Fast Interpolation of a 16-entry table by scaling a value to 12-bit

A 12-bit value can be divided into a 4-bit index and an 8-bit residual value.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Field | Unused | | | | Index | | | | Residual | | | | | | | |

In a typical 8-bit processor the 12-bit value is stored in a double accumulator D which is a concatenation of two 8-bit registers, typically A and B.

|  |  |  |
| --- | --- | --- |
| Double Accumulator D | | |
| Accumulator A | | Accumulator B |
| Unused | Index | Residual |

A voltage of 7.25 can be represented in fixed-point binary notation as 111,0100 (7, 0.25) and scaled to 12-bit as 111, 0100, 000

If take an example of a 16-entry table that looks up a time in ms for a measured value of voltage, for clarity the example calculations are done in decimal but the method is the same for binary, the input voltage value is scaled such that the index is 7 and the residual is 0.25 (in binary 0111, 0100,0000)

Table at 0xC000

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n(Volts) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| ms | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 | 140 | 150 | 160 |

A voltage of 7.25V will equate to a time value of 82.5ms.

**Interpolation routine**

The result is derived as: n[index] + ((n[index+1] – n[index]) \* residual)  
 Result = n[7] + (n[8] – n[7])\* 0.25)

Result = 80 + ((90-80)\*0.25)  
Result = 80 + (10\*0.25)  
Result = 82.5

X is loaded with the table base address = 0xC000  
D is loaded with the voltage = 7.25 (A=7, B=0.25)  
The result is returned in A

Table\_LU: ldx #$C000  
 ldab voltage ; B = index byte

abx ; X is now pointing to n[index]

ldaa voltage+1 ; get the residual value

beq done ; if residual is 0 the answer is cell n

Lin\_interp: ldab 1,x ; get value from n[index+1]

subb 0,x ; n[index+1] – n[index]

mul ; multiply difference by residual

done: adda 0,x ; A = result + n[index]

rts

The previous algorithm assumes that the values in the table increase from left to right. The code is expanded to deal with the alternate case below:

Table\_LU: ldx #$C000  
 ldab voltage ; B = index byte

abx ; X is now pointing to n[index]

ldaa voltage+1 ; get the residual value

beq loc\_D621 ; if remainder is 0 the answer is n[index]

lin\_interp: ldab 1,x ; fetch n[index+1]

subb 0,x ; Subtract

bcs loc\_D61E ; bra if n[index+1] < n[index]

positive: mul ; if n[index] > n[index+1]

bra done ;

negative: negb ; Negate (1s complement)

mul ; Unsigned multiply

nega ; Negate (1’s complement)

done: adda 0,x ; Add result to n[index]

rts ; Return from subroutine

In the above routine n[index+1] – n[index] is a negative number. This is inverted to become a positive (1s complement) number and the multiplication performed, the answer is then inverted again to become a negative number which is added to n[index].

If there are multiple tables with the same axis then the table\_lu section only needs to be performed once - the index and residual will be the same.

**C code**

*index = Voltage >> 8; / / compiler will optimise this to a byte swap  
residual = (uint8\_t) (Voltage & 0x00FF);  
table[index] + ((table[index+1] – table[index]) \* residual);*

Need to deal with a negative number?

**Resolution**

The number of interpolation points is determined by the width of the table and width of the input variable. For a 16-cell table and a 12-bit variable there will be 255 interpolation points, if the variable is 8-bits then there are only 16 interpolation points.

16-bit RPM @ 1rpm per bit.  
Max rpm @8500rpm, effective resolution = 14bits.