3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

Recepción: 21 de marzo de 2017
Aceptación: 30 de junio de 2017
Publicación: 29 de septiembre de 2017

ALGORITMO DE BOOTH EN OPERACIONES DE SUMA Y RESTA DE ENTEROS

BOOTH ALGORITHM HARDWARE OPERATIONS ADDITION AND SUBTRACTION

Jesús Ayuso Pérez[1]

1. Compositor musical y desarrollador. Licenciado en Ingeniería Informática por la Universidad Carlos III de Madrid (UC3M). E-Mail: ayusoperez@terra.com

Citación sugerida:

Ayuso Pérez, J. (2017). Algoritmo de Booth en operaciones de suma y resta de enteros. *3C TIC: Cuadernos de desarrollo aplicados a las TIC*, 6(3), 1-9. DOI: <http://dx.doi.org/10.17993/3ctic.2017.57.1-9/>.

3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

RESUMEN

El algoritmo dado por Andrew Donald Booth en 1950 (Booth, 1951) para la multiplicación es considerablemente potente como para limitar su uso únicamente a dicha operación, se puede aplicar a cualquier operación algebraica que se construya como una sucesión de cómputos de otra operación que la compone (Ayuso 2015, pp. 113-119). De ahí que en el presente documento, se proponga una implementación física de suma y resta de números enteros basado en dicho concepto. Se verá una alternativa al hardware tradicional basado en celdas FULL-ADDER y FULL-SUBTRACTOR, describiendo un nuevo tipo de celdas, que serán designados como FULL-SUCCESSOR y FULL-PREDECESSOR, fruto de llevarnos a nivel físico la técnica de Booth para implementar la suma o resta de 2 números enteros.

ABSTRACT

The algorithm given by Andrew Donald Booth in 1950 (Booth, 1951) for multiplication is considerably powerful to limit its use to only one operation of the word, any operation can be applied that is constructed as a succession of computations of another operation that the compone (Ayuso 2015, pp. 113-119). Hence, in this paper, a physical implementation of addition and subtraction of integers based on this concept is proposed. You will see an alternative to traditional hardware based on FULL-ADDER and FULL-SUBTRACTOR cells, describing a new type of cells, which have been designated as FULL-SUCCESSOR and FULL-PREDECESSOR, the result of carrying out a physical level of Booth's technique for implement the addition or subtraction of 2 integers.

PALABRAS CLAVE

Booth, Algoritmo, Adición, Sustracción, Hardware.

KEYWORDS

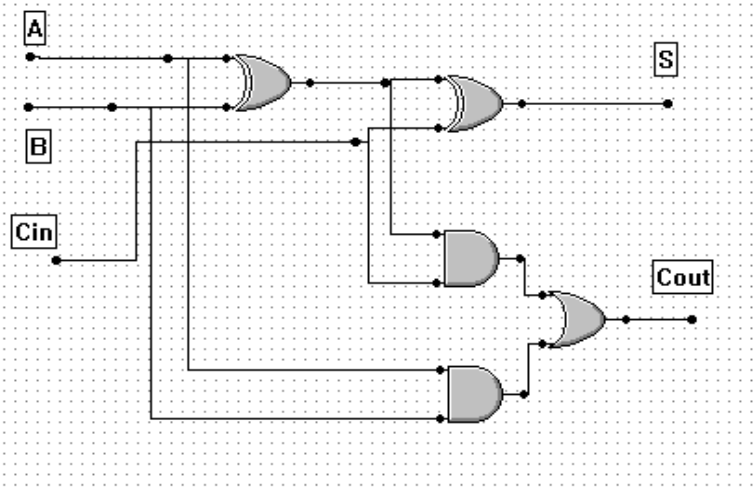
Booth, Algorithm, Addition, Subtraction, Hardware.

3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

1. INTRODUCCIÓN

El concepto de Booth se presenta perfectamente aplicable a la operación de suma y resta entre enteros dando una alternativa a la implementación clásica para realizar dichos cálculos (Burks, A., Goldstein, H. and Von Neumann, J. 1946). La solución física con la que tradicionalmente se suele abordar, por ejemplo, la operación de suma se basa en la celda que se conoce como FULL-ADDER:

Ilustración 1. Celda clásica FULL-ADDER.



Fuente: elaboración propia.

La anterior celda (Ilustración 1) es capaz de realizar la suma de 2 operandos que constarán de 1 bit cada uno. De manera que para sumar operandos de *n* bits lo único que se hace es encadenar *n* celdas FULL-ADDER (Burks, A., Goldstein, H. and Von Neumann, J. 1946) donde cada celda proporciona su posible acarreo como acarreo de entrada a la celda siguiente:

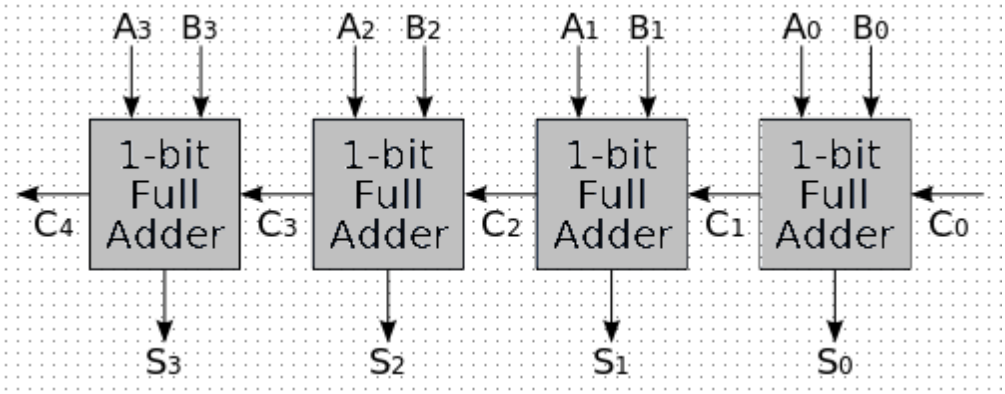


Ilustración 2. Sumador clásico con celdas FULL-ADDER.
Fuente: elaboración propia.

Partiendo de esto, la implementación física que se explicará en el presente documento es una alternativa al uso de las celdas mostradas en la Ilustración 1 y la Ilustración 2 derivada de interpretar nuestra operación de suma o nuestra operación de resta como una sucesión de incrementos o decrementos, respectivamente; en lugar de como una operación binaria de suma entre números representados en binario.

3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

Va a exponerse una celda alternativa, basada en los conceptos ideados por Booth (Booth, pp.236-240), que simplificarán el hardware de cada celda, con la consiguiente mejora en la latencia, además de parámetros como ahorro energético, problemas de disipación de calor y la velocidad. Y de igual modo que en el esquema clásico se requerirá encadenar las celdas para gestionar el sucesor y el antecesor, como se muestra en la Ilustración 2.

2. METODOLOGÍA Y RESULTADOS

Para este apartado, lo primero que haremos será recordar la solución para la operación de adición y sustracción expuestas en la referencia bibliográfica titulada: '*Booth algorithm operations addition and subtraction*' (Ayuso 2015, pp. 113-119). Pero ésta será matizada un poco, ya que así nos va a resultar más simple entender la nueva implementación presentada en este documento, la cual está particularizada al caso de suma y resta de números enteros.

Básicamente, la única diferencia con respecto a la que figura en la bibliografía (Ayuso 2015, pp. 113-119) va a ser que en lugar de entender que operamos un elemento contra otro, con respecto a determinada operación algebraica, vamos a enfocarlo que operamos 2 elementos contra el elemento neutro de dicha operación algebraica. Con ello, y dando por sentado que disponemos de una operación de sucesor y antecesor, *successor* y *predecessor*, que nos permiten realizar un incremento y un decremento, respectivamente, en la posición *i-esima* de un operando cualquiera de *n* bits, nuestra versión del algoritmo de suma de dos números, *a* y *b*, de longitud *n* quedaría:

Fórmula 1. Algoritmo de suma 1.

```
result = 0; // ELEMENTO NEUTRO
for(int i = 0; i < n; i++)  {
      if(a[i] == 1) result = successor(result, i);
      if(b[i] == 1) result = successor(result, i);
}
```

Fuente: elaboración propia.

Y la versión del algoritmo de resta de dos números, *a* y *b*, de longitud *n* (presuponiendo por simplicidad que *a* siempre es mayor o igual que *b*) quedaría:

Fórmula 2. Algoritmo de resta 2.

```
result = 0; // ELEMENTO NEUTRO
for(int i = 0; i < n; i++)  {
      if(a[i] == 1) result = successor(result, i);
      if(b[i] == 1) result = predecessor(result, i);
}
```

Fuente: elaboración propia.

3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

De acuerdo, llegados a este punto merece la pena repasar la tabla dada por Booth para el cálculo de operaciones según sus conceptos, apoyándose en la capacidad cancelativa de la operación con la que se construye:

bit menos significativo	bit extra	Interpretación	Acción
0	**0**	intermedio cadena de 0s	ninguna
0	**1**	final cadena de 1s	operación
1	**0**	comienzo cadena de 1s	operación inversa / inverso misma operación
1	**1**	intermedio cadena de 1s	ninguna

Tabla 1. Acciones de Booth.
Fuente: elaboración propia.

El siguiente paso será librarnos de la pesada carga de tener que disponer de las operaciones de sucesor y antecesor, *successor* y *predecessor*, para el caso que nos ocupa, como decimos: cuando la adición se particulariza a la suma de 2 números enteros. Expongámoslo de forma algorítmica partiendo del Algoritmo de suma 1, quedando la suma de dos números, *a* y *b*, de longitud n como:

Fórmula 3. Algoritmo de suma 2.

```
result = 0;bitExtra = 0;
for(int i = 0; i < n; i++)  {
     if(bitExtra == 1) {
          if(a[i] == 1 && b[i] == 1)
       result[i] = 1;
     else if(a[i] == 0 && b[i] == 0) {
       result[i] = 1;
       bitExtra = 0;
     }
     } else { // BIT EXTRA IGUAL 0
     if(a[i] == 1 && b[i] == 1)
       bitExtra = 1;
     else if(a[i] == 1 || b[i] == 1)
       result[i] = 1;
     }
}
```

Fuente: elaboración propia.

Y el caso de la resta de dos números, *a* y *b*, de longitud *n* a partir del Algoritmo de resta 1 quedaría:

3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

Fórmula 4. Algoritmo de resta 2.

```
result = 0;
bitExtra = 0;
for(int i = 0; i < n; i++)  {
      if(bitExtra == 1) {
            if((a[i] == 1 && b[i] == 1)
            ||  (a[i] == 0 || b[i] == 0))
        result[i] = 1;
      else if(a[i] == 1 && b[i] == 0)
        bitExtra = 0;
      } else { // BIT EXTRA IGUAL 0
      if(a[i] == 1 && b[i] == 0)
        result[i] = 1;
      else if(a[i] == 0 && b[i] == 1) {
        result[i] = 1;
        bitExtra = 1;
      }
      }
}
```

Fuente: elaboración propia.

Si nos fijamos, las variaciones anteriores de lo único que se sirven para eliminar la necesidad de operación de sucesor y/o antecesor es el entender que, como operamos contra el elemento neutro de la operación algebraica, que para el caso que nos ocupa es el 0, sabe que dicha operación nunca puede conllevar más de un acarreo. Aunque cabe insistir que en nuestro caso, no se trata tanto de un acarreo en su sentido clásico, es decir, como un desbordamiento producto de una operación entre dígitos, sino que es más un peso simbólico fruto de proyectar un elemento contra su sucesor o su antecesor.

Dicho lo anterior, y ya con la lógica absorbida en el código anterior, sí resulta mucho más fácil poder inferir lo que será nuestro nuevo hardware. Para el caso de la suma descrita en el Algoritmo de suma 2 nuestra celda FULL-SUCCESSOR:

3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

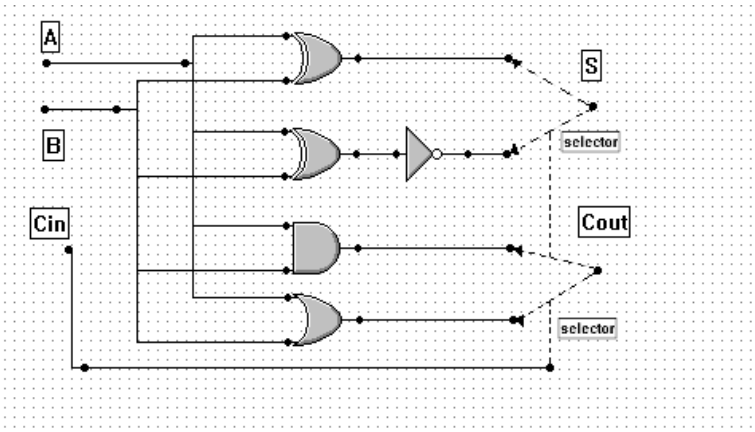


Ilustración 3. Celda nueva FULL-SUCCESSOR.
Fuente: elaboración propia.

La Ilustración 3 ya en un primer análisis visual parece que se trata de una celda más homogénea que la de FULL-ADDER (Ilustración 1): con menos niveles de puertas lógicas, es decir, que no hay puertas lógicas que alimenten la entrada de otras puertas lógicas, como ocurre en la celda del sumador clásico (Burks, A., Goldstein, H. and Von Neumann, J. 1946). Destacamos esto porque es crucial de cara a la lantencia. Únicamente existe una puerta XOR que alimenta una puerta NOT, por lo demás están todas al mismo nivel, pudiendo ejecutarse, por así decirlo, totalmente en paralelo, sin retardos ni latencias por esperar a otras puertas lógicas.

Aunque bien es cierto que hay que matizar, y explicar detenidamente, el último nivel de la celda anterior. En él se describe un comportamiento a modo de selector, donde dependiendo del valor de *Cin*, lo que podría identificarse, siguiendo la analogía con la celda FULL-ADDER clásica, como el acarreo entrante; decíamos, que dependiendo de este valor, se selecciona como salida para el valor de *S* y de *Cout*, lo que llega de la puerta superior o de la inferior, del XOR sin negar o de la puerta AND, respectivamente. El motivo de describirlo de esa manera, es que, como sabemos, esa función de selección entre 2 bits, típicamente un multiplexor 2 a 1, puede implementarse de muchas maneras, y en nuestro caso es algo que vamos a dejar al margen del concepto explicado en el presente documento. Se actúa de esa manera, porque incluso dicha función de selección, podría sacarse de la celda FULL-SUCCESSOR (Ilustración 3), y tratarse desde fuera de la celda con multiplexores de un mayor número de entrada, o con otro tipo de implementaciones hardware.

Vayamos ahora al caso de la celda FULL-PREDECESSOR extraída del Algoritmo de resta 2. La describiríamos de la siguiente manera:

3C TIC *(Edición 22)* Vol.6 – Nº 3
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

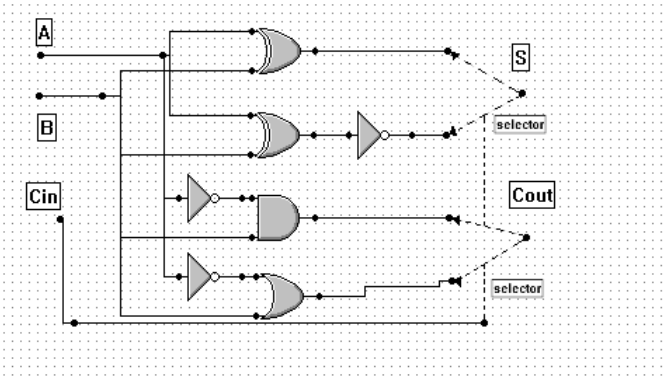


Ilustración 4. Celda nueva FULL-PREDECESSOR.
Fuente: elaboración propia.

Igualmente que en el caso anterior, en la Ilustración 4 requerimos de la capacidad de poder seleccionar entre 2 de las salidas que generan nuestra lógica de puertas digitales, tanto para el valor del *i-esimo* bit, como para el del *i-esimo* acarreo. También vemos que tenemos únicamente una profundidad de puertas lógicas (si obviamos la añadida por las operaciones NOT), luego volvemos a resaltar la mejora en la latencia y en los tiempos de respuesta de esta celda con respecto a la celda FULL-SUBTRACTOR clásica.

Llama aún más la atención el enorme parecido entre ambas celdas: entre la Ilustración 3 y la Ilustración 4. Analizadas detenidamente, puede fijarse que son exactamente iguales, salvo en el cálculo del acarreo saliente, donde en lugar de entrar el *i-esimo* bit de A, se alimenta del *i-esimo* bit de A negado.

3. CONCLUSIONES

Aplicar el algoritmo de Booth en la implementación física de las operaciones de suma y resta entre números enteros nos ofrece una inesperada alternativa a la solución hardware actual. Dicha alternativa tiene un esquema con menos niveles de puertas lógicas, lo que la hace mucho menos susceptible de problemas de latencia, consumo y generación de calor. Además, si se obvia la negación, el NOT, tanto en el caso de las celdas clásicas, FULL-ADDER y FULL-SUBTRACTOR, como en las celdas presentadas en este documento, tenemos que en la implementación clásica cada celda requiere de unas 5 puertas lógicas para realizar el cómputo. En cambio, las nuevas celdas presentadas, requieren de 4 puertas lógicas. Es decir, que además se trata de una solución más eficiente y económica.

En conclusión, la idea propuesta por Booth (Booth, 1951), simplifica la implementación de un sumador o un restador hardware, además de ofrecer un diseño más compacto y eficiente. Con las consiguientes mejoras en aspectos tan cruciales como son los retardos entre puertas lógicas y la cantidad de puertas necesarias.

Como posible futura línea de investigación, el enorme parecido entre las nuevas celdas hardware tratadas durante este documento podría dar pie a una mejora en la descripción física de la celda FULL-PREDECESSOR (**Ilustración 4**) de manera que finalmente ésta fuese implementada simplemente negando la entrada **A** a una celda FULL-SUCCESSOR (**Ilustración 3**).

3C TIC ***(Edición 22)*** *Vol.6 – Nº 3*
Septiembre – diciembre '17, 1 - 9
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.57.1-9

4. REFERENCIAS BIBLIOGRÁFICAS

Booth, A. D. (1945). A method of calculating reciprocal spacings for X-ray reflections from a monoclinic crystal, *J. Sci. Instr*, 22, 74.

Burks, A., Goldstein, H. and Von Neumann, J. (1946). Logical Design of an Electronic Computing Instrument.

Booth, A. D. and Britten, K. H. V. (1947). General Considerations in the Design of an Electronic Computer.

Booth, A. D. (1951). A signed binary multiplication technique, *Q.J. Mech. and Appl. Mat,* 4, (2), 236-240.

Ayuso, J. (2015). "Booth algorithm operations addition and subtraction", *3C TIC*. 4, (2), 113-119.

Ayuso, J. (2015). Booth algorithm modular arithmetic operations of addition and subtraction, *3C TIC*, 4, (3), 222-229.