Let's Play with Hardcaml!

Building Digital Circuits in an OCaml REPL

Kiwamu Okabe
kiwamu@metasepi.org
https://metasepi.org/



What Is Hardcaml?

A library for describing digital circuits (RTL) in OCaml

- Build circuits with ordinary OCaml functions
- Simulate a circuit cycle by cycle
- Generate Verilog and run it on an FPGA

Today we will use neither an FPGA nor a Verilog simulator.

We will play with just the OCaml REPL!



Let's Install It

On Debian / Ubuntu, use `opam`, the OCaml package manager.

```
$ sudo apt install opam
$ opam init
$ opam switch create hardcaml-5.3.0 ocaml-base-compiler.5.3.0
$ eval "$(opam env --switch=hardcaml-5.3.0)"
$ opam install hardcaml ppx_hardcaml hardcaml_waveterm utop
```




Start the OCaml REPL

```
$ utop
```

```
# #require "hardcaml";;
# #require "hardcaml_waveterm";;
# open Hardcaml;;
# open Signal;;
```

- `hardcaml`: describing and simulating digital circuits
- `hardcaml_waveterm`: recording and displaying waveforms

You can start without creating a project or source files.



Let's Build a 5-bit Counter!

```
# module Counter = struct
    let clock = input "clock" 1
    let count = reg_fb (Reg_spec.create ~clock ())
                  ~width:5 ~f:(fun d -> d +:. 1)
    let circuit = Circuit.create_exn ~name:"top"
                    [ output "count" count ]
  end;;
```

`reg_fb` is a register that feeds its own value back as the next input.

`~f:(fun d -> d +:. 1)` adds 1 on every clock.

That is all it takes to make a digital circuit!



What If We Wrote It in Verilog?

```
# Hardcaml.Rtl.print Verilog Counter.circuit;;
```

Rewriting the generated Verilog in a more readable form gives:

```
module top (
    input wire clock,
    output reg [4:0] count
);
    always @(posedge clock)
        count <= count + 5'd1;
endmodule
```

We were writing RTL in OCaml all along.



Let's Run the Circuit Right There!

```
# let waves, sim =
    Hardcaml_waveterm.Waveform.create
      (Cyclesim.create Counter.circuit);;
# for _ = 1 to 34 do
    Cyclesim.cycle sim;
    Bits.to_int !(Cyclesim.out_port sim "count") |> print_int;
    print_string " "
  done;;
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 0 1 2 - : unit = ()
```

An ordinary OCaml `for` loop can drive a circuit!



We Can Even See Waveforms!

```
# Hardcaml_waveterm.Waveform.print
    ~display_width:80 ~display_height:10 ~wave_width:1 waves;;
```

```
+Signals-----------+ +Waves-------------------------------------------------+
|clock              | |+-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-|
|                   | |  +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ |
|                   | |----+---+---+---+---+---+---+---+---+---+---+---+---+---+-|
|count              | | 00 |01 |02 |03 |04 |05 |06 |07 |08 |09 |0A |0B |0C |0D |0|
|                   | |----+---+---+---+---+---+---+---+---+---+---+---+---+---+-|
|                   | |                                                         |
|                   | |                                                         |
|                   | |                                                         |
+-------------------+ +---------------------------------------------------------+
```

We could do everything inside utop!



And Then, Linux Booted!

While playing with Hardcaml,
I ended up building a RISC-V CPU together with AI.

- RV32IMA / Sv32 MMU / 2 cores / 50 MHz
- Colorlight i9 v7.2 (ECP5 FPGA, 8 MiB SDRAM)
- **Linux 6.18 SMP** boots on real hardware

github.com/takibi-lang/takibi-hardware

