EE2703 - Assignment 3

Ata Karim Subhani <EE21B025>

March 1, 2023

1 AIM:

- From a given .net file and .input file of logical circuit, evaluate logical value of rest of the nets using **Topological Sort** and **Event Driven** method
- To estimate the efficiencies of both method.

[48]:
```
import networkx as nx
```

This function takes the **.net** file and return DAG if there is no error in the **.net** file. Otherwise it throws an error message

[61]:
```
def make_graph(netlist):
    """
    netlist : .net file
    return graph
    """
    try:
        g = nx.DiGraph()
        lines = []
        with open(netlist, 'r') as data:
            lines = data.readlines()
        for line in lines:
            line = line.split()
            if(line[1]=='inv'):
                g.add_edges_from([(line[2], line[3])])
                nx.set_node_attributes(g, {line[3]:{"GateType":"INV"}})
            elif(line[1]=='buf'):
                g.add_edges_from([(line[2], line[3])])
                nx.set_node_attributes(g, {line[3]:{"GateType":"BUF"}})
            else:
                g.add_edges_from([(line[2], line[4]), (line[3], line[4])])
                nx.set_node_attributes(g, {line[4]:{"GateType":line[1]}})
        return g
    except:
        return "Error: Node is missing"
```

logical_operation function is to do logical operation.

[62]:
```
def logical_operation(input1, input2, operation):
    """
    input1 and input2 are input to gate.
    opeartion : Logic Operation
    return : Answer after operation on input1 and input2
    """
    if(operation=="and2"):
        return (input1&input2)
    if(operation=="nand2"):
        return (1 - (input1&input2))
    if(operation=="or2"):
        return (input1|input2)
    if(operation=="xor2"):
        return (input1^input2)
    if(operation=="nor2"):
        return (1 - (input1|input2))
    if(operation=="xnor2"):
        return (1 - (input1^input2))
```

[63]:
```
def format_output_file(netlist_file, output_file):
    """
    netlist_file : .net file to make graph
    output_file : file to which net value to be written
    return : A file format i.e in the first line, net names are written in␣
 ↪sorted order
    """
    graph = make_graph(netlist_file)
    open(output_file, "w") # it will clear the junk data if there is any.
    nets = []
    for ele in graph.nodes():
        nets.append(ele)
    nets.sort()
    for ele in nets:
        with open(output_file, 'a') as OutputFile:
            OutputFile.write(f"{ele:<8} ")
    file = open(output_file, 'a')
    file.write(f'\n')
    file.close()
```

1.1 Topological Sort Method

This **solve1** function is called for the logical operation and writing the output to the file when we used **Topological Sort** method to solve the digital circuit.

[64]:
```
def solve1(netlist_file, input_value, output_file):
    """
    This function is used to solve digital circuit using Topological sort method.
    netlist_file : .net file in which logic circuit discription is given
    input_value : It is a list of dictionary where each dictionary values is␣
↪logic of each primary input with key as net name.
    output_file : file where logic of each net to be written finally.
    return : An output_file
    """
    graph = make_graph(netlist_file)
    for graphs in graph.nodes(data=True):
        if(len(graphs[1])==0):
            graphs[1]["GateType"] = "PI"
    nets = [ele for ele in graph.nodes()]

    for net_value in input_value: # Here net_value is dictionary
        for node in list(nx.topological_sort(graph)):
            if(graph.nodes[node]["GateType"]=="PI"):
                graph.nodes[node]["Value"] = int(net_value[node])

            elif(graph.nodes[node]["GateType"]=="BUF"):
                net = list(graph.predecessors(node))[0]
                graph.nodes[node]["Value"] = int((net_value[net]))
                net_value[node] = graph.nodes[node]["Value"]

            elif(graph.nodes[node]["GateType"]=="INV"):
                net = list(
                    graph.predecessors(node))[0]
                graph.nodes[node]["Value"] = (1-int(net_value[net]))
                net_value[node] = graph.nodes[node]["Value"]

            else:
                net1 = list(graph.predecessors(node))[0]
                net2 = list(graph.predecessors(node))[1]
                graph.nodes[node]["Value"] =␣
↪logical_operation(int(net_value[net1]), int(net_value[net2]), graph.
↪nodes[node]["GateType"])
                net_value[node] = graph.nodes[node]["Value"]
        # Write to the file
        for ele in nets:
            with open(output_file, 'a') as f:
                f.write(f'{graph.nodes[ele]["Value"]:<8} ')
        f = open(output_file, 'a')
        f.write(f"\n")
        f.close()
```

This function will throw an error message if the given circuit is cyclic.

[65]:
```
def solve_digital_circuit_using_TopoplogicalSort(netlist_file, input_file,␣
↪output_file):
    """
    It takes .net file and input_file and finally write the final net values in␣
↪output_file
    using solve1() function.
    """
    graph = make_graph(netlist_file)
    if(type(graph)==str):
        return graph
    else:
        try:
            topo = list(nx.topological_sort(graph))
            inputs = []
            with open(input_file, 'r') as input:
                inputs = input.readlines()
            format_output_file(netlist_file, output_file)

            nets = [ele for ele in graph.nodes()]
            # Make list of dictionary to store the inputs.
            input_value = [] # It is a list of dictionary.
            for ele in inputs[1:]:
                dict = {}
                for i in range(len(ele.split())):
                    dict[inputs[0].split()[i]] = int(ele.split()[i])
                input_value.append(dict)
            solve1(netlist_file,input_value,output_file)
        except:
            print(f"Given logic circuit is cyclic")
```

1.2 Event Driven Simulation

This **solve2** function is called for the logical operation and writing the output to the file when we used **Event Driven Simulation** method to solve the digital circuit.

[66]:
```
def solve2(netlist_file, input_value, inputs, output_file):
    """
    netlist_file :  .net file describing the logical circuit
    input_value : It is a list of dictionary. Key is primary net name. values is␣
↪a list representing
                  values of the net and a flag indicating if the net logic has␣
↪changed from the previous
                  input set or not
    inputs  : It is a list of inputs
    output_file : destination file for the output
    return : output_file
    """
    graph = make_graph(netlist_file)
    for node in graph.nodes:
        graph.nodes[node]["Value"] = "x"

    nets = [ele for ele in graph.nodes()]

    for input_set in input_value: # here input_set is dictionary with key as PI␣
↪and value as List([input, True/False])
        queue = []
        for primary_input in (inputs[0].split()):
            if(input_set[primary_input][0]=="True"):
                queue.append(primary_input)
        while(len(queue)>0):
            node = queue.pop()
            a = list(graph.successors(node))
            temp  = graph.nodes[node]["Value"]
            b = list(graph.predecessors(node))
            if(len(b)==0): # means node is from PI
                graph.nodes[node]["Value"] = int(input_set[node][1])
                for ele in a:
                    queue.insert(0, ele)
            elif(len(b)==1 and graph.nodes[node]["GateType"]=="BUF"): # node is␣
↪buffer's output
                if(graph.nodes[b[0]]["Value"]=="x"): # parent is not known
                    continue
                else:
                    graph.nodes[node]["Value"] = int(graph.nodes[b[0]]["Value"])
                    if(graph.nodes[node]["Value"]!= temp):
                        for ele in a:
                            queue.insert(0, ele)
            elif(len(b)==1 and graph.nodes[node]["GateType"]=="INV"): # node is␣
↪Inverter's output
                if(graph.nodes[b[0]]["Value"]=="x"):
                    continue
                else:
                    graph.nodes[node]["Value"] = 1-int(graph.
↪nodes[b[0]]["Value"])
                    if(graph.nodes[node]["Value"]!= temp):
                        for ele in a:
                            queue.insert(0, ele)
            else:
                if(graph.nodes[b[0]]["Value"]=="x" or graph.
↪nodes[b[1]]["Value"]=="x"):
                    continue
                else:
                    y = [graph.nodes[ele]["Value"] for ele in b]
                    gate = graph.nodes[node]["GateType"]
                    graph.nodes[node]["Value"] = logical_operation(int(y[0]),␣
↪int(y[1]), gate)
                    if(graph.nodes[node]["Value"]!= temp):
                        for ele in a:
                            queue.insert(0, ele)
        # Write to the file
        for ele in nets:
            with open(output_file, 'a') as f:
                f.write(f'{graph.nodes[ele]["Value"]:<8} ')
        f = open(output_file, 'a')
        f.write(f"\n")
        f.close()
```

This function will throw an error message if the given circuit is cyclic.

[67]:
```
def event_driven_simulation(netlist_file, input_file, output_file):
    """
    It takes .net file and input_file and finally write the final net values in␣
↪output_file
    using solve2() function.
    """
    graph = make_graph(netlist_file)
    if(type(graph)==str):
        return graph
    else:
        try:
            topo = list(nx.topological_sort(graph))
            for graphs in graph.nodes(data=True):
                if(len(graphs[1])==0):
                    graphs[1]["GateType"] = "PI"

            format_output_file(netlist_file, output_file)

            inputs = [] # inputs is a list of string
            with open(input_file, 'r') as input:
                inputs = input.readlines()

            input_value = [] # It is a list of dictionary.
            for k, ele in enumerate(inputs[1:]):
                dict = {}
                if(k==0):
                    for i in range(len(ele.split())):
                        dict[inputs[0].split()[i]] = ["True", int(ele.
↪split()[i])]
                else:# check if the input changes or not
                # True means we need to add that inputs in queue
                # False means we don't need to add that inputs in queue
```

```
                    for i in range(len(ele.split())):
                        if(inputs[k].split()[i]!=ele.split()[i]): # input changes
                            dict[inputs[0].split()[i]] = ["True", int(ele.
↪split()[i])]
                        else: # When input doesn't changes
                            dict[inputs[0].split()[i]] = ["False", int(ele.
↪split()[i])]
                input_value.append(dict)

            solve2(netlist_file, input_value, inputs, output_file)
        except:
            print(f"Given logic circuit is cyclic")
```

1.3 Writing to output file

There are 4 blocks. To generate the output to a particular net comment out the corresponding block and comment in the rest of the 3 blocks.

[70]:
```
# netlist_file = 'c8.net'
# input_file = 'c8.inputs'
# output_fileT = "c8T.outputs"
# output_fileE = "c8E.outputs"

# netlist_file = 'parity.net'
# input_file = 'parity.inputs'
# output_fileT = "parityT.outputs"
# output_fileE = "parityE.outputs"

# netlist_file = 'c17.net'
# input_file = 'c17.inputs'
# output_fileT = "c17T.outputs"
# output_fileE = "c17E.outputs"


# netlist_file = 'c432.net'
# input_file = 'c432.inputs'
# output_fileT = "c432T.outputs"
# output_fileE = "c432E.outputs"

netlist_file = 'c17_1.net'
input_file = 'c17_1.inputs'
output_fileT = "c17_1T.outputs"
output_fileE = "c17_1E.outputs"
```

[71]:
```
solve_digital_circuit_using_TopoplogicalSort(netlist_file, input_file,␣
 ↪output_fileT)
event_driven_simulation(netlist_file, input_file, output_fileE)
```

```
Given logic circuit is cyclic
Given logic circuit is cyclic
```

1.4 Check if both method gives same output or not

Here if the output of any net is different the below code will print "False". You can check it by running for different .net file that nothing will be printed which means that indeed, both method gives the same result.

[58]:
```
def check(output_fileE, output_fileT):
    T = open(output_fileT, 'r')
    E = open(output_fileE, 'r')
    T = T.readlines()
    E = E.readlines()
    for i in range(1, len(T)):
        for j in range(len(E[0].split())):
            if(T[i].split()[j]!=E[i].split()[j]):
                print("False")
check(output_fileE, output_fileT)
```

1.5 Time Comparision

[59]:
```
netlist_file = 'test.net'
input_file = 'test.inputs'
output_fileT = "testT.outputs"
output_fileE = "testE.outputs"

%timeit solve_digital_circuit_using_TopoplogicalSort(netlist_file, input_file,␣
 ↪output_fileT)

%timeit event_driven_simulation(netlist_file, input_file, output_fileE)
```

```
29.2 ms ± 608 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
28.7 ms ± 1.91 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[43]:
```
netlist_file = 'c8.net'
input_file = 'c8.inputs'
output_fileT = "c8T.outputs"
output_fileE = "c8E.outputs"

%timeit solve_digital_circuit_using_TopoplogicalSort(netlist_file, input_file,␣
 ↪output_fileT)

%timeit event_driven_simulation(netlist_file, input_file, output_fileE)
```

```
30.5 ms ± 2.44 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
28.9 ms ± 920 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[45]:
```
netlist_file = 'parity.net'
input_file = 'parity.inputs'
output_fileT = "parityT.outputs"
output_fileE = "parityE.outputs"

%timeit solve_digital_circuit_using_TopoplogicalSort(netlist_file, input_file,␣
 ↪output_fileT)

%timeit event_driven_simulation(netlist_file, input_file, output_fileE)
```

```
7.85 ms ± 407 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
7.12 ms ± 182 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[46]:
```
netlist_file = 'parity.net'
input_file = 'parity.inputs'
output_fileT = "parityT.outputs"
output_fileE = "parityE.outputs"

%timeit solve_digital_circuit_using_TopoplogicalSort(netlist_file, input_file,␣
 ↪output_fileT)

%timeit event_driven_simulation(netlist_file, input_file, output_fileE)
```

```
7.09 ms ± 231 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
6.97 ms ± 291 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

1.5.1 Inferences

: From the above time output we can see that time taken by topological sort is little more than than time taken by event driven method. This is because in Topological sort method, for each set of Primary input, all the nets value are calculated each time, but in later case we calculate the net value of only those nets which depends on primary input which has changed.
Furthermore, this difference will increase if number of nets will increase and at the same time only few primary inputs will change in any interval of time.
This happens because as there are very few changes in primary inputs, there are few nets which need to be evaluated again but in former case we will evaluate all the nets value every time even when there is only one changes in input primary side. So, **Topological sort** method is in general **<u>less efficient</u>** than the **Event Driven Method**