1 Изворен код на програмата

Затоа што најголем дел од кодот е автоматски генериран од Processor Expert додатокот, во продолжение ќе биде прикажан кодот само на фајловите кои се изработени или корегирани за потреби на апликацијата опишана во овој труд.

1.1 DSP_Seminarska.c

```
/** ###################################################################
**     Filename  : DSP_Seminarska.C
**     Project   : DSP_Seminarska
**     Processor : 56F8322
**     Version   : Driver 01.13
**     Compiler  : Metrowerks DSP C Compiler
**     Date/Time : 07.08.2010, 13:53
**     Abstract  :
**         Main module.
**         This module contains user's application code.
**     Settings  :
**     Contents  :
**         No public methods
**
**     (c) Copyright UNIS, a.s. 1997-2008
**     UNIS, a.s.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
** ###################################################################*/
/* MODULE DSP_Seminarska */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "AD1.h"
#include "TMR1.h"
#include "PWMC1.h"
#include "TI1.h"
#include "AS1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

#include "printf.h"
#include "custom_types.h"
#include "configuration_file.h"

/* Internal Variables */

u16 Kp = 1;
u16 Ki = 0;
u16 Kd = 0;

u16 DesiredSpeed=1000;

u8 ExecuteCommand = 0;
u8 EditMode = 0;

u8 PID_Enable = 1;
u8 PID_Restart = 0;

/* External Variables */
extern unsigned int NewAD;
extern unsigned int AdcSample;
extern u16 AdcFilterBuf[];

extern u8 RxBuff[];
extern u8 RxNrChars;


/* Prototypes */
void __putc_b(void *p, char c);
void CommandParser(void);
void PID_Control (void);


/* F U N C T I O N S */
```

```
/*
 * Helper function used by tiny_printf. This function is called when a character should be sent
 * to the output stream. In this application the output goes to UART0.
 */
void __putc_b(void *p, char c)
{
  while (ERR_OK != AS1_SendChar((AS1_TComData)c))
  {
  };
  (void)p;
}

/*
 * Function used to parse the incoming commands from the UART0 channel. The commands are used to change values of PID
 * control or to enable/disable PID algorithm.
 * Entering the commands begin by pressing [Enter] and then entering the commands. When command is entered it is
executed
 * by pressing [Enter]
 * --------
 * Supported commands:
 * Kp [value] - update Kp coefficient in PID
 * Ki [value] - update Ki coefficient in PID
 * Kd [value] - update Kd coefficient in PID
 *
 * Pp [value] - disable PID control and set fixed PWM value. The value entered should be devided with 10 to reflect
actual value.
 *              Therefore if desired PWM is 42.3 the command will be "Pp 423".
 * Pd          - enable PID control.
 * Ds [value] - enter new desired speed of the motor.
 * --------
 */
void CommandParser(void)
{
  u8 temp = RxNrChars;
  u16 Value = 0;
  u16 Base = 1;

  while ( temp > 3 )
  {
        Value = Value + (Base * (RxBuff[temp-1]-'0'));
        Base = Base * 10;
        temp--;
  }
  if ( (RxBuff[0] == 'K') || (RxBuff[0] == 'k') )
  {
    switch (RxBuff[1])
        {
          case 'P':
          case 'p':
            Kp = Value;
            break;

          case 'I':
          case 'i':
            Ki = Value;
            PID_Restart = 1;
            break;

          case 'D':
          case 'd':
            Kd = Value;
            break;
          default:
            break;
    }
  }
  else if ( (RxBuff[0] == 'P') || (RxBuff[0] == 'p') )
  {
    switch (RxBuff[1])
        {
          case 'P':
          case 'p':
            PWMC1_SetDuty(0,(int)Value * PWM_MULTIPLIER);
            PWMC1_Load();
            PID_Enable = 0;
            break;
          case 'd':
          case 'D':
                PID_Enable = 1;
                break;
          default:
            break;
    }
  }
  else if ( (RxBuff[0] == 'D') || (RxBuff[0] == 'd') )
  {
    switch (RxBuff[1])
        {
```

```
          case 'S':
          case 's':
            DesiredSpeed = Value;
            break;
          default:
            break;
    }
  }
}
/*
 * Function that is used to calculate new PWM value using PID control algorithm.
 */
void PID_Control(void)
{
  long NewPWM_duty;
  static long Integral = 0;
  static int PreviousErr;
  int Err;
  unsigned long NewAdc = 0;
  u8 i;

  if (PID_Restart)
  {
        Integral = 0;
        PID_Restart = 0;
  }

  for (i=MAX_ADC_ELEMENTS; i!=0; i--)
  {
        NewAdc += AdcFilterBuf[i-1];
  }

  NewAdc = NewAdc >> 3;
  NewAdc /= MAX_ADC_ELEMENTS;

  Err = (int)(DesiredSpeed - (int)(NewAdc));

  Integral += Err;

  if (Integral > INTEGRAL_WINDUP)
  {
        Integral = INTEGRAL_WINDUP;
  }
  else if (Integral < -INTEGRAL_WINDUP)
  {
        Integral = -INTEGRAL_WINDUP;
  }

  NewPWM_duty=((long)Kp*Err)+((long)Ki*Integral*T_SAMPLE/1000)+((long)Kd*(Err-PreviousErr)/T_SAMPLE);

  PreviousErr = Err;

  NewPWM_duty = NewPWM_duty/100;

  if (NewPWM_duty < 0)
  {
        NewPWM_duty = 0;
  }
  else if (NewPWM_duty > 500)
  {
        NewPWM_duty = 500;
  }

  if (!EditMode)
  {
    printf("N=%d ", (u16)NewPWM_duty);
    printf("A=%d\r\n", (u16)NewAdc);
  }

  if (PID_Enable)
  {
        PWMC1_SetDuty(0,(int)(NewPWM_duty * PWM_MULTIPLIER));
    PWMC1_Load();
  }

}


void main(void)
{
  /* Write your local variable definition here */


  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/
  /* My test code */
  GPIO_C_DDR = 3;         // Config port C bit 0 and 1 as output.
  GPIO_C_PER &=~3;
```

```
  GPIO_C_DR = 1; // Init port C bit 0 and 1

  if (AD1_EnableIntTrigger() != ERR_OK)
  {
        asm(debughlt);
  }

  /*
   * Initialize tiny_printf module. This module is used instead of standard library
   * due to smaller size that will result in smaller application footprint.
   */
  init_printf(NULL, __putc_b);

  /* Write your code here */

  for(;;)
  {
        if (NewAD)
        {
           /*
            * New value from AD converter is available. Process it and calculate new PWM value.
            */
            NewAD = 0;
            PID_Control();
        }

        if (ExecuteCommand)
        {
           /*
            * New command is received, process it.
            */
                CommandParser();
                ExecuteCommand = 0;
        }
  }
}

/* END DSP_Seminarska */
/*
** ###################################################################
**
**     This file was created by UNIS Processor Expert 2.99 [04.17]
**     for the Freescale 56800 series of microcontrollers.
**
** ###################################################################
*/
```

1.2 Events.c

```
/** ###################################################################
**     Filename  : Events.C
**     Project   : DSP_Seminarska
**     Processor : 56F8322
**     Beantype  : Events
**     Version   : Driver 01.03
**     Compiler  : Metrowerks DSP C Compiler
**     Date/Time : 07.08.2010, 13:53
**     Abstract  :
**         This is user's event module.
**         Put your event handler code here.
**     Settings  :
**     Contents  :
**         AD1_OnEnd - void AD1_OnEnd(void);
**
**     (c) Copyright UNIS, a.s. 1997-2008
**     UNIS, a.s.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
** ###################################################################*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"
#include "custom_types.h"
#include "configuration_file.h"

unsigned int NewAD = 0;


/*
** ===================================================================
**     Event       :  AD1_OnEnd (module Events)
**
```

```
**     From bean   :  AD1 [ADC]
**     Description :
**         This event is called after the measurement (which consists
**         of <1 or more conversions>) is/are finished.
**         The event is available only when the <Interrupt
**         service/event> property is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
unsigned int AdcSample;
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the ISR)      */

u16 AdcFilterBuf[MAX_ADC_ELEMENTS];
u16 AdcIndex = 0;

/*
 * This function is called when AD conversion is over. Due to existence of noise in the acquired values,
 * buffer with more than one value is created that is then averaged prior it is used. Using this approach,
 * the PWM control works more stabile.
 */
void AD1_OnEnd(void)
{
  /* Write your code here ... */
  unsigned int i= 0x08;
  if(AD1_GetValue(&AdcSample) != ERR_OK)
  {
        asm(debughlt);
  }

  AdcFilterBuf[AdcIndex++] = AdcSample;
  if (AdcIndex == MAX_ADC_ELEMENTS)
  {
        AdcIndex = 0;
  }

  GPIO_C_DR |= 0x0001;
  while(i--)
  {
        asm
        {
                nop;
        }
  }
  GPIO_C_DR &= 0x0001;
}

/*
** ===================================================================
**     Event       :  TI1_OnInterrupt (module Events)
**
**     From bean   :  TI1 [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the bean is enabled - <Enable> and the events are
**         enabled - <EnableEvent>). This event is enabled only if a
**         <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the ISR)      */
void TI1_OnInterrupt(void)
{
  /* Write your code here ... */
  NewAD = 1;
}

/*
** ===================================================================
**     Event       :  AS1_OnError (module Events)
**
**     From bean   :  AS1 [AsynchroSerial]
**     Description :
**         This event is called when a channel error (not the error
**         returned by a given method) occurs. The errors can be
**         read using <GetError> method.
**         The event is available only when the <Interrupt
**         service/event> property is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the ISR)      */
void AS1_OnError(void)
{
```

```
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  AS1_OnTxChar (module Events)
**
**     From bean   :  AS1 [AsynchroSerial]
**     Description :
**         This event is called after a character is transmitted.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the ISR)      */
void AS1_OnTxChar(void)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  AS1_OnRxChar (module Events)
**
**     From bean   :  AS1 [AsynchroSerial]
**     Description :
**         This event is called after a correct character is
**         received.
**         The event is available only when the <Interrupt
**         service/event> property is enabled and either the
**         <Receiver> property is enabled or the <SCI output mode>
**         property (if supported) is set to Single-wire mode.
**         Version specific information for Freescale 56800
**         derivatives ]
**         DMA mode:
**         If DMA controller is available on the selected CPU and
**         the receiver is configured to use DMA controller then
**         this event is disabled. Only OnFullRxBuf method can be
**         used in DMA mode.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
extern u8 ExecuteCommand;
extern u8 EditMode;

u8 RxBuff[MAX_NR_CHARS];
u8 RxNrChars = 0;

#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the ISR)      */
void AS1_OnRxChar(void)
{
  /* Write your code here ... */
  u8 RecvChr;

  /* If previously received command is not executed or error in receive is detected
   * exit from the interrupt.
   */
  if ((ERR_OK != AS1_RecvChar(&RecvChr)) || ExecuteCommand)
  {
        return;
  }

  if (RecvChr == '\r')
  {
        if (EditMode)
        {
                EditMode = 0;
                ExecuteCommand = 1;
        }
        else
        {
                EditMode = 1;
                RxNrChars = 0;
        }

        return;
  }

  RxBuff[RxNrChars]=RecvChr;
  AS1_SendChar(RecvChr);

  if ((RxNrChars++) >= MAX_NR_CHARS)
  {
        RxNrChars--;
  }
```

```
}

/* END Events */

/*
** ###################################################################
**
**     This file was created by UNIS Processor Expert 2.99 [04.17]
**     for the Freescale 56800 series of microcontrollers.
**
** ###################################################################
*/
```

1.3 configuration_file.h

```
#ifndef __CONFIGURATION_FILE_H
#define __CONFIGURATION_FILE_H

#define MAX_NR_CHARS 10
#define PWM_MULTIPLIER 30
#define INTEGRAL_WINDUP 100000
#define MAX_ADC_ELEMENTS 10

#define T_SAMPLE 5 // Time is represented in ms

#endif //#ifndef __CONFIGURATION_FILE_H
```

1.4 custom_types.h

```
#ifndef __CUSTOM_TYPES_H
#define __CUSTOM_TYPES_H

typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;

#endif //#ifndef __CUSTOM_TYPES_H
```