Digital Filtering of Numeric Signals (First Order Time Constant)

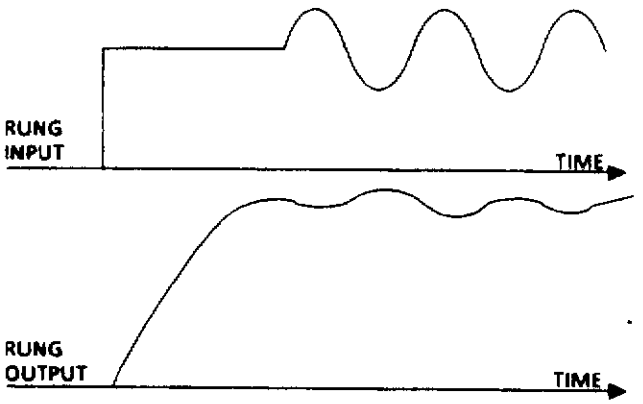
1. Description

The TCONST Special Function provides a digital filter for smoothing fluctuating numeric values, such as those resulting from process control measured variables when fed into a GEM80 Controller via analog input modules.

It is similar to the existing Special Function S37, ANALAG, but allows direct entry to the time constant required.

2. Features

- Equivalent to a single time constant filter in analog systems.
- User programmable time constant.
- Provision for switching the filtering action into or out of the system, and for initializing the output.
- In combination with gains (using LINCON, MULT or DIV), can be used to generate lead or lag compensators, or more complex transfer functions.



3. Specification

Control language:

```
          TCONST
X--------SPEC--------VALUE--------Q
          S60
```

Quantity of value locations : 6

Equation when filter switched out : Q = X

Equation when filter switched in : $Q = KX + (1 - K)Q_{n-1}$

where:

$$K = \frac{T}{T_c + \frac{T}{2}}$$

T = time since last execution
T_c = time constant
Q_{n-1} = output on last execution
subject to: $0 < K < +1$

When filtering is switched in:

1. If the input X remains constant, Q eventually reaches a steady level (i.e. $Q = Q_{n-1}$) equal to X.

2. The internal calculations within TCONST often result in a non-integer value for the desired output. Since Q can only be an integer, any fractional part that cannot be outputted is held over and used in the calculation for the next value of Q.

For operation in single cycle mode:

Time since last execution is artificially set to 100ms.

The following factors determine the choice of which table to use for the values Zm to Zm + 5:-

1. The table must not be write-protected.

   If it were, TCONST would not be able to store the values of Q_{n-1} and T required for the next execution.

2. Whether the Table contents are cleared on power up.

   You may want the values of Q_{n-1} and R to be retained while the Controller is powered down for some applications, whereas, for other applications, it may be unnecessary.

3. Whether the Table contents are accessible by video. (This only applies to controllers that include video).

   If you need to display any of the values Zm to Zm + 5, numerically or graphically, in a video format, the table used of must be accessible to video.

   Refer to the Product Data Sheet or the Technical Manual for the particular controller to find the attributes of the various Tables included in it.

Table 1 Program Rung and Value Data

Sym	Location	Use	Range
F	Zm	Fault code and flags. Set by TCONST	See Table 2
Tc	Zm + 1	Time constant. Set by user.	0-32,767 in units of 10ms. i.e. times of 0 to 327.67s
T	Zm + 2	Time at last execution. Set by TCONST.	-32,767 to +32,767 in units of 1ms. Recycles when +32,767 is reached.
HOLD	Zm + 3	Holds rung at previous value.	0 (zero) to hold. Any non-zero value for normal operation.
Q_{n-1}	Zm + 4	Previous output. Stored by TCONST.	±32,767. This value can be overwritten if required, e.g. for bumpless transfer.
R	Zm + 5	Previous remainder. Stored by TCONST.	
X	-	Input to TCONST from rung.	±32,767.
Q	-	Output from TCONST.	±32,767.

Table 2 Fault Conditions and Flags

Condition	Result	Code in Zm	
		Value	Bits Set
PROGRAMMING ERRORS			
No VALUE after SPEC.	Q = 0	-	-
VALUE given a number instead of an address.	Q = 0	-	-
VALUE address Zm in write-protected memory.	Q = 0	-	-
VALUE address Zm in table not usable by Special Functions.	Q = 0	-	-
Sixth address Zm + 5 beyond end of data tables memory area.	Q = 0	-	-
DATA ERRORS			
Zm + 1 contains negative number	Q = 0	5	Zm.0 = 1 Zm.2 = 1
T_c = 0	Q = X Q_{n-1} = X R = 0	9	Zm.0 = 1 Zm.3 = 1
STATUS			
$T>2T_c$	Q = X Q_{n-1} = X R = 0	9	Zm.0 = 1 Zm.3 = 1
HOLD ON	Q = Q_{n-1} R = 0	17	Zm.0 = 1 Zm.4 = 1

Note... If more than one error or status condition occurs at the same time, the bits in Zm will be set to 1 in combination. For instance, if Zm + 1 contains a negative number and HOLD is on, bits Zm.0, Zm.2 and Zm.4 will all be set to 1, giving a code value in Zm of 21.

4. Application Notes

4.1 Size of Time Constant

With TCONST, time constants with lengths from 0.01s to 327.67s can be set directly; that is , from 10ms to nearly 5.5 minutes.

Note... It is not possible to make a longer time constant by cascading two TCONST Special Functions. The response of two time constants in cascade is not the same as that of a single time constant.

If a time constant longer than 327.67s is ever required, use the S37 Special function ANALAG. This function is less user friendly than TCONST because it is necessary to set the ratio of the time constant to the program repetition interval. Also, it ceases operation when in a BLOCK that is not operated.

4.2 Method of Operation

Each time TCONST operates, it reads the time from the Controller. It compares this with the time held in Zm + 2 to find the elapsed time since its last execution. It then adjusts the output as given in the equations above, depending on this elapsed time. The value of time read from the controller has a resolution of 1 millisecond.

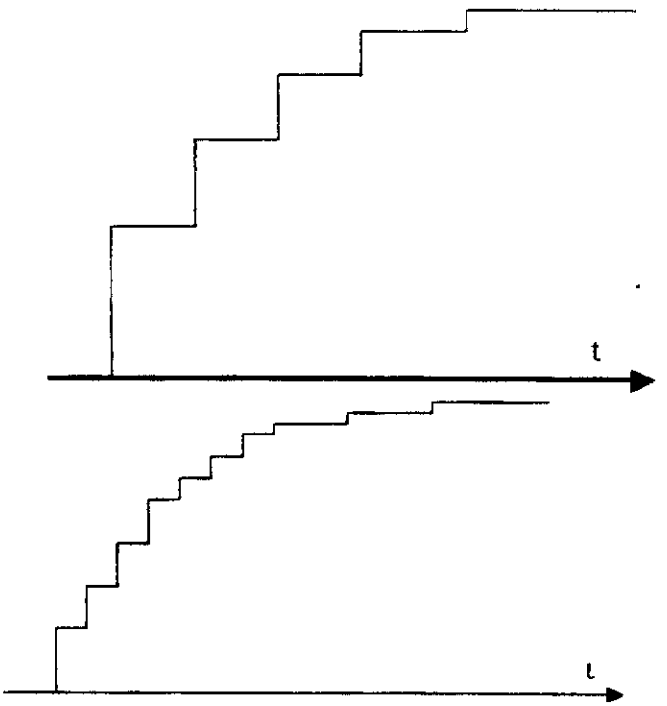
With TCONST it is possible to:

- alter the program cycle time with P-table value P1 without affecting the size of the time constant which has been set,

or

- include the TCONST function in a block that is operated less frequently than the program itself.

However, the output from TCONST will be smoother if the execution intervals are made shorter. The following diagrams illustrate this:-



If a time constant is set which is shorter than half the elapsed time since last execution, the smoothing factor K is set to 1, i.e. the rung output from TCONST equals the rung input.

4.3 Initialization

The first time that TCONST operates after power up, the time value held in Zm + 2 will either be zero or an irrelevant value from the last time the GEM80 was run.

As a result, TCONST would almost certainly calculate a spurious value for the elapsed time.

The same effect could occur if TCONST is put inside a block which is not operated for more than 65 seconds. The time read could then have been 'once round the clock'.

To prevent these effects occuring, it is necessary to initialize TCONST. Operate it just once to set up the time in Zm + 2 correctly, and while doing this, hold the output by setting Zm + 3 to zero. For all subsequent executions of TCONST, allow it to operate normally with table location Zm + 3 set to a non-zero value.

4.4 Step Response

Suppose that the Controller is operating with a constant program cycle time of 20ms. Suppose that you set a time constant of 30ms by setting Zm+1 to 3 (see Table 1).

$$\text{Then } K = 20/(30 + 10) \text{ (see specification)}$$
$$= 0.5$$

Consequently, the equivalent equation for the rung output Q is:

$$Q = 0.5X + (1 - 0.5)Q$$

$$= 0.5X + 0.5Q_{n-1}$$

Now suppose that, with the output initially zero, the ring input X is switched from 0 to 32. That is, $Q = Q_{n-1} = X = 0$ initially, and then X becomes 32. The value for Q subsequently will be:

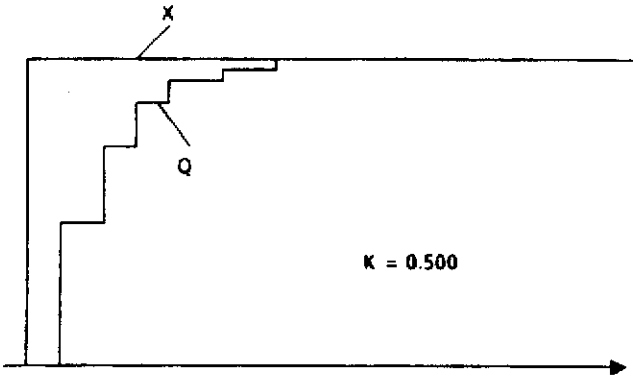
$$Q = 16 + 0.5Q_{n-1}$$

and Q will take on values on successive scan as follows:

16 (=16 + 0)
24 (=16 + 0.5x16)
28 (=16 + 0.5x24)
30 (=16 + 0.5x28)
31 (=16 + 0.5x30)
31 (=16 + 0.5x31 to the nearest integer)
32 (=16 + 0.5x31 + remainder)
32 (=16 + 0.5x32)

and the output will thereafter remain at 3 unitl the input X changes again.

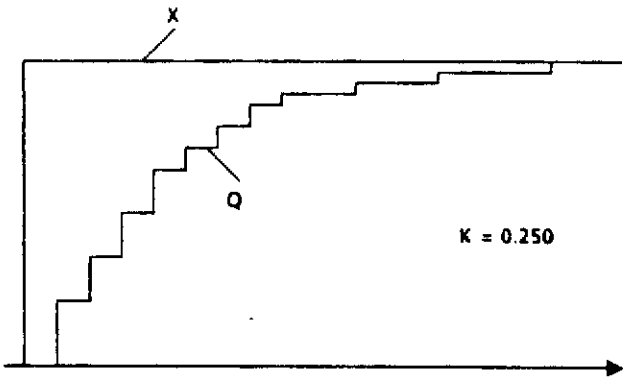
As a graph, the output Q for the previous example is as follows, where the time scale is drawn in units of the 20ms program repetition interval:-



Suppose you repeat the example for the same input conditions but with T_c = 70ms. This would make K = 0.25, and you would find by a similar calculation to that above that Q would take on values on successive scans of:

8, 14, 18, 22, 24, 26, 28, 29, 29,30, 30, 31, 31, 31, 31, 32 ....

giving a graph of:

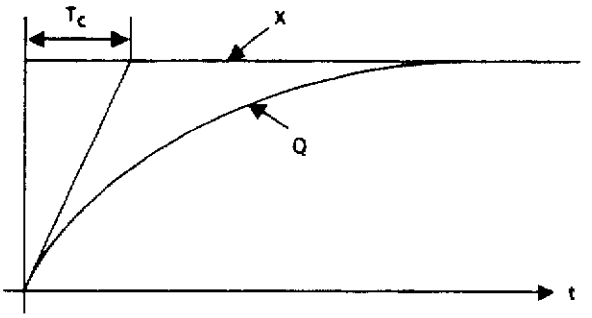


Note... Both these responses look rather 'lumpy'. This is because, in order to keep the examples simple, very short time constants have been chosen and a very small input step size (only about 0.1% of full scale). With longer time constants and larger step inputs, response graphs look much smoother, and approximate more closely to an equivalent analog filter.

With an analog filter, the response to a step input is a curve whose equation is:

$$Q = X(1 - e^{-t/T_c})$$

which looks like the diagram below:



It can be seen that this is generally of the same form as the curves previously given for the output of TCONST. Note the salient points:

- A tangent to the start of the curve intersects the final value after a time equal to the time constant T_c.

- At this time, the output is 63% of the final value

4.5 Fluctuating Signal

Suppose that, as in the last example, we have a program repetition interval of 20ms and a time constant of 70ms, giving an equation of:

$$Q = 0.25X + 0.75Q_{n-1}$$

Suppose that the input X is nominally 32 but with noise such that it cycles through values as follows:-

16, 44, 48, 20, 16, 44, 48, 20, 16 etc....

The output Q cycle through the following values:-

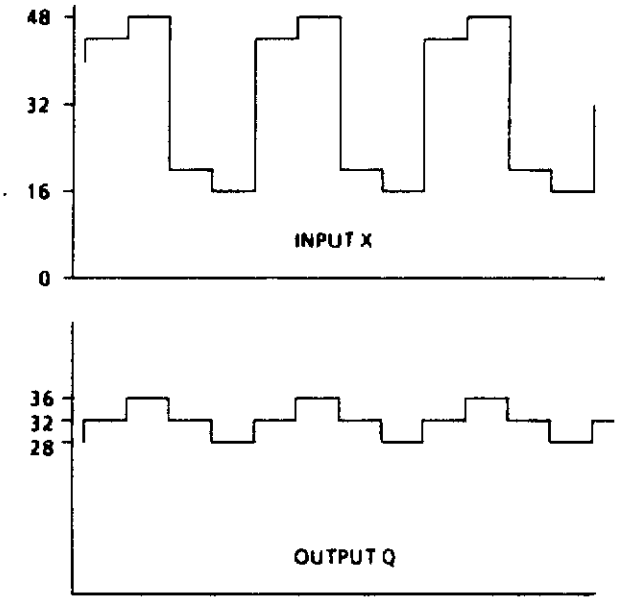
28, 32, 36, 32, 28, 32, 36, 32, 28 etc....

These output values can be checked from the equation earlier:

28 = 0.25 x 16 + 0.75 x 32
32 = 0.25 x 44 + 0.75 x 28
36 = 0.25 x 48 + 0.75 x 32
32 = 0.25 x 20 + 0.75 x 36
28 = 0.25 x 16 + 0.75 x 32

etc.

Thus, whereas the input signal X varies ±16 either side of the mean of 32, the output Q only varies ±4, and the graphs of the input and output are as follows:-



Note...The sequence of values used in this example has been chosen to make the calculations easy by not involving remainders. In practice, it is highly probable that each calculation of Q will involve a remainder, which is carried over to the calculation of the next value of Q.

The reduction in ripple on the output signal Q compared with the input signal X depends on:

(a) the value of the time constant T_c set in Zm + 1

(b) the frequency of the ripple.

For a sinewave input X, the attentuation is given approximately by:

$$A = \frac{2\pi fT}{K}$$

This formula is more accurate the higher the frequency, that is, when one period of the ripple is much shorter than the time constant. For the previous example, the period of the ripple was 80ms, and so its frequency was 1/0.08 = 12.5 Hz. The formula would therefore give, for a 70ms time constant, an attenuation of:

$$A = 2\pi \times 12.5 \times 0.07$$
$$= 5.50$$

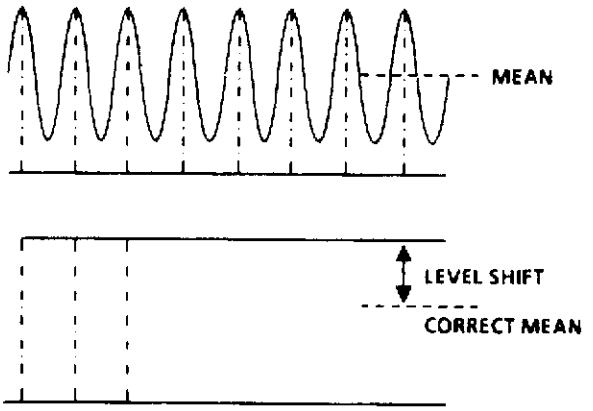
For the actual example with an input that was **not** sinusoidal, and where one period of the ripple was **not** much shorter than the time constant, an attenuation was obtained on the peak-to-peak size of the ripple of 4 times rather than 5.5. the formula, however, gave an answer of the right order.

4.6 Effect of Sampling on High Frequency Ripple

TCONST and similar types of digital filter are not suitable for attenuating ripple of frequencies higher than half the program repetition frequency. For this reason, all GEM80 analog input modules include hardware filtering for attenuating high frequencies.

Digital filters can't deal with high frequencies because of the sampling process. When a high frequency signal is sampled, the ripple on the resultant digital signal is of a **different** , lower frequency. The effect is known as aliasing. If the alias frequency produced by sampling is too low, it will not be attenuated by any digital filter such as TCONST.

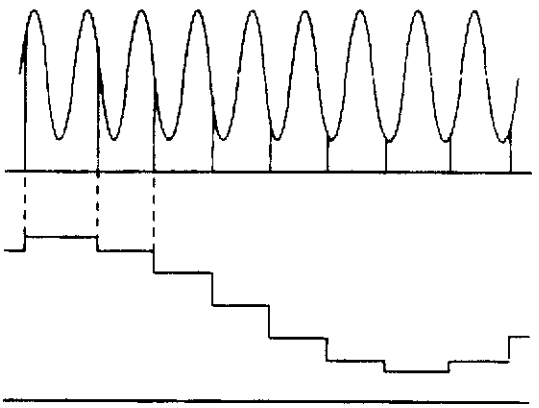
Consider a signal that contains ripple whose frequency precisely equals the program repetition frequency. The digital signal obtained by sampling it will be as shown below:-



where it can be seen that the sampled signal has a shift of level compared with the mean of the input signal. A digital filter such as TCONST cannot remove this level shift, as the signal into it appears exactly the same as a steady valued signal.

A similar effect occurs with sampling a signal whose ripple frequency is an exact multiple of the program repetition frequency.

Consider a signal with a ripple frequency close to, but not equal to, the program repetition frequency or some multiple of it. The effect of sampling is to produce a signal containing ripple at the beat frequency between the two frequencies, e.g.



The resulting frequency could be too low for TCONST or any similar type of digital filter to attenuate.

The above notes are intended to explain why digital filters cannot be trusted to attenuate high frequency ripple and why some hardware filtering is always included on analog input modules.

4.7 Holding and Pre-setting the Output

If Zm + 3 is set to zero, the output of TCONST will hold at whatever value it has reached. That is, its output will equal Zm + 4.

However, if Zm + 4 is overwritten (whether or not HOLD has been applied by setting Zm + 3 to zero), the TCONST output immediately jumps to a value equal to that written in Zm + 4 the next time the function is executed.

The HOLD facility is useful for bumpless transfer where it is necessary to change a signal from one source to another, without any change in value. A TCONST output can be pre-set equal to the size of signal being changed from, to avoid any bump.

4.8 Operation in Single Cycle Mode

If the controller is operated in single cycle mode, the time value increments by 100ms on each cycle. This allows the operation of TCONST Special Functions to be checked in most cases.

However, if a time constant value is set to less than 50ms, the output of TCONST will follow its input in single cycle mode without providing any smoothing.

Similarly, if time constant values are set to not much longer than 50ms, the smoothing effect of TCONST may only last for one or two cycles after an input change.

4.9 Switching Filtering Out

If you need to switch filtering in or out, it can be done by setting the time constant value held in Zm + 1 to zero. The output of TCONST will follow its input immediately without any smoothing.

4.10 Producing More Complicated Transfer Functions

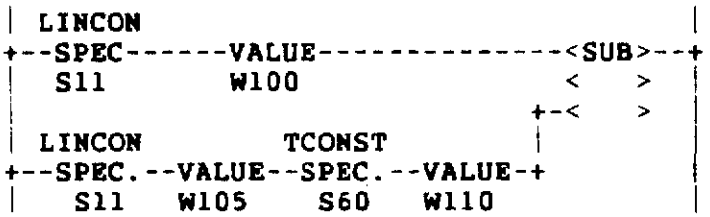
Quite complicated transfer functions can be produced using TCONSTs and gains provided by LINCONs, MULTs or DIVs. For instance, to produce a lead compensator with transfer function:

$$\frac{1 + p}{1 + 0.1p}$$

This transfer function can be produced using TCONST if it is re-written as:

$$10 - \frac{9}{1 + 0.1p}$$

In ladder diagram form it can be represented as:



The relevant W-table values for the Special Functions are:

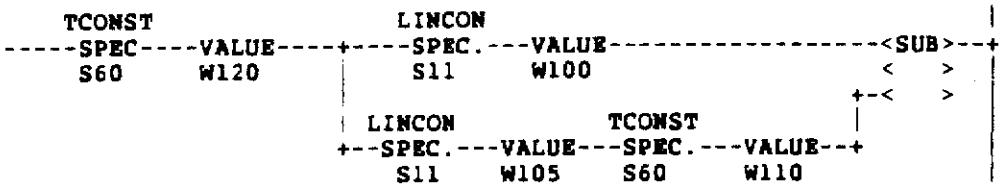
W101	10	(multiplier)
W102	1	(divisor)
W103	0	(offset)
W106	9	(multiplier)
W107	1	(divisor)
W108	0	(offset)
W111	10	(10 x 10ms = 100ms = 0.1s)
W113	-1	(or other non-zero value, to remove Hold)

The only thing to beware of in this implementation of the transfer function is what the maximum size of input is. Since the LINCON in the upper branch has a gain of 10, ensure that the input to this section of rung does not exceed ±3,276. If it did, the LINCON output would have to exceed the permissible number range of ±32,767, in which case the output would limit at ±32,767.

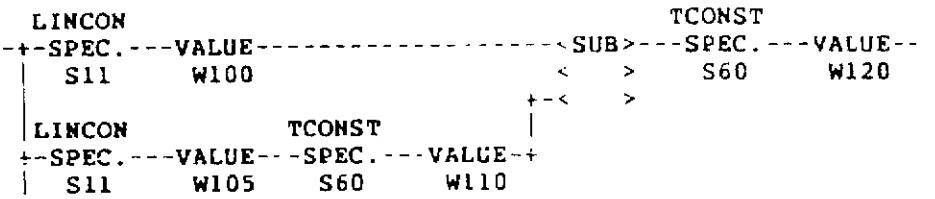
Next, consider the previous transfer function with a further time constant added, giving an overall transfer function of:

$$\frac{(1 + p)}{(1 + 2_p)(1 + 0.1_p)}$$

The extra 2 second time constant may be added at either end of the previous section of rung. That is, you could realise the transfer function either as:



or as:

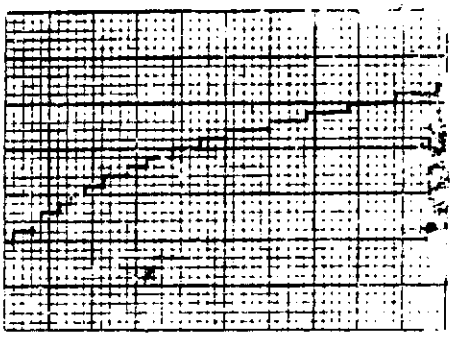


Although these two implementations of the transfer function are algebraically equivalent, the second one is much better. This is because digital systems are **not** exactly equivalent to their analog counterparts. Values can only be represented to the nearest integer. Because LINCON and TCONST Sepcial Functions round their output values to the nearest integer, their outputs tend to move in steps.

Wherever you have a LINCON or MULT Special Function whose gain is greater than unity, its output will move in steps of more than one bit. In the examples above, one LINCON has a gain of 10, so a 1-bit change of input will cause a 10-bit change in output. The second LINCON similarly can only give changes of output in 9-bit steps.

In the first implementation, with the additional TCONST at the start, whenever the output from this first TCONST changes by 1 bit, the output of the upper branch LINCON changes by 10 bits, and the output of the lower branch LINCON by 9 bits. However, the change in the lower branch is then smoothed by the following TCONST, so that the number reaching the lower of the <SUB>is less than 9. Consequently, the rung output changes by a step of maybe 5 bits; the actual change depends on the interval between executions.

In the second implementation, with the additional TCONST at the end, a 1-bit change in the rung input causes a change of maybe 5 bits as the change of output from the <SUB>. However, this is now smoothed by the following TCONST. The diagrams below show the response of the first and second implementation under identical conditions of program execution interval and input.



When complex transfer functions are implemented using several TCONST and LINCON or MULT Special Functions, the rules to follow are:

- Place the TCONSTs at the end and the LINCONs or MULTs at the start.
- If there are any places where the order of TCONSTs themselves is optional, place the longer time constants last.
- Check that input signals are not so large that they will cause any of the LINCONs or MULTs to limit.

4.11 Further Examples

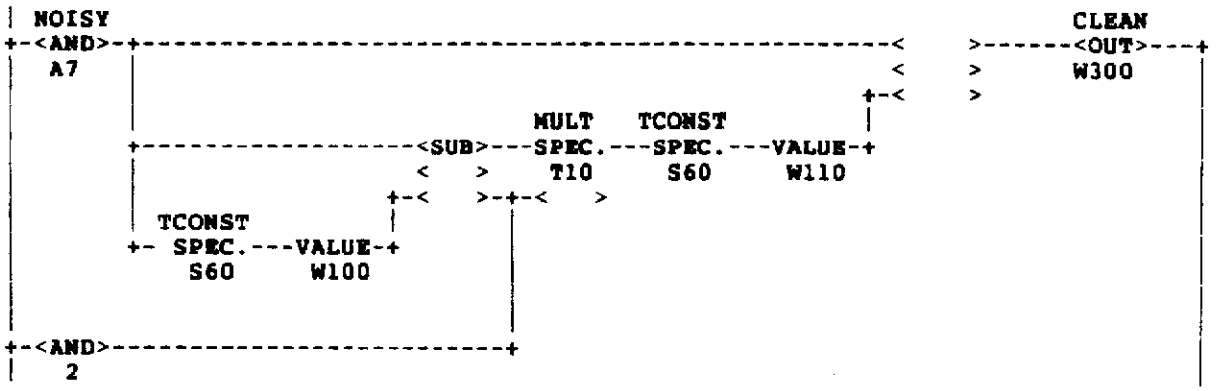
Without going into too much detail, further possible uses of TCONST are given below, to generate notch filters and quadratic terms. It is left as an exercise to the reader to verify these examples.

A notch filter can be generated with transfer function:

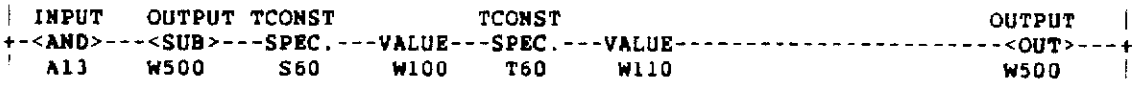
$$\frac{1 + T^2p^2}{(1 + Tp)2}$$

which can be re-writen as: $$1 - \frac{2}{1 + Tp}\left(1 - \frac{1}{1 + Tp}\right)$$

The following ladder diagram shows how this can be implemented.



With TCONSTs in cascade as in the examples dealt with so far, only functions with factorizable denominators can be produced. If, for example, a non-factorizable quadratic denominator is required, it can be produced with a rung such as:

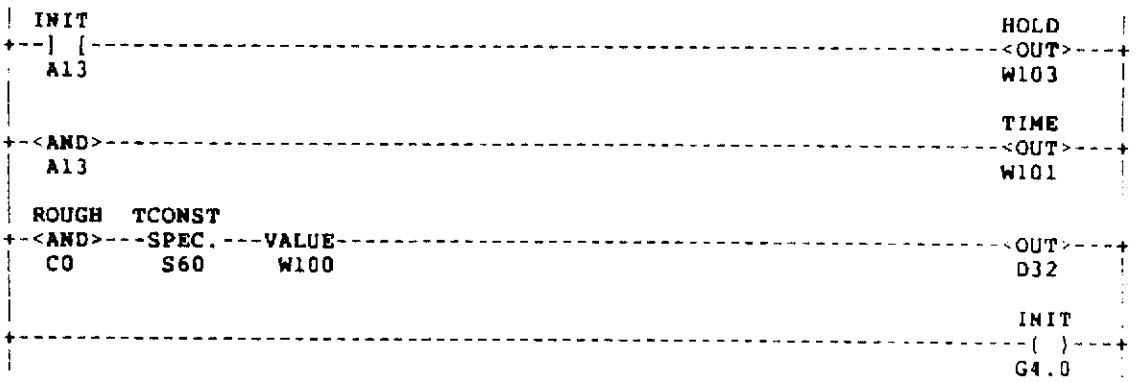


EXAMPLE

Requirements

To condition the analog input C0 with a 5s time constant filter and output it to D32.

Program Rungs



Explanation

In the third rung, the ROUGH input from C0 is fed via the TCONST Special Function to provide the SMOOTH output into D32. W100 holds any fault code. The time constant value is set into W101 by the second rung as 500 units of 10ms.

To initialize TCONST, the first rung sets W103 to zero, to hold TCONST during its first execution after power up. The fourth rung energizes coil G4.0, so that, on all subsequent executions of the program, W103 is set to -1 by the first rung. This allows TCONST to operate normally every time, except for the very first time it is encountered.

Note... It is not necessary to know the program repetition interval of the Controller, only that it is less than 65 seconds. The SMOOTH output from D32 will, however, be smoother the shorter the interval is.

Remarks

The above example illustrates the use of a program rung to set the value data. The same effect can be obtained by entering the data in the P-table and copying to a working value table on start up, using the Special Functions LOCATE (S20) and MOVE (T20).

CEGELEC
INDUSTRIAL CONTROLS

Kidsgrove, Stoke-on-Trent, ST7 1TW, England
Tel: (0782) 783511 Fax: (0782) 776329 Telex: 36293/4 CICKID G
Registered in England No. 2259095

T1309 Issue 1