Design of Embedded Hardware and Firmware

"System On Programmable Chip"

Programmable interface design

GPIO exemple

Andres Upegui, René Beuchat

Laboratoire de Systèmes Numériques
hepia/HES-SO
4, rue de la Prairie, CH-1202 Genève

Andres.upegui@hesge.ch

MASTER OF SCIENCE
IN ENGINEERING

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Goal

Example of a development methodology of a programmable parallel port interface

The objective here is to design an interface for an Avalon bus as a slave module.

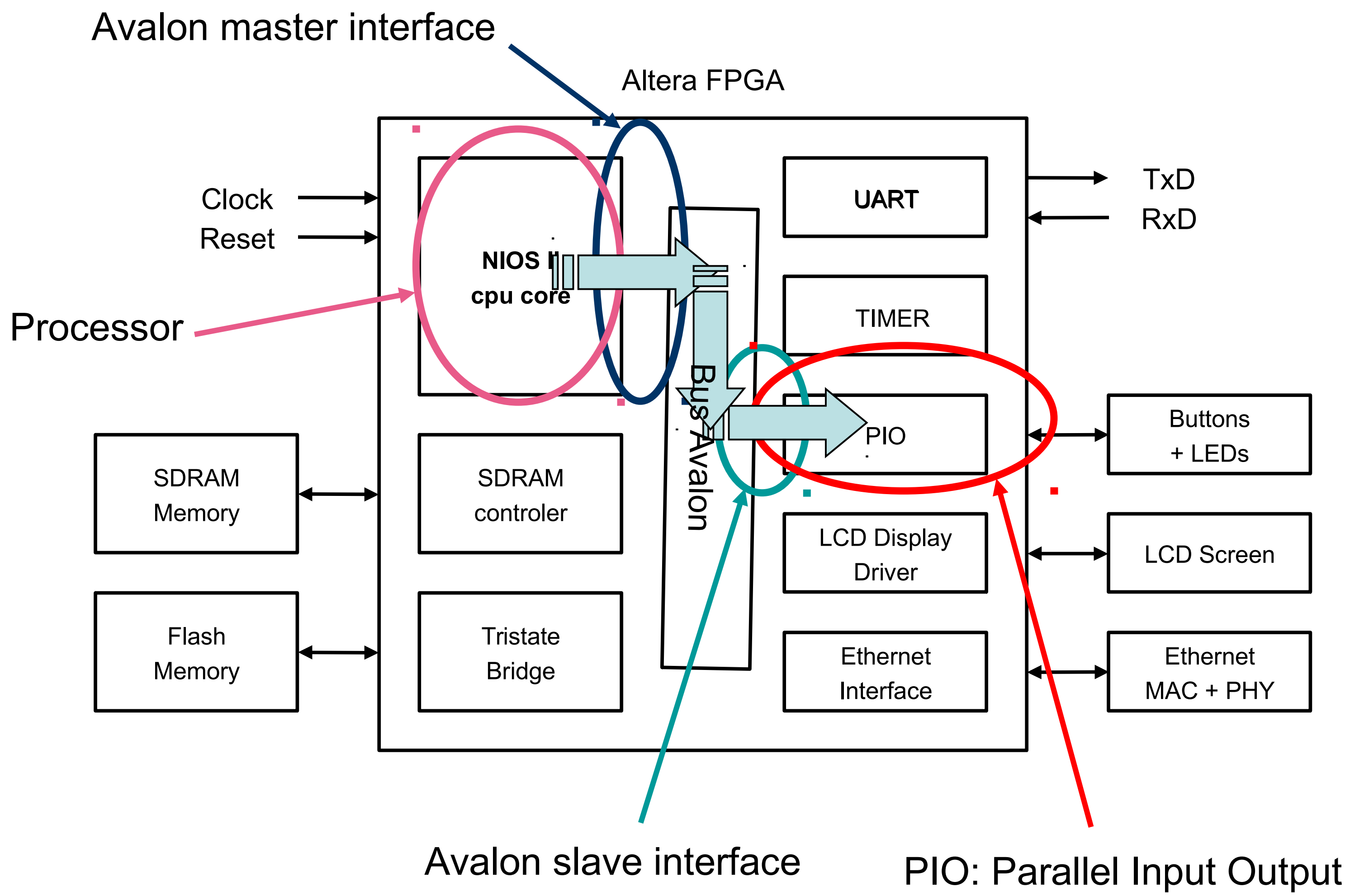
The main characteristics of the module are:

➔ Bidirectional Port,

➔ Programmable Direction for each bit

➔ Special features for modifying the port bits

Typical SOPC



Parallel Port Interface (1)

- 8 bits bidirectional Port,
  - Each pin can be specified as input or output
  - The direction is specified in **RegDir, (0 : input, 1 : output)**
  - The direction can be read back

- The state of the port at the pin level can be read in : **RegPin**

Parallel Port Interface (2)

- The state value is stored in a register:
  RegPort → Port Register
- 3 possible ways to modify register value :

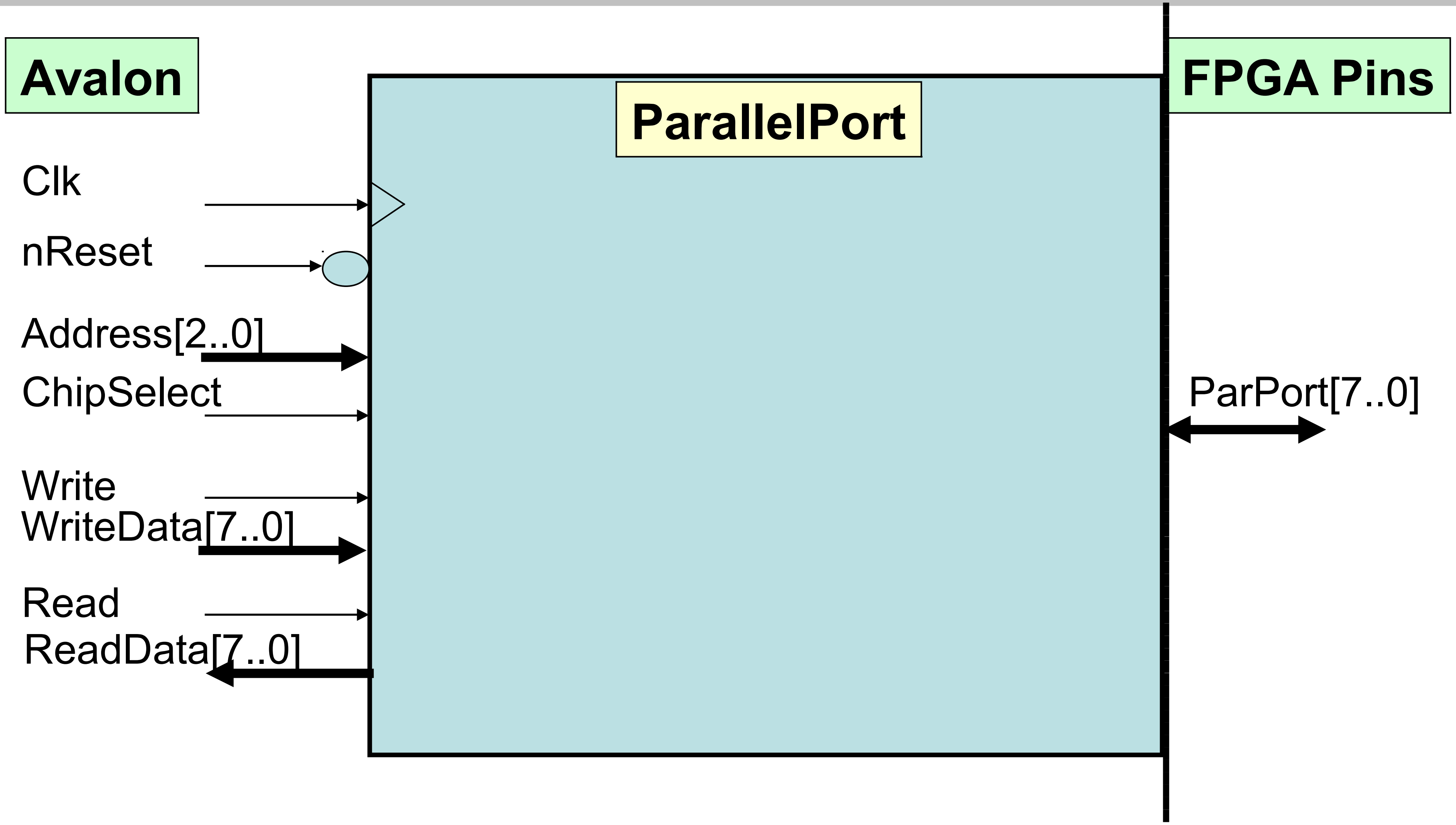
1. **RegPort** : Direct memorized value : '0' or '1'
2. **RegSet** : The bits specified at '1' level during the write cycle at this address, are saved as '1' in the register, the others bits are not changed
3. **RegClr** : The bits specified at '1' level during the write cycle at this address, are saved as '0' in the register, the others bits are not changed

- This register can be read back

Programmable interface design : Methodology

- Identify inputs-outputs of the interface (entity)
- Define a register model.
  - The register model is the interface between hardware and software.
  - Typically there are control, status and data registers.
  - For the software programmer the interface must remain as simple to use as possible.
  - Try to avoid unnecessary hardware complexity.
- Create your interface architecture:
  - From registers generate outputs.
  - From inputs write on registers.
  - Other registers may also store system state

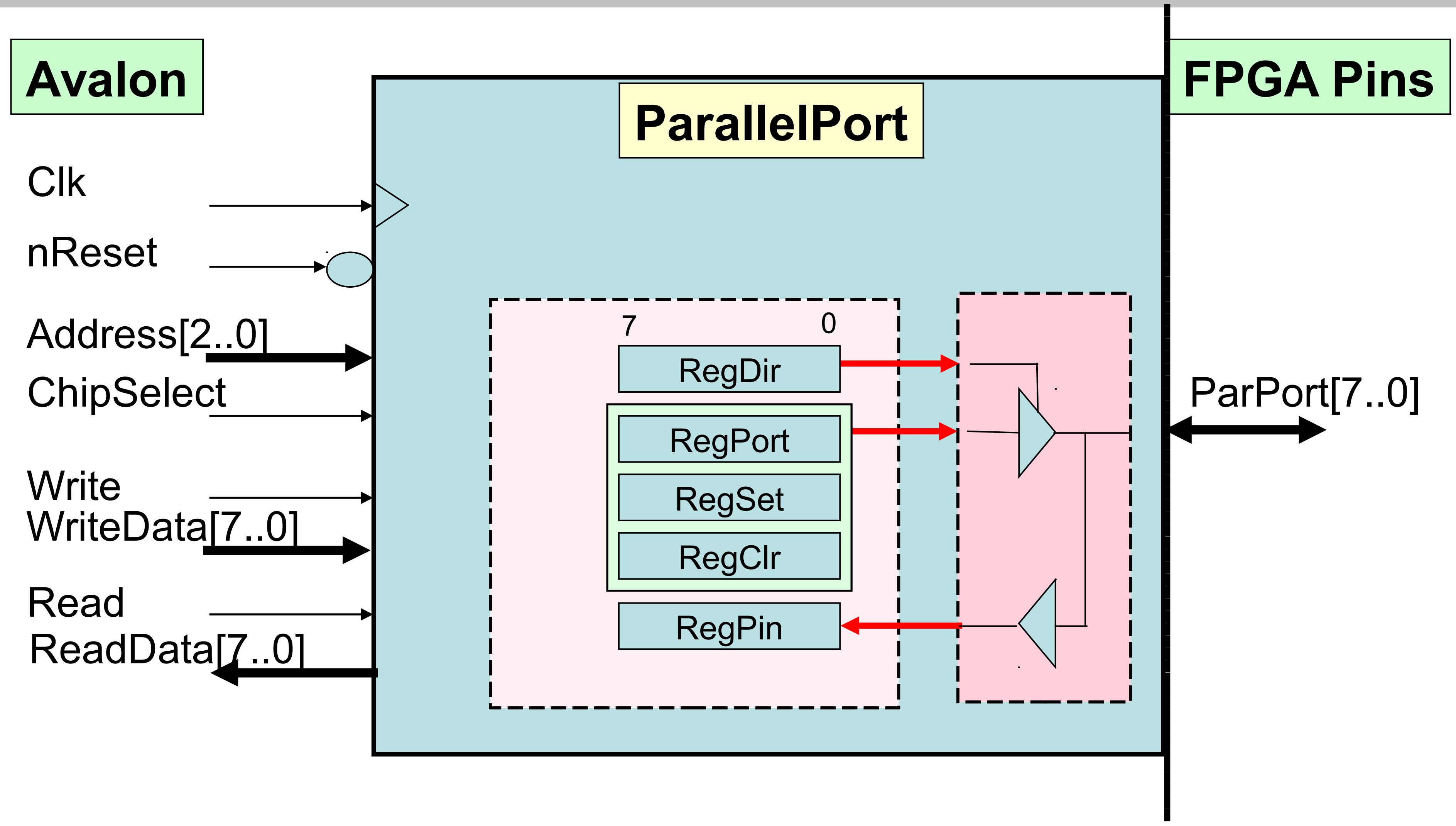
Parallel Port Module on Avalon : I/O



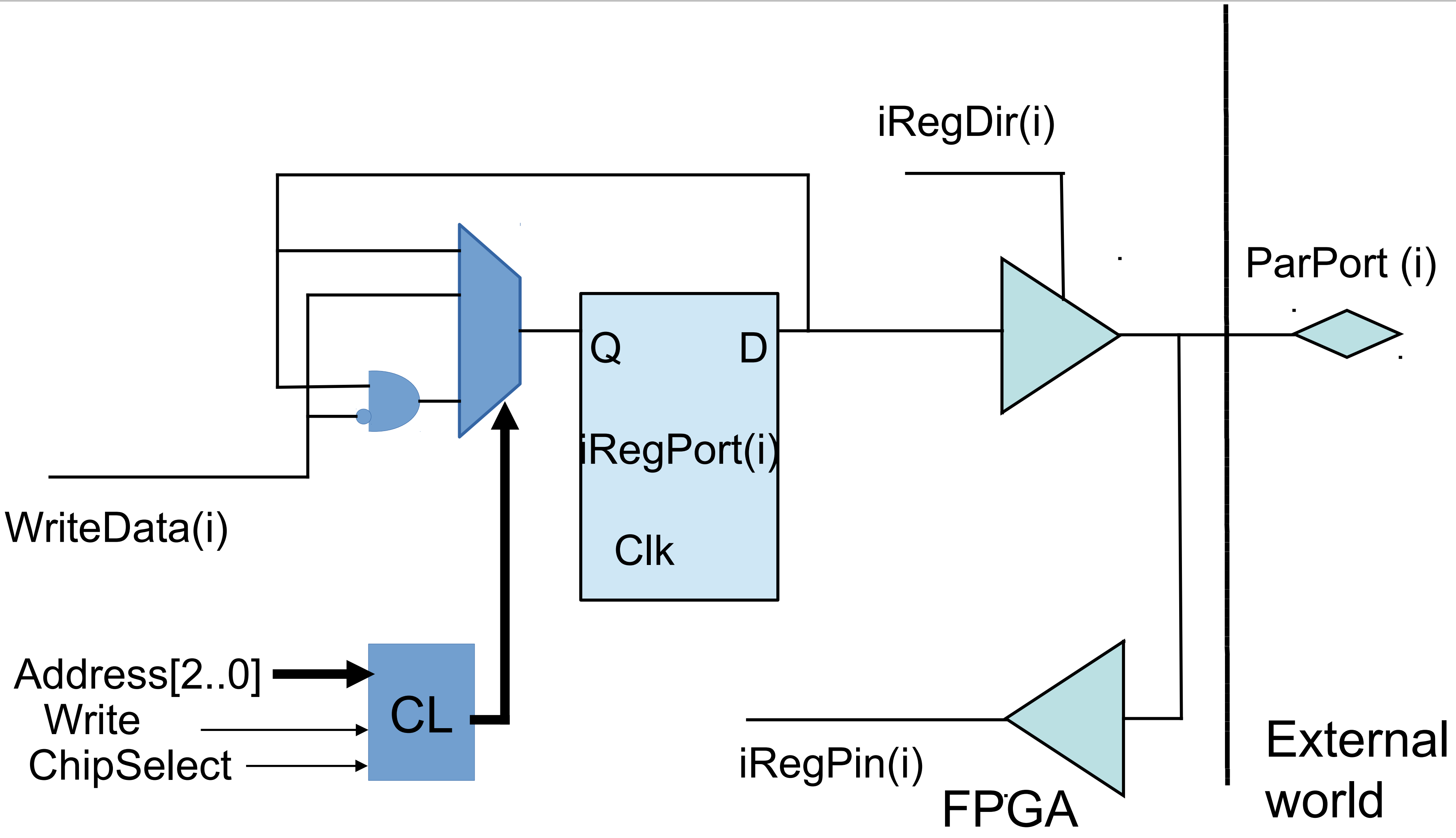
I/O Addresses in the module: register model

Adresses in the module	Write Registers	DataWrite [7..0]	Read Registers	DataRead [7..0]
0	RegDir	→ iRegDir	RegDir	iRegDir →
1	-	Don't care	RegPin	ParPort →
2	RegPort	→ iRegPort	RegPort	iRegPort →
3	RegSet	→ iRegPort	-	0x00
4	RegClr	→ iRegPort	-	0x00
5	-	Don't care	-	0x00
6	-	Don't care	-	0x00
7	-	Don't care	-	0x00

Parallel Port Module on Avalon



Parallel Port Module : from Registers to outputs



Entity

```
ENTITY ParallelPort IS
   PORT(
   -- Avalon interfaces signals
      Clk : IN    std_logic;
      nReset   : IN    std_logic;

      Address : IN    std_logic_vector (2 DOWNTO 0);
      ChipSelect  : IN    std_logic;

      Read: IN    std_logic;
      Write    : IN    std_logic;

      ReadData: OUT   std_logic_vector (7 DOWNTO 0);
      WriteData   : IN    std_logic_vector (7 DOWNTO 0);

   -- Parallel Port external interface
      ParPort  : INOUT std_logic_vector (7 DOWNTO 0)
   );
End ParallelPort;
```

Architecture: Internal signals

-- signals for register access

```
iRegDir   :  std_logic_vector (7 DOWNTO 0);
iRegPort: std_logic_vector (7 DOWNTO 0);
iRegPin   :  std_logic_vector (7 DOWNTO 0);
```

ParallelPort Architecture, registers access (write)

```
ARCHITECTURE comp OF ParallelPort IS
… SIGNAL ...
BEGIN
    process(Clk, nReset)
    begin
        if nReset = '0' then
          iRegDir <= (others => '0');        -- Input by default
          …..
        elsif rising_edge(Clk) then
            if ChipSelect = '1' and Write = '1' then -- Write cycle
                case Address(2 downto 0) is
                    when "000" => iRegDir <= WriteData ;
                    when "010" => iRegPort <= WriteData;
                    when "011" => iRegPort <= iRegPort OR WriteData;
                    when "100" => iRegPort <= iRegPort AND NOT WriteData;
                    when others => null;
                end case;
            end if;
        end if;
    end process pRegWr;
```

ParallelPort Architecture, registers access (read)

```
-- Read from registers with wait 0
ReadData  <=      iRegDir  when Address= "000" else
                  iRegPin  when Address= "001" else
                  iRegPort when Address= "010" else
                  "00000000";
```

Partially ok… because:

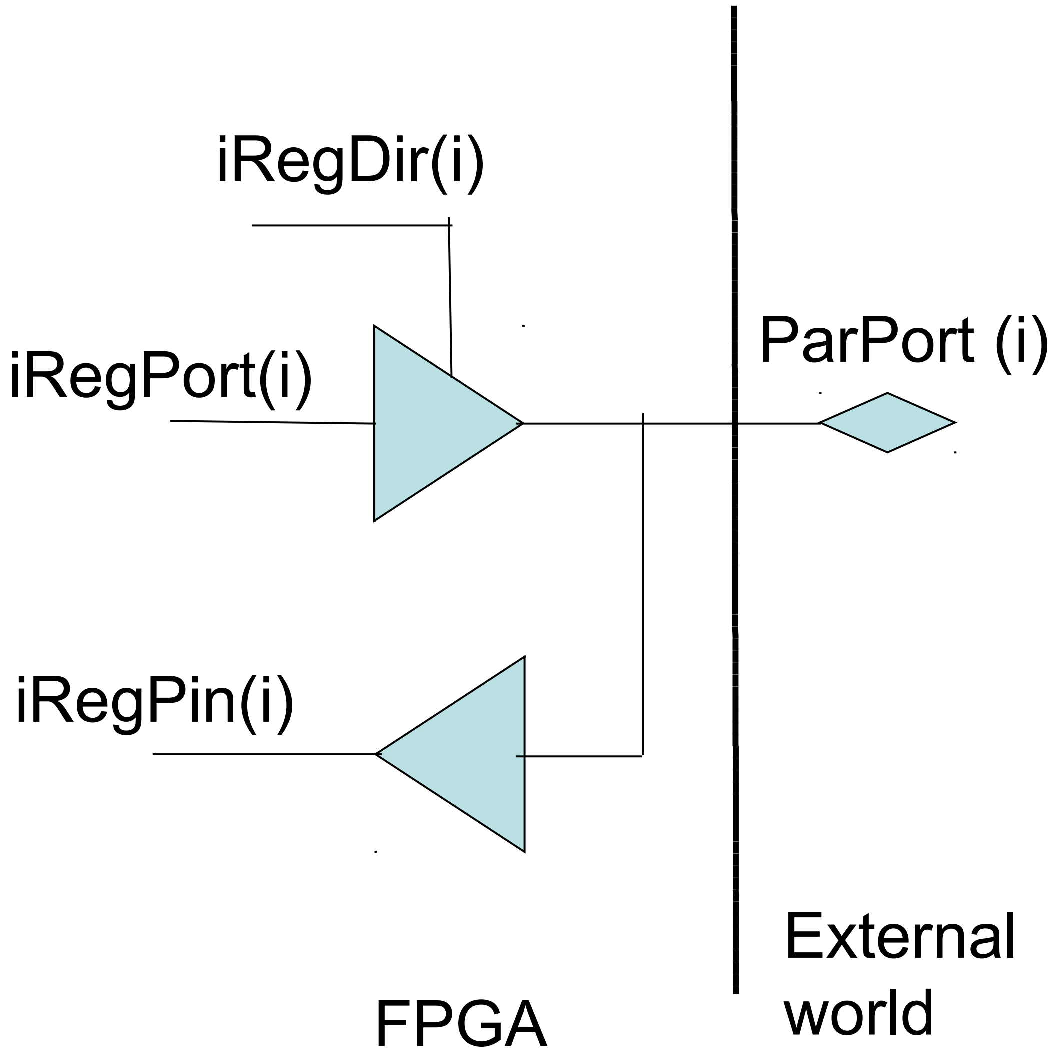
ParallelPort Architecture, registers access (read)

```
-- Read Process from registers with wait 1
pRegRd:
    process(Clk)
    begin
       if rising_edge(Clk) then
          ReadData <= (others => '0');          -- default value
          if ChipSelect = '1' and Read = '1' then    -- Read cycle
              case Address(2 downto 0) is
                when "000" => ReadData <= iRegDir ;
                when "001" => ReadData <= iRegPin;
                when "010" => ReadData <= iRegPort;
                when others => null;
              end case;
            end if;
       end if;
    end process pRegRd;
```

ParallelPort Architecture, external interface

```
 -- Parallel Port output value
 pPort:
   process(iRegDir, iRegPort)
   begin
   for i in 0 to 7 loop
    if iRegDir(i) = '1' then
     ParPort(i) <= iRegPort(i);
    else
     ParPort(i) <= 'Z';
    end if;
   end loop;
   end process pPort;

 -- Parallel Port Input value
   iRegPin <= ParPort;
END comp;
```


iRegDir(i)
iRegPort(i)
ParPort (i)
iRegPin(i)
FPGA
External world