Fast decimal-to-binary conversion in C

Giovanni Motta, Brandeis University, Waltham, MA

In a past issue of EDN magazine a C++/C function that provides integer-to-binary conversion was described (DI #2156, March 2, 1998). The function, named **cintbin/classicC** implements a digital to binary conversion in a straight, direct way that uses the definition of decimal and binary numbers in terms of powers. Although this function can be useful for testing or for didactic purposes, its low efficiency precludes its use in embedded or fast applications. The main source of inefficiency, is the use of the **pow** C function that requires linking the math libraries.

By carefully rewriting the function and by accessing directly the binary representation of the integers in C, it is possible to speed the conversion up to 58 times. This solution will also result in a more compact code because linking of the math library will be unnecessary.

Listing 1 shows a simple calling program and two functions: **fast_d2b** and **fast_b2d**, that implement conversion between decimal and binary representations of integers.

In **fast_d2b** the main loop is implemented as a **for** loop in which the variable **i** is used to access the array **c** sequentially, from **0** up to **31**. At each iteration, **x**, the number being converted, is repeatedly right shifted by **i** positions (>> operator) and its least significant bit is extracted by masking **x** with the constant **1** (**&** operator). The result is finally assigned to the **i**-th item of the vector **c**. **Fast_b2d** is a symmetric function that converts a binary number in its decimal representation. The function uses the same principle and performs the conversion via a sequence of left shifts and sums.

In order to compare the performance of **fast_d2b** with the function **classicC**, we ran several tests on two different platforms, both running Unix-like operating systems. We used the GNU C compiler (gcc) with and without turning on "aggressive optimization" (-O3 flag). The time necessary to convert the first 100.000.000 integers is compared in the **Table 1**. Results were obtained by reading the user running time with the **time** Unix command.

It is evident that the function **fast_d2b** is 20 to 58 times faster than **classicC** and that the compiler optimization is much more effective on **fast_d2b**.

The listing file can also be downloaded from EDN's Web site, www.ednmag.com. At the registered-user area, going into the Software Center and downloading the file from **DI-SIG, #XXXX** (**DI #XXXX**).

TABLE 1 - TIME TO CONVERT 100.000.000 INTEGERS

	Irix (SGI Indy R5000)		*Linux (PIII/500)*	
	gcc	**gcc -O3**	**gcc**	**gcc -O3**
fast_d2b	476 sec.	160 sec.	80 sec.	28 sec.
classicC	9377 sec.	8542 sec.	2206 sec.	1672 sec.

LISTING 1 - DECIMAL-TO-BINARY CONVERSION IN C

```
/*
   Fast decimal-to-binary conversions
   Author: Giovanni Motta (gim@ieee.org)
*/
#include <stdio.h>
int c[32];

/*
   Decimal to binary
*/
void fast_d2b(unsigned long x, int * c) {
   int i;

   for(i=0;i<32;i++)
      *(c++) = (x >> i) & 0x1;
}

/*
   Binary to decimal
*/
void fast_b2d(unsigned long int * n, int * c)
{
   int i = 32;

   *n = 0;
   while(i--) {
      *n <<= 1;
      *n += *(c+i);
   }
}

main() {
   int k;
   unsigned long int x,  y;

   printf("\nEnter an integer number");
   printf(" smaller than 4,294,967,296 : ");
   scanf("%lu", &x);
   printf("\nCalling fast_d2b for ");
   printf("decimal to binary conversion :\n");
   fast_d2b(x, c);
   printf("   Bin # =");
   for (k=31; k>=0; k--)
      printf(" %d",c[k]);
   printf("\n\nCalling fast_b2d for ");
   printf("binary to decimal conversion :\n");
   fast_b2d(&y, c);
   printf("   Dec # = %lu\n", y);
   return 0;
}
```