Lab 4

Babbage Difference Engine Emulation Circuit

# Introduction

The Babbage Difference Engine was invented by Charles Babbage based on a method discovered by Isaac Newton as a way to calculate polynomials without needing to multiply. We are going to implement this machine for the following second order polynomial:

The polynomial can be defined recursively using the following functions:

Using the above equations we can develop a machine to recursively calculate the value of the polynomial for a given input.

# Design

We need a finite state machine with data path to track the changes and calculate the result. We will keep track of the current running index n, the previous f(n) value, and the previous g(n) value. We will have three states: an idle state to wait for start and initialize data, a calculate state to calculate the outcome, and a done state to trigger the done indicator bit. During the calculation state we will conditionally increment g by four, increment f by g, and decrement the counter n. The following ASM graphically represents our design:

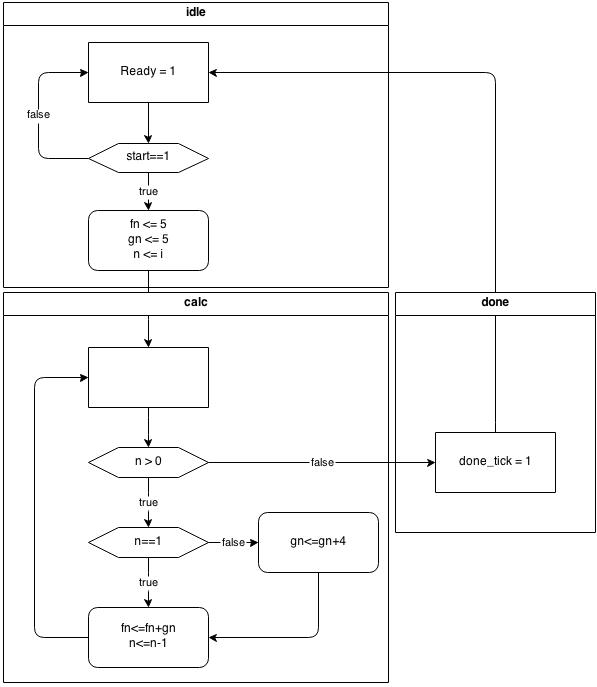


Figure 1: Babbage Engine ASM

# Implementation

The following Verilog code is our implementation of the above ASM:

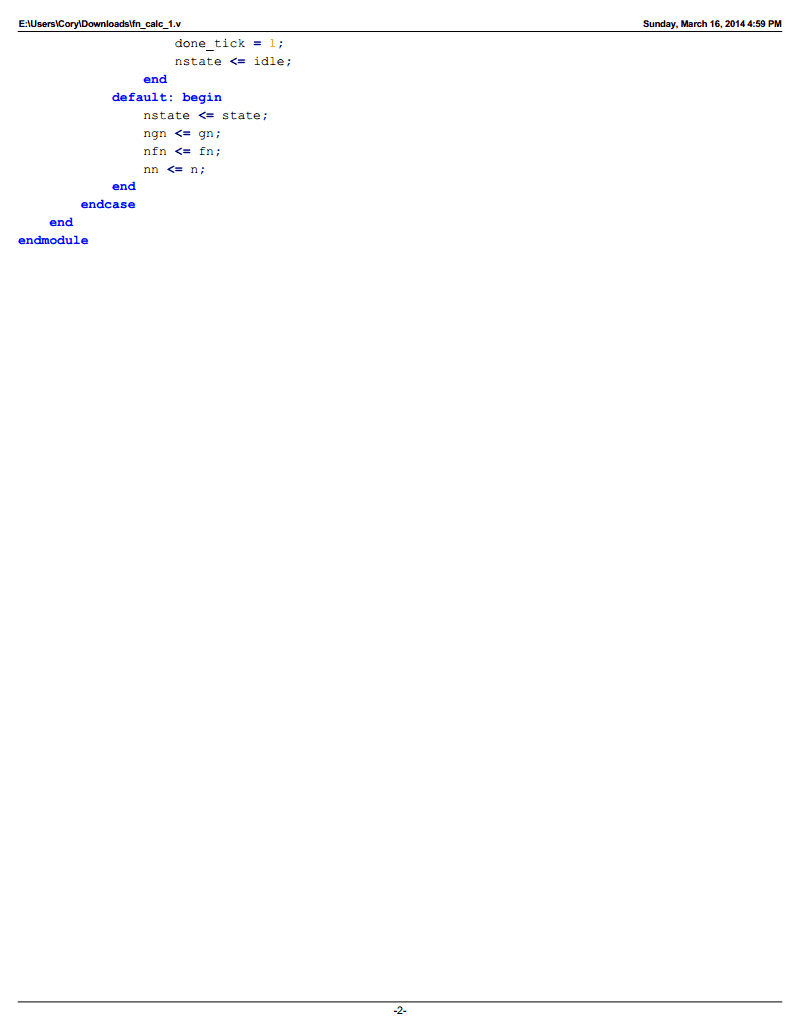
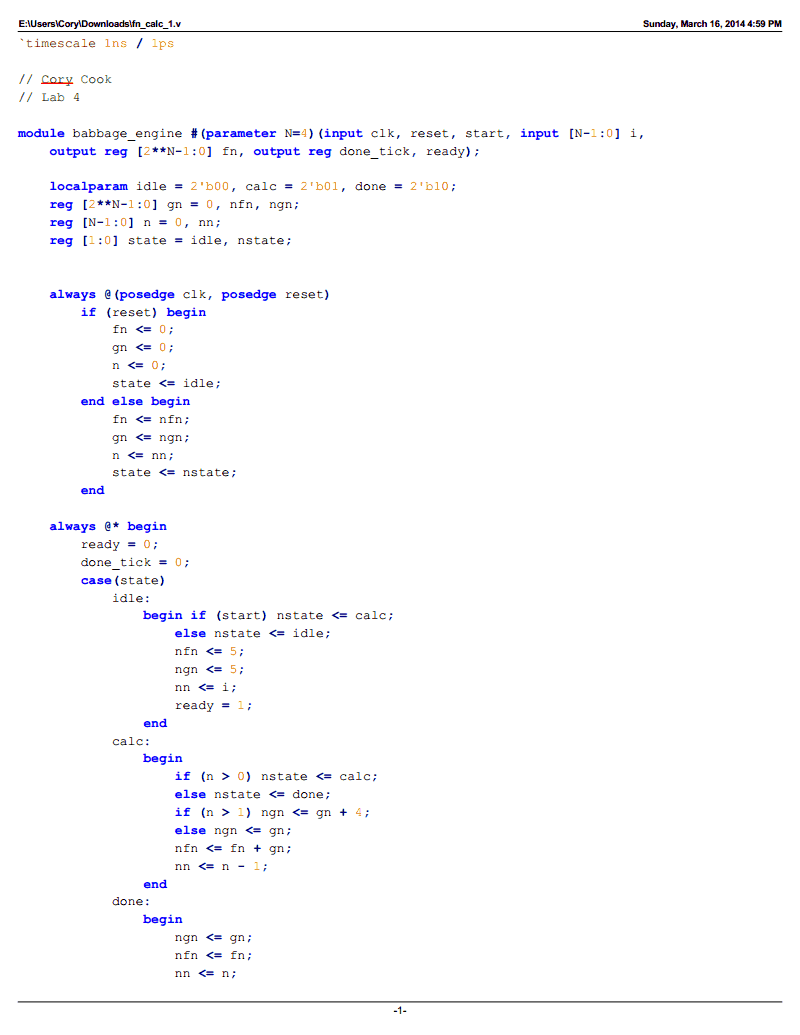


Figure 2: Babbage Engine Verilog

The code uses completed if-else and case statements to resolve warnings in the compiler and promote good design practices. This is apparent in the unused default case that merely assigns the current value to the next value. The above code produces a module that is usable directly in the main module of the application.

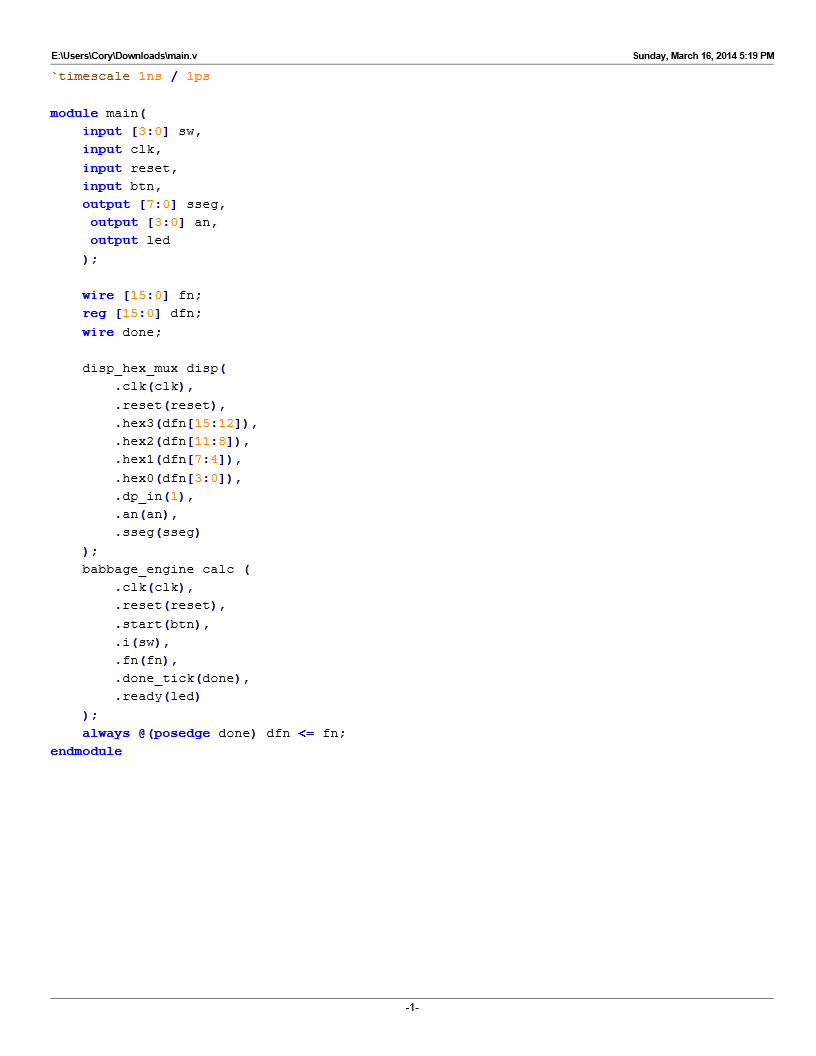


Figure 3: Main Module Verilog

The main module attaches the inputs from the board (.ucf file) to the inputs on the Babbage engine module, buffers the output of the Babbage engine module through a register that updates when the module indicates that it has completed, routes the value of the register through the book’s display hex module, and attaches the outputs to the board. The main module and the Babbage engine module produce the following RTL level design:

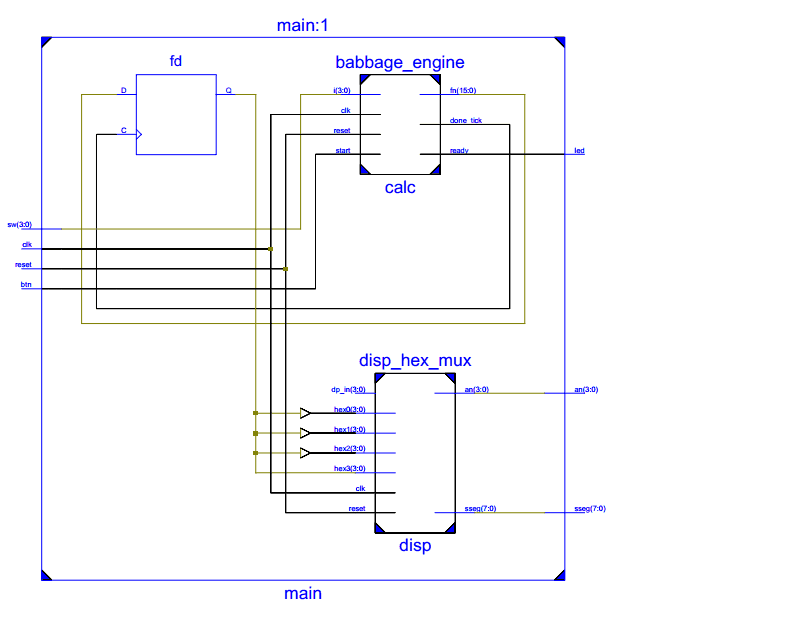


Figure 4: RTL Level Design

Here the RTL design shows how the input is routed to the Babbage engine module, the output of which is routed to the register labelled “fd,” the output of which is routed to the display hex multiplexer, the output of which is routed back to the board.

# Testing

The following shows the test of the design as run by the simulator:

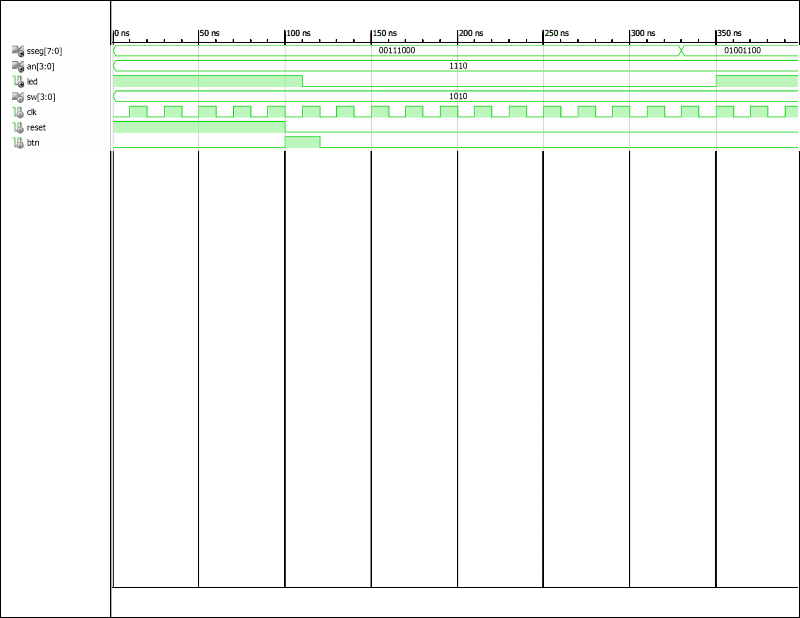


Figure 5: Main Test

The output (sseg) does not change until the Babbage engine module signals that the value is computed. The done state lasts for one cycle during which the done signal ticks and the ready signal is still low so the value changes one clock cycle before the ready led lights up.

# Conclusion

The Babbage engine design and implementation was simple, almost tedious, to implement following the design principles involving finite state machines with data path. It is encouraging to see how a seemingly difficult problem can have a simple solution. Finite state machine with data path gives us an opportunity to express complex problems in a simple way that is easy to implement.