**İ.T.Ü.**

**Faculty of Computer and Informatics**

**Computer Engineering**



**MICROCOMPUTER LAB**

**REPORT**

|  |  |  |
| --- | --- | --- |
| **Lab No** | **:** 06 | |
| **Lab Date** | **:** 31.10.2013 | |
| **Group** | **:** B09 | |
| **Group Members** | **:** 040100014 Teoman Turan  040100018 Mustafa Durmuş  040100117 Tuğrul Yatağan  040100124 Emre Gökrem | |
|  |  | |
|  |  | |
|  |  | |
| **Research Assistant :** Ahmet Aycan Atak | | |  |
|  | |  |

**1. THE AIM/CONTENT of THE EXPERIMENT**

The purpose of doing this experiment is to take information about external interrupt mechanism of CSM12C32, and to learn differences between polling and interrupt methods.

**2. EQUIPMENT**

To write and compile the code on CodeWarrior software, one of the laboratory’s desktop computers has been used. CSM12CS32 Kit has been connected to that computer. After the compilation of the code, its result has been observed on the kit.

**3. EXPERIMENTS**

**3.1. Section 1: Introduction**

The difference between polling and interrupt methods: Polling method does not let the processor execute any process until the signal has come, and continuously checks if the signal has been received. For a too long program, this method might not work as the processor can miss the signal while dealing with that program. On the other hand, interrupt method does not check the signal. The processor immediately jumps to the interrupt subroutine when the signal comes, then continues its actual execution from the point before the jump.

**3.2. Section 2: Interrupt**

Interrupt working mechanism can be implemented in C/C++ like the following code fragment:

while(1)

{

// Some programs could be here and its size do not affect anything.

}

void ISR(void)

{

// When interrupt occurs, processor executes this function.

// Your Program

}

When needed, the interrupt function, ISR, is called in the mart part of the program. The function is executed, then the program leaves the scope of the function, and returns to where it left off.

**3.3. Section 3: Experiment**

**3.3.1. Interrupt**

Initially, LED1 is on whereas LED2 being off. These LED states are inverted when SW1 is pressed. After releasing SW1, current LED states do not change.

The program for this mechanism, which initializes interrupt on SW1:

; export symbols

XDEF Entry ; export 'Entry' symbol

ABSENTRY Entry ; for absolute assembly mark this as application entry point

; include derivative specific macros

INCLUDE 'mc9s12c32.inc'

ROMStart EQU $4000 ; absolute address to place my code/constant data

; variable/data section

ORG RAMStart

Counter1 DC.W $FFFF

Counter2 DC.W $0010

temp DC.B $01

; code section

ORG ROMStart

Entry:

LDS #RAMEnd+1 ; initialize the stack pointer

ANDCC #$BF ; Enable XIRQ interrupt on SW1

; These blue codes are wanted.

; Here is our main program.

LDAA #$01

STAA DDRA ; PORTA.0 is connected to LED1, and it becomes output.

LDAA #$10

STAA DDRB ; PORTB.4 is connected to LED2, and it becomes output

LDAA #$01

STAA PORTA ; LED1 is on as PORTA.0 is 1.

LDAA #$10

STAA PORTB ; LED2 is off as PORTB.4 is 1.

loop1:

BRA loop2

; And here is our interrupt program.

ISR:

LDAA PORTA

CMPA #$00

BEQ BT1

BRA BT2

BT1:

LDAA #$01

STAA PORTA

LDAA #$00

STAA PORTB

BRA loop2

BT2:

LDAA #$00

STAA PORTA

LDAA #$10

STAA PORTB

loop2:

JSR delayS

RTI

; Here are our delay subroutines.

delayS: LDY Counter2

delaySloop: JSR delayMs

DEY

BNE delaySloop

RTS

delayMs: LDX Counter1

delayMsloop: DEX

BNE delayMsloop

RTS

ORG $FFF4 ; XIRQ interrupt vector

DC.W ISR ; Beginning address of ISR

**3.3.2. Polling**

With the same specifications in the previous experiment, polling method has been carried out here instead of interrupt. The program for this mechanism:

; export symbols

XDEF Entry ; export 'Entry' symbol

ABSENTRY Entry ; for absolute assembly mark this as application

; include derivative specific macros

INCLUDE 'mc9s12c32.inc'

ROMStart EQU $4000 ; absolute address to place my code/constant data

; variable/data section

ORG RAMStart

Counter1 DC.W $FFFF

Counter2 DC.W $0010

temp DC.B $01

; code section

ORG ROMStart

Entry:

LDS #RAMEnd+1 ; initialize the stack pointer

; ANDCC #$BF ; Enable XIRQ interrupt on SW1

; These blue codes are wanted.

LDAA #$01

STAA DDRA ; PORTA.0 is connected to LED1, and it becomes output.

LDAA #$10

STAA DDRB ; PORTB.4 is connected to LED2, and it becomes output.

LDAA #$00

STAA DDRE

LDAA #$01

STAA PORTA ; LED1 is on as PORTA.0 is 1.

LDAA #$10

STAA PORTB ; LED2 is off as PORTB.4 is 1.

LDAB #$10

loop:

LDAA PORTE

ANDA #$01

CMPA #$01

BEQ skip

COMB

STAB PORTA

STAB PORTB

BRA hold

skip:

BRA loop

hold:

JSR delayS

LDAA PORTE

ANDA #$01

CMPA #$01

BEQ loop

BRA hold

delayS: LDY Counter2

delaySloop: JSR delayMs

DEY

BNE delaySloop

RTS

delayMs: LDX Counter1

delayMsloop: DEX

BNE delayMsloop

RTS