**Creating the Plava Gaming Platform Using the Spartan-3E**

|  |  |  |  |
| --- | --- | --- | --- |
| Ashton Snelgrove  University of Utah  Address  Phone  e-mail | Jacob Sanders  University of Utah  Address  Phone  E-mail | Matthew Steadman  University of Utah  4318 Arden Ct  Taylorsville, UT 84123  801-783-0989  steadman@eng.utah.edu | William Graham  University of Utah  737 E 700 S  SLC, UT 84111  801-620-0266  U0527893@utah.edu |
|  |  |  |  |

**ABSTRACT**

In this paper, we will describe the work done in an attempt to create a generic console game platform, using the Xilinx Spartan-3E FPGA board, I/O devices, and other Additional Circuitry. We will then demonstrate a sample application of the platform, implementing a clone of the Nintendo Entertainment System game Duck Hunt.

**Categories and Subject Descriptors**

B.6.3 [**Verilog]**: Design and implementation through a Xilinx FPGA of a gaming platform – *Processor, GPU, digital sound reproduction*

**General Terms**

Design, Languages

**Keywords**

Spartan-3E, Processor Design, 24-bit color, GPU, Nintendo, Assembly language, NES-Light gun

# INTRODUCTION

The intent of this project was to create Plava, a 16-bit generic gaming platform that can be used to run and play video games. Plava was the name chosen by the Engineers, and is the Croatian word for Blue.

Plava is capable of generating 24-bit color images, can play up to ten sounds at the same time, and is compatible with the Super Nintendo controller, and the Nintendo Entertainment System’s light gun. Use of the light gun requires an external VGA to TV converter box that is capable of displaying the images generated by the GPU on a standard tube television.

Permission is granted to make digital or hard copies of this work for any educational or personal purposes provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice, all author names, and full citation on the first page. Written permission to otherwise copy or distribute this work must be obtained from any one of the authors, according to the contact information given.

# THE PLAVA PROCESSOR

The Plava processor was built to utilize a simplified version of the CR-16 instruction set. The processor contains sixteen 16-bit registers and an additional 5-bit program state register (PSR).

## Instruction Set

The Plava instruction set is a binary (two argument) instruction set. In most instructions the first input is the destination of the data following the execution of the instruction. The second instruction is an argument to the operation. There are two types of most instructions, a register based operation where two registers are given and the result stored in the first, and an immediate type instruction where a hard coded value can be processed in place of the secondary register.

There are a total of twelve basic arithmetic instructions, including addition, subtraction, and multiplication. Six of these are addition instructions. These include both register and immediate type instructions for a standard addition, an add with carry, and an add that does not affect the PSR. There are 4 subtraction operations, again both register and immediate versions of a standard subtraction, and a subtraction with carry. The other two basic arithmetic operations are a register and immediate type of multiply.

There register and immediate based and, or, and exclusive or operations comprising a total of six bitwise operations. There are also four left shifting functions that take the value in the first register and then shift it by the number stored in the second register or an immediate value. There are arithmetic and logical shifting functions, and shifting by a negative amount constitutes a right shift.

For getting and setting data values there are several useful instructions. The move commands allow you to move a value stored in a register or from an immediate hard coded value, into another register. Since there is a limitation in the number of bits available for an immediate value there is an additional instruction called load upper immediate which is similar to a move immediate, except that it loads the value into the upper 8-bits of the register rather than the lower 8-bits that would occur in a move immediate. The move immediate command will overwrite the 8 bits that it cannot reach with either 1’s or 0’s based on it’s sign so to move a full 16-bit immediate value into a register the move command and then the load command, which only overwrites the upper 8-bits, leaving the lower 8-bits alone.   
There is also a load and store command which enables you to load or store values from the system’s memory.

We decided to add two additional instructions to our instruction set outside of those recommended to us which were test, and not. Test is used to test for equality of values only, and not will perform a bitwise inversion on the value stored in a register.

The PSR was used to store information on the result of certain instructions. The five bits store whether the previous instruction resulted in a carry or borrow, if the second operand was lower than the first, if the flag was set, if the result of the operation was zero, and if the result was negative. Any branch or jump instructions following the instruction that set the PSR can decide whether or not it takes its jump or branch based on the content of the PSR.

The branch and jump commands enable choices to be made within the execution of programs and are the only conditional instructions in the instruction set. There are a total of sixteen mnemonics used with branches and jumps that signify a unique condition determined by previous instructions in that have affected the PSR. These mnemonics as well as their meanings are shown in Table 1.

## Architecture

The Plava processor architecture consists of a controller that interprets and fetches instructions, and an arithmetic logic unit (ALU) which processes the instructions. This two part design enables all instructions except the load instruction to be executed in two clock cycles. For the first clock cycle the instruction is fetched from current program counter location in memory and sets all appropriate multiplexer values in the rest of the processor. Then in the second clock cycle values are loaded from registers and sent into the ALU to be processed. The results from the ALU are actually written back during the fetch phase of the next instruction making the write back and instruction fetch stages pipelined enabling us to effectively execute 25-million instructions per second. The only exception to this rule is the load instruction, which requires three clock cycles. This is caused because the address being loaded from must be calculated, then the data loaded from that address, and finally written back during the following fetch phase.

### Arithmetic Logic Unit

The ALU-chip is the most complex piece of the processor and as such will be addressed individually. The ALU-chip is the place where all calculation is actually performed. It expects as inputs the values to be used in the calculation, an operation code, a function code, and a condition code; the latter three values are determined by the controller, which gets them from the instruction. It also reads the current values of the PSR. Its outputs are the result of the calculation, and the PSR write values as well as an enable value to write to the PSR. Each and every instruction (with the exception of load and store) has a unique output from the ALU which is in effect the core of the processor.

# MEMORY MAP

Describe the how the memory map does what it does, and the purposes of it

# GRAPHICS PROCESSING UNIT

Describe the GPU

# SOUND SYSTEM

Describe the sound

# USER INPUT

Introduce the Input devices (can be omitted if sub points are sufficiently detailed)

Table . Conditional Values for Jump and Branch

## Light Gun

Describe how it works and how to interface with the light-gun. Include a schematic of the circuitry to connect it with the system

## Super Nintendo Controller

Describe how the controller works and how to interface with it.

# PROGRAMMING

## Assembler

Describe the Assembler

## Demo Application – Duck Hunt

Describe the Duck Hunt design

# ACKNOWLEDGEMENTS

Provide any necessary acknowledgements.

# REFERENCES

Provide any references that we used to create the project.