Implementation of Coder (for C(7, 4)) and Decoder for Meggitt Algorithm

In [1]:
```python
import numpy as np
from time import time
import itertools as it
from termcolor import colored
```

Implementation of Systematic Coder :

- - **Note:**

- - - For C(7, 4) we have: $\mathbf{G} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$

- **Lookup Table that includes P Array:**

In [2]:
```python
LOOKUP_TABLE = {(7, 4): {'P': np.array([[1, 1, 0], \
                                        [0, 1, 1], \
                                        [1, 1, 1], \
                                        [1, 0, 1]], dtype=np.int64)}}
```

- **G Generation:**

In [3]:
```python
def G_generator(Linear_Block_Code: tuple) -> np.ndarray:
    n, k = Linear_Block_Code
    I_k = np.identity(k, dtype=np.int64)
    P = LOOKUP_TABLE[Linear_Block_Code]['P']
    G = np.concatenate((P, I_k), axis=1, dtype=np.int64)
    return G
```

- - **Test:**

In [4]:
```python
G = G_generator(Linear_Block_Code=(7, 4))
print(f'\n{colored("For Systematic Linear Block Code of C(7, 4) G Matrix will be:", "blue", attrs=["bold"])}\n\n{colored("G =", "black", attrs=["bold"])} \n{
```

```
For Systematic Linear Block Code of C(7, 4) G Matrix will be:

G =
[[1 1 0 1 0 0 0]
 [0 1 1 0 1 0 0]
 [1 1 1 0 0 1 0]
 [1 0 1 0 0 0 1]]
```

- **U Generation:**

In [5]:
```python
def U_generator(k: int) -> np.ndarray:
    U = np.array(list(it.product([0, 1], repeat=k)), dtype=np.int64)
    return U
```

- - **Test:**

In [6]:

```
U = U_generator(k=4)
print(f'\n{colored("For k=4 U Matrix that includes our Messages will be: ", "blue", attrs=["bold"])}\n\n{colored("U =", "black", attrs=["bold"])}\n{U}\n')
```

```
For k=4 U Matrix that includes our Messages will be:

U =
[[0 0 0 0]
 [0 0 0 1]
 [0 0 1 0]
 [0 0 1 1]
 [0 1 0 0]
 [0 1 0 1]
 [0 1 1 0]
 [0 1 1 1]
 [1 0 0 0]
 [1 0 0 1]
 [1 0 1 0]
 [1 0 1 1]
 [1 1 0 0]
 [1 1 0 1]
 [1 1 1 0]
 [1 1 1 1]]
```

- **V Generation**

In [7]:

```
def Coder(Linear_Block_Code: tuple) -> np.ndarray:
    k = Linear_Block_Code[1]
    P = LOOKUP_TABLE[Linear_Block_Code]['P']
    U = U_generator(k=k)
    Parity_mat = (U @ P) % 2
    V = np.concatenate((Parity_mat, U), axis=1)
    return V
```

- - **Test:**

In [8]:

```
V = Coder(Linear_Block_Code=(7, 4))
print(f'\n{colored("For Systematic Linear Block Code of C(7, 4) V Matrix that includes our Codewords will be:", "blue", attrs=["bold"])}\
\n\n{colored("V =", "black", attrs=["bold"])} \n{V}\n')
```

```
For Systematic Linear Block Code of C(7, 4) V Matrix that includes our Codewords will be:

V =
[[0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1]
 [1 1 1 0 0 1 0]
 [0 1 0 0 0 1 1]
 [0 1 1 0 1 0 0]
 [1 1 0 0 1 0 1]
 [1 0 0 0 1 1 0]
 [0 0 1 0 1 1 1]
 [1 1 0 1 0 0 0]
 [0 1 1 1 0 0 1]
 [0 0 1 1 0 1 0]
 [1 0 0 1 0 1 1]
 [1 0 1 1 1 0 0]
 [0 0 0 1 1 0 1]
 [0 1 0 1 1 1 0]
 [1 1 1 1 1 1 1]]
```

Implementation of Meggitt Decoder :

- **U Generation:**

In [9]:

```
U = U_generator(k=4)
print(f'\n{colored("U =", "black", attrs=["bold"])}\n\n{U}\n')
```

```
U =

[[0 0 0 0]
 [0 0 0 1]
 [0 0 1 0]
 [0 0 1 1]
 [0 1 0 0]
 [0 1 0 1]
 [0 1 1 0]
 [0 1 1 1]
 [1 0 0 0]
 [1 0 0 1]
 [1 0 1 0]
 [1 0 1 1]
 [1 1 0 0]
 [1 1 0 1]
 [1 1 1 0]
 [1 1 1 1]]
```

- **V Generation:**

- - **Note:**

- - - We assume that the transmitted array is an array that includes all of the codewords

In [10]:
```python
V = Coder(Linear_Block_Code=(7, 4))
print(f'\n{colored("V =", "black", attrs=["bold"])}\n\n{V}\n')
```

```
V =

[[0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1]
 [1 1 1 0 0 1 0]
 [0 1 0 0 0 1 1]
 [0 1 1 0 1 0 0]
 [1 1 0 0 1 0 1]
 [1 0 0 0 1 1 0]
 [0 0 1 0 1 1 1]
 [1 1 0 1 0 0 0]
 [0 1 1 1 0 0 1]
 [0 0 1 1 0 1 0]
 [1 0 0 1 0 1 1]
 [1 0 1 1 1 0 0]
 [0 0 0 1 1 0 1]
 [0 1 0 1 1 1 0]
 [1 1 1 1 1 1 1]]
```

- **For Desired Error Pattern Matrix:**

In [11]:
```python
E = np.array([[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 1, 0, 0], \
              [0, 0, 0, 1, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], \
              [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 1, 0, 0], \
              [0, 0, 0, 1, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0]], dtype=np.int64)
print(f'\n{colored("Desired Error Patterns:", "blue", attrs=["bold"])} \n\n{colored("E = ", "black", attrs=["bold"])}\n{E}\n')
```

```
Desired Error Patterns:

E =
[[0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1]
 [0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0]
 [0 0 0 1 0 0 0]
 [0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0]
 [1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1]
 [0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0]
 [0 0 0 1 0 0 0]
 [0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0]
 [1 0 0 0 0 0 0]]
```

- **Received Vectors (R) Generation:**

In [12]:
```python
def Channel_Out(Codewords: np.ndarray, E_patts: np.ndarray) -> np.ndarray:
    R = (Codewords + E_patts) % 2
    return R
```

In [13]:
```python
R = Channel_Out(Codewords=V, E_patts=E)
print(f'\n{colored("R =", "black", attrs=["bold"])}\n\n{R}\n')
```

```
R =

[[0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0]
 [1 1 1 0 0 0 0]
 [0 1 0 0 1 1 1]
 [0 1 1 1 1 0 0]
 [1 1 1 0 1 0 1]
 [1 1 0 0 1 1 0]
 [1 0 1 0 1 1 1]
 [1 1 0 1 0 0 0]
 [0 1 1 1 0 0 0]
 [0 0 1 1 0 0 0]
 [1 0 0 1 1 1 1]
 [1 0 1 0 1 0 0]
 [0 0 1 1 1 0 1]
 [0 0 0 1 1 1 0]
 [0 1 1 1 1 1 1]]
```

- **H Generation:**

In [14]:
```python
def H_generator(Linear_Block_Code: tuple) -> np.ndarray:
    n, k = Linear_Block_Code
    P = LOOKUP_TABLE[Linear_Block_Code]['P']
    I_n_k = np.identity(n - k, dtype=np.int64)
    H = np.concatenate((I_n_k, P.T), axis=1)
    return H
```

- - **Test:**

In [15]:

```
H = H_generator(Linear_Block_Code=(7, 4))
print(f'\n{colored("For Systematic Linear Block Code of C(7, 4) Parity-Check Matrix will be:", "blue", attrs=["bold"])}\n\n\
{colored("H =", "black", attrs=["bold"])} \n{H}\n')
```

```
For Systematic Linear Block Code of C(7, 4) Parity-Check Matrix will be:

H =
[[1 0 0 1 0 1 1]
 [0 1 0 1 1 1 0]
 [0 0 1 0 1 1 1]]
```

- **S Generation:**

In [16]:

```
def S_generator(R: np.ndarray, H: np.ndarray) -> np.ndarray:
    S = (R @ H.T) % 2
    return S
```

- - **Test:**

In [17]:

```
S = S_generator(R=R, H=H)
print(f'\n{colored("S =", "black", attrs=["bold"])}\n\n{S}\n')
```

```
S =

[[0 0 0]
 [1 0 1]
 [1 1 1]
 [0 1 1]
 [1 1 0]
 [0 0 1]
 [0 1 0]
 [1 0 0]
 [0 0 0]
 [1 0 1]
 [1 1 1]
 [0 1 1]
 [1 1 0]
 [0 0 1]
 [0 1 0]
 [1 0 0]]
```

- **Decoder Generation:**

In [45]:

```
def Meggitt_Decoder(R: np.array, C: tuple=(7, 4)) -> np.array:
    n, k = C
    H = H_generator(Linear_Block_Code=C)
    decoded_r_list = []
    for r in R:
        shift_register_r = r
        for j in range(n):
            s = (shift_register_r @ H.T) % 2
            if list(s) == [1, 0, 1]:
                shift_register_r[-1] = (shift_register_r[-1] + 1) % 2
            shift_register_r = np.roll(shift_register_r, 1)

        corrected_r = shift_register_r
        decoded_r_list.append(corrected_r)

    return np.array(decoded_r_list)
```

- - **Test:**

In [50]:

```
V_hat = Meggitt_Decoder(R=R)
print(f'\n{colored("V = ", "black", attrs=["bold"])}\n{V}\n')
print(f'\n{colored("V-hat = ", "black", attrs=["bold"])}\n{V_hat}\n')
```

```
V =
[[0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1]
 [1 1 1 0 0 1 0]
 [0 1 0 0 0 1 1]
 [0 1 1 0 1 0 0]
 [1 1 0 0 1 0 1]
 [1 0 0 0 1 1 0]
 [0 0 1 0 1 1 1]
 [1 1 0 1 0 0 0]
 [0 1 1 1 0 0 1]
 [0 0 1 1 0 1 0]
 [1 0 0 1 0 1 1]
 [1 0 1 1 1 0 0]
 [0 0 0 1 1 0 1]
 [0 1 0 1 1 1 0]
 [1 1 1 1 1 1 1]]


V-hat =
[[0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1]
 [1 1 1 0 0 1 0]
 [0 1 0 0 0 1 1]
 [0 1 1 0 1 0 0]
 [1 1 0 0 1 0 1]
 [1 0 0 0 1 1 0]
 [0 0 1 0 1 1 1]
 [1 1 0 1 0 0 0]
 [0 1 1 1 0 0 1]
 [0 0 1 1 0 1 0]
 [1 0 0 1 0 1 1]
 [1 0 1 1 1 0 0]
 [0 0 0 1 1 0 1]
 [0 1 0 1 1 1 0]
 [1 1 1 1 1 1 1]]
```

- ▪ **Equality Check between V and $\hat{V}$ for [Meggit Decoder]**

In [54]:
```
Equality_Check = np.unique(V == V_hat)[0]
print(f'\n{colored("Equality Check: ", "black", attrs=["bold"])}{colored(Equality_Check, "blue", attrs=["bold"])}\n')
```

```
Equality Check: True
```

- **Conclusion:**

- ▪ As we saw for error patterns that have Hamming Distance of 1, the Meggitt Decoder does decoding as correctly when we use C(7, 4) for Coding.