Digital Systems - Mini AVR 4

Pere Palà - Alexis López

iTIC http://itic.cat

May 2016

Input Ports

- We have mapped some registers as outputs
- Inputs?
- Specific instructions to interact with the world.
  - IN Rd,A
  - OUT A,Rr

Documentation of AVR instructions : IN

- IN - Load an I/O Location to Register
- Description:
  - Loads data from the I/O Space (Ports, Timers, Configuration Registers etc.) into register Rd in the Register File.
- Rd $\leftarrow$ I/O(A)
- Syntax: IN Rd,A
- Operands: $0 \leq d \leq 31$, $0 \leq A \leq 63$
- Program Counter: PC $\leftarrow$ PC $+ 1$
- 16-bit Opcode: `1011 0AAd dddd AAAA`
- Status Register (SREG) and Boolean Formula:

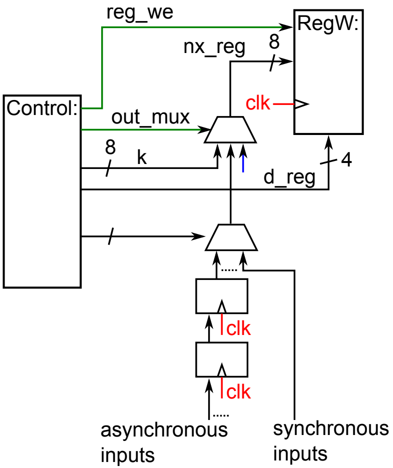
```
I T H S V N Z C
- - - - - - - -
```

Documentation of AVR instructions : OUT

- OUT – Store Register to I/O Location
- Description:
  - Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers etc.).
- I/O(A) $\leftarrow$ Rr
- Syntax: OUT A,Rr
- Operands: $0 \leq r \leq 31$, $0 \leq A \leq 63$
- Program Counter: PC $\leftarrow$ PC $+ 1$
- 16-bit Opcode: `1011 1AAr rrrr AAAA`
- Status Register (SREG) and Boolean Formula:

```
I T H S V N Z C
- - - - - - - -
```

Schematic: IN



VHDL implementation. IN

```vhdl
case pr_op(15 downto 12) is
  ...
  when IN_OUT =>
     if pr_op(11)='1' then -- OUT
     else                  -- IN
        out_mux <= mux_in_out;
        d_reg  <= pr_op(7 downto 4); --Destination register
        reg_we <= '1';
        case pr_op(1 downto 0) is    --Decode IN address
           when "00" => in_data <= synchA0;
           when "01" => in_data <= synchA1;
           when "10" => in_data <= timer_state;
           when others => null;
        end case;
     end if;
```

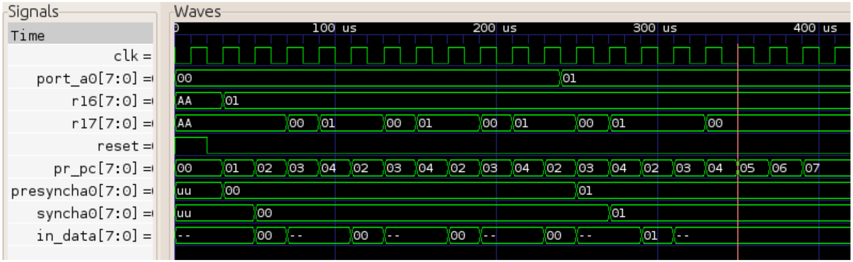
```vhdl
MUX: process(out_mux, alu_out, k, in_data) --Out MUX
   begin
      case out_mux is
      ...
      when mux_in_out => nx_reg <= in_data;
      when others     => nx_reg <= (others => '-');
      end case;
   end process;
```

VHDL implementation. IN (synchronizer)

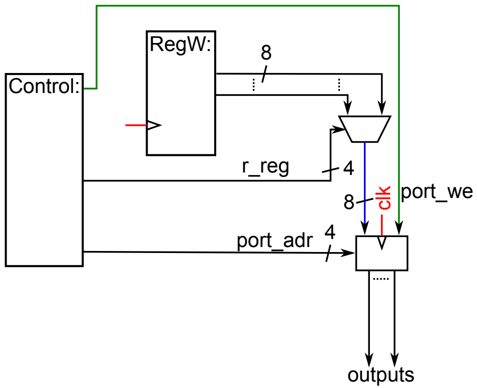
```
process(clk,reset) --Synchronous elements
begin
   if reset='1' then
   ...
   elsif (rising_edge(clk)) then
      ...
      -- synchronizer for inputs ports
      presynchA0 <= port_A0;
      synchA0    <= presynchA0;
      presynchA1 <= port_A1;
      synchA1    <= presynchA1;
   end if;
end process;
```

Sample code

```
NOP                 ; program waits until r17=x01
LDI   r16,x01
IN    r17,x0        ; <----
EOR   r17,r16       ;       |
BRNE  -3            ; -----
NOP
NOP
RJMP  -1            ; HALT
```



Schematic: OUT



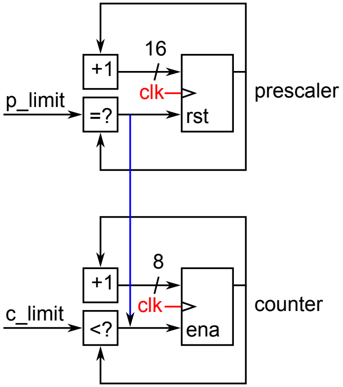
VHDL implementation. OUT

```
... -- add new default (port_we,...)
case pr_op(15 downto 12) is
  ...
  when IN_OUT =>
     if pr_op(11)='1' then -- OUT
        port_we  <= '1';
        port_adr <= pr_op(3 downto 0);
        r_reg <= pr_op(7 downto 4);
     else                         -- IN
     end if;
```

VHDL implementation. OUT

```
PortW: process(clk)
begin
   if rising_edge(clk) then
      if port_we = '1' then
         case port_adr is
         when "1000" =>
            port_A8i <= regs(to_integer(unsigned(r_reg)));
         when "1001" =>
            port_A9i <= regs(to_integer(unsigned(r_reg)));
         when "1010" =>
          timer_limit<= regs(to_integer(unsigned(r_reg)));
         when others => null;
         end case;
      end if;
   end if;
end process PortW;
```

Implementing a Timer



- Prescaler limit : hard coded
  - Each prescaler terminal count enables the timer counter
- Counter limit : `timer_limit` mapped to port 10 `1010`
- Counter keeps counting until reaching the limit
- This is signaled through `timer_state` mapped to port 2 `0010`

VHDL implementation of timer

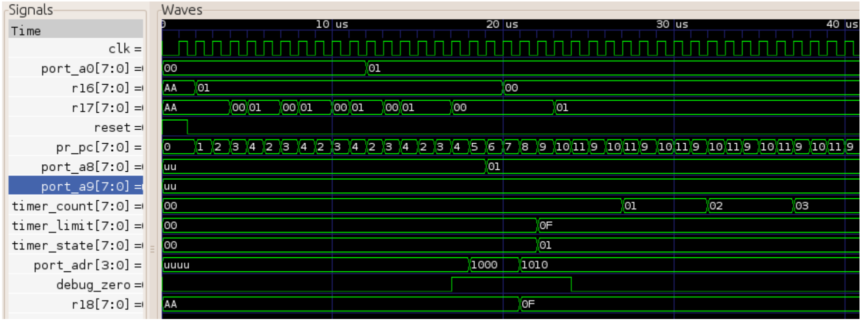
```vhdl
timer_state <= x"00" when timer_count = timer_limit
                     else x"01";  -- 1:counting 0:finished
presc_tc <= '1' when presc_count=presc_count_limit --constant
                     else '0';
process(clk)
begin
  if rising_edge(clk) then
     if port_we='1' and port_adr="1010"then --reset counter
        timer_count <= (others => '0');     --when writing
        presc_count <= (others => '0');     --new timer_limit
     else
        if timer_state /= x"00" and presc_tc='1' then
           timer_count <=
              std_logic_vector(unsigned(timer_count)+1);
        end if;
        if presc_tc='1' then
           presc_count <= (others => '0');
        else
           presc_count <=
              std_logic_vector(unsigned(presc_count)+1);
        end if;
     end if;
   end if;
end process;
```

Sample code with Timer

```
NOP              ; program waits until r17=x01
LDI  r16,x01
IN   r17,x0      ; <----
EOR  r17,r16     ;      |
BRNE -3          ; -----
OUT 8,r16
EOR r16,r16     ; clear r16
LDI r18,x0F     ; value of timer_limit
OUT 10,r18      ; set timer_limit
IN  r17,2       ; read timer_state  <-----------------
OR  r17,r16     ; r16 is zero, this tests for zero    |
BRNE -3         ; ------------------------------------
RJMP -1          ; HALT
```

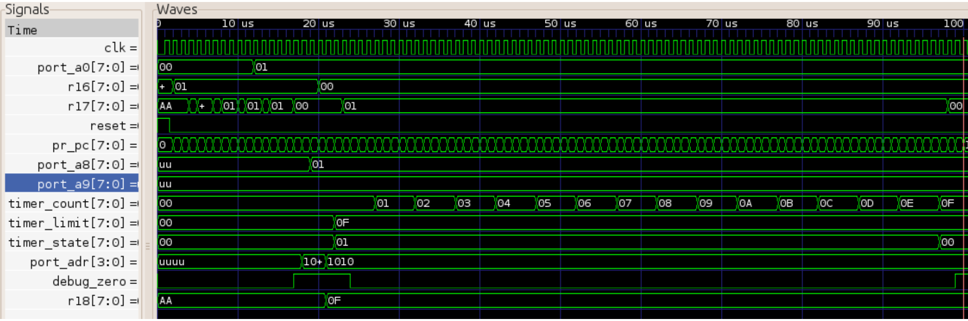
Sample code with Timer. Initial Waveforms

Simulation with `presc_count_limit` = 4, meaning 5 system clocks per terminal count



Sample code with Timer. Initial Waveforms

Simulation with `presc_count_limit` $= 4$



- Instruction #8 (`OUT 10,r18`) sets `timer_limit=x0F` at $t = 21\ \mu$s, its effect being seen at $t = 22\ \mu$s
- `timer_state` becomes 1 at $t = 22\ \mu$s and reaches 0 at $t = 97\ \mu$s. This means an interval of 85 $(5 \times 15)\ \mu$s
- loop exits after the next read of `timer_state` at $t = 100\ \mu$s