Syzygy B100 - Hardware Specifications

Timothy M. Backus

2017-9-1

Contents

1 Overview

The Syzygy system was implemented on Digilent's Basys 3 FPGA for the purpose of creating a homebrew computer system from scratch, or at least as "from scratch" as possible without fabricating the chip on silicon. The Syzygy system includes a single-core CPU and 128kB of memory. In addition, the system supports a single SDHC card to boot from, a PS/2 keyboard for user input, and a VGA monitor for output.

2 CPU

The Syzygy B100 is a 16-bit RISC CPU built with programming ease as its focus. The processor has seventeen registers in total, with fourteen of them being directly writable. The ALU is capable of performing six different base operations, in addition to manipulating the bits of both inputs as well as the output.

This CPU runs at 33.3MHz, although it may be possible for it to run at a higher frequency. This clock speed was chosen because the block memory on the Basys 3 has a latency of one clock cycle, and the CPU expects a new instruction every rising-edge of its clock. All registers in the CPU are falling-edge triggered.

2.1 Special Registers

· R0: Instruction Register
  Holds the current instruction as a 16-bit value. It is written to when the clock phase signal is low, and releases its data if the clock phase signal is high. This register is not directly writable with an instruction.

· R1: Program Counter
  Holds the 16-bit index of the current instruction. It will increment its value on the falling-edge of its clock when the enable signal is high. Additionally, its value will be set to its data-in signal when the set signal is high. This register is only writable through R3 (Jump Address).

· FR: Flag Register
  This register holds a single 16-bit value where each bit corresponds to a flag signal. The only flag implemented is f_0, which is the signal to choose between fetching the instruction from memory rather than the boot ROM. This register is only writable through the "sys" instruction (see Instruction Set below).

2.2 Other Registers

· R2: Accumulator
  This is the only register that can be immediately set by one instruction

(push instruction). Any ALU instruction will overwrite this register's value with the result of the ALU instruction. Jump instruction's comparisons will also read this register's value.

· R3: Jump Address
Contains the address R1 (Program Counter) will be set to if the comparison to zero using R2's value is true.

· R4: I/O Bits 0-15
Contains the lower 16 bits of a 32-bit value to or from a peripheral interface. Whenever an I/O instruction is executed, both R4 and R5 will be read from or written to at the same time.

· R5: I/O Bits 16-31
Contains the upper 16 bits of a 32-bi value to or from a peripheral interface.

· R6: ALU Operand A
The value that is contained in this register will be used during an ALU instruction. If the instruction is a unary operation, this value alone will be used. If the ALU instruction is a binary operation, this value will be the left operand.

· R7: ALU Operand B
Contains the right operand of an ALU operation. If the ALU receives a unary operation, this value is ignored.

· R8 - R15: General Purpose Registers
These registers have no special purpose and may be used as a temporary place to store intermediate values.

2.3 Layout

ALU diagram goes here.

2.4 ALU

The ALU (arithmetic-logic unit) is capable of performing seven different base mathematical or logical operations:

1. **Pass value**:
Pass the value of operand A without performing any arithmetic or logic operations. Unary operation.

Other applications:

· Invert: Set ALU output invert flag.

2. **Logical OR**:
   Performs a bitwise OR operation using both operand A and operand B. Binary operation.

   Other applications:

   · NOR: Set ALU output invert flag.

   · NAND: Set operand A, and operand B invert flags.

   · AND: Set operand A, operand B, and ALU output invert flags.

3. **Addition**:
   Performs addition on both operands. Any overflow from adding or subtracting is ignored. Binary operation.

   · *Optional flag: Increment value after addition.*

   Other applications:

   · Increment: Set increment and zero operand B flags.

   · Decrement: Set increment, zero operand B, and negate operand B flags.

   · Subtract: Set increment, invert operand A, and invert operand B flags.

4. **Left Shift**:
   Left shifts the value of operand A by the value of operand B. Any bits shifted out of the value will be lost. Binary operation.

   · *Optional flag: perform a right shift instead of a left shift.*

   · *Optional flag: perform a bit rotation instead of a shift. Bits shifted out of the 16-bit value will be shifted back into the value on the opposite side.*

   · *Optional flag: Preserve the sign bit while shifting (arithmetic shift).*

5. **Exclusive OR (XOR)**:
   Performs an XOR operation using both operand A and operand B. Binary operation.

   Other applications:

   · XNOR: Set ALU output invert flag.

3 Instruction Set

The Syzygy B100 uses 16-bit instructions. There are two instruction formats – the first type uses only one bit as its opcode, and the second type uses four bits as its opcode.

The machine code syntax is displayed as 16 bits, grouped in nybbles of four bits. Any time a 0 or 1 appears for a bit, the machine instruction must be that 0 or 1. If a letter appears, the bit(s) can be changed (see descriptions for explanations of bit arguments). If an underscore (_) appears, the bit(s) are ignored.

1. **Push Instruction**
   Format: Single-bit
   Description: Sets R2 (accumulator) to the specified value, n.

   Syntax: `1nnn nnnn nnnn nnnn`

   - `n`: the 15-bit value that will be contained in R2 after this instruction is executed.

   Example: `0x8010` will set R2 to `0x0010`, or 16 in decimal.
   Notice: Negative values cannot be directly pushed.

2. **System Instruction**
   Format: Four-bit
   Description: Performs a special system instruction not available with the standard instructions.

   Syntax: `0000 pppp nnnn nnnn`

   - `p`: This 4-bit value chooses the type of system instruction that will be executed.
     `p` = **0x1:** Execute command
     `p` = **anything else:** Unused (no-op)
   - `n`: The arguments of the system instruction type.
     when p = **0x1:** Command to execute
     Commands:
     0x00: Von Neumann mode on (fetch instructions from system memory rather than boot ROM)
     0x01: Von Neumann mode off (fetch instructions from boot ROM rather than system memory)
     when p = **anything else:** Unused (no-op)

3. **Copy Instruction**
   Format: Four-bit
   Description: Copies the value of one register to another register. The source register's value remains unchanged, while the destination register's value is overwritten. If R0 is chosen as the source, a value of `0x0000` will be copied. Likewise, if R1 is chosen, `0xFFFF` will be copied.

   Syntax: `0001 ssss dddd ____`

- `s`: the register number that contains the value to be copied (source register) as a four-bit value.
- `d`: the register number to write the value being copied to (destination register) as a four-bit value.

Example: `0x1A60` will overwrite R6 with R10's value.

4. **Jump Instruction**
   Format: Four-bit
   Description: Sets R1 (Program Counter) to the value contained in R3 (Jump Address) if R2's (Accumulator) value passes the given comparison check to zero. Comparisons can be used together to form compound comparisons (for unconditional jumps, all three checks should be used).

   Syntax: `0010 leg_ ____ ____`

   - `l`: check if $R2 < 0$.
   - `e`: check if $R2 = 0$.
   - `g`: check if $R2 > 0$.

   Example: `0x2C00` will test if $R2 \leq 0$.

5. **ALU Instruction**
   Format: Four-bit
   Description: Uses the values of only R6 or both R6 and R7 as operands for an arithmetic or logic instruction. The result will be placed in R2. R2's value will be overwritten on any ALU instruction.

   Syntax: `0011 _ppp abzn rff_`

   - `p`: The operation the ALU will perform.
     0x0 : No operation; pass operand A
     0x1 : Bitwise OR operand A with operand B
     0x2 : Addition; A + B. Overflow is ignored.
     0x3 : Left shift operand A by operand B's value.
     0x4 : Exclusive-OR operand A with operand B.
     0x5 : Count the number of 1's in operand A.
     0x6 - 0x7 : Unused
   - `a`: Invert operand A (from R6, R6's value will not be changed) before calculating.
   - `b`: Invert operand B (from R7, R7's value will not be changed) before calculating.

- `z`: Use zero for operand B (from R7, R7's value will not be changed) before calculating.
- `n`: Invert the result's bits before writing to R2.
- `r`: Operation argument. Dependant on `p`:
  p = 2 : Increment the result of the addition.
  p = 3 : Shift right instead of left.
- `f`: Two bit argument that chooses the shift operation type to perform:
  0x0 : Logical shift
  0x1 : Logical rotation
  0x2 : Arithmetic shift
  0x3 : Arithmetic rotation

Example 1: `0x3208` will add $R6 + R7 + 1$.
Example 2: `0x331A` will shift R6's value right by R7, while preserving the sign bit, and then invert the output.
Example 3: `0x31D0` will perform a bitwise AND by taking advantage of DeMorgan's law (invert R6 and R7, OR, invert output).

6. **I/O Instruction**
   Format: Four-bit
   Description: Transfers data to or from a peripheral interface. Both R4 and R5 are used for a 32-bit value if the peripheral interface supports them. If not, R4 is used for the 16-bit value. R4 contains the 16 least-significant bits, and R5 contains the most-significant bits of a 32-bit value.

   Syntax: `0100 dddd rrrr xm__`

   - `d`: The four-bit ID of the peripheral interface to access:
     0x0 : LEDs
     0x1 : Seven-segment display
     0x2 : System memory
     0x3 : SD controller
     0x4 : Unused
     0x5 : VGA controller (monitor)
     0x6 : PS/2 controller (keyboard)
     0x7 - 0xF : Unused
   - `r`: The interface's register number to access. Ignored when `x` is set:
     R0 : Instruction register
     R1 : Status register
     R2 - R15 : Peripheral-specific

- `x`: Instruct the peripheral interface to execute the command in its instruction register.
- `m`: Data access mode. Ignored when `x` is set:
  - **0** : Read the 16 or 32-bit value from the interface's register, and overwrite R4 and R5 with its value.
  - **1** : Write the peripheral's chosen register with R4 and R5's value.

Example 1: `0x4344` will set peripheral interface 3's (SD controller by default) register number 4 (SD block number) to the value of R4 and R5.
Example 2: `0x4208` will instruct peripheral interface 2 (system memory by default) to execute the command in its instruction register.

7. **Opcodes `0x5` - `0x7` are unused.**

4 Memory

The Syzygy system has 128kB of memory available. Values are written 16 bits at a time, and data is manipulated through the memory interface (PID 2).

4.1 Memory Specifications

The system memory is designed for the Basys 3's block memory. The memory clock runs at 100MHz and is rising-edge triggered. The block memory is true dual-port, with one port being used by the CPU, and the other is read-only for the VGA output and runs from a different clock.

The CPU frequency runs slower than the memory clock because the FPGA's block RAM has a one cycle delay. Driving the memory clock faster allows a value to be fetched from memory on the rising edge of the CPU clock.

5 Peripheral Interfaces

In order to communicate with external systems, the Syzygy system follows a specific I/O protocol abstracted away by a peripheral's interface. Each interface must be compatible with the CPU's I/O instruction.

5.1 Interface Connection

A valid Syzygy interface supports the following connections (required connections are in **bold**, optional connections are in *italics.*):

- **Inputs:**

1. **CPU clock** - The CPU's clock signal.

2. **Peripheral select** - High if this peripheral is selected.
3. **Register select** - Four-bit value for the peripheral's register to read from or write to.
4. **Read enable** - If this signal is high , register reads will be enabled on the CPU clock's rising edge (data available in the interface's data-out connection).
5. **Write enable** - If this signal is high, the interface's selected register (by register select connection) will be written with data-in's value on the rising edge of the CPU clock.
6. *Reset* - Zeroes all registers in this interface on the next rising edge of the CPU clock signal.
7. *Data-In* - The 32-bit value from the CPU (using R4 and R5) that will be written to the selected register if the write enable signal is high.
8. *Execute* - Instructs the interface to execute the instruction stored in its instruction register. Instructions are interface-defined.
9. *Debug register select* - 4-bit value that selects the register whose value will be output to the debug-out connection.

· **Outputs:**

1. *Data-out* - The 32-bit value that will be sent back to the CPU. If the interface does not need a data-out, it should be kept as `0x00000000`.
2. *Debug value* - A connection that outputs a value that can be used in any way to assist in the debugging process.

An additional requirement is at least one register – the instruction register. If any other register is not needed, it shall output `0x0000`, or `0x00000000` if 32-bit values are supported, if the register is read from.

5.2 Interface Specifications

Below are descriptions of the various interfaces supported by the Syzygy system:

1. **LEDs**
   The instruction register's value will be displayed on the 16 LEDs on the FPGA. Status is always `0x0000`.
   Registers:
   · R0: Instruction register
   · R1: Status register
2. **Seven-segment display**
   The instruction register's value will be displayed on the four seven-segment display on the FPGA. Status is always `0x0000`.
   Registers:

- R0: Instruction register
- R1: Status register

3. **Memory**
   This interface handles memory writes and reads controlled by the CPU by I/O instructions. There are two instructions – read (`0x0000`), and write (`0x0001`) that can be executed, and if the memory is unavailable, the status register will be non-zero. On a data read or write, the interface will access the address given by the interface's R4 register. The data-in or data-out register will be read from or written to (respectively) after the interface's instruction completes.
   Registers:

   - R0: Instruction register
   - R1: Status register
   - R2: Data-in register
   - R3: Data-out register
   - R4: Address register

4. **SD Card**
   This interface abstracts away the complex process of using the SD controller. There are four instructions, and eight statuses (see below). This interface's status register starts in the BUSY_INIT status while waiting for the controller to complete the initialization process.

   Like the memory interface, this interface has a data-in and data-out register that is used in a similar fashion. However, the address register selects the 512-byte sector number on the SD card.
   Registers:

   - R0: Instruction register
   - R1: Status register
   - R2: Data-in register
   - R3: Data-out register
   - R4: Address register

5. **PS/2 (keyboard)**
   This interface contains one physical register that holds the currently-pressed key on the connected keyboard. R0 and R1 are ignored/unused.

   Key codes are converted from PS/2 scan codes to 8-bit values and are defined as followed:

   0x00 : null

   0x01 : ESC

0x02 - 0x0D : F1 - F12

0x0E : Return

0x0F : Backspace

0x10 : Caps Lock

0x11 : Num Lock

0x12 : Scroll Lock

0x13 - 0x1C : Numpad 0 - Numpad 9

0x1D : Numpad Decimal

0x1E : Tab

0x20 - 0x39 : A - Z

0x3A : Grave

0x3B : Hyphen

0x3C : Equals

0x3D : Left Curly Brace

0x3E : Right Curly Brace

0x3F : Backslash

0x40 - 0x48 : 0 - 9

0x49 : Semicolon

0x4A : Single Quote

0x4B : Comma

0x4C : Period

0x4D : Slash

0x4E : Space

Registers:

· R0: Instruction register

· R1: Status register

· R2: Syzygy key code