



BCD adder Core Specification

Author: Gilian Breysens
gilianbreysens@opencores.org

Rev. [0.1]
May 10, 2022

This page has been intentionally left blank.

Revision History

Rev	Date	Author	Description
.			
0.1	29/08/16	Gilian Breysens	First Draft

Contents

INTRODUCTION.....	1
ARCHITECTURE.....	4
OPERATION	ERROR! BOOKMARK NOT DEFINED.
CLOCKS.....	ERROR! BOOKMARK NOT DEFINED.
IO PORTS.....	6
INDEX.....	8

1

Introduction

1.1 Binary coded decimal

Todo: testbench, diagram

This core implements a BCD adder. BCD (binary coded decimal) is a way to represent decimal numbers using binary representation. To convert a decimal number to BCD, simply replace each decimal digit by its 4 bit binary equivalent. Note that the binary values 1010 to 1111 are never used.

Decimal digit	BCD			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
Invalid	1	0	1	0
Invalid	1	0	1	1
Invalid	1	1	0	0
Invalid	1	1	0	1
Invalid	1	1	1	0
Invalid	1	1	1	1

Table 1: Table to convert a decimal digit to its BCD equivalent.

Example:

$198_{10} = 0001\ 1001\ 1000_{\text{BCD}}$

BCD's advantages are that it can easily be displayed on a 7 segment display and that it can represent decimal numbers exactly, while a binary representation may be infinitely long.

Example:

$$0.9_{10} = 0.0001100110011001100110..._2$$

BCD's disadvantages are that BCD circuits require more space and energy, and that they are slower than binary circuits.

Decimal arithmetic is preferred in financial and monetary applications because of the rounding errors binary arithmetic can introduce in fixed and floating point implementations.

Microprocessors usually handle decimal arithmetic using software routines. This can lead to a slow implementation when dealing with a workload that has to handle a lot of decimal arithmetic. Using dedicated circuitry can thus significantly improve the execution of programs where decimal arithmetic is often used.

1.2 Speculative decimal addition

This BCD adder core uses the speculative decimal addition algorithm. The algorithm to add two BCD numbers A and B and the carry in is as follows:

- 1) Add 6 to each decimal digit of operand B.
- 2) Use a binary adder to add A, B and the carry in.
- 3) Whenever the addition of two decimal digits (4 bit binary segments) doesn't overflow, subtract 6 from the decimal digit (4 bit binary segment) in the previously calculated sum. The sum is now the BCD sum of A, B and the carry in.

Example:

$$156 + 784 = ?$$

$$156_{10} = 0001\ 0101\ 0110_{\text{BCD}}$$

$$784_{10} = 0111\ 1000\ 0100_{\text{BCD}}$$

- 1) We add 6 to each 4 bit segment of operand B (784).
 $0111\ 1000\ 0100$ becomes $1101\ 1110\ 1010$
- 2) Add $0001\ 0101\ 0110$ and $1101\ 1110\ 1010$ with a carry in of 0.
 $0001\ 0101\ 0110_2 + 1101\ 1110\ 1010_2 = 1111\ 0100\ 0000_2$
- 3) Segment 3 (most significant 4 digit segment) didn't overflow, so we subtract 6 from that segment.
 $1111\ 0100\ 0000$ becomes $1001\ 0100\ 0000$
 $1001\ 0100\ 0000_{\text{BCD}}$
This is correct, since $156 + 784 = 940$

2

Architecture

Below is a top level diagram of the core.

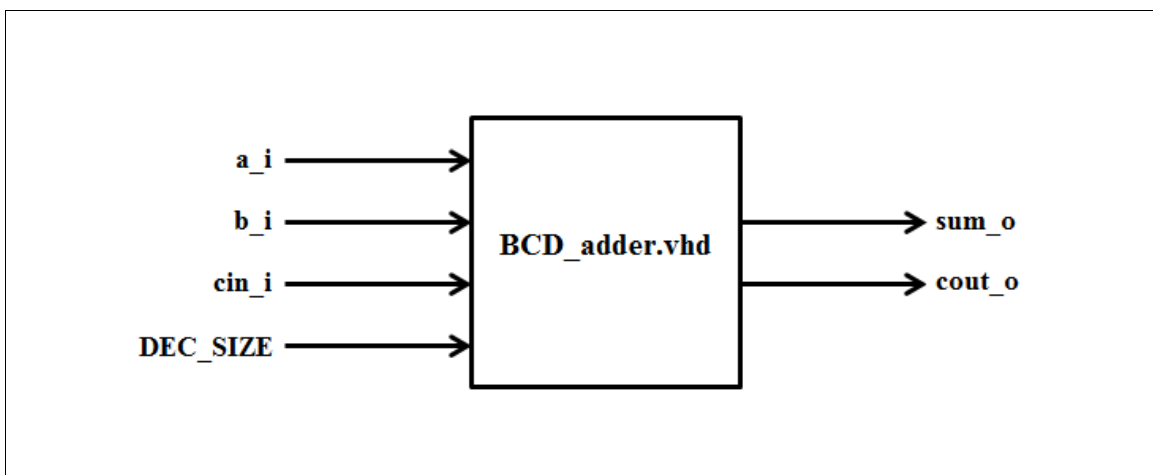


Figure 1: Top level diagram of the BCD adder core.

The core implements the speculative decimal addition algorithm as specified in the introduction.

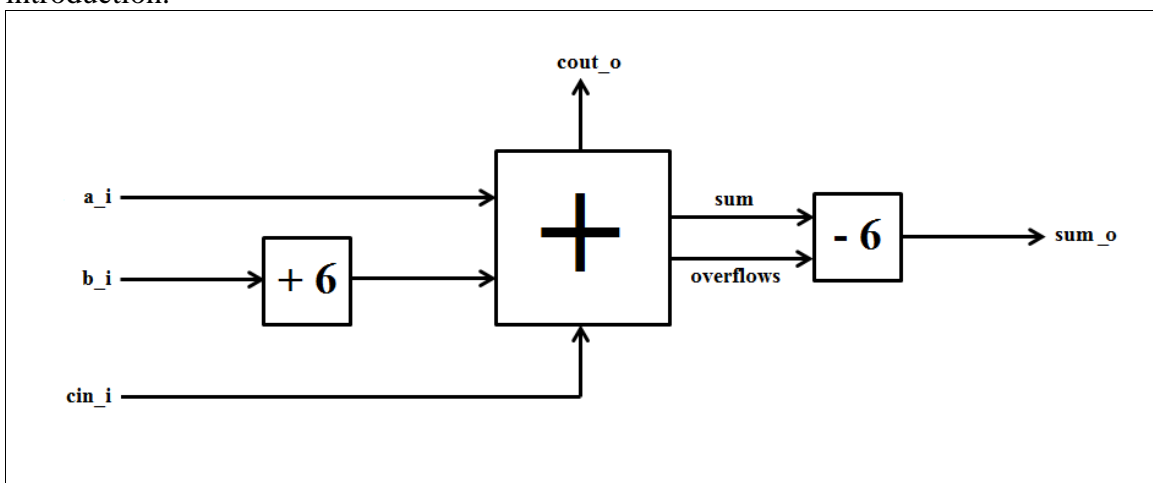


Figure 2: Diagram of the BCD adder implementing the speculative decimal addition algorithm.

The architecture contains three main structures:

- 1) Logic to add 6 to all 4 bit segments of operand b.
- 2) A binary adder to add the operands and the carry in. It also outputs a '1' each time a 4 bit segment overflows during the addition.

Example:

0111 1000 0010 + 0110 1000 1111 = 1110 0001 0001

The 1st and 2nd segment overflowed, so the second output is 011.

- 3) Sum corrector logic: subtracts 6 the nth 4 bit segment if the sum of nth 4 bit segments in step 2 didn't overflow.

3

IO Ports

Below is a table of the input and output ports of the BCD adder core.

Port	Width	Direction	Description
DEC_SIZE	DEC_SIZE	Generic input	Amount of decimal digits of operands
clk_i	1	Input	System clock
a_i	4*DEC_SIZE-1	Input	Operand a
b_i	4*DEC_SIZE-1	Input	Operand b
cin_i	1	Input	Carry in
sum_o	4*DEC_SIZE-1	Output	BCD sum
cout_o	1	Output	Carry out

Table 2: List of IO ports

4

Synthesis results

The following synthesis results were found using the Quartus II software for the Altera Cyclone III EP3C10F256C6. The results shown are with DEC_SIZE set to 3. Changing this parameter, as well as the used device, will change the speed and resource usage of the design.

Device	Altera Cyclone III EP3C10F256C6
Fmax	345.54 MHz
Total logic elements	44
Total combinational functions	31
Dedicated logic registers	38
Total registers	38
Total pins	39
Total memory bits	0
Embedded Multiplier 9-bit elements	0

Table 3: Synthesis results.

5

Simulation and testing

