**الجلسة الرابعة:**

**تصميم الدارات الالكترونية بلغة VHDL / 3/**

**البنية التفرعية للغة VHDL**

**مقدمة :**

هنالك صنفان رئيسيان من الدارات المنطقية :

**الدارات التركيبية** : وهي مجموعة من البوابات الأساسية And , Or , كما تعرف أيضا بأنها الدارات التي لا يتعلق خرجها بالزمن بل فقط يرتبط بالحالات المنطقية للمداخل الدارات بدون ذاكر ة, وفي مجال المنطق المبرمج لا يهم ترتيب التعليمات عند التعامل مع هذه الدارات .

**الدارات التتابعية** : ترتبط فيها القيمة المنطقية بالزمن ويتم التمييز بين الحالة الآنية في اللحظة n وسابقتها n-1 , تستخدم في هذه البنى القلابات على اختلاف أنواعها (الدارات ذات الذاكرة) , وفي مجال المنطق المبرمج فإنه من المهم جداً ترتيب التعليمات عند التعامل مع هذه الدارات .

**طريقة تنفيذ البرنامج في لغة VHDL:**

بعد أن تعرفنا على أسس و مبادىء لغة VHDL نبدأ الآن بالتعرف على طرق كتابة الكود البرمجي , إذ تختلف طريقة كتابة الكود باختلاف الغاية المرجوة منه وطريقة التنفيذ المعتبرة. على خلاف لغات البرمجة التقليدية يتم تنفيذ التعليمات في لغة VHDL على التوازي Concurrent (parallel) , في هذه الطريقة يتم تنفيذ الكود البرمجي بنفس الوقت تقريباً لكافة التعليمات ، وكون هذه اللغة تفسر عمليا على شكل كيان صلب ) دارة منطقية ( وكل تعليمة تؤول إلى حجز كتل منطقية قابلة للبرمجة او توصيلها معاً . إذاً يمكننا القول بأن هذه التعليمات قد تحولت إلى دارة بكل ما للكلمة من معنى ، وبهذه الحالة فإن التصميم المتكامل هو مجموعة دارات تعمل على التوازي وتنقل إشارات الدخل إلى الخرج بزمن مهمل تقريباً باستثناء تأخير الانتشار . PD

نظراً لهذه الخصوصية في تنفيذ التعليمات ونظراً للحاجة في بعض الحالات من تحويل التنفيذ إلى النمط التقليدي أي ترتيب تنفيذ التعليمات بحسب حاجة المبرمج وما يراه من ضرورة في عمل النظام ، تم إيجاد مجموعة تعليمات كتعليمة End process .. Process التي يمكن إضافتها في وحدة التوصيف الداخلي ليتم تنفيذ التعليمات التي تذكر بعدها بترتيب ورودها من الأعلى إلى الأسفل ليحول بذلك آلية التنفيذ إلى النمط التتابعي . Sequential

من أنماط التعليمات التي يتم تنفيذها بصورة تسلسلية نذكر PROCESS, FUNCTION, PROCEDURE , حيث يتم تنفيذ ما بداخل هذه الكتل تسلسلياً , ولكن على مستوى الكتلة نفسها فإنها تنفذ تفرعياً شأنها شأن أي تعليمة في VHDL , وبالتالي ضمن هذه الكتل يتم التعامل مع تعليمات تكون مهمتها تحقيق الترتيب أو التزامن في تنفيذ التعليمات تسمى هذه الكتل ب GUARDED BLOCK, أو محطات الانتظار التي يكون من مهامها الأساسية منع الانتقال إلى كتلة برمجية أخرى حتى يتحقق شرط ما يكون هو شرط التزامن مثلاً .

إذاً فالصفة المهيمنة على لغة VHDL هي صفة التنفيذ التفرعي أي ما يناسب الدارات التجميعية ، إلا أنها تدعم بتعليمات خاصة تدعم عمليات التنفيذ التتابعي وبالتالي توفر ما يناسب الدارات التتابعية .

سوف نتعامل في هذه الجلسة مع تعليمات البنية التفرعية في حين أننا خصصنا الجلسة التالية للتعامل مع البنية التتابعية (التسلسلية) وما لها من خصوصية ☺ .

**البنية التفرعية في لغة VHDL :**

* تشكل التعليمات التالية البنية التفرعية في لغة VHDL :

1. عمليات الإسناد .
2. العمليات المنطقية.
3. العمليات الحسابية.
4. تعليمة when .
5. تعليمة block.
6. تعليمة generate.

**تطبيق1**

يبين الشكل التالي ناخب ذو أربعة مداخل منطقية وخرج منطقي وحيد مع مدخلي انتخاب سنقوم بتوصيفه بعدة طرق لنتعلم من خلاله أشكال التعليمة WHEN **.**



-- Multiplexer #1

---------------------------------------

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

---------------------------------------

ENTITY mul IS

PORT ( a, b, c, d, s0, s1: IN STD\_LOGIC;

y: OUT STD\_LOGIC);

END mul;

---------------------------------------

ARCHITECTURE MTK\_logic OF mul IS

BEGIN

y <= (a AND NOT s1 AND NOT s0) OR

(b AND NOT s1 AND s0) OR

(c AND s1 AND NOT s0) OR

(d AND s1 AND s0);

END MTK\_logic;

**تعليمة when:**

تأخذ هذه التعليمة عدة أشكال , من أهم هذه الأشكال :

1. **الشكل الأول:**

assignment **WHEN** condition **ELSE**

assignment **WHEN** condition **ELSE**

**... ;**

1. **الشكل الثاني:**

**WITH** identifier **SELECT**

assignment **WHEN** value,

assignment **WHEN** value,

assignment **WHEN** value,

assignment **WHEN** OTHERS**;**

-----------------------------------------------------------------------------------------

عند استخدام هذا الأشكال يصبح المثال السابق كمايلي:

-- Multiplexer #2

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-------------------------------------------

ENTITY mul IS

PORT ( a, b, c, d: IN STD\_LOGIC;

s: IN STD\_LOGIC\_VECTOR (1 DOWNTO 0);

y: OUT STD\_LOGIC);

END mul;

-------------------------------------------

ARCHITECTURE mux OF mul IS

BEGIN

**y <= a WHEN s="00" ELSE**

**b WHEN s="01" ELSE**

**c WHEN s="10" ELSE**

**d ;**

END mux;

عند استخدام هذا الشكل يصبح المثال السابق كمايلي:

-- Multiplexer #3

LIBRARY ieee ;

USE ieee.std\_logic\_1164.all;

-------------------------------------------

ENTITY mul IS

PORT ( a, b, c, d: IN STD\_LOGIC;

s: IN STD\_LOGIC\_VECTOR (1 DOWNTO 0);

y: OUT STD\_LOGIC);

END mul;

-------------------------------------------

ARCHITECTURE mux OF mul IS

BEGIN

**WITH s SELECT**

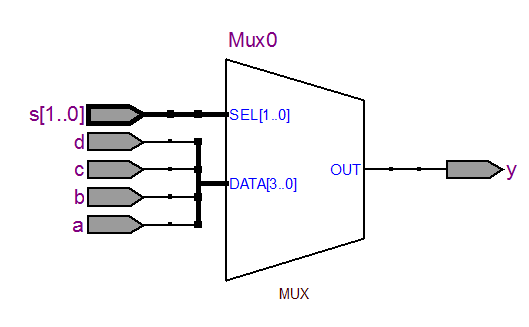
**y <= a WHEN "00", -- notice "," instead of ";"**

**b WHEN "01",**

**c WHEN "10",**

**d WHEN OTHERS; -- cannot be "d WHEN "11" "**

END mux;

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

----------------------------------------------

ENTITY mux IS

PORT ( a, b, c, d: IN STD\_LOGIC;

**sel: IN INTEGER RANGE 0 TO 3;**

y: OUT STD\_LOGIC);

END mux;

--------------------------------------------

ARCHITECTURE mux1 OF mux IS

BEGIN

**y <= a WHEN sel=0 ELSE**

**b WHEN sel=1 ELSE**

**c WHEN sel=2 ELSE**

**d;**

END mux1;

---------------------------------------------------------------------------------------

**تطبيق 2**

تصميم دارة عازل بمدخلي تمكين متعاكسيين :

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY tri IS

PORT ( E: IN STD\_LOGIC\_VECTOR(1 downto 0);

input: IN STD\_LOGIC\_VECTOR (7 DOWNTO 0);

Output: OUT STD\_LOGIC\_VECTOR (7 DOWNTO 0));

END tri ;

ARCHITECTURE tri\_state OF tri IS

BEGIN

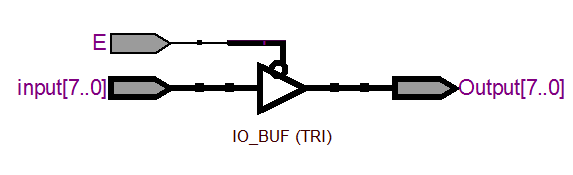
with E select

output <= input WHEN "01" | "10",

"ZZZZZZZZ" when others;

END tri\_state;

حاول تعديل التوصيف السابق لتأخذ بنية RTL هذا الشكل .



**تطبيق 3 :**

ماذا يمثل هذا التوصيف ؟

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

---------------------------------------------

ENTITY qqq IS

PORT ( x: IN STD\_LOGIC\_VECTOR (7 DOWNTO 0);

y: OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0));

END qqq;

---------------------------------------------

ARCHITECTURE qqqq OF qqq IS

BEGIN

y <= "000" WHEN x="00000001" ELSE

"001" WHEN x="00000010" ELSE

"010" WHEN x="00000100" ELSE

"011" WHEN x="00001000" ELSE

"100" WHEN x="00010000" ELSE

"101" WHEN x="00100000" ELSE

"110" WHEN x="01000000" ELSE

"111" WHEN x="10000000" ELSE

"ZZZ";

END qqqq ;

-----------------------------------------------------------------------------**تطبيق4 :**

المطلوب تصميم وحدة حساب ومنطق مصغرة ALU(Arithmetic Logic Unit) وفق التوصيف التالي : 



**تعليمة GENERATE :**

وهي تعليمة التكرار الحلقي لكتلة ما ضمن البرنامج , وهي خاصة بالنمط التفرعي ويقابلها في النمط التتابعي التعليمةLOOP**,** تسمح هذه التعليمة بتكرار كتلة من الكود البرمجي عدة مرات بحيث تسهل على المبرمج عملية إعادة كتابة اجزاء متشابهة من الكود مع اختلافات بسيطة .

تترافق مع هذه التعليمة الكلمة FOR التي تحد عدد مرات التكرار ، والشكل العام لهذه التعليمة هو :

-- FOR / GENERATE:

Label : **FOR** identifier **IN** range **GENERATE**

(concurrent assignments)

…..

END GENERATE ;

مثال:

SIGNAL x: BIT\_VECTOR (7 DOWNTO 0);

SIGNAL y: BIT\_VECTOR (15 DOWNTO 0);

SIGNAL z: BIT\_VECTOR (7 DOWNTO 0);

...

**G1: FOR i IN x'RANGE GENERATE**

**z(i) <= x(i) AND y(i+8);**

**END GENERATE;**

من الملاحظات الهامة على هذه التعليمة نذكر بأنه ينبغي تحديد عدد مرات التكرار بصورة مسبقة . كما يتبين لنا من خلال المثال التالي :

طريقة خاطئة :

NotOK: FOR i IN 0 TO choice GENERATE

(concurrent statements)

END GENERATE;

طريقة صحيحة :

OK: FOR i IN 0 TO 7 GENERATE

output(i)<='1' WHEN (a(i) AND b(i))='1' ELSE '0';

END GENERATE;

**التعليمة**  **BLOCK**

تقوم هذه التعليمة بتقسيم الكود البرمجي إلى كتل الأمر الذي يساعد في كتابة البرامج الطويلة دون خطأ .

في حال استخدام هذه التعليمة يصبح البرنامج على الشكل التالي :

ARCHITECTURE example ...

BEGIN

...

block1: BLOCK

BEGIN

...

END BLOCK block1

...

block2: BLOCK

BEGIN

...

END BLOCK block2;

...

END example;

يوجد شكلان لهذه التعليمة :

**الشكل الأول البسيط** Simple BLOCK

label: BLOCK

[declarative part]

BEGIN

(concurrent statements)

END BLOCK label;

**مثال :**

b1: **BLOCK**

SIGNAL a: STD\_LOGIC;

**BEGIN**

a <= input\_sig WHEN e ='1' ELSE '0';

**END BLOCK** b1;

كما يمكن أن يتم استعمال تعليمتين بصورة متداخلة كما يلي :

label1: BLOCK

[declarative part of top block]

BEGIN

[concurrent statements of top block]

label2: BLOCK

[declarative part nested block]

BEGIN

(concurrent statements of nested block)

END BLOCK label2;

[more concurrent statements of top block]

END BLOCK label1;

**الشكل الثاني** المركب Guarded BLOCK

وهو شكل خاص يتم من خلاله استخدام كتلة إضافية guard expression تمنع تنفيذ الكتلة البرمجية ضمن Block اذا لم يتحقق شرط منطقي يذكر في بدايتها . وبالتالي نستطيع تقسيم البرنامج إلى مجموعة كتل لا تتداخل فيما بينها وهي محمية بشروط منطقية .

الصيغة العامة لهذا الشكل :

label: BLOCK (guard expression)

[declarative part]

BEGIN

(concurrent guarded and unguarded statements)

END BLOCK label;

مثال 1

في القلاب من النوع D يتم منع تغيير حالة الخرج إلا بشرط وجود نبضات الساعة.

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-------------------------------

ENTITY latch IS

PORT (d, clk: IN STD\_LOGIC;

q: OUT STD\_LOGIC);

END latch;

-------------------------------

ARCHITECTURE latch OF latch IS

BEGIN

**b1: BLOCK (clk='1')**

**BEGIN**

**q <= GUARDED d;**

**END BLOCK b1;**

END latch;

مثال2 :

في هذا المثال سوف يتم تغيير حالة خرج القلاب فقط عند ورود الجبهة الصاعدة لنبضات الساعة

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-------------------------------

ENTITY dff IS

PORT ( d, clk, rst: IN STD\_LOGIC;

q: OUT STD\_LOGIC);

END dff;

-------------------------------

ARCHITECTURE dff OF dff IS

BEGIN

**b1: BLOCK (clk'EVENT AND clk='1')**

**BEGIN**

**q <= GUARDED '0' WHEN rst='1' ELSE d;**

**END BLOCK b1;**

END dff;