2. MC9S12DJ256 microcontroller ports

2.1 The purpose of the work

This paper aims to familiarize at one hand, to the concepts of input port, output port and configuration of general purpose ports within a microcontroller. On the other hand, the specific methodology presented is used in the development of an embedded application. At the end of the lab skills to be learned are:

- using the CodeWarrior development environment, the application PcanView and the methodology presented;
- MC9S12DJ256 microcontroller port configuration depending on a particular application;
- develop and test an application that involves the use of general-purpose ports;
- recognition of component sections and facilities offered by the development board ZK-12-B.

2.2 MC9S12DJ256 microcontroller

MC9S12DJ256 microcontroller (MCU for short) is a 16-bit microcontroller comprising:

1. One 16-bit HCS12 CPU;
2. 256 KB Flash memory;
3. 12 MB RAM;
4. 4 KB EEPROM;
5. two asynchronous serial communication interface (SCI);
6. three serial peripheral interface (SPI);
7. 8-channel timer;
8. two 8-channel, 10-bit analogue to digital converts (ADC);
9. PWM module (Pulse-Width Modulator) with 8 channels;
10. Byte Data Link Controller module (BDLC);
11. input / output digital channels
12. lines of digital input / output with interrupt and wake-up facilities;
13. 2 modules CAN 2.0 (CAN 0 and CAN 4);
14. Inter-IC bus.

The configuration of the terminals is shown in Figure 15, and Figure 16 shows the block diagram (Freescale, 2005).

Figure 15 - Terminal configuration to MC9S12DJ256 (Freescale, 2005)

Figure 16 - Block diagram of MC9S12DJ256 (Freescale, 2005)

Pin name Function. 1	Pin name Function. 2	Pin name Function. 3	Pin name Function. 4	Pin name Function. 5	Voltage	Internal resistor		Description
						CTRL	Reset State	
EXTAL	—	—	—	—	VDDPLL	NA	NA	Oscillator Pins
XTAL	—	—	—	—	VDDPLL	NA	NA	
RESET	—	—	—	—	VDDR	None	None	Extern Reset
TEST	—	—	—	—	N.A.	NA	NA	Test Input
VREGEN	—	—	—	—	VDDX	NA	NA	Voltage Regulator
XFC	—	—	—	—	VDDPLL	NA	NA	PLL Filter
BKGD	TAGHI	MODC	—	—	VDDR	Up	Up	Debug
PAD[15]	**AN1[7]**	**ETRIG1**	**—**	**—**	**VDDA**	**None**	**None**	**ADC pins**
PAD[14:8]	**AN1[6:0]**	**—**	**—**	**—**	**VDDA**	**None**	**None**	**ADC pins**
PAD[7]	AN0[7]	ETRIG0	—	—	VDDA	None	None	ADC pins
PAD[6:0]	AN0[6:0]	—	—	—	VDDA	None	None	ADC pins
PA[7:0]	ADDR[15:8]/ DATA[15:8]	—	—	—	VDDR	PUCR	Deactivated	Port A, Bus A/D multiplexed
PB[7:0]	ADDR[7:0]/ DATA[7:0]	—	—	—	VDDR	PUCR	Deactivated	Port A, Bus. A/D multiplexed
PE7	NOACC	XCLKS	—	—	VDDR	PUCR	Up	Port E, Access, Clock Select
PE6	IPIPE1	MODB	—	—	VDDR	Down, if RESET is driven		Port E, Pipe Status, Mode Input
PE5	IPIPE0	MODA	—	—	VDDR	Down, if RESET is driven		Port E, Pipe Status, Mode Input
PE4	ECLK	—	—	—	VDDR	PUCR	Up	Port E, Bus Clock Output
PE3	LSTRB	TAGLO	—	—	VDDR	PUCR	Up	Port E, Byte Strobe, Tag Low
PE2	R/W	—	—	—	VDDR	PUCR	Up	Port E, R/W in expanded modes
PE1	IRQ	—	—	—	VDDR	Up		Port E Input, Interrupt may be masked
PE0	XIRQ	—	—	—	VDDR			Port E Input, Interrupt may be masked
PH7	**KWH7**	**SS2**	**—**	**—**	**VDDR**	**PERH PPSH**	**Deactivated**	**Port H, Interrupt /SS from SPI2**

PH6	**KWH6**	**SCK2**	**—**	**—**	**VDDR**	**PERH PPSH**	**Deactivated**	**Port H, Interrupt SCK from SPI2**

Table 1 - Functionalities of terminals (those shown in bold are not available in 80-pin version)

Pin Name Function. 1	Pin Name Function. 2	Pin Name Function. 3	Pin Name Function. 4	Pin Name Function. 5	Voltage	Intern Resistor	Description	
PH5	**KWH5**	**MOSI2**	**—**	**—**	**VDDR**	**PERHPPS H**	**Deactivated**	**Port H, Interrupt, MOSI SPI2**
PH4	**KWH4**	**MISO2**	**—**	**—**	**VDDR**	**PERHPPS H**	**Deactivated**	**Port H, Interrupt, MISO SPI2**
PH3	**KWH3**	**SS1**	**—**	**—**	**VDDR**	**PERHPPS H**	**Deactivated**	**Port H, Interrupt, SS la SPI1**
PH2	**KWH2**	**SCK1**	**—**	**—**	**VDDR**	**PERH PPSH**	**Deactivated**	**Port H, Interrupt, SCK la SPI1**
PH1	**KWH1**	**MOSI1**	**—**	**—**	**VDDR**	**PERHPPS H**	**Deactivated**	**Port H, Interrupt, MOSI SPI1**
PH0	**KWH0**	**MISO1**	**—**	**—**	**VDDR**	**PERHPPS H**	**Deactivated**	**Port H, Interrupt, MISO SPI1**
PJ7	KWJ7	TXCAN4	SCL	TXCAN0	VDDX	PERJPPSJ	Up	Port J, Interrupt, TX at CAN4, SCL at IIC, TX at CAN0
PJ6	KWJ6	RXCAN4	SDA	RXCAN0	VDDX	PERJPPSJ	Up	Port J, Interrupt, RX at CAN4, SDA at IIC, RX at CAN0
PJ[1:0]	**KWJ[1:0]**	**—**	**—**	**—**	**VDDX**	**PERJPSJ**	**Up**	**Port J, Interrupt**
PK7	**ECS**	**ROMONE**	**—**	**—**	**VDDX**	**PUCR**	**Up**	**Port K, Emulation Chip Select, ROM On Enable**
PK[5:0]	**XADDR [19:14]**	**—**	**—**	**—**	**VDDX**	**PUCR**	**Up**	**Port K, Extended Addresses**
PM7	**TXCAN3**	**TXCAN4**	**—**	**—**	**VDDX**	**PERMPPS M**	**Deactivated**	**Port M, TX at CAN3, TX at CAN4**
PM6	**RXCAN3**	**RXCAN4**	**—**	**—**	**VDDX**	**PERMPPS M**	**Deactivated**	**Port M, RX at CAN3, RX at CAN4**
PM5	TXCAN2	TXCAN0	TXCAN4	SCK0	VDDX	PERMPPSM	Deactivated	Port M, TX at CAN2, CAN0, CAN4, SCK at SPI0
PM4	RXCAN2	RXCAN0	RXCAN4	MOSI0	VDDX	PERMPPSM	Deactivated	Port M, RX at CAN2, CAN0, CAN4, MOSI at SPI0
PM3	TXCAN1	TXCAN0	—	SS0	VDDX	PERMPPSM	Deactivated	Port M, TX at CAN1, CAN0, SS at SPI0
PM2	RXCAN1	RXCAN0	—	MISO0	VDDX	PERMPPSM	Deactivated	Port M, RX at CAN1, CAN0, MISO at SPI0
PM1	TXCAN0	TXB	—	—	VDDX	PERMPPSM	Deactivated	Port M, TX at CAN0, TX at BDLC

PM0	RXCAN0	RXB	—	—	VDDX	PERMPPSM	Deactivated	Port M, RX at CAN0, RX at BDLC
PP7	KWP7	PWM7	SCK2	—	VDDX	PERPPPSP	Deactivated	Port P, Interrupt, Channel 7 at PWM SCK at SPI2

Table 2 - Functionalities of terminals (those shown in bold are not available in 80-pin version)

Pin Name Function. 1	Pin Name Function. 1	Pin Name Function. 1	Pin Name Function. 1	Pin Name Function. 1	Voltage	Intern Resistor		Description
						CTRL	Reset State	
PP6	**KWP6**	**PWM6**	**SS2**	**—**	**VDDX**	**PERP PPSP**	**Deactivated**	**Port P, Interrupt, Channel 6 at PWM, SS at SPI2**
PP5	KWP5	PWM5	MOSI2	—	VDDX	PERP PPSP	Deactivated	Port P, Interrupt, Channel 5 at PWM, MOSI at SPI2
PP4	KWP4	PWM4	MISO2	—	VDDX	PERP PPSP	Deactivated	Port P, Interrupt, Channel 4 at PWM, MISO2 at SPI2
PP3	KWP3	PWM3	SS1	—	VDDX	PERP PPSP	Deactivated	Port P, Interrupt, Channel 3 at PWM, SS at SPI1
PP2	KWP2	PWM2	SCK1	—	VDDX	PERP PPSP	Deactivated	Port P, Interrupt, Channel 2 at PWM, SCK at SPI1
PP1	KWP1	PWM1	MOSI1	—	VDDX	PERP PPSP	Deactivated	Port P, Interrupt, Channel 1 at PWM, MOSI at SPI1
PP0	KWP0	PWM0	MISO1	—	VDDX	PERP PPSP	Deactivated	Port P, Interrupt, Channel 0 at PWM, MISO2 at SPI1
PS7	**SS0**	**—**	**—**	**—**	**VDDX**	**PERS PPSS**	**Up**	**Port S, SS at SPI0**
PS6	**SCK0**	**—**	**—**	**—**	**VDDX**	**PERS PPSS**	**Up**	**Port S, SCK at SPI0**
PS5	**MOSI0**	**—**	**—**	**—**	**VDDX**	**PERS PPSS**	**Up**	**Port S, MOSI at SPI0**
PS4	**MISO0**	**—**	**—**	**—**	**VDDX**	**PERS PPSS**	**Up**	**Port S, MISO at SPI0**
PS3	TXD1	—	—	—	VDDX	PERS PPSS	Up	Port S, TXD at SCI1
PS2	RXD1	—	—	—	VDDX	PERS PPSS	Up	Port S, RXD at SCI1
PS1	TXD0	—	—	—	VDDX	PERS PPSS	Up	Port S, TXD at SCI0
PS0	RXD0	—	—	—	VDDX	PERS PPSS	Up	Port S, RXD at SCI0
PT[7:0]	IOC[7:0]	—	—	—	VDDX	PERT PPST	Deactivated	Port T, Timer

Table 3 - Functionalities of terminals (those shown in bold are not available in 80-pin version)
Most of terminals have one or more functions, shown in Table 1, Table 2 and Table 3.

2.3. Presentation of MC9S12DJ256 microcontroller ports

Family of microcontrollers HCS12 with 120 pins has ports A, B, E, K, T, S, M, P, H, J (Figure 17). As can be seen in Table 1, Table 2 and Table 3, in the variant with 80 pins some of the terminals pin are not reflected. Thus, in this variant the ports will not include H, J [1: 0], K, M, [6: 7], P [6], S [4: 7]. The designer has available the facility to configure an input / output (I/O) port as input or output by programming the direction register (DDRx) associated. When programmed as output, the user writes data to the port. If the port is programmed as input, designer reads data from the port. Most of these ports have several other registers that can control their operations. Terminals of the ports also perform other functions in addition to the characteristic of ports as input / output, as can be traced in Table 1, Table 2 and Table 3.

Figure 17 – Ports of HCS12 (Freescale, 2002)

Port A
Terminals PA[7: 0] are general input / output terminals. Can be used for multiplexed address and data buses.
Port B
Terminals PB[7: 0] are general input / output terminals. Can be used for multiplexed address and data buses.
Port E
Terminals PE[7: 2] are general input / output terminals, but also have associated other functions.

> *Refer to pages 61 and 62 of 9S12DP256BDGV2-userguide.pdf in the References commented section . Worksheet to be completed.*

PE1 and PE0 are general input terminals, but can be used as terminals for unmasked interrupt request for asynchronous interrupts. When the port's terminals do not meet these interruptions related functions, they can be used as general input / output terminals. In this sense, for setting port's direction and for access we are provided with DDRE and PORTE registers.
Port J
This port is associated with CAN4 and IIC modules. The terminals can be used either for I / O operations or with CAN and IIC subsystems. By default PJ6 and PJ7 are shared between CAN4 and IIC. If CAN4 is enabled, terminals are terminals for CAN transmitting and receiving. If the IIC is enabled, CAN4 disabled, the terminals are open collector IIC terminals.
Port M
This port is associated with the two CAN modules and BLDC (J1850). The terminals can be used either for I / O operations, or with subsystems CAN and BLDC. By default PM0 and PM1 are shared between CAN0 and BLDC. If CAN0 is enabled, terminals are terminals for CAN transmitting and receiving. If BLDC is enabled,CAN0 disabled, terminals are BLDC transmitting and receiving terminals.
Port P
This port is associated with PWM and SPI modules. The terminals can be used either for I / O operations, or with PWM and SPI subsystems. The terminals are shared between PWM channels and SPI1 and SPI2 modules. If the PWM is enabled, terminals become PWM output channels, except pin 7 which can be input or output. If SPI1 or SPI2 are enabled and PWM disabled, terminals configuration is determined by the status bits in the SPI modules. Has several registers that are described in detail in the datasheet, among which PTP (register to initialize port P), DDRP (port direction register setup).

> *Each port is associated with different registers described in detail in the S12DP256PIMV2-portguide.pdf. Worksheet must be completed.*

Port S
This port is associated with SCI and SPI modules. The terminals can be used either for I / O operations, or with SCI and SPI subsystems.
Port T
This port is associated with Enhanced Capture Timer module. Terminals PT [7: 0] can be used either for input-output operations (I / O) or SCI and SPI subsystems.

2.4. The methodology implemented in conducting the work

The system used in this laboratory work consists of ZK-S12-B board, which on one hand, connects to the PC via USB port. On the other hand, the board will communicate with PC unit via CAN bus, using adapter USBtoCAN from Systec Electronic, that is USB-CANmodul. This adapter has an associated program called PcanView (USBCAN) through which you can send and accept various messages in hexadecimal format to / from the S12 microcontroller from ZK-S12-B board (Figure 18).

Figure 18 – System Architecture

Figure 19 – The methodology used

The microcontroller will be equipped with a program developed in such a way as to accept certain requests on the CAN bus, then make certain requirements of these requests, and finally send the results via CAN bus in the form of responses or board changes (activation of LEDs etc.). The format of these messages will be discussed in the Development of the work section .
In other words, the microcontroller is equipped with a code that can receive commands (or requests) from the PcanView via CAN bus. When such a request is received, a specific requirement should run, namely the requirement of programming to be implemented in the laboratory (eg, programming ports). Further, the microcontroller will send the results of running these requirements. According to the results received, we can test whether the programming was done correctly or not (Figure 19). The response can be observed depending on the requirement to implement: as PcanView message, as a result of board, usually in OUTPUTS section etc.

The development of a system	**Application of the process in the laboratory**
Specifications	Theme of the lab (programming ports, control of ADC, PWM signal generation, etc.)
Development	Using laboratory paper, data sheet, commented references and recommendations made, make the given theme
Testing	Testing results throught industrial way (request/response) of embedded systems development
Providing customer (Delivery)	Providing worksheets with the results obtained

Table 4 - Implementation of industrial process of developing a system in the laboratory work

This way of working is specific to industrial development of embedded systems. It can further be understood as an analogy with classic development of a system, as can be seen in Table 4, based on some specification, following development, testing what was developed, and finally, supply the system to customer.

2.5. PcanView application (USBCAN)

Start from 'quick start menu' or **Start> All Programs> USB-CAN Module Utilities> Tools> PcanView** and configure as shown in Figure 20 and Figure 21.

Figure 20 - Setup "settings" Window in the USB-CANmodul

Figure 21 - Setting window "Connect to net" under USB-CANmodul

Attention! It will only be send messages with 'extended ID' in the PcanView!

Introducing a new message is done by pressing the button called New Message.

2.6. Charging code to ZK-S12-B board

ZK-S12-B board is associated with CodeWarrior development environment, which brings together an editor, assembler, compiler, and debugger for C language and interface Freescale BDM (Background Debug Mode), which allows the download and debug procedure of the application loaded in the Flash memory of the microcontroller.

Accessing CodeWarrior environment shall be: **Start> All Programs> Softec Microsystems> SK and ZK-S12 (X) Series> CodeWarrior Development Studio**

- **To create a new application** using the CodeWarrior environment follow the steps:

1. From the main menu, select **"File> New"**.
2. A dialog window appears. Select **"HC(S)12 New Project Wizard"**.
3. Enter the project name and set the desired location (eg working directory on drive E:\Labs\S \)
4. Follow the steps provided by the Project Wizard, at microcontroller choice enter MC9S12DJ25B.

- **To launch an existing project** to make further modifications follow the following steps:

1. Ensure that the *board is connected first* (you can check this by lighting the green LED), then connected via USB to the workstation.
2. All the connectors are in position recommended (possibly original).
3. From the main menu, select "**File> Open**". Select the file ending **.mcp** from the desired location. A window will appear as that shown in Figure 22.
4. C code is in file **main.c**. By double-clicking, you can access any file in the project.

Figure 22 – CodeWarrior Window

Figure 23 – Debug Window

5. From the main menu, choose "**Project> Debug**". This will compile the source code, will generate an executable and download it to the board.
6. A debug environment opens that will be as shown in Figure 22.
7. From the main menu, choose "**Run> Start / Continue**". The program will run in real-time.
8. From the main menu, choose "**Run> Halt**". Program execution will stop. The next instruction to be executed will be highlighted in the Source window.
9. From the main menu, choose "**Run> Single Step**". Instruction highlighted in Source window will be executed, program execution will be stopped immediately thereafter.
10. From the main menu, choose "Run> Start / Continue". The application will continue to run from where it was stopped.

2.7. Carrying out the work

1. Define the work requirements

    a. Configure the ports A and P as input ports, which involves reading values from these ports. In port P case, read the values of PP0-PP3 buttons from INPUTS section of the development board. Correct programming of port P must be tested in PcanView by the change of the value from 1 to 0.

    b. Configure port B as output port, which involves writing some value to this port. Ensure the correct programming of the port by following PB0 - PB7 LEDs from OUTPUTS section of the development board.

2. Open the CodeWarrior project related the work two, namely the file with .mcp ending. The code for this work is presented in the following (main.c).

```
////////////////////////////////////////////////////////////////////////////
//
// 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 DEFAULT

void PeriphInit(void)
{

    //Configuration of PB[7..0] as output port
    //DDRB = ;
    //PORTB = ;

   // Configuration of PA[7..0] as input port
   //DDRA =;

    //Configuration of PP[7..0] as input port and activation of internal resistance
    //PTP = ;
    //DDRP = ;
    //PERP = ;
    //PPSP = ;

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

      if(msg->id==CAN_MSG_ID_CMD){
       // decoding command message
       switch(msg->data[0]){

       case 0:{ // COMMAND 00: software version
              memcpy(msg_send.data,"SI-SW1.0",8);
              msg_send.len = 8;
              break;
        }
        case 1:{ // COMMAND 01: digital read (read from 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:{ // COMMAND 02: digital write (writing to the corresponding port)
             // 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;
```

```
            }
           // maximum 8 pins
           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;
       }

       default: {// sending error message
          memcpy(msg_send.data,"bad_cmd",7);
          msg_send.len = 7;
       }
      }
     }
     else{// sending 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;
    }
   }
}

//////////////////////////////////////////
#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. Configure ports A and P as digital inputs and port B as digital output by removing comments from main.c file (see code above), PeriphInit function () and completing the corresponding values of each register (PORTB, DDRB, DDRA, DDRP, PERP, PPSP). After setting pins, send the program on the development board and run it. For programming registers use the descriptions from data sheet: S12MEBIV3-mux-extbus.pdf and S12DP256PIMV2-portguide.pdf.

4. Commands used in PcanView are (follow the code and presented comments above):

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

b. **Command 01:** command for reading a channel configured as digital input. Has the structure:

   i. Byte 0: command [01]

   ii. Byte 1: port [41 (port A), 50 (port P)]

   iii. Byte 2: pin value

Example:

01 41 01 introduced in PcanView as test message, if the programming was done correctly, should return also in PcanView, as a response to the test message, 0 or 1 depending on the value of the terminal (development phase test!)

c. **Command 02:** control for change the value in a channel configured as digital output. Has the structure:

   i. Byte 0: command [02]

   ii. Byte 1: port [42 (ie B)]

   iii. Byte 2: terminal

   iv. Byte 3: value of the terminal

Example:
02 42 06 01 inserted into PcanView as test message, if the programming was done correctly, should activate the LED 06, as value one was sent it *(development phase test!).*

5. Manually send these commands from application PcanView:

5.a ID: 00000h MSG: 00h

5.b ID: 00001h MSG: 00h

5.c ID: 00001h MSG: 03h

5.d ID: 00001h MSG: 01h

5.e ID: 00001h MSG: 01h 41h 11h

5.f ID: 00001h MSG: 01h 42h 01h

5.g ID: 00001h MSG: 01h 41h 01h

5.h ID: 00001h MSG: 01h 50h 01h

6. Send with a recurrence of 500ms the following commands from PcanView application, identifying the

value returned if pressing the 4 buttons from INPUTS section of the development board:

6.a ID: 00001h MSG: 01h 50h 01h

6.b ID: 00001h MSG: 01h 50h 02h

6.c ID: 00001h MSG: 01h 50h 06h

6.d ID: 00001h MSG: 01h 50h 07h

7. Send the following commands manually from the Pcan View application and follow the configuration on LEDs:

7.a ID: 00001h MSG: 02h

7.b ID: 00001h MSG: 02h 41h 01h 00h

7.c ID: 00001h MSG: 02h 42h 01h 01h

7.d ID: 00001h MSG: 02h 42h 02h 01h

7.e ID: 00001h MSG: 02h 42h 07h 01h

7.f ID: 00001h MSG: 02h 42h 06h 01h

7.g ID: 00001h MSG: 02h 42h 02h 00h

2.8. Abstract of the work

- Direction of different HCS12 microcontroller ports can be set by the related DDRx registers.
- Setting values for all registers shall be done as required, based on the data sheet.
- Code is developed using CodeWarrior IDE.
- Testing an implemented portion of the code is based on message system.
- Messages are entered in hexadecimal in PcanView application. Depending on the requirement, the result of running the code can be seen throughout in PcanView, as a response message in hexadecimal, in OUTPUTS section of the board or at terminals.

2.9. References commented

1. Datasheet for family HCS12: 9S12DP256BDGV2-userguide.pdf (Freescale, 2005) presents the various functions that the microcontroller terminals may have.
2. Datasheet of microcontroller related ports: S12DP256PIMV2-portguide.pdf (Freescale, Port Integration Module, 2002) presents a description of the various port registers.
3. Datasheet related to external bus multiplexing: S12MEBIV3-mux-extbus.pdf (Freescale Multiplexed

External Bus Interface, 2003) presents a description of the various port registers.

4. Deepening HCS12 microcontroller ports may be done by referring to Sections 4.10, 7.5, 7.6, 7.9 in (Huang, 2010).

5. Compulsory consultation (Popa, 2011).

2.10. Worksheet

Enter you name:……

Semigroup:……..

Section 2.7.5. Contents of registerd

PORTB:	DDRB:	DDRA:
DDRP:	PERP:	PPSP:

Section 2.7.6. Messages received in PcanView application

5.a ID:	MSG:
5.b ID:	MSG:
5.c ID:	MSG:
5.d ID:	MSG:
5.e ID:	MSG:
5.f ID:	MSG:
5.g ID:	MSG:
5.h ID:	MSG:

6.a ID:	MSG:
6.b ID:	MSG:
6.c ID:	MSG:
6.d ID:	MSG:

Section 2.7.7. LEDs configuration

7.a

7.b

7.c

7.d

7.e

7.f

7.g

After consulting the documentation, write to the port E other functions (see first box in section 2.3).

_____

_____

_____

_____

After consulting the documentation, write which are the common functions found at different ports, carried out different registries. What registers are specific to certain ports? What are these? (see the second box in section 2.3).

_____

_____

_____

_____

_____

_____

_____

Observations