/02/

ALGORITMO DE BOOTH-KARATSUBA EN OPERACIONES ADITIVAS

BOOTH-KARATSUBA ALGORITHM ADDITIVE OPERATIONS

Jesús Ayuso Pérez
Compositor musical. Licenciado en Ingeniería Informática por la Universidad Carlos III de Madrid (UC3M).
E-mail: ayusoperez@terra.com

Recepción: 24/09/2018. Aceptación: 29/11/2018. Publicación: 28/12/2018

Citación sugerida:
Ayuso Pérez, J. (2018). Algoritmo de booth-karatsuba en operaciones aditivas. *3C TIC. Cuadernos de desarrollo aplicados a las TIC*, 7(4), pp.30-59. doi:http://dx.doi.org/10.17993/3ctic.2018.62.30-59

RESUMEN

El algoritmo dado por Andrew Donald Booth en 1950 (Booth, 1951) para la multiplicación y el algoritmo dado por Anatoly Alexeevitch Karatsuba en 1960 (Karatsuba, 1962) a su vez también para la multiplicación demuestran tener mucho más en común de lo que sus autores describieron al definirlos (Ayuso, 2006-2018). Tal relación se estrecha al ser generalizados llegando al punto incluso de acabar encontrándose. De ahí que en el presente documento se proponga una solución hardware para adiciones y sustracciones entre enteros basada en los citados algoritmos, evidenciando la convergencia insoslayable entre los citados conceptos.

ABSTRACT

The algorithm given by Andrew Donald Booth in 1950 (Booth, 1951) for multiplication and the algorithm given by Anatoly Alexeevitch Karatsuba in 1960 (Karatsuba, 1962) in turn for multiplication prove to have much more in common than what its authors described when defining them (Ayuso, 2006-2018). Such a relationship is narrowed to be widespread reaching the point of ending up meeting. Hence, in this document a hardware solution is proposed for additions and subtractions between integers based on the aforementioned algorithms, demonstrating the evident convergence between the aforementioned concepts.

PALABRAS CLAVE

Booth, Karatsuba, Algoritmo, Adición, Operaciones.

KEY WORDS

Booth, Karatsuba, Algorithm, Addition, Operations.

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

1. INTRODUCCIÓN

Puede documentarse que el algoritmo de Booth (Booth, 1951) es extrapolable a otros contexto algebraicos, ya sea por ejemplo un contexto aditivo (Ayuso 2015, pp. 113-119), tanto modular o no (Ayuso 2015, pp. 222-229), llevarse en ese mismo contexto a mundo físico (Ayuso 2017, pp. 1-9), al margen del consabido ámbito donde fue definido en primera instancia (Ayuso 2015, pp. 255-221), a la vez que también puede perfectamente adaptarse a un grupo multiplicativo (Ayuso 2017, pp. 1-12); es más, incluso trabajar en una dimensión diferente a la utilizada en el algoritmo original (Ayuso 2018, pp. 10-35). Igualmente se puede consultar la adaptabilidad del concepto expuesto por Karatsuba (Karatsuba, 1962) nuevamente a un entorno aditivo (Ayuso 2018, pp. 10-21), como a la operación definida sobre ese referenciado grupo multiplicativo (Ayuso 2018, pp. 13-20). Por otra parte, esa versatilidad de los conceptos posibilita su uso en operaciones muy dispares a la de multiplicación (Ayuso 2016, pp. 28-41), que, como ya se ha mencionado, fue la empleada nativamente por los autores; amén de explotar su rendimiento utilizando un concepto representacional más óptimo (Ayuso 2016, pp. 33-43), o hasta el extremo de simultanear su aplicabilidad dentro de la misma operación (Ayuso 2017, pp. 19-26) a distintos operandos.

Lo cierto es que el gran potencial de estos algoritmos guarda cierto vínculo, pues tanto en el caso de la idea de Booth como la de Karatsuba acababan ahorrando coste computacional (Turing 1937, pp. 230-265), o disminuyendo orden de complejidad, apoyándose en las operaciones que a su vez componen a la operación que buscaban calcular de una forma más óptima. Es decir, eran capaz de proyectarse contra el mismo elemento algebraico de la transformación que se estaba realizando, pero usando atajos dentro de la estructura algebraica facilitados por las propias relaciones de las operaciones que definen a la operación compuesta del contexto en que se estuviera trabajando.

Por existir ese vínculo conceptual entre ambos y por el hecho de que a toda estructura algebraica a la que se puede adaptar uno también se puede adaptar el otro, se llega a derivar que ambos algoritmos se tocan en un punto: acaban hiriéndose el uno al otro, confundiéndose o complementándose, haciendo prácticamente indistinguible si se trata de lo primero o de lo segundo. En el presente artículo se va a tratar de exponer ese punto de encuentro entre los dos algoritmos, ofreciendo una solución hardware para el cómputo de adiciones y sustracciones entre números enteros. En el fondo, básicamente se tratará una hibridación de lo expuesto en Ayuso 2017, pp. 1-9 y lo que se postula en Ayuso 2018, pp. 10-21. Pero como adelantamos, con el actual enfoque, mirados como un todo donde esa línea que los separa se vuelve fina y difusa.

En definitiva, lo presentado durante el próximo desarrollo va a ser la consecuencia inevitable de afrontar la adición de enteros a través del proceso señalado por Booth y Karatsuba para ahorrar cálculos y obtener la solución final de una manera más eficiente y directa. Rebajando los órdenes de complejidad clásicos. Con el añadido de llevarnos esos conceptos al mundo físico, con lo que se ofrece una alternativa eficiente a las soluciones implementativas actuales. Es por ello que, por motivos evidentes, derivados de esa solución hardware supeditada a lógica digital (Shannon 1938, pp. 713–723), se diferirá con respecto al trabajo visto en: Ayuso 2018, pp. 10-21 en que en este caso utilizaremos siempre base binaria.

2. MÉTODOS

Partiendo de esa premisa, y aplicando directamente los conceptos vistos en: Ayuso 2015, pp. 113-119 y el algoritmo descrito en: Ayuso 2018, pp. 10-21, podemos deducir de forma inmediata el siguiente código para la suma de dos números enteros, a y b, de longitud n:

```
addBoothKaratsuba(a, b, result, S, index, n)
{
if(n == 1)        // CASO BASE
{
                  result[index] = XOR(a[0], b[0]);
                  S[index + 1] = AND(a[0], b[0]);
        return;
        }

        w = a / 2^(n / 2);
        x = a % 2^(n / 2);

        y = b / 2^(n / 2);
        z = b % 2^(n / 2);

d = n % 2 == 0 ? n / 2 : (n / 2) + 1;
```

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

```
addBoothKaratsuba(w, y, result, S, index + d, n / 2);
        addBoothKaratsuba(x, z, result, S, index, d);
}

addBoothKaratsuba(a, b, result)
{
        S = 0;
        n = length(a) > length(b) ? length(a) : length(b);

        addBoothKaratsuba(a, b, result, S, 0, n);

bitExtra = 0;
for(i = 0; i < n; i++) {
        switch(actionBooth(S[i], bitExtra) {
                case ( 0 1 ):
                        result = successor(result, i);
                        break;
                case ( 1 0 ):
                        result = predecessor(result, i);
                        break;
                }
                bitExtra = S[i];
        }
}
```

Algoritmo 1. Algoritmo recursivo de adición por Booth-Karatsuba (addBoothKaratsuba).

Se entiende que la operación *length* del código que figura sobre estas líneas, Algoritmo 1, devuelve el número de dígitos, en base 2 en nuestro actual caso, de los que consta el operando. Y se da por sentado que la rutina *accionBooth* actúa según el criterio marcado por Booth (Ayuso 2015, pp. 113-119).

El primer paso importante en este desarrollo, para comprender lo que realmente busca exponer el presente artículo, será llevarnos el concepto de Booth al terreno de Karatsuba, es decir, ensamblar las 2 ideas bajo el paradigma recursivo que empleó Karatsuba.

Para conseguir dar ese salto, vamos a redefinir la tabla de acciones dada por Booth (Booth 1951, pp. 74) pero separando las acciones a realizar frente a las operaciones a propagar (Ayuso 2018, pp. 10-35), para poder así encajarlo en un esquema recursivo. Para ello, se optará por una nomenclatura ternaria, como se hace en Ayuso 2016, pp. 33-43. Luego tendríamos:

Tabla 1. Tabla de acciones de Booth a propagar.

operations	0	1	-1	1 Λ -1
0	0	0	-1	-1
1	1	1	1 Λ -1	1 Λ -1
-1	0	0	-1	-1
1 Λ -1	1	1	1 Λ -1	1 Λ -1

En Tabla 1 se representa el criterio de propagación de las distintas acciones del algoritmo de Booth al cruzarse entre ellas (Ayuso 2016, pp. 28-41) debido a ese efecto de particionado binario realizado por la técnica de Karatsuba. Por otra parte tendríamos:

Tabla 2. Tabla de acciones de Booth a realizar.

operations	0	1	-1	1 Λ -1
0	0	1	0	1
1	0	1	0	1
-1	-1	0	-1	0
1 Λ -1	-1	0	-1	0

En Tabla 2 se establecen las operaciones a realizar cuando las diferentes acciones de Booth se encuentran (Ayuso 2017, pp. 19-26) durante la subdivisión descrita por Karatsuba (Karatsuba 1962, pp. 293–294).

Partiendo de los criterios de actuación precisados en las dos tablas anteriores, y nuevamente basándonos en los conceptos vistos tanto: Ayuso 2015, pp. 113-119 como lo descrito en: Ayuso 2018, pp. 10-21, podemos redefinir la suma de dos números enteros, a y b, de longitud n:

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

```
OpBooth addBoothKaratsuba (a,b,result, index,n)
{
if(n == 1)        // CASO BASE
{
                result[index] = XOR(a[0], b[0]);
        return AND(a[0], b[0]) ? ( 1 ^ -1 ) : ( 0 );
        }

        w = a / 2^(n / 2);
        x = a % 2^(n / 2);

        y = b / 2^(n / 2);
        z = b % 2^(n / 2);
d = n % 2 == 0 ? n / 2 : (n / 2) + 1;
OpBooth op1 =
        addBoothKaratsuba(w, y, result, index + d, n / 2);
        OpBooth op2 = addBoothKaratsuba(x, z, result, index, d);
        switch(op1) {
                case ( 0 ):
                        switch(op2) {
                                case ( 0 ):
                                        return ( 0 );
                                case ( 1 ):
                                        successor(result, index + d + 1);
                                        return ( 0 );
                                case ( -1 ):
                                        return ( -1 );
                                case ( 1 ^^ -1 ):
                                        successor(result, index + d + 1);
                                return ( -1 );
                                }
```

```
break;
                case ( 1 ):
                        switch(op2) {
                                case ( 0 ):
                                        return ( 1 );
                                case ( 1 ):
                                        successor(result, index + d + 1);
                                        return ( 1 );
                                case ( -1 ):
                                        return ( 1 ^^ -1 );
                                case ( 1 ^^ -1 ):
                                        successor(result, index + d + 1);
                                        return ( 1 ^^ -1 );
                        }
                        break;
                case ( -1 ):
                        switch(op2) {
                                case ( 0 ):
                                        predecessor(result, index + d + 1);
                                        return ( 0 );
                                case ( 1 ):
                                        return ( 0 );
                                case ( -1 ):
                                        predecessor(result, index + d + 1);
                                        return ( -1 );
                                case ( 1 ^^ -1 ):
                                        return ( -1 );
                        }
                        break;
                case ( 1 ^^ -1 ):
                        switch(op2) {
```

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

La principal diferencia entre **Algoritmo 1** y la variación que acabamos de dar en **Algoritmo 2** es que, en el primero, después de realizar los cálculos de la adición a través del proceso de partición binaria definido por Karatsuba, había un paso final de ajuste de los sucesores y antecesores a través del cálculo simplificado definido por Booth (Ayuso 2017, pp. 1-12). En cambio en esta segunda variación, dicho cómputo final desaparece.

Una vez alcanzada la descripción recursiva de la fusión de ambos conceptos (Ayuso 2018, pp. 10-21), se va a proceder justo a la inversa, es decir, ahora vamos a llevarnos el algoritmo de Karatsuba al terreno de Booth, buscando ensamblar las 2 ideas bajo un paradigma iterativo, de cara a materializar una implementación física (Ayuso 2017, pp. 1-9).

Volvemos sobre nuestros pasos. Del contenido de **Tabla 1** podemos extraer los siguientes mapas de Karnaugh (Karnaugh 1953, pp. 53-217) a modo de minitérminos:

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

CD \ AB	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

Ilustración 1. Mapa de Karnaugh de las funciones algebraicas Booleanas de Tabla 1.
Fuente: elaboración propia.

Y procediendo exactamente de la misma manera, para la lógica absorbida en **Tabla 2** que contiene esa otra dualidad de la descomposición algebraica derivada de la relación binaria de equivalencia definida en la estructura tendríamos los siguientes términos en forma de diagrama:

CD \ AB	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	0	0	0
10	0	0	0	0

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	1	0
10	1	0	1	0

Ilustración 2. Mapa de Karnaugh de las funciones algebraicas Booleanas de Tabla 2.
Fuente: elaboración propia.

Previamente al diseño de la lógica de actuación, establecemos el módulo para los cómputos de los antecesores y sucesores inherentes a los conceptos definidos por Booth y por Karatsuba, los cuales con el enfoque tradicional pueden entenderse como el cálculo del acarreo. En principio, el circuito es tal simple como:

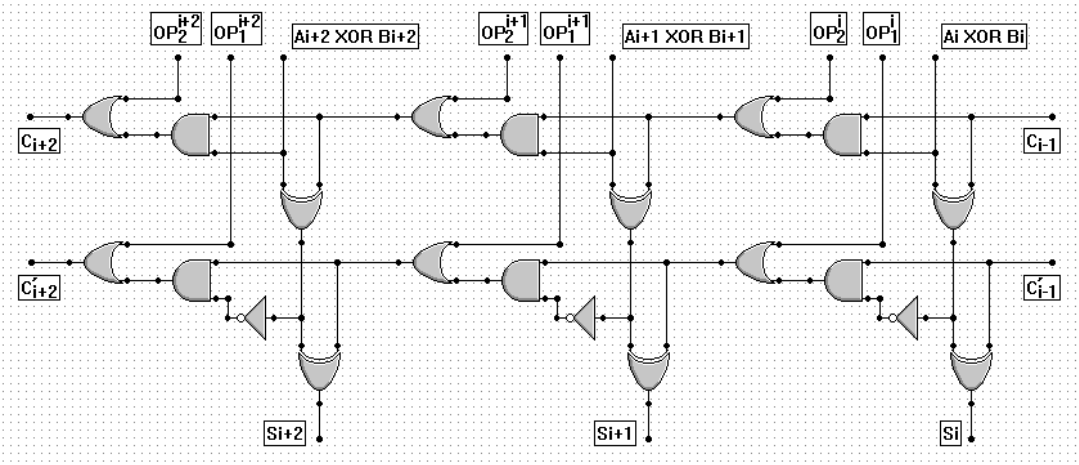

Ilustración 3. Módulo de sucesores/antecesores de Booth-Karatsuba.
Fuente: elaboración propia.

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

En Ilustracion 3 se muestra un fragmento central de módulo de cálculos de sucesores y antecesores hardware, coordinados por las operaciones designadas según los casos identificados en Tabla 2 con las acciones de Booth a realizar. Como puede apreciarse, quedan modelados con 2 bits, cada operación, pudiendo generar un dígito, denotado en la imagen con C, de peso representacional para los sucesores, y un peso digital, denotado con C', para los antecesores; a través los cuales acaba por determinarse la salida para cada *i-ùsimo* bit. Ya se ha concretado que, tanto ese C como C' con un enfoque clásico pueden interpretarse como el típico acarreo, pero dentro del contexto que nos ocupa adquieren un carácter más abstracto de proyección ordinal algebraica, permitiendo ir mapeándonos o ir saltando entre distintos elementos de la estructura.

Basado en lo dicho con anterioridad, y con el apoyo de lo inferido de Ilustración 1 e Ilustración 2, se tiene como módulo para la lógica señalada por Booth y Karatsuba, el siguiente circuito físico:

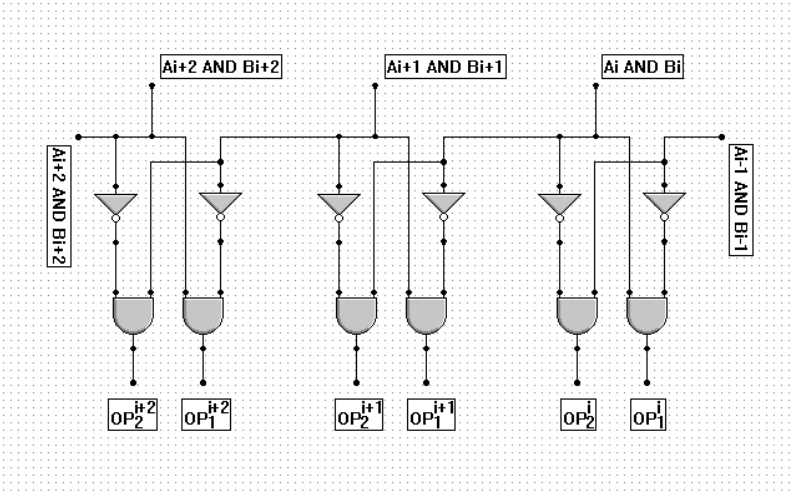

Ilustración 4. Módulo de lógica operacional de Booth-Karatsuba.
Fuente: elaboración propia.

Previamente a juntar los módulos algorítmicos diseñados, se va a repasar el hardware habitual utilizado en los circuitos actuales, con su coste computacional (Turing 1937, pp. 230-265). A partir de este punto, para dar paso a las descripciones hardware nos centraremos en operandos binarios de 8 dígitos; es decir, en los ejemplos n será igual a 8. Con ello, laimplementación clásica de la suma de dos números enteros, a y b, de longitud n suele quedar documentada:

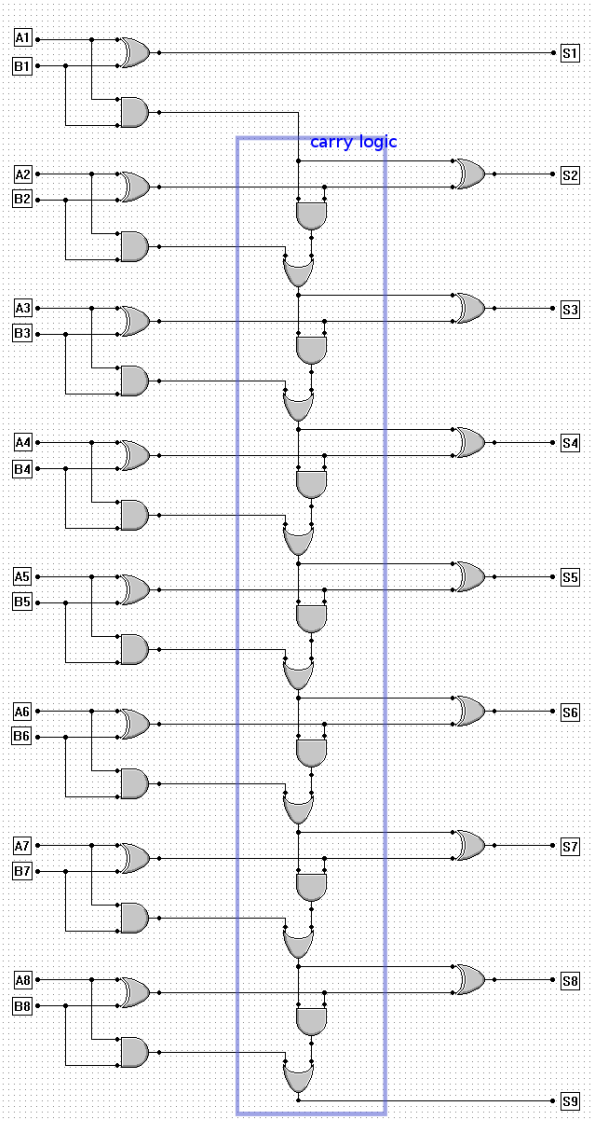

Ilustración 5. Implementación física del algoritmo clásico de adición de enteros.
Fuente: elaboración propia.

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

El algoritmo clásico de Ilustración 5 evidencia ese orden de complejidad lineal, ya que no es posible saber el valor final del último bit, en el ejemplo, el noveno bit, sin esperar a que se resuelvan los acarreos de los n bits anteriores, en la imagen: *carry logic*.

Siguiendo el hilo del pensamiento computacional tradicional (Von Neumann 1946, pp. 39-48), buscando reducir la mencionada complejidad algorítmica, es más que consabida una alternativa basada en la replicación de cómputos los cuales a nivel físico se realizan de forma paralela. Lamentablemente, la escalabilidad de esa variación, rememorada en Ilustración 6, es más que dudosa, como por ejemplo en el caso de las soluciones criptográficas actuales (Diffie-Hellman 1976, pp. 644–654); pues en cierto modo lo único que ofrece es intercambiar orden de complejidad en tiempo por orden de complejidad en espacio (Turing 1937, pp. 230-265).

Para respaldar el razonamiento y las anteriores afirmaciones, obsérvese la programación de la suma de dos números enteros, a y b, de longitud n con la técnica de adelantamiento de acarreo:

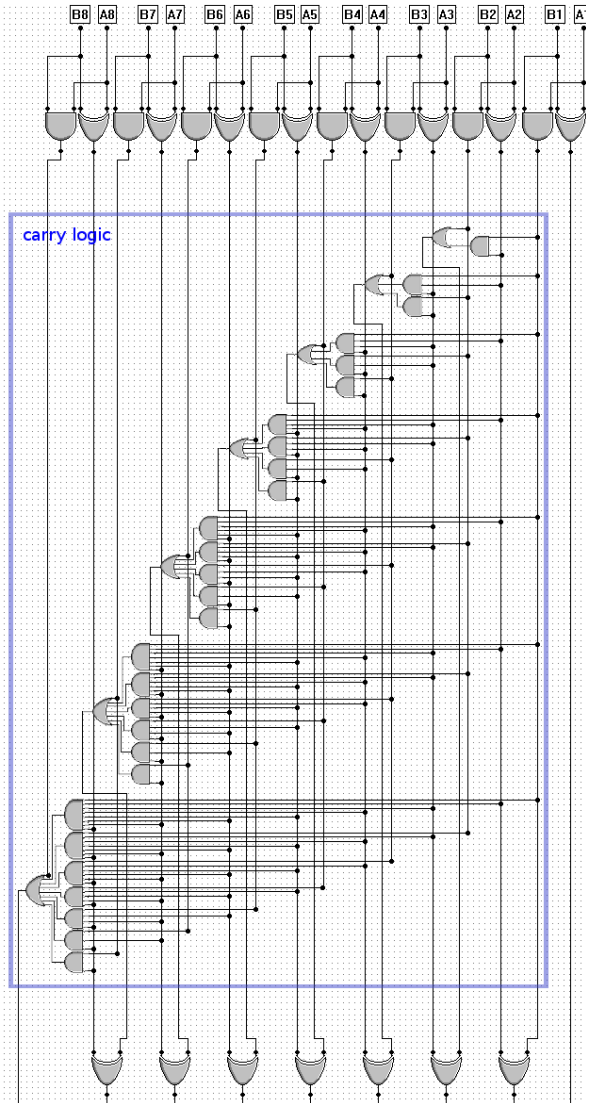

Ilustración 6. Implementación física de adición de enteros con carry lookahead.
Fuente: elaboración propia.

Ed. 27. Vol.7 Nº 4. Diciembre'18 - Marzo'19
DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

Asentados los 2 circuitos físicos anteriores, queda abonado el terreno para mostrar la implementación basada en los de Booth (Booth 1951, pp. 74) y Karatsuba (Karatsuba 1962, pp. 293–294) con la comparativa de esas referencias hardware, lo cual permitirá una evaluación justa, además de visualizar claramente las ventajas de nueva solución. Siempre recalcando la enorme influencia de los conceptos de Booth y Karatsuba en cualquier hallazgo, o en todos los diseños que se van construyendo.

Utilizando todo lo expuesto y logrado durante el análisis previo que se ha realizado y, como ya hemos repetido en varias ocasiones a lo largo del mismo, persiguiendo fusionar lo defendido en Ayuso 2017, pp. 1-9 y lo que se formula en Ayuso 2018, pp. 10-21, se tiene pues que la suma de dos números enteros, a y b, de longitud n con la técnica de Booth-Karatsuba puede expresarse con el siguiente soporte:

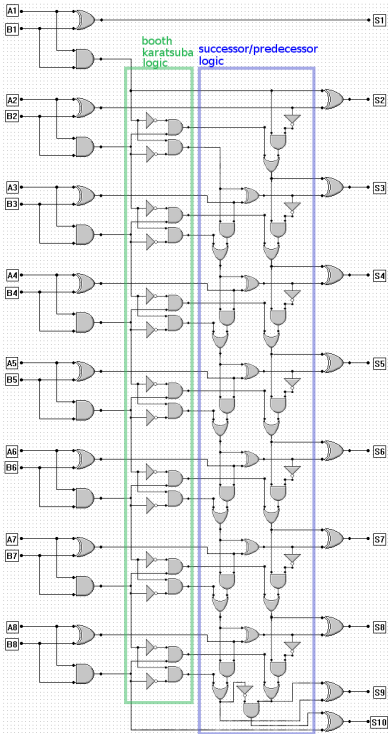

Ilustración 7. Implementación física de adición de enteros mediante Booth-Karatsuba.
Fuente: elaboración propia.

Ciertamente, de primeras se antoja como una solución aún más enrevesada y compleja que las que existen actualmente, pero a poco que se madura la implementación de Ilustración 7 puede intuirse una variación más interesante de la misma. Aun así, ya esta primera versión en cierto modo apunta a una solución intermedia, en el sentido de que mantiene constante el número de puertas lógicas necesarias.

Pero por simplicidad implementacional puede pensarse en realizar la operación descrita por Booth-Karatsuba en el punto donde se encuentran las operaciones de sucesor y antecesor; en el lugar donde acaban encontrándose, la propagación ordinal producto de la transformación algebraica, o mirado desde un enfoque clásico: el acarreo incrementativo y el acarreo decrementativo acaban colisionando en un dígito. Reconfigurada la perspectiva de difusión de la operación Booth-Karatsuba, redefinimos la suma de dos números enteros, a y b, de longitud n a través del siguiente nuevo soporte:

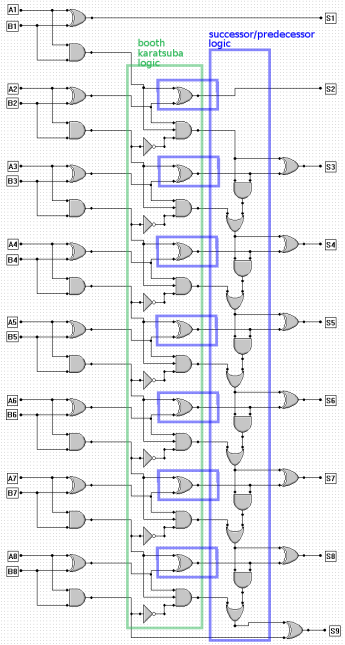

Ilustración 8. Implementación física de adición de enteros mediante Booth-Karatsuba.
Fuente: elaboración propia.

Ed. 27. Vol.7 Nº 4. Diciembre'18 - Marzo'19
DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

A estas alturas del artículo, consideramos necesario recapitular, para presentar también la aplicabilidad a otras operaciones sustractivas, a través de una adición modular; las cuales son de gran importancia en distintos ámbitos del conocimiento, y concretamente en el campo de la Criptología (Diffie-Hellman, 1976).

Para escenificarlo, ese matiz sustractivo básicamente afecta a la lógica de nuestro caso base. Volvemos a recurrir a una tabla de verdad para escudriñarlo:

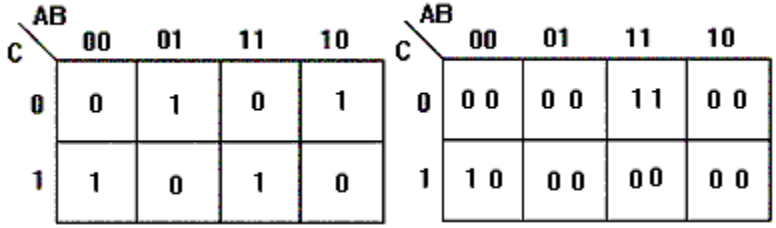

Ilustración 9. Tablas de verdad para adicción con sustracción simultánea.
Fuente: elaboración propia.

Puede observarse en la Ilustración 9 como se tiene que al incluir el proceso de sustracción dentro del módulo de nuestra operación algebraica cambia el criterio de acumulado o desacumulado en el resultado del cálculo, además de la lógica de propagación de la operación de Booth-Karatsuba a realizar.

Encaminado el código para el nuevo caso base, nos encontramos perfectamente en disposición abordar la suma de dos números enteros, a y b, módulo m con longitudes de n dígitos como:

```
OpBooth addmBoothKaratsuba(a, b, m, result, index, n)
{
if(n == 1)        // CASO BASE
{
        if(m[0]) {
                result[index] = XNOR(a[0], b[0]);
         return NOR(a[0], b[0]) ? ( -1 ) : ( 0 );
        } else { // m[0] == 0
                result[index] = XOR(a[0], b[0]);
         return AND(a[0], b[0]) ? ( 1 ^ -1 ) : ( 0 );
        }
        }

w = a / 2^(n / 2);
        x = a % 2^(n / 2);

        y = b / 2^(n / 2);
        z = b % 2^(n / 2);

        u = m / 2^(n / 2);
        v = m % 2^(n / 2);

d = n % 2 == 0 ? n / 2 : (n / 2) + 1;

OpBooth op1 =
 addmBoothKaratsuba(w, y, u, result, index + d, n / 2);
        OpBooth op2= addmBoothKaratsuba(x, z, v, result, index, d);

        switch(op1) {
                case ( 0 ):
                        switch(op2) {
                                case ( 0 ):
                                        return ( 0 );
                                case ( 1 ):
                                        successor(result, index + d + 1);
                                        return ( 0 );
```

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

```
                    case ( -1 ):
                        return ( -1 );
                    case ( 1 ^ -1 ):
                        successor(result, index + d + 1);
                        return ( -1 );
                    }
            break;
case ( 1 ):
            switch(op2) {
                    case ( 0 ):
                        return ( 1 );
                    case ( 1 ):
                        return ( 1 );
                    case ( -1 ):
                        return ( 1 ^^ -1 );
                    case ( 1 ^ -1 ):
                        successor(result, index + d + 1);
                        return ( 1 ^ -1 );
            }
            break;
                        successor(result, index + d + 1);
case ( -1 ):
            switch(op2) {
                    case ( 0 ):
                        predecessor(result, index + d + 1);
                        return ( 0 );
                    case ( 1 ):
                        return ( 0 );
                    case ( -1 ):
                        predecessor(result, index + d + 1);
                        return ( -1 );
                    case ( 1 ^ -1 ):
                        return ( -1 );
            }
            break;
```

```
                case ( 1 ^^ -1 ):
                        switch(op2) {
                                case ( 0 ):
                                        predecessor(result, index + d + 1);
                                        return ( 1 );
                                case ( 1 ):
                                        return ( 1 );
                                case ( -1 ):
                                        predecessor(result, index + d + 1);
                                        return ( 1 ^^ -1 );
                                case ( 1 ^^ -1 ):
                                        return ( 1 ^^ -1 );
                        }
                        break;
        }
        return ( 0 );
}
addmBoothKaratsuba(a, b, m, result)
{
        n = length(m);

switch( addmBoothKaratsuba(a, b, m, result, 0, n) ) {
                case ( 1 ):
                        result = successor(result, n + 1);
                        break;
                case ( -1 ):
                        result = predecessor(result, 1);
                        break;
                case ( 1 ^ -1 ):
                        result = successor(result, n + 1);
                        result = predecessor(result, 1);
                        break;
        }
}
```

Algoritmo 3. Algoritmo recursivo de adición modular por Booth-Karatsuba (addmBoothKaratsuba).

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

Efectivamente se comprueba cómo el añadir modularidad a la indicada operación aditiva, con la consiguiente sustrabilidad embebida, afecta en exclusividad al código correspondiente en el tratamiento del caso base en nuestro algoritmo.

Exactamente igual que para el caso de la adición previa se describió un circuito que simulaba el código de Algoritmo 2, se procede nuevamente con la operación declarada en Algoritmo 3. Conceptualmente ahora trabajaríamos sobre un conjunto finito (Galois 1846, pp. 381-384), a diferencia del comienzo de este trabajo, con lo que ese bit de *overflow* que podía producirse (Von Neumann 1946, pp. 39-48) en el cómputo final, desaparece. Evidentemente, por la naturaleza misma de la implementación física de máquinas de Turing (Turing 1937, pp. 230-265), en el mundo de la informática siempre trabajaremos dentro de la finitud, pero ahora lo haremos formalmente dentro de un cuerpo de Galois.

Puntualizada esa reflexión, iniciamos el análisis de la venidera Ilustración 10. Indiferentemente al ajuste algorítmico, entre la lógica y la física, se logra una doble visión de las reducciones computacionales usadas tanto por Booth como por Karatsuba, con la particularidad de que en el caso de la mencionada ilustración se tiene que la primera de las simplificaciones es aplicada desde el segundo dígito (Ayuso 2018, pp. 10-21) y la segunda desde el primero (Ayuso 2017, pp. 19-26).

Por otra parte, en Ilustración 10 puede verse un bit de salida etiquetado como *flag*, esto ocurre ya que el hardware plasmado da por sentado que la adición entre a y b, vista de manera tradicional, desborda el modulo en que se está trabajando. Expresado con un enfoque más algebraico: se proyecta contra un elemento ordinalmente congruente y, si no se controla la representación del mismo, ésta puede derivar en problemas computaciones.

Puntualizada esas reflexiones, nos llevamos a mundo físico la suma de dos números enteros, a y b, módulo m con longitudes de n dígitos así:

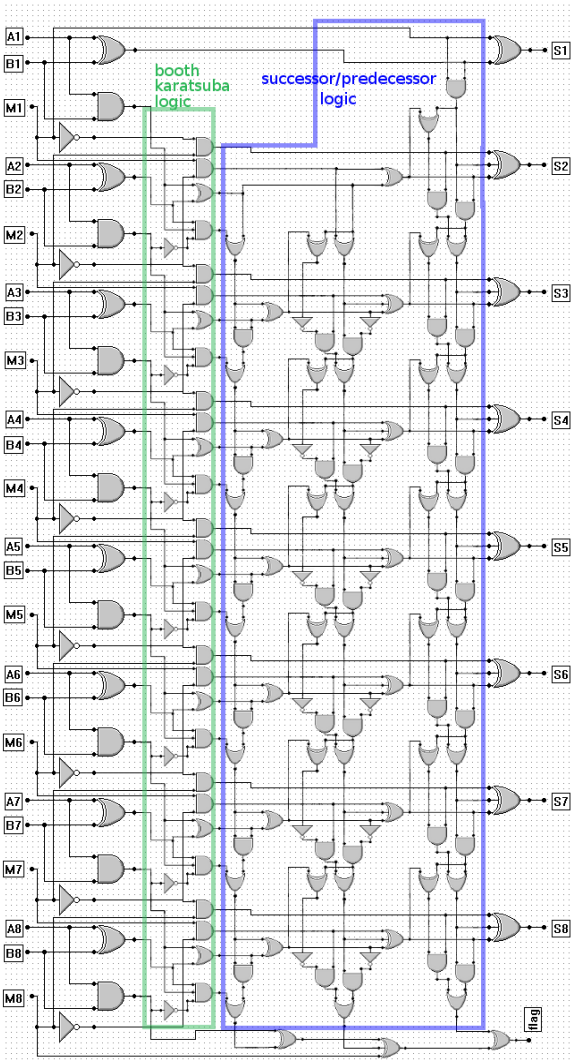

Ilustración 10. Implementación física de adición modular mediante Booth-Karatsuba.
Fuente: elaboración propia.

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

Con anterioridad a Ilustración 10 se ha disertado sobre la doble interpretación de las ideas publicadas por Booth y Karatsuba y cómo eran llevadas a cabo a nivel físico. Se contaba la ejecución a partir del segundo bit de menor peso (Ayuso 2018, pp. 10-21) en contraposición al empleo desde el primer bit de menor valor (Ayuso 2018, pp. 113-119) producto de ese efecto disociado de provenir de 2 ideas diferentes pero que resultan relativamente coincidentes; de manera que la actuación sustractiva quedaba dibujada de la segunda manera, y la aditiva, del primer modo. Pero la próxima Ilustración 11 demuestra cómo perfectamente puede afrontarse la sustrabilidad del contexto aditivo en que se enmarcan estos resultados basándose exactamente en el mismo paradigma de la fusión de los métodos de Booth y Karatsuba.

Obsérvese esa segunda visión, que versaría más adecuadamente como una especie de resultado de aplicar 2 iteraciones para la obtención del cómputo modular que se busca. Es decir, sería como aplicar lo mostrado en Ilustración 8 recurrentemente. De hecho, por esa reiteración conceptual, se ha optado por mostrar primero la alternativa de Ilustración 10, ya que puede tratarse de una versión más enriquecedora o más completa, que permite un entendimiento más profundo del material manejado.

Dispuesto todo y, por descontado, avanzando argumentalmente, la adición de dos números enteros, a y b, con la sustracción de un módulo m, los tres elementos con longitudes de n dígitos quedaría:

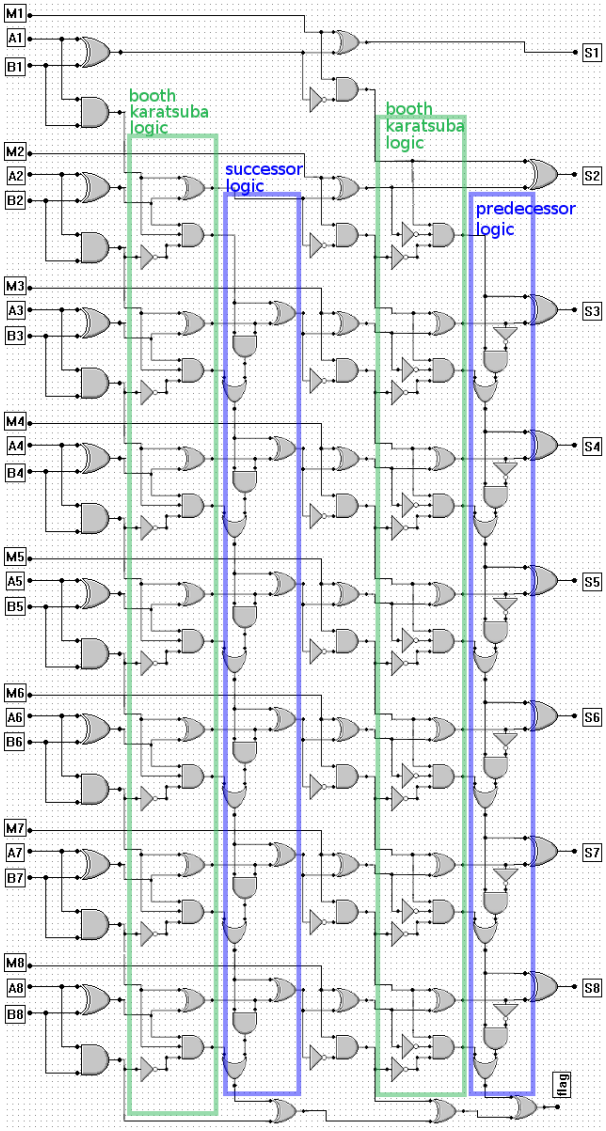

Ilustración 11. Implementación física de adición modular mediante Booth-Karatsuba.
Fuente: elaboración propia.

Ed. 27. Vol.7 Nº 4. Diciembre'18 - Marzo'19
DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

De Ilustración 11 queda asentada esa doble ejecución del algoritmo Booth-Karatsuba. Aporta un primer estrato aditivo, con su módulo de cálculo de sucesores: *successor logic*; y un segundo nivel sustractivo, con su módulo de antecesores: *predecessor logic*. De ahí que anticipáramos la implementación diseñada en Ilustración 10, que tal vez sea más enrevesada pero desde cierto punto de vista resulta mucho más interesante y divertida. Además de posiblilitar un mejor careo entre Karatsuba y Booth.

Entramos ahora a completar el estudio del bit etiquetado como *flag* en estas postreras soluciones modulares. Se ha explicado ya que, aunque tradicionalmente podía verse como un *overflow*, inherente a las implementaciones físicas, en nuestro análisis adquiere el papel de señalizador del problema representacional fruto de expresar los elementos de nuestra estructura algebraica como números enteros por dígitos pesados: por regla general, de izquierda a derecha. Este problema representacional, es fácilmente corregible incorporado el siguiente módulo físico al anterior hardware modular:

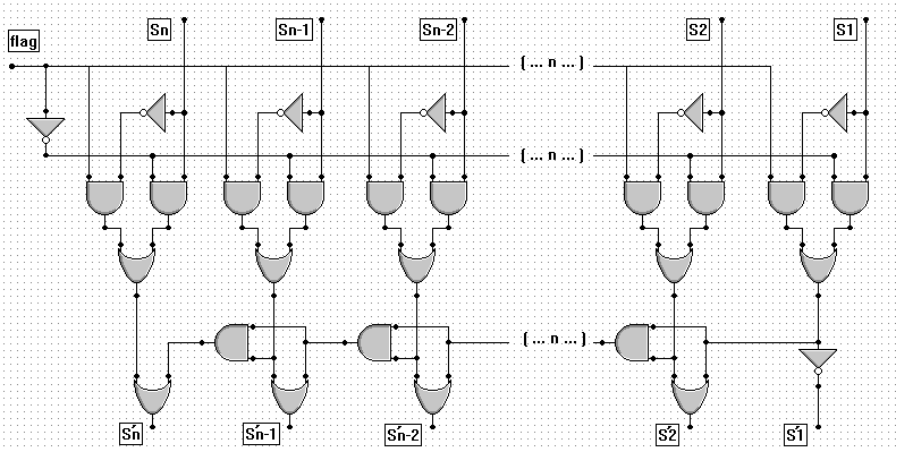

Ilustración 12. Módulo de lógica para congruentes aditivos de Booth-Karatsuba.
Fuente: elaboración propia.

El módulo de Ilustración 12 se explica bastante a sí mismo. Simplemente tenemos un primer nivel a modo de conducta de *multiplexor* (Von Neumann 1946, pp. 39-48) en función del valor del citado *flag*, y un último paso de obtención del congruente. Añadiendo dicho módulo a las implementaciones modulares dadas, tendríamos que, aunque estas siempre reduzcan en función del

módulo, en cualquier caso obtendríamos un resultado correcto, desde un punto de vista algebraico, y simplementemente teniendo en cuanta si el *flag* está activo o no, se podría trabajar de la forma en la que se explica en artículos como: Ayuso 2015, pp. 255-221 o, reutilizando la idea, en dimensiones mayores como en: Ayuso 2018, pp. 10-35, apoyándonos en elementos congruentes para seguir operando matemáticamente.

3. CONCLUSIONES

Los algoritmos definidos de forma independiente por Booth en 1950 y por Karatsuba en 1960, y que llevan sus nombres, se basan en una idea fundamental muy similar, desde un punto de vista de simplificación algebraica, aunque por otra parte es cierto que el recorrido sobre los elementos de la estructura lo realizaban cada uno de una manera muy distinta. No por ello conlleva ningún tipo de incompatibilidad entre los mimos, más bien todo lo contrario: se pueden complementar el uno al otro, e incluso podría llegar a decirse que en cierto modo se trata del mismo concepto.

En conclusión, el concepto propuesto por Booth y Karatsuba comulgan el uno con el otro, permitiendo fusionarse en uno solo. Soportan un enfoque orientado en exclusiva a la suma y resta de enteros, uniéndose en una solución hardware para el cómputo de ese tipo de adicciones. Paradójicamente los dos autores documentaron sus algoritmos exactamente para la misma operación, algo que en principio podría considerarse intrascendente, pero que con lo visto en este artículo puede que no sea ni mucho menos casual.

Como posible futura línea de investigación queda allanado el camino a una implementación física de un módulo critográfico, que complementado con los algoritmos referenciados en las distintas citas bibliográficas, podría formar parte de una solución para cálculos de cifrado y autenticación. La medición de retardos, el estudio de escalabilidad, el rendimiento computacional... son temas totalmente abiertos tras la exposición realizada durante las páginas de las que consta el presente documento, propiciando un más que excitante trabajo de estudio y esquematización de datos empíricos y diversas evaluaciones.

DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59

4. REFERENCIAS BIBLIOGRAFICAS

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

Burks, A., Goldstein, H. y Von Neumann, J. (1946). Logical Design of an Electronic Computing Instrument, *The Origins of Digital Computers,* pp. 39-48.

Booth, A. D. y 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. Math.,* 4(2), pp. 236-240.

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

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

Euclid of Alexandria. (1557). *Elements*, T.L. Heath's.

Reitwiesner, W. G. (1960). *Binary Arithmetic,* pp. 231-308.

Newton, I. (1669). *De analysi per aequationes numero terminorum infinitas*, 206 ff.

Pascal, B.(1654). *Traité du Triangle Arithmétique.*

Karatsuba, A. (1962). Multiplication of multidigit numbers on automata. *Doklady Akademii Nauk SSSR*, 145, pp. 293–294.

Bellman, R. (1954). The theory of dynamic programming, *Bulletin of the American Mathematical Society,* pp. 503–516.

Ayuso, J. (2015). Booth algorithm modular arithmetic operations of multiplication, *3C TIC,* 4(4), pp. 255-221.

Ayuso, J. (2016). Booth algorithm operations modular inverse, *3C TIC,* 5(2), pp. 28-41.

Ayuso, J. (2016). Booth algorithm in signed-digit representation, *3C TIC*, 5(3), pp. 33-43.

Ayuso, J. (2017). Booth algorithm in modular exponentiation operations, *3C TIC*, 6(2), pp. 1-12.

Ayuso, J. (2017). Booth algorithm hardware operations addition and subtraction, *3C TIC*, 6(3), pp. 1-9.

Ayuso, J. (2017). Booth algorithm in arity with multiple operands, *3C TIC*, 6(4), pp. 19-26.

Ayuso, J. (2018). Karatsuba algorithm operations exponentiation , *3C TIC*, 7(1), pp. 13-20.

Karnaugh, M. (1953). The Map Method for Synthesis of Combinational Logic Circuits, *Transactions of the American Institute of Electrical Engineers part I,* pp. 53-217.

Diffie, W., y Hellman, E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory,* 22(6), pp. 644–654.

Shannon, C. E. (1938). A Symbolic Analysis of Relay and Switching Circuits. *Trans. AIEE,* 57(12), pp. 713–723.

Ayuso, J. (2018). Booth algorithm modular arithmetic for scalar multiplication operations, *3C TIC*, 7(2), pp. 10-35.

Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. *London Mathematical Society,* 42, pp. 230-265.

Galois, É. (1846). Oeuvres mathématiques, *Journal de Mathématiques Pures et Appliquées,* 11, pp. 381-384.

Ayuso, J. (2018). Karatsuba algorithm in additive context, *3C TIC*, 7(3), pp. 10-21.

Ed. 27. Vol.7 Nº 4. Diciembre'18 - Marzo'19
DOI: http://dx.doi.org/10.17993/3ctic.2018.62.30-59