FI.
```
\**** 'Library' of primitive gate functions **** \\

FN INV = (bool: a ) -> bool: NOT a.
FN NAND2 = (bool: a b ) -> bool: NAND(a,b).
FN EXNOR = (bool: a b ) -> bool: a = b.
FN ORNAND = (bool: a b c d) -> bool: NAND(a OR b, c OR d).

\ NB. NAND3 & NAND4 are built-in functions \\

\ **** Correct gate level implementation **** \\

BLOCK MINOR = (bool: nextmnbar advance reset intrstbar)
  -> (^word3: minor):
BEGIN
  LET qbar_1 := NOT (minor[1]),
  qbar_2 := NOT (minor[2]),
  qbar_3 := NOT (minor[3]).
  LET gb2 := INV(advance).
  LET gb4 := INV(reset).
  LET gb1 := NAND4(nextmnbar, advance, gb4, intrstbar).
  LET gb3 := NAND3(gb2, gb4, intrstbar).
  LET gb7 := INV(qbar_1).
  LET gb8 := EXNOR(qbar_1, qbar_2).
  LET gb11 := INV(qbar_2).
  LET gb12 := NAND2(gb7, gb11).
  LET gb13 := EXNOR(gb12, qbar_3).
  OUTPUT (ORNAND(gb7, gb1, gb3, qbar_1),
          ORNAND(gb8, gb1, gb3, qbar_2),
          ORNAND(gb13, gb1, gb3, qbar_3))
END.
BLOCK M_ERR = (bool: nextmnbar advance reset intrstbar)
   -> (^word3: minor):
BEGIN
   LET qbar_1 := NOT (minor[1]),
       qbar_2 := NOT (minor[2]),
       qbar_3 := NOT (minor[3]).

   LET gb2 := INV(advance).
   LET gb4 := INV(reset).
   LET gb1 := NAND4(nextmnbar, advance, gb4, intrstbar).
   LET gb3 := NAND3(gb2, gb4, intrstbar).
   LET gb7 := INV(qbar_1).

   \ ** Inverted qbar_2 ** \\
   LET gb8 := EXNOR(qbar_1, NOT qbar_2).

   LET gb11 := INV(qbar_2).

   \ ** Missing NAND with gb7 ** \\
   LET gb12 := gb11.

   LET gb13 := EXNOR(gb12, qbar_3).

   \ ** Inverted first output ** \\
   OUTPUT (NOT(ORNAND(gb7, gb1, gb3, qbar_1)),
           ORNAND(gb8, gb1, gb3, qbar_2),
           ORNAND(gb13, gb1, gb3, qbar_3))
END.
Specification: 'MINOR'   Implementation: 'M_ERR'

COMPARISON ERROR: Implementation output 'minor[1]' is always incompatible with the specification of 'minor[1]'; output inverted?

COMPARISON ERROR: Implementation output 'minor[2]' is incompatible with the specification of 'minor[2] under the following circumstances:

\[
\begin{align*}
\text{nextmainbar} & = t \\
\text{advance} & = t \\
\text{reset} & = f \\
\text{intresetbar} & = t
\end{align*}
\]

For specification output 'minor[3]' - implementation output 'minor[3]':

WARNING: Specification depends on minor[1] and implementation doesn't

COMPARISON ERROR: Implementation output 'minor[3]' is incompatible with the specification of 'minor[3] under the following circumstances:

\[
\begin{align*}
\text{nextmainbar} & = t \\
\text{advance} & = t \\
\text{reset} & = f \\
\text{intresetbar} & = t \\
\text{minor[2]} & = f
\end{align*}
\]

*** Comparison fails, invalid implementation ***
NODEN changes

- Negative integer subranges allowed
  E.g. TYPE i8 = INT[-128..127].

- Automatic casts between types
  E.g. (t,t,f) + bool3_val + i8_val

- 2's compliment []bool to integer ops.

- Explicit legal value, !bool

- Compiler about four times faster.

- Analyer about twice as fast.
Old NODEN_HDL

FN INCWORD3 = (word3: minor) -> word3:
  IF (VAL3 minor) == 7 THEN WORD3 0 ELSE WORD3 ((VAL3 minor) + 1) FI.

New NODEN_HDL

FN INCWORD3 = (word3: minor) -> word3:
  IF minor == 7 THEN 0 ELSE minor + 1 FI.
Bibliography


Why VIPER2?

- Faster, 32 and 64 bit multiply
- Improved interface to outside world
- New design methods now available
Extra speed by ..

- Instruction pre-fetch

- Dedicated adders for P and indexing

- Half-cycle overlaps rather than full cycle

Speed more than 3x at same clock frequency
On-board Multiply Instructions

Three separate instructions, F = 13, 14, 15

- Signed, 32 bit product, stop on OVF
- Unsigned, LS 32 bits of product
- Unsigned, MS 32 bits of product
Improved interface

• "Call on signal" instruction

• "Frame restart" input

• Longer setup and hold times on memory and I/O cycles
New design methods

Top-down synthesis by correctness-preserving transformations

- Starts from specification in MIRANDA
- Generates proof as part of design process
- May scale up better than post hoc proof
VIPER 1A perspective

The present chip falls in between the main application areas:

- Automotive and comms: too expensive, minimum system too big (5 memory chips)
- Avionics: not fast enough, no multiply
- Space: about right, tiny market
Processor 1

Exactly Equal?

STOP

No

Yes

OPERATE

Processor 2
Dependable Error Reporting