SECDED Templates

For 8/4, 16/11, 32/26 bit cases of code/data.

Python program for coding

SECDED, 8 bit code, 4 bit data A

Orig Data				**Code**						
Bit Position	7	6	5	4	3	2	1	0	Calc	✗ ✓
Parity | Bitmask	**D4**	**D3**	**D2**	**P4**	**D1**	**P2**	**P1**	**P0**	**P's**	
P0 | FF				•				**?**		
P1 | AA							**?**			
P2 | CC						**?**				
P4 | F0				**?**						
Correct Bit(s)										
Correct Data				Code						

SECDED, 16 bit code,11 bit data C

Orig Data						**Code**												
Bit Position	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Calc**	✗ ✓
Parity | Bitmask	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**P8**	**D4**	**D3**	**D2**	**P4**	**D1**	**P2**	**P1**	**P0**	**P's**	
P0 | FFFF																		
P1 | AAAA																		
P2 | CCCC																		
P4 | F0F0																		
P8 | FF00																		
Correct Bit(s)																		
Correct Data						**Code**												

SECDED 32 bit code, 26 bit data D

Orig Data																																									**Calc**
Bit Pos'n						28				24				20					16				12				8				4			P2		P1		P0		**P's**
Data + Parity	**P0**																																					?		
AAAAAAAA	**P1**																																			?				
CCCCCCC	**P2**																																	?						
F0F0F0F0	**P4**																														?									
FF00FF00	**P8**																										?													
FFFF0000	**P16**																		?																					
Correct Bit(s)																																								
Correct Data														**Code**																										

SECDED program in Python

```
def SECDED_correct(N, debug=0) :
    # returns N if correct(ed), or 0xFFFF if uncorrectable
    mask = [0xFFFFFFFE, 0xAAAAAAA8, 0xCCCCCCC8, 0xF0F0F0E0, 0xFF00FE00]
    pPos = [         0,          1,          2,          4,         8]
    pCalc = [0,0,0,0,0]
    pStored = [0,0,0,0,0]
    bittocorrect = 0
    for i in range(5) :
        pCalc[i] = (count1bits(N & mask[i]) & 31)
        pStored[i] = 1 if (N & (1 << pPos[i])) > 0 else 0
        if (i > 0 and (1&pCalc[i]) != pStored[i]):   # found an error
            bittocorrect += pPos[i]
        if (debug & 1):
            print(i, bittocorrect, pPos[i], pCalc[i], pStored[i])
    error_detected = (bittocorrect > 0)
    can_correct = (error_detected and (1 & pCalc[0] != pStored[0]))
    newN = N ^ (1 << bittocorrect)     # Flip the erroneous bit
    if (debug & 2):
        print("N ",error_detected,can_correct,hex(N),hex(newN))
    if (error_detected):
        if (can_correct):
            return(newN)     #  bit position of error
        else:
            return(-1)       # -1 is never a valid code
    else:
        return(N)            # corrected or unchanged value
```

```
def count1bits):
    count = 0
     while n:
         n &= n-1
         count += 1
     return cnt
```