Литературный RTL

Васил Дядов

Воскресенье 22 июля 2018 г.

Содержание

1 Настройка Merlin для проекта

Добавляем все пакеты, что установленны в текущей конфигурации Opam, в файл `.merlin`.

```
opam list | grep -v '#' | awk '{print "PKG ", $1}'
```

2 Кодогенераторы на Hardcaml

2.1 Счётчик по-модулю

В основе своей, счётчик использует обычный регистр с обратной связью.

```
module ModCounter = struct
  <<open_modules>>
  let gen ~clr ~cntr_modulo ~cntr_width =
    let zero = consti cntr_width 0 in
    <<feedback_register>>
```

```
6
7  end
```

Раскрываемые в пространстве имён модули:

```
1  open HardCaml
2  open Signal.Comb
3  open Signal.Seq
```

Регистр задаётся следующим образом:

- `rsync` - шаблон для регистра с синхронным сбросом
- `(const "1'b1")` - константа для `enable` входа
- `cntr_width` - ширина
- и комбинаторная функция обратной связи

```
1  reg_fb
2    r_sync
3    (const "1'b1")
4    cntr_width
5    <<feedback_function>>
```

Функция для синхронного обнуления счётчика:

- входное значение `d` передаётся в функцию счёта
- результат идёт на мультиплексор который управлеятся сигналом сброса `clr`

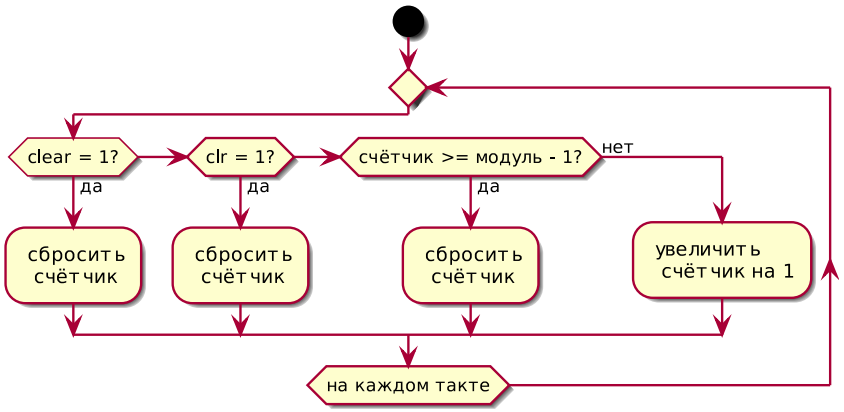
```
1  (fun d -> d |>
2            <<count_function>> |>
3              mux2 clr zero)
```

Функция для счёта по модулю:

- принимает входное значение `d`
- на выходе даёт либо `d` + 1, если `d` меньше модуля, либо 0

```
1  (fun d -> mux2 (d >=: (cntr_modulo -:. 1))
2              zero
3              (d +:. 1))
```

2.1.1 Алгоритм работы счётчика



2.1.2 Тестирование

1. Тестовое окружение Основой модуль с тестовым окружением.

```
module ModCounterTest = struct
  <<open_modules>>

  <<declare_inputs>>

  <<declare_outputs>>

  let gen_if i =
    let cntr_width = snd In.(t.modulo) in
    let q = In.(ModCounter.gen
                  ~clr:i.clr
                  ~cntr_modulo:i.modulo
                  ~cntr_width)
    in
    Out.({q})

  module B = Bits.Comb.IntbitsList
  module Builder = Interface.Gen(B)(In)(Out)

  let circuit, sim, i, o, _ =
    Builder.make "ModCounter" gen_if

  module S=Cyclesim.Api
end
```

Объявление входящих сигналов.

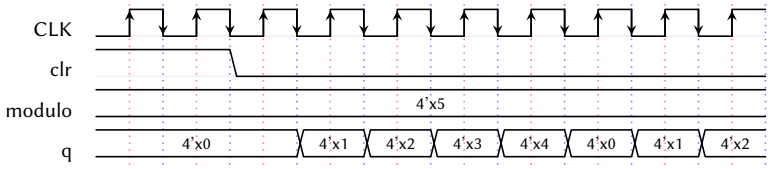
```
module In = struct
  type 'a t = {
      clr: 'a;
      modulo: 'a [@ bits 4]
    } [@@deriving hardcaml]
end
```

Объявление исходящих сигналов.

```
module Out = struct
  type 'a t = {
      q: 'a [@bits 4]
    } [@@deriving hardcaml]
end
```

2. Запуск теста

```
let _  =
  let open ModCounterTest in
  let open In in
  let open Out in
  let module TD = TimingDiagram(B) in
  let module R = TD.Recorder in
  let record = ref @@ R.of_interfaces [(module In); (module Out)] in
  let update = R.make_updater_ref [
       (module struct module Intf = In let intf = i end);
       (module struct module Intf = Out let intf = o end);
       ] in
  let step () =
    S.cycle sim;
    update record;
  in
  S.reset sim;
  i.modulo := B.consti 4 5;
  i.clr := B.vdd;
  step (); step ();
  i.clr := B.gnd;
  for _ = 0 to 7 do
    step ();
  done;
  !record
  |> TD.gen_latex
  |> print_string
```



3. Тест экспорта верилога

```
let _ =
  HardCaml.Rtl.Verilog.write
    print_string
    ModCounterTest.circuit
```

```
module ModCounter (
    clear,
    clock,
    modulo,
    clr,
    q
);

    input clear;
    input clock;
    input [3:0] modulo;
    input clr;
    output [3:0] q;

    /* signal declarations */
    wire _40 = 1'b1;
    wire [3:0] _42 = 4'b0000;
    wire vdd = 1'b1;
    wire [3:0] _43 = 4'b0000;
    wire [3:0] _39 = 4'b0000;
    wire [3:0] _45 = 4'b0001;
    wire [3:0] _46;
    wire [3:0] _47 = 4'b0001;
    wire [3:0] _48;
    wire _49;
    wire _50;
    wire [3:0] _51;
    wire [3:0] _52;
    wire [3:0] _41;
    reg [3:0] _44;

    /* logic */
    assign _46 = _44 + _45;
```

```
    assign _48 = modulo - _47;
    assign _49 = _44 < _48;
    assign _50 = ~ _49;
    assign _51 = _50 ? _39 : _46;
    assign _52 = clr ? _39 : _51;
    assign _41 = _52;
    always @(posedge clock) begin
        if (clear)
            _44 <= _42;
        else
            if (_40)
                _44 <= _41;
    end

    /* aliases */

    /* output assignments */
    assign q = _44;

endmodule
- : unit = ()
```

2.2 Разное тестирование

```
module FreqDivider = struct
  open HardCaml
  open Signal.Comb
  open Signal.Seq

  let gen ~clr ~div_by ~cntr_width =
    let mod_counter = ModCounter.gen
                        ~clr
                        ~cntr_modulo:div_by
                        ~cntr_width
    in
    ()
end
```

7

```
(* x 10)
```

```
let () = Printf.printf "\nlet max_value = %d;;\n" max_value;;
```

3 Код verilog

3.1 Общие параметры проекта

Параметр	Значение
Частота	100000000
Скорость UART	115200

Таблица 1: Таблица параметров

3.2 Счетчик по модулю

Параметр	Значение
Модуль	868
Ширина регистра	10

Таблица 2: Параметры счётчика

- Период счетчика и ширина регистра

10

При частотах
Test call:

```
let max_value = 70;;
```