



DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA
INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

“CIRCUITOS DIGITALES”



TRABAJO DE INVESTIGACIÓN

TEMA:

DETECTOR DE ERRORES MEDIANTE PARIDAD PAR O IMPAR

AUTORES:

HUGO DAVID ANDRADE SOLORZANO

JAMES MARSHALL FLORES PROAÑO

DOCENTE:

ING. DARWIN OMAR ALULEMA FLORES

NRC: 9434

Quito - Ecuador, 15 de junio de 2020

1. PLANTEAMIENTO DEL PROBLEMA

La detección de errores en códigos binarios es muy importante para saber identificar si existe algún fallo en sistemas de comunicación digital, los cuales pueden sufrir alteraciones, por diferentes fenómenos físicos, al momento de la transmisión de información que parte desde un transmisor hacia un receptor. Es por ello que nuestro propósito en este trabajo de investigación es realizar un programa (app) en el cual se pueda realizar la detección de errores en números binarios sea cualquier longitud que se nos sea presentada, a través de la paridad par o impar, en este programa debe ser implementado un algoritmo propio para realizar la detección del error de cualquier número binario, para realizar este programa no se admite el uso de librerías que ya se encuentren preestablecidas, todo esto con el uso del software online propuesto que es MIT APP INVENTOR 2.

2. OBJETIVOS

2.1.OBJETIVO GENERAL

Diseñar e implementar un programa (App) que realice la detección de error de números binarios, mediante la programación en un entorno gráfico, a través de bloques que ofrece el software online MIT APP INVENTOR 2.

2.2.OBJETIVOS ESPECÍFICOS

- Buscar el concepto de errores binarios para un mejor entendimiento del programa y así ampliar el conocimiento sobre cómo funcionan cada uno de los errores en la paridad par o impar.
- Investigar el funcionamiento del código de detección de errores en los números binarios que sean presentados dentro de nuestro programa, ya sea para la paridad par o impar.
- Realizar un algoritmo propio en el desarrollo del programa, el cual solventa el problema de detectar y determinar si existe error en un número binario de cualquier longitud. Todo esto con el apoyo de la investigación que se habrá realizado y de los artículos de

diferentes autores que permitan tener una idea mucho más clara y significativa de lo que está por realizarse, relacionándolos a su vez, con nuestro programa.

3. ESTADO DEL ARTE

Generador de bits de paridad fotónica digital uniforme

Fue desarrollado por F. K. Law, M. Rakib Uddn, Nur Musyiirah Masir¹ and Yong Hyub Won

En el año de 2018 el 30 de septiembre al 4 de octubre de ese año se dio una conferencia del tema junto con una explicación y fue añadido a la IEEE el 08 de noviembre del 2018

El objeto del estudio es presentar un novedoso diseño de un sistema digital generador de bits de paridad paritaria fotónica basado en un solo fotón micro-ring resonador. Este trabajo proporciona la operativa principio del circuito propuesto, con sus entradas son el señales eléctricas que se alimentan en el micro-ring resonador, y la salida de la señal óptica se toma del puerto de caída del anillo

Luego se nos presenta una introducción en la cual se nos informa primero sobre los componentes y medidas que estos tienen y los cuales van a ser usados además de cómo va a ser su diseño cuales van a ser las medidas de los componentes el orden en cómo van a ser colocados para que pueda producir la modulación electro-óptica en el generador se ingresa seleccionado 0,45V como entrada eléctrica representación del estado lógico digital de 0, y 0,65V como el estado lógico digital 1 para el funcionamiento del modo XOR. . [1]

Se relaciona con nuestro proyecto a partir de 2 estados lógicos que son los que ingresan este generador de paridad nos va a entregar los bits de paridad y estos son los que podemos usar dentro de nuestro programa para comprobar si existe error y de ser así detectar el error puesto que lo que este proyecto nos entrega como resultados bits de paridad par e impar este es apto para nuestro uso

La investigación se presentó en Reston, VA, EE. UU. Y fue desarrollado en Electrical and Electronic Engineering Programme Area, Faculty of Engineering Universiti Teknologi Brunei (UTB), Gadong, Brunei Darussalam 2KAIST, Daejeon, South Korea

El correo del autor es :

rakib.uddin@utb.edu.bn

FK Lawl, MR Uddn, NM Masir y YH Won, "Digital Photonic Even Parity Bit Generator", 2018 IEEE Photonics Conference (IPC) , Reston, VA, 2018, pp. 1-2, doi: 10.1109 / IPCon.2018.8527192

En códigos de control de errores limitados de longitud de ejecución contruidos a partir de códigos de producto de verificación de paridad binaria

Fue desarrollado por: Peter Farkaš, Tomáš Janvars y Katarína Farkašová, Eugen Ružický

La investigación por parte de Peter Farkaš, Tomáš Janvars fue desarrollada en Institute of Multimedia ICT, Faculty of electrical Engineering and Information Technology, Bratislava, Slovakia su correo es:

p.farkas@ieee.org

Y por parte de Katarína Farkašová, Eugen Ružický en Institute of Applied informatics,.

Faculty of Informatics Pan European University, Bratislava, Slovakia

Su correo es :

Eugen.Ruzicky@paneurouni.com

El en objetivo de este estudio se nos muestra los códigos de control de errores podrían ser contruidos a partir de códigos de productos de control de paridad única bidimensional en

caso de que en al menos uno de los códigos de los componentes tiene una longitud de código uniforme. La principal ventaja es que esto podría hacerse sin ningún tipo de redundancia y sin modificación de la codificación y procedimiento de decodificación utilizado para el código de control de errores subyacente.

Se nos da a conocer los códigos de algunos productos con los cuales se va a realizar la comprobación en códigos del control de error a partir de la paridad binaria además también se nos muestran los códigos de productos binarios obtenidos de una sola, los códigos de control de paridad (SPC) podrían convertirse en RLL-ECC en el caso de que al menos un código de componente tenga incluso una palabra clave longitud esto nos ayudaría comprobar si nuestro programa funciona de manera correcta puesto que ambos tratan de encontrar los errores que se puede dar con la paridad binaria y al comprobar podríamos ver si los resultados de nuestro programas son correctos o no también como podría ser que nuestro programa este bien hecho y en cambio existan errores dentro de este control de errores

Los códigos de duración limitada (RLL) son códigos con restricciones a las ejecuciones de símbolos idénticos en sus palabras clave secuencias dictadas por los requisitos prácticos de los sistemas. [2]

P. Farkaš, T. Janvars, K. Farkašová y E. Ružický, "En códigos de control de errores limitados de longitud de ejecución contruidos a partir de códigos de productos de verificación de paridad única binaria", 2018 Cybernetics & Informatics (K&I) , Lazy pod Makytou, 2018, pp 1-4, doi: 10.1109 / CYBERI.2018.8337547.

Arquitectura de codificador optimizada para códigos de verificación de paridad de baja densidad estructurados de longitud corta

Su autor es Silvia Anggraeni, Fawnizu Azmadi Hussin and Varun Jeoti

Fue realizado en Electrical and Electronics Engineering Department Universiti Teknologi PETRONAS Bandar Seri Iskandar, 31750 Tronoh, Perak, Malaysia

La fecha de la conferencia donde se presentó fue 3-5 de junio de 2014 en Kuala Lumpur, Malasia y fue añadido a IEEE Xplore el 04 de agosto de 2014

Los correos de los autores son :

silvia31804@yahoo.com,

fawnizu@petronas.com.my

varun_jeoti@petronas.com.my

El objetivo de este artículo propone una arquitectura para codificador de verificación de paridad de baja densidad estructurado. La arquitectura propuesta admite la limitación de los pines de entrada / salida de la matriz de compuerta programable de campo mediante la división de bits de información. La división de bits de información genera latencia de codificación. La arquitectura propuesta no almacena la matriz requerida para la multiplicación en bits y no utiliza el desplazamiento cíclico del desplazador de barril

Gracias a esto se puede trabajar con longitudes cortas de código las cuales van a abarcar de los 1000 a los 2000 bytes esto nos puede ser mucha ayuda a la hora de buscar datos de paridad ya sea par o impar puesto que al ser bits con una longitud más corta el programa detectará más rápido los errores y podemos identificarlos de una manera más fácil ya que no tendremos que estar buscando mucho dado que la longitud del código fue reducida. Peor esto no significa que su rendimiento de información vaya a ser bajo ya que realizado la arquitectura esta puede llegar hasta un rendimiento de información de 30.178 gbps con

área de elemento lógico y siendo la longitud de su código N igual a 1944 con la tasa de código R igual a $5/6$.

Para realizar el programa se usa el método de BMVM y el esquema de este es:

Paso 1 que su submatriz p y el vector a usar en este caso S sean binarios, Paso 2 Representar los no ceros de la sub-matriz P por la columna, a ubicación de la columna en este paso se utilizará por los bits de información s . El resultado se almacena en P' , Paso 3 sustituya los no ceros del paso 2 por los valores de s y guarda el resultado en $P'' = s(P'(x, y))$, Paso 4 Hacer la operación XOR en P'' para obtener los bits de paridad p .

Con este método es que vamos a ir obteniendo los bits de paridad que posteriormente usaremos en nuestro programa ver si se encuentra algún error en estos y de era si nuestra app implementada procederá a detectar los errores que existan. [3]

S. Anggraeni, FA Hussin y V. Jeoti, "Arquitectura de codificador optimizada para códigos de verificación de paridad de baja densidad estructurados de corta duración", 5ta Conferencia Internacional 2014 sobre Sistemas Inteligentes y Avanzados (ICIAS) , Kuala Lumpur, 2014, pp. 1-4 , doi: 10.1109 / ICIAS.2014.6869526.

4. MARCO TEÓRICO

¿Qué es un bit?

Es una unidad mínima de información, que puede tener solo dos valores (cero o uno).

Cuando recibimos un paquete de bits, estos pueden estar mal, para saber o reconocer donde ha habido un error utilizamos:

El **bit de Paridad**: es como un comprobador. Comprueba si los valores llegan bien o no. Comprueba también si los bits son pares o impares.

Bit de Redundancia: Añade más bits para encontrar el error.

Código de paridad

Consiste en agregar un uno o un cero a una trama (en el transmisor), con el objetivo que la cantidad de unos de la trama completa (con el bit agregado) posea una cantidad par o impar

de unos según sea el caso (paridad par o impar). En el receptor se "cuentan" los unos que llegan, y así se sabe si la trama llegó bien.

Bit de paridad

Existe dos tipos, Bit de Paridad Par e Impar.

El bit de paridad será un 0 si el número total de 1 a transmitir es par, y un 1 si el número total de 1 es impar.

Códigos de paridad

Paridad par

El bit que se agrega será tal, que haga que el número de unos de cada combinación sea par.

Por ejemplo:

Combinación	Bit de paridad par
0101	0
0111	1
1000	1
1001	0

Paridad impar.

El bit de paridad será un 1 si el número total de 1 a transmitir es par y un 0 si el número total de 1 es impar.

Normalmente el bit de paridad se añade a la izquierda del carácter original.

El bit que se agrega será tal, que haga que el número de unos de cada combinación sea impar.
Por ejemplo:

Combinación	Bit de paridad impar
0101	1
0111	0
1000	0
1001	1

Control de errores

El problema que tiene el receptor de poder verificar que hay un error se puede resolver utilizando una codificación de control de errores.

La idea central de la codificación de control de errores es agregar un bit adicional en la información a ser enviada para que se detecte el error y se corrija. Hay muchas codificaciones de control de errores. La más simple es el bit de paridad.

A cada byte que se transmita se le agrega el bit de paridad. Este bit se utiliza para comprobar que la información se haya entregado con exactitud.

El bit de paridad para cada byte se implanta de tal manera que todos los bytes tengan una cantidad impar o una cantidad par de bits 1.

Detección de errores

La verificación de paridad es la técnica más simple para detectar errores en la comunicación.

Sin embargo, aunque puede detectar muchos errores no resulta infalible, ya que no es capaz de detectar la disposición cuando en el mismo byte se cambia un número par de bits por el ruido eléctrico.

La verificación de paridad se usa no solo en las comunicaciones, sino además para probar los dispositivos de almacenamiento de memoria. Por ejemplo, muchas computadoras personales realizan una verificación de paridad siempre que se lee un byte de datos en la memoria.

¿Cómo funciona?

Supongamos que se tienen códigos de datos de 7 bits y se agrega un bit adicional, que es el bit de paridad, para así formar un código de datos de 8 bits. Existen dos métodos que se pueden utilizar: paridad par y paridad impar.

Como muestra, se puede tomar el método de paridad par. Se haría lo opuesto si se tomara el método de paridad impar.

Método de paridad par

Este método indica que el bit de paridad a agregar debe ser de tal manera que la cantidad total de 1 en el código final sea par. Por ejemplo:

Código de informacion	Cantidad de 1 en el codigo	Bit de paridad
0010010	2	0
1110110	5	1

Por tanto, para el primer código de 7 bits: 0010010, con una cantidad par de 1, el código de 8 bits transmitido será: 00100100, con una cantidad par de 1.

Para el código de 7 bits 1110110, con una cantidad impar de 1, el código de 8 bits transmitido será 11101101, con una cantidad par de 1.

Después que el receptor obtenga los 8 bits, verificará la cantidad de 1 en el código recibido, si la cantidad de 1 es par, eso significa que no hay error, si la cantidad es impar, eso significa que ha ocurrido un error.

Pero existe inconvenientes con estos métodos de paridad si el código 1110110 se convierte por el sonido de la línea en 11111001, esto hace que se produzca un error de 2 bits entonces esto produce que no se pueda detectar el error

La paridad sirve para detectar error de cantidad impar de errores en un byte recibido sin embargo existe una cantidad par de errores, el verificador de paridad no lograra dar con el error y nos tirara sin error el resultado por lo tanto esto es un inconveniente para nosotros

Cuando la paridad calculada del byte recibido no coincide con el valor del bit de paridad recibido, se dice que se ha producido un error de paridad y normalmente el byte se descarta.

En caso que haya un error, el receptor avisará al transmisor para que vuelva a enviar el código nuevamente.

5. DIAGRAMAS

5.1. DIAGRAMA DE BLOQUES

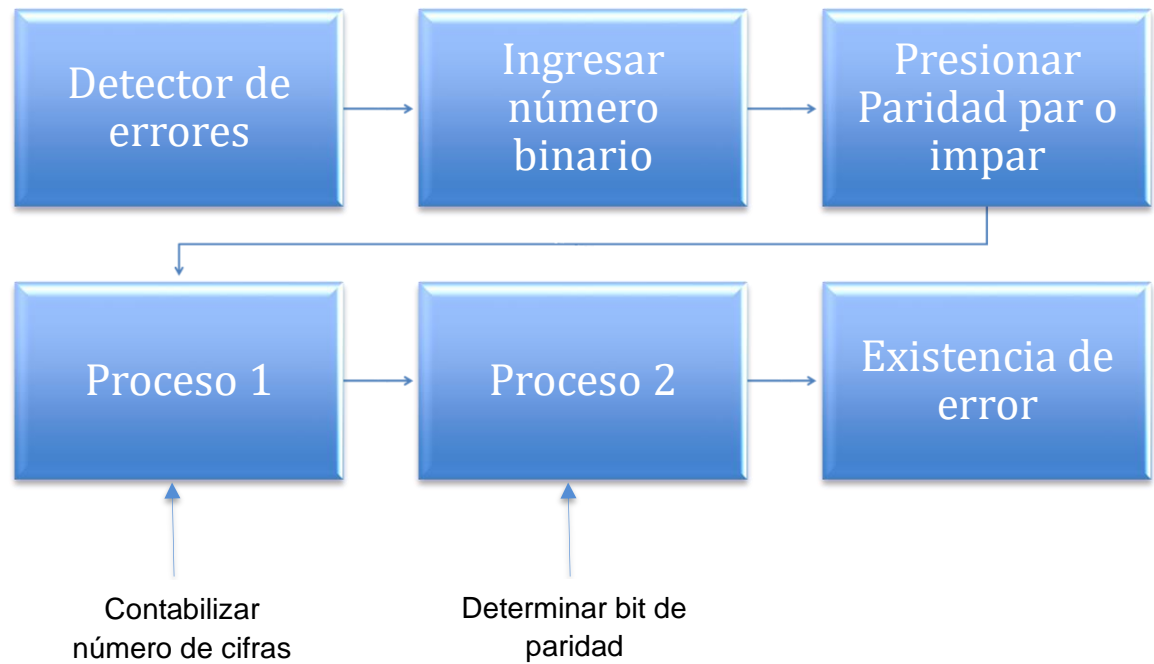


Diagrama 1.1 – Diagrama de bloques.

5.2. DIAGRAMA UML (CASO DE USO – CLASE)



Diagrama 1.2 – Diagrama de uso.

5.3.DIAGRAMA DE FLUJO

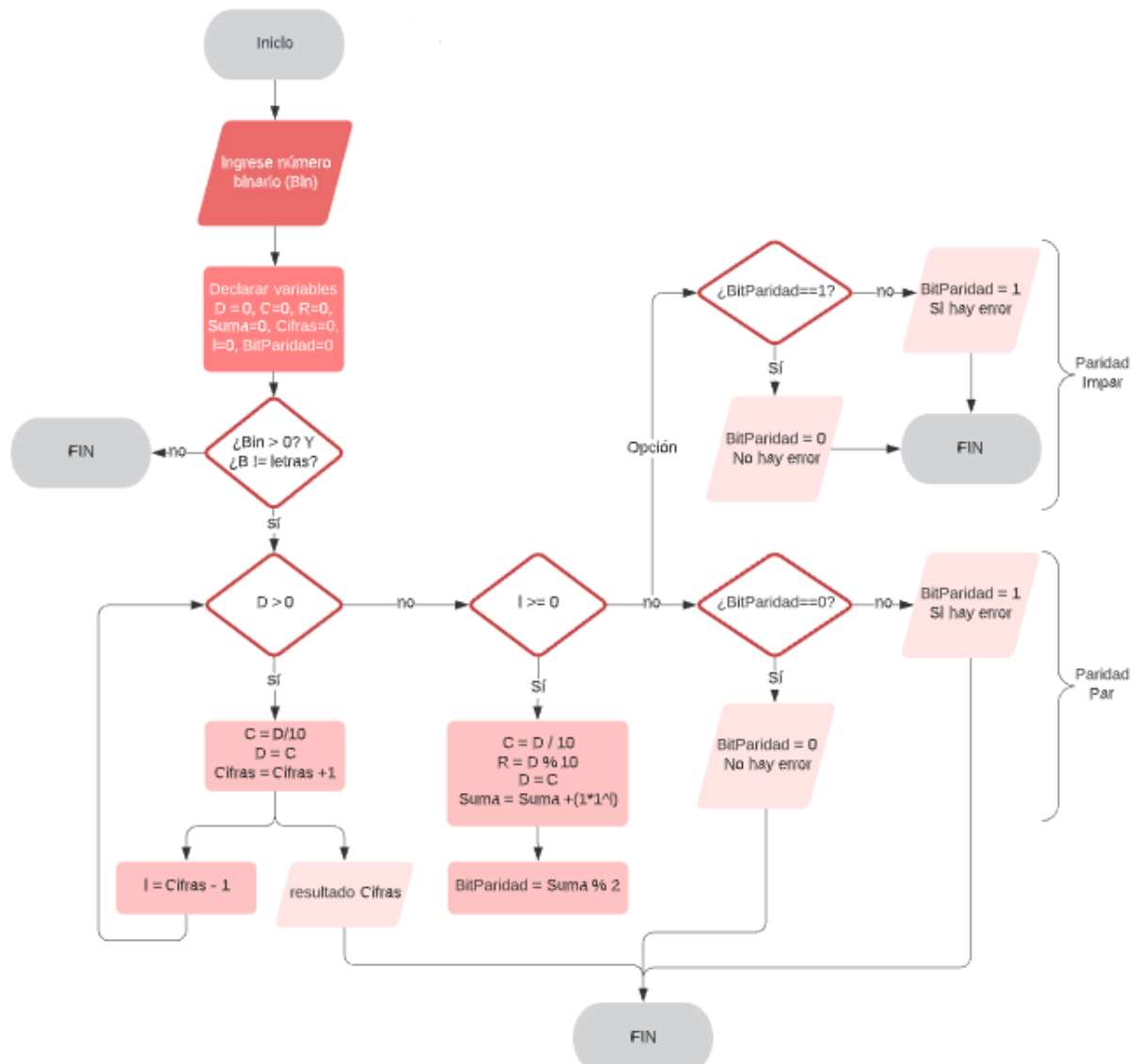


Diagrama 1.3 – Diagrama de flujo.

6. LISTA DE COMPONENTES

Componentes utilizados	
Software online MIT APP INVENTOR 2	Software utilizado para crear la app, cuya programación se la realiza mediante un entorno gráfico (programación por bloques).
Compilador móvil AI Companion	Después de realizar la programación en el software MIT APP INVENTOR 2, es necesario realizar las respectivas pruebas mediante el compilador móvil AI Companion para comprobar que el programa se esté ejecutando de manera correcta
Software Java NetBeans (Apoyo)	Software utilizado como método de apoyo para la elaboración del mismo programa en un entorno diferente (No gráfico).

Tabla 1.1 – Lista de componentes utilizados.

7. MAPA DE VARIABLES

- **Variables utilizadas para el funcionamiento de la app**

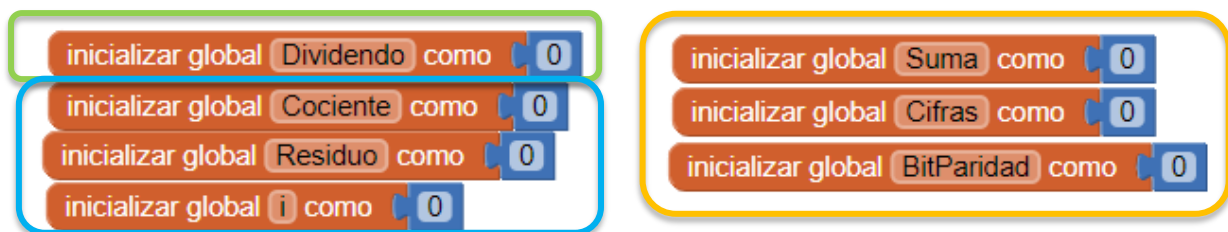


Figura 1.1 – Variables utilizadas.

Como podemos observar la figura 1.1 nos encontramos con todas las variables que han sido utilizadas para el correcto funcionamiento del programa (App).

En el recuadro verde podemos observar la variable de entrada, la cual corresponde al número binario ingresado por el usuario.

En el recuadro celeste podemos observar variables auxiliares que se han utilizado para realizar todos los cálculos matemáticos que fueron necesarios para que el funcionamiento de la App sea correcto.

Y, por último, en un recuadro amarillo podemos observar las variables de salida, las cuales representan a los resultados que se visualizan en la interfaz del programa. Correspondientes al

número total de cifras, la suma total de 1's y el Bit de paridad que determina la existencia de error. (También ilustrados en la figura 1.2.).

- **Interfaz de usuario**

Detector de errores

Ingrese el número binario

11101

Seleccione una opción para detectar el error

Paridad Par

Paridad Impar

El número de cifras es: 5

La suma total de 1's es: 4

No hay error. Bit de paridad es: 0

Figura 1.2 – Ejemplo - Variables de salida visibles en la interfaz de usuario.

- **Tabla de variables visibles o no visibles**

Detector de errores por paridad par o impar		
Variables	Visibles	No visibles
Dividendo	X	
Cociente		X
Residuo		X
i		X
Suma	X	
Cifras	X	
BitParidad	x	

Tabla 1.2 – Variables visibles y no visibles.

Ahora bien, necesitamos saber el tipo de cada variable que se ha utilizado, además de la función que realiza cada una en el programa. Esto se ilustra en la tabla 1.3.

Detector de errores por paridad par o impar		
Variables	Tipo	Función
Dividendo	Long	A esta variable se le asigna el valor del número binario ingresado por el usuario.
Cociente	Long	A esta variable se le asigna el resultado de la división, entre “Dividendo” dividido para 10.
		Al final del bucle While, asignamos el valor de “Cociente” a la variable “Dividendo”.
Cifras	Entero	Esta variable es inicializada con el valor de 0. Después, mediante un bucle While, actúa como un contador, el cual va contabilizando el número de cifras del número binario.
i	Long	A la variable “i” se le asigna el valor de “Cifras” – 1. Actúa como exponente en el segundo bucle While.
Residuo	Long	La variable “Residuo” resulta del módulo entre “Dividendo” dividido para Cociente”.
Suma	Long	La variable suma es inicializada con el valor de 0. Es un contador de 1’s que depende de las variables “Residuo” e “i”.
BitParidad	Long	La variable “BitParidad” arroja valores de 1 ó 0 para determinar si existe error en el número binario, ya sea por paridad par o impar.

Tabla 1.3 – Variables visibles y no visibles.

8. EXPLICACIÓN DEL CÓDIGO FUENTE

Programación en MIT App Inventor 2 mediante bloques

- Presionando el botón Paridad par

Al momento de presionar el botón Paridad par, el programa ejecutará la siguiente secuencia de bloques (Ilustrada en la figura 1.3), en la que se resuelve todo el algoritmo correspondiente a la detección de errores por paridad par.

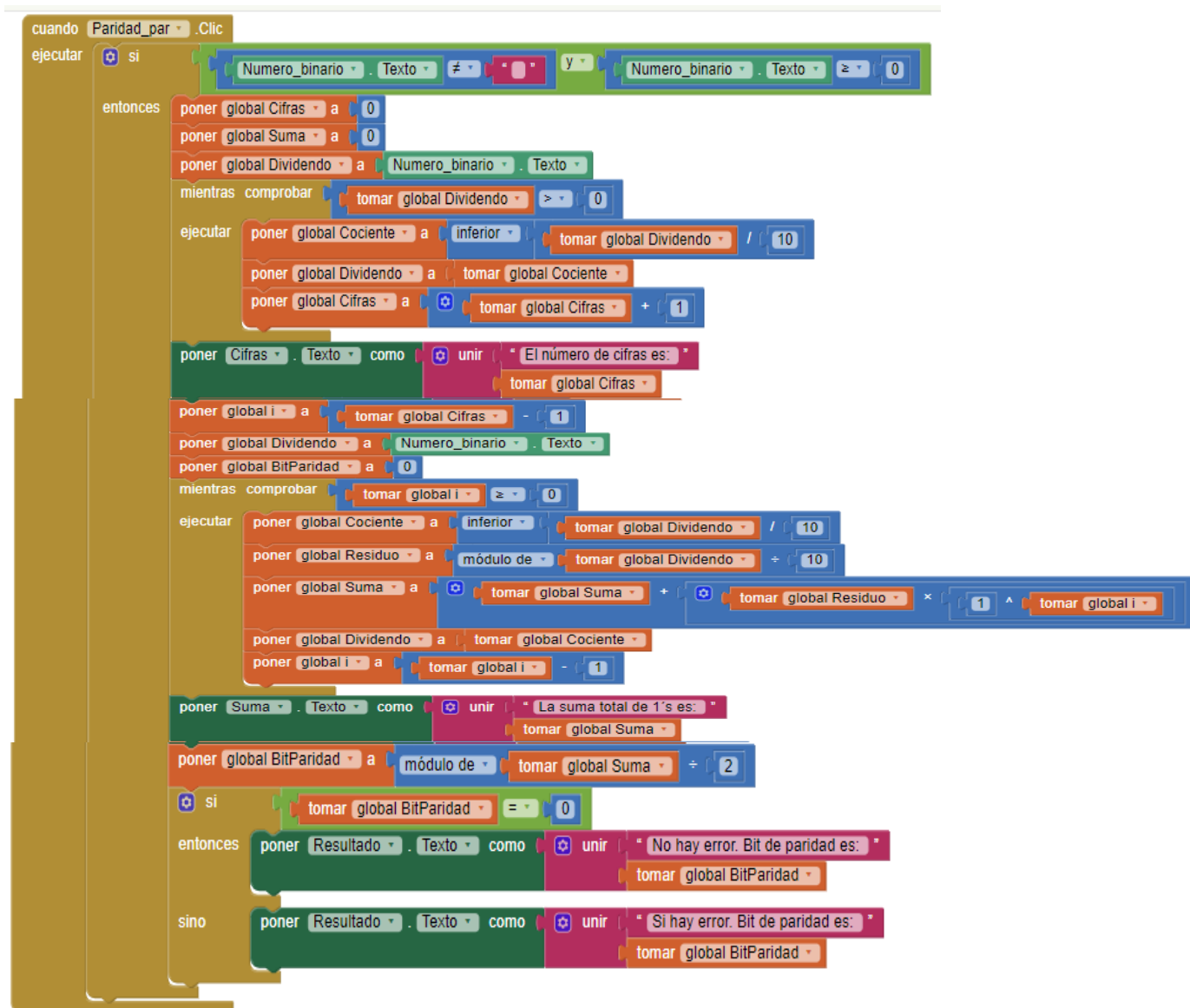


Figura 1.3 – Detección de errores mediante paridad par.

El programa básicamente consiste en realizar divisiones sucesivas, mediante dos ciclos While o Entonces. El primer ciclo nos permite realizar el conteo del número de cifras totales del

número binario ingresado por el usuario que puede ser de cualquier longitud. En la figura 1.4. podemos ver el proceso en el cual se basa este apartado.

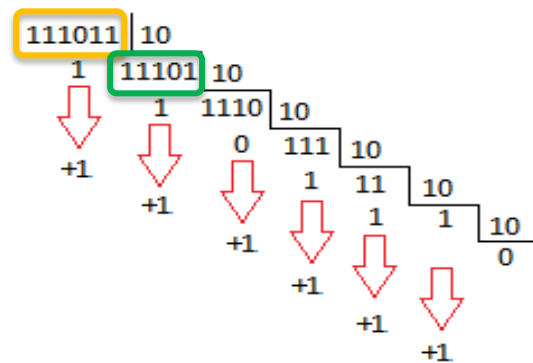


Figura 1.4 – Ilustración del funcionamiento del conteo de cifras.

Como podemos observar en la figura 1.4. El dividendo (Recuadro amarillo) siempre se va a dividir para 10. Y el cociente resultante (Recuadro verde) es un número entero que va a asignarse nuevamente a la variable dividendo. Y así sucesivamente, hasta que el dividendo sea 0. Cada vez que ocurra este proceso (mediante el bucle repetitivo While) se va a sumar en una unidad el número de cifras.

Ahora bien, si observamos la figura 1.5 y teniendo el número de cifras obtenidas en el primer ciclo While, nosotros creamos una variable i , la cual es igual al número de cifras -1. Este valor asignado en i , es el exponente (Recuadro amarillo) de la base 1 que va a multiplicarse por el valor del Residuo obtenido (Recuadro verde). Entonces al aplicar el segundo ciclo While y al sumar todos estos valores asignándolos a la variable Suma, obtenemos el número total de 1's. Los cuales nos indicarán si el número es par o impar realizando el módulo entre la variable suma y el valor de 2 y esto asignándolo a la variable BitParidad. De esta manera será posible detectar o determinar si existe o no error en el caso de paridad par, mediante una estructura condicional If-Else.

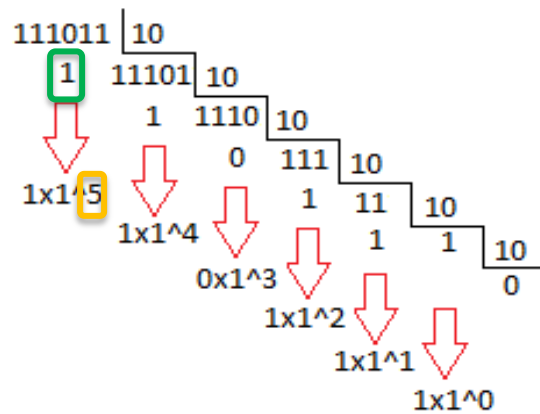


Figura 1.5 – Ilustración del funcionamiento del conteo de 1's.

Para entender de mejor manera el funcionamiento del programa, hemos realizado un código mediante Java NetBeans, en el que explicamos paso a paso, el proceso y resolución del programa.

Declaración de variables

```
package PaqueteLogica;

import java.util.Scanner;

public class Programa {

    public static void main(String[] args)
    {
        // Declaración de variables
        long Numero_binario;
        long Dividendo;
        long Cociente;
        int Cifras;
        long Residuo;
        long Suma = 0;
        long i;
        long BitParidad;
        Scanner escaneo = new Scanner(System.in);
    }
}
```

Figura 1.6 – Declaración de variables.

Desarrollo del programa

```
//Mensaje Información - Título del programa
System.out.print("\tDetector de errores.\n\n");

//Mensaje Información - Ingresar el número binario
System.out.print("Ingrese el número binario: ");
Numero_binario = escaneo.nextLong(); //Leer número binario

Dividendo = Numero_binario; //Asignación del número binario a la variable Dividendo
Cifras = 0; //Inicializar variable Cifras con el valor de 0

//Mientras Dividendo sea diferente de 0
while(Dividendo!=0)
{
    Cociente = Dividendo/10; //Asignación de la división entre Dividendo y 10 a la variable COciente
    Dividendo = Cociente; //Asignación de la variable Cociente a la variable Dividendo
    Cifras++; //Contabilizar el número de cifras mientras se cumpla la condición
}

//Mensaje Información - Número de Cifras
System.out.println("El numero de cifras es: "+Cifras);

Dividendo = Numero_binario; //Asignación del número binario a la variable Dividendo
i = Cifras-1; //Número de cifras totales menos 1, asignado a la variable i (Exponente)

//Mientras i sea mayor o igual que 0
while(i>=0)
{
    Cociente = Dividendo/10; //Asignación de la división entre Dividendo y 10 a la variable COciente
    Residuo = Dividendo%10; //Módulo de dividendo entre 10, asignado a la variable Residuo

    Suma = Suma + Residuo *(long)Math.pow(1,i); //Acumular en Suma el número total de 1's
    Dividendo = Cociente; //Asignación de la variable Cociente a la variable Dividendo
    i--; //Restar el valor de i menos 1, cada vez que se repita el bucle
}
```

Figura 1.7 – Desarrollo del programa.

Determinar existencia de error

```
//Mensaje Información - Número total de 1's
System.out.println("La suma total de 1's es: "+Suma);

BitParidad = Suma%2;          //Módulo de Suma entre 2 (Para verificar si la suma total de 1's es par o impar)

if(BitParidad == 0)           //Si BitParidad == 0 (Suma total de 1's es par)
{
    System.out.println("Sí hay error. Bit de paridad es "+BitParidad);
}
else                           //Sino BitParidad !=0 (Suma total de 1's es impar)
{
    System.out.println("No hay error. Bit de paridad es: "+BitParidad);
}

}
```

Figura 1.8 – Determinar existencia de error, mediante el bit de paridad y una condición If-Else.

- Presionando el botón Paridad Impar

Para el caso en el que se presione el botón de paridad impar, se efectuará el mismo proceso que se realizó con el caso de paridad par. La única diferencia la encontramos en la sección última del programa (Recuadro amarillo), es decir, en la condición If-Else. Tal como se muestra en la figura 1.9.

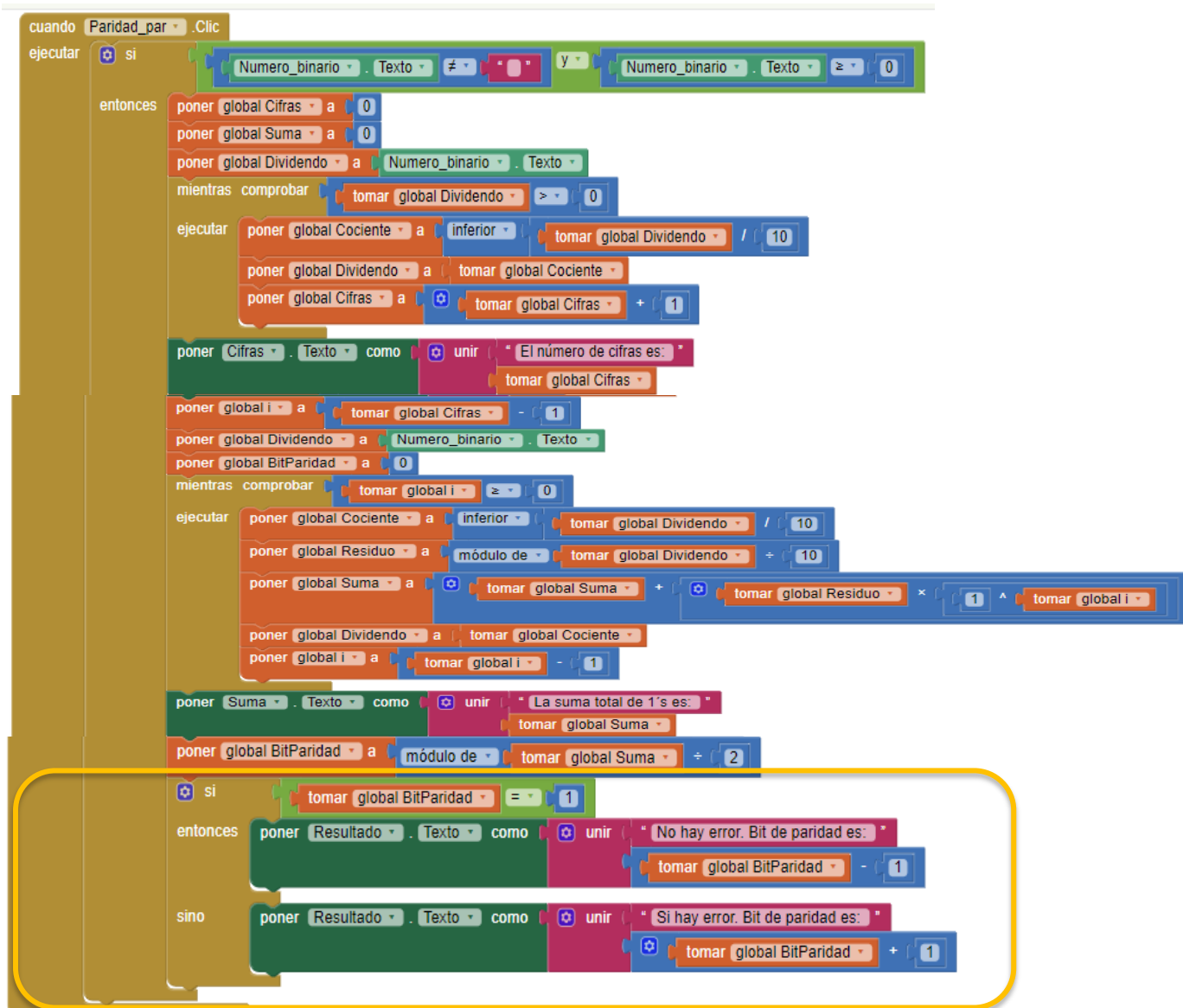


Figura 1.9 – Detección de errores mediante paridad impar.

Como podemos observar, en este caso, si el bit de paridad $\text{BitParidad} == 1$, entonces no existirá error y se mostrará en su salida que el bit de paridad es igual a 0, ya que le restamos el valor de 1. Caso contrario, si existirá error y en su salida se mostrará que el bit de paridad es igual a 1, ya que, en este caso le sumamos el valor de 1.

De igual manera presentamos el código realizado en Java NetBeans (Figura 2.1) para comprender de mejor manera esta apartado.

```

    if(BitParidad == 1)           //Si BitParidad == 1 (Suma total de 1's es impar)
    {
        System.out.println("No hay error. Bit de paridad es "+(BitParidad-1));
    }
    else                           //Sino BitParidad !=1 (Suma total de 1's es par)
    {
        System.out.println("No hay error. Bit de paridad es: "+(BitParidad+1));
    }
}
}

```

Figura 2.1 – Detección de errores mediante paridad impar (Código).

9. DESCRIPCIÓN DE PRERREQUISITOS Y CONFIGURACIÓN

En lo que corresponde a la utilización de aplicaciones secundarias para que el programa funcione correctamente, podemos decir que no se ha necesitado de ninguna que influya directamente sobre su programación. Sin embargo, no está por demás mencionar que se ha utilizado una herramienta de software extra (Java NetBeans) como un programa de apoyo para poder comprender de mejor manera el funcionamiento del programa, además de la aplicación móvil MIT AI2 Companion para poder realizar pruebas e ir compilando el programa a la vez que se iba realizando la programación en MIT APP INVENTOR 2.

Por otro lado, para el correcto funcionamiento del programa, hay que tener en cuenta varios puntos específicos. Los cuales son indispensables para no tener ningún inconveniente al momento de ejecutar el programa.

- El ingreso de datos debe ser única y exclusivamente en número binario. No se aceptan caracteres, ni números negativos.
- Para la descarga e instalación de la aplicación en nuestros dispositivos móviles, debemos descargar el archivo en formato APK y posterior a ello realizar la instalación de la aplicación. Se tendrá que permitir el origen de archivos desconocidos en la sección

de ajustes del dispositivo móvil para su correcta instalación. En nuestro caso la aplicación que hemos elaborado se llama Proyecto1_Derrores.

10. APORTACIONES

Ejemplo de paridad

Tenemos el carácter original 0111001. Vemos que la trama a transmitir tiene un número par de unos (4 unos). Al añadir el bit de paridad obtendremos el siguiente carácter, que es el que se transmitirá a destino:

Si usamos paridad par, ya hay un número par de unos, por tanto, se añade un 0, y transmitiremos 0111001'0'.

Si usamos paridad impar, como hay un número par de unos, añadiremos otro 1 para conseguir un número impar, y transmitiremos 0111001'1'.

Si se envía un dato y durante la transmisión se produce un único error, el destinatario puede detectarlo al comprobar la paridad en el destino. Usando los ejemplos anteriores, y alterando un solo bit de la trama transmitida, nos quedaría.

Paridad par: se recibe 00110001 en vez de 00111001. En la comprobación, al contar el número de unos salen 3 (impar), por lo que se ha producido un error.

Paridad impar: se recibe 10110001 en vez de 10111001. En la comprobación, al contar el número de unos salen 4 (par), por lo que se ha producido un error.

Siguiendo los ejemplos anteriores, y alterando dos bits en la transmisión, veremos como el método de detección de errores falla:

Paridad par: se recibe 00110101 en vez de 00111001. Al comprobar el número de unos salen 4 (par), y no se detectan los errores.

Paridad impar: se recibe 10110101 en vez de 10111001. Al comprobar el número de unos salen 5 (impar), y no se detectan los errores.

Ejemplo de control de errores

Supongamos que dos unidades se comunican con una paridad par, que es la forma de verificación de paridad más común.

Según la unidad de transmisión, va enviando los bytes y primero recuenta el número de bits 1 en cada grupo de siete bits. Si resulta par la cantidad de bits 1, coloca en 0 el bit de paridad. Si es impar la cantidad de bits 1, coloca en 1 el bit de paridad. De esta manera, cada byte tendrá una cantidad par de bits 1.

Por parte del receptor se verifica cada byte para así asegurarse que tenga una cantidad par de bits 1. En caso de encontrar una cantidad impar de bits 1 en el byte, el receptor sabrá que durante la transmisión se produjo un error.

Previamente, tanto el ente receptor como el emisor deben haber acordado el uso de la verificación de paridad y si la paridad debe ser impar o par. Si no están configurados ambos lados con el mismo sentido de paridad será imposible poder comunicarse.

Que es App Inventor

Es un entorno de programación visual e intuitivo que permite a todos, incluso a los niños, crear aplicaciones totalmente funcionales para teléfonos inteligentes y tabletas. Los nuevos en MIT App Inventor pueden tener una primera aplicación simple en funcionamiento en menos de 30 minutos. Y lo que es más, nuestra herramienta basada en bloques facilita la creación de aplicaciones complejas y de alto impacto en mucho menos tiempo que los entornos de programación tradicionales. El proyecto MIT App Inventor busca democratizar el desarrollo de software al empoderar a todas las personas, especialmente a los jóvenes, para pasar del consumo de tecnología a la creación de tecnología.

Un pequeño equipo de personal y estudiantes de CSAIL, dirigido por el profesor Hal Abelson, forma el núcleo de un movimiento internacional de inventores. Además de liderar el alcance educativo sobre MIT App Inventor y realizar investigaciones sobre sus impactos, este equipo

central mantiene el entorno gratuito de desarrollo de aplicaciones en línea que sirve a más de 6 millones de usuarios registrados.

Ejemplo de uso de la aplicación

- Paridad Par

Detector de errores
Ingrese el número binario
11101

Seleccione una opción para detectar el error

Paridad Par

Paridad Impar

El número de cifras es: 5

La suma total de 1's es: 4

No hay error. Bit de paridad es: 0

Figura 2.1 – Ejemplo Parida Par.

- Paridad Impar

Detector de errores
Ingrese el número binario
11101

Seleccione una opción para detectar el error

Paridad Par

Paridad Impar

El número de cifras es: 5

La suma total de 1's es: 4

Si hay error. Bit de paridad es: 1

Figura 2.1 – Ejemplo Parida Impar.

11. CONCLUSIONES

- El diseño e implementación del programa, se lo ha realizado de manera que cumpla el objetivo de realizar la detección de error de un número binario, mediante paridad par e impar. Todo este proceso se llevó a cabo con la programación en un entorno gráfico que nos ofrece MIT APP INVENTOR 2, el cual nos permitió realizar la aplicación para que cualquier usuario, pueda utilizarlo desde su dispositivo móvil. Tal como se ha podido observar en las diferentes figuras del presente informe, el diseño de la app es muy intuitiva y amigable al usuario.
- La detección de errores por paridad par o impar es muy esencial para sistemas de comunicación digital, los cuales necesitan de valores precisos y sin equivocaciones. La manera de entender cómo funciona cada paridad es muy sencilla. En el caso de paridad par, tendríamos un bit de paridad de 0 si la suma total de 1's es par, es decir, no existiría error. Caso contrario el bit de paridad tendría el valor de 1 y por lo tanto existiría error. Por otro lado, en el caso de paridad impar, tendríamos un bit de paridad de 0 si la suma total de 1's es impar. Caso contrario el bit de paridad tendría el valor de 1 si la suma total de 1's es par. Cada uno de los conceptos de los métodos de detección de error mencionados, fueron muy importantes y necesarios para la realización del algoritmo, el cual fue implementado en App Inventor.
- La realización del programa se la realizó en base a todo lo investigado. Prácticamente necesitamos realizar el conteo de 1's, y mediante el método de paridad que se escoja, se determinará si existe error dependiendo si la suma mencionada es par o impar. De esta manera logramos diseñar un algoritmo, el cual se basó principalmente en divisiones sucesivas para realizar el conteo de cifras, la suma total de 1's y por último encontrar el bit d paridad que determinó el error. Todo esto, mediante el uso de la estructura repetitiva While. Cabe mencionar que todo el algoritmo se lo ha hecho de manera correcta, incluso utilizamos un software de apoyo (Java NetBeans) para realizar las

diferentes comprobaciones y pruebas que el programa requería. Y al momento de realizar la investigación con los artículos de los autores investigados, podemos darnos cuenta de la gran similitud y relación que encontramos con respecto a nuestro trabajo de investigación, todo esto se lo ha detallado en sección de Estado del arte.

12. RECOMENDACIONES

- Se recomienda realizar la investigación única y exclusivamente de los temas a ser tratados durante la ejecución y realización del informe que a su vez sustenten los fundamentos para la elaboración del algoritmo y su programación.
- Implementar y declarar correctamente las variables necesarias del programa, de tal manera que permitan realizar el código de programación sin que exista ningún tipo de error al momento de ejecutar y compilar el programa.
- Si no existe familiarización con la programación mediante interfaz gráfica que ofrece MIT APP INVENTOR 2, recomendamos realizar el algoritmo de programación en una interfaz de código (de manera de apoyo) para posteriormente plasmarlo en la programación mediante bloques, tal como nosotros hemos realizado y de esta manera también comprobar que la realización del programa se encuentra bien elaborada.
- Uno de los inconvenientes del software MIT APP INVENTOR es que no se pueden declarar variables de cualquier tipo (entero, flotante, long, etc). Razón por la cual recomendamos utilizar el bloque “Redondear”, el cual nos permite, valga la redundancia, redondear un número flotante al entero más cercano. En nuestro caso hemos utilizado el bloque “Inferior” el cual redondea un número flotante al entero inferior.

13. CRONOGRAMA

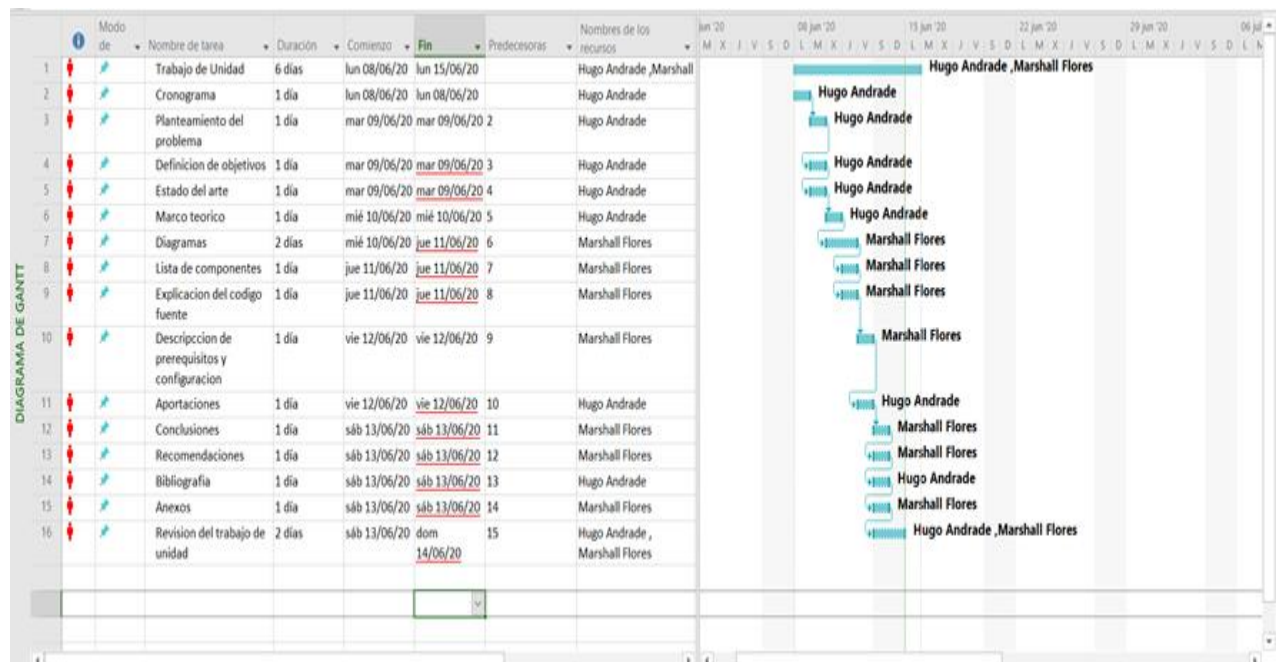


Diagrama 1.4 – Diagrama de Gantt.

14. BIBLIOGRAFÍA

[1] F. K Law, M. Rakib Uddn, Nur Musyirah Masir1, Ynog Hyub Won, obtenido de:

<https://sci-hub.tw/https://ieeexplore.ieee.org/document/8527192>

[2] Peter Farkaš, Tomáš Janvars y Katarína Farkašová, Eugen Ružický, obtenido de :

<https://sci-hub.tw/https://ieeexplore.ieee.org/document/8337547>

[3] Silvia Anggraeni, Fawnizu Azmadi Hussin and Varun Jeoti obtenido de:

<https://sci-hub.tw/https://ieeexplore.ieee.org/document/6869526>

(Perez, Mandado, 2016) - libro Sistemas Electrónicos Digitales obtenido de:

<https://books.google.com.ec/books?id=V7JpKkZaEYMC&pg=PA18&lpg=PA18&dq=codigos+de+paridad&source=bl&ots=sTHg4rJA6R&sig=ACfU3U3FsgLxKEJBQNhOunOHHGr4NbZgNg&hl=es&sa=X&ved=2ahUKEwjzprfus4TqAhXNQjABHTmsA98Q6AEwEXoECBEQAAQ#v=onepage&q=codigos%20de%20paridad&f=false>

(Fernandez, Gil, 2015) - Una introducción a los códigos detectores de error y correctores de errores obtenido de:

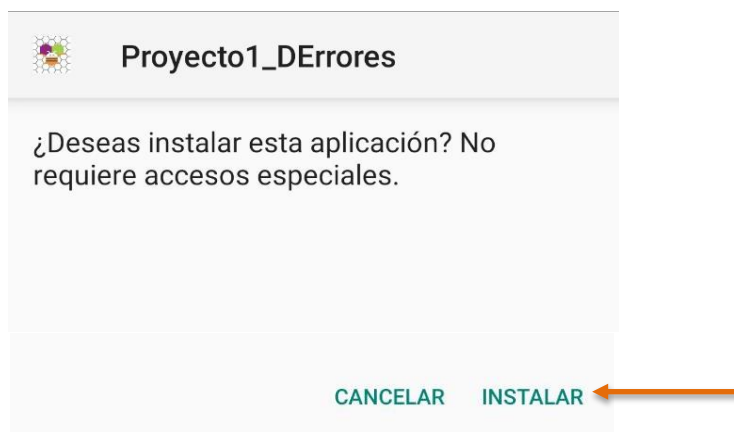
<https://www.matematicaparatodos.com/varios/Codigos.pdf>

15. ANEXOS

15.1. MANUAL DE USUARIO

El presente manual está diseñado para facilitar el uso de la aplicación que nos permite detectar el error un número binario de cualquier longitud, mediante paridad par o impar.

Lo primero que debe realizar es proceder a descargar el archivo APK de la aplicación creada en MIT APP INVENTOR 2. Le aparecerá una ventana en la que tendrá que presionar en el botón Instalar, tal como se muestra a continuación.



Posterior a ello se abrirá que otra ventana que va a requerir de la aceptación de los permisos en la sección de ajustes, donde debe permitir orígenes de archivos desconocidos, luego de realizar ese requerimiento se abrirá otra ventana en la que tendrá que presionar el botón Instalar de todas formas.

Play Protect bloqueó la instalación



Proyecto1_DErrores

Play Protect no reconoce al desarrollador de esta app.
Algunas apps de desarrolladores desconocidos podrían no ser seguras.

ACEPTAR

INSTALAR DE TODAS FORMAS



Realizando todo lo mencionado, usted ya habrá instalado la aplicación. Por último, se abrirá otra ventana, en la que le da la opción de abrir la aplicación.



Al abrir la aplicación se encontrará con la interfaz principal del programa, tal como se muestra en la siguiente ilustración.

Detector de errores

Ingrese el número binario

Seleccione una opción para detectar el error

Paridad Par

Paridad Impar


Para utilizar la aplicación, siga las siguientes instrucciones:

1. Ingreso de datos

En esta pantalla el usuario debe digitar el número binario en la casilla que se encuentra señalada a continuación

Detector de errores


Ingrese el número binario




2. Escoja una opción

Presione el botón Paridad Par para detectar el error del número binario ingresado, mediante el método de paridad par. O Presione el botón Paridad Impar para detectar el error mediante el método de paridad impar.

Seleccione una opción para detectar el error





Paridad Par



Paridad Impar

3. Existencia de error

En los tres recuadros siguientes le aparecerá el número de cifras del número ingresado, la suma total de 1's y la existencia de error con su bit de paridad.

Número de cifras		<input data-bbox="874 353 1375 405" type="text"/>
Suma total de 1's		<input data-bbox="874 423 1375 474" type="text"/>
Existencia de error y bit de paridad		<input data-bbox="874 492 1375 544" type="text"/>