4. PWM of MC9S12DJ256 microcontroller

4.1 Purpose of the work

Developing skills related to PWM (Pulse Width Modulation) modulation and generate PWM signals, reading duty cycle and frequency of signals on some inputs, configured using a microcontroller.

Skills at the end of this lab you should be:

- programming the microcontroller PWM;
- application testing using the methodology proposed in the paper presented in Chapter 2.
- correct measurement of the generated PWM signal using oscilloscope probe and board terminals.

4.2 Types of PWM signals

In the literature, several types of PWM signals are permitted depending on whether the defining of the duty cycle (duty cycle) or as it is determined the reference. Thus, if the S12 microcontroller we can configure:

- type A (trail): the reference is given by the rising edge and the duty cycle is defined as the ratio of pulse duration on the level "HIGH" and the signal period(Figure 27);

Figure 27 – Type A PWM signal

- type B (lead): the reference is given by the falling edge of the signal and the duty cycle is defined as the ratio of pulse duration per level "LOW" and the signal period(Figure 28).

Figure 28 - Type B PWM signal

- Type C (center): the reference is given by the "center" of the pulse on "HIGH" level of the signal and the duty cycle is defined as the ratio of pulse duration per level "HIGH" and the signal period (Figure 29).

Figure 29 – Type C PWM signal

4.3 PWM module

80-pin version of the HCS12 microcontroller has PWM_8B8C module on 8 channels, with independent control of the left and central alignment of the outputs. Each of the 8 channels has facilities of programmable period and duty cycle and a dedicated counter. Can be used four different clock sources with the counters. The signals on the 8 channels can be generated independently for each channel, with a duty cycle ranging from 0% to 100%. The block diagram of this module is shown in Figure 30.

The module also has additionally the following features:

- Period and duty cycle have double buffer feature. Changes take effect when it is reached the end of the effective period (PWM counter reaches 0), or when the channel is disabled.

- 8 channels with 8-bit resolution or 4 channels with 16-bit resolution. 8

- The four clock sources are A, B, SA (Scaled A), SB (Scaled B).

Figure 30 – Block diagram of PWM_8B8C module (Freescale, 2004)

PWM Clock Select

Clock A and clock B can be software selected as 1, 1/2, 1/4, 1/8, ..., 1/64, 1/128 of the bus clock. Clock SA uses clock A as input and divides it further with a default value (2, 4, 6, 8, ... or 512). Clock SB works similarly, with the clock B input. Each PWM channel has the ability to select one of the two clock signals or prescaled clock signal (A or B) or the scaled (SA or SB). Clock scaling values for A and B can be set by bits PCKA2, PCKA1, PCKA0 or PCKB2, PCKB1, PCKB0 from PWMPRCLK register**.**

PWM Channel Timers

Each timer channel has a counter, a period register and duty cycle register. Each such register is on 8 bits. The period of the waveform supplied is controlled by matching the period register and counter value. Duty cycle is controlled by a match between the duty cycle register and counter value and causes the output state change. Its polarity is also selected for each channel.

Interrupts

Supports only one interrupt which is generated at the emergency stop moment if PWMIE bit is set. This bit represents enabled interruption.

PWM registers are shown in Table 7.

Address	Use	Access
$_00	PWM Enable Register (PWME)	R/W
$_01	PWM Polarity Register (PWMPOL)	R/W
$_02	PWM Clock Select Register (PWMCLK)	R/W
$_03	PWM Prescale Clock Select Register (PWMPRCLK)	R/W
$_04	PWM Center Align Enable Register (PWMCAE)	R/W
$_05	PWM Control Register (PWMCTL)	R/W
$_06	PWM Test Register (PWMTST)	R/W
$_07	PWM Prescale Counter Register (PWMPRSC)	R/W
$_08	PWM Scale A Register (PWMSCLA)	R/W
$_09	PWM Scale B Register (PWMSCLB)	R/W
$_0A	PWM Scale A Counter Register (PWMSCNTA)	R/W
$_0B	PWM Scale B Counter Register (PWMSCNTB)	R/W
$_0C	PWM Channel 0 Counter Register (PWMCNT0)	R/W
$_0D	PWM Channel 1 Counter Register (PWMCNT1)	R/W
$_0E	PWM Channel 2 Counter Register (PWMCNT2)	R/W
$_0F	PWM Channel 3 Counter Register (PWMCNT3)	R/W
$_10	PWM Channel 4 Counter Register (PWMCNT4)	R/W
$_11	PWM Channel 5 Counter Register (PWMCNT5)	R/W
$_12	PWM Channel 6 Counter Register (PWMCNT6)	R/W
$_13	PWM Channel 7 Counter Register (PWMCNT7)	R/W
$_14	PWM Channel 0 Period Register (PWMPER0)	R/W
$_15	PWM Channel 1 Period Register (PWMPER1)	R/W
$_16	PWM Channel 2 Period Register (PWMPER2)	R/W
$_17	PWM Channel 3 Period Register (PWMPER3)	R/W
$_18	PWM Channel 4 Period Register (PWMPER4)	R/W
$_19	PWM Channel 5 Period Register (PWMPER5)	R/W
$_1A	PWM Channel 6 Period Register (PWMPER6)	R/W
$_1B	PWM Channel 7 Period Register (PWMPER7)	R/W
$_1C	PWM Channel 0 Duty Register (PWMDTY0)	R/W
$_1D	PWM Channel 1 Duty Register (PWMDTY1)	R/W
$_1E	PWM Channel 2 Duty Register (PWMDTY2)	R/W
$_1F	PWM Channel 3 Duty Register (PWMDTY3)	R/W
$_20	PWM Channel 4 Duty Register (PWMDTY4)	R/W
$_21	PWM Channel 5 Duty Register (PWMDTY5)	R/W
$_22	PWM Channel 6 Duty Register (PWMDTY6)	R/W
$_23	PWM Channel 7 Duty Register (PWMDTY7)	R/W
$_24	PWM Shutdown Register (PWMSDN)	R/W
$_25	Reserved	R
$_26	Reserved	R
$_27	Reserved	R

Table 7 – Registers of PWM module

4.4 Development of the work

1. Defining the requirements of the work.

    a. Generating signals using PWM module with a certain period and a certain duty cycle.

2. Open the CodeWarrior project related to work 4 ie .mcp file ending. The code for this work is presented below.

```
////////////////////////////////////////////////////////////////////////////
//
// Sample for SofTec Microsystems ZK-S12-B Starter Kit
//
// ---------------------------------------------------------------------------------------
//
// This project has been written for CodeWarrior 3.1 for HC(S)12
// and uses Registers Files (MC9S12DJ256.h and MC9S12DJ256.c) v. 1.052
//
// ---------------------------------------------------------------------------------------
//
// Copyright (c) 2005 SofTec Microsystems
// http://www.softecmicro.com/
//
////////////////////////////////////////////////////////////////////////////

#include <hidef.h>
#include "mc9s12dj256.h"
#include "mscan.h"
#pragma LINK_INFO DERIVATIVE "mc9s12dj256b"

////////////////////////////////////////////////////////////////////////////
// Defines and variables
////////////////////////////////////////////////////////////////////////////
unsigned char potentiometer_value;
Bool can_send_enable;

////////////////////////////////////////////////////////////////////////////
// Variables
////////////////////////////////////////////////////////////////////////////
extern unsigned char *can_periph[];

////////////////////////////////////////////////////////////////////////////
// Peripheral Initialization
////////////////////////////////////////////////////////////////////////////
#pragma CODE_SEG
#define ADC_8_BIT 1
// this value sets the conversion on 8-bit or 10-bit

void PeriphInit(void)
{
  //Configure PB[7..0] as output port
   PORTB = 0x00;
   DDRB = 0xFF;
```

```
   // Configure PP[7..0]
   PTP = 0x00;

   // Configure  PWM module
   DDRP = 0xFF;
   RDRP = 0x00;
   PWME=0x00;
   PWMPOL=0xF0;
   PWMCLK=0x8C;
   PWMPRCLK=0x43;
   PWMSCLB = 250;
   PWMCTL=0x00;
   PWMSDN=0x00;
   PWMCAE=0x00;

   // Configure ADC peripheral
#   if(ADC_8_BIT == 1)
   // ATD0: 8 conversions per sequence, 8 bit resolution, continuous conversion
     ATD0CTL3 = 0x00; // 8 conversions per sequence
     ATD0CTL4 = 0x80; // 8 bit resolution
     ATD0CTL2 = 0x80; // normal operation
     ATD0CTL5 = 0x20; // left justified; scan mode across channels
#   else
   // ATD0: 8 conversions per sequence, 10 bit resolution, continuous conversion
     ATD0CTL3 = 0x; // eight conversions per sequence.
     ATD0CTL4 = 0x; // 10 bit resolution
     ATD0CTL2 = 0x; // normal operation
     ATD0CTL5 = 0x; // right justified; scan mode across channels
#   endif
   CANInit(0);
   EnableInterrupts;
}

////////////////////////////////////////////////////////////////////////////////
// Entry point
////////////////////////////////////////////////////////////////////////////////

void main(void)
{   unsigned char i=0;
   struct can_msg msg_send;
   struct can_msg *msg;
   PeriphInit();
   msg=msg_rdy=&msg_buff[0];

    asm("cli");
   for(;;)
   {
    while(!(ATD0STAT0 & 0x80));
    # if(ADC_8_BI T== 1)
    PORTB=ATD0DR0H;
    # else
    PORTB=ATD0DR0L;
    # endif
    // Resets SCF flag
    ATD0STAT0 = 0x80;
```

```
if (msg!=msg_rdy){
 // message received on CAN BUS
 msg_send.id = (unsigned int)0x000F00|msg->id;

 if(msg->id==CAN_MSG_ID_CMD){
  //decode command message
  switch(msg->data[0]){
   case 0: { // COMMAND 00: software version
          memcpy(msg_send.data,"SI-SW2.0",8);
          msg_send.len = 8;
          break;
   }
   case 1:{ // COMANDA 01: digital read (read from the corresponding port)
          char *p;
          // message length is less than 3-digit hexadecimal
          if(msg->len<3){
            memcpy(msg_send.data,"bad-len",7); // sending an error message
            msg_send.len = 7;
            }
         //the readings have to do with A or P ports
          else if(msg->data[1]!='A'&&msg->data[1]!='P'){
            memcpy(msg_send.data,"bad-port",8); // sending an error message
            msg_send.len = 8;
            }
          // maximum 8 pins can be read
          else if(msg->data[2]>7){
            memcpy(msg_send.data,"bad-pin",7);// sending an error message
            msg_send.len = 7;
            }
          else{unsigned char i=1;
            p=msg->data[1]!='A'?&PTP:&PORTA;
            i<<=msg->data[2];
            msg_send.data[0]=(*p&i)>>msg->data[2];
            msg_send.len = 1;
            }
          break;
   }
   case 2:{// COMANDA 02: digital write (write at the corresponding por)
           // message length is less than 4-digit hexadecimal
          if(msg->len<4){
            memcpy(msg_send.data,"bad-len",7); // sending an error message
            msg_send.len = 7;
            }
          // writing, it's about port B
          else if(msg->data[1]!='B'){
            memcpy(msg_send.data,"bad-port",8); // sending an error message
            msg_send.len = 8;
            }
           // se pot citi maxim 8 pini
          else if(msg->data[2]>7){
            memcpy(msg_send.data,"bad-pin",7); // sending an error message
            msg_send.len = 7;
            }
          else{unsigned char i=1;
```

```
            i<<=msg->data[2];
            PORTB =msg->data[3]?PORTB|i:PORTB&(~i);
           // memcpy(msg_send.data,"out:",4);
            msg_send.data[0]=PORTB;
            msg_send.len = 1;
            }
           break;
      }

// ---------------------------------------------------------------------------------------
// Conversion COMMAND
// ---------------------------------------------------------------------------------------

      case 3:{ // COMMAND 03: analogic read
      // message length is less than 2-digit hexadecimal
      if(msg->len<2){
            memcpy(msg_send.data,"bad-len",7); // sending an error message
            msg_send.len = 7;
            }
           else if(msg->data[1]>7){
            memcpy(msg_send.data,"bad-ch",6); // sending an error message
            msg_send.len = 6;
            }
           else{unsigned char i=1;
            i<<=msg->data[1];
            # if(ADC_8_BIT!=1)
              // reset SCF flag
              //ATD0CTL5 right justified; scan mode across channels
              // Reads the ADC channel
             while(!(ATD0STAT0 & 0x80));
             switch(msg->data[1]){
              //result is in the result registers, 10-bit conversion
              case 0:{
                   msg_send.data[0] = ATD0DR0H&0x03;
                      msg_send.data[1] = ATD0DR0L;
                break;
               }
              case 1:{
                   msg_send.data[0] = ATD0DR1H&0x03;
                      msg_send.data[1] = ATD0DR1L;
                break;
               }
              case 2:{
                   msg_send.data[0] = ATD0DR2H&0x03;
                      msg_send.data[1] = ATD0DR2L;
                break;
               }
              case 3:{
                   msg_send.data[0] = ATD0DR3H&0x03;
                      msg_send.data[1] = ATD0DR3L;
                break;
               }
              case 4:{
                   msg_send.data[0] = ATD0DR4H&0x03;
                      msg_send.data[1] = ATD0DR4L;
```

```
    break;
   }
  case 5:{
       msg_send.data[0] = ATD0DR5H&0x03;
          msg_send.data[1] = ATD0DR5L;
    break;
   }
  case 6:{
       msg_send.data[0] = ATD0DR6H&0x03;
          msg_send.data[1] = ATD0DR6L;
    break;
   }
  default:{
       msg_send.data[0] = ATD0DR7H&0x03;
          msg_send.data[1] = ATD0DR7L;
   }
  }
      msg_send.len=2;
      ATD0STAT0 = 0x80;
#else
// reset SCF flag
//ATD0CTL5 left justified; scan mode across channels
// Reads the ADC channel
while(!(ATD0STAT0 & 0x80));
switch(msg->data[1]){
 //the result is in only one result register, 8-bit conversion
   case 0:{
       msg_send.data[0] = ATD0DR0H;
    break;
   }
  case 1:{
       msg_send.data[0] = ATD0DR1H;
    break;
   }
  case 2:{
       msg_send.data[0] = ATD0DR2H;
    break;
   }
  case 3:{
       msg_send.data[0] = ATD0DR3H;
    break;
   }
  case 4:{
       msg_send.data[0] = ATD0DR4H;
    break;
   }
  case 5:{
       msg_send.data[0] = ATD0DR5H;
    break;
   }
  case 6:{
       msg_send.data[0] = ATD0DR6H;
    break;
   }
  default:{
```

```
                msg_send.data[0] = ATD0DR7H;
              }
             }
            ATD0STAT0 = 0x80;
                msg_send.len=1;
          #endif
          }
          break;
      }
      case 4:{ // COMMAND 04: PWM control signal generating handling
          // Message length is less than 4 hexadecimal digits
          if(msg->len<4){
            memcpy(msg_send.data,"bad-len",7);
            msg_send.len = 7;
            }
          else if(msg->data[1]>7||msg->data[1]==6){ // handling channel 6 situation
            memcpy(msg_send.data,"bad-chan",8);
            msg_send.len = 8;
            }
          else if(msg->data[3]>100){
            memcpy(msg_send.data,"bad-DC",6);
            msg_send.len = 6;
            }
          else{unsigned char i=1;
            set_pwm(msg->data[1],msg->data[2],msg->data[3]);
            msg_send.data[0]=0;
            msg_send.len = 1;
            }
          break;
      }
      default: { // send error message
        memcpy(msg_send.data,"bad_cmd",7);
        msg_send.len = 7;
      }
     }
    }
    else{ // send error message
        memcpy(msg_send.data,"bad_msg",7);
        msg_send.len = 7;
    }

    // prepare for next message
    SendCANMessage(msg->dev, msg_send);
    msg=msg_rdy;
   }
  }
}

void set_pwm(unsigned char ch, unsigned char per, unsigned char dc){
 unsigned char pwm_e;
 pwm_e=PWME;
 switch (ch){
  case 0:{unsigned int dc_v;
   PWME=pwm_e&0xFE;
   if (!per || !dc) break;
```

```
  dc_v= per * dc / 100;
  PWMPER0=per;
  PWMDTY0=dc_v;
  PWME=pwm_e|0x1;
  break;
}
case 1:{unsigned int dc_v;
  PWME=pwm_e&0xFD;
  if (!per || !dc) break;
  dc_v= per * dc / 100;
  PWMPER1=per;
  PWMDTY1=dc_v;
  PWME=pwm_e|0x2;
  break;
}
case 2:{unsigned int dc_v;
  PWME=pwm_e&0xFB;
  if (!per || !dc) break;
  dc_v= per * dc / 100;
  PWMPER2=per;
  PWMDTY2=dc_v;
  PWME=pwm_e|0x4;
  break;
}
case 3:{unsigned int dc_v;
  PWME=pwm_e&0xF7;
  if (!per || !dc) break;
  dc_v= per * dc / 100;
  PWMPER3=per;
  PWMDTY3=dc_v;
  PWME=pwm_e|0x8;
  break;
}
case 4:{unsigned int dc_v;
  PWME=pwm_e&0xEF;
  if (!per || !dc) break;
  dc_v= per * dc / 100;
  PWMPER4=per;
  PWMDTY4=dc_v;
  PWME=pwm_e|0x10;
  break;
}
case 5:{unsigned int dc_v;
  PWME=pwm_e&0xDF;
  if (!per || !dc) break;
  dc_v= per * dc / 100;
  PWMPER5=per;
  PWMDTY5=dc_v;
  PWME=pwm_e|0x20;
  break;
}
default:{unsigned int dc_v;
  PWME=pwm_e&0x7F;
  if (!per || !dc) break;
  dc_v= per * dc / 100;
```

```
    PWMPER7=per;
    PWMDTY7=dc_v;
    PWME=pwm_e|0x80;
  }
 }
}
//////////////////////////////////////////
#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void CAN0RcvISR(void){
  ReceiveCANMessage(0);
}

#pragma CODE_SEG DEFAULT
 //////////////////////////////////////////
#pragma CODE_SEG __NEAR_SEG NON_BANKED

interrupt void CAN4RcvISR(void){
  ReceiveCANMessage(1);
}
#pragma CODE_SEG DEFAULT
```

3. Analysis of the proposed configurations. Emphasis will be on understanding how the registers were set in PeriphInit function () from C code associated to this paper. In worksheet will just mention bits significance vis-a-vis the values proposed, considering the PWM types convention given in paragraph 4.2. Use PWM module specification S12PWM8B8CV1-pwm.pdf and and documentation for HCS12 ports: S12DP256PIMV2-portguide.pdf.

a) PTP = 0x00;

b) DDRP = 0xFF;

c) RDRP = 0x00;

d) PWME=0x00;

e) PWMPOL=0xF0;

f) PWMCLK=0x8C;

g) PWMPRCLK=0x43;

h) PWMSCLB = 250;

i) PWMCTL=0x00;

j) PWMSDN=0x00;

k) PWMCAE=0x00;

4. Command used in PcanView are (see the above code and comments):

a. Command 00: involves returning as message the software version used

b. Command 04: command for generating a PWM signal. Has the structure:

i. Byte 0: command [04]

ii. Byte 1: channel PWM [00...07]

iii. Byte 2: period in milliseconds or microseconds [00...FF]

iv. Duty cycle in percent ([0...64])

Example:

04 00 64 19 introduced in PcanView as test message, if the programming was done correct, should generate a PWM signal with 100ms period and 25% duty cycle(testing development phase *!*).

If the duty cycle or the period are null, the generated PWM signal on that channel will be stopped.

5. Compile the project in CodeWarrior and send it to the microcontroller. Run code written in Flash memory

6. Send the following commands by USB2CAN to the development board, with the measurement of the signal on pins of the PWM channels that are referenced. For the measurements, the pins are identified using appropriate schema of the HCS12 80-pin capsule.

6.a Extended ID: 001h Message: 00

6.b Extended ID: 001h Message: 04 00 64 19

6.c IExtended ID : 001h Message: 04 01 64 46

6.d Extended ID: 001h Message: 04 02 64 28

6.e Extended ID: 001h Message: 04 00 64 0

6.f Extended ID: 001h Message: 04 03 64 32

6.gExtended ID : 001h Message: 04 04 64 3C

6.h Extended ID: 001h Message: 04 05 64 55

6.i Extended ID: 001h Message: 04 07 64 23

4.5 Abstract of the paper

- There are various types of PWM signals based on the mode of defining the duty cycle or determined by how reference is establish.
- Generating a PWM signal is achieved by command 04.
- Testing the correct programming of the code for the PWM module is achieved by obtaining the desired signal on sending a message on PcanView. Measurements are performed using the oscilloscope.

4.6 Commented References

1. Datasheet for the family HCS12: 9S12DP256BDGV2-userguide.pdf (Freescale, 2005) presents the different functions of the microcontroller terminals.
2. Datasheet for the ports of HCS12 microcontroller: S12DP256PIMV2-portguide.pdf (Freescale, Port Integration Module, 2002) presents the description of the ports.
3. Datasheet for the PWM module of the HCS12 microcontroller: S12PWM8B8CV1-pwm.pdf (Freescale, PWM_8B8C, 2004) presents the description of this module
4. Deepening the PWM of HCS12 microcontroller can be done by referring to sections 8.10 in (Huang, 2010).
5. Mandatory consultation of (Popa, 2011).

4.7 Worksheet

Write you name:……

Semigroup:……..

1. What can you tell about PWM6 channel vis-à-vis the scheme of the S12 80 pin capsule
…….....…

2. What is the meaning of the configuration from 4.4.3 section:

a) …….

b) …….

c) …….

d) …….

e) …….

f) …….

g) …….

h) …….

i) …….

j) …….

k) …….

3. What message you received at dection 4.4.6.a:ID: __ MSG: _____

4. Performing the experiments in 4.4.6, write values and units between square brackets [], obtained on oscilloscope for pins associated with data commands onUSB2CAN:

6.b PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]

6.c PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]

6.d PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]

6.e PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]

6.f PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]

6.g PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]

6.h PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]

6.i PosWidth: ____ [___] NegWidth: ____ [___] Period: ____ [___]