ECE2277A Lab 3: Adder/Subtractor Circuits

Prepared by John McLeod

Objectives

This lab is an exercise in designing and constructing a circuit that can add and/or subtract unsigned 4-bit numbers. The objectives of this lab are:

1. To gain familiarity with designing a circuit based on a few design requirements,
2. To design custom symbols in Quartus,
3. To construct a 4-bit circuit for adding and subtracting using custom symbols,
4. To implement and test this circuit in hardware.

This lab will be done using Quartus and implemented in hardware, employing the techniques described in the previous labs.

Summary

A summary of the deliverables for this lab are enumerated below.

1. Derive the logic circuit for displaying a minus sign or a `1` on the 7-segment display panel `HEX1`, based on the output from BCD addition.
2. Design a circuit in Quartus to implement a 4-bit unsigned binary or BCD adder/subtractor.
3. Implement this circuit on the DE10-Standard FPGA development board.
4. Demonstrate to a TA that your circuit works correctly.

You **must** derive the logic circuit in step 1 *before* coming to the lab. Furthermore, constructing the circuit in Quartus for lab 3 may take some time, you are advised to do some (or all) of the circuit construction before coming to the lab.

Grading Scheme

The lab is graded as follows, out of a total of 50.

Section	Details	Mechanism	Grade
Circuit Design	BCD Correction Logic	Show written work to TA	5
Quartus Circuits	Binary Adder/Subtractor	Show Quartus Design to TA	10
	BCD Adder/Subtractor	Show Quartus Design to TA	5
FPGA Hardware	Binary Adder/Subtractor Implementation	Demonstrate to TA	15
	BCD Adder/Subtractor Implementation	Demonstrate to TA	5
Concepts	Answer Questions	Discuss exercise with TA	10

Note that there is a single circuit design of this project, however the grade is subdivided into part for the basic binary adder/subtractor (the circuit is given, above) and part for the completed BCD adder/subtractor.

- If you finish the entire design, you only need to demonstrate the BCD adder/subtractor to obtain all the marks. (We will assume if the BCD adder/subtractor works, then the underlying binary adder/subtractor also works.)
- If you are unable to finish the entire design, please at least try to implement a binary adder/subtractor to obtain some marks.

Suggestions for ways to partially complete this lab but still maximize your grade are found in *Lab Procedure*, below.

Quartus Tips

This lab requires constructing a circuit that will use a large number of logic elements — significantly more than the other lab exercises thus far. However a lot of this circuit will be repeated copies of the same block of logic elements. You can simplify the design approach by constructing your own symbols.

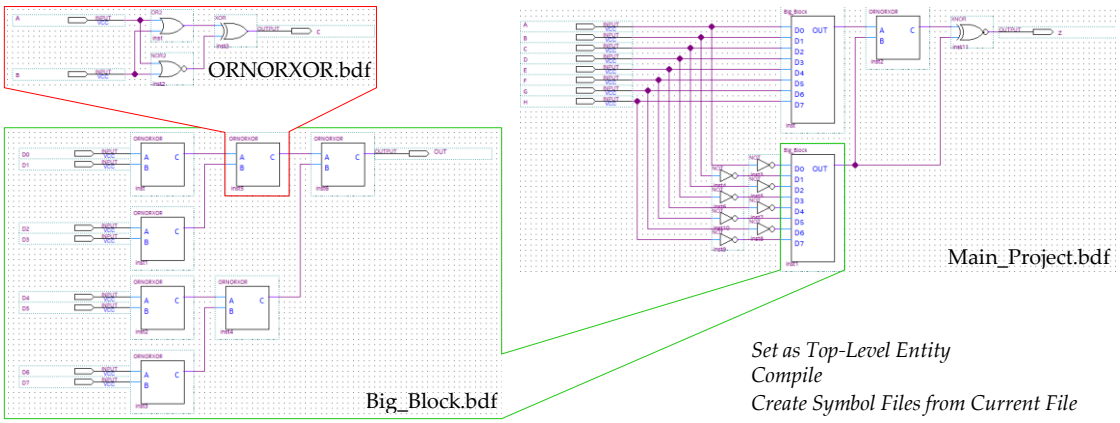
To construct your own symbolic logic blocks in Quartus, do the following:

1. In your project, create a new block diagram/schematic file with a representative name (for example, "HA.bdf" for a half-adder). Make sure this file is saved in the same directory as the project.
2. Draw the logic circuit for that block, complete with input and output pins. Don't do any pin assignments, though.
3. Go to the *Project* menu and choose *Set as Top-Level Entity*.
4. Compile your project, and fix any errors that may occur.
5. Go to the *File* menu, and under *Create/Update* choose *Create Symbol Files from Current File*.
6. Go back to your main schematic file. Go back to the *Project* menu and choose *Set as Top-Level Entity* to return the main schematic file to be the primary design for your project.

Now when you place logic elements in your main schematic file, you should see a "Project" folder above the usual ".../quartus/libraries/" folder. The name of your custom block file should be available in that folder.

It is possible to chain these block diagrams together. For example (hint, hint!), you can make a single-bit half-adder logic block, then combine these to make a single-bit full-adder logic block, then combine for of those to make a 4-bit full adder/subtractor block; just place the the intermediate blocks in new block diagram/schematic files and compile each as the *Top-Level Entity*. Finally, when you have created all the necessary blocks, you can draw your main schematic file. If it all works correctly, double-clicking on your own custom blocks used in a circuit should open up your original block diagram file that the element is based on.

A video tutorial of this process is available on OWL, demonstrating constructing the famous "or-nor-xor" logic element and a "big block" logic element. Please note that neither of these are useful for the current lab, they are just used to demonstrate the process.



Lab Description

In this lab you need to design and implement a BCD adder/subtractor, that can read input from the slider switches and outputs the sum/difference to the seven-segment displays. For two BCD inputs $0 \leq A \leq 9$ and $0 \leq B \leq 9$, the possible outputs range from -9 to 18. Your circuit must adhere to the following design requirements:

1. Accept two 4-bit numbers $A = a_3a_2a_1a_0$ and $B = b_3b_2b_1b_0$, and a control input M as inputs. These inputs should be controlled by slider switches according to the table below.

Slider Switch	SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
Input	a_3	a_2	a_1	a_0	b_3	b_2	b_1	b_0	—	M

   The control input M selects the operation: if $M = 0$, addition is performed. If $M = 1$, subtraction is performed.

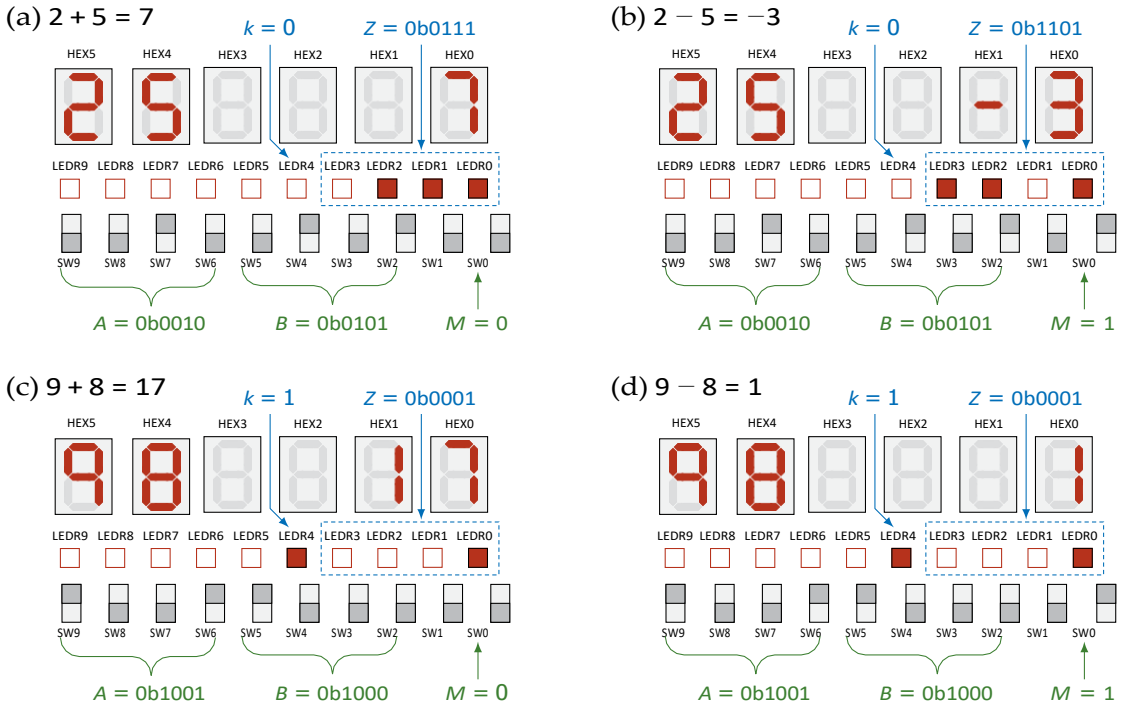
2. The two inputs A and B should be directly displayed, each on a seven-segment display panel. Use HEX5 to display A, and use HEX4 to display B.

3. Use your own implementation of a 4-bit *binary* unsigned adder/subtractor, created as a custom block diagram in Quartus, to perform the required mathematical operations.

4. The four-bit *binary* number $Z = z_3z_2z_1z_0$ and *binary* carry-out k that results from the math operation ($A + B$ if $M = 0$, $A - B$ if $M = 1$) should be displayed on the red LEDs according to the table below.

LED	LEDR9	LEDR8	LEDR7	LEDR6	LEDR5	LEDR4	LEDR3	LEDR2	LEDR1	LEDR0
Output	—	—	—	—	—	k	z_3	z_2	z_1	z_0

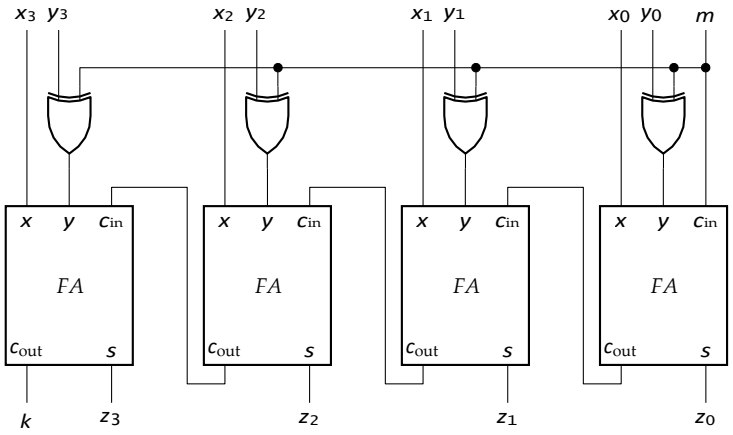
5. Finally, The *BCD* output of the math operation ($A + B$ if $M = 0$, $A - B$ if $M = 1$) should also be displayed on the seven-segment display panel. Use HEX0 to display the single-digit BCD output of the math operation. Use HEX1 as the "ten's place" for large addition (i.e. if $A + B \geq 10$), or the "minus sign" if the subtraction is negative (i.e., for $A - B$ if $B > A$).



The image above shows four examples of how this circuit should operate. The major component of this circuit is a 4-bit unsigned *binary* adder/subtractor, and of course some 7447 BCD-to-7Seg blocks to drive the seven-segment display panels. As mentioned above, you must implement a 4-bit unsigned binary adder/subtractor on your own as a custom block diagram in Quartus for this circuit (the symbol libraries in Quartus do have various adder/subtractor blocks already — you may *not* use those).

- You are welcome to design a fast adder/subtractor with a carry-lookahead generator, but that is not necessary. Instead you probably want to implement a simpler adder/subtractor using chained full adder blocks.
- You can implement a *signed* adder/subtractor if you wish, but again that is not necessary as the overflow output will be ignored in this circuit.

The most straight-forward implementation of a 4-bit adder is probably the circuit shown below. It is probably best to use a custom block diagram in Quartus (see "Quartus Tips", above) to implement the full adders: these, in turn, could be constructed from half-adder blocks; or could be constructed using any of the numerous approaches outlined in the lesson notes (two-level circuit, decoder circuit, etc.).



Because your BCD adder/subtractor will probably need more than one of these 4-bit unsigned binary adder/subtractors, you should make the entire circuit above as a custom block in Quartus. The BCD adder/subtractor circuit you need to design is basically an extension of the BCD adder designed in the lectures. In Lesson 4 we developed a BCD *adder* from the 4-bit binary adder to have the correct BCD carry-out C given by:

$$C = k + z_3z_2 + z_3z_1$$

With this, the correct BCD sum $S = s_3s_2s_1s_0$ is the algebraic sum $S = Z + \texttt{0b0}CC\texttt{0}$ —since if $Z \geq 10$ the correct BCD sequence is $S = Z + 6$ with $C = 1$, and if $Z < 10$ the correct BCD sequence is $S = Z$ with $C = 0$. If we also want a BCD *subtractor*, we need to modify this design.

- If inputs A and B are both valid BCD values($0 \leq A, B \leq 10$), then if $A \geq B$ we always have $0 \leq A - B < 10$, so the result is *always* a valid BCD value.
- The general approach of calculating $A - B$ using $A + (\bar{B} + 1)$, where $\bar{B}$ is the logical complement (or one's complement) of B (so $\bar{B} + 1$ is the two's complement of B) is still valid, even in BCD ($\bar{B}$ may not be a valid BCD value, but if $A > B$ then $A+(\bar{B} + 1)$ is the correct answer).
- If A and B are both valid BCD values, then if $A < B$ the result $A - B = A + (\bar{B} + 1)$ is the *two's complement* of the correct BCD value for the negative answer:

  $$2 - 7 = 0 \quad \rightarrow \quad \texttt{0b0010} - \texttt{0b0111} = \texttt{0b0010} + \texttt{0b1001} = \texttt{0b1011}$$

  Although `0b1011` is not a valid BCD value, the two's complement of this is `0b0101` = 5, and of course $2 - 7 = -5$.
- Recall that a correct subtract generates a borrow of 0. Since borrow is the complement of carry-out, if $A > B$ we will have $k = 1$. Consequently, our circuit must do some slightly different things to generate the correct output. The possible cases are listed in the table below.

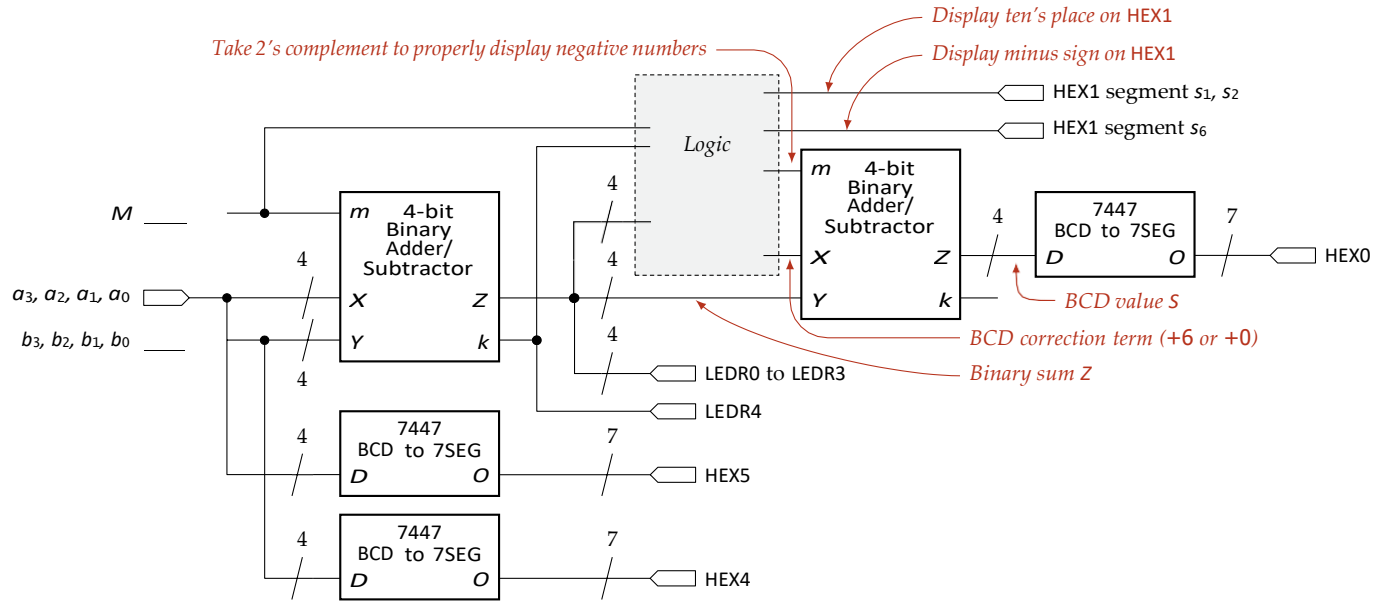
Operation	Control M	Input Condition	Carry-out C	Sum S
Addition	$M = 0$	$A + B < 10$	$C = 0$	$S = Z$
Addition	$M = 0$	$A + B \geq 10$	$C = 1$	$S = Z + 6$
Subtraction	$M = 1$	$A \geq B$	$C = 1$	$S = Z$
Subtraction	$M = 1$	$A < B$	$C = 0$	$S = -\left(\bar{Z} + 1\right)$

The last possibility, for $A < B$, is for the case where we wish to display the sum as a negative number.

Using this table it is possible to include M as an input variable, and then modify the original logic expression $C = k + z_3z_2 + z_3z_1$, and the original logic determining when to add 6 to Z, such that the correct results are also obtained for subtraction. In principle C is now a Boolean function of six variables (M, k, z_3, z_2, z_1, z_0), but we don't need to solve a tedious 6-variable K-map to deduce how to include the influence of M on the circuit (you **are not** graded on the efficiency of your circuit, only on whether or not it works).

- The original BCD adder design used two 4-bit adder circuits, one to compute the binary sum Z and one to obtain the correct BCD sum S.
- In the circuit derived in the lessons, we connected the first binary sum Z to the X input of the second adder, and 0b0CC0 to the Y input.
- Our BCD adder/subtractor will sometimes require the two's complement of the sum S: if we are computing $A - B$ for $B > A$, we want the seven-segment displays to show $-|S|$. Taking the two's complement of S can be easily done if we connect the binary sum Z to the Y input of the second adder. We will therefore use the X input of the second adder for the "BCD correction term" (+6 or +0, depending on the circumstances).

The basic block diagram of the circuit we need to design is given below.



In addition to implementing the 4-bit binary adder/subtractor block, you need to design the combinational logic (shown in grey and labelled "*Logic*" in the above diagram) to generate the correct output on the seven-segment displays. This logic was discussed above, it accepts the binary carry-out k and binary sum Z from the first adder/subtractor, and the operation control input M. It outputs the correction term (either +6 or +0) to generate the correct BCD value, a control output indicating when a subtraction operation is negative, and the control outputs to display a negative sign (i.e., turn on segment s_6 on a seven-segment display) or a 1 in the "ten's place" (i.e., turn on segments s_1 and s_2 on a seven-segment display — in principle you could use another 7447 for this, but since this circuit will only have a 1 or nothing in the "ten's place" it is probably easier to just directly control the segments).

To summarize:

- When computing $A + B$ the result is < 10, we have $S = Z$. HEX0 should display the digit Z, and HEX1 is blank.
- When computing $A + B$ the result is ≥ 10, we have $S = Z + 6$. HEX0 should display the digit $Z + 6$, and HEX1 should display the digit 1.
- When computing $A - B$ and $A \geq B$, we have $S = Z$. HEX0 should display the digit Z, and HEX1 is blank.

- However, when computing $A - B$ with $A < B$, we have $S = Z$ and $S < 0$. To correctly display a negative number, HEX0 should display the two's complement, i.e. $(\bar{Z} + 1)$, and HEX1 should show a minus sign. Note that the two's complement of Z can be obtained by the 4-bit binary adder/subtractor by calculating $0 - Z$.

In the circuit above, the output S from the second adder is always the correct BCD value to show on the seven-segment display: we have either $S = Z$, $S = Z + 6$, or $S = 0 - Z$ depending on the circumstances. Note that the BCD carry-out C is not an explicit output in this circuit, but is a useful intermediate for determining whether to display a 1, a minus sign, or nothing on HEX1, and for determining how to calculate S from Z.

Lab Preparation

Please design the logic required to generate m and X for the second-stage 4-bit binary adder/subtractor, and s_1, s_2, s_6 for the HEX1 seven-segment display from k, Z, and M.

- A good approach is to start with the logic for the BCD addition carry-out $C = k + z_3z_2 + z_3z_1$, and think how to combine C with M and k to control subtraction.
- HEX1 only displays a "minus sign" when the operation is subtraction, and the result is negative. Likewise, HEX1 only displays a 1 in the "ten's place" when the operation is addition and the result is ≥ 10. Based on M, k, and C it should be easy to figure out when these conditions apply.

You **must** complete this part of the lab exercise *before* coming to your lab section, and bring in some evidence of your work to show the TA.

Lab Procedure

Implement a 4-bit binary adder/subtractor as a custom block symbol, and use that to complete the BCD adder/subtractor circuit shown in the block diagram above. You obviously also need to implement the intermediate logic from the Lab Preparation between the two 4-bit binary adder stages.

Program this circuit to the FPGA board and verify that the correct output is shown.

Partial Completion: This lab exercise is a significantly more complicated design than previous exercises. It is fairly straightforward to complete, but does require a lot of design work. It also has a lot of input/output pins that need to be assigned.

- It is strongly advised that you complete some Quartus design work *before* coming into the lab. You can bring the Quartus project into the lab, or at least bring the pin assignments (.QSF), block diagram schematics (.BDF), and any custom block symbols (.BSF) files and import them into a blank project.

In the event that you have trouble completing the entire lab exercise, please take note of the grading scheme to see how to maximize your grade.

1. At *minimum* you should implement a 4-bit *binary* adder/subtractor that accepts inputs from the slider switches and sends the binary output Z, k to be displayed on LEDs.
2. If you are having difficulty deriving the logic for the BCD adder/subtractor, then you can at least implement a BCD adder (only) using the design shown in the lesson notes.
3. Extending the BCD adder design to an adder/subtractor is not very difficult, so implementing subtraction capability is the next step.
4. Finally, connecting A, B, S, and the minus sign/ten's place digit to seven-segment display panels is the last step. This is conceptually easy, but assigning all the pins can take some time.

If you are unable to complete the entire lab exercise, and want to maximize partial credit, make sure you have *something* that you can program to the FGPA board to demonstrate a working circuit (worth an additional 10 marks). The 4-bit binary adder/subtractor only requires LED output and slider switch input, so that is pretty easy. If you don't have time to connect all the seven-segment display panels, then having the BCD code shown on the remaining 5 LEDs (LEDR5 to LEDR9 for S and carry-out C) is probably the easiest way of showing that your design works in hardware.

DE10-Standard Pin Assignments

The pin assignments for the ten slide switches are given below.

Hardware Label	Quartus Pin	Hardware Label	Quartus Pin
SW0	PIN_AB30	SW5	PIN_V25
SW1	PIN_Y27	SW6	PIN_AC28
SW2	PIN_AB28	SW7	PIN_AD30
SW3	PIN_AC30	SW8	PIN_AC29
SW4	PIN_W25	SW9	PIN_AA30

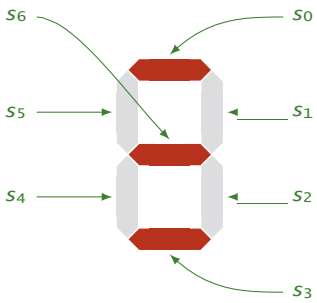
The pin assignments for the ten red LEDs are given below. The LEDs are all **active-high**: an output of 1 will light up the LED, an output of 0 will leave the LED dark.

Hardware Label	Quartus Pin	Hardware Label	Quartus Pin
LEDR0	PIN_AA24	LEDR5	PIN_AF25
LEDR1	PIN_AB23	LEDR6	PIN_AE24
LEDR2	PIN_AC23	LEDR7	PIN_AF24
LEDR3	PIN_AD24	LEDR8	PIN_AB22
LEDR4	PIN_AG25	LEDR9	PIN_AC22

The pin assignment for all six seven-segment display panels are given below. You only need HEX0, HEX1, HEX4, and HEX5 for this exercise. The segments on each panel are all **active-low**: an output of 0 will light up that segment, an output of 1 will leave the segment dark.

Hardware Label	Segment	Quartus Pin	Hardware Label	Segment	Quartus Pin
HEX0	s_0	PIN_W17	HEX3	s_0	PIN_Y19
	s_1	PIN_V18		s_1	PIN_W19
	s_2	PIN_AG17		s_2	PIN_AD19
	s_3	PIN_AG16		s_3	PIN_AA20
	s_4	PIN_AH17		s_4	PIN_AC20
	s_5	PIN_AG18		s_5	PIN_AA19
	s_6	PIN_AH18		s_6	PIN_AD20
HEX1	s_0	PIN_AF16	HEX4	s_0	PIN_AD21
	s_1	PIN_V16		s_1	PIN_AG22
	s_2	PIN_AE16		s_2	PIN_AE22
	s_3	PIN_AD17		s_3	PIN_AE23
	s_4	PIN_AE18		s_4	PIN_AG23
	s_5	PIN_AE17		s_5	PIN_AF23
	s_6	PIN_V17		s_6	PIN_AH22
HEX2	s_0	PIN_AA21	HEX5	s_0	PIN_AF21
	s_1	PIN_AB17		s_1	PIN_AG21
	s_2	PIN_AA18		s_2	PIN_AF20
	s_3	PIN_Y17		s_3	PIN_AG20
	s_4	PIN_Y18		s_4	PIN_AE19
	s_5	PIN_AF18		s_5	PIN_AF19
	s_6	PIN_W16		s_6	PIN_AB21

The correlation between segment number and the actual display is shown below. Because the segments are **active-low**, the pattern shown below is realized by the output $s_6 = 0$, $s_5 = 1$, $s_4 = 1$, $s_3 = 0$, $s_2 = 1$, $s_1 = 1$, and $s_0 = 0$.



Note that display segments remain dark if they are not *explicitly* controlled as an output in your design. So for the "minus sign/ten's place" you only need to assign pins to segments s_1, s_2, and s_6:

- To display a 1, set $s_1 = s_2 = 0$ and $s_6 = 1$.
- To display a − for the minus sign, set $s_1 = s_2 = 1$ and $s_6 = 0$.
- All other segments remain dark, and there is no need to include them in your design.