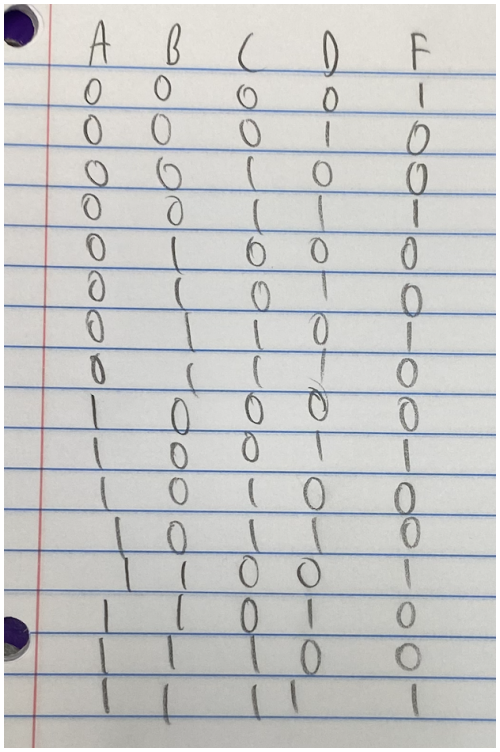
Jonathan Ameri
3/22/22
HW4

Q1:

```
@always_comb
def comb():
    # use and or not
    # or use & | ~, but only keep the LSB in the end
    # change the following line to give f correct value

    nota = not a
    notb = not b
    notc = not c
    notd = not d

    term1 = nota & notb & notc & notd
    term2 = nota & notb & c & d
    term3 = nota & b & c & notd
    term4 = a & notb & notc & d
    term5 = a & b & notc & notd
    term6 = a & b & c & d

    f.next = term1 | term2 | term3 | term4 | term5 | term6

# return the logic
return comb
```

The logic table of my myHDL file is the same as the manually created table

a	b	c	d	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



Q2:

```
reg1 = Register(state, next_state, clock, reset) #dout = state, din = next_state

# TODO
# instantiate a register here.
# next_state is the input and state is the output
```

```python
# generate next_state, based on state and b
@always_comb
def next_state_logic():
    if state == 0:
        if b == 0:
            next_state.next = 0
        elif b == 1:
            next_state.next = 1
    elif state == 1:
        if b == 0:
            next_state.next = 2
        elif b == 1:
            next_state.next = 0
    elif state == 2:
        if b == 0:
            next_state.next = 1
        elif b == 1:
            next_state.next = 2


    # TODO
    # We can use if-elif-else statements in Python
    # next_state.next = ...
    pass
```

```python
# generate output
@always_comb
def z_logic():
    if state == 0:
        z.next = 1
    else:
        z.next = 0
    # TODO
    # generate z from state
    pass
```

```
(hdlvenv) MacBook-Pro-4:hw4-code jonathanameri$ python q2.py 10100101
b | z v
1 | 0 1
0 | 0 2
1 | 0 5
0 | 0 10
0 | 0 20
1 | 0 41
0 | 0 82
1 | 1 165
```

Q5:

```
# -1:    4294967295
uint2decstr:
        addi sp, sp, -12        # Reserve space to store data
        sw s0, 8(sp)            # save values of s0, s1, and ra
        sw s1, 4(sp)
        sw ra, 0(sp)
        add s0, x0, a0          # save the address of input string into s0
        add s1, x0, a1          # save the value of v into s1
        addi t0, x0, 10         # save the value 10 into t0
        blt s1, t0, skip        # If v >= 10, recursive call
        divu a1, s1, t0         # save new value of v for recursive call into a1
        jal ra, uint2decstr     # recursive call
        add s0, x0, a0          # s0 = return value from recursive call
skip:
        remu  t1, s1, t0        # finds remainder of v % 10 and saves into t1
        addi t1, t1, '0'        # add value of remainder to the value of '0'
        sb t1, 0(s0)            # save resulting int into s[0]
        sb x0, 1(s0)            # save 0 into s[1]

        addi a0, s0, 1          # return value = s[1]
        lw ra, 0(sp)            # load values of ra, s1, and s0
        lw s1, 4(sp)
        lw s0, 8(sp)
        addi sp, sp, 12         # move stack pointer back to original position
        jr      ra              # return
```