Application Note AN2208

Universal PID-Thermoregulator

Author: Andrew Smetana
Associated Project: Yes
Associated Part Family: CY8C27xxx
PSoC Designer Version: 4.2
Associated Application Notes: AN2120, AN2148

Abstract
This Application Note describes implementation of a digital temperature PID controller. Various temperature control system configurations and aspects of their practical implementation are analyzed. Two types of temperature sensors are applied: thermocouple and platinum RTD sensor. The PSoC™ device supports two heat power control methods: phase and numeral impulse. The wide variety of system configurations allows use of thermoregulators in different industrial, commercial, and residential systems, where temperature control is needed.

Introduction

Regulators can be developed using analog and digital techniques. Different mathematical methods are needed to analyze and design analog and digital regulators. Though digital technology can replicate analog system operation, its abilities go much further. For example, nonlinear and self-adjusting systems, which are difficult to create using only an analog system, can be designed. The main issue in digital control is regulator structure and parameter definition.

After the parameters are determined, implementation of controller algorithms is a simple task.

Regulator systems are widespread in industry applications. In many cases, the process is passed with a preset temperature profile. These applications need a corresponding regulator to satisfy process requirements. The structure of the simplest regulator is presented in Figure 1.

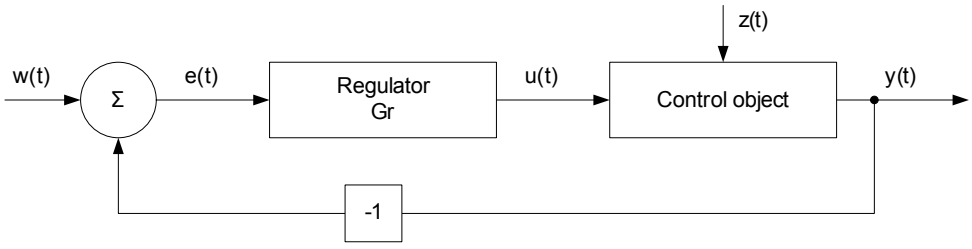


Figure 1. Structure of the Simplest Regulator

This structure presents an automatic control system with feedback. See the following definitions:

w(t): System function algorithm
u(t): Control effect
z(t): External disturbance impact, which must be minimized
y(t): Output variable
e(t) = w(t) – y(t): Output variable y(t) deviation from required value w(t)

Examples of output variables are: temperature in the stove, the engine shaft rotation speed, liquid level in the cistern, etc. The key to temperature control is to constantly adjust the output variable, y(t), so that it is near the value of w(t). Doing this, will minimize the control error, e(t).

Temperature adjustments can be made with an automatic Regulator, Gr (Figure 1), which is described by control law:

u(t) = Gr[e(t)].

To select the correct control law, the automatic regulator must know the mathematical model of the control object:

y(t) = Go[u(t)].

The mathematical model is usually a nonlinear, ordinary system of differential equations or differential equations in partial derivatives. Identifying the form and coefficients of these equations is done via the control object identification task. For conventional systems, mathematical models are commonly used and then the principal task is identification of equation coefficients. In many cases, these coefficients can be selected empirically during the system tuning process or by performing some special tests.

Some features of control systems with feedback indicators are:

- Independent corrective action initialization when control variables deviate from reference values.
- Dynamic regulation of temperature variation with minimal detail.

The control law in Equation (1) is the main factor for designing automatic control systems. Ideally, an optimal regulator synthesis could fulfill these requirements. However, it is a challenge to find one of good quality at an economical price. Inexpensive alternatives used in many industry applications are the simplest and most common types of linear regulators: P-, PI- and PID-regulators.

The ideal equation of a PID-regulator is:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau + T_D\frac{de(t)}{dt}\right] \qquad \textbf{(1)}$$

Where K is the controller gain, T_I is the integral time constant and T_D is the derivative time constant.

These three parameters can be selected during the regulator tuning process to calculate the system functioning algorithm.

The described automatic control system is continuous. That is, it uses continuous time.

Using microprocessor techniques during construction of the regulator, the input and output variables must be measured in time and converted to digital using the ADC. At the same time, the PID-regulator equation should be transformed by changing the derivatives by finite differences and integrals by finite sums. Using the following method for substitution of integrals by finite sums, yields:

$$u(k) = K\left[e(k) + \frac{T_0}{T_I}\sum_{i=0}^{k} e(i-1) + \frac{T_D}{T_0}\left[e(k) - e(k-1)\right]\right] \qquad \textbf{(2)}$$

$k = 0,1,...\frac{t}{T_0}$ counts out the discrete time.

The advantage of digital regulators is their ability to operate remotely to easily exchange data through multiple communication channels. Despite this advantage, the analog method is still the most reliable, most common, and its data is easily converted to digital.

Regulator Characteristics

The developed regulator is implemented using principles of a digital PID-controller. Its main task is maintaining the object's preset temperature level in low- or high-wattage heaters. For example, a low-wattage heater could be a soldering iron, where a preset soldering temperature is required. An example of a high-wattage heater is an industrial electric stove. For temperature sensors in this project, both resistive temperature detectors RTD (HEL-700) and thermocouple (T-type) sensors are used. Both are widely used in the industry. There are two methods for power transfer: phase and numeral-impulse. The numeral-impulse method is used to control the inertial load, which works on the principle of sending in load some half-cycled of AC mains with an immediate delay. In this project, this method was modified to take into account constant temperature fluctuation.

To simplify the tuning process for the thermoregulator, the EIA-232 (RS-232) interface is used and a command language developed. Therefore, it is possible to use standard terminal programs to set system parameters and obtain current status information. The user-friendly interface to control the device and monitor the regulator process was developed using the Win32 Application and C++Builder.

Using this setup, the universal thermoregulator with different kinds of sensors, loads and power control methods is determined. The desired combination of system functions is dependent on industry regulator requirements.

Regulator Flowchart

The regulator flowchart is presented in Figure 2. Note that the gray blocks identify the external units for the PSoC device.

The regulator structure consists of two main parts: a synchronization system with a power control block and a measurement module with a PID-controller. These two subsystems work independently and concurrently. The central unit of the synchronization system is the interrupt service (*Network_Sync_ISR*), which is called each time a signal in the AC network crosses zero level. Then this subroutine reads the last value of the control signal (*PID control value*), which is formed by the PID-controller block (*PID controller*). Depending on the power control method, the phase method or numeral-impulse method is initiated to generate the control signals. The power control method settings are determined during PSoC device initialization after reset, by questioning the jumpers (*JUMPERS*). The phase method uses a pulse width modulator (*PWM*) to form the control signal.

This signal is used to control the electronic switch *Power Switch* (FET for the low power heater or optotriac for the high power heater).

The analog piece of the synchronization system is implemented by an external node (*Network Sync Circuit*) and internal PSoC blocks. The zero crossing detection module (*Zero Crossing Detection*) uses the Schmitt trigger scheme, which features internal hysteresis. This approach preserves generation of multiple false synchronization signals, especially in moments of load commutation. Upon output from these blocks, a digital signal is formed, which generates an interrupt signal.

Consider the working principles. The active block of this subsystem is the translation module (*Translation Module*). It selects the sensor to read its signals, depending on the jumpers' states. Sensor selection is completed by means of an internal multiplexer (*MUX*). The multiplexed signal is gained by the instrumentation amplifier module (*INSTR AMP*). The gain factor is different for the thermocouple and RTD. The analog signal is then converted into a digital representation using ADC_1.

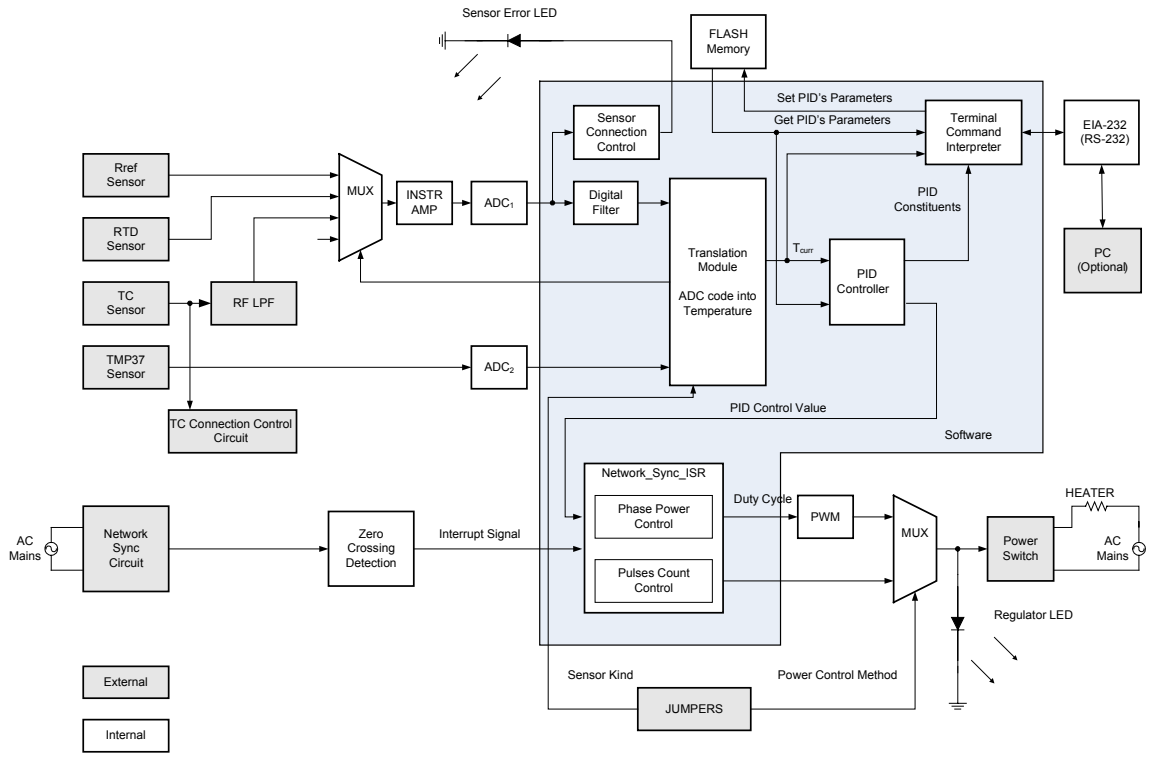


Figure 2. Regulator Flowchart

To prevent external noise from influencing the regulation process, the digital signal is passed through a nonlinear digital filter (*Digital Filter*), on which the output code ready for processing is set. The translation module (*Translation Module*) translates the ADC code into temperature values using the sensors' characteristic tables. The translation methods and specific measurement methods between the thermocouple and the RTD are different. Thus, it is necessary to read the voltage using RTD measurement on two resistors, *Rref Sensor* and *RTD Sensor*, at the same time. The RTD resistance is then calculated and the corresponding temperature value found using a table. The thermocouple measurement method uses an additional low cost temperature sensor *TMP37*, which measures the thermocouple's cold junction temperature. To accomplish this, it utilizes ADC_2. Note, ADC_1 and ADC_2 work simultaneously. To get the absolute temperature of a control object, the thermocouple and *TMP37* temperatures are calculated. The object's temperature is then passed to a PID-controller unit, which compares this value with a reference temperature and generates corresponding control signals.

To reduce unwanted influence of stray radio noise on thermocouple wires, the signal is passed through an external RC low-pass filter (*RF LPF*). The sensor break control is provided by two units: *TC Connection Control* and *Sensor Connection Control*. If a sensor break is detected, the red LED turns on (*Sensor Error LED*), the regulator output is set to zero (i.e., it disconnects load), and the device waits for sensor connection.

A UART User Module is used for debugging and to tune the thermoregulator block. The regulator's parameters are loaded through the PSoC's Flash memory. The UART also provides a means for receiving regulator status. The software implemented *Terminal Command Interpreter* block interprets the command language of the device terminal module.

Power Control

Two methods can be used to transfer power in the load:

- Phase method, in which the value of power delivered is determined in changing of phase angle.
- Numeral-impulse control is performed by whole half-cycles. Impulses are directly carried after voltage in the AC mains crosses zero level *and* during the time the load is connected to the AC mains. Some half-cycles are passed through the load during this time.

The numeral-impulse method is utilized for power control in loads with long reaction times (inertial load). The advantage of this method is that load commutation moments concur with moments of zero crossing, so the level of radiated radio noise is sharply reduced. The minimal amount of energy entered in the load equals the energy that is supplied during one AC mains half-cycle. For example, to get a 10% increase, there is a need to have a period of 10 half-cycles. In Figure 3A is a sequence of impulses on control electrode for 30% power in load. The electronic switch is on during the first three half-cycles, and off during the last seven. This sequence is repeated.

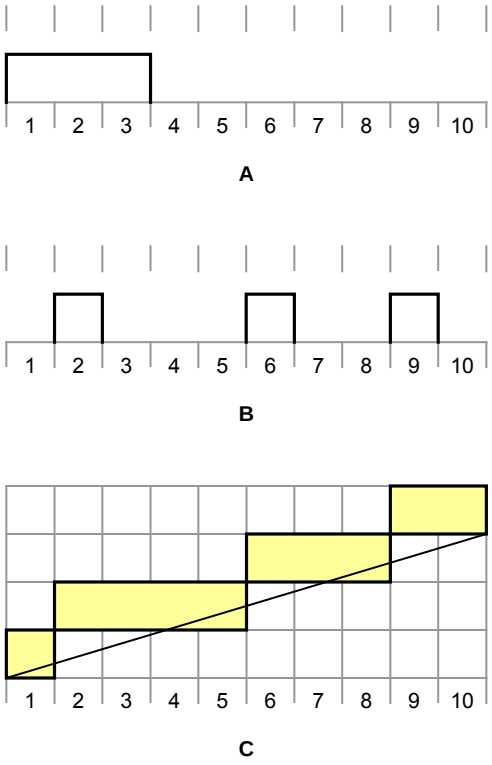


Figure 3. Numeral-Impulse Method

It is more logical to distribute half-cycles uniformly during the whole sequence period when the switch is on. The problem of uniform distribution of N impulses in sequence with length M (N ≤ M) is solved by Bresenham's algorithm, which is used in raster graphics for drawing slanting lines. This algorithm is implemented using integer arithmetic, simplifying the programming. Figure 3B depicts impulse distribution for the same 30% power, but using Bresenham's algorithm. The raster line drawing using the same algorithm is shown in Figure 3C. In this project, Bresenham's algorithm provides gradual temperature adjustments without large fluctuations during the regulation period.

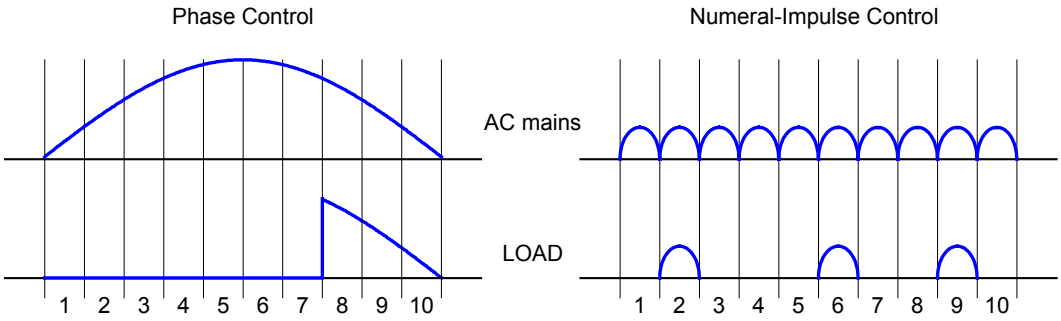


Figure 4. Power Control Method Comparison

Power control method comparisons are shown in Figure 4. Note, a 30% half-cycle time of the switch on in phase method, does not correspond to 30% power transfer in load. For sinusoidal signals, the output power has nonlinear dependence on electronic switch phase angle opening. This feature does not prevent correct temperature regulation since the system includes feedback contours.

Hardware Implementation

The hardware solution of this project is presented in two parts: the PSoC internals and the regulator's external schematics.

Regulator Schematics

The regulator schematic circuit is presented in Figure 5. The connecting link of all components is the PSoC device, which reads and processes sensor signals. It then generates control signals and provides communication through the UART.

To provide network synchronization, the following elements are used: a transformer, a diode rectifier, and also a circuit formed from resistor *R14* and Zener Diode *D5* with a stabilizing voltage, 3.3V. The output *Net_Sync* of this circuit is connected to the PSoC comparator. The resistor *R15* is used for discharging stray capacitance, which is not able to reset the sinusoidal signal to reach zero level.

The resistors *R16, R17* and FET *Q1* are used to control the low power load. This load on the schematic is depicted as resistor *R12*. The optotriac *U5* is used to control high power heaters. The control signal, *Gate Control*, is generated by the PSoC. The regulation process is provided by LED *D6*.

The power supply of digital electronics is recognized by the voltage regulator L78M05 and corresponding capacitors/filters *C9*, *C10*, *C11*, *C12*. The supply unit diode *D2* prevents smooth signal flow into the synchronization circuit.

The regulator uses three kinds of sensors: resistors (R11, RTD1), thermocouple, and temperature sensor TMP37. For temperature measurement on the basis of platinum RTD, a 4-wire measurement circuit is used. The advantage of this method is precise measurement regardless of the length of connecting wires (from sensor to board). The RTD sensor resistance is calculated according to the following equation:

$$R_{RTD} = \frac{V_{R_{RTD}}}{V_{R_{11}}} R_{11} \quad \textbf{(3)}$$

The circuit for measuring voltage on the thermocouple is presented by functional blocks. Components *R5*, *R6*, *R8*, *R9* and *C3*, *C6*, *C7* form a low-pass filter to filter out RF noise. Resistors *R1*, *R2*, *R3 and R4* are used to bring the thermocouple signal into the working range of the instrumentation amplifier. For the thermocouple connection, two digital pins are utilized (P2[5] and P2[7]). These pins are set to logic 1 and 0. If the thermocouple breaks, the ADC output is set at the maximum code and the LED *D1* turns on. The RTD connection control is achieved without use of digital pins; only the ADC code is controlled digitally. In temperature measurement mode, P2[5] and P2[7] are in High-Z state. The sensor, TMP37, is used to measure temperature of thermocouple cold junction. According to its output, the set voltage is directly proportional to the temperature of its package. This sensor provides 500 mV output at 25ºC and its scale factor equals 20 mV/ºC. The absolute temperature is measured as:

$$T = T_{TMP37} + \Delta T_{TC} \quad \textbf{(4)}$$

The type of sensor (thermocouple or RTD) and power control method (phase or numeral-impulse) can be selected using jumper *J2*.

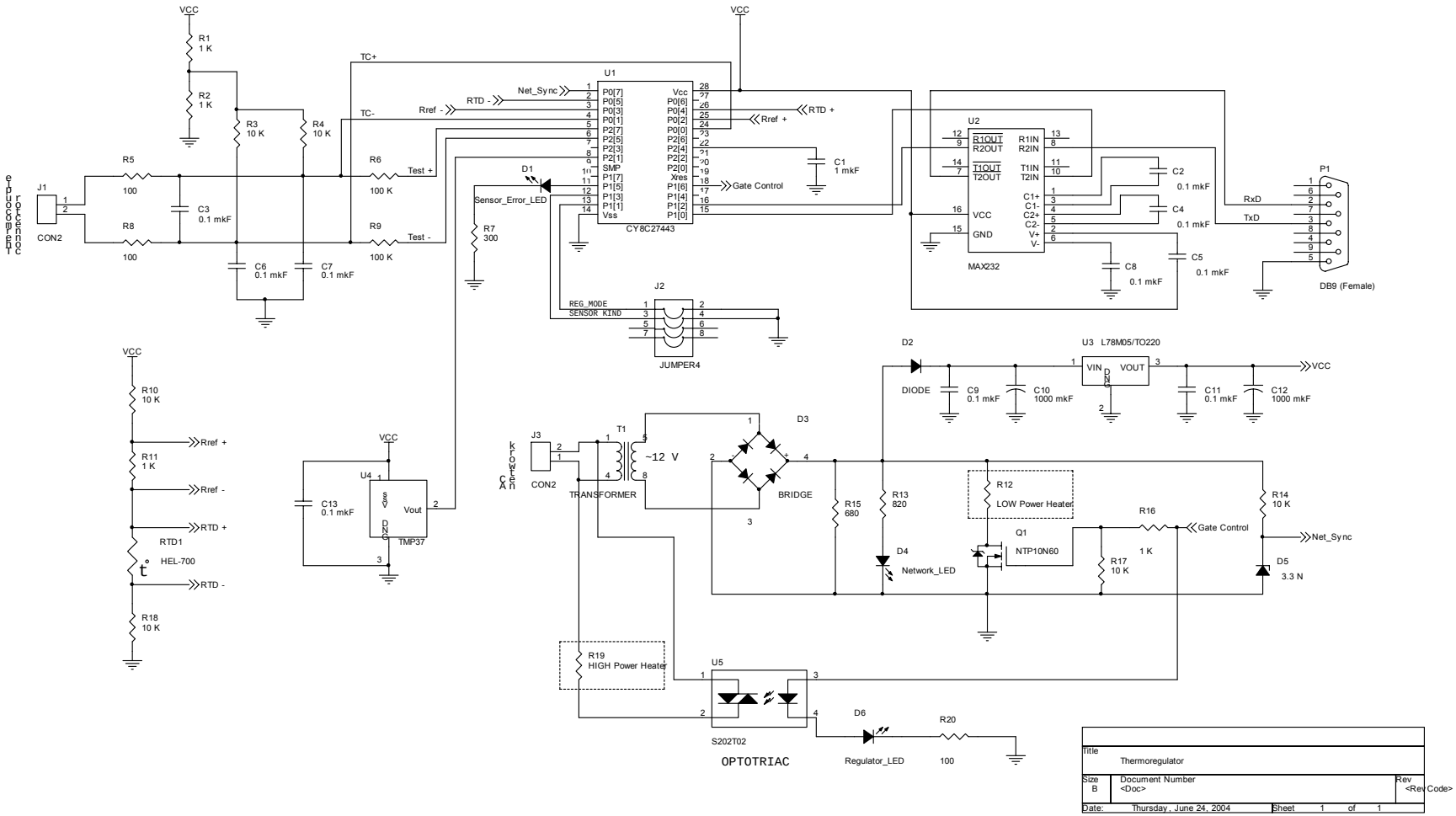


Figure 5. Regulator Schematic Circuit

To provide communication with the PC through the COM port, a standard level shifter, such as MAX232, can be used.

PSoC Internals

The internal chip configuration is presented in Figure 6. Implementation of the internal structure consists of the following components: an instrumentation amplifier (*INSAMP*), two ADCs (*ADC*) united in a single module, a comparator (*CMP*), a digital communication module (*UART*), and the pulse width modulator (*PWM*).

To gain signals from the sensors, the instrumentation amplifier is used and is placed in analog blocks *ACB02*, *ACB03*, *and ASD13*. For multiplexing the input signal from the sensor, the multiplexers in 2-d and 3-d analog columns are used. The bit assignments for the amplifier inputs are defined in register *AMX_IN*.

The amplifier zero shift measurement is performed by short circuiting its inputs. For this, the input's multiplexer in 2-d analog column is used (controlled by register *ABF_CR0*). The gain factor is dependent on the sensor type. The gain signal is given on the input of the ADC_2

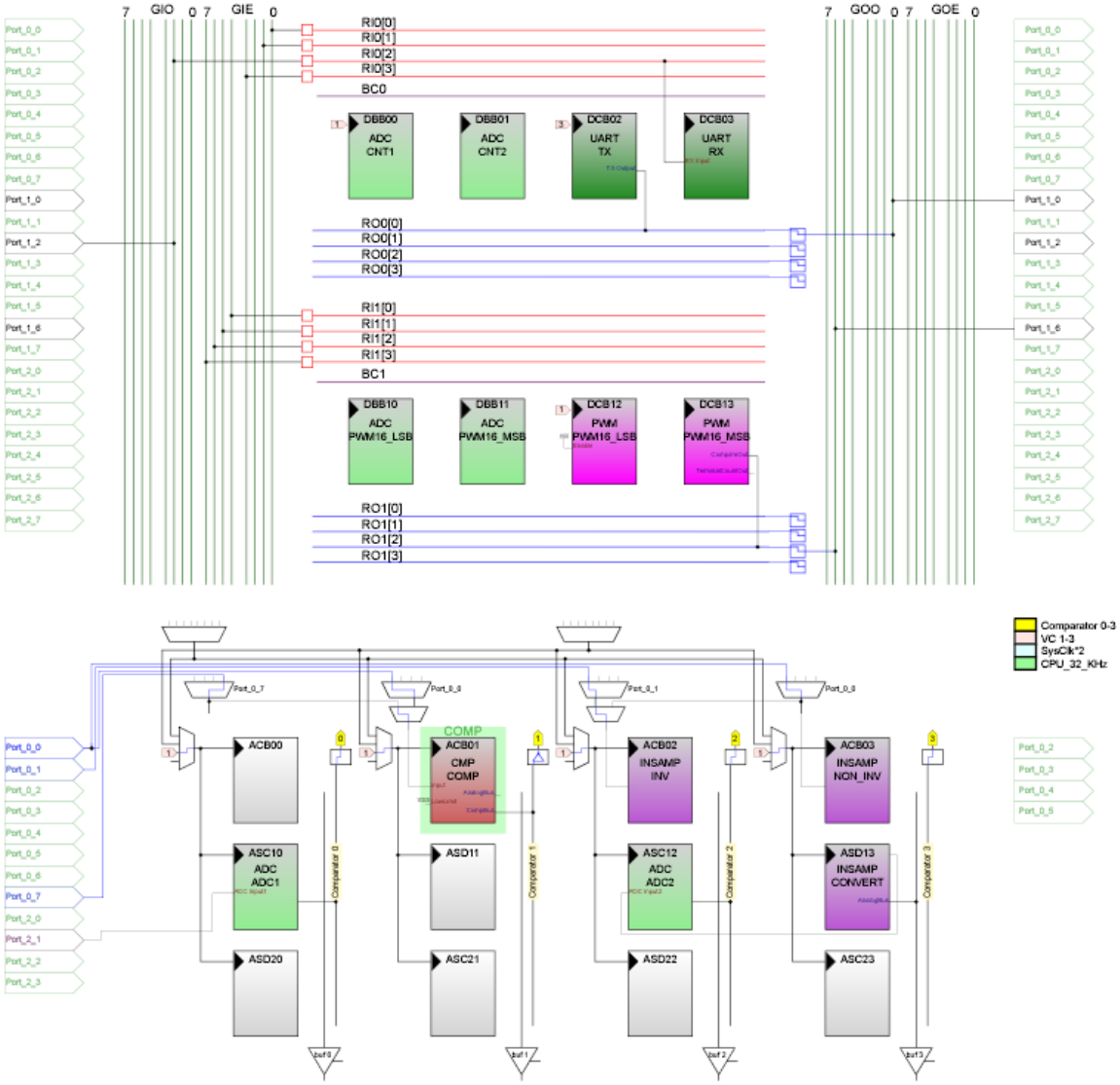


Figure 6. PSoC Internal Configuration

This ADC_2 property *ClockPhase2* is set to *SWAP* to provide synchronization with the output of the instrumentation amplifier. The ADC is a dual ADC and occupies the following analog and digital blocks: *ASC10*, *ASC12*, *DBB00*, *DBB01*, *DBB10*, and *DBB11*. Another ADC (ADC_1) is used to measure the signal from the TMP37 sensor. Its input is connected directly to P2[1].

The comparator (*CMP*), in *ACB01*, provides AC synchronization. This module is software configured to create a Schmitt trigger. The analog signal enters the comparator's input from P0[7] and is digitally output. The Schmitt trigger output is connected to *Comparator Bus 1*, which generates an interrupt signal upon zero crossing detection.

When using the phase power control method, a PWM is needed. With its help, the AC mains' frequency is calculated during device initialization and load commutation is determined.

The UART occupies the *DCB02* and *DCB03* blocks. It operates at a baud rate of 115200. The UART utilizes the 16-byte command buffer to implement the communication interface. The receiver's input is on P1[2], and the transmitter output is on P0[1]. The module is clocked from frequency source VC3 equal to *SysClk/26.*

Firmware Implementation

The regulator firmware consists of modules based in separate source files. Each module executes its own functions. See the following:

- Initialization procedure and main regulation cycle. (***main.c***).
- PID-controller subroutine (***regulator.c***).
- Synchronization module presented in the interrupt service routine (ISR) (***sync_int.c***).
- Interpolation and searching subroutines in sensor tables (***tc_get_t.c***).
- Terminal interface, which provides communication using EIA-232 (RS-232) interface (**terminal.c**).

Each module is discussed in more detail below.

main.c

This file is the main connection/link of the firmware. Its flowchart is shown in Figure 9. It executes two main functions: device initialization and regulation. During the initialization process, when all of the PSoC modules start, jumpers determine which current sensor type and power control method is used. Current settings are saved in variables *SENSOR_TYPE* and *REG_METHOD*, which are passed in subroutines *Set_Sensor()* and *Set_Regulation_Mode()*.

When the sensor type is assigned, the pointer to the corresponding sensor's table is initialized, and the instrumentation amplifier gain is set.

For the phase method, the AC main half-cycle is determined using the *Calculate_Circuit_Frequency()* subroutine and the phase graduation step is calculated. The half-cycle is saved in variable *Current_Network_Period*, and the phase graduation step in variable *phase_step*. To calculate the AC main half-cycle, the interrupt service routine is used and its working principles are implemented. After zero crossing detection, the PWM is started and it works towards the next zero crossing condition. After that, it is stopped, and its current counter value is read and considered as a half-cycle.

When using the numeral-impulse power control method, it is necessary to disconnect the PWM from the GPIO. This is done by modifying the *PRT1GS* register.

After the initialization procedure is complete, the regulation cycle begins. This cycle (besides the main tasks of reading and processing sensor signals and sending their results to the regulator) executes the sensor connection control function, *is_sensor_connected()*. This control is done once every 32 cycles. If a sensor break is detected, the heater is disconnected and a red LED turns on. Also, during each cycle, the presence of a UART command is verified. If a command is detected, then its processing subroutine *terminal()* is called. A visual description of this cycle is shown in Figure 9.

tc_get_t.c

After the sensor signals are read, the corresponding object's temperature must be determined. The sensor (RTD and thermocouple) characteristic tables are stored in the PSoC Flash memory. In this project, tables are built with 1ºC-steps in the temperature range [-100 to 400]ºC. The tables for the thermocouple and the RTD are based in the *tc_data.h* and *rtd_data.h* files. The last table's element index is assigned in macro definition *MEASURE_RANGE* in the *includes.h* file.

To obtain more precise regulation, temperature interpolation is done to a quarter degree. The user can modify the interpolation step by changing the macro definition *N* in the *includes.h* file. Note that *N* is a power of two. For example, if *N=2*, then the interpolation step equals $1/2^N$ = 1/4ºC. It enables the PID-controller to react from a small temperature deviation and restore the reference temperature.

Interpolation assumes that the characteristic interval of the sensor is linear.

When the temperature is measured using a thermocouple and after the signals are digitized, the ADC code clearly corresponds to the defined temperature. To minimize presser time, the table is built from ADC code, but does not use corresponding voltages. All of the table's calculations can be implemented using any programming language. The table's file for this project *tc_data.h* is generated automatically. The value of table's element index was calculated according to the following equation:

$$ADC\_code = U_{TC} \bullet 93.024 \bullet 2^{13}/2.6 \quad \textbf{(5)}$$

In this equation, voltage is expressed in volts. 93.024 is the gain of the instrumentation amplifier. (Gain can be an additional source of calculation error, as it can float slightly.) 13 is the digit capacity of the ADC and 2.6 volts is the measurement range of the ADC.

The ADC code is calculated and its corresponding temperature value in the table is found. The accordance between temperature and the table's element index is calculated using the following equation:

$$T = Element\_No - T\_Zero\_Index \quad \textbf{(6)}$$

Element_No is the table's element index and *T_Zero_Index* is the table's element index that corresponds to 0ºC.

This parameter is assigned in the *includes.h* file.

The elements in the RTD table are resistance values corresponding to temperatures. To increase the quality of interpolation, each element is multiplied by 10. The following equation is used:

$$R_T = R_0(1 + A \bullet T + B \bullet T^2 - 100 \bullet C \bullet T^3 + C \bullet T^4) \bullet 10 \quad \textbf{(7)}$$

R_0 is the sensor resistance at 0ºC. A, B, and C are the sensor characteristic coefficients and 10 is the interpolation enhancing coefficient.

Searching the Algorithm

Beginning with the physical properties of the heating/cooling process, an object's temperature cannot change in spurts. In other words, temperature change is a continuous function through time. Searching is carried out relative to last fixed temperature.

The previous and next temperatures are found using interpolation. The direction along the table is determined to reach the current temperature. A flowchart for this algorithm is shown in Figure 7. This flowchart uses the following symbols:

- *ADC* – current ADC code, for which the temperature is looking.
- *curr* – determines the result of previous search at the start of search procedure.
- *prev* – stores previous values to *curr.*
- *next* – stores value following *curr*.

The algorithm executes cyclically. The absolute difference between the desired quantity *ADC* and the current value from table *curr* is not minimal.

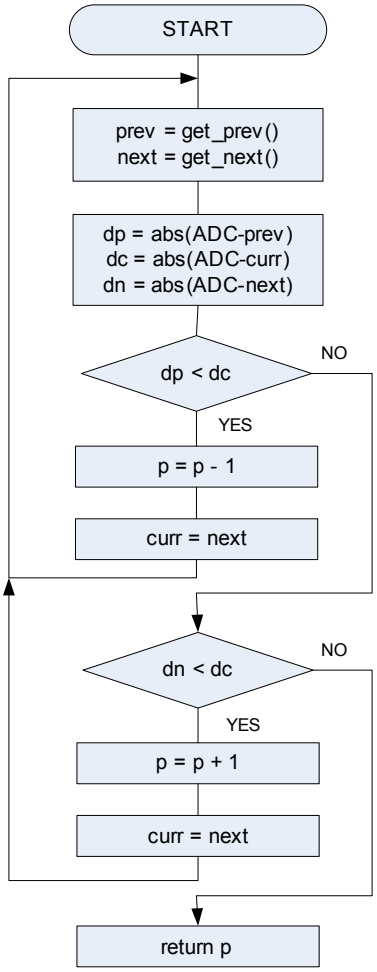


Figure 7. Search Algorithm

regulator.c

After interpolating and searching the tables, the subroutine of the PID-controller is called in the manner by which the current temperature was determined. The subroutine's parameter is the object's temperature.

Upon the output of the regulator, the digital code is set based on the control signals.

The regulation algorithm is implemented according to the following equation:

$$u(k) = K_p \bullet e(k) + K_I \bullet \sum_{i=0}^{k} e(i) + K_D\left[e(k) - e(k-1)\right] \quad \textbf{(8)}$$

Differentiation amplifies noise. To reduce the influence of differentiation, derivatives are calculated using two data points. To calculate the derivative in this project, the numerical method of left fifth differences was utilized:

$$\frac{dy}{dt_{t=t0}} \approx \frac{1}{12h}(25y_0 - 48y_{-1} + 36y_{-2} - 16y_{-3} + 3y_{-4}) \quad \textbf{(9)}$$

This method reduces the influence of noise on the final result.

The flowchart of the PID-controller algorithm is presented in Figure 10. The software control implementation must limit each element to a fixed range.

Following is a description of each regulator parameter and corresponding variable in the program:

- *Tref* – reference temperature, which is kept up by the regulator.
- *Prop_Gain* – proportional gain parameter (K_P).
- *Intl_Gain* – integral gain parameter (K_I).
- *Deri_Gain* – derivative gain parameter (K_D).
- *PROP_REG_LIMIT* – working range of proportional element.
- *INTL_REG_LIMIT* – working range of integral element.
- *DERI_REG_LIMIT* – working range of derivative element.
- *REG_RANGE* – working range of regulator (set in degrees).
- *PWM_RESOLUTION* – determines the number of regulation steps. For the phase method, this is the amount of segments on which a half-cycle is divided. For the numeral-impulse method, this is the regulation period in half-cycles.

All of these parameters are based in the 255th block of Flash memory starting at address 0x3FC0. Parameters can be changed in-system using the terminal program.

- *SCALE_FACTOR* – this is a divider by which the calculated control value is limited in the range *[0..PWM_Resolution]*. This parameter is used in the following equation:

$$PWMcount = \frac{PropTerm + IntlTerm + DenTerm}{SCALE\_FACTOR} \quad \textbf{(10)}$$

The SCALE_FACTOR parameter is set in firmware and cannot be modified via the UART. This should be noted when tuning the PID-controller.

The output controller value is determined by the value of *PWMcount* and is limited in range by *[0..PWM_Resolution]*.

sync_int.c

The network synchronization subroutine is implemented as an ISR from the comparator bus. Figure 8 shows its flow.

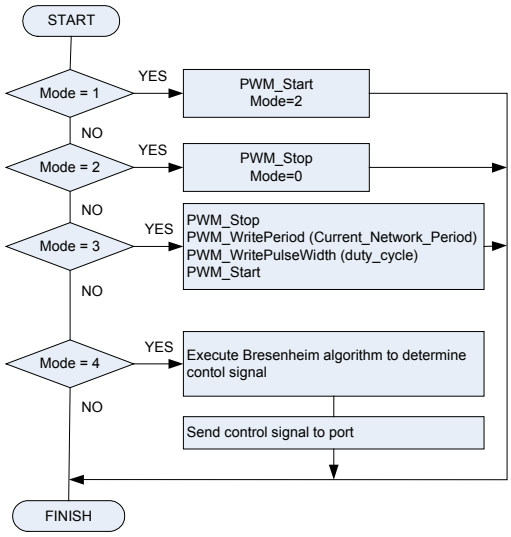


Figure 8. Synchronization ISR

This ISR is implemented in the form of a finite state machine. That is, it remembers its previous state. This feature is used by Bresenham's algorithm to generate control signals during periodic AC main half-cycles.

As can be seen in Figure 8, the subroutine works in four modes, each determined by the global variable *mode*. Modes 1 and 2 are used to determine the AC main half-cycle. In Mode 1 (after zero crossing detection), the PWM starts and Mode 2 switches on automatically. In Mode 2, the PSoC device waits for the next zero crossing condition.

When this condition becomes true, the PWM is stopped and the variable *mode* is set to 0 (this is the signal for the subroutine *Calculate_Circuit_Frequency()* to finish half-cycle measuring). Then, in the main program, the current PWM counter is read and treated as half-cycle.

Mode 3 is used by the phase method to form control signals. This subroutine branch starts the PWM with a given duty cycle. In Mode 4, the Bresenham's algorithm, which is utilized by the numeral-impulse power control method, is implemented.

terminal.c

To make in-system regulator tuning and configuration easier, a UART is included in its structure. The command language that enables communication with the PSoC device by means of standard terminal programs (e.g., *HyperTerminal*) is based on the UART. The COM-port is configured as follows:

- Baud Rate (bit/sec): 115200
- Data Bits: 8
- Parity: None
- Stop Bits: 1

Using the implemented commands, regulator parameters can be edited and saved to Flash. Also, there are commands for receiving current status information. The full list of supported commands is presented in Table 1.

The carriage return symbol is used (ASCII code 0x0D) as a command terminator. Therefore, after typing a command, it is necessary to press the [**Enter**] key. The UART uses [**Space Bar**] (ASCII code 0x20) as a parameter delimiter in the command. All commands are to be typed using capital letters.

Note that after the setting command is entered, and in the case of its successful completion, the UART sends the answer "*OK!*" If an inadmissible command is entered, the answer is "*Unknown!*"

Terminal Program

To send commands automatically to the PSoC and to create a user-friendly interface, a program was developed using Win32 Application and C++Builder. The program allows the user to visually parameterize, and also enables real-time graph building. Graphs are constructed simultaneously. The first graph shows the dependence of temperature on time. In the second graph, the values of proportional integral and derivative constituents are traced. These tools make it easier for the user to tune the regulator. Figure 11 shows the appearance of the main application.

Interface Elements

Once the program is started, communication with the thermoregulator must be established. Select the COM-port to which the PSoC device is connected. Communication is then set by pressing the *CONNECT with PSoC* button. After connection is established, the user can tune the regulator.

Controller parameters are entered in the text fields. To save the typed parameters to Flash, press the *SAVE REGULATOR PARAMETERS IN PSoC'S FLASH* button. The terminal program does not correct entered data. It merely sends the data to the PSoC, which translates data strings into a digital representation. If invalid symbols are detected, this parameter is assigned a zero. The data is returned to the terminal program and shown in the corresponding text fields. Note that writing to Flash works correctly if the device is not in debug mode. Using the PSoC ICE, it takes about 150 ms to write one Flash block, but on an actual chip, this time is about 10 ms.

Use the following buttons to construct the graph:

- *START construction of graph* – start graph constriction.
- *CLEAR graphs* – clear graph forms.
- *STOP construction of graph* – stop graph construction.

To show the graphs on the screen, check *Show Temperature Graph* and *Show PID Constituents Graph*.

As graphs are constructed in real-time, note that the regulator status is received with the frequency of 5 Hz. Figure 12 shows examples of graph forms.

Table 1. List of Terminal Commands

Command Format	Description

`GET T`	To get object's temperature (integer and fraction).
	For example:
	`GET T`
	`70 3`
	Current temperature equals 70 + ¾ = 70.75ºC (if interpolation step is ¼).

`GET P`	Get current proportional element (*PropTerm*).
`GET I`	Get current integral element (*IntlTerm*).
`GET D`	Get current derivative element (*IntlTerm*).

`GET`	Get current temperature, proportional, integral and derivative elements. For example:
	`GET`
	`70 3 1000 -500 300`
	70 3 – temperature
	1000 – proportional element
	-500 – integral element
	300 – derivative element

`PG <NUMBER>`	Set proportional gain (*PropGain*).
	For example:
	`PG 4000`
	`OK!`

`IG <NUMBER>`	Set integral gain (*IntlGain*).
`DG <NUMBER>`	Set derivative gain (*DeriGain*).
`PL <NUMBER>`	Set limitation of proportional element (*PROP_REG_LIMIT*).
`IL <NUMBER>`	Set limitation of integral element (*INTL_REG_LIMIT*).
`DL <NUMBER>`	Set limitation of derivative element (*DERI_REG_LIMIT*).
`RR <NUMBER>`	Set regulation range in degrees (*REG_RANGE*).
`PWM_RES <NUMBER>`	Set regulation step quantity (*PWM_RESOLUTION*).

`CONNECT`	Set connection with PSoC. For example:
	`CONNECT`
	`OK!`

`SET_T <NUMBER>`	Set reference temperature (*Tref*).

`GET_ALL_PARAMS`	Get all parameters. For example:
	`GET_ALL_PARAMS`
	`70 250 3 0 7000 20000 8000 15 10 2`
	70 – reference temperature (*Tref*)
	250 – proportional gain (*PropGain*)
	3 – integral gain (*IntlGain*)
	0 – derivative gain (*DeriGain*)
	7000 – limitation of proportional element (*PROP_REG_LIMIT*)
	20000 – limitation of integral element (*INTL_REG_LIMIT*)
	8000 – limitation of derivative element (*DERI_REG_LIMIT*)
	15 – regulation range (*REG_RANGE*)
	10 – regulation step quantity (*PWM_RESOLUTION*)
	2 – interpolation coefficient (*N*). Interpolation step = $1/2^{N}$ºC

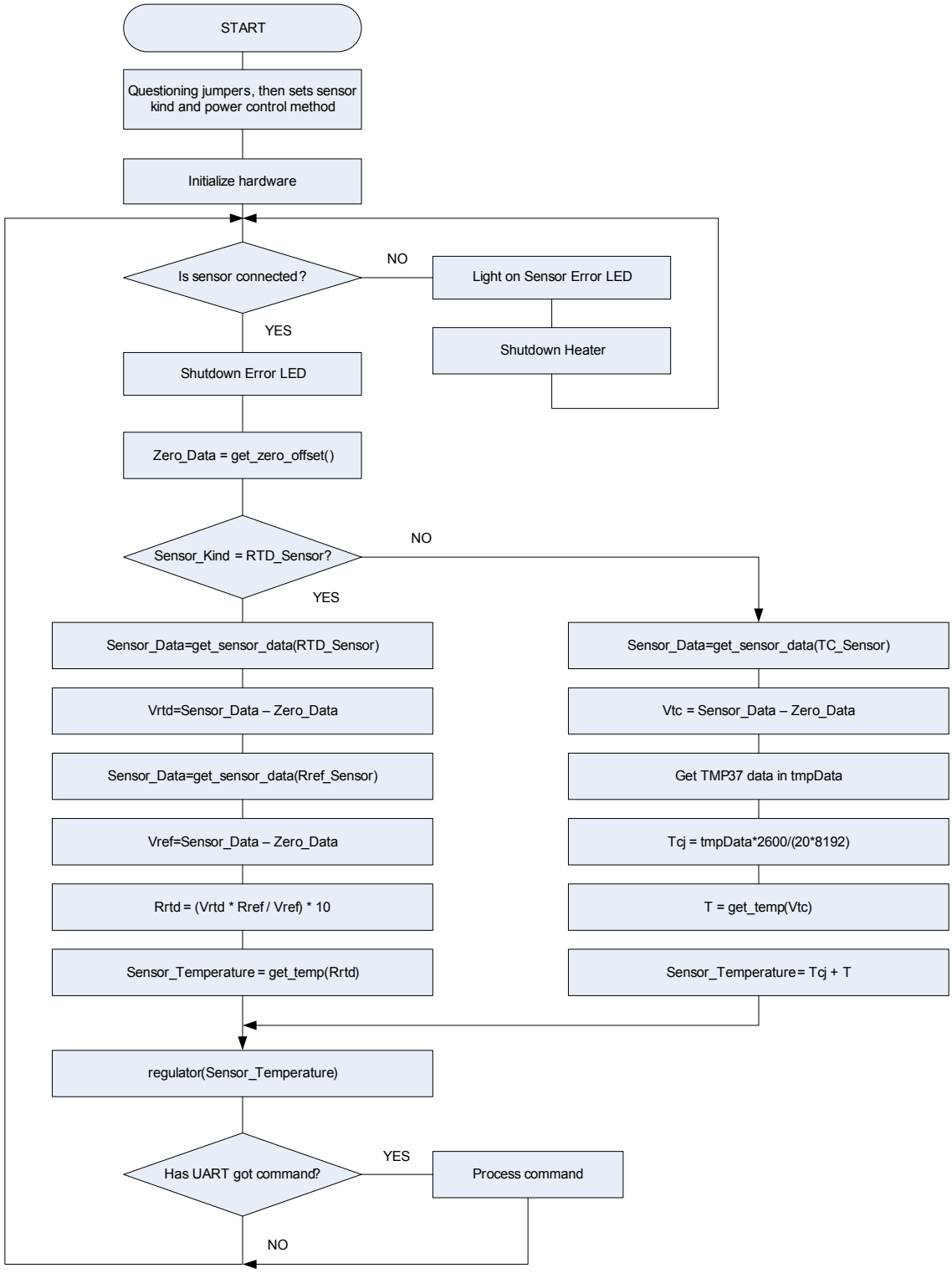


Figure 9. Main Loop

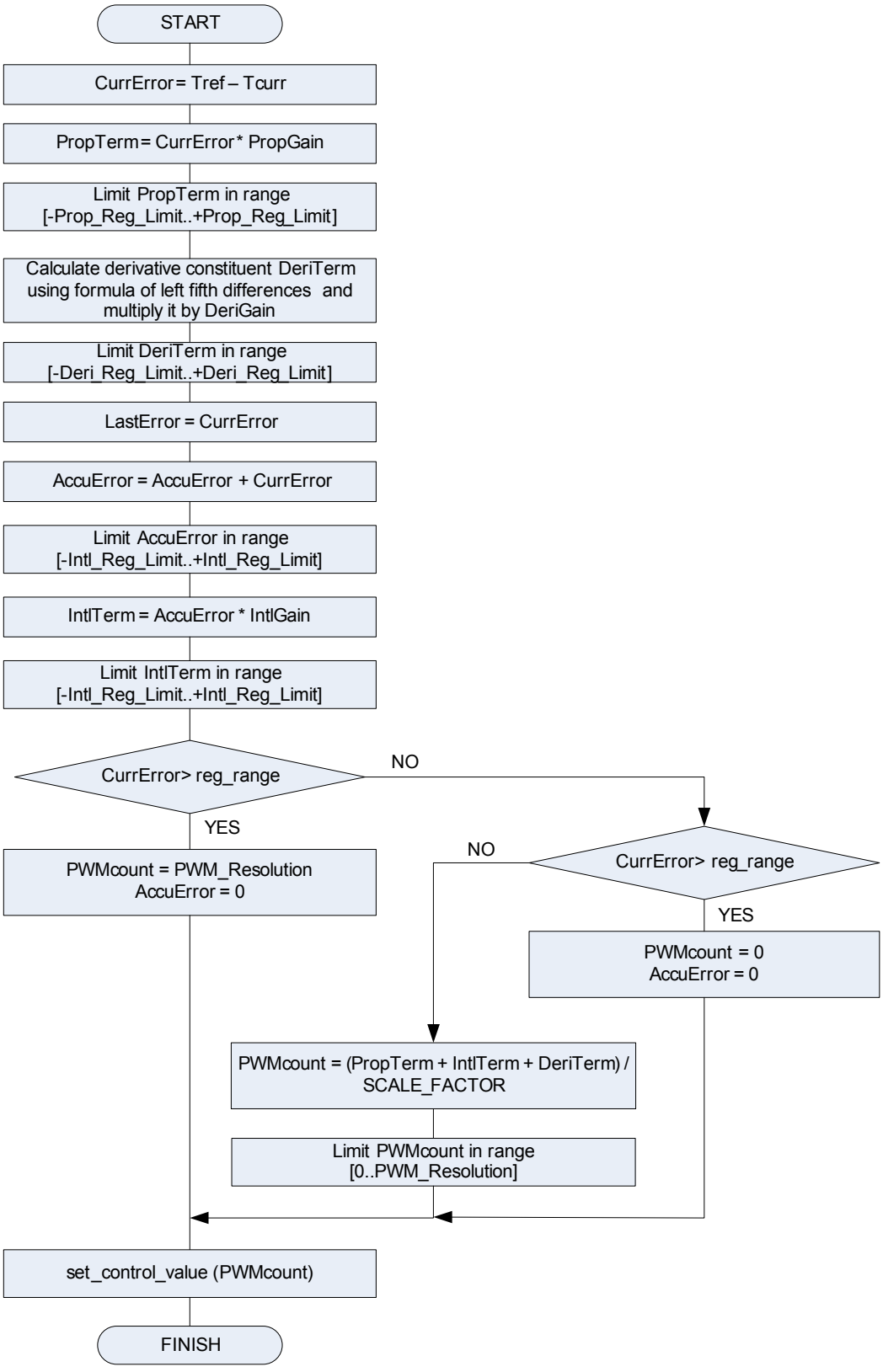


Figure 10. PID-Controller Flowchart

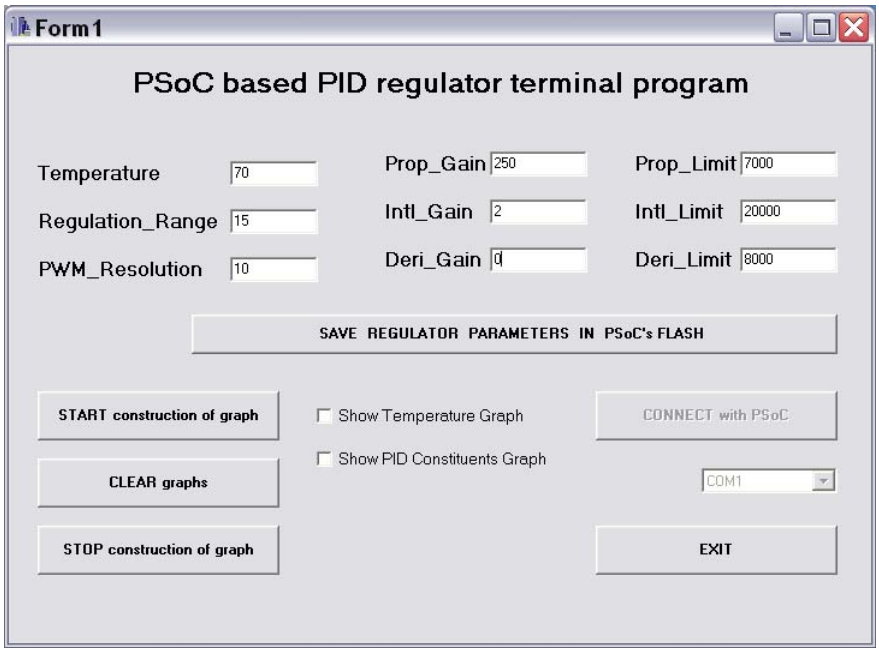


Figure 11. Terminal Program's Main Window

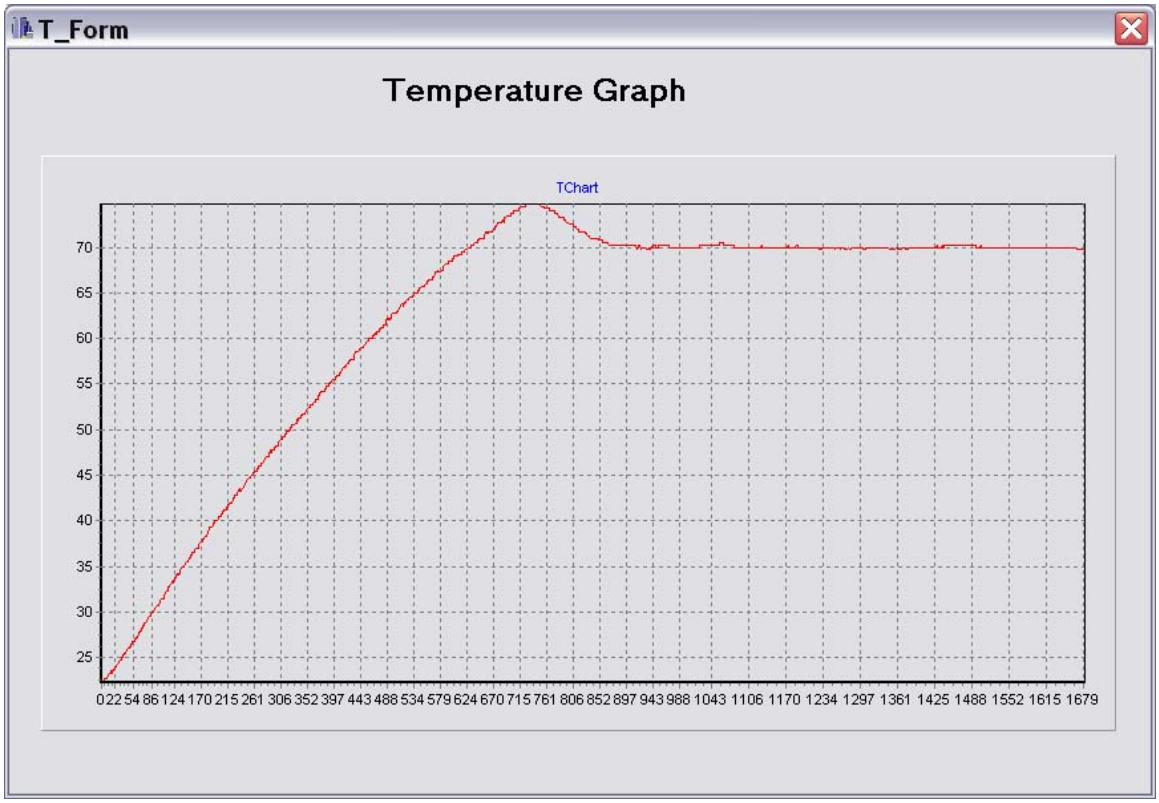
Conclusion

The PID-thermoregulator described in this Application Note involves different variations of temperature control systems. The following options are available: load type, temperature sensor type (thermocouple or RTD), and power control method (phase or numeral-impulse). These options simplify thermoregulator design for specific applications. Among the advantages of the presented thermoregulator is the UART's precise tuning capability. The associated terminal program allows users to visually observe the regulation process and easily adjust as needed.

To adapt this thermoregulator to specific technology conditions, only minimal changes to software are necessary. Most of the changes are due to the type of sensor used (thermocouple or RTD). In this case, the sensor's characteristic table must be generated and inserted in the regulator project along with minute modifications to the macro definitions in the *includes.h* file. Then the PID-parameter's tuning can be performed by means of the terminal program.

About the Author

Name: Andrew Smetana
Title: Undergraduate Student
Education: Ukraine National University "Lvov Poltechnic." Computer Engineering Chair.
Contact: andrew_smetana@ukr.net



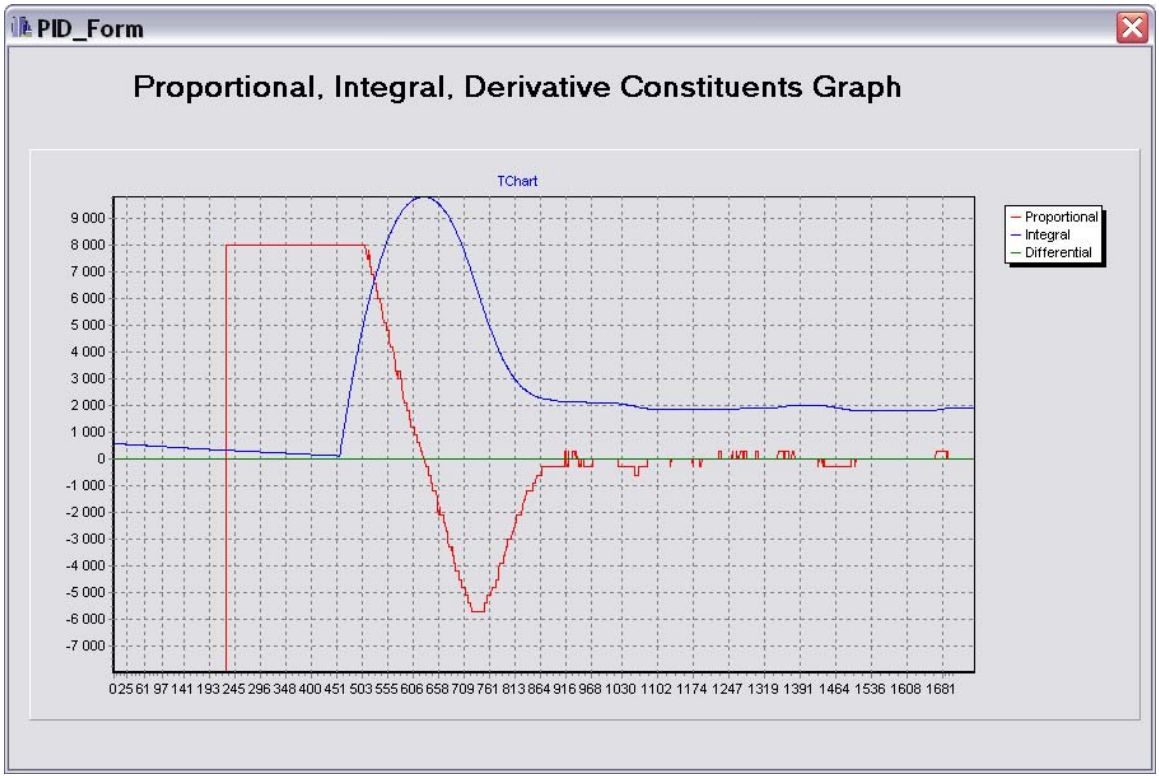


Figure 12. Graph Construction using Terminal Program

Cypress Semiconductor
2700 162nd Street SW, Building D
Lynnwood, WA 98037
Phone: 800.669.0557
Fax: 425.787.4641

http://www.cypress.com/ / http://www.cypress.com/support/mysupport.cfm