Interrupciones y microcontroladores (y sus periféricos)

Clase 2

Sistemas embebidos para tiempo real

Agenda próximas tres clases

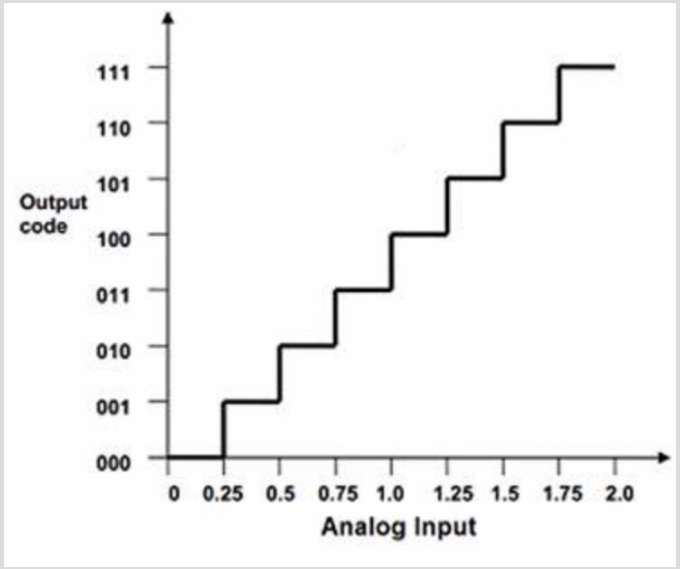
- Primera clase
  - Fundamentos de las Interrupciones
  - Periféricos del MSP430G2553
  - Periférico básico: Reloj (Basic Clock Module+)
- Segunda clase (hoy)
  - Periféricos avanzado: Conversor A/D (ADC10)
  - Problema de datos compartidos (y solución básica)
  - Latencia en las interrupciones
- Tercer clase
  - Soluciones alternativas al problema de los datos compartidos

Objetivos

- Utilizar e interpretar manuales de usuario y hojas de datos.
- Familiarizarse con la configuración de periféricos **avanzados**.
- Reconocer el problema de datos compartidos.
- Identificar los tiempos involucrados en las interrupciones.

Conversor A/D (ADC)

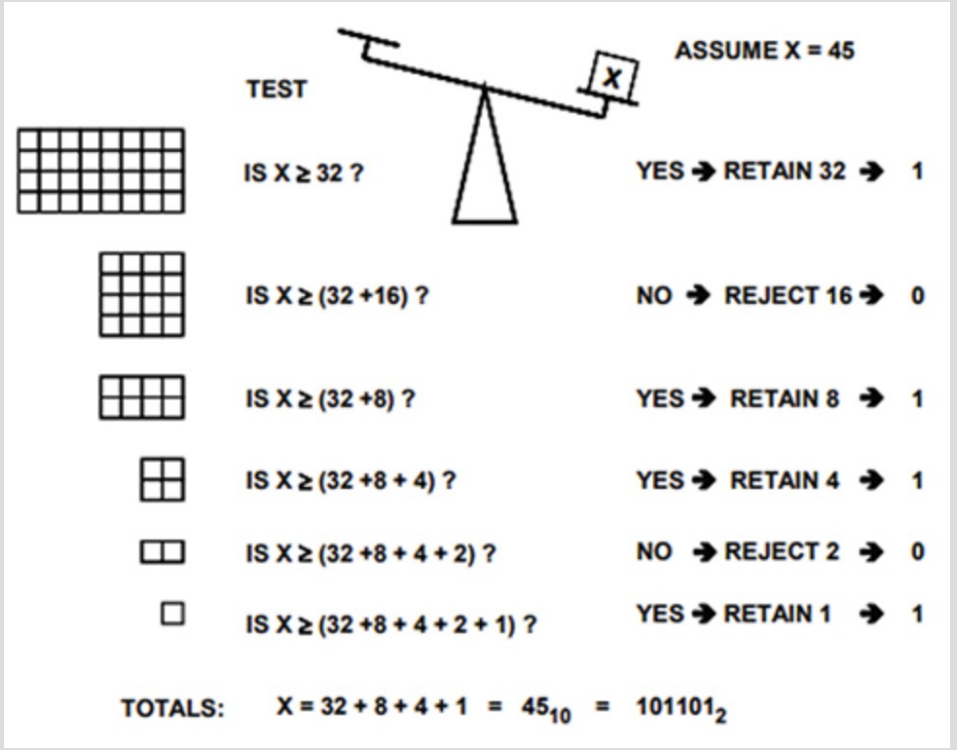
Figuras tomadas de https://www.digikey.com/en/articles/techzone/2017/sep/adc-dac-tutorial



3-bit ADC con V_{REF}=1.75V (FS)

$$V_{IN} = N_{ADC} * (1.75V / 7)$$

$$V_{IN} = N_{ADC} * [V_{REF} / (2^N-1)]$$



6-bit SAR ADC

Conversor A/D (ADC)

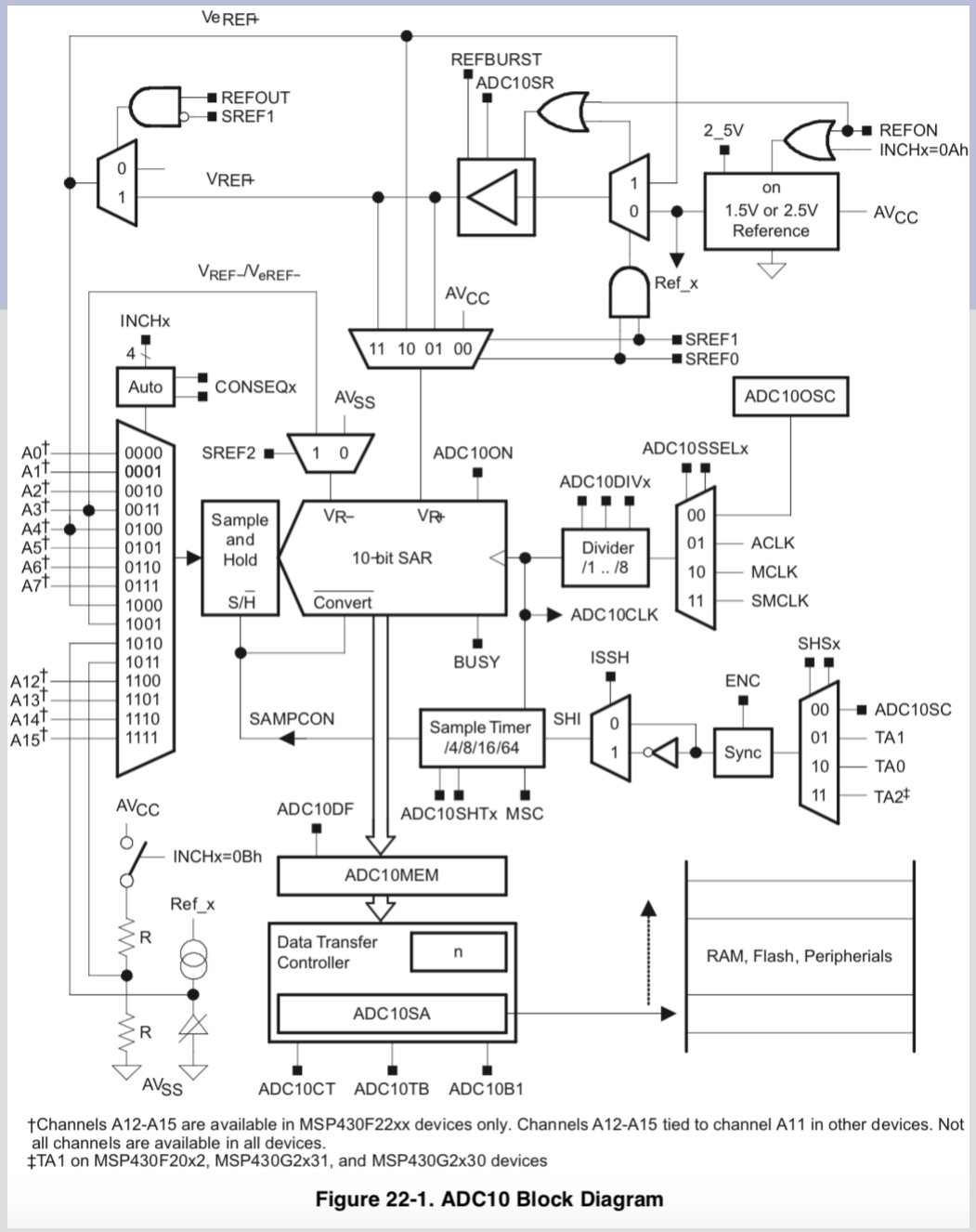
- Conversor ADC10 (periférico)
  - SAR 10 bits
  - Hasta 8 entradas (externas)
  - Frecuencia de muestro <= 200-ksps
  - V_{REF}: on-chip (1.5 V or 2.5 V) o externa

Actividad en grupo

- Estudiar funcionamiento del ADC10
  - Responder:
    - Identificar bloques en Fig. 22-1
  - Grupos:
    - 2 a 4 participantes
  - Tiempo:
    - 5 minutos
  - Material:
    - Manual familia MSP430x2xx y hoja de datos MSP430G2553

ADC10

- Bloques:
  - Gen Ref
  - Reloj
  - Fmuestreo
  - Entrada data
  - Salida data



†Channels A12-A15 are available in MSP430F22xx devices only. Channels A12-A15 tied to channel A11 in other devices. Not all channels are available in all devices.
‡TA1 on MSP430F20x2, MSP430G2x31, and MSP430G2x30 devices

Figure 22-1. ADC10 Block Diagram

ADC10

$$N_{ADC} = 1023 \times \frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}}$$

- N = 10 bits => 2^N-1=1023
- V_{R+}= REF positiva
- V_{R-}= REF negativa
- V_{REF}= REF
- V_{IN} = entrada analógica

ADC10

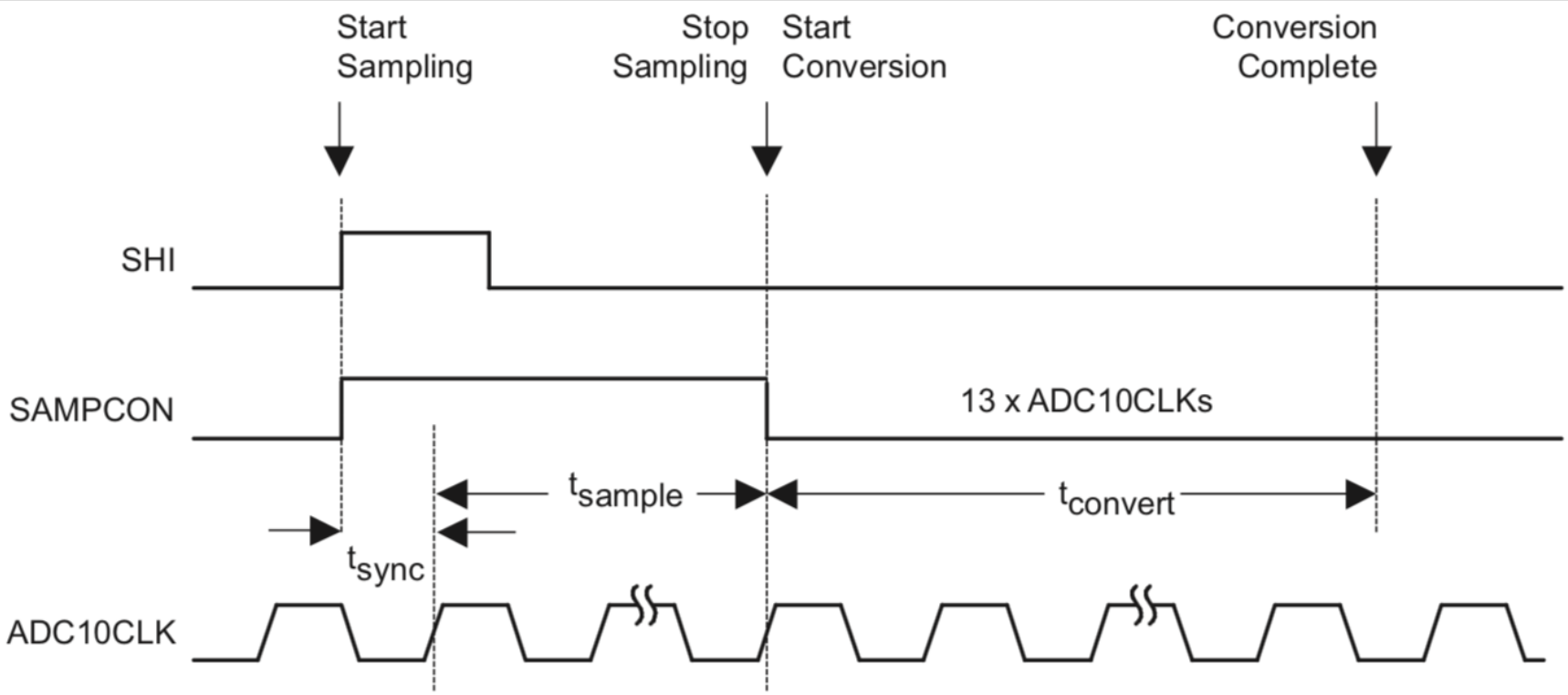


Figure 22-3. Sample Timing

ADC10: modo de operación

Table 22-1. Conversion Mode Summary

CONSEQx	Mode	Operation
00	Single channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat single channel	A single channel is converted repeatedly.
11	Repeat sequence-of-channels	A sequence of channels is converted repeatedly.

ADC10

- Single-channel single-conversion
- Arranque (SHS=0):
  - ENC=1 y ADC10SC=1
- Fin:
  - ADC10IF=1
  - Resultado en ADC10MEM

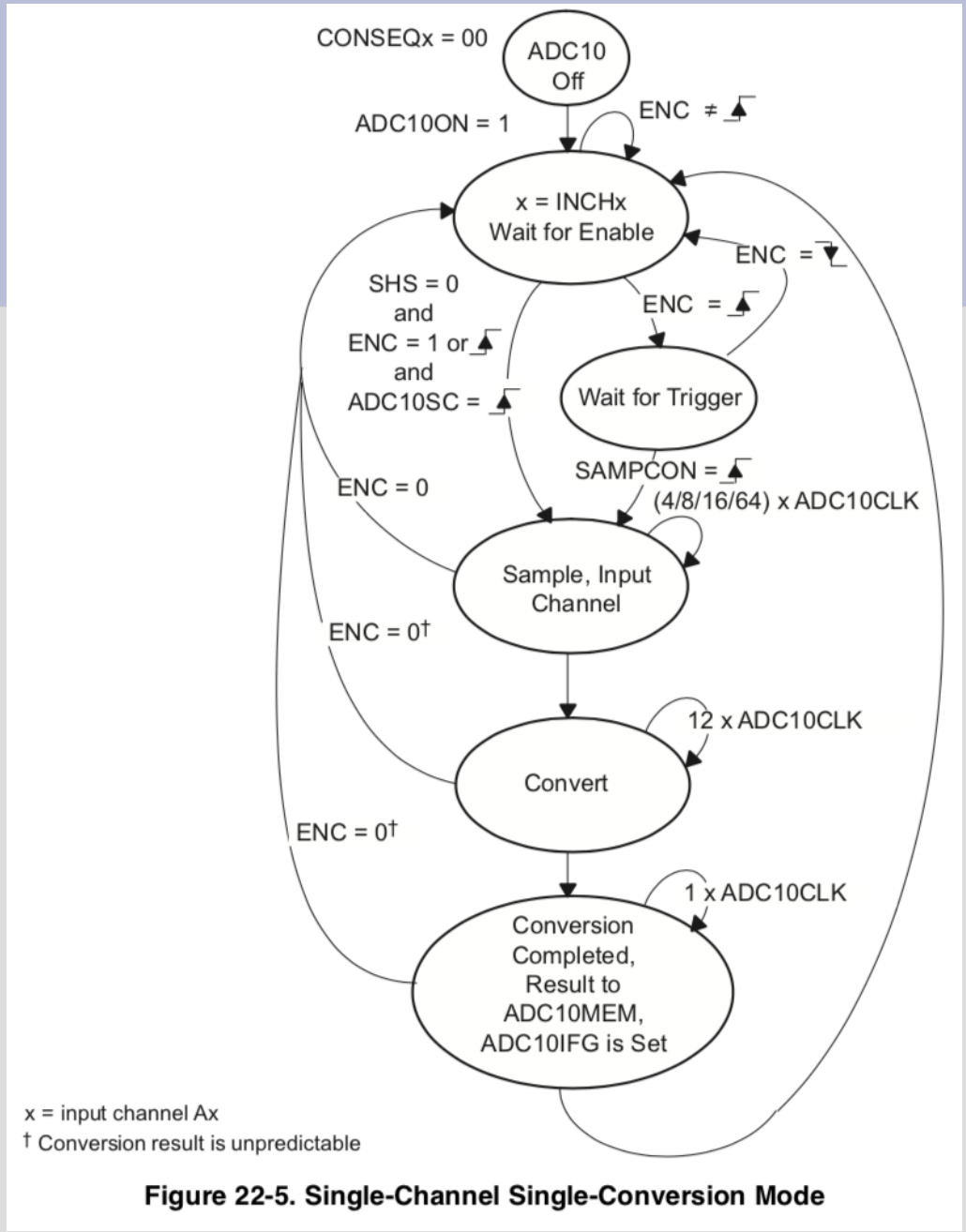


Figure 22-5. Single-Channel Single-Conversion Mode

ADC10

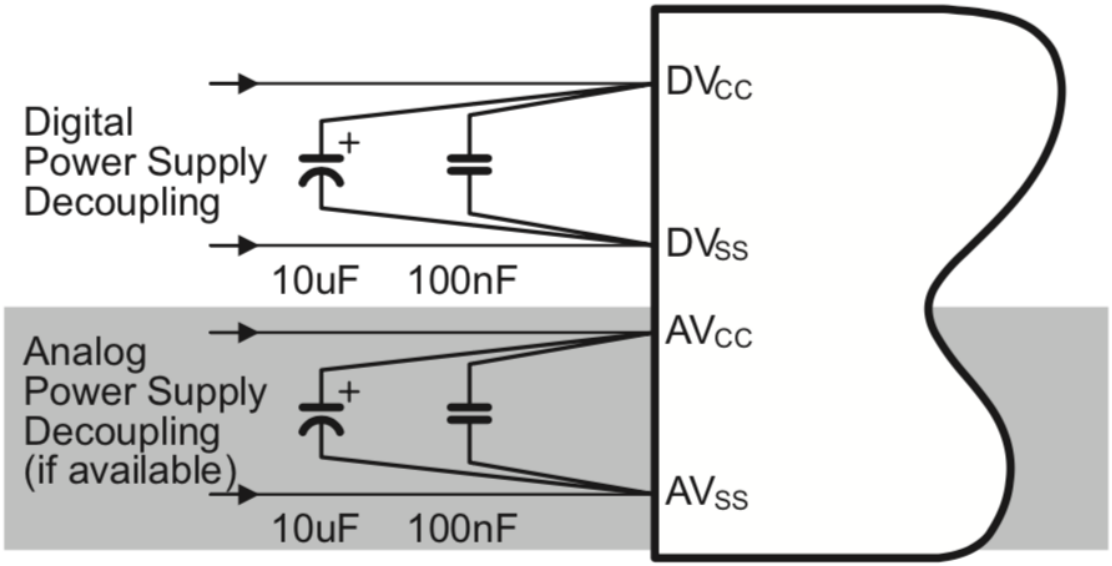


Figure 22-14. ADC10 Grounding and Noise Considerations (Internal V_{REF})

Actividad en grupo

- Configuración de registros del ADC10
  - Especificaciones:
    - 1 sola entrada analógica en pin A5
    - Período de muestreo = 250ms (usando interrupción del TIMER_A cada 250 ms: similar LAB2 pero usando VLOCLK como ACLK)
    - Modo: Single-channel-single-conversion
    - Reloj: interno del ADC (ADC10OSC)
    - Funcionamiento lo más rápido que se pueda
    - Maximizar rango dinámico: $V_{R+}=V_{CC}$ and $V_{R-}=V_{SS}$
    - Referencia interna = 1.5 V, disponible en pin A4 (para test)
  - Grupos:
    - 2 a 4 participantes
  - Tiempo:
    - 15 minutos
  - Material:
    - Manual familia MSP430x2xx y hoja de datos MSP430G2553

ADC10CTL0

```
// asume que el uC viene de un reset y los registros están
inicializados con sus valores por  defecto

ADC10CTL0 |= SREF_0;      // (VR+=VCC and VR-=VSS)
ADC10CTL0 |= ADC10SHT_0; // ADC10SHTx=00 (4x ADC10CLKs) queremos que
demore lo menos posible en el muestreo
ADC10CTL0 |= ADC10SR;     // ADC10SR= 1, ADC10SR=0 → fs<200ksps /
                        ADC10SR=1 → fs<50ksps (ADC10SR=1 reduce consumo)
ADC10CTL0 |= REFOUT;       // REFOUT=1, REF disponible en pin VREF+
ADC10CTL0 &= ~REFBURST;   // REFBURST=0 prende el buffer de Vref
siempre, Si REFBURST=1 el buffer está ON solo durante sample-and-
conversion (ver Sección 22.2.3.1)
ADC10CTL0 &= ~REF2_5V;    // REF2_5V=0, Vref = 1.5 V.
ADC10CTL0 |= REFON;        // REFON=1 habilita Vref.
ADC10CTL0 |= ADC10ON;    //  prende ADC
ADC10CTL0 |= ADC10IE;      // ADC10IE=1 habilita interrupción de ADC10
// ADC10CTL0 &= ~MSC;         // MSC=0, The sampling requires a rising
edge of the SHI signal to trigger each sample-and-conversion.
```

Otros registros

```
// ADC10 control register 0: ADC10CTL0
ADC10CTL1 |= INCH_5;       // entrada por pin A5
ADC10CTL1 |= SHS_0;        // SHSx=00 la conversión se dispara
seteando el bit ADC10SC
ADC10CTL1 &= ~ADC10DF;     // ADC10DF=0 (binario)
ADC10CTL1 |= ADC10DIV_0;   // sin prescaler, tan rápido como sea
posible
ADC10CTL1 |= ADC10SSEL_0; // ADC10OSC
ADC10CTL1 |= CONSEQ_0;     // single-channel-single-conversion


// ADC10 input enable register 0
ADC10AE0 |= BIT5;     // input = A5
```

Deberes

- Implementar la digitalización de una señal analógica en un MSP430G2553
  - Especificaciones:
    - Utilizar el ADC10 y el Basic Module Clock+ según se configuraron durante esta clase
    - Definir adc10.c y adc10.h
    - Mostrar el valor digitalizado en una variable en el CCS
  - Grupos:
    - 2 a 4 participantes
  - Tiempo:
    - 15 minutos
  - Material:
    - Manual familia MSP430x2xx y hoja de datos MSP430G2553

Deberes: pistas para la implementación

```
// En adc10.c
void adc_start(){
  while (ADC10CTL1 & ADC10BUSY){}; //if ADC10BUSY=1 then ADC busy
  ADC10CTL0 |= ADC10SC + ENC;
  //ADC10SC = 1 start conversion, ENC = 1 enable conversion
  return;
}

#pragma vector=ADC10_VECTOR // ADC10IFG is automatically reset when IRQ is accepted
__interrupt void ADC10_ISR(void){
    static const int ADC_FSR = 1023;
    static const float Vcc = 3.3; // Vss=0
    int Nadc=ADC10MEM;
    Vin = Vcc*Nadc/ADC_FSR; // Vss=0
}

// En timerA_hw.c
#pragma vector=TIMER0_A0_VECTOR
__interrupt void tic250 (void){
    timer_inc();
    P1OUT ^= LED1;              // Conmuta LED1 (P1.0 ) usando XOR
    adc_start();
    }
```

Deberes: optimizaciones

- Bajar consumo:
  - Dormir uC cuando no se hace nada
  - Configurar puertos I/O no usados
  - ¿Apagar ADC entre conversiones?
  - Inhabilitar Vref externa (era solo para test)
  - Ver sugerencias de CCS

Ejemplo de uso de interrupciones

- Descripción
  - Control de temperaturas en dos tanques.
  - Las temperaturas deben ser iguales.
  - Si son diferentes disparar alarma.
- Implementación
  - Interrupción periódica para lectura de temperaturas.
  - Lectura de temperatura es “instantánea”.
  - Verificación de temperaturas se hace en loop principal.

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;
  }
}
```

control-oculto.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)
  {
    if (iTemperatures[0] != iTemperatures[1])
      !! Se activa una alarma muy molesta;;
  }
}
```

control-oculto.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)
  {
    if (iTemperatures[0] != iTemperatures[1])
      !! Se activa una alarma muy molesta;
  }
}
```

```
          MOV     R1, (iTemperature[0])
          MOV     R2, (iTemperature[1])
          SUB     R1, R2
          JCOND ZERO, TEMP_OK
          ;
          ; Codigo para disparar alarma
          ;
TEMP_OK:
```

Problema de los datos compartidos

- ¿Cuándo está potencialmente presente?
  - Cuando se comparten datos entre ISR y el resto del código.
- ¿Por qué se comparten datos?
  - Las ISR deben hacer el trabajo estrictamente necesario para atender el hardware (lo veremos más tarde).
- ¿Cuándo surge y cuál es el problema ?
  - Cuando la ISR se ejecuta en el instante “inesperado”
  - Se produce inconsistencia de datos entre la ISR y el resto del código

Bug de los datos compartidos

- Características
  - No ocurre siempre
    - cuando se produce una interrupción entre dos instrucciones criticas
  - Baja probabilidad de ocurrencia, sin embargo ocurre:
    - cuando no se presta mucha atención
    - cuando no se está conectado para debugging
    - después que el *rover* “aterrizó” en Marte
    - en la demo con los clientes
  - Difícil de encontrar (típicamente se “arregla” solo)
  - En consecuencia: evitar este bug

Terminología

- Atómico:
  - una parte de un programa se dice atómico si no puede ser interrumpido, es decir se puede garantizar que será ejecutado como una unidad inseparable (atómica)
- Sección crítica:
  - conjunto de instrucciones que deben ser atómicas para asegurar el funcionamiento correcto.

Resumen hasta ahora...

- Problema de datos compartidos:
  - surge cuando código de “contexto main” accede datos compartidos con ISR de manera no atómica.
- Instrucciones de máquina
  - es la unidad atómica natural de ejecución
- Líneas de código C
  - muchas veces se mapea en varias líneas en assembler, entonces no son atómicas
  - si una línea de C necesita que sea atómica pasa a ser una sección critica
- ¿Cómo podemos hacer atómica una sección critica?
  - Enfoque preferido: deshabilitar y rehabilitar interrupciones
  - Veremos otros...

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;
  }
}
```

Más ejemplos

- Temporizador

timer.c

```
static int iSeconds, iMinutes, iHours;

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

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

timer.c

```
static int iSeconds, iMinutes, iHours;

#pragma vector=TIMERX
__interrupt void vUpdateTime(void){
  // ...
}
```

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

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

Tiempo de respuesta

- Interrupciones:
  - herramienta para obtener buenos tiempos de respuesta
- Concepto:
  - cantidad de tiempo que le lleva al sistema responder a una interrupción.
- ¿Cuán rápido se responde a las interrupciones?

Tiempo de respuesta

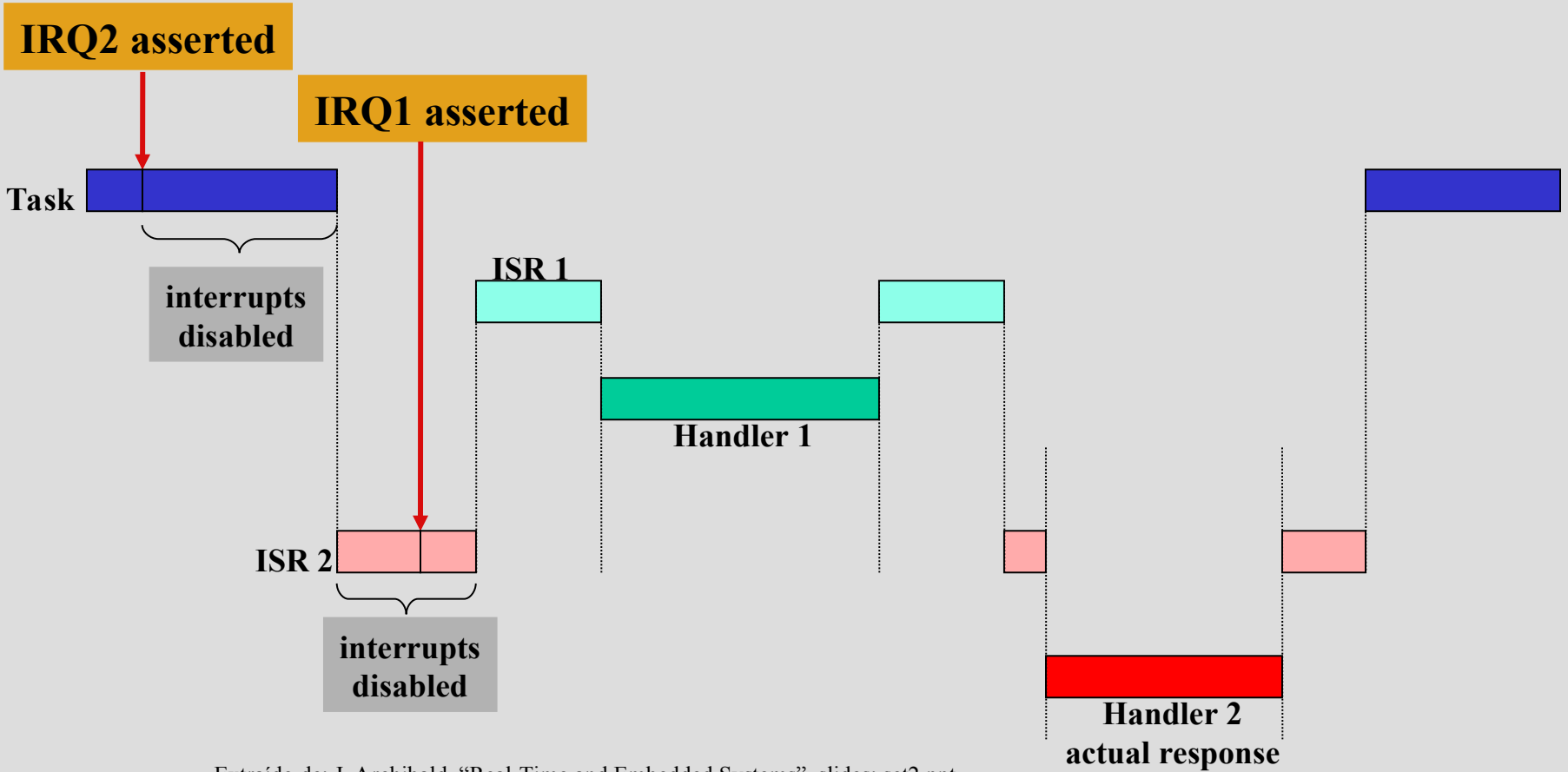
- Depende de:

  1. El mayor período de tiempo en el cual las interrupciones están deshabilitadas.
  2. El tiempo que lleva ejecutar las ISR asociadas a las interrupciones de mayor prioridad.
  3. Cuánto tiempo le lleva al microprocesador finalizar lo que está haciendo y ejecutar la ISR.
  4. Cuánto tiempo le lleva a la ISR guardar el contexto y ejecutar instrucciones hasta considerarse "respuesta”.

Análisis de los factores

- Tiempo máximo de las secciones críticas (factor 1)
  - mantenerlas cortas.
- Tiempo de ISR de mayor prioridad (factor 2)
  - asignar prioridades cuidadosamente
  - escribir subrutinas cortas
- Tiempo de respuesta del uP (factor 3)
  - fijo una vez seleccionado
- Tiempo para salvar contexto e ISR (factor 4)
  - salvar contexto: cantidad de registros
  - eficiencia de la subrutina: buena codificación

Tiempo de respuesta



Extraído de: J. Archibald, “Real-Time and Embedded Systems”, slides: set2.ppt

Próxima clase

- Alternativas a deshabilitar interrupciones para el bug de datos compartidos (clase 3)

Bibliografía

- “An Embedded Software Primer” David E. Simon, Chapter 4: Interrupts
- “MSP430x2xx Familiy User's Guide”
- Hoja de datos MSP430G2553
- MSP430 Optimizing C/C++ Compiler v18.1.0 LTS. User's Guide.