

## DESIGN OF AN EFFICIENT AND LESS MEMORY REQUIRED ALGORITHMIC PROCEDURE FOR COMPUTING GRAY CODES

Afaq Ahmad and Mohammed M. Bait Suwailam

Department of Electrical and Computer Engineering  
College of Engineering, Sultan Qaboos University  
P. O. Box 33, Postal Code 123; Muscat, Sultanate of Oman  
Telephones: (968) 2414 1327, (968) 2414 2191; Fax. (968) 2441 3416  
E-mails: [afaq@squ.edu.om](mailto:afaq@squ.edu.om); [msuwailam@squ.edu.om](mailto:msuwailam@squ.edu.om)

### ABSTRACT:

Through this paper we have presented the derivation, design and implementation of a newly developed algorithm for the generation of an n-bit binary reflected Gray code sequences. The developed algorithm is stemmed from the fact of generating and properly placing the min-terms from the universal set of all the possible min-terms  $[m_0 m_1 m_2 \dots m_N]$  of Boolean function of n variables, where,  $0 < N \leq 2^n - 1$ . The resulting algorithm is in concise form and trivial to implement. Furthermore, the developed algorithm is equipped with added attributes of optimizing of time and space while executed.

### KEYWORDS:

Gray code, Min-terms, Boolean function, Algorithm, Processing time, Memory space, Binary

## 1. INTRODUCTION

Although many codes for example Binary Coded Decimal (BCD), Excess-3 code, Hamming code, Cyclic Redundancy Code (CRC), check sum and many others exist and are in use. But the Gray codes which are named for Frank Gray who patented the use of them in shaft encoders in 1953 [1] due to its attribute of single distance only, which avoids ambiguous switching situations, is particularly used to handle safely and conveniently the control problems. Unlimited applications are accounted for the Gray code. Some of them are mentioned here as follows. However, more can be learned through the references provided as in the references [2] – [8].

Mechanical position sensors use Gray code to convert the angular position (angle-measuring devices) of a shaft to digital form. Gray codes were used in telegraphy. The Gray code also forms a Hamiltonian cycle on a hypercube, where each bit is seen as one dimension. In data transmission, Gray codes play an important role in error detection and correction. Solving puzzles such as the Tower of Hanoi and the Brain, the study of bell-ringing, analog-digital signal conversion, classifying of Venn diagrams, continuous space-filling curves, enhancing the

resolution of spectrometer for a satellite application, labeling the axes of Karnaugh maps are the processes where Gray codes are used due to its uniqueness. Gray codes are also beneficial in Genetic Algorithms due to its incremental change property. Using Gray codes for addressing the memory results in saving of the power because a few address lines change as the program counter advances to the next location. Also, Gray codes are extensively used by digital system designers for passing multi-bit count information between synchronous logic that operates at different clock frequencies. In some numerical problems, Gray codes can be useful in situations of looping over many values of a bit. Furthermore, due to its attribute Gray code could be a good choice for the search of the optimal test-sequences in digital system testing [2] – [8].

This paper presents a new concept of generating the Gray code of n-bit size. We designed an efficient algorithm to write the codes which reduces the processing time and memory space requirements.

## 2. BINARY TO GRAY CODE CONVERSION – AN ANALYSIS

No doubt, a simple recursive equation in mod (2) operation can convert a simple binary -...-8-4-2-1 codes to the Gray one. The hardware is also simple which is based on the bank of Exclusive-OR gates. To make the content of this paper more readable to audience the description of the generation procedure of the Gray code is given below:

To convert a binary number  $[b_{n-1} b_{n-2} \dots b_1 b_0]$  to its corresponding Gray code  $[g_{n-1} g_{n-2} \dots g_1 g_0]$ , start at the left with the bit  $b_{n-1}$  (the  $n^{\text{th}}$ , most significant bit) and use the following recursive equations.

$$g_{n-1} = b_{n-1} \quad (1)$$

$$g_{n-2} = b_{n-1} \oplus b_{n-2} \quad (2)$$

Or, in general for  $i^{th}$  bit where  $i$  varies as,  $2 \leq i \leq n$ , the following Equation (3) can be used.

$$g_{n-i} = b_{n-i+1} \oplus b_{n-i} \tag{3}$$

The above Equations (1) and (3) are illustrated for a 3-bit Gray code encoder in an example below.

**Example 1:**

Let a binary code  $[b_2 b_1 b_0] = [1 0 0]$ .

So,  $g_2 = b_2 = 1$ ;  $g_1 = b_2 \oplus b_1 = (1 + 0) \bmod 2 = 1$ ;  $g_0 = b_1 \oplus b_0 = (0 + 0) \bmod 2 = 0$ . Therefore,  $[b_2 b_1 b_0] = [1 0 0]$  when encoded in Gray code gives  $\rightarrow [g_2 g_1 g_0] = [1 1 0]$ .

In hardware binary to Gray encoder can be implemented using a combinational circuits based on exclusive-OR operations. Figure 1 depicts the implementation of respective circuit.

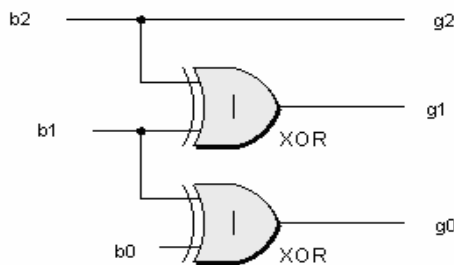


Figure 1: Logic circuit to convert binary to Gray codes of Example 1

For listing the all possible Gray code of bit size 3 (Example 1), the following Equations ( 4) – ( 6), which are derived from the truth table of 3-bit binary to Gray code encoder can be used.

$$g_2 = f_{(b_2, b_1, b_0)} = \Sigma(4, 5, 6, 7) = b_2 \tag{4}$$

$$g_1 = f_{(b_2, b_1, b_0)} = \Sigma(2, 3, 4, 5) = b_2 \oplus b_1 \tag{5}$$

$$g_0 = f_{(b_2, b_1, b_0)} = \Sigma(1, 2, 4, 6) = b_1 \oplus b_0 \tag{6}$$

Since the Gray code encoder needs first to obtain the binary data (through a binary counter) before generating the Gray code. This process requires 2-stages as shown in Figure 2. Thus it is imperative to derive a mechanism to avoid this situation which needs much time and space to implement. The ensuing section is a consequence to it.

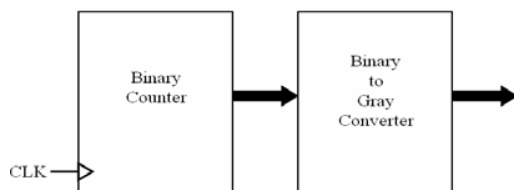


Figure 2: Block model of binary to Gray code conversion

### 3. COMPUTING GRAY CODES – A DERIVATION

If we look to the Table 1 and 2 which lists the Gray codes, respective min-terms and equivalent decimals for  $n = 1$  to 6 forced us to derive the following conclusions.

1. Gray code of size  $n$  has a specific pattern relationship between min-terms of the Gray code of its predecessor code of size  $n-1$ .
2. The Gray code of size  $n$  can be directly scripted using the  $n$ -bit K-map where the min-terms cells are to be read clock-wise and down to the row as explained in the Figure 3.

The example of Figure 3 is a 4-variable K-map used to generate Gray code which reads as  $[m_0 m_1 m_3 m_2 m_6 m_7 m_5 m_4 m_{12} m_{13} m_{15} m_{14} m_{10} m_{11} m_9 m_8]$ . The derivation 2 is not feasible since generating the K-map for higher variables are difficult to manipulate. Therefore, the derivation 1 is the point of our work.

Table 1: Decimal, Gray code, and (min-terms) For,  $n = 1$  to 5

Decimal	5-bit	4-bit	3-bit	2-bit	1-bit
0	00000 ( $m_0$ )	0000 ( $m_0$ )	000 ( $m_0$ )	00 ( $m_0$ )	00 ( $m_0$ )
1	00001 ( $m_1$ )	0001 ( $m_1$ )	001 ( $m_1$ )	01 ( $m_1$ )	01 ( $m_1$ )
3	00011 ( $m_3$ )	0011 ( $m_3$ )	011 ( $m_3$ )	11 ( $m_3$ )	
2	00010 ( $m_2$ )	0010 ( $m_2$ )	010 ( $m_2$ )	10 ( $m_2$ )	
6	00110 ( $m_6$ )	0110 ( $m_6$ )	110 ( $m_6$ )		
7	00111 ( $m_7$ )	0111 ( $m_7$ )	111 ( $m_7$ )		
5	00101 ( $m_5$ )	0101 ( $m_5$ )	101 ( $m_5$ )		
4	00100 ( $m_4$ )	0100 ( $m_4$ )	100 ( $m_4$ )		
12	01100 ( $m_{12}$ )	1100 ( $m_{12}$ )			
13	01101 ( $m_{13}$ )	1101 ( $m_{13}$ )			
15	01111 ( $m_{15}$ )	1111 ( $m_{15}$ )			
14	01110 ( $m_{14}$ )	1110 ( $m_{14}$ )			
10	01010 ( $m_{10}$ )	1010 ( $m_{10}$ )			
11	01011 ( $m_{11}$ )	1011 ( $m_{11}$ )			
9	01001 ( $m_9$ )	1001 ( $m_9$ )			
8	01000 ( $m_8$ )	1000 ( $m_8$ )			
24	11000 ( $m_{24}$ )				
25	11001 ( $m_{25}$ )				
27	11011 ( $m_{27}$ )				
26	11010 ( $m_{26}$ )				
30	11110 ( $m_{30}$ )				
31	11111 ( $m_{31}$ )				
29	11101 ( $m_{29}$ )				
28	11100 ( $m_{28}$ )				
20	10100 ( $m_{20}$ )				
21	10101 ( $m_{21}$ )				
23	10111 ( $m_{23}$ )				
22	10110 ( $m_{22}$ )				
18	10010 ( $m_{18}$ )				
19	10011 ( $m_{19}$ )				
17	10001 ( $m_{17}$ )				
16	10000 ( $m_{16}$ )				

Table 2: Decimal, Gray code, and (min-terms)  
For, n = 6

Decimal	Gray code (m)	Decimal	Gray code (m)
0	000000 (m <sub>0</sub> )	48	110000 (m <sub>48</sub> )
1	000001 (m <sub>1</sub> )	49	110001 (m <sub>49</sub> )
3	000011 (m <sub>3</sub> )	51	110011 (m <sub>51</sub> )
2	000010 (m <sub>2</sub> )	50	110010 (m <sub>50</sub> )
6	000110 (m <sub>6</sub> )	54	110110 (m <sub>54</sub> )
7	000111 (m <sub>7</sub> )	55	110111 (m <sub>55</sub> )
5	000101 (m <sub>5</sub> )	53	110101 (m <sub>53</sub> )
4	000100 (m <sub>4</sub> )	52	110100 (m <sub>52</sub> )
12	001100 (m <sub>12</sub> )	60	111100 (m <sub>60</sub> )
13	001101 (m <sub>13</sub> )	61	111101 (m <sub>61</sub> )
15	001111 (m <sub>15</sub> )	63	111111 (m <sub>63</sub> )
14	001110 (m <sub>14</sub> )	62	111110 (m <sub>62</sub> )
10	001010 (m <sub>10</sub> )	58	111010 (m <sub>58</sub> )
11	001011 (m <sub>11</sub> )	59	111011 (m <sub>59</sub> )
9	001001 (m <sub>9</sub> )	57	111001 (m <sub>57</sub> )
8	001000 (m <sub>8</sub> )	56	111000 (m <sub>56</sub> )
24	011000 (m <sub>24</sub> )	40	101000 (m <sub>40</sub> )
25	011001 (m <sub>25</sub> )	41	101001 (m <sub>41</sub> )
27	011011 (m <sub>27</sub> )	43	101011 (m <sub>43</sub> )
26	011010 (m <sub>26</sub> )	42	101010 (m <sub>42</sub> )
30	011110 (m <sub>30</sub> )	46	101110 (m <sub>46</sub> )
31	011111 (m <sub>31</sub> )	47	101111 (m <sub>47</sub> )
29	011101 (m <sub>29</sub> )	45	101101 (m <sub>45</sub> )
28	011100 (m <sub>28</sub> )	44	101100 (m <sub>44</sub> )
20	010100 (m <sub>20</sub> )	36	100100 (m <sub>36</sub> )
21	010101 (m <sub>21</sub> )	37	100101 (m <sub>37</sub> )
23	010111 (m <sub>23</sub> )	39	100111 (m <sub>39</sub> )
22	010110 (m <sub>22</sub> )	38	100110 (m <sub>38</sub> )
18	010010 (m <sub>18</sub> )	34	100010 (m <sub>34</sub> )
19	010011 (m <sub>19</sub> )	35	100011 (m <sub>35</sub> )
17	010001 (m <sub>17</sub> )	33	100001 (m <sub>33</sub> )
16	010000 (m <sub>16</sub> )	32	100000 (m <sub>32</sub> )

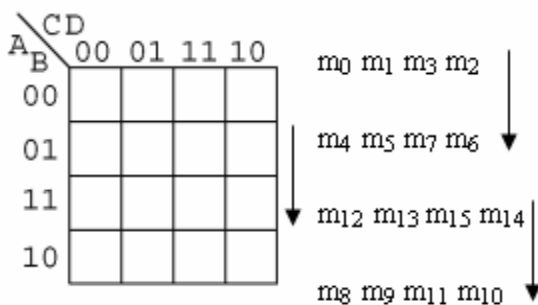


Figure 3: A 4-variable K-map

#### 4. GRAY CODE CONVERSION – ALGORITHM DESIGN

The following Equations (7) to (11) describe the Gray codes for bit size 1 to 5 respectively. A specific relation

between the min-terms patterns is visible and summarized in the form of a theorem below.

$$G(1) = [m_0 \mid m_1] \quad (7)$$

$$G(2) = [m_0 \ m_1 \mid m_3 \ m_2] \quad (8)$$

$$G(3) = [m_0 \ m_1 \ m_3 \ m_2 \mid m_6 \ m_7 \ m_5 \ m_4] \quad (9)$$

$$G(4) = [m_0 \ m_1 \ m_3 \ m_2 \ m_6 \ m_7 \ m_5 \ m_4 \mid m_{12} \ m_{13} \ m_{15} \ m_{14} \ m_{10} \ m_{11} \ m_9 \ m_8] \quad (10)$$

$$G(5) = [m_0 \ m_1 \ m_3 \ m_2 \ m_6 \ m_7 \ m_5 \ m_4 \ m_{12} \ m_{13} \ m_{15} \ m_{14} \ m_{10} \ m_{11} \ m_9 \ m_8 \mid m_{24} \ m_{25} \ m_{27} \ m_{26} \ m_{30} \ m_{31} \ m_{29} \ m_{28} \ m_{20} \ m_{21} \ m_{23} \ m_{22} \ m_{18} \ m_{19} \ m_{17} \ m_{16}] \quad (11)$$

#### Theorem 1

G (n) is a matrix of order  $2^n \times n$  in binary format of n-bit forming  $2^n$  min-terms of n variables. By analyzing the patterns we reach to the conclusion that G(n) can be obtained first by writing the min-terms of G(n-1) then appending the min-terms by advancing each of the min-terms starting from the last to the first by a value of  $2^{n-1}$ .

#### Proof:

Proof is as visible through the Equations from (7) to (11).

An algorithm is designed on the basis of the study of Theorem 1 and is as given below.

#### Algorithm

##### STEP 0:

START by inputting the bit size (n) of the Gray code to be generated;

##### STEP 1:

Initialize a vector  $V = [0 \ 1]$ ;

##### STEP 2:

Count a loop for  $k = 1$  to  $n$ ;

##### STEP 3:

Check that  $k = n$  or nor, if YES GO TO **STEP 8** ELSE, **STEP 4**;

##### STEP 4:

Increment the counter k by 1 i.e.  $k = k + 1$ ;

##### STEP 5:

For  $i = 0$  to  $2^{n-1} - 1$ ;  
 $V_i = [V(2^{n-1} - i) + 2^{n-1}]$ ;

##### STEP 6:

Modify vector V as  $V = [V, V_i]$ ;

##### STEP 7:

GO TO **STEP 3**;

##### STEP 8:

Transpose V i.e.  $V^T$ ;

##### STEP 9:

Convert  $V^T$  into binary format i.e.  $(V^T)_{10} \rightarrow (V^T)_2 =$  Gray code of n-bit size i.e. a matrix G (n) of size  $(N+1)$  by  $n$ ;

##### STEP 10:

Output G(n);

##### STEP 11:

STOP

**A DEBUG EXAMPLE:**

**Example 2:**

A debug test of the above algorithm is carried out for n = 3 as presented below in Table 3.

Table 3: A debug test of algorithm for, n = 3

i	V <sub>i</sub>	V	(V) <sub>base 2</sub>
-		[0 1]	-
0 to 1	[3 2]	[0 1 3 2]	-
0 to 3	[6 7 5 4]	[0 1 2 3 6 7 5 4]	-
			000
			001
			011
			010
			110
			111
			101
			100

**5. IMPLEMENTATION OF ALGORITHM**

The above designed algorithm is implemented using the MATLAB codes. The out put of the m-file with name “gray\_generator\_proposed” can be visualized as shown in Figure 4. A sample output of the program shown in the figure is only for n = 3. This is provided just to make it more readable, otherwise the output for the higher values of n will require much space to present. Similarly, we also encoded the conventional method of Gray code conversion into MATLAB script. To compare the efficiency amongst these two approaches we run both of the programs for n = 2 to 10 and some of the results are tabulated and made available through Table 4 in the ensuing section below.

```
>>gray_generator_proposed
Please Enter a valid positive integer (> 1) : 3
Gray output(s)
=====
0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

Elapsed time is 0.000000 seconds.
>>
```

Figure 4: MATLAB command window

**6. RESULTS**

The processing time and memory space required to implement both of the programs are given as in the table below (Table 4). The bench marks for testing these codes were MATLAB 7.0 on a P 4 CPU 1.5 GHz, 512 MB of RAM.

The results for a subset of the study, for n = 2 to 10 where the memory requirement comparison of the conventional approach of Gray code converter and the proposed approach is presented in Figure 5. Further, to judge the efficiency of the proposed algorithm relative error plot is shown in Figure 6.

Table 4: Comparison between conventional Gray code converter and the proposed method

Bit size	Comparison Criterion	Conventional Gray code converter	Proposed Method
5	Processing Time (seconds)	0.016	0.00
	Memory Resources (size in Bytes)	4120	3248
7	Processing Time (seconds)	0.016	0.00
	Memory Resources (size in Bytes)	22552	16560
10	Processing Time (seconds)	0.094	0.016
	Memory Resources (size in Bytes)	245784	180400

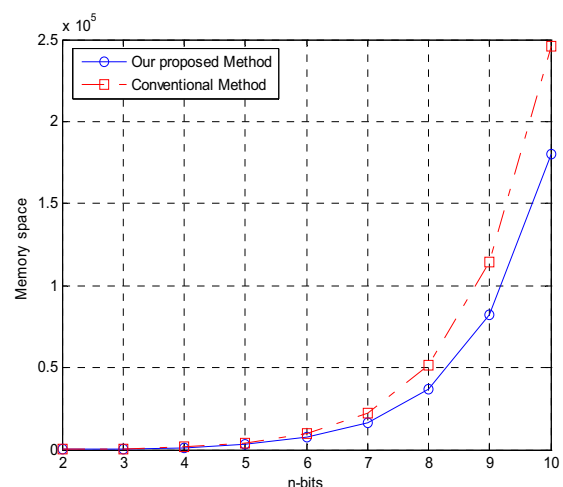


Figure 5: Memory requirement comparison

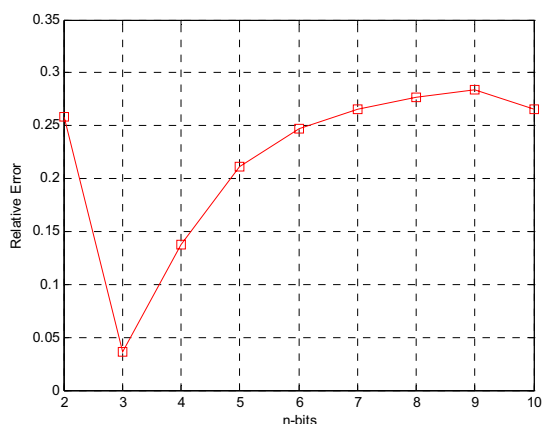


Figure 6: Relative error judgment

## 7. CONCLUSIONS

A time space optimal algorithmic procedure to generate Gray code-words of any bit length  $n$  is presented through this paper. The comparative study reveals that the proposed approach is not only faster but also, requires about 25% less memory space on average while compared with the conventional method of Gray code-word generation technique.

## 8. REFERENCES

- [1] Gray, F. 'Pulse Code Communication'. United States Patent Number 2,632,058, March 17, 1953.
- [2] Gilbert, E.N. 'Gray Codes and Paths on the  $n$ -Cube'. Bell System Tech. Journal, 37 (1958), pp. 815-826.
- [3] F. Ruskey, 'A Survey of Venn Diagrams', Elec. J. of Comb. (1997).
- [4] Goddyn, L., Gvozdzak, P. 'Binary gray codes with long bit runs', Electron. Journal Combin 10 (2003), pp. 1-10.
- [5] Moshe Schwartz and Tuvi Etzion, 'The Structure of Single-Track Gray Codes', IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 45, NO. 7, NOVEMBER 1999, pp. 2383 - 2396
- [6] <http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/10-gates/15-graycode/dual2gray.html> (December, 2006)
- [7] <http://computingdictionary.thefreedictionary.com/gray+code> (November, 2006)
- [8] <http://www.theory.csc.uvic.ca/~cos/inf/comb/SubsetInfo.html>. (November, 2006)