

Lab04b. Job Assignment Machine (JAM)

Description











					
	12	22	34	54	12
	45	21	97	98	34
	54	88	21	22	34
	12	43	57	21	33
	35	98	32	1	13

Figure 1: The job cost associated with each worker performing a different job.

This lab is centered around optimizing job assignments using the lexicographical algorithm to explore all possible combinations of assigning workers to jobs, aiming to minimize the overall cost. Figure 1 illustrates an instance involving five workers and five distinct jobs; however, the scenario addressed in this lab encompasses eight workers ($W=0$ to 7) and eight jobs ($J=0$ to 7), with a unique cost associated with assigning a specific job to a specific worker. This cost information is stored in an external read-only memory (ROM), accessible by submitting the worker (W) and job (J) identifiers to retrieve the cost of assigning job J to worker W . Also, each worker must be assigned to one job.

The job assignment machine (JAM) circuit should use the lexicographical algorithm to systematically generate all possible assignments of workers to jobs, in order to:

1. Calculate the total cost for each possible combination of assignments.
2. Determine the minimum cost achievable across all combinations.
3. Count the number of combinations that result in this minimum cost.

The Lexicographical Algorithm

Considering a set of n distinct elements, the total number of possible permutations is $n!$. The process of generating all possible permutations from this set is known as the lexicographical algorithm. To illustrate, for a set where n equals 4, such as **[0,1,2,3]**, the complete set of permutations would be meticulously enumerated as follows:

[0,1,2,3] [0,1,3,2] [0,2,1,3] [0,2,3,1] [0,3,1,2] [0,3,2,1]
[1,0,2,3] [1,0,3,2] [1,2,0,3] [1,2,3,0] [1,3,0,2] [1,3,2,0]
[2,0,1,3] [2,0,3,1] [2,1,0,3] [2,1,3,0] [2,3,0,1] [2,3,1,0]
[3,0,1,2] [3,0,2,1] [3,1,0,2] [3,1,2,0] [3,2,0,1] [3,2,1,0]

First, the lexicographical algorithm is introduced, where elements are sorted by their numbers. For example, **[2,3,1,0]** comes after **[2,3,0,1]**. This algorithm can find the next sequence for any given series and can be continuously used to generate subsequent sequences.

The lexicographical algorithm contains three steps and following illustrates an example of $n = 7$. Assuming the current sequence is **[3, 0, 4, 6, 5, 2, 1]**, find the next sequence:

1. Starting from the right, find the first neighboring location where the right-hand side (RHS) is larger than the left-hand side (LHS). Take the aforementioned sequence as the example, **[2, 1]** does not meet the condition since RHS is smaller than LHS; **[5, 2]** does not meet the condition; **[6, 5]** does not meet the condition. **[4, 6]** meet the condition since RHS is larger than LHS. We call the position of **4** the **replacement point** and **6** is the replacement number. **[3, 0, 6, 5, 2, 1]**
2. For the numbers on the RHS of the replacement point, find the minimum number that is larger than the replacement number and exchange that number with the replacement number. For this example, the replacement number is **4** and the numbers on the RHS are **[6, 5, 2, 1]**. The minimum number that is larger than **4** in this sequence is **5**. Thus **4** and **5** are exchange and the sequence becomes **[3, 0, 5, 6, 2, 1]**.
3. Finally, reverse the sequence to the right of the replacement point to get the next sequence. For example, to reverse the order of **[6, 4, 2, 1]** to get the sequence of **[3, 0, 5, 1, 2, 4, 6]**. This sequence is the next sequence of **[3, 0, 4, 6, 5, 2, 1]**.
4. The termination condition for the lexicographical algorithm occurs when no new permutation can be found, specifically, when the sequence is already arranged in descending order.

Specification

1. Top module name: JAM
2. Top module filename: JAM.v
3. Input/output definition

Signal Name	I/O	Width	Simple Description
CLK	I	1	Clock Signal (positive edge trigger)
RST	I	1	Reset Signal (active high). Provided by testbench, restoring to low 2 cycles after pull-up.
W	O	3	Assign to acquire the cost information for the W-th worker, $0 \leq W \leq 7$
J	O	3	Assign to acquire the the J-th cost information, $0 \leq J \leq 7$
Cost	I	7	The value of the cost. When W and J are assigned, Cost responds with the value of the cost for the W-th worker on the J-th job. Cost is an unsigned integer with the range between 0 and 100.
MatchCount	O	4	Output the number of combinations with minimum cost.
MinCost	O	10	Output the value of the minimum total job cost. MinCost is an unsigned integer. The minimum total job cost is not larger than 1024 in testbench.
Valid	O	1	When Valid is high, the MatchCount and MinCost are valid output, and the testbench finishes the simulation in the next cycle.

4. Input signals are unsigned number and **synchronized at the clock rising edge.**
5. The reset scheme is an **active-high asynchronous reset.**
6. The cost information related to the job is stored within an 8×8 synchronized memory block called costrom. Upon a system reset, the JAM circuit allocates W (worker identifier) and J (job identifier) signals to retrieve the specific cost information for the worker indexed by W for the job indexed by J. Both W and J indices range between 0 and 7. This cost data is fed into the system via the Cost input, and it is triggered by the rising edge of the CLK signal, allowing the JAM circuit to fetch data from the cost_rom multiple times.
7. Upon determining the lowest job cost and the total number of combinations yielding this minimum cost, the outcomes are output via the MinCost and MatchCount outputs. Simultaneously, the Valid signal is pulled up. When the testbench receives the Valid signal, it halts the simulation and proceeds to compare the outcomes.

- Try to write your design in a synthesizable coding style.

Example Timing Diagram

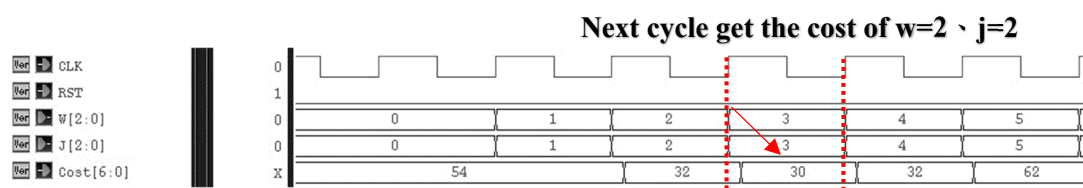


Figure 2: The Input Signals for the Job Assignment Machine

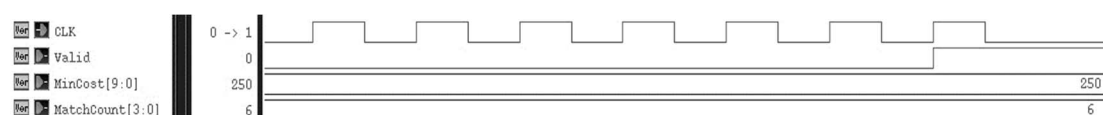


Figure 3: The Output Signals for the Job Assignment Machine

Lab Instructions

- Extract LAB data from TA's directory:

```
% tar xzvf ~dicta/lab04b.tar.gz
```
- The extracted LAB directory (**lab04b**) contains:
 - tb.sv** : Design test bench
 - JAM.v** : Empty design module
 - JAM_ans.vp** : Encrypted reference design module
 - cost_rom** : Cost of workers
 - lab04b.rc** : nWave saved signals file
- Change directory to **lab04b**

```
% cd lab04b
```
- Execute the Verilog simulation using the encrypted reference design (**JAM_ans.vp**). For your guidance in unstandardizing the input and output timing diagrams for this lab, protected Verilog modules are for your reference.

```
% ncoverilog -f run.f
```
- Open simulation waveform:

```
% nWave &
```

Choose **File** → **Open** from the menu, then select "**JAM.fsd**" and press OK. To restore the recorded signal, choose **File** → **Restore Signal** from the menu and select "**lab04b.rc**" and press OK.
- Write your RTL code in synthesizable coding style:

```
% gedit JAM.v &
or % gvim JAM.v &
or % joe JAM.v
```

or % vim JAM.v

7. Run RTL simulation with your design
(**Remember to modify the include file in tb.sv from JAM_ans.vp to JAM.v**)
% ncverilog -f run.f
8. Meaning of signals in DEBUG group:
goldMinCost : correct minimum cost
goldMatchCount : correct match count

