5. Interruption system of MC9S12DJ256 microcontroller

5.1. Purpose

Developing skills relating to the interruption system of MC9S12DJ256 microcontroller.

At the final of this lab are necessary the following abilities:

- Programming of microcontroller system interrupting
- Testing of proposed application using the methodology presented in Chapter 2 customized for this application.

5.2. Interruption Mode

An interrupt is a special event that requires the CPU to temporarily stop normal operation and treat this event.. Examples of interruption are events such as "divide-by-0", "arithmetic overflow", the expiry of a timer and so on. For most systems, an interrupt is caused by an external event or a poor functioning of the system. Generally, interrupts are prioritized so that the most important to be served first. There are different interrupt sources:

- external: terminal or external level causes interruption;
- internal: internal peripheral equipment of microprocessor or his processor causes interruption;
- software: instruction in assembly language *swi* causes interruption.

Those interruptions that can not be ignored by the processor are called *unmasked interrupts*. Typically, such an interrupt has the highest priority. Interruptions can be ignored by the processor are called *masked interrupts*. An interrupt mask must be activated before you can interrupt the processor and can not be disabled when the CPU interrupt is not possible. An interrupt is enabled by setting bit type enable / disable or a flag.

To treat a interruption, the processor executes program called Interrupt Service Routine (ISR). A full course of treatment interruption includes the following steps:

1. saving current program counter value (PC) on the stack;
2. saving processor state (including the processor status register and other registers) on the stack;

3. identify the cause of interruption;
4. establish the beginning address of the interrupt handling routine (ISR);
5. execution of the interrupt handler;
6. restoring the processor state and the program counter on the stack;
7. return to the interrupted program.

An interrupt handler often presents and it works as another subroutine or method, just that it will be called due to an interruption treat by system. Starting address of the interrupt handler is called *interrupt vector*. *Interrupt vector table* is a table where is stored all interrupt vectors. For HCS12 microcontroller, interrupt vectors are preset according to predefined memory addresses (see the section Conducting Work).

Following steps are required to program an interruption:

1. initialize interrupt vector table;
2. the actual programming of the interrupt handler;;
3. enabled desired interrupt.

5.2.1. Case of microcontroller interrupt module MC9S12DJ256

80-pin version of the HCS12 microcontroller has a dedicated sub-module for interrupts. Its block diagram is shown in Figure 31. The sub-module interrupt (named INT) role is to decode all requests for exceptions priority system (interruptions) and provide corresponding vector processing exceptions.

INT module has the following features:

- provides 2 to 122 masked interrupt vectors I-bit ($ FF00- $ FFF2).
- provides an masked interrupt vector X-bit ($ FFF4)
- provides a software interrupt unmasked (SWI) or vector for Background Debug Mode request ($ FFF6).

Figure 1 - Block diagram of interruption module (Freescale, 2003)

• provide a vector for Unimplemented Opcode Trap (TRAP) unmasked ($ FFF8)

• provides 3 vectors for System reset ($ FFFA- $ FFFE) (Reset, CMR and COP)

• determine the appropriate vector and provide it on address bus

• announces CPU that exist waiting interrupts.

• provides control registers that allow interruption testing.

• provides additional input signals which prevent interrupt request treating type I and X

• awakens the system from stop or wait mode when an interrupt occurs or when appropriate / XIRQ is active, even if / XIRQ is masked

• (Optional) select and retain the highest priority interrupt according to the value written in the register HPRIO.

Interrupt module processing so all interrupt requests made by processor. These exceptions include requests of interrupt vectors (vector interrupt requests) and requests of reset vector (reset vector requests).

Vector address	Sursă
$FFFE–$FFFF	System reset
$FFFC–$FFFD	Crystal monitor reset
$FFFA–$FFFB	COP reset
$FFF8–$FFF9	Unimplemented opcode trap
$FFF6–$FFF7	Software interrupt instruction (SWI) or BDM vector request
$FFF4–$FFF5	$\overline{\text{XIRQ}}$ Signal
$FFF2–$FFF3	$\overline{\text{IRQ}}$ Signal
$FFF0–$FF00	Device-specific I-bit maskable interrupt sources (priority in descending order)

Table 1 – Vector Interrupt Address, with associate priority

Interrupt Vector Requests

As shown in Figure 31, INT module contains a block of registers that provide control and monitoring of interrupt, optionally block Highest Priority Interrupt (HPRIO) and a priority decoder to assess whether pending interrupts are valid, in affirmative case to allocate priority.

INT module registers are described in detail in (Freescale, Interrupt Module, 2003) (see References Commented!). If HPRIO block is implemented, the user is allowed to select an masked interrupt I -bit like having the highest priority.

Priority decoder evaluates all pending interrupts and determines the validity and priority. When the CPU requests an interrupt vector, the decoder will provide the vector for the interrupt request with the highest priority. If the processor requests an interrupt vector even if there was no pending interrupt request, block INT will provide the address SWI vector (Software Interrupt).

Reset Vector Requests

INT module supports 3 types of requests of reset vectors: *normal system reset* (power-on-reset request), request Crystal Monitor reset and request COP watchdog reset . The INT module will provide interrupt vector address for the type of reset required.

The priority (from highest to lowest), and the addresses of all interrupt vectors provided by INT at CPU request are shown in Table 8.

5.3. Conducting work

1. Defining the requirements of the work
    a. Count of interruption introduce by operating buttons, PP0,PP1, PP2 and PP03 from INPUTS .
2. Open the CodeWarrior project related in work number 5 , namely file ending .mcp. This essential code of this work is presented below.

```
void PeriphInit(void)
{

  //Configure PB[7..0] as output pot
   PORTB = 0x00;
   DDRB = 0xFF;

   // Configure interrupt PP[7..0]
   PTP = 0x00;
   DDRP = 0xF0;

   PERP = 0x0F;
   PPSP = 0x00;
   PIEP = 0x0F;
```

```
   PIFP = 0x0F;

   // Configure peripheral ADC
   #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;
}
```

Code for the interrupt handler is as follows:

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void PortPISR(void){
unsigned char ch=0,chm=1;
 struct can_msg msg_send;
 DisableInterrupts;
 while(ch<8){
                //interrupt triggered on Port P
                while (ch<8&&!(PIFP&chm)) ch++,chm<<=1;
                 if(ch<8){
                                msg_send.id = (unsigned int)0x000F50|ch;
                                msg_send.len = 1;
                                msg_send.data[0]=portP_evt;
                                SendCANMessage(0, msg_send);
                                PIFP=chm;ch++,chm<<=1;
                        }

               }
  PORTB=portP_evt;
  portP_evt++;
  EnableInterrupts;
}
#pragma CODE_SEG DEFAULT
```

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

3. Analysis suggested configurations. Emphasis will be on understanding how the register were configure in PeriphInit function () in C code associated of this work, the related part of interruptions.

Use specification of Interrupt module INT, S12INTV1-int.pdf and afferent documentation of ports and configuration HCS12: S12DP256PIMV2-portguide.pdf, 9S12DP256BDGV2-userguide.pdf, ZK-S12-b_schematic.pdf.

a) DDRP = 0xF0;

b) PERP = 0x0F;

c) PPSP = 0x00;

d) PIEP = 0x0F;

e) PIFP = 0x0F.

4. In this work, except the command 0, only messages are received in application PcanView (USB2CAN).
5. Compiles CodeWarrior project and send it to microcontroller. Run code written in microcontroller Flash memory.
6. Press for about one second PP0, PP1, PP2 and PP3 buttons from INPUTS section on the development board in the next sequence and check incoming messages in PcanView and LED configuration:

a) Send CAN command with extended ID: 001h Message: 00

b) Press for one second PP3

c) Press for one second PP3

d) Press for one second PP1

e) Press for one second PP2

f) Press for one second PP0

g) Press for one second PP0

h) Press for one second PP1

5.4. Summary

- An interrupt is a special event that requires the processor to stop normal operation and treat this event.
- An interrupt handler often presents and it works as another subroutine or method, just that it will be called due to an interruption treat by system.
- Testing the correct programming of code for the module INT is performed by counting interruptions introduced by operating buttons PP0, PP1, PP2 and PP03 from INPUTS section.

5.5. Commented references

- Related Data Sheet to HCS12 family 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).
- Deep read about INT module of HCS12 microcontroller can be done consulting the sections from Huang - Chapter 6 (Huang, 2010).
- Mandatory consultation (Popa, 2011).

5.6. Worksheet

Name, Surname:……

Semigroup:……..

1. Write the types of know interruptions …….....…
…….....…….

2. What is the significance of 3.2 Configuration :

a) DDRP:... ……

b) PERP:……

c) PPSP:…….

d) PIEP:……..

e) PIFP:.....……

3.Which is the address of interrupt vector associated with analog digital converter 1 at S12?

WARNING, do not confuse it with ADC0! **R:** 0x…….....

4. Wich is the purpose and content of PortPISR function () from main.c ?

```
interrupt void PortPISR(void)
{unsigned char ch=0,chm=1;
struct can_msg msg_send;
DisableInterrupts;
while(ch<8){
        while (ch<8&&!(PIFP&chm)) ch++,chm<<=1;
         if(ch<8){
                msg_send.id = (unsigned int)0x000F50|ch;
                msg_send.len = 1;
                 msg_send.data[0]=portP_evt;
                PIFP=chm;ch++,chm<<=1;
                SendCANMessage(0, msg_send);
         } // if
} // while
PORTB=portP_evt;
portP_evt++;
EnableInterrupts;

}
```

5. What messages have been received on the CAN and which is the LEDs configuration from 5.3.6 *

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]
* Tick with X the heated and with - the off Led.