



INSTITUTO DE
INGENIERÍA
ELÉCTRICA



FACULTAD DE
INGENIERÍA
UDELAR



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Datos compartidos: alternativas a deshabilitar interrupciones

Sistemas embebidos para tiempo real

Este material didáctico fue elaborado por docentes del Departamento de Electrónica de la Universidad de la República a lo largo a varios años. Se pone a disposición de la comunidad bajo la licencia “Creative Commons Attribution 4.0 International License”.

Ver detalles de la licencia aquí: <https://creativecommons.org/licenses/by/4.0/>



Co-funded by the
Erasmus+ Programme
of the European Union

Agenda

- Primera clase
 - Fundamentos de las Interrupciones
 - Problema de datos compartidos
 - Latencia en las interrupciones
- Segunda clase
 - Soluciones al problema de los datos compartidos

Índice

- Repaso: Problema de los datos compartidos
- Alternativas a deshabilitar interrupciones
 - Lecturas sucesivas
 - Doble buffer
 - Cola circular de datos

Problema de los datos compartidos

- Cuándo está potencialmente presente
 - Cuando se comparten datos entre ISR y el *main*
- Por qué se comparten datos
 - Las ISR deben hacer el trabajo estrictamente necesario para atender el hardware
 - En el *loop principal* se hace el resto del trabajo
- Cuándo surge y cuál es el problema
 - Cuando la ISR se ejecuta en el instante “inesperado”
 - Se produce inconsistencia de datos

control-obvio.c

```
static int iTemperatures[2];

#pragma vector=TIMERX
__interrupt void vReadTemperatures(void)
{
    iTemperatures[0] = !! lee valor desde hardware
    iTemperatures[1] = !! lee valor desde hardware
}

int iTemp0;
int iTemp1;

void main (void)
{
    while(TRUE)
    {
        iTemp0 = iTemperatures[0];
        iTemp1 = iTemperatures[1];
        if (iTemp0 != iTemp1)
            !! Se activa una alarma muy molesta;
    }
}
```

solucion.c

```
static int iTemperatures[2];

#pragma vector=TIMERX
__interrupt void vReadTemperatures(void) {
    iTemperatures[0] = !! lee valor desde hardware
    iTemperatures[1] = !! lee valor desde hardware
}

int iTemp0;
int iTemp1;

void main (void) {
    while (TRUE) {
        __disable_interrupt();
        iTemp0 = iTemperatures[0];
        iTemp1 = iTemperatures[1];
        __enable_interrupt();

        if (iTemp0 != iTemp1)
            !! Se activa una alarma muy molesta;
    }
}
```

solucion.c

```
static int iTemperatures[2];

#pragma vector=TIMERX
__interrupt void vReadTemperatures(void) {
    iTemperatures[0] = !! lee valor desde hardware
    iTemperatures[1] = !! lee valor desde hardware
}

int iTemp0;
int iTemp1;

void main (void) {
    while(TRUE) {
        short s = __get_interrupt_state();
        __disable_interrupt();
        iTemp0 = iTemperatures[0];
        iTemp1 = iTemperatures[1];
        __set_interrupt_state(s);

        if (iTemp0 != iTemp1)
            !! Se activa una alarma muy molesta;
    }
}
```

timer.c

```
static int iSeconds, iMinutes, iHours;

#pragma vector=TIMERX
__interrupt void vUpdateTime(void) {
    // Hacer lo que haya que hacer con el hardware
    ++iSeconds;
    if (iSeconds >= 60){
        iSeconds = 0;
        ++iMinutes;
        if (iMinutes >= 60){
            iMinutes = 0;
            ++iHours;
            if (iHours >= 24)
                iHours = 0;
        }
    }
}

long iSecondsSinceMidnight(void) {
    return ( ((iHours*60) + iMinutes) *60) + iSeconds );
}
```


timer.c

```
static int iSeconds, iMinutes, iHours;

#pragma vector=TIMERX
__interrupt void vUpdateTime(void) {
    // Hacer lo que haya que hace con el hardware
    ++iSeconds;
    if (iSeconds >= 60){
        iSeconds = 0;
        ++iMinutes;
        if (iMinutes >= 60)
        {
            iMinutes = 0;
            ++iHours;
            if (iHours >= 24)
                iHours = 0;
        }
    }
}

long iSecondsSinceMidnight(void){
    long lRetrunVal;
    __disable_interrupt();
    lRetrunVal = (((iHours*60) + iMinutes) *60) + iSeconds;
    __enable_interrupt();
    return lReturnVal;
}
```

timer2.c

```
static long int lSecondsToday;

#pragma vector=TIMERX
__interrupt void vUpdateTime (void)
{
    ...
    ++lSecondsToday;
    if (lSecondsToday == 60 * 60 * 24)
        lSecondsToday = 0;
    ...
}

long lSecondsSinceMidnight (void)
{
    return lSecondsToday;
}
```

Alternativas a deshabilitar interrupciones

- Lecturas sucesivas
- Doble buffer
- Cola circular de datos

Lecturas sucesivas

- Idea:
 - leer repetidamente la variable y una copia hasta que sean iguales.
- Problema:
 - Optimización del compilador:
 - porque no intervienen escrituras entre las dos lecturas.
- Solución:
 - usar la palabra reservada “volatile”
- Desventajas & limitaciones:
 - ahora veremos...

lecturas-sucesivas.c

```
static volatile long int iSecondsToday;

void interrupt vUpdateTime (void)
{
    ...
    ++lSecondsToday;
    if (lSecondsToday == 60 * 60 * 24)
        lSecondsToday = 0;
    ...
}

long lSecondsSinceMidnight (void)
{
    volatile long lReturn;
    lReturn = lSecondsToday;
    while (lReturn != lSecondsToday)
        lReturn = lSecondsToday;
    return lReturn;
}
```

Doble buffer

- Idea: usar doble buffer + bandera global
 - bandera global
 - cambiada por el código de la tarea
 - indica cual de los buffer utilizar
 - cada buffer es accedido/modificado en conjunto
- Funcionamiento:
 - La bandera global se alterna (cambia) en el main (al final).
 - En particular no cambia mientras se leen las variables del main.
 - La ISR siempre va a escribir en el buffer no usado.
- Desventajas:
 - ahora veremos...

```
static int iTemperaturesA[2], iTemperaturesB[2];
static bool fTaskCodeUsingTempsB = false;

void interrupt vReadTemperatures (void)
{
    if (fTaskCodeUsingTempsB) {
        iTemperaturesA[0] = !! read in value from HW
        iTemperaturesA[1] = !! read in value from HW
    }
    else {
        iTemperaturesB[0] = !! read in value from HW
        iTemperaturesB[1] = !! read in value from HW
    }
}

void main (void) {
    while (true) {
        if (fTaskCodeUsingTempsB)
            if (iTemperaturesB[0] != iTemperaturesB[1])
                !! Set off howling alarm;
            else
                if (iTemperaturesA[0] != iTemperaturesA[1])
                    !! Set off howling alarm;
            fTaskCodeUsingTempsB = !fTaskCodeUsingTempsB;
    }
}
```

Actividad en grupo

- Implementación: Cola circular
 - Objetivo:
 - Definir interfaz pública de la cola.
 - Implementar la misma (papel).
 - Grupos:
 - 2 a 4 participantes
 - Tiempo:
 - 5 minutos para definir la interfaz
 - 15 minutos para implementar el módulo
 - Puesta en común.

Cola circular

- Idea: se comparten datos a través de una cola circular
 - ISR obtiene datos y los agrega a la cola
 - Tarea (main) extrae datos de la cola y los procesa
 - En este caso se agregan de a pares: itemp1 e itemp2
- Manejo de la cola:
 - Queue full: $\text{head} + 2 = \text{tail}$ (2 lugares usados por vez)
 - Queue empty: $\text{head} = \text{tail}$
- Ventajas:
 - Cola desacopla velocidad de llegada y procesamiento de datos
- Diseño del tamaño de la cola:
 - Velocidad de procesado = tan grande como la tasa de llegada de datos.
- Desventajas:
 - ahora veremos...

Cola circular

- Desventajas:
 - Tarea debe leer el dato primero y modificar el puntero después.
 - Invertir la operación podría llevar a sobre-escribir datos antes de ser escritos.
 - Cuando se incrementada el puntero del final (tail), la escritura tiene que ser atómica.
 - Sino el “consumidor” y el “productor” pueden ver diferente la cola.
 - Depende del tamaño de la cola (tamaño del índice) y del procesador
Problema: tamaño > 256 y procesador 8-bits.
- Recordar
 - Buscábamos alternativas a deshabilitar interrupciones.

Actividad en grupo

- Cola circular: alternativa a deshabilitar interrupciones:
 - Objetivo:
 - Utilizar el módulo de cola ya implementado
 - Escribir un pseudocódigo que use una cola para “producir” (ISR) y “consumir” datos (main).
 - Grupos:
 - 2 a 4 participantes
 - Tiempo:
 - 10 minutos para definir la interfaz
 - Puesta en común.

cola-circular.c

```
#include "queue_int2.h" // cola de enteros (maneja de a dos datos)

int itemps[2]; // el arreglo guarda

void interrupt vReadTemperatures (void) {
    if ( !q_isfull() ) {
        !! lee los dos sensores de temperatura desde hw
        q_add(...); // agrega datos a la cola
    } else
        !! la cola está llena, descarta dato
}

void main (void) {
    q_init();
    while (TRUE) {
        if ( !q_isempty() ) {
            itemps = q_get();
            !! Compara valores y si corresponde da alarma
        }
    }
}
```

cola-circular.c

```
#define Q_SIZE 100
int iTemperatureQ[Q_SIZE];
int iHead = 0;
int iTail = 0;

void interrupt vReadTemperatures (void) {
    if ( !(( ihead+2==iTail) || (iHead== Q_SIZE-2 && iTail==0)) ) {
        iTemperatureQ[iHead] = !! read one temperature
        iTemperatureQ[iHead+1] = !! read other temperature
        iHead += 2;
        if (iHead== Q_SIZE)
            iHead = 0;
    } else !! throw away next value
} //end-interrupt

void main (void) {
    int iTemp1, iTemp2;

    while (TRUE) {
        if (iTail != iHead) {
            iTemp1 = iTemperatureQ[iTail];
            iTemp2 = iTemperatureQ[iTail+1];
            iTail += 2;
            if (iTail == Q_SIZE)
                iTail = 0;
            !! Compare values
        } //end-iff
    } //end-while
} //end-main
```

Bibliografía

- “An Embedded Software Primer” David E. Simon
 - Chapter 4: Interrupts
- MSP430 Optimizing C/C++ Compiler v18.1.0 LTS. User's Guide
- "A Firmware Development Standard", Jack G. Ganssle, Version 1.4, Updated May, 2007.