

**School of Electrical Engineering (SELECT)**

**Winter Semester 2023-24**

**BEEE206P – Digital Electronics Laboratory**

**Register No.:23BEE1146**

**Student Name :Tarun Prabhakar S**

**List of Experiments**

|  |  |
| --- | --- |
| **Exp.No.** | **Experiment Description** |
| **Hardware Experiments** | |
|  | Verification of Logic Gates |
|  | Verification of Boolean and De Morgan’s laws |
|  | Adder & Subtractor (Half & Full) |
|  | Magnitude Comparator |
|  | Code Converters |
| **Machine Learning Approach to Digital Electronics** | |
|  | Half Adder & Full Adder |
|  | Half Subtractor & Full Subtractor |
|  | 2 to 4 Decoder |
|  | 4 to 2 Encoder |
|  | 4 Bit Ripple Carry Adder |
|  | 4 Bit Binary to Gray Code Converter |
|  | 4 Bit Gray to Binary Code Converter |
|  | 4 Bit Binary to XS3 Code Converter |
|  | 4 Bit XS3 to Binary Code Converter |
|  | 2 Bit Magnitude Comparator |
|  | Parity Generator |
|  | Parity Detector |
|  | 4x1 Multiplexer |
|  | 1x4 Demultiplexer |
| **Verilog for Digital Electronics** | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| **Experiment No** |
| **Date:** |
| **Title of the Experiment:** |
| **Aim of the Experiment:** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:** |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No. 1** |
| **Date:** |
| **Title of the Experiment:**  **HALF ADDER AND FULL ADDER** |
| **Aim of the Experiment:**  **VERIFYING HALF ADDER AND FULL ADDER BY USING PYTHON CODE, LOGIC GATES AND TRUTH TABLE** |
| **Digital Logic Circuit diagram:** |

|  |
| --- |
| **Truth Table:** |
| **Python Code:**  #half adder  def halfadder(a,b):    sum=a^b    carry=a&b    return sum,carry  x=int(input("enter a"))  y=int(input("enter b"))  print(halfadder(x,y))  #fullader  def fulladder(a,b,c):    sum=a^b^c    carry=a&b|b&c|c&a    return sum,carry  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter Cin"))  print(fulladder(x,y,z)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 2** |
| **Date: 27-02-2024** |
| **Title of the Experiment: HALF AND FULL SUBRACTOR** |
| **Aim of the Experiment: VERIFYING HALF SUBRACTOR FULL SUBRACTOR BY USING PYTHON CODE, LOGIC GATES AND TRUTH TABLE** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #half sub  def halfsub(a,b):    diff=a^b    z=not a    borow=z&b    return diff,borow  x=int(input("enter a"))  y=int(input("enter b"))  print(halfsub(x,y))  #fullsub  def fullsub(a,b,c):    diff=a^b^c    z=not a    borow=(z&b|b&c|z&c)    return diff,borow  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter c"))  print(fullsub(x,y,z)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 3** |
| **Date:27-02-2024** |
| **Title of the Experiment: 2 to 4 Decoder** |
| **Aim of the Experiment:**  **To verify 2 to 4 decoder using basic logic gates** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #2 to 4 decoder  def decoder(a,b):    notb=not b    nota=not a    d0=a&b    d1=a&notb    d2=nota&b    d32=nota&notb    d3=d32&1    return d0,d1,d2,d3  x=int(input("enter a"))  y=int(input("enter b"))  print(decoder(x,y)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 4** |
| **Date:27-02-2024** |
| **Title of the Experiment:**  **4 to 2 ENCODER** |
| **Aim of the Experiment: To verify 4 to 2 encoder using logic gates** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #4to2 encoder  def encoder(a,b,c,d):    d1=not a    d2=not b    d3=not c    d4=not d    q1=(a|b)    q2=(a|c)    return q1,q2  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter c"))  z1=int(input("enter d"))  print(encoder(x,y,z,z1)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 5** |
| **Date:27-02-2024** |
| **Title of the Experiment: 4 BIT RIPPLE CARRY ADDER** |
| **Aim of the Experiment:**  **To implement 4 bit ripple carry adder using logic circuits** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #ripple carry adder  def fulladder(a,b,c):    sum=a^b^c    carry=a&b|b&c|c&a    return sum,carry  cin=0  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter c"))  z1=int(input("enter d"))  l=int(input("enter a"))  m=int(input("enter b"))  n=int(input("enter c"))  n1=int(input("enter d"))  k=fulladder(x,l,cin)  print("sum of 1st bit",k[0])  k=fulladder(y,m,k[1])  print("sum of 2nd bit",k[0])  k=fulladder(z,n,k[1])  print("sum of 3rd bit",k[0])  k=fulladder(z1,n1,k[1])  print("sum of 4th bit",k[0])  print("cout",k[1]) |
| **Screenshot of the code and output** |
|  |
| **Experiment No 6** |
| **Date:27-02-2024** |
| **Title of the Experiment: 4 bit binary to graycode converter** |
| **Aim of the Experiment: to implement 4 bit binary to gray code converter using logic gates** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #greycode  def greycode(a,b,c,d):    g1=a    g2=a^b    g3=b^c    g4=c^d    return g1,g2,g3,g4  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter c"))  z1=int(input("enter d"))  print(greycode(x,y,z,z1)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 7** |
| **Date:27-02-2024** |
| **Title of the Experiment: 4 bit grey to binary** |
| **Aim of the Experiment: to implement grey to binary converter using logic gates** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #grey to binary  def greycode(a,b,c,d):    g3=a    g2=a^b    g1=a^(b^c)    g0=(a^b)^(c^d)    return g3,g2,g1,g0  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter c"))  z1=int(input("enter d"))  print(greycode(x,y,z,z1)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 8** |
| **Date:27-04-2024** |
| **Title of the Experiment: 4 bit binary to ex 3 converter** |
| **Aim of the Experiment: to implement 4 bit binary to ex3 using logic gates** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #bcd to excess 3  def excess3(a,b,c,d):    notb=not b    notc=not c    a1= a|(b&(c|d))    a2=b^(c|d)    a3= not (c^d)    c=a3    d=not d    return a1,a2,c,d  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter c"))  z1=int(input("enter d"))  print(excess3(x,y,z,z1)) |
| **Screenshot of the code and output** |
|  |
|  |

|  |
| --- |
| **Experiment No 9** |
| **Date:27-02-2024** |
| **Title of the Experiment: 3 EX to bcd converter** |
| **Aim of the Experiment: to implement 3 ex to bcd converter** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  #ex 3 to bcd  def ex3(a,b,c,d):    notb =not b    notc=not c    notd=not d    nota=not a    d1=notd    c1=c^d    b1=(notb & notc)|(notb &notd)|(b&c&d)    a1=(c&d)|(a&b)    return a1,b1,c1,d1  x=int(input("enter a"))  y=int(input("enter b"))  z=int(input("enter c"))  z1=int(input("enter d"))  print(ex3(x,y,z,z1)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 10** |
| **Date: 27-02-2024** |
| **Title of the Experiment: 2 bit magnitude comperator** |
| **Aim of the Experiment: to implement 2 bit magnitude comparator using logic circuits** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  def bit(a,b):    notb=not b    nota=not a    ab1=a&notb    ab=nota&b    ab2=a^b    ab3=not(ab2)    return ab1,ab,ab3  x=int(input("enter a"))  y=int(input("enter b"))  k=bit(x,y)  print("a>b",k[0],"a<b",k[1],"a=b",k[2]) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 11** |
| **Date:27-02-24** |
| **Title of the Experiment: parity generator** |
| **Aim of the Experiment: to verify parity generator using logic circuits** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:**  **Odd parity truth table**    **Even parity truth table** |
| **Python Code:**  def evenparity(a,b,c):    x=a^b    p=x^c    return p  def oddparity(a,b,c):    x=b^c    y=not x    z=y^a    return z  l=int(input("enter l"))  m=int(input("enter m"))  n=int(input("enter n"))  h=evenparity(l,m,n)  u=oddparity(l,m,n)  print("even bit",h)  print("odd bit",u) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 12** |
| **Date:27-02-2024** |
| **Title of the Experiment: parity checker** |
| **Aim of the Experiment: To find parity bits using logic gates** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:**  **Even parity checker**    **Odd parity checker** |
| **Python Code:**  def evenchk(a,b,c,p):    g=not(a^b^c^p)    return g  def oddchk(a,b,c,p):    h=a^b^c^p    return h  l=int(input("enter l"))  m=int(input("enter m"))  n=int(input("enter n"))  x=int(input("enetr p"))  print(evenchk(l,m,n,x))  print(oddchk(l,m,n,x)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 13** |
| **Date:27-02-2024** |
| **Title of the Experiment: 4X1 mux** |
| **Aim of the Experiment: to verify 4X1 MUX using logic circuit** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  def mux(l,m,n,o,a,b):    nota=not a    notb=not b    l1=nota&l&notb    l2=nota&m&b    l3=a&n&notb    l4=a&o&b    return l1,l2,l3,l4  x=int(input("enter l"))  y=int(input("enter m"))  z=int(input("enter n"))  z1=int(input("enter o"))  x1=int(input("enter a"))  y1=int(input("enter b"))    print(mux(x,y,z,z1,x1,y1)) |
| **Screenshot of the code and output** |
|  |

|  |
| --- |
| **Experiment No 14** |
| **Date:27-02-2024** |
| **Title of the Experiment: 1X4 DEMUX** |
| **Aim of the Experiment: to implement 1X4 demux using logic gates** |
| **Digital Logic Circuit diagram:** |
| **Truth Table:** |
| **Python Code:**  def demux(a,b):    nota=not a    notb=not b    l=nota&notb    m=nota&b    n=a&notb    o=a&b    return l,m,n,o  x1=int(input("enter a"))  y1=int(input("enter b"))  print(demux(x1,y1)) |
| **Screenshot of the code and output** |
|  |