Simulation of the Fibonacci algorithm was left as last. It is the most complex of the three. The only one to utilize the stack. After some debugging related to POP and PUSH instructions, the algorithm calculated the results correctly. It has come to our attention that in the C++ version the 0th element of the series is 1 as well. Our code is implemented accordingly. This means there will be a shift in the results. In the conventional Fibonacci sequence the 5th element is equal to 5, but this version produces 8 as result for n =5. The simulation can be followed through on the screenshots in the Appendix. R2 holds the final result. In the beginning, the program keeps entering a new recursion. R1 is decremented with each iteration. The numbers in R4 change each time the condition is evaluated. The stack register increases as expected. Then the algorithm calls for fib(1). R2 is set to equal 1. The function returns for the first time. The progress can be tracked on R2. The inefficiency of the algorithm is visible, every time the function is called with a larger n, it has to start over and revaluate each element that comes before. Even for such a low input, the number of iterations is too high to fit the simulation into 1000 ns. Quartus only allows for a maximum period of 100 microseconds. The clock rate was 10 ns and the algorithm only managed to calculate the series up to its 10th element. This was expected, the purpose of the code is to act as a good benchmark to test the CPU's capabilities for using the stack. We initiated the stack register with a pointer E00. That means a maximum of 1000 iterations can be called before a stack overflow occurs and the CPU halts.