SYSTEM-VERILOG

Méthodologie de conception et langages de description matérielle

VHDL

RISC - V

Support de cours

Master SECIL

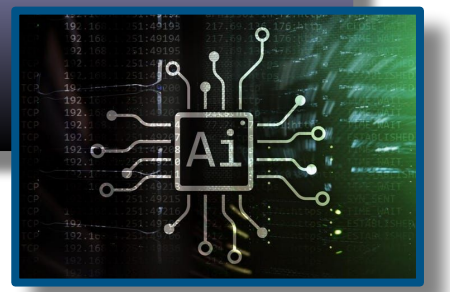
François Thiebolt
Daniela Dragomirescu

FPGA

Q1 2025

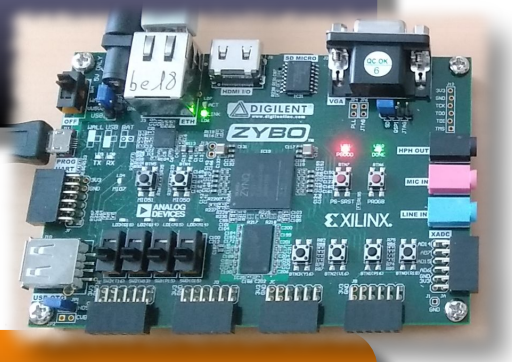
UE-VHDL | Plan

Part I – VHDL language



Part II – Synthesis for FPGA targets

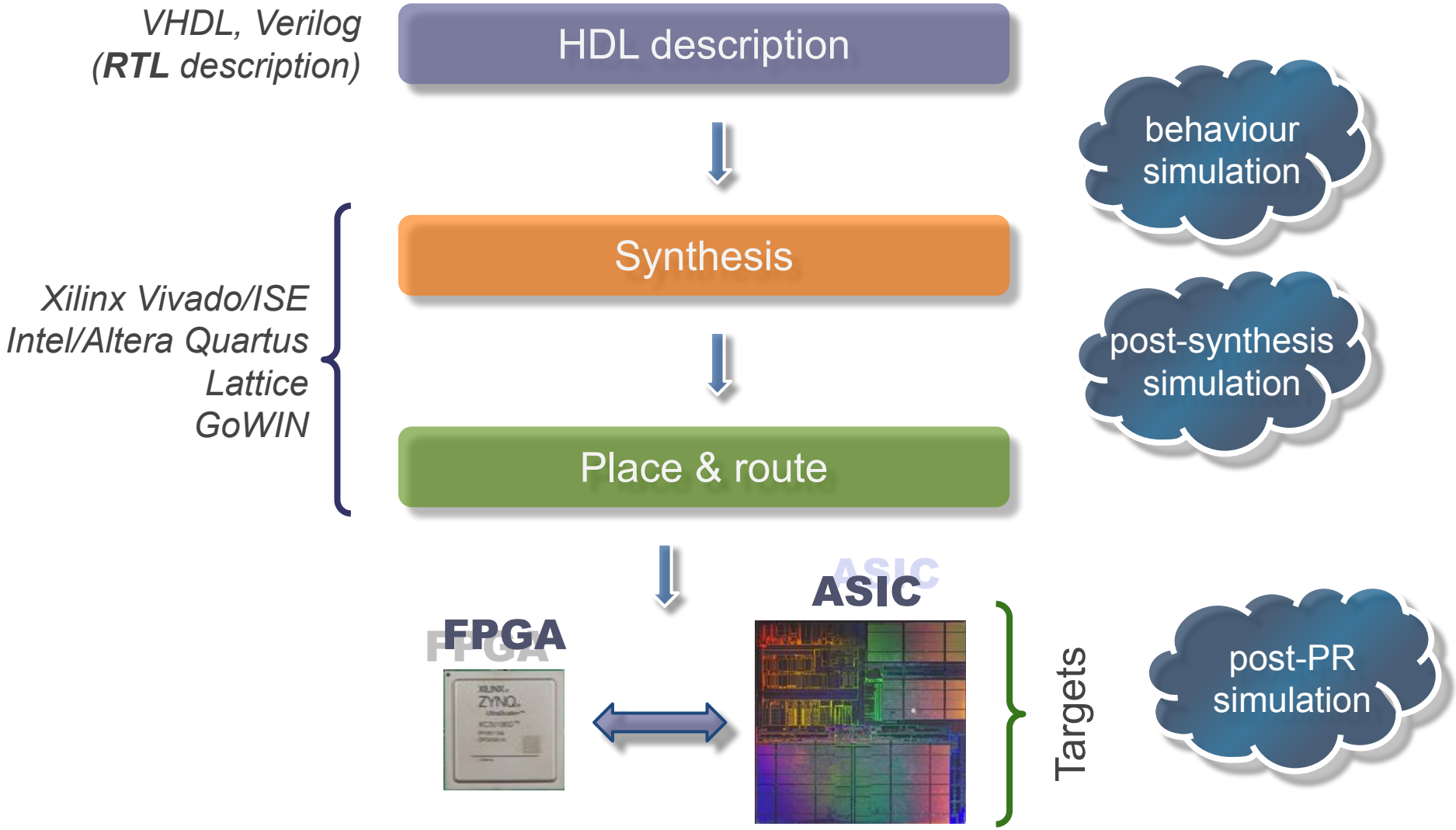
Part IIa – [Advanced] Xilinx Zynq architecture



Part III – Electronics for ~~dummies~~ all :)

Foreword

- VHDL | The big picture



Foreword

- … one step beyond

XILINX®

VIVADO

export Hardware

PetaLinux

FSBL + uboot

linux kernel with Zynq support + DTB

armbian universal operating system

ARMBIAN ARM rootFS

Linux on Zynq with HW accelerated features

Foreword

For those who can't stand waiting for things to come …

Sipeed Tang Nano 4K

FPGA GW1NSR-LV4C
- 4608 LUTs
- 3456 FFs
- 180K BSram
- 2xPLL + 2xDLL
- + embedded cortex M3 !

BOARD
- 4MB Flash
- camera support
- hdmi output

GoWin toolchain

ghdl

gtkwave

Additional online resources like
https://edaplayground.com

https://tangnano.sipeed.com/en/

What's all this VHDL stuff, Anyhow* ?

Part I – VHDL language

- Introduction,
- VHDL basics,
- Exercises.

**Robert Allen Pease [1940 - 2011] was a famous TI's analog engineer, his favourite programming language was ... solder!*

Introduction

Rappels :

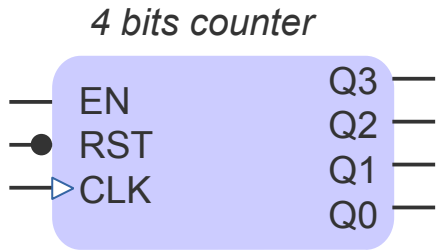
- Circuits logiques combinatoires

  Leur sortie est déterminé par la valeur courante de l'entrée:
  portes logiques, multiplexeurs, démultiplexeurs, décodeurs, circuits arithmétiques

- Circuits logiques séquentielles

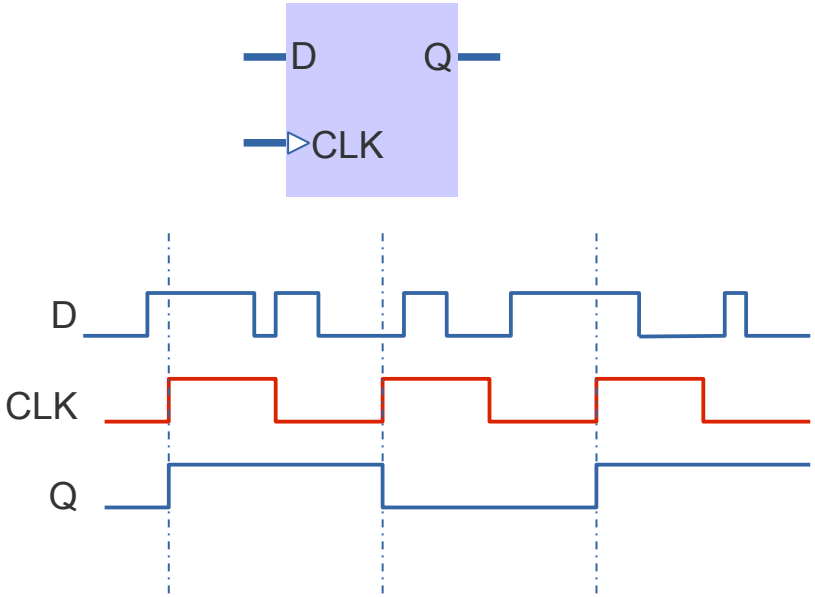
  Leur sortie est partiellement déterminée par l'évolution de l'entrée. La réponse du circuit dépend de l'histoire plus ou moins récente du fonctionnement du circuit.
  Fonction **MÉMOIRE** interne:
  Bascule D, registres, mémoires RAM, compteurs

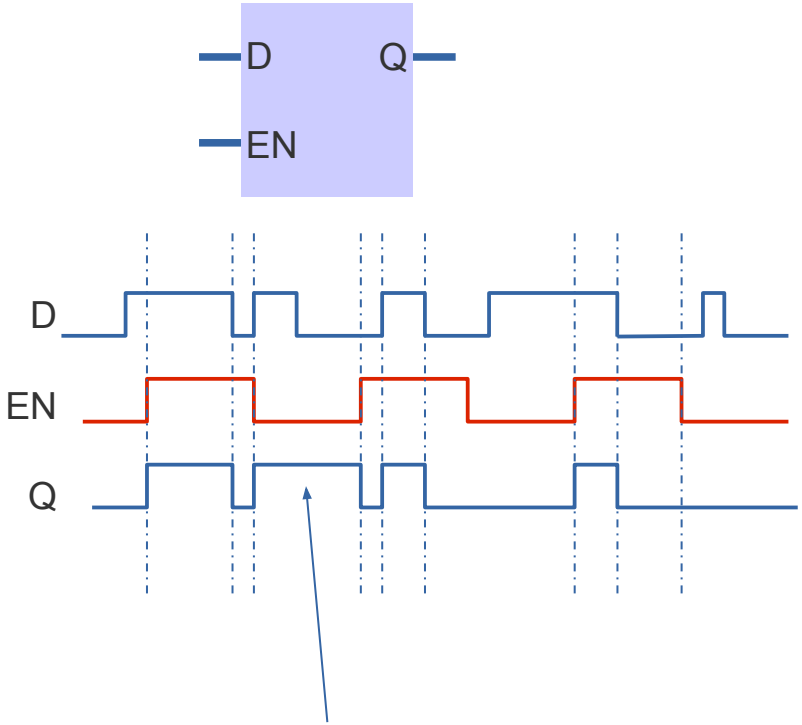
- Exercice : réalisez un compteur 4 bits en utilisant des bascules D en logique séquentielle synchrone



Introduction

Rappels : D flip-flop vs D latch



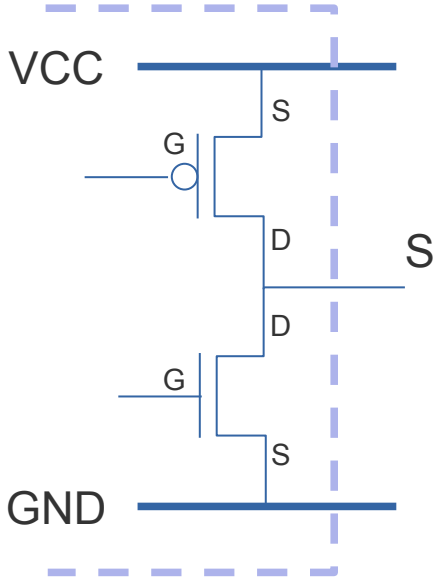


Unsure about the output …
guess why ??

Introduction

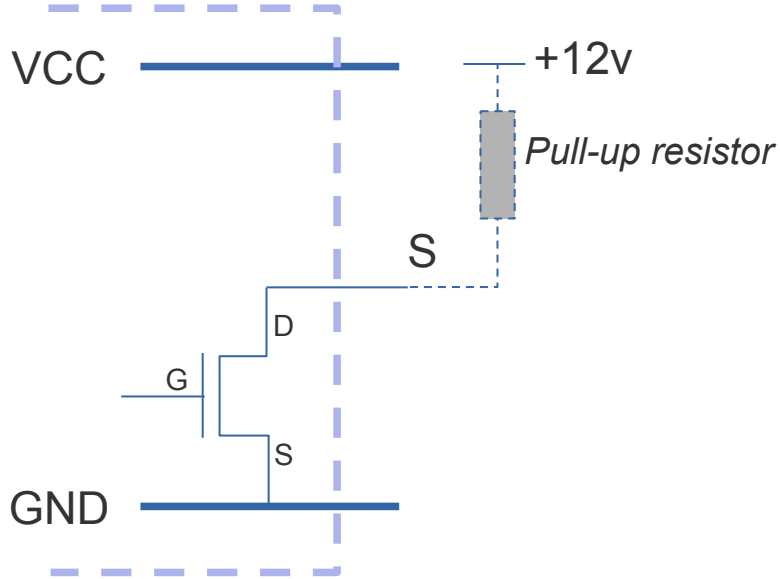
Rappels :

- Totem-pole (*push-pull*) output



Note: same design with bipolar transistors

- Open-Drain / Open-Collector output



Note: same design with bipolar transistors

Introduction

Intel's XEON family starts to feature FPGA :)

- Évolution des puces

Chips	Year	Frequency	Transistors	Tech.
Intel 8086	1978	8 MHz	29K	3.2µm
Intel 80486 DX	1989	25 MHz	1.2M	1µm
Pentium III XEON	1999	700 MHz	28M	180nm
Pentium IV	2001	1.7 GHz	42M	180nm
Core i7 Westmere	2010	3.2 GHz	1.17B	32nm
nVidia GT400	2010		3B	40nm
nVidia Tesla V100	2017		21.1B	12nm
ARM Cortex A-76 (Kirin 990)	2019	2.86 GHz	10.3B	7nm
Apple M1	2020	3.1 GHz	16B	5nm
Apple M2	2023	3.5 GHz	20B	5nm
SiFive P670 **RISC-V** (IP)	2023	> 3GHz		5nm

2018

Introduction

- Accroissement de la complexité des puces:

Impose une évolution des méthodes des conception:

→ *la demande viendra du département de la défense des États-Unis en 1980:*

ADA pour le logiciel, VHDL pour le matériel.

Apparition des langages HDL - Hardware Description Language

Verilog

VHDL

Même principe, syntaxe différente.

Sont des langages de niveau Register Transfert Level

Ces langages 'décrivent' le matériel, ils n'exécutent rien!

Travail avec des cellules standard - briques de base appartenant à une bibliothèque spécifique à chaque fondeur de circuits intégrés

→ Standard cells - design kit

Introduction

- VHDL: pro

VHDL est standardisé - standard IEEE

pérennité assurée par la norme
conception modulaire et hiérarchique

VHDL est un langage moderne, puissant et général

haute modularité

sécurité d'emploi

fiabilité
notion de temps bien définie

unités de compilation séparées

typage fort

généricité

En utilisant VHDL on minimise le risque d'erreur sur le circuit intégré en silicium - on diminue le coût de production

- VHDL: cons

langage de simulation  tout n'est pas synthétisable !

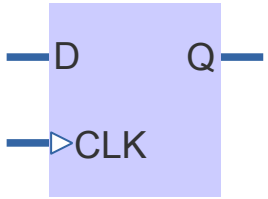
Introduction

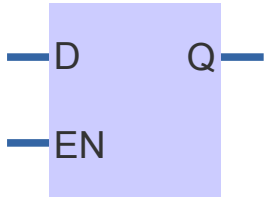
- Standardization
  - IEEE VHDL'87 - 1987
  - IEEE VHDL'93 - 1993
  - IEEE VHDL'03 - 2003
  - IEEE VHDL'08 - 2008

Never forget the overall objective: to produce chips through the use of synthesis tools!

Quick start

- VHDL: D flip-flop vs D latch



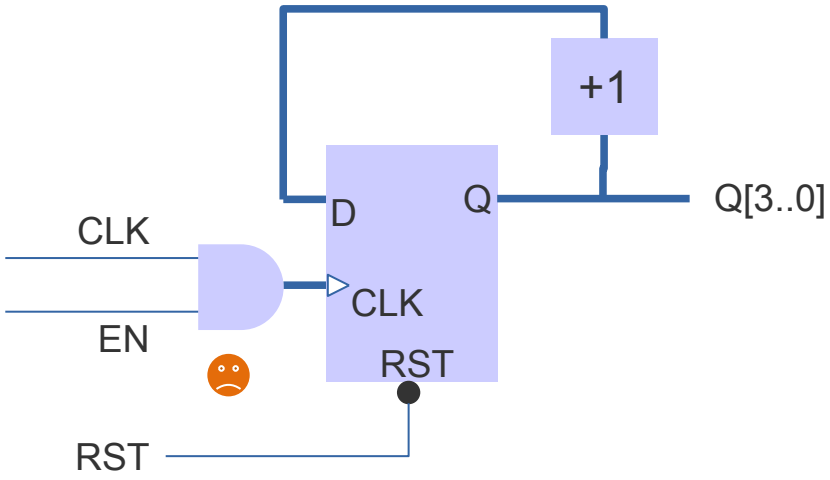



```
process( CLK )
begin
  if CLK'event and CLK='1' then
    Q <= D;
  end if;
end process;
```

```
process( D, EN )
begin
  if EN='1' then
    Q <= D;
  end if;
end process;
```

Quick start

- VHDL: 4 bits counter



4 bits counter

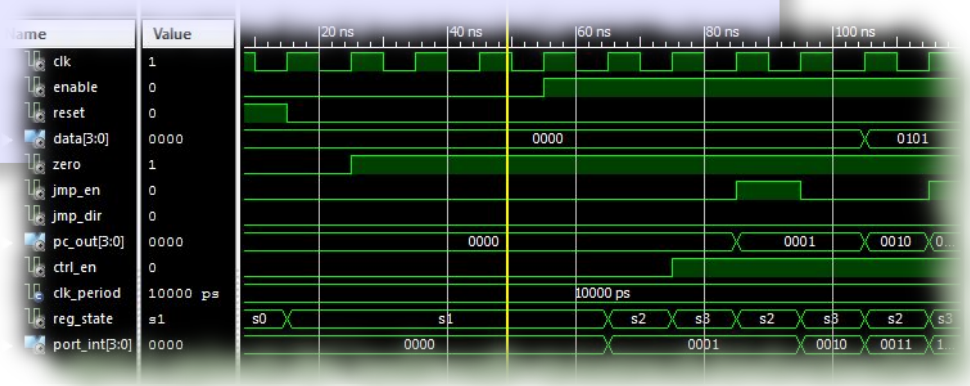

```
process( RST,CLK )
  variable cpt:std_logic_vector(3 downto 0);
begin
  if RST='0' then
    cpt := (others=>'0');
  elsif CLK'event and CLK='1'  and EN='1' then
    cpt := cpt + '1';
  end if;
end process;
```

Some adjustment required … we'll discover later.

Plan

Part I – VHDL language

- Introduction,
- VHDL basics,
- Exercises.



VHDL basics

Language features

- Packages

  Along with package_body will hold all of your **types**, **constants**, **functions** and **procedures** definitions.

- Libraries

  IEEE_std_logic_1164 is a compiled package provided as a system library.
  Your compiled stuff will go in your **WORK** library.

- Entity + Architecture of components

  For each of your components: **entity** + **architecture** descriptions.

- Configurations

  For each instantiated component, you can specify an architecture;
  e.g: `behaviour, synthesized, routed.`

- Tests architecture

  Yes, it's like a component with an empty entity (usually) along with instantiated components you want to test in its architecture.

VHDL basics

- Libraries

  Langage modulaire - unités petites et hiérarchisées

  Unités de conception - peuvent être compilées séparément

  Description VHDL correcte - bibliothèque de travail - WORK

  portabilité

  Utilitaires ou modèles généraux - bibliothèques de ressources

  facilite le travail en équipe

  Bibliothèques : IEEE , STD

  package IEEE.std_logic_1164 et **package** IEEE.std_logic_arith

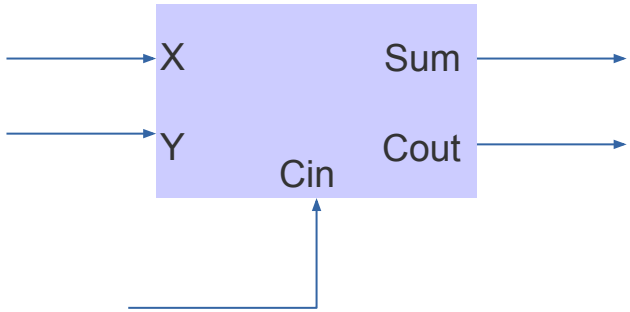
  package STD.textio et **package** STD.STANDARD

VHDL basics

- Unités de conception: entity

Entité - modèle VHDL

Vue externe

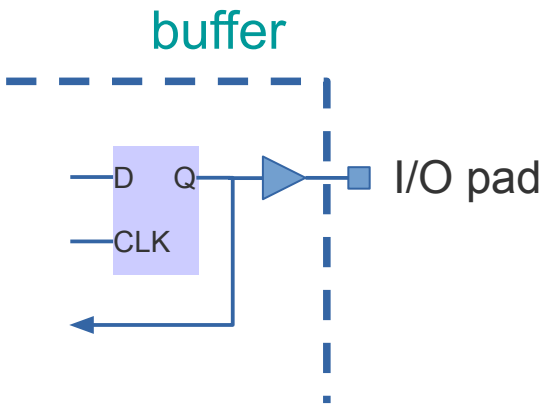


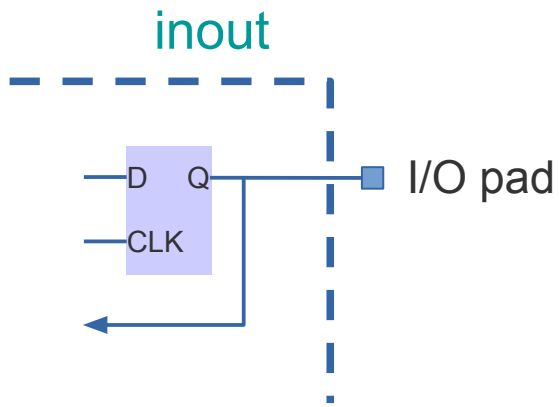

```
entity myadder is
        port (
                X, Y, Cin:      in std_logic;
                Sum, Cout:      out std_logic  );
end myadder;
```

- ports modes

Formal def. vs Connected object	Mode IN	Mode OUT	Mode INOUT	Mode BUFFER
Port de mode **IN**	☑			
Port de mode **OUT**		☑		
Port de mode **INOUT**	☑	☑	☑	
Port de mode **BUFFER**	☑			☑
Local signal	☑	☑	☑	☑
open keyword (not connected)		☑	☑	☑

- ports modes: **BUFFER** vs **INOUT**





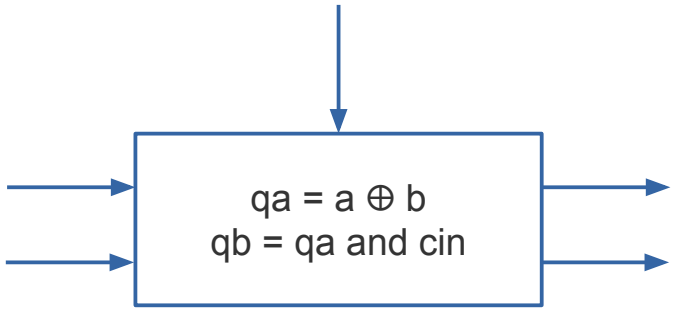
Output pin is buffered: you can **read what you wrote** but you can't read the value of the pin.

The value on the pin may differ from what you wrote !

VHDL basics

- Unités de conception: architecture

Vue interne d'une entité - ARCHITECTURE



Style de description de l 'architecture :

- ➢ Description structurelle : interconnexion de composants
- ➢ Description flot de données : comportement sous forme d'équations
- ➢ Description comportementale : comportement sous forme algorithmique

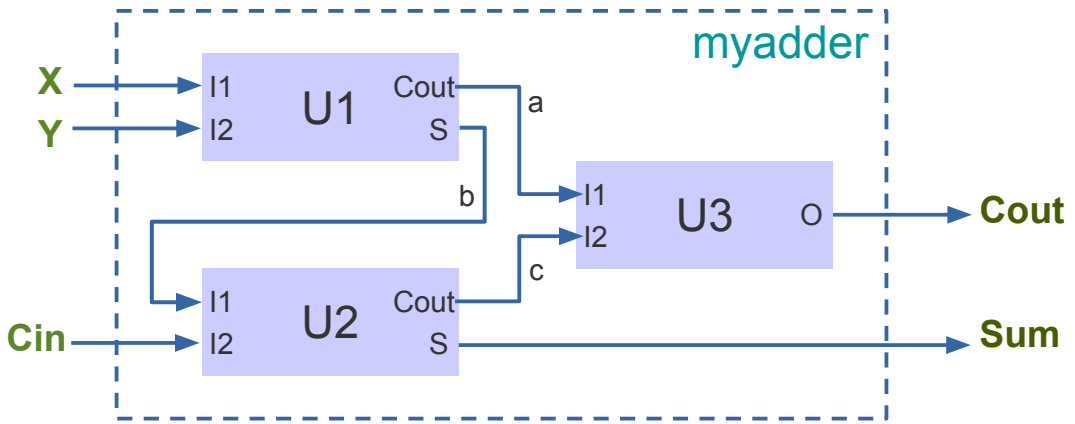
On peut combiner les différents styles de descriptions dans une même architecture.

*La synthèse produit une description **structurelle**.*

VHDL basics

- Architecture structurelle de l'additionneur

```
architecture vue_structurelle of myadder is
    signal a,b,c : std_logic;
begin;
    U1:entity work.adder(behaviour)
            port map(X,Y,a,b);
    U2:entity work.adder(behaviour)
            port map(b,Cin,c,Sum);
    U3:entity work.or_gate(behaviour)
            port map(a,c,Cout);
end vue_structurelle;
```



VHDL basics

- Architecture flot de données de l'additionneur

$S = X \oplus Y$

$Somme = S \oplus Cin$

$Cout = X.Y + S.Cin$

Equations booléennes de l'additionneur

```
architecture data_flow of myadder is
    signal S : std_logic;
begin;
    Somme <= S xor Cin after 10 ns;
    S <= X xor Y after 10 ns;
    Cout <= (X and Y) or (S and Cin) after 20 ns;
end data_flow;
```

VHDL basics

- Architecture comportementale de l'additionneur
  décrite par une table de vérité

```
architecture vue_comportementale of myadder is
begin;
    process -- instruction concurrente
            variable N: integer;
            constant sum_vector: std_logic_vector(3 downto 0):= "1010";
            constant carry_vector: std_logic_vector(3 downto 0):= "1100";
    begin
            N:=0;
            if X= '1' then N:=N+1; end if;
            if Y= '1' then N:=N+1; end if;
            if Cin=1 then N:=N+1; end if;
            Somme <= sum_vector(N) after 20 ns;
            Cout <= carry_vector(N) after 30 ns;
            wait on X, Y, Cin;
    end process;
end vue_comportementale;
```

X	Y	Cin	Sum	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

- Configurations

  Donne la correspondance entre le composant et le modèle dont il est instancié

  Permet de faire la liaison entre des composants utilisés dans une architecture et leur réalisation effective

Ex: Soit une entité « trois_registres » avec une architecture « beh » ayant 3 registres R1, R2 et R3

```
use work.trois_registres;
configuration alpha of trois_registres is
    for beh
        for R1,R2 : registre_8bit use entity work.registre_8bit(arch);
        for R3 : registre_8bit use entity work.registre_8bit(decalage);
    end for;
end alpha;
```

alternative to apply the same architecture for all instantiated components:

```
FOR ALL : registre_8bit use entity work.registre_8bit(arch);
```

VHDL basics

- Configuration example for our adder

```
library work;
entity myadder is
      port (X, Y, Cin : in std_logic;
            Sum, Cout : out std_logic);
end myadder;
architecture vue_structurelle of myadder is
   for all : adder use entity work.adder(beh);
   for all : or_gate use entity work.or_gate(struct);
signal a,b,c : std_logic;
begin
   U1:entity work.adder(behaviour)
                port map(X,Y,a,b);
   U2:entity work.adder(behaviour)
                port map(b,Cin,c,Sum);
   U3:entity work.or_gate(behaviour)
                port map(a,c,Cout);
end vue_structurelle;
```

```
library my_lib;

architecture …
      for all : adder use entity my_lib.adder(beh);
begin
```

VHDL basics

- Packages

Ensemble des algorithmes, sous-programmes, nouveau types et sous-types, des objets: signaux et constantes (pas de variables)

Un ou plusieurs paquetage dans la même bibliothèque

2 unités de conception :

Spécification d'un paquetage - vue externe

présente tout ce qu'exporte le paquetage (algorithmes, objets, types)

Corps du paquetage - vue interne - *optionnel*

contient la description des algorithmes, déclarations locales des types, objets

Pour avoir accès à un paquetage il faut le référencer par une clause **use**

```
library IEEE;
    use IEEE.std_logic_1164.all ;
library my_lib;
    use my_lib.mon_paquetage.all;
```

- Packages

Spécification du paquetage :

```
package SIMPLE is

   constant TAILLE_MAX: integer :=1024;

   type mon_integer is INTEGER range 0 to TAILLE_MAX;

   signal addition_10bits : mon_integer ;

   function MIN(A,B:INTEGER) return INTEGER;
   function MAX(A,B:INTEGER) return INTEGER;

end SIMPLE;
```

Les déclarations faites dans la spécification du paquetage sont connues dans le corps du paquetage.

- Packages - body

Corps du paquetage

```
package body SIMPLE is
    function MIN(A,B:INTEGER) return INTEGER is
    begin
            if A<B then return A;
            else return B;
            end if;
    end MIN;
    function MAX(A,B:INTEGER) return INTEGER is
    begin
            if B<A then return A;
            else return B;
            end if;
    end MAX;
end SIMPLE;
```

VHDL basics

- Concurrent and Sequential domains

La description d'un système matériel est naturellement concurrente.

Le fond d'une description VHDL est concurrent.

Les 2 domaines cohabitent en VHDL

Domaine concurrent

- La zone de déclarations des entités
- La zone de déclarations des architectures

Domaine séquentiel

- La zone de déclarations de processus
- Le corps du paquetage

VHDL basics

- Tests

Pour vérifier (simuler) le comportement du composant décrit en VHDL il faut le tester

Le programme de test - un programme VHDL, avec la même structure (entity, architecture)

Applique les stimuli en entrée et à vous de contrôler les sorties du composant en test (e.g assertions)

Si possible - test exhaustif

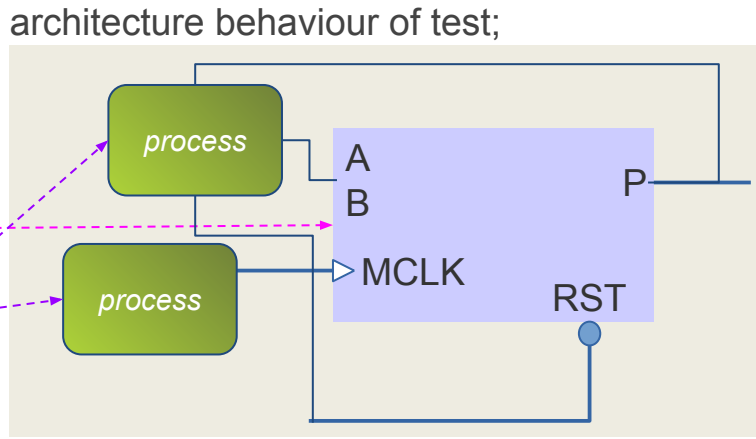
Difficile à mettre en œuvre pour des systèmes complexes

Preuve formelle

Create a test component **without I/O**

Instantiation of **your** component

Processes to generate a clock, inputs stimulus, asserts on outputs …



Exemple: test de l'additionneur

```
entity test_adder is
end test_adder;
architecture bench of test_adder is
    For all : adder use entity work.adder(vue_structurelle);

    signal data1, data2, data3 : std_logic;
    signal data_out, carry_out : std_logic;
begin
        additionneur: entity work.adder PORT MAP (data1,data2, data3,data_out,carry_out);
        data1 <= '0', '1' after 30 ns;
        data2 <= '1', '0' after 50 ns, '1' after 60 ns;
        data3 <= '0', '1' after 12 ns;
end bench;
```

Test non exhaustif !!

- **Opérateurs** et littéraux

Classes d'opérateurs par ordre de priorité croissante

1. Logiques : and or nand nor xor;
   - défini sur les types booléen et BIT
2. Relationnels: **=** **/=** **<** **<=** **>** **>=**
   - défini sur tous les types sauf le type file
3. Arithmétiques et concaténation: **+** - &
4. Signe: **+** -
   - les opérateurs de signe sont moins prioritaires que les opérateurs de multiplication !!!!!
5. Multiplication: * / mod rem
6. Exposant, valeur absolue, complément: ** abs not
   - ** élévation à la puissance : opérande de droite (la puissance) doit être entière

Il est possible de surcharger les opérateurs; ceci ne change pas leur priorité.

VHDL basics

- Opérateurs et **littéraux**

Classifications:

numérique - notation décimale - entier (1345 ou 1_345) ou réel (13.0)
- notation basée - base 16 : X"A2"

caractères ou chaîne de caractères ('a' , "Bonjour", "1234")

énumérés (type ALU_OPS is (ADD,SUB,ADDI,SUBU, …))

chaînes de bits ("1010")

null

WARNING: 1 - nombre entier '1' - bit

Expressions - portent un type, et au moment de l ’exécution une valeur

Identificateurs - ils commencent avec une lettre; pas de différence entre majuscules et minuscules.

VHDL basics

- Objets

Les objets contiennent des valeurs. Il y a 4 classes d ’objets en VHDL : constantes, variables, fichiers et signaux.

Les constantes ont une valeur unique fixe

Les variables ont une valeur unique modifiable

Les fichiers contiennent des séquences de valeurs qui peuvent être lues ou écrites

Les signaux conservent l’histoire des valeurs passées, de la valeur présente et des valeurs prévues dans le futur : seules les valeur futures peuvent être modifiées par affectation de signal

Tous les objets ont un type

VHDL basics

- Types

VHDL est un langage typé

Un type est un ensemble de valeurs ordonnées

Le type est statique : il ne peut pas être modifié

Il est possible de définir de nouveaux types en utilisant les types prédéfinis et des constructeurs de types

Classification :

- Types scalaires
- Types composites
- Types access (pointeur) - seules les variables peuvent être de type access
- Types fichiers - mot clé file

VHDL basics

- Types scalaires

Les entiers

```
type mon_integer is INTEGER range -65_536 to 65_535 ;
```

Les flottants

```
type mon_flottant is REAL range 5.36 downto 2.15;
```

Les types énumérés

```
type COULEUR is (BLEU, VERT, ROUGE);
type BOOLEAN is (FALSE, TRUE);
type LOGIC4 is ('X', '0', '1', 'Z');
```

Les types physiques

TIME - défini dans le paquetage STANDARD

TIME - la notion de temps que connaît le simulateur
Sont caractérisés par l'unité de base, l'intervalle de ses valeurs autorisées, collection des sous-unités et leur correspondance.

VHDL basics

- Types physiques

```
type TIME is range -9_223_372_036_854_775_808 to 9_223_372_036_854_775_807
-- codage sur 64 bits ;
        units fs ;
                ps =1000 fs;            ms = 1000 us;
                ns = 1000 ps;           sec = 1000 ms;
                us = 1000 ns;           min = 60 sec ;
                                        hr = 60 min;
        end units;

type DISTANCE is range 0 to 1E16
        units A ;
                nm =10 A;               um = 1000 nm;
                mm = 1000 um;           cm = 10 mm;
                m = 1000 mm;            km = 1000 m;
        end units;
```

VHDL basics

- Type STD_LOGIC

Se trouve dans le paquetage IEEE.std_logic_1164

Paquetage IEEE.std_logic_unsigned

Paquetage IEEE.std_logic_arith

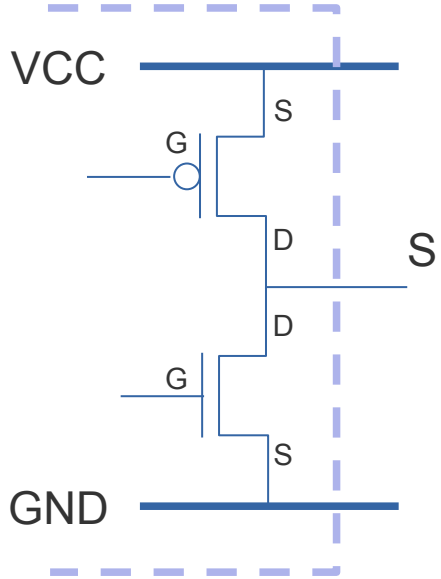
u stands for unsolved … not unsigned!

std_logic est le type résolu de std_ulogic

'1' – 1 *strong*
'0' – 0 *strong*
'Z' – high impedance
'U' – uninitialized
'X' – undetermined
'H' – 1 weak
'L' – 0 weak
'W' – weak unknown

std_logic resolution function

A	B	out
'1'	'Z'	'1'
'0'	'H'	'0'
'0'	'1'	'X'
'H'	'L'	'W'
...	...	...



STD_LOGIC_VECTOR

```
signal A,B : std_logic_vector(7 downto 0);
signal mycpt: std_logic_vector(0 to 3);
```

array of std_logic

- Types composites

Classification :

Tableaux : collection d'objets de même type - **array**

Articles : collection d'objets de types différents - **record**

Tableaux

Ses éléments sont désignés par un indice:

- Constraint (un entier)

```
type MOT is array (integer range 0 to 31) of STD_LOGIC;
```

- Non-constraint

```
type STD_LOGIC_VECTOR is array ( NATURAL range <>) of STD_LOGIC;
```

Définition d'un intervalle d'indiçage lors de l'exécution

```
signal bus : STD_LOGIC_VECTOR ( 63 downto 0);
```

Attributs de tableaux :

LEFT, RIGHT, HIGH, LOW, RANGE, REVERSE_RANGE, LENGTH

```
type FILE_REGS is array(0 to (2**3)-1) of std_logic_vector(31 downto 0);
signal regs : FILE_REGS;
```

- Tableaux - attributs

Exemple :

```
type INDEX 1 is INTEGER range 1 to 20;
type INDEX2 is INTEGER range 7 downto 3;
type VECTEUR1 is array (INDEX1) of STD_LOGIC;
type VECTEUR2 is array (INDEX2) of STD_LOGIC;
```

VECTEUR1'LEFT rendra 1 et VECTEUR2'RIGHT rendra 3

VECTEUR1'HIGH rendra 20 et VECTEUR2'HIGH rendra 7

VECTEUR1'LOW rendra 1 et VECTEUR2'LOW rendra 3

VECTEUR1'RANGE rendra 1 to 20 - utilisation pour des boucles

VECTEUR1'REVERSE_RANGE rendra 20 downto 1
VECTEUR1'LENGTH rend le nombre d'éléments du tableau - donc 20

- Types composites - articles

Articles - mot clé record - end record

Contient des éléments de types différents

Ses éléments sont désignés par un nom
Exemple :

```
type BIT_COMPLET is record
    valeur : std_logic;
    sortance_min, sortance_typ, sortance _max: integer;
end record;
signal A : BIT_COMPLET;
variable B : BIT_COMPLET;
```

alors A.valeur est de type std_logic, A.sortance_typ est de type entier

Affectation

```
A <= B ;
ou
A.valeur <= B.valeur after 50 ns;
```

VHDL basics

- Aggrégats

Un agrégat est une façon d'indiquer la valeur d'un type composite

Un agrégat se note entre parenthèse, les éléments étant séparés par des virgules

Vérification de type faite par le compilateur

Exemple : soit

```
type TAB is array (1 to 3) of INTEGER ;
type ART is record
        Champ1 : NATURAL;
        Champ2 : STD_LOGIC;
        Champ3 : NATURAL;
end record;
signal A : TAB; signal B : ART;
```

3 notations distinctes de l'agrégat :

notation positionnelle

```
A <= (12, 13, 14);                    B <= (5, '0', 15);
```

notation par denomination

```
A <= (1=>12, 2=>13, 3=>14);           B <=(Champ3 =>15,Champ1 => 5, Champ2 =>'0' );
```

notation mixte (positionnelle -denomination)

```
A <=(5, others =>0);                  B <=(Champ2 =>'0', others =>0);

A <=(others =>0);
```

VHDL basics

- Sous-types

Déclaration de sous-type quand on souhaite restreindre les valeurs d'un certain type

Un sous-type est toujours compatible avec son type de base

Exemples:

```
subtype integer_8bits is INTEGER range 0 to 255 ;
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
```

Sous-types dynamiques:

Font intervenir des expressions calculables à la simulations

```
Subtype MOT is STD_LOGIC_VECTOR (1 to MAX);
```

ou MAX est un paramètre d'un sous-programme ou un paramètre générique d'une entité

VHDL basics

- Notions de signal et affectations du signal

Un signal correspond à la représentation matérielle du support de l'information en VHDL.

Un signal a une évolution dans le temps (une variable - non)

Toute déclaration de signal se fait dans le domaine concurrent - la zone de déclaration des architectures,des entités et la spécification du paquetage

Tout signal peut conserver l'historique de ses valeurs passées si celles-ci sont utilisées par ailleurs, la valeur présente et les valeurs prévues dans le futur - le «pilote » du signal

Affectation du signal :

connexion à un port de sortie d'un composant

affectation du signal dans le domaine concurrent

affectation du signal dans le domaine séquentiel

- Affectation inconditionnelle de signal

S <='0', '1' after 20 ns;

La valeur doit avoir un type compatible avec celui du signal affecté

Chaque élément de l'expression a une partie valeur et une partie délai de type physique TIME (délai nul par défaut sans la clause after)

S <= A after 10 ns ;

Tous les signaux utilisés dans l'expression sont soit locaux, soit des ports définis en entrée

VHDL basics

- Exécution d'une affectation de signal

L'affectation du signal ne change pas la valeur présente du signal, mais modifie les valeurs futures que ce signal sera susceptible de prendre

A<= B ; ou A<=B after 0 ns;

Le signal A prend la valeur présente du signal B après un delta délai

Delta délai - est un délai qui est nul pour la simulation et ne représente que la causalité des événements.

- Exécution d'une affectation de signal

S <= A+B after 20 ns;

L'expression affectée à un signal est évaluée à chaque changement de valeur sur l'un de ses signaux d'entrée (réponse évènementielle)

Chaque valeur calculée par l'expression est mémorisée dans le signal destination avec son délai d'apparition

Si l'affectation est de nouveau évaluée, les anciennes valeurs prévues pour le futur peuvent être remplacées par les nouvelles valeurs - voir exemple affectation conditionnelle

- Affectation conditionnelle de signal

Condition simple:

```
S <= A when condition else
     B;
```

Instruction conditionnelle concurrente

```
S <= A after 20 ns when condition else
     B after 10ns;
```

Condition multiple avec ordre de précédence

```
S <= 5 when A='1' and B='1' else
     4 when A='1' else
     0;
```

Instruction conditionnelle concurrente

Quand n'importe quel signal d'entrée change de valeur,
l'ensemble des conditions est évalué dans l 'ordre (de gauche à droite)

La première valeur produite par la première expression dont la condition est vraie est affectée au signal !

- Affectation sélective de signal

Une condition unique détermine quelle expression sera affectée au signal

```
type alu_ops is (add, sub, and, xor) ;
signal opcode: alu_ops;
signal result, A, B, : std_logic;

with opcode select     – instruction de choix concurrente
    result <= A + B when add,
              A - B when sub,
              A and B when and,
              A xor B when xor,
              '0' when others;
```

```
result <= A + B when opcode==add else
          A - B when opcode==sub else
          A and B when opcode==and else
          A xor B when opcode==xor;
```

L'affectation sélective modélise les composants de type multiplexeurs et décodeurs.

not anymore a muxer ...
guess why ?

- Attributs

L'attribut est une caractéristique associée à un type ou à un objet

L'utilisateur peut définir des nouveaux attributs

Prédéfini :

S'quiet(T) - TRUE si le signal S est "tranquile" pendant au moins T

aucune affectation du pilote.

S'stable(T) -TRUE si aucun événement sur le signal S depuis T

S'delayed(T) -signal S différé de T après NOW

des affectations au delà du temps T

S'event - TRUE si un événement vient d'arriver sur S

des affectations mais qui ne changent pas la valeur

S'last_value - dernière valeur de S avant le dernier événement

S'last_event - valeur du temps écoulé depuis le dernier événement de S

S <= transport A and B after 20ns;

also inertial and reject to model propagation on wires.

VHDL basics

- Variable vs Signal

Les variables se déclarent dans le domaine séquentiel des processus ou des sous-programmes

Les variables s'utilisent et s'affectent dans le domaine séquentiel uniquement a := '1'

Les signaux se déclarent dans le domaine concurrent des entités, architectures, spécification du paquetage, des blocs

Les signaux s'utilisent dans les 2 domaines séquentiel et concurrent a <= '1'

Une affectation de variable est immédiate, alors qu'une affectation de signal est différée jusqu'à la fin de l'évaluation de toutes les affectations.

i.e lorsque l'on arrive à la fin d'un `process` ou lorsque l'on tombe sur une instruction `wait`

VHDL basics

- Instructions concurrentes

On ne simule pas en temps « réel » - on simule en temps virtuel (temps de simulation)

Ces fonctionnements seront décrits par des instructions concurrentes s'exécutant de manière asynchrone

Assignations de signaux (conditionnelle ou sélective ou inconditionnelle)

Instanciations de composants

Instruction processus

Appels concurrents de procédures

Instructions d'assertion

Instructions de bloc

Instruction generate

- Instructions concurrentes

  Processus

  est l'objet fondamental manipulé par le simulateur

  toute instruction concurrente peut toujours être traduite par un processus

  un processus est sensible aux signaux

  il contient que des instructions séquentielles

  la durée de vie d'un processus est celle de la simulation; un processus est cyclique

e.g affectation dans le domaine concurrent:

```
S <= A + B;
```

est en fait un processus avec en liste de sensibilité les signaux situés en partie droite de l'affectation (i.e A, B)

VHDL basics

- Instructions concurrentes - Processus

{label: } **process** { liste_des_signaux_surveillés}
Déclarations
begin
Instructions séquentielles
end process {label};

liste de sensibilité

Les déclarations sont élaborés une seule fois, à l'initialisation

un processus s 'exécute au moins une fois à l 'initialisation jusqu'à l'instruction WAIT (si elle existe)

à chaque événement intervenant sur un des signaux surveillés, le processus va s'exécuter

Restrictions:

l'instruction wait bloque le processus – voir instruction wait plus loin

l'instruction wait ne peut s'utiliser à l'intérieur d'un processus que si celui-ci ne possède pas de liste de signaux surveillés

- Instructions concurrentes - Processus

Processus avec liste de sensibilité:

{label: } **process**({watched_signals})
    Déclarations
begin
  Instructions séquentielles
end process {label};

Not executed at initialization

Processus sans liste de sensibilité:

{label: } **process**
    Déclarations
begin
    wait on {signaux_surveillés }
    Instructions séquentielles
end process {label};

Executed one time during initialization

- Instructions concurrentes

Block

réuni des instructions concurrentes

est la base de hiérarchie en VHDL

toute hiérarchie VHDL se ramène à un ou plusieurs blocs imbriqués

permet de garder des affectations de signaux - circuits synchrones

```
label : block
        { généricité et port }
        … Déclarations …
        begin
        ….Instruction concurrentes
        end block;
```

Appel concurrent de procédure

a la même syntaxe que l'appel séquentiel de procédure

domaine concurrent - paramètres de la procédure ne peuvent être que de signaux ou de constantes

```
procedure alu(A:in std_logic_vector, …., signal S: out std_logic,… );
```

Instruction concurrente d'assertion

VHDL basics

- Instructions séquentielles

Domaine d'utilisation des instructions séquentielles : dans le corps d'un processus ou sous-programme

Instruction WAIT

Instruction ASSERT

Instructions d'affectation de variables et de signaux

Appel de procédures

Instructions conditionnelles

Instructions de contrôle

Instruction nulle - null

- Instructions séquentielles

Instruction WAIT

L'exécution de processus ou de programme peut être suspendue en fonction d'une condition donnée et pour un temps indiqué par l'instruction wait

wait {on liste _signaux} {until condition_booléenne} { for temps};

Tout événement arrivant sur un signal de la liste donnée provoque l'évaluation de la condition booléenne.

Exemple:

```
process
begin
    wait on CLK;
    if clk'event and CLK='1' then
        q <=d after 10 ns;
    end if;
end process;
```

```
process
begin
   wait until CLK'event and CLK ='1';
   q <=d after 10 ns;
end process;
```

`wait until <condition>`
attend un événement sur l'un des signaux ET teste ensuite la condition !!

Clock generator in a test architecture:

```
  signal clk: std_logic;
.....
clock: process
begin
  clk <= '1';
  wait for 5ns;
  clk <= '0';
  wait for 5ns;
end process clock;
```

- Instructions séquentielles

Instruction ASSERT

Surveille une condition et émet un message dans le cas où celle-ci est fausse.

L'exécution du programme reprend alors immédiatement derrière l'instruction d'assertion.

assert condition {report mesg} {severity niveau}

assert NOW<1 min report "Fin simu" severity ERROR;

Type Severity_Level is (note, warning, error, failure);

Timeout generator in a test architecture:

```
timeout: process
begin
  wait for 150ns;
  assert false report "Timeout !!!" severity failure;
end process timeout;
```

- Instructions séquentielles

Instructions d'affectation de variables et de signaux

```
Variable1 := 2;
Signal1 <= 2 after 20 ns;
```

Appel de procedure

```
procedure alfa (nr : integer, msg : string) --définition de la procédure
Alfa(4, "GO");                               --appel positionnel
Alfa(nr =>4, msg =>"GO");
```

Instruction return

Réservée aux sous-programmes

Instruction nulle

null

l'exécution passe à la ligne suivante

L'instruction nulle n'est pas nécessaire à la compilation de processus ou de corps de procédures vides.

- Instructions séquentielles

Instructions conditionnelles

```
if condition1 then traitement1
elsif condition2 then traitement2
else traitement3
end if;
```

Instructions de choix:

```
case expression is
   when val1 => instructions1
   when val2 => instructions2
   when others => instr3
end case;
```

e.g procedure alu

```
  type ALU_OPS is (ALU_ADD, ALU_SUB, ALU_AND ...);
  signal CTRL_ALU : ALU_OPS;
……
case CTRL_ALU is
  when ALU_ADD | ALU_SUB | ALU_SLT  =>
    b_in := B;
    c_in := '0';
    if (CTRL_ALU /= ALU_ADD) then
      b_in := not(B);
      c_in := '1';
    end if;
    adder_cla(A,b_in,c_in,tmp_S,tmp_C,tmp_V);
    if (CTRL_ALU = ALU_SLT) then
      tmp_S := ……
      tmp_C := '0';
      tmp_V := '0';
    else
      tmp_C := not(SIGNED_OP) and tmp_C;
      tmp_N := SIGNED_OP and tmp_S(DATA_WIDTH-1);
      tmp_V := SIGNED_OP and tmp_V;
    end if;
  when ALU_AND =>
    tmp_S := A and B;
  ……
  when others =>
end case;
```

- Instructions séquentielles

Instructions de boucles

```
loop
    instructions séquentielles;
end loop;
```

```
for indice in intervalle loop
    instructions séquentielles;
end loop;
```

```
while condition loop
    Instructions séquentielles;
end loop;
```

```
next label_de_boucle when condition --arrête l'itération en cours
```

```
exit label_de_boucle when condition
```

zero detector

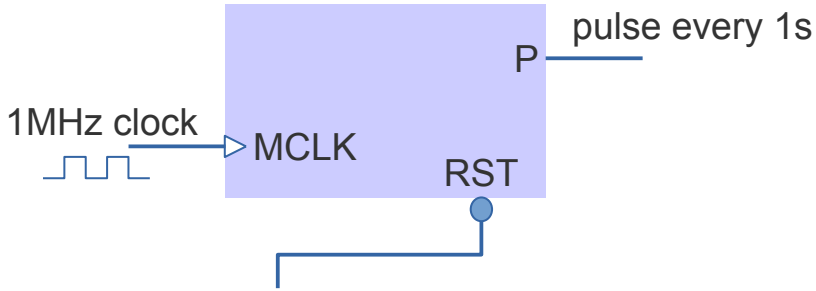
```
  signal reg:std_logic_vector(7 downto 0);
……
process( reg )
  variable zero:std_logic := '1';
begin
  zero:='1';
  for i in reg'range loop
    if reg(i)/='0' then
      zero:='0';
      exit;
    end if;
  end loop;
  if zero='1' then …
  end if;
end process;
```

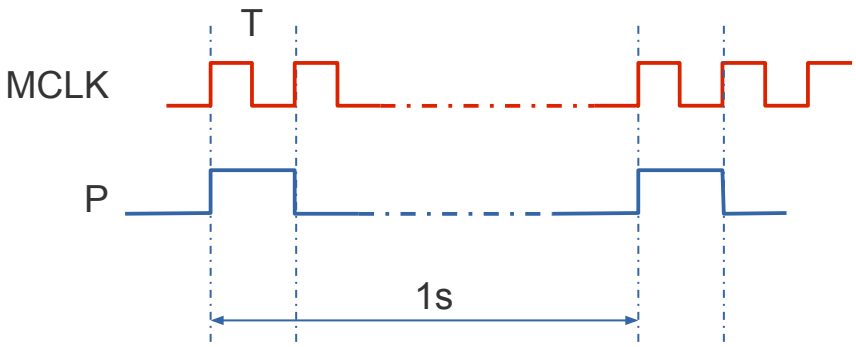
… then the same using vector comparison :)

```
if reg/=conv_std_logic_vector(0,reg'length) then
  ……
end if;
```

Exercises

- Pulse generator






```
entity pulse_gen is
  port( RST,MCLK:in std_logic;
        P:out std_logic );
end pulse_gen;
```

Whenever a reset occurs (i.e `RST='0'`), system resets itself. Normal operation restarts when reset is no more active.

Retrieve and edit the following files:

- `pulse_gen.vhd` → entity + architecture of your component
- `test_pulse_gen.vhd` → test architecture.

Compilation:

```
ghdl -a --ieee=synopsys -fexplicit \
pulse_gen.vhd test_pulse_gen.vhd
```

Elaborate:

```
ghdl -e --ieee=synopsys -fexplicit test_pulse_gen
```

Run:

```
ghdl -r --ieee=synopsys -fexplicit test_pulse_gen \
--wave=test_pulse_gen.ghw
```

Visualisation:

```
gtkwave test_pulse_gen.ghw
```

Note: search for ghdl makefile to ease things a bit.

Exercises

- *log2* function

Allows you to get the number of bits required to encode a value:

log2(1)=0
log2(2)=1
log2(4)=2
log2(8)=3

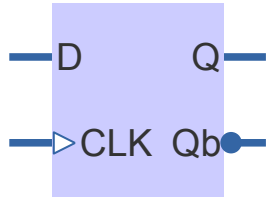
```
function log2(V:natural) return natural is
begin
   ……
end log2;
```

VHDL basics

- D flip-flop

What's wrong here ??

```
entity basculeD is
port(
    D, CLK: in std_logic;
    Q: inout std_logic;
    Qb: out std_logic);
end basculeD;
architecture beh1 of basculeD is
begin
  process
  begin
    wait until clk'event and clk='1';
    Q <= d;
    Qb <= not Q;
  end process;
end beh1 ;
```

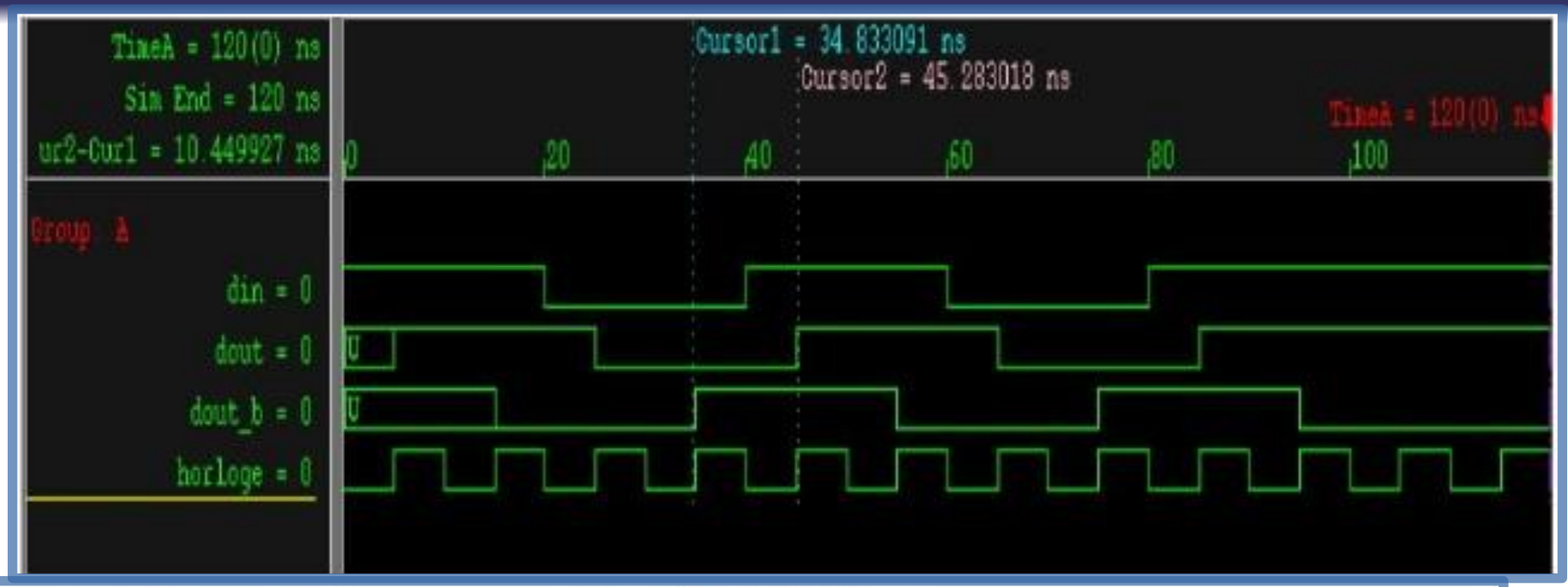



```
entity basculeD is
port(
    D, CLK: in std_logic;
    Q: inout std_logic;
    Qb: out std_logic);
end basculeD;
architecture beh2 of basculeD is
Begin
  Qb <= not Q;                    process001
  process                         process002
  begin
    wait until clk'event and clk='1';
    Q <= d;
  end process;
end beh2 ;
```

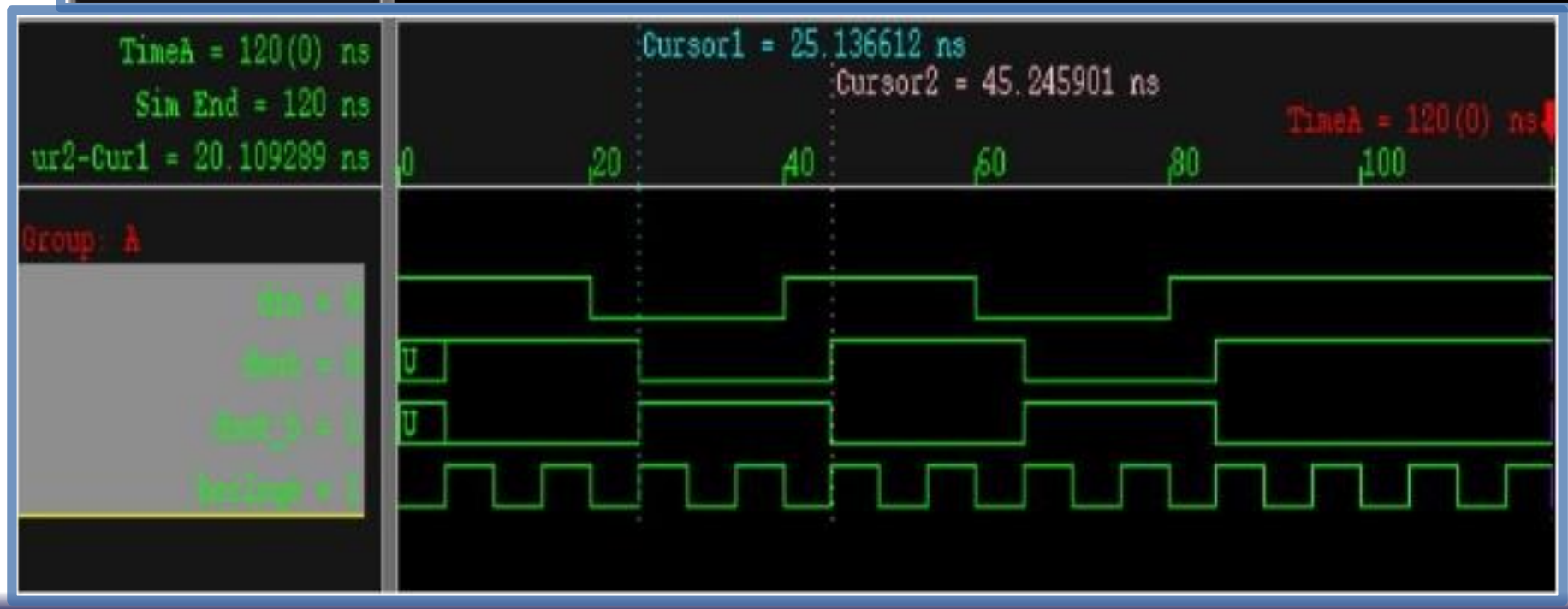
really ??

VHDL basics

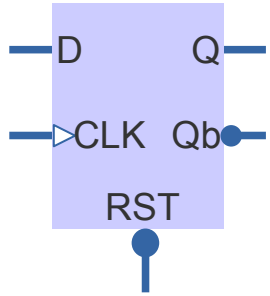
beh1 architecture



beh2 architecture



- D flip-flop – Reset featured



asynchronous

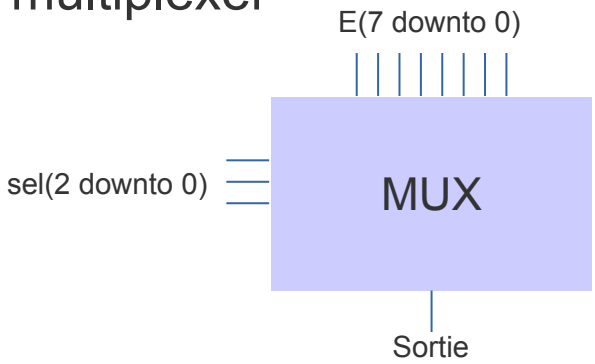
```
entity basculeD is
  port(
    RST, D, CLK: in std_logic;
    Q: out std_logic;
    Qb: out std_logic );
end basculeD;

architecture beh3 of basculeD is
begin
  process( RST, CLK )
  begin
    if RST='0' then
      Q <= '0'; Qb <= '1';
    elsif rising_edge( CLK ) then
      Q <= d;
      Qb <= not D;
    end if;
  end process;
```

synchronous

```
entity basculeD is
  port(
    RST, D, CLK: in std_logic;
    Q: out std_logic;
    Qb: out std_logic );
end basculeD;

architecture beh3 of basculeD is
begin
  process( CLK )
  begin
    if rising_edge( CLK ) then
      if RST='0' then
        Q <= '0'; Qb <= '1';
      else
        Q <= d; Qb <= not D;
      end if;
    end if;
  end process;
```

Exercises

- 8 bits multiplexer



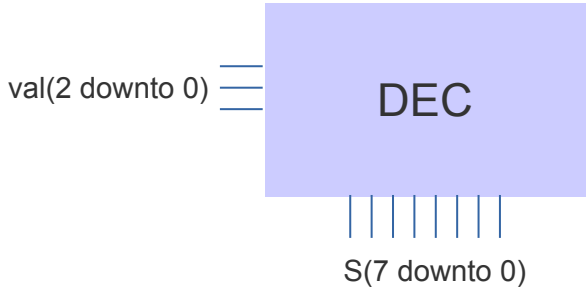

```
entity mux is
  port (
      sel : in integer range 0 to 7 ;
      entree : in std_logic_vector(7 downto 0);
      Sortie : out std_logic);
end mux;
architecture beh of mux is
begin
   with sel select -- instruction de choix concurrente
        sortie <= entree(0) when 0,
                  entree(1) when 1,
                  entree(2) when 2,
                  entree(3) when 3,
                  entree(4) when 4,
                  entree(5) when 5,
                  entree(6) when 6,
                  entree(7) when 7;
end beh;
```

Ré-écrivez ce multiplexeur en utilisant les affectations conditionnelles :

- dans le domaine combinatoire,
- puis dans un process.

Exercises

- 3 → 8 decoder



val	S0	S1	S2	S3	S4	S5	S6	S7
0	1							
1		1						
2			1					
3				1				
4					1			
5						1		
6							1	
7								1

Only one output activated for each input value … now it's up to you !

VHDL basics

- Généricité

une entité est générique, jamais une architecture.

permet de définir une famille de composants par rapport à une caractéristique donnée.

valeur fixée lors de l'instance de l'entité

default value

```
entity mydecoder is
   generic ( size: natural:=4 );
    port ( S: out STD_LOGIC_VECTOR (size-1 downto 0);
           val: in STD_LOGIC_VECTOR( ____________ downto 0) );
end myadder;
```

guess what to set here ??

Instanciation du composant :

```
dec1 : entity work.mydecoder(behaviour)
        generic map (8)
        port map (val=> xxx, S=> yyy);
```

- Généricité – instruction concurrente generate

Permet l'élaboration itérative ou conditionnelle de lignes de code

2 formes : conditionnelle et itérative
Forme conditionnelle

```
label : if condition_booléenne generate
        suite d'instructions concurrentes
end generate {label};
```

Forme itérative

```
label : for nom_du_paramètre_de_la_génération in intervalle discret generate
        suite d'instructions concurrentes
end generate {label};
```

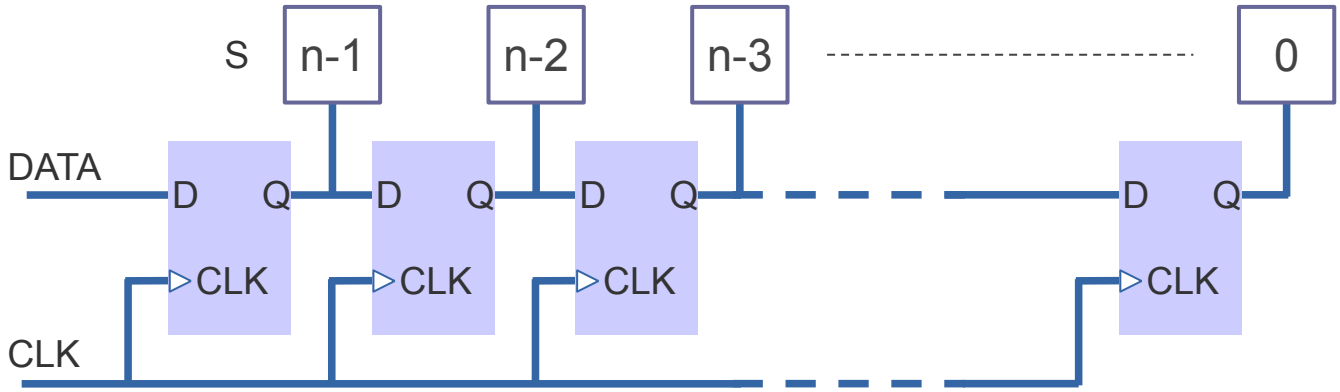
- Généricité

Exemple: registre à décalage générique (N)

entrées Data et Clock - std_logic;
sortie S est un std_logic_vector (N-1 downto 0)

architecture structurelle en utilisant un composant de type bascule D

```
entity reg_decalage is
   GENERIC( N: natural :=8 );
   port ( Data, clock : in std_logic;
          S : out std_logic_vector(N-1 downto 0) );
end reg_decalage;
```



- Généricité

```
library work;
use work.basculeD.all;

ARCHITECTURE structurelle OF reg_decalage IS

    for all : basculeD use entity work.basculeD(beh3);

begin

    gauche : basculeD port map (data, clock, S(N-1), open);

    boucle : for i IN 1 to N-1 generate
       circ : basculeD PORT MAP (S(N-i), clock, S(N-i-1), open);
    END GENERATE boucle;

END structurelle ;
```

What's wrong here ??

- Généricité

```
library work;
use work.basculeD.all;

ARCHITECTURE structurelle OF reg_decalage IS
    signal aux : std_logic_vector (N-1 downto 0);

    for all : basculeD use entity work.basculeD(beh3);

begin
    gauche : basculeD port map (data, clock, aux(N-1), open);

    boucle : for i IN 1 to N-1 generate
       circ : basculeD PORT MAP (aux(N-i), clock, aux(N-i-1), open);
    END GENERATE boucle;
    S <= aux;
END structurelle ;
```

Now it’s ok :)

- Généricité

architecture comportementale

```
ARCHITECTURE beh OF reg_decalage IS

begin

        comport : process
        begin
                wait until clock'event and clock ='1';
                s(n-1) <= Data ;
                copie : For i IN n-1 to 1 LOOP
                s(n-i-1) <= s(n-i);
                end loop copie;
      end process comport ;

END beh;
```

What's wrong here ??

- Généricité

```
ARCHITECTURE beh OF reg_decalage IS

signal aux : std_logic_vector (N-1 downto 0);
begin

        s <= aux ;
        comport : process
        begin
                wait until clock'event and clock ='1';
                aux(n-1) <= Data ;
                copie : For i IN n-1 to 1 LOOP
                aux(n-i-1) <= aux(n-i);
                end loop copie;
        end process comport ;

end beh;
```

Now it's ok :)

VHDL basics

- Compilation, élaboration, exécution, exploitation

Utilisation du VHDL :

- compilation et éditions de liens
  - aspects statiques de la description

- élaboration
  - aspects paramétrables de la description;
  - effectue l'instanciation de la description VHDL;
  - si la structure est hiérarchique il faut commencer par le plus haut niveau

- exécution
  - pour un simulateur <=> simulation
  - pour un synthétiseur <=> la synthèse

- exploitation des résultats plus spécifique de la simulation

VHDL basics

concurrent domain

sequential domain

affectations conditionnelles

```
Q <= value1 when condition else
     value2;
```

```
if condition then
    Q <= value1;
else
    Q <= value2;
end if;
```

boucles

```
for indice in intervalle generate
affectations concurrentes;
end generate;
```

```
for indice in intervalle loop
    instructions séquentielles;
end loop;

while condition loop
    Instructions séquentielles;
end loop;
```

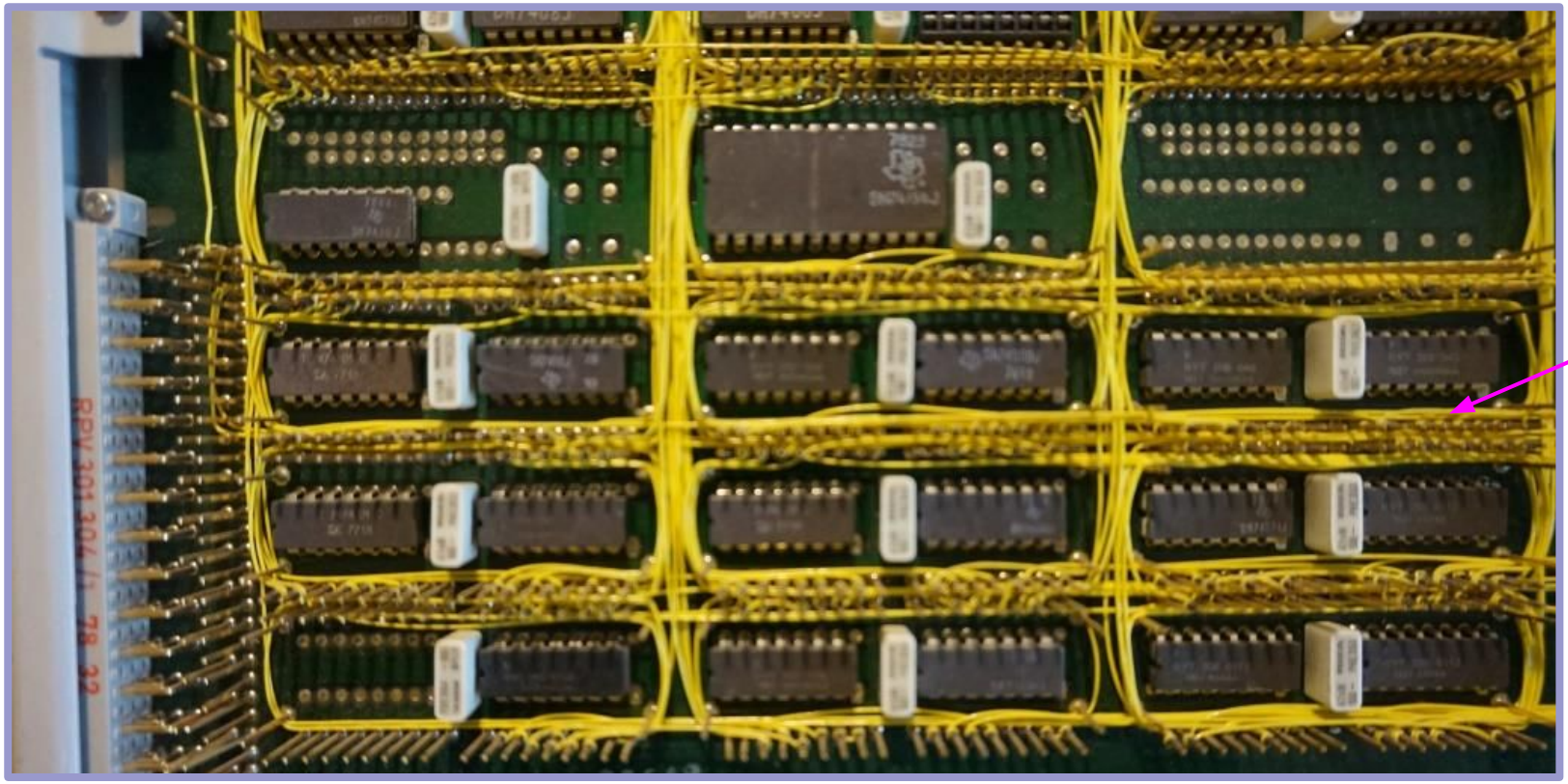
Plan

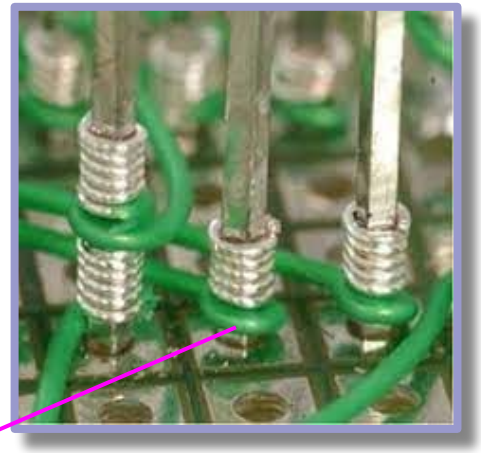
Part II – Synthesis for FPGA targets

- Introduction,
- VHDL logic synthesis,
- Xilinx Zynq,
- GoWin GW1NSR-xC (cortex M3+FPGA)

Introduction

- ICs wrapping nightmare in the 80's …





*[PAL] Programmable Array Logic collapsed **numerous** combinatorial functions **into a single chip** :)*
… then PALs evolved to [GALs]
While PALs were OTP, GALs enabled reprogramming and added sequential logic (D flip-flop, …)*

By the end of the 80's, Xilinx invents FPGA !

Introduction

- starting in the 90's, **FPGA** & **CPLD** as synthesis targets :)

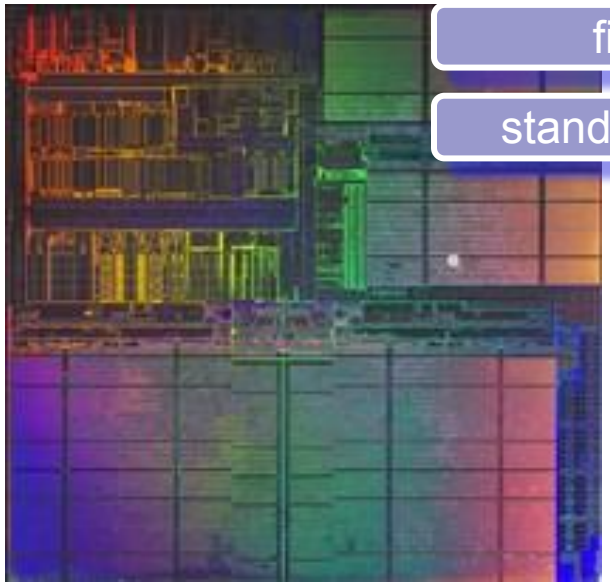
FPGA / CPLD

reconfigurable

gates arrays

€€, Moderate speed, convert to ASIC for high volumes, partially customisable, errors ⇒ just **reflash** :)

ASIC



fixed

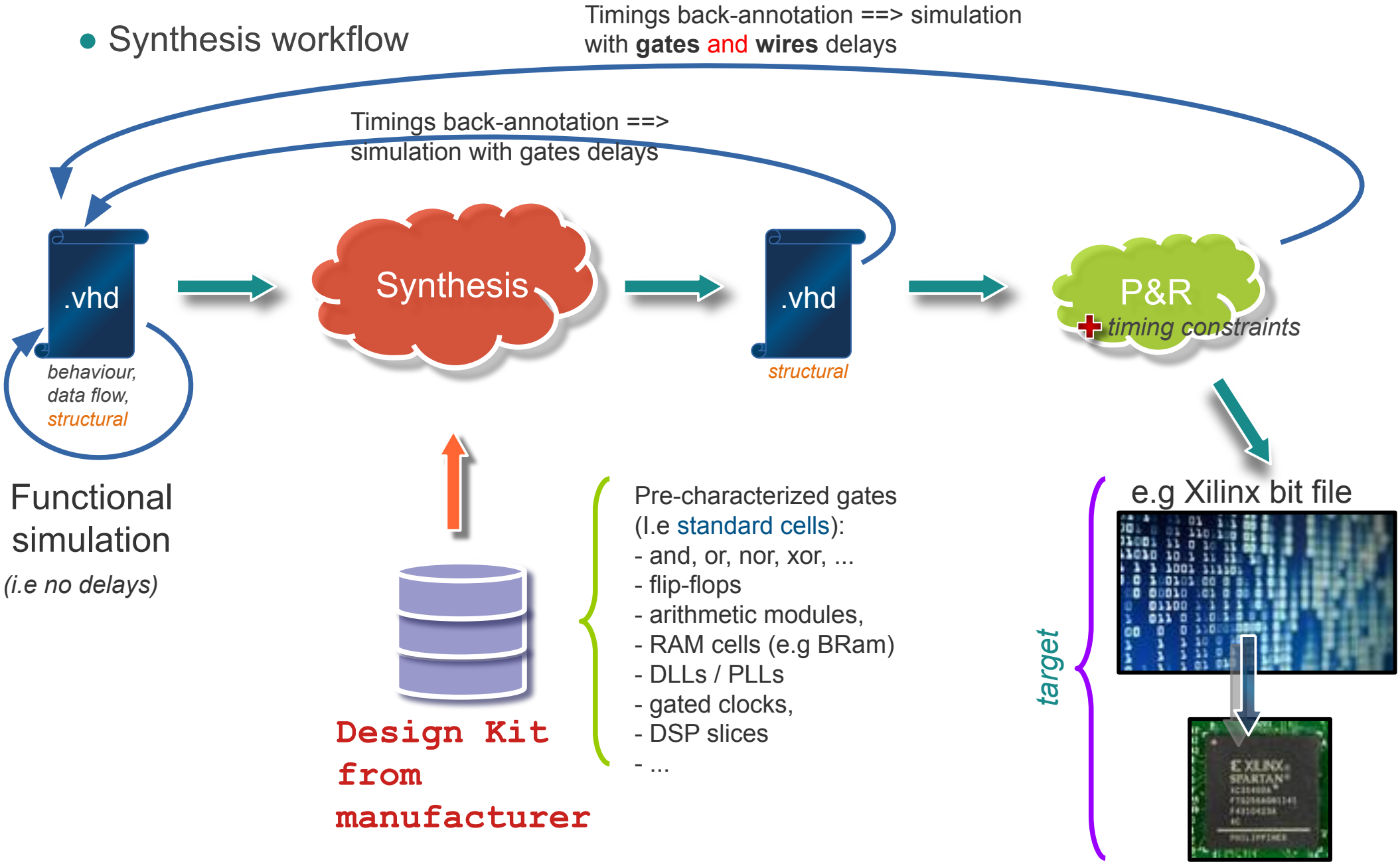
standard cells

(HP PA-RISC 8500 die)

€€€€, Highest speed, low-cost on (*high*) volumes, fully customisable … but errors ⇒ you're dead!

Introduction

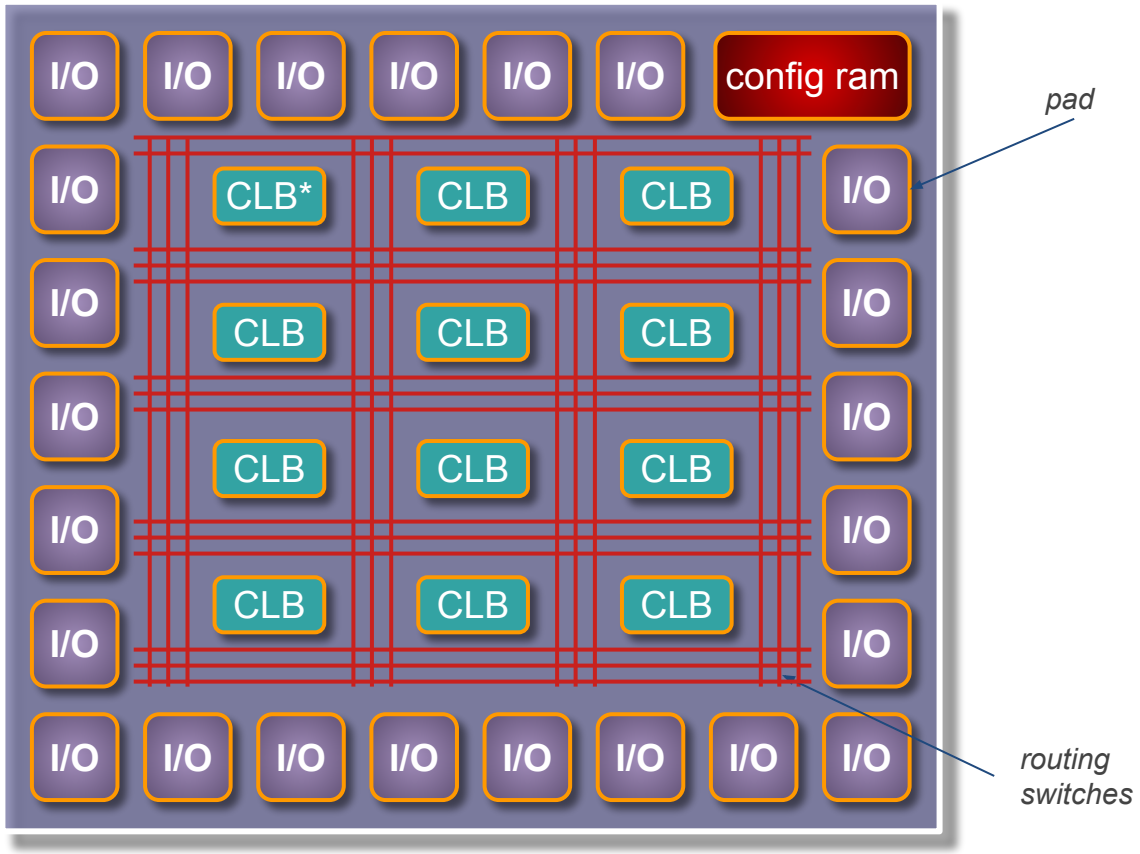
- Synthesis workflow



Introduction

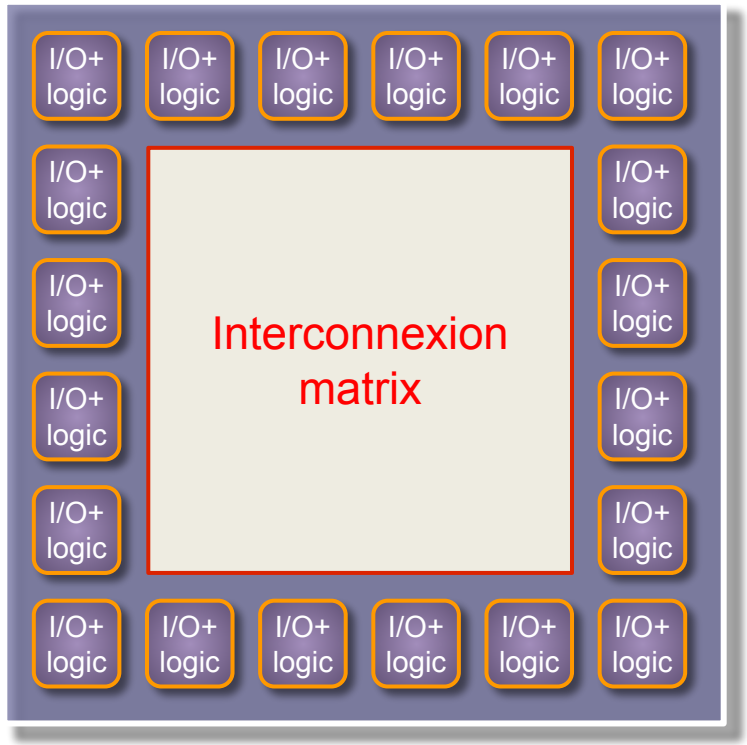
- Synthesis to reconfigurable hardware

FPGA



pad 2 pad delay depends on internal routing
⇒ non deterministic !

CPLD



pad 2 pad propagation delay is deterministic.

** **C**onfigurable **L**ogic **B**loc features LUT(s) with registers.*

Introduction

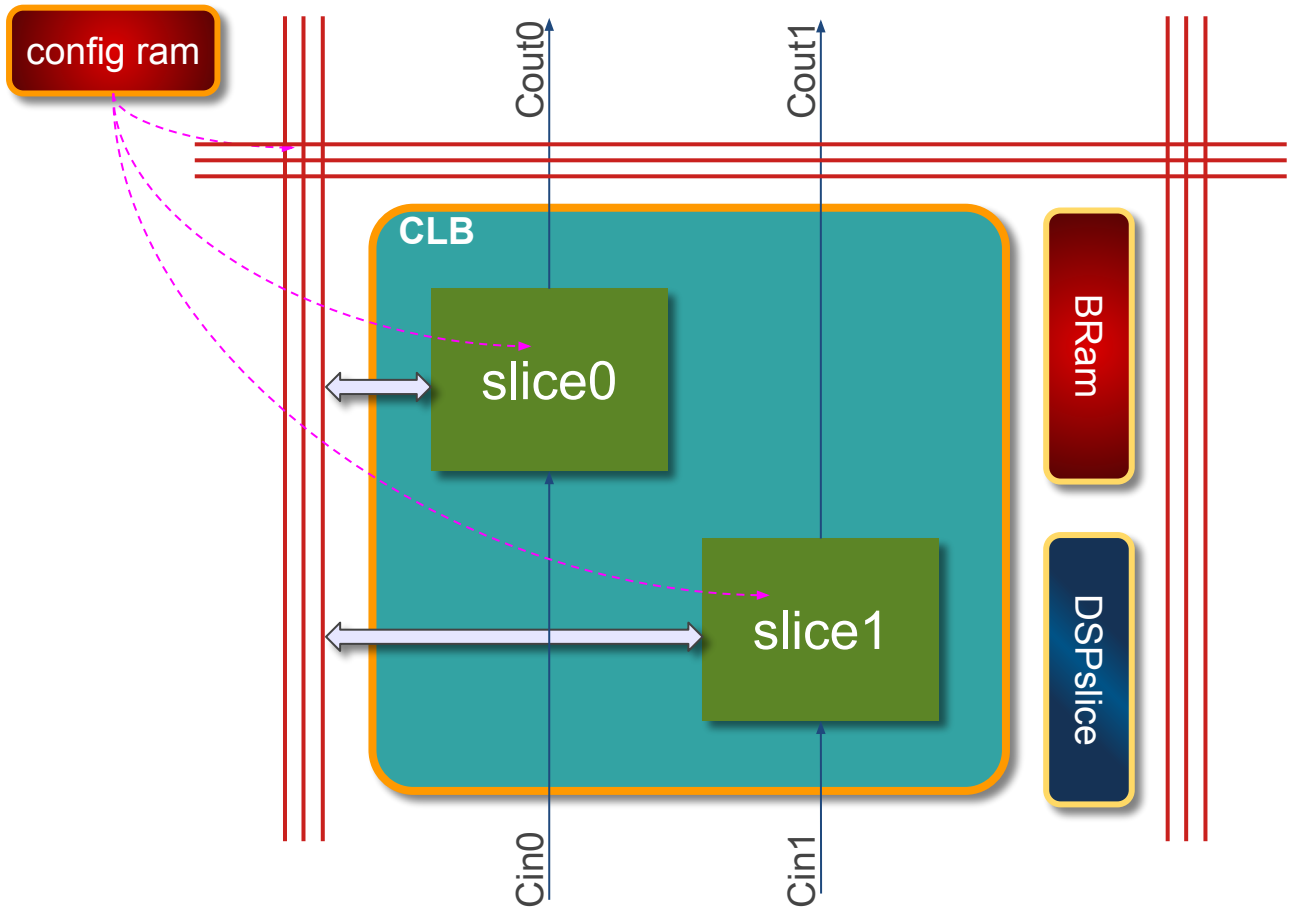
- Hardware (re)configurable targets
  - CPLD → huge crossbar with surrounding I/O pads
  - FPGA → CLBs spread over the entire chip with huge interconnexion matrix

High-End FPGA may contains either
Hard-IP processors (e.g dual ARM9 in Xilinx **Zynq**)
Softcore processor (e.g RISC-V, MicroBlaze or Leon)

CPLD features less logic blocks but it exhibits deterministic propagation delays.

Introduction

- Anatomy of a CLB (**C**onfigurable **L**ogic **B**lock)

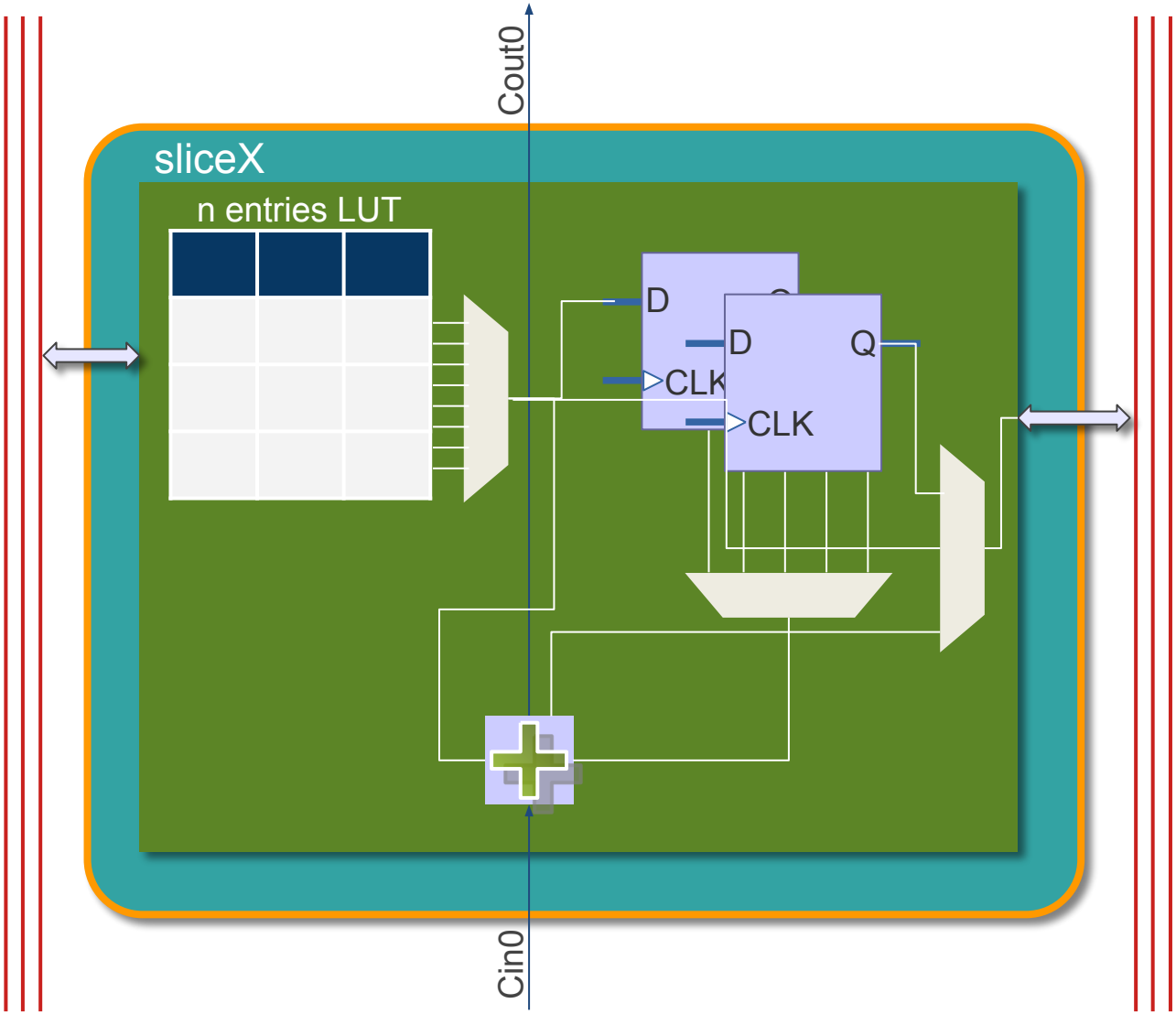


- **config ram** configure every piece of inner hardware. It holds content of the bitfile generated by synthesis tools.
- **sliceX** hold LUTs, registers, adders, …
- BlockRAMs spread within routing matrix,
- DSPslices to build signal processing engines,
- Clock distribution management along with PLL and/or DLL (not shown), …

Excerpt from SOC Zynq-7000 CLB architecture. Slices count may differ according to versions and manufacturers.

Introduction

- CLB (**C**onfigurable **L**ogic **B**lock) anatomy: slice inner view



- LUT stands for **L**ook **U**p **T**able,
- a LUT features n inputs, 1 or 2 outputs
- a LUT is basically a table of truth: determine the output for any given input,
- LUT represents the combinatorial logic,
- D flip-flop enables sequential logic,
- all configurations from **config ram**

Plan

Part II – Synthesis for FPGA targets

- Introduction,
- VHDL logic synthesis,
- Xilinx Zynq,
- GoWin GW1NSR-xC (cortex M3+FPGA),

VHDL logic synthesis

- RTL level synthesis*

  C'est un processus de traduction :

  entrée : description comportementale ou de type flot de données

  sortie : description structurelle à partir d'éléments de base prédéfinis - standard cells

  Respect de la fonctionnalité originale

  Respect des contraintes en temps et en topologie

  Représentation physique :

  - ASIC
  - FPGA

  IEEE 1076.6-2004 defines a subset of VHDL for RTL synthesis

**__H__igh __L__evel __S__ynthesis on its side (e.g Vivado "HLS") enables architectures explorations as a custom partitioning between hardware and software. This way, you usually start from a C,C++ or SystemC application for example.*

VHDL logic synthesis

- Types pour la synthèse logique

  énumérés

  type BIT et opérations associées

  types du package STD_LOGIC

  entiers :
  - la plage de variation détermine le nombre de bits nécessaires
  - les bits ne sont pas individuellement accessibles

  Types **non synthétisables**
  - REAL, ACCES, FILE
  - Types physiques: TIME ou définis par l'utilisateur

VHDL logic synthesis

- Remarques

  usage des types énumérés fortement recommandé : on reste synthétique et on laisse à l'outil de synthèse le choix de la bonne stratégie de codage

  il n'y a pas de consensus quant au codage des types énumérés

  seuls les tableaux à une dimension sont synthétisables à cause de la difficulté du calcul d'adresse. Les agrégats sont mis à plat.

  les types prédéfinis du package IEEE_1164 sont fortement recommandés

VHDL logic synthesis

- Objets et synthèse logique
  - Constantes

    Acceptées pour tous les types synthétisables. Leur déclaration ne produit aucun matériel. Leur usage produit du matériel dans les cas suivants :

    - partie droite d'affectation de signal
    - dans une instruction if ou case
    - dans une instruction concurrente conditionnelle
  - Signaux

    assimilables à des fils ou bus
  - Variables (affectation de variable)

    pas de règle générale pour déduire le matériel produit

    c'est le contexte d'utilisation qui est déterminant

VHDL logic synthesis

- Initial values

  En VHDL 3 types:

  - Valeurs par défaut héritée de la définition du type ou du sous-type
  - Initialisation explicite à la déclaration de l'objet
  - Valeur affectée par instruction au début d'un *process*

  Synthèse - les 2 premiers cas sont ignorés par les outils de synthèse

VHDL logic synthesis

- Generic parameters

  Permet de définir une famille de composants par rapport à une caractéristique donnée

  Valeur est fixé lors de l'instanciation de l'entité

  Synthèse :

  Le type de paramètres génériques synthétisable est restreint en fonction de l'outil
  Pour SYNOPSYS : seulement les paramètres génériques de type INTEGER ou énumérés sont synthétisable

VHDL logic synthesis

- Sequential instructions and logic synthesis
  - Instruction de synchronisation :

    sur signal de la liste de sensibilité (process (A,B,C) ou wait A,B,C)

    ⇒ matériel combinatoire

    sur signal d'horloge (clk'**event** and clk='1') ou (clk'**event** and clk='0')

    ⇒ D flip-flop (FF)

    sur niveau de signal (en='1' ou en='0')

    ⇒ Latch

  - Instructions d'itération

    for : synthétisable dans la mesure où les bornes de variation de l'index sont statiques.

    while : non-synthétisable au niveau RTL

  - Appel de sous-programmes

    certaines fonctions sont connues de la synthèse - pas de matériel additionnel

    fonctions passage de paramètres par valeurs - combinatoire

    fonctions passage de paramètres par référence - pas du domaine de la synthèse logique

VHDL logic synthesis

- Processes and logic synthesis

  Problème principal : combinatoire ou séquentiel ?

  Processus combinatoire si :
  - liste de sensibilité ou un seul point de synchronisation (wait)
  - pas de déclaration de variable ou bien variables locales systématiquement affectées avant d'être lues
  - les signaux lus se trouvent tous dans la liste de sensibilité
  - Les signaux écrits sont tous affectés quelques soient les branchements parcourus

  Si création de mémoire - processus séquentiel
  - style synchrone - bascule D

VHDL logic synthesis

- Concurrent instructions and logic synthesis

  Affectation de signal simple - matériel combinatoire

  Affectation conditionnelle ou sélectée

  S <= A when X=‘ 1 ’ else B when Y=‘ 1 ’ else C - combinatoire

  S <= A when X=‘ 1 ’ else B when Y=‘ 1 ’ else S - séquentiel

  Instanciation (de composants, generate)

  Les notions d'entité et de composant permettent de hiérarchiser la description d'un système

  Configuration - ne sont pas supportées

  Durant le processus de synthèse, différentes interprétations possibles:

  - flattened instances
  - composant synthétisé une fois. Seule la frontière est re-synthétisée pour chaque instance
  - composant synthétisé une fois et dupliqué pour chaque instance

VHDL logic synthesis

- Performances

  Timing
  - Estimation des retards induits par les éléments combinatoires, en particuliers détection des "chemins critiques"
  - Estimation des retards induits par les éléments des mémoire
  - Estimation du temps de cycle de l'horloge

  Surface
  - Estimation en fonction des composants utilisées:
    - Standard cells (ASIC)
    - LUT, FF, DSPslices, BRam … (FPGA)

  Consommation

  Testabilité, fiabilité, "manufacturabilité"

- Timing constraints

4 catégories de contraintes temporelles peuvent être introduites pour optimiser un circuit avec temps minimums ou maximums entre différents points du circuit:

- Entrées - registres
- Registres - sorties
- Entrées - sorties
- Registres - registres

Définition de la période de l'horloge globale $T_{période}$ + le temps additionnel sur les entrées T_{in} et le temps additionnel sur le sorties T_{out}

- T_{out} - information sur la charge équivalente sur chaque sortie ⇒ *fan-out*
- T_{in} - information sur la puissance de sortie du composant connecté en entrée
- $T_{période}$:

  information de "skew" = déphasage maximal entre les arrivées de l'horloge sur les bascules

  information sur la latence = temps maximal entre la commutation de l'horloge et la dernière bascule recevant l'horloge.

VHDL logic synthesis

- Timing optimisations
  - Permet d'améliorer les caractéristiques de temps de traversée et de fréquence d'horloge
  - Les outils de CAO ne modifient pas le nombre de bascules du circuits lors de l'optimisation
  - Les outils CAO optimisent la logique combinatoire interne
  - Il faut isoler le chemin le plus long de traversée de la logique combinatoire, c'est le chemin critique ⇒ OPTIMISATION DU CHEMIN CRITIQUE
    - Toute logique combinatoire partagée entre le chemin critique et d'autres chemins de logique est dupliquée pour permettre une optimisation poussée sur le chemin critique,
    - Augmentation de la taille des circuits après une optimisation temporelle.

Links

- additional resources

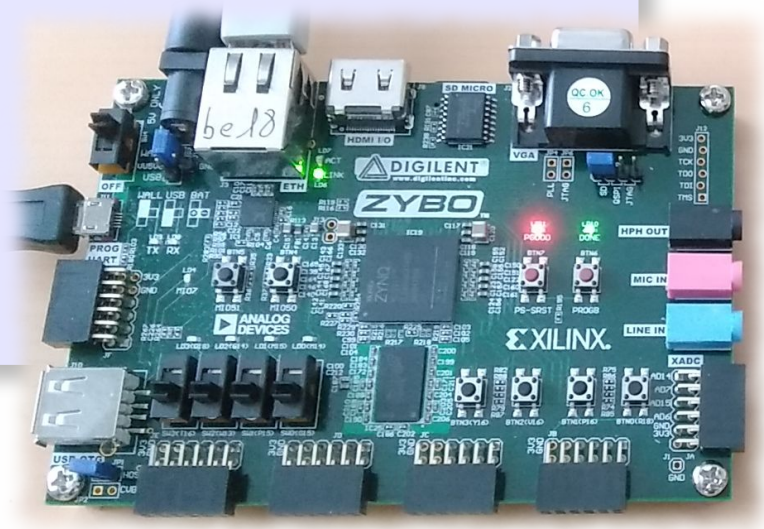
ENSSAT Lannion | VHDL logic synthesis
http://people.rennes.inria.fr/Olivier.Sentieys/teach/VHDL_Logic_Synthesis_2019.pdf

Open source synthesis tools (verilog)
https://github.com/YosysHQ/yosys

Plan

Part II – Synthesis for FPGA targets

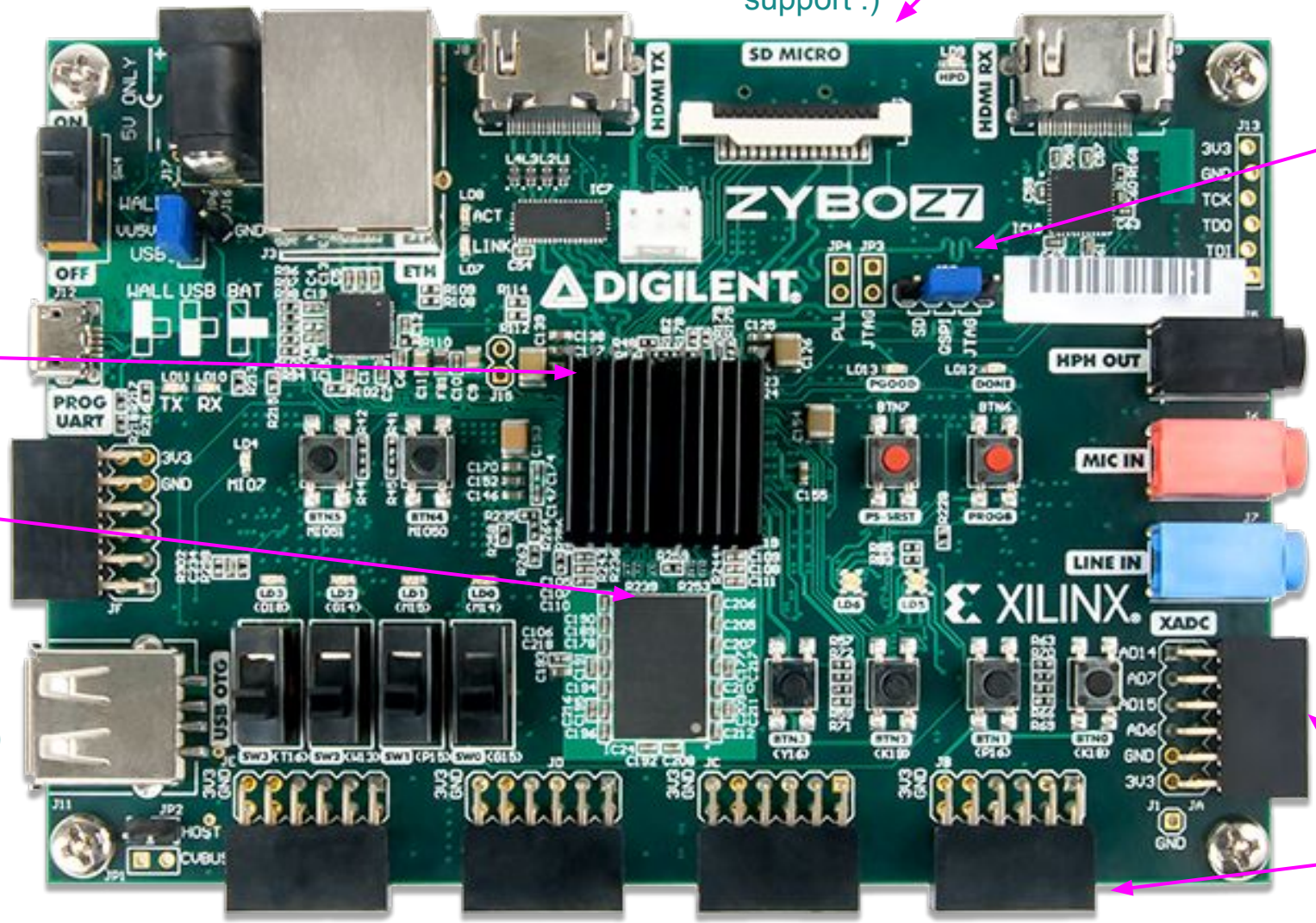
- Introduction,
- VHDL logic synthesis,
- Xilinx Zynq,
- GoWin GW1NSR-xC (cortex M3+FPGA),



Digilent Zybo, Xilinx *Zynq 7010*

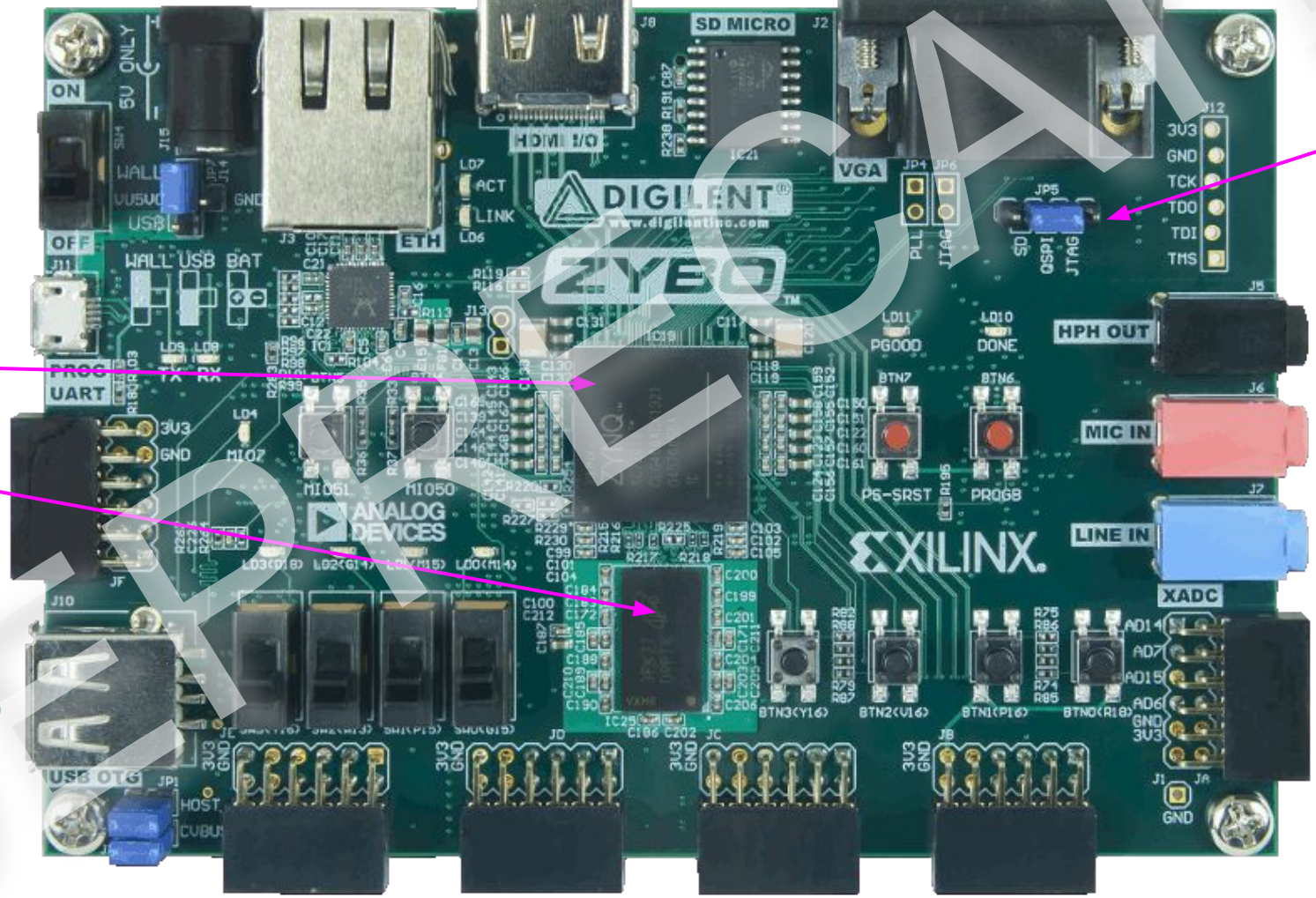
Xilinx Zynq (Zybo-Z7)

- Digilent Zybo Z7-20 (Xilinx Zynq 7020)



Xilinx Zynq (Zybo)

- Digilent Zybo (Xilinx Zynq 7010)



Xilinx Zynq*

** part of series-7 architecture (Artix-7, Virtex7, Spartan7 …)*

- SOC that combines CPU + FPGA
  - For decades, FPGAs get connected through external buses to a main CPU
    PCI, PCI express, SPI …
  - … then Xilinx invents the Zynq SOC: CPU combined with a FPGA
  - Using **FSBL** and **Uboot** ⇒ Linux booting :)
  - dynamic reconfiguration of the FPGA while Linux is running !
    `cat bit_file > /dev/xdevcfg`
  - 2015, Intel acquired FPGA company Altera ⇒ Xeon D started to feature FPGA to overcome the crowdy CPU instructions extensions !
    SSE2, SSE3, SSE4 … SSExx ! instead of fixed extended instructions that won't ever match all specific apps, FPGA enables tailored instructions to perfectly fit your needs !

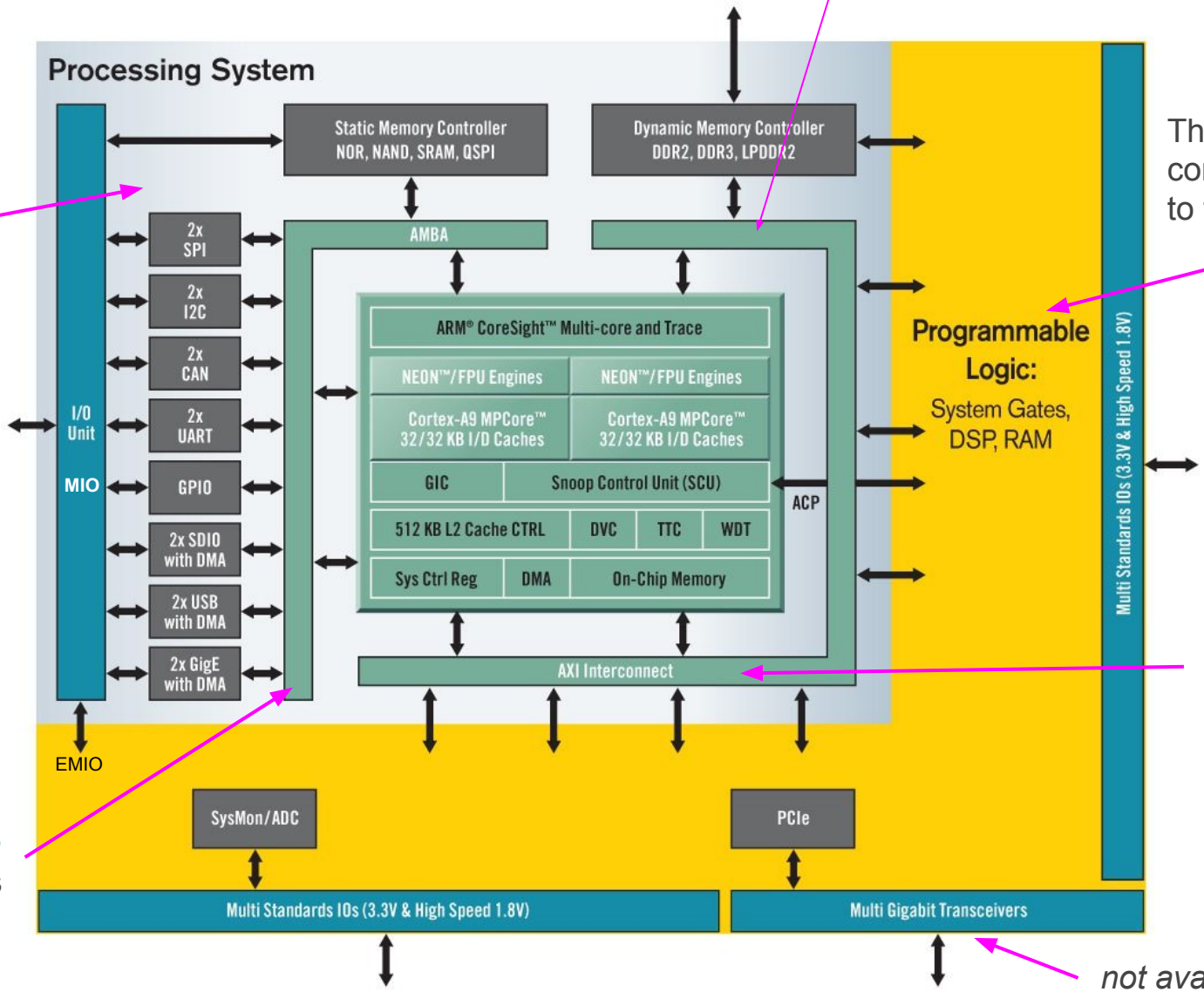
Xilinx Zynq

- Zynq architecture overview

AHB
Amba **H**igh-performance **B**us

PS area
hard IP dual ARM9 with numerous IO controllers.

PL area
The FPGA part that gets connected through AXI bus to the PS area.



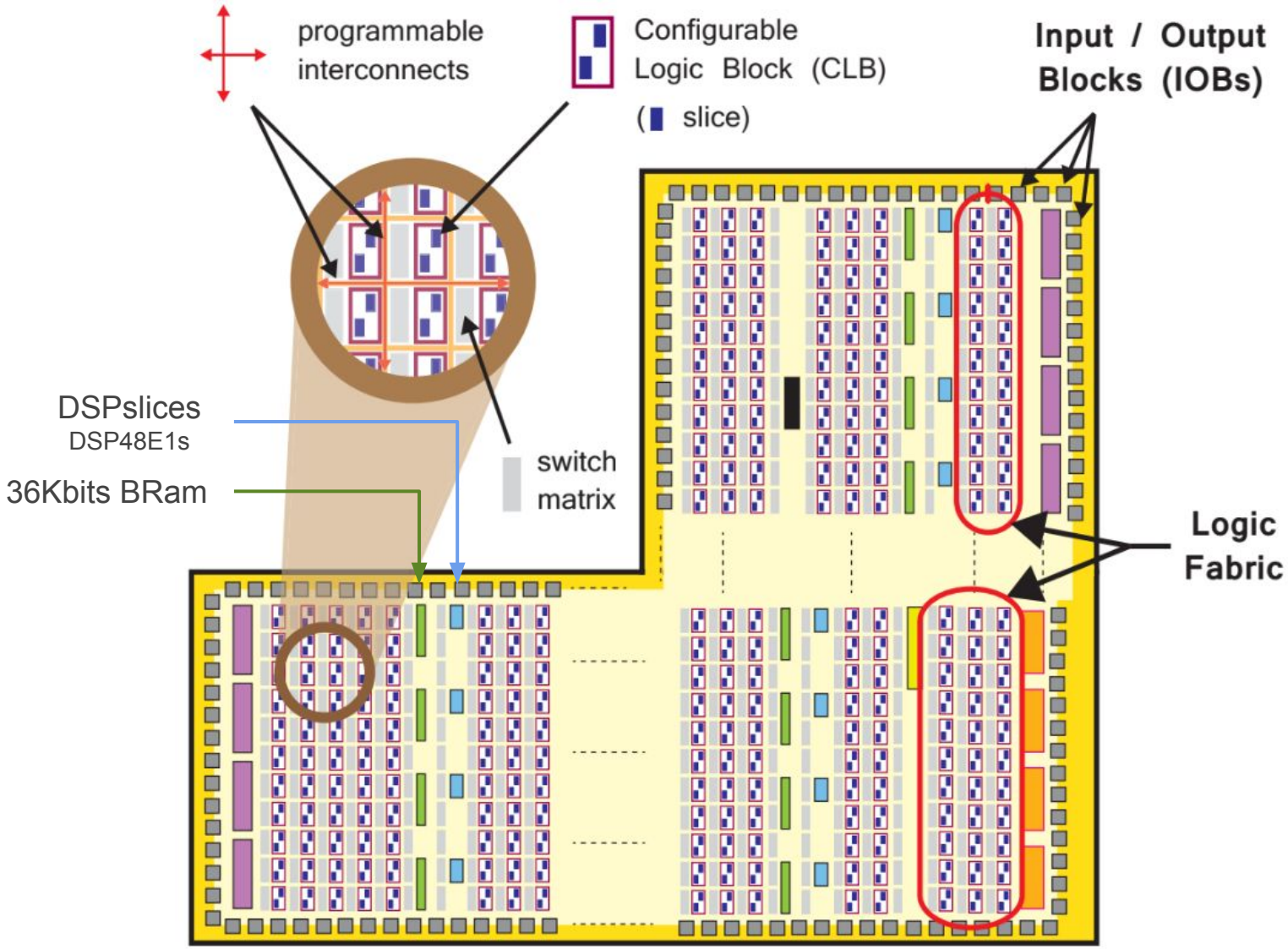
AXI
interconnect through
- **AHB** bridge
- **APB** bridge

APB
Amba **P**eripheral **B**us

not available on all releases!

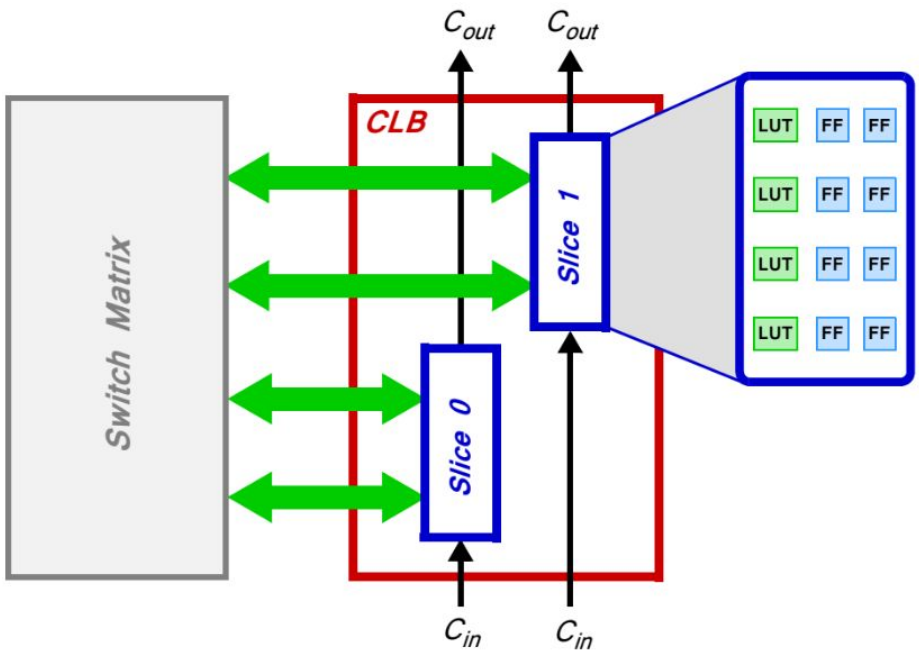
Xilinx Zynq

- Zynq architecture overview - **PL area**



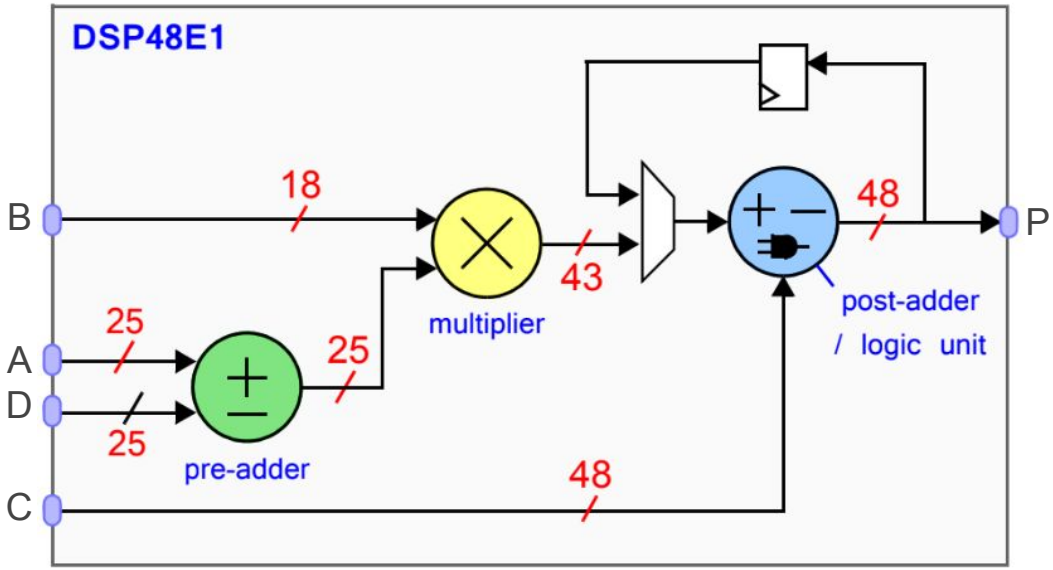
Xilinx Zynq

- Zynq architecture - **CLBs**

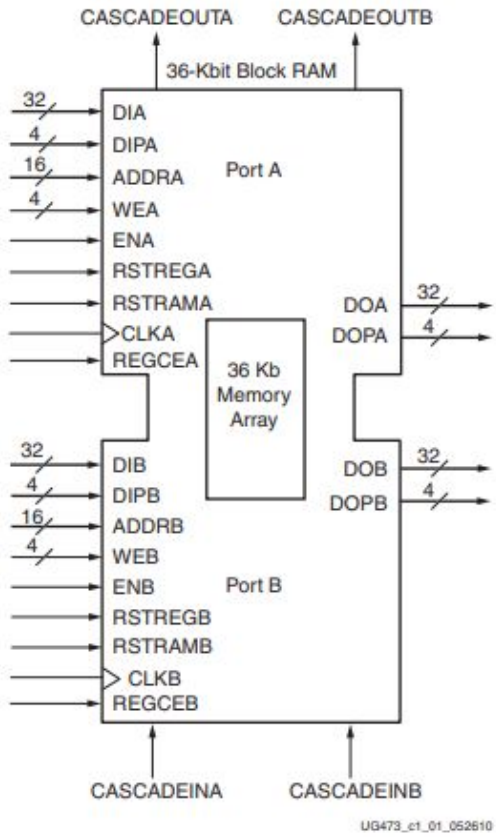


- Zynq architecture - **DSPslices**

The **MAC** function: **M**ultiply **A**nd a**C**cumulate
P = (A+D)*B or P = P' + C



- Zynq architecture - **Block RAM (BRAM)**
  - 36K**bits** configurable as single/dual banks (i.e 36 Kb or 2 x 18 Kb)
  - may get configured with **single** or **dual** read/write ports
  - independent **synchronous read** and write accesses
  - integrated *bypass* policy
    its named WRITE_FIRST
  - may get used as **FIFOs**
    embeds all necessary logic to achieve this.
  - NO NEED FOR AN EXPLICIT inference !
    side example of RAMB36 that can get automagically inferred.



ug473 *Memory Resources user guide*

ug901 *synthesis guide refers as* ***distributed RAM*** *vs* ***Block RAM*** *inferences.*
→ *Distributed RAM consumes LUTs, is a bit faster than dedicated Block RAM but only usable for small amount of RAM*

Zynq®-7000 SoC Family

		Cost-Optimized Devices						Mid-Range Devices			
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System (PS)	Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz			Dual-Core ARM Cortex-A9 MPCore Up to 866MHz			Dual-Core ARM Cortex-A9 MPCore Up to 1GHz[1]			
	Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor									
	L1 Cache	32KB Instruction, 32KB Data per processor									
	L2 Cache	512KB									
	On-Chip Memory	256KB									
	External Memory Support[2]	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support[2]	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to PL)									
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA[2]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security[3]	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts									
Programmable Logic (PL)	7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
	Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Total Block RAM (# 36Kb Blocks)	1.8Mb (50)	2.5Mb (72)	3.8Mb (107)	2.1Mb (60)	3.3Mb (95)	4.9Mb (140)	9.3Mb (265)	17.6Mb (500)	19.2Mb (545)	26.5Mb (755)
	DSP Slices	66	120	170	80	160	220	400	900	900	2,020
	PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC[2]	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[3]	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
	Speed Grades: Commercial	-1			-1			-1			-1
	Speed Grades: Extended	-2			-2,-3			-2,-3			-2
	Speed Grades: Industrial	-1, -2			-1, -2, -1L			-1, -2, -2L			-1, -2, -2L

Links

- additional resources

The Zynq book
http://www.zynqbook.com/
Zynq Tutorials Handbook
http://www.zynqbook.com/download-tuts.html

Digilent Zybo reference manual
https://www.xilinx.com/content/dam/xilinx/support/documentation/university/XUP%20Boards/XUPZYBO/documentation/ZYBO_RM_B_V6.pdf
https://m2siame.univ-tlse3.fr/teaching/francois/UE-VHDL/ZYBO-Z7-reference-manual-B.pdf

ug901: Xilinx synthesis guide
https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2021_2/ug901-vivado-synthesis.pdf

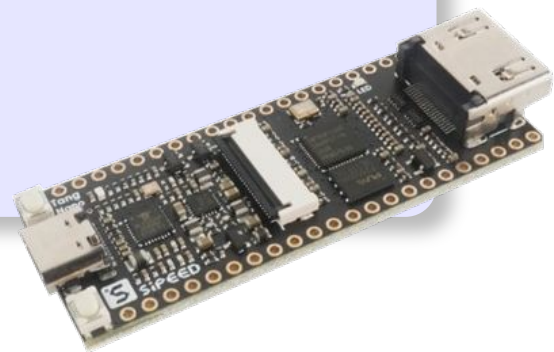
Understanding the Xilinx Zynq **F**irst **S**tage **B**oot **L**oader (FSBL)
https://www.xilinx.com/video/soc/understanding-the-zynq-fsbl.html
*(very first step, then **U-boot** as second step on way to booting Linux for example :D)*

IUT Nantes / GEii / Eric Peronnin Youtube channel
https://www.youtube.com/c/EricPeronnin/videos
Découverte de Vivado part2 https://www.youtube.com/watch?v=djMAXkvw7UI

Plan

Part II – Synthesis for FPGA targets

- Introduction,
- VHDL logic synthesis,
- Xilinx Zynq,
- GoWin GW1NSR-xC (cortex M3+FPGA),

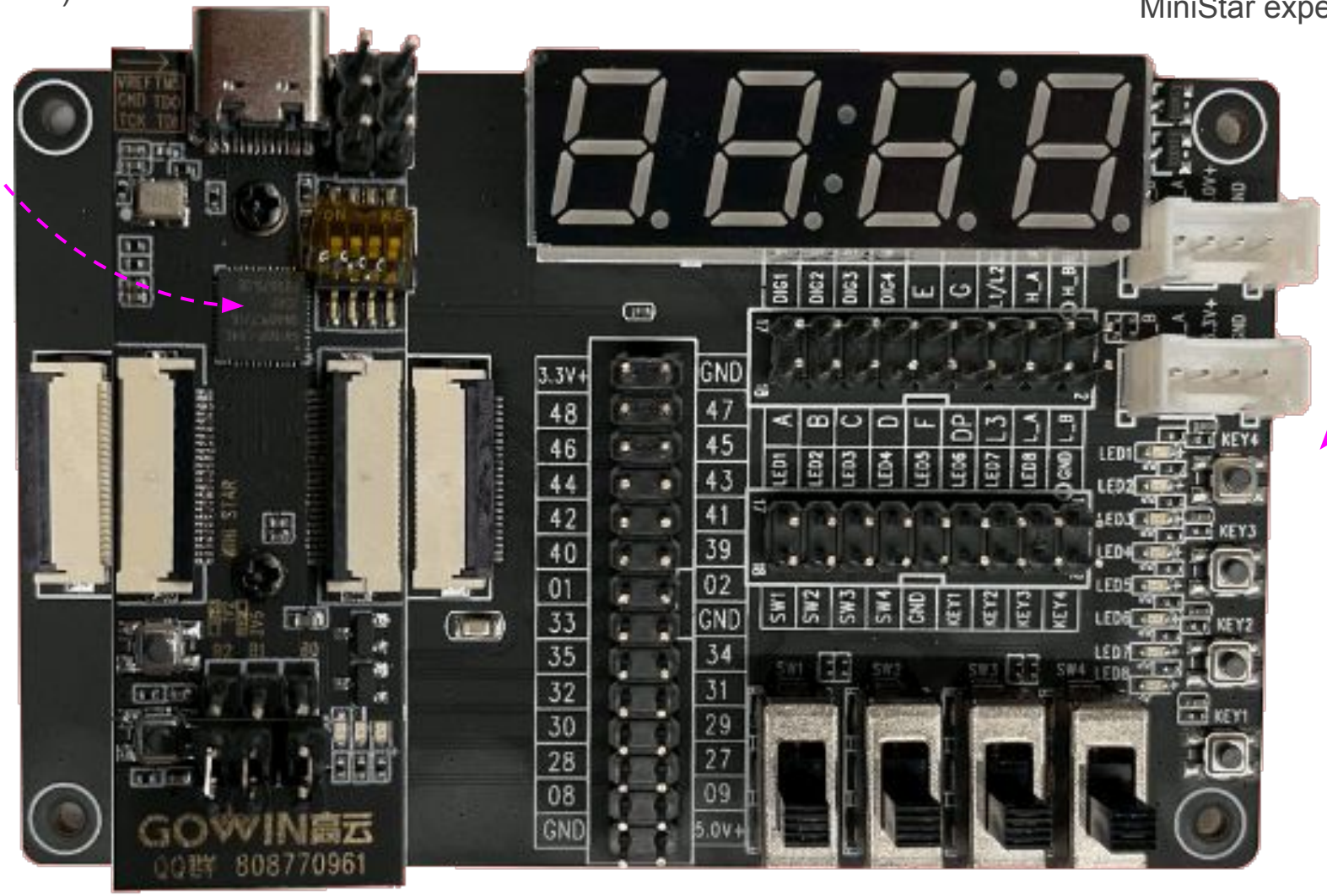


Sipeed Tang Nano 4K, GoWin GW1NSR-LV4C

GoWIN MiniStar dev kit

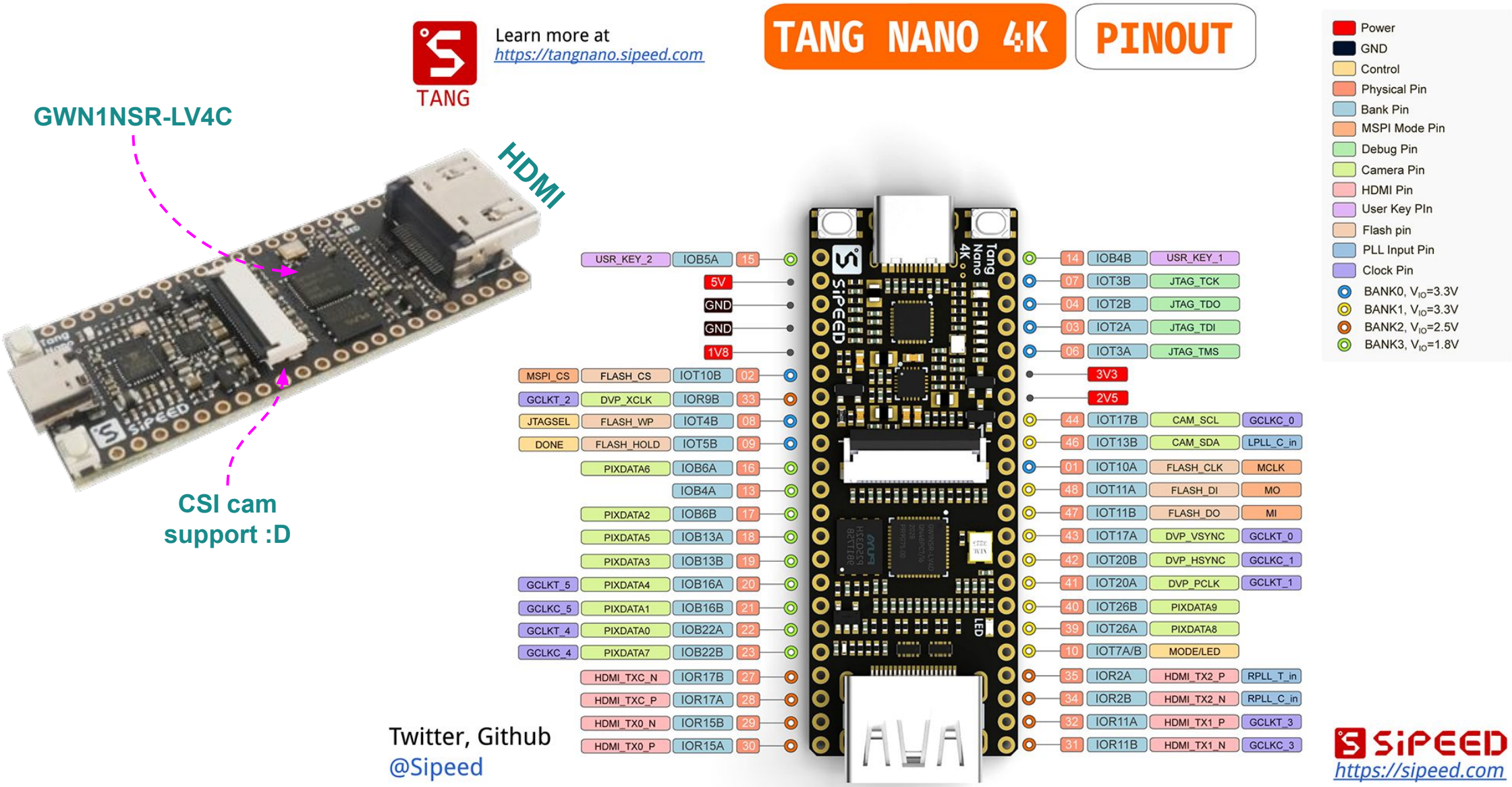
MiniStar board
(**GWN1NSR-LV4C**)

MiniStar experiment carrier board



GoWin Ministar CortexM3+FPGA
https://github.com/magicjellybeanfpga/MiniStar

Sipeed Tang Nano 4K



GoWin GW1NSR-xC

- entry-level SOC that combines ARM CortexM3 + FPGA
  - Please note this is a very preliminary course support about this device,
  - You can use our GoWIN floating license server, even from home :)
  - `arm-none-eabi-gcc` to compile code for the Hard IP Cortex M3,
  - Sipeed Tang Nano 4K is available online,
  - Q3-23, new RISC-V hard IP @ FPGA → GW5AST (AE350_SOC)

GoWin GW1NSR-xC

Device	GW1NSR-2	GW1NSR-2C	GW1NSR-4	GW1NSR-4C
LUT4	1,728	1,728	4,608	4,608
Flip-Flop (FF)	1,296	1,296	3,456	3,456
Block SRAM BSRAM (bits)	72K	72K	180K	180K
BSRAM quantity BSRAM	4	4	10	10
18 x 18 Multiplier			16	16
User Flash (bits)	1M	1M	256K	256K
PSRAM (bits)	32M	32M	64M	64M
HyperRAM(bits)	-	-	-	64M
NOR FLASH (bits)	-	-	-	32M
PLLs+DLLs	1+2	1+2	2+2	2+2
OSC	1,±5% accuracy	1, ±5% accuracy	1,±5% accuracy	1,±5% accuracy
Hard core processor	-	Cortex-M3	-	Cortex-M3
USB PHY	USB 2.0 PHY	USB 2.0 PHY	-	-
ADC[1]	1	1	-	-
Total number of I/O banks	4	4	4	4
Max. I/O	102	102	106	106
Core voltage	1.2V	1.2V	1.2V	1.2V

[dec.21] excerpt from GoWin GW1NSR handbook

Links

- additional resources

FAE Redtree solution GoWIN introduction
https://redtree-solutions.com/images/pdf/fpga-pld-basics-Gowin-webex-202106.pdf

GW1NSR datasheets
http://cdn.gowinsemi.com.cn/DS861E.pdf

[Risc-V] AE350_SOC
https://github.com/riscv-software-src/opensbi/blob/master/docs/platform/andes-ae350.md

Links

- **RISC-V** additional resources

RISC-V open specifications
https://riscv.org/technical/specifications/
RISC-V ISA (2019) https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

alibaba to open its RISC-V implementations (verilog) !
https://github.com/T-head-Semi
https://www.cnx-software.com/2021/10/20/alibaba-open-source-risc-v-cores-xuantie-e902-e906-c906-and-c910/

Microchip HelloFPGA
https://www.microchip.com/en-us/products/fpgas-and-plds
Microchip Mi-V Risc-V SOC FPGA architecture

Thales RISC-V softCore contest
CVA6 risc-v synthesis to Zybo z7-7020 target
https://github.com/ThalesGroup/cva6-softcore-contest

Traces team involved !

OpenHwGroup, Ariane/CVA6 Risc-V core, linux capable, 6 stages pipeline, out-of-order
https://github.com/openhwgroup/cva6

PicoRV32 - a size optimized RISC-V
https://github.com/YosysHQ/picorv32

RISC-V software sources to build various platforms
e.g Andes AE-350 (Gowin GW5AST)
https://github.com/riscv-software-src/opensbi/blob/master/docs/platform/andes-ae350.md

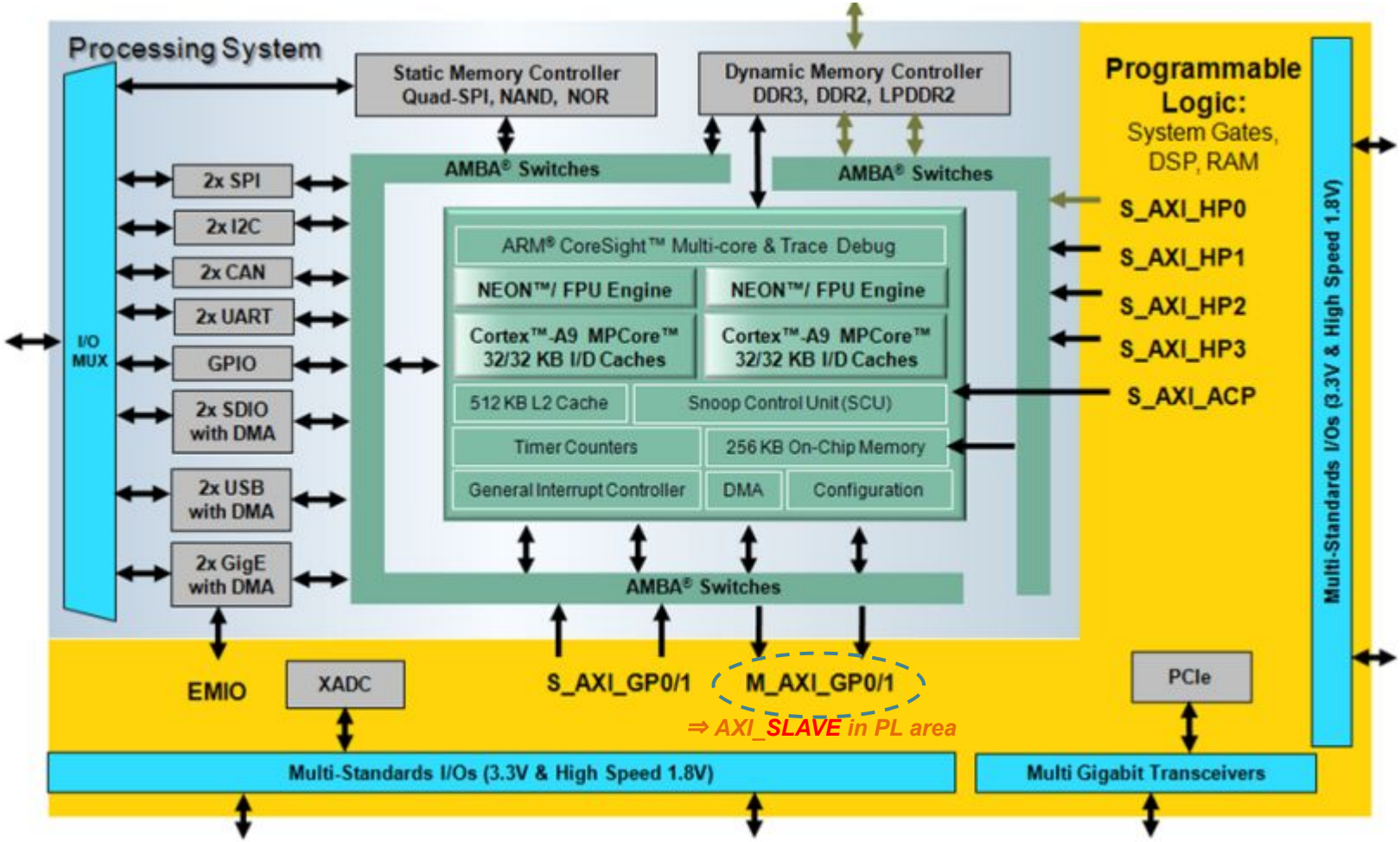
Plan

Part IIa – [Advanced] Xilinx Zynq architecture

- Zynq architecture overview,
- AXI bus,
- Hands-on lab: my first soft IP,
- Next lab: RISC-V integration ?

Zynq architecture overview

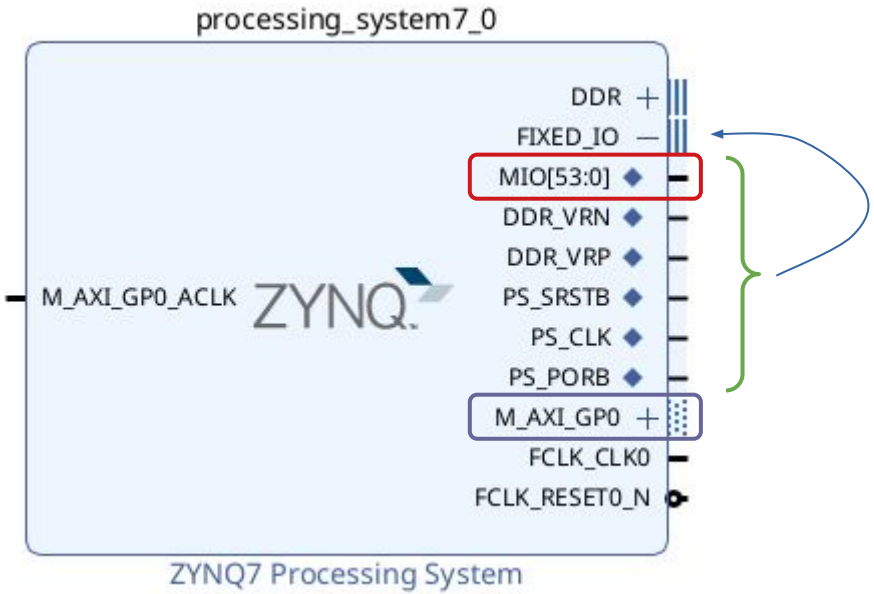
- Zynq SoC block diagram

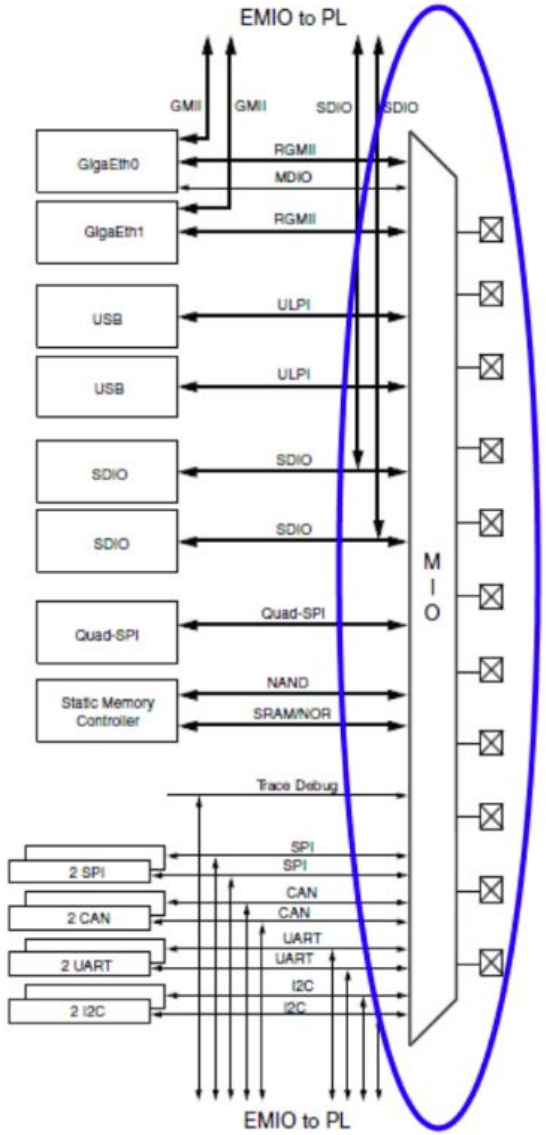


MIO / EMIO

- Multiplexed IO (MIO) are PS dedicated I/O
  - 54 dedicated pins
  - Software configurable
  - IO banks 500 & 501

https://m2siame.univ-tlse3.fr/teaching/francois/UE-M2-VHDL/ZYBO-Z7_schematic-D1.pdf

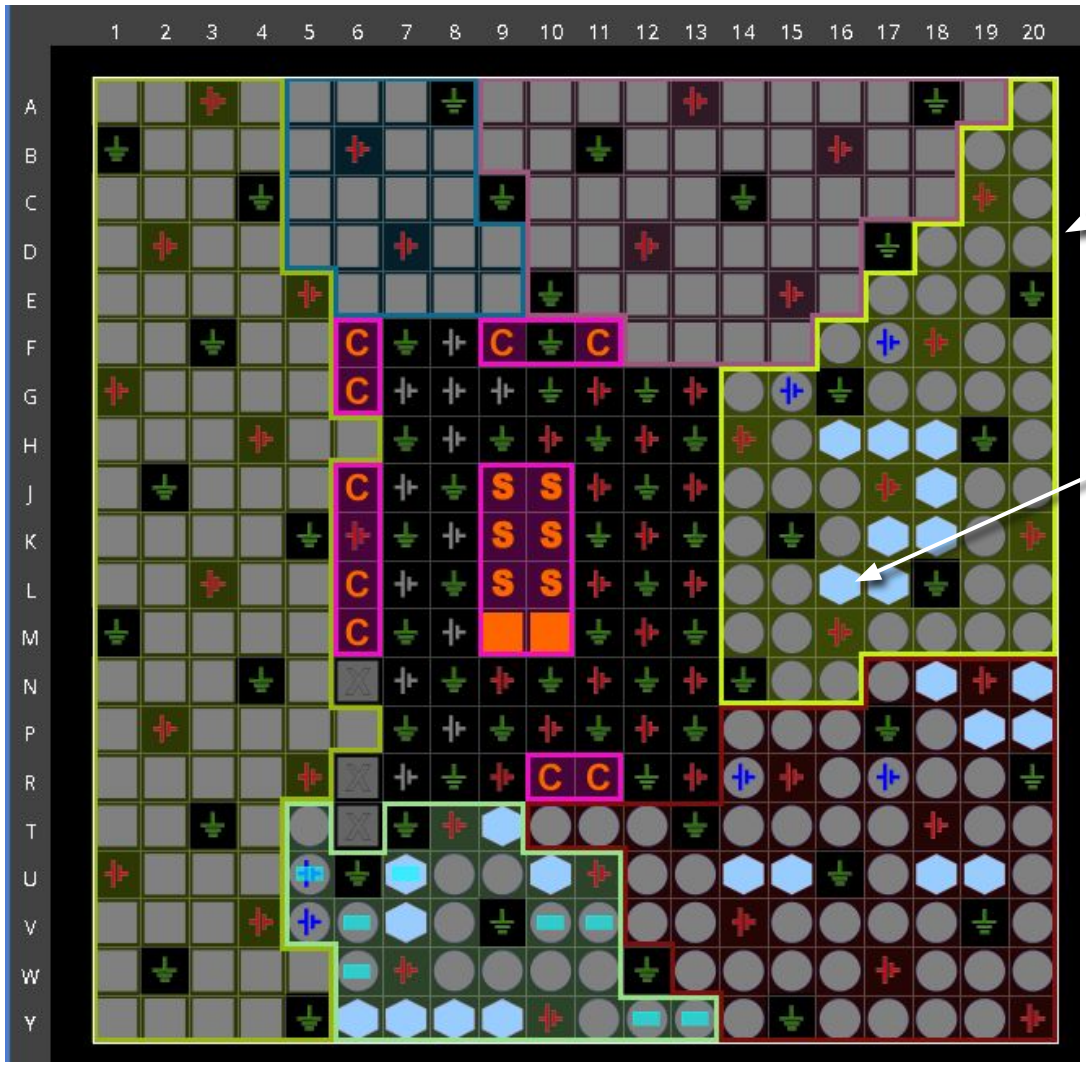




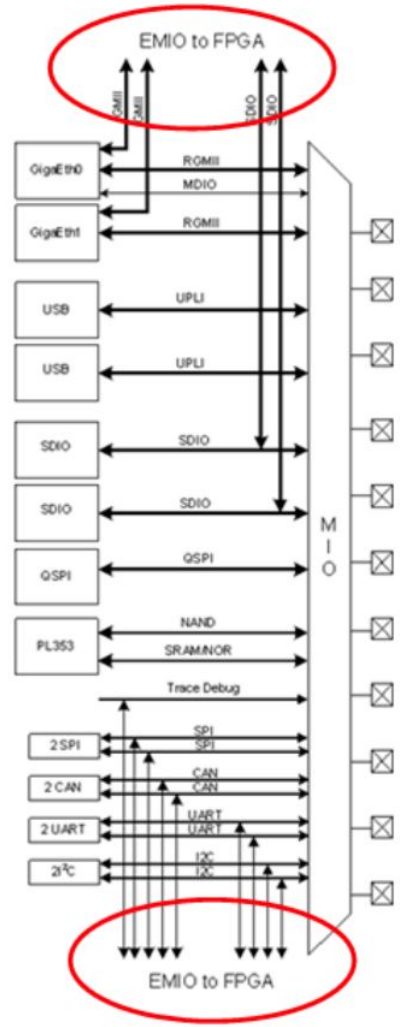
excerpt from Xilinx Zynq architecture slides

MIO / EMIO

- Extended Multiplexed IO (EMIO) enables PL access to PS I/O ports
  - Broader range of I/O through PL accessible banks



Layout view → IO planning from Vivado



excerpt from Xilinx Zynq architecture slides

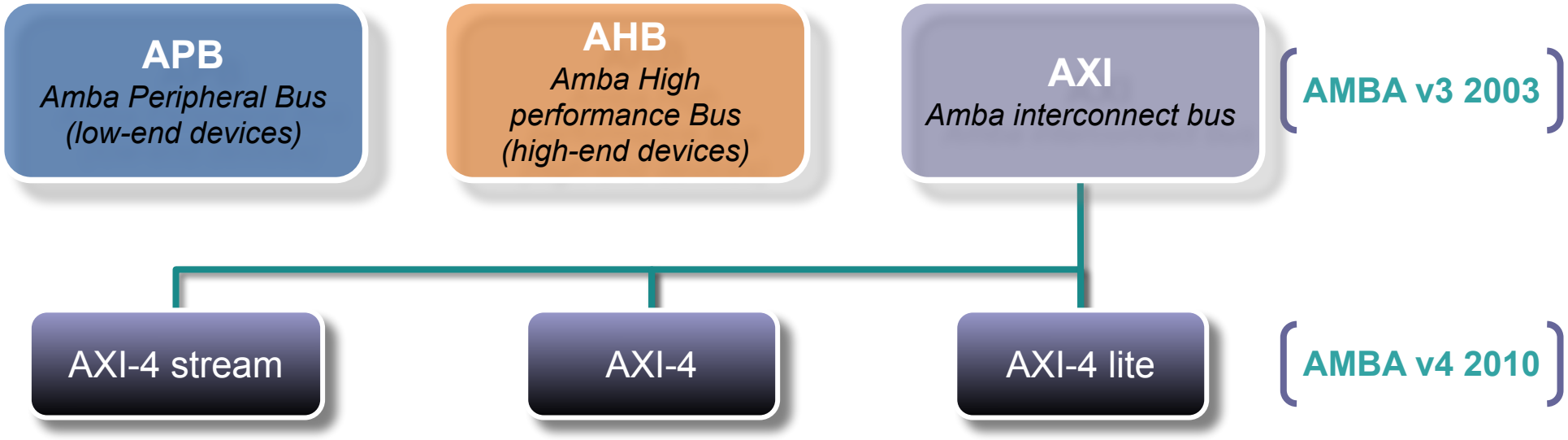
Plan

Part IIa – [Advanced] Xilinx Zynq architecture

- Zynq architecture overview,
- AXI bus,
- Hands-on lab: my first soft IP,
- Next lab: RISC-V integration ?

AXI bus

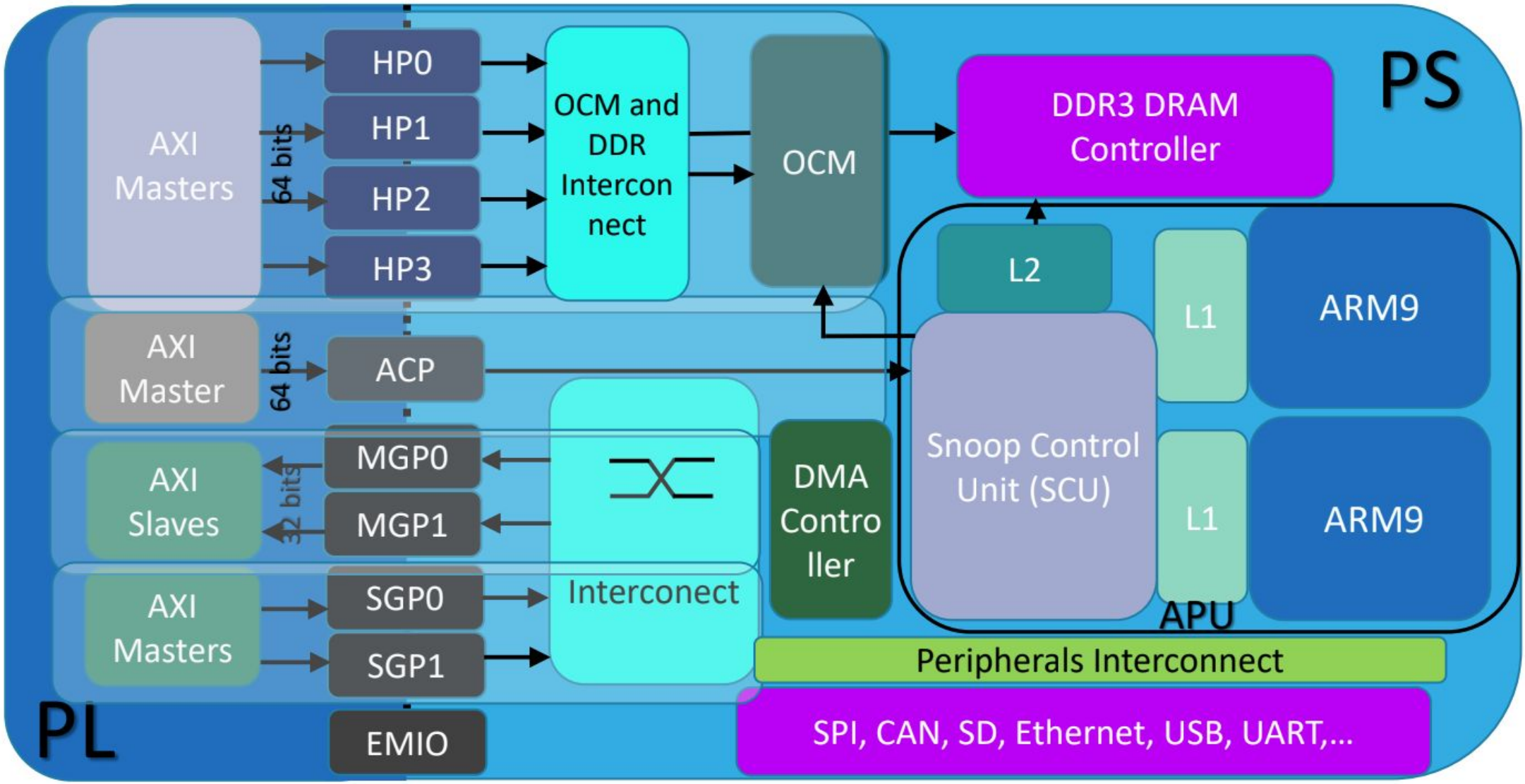
- The Advanced Microcontroller Bus Architecture (AMBA) bus



- up to 1024 bits wide
- up to 256 burst xfers
- 5 channels
- **memory mapped**

- FPGA intended
- 32-bit wide
- up to 512 registers
- **no burst** xfers
- 5 channels
- **memory mapped**
- requires AXI_Interconnect IP

AXI bus @ Zynq



Excerpt from Cristian Sisterna slide 21 https://m2siame.univ-tlse3.fr/teaching/francois/UE-M2-VHDL/Sisternal_Zynq-architecture_C7T.pdf

AXI bus @ Zynq

- PS - PL interface: the **AXI** bus
  - AMBA evolution
  - memory mapped registers

AXI high-performance slave ports (HP0-HP3)

- Configurable 32-bit or 64-bit data width
- Access to OCM and DDR only
- Conversion to processing system clock domain
- AXI FIFO Interface (AFI) are FIFOs (1KB) to smooth large data transfers

AXI general-purpose ports (GP0-GP1)

- Two masters from PS to PL
- Two slaves from PL to PS
- 32-bit data width
- Conversion and sync to processing system clock domain

AXI bus @ Zynq

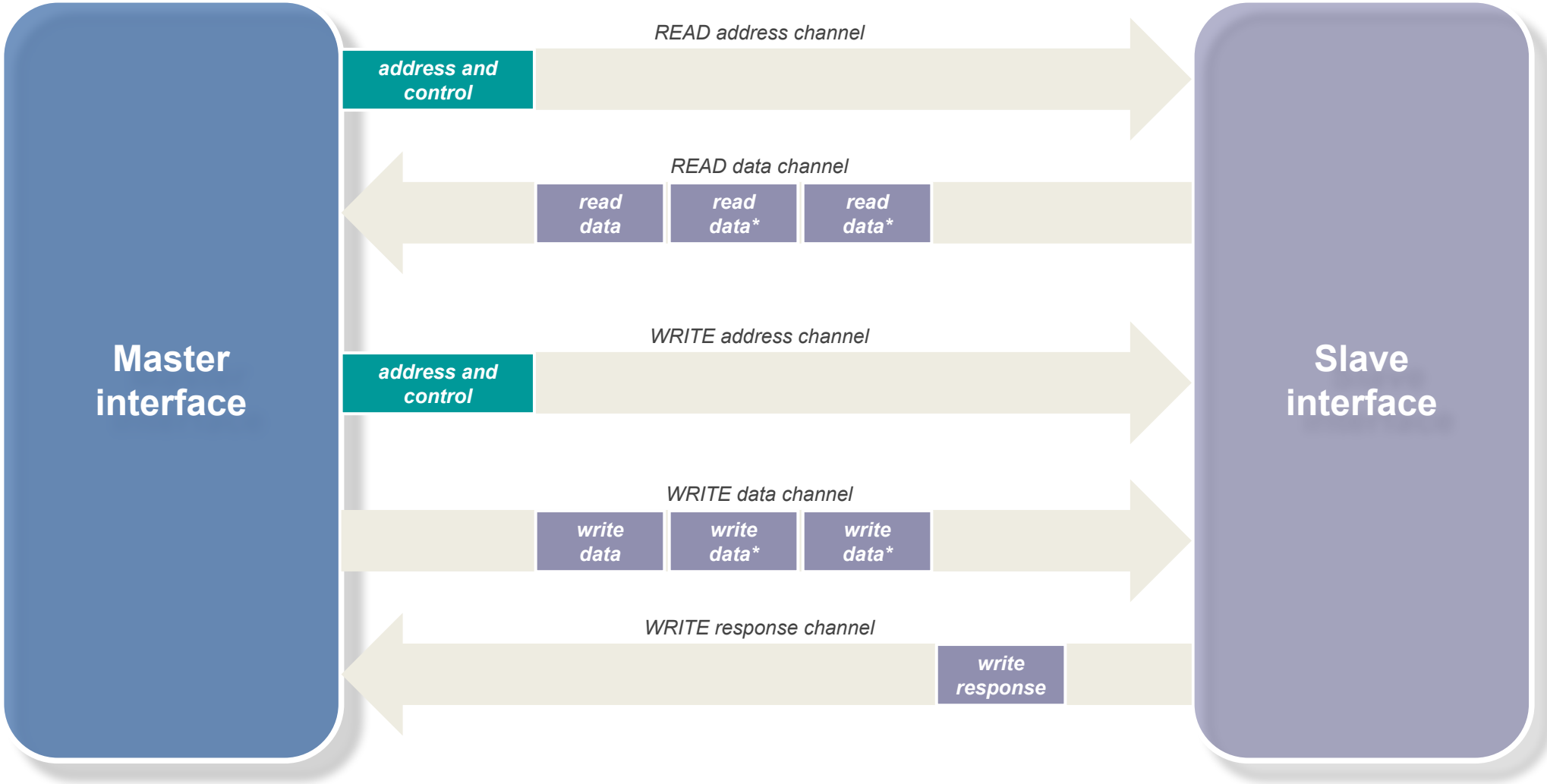
- PS - PL **AXI** interfaces synthesis

Interface Name	Interface Description	Master	Slave
M_AXI_GP0	General Purpose (AXI_GP)	PS	PL
M_AXI_GP1		PS	PL
S_AXI_GP0	General Purpose (AXI_GP)	PL	PS
S_AXI_GP1		PL	PS
S_AXI_ACP	Accelerator Coherency Port (ACP), cache coherent transaction	PL	PS
S_AXI_HP0	High Performance Ports (AXI_HP) with read/write FIFOs. (Note that AXI_HP interfaces are sometimes referred to as AXI Fifo Interfaces, or AFIs).	PL	PS
S_AXI_HP1		PL	PS
S_AXI_HP2		PL	PS
S_AXI_HP3		PL	PS

Excerpt from https://m2siame.univ-tlse3.fr/teaching/francois/UE-M2-VHDL/The_Zynq_Book_ebook_3.pdf p.32

AXI bus @ Zynq

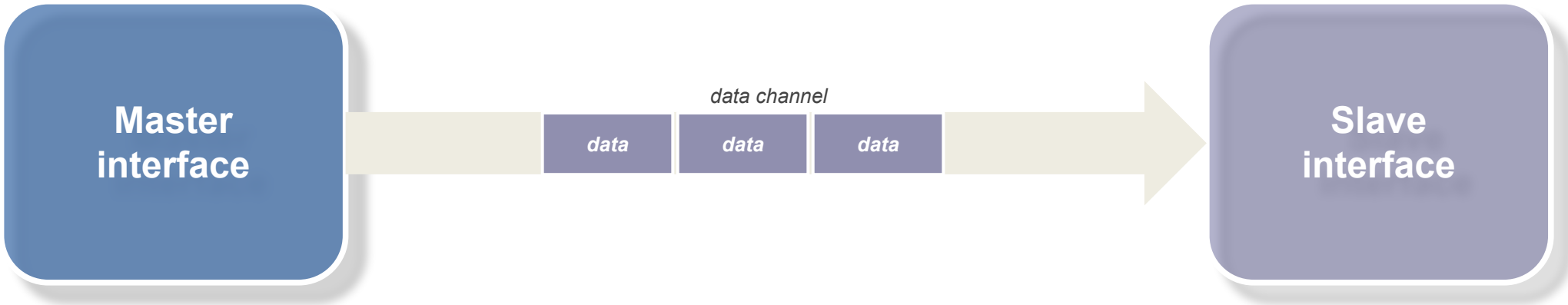
- AXI4 and AXI4-lite bus channels



* no burst mode for AXI4-lite

AXI bus @ Zynq

- AXI4-stream bus channels
  - AXI4 "write data" channel
  - unlimited burst length



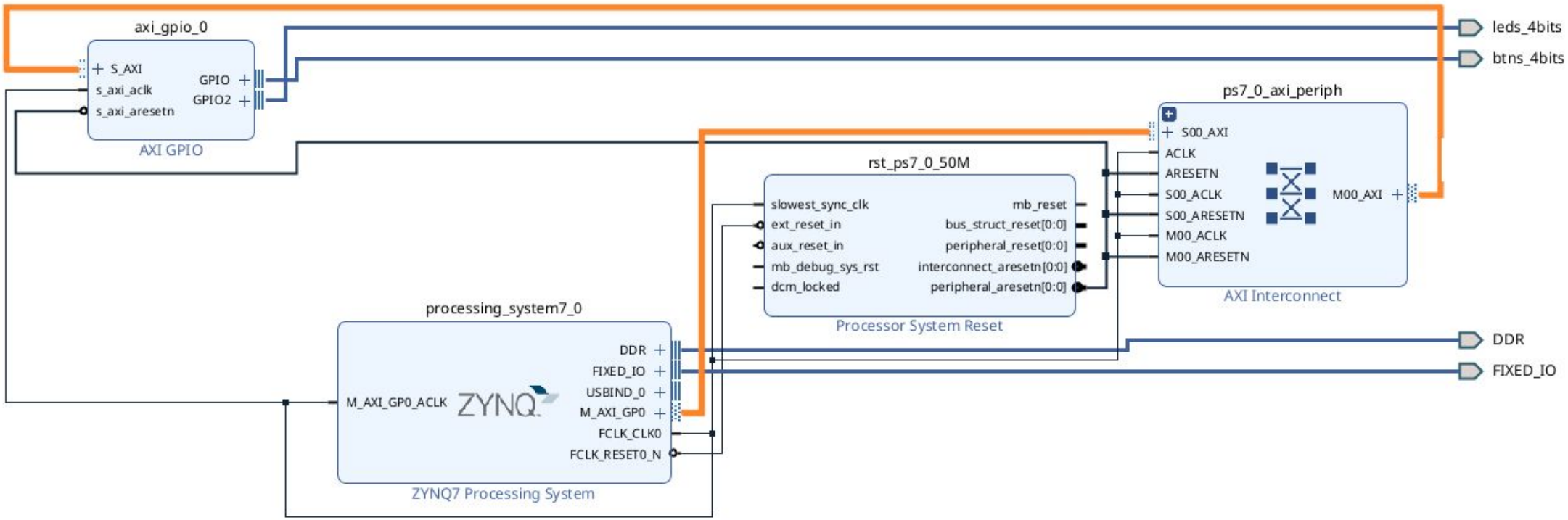
AXI bus @ Zynq

- AXI VHDL interface

Fixing Xilinx's Broken AXI-lite Design in VHDL
https://zipcpu.com/blog/2021/05/22/vhdlaxil.html
[sample] https://github.com/ZipCPU/wb2axip/blob/master/bench/formal/xlnxdemo_2020_2.vhd

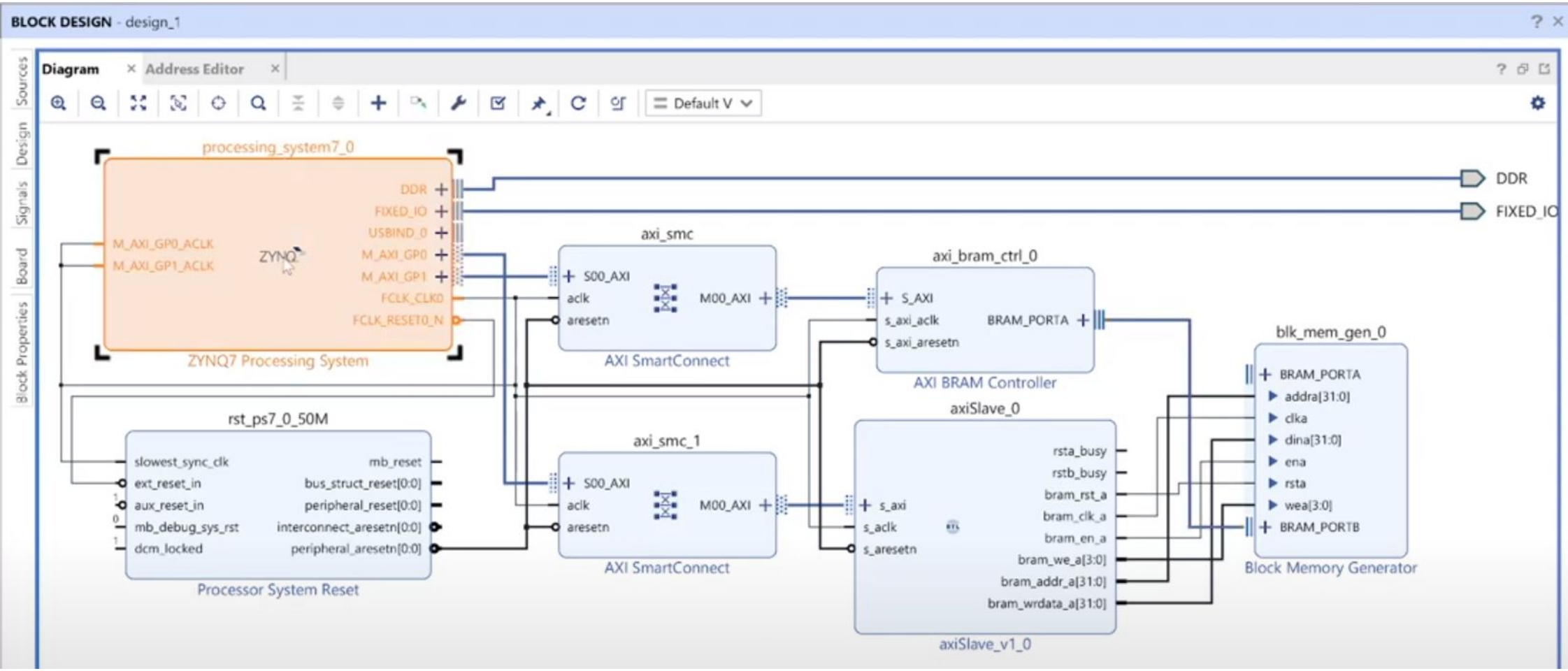
Zynq PS-PL

- AXI bus @ Vivado



Zynq PS-PL

- AXI bus @ Vivado | new **AXI_smc** + AXI_slave https://www.youtube.com/watch?v=UZ3yEFdgFVs



AXI SmartConnect (axi_smc)
https://www.xilinx.com/products/intellectual-property/smartconnect.html
AXI_smc is the AXI_Interconnect successor
WARNING: axi_smc is not intended to AXI4-lite !

Zynq PS Memory Map

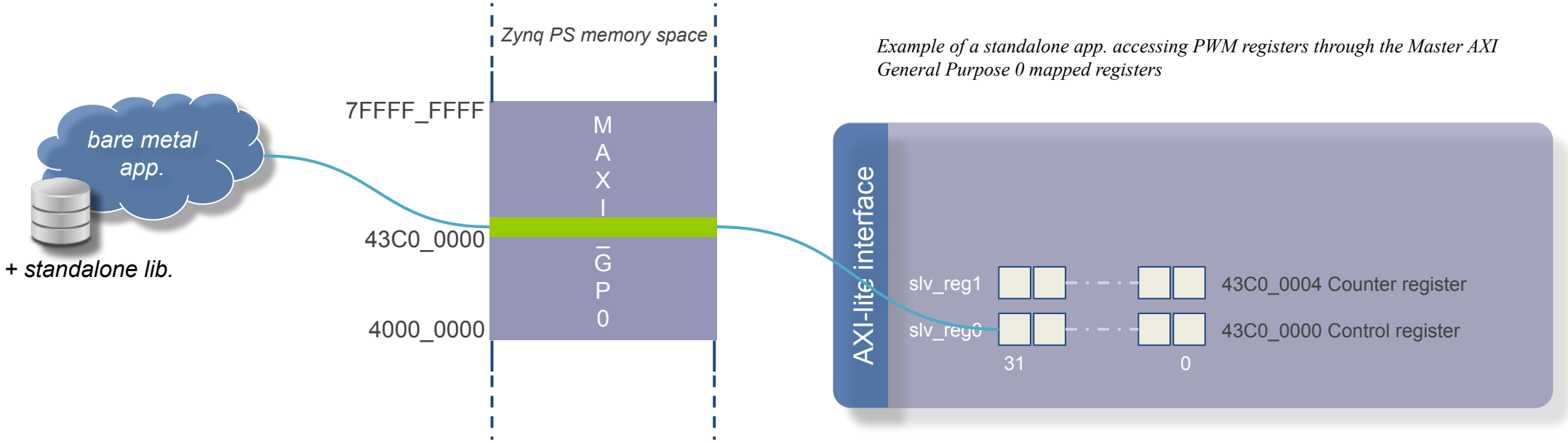
- Memory Map

- **The Cortex-A9 processor uses 32-bit addressing**
- **All PS peripherals and PL peripherals are memory mapped to the Cortex-A9 processor cores**
- **All slave PL peripherals will be located between 4000_0000 and 7FFF_FFFF (connected to GP0) and 8000_0000 and BFFF_FFFF (connected to GP1)**

FFFC_0000 to FFFF_FFFF	OCM
FD00_0000 to FFFB_FFFF	Reserved
FC00_0000 to FCFF_FFFF	Quad SPI linear address
F8F0_3000 to FBFF_FFFF	Reserved
F890_0000 to F8F0_2FFF	CPU Private registers
F801_0000 to F88F_FFFF	Reserved
F800_1000 to F880_FFFF	PS System registers,
F800_0C00 to F800_0FFF	Reserved
F800_0000 to F800_0BFF	SLCR Registers
E600_0000 to F7FF_FFFF	Reserved
E100_0000 to E5FF_FFFF	SMC Memory
E030_0000 to E0FF_FFFF	Reserved
E000_0000 to E02F_FFFF	IO Peripherals
C000_0000 to DFFF_FFFF	Reserved
8000_0000 to BFFF_FFFF	PL (MAXI _GP1)
4000_0000 to 7FFF_FFFF	PL (MAXI _GP0)
0010_0000 to 3FFF_FFFF	DDR(address not filtered by SCU)
0004_0000 to 000F_FFFF	DDR(address filtered by SCU)
0000_0000 to 0003_FFFF	OCM

Zynq PS Memory Map

- AXI IP Memory Map setup
  - we'll setup base address of an AXI4 IP
  - then we'll define the number of mapped registers
  - hence accessing hardware is just a matter of memory accesses :)



Example of a standalone app. accessing PWM registers through the Master AXI General Purpose 0 mapped registers

Links

- **Xilinx** additional related links

AXI IP creation
https://indico.ictp.it/event/a14283/session/62/contribution/252/material/slides/2.pdf

Create Custom AXI Cores
https://www.hackster.io/cospan/create-custom-axi-cores-part-1-straight-to-the-finish-line-a70e5e

Designing a Custom AXI IP on Vitis (software part while Vivado is HW part)
https://www.hackster.io/pablotrujillojuan/designing-a-custom-axi-ip-on-vitis-a0ad06

Zybo-z7-20 AXI_SMC + AXI_SLAVE (2022)
https://www.youtube.com/watch?v=UZ3yEFdgFVs

M2secil additional resources
https://m2siame.univ-tlse3.fr/info/vhdl
→ have a look to Cristian Sisterna's slides and the "Xilinx Zynq Architecture" slides from Xilinx

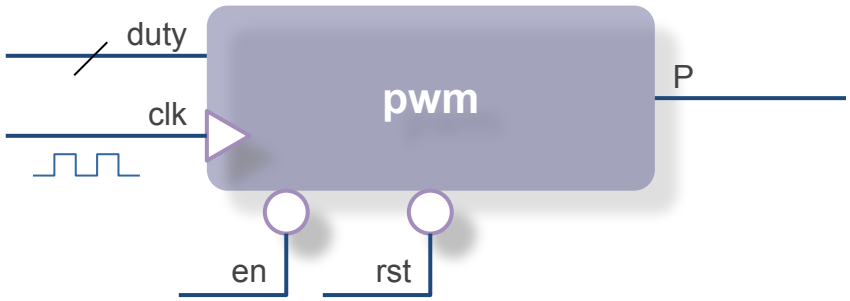
Plan

Part IIa – [Advanced] Xilinx Zynq architecture

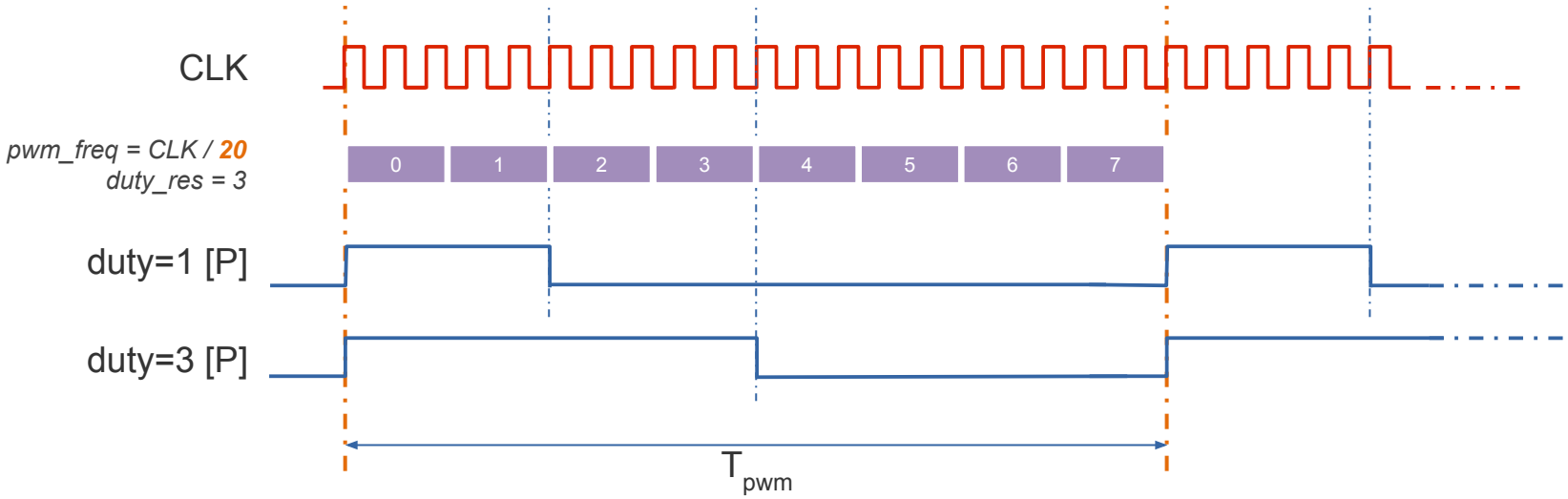
- Zynq architecture overview,
- AXI bus,
- Hands-on lab: my first soft IP,
- Next lab: RISC-V integration ?

[Lab] PWM generator

- generate a 1s pulse from the 125 MHz Ethernet clock @ Zybo
  - adjustable duty cycle + enable/disable cmd
    hint: you may use Zybo's switches

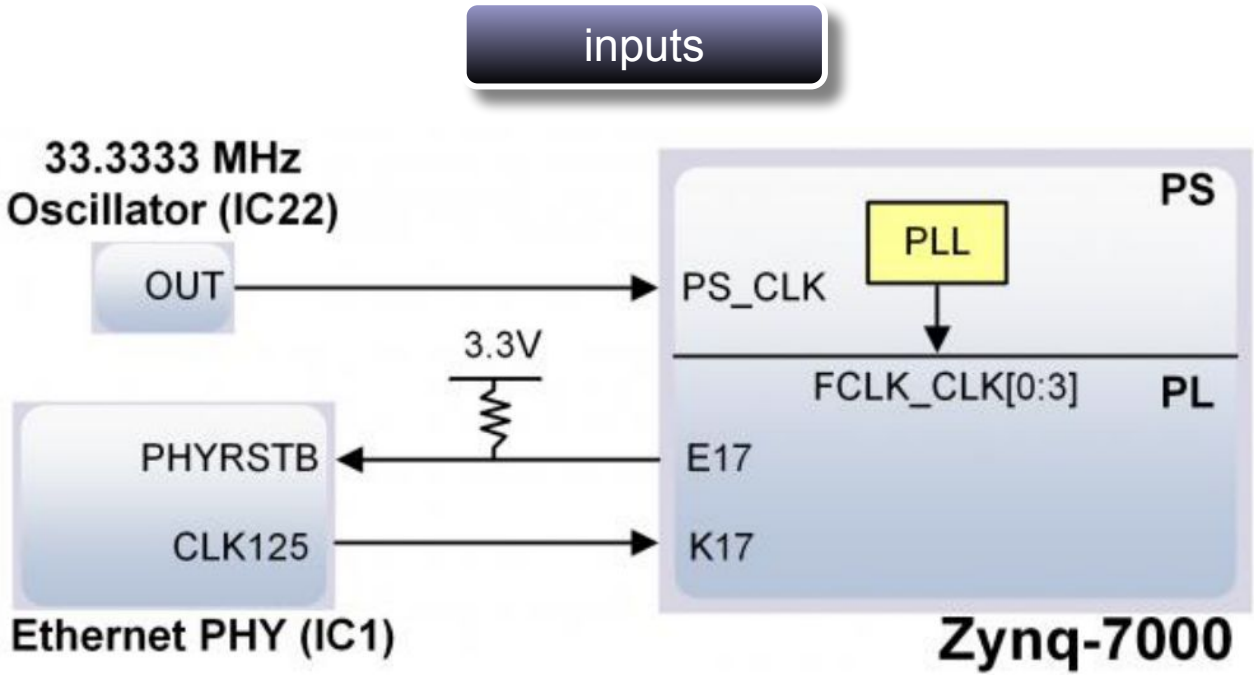



```
entity pwm is
  generic(
    sys_clk : natural := 1_000_000; --system clock in Hz
    pwm_freq: natural := 1E5; --PWM frequency in Hz
    duty_res : natural := 8 --duty-cycle resolution
  );

  port( RST,EN,CLK:in std_logic;
        DUTY: in std_logic_vector(duty_res-1 downto 0);
        P:out std_logic );
end pwm;
```

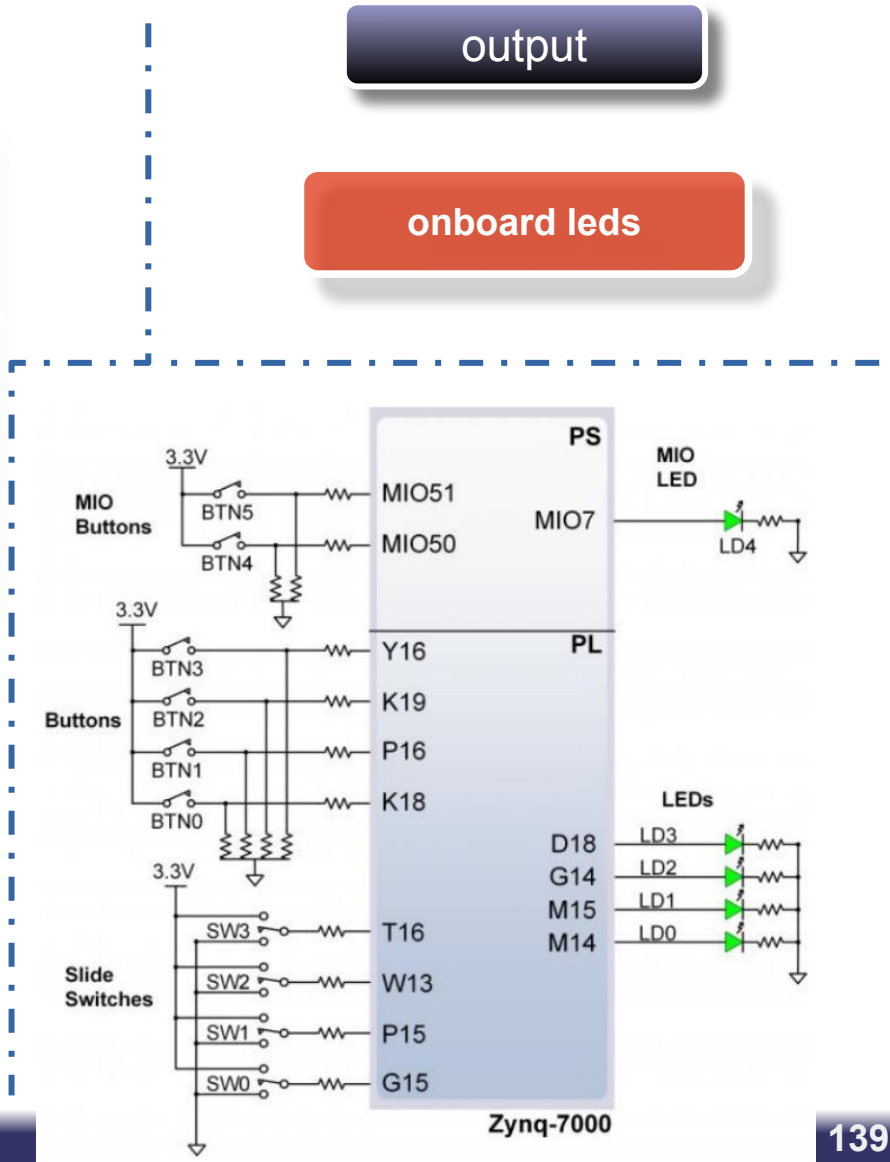


[Lab] PWM generator

- generate a 1s pulse from the 125 MHz Ethernet clock @ Zybo



onboard switches



[Lab] PWM generator

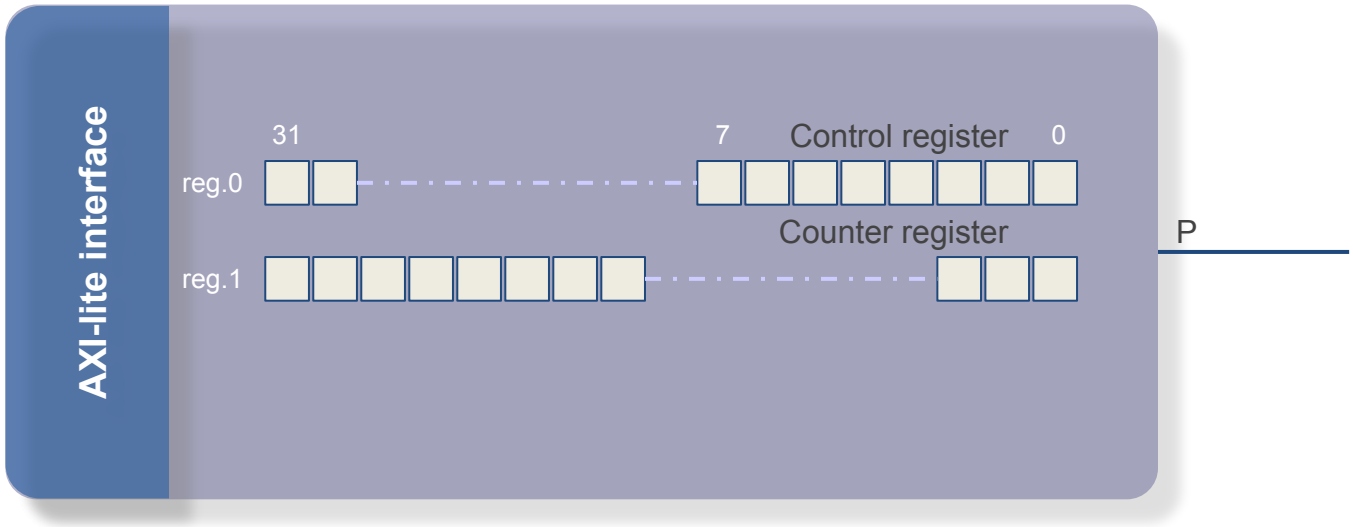
- generate a 1s pulse from the 125 MHz Ethernet clock @ Zybo

`pwm.xdc` constraint file

```
# Master Clock timing constraint
create_clock -period 8.000 -name CLK -waveform {0.000 4.000} [get_ports CLK]
set_property PACKAGE_PIN K17 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]

set_property PACKAGE_PIN K18 [get_ports EN]
set_property IOSTANDARD LVCMOS33 [get_ports EN]
set_property PACKAGE_PIN P16 [get_ports RST]
set_property IOSTANDARD LVCMOS33 [get_ports RST]

set_property PACKAGE_PIN G15 [get_ports {DUTY[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[0]}]
set_property PACKAGE_PIN P15 [get_ports {DUTY[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[1]}]
set_property PACKAGE_PIN W13 [get_ports {DUTY[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[2]}]
set_property PACKAGE_PIN T16 [get_ports {DUTY[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[3]}]

set_property PACKAGE_PIN D18 [get_ports P]
set_property IOSTANDARD LVCMOS33 [get_ports P]
```

[Lab] PWM generator Soft-IP

- AXI compliant PWM generator
  - AXI-enabled internals registers to control and monitor

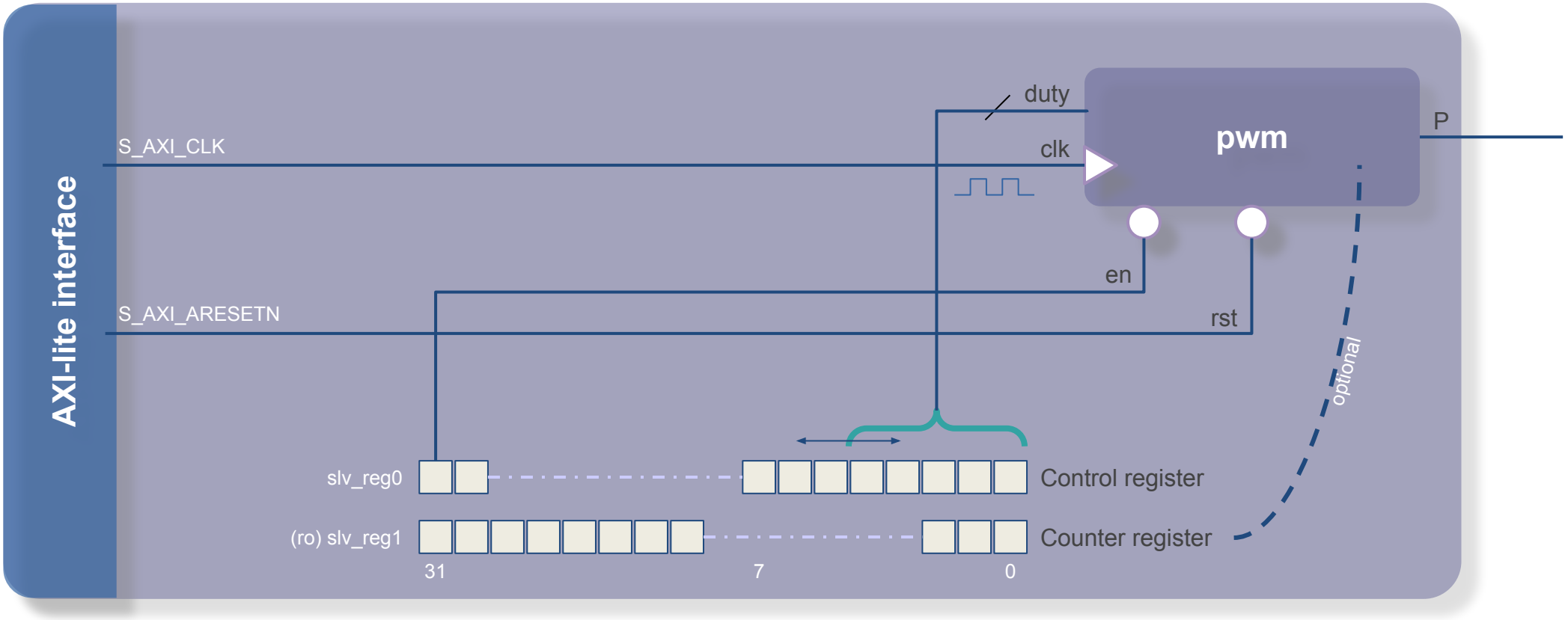


AXI IP creation
https://indico.ictp.it/event/a14283/session/62/contribution/252/material/slides/2.pdf

[Lab] PWM generator Soft-IP

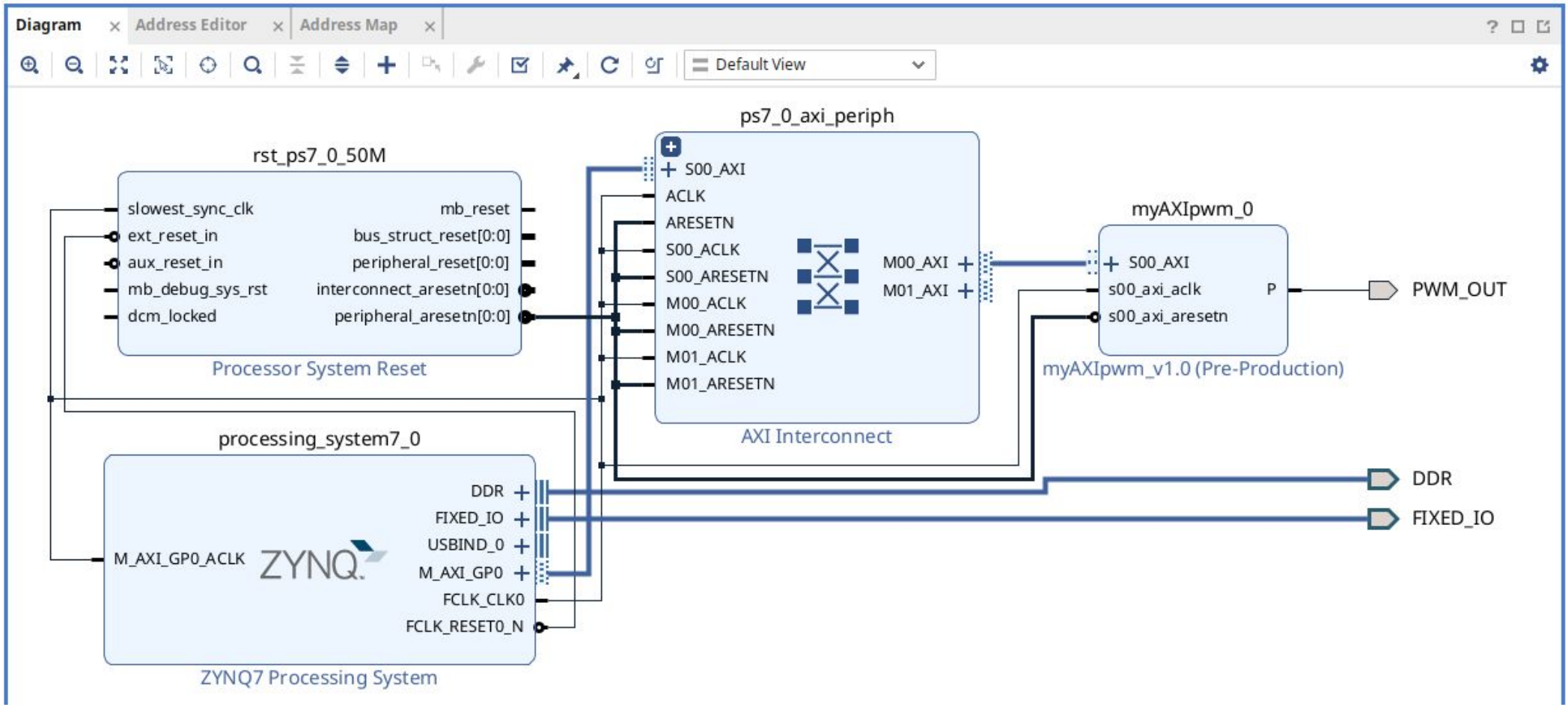
- AXI-PWM IP internals
  - [reg.0] control register **duty-bits** size depends on generic parameter
  - [reg.0] bit 31 is enable(0) or disable(1)
  - [reg.1] counter register (read-only) is your PWM tick counter*

* 2^{32} ⇒ SYS_CLK max > 4GHz



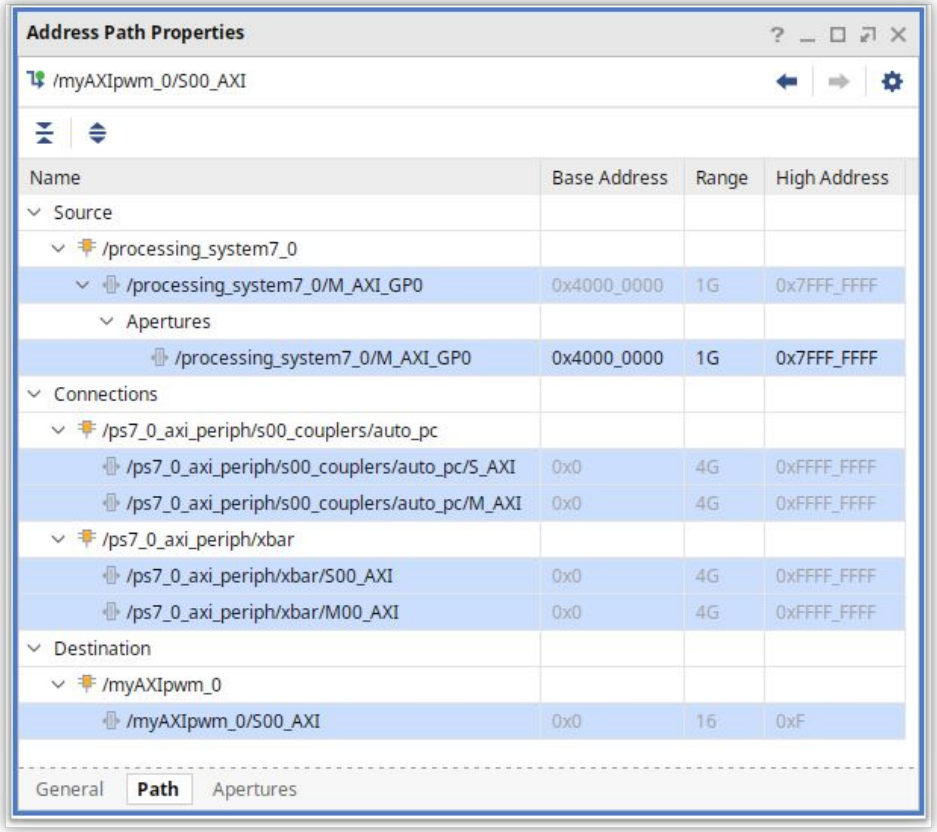
[Lab] Soft-IP block design integration

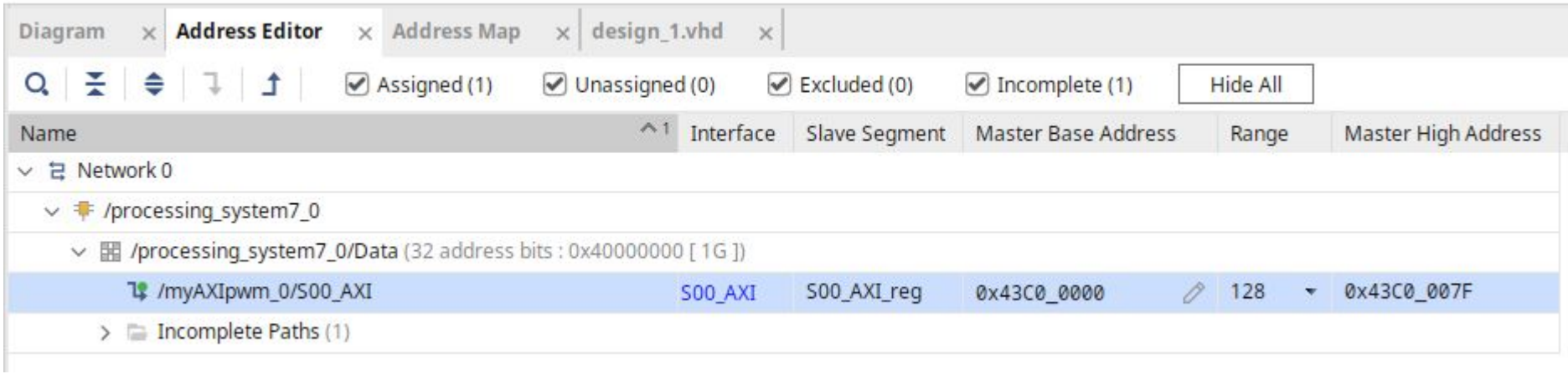
- Generate a Block design featuring:
  - Zynq PS + *generate Block Design (BD)*
  - add your AXI-PWM **Soft-IP** + PWM_OUT port



[Lab] Soft-IP block design integration

- Block Design configuration:
  - PWM **Soft-IP** generic parameters
  - Memory mapped registers base address
  - add constraint: PWM_OUT → D18 pin





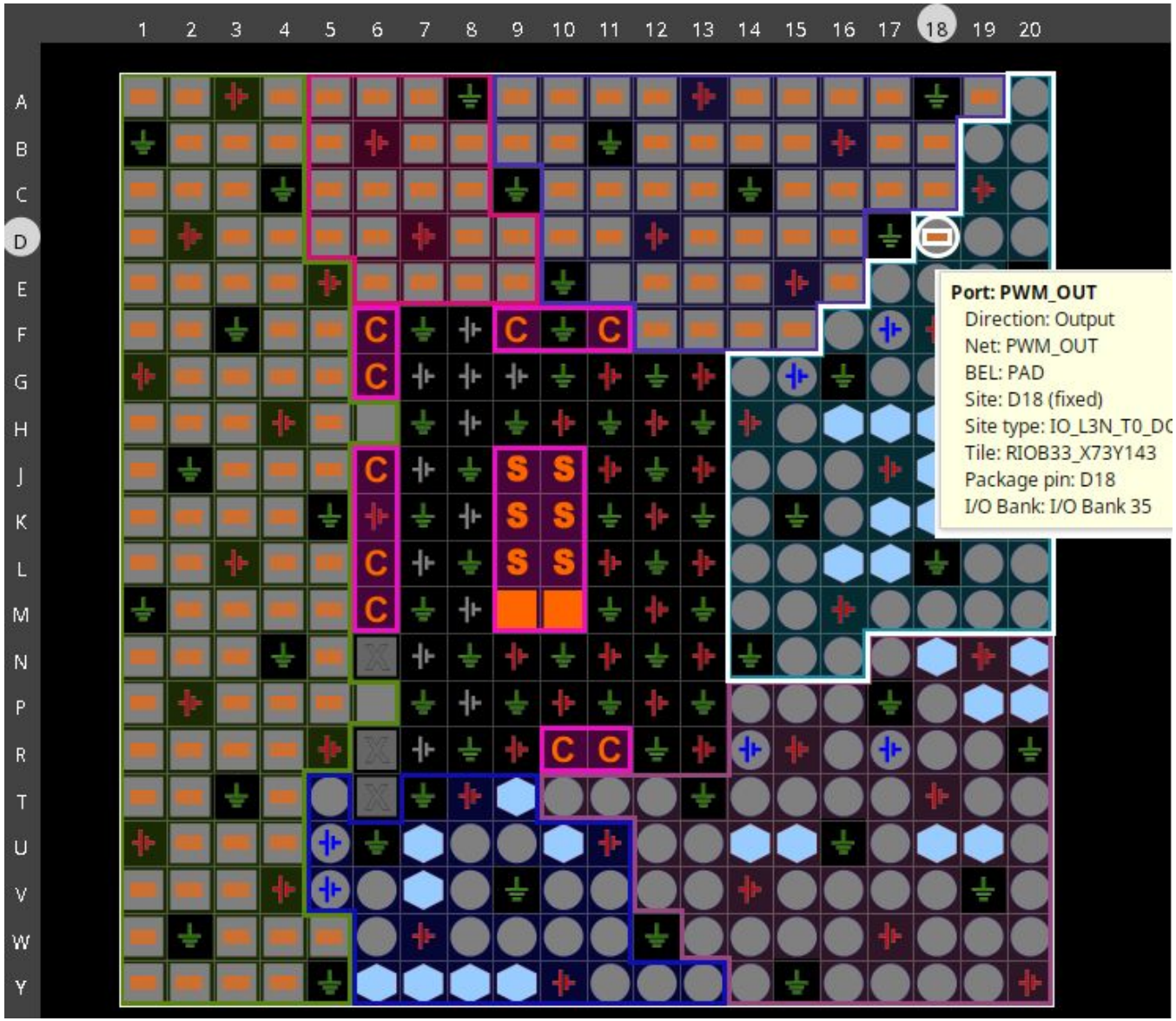
[Lab] Soft-IP block design integration

- Synthesis

  You'll need to create a HDL-wrapper since your design does not feature any top-level component (right-click)

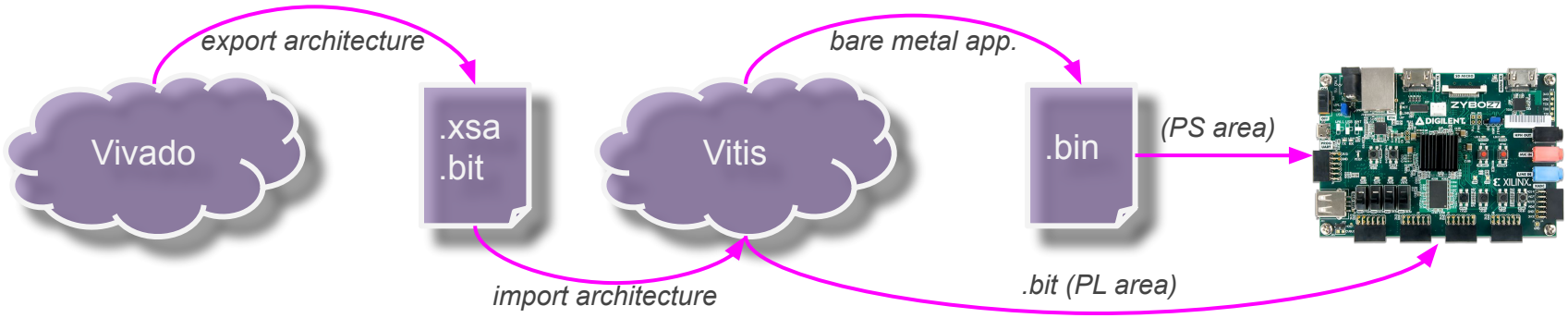
- Implementation

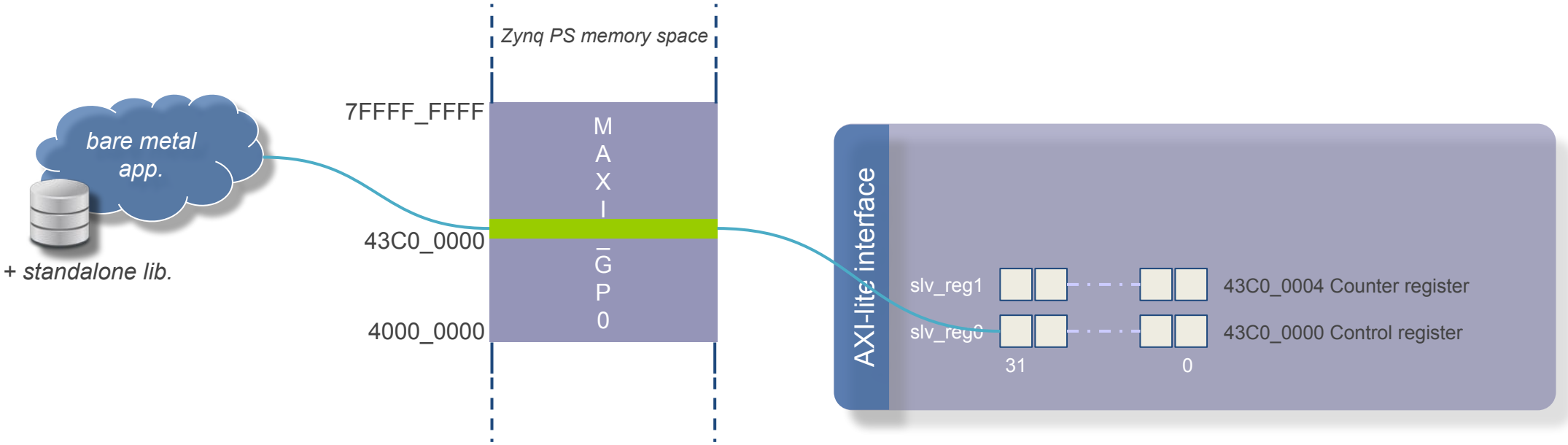
- Bit stream



[Lab] PWM generator app.

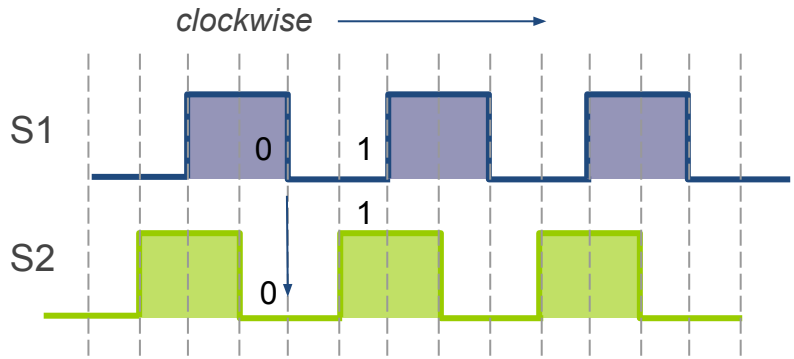
- CPU interactions with your AXI-enabled PWM generator

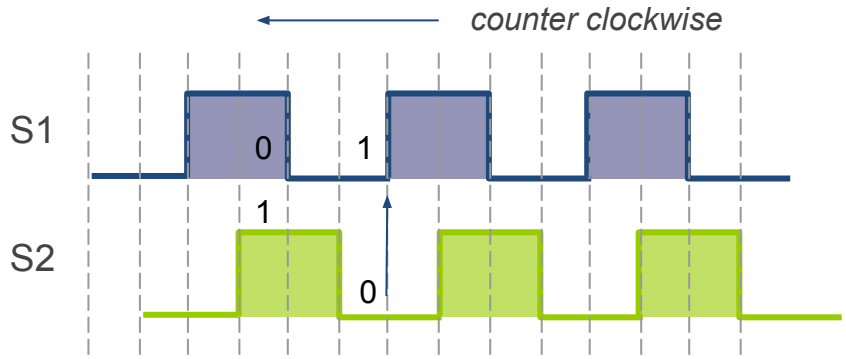




[Lab] Rotary encoder

- IP rotary encoder
  an endless rotary knob

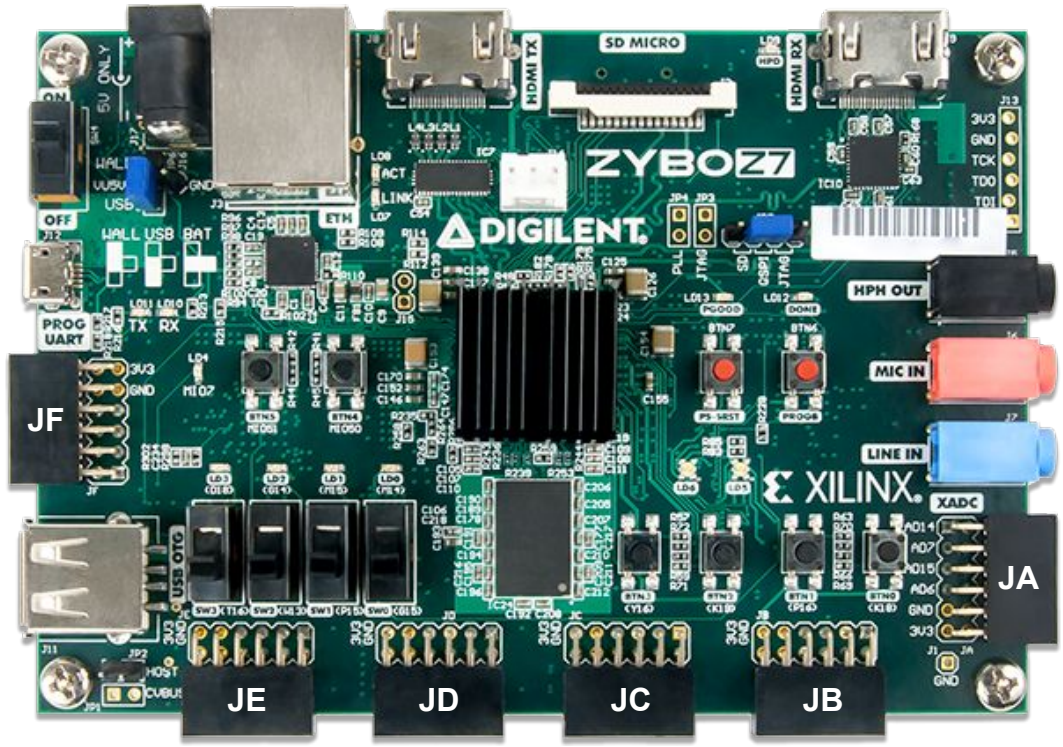




https://lastminuteengineers.com/rotary-encoder-arduino-tutorial/
https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/
https://github.com/Yourigh/Rotary-encoder-VHDL-design

[Lab] Rotary encoder

- Pmod connection
  - MIO signals from PS part
  - JE pins are 200ohm protected
  - High speed Pmod ports only for fast differential signaling



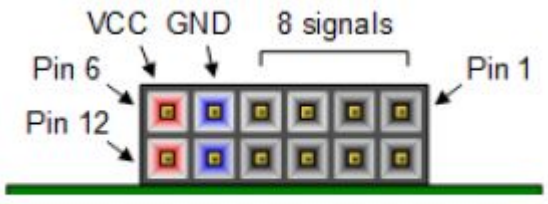


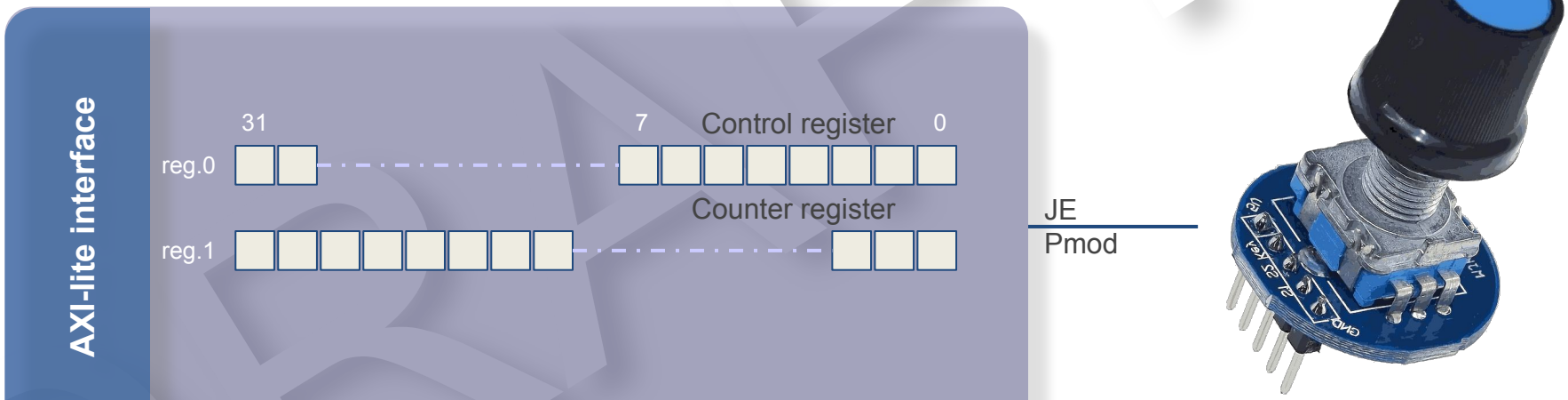
Figure 20. PMOD connectors; front view, as loaded on PCB.

	Pmod JA	Pmod JB*	Pmod JC	Pmod JD	Pmod JE	Pmod JF
Pmod Type	**XADC**	**High-Speed**	**High-Speed**	**High-Speed**	**Standard**	**MIO**
Pin 1	N15	V8	V15	T14	V12	MIO-13
Pin 2	L14	W8	W15	T15	W16	MIO-10
Pin 3	K16	U7	T11	P14	J15	MIO-11
Pin 4	K14	V7	T10	R14	H15	MIO-12
Pin 7	N16	Y7	W14	U14	V13	MIO-0
Pin 8	L15	Y6	Y14	U15	U17	MIO-9
Pin 9	J16	V6	T12	V17	T17	MIO-14
Pin 10	J14	W6	U12	V18	Y17	MIO-15

MIO signals are related to the PS part

[Lab] Soft-IP Rotary encoder

- AXI compliant Rotary Encoder IP block



[Lab] Soft-IP Brushless controller

- AXI compliant Brushless engine controller IP block

[Lab] Zybo booting Linux :D

- Petalinux or ARMbian (better)

[Lab] Digilent Zybo resources

- Digilent getting started:
  https://digilent.com/reference/programmable-logic/guides/start
  [Zybo-Z7] https://digilent.com/reference/programmable-logic/zybo-z7/start
- Vivado & Vitis for Bare Metal Software Projects
  https://digilent.com/reference/programmable-logic/guides/getting-started-with-ipi
- Zybo-Z7-Petalinux
  https://digilent.com/reference/programmable-logic/zybo-z7/demos/petalinux
  https://github.com/Digilent/Zybo-Z7/tree/20/Petalinux/master

Plan

Part IIa – [Advanced] Xilinx Zynq architecture

- Zynq architecture overview,
- AXI bus,
- Hands-on lab: my first soft IP,
- Next lab: RISC-V integration ?

[Lab] RISC-V

- RISC-V @ Zynq

PicoRV32 - a size optimized RISC-V
https://github.com/YosysHQ/picorv32

Plan

Part III – Electronics for ~~dummies~~ all :)

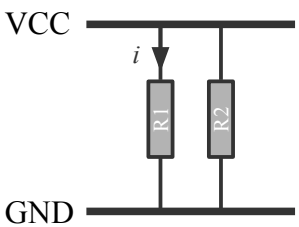
- Basics,
- CMOS transistors,
- Bipolar transistors,
- Sink / Source output modes,
- Operational Amplifier,
- Motors,
- Wavelengths,
- KiCad PCB software.

https://www.electronics-tutorials.ws/

https://doc.lagout.org/electronics/Electronics%20for%20Dummies%20%282005%29.pdf

Ohm's law and friends

- Resistors

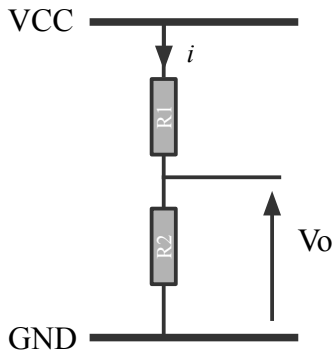


$$i = \frac{VCC}{R1}$$

$$R = \frac{R1 \times R2}{R1 + R2}$$

If R1 == R2
==> Req = R/2

- R-based adaptation levels



$$i = \frac{VCC}{R1 + R2}$$

$$Vo = R2 \times i$$

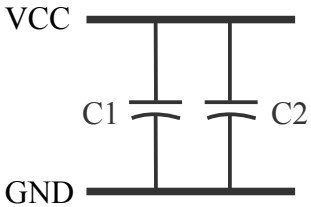
$$Vo = \frac{R2 \times VCC}{R1 + R2}$$

Q

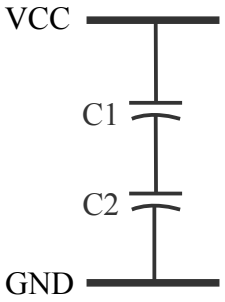
Arduino features VCC = 5v but we need to drive a 3.3v input of a Raspberry Pi

- *Compute R1 and R2 values*
- *If R1 == R2, compute Vo*

- Capacitors

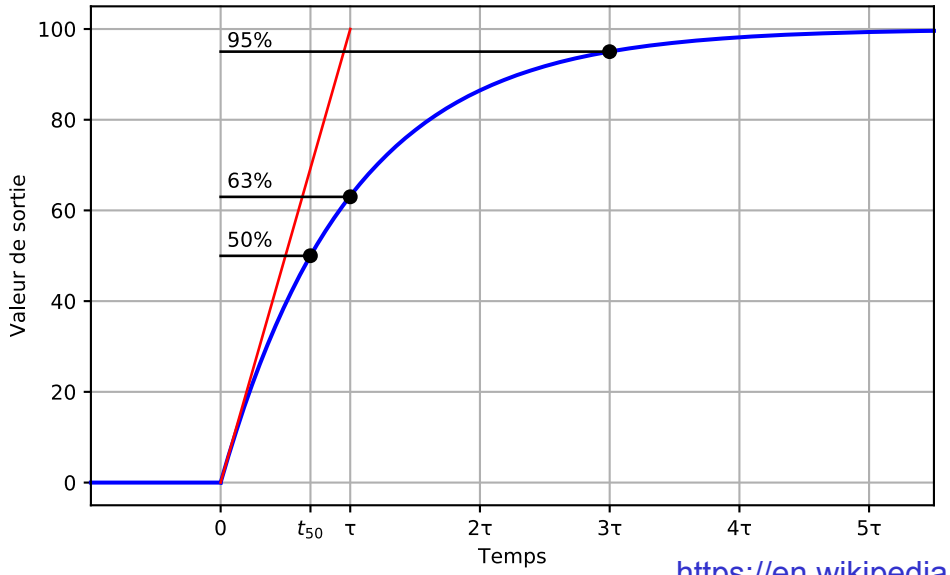


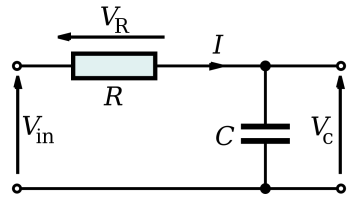
$$C = C1 + C2$$



$$C = \frac{C1 \times C2}{C1 + C2}$$

- RC time constant





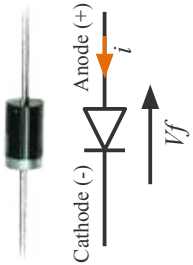
$$\tau = RC$$

https://en.wikipedia.org/wiki/RC_time_constant

Ohm's law and friends

- Diodes … almost unidirectional devices

Common diode



Vf = 0.6v

Schottky

Vf = 0.3v

*Light Emitting Diodes (**LED**)*

Vf red = 1.2v

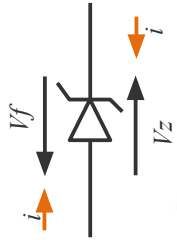
Vf green = 1.8v

Vf blue = 2.5v

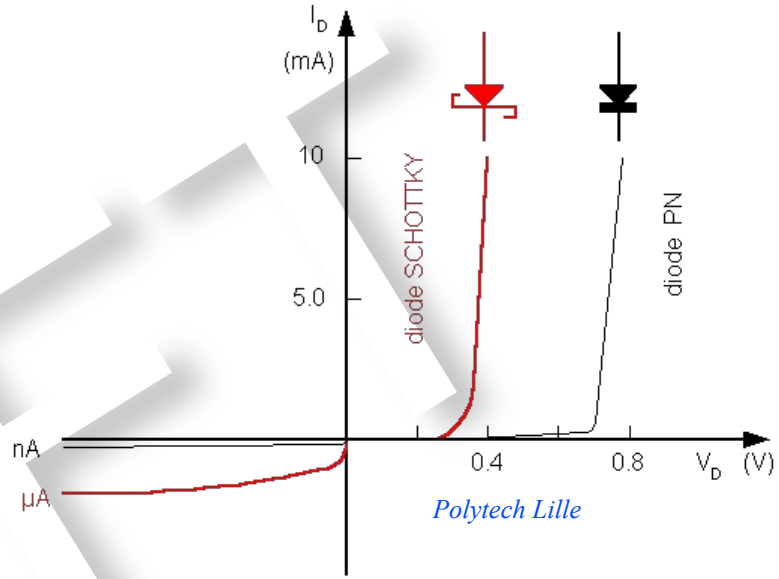
Vf white = 4v

Different Vf according to LED's colour.

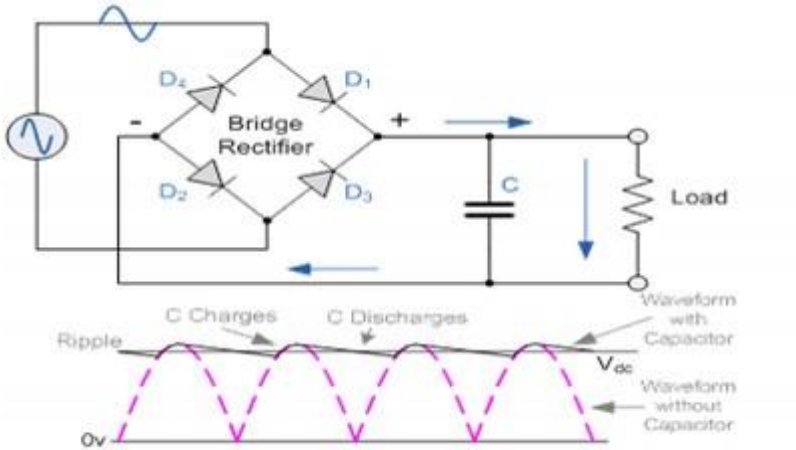
Zener (bi-directional)



Large range of Vz



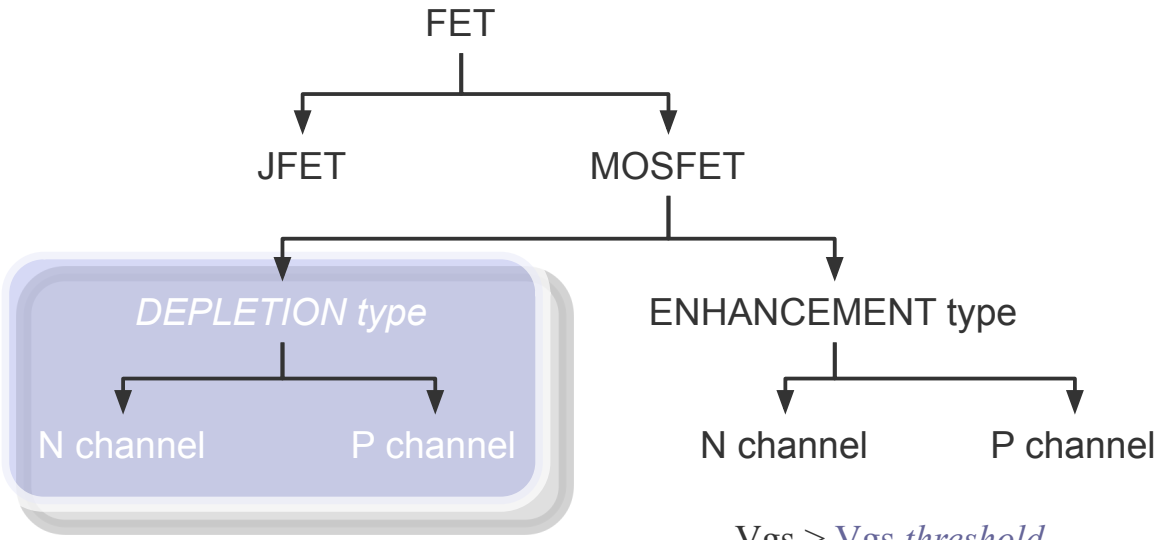
- Bridge rectifier



- Self

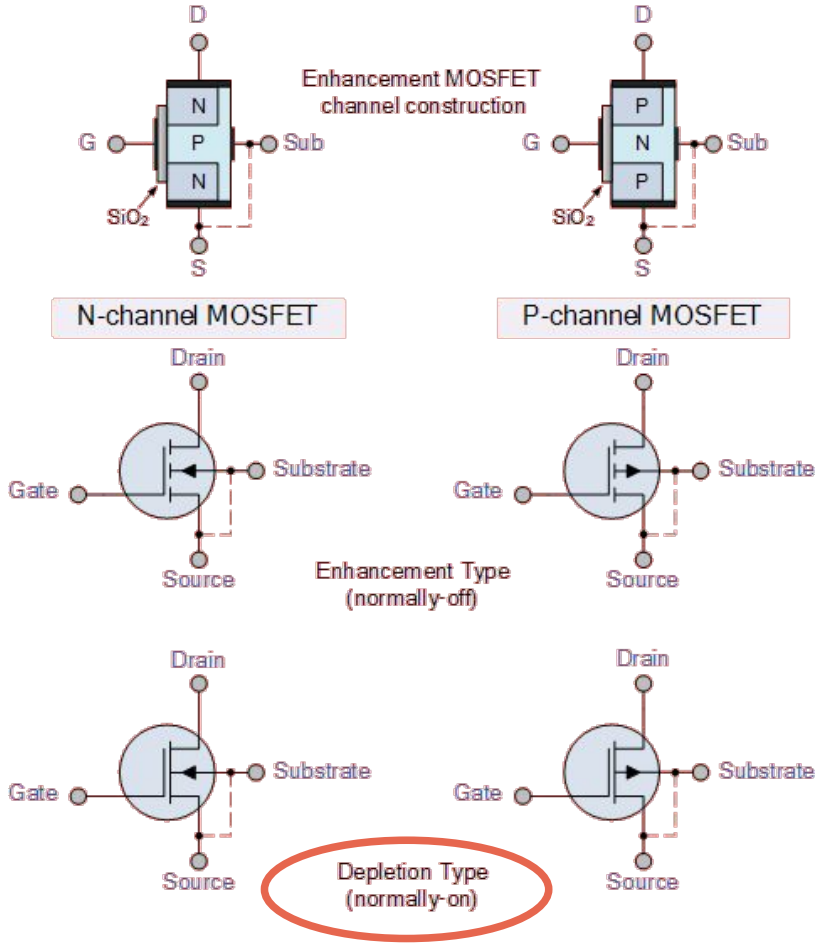
CMOS transistors

- **Voltage** commanded transistors

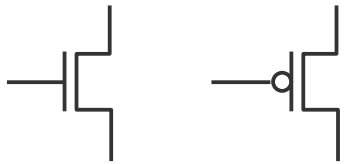


Vgs = 0v → Transistor ON

Vgs > Vgs *threshold* → Transistor ON



- Common symbols

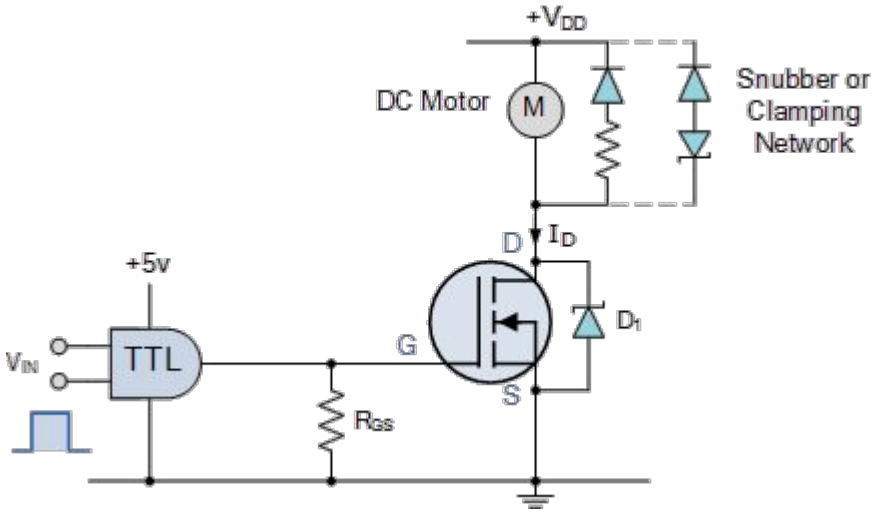


Nmos Pmos

http://www.electronics-tutorials.ws/transistor/tran_1.html

CMOS transistors

- Example: motor driver



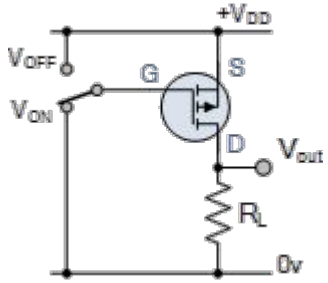
http://www.electronics-tutorials.ws/transistor/tran_7.html

Q

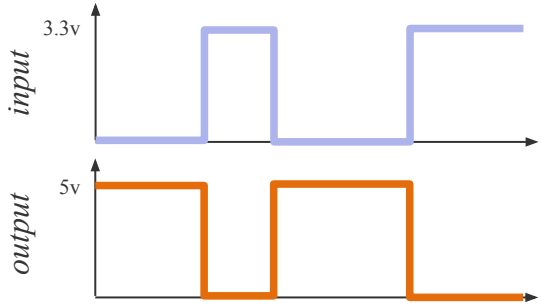
*Considering a NMOS enhanced **2N7002** transistor:*
(see datasheet link below)

- *Which applied Gate voltage will turn ON the actuator ?*
- *What RGS resistor is useful for ?*

- Example: P-MOS driver



- Your first electronic schematic: *Draw a signal inverter*



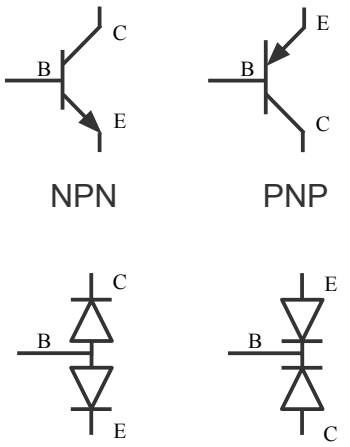
- 2N7002 datasheets

http://m2siame.univ-tlse3.fr/teaching/francois/NMOS-transistor_2N7002.pdf

... and what about a NAND gate ?

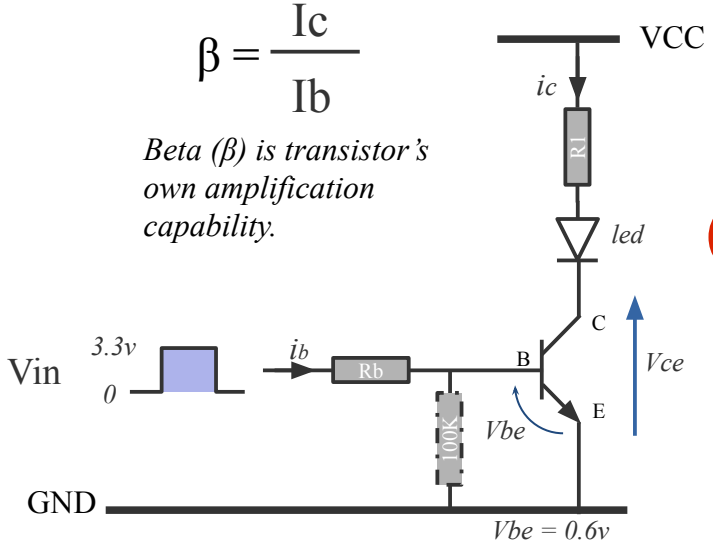
Bipolar transistors

- **Current** commanded transistors



$$\beta = \frac{Ic}{Ib}$$

Beta (β) is transistor's own amplification capability.



Q

Considering a NPN BC547 transistor with Vf led = 1.2v, ic = 20mA

- *Compute R1 and Rb values*

- Your own schematic to drive a led considering:

Vin=0v → led ON
Vin=VCC → led OFF

[addon] What happens if VCC > Vin ?

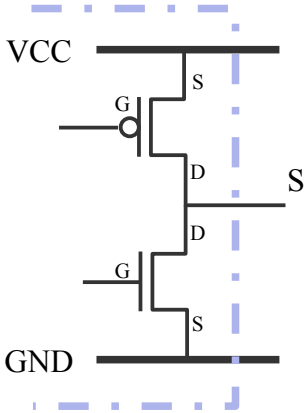
- NPN BC547 datasheet

http://m2siame.univ-tlse3.fr/teaching/francois/NPN-transistor_BC547-BC550.pdf

http://www.electronics-tutorials.ws/transistor/tran_1.html

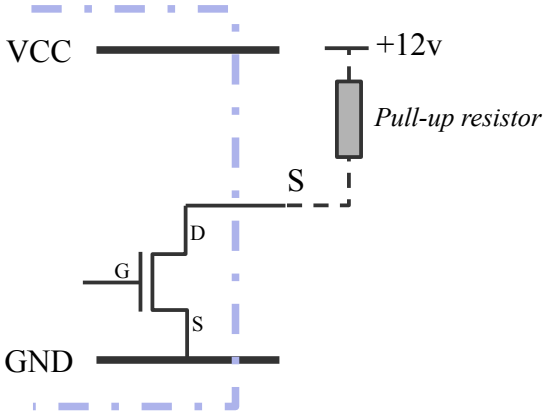
Sink or Source ??

- Totem-pole (*push-pull*) output



Note: same design with bipolar transistors

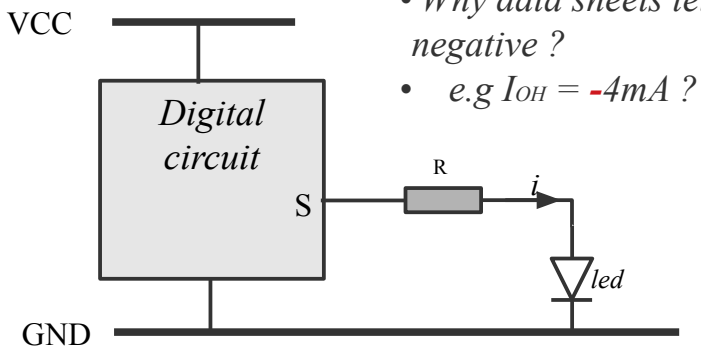
- Open-Drain / Open-Collector output



Note: same design with bipolar transistors

- Source mode:

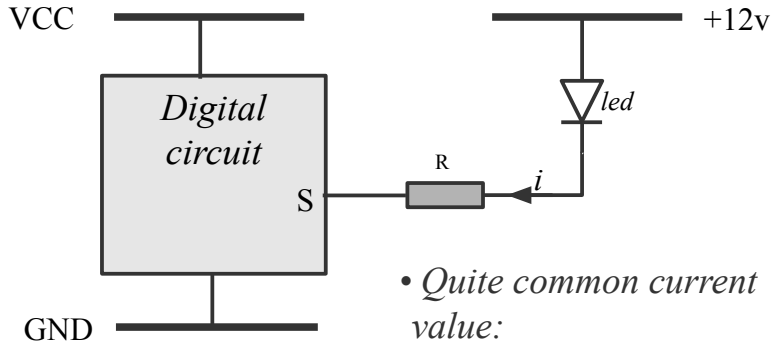
Output provides power to the connected devices



- *Why data sheets tells current is negative ?*
- *e.g I_{OH} = -4mA ?*

- Sink mode:

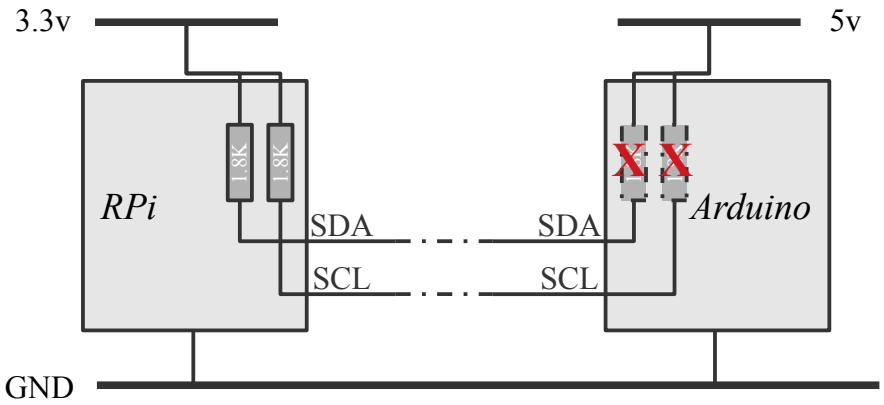
Output absorbs current from connected devices



- *Quite common current value:*
- *e.g I_{OL} = 20mA*

Voltage thresholds

- Application to the I2C bus

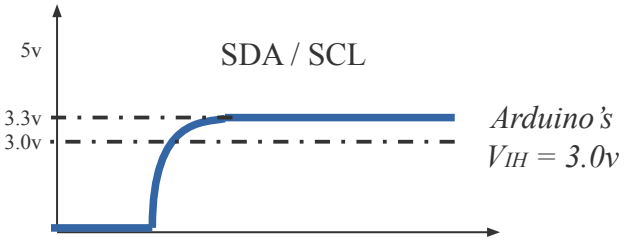


Q

I2C bus master: Raspberry Pi is 3.3v powered
I2C bus slave: Arduino is 5v powered

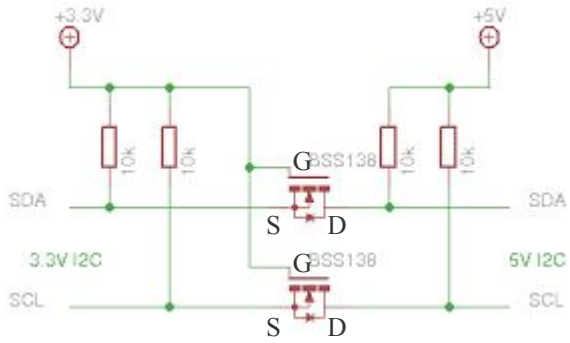
- *Why Arduino and RPi are able to communicate ?*
- *What happens if you don't remove Arduino's pull-up resistors ?*

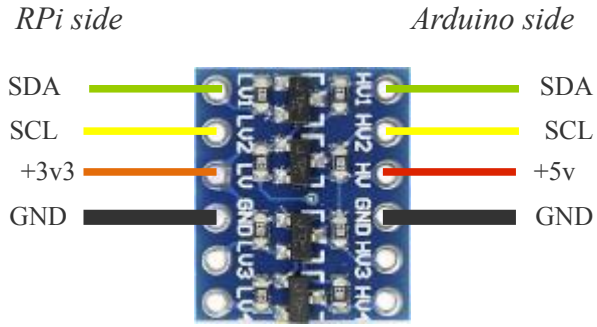
Remember: RPi's GPIO are 3.3v ONLY !!



... hence, only 300mv noise immunity!!

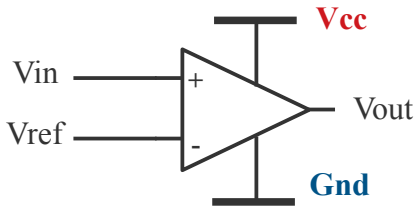
- I2C level shifter





Operational Amplifier

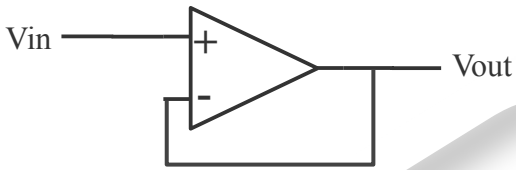
- Comparator



Vout = Ao (V+ - V-)
Note: Ao is open-loop gain

If Vin > Vref then
*Vout = **VCC***
Else
*Vout = **GND***
Endif

- Follower (impedance adaptation level)



Vout = Vin

- Amplifier ...

https://www.electronics-tutorials.ws/category/opamp

Motors

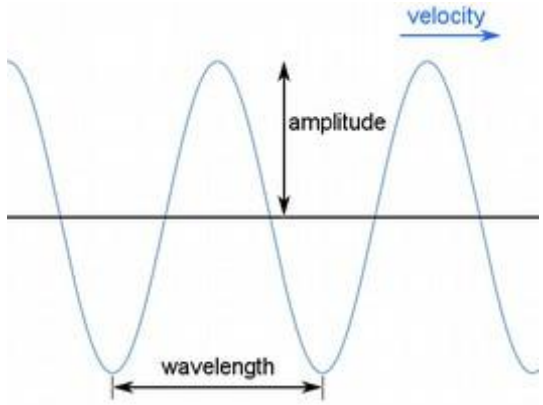
- Servomotor
- Step-motor
- Brushless motor
- H bridge

Wavelengths

- Antenna lengths calculation

$$\lambda = \frac{C}{f}$$

λ (wavelength) represents the distance over which the wave's shape repeat. (Wikipedia)



$$f = \frac{1}{T}$$

frequency (Hz) — *period (second)*

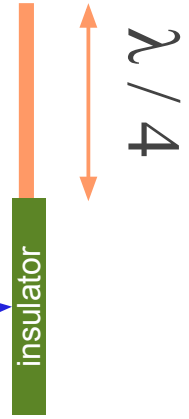
- Speeds

C = 300.000 km/s

Photons in an empty space

λ / 4 wire antenna:

- @868MHz → ~86mm
- @2.4GHz → ~31mm
- @6GHz → ~13mm

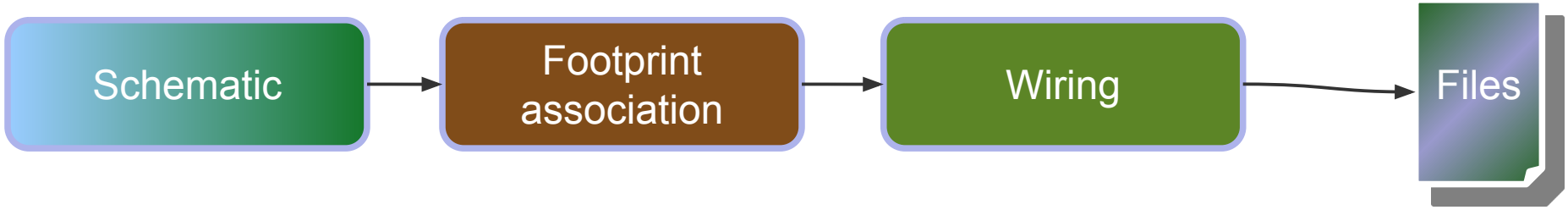


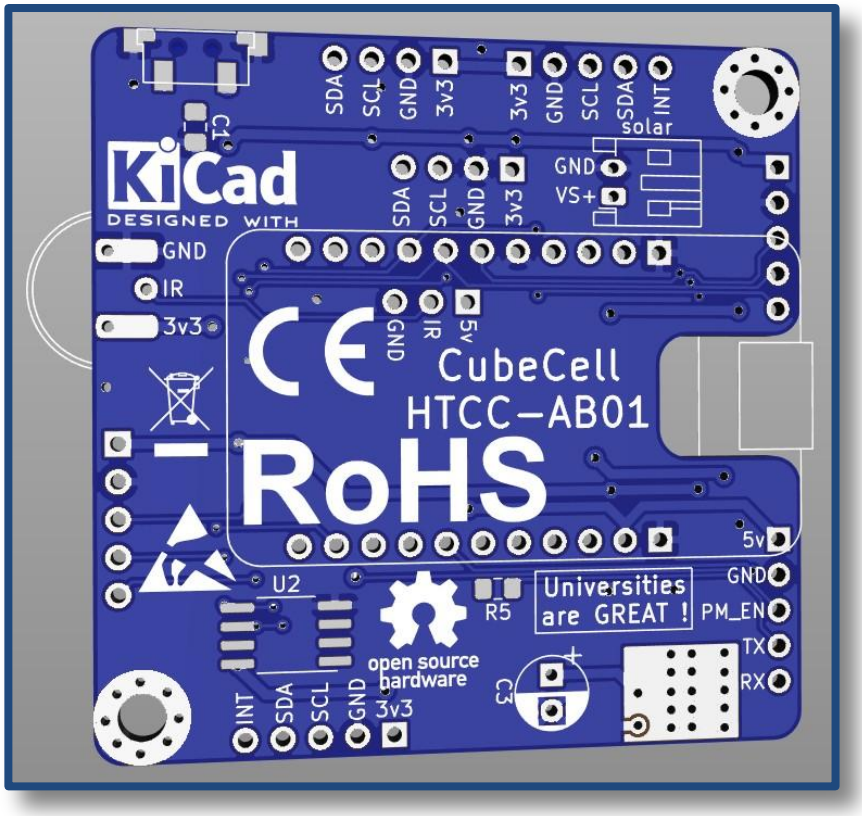
$\mathcal{V}$ = 280.000* km/s — cooper wire — e.g @ 1GHz λ = 28cm

**Electric field propagation in a 12AWG wire. Speed goes from 50 to 99% of C depending on velocity drift. In a 14AWG → 97% of C*

KiCad

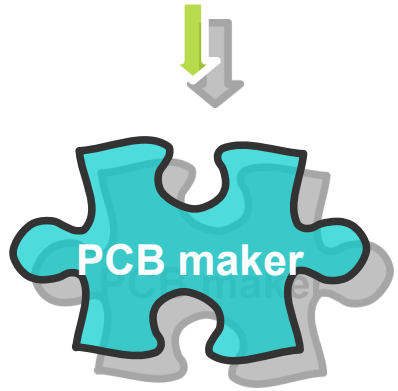
- Creating your own **PCB** … welcome to the master level !





Gerber files
(*pads, wires …*)

Excellon files
(*holes*)



[INSA] Arnaud Bigganzoli KiCad course https://docs.google.com/presentation/d/1c8y2dSM7ou4Fs_ESQGD6nbiX78ckDWkQml1QuDO1WBo

END

strong empathy for hardware

perseverance

some soldering sessions ;)

a bit of electronics

ghdl + gtkwave

secret ingredient*

linux

path to YOUR success in
the FPGA world !

**Po's father knows it!*