Self-tuning Digital PID Controller Implemented on 8-bit Freescale Microcontroller

Petr Dostálek, Jan Dolinay, Vladimír Vašek and Libor Pekař

***Abstract*—** In this paper implementation of self-tuning digital PID controller on 8-bit Freescale MC68HC908GB60 microcontroller which is intended for general purpose applications is described. Controller firmware was created on development board M68EVB908GB60 by Axiom manufacturing providing number of useful peripherals for comfortable application development. Controlled process is identified using modified recursive least squares method with adaptive directional forgetting resulting in δ-model representation of controlled plant. This approach results in better numerical stability of identification process and allows lower sampling periods. Parameters of PSD controller are designed by pole placement method on the basis of estimated plant coefficients. Controller firmware was created in Freescale CodeWarrior integrated development environment in C and assembly language. Software works under real-time operating system RTMON for HCS08 which was created on our department. Controller was verified by temperature controlling of two different heat plant models.

***Keywords*—** Delta models, Freescale MC68HC908GB60, microcontroller, pole placement, self-tuning control.

I. Introduction

PRESENT-day very rapid progress in electronics and computer science influences all areas of human activities. Production technology improvements of new microcontrollers lead to their miniaturization, increased central processor unit performance, decreased power consumption and price. Thus modern 8-bit one-chip microcontrollers have enough computing power not only for simple control loops consisting of fixed parameters controllers like PS or PSD. They are able to handle tasks from the origin of modern control methods such as adaptive control. Due to some limitations, mainly in main memory capacity, microcontrollers cannot substitute powerful industrial PCs or special programmable logic controllers, which can work with number of control loops simultaneously. However, area of microcontroller usage is a bit different – in embedded systems that is in systems where are laid stress on low price, compact dimensions, low power consumption, high reliability and immunity against environmental influences and other specific requirements.

This work presents implementation of self-tuning digital PID controller on a member of wide family of 8-bit Freescale HCS08 microcontrollers. Concretely was chosen general-purpose 8-bit Freescale MC68HC908GB60 microcontroller which is a part of development board M68EVB908GB60 by Axiom Manufacturing. First part of the paper describes implemented algorithms for process identification and PSD controller design using pole placement method. Next two chapters deal with hardware overview of selected microcontroller including evaluation board basic properties followed by software implementation. Last part of the paper is focused on experimental verification of designed controller.

The work was performed with financial support of research project MSM7088352102. This support is very gratefully acknowledged.

II. Implemented algorithms

A. Process identification

For process identification was used recursive least square algorithm with adaptive directional forgetting. In order to achieve lower sampling time periods and mainly better numerical stability of identification process forward δ-model was chosen:

$$\delta = \frac{z-1}{T_0}. \tag{1}$$

Identified system is described by second order transfer function in δ-representation (2).

$$G_s(\delta) = \frac{\beta_1\delta + \beta_2}{\delta^2 + \alpha_1\delta + \alpha_2} \tag{2}$$

ARX regression model expressed in compact vector form is described by equation [1]:

$$y(k) = \Theta^T(k)\phi(k-1) + n_s(k) \tag{3}$$

where $\Theta^T(k)$ is vector of parameters of identified system and $\phi(k-1)$ is regression vector.

$$\Theta^T(k) = [a_1, a_2, ..., a_{na}, b_1, b_2, ..., b_{nb}] \tag{4}$$

$$\begin{aligned}\phi^T(k-1) = [&-y(k-1), -y(k-2), ..., -y(k-na),\\ &u(k-1), u(k-2), ..., u(k-nb)]\end{aligned} \tag{5}$$

Vector of parameter estimations is updated using:

$$\hat{\Theta}(k) = \hat{\Theta}(k-1) + \frac{C(k-1)\phi(k-1)}{1+\xi(k)}\hat{e}(k-1) \tag{6}$$

where e(k) is prediction error and C(k) is covariant matrix [1]:

$$\hat{e}(k) = y(k) - \hat{\Theta}^T(k)\phi(k-1) \tag{7}$$

$$C(k) = C(k-1) - \frac{C(k-1)\phi(k-1)\phi^T(k-1)C(k-1)}{\varepsilon^{-1}(k)+\xi(k)} \tag{8}$$

where ε(k) and ξ(k) are defined as [1]:

$$\varepsilon(k) = \varphi(k) - \frac{1-\varphi(k)}{\xi(k)}, \tag{9}$$

$$\xi(k) = \phi^T(k-1)C(k-1)\phi(k-1). \tag{10}$$

The directional forgetting factor φ(k) is calculated using equation [1]:

$$\begin{aligned}\varphi(k)^{-1} &= 1 + (1+\rho)\left[\ln\left(1+\xi(k-1)\right)\right] \\ &+ \left[\frac{(\nu(k-1)+1)\eta(k-1)}{1+\xi(k-1)+\eta(k-1)} - 1\right]\frac{\xi(k-1)}{1+\xi(k-1)}\end{aligned} \tag{11}$$

where η(k), ν(k) and λ(k) are defined as [1]:

$$\eta(k) = \frac{\hat{e}^2(k)}{\lambda(k)}, \tag{12}$$

$$\nu(k) = \varphi(k)\left[(\nu(k-1)+1\right], \tag{13}$$

$$\lambda(k) = \varphi(k)\left[\lambda(k-1) + \frac{\hat{e}^2(k-1)}{1+\xi(k-1)}\right]. \tag{14}$$

For parameters identification of transfer function (2), vector of parameters and vector of data must be modified to the form of (15) and (16) [2].

$$\Theta^T(k) = [\alpha_1, \alpha_2, \beta_1, \beta_2] \tag{15}$$

$$\begin{aligned}\phi^T(k-1) = [&-\frac{y(k-1)-y(k-2)}{T_0}, -y(k-2), \\ &\frac{u(k-1)-u(k-2)}{\mathrm{T}_0}, u(k-2)]\end{aligned} \tag{16}$$

B. Digital PID controller design

The controller based on the placement of poles of a feedback system is designed so that it stabilizes the closed feedback loop whereas the characteristic polynomial has pre-defined poles. For feedback system the synthesis consists in solving the Diophantine equation:

$$AP + BQ = D, \tag{17}$$

where A, B are polynomials of the plant, Q, P are polynomials of the controller and D is characteristic polynomial, which is defined in the following form:

$$D(\delta) = \delta^4 + d_1\delta^3 + d_2\delta^2 + d_3\delta + d_4 \tag{18}$$

The transfer function of the digital PID controller in δ-modification is:

$$G_R(\delta) = \frac{Q(\delta)}{P(\delta)} = \frac{q_0\delta^2 + q_1\delta + q_2}{\delta(\delta+\gamma)}. \tag{19}$$

Solving of Diophantine equation (17) leads to a system of four algebraic equations, which can be written in matrix form [2]:

$$\begin{bmatrix} \beta_1 & 0 & 0 & 1 \\ \beta_2 & \beta_1 & 0 & \alpha_1 \\ 0 & \beta_2 & \beta_1 & \alpha_2 \\ 0 & 0 & \beta_2 & 0 \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \gamma \end{bmatrix} = \begin{bmatrix} d_1-\alpha_1 \\ d_2-\alpha_2 \\ d_3 \\ d_4 \end{bmatrix}. \tag{20}$$

PSD controller output value u(k) is computed using (21).

$$\begin{aligned}u(k) &= q_0[e(k) - 2e(k-1) + e(k-2)] + q_1T_0[e(k-1) - e(k-2)] + \\ &+ q_2T_0^2e(k-2) - \gamma T_0[u(k-1) - u(k-2)] + 2u(k-1) - \\ &- u(k-2)\end{aligned} \tag{21}$$

III. Hardware overview

Self tuning digital PID controller was implemented on development board M68EVB908GB60 by Axiom manufacturing which is based on general purpose 8-bit Freescale microcontroller MC9S08GB60. Board is equipped with number of useful peripherals enabling comfortable development of new applications for this microcontroller.

User application can utilize 4 LED indicators, 4 push buttons, 4 position DIP switch, 2x16 character LCD Module, buzzer and potentiometer. For communication purposes it provides two serial asynchronous communication interfaces RS232 with standard DB9-S connectors. Second RS232 interface can be switched to RS422/485 mode in which all signals are redirected to dedicated 5 pin terminal. In case of need other external peripherals can be connected to MCU port (provides all digital I/O) and ANALOG port (provides analog inputs) incorporating all necessary signals including power supply pins.

Development kit is provided with Freescale binary monitor located in protected area of the internal FLASH memory. Monitor program enables loading and debugging of user program via standard RS232 interface, so there is no need to have specialized BDM adapter. Development board photograph is in Fig.1 [6].

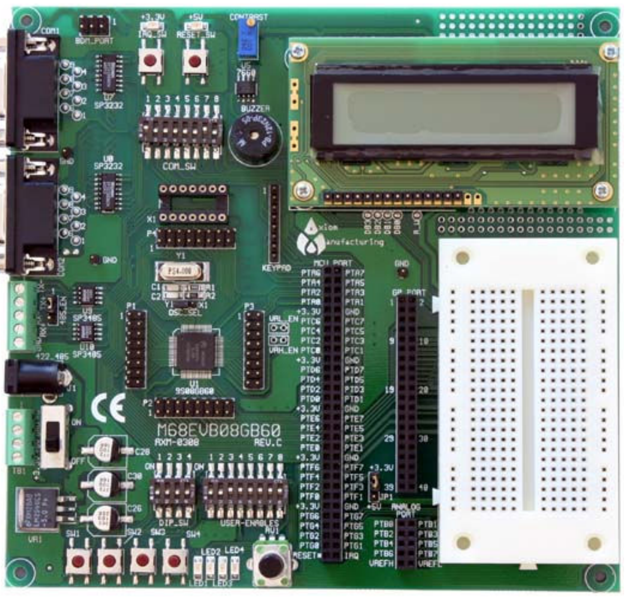


Fig. 1 Development board M68EVB908GB60 [6].

A. Microcontroller MC9S08GB60

The MC9S08GB60 is low-cost, general purpose, high-performance 8-bit flash-based microcontroller with von-Neumann architecture. Central processor unit with enhanced HCS08 core is fully upward compatible with Freescale 68HC05 family. Their architecture is fully optimized for C language compilers.
On the chip are integrated following modules:

- One 3-channel and one 5-channel 16-bit timer/pulse width modulator modules
- Two serial communication interfaces
- Serial peripheral interface
- Inter-integrated circuit bus module
- Internal clock generator module
- 10-bit analog-to-digital converter with 8-channel analog multiplexer
- On chip 64KB FLASH memory with in-circuit programming capability
- 4KB on-chip RAM
- 56 general-purpose I/O pins (16 high-current pins)
- Software selectable pull-ups on ports when used as input
- 8-pin keyboard interrupt module
- Watchdog system
- Low-voltage detection
- Illegal operational code and address detection
- On-chip debug module (DBG) [4]

Central processing unit (CPU) features:

- 40 MHz operation at 3V
- 8-bit accumulator (A)
- 16-bit stack pointer (SP) with new stack manipulation instructions
- 16-bit index register (H:X) with index register instructions
- Memory to memory moves without using the accumulator
- Fast 8-bit by 8-bit multiply and 16-bit by 8-bit divide instructions
- 64 Kbytes program/data memory space [5]

IV. CONTROLLER FIRMWARE DESIGN

Self-tuning digital PID controller internal software is based on real-time operating system RTMON for HC08, which was developed on our department especially for microcontroller-based embedded systems with CPU08 main processor core. So software is formed of RTMON core and individual processes which perform all necessary tasks. Each process activity is controlled by operating system core on the basis of process priority and other information stored in the task descriptor. Structure of the firmware is depicted in the Fig. 2. There are 7 main processes and 1 interrupt handling routine. RTMON core and program processes functions are in detail described in next chapters.

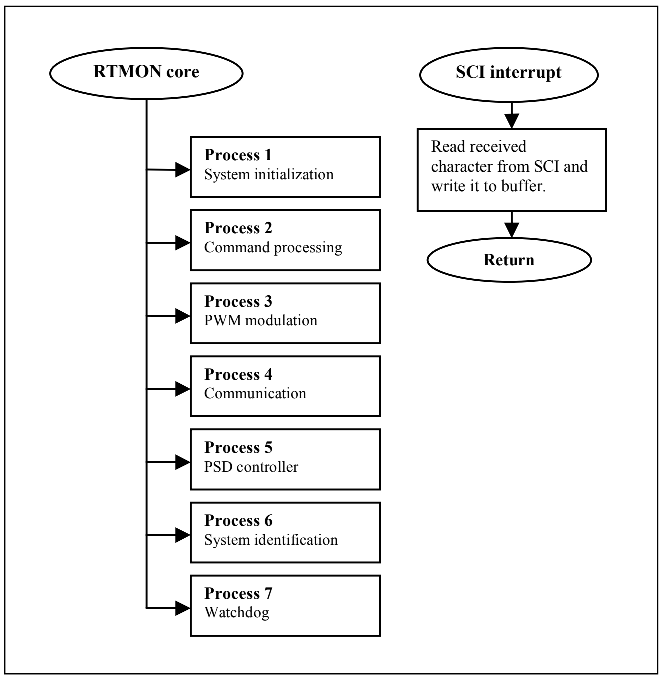


Fig. 2 Controller software structure.

A. Real-time operating system RTMON

RTMON is preemptive multitasking operating system which is simplified to great extend to allow easy use for programmers. It is written in C language with the exception of small platform-specific code written in assembler. The scheduler assigns time slices to processes based on their priority. The priority is integer in the range 1 to 254. Priority 0 is the highest and is reserved for the RTMON initialization process and priority 255 is the lowest and is reserved for the idle process (called dummy in RTMON).

RTMON allows execution of two different types of processes (tasks): normal processes which execute only once (such process typically contains infinite loop) and periodical process which is started automatically by RTMON with given period. These periodical processes are useful for many applications, for example, in discrete controllers which need to periodically sample the input signal and update the outputs.

For the sake of simplicity of both the implementation and usage, several restrictions are applied. First, the RAM memory for processes and their stacks is statically allocated for the maximal number of processes as defined in configuration file. In the user program, it is not possible to use this memory even if there are fewer processes defined. In case more RAM is needed for the user program, the maximum number of tasks and/or stack-pool size can be changed in configuration file and the RTMON library must be rebuild.

The priority of each task must be unique, so that in each moment one task (the one with highest priority) can be selected and executed on the CPU. The scheduler does not support cyclical switching of several processes with the same priority on the CPU in round-robin fashion; it simply chooses the task with highest priority from the list of tasks which are ready to run. Processes can be created on the fly, but it is not possible to free and reuse memory of a process. No more than the maximal number of processes can be created, even if some processes were previously deleted.

These restrictions, however, do not present any big problem for most applications and allow for small kernel code size and ease of use.

There are only two objects (data structures) which RTMON contains: process (task) and queue. The queues are buffers for transferring data between processes. It would more properly be called mailboxes in our implementation as each queue can contain only 1 message. Several queues can be created, each containing a message (data buffer) of certain size. The size can be specified when creating the queue and is limited by the total size of RAM reserved for all buffers of all the queues (queue pool size). Processes can read and write data to the queue and wait for the queue to become empty or to become full. This allows for use of the queue also as a synchronization object (semaphore).

The RTMON uses timer interrupt which occurs at certain period (e.g. 10 ms) to periodically execute the scheduler, which decides which process will run in the next time slice. The timer interrupt routine is implemented in assembly language. It first stores CPU registers onto the stack and then calls RTMON kernel, which is a C function. The kernel then finds the process with highest priority which is in ready-to-run state and switches the context, so that the code of this process is executed after return from the interrupt service routine. If no process is ready to run, then a special dummy process is executed. This dummy process is contained within RTMON code and does nothing.

The following basic operations can be performed with a process in RTMON. Each operation corresponds to a function in the RTMON library which user program can call:

- create process
- start process
- stop process
- delay process
- continue process execution
- abort (delete) process

For queues there are the following functions:

- create queue (specify size)
- write to a queue with/without waiting
- read from a queue with/without waiting

There are also two functions for controlling the RTMON core:

- initialize RTMON
- end RTMON operation

The RTMON system is used as a precompiled library accompanied by a header file. This simplifies the organization of the project and the build process. User enables RTMON usage in his program by including the header file (rtmon.h) in his source and adding the library to his project.

B. *Processes function description*

Process 1 is highest priority process which performs controller hardware initialization after power up or reset. It sets all digital outputs to low state (logic 0), setups serial communications interface to communication speed of 57600 Bd, 8-bit data frame, 1 start bit and 1 stop bit and finally initializes all necessary data structures. Because of its highest priority no other processes can be switched by RTMON core into the “run” state before this process is completely finished. After all initializations are done it suspends itself.

Process 2 performs all tasks related to command interpretation and execution. It waits for complete command string in the receiver buffer which is handled by serial communication interface (SCI) interrupt routine. This interrupt routine is automatically called when SCI receive one character from the higher-level control system (personal computer or programmable logic controller, for example). When command is completely received in the buffer, process will decode it and executes required action.

Process 3 is periodically activated process performing pulse-width modulation (PWM) on all digital output channels when it is demanded. Its priority is set to higher level than process 2 and process 4 because the PWM is time critical function sensitive to accurate timing. Its resolution is 8-bit allows setting 256 different duty cycles at output. Period of the PWM signal is set to value of actual controller sampling period.

Process 4 provides communication via RS232 serial interface with supervisory system. It generates responses to all commands regarding to defined communication protocol include error processing. It has defined lowest priority among of all processes.

Process 5 is periodically activated process with period set to user specified sampling rate performing computation of digital PID controller output value u(k) from control error value e(k).

Process 6 performs controlled system identification by recursive least square algorithm with adaptive directional forgetting described in the chapter 2.A. It is resulting to controlled system parameters α_1, α_2, β_1 and β_2 which are necessary for PSD controller coefficients calculation.

Process 7 periodically checks correct function of all running RTMON processes by polling their status. Controller is stopped and all actuating signals are set to inactive state when problem is detected.

C. Communication protocol

In order to achieve compatibility with many software platforms, universal ASCII-based communication protocol was chosen. Very advantageous is the possibility to send all implemented commands using generic terminal program that is included in most operating systems. Each command can be divided up to five parts depending on function implementation. Communication starts with character "~" then must follow command name with fixed length of two characters (for example "CW" means set new set point value). After it is the first command parameter with length of one character (channel index – reserved for multichannel controller) next character is space followed by second parameter (value) or parameters (values). Command must be terminated by CR, LF sequence. Communication protocol example is depicted in the Fig. 3. Controller supported commands are provided in the TABLE I. First command parameter <chann> must be always set to zero in current SW implementation (only single channel operation is allowed).

TABLE I Commands supported by controller.

Command	Description
~CW<chann> <val>CRLF	Set new set point value in deg. C for channel <chann>
~CT<chann> <val>CRLF	Set controller sampling period in seconds on channel <chann>
~CP<chann> <d1> <d2> <d3> <d4>CRLF	Set characteristic polynomial coefficients on channel <chann>
~CC<chann> <q0> <q1> <q2>CRLF	Set controller coefficients on channel <chann>
~CU<chann> <val>CRLF	Set control signal value in % on channel <chann> (monitor mode)
~AE<chann>CRLF	Enable self-tunning feature for PSD controller on channel <chann>
~AD<chann>CRLF	Disable self-tunning feature for PSD controller on channel <chann>
~MM<chann>CRLF	Change mode to monitoring on channel <chann>
~MC<chann>CRLF	Change mode to "PSD controller" on channel <chann>
~GW<chann>CRLF	Get actual set point value on channel <chann>
~GU<chann>CRLF	Get actual control signal value on channel <chann>
~GY<chann>CRLF	Get actual conrolled value on channel <chann>
~GP<chann>CRLF	Get estimated controlled system parameters on channel <chann>

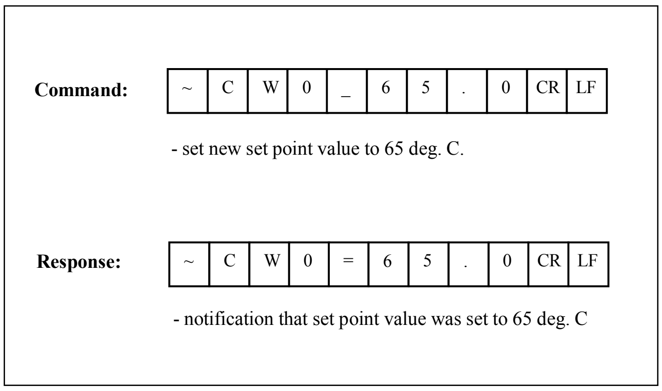

Command:
~ C W 0 _ 6 5 . 0 CR LF
- set new set point value to 65 deg. C.
Response:
~ C W 0 = 6 5 . 0 CR LF
- notification that set point value was set to 65 deg. C

Fig. 3 Communication protocol example.

V. SOFTWARE SUPPORT

Although communication protocol is very simple and easy to understand it is more comfortable in a control application to call functions which can automatically generate commands for the data acquisition device and consequently process its response. Application developer then does not need to know exact communication protocol and do not need to program it. This simplification results in faster program development and reduction of debugging time. There were created support program libraries for Visual C++, Control Web 5 and Matlab 6.5 (and higher versions with serial port object support) software environments.

A. Support libraries for Visual C and Matlab

Created libraries incorporate all functions implemented in the controller including error processing. In order to controller setup and diagnosis simple program utility was created. Program is able to set controller set point value, controller parameters q_0, q_1 and q_2, sampling rate, characteristic polynomial coefficients d_1, d_2, d_3 and d_4, enable / disable adaptation, switch to monitoring or controller mode and other functions.

In monitoring mode can be PSD controller deactivated and then all measured values are saved to text file in format suitable for import to MS Excel software. This feature enables efficient process identification by measuring and evaluating system step response.

Matlab 6.5 support library has implemented same functions with only one difference – in place of device handle is serial port object. Each function is available in separate m-file, so it is very simple to modify them by user.

TABLE II Implemented library functions for C

Function	Description
HANDLE Open_device (const char*)	Opens device connected to specified serial port ("COM1", "COM2",…) and returns device handle.
int Close_device (HANDLE h)	Closes controller device with specified handle.

int Set_W (HANDLE handle, int channel, double value)	Set new set point value in deg. C for specified channel.
int Set_Ts (HANDLE handle, int channel, double value)	Set new sampling rate for specified channel in seconds.
int Set_CharPoly (HANDLE handle, int channel, double *d)	Set characteristic polynomial coefficients for specified channel.
int Set_CntrCoef (HANDLE handle, int channel, double *q)	Set controller coefficients for specified channel.
int Set_U (HANDLE handle, int channel, double value)	Set control signal value in % for specified channel. Applicable in monitoring mode only.
int Set_STFeature (HANDLE handle, int channel, int value);	Enable / disable self-tuning feature of the PSD controller on specified channel.
int Set_CntrMode (HANDLE handle, int channel, int mode);	Change controller mode to “PSD controller” or monitoring for specified channel.
double Get_W (HANDLE handle, int channel)	Get actual set point value for specified channel.
double Get_U (HANDLE handle, int channel)	Get actual control signal value for specified channel.
double Get_Y (HANDLE handle, int channel)	Get actual controlled signal value for specified channel.
int Get_SysParam (HANDLE handle, int channel, double *par)	Get estimated parameters of the controlled system for specified channel.

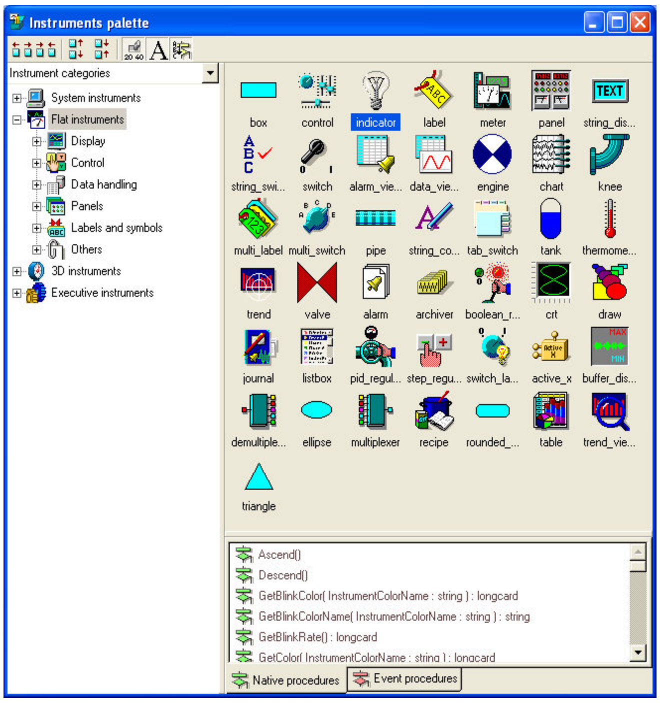
B. Driver support for Control Web

Control Web is Rapid Application Development system developed by Moravian Instruments which is suitable for process visualization and real-time control, HMI applications, technological information systems and other applications.

Application design in Control Web 5 is very fast and comfortable due to integrated development environment, which supports more possibilities how to create new application. The basic idea is to build in graphical editor basic components to the larger block, which can gradually form whole system. Each component can be configured in sheet editor enabling transparent parameter settings. Because resulting code saved in text form and then compiled there is in parallel also text editor available. Each component can be selected from instruments palette organizing them in subcategories – for example system instruments, flat instruments and so on. Selected category can be expanded if it is possible to next sub trees. Expanded category called Flat Instruments can be seen in Fig. 4. Another type of the object in Control Web is data element. Each data element represents location in system memory, which can save value of the measured quantity, for example [10].

Control Web is in standard installation equipped with several drivers which can be divided to two main categories – for demonstration and testing purposes (Virtual Driver, Model driver, Simulation Driver, Simulating Driver) and general drivers for use in real applications (DDE Client Driver, ASCDRV5 driver, TCP/IP driver). Device driver is independent component in a form of dynamically linked library with standardized interface. During development of the Control Web system gradually originated three versions of the interface. Basic interface was defined for Control Web version 3 and must be implemented in every driver. Newer interface version 4 was created with Control Web 2000 and finally newest version 5 was defined for Control Web 5. Back compatibility is guaranteed by implementation of the basic interface in all higher versions of the interface.

Control Web communicates with driver using channels, which must be defined in the driver map file (file with extension dmf). Each channel is defined by number, direction (input, output) and data type (real, boolean, string, and others). Driver configuration is stored in the parametric text file (extension par) containing specific information for correct driver initialization. All parameters can be easily viewed and modified using Driver inspector.



```
Ascend()
Descend()
GetBlinkColor( InstrumentColorName : string ) : longcard
GetBlinkColorName( InstrumentColorName : string ) : string
GetBlinkRate() : longcard
GetColor( InstrumentColorName : string ) : longcard
```

Fig. 4 Control Web 5 Instruments palette.

On the basis of requirements of the driver interface version 3 device driver for digital PID controller was created. It supports 8 input real channels and 10 output real channels. Each channel assignment and its direction are obvious from TABLE III.

TABLE III Control Web driver channel assignment.

Channel	Direction	Description
1	Output	Set point value in deg. C.
2	Output	Controller sampling period in seconds.
3 – 6	Output	Characteristic polynomial coefficients d_1, d_2, d_3, d_4.
7 – 9	Output	Controller coefficients q_0, q_1, q_2.
9	Output	Control signal value in % (applicable in monitoring mode).
10	Output	Enable / disable self-tuning feature of the PSD controller.
11	Input	Actual set point value in deg. C.
12	Input	Actual control signal value in %.
13	Input	Actual controlled signal value in deg. C.
14 – 17	Input	Estimated parameters of the controlled system α_1, α_2, β_1, β_2.
18	Input	Device status information.

VI. Controller Verification and Results

Functionality of the implemented controller was verified by controlling two different real laboratory heat plants using Axiom M68EVB908GB60 development board. Connection with real system was realized via MCU and ANALOG port connectors. Control signal is generated using standard digital output pin with utilization of pulse-width modulation. PWM period was chosen with respect to controlled system dynamics $T_{PWM} = 0.5$ s.

Step responses of the heating plant models 1 and 2 are depicted in the Fig. 5 and Fig. 6. They were measured with control signal change from 40 % to 60 % of its maximum value. Controlled systems were approximated with second order transfer functions (22) for plant 1 and (23) for plant 2. Figures 7 and 9 shows control processes for both heating plants models with utilization of self-tuning PSD controller. Sampling period T_0 of the controller was set to 0.5 s.

$$G_{S1}(s) = \frac{k}{(T_1 s+1)(T_2 s+1)} = \frac{2.3}{(168s+1)(6s+1)} \qquad (22)$$

$$G_{S2}(s) = \frac{k}{(T_1 s+1)(T_2 s+1)} = \frac{2.2}{(121s+1)(7s+1)} \qquad (23)$$

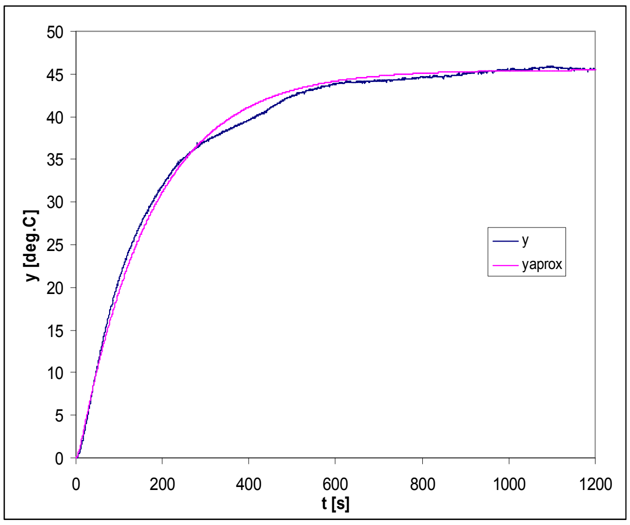


Fig.5 Heating plant model 1 step response.

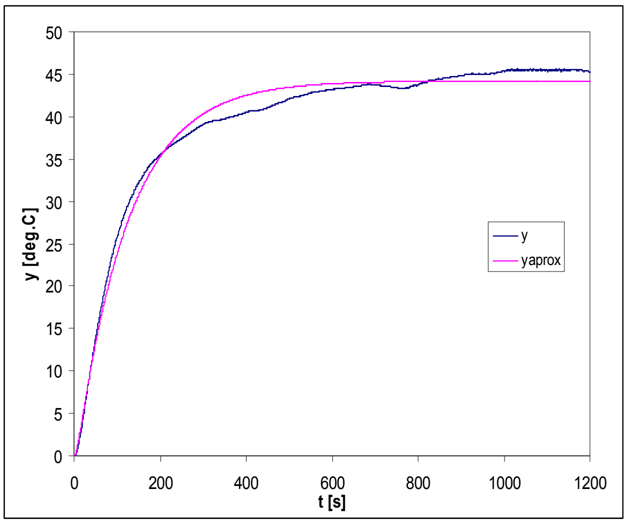


Fig.6 Heating plant model 2 step response.

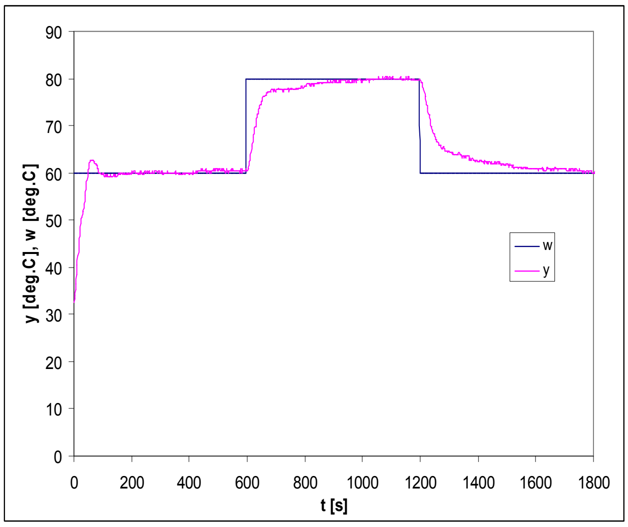


Fig.7 Heating plant model 1 control process.

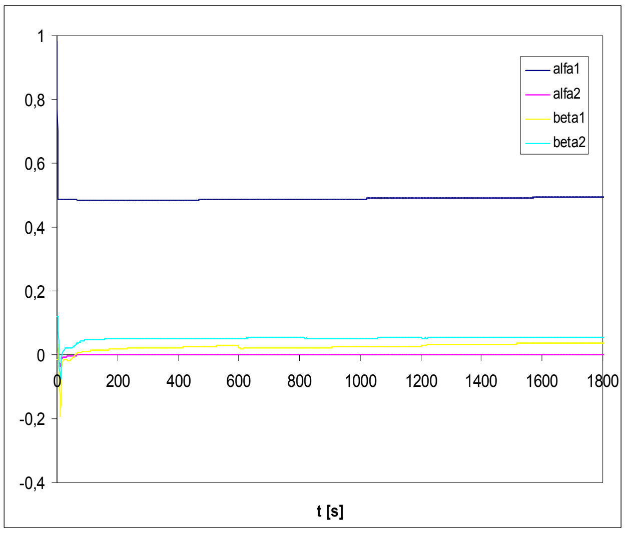


Fig.8 Model 1 control process – parameters estimations.

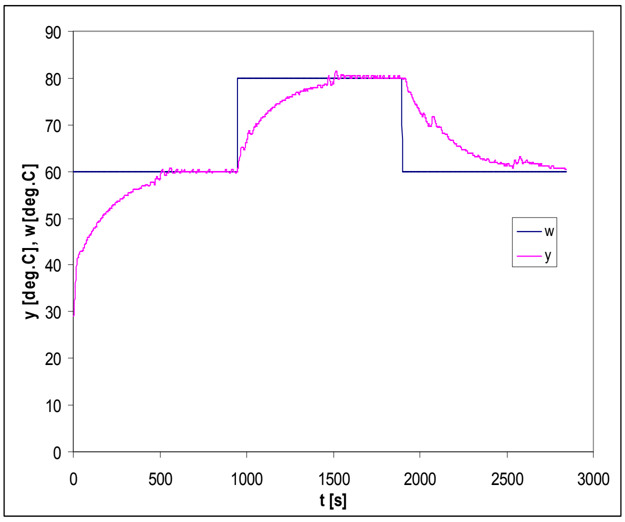


Fig.9 Heating plant model 2 control process.

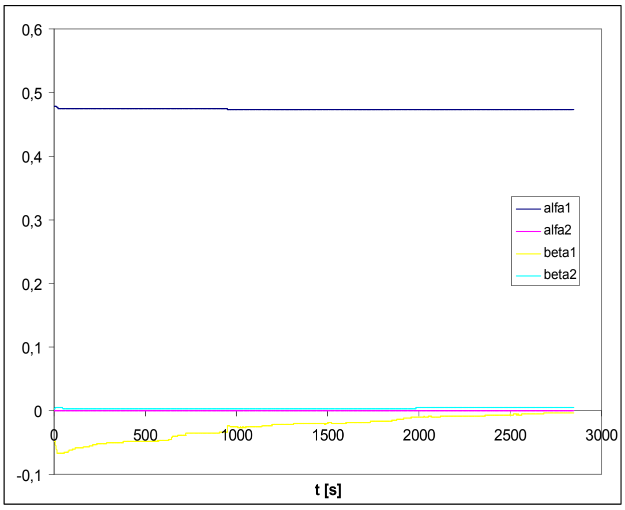


Fig.10 Model 2 control process – parameters estimations.

VII. CONCLUSION

This paper deals with implementation of self-tuning PID controller on low-cost 8-bit one-chip microcontroller Freescale MC68HC908GB60. On the basis of described algorithms and existing program modules for adaptive control for Motorola MC68HC11 microcontrollers developed at our department [3], software in assembly and C language for microcontroller MC68HC908GB60 was created. Controller uses universal ASCII-based communication protocol which can be easily successfully implemented in many control and monitoring software environments. In order to improve development of monitoring and control application for this device a support program libraries for Matlab/Simulink, Visual C++ and Control Web 5 were created.

Controller correct function was verified by controlling real laboratory heating plant models using Axiom M68EVB908GB60 development board, which is based on Freescale MC68HC908GB60 microcontroller. Performed experiments indicate that modern 8-bit microcontrollers have sufficient arithmetic power to handle tasks such as adaptive control. It brings to the area of embedded systems better control quality on variety of systems.

REFERENCES

[1] Bobál, V.; Böhm, J.; Prokop, R. & Fessl, J. (1999). Praktické aspekty samočinně se nastavujících regulátorů, algoritmy a implementace, VUT Brno, ISBN 80-214-1299-2
[2] Bobál, V.; Dostál, P. & Sysel, M. (2000). Delta modification of self tuning pole placement PID controllers, Proceedings of Symposium on System identification SYSID 2000, CD-ROM, University of California, June 2000
[3] Dolinay, J. & Vašek, V. (2003). Program library for Motorola HC11 microcontroller, Proceedings of International Carpathian Control Conference ICCC'2003, pp. 467-470, ISBN 80-7099-509-2, High Tatras, Grandhotel Praha, Slovak Republic, May 2003
[4] Freescale semiconductor. MC9S08GB60A Data Sheet [online]. 2008. Freescale Semiconductor, Inc. : Freescale Semiconductor Literature Distribution Center, 2008 [cit. 2010-02-21]. Available on WWW: <http:\\www.freescale. com>.
[5] Freescale semiconductor. CPU08 Central Processor Unit [online]. 2001. Freescale Semiconductor, Inc.: Freescale Semiconductor Literature Distribution Center, 2001 [cit. 2010-02-23]. Available on WWW: <http:\\www. freescale.com>.
[6] Axiom Manufacturing. M68EVB908GB60 Development Board for Freescale MC9S08GB60 [online]. 2006. Axiom Manufacturing, 2006. [cit. 2010-02-22]. Available on WWW: <http:\\www.axman.com>.
[7] Vasek V.; Dostalek P.; Janacova D.; Kolomaznik K.; Zalesak M., Applied Informatics in Automatic Control Education, In Proceedings of the 7th WSEAS International Conference on Applied Informatics and Communications, Athens, Greece, August 24-26, 2007.
[8] Dolinay J.; Dostalek P.; Vasek V.; Kolomaznik K.; Janacova D., New Embedded Control System for Enzymatic Hydrolysis, In Proceedings of The 8th WSEAS International Conference on Applied Informatics and Communications, Rhodes Island, Greece, August 20-22 2008.
[9] Dolinay, J.; Vašek, V., Utilization of Motorola HC11 in Process Control. Proceedings of the 6th international scientific - technical conference Process Control 2004, ISBN: 80-7194-662-1.
[10] Moravian Instruments, Control Web 5 software documentation, Moravian Instruments, Inc., Zlín, 2005.