Serpent X8-64K

Flight Manual

John Wallis

2021

Contents

[Chapter 1: Getting Started 3](#_Toc87903366)

[Chapter 2: Hardware 3](#_Toc87903367)

[Chapter 3: Programming 3](#_Toc87903368)

# Chapter 1: Getting Started

**1.1 Introduction**

The X8-64K is a simple 8-bit computer with the ability to address up to 64kB of memory, it has two general purpose registers, conditional branching and a dedicated hardware stack controller. The schematics and hardware have all been released as open source as well as the relevant firmware for the device.

The system has been designed to use mostly 74 Series components of equivalent and components of similar complexity. Exceptions to this rule include a 64k x 8 ram chip that serves as the program and general-purpose memory, as well as an Altera MAX CPLD which is used to emulate the vintage 74181 ALU chip which is no longer manufactured. Additionally, the CPLD implements a state machine and ROM which are used in order to decode instructions from RAM.

This system is aimed at being as simple as possible with every component being easily understandable by someone with a basic grasp of digital logic and some electronics.

**1.2 Power**

In the top left corner of the board there is a series of pin headers, a 2x5 row used for the JTAG debug port. This port is used in order to program the CPU outlined in Chapter 3: Programming. Additionally, there is a 1x5 row of pin headers, the top two pins are labelled GND and +5V respectively, GND and 5V to the respective pins is all that is necessary for power up. Initially there will only blank instructions in memory. Because of this the CPU will be sit idle until programmed with external hardware using the JTAG link.

**1.3 Quick Start Programming**

It is highly recommended to familiarize yourself with the hardware of the CPU in Chapter 2, before programming. However, the CPU can be programmed very easily be connecting an Arduino to the JTAG interface. Specifically, the TCK, TMS, TDI and GND pins which can be determined from the Pinout given in Figure 1.

A simple program is given:

STC 0

JBHI

STC &LOOP

JBI

STC 1

AIN

BIN

&LOOP SUM

AIN

DSP

JMP

This program will continuously count by 1 and display it on the output port, with a low enough clock speed this binary counter will be visible. To compile and upload this program with the JTAG link navigate to the Toolchain directory in the repository, save this program to a .sasm file in the directory then from the command-line run:

Toolchain> python3 SASM\_X8-64k.py file.sasm

Toolchain> python3 Legacy-Upload.py out.bin

# Chapter 2: Hardware

**2.1 CPU block Diagram/Timing**

**2.2 Register Descriptions**

The X864K has two general purpose registers A and B which can be used for storing values and ALU operations, additionally it has the following special registers; two registers which address the random-access memory, two registers that point to a branch location and a special register connected to a parallel port. Most importantly there is the main registers which is used as intermediate register when transferring data around the CPU. Table 1 contains a description of these registers and any relevant SASM instructions.

Table 1: Register Description

|  |  |  |
| --- | --- | --- |
| Register Name | Purpose | Relevant Instructions |
| regA | regA is a general-purpose register, it can be used along with regB in a variety of arithmetic and logic operations including Addition, Subtraction, Bitwise AND, OR, XOR, NOT, as well as comparing if regA is equal to regB.  Additionally, regA can be directly compared with zero and shifted right. | AIN: Copies the value in MainReg into regA  AOT: Copies the value in regA into MainReg  AMM: will set the equals flag if A is equal to zero |
| regB | regB is a general-purpose register, it can be used along with regA in a variety of arithmetic and logic operations including Addition, Subtraction, Bitwise AND, OR, XOR, NOT, as well as comparing if regA is equal to regB. | BIN: Copies the value in MainReg into regB  BOT: Copies the value in regB into MainReg |
| MainReg | MainReg is a register that is used to transfer data between registers and memory, all instructions that result in data transfer will either load to or from MainReg. Additionally, this register can be used to load numerical values specified by the program without the need to store these values at sperate locations in memory. | STC X: Loads the value of X into MainReg |
| JumpBufferLow | This register forms the lower byte of a 16-bit register. When a branch command is successful the program counter will jump to the memory location specified by this 16-bit register. | JBI: Copies the value of MainReg into JumpBufferLow |
| JumpBufferHigh | This register forms the higher byte of a 16-bit register. When a branch command is successful the program counter will jump to the memory location specified by this 16-bit register. | JBHI: Copies the value of MainReg into JumpBufferHigh |
| MemAddressLow | This register forms the lower byte of a 16-bit register. Along with MemAddressHigh they span the entire 64kB of memory and specify the addressed use in load from and store to, memory commands | SMA: Copies the value of MainReg into MemAddressLow |
| MemAddressHigh | This register forms the lower byte of a 16-bit register. Along with MemAddressLow they span the entire 64kB of memory and specify the addressed use in load from and store to, memory commands | SMAH: Copies the value of MainReg into MemAddressHigh |
| OutputRegister | The OutputRegister is connected to the 8 red LEDS and Output Pin Header, it is important to note that the Output of this register is Little Endian which means that the leftmost bit is the least significant bit, (this is backwards to how the rest of the system is and how the decimal number system is). | DSP: Copies the value of MainReg into OutputRegister |

**2.3 Hardware Counters**

There are two hardware counters.

The first and most important is the program counter, The program counter is used to address the location in memory which an instruction is loaded from, this counter is incremented every 6 cycles normally. It can also have its value changed using jump and branch commands based on the values of JumpBufferLow and JumpBufferHigh.

The second hardware counter is the stack point. This counter is used to address the location in memory used for the stack. This counter cannot be loaded to or modified directly; however, it will increment before placing a byte onto the stack, and decrement after pulling a byte of the stack. By default, it initialises to memory address 0x0000.

**2.4 8-bit 74181 ALU Description**

The CPU uses a 74181 based ALU design which is emulated on board the CPLD. The ALU consists of a series of logic blocks that depending on which 4-bit input is sent on the instruction lines any of the possible 16 bitwise Boolean functions acting on its two inputs can be calculated. Additionally, if the 5th instruction line is active then the internal carries are enabled (known as arithmetic mode), and the outputs of this binary function will be added together by selecting the correct function, addition and two’s complement subtraction can be enabled. A list of Instructions and the corresponding machine code can be found in Table 2. An example of a full 8-bit machine code instruction is for the A\* output, first both highest bits (7 and 6) must be set to 1’s then the Carrier must be set as bit 5 of the word, in the case of logic mode it is a don’t care X, then Logic mode must be specified by setting bit 4 to 1, then the corresponding select data is set with bit 3 down to 0. Example:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ALU\_En1 | ALU\_En0 | Carry | Mode | S3 | S2 | S1 | S0 |
| 1 | 1 | X | 1 | 0 | 0 | 0 | 0 |

Table 2: List of ALU instructions

|  |  |  |  |
| --- | --- | --- | --- |
| Select Data | Logic Mode | Arithmetic Mode 0 Carry | Arithmetic Mode 1 Carry |
| 0000 | A\* | A | A plus 1 |
| 0001 | (A OR B)\* | A OR B | (A + B) plus 1 |
| 0010 | A\* AND B | A OR B\* | (A OR B\*) plus 1 |
| 0011 | Logic 0 (all 0) | -1 (two’s complement) or 255 (unsigned) | 0 |
| 0100 | (A AND B)\* | A plus (A AND B\*) | A plus (A and B\*) plus 1 |
| 0101 | B\* | (A OR B) |  |
| 0110 | A XOR B | A minus B minus 1 (compare) | A minus B |
| 0111 | A AND B\* | A AND B\* minus 1 | A AND B\* |
| 1000 | A\* OR B | A plus (A AND B) | A plus (A AND B) plus 1 |
| 1001 | (A XOR B)\* | A plus B | A plus B plus 1 |
| 1010 | B | (A OR B\*) plus (A and B) | (A OR B\*) plus (A and B) plus 1 |
| 1011 | A AND B | (A AND B) minus 1 | A AND B |
| 1100 | Logical 1 (All 1s) | A plus A (Right Shift) | A plus A plus 1 |
| 1101 | A OR B\* | (A OR B) plus A | (A OR B) plus A plus 1 |
| 1110 | A OR B | (A OR B\*) plus A | (A OR B\*) plus A plus 1 |
| 1111 | A | A minus 1 | A |

For many of these commands they are very rarely used, because of this they do not have corresponding SASM instructions, In order to use them determine the corresponding machine code and directly execute this instruction through the use of an EXE directive.

**2.5 JTAG Port**

JTAG stands for Joint Test Action Group and is an industry stand tool for debugging integrated circuits and interconnects on circuit boards. For the purpose of this system the JTAG port can be used to take direct control of the CPU and remotely execute specific instructions, as well as modify memory values. The Pinout for the JTAG port can be seen in Figure 1.

JTAG Pinout


Figure 1: JTAG Pinout

**2.6 JTAG Debug Protocol and Data Transmission**

# Chapter 3: Programming

**3.1 Intro to the SASM assembly language**

**3.2 List of SASM Instructions**

**3.3 Defining Constants in SASM**

**3.4 Using Constants to Define Labels and Variables**

**3.5 Comparisons, Flags and Branching**

**3.6 Using the Stack**

**3.7 Calling Subroutines**

**3.8 Time Delays**

**3.9 Example Programs**