

AN2112

Author: M. Ganesh Raaja

Associated Project: Yes

Associated Part Family: CY8C27x43, CY8C29x66, CY8C24x94, CY8C24x23A, CY8C21x34, CY8C21x23, CY8C20x34

[GET FREE SAMPLES HERE](#)

Software Version: PSoC Designer 5.0™ SP1

Associated Application Notes: None

Application Note Abstract

This application note describes a method to convert 8-bit, 16-bit, and 24-bit binary numbers to BCD (binary coded decimal). By following the algorithm presented in this application note any binary word length can be converted to BCD.

Introduction

Many applications interface with BCD numbers. These include applications that interface with real time clocks, display data on an LCD, and so on. The initial data for these applications is usually in binary format. This format can be uninformative to users; therefore, the need for BCD. However, conversion between binary and BCD can be cumbersome and unintuitive. This application note explains an algorithm to convert binary to BCD.

A processor that has a shift instruction best accomplishes this algorithm. Without a shift instruction, the conversion is code inefficient and speed inefficient. The PSoC® has several shift instructions, which allow easy implementation of the binary to BCD algorithm.

The algorithm discussed in this document uses only bit shifting and comparisons to convert binary to BCD. The algorithm can be applied to any word length.

The Data Structure

Before looking at the algorithm, it is necessary to understand the data structures used in this algorithm. The input is straightforward binary. The output is in packed BCD. The first nibble holds the ones, second nibble holds tens, third nibble the hundreds, and so on.

The first step is to decide the number of output bytes. For a 16-bit binary input, the maximum output is 65535. So the output has 3 bytes, 06 55 35. For a 24-bit binary input, the maximum output is 16777215. So the output has 4 bytes, 16 77 72 15.

The Algorithm

- Rotate the MSB (Most Significant Byte) of the input to the LSB (Least Significant Byte) of the output.
- If any BCD digit is 5 or more, then add 3 to that digit. A BCD digit corresponds to a nibble of the output byte. This step may occur more than once for 1 hex digit
- Repeat the above steps 'n' number of times where 'n' is the input word length.

In our program we have used a small variation. We first add 3 to the BCD digits. Then we check if the result is $\Rightarrow 8$. If yes, we leave the result as is. If no, we restore the previous value. This method is easier to implement.

Program for 8-Bit Conversion

The following program implements the above algorithm to perform an 8-bit binary to BCD conversion:

```
bin2bcd8:
    mov [bitCount],8
    mov [tBcd1],00h
    mov [tBcd0],00h

bBCD8_1:
    rlc [binary0]
    rlc [tBcd0]
    rlc [tBcd1]
    dec [bitCount]
    jnz bBCD8_2
    ret

bBCD8_2:
    mov [byteCount],2
    call checkDigits
    jmp bBCD8_1
```

```

checkDigits:
    mov A, tBcd0-1
    add A, [byteCount]
    mov X, A
    mov A, [X+0]
    mov [temp], A
    add [X+0], 03h
    tst [X+0], 8h
    jnz chkDigit2
    mov A, [temp]
    mov [X+0], A

```

```

chkDigit2:
    mov A, [X+0]
    mov [temp], A
    add [X+0], 30h
    tst [X+0], 80h
    jnz retCheckDigits
    mov A, [temp]
    mov [X+0], A

```

```

retCheckDigits:
    dec [byteCount]
    jnz checkDigits
    ret

```

Program Analysis

In line numbers 1, 2, and 3, the parameters are initialized. Result is made 0, and the bit-loop counter is set to 8, as an 8-bit conversion must be performed.

In line numbers 4, 5, and 6 the input and the output are rotated one bit to the left. This means, the MSb (Most Significant bit) of the binary is shifted to the LSb (Least Significant bit) of the output.

In line 7 and 8 the loop counter is decrement and then checked to see if it is zero. If not zero, the control branches to `bBCD8_2`. If zero, the program returns with the result in `tBCD1` and `tBCD0`.

bBCD_2

This section loads the byte loop counter with the number of output bytes and calls another subroutine, 'checkDigits'. It then branches back to `bBCD8_1`.

checkDigits

This subroutine checks each nibble of the output.

1. In lines 1, 2, and 3 the address of respective byte indicated by `byteCount` is calculated and moved to `X`.
2. A copy of the byte is made in `temp`.
3. Then '3' is added to the lower nibble.
4. The result is tested for `>=8`.

5. If yes, then the program branches to check the higher nibble.
6. If no, the byte is restored from `temp`.
7. This procedure is repeated with the higher nibble in `chkDigit2`, by adding 30h and checking for `=>80h`.
8. The `byteCount` is decremented to check if all bytes of the result are processed. If not, the whole procedure from steps 1 to 7 is repeated.

Performing 16-Bit Conversion

The following modifications to the program are necessary for a 16-bit conversion:

1. When initializing, `tBCD2`, `tBCD1`, and `tBCD0` are made 00.
2. The bit counter is initialized to 16.
3. RLC is carried out through `[tBCD2]` `[tBCD1]` `[tBCD0]` `[binary1]` `[binary0]`.
4. In `bBCD8_2`, `[byteCount]` is loaded with 3, the number of output bytes. Then the subroutine `checkDigits` is called.

Following this example, code can be written for 24-bit, 32-bit, 40-bit, and so on.

Comments

When the above algorithm is used for 8-bit conversion, the code size and execution time are not efficient. The 8-bit program is given to explain the conversion principle. For more efficient 8-bit conversion, use the following method:

1. Divide the binary input by 100.
2. Store the result in [hundreds].
3. Divide the remainder of step 1 by 10.
4. Store the result in [tens].
5. Store the remainder in [ones].

This method gives a split BCD output.

For 8-bit division, refer to the application note [AN2101](#), Unsigned Division Routines.

When `div8` of [AN2101](#) is used for the above method, the code size is 65 bytes (including the `div8` subroutine) and execution time is 1406 CPU cycles maximum.

Following this example, 16-bit binary can also be converted by successively dividing by 10000, 1000, 100, and 10. But the execution time may be higher than that of `bin2bcd16`.

Table 1. Input and Output Details for Subroutines

Name	Input	Output
bin2bcd8	[binary0]	[tBCD1] [tBCD0]
bin2bcd16	[binary1] [binary0]	[tBCD2] [tBCD1] [tBCD0]
bin2bcd24	[binary2] [binary1] [binary0]	[tBCD3] [tBCD2] [tBCD1] [tBCD0]

Table 2. Code Size and Execution Time Details

Name	Description	Code Size	Execution Time	
			Minimum	Maximum
bin2bcd8	8-Bit Binary to BCD Conversion	70 bytes	2045 CPU Cycles	2429 CPU Cycles
bin2bcd16	16-Bit Binary to BCD Conversion	77 bytes	5819 CPU Cycles	6971 CPU Cycles
bin2bcd24	24-Bit Binary to BCD Conversion	84 bytes	11353 CPU Cycles	13657 CPU Cycles

About the Author

Name: M. Ganesh Raaja

Title: R&D Engineer and owner of Omega Electronics & Consultants

Background: M. Ganesh does freelancing R&D work for various manufacturing industries. He obtained his diploma in Electronics and Communications Engg. in 1992. He has his own consultant firm called Omega Electronics & Consultants, which develops products for instrumentation and process control. He also develops and transfers technologies for manufacturing companies.

Contact: 856, 9th Cross, Hebbal 2nd Stage, Mysore - 570 017, India.

Document History

Document Title: Algorithm – Binary to BCD Conversion

Document Number: 001-36020

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1514403	BRW	09/27/07	New application note.
*A	2640726	GRAA	01/20/09	Updated content to make it current

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

PSoC is a registered trademark of Cypress Semiconductor Corp. "Programmable System-on-Chip," PSoC Designer, and PSoC Express are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
<http://www.cypress.com/>

© Cypress Semiconductor Corporation, 2003-2009. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.