CSE 331

HW_BONUS

Burak Kocausta

1901042605

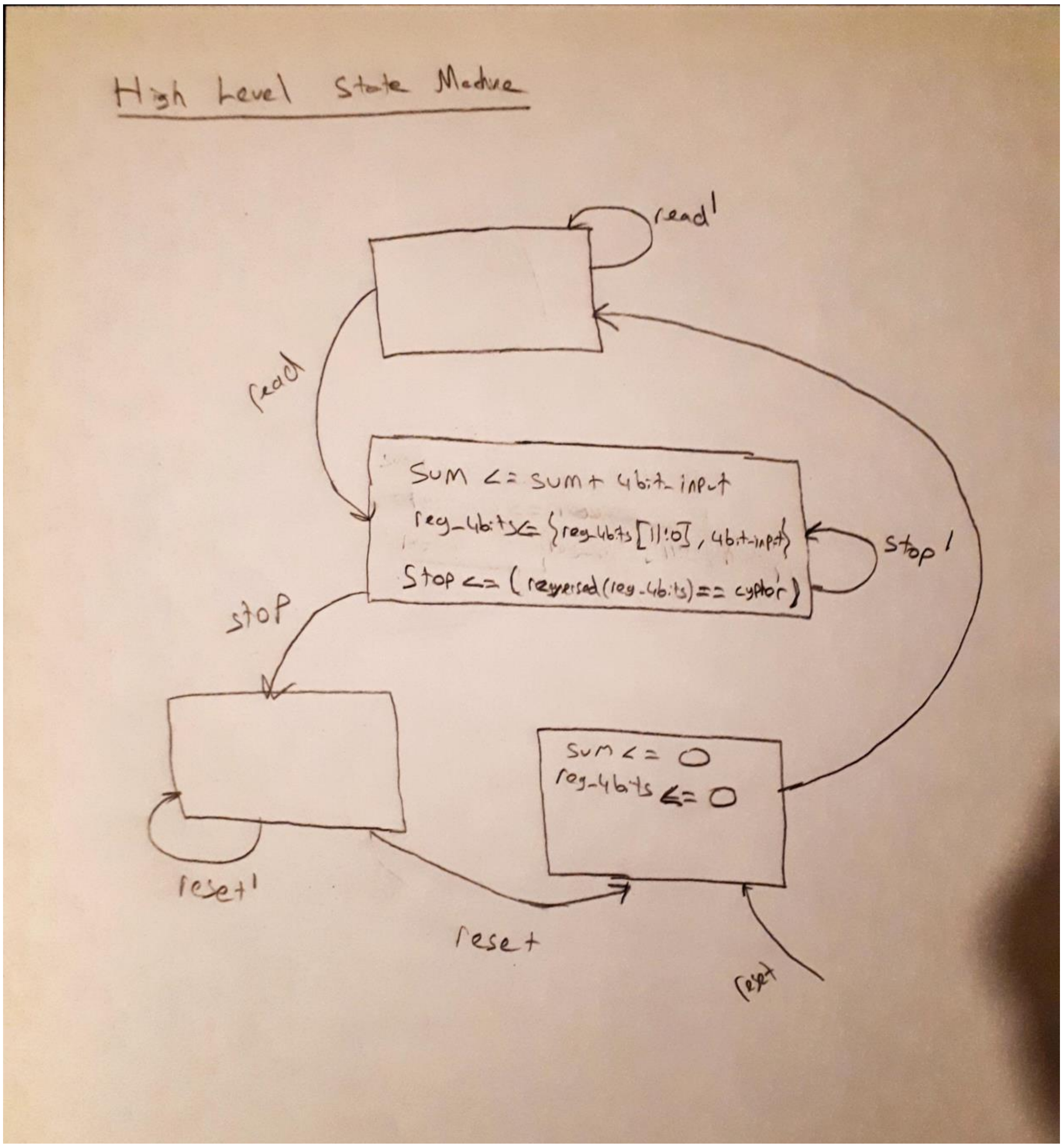
Content

Initial Design

Firstly, I draw the high level state machine as in the last lecture. Then I designed the datapath, and final state machine. High level state machine is quite similar to the final state machine, but fsm only includes 1 bit signals. I designed the high level state machine to determine what kind of registers, components needed in my datapath, also it helps to determine the signals between controller and datapath.

I used 4 states to perform cypher detection. First state waits for the read signal to perform operations. On that signal, datapath does not make sum or shift operations. When read signal becomes 1, it goes to the other state, and datapath performs summation of 4 bit input and sum registers value, then shifts the 16 bit register to hold last 4 bit inputs. Then the stop signal is generated with comparing the 16 bit cypher and reg_4bits. reg_4bits holds the 4 bit numbers in reversed order, therefore comparison must be made considering this condition. For changing state stop signal becomes important, if it is 1, then it waits in a step to hold the result, no operation is made on that state. It waits for the reset signal. After the reset signal, new state is the reset state, sum register, and reg_4bits is filled with 0. On that state it goes to the start state without condition

Here is the HSM:



It can be seen that I need 2 signal for the datapath from controller. And 1 signal from datapath to controller. Controller's signals for the datapath are sl_res, and sl_op. Datapaths signal for the controller is stop signal. Also controller needs reset, and read signals as input.

sl_res: reg_4bits, and sum register on the datapath is set to 0 when sl_res is 1.

sl_op: controller generates this signal for datapath to perform datapath operations, like summation, and changing the 4bits register in the datapath.

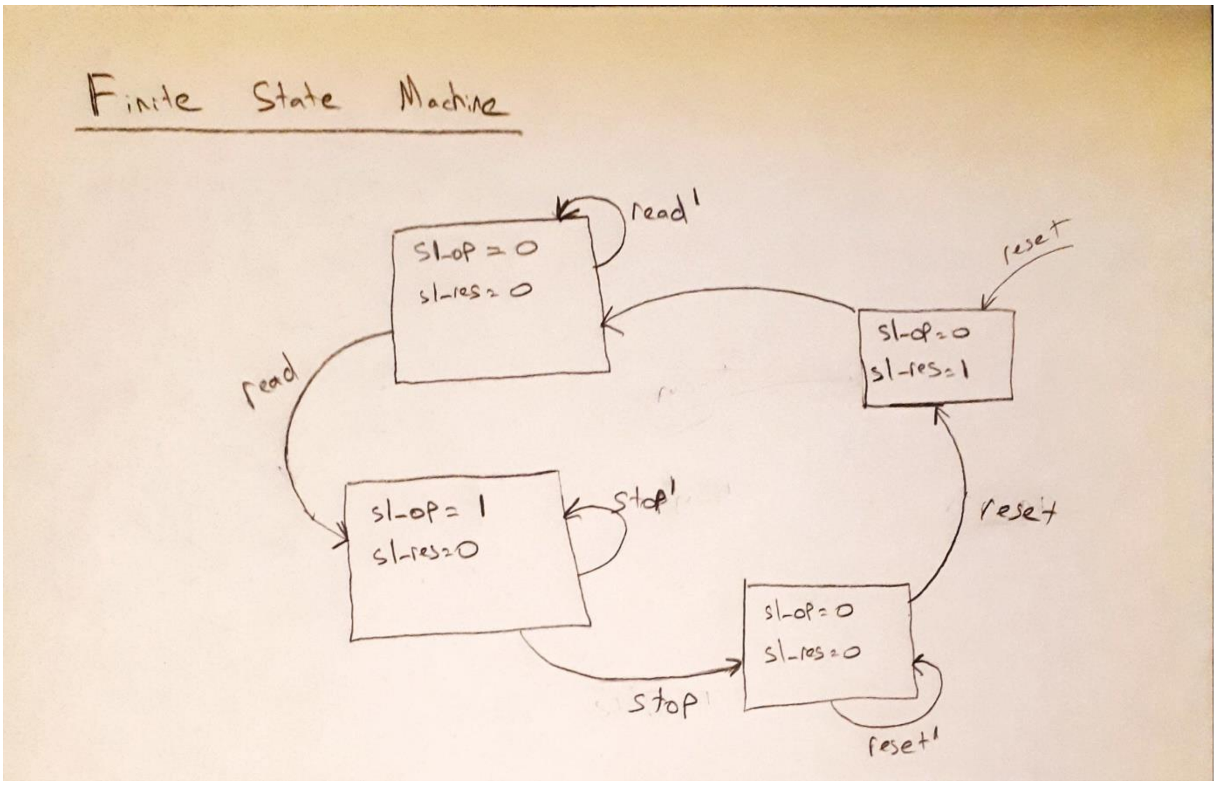
stop: This signal comes from datapath to force controller to stop. It is also finish signal to check cypher is detected or not. right signal is that stop signal.

read: It starts the procedure. Controller gets this signal as input.

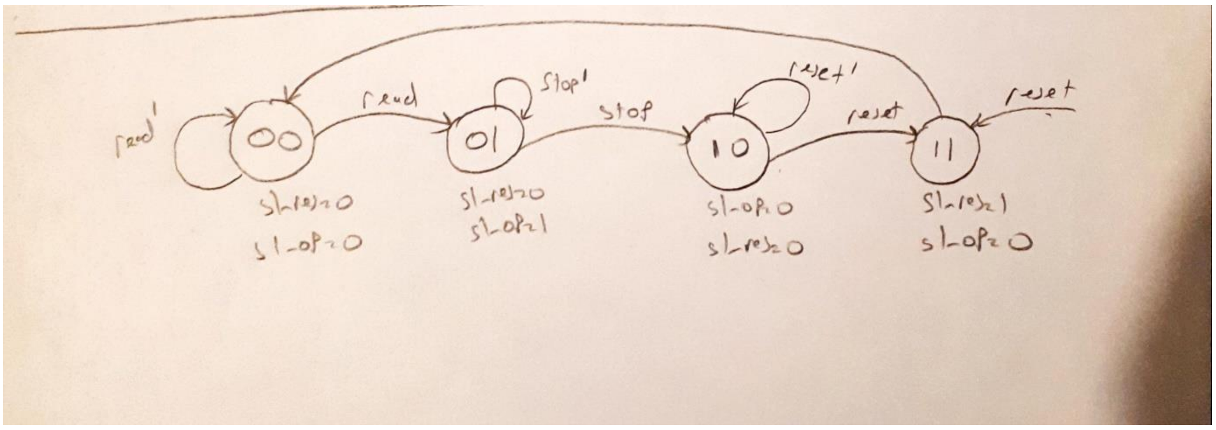
reset: It resets all registers, and results. Controller comes to initial state. Controller gets this signal, and controls datapath according to that.

- After determining that 5 signal, Final State Machine can be draw. Using 4 state, and 5 signal. When reset signal is 1 next state is reset state without condition.

Here is the FSM:



Simpler:

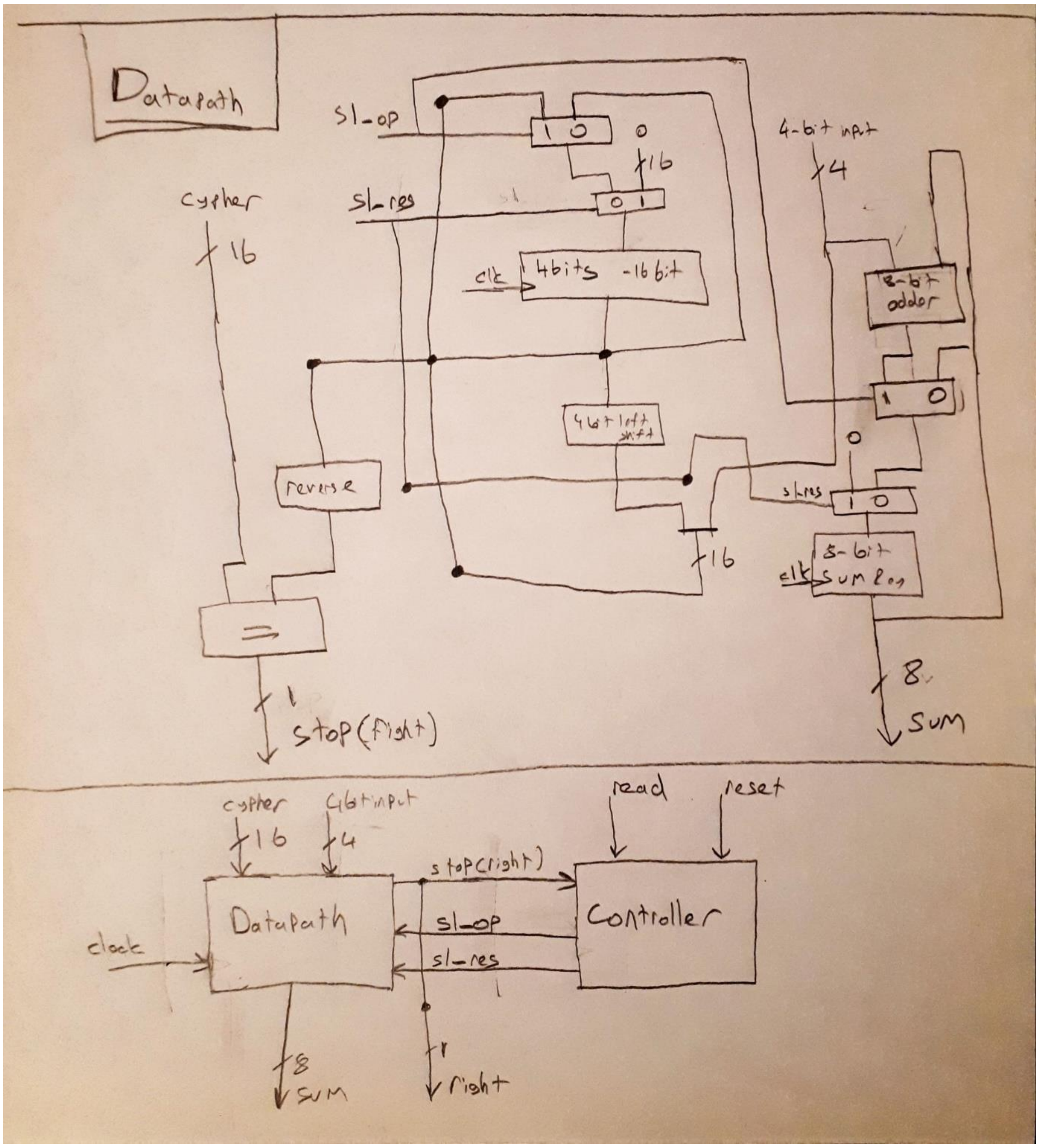


I designed it with 4 states. Initially it waits the read signal(00), when read signal comes, it goes to the operation state(01), on that state operation signal is set to 1. After that, it performs operation till stop signal becomes 1. When it becomes 1 it goes to the other stop state(10). And it stays on that state till reset is 1. If reset is 1(11), controller generates reset signal for the datapath, then goes to the initial state. Reset state can be reached from any state. If signal is one it directly resets all values, then goes to the initial state.

Datapath

After designing the finite state machine, datapath can be designed. There are 2 signals come from the finite state machine, these 2 signals are used in multiplexers. I need 2 register, one is for holding 4 bit inputs in 16 bit. Other one is for holding sum. It is 8 bit. No need to hold the cypher in the register. It will always be the same as input. 8-bit adder, and comparator is also needed. Shifter is not needed, because operation is simple, and can be done with using bus. This bus simply puts the 4 bit input, at the end of the 16 bit register, with removing the first 4 bit of it. Reversing is also can be done with buses. Getting the 4 bits, and putting them in a bus with reverse order can be enough to compare it with cypher. This comparator generates the result which output, and a signal that sent to the controller. Sum register and 16 bit register will need multiplexers to check operation signal, and reset signal. When reset signal is 1, registers value will be 0.

Here is the datapath:

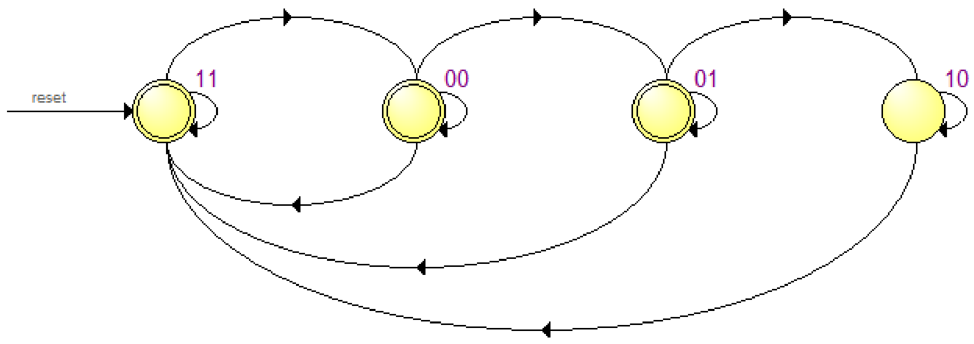


Verilog Part

I used 3 modules for the implementation, other than these 3 modules are test benches. These 3 modules are, datapath.v, control.v, cypher_detector.v.

datapath.v: Implementation the datapath that I draw. I needed 4 different wires for the sum and 4bit registers. Registers are updated in the positive edge of the clock.

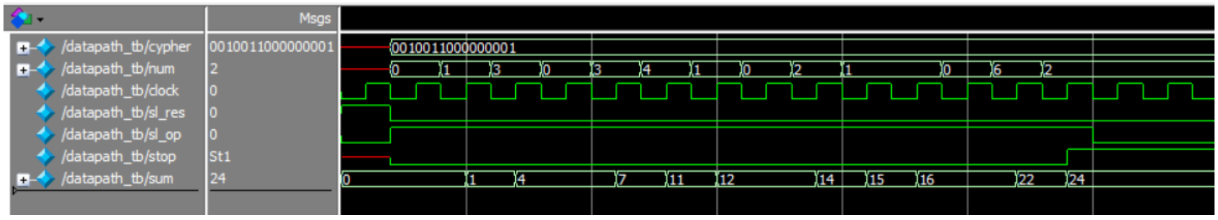
control.v: Implementation of the controller which is designed using fsm that I draw. Here is the state machine that generated from quartus according to that implementation.



cypher_detector.v: Main module that combines, controller and datapath. Inputs are clock, reset, num(4 bit input), cypher(16 bit). Outputs are sum(8 bit), and right(1 bit).

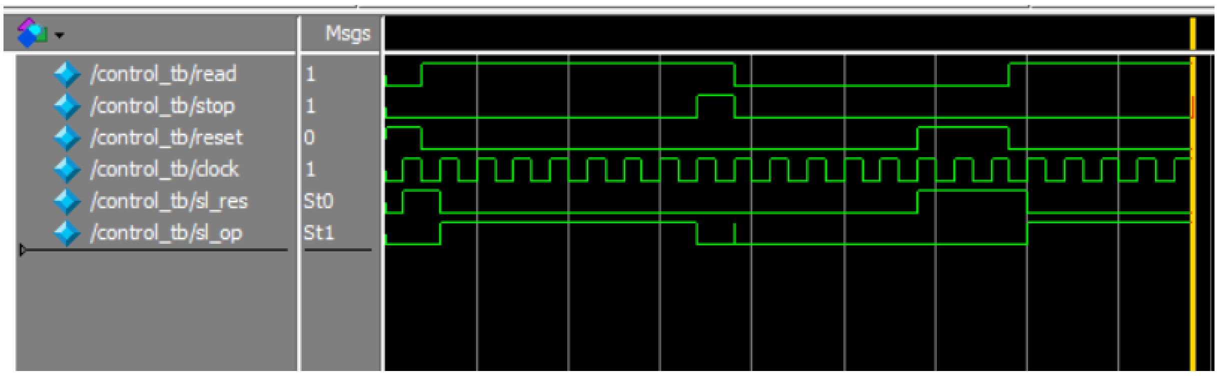
Test Bench

datapath_tb: For testing the datapath



- sum, and stop is correctly calculated with the input of sl_op, and sl_res.

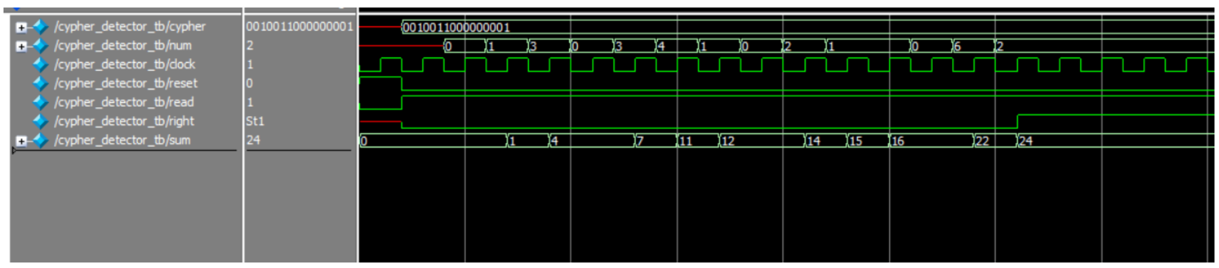
control_tb: For testing the controller



- sl_res, and sl_op signals are generated correctly.

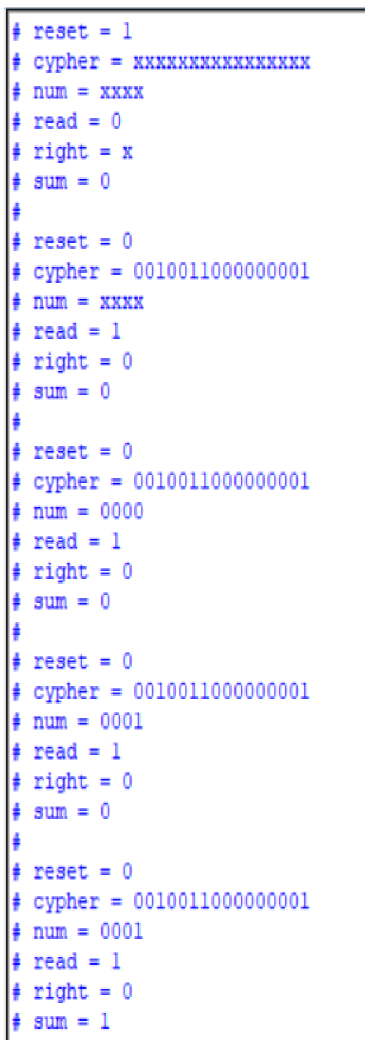
cypher_detector_tb: Testing cypher_detector with the example in the pdf

Wave Form:

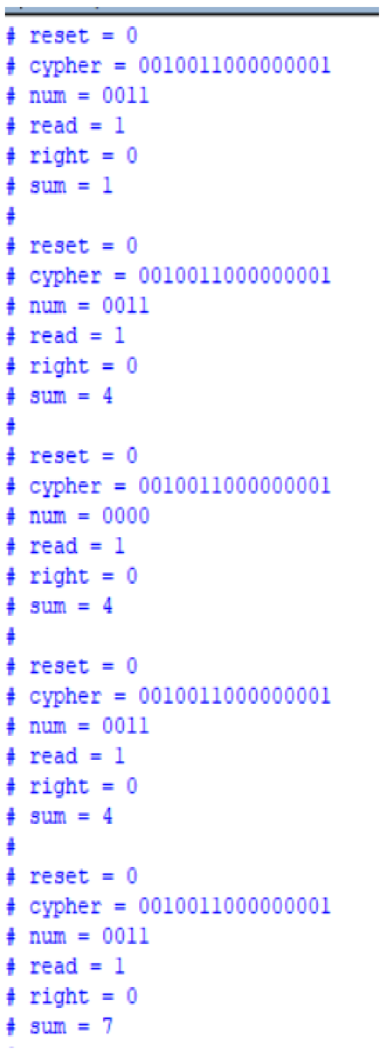

/cypher_detector_tb/cypher 0010011000000001
/cypher_detector_tb/num 2
/cypher_detector_tb/clock 1
/cypher_detector_tb/reset 0
/cypher_detector_tb/read 1
/cypher_detector_tb/right St1
/cypher_detector_tb/sum 24
0010011000000001
0 1 3 0 3 4 1 0 2 1 0 6 2
0 1 4 7 11 12 14 15 16 22 24

- cypher is detected, and sum is computed accurate, also it stopped when right pattern is found.

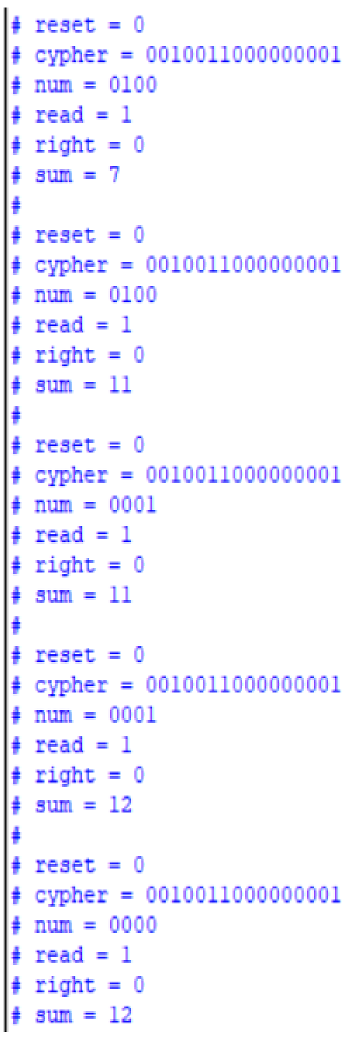
Monitor:



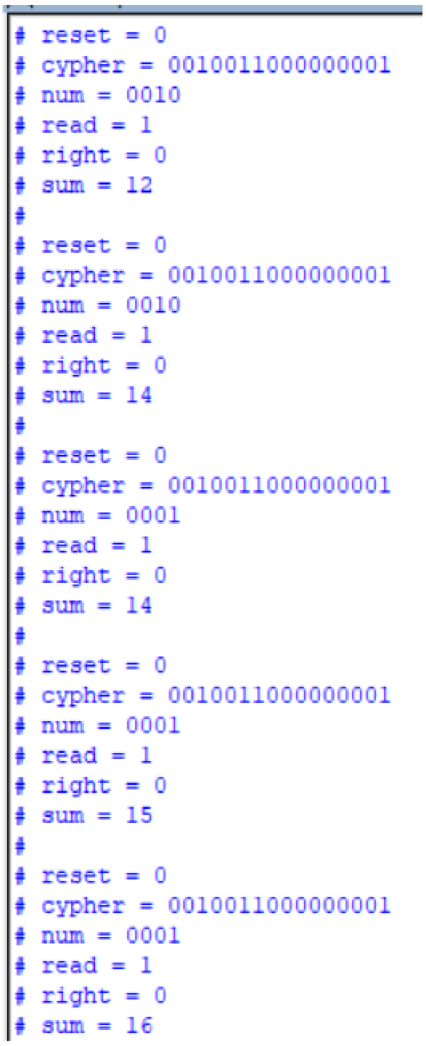
```
# reset = 1
# cypher = xxxxxxxxxxxxxxxx
# num = xxxx
# read = 0
# right = x
# sum = 0
#
# reset = 0
# cypher = 0010011000000001
# num = xxxx
# read = 1
# right = 0
# sum = 0
#
# reset = 0
# cypher = 0010011000000001
# num = 0000
# read = 1
# right = 0
# sum = 0
#
# reset = 0
# cypher = 0010011000000001
# num = 0001
# read = 1
# right = 0
# sum = 0
#
# reset = 0
# cypher = 0010011000000001
# num = 0001
# read = 1
# right = 0
# sum = 1
```



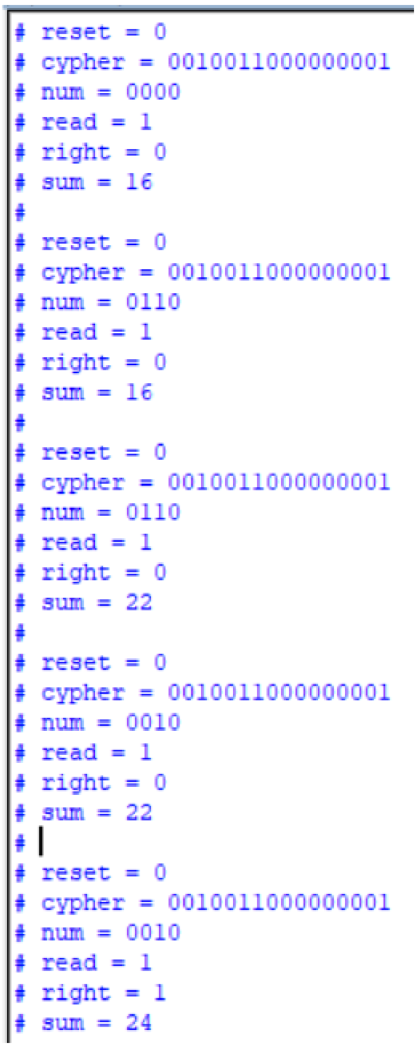
```
# reset = 0
# cypher = 0010011000000001
# num = 0011
# read = 1
# right = 0
# sum = 1
#
# reset = 0
# cypher = 0010011000000001
# num = 0011
# read = 1
# right = 0
# sum = 4
#
# reset = 0
# cypher = 0010011000000001
# num = 0000
# read = 1
# right = 0
# sum = 4
#
# reset = 0
# cypher = 0010011000000001
# num = 0011
# read = 1
# right = 0
# sum = 4
#
# reset = 0
# cypher = 0010011000000001
# num = 0011
# read = 1
# right = 0
# sum = 7
```



```
# reset = 0
# cypher = 0010011000000001
# num = 0100
# read = 1
# right = 0
# sum = 7
#
# reset = 0
# cypher = 0010011000000001
# num = 0100
# read = 1
# right = 0
# sum = 11
#
# reset = 0
# cypher = 0010011000000001
# num = 0001
# read = 1
# right = 0
# sum = 11
#
# reset = 0
# cypher = 0010011000000001
# num = 0001
# read = 1
# right = 0
# sum = 12
#
# reset = 0
# cypher = 0010011000000001
# num = 0000
# read = 1
# right = 0
# sum = 12
```

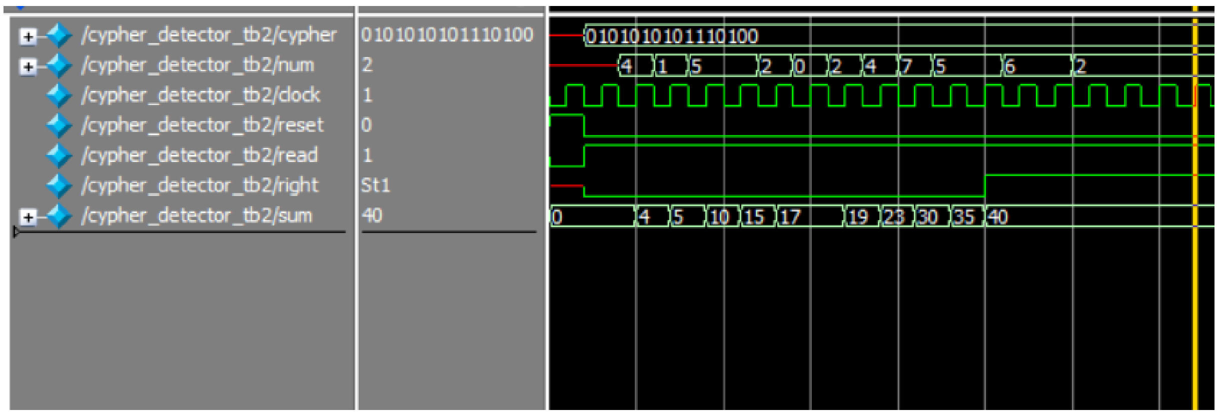


```
# reset = 0
# cypher = 0010011000000001
# num = 0010
# read = 1
# right = 0
# sum = 12
#
# reset = 0
# cypher = 0010011000000001
# num = 0010
# read = 1
# right = 0
# sum = 14
#
# reset = 0
# cypher = 0010011000000001
# num = 0001
# read = 1
# right = 0
# sum = 14
#
# reset = 0
# cypher = 0010011000000001
# num = 0001
# read = 1
# right = 0
# sum = 15
#
# reset = 0
# cypher = 0010011000000001
# num = 0001
# read = 1
# right = 0
# sum = 16
```

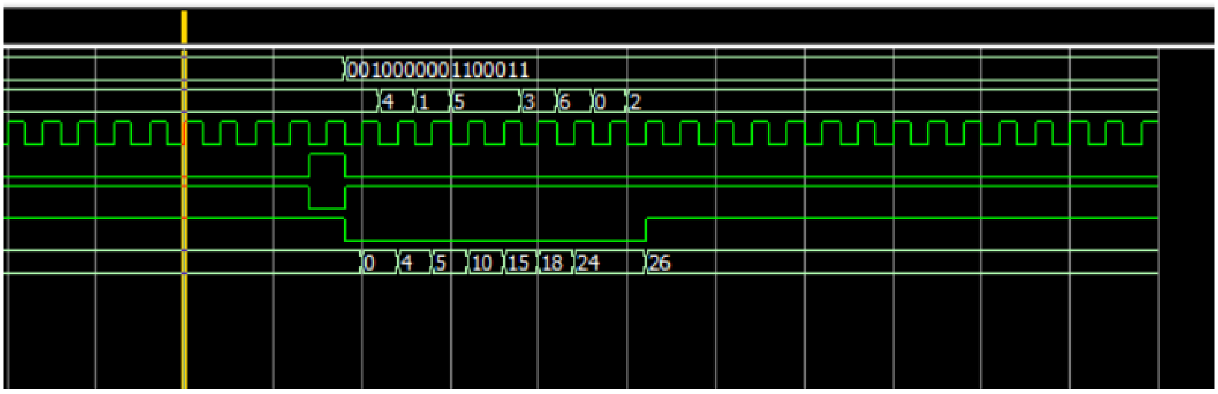


```
# reset = 0
# cypher = 0010011000000001
# num = 0000
# read = 1
# right = 0
# sum = 16
#
# reset = 0
# cypher = 0010011000000001
# num = 0110
# read = 1
# right = 0
# sum = 16
#
# reset = 0
# cypher = 0010011000000001
# num = 0110
# read = 1
# right = 0
# sum = 22
#
# reset = 0
# cypher = 0010011000000001
# num = 0010
# read = 1
# right = 0
# sum = 22
#
# reset = 0
# cypher = 0010011000000001
# num = 0010
# read = 1
# right = 1
# sum = 24
```

cypher_detector_tb2: Another test for cypher_detector

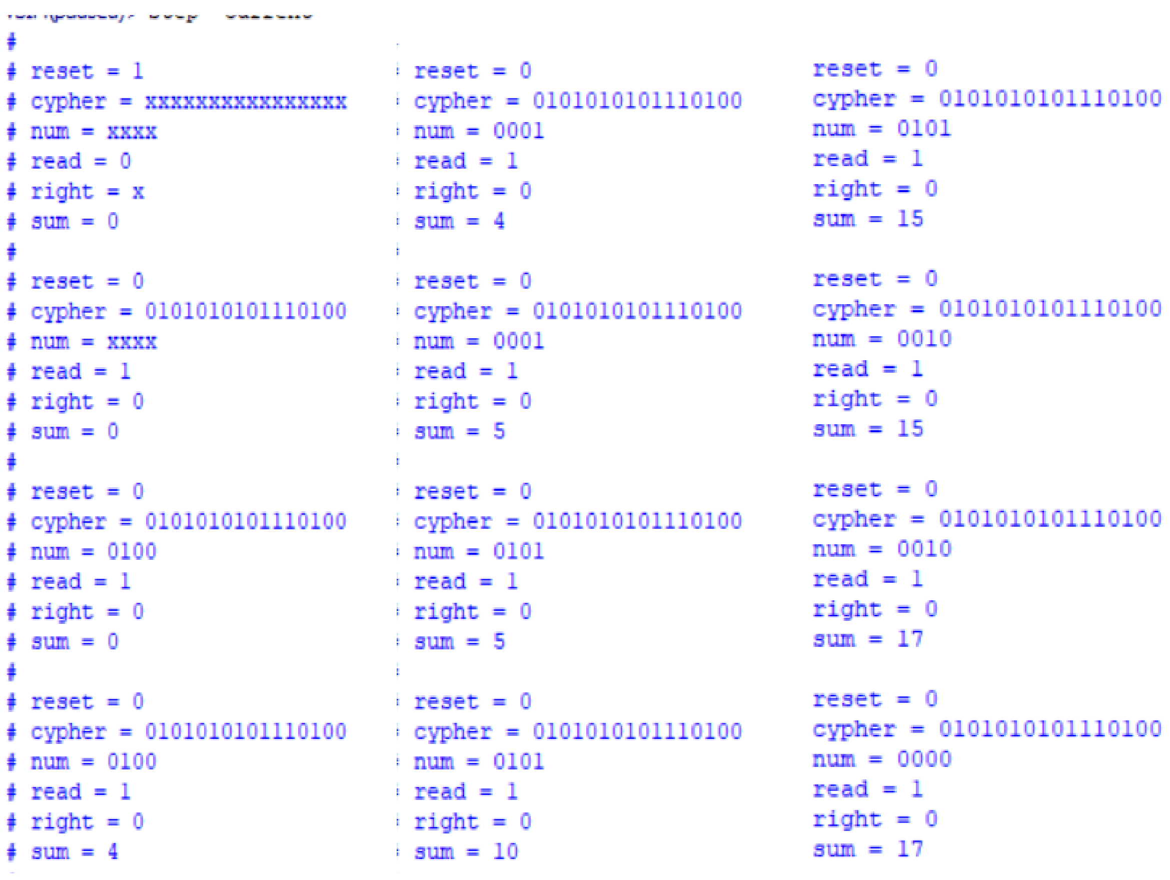


```
/cypher_detector_tb2/cypher   0101010101110100
/cypher_detector_tb2/num      2
/cypher_detector_tb2/clock    1
/cypher_detector_tb2/reset    0
/cypher_detector_tb2/read     1
/cypher_detector_tb2/right    St1
/cypher_detector_tb2/sum      40
```

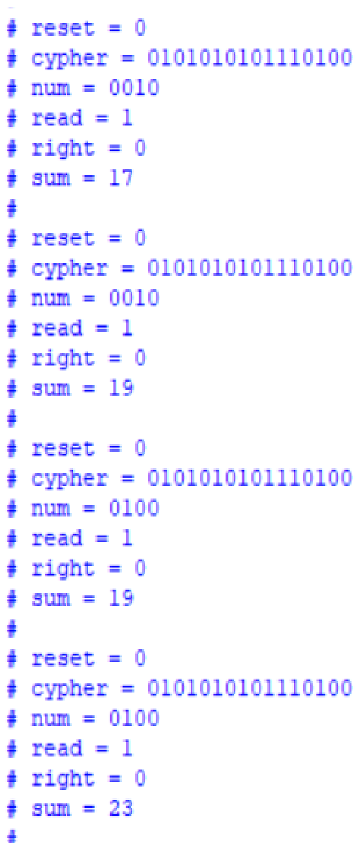


- It can be seen that design works for consecutive cyphers. After using reset, other cypher is sent, and it calculates the sum, and right signal. Both cypher is detected accurately.

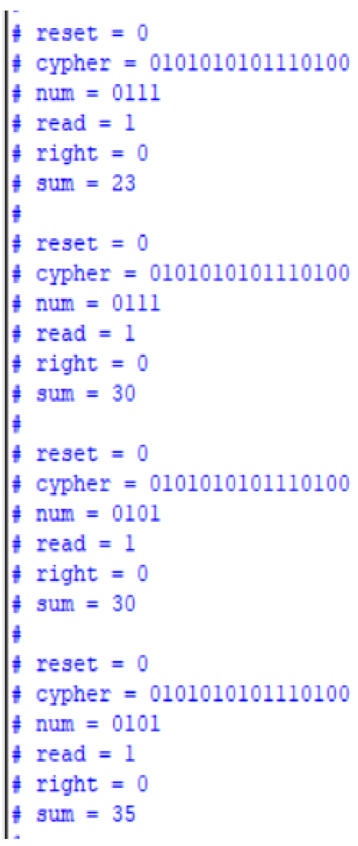
These are the monitor outputs:



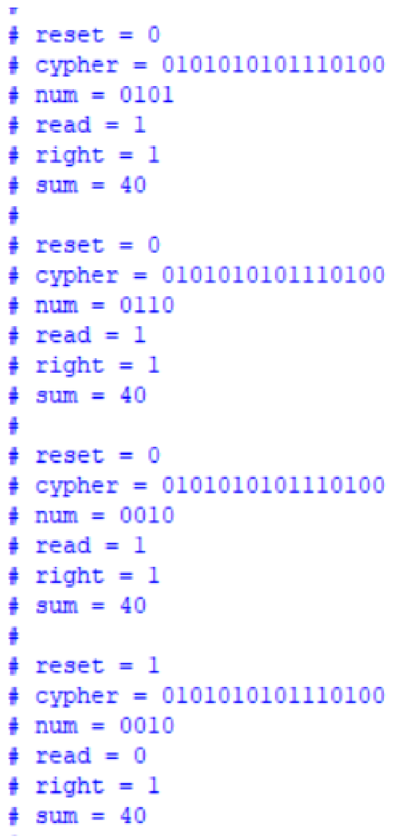
```
#
# reset = 1
# cypher = xxxxxxxxxxxxxxxx
# num = xxxx
# read = 0
# right = x
# sum = 0
#
# reset = 0
# cypher = 0101010101110100
# num = xxxx
# read = 1
# right = 0
# sum = 0
#
# reset = 0
# cypher = 0101010101110100
# num = 0100
# read = 1
# right = 0
# sum = 0
#
# reset = 0
# cypher = 0101010101110100
# num = 0100
# read = 1
# right = 0
# sum = 4

: reset = 0
: cypher = 0101010101110100
: num = 0001
: read = 1
: right = 0
: sum = 4
:
: reset = 0
: cypher = 0101010101110100
: num = 0001
: read = 1
: right = 0
: sum = 5
:
: reset = 0
: cypher = 0101010101110100
: num = 0101
: read = 1
: right = 0
: sum = 5
:
: reset = 0
: cypher = 0101010101110100
: num = 0101
: read = 1
: right = 0
: sum = 10

reset = 0
cypher = 0101010101110100
num = 0101
read = 1
right = 0
sum = 15

reset = 0
cypher = 0101010101110100
num = 0010
read = 1
right = 0
sum = 15

reset = 0
cypher = 0101010101110100
num = 0010
read = 1
right = 0
sum = 17

reset = 0
cypher = 0101010101110100
num = 0000
read = 1
right = 0
sum = 17
```



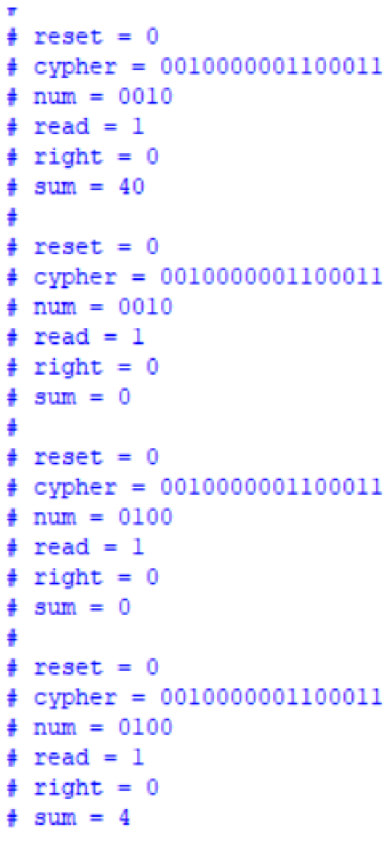
```
# reset = 0
# cypher = 0101010101110100
# num = 0010
# read = 1
# right = 0
# sum = 17
#
# reset = 0
# cypher = 0101010101110100
# num = 0010
# read = 1
# right = 0
# sum = 19
#
# reset = 0
# cypher = 0101010101110100
# num = 0100
# read = 1
# right = 0
# sum = 19
#
# reset = 0
# cypher = 0101010101110100
# num = 0100
# read = 1
# right = 0
# sum = 23
#
```



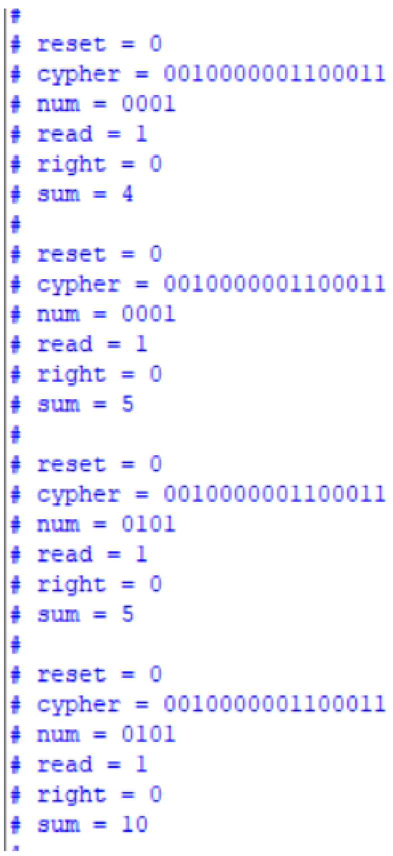
```
# reset = 0
# cypher = 0101010101110100
# num = 0111
# read = 1
# right = 0
# sum = 23
#
# reset = 0
# cypher = 0101010101110100
# num = 0111
# read = 1
# right = 0
# sum = 30
#
# reset = 0
# cypher = 0101010101110100
# num = 0101
# read = 1
# right = 0
# sum = 30
#
# reset = 0
# cypher = 0101010101110100
# num = 0101
# read = 1
# right = 0
# sum = 35
```



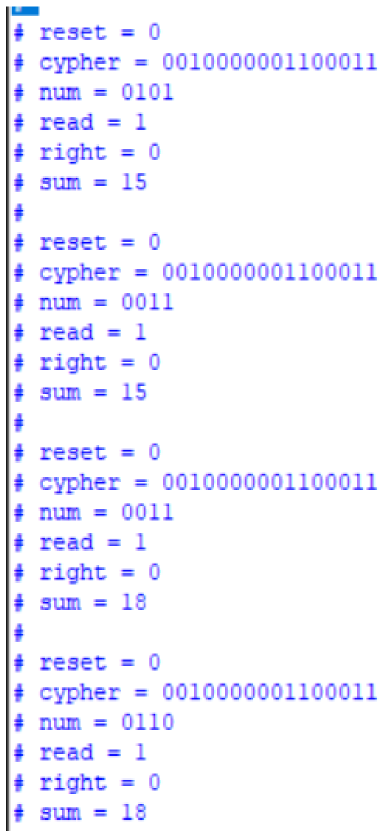
```
# reset = 0
# cypher = 0101010101110100
# num = 0101
# read = 1
# right = 1
# sum = 40
#
# reset = 0
# cypher = 0101010101110100
# num = 0110
# read = 1
# right = 1
# sum = 40
#
# reset = 0
# cypher = 0101010101110100
# num = 0010
# read = 1
# right = 1
# sum = 40
#
# reset = 1
# cypher = 0101010101110100
# num = 0010
# read = 0
# right = 1
# sum = 40
```



```
# reset = 0
# cypher = 0010000001100011
# num = 0010
# read = 1
# right = 0
# sum = 40
#
# reset = 0
# cypher = 0010000001100011
# num = 0010
# read = 1
# right = 0
# sum = 0
#
# reset = 0
# cypher = 0010000001100011
# num = 0100
# read = 1
# right = 0
# sum = 0
#
# reset = 0
# cypher = 0010000001100011
# num = 0100
# read = 1
# right = 0
# sum = 4
```



```
#
# reset = 0
# cypher = 0010000001100011
# num = 0001
# read = 1
# right = 0
# sum = 4
#
# reset = 0
# cypher = 0010000001100011
# num = 0001
# read = 1
# right = 0
# sum = 5
#
# reset = 0
# cypher = 0010000001100011
# num = 0101
# read = 1
# right = 0
# sum = 5
#
# reset = 0
# cypher = 0010000001100011
# num = 0101
# read = 1
# right = 0
# sum = 10
```



```
# reset = 0
# cypher = 0010000001100011
# num = 0101
# read = 1
# right = 0
# sum = 15
#
# reset = 0
# cypher = 0010000001100011
# num = 0011
# read = 1
# right = 0
# sum = 15
#
# reset = 0
# cypher = 0010000001100011
# num = 0011
# read = 1
# right = 0
# sum = 18
#
# reset = 0
# cypher = 0010000001100011
# num = 0110
# read = 1
# right = 0
# sum = 18
# reset = 0
# cypher = 0010000001100011
# num = 0110
# read = 1
# right = 0
# sum = 24
#
# reset = 0
# cypher = 0010000001100011
# num = 0000
# read = 1
# right = 0
# sum = 24
#
# reset = 0
# cypher = 0010000001100011
# num = 0010
# read = 1
# right = 0
# sum = 24
#
# reset = 0
# cypher = 0010000001100011
# num = 0010
# read = 1
# right = 1
# sum = 26
```