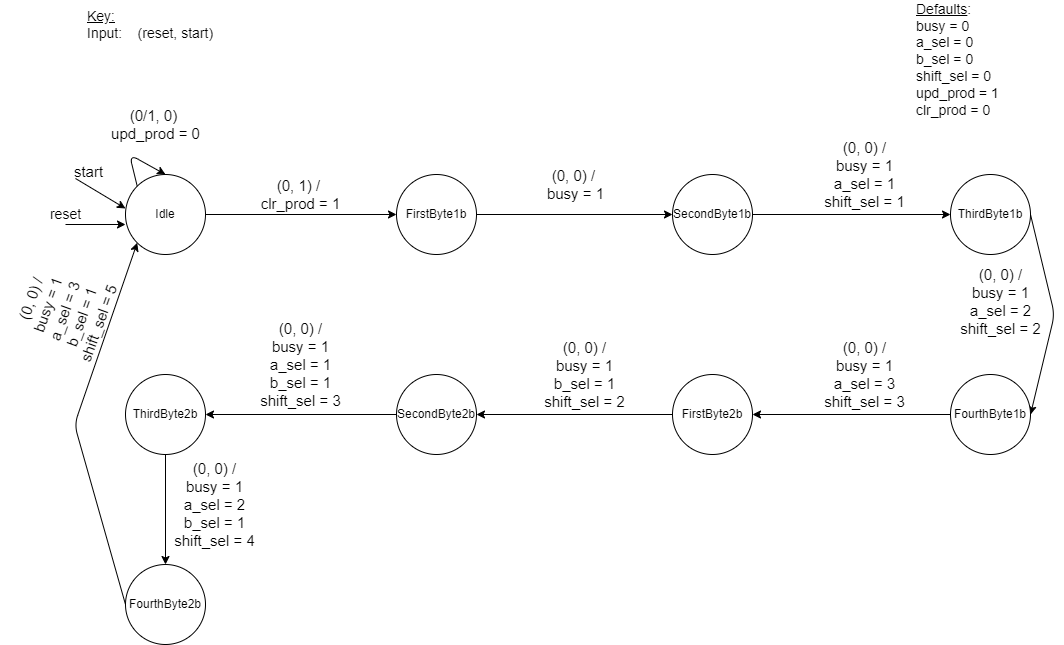
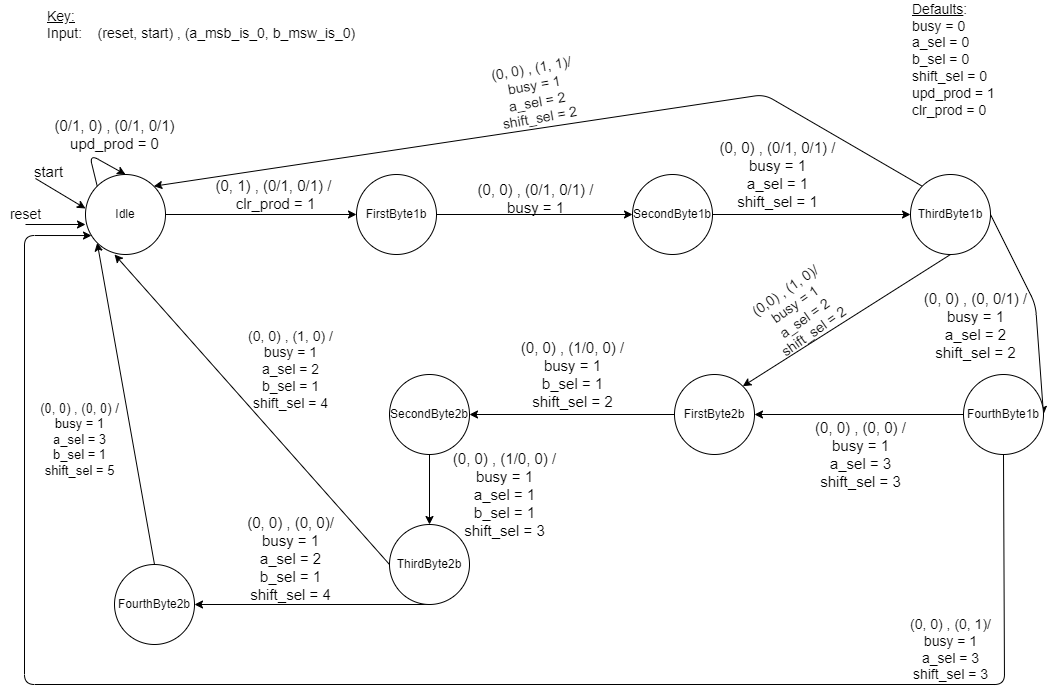
|  |  |
| --- | --- |
| 341312304 | שרה גריפית |
| 207223066 | טל שמיר |

2.1: Diag

The following is a diagram of the FSM we will implement:

Each multiplication takes 8 clock cycles, from the moment busy goes up to 1, until it goes down to 0.

2.2: 

The fastest calculation of the multiple will be given when both a\_msb\_is\_0 and b\_msw\_is\_0 are 1. In this case, the calculations after the state ThirdByte1b are irrelevant, and so the calculations will be completed after 3 clock cycles, from the moment busy goes up until it goes back down to 0.

If a\_msb\_is\_0 is 1 while b\_msw\_is\_0 is 0, the calculation will continue to the state FirstByte2b after ThirdByte1b, skipping the FourthByte1b. Likewise, the calculation will skip over the state FourthByte2b and instead return to Idle after ThirdByte2b. The calculation in this case will take 6 clock cycles, from the moment busy goes up until it goes back down to 0.

If a\_msb\_is\_0 is 0 while b\_msw\_is\_0 is 1, the calculation will continue to Idle after the state ourthByte1b. The calculation in this case will take 4 clock cycles, from the moment busy goes up until it goes back down to 0.

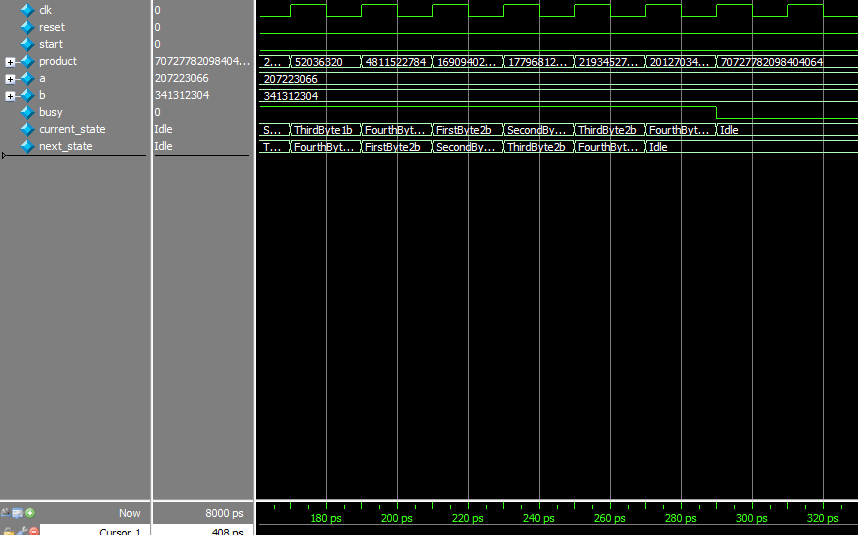
In every other instance (when both a\_msb\_is\_0 and b\_msw\_is\_0 are 0), the calculation time will remain the same as it was in the previous question, 8 clock cycles, from the moment busy goes up until it goes back down to 0.

3.4: Simulation of Multiplier 32x32

The following are the wave lines of the simulation and test bench created:

Chart

Description automatically generated



The clock starts at the value of 0 and changes its value every 10 units of time. Once the clock changes to 1 for the first time, the values of product, start, and reset are initialized: product and start are equal to 0, and reset is equal to 1 as requested. This lasts for 4 cycles of the clock.

The values of a and b are initialized after these 4 cycles, and thereafter, start is changed to 1 for one cycle, starting the multiplication process. One cycle after start changes, the value of busy is changed to 1, in accordance with the requirements of the exercise. As seen in the diagrams, with each clock cycle, the next\_state and current\_state change in the order described in the diagram from question 2.1. Finally, after 8 clock cycles (from the moment busy goes up until it goes back down to 0), busy goes down to the value of 0, the current and next states return to Idle, and the product from the multiplication of our ID numbers is received.

As shown, the result is: 341312304 x 207223066 = 70727782098404064

3.7: Simulation of Fast Multiplier 32x32

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated

The clock starts at the value of 0 and changes its value every 10 units of time. Once the clock changes to 1 for the first time, the values of product, start, and reset are initialized: product and start are equal to 0, and reset is equal to 1 as requested. This lasts for 4 cycles of the clock.

After these four cycles, the values of a and b are initialized to our ID numbers. Start is then changed to 1 for one clock cycle, after which busy becomes 1 and the multiplication process begins. The current and next states change throughout the calculation in accordance with the state diagram as shown in part 2.2. The calculation process in this case is identical to the process shown in the regular multiplier from the previous question. After 8 clock cycles, measured from the moment busy is equal to 1 until busy is equal to 0, the product of the multiplication is received. As in the previous simulation, the result is 341312304 x 207223066 = 70727782098404064.

After one clock cycle, a and b are initialized to new values: our ID numbers, such that the first two bytes of each number in their binary representation are 0. After one clock cycle, start is changed to the value of 1 for one clock cycle. Thereafter, busy changes to 1, and after 3 clock cycles, returns to the value of 0, and the product of the multiplication is received. The next and current states are as shown in the state diagram from question 2.2. The time this calculation takes is in accordance with our predictions given the ability to skip certain states as described in question 2.2. This is indeed significantly faster than if the calculation were done with the original implementation. The result received from this calculation is: 2330 x 816 = 1901280.