Version 1.0

MACH64 Programmble Logic Starter Kit

Hands-On™ Guide

Andre' LaMothe

Nurve Networks LLC

MACH64 Programmable Logic Starter Kit Hands-On™ Guide Version 1.0

Author
Andre' LaMothe

Editor/Technical Reviewers
Patrick Tanner
Glenn Jones
Maksim Djackov
Daniel Quakenbush
Jodell Bumatay

A very special thanks to all the editors. Each had a different style and hopefully caught most typographical and technical errors ☺. I am still mystified how every single "it's" got turned into "its", but I think it has something to do with <Replace All> and <Find Next> ☺

Printing
0001

ISBN
Pending

Trademarks
All terms mentioned in this user manual that are known to be trademarks or service marks have been appropriately capitalized. Nurve Networks LLC cannot attest to the accuracy of this information. Use of a term in this user manual should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer
Every effort has been made to make this user manual as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an ***"as is"*** basis. The authors and the publisher shall have neither liability nor any responsibility to any person or entity with respect to any loss or damages arising from the information contained in this user manual.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Licensing, Terms & Conditions

NURVE NETWORKS LLC, . END-USER LICENSE AGREEMENT FOR HYDRA HARDWARE, SOFTWARE , EBOOKS, AND USER MANUALS

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS PRODUCT. IT CONTAINS SOFTWARE, THE USE OF WHICH IS LICENSED BY NURVE NETWORKS LLC, INC., TO ITS CUSTOMERS FOR THEIR USE ONLY AS SET FORTH BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE OR HARDWARE. USING ANY PART OF THE SOFTWARE OR HARDWARE INDICATES THAT YOU ACCEPT THESE TERMS.

GRANT OF LICENSE: NURVE NETWORKS LLC (the "Licensor") grants to you this personal, limited, non-exclusive, non-transferable, non-assignable license solely to use in a single copy of the Licensed Works on a single computer for use by a single concurrent user only, and solely provided that you adhere to all of the terms and conditions of this Agreement. The foregoing is an express limited use license and not an assignment, sale, or other transfer of the Licensed Works or any Intellectual Property Rights of Licensor.

ASSENT: By opening the files and or packaging containing this software and or hardware, you agree that this Agreement is a legally binding and valid contract, agree to abide by the intellectual property laws and all of the terms and conditions of this Agreement, and further agree to take all necessary steps to ensure that the terms and conditions of this Agreement are not violated by any person or entity under your control or in your service.

OWNERSHIP OF SOFTWARE AND HARDWARE: The Licensor and/or its affiliates or subsidiaries own certain rights that may exist from time to time in this or any other jurisdiction, whether foreign or domestic, under patent law, copyright law, publicity rights law, moral rights law, trade secret law, trademark law, unfair competition law or other similar protections, regardless of whether or not such rights or protections are registered or perfected (the "Intellectual Property Rights"), in the computer software and hardware, together with any related documentation (including design, systems and user) and other materials for use in connection with such computer software and hardware in this package (collectively, the "Licensed Works"). ALL INTELLECTUAL PROPERTY RIGHTS IN AND TO THE LICENSED WORKS ARE AND SHALL REMAIN IN LICENSOR.

RESTRICTIONS:
(a) You are expressly prohibited from copying, modifying, merging, selling, leasing, redistributing, assigning, or transferring in any matter, Licensed Works or any portion thereof.
(b) You may make a single copy of software materials within the package or otherwise related to Licensed Works only as required for backup purposes.
(c) You are also expressly prohibited from reverse engineering, decompiling, translating, disassembling, deciphering, decrypting, or otherwise attempting to discover the source code of the Licensed Works as the Licensed Works contain proprietary material of Licensor. You may not otherwise modify, alter, adapt, port, or merge the Licensed Works.
(d) You may not remove, alter, deface, overprint or otherwise obscure Licensor patent, trademark, service mark or copyright notices.
(e) You agree that the Licensed Works will not be shipped, transferred or exported into any other country, or used in any manner prohibited by any government agency or any export laws, restrictions or regulations.
(f) You may not publish or distribute in any form of electronic or printed communication the materials within or otherwise related to Licensed Works, including but not limited to the object code, documentation, help files, examples, and benchmarks.

TERM: This Agreement is effective until terminated. You may terminate this Agreement at any time by uninstalling the Licensed Works and destroying all copies of the Licensed Works both HARDWARE and SOFTWARE. Upon any termination, you agree to uninstall the Licensed Works and return or destroy all copies of the Licensed Works, any accompanying documentation, and all other associated materials.

WARRANTIES AND DISCLAIMER: EXCEPT AS EXPRESSLY PROVIDED OTHERWISE IN A WRITTEN AGREEMENT BETWEEN LICENSOR AND YOU, THE LICENSED WORKS ARE NOW PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR THE WARRANTY OF NON-INFRINGEMENT. WITHOUT LIMITING THE FOREGOING, LICENSOR MAKES NO WARRANTY THAT (i) THE LICENSED WORKS WILL MEET YOUR REQUIREMENTS, (ii) THE USE OF THE LICENSED WORKS WILL BE UNINTERRUPTED, TIMELY, SECURE, OR ERROR-FREE, (iii) THE RESULTS THAT MAY BE OBTAINED FROM THE USE OF THE LICENSED WORKS WILL BE ACCURATE OR RELIABLE, (iv) THE QUALITY OF THE LICENSED WORKS WILL MEET YOUR EXPECTATIONS, (v) ANY ERRORS IN THE LICENSED WORKS WILL BE CORRECTED, AND/OR (vi) YOU MAY USE, PRACTICE, EXECUTE, OR ACCESS THE LICENSED WORKS WITHOUT VIOLATING THE INTELLECTUAL PROPERTY RIGHTS OF OTHERS. SOME STATES OR JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES OR LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY MAY LAST, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. IF CALIFORNIA LAW IS NOT HELD TO APPLY TO THIS AGREEMENT FOR ANY REASON, THEN IN JURISDICTIONS WHERE WARRANTIES, GUARANTEES, REPRESENTATIONS, AND/OR CONDITIONS OF ANY TYPE MAY NOT BE DISCLAIMED, ANY SUCH WARRANTY, GUARANTEE, REPRESENATION AND/OR WARRANTY IS: (1) HEREBY LIMITED TO THE PERIOD OF EITHER (A) Five (5) DAYS FROM THE DATE OF OPENING THE PACKAGE CONTAINING THE LICENSED WORKS OR (B) THE SHORTEST PERIOD ALLOWED BY LAW IN THE APPLICABLE JURISDICTION IF A FIVE (5) DAY LIMITATION WOULD BE UNENFORCEABLE; AND (2) LICENSOR'S SOLE LIABILITY FOR ANY BREACH OF ANY SUCH WARRANTY, GUARANTEE, REPRESENTATION, AND/OR CONDITION SHALL BE TO PROVIDE YOU WITH A NEW COPY OF THE LICENSED WORKS. IN NO EVENT SHALL LICENSOR OR ITS SUPPLIERS BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER, INCLUDING, WITHOUT LIMITATION, THOSE RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT LICENSOR HAD BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OF THE LICENSED WORKS. SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

SEVERABILITY: In the event any provision of this License Agreement is found to be invalid, illegal or unenforceable, the validity, legality and enforceability of any of the remaining provisions shall not in any way be affected or impaired and a valid, legal and enforceable provision of similar intent and economic impact shall be substituted therefore.

ENTIRE AGREEMENT: This License Agreement sets forth the entire understanding and agreement between you and NURVE NETWORKS LLC, supersedes all prior agreements, whether written or oral, with respect to the Software, and may be amended only in a writing signed by both parties.

NURVE NETWORKS LLC
12724 Rush Creek Lane
Austin, TX 78732

Version & Support/Web Site

This document is valid with the following hardware, software and firmware versions:

- MACH64 Programmable Logic Starter Kit
- Lattice ispLever Classic Version 7.0.

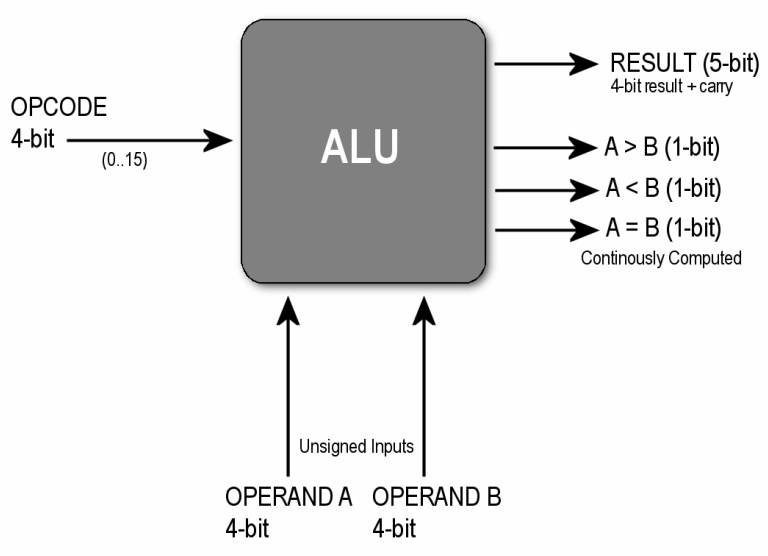
The information herein will usually apply to newer versions but may not apply to older versions. Please contact Nurve Networks LLC for any questions you may have.

Visit **www.xgamestation.com** for downloads, support and access to the XGameStation/HYDRA user community and more!

For technical support, sales, general questions, share feedback, please contact Nurve Networks LLC at:

support@nurve.net / **nurve_help@yahoo.com**

Figure 105.0 – Block diagram of the simple ALU.



7.10 Lab 10 – Simple ALU (Arithmetic Logic Unit)

Difficulty – Hard

In this lab we are going to really push the ispMACH 4064 to its limits and design a little ALU (arithmetic logic unit). The ALU is the basis of all modern microprocessors and with a proper set of registers and data paths is the core of the microprocessor. So, if we can build an ALU with the CPLD then we are 90% the way to building a little microprocessor with it! However, the ispMACH 4064 only has 64 flip flops in it, so it would be a challenge to make a working microprocessor with it. However, never say never. I believe that a 2-4 bit microprocessor with a couple instructions and a 6-8 word program memory would be possible. Of course your programs would look like:

```
0: LDI #0 // load the 4-bit accumulator with the immediate value 0
1: STA 3  // store the value 0 in memory location 3 (overwrites the NOP)
2: JMP #0 // jump back to memory location 0, that is Program Counter = 0
3: NOP    // dead code
4:
5:
.
.
15: NOP   // last instruction goes here with a 4-bit program counter
```

But, that's better than nothing!

Anyway, we aren't going that far in this lab, just the ALU part. Hence, the idea here is to design a 4-bit ALU that can perform logical and simple math operations to two 4-bit input operands, display the results, and support up to 16 opcodes. Figure 105.0 above shows a block diagram of the planned ALU. You will use the ALU by inputting operands A and B on the DIP switches, then selecting an opcode via the push buttons, then viewing the results on the LEDs. Also, this ALU is going to be totally asynchronous; meaning that it simply computes the results constantly. In a real system, you would want to latch the results out on a clock after the longest computational path time, so that you always have stable results at a deterministic point. But, for our needs asynchronous design works great and makes it easier, so we don't have to deal with a clock.

The operations supported by the ALU are shown in Table 25.0 below. The inputs are ***A*** and ***B*** (both 4-bit) and the result is X (5-bit).

Table 25.0 – ALU operations supported (taken directly from source).

Symbolic Name	Value	Description
OP_NOP	0	no operation
OP_INCA	1	X = A++
OP_DECA	2	X = A--
OP_NOR	3	X = A NOR B
OP_OR	4	X = A OR B
OP_XOR	5	X = A XOR B
OP_XNOR	6	X = A XNOR B
OP_NAND	7	X = A NAND B
OP_AND	8	X = A AND B
OP_NOTA	9	X = NOT A
OP_NOTB	10	X = NOT B
OP_SHR	11	X = A >> 1
OP_SHL	12	X = A << 1
Note: *opcodes 13, 14, 15 not allocated yet, feel free to add them.*		

NOTE	As you can see the ALU is missing addition and subtraction of ***A*** and ***B***. This is due to space in the silicon. When I had add/subtract of the 4-bit operands, almost nothing else would fit. Thus, in the end, I decided better to have a large repertoire of logical operations rather than only a couple math operations.

Finally, since the ALU is asynchronous, all operations are not stored. In other words, when you increment ***A*** for example with the opcode ***OP_INCA***, ***A*** is not incremented, rather a copy of ***A*** is made then +1 added to that and assigned to ***X***. But, the moment you remove the opcode, the results are removed as well. Thus, to store this result you would need a clock and final assignment back to ***A***.

7.10.1 Setup and Parts Needed

For this lab you need:

- 28 wires, (17) short, (11) long.
- A lot of patience.

Make sure you have the MACH64 powered off while connecting everything and you should have the following settings:

- VCCIO SEL switch set at 3.3V (might as well conserve power).
- J15 jumpers all on (to make sure the on-board CPLD is programmed).

7.10.2 Assembly

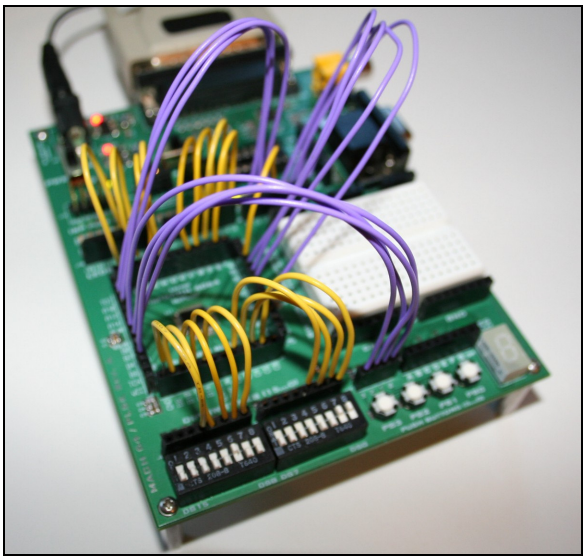
Step 1: Netlist connections – In this step you will connect up all the wires for the project with the power ***OFF***. In this lab there is only a netlist of ***from*** and ***to*** points, there are no other components and the solderless breadboard area isn't needed. Also, as usual the design could use any IO pins you like, but I try to use IO pins that are closest to the headers they are going to plug into. Additionally, now and then I like to throw in an example where an IO on the far side of the chip is mixed with another IO, just to show that the fitter will route whatever you tell it to.

NOTE	Make sure power is OFF while you are building the circuit! And pay close attention to long and short wire suggestions. And as always triple check your connections especially with this circuit, since it nearly uses up all the I/O pads and it's really easy to short one of the power supply lines if you plug something into the wrong header port, so be careful.

Table 26.0 – Netlist for the "Simple ALU" lab.

Net #	Instructions
Input A connections (4) from DIP switches	
1	Connect a short wire from **pin 14 (B8)** on the CPLD header to DIP switch **DS11** (leftmost bank).
2	Connect a short wire from **pin 15 (B10)** on the CPLD header to DIP switch **DS10** (leftmost bank).
3	Connect a short wire from **pin 16 (B12)** on the CPLD header to DIP switch **DS9** (leftmost bank).
4	Connect a short wire from **pin 17 (B14)** on the CPLD header to DIP switch **DS8** (leftmost bank).
Input B connections (4) from DIP switches	
5	Connect a short wire from **pin 20 (C0)** on the CPLD header to DIP switch **DS3** (rightmost bank).
6	Connect a short wire from **pin 21 (C2)** on the CPLD header to DIP switch **DS2** (rightmost bank).
7	Connect a short wire from **pin 22 (C4)** on the CPLD header to DIP switch **DS1** (rightmost bank).
8	Connect a short wire from **pin 13 (C6)** on the CPLD header to DIP switch **DS0** (rightmost bank).
Output A connections (4) to LEDs	
9	Connect a short wire from **pin 48 (A8)** on the CPLD header to **LED L19** (leftmost bank).
10	Connect a short wire from **pin 46 (A4)** on the CPLD header to **LED L18** (leftmost bank).
11	Connect a short wire from **pin 45 (A2)** on the CPLD header to **LED L17** (leftmost bank).
12	Connect a short wire from **pin 44 (A0_GOE0)** on the CPLD header to **LED L16** (leftmost bank).
Output B connections (4) to LEDs	
13	Connect a short wire from **pin 41 (D14_GOE1)** on the CPLD header to **LED L11** (middle bank).
14	Connect a short wire from **pin 40 (D12)** on the CPLD header to **LED L10** (middle bank).
15	Connect a short wire from **pin 39 (D10)** on the CPLD header to **LED L9** (middle bank).
16	Connect a short wire from **pin 38 (D8)** on the CPLD header to **LED L8** (middle bank).
Output X connections (5) to LEDs	
17	Connect a short wire from **pin 34 (D6)** on the CPLD header to **LED L4** (rightmost bank).
18	Connect a long wire from **pin 33 (D4)** on the CPLD header to **LED L3** (rightmost bank).
19	Connect a long wire from **pin 32 (D2)** on the CPLD header to **LED L2** (rightmost bank).
20	Connect a long wire from **pin 31 (D0)** on the CPLD header to **LED L1** (rightmost bank).
21	Connect a long wire from **pin 28 (C14)** on the CPLD header to **LED L0** (rightmost bank).
Output LT, GT, EQ connections (3) to LEDs	
22	Connect a long wire from **pin 2 (A10)** on the CPLD header to **LED L7** (rightmost bank).
23	Connect a long wire from **pin 3 (A12)** on the CPLD header to **LED L6** (rightmost bank).
24	Connect a long wire from **pin 4 (A14)** on the CPLD header to **LED L5** (rightmost bank).
Input Opcode connections (4) to pushbuttons	
25	Connect a short wire from **pin 7 (B0)** on the CPLD header to push button **PB3** at **J17 pin 3**.
26	Connect a short wire from **pin 8 (B2)** on the CPLD header to push button **PB2** at **J17 pin 2**.
27	Connect a short wire from **pin 9 (B4)** on the CPLD header to push button **PB1** at **J17 pin 1**.
28	Connect a short wire from **pin 10 (B6)** on the CPLD header to push button **PB0** at **J17 pin 0**.

Figure 106.0 – The completed Simple ALU.



Your project should look like the image shown in Figure 106.0 more or less. You might have different color wires with your kit and a different PCB color, but the "net list" and setup should be identical.

7.10.3 Compiling the Project

The code for the project is named **mach64_kit_alu_01.abl** and is located here:

\MACH64\sources\ mach64_kit_alu_01.abl

Step 1 – Loading the code into tool

Go ahead and import it into your project by selecting the ***left pane*** of the ispLEVER Classic navigator and clicking on the device node ***LC4064V-75T48I*** and then right clicking to get the context menu, then select ***"Import"*** and locate the file on the CD or your local hard drive. Make sure to make it the active or top most file if you have more in your project.

Step 2 – Compiling the code

Select the filename in the ***"Sources"*** pane on the left, this should change the context to the processes for a source file. Then double click ***"Compile Logic"*** in the ***"Processes for current source"*** pane and if all goes well you should see a green arrow. The code is listed below for reference:

```
MODULE mach64_kit_alu_01

////////////////////////////////////////////////////////////////////////////////////////////////
// Author: Andre' LaMothe
//
// Last Modified:
//
// Description: Simple ALU (arithmetic logic unit), non-clocked asynchronous
//
// Inputs: two 4-bit operands input on DIP switches, one 4-bit operator input on pushbuttons.
//
// Outputs: copies of the 4-bit inputs, so they are visible, connected to LEDs, along with the
//          5-bit results of the ALU operation (4-bit plus carry) as well as a 3-bit relational
//          output that indicates if the operators are less than, equal, greater than.
//
////////////////////////////////////////////////////////////////////////////////////////////////

DECLARATIONS

       // ALU operation constants, set the most common operations to 1,2,4,8,
       // so they are easy to press the buttons
       // operation   output

       OP_NOP  = 0;   // no operation
```

```
        OP_INCA = 1;   // X = A++
        OP_DECA = 2;   // X = A--
        OP_NOR  = 3;   // X = A NOR B
        OP_OR   = 4;   // X = A OR B
        OP_XOR  = 5;   // X = A XOR B
        OP_XNOR = 6;   // X = A XNOR B
        OP_NAND = 7;   // X = A NAND B
        OP_AND  = 8;   // X = A AND B
        OP_NOTA = 9;   // X = NOT A
        OP_NOTB = 10;  // X = NOT B
        OP_SHR  = 11;  // X = A >> 1
        OP_SHL  = 12;  // X = A << 1

        // inputs and outputs for operands and result

        !in_a3..!in_a0 pin 14..17;    // inputs for register A, not registered since this system is non-clocked
        !in_b3..!in_b0 pin 20..23;    // inputs for register B, not registered since this system is non-clocked

        out_a3..out_a0 pin 48, 46..44 istype 'com';  // output for register A, not registered
        out_b3..out_b0 pin 41..38 istype 'com';      // output for register B, not registered

        x4..x0 pin 34..31, 28 istype 'com';  // output for result (5-bit to hold carry for addition)

        // for magnitude comparisons
        gt pin 2 istype 'com';               // output for A > B
        lt pin 3 istype 'com';               // output for A < B
        eq pin 4 istype 'com';               // output for A == B

        // ALU operand selection push button inputs
        !pb3..!pb0 pin 7..10;         // used for operation select buttons

        // set declarations
        opcode = [pb3..pb0];          // set for operation selected
        in_a   = [in_a3..in_a0];      // set for input A
        in_b   = [in_b3..in_b0];      // set for input B

        out_a  = [out_a3..out_a0];    // set for output A
        out_b  = [out_b3..out_b0];    // set for output B

        x      = [x4..x0];            // holds output result of ALU operation

EQUATIONS

        // the ALU is totally combinatorial and not clocked,
        // so we simply generate all the outputs based on the inputs
        // of course some operations take longer than others to compute,
        // the "fitter report" can give insight into this showing
        // the slowest paths in the system and the maximum operational rate
        // NOTE: there isn't enough silicon to support addition with all these other operands,
        // so if you want to add that then you
        // have to remove quite a few others!

        // first the magnitude comparisons
        gt = (in_a > in_b);
        lt = (in_a < in_b);
        eq = (in_a == in_b);

        // send inputs to outputs unscathed
        out_a = in_a;
        out_b = in_b;

        // now ALU operation, lots of ways to do this with clever syntax, but let's use straightforward syntax
        when (opcode == OP_NOP) then // = 0, no operation
                {
                // do nothing
                // set x = 0
                x = 0;
                } // end when
        when (opcode == OP_INCA) then // = 1
                {
                // set x = A + 1
                x = [0, in_a3, in_a2, in_a1, in_a0] + [0,0,0,0,1];
                } // end when
        else
        when (opcode == OP_DECA) then // = 2
                {
                // set x = A - 1
                x = [0, in_a3, in_a2, in_a1, in_a0] - [0,0,0,0,1];
                } // end when
        else
         when (opcode == OP_NOR) then // = 3,  X = A NOR B
             {
             // set x = in_a NOR in_b
             x = !([1, in_a3..in_a0] # [1, in_b3..in_b0]);
             } // end when
        else
         when (opcode == OP_OR) then  // = 4,  X = A OR B
```

```
            {
            // set x = in_a OR in_b
            x = ([0, in_a3..in_a0] # [0, in_b3..in_b0]);
            } // end when
        else
        when (opcode == OP_XOR) then  // = 5,  X = A XOR B
            {
            // set x = in_a XOR in_b
            x = ([0, in_a3..in_a0] $ [0, in_b3..in_b0]);
            } // end when
        else
        when (opcode == OP_XNOR) then // = 6,  X = A XNOR B
            {
            // set x = in_a XNOR in_b
            x = !([1, in_a3..in_a0] $ [0, in_b3..in_b0]);
            } // end when
        else
        when (opcode == OP_NAND) then // = 7,  X = A NAND B
            {
            // set x = in_a NAND in_b
            x = !([1, in_a3..in_a0] & [1, in_b3..in_b0]);
            } // end when
        else
        when (opcode == OP_AND) then  // = 8,  X = A AND B
            {
            // set x = in_a AND in_b
            x = ([0, in_a3..in_a0] & [0, in_b3..in_b0]);
            } // end when

        else
        when (opcode == OP_NOTA) then // = 9,  X = NOT A
            {
            // set x = !in_a
            x = !([1, in_a3..in_a0]);
            } // end when
        else
        when (opcode == OP_NOTB) then // = 10, X = NOT B
            {
            // set x = !in_a
            x = !([1, in_b3..in_b0]);
            } // end when
        else
        when (opcode == OP_SHR) then  // = 11, X = A >> 1
            {
            // set x = A >> 1
            x = [0, 0, in_a3..in_a1];
            } // end when
        else
        when (opcode == OP_SHL) then  // = 12, X = A << 1
            {
            // set x = A << 1
            x = [in_a3..in_a0, 0];
            } // end when

END
```

Code Analysis

There is way too much code to cover all of it, so we are only going to discuss the general operation and a single example. The idea of the ALU is that two 4-bit ***"operands"*** are fed into it along with a 4-bit ***"opcode"***. The operands are input on DIP switches, the opcode on the pushbuttons which when un-pressed input code 0000 (due to inversion and pullups) which is the ***OP_NOP***. This is good, since we don't want the ALU to do anything when you aren't pressing any of the pushbuttons.

Now, since the ALU is totally non-synchronous, it's continuously evaluating the opcode and then running thru the large ***WHEN-THEN*** case statement where there is a clause for each possible opcode. Whichever opcode is on the bus will be combinatorially computed immediately. But, before the WHEN-THEN statement there are a couple code fragments of interest. One of them simply copies the inputs to the outputs for display on the LEDs:

```
        // send inputs to outputs unscathed
        out_a = in_a;
        out_b = in_b;
```

The second does a magnitude comparison which is constantly output to LEDs as well:

```
        // first the magnitude comparisons
        gt = (in_a > in_b);
        lt = (in_a < in_b);
        eq = (in_a == in_b);
```

And remember, all this happens at the same time. There is no first or second (the comment is just for organization). Alright, now let's get back to the big case statement. Let's say that the opcode ***OP_OR*** is on the inputs then this piece of code will start to execute:

```
when (opcode == OP_OR) then  // = 4,  X = A OR B
    {
    // set x = in_a OR in_b
    x = ([0, in_a3..in_a0] # [0, in_b3..in_b0]);
    } // end when
```

All it does is OR the two operands together and push them out to X for display. All the ALU operations work this way. Lastly, say you wanted to add an opcode? Let's just make something up. How about reverse the bits of operand A? First, we need to define the opcode, let's call it OP_REVA, we need to add it at the top of the source with the other opcodes like this:

```
.
.
OP_NOTB = 10;  // X = NOT B
OP_SHR  = 11;  // X = A >> 1
OP_SHL  = 12;  // X = A << 1
OP_REVA = 13;  // X = [a0, a1, a2, a3], bitwise reversal
```

So far so good. Next, we need to add the case statement to the **WHEN-THEN** to do the work. Let's place it at the end right after the last opcode:

```
.
.
else
when (opcode == OP_SHL) then  // = 12, X = A << 1
    {
    // set x = A << 1
    x = [in_a3..in_a0, 0];
    } // end when
else
when (opcode == OP_REVA) then // = 13, X = [a0, a1, a2, a3]
    {
    // set x = A << 1
    x = [0,in_a0..in_a3];
    } // end when
```

Almost too easy isn't it! But, watch out if you try to implement any math related operations they will use up nearly all of the silicon. Keep an eye on this by reviewing the ***"Fitter Report"***.

Step 3 – Fitting the Design

After the code has successfully compiled, it's time to fit the design into the target device. So, select the device node in the ***"Sources"*** pane to the left. This should bring up the much more complex context menu in the ***"Processes for current source"*** pane that have the ***"Fit Design"*** option. Go ahead and double click ***"Fit Design"***. You might get a dialog for the constraints that pops up, select ***"Import"*** if it does. This simply means to import the constraints (like ***pin*** declarations) from your file and not to read them from an external constraint file. This is what we will do in most cases, but in some cases we might create a constraint file to fine tune the constraints. After the fitting process, you should get a green check mark, or at least yellow check mark next to the ***"Fit Design"*** process.

Step 4 – Programming the CPLD with the JEDEC File

The final step in the process is to download the JEDEC bitstream file into the CPLD itself. Finally, turn the power ***ON***, so the PC can see the onboard programmer and initiate communications.

INFO	The reason why we keep the power off until now is that the previous designs could have inputs going to outputs and outputs going to inputs which cause a lot of current. So each new netlist will potentially cause electrical shorts until the new program is downloaded, thus we want to keep the power OFF until we are ready to download the correct program for the netlist to minimize wear and tear on the driver IO circuits.

Go ahead and run the **ispVM System** from the toolbar or from the ***Tools*** menu on the main menubar. You should already have a configuration file that represents the ispMACH 4064 chip that refers to the output JEDEC file in your project which should be called something like **mach_64_test_01.jed** or whatever you named it. Remember, the configuration file for the ispVM System has both the device (the ispMACH 4064 in this case) and the JEDEC file in the configuration file. This way, no matter what you compile in the compiler and fitter, they always output the same JEDEC filename which you can always program into the chip with the ispVM System tool. Anyway, download the bits into the chip by pressing the green ***GO*** button on the toolbar of the ispVM System tool. It should pause for a few seconds and you should have success and see ***PASS*** next to the device line in the window.

If there is a problem, make sure of the following:

1. The parallel port cable is plugged in.
2. The power is on.
3. The JEDEC file referred to in the configuration file does indeed exist and the device (the chip) is correct.
4. You have installed the drivers for the parallel port.

You should have done all these things previously in the setup and test run, so they should all be familiar to you.

7.10.4 Hands-On

Experiment 1: Input two values on **A** and **B** DIP switches, then try every single opcode with the pushbuttons. It's a bit hard pressing all four pushbuttons, but you will manage. Confirm that all the opcodes work and the results on output **X** and the **LT**, **GT**, **EQ** outputs are always correct.

7.10.5 Summary

This lab shows how to use the CPLD to create a fairly full featured ALU (minus the math operations). The code is amazingly straightforward as well. However, this lab also shows that math operation on the CPLD eat silicon up very quickly, so if you want to perform something as simple as 8-bit addition, be prepared to watch your silicon eaten alive.

7.10.6 Exercises

1. See if you can output the result X on the 7-segment display. You just have enough free IOs to do this, but you won't be able to drive the LEDs with X anymore, you will only be able to drive the 7-segment.

2. Add opcodes for bit reversal of **A** and **B**, add an opcode that counts the number of **ON** bits in **A**.

3. Add a clock signal with the 1.0MHz clock that latches **X** always, then instead of the **OP_NOP** assigning 00000 to **X** make it maintain state. This way, you can see the results of your last operation on X.

4. Try and create an additional operation, see how many of the other opcodes you have to comment out to fit addition.

5. If we have a 8-bit opcode and we wanted to support immediate values of 3-bits in the opcode itself, how many opcodes are possible?

Answers

5. Given that the desired opcode format is CCCCCDDD where Ds stand for the immediate value bits and Cs stand for the opcode bits, then there are 5-bits for opcodes, or 2^5 = 32 possible opcodes. This is how many ARM processors encode constants as part of the 16/32 bit instruction itself, that's why they are limited to 8-12 bits usually.