HW 5 電路設計說明

第14組

黃偉峰 E34106010

陳識博 E94106096

黃芊 F74104040

Problem 1 - Simple Computing System

電路設計說明

- 本次實驗的目的在於如何操作BRAM和DSP，由BRAM讀出資料送到DSP的input A, B進行運算，運算完的結果存回BRAM1的指定位址。
- 本次實驗主要有幾個重要模組，分別為gpio_input_decoder, dsp_direct, bram0, bram1，以下會逐一說明。

gpio_input_decoder

- 此模組為**Controller**，對送進來的instruction做decode的動作。依作業說明中的definition，將對應的資料送到對應的輸出。
- 這次實驗中，需要進行BRAM1[3] <= BRAM0[0] * BRAM1[2]等等的運算，但這些運算會需要兩個instruction來完成，第一個是對BRAM1的port B進行讀取，來做DSP的運算，第二個是寫回BRAM1 port B的指令，因此，**當execute=0時，對BRAM1的port B進行讀取；當execute=1時，將運算結果寫回BRAM1指定的位址。**

```
module gpio_input_decoder (
    input [31:0] inst,
    output [4:0] bram0_raddr,
    output [4:0] bram1_addr,
    output [4:0] dsp_inmode,
    output [6:0] dsp_opmode,
    output [3:0] dsp_alumode,
    output execute
);

    assign bram0_raddr = inst[4:0];
    assign bram1_addr = inst[31] ? inst[14:10] : inst[9:5];
    assign dsp_inmode = inst[19:15];
    assign dsp_opmode = inst[26:20];
    assign dsp_alumode = inst[30:27];
    assign execute = inst[31];

endmodule
```

dsp_direct

- 根據作業的description:
  - Data Input 只有 A, B，Data Output 只有 P。
  - Port C 固定為常數 0x0000_0009_5514。
  - 控制訊號有 INMODE, OPMODE, ALUMODE，CARRYINSEL 設為 000 並將 CARRYIN 設為 0 即可。
  - Pipeline Register 數量自訂，但 AREG, BREG, PREG 需至少為1。
- 因為有用到pipeline registers A, B, C, P，須將其對應的clock enable訊號開啟。
- 依照題目要求，P port的register必須開啟，因為我們一個procedure分成兩個instruction執行，第一個階段DSP算出的結果暫存在register中，第二階段會將計算結果**feedback回DPS，執行加0的加法**，再將P register中的值寫入BRAM1中，若沒有feedback的動作，則DSP又會從input A、B的地方取到錯的值計算。

```
DSP48E1 #(
    // Feature Control Attributes: Data Pat
    .A_INPUT("DIRECT"), // Selects A input
    .B_INPUT("DIRECT"), // Selects B input
    .USE_DPORT("FALSE"), // Select D port u
    .USE_MULT("MULTIPLY"), // Select multip
    .USE_SIMD("ONE48"), // SIMD selection (
    // Pattern Detector Attributes: Pattern
    .AUTORESET_PATDET("NO_RESET"), // "NO_R
    .MASK(48'h3fffffffffff), // 48-bit mask
    .PATTERN(48'h000000000000), // 48-bit p
    .SEL_MASK("MASK"), // "C", "MASK", "ROU
    .SEL_PATTERN("PATTERN"), // Select patt
    .USE_PATTERN_DETECT("NO_PATDET"), // En
    // Register Control Attributes: Pipelin
    .ACASCREG(1), // Number of pipeline sta
    .ADREG(0), // Number of pipeline stages
    .ALUMODEREG(1), // Number of pipeline s
    .AREG(1), // Number of pipeline stages
    .BCASCREG(1), // Number of pipeline sta
    .BREG(1), // Number of pipeline stages
    .CARRYINREG(0), // Number of pipeline s
    .CARRYINSELREG(0), // Number of pipelin
    .CREG(1), // Number of pipeline stages
    .DREG(0), // Number of pipeline stages
    .INMODEREG(1), // Number of pipeline st
    .MREG(1), // Number of multiplier pipel
    .OPMODEREG(1), // Number of pipeline st
    .PREG(1) // Number of pipeline stages f
)
```

```
DSP48E1_inst (
    .ALUMODE(ALUMODE), // 4-bit input: ALU
    .CARRYINSEL(0), // 3-bit input: Carry
    .CLK(clk), // 1-bit input: Clock input
    .INMODE(INMODE), // 5-bit input: INMOD
    .OPMODE(OPMODE), // 7-bit input: Opera
    // Data: 30-bit (each) input: Data Por
    .A(A), // 30-bit input: A data input
    .B(B), // 18-bit input: B data input
    .C(48'h000000095514), // 48-bit input
    .CARRYIN(0), // 1-bit input: Carry inp
    .D(), // 25-bit input: D data input
    // Reset/Clock Enable: 1-bit (each) in
    .CEA1(1), // 1-bit input: Clock enable
    .CEA2(0), // 1-bit input: Clock enable
    .CEAD(0), // 1-bit input: Clock enable
    .CEALUMODE(1), // 1-bit input: Clock e
    .CEB1(1), // 1-bit input: Clock enable
    .CEB2(0), // 1-bit input: Clock enable
    .CEC(1), // 1-bit input: Clock enable
    .CECARRYIN(0), // 1-bit input: Clock e
    .CECTRL(1), // 1-bit input: Clock enab
    .CED(0), // 1-bit input: Clock enable
    .CEINMODE(1), // 1-bit input: Clock en
    .CEM(1), // 1-bit input: Clock enable
    .CEP(1), // 1-bit input: Clock enable
    .RSTA(0), // 1-bit input: Reset input
    .RSTALLCARRYIN(0), // 1-bit input: Res
    .RSTALUMODE(0), // 1-bit input: Reset
    .RSTB(0), // 1-bit input: Reset input
    .RSTC(0), // 1-bit input: Reset input
    .RSTCTRL(0), // 1-bit input: Reset inp
    .RSTD(0), // 1-bit input: Reset input
```

bram0

- 下表為RAMB36E1針對不同port data width所對應的address bus長度，可以看到port data width為36(32 data + 4 parity)所對應的ADDR Bus只有[14:5]是有效地址。

Table 1-13: **Port Aspect Ratio for RAMB36E1 (in TDP Mode)**

Port Data Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
1	15	32,768	[14:0]	[0]	NA
2	14	16,384	[14:1]	[1:0]	NA
4	13	8,192	[14:2]	[3:0]	NA
9	12	4,096	[14:3]	[7:0]	[0]
18	11	2,048	[14:4]	[15:0]	[1:0]
36	10	1,024	[14:5]	[31:0]	[3:0]
1 (Cascade)	16	65536	[15:0]	[0]	NA

bram0

- Port A(**可以讀以及寫**)與CPU進行溝通，Port B(**單純讀不能寫**)則是將instruction傳進來的地址(bram0_raddr)讀出資料(**DOBDO**)至DSP的A port。
- 經過實驗得知，需要將DOA以及DOB的output register關掉，DSP的input port才會讀到正確且立即性的值。
- ADDRARDADDR為從processer進來的地址，因為在PS端與PL端溝通的方式都是透過XPAR_AXI_BRAM_CTRL_NUM_S_AXI_BASEADDR + 4i 這行指令進行，其地址的增長是每四個增加，為byte addressing mode，因此只須取ADDRARDADDR[10:2]，而忽略ADDRARDADDR[1:0]。
- ADDRBWRADDR則是從instruction端進來的地址，是word addressing mode，因此不能省略任何部分，為ADDRBWRADDR[4:0]。

Block RAM initial data

```
RAMB36E1 #(
    .INITP_0E(256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INITP_0F(256'h0000000000000000000000000000000000000000000000000000000000000000),
    // INIT_00 to INIT_7F: Initial contents of the data memory array
    .INIT_00(256'h0000000000005310000000000000000000000120100000010000000032),
    .INIT_01(256'h00000000000000000000000000000FFFFFF320000000000000000000000),
    .INIT_02(256'h0000000000000000000000000000000000000000000000000000000000),
    .INIT_03(256'h0000226300000000000000000000000000000000000000000000000000),
    .INIT_04(256'h0000000000000000000000000000000000000000000000000000000000),
```

ADDRARDADDR &
ADDRBWRADDR

```
// when RAM_MODE="SDP")
.ADDRARDADDR({1'b0, 1'b0, ADDRARDADDR[10:2], 5'b11111}), // 16-bit input
.CLKARDCLK(clk), // 1-bit input: A port clock/Read clock
.ENARDEN(ENARDEN), // 1-bit input: A port enable/Read enable
.REGCEAREGCE(1), // 1-bit input: A port register enable/Register enable
.RSTRAMARSTRAM(), // 1-bit input: A port set/reset
.RSTREGARSTREG(), // 1-bit input: A port register set/reset
.WEA(WEA), // 4-bit input: A port write enable
// Port A Data: 32-bit (each) input: Port A data
.DIADI(DIADI), // 32-bit input: A port data/LSB data
.DIPADIP(), // 4-bit input: A port parity/LSB parity
// Port B Address/Control Signals: 16-bit (each) input: Port B address a
// when RAM_MODE="SDP")
.ADDRBWRADDR({1'b0, 5'b00000, ADDRBWRADDR[4:0], 5'b11111}), // 16-bit in
.CLKBWRCLK(clk), // 1-bit input: B port clock/Write clock
.ENBWREN(1'b1), // 1-bit input: B port enable/Write enable
.REGCEB(1), // 1-bit input: B port register enable
```

bram1

- 與bram0最大的不同是bram1 port B可以讀也可以寫，其讀與寫的憑據WEB是來自instruction的execute，讀則是將instruction傳進來的地址(bram1_addr)讀出資料(**DOBDO**)至DSP的B port；寫則是將DSP的P port寫入BRAM1的bram1_addr位置。
- DIBDI是來自DSP P port的運算結果。

DIBDI & WEB

```
module bram1(
    input clk,
    input rst_a,
    input [10:0] ADDRARDADDR,
    input ENARDEN,
    input [3:0] WEA,
    input [31:0] DIADI,
    input [31:0] DIBDI,
    input [4:0] ADDRBWRADDR,
    input WEB,
    output [31:0] DOADO,
    output [31:0] DOBDO
);
```

```
RAMB36E1_inst (
    // Port B Address/Control Signals: 16-bit (each) input: Port B address and control si
    // when RAM_MODE="SDP")
    .ADDRBWRADDR({1'b0, 5'b00000, ADDRBWRADDR[4:0], 5'b11111}), // 16-bit input: B port a
    .CLKBWRCLK(clk), // 1-bit input: B port clock/Write clock
    .ENBWREN(1'b1), // 1-bit input: B port enable/Write enable
    .REGCEB(1), // 1-bit input: B port register enable
    .RSTRAMB(), // 1-bit input: B port set/reset
    .RSTREGB(), // 1-bit input: B port register set/reset
    .WEBWE({4'd0, WEB, WEB, WEB, WEB}), // 8-bit input: B port write enable/Write enable
    // Port B Data: 32-bit (each) input: Port B data
    .DIBDI(DIBDI), // 32-bit input: B port data/MSB data
    .DIPBDIP() // 4-bit input: B port parity/MSB parity
);
```

Instruction 歸納

- ALUMODE(4bit)是在決定X，Y以及Z MUXes出來output的運算模式。
- OPMODE(7bit)是在決定X，Y，以Z muxes的output。

Operation	ALUMODE	OPMODE	Z output
A*B = Z + X + Y	0000	000_01_01	0
A*B+C = Z + X + Y	0000	011_01_01	C
C-A*B = Z-(X+Y)	0011	011_01_01	C
A*B-C-1 = -Z-1+(X+Y)	0001	011_01_01	C

Instruction 歸納

- INMODE(5bit)的[3:0]是在決定 Dual A,D and pre-adder logics，使得multiplier A port 有不同的輸出。INMODE[4]是在決定multipler B port的輸出。

Operation	INMODE
DSP運算	1_0001
寫回BRAM	0_0000

Table 2-5: **INMODE[3:0] Functions (when AREG = 1 or 2)**

INMODE[3]	INMODE[2]	INMODE[1]	INMODE[0]	USE_DPORT	Multiplier A Port
0	0	0	0	FALSE	A2
0	0	0	1	FALSE	A1
0	0	1	0	FALSE	Zero
0	0	1	1	FALSE	Zero
0	0	0	0	TRUE	A2
0	0	0	1	TRUE	A1
0	0	1	0	TRUE	Zero
0	0	1	1	TRUE	Zero
0	1	0	0	TRUE	D + A2(1)
0	1	0	1	TRUE	D + A1(1)
0	1	1	0	TRUE	D
0	1	1	1	TRUE	D
1	0	0	0	TRUE	-A2
1	0	0	1	TRUE	-A1
1	0	1	0	TRUE	Zero
1	0	1	1	TRUE	Zero
1	1	0	0	TRUE	D – A2(1)
1	1	0	1	TRUE	D – A1(1)

Table 2-6: **INMODE[4] Encoding (when BREG = 1 or 2)**

INMODE[4]	Multiplier B Port
0	B2
1	B1

Our code (For example)

以BRAM1[3] <= BRAM0[0] * BRAM1[2]為例:
首先先做BRAM0[0] * BRAM1[2]，為A*B的形式，因此，
```
E = 0
ALUMODE = 0000
OPMODE = 000_01_01
INMODE = 1_0001
```
接下來寫回BRAM1，
```
E = 1
ALUMODE = 0000
OPMODE = 000_00_10
INMODE = 0_0000
```

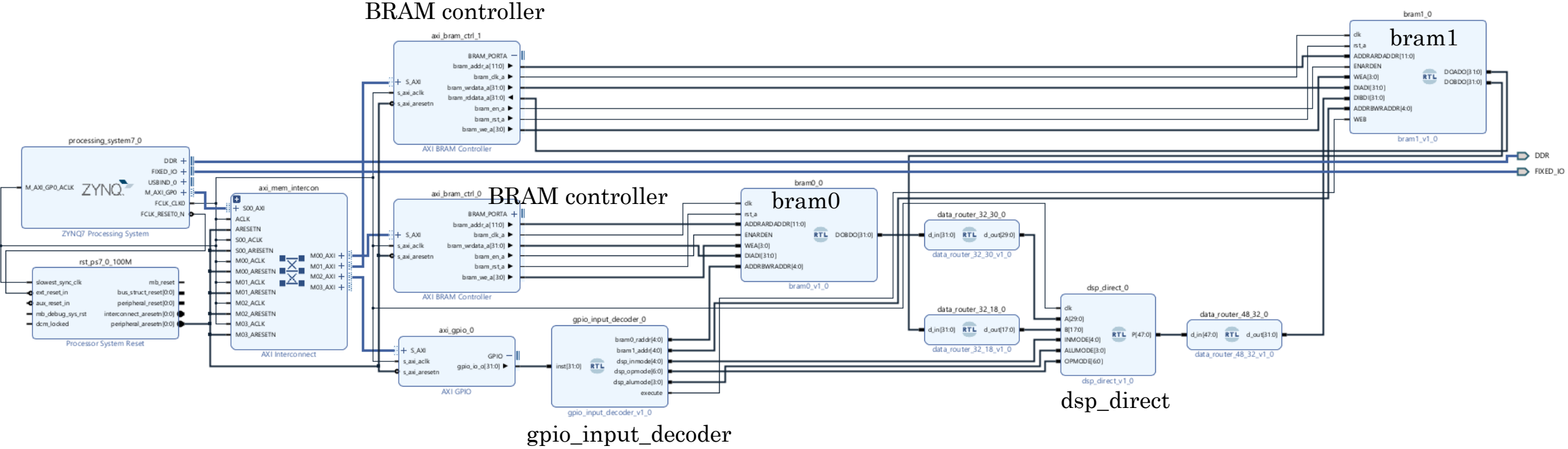
```
/* ------------------DSP computing stage 1------------------------*/
printf("--------------------------------\r\n");
printf("BRAM1[3] <= BRAM0[0] * BRAM1[2]\r\n"); // 38432 -> check
//        E/ ALU/    OP /   IN/ B1WR/ B1R /  B0R
// inst = 0_0000_0000101_10001_00000_00010_00000
inst = 0b00000000010110001000000001000000;
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR, inst);

/*---------------------Write to BRAM1[3]-------------------------*/
//        E/ ALU/    OP /   IN/ B1WR/ B1R /  B0R
// inst = 1_0000_0000010_00000_00011_00000_00000
inst = 0b10000000001000000000110000000000;
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR, inst);

printf("--------------------------------\r\n");
printf("BRAM1[7] <= BRAM0[11] * BRAM1[3]\r\n");
//        E/ ALU/    OP /   IN/ B1WR/ B1R /  B0R
// inst = 0_0000_0000101_10001_00000_00011_01011
inst = 0b00000000010110001000000001101011;
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR, inst);

/*---------------------Write to BRAM1[7]-------------------------*/
//        E/ ALU/    OP /   IN/ B1WR/ B1R /  B0R
// inst = 1_0000_0000010_00000_00111_00000_00000
inst = 0b10000000001000000001110000000000;
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR, inst);
```

Block Design



Result

```
--------------------------------
BRAM1[3] <= BRAM0[0] * BRAM1[2]
--------------------------------
BRAM1[7] <= BRAM0[11] * BRAM1[3]
--------------------------------
BRAM1[10] <= BRAM0[31] * BRAM1[7] + C
--------------------------------
BRAM1[13] <= C - BRAM0[1] * BRAM1[6]
--------------------------------
BRAM1[15] <= BRAM0[0] * BRAM1[31] - C - 1
First Print BRAM1:
BRAM1[0] = 0x32
BRAM1[1] = 0x10
BRAM1[2] = 0x1201
BRAM1[3] = 0x38432
BRAM1[4] = 0x0
BRAM1[5] = 0x0
BRAM1[6] = 0x531
BRAM1[7] = 0x639fc4
BRAM1[8] = 0x0
BRAM1[9] = 0x0
BRAM1[10] = 0xf31c25e0
BRAM1[11] = 0xffffff32
BRAM1[12] = 0x0
BRAM1[13] = 0x90204
BRAM1[14] = 0x0
BRAM1[15] = 0xfffd6241
BRAM1[16] = 0x0
BRAM1[17] = 0x0
BRAM1[18] = 0x0
BRAM1[19] = 0x0
BRAM1[20] = 0x0
BRAM1[21] = 0x0
BRAM1[22] = 0x0
BRAM1[23] = 0x0
BRAM1[24] = 0x0
BRAM1[25] = 0x0
BRAM1[26] = 0x0
BRAM1[27] = 0x0
BRAM1[28] = 0x0
BRAM1[29] = 0x0
BRAM1[30] = 0x0
BRAM1[31] = 0x2263
====================
====================
====================
--------------------------------
BRAM1[16] <= BRAM0[0] * BRAM1[2]
--------------------------------
BRAM1[17] <= BRAM0[11] * BRAM1[3]
--------------------------------
BRAM1[18] <= BRAM0[31] * BRAM1[7] + C
--------------------------------
BRAM1[19] <= C - BRAM0[1] * BRAM1[6]
--------------------------------
BRAM1[20] <= BRAM0[0] * BRAM1[31] - C - 1
Second Print BRAM1:
BRAM1[0] = 0x32
BRAM1[1] = 0x10
BRAM1[2] = 0x1201
BRAM1[3] = 0x38432
BRAM1[4] = 0x0
BRAM1[5] = 0x0
BRAM1[6] = 0x531
BRAM1[7] = 0x639fc4
BRAM1[8] = 0x0
BRAM1[9] = 0x0
BRAM1[10] = 0xf31c25e0
BRAM1[11] = 0xffffff32
BRAM1[12] = 0x0
BRAM1[13] = 0x90204
BRAM1[14] = 0x0
BRAM1[15] = 0xfffd6241
BRAM1[16] = 0x1201
BRAM1[17] = 0xffba5c20
BRAM1[18] = 0xfe886514
BRAM1[19] = 0x94050
BRAM1[20] = 0xfff6cd4e
BRAM1[21] = 0x0
BRAM1[22] = 0x0
BRAM1[23] = 0x0
BRAM1[24] = 0x0
BRAM1[25] = 0x0
BRAM1[26] = 0x0
BRAM1[27] = 0x0
BRAM1[28] = 0x0
BRAM1[29] = 0x0
BRAM1[30] = 0x0
BRAM1[31] = 0x2263
====================
```

Problem 2 - Q&A

PYNQ-Z2 上共有多少個 DSP48E1 Slice？

- 由 PYNQ-Z2 Reference Manual v1.0 p.4可知， DSP48E1 Slice 有220個，以下為文件中截圖:

1 PYNQ-Z2 features

- **ZYNQ XC7Z020-1CLG400C**
  - 650MHz ARM® Cortex®-A9 dual-core processor
  - Programmable logic
    - 13,300 logic slices, each with four 6-input LUTs and 8 flip-flops
    - 630 KB block RAM
    - 220 DSP slices
    - On-chip Xilinx analog-to-digital converter (XADC)
  - Programmable from JTAG, Quad-SPI flash, and MicroSD card

Reference

- DSP48E1 USER GUIDE NOTES
- 7 Series FPGAs Memory Resources User Guide
- 7 Series DSP48E1 Slice User Guide