Technical Report TR-CST17OCT03

University of Pennsylvania
Center for Sensor Technology
Philadelphia, PA 19104

SUNFEST REU Program

IMPLEMENTATION OF THE ENIAC ACCUMLATOR AND CYCLING UNIT ON A FIELD PROGRAMMABLE GATE ARRAY (FPGA)

NSF Summer Undergraduate Fellowship in Sensor Technologies
Emily Rose Blem (Mathematics and Engineering) – Swarthmore College
Advisor: Dr. Jan Van der Spiegel

ABSTRACT

An accumulator and cycling unit from the Electronic Numerical Integrator and Calculator (ENIAC) are implemented on a Field Programmable Gate Array (FPGA). The FPGA implementation is not architecturally identical to the original ENIAC, but the original architectural design is maintained where possible. The design maintains decimal number representation, ring counters to increment numbers, and ten different clocks to coordinate various accumulator functions. The implementation is a preliminary step in the design of a series of FPGAs that will include a constant transmitter, three accumulators, be connected to panels similar to those of the original ENIAC and be an exact replica in the programmer's experience.

Table of Contents

IMPLEMENTATION OF THE ENIAC ACCUMLATOR AND CYCLING UNIT ON A FIELD PROGRAMMABLE GATE ARRAY (FPGA)

Emily Rose Blem (Mathematics and Engineering) – Swarthmore College
Advisor: Dr. Jan Van der Spiegel

1. INTRODUCTION

Completed in 1946, the ENIAC (Electronic Numerical Integrator and Calculator) was the first general use electronic computer. It was re-created in 1997 on a silicon chip using CMOS technology at Moore School of Electrical Engineering at the University of Pennsylvania, and this paper explores implementation of the ENIAC on a series of Field Programmable Gate Arrays (FPGAs). The accumulator and cycling units have been implemented on an FPGA. The FPGA implementation is not architecturally identical to the original ENIAC, but the original architectural design is maintained where possible. The series of FPGAs will be connected to panels similar to those of the original ENIAC and be an exact replica in the programmer's experience. While the ENIAC-on-a-Chip demonstrated technological advances over the past 50 years, the FPGA implementation allows users to program as if they were interacting with the original ENIAC and serves as a valuable historical teaching tool.

The original ENIAC (Electronic Numerical Integrator and Calculator) was introduced as a tool for ballistics calculations as well as the first electronic general use computer. It was originally designed to calculate weapons firing tables for the Ballistics Research Laboratory during World War II. However, it was not completed in time for this purpose, and its first calculations were for nuclear research. This original ENIAC filled a room that was approximately 1800 square feet and contained thousands of components. The computer broke down an average of once every 5.6 hours, mostly because of its more than 40,000 vacuum tubes [1, 2]. The ENIAC was dismantled upon obsolescence, and the programming panels lent to various museums in celebration of the advance represented by the ENIAC.

In 1997, a team at Penn reconstructed the ENIAC, preserving the original architecture, onto a microchip as a tribute to the advances since the first computer. The microchip reproduction did not, however, preserve the original programming interface but instead required a software interface to program the chip. This paper describes a new implementation of the ENIAC on a series of FPGAs (Field Programmable Gate Arrays) to be controlled by panels similar to those used in the original ENIAC. FPGAs are logic devices that can be programmed using either graphical schematics of logic gates or VHDL code (described below). For this project, a Xilinx Spartan IIe FPGA was programmed using VHDL code and the ISE 5 software package. The FPGAs were interfaced with a custom-designed printed circuit board (PCB) for the input and output functions. The project goal is to preserve as much of the original structure as possible while maintaining the original user interface.

1.1 Basic ENIAC Architecture

A computer's architecture encompasses all the decisions made in the design process, such as number format, program structure, and data transfer system. The ENIAC had 30 individual units coordinated by digit and pulse trunks, equivalent to modern data and address buses. While modern architectures use a binary format to express numbers, the original ENIAC used decimal numbers. Numbers were transmitted

as pulses over the digit trunk, with one physical wire or line per digit. The digit trunk had 11 lines (wires), with 10 wires were for digits and a single wire or line for number sign information. Twenty of the units were accumulators; they could add or subtract a number from the number previously stored in the unit. Other units included a master programmer, initiating unit, function generators, multipliers, a divider and square rooter, a cycling unit, and input and output devices. The accumulator, cycling unit, and constant transmitter are discussed in detail in Section 2.

1.2 VHDL

VHDL is the acronym for Very High Speed Integrated Circuit (VHSIC) Hardware Description Language. It is a hardware description language supported by software packages such as ISE to design circuitry for implementation on logic devices. Logic devices are used to hardwire a program rather than implementing a software program on a general purpose platform. Logic circuit layouts were originally designed by hand and then built from individual components. The design process then shifted to drawing schematics with logic gates on a computer, and VHDL was eventually introduced to allow the user to program a logic device (FPGA in this case) using a low-level programming language. The VHDL compiler converts code into an arrangement of logic gates that the computer can then program onto a logic device. VHDL code is clean, portable, fast to design, and easy to simulate.

The top layer of a system designed using VHDL contains port statements for each entity that makes up the system. The output of one entity is connected to the input of another through the assignment of both the input and output to the same signal. Signals are declared within entities (or within the top-level port statement) and act like wires between components. Port statements allow entities to be connected to each other and to outside inputs and outputs.

Within an entity, there is a port statement and an architecture. The entity port statement declares inputs and outputs. The entity architecture contains the code that affects entity outputs. Since VHDL is a hardware description language, it acts like a circuit: all lines of code are run concurrently. Sequential statements, like those in most software programming languages, can be run within processes. Processes are updated each time that one of the variables upon which they are dependent changes, and generally contain a series of if statements. If statements can be dependent upon events in the variable value; they are run once when a change is detected in the variable. If a variable is used on the rising or falling edge, it is considered a clock. Each process can check for events in only one variable (although it can check the value of many variables), and variables can be modified within only one process.

1.3 Project Goals

The design constraints for the ENIAC reconstruction require maintenance of the spirit of the original architecture. Each of the original units of the ENIAC is designed in a separate VHDL file, to be implemented on a separate FPGA chip. FPGA chips have a

limited number of input and output lines, and to minimize the lines required, a cycling unit is implemented on each chip. The control unit produces a synchronizing clock and pulse, which are transmitted to each unit for pulse generation. The 10 different signals produced in the cycling unit are, as in the original ENIAC, used to control different processes. The project's final product will be a constant transmitter, cycling unit, initiating unit, and three accumulators implemented on FPGAs with an interface of switches and dials very similar to the original ENIAC. The paper below describes implementation of a clock divider, control unit (part of the initiating unit), cycling unit, and accumulator on a single FPGA as a preliminary step in the project.

2. IMPLEMENTATION

The design implemented on the FPGA has nine components. The clock divider slows down the oscillator included on the Digilent Digilab Board. The control unit starts and stops the clock provided to the cycling unit based on external operation mode settings. The cycling unit generates the pulse train. The start up unit reads in switches, saves their settings to RAM, and takes care of other output functions. The receiver updates digit values based upon received pulses and switch settings, which are displayed by the LED unit in real time. The sign unit keeps track of the current sign and the transmitter converts the values stored in digit registers to pulses on the digit trunks. Interaction between components is shown in Figure 1 and individual components are described in detail below.

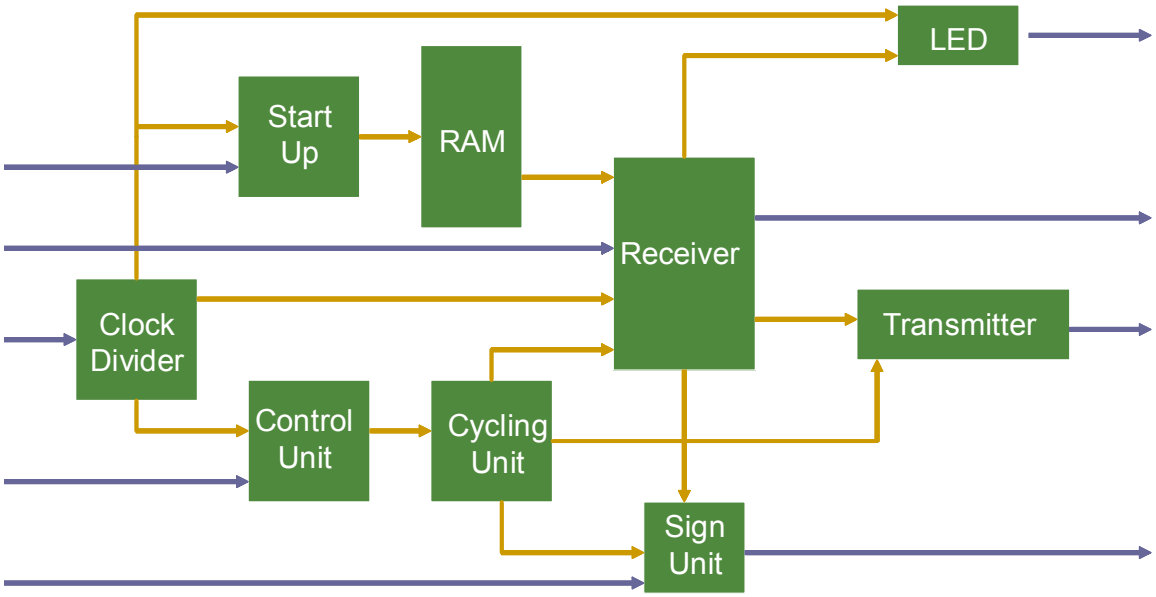

Figure 1: Interaction between units of the accumulator. Yellow lines are internal connections and blue lines are input and output lines.

2.1 Clock Divider

The ENIAC ran at 5000 addition cycles per second, or about 100 kHz, where one addition cycle is 20 pulses long. The Xilinx IIe FPGA is mounted on a Digilab II XL board with a 50 MHz oscillator. To make the LED display easier to follow, the clock

was slowed down to 10 pulses per second with a clock divider. The clock divider uses a counter to slow down the clock and preserves a 50% duty cycle.

2.2 Control Unit

The original ENIAC had one cycling unit that provided a pulse train to all other units. To minimize input and output connections between chips, the FPGA implementation generates a pulse train in each unit. The units are synchronized using a control unit that outputs the clock used by the cycling units in each accumulator, based upon the operation mode. The ENIAC has three operation modes: continuous, addition cycle, and pulse mode. Continuous mode is most commonly used for actual computations, while the other two modes are for debugging purposes. Pulse mode proceeds pulse by pulse, and the user must push an advance button between each pulse. Addition cycle mode runs for 20 pulses and then waits for the user to push the advance button before proceeding to the next 20-pulse cycle. The control unit starts and stops the output clock based upon these three modes and also outputs a synchronizing pulse once per addition cycle (at the same time as the cpp, central programming pulse) to ensure all units are at the same place in the cycle. The control unit uses a combination of "if" statements and counters to operate in the correct mode and output synchronization pulses at the correct time.

2.3 Cycling Unit

The ENIAC's cycling unit generates a pulse train with 10 individual signals. The pulse train is available to each unit of the system, and each signal acts as a clock to trigger internal operations. To minimize input and output connections in the FPGA implementation, each unit generates its own pulse train, using a synchronized clock, once per cycle pulse from the control unit (described above). The cycling unit is a state machine with 100 individual states, divided into 20 main states with five sub states each. The 20 main states are based upon the partition of the original ENIAC's pulse train into 20 time divisions, as shown in Figure 2 below. Pulses do not, however, necessarily occur at the beginning of one of these 20 divisions. The further division of each main state into five sub states provides the necessary resolution to produce a pulse train identical to the original. Pulses are generated by setting signals high during some sub states and low during others.

The cycling units on each unit are not identical to that of the original ENIAC. Since the operating modes are taken care of by the control unit, the individual control units do not have operating mode controls built into them. They do, however, look for the synchronizing pulse from the control unit. The synchronizing pulse arrives at the same time that the cycling unit is generating the cpp, and the cycling unit waits until the reception of the synchronizing pulse before proceeding. The accumulator does not require four of the clocks produced by the cycling unit (4P, 2P, 2'P, and 1P), and space constraints prompted their removal from the cycling units designed for the accumulators. The constant transmitter will, however, require these four clocks, and the cycling unit for

the constant transmitter contains those pulses. The clocks used in the accumulator are described below:

- 10P – The tenp clock cycles through each digit register during digit transmission.
- 9P – The ninep clock is used for digit transmission, reception, and sign unit operation. The ninep clock is gated to transmit the correct number of pulses in the transmitter, is monitored to check for incoming pulses in the receiver, cycles the sign unit when the number being received is negative, and is gated to transmit ninep pulses on the sign line of the digit trunk when a negative number is transmitted.
- 1'P – The onepprime clock is used to correct numbers sent in nine's complement. It is either added to the least significant digit of the number being transmitted by the transmitter, or to the unit's digit upon reception if the clear correct switch is set for the current program.
- cpp – The cpp, or central programming pulse, synchronizes program flow. It is generated by both the cycling unit and the control unit. Since the control unit is specific to the FPGA implementation, it was not used in the original ENIAC. The cpp generated by the control unit is called "synchpulse." The cycling unit on each accumulator waits for the synchpulse before generating its own cpp. This keeps all cycling units synchronized. The cpp generated by the cycling unit also waits for reception of a cpp from the last active unit. The line on which this cpp is received indicates which program to use, and ensures that each unit waits for the last operation to complete before beginning a new operation.
- ccg – The ccg, or clear correct gate, lasts seven pulse cycles and allows carries to occur after receiving pulses.
- rp – The rp, or reset pulse, is used to reset various registers and prepare for the next cycle.

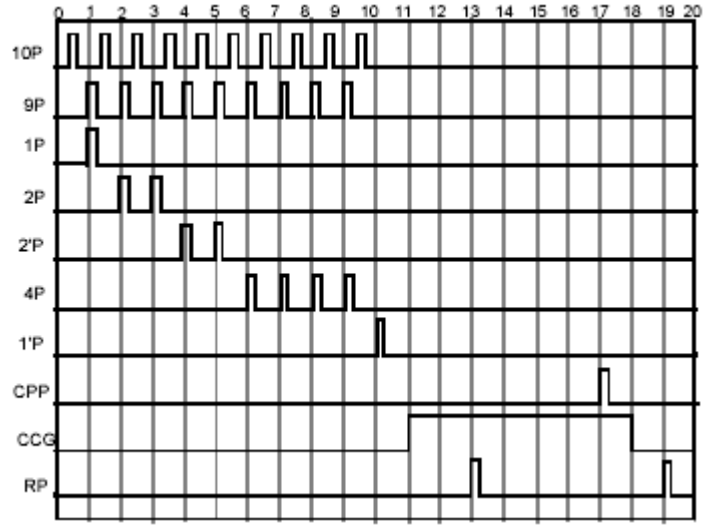

Figure 2: Pulse trains [1].

2.4 Accumulator

The original ENIAC used a set of 10 vacuum tubes for each of 10 digits. The unit is initialized with the first tube on and all others off, to represent zero. If a digit pulse is received, the first tube is turned off and the second is turned on, to represent a 1. The FPGA implementation uses a 10-bit register to represent these 10 vacuum tubes; zeros represent off and ones represent on. Numbers are transmitted between units on a digit trunk. The digit trunk has 11 wires: one for each of the 10 digits and one for the plus or minus setting. Digits are transmitted using the corresponding number of pulses; a 3 would be transmitted by sending three pulses. The plus or minus line transmits nothing if the number is positive and transmits 9 pulses if the number is negative.

The accumulator has 34 switches on a panel similar to that shown in Figure 3. The significant digit switch goes from 0 to 10 and allows the user to select how many significant digits the accumulator uses. If the selective clear switch is turned on and the accumulator receives a selective clear signal from the initiating unit, the accumulator resets its digit registers according to the significant digits setting. Twelve programs can be set on each accumulator. Each program has an operation switch, which can be set to receive from one of five ports, transmit from one of two ports (or both), or do nothing. Each program also has a clear correct switch. If the clear correct switch is set, the accumulator will send the 1'P pulse to the unit's digit if in the receiving mode and clear the accumulators at the end of cycle if in the transmitting mode. Eight of the programs can be set to repeat up to nine times.

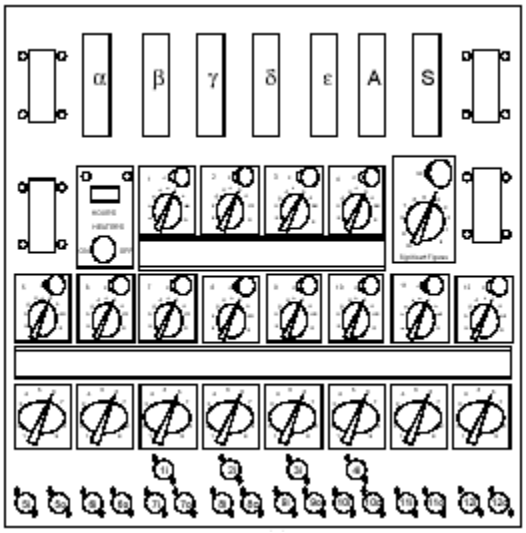

Figure 3: Accumulator panel [1].

The FPGA implementation of the accumulator has five components. The start-up component executes the initialization cycle by reading in switch settings, setting the RAM (random access memory), and resetting registers. The receiver shifts digit registers as digit pulses are received. The transmitter produces digit pulses according to the values stored in the digit registers. The PM Sign Unit keeps track of the sign of the stored number (plus or minus), and the LED component continuously updates the LEDs to

reflect the number currently stored in each digit register. Interaction between the units is shown in Figure 1.

2.4.1 Start-up

When the FPGA system is turned on, the start-up unit runs an initialization cycle. The initialization cycle stores the setting of each switch into memory. The start-up unit runs off the clock from the control unit (not one of the clocks generated by the cycling unit). To read in switch settings, switches are scanned using a state machine with two states per switch. The first state sends out the address of each switch and the second state reads in the corresponding setting. The switch addresses are encoded and pass through an off chip decoder, activating the correct switch. The setting is received through a data bus that is used as program pulse input lines after the start-up cycle.

The selective clear and significant digit settings are stored in registers available to other accumulator components. Program settings are saved in a 9-bit word. Operation switch and repeat switch settings are encoded before saving to decrease RAM size. The first bit represents the clear correct setting, the next four bits represent the operation switch setting, and the last four bits represent the repeat switch settings. The resultant word is saved into RAM at the address corresponding to the program number (for example, settings for program one are saved to RAM address “0001”).

The off chip decoder is also used to send program pulse outputs and select one of 4 digit trunks. If the current program has completed, the receiver sets the done flag high. The start up unit monitors the done flag, and emits the program pulse output on the correct line to the off chip decoder during the cpp. When the done flag is not high and the switches have been read in, the start up unit monitors the receive from register and sets the corresponding line of the off chip decoder high, allowing the selection of the correct digit trunk.

2.4.2 RAM

The random access memory (RAM) is dual port block RAM generated using the Core Generator feature of the ISE software package. Port A is used to write to RAM after setting the write enable bit high, and port B is used to read the RAM during accumulator operation. The RAM has 13 addresses and each word in RAM is nine bits long. The RAM address corresponds directly to the program being run. Words are described in 2.4.1: the first bit corresponds to the clear correct setting, the next four bits to the operation switch setting, and the last four bits to the repeat switch setting.

2.4.3 Receiver

The receiver performs a variety of actions based upon the current place in the pulse train and various register settings. The basic purpose of the receiver is to increment digit registers as pulses are received. When the reception of a number causes a carry (a digit is at 9 and a pulse is received), the receiver increments the appropriate digit. The

receiver saves the values in the digit registers so that they can be transmitted on the next cycle or a number can be added to or subtracted from them. Register clears according to significant digit settings can occur in several different cases in the receiver, and a few other housekeeping occur in the receiver. It is the largest component of the accumulator.

The receiver contains four processes. One process uses the cpp as a clock and is responsible for decrementing the repeat register and correctly setting the done flag. One process runs off the ninep clock and sets a flag whenever there are incoming digit pulses. The other two processes run off a faster clock, called ledclock, which was originally designed to run the LEDs. Since each process can monitor for events in only one clock, it was more convenient to run these two processes off the ledclock and use the other clocks as flags more than clocks. One of these processes sets flags to take care of carries generated when adding numbers in the accumulator. The other process contains the actual digit value registers and all other functionality. The process increments digit values when flags are set by either of the other two processes and runs various special functions.

The receiver uses the following signals as clocks: the cpp, registerreset, onepprime, ccg, rp, and ninep. The current value of each digit, receive flag, and transmit flag outputs to other units are updated independent of any clock.

The receiver stores the current value of each digit. The numbers are stored as 10-bit registers and numbers transmitted as pulses over a digit trunk as described above. The pulses are transmitted on the ninep clock. Each time the ninep clock goes high, one process in the receiver checks each line of the digit trunk. If the line is high, a pulse is being transmitted. The receiver sets “pshift” high, indicating that there is a shift due to a pulse. It also checks the value of the last bit of the corresponding digit register. If the last digit is high, the value currently stored is nine, and the pulse input should cause a carry to the next digit. The corresponding flag bit is set.

Carries, as necessitated by flags set while transmitting, are preformed while the ccg gate is high in their own process. The process is run on the rising edge of the ledclock and runs from the least significant digit to the most significant digit using a state machine. If the flag for a particular digit is high, “fshift” is set high. The last bit of the digit is checked, and if it is high, “cshift” is set high for the next digit. If “cshift” must be set high, the next digit’s last bit is checked, and if it is high, the next “cshift” is set. The process continues until no more carries are triggered. “fshift” is used for carries triggered while receiving pulses from the digit trunk, while “cshift” is used for carries triggered while implementing those carries. Flags from each state are reset in the next state.

Each bit of all three shift flag registers are “or”ed together outside of any process. If any of the bits are high, then the final shift register will have a high bit. The main process increments the value stored in the registers based upon this final shift register by shifting the first nine bits of the digit register to the right and moving the tenth bit around to the first bit. The shift simulates the ring counters used to store and

increment digits in the original ENIAC. The main process also controls all of the special functionality of the accumulator.

When the cpp is high, the receiver component resets flags, loads the next program from memory if necessary, and sets the registerreset flag as necessary. The registerreset flag is set if the current program is a transmit cycle and clear correct is set, the accumulator's selective clear switch is set and a selective clear signal is received from the initiating unit, or the initialization cycle is being run. If register reset is high, each digit is reset according to the accumulator's significant digit setting. These settings occur within the main receiver process. Program repetitions are monitored in a small separate process that uses the cpp as a clock. On a cpp event, the repeat register is decremented. When the register reaches 0, the repeat register resets and sets the done flag high.

If clear correct is set high and the unit is receiving, the unit's digit is incremented by one on the 1'P clock. Negative numbers are incremented using nine's complement, but the accumulator works in ten's complement. The transmitted signal is usually corrected to ten's complement according to the significant digits setting, but in some cases this will not occur and the clear correct switch is set preemptively. In this case, the unit's digit is incremented (no matter the significant digit setting) and so the option is ineffective unless the significant digit setting is 10.

When rp is high, the data loaded from memory during cpp is distributed as appropriate. The data is stored as a 9-bit word, which then must be split into three pieces (clear correct, operation mode, and repeat switch setting). The first bit of the word is the clear correct setting, the next four bits are the encoded operation mode setting, and the last four bits are the encoded repeat switch setting. Flags receive, transmit, aout, and sout are set according to the operation mode setting (aout and sout signal the output through the A or S ports, for addition or subtraction). Register receivefrom is also set according to the operation mode setting; if the accumulator is set to receive, then one of five receive ports (alpha, beta, gamma, delta, or epsilon) must be chosen. The null operation (O on the accumulator panel) is performed by setting both receive and transmit flags low. The encoded words for each operation mode are shown in appendix 1. [3]

2.4.4 Transmitter

The main function of the transmitter is to turn the zeros and ones in the digit registers into a set of pulses to transmit on the digit trunk. Each digit register is shifted to the right on the tenp clock until the '1' in the register is reached. At that point, a transmission flag is set and shifting continues to ensure that the number in the register returns to its original state. The transmission flag is combined with the ninep clock so that the number is correctly transmitted in nine's complement. If the number is positive, the nine's complement equivalent is just the number, so a 3 would be transmitted as three pulses. However, on transmission of a negative number, nine minus the digit value pulses are transmitted, with one pulse added to the least significant digit on the onepprime signal; it is thus transmitted as 9999999999 – the number + 1. In some cases, addition of the onepprime pulse is missed by the transmitter. Programmers set the clear

correct switch on the receiving accumulator in anticipation of this mistake. This the only function preformed by the transmitter.

2.4.5 PM Sign Unit

The plus-minus unit tracks the sign of the number stored in the accumulator. On transmission of a negative number, it sends nine pulses on the sign line of the digit trunk. The nine pulses change the sign of the receiving plus minus unit nine times, so the sign is flipped upon completion. The unit gates the ninep clock so that no pulses come out on the sign line if the number being transmitted is positive. If the most significant digit in the accumulator generates a carry, generating overflow, the sign unit flips. The overflow could be due to a true overflow, but is more likely the result of a subtraction using tens complement. In the case of a subtraction, the correct answer is produced by switching the sign of the number. If the sign of an operation is unexpected (for example, adding two numbers produces a negative number), a true overflow occurred. [3]

2.4.6 LEDs

The LED display serves no purpose in the actual accumulator function. It is, however, an important debugging tool, and a similar display, using lamps behind halved ping pong balls, was present in the original ENIAC. The LED display is updated in real time with the current value of each digit register. Each column of the display represents a digit, and each row represents a value from zero to nine. The LED unit scans through each column and determines the value of the corresponding digit. It then selects the row corresponding to that value. The column and row address are then output to light the correct LED. To prevent the row value being output before the column value, or vice versa, the column value is set to a dummy address, the row value is set, and then the column is set correctly. The LED unit runs at a clock 10 times fast than the cycling unit, so that all 10 digits (columns) can be updated during each cycle and the displayed digit is the value currently stored.

Two additional LEDs were added to the board, but are not controlled by the LED unit. One LED lights up when the number stored in the accumulator is negative, similar to a light on the original ENIAC. The other LED lights up when a program output pulse is emitted and is included for debugging and demonstration purposes. These LEDs direct outputs of the sign and start up components, respectively.

2.5 Constant Transmitter

The constant transmitter allows the programmer to load a constant value directly to an accumulator by setting a variety of switches. The value indicated by reading in the switches is then generated using a combination of the 1P, 2P, 2'P, and 4P pulses. The constant transmitter can be generated using combinational logic in VHDL.

The constant transmitter has two panels. One panel is used to choose the digit output terminal and the other to choose the constant. The constant is chosen by setting

between five and twenty dials. Each dial has the values zero through nine and represents a digit of the constant. The dials are arranged in groups of five, and the number to be transmitted can be composed from some combination of those groups. [1]

3. IMPLEMENTATION ON CHIP

The final implementation will be on a series of FPGAs with large panels similar to those of the original ENIAC. To test the work performed thus far, a PCB was designed to interface with a Xilinx IIe chip mounted on a Digilent Digilab II XL circuit board by Zheng Yang of the University of Pennsylvania. FPGA pins were assigned using the Digilent instruction manual and addresses for switches currently on the board are shown in appendix 2. [4]

The PCB board is designed for testing purposes, and does not include the functionality of the full accumulator panels. The board has four program mode switches, four clear correct switches, and two repeat switches, so it can run four different programs, two with repeat capabilities. It has two transmit ports (for each addition and subtraction, like the original ENIAC) and four receive ports (as opposed to the ENIAC's five). It has ports to transmit and receive coordinating cpps. There is a significant digits dial (with choices from 0 to 9) and a selective clear switch. Nine LED banks with 10 LEDs each display the value currently held in the accumulator. Two additional LEDs give the current sign of the accumulator value and the current status of the program pulse output line. The PCB board is shown connected to the Digilab board in Figure 4, and the I/O structure is shown in Figure 5.

The Spartan II FPGA has 96 input and output pins, but the full ENIAC accumulator panel requires over 100 input and output lines. Using encoded signals and a shared data bus drastically decreases the number of lines, and thus the number of pins, required to implement the design. The decoder shown in Figure 6 performs the switch selection, digit trunk input selection, and program pulse output depending upon the current cycle. The decoder takes a 4-bit binary address and converts it to one of 16 choices.

During the start-up cycle, the start-up unit scans through the addresses of all the switches and outputs them one at a time through the decoder. When a switch's address is output by the decoder, that switch becomes active. The switch's data then floods the data bus and is input to the FPGA. When the next switch address is selected, the last switch becomes inactive, the new switch becomes active, and the correct data is received. The significant digits switch, four operation mode switches, two repeat switches, and four clear correct switches are accessed in this manner. The significant digits switch has 10 options, the operation mode has eight options, and the repeat switches have 10 options; their outputs are encoded to four bits within each switch. Each clear correct switch has only two options (high or low), but all four are accessed at the same time and so require all four input lines. The selective clear switch does not use this system, but is directly input to the FPGA as it has only two options (high or low).

During receive cycles, the receiver unit of the accumulator pulls down the current program's settings from RAM and determines the digit trunk from which to receive digit pulses. It then outputs the address of that digit trunk to the decoder. The decoder sends an output enable signal to the correct digit trunk based upon that address. The output enable signal allows the data from that digit trunk to reach the digit trunk data bus. All four receive digit trunks use the same pins into the FPGA. However, only one output enable can be active at a time, and thus only one digit trunk's data arrives per cycle. Two pins output program output pulses at the end of a program to activate the next program; there are only two program output pins, as only programs with repeat switches send the program output pulses. The program pulse output pin for program four has been connected to an LED for debugging and demonstration. The decoder also has two empty output lines; one of these lines must be selected when none of the other functions are desired.

The data bus for the switches serves an additional purpose. After the switches have been scanned and their values stored, the buffer shown in the figure is activated, allowing the program pulse input lines to use the data bus. When a program pulse input is received on one of the four lines, it uses the data bus. The LEDs have eight output lines: four for column selection and four for row selection. There is also an LED which is on when the accumulator value is negative and off when it is positive. The additive and subtractive digit trunk outputs each have 10 lines, 9 for digits and 1 for the sign.

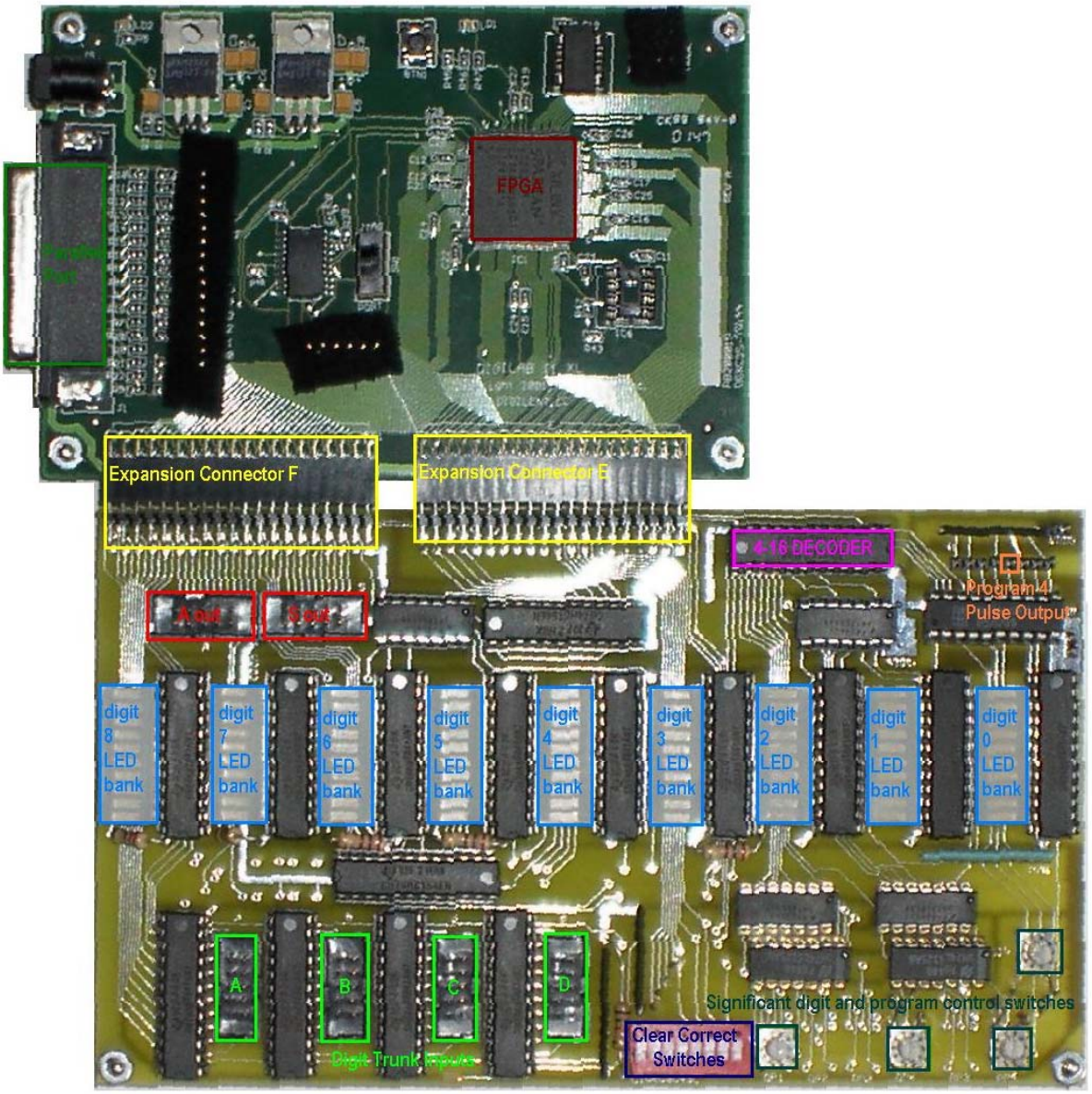

Figure 4: The board upon which the accumulator and cycling unit were implemented.

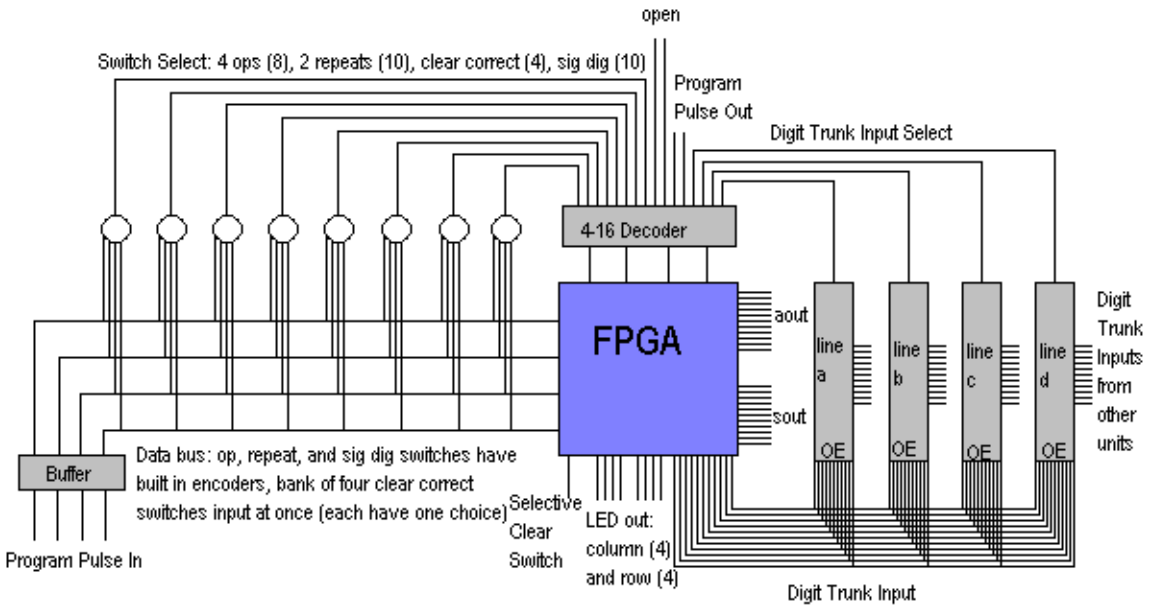

Figure 5: Connections between the FPGA and other devices.

4. RESULTS

The accumulator is fully functional in logical simulations, as shown in Figure 6. Timing simulations and actual implementation upon the board are less successful. The first units implemented on the board were the LED unit and clock divider. The LEDs can be directly loaded with a number by replacing the connection with the receiving unit by a constant value. During this implementation, the clock divider was used to decrease the FPGA device. Slowing down the clock is very useful for debugging.

The cycling unit and receiver were implemented next. The cycling unit is required for operation of all units other than the LED unit. The accumulator was first set to display a certain value and then reset itself at the end of each addition cycle, to ensure the unit was functioning. The accumulator was then set to receive nine pulses per addition cycle in the unit's digit by tying the digit trunk to "00000000001," where the first bit represents the sign line, and the next 10 bits represent the 10 digits from digit 9 down to digit 0. The accumulator was thus expected to count by nines from zero to 9999999999.

Counting on the unit's digit was relatively simple to debug. The carry operation was not as successful. The original design used a second set of ring counters to increment digits when flags were high during the ccg. This design was very similar to that of the original ENIAC, which simply let the carries run through the digits until the value settled. On the FPGA, however, the design created race conditions, and values did not have time to settle before they were tested. A single flag tended to cause the next digit to increment twice, and carries that triggered additional carries were not recognized. The current receiver layout presented the solution to this flaw. When counting by nines, the accumulator functions smoothly up to the transition between 999 and 1000. Each

time that a carry is triggered in the thousand's digit, the accumulator stops for approximately seven minutes. It then resumes and runs smoothly, with the correct answer for the last operation. The accumulator was run for a long period of time to observe the transition from 9999 to 10000. The transition again took approximately seven minutes.

The start-up unit was tested by reading the value of the switches into memory and setting the LEDs to display values corresponding to the contents of RAM. The original design led to race conditions between selecting a switch and reading the corresponding input on the data bus, but using a state machine to step between choosing switches and reading inputs eliminated the problem. The sign indicator LED and program pulse output LED were added to the PCB to check for further functionality. Subtraction was simulated tying the digit trunk input to "10000000001." The first bit is '1,' so the sign of the number is negative. The number being received is thus -9 (since the pulse in the unit's place is received 9 times per addition cycle). The LEDs correctly displayed the value in the accumulator as 9 and the sign as negative. The clear correct switches were tested by setting the digit trunk to "10000000000," such that the unit was receiving "negative zero," while the clear correct switch was high. The accumulator incremented the unit's digit once per cycle on the onepprime pulse, as expected. The program pulse output functionality was tested by setting the repeat switch to various values and monitoring the program pulse output LED; it flashed after the correct number of program repetitions.

The sign LED and program pulse output LED provided unexpected debugging information. A negative sign is represented by nine pulses during the first half of the addition cycle (on the ninep clock), with the sign flipping each time a pulse is received. The sign thus flips on the ninep, and the sign LED flashes on the ninep. When the accumulator was allowed to run to the point that the seven minute delay occurs, the digit LEDs again stopped. The sign LED, however, continued flashing. The program pulse output LED continued flashing at the correct times. The accumulator, then, continues functioning while the logic delay occurs.

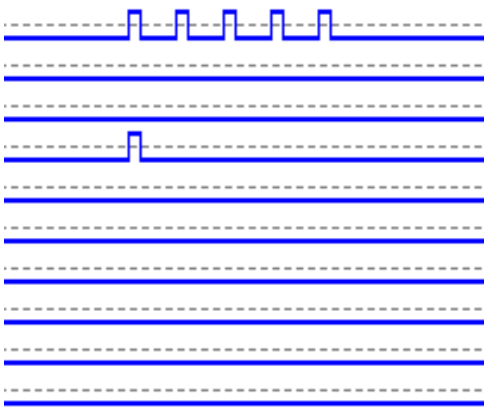

Figure 6: Transmitter outputting pulses in a logic simulation. The value in the accumulator was obtained by adding 6 to 999, and the output of 1005 on the digit trunk is shown, with the least significant digit line shown on top.

5. DISCUSSION AND FUTURE WORK

Difficulties experienced in the project stemmed from the adherence to the original pulse train and resultant timing issues. Running off the pulse train increased program complexity and decreased productivity. The original ENIAC was more efficient and fail proof than this implementation. The FPGA chip used for implementation was a tight fit, and a larger chip would improve system performance.

Space constraints were a constant limitation. The FPGA used has 30,000 logic gates; devices with up to 500,000 gates are commonly available. The code was optimized several times to fit onto the chip. The optimization included converting behavioral statements (if…else statements) into combinational logic statements and reducing the number of registers used in the code. The carry operation was originally implemented in a separate entity than the receiver, but the two entities were combined to reduce size. The optimization significantly reduced program size, but the chip still did not contain sufficient space. After changing compiler preferences to emphasize optimization during the translation and mapping processes, the entire implementation fit onto the desired FPGA.

The problems experienced during the counting trial suggest timing will be an issue. The reason for the seven minute delay after 110 addition cycles is unclear. The sign and program pulse output LEDs show that the cycling unit, sign component, and even some of the receiver continue functioning. When the significant digit setting is set so that the second least significant bit is five upon start up, the seven minute delay occurs after 110 cycles, not at the value 993. The delay appears unrelated to the cycling unit or carry operation. The delay will cause major synchronization problems with the other units as the program pulse output continues during the delay.

Implementation with the full input and output capabilities will require several changes to the program. The start-up unit, in which the switches are read, needs several changes for the full implementation. It will need to be redesigned for the encoders, decoders, and I/O structure of the full panels. The words stored to RAM are designed for the full implementation, so RAM and the word convention associated with it will not need to be modified. The receiver decodes information from RAM, and assumes the full data format that will be needed for the full unit. Addresses will change with the full implementation, but the current format should be sufficient for the required increased input and output.

The final product will include a system of three accumulators and a constant transmitter. The constant transmitter was not designed for this report. The cycling unit designed for the accumulators does not contain the 4P, 2P, 2'P, and 1P pulses that are unique to the constant transmitter. However, the first cycling unit was designed to work for any unit and thus includes these pulses. The constant transmitter should be a relatively simple piece to design and implement. The interaction between three accumulators will need to be thoroughly tested and debugged, as it has been performed only in simulation for this report.

6. CONCLUSIONS

The work described in this paper achieved the original goals for the summer project. Two accumulators were not hooked together, preventing full testing, and debugging may still be necessary for the completed units. The project proved more difficult than expected due to timing issues and unfamiliarity with the Xilinx platform. Learning the architecture of the original ENIAC took a significant amount of time, but the architecture is relatively straightforward. The architecture was designed for the hardware available at the time. Implementing it on modern hardware was difficult, as the concept of a system running off a set of 10 clocks is unusual and the FPGA is not designed to run on more than one clock. Significant progress toward the final product was made on this project. The final product will provide an excellent learning opportunity for people to experience how the first general purpose computer was programmed.

7. ACKNOWLEDGMENTS

I would like to take this opportunity to thank my advisor, Professor Jan Van der Spiegel, and Zheng Yang, the graduate student who helped me with this project, including designing and building the PCB on which I implemented my work. I would also like to thank the National Science Foundation for providing funding for my project through the SUNFEST REU program. Finally, I would like to thank my fellow SUNFEST student, Nicole Dilello, and my lab partner from previous projects, Seth Jacobson, for their advice in various stages of the project.

8. REFERENCES

1. J. Van der Spiegel, J. Tau, T. Ala'ilima, and L. P. Ang, The ENIAC: History, Operation, and Reconstruction in VLSI, *The First Computers—History and Architectures*, MIT Press, eds. R. Rojas, 2000, p.121-178.

2. J. Tau, ENIAC on a Chip: The Monolithic ENIAC, Master's Thesis, Philadelphia, PA, University of Pennsylvania School of Engineering and Applied Science, December 1996.

3. A. Goldstine, Technical Description of the ENIAC, Part I, Philadelphia, PA, University of Pennsylvania Moore School of Electrical Engineering, June 1, 1946.

4. Digilab 2 XL Reference Manual, Pullman, WA, Digilent, Inc., May 7, 2002, p. 6.

9. APPENDICES

APPENDIX 1: OPERATING MODE ENCODING

Switch Value	Port	Mode	Encoded
0	None	Null	0000
1	Alpha	Receive	0001
2	Beta	Receive	0010
3	Gamma	Receive	0011
4	Delta	Receive	0100
5	Add	Transmit	0101
6	Subtract	Transmit	0110
7	Add and Subtract	Transmit	0111

APPENDIX 2: DECODER SWITCH ADDRESSES

Address	Function
0100	Clear Correct Switches
0101	Significant Digits Switch
1011	Program 4 Repeat Switch
1110	Program 4 Operation Mode Switch