Loop Indexes

A LOOPINDEX is an object that counts from a start value to an end value. Its name comes from the fact that it resembles the I index of a DO loop. However, LOOPINDEXes can be used anywhere, not just in DO loops. In particular, they can be used in IsoMax state machines to perform a counting function.

Defining a Loop Index

You define a LOOPINDEX just like you define a variable:

```
LOOPINDEX name
```

...where you choose the "name." For example,

```
LOOPINDEX CYCLE-COUNTER
```

Once you have defined a LOOPINDEX, you can specify a starting value, an ending value, and an optional step (increment) for the counter. For example, to specify that the counter is to go from 0 to 100 in steps of 2, you would type:

```
0 CYCLE-COUNTER START
100 CYCLE-COUNTER END
2 CYCLE-COUNTER STEP
```

You can specify these in any order. If you don't explicitly specify `START`, `END`, or `STEP`, the default values will be used. The default for a new counter is to count from 0 to 1 with a step of 1. So, if you want to define a counter that goes from 0 to 200 with a step of 1, all you have to change is the `END` value:

```
LOOPINDEX BLINK-COUNTER
200 BLINK-COUNTER END
```

If you use a negative `STEP`, the counter will count backwards. In this case the `END` value must be less than the `START` value!

You can change the `START`, `END`, and `STEP` values at any time, even when the counter is running.

Counting

The loopindex is incremented when you use the statement

```
name COUNT
```

For example,

```
CYCLE-COUNTER COUNT
```

`COUNT` will always return a truth value which indicates if the loopindex has *passed* its limit. If it has not, `COUNT` will return false (zero). If it has, `COUNT` will return true (nonzero), and it will also reset the loopindex value to the `START` value.

This truth value allows you to take some action when the limit is reached. This can be used in an IF..THEN statement:

```
CYCLE-COUNTER COUNT IF  GRNLED OFF  THEN
```

It can also be used as an IsoMax condition:

```
CONDITION CYCLE-COUNTER COUNT CAUSES GRNLED OFF ...
```

In this latter example, the loopindex will be incremented every time this condition is tested, but the `CAUSES` clause will be performed only when the loopindex reaches its limit.

Note that the limit test depends on whether STEP is positive or negative. If positive, the loopindex "passes" its limit when the count value + STEP value is *greater than* the END value. If negative, the loopindex passes its limit when the count value + STEP value is *less than* the END value.

In both cases, signed integer comparisons are used. **Be careful** that your loopindex limits don't result in an infinite loop! If you specify an END value of HEX 7FFF, and a STEP of 1, the loopindex will *never* exceed its limit, because in two's complement arithmetic, adding 1 to 7FFF gives -8000 hex -- a negative number, which is clearly *less* than 7FFF.

Also, be careful that you always use or discard the truth value left by COUNT. If you just want to increment the loopindex, without checking if it has passed its limit, you should use the phrase

```
CYCLE-COUNTER COUNT DROP
```

Using the Loopindex Value

Sometimes you need to know the value of the index while it is counting. This can be obtained with the statement

```
name VALUE
```

For example,

```
CYCLE-COUNTER VALUE
```

Sometimes you need to manually reset the count to its starting value, before it reaches the end of count. The statement

```
name RESET
```

will reset the index to its START value. For example,

```
CYCLE-COUNTER RESET
```

Remember that you *don't* need to explicitly RESET the loopindex when it reaches the end of count. This is done for you automatically. The loopindex "wraps around" to the START value, when the END value is passed.

A "DO loop"Example

This illustrates how a loopindex can be used to replace a DO loop in a program. This also illustrates the use of VALUE to get the current value of the loopindex.

```
LOOPINDEX BLINK-COUNTER
DECIMAL 20 BLINK-COUNTER END
2 BLINK-COUNTER STEP
: TEST   BEGIN  BLINK-COUNTER VALUE .  BLINK-COUNTER COUNT UNTIL ;
```

If you now type TEST, you will see the even numbers from 0 (the default START value) to 20 (the END value).[1] This is useful to show how the loopindex behaves with negative steps:

```
-2 BLINK-COUNTER STEP
40 BLINK-COUNTER START
BLINK-COUNTER RESET
TEST
```

This counts backwards by twos from 40 to 20. Note that, because we changed the START value of BLINK-COUNTER, we had to manually RESET it. Otherwise TEST would have started with the index

[1] Forth programmers should note that the LOOPINDEX continues *up to and including* the END value, whereas a comparable DO loop continues only *up to* (but not including) its limit value.

value left by the previous TEST (zero), and it would have immediately terminated the loop (because it's less than the END value of 20).

An IsoMax Example

This example shows how a loopindex can be used within an IsoMax state machine, and also illustrates one technique to "slow down" the state transitions. Here we wish to blink the green LED at a rate 1/100 of the normal state processing speed. (Recall that IsoMax normally operates at 100 Hz; if we were to blink the LED at this rate, it would not be visible!)

```
LOOPINDEX CYCLE-COUNTER
DECIMAL 100 CYCLE-COUNTER END
1 CYCLE-COUNTER START

MACHINE SLOW_GRN

ON-MACHINE SLOW_GRN
 APPEND-STATE SG_ON
 APPEND-STATE SG_OFF

IN-STATE SG_ON
 CONDITION CYCLE-COUNTER COUNT
 CAUSES GRNLED OFF
 THEN-STATE SG_OFF
 TO-HAPPEN

IN-STATE SG_OFF
 CONDITION CYCLE-COUNTER COUNT
 CAUSES GRNLED ON
 THEN-STATE SG_ON
 TO-HAPPEN

SG_ON SET-STATE
INSTALL SLOW_GRN
```

Here the loopindex CYCLE-COUNTER counts from 1 to 100 in steps of 1. It counts in *either* state, and only when the count reaches its limit do we change to the other state (and change the LED). That is, the end-of-count CAUSES the LED action and the change of state. Since the counter is automatically reset after the end-of-count, we don't need to explicitly reset it in the IsoMax code.

Summary of Loopindex Operations

`LOOPINDEX name` Defines a "loop index" variable with the given `name`. For example,

```
LOOPINDEX COUNTER1
```

`START`
`END`
`STEP`

These words set the start value, the end value, or the step value (increment) for the given loop index. All of these expect an integer argument and the name of a loopindex variable. Examples:

```
  1 COUNTER1 START
100 COUNTER1 END
  3 COUNTER1 STEP
```

These can be specified in any order. If any of them is not specified, the default values will be used (START=0, END=1, STEP=1).

`COUNT` This causes the given loop index to increment by the STEP value, and returns a true or false value: true (-1) if the end of count was reached, false (0) otherwise. For example:

```
COUNTER1 COUNT
```

End of count is determined after the loop index is incremented, as follows: If STEP is positive, "end of count" is when the index is greater than the END value. If STEP is negative, "end of count" is when the index is less than the END value. Signed integer comparisons are used. In either case, when the end of count is reached, the loop index is reset to its START value.

`RESET` This word manually resets the given loop index to its START value. Example:

```
COUNTER1 RESET
```

`VALUE` This returns the current index value (counter value) of the given loop index. It will return a signed integer in the range -32768..+32767. For example:

```
COUNTER1 VALUE .        ...prints the loop index COUNTER1
```