

An extended length integer calculator implemented in an FPGA

Austin Beer

Department of Computer Science, Taylor University, Upland, Indiana 46989

Introduction

Almost all handheld calculators in use today have a limit on the size of the numbers on which they can operate. Simple personal calculators are limited to integers no greater than 8 or 10 digits in length. The high-end TI-89 can handle integers all the way up to 614 digits¹. However, no standalone calculator in use today can truly be said to be able to handle integers of virtually unlimited length. That is the goal of this project.

This project is based upon one I completed my freshman year here at Taylor. For the class "Data Structures" I wrote a C++ based calculator that could add, subtract, multiply, divide, and modulo integers of arbitrary length. The size of the numbers was limited only by the amount of available memory. Now that I am a senior I decided to solve the same problem again, but instead of using software I wanted to implement the solution in hardware using a field programmable gate array (FPGA).

Specifications

- Accepts and returns only **integer** numbers (both positive and negative).
- Input and answers are limited to **524,272** digits each (limit imposed by the size of available memory).
- Performs **addition, subtraction, and multiplication** in hardware (implemented and tested in hardware).
- Performs **division and modulo** in simulation (coded in VHDL and tested in simulation). These functions cannot yet be implemented in hardware due to size limitations of the current Spartan 3 FPGA.
- Entering and displaying numbers is user friendly and mimics the operation of an average desktop calculator.
 - Shift keys are available to shift numbers left and right that exceed the display size of 18 digits.
 - A backspace key (BACK) is available to easily erase the last digit entered.
 - Both clear current entry (C) and clear everything (CE) keys are available.



Figure 1. Display in operation

Bibliography

1. TI-89 / TI-92 Plus Guidebook
2. Spartan 3 Data Sheet <<http://www.xilinx.com/bvdocs/publications/ds099.pdf>>
3. Memory Data Sheet <<http://www.issi.com/pdf/61LV25616AL.pdf>>
4. Spartan 3 Starter Board User Guide <<http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-rm.pdf>>

Hardware

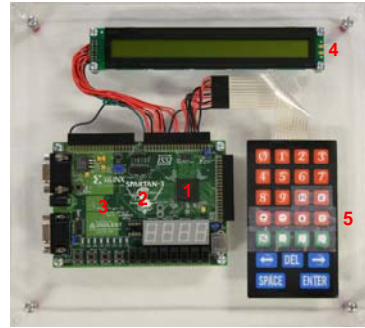


Figure 2. Fully assembled calculator

(1) Field Programmable Gate Array (FPGA)

- Xilinx Spartan 3 (XC3S200)²
- Can be programmed to represent virtually any digital logic circuit.
 - Uses look up tables (LUTs) and programmable routing to implement digital logic functions.
 - Reprogrammed at each power-up from an external flash chip.
 - 200,000 System Gates, 216 Kbits Block RAM, 173 User I/O Pins
 - 50 MHz Operating Speed (via external clock)
 - Built in Digital Clock Managers (eliminates clock skew)
 - JTAG Programmable (via external header)

(2) Memory (underside of PCB board)

- ISSI CMOS 512 KB Static RAM (61LV25616AL)³ x 2
- 10 ns access time with 16 bit bidirectional data buses

(3) PCB Board

- Digilent Spartan 3 Starter Board⁴
- Contains all components mentioned above, plus:
 - 4 7-segment LEDs and 8 individual LEDs
 - 8 push buttons and 8 switches
 - PS2, VGA, and RSA-232 connectors
 - Expansion connectors for FPGA pins

(4) Display

- Varitronix MDLS-20189-SS-LV-G
- Displays up to 20 5 pixel by 8 pixel characters.
 - Over one hundred pre-defined character patterns available.
 - Interface uses a single, 8 bit data register and a single, 8 bit instruction register.

(5) Keypad

- 25-key foil keypad
- 5 input lines and 5 output lines
 - Strobing used to determine which key is being pressed

Software

I decided to program the FPGA using **VHDL** (Very high level Hardware Description Language) both because a schematic-based design (using gates and wires) would take too long and because I wanted to gain experience working with VHDL.

I used **Xilinx's ISE WebPack** for my programming environment. This is a fully-functional but free version of their commercial software suite and includes software for every step of converting a VHDL, Verilog, or schematic-based design into an FPGA-friendly bitstream.

```
when 20 => -- decide whether or not to add reg1 to answer
if bit_index > 31 then
bit_index <= 0;
cur_word <= cur_word + 1;
alu_reg <= 18; -- loop back and get next word
elsif multdiv_data(bit_index)='1' then
carry <= '0';
next_reg2 <= 21; -- do shift after addition
alu_reg <= 50; -- start addition
else
alu_reg <= 21; -- do shift
end if;
```

Figure 4. Section of VHDL code

Design Overview

Memory is divided up into four registers of equal size:

- Register 0** – BCD data, holds what is displayed on the LCD
- Registers 1, 2** – Binary data, holds the ALU operands
- Register 3** – Binary data, holds the ALU answer

I partitioned up the design into seven modules to facilitate coding and debugging:

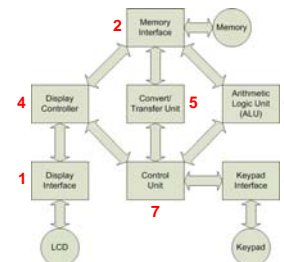


Figure 3. Functional block diagram

- (1) **Display Interface** – Converts the simple internal display commands into commands that the LCD can understand.
- (2) **Memory Interface** – Masks the timing requirements of the external memory bus and provides a simple interface for other modules to access memory.
- (3) **Keypad Interface** – Takes care of strobing the keypad to determine which keys are being pressed.
- (4) **Display Controller** – Controls what is displayed on the LCD and also what is contained in the BCD (binary coded decimal) register in memory. All commands that add and delete digits and shift and clear the display go through this unit.
- (5) **Convert/Transfer Unit** – Converts BCD numbers to and from binary and also moves binary numbers from one register to another.
- (6) **Arithmetic Logic Unit (ALU)** – Performs the actual arithmetic operations on two binary numbers.
- (7) **Control Unit** – Coordinates the activities of the above six modules.

Challenges

Fast BCD to binary and binary to BCD conversion

Binary Coded Decimal (BCD) numbers are represented using an individual 4-bit binary representation for each decimal digit. The process of converting a BCD number into a purely binary number is fairly straightforward. However, doing so on a 500,000 digit number with limited resources can take a very long time if not done correctly. I used a 64 bit internal data register and a staggered conversion scheme in order to achieve reasonable results (32 minutes worst case). With a larger FPGA it would have been possible to lower this to 1.5 minutes.

Limited size of the FPGA

While the interface modules did not take up a large number of gates on the FPGA, both the convert/transfer unit (CTU) and the arithmetic logic unit (ALU) required considerable real estate. In the end I discovered division and modulo could not be implemented in hardware because of this. A potential solution to this problem would involve redesigning the whole calculator around a 16 bit word instead of a 32 bit word. However, this would come at the expense of speed. Larger Spartan 3 FPGAs are available and so using one of these in the future would be a much easier and more long-term solution to this problem.

Conclusion

The primary goal of this project was completed successfully and performed as required. Addition, subtraction, and multiplication on integers of up to 524,272 digits in size was made possible via a hardware implementation. While division and modulo did not make it into the final product, the VHDL code for these functions is tested and available.

Through this project, I learned a lot about FPGAs, VHDL, and hardware-level logic and interface design. Hopefully I will be able to use this knowledge in the future to successfully work on other projects that are even larger and more complex than this one.