

HADES Project #2

Part I due 11 a.m., Friday, March 3

Part II due 12 noon, March 10

**Part I: 8-bit ALU with Carry-Lookahead Adder (CLA).** For this project, you need to modify/update your 8-bit ALU with ripple-carry adder and convert it into an 8-bit ALU with carry-lookahead adder. Instead of relying on Figure B.6.3 of our textbook (reproduced in Figure 1), you may refer to the Hades carry-lookahead adder (8-bit) demonstration applet<sup>1</sup> which is organized as shown in Figure 3. You may want to organize your CLA unit similarly into several CLA blocks.

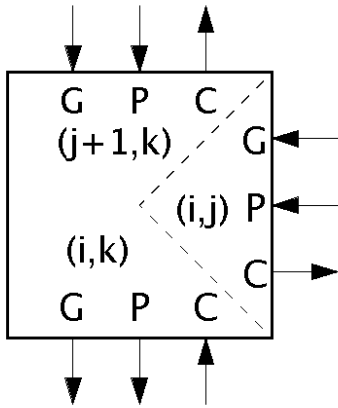


Figure 2: Detail of Hades CLA block ports.

The Hades 8-bit CLA adder (see Figure 3) consists of 8 1-bit adder blocks and a tree of (4+2+1) CLA blocks. To facilitate connections between CLA blocks, you may wish to refer to Figure 2. The advantage of the CLA scheme used in this circuit is its simplicity, because each CLA block calculates the generate and propagate signals for two bits only. This is much easier to understand than the more complex variants present-

ed in other textbooks, where combinatorial logic is used to calculate the *G* and *P* signals of four or more bits, and the resulting adder structure is slightly faster but also less regular.

So, you need to decide how you want to design your CLA circuit – either as a single CLA block as in Figure 1 or as multiple CLA blocks as in Figure 3. Store your CLA block design under the filename `cla.hds` and have Hades generate a symbol for your subdesign. Next, integrate your CLA block(s) with your 1-bit ALU designs from Hades Lab #1 to build an 8-bit ALU with CLA<sup>2</sup>. Save your 8-bit ALU design under the filename `alu8cla.hds` and have Hades generate a symbol for your subdesign. Edit the Hades-generated symbol for your 8-bit ALU so that it looks more like the traditional “vee”-like block used to represent ALUs. Use *Hex Switch* and *Hex Display* units to provide input operands to your ALU and *Hex Display* units to display the output of your ALU. Be sure to position your *Hex Switches* and *Hex Displays* using the most logical arrangement possible. Also use *Label* units to provide/convey as much information about your design as possible. Use the filename `_alu8cla.hds` to save your test circuit under.

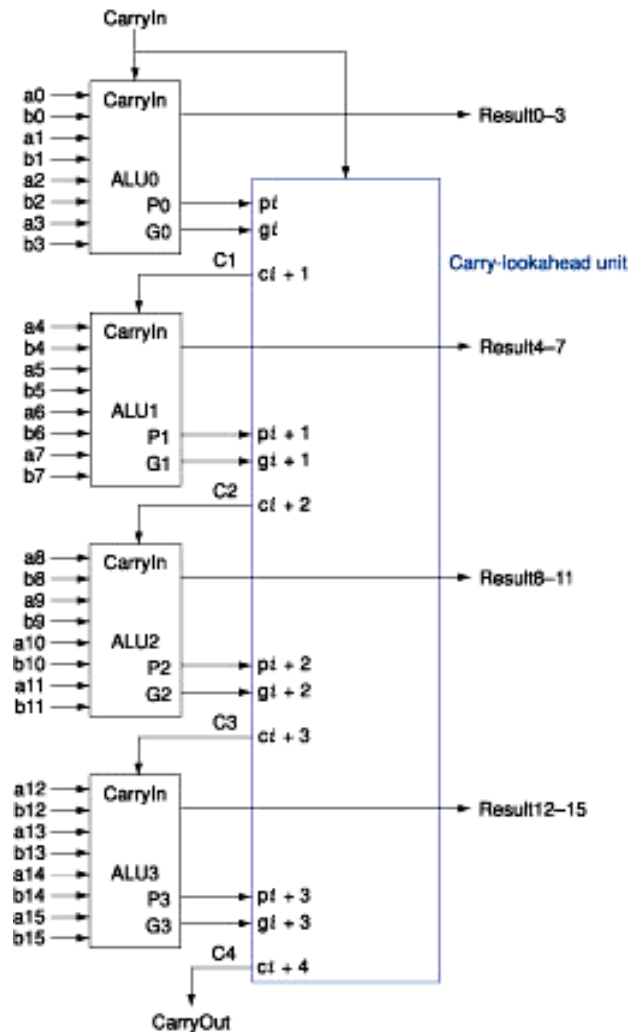


Figure 1: 32-bit carry-lookahead adder (CLA), from Fig. B.6.3 of our textbook.

1 <http://tech-www.informatik.uni-hamburg.de/applets/hades/webdemos/20-arithmetic/30-cla/adder8.html>  
 2 Alternately, you can build a CLA block for a 4-bit ALU and just connect two 4-bit ALUs (one with that special 1-bit ALU with overflow detection) with CLA hardware together to create your 8-bit ALU.

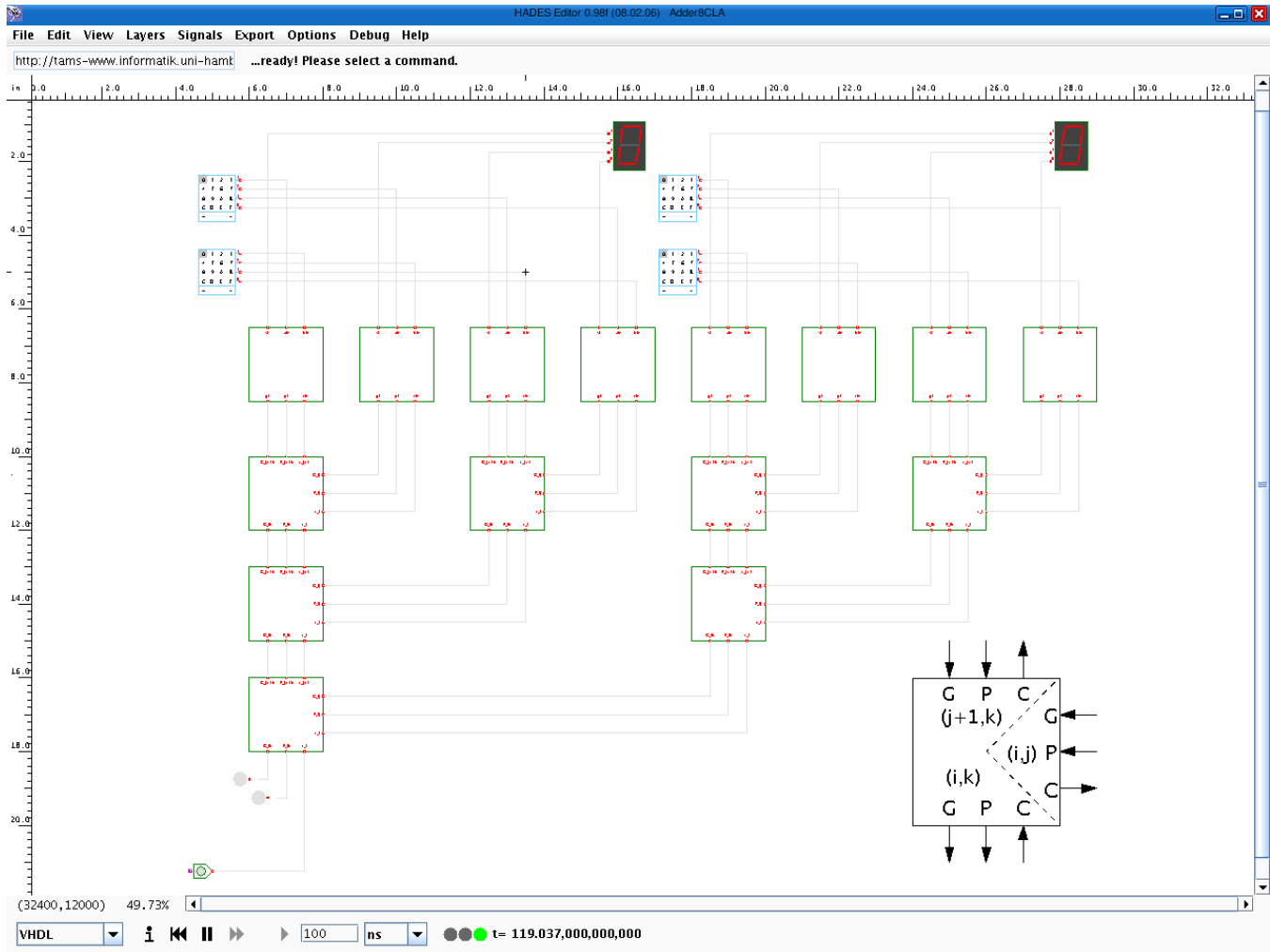


Figure 3: Hades carry-lookahead adder (8 bit) applet.

**Part II: 8-bit ALU with Multiplier Hardware.** The next modification to your design will be to give your ALU the necessary circuitry to perform 8-bit multiplication. You may refer to Section 3.4 on pages 176 through 180 of our textbook, or *Booth's algorithm*<sup>1</sup> as presented in IMD3 pages 5 through 9 in the supplementary CD that accompanies our textbook.

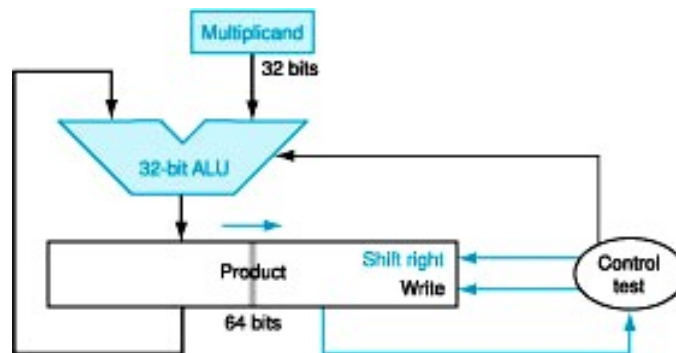


Figure 4: Multiplication hardware from Figure 3.7 of our textbook.

1 Wikipedia, Booth's multiplication algorithm, [http://en.wikipedia.org/wiki/Booth's\\_multiplication\\_algorithm](http://en.wikipedia.org/wiki/Booth's_multiplication_algorithm)

Use the *Colibri* Hades component and library browser, accessible through the Hades *pop-up*, *create*, and *browse ...* options. In either implementation, you are going to need an 8-bit *Multiplicand* register and a 16-bit *Product* register (plus one bit for Booth's algorithm) – see Figure 4 where the register widths are four times the size you need. Generic  $n$ -bit edge-triggered D-type shift registers are under built-in – *hades* – models – register – ShiftRegister in *Colibri*. To synchronize the timing of your elements, be sure to use a *Clock Generator* component. Be sure to use your `alu8cla.hds` in your solution.

Similar to what you did for Part I, save your new 8-bit ALU design (with multiplication hardware) under the filename `alu-with-mult.hds` and have Hades generate a symbol for your subdesign. Edit the Hades-generated symbol for your 8-bit ALU so that it looks more like the traditional “vee”-like block used to represent ALUs. (You may use the `.sym` file from Part I as a reference for your `.sym` file for your ALU with multiplier.) Use *Hex Switch* and *Hex Display* units to provide input operands to your ALU and *Hex Display* units to display the output of your ALU. Be sure to position your *Hex Switches* and *Hex Displays* using the most logical arrangement possible. Also use *Label* units to provide/convey as much information about your design as possible. Use the filename `_aluwith-mult.hds` to save your test circuit under.

Clearly label all your designs, specially all your control lines, so that a user can easily figure out how to test your circuits.

*Be sure to thoroughly test your circuit subdesigns before proceeding to higher level designs.*

### What to submit:

1. Printouts of all your circuit designs for `cla.hds`, `alu8cla.hds` and `alu-with-mult.hds`. Make sure all your circuit diagrams have *label* elements that contain your name, course name and number, and the current semester on it – hand-written names on the printouts will not be accepted. Be sure to clearly label important parts of your design, specially those that pertain to input and output values. Use *label* elements extensively to give instructions or to point out certain features of your design.
2. E-mail all your `.hds` (you should have five – three design files and two test circuit files with file-names that begin with “\_”) and `.sym` (you should have three) files to `Juliano@csuChico.edu` for testing. Provide your name and the phrase “Hades Lab#2” on the subject line of your e-mail message. If you archive the files you are submitting, be sure to change the file extension from `.zip` or `.rar` or `.tar` to `.foo` or `.txt` so that the mailer does not block it.