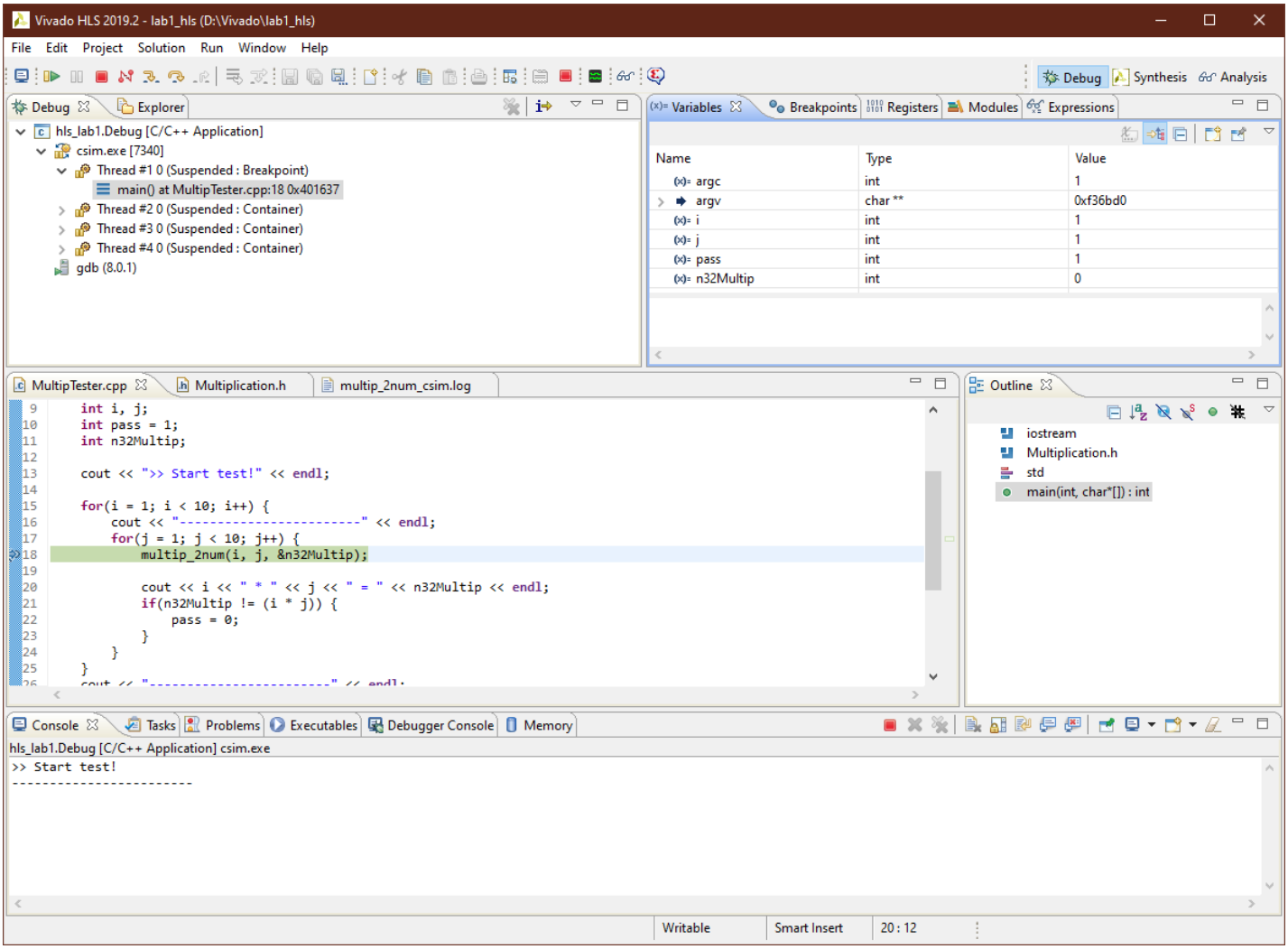
HLS Lab 1 - Multiplier

B07902143 資訊三 陳正康

HLS - C Simulation

用C/C++ 寫完design之後，用C/C++寫的testbench做初步simulation
跑的是sequential的c code，跟software邏輯相同，可以印出debug message，也可以用類似software的debugger下breakpoint，看變數等等



HLS - Synthesis

這個階段是做HLS的合成，會產生multip_2num,
multip_2num_AXILiteS_s_axi 和 multip_2num_mul_3bkb三個verilog module file

multip_2num_mul_3bkb是乘法的邏輯

```
assign p = buff2;
assign tmp_product = a_reg0 * b_reg0;
always @ (posedge clk) begin
    if (ce) begin
        a_reg0 <= a;
        b_reg0 <= b;
        buff0 <= tmp_product;
        buff1 <= buff0;
        buff2 <= buff1;
    end
end
endmodule
```

multip_2num_AXILiteS_s_axi定義了AXI Lite的介面，用於ZYNQ 7020 Processing System和Programming Logic主要的溝通，如控制訊號和kernel function參數傳遞等。可以看到n32in1等參數還有return value pn32resout

```
//------------------------Register logic-----------------
assign n32In1 = int_n32In1;
assign n32In2 = int_n32In2;
// int_n32In1[31:0]
always @(posedge ACLK) begin
    if (ARESET)
        int_n32In1[31:0] <= 0;
    else if (ACLK_EN) begin
        if (w_hs && waddr == ADDR_N32IN1_DATA_0)
            int_n32In1[31:0] <= (WDATA[31:0] & wmask) | (int_n32In1[31:0] & ~wmask);
    end
end

// int_n32In2[31:0]
always @(posedge ACLK) begin
    if (ARESET)
        int_n32In2[31:0] <= 0;
    else if (ACLK_EN) begin
        if (w_hs && waddr == ADDR_N32IN2_DATA_0)
            int_n32In2[31:0] <= (WDATA[31:0] & wmask) | (int_n32In2[31:0] & ~wmask);
    end
end

// int_pn32ResOut
always @(posedge ACLK) begin
    if (ARESET)
        int_pn32ResOut <= 0;
    else if (ACLK_EN) begin
        if (pn32ResOut_ap_vld)
            int_pn32ResOut <= pn32ResOut;
    end
end

// int_pn32ResOut_ap_vld
always @(posedge ACLK) begin
    if (ARESET)
        int_pn32ResOut_ap_vld <= 1'b0;
    else if (ACLK_EN) begin
        if (pn32ResOut_ap_vld)
            int_pn32ResOut_ap_vld <= 1'b1;
        else if (ar_hs && raddr == ADDR_PN32RESOUT_CTRL)
            int_pn32ResOut_ap_vld <= 1'b0; // clear on read
    end
end
```

multip_2num是兩者的wrapper，加上一些狀態機的維護

```
always @ (*) begin
    if ((1'b1 == ap_CS_fsm_state7)) begin
        pn32ResOut_ap_vld = 1'b1;
    end else begin
        pn32ResOut_ap_vld = 1'b0;
    end
end

always @ (*) begin
    case (ap_CS_fsm)
        ap_ST_fsm_state1 : begin
            ap_NS_fsm = ap_ST_fsm_state2;
        end
        ap_ST_fsm_state2 : begin
            ap_NS_fsm = ap_ST_fsm_state3;
        end
        ap_ST_fsm_state3 : begin
            ap_NS_fsm = ap_ST_fsm_state4;
        end
        ap_ST_fsm_state4 : begin
            ap_NS_fsm = ap_ST_fsm_state5;
        end
        ap_ST_fsm_state5 : begin
            ap_NS_fsm = ap_ST_fsm_state6;
        end
        ap_ST_fsm_state6 : begin
            ap_NS_fsm = ap_ST_fsm_state7;
        end
        ap_ST_fsm_state7 : begin
            ap_NS_fsm = ap_ST_fsm_state1;
        end
        default : begin
            ap_NS_fsm = 'bx;
        end
    endcase
end
```

以下是synthesis report

performance

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.950 ns	0.63 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
6	6	30.000 ns	30.000 ns	6	6	none

Detail

- Instance
- Loop

utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	0	3	359	233	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	41	-
Register	-	-	103	-	-
Total	0	3	462	274	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0

Detail

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
multip_2num_AXILiteS_s_axi_U	multip_2num_AXILiteS_s_axi	0	0	144	232	0
multip_2num_mul_3bkb_U1	multip_2num_mul_3bkb	0	3	215	1	0
Total	2	0	3	359	233	0

DSP48E

Memory

FIFO

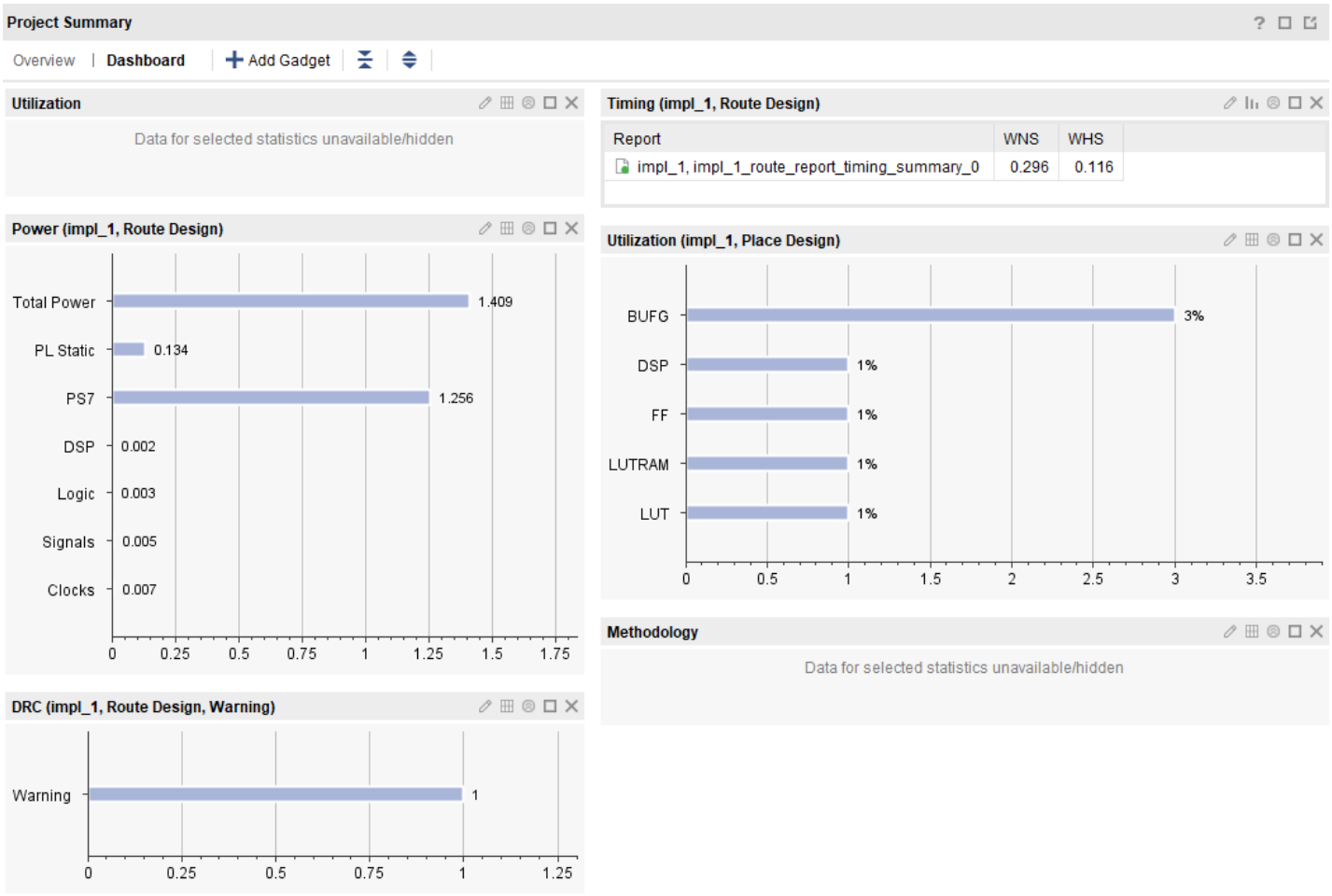
Expression

Multiplexer

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	41	8	1	8
Total	41	8	1	8

Register

Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	7	0	7	0
mul_ln18_reg_59	32	0	32	0
n32ln1_read_reg_54	32	0	32	0
n32ln2_read_reg_49	32	0	32	0
Total	103	0	103	0


Project Summary
Overview
Dashboard
Add Gadget
Utilization
Data for selected statistics unavailable/hidden
Timing (impl_1, Route Design)
Report
WNS
WHS
impl_1, impl_1_route_report_timing_summary_0
0.296
0.116
Power (impl_1, Route Design)
Total Power
1.409
PL Static
0.134
PS7
1.256
DSP
0.002
Logic
0.003
Signals
0.005
Clocks
0.007
0
0.25
0.5
0.75
1
1.25
1.5
1.75
Utilization (impl_1, Place Design)
BUFG
3%
DSP
1%
FF
1%
LUTRAM
1%
LUT
1%
0
0.5
1
1.5
2
2.5
3
3.5
Methodology
Data for selected statistics unavailable/hidden
DRC (impl_1, Route Design, Warning)
Warning
1
0
0.25
0.5
0.75
1
1.25

interface

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_AXILiteS_AWVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_AWREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_AWADDR	in	6	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WDATA	in	32	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WSTRB	in	4	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARADDR	in	6	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RVALID	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RREADY	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RDATA	out	32	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RRESP	out	2	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BVALID	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BREADY	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BRESP	out	2	s_axi	AXILiteS	pointer
ap_clk	in	1	ap_ctrl_none	multip_2num	return value
ap_rst_n	in	1	ap_ctrl_none	multip_2num	return value

HLS - Co-simulation

把c和rtl一起做simulation並測試正確性

waveform

在vivado hls裡按solution->open wave viewer

觀察波形，每個乘法花了20ns(4個cycle)才完成(pn32ResOut_al_vld變成1)，再1個cycle才換下一個input，共5個cycle
而report中overall的latency是6個cycle。還不清楚原因

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
6	6	30.000 ns	30.000 ns	6	6	none

HLS - Export RTL

把rtl design匯出成ip

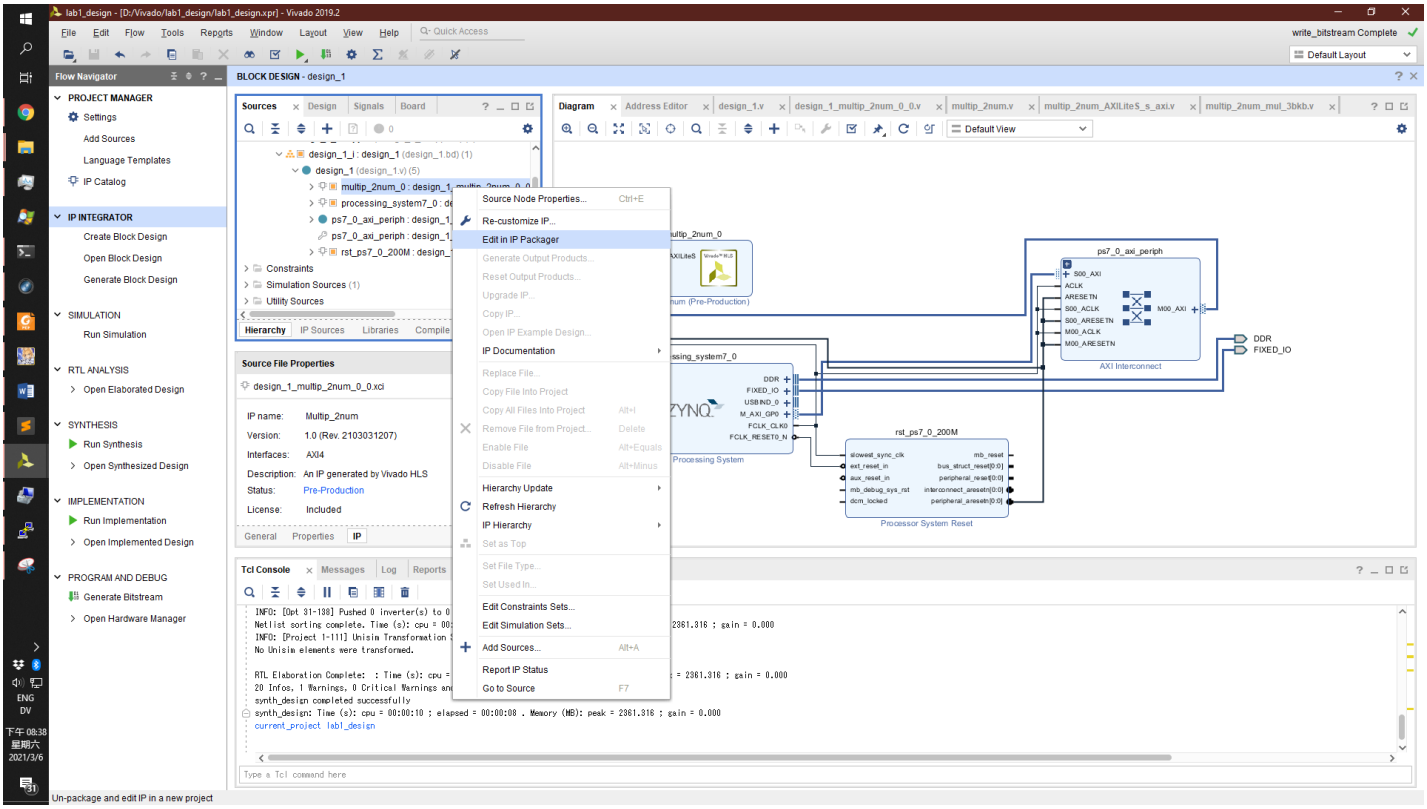
Block Design

先import剛匯出的ip

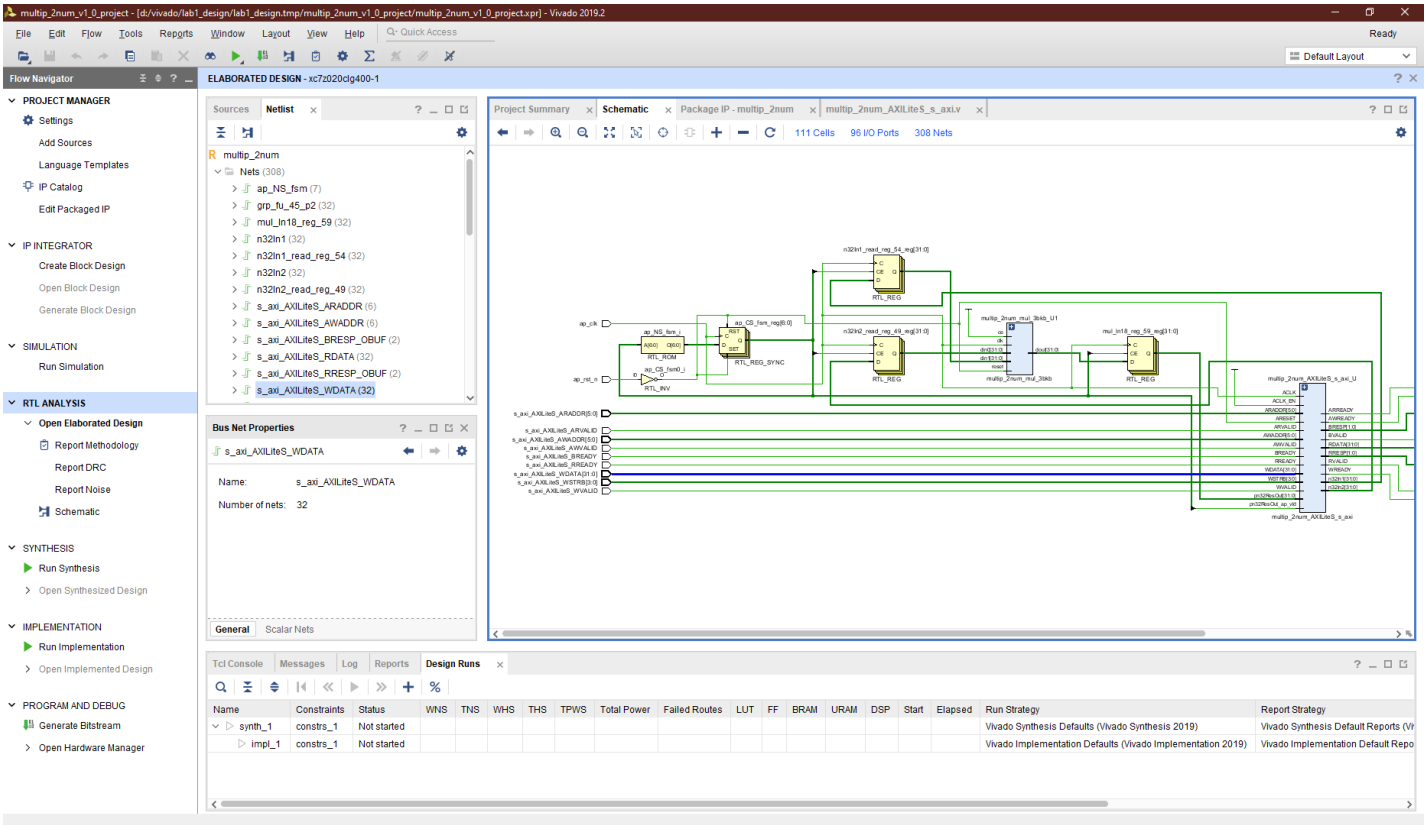
把zynq processing system (PS)和programming logic (PL, programmed with our mulitplier RTL kernel function)做適當接線

PS的GP(general purpose port, master)要接到kernel function的AXI Lite(slave)，基本上都自動接好了

Schematic of Synthesised HLS Design



對匯入的HLS design按下Edit in IP Packager，再開啟elaborated design，就可以看到如下的RTL schematic



稍微觀察一下，匯入的hls design為multip_2num，包含2個模組：multip_2num_AXILiteS_s_axi和multip_2num_mul_3bkb_U1

multip_2num_AXILiteS_s_axi是hls模組的interface，有s_axi_AXILiteS_XXXX開頭的signal，依據input address把data送到我們乘法器模組對應的input，如in1, in2等，或把乘法器模組的output傳出來

而multip_2num_mul_3bkb_U1是我們的乘法器模組，有幾個ce(enable)是由fsm(finite state machine)控制

Upload Hardware Handoff and Bitstream to PYNQ

hwh是一些介面描述，而bitstream是要program進fpga的邏輯，這些傳上pynq板子後就可以在python code裡include它們並進行互動

PYNQ Board Info

IP Address: 192.168.1.57 (offical says 192.168.2.99)
Python package `pynq` version: 2.5

Notes

.hwh and .bit file should have same filename when uploading to PYNQ board

Brief Design Flow

Xilinx Installation

- Remember to check “ZYNQ 7000” under “SoC”

HLS Flow

- Create project
  - Choose PYNQ board
  - Set clock period to >2ns (PYNQ-Z2 (https://www.tul.com.tw/productspynq-z2.html) support up to 650MHz). We use 5ns
- Design in C/C++
- Run C Simulation
- Synthesis
  - Set directives for cosimulation

    - Set top function
- Run Cosimulation
    - Vivado simulator
    - Dump trace: all
- Synthesis
    - Set directives for FPGA
- Export IP

Vivado Flow

- Create design project
- Add IP Catalog Repo
    - Select our HLS design directory
- Create block design
    - Add a ZYNQ7 Processing System block
        - Run block automation
        - Set PL fabric clock to 200MHz (correspond to the 5ns period)
    - Add our multip_2num block
        - Run connection automation
    - Create HDL wrapper for the design
- Generate bit-stream
    - Xilinx will automatically do systhesis, placement and routing for us

PYNQ Flow

- Upload .hwh (hardware handoff) and .bit (bitstream) to PYNQ, and include them in Python code