**EC-410 Digital System Design**

**Assignment 3**

**Due Date:** 31 December 2024, 0900 hrs. Submit on LMS.
Note: Plagiarism will lead to zero marks.

### Objective: Digital Circuit Critical Path Analysis

Write a Python/C++ program that analyzes digital circuits to find their critical path - the longest combinational path that determines the maximum operating frequency of the circuit.

### Problem Description

Your program should analyze both combinational and sequential circuits to determine their critical paths. The critical path is defined as the longest path through combinational logic elements between:

- Primary inputs and primary outputs (for combinational circuits)
- Primary inputs to registers, register to register, or registers to primary outputs (for sequential circuits)

### Input Format

Your program should read a circuit description from a text file with the following format:

```
# Circuit name
# Format: <node_type> <node_id> <input_nodes...>

INPUT in1

INPUT in2

ADD add1 in1 in2

MUL mul1 in1 add1

REG reg1 mul1

ADD add2 reg1 in2

OUTPUT out1 add2
```

### Component Delays

Use the following delay values for your calculations:

- Adder (ADD): 1.0 time units
- Multiplier (MUL): 1.0 time units

- Register (REG): 0.2 time units
- Multiplexor: 1 time units
- Any other structure: assume reasonable values

**Example Usage of the Program**

```
def main():

  # Read circuit description

 cir_names = [ "cir1.txt", "cir2.txt", "cir3.txt", "cir4.txt", "cir5.txt"]
 for cur_ circuit in names:
        circuit_graph = parse_circuit(cur_circuit)

       # Find critical path

       critical_path, total_delay = find_critical_path(circuit_graph)

       # Print results

       print(f"Critical Path: {' -> '.join(critical_path)}")

       print(f"Total Delay: {total_delay:.2f} time units")
```

**Sample Output**

```
________________________________________________

Circuit name: adder1

Critical Path: in1 -> add1 -> mul1 -> reg1 -> add2 -> out1

Path Components:

- ADD (add1): 1.0 tu

- MUL (mul1): 1.0 tu

- REG (reg1): 0.2 tu

- ADD (add2): 1.0 tu

Total Delay: 3.2 time units
```

**Required Output**

Your program should output:

1. The critical path as a **sequence of node IDs**
2. The **total delay** along the critical path
3. **List** of all components in the critical path with their individual delays
4. **Visualize** the circuit and critical path using a library like NetworkX

**Implementation Requirements**

1. Use appropriate data structures (e.g., dictionaries, graphs) to represent the circuit
2. Implement proper error handling for invalid input files
3. Use object-oriented programming principles where appropriate
4. Include comments and documentation
5. Follow Python PEP 8 style guidelines

**Submission Requirements**

1. Source code files (.py)
2. Example circuit files (.txt)
3. README file with:
    - Installation instructions
    - Usage instructions
    - Example inputs and outputs
    - Design decisions and assumptions
4. Unit tests on 5 circuits, 3 are given below. Choose two circuits on your own

**Grading Criteria**

- Correct critical path identification (40%)
- Unit tests on 5 circuits (20%)
- Visualization of circuit graph (30%)
- Documentation and comments (10%)

**Hints: Functions you can create.**

1. parse_circuit(filename: str) -> Dict:
    - Parse the circuit description file
    - Create a graph representation of the circuit
    - Return the graph data structure
2. identify_node_type(node_id: str, graph: Dict) -> str:
    - Determine if a node is input, output, register, logic gate, or a computational structure, like adder, multiplier etc
    - Return the node type as a string
3. find_critical_path(graph: Dict) -> Tuple[List[str], float]:
    - Find the longest path in terms of accumulated delay
    - Return both the path (as a list of node IDs) and total delay
4. calculate_path_delay(path: List[str], graph: Dict) -> float:

- Calculate the total delay along a given path
- Account for different component delays

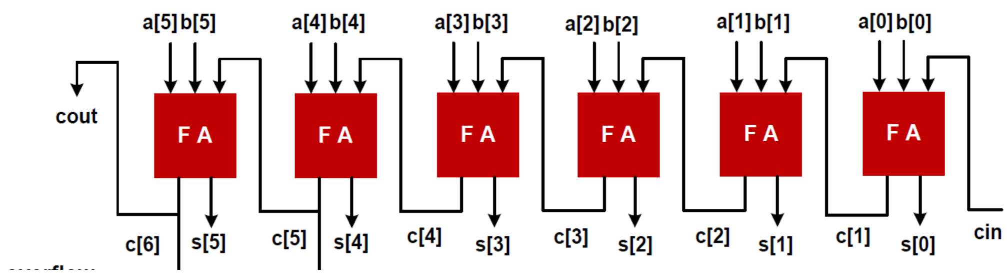
**Other Hints**

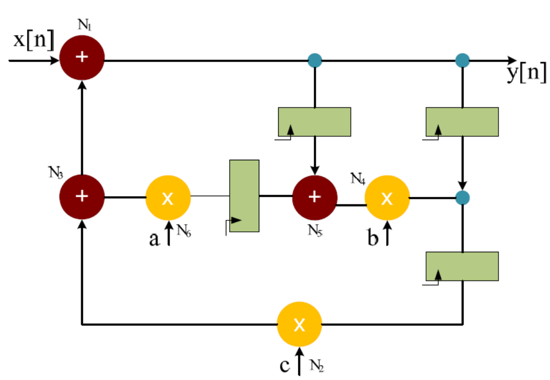
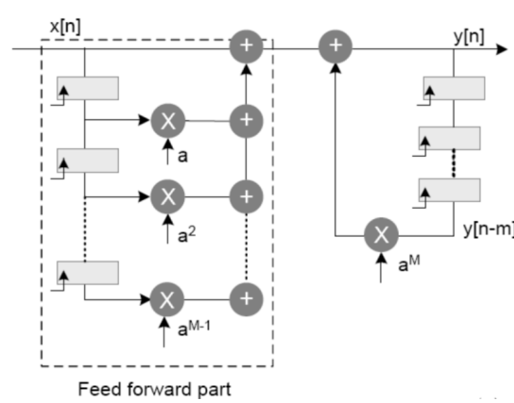
1. Consider using a topological sort to process nodes in the correct order
2. Use dynamic programming to avoid recalculating delays for shared paths
3. Pay special attention to paths through registers in sequential circuits
4. Consider using Python's networkx library for graph operations
5. Test your code with both simple circuit configurations first

**Circuits to consider**

Three circuits are given as examples, choose total of 5 circuits of you choice from book, slides, and any other resources.

**CIR_YAS1**




| **CIR_YAS2** | **CIR_YAS3 (assumer M=3)** |
|---|---|
|  |  |