Application using microcontroller MC9S12DJ256

6.1. Purpose

Familiarity with using the module CAPTURE-COMPARE to generate periodic events. Using the knowledge gained to create an application: measuring the distance using HCS12 microcontroller, distance sensor GP2Y0A21YK0F and interface USB2CAN.

Skills required at the end of this lab:

- Programming of capture-compare module based on related registers;
- testing proposed application in this paper using the methodology presented in Chapter 2 and customized for this work.

6.2. Capture-compare module

These type modules capture-compare have multiple functions, generally used to measure the waveform of input signals ("duty cycle measurement"), to generate some output waveform or just as timer, as used in this paper.

Figure 1 - ECT_16B8C Module (Freescale, 2002)

Configured as timer, each channel has assigned a 16-bit register whose value will be compared with the value of the counter common to all channels (Figure 32). That number is incremented by a signal derived from the bus-clock after division by prescale register. If the value of the timer coincides with the register value associated to a channel, an interrupt can be generate, and the count goes to "overflow" or resumes from 0 immediately after the event.

6.3. GP2Y0A21YK0F Sensor

It can be used to measure distances between 10 and 80 cm. Distance is "read" as an analogue outputs (values between -0.3V and VCC + 0.3V) and the supply voltage is between 4.5V to 5.5V. The transfer characteristic is nonlinear, the output voltage

decreasing as the distance increases.

Figure 2 - The transfer characteristic of the sensor (Sharp, 2006)

Under voltage, the output is unstable for a period that not exceeding 48 ms, and the distance measured is available with a maximum of 5ms delay, as show in the manufacturer specifications.

6.4. Conducting work

1. Defining the requirements of the work

   a. The code associated with the work, implemented distance measurement using sensor GP2Y0A21YK0F, given the constraints in terms of response time. When a command is received throw USB2CAN, the application must wait until information is available about the relative distance at that point of time and to return in centimeters using a table of the type "look-up" named CalibTab. For this purpose will be calibrate some reference values (using numerical value acquired by performing measurements) and for intermediate distances values are obtained by interpolation. For measurements using a blank sheet of A4, which are positioned perpendicular to the beam emitted by the distance sensor:

10 cm	15 cm	20 cm	25 cm	40 cm	70 m

2. Open the CodeWarrior project related in this paper , namely file ending .mcp. This essential code for execute measurements 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
```

```
////////////////////////////////////////////////////////////////////////////////
#define ADC_8_BIT  1
#define MAX_CALIB  6

// SHARP distance sensor unstable output time in [ms]
#define DIST_UNSTABLE
// SHARP distance sensor measure time in [ms]
#define DIST_READY

unsigned short ms_ctr=0;
unsigned char CalibTab[MAX_CALIB][2]={
        {10, },
        {15, },
        {20, },
        {25, },
        {40, },
        {70, },
};

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

////////////////////////////////////////////////////////////////////////////////
// Distance sensor initialization
////////////////////////////////////////////////////////////////////////////////

unsigned char GetDistance(unsigned char val){
 unsigned char i;
 unsigned char max;

 max=MAX_CALIB-1;
 if(val>=CalibTab[0][1]) return CalibTab[0][0];
 if(val<=CalibTab[max][1]) return CalibTab[max][0];

 for (i=1;val<CalibTab[i][1] && i<max;i++);

 return CalibTab[i-1][0] +
     (unsigned int)(CalibTab[i][0]-CalibTab[i-1][0]) *
     (CalibTab[i-1][1]-val)/(CalibTab[i-1][1]-CalibTab[i][1]);

}

////////////////////////////////////////////////////////////////////////////////
// Distance sensor initialization
```

```
////////////////////////////////////////////////////////////////////////////
void DistSensInit(unsigned short t_ms){
  while (ms_ctr<t_ms);
}

////////////////////////////////////////////////////////////////////////////
// Peripheral Initialization
////////////////////////////////////////////////////////////////////////////
#pragma CODE_SEG DEFAULT

void PeriphInit(void)
{
   // Configurare PB[7..0] ca şi ieşire
   PORTB = 0x00;
   DDRB = 0xFF;

   DDRA = 0x00;
   // Configurare periferie ATD
   // ATD0: 1 conversions per sequence, 8 bit resolution, continuous conversion
   ATD0CTL3 = 0x00; // one conv per sequence
   ATD0CTL4 = 0x82; // 8 bit resolution
   ATD0CTL2 = 0x80; // normal operation
   ATD0CTL5 = 0x30; // left justified; scan mode across channels

   // Configurare Enhanced Capture Timer
   // to generate the interruption at every 1 ms
   // (on Chanel 7)
   PTT = 0x00;
   DDRT = 0x00;
   PERT = 0xFF;
   PPST = 0xFF;

   TIOS = 0x80;
   TIE = 0x80;
   TC7 = 8000;
   TSCR2 = 0x08;
   TSCR1 = 0xA0;

   CANInit(0);
   EnableInterrupts;
}

////////////////////////////////////////////////////////////////////////////
// Entry point
////////////////////////////////////////////////////////////////////////////
void main(void)
```

```
{   unsigned char i=0;
    struct can_msg msg_send;
    struct can_msg *msg;
    PeriphInit();
    DistSensInit((unsigned short)(DIST_UNSTABLE));

    msg=msg_rdy=&msg_buff[0];

    asm("cli");
    for(;;)
    {
     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:{ // software version
                memcpy(msg_send.data,"SI-SW6.0",8);
                msg_send.len = 8;
                break;
        }
        case 1:{ // digital read
                char *p;
                if(msg->len<3){
                  memcpy(msg_send.data,"bad-len",7);
                  msg_send.len = 7;
                  }
                else if(msg->data[1]!='A'&&msg->data[1]!='P'){
                  memcpy(msg_send.data,"bad-port",8);
                  msg_send.len = 8;
                  }
                else if(msg->data[2]>7){
                  memcpy(msg_send.data,"bad-pin",7);
                  msg_send.len = 7;
                  }
                else{unsigned char i=1;
                  p=msg->data[1]!='A'?&PTP:&PORTA;
                  i<<=msg->data[2];
                 // memcpy(msg_send.data,"in:",3);
                  msg_send.data[0]=(*p&i)>>msg->data[2];
                  msg_send.len = 1;
                  }
                break;
        }
```

```
case 2:{ // digital write
      if(msg->len<4){
        memcpy(msg_send.data,"bad-len",7);
        msg_send.len = 7;
        }
      else if(msg->data[1]!='B'){
        memcpy(msg_send.data,"bad-port",8);
        msg_send.len = 8;
        }
      else if(msg->data[2]>7){
        memcpy(msg_send.data,"bad-pin",7);
        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;
}
case 6:{ // COMANDA 06: distance read
if(msg->len<2){ // two hex digits will command
        memcpy(msg_send.data,"bad-len",7); // error message
        msg_send.len = 7;
        }
      else if(msg->data[1]>7){
        memcpy(msg_send.data,"bad-ch",6); // error message
        msg_send.len = 6;
        }
      else{  unsigned short new_ctr;
         new_ctr=ms_ctr+DIST_READY;
         while(new_ctr!=ms_ctr);
         // reset SCF flag
         //ATD0CTL5 left justified; scan mode across channels
         ATD0STAT0 = 0x80;
         // Citeşte canalul ADC
         while(!(ATD0STAT0 & 0x80)&&(ATD0STAT0 & 0x07)!=msg->data[1]);
         switch(msg->data[1]){
           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;
               }
              }
            PORTB=msg_send.data[0];
            msg_send.data[1] = GetDistance(msg_send.data[0]);
                  msg_send.len=2;
         }
         break;
  }
  default: {// error message
    memcpy(msg_send.data,"bad_cmd",7);
    msg_send.len = 7;
  }
 }
}
else{// 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;
```

```
    }
```

ATTENTION! It will also consider the last section of the file \prm\SofTec_linker.prm of project \c-code.

```
    }
  }

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

  interrupt void TimingISR(void)
  {
   ms_ctr++;TFLG1 = 0x80;
  }
  #pragma CODE_SEG DEFAULT
```

3. Analyze setting of associated registry to module ECT in PeriphInit function () in C code associated with this paper (see above) based on available resources: documentation ECT: S12ECT16B8CV1-cap_comp.pdf, specification of interrupt module INT, S12INTV1-int.pdf, afferent documentation of ports and configuration of HCS12: S12DP256PIMV2-portguide.pdf, 9S12DP256BDGV2-userguide.pdf, ZK-S12-b_schematic.pdf and catalog sheet of presented sensor gp2y0a21yk_e.pdf.

a) TIOS = 0x80;

b) TIE = 0x80;

c) TC7 = 8000;

d) TSCR2 = 0x08;

e) TSCR1 = 0xA0.

4. Set values for:

   a) DIST_UNSTABLE: time in milliseconds required to stabilize the signal at distance sensor output after it is powered on.

   b) DIST_READY : time in milliseconds, after which the distance measured is available via signal V_{out}.

5. The commands used in PcanView are (follow the code and comments above): **Command 06:** corresponding messages sent by USB2CAN to find the distance at that moment on analog channel. These messages have the structure (**extended ID 0x0001**):

   i. Byte 0: command [06]

   ii. Byte 1: analogic chanel [0-7]

   Response received after about 5 ms has the following form (**extended ID 0x0F01**):

   i. Byte 0: numerical value obtained from convertor

   ii. Byte 1: distance in cm calculated using interpolation.

6. Experiments

   1. C code from appropriate folder of this work ,will be load in CodeWarrior, fill with 6.4.4.a and 6.4.4.b requirements, to be compiled and sent to the microcontroller.

   2. Using the connection wires available at each workstation, connect sensor at GND and 5V pins of the development board section and its output connect to one of the analog channels.

   3. Run code written in Flash memory of the microcontroller.

   4. Calibrate CalibTab table as required in 6.4.3 using an white A4 sheet, which is positioned perpendicular to the beam emitted by the sensor at required distances and give CAN command for each case:

Extended ID: 001h Message: 06 ch

where *ch* is the analog channel that was linked to the sensor.

5. Obtained code is rebuilds and rewritten in the flash memory of the microcontroller.
6. Again, use the white A4 sheet, which is positioned perpendicular to the beam emitted by the sensor at following distances and give for each case command:

Extended ID: 001h Message: 06 channel-sensor

a) < 10 cm	d) 23 cm	g) 50 cm
b) 10 cm	e) 30 cm	h) 60 cm
c) 17cm	f) 45 cm	i) 70 cm.

6.5. Commented references

- Related Data Sheet to HCS12 familly HCS12 : 9S12DP256BDGV2-userguide.pdf (Freescale, 2005) presents the different functions of microcontroller terminals.
- Related Data Sheet to microcontroller ports : S12DP256PIMV2-portguide.pdf (Freescale, Port Integration Module, 2002) presents a description of the various ports registers.
- Related Data Sheet to interrupt module INT of HCS12 microcontroller: HCS12: S12INTV1-int.pdf (Freescale, Interrupt Module, 2003) presents description of this module.
- Connection to the board level of ZK-S12-B is presented in zk-s12-b_schematic.pdf (Freescale, ZK-S12-B, Schematic and Bill of Material).
- Related Data Sheet to Capture-Compare module of HCS12 microcontroller : S12ECT16B8CV1-cap_comp.pdf (Freescale, ECT_16B8C, 2002) presents description of this module.
- Related Data Sheet to analog-digital converter of HCS12 microcontroller : S12ATD10B8CV2-ADC.pdf (Freescale, ATD_10B8C, 2005) presents description of this module.

- Related Data Sheet to GP2Y0A21YK0F sensor : gp2y0a21yk_e.pdf (Sharp, 2006).
- Mandatory consultation (Popa, 2011).

6.6. Worksheet

Name, Surname:……

Semigroup:……..

1. Specify the value for:
a) DIST_UNSTABLE: b) DIST_READY:
2. Explain the content of function *GetDistance(unsigned char val)* from main.c .
3. Explain following configurations:

TIOS = 0x80:…….

TIE = 0x80:……

TC7 = 8000:……

.....……

TSCR2 = 0x08:……...……

.....…….....

TSCR1 = 0xA0:.....……

.....……

4. Which is the bus-clock frequency? Explain according to previous settings.

.....……

5. Which is the content of "Look_up" table:

Distance [cm]	10	15	20	25	40	70
Acquisition [hex]	0x	0x	0x	0x	0x	0x

6. What messages have been received on the CAN and which is the LEDs configuration from 4.5*

a) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
b) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
c) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
d) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
e) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
f) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
g) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]

h) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
i) ID: ..... Message: ..... LED [7] [6] [5] [4] [3] [2] [1] [0]
* Tick with X the heated and with - the off Led.