

**Introduction**

The purpose of this 4-week lab is to convert the 32-bit single-cycle MIPS processor into a five-stage pipelined design. This processor is tested by interfacing it with a factorial accelerator and a general-purpose I/O unit by memory-mapped interface registers.

**Tasks**

* **Week #1**
  + Draft of pipelined MIPS microarchitecture schematic
  + Draft of SoC interface schematic

Shown in Figure 1 in Appendix A is the pipelined MIPS processor in 5 stages. The first register to indicate the first stage is the PC register which signifies the fetch stage. The second register to indicate the second stage is after the instruction memory to indicate it is in the decoding stage. This register is referred as the D\_reg. At this stage, the instruction is passed to the register file and control unit. The next register between the register file and ALU is the E\_reg which signifies the execute stage. After the computation in the execute stage, the M\_reg stores the data for data memory to use or to bypass the data memory to the last stage and register. This stage is referred to as the memory stage. The last stage is known as the writeback stage with the register named W\_reg. This stage writes the data back into the register file either reading from the data memory or reading from the bypassed data of M-reg.

Shown in Figure 2 in Appendix A is the overall schematic of the system on chip (SoC) schematic of the design. Figure 3 and 4 are more detailed schematics of the factorial accelerator and general-purpose I/O, respectively.

* **Week #2**
  + Tables for MIPS control unit and memory maps for SoC interface
  + Performance analysis of hardware accelerated n!
  + Unit-level (with interface wrappers) simulation waveforms

Planning of the control interface and memory management began in week 2 of this assignment. The logic tables for the MIPS control unit and memory maps for the System on a Chip design can be seen in Table 1 of Appendix B and Figure 5 of Appendix B. The control unit has the ability to process 15 instructions and generate 11 control signals to control the datapath, accounting for conditions such as jumps, branches, memory reads and writes, and arithmetic processes.

In week 2 of this project, work from previous labs began to be integrated. This started with a performance analysis of the hardware accelerated factorial calculator designed in the first lab of the semester, which can be seen in Figure 5 of Appendix B. As seen, the factorial accelerator has a significantly lower cycle count compared to a bare MIPS processor. For example, calculating 10! would only take 24 cycles in the accelerator as opposed to 96 cycles in a 4 stage MIPS processor.

The last task that was completed in week 2 was the unit-level validation of component wrappers using verilog waveforms. These can be seen in Figures 6 - 8 of Appendix B, and demonstrate proper functionality of base components.

* **Week #3**
  + Completed interface design for the SoC with the single-cycle MIPS processor (from Lab 7), the factorial unit (from Lab 1) and the simple GPIO
  + Demo of the integrated SoC on Nexys4 FPGA board.

The third week of this project saw the integration of the single-cycle MIPS processor from Lab 7 with the factorial accelerator from Lab 1 and a simple GPIO component of new design. Schematics for the integrated System on a Chip can be seen in Figure 2 of Appendix A, while the schematics for the factorial accelerator and the GPIO wrapper can be seen in Figures 3 and 4 of Appendix A, respectively.

As seen in the SoC schematic, the system is composed of 6 modules: Instruction Memory, the MIPS processor, an Address Decoder, Data Memory, the Factorial Calculator, and the GPIO Wrapper. The output of the Data Memory, Factorial Accelerator, and the GPIO is multiplexed per signals from the Address decoder.

The Factorial Accelerator contains its own address decoder to control timing and status signals output from the module. Input to the accelerator is determined by the decoded address and several registers. The output of the accelerator is held in two flip-flops until a done or error signal is generated. A multiplexer selected by the address decoder determines what to output from the module.

The GPIO wrapper can handle two input ports and two output ports. The input is fed through several registers and met in a multiplexer, which determines the output signal per an address decoder internal to the GPIO wrapper.

The last task for this week was a demonstration of the SoC design implemented on a FPGA board. This demonstration was mostly successful, but lacked proper processing of the error factorial accelerator’s error flag. Figures 9 - 10 of Appendix C showcase the validation.

* **Week #4**
  + Completed integration of the SoC using a pipelined MIPS processor
  + Demo of this upgraded SoC on Nexys4 FPGA board.

The fourth week of this project focused on adapting the single-cycle MIPS processor into a pipelined processor and integrating that into the SOC. The schematic for the pipelined MIPS processor can be found in Figure 1 of Appendix A.

As seen in the schematic, four stage registers were added in order to divide the processor into five stages. The first stage is the fetch stage, where the next instruction is pulled from instruction memory. The second stage is the decode stage, where the control unit and register file receive information from the current instruction about where the target, source, and destination registers are, and the type of instruction being executed. The third stage is the execute stage, where the alu applies operations to the specified registers based on the control unit. The fourth stage is the memory stage where the calculations can be written to the data memory. Finally, the fifth stage is the writeback stage, where the calculations are written back the specified register.

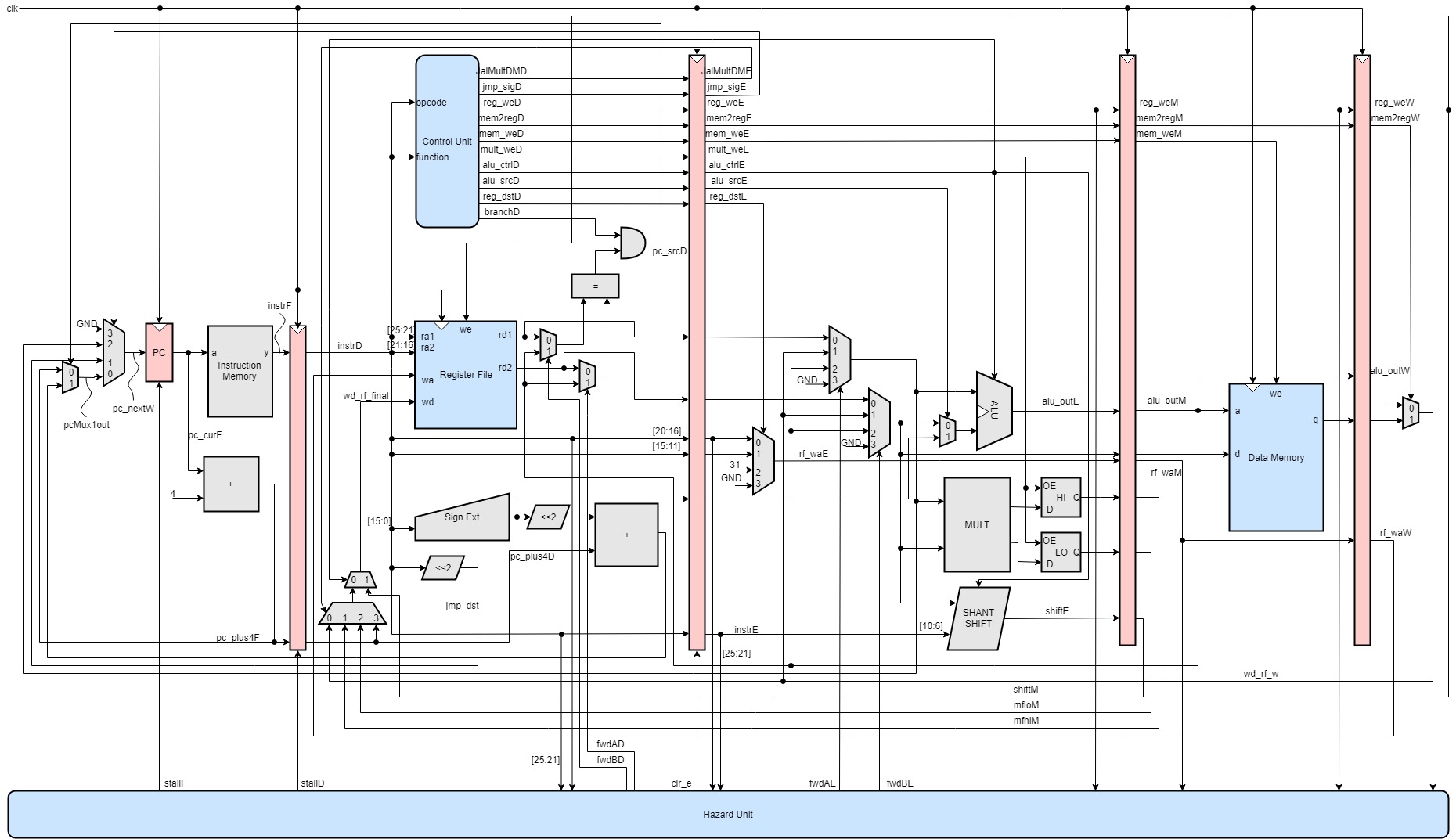
The pipelined MIPS processor also includes a hazard unit, which detects and accounts for data and control hazards. Data hazards are where an instruction is waiting for data from a previous instruction and the pipeline prevents the instruction from receiving that data. The hazard unit deals with this error by forwarding the data. In the case of the load word instruction, the hazard unit also stalls the fetch and decode registers and flushes the execution register so that the data forwarding will work. Control hazards occur when there is a branch in the instructions and the processor has one or more clock cycles where it does not know which instruction to fetch. This is dealt with by the hazard unit by stalling the fetch and decode registers, and flushing the execution register until the processor knows which instruction to fetch. This is commonly used for the BEQ instruction.

We were able to build the pipelined MIPS processor, but we were not able to functionally verify the pipelined processor or validate the design.

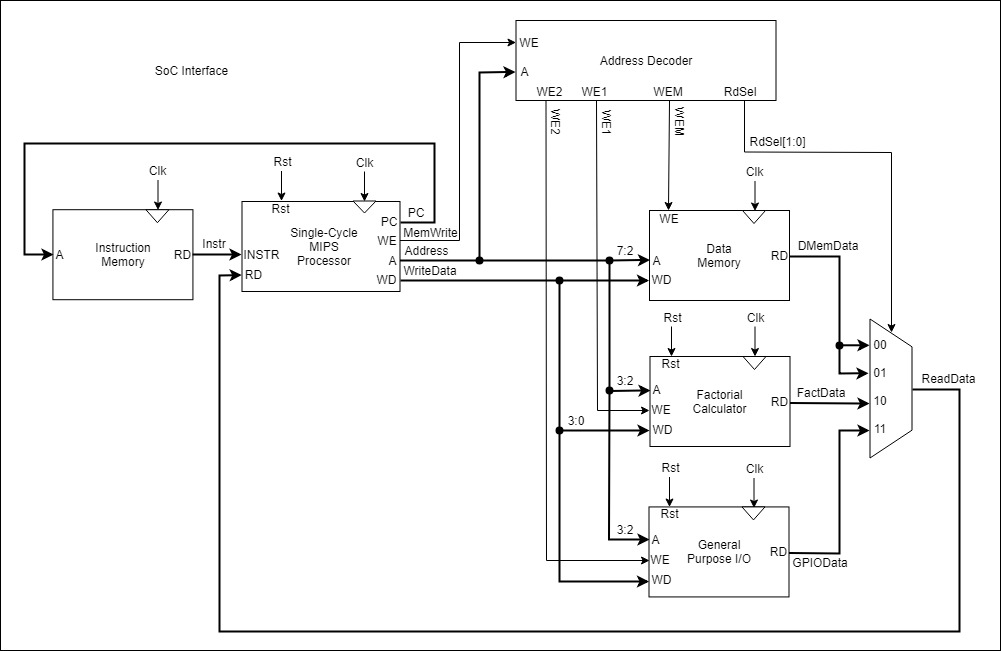
**Conclusion**

In conclusion, we were able to functionally verify and validate the single-cycle MIPS processor with the SOC design, but the error signal was not correctly implemented so we could not see when the design hit an error with a number greater than 12. We were not able to functionally verify or validate the pipelined MIPS processor.

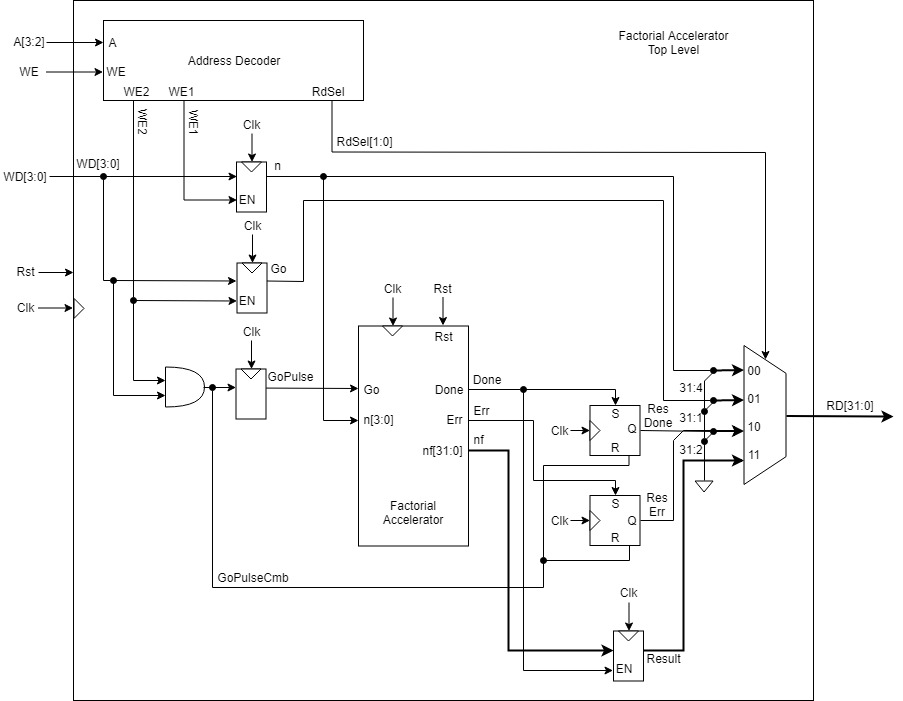
**Appendix A - Schematics**

****

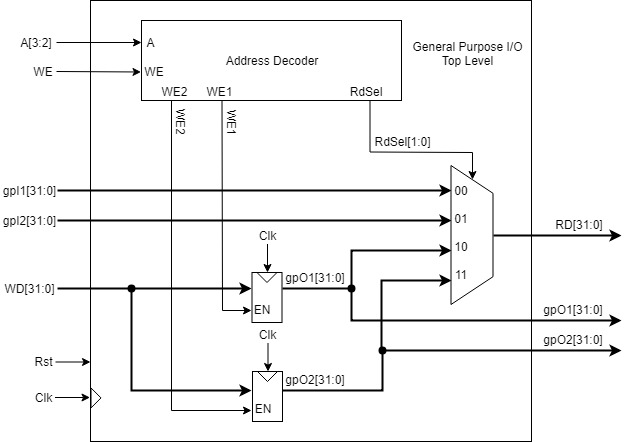
**Figure 1: Pipelined MIPS Processor Schematic**

****

**Figure 2: SoC Interface Schematic**

****

**Figure 3: Factorial Accelerator Schematic**

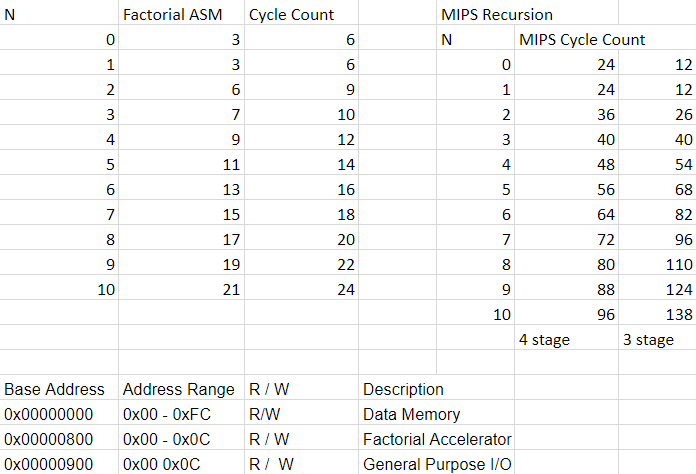
****

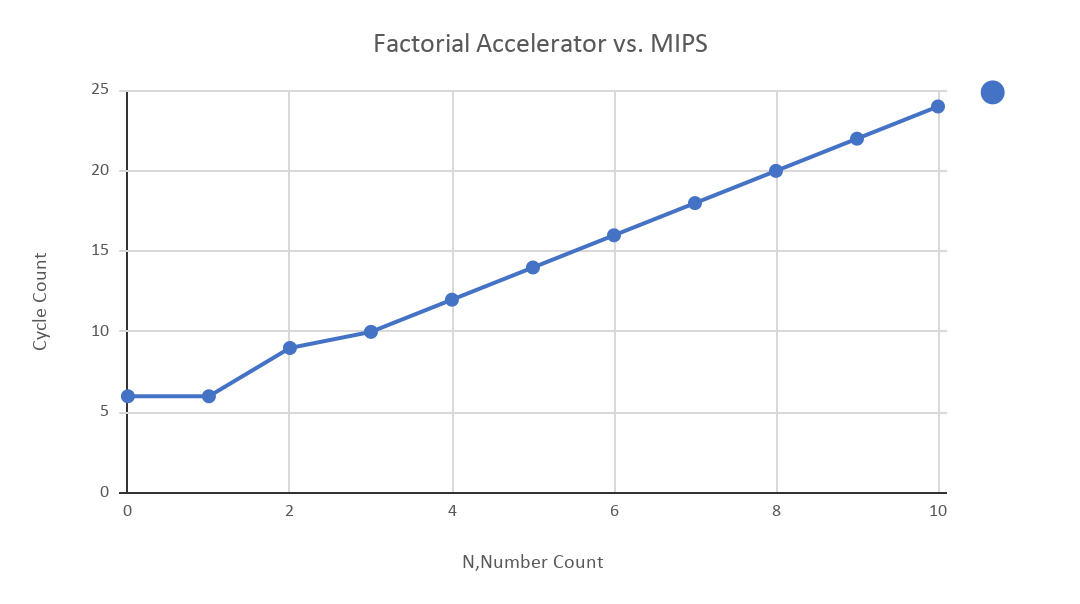
**Figure 4: General-Purpose I/O Schematic**

**Appendix B - MIPS Control Table, Performance Analysis, Wrapper Simulation Waveforms**

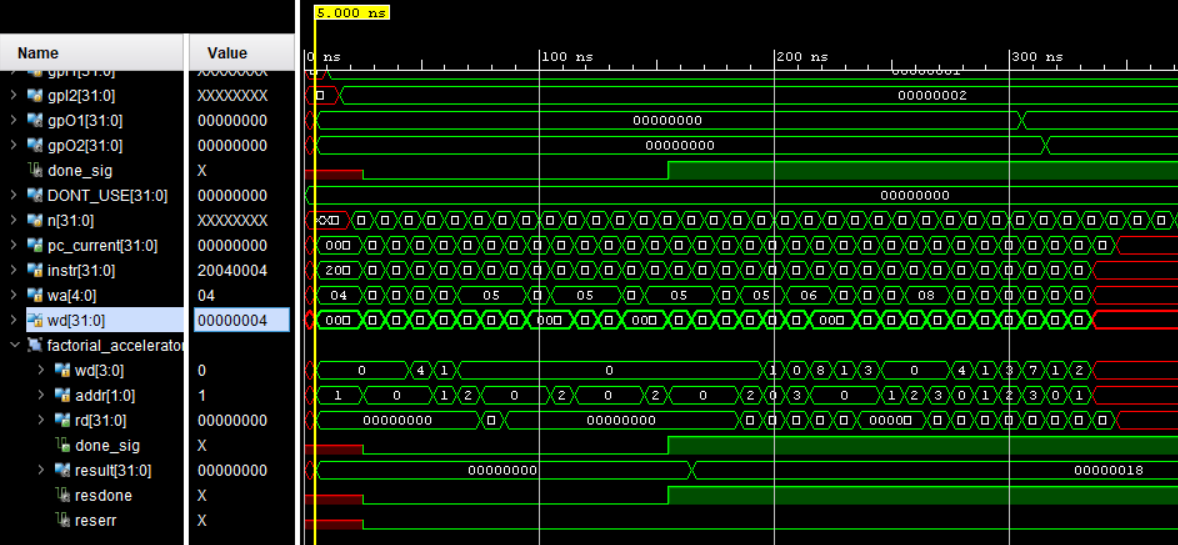
| **Instr** | **Control Signals** | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| jmp\_sigD [1:0] | alu\_srcD | reg\_dstD [1:0] | branchD | reg\_weD | alu\_ctrlD [2:0] | mem2regD | pc\_srcD | mult\_weD | JalMultDMD [1:0] | Instruction Type |
| ADD | 00 | 0 | 10 | 0 | 1 | 010 | 0 | 0 | 0 | 00 | R |
| SUB | 00 | 0 | 10 | 0 | 1 | 110 | 0 | 0 | 0 | 00 | R |
| AND | 00 | 0 | 10 | 0 | 1 | 000 | 0 | 0 | 0 | 00 | R |
| OR | 00 | 0 | 10 | 0 | 1 | 001 | 0 | 0 | 0 | 00 | R |
| SLT | 00 | 0 | 10 | 0 | 1 | 111 | 0 | 0 | 0 | 00 | R |
| LW | 00 | 1 | 00 | 0 | 1 | XXX | 1 | 0 | 0 | 00 | I |
| SW | 00 | 1 | 00 | 1 | 0 | XXX | X | 0 | 0 | XX | I |
| BEQ | 00 | 0 | XX | 0 | 0 | 110 | X | 0 | 0 | XX | I |
| J | 01 | X | XX | 0 | 0 | XXX | X | 0 | 0 | XX | J |
| ADDI | 00 | 1 | 00 | 0 | 1 | 010 | 0 | 0 | 0 | 00 | I |
|  |  |  |  |  |  |  |  |  |  |  |  |
| JR | 10 | X | 01 | 0 | 0 | XXX | X | 0 | 0 | XX | R |
| JAL | 00 | X | 10 | 0 | 1 | XXX | X | 1 | 0 | 11 | J |
| MULTU | 00 | 0 | XX | 0 | 0 | 011 | X | 0 | 0 | XX | R |
| MFLO | 00 | X | XX | 0 | 1 | XXX | X | 0 | 1 | 10 | R |
| MFHI | 00 | X | XX | 0 | 1 | XXX | X | 0 | 1 | 01 | R |
| SLL | 00 | X | XX | 0 | 1 | 100 | X | 0 | 0 | 00 | R |
| SRL | 00 | X | XX | 0 | 1 | 101 | X | 0 | 0 | 00 | R |

**Table 1: MIPS Control Unit Truth Table**

****

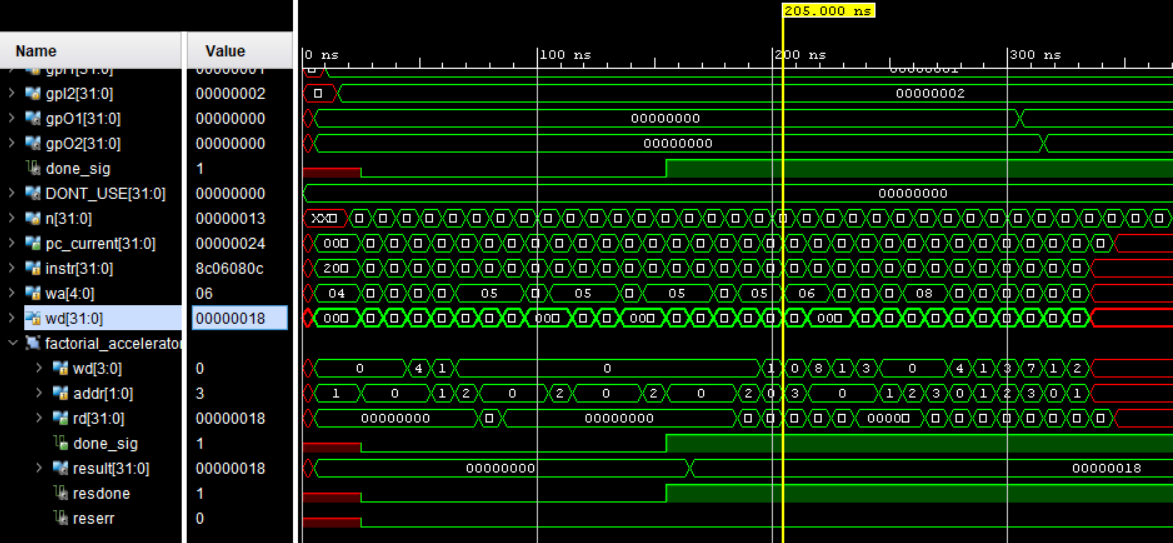
****

**Figure 5: Performance Analysis of Factorial Accelerator vs MIPS**

****

**Figure 6: Factorial Accelerator Wrapper Waveform**

**n = 4 is loaded into register**

****

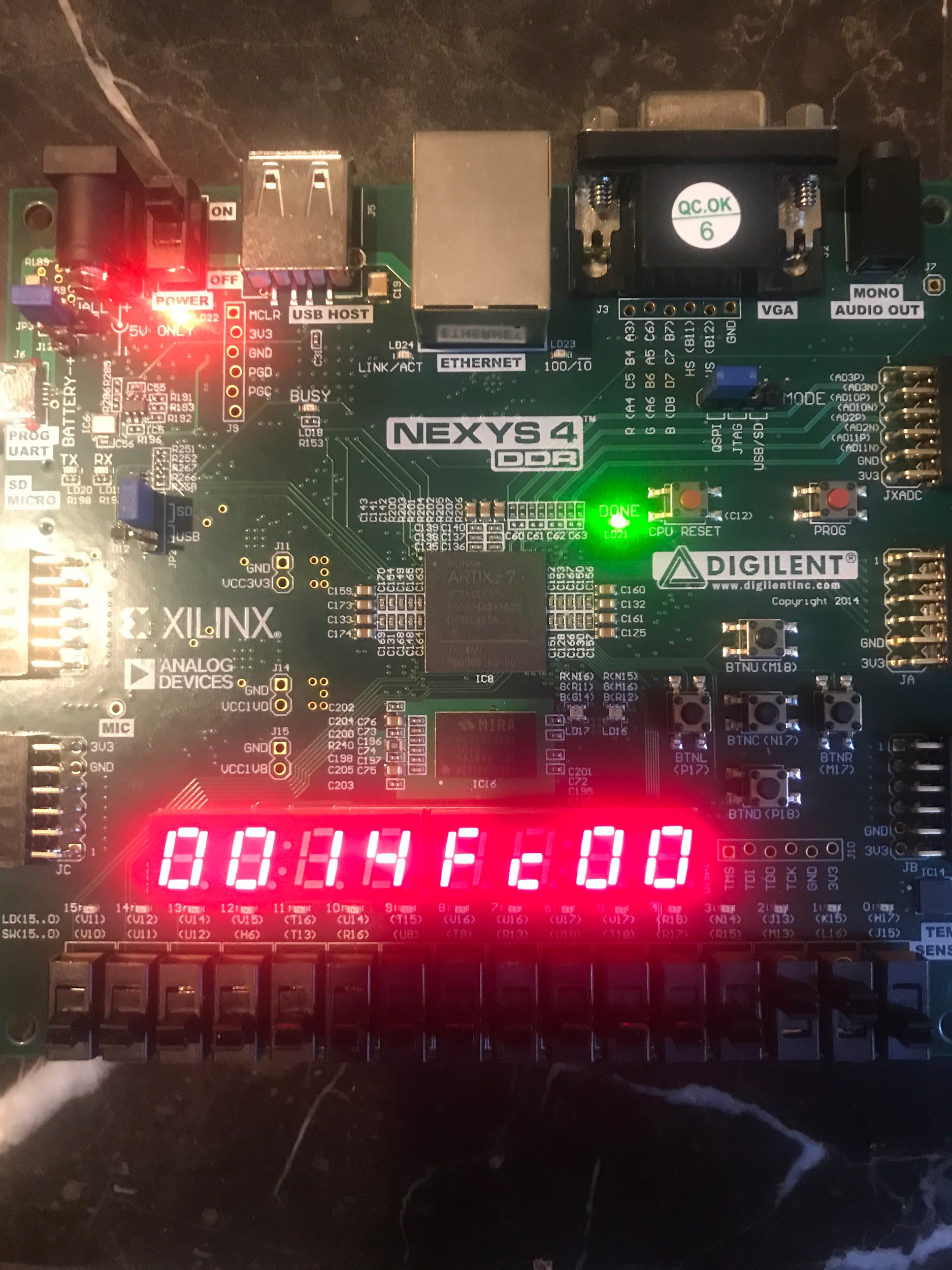
**Figure 7: Factorial Accelerator Wrapper Waveform**

**n! = 24 (0x18) is loaded into register**

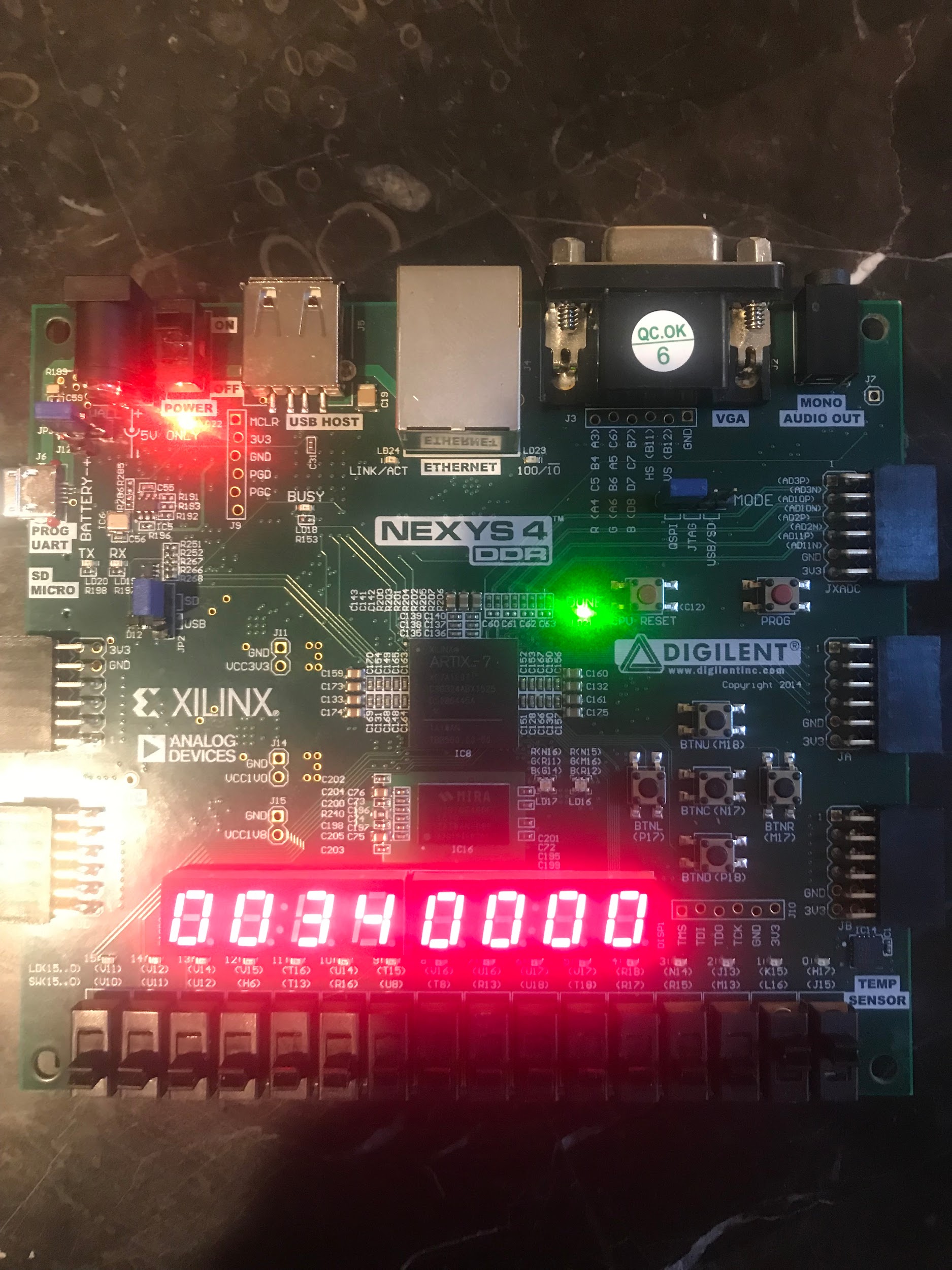
****

**Figure 8: GPIO Wrapper Waveform**

**Appendix C - Single-Cycle MIPS FPGA Validation**

****

**Figure 9: Factorial Hardware Validation of 0110**

****

**Figure 10: Factorial Hardware Validation of 0011**