

# Lecture 3

## Design of Embedded Hardware and Firmware

Programmable interface design – GPIO  
PIO - example

*TSM\_EmbHardw*  
*Mar. 08, 2016*

# Agenda

## ▶ Intro

### ▶ Specification

## ▶ PIO Design

- ▶ Design Methodology
- ▶ The Address Mapping

Function

- ▶ The Module Interface
- ▶ The Module



Intro

Specs

PIO Design

Method

Entity

AddrMap

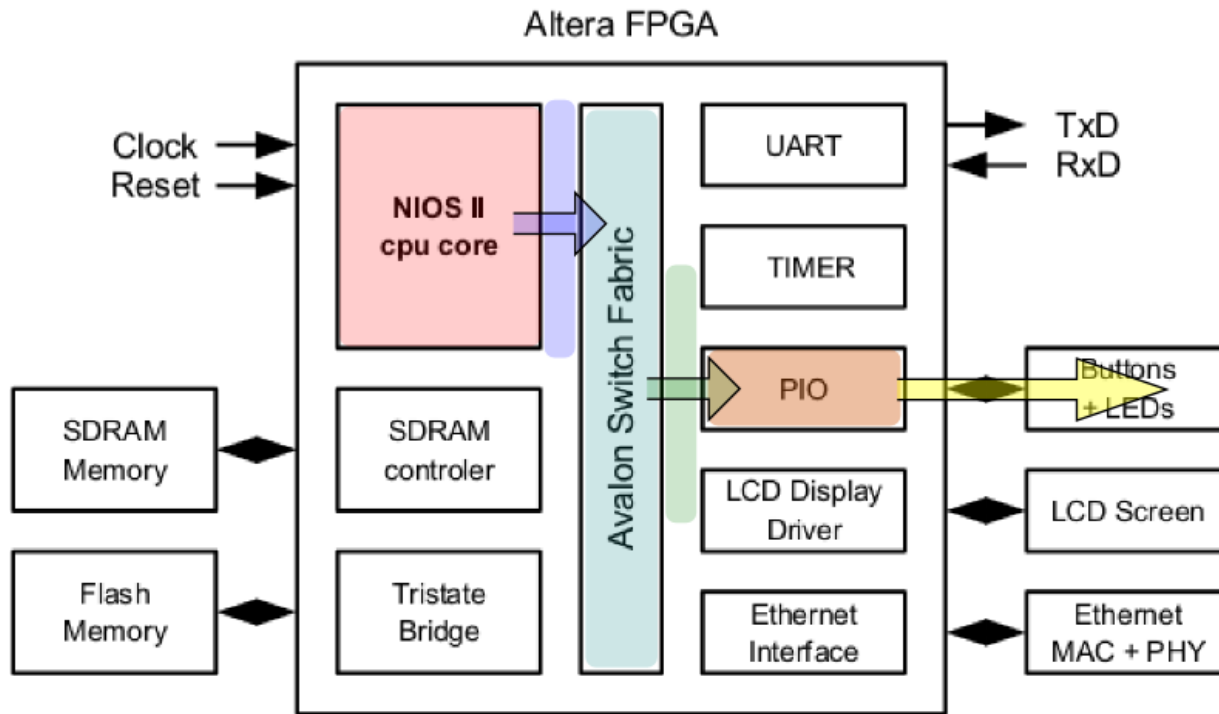
Arch

# The Goals

- ▶ 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 communication capability
  - ▶ Programmable direction on bit level (from SW)
  - ▶ Special features for modifying the port bits (value)



# Top View of Example SoPC



## Intro

Specs

## PIO Design

Method

Entity

AddrMap

Arch



# Parallel Port Interface

- ▶ The Port to external components (8 bits bidirectional)
  - ▶ 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`
- ▶ The state value is stored in a register:
  - ▶ `RegPort` → Port Register



# Parallel Port Interface

- Possibilities to modify register value are listed next:

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

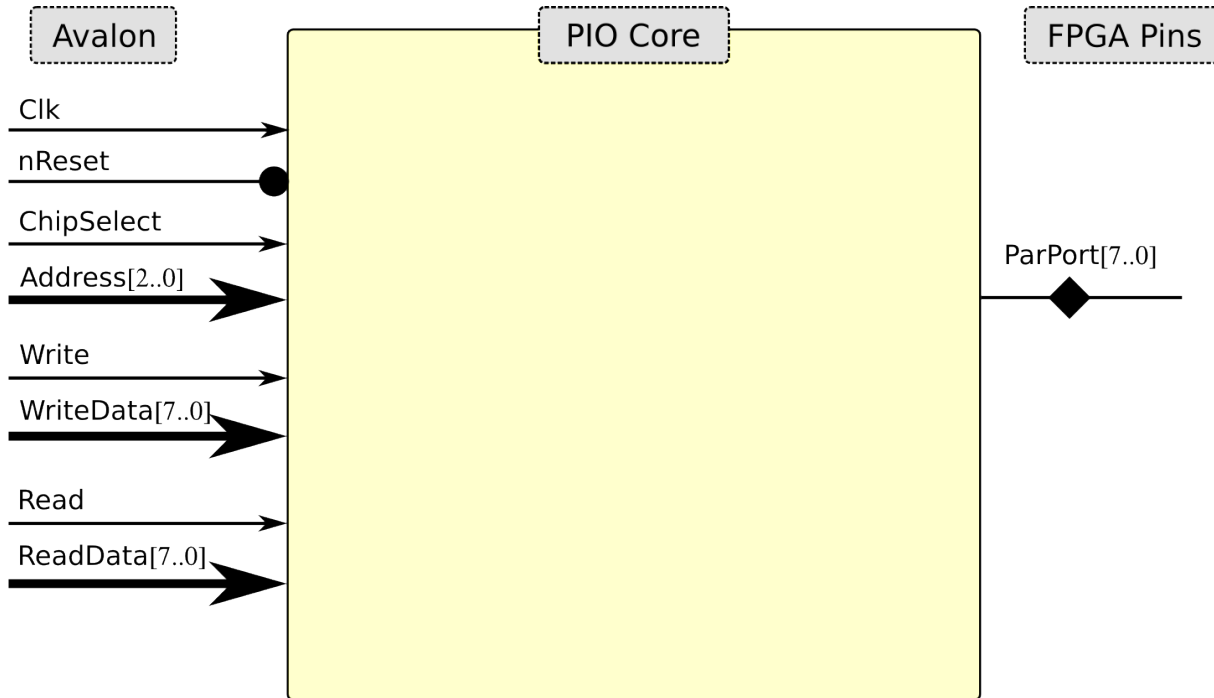
This register can be read back



# Programmable Interface Design

- ▶ Identify inputs-outputs of the interface (VHDL `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 derive outputs
  - ▶ From inputs write on registers
  - ▶ Other registers may also store system state (decoding of states)

# The Entity



Based on the block interface we derive entity signals.

1

<sup>1</sup> Use std\_logic signal type, only.





# VHDL Description

## File : simplePIO-entity.vhdl

```
library ieee;
use ieee.std_logic_1164.all;

entity SimplePIO is
  port (
    -- Avalon interfaces signals
    Clk_CI      : in    std_logic;
    Reset_RLI   : in    std_logic;
    Address_DI  : in    std_logic_vector (2 downto 0);
    ChipSelect_SI : in    std_logic;
    Read_SI     : in    std_logic;
    Write_SI    : in    std_logic;
    ReadData_DO : out   std_logic_vector (7 downto 0);
    WriteData_DI : in    std_logic_vector (7 downto 0);
    -- Parallel Port external interface
    ParPort_DIO : inout std_logic_vector (7 downto 0)
  );
end entity SimplePIO;
```

- ▶ A good naming convention is the one used at ETH. It introduces a suffix convention as for specify signal property more detailed. Look it up  
<https://www.dz.ee.ethz.ch/de/information/hdl-hilfe/vhdl-namenskonvention.html>

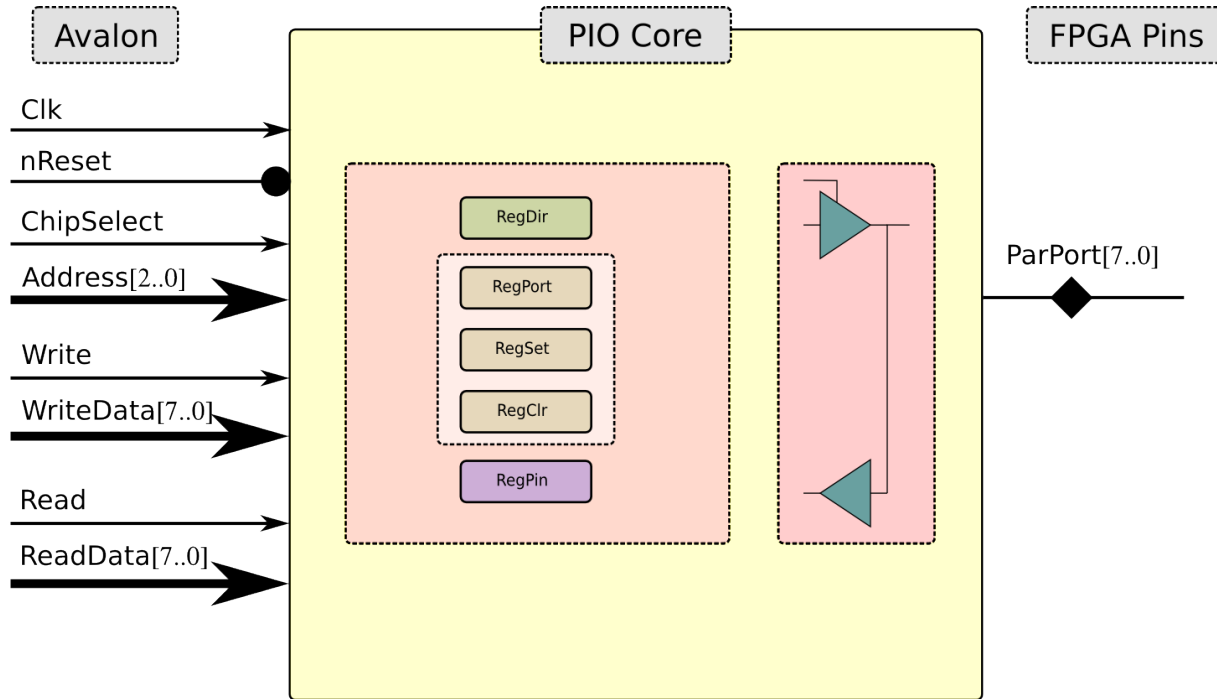
# The Register Model



Addr	Write Register		Read Register	
	Name	[7..0]	Name	[7..0]
0	RegDir	→RegDir	RegDir	RegDir→
1	—	Don't care	RegPin	ParPort→
2	RegPort	→RegPort	RegPort	RegPort→
3	RegSet	→RegPort	—	0x00
4	RegClr	→RegPort	—	0x00
5	—	Don't care	—	0x00
6	—	Don't care	—	0x00
7	—	Don't care	—	0x00

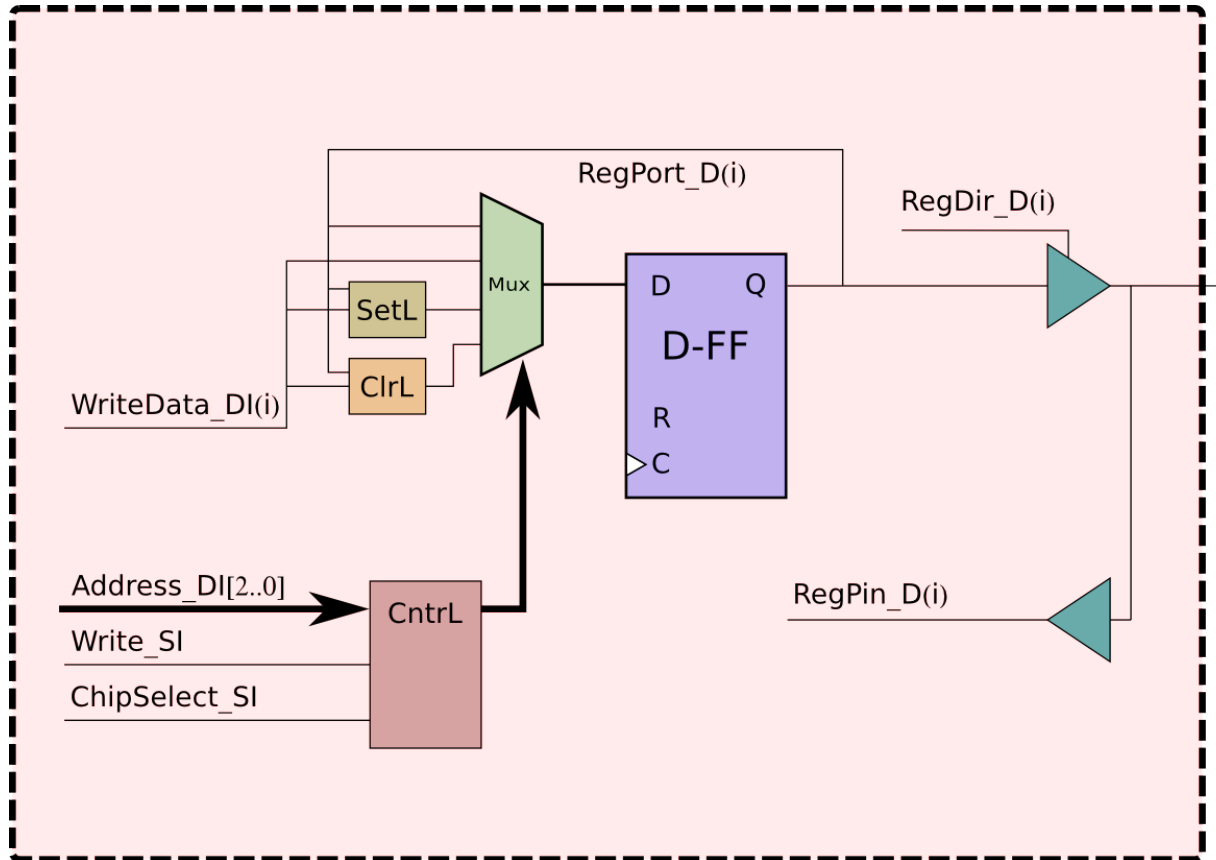
- Calculate the width of address bus by :  
bit width  $\rightarrow \text{ceil}(\frac{\log(\text{RegNo})}{\log(2)})$

# Parallel Port Module on Avalon



Based on the address mapping we develop the functional behavior

# From Registers to Outputs



Develop the FSM-D interface controller.

# The Architecture

## Internal signals



### File : simplePIO-archNoWait.vhdl

```
architecture noWait of simplePIO is
    signal RegDir_D  : std_logic_vector (7 DOWNTO 0);
    signal RegPort_D : std_logic_vector (7 DOWNTO 0);
    signal RegPin_D  : std_logic_vector (7 DOWNTO 0);
begin
```

Intro

Specs

PIO Design

Method

Entity

AddrMap

Arch

# The Architecture

## Registers access (write)



### File : simplePIO-archNoWait.vhdl

```
pRegWr : process(Clk_CI, Reset_RLI)
begin
  if (Reset_RLI = '0') then
    -- Input by default
    RegDir_D <= (others => '0');
    RegPort_D <= (others => '0');
  elsif rising_edge(Clk_CI) then
    if ChipSelect_SI = '1' and Write_SI = '1' then
      -- Write cycle
      case Address_DI(2 downto 0) is
        when "000" => RegDir_D <= WriteData_DI;
        when "010" => RegPort_D <= WriteData_DI;
        when "011" => RegPort_D <= RegPort_D OR WriteData_DI;
        when "100" => RegPort_D <= RegPort_D AND NOT WriteData_DI;
        when others => null;
      end case;
    end if;
  end if;
end process pRegWr;
```

# The Architecture

## Registers access (read) – 0 wait

### File : simplePIO-archNoWait.vhdl

```
-- Read from registers with wait 0
ReadData_DO <= RegDir_D  when Address_DI = "000" else
                RegPin_D   when Address_DI = "001" else
                RegPort_D  when Address_DI = "010" else
                (others => '0');
```



# The Architecture

## Registers access (read) – 1 wait

### File : simplePIO-arch1Wait.vhdl

```
-- Read Process from registers with wait 1
pRegRd : process(Clk_CI)
begin
  if rising_edge(Clk_CI) then
    ReadData_DO <= (others => '0');
    if ChipSelect_SI = '1' and Read_SI = '1' then
      case Address_DI(2 downto 0) is
        when "000" => ReadData_DO <= RegDir_D;
        when "001" => ReadData_DO <= RegPin_D;
        when "010" => ReadData_DO <= RegPort_D;
        when others => null;
      end case;
    end if;
  end if;
end process pRegRd;
```



# The Architecture

## External interface

File :  
`simplePIO-archNoWait.vhdl`

```
-- Parallel Port output value
pPort : process(RegDir_D, RegPort_D)
begin
  for idx in 0 to 7 loop
    if RegDir_D(idx) = '1' then
      ParPort_DIO(idx) <= RegPort_D(idx);
    else
      ParPort_DIO(idx) <= 'Z';
    end if;
  end loop;
end process pPort;

-- Parallel Port Input value
RegPin_D <= ParPort_DIO;
```

