

Библиотека кода для литературного RTL

Васил Дядов

Среда 25 июля 2018 г.

Содержание

1	Инициализация OCaml сессии	1
2	Установка принтера для сигналов	1
3	Записыватель сигналов	1
4	Генератор временных диаграмм	5
4.1	Тест генератора временных диаграмм	8

1 Инициализация OCaml сессии

```
1 #use "topfind";;  
2 #require "hardcaml";;  
3 #require "ppx_deriving_hardcaml";;  
4 #require "ppx_hardcaml";;
```

2 Установка принтера для сигналов

```
1 let print_signal fmt signal =  
2   Format.fprintf fmt "%s\n"  
3     (HardCaml.Signal.Comb.to_string signal);;  
4 #install_printer print_signal;
```

3 Записыватель сигналов

```

1 module SignalRecorder (S : HardCaml.Comb.S) =
2 struct
3   type data = (int * S.t) list
4
5   type signal = {name: string; data: data}
6
7   type t =
8     { from_time: int
9       ; to_time: int
10      ; signals: signal list }
11
12   let cut ~from_time ~to_time {name; data} =
13     let new_data =
14       data
15       |> List.filter (fun (t, v) ->
16         t > from_time && t <= to_time
17       )
18     in
19     let start_val =
20       data
21       |> List.find_opt (fun (t, _) ->
22         t <= from_time )
23     in
24     let data =
25       match start_val with
26       | None -> new_data
27       | Some (_, v) ->
28         new_data @ [(from_time, v)]
29     in
30     {name; data}
31
32   module type INTF = sig
33     type 'a t
34
35     val t : (string * int) t
36
37     val to_list : 'a t -> 'a list
38   end
39
40   module type INTF_IMPL = sig
41     module Intf : INTF
42
43     val intf : S.t ref Intf.t
44   end
45
46   let of_strings s =
47     let signals =
48       List.map

```

```

49         (fun name -> {name; data= []})
50     S
51     in
52     {from_time= 0; to_time= 0; signals}
53
54     let of_interfaces m =
55         let signals =
56             m
57             |> List.map (fun m ->
58                 let module M = ( val ( m
59                     : (module
60                         INTF)
61                     ) ) in
62                     M.to_list M.t |> List.map fst
63                 )
64             |> List.concat
65             |> List.map (fun name ->
66                 {name; data= []} )
67         in
68         {from_time= 0; to_time= 0; signals}
69
70     let update_signals ~time ~signals ~values =
71         signals
72         |> List.map
73             (fun ({name; data} as signal) ->
74                 let item =
75                     List.assoc_opt name values
76                 in
77                 match item with
78                 | None -> signal
79                 | Some v ->
80                     match data with
81                     | (_, old_v) :: _
82                     when old_v = v ->
83                         signal
84                     | _ ->
85                         { name
86                           ; data= (time, v) :: data
87                         } )
88
89     module Updater (P : INTF_IMPL) = struct
90         let update ?(time_inc= 2) ?time
91             {from_time; to_time; signals} =
92             let to_time =
93                 match time with
94                 | None -> to_time
95                 | Some t -> t
96         in

```

```

97   let updates =
98     let names =
99       P.Intf.to_list P.Intf.t
100     |> List.map fst
101   and vals =
102     P.Intf.to_list P.intf
103     |> List.map (fun x -> !x)
104   in
105     List.map2
106       (fun n v -> (n, v))
107       names vals
108   in
109     let signals =
110       update_signals to_time signals
111       updates
112   in
113     let to_time = to_time + time_inc in
114     {from_time; to_time; signals}
115
116   let update_ref ?time_inc ?time recorder =
117     recorder :=
118       update ?time_inc ?time !recorder
119 end
120
121 let make_updater i =
122   let rec loop i =
123     match i with
124     | [] -> fun ?time_inc ?time x -> x
125     | i :: rest ->
126       let module I = ( val ( i
127                           : (module
128                               INTF_IMPL)
129                           ) ) in
130       let module U = Updater (I) in
131       fun ?time_inc ?time s ->
132         s
133         |> U.update ?time_inc ?time
134         |> loop rest ?time_inc ?time
135   in
136   let updater = loop i in
137   fun ?time_inc ?time ({to_time; _} as s) ->
138     let time =
139       match time with
140       | None -> to_time
141       | Some t -> t
142   in
143   updater ?time_inc ~time s
144

```

```

145 let make_updater_ref i =
146   let updater = make_updater i in
147   fun ?time_inc ?time recorder ->
148     recorder :=
149       updater ?time_inc ?time !recorder
150
151 let update ?(time_inc= 2) ?time
152   {from_time; to_time; signals} values =
153   let to_time =
154     match time with
155     | None -> to_time
156     | Some t -> t
157   in
158   let signals =
159     update_signals ~time:to_time ~signals
160     ~values
161   in
162   let to_time = to_time + time_inc in
163   {from_time; to_time; signals}
164
165 let update_ref ?time_inc ?time recorder
166   values =
167   recorder :=
168     update ?time_inc ?time !recorder
169     values
170
171 let get_view ~from_time ~to_time
172   {signals; _} =
173   { from_time
174   ; to_time
175   ; signals=
176     signals
177     |> List.map
178       (cut ~from_time ~to_time) }
179 end

```

4 Генератор временных диаграмм

```

1 module TimingDiagram (S : HardCaml.Comb.S) =
2 struct
3   let header =
4     <<timing_diagram_header>>
5
6   let footer ~periods =
7     <<timing_diagram_footer>>

```

```

8
9 module Recorder = SignalRecorder (S)
10
11 type format = Bin | Dec | Hex
12
13 type spec = {fmt: format}
14
15 type specs = (string * spec) list
16
17 let default_spec = {fmt= Hex}
18
19 let draw_signal ~spec:{fmt; _} ~from_time
20   ~to_time ~signal:{Recorder.name; data} =
21   match data with
22   | [] -> "U"
23   | _ ->
24     let width =
25       S.width (snd (List.hd data))
26     in
27     let string_of_value v =
28       match width with
29       | 1 ->
30         if v = S.vdd then "h" else "l"
31       | _ ->
32         let open Printf in
33         let v = S.to_int v in
34         ( match fmt with
35         | Bin -> sprintf "d{%d}'d%d'"
36         | Dec -> sprintf "d{%d}'d%d'"
37         | Hex -> sprintf "d{%d}'x%X'"
38         )
39         width v
40     in
41     let times = List.map fst data in
42     let timediffs =
43       let open List in
44       map2 ( - ) (to_time :: times)
45       (rev (from_time :: rev times))
46       |> rev
47     in
48     let first_timediff, timediffs =
49       List.(hd timediffs, tl timediffs)
50     in
51     let values =
52       let open List in
53       data |> map snd
54       |> map string_of_value
55       |> rev
56     in

```

```

57     let timediffs, values =
58         if first_timediff = 0 then
59             (timediffs, values)
60         else
61             ( first_timediff :: timediffs
62               , "u" :: values )
63     in
64     let waveform =
65         List.(
66             map2 ( ^ )
67                 (map string_of_int timediffs)
68                 values)
69     |> String.concat " "
70     in
71     name ^ "&" ^ waveform ^ "\\\\\n"
72
73 let out_signals ~clock_name ~default_spec
74     ~specs
75     ~recorder:{ Recorder.from_time
76                 ; to_time
77                 ; signals } =
78     let periods = to_time - from_time in
79     clock_name ^ "& "
80     ^ string_of_int periods
81     ^ "{c}\\\\\n"
82     ^ ( signals
83         |> List.map
84           (fun ( {Recorder.name; data} as
85                 signal )
86             ->
87                 let spec =
88                     List.assq_opt name specs
89                 in
90                 let spec =
91                     match spec with
92                     | None -> default_spec
93                     | Some spec -> spec
94                 in
95                 draw_signal ~spec ~from_time
96                     ~to_time ~signal )
97         |> String.concat " " )
98     ^ footer ~periods:(periods / 2)
99
100 let gen_latex ?(clock_name= "CLK")
101     ?(default_spec= default_spec)
102     ?(specs= []) recorder =
103     header
104     ^ out_signals ~clock_name ~default_spec

```

```
105 ~specs ~recorder
106 end
```

4.1 Тест генератора временных диаграмм

```
1  let _ =
2    let module B = HardCaml.Bits.Comb.
3      IntbitsList in
4    let module TD = TimingDiagram (B) in
5    let module R = TD.Recorder in
6    let recorder =
7      R.of_strings ["clear"; "data"]
8    in
9    List.fold_left R.update recorder
10     [ [ ("clear", B.constb "1")
11       ; ("data", B.constb "0010") ]
12     ; [ ("clear", B.constb "1")
13       ; ("data", B.constb "0001") ]
14     ; [ ("clear", B.constb "0")
15       ; ("data", B.constb "1000") ]
16     ; [ ("clear", B.constb "0")
17       ; ("data", B.constb "0101") ] ]
18    |> TD.gen_latex
19    |> Printf.printf "%s\n%!"
```