Danmarks
Tekniske
Universitet

02135 Introduction to Cyber System - Assignment 1

FSMD simulator in Python

Authors

Group 8
Jianan Xu - s204698
Rune Bjerre Clausen- s204702
Simona Tican - s204703

September 25, 2021

1 Introduction

The purpose of this paper is to show the testing of a general purpose FSMD simulator with three separate and simple tests, two of which is based around finding the greatest common divisor with different methods. The simulator is realized in python and takes XML files as inputs. Comments in the simulator script give some explanations of the implemented functions used to handle the XML file inputs for the tests.

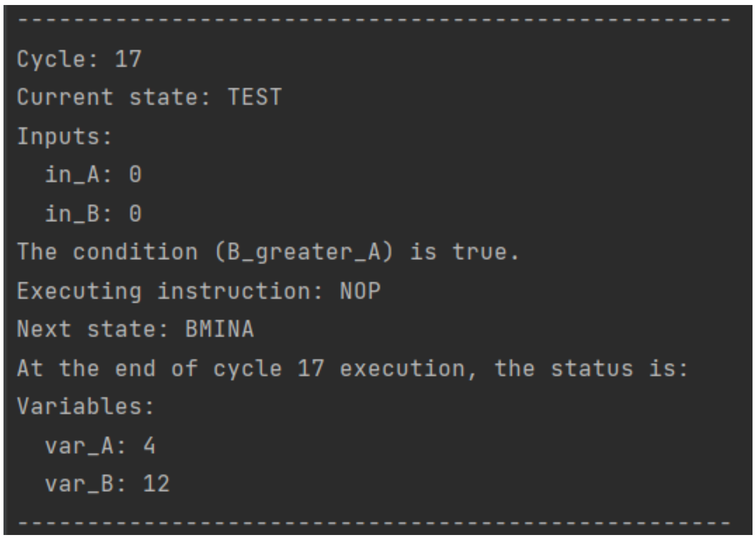
Test 1 and test 2 were given as a framework to build the simulator on, while test 3 was thought up after the simulator script was completed, and is meant to provide a more complex, while still quite simple, final test for the created simulator. It is also meant to provide understanding of how to write and parse XML file inputs.

2 Implementation of FSMD simulator

The simulator is cycle-based, which means that in each cycle a single transition of FSMD is performed. It can be implemented by an outer `for` loop, that tracks one cycle each iteration between the 0th cycle and the maximum cycle. The loop continues until the maximum cycle, unless when the `endstate` defined in `<stimuli_file>` is reached before the maximum cycle. If so, the variable `repeat` will be updated with *False*. The boolean of `repeat` will be checked at the end of each loop. If `repeat` is *False*, jump out of the outer `for` loop.

Some information for each cycle will be printed. An example of the terminal output is shown in Figure1. A state-transition table is a good way to specify a finite-state machine. Only deterministic FSMD is considered in this assignment, that means there is only one path for specific condition and instruction from the current state to the next state. Thus, one of the core snippets is to find the condition and instruction of current state, then map them and current state to the next state. It was realized by nesting a inner `for` loop inside the outer `for` loop. When traversing the transition(s) of current state by the inner `for` loop, there are two cases should be considered. One is if there is only one transition in current state, and the other is if there are more than one transition in current state. Also, the values of the variables can be updated by calling the function `execute_instruction`(*instruction*) in the inner `for` loop. Here is the inner `for` loop, which can be found in `fsmd-sim.py` file.

```
for transition in fsmd[state]:
    # Only one element
    if type(transition) is str:
        ...
    # More than one element
    elif evaluate_condition(transition['condition']):
        ...
```



```
--------------------------------------------------
Cycle: 17
Current state: TEST
Inputs:
  in_A: 0
  in_B: 0
The condition (B_greater_A) is true.
Executing instruction: NOP
Next state: BMINA
At the end of cycle 17 execution, the status is:
Variables:
  var_A: 4
  var_B: 12
--------------------------------------------------
```

Figure 1: An example of terminal output from TEST2 17th cycle

3 Test 1

The first test of a Mealy FSMD is based on a very simple operation. It's desirable output is the first variable `var_A` being equal to the second one `var_TH`. The program takes no input and operates the comparison operators in the state `COMPUTE`. Once the variables are equal to each other, the program will execute no operation, and then the state `DONE` is reached. In the case where `var_A` is greater than `var_TH`, the program decreases `var_A` with 1. The result becomes new `var_A` so that the next state is operating the comparison operators again, until it reaches the final desirable output. In the case where `var_TH` is greater than `var_A`, the program increases `var_A` with 1, similarly, the next state operates the comparison operators on the new `var_A` and the `var_TH`. Because there is no `endstate` defined in `<description_file>`, so the simulator will terminate only when the cycle counter reaches `<max_cycles>`.

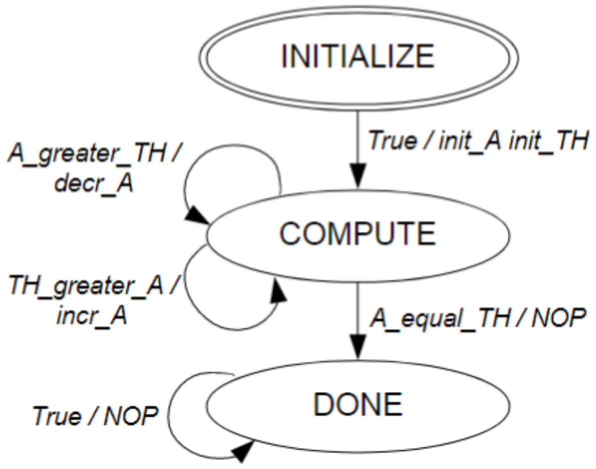


Figure 2: State graph of TEST 1

Present state	Next state	Condition	Instruction
INITIALIZE	COMPUTE	True	init_A init_TH
COMPUTE	DONE	A_equal_TH	NOP
	COMPUTE	A_greater_TH	decr_A
	COMPUTE	TH_greater_A	incr_A
DONE	DONE	True	NOP

Table 1: Transition table of TEST 1

4 Test 2

The second test of Moore style FSMD was a predefined test. It is a simple FSMD that computes the greatest common divisor from two inputs, `in_A` and `in_B`, which will be assigned to `var_A` and `var_B`. As seen in the state graph as well as the transition table it works around the central stage `TEST`. Here we test the relationship between `var_A` and `var_B` - i.e are they equal or is one greater than the other. Depending on the relationship the simulator enters one of three new possible states; if they are equal, it enters the `FINISH` state and the greatest common divisor is found. If one is greater than the other, it enters `AMINB`/`BMINA`, with the lesser is subtracted from the greater, and the result is then substituted in as the new value of the greater. This cycle runs till the two variables become equal, and then the `FINISH` state is reached. All way through the simulation the `var_A` and `var_B` are reported, and when they are equal, that value is the greatest common divisor of the initial inputs. But the simulator will terminate after the maximum cycle even though it does not find the greatest common divisor.

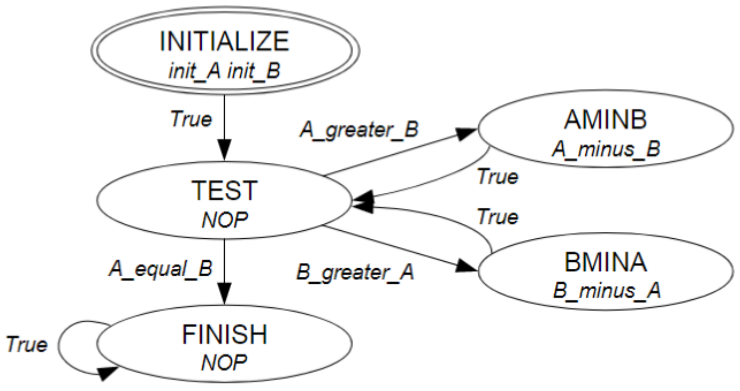


Figure 3: State graph of TEST 2

Present state	Next state	Condition	Instruction
INITIALIZE	TEST	True	init_A init_B
TEST	AMINB	A_greater_B	NOP
	BMINA	B_greater_A	NOP
	FINISH	A_equal_B	NOP
AMINB	TEST	True	A_minus_B
BMINA	TEST	True	B_minus_A
FINISH	FINISH	True	NOP

Table 2: Transition table of TEST 2

5 Test 3

The third test of Moore style FSMD is another way to find the greatest common divisor of two natural numbers by FSMD simulator. The two natural numbers, `in_A` and `in_B`, is provided by the `<stimuli_file>`. Similarly, they will be assigned to `var_A` and `var_B` respectively, which will change as the simulator cycles and will be compared in `TEST` state. If `var_A` and `var_B` are identical, that is `A_equal_B`, the `FINISH` state will be reached directly, because the greatest common divisor is the number itself. Otherwise, the greater number(divisible) is divided by the smaller one(divisor) in state `ADIVB`/`BDIVA`. If the remainder is not zero, the smaller number(divisor) becomes the new divisible, and the remainder is the new divisor. The division repeats until the remainder is zero. If so, the `FINISH` state is reached, and the greatest common divisor is the divisor of the last division. The simulator terminates when it reaches the `FINISH` state. Otherwise, it terminates when the cycle counter reaches the value specified as `<max_cycles>`.

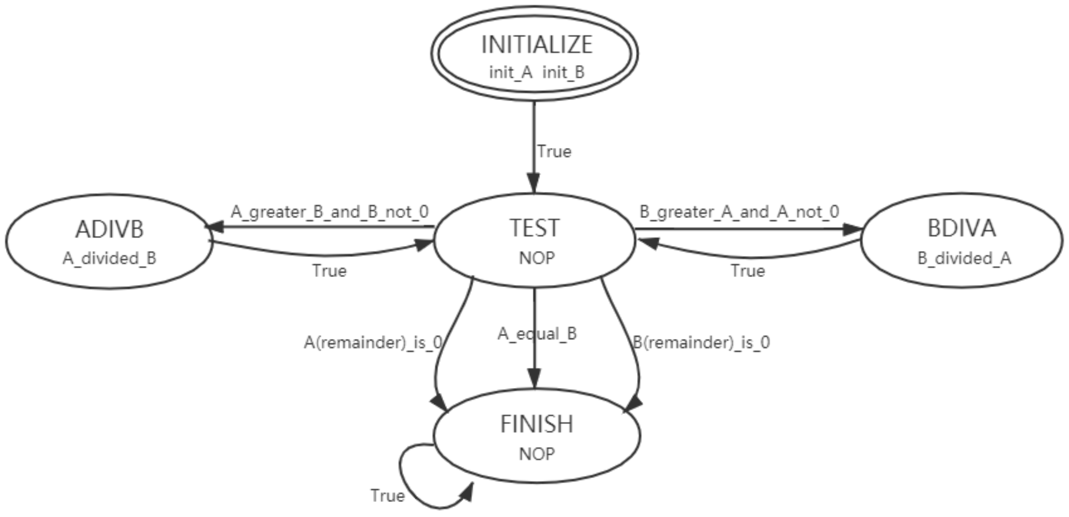


Figure 4: State graph of TEST 3

Present state	Next state	Condition	Instruction
INITIALIZE	TEST	True	init_A init_B
TEST	FINISH	A(remainder)_is_zero	NOP
	FINISH	B(remainder)_is_zero	NOP
	FINISH	A_equal_B	NOP
	ADIVB	A_greater_B_and_B_not_0	NOP
	BDIVA	B_greater_A_and_A_not_0	NOP
ADIVB	TEST	True	A_divided_B
BDIVA	TEST	True	B_divided_A
FINISH	FINISH	True	NOP

Table 3: Transition table of TEST 3

6 Conclusion

FSMD's are a valuable tool for completing tasks through computations. This exercise gave a good baseline understanding of why, and how to implement it. Even though the simulation here was on simple tests, it opens the door for more complex use of an FSMD in our further studies.