



國立臺北科技大學
National Taipei University of Technology

AP SoC軟硬體開發簡介

INTRODUCTION TO SOFTWARE AND HARDWARE DEVELOPMENT
BASED ON ALL PROGRAMMABLE SOC

講師：陳立業 博士

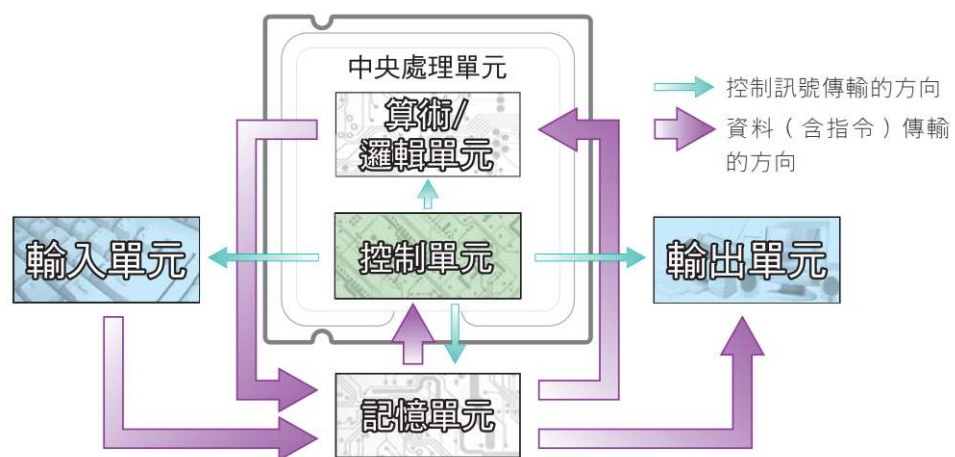
大綱

- 基礎概述
 - SoC
 - MCU
 - FPGA
 - All Programmable SoC
- Zynq軟硬體開發入門
 - 說明與工具
 - PL GPIO範例
 - 軟硬體FFT範例
- 討論

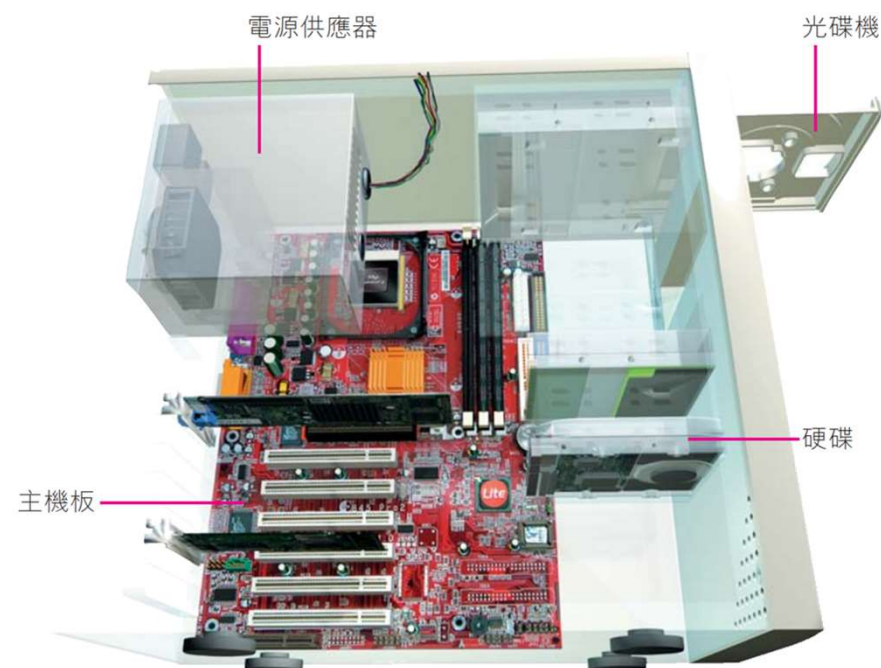
基礎概述

基礎概述—SoC

計概基礎



電腦五大單元



個人電腦內部元件

基礎概述—SoC

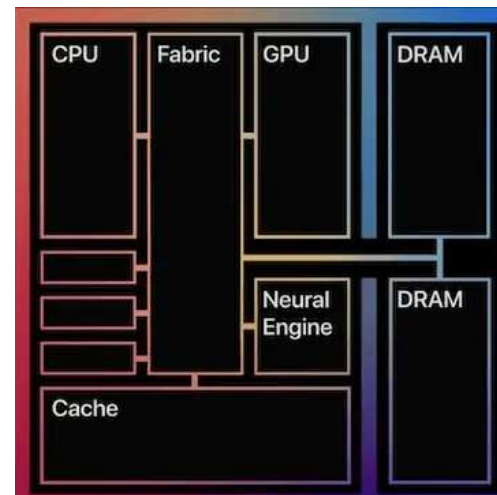
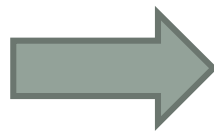
什麼是 SoC?

SoC 的縮寫是從 System on a Chip 來的，意思就是說：

把一整個系統「整合」在一片晶片上。



電腦主機板+CPU+RAM

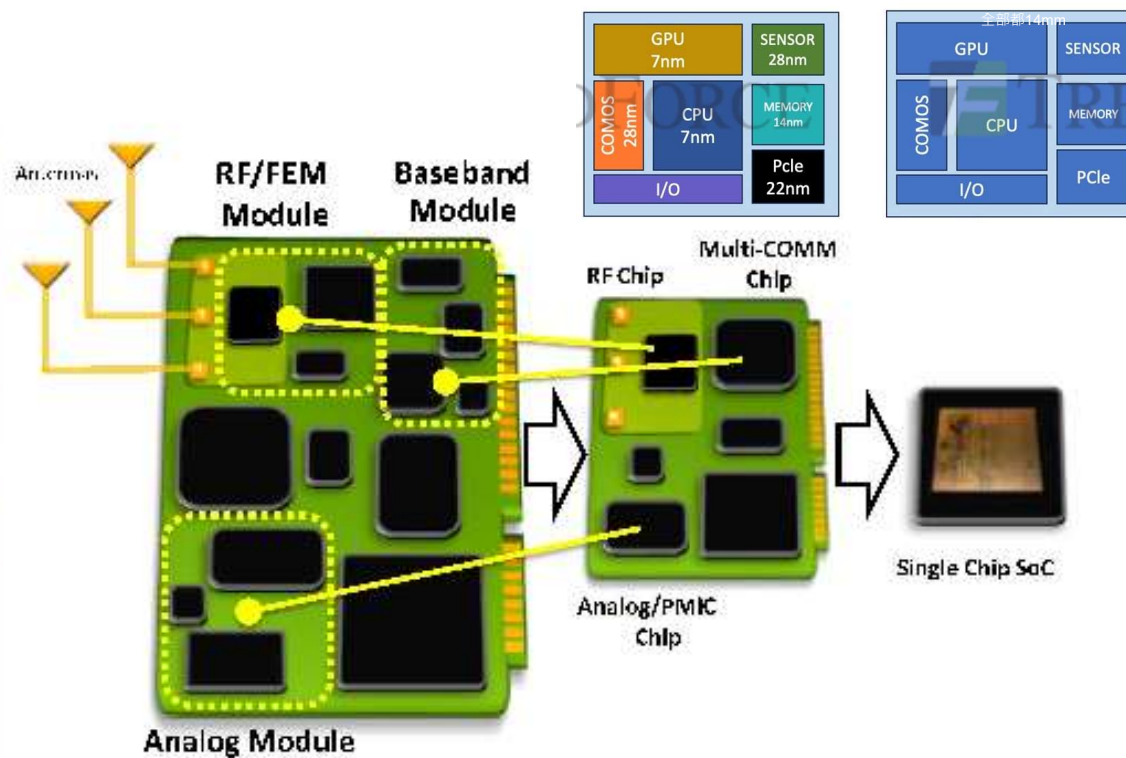


Apple M1 Chip Block Diagram

基礎概述—SoC

系統技術演進

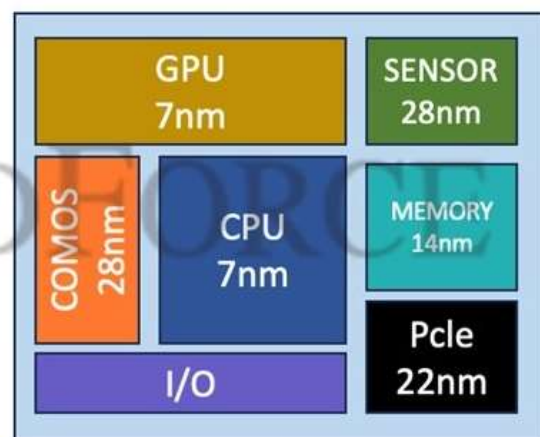
Processor	I/O Analog
CPU	USB
GPU	Audio Codec
	Video Enc
Memory	Video Dec
SRAM	Imaging
DRAM	Power Mgt
NAND Flash	
NOR Flash	
Sensor	Wireless
Accelerometer	Wi-Fi
Gyroscope	GSM
Compass	Edge
Wireless Mod.	3G
Baseband	4G/LTE
Transceiver	Blue Tooth
Power Amp	GPS
Front End	FM



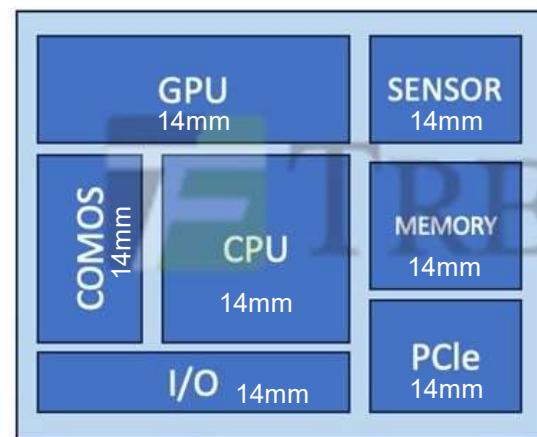
SoB → SiP → SoC

基礎概述—SoC

SiP與SoC差異



SiP



SoC

SoC與SiP比較表

	SoC (System On Chip)	SiP (System In Package)
設計成本	高	低
功耗	較低	較高
尺寸	較小	較大
良率	較低	較高
設計彈性	低	高
開發時間	長	短

Source: TrendForce, Aug., 2023

基礎概述—SoC

行動電話的演進如同SoC的進化史



黑金剛



Nokia 3310



Iphone 1



Iphone 14

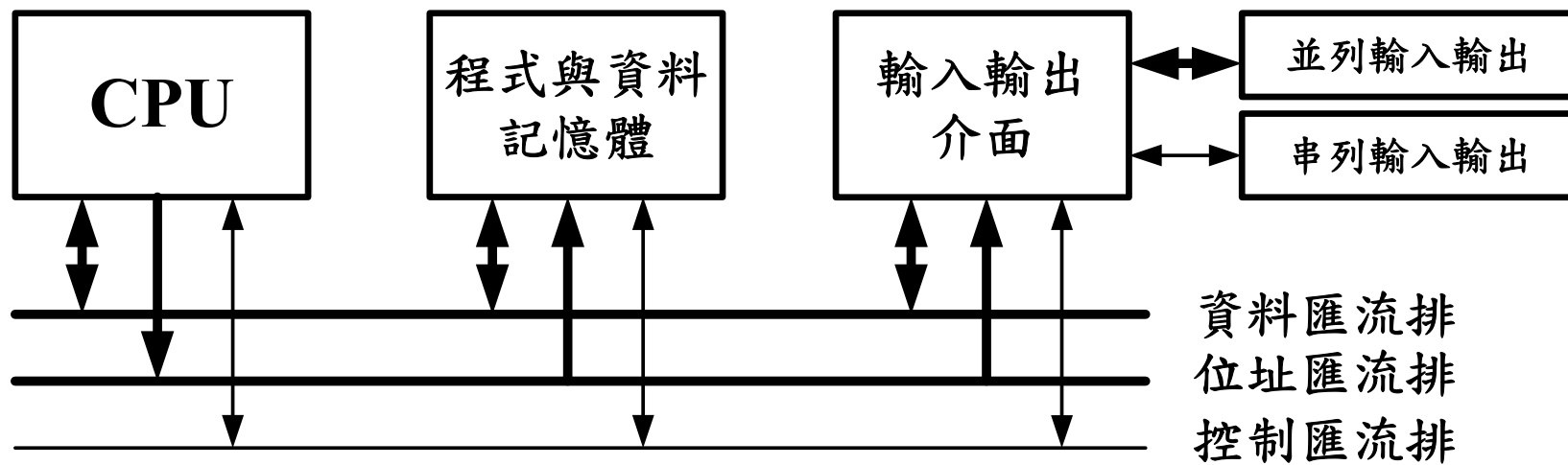
基礎概述—SoC

系統單晶片的優點

- 減少體積：以印刷電路板組合數個不同功能的積體電路，體積較大；如果整合成一個 SoC 晶片，則體積變小。
- 減少成本：需要封裝測試多顆積體電路，成本較高；如果整合成一個 SoC 晶片，只需要封裝測試一顆積體電路，成本較低。
- 降低耗電量同時提高運算速度：以印刷電路板組合數個不同功能的積體電路，電訊號必須在印刷電路板上傳送較長的距離才能進行運算，耗電量較高，運算速度較慢；如果整合成一個 SoC 晶片，電訊號在同一個積體電路內傳送較短的距離就能進行運算，耗電量較低，運算速度較快。
- 提升系統功能：將不同功能的積體電路整合成一個 SoC 晶片，體積較小，可以整合更多的「功能單元」，形成功能更強大的晶片。
- 達成自有IP。

基礎概述—MCU

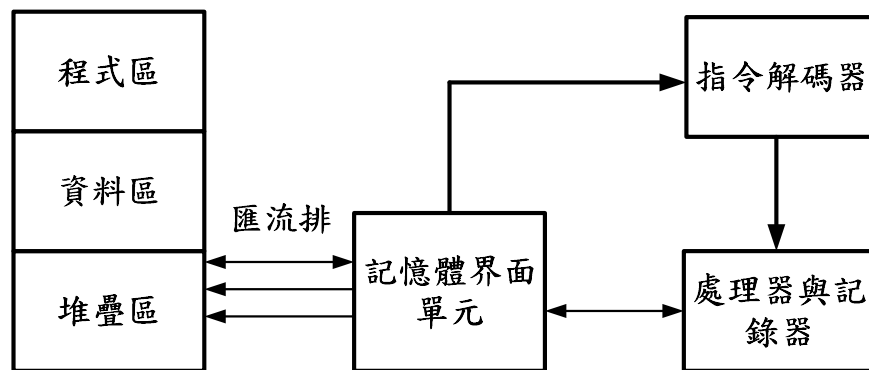
電腦汎指具有中央處理單元加上記憶體以及輸入輸出設備的系統



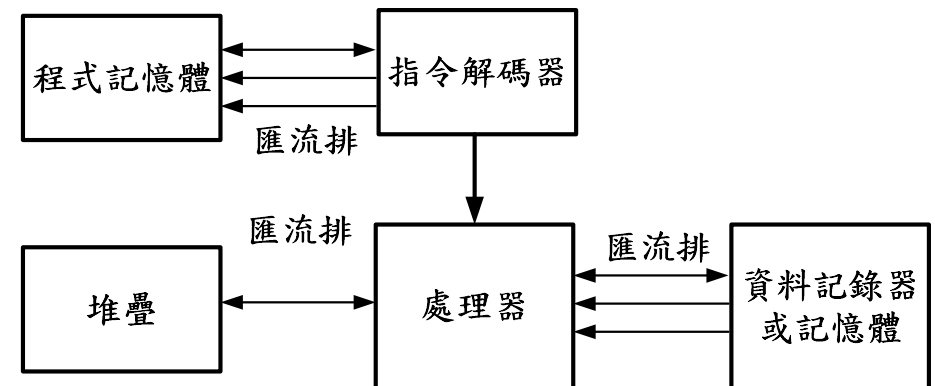
MCU基本架構圖

基礎概述—MCU

Von Neumann結構清楚地定義了嵌入式系統所必需的四個基本部分：一個中央處理器核心，程式記憶體（唯讀記憶體或者快閃記憶體）、資料記憶體（隨機存取記憶體）、一個或者更多的定時/計數器，還有用來與外圍裝置以及擴充資源進行通信的輸入/輸出埠——所有這些都被整合在單個積體電路晶片上。



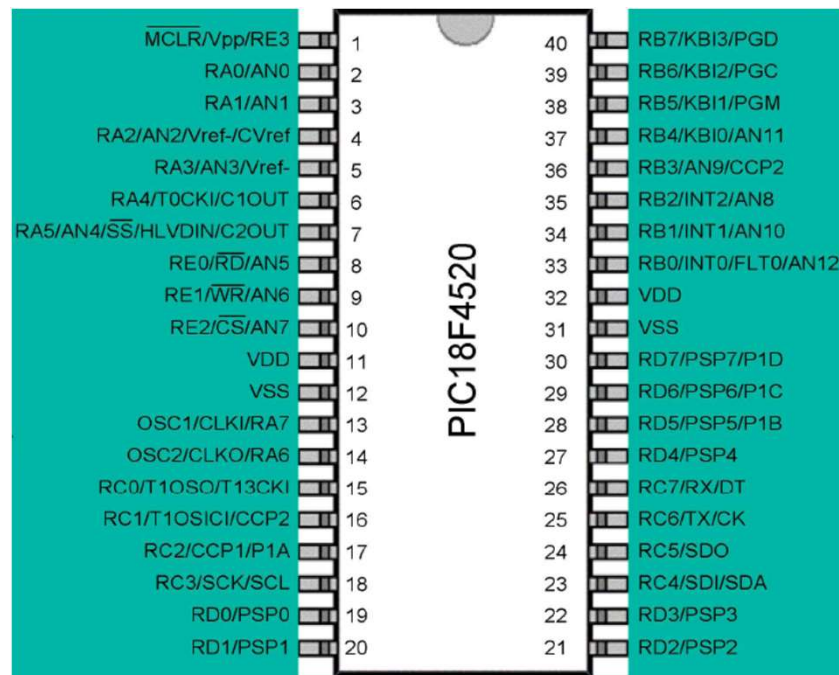
Von Neumann架構的微電腦系統方塊圖



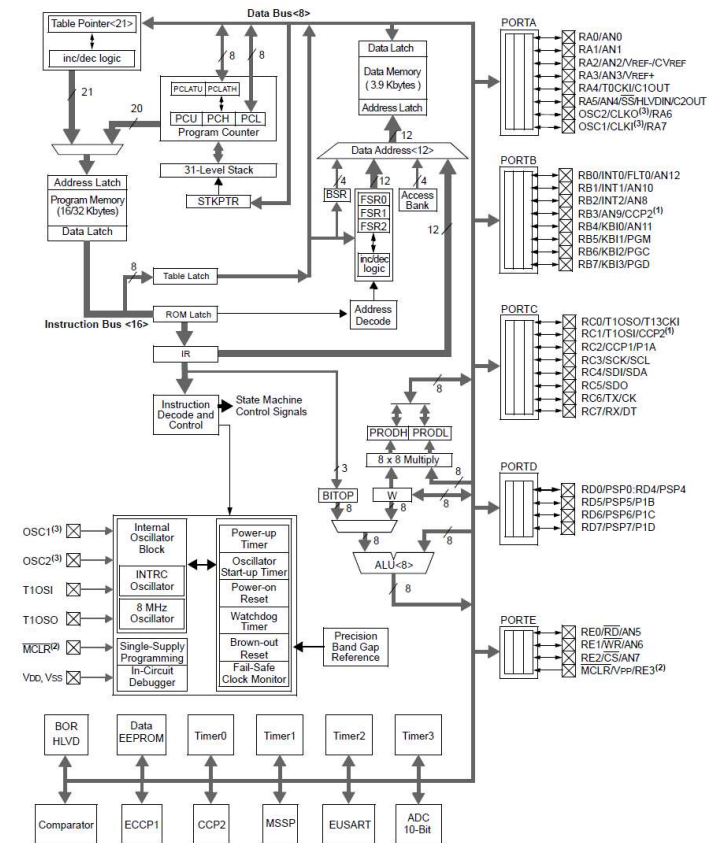
Harvard架構的微電腦系統方塊圖

基礎概述—MCU

8位元 MCU參考。



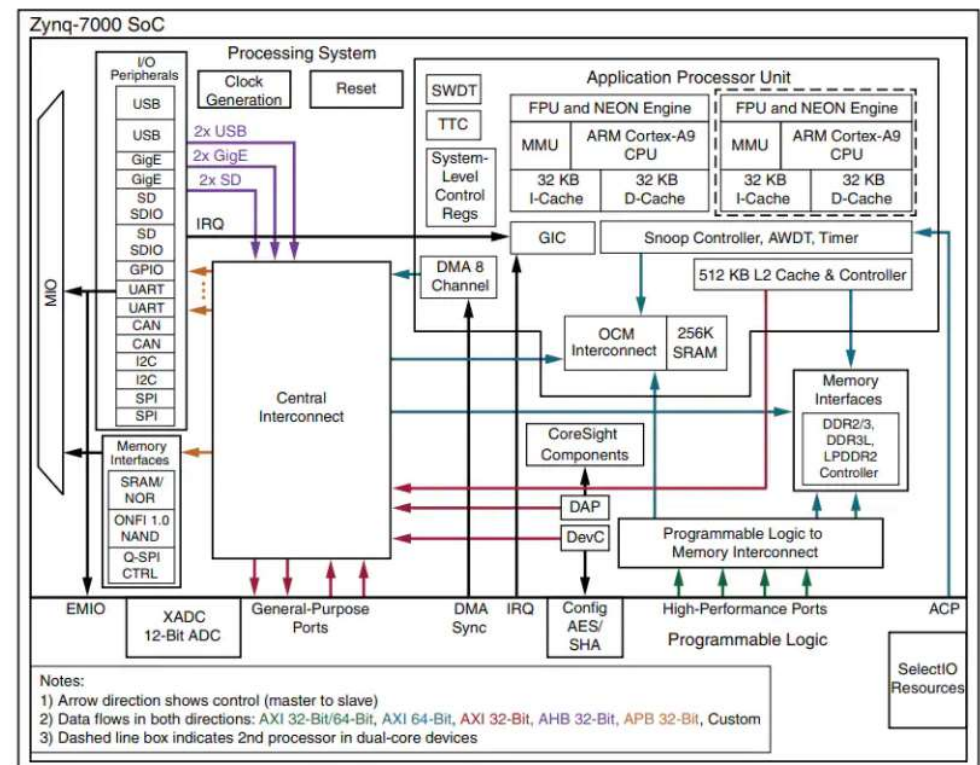
8位元MCU接腳



8位元MCU內部架構(哈佛結構)

基礎概述—MCU

ZYNQ參考。

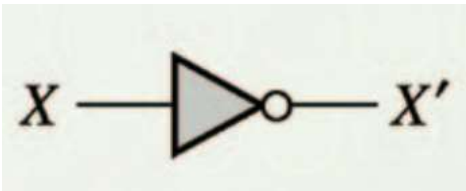


Zynq實體核心架構

基礎概述—FPGA

邏輯

NOT :



X	X'
1	0
0	1

AND :

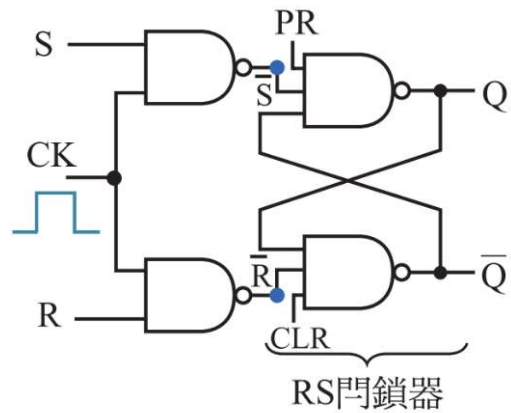
	AB	$C = A \cdot B$	
	00	0	不投幣，不按按鍵 → 沒有飲料
	01	0	不投幣，只按按鍵 → 沒有飲料
	10	0	投幣，不按按鍵 → 沒有飲料
	11	1	投幣，按下按鍵 → 拿到飲料

OR :

	AB	$C = A + B$	
	00	0	不吃飯，不吃麵 → 餓肚子
	01	1	吃麵不吃飯 → 飽 !!
	10	1	吃飯不吃麵 → 飽 !!
	11	1	吃飯也吃麵 → 飽 !!

基礎概述—FPGA

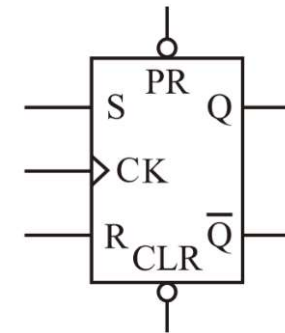
正反器



(a) 電路

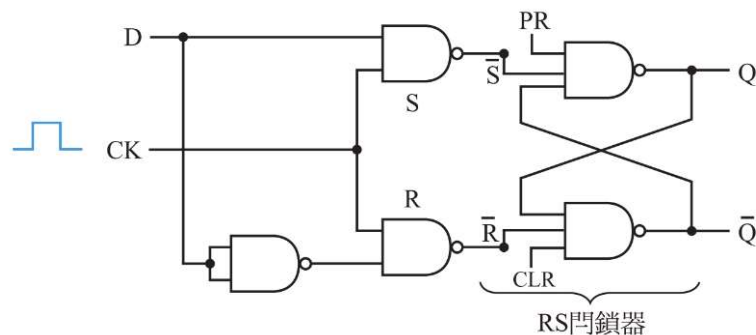
PR	CLR	CK	S	R	Q_{n+1}
0	0	×	×	×	?
0	1	×	×	×	1
1	0	×	×	×	0
1	1	↑	0	0	Q_n
1	1	↑	0	1	0
1	1	↑	1	0	1
1	1	↑	1	1	?

(b) 真值表



(c) 符號 (正緣觸發型)

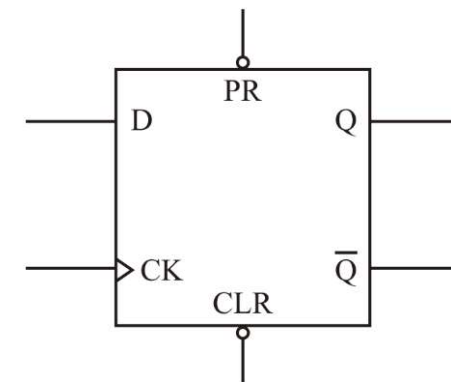
RS正反器



(a) 電路

PR	CLR	CK	D	Q_{n+1}
0	0	×	×	?
0	1	×	×	1
1	0	×	×	0
1	1	↑	0	0
1	1	↑	1	1

(b) 真值表



(c) 符號 (正緣觸發型)

D型正反器

基礎概述— FPGA

- 正反器在數位邏輯電路中，常見的主要功用有下列幾種：
 - 儲存資料(記憶體)
 - 計時/記數(計數器)
 - 改變資料的型態，如串列、並列的轉換
 - 組成控制電路控制其他元件

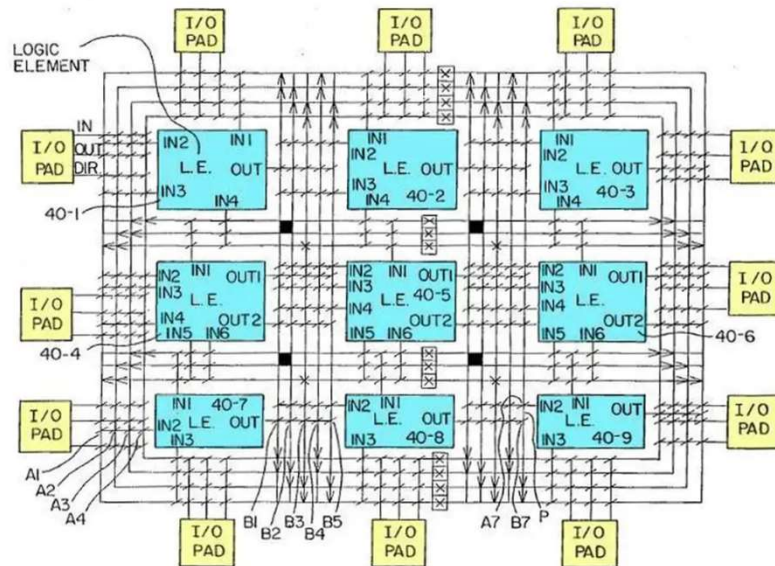
基礎概述—FPGA

什麼是 FPGA？

FPGA 是Field Programmable Gate Array (現場可規劃邏輯陣列)
含有可編輯元件的半導體設備
可供使用者程式化邏輯閘元件
以硬體描述語言HDL (Verilog或VHDL)所完成的電路



第一款FPGA晶片Xilinx XC2064

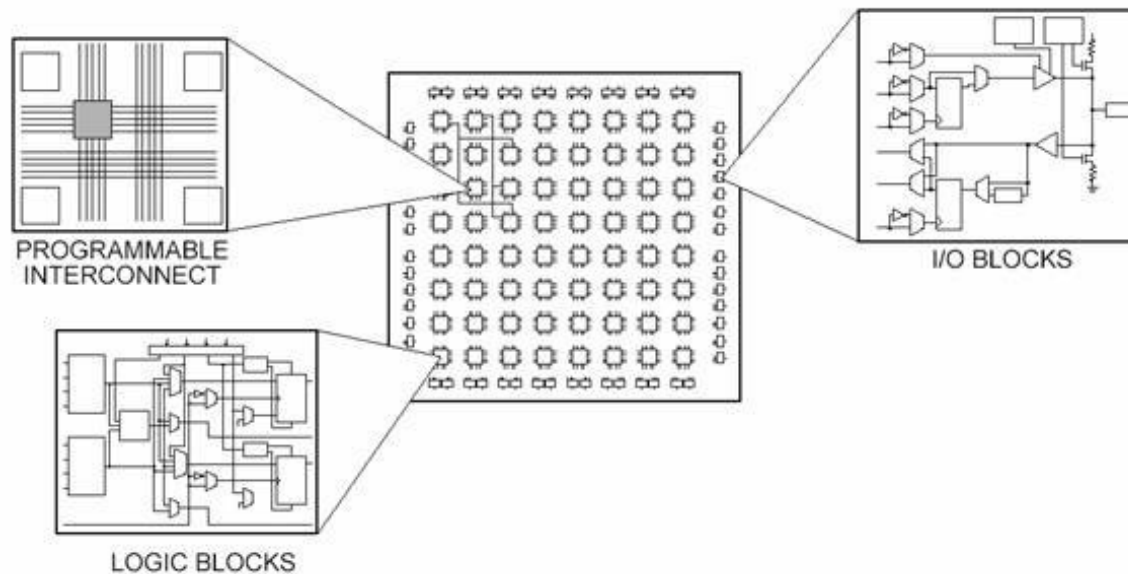


FPGA專利顯示通過互接的邏輯模組 (LE)

基礎概述—FPGA

FPGA結構

- 1.可配置邏輯模組CLB (Configurable Logic Block)
- 2.輸入輸出模組IOB (I/O Blocks)
- 3.內部連線 (Interconnect)



FPGA結構

基礎概述— FPGA

硬體描述語言 (hardware description language, HDL)

- VHDL
- Verilog

開發除錯工具

- ModelSim
- 線上開發 <https://www.edaplayground.com/>

例如：

全加器Verilog module

```
module FullAdd(a,b,CarryIn,Sum,CarryOut);  
  input a,b,CarryIn;  
  output Sum,CarryOut;  
  wire Sum,CarryOut;  
  
  assign {CarryOut,Sum} =a+b+CarryIn;  
endmodule
```

測試module

```
module test;  
  reg Sum;  
  reg CarryOut;  
  reg a;  
  reg b;  
  reg carryin;  
  FullAdd fulladd(a,b,carryin,Sum,CarryOut);  
  initial begin  
    $dumpfile("dump.vcd");  
    $dumpvars(1);  
    a=0;  
    b=1;  
    carryin = 1;  
    display;  
  end  
  
  task display;  
    #1 $display("Sum:%0h, CarryOut:%0h",Sum,CarryOut);  
  endtask  
endmodule
```

基礎概述—AP SoC

在做解決方案我們可能會考慮

FPGA—擴充功能強大，只有想不到沒有做不到。

MCU—計算、記憶體管理皆為強項，選對MCU上天堂。

但

只有小孩才做選擇

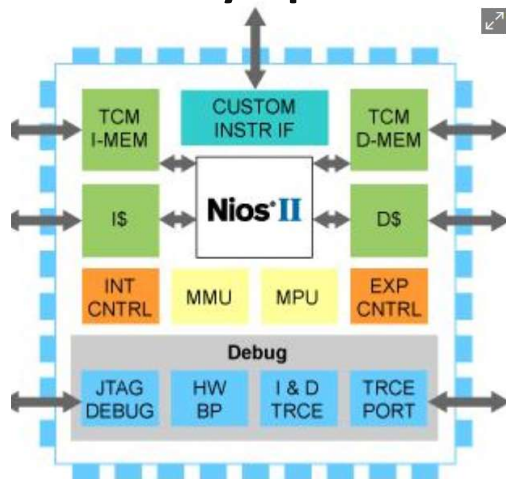
全加在一起就成為晶片界的瀨尿牛丸

All Programmable SoC (AP SoC)

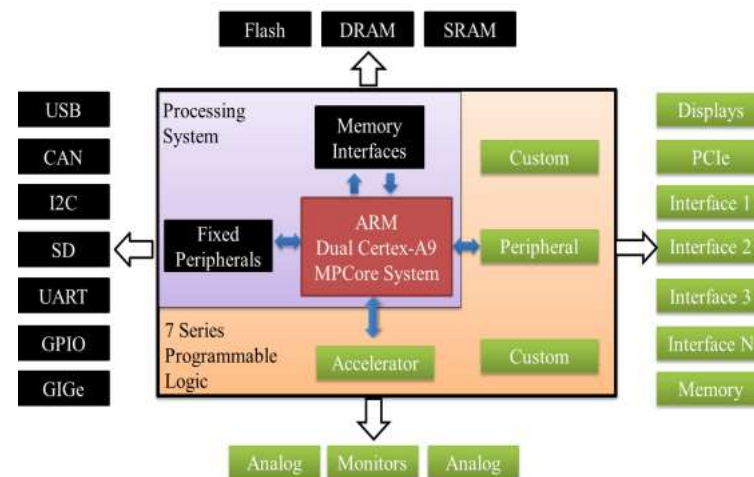
基礎概述—AP SoC

AP SoC兩大陣營

- **SOPC FPGA——軟核處理器**
 - Altera公司提出
 - 靈活性高、性能較低
 - 8051，RISC-V，Xilinx的 MicroBlaze，Altera的Nios-II
- **SoC FPGA——硬核處理器**
 - 內部的硬體電路上添加了硬核處理器不會消耗FPGA的邏輯資源
 - Zynq



Nios II軟核代表

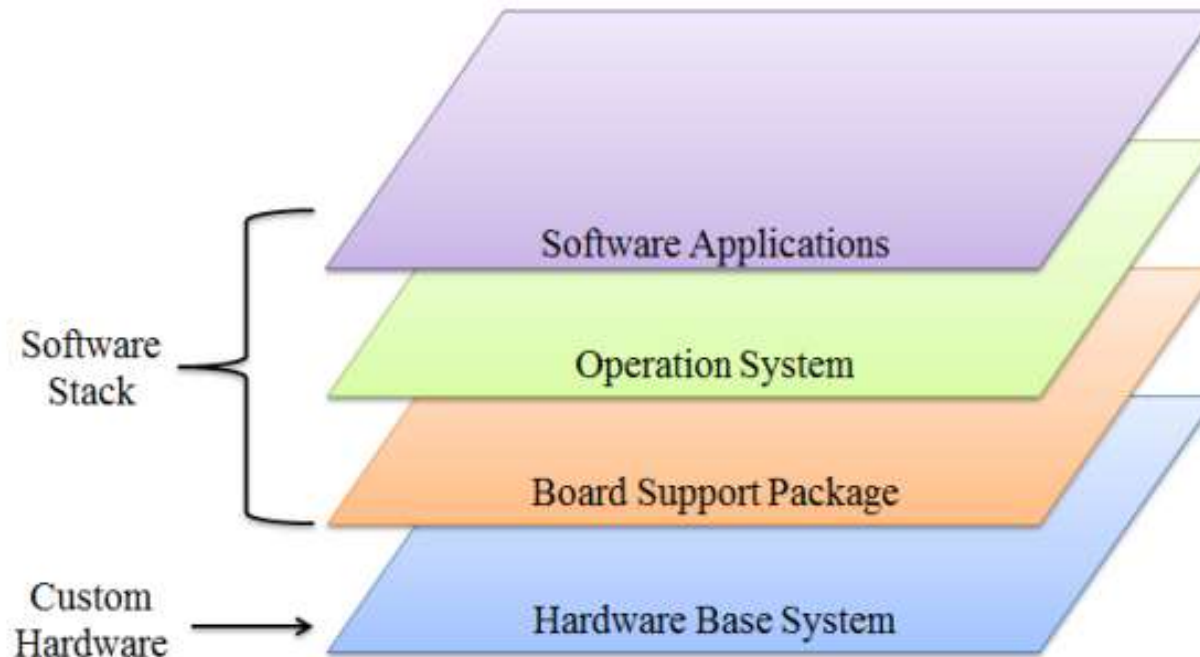


Zynq硬核代表

Zynq軟硬體開發入門

Zynq軟硬體開發入門— 說明與工具

由於Zynq硬體架構可以隨著設計者設計而改變，因此在軟體設計部分也不同於以往的嵌入式系統開發流程，軟體開發架構如圖所示，完成的硬體設計可以稱為硬體基礎系統，在硬體基礎平台上方必須存在著Board Support Package (BSP)做為Operating System與硬體之間的驅動，最上層的應用軟體可以透過作業系統呼叫設計者設定的硬體設備。

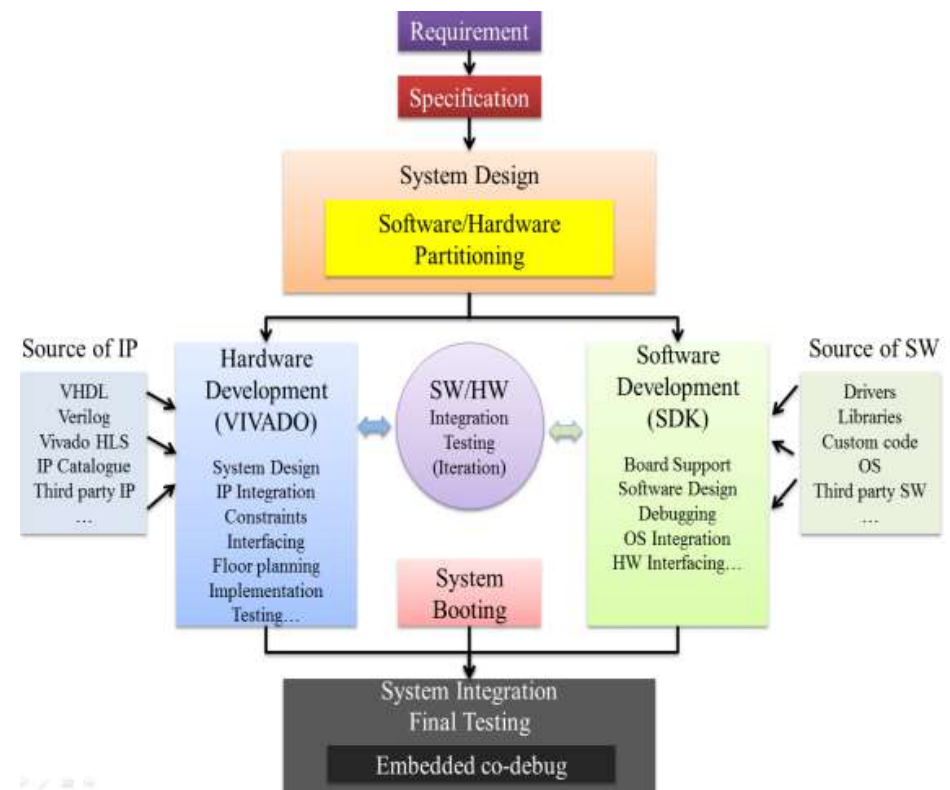


Zynq-7000 AP SoC 軟體開發架構

Zynq軟硬體開發入門—說明與工具

Zynq SoC發展出來的開發板具備了獨特的開發流程與工具，Zynq SoC開發設計開發流程可以分成五大步驟：

1. 開發前置作業
2. 系統設計
3. 硬體設計
4. 軟體設計與系統整合
5. 軟體設計與系統測試

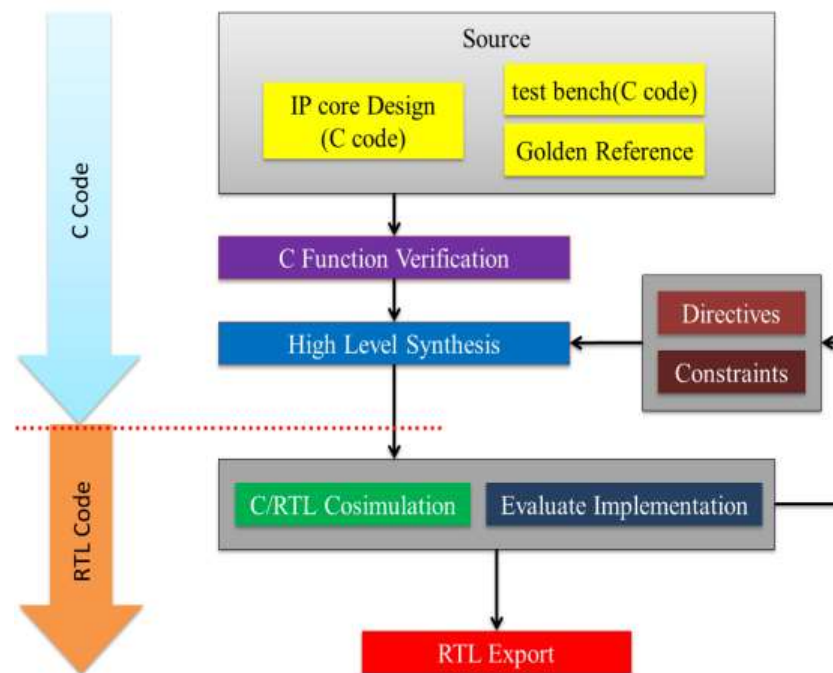


Zynq-7000 AP SoC 設計流程圖

Zynq軟硬體開發入門—說明與工具

Xilinx提供一套專門適用於Zynq-7000 AP SoC嵌入式開發板的開發工具，這套工具包含了

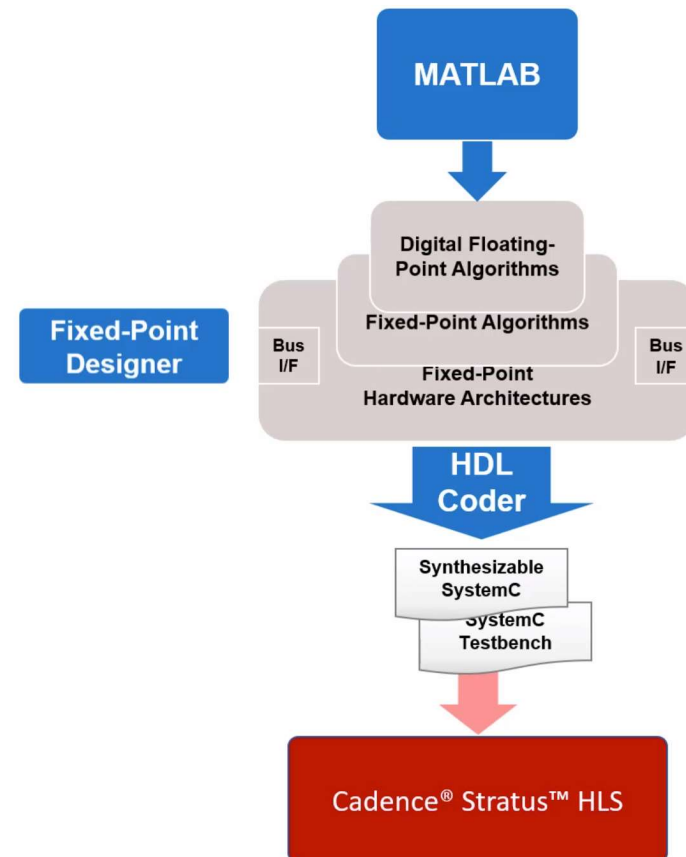
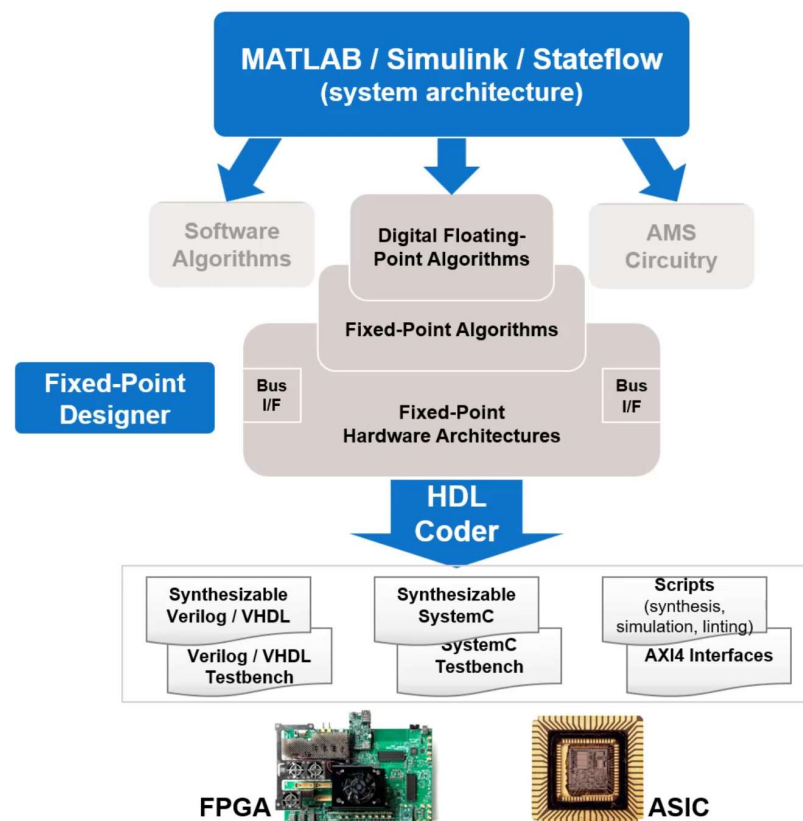
- Vivado High Level Synthesis(HLS)
- Vivado
- Xilinx Software Development Kit(SDK)



Vivado HLS 開發流程圖

Zynq軟硬體開發入門—說明與工具

High Level Synthesis概念



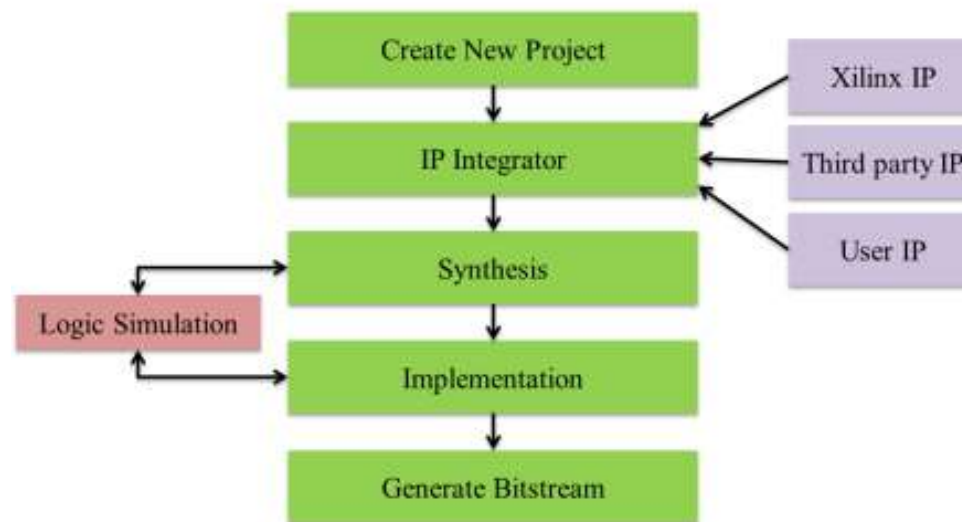
Zynq軟硬體開發入門—說明與工具

HDL 與 HLS 在一些案例的比較

最佳化手段	HDL	HLS
BRAM分割	○	○
單次遍歷	○	○
無縫數據交換	○	×
背靠背迭代	○	×
浮點除法最佳化	○	×
浮點累加最佳化	○	×

Zynq軟硬體開發入門—說明與工具

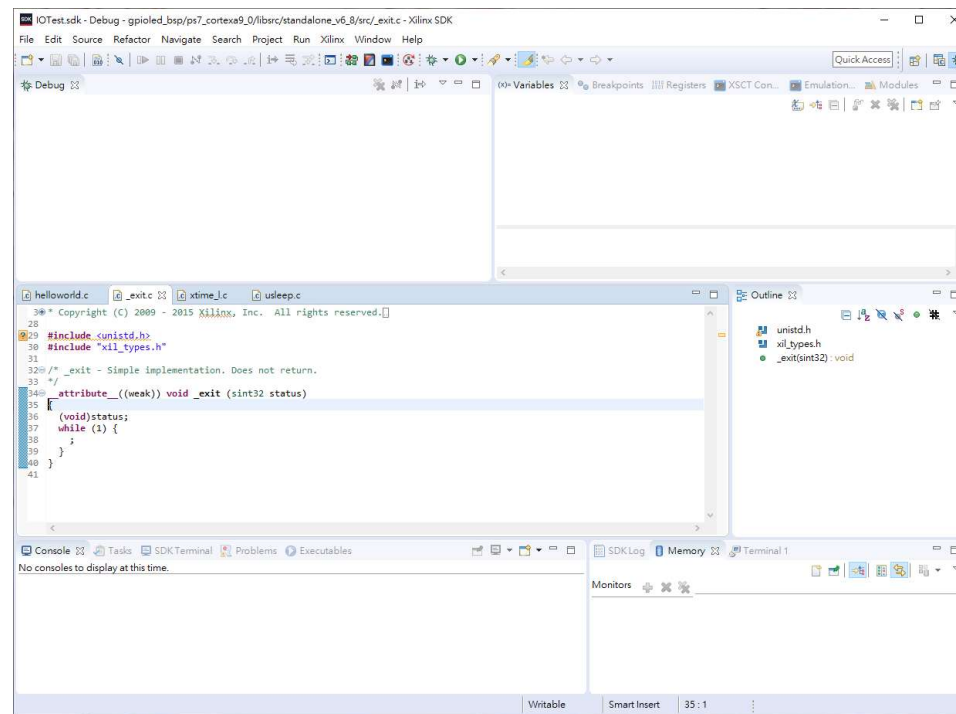
Vivado 是一個專門為Zynq SoC打造的系統級開發工具，Vivado適用於使用 7系列FPGA發展的開發平台，例如MicroBlaze、Zedboard、ZC702與ZC706等開發板。相較於傳統的register transfer level(RTL)-to-bitstream FPGA的開發流程，Vivado開發工具與開發流程使設計者可以更快速的完成系統設計、開發過程能更注重於功能上的創新、提高系統性能、降低系統功率消耗量。



Vivado 開發流程圖

Zynq軟體開發入門—說明與工具

Xilinx Software Development Kit (SDK)是第一個提供同質多核心與異質多核心處理器的軟體開發平台，專門用於Xilinx嵌入式開發板的應用開發。其中最大特點是在基礎的嵌入式系統開發中，**SDK**可以在許多關鍵的系統參數自動定義與配置，使設計者可以在最小的開發知識背景下進行開發與學習。



SDK開發界面

Zynq軟硬體開發範例

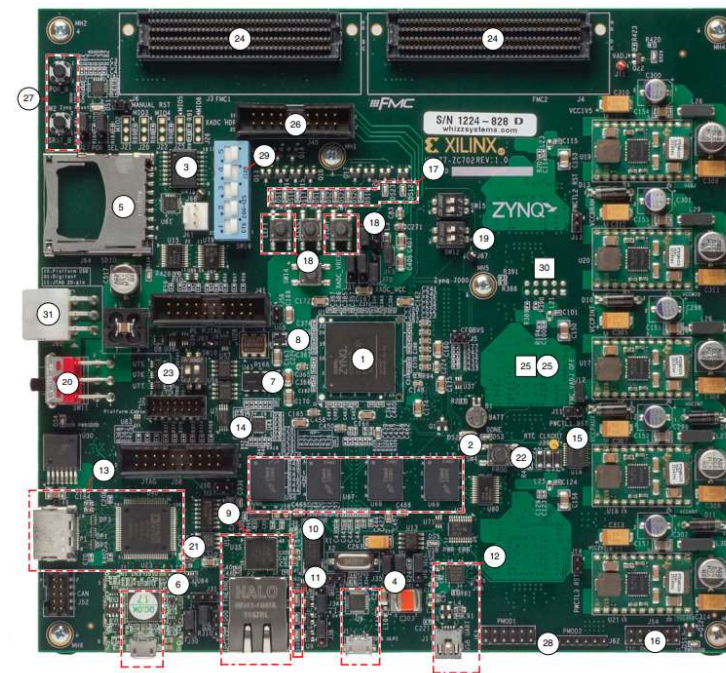
Zynq軟硬體開發入門— PL GPIO範例

範例：利用Zynq中的FPGA建立控制LED燈的GPIO，並使用Zynq中的ARM來使用控制LED的IP。

Table 1-23 lists the user LED connections to XC7Z020 SoC U1.

Table 1-23: User LED Connections to XC7Z020 SoC U1

XC7Z020 (U1) Pin	Net Name	I/O Standard	LED Reference Designator
E15	PMOD1_0	LVC MOS25	DS19
D15	PMOD1_1	LVC MOS25	DS20
W17	PMOD1_2	LVC MOS25	DS21
W5	PMOD1_3	LVC MOS25	DS22
V7	PMOD2_0	LVC MOS25	DS18
W10	PMOD2_1	LVC MOS25	DS17
P18	PMOD2_2	LVC MOS25	DS16
P17	PMOD2_3	LVC MOS25	DS15
Bank 501 G7	PS_LED1	N/A	DS23
Bank 500 E5	PS_MIO8_LED0	N/A	DS12

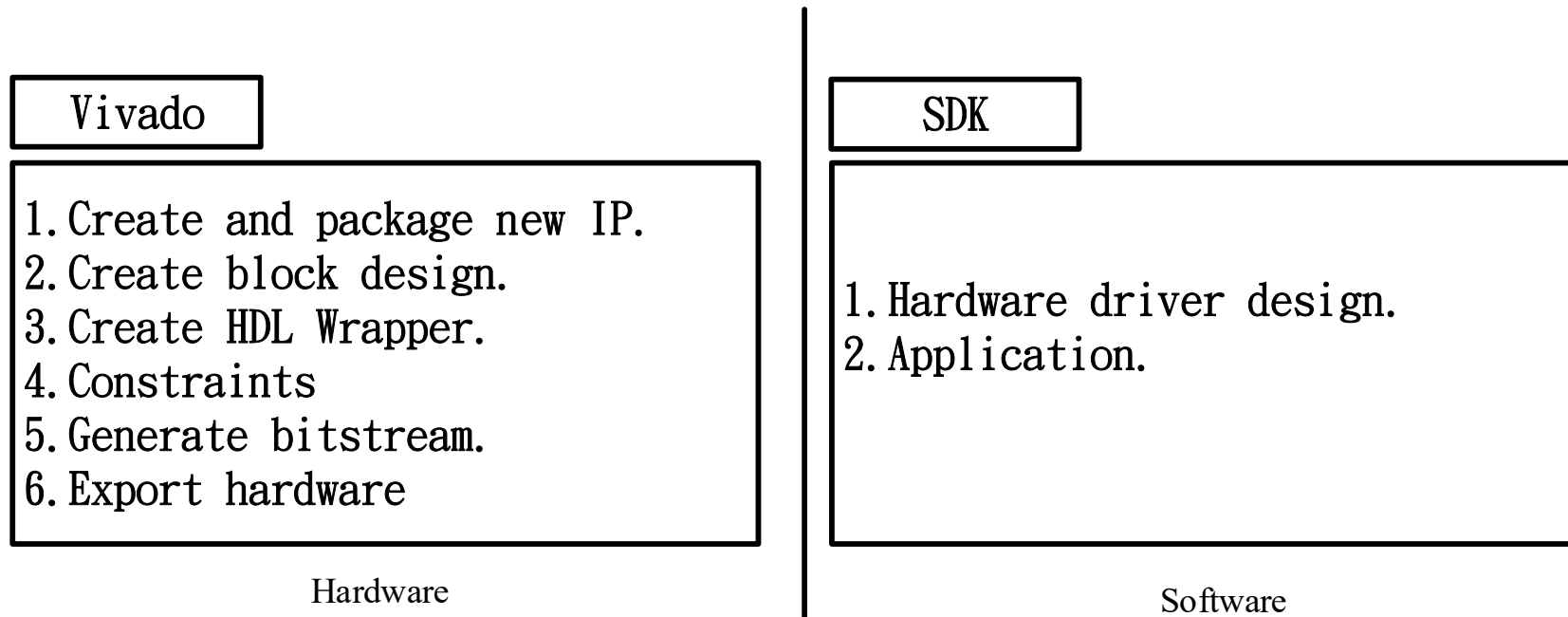


UG650_c1_02_032013

Zynq ZC702開發板

Zynq軟硬體開發入門— PL GPIO範例

步驟與開發環境



實作步驟與相應開發環境

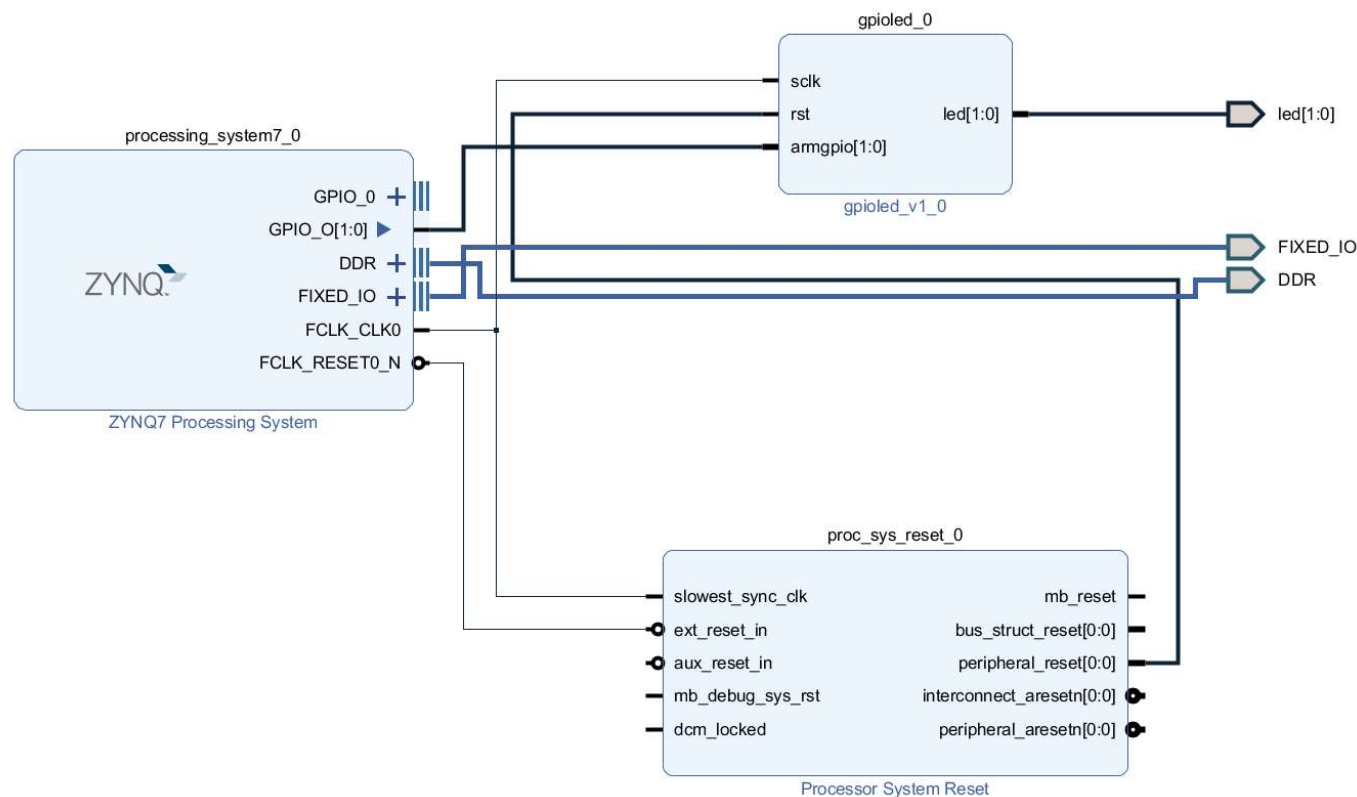
Zynq軟硬體開發入門— PL GPIO範例

Hardware Build步驟1: Create and package new IP.

```
1 module gpioled(  
2     input wire sclk,    //from zynq arm io 10ns  
3     input wire rst,  
4     input wire [1:0] armgpio, // from zynq arm pin  
5     output wire [1:0] led    // connect fpga pin  
6 );  
7     reg [26:0] div_cnt;  
8  
9     reg [2:0] div_cnt_ctrl;  
10    reg pulse_led;  
11  
12    assign led[0] = armgpio[0];  
13    assign led[1] = pulse_led;  
14  
15  
16    always @(posedge sclk) begin  
17        div_cnt_ctrl <= {div_cnt_ctrl[1:0],armgpio[1]};  
18    end  
19  
20    always @(posedge sclk) begin  
21        if(rst == 1'b1) begin  
22            div_cnt <= 'd0;  
23        end  
24        else if(div_cnt_ctrl[2] == 1'b1) begin  
25            if (div_cnt[26] == 1'b1) begin  
26                div_cnt <= 'd0;  
27            end  
28            else begin  
29                div_cnt <= div_cnt + 1'b1;  
30            end  
31        end  
32        else begin  
33            div_cnt <= 'd0;  
34        end  
35    end  
36  
37    always @(posedge sclk) begin  
38        if(rst == 1'b1) begin  
39            pulse_led <= 1'b0;  
40        end  
41        else if(div_cnt[26] == 1'b1) begin  
42            pulse_led <= ~pulse_led;  
43        end  
44    end  
45 endmodule  
46
```

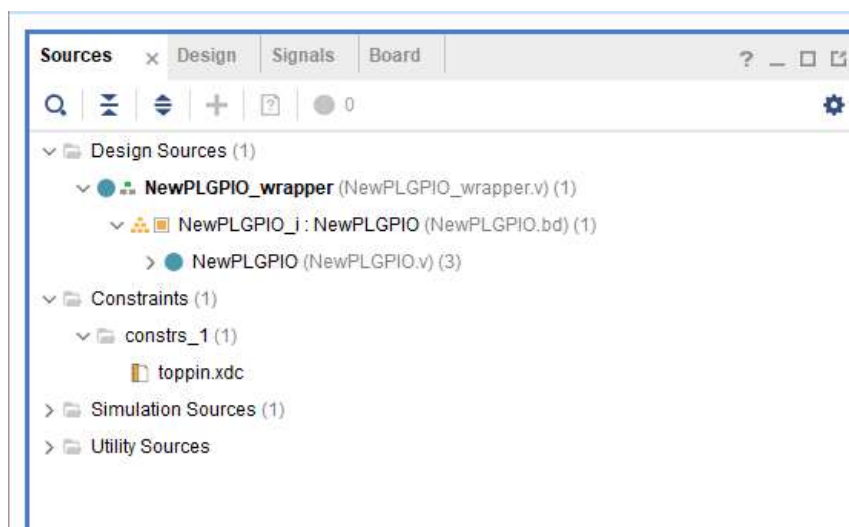
Zynq軟硬體開發入門— PL GPIO範例

Hardware Build步驟2: Create block design.



Zynq軟硬體開發入門— PL GPIO範例

Hardware Build步驟3&4: Create HDL wrapper & Constraints



toppin.xdc

```
set_property PACKAGE_PIN E15 [get_ports led[0]]
set_property PACKAGE_PIN D15 [get_ports led[1]]
set_property IOSTANDARD LVCMOS25 [get_ports led[*]]
```

Zynq軟硬體開發入門— PL GPIO範例

Hardware Build步驟5&6: Generate bitstream & Export hardware

/ado > NTUTPLGPIO > NTUTPLGPIO.sdk				
名稱	修改日期	類型	大小	
.metadata	2023/11/12 下午 05:42	檔案資料夾		
.sdk	2023/11/12 下午 05:51	檔案資料夾		
NTUTgpioled	2023/11/12 下午 05:44	檔案資料夾		
NTUTgpioled_bsp	2023/11/12 下午 05:43	檔案資料夾		
NTUTPLGPIO_wrapper_hw_platform_0	2023/11/12 下午 05:42	檔案資料夾		
RemoteSystemsTempFiles	2023/11/12 下午 05:42	檔案資料夾		
webtalk	2023/11/12 下午 09:54	檔案資料夾		
NTUTPLGPIO_wrapper.hdf	2023/11/12 下午 05:34	HDF 檔案	363 KB	
SDK.log	2023/11/12 下午 09:54	文字文件	15 KB	

Zynq軟硬體開發入門— PL GPIO範例

Software Build步驟1: Hardware driver design.

初始化

```
int initGpio()
{
    int status;
    GpioPsCfgPtr = XGpioPs_LookupConfig(GPIO_DEV_ID);
    status = XGpioPs_CfgInitialize(&GpioPs, GpioPsCfgPtr, GpioPsCfgPtr->BaseAddr);
    if(status != XST_SUCCESS)
    {
        return status;
    }
    XGpioPs_SetDirectionPin(&GpioPs, LED0, 0X01);
    XGpioPs_SetDirectionPin(&GpioPs, LED1, 0X01);
    XGpioPs_SetOutputEnablePin(&GpioPs, LED0, 0X01);
    XGpioPs_SetOutputEnablePin(&GpioPs, LED1, 0X01);
    return status;
}
```

GPIO設定

```
XGpioPs_WritePin(&GpioPs, LED1, 0X00);
```

Zynq軟硬體開發入門— PL GPIO範例

Software Build步驟2: Application.

```
int main()
{
    init_platform();

    int i;

    initGpio();
    XGpioPs_WritePin(&GpioPs, LED1, 0X00);
    for(i=0; i<8; i++)
    {
        XGpioPs_WritePin(&GpioPs, LED0, 0X01);
        usleep(1000000);
        XGpioPs_WritePin(&GpioPs, LED0, 0X00);
        usleep(1000000);
    }
    print("Hello World\n\r");

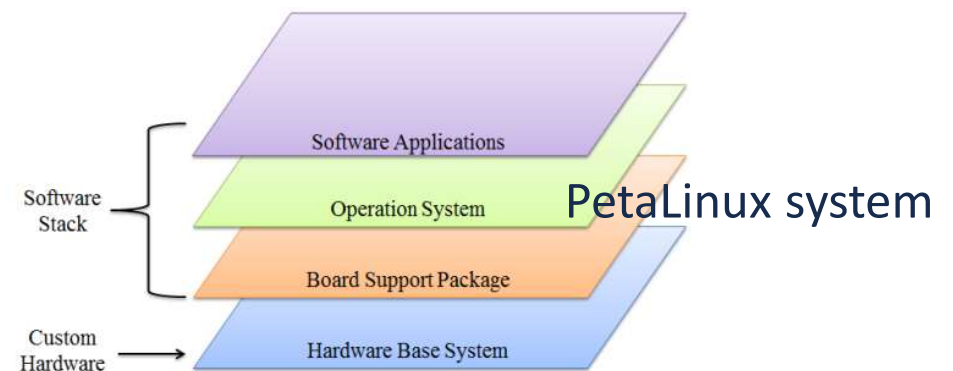
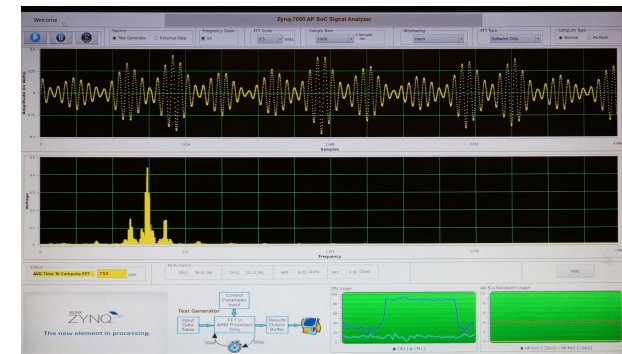
    cleanup_platform();
    return 0;
}
```

Zynq軟硬體開發入門—FFT範例

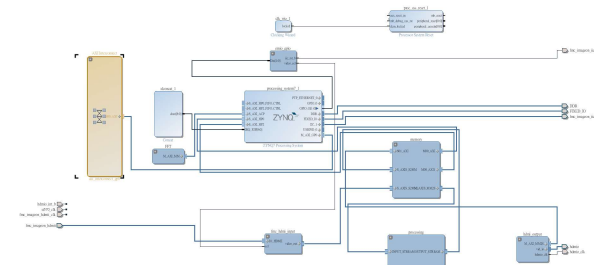
範例：FFT訊號處理加速

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842102/Zynq-7000+AP+SoC+Spectrum+Analyzer+part+4+-+Accelerating+Software+-+Building+and+Running+an+FFT+Tech+Tip+2014.3>

FFT應用範例



ZYNQ Base TRD 2014.2



Zynq軟硬體開發入門—FFT範例

範例：FFT訊號處理加速

三種架構比較

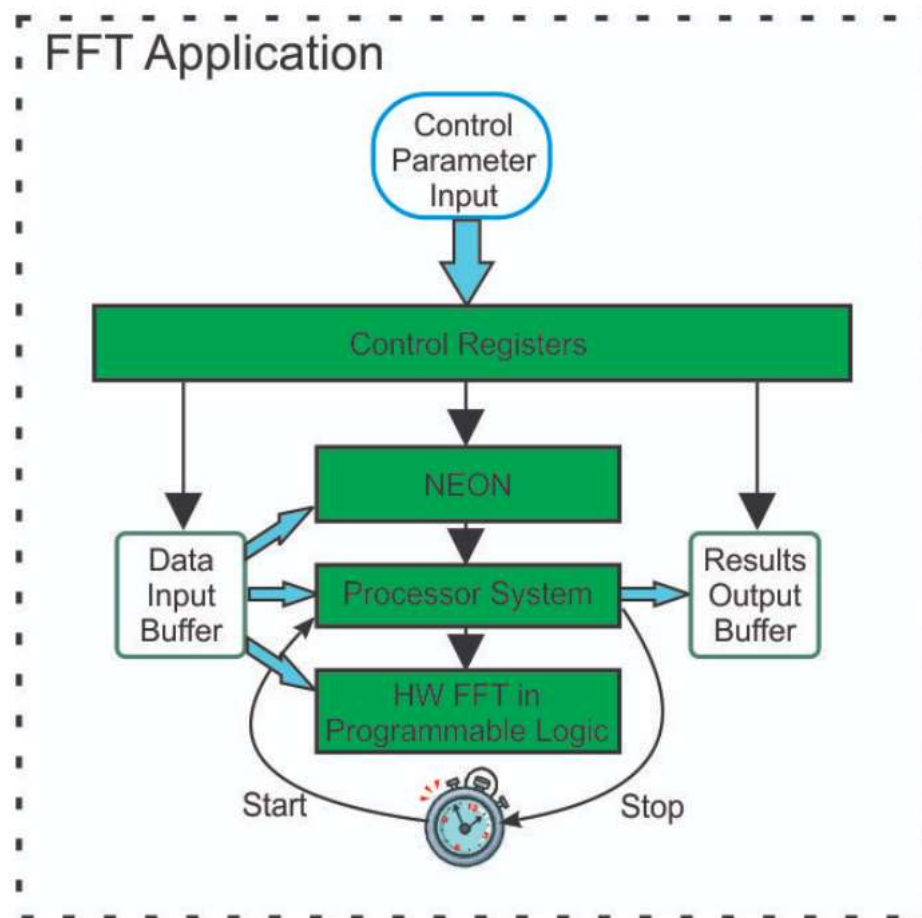
1. 使用 ARM軟體計算
2. 使用NEON軟體計算
3. 使用FPGA硬體計算

4096點FFT計算分別在不同架構下所花費時間

ARM alone - 1325 usec

NEON SIMD engine - 885 usec

Hardware FFT - 221 usec

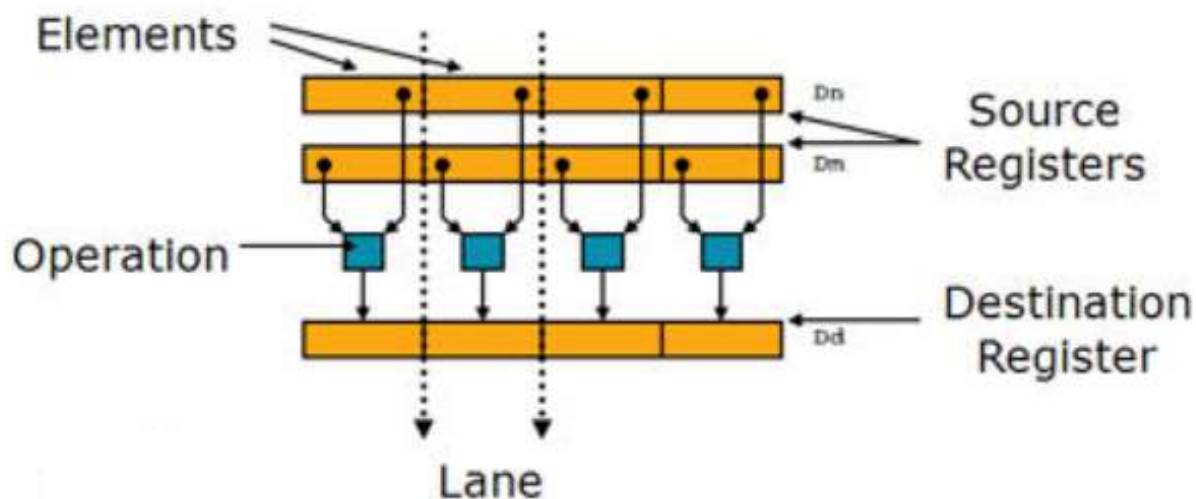


FFT應用範例

Zynq軟硬體開發入門—FFT範例

NEON指令集

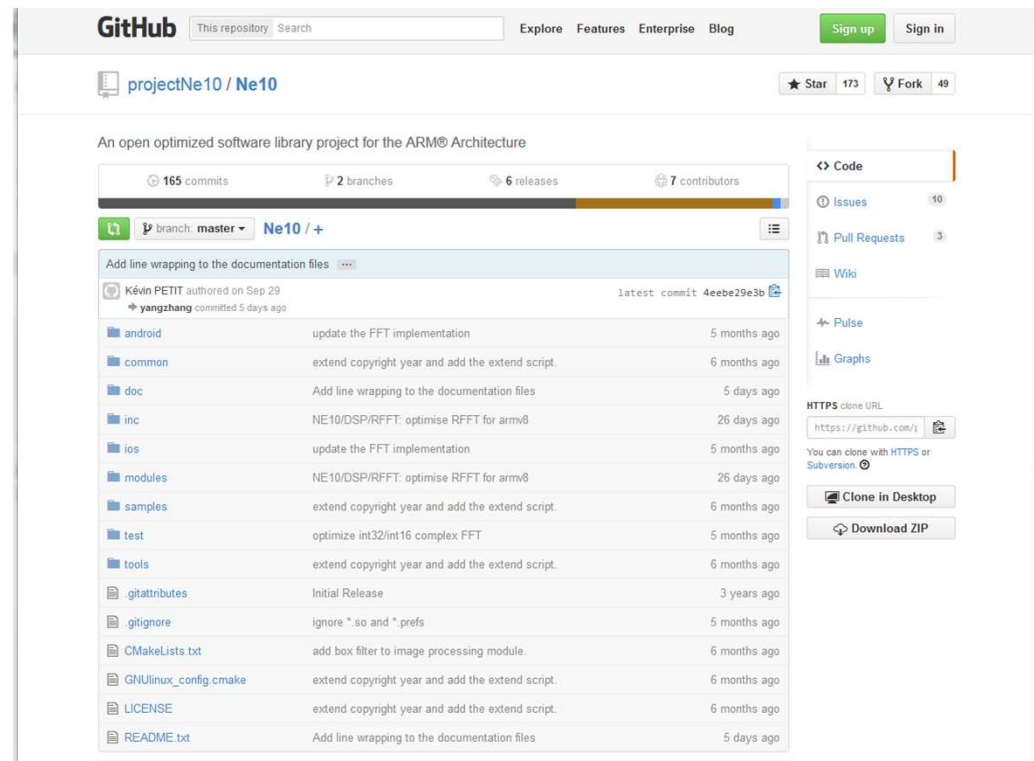
NEON是ARM架構下的一種優化指令集，主要為實現SIMD (Single instruction multiple data,單指令多數數據流)



NEON架構

Zynq軟硬體開發入門—FFT範例

NEON指令集



NEON指令集

Zynq軟硬體開發入門—FFT範例

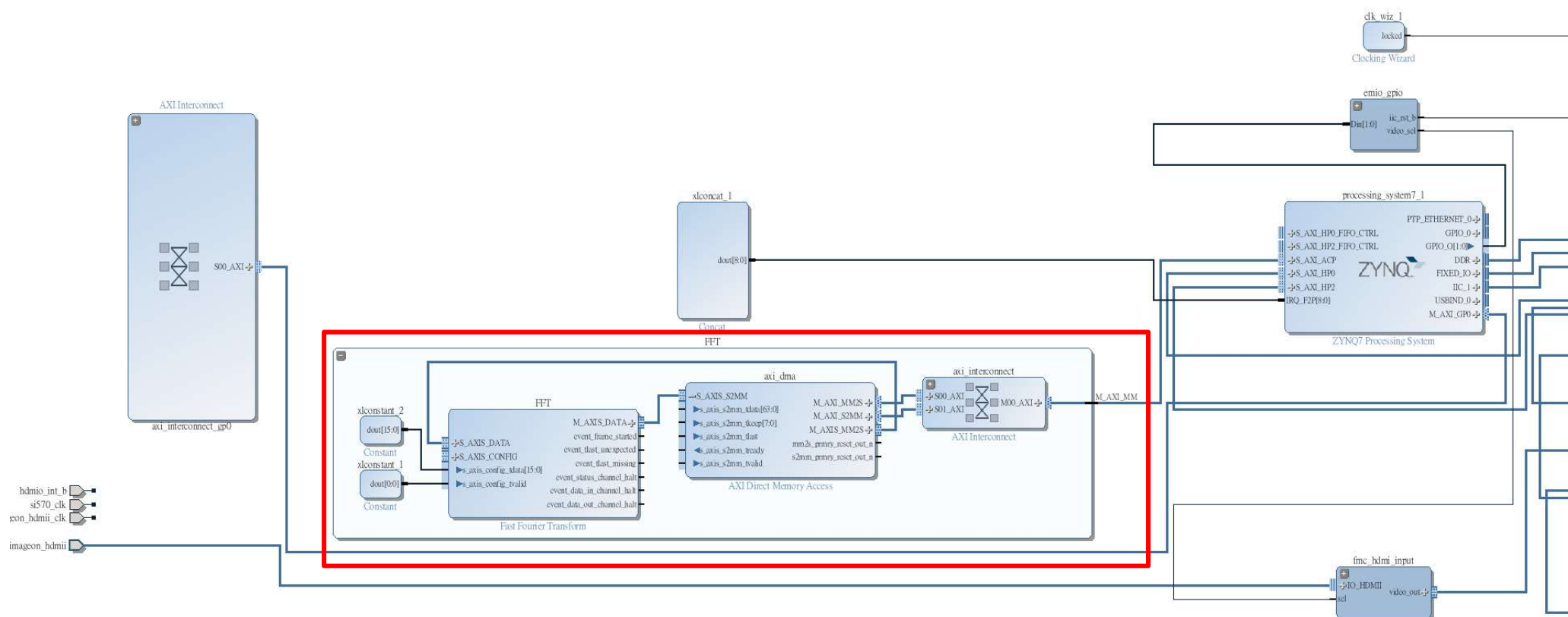
NEON指令集

Math Functions		Vector and Matrix algebra functions
	Vector Add	Vector Add for Float data - 2 2D, 3D, 4D vectors
	Matrix Add	Matrix Add for Float data - 2X2 to 4X4
	Vector Sub	Vector Sub for Float data - 2 2D, 3D, 4D vectors
	Vector RSBC	Vector sub from a constant - 2D, 3D, 4D vectors
	Matrix Sub	Matrix Sub for Float data - 2X2 to 4X4
	Vector Multiply	Vector Multiply by constant or other Vector
	Vector Multiply - Accumulator	Vector Multiply / Accumulate for Float data
	Matrix Multiply	Matrix Multiply for Float data - 2X2 to 4X4
	Matrix Vector Multiply	Multiply Matrix by Vector X2, X3 or X4
	Vector Div	Vector Divide by constant or other Vector
	Matrix Div	Matrix Divide for Float data - 2X2 to 4X4
	Vector Setc	Sets Vector to a constant or Constant Vector
	Vector Len	Length of 2D, 3D and 4D vectors
	Vector Normalize	Normalizes array of Vectors - 2D, 3D or 4D
	Vector Abs	Absolute value of Input Vector(s)
	Vector Dot	Performs Dot product of two vectors - 2D, 3D, 4D
	Vector Cross	Cross product of two vectors
	Matrix Determinant	Determinant for 2X2, 3X3 or 4X4 Matrix
	Matrix Invertible	Find Invertible of Matrix for 2X2, 3X3 or 4X4
	Matrix Transpose	Find transpose of Matrix for 2X2, 3X3 or 4X4
	Matrix Identity	Find Identity Matrix for 2X2, 3X3 or 4X4

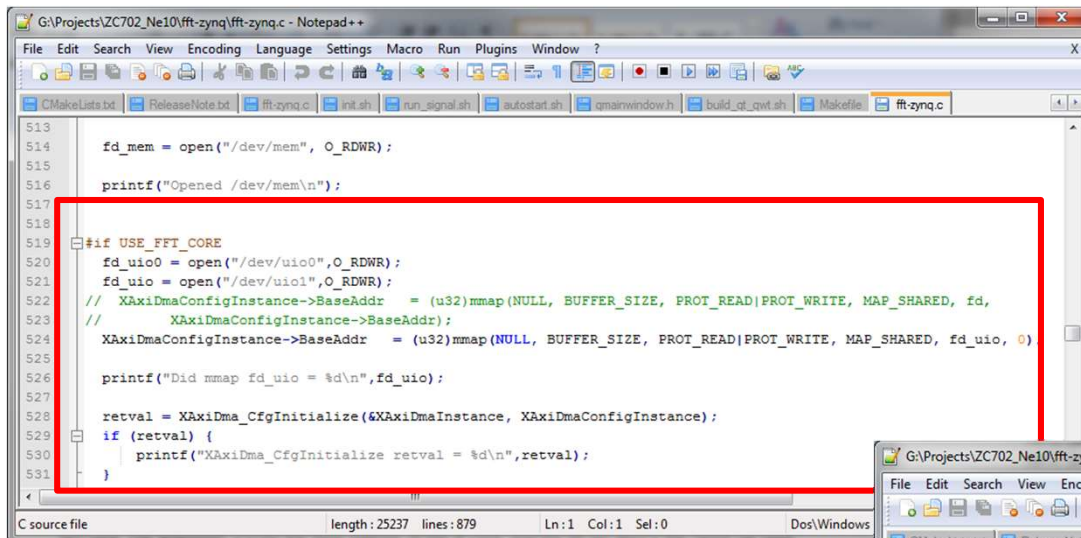
Signal Processing		
	Complex FFT	CCFFT/CIFFT for Radix 4 binary lengths from 4 to 32768
	FIR Filters	Finite Impulse Response Filter on Float data
	FIR Decimator	Optimized FIR and Decimator function
	FIR Interpolator	FIR and Interpolator using Polyphase structure
	FIR Lattice Filters	FIR Lattice using feedforward structure
	FIR Sparse Filters	Sparse FIR for simulating reflections, etc.
	IIR Lattice Filters	IIR Filter with feedforward and feedback
	Real FFT	Real FFT and Real IFFT for Float data
Image Processing		
	Hresize	Interpolation of horizontal data
	Vresize	Interpolation of vertical data
	Img_rotate	Rotate image by 90 degrees
	Box Filter	
Physics	AABB	Compute AABB for a Polygon
Sample Functions		Code samples for calling NEON, etc.

Zynq軟硬體開發入門—FFT範例

硬體 FFT 架構

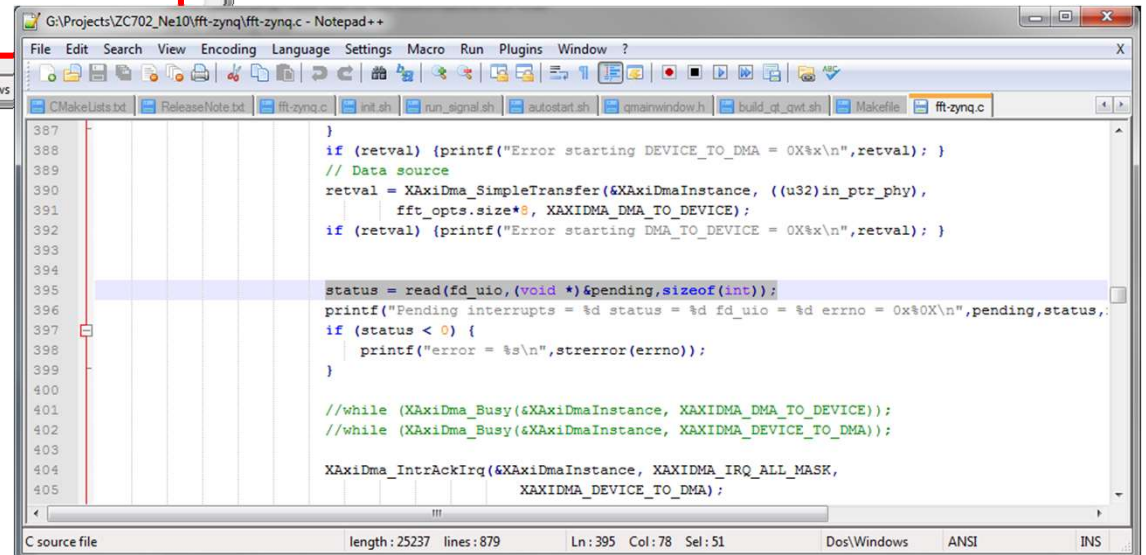


Zynq軟硬體開發入門—FFT範例



```
513
514 fd_mem = open("/dev/mem", O_RDWR);
515
516 printf("Opened /dev/mem\n");
517
518
519 #if USE FFT_CORE
520 fd_uio0 = open("/dev/uio0", O_RDWR);
521 fd_uio = open("/dev/uio1", O_RDWR);
522 // XAxiDmaConfigInstance->BaseAddr = (u32)mmap(NULL, BUFFER_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd,
523 // XAxiDmaConfigInstance->BaseAddr);
524 XAxiDmaConfigInstance->BaseAddr = (u32)mmap(NULL, BUFFER_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd_uio, 0);
525
526 printf("Did mmap fd_uio = %d\n", fd_uio);
527
528 retval = XAxiDma_CfgInitialize(&XAxiDmaInstance, XAxiDmaConfigInstance);
529 if (retval) {
530     printf("XAxiDma_CfgInitialize retval = %d\n", retval);
531 }
```

DMA裝置開啟並初始化



```
387
388 if (retval) {printf("Error starting DEVICE_TO_DMA = 0x%x\n", retval); }
389 // Data source
390 retval = XAxiDma_SimpleTransfer(&XAxiDmaInstance, ((u32)in_ptr_phy),
391     fft_opts.size*8, XAXIDMA_DMA_TO_DEVICE);
392 if (retval) {printf("Error starting DMA_TO_DEVICE = 0x%x\n", retval); }
393
394
395 status = read(fd_uio, (void *)&pending, sizeof(int));
396 printf("Pending interrupts = %d status = %d fd_uio = %d errno = 0x%x\n", pending, status,
397     if (status < 0) {
398         printf("error = %s\n", strerror(errno));
399     }
400
401 //while (XAxiDma_Busy(&XAxiDmaInstance, XAXIDMA_DMA_TO_DEVICE));
402 //while (XAxiDma_Busy(&XAxiDmaInstance, XAXIDMA_DEVICE_TO_DMA));
403
404 XAxiDma_IntrAckIrq(&XAxiDmaInstance, XAXIDMA_IRQ_ALL_MASK,
405     XAXIDMA_DEVICE_TO_DMA);
```

啟動中斷，讀取DMA處理完的資料

討論