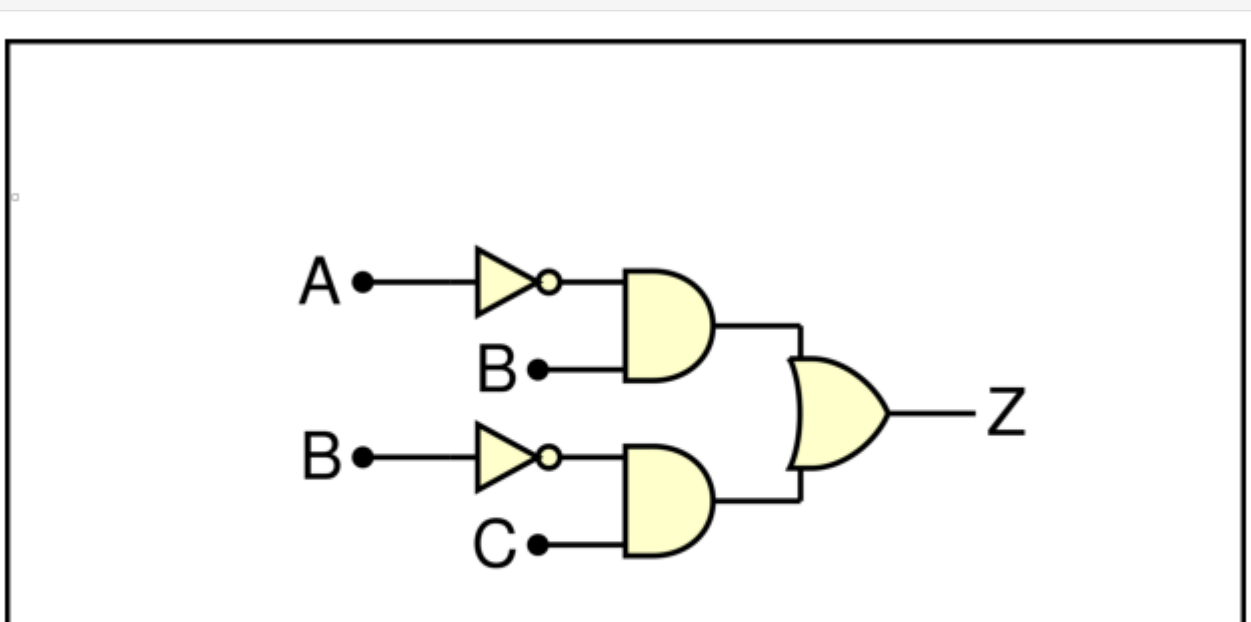
```
In [2]: import wavedrom
import cairosvg
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import numpy as np
from BooleanAlgebra import Solver

class LogicPlotter():
    def my_lines(self,ax, pos, *args, **kwargs):
        if ax == 'x':
            for p in pos:
                plt.axvline(p, *args, **kwargs)
        else:
            for p in pos:
                plt.axhline(p, *args, **kwargs)

    def make_logic(self,circuit):
        svg = wavedrom.render(f"""{circuit}
        """)
        svg["width"] ="2500px"
        svg["height"] ="1500px"
        svg.saveas("/tmp/output.svg")
        cairosvg.svg2png(url='/tmp/output.svg', write_to='/tmp/output.png')
        image = Image.open("/tmp/output.png")
        image = ImageOps.expand(image, border=10, fill='black')
        plt.figure(figsize=(8, 8))  # Adjust the width and height values as desired
        plt.imshow(image)
        plt.axis('off')
        plt.show()

    def make_timing_diagram(self,dict_of_functions):
        plt.rcParams['figure.figsize'] = [15, 10]
        plt.rcParams['figure.dpi'] = 100 # 200 e.g. is really fine, but slower
        fns = dict_of_functions
        fns = {key:np.repeat(fn,1) for (key,fn) in fns.items()}
        fns2 = {key:np.repeat(fn,2) for (key,fn) in fns.items()}
        t = 0.5 * np.arange(len(fns2["A"]))

        lenrange = range(len(fns2["A"]))
        self.my_lines('x',lenrange, color='.5', linewidth=2)
        self.my_lines('y', [-0.5+2*i for i in lenrange], color='.5', linewidth=1)

        for i, (label,fn) in enumerate(fns.items()):
            plt.step(t, np.repeat(fn,2) + i*2, 'r', linewidth = 2, where='post')
            plt.text(-1.75,i*2, label)# Add the letter "a" at position (1, 1)
            for tbit, bit in enumerate(fn):
                plt.text(tbit+0.25, (0.5+2*i), str(bit))

        plt.ylim([-1.5,2*len(fns)])
        plt.xlim([-2,len(fns['A'])])
        plt.yticks([])
        plt.gca().axis('on')
        plt.show()
```

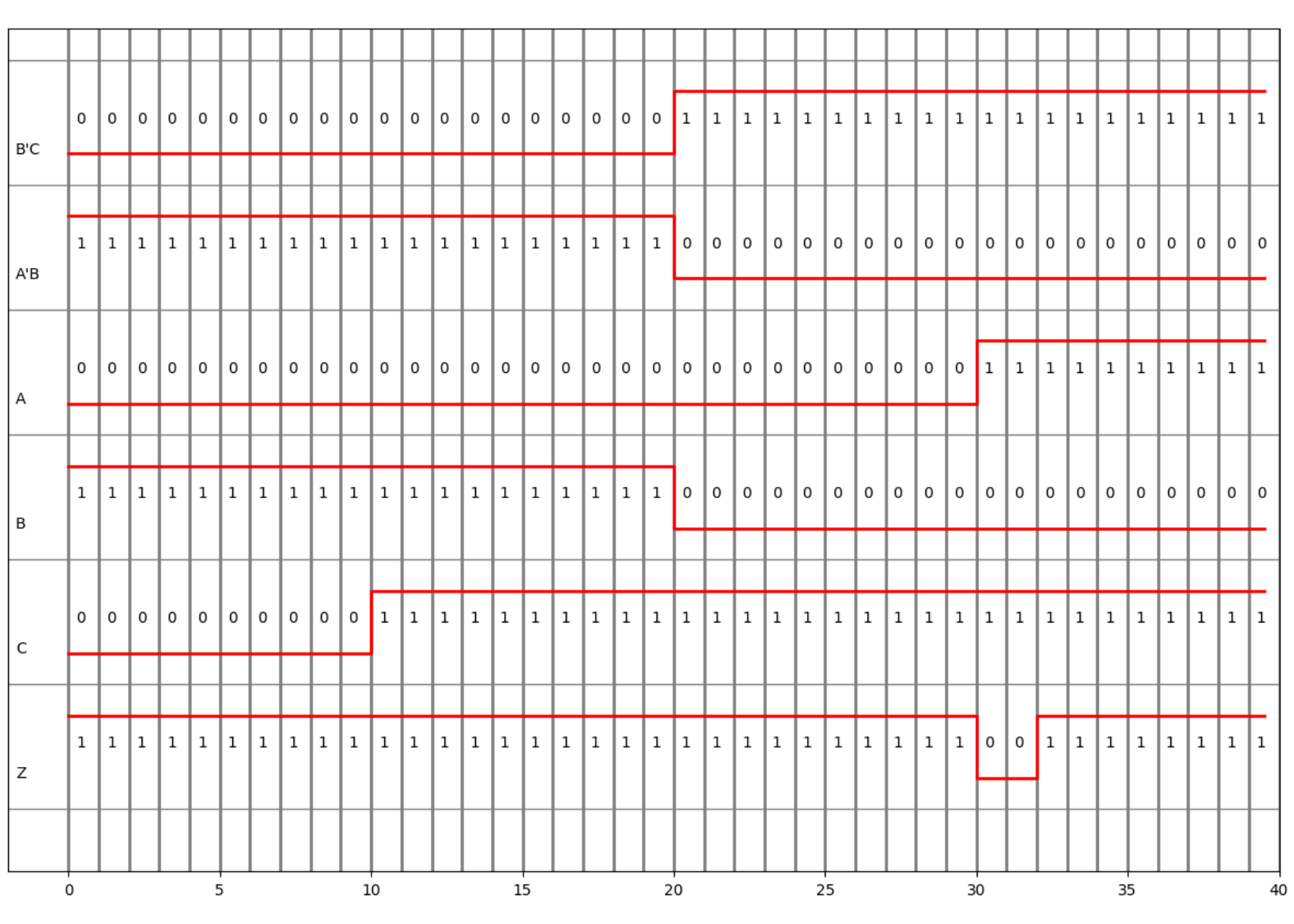
EET3300 Module 7 - Timing

1. Draw a timing diagram for $Z(A,B,C) = (A'B + B'C)$.

- Inverters have a propagation delay of 2 ns
- AND and OR gates have a propagation delay of 5 ns. The input signals are shown in figure 9.5.

```
In [3]: tp = LogicPlotter()
circuit = { "assign":[
      ["Z",
        ["|",
          ["&", ["~", "A"], "B"],
          ["&", ["~", "B"], "C"],
        ]
      ]
    ]}

tp.make_logic(circuit)
fns= {
    "Z":[
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        0,0,1,1,1,1,1,1,1,1,
       ],
    "C":[
        0,0,0,0,0,0,0,0,0,0,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
       ],
    "B":[
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        ],
    "A":[
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        1,1,1,1,1,1,1,1,1,1,
       ],
     }

fns["A\'B"] =   [int(bit) for bit in np.logical_and(np.logical_not(fns['A']), fns['B'])]
fns["B\'C"] =   [int(bit) for bit in np.logical_and(np.logical_not(fns['B']), fns['C'])]
tp.make_timing_diagram(fns)
```

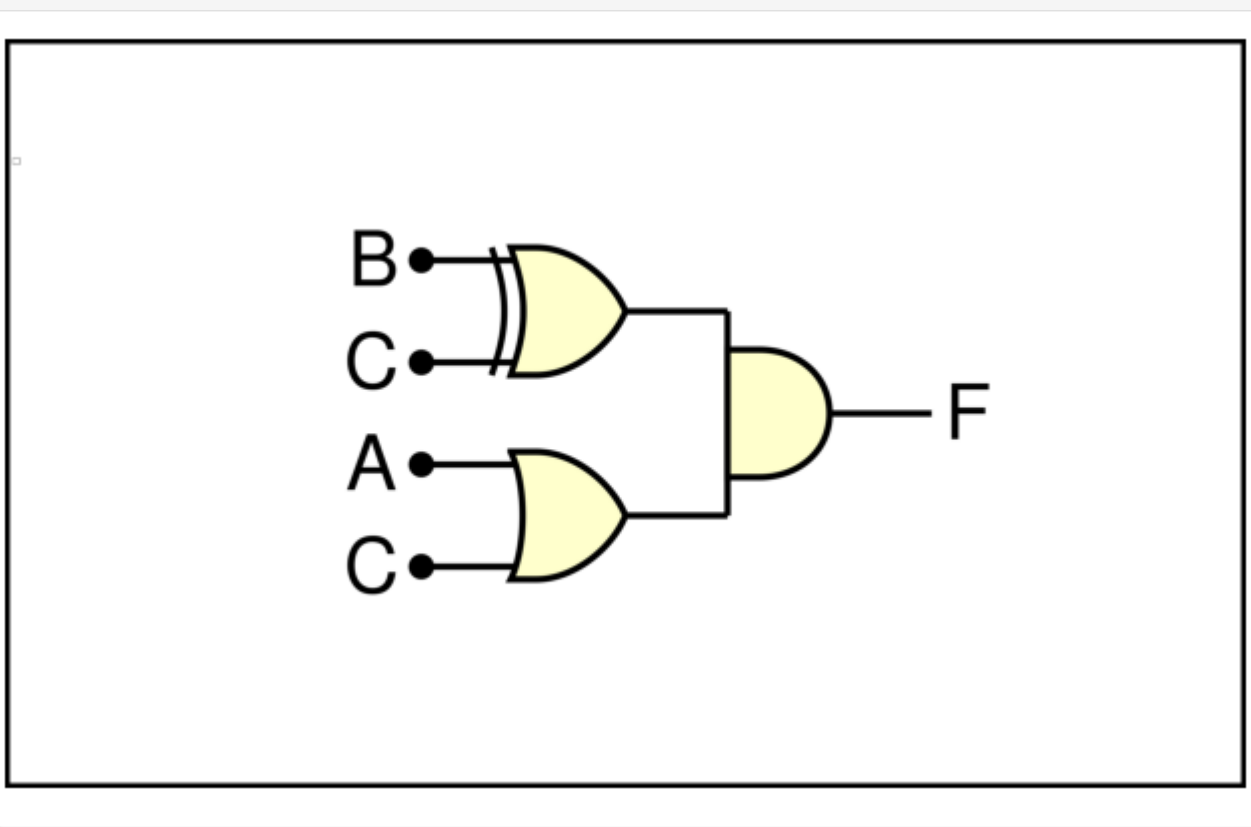


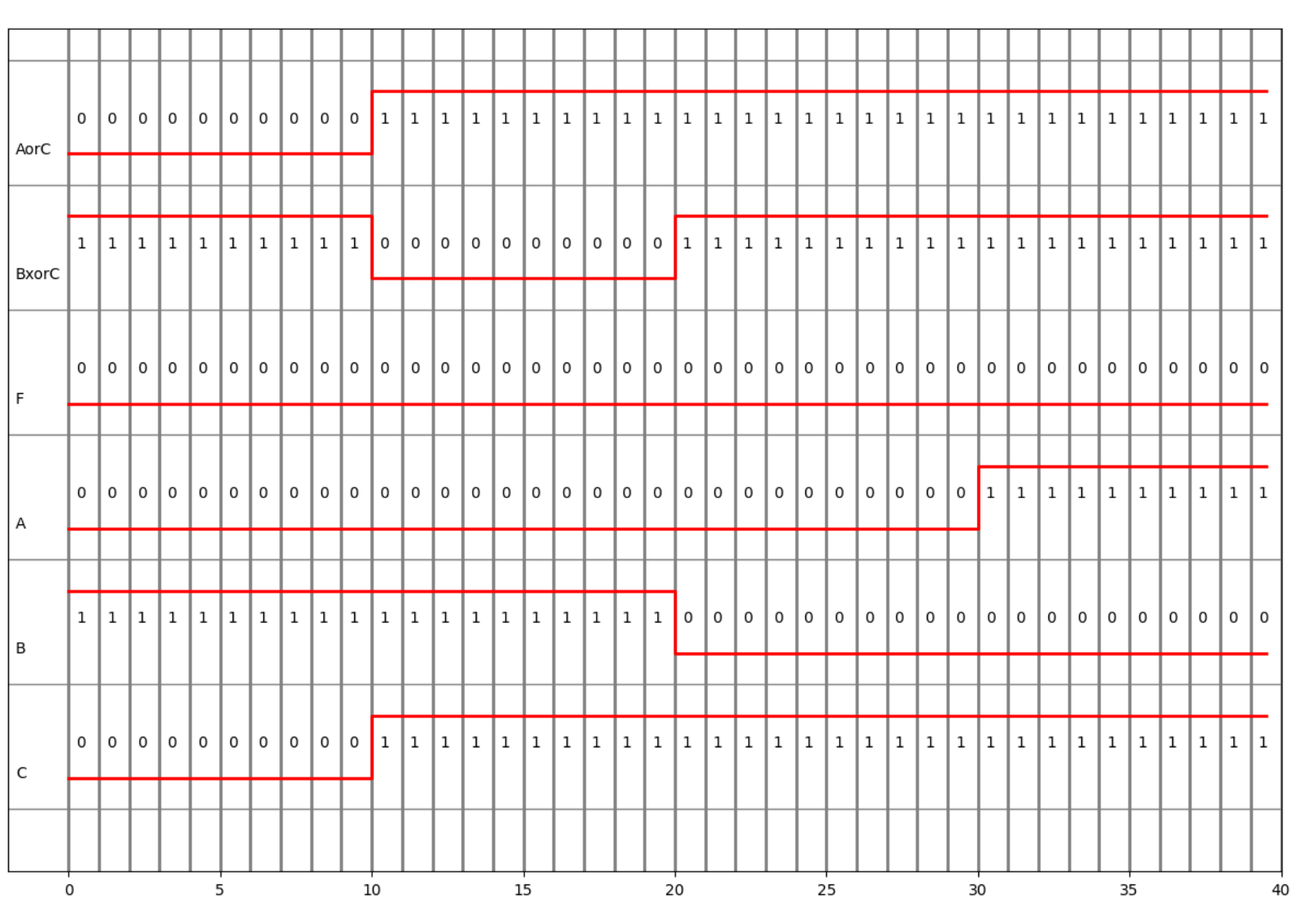


2. Draw a timing diagram for $F(A,B,C) = (B \oplus C)(A + C)$.

- AND gates have a delay of 2 ns
- OR gateshave a delay of 3 ns
- XOR gates have a delay of 5 ns. The input signals are shown in figure 9.5.

```
In [4]: tp = LogicPlotter()
circuit = { "assign":[
      ["F",
        ["&",
          ["^", "B", "C"],
          ["|", "A", "C"],
        ]
      ]
    ]}

fns= {
    "C":[
        0,0,0,0,0,0,0,0,0,0,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
       ],
    "B":[
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        ],
    "A":[
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        1,1,1,1,1,1,1,1,1,1,
       ],
    "F":[
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
       ],
     }
fns['BxorC'] =   [int(bit) for bit in np.logical_xor(fns['B'], fns['C'])]
fns['AorC'] =   [int(bit) for bit in np.logical_or(fns['A'], fns['C'])]
tp.make_logic(circuit)
tp.make_timing_diagram(fns)
```

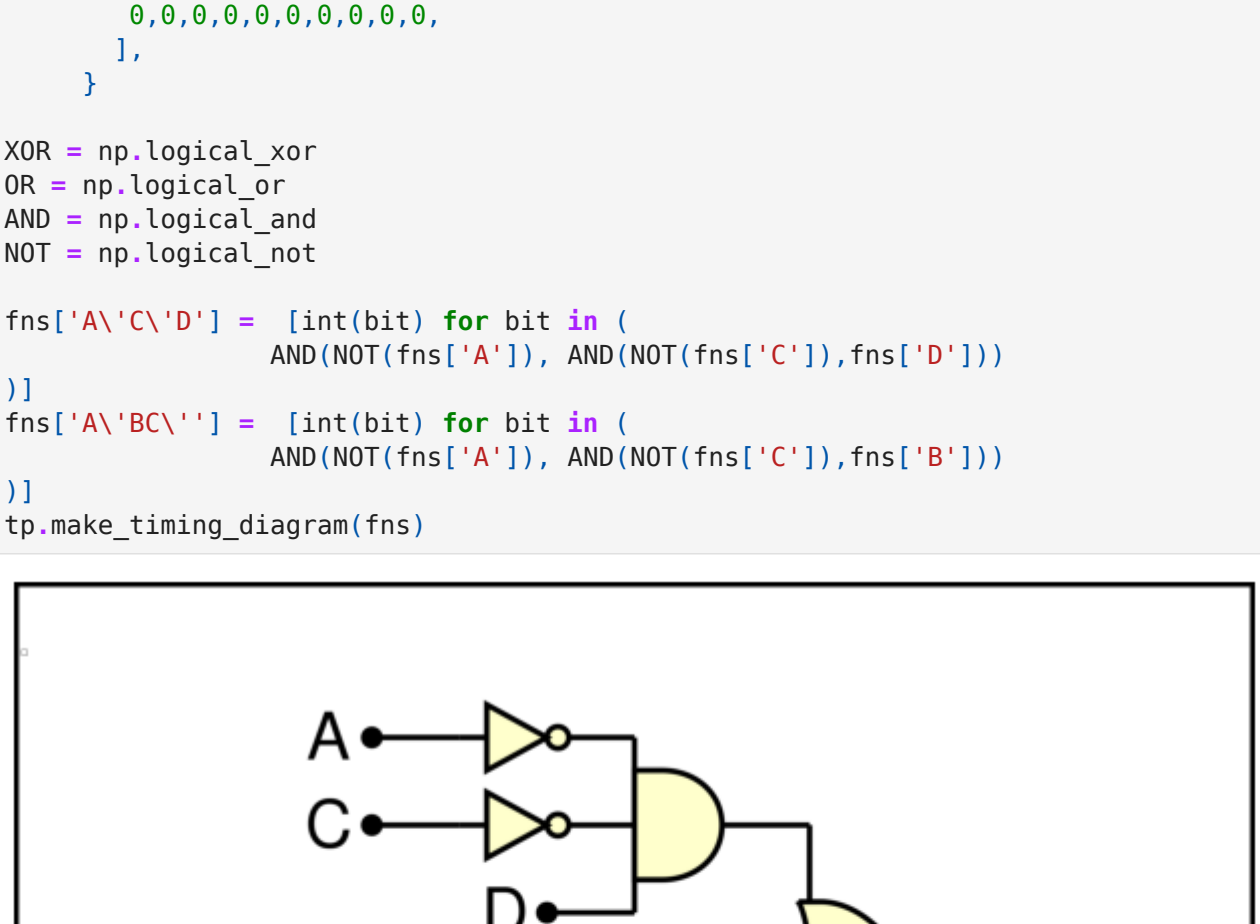


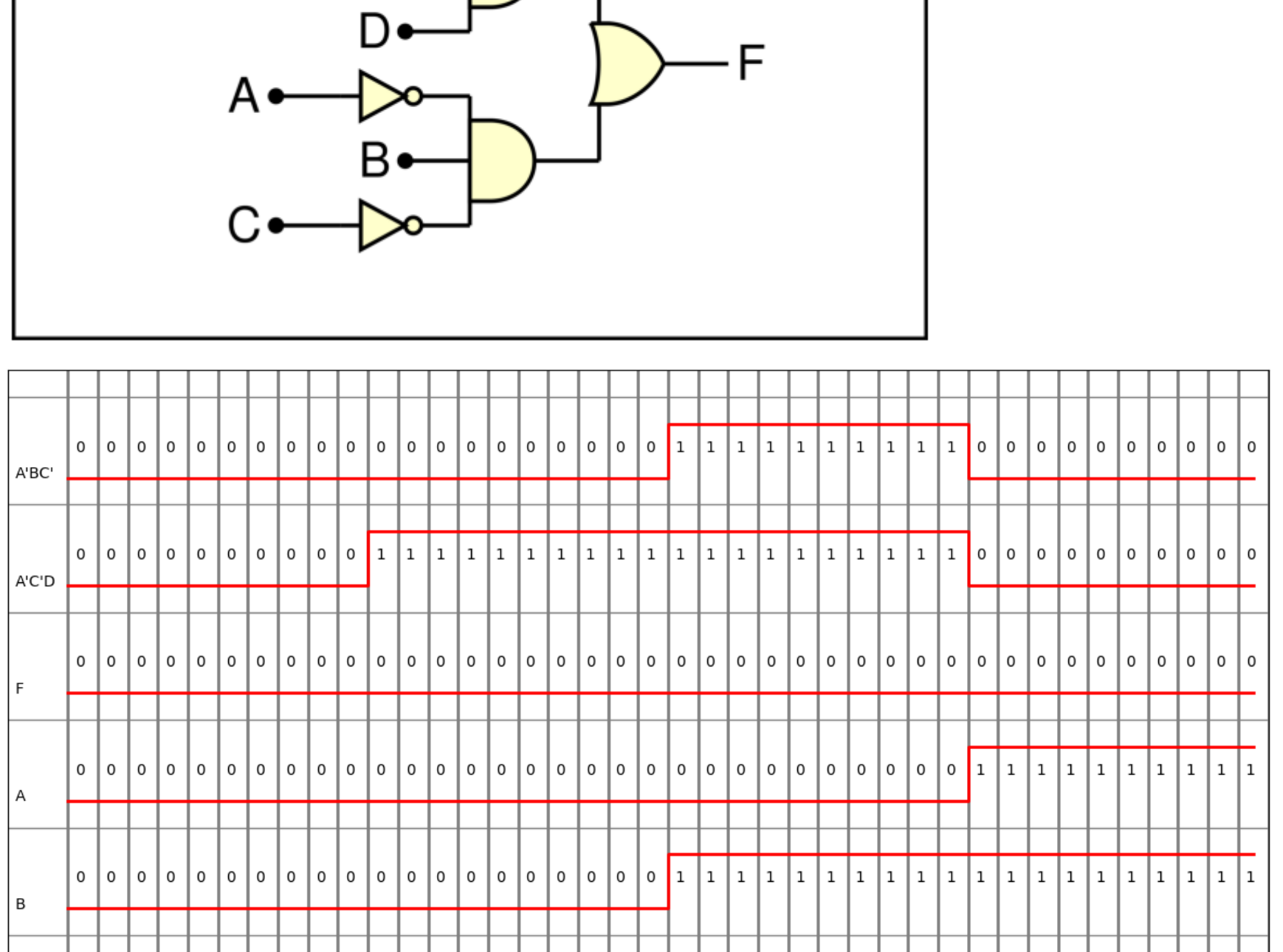


4. Draw a timing diagram for $F(A,B,C,D) = A'C'D + A'BC'$

All logic gates have a propagation delay of 5 ns. The input signals are shown in figure 9.6.

```
In [5]: tp = LogicPlotter()
circuit = { "assign":[
      ["F",
        ["|",
          ["&", ["~","A"], ["~","C"],"D"],
          ["&", ["~","A"],"B" ,["~","C"]],
        ]
      ]
    ]}

tp.make_logic(circuit)

fns= {
    "D":[
        0,0,0,0,0,0,0,0,0,0,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
       ],
    "C":[
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
       ],
    "B":[
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        1,1,1,1,1,1,1,1,1,1,
        1,1,1,1,1,1,1,1,1,1,
        ],
    "A":[
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        1,1,1,1,1,1,1,1,1,1,
       ],
    "F":[
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
       ],
     }

XOR = np.logical_xor
OR = np.logical_or
AND = np.logical_and
NOT = np.logical_not

fns['A\'C\'D'] =  [int(bit) for bit in (
                AND(NOT(fns['A']), AND(NOT(fns['C']),fns['D']))
)]
fns['A\'BC\''] =  [int(bit) for bit in (
                AND(NOT(fns['A']), AND(NOT(fns['C']),fns['B']))
)]
tp.make_timing_diagram(fns)
```





5. Look up the datasheet for the 74LS139A 2 to 4 decoder.

- Although we haven't discussed decoders yet, you will be able to interpret the logic diagram which only uses inverters and NAND gates. Assuming that every logic gate has a delay of 10 ns, draw the output for 1Y 1 giving the input signals shown in figure 9.7

Hazards

1. Find a hazard-free SOP expression for $F(A,B,C) = \Sigma m(2,3,5,7)$.

```
In [9]: from BooleanAlgebra import Solver
Solver(num_vars = 3, fn = [0,0,1,1,0,1,0,1],show_legend=False).display()
```

Truth Table

	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

KMAP

		0	1
		C'	C
00	A'B'	0	0
01	A'B	1	1
11	AB	0	1
10	AB'	0	1

KMAP Legend

		0	1
		C'	C
00	A'B'	0	1
01	A'B	2	3
11	AB	6	7
10	AB'	4	5

use all prime implicants to avoid hazards groups:

(2,3) --> A'B

(11,10)--> AC

(7,3)--> BC

$F = A'B + AC + BC$

3. Find a NAND-only hazard-free expression for $F(A,B,C,D) = \Sigma m(0,1,3,4,6,11) + \Sigma d(8,12,13)$

```
In [10]: Solver(num_vars = 4, fn = [0,1,0,1,1,0,1,0,"x",0,0,1,"x","x",0,0],show_legend=False).display()
```

Truth Table

	A	B	C	D	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	x
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	x
13	1	1	0	1	x
14	1	1	1	0	0
15	1	1	1	1	0

KMAP

		00	01	11	10
		C'D'	C'D	CD	CD'
00	A'B'	0	1	1	0
01	A'B	1	0	0	1
11	AB	x	x	0	0
10	AB'	x	0	1	0

KMAP Legend

		00	01	11	10
		C'D'	C'D	CD	CD'
00	A'B'	0	1	3	2
01	A'B	4	5	7	6
11	AB	12	13	15	14
10	AB'	8	9	11	10

use all prime implicants to avoid hazards

groups (1,3)-->A'B'D

(3,11)--> B'CD

(4,6)-->A'BD'k

(4,12,8)--> C'D'

(12,13) --> ABC'

NAND only so take double complement and apply demorgans to inner complement

$F = A'B'D + B'CD + A'BD' + C'D' + ABC'$

$F = (A'B'D + B'CD + A'BD' + C'D' + ABC')''$

$F = ((ABD')(BC'D')(AB'D)(CD)(A'B'C))'$

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```