ASTEP
Control software

List of tips and information

I –Floating point precision

The micro-controller does not have floating point support. For this reason, the software implements a simple floating point number that is represented by a 32 bit integer and an 8 bit power of 10 exponent.

What this means:

- Numbers can go from 0 to +/-2,147,483,647.
- The available precision limits the maximum range:
  - A range of 0 to +/-2,147,483,647 has only a maximum precision of 1
  - A range of 0 to +/-2,147,483 has a maximum precision of 0.001
  - A range of 0 to +/-2,147 has a maximum precision of 0.000001
- All numbers related to the same computation share the same range and precision

Temperature and PID parameters are the only modules requiring floating point parameters.

II – Temperature Calibration

The temperature calibration will define how much precision will be available in degree Celsius. The polynomial coefficients will define this as follows:

- The maximum precision of any coefficient will become the precision for any temperature command
- The computation must stay within the range for a valid range/precision chosen. This means knowing that the ADC range is 0 to 4095
  - The Coefficient zero must be within the range divided by 4
  - The coefficient 1 must be within the range divided by 16,384
  - The coefficient 2 must be within the range divided by 67,108,864

For instance, if a polynomial of degree 2 is required, this forces the precision and range to be very minimal and that the coefficient value divided by the precision cannot be more than 32. Valid coefficient range will be given by this table

Precision	Coeff 0 range	Coeff 1 range	Coeff 2 range
1 degree C	0 to +/-536,870,912	0 to +/-131,072	0 to +/-32
0.1degree C	0 to +/-53,687,091.2	0 to +/-131,07.2	0 to +/-3.2
0.01degree C	0 to +/-5,368,709.12	0 to +/-1,310.72	0 to +/-0.32
0.001degree C	0 to +/-536,870.912	0 to +/-131.072	0 to +/-0.032
…	…	…	…

Table 1: Valid temperature calibration coefficients ranges

Note: The precision also affects the melting point programming. If the temperature step requested in the melting point TSTP command requires more precision than available from the calibration values, the step will be rounded off to the available precision. This can cause some step to be set to zero. For example
CALTEMP 20.00,0.02,0.00
PROGRAM 0:TSTP,50,5000,0.001;1:LOOP,0,10
Will create a loop that does not increment temperature since the TSTP argument requires 3 digits precision but the calibration has only 2 digit precision.

Note: In order to check how the precision is affecting the temperature commands based on the current precision, the call back commands can be used:

- *PROGRAM? Returns the current program with precision correction for TEMP and TSTP instructions.*
- *CALTEMP? Returns the current calibration with the current precision correction*

III – PID loop

The PID loop has been implemented using also a basic floating point arithmetic. Once again, the precision used is the maximum precision that any of the P,I or D coefficients have. The maximum range for those coefficients is based on the time based requested for computing the PID loop. The following table summarizes the valid ranges for different time base and coefficient precision.

Time Period	Precision of 1	Precision of 0.1	Precision of 0.01	Precision of 0.001
Less than 160ms	Coeff P: 0 to +/- 2,048 Coeff I: 0 to +/- 128 Coeff D: 0 to +/- 32,768	Coeff P: 0 to +/- 204.8 Coeff I: 0 to +/- 12.8 Coeff D: 0 to +/- 3,276.8	Coeff P: 0 to +/- 20.48 Coeff I: 0 to +/- 1.28 Coeff D: 0 to +/- 327.68	Coeff P: 0 to +/- 2.048 Coeff I: 0 to +/- 0.128 Coeff D: 0 to +/- 32.768
Less than 320ms	Coeff P: 0 to +/- 1,024 Coeff I: 0 to +/- 32 Coeff D: 0 to +/- 32,768	Coeff P: 0 to +/- 102.4 Coeff I: 0 to +/- 3.2 Coeff D: 0 to +/- 3,276.8	Coeff P: 0 to +/- 10.24 Coeff I: 0 to +/- 0.32 Coeff D: 0 to +/- 327.68	Coeff P: 0 to +/- 1.024 Coeff I: 0 to +/- 0.032 Coeff D: 0 to +/- 32.768
Less than 640ms	Coeff P: 0 to +/- 512 Coeff I: 0 to +/- 8 Coeff D: 0 to +/- 32,768	Coeff P: 0 to +/- 51.2 Coeff I: 0 to +/- 0.8 Coeff D: 0 to +/- 3,276.8	Coeff P: 0 to +/- 5.12 Coeff I: 0 to +/- 0.08 Coeff D: 0 to +/- 327.68	Coeff P: 0 to +/- 0.512 Coeff I: 0 to +/- 0.008 Coeff D: 0 to +/- 32.768
Less than 1.28s	Coeff P: 0 to +/- 256 Coeff I: 0 to +/- 2 Coeff D: 0 to +/- 32,768	Coeff P: 0 to +/- 25.6 Coeff I: 0 to +/- 0.2 Coeff D: 0 to +/- 3,276.8	Coeff P: 0 to +/- 2.56 Coeff I: 0 to +/- 0.02 Coeff D: 0 to +/- 327.68	Coeff P: 0 to +/- 0.256 Coeff I: 0 to +/- 0.002 Coeff D: 0 to +/- 32.768
Less than 2.56s	Coeff P: 0 to +/- 128 Coeff I: 0 Coeff D: 0 to +/- 32,768	Coeff P: 0 to +/- 12.8 Coeff I: 0 Coeff D: 0 to +/- 3,276.8	Coeff P: 0 to +/- 1.28 Coeff I: 0 Coeff D: 0 to +/- 327.68	Coeff P: 0 to +/- 0.128 Coeff I: 0 Coeff D: 0 to +/- 32.768

Less than 5.12s	Coeff P: 0 to +/- 64 Coeff I: 0 Coeff D: 0 to +/- 32,768	Coeff P: 0 to +/- 6.4 Coeff I: 0 Coeff D: 0 to +/- 3,276.8	Coeff P: 0 to +/- 0.64 Coeff I: 0 Coeff D: 0 to +/- 327.68	Coeff P: 0 to +/- 0.064 Coeff I: 0 Coeff D: 0 to +/- 32.768
Less than 10.24s	Coeff P: 0 to +/- 32 Coeff I: 0 Coeff D: 0 to +/- 32,768	Coeff P: 0 to +/- 3.2 Coeff I: 0 Coeff D: 0 to +/- 3,276.8	Coeff P: 0 to +/- 0.32 Coeff I: 0 Coeff D: 0 to +/- 327.68	Coeff P: 0 to +/- 0.032 Coeff I: 0 Coeff D: 0 to +/- 32.768

Table 2: Valid PID coefficient ranges with precision

The grey rows are the recommended area of operation since a temperature PID loop requires a I coefficient and the realistic time period that should be used is from 200ms to 1s.

The PID controller operate based on the following derivation. We assume a system that we can control using a 12 bit DAC using programming values from 0 to 4095. The temperature is readout using a 12 bit DACs producing values from 0 to 4095. The PID controller will attempt to correct the output of the ADC by varying the DAC.

The Ideal parallel PID equation provides the new value that needs to be provided at the input given the error from the desired value at the output. The following equation shows the relationship:

$$NewValue(t) = K_p \cdot Error(t) + K_i \cdot \int_0^t Error(t).dt + K_d \cdot \frac{dError(t)}{dt}$$

With:

Kp: Proportional constant used to handle immediate error
Ki: Integral constant used to learn from the past. It is used when a non null input is required to produce a steady state output.
Kd: Derivative constant used to anticipate the future.

The Error is given as *Error(t) = SetPointValue - CurrentValue*

This equation can be dicretized over a time periods *T* by taking the derivative once more:

$$\frac{dNewValue(t)}{dt} = K_p \cdot \frac{dError(t)}{dt} + K_i \cdot Error(t) + K_d \cdot \frac{d^2 Error(t)}{dt^2}$$

The discretization is obtained by using successive difference equations

- $\frac{df(t-T/2)}{dt} \approx \frac{f(t)-f(t-T)}{T}$ using a first order difference equation
- $\frac{d^2 f(t-T/2)}{dt^2} \approx \frac{d\left(\frac{f(t)-f(t-T)}{T}\right)}{dt} \approx \frac{f(t)-2f(t-T)+f(t-2T)}{T^2}$ using a first order difference equation twice

Because the difference equations are valid for a time in-between two time samples, the actual time value must be approximated for this time at which we do not have any data.

We therefore have $f(t-T/2) \approx \frac{f(t)+f(t-T)}{2}$

We can then rewrite the PID equation using the discretization approximation at time t-T/2. We have:

$$\begin{aligned}\frac{NewValue(t)-NewValue(t-T)}{T} &= K_p \cdot \frac{Error(t)-Error(t-T)}{T} \\ &+ K_i \cdot \frac{Error(t)+Error(t-T)}{2} \\ &+ K_d \cdot \frac{Error(t)-2.Error(t-T)+Error(t-2T)}{T^2}\end{aligned}$$

If we use the notation indexing the various function value at a given sampling time such that *f(k) = f(kT)*

We have:

$$\begin{aligned}NewValue(k) &= NewValue(k-1) \\ &+ Error(k).\left(K_p + \frac{TK_i}{2} + \frac{K_d}{T}\right) \\ &+ Error(k-1).\left(-K_p + \frac{TK_i}{2} - \frac{2K_d}{T}\right) \\ &+ Error(k-2).\left(\frac{K_d}{T}\right)\end{aligned}$$

The equation in this final form is used in the microcontroller with the following notation for ease of understanding:

$$\begin{aligned}DACValue(k) &= DACValue(k-1) \\ &+ (SetPoint(k) - ADCValue(k)).\left(K_p + \frac{TK_i}{2} + \frac{K_d}{T}\right) \\ &+ (SetPoint(k-1) - ADCValue(k-1)).\left(-K_p + \frac{TK_i}{2} - \frac{2K_d}{T}\right) \\ &+ (SetPoint(k-2) - ADCValue(k-2)).\left(\frac{K_d}{T}\right)\end{aligned}$$

Note: In order to test the functionality of the PID loop, several commands can be used for monitoring the continuous behavior:

- *HEATSET #1000: for instance would monitor every second the DAC value set by the PID loop*
- *TEMP #1000: for instance would monitor every second the ADC values directly read from the ADC that are used to define the new DAC value.*

Note: Reading the diodes while the PID loop is actively trying to set a temperature is not a really good idea as it will cause the actual PID time period (as opposed to the programmed one) to be affected during the integration time. This is especially true when:

- *Diode command integration time is large or comparable to the PID period time.*
- *A lot of diode commands are issued in a row (diode streaming for instance)*

When using the PIDTEMP command, the two last arguments correspond to threshold used for turning the Fan on and off. Those thresholds are applied to the temperature difference between the current ADC temperature and the requested ADC setpoint. There are two values in order to provide hysteresis:

- The fan will turn on if the actual temperature is larger than the setpoint by the first threshold.
- The fan will turn off when the actual temperature is less than the setpoint by the second threshold.

Note: the thresholds can not be more than 4095, since this is the maximum range of the ADC. Temperature calibration should help define roughly how many ADC values correspond to a proper temperature offset that would require the fan to be turned on.

IV – Programs

The micro controller provides the ability to enter programs using 5 main instructions. Here is a list of tips and notes when using this feature:

- The TEMP and TSTP instructions require a valid calibration and a functioning PID loop. The precision used here will be truncated to the calibration coefficient precision.

  Note: The precision correction can be checked by retrieving the program using the PROGRAM? command
  Note: If the PID loop is not tuned or a temperature setpoint is chosen so that it cannot be reached, the program will stay in this instruction for ever and only an ABORT command will allow the user to end the program

- The program always start at line 0 when the STARTPROG command is used. This means that line 0 must have an instruction different than ENDP. When a program is created, it should always start at line 0.

- Each line of the program can be loaded independently. This is especially usefull when using hyper terminal and changing a few parameters without having to re-type the entire program.

  Example 1: PROGRAM 0:READ,100,100,100,100;1:LOOP,0,10;2:ENDP

  Is equivalent to

  PROGRAM 0:READ,100,100,100,100
  PROGRAM 1:LOOP,0,10
  PROGRAM 2:ENDP

  Example 2: PROGRAM 0:READ,100,100,100,100;1:LOOP,0,10;2:ENDP

  Can be easily modified to loop 15 times instead of 10 by just typing

  PROGRAM 1:LOOP,0,15

  This would be equivalent to re-typing the entire following line

  PROGRAM 0:READ,100,100,100,100;1:LOOP,0,15;2:ENDP

- GOTO is not a useful instruction unless the user knows what he is doing. This is because this small program language does not provide branching instruction. GOTO should be only used to bypass some instructions while in debug mode without having to reload the entire program
- The READ instruction is being multiplexed with PID loop. One PID loop computation will have the chance to be executed between each diode readout. Diodes are read in numerical order, Diode1, Diode2, Diode3 and Diode4
- When the micro controller is first turned on, the default program has all of its line set to ENDP. Executing this program with STARTPROG will not do any harm as it will terminate immediately.
- The program command has a limitation on the number of lines it can handle. There are only 64 lines of program available from line 0 to line 63. Attempting to set a line greater than 63 with an instruction will cause the PROGRAM command to fail.
- The program command has a limitation on how many READ instructions can be performed in one program. When the STARTPROG command is launched, it reset all the data to zero (data is accessed by the DATA? Command). The data buffer has a length of 256 records for now. This means that a program cannot loop over the READ instruction more than 256 times. This limits can be increased upon requests as we still seem to have some room in the RAM to store some more.

  Example: PROGRAM 0:READ,100,100,100,100;1:LOOP,0,300;2:ENDP

  This will execute and physically will read the diodes 300 times but only the first 256 reads will actually be stored and retrieved with the DATA? Command.

V – Error Messages

When a command is entered through RS232, two messages can appear if the '?' character was appended:

1. "OK": This means that the command was entered properly and executed successfully as far as the controller knows. The controller does not always knows that an error occurred.
2. "FAIL": This means that the command was invalid or that the controller could not successfully execute the command. After a FAIL, the command STATUS? can be used to retrieve the last error. As a matter of fact it should always be used when FAIL has been received in order to clear the last error.

When RS232 commands are entered, the following error messages can be generated:

- "ERROR:Cmd Overflow": this means that the command is too long and cannot fit in the input buffer. This should never be an issue for normal commands except for the PROGRAM command. The command buffer has a maximum length of 1024 characters
- "ERROR:Cmd Unknown": this means that the command is not part of the valid command set. Watch for typos!!
- "ERROR:Busy": this means that the command cannot be executed because the micro-controller is busy running a program. In short, the command has been recognized successfully but is not allowed at this time.
- "ERROR:No Streaming": this error message is exclusively associated with the STPSTREAM command. This means that the stop stream command cannot be executed as there is no active streaming.
- "ERROR:Hardware Error": this means that the command has been recognized successfully and that an attempt to execute the command has occurred but that for some reason the hardware reported an error. This error message is mainly related to command turning devices on and off such has the LED and FAN command.
- "ERROR:Invalid Stream Period": the period set for the stream option is not valid.
- "ERROR:Invalid argument": this means that the program entered has been recognized but that the argument values are:
    - out of range: most raw setpoint values are from 0 to 4095
    - in the wrong format: only temperature, calibration and PID parameters are allowed to have floating point argument (numbers with a dot such as 0.1)
    - in the wrong sign: most values should be positive numbers except for the temperature calibration and the PID parameters.
    - Note that this does not detect precision and range issues stated in Table 1 and Table 2
- "ERROR:No Data Stored": this error message is exclusively associated with the DATA? command. This means that there is no data available in the buffer. This is most likely caused by the fact that no program has been run yet that would store data using the READ instruction.
- "ERROR:Program Overflow": this error message is exclusively associated with the PROGRAM command. This means that the user requested more than the 64 available line in the program. Please only use line 0 to line 63 when creating a program.

When a program is entered, a syntax check is run for each line. If the syntax check fails, the micro-controller will return FAIL and the available error messages though the STATUS? command will be:

- "ERROR:LINE <num> Syntax Error": this indicates that in program line <num> the instruction was not recognized as a valid instruction. Check for typos on the instruction name and the number of arguments.
- "ERROR:LINE <num> Invalid Argument": this indicates that at line <num> the instruction was valid but the arguments are of wrong sign or format or that there are not enough of them.
- "ERROR:LINE <num> Argument out of range": this indicates that at line <num> the instruction was valid but the arguments are out of valid range. Please note that this does not detect loss of precision.