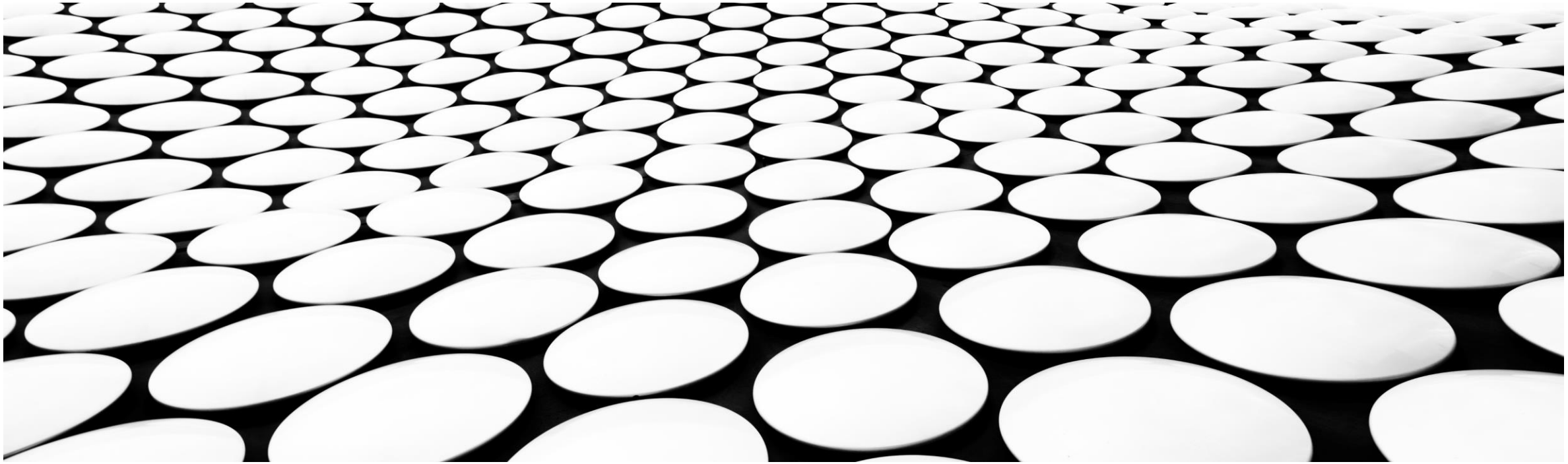
SMART STACKS IN VHDL

EuroForth 2020
Andrew Read, Ulli Hoffmann



WHY VHDL AT EUROFORTH?

SOFTWARE
- C

HARDWARE
- CPU, MCU

Do you prefer a hard or soft boundary?

SOFTWARE
- FORTH

HARDWARE
- FPGA (VHDL)

"Delegate to the hardware"

*"Delegate to the hardware" concept credit Klaus Schleisiek and Microcore

A STACK AS MEMORY

```
1  entity stack_1 is
2      generic(width : natural;
3          depth : natural );
4      port(   clk : in std_logic;
5          rst : in std_logic;
6          input : in std_logic_vector(width - 1 downto 0);
7          stack_pointer_n : in integer range 0 to depth - 1;
8          write_enable : in std_logic;
9          output : out std_logic_vector(width - 1 downto 0);
10         stack_pointer : out integer range 0 to depth - 1
11     );
12 end entity;
```

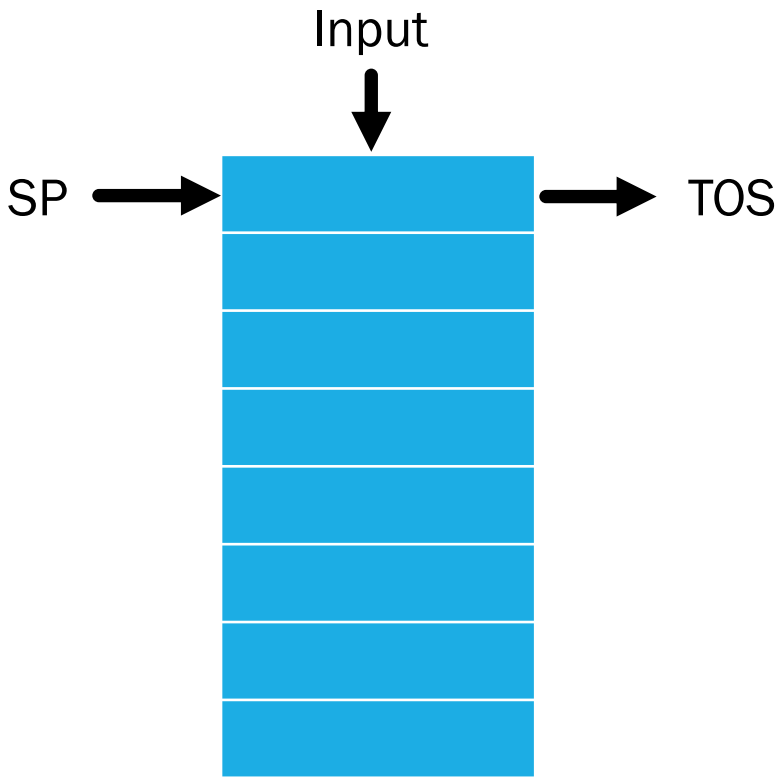
A STACK WITH OPERATIONS

```
15  entity stack_2 is
16      generic(width : natural;
17          depth : natural );
18      port(   clk : in std_logic;
19          rst : in std_logic;
20          input : in std_logic_vector(width - 1 downto 0);
21          stack_op : in stack_op_type;
22          output : out std_logic_vector(width - 1 downto 0);
23          stack_pointer : out integer range 0 to depth - 1;
24          err_under : out std_logic;
25          err_over : out std_logic
26      );
27  end entity;
28
29  type stack_op_type is
30      (s_nop, s_push, s_drop, s_replace, s_reset);
```

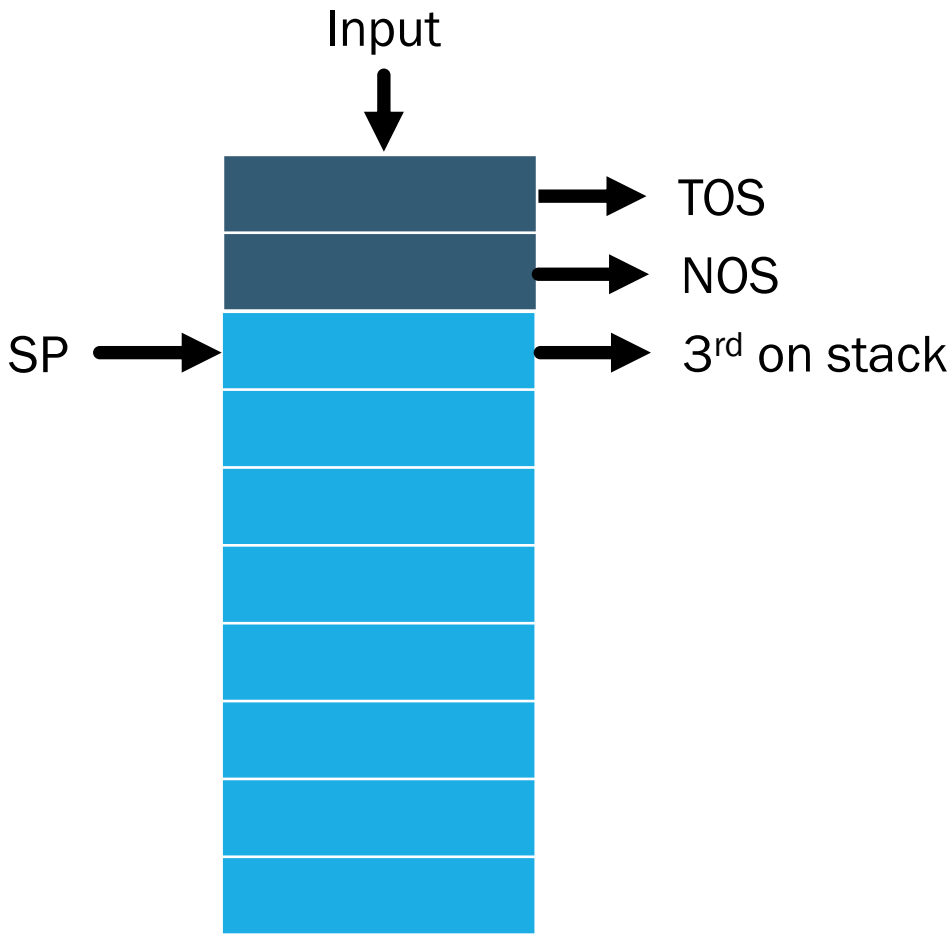
INSIDE THE ARCHITECTURE

```
33  case stack_op is
34      when s_push =>
35          we <= '1';  sp_n <= sp_inc;
36
37      -- other cases
38
39      when s_nop =>
40          we <= '0';  sp_n <= sp;
41
42  end case;
43
```

AUGMENTING WITH REGISTERS



Stack implemented in memory



Stack implemented in registers and memory

SOME MORE OPERATIONS

```
61  type stack_op_type is
62      (s_nop, s_push, s_drop, s_replace, s_reset,
63      s_nip, s_replaceAndNip, s_dup, s_ifDup,
64      s_swap, s_rot, s_over,
65      s_depth
66      );
```

A SMART STACK

```
45  entity stack_3 is
46      generic(width : natural;
47          depth : natural );
48      port(   clk : in std_logic;
49          rst : in std_logic;
50          input : in std_logic_vector(width - 1 downto 0);
51          stack_op : in stack_op_type;
52          tos : out std_logic_vector(width - 1 downto 0);
53          nos : out std_logic_vector(width - 1 downto 0);
54          stack_pointer : out integer range 0 to depth - 1;
55          err_under : out std_logic;
56          err_over : out std_logic
57      );
58  end entity;
```

EXCEPTION HANDLING?

- CATCH and THROW necessitate special stack handling
- Typical software implementations rely on hooks for reading and writing the stack pointers
- But..
- This breaks the hardware abstraction
- Can we delegate exception handling to hardware?

A STACK WITH TWO STACKS INSIDE

```
45  entity stack_3 is
46      generic(width : natural;
47          depth : natural );
48      port(   clk : in std_logic;
49          rst : in std_logic;
50          input : in std_logic_vector(width - 1 downto 0);
51          stack_op : in stack_op_type;
52          tos : out std_logic_vector(width - 1 downto 0);
53          nos : out std_logic_vector(width - 1 downto 0);
54
55
56
57
58
```

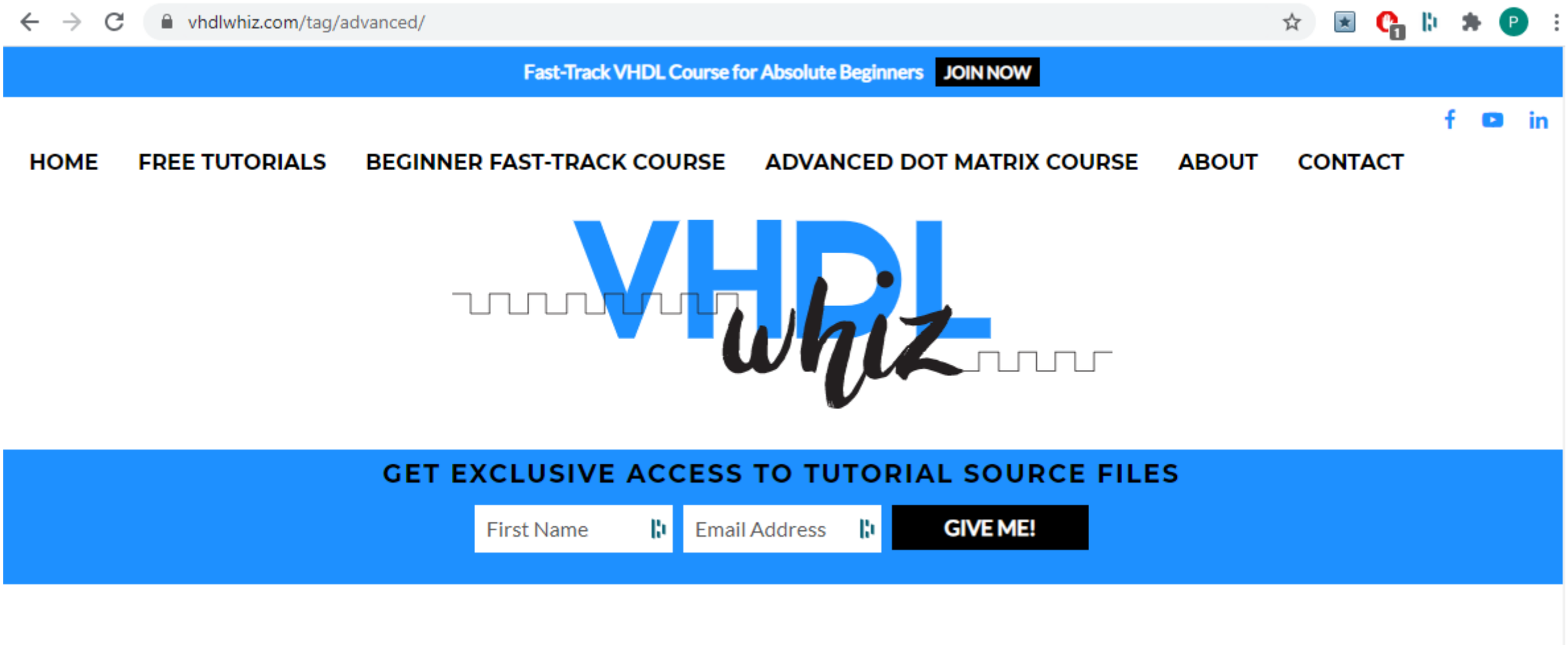
```
1   entity stack_1 is
2       generic(width : natural;
3           depth : natural );
4       port(   clk : in std_logic;
5           rst : in std_logic;
6           input : in std_logic_vector(width - 1 downto 0);
7           stack_pointer_n : in integer range 0 to depth - 1;
8           write_enable : in std_logic;
9           output : out std_logic_vector(width - 1 downto 0);
10          stack_pointer : out integer range 0 to depth - 1
11      );
12  end entity;
```

```
15  entity stack_2 is
16      generic(width : natural;
17          depth : natural );
18      port(   clk : in std_logic;
19          rst : in std_logic;
            input : in std_logic_vector(width - 1 downto 0);
21          stack_op : in stack_op_type;
22          output : out std_logic_vector(width - 1 downto 0);
23          stack_pointer : out integer range 0 to depth - 1;
24          err_under : out std_logic;
25          err_over : out std_logic
26      );
27  end entity;
```

OPERATIONS FOR EXCEPTION HANDLING

```
69    s_saveSP, s_restoreSP, s_dropS
70
71    s_saveSPAndPush, s_restoreSPAndPush, s_dropSPAndDrop
72
```

RECOMMENDATION: vhdlwhiz.com



BRIEF DEMONSTRATION!

CONCLUSION

We have developed a *smart stack* approach to hardware stacks in VDHL which focuses on abstraction and scalability.

Two smart stacks, encapsulated as a single entity, provide simple exception handling.

This work is a spin-off of our research and development in seedForth.

Ulrich Hoffmann (FH Wedel University of Applied Sciences),
uh@fh-wedel.de

Andrew Read
andrew81244@outlook.com