September 29, 2003

- Lab Access
  - No complaints received …
- Up-Down Counter Design
- Handel-C Basics *continued*

Up-Down Counter Design

- Simpler version: Count button presses
  - PalReadSwitch(handle, pointer)
    - Debouncing not an issue
    - Simulated pushbuttons *vs.* switches
  - Infinite recursion warnings?
  - Need to count in binary and convert to BCD.
    - Or count in BCD
  - Sample code: up_down_counter.hcc

Up-Down Timer Design

- Timing Resolution Issues
- Suggest State Machine Design
- Sample Code Walkthrough

Timer Resolution Issues

- To what precision will one second intervals be kept?
  - We already have a msec_delay() macro.
  - Stopwatches are accurate to 0.01 sec.
- How are button presses synchronized with the one second interval counter?
- What happens to an interval in progress when the timer is stopped and restarted?
  - Reset to zero
  - Continue last interval

Suggest State Machine Design

- Need to keep track of previous and current states of buttons to detect edges.
- The C *enum* mechanism is a good for state variables because the state names can be used as constants.
  - typedef lets you use enums as data types.
- Switch statements can be translated into hardware decoders if there is a case for every possible value of the test variable, such as when the test variable is an enum.
  - Especially when there are 2^n cases.

Language Basics *continued*

- Bit Operators
- Macros

Bit Operators

- Conventional C Bitwise Operators
  - AND (&)
  - OR (|)
  - XOR (^)
  - Shift (<< and >>)
- Handel-C Additions
  - Take (←)
  - Drop (\\ )
  - Concatenate (@)
  - Select ([m:n])

Macros

- Conventional C Macros
  - Preprocessor Directives
    - #define PRECISION 5
    - #define add(x,y) (x) + (y)
    - Note spacing, lack of semicolons, and how to handle multi-line definitions
- Handel-C Macros
  - Macro Expressions
  - Macro Procedures

Macro Expressions

- macro expr precision = 5;
- macro expr add(x,y) = (x)+(y);
- Can be recursive
  - macro expr copy(x, n) =
    select(n==1, x, (x @ copy(x, n-1)));
  - macro expr extend(y, m) =
    copy(y[width(y)-1],m-width(y)) @ y;

The *adjust()* Macro Expression

- Generate a value whose width matches the size of another a variable
  - Part of stdlib.hch
  - Language Reference Manual pg. 125
- macro expr adjust(x, n) =
  select(width(x) < n, (0@x),(x←n));
- a, b, c with widths of 4, 5, 6
  - b = a; b = c;
- Contrast to conditional operator (? :)

Macro Procedures

- macro proc fun(arg1, arg2) { … }
  - Equivalent to an inline function
  - No compile-time type checking of arguments.

    “Macro procs have become largely obsolete by inline functions but not by ordinary (reused) functions. Functions provide for more portable code and are the preferred method of writing in Handel-C. Macro procs are included in the language primarily for compatibility with earlier versions of the language. Macro procs may provide an area saving for small blocks of code and do allow for more extensive parameterization of the code. For example, macro procs are very useful for functions that will work on different width arguments.

    “Macro procs share datapath if possible, but duplicate control path for each call. Function calls share control path and datapath. In some cases the overhead of a function call may be larger than the duplicated control logic; for example when the subroutine has a small amount of control logic and is only used a few times.” (Handel-C FAQ, page 4)