dit
UPM

Programación de bajo nivel

Juan Antonio de la Puente
DIT/UPM

Transparencias basadas en el capítulo 15 del libro de A. Burns y A. Wellings *Real-Time Systems and Programming Languuages*, 3ª edición (2001)

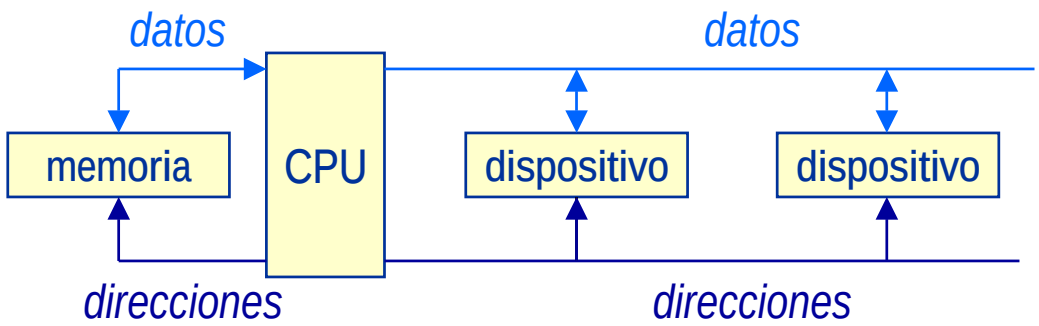
Objetivos

- Los sistemas de tiempo real suelen tener dispositivos de entrada y salida especiales
- Los manejadores de dispositivos forman parte del software de aplicación
- Veremos cómo controlar los dispositivos de hardware en un lenguaje de alto nivel
- Veremos también cómo incluir los manejadores de dispositivos en el modelo de tareas de tempo real

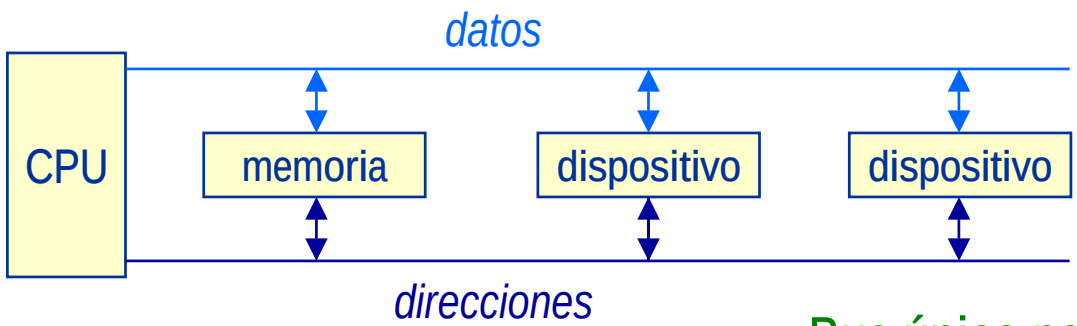
Índice

- **Mecanismos de hardware para entrada y salida y manejadores de dispositivos**
- Mecanismos de bajo nivel en Ada
  - cláusulas de representación
  - manejo de interrupciones
  - ejemplo
  - código de máquina
- Programación de dispositivos en C
- Planificación con manejadores de dispositivos

Arquitecturas de entrada-salida



Bus de E/S separado del de memoria



Bus único para E/S y memoria ***(memory mapped)***

Interfaz con los dispositivos de entrada y salida

- Los dispositivos de entrada-salida se conectan a los otros elementos del computador mediante *controladores* que presentan una interfaz homogénea
- El procesador intercambia datos e información de control y estado con los controladores mediante registros de hardware
- La forma concreta de hacerlo depende de la arquitectura de entrada y salida

Entrada y salida en lenguaje de máquina

Bus de E/S separado

- Registros en espacio de direcciones de E/S (direcciones de *puerto*)
- Se leen y escriben mediante instrucciones de E/S específicas

```
in  r, port
out r, port
```

- Ejemplo: Intel 486

Bus de memoria

- Registros en el espacio de direcciones de memoria
- Se leen y escriben mediante instrucciones de transferencia de datos

```
mov r, address
mov address, r
```

- Ejemplo: Motorola 68000

Sincronización

Por consulta *(status driven)*

- El procesador interroga al controlador para comprobar el estado del dispositivo
- Complicado e ineficiente
- A veces es la única opción (en sistemas muy críticos es posible que no se permita usar interrupciones)

Por interrupción *(interrupt driven)*

- El controlador presenta una interrupción en determinadas circunstancias
- Un *manejador de interrupción* se encarga de tomar la acción adecuada
- Varios tipos
  - controlado por programa
  - acceso directo a memoria
  - controlado por canal
- Puede ser difícil estimar el tiempo de computo

Mecanismos de interrupción (1)

- Cambio de contexto
  - se guarda el estado del procesador antes de la interrupción
  - se carga el estado del manejador de interrupción
  - cuando se completa el manejador, se restaura el estado anterior
- Identificación del dispositivo que interrumpe
  - por *vector de interrupción:* array asociado al hardware de interrupción que contiene información sobre el estado del manejador de interrupción
  - por *estado*: cada interrupción tiene una *palabra de estado* asociada que indica cuál es el origen de la misma
  - por *consulta*: un manejador general consulta a los dispositivos para averiguar cuál ha interrumpido

Mecanismos de interrupción (2)

- Identificación de la causa de la interrupción
  - consultando el estado del dispositivo
  - con diferentes interrupciones para el mismo dispositivo
- Control de interrupciones

  Las interrupciones de un dispositivo pueden estar *permitidas (enabled)* o *inhibidas (disabled)*
  - mediante *indicadores (flags)* en registros de estado
  - mediante una *máscara (mask)*, con un bit por dispositivo
  - mediante un *nivel* de prioridad de hardware
    - » cada dispositivo tiene un nivel asociado
    - » si el nivel del procesador es mayor o igual que el de un dispositivo, no se aceptan interrupciones de éste

Mecanismos de interrupción (3)

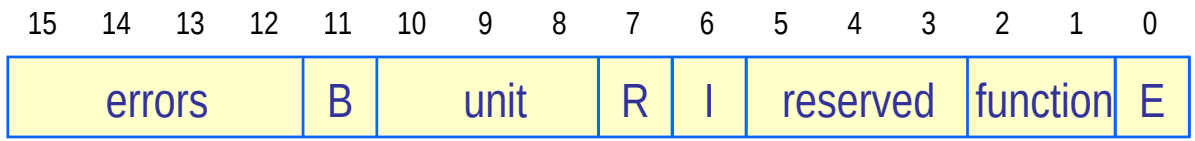
- Control de prioridad
  - a veces se asocia una *prioridad* a cada fuente de interrupciones
  - la prioridad indica la urgencia relativa de la interrupción
  - puede ser estática o dinámica
  - normalmente está relacionada con los niveles de prioridad del procesador

Ejemplo de sistema de entrada y salida

- Suponemos una arquitectura con E/S por bus de memoria, con interrupciones (como la M68000)
- Cada dispositivo tiene dos tipos de registros
  - Registros de control y estado (CSR), que contienen información sobre el dispositivo y el control de interrupciones
  - Registros de datos (DBR), que contienen los datos que se transmiten al dispositivo o desde él
- Un mismo dispositivo puede tener varios CSR o DBR
- Cuando se produce una interrupción, se guarda el estado (PC y PSW) en la pila, y se carga el nuevo estado del vector de interrupción correspondiente
  - Las interrupciones y el procesador tienen asociada una prioridad
  - Una interrupción puede desalojar a un manejador con prioridad más baja

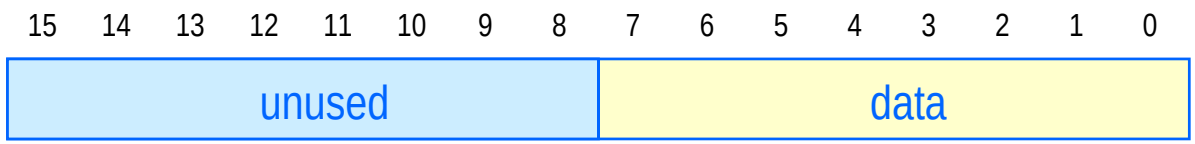
Estructura de los registros

CSR

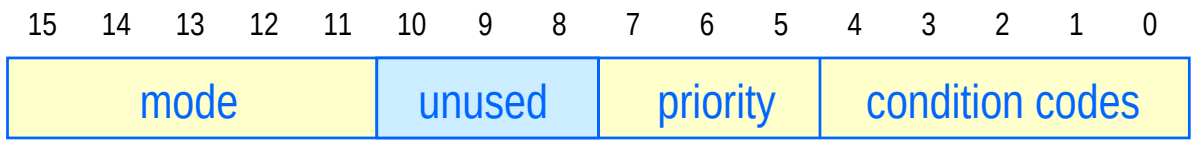


B device busy
R device ready / done
I interrupt enable
E device enable

DBR



PSW



Manejadores de dispositivos

- Para realizar manejadores de dispositivos hace falta:
  - Manipular registros de hardware y direcciones
    - » un registro se puede representar como una variable o como un canal de comunicación
  - Enlazar interrupciones con código. Algunas posibilidades son:
    - » llamada a procedimiento
    - » activación de proceso esporádico
    - » suceso asíncrono
    - » sincronización con variable de condición
    - » mensaje

Índice

- Mecanismos de hardware para entrada y salida y manejadores de dispositivos
- **Mecanismos de bajo nivel en Ada**
  - **cláusulas de representación**
  - **manejo de interrupciones**
  - ejemplo
  - código de máquina
- Programación de dispositivos en C
- Planificación con manejadores de dispositivos

Mecanismos de bajo nivel en Ada

- Ada 95 tiene varios mecanismos de bajo nivel que permiten acceder a registros, direcciones e interrupciones
  - cláusulas de representación
    - » permiten especificar la representación de tipos y objetos en la arquitectura de hardware
      - representación de atributos (tamaño, dirección, alineación)
      - representación de tipos enumerados
      - representación de registros
  - manejadores de interrupciones
    - » permiten asociar una interrupción a un procedimiento protegido
  - subprogramas en lenguaje de máquina
    - » permiten un control total sobre la arquitectura de hardware

Cláusulas de representación de atributos

- Tamaño

```
type Data_Register is mod 2**8;
for Data_Register'Size use 16;
-- los objetos de tipo Data_Register ocupan 16 bits
```

- Dirección

```
Data_Buffer : Data_Register;
for Data_Buffer'Address use 8#177560#;
-- Data_Buffer se ubica en la dirección 177560 octal
```

- Alineación

```
for Data_Register'Alignment use 2;
-- los objetos de tipo Data_Register se tienen que ubicar
-- en direcciones múltiplo de 2
```

Cláusulas de representación de tipos enumerados

- Codificación interna de los valores de un tipo enumerado

```
type Flag is (On, Off);
for Flag use (On => 1, Off => 0);
-- representación numérica de los valores del tipo Flag
```

- Orden de bits en los objetos de un tipo de datos

```
type Register is mod 256;
for Register'Bit_Order use Low_Order_First;
  -- el bit 0 es el menos significativo
```

Cláusulas de representación de registros

- Orden, posición y tamaño de los componentes de un tipo registro

```
type Control_Register is
record
   Enable : Flag;
   Done   : Flag;
   Busy   : Flag;
   Error  : Error_Type;
end record;

for Control_Register use
record
   Enable at 0 range  0 ..  0;
   Done   at 0 range  7 ..  7;
   Busy   at 0 range 11 .. 11;
   Error  at 0 range 12 .. 15;
end record;
```

Manejo de interrupciones en Ada

- La *ocurrencia* de una interrupción tiene dos fases:
  - *generación:* suceso que hace que la interrupción esté disponible para el programa
  - *entrega:* acción que invoca un *manejador de interrupción*
- Entre la generación y la entrega la interrupción está *pendiente* de manejar
  - la *latencia* de la interrupción es el tiempo durante el cual está pendiente
- Mientras se ejecuta el manejador la interrupción se *bloquea* (se impide que se generen nuevas ocurrencias)
- Hay interrupciones *reservadas:* no se pueden escribir manejadores para ellas (las maneja el núcleo de ejecución)
  - las interrupciones que no están reservadas tienen un manejador por defecto
  - se pueden escribir manejadores adaptados a la aplicación

Manejadores y objetos protegidos

- Los manejadores son procedimientos protegidos sin parámetros
- Cada interrupción tiene un identificador único de tipo `Interrupt_Id` (definido en el paquete `Ada.Interrupts`)
- El pragma `Attach_Handler` especifica que un procedimiento protegido es un manejador asociado a una interrupción

```
pragma Attach_Handler(Handler_Name, Interrupt);
```

- El pragma `Interrupt_Handler` permite especificar que un un procedimiento protegido es un manejador, pero no lo asocia con ninguna interrupción

```
pragma Interrupt_Handler(Handler_Name);
```

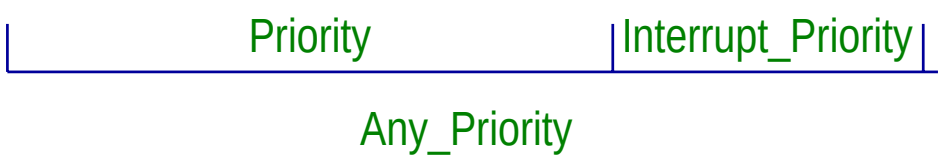
  - la asociación con una interrupción se hace dinámicamente

Prioridad de un manejador

- La prioridad de un manejador es del objeto protegido donde se encuentra (especificada con un pragma `Interrupt_Priority`)

```
pragma Interrupt_Priority(Priority_Value);
```

- El valor de la prioridad debe pertenecer al subtipo `System.Interrupt_Priority`
  - es un subtipo de `System.Any_Priority`



Ada.Interrupts

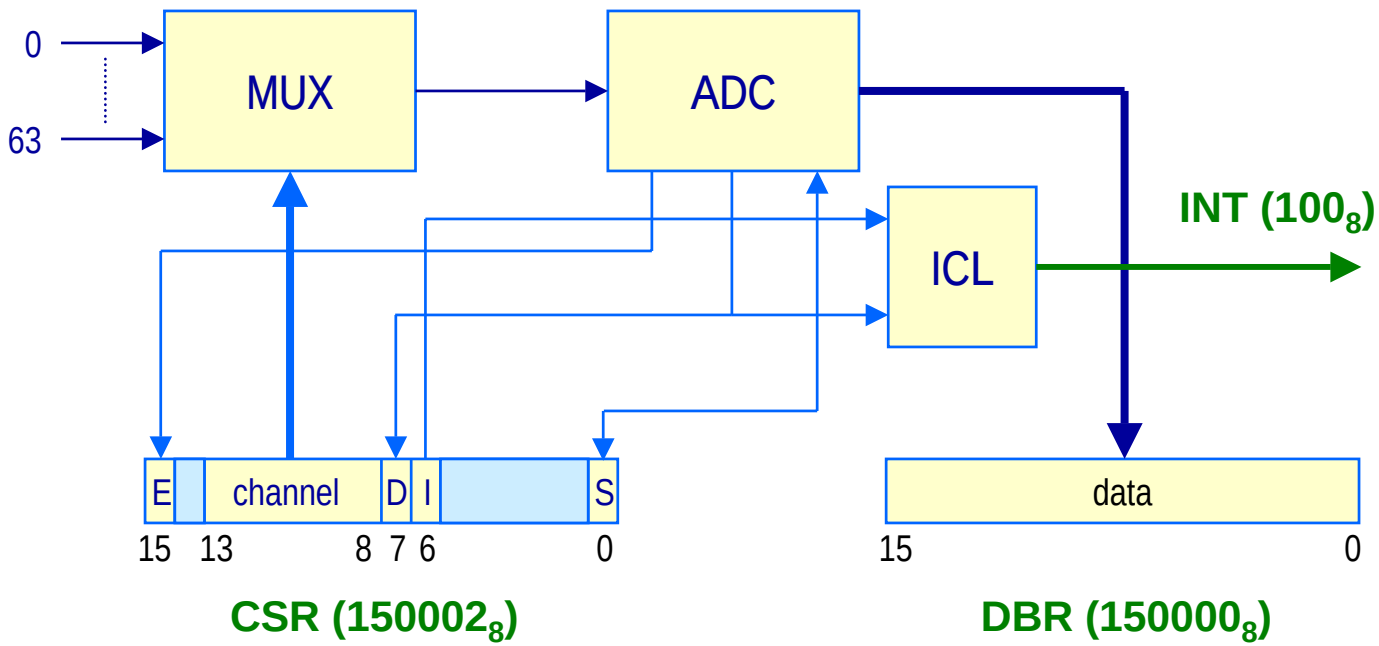
```
package Ada.Interrupts is
   type Interrupt_ID is ...;  -- discreto
   type Parameterless_Handler is access protected procedure;
   function  Is_Reserved (Interrupt : Interrupt_ID) return Boolean;
   function  Is_Attached (Interrupt : Interrupt_ID) return Boolean;
   function  Current_Handler (Interrupt : Interrupt_ID)
     return Parameterless_Handler;

   procedure Attach_Handler  (New_Handler :     Parameterless_Handler;
                              Interrupt   :     Interrupt_ID);
   procedure Exchange_Handler(Old_Handler : out Parameterless_Handler;
                              New_Handler :     Parameterless_Handler;
                              Interrupt   :     Interrupt_ID);
   procedure Detach_Handler  (Interrupt : Interrupt_ID);
   ...
end Ada.Interrupts;
```

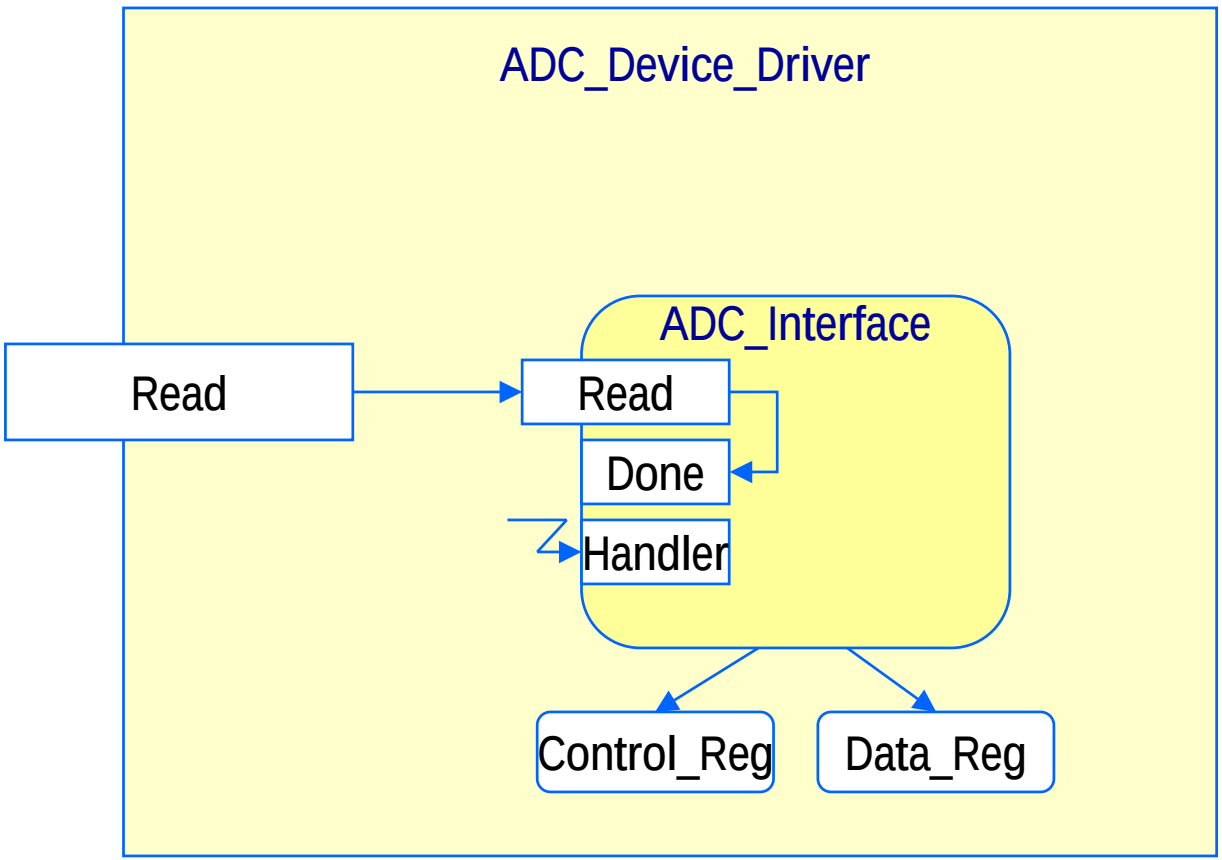
Índice

- Mecanismos de hardware para entrada y salida
- Manejadores de dispositivos
- **Mecanismos de bajo nivel en Ada**
  - cláusulas de representación
  - manejo de interrupciones
  - **ejemplo**
  - código de máquina
- Programación de dispositivos en C
- Planificación con manejadores de dispositivos

Ejemplo : convertidor A/D



Estructura del manejador



ADC_Device_Driver

```
package ADC_Device_Driver is

   Bits        : constant := 16;
   Max_Measure : constant := 2**Bits - 1;

   type    Channel     is range 0 .. 63;
   subtype Measurement is Integer range 0 .. Max_Measure;

   procedure Read (C :     Channel;
                   M : out Measurement);
   -- potentially blocking
   -- may raise Conversion_Error

   Conversion_Error : exception;
private
   for Channel'Size use 6;
end ADC_Device_Driver ;
```

ADC_Device_Driver (implementación)

```
with Ada.Interrupts;           use Ada.Interrupts;
with Ada.Interrupts.Names;     use Ada.Interrupts.Names;
with System;                   use System;
with System.Storage_Elements; use System.Storage_Elements;

package body ADC_Device_Driver is

   Bits_In_Word  : constant := 16;
   Word          : constant :=  2;
```

Registro de control (1)

```
type Flag is (Off, On);
  for Flag use (Off => 0, On => 1);
  for Flag'Size use 1;

type Control_Register is
   record
      Start    : Flag;
      I_Enable : Flag;
      Done     : Flag;
      Chan     : Channel;
      Error    : Flag;
   end record;
```

Registro de control (2)

```
pragma Pack(Control_Register);
for Control_Register use
   record
      Start    at 0*Word range  0 ..  0;
      I_Enable at 0*Word range  6 ..  6;
      Done     at 0*Word range  7 ..  7;
      Chan     at 0*Word range  8 .. 13;
      Error    at 0*Word range 15 .. 15;
   end record;
for Control_Register'Size      use Bits_In_Word;
for Control_Register'Alignment use Word;
for Control_Register'Bit_Order use Low_Order_First;
```

Registro de datos

```
type Data_Register is range 0 .. Max_Measure;
  for  Data_Register'Size      use Bits_In_Word;
  for  Data_Register'Alignment use Word;
```

Objetos registro

```
Control_Reg_Address : constant Address
  := To_Address(8#150002#);

Data_Reg_Address    : constant Address
  := To_Address(8#150000#);

ADC_Priority : constant Interrupt_Priority := 31;

Control_Reg : aliased Control_Register;
  for Control_Reg'Address use Control_Reg_Address;

Data_Reg    : aliased Data_Register;
  for Data_Reg'Address use Data_Reg_Address;
```

Interrupt_Interface

```
protected type Interrupt_Interface
                 (Int_Id : Interrupt_Id;
                  CR     : access Control_Register;
                  DR     : access Data_Register) is

   entry Read (C : Channel; M : out Measurement);

private

   entry Done (C : Channel; M : out Measurement);

   procedure Handler;
     pragma Attach_Handler(Handler, Int_Id);
     pragma Interrupt_Priority(ADC_Priority);

   Interrupt_Occurred : Boolean := False;
   Next_Request       : Boolean := True;

end Interrupt_Interface;
```

ADC_Interface

```
ADC_Interface : Interrupt_Interface
                  (INTADC,
                   Control_Reg'Access,
                   Data_Reg'Access);
```

Interrupt_Interface (1)

```
protected body Interrupt_Interface is

  entry Read (C : Channel; M : out Measurement)
   when Next_Request is
     Shadow_Register : Control_Register;
  begin
     Shadow_Register := (Start     => On,
                         I_Enable  => On,
                          Done     => Off,
                          Chan     => C,
                          Error    => Off);
     CR.all := Shadow_Register;
     Interrupt_Occurred := False;
     Next_Request       := False;
     requeue Done;
  end Read;
```

Interrupt_Interface (2)

```
   procedure Handler is
   begin
      Interrupt_Occurred := True;
   end Handler;

   entry Done(C : Channel; M : out Measurement)
    when Interrupt_Occurred is
   begin
      Next_Request := True;
      if CR.Done = On and CR.Error = Off then
         M := Measurement(DR.all);
      else
          raise Conversion_Error;
      end if;
   end Done;

end Interrupt_Interface;
```

Read

```
   procedure Read (C : Channel; M : out Measurement) is
   begin
      for I in 1..3 loop
         begin
            ADC_Interface.Read(C, M);
            return;
         exception
            when Conversion_Error => null;
         end;
      end loop;
      raise Conversion_Error;
   end Read;

end ADC_Device_Driver;
```

Conversión de tipos

- La función genérica *Unchecked_Conversion* convertir valores de cualquier tipo a cualquier otro:

```
  generic
    type Source (<>) is limited private;
    type Target (<>) is limited private;
function Ada.Unchecked_Conversion(S : Source)
  return Target;
```

```
function To_Measurement is
  new Ada.Unchecked_Conversion(Data_Register, Measurement);
```

- No se hace ningún tipo de comprobación
- Se copia la representación de un dato al otro sin hacer ninguna otra operación

Subprogramas en lenguaje de máquina

- A veces hace falta escribir parte del programa en lenguaje de máquina (por ejemplo, si hay que usar instrucciones de E/S específicas)
- Se hace con una sintaxis parecida a la de Ada
- Sólo se puede hacer en el cuerpo de un subprograma, y en este caso sólo puede contener código de máquina y cláusulas *use*
- Depende de la implementación, pero el esquema general es

```
instrucción_de_máquina ::= expresión_cualificada
```

- La expresión es de un tipo declarado en el paquete `System.Machine_Code`
  - en este paquete se declaran tipos registro para los distintos formatos de instrucción de máquina

Ejemplo (1)

```
M : Mask;

procedure Set_Mask;
   pragma Inline(Set_Mask);

procedure Set_Mask is
  use System.Machine_Code;
  -- suponemos que System.Machine_Code es visible
aquí
begin
   SI_Format'(Code => SSM, B => M'Base_Reg, D =>
M'Disp);
end Set_Mask;
```

Ejemplo (2)

```
procedure Input;
   pragma Inline(Input);

procedure Input is
  use System.Machine_Code;
begin
   SI_Format'(Code => IN,
              R    => A,
              Port => ADC_PORT_1);
   SI_Format'(Code => SAVE, R => A,
Data'Adress);
end Input;
```

Índice

- Mecanismos de hardware para entrada y salida
- Manejadores de dispositivos
- Mecanismos de bajo nivel en Ada
  - cláusulas de representación
  - manejo de interrupciones
  - código de máquina
- **Programación de dispositivos en C**
- Planificación con manejadores de dispositivos

Manejadores en C

- Se usan punteros para direccionar los registros
- Hay operaciones que actúan bit a bit

Ejemplo

```
#define START   01     /* hexadecimal */
#define ENABLE  040
#define ERROR   08000

unsigned sort int *register, shadow, channel:

register = 0AA12;
channel  = 12;
shadow  |= (channel < < 8) | START | ENABLE;
*register = shadow
```

Interrupciones en C

- Hay que programar en un nivel de abstracción bajo
  - el manejador es un procedimiento sin parámetros
  - su dirección se copia al vector
  - hay que programar directamente la comunicación y sincronización con el resto del programa cuando termina el manejador de interrupción
- POSIX no ayuda mucho

Índice

- Mecanismos de hardware para entrada y salida
- Manejadores de dispositivos
- Mecanismos de bajo nivel en Ada
  - cláusulas de representación
  - manejo de interrupciones
  - código de máquina
- Programación de dispositivos en C
- **Planificación con manejadores de dispositivos**

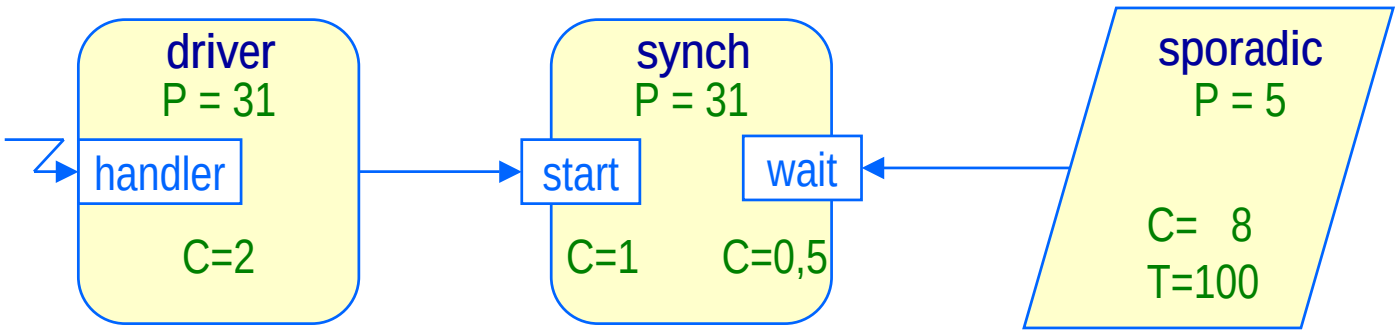
Planificación y tiempo de respuesta

- La sincronización por interrupción controlada por DMA o por canal no permite analizar los tiempos de respuesta
- Consideraremos sólo interrupciones controladas por programa y sincronización por consulta de estado
- Partimos del análisis de tiempos con prioridades fijas

Sincronización por interrupción

- Suponemos que el manejador lanza una tarea esporádica τ_S
  - La prioridad P_H del manejador suele ser mas alta que la de cualquier tarea de aplicación
  - Esto causa interferencia en las tareas de prioridad superior a la esporádica
  - Para modelar el manejador añadimos una tarea τ_S' con prioridad P_H, separación $T_H = T_S$, y tiempo de cómputo C_H, igual al del manejador
  - También hay que tener en cuenta el efecto sobre los bloqueos cuando se usa un objeto protegido para sincronizar el manejador con la tarea periódica

Ejemplo



Tarea	P	T	D	C	B
S'	31	100	–	2	1
…	…	…	…	…	…
S	5	100	15	8	…
…	…	…	…	…	…

Sincronización por consulta

- Hay varias formas de esperar que el dispositivo esté listo en una operación de E/S:
  - **Espera ocupada en el indicador *Done***

```
begin
-- iniciar operación
   while not Done loop null; end loop;
-- completar operación
end;
```

  - **Espera temporizada** (si la duración de la operación está acotada)

```
begin
-- iniciar operación
   delay 0.030;
-- completar operación
end;
```

Espera ocupada

- Solo es aceptable si la operación es muy breve
- Hay que acotar la duración de la espera en el bucle
  - se puede añadir un *timeout*
- El tiempo de cómputo es igual a la suma de los dos segmentos más la duración máxima de la espera

Espera temporizada

- Se analiza por separado cada segmento
  - iniciación
  - terminación

  (equivale a dos tareas con el mismo período)
- Se puede bloquear al principio de cada uno de los segmentos aunque se use el protocolo del techo de prioridad inmediato
- El tiempo de respuesra es igual a la suma de los tiempos de respuesta de cada segmento más el retardo

Desplazamiento de períodos

- Con tareas periódicas se puede preparar la siguiente operación al final del ciclo

```
loop
   delay until Next_Release;
-- completar operación
-- utilizar el resultado
-- iniciar siguiente operación
   Next_Release := Next_Release + Period;
end;
```

- Se analiza como una tarea periódica convencional
- No siempre es válido: los datos pueden ser viejos cuando se usan
- Para asegurar que da tiempo, tiene que ser $D \leq T - S$
  - S es el tiempo necesario para la operación
  - La máxima antigüedad de los datos es $T + R$

Resumen

- Para programar manejadores de dispositivos hace falta
  - intercambiar datos e información de control con el dispositivo
  - poder manejar interrupciones
- Ada 95 tiene mecanismos de bajo nivel que permiten programar dispositivos
  - cláusulas de representación
  - procedimientos protegidos enlazados a interrupciones
  - subprogramas en lenguaje de máquina