**IMPORTANT:**
- Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
- As always, the best answer gets the most points.

**COVER SHEET:**

| Problem: | Points: |
|---|---|
| **1 (20 pts)** | |
| **2 (8 pts)** | |
| **3 (18 pts)** | |
| **4 (22 pts)** | |
| **5 (14 pts)** | |
| **6. (18 pts)** | |

**Total** 

**Re-Grade Information:**

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

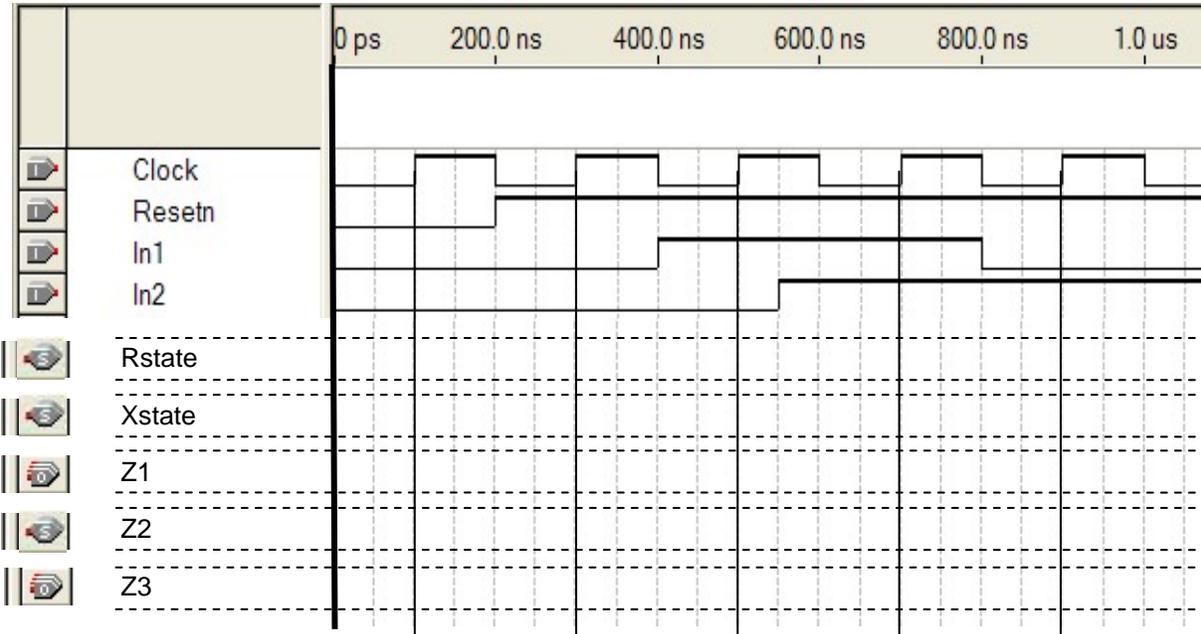**1.** Based on the VHDL code at the bottom of this page and on the next page, complete the following diagram. Specify the values for the signals (Rstate, Xstate, Z1, Z2, and Z3) **as the appropriate signal types** (i.e., how Quartus would display them).

20 pts.



> **Be sure to show timing delays. Also, complete the timing diagrams to the end.**

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY T2AsmProb IS
        PORT ( Clock, Resetn, In1, In2  : IN      STD_LOGIC ;
                Z1, Z3 : OUT STD_LOGIC ) ;
END T2AsmProb ;
```

1

```
ARCHITECTURE ASMArch OF T2AsmProb IS
      TYPE MyType IS (A, B, C) ;
      SIGNAL Rstate, Z2 : MyType ;
      SIGNAL Xstate : STD_LOGIC_Vector (1 DOWNTO 0);
      CONSTANT D : STD_LOGIC_Vector (1 DOWNTO 0):= "00";
      CONSTANT E : STD_LOGIC_Vector (1 DOWNTO 0):= "01";
      CONSTANT F : STD_LOGIC_Vector (1 DOWNTO 0):= "10";
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn = '0' THEN
                  Rstate <= B;
                  Xstate <= E;
            ELSIF (Clock'EVENT AND Clock = '1') THEN
                  CASE Rstate IS
                        WHEN A =>
                              IF In1 = '0' THEN Rstate <= B ;
                              ELSE Rstate <= C ;
                              END IF ;
                        WHEN B =>
                              Rstate <= A;
                              Z1 <= '1';
                        WHEN C =>
                              IF In2 = '1' THEN Rstate <= C;
                              ELSE Rstate <= B;
                              END IF ;
                              Z1 <= '0';
                  END CASE ;
                  CASE Xstate IS
                        WHEN D =>
                              IF In1 = '0' THEN Xstate <= E ;
                              ELSE Xstate <= "10";
                              END IF ;
                        WHEN E =>
                              IF IN2 = '1' THEN Xstate <= "10";
                              ELSE Xstate <= D;
                              END IF ;
                        WHEN OTHERS =>
                              Xstate <= "00";
                  END CASE ;
            END IF;
      END PROCESS ;
      Process (Xstate)
      Begin
            CASE Xstate IS
                  WHEN D =>
                        Z2 <= A;
                  WHEN OTHERS =>
                        Z2 <= B;
            END CASE;
      END PROCESS ;
      Z3 <= '1' WHEN Rstate = C AND In1 = '1' AND In2 = '1' ELSE '0';
END ASMArch ;
```

8 pts.

## 2. **Miscellaneous.**

(a) **VGA display calculation:**
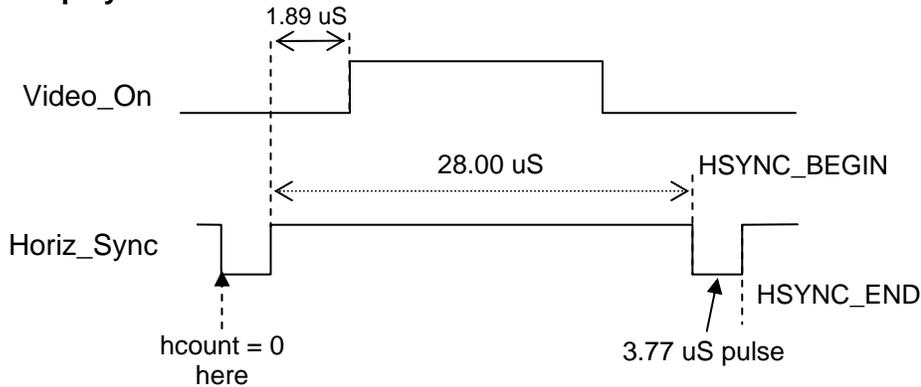


Assume a 25.175 MHz board clock rate and assume we set the "reference point" for hcount = 0 as shown, what would be the constant values for hcount for HSYNC_BEGIN and HSYNC_END?  (3 pts)

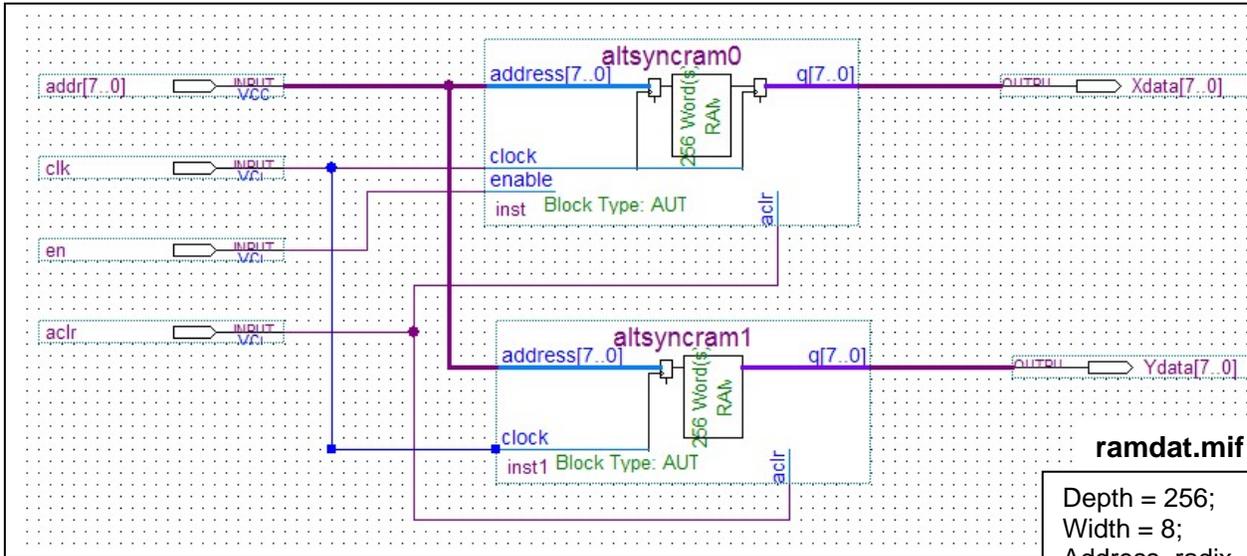        HSYNC_BEGIN = _____            HSYNC_END = _____

For credit, please show work:

(b) What is the purpose of the HSYNC signal? (1 pt)

(c) There are 5 types of interconnects in the Cyclone FPGA family. Name 4 of them and give a one-sentence description of each. (4 pts)

2

### 3. AltSyncRAM

18 pts.



**ramdat.mif**

Depth = 256;
Width = 8;
Address_radix = hex;
Data_radix = hex;
Content
Begin

| | |
|---|---|
| 00 : 70; | 07 : 77; |
| 01 : 71; | 08 : 78; |
| 02 : 72; | 09 : 79; |
| 03 : 73; | 0A : 7A; |
| 04 : 74; | 0B : 7B; |
| 05 : 75; | 0C : 7C; |
| 06 : 76; | etc. |

**Aclr = '1'** will asynchronously clear all the flipflops in the altsyncram component to '0'.

**enable** = 1 enables the synchronous loading of all the flipflops in the altsyncram component (if enable is used in that altsyncram).

**enable** = 0 disables the loading of all the flipflops in the altsyncram component (i.e. "holds"), if enable is used.

**Based on your understanding of the altsyncram component from the labs, and the explanation above, complete the following timing diagram.**

**Assume** all flip-flops are initialized to '0'.

Both RAM's has the same data (ramdat.mif).
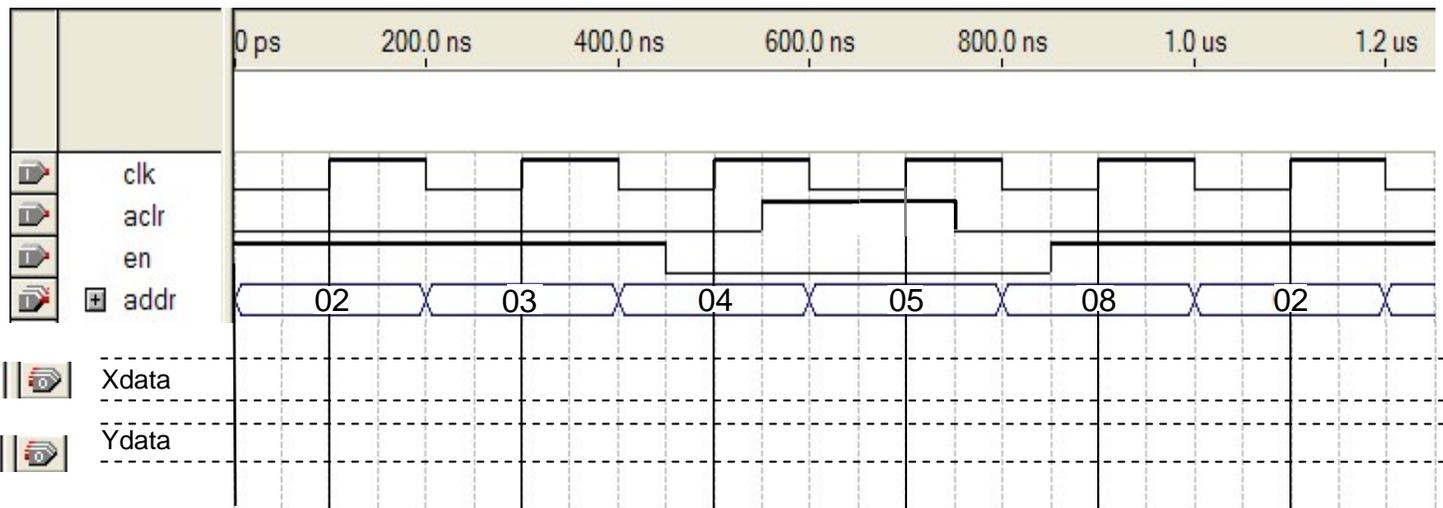


Please show <u>delays</u> and put answers for Xdata and Ydata in <u>hex</u>.

3

**4. Digital design using ASM:**

22 pts.

A parallel-to-serial (PS) converter is the "transmitter" part of a UART (Universal Asynchronous Receiver/Transmitter) commonly used in microprocessor systems and micro-controllers. As shown in Figure 4(a), a SOURCE (e.g., a microprocessor under software control) provides data bytes (not necessarily continuously or in regular intervals) to the PS converter, which transmits each byte out serially as a data frame. For example, this can be used to produce and transmit the "keydata" signal that your keyboard controller received in Lab 6, as illustrated in Figure 4(b).

- "Handshaking" between the SOURCE and the PS converter. When the PS converter is ready for a byte (i.e., the SHIFT register is empty), it should signal the SOURCE to send the next byte by setting the Empty signal to '1'. It is the responsibility of the SOURCE to put the 8-bit data on BYTE and set Ready to '1' (and hold them there until Empty becomes '0'). The PS converter should keep Empty '0' while it shifts out the data frame. The SOURCE will keep READY '0' while Empty is '0' and will not send another byte until Empty is '1' again.

  **NOTE:** In this problem you are responsibility for PS-Converter, **not** the actions of SOURCE.

The behavior of the PS converter is outlined as follows:

- If the Reset signal is asserted '1', the ASM is asynchronously reset to the initial state. The PS converter will begin to output a "MARK" bit ('1') at the TxD output (See Figure 4(b)). It will continue to output MARK bits until the Ready signal becomes '1', indicating a data byte should be loaded into the SHIFT register at the next active edge of the PSCLK.

- At this point, the PS converter should output a START bit ('0') at TxD.

- Subsequently, the data byte is shifted out, starting with D(0) and ending with D(7). Then, a STOP bit (high) is outputted.

Ideally, we would like to shift the data out one byte right after another (along with the START and STOP bits) without any MARK bits if the SOURCE can keep up. Otherwise, MARK bits will be inserted between bytes.

### 4. **Digital design using ASM:** (continued)

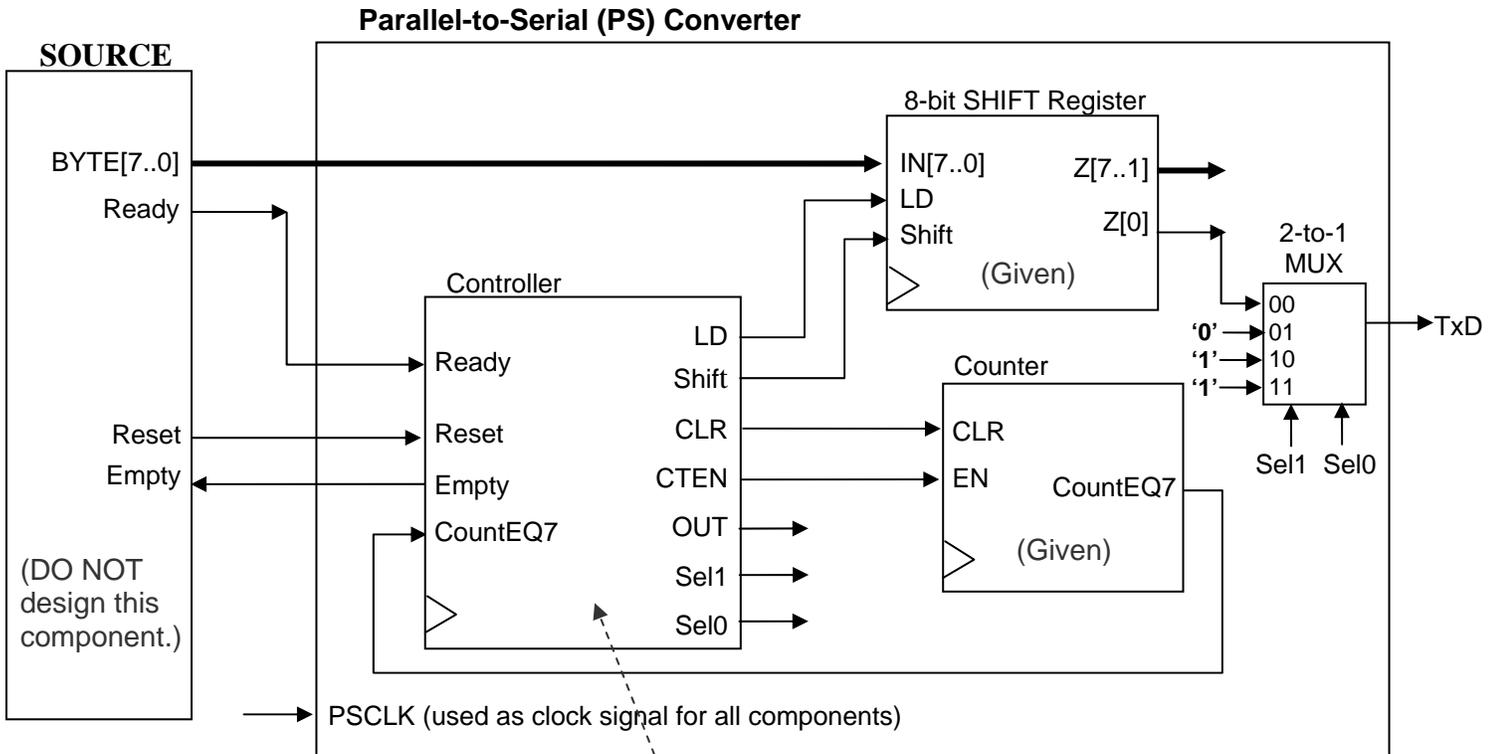**Parallel-to-Serial (PS) Converter**



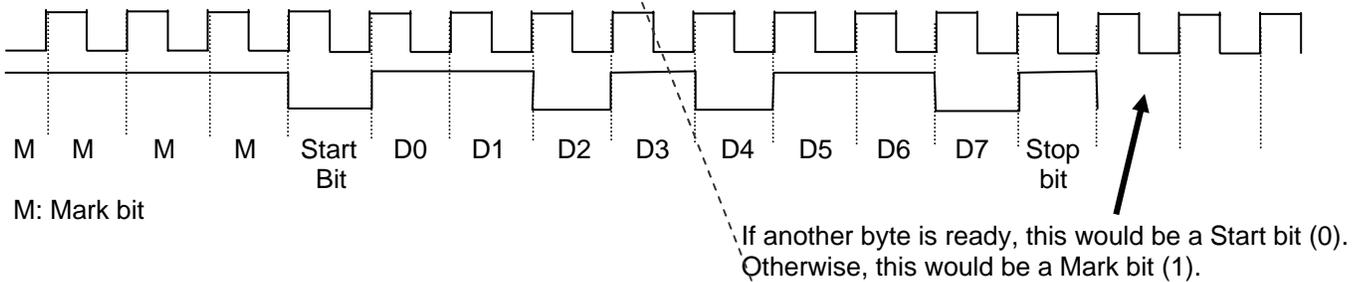Figure 4(a). Block diagram design of the PS converter.



Figure 4(b). Timing diagram illustrating the behavior of the PS converter.
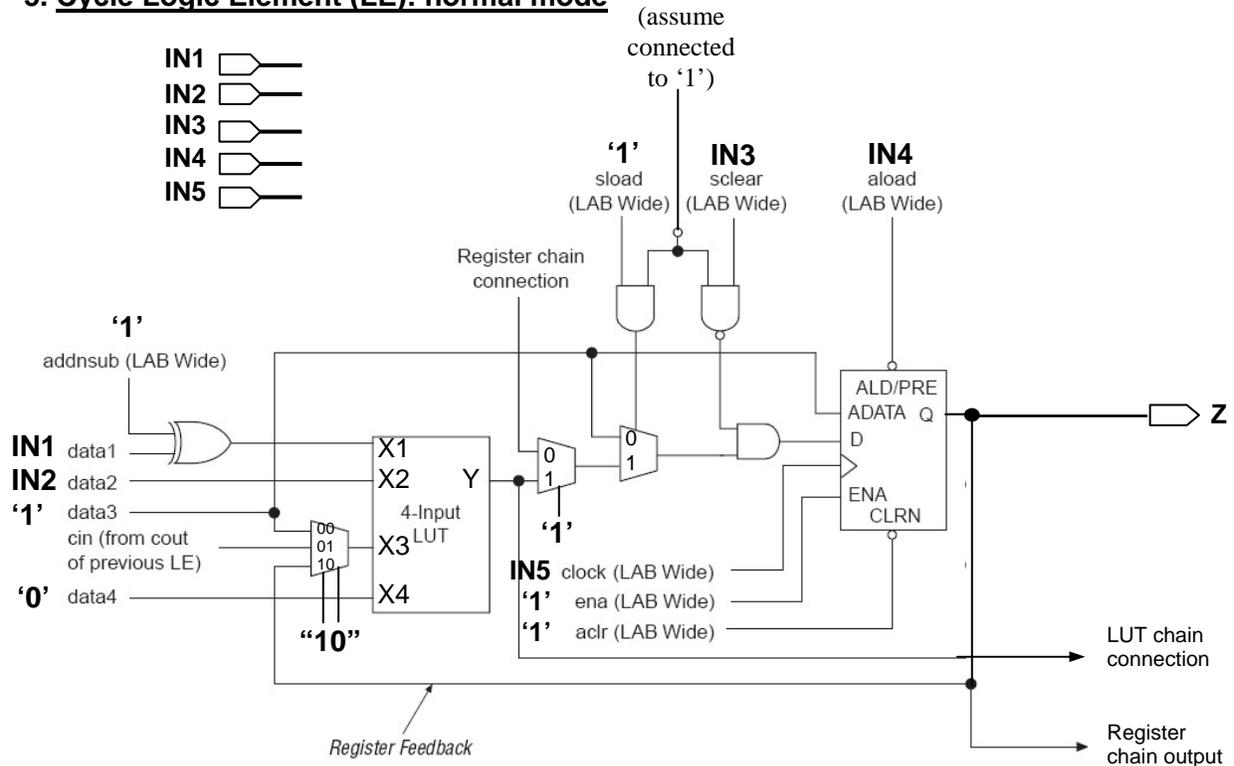
In this problem, all you <u>only</u> need is to produce the <u>ASM chart</u> for the **controller** of the PS Converter. The best answer gets the most points. In other words, using less states and conditional outputs to improve performance will result in more points. However, it is better to get a correct answer with more than minimum states than to get an erroneous answer with fewer states. (Put your ASM chart on the next page.)

**4.** Put your ASM chart for Problem 4 here.

**14 pts.**

### 5. <u>Cycle Logic Element (LE): normal mode</u>

IN1
IN2
IN3
IN4
IN5

(assume connected to '1')

'1'
sload
(LAB Wide)

IN3
sclear
(LAB Wide)

IN4
aload
(LAB Wide)

Register chain connection

'1'
addnsub (LAB Wide)

ALD/PRE
ADATA   Q
D

IN1  data1
IN2  data2
'1'   data3
cin (from cout of previous LE)
'0'   data4

X1
X2        Y
X3   4-Input
       LUT
X4

00
01
10

"10"

0
1

0
1

'1'

ENA
CLRN

IN5  clock (LAB Wide)
'1'   ena (LAB Wide)
'1'   aclr (LAB Wide)

Z

LUT chain connection

Register Feedback

Register chain output

Show above is circuit diagram of a logic element (LE) in the normal mode, already "configured by Quartus". Reverse-engineer it and draw the corresponding circuit diagram, as in a .bdf file. For a clear or set input, specify whether it is synchronous or asynchronous.
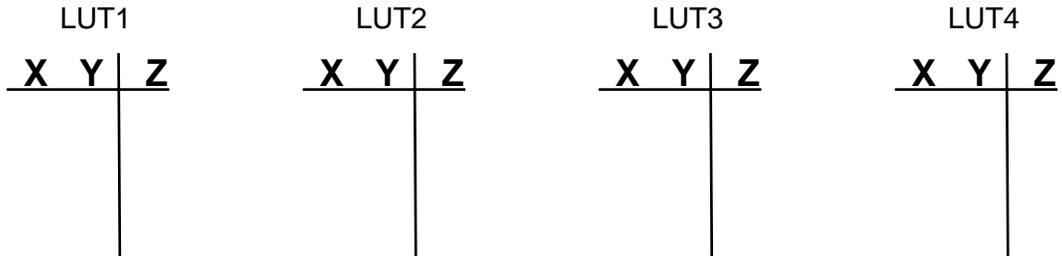
**Draw answer here:**

Contents of LUT:

| X4 | X3 | X2 | X1 | Y |
|----|----|----|----|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**5**

**6. Cycle Logic Element (LE): arithmetic mode**

14 pts.

Show on Figure 6 on the next page is an implementation of an 8-bit adder, using 8 of the 10 LE's in a Logic Array Block (LAB).

(a) Show on the right of the figure is a detailed "blow-up" of one of the LE's, containing 4 look-up tables (LUT's). Give the content of each LUT (LUT1, LUT2, LUT3, and LUT3).

| LUT1 | | | LUT2 | | | LUT3 | | | LUT4 | | |
|------|---|---|------|---|---|------|---|---|------|---|---|
| **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| | | | | | | | | | | | |

(b) The signals "a", "b", … , "r" represents the carry outs coming out of each LE. (a, c, e, g, i, l, n, and p) comprise carry-out chain 0. (b, d, f, h, j, m, o, and q) comprise carry-out chain 1. "r" is the LAB Carry-in at the top.

Assume that we are adding the following two 8-bit numbers:

A8 – A1 = 1010 0100 and B8 – B1 = 0011 1101 (Note the sum should equal 1110 0001.)

Give the value for each of the signals ("a", "b", … , "r"). You should put the answer (either a '0' or a '1'), **right near the letters in Figure 6**.

(c) Explain which carry-out chain (or chains) is(are) used.

**Figure 6 (used for Problem 6 on the previous page)**

| 6 |
|---|
|   |