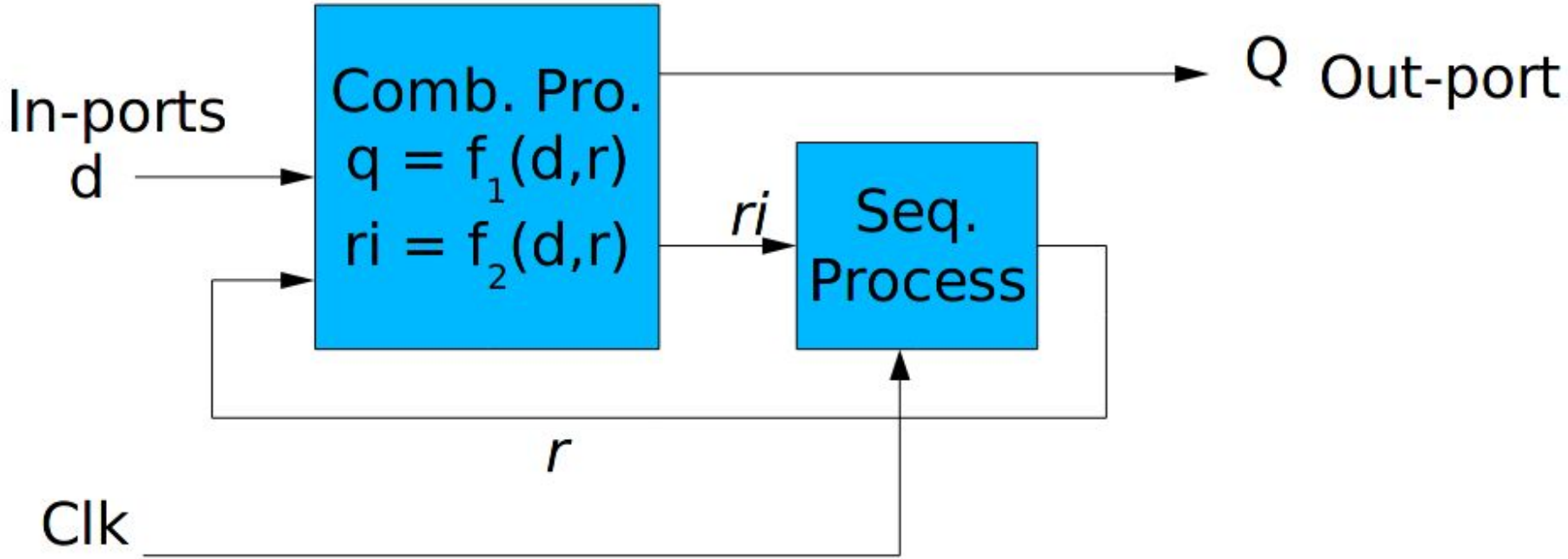
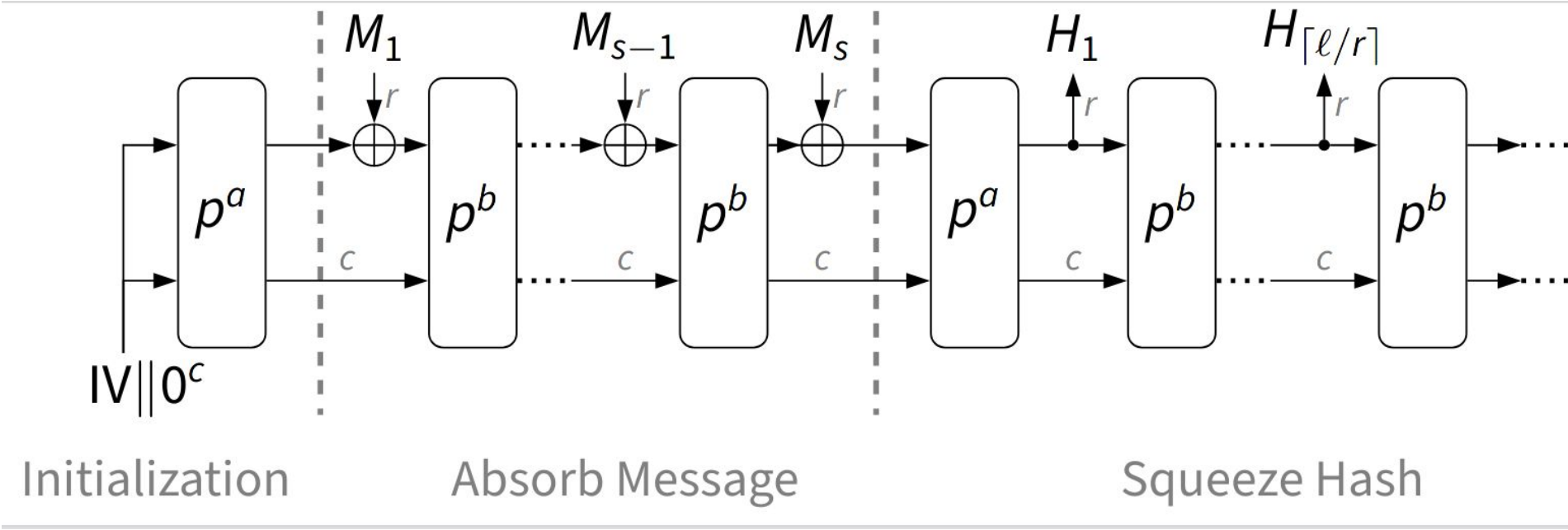
A Structured VHDL Design Method (Gaisler Method)


In-ports
d
Comb. Pro.
q = f1(d,r)
ri = f2(d,r)
ri
Seq.
Process
Q
Out-port
r
Clk

Ascon Hash


M_1
M_{s-1}
M_s
H_1
$H_{\lceil \ell/r \rceil}$
r
c
p^a
p^b
$IV||0^c$
Initialization
Absorb Message
Squeeze Hash

Record types as Abstraction Layers

```
type control_t is record
  valid : std_logic;
  ready : std_logic;
  last : std_logic;
  start : std_logic;
end record control_t;

type status_t is record
  valid : std_logic;
  ready : std_logic;
  done : std_logic;
end record status_t;
```

Record types as Abstraction Layers

```
type control_t is record
  valid : std_logic;
  ready : std_logic;
  last : std_logic;
  start : std_logic;
end record control_t;

type status_t is record
  valid : std_logic;
  ready : std_logic;
  done : std_logic;
end record status_t;
```

```
type data_in_t is record
  data : std_logic_vector(63 downto 0);
  control : control_t;
end record data_in_t;

type data_out_t is record
  data : std_logic_vector(63 downto 0);
  status : status_t;
end record data_out_t;
```

Record types as Abstraction Layers

```
type data_in_t is record
  data : std_logic_vector(63 downto 0);
  control : control_t;
end record data_in_t;

type data_out_t is record
  data : std_logic_vector(63 downto 0);
  status : status_t;
end record data_out_t;
```

```
entity AsconHash is

  port (
    clk : in std_logic;
    rst : in std_logic;
    in_bus : in data_in_t;
    out_bus : out data_out_t
  );
end entity AsconHash;
```

Port mappings become simpler and never need to be changed in multiple places!

AsconHash Implementation

```
type state_t is (
  IDLE,
  INIT,
  ABSORB,
  PERMUTE,
  SQUEEZE
);

type registers_t is record
  ascon_state : std_logic_vector(319 downto 0);
  round_cnt : integer range 0 to 11;
  squeeze_cnt : integer range 0 to 3;
  state : state_t;
  last : std_logic;
end record registers_t;
```

```
if rising_edge(clk) then
  if en_tag = '1' then
    tag_r <= cascade_key & tag_r(DBLK_SIZE -1 downto CCW);
  end if;
  if en_bdi_eot = '1' then
    bdi_eot_r <= bdi_eot;
  end if;
  if en_bdi_eoi = '1' then
    bdi_eoi_r <= bdi_eoi;
  end if;
  ...

end if;
```

```
seq : process(clk)
begin
  if rising_edge(clk) then
    if rst = '1' then
      reg_s.round_cnt <= 0;
      reg_s.squeeze_cnt <= 0;
      reg_s.ascon_state <= (others => '0');
      reg_s.state <= INIT;
      reg_s.last <= '0';
    else
      reg_s <= n_reg_s;
    end if;
  end if;
end process seq;
```

Bulk assignment prevents accidental latches

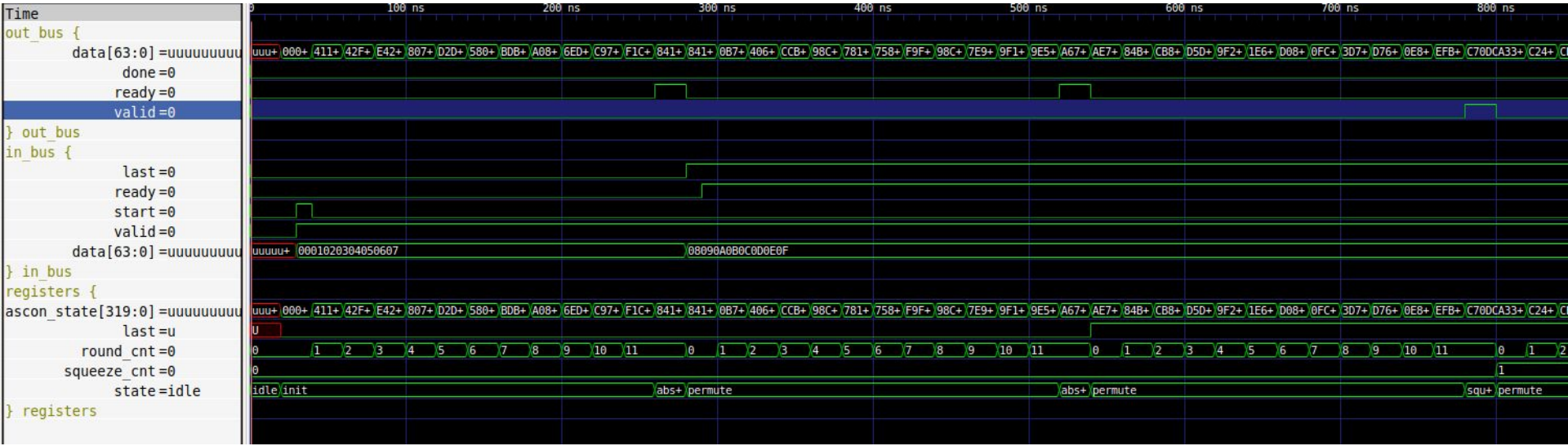
```
comb : process(all)
  variable reg_nx : registers_t;
begin
  reg_nx := reg_s;
  out_bus.status.valid <= '0';
  out_bus.status.ready <= '0';
  out_bus.status.done <= '0';
  out_bus.data <= reg_s.ascon_state(319 downto 256);
```

```
case reg_s.state is
  when IDLE =>
    if in_bus.control.start = '1' then
      -- set IV
      reg_nx.ascon_state(x0'range) := x"00400c0000000100";
      reg_nx.ascon_state(255 downto 0) := (others => '0');
      reg_nx.round_cnt := 0;
      reg_nx.squeeze_cnt := 0;
      reg_nx.state := INIT;
    end if;
  when INIT =>
    reg_nx.ascon_state := n_ascon_state;
    if reg_s.round_cnt = 11 then
      reg_nx.state := ABSORB;
    else
      reg_nx.round_cnt := reg_s.round_cnt + 1;
    end if;
```

```
when ABSORB =>
  out_bus.status.ready <= '1';
  if in_bus.control.valid = '1' then
    -- abosrb msg
    reg_nx.ascon_state(x0'range) := reg_s.ascon_state(x0'range) xor
                                    in_bus.data;
    reg_nx.round_cnt := 0;
    reg_nx.state := PERMUTE;
    reg_nx.last := in_bus.control.last;
  end if;
when PERMUTE =>
  reg_nx.ascon_state := n_ascon_state;
  if reg_s.round_cnt = 11 then
    if reg_s.last = '1' then
      reg_nx.state := SQUEEZE;
    else
      reg_nx.state := ABSORB;
    end if;
  else
    reg_nx.round_cnt := reg_s.round_cnt + 1;
  end if;
```

```
      when SQUEEZE =>
        out_bus.status.valid <= '1';
        --if in_bus.control.ready = '1' then
          if reg_s.squeeze_cnt = 3 then
            reg_nx.state := IDLE;
            out_bus.status.done <= '1';
          else
            reg_nx.squeeze_cnt := reg_s.squeeze_cnt + 1;
            reg_nx.round_cnt := 0;
            reg_nx.state := PERMUTE;
          end if;
        --  end if;
      end case;
    n_reg_s <= reg_nx;
  end process comb;
end architecture rtl;
```

Logically group signals during debugging



https://gaisler.com/doc/structdesign.pdf