

## EN103 : électronique numérique

### Logique combinatoire et séquentielle

Organisation

Énoncés TD et TP CAO

Énoncé du projet

Présentation du langage VHDL

Equipe pédagogique :

BORNAT	Yannick
DALLET	Dominique
DELOMIER	Yann
JEGO	Christophe
LALEVEE	André
LEBRET	Valéry
MORIZET	Guy

Dépt. Electronique



# TABLE DES MATIERES

<b>ORGANISATION .....</b>	<b>5</b>
SEANCES DE COURS .....	5
SEANCES DE TD ET TP CAO .....	6
SEANCES DE PROJET .....	7
<b>ÉNONCES TD ET TP CAO.....</b>	<b>9</b>
SEANCE 1 : SYNTHESE DE FONCTIONS COMBINATOIRES .....	9
SEANCE 2 : INITIATION AU LANGAGE DE CONCEPTION MATERIEL VHDL.....	15
SEANCE 3 : LES BASCULES .....	18
SEANCE 4 : FONCTIONS SEQUENTIELLES COMPLEXES .....	23
SEANCE 5 : LE COMPTEUR A MODULO VARIABLE.....	25
SEANCE 6 : SYNTHESE D'UNE MACHINE A ETATS FINIS .....	27
<b>ÉNONCE DU PROJET .....</b>	<b>30</b>
PRINCIPE DE FONCTIONNEMENT DU PROJET MONTE CHARGE .....	31
CARTE FPGA BASYS3 .....	32
CAHIER DES CHARGES .....	34
<b>PRESENTATION DU LANGAGE VHDL.....</b>	<b>39</b>
ORGANISATION FONCTIONNELLE DE DEVELOPPEMENT D'UN CIRCUIT PROGRAMMABLE .....	41
STRUCTURE D'UNE DESCRIPTION VHDL .....	42
LE MODE COMBINATOIRE .....	46
LE MODE SEQUENTIEL.....	51
LA DESCRIPTION STRUCTURELLE EN VDHL .....	57
BIBLIOGRAPHIE.....	60
<b>DATA SHEET POUR LA SEANCE 5 DE TD .....</b>	<b>61</b>
DATA SHEET DU CIRCUIT 74 HC163 .....	62
DATA SHEET DU CIRCUIT 74 HC02 .....	73
<b>TUTORIAL VIVADO POUR LE TP N°1 .....</b>	<b>75</b>



<b>ORGANISATION EN102 &amp; EN103</b> <b>Année 2016-2017</b>
---

Déroulement de l'enseignement en 3 phases :

- |                                |                |
|--------------------------------|----------------|
| ✓ 5 cours de 1h20              | volume : 6h40  |
| ✓ 6 séances TD et TP CAO de 4h | volume : 24h00 |
| ✓ 4 séances projet de 4h       | volume : 16h00 |

**I) 5 séances de cours en promotion entière**

<b>Cours N°1/5</b> (1h20)	08 septembre (semaine 36) : <b>Cours</b> : représentation des nombres & algèbre de Boole
------------------------------	---

---

<b>Cours N°2/5</b> (1h20)	14 septembre (semaine 37) : <b>Cours</b> : fonctions combinatoires
------------------------------	---

---

<b>Cours N°3/5</b> (1h20)	19 septembre (semaine 38) : <b>Cours</b> : éléments de base en logique séquentiel
------------------------------	--

---

<b>Cours N°4/5</b> (1h20)	22 septembre (semaine 38) : <b>Cours</b> : compteurs
------------------------------	---

---

<b>Cours N°5/5</b> (1h20)	25 septembre (semaine 39) : <b>Cours</b> : fonctions séquentielles complexes
------------------------------	---

---

## II) 6 séances TD/TP en 1/8 de promotion

---

**SEANCE N°1** semaine 40

(2H00) TD

(2H00) TP

Thème : synthèse de fonctions combinatoires  
initiation à l'environnement VIVADO

---

**SEANCE N°2** semaine 41

(4H00) TP

Thème : initiation au langage de conception matériel VHDL

---

**SEANCE N°3** semaine 42

(2H00) TD

(2H00) TP

Thème : les bascules

---

**SEANCE N°4** semaine 43

(2H00) TD

(2H00) TP

Thème : fonctions séquentielles complexes

---

**SEANCE N°5** semaine 45

(2H00) TD

(2H00) TP

Thème : le compteur à modulo variable

---

**SEANCE N°6** semaines 46

(2H00) TD

(2H00) TP

Thème : contrôleur d'un bus

---

### III) 4 séances de projet en 1/8 de promotion

Sujet du projet : **Monte Charge**

---

**SEANCE N°1** semaines 47 :

Présentation du projet **Monte Charge**

Découpe architecturale de la fonctionnalité en blocs.

Découpage

---

**SEANCE N°2** semaines 48 :

Description des différents blocs du projet en VHDL

Simulation de chaque bloc avec ISIM

---

**SEANCE N° 3** semaines 49 :

Description des différents blocs du projet en VHDL

Simulation de chaque bloc avec ISIM

---

**SEANCE N°4** semaines 50 :

Assemblage final des différents blocs constituant l'architecture globale

Placement et routage de l'architecture (observation du schéma obtenu et des ressources utilisées)

Programmation du circuit et validation par prototypage de la fonctionnalité.

---



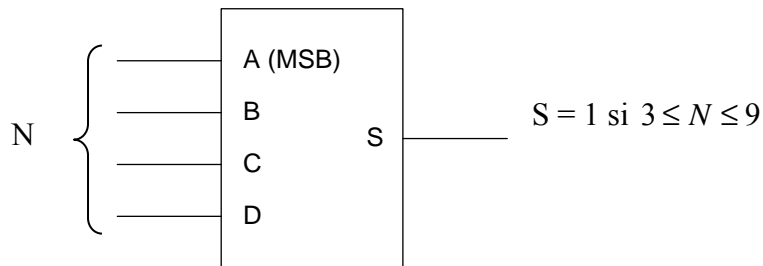


# Thème de la séance 1 : synthèse de fonctions combinatoires

## – Partie TD –

### I) Synthèse d'un circuit combinatoire

On désire réaliser un système logique tel que la sortie S passe à 1 lorsque le nombre N de 4 bits présent en entrée est compris entre 3 et 9.



- 1) Donner la (ou les) 1<sup>ère</sup> et 2<sup>ème</sup> forme normale de S.
- 2) Reprendre la même question si l'on suppose maintenant que N est codé en BCD.
- 3) Etablir le schéma technologique le plus simple avec des portes NOR et/ou NAND.

### II) Synthèse avec des opérateurs de multiplexage

**Question 2.1** : Caractérisation d'un multiplexeur 2 vers 1 (MUX2:1)

- a-) : Combien de bits faut-il pour assurer l'adressage d'un MUX2:1 ?
- b-) : Donner la table de vérité d'un MUX2:1
- c-) : Donner l'expression logique de la sortie d'un MUX2:1 à partir de la table de vérité et en appliquant une simplification à l'aide d'un tableau de Karnaugh.
- d-) : En déduire un logigramme reposant sur les éléments logiques suivants :

Inverseur, OU logique, ET logique

e-) : Nous supposons que les temps de propagation des différents éléments logiques sont :

- ✓ Inverseur : 0.5 ns
- ✓ OU logique : 2 ns
- ✓ ET logique : 2 ns

Quel est le temps de propagation de ce multiplexeur 2 vers 1 ?

**Question 2.2 :** Synthèse d'une fonction logique à l'aide d'une structure de multiplexage

a-) : Faire la synthèse à l'aide d'un multiplexeur 4 vers 1 de la fonction logique S correspondant à la table de vérité suivante :

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Les 0 et 1 logiques seront respectivement modélisés par la masse (GND) et la tension d'alimentation du circuit (VDD) comme expliqué en cours (Cours 2, Transparent n° 23).

b-) : Faire la synthèse à l'aide d'un multiplexeur 2 vers 1 de la même fonction logique S.

### **III) Synthèse d'un comparateur (optionnel en fonction du temps)**

On souhaite réaliser un comparateur de deux mots de  $n$  bits (figure 1)  $A = (A_n \cdots A_i \cdots A_1)$  et  $B = (B_n \cdots B_i \cdots B_1)$

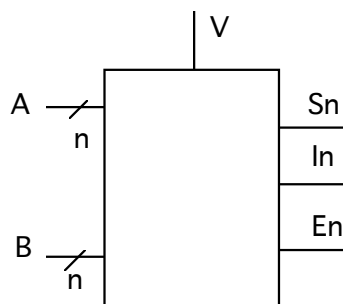


Figure 1

### Fonctionnement du comparateur :

- Si l'entrée de validation ( $V$ ) vaut 0, alors les trois sorties (Egal :  $E_n$ ), (Supérieur :  $S_n$ ) et (Inférieur :  $I_n$ ) sont égales à 0.
- Si l'entrée de validation ( $V$ ) vaut 1, alors :
  - $S_n = 1$  ssi  $A > B$
  - $I_n = 1$  ssi  $A < B$
  - $E_n = 1$  ssi  $A = B$

#### ***Question 3.1***

Dans un premier temps, on souhaite réaliser un comparateur élémentaire de deux mots de 1 bit ( $n = 1$ ). Etablir les équations des sorties  $S_1$ ,  $I_1$ , et  $E_1$  en fonction des entrées  $A_1$ ,  $B_1$ , et  $V$ .

#### ***Question 3.2***

Dessiner le schéma de ce comparateur à partir d'opérateurs élémentaires (INV, NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

#### ***Question 3.3***

On souhaite maintenant étendre l'amplitude du comparateur à deux mots de 2 bits. Après avoir établi les équations de  $S_2$ ,  $I_2$ , et  $E_2$  en fonction des bits  $A_2$ ,  $A_1$ ,  $B_2$ ,  $B_1$ , et de l'entrée de validation  $V$ , concevoir le schéma de ce comparateur en associant des comparateurs élémentaires et un minimum d'opérateurs (NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

#### ***Question 3.4***

A partir des équations trouvées précédemment, établir les relations de récurrence ci-dessous :

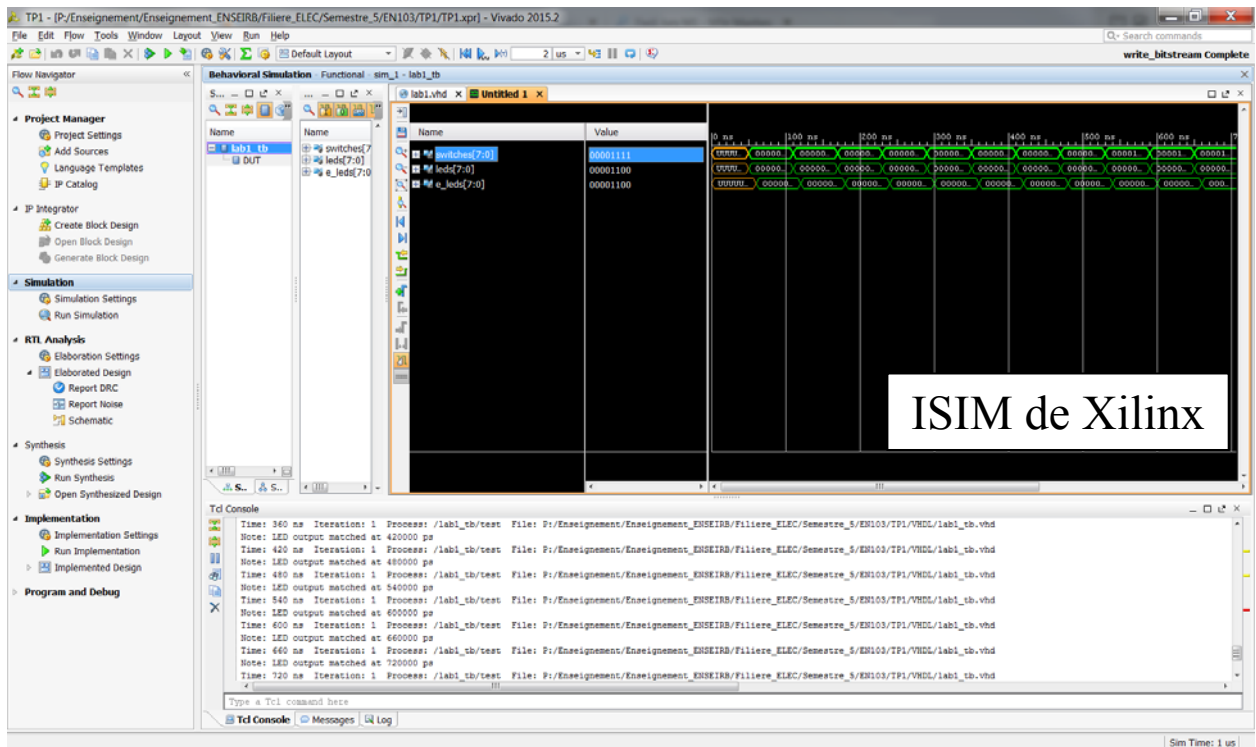
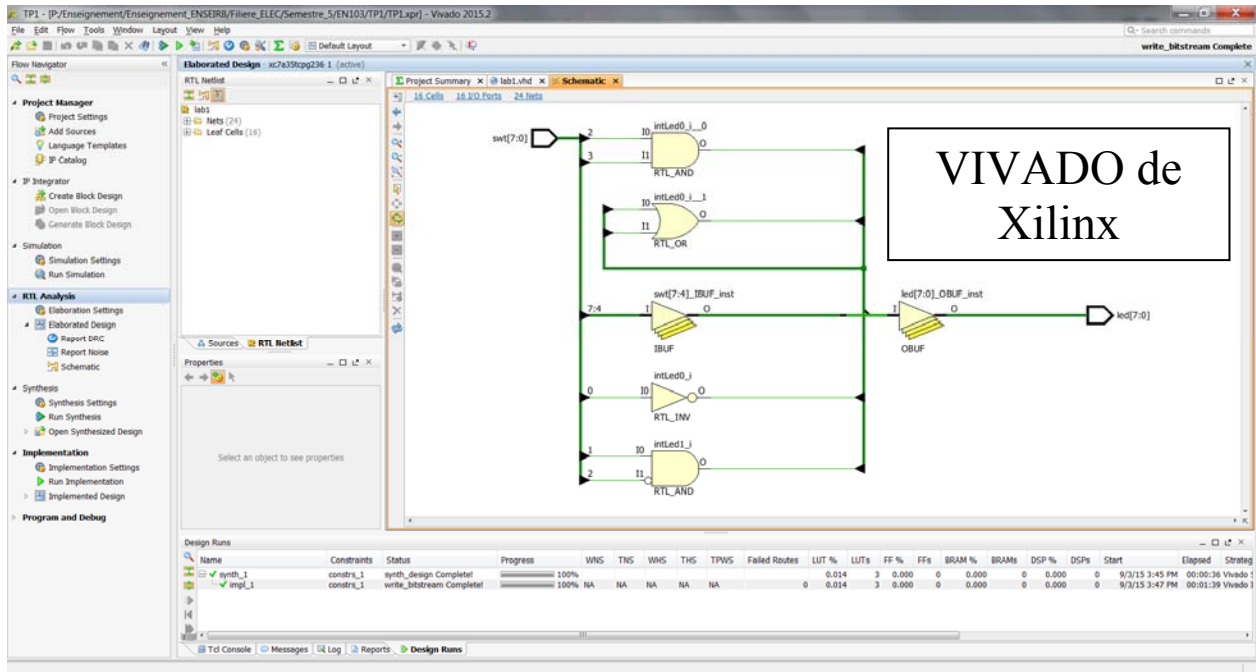
$$S_n = f(S_{n-1}, A_n, B_n, V)$$

$$I_n = g(I_{n-1}, A_n, B_n, V)$$

$$E_n = h(E_{n-1}, A_n, B_n, V)$$

## – Partie TP CAO –

Initiation à l'environnement VIVADO qui sera utilisé au cours du module EN103 s'appuyant sur un tutorial de chez Xilinx



## 1-) Initiation à l'environnement VIVADO

Dans le cadre de ce TP, une initiation à l'environnement VIVADO est proposée. Pour ce faire, prière de vous reporter au tutorial fourni à la page 61 de ce document

## 2-) Simulation à l'aide de fichiers de type Tcl

Une simulation dans l'environnement VIVADO de Xilinx d'une architecture décrite à l'aide du langage VHDL implique la définition d'un fichier de test (*testbench* en anglais) de type VHDL ou d'un fichier force de type Tcl (Tool Command Language).

Au cours du tutorial, une simulation à partir d'un fichier de test (*testbench* en anglais) de type VHDL est proposée. Afin de simplifier cette étape, nous proposons dans le cadre du module EN103 de plutôt utiliser des fichiers force de type Tcl (Tool Command Language).

Un exemple de fichier test (*testbench.tcl*) est décrit ci-dessous :

```
#Initialisation de l'affichage des courbes
restart
remove_forces -all

# Stimulation du signal swt
add_force swt -radix dec {0 0ns} {1 10ns} {2 20ns} {3 30ns} {4 40ns} {5 50ns} {6 60ns}
{7 70ns} {8 80ns} {16 90ns} {32 100ns} {64 110ns} {128 120ns} {255 130ns} -repeat 200ns

# Lancement de la simulation pour t=400 ns
run 400 ns
```

Ce fichier permet de simuler l'architecture décrite dans le tutorial précédent.

Simulation de l'architecture *lab1* à l'aide d'un fichier de type Tcl:

1-) Décrire le fichier *testbench.tcl* et le positionner sous la racine du projet (*par exemple : c:\Digilent/Basys3Workshop/2014\_4\_artix7\_sources/lab1*)

1-) Dans l'environnement VIVADO, se positionner sur le fichier décrivant l'architecture (*par exemple : lab1.vhd*).

2-) Lancer la simulation : *Run Simulation*.

3-) Dans la fenêtre *TCL Console*, il faut se placer sous la racine du projet :  
*cd c:\Digilent/Basys3Workshop/2014\_4\_artix7\_sources/lab1*

4-) Enfin, l'exécution de ce fichier s'effectue en écrivant dans la fenêtre *TCL Console* la commande suivante : *source testbench.tcl*

## Complément sur les commandes Tcl

1-) Construction d'un signal d'horloge :

Caractéristiques du signal : période 20 ns et rapport cyclique 0,5

```
add_force clk 0 {0 0ns} {1 10ns} -repeat 20 ns
```

2-) Construction d'un bus contenant des nombres entiers naturels :

```
add_force Valeur -radix dec {0 0} {1 80ns} {2 160ns} {3 240ns} {4 320ns} {5 400ns} {6  
480ns} {7 560ns} -repeat 640ns
```

# Thème de la séance 2 : synthèse de fonctions combinatoires

## – *Partie TD* –

*Pas de TD durant cette séance mais une introduction progressive au langage VHDL et aux outils associés*

## – *Partie TP CAO* –

### 1-) **Initiation à l'organisation du langage : (entité et architecture)**

L'analyse d'un modèle VHDL peut s'effectuer sur des parties du code ou "unités de compilation". Il existe 5 types d'unités de compilation :

- L'entité
- L'architecture
- Le paquetage
- Le corps du paquetage
- La configuration

L'**entité** et l'**architecture** sont deux unités **obligatoires** pour décrire un modèle. Les autres unités sont optionnelles

1.1 Familiarisation à la description d'une entité et d'une architecture sur un cas d'école : la porte NAND.

Simuler la description VHDL de la porte NAND sous ISIM après l'écriture d'un fichier de test.

Visualiser le schéma RTL de la description dans l'environnement VIVADO.

1.2 Décrire un modèle VHDL donnant 7 fonctions logiques (NON, ET, OU, NON ET, NON OU, OU Exclusif, ET inclusif) de deux entrées A et B.

Simuler la description VHDL sous ISIM après l'écriture d'un fichier de test.

Visualiser le schéma RTL de la description dans l'environnement VIVADO.

### 2-) **Initiation à la description structurelle en VHDL : (composant et instanciation)**

L'objectif de cet exercice est de voir précisément comment coder une représentation structurelle d'un circuit, autrement dit :

- comment déclarer des blocs (qu'on appellera composant)
- comment utiliser un composant (qui devient une "instance") et déclarer la façon dont il est connecté.

## 2.1 Le comparateur 1bit

Rappel : le principe de fonctionnement est donné dans l'exercice N°3 de la séance 1. Il a également été présenté en cours.

les équations des sorties  $S_i$ ,  $I_i$ , et  $E_i$  en fonction des entrées  $A_i$ ,  $B_i$ , et  $V$  sont :

$$S_i = V.A_i.\overline{B_i}$$

$$I_i = V.\overline{A_i}.B_i$$

$$E_i = V.(A_i.B_i + \overline{A_i}.\overline{B_i}) = V.(A_i \oplus B_i)$$

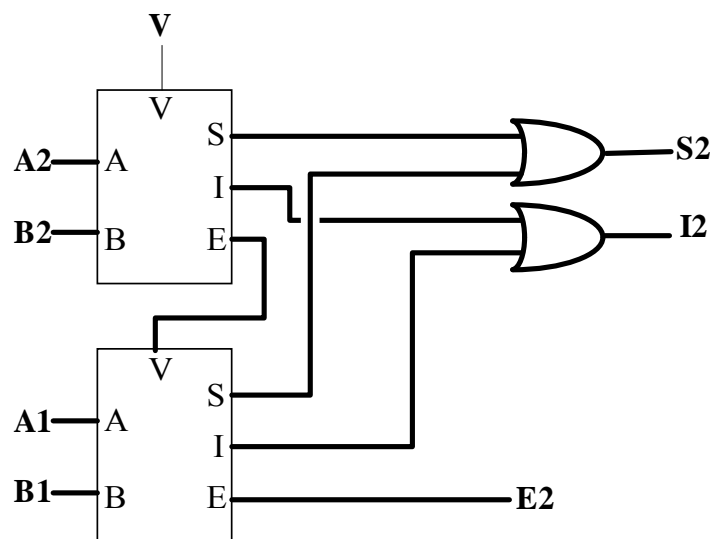
Décrire l'entité et l'architecture pour un comparateur 1 bit

Simuler la description VHDL du comparateur 1 bit sous ISIM après l'écriture d'un fichier de test.

Visualiser le schéma RTL de la description dans l'environnement VIVADO.

## 2.2 Le comparateur 2 bit

Comme démontré au cours de la séance 1, il est possible de concevoir un comparateur 2 bits à partir de deux comparateurs 1 bit (cf schéma suivant)



Réaliser une description structurée en VHDL d'un comparateur 2 bits. Pour ce faire, il faudra déclarer un composant comparateur 1 bit au sein de l'architecture et en faire deux instanciations. Simuler la description VHDL du comparateur 2 bits sous ISIM après l'écriture d'un fichier de test.

Visualiser le schéma RTL de la description dans l'environnement VIVADO.



### 3-) Initiation à la description structurelle en VHDL : (composant et instanciation)

#### 3.1 L'additionneur 1bit

Rappel : le principe de fonctionnement de l'additionneur binaire est donné en cours

les équations des sorties S et C en fonction des entrées A, B et C<sup>-1</sup> sont :

$$S = A \text{ XOR } B \text{ XOR } C^{-1}$$

$$C = (A \text{ ET } B) \text{ OU } (A \text{ ET } C^{-1}) \text{ OU } (B \text{ ET } C^{-1})$$

Décrire l'entité et l'architecture pour un additionneur 1 bit

Simuler la description VHDL de l'additionneur 1 bit sous ISIM après l'écriture d'un fichier de test.

Visualiser le schéma RTL de la description dans l'environnement VIVADO.

#### 3.2 L'additionneur 3 bits

Réaliser une description structurelle en VHDL d'un additionneur 3 bits. Pour ce faire, il faudra déclarer un composant additionneur 1 bit au sein de l'architecture et en faire trois instanciations comme expliquer en cours.

Simuler la description VHDL de l'additionneur 3 bits sous ISIM après l'écriture d'un fichier de test.

Visualiser le schéma RTL de la description dans l'environnement VIVADO.

# Thème de la séance 3 : les bascules

## - Partie TD -

### 1) Bascule RS

Soit la bascule RS de la Figure 1, nous considérons que le temps de propagation  $t_p$  dans une porte NOR est égal à 10 ns et en négligeant le temps de montée  $t_r$  et le temps de descente  $t_f$ .

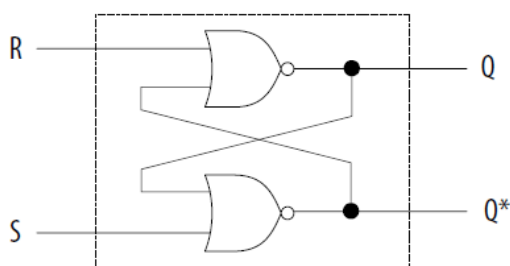


Figure 1 : Bascule RS

1.1 Compléter la table de vérité de la bascule RS

R	S	Q	Q*
0	0		
0	1		
1	0		
1	1		

1.2 Compléter le chronogramme de la Figure 2 (pour le tracé des chronogrammes, on considère que **Q est initialement à 0** )

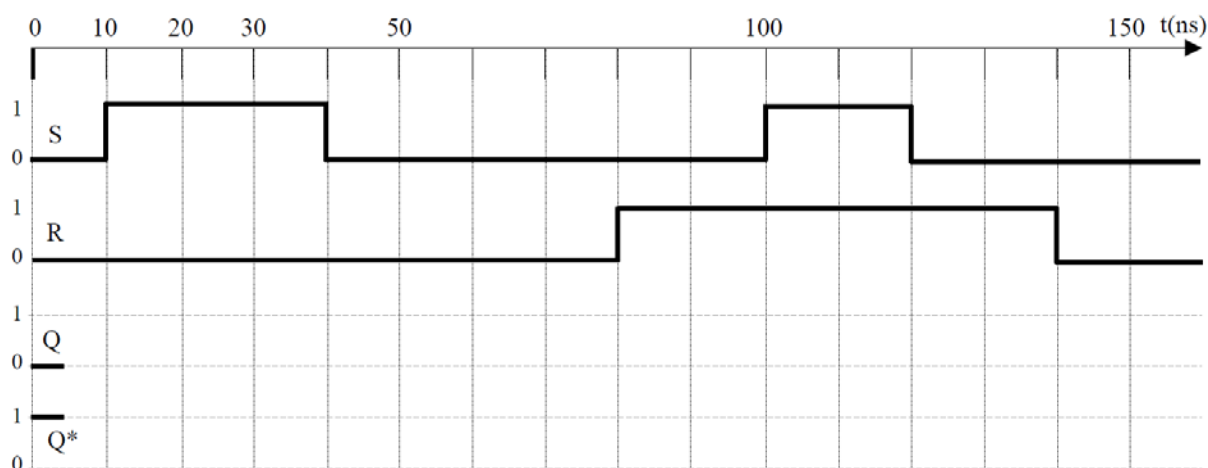


Figure 2 : chronogrammes à compléter

1.3 Compléter les chronogrammes de la Figure 3 (pour le tracé des chronogrammes, nous considérons que **Q est initialement à 0**). Un commentaire ?

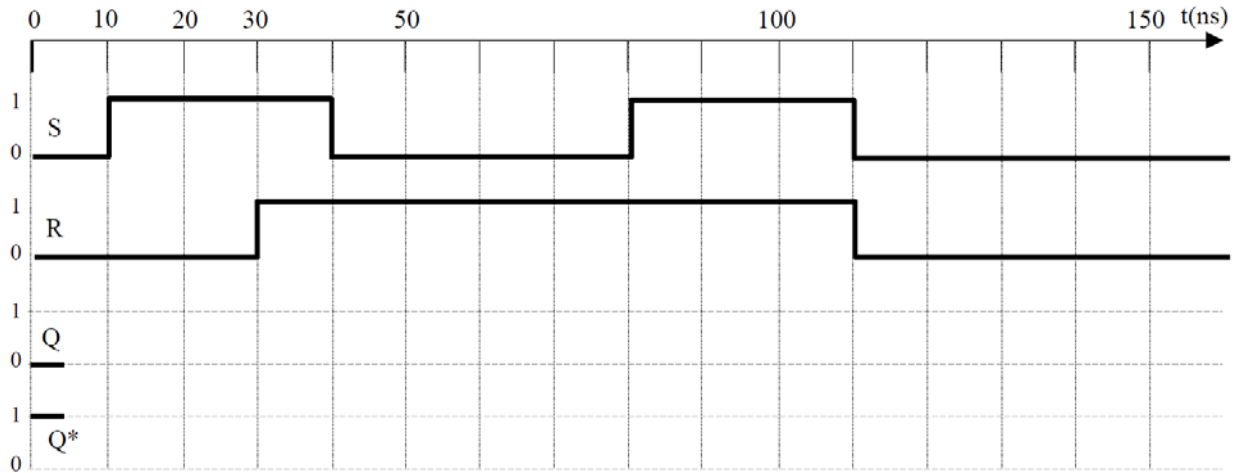


Figure 3 : chronogrammes à compléter

## 2) Bascule D « latch »

La structure de la bascule **D "latch"** CMOS est donnée en Figure 4. Cette bascule contient deux interrupteurs (T1, T2) et deux inverseurs (I1, I2). Le signal de commande  $E^*$  est obtenu par inversion de  $E$  (inverseur I3).

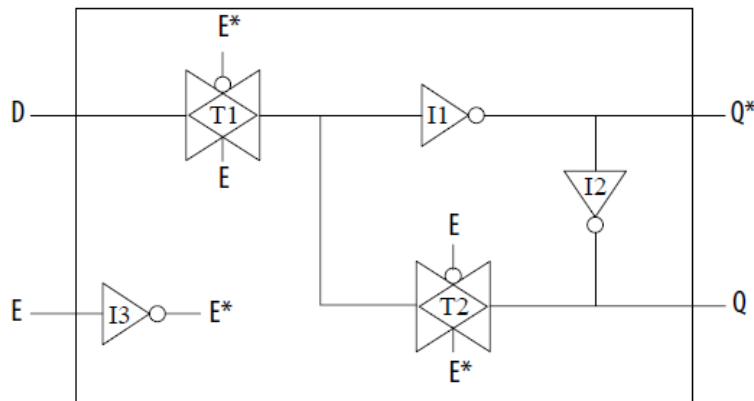


Figure 4 : bascule D latch

2.1 En considérant des temps de commutation nuls ( $t_p$  temps de propagation,  $t_r$  temps de montée et  $t_f$  temps de descente), compléter les chronogrammes de la Figure 5. Établir la table de transition de la bascule.

2.2 En admettant que chaque porte (I ou T) présente un temps de propagation  $t_p$  égal à  $\tau = 0,2$  ns ( $t_r = t_f = 0$  pour simplifier), retracer les chronogrammes de la Figure 5.

En déduire les valeurs suivantes :

- *temps de prépositionnement*  $t_{su}$  ( $t_{setup}$ ) de  $D$  : durée minimale de la valeur de  $D$  à verrouiller, avant transition de  $E$ ,
- *temps de maintien*  $t_h$  ( $t_{hold}$ ) de  $D$  : durée minimale de la valeur de  $D$  à verrouiller, après transition de  $E$ .

Peut-on facilement calculer les durées minimales d'impulsion à 0 et 1 de l'entrée E,  $t_{w0}(E)$  et  $t_{w1}(E)$  ?

2.3 Élaborer une bascule "latch" avec remise à zéro prioritaire (reset).

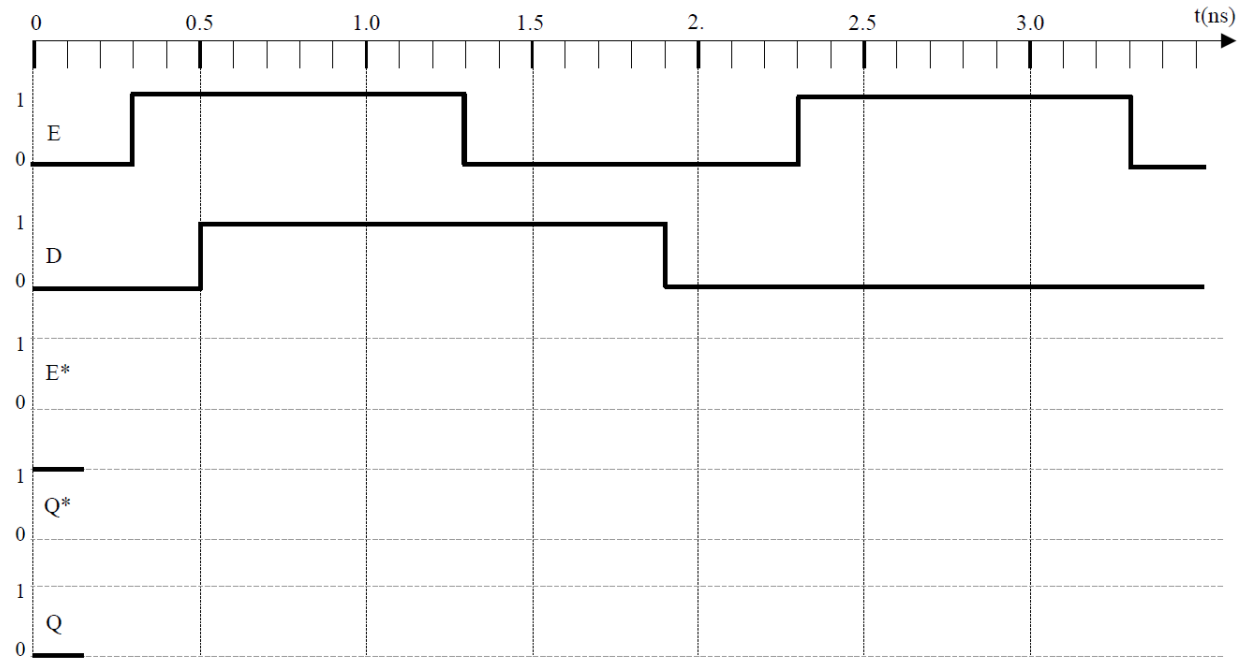


Figure 5 : chronogrammes à compléter

### 3) Bascule D « flip-flop »

La Figure 6 représente une structure maître-esclave de la bascule D "flip-flop" constituée par la mise en série de deux bascules D "latch".

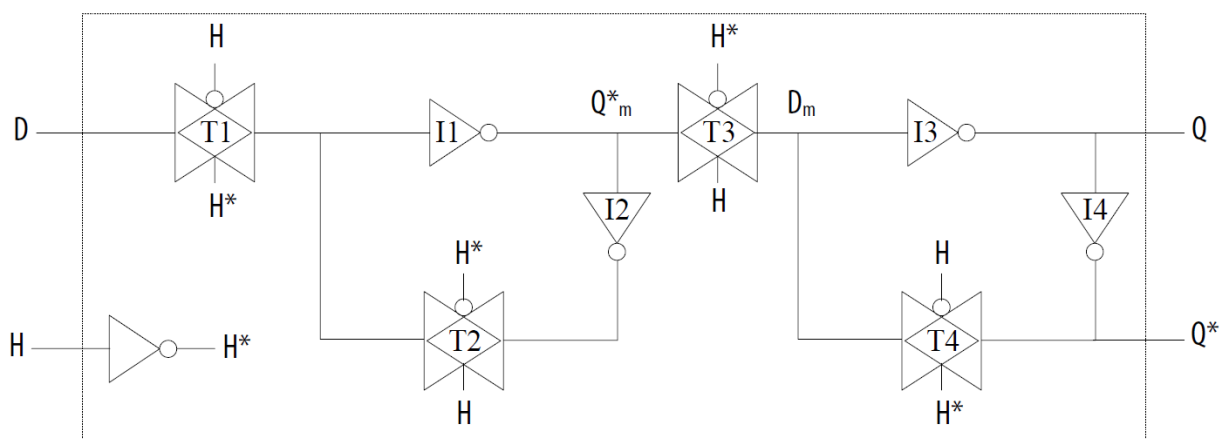


Figure 6. Bascule D "flip-flop" maître-esclave

3.1 Compléter les chronogrammes de la figure 7 ( $t_p = t_r = t_f = 0$ ) et établir la table de transition de la bascule.

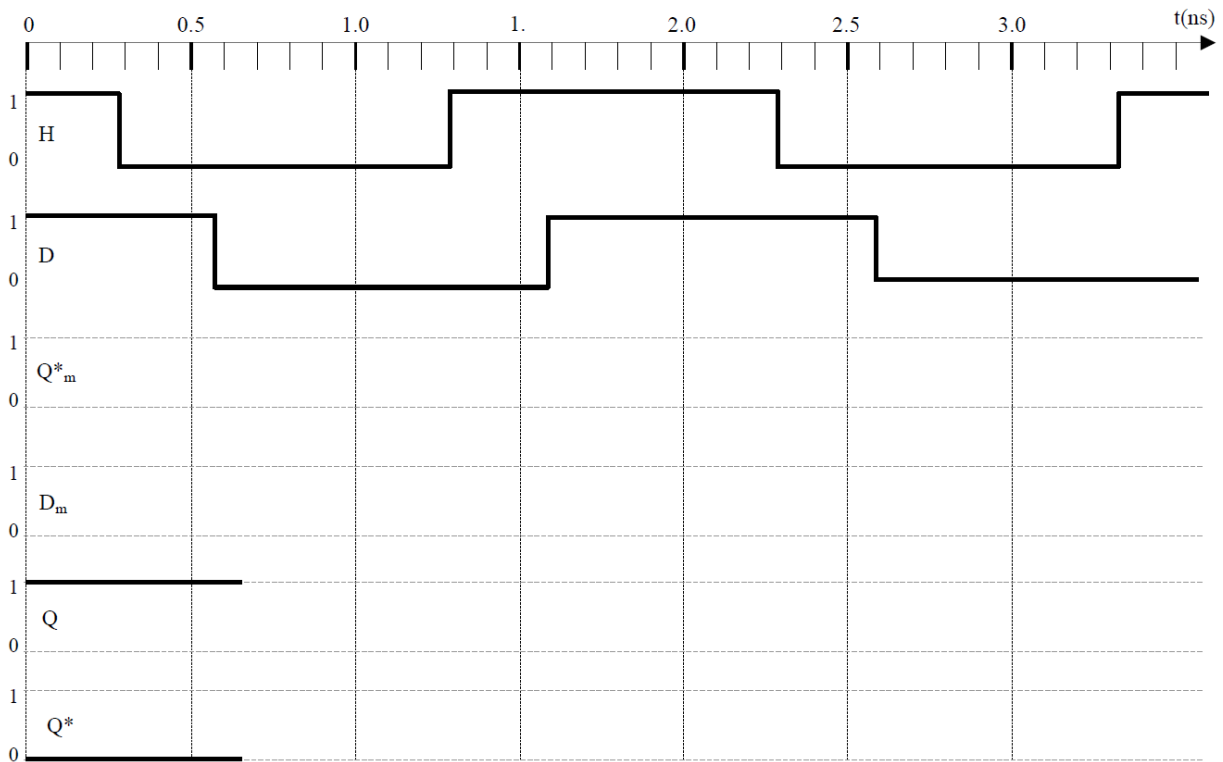


Figure 7. Chronogrammes (à compléter)

3.2 Quel est l'intérêt d'une telle structure "maître-esclave" (par rapport à la conception d'un registre à décalage par exemple) ?

3.3 Que vaut le temps de propagation de cette bascule (propagation de H vers Q) ? A quoi correspondent les temps de prépositionnement et de maintien ?

3.4 On boucle la sortie Q\* sur l'entrée D. Quelle fonction a-t-on réalisé ? A quelle fréquence maximale peut fonctionner un tel montage ?

### – Partie TP CAO –

#### Initiation au mode séquentiel (processus et assignation conditionnelle)

Un **processus** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement c'est à dire les unes à la suite des autres. Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standards de la programmation structurée comme dans les systèmes à microprocesseurs.

L'exécution d'un **processus** est déclenchée par un ou des changements d'états de signaux logiques. Le nom de ces signaux est défini dans la **liste de sensibilité** lors de la déclaration du **processus**.

**Règle à respecter pour un processus purement combinatoire :**

- Tous les signaux lus DOIVENT être dans la liste de sensibilité
- Toutes les situations doivent être prises en compte

**Règle à respecter pour un processus séquentiel synchrone :**

- Le signal horloge est obligatoirement dans la liste de sensibilité
- Seuls des signaux d'initialisation asynchrone (RAZ ou RAU) peuvent être ajoutés à la liste de sensibilité en plus du signal d'horloge.
- Toutes les situations doivent être prises en compte

1. La bascule D « flip-flop »

1.1 Décrire l'entité et l'architecture pour une bascule D « flip-flop » avec une remise à zéro (RAZ) synchrone.

Simuler la description VHDL de la bascule D sous ISIM après l'écriture d'un fichier test

1.2 Décrire l'entité et l'architecture pour une bascule D « flip-flop » avec une remise à zéro (RAZ) asynchrone.

Simuler la description VHDL de la bascule D sous ISIM après l'écriture d'un fichier test

2. Association de bascules D « flip-flop »

2.1 Décrire l'entité et l'architecture d'un registre tampon 3bits.

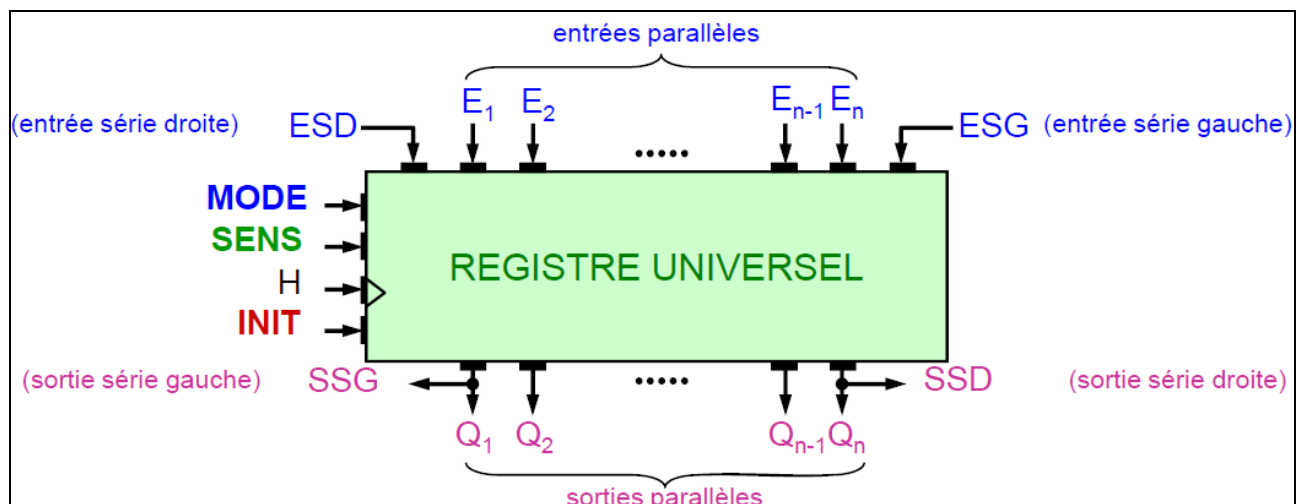
Simuler la description VHDL du registre tampon sous ISIM après l'écriture d'un fichier de test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.

2.2 Décrire l'entité et l'architecture d'un registre à décalage 3bits.

Simuler la description VHDL du registre à décalage sous ISIM après l'écriture d'un fichier de test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.

2.3 Décrire l'entité et l'architecture d'un registre universel 3 bits comme décrit par la suite.

Simuler la description VHDL du registre universel sous ISIM après l'écriture d'un fichier de test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.



## Thème de la séance 4 : fonctions séquentielles complexes

### – Partie TD –

#### Compteur modulo 6

Pour rappel, les équations séquentielles permettant d'implémenter un compteur binaire synchrone modulo  $2^n$  (à cycle complet) peuvent être mises sous la forme suivante :

$$\begin{aligned} D_1 &= \overline{Q_1} \\ D_i &= (Q_1 \dots Q_{i-1}) \oplus Q_i, \text{ pour } i > 1 \end{aligned}$$

1-1 Faire la synthèse d'un compteur binaire modulo 6 synchrone passant par les états (en décimal) 0, 1, 2, 3, 4 et 5. Remarques ?

1-2 Comment peut-on modifier ce compteur pour lui ajouter une commande de remise à zéro synchrone et une commande validation / inhibition ?

1-3 Comment peut-on modifier ce compteur pour lui ajouter une fonction de chargement parallèle ?

1-4 Où se trouve le chemin critique dans le logigramme correspondant au compteur modulo 6.

### – Partie TP CAO –

**Renforcement des acquis autour du mode séquentiel (processus et assignation conditionnelle)**

**Familiarisation avec les niveaux de description (structurel et comportemental)**

#### 1. Description d'un compteur modulo 6 au niveau structurel

Décrire en VHDL l'entité et l'architecture d'un compteur modulo 6 avec une remise à zéro (RAZ) asynchrone. Cette description correspondra à l'architecture issue de la synthèse effectuée lors de la séance TD.

Simuler la description VHDL du compteur modulo 6 sous ISIM après l'écriture d'un fichier test.

## 2. Description d'un compteur modulo 6 au niveau comportemental

2.1 Décrire en VHDL l'entité et l'architecture d'un compteur modulo 6 avec une remise à zéro (RAZ) asynchrone. La partie architecture contiendra une description du comportement du compteur au sein d'un processus.

Simuler la description VHDL du compteur modulo 6 sous ISIM après l'écriture d'un fichier test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.

2.2 Décrire en VHDL l'entité et l'architecture d'un compteur/décompteur modulo 6 avec une remise à zéro (RAZ) asynchrone. La partie architecture contiendra une description du comportement du compteur/décompteur au sein d'un processus.

Simuler la description VHDL du compteur/décompteur modulo 6 sous ISIM après l'écriture d'un fichier test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.

2.2 Compléter la description VHDL du compteur/décompteur modulo 6 avec les fonctions ENABLE et LOAD.

- ✓ Le signal ENABLE permet de bloquer le compteur/décompteur dans son état courant lorsqu'il est actif (=1).
- ✓ Quant au signal LOAD, il permet de forcer l'état courant du compteur/décompteur à une valeur fixée lorsqu'il est actif (=1).
- ✓ Simuler la nouvelle description VHDL du compteur/décompteur modulo 6 sous ISIM après l'écriture d'un fichier test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.



# Thème de la séance 5 : le compteur à modulo variable

## – Partie TD –

On souhaite réaliser un compteur 2 bits de sorties  $Q_1$   $Q_0$  avec un MODULO (nombre d'états) variable commandé par une entrée  $E_{2/3}$  :

- si  $E_{2/3} = 0$  le MODULO sera égale à 2
- si  $E_{2/3} = 1$  le MODULO sera égale à 3

### 1. Réalisation avec une machine d'états

$Q_1$  et  $Q_0$  seront directement les bits d'état

- 1.1 Donner le diagramme d'états de cet automate
- 1.2 Déterminer les équations de  $Q_{1n+1}$  et  $Q_{0n+1}$
- 1.3 Etablir le schéma complet

### 2. Réalisation avec un compteur

On dispose d'un compteur 2 bits de sorties  $Q_1$   $Q_0$  possédant une entrée de remise à 0 R asynchrone

- 2.1 Donner l'équation de la commande R en fonction des sortie  $Q_1$ ,  $Q_0$  et de l'entrée  $E_{2/3}$ .  
On utilisera pour cela un tableau de Karnaugh en remarquant que l'état de R peut être indifférent pour les états non utilisés.
- 2.2 En factorisant l'expression obtenue, montrer que la réalisation de la commande R peut se faire en utilisant 4 portes NAND à 2 entrées, soit un circuit intégré CMOS 4011.
- 2.3 En considérant le schéma adopté, donner l'équation réelle de R (t) en tenant compte du temps de propagation  $T_P$  introduit par chaque porte.
- 2.4 Pour  $E_{2/3} = 0$  simplifier l'équation de R (t) et tracer son chronogramme (on prendra  $T_P$  environ égale à  $1/10^e$  de la période d'horloge)
- 2.5 Même question pour  $E_{2/3} = 1$ . Le système fonctionne-t-il ?
- 2.6 Reprendre l'étude si l'on choisi maintenant un compteur à remise à 0 synchrone (établir l'équation de R et en déduire directement sa réalisation avec 4 portes NOR à 2 entrées)
- 2.7 On suppose que le compteur est le circuit 74HC163, et que les portes NOR sont issues du circuit 74HC02 (seules 3 portes suffiront car le dernier inverseur du schéma sera supprimé puisque l'entrée RAZ du 74HC163 est active au niveau bas). Les data sheet sont fournies dans la dernière section du poly.

Calculer la fréquence maximum de fonctionnement du système à partir des données fournies dans les datasheets. Montrer que le système peut servir de diviseur de fréquence vis-à-vis de l'horloge.

## – Partie TP CAO –

### **Initiation à la description d'une machine d'états finis en VHDL. Description à base de deux processus (un processus séquentiel et un processus combinatoire)**

#### **Présentation des types énumérés en VHDL**

##### 1. Description VHDL du compteur modulo variable réalisé à partir une machine d'états

Décrire en VHDL l'entité et l'architecture du compteur modulo variable avec une remise à zéro (RAZ) asynchrone. La partie architecture contiendra une description de la machine d'états finis. Pour ce faire, deux processus (un processus séquentiel et un processus combinatoire) seront écrits en VHDL. Par ailleurs, un type énuméré sera utilisé pour décrire l'état présent et l'état futur.

Simuler la description VHDL de la machine d'états sous ISIM après l'écriture d'un fichier test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.

##### 2. Description VHDL du compteur modulo variable réalisé à partir d'un compteur

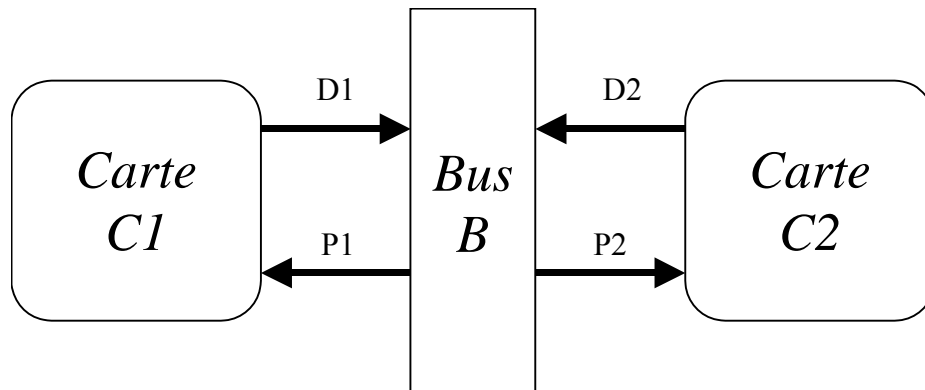
Décrire en VHDL l'entité et l'architecture du compteur modulo variable avec une remise à zéro (RAZ) asynchrone. La partie architecture sera conçu autour d'un compteur. Une description VHDL comportemental du compteur sera à privilégier.

Simuler la description VHDL de la machine d'états sous ISIM après l'écriture d'un fichier test. Visualiser le schéma RTL de la description dans l'environnement VIVADO.

## Thème de la séance 6 : synthèse d'une machine à états finis

### – Partie TD –

Deux cartes électroniques C1 et C2 ont accès à un même bus B:



Chacune des deux cartes demande sa connexion au bus en activant une entrée D qui est maintenue à 1 jusqu'à la fin de l'utilisation du bus.

Lorsque le bus établit la connexion avec l'une des cartes, le signal P correspondant est activé. Ce signal reste à 1 pendant toute la durée de la liaison.

La carte qui a utilisé le bus en dernier n'est pas prioritaire en cas de demande simultanée du bus. (à l'initialisation C1 est prioritaire)

Sachant qu'une demande de connexion doit être servie le plus rapidement possible, réaliser un automate synchrone, dont les entrées sont D1 et D2 et les sorties P1 et P2, qui permet de gérer l'accès au bus.

1-) Etablir un graphe d'états permettant de décrire le comportement de l'automate.

2-) Tracer les chronogrammes dans les deux cas suivants :

- ✓ Une demande de connexion de la carte C1 ( $D1=1, D2=0$ )
- ✓ Deux demandes simultanées de connexion au bus ( $D1=D2=1$ )

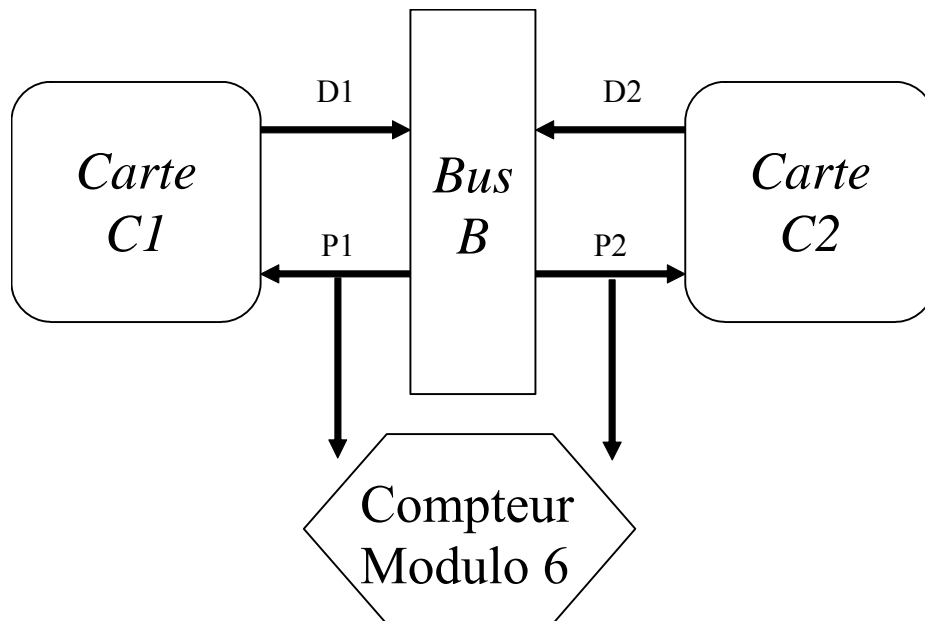
Tous les signaux (horloge, entrées, sorties, état\_présent, état\_futur) de la machine à états finis doivent être tracés.

3-) Faire la synthèse logique de l'automate à l'aide d'opérateurs logiques élémentaires.

#### 4-) *Partie optionnelle* :

Revoir le graphe d'états permettant de décrire le comportement de l'automate pour la condition suivante sur les connexions entre les cartes et le bus : **la durée d'une connexion est au minimum de 6 cycles d'exécutions.**

Cette condition peut être assurée au niveau architectural par l'utilisation d'un compteur modulo 6 comme indiqué dans le schéma suivant



– *Partie TP CAO* –

**Renforcement des acquis autour de la description d'une machine d'états finis en VHDL. Description à base de deux processus (un processus séquentiel et un processus combinatoire).**

#### Description VHDL du contrôleur d'accès au bus

Décrire en VHDL l'entité et l'architecture du contrôleur d'accès au bus étudié lors de la séance TD.

La partie architecture devra contenir une description de la machine d'états finis. Pour ce faire, deux processus (un processus séquentiel et un processus combinatoire) seront écrits en VHDL. Par ailleurs, un type énuméré sera utilisé pour décrire l'état présent et l'état futur.

Simuler la description VHDL de la machine d'états finis sous Modelsim après l'écriture d'un fichier test.

Visualiser le schéma RTL de la description dans l'environnement VIVADO.

Faire une synthèse logique automatique dans l'environnement VIVADO de l'architecture décrite en VHDL.

Visualiser le rapport de synthèse et comparer ce rapport avec la synthèse manuelle que vous avez faite lors de la séance TD. Un commentaire ?

***Partie optionnelle :***

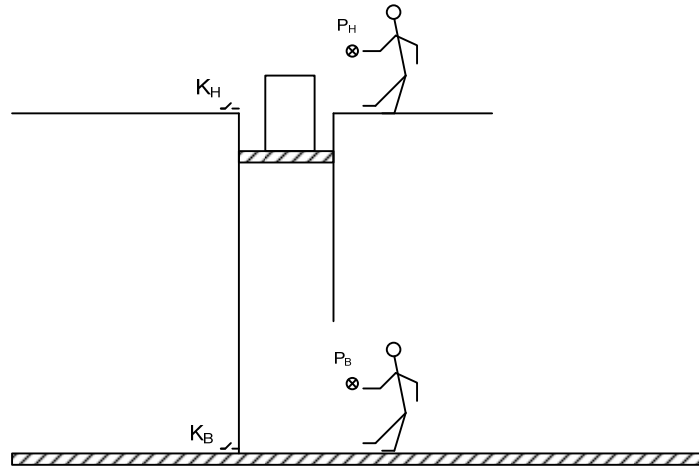
Modifier la description VHDL de l'entité et de l'architecture du contrôleur d'accès au bus pour prendre en compte une durée minimale d'accès comme vu à la fin du TD. Cela implique également d'ajouter la description d'un compteur modulo 6. Cette dernière a été faite lors du TP de la séance 4.

# Projet expérimental

Ce projet expérimental est une initiation à la conception d'architectures numériques. Il permet de passer en revue les différentes étapes qui, à partir d'un cahier des charges, aboutissent à la description d'une architecture en termes de composants symboliques pris dans une bibliothèque et de leurs interconnexions. De plus, comme l'implémentation est réalisée sur circuit FPGA, le projet expérimental doit permettre d'aller jusqu'à la configuration et au prototypage sur carte.

# Projet monte charge à deux niveaux

## 1. Description :



Un plateau monte ou descend entre le niveau du sol et un niveau supérieur.

Un capteur de position haute indique que le plateau est en haut ( $K_H = 1$ ). Un autre indique que le plateau est en bas ( $K_B = 1$ ).

Un bouton poussoir  $P_H$  est placé en haut et peut être actionné par un opérateur ( $P_H = 1$  lorsqu'il appuie).

Un autre bouton poussoir  $P_B$  est au rez-de-chaussée ( $P_B = 1$  si un opérateur à ce niveau appuie sur ce poussoir).

Un moteur permet la montée ou la descente du plateau. Il est commandé par deux signaux logiques  $S_M$  et  $S_D$  avec la convention suivante :

- ✓  $S_M = 1$  et  $S_D = 0$  : montée
- ✓  $S_M = 0$  et  $S_D = 1$  : descente
- ✓  $S_M = 0$  et  $S_D = 0$  : arrêt
- ✓ (Si  $S_M = 1$  et  $S_D = 1$  : arrêt)

Par ailleurs, un affichage s'incrémentant ou se décrémentant entre les nombres 0 et 15 renseigne les opérateurs sur les phases de montée et de descente du plateau. Lorsque le nombre 0 est affiché, cela signifie que le plateau se trouve au niveau bas. Si le nombre 15 est affiché, le plateau se trouve à l'étage supérieur.

## 2. Fonctionnement :

En marche normale, la descente se termine quand le capteur de position basse s'active, la montée se termine quand le capteur de position haute s'active.

Quand le monte charge est en haut ou en bas, il descendra (ou montera) si un opérateur quelconque appuie sur un bouton poussoir. Un arrêt d'urgence est toujours possible pendant la montée ou la descente si l'un ou l'autre des opérateurs appuie sur son bouton poussoir. Pour sortir de cette position intermédiaire, il suffit qu'un opérateur appuie sur

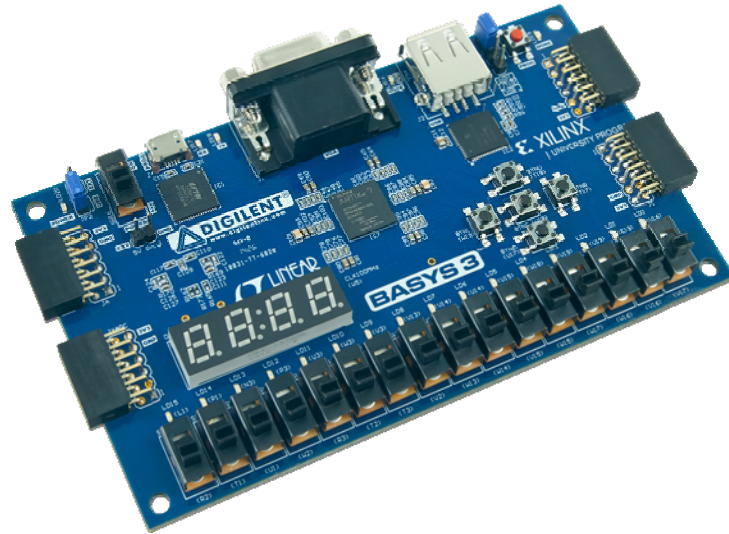
son bouton poussoir auquel cas le monte charge redémarrera dans la direction de cet opérateur. Si  $P_H$  et  $P_B$  sont détectés simultanément, le moteur resterait à l'arrêt.

La fréquence de fonctionnement de l'horloge associée au projet est de 100 MHz. Cela signifie que des signaux d'activation sur les fronts montants de l'horloge (*Clock Enable*) sont nécessaires pour afficher de l'information à des fréquences exploitables par l'œil humain. Dans ce projet, trois signaux d'activation sont nécessaires :

- $CE_{\text{traitement}}$  : ce signal commande le traitement de l'architecture proprement dit.  
Fréquence élevée pour garantir l'aléa des valeurs.  
 $\Rightarrow$  *Fréquence ~ 25 mHz*
- $CE_{\text{affichage}}$  : ce signal permet de cadencer les quatre afficheurs 7 segments en tenant compte de la perception rétinienne.  
 $\Rightarrow$  *Fréquence ~ 3 kHz*
- $CE_{\text{increment}}$  : ce signal permet de gérer le défilement des nombres de 1 à 15.  
*Fréquence ~ 1 Hz*



# Carte FPGA BASYS3



Le circuit à configurer est un FPGA Xilinx Artix 7 qui est inséré sur une carte BASYS3 commercialisée par la société Digilent. La référence complète du circuit FPGA est **XC7A35T-1CPG236C**. Cette carte comprend, outre le circuit, quatre afficheurs 7 segments, un quartz qui fournit par défaut un signal d'horloge de 100 MHz, des LEDs, des interrupteurs et d'autres périphériques comme décrit sur la Figure 1. Un site web dédié à cette carte contient toutes les informations nécessaires à l'utilisation de la carte : <http://www.digilentinc.com/Products/Detail.cfm?Prod=BASYS3>

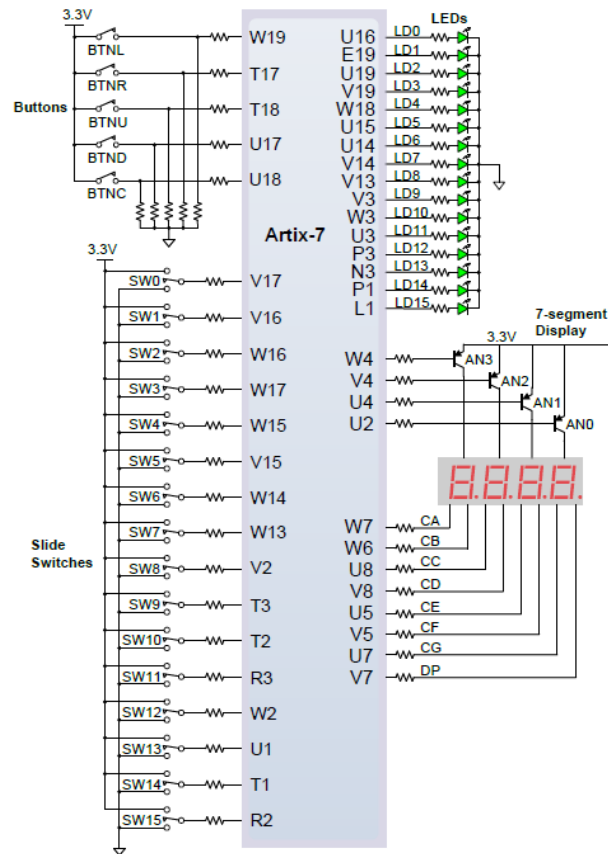


Figure 1 : schéma bloc des accès au circuit FPGA de la carte BASYS3

### Particularité dans la gestion des afficheurs 7 segments :

La carte comprend quatre afficheurs à anode commune comme le montre la Figure 2. Cela signifie que chaque segment est activé (allumé) en appliquant un 0 sur l'entrée adéquate. Par exemple, pour afficher le nombre deux, il faut la combinaison a=0, b=0, c=1, d=0, e=0, f=1 et g=0 sur les entrées.

Par ailleurs, un seul des afficheurs n'est accessible à un instant donné. C'est pourquoi, il faudra se baser sur la persistance rétinienne (capacité de l'œil à conserver des images vues superposées à une image que l'on est en train de voir) pour gérer l'affichage.

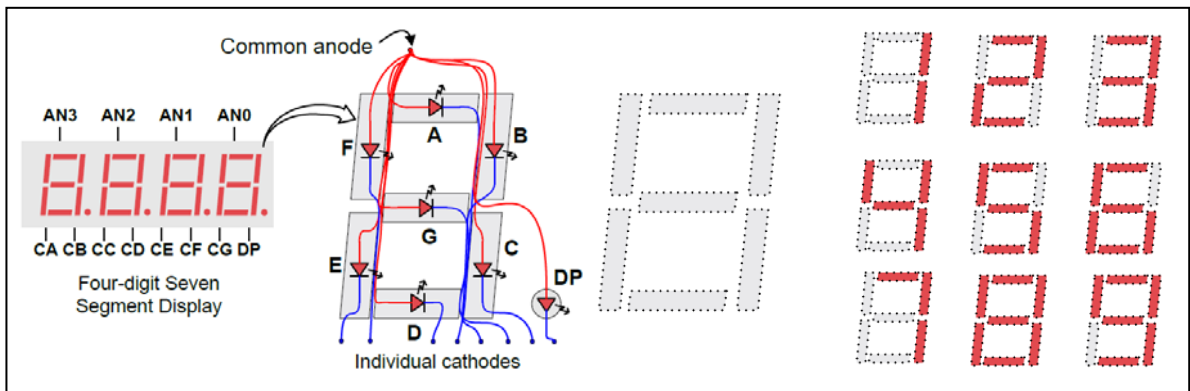


Figure 2 : principe de fonctionnement des afficheurs 7 segments

### Affectation des entrées/sorties du circuit FPGA:

L'affectation des entrées/sorties du circuit FPGA implique la spécification d'un fichier de type XDC. Les affectations nécessaires à chacun des périphériques sont données sur les cartes BASYS3. Ce fichier sera fourni par l'équipe pédagogique.

Par exemple, l'affectation des entrées/sorties pour accéder aux quatre afficheurs 7 segments est :

```
set_property PACKAGE_PIN W7 [get_ports {Sept_Segments[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sept_Segments[6]}]
set_property PACKAGE_PIN W6 [get_ports {Sept_Segments[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sept_Segments[5]}]
set_property PACKAGE_PIN U8 [get_ports {Sept_Segments[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sept_Segments[4]}]
set_property PACKAGE_PIN V8 [get_ports {Sept_Segments[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sept_Segments[3]}]
set_property PACKAGE_PIN U5 [get_ports {Sept_Segments[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sept_Segments[2]}]
set_property PACKAGE_PIN V5 [get_ports {Sept_Segments[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sept_Segments[1]}]
set_property PACKAGE_PIN U7 [get_ports {Sept_Segments[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sept_Segments[0]}]
```

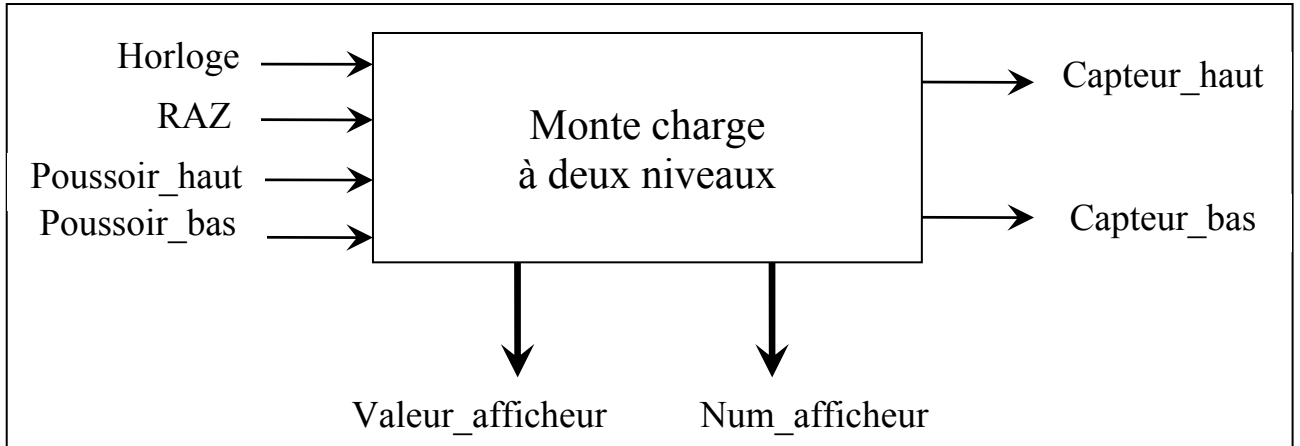
L'accès au quartz d'horloge implique quant à lui l'affectation suivante :

```
set_property PACKAGE_PIN W5 [get_ports horloge]
set_property IOSTANDARD LVCMOS33 [get_ports horloge]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports horloge]
```

# Cahier des charges

## 1. Consignes à respecter pour la définition de l'architecture

L'architecture devra respecter le schéma suivant au niveau des accès :



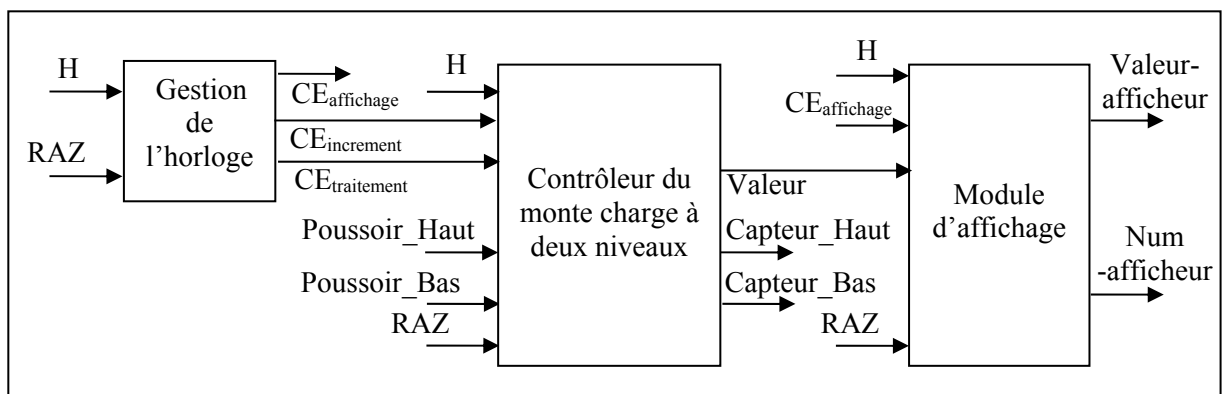
Entrées:

- ✓ Horloge: horloge du circuit fournie par un quartz à 50MHz
- ✓ RAZ : remise à zéro asynchrone de l'architecture globale
- ✓ Poussoir\_Haut : demande d'activation ou d'arrêt du monte charge
- ✓ Poussoir\_Bas : demande d'activation ou d'arrêt du monte charge

Sorties:

- ✓ Capteur\_Haut : Indication que le plateau est en haut
- ✓ Capteur\_Bas : Indication que le plateau est en bas
- ✓ Valeur\_afficheur : donnée à afficher informant de la phase de montée si incrément ou de descente si décrémentation. Les valeurs sont comprises entre 0 et 15
- ✓ Num\_Afficheur : numéro de l'afficheur 7 segments devant recevoir une valeur

L'architecture globale sera quant à elle constituée de trois modules hiérarchiques comme décrit dans le schéma suivant :



Par ailleurs, l'architecture est entièrement synchrone. Ceci implique que :

- les bascules, les registres et les compteurs sont commandés par la même horloge.
- la copie de l'entrée sur la sortie d'un bloc séquentiel sur le front montant de l'horloge est autorisée par un signal CE (Clock Enable).
- l'entrée asynchrone de mise à 0 n'est utilisée qu'à l'initialisation du circuit.  
**Elle ne doit en aucun cas être utilisée pendant le fonctionnement normal du circuit.**

## 2. Consignes à respecter pour le déroulement du projet

Il est demandé de travailler de manière progressive avec des étapes intermédiaires devant être validée par votre encadrant.

L'utilisation des outils ne doit débiter que lorsque le circuit est découpé en fonctions élémentaires (compteur, registre, décodeur, etc) et que les interconnexions entre ces fonctions sont parfaitement définies. Le schéma bloc doit être validé par votre encadrant.

## 3. Consignes à respecter pour la rédaction du rapport

Le projet numérique est en partie évalué par un rapport (un par binôme) qui résume le travail effectué lors des séances. Ce rapport devra être remis à l'enseignant qui vous a encadré. Les modalités de restitution doivent être discutées au sein de chaque groupe car les séances se déroulent du lundi au vendredi. De même, le format de restitution (papier ou fichier électronique) sera fixé pour chaque groupe par l'enseignant l'ayant encadré.

Les consignes de rédaction sont les suivantes :

### 1-) Sur le fond :

- Le schéma de l'ensemble du projet doit être présenté en expliquant le rôle de chaque fonction. Ne pas faire un copié/collé de l'énoncé mais le reformuler.
- Pour chacune des fonctions, donner le listing du code VHDL (à **commenter** en quelques lignes) et les chronogrammes de simulation (capture d'écran ISIM) permettant de vérifier que le bloc est opérationnel. Les chronogrammes devront être lisibles (!) et suffisamment commentées pour être exploitables (flèches d'indications, etc).
- Le diagramme d'états décrivant l'automate doit être donné et expliqué à partir du schéma global de la partie opérative.
- Les **résultats de synthèse** doivent être donnés. En particulier le codage des états, le nombre de ressources logiques nécessaires, l'estimation de la fréquence maximale de fonctionnement (toutes ces informations se trouvent dans le rapport de synthèse fourni par VIVADO, et **doivent être comparés avec les autres binômes afin d'en tirer des explications**).

## 2-) Sur la forme :

- La longueur conseillée pour le rapport est d'une quinzaine de pages (17 max) hors annexes.
- Il doit contenir une introduction et une conclusion.
- Numéroté toutes les figures avec une légende (automatique avec Word, se renseigner).
- Mettre un sommaire (automatique avec Word, se renseigner).
- Justifier le texte en ce qui concerne l'alignement.

En résumé, le rapport doit permettre à un lecteur connaissant la méthodologie de conception de comprendre les choix effectués et les résultats obtenus. Vous pourrez conclure en donnant votre avis sur le déroulement de ce projet dans le but d'en améliorer l'organisation.

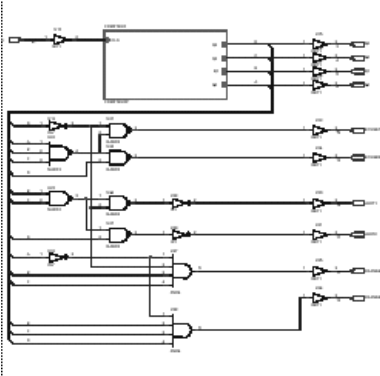
# **Présentation du langage VHDL**

VHDL signifie VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit). Ce langage a été écrit au milieu des années 80 pour réaliser la simulation de circuits électroniques. On l'a ensuite étendu en lui rajoutant des extensions pour permettre la conception (synthèse) de systèmes numériques.

Le VHDL est un langage de haut niveau qui n'est pas lié à une cible technologique (ASIC ou FPGA). Un autre de ces avantages est qu'il permet de décrire de façon très lisible et compacte le fonctionnement du système à concevoir. Il faut par contre ne jamais perdre de vue que l'exécution du code VHDL ne va pas effectuer les opérations que l'on attend du système, mais bien définir le système lui-même qui sera capable de les effectuer. En clair, les instructions VHDL seront traduites au final par des portes logiques et des bascules mais ne vont pas exécuter de tâches comme le ferait un programme en langage C par exemple. Cette confusion est pourtant très fréquente...

Ce document est une introduction à la synthèse de circuit en VHDL et présente la syntaxe de ce langage illustrée par quelques exemples. C'est un extrait du document rédigé par Philippe LETENNEUR et Philippe LECARDONNEL qui peut être consulté (<ftp://ftp.discip.crdp.ac-caen.fr/discip/crgelec/Cours/vhdl.pdf>) pour plus d'informations et d'exemples.

# I) Organisation fonctionnelle de développement d'un circuit de logique programmable



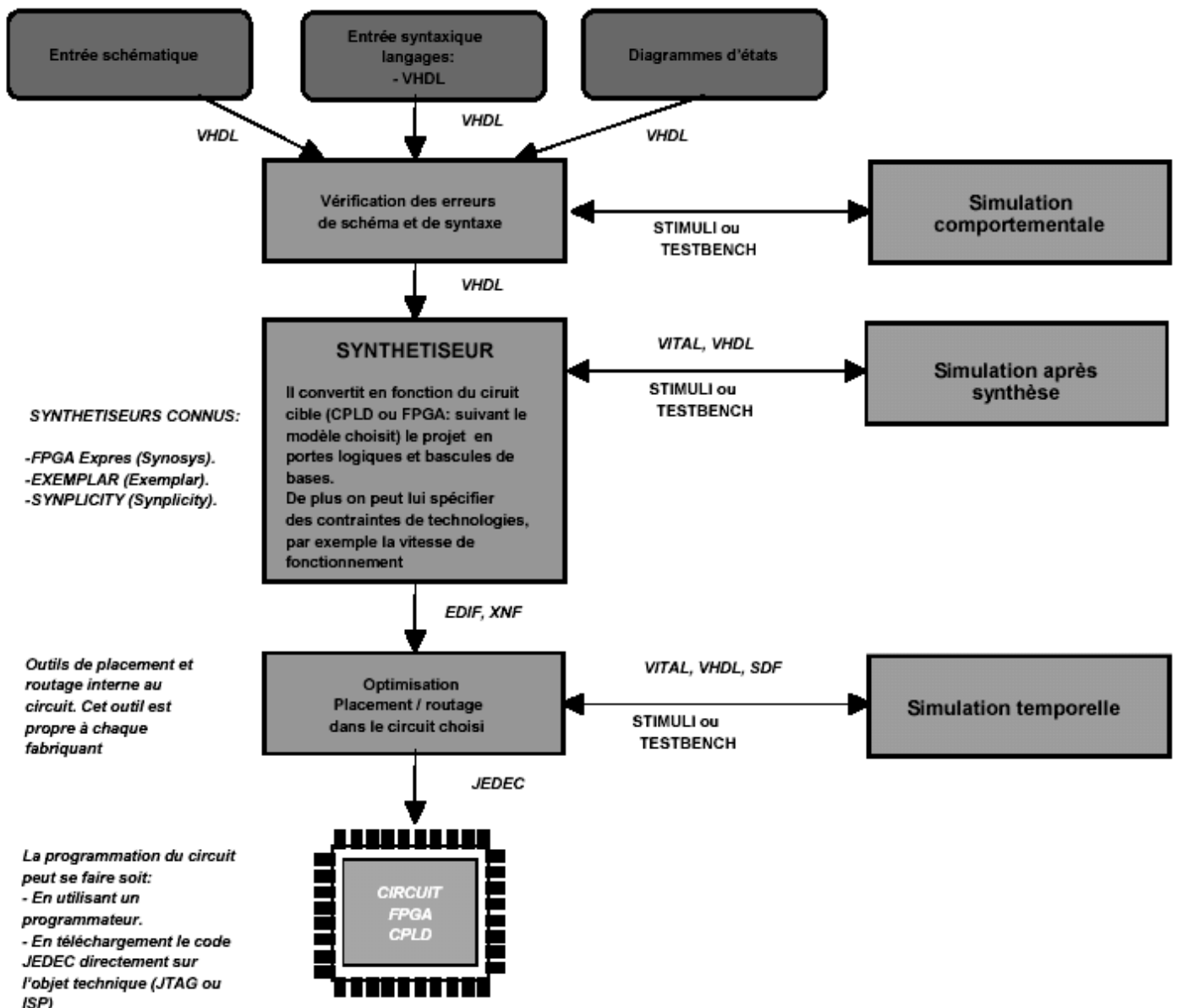
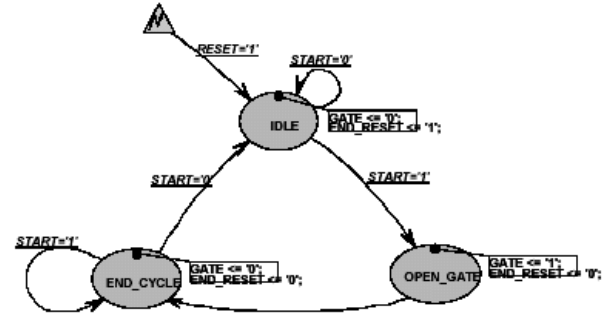
```

library IEEE;
use IEEE.std_logic_1164.all;

entity CNT_4B is
port (
CLK: in STD_LOGIC;
RESET: in STD_LOGIC;
ENABLE: in STD_LOGIC;

FULL: out STD_LOGIC;
Q: out STD_LOGIC_VECTOR
(3 downto 0)
);
end entity CNT_4B;

```



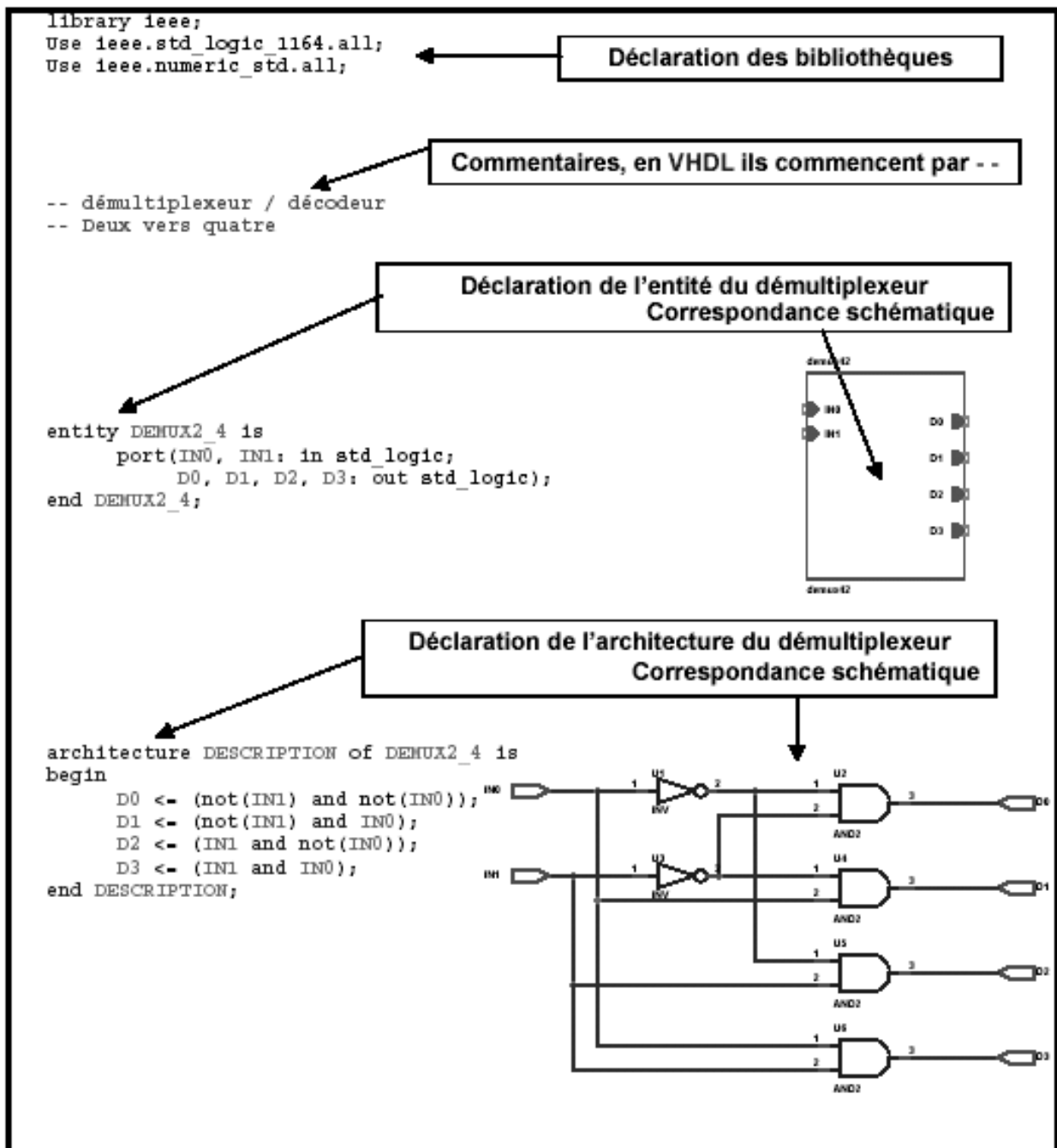


## II) Structure d'une description VHDL

Une description VHDL est composée de 2 parties indissociables :

- L'entité (*ENTITY*) où l'on définit les entrées et les sorties
- L'architecture (*ARCHITECTURE*) qui va décrire le fonctionnement attendu

Exemple : DECODEUR/DEMULPLEXEUR 2 VERS 4



## 1) Déclaration des bibliothèques

Toute description **VHDL** utilisée pour la synthèse a besoin de bibliothèques. L'**IEEE** (Institut of **E**lectrical and **E**lectronics **E**ngineers) les a normalisées et plus particulièrement la bibliothèque **IEEE1164**. Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques,...

**Library ieee;**

**Use ieee.std\_logic\_1164.all;**

**Use ieee.numeric\_std.all;**

La directive **Use** permet de sélectionner les bibliothèques à utiliser.

## 2) Déclaration de l'entité et des entrées / sorties (I/O)

Elle permet de définir le NOM de la description **VHDL** ainsi que les entrées et sorties utilisées, l'instruction qui les définit c'est port :

Syntaxe:

```
entity NOM_DE_L_ENTITE is
```

```
    port ( Description des signaux d'entrées /sorties ...);
```

```
end NOM_DE_L_ENTITE;
```

Exemple :

```
entity SEQUENCEMENT is
```

```
port (
```

```
    CLOCK      : in std_logic;
```

```
    RESET      : in std_logic;
```

```
    Q          : out std_logic_vector(1 downto 0)
```

```
);
```

```
end SEQUENCEMENT;
```

Remarque : Après la dernière définition de signal de l'instruction **port** il ne faut jamais mettre de point virgule.

L'instruction `port` .

Syntaxe: `NOM_DU_SIGNAL : sens type;`

Exemple: `CLOCK: in std_logic;`  
`BUS : out std_logic_vector (7 downto 0);`

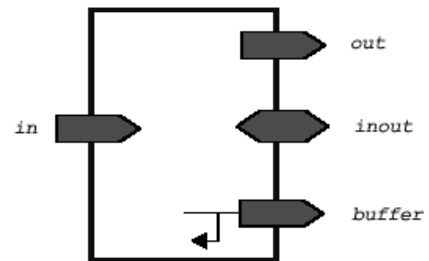
On doit définir pour chaque signal : le NOM DU SIGNAL, le sens et le type.

➤ *nom du signal :*

Le nom du signal est composé de caractères, le premier caractère doit être une lettre, sa longueur est quelconque, mais elle ne doit pas dépasser une ligne de code. Le **VHDL** n'est pas sensible à la « casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.

➤ *sens :*

- ***in*** : pour un signal en entrée.
- ***out*** : pour un signal en sortie.
- ***inout*** : pour un signal en entrée sortie
- ***buffer*** : pour un signal en sortie mais utilisé comme entrée dans la description.



➤ *type :*

Un grand nombre de types sont définis en VHDL en fonction des opérations permises sur les signaux. Le type recommandé pour les signaux d'entrées / sorties est :

- le ***std\_logic*** pour un signal.
- le ***std\_logic\_vector*** pour un bus composé de plusieurs signaux.

Un changement de type est toujours possible dans la description de l'architecture pour permettre localement certaines opérations (arithmétiques,...) sur le signal.

Par exemple un bus d'entrée de 5 bits s'écrira :

***LATCH : in std\_logic\_vector (4 downto 0) ;***

où ***LATCH(4)*** correspond au **MSB** et ***LATCH(0)*** correspond au **LSB**.

Les valeurs que peuvent prendre un signal de type *std\_logic* sont au nombre de 9:

- '0' ou 'L' : pour un niveau bas.
- '1' ou 'H' : pour un niveau haut.
- 'Z' : pour état haute impédance.
- 'W' : pour un niveau inconnu forçage faible.
- 'X' : pour un niveau inconnu forçage fort.
- 'U' : pour un signal non initialisé.
- '-' : quelconque, c'est à dire n'importe quelle valeur.

### 3) Déclaration de l'architecture

L'architecture décrit le fonctionnement de l'entité correspondante.

#### Exemples :

```
-- Opérateurs logiques de base
entity PORTES is
  port (A,B :in std_logic;
        Y1,Y2,Y3,Y4,Y5,Y6,Y7:out std_logic);
end PORTES;

architecture DESCRIPTION of PORTES is
begin
  Y1 <= A and B;
  Y2 <= A or B;
  Y3 <= A xor B;
  Y4 <= not A;
  Y5 <= A nand B;
  Y6 <= A nor B;
  Y7 <= not(A xor B);
end DESCRIPTION;

-- Décodeurs 7 segments
entity DEC7SEG4 is
  port (DEC :in std_logic_vector(3 downto 0);
        SEG:out std_logic_vector(0 downto 6));
end DEC7SEG4;

architecture DESCRIPTION of DEC7SEG4 is
begin
  SEG <= "1111110" when DEC = 0
  else "0110000" when DEC = 1
  else "1101101" when DEC = 2
  else "1111001" when DEC = 3
  else "0110011" when DEC = 4
  else "1011011" when DEC = 5
  else "1011111" when DEC = 6
  else "1110000" when DEC = 7
  else "1111111" when DEC = 8
  else "1111011" when DEC = 9
  else "-----";
end DESCRIPTION;
```

### III) Le mode combinatoire

Pour une description **VHDL** toutes les instructions sont évaluées et affectent les signaux de sortie en même temps. **L'ordre dans lequel elles sont écrites n'a aucune importance**. En effet la description génère des structures électroniques, c'est la grande différence entre une description **VHDL** et un langage informatique classique.

Dans un système à microprocesseur, les instructions sont exécutées les unes à la suite des autres. En **VHDL** il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description, ce qui n'est pas toujours évident...

#### 1) Les opérateurs

##### a) L'affectation simple <=

Dans une description VHDL c'est certainement l'opérateur le plus utilisé. En effet il permet de modifier l'état d'un signal en fonction d'autres signaux et/ou d'autres opérateurs.

**Exemple avec des portes logiques :  $S1 <= E2 \text{ and } E1$  ;**

Les valeurs numériques que l'on peut affecter à un signal sont au nombre de 9 :

- '1' ou 'H' pour un niveau **haut** avec un signal de **1** bit.
- '0' ou 'L' pour un niveau **bas** avec un signal de **1** bit.
- 'W' pour un niveau inconnu forçage faible avec un signal de **1** bit.
- 'X' pour un niveau inconnu forçage fort avec un signal de **1** bit.
- 'Z' pour un état haute impédance avec un signal de **1** bit.
- 'U' pour un signal non initialisé avec un signal de **1** bit.
- '-' pour un état quelconque, c'est à dire '0' ou '1'. Cette valeur est très utilisée avec les instructions : *when ... else* et *with .... Select ....*
- Pour les **signaux** composés de plusieurs bits on utilise les guillemets " ... " , voir les exemples ci dessous :
- Les bases numériques utilisées pour les bus peuvent être :

<b>BINAIRE</b> ,	exemple : $BUS <= "1001"$ ;	-- $BUS = 9$ en décimal
<b>HEXA</b> ,	exemple : $BUS <= X"9"$ ;	-- $BUS = 9$ en décimal
<b>OCTAL</b> ,	exemple : $BUS <= O"11"$ ;	-- $BUS = 9$ en décimal

**Exemple:**

```
Library ieee;
Use ieee.std_logic_1164.all;

entity AFFEC is
port (
    E1,E2           : in std_logic;
    BUS1,BUS2,BUS3  : out std_logic_vector(3 downto 0);
    S1,S2,S3,S4     : out std_logic);
end AFFEC;

architecture DESCRIPTION of AFFEC is

begin
S1 <= '1'; -- S1 = 1
S2 <= '0'; -- S2 = 0
S3 <= E1;  -- S3 = E1
S4 <= '1' when (E2 = '1') else 'Z'; -- S4 = 1 si E1=1 sinon S4
-- prend la valeur haute impédance
BUS1 <= "1000"; -- BUS1 = "1000"
BUS2 <= E1 & E2 & "10"; -- BUS2 = E1 & E2 & 10
BUS3 <= x"A"; -- valeur en HEXA -> BUS3 = 10(déc)

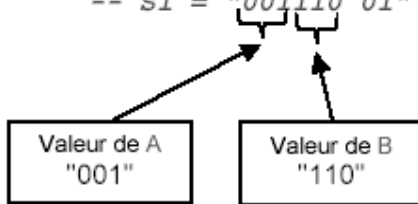
end DESCRIPTION;
```

**b) Opérateurs de concaténation &**

Cet opérateur permet de joindre des signaux entre eux.

**Exemple :**

```
-- Soit A et B de type 3 bits et S1 de type 8 bits
-- A = "001" et B = "110"
S1 <= A & B & "01" ;
-- S1 prendra la valeur suivante après cette affectation
-- S1 = "001110 01"
```



**c) Opérateurs logiques**

Opérateur	VHDL
ET	and
NON ET	nand
OU	or
NON OU	nor
OU EXCLUSIF	xor
NON OU EXCLUSIF	xnor
NON	not
DECALAGE A GAUCHE	sll
DECALAGE A DROITE	srl
ROTATION A GAUCHE	rol
ROTATION A DROITE	ror

Exemples :

**S1 <= A sll 2 ;** -- S1 = A décalé de 2 bits à gauche.

**S2 <= A rol 3 ;** -- S2 = A avec une rotation de 3 bits à gauche

**S3 <= not (R);** -- S3 = R

Remarque : Pour réaliser des décalages logiques en synthèse logique, il est préférable d'utiliser les instructions suivantes :

**Décalage à droite :**

```
-- Si A est de type std_logic_vector(7 downto 0)
S1 <= '0' & A(7 downto 1); -- décalage d'un bit à droite
S1 <= "000" & A(7 downto 3); -- décalage de trois bits à droite
```

**Décalage à gauche :**

```
-- Si A est de type std_logic_vector(7 downto 0)
S1 <= A(6 downto 0) & '0'; -- décalage d'un bit à gauche
S1 <= A(4 downto 0) & "000"; -- décalage de trois bits à gauche
```

(cette autre écriture évite ainsi l'utilisation de portes logiques !)

*e) Opérateurs relationnels*

Ils permettent de modifier l'état d'un signal ou de signaux suivant le résultat d'un test ou d'une condition. En logique combinatoire ils sont souvent utilisés avec les instructions :

- **when ... else ...**
- **with .... Select ....**

Opérateur	VHDL
Egal	=
Non égal	/=
Inférieur	<
Inférieur ou égal	<=
Supérieur	>
Supérieur ou égal	>=

## 2) Les instructions

### a) Affectation conditionnelle

Cette instruction modifie l'état d'un signal suivant le résultat d'une condition logique entre un ou des signaux, valeurs, constantes.

```
SIGNAL <= expression when condition  
         [else expression when condition]  
         [else expression];
```

**Remarque :** l'instruction **[else expression]** n'est pas obligatoire mais elle est fortement conseillée car elle permet de définir la valeur du **SIGNAL** dans le cas où la condition n'est pas remplie.

On peut mettre en cascade cette instruction (voir l'exemple N°2).

#### Exemple N°1 :

```
-- S1 prend la valeur de E2 quand E1='1' sinon S1 prend la --  
valeur '0'  
S1 <= E2 when ( E1= '1' ) else '0';
```

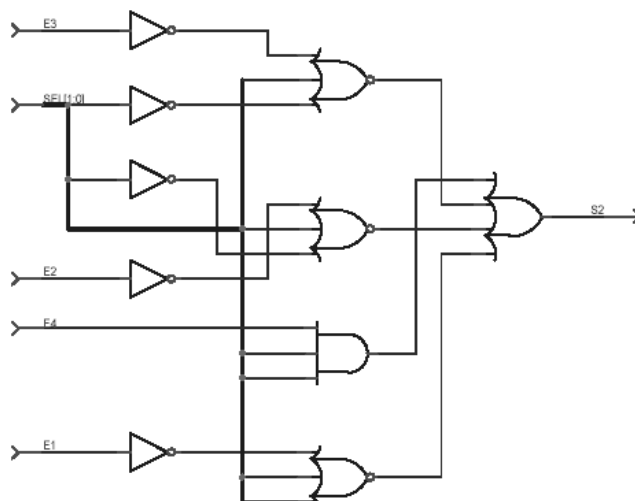
#### Schéma correspondant :



#### Exemple N°2 :

```
-- Structure évoluée d'un multiplexeur 4 vers 1  
S2 <= E1 when (SEL="00" ) else  
      E2 when (SEL="01" ) else  
      E3 when (SEL="10" ) else  
      E4 when (SEL="11" )  
      else '0';
```

#### Schéma correspondant après synthèse:





### b) Affectation sélective

Cette instruction permet d'affecter différentes valeurs à un signal, selon les valeurs prises par un signal dit de sélection.

```
with SIGNAL_DE_SELECTION select  
    SIGNAL <= expression when valeur_de_selection,  
             [expression when valeur_de_selection,]  
             [expression when others];
```

**Remarque:** l'instruction **[expression when others]** n'est pas obligatoire mais fortement conseillée car elle permet de définir la valeur du **SIGNAL** dans le cas où la condition n'est pas remplie.

#### Exemple N°1 :

```
-- Multiplexeur 4 vers 1  
with SEL select  
    S2 <=  E1 when "00",  
           E2 when "01",  
           E3 when "10",  
           E4 when "11",  
           '0' when others;
```

**Remarque:** **when others** est nécessaire car il faut toujours définir les autres cas du signal de sélection pour prendre en compte toutes les valeurs possibles de celui-ci.

## IV) Le mode séquentiel

### 1) Définition d'un PROCESS

Un **process** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement c'est à dire les unes à la suite des autres. Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standards de la programmation structurée comme dans les systèmes à microprocesseurs.

L'exécution d'un **process** est déclenchée par un ou des changements d'états (*event*) de signaux logiques. Le nom de ces signaux est défini dans **la liste de sensibilité** lors de la déclaration du **process**.

[Nom\_du\_process :] process (Liste\_de\_sensibilité\_nom\_des\_signaux)

**Begin**

-- instructions du process

**end process** [Nom\_du\_process] ;

**Remarque:** Le nom du **process** entre crochet est facultatif, mais il peut être très utile pour repérer un **process** parmi d'autres lors de phases de mise au point ou de simulations.

### Règles de fonctionnement d'un process :

- 1) L'exécution d'un **process** a lieu à chaque changement d'état d'un signal de la liste de sensibilité.
- 2) Les instructions du **process** s'exécutent séquentiellement.
- 3) Les changements d'état des signaux par les instructions du **process** sont pris en compte à la **fin** du **process**.

### 2) Les 2 principales structures utilisées dans un process

L'assignation conditionnelle	L'assignation sélective
<pre>if condition then     instructions [elsif condition then instructions] [else instructions] end if ;</pre> <p><b>Exemple:</b> if (RESET='1') then SORTIE &lt;= "0000"; end if ;</p>	<pre>case signal_de_slection is when valeur_de_sélection =&gt; instructions [when others =&gt; instructions] end case;</pre> <p><b>Exemple:</b> case SEL is when "000" =&gt; S1 &lt;= E1; when "001" =&gt; S1 &lt;= '0'; when "010"   "011" =&gt; S1 &lt;='1'; -- La barre   permet de réaliser -- un ou logique entre les deux -- valeurs "010" et "011" when others =&gt; S1 &lt;= '0'; end case;</p>

### 3) Exemples de process

#### Exemple N°1 : Déclaration d'une bascule D.

```
Library ieee;
Use ieee.std_logic_1164.all;

entity BASCULED is
  port (
    D,CLK : in std_logic;
    S      : out std_logic);
end BASCULED;
architecture DESCRIPTION of BASCULED is
begin
  PRO_BASCULED : process (CLK)
  begin
    if (CLK'event and CLK='1') then
      S <= D;
    end if;
  end process PRO_BASCULED;
end DESCRIPTION;
```

#### Remarques

- Seul le signal *CLK* fait partie de la liste de sensibilité. D'après les règles de fonctionnement énoncées précédemment, seul un changement d'état du signal *CLK* va déclencher l'exécution du process et par conséquent évaluer les instructions de celui-ci.
- L'instruction *if (CLK'event and CLK='1') then* permet de détecter un front montant du signal *CLK*. La détection de front est réalisée par l'attribut *event* appliqué à l'horloge *CLK*. Si on veut un déclenchement sur un front descendant, il faut écrire l'instruction suivante : *if (CLK'event and CLK='0')*.
- Les bibliothèques IEEE possèdent deux instructions permettant de détecter les fronts montants) *rising\_edge(CLK)* ou descendants *falling\_edge(CLK)*.
- Si la condition est remplie alors le signal de sortie *S* sera affecté avec la valeur du signal d'entrée *D*.

**Exemple N°2** : Même exemple que précédemment mais avec des entrées de présélections de mise à zéro *RESET* prioritaire sur l'entrée de mise à un *SET*, toutes les deux sont synchrones de l'horloge *CLK*.

```
Library ieee;
Use ieee.std_logic_1164.all;

entity BASCULEDSRS is
  port (
    D,CLK,SET,RESET  : in std_logic;
    S                : out std_logic);
end BASCULEDSRS;

architecture DESCRIPTION of BASCULEDSRS is
begin
  PRO_BASCULEDSRS : process (CLK)
  Begin
    if (CLK'event and CLK = '1') then
      if (RESET = '1') then
        S <= '0';
      elsif (SET = '1') then
        S <= '1';
      else
        S <= D;
      end if;
    end if;
  end process PRO_BASCULEDSRS;
end DESCRIPTION;
```

**Exemple N°3** : Même exemple que précédemment mais avec des entrées de présélections, de mise à zéro *RESET* prioritaire sur l'entrée de mise à un *SET*, toutes les deux sont asynchrones de l'horloge *CLK*.

```
Library ieee;
Use ieee.std_logic_1164.all;

entity BASCULEDSRA is
  port (
    D,CLK,SET,RESET  : in std_logic;
    S                : out std_logic);
end BASCULEDSRA;

architecture DESCRIPTION of BASCULEDSRA is
begin
  PRO_BASCULEDSRA : process (CLK,RESET,SET)
  Begin
    if (RESET = '1') then
      S <= '0';
    elsif (SET = '1') then
      S <= '1';
    elsif (CLK'event and CLK = '1') then
      S <= D;
    end if;
  end process PRO_BASCULEDSRA;
end DESCRIPTION;
```

**Remarque** : l'entrée **RESET** est prioritaire sur l'entrée **SET**. Elles sont toutes les deux asynchrones car les signaux d'entrée **SET** et **RESET** sont mentionnés dans la liste de sensibilité du *process*.

#### **Exemple N°4 : compteur 3 bits avec remise à 0 synchrone**

##### **1-) Description recommandée :**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CMP3BITS is
  Port ( CLOCK : in STD_LOGIC;
        RESET : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR (2 downto 0));
end CMP3BITS;

architecture DESCRIPTION of CMP3BITS is

  signal CMP : unsigned (2 downto 0);

begin

  Synchrone : Process(CLOCK)
  begin
    if (CLOCK='1' and CLOCK'event) then
      if (RESET='1') then
        CMP <= "000";
      else
        CMP <= CMP + "001";
      end if;
    end if;
  end Process Synchrone;

  Q <= std_logic_vector (CMP);

end DESCRIPTION;
```

**Pour pouvoir utiliser les opérateurs arithmétiques, il faut rajouter le package normalisé: ieee.numeric\_std.all.** Ce dernier permet de manipuler uniquement des signaux de type *unsigned* ou *signed*.

Le type *unsigned* représente des valeurs numériques entières entre 0 et  $2^{N-1}$ .

Le type *signed* représente des valeurs numériques entières entre  $-2^{N-1}$  et  $2^{N-1}-1$  en complément à 2.

C'est la raison pour laquelle des transformations de type sont nécessaires en amont et en aval d'un calcul.

- ✓ Transformation Std\_logic vers Unsigned : unsigned(operand)
- ✓ Transformation Std\_logic vers Signed : signed(operand)
- ✓ Transformation Unsigned/Signed vers Std\_logic: std\_logic(operand)

L'incrémentation du compteur est réalisée par l'opérateur + associé à la valeur **1**. Cela est logique pour nous, mais elle l'est beaucoup moins pour les architectures numériques. En

effet, les entrées et sorties sont déclarés de type *std\_logic* ou *std\_logic\_vector*, c'est-à-dire qu'un bit peut prendre comme valeur les états '1' ou '0' et un bus n'importe quelle valeur, du moment qu'elle est écrite entre deux guillemets "1010" ou X"A" ou o"12", mais pas une valeur comme par exemple 1,2,3,4. Les valeurs décimales sont interprétées par le synthétiseur comme des valeurs entières (**integer**). Du coup, nous ne pouvons pas par défaut additionner un nombre entier 1 avec un bus de type électronique (*std\_logic\_vector*). C'est pour cela que l'on rajoute la valeur binaire "001" au signal CMP de type unsigned.

La description fait appel à un signal interne *CMP*. En effet, le signal *S* est déclaré comme une sortie dans l'entité et ne peut donc être lu. Pour contourner cette difficulté on utilise un signal interne qui peut être à la fois lu et modifié.

La mise à zéro des sorties du compteur passe par l'instruction : ***CMP* <= "0000";**

Une autre façon d'écrire cette instruction est : ***CMP* <= (others => '0');**

Elle est très intéressante car elle permet de s'affranchir de la taille du bus.

## 2-) Autre description possible mais fortement déconseillée :

```

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (
    CLOCK : in std_logic;
    RESET : in std_logic;
    Q      : out std_logic_vector (2 downto 0));
end CMP3BITS;
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
    process (CLOCK)
    begin
        if (CLOCK = '1' and CLOCK'event) then
            if RESET = '1' then
                CMP <= "000";
            else
                CMP <= CMP + 1;
            end if;
        end if;
    end process;
    Q <= CMP;
end DESCRIPTION;

```

**ATTENTION :** Cette description est déconseillée car elle repose sur le package *ieee.std\_logic\_unsigned.all* qui n'est pas normalisé.

## 4) Recommandations

- ① **Ne pas oublier que l'affectation des signaux n'a lieu qu'à la sortie du process !** (c'est avec les signaux internes qu'on se fait souvent piéger...)
  
- ② **Un même signal doit toujours être affecté dans un même process.** Sinon le synthétiseur crée automatiquement un bus car on ne peut relier entre elles 2 sorties de bascules ! (Il peut être préférable d'utiliser le type *std\_ulogic* au lieu de *std\_logic* ce qui générera une erreur au moment de la compilation)

## V) La description structurelle en VHDL

### 1) Préambule

La description structurelle d'un circuit complexe en vhdl présente de nombreux avantages :

- Une architecture hiérarchique compréhensible : il est plus simple de séparer un circuit en un ensemble de blocs plus petits, ayant des fonctions bien identifiées. Ces blocs pourront alors être décrits sous forme comportementale, ou bien à leur tour être séparés en blocs encore plus simples.
- Une synthèse logique efficace : la synthèse est un processus lent (en terme de temps de calcul). Plus un bloc est gros et complexe, plus sa synthèse prendra du temps. Il vaut donc mieux travailler sur des blocs plus petits, plus simples à synthétiser, et rassembler le tout à la fin.

L'objectif de cette section est de voir précisément comment coder une représentation structurelle d'un circuit, autrement dit :

- comment déclarer des blocs (qu'on appellera composant)
- comment utiliser un composant (qui devient une "instance") et déclarer la façon dont il est connecté.
- comment choisir l'architecture d'un composant quand il y en a plusieurs (configurations)

### 2) Elément de base

VHDL permet l'assemblage de "**composants**" ce qui constitue une description structurelle. Ce composant peut être appelé plusieurs fois dans un même circuit. Pour différencier ces mêmes composants, il est nécessaire de leur donner un nom d'"**instance**". L'appel d'un composant se dit aussi "**instanciation**"

De façon à instancier un composant il est nécessaire de connaître :

- Le prototype du composant (ses ports d'entrée et de sortie). La directive component peut être utilisée à cette fin.
- A quelle entité et architecture est lié chaque instance de composant. Ce lien peut être connu grâce à l'unité de configuration.

Il est important de noter :

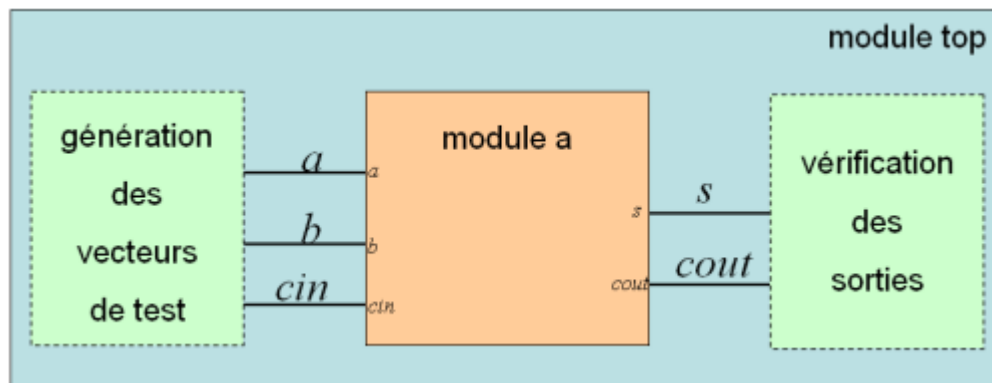
- La déclaration du composant (directive component ) est redondante textuellement avec celle de l'entité associée mais permet :
  1. Une compilation indépendante entre l'entité associée au composant et le circuit utilisant le composant.
  2. La conception descendante. Le composant peut être déclaré avant l'entité associée.



- La configuration est une unité de compilation **optionnelle**, très utile pour les gros circuits. Par exemple pour accélérer la simulation, un même composant peut être associé à un couple entité/architecture détaillé et synthétisable ou un autre couple plus abstrait et plus rapide à simuler. Pour ne pas utiliser de configuration, une règle fréquente est d'utiliser le même nom pour le composant et l'entité associée, c'est le cas pour ISIM et les outils de synthèse FPGA.

La description structurelle est nécessaire pour simuler un circuit dont les vecteurs stimulés sont eux mêmes issus d'un modèle VHDL. Le modèle de plus haut niveau fait donc appel au circuit à tester (Device Under Test) et d'un générateur de stimuli. Ces deux objets sont instanciés dans un même circuit, généralement appelé "testbench" (mais ce n'est pas une obligation) qui est autonome : il n'aura *pas* d'entrées ni de sorties.

Exemple : le circuit top, servant à simuler le circuit "module a" doit être autonome : son entité n'a pas d'entrée ni de sortie.



**Cas particulier de la simulation : circuit "top" sans entrée ni sortie**

### 3) Déclaration et instanciation des composants

#### Déclaration

Le mot clé component sert à déclarer le prototype d'interconnexion. La syntaxe est presque identique à celle de l'entité :

```
component AND_2
port (
  a : in std_logic;
  b : in std_logic;
  s : out std_logic);
end component;
```

Pour créer rapidement un composant, une opération copier/coller de l'entité en enlevant le littéral "IS" suffit.

## Instanciation

L'instanciation d'un composant se fait dans le corps de l'architecture de cette façon :

```
<NOM_INSTANCE>:<NOM_COMPOSANT> port map(LISTE DES CONNEXIONS);
```

### Exemple:

```
entity AND_3 is
  port (
    e1 : in std_logic;
    e2 : in std_logic;
    e3 : in std_logic;
    s  : out std_logic
  );
end entity;
--
architecture arc of AND_3 is
  --
  signal z : bit;
  component and2
    port (
      a : in std_logic;
      b : in std_logic;
      s : out std_logic);
  end component;
  --
  begin
    inst1 : and2 port map (a=>e1, b=>e2 , s=>z);
    inst2 : and2 port map (z, e3, s);
  end arc
```

Dans cet exemple, 2 instances de composant "and2" sont appelées pour créer une porte ET à 3 entrées.

L'association des ports du composant aux signaux de l'instance se fait à l'aide de la clause *port map*.

La syntaxe des associations est soit

1. par nom où chaque broche du composant est associée à un signal : cas de *inst\_1*
2. positionnelle où l'ordre des signaux correspond à l'ordre des broches : cas de *inst\_2*

## V) Bibliographie

- J. Weber, S. Moutault, M. Meaudre, *Le langage VHDL – du langage au circuit, du circuit au langage*, 3<sup>ème</sup> éd., Dunod – 2007.
- R. Airiau, J.M. Bergé, V. Olive, J. Rouillard, *VHDL du langage à la modélisation*, Presses Polytechniques et Universitaires Romandes – 1990.
- Z. Navabi, *VHDL : Analysis and Modeling of Digital Systems*, McGraw Hill – 1993.

# **DATA SHEET**



# 74HC02

## Quad 2-Input NOR Gate

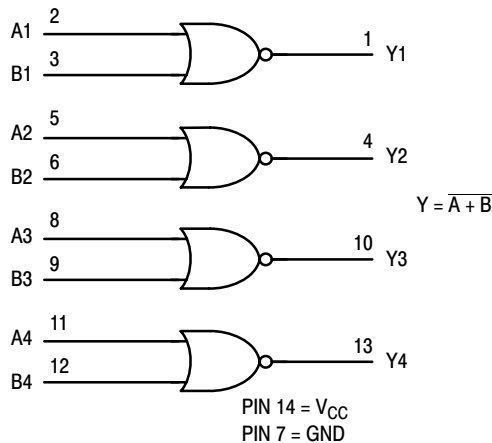
### High-Performance Silicon-Gate CMOS

The 74HC02 is identical in pinout to the LS02. The device inputs are compatible with standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

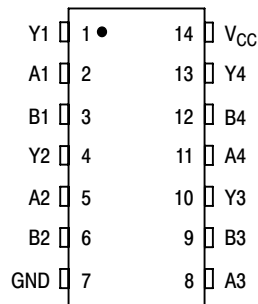
#### Features

- Output Drive Capability: 10 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS, and TTL
- Operating Voltage Range: 2.0 to 6.0 V
- Low Input Current: 1.0  $\mu$ A
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance with the Requirements Defined by JEDEC Standard No. 7A
- ESD Performance: HBM > 2000 V; Machine Model > 200 V
- Chip Complexity: 40 FETs or 10 Equivalent Gates
- These are Pb-Free Devices

#### LOGIC DIAGRAM



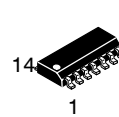
#### PIN ASSIGNMENT



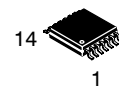
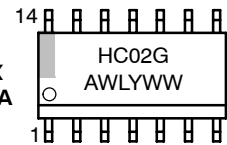
**ON Semiconductor**<sup>®</sup>

<http://onsemi.com>

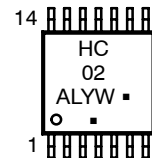
#### MARKING DIAGRAMS



**SOIC-14  
D SUFFIX  
CASE 751A**



**TSSOP-14  
DT SUFFIX  
CASE 948G**



HC02 = Device Code  
A = Assembly Location  
WL or L = Wafer Lot  
Y = Year  
WW or W = Work Week  
G or ■ = Pb-Free Package

(Note: Microdot may be in either location)

#### FUNCTION TABLE

Inputs		Output
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

#### ORDERING INFORMATION

See detailed ordering and shipping information in the package dimensions section on page 4 of this data sheet.

## 74HC02

### MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
$V_{CC}$	DC Supply Voltage (Referenced to GND)	- 0.5 to + 7.0	V
$V_{in}$	DC Input Voltage (Referenced to GND)	- 0.5 to $V_{CC} + 0.5$	V
$V_{out}$	DC Output Voltage (Referenced to GND)	- 0.5 to $V_{CC} + 0.5$	V
$I_{in}$	DC Input Current, per Pin	$\pm 20$	mA
$I_{out}$	DC Output Current, per Pin	$\pm 25$	mA
$I_{CC}$	DC Supply Current, $V_{CC}$ and GND Pins	$\pm 50$	mA
$P_D$	Power Dissipation in Still Air, SOIC Package† TSSOP Package†	500 450	mW
$T_{stg}$	Storage Temperature	- 65 to + 150	°C
$T_L$	Lead Temperature, 1 mm from Case for 10 Seconds SOIC or TSSOP Package	260	°C

This device contains protection circuitry to guard against damage due to high static voltages or electric fields. However, precautions must be taken to avoid applications of any voltage higher than maximum rated voltages to this high-impedance circuit. For proper operation,  $V_{in}$  and  $V_{out}$  should be constrained to the range  $GND \leq (V_{in} \text{ or } V_{out}) \leq V_{CC}$ . Unused inputs must always be tied to an appropriate logic voltage level (e.g., either GND or  $V_{CC}$ ). Unused outputs must be left open.

Stresses exceeding Maximum Ratings may damage the device. Maximum Ratings are stress ratings only. Functional operation above the Recommended Operating Conditions is not implied. Extended exposure to stresses above the Recommended Operating Conditions may affect device reliability.

†Derating — SOIC Package: - 7 mW/°C from 65° to 125°C  
TSSOP Package: - 6.1 mW/°C from 65° to 125°C

For high frequency or heavy load considerations, see Chapter 2 of the ON Semiconductor High-Speed CMOS Data Book (DL129/D).

### RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Unit	
$V_{CC}$	DC Supply Voltage (Referenced to GND)	2.0	6.0	V	
$V_{in}, V_{out}$	DC Input Voltage, Output Voltage (Referenced to GND)	0	$V_{CC}$	V	
$T_A$	Operating Temperature, All Package Types	- 55	+ 125	°C	
$t_r, t_f$	Input Rise and Fall Time (Figure 1)	$V_{CC} = 2.0 \text{ V}$ $V_{CC} = 4.5 \text{ V}$ $V_{CC} = 6.0 \text{ V}$	0 0 0	1000 500 400	ns

# 74HC02

## DC ELECTRICAL CHARACTERISTICS (Voltages Referenced to GND)

Symbol	Parameter	Test Conditions	V <sub>CC</sub> (V)	Guaranteed Limit			Unit
				- 55 to 25°C	≤ 85°C	≤ 125°C	
V <sub>IH</sub>	Minimum High-Level Input Voltage	V <sub>out</sub> = 0.1 V or V <sub>CC</sub> - 0.1 V  I <sub>out</sub>   ≤ 20 μA	2.0	1.5	1.5	1.5	V
			3.0	2.1	2.1	2.1	
			4.5	3.15	3.15	3.15	
			6.0	4.2	4.2	4.2	
V <sub>IL</sub>	Maximum Low-Level Input Voltage	V <sub>out</sub> = 0.1 V or V <sub>CC</sub> - 0.1 V  I <sub>out</sub>   ≤ 20 μA	2.0	0.5	0.5	0.5	V
			3.0	0.9	0.9	0.9	
			4.5	1.35	1.35	1.35	
			6.0	1.8	1.8	1.8	
V <sub>OH</sub>	Minimum High-Level Output Voltage	V <sub>in</sub> = V <sub>IH</sub> or V <sub>IL</sub>  I <sub>out</sub>   ≤ 20 μA	2.0	1.9	1.9	1.9	V
			4.5	4.4	4.4	4.4	
			6.0	5.9	5.9	5.9	
		V <sub>in</sub> = V <sub>IH</sub> or V <sub>IL</sub>  I <sub>out</sub>   ≤ 2.4 mA  I <sub>out</sub>   ≤ 4.0 mA  I <sub>out</sub>   ≤ 5.2 mA	3.0	2.48	2.34	2.20	
			4.5	3.98	3.84	3.7	
			6.0	5.48	5.34	5.2	
V <sub>OL</sub>	Maximum Low-Level Output Voltage	V <sub>in</sub> = V <sub>IH</sub> or V <sub>IL</sub>  I <sub>out</sub>   ≤ 20 μA	2.0	0.1	0.1	0.1	V
			4.5	0.1	0.1	0.1	
			6.0	0.1	0.1	0.1	
		V <sub>in</sub> = V <sub>IH</sub> or V <sub>IL</sub>  I <sub>out</sub>   ≤ 2.4 mA  I <sub>out</sub>   ≤ 4.0 mA  I <sub>out</sub>   ≤ 5.2 mA	3.0	0.26	0.33	0.4	
			4.5	0.26	0.33	0.4	
			6.0	0.26	0.33	0.4	
I <sub>in</sub>	Maximum Input Leakage Current	V <sub>in</sub> = V <sub>CC</sub> or GND	6.0	±0.1	±1.0	±1.0	μA
I <sub>CC</sub>	Maximum Quiescent Supply Current (per Package)	V <sub>in</sub> = V <sub>CC</sub> or GND  I <sub>out</sub>   = 0 μA	6.0	2.0	20	40	μA

NOTE: Information on typical parametric values can be found in Chapter 2 of the ON Semiconductor High-Speed CMOS Data Book (DL129/D).

## AC ELECTRICAL CHARACTERISTICS (C<sub>L</sub> = 50 pF, Input t<sub>r</sub> = t<sub>f</sub> = 6.0 ns)

Symbol	Parameter	V <sub>CC</sub> (V)	Guaranteed Limit			Unit
			- 55 to 25°C	≤ 85°C	≤ 125°C	
t <sub>PLH</sub> , t <sub>PHL</sub>	Maximum Propagation Delay, Input A or B to Output Y (Figures 1 and 2)	2.0	75	95	110	ns
		3.0	30	40	55	
		4.5	15	19	22	
		6.0	13	16	19	
t <sub>TLH</sub> , t <sub>THL</sub>	Maximum Output Transition Time, Any Output (Figures 1 and 2)	2.0	75	95	110	ns
		3.0	30	40	55	
		4.5	15	19	22	
		6.0	13	16	19	
C <sub>in</sub>	Maximum Input Capacitance	—	10	10	10	pF

NOTE: For propagation delays with loads other than 50 pF, and information on typical parametric values, see Chapter 2 of the ON Semiconductor High-Speed CMOS Data Book (DL129/D).

C <sub>PD</sub>	Power Dissipation Capacitance (Per Gate)*	Typical @ 25°C, V <sub>CC</sub> = 5.0 V	
		22	
		pF	

\* Used to determine the no-load dynamic power consumption: P<sub>D</sub> = C<sub>PD</sub> V<sub>CC</sub><sup>2</sup>f + I<sub>CC</sub> V<sub>CC</sub>. For load considerations, see Chapter 2 of the ON Semiconductor High-Speed CMOS Data Book (DL129/D).



# 74HC02

## ORDERING INFORMATION

Device	Package	Shipping†
74HC02DR2G	SOIC-14 (Pb-Free)	2500/Tape & Reel
74HC02DTR2G	TSSOP-14*	

†For information on tape and reel specifications, including part orientation and tape sizes, please refer to our Tape and Reel Packaging Specifications Brochure, BRD8011/D.

\*This package is inherently Pb-Free.

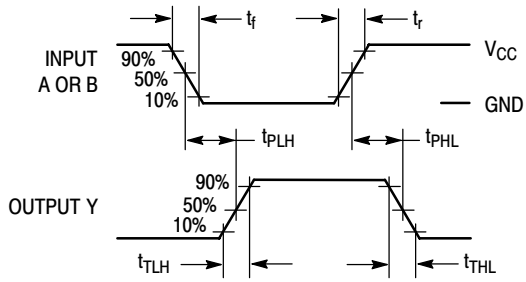
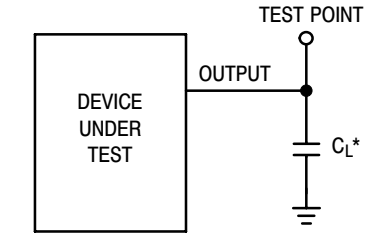


Figure 1. Switching Waveforms



\*Includes all probe and jig capacitance

Figure 2. Test Circuit

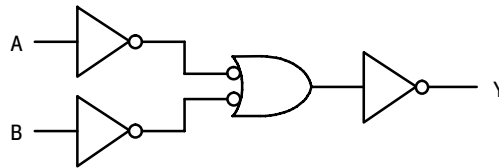
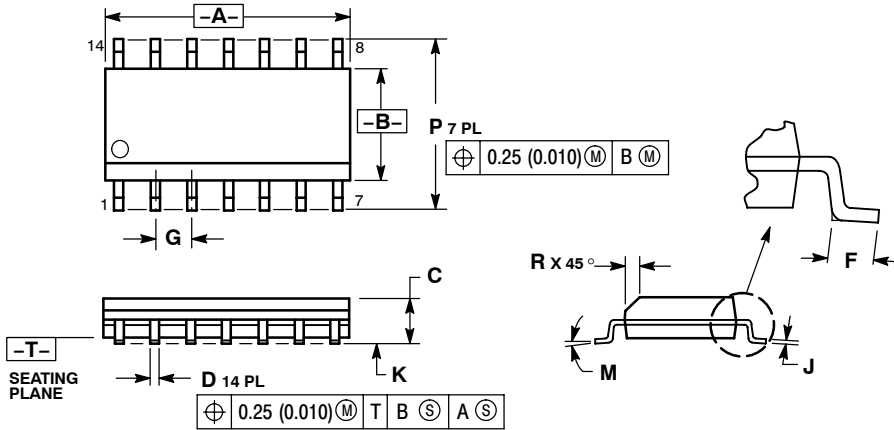


Figure 3. Expanded Logic Diagram  
(1/4 of the Device)

# 74HC02

## PACKAGE DIMENSIONS

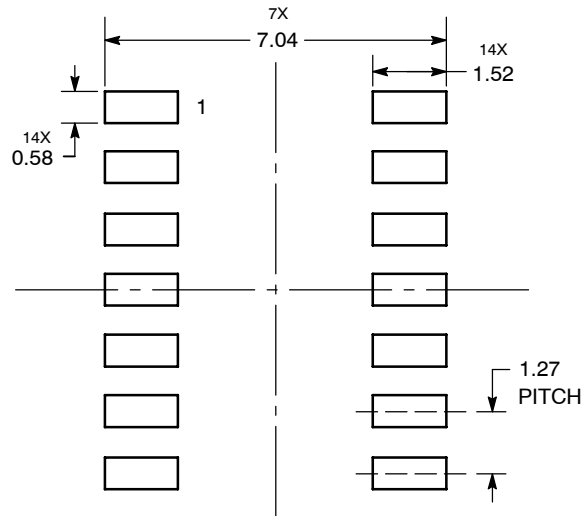
SOIC-14  
CASE 751A-03  
ISSUE H



- NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
  2. CONTROLLING DIMENSION: MILLIMETER.
  3. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION.
  4. MAXIMUM MOLD PROTRUSION 0.15 (0.006) PER SIDE.
  5. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.127 (0.005) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION.

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	8.55	8.75	0.337	0.344
B	3.80	4.00	0.150	0.157
C	1.35	1.75	0.054	0.068
D	0.35	0.49	0.014	0.019
F	0.40	1.25	0.016	0.049
G	1.27 BSC		0.050 BSC	
J	0.19	0.25	0.008	0.009
K	0.10	0.25	0.004	0.009
M	0°	7°	0°	7°
P	5.80	6.20	0.228	0.244
R	0.25	0.50	0.010	0.019

### SOLDERING FOOTPRINT\*



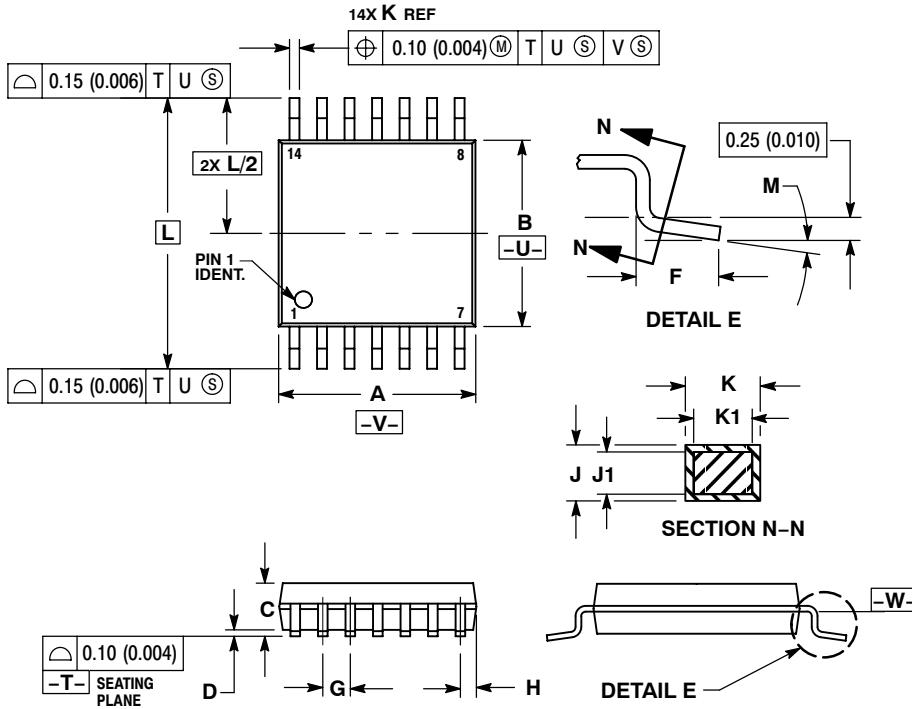
DIMENSIONS: MILLIMETERS

\*For additional information on our Pb-Free strategy and soldering details, please download the ON Semiconductor Soldering and Mounting Techniques Reference Manual, SOLDERRM/D.

# 74HC02

## PACKAGE DIMENSIONS

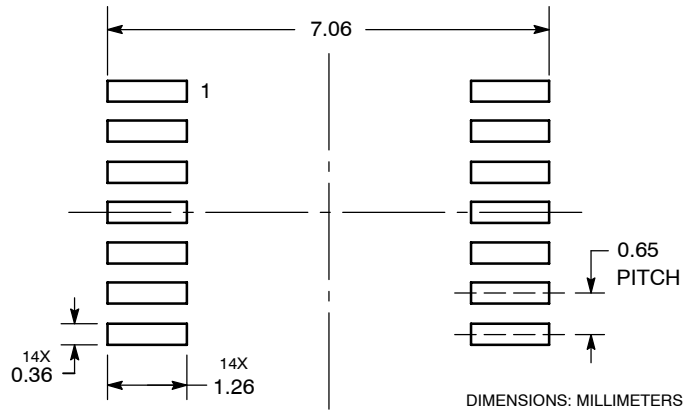
TSSOP-14  
CASE 948G-01  
ISSUE B



- NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
  2. CONTROLLING DIMENSION: MILLIMETER.
  3. DIMENSION A DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS. MOLD FLASH OR GATE BURRS SHALL NOT EXCEED 0.15 (0.006) PER SIDE.
  4. DIMENSION B DOES NOT INCLUDE INTERLEAD FLASH OR PROTRUSION. INTERLEAD FLASH OR PROTRUSION SHALL NOT EXCEED 0.25 (0.010) PER SIDE.
  5. DIMENSION K DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE K DIMENSION AT MAXIMUM MATERIAL CONDITION.
  6. TERMINAL NUMBERS ARE SHOWN FOR REFERENCE ONLY.
  7. DIMENSION A AND B ARE TO BE DETERMINED AT DATUM PLANE -W-.

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	4.90	5.10	0.193	0.200
B	4.30	4.50	0.169	0.177
C	---	1.20	---	0.047
D	0.05	0.15	0.002	0.006
F	0.50	0.75	0.020	0.030
G	0.65 BSC		0.026 BSC	
H	0.50	0.60	0.020	0.024
J	0.09	0.20	0.004	0.008
J1	0.09	0.16	0.004	0.006
K	0.19	0.30	0.007	0.012
K1	0.19	0.25	0.007	0.010
L	6.40 BSC		0.252 BSC	
M	0°	8°	0°	8°

### SOLDERING FOOTPRINT\*



\*For additional information on our Pb-Free strategy and soldering details, please download the ON Semiconductor Soldering and Mounting Techniques Reference Manual, SOLDERRM/D.

**ON Semiconductor** and **ON** are registered trademarks of Semiconductor Components Industries, LLC (SCILLC). SCILLC reserves the right to make changes without further notice to any products herein. SCILLC makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does SCILLC assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. "Typical" parameters which may be provided in SCILLC data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. SCILLC does not convey any license under its patent rights nor the rights of others. SCILLC products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the SCILLC product could create a situation where personal injury or death may occur. Should Buyer purchase or use SCILLC products for any such unintended or unauthorized application, Buyer shall indemnify and hold SCILLC and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that SCILLC was negligent regarding the design or manufacture of the part. SCILLC is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

## PUBLICATION ORDERING INFORMATION

### LITERATURE FULFILLMENT:

Literature Distribution Center for ON Semiconductor  
P.O. Box 5163, Denver, Colorado 80217 USA  
**Phone:** 303-675-2175 or 800-344-3860 Toll Free USA/Canada  
**Fax:** 303-675-2176 or 800-344-3867 Toll Free USA/Canada  
**Email:** [orderlit@onsemi.com](mailto:orderlit@onsemi.com)

**N. American Technical Support:** 800-282-9855 Toll Free  
USA/Canada  
**Europe, Middle East and Africa Technical Support:**  
Phone: 421 33 790 2910  
**Japan Customer Focus Center**  
Phone: 81-3-5773-3850

**ON Semiconductor Website:** [www.onsemi.com](http://www.onsemi.com)  
**Order Literature:** <http://www.onsemi.com/orderlit>  
For additional information, please contact your local  
Sales Representative

# DATA SHEET

For a complete data sheet, please also download:

- The IC06 74HC/HCT/HCU/HCMOS Logic Family Specifications
- The IC06 74HC/HCT/HCU/HCMOS Logic Package Information
- The IC06 74HC/HCT/HCU/HCMOS Logic Package Outlines

## **74HC/HCT163**

**Presettable synchronous 4-bit  
binary counter; synchronous reset**

Product specification  
File under Integrated Circuits, IC06

December 1990

# Presettable synchronous 4-bit binary counter; synchronous reset

## 74HC/HCT163

### FEATURES

- Synchronous counting and loading
- Two count enable inputs for n-bit cascading
- Positive-edge triggered clock
- Synchronous reset
- Output capability: standard
- I<sub>CC</sub> category: MSI

### GENERAL DESCRIPTION

The 74HC/HCT163 are high-speed Si-gate CMOS devices and are pin compatible with low power Schottky TTL (LSTTL). They are specified in compliance with JEDEC standard no. 7A.

The 74HC/HCT163 are synchronous presettable binary counters which feature an internal look-ahead carry and can be used for high-speed counting. Synchronous operation is provided by having all flip-flops clocked simultaneously on the positive-going edge of the clock (CP).

The outputs (Q<sub>0</sub> to Q<sub>3</sub>) of the counters may be preset to a HIGH or LOW level. A LOW level at the parallel enable input ( $\overline{PE}$ ) disables the counting action and causes the data at the data inputs (D<sub>0</sub> to D<sub>3</sub>) to be loaded into the counter on the positive-going edge of the clock (providing that the set-up and hold time requirements for  $\overline{PE}$  are met).

Preset takes place regardless of the levels at count enable inputs (CEP and CET).

For the "163" the clear function is synchronous.

A LOW level at the master reset input ( $\overline{MR}$ ) sets all four outputs of the flip-flops (Q<sub>0</sub> to Q<sub>3</sub>) to LOW level after the next positive-going transition on the clock (CP) input (provided that the set-up and hold time requirements for  $\overline{MR}$  are met). This action occurs regardless of the levels at  $\overline{PE}$ , CET and CEP inputs.

This synchronous reset feature enables the designer to modify the maximum count with only one external NAND gate.

The look-ahead carry simplifies serial cascading of the counters. Both count enable inputs (CEP and CET) must be HIGH to count. The CET input is fed forward to enable the terminal count output (TC). The TC output thus enabled will produce a HIGH output pulse of a duration approximately equal to a HIGH level output of Q<sub>0</sub>. This pulse can be used to enable the next cascaded stage.

The maximum clock frequency for the cascaded counters is determined by the CP to TC propagation delay and CEP to CP set-up time, according to the following formula:

$$f_{\max} = \frac{1}{t_{P(\max)}(\text{CP to TC}) + t_{\text{SU}}(\text{CEP to CP})}$$

### QUICK REFERENCE DATA

GND = 0 V; T<sub>amb</sub> = 25 °C; t<sub>r</sub> = t<sub>f</sub> = 6 ns

SYMBOL	PARAMETER	CONDITIONS	TYPICAL		UNIT
			HC	HCT	
t <sub>PHL</sub> / t <sub>PLH</sub>	propagation delay CP to Q <sub>n</sub> CP to TC CET to TC	C <sub>L</sub> = 15 pF; V <sub>CC</sub> = 5 V	17	20	ns
			21	25	ns
			11	14	ns
f <sub>max</sub>	maximum clock frequency		51	50	MHz
C <sub>I</sub>	input capacitance		3.5	3.5	pF
C <sub>PD</sub>	power dissipation capacitance per package	notes 1 and 2	33	35	pF

### Notes

1. C<sub>PD</sub> is used to determine the dynamic power dissipation (P<sub>D</sub> in μW):

$$P_D = C_{PD} \times V_{CC}^2 \times f_i + \sum (C_L \times V_{CC}^2 \times f_o)$$

where:

f<sub>i</sub> = input frequency in MHz  
 f<sub>o</sub> = output frequency in MHz  
 ∑ (C<sub>L</sub> × V<sub>CC</sub><sup>2</sup> × f<sub>o</sub>) = sum of outputs

C<sub>L</sub> = output load capacitance in pF

V<sub>CC</sub> = supply voltage in V

2. For HC the condition is  
 V<sub>I</sub> = GND to V<sub>CC</sub>  
 For HCT the condition is  
 V<sub>I</sub> = GND to V<sub>CC</sub> – 1.5 V

Pre-settable synchronous 4-bit binary counter; synchronous reset

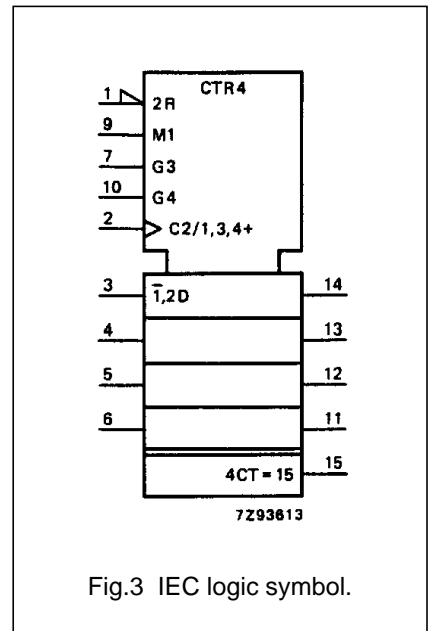
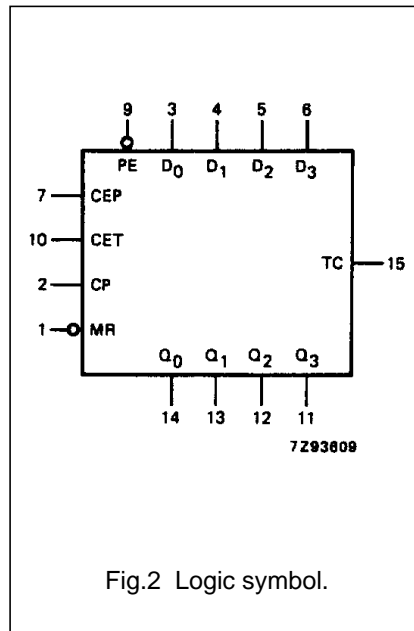
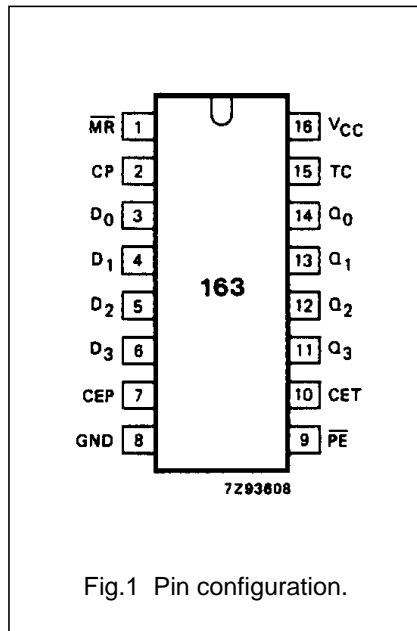
74HC/HCT163

ORDERING INFORMATION

See "74HC/HCT/HCU/HCMOS Logic Package Information".

PIN DESCRIPTION

PIN NO.	SYMBOL	NAME AND FUNCTION
1	$\overline{MR}$	synchronous master reset (active LOW)
2	CP	clock input (LOW-to-HIGH, edge-triggered)
3, 4, 5, 6	D <sub>0</sub> to D <sub>3</sub>	data inputs
7	CEP	count enable input
8	GND	ground (0 V)
9	$\overline{PE}$	parallel enable input (active LOW)
10	CET	count enable carry input
14, 13, 12, 11	Q <sub>0</sub> to Q <sub>3</sub>	flip-flop outputs
15	TC	terminal count output
16	V <sub>CC</sub>	positive supply voltage



Pre-settable synchronous 4-bit binary counter; synchronous reset

74HC/HCT163

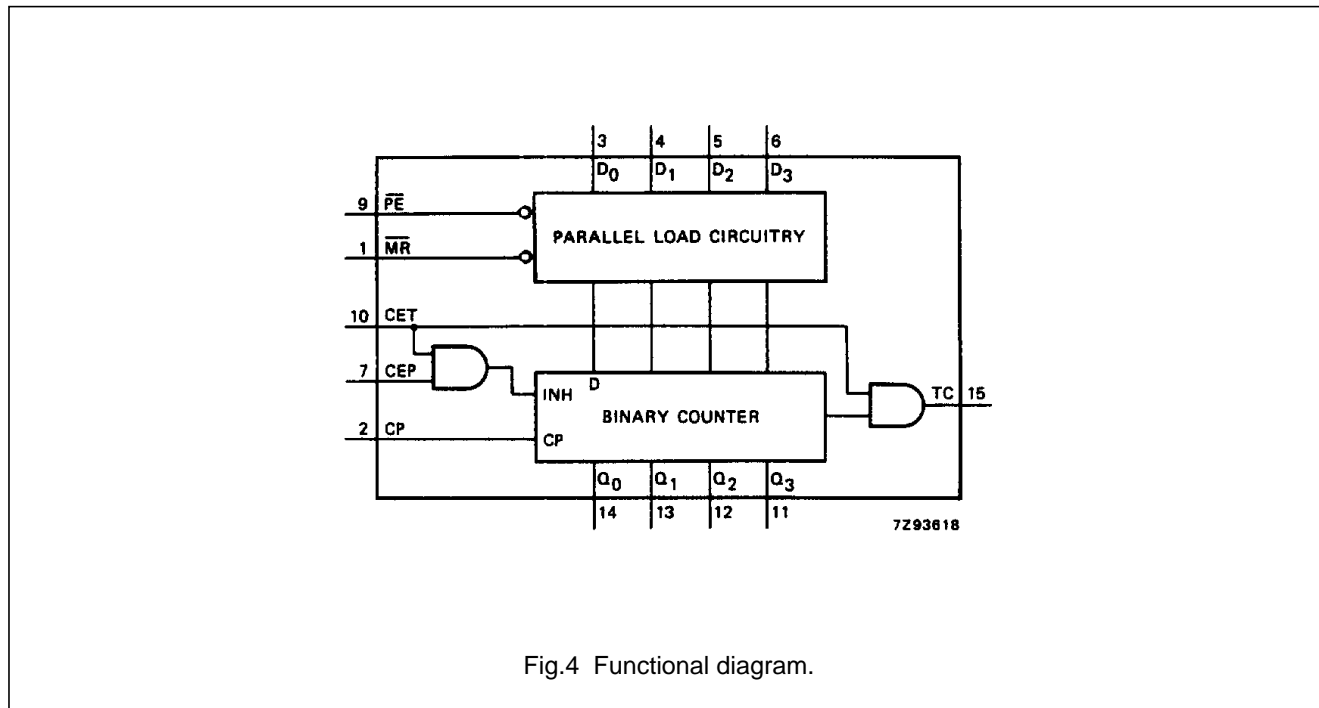


Fig.4 Functional diagram.

FUNCTION TABLE

OPERATING MODE	INPUTS						OUTPUTS	
	$\overline{MR}$	CP	CEP	CET	$\overline{PE}$	$D_n$	$Q_n$	TC
reset (clear)	l	$\uparrow$	X	X	X	X	L	L
parallel load	h	$\uparrow$	X	X	l	l	L	L
	h	$\uparrow$	X	X	l	h	H	(1)
count	h	$\uparrow$	h	h	h	X	count	(1)
hold (do nothing)	h	X	l	X	h	X	$q_n$	(1)
	h	X	X	l	h	X	$q_n$	L

Notes

- The TC output is HIGH when CET is HIGH and the counter is at terminal count (HHHH).  
 H = HIGH voltage level  
 h = HIGH voltage level one set-up time prior to the LOW-to-HIGH CP transition  
 L = LOW voltage level  
 l = LOW voltage level one set-up time prior to the LOW-to-HIGH CP transition  
 q = lower case letters indicate the state of the referenced output one set-up time prior to the LOW-to-HIGH CP transition  
 X = don't care  
 $\uparrow$  = LOW-to-HIGH CP transition



Pre-settable synchronous 4-bit binary counter; synchronous reset

74HC/HCT163

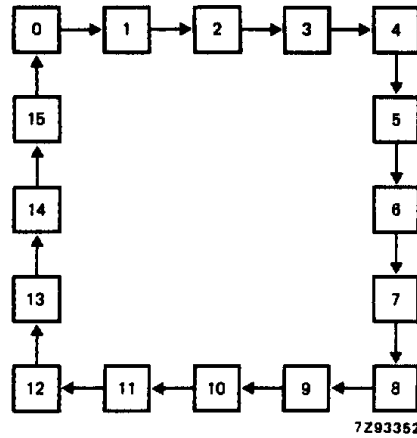


Fig.5 State diagram.

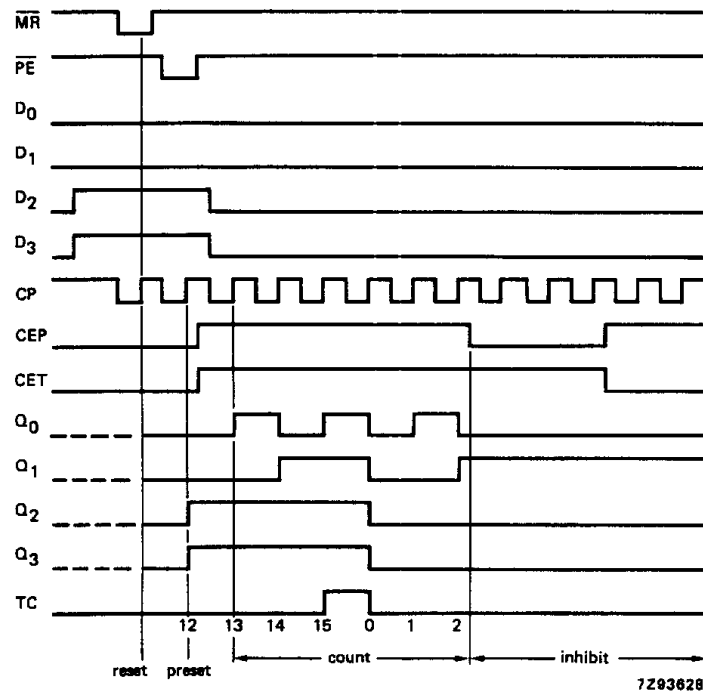


Fig.6 Typical timing sequence: reset outputs to zero; preset to binary twelve; count to thirteen, fourteen, fifteen, zero, one and two; inhibit.

# Pre-settable synchronous 4-bit binary counter; synchronous reset

74HC/HCT163

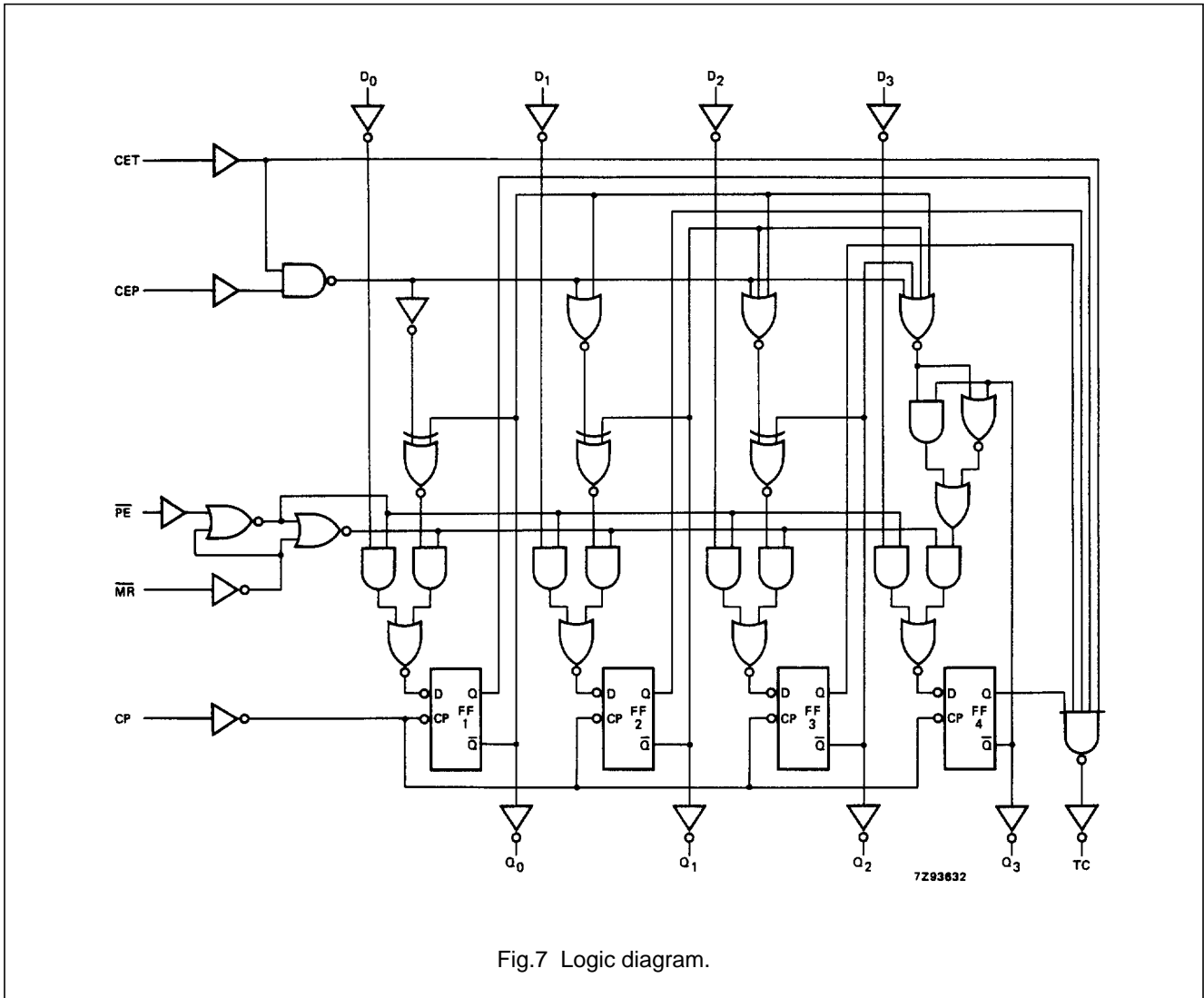


Fig.7 Logic diagram.

# Presettable synchronous 4-bit binary counter; synchronous reset

74HC/HCT163

## DC CHARACTERISTICS FOR 74HC

For the DC characteristics see *"74HC/HCT/HCU/HCMOS Logic Family Specifications"*.

Output capability: standard

I<sub>CC</sub> category: MSI

## AC CHARACTERISTICS FOR 74HC

GND = 0 V; t<sub>r</sub> = t<sub>f</sub> = 6 ns; C<sub>L</sub> = 50 pF

SYMBOL	PARAMETER	T <sub>amb</sub> (°C)						UNIT	TEST CONDITIONS		
		74HC							V <sub>CC</sub> (V)	WAVEFORMS	
		+25			-40 to +85		-40 to +125				
		min.	typ.	max.	min.	max.	min.				max.
t <sub>PHL</sub> / t <sub>PLH</sub>	propagation delay CP to Q <sub>n</sub>		55 20 16	185 37 31		230 46 39		280 56 48	ns	2.0 4.5 6.0	Fig.8
t <sub>PHL</sub> / t <sub>PLH</sub>	propagation delay CP to TC		69 25 20	215 43 37		270 54 46		320 65 55	ns	2.0 4.5 6.0	Fig.8
t <sub>PHL</sub> / t <sub>PLH</sub>	propagation delay CET to TC		36 13 10	120 24 20		150 30 26		180 36 31	ns	2.0 4.5 6.0	Fig.9
t <sub>THL</sub> / t <sub>TLH</sub>	output transition time		19 7 6	75 15 13		95 19 16		110 22 19	ns	2.0 4.5 6.0	Figs 8 and 9
t <sub>W</sub>	clock pulse width HIGH or LOW	80 16 14	17 6 5		100 20 17		120 24 20		ns	2.0 4.5 6.0	Fig.8
t <sub>SU</sub>	set-up time $\overline{\text{MR}}$ , D <sub>n</sub> to CP	80 16 14	17 6 5		100 20 17		120 24 20		ns	2.0 4.5 6.0	Figs 10 and 11
t <sub>SU</sub>	set-up time $\overline{\text{PE}}$ to CP	80 16 14	22 8 6		100 20 17		120 24 20		ns	2.0 4.5 6.0	Fig.10
t <sub>SU</sub>	set-up time CEP, CET to CP	175 35 30	58 21 17		220 44 37		265 53 45		ns	2.0 4.5 6.0	Fig.12
t <sub>H</sub>	hold time D <sub>n</sub> , $\overline{\text{PE}}$ , CEP, CET, $\overline{\text{MR}}$ to CP	0 0 0	-14 -5 -4		0 0 0		0 0 0		ns	2.0 4.5 6.0	Figs 10, 11 and 12
f <sub>max</sub>	maximum clock pulse frequency	5 27 32	15 46 55		4 22 26		4 18 21		MHz	2.0 4.5 6.0	Fig.8

---

Presettable synchronous 4-bit binary  
counter; synchronous reset

---

74HC/HCT163

**DC CHARACTERISTICS FOR 74HCT**

For the DC characteristics see *"74HC/HCT/HCU/HCMOS Logic Family Specifications"*.

Output capability: standard

I<sub>CC</sub> category: MSI

**Note to HCT types**

The value of additional quiescent supply current ( $\Delta I_{CC}$ ) for a unit load of 1 is given in the family specifications. To determine  $\Delta I_{CC}$  per input, multiply this value by the unit load coefficient shown in the table below.

INPUT	UNIT LOAD COEFFICIENT
$\overline{MR}$	0.95
CP	1.10
CEP	0.25
$D_n$	0.25
CET	0.75
$\overline{PE}$	0.30

# Presettable synchronous 4-bit binary counter; synchronous reset

74HC/HCT163

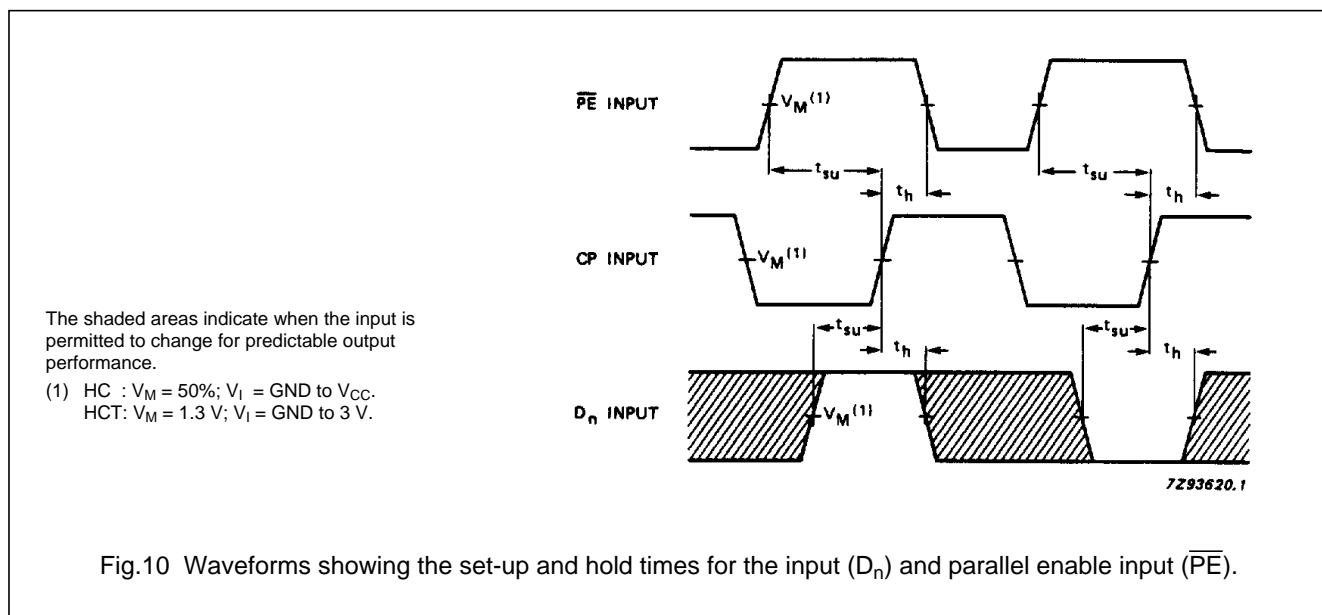
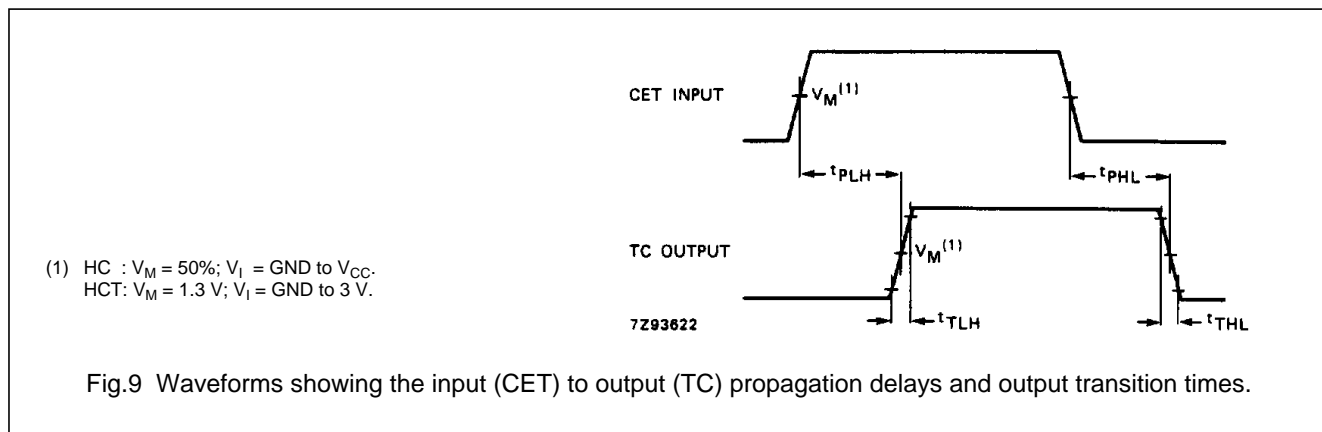
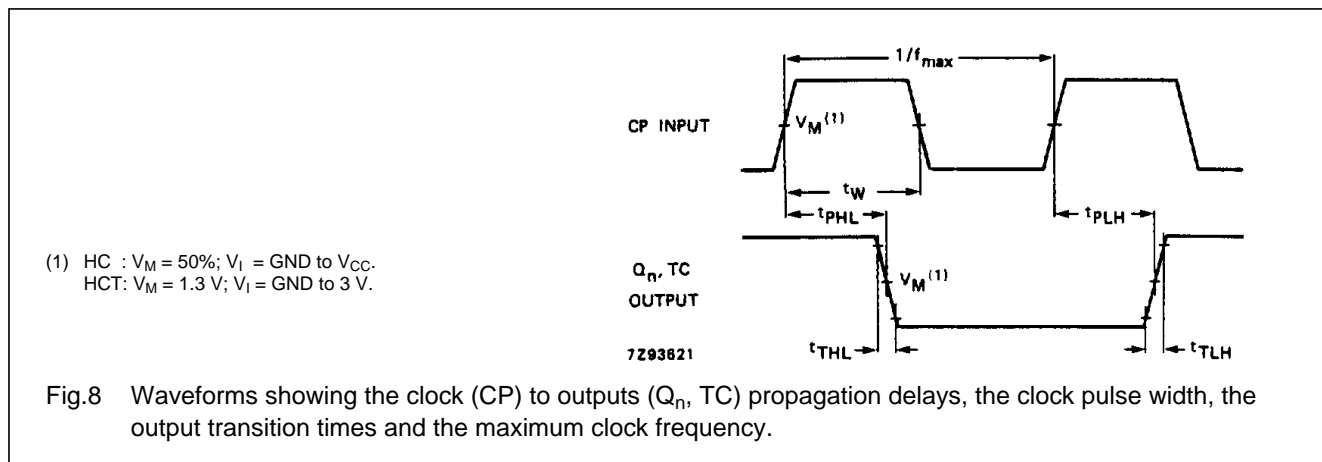
**AC CHARACTERISTICS FOR 74HCT**GND = 0 V;  $t_r = t_f = 6$  ns;  $C_L = 50$  pF

SYMBOL	PARAMETER	T <sub>amb</sub> (°C)						UNIT	TEST CONDITIONS		
		74HCT							V <sub>CC</sub> (V)	WAVEFORMS	
		+25			-40 to +85		-40 to +125				
		min.	typ.	max.	min.	max.	min.				max.
t <sub>PHL</sub> / t <sub>PLH</sub>	propagation delay CP to Q <sub>n</sub>		23	39		49		59	ns	4.5	Fig.8
t <sub>PHL</sub> / t <sub>PLH</sub>	propagation delay CP to TC		29	49		61		74	ns	4.5	Fig.8
t <sub>PHL</sub> / t <sub>PLH</sub>	propagation delay CET to TC		17	32		44		48	ns	4.5	Fig.9
t <sub>THL</sub> / t <sub>TLH</sub>	output transition time		7	15		19		22	ns	4.5	Figs 8 and 9
t <sub>W</sub>	clock pulse width HIGH or LOW	20	6		25		30		ns	4.5	Fig.8
t <sub>su</sub>	set-up time MR, D <sub>n</sub> to CP	20	9		25		30		ns	4.5	Figs 10 and 11
t <sub>su</sub>	set-up time PE to CP	20	11		25		30		ns	4.5	Fig.10
t <sub>su</sub>	set-up time CEP, CET to CP	40	24		50		60		ns	4.5	Fig.12
t <sub>h</sub>	hold time D <sub>n</sub> , PE, CEP, CET, MR to CP	0	-5		0		0		ns	4.5	Figs 10, 11 and 12
f <sub>max</sub>	maximum clock pulse frequency	26	45		21		17		MHz	4.5	Fig.8

Pre-settable synchronous 4-bit binary counter; synchronous reset

74HC/HCT163

AC WAVEFORMS



Pre-settable synchronous 4-bit binary counter; synchronous reset

74HC/HCT163

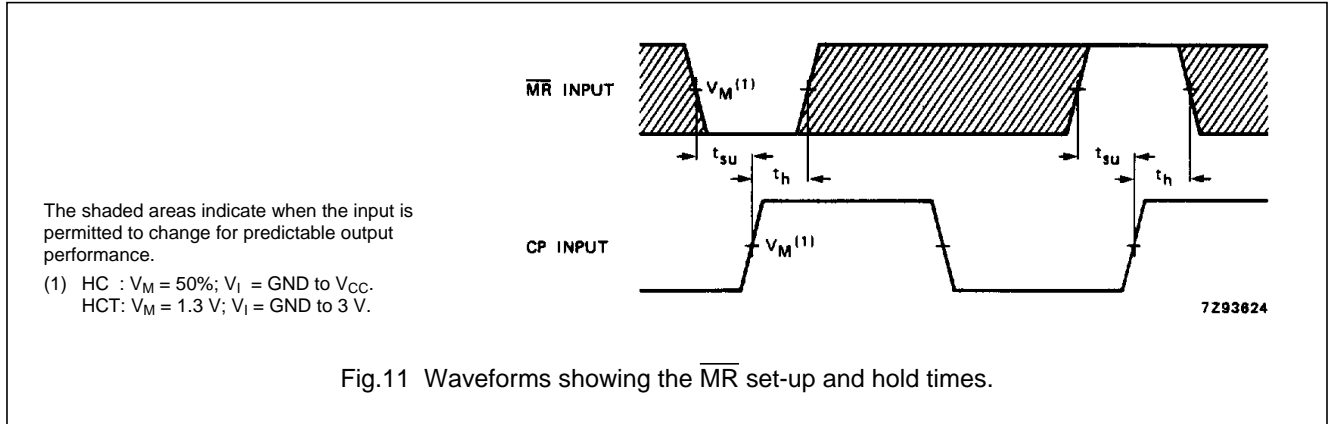


Fig.11 Waveforms showing the  $\overline{\text{MR}}$  set-up and hold times.

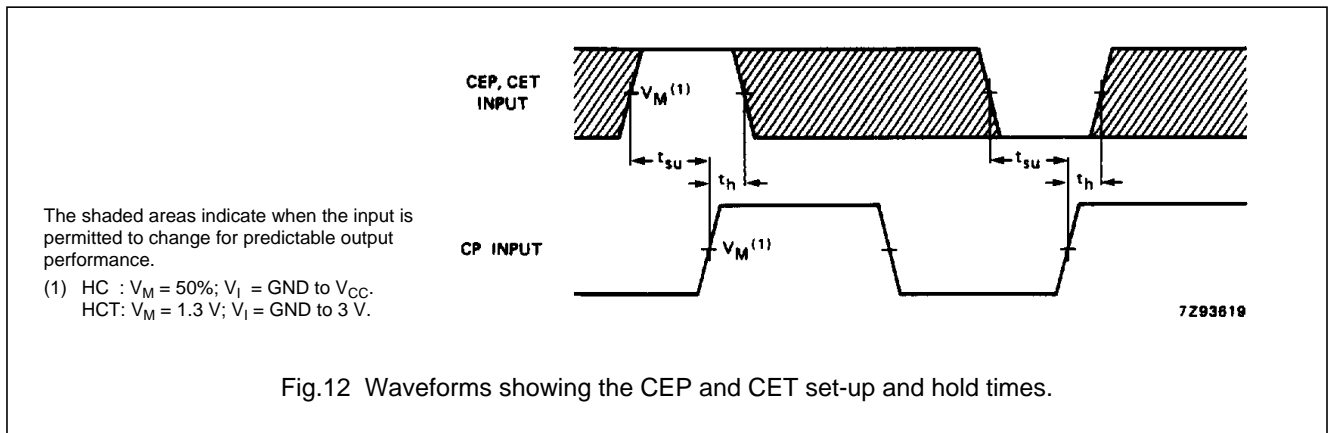


Fig.12 Waveforms showing the CEP and CET set-up and hold times.

APPLICATION INFORMATION

The HC/HCT163 facilitate designing counters of any modulus with minimal external logic. The output is glitch-free due to the synchronous reset.

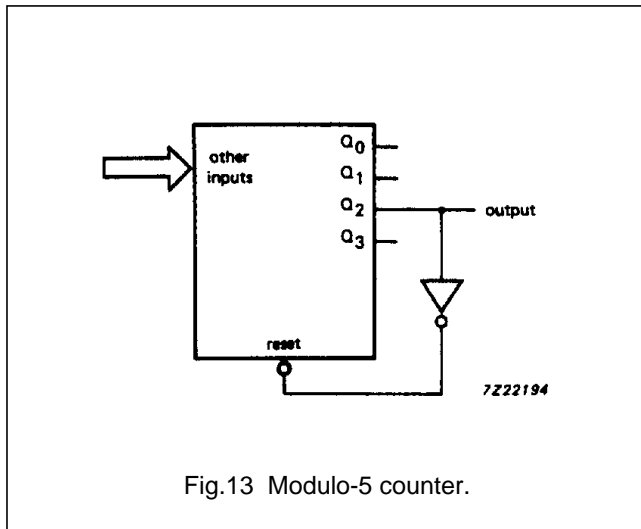


Fig.13 Modulo-5 counter.

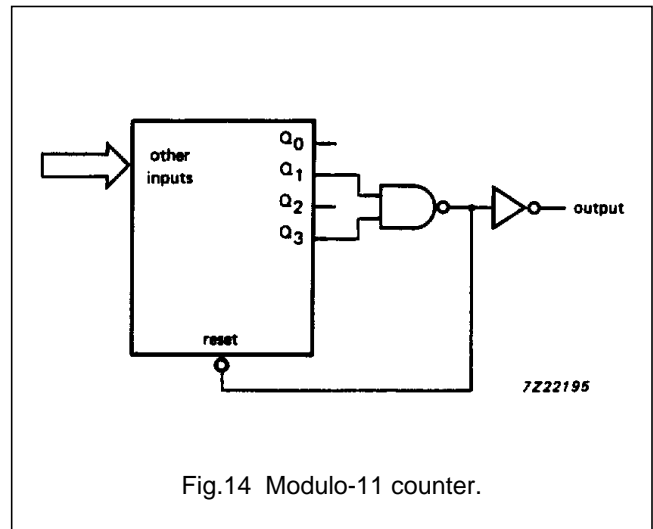


Fig.14 Modulo-11 counter.

PACKAGE OUTLINES

See "74HC/HCT/HCU/HCMOS Logic Package Outlines".

# **TUTORAIL VIVADO**





# Vivado Design Flow

## Introduction

This lab guides you through the process of using Vivado IDE to create a simple HDL design targeting the Nexys4 or the Basys3 board. You will simulate, synthesize, and implement the design with default settings. Finally, you will generate the bitstream and download it in to the hardware to verify the design functionality

## Objectives

After completing this lab, you will be able to:

- Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device located on the Nexys4 or the Basys3 board
- Use the provided Xilinx Design Constraint (XDC) file to constrain the pin locations
- Simulate the design using the Vivado simulator
- Synthesize and implement the design
- Generate the bitstream
- Configure the FPGA using the generated bitstream and verify the functionality

## Procedure

This lab is broken into steps that consist of general overview statements providing information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab1.

## Design Description

The design consists of some inputs directly connected to the corresponding output LEDs. Other inputs are logically operated on before the results are output on the remaining LEDs as shown in **Figure 1**.

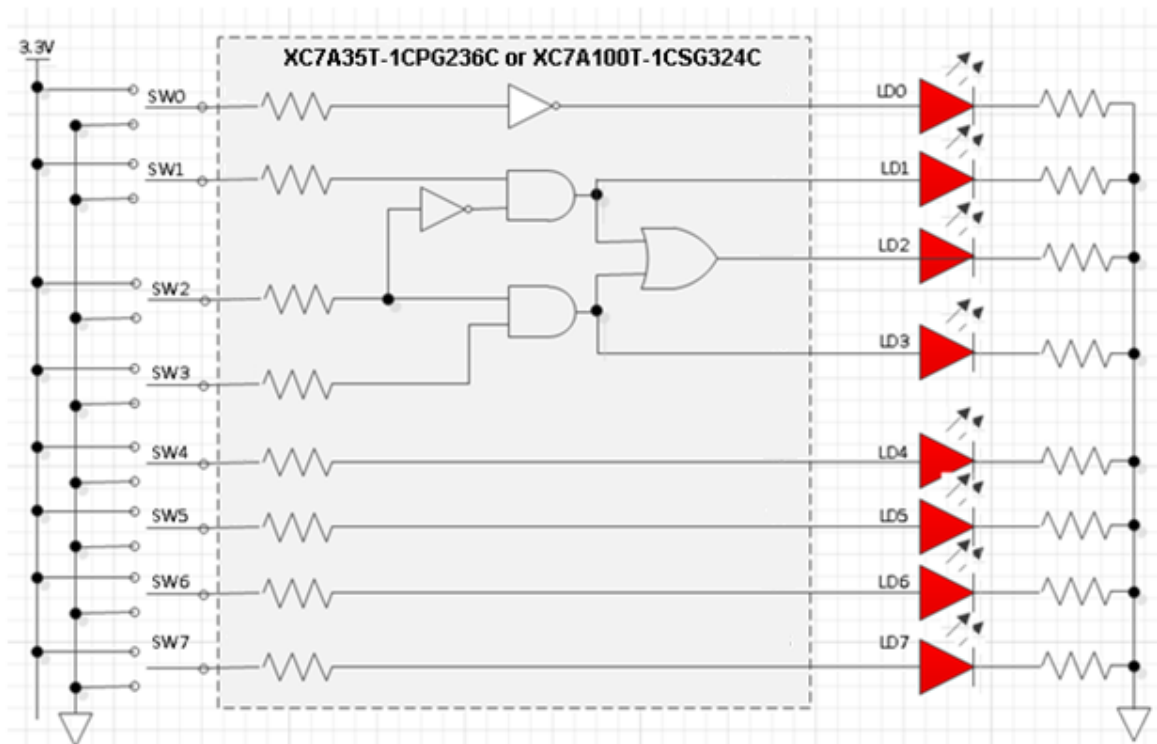
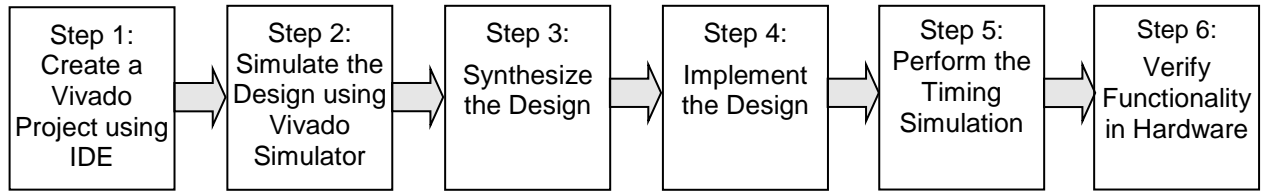


Figure 1. The Completed Design

## General Flow



## Create a Vivado Project using IDE

### Step 1

- 1-1. Launch Vivado and create a project targeting ~~the XC7A100TCSG324-1 device (Nexys4) or the XC7A35TCPG236-1 (Basys3)~~ and using the VHDL HDL. Use the provided lab1.vhd and lab1.xdc files ~~from the 2014\_4\_artix7\_sources\lab1 directory.~~
- 1-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.4 > Vivado 2014.4**
- 1-1-2. Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.
- 1-1-3. Click the Browse button of the *Project location* field of the **New Project** form, browse to **c:\Digilent\Basys3Workshop\2014\_4\_artix7\_labs**, and click **Select**.
- 1-1-4. Enter **lab1** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

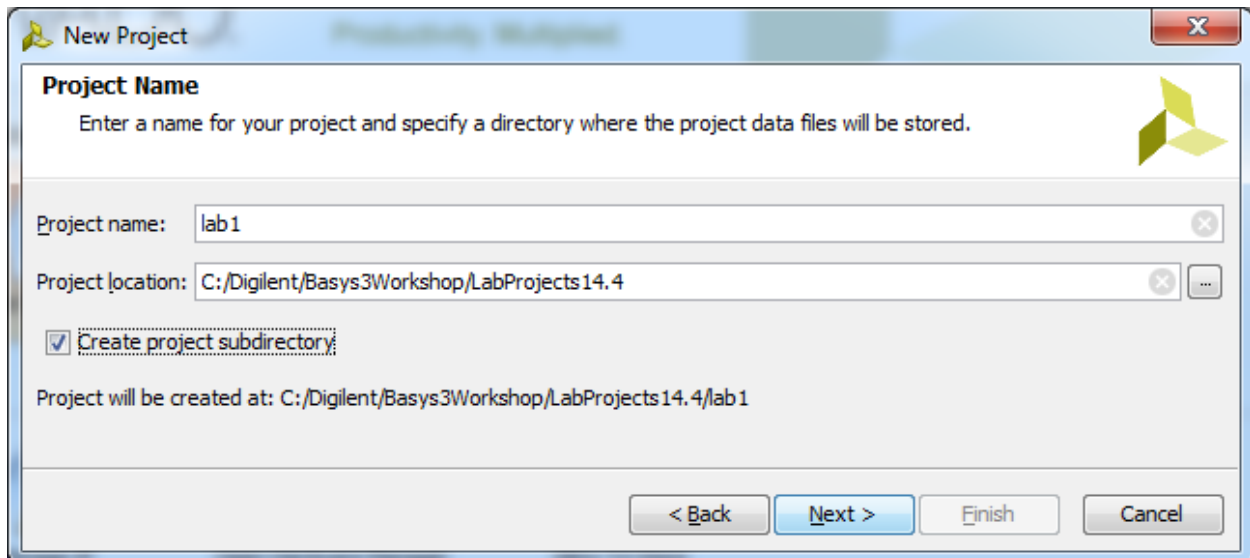
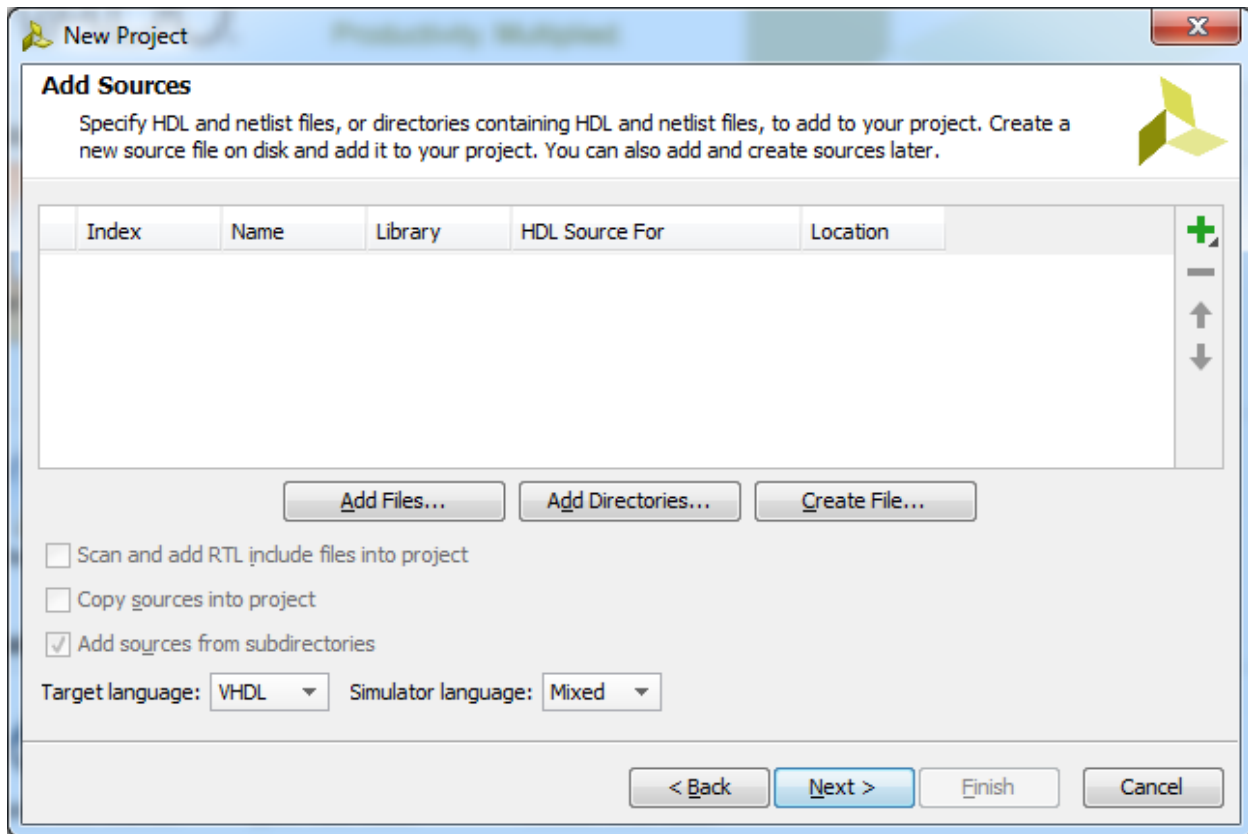


Figure 2. Project Name and Location entry

- 1-1-5. Select **RTL Project** option in the *Project Type* form. Make sure that the *Do not specify sources at this time* box is unchecked. Click **Next**.
- 1-1-6. Using the drop-down buttons, select **VHDL** as the *Target Language* and **Mixed Simulator Language** in the *Add Sources* form.



**Figure 3. Selecting Target and Simulator language**

**1-1-7.** Click on the **Add Files...** button, browse to the `c:\Diligent\Basys3Workshop\2014_4_artix7_sources\lab1` directory, select `lab1.vhd`, click **OK**. Make sure that the *Copy sources into project* box is checked. Click **Next** to get to the *Add Existing IP* form.

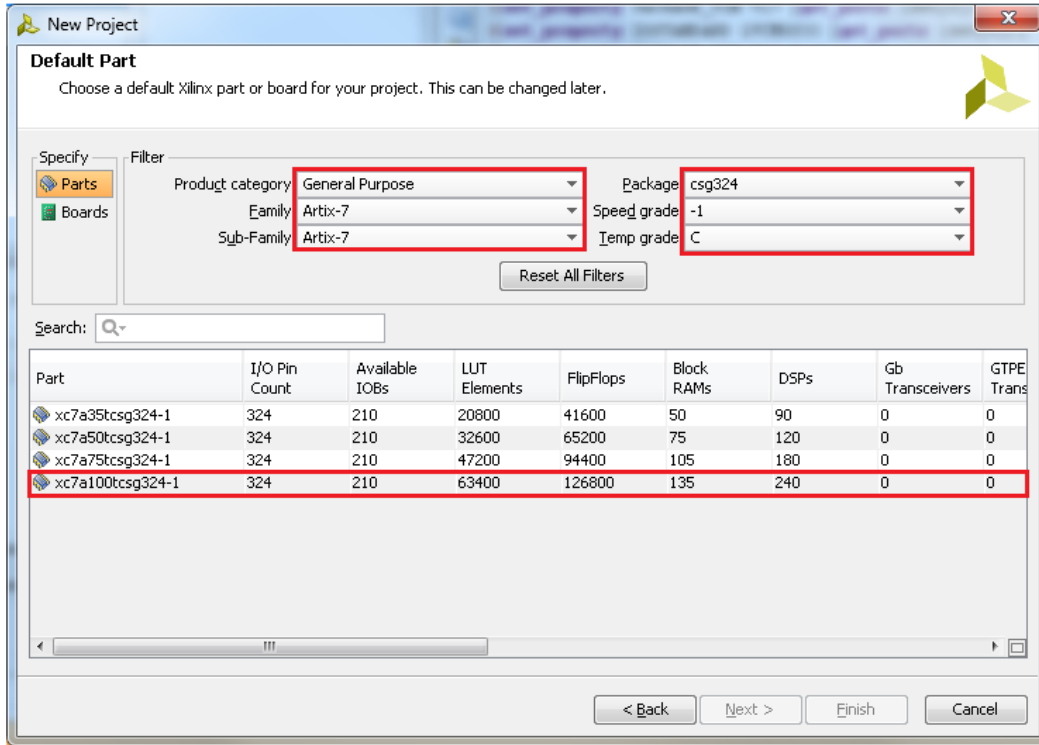
**1-1-8.** Since we do not have any IP to add, click **Next** to get to the *Add Constraints* form.

**1-1-9.** The constraint file `lab1_basys3.xdc` and `lab1_nexys4.xdc` are automatically added. Highlight and delete the XDC file not for the target board by clicking on the "X" on the right hand side of the window. Make sure that the *Copy constraints files into project* box is checked.

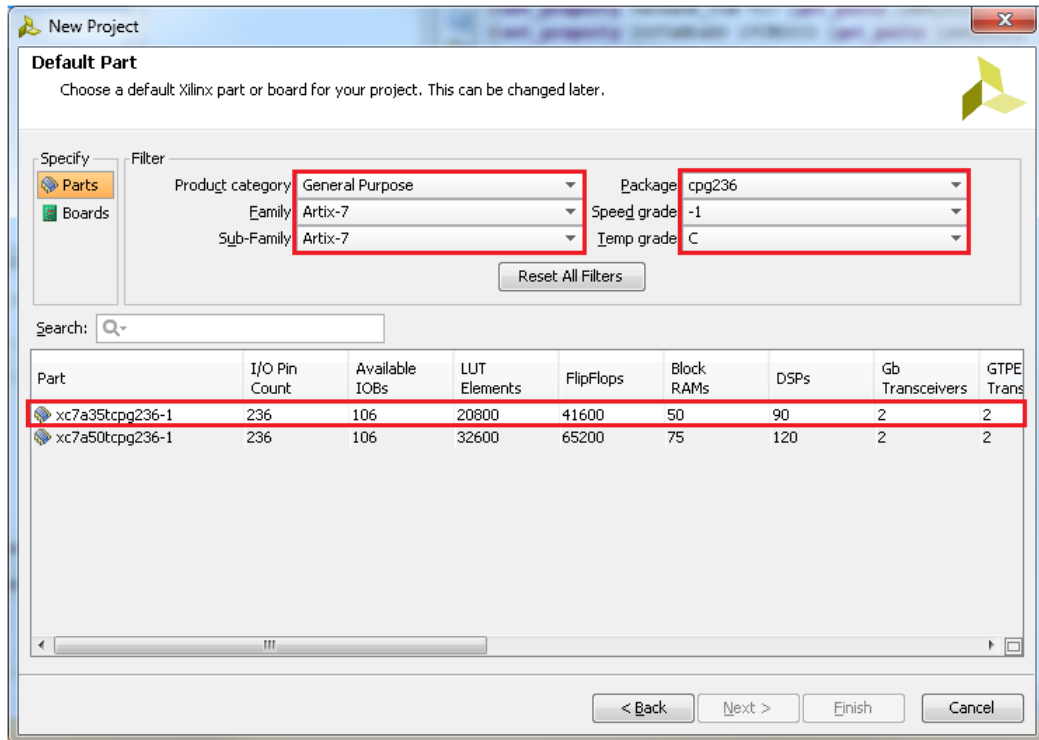
If no files are added, click on the **Add Files...** button, browse to the `C:\Diligent\Basys3Workshop\2014_4_artix7_sources\lab1` directory (if necessary), select `lab1_basys3.xdc` or `lab1_nexys4.xdc` and click **OK** (if necessary), and then click **Next**.

This Xilinx Design Constraints file assigns the physical IO locations on FPGA to the switches and LEDs located on the board. This information can be obtained either through the board's schematic or the board's user guide.

**1-1-10.** In the *Default Part* form, using the **Parts** option and various drop-down fields of the **Filter** section. ~~If using the Nexys4 board, select the **XC7A100TCSG324-1** part.~~ If using the Basys3 board, select the **XC7A35TCPG236-1**. Click **Next**.



**Figure 4. Part Selection for the Nexys4**

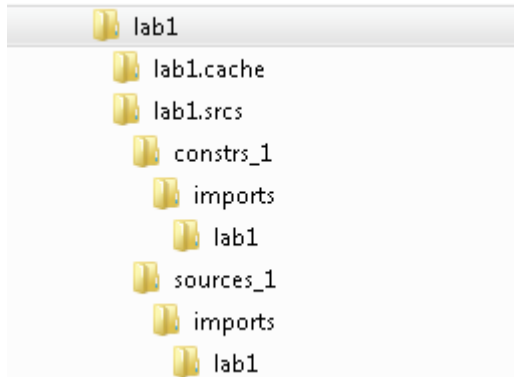


**Figure 4. Part Selection for the Basys3**

You can select the Boards Specify option, select Artix-7 under the Library filter and select the appropriate board. Notice that Nexys4 and the Basys3 are not listed as they are not in the tools database.

**1-1-11.** Click **Finish** to create the Vivado project.

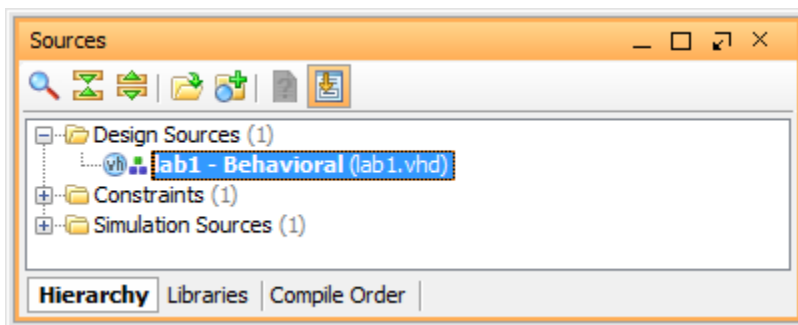
Use the Windows Explorer and look at the C:\Digilent\Basys3Workshop\2014\_4\_artix7\_labs\lab1 directory. You will find that the lab1.cache and lab1.srcs directories and the lab1.xpr (Vivado) project file have been created. The lab1.cache directory is a place holder for the Vivado program database. Two directories, constrs\_1 and sources\_1, are created under the lab1.srcs directory; deep down under them, the copied lab1.xdc (constraint) and lab1.v (source) files respectively are placed.



**Figure 5. Generated directory structure**

**1-2. Open the lab1.vhd source and analyze the content.**

**1-2-1.** In the *Sources* pane, double-click the **lab1.vhd** entry to open the file in text mode.



**Figure 6. Opening the source file**

**1-2-2.** In the VHDL code, lines 1-3 are comment lines describing the entity name and the purpose of the module. Lines 5-6 invoke the standard IEEE library and STD\_LOGIC\_1164 package.

**1-2-3.** Lines 8-11 declare the entity called lab1 and define the port interface signals. Two std\_logic vectors are used, 8 signals each: inputs swt (switches) and outputs led.

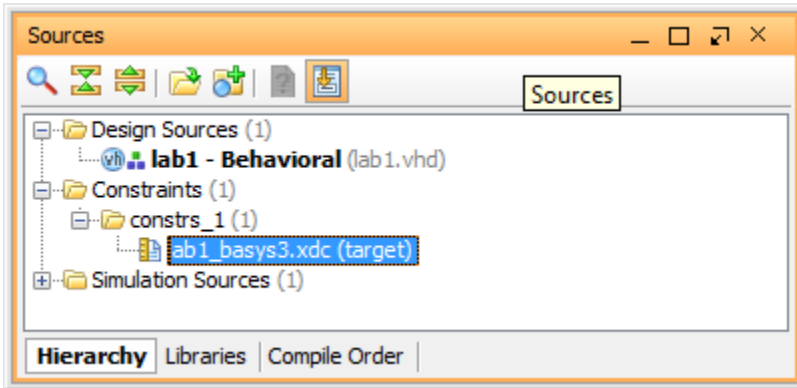
**1-2-4.** Line 13 declares the architecture called Behavioral of the lab1 entity. Line 15 declares an internal signal to build the LED state.

**1-2-5.** Lines 19-24 describe the functionality, while line 26 assigns computed values to the output signals led.

**1-2-6.** Line 28 closes the architecture.

**1-3. Open the lab1\_basys3.xdc or lab1\_nexys4.xdc source and analyze the content.**

1-3-1. In the *Sources* pane, expand the *Constraints* folder and double-click the **lab1.xdc** entry to open the file in text mode.



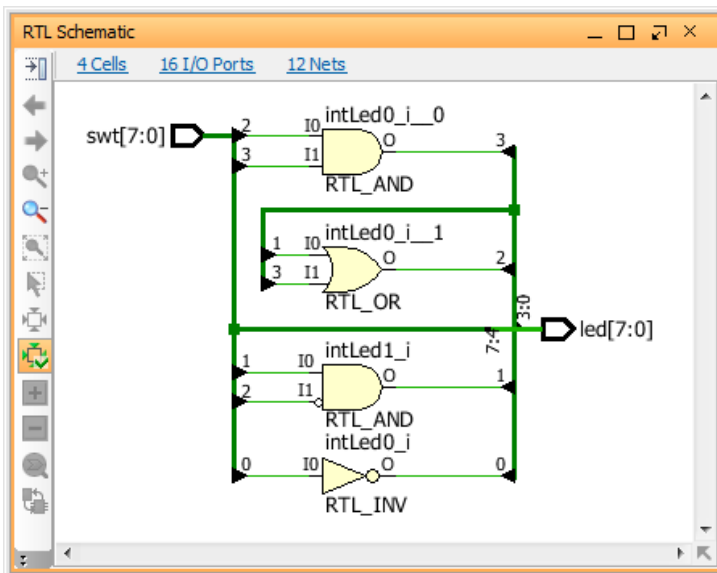
**Figure 7. Opening the constraint file**

1-3-2. Lines 5-20 define the pin locations of the input switches [7:0] and lines 25-40 define the pin locations of the output LEDs [7:0].

**1-4. Perform RTL(register-transfer level ) analysis on the source file.**

1-4-1. Expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane and click on **Schematic**.

The model (design) will be elaborated and a logic view of the design is displayed.



**Figure 8. A logic view of the design**

Notice that some of the switch inputs go through gates before being output to LEDs and the rest go straight through to LEDs as modeled in the file.

## Simulate the Design using the Vivado Simulator

## Step 2

### 2-1. Add the lab1\_tb.vhd testbench file.

2-1-1. Click **Add Sources** under the *Project Manager* tasks of the *Flow Navigator* pane.

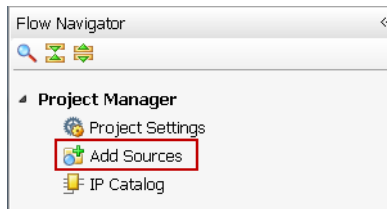


Figure 9. Add Sources

2-1-2. Select the *Add or Create Simulation Sources* option and click **Next**.

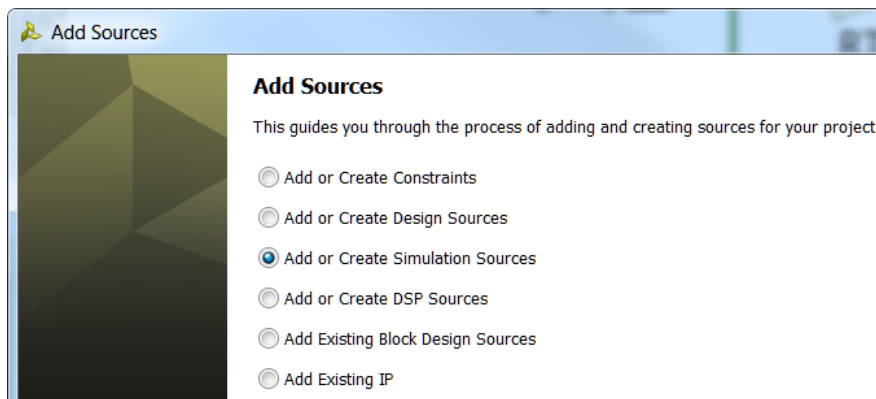


Figure 10. Selecting Simulation Sources option

2-1-3. In the *Add Sources Files* form, click the **Add Files...** button. Browse to the `c:\Digilent/Basys3Workshop/2014_4_artix7_sources/lab1` folder and select `lab1_tb.vhd` and click **OK**.

2-1-4. Make sure that the *Copy sources into project* box is checked. Click **Finish**. Select the *Sources* tab and expand the *Simulation Sources* group. The `lab1_tb.vhd` file is added under the *Simulation Sources* group, and `lab1.vhd` is automatically placed in its hierarchy as a dut (device under test) instance.

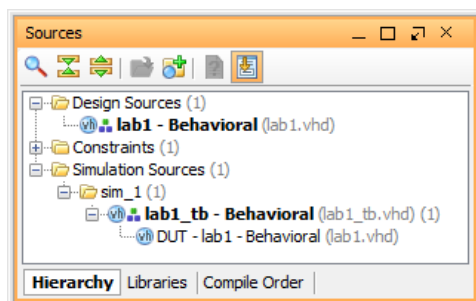


Figure 11. Simulation Sources hierarchy

2-1-5. Using the Windows Explorer, verify that the `sim_1` directory is created at the same level as `constrs_1` and `sources_1` directories under the `lab1.srcs` directory, and that a copy of `lab1_tb.vhd`



is placed under **lab1.srcs > sim\_1 > imports > lab1**. Double-click on the **lab1\_tb** in the *Sources* pane to view its contents.

```
-----
-- Entity Name: lab1_tb
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity lab1_tb is
end lab1_tb;

architecture Behavioral of lab1_tb is
-- component declaration for the Unit Under Test (UUT)
  COMPONENT lab1 is
    Port (swt : in STD_LOGIC_VECTOR (7 downto 0);
          led : out STD_LOGIC_VECTOR (7 downto 0));
  end COMPONENT;

  signal switches, leds, e_leds: std_logic_vector(7 downto 0);

  function expected_led (swt : std_logic_vector(7 downto 0))
    return std_logic_vector is
    variable exp_led: STD_LOGIC_VECTOR (7 downto 0);
  begin
    exp_led(0) := not swt(0);
--    exp_led(0) := swt(0); -- replace the previous line with this one, to fail the test
    exp_led(1) := swt(1) and not swt(2);
    exp_led(3) := swt(2) and swt(3);
    exp_led(2) := exp_led(1) or exp_led(3);
    exp_led(7 downto 4) := swt(7 downto 4);
    return exp_led;
  end expected_led;

begin

  DUT: lab1
  port map(
    swt => switches,
    led => leds);

  test: process
  begin
    for i in 0 to 255 loop
      wait for 50ns;
      switches <= conv_std_logic_vector(i,8);
      wait for 10ns;
      e_leds <= expected_led(switches);
      wait for 0ns;
      if leds = e_leds then
        report "LED output matched at " & time'IMAGE(now);
      else
        report "LED output mis-matched at " & time'IMAGE(now) & ": expected: "
          & std_logic'image(e_leds(7)) & std_logic'image(e_leds(6)) & std_logic'image(e_leds(5))
          & std_logic'image(e_leds(4)) & std_logic'image(e_leds(3)) & std_logic'image(e_leds(2))
          & std_logic'image(e_leds(1)) & std_logic'image(e_leds(0)) & " actual: "
          & std_logic'image(leds(7)) & std_logic'image(leds(6)) & std_logic'image(leds(5))
          & std_logic'image(leds(4)) & std_logic'image(leds(3)) & std_logic'image(leds(2))
          & std_logic'image(leds(1)) & std_logic'image(leds(0));
      end if;
    end loop;
  end process;
end Behavioral;
```

**Figure 12. The self-checking testbench**

The testbench invokes IEEE library and needed packages. The IEEE.STD\_LOGIC\_ARITH package is required for using the conv\_std\_logic\_vector function.

The entity is void for a testbench (there is no input or output signal for test).

The architecture declarations are:

- The tested component DUT (device under test)
- Some signals
- A function describing the same module functionality for the expected value computation

The architecture body:

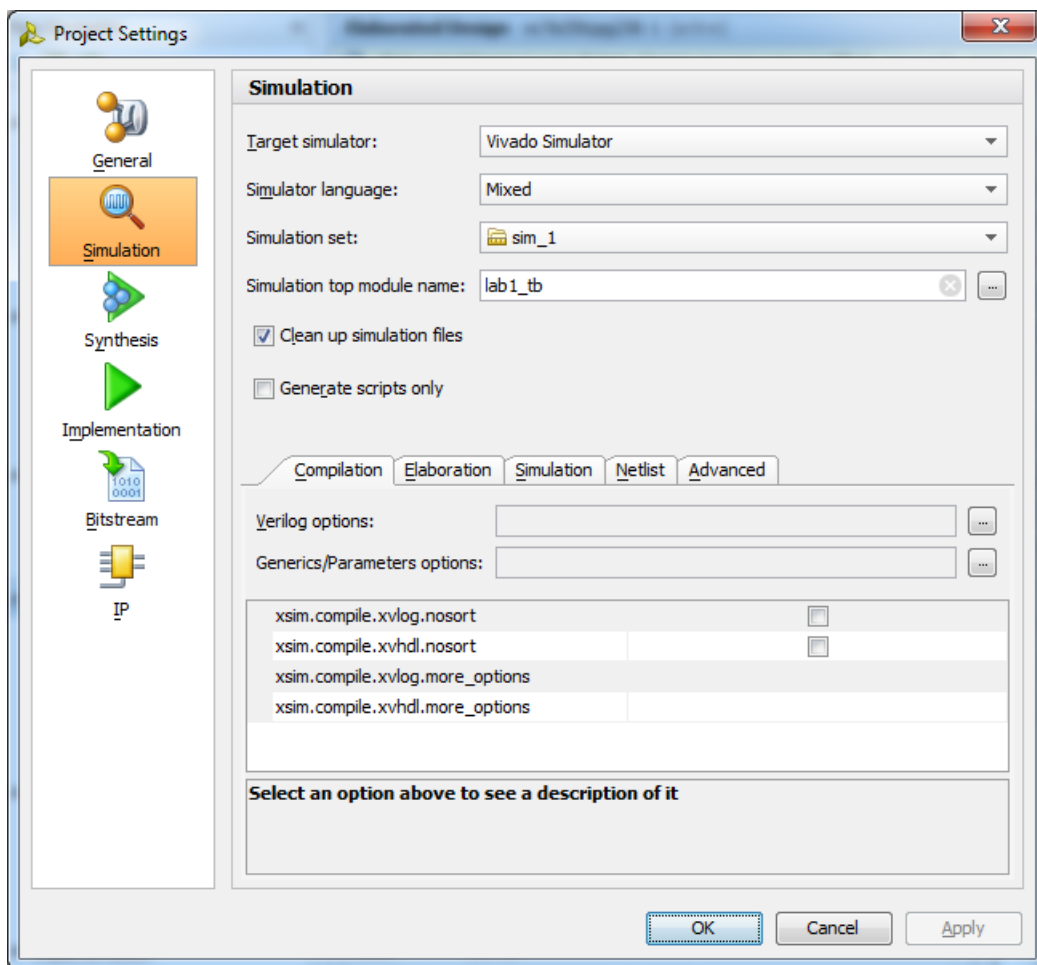
- instantiates the DUT (device/module under test).
- define the stimuli generation, and compare the expected output with what the DUT provides.
- print the message in the simulator console window when the simulation is run.

## 2-2. Simulate the design for 200 ns using the Vivado simulator.

**2-2-1.** Select **Simulation Settings** under the *Project Manager* tasks of the *Flow Navigator* pane.

A **Project Settings** form will appear showing the **Simulation** properties form.

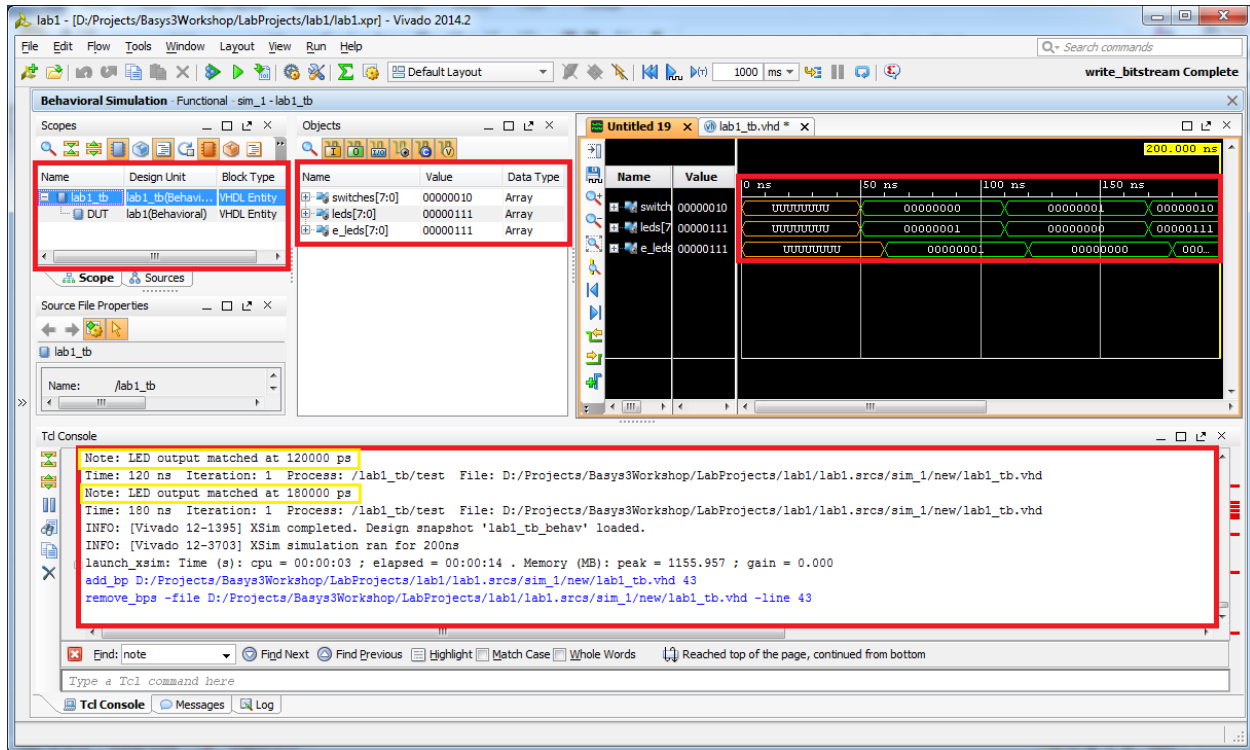
**2-2-2.** Select the **Simulation** tab, and set the **Simulation Run Time** value to 200 ns and click **OK**.



**Figure 13. Setting simulation run time**

**2-2-3.** Click on **Run Simulation > Run Behavioral Simulation** under the *Project Manager* tasks of the *Flow Navigator* pane.

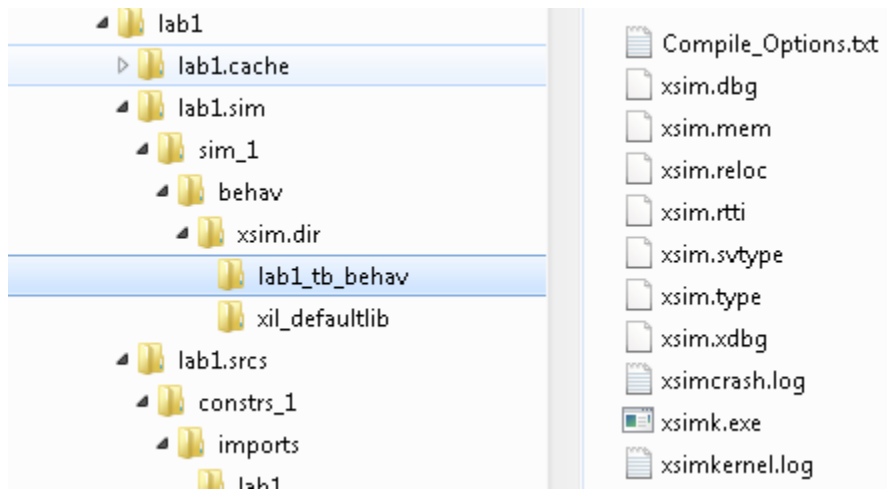
The testbench and source files will be compiled and the Vivado simulator will be run (assuming no errors). You will see a simulator output similar to the one shown below.



**Figure 14. Simulator output**

You will see four main views: (i) *Scopes*, where the testbench hierarchy as well as gbl instances are displayed, (ii) *Objects*, where top-level signals are displayed, (iii) the waveform window, and (iv) *Tcl Console* where the simulation activities are displayed. Notice that since the testbench used is self-checking, the results are displayed as the simulation is run.

















Notice that the **lab1.sim** directory is created under the **lab1** directory, along with several lower-level directories.



**Figure 15. Directory structure after running behavioral simulation**

You will see several buttons next to the waveform window which can be used for the specific purpose as listed in the table below.

Table 1: Various buttons available to view the waveform

	Waveform options
	Save the waveform
	Zoom In
	Zoom Out
	Zoom Fit
	Zoom to cursor
	Go to Time 0
	Go to Last Time
	Previous Transition
	Next Transition
	Add Marker
	Previous Marker
	Next Marker
	Swap Cursors
	Snap to Transition
	Floating Ruler

**2-2-4.** Click on the *Zoom Fit* button () to see the entire waveform.

Notice that the output changes when the input changes.

You can also float the simulation waveform window by clicking on the Float button on the upper right hand side of the view. This will allow you to have a wider window to view the simulation waveforms. To reintegrate the floating window back into the GUI, simply click on the Dock Window button.

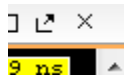


Figure 16. Float Button



Figure 17. Dock Window Button

## 2-3. Change display format if desired.

**2-3-1.** Select **switches[7:0]** in the waveform window, right-click, select *Radix*, and then select *Hexadecimal*. Leave the **leds[7:0]** and **e\_led[7:0]** radix to *binary* as we want to see each output bit.

**2-4. Add more signals to monitor the lower-level signals and continue to run the simulation for 500 ns.**

**2-4-1.** Expand the **lab1\_tb** instance, if necessary, in the *Scopes* window and select the **dut** instance. The **swt[7:0]** and **led[7:0]** signals will be displayed in the *Objects* window.

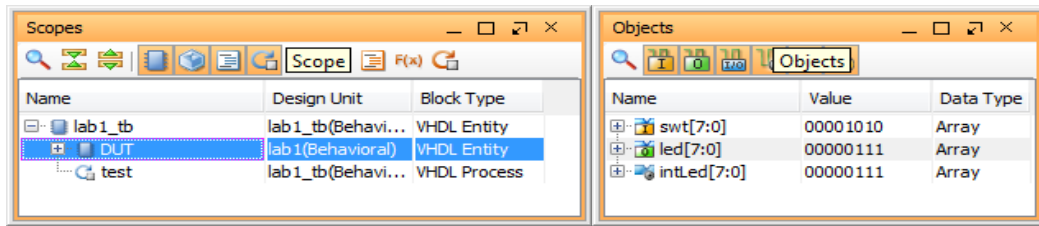


Figure 18. Selecting lower-level signals

2-4-2. Select **swt[7:0]** and **led[7:0]** and drag them into the waveform window to monitor those lower-level signals.

2-4-3. On the simulator tool buttons ribbon bar, type 500 over in the simulation run time field, click on the drop-down button of the units field and select ns ( 500 ns ) if we want to run for 500 ns (total of 700 ns), and click on the ( ) button. The simulation will run for an additional 500 ns.

2-4-4. Click on the *Zoom Fit* button and observe the output.

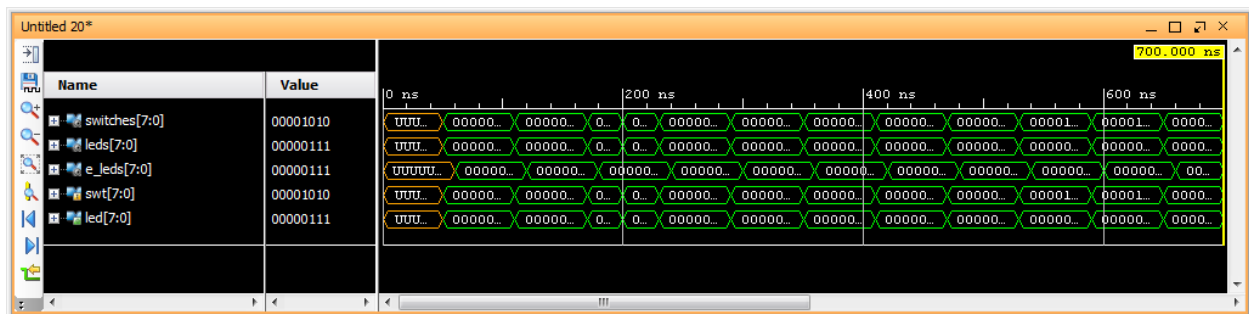


Figure 19. Running simulation for additional 500 ns

Observe the Tcl Console and see the output is being displayed as the testbench uses the report task.

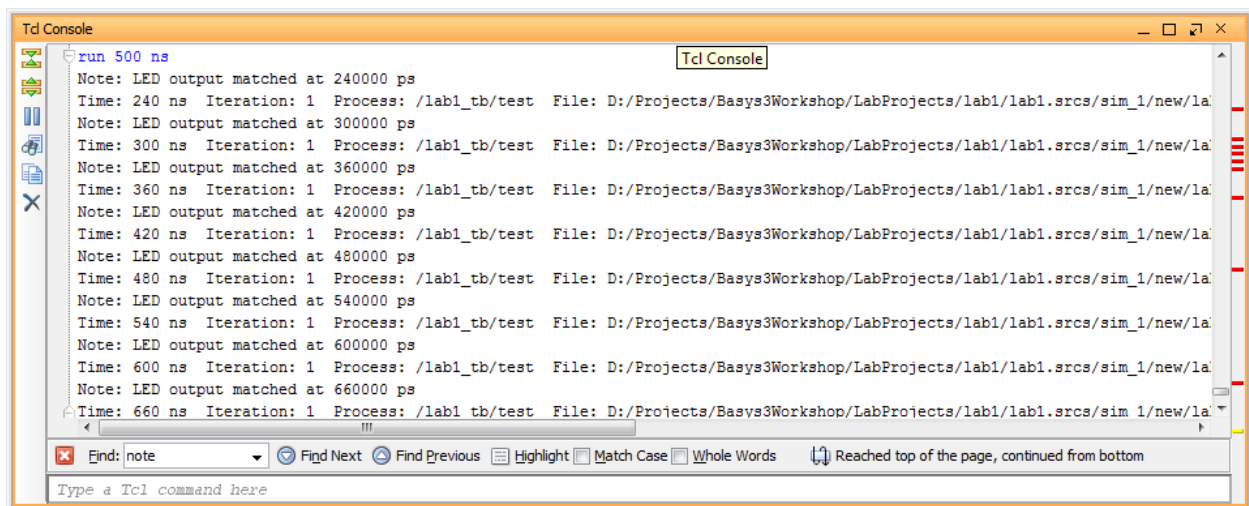


Figure 20. Tcl Console output after running the simulation for additional 500 ns

2-4-5. Close the simulator by selecting **File > Close Simulation**. Click **OK** and then click **No** to close it without saving the waveform.

## Synthesize the Design

## Step 3

### 3-1. Synthesize the design with the Vivado synthesis tool and analyze the Project Summary output.

#### 3-1-1. Click on **Run Synthesis** under the *Synthesis* tasks of the *Flow Navigator* pane.

The synthesis process will be run on the lab1.vhd file (and all its hierarchical files if they exist). When the process is completed a *Synthesis Completed* dialog box with three options will be displayed.

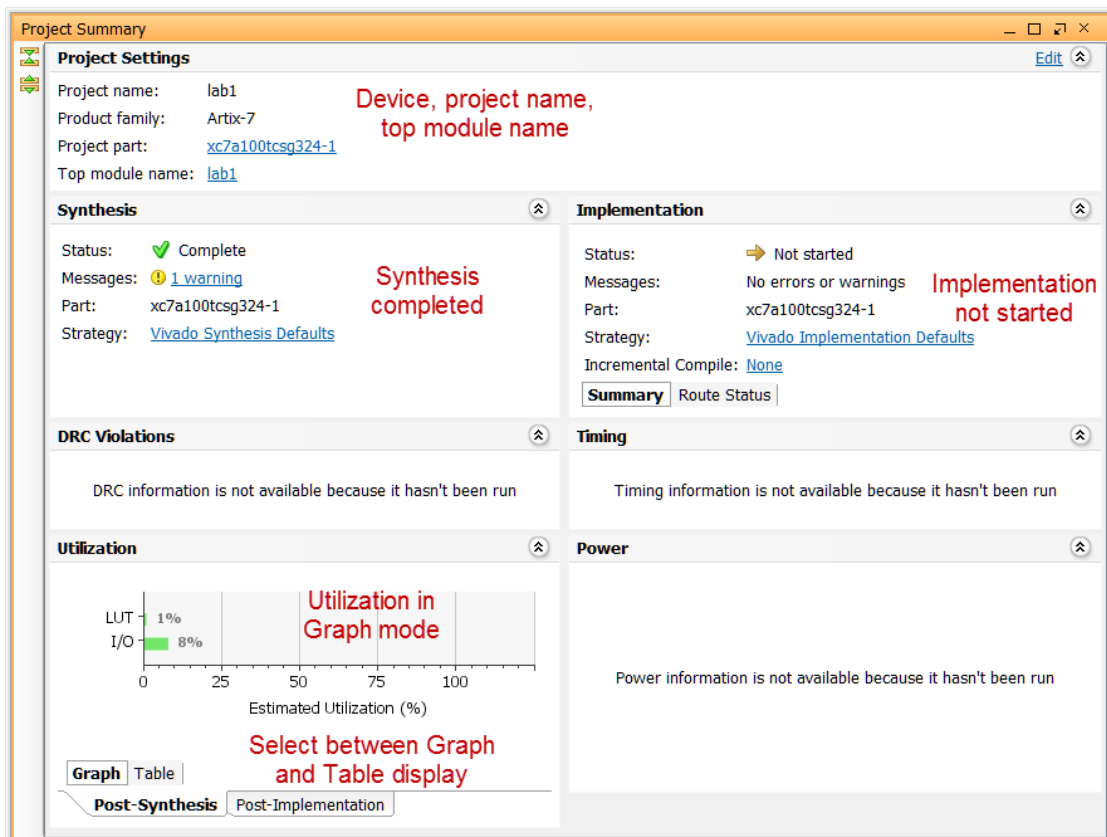
#### 3-1-2. Select the *Open Synthesized Design* option and click **OK** as we want to look at the synthesis output before progressing to the implementation stage.

Click **Yes** to close the elaborated design if the dialog box is displayed.

#### 3-1-3. Select the **Project Summary** tab and understand the various windows.

If you don't see the Project Summary tab then select **Layout > Default Layout**, or click the

**Project Summary** icon .

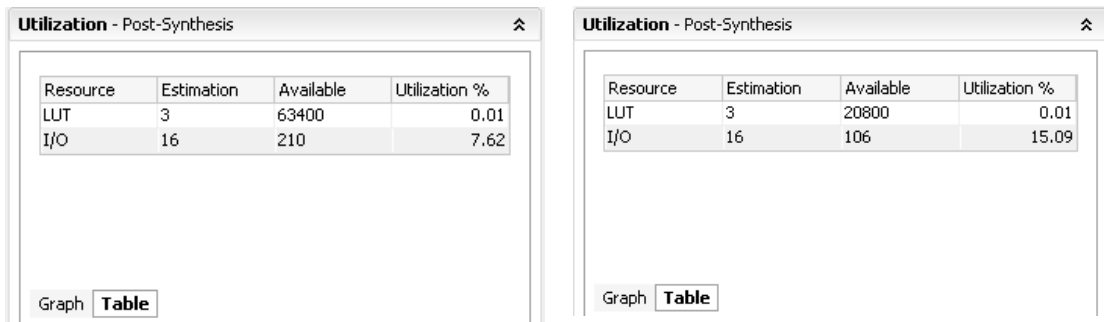


**Figure 21. Project Summary view**

Click on the various links to see what information they provide and which allows you to change the synthesis settings.

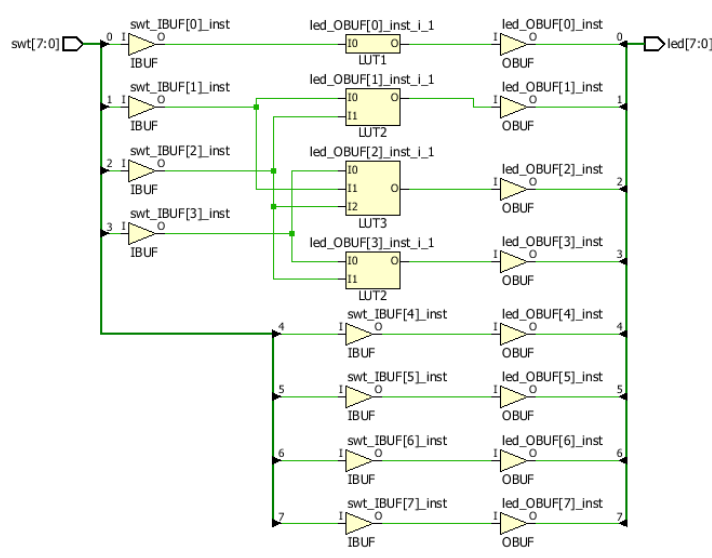
#### 3-1-4. Click on the **Table** tab in the **Project Summary** tab.

Notice that there are an estimated three LUTs and 16 IOs (8 input and 8 output) that are used.



**Figure 22. Resource utilization estimation summary for the Nexys4 (left) and Basys3 (right)**

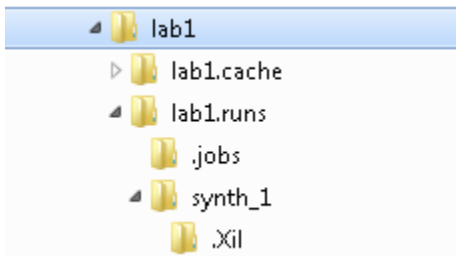
**3-1-5.** In The *Flow Navigator*, under *Synthesis* (expand *Synthesized Design* if necessary), click on **Schematic** to view the synthesized design in a schematic view.



**Figure 23. Synthesized design's schematic view**

Notice that IBUFs and OBUFs are automatically instantiated (added) to the design as the input and output are buffered. The logical gates are implemented in LUTs (1 input is listed as LUT1, 2 input is listed as LUT2, and 3 input is listed as LUT3). Four gates in RTL analysis output are mapped onto four LUTs in the synthesized output.

Using Windows Explorer, verify that **lab1.runs** directory is created under **lab1**. Under the **runs** directory, **synth\_1** directory is created which holds several files related to synthesis.



**Figure 24. Directory structure after synthesizing the design**

## Implement the Design

## Step 4

### 4-1. Implement the design with the Vivado Implementation Defaults (Vivado Implementation 2014) settings and analyze the Project Summary output.

4-1-1. Click on **Run Implementation** under the *Implementation* tasks of the *Flow Navigator* pane.

The implementation process will be run on the synthesized design. When the process is completed an *Implementation Completed* dialog box with three options will be displayed.

4-1-2. Select **Open implemented design** and click **OK** as we want to look at the implemented design in a Device view tab.

4-1-3. Click **Yes**, if prompted, to close the synthesized design.

The implemented design will be opened.

4-1-4. In the *Netlist* pane, select one of the nets (e.g. led\_OBUF[1]) and notice that the net displayed in the X1Y1 clock region in the Device view tab (you may have to zoom in to see it).

4-1-5. If it is not selected, click the *Routing Resources* icon  to show routing resources.

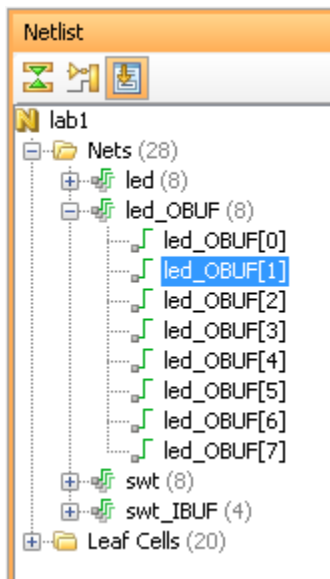
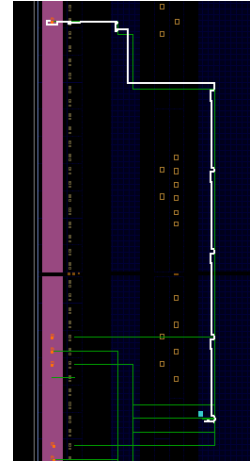
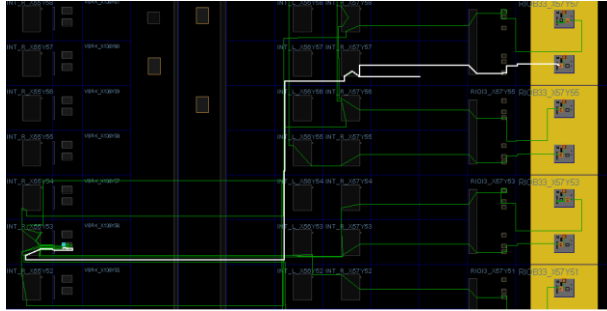


Figure 25. Selecting a net



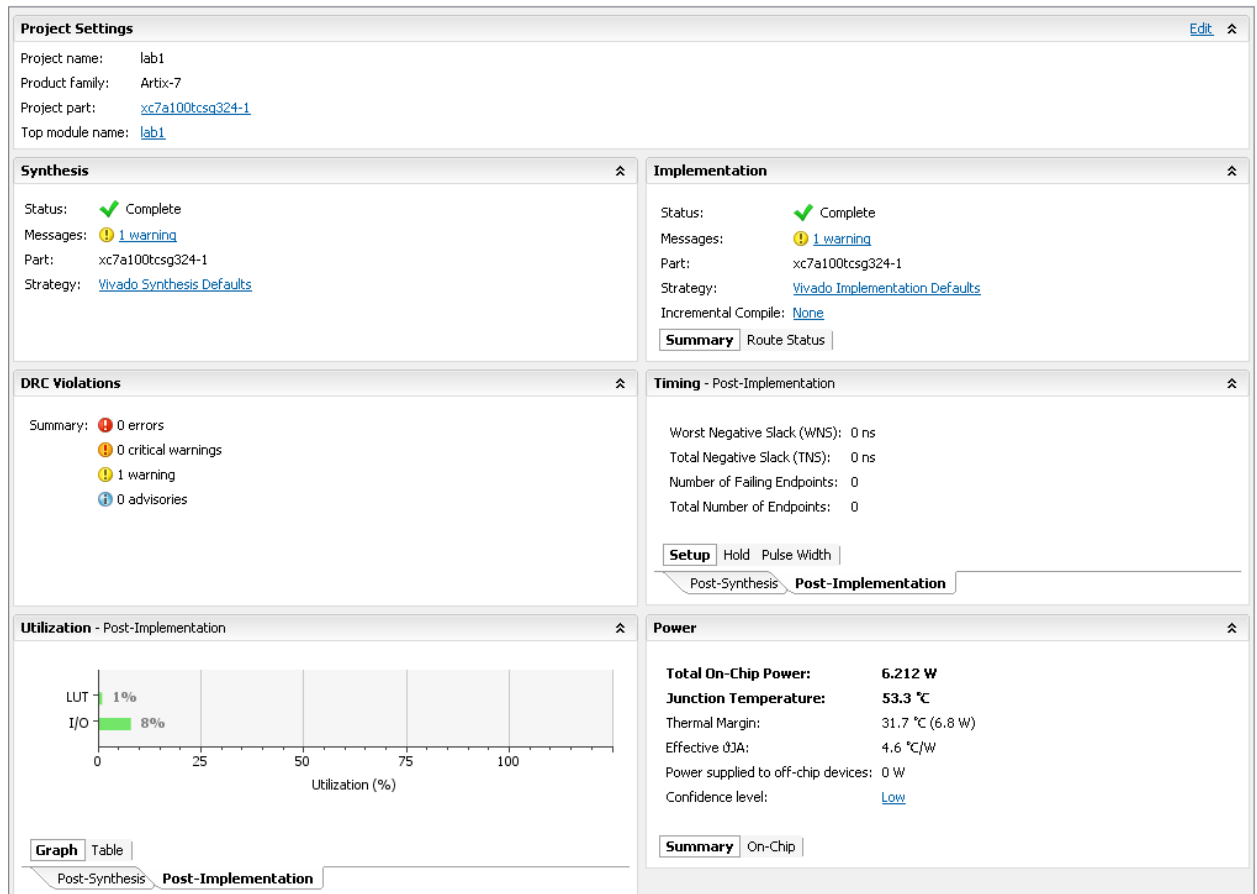


**Figure 26. Viewing implemented design for the Nexys4 (left) and Basys3 (right)**

**4-1-6.** Close the implemented design view and select the **Project Summary** tab (you may have to change to the Default Layout view) and observe the results.

Select the Post-Implementation tab.

Notice that the actual resource utilization is three LUTs and 16 IOs. Also, it indicates that no timing constraints were defined for this design (since the design is combinatorial).



**Figure 27. Implementation results for the Nexys4**

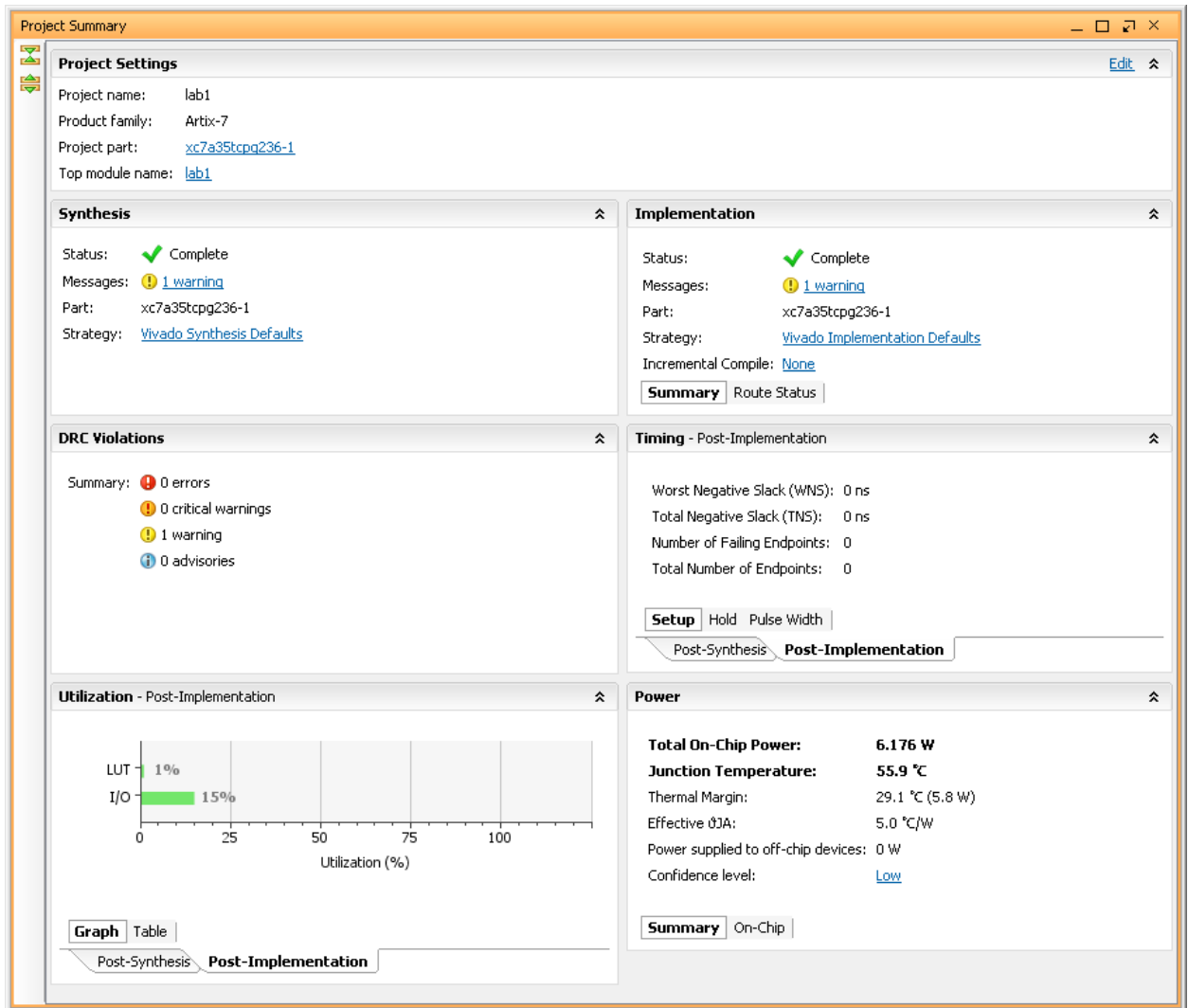


Figure 27. Implementation results for the Basys3

Using the Windows Explorer, verify that **impl\_1** directory is created at the same level as **synth\_1** under the **lab1.runs** directory. The **impl\_1** directory contains several files including the implementation report files.

- 4-1-7.** In Vivado, select the **Reports** tab in the bottom panel (if not visible, click *Window* in the menu bar and select **Reports**), and double-click on the *Utilization Report* entry under the *Place Design* section. The report will be displayed in the auxiliary view pane showing resource utilization. Note that since the design is combinatorial no registers are used.

Reports			
Name	Modified	Size	
[-] Synth Design (synth_design)			
Vivado Synthesis Report	6/20/14 1:25 PM	15.8 KB	
Utilization Report	6/20/14 1:25 PM	5.7 KB	
[-] Place Design (place_design)			
Vivado Implementation Log	6/20/14 1:34 PM	16.6 KB	
Pre-Placement Incremental Reus...			
IO Report	6/20/14 1:33 PM	111.5 KB	
Clock Utilization Report	6/20/14 1:33 PM	5.0 KB	
Utilization Report	6/20/14 1:33 PM	7.7 KB	
Control Sets Report	6/20/14 1:33 PM	2.5 KB	
Incremental Reuse Report			
[-] Route Design (route_design)			
Vivado Implementation Log	6/20/14 1:34 PM	16.6 KB	
WebTalk Report			
DRC Report	6/20/14 1:34 PM	1.8 KB	
Power Report	6/20/14 1:34 PM	7.3 KB	
Route Status Report	6/20/14 1:34 PM	0.6 KB	
Timing Summary Report	6/20/14 1:34 PM	7.0 KB	
Incremental Reuse Report			
[-] Post-Route Phys Opt Design (post_route_phys_opt_design)			
Post-Route Physical Optimizatio...			
[-] Write Bitstream (write_bitstream)			
Vivado Implementation Log			
WebTalk Report			

Figure 28. Available reports to view

## Perform Timing Simulation


## Step 5

### 5-1. Run a timing simulation.

- 5-1-1. Select **Run Simulation > Run Post-Implementation Timing Simulation** process under the *Simulation* tasks of the *Flow Navigator* pane.

The Vivado simulator will be launched using the implemented design and **lab1\_tb** as the top-level module.

Using the Windows Explorer, verify that **timing** directory is created under the **lab1.sim > sim\_1 > impl** directory. The **timing** directory contains generated files to run the timing simulation.

- 5-1-2. Click on the **Zoom Fit** button to see the waveform window from 0 to 200 ns.
- 5-1-3. Right-click at 50 ns (where the switch input is set to 0000000b) and select **Markers > Add Marker**.
- 5-1-4. Similarly, right-click and add a marker at around 55.000 ns where the **leds** changes.
- 5-1-5. You can also add a marker by clicking on the Add Marker button (  ). Click on the **Add Marker** button and left-click at around 60 ns where **e\_led** changes.

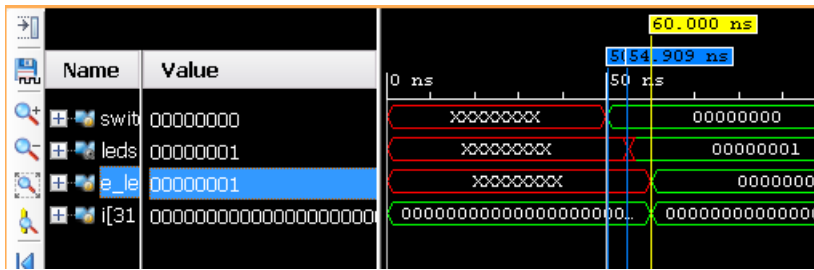


Figure 29. Timing simulation output

Notice that we monitored the expected led output at 10 ns after the input is changed (see the testbench) whereas the actual delay is about 5.000 ns.

5-1-6. Close the simulator by selecting **File > Close Simulation** without saving any changes.

## Generate the Bitstream and Verify Functionality

## Step 6

6-1. Connect the board and power it ON. Generate the bitstream, open a hardware session, and program the FPGA.

6-1-1. Make sure that the Micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector).

6-1-2. Make sure that the board is set to use USB power (via the Power Select jumper JP3 on the Nexys4 and JP2 on the Basys3)

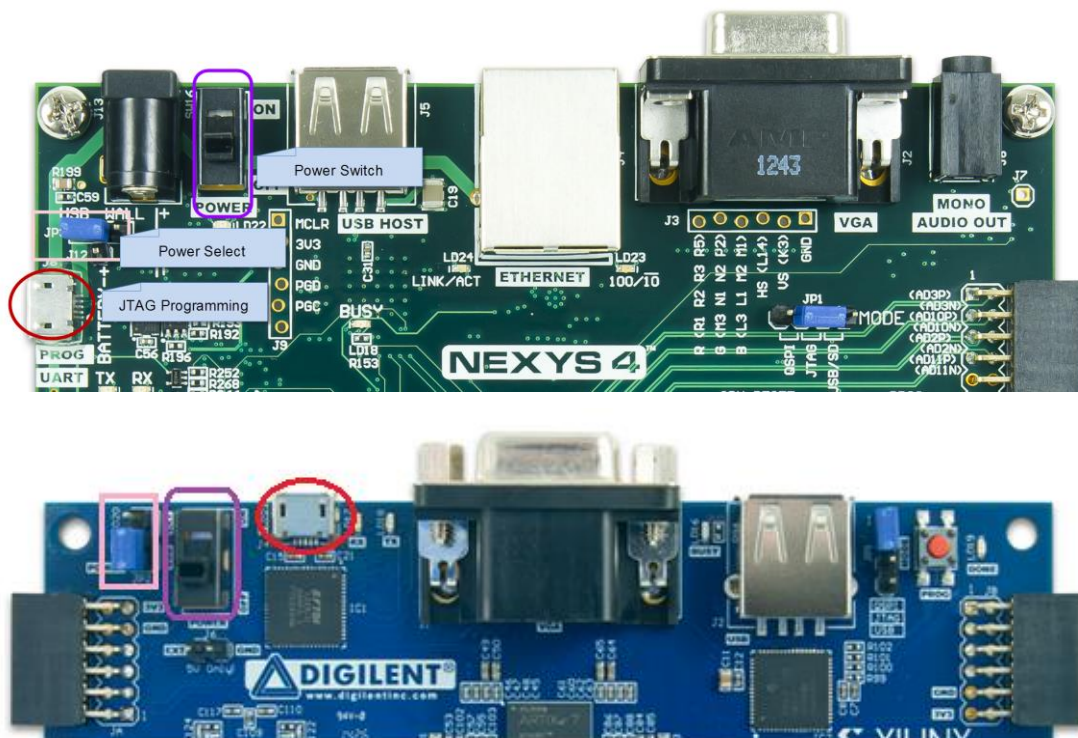
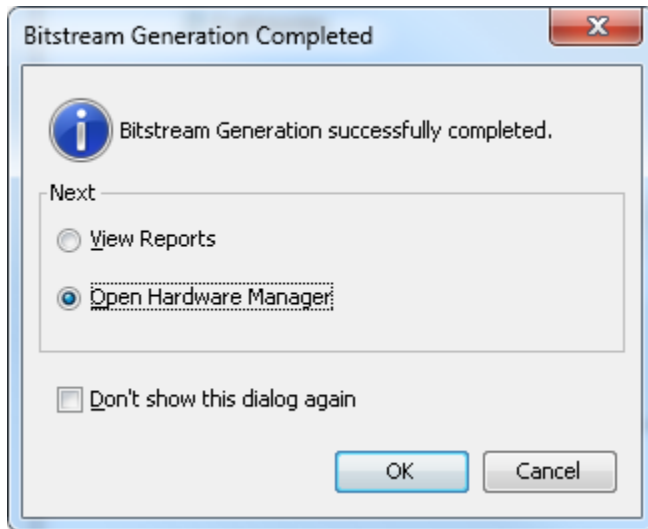


Figure 30. Board connection for the Nexys 4 (top) and Basys3 (bottom)

- 6-1-3. Power **ON** the board.
- 6-1-4. Click on the **Generate Bitstream** entry under the *Program and Debug* tasks of the *Flow Navigator* pane.

The bitstream generation process will be run on the implemented design. When the process is completed a *Bitstream Generation Completed* dialog box with two options will be displayed.



**Figure 31. Bitstream generation**

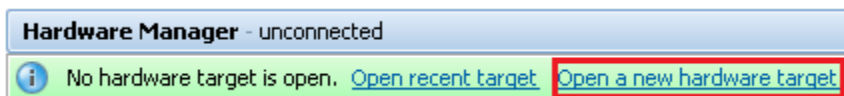
This process will have generated a **lab1.bit** file under **impl\_1** directory in the **lab1.runs** directory.

- 6-1-5. Select the *Open Hardware Manager* option and click **OK**.

The Hardware Manager window will open indicating “unconnected” status.

- 6-1-6. Click on the **Open a new hardware target** link.

You can also click on the **Open recent target** link if the board was already targeted before.



**Figure 32. Opening new hardware target**

- 6-1-7. Click **Next** to see the Hardware Server Settings form.

- 6-1-8. Click **Next** with the Hardware Target selected.

The JTAG cable which uses the Xilinx\_tcf should be detected and identified as a hardware target. It will also show the hardware devices detected in the chain.

- 6-1-9. Click **Next** and then **Finish**.

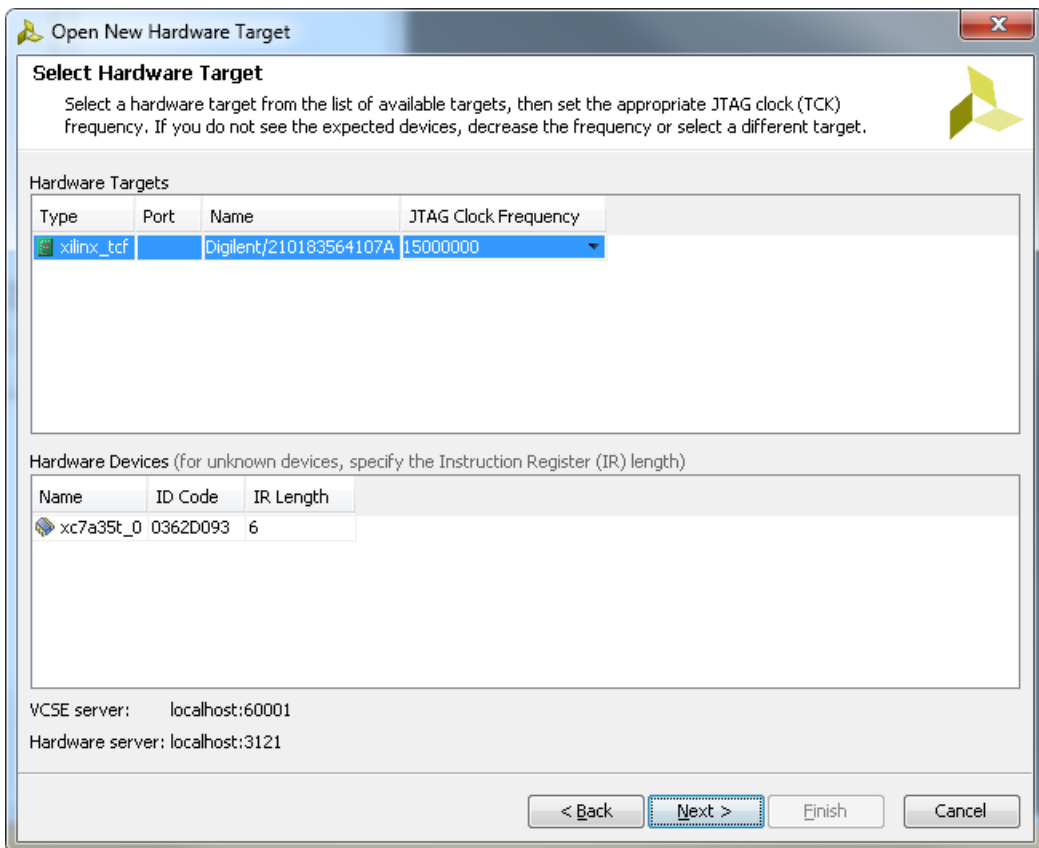
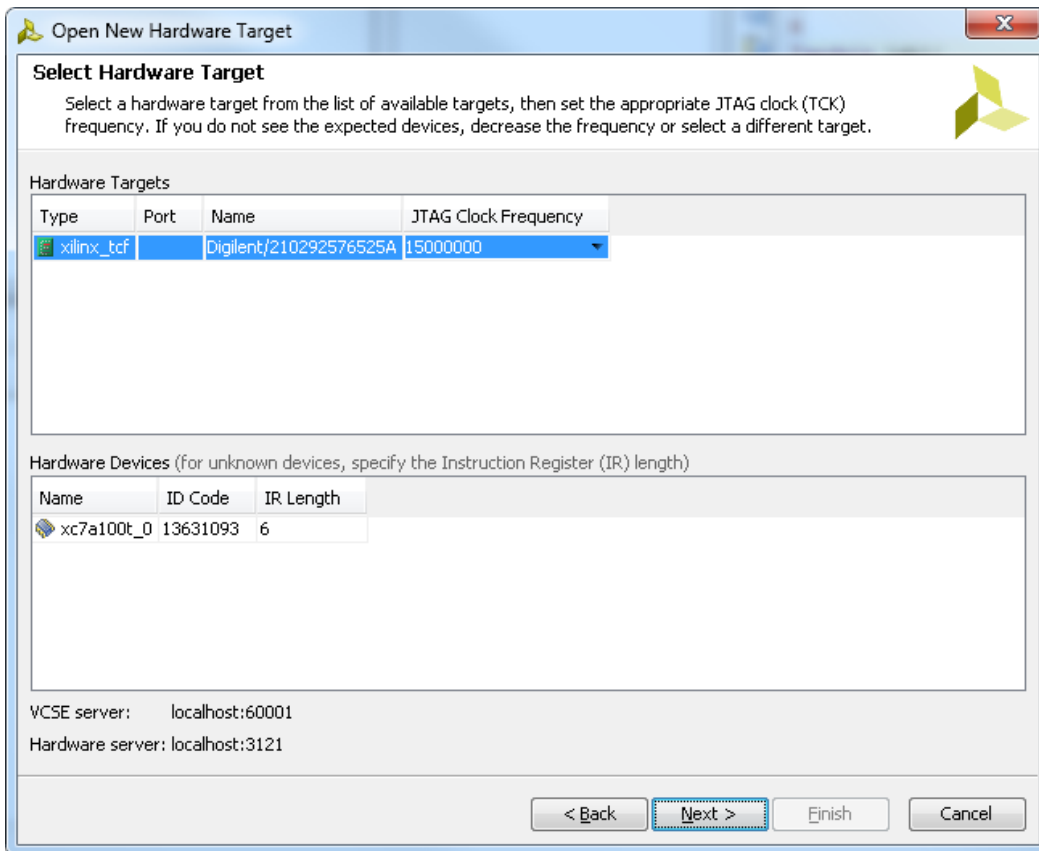
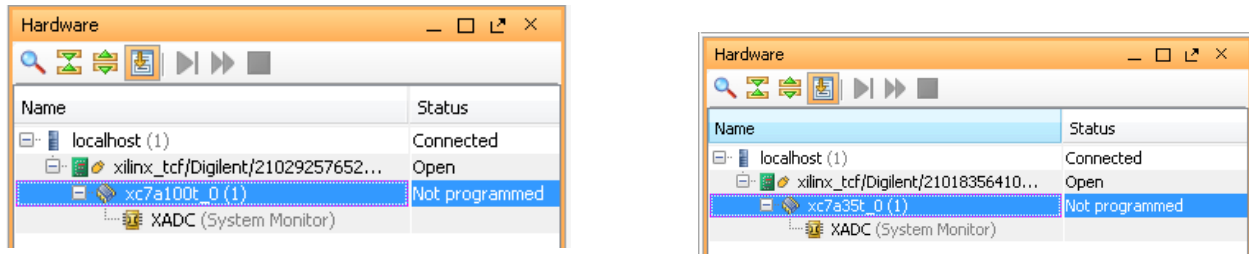


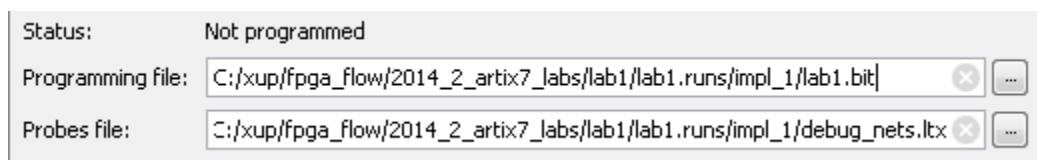
Figure 33. New hardware target detection for the Nexys 4 (top) and Basys3 (bottom)

The Hardware Session status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.



**Figure 34. Opened hardware session for the Nexys4 (left) and Basys3 (right)**

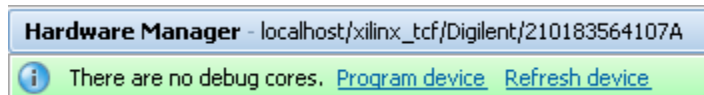
**6-1-10.** Select the device and verify that the lab1.bit is selected as the programming file in the General tab.



**Figure 35. Programming file**

**6-1-11.** Click on the *Program device > XC7A100T\_0* or the *XC7A35T\_0* link in the green information bar to program the target FPGA device.

Another way is to right click on the device and select *Program Device...*



**Figure 36. Selecting to program the FPGA**

**6-1-12.** Click **Program** to program the FPGA.

The DONE light will light when the device is programmed. You may see some other LEDs lit depending on switch positions.

**6-1-13.** Verify the functionality by flipping switches and observing the output on the LEDs (Refer to the earlier logic diagram).

**6-1-14.** When satisfied, power **OFF** the board.

**6-1-15.** Close the hardware session by selecting **File > Close Hardware Manager**.

**6-1-16.** Click **OK** to close the session.

**6-1-17.** Close the **Vivado** program by selecting **File > Exit** and click **OK**.

## Conclusion

The Vivado software tool can be used to perform a complete design flow. The project was created using the supplied source files (HDL model and user constraint file). A behavioral simulation using the provided testbench was done to verify the model functionality. The model was then synthesized, implemented, and a bitstream was generated. The timing simulation was run on the implemented design using the same testbench. The functionality was verified in hardware using the generated bitstream.