A Virtual Machine Approach for High-level FPGA Programming

Loïc Sylvestre[1] Jocelyn Sérot[2] Emmanuel Chailloux[1]

[1]LIP6, CNRS, Sorbonne Université [2]Université Clermont Auvergne, CNRS, SIGMA, Institut Pascal

❶ High-level FPGA programming

- **Objectives:**
  - enhance the programmability of FPGAs
  - allow quick prototyping
  - simulation and acceleration of applications
- **A virtual machine approach:**
  - **O2B** (*OCaml on board*)
    https://github.com/jserot/O2B
  - **Macle** (*ML accelerator*)
    https://github.com/lsylvestre/macle
- **OCaml:** A high-level programming language (functional, imperative, modular, object-oriented)

```
let main() =
  print_int (f 80);
  let x = 1000 and y = 12000000 in
  let t_c   = chrono gcd_c x y in
  let t_rtl = chrono gcd x y in
  let t_par =
    let src = Array.create (128*10) x in
    let dst = Array.create (128*10) x in
    chrono (map_gcd_by_100 y) src dst
  in
  print_int (t_c / t_rtl);
  print_int (t_c / t_par);
  try print_int (nth 42 [1;2;3;4]) with
  | Failure s -> print_string s ;;

main() ;;
```

❻ Preliminary evaluation

- on a small FPGA: 50K logic cells and 1,6 Kbits of on-chip memory with a clock frequency of 50 MHz
- `t_c / t_rtl`: hardware acceleration of the function `gcd` *versus* a C version running on the softcore provides a × **30 speedup**. Speedups depend on the nature of the computations. They can be higher than 30.
- `t_rtl / t_par`: hardware acceleration of `map_gcd_by_100` (using a parallel skeletons) provides an **extra speedup of almost 100**.
- `t_c / t_par`: resulting speedup of almost 3000.

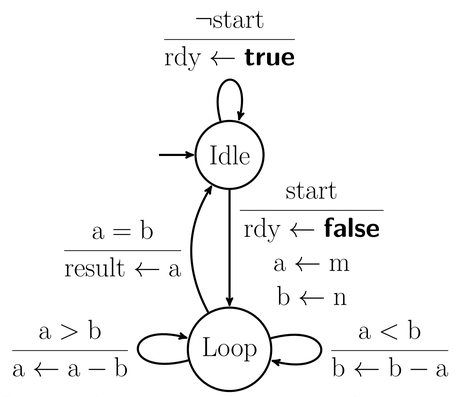
❸ Macle : a compiler for a subset of OCaml targeting the register transfer level

```
circuit gcd m n =
  let rec loop a b =
    if a > b then loop (a-b) b else
    if a < b then loop a (b-a)
             else a
  in loop m n ;;
```

Accelerated function `gcd`

```
circuit f x =
  let a = gcd x 120 and b = gcd x 2 in
  a + b
```

Macle exploits implicite parallelism of OCaml code.

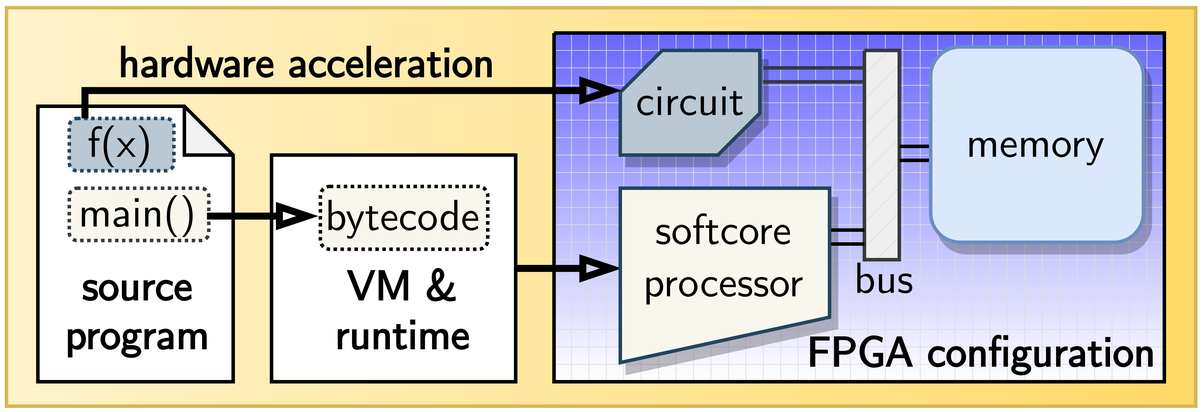


Intermediate representation of the function `gcd`

```
entity gcd is
  port(signal clk, reset : in std_logic;
       signal start : in std_logic;
       signal rdy : out std_logic;
       signal m, n : in signed(30 downto 0);
       signal result : out signed(30 downto 0));
end entity;
architecture rtl of gcd is
  type t_state is (Idle, Loop);
  signal STATE : t_state;
  signal a, b : signed(30 downto 0);
  begin process(reset,clk) begin
    if reset = '1' then
      STATE <= Idle;
    elsif rising_edge(clk) then
      case STATE is
        when Idle =>
          if start then
            rdy <= false;
            a <= m;
            b <= n;
            STATE <= Loop;
          else
            rdy <= true;
            STATE <= Idle;
          end if;
        when Loop =>
          if a > b then
            a <= a - b;
            STATE <= Loop;
          elsif a < b then
            b <= b - a;
            STATE <= Loop;
          else
            result <= a;
            STATE <= Idle;
          end if;
      end case;
    end if;
  end process;
end architecture;
```

VHDL description produced from the function `gcd`

❷ O2B : a softcore-based implementation of the OCaml Virtual Machine



❺ Parallelism skeletons

```
circuit map_gcd_by_100 y src dst =
  let gcd_y x = gcd x y in
  array_map 100 gcd_y src dst ;;
```

Accelerated function `map_gcd_by_100` processes an OCaml array `src` in parallel by packet of 100 elements (using 100 instances of function `gcd`). Results are transfered in the array `dst`. The level of parallelism (*ie.*, the time-space trade-off) is determined by the programmer. Data transfers are optimized to reduce the overhead of accessing the memory bus.

❹ Dynamic features

```
circuit nth n l =
  let rec loop i l =
    match y with
    | [] -> raise (Failure "nth")
    | x::t ->
      if i = n then x else loop (i+1) t
  in loop 0 l ;;
```

Accelerated function `nth` is able to manipulate lists dynamically allocated in the OCaml heap and raise an exception catchable by OCaml caller functions.