

**GENERAL**  
**ADVANCED XILINX and DIGILENT FEATURES**

**1. Introduction**

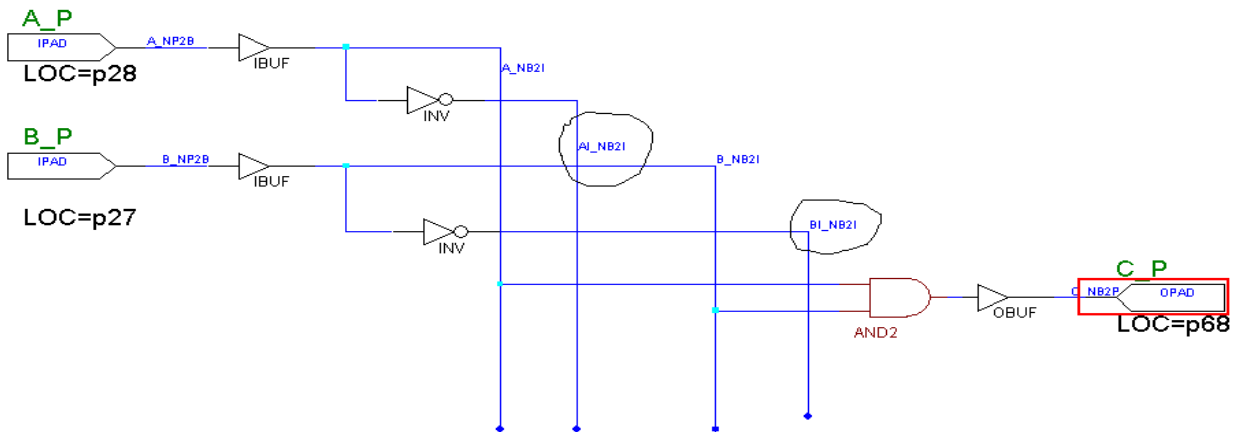
This handout describes advanced Xilinx and Digilent points that will be helpful throughout the semester. Students are advised that they bring this handout to the lab together with all the other handouts distributed.

**2. Team-Oriented Schematic Design and Sourcing Schematic Files**

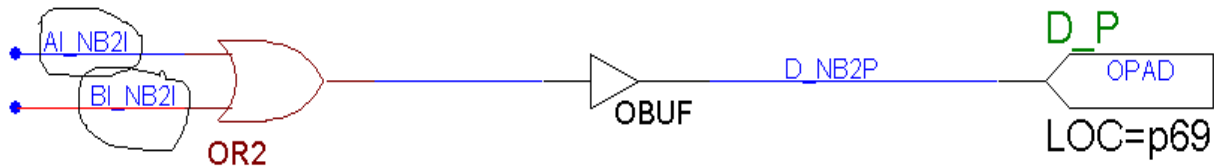
Real life circuits contain a large number of wires, connecting components. Wires take a substantial amount of screen space and time to draw. To reduce the work and the visual complexity of the schematic, two Xilinx design features can be used. They also allow team members to develop schematic designs in parallel :

- Wires with identical names are automatically connected by Xilinx even though they are “physically” disconnected from each other on the screen.
- A project can have two or more schematic files which are automatically “combined” by the software into one complete schematic circuit.

Two students can develop two parts of a schematic in parallel. When they finish, the schematic file of one student is copied to the project directory of the other student whose project would have then two schematic files. Then, the software would automatically “combine” them. For example consider functions: “ $C = AB$ ,” and “ $D = \overline{A} + \overline{B}$ .” One student of a team can work on the inputs, the “C” output and “ $C = AB$ .” At the same time, the other student works on “ $D = \overline{A} + \overline{B}$ ” and the “D” output. The following schematic is developed by member 1 of the team :

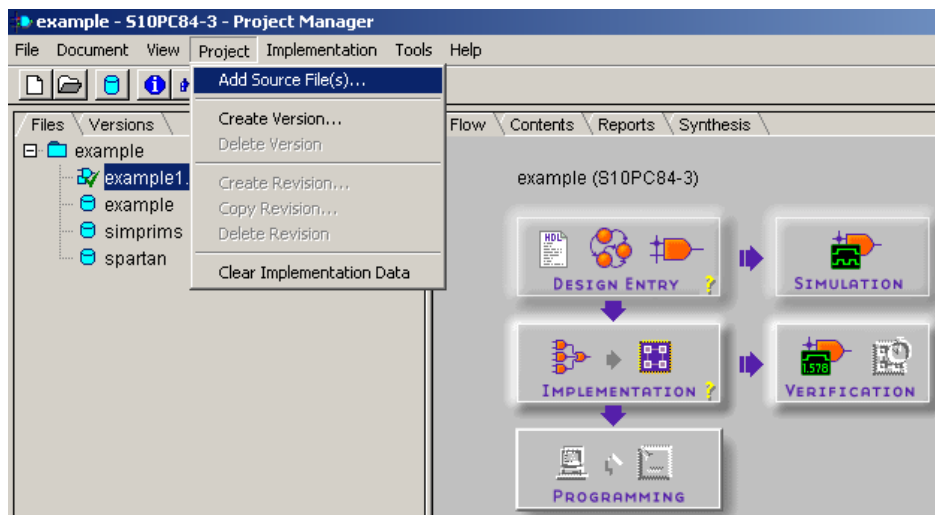


Member 2 of the team develops the following schematic at the same time :

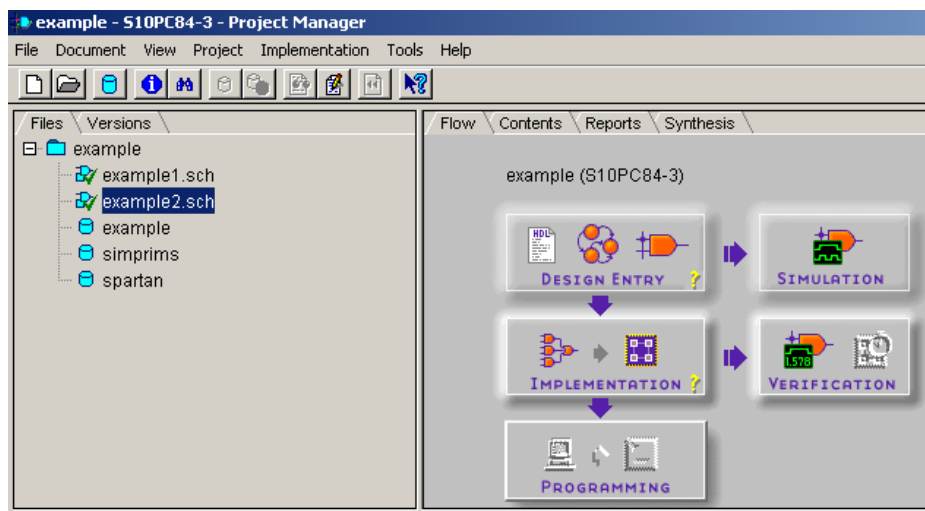


If students use the same name for the wires that need to be connected to each other in the overall schematic, the combined circuit works. For example, the wire names used by the second member (AI\_NB2I and BI\_NB2I) are identical to the wire names used by the first member. This is because, the wires should be connected to the same input buffers drawn by member 1. When both schematic designs are “combined” into one project, the AI\_NB2I and BI\_NB2I wires of the second member are automatically “connected” to the AI\_NB2I and BI\_NB2I wires of the first member, even if they are in two separate schematics. Here is the sequence of steps to implement these ideas :

- Start your project by discussing with your partner how to partition the work and the wire names. Then, both partners start the work. When the team members are finished, one of them needs to transfer the schematic to the other computer. Assume that student 2 transfers his/her schematic file to the PC of student 1. A memory stick or a floppy disk or an email message is needed to do this transfer.
- **Student 2 first copies the schematic file to the memory stick :** Assuming the project name is “example, ” there is a project description file called “example.pdf” and a directory “example” on your S drive. The schematic file has a default name “example1.sch” and is located under “example.” Copy this file to the memory stick from the “example” directory.
- **Rename the file on the stick : “example2.sch.” Transfer the memory stick to the computer of Student 1 :** If not renamed, copying will overwrite the file with the same name in the directory of Student 1.
- **Copy “example2.sch” to the “example” directory of Student 1 :** Now the Student 1 project directory has two files with extension “.sch”
- **Add the newly copied schematic file “example2.sch” to the project of Student 1 :** Pull down the “Project” menu :



- Click on the first menu choice “Add Source File(s)...” which results in a new window,
- In the new window click on the newly copied file “example2.sch”. Then press the “open” button. The following project manager window will appear :



- Now the two schematic files belong to the same project. One does **not** even need to copy the two schematics to one schematic sheet.
- One continues with the remaining steps as before : “Simulation,” “Implementation” and “Programming.”

*When the Xilinx IMPLEMENTATION is started, there will be warning messages in blue on the message panel section of the Project Manager. They complain about unconnected outputs, or loadless wires. These are due to the fact that each schematic sheet is “parsed” separately first and then the results are put together. These warnings are redundant if the Xilinx IMPLEMENTATION completes without warnings and errors.*

### 3. Using Xilinx Constraint Editor and Clearing the Implementation Data

The Xilinx software does **not** allow assigning pin numbers to blocks of pads, such as IPAD4, IPAD8, IPAD16, OPAD4, OPAD8 and OPAD16 in the schematic. That is, blocks of pads cannot be assigned pin numbers by using the “LOC” option. In order to assign the pin numbers, one has to use the Constraint Editor, reached from the Project Manager window. Typically, the Constraint Editor is used to enter certain information about the design, such as timing constraints. The information is kept in a file with the type “ucf.” The ucf file is, however, created after the **Translate** step of the Xilinx Implementation. Thus, one needs to start Xilinx Implementation, wait until the Translate step is completed. Then, abort the Implementation by clicking on the “**Abort**” button on the bottom of the Implementation window and close this window. Then follow the steps below, starting with the **Project Manager** window :

- Follow the chain menu selections : **Tools -> Implementation -> Constraint Editor.**
- In the Constraint Editor window select “**Ports**”
- Fill in the pin table that appears. That is, for each pad, enter its pin number.

Note that, if one makes changes on the wires connected to the pads, such as changing the name of a wire connected to a pad. Then, Xilinx Implementation will give errors even if the design is correct. The reason is that the Implementation uses the old ucf file which has a different name for the wire. The solution is that, one needs to delete this constraint file content. One can do it by deleting what Xilinx calls “**Implementation Data.**” Follow the steps below to delete it starting with the **Project Manager** window :

- Follow the chain menu selections : **Project -> Clear Implementation Data.**

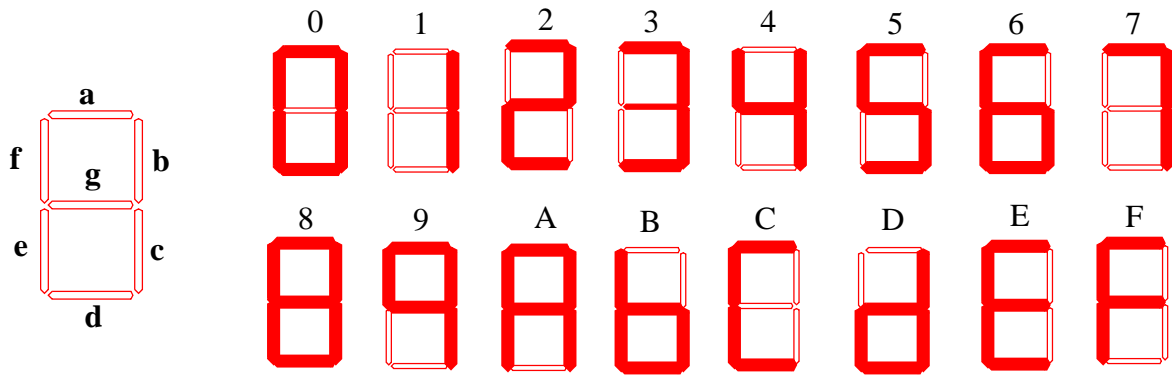
This process deletes the old ucf file. Now it is as if no IMPLEMENTATION has been done. Thus, one needs to restart the Xilinx IMPLEMENTATION steps described above to create the ucf file and then enter the new data.

*Note that clearing the implementation data is needed even if the Constraints Editor is **not** used. The reason is that back to back Xilinx implementations utilize the implementation data already generated and eventually the implementation data becomes “corrupt.” Thus, students are suggested that they frequently clear the implementation data.*

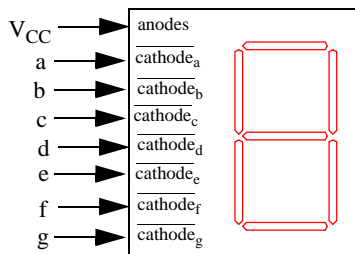
### 4. Digilent FPGA Board 7-Segment Display Hardware

A 7-segment display has seven light emitting segments that collectively display a digit. A segment is a light emitting diode (LED) which emits light if an electric current passes through it. Every segment of the 7-segment display has a unique position and label for reference. In the figure below, we show the placement and naming of the segments and how hexadecimal digits will be shown **this** semester. One can clearly show all decimal digits. But, hexadecimal digits (0 through F) can be shown in a nonuniform format where some alphabetical hexadecimal digits are shown in capital and some in lower case. This is because more than seven segments are needed for a uniform format.

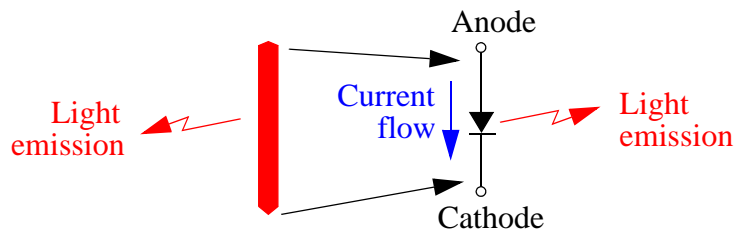
A current flow through the LED segment is possible only if there is a voltage difference, typically 5 volts, between its two terminals. The name of the terminals are called **anode** and **cathode**. The anode is the positive terminal and the cathode is the negative terminal. Theoretically, a 7-segment display would need 14 terminals, two for each diode, for proper operation. But, practically, it is not desirable since the display physically would be too large. Thus, today, a 7-segment display is used either in the common anode form or in the common cathode form, leading to eight terminals. In the common anode form, the anodes of all the segments are connected (tied to)  $V_{CC}$ . To turn on an individual segment, the cathode of the intended segment must be applied 0v. This will result in an electric current through the segment which will cause the segment to emit light. The common cathode form is similar where all the cathodes are connected 0v. To turn on a segment, its anode is applied  $V_{CC}$ . The 7-segment displays on our FPGA board are **common anode** displays. Thus, we will restrict our discussion to this form from now on.



The reason for eight inputs (terminals) for the common anode display is that seven inputs are needed to apply 0v to the cathodes of segments “a” through “g,” and one input is needed to apply  $V_{CC}$  to all the anodes of the segments. The anode input is connected to +5v ( $V_{CC}$ , high) and cathode inputs are **normally** +5v (high) and when we want to turn on a segment, its cathode input is pulled to 0v (low). Therefore, the **cathode inputs are active-low inputs**. The figure below shows a 7-segment display device in the common anode form and a typical LED segment :



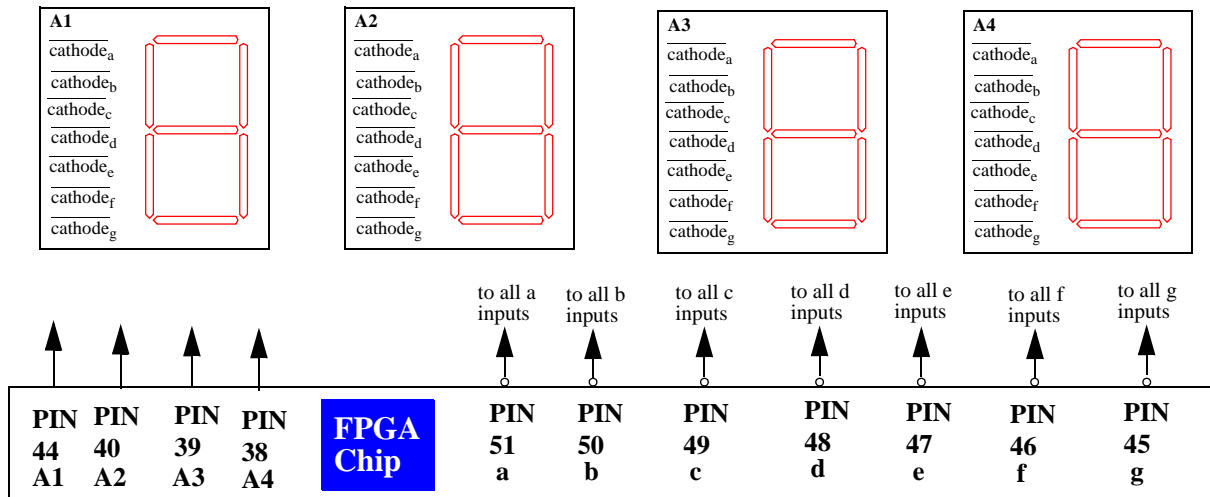
A 7-segment display device



A typical LED segment

#### 4.1. Introduction to 7-Segment Display Usage on the Digilent FPGA Board

The Digilent FPGA board has four 7-segment displays operated in the common anode form. This could mean the FPGA chip has 28 outputs for the segments and four more for the anodes. However, the FPGA chip has only 84 pins, thus it is impossible to reserve 32 pins for the displays. The solution implemented by the Digilent board designer is to have 11 outputs to control all four 7-segment displays as shown below :

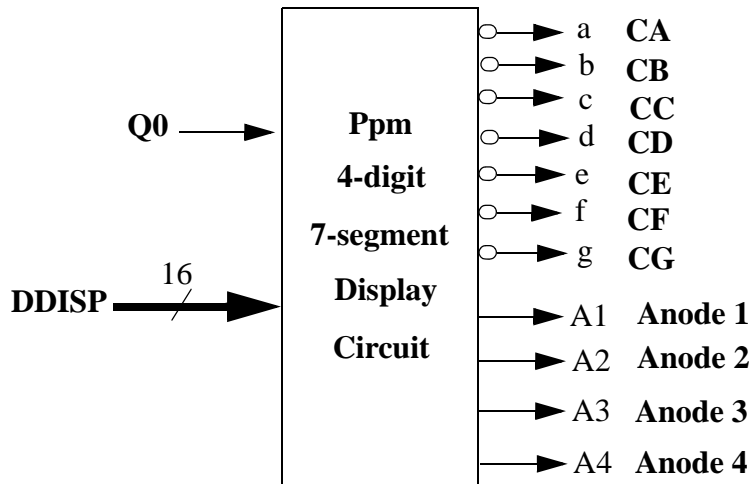


The Digilent FPGA board 7-Segment display organization

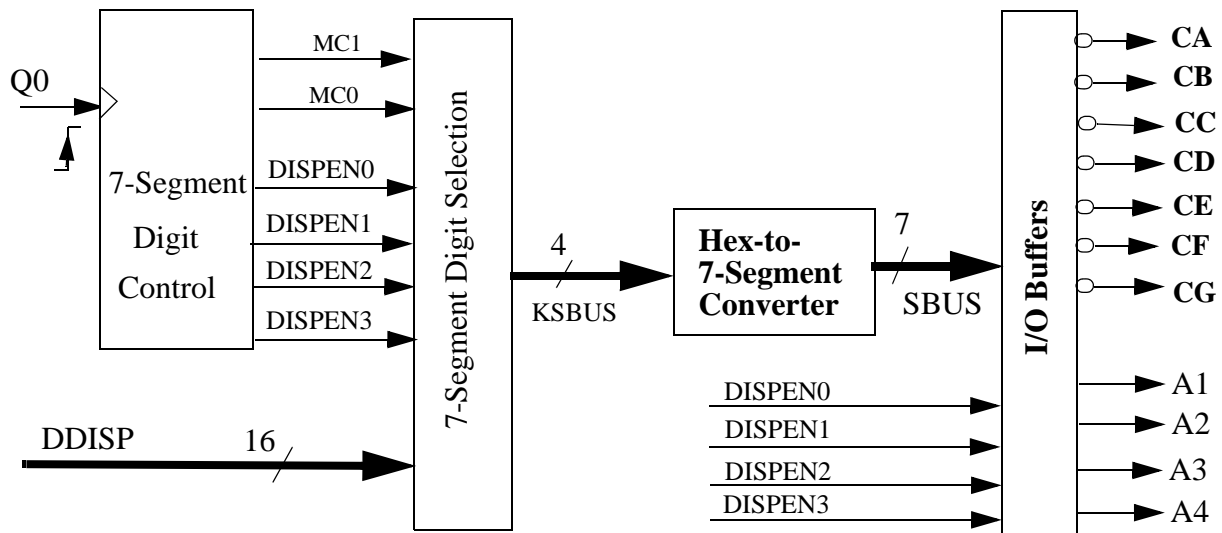
There is a circuit on the FPGA board which supplies +5v to all the anodes, A1, A2, A3 and A4. What one needs is determining which segments have to be on for a specific digit. However, this digit is then displayed on all **four** displays ! If one wants to display that digit on one display, for example, on the rightmost display (A4), the other displays (A1, A2 and A3) must be applied 0 on the anode inputs to blank them and only the A4 (Anode 4) line must be at +5v.

While FPGA pins are saved, an extra circuit is required on the FPGA chip if one wants to display different digits on different displays **simultaneously**. What one has to do is to send seven signals to a display and turn on its anode at the same time. Then swiftly, send seven signals for the next display and turn on its anode. Repeat this round robin display activation at a high speed so that the human eye cannot notice that at any moment only one display actually shows a digit. That is, the 7-segment displays are refreshed at a high-speed when they are on.

The Ppm project second schematic sheet (ppm2.sch, Block 2, the Input/Output Block) has the above mentioned circuit. Below, we discuss a simplified view of the circuit. The black box view of this circuit is as follows :



Signal Q0 is the “clock” whose frequency is the refreshing rate of 192Hz. Signals DDISP represent four Hex digits and so DDISP is 16 bits wide. The circuit in a simplified way consists of four blocks :



The FPGA board has a clock signal named CLK1 at the frequency of 25.175 MHz. It is permanently connected to pin 13 (CLK1 pin) of the FPGA chip. The CLK1 is too fast to be used as the clock signal for refreshing, hence the need for the slower Q0 signal. Q0 is generated by a **frequency divider** in Block 2. The frequency divider is a series of counters in the Timing Subblock. It divides the 25.175 MHz clock signal, by using two 16-bit Xilinx Design Block (XDB) counters (two CB16CE counters), forming a 32-bit counter whose least significant 16 outputs provide a num-

ber of low frequencies needed, including Q0. Thus, the 25MHz clock frequency is divided by up to  $2^{32}$  times. A counter is a sequential circuit. It contains flip-flops. The flip-flops remember what the last count was. Since it is a sequential circuit it also means we cannot use the Karnaugh map method to design it. Sequential circuits and how to design them are introduced in the middle of the semester.

By applying Q0 to the clock input of a 2-bit counter, the 7-Segment Digit Control circuit generates two outputs that constantly count between 0 (00) and 3 (11) : 00, 01, 10 and 11. Each count is for a specific display. If the count is 00, display A4 is activated (the rightmost display). When the count is 01, display A3 is activated and so on. The counter clock input, Q0, indicates when the count has to change. It “triggers” the counter. That is, at every positive edge (low-to-high transition) of Q0, the counter changes the count. The circuit outputs four DISPEN signals that turn on one of the displays at any moment. For that, only one of the DISPEN signals is one at a time. Each DISPEN is connected to a display anode input to turn the display on and off. This round robin refreshing is done at the high speed of Q0 so that the human eye cannot notice the refreshing.

Outputs A1, A2, A3 and A4 are also determined in this circuit by MC1 and MC0 signals such that at any moment only one of them is 1, turning on the corresponding display. These two inputs are connected to a 2-to-4 decoder (D2\_4E) whose four outputs are eventually output as DISPEN0-DISPEN3 and then as A1 through A4 :

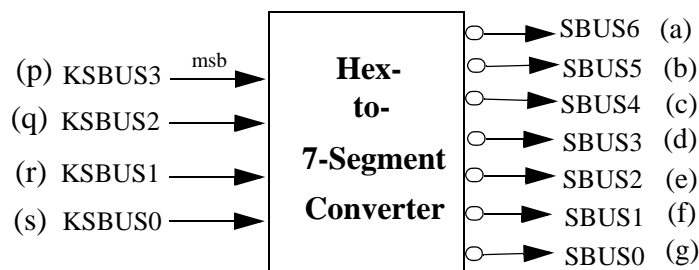
| MC1, MC0 value | D2_4E active output | Display selected |
|----------------|---------------------|------------------|
| 0              | DISPEN0             | A4               |
| 1              | DISPEN1             | A3               |
| 2              | DISPEN2             | A2               |
| 3              | DISPEN3             | A1               |

The 7-Segment Digit Selection circuit selects one of the four digits from DDISP and outputs it on KSBUS to the Hex-to-7-segment converter circuit. The circuit uses a 4-bit 4-to-1 MUX to select a digit where MC1 and MC0 signals are input to the MUX as select signals. The MC1 and MC0 select from DDISP in a round-robin fashion to give the impression that the four displays show digits continuously :

| Bits selected and connected to KSBUS3, KSBUS2, KSBUS1 and KSBUS0 | Display |
|--|---------|
| DDISP3, DDISP2, DDISP1, DDISP0                                   | A4      |
| DDISP7, DDISP6, DDISP5, DDISP4                                   | A3      |
| DDISP11, DDISP10, DDISP9, DDISP8                                 | A2      |
| DDISP15, DDISP14, DDISP13, DDISP12                               | A1      |

## 4.2. The Hex-to-7-Segment Converter

The Hex-to-7-segment decoder (conversion) circuit has four inputs, representing a digit, and 7 outputs. The block view of the converter is as follows :



Since there are only four inputs, deriving the input-output relationship for the seven outputs is not difficult, except that the large number of outputs requires seven Karnaugh maps, a long simplification process.

Textually, we can describe the input-output relationship as follows :

**a** is active (0) when KSBUS is 0, 2, 3, 5, 6, 7, 8, 9, 10, 12, 14, or 15 and inactive (1) when K is 1, 4, 11 or 13

**b** is active (0) when KSBUS is 0, 1, 2, 3, 4, 7, 8, 9, 10, or 13 and inactive (1) when K is 5, 6, 11, 12, 14 or 15

**c** is active (0) when KSBUS is 0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, or 13 and inactive when K is 2, 12, 14 or 15

**d** is active (0) when KSBUS is 0, 2, 3, 5, 6, 8, 11, 12, 13, or 14 and inactive when K is 1, 4, 7, 9, 10 or 15

**e** is active (0) when KSBUS is 0, 2, 6, 8, 10, 11, 12, 13, 14, or 15 and inactive when K is 1, 3, 4, 5, 7 or 9

**f** is active (0) when KSBUS is 0, 4, 5, 6, 8, 9, 10, 11, 12, 14, or 15 and inactive when K is 1, 2, 3, 7 or 13

**g** is active (0) when KSBUS is 2, 3, 4, 5, 6, 8, 9, 10, 11, 13, 14, or 15 and inactive when K is 0, 1, 7 or 12

One can immediately convert the above textual input-output relationships to minterm lists :

$$a(p,q,r,s) = \sum m(1,4,11,13)$$

$$e(p,q,r,s) = \sum m(1,3,4,5,7,9)$$

$$b(p,q,r,s) = \sum m(5,6,11,12,14,15)$$

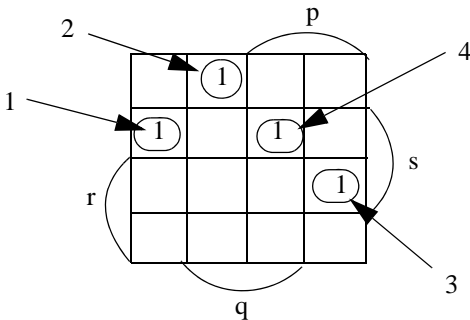
$$f(p,q,r,s) = \sum m(1,2,3,7,13)$$

$$c(p,q,r,s) = \sum m(2,12,14,15)$$

$$g(p,q,r,s) = \sum m(0,1,7,12)$$

$$d(p,q,r,s) = \sum m(1,4,7,9,10,15)$$

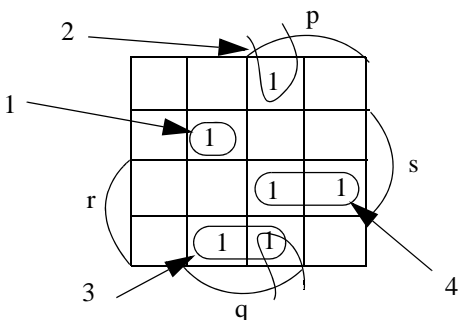
One can implement the converter by using seven gate networks for which seven switching expressions have to be determined. By using the K-map technique, the expressions are easy to obtain. In order to save time, we will obtain SOP expressions. Students are asked to obtain also the minimal POS expressions and compare the amount of complexities. Below, we give the Karnaugh Maps of the five outputs to obtain their minimal expressions. Students will obtain the remaining two expressions themselves :



$$a(p,q,r,s) = \sum m(1,4,11,13)$$

$$a(p, q, r, s) = \bar{p} \bar{q} \bar{r} s + \bar{p} q \bar{r} \bar{s} + p \bar{q} r s + p q \bar{r} \bar{s}$$

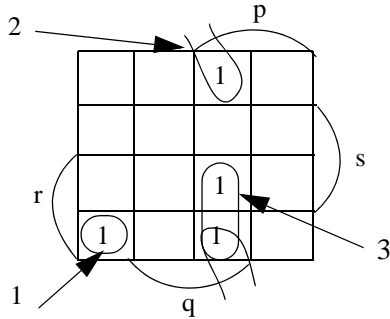
|     |     |     |     |
|-----|-----|-----|-----|
| 1   | 2   | 4   | 5   |
| epi | epi | epi | epi |
| ①   | ④   | ⑪   | ⑬   |



$$b(p,q,r,s) = \sum m(5,6,11,12,14,15)$$

$$b(p, q, r, s) = \bar{p} q \bar{r} s + p q \bar{s} + q r \bar{s} + p r s$$

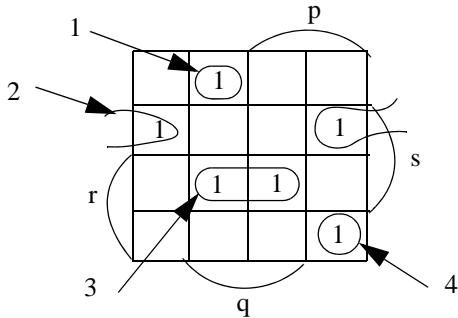
|     |     |     |     |
|-----|-----|-----|-----|
| 1   | 2   | 3   | 4   |
| epi | epi | epi | epi |
| ⑤   | ⑫   | ⑥   | ⑮   |



$$c(p,q,r,s) = \sum m(2,12,14,15)$$

$$c(p, q, r, s) = \bar{p} \bar{q} r \bar{s} + p q \bar{s} + p q r$$

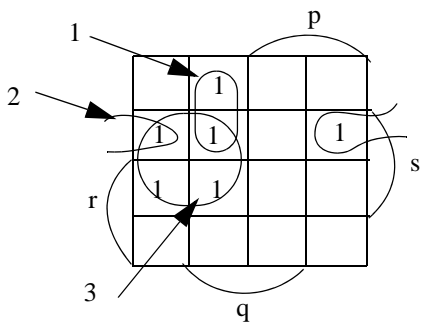
|     |  |      |                            |
|-----|--|------|----------------------------|
| 1   | 2  | 3    |                            |
| epi | epi  | epi  | Output b has the same term |
| (2) | (12)   | (15) |                            |
|     | <span style="background-color: blue; color: blue;">████████</span> |      |                            |



$$d(p,q,r,s) = \sum m(1,4,7,9,10,15)$$

$$d(p, q, r, s) = \bar{p} q \bar{r} \bar{s} + \bar{q} \bar{r} s + q r s + p \bar{q} r \bar{s}$$

|  |  |      |      |                            |
|--|--|------|------|----------------------------|
| 1  | 2  | 3    | 4    |                            |
| epi  | epi  | epi  | epi  | Output a has the same term |
| (4)  | (1)  | (7)  | (10) |                            |
| <span style="background-color: blue; color: blue;">████████</span> | (9)  | (15) |      |                            |
|  | <span style="background-color: blue; color: blue;">████████</span> |      |      |                            |



$$e(p,q,r,s) = \sum m(1,3,4,5,7,9)$$

$$e(p, q, r, s) = \bar{p} q \bar{r} + \bar{q} \bar{r} s + \bar{p} s$$

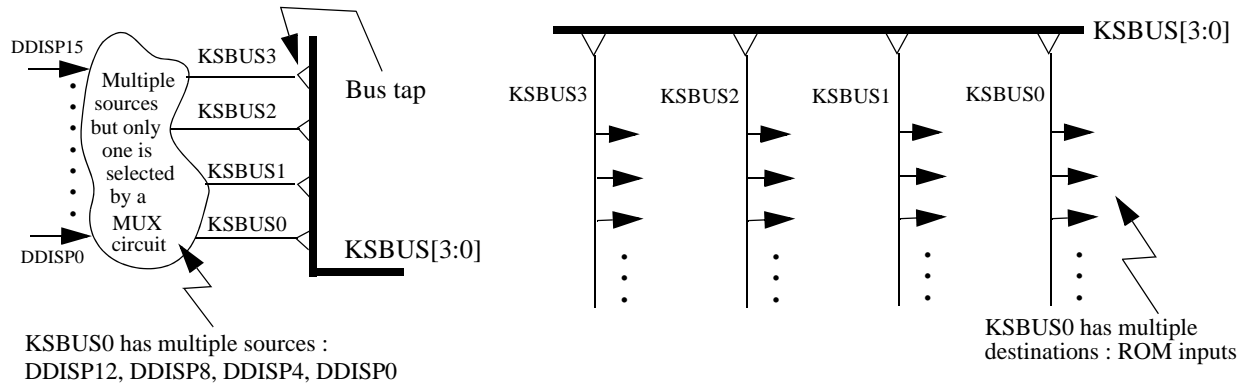
|     |  |     |                            |
|-----|--|-----|----------------------------|
| 1   | 2  | 3   |                            |
| epi | epi  | epi | Output d has the same term |
| (4) | (9)  | (3) |                            |
|     | <span style="background-color: blue; color: blue;">████████</span> | (7) |                            |

## 5. The Xilinx Bus Structure

Real life circuit schematics require a large number of wires drawn to connect components. Often, many of these wires are logically related to each other. For example, a bundle of eight wires may represent an Unsigned Binary number. In order to reduce the amount of work and the visual complexity of the schematic further, **busing** concept can be used. Busing reduces the number of wires drawn.

A bus is a set of lines. A bus is shared. It has multiple sources and one or more destinations. At any moment only **one** source can be on the bus (only **one** value can be on the bus) while multiple destinations can receive the same value from the bus. Busing in a schematic means that we bundle together wires that are related to each other so that we draw just **one** (thick) bus wire, instead of several (thin) wires. We give a name to this bus. As an example, consider the 7-Segment Digit Selection circuit with four outputs. The four outputs are certainly related to each other as they indicate a digit, so we form a bus named "KSBUS" with four wires.

Note that Xilinx calls **all** bundles of wires buses. It is **not** correct. If a Xilinx bus has a single source, it is just a bundle of wires, **not a bus**. In addition, the Xilinx software classifies buses into input, output, bidirectional and none. All the buses we use are type "**none**" this semester. As shown below, KSBUS is a bus since it has four sources. i.e. multiple sources are connected to it. However only one value (the value of a source) is on it at a time. The selection of one source from multiple ones is done by a 4-bit 4-to-1 MUX in the "cloud" in the figure. Again, KSBUS is a type "**none**" bus.



Note that there is no physical connection from the left KSBUS bus to the right KSBUS bus. The software automatically “connects” them since they have identical names. Also, the bus concept makes it easier to use blocks of input buffers, input pads, output buffers and output pads : IBUF4, IPAD4, OBUF4, OPAD4, etc.

## 6. Using Xilinx ROM Blocks

The Xilinx software has only two ROM XDBs : 16x1-bit and 32x1-bit. These ROMs are small, limiting their usage for real life circuits. That is, if a larger ROM space is needed, one can implement it by using Xilinx ROMs, but, they would occupy a substantial area on the screen.

The Xilinx software allows storing the ROM content by using the INIT parameter :

- Right click on the ROM
- Select “**Symbol Properties...**”
- In the **Parameter** section, select “**INIT**”
- In the description enter the content, by converting bits to **Hex** digits, starting from location **15** or location **31**.

ROMs can be used to implement combinational circuits. We show how to use Xilinx ROM blocks by working on the Hex-to-7-segment converter circuit which is in Block 2, the Input/Output Block. The Hex-to-7-segment circuit is implemented by using Xilinx ROM blocks, instead of gate networks. Since there are only four inputs and seven outputs, we need a 16x7-bit ROM. But, the best matching Xilinx ROM is 16x1-bit. Thus, we use seven of these and program them with the values obtained from the minterm lists above which are shown below again:

$$a(p,q,r,s) = \sum m(1,4,11,13)$$

$$e(p,q,r,s) = \sum m(1,3,4,5,7,9)$$

$$b(p,q,r,s) = \sum m(5,6,11,12,14,15)$$

$$f(p,q,r,s) = \sum m(1,2,3,7,13)$$

$$c(p,q,r,s) = \sum m(2,12,14,15)$$

$$g(p,q,r,s) = \sum m(0,1,7,12)$$

$$d(p,q,r,s) = \sum m(1,4,7,9,10,15)$$

Each ROM implements one of the “a” through “g” outputs. Thus, the programming of ROM involves storing one vertical column of the truth table on one ROM. For example the content of the ROM that generates output “a” is “2812” in terms of Hex digits. The leftmost Hex digit shows the content of ROM locations 15, 14, 13, and 12. Therefore, “2812” in Hex corresponds to “0010 1000 0001 0010” in binary and to locations 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 and 0 in the ROM, respectively. By following this example, students can then program the remaining ROMs by themselves.

## CONTACTS :

1) Students can see the professor or TAs about the course, homework, simulator and lab experiments.

2) Professor's contact information :

Room : 114 LC

(718) 260-3101

Fax : (718) 260-3609

haldun@photon.poly.edu

**Open-door** policy to see the professor. If the door is closed, he might be teaching.

Present in the lab : Tuesday (9-12), Thursday (9-12), Friday (1-4) and (4-7)

3) Below are the contact information and assignments of the TAs :

**Minyu Li** : mli10@students.poly.edu

- Present in lab session : Friday : 1 - 3:50 (C)
- Lab monitor : Friday : 11 - 1
- Grading the homework

**Vivek S. Pujeri** : vpujer01@students.poly.edu

- Present in lab session : Friday : 1 - 3:50 (C)
- Lab monitor : Monday : 4 - 5

**Ajay Prabhakar** : aajayp01@students.poly.edu

- Present in lab sessions : Tuesday : 9 - 11:50 (B) ; Thursday : 9 - 11:50 (A)

**Hui Wang** : hwang07@students.poly.edu

- Present in lab sessions : Friday : 1 - 3:50 (C) ; Friday : 4 - 6:50 (D)
- Lab monitor : Wednesday : 11 - 12

**Tung-Hua Wu** : two04@students.poly.edu

- Present in lab sessions : Thursday : 9 - 11:50 (B)
- Lab monitor : Thursday : 12 - 2

**Chengzhi Zong** : czong01@students.poly.edu

- Present in lab session : Friday : 1 - 3:50 (C)
- Lab monitor : Tuesday : 12 - 2

4) All handout and lab files are at the course web site : **<http://cis.poly.edu/cs2204>**

5) The Open Lab hours with a TA present start in week 4, on Thursday, September 30, 2010 :

**Monday** : 4 - 5 ; **Tuesday** : 12 - 2 ; **Wednesday** : 11 - 12 ; **Thursday** : 12 - 2 ; **Friday** : 11 - 1

6) Students are asked to give feedback about the announced lab hours in case the hours are not sufficient and/or need to be rescheduled.

7) For any problem, students can contact the professor and the TAs.

When short-term problems are encountered in PC labs, students are advised to contact : help@duke.poly.edu or (718) 260- 3123 or go to Room : 337 RH.

For CS2204 lab related issues and to have the lab open, students need to contact the CIS lab supervisor Mr. Keni Yip at (718) 260-3023, keni@poly.edu. His office is 225RH