Project Overview

The ATHENA S Microprocessor is an 8-bit Harvard architecture microprocessor. It is an improved version that keeps all the features of the original design and includes a serial interface.

NOTE: HIGHLIGHTED ELEMENTS ARE CHANGES MADE SINCE INITIAL PROPOSAL

Feature Overview

- One external read only memory that stores instructions.
- One external data read/write memory that stores its working data.
- Addresses are sent to the instruction memory over a 16-bit bus, and instructions are received over an 8-bit (one byte) bus. (Instructions are single byte instructions, but there are also a few 2 and 3 byte instructions)
- Instructions are received sequentially over the 8-bit instruction bus.
- The data memory has an 8-bit address bus, and 8-bit busses for data input and data output from the CPU. The instruction set supports up to eight 8-bit registers.

Figure 1:
Dedicated asynchronous Serial Interface

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock (4)</td>
<td>clock signals phi0 and phi1 and their complements phi0_ and phi1_</td>
</tr>
<tr>
<td>power (2)</td>
<td>2 GND , 2 Vdd inputs</td>
</tr>
<tr>
<td>reset (1)</td>
<td>resets the processor to a known state (initializes all registers to 0 and resets the PLA)</td>
</tr>
<tr>
<td>in0-7</td>
<td>8 bit bus to take in input from either the data memory or the instruction memory</td>
</tr>
<tr>
<td>rxd</td>
<td>Serial output</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OUTPUTS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datawrite</td>
<td>selects whether to write to the data memory from out0-7 (1 to write)</td>
</tr>
<tr>
<td>addrselect</td>
<td>selects whether to write to the address register from out0-7 (1 to write)</td>
</tr>
<tr>
<td>out0-7</td>
<td>8 bit bus to output to the data memory or the external address register</td>
</tr>
<tr>
<td>memselect</td>
<td>selects whether to read from the data memory (1) or the instruction memory (0) to in0-7</td>
</tr>
<tr>
<td>addr0-15</td>
<td>16 bit bus to select the address of the instruction memory word to read</td>
</tr>
<tr>
<td>txd</td>
<td>Serial Input</td>
</tr>
</tbody>
</table>

Table 1:

Operation:
The two external instruction memory and the data memory are chip devices. There is a single 8-bit output bus and a single 8-bit input bus, and additional external hardware (not included) is needed to interface with the external memories.

When the addrselect line is asserted, the value on the output bus is written to the external Address register. When the datawrite line is asserted, the value on the output bus is written to the data memory (the result will be undone if the output bus changes while datawrite is asserted).

The input bus is driven from the instruction memory when memselect is 0, and is driven from the data memory when memselect is 1. Both memories implement an asynchronous read, which means that when the memory address bus is updated, the input bus will not wait for any particular clock phase before updating its value.
Major Sections:

1. **Main Programmable logic array**: This PLA houses the main finite state machine that handles and executes the instructions. The PLA takes 13 inputs, 8 of which are the instruction fed in through the instruction register, and 19 outputs. This was created with PEG tools.

2. **Serial Programmable logic array**: This PLA houses the finite state machine that specializes in sending and receiving 8 bit serial data. The PLA takes 3 inputs and 4 outputs. The serial PLA communicates with the MAIN PLA with a synchronous four signal handshake. This was created with PEG tools.

3. **3 to 8 demultiplexer**: Two of these demux are used to map the register binary number into the eight read and write signals needed.

4. **8 Registers**: These registers can all be written and read to as general registers.
   a. Register 0: Arithmetic Register (With built in 8-bit adder/subtractor)
   b. Register 1: Logic Register (With built in F-Block for AND/OR/XOR)
   c. Register 2: Shift Register
   d. Register 3 – 5: General only register
   e. Register 6: RXD Register
   f. Register 7: TXD Register

5. **8-bit fblock**: This is for AND/XOR/OR operations

6. **Instruction Register**: This half register is used to store instruction from the bus for the FSM.

7. **16 bit address register**: This register stores the instruction pointer for the external instruction memory. It is able to increment the pointer as needed and can be written to during JMP, BRC, BRZ operations.

A diagram of the major sections is shown in the following figure.
Serial handshaking protocol:
The Main PLA communicates with the Serial PLA in the following handshake

Serial RX
1. Main pla – Enter RXD state. Continually raise RXGO while polling for RXDONE to be raised.
2. Serial pla – Polls RXGO and leaves idle state after RXGO is raised. Begins inputting each bit one by one per clock cycle.
3. Serial pla – when done, raise RXDONE and poll RXGO, waiting for it to go low.
4. Main pla – lowers RXGO and goes to next state
5. Serial pla – lowers RXDONE and goes to idle state once again.

Serial TX
1. Main pla – Enter RXD state. Continually raise TXGO while polling for TXDONE to be raised.
2. Serial pla – Polls TXGO and leaves idle state after TXGO is raised. Begins outputting each bit one by one per clock cycle.
3. Serial pla – when done, raise TXDONE and poll TXGO, waiting for it to go low.
4. Main pla – lowers TXGO and goes to next state
5. Serial pla – lowers TXDONE and goes to idle state once again.

Figure 3: The rate of serial transmission and receive -> gross bit rate = \frac{\text{bit}}{\text{clock}} = \text{clock} \times \text{bit}
### Instruction Set Overview

Below is the table of the 17 built-in instructions of Athena S

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>First Byte</th>
<th># of bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV d, r</td>
<td>11rrddd</td>
<td>1</td>
<td>Overwrite register d with value in register r</td>
</tr>
<tr>
<td>ADD a, r</td>
<td>10rr000</td>
<td>1</td>
<td>Add register r to register a, overwriting register a</td>
</tr>
<tr>
<td>SUB a, r</td>
<td>10rr001</td>
<td>1</td>
<td>Subtract register r from register a, overwriting register a</td>
</tr>
<tr>
<td>OUT r</td>
<td>10rr010</td>
<td>1</td>
<td>Write the value in register r to the data memory</td>
</tr>
<tr>
<td>ADDR r</td>
<td>10rr011</td>
<td>1</td>
<td>Write register r to the external address register</td>
</tr>
<tr>
<td>AND l, r</td>
<td>10rr100</td>
<td>1</td>
<td>Bitwise AND of registers r and l, stored in register l</td>
</tr>
<tr>
<td>OR l, r</td>
<td>10rr101</td>
<td>1</td>
<td>Bitwise OR of registers r and l, stored in register l</td>
</tr>
<tr>
<td>XOR l, r</td>
<td>10rr110</td>
<td>1</td>
<td>Bitwise XOR of registers r and l, stored in register l</td>
</tr>
<tr>
<td>IN d</td>
<td>01001ddd</td>
<td>1</td>
<td>Read from data memory into register d</td>
</tr>
<tr>
<td>CIN d</td>
<td>01010ddd</td>
<td>2</td>
<td>Load 2nd instruction byte into register d</td>
</tr>
<tr>
<td><strong>JMP</strong></td>
<td>00110101</td>
<td>3</td>
<td>Set instruction address to the trailing instruction bytes. This is done by storing half of the address (one byte) into Register 5 by default</td>
</tr>
<tr>
<td><strong>BRZ</strong></td>
<td>00101101</td>
<td>3</td>
<td>If register l is zero, set instruction address as above. This is done by storing half of the address (one byte) into Register 5 by default</td>
</tr>
<tr>
<td><strong>BRC</strong></td>
<td>00100101</td>
<td>3</td>
<td>If the last ADD/SUB had the carry bit out of the top bit set, set the instruction address as in JMP above. This is done by storing half of the address (one byte) into Register 5 by default</td>
</tr>
<tr>
<td>LSR</td>
<td>00000010</td>
<td>1</td>
<td>Shift value in register s to the right</td>
</tr>
<tr>
<td>ASR</td>
<td>00000011</td>
<td>1</td>
<td>Shift value in register s to the right with sign extension</td>
</tr>
<tr>
<td>TXD</td>
<td>00001111</td>
<td>1</td>
<td>Begins transmitting serial data</td>
</tr>
<tr>
<td>RXD</td>
<td>00001010</td>
<td>1</td>
<td>Begins receiving serial data</td>
</tr>
</tbody>
</table>

**Table 2:** Notice that the **green highlighted** bits are ones that are also hardware mapped by the demux for read/write signals.
How to Test Athena

IRSIM TESTING

1. In the Annenberg Instruction lab, find the directory “Athena” under “cd kchen2” or “cd awwu”.
2. Open terminal inside the Athena folder, where all the magic, CAST, and sim files are, and type
   “irm sim athena”
3. Now enter the “Athena Testing” subfolder. There you will find five IRSIM test cases that covers all the
   instruction sets. The IRSIM commands will simulate the data values from Data memory AND Instruction
   Memory. Each IRSIM COMMAND provides the Athena with the simulated environment for the following
   conditions.
   a. Data memory will change input only when memselect is RAISED
   b. Values of input 0-7 that correspond to the instruction register changes only when the pointer is
      changed.
4. One can also edit the test cases, but be careful in following the simulation rules from the prior statement
   so that the instructions are properly taken.

CAST Testing

1. In the Annenberg Instruction lab, find the directory “Athena” under “cd kchen2” or “cd awwu”.
2. Open terminal inside the Athena folder, where all the magic, CAST, and sim files are, and type
   “% cflat -lvs athena_lvs.cast | alvs -Ddvv athena_lvs.sim”
3. This will layout-versus-schematic check the complete datapath with the demux. The PLAs will not be
   includes since alvs does not work with it.
Finite State Machine Design

-- CS 181a Fall 2012
-- by Anjian Wu/ Kevin Chen
-- A complex FSM - Athena Control

INPUTS

: RESET
b0 b1 b2 b3 b4 b5 b6 b7
carry reg1flag
TXDONE RXDONE;

OUTPUTS

: reg_regw reg_regwr
reg0_regw reg1_regw
reg5_regw
regInstr_regr
MUXSIGNAL
UPDATE regA1_regw regA2_regw
regIO_regw regIO_regw
addrsselect
datawrite
shift extend
memselect
TXGO RXGO;

Main

: CASE (b7 b6 b5 b4 b3 b2 b1 b0)

1 0 ? ? ? 0 0 0 => ADDSUB;
1 0 ? ? ? 0 0 1 => ADDSUB;
1 0 ? ? ? 0 1 0 => OUT;
1 0 ? ? ? 0 1 1 => ADDR;
1 0 ? ? ? 1 0 0 => LOGIC;
1 0 ? ? ? 1 0 1 => LOGIC;
1 0 ? ? ? 1 1 0 => LOGIC;
1 0 ? ? ? 1 1 1 => LOGIC;
0 1 0 0 1 ? ? ? => IN;
0 1 0 1 0 ? ? ? => CIN;

0 0 1 1 0 1 0 1 => JMP;
0 0 1 0 1 1 0 1 => BRZ;
0 0 1 0 0 1 0 1 => BRC;

0 0 0 0 0 0 1 0 => LSR;
0 0 0 0 0 0 1 1 => ASR;
0 0 0 0 1 1 1 1 => TXD;
0 0 0 0 1 0 1 0 => RXD;

ENDCASE => NEXTINSTRUCTION;

TXD

: ASSERT TXGO;
IF TXDONE THEN NEXTINSTRUCTION ELSE LOOP;

RXD

: ASSERT RXGO;
IF RXDONE THEN NEXTINSTRUCTION ELSE LOOP;
NEX TinSTRUCTIO N: ASSERT UPDATE regA1_regw regA2_regw; -- Next instruction state
: ASSERT reg10_regr;
: ASSERT regInstr_regr;
: GOTO Main;

MOV: ASSERT reg_regr reg_regw; -- Read out rrr value -- Write into ddd reg
: GOTO NEX TinSTRUCTIO N;

ADD SUB: ASSERT reg_regr;
: ASSERT reg0_regw MUXSIGNAL; -- Read out rrr value -- read ADD/SUB value
: GOTO NEX TinSTRUCTIO N;

OUT: ASSERT reg_regr; -- write out rrr value
: ASSERT reg10_regw datawrite; -- write to Data Memory
: GOTO NEX TinSTRUCTIO N;

ADDR: ASSERT reg_regr; -- write out rrr value
: ASSERT reg10_regw addrselect; -- write to addr reg
: GOTO NEX TinSTRUCTIO N;

LOGIC: ASSERT reg_regr; -- write out rrr value
: ASSERT reg1_regw MUXSIGNAL; -- AND FBLOCK OPERATION
: GOTO NEX TinSTRUCTIO N;

IN: ASSERT regIO_regr memselect reg_regw; -- read in data memory-- write out ddd value
: GOTO NEX TinSTRUCTIO N;

CIN: ASSERT UPDATE regA1_regw regA2_regw; -- Next instruction state
: ASSERT reg10_regr reg_regw; -- read in external -- write out ddd value
: GOTO NEX TinSTRUCTIO N;

JMP: ASSERT UPDATE regA1_regw regA2_regw; -- Next instruction state
: ASSERT reg10_regr reg_regw; -- get next instruction, read in external-- temp save value in reg5
: ASSERT UPDATE regA1_regw regA2_regw; -- Next instruction state
: ASSERT reg10_regr regA1_regw; -- store lower addr
: ASSERT reg5_regr regA2_regw; -- read back stored addr-- store above to upper addr
: ASSERT reg10_regr;
: ASSERT regInstr_regr;
: GOTO Main;

BRZ: IF reg1flag THEN JMP ELSE NEX TinSTRUCTIO N;

BRC: IF carry THEN JMP ELSE NEX TinSTRUCTIO N;
LSR
: ASSERT shift; -- shift value
: GOTO NEXTINSTRUCTION;

ASR
: ASSERT shift extend; -- shift value with extend
: GOTO NEXTINSTRUCTION;

---
--- CS 181a Fall 2012
--- by Anjian Wu/ Kevin Chen
--- A complex FSM - Athena Serial Control

INPUTS
: RESET
: TXGO RXGO;

OUTPUTS
: shift_tx shift_rx
: TXDONE RXDONE;

Main
: CASE (TXGO RXGO)
  1 0 => TXD;
  0 1 => RXD;
  ENDCASE => LOOP;

TXD
: ASSERT shift_tx;
: ASSERT shift_tx;
: ASSERT shift_tx;
: ASSERT shift_tx;
: ASSERT shift_tx;
: ASSERT shift_tx;
: ASSERT shift_tx;
: ASSERT shift_tx;
: GOTO TXDFINISHED;

TXDFINISHED
: ASSERT TXDONE;
: IF TXGO THEN LOOP ELSE Main; -- Handshaking Protocol

RXD
: ASSERT shift_rx;
: ASSERT shift_rx;
: ASSERT shift_rx;
: ASSERT shift_rx;
: ASSERT shift_rx;
: ASSERT shift_rx;
: ASSERT shift_rx;
: ASSERT shift_rx;

RXDFINISHED
: ASSERT RXDONE;
: IF RXGO THEN LOOP ELSE Main;