About Some Magic Numbers in the Standard Model of Particle Physics

Arkady L. Shifrin

Abstract

In this article I show that by applying inversions to a set of bit-strings generated in a naturally arising way, I obtain subsets of bit-strings which exactly mimic classes of confirmed particles in the Standard Model. I show that questions regarding the number of particle families, number of particles in each class, and such become trivial in this model.

1. Introduction

The Standard Model (SM) of particle physics describes all known subatomic particles, as well as electromagnetic, weak, and strong interactions among them. Its success in explaining and predicting experimental results, as well as its mathematical consistency and beauty, leaves no room for doubt of its validity.

It is also known that the SM does not explain all physical phenomena. General relativity, dark energy, and dark matter are among the most well-known phenomena that it does not yet incorporate. The SM also provides no strong explanation about its numerical facts. Not only do the values of its constants have to be determined experimentally, but also the number of subatomic particles and number of particles in the families they are classified into remain unexplained.

The quest for answers has taken many paths. Personally, I found these two ideas the most inspiring: “Digital Philosophy” of Prof. E. Fredkin, and “Mathematical Universe Theory” of Prof. M. Tegmark.

Leibniz said almost 350 years ago [in regards to a binary system he invented] “Omnibus ex nihil ducendis sufficit unum!” (“One suffices to create Everything of nothing!”). In this work, I wanted to explore this idea: to build a model starting with a bit and a set of “natural” operations such as concatenation, inversions, and Boolean operations. It turns out the result is a set of objects which exactly recreate the classification of particles of the SM.

I do not claim that the proposed model is equivalent in all respects to the SM. Rather, I believe that it may shed light on some of its fundamental characteristics.

2. Bit-machines

Let us begin with the two simplest possible machines: the first one outputs all zeroes (M0), the second all ones (M1). Stuck together, they produce an M01 machine that outputs the sequence "01010101..."

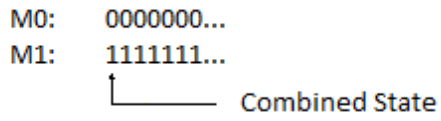

Figure 2.1 *M01 machine as a composition of M0 and M1 machines. The output is read column by column.*

Combining two M01 machines produces an M0011 machine that generates the sequence "001100110011..."

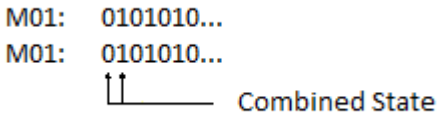

Figure 2.2 *M0011 machine as a composition of two M01 machines. The output is read column by column.*

Combining two M0011 machines with their outputs shifted by one bit results in a machine with 4 states which I call the S4 machine:

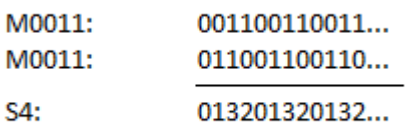

Figure 2.3 *S4 machine as a composition of two M0011 machines. The output is read column by column and converted to decimal.*

In fact, the last step – combining two M0011 machines and converting the resultant output to decimal - can be skipped. The same sequence is produced by reading the sequence "0011001100110..." using a two-bit window and shifting it by one bit to the right after each read:

S4: 00,01,11,10,00… or 01320... **(2.1)**

In a similar way, I build a machine with 8 states (S8). First, I build an M0001 machine by combining M0 with M01, and an M0111 machine by combining M01 with M1:

Figure 2.4 *M0001 and M0111 machines.*

Next, I stick together the M0001 and M0111 machines to produce the M00010111:

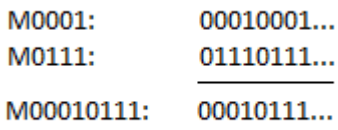

Figure 2.5 *M00010111 machine.*

Finally, combining three M00010111 machines with their outputs shifted by one bit produces the S8 machine:

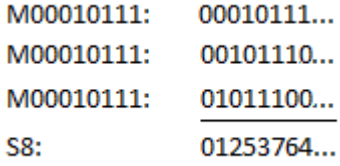

Figure 2.6 *S8 machine as a composition of three M00010111 machines. The output is read column by column, and converted to decimal.*

Again, I can skip the last step, and read the sequence "00010111000..." using a three-bit window, shifting the window to the right one bit at a time:

S8: 000,001,010,101,011,111,110,100,000,… or 012537640... (2.2)

Looking at the outputs of S4 and S8, I notice that the numerical values of the states obtained follow a peculiar order. In fact, each next number (N) is obtained by multiplying the previous one (P) by two, and adding either zero or one:

$$N = (2P + C)\ mod\ 2^k \qquad \textbf{(2.3)}$$

where C=0 or 1 and 2^k is the number of states desired in the model.

3. The Model

I chose k=4, or 16 states, for my model. I used SQL to generate the sequences and analyze them. The code and lists of complete results are presented in the Appendices. Here I describe the main steps.

3.1. Sequence Generator

Let us introduce a 6-bit string and call it a mask. It can have any 6-bit value from 000000 to 111111. The mask controls the transition from P to N; specifically it determines whether we add zero or one to 2*P in order to get N:

$$N = (2P + (Mask\ (P \bmod 8)\ xor\ (P \backslash 8))) \bmod 16 \qquad (3.1)$$

The first bit of a mask controls the transition for P mod 8 = 1, the second for P mod 8 = 2, the last for P mod 8 = 6. Let us also define that

$$0 \rightarrow 1,\ 8 \rightarrow 0,\ 7 \rightarrow 15,\ 15 \rightarrow 14 \qquad (3.2)$$

These conditions prevent having sequences containing just 0's or 15's, and sequences with repeated 0's and 15's (including infinite).

For a given mask, the algorithm starts with 0 and generates the next number until it gets back to where it started from (zero). It then picks the lowest unvisited number, and builds the next sequence in the same fashion. It continues until all 16 states have been visited. Sequences for all 64 possible masks are presented in Appendix 1. Here are a few examples of the sequences:

Mask	Sequences
000000	0,1,2,4,8; 3,6,12,9; 5,10; 7,15,14,13,11;
001000	0,1,2,4,8; 3,7,15,14,13,11,6,12,9; 5,10;
010000	0,1,2,5,10,4,8; 3,6,12,9; 7,15,14,13,11;
111110	0,1,3,7,15,14,13,10,4,9,2,5,11,6,12,8;

Figure 3.2 ***Examples of masks and resulting sequences.***

Some masks produce sequences that cycle through all 16 states; others produce 2, 3, or 4 shorter sequences. Some sequences could be generated by more than one mask (i.e. "0,1,2,4,8"). In general, there is a many-to-many relationship between masks and patterns.

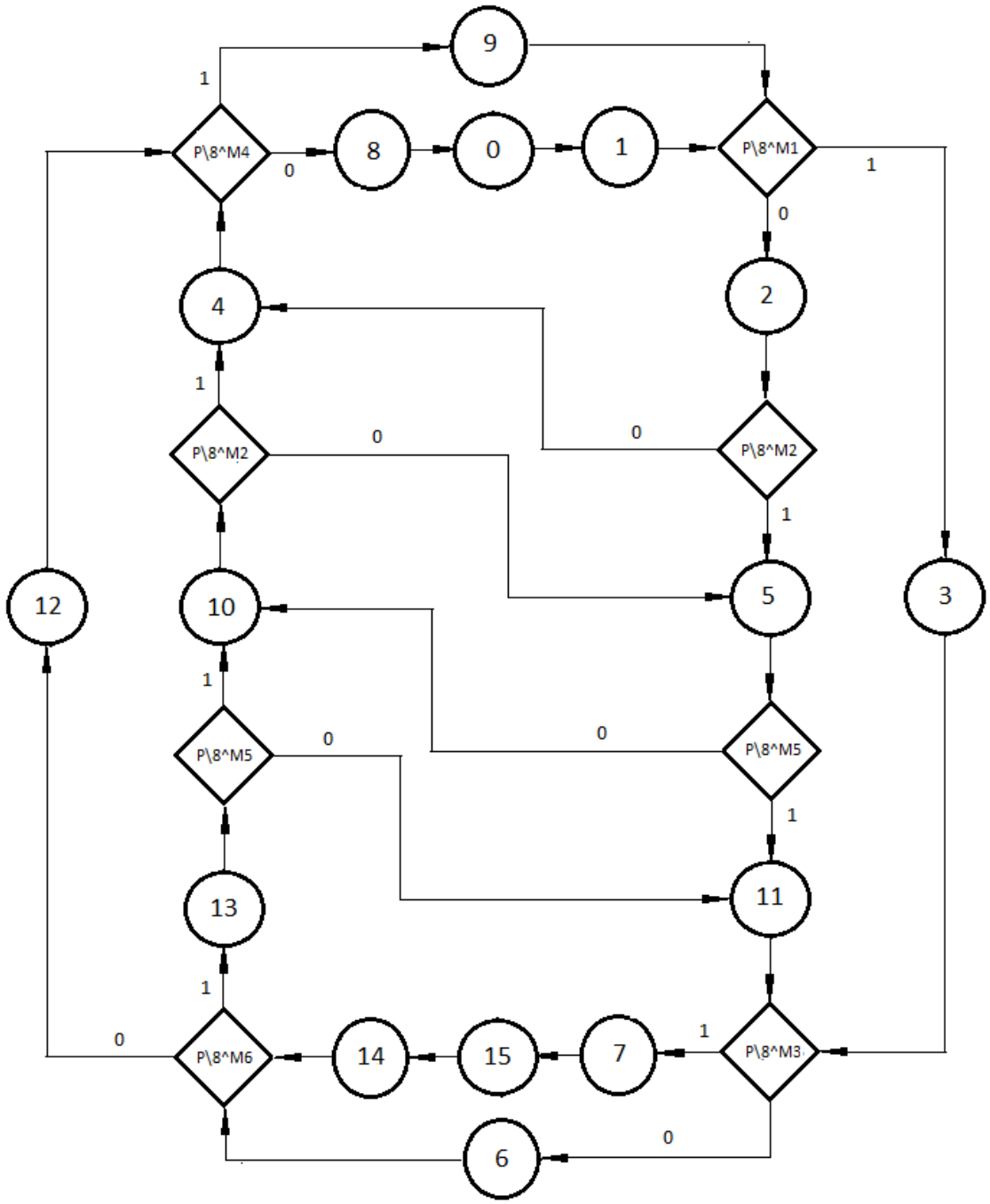

Figure 3.1 ***A Diagram of S16 Machine. Here, circles are states; diamonds hold bits of a mask: M1-M6.***

3.2. Analysis

First, I extract unique sequences. All 61 of these, along with their binary presentations, are shown in Appendix 2. I will be working with binary strings (or bit-strings) for now.

Second, I introduce a few operations:

- Rotational Equivalence (R): since we can start generating a sequence from any of its elements, we can rotate its bit-string. We consider two bit-strings equivalent if we can get one from another by rotating any of them, i.e. R(010) = 001

- Swap (S): this operations reverses a bit-string, i.e. S(001101) = 101100

- Binary Compliment (C): this operation replaces zeroes with ones and ones with zeroes, i.e. C(111001) = 000110.

It is easy to see that for some bit-strings, some or all of the following identities may hold:

$$\mathbf{S(b) = R(b)} \qquad \mathbf{(3.2)}$$
$$\mathbf{C(b) = R(b)} \qquad \mathbf{(3.3)}$$
$$\mathbf{S(C(b)) = R(b)} \qquad \mathbf{(3.4)}$$

Let us introduce three Boolean variables:

$$\mathbf{SR:=(S(b) = R(b)} \qquad \mathbf{(3.5)}$$
$$\mathbf{CR:=(C(b) = R(b))} \qquad \mathbf{(3.6)}$$
$$\mathbf{SCR:= (S(C(b)) = R(b))} \qquad \mathbf{(3.7)}$$

SR=0 means that the identity SR (3.5) is not satisfied for a bit-string, while SR=1 means it is; similarly for CR (3.6) and SCR (3.7).

For each bit-string, I calculate SR, CR, and SCR; group strings with identical values of SR, CR, and SCR together; and count the number of bit-strings in each of the groups. An SQL script is presented in Appendix 3. Here are the results:

SR	CR	SCR	# of bit-strings
0	0	0	36
0	0	1	6
0	1	0	2
1	0	0	12
1	1	1	5

Table 3.1 *Number of bit-strings in SR-CR-SCR groups.*

Finally, I set

CS = CR | SCR **(3.8)**

(Pipe stands for logical OR). The results are as follows:

SR	CS	# of bit-strings
0	0	36
1	0	12
0	1	8
1	1	5

Table 3.2 *Number of bit-strings in SR-CS groups.*

As I mentioned above, there are 61 unique sequences or bit-strings in this model. With left-right, zero-one, and rotational symmetries, they were separated into 4 classes with 36, 12, 8, and 5 elements. In the SM, there are 61 observed (confirmed) particles, which include 36 quarks, 12 leptons, 8 gluons, and 5 other bosons.

Is this a coincidence? In the next article I argue that this is not coincidental by demonstrating that bit-string interactions in this model mimic particle interactions in the SM.

5. Discussion

Using the notions of a bit, bit-string (bit-concatenation), inversions of bit-strings, and a few Boolean operations (AND, OR, XOR), I built a model that matches some properties of the Standard Model of particle physics. While I suspect this match is not incidental, I leave it to the physics community to decide, should it happen to have interest in this approach.

However, if it turns out that the semblance of the models is not incidental, there are certain insights that this model can provide into the SM.

There seems to be no room for any new particles if we stay with the choice of k=4 (see part 3). There are three ways in which this model can be extended to accommodate new particles. First, a higher value of k can be used. For k=5, the number of unique bit-strings (“particles”) will be 9155; it is not clear whether this approach has any merits. Second, relaxing conditions 3.2 would multiply the number of “particles” by a factor of 2^n. SR, CR, SCR (3.5-3.7) properties will change in some cases - meaning that the new “particles” would have a different class identity than their base partners. This resembles some of the Supersymmetric approaches. However, the most “natural” way to accommodate additional particles into the model is to merge k=4 with all models with lower values of k. This will add a few more particles and a limited number of additional interaction types.

One of the essential properties of this approach is that nowhere does it require any hidden machinery or an "actor". Everything follows naturally; the only required ingredients are M0 and M1 machines (oscillators) that can combine into more advanced machines (interfere). Therefore, a model with k=1,2,3,4 is more plausible than just a model with k=4, since the latter would require something or someone to "forbid" models with k=1,2,3.

I believe that this model has grounds in reality because it is built using the minimal ingredients that any theory of reality could have. A frequent argument against models based on deterministic finite state machines is that they cannot reproduce the inherently probabilistic nature of Quantum Mechanics. I do not need to get into this discussion here. It is sufficient to point out that while Sn machines are deterministic finite state machines, their sets (either finite or infinite) do not have to be. Undoubtedly, to model any deep aspects of reality within this approach one would have to deal with the sets of Sn machines.

An important difference between this model and models built around the idea of cellular automata is that it does not require an arbitrary chosen lattice to operate. Though a deeper development of this model will require some kind of space, chances are that it will emerge naturally. For example, we already know that the space should allow interference and inversions. Furthermore, the model is free from any initial conditions or artificially tuned rules – all of which (including a lattice) are products of an "actor" or hidden machinery that demands another model to explain it.

In conclusion, I wished to demonstrate how the most basic approach may reveal insights into the fundamental properties of the Standard Model.

6. References

Dantzig, Tobias. *Number: The Language of Science.* New York: Pi Press, 2005.

Fredkin, Edward. "An Introduction to Digital Philosophy." *International Journal of Theoretical Physics*, 42.2 (2003): 189-247.

Fredkin, Edward. "Five big questions with pretty simple answers." *IBM Journal of Research,* 48.1 (2004): 31. Available from: http://dl.acm.org/citation.cfm?id=1014622

Fredkin, Edward. 2005. "A computing architecture for physics." *Proc. 2nd Conf. on Computing Frontiers,* Ischia, Italy. Available from: http://dl.acm.org/citation.cfm?id=1062261.1062307

Fredkin, Edward. "Discrete Theoretical Processes." In *A Computable Universe*, Hector Zenil, ed., World Scientific (publication pending).

Fredkin, Edward, and Toffoli, Tommaso. "Conservative logic, The Proceedings of the Physics of Computation Conference." *International Journal of Theoretical Physics*, 21.3/4, 21.6/7, 21.12 (1982): 219-253.

't Hooft, Gerard. Entangled Quantum States in a Local Deterministic Theory arXiv:0908.3408v1 [quant-ph]

Miller, Daniel B., and Fredkin, Edward. "Circular Motion of Strings in Cellular Automata, and Other Surprises." http://arxiv.org/ftp/arxiv/papers/1206/1206.2060.pdf

Miller, Daniel B., and Fredkin, Edward. 2005. "Two-state, Reversible, Universal Cellular Automata in Three Dimensions." *Proc. 2nd Conf. on Computing Frontiers*, Ischia, Italy: ACM 45, doi: 10.1145/1062271, arXiv:nlin/0501022.

Penrose, Roger. *The Road to Reality: A Complete Guide to the Laws of the Universe.* New York: A. A. Knopf, 2005.

Sanchez, Miguel, and Soler Gil, Francisco Jose. "Letter on M. Tegmark's 'The Mathematical Universe'", http://arxiv.org/pdf/0803.0944.pdf

Soler Gil, Francisco José, and Alfonseca, Manuel. "Is the multiverse hypothesis capable of explaining the fine tuning of nature laws and constants? The case of cellular automata", http://arxiv.org/ftp/arxiv/papers/1105/1105.4278.pdf

Tegmark, Max. "The mathematical universe." *Foundations of Physics*, 38 (2008): 101-150.

Wolfram, Stephen. (1986). Theory and applications of cellular automata, (1st ed.), (Singapore, World Scientific).

Wolfram, Stephen. (2002), A New Kind of Science (Wolfram Media).

Zahedi, Ramin. "On Discrete (Digital) Physics: as a Perfect Deterministic Structure for Reality - And the Fundamental Field Equations of Physics", http://arxiv.org/ftp/arxiv/papers/1501/1501.01373.pdf

Appendix 1. *64 Masks and Patterns They Generate*

MASK	PATTERN
000000	1,2,4,8,0; 3,6,12,9; 5,10; 7,15,14,13,11
000001	1,2,4,8,0; 3,6,13,11,7,15,14,12,9; 5,10
000010	1,2,4,8,0; 3,6,12,9; 5,11,7,15,14,13,10
000011	1,2,4,8,0; 3,6,13,10,5,11,7,15,14,12,9
000100	1,2,4,9,3,6,12,8,0; 5,10; 7,15,14,13,11
000101	1,2,4,9,3,6,13,11,7,15,14,12,8,0; 5,10
000110	1,2,4,9,3,6,12,8,0; 5,11,7,15,14,13,10
000111	1,2,4,9,3,6,13,10,5,11,7,15,14,12,8,0
001000	1,2,4,8,0; 3,7,15,14,13,11,6,12,9; 5,10
001001	1,2,4,8,0; 3,7,15,14,12,9; 5,10; 6,13,11
001010	1,2,4,8,0; 3,7,15,14,13,10,5,11,6,12,9
001011	1,2,4,8,0; 3,7,15,14,12,9; 5,11,6,13,10
001100	1,2,4,9,3,7,15,14,13,11,6,12,8,0; 5,10
001101	1,2,4,9,3,7,15,14,12,8,0; 5,10; 6,13,11
001110	1,2,4,9,3,7,15,14,13,10,5,11,6,12,8,0
001111	1,2,4,9,3,7,15,14,12,8,0; 5,11,6,13,10
010000	1,2,5,10,4,8,0; 3,6,12,9; 7,15,14,13,11
010001	1,2,5,10,4,8,0; 3,6,13,11,7,15,14,12,9
010010	1,2,5,11,7,15,14,13,10,4,8,0; 3,6,12,9
010011	1,2,5,11,7,15,14,12,9,3,6,13,10,4,8,0
010100	1,2,5,10,4,9,3,6,12,8,0; 7,15,14,13,11
010101	1,2,5,10,4,9,3,6,13,11,7,15,14,12,8,0
010110	1,2,5,11,7,15,14,13,10,4,9,3,6,12,8,0
010111	1,2,5,11,7,15,14,12,8,0; 3,6,13,10,4,9
011000	1,2,5,10,4,8,0; 3,7,15,14,13,11,6,12,9
011001	1,2,5,10,4,8,0; 3,7,15,14,12,9; 6,13,11
011010	1,2,5,11,6,12,9,3,7,15,14,13,10,4,8,0
011011	1,2,5,11,6,13,10,4,8,0; 3,7,15,14,12,9
011100	1,2,5,10,4,9,3,7,15,14,13,11,6,12,8,0
011101	1,2,5,10,4,9,3,7,15,14,12,8,0; 6,13,11
011110	1,2,5,11,6,12,8,0; 3,7,15,14,13,10,4,9
011111	1,2,5,11,6,13,10,4,9,3,7,15,14,12,8,0

MASK	PATTERN
100000	1,3,6,12,9,2,4,8,0; 5,10; 7,15,14,13,11
100001	1,3,6,13,11,7,15,14,12,9,2,4,8,0; 5,10
100010	1,3,6,12,9,2,4,8,0; 5,11,7,15,14,13,10
100011	1,3,6,13,10,5,11,7,15,14,12,9,2,4,8,0
100100	1,3,6,12,8,0; 2,4,9; 5,10; 7,15,14,13,11
100101	1,3,6,13,11,7,15,14,12,8,0; 2,4,9; 5,10
100110	1,3,6,12,8,0; 2,4,9; 5,11,7,15,14,13,10
100111	1,3,6,13,10,5,11,7,15,14,12,8,0; 2,4,9
101000	1,3,7,15,14,13,11,6,12,9,2,4,8,0; 5,10
101001	1,3,7,15,14,12,9,2,4,8,0; 5,10; 6,13,11
101010	1,3,7,15,14,13,10,5,11,6,12,9,2,4,8,0
101011	1,3,7,15,14,12,9,2,4,8,0; 5,11,6,13,10
101100	1,3,7,15,14,13,11,6,12,8,0; 2,4,9; 5,10
101101	1,3,7,15,14,12,8,0; 2,4,9; 5,10; 6,13,11
101110	1,3,7,15,14,13,10,5,11,6,12,8,0; 2,4,9
101111	1,3,7,15,14,12,8,0; 2,4,9; 5,11,6,13,10
110000	1,3,6,12,9,2,5,10,4,8,0; 7,15,14,13,11
110001	1,3,6,13,11,7,15,14,12,9,2,5,10,4,8,0
110010	1,3,6,12,9,2,5,11,7,15,14,13,10,4,8,0
110011	1,3,6,13,10,4,8,0; 2,5,11,7,15,14,12,9
110100	1,3,6,12,8,0; 2,5,10,4,9; 7,15,14,13,11
110101	1,3,6,13,11,7,15,14,12,8,0; 2,5,10,4,9
110110	1,3,6,12,8,0; 2,5,11,7,15,14,13,10,4,9
110111	1,3,6,13,10,4,9,2,5,11,7,15,14,12,8,0
111000	1,3,7,15,14,13,11,6,12,9,2,5,10,4,8,0
111001	1,3,7,15,14,12,9,2,5,10,4,8,0; 6,13,11
111010	1,3,7,15,14,13,10,4,8,0; 2,5,11,6,12,9
111011	1,3,7,15,14,12,9,2,5,11,6,13,10,4,8,0
111100	1,3,7,15,14,13,11,6,12,8,0; 2,5,10,4,9
111101	1,3,7,15,14,12,8,0; 2,5,10,4,9; 6,13,11
111110	1,3,7,15,14,13,10,4,9,2,5,11,6,12,8,0
111111	1,3,7,15,14,12,8,0; 2,5,11,6,13,10,4,9

Appendix 2. *61 Unique Patterns with Attributes*

CLASS	SR	CR	SCR	PATTERN	BITSTRING
B	1	1	1	1,2,5,11,7,15,14,13,10,4,8,0	101111010000
B	1	1	1	1,3,7,15,14,12,8,0	11110000
B	1	1	1	2,5,11,6,13,10,4,9	01101001
B	1	1	1	3,6,12,9	1001
B	1	1	1	5,10	10
G	0	0	1	1,2,4,9,3,6,13,11,7,15,14,12,8,0	10011011110000
G	0	0	1	1,2,5,11,7,15,14,12,8,0	1011110000
G	0	0	1	1,3,7,15,14,13,10,4,8,0	1111010000
G	0	0	1	1,3,7,15,14,13,11,6,12,9,2,4,8,0	11110110010000
G	0	0	1	2,5,11,6,12,9	011001
G	0	0	1	3,6,13,10,4,9	101001
G	0	1	0	1,2,4,9,3,7,15,14,13,11,6,12,8,0	10011110110000
G	0	1	0	1,3,6,13,11,7,15,14,12,9,2,4,8,0	11011110010000
L	1	0	0	1,2,4,8,0	10000
L	1	0	0	1,2,5,10,4,8,0	1010000
L	1	0	0	1,2,5,11,6,13,10,4,8,0	1011010000
L	1	0	0	1,3,6,12,8,0	110000
L	1	0	0	2,4,9	001
L	1	0	0	2,5,10,4,9	01001
L	1	0	0	2,5,11,7,15,14,13,10,4,9	0111101001
L	1	0	0	3,7,15,14,12,9	111001
L	1	0	0	5,11,6,13,10	11010
L	1	0	0	5,11,7,15,14,13,10	1111010
L	1	0	0	6,13,11	011
L	1	0	0	7,15,14,13,11	11011
Q	0	0	0	1,2,4,9,3,6,12,8,0	100110000
Q	0	0	0	1,2,4,9,3,6,13,10,5,11,7,15,14,12,8,0	1001101011110000
Q	0	0	0	1,2,4,9,3,7,15,14,12,8,0	10011110000
Q	0	0	0	1,2,4,9,3,7,15,14,13,10,5,11,6,12,8,0	1001111010110000
Q	0	0	0	1,2,5,10,4,9,3,6,12,8,0	10100110000
Q	0	0	0	1,2,5,10,4,9,3,6,13,11,7,15,14,12,8,0	1010011011110000
Q	0	0	0	1,2,5,10,4,9,3,7,15,14,12,8,0	1010011110000
Q	0	0	0	1,2,5,10,4,9,3,7,15,14,13,11,6,12,8,0	1010011110110000
Q	0	0	0	1,2,5,11,6,12,8,0	10110000
Q	0	0	0	1,2,5,11,6,12,9,3,7,15,14,13,10,4,8,0	1011001111010000
Q	0	0	0	1,2,5,11,6,13,10,4,9,3,7,15,14,12,8,0	1011010011110000
Q	0	0	0	1,2,5,11,7,15,14,12,9,3,6,13,10,4,8,0	1011110011010000
Q	0	0	0	1,2,5,11,7,15,14,13,10,4,9,3,6,12,8,0	1011110100110000
Q	0	0	0	1,3,6,12,9,2,4,8,0	110010000
Q	0	0	0	1,3,6,12,9,2,5,10,4,8,0	11001010000
Q	0	0	0	1,3,6,12,9,2,5,11,7,15,14,13,10,4,8,0	1100101111010000
Q	0	0	0	1,3,6,13,10,4,8,0	11010000
Q	0	0	0	1,3,6,13,10,4,9,2,5,11,7,15,14,12,8,0	1101001011110000
Q	0	0	0	1,3,6,13,10,5,11,7,15,14,12,8,0	1101011110000
Q	0	0	0	1,3,6,13,10,5,11,7,15,14,12,9,2,4,8,0	1101011110010000
Q	0	0	0	1,3,6,13,11,7,15,14,12,8,0	11011110000
Q	0	0	0	1,3,6,13,11,7,15,14,12,9,2,5,10,4,8,0	1101111001010000
Q	0	0	0	1,3,7,15,14,12,9,2,4,8,0	11110010000
Q	0	0	0	1,3,7,15,14,12,9,2,5,10,4,8,0	1111001010000
Q	0	0	0	1,3,7,15,14,12,9,2,5,11,6,13,10,4,8,0	1111001011010000
Q	0	0	0	1,3,7,15,14,13,10,4,9,2,5,11,6,12,8,0	1111010010110000
Q	0	0	0	1,3,7,15,14,13,10,5,11,6,12,8,0	1111010110000
Q	0	0	0	1,3,7,15,14,13,10,5,11,6,12,9,2,4,8,0	1111010110010000
Q	0	0	0	1,3,7,15,14,13,11,6,12,8,0	11110110000
Q	0	0	0	1,3,7,15,14,13,11,6,12,9,2,5,10,4,8,0	1111011001010000
Q	0	0	0	2,5,11,7,15,14,12,9	01111001
Q	0	0	0	3,6,13,10,5,11,7,15,14,12,9	10101111001
Q	0	0	0	3,6,13,11,7,15,14,12,9	101111001
Q	0	0	0	3,7,15,14,13,10,4,9	11101001
Q	0	0	0	3,7,15,14,13,10,5,11,6,12,9	11101011001
Q	0	0	0	3,7,15,14,13,11,6,12,9	111011001

Appendix 3. ***SQL Code***

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO


-- Script creating a table that contains all bit-strings along with their patterns, masks, and other attributes

  CREATE TABLE [dbo].[UFB](
            [MASK] [nvarchar](max) NULL,
            [PATTERN] [nvarchar](max) NULL,
            [BITSTRING] [nvarchar](max) NULL,
            [SR] [bit] NULL,
            [CR] [bit] NULL,
            [SCR] [bit] NULL,
            [T] [nvarchar](1) NULL,
            [QE] [smallint] NULL
  ) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

  GO


-- Helper Functions


CREATE FUNCTION [dbo].[IB]
(
 @I INT,
 @D INT = 0
)
RETURNS NVARCHAR(MAX)
AS
BEGIN
 DECLARE @OUTPUT NVARCHAR(MAX) = ''
 WHILE (@I != 0)
 BEGIN
            SELECT @OUTPUT = CONVERT(NVARCHAR(1), (@I%2)&1) + @OUTPUT, @I = @I / 2
 END
 IF(@D > 0 AND LEN(@OUTPUT) < @D) SET @OUTPUT = RIGHT('00000000000000000000' + @OUTPUT, @D)
 RETURN @OUTPUT
END
GO


CREATE FUNCTION [dbo].[NP]
(
            @MASK NVARCHAR(MAX),
            @S        NVARCHAR(MAX),
            @M        INT
)
```

```
RETURNS NVARCHAR(MAX)
AS
BEGIN
	DECLARE  @A INT, @P INT
	SELECT @P = CONVERT(INT, REPLACE(REPLACE(@S,'[',''),']',''))
	SET  @A = CONVERT(INT, SUBSTRING(@MASK,@P%@M + 1,1))^(@P/@M)
	RETURN '['+CONVERT(NVARCHAR(MAX),(2*@P + @A) % (2*@M))+']'
END
GO


CREATE FUNCTION [dbo].[NNP]
(
	@PATH  NVARCHAR(MAX),
	@S	NVARCHAR(MAX),
	@M	INT,
	@K	INT
)
RETURNS NVARCHAR(MAX)
AS
BEGIN
	DECLARE  @A INT
	SELECT  @A = CHARINDEX(@S,@PATH,1)
	IF @A = 0 RETURN @S
	IF @K < 2*@M-1 RETURN dbo.NNP(@PATH,'['+CONVERT(NVARCHAR(MAX),@K+1)+']',@M,@K+1)
	RETURN ''
END
GO


CREATE FUNCTION [dbo].[MP]
(
	@MASK NVARCHAR(MAX),
	@D	INT
)
RETURNS NVARCHAR(MAX)
AS
BEGIN
	DECLARE @K INT = 0, @S NVARCHAR(MAX)='[0]', @PATH NVARCHAR(MAX)='', @Z NVARCHAR(MAX)
	WHILE @K<2*@D
	BEGIN
		SELECT @S = dbo.NP(@MASK,REPLACE(@S,';',''),@D)
		SELECT @Z = dbo.NNP(@PATH,@S,@D,0)
		IF @S <> @Z SELECT @S = ';' + @Z
		SELECT @PATH=@PATH+@S, @K=@K+1, @S = @Z
	END
	RETURN @PATH
END
GO


CREATE FUNCTION [dbo].[SplitString]
(
```

```
   @S NVARCHAR(MAX),
   @D CHAR(1)
)
RETURNS @OUT TABLE(SUBS NVARCHAR(MAX)
)
BEGIN
   DECLARE @START INT = 1, @END INT = 0
   WHILE @START < LEN(@S) + 1 BEGIN
                        SET @END = CHARINDEX(@D, @S, @START)
      IF @END = 0 SET @END = LEN(@S) + 1
      INSERT INTO @OUT VALUES(SUBSTRING(@S, @START, @END - @START))
      SET @START = @END + 1
   END
   RETURN
END
GO


CREATE FUNCTION [dbo].[BitString]
(
          @S NVARCHAR(MAX),
          @D CHAR(1)
)
RETURNS NVARCHAR(MAX)
AS
BEGIN
          DECLARE  @OUTPUT NVARCHAR(MAX) = '', @START INT=1, @END INT=0
          WHILE @START < LEN(@S) + 1 BEGIN
                    SET @END = CHARINDEX(@D, @S,@START)
                    IF @END = 0 SET @END = LEN(@S) + 1
                    SELECT @OUTPUT = @OUTPUT + Convert(varchar(1),Convert(int,(SUBSTRING(@S, @START,
@END - @START)))&1), @START = @END + 1
          END
          RETURN @OUTPUT
END
GO


CREATE FUNCTION [dbo].[R]
(
          @S1 NVARCHAR(MAX),
          @S2 NVARCHAR(MAX)
)
RETURNS INT
AS
BEGIN
          DECLARE @LN INT, @K int=0, @C NVARCHAR(MAX)
          IF LEN(@S1) <> LEN (@S2) RETURN 0
          SELECT @LN = LEN(@S1), @C = @S2
          WHILE @K < @LN BEGIN
                    IF @S1 = @C RETURN 1
                    SELECT @C = SUBSTRING(@C,@LN,1) + SUBSTRING(@C,1,@LN-1),@K = @K + 1
          END
```

```
	RETURN 0
END
GO


CREATE FUNCTION [dbo].[C]
(
   @S NVARCHAR(MAX)
)
RETURNS NVARCHAR(MAX)
AS
BEGIN
	RETURN REPLACE(REPLACE(REPLACE(@S, '0', '#'), '1', '0'), '#', '1')
END
GO



CREATE PROCEDURE [dbo].[MPP] -- The Generator
(
	@D		INT,
	@MD		INT,
	@F		NVARCHAR(1),
	@L		NVARCHAR(1)
)
AS
BEGIN
	DECLARE @PATH NVARCHAR(MAX), @MASK NVARCHAR(MAX), @M INT = 0
	WHILE @M < @MD
	BEGIN
		SELECT @MASK = @F + dbo.IB(@M,@D-2) + @L, @M=@M+1
		SELECT @PATH=dbo.MP(@MASK,@D)
		INSERT UFB
		SELECT @MASK,SUBS,dbo.BitString(SUBS,','),0,0,0,'',0
		FROM dbo.SplitString(REPLACE(REPLACE(REPLACE(@PATH,'][',','),'[',''),']',''),';') S
	END
END
GO


-- Script that populates the UFB table

DELETE FROM UFB -- clean up the table
DECLARE @D INT = 8  -- mask size
DECLARE @MD INT = 64 -- number of masks

EXEC dbo.MPP @D, @MD,'1','1' -- bit-string generator

-- Calculating SR, CR, and SCR values for each string

UPDATE UFB SET SR=dbo.R(BITSTRING, REVERSE(BITSTRING)),
		CR=dbo.R(BITSTRING, dbo.C(BITSTRING)),
		SCR=dbo.R(BITSTRING, dbo.C(REVERSE(BITSTRING)))
```

```
-- Setting up type (class) of a bit-string

UPDATE UFB SET T = 'Q' WHERE SR|CR|SCR=0
UPDATE UFB SET T = 'G' WHERE SR=0 AND CR|SCR=1
UPDATE UFB SET T = 'L' WHERE SR=1 AND CR|SCR=0
UPDATE UFB SET T = 'B' WHERE SR&CR&SCR=1

-- Mask cleanup - removing first and last bit

UPDATE UFB SET MASK = SUBSTRING(MASK,2,@D-2)


-- Generating result tables

SELECT DISTINCT T,SR,CR,SCR,PATTERN,BITSTRING FROM UFB ORDER BY T,SR,CR,SCR,PATTERN

SELECT SR,CR,SCR, COUNT(*) AS "# OF BIT-STRINGS" FROM (SELECT DISTINCT SR,CR,SCR,PATTERN FROM UFB) D
GROUP BY SR,CR,SCR ORDER BY SR,CR,SCR

SELECT SR,CR|SCR AS "SC", COUNT(*) AS "# OF BIT-STRINGS" FROM (SELECT DISTINCT SR,CR,SCR,PATTERN FROM
UFB) D GROUP BY SR,CR|SCR ORDER BY SR,CR|SCR

SELECT DISTINCT MASK, BITSTRING, PATTERN FROM UFB WHERE T='G' ORDER BY  MASK, BITSTRING, PATTERN

SELECT DISTINCT MASK, BITSTRING, PATTERN FROM UFB WHERE T='Q' ORDER BY  MASK, BITSTRING, PATTERN
```