INSTITUTO DE INGENIERÍA ELÉCTRICA

FACULTAD DE INGENIERÍA UDELAR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Test de software embebido

Sistemas embebidos para tiempo real

NEON
Network of Competence on Internet of Things

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

Objetivos

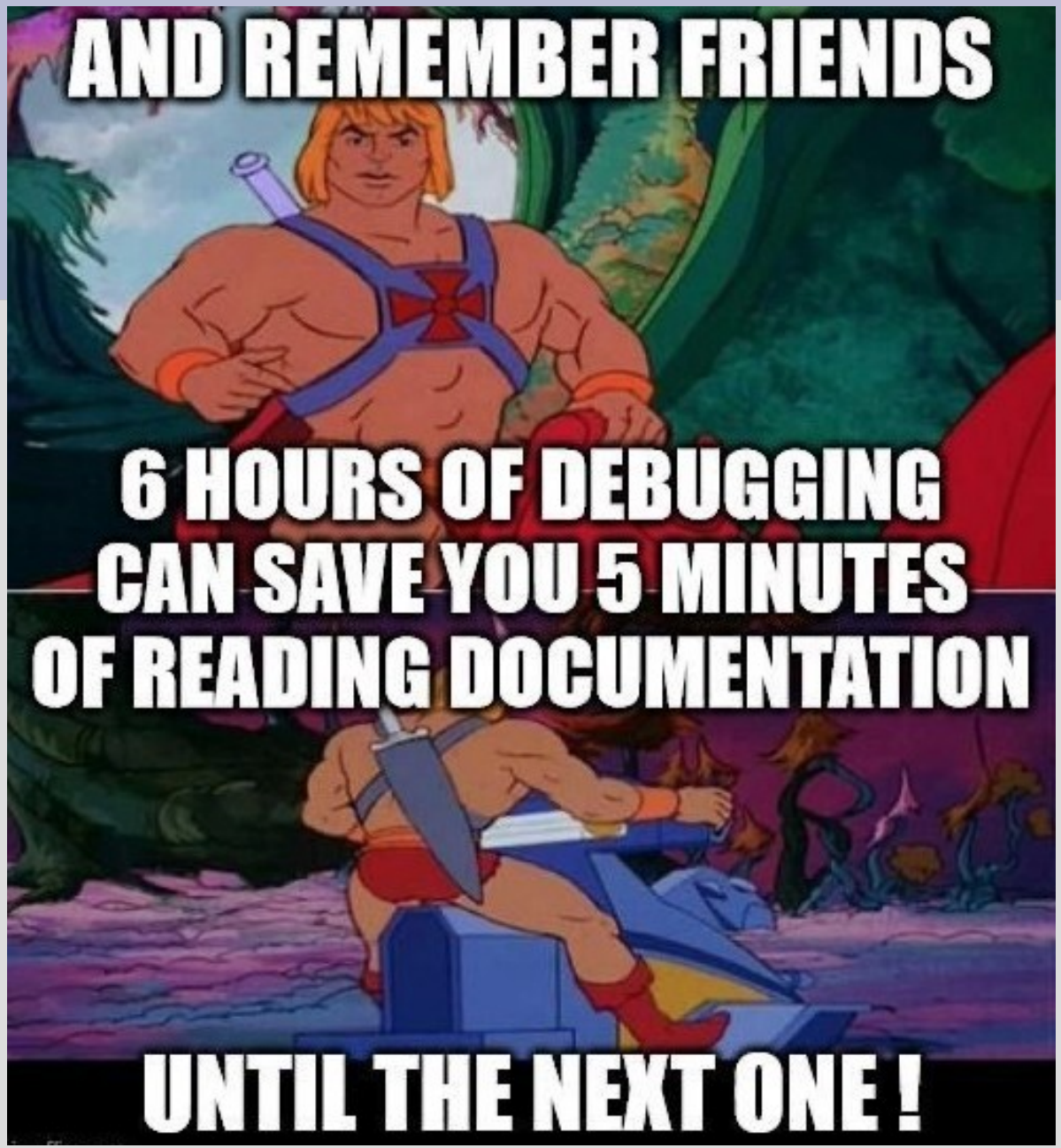
- Definir los principales conceptos de test
- Identificar las diferencias de test en PC
- Explicar los métodos de test embebido: en host y con simuladores.
- Separar código dependiente e independiente de hardware
- Reconocer los beneficios del “assert”

Índice

- Test de software embebido
  - Conceptos generales
  - Tipos de test
- Técnicas de depuración/debug
  - Test en *host*
  - Simuladores
  - Aserciones

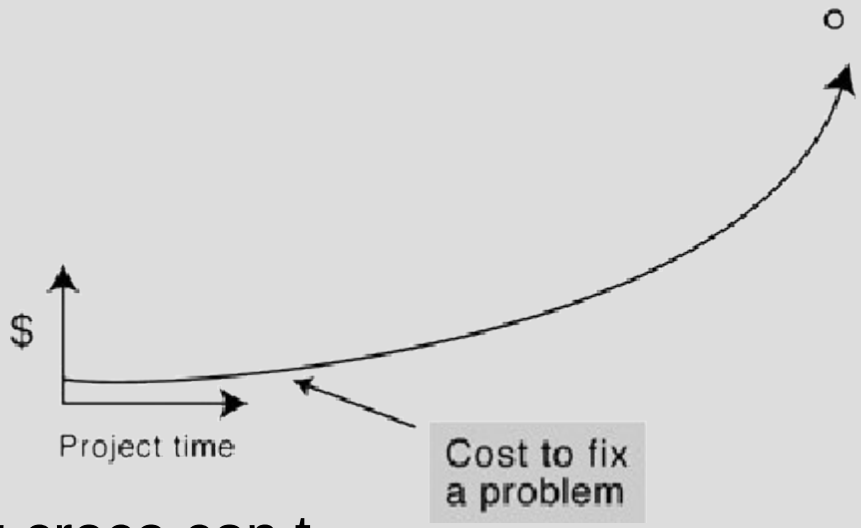
Consideraciones previas

- Test vs debug
- Embebidos vs. aplicaciones de PC
  - Técnicas base de testing son las mismas
  - Diferencias:
    - es “más difícil” el testing embebido
    - ejecutar sin caerse por largos períodos de tiempo
    - aplicaciones de soporte de vida
    - usuarios: ¿quiénes son?


AND REMEMBER FRIENDS
6 HOURS OF DEBUGGING
CAN SAVE YOU 5 MINUTES
OF READING DOCUMENTATION
UNTIL THE NEXT ONE !

Conceptos básicos

- Test:
  - Proceso cuyo objetivo es encontrar defectos
  - Destructivo: se busca demostrar que el programa no funciona.
- ¿Para qué?
  - Aumentar la calidad
  - Reducir costos
  - Mejorar performance
- ¿Cuándo?
  - Cuanto antes:
  - el costo de arreglar un bug crece con t



Conceptos básicos

- Test de unidad
  - Nivel de módulo: escribir código que sustituye el resto del sistema (hw/sw)
  - Desarrollar tests reusables y repetibles
  - Guardar los resultados de los test ("audit trail")
- Regression Testing
  - No es suficiente pasar el test una vez
  - Repetir el test cada vez que se modifica el programa

Conceptos básicos

- ¿Qué testear?
  - Diseño del caso de test: seleccionar el conjunto de test apropiados.
- Clasificación
  - Test funcional (black-box)
    - Verifica se cumplan con los requerimientos funcionales
    - Puede ser desarrollado luego de los requerimientos
  - Test de cobertura (white-box)
    - Busca ejecutar (ejercitar) ciertas (todas?) partes del código

Conceptos básicos

- ¿Cuándo parar de testear?
  - Criterios aceptados
    - alguien lo diga (jefe)
    - nueva iteración de test encuentra menos que X cantidad de bugs
    - cierto grado de cobertura se alcanza sin descubrir nuevos bugs

Test funcional (black box)

- Stress test
  - Sobrecargar: entradas canales de comunicación, buffers, etc.
- Boundary value test
  - Entradas y salidas: dentro de cierto rango
  - Ejemplo: actualización de hora:minuto:segundo
- Excepcion test:
  - Disparar modos de fallo o de excepción
  - Ejemplo: intentar agregar algo a un cola llena
- Error guessing:
  - Basado en experiencia previa
- Random test
  - No muy productivo pero usado.
- Performance test
  - Si el rendimiento es parte de los requerimientos.

Test de cobertura (white box)

- Statement coverage
  - Ejecutar cada sentencia al menos una vez
  - IAR provee herramienta para esto
- Decision or branch coverage
  - Ejecutar cada bifurcación al menos una vez (true y false)
- Condition coverage
  - Forzar cada condición (termino) para que tome todos los valores lógico posibles

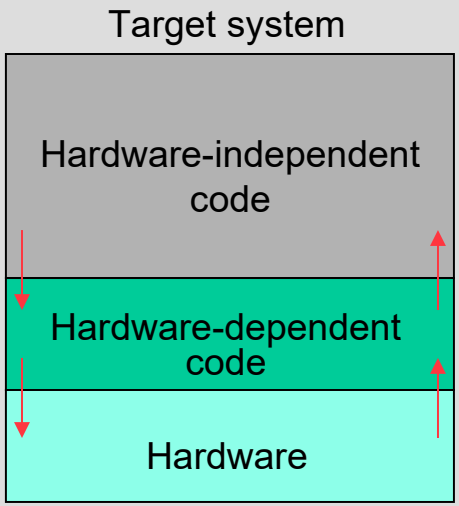
Test de SW embebido: problemas

- Plataforma hardware al escribir código
  - no está disponible o estable
- En general es difícil:
  - generar escenarios de prueba
  - testear todas las combinaciones (sino imposible)
  - saber cuales combinaciones causarán un problema de repetir
  - registrar eventos (logging) para averiguar una falla

Test en host

- Test mediante **ejecución** en *host*
- Es conveniente realizarlo
  - pero no todo va a funcionar.
- El temporizado no siempre es el mismo
  - difícil para detectar algunos tipos de bugs
  - bugs de datos compartidos

Método básico para test en host



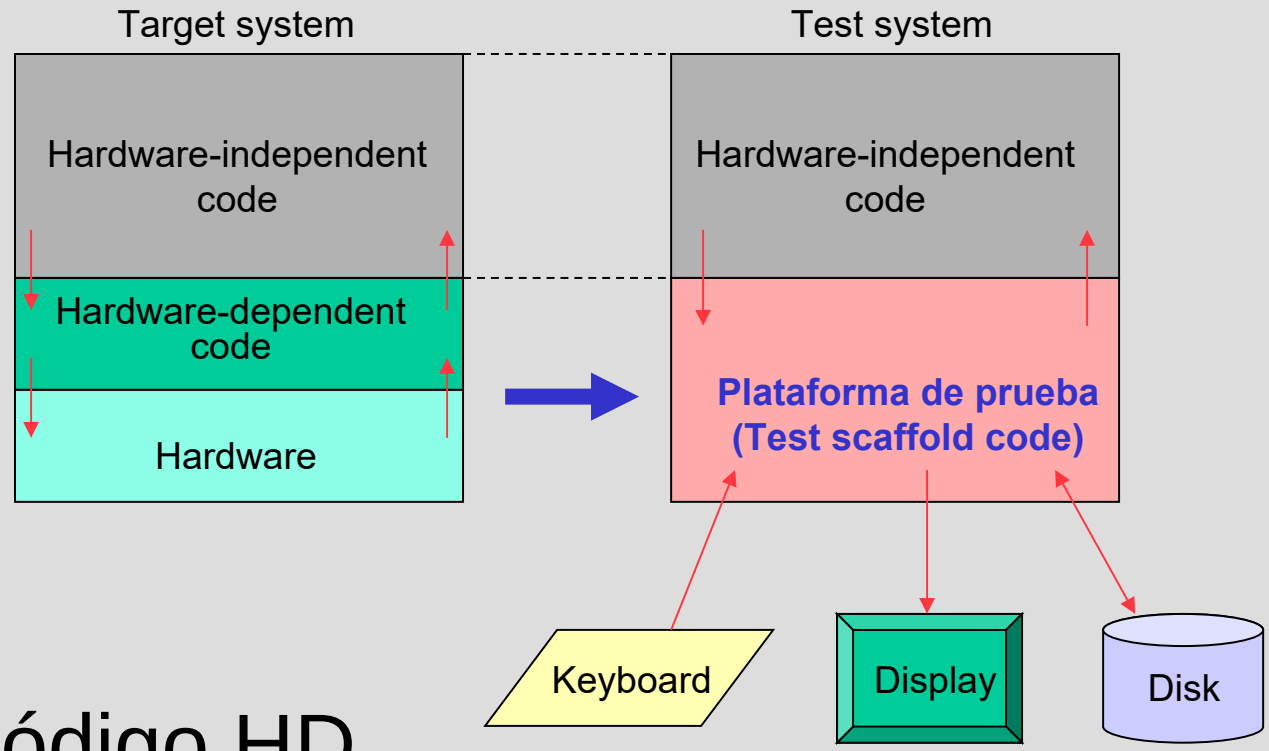
Extraído de: J. Archibald, "Real-Time and Embedded Systems", slides: set8.ppt

- Dividir el código de aplicación:
  - HD: código dependiente del HW
  - HI: código independiente del HW
- HD es código *especial* que depende del μC
- HI es código que compila y corre en PC

- **Fundamental definir claramente la interfaz**

Plataforma de prueba

Extraído de: J. Archibald, “Real-Time and Embedded Systems”, slides: set8.ppt



- Sustituye código HD
- Misma interfaz que código HD

Ejemplo

- Caso de estudio: RTC (reloj de tiempo real) de lab1 y lab2
- Objetivo: ver en un ejemplo separación de código en HI y HD.
- Actividad:
  - Partiendo de main donde el código HI y HD está entreverado, separarlo entre HI y HD definiendo la interfaz, escribiendo timer_MSP430.h y timer_MSP430.c
  - ¿Por qué en el lab2 hablamos de timer_hw?

main_lab2_hd.c

```
#include "timer.h"
#include "msp430.h"

void init_timerA();
void init_clock();

void main (void){
    time_t hora_Actual={23,59,59,750};
    set_time(hora_Actual);           // Inicializo hora
    init_clock();                    // Inicializo Basic Clock Module
    init_timerA();                   // Inicializo configuraciones del timer
    __enable_interrupt();
    while (1){
          get_time(&horaActual);
         }
}

void init_clock(){          // Configura el reloj
}

void init_timerA(){         // Configuro el timer
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void ISR_timerA0(void){
          inc_time();
}
```

timer.h (Lab1)

```
typedef struct{
        int hour;
        int minute;
        int second;
        int millisec;
} time_t;

void inc_time();
void set_time(time_t t_ini);
void get_time(time_t* t_ret);
```

timer.c (Lab1)

```
static time_t actual_time;
void inc_time(){
     if (actual_time.millisec < 750){
         actual_time.millisec += 250;
     }else{
         actual_time.millisec = actual_time.millisec - 750;
         if (actual_time.second < 59){
         … // Actualiza segundos, minutos y horas
}
void set_time(time_t t_ini){
     actual_time = t_ini;
}
void get_time(time_t* t_ret){
     *t_ret = actual_time;
}
```

timer_MSP430.h

```
#include "msp430.h"
void init_timerA();
void init_clock();
```

timer_MSP430.c

```
void init_clock(){
    //ACLK
    BCSCTL1 |= DIVA_0;                  // ACLK/1 (DIVA_3 = /8)
    BCSCTL3 |= XCAP_3;                  //12.5pF cap- setting for 32768Hz crystal
}

void init_timerA(){
   // Configuro contador
   TA0CCTL0 = CCIE;                     // Habilito interrupciones del contador CCR0
   TA0CCR0 = 1023;                      // Hasta cuanto tiene que contar
   // Configuro el timer
   TA0CTL = TASSEL_1 + ID_3 + MC_1 + TACLR              // ACLK, div /8, upmode
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void ISR_timerA0(void){
          inc_time();
}
```

main.c (MSP430)

```
#include "timer.h"
#include "timer_msp430.h"
#include "msp430.h"

void main (void){
    time_t hora_actual={23,59,59,750};
    set_time(hora_actual);                    // Inicializo hora
    init_clock();                             // Configuro Clock
    init_timerA();                            // Configuro TimerA
    __enable_interrupt();
    while (1){
          get_time(&hora_actual);
    }
}
```

main.c (PC)

```
#include "timer.h"
void main (void){
    time_t hora_actual={23,59,59,750};
    set_time(hora_actual);           //Inicializo hora
    While (1){
          inc_time();
          inc_time();
          get_time(hora_actual);
          printf("%d\n", hora_actual);
    }
}
```

Interacción: plataforma de prueba y código HI

- Código HI:
  - **entradas**:
    - respuestas a eventos/interrupciones: interfaz con sensores, periféricos, dispositivos, etc.
  - **salidas**:
    - interacción con el exterior: manejo de “actuadores”.
- Plataforma de prueba:
  - recibe comandos (que simulan eventos de hardware)
    - interactivamente del teclado / archivo
  - dispara acciones en el código HI (llamando funciones)
  - simula el hardware (recibiendo llamadas del HI)
  - responde con acciones
    - escribiendo a pantalla / archivos de registro (log files)

Rutinas de atención a interrupciones

- Estructurar ISRs para que estén divididas:
  - HD: interactúa con el hardware.
  - HI: interactúa con el resto del módulo/sistema
- Plataforma de prueba:
  - simula el comportamiento del código HD
  - llama código HI para probar la respuesta.

Instruction set simulators

- Simulador en *host* de instrucciones del μC.
- Características:
  - permite probar código HD (dependiente de HW)
  - mismo código binario que correrá en destino
    - no existe problema de migración de *host* a *target*.
- Pregunta: ¿Quién o cómo se simula el entorno?
  - IAR simulación de interrupciones scripts
  - Proteus Virtual System Modelling (VSM): varios uCs
  - Cooja (IoT): varios uC orientado a WSN

Aserciones: el macro `assert`

- Definición

  aserción.

  (Del lat. *assertĭo, -ōnis*).

  1. f. Acción y efecto de afirmar o dar por cierto algo.

  2. f. Proposición en que se afirma o da por cierto algo.

- Técnica para detectar bugs.
  - Desperdigar llamadas a “assert” en el código.

    ```
    assert (pFrame != NULL);
    assert (byMacAddrFrom <= ADDR_MAX);
    ```

  - Si la condición es:
    - verdadera, no pasa nada.
    - falsa, se detiene el programa pero antes indica el error.
  - Ejemplo:

    ```
    Assertion failed: ptr != 0, file foo.c, line 27
    ```

Aserciones: beneficios

- Utilidad
  - hacer explícitas ciertas asunciones
  - corroborar parámetros en las funciones
  - ayuda a documentar asunciones.
- Implementación como macro (`#define`)
  - pueden ser “eliminadas” para el código del producto final.
- Detección de errores en el punto que se producen
  - ya que se detiene en el punto del error y no más adelante.
- Linux magazine June 2003:

  “Enthusiastic use of assert( ) can turn a three-day debug fest into a three minute bug fix. Practice the lazy developer mantra: An assertion failed is an hour saved.”

Aserciones en sistemas embebidos

- Muy útil durante desarrollo
  - prueba en host y simulador
- Más difícil de usar en target
  - no siempre hay pantalla y se detiene ejecución
- En target se puede (cuando falla un assert):
  - Poner el CPU en un estado dado (se detiene).
    - Deshabitar interrupciones y entrar en bucle infinito.
    - Encender un patrón especial de LEDs.
  - Escribir códigos de error en memoria (luego accesibles)
  - En lugar de printf a pantalla, escribir a puerto serie, etc.

Aserciones en MSP430

assert_prueba.h

```
#define LED3 (0x0003)
#define DEBUG

#ifdef DEBUG
#define ASSERT(expr) {\
    if (!(expr)) {\
    P2OUT |= LED3;\
    while (1) {};\
    }\
    //    __error__(__FILE__, __LINE__); // para PC
}

#else
#define ASSERT(expr)
#endif
```

Actividad en grupo

- Aserciones en MSP430
  - Actividad:
    - “Proteger” la cola implementada en clase 5 usando ASSERTS para evitar sacar un dato de una cola vacía o poner un dato en una cola llena (ver slide siguiente).
  - Grupos:
    - 3 a 4 participantes
  - Tiempo:
    - 5 minutos

Aserciones en MSP430

queue.h

```
#define QUEUE_SIZE 20

void q_init();
void q_add(char i);
char q_get();
int  q_isempty();
int  q_isfull();
```

queue.c

```
#include "queue.h"
#include "assert_prueba.h"

static int head;
static int tail;
static char q_elements[QUEUE_SIZE];


void q_add(char in){
  q_elements[head] = in;
  head++;
  if (head == QUEUE_SIZE){
    head = 0;
  }

...
```

Aserciones en MSP430

queue.h

```
#define QUEUE_SIZE 20

void q_init();
void q_add(char i);
char q_get();
int  q_isempty();
int  q_isfull();
```

queue.c

```
#include "queue.h"
#include "assert_prueba.h"

static int head;
static int tail;
static char q_elements[QUEUE_SIZE];


void q_add(char in){
   ASSERT(!q_isfull()); // if queue if full => abort
  q_elements[head] = in;
  head++;
  if (head == QUEUE_SIZE){
    head = 0;
  }

...
```

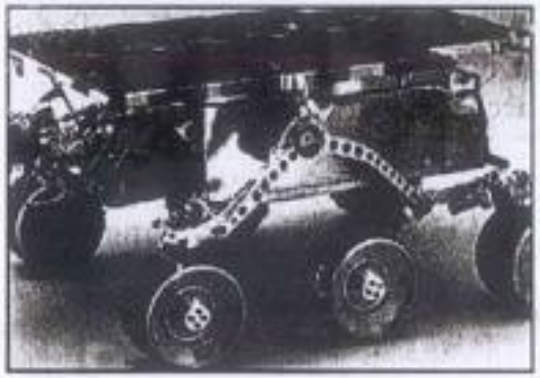
Bibliografía

- “An Embedded Software Primer” David E. Simon
  - Herramientas de desarrollo
    - Cap. 9 Embedded Software Development Tools
  - Técnicas de depuración
    - Cap. 10 Debugging Techniques

Bug of the Month

Man Finds Bugs on Mars

Wherever a computer goes, bugs are sure to follow. When the Mars Pathfinder developed a glitch, NASA had to somehow upload new code without losing valuable time needed for exploration. The most confounding bug on the Pathfinder mission appeared July 10. Steven Stolper, software engineer for the Mars Pathfinder, calls it "one in a million, insidious, and hard to replicate." The snafu arose because the OS, Wind River's VxWorks, developed a mutual-exclusion problem: A low-priority function (in this case, recording weather) interfered with the system's multitasking schedule. The system couldn't finish all the tasks it needed to, missed a real-time deadline, and then shut itself down. "It's a kind of interplanetary Control-Alt-Delete," says Stolper. "When things go wrong, the system goes into a power-safe mode and waits for ground control to help out." Without a fix being implemented, this problem would replay itself over and over.



PHOTOGRAPH: NASA/JPL/CALTECH © 1997

Pathfinder bugs inhibited the Rover.

To identify the bug, engineers recreated the malfunction on Earth, identified the offending subroutine, and uploaded the binary difference between the new code and the buggy code on the Pathfinder. –Jason Krause

Send yours to jkrause@mgh.com!