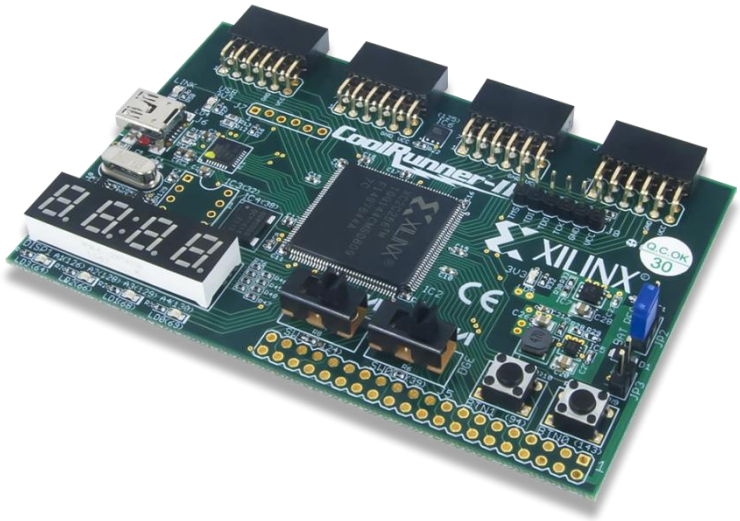
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY – FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

BINARY – BCD CONVERTER



CICT – PULSE AND DIGITAL TECHNIQUES

COURSE 2017 – 2018

Alba Ruiz Gómez and Norma Schulze Giménez

INDEX

1. INTRODUCTION

In the present document we have detailed the analysis and development of a binary to BCD converter. The converter has been implemented using VHDL. To show the converter functionality we used CoolRunner-II CPLD starter board and an additional board with sixteen switches and LEDs, for introduce the binary value and to show the result respectively.

2. UTILITY OF BCD

BCD code is very useful in electronic systems where numeric values must be shown, especially in non-programmer digital systems (without microprocessor or microcontroller).

Some companies such as IBM used it in their products. IBM used the terms binary and BCD decimal codification for a binary code of six bits, with this method numbers, capital letters and special character could be represented. One of the variants of BCD code was used in most of the first computers of IBM, such as IBM1620 and IBM 1400. With the developed of System/360, BCD code was substituted by EBCDIC, of 8 bits.

3. THEORETICAL BASIS

First, it is important to know how BCD code works. These are some of the most important facts of BCD code:

- BCD: Binary-Coded Decimal
- Simplest binary code for decimals digits
- Only encodes ten digits from 0 to 9
- BCD is a weighted code
- The weights are 8, 4, 2, 1 (as binary number)
- There are six invalid code words: 1010, 1011, 1100, 1101, 1110, 1111

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

Figure 1. BCD code

Unfortunately, to do the conversion binary to BCD it cannot be done by separating the binary number into groups of four bits. To make the conversion, it is required to use an algorithm denominated **shift to the left and add three**. This is how this algorithm works:

1. Shift the binary number to the left one bit
2. If some of the digits has a value equal or higher than five, add three
3. Repeat steps 1 and 2 the amounts of bits that the binary number that we want to convert.

To understand the method the best way is with an example. If the number has 9 bits in binary will have 3 digits in BCD. The result will be like this:

BCD			BINARY
Hundreds	**Tens**	**Units**	**Binary number**
(4 bits)	(4 bits)	(4 bits)	(9 digits)

The number of the example will be 417 (110100001) and the algorithm is the following:

OPERATION	**BCD**			**BINARY**
	HUNDREDS (4 BITS)	**TENS (4 BITS)**	**UNITS (4 BITS)**	**BINARY NUMBER (9 BITS)**
Initial state				110100001
Shift to the left (1)			1	10100001
Shift to the left (2)			11	0100001
Shift to the left (3)			110	100001
Add three to the units			1001	100001
Shift to the left (4)		1	0011	00001
Shift to the left (5)		10	0110	0001
Add three to the units		10	1001	0001
Shift to the left (6)		101	0010	001
Add three to the tens		1000	0010	001
Shift to the left (7)	1	0000	0100	01
Shift to the left (8)	10	0000	1000	1
Add three to the units	10	0000	1011	1
Shift to the left (9)	100	0001	0111	

Finally, the result corresponds with 417 in BCD = (0100)(0001)(01111).

In the case of the project, the result will be represented in the displays of the boards, so the maximum number, that can be converted, will be 9999. That means that it will be necessary 4 groups of 4 bits (thousands, hundreds, tens and units).

4. INSTRUMENTS USED

CoolRunner-II CPLD starter board

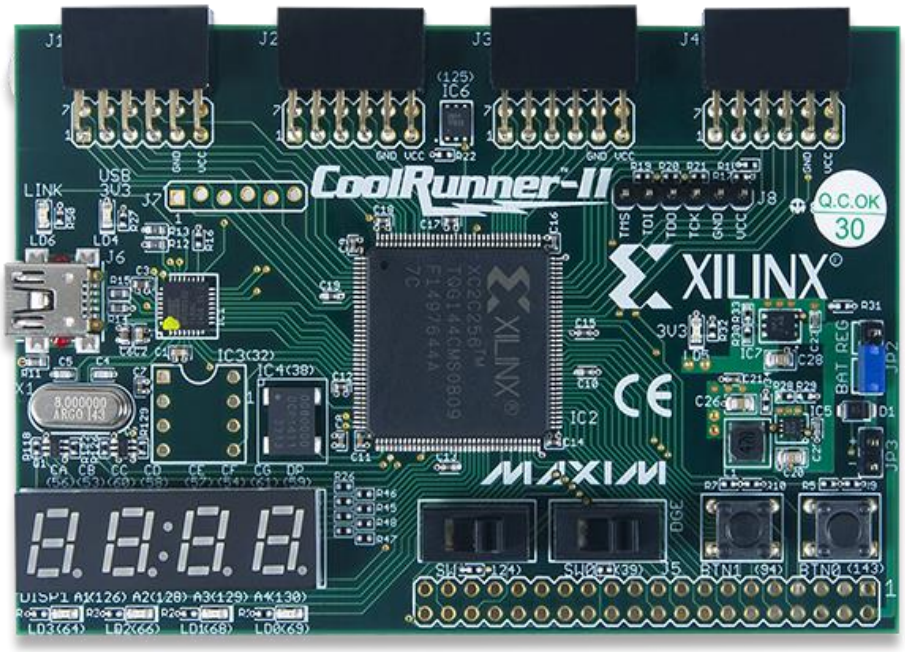


Figure 2. CoolRunner

Expansion switches and LEDs board

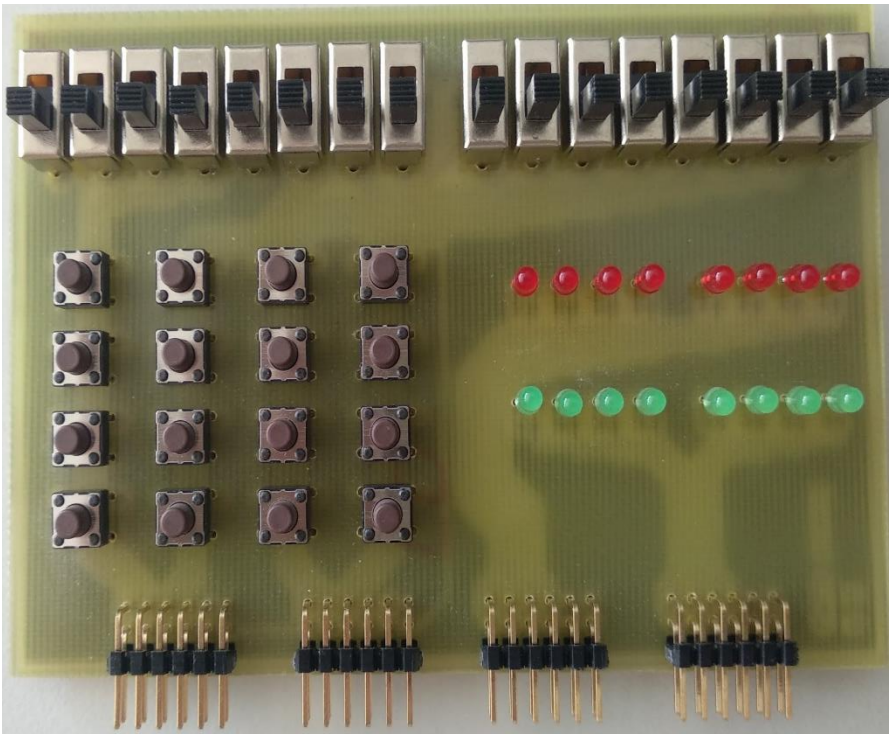


Figure 3. Expansion

5. CONNECTION BOARD

Our application has as input ports 14 input bits corresponding with the switches, the system clock signal and as output we configured the 16 bits LEDs ports and to show the values on the displays we used two ports as outputs, one for choose the display to be shown and other for the relevant segments. The detail of the entity is the following:

```
entity binbcd is
    GENERIC(
        NBITS  : integer :=  14; -- Number of bits of the binary number
        NOUT: integer := 16 -- Number of bits of the BCD number (output).
    );

    PORT(
        SW_EXP: in std_logic_vector(NBITS-1   downto 0); -- Input, switches (Binary)
        CLK: in std_logic;                               -- Clock signal
        LED_EXP: out std_logic_vector(NOUT-1 downto 0);  -- Output,LEDs (BCD)
        D_POS: out std_logic_vector(3 downto 0);         -- Display positions
        D_SEG: out std_logic_vector(6 downto 0)          -- Display segments
       );
end binbcd;
```

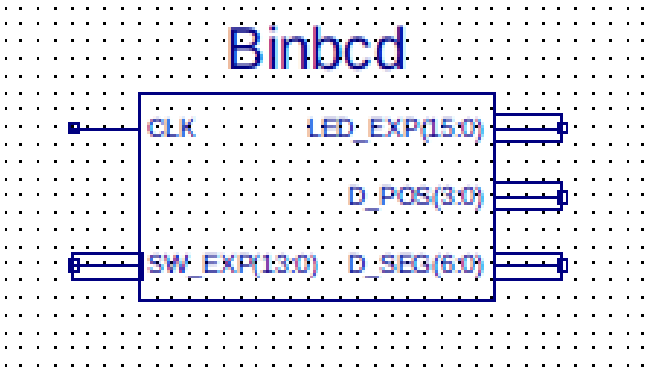


Figure 4. Component

6. IMPLEMENTATION

As we said, the maximum number that we can show in the displays is 9999. To represent this number, we need at least 14 input bits (9999D → 10 0111 0000 1111B). To show this value also in the LEDs, as we now from the BCD algorithm, we need to represent 4 digits by 4 groups of 4 bits, which made a total of 16 LEDs.
The algorithm implementation of the binary BCD converter is the one that follows:

```
process_bcd: process(SW_EXP)
    variable z: STD_LOGIC_VECTOR(NBITS+NOUT-1 downto 0);
begin
    -- Initialization of data to zero
    z := (others => '0');
    -- First three left shifts
    z(NBITS+2 downto 3) := SW_EXP;
    -- Loop for the remaining shifts
    for i in 0 to NBITS-4 loop
        -- Units (4 bits).
        if z(NBITS+3 downto NBITS) > 4 then
            z(NBITS+3 downto NBITS) := z(NBITS+3 downto NBITS) + 3;
        end if;
        -- Tens (4 bits).
        if z(NBITS+7 downto NBITS+4) > 4 then
            z(NBITS+7 downto NBITS+4) := z(NBITS+7 downto NBITS+4) + 3;
        end if;
        -- Hundreds (4 bits).
        if z(NBITS+11 downto NBITS+8) > 4 then
            z(NBITS+11 downto NBITS+8) := z(NBITS+11 downto NBITS+8) + 3;
        end if;
        -- Thousands (4 bits).
        if z(NBITS+14 downto NBITS+12) > 4 then
            z(NBITS+14 downto NBITS+12) := z(NBITS+14 downto NBITS+12) + 3;
        end if;
        -- Shift to the left.
        z(NBITS+NOUT-1 downto 1) := z(NBITS+NOUT-2 downto 0);
    end loop;
    -- Assign z data to our BCD variable.
    num_bcd <= z(NBITS+NOUT-1 downto NBITS);
end process;
```

We can change the number of bits of the input and outputs. This is possible because we used two variables (NBITS, NOUT) that are declared as a generic type in the top of the entity. This fact makes easier to change the number of bits in the algorithm in case we need it.

```
entity binbcd is
    GENERIC(
        NBITS  : integer :=  14; -- Number of bits of the binary number
        NOUT: integer := 16 -- Number of bits of the BCD number (output).
    );
```

The algorithm is described in a process that **depends on the switches** (SW_EXP). That means that every time a switch is changing, the process will be executed. This algorithm is valid for 4 groups of 4 bits maximum, if we need more bits we must add another *if* sentence that cover these range of extra bits.

The result from BCD converter will be shown in two ways: by LEDs and displays. To show the result in the leds only the variable of BCD result (num_bcd) must be assigned

to LED_EXP, the output port of the LEDs.

```
------------------------------------
--    Show BCD value in the LEDs   --
------------------------------------
LED_EXP <= num_bcd;
```

Seven segment display implementations

As we can see in the Figure 1 below, the 7 segment displays are active low.

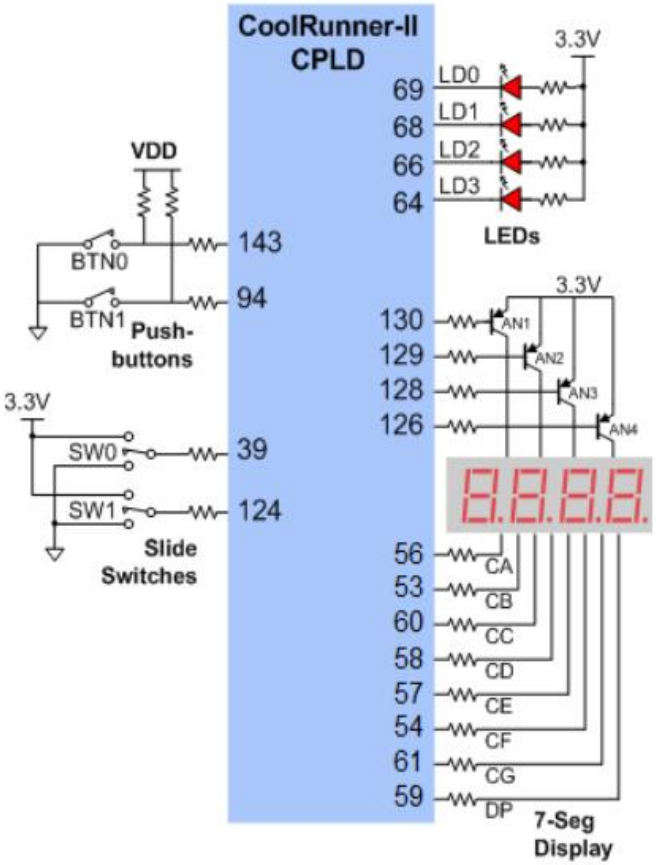


Figure 5. Schematic

To show properly the values of every BCD digit, we need to take in account that to represent more than one digit in a display, we cannot use directly the output pins and assign them directly the value at the same time, because the same digit would appear in every display. For this reason, we need to implement a mechanism that allows us to switch really quick from one display to another and so on to be able to show the correct value of the BCD digit.

```
-- Switch between displays and its corresponding values --
process (clk_10)
begin
    if rising_edge(clk_10) then
        if position = "0111" then
            numdis <= num_bcd(3 downto 0);
            position <= "1110";
        elsif position = "1110" then
            numdis <= num_bcd(7 downto 4);
            position <= "1101";
        elsif position = "1101" then
            numdis <= num_bcd(11 downto 8);
            position <= "1011";
        elsif position <= "1011" then
            numdis <= num_bcd(15 downto 12);
            position <= "0111";
        end if;
     end if;
end process;
```

The signal numdis is used for splitting the BCD value into groups of 4 bits to represent each digit on the corresponding displays.

For this purpose, we will use a mechanism based in the laboratory practice, using this time a 0.1ms period clock to switch the displays.

On CoolRunner-II CPLD starter board, it is possible to use the oscillator with a frequency of fclk = 10 kHz, 100 kHz or 1 MHz. The clock signal is connected to pin P38 of the target CPLD device and the frequency is setup by jumper JP1 as shown in Figure 2

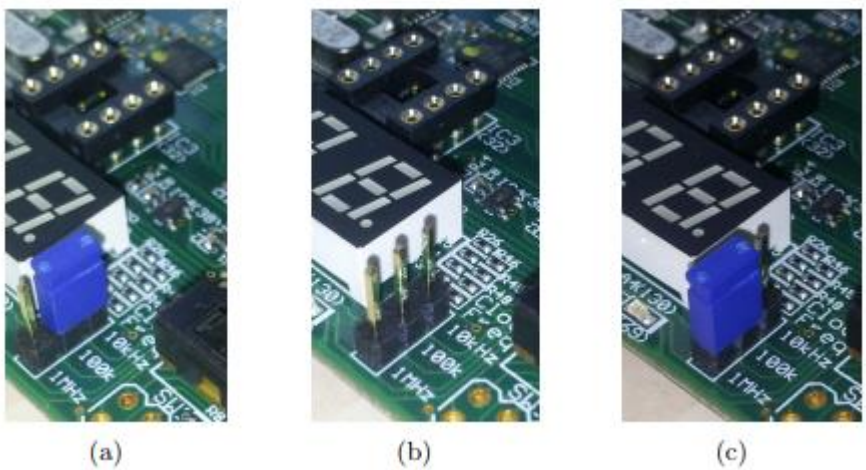


Figure 6. Clock signal frequency selection on CoolRunner-II CPLD starter board: (a) 10kHz, (b)100kHz, (c)1MHz

Therefore, the clock signal frequency *fclk* must be decreased by a software divider. Due to the divider, any synchronous change will occur every Nth period of clock signal clk. In the Figure 3 we can observe the relation between input and output clock divider frequencies.

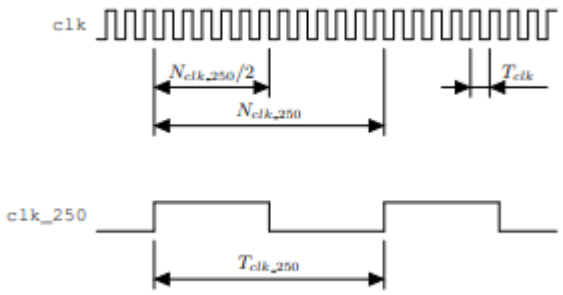


Figure 7. Clock divider

In our project we used 1MHz frequency clock. The results obtained are:

Tx	N0.1ms	N0.1ms/2	N0.1ms/2
0.1ms	100	50	32

fclk=1MHz→ Tclk=1us
N0.1ms=0.1ms/1us=100

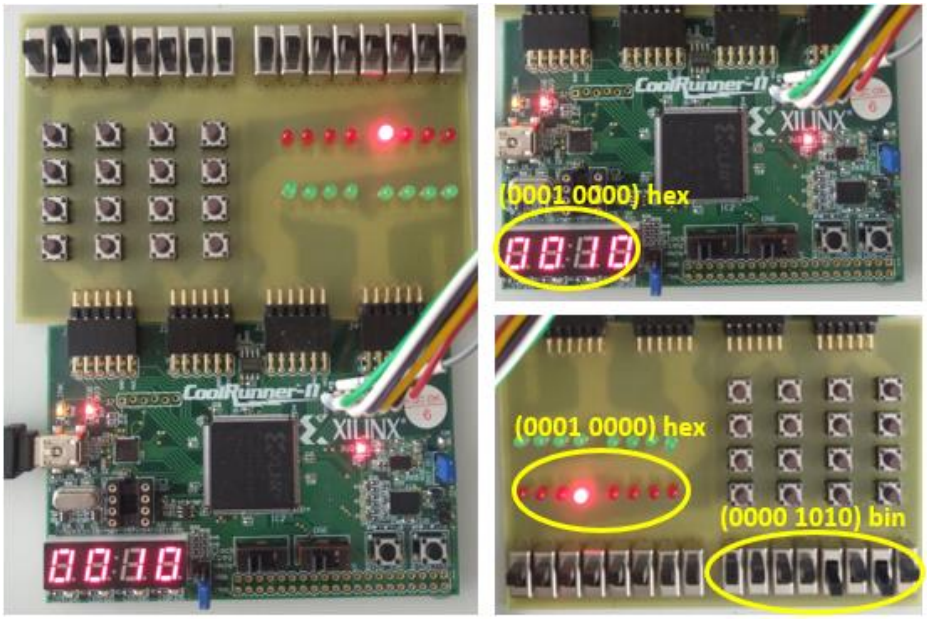
```
-------------------------------------
-- Show BCD value into the display --
-------------------------------------

-- Clock divider to 0.1 ms --
-- Increment auxiliary counter every rising edge of CLK
-- if you meet half a period of 0.1 ms invert clk_10
-- Switch counter --
process (CLK)
begin
    if rising_edge(CLK) then
        tmp_10 <= tmp_10 + 1;
        if tmp_10 = x"32" then
            tmp_10 <= x"00";
            clk_10 <= not clk_10;
        end if;
    end if;
end process;
```

7. EXAMPLES

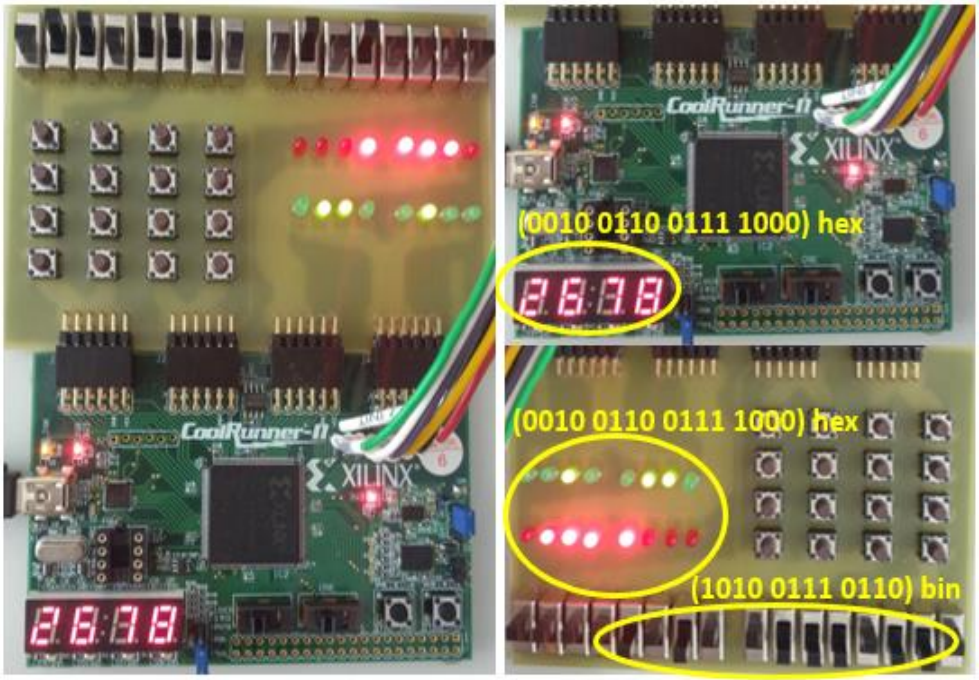
Now, let's check for example some of the possible combinations. We select some combinations that will show that our application works correctly.

- First, we are going to introduce, by the help of the switches, the number 10 (decimal) in binary (1010). This is an invalid number for BCD code, in other words, we will need two groups of 4 bits to represent it (1000)(0000). As we are expecting, this is our result:

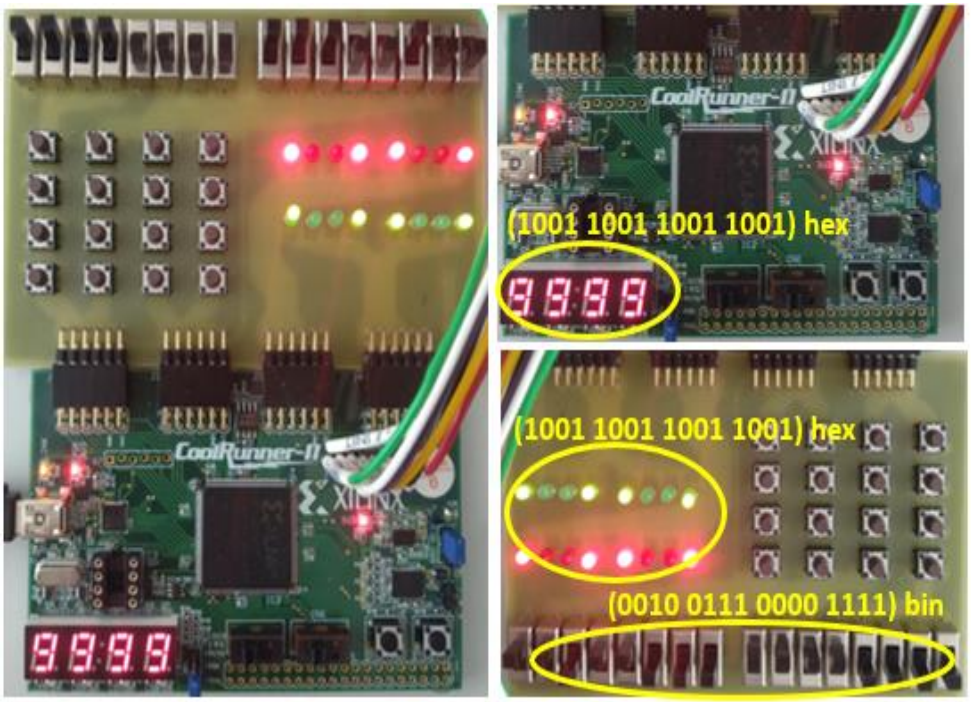


- Now, we are going to introduce another intermediate number such as 345 (decimal) and (0001 0101 1001) in binary. In BCD code, the result must be (0011)(0100)(0101) like in the photo.

- Another intermediate number, but this time of four digits, that we are going to proved will be 2678 (decimal) = (1010 0111 0110) (binary). In BCD number the result will be (0010)(0110)(0111)(1000).



- Finally, the highest number that we can represent in the displays is 9999 (decimal) = (0010 0111 0000 1111) in binary. In BCD code the result must be (1001)(1001)(1001)(1001) as we show in the photo below.



8. CONCLUSION

With the help of BCD code, the manipulation of the numeric data that must be shown, (in our case in a seven-segment display) becomes easier. This fact makes a simplification also in the physical design of the circuit (hardware). If the numeric quantity is saved and used in binary, the circuit will become more complex than using BCD.

The difficulties that we found in the present project are first understanding how the converter works. For one-digit BCD the combinations higher than 9 are no valid digits and we need more bits to represent it.

Then we had some problems while choosing the clock to display the value of the 7-segment display. We first thought about using a 10ms period clock with the 1kHz clock system but after testing it in the board we observe that it wasn't enough and we decided to change it to 0.1ms using the 1MHz clock system and worked as we desired.

Also, we had problems with the device when suddenly for no reason apparently, it stops to recognize it and we had to reboot the whole ISE program again.