INSTITUTO DE INGENIERÍA ELÉCTRICA

FACULTAD DE INGENIERÍA UDELAR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Sistemas embebidos de bajo consumo

Sistemas embebidos para tiempo real

NEON
Network of Competence on Internet of Things

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

Objetivos

- Describir y distinguir los diferentes conceptos relacionados con el tema de “bajo consumo”.
- Identificar las fuentes de consumo de potencia en un microcontrolador (dinámicos y estático).
- Indicar las formas de disminuir su consumo.
- Escribir código que minimice el consumo.
- Estimar el consumo promedio de potencia de un microcontrolador.

Índice

- Definiciones & conceptos
- Nivel circuito
- Nivel sistema
- Programación para bajo consumo

Definiciones & conceptos

- Básicos: potencia, corriente, tensión, energía, carga, etc.
- ¿Porqué importa?
  - Disponibilidad limitada: E ! = P
- Fuentes de energía/potencia
  - Baterías
  - Recolección de energía (Energy scavening)
  - Híbrido
  - Red eléctrica (bajo consumo?)

Definiciones & conceptos

- Low {power, energy} :
  - minimizar la potencia/energía consumida
- {Power, Energy} Aware:
  - modificación del comportamiento para maximizar otra métrica de desempeño (throughput, bandwidth, QoS) restringido a la disponibilidad de energía/potencia

Nivel circuito: fuentes de consumo

- Analógico
  - Compromiso: Potencia vs. SNR
- Digital
  - Dinámico: conmutación de circuitos
    - Potencia dinámica (P_{DIN}): $P_{DIN} \propto N_{Act} C_L V_{DD}^2 f$ (1)
    - Retardo (δ): $\delta \propto C_L V_{DD} / (V_{DD} - v_T)^{\alpha}$ (2)
    - Observación:
      - Eq (1): bajo V_{DD} para aumentar f con P_{DIN} cte: **$V_{DD}\downarrow \Rightarrow$ f $\uparrow$**
      - Eq (2): pero si **$V_{DD}\downarrow$** $\Rightarrow \delta\uparrow$ (porque $\alpha > 1$) $\Rightarrow$ T$\uparrow$ y **f $\downarrow$**
        porque δ máximo < T = 1/f **¡Contrapuesto!**
  - Estático (fugas=leakage): $I_{leak} \propto 1 / v_T$ $v_T\downarrow \Rightarrow I_{leak}\uparrow$

Nivel circuito: minimizar potencia

- Dinámico
  - Clock gating
    - deshabilito reloj si la salida no es usada
  - DVFS (Dynamic Voltage and Frequency Scaling)
    - Variar V_{DD} y f para procesar la carga con la potencia justa.
    - Ecuaciones (vimos recién)
      - $f \propto V_{DD}$ $(V_{DD}\downarrow \Rightarrow \delta\uparrow \Rightarrow f\downarrow)$
      - $P \propto f$
      - $P \propto V_{DD}^2$
    - Importante $\rightarrow$ predicción de la carga
  - Hay otros...

Nivel circuito: minimizar potencia

- Estático
  - Power gating
    - STANDBY: apago tensión de alimentación
    - Arquitectura: PMOS y NMOS en serie.
    - Corriente de fuga de transistores *sleep*: $v_T \downarrow \Rightarrow I_{leak} \uparrow$
      - v_T bajo para lógica $\rightarrow$ rápidos
      - v_T alto para transistores *sleep* $\rightarrow$ baja fuga pero lentos.
    - Caída de tensión en estado ON
      - Area $\uparrow \Rightarrow V_{drop} \downarrow$
      - Area $\uparrow \Rightarrow P_{drop} \uparrow$ (potencia dinámica de los transistores *sleep*)
  - Observación:
    - Conmutar entre ACTIVO y STANDBY tiene su costo.

Nivel Sistema: introducción

- Sistema compuesto por componentes (subsistemas)
- Diferentes modos de operación
  - Mínimo: ACTIVO, STANDBY
- Ejemplo: microcontroladores
  - Subsistemas: CPU, ADC, TIMER, etc.
  - Modos operación:
    - Atmega: ACTIVE, IDLE, POWER-DOWN, POWER-SAVE, STANDBY.
    - MSP430: ACTIVE, LPM0, ... , LPM4.
- Metodo básico para minimizar potencia:
  - Ciclo de trabajos mínimos

Nivel Sistema: DPM

- Gestión dinámica de potencia (DPM)
  - Apagar componentes no usados
    - Trabajar con los mínimos componentes habilitados (ej. μC)
  - Bajar desempeño
    - Por ejemplo: bajar alimentación y/o frecuencia
  - Premisas
    - carga de procesamiento no uniforme
    - es posible predecir la carga con cierta precisión
  - Costos
    - Latencia de transición
    - Energía de transición

Actividad en grupo

- Programación de bajo consumo en MSP430
  - Objetivo:
    - Escribir un fragmento de código en una arquitectura “round-robin con interrupciones” para pasar a “LPM2” cuando no hay tareas que realizar en “background”.
    - Prestar atención a que estado se vuelve luego de una interrupción.
  - Material:
    - MSP430x2xx Family User's Guide (Rev. J). Sección 2.3
    - MSP430 Optimizing C/C++ Compiler v18.1.0.LTS User's Guide
  - Tiempo:
    - 10 minutos

Round-Robin con interrupciones

```
// Rutinas de atención a las
   interrupciones
void interrupt ISR_A() {
   !! Respuesta inicial para A
   flagA = true;
}
void interrupt ISR_B() {
   !! Respuesta inicial para B
   flagB = true;
}
void interrupt ISR_C() {
   !! Respuesta inicial para C
   flagC = true;
}
```

```
while (true) { //código en main
   if (flagA) {
         flagA = false;
         handle_eventA();
   }
   if (flagB) {
         flagB = false;
         handle_eventB();
   }
   if (flagC) {
         flagC = false;
         handle_eventC();
   }
}
```

Round-Robin con interrupciones

```
// Rutinas de atención a las
   interrupciones
void interrupt ISR_A() {
    !! Respuesta inicial para A
    flagA = true;
    Salir_de_int_despierto();
}
void interrupt ISR_B() {
    !! Respuesta inicial para B
    flagB = true;
    Salir_de_int_despierto();
}
void interrupt ISR_C() {
    !! Respuesta inicial para C
    flagC = true;
    Salir_de_int_despierto();
}
```

```
while (true) { //código en main
    if (flagA) {
            flagA = false;
            handle_eventA();
    }
    if (flagB) {
            flagB = false;
            handle_eventB();
    }
    if (flagC) {
            flagC = false;
            handle_eventC();
    }
    Si no hay flags en 1 => Go_to_sleep();
}
```

Round-Robin con interrupciones

```
// Rutinas de atención a las
   interrupciones
void interrupt ISR_A() {
   !! Respuesta inicial para A
   flagA = true;
   __low_power_mode_off_on_exit();
}
void interrupt ISR_B() {
   !! Respuesta inicial para B
   flagB = true;
   __low_power_mode_off_on_exit();
}
void interrupt ISR_C() {
   !! Respuesta inicial para C
   flagC = true;
   __low_power_mode_off_on_exit();
}
```

```
while (true) { //código en main
   if (flagA) {
           flagA = false;
           handle_eventA();}
   if (flagB) {
           flagB = false;
           handle_eventB();}
   if (flagC) {
           flagC = false;
           handle_eventC();}
  if (!flagA) && (!flagB) && (!flagC) {
     __low_power_mode_2();
  }
}
```

Round-Robin con interrupciones

```
// Rutinas de atención a las
   interrupciones
void interrupt ISR_A() {
    !! Respuesta inicial para A
    flagA = true;
   __low_power_mode_off_on_exit();
}
void interrupt ISR_B() {
    !! Respuesta inicial para B
    flagB = true;
   __low_power_mode_off_on_exit();
}
void interrupt ISR_C() {
    !! Respuesta inicial para C
    flagC = true;
   __low_power_mode_off_on_exit();
}
```

```
while (true) { //código en main
    if (flagA) {
            flagA = false;
            handle_eventA();}
    if (flagB) {
            flagB = false;
            handle_eventB();}
    if (flagC) {
            flagC = false;
            handle_eventC();}
  __disable_interrupt();
  if (!flagA) && (!flagB) && (!flagC) {
      __bis_SR_register(LPM2_bits + GIE);
      // __low_power_mode_2();
  }
  __enable_interrupt();
}
```

Nivel Sistema: DPM

- Gestión dinámica de potencia (DPM)
  - Cálculo de tiempo mínimo en STANDBY
    - Gestor de potencia ideal
  - Se puede demostrar que:
    - $T_{min} = [\ E_{AS} + E_{SA} - P_A(T_{AS} + T_{SA})\ ]\ /\ \ (P_A - P_S)$
    - $E_{AS} + E_{SA} > \ P_A(T_{AS} + T_{SA})$ se cumple siempre => T_{min} existe siempre
  - Igual no siempre va convenir irse a dormir:
    - Latencia
    - Overhead de programación
    - El ahorro en consumo tiene que valer la pena

Nivel sistema: criterios

- Greedy (voraz, codicioso, ansioso)
  - Tan pronto cuando no tenga que hacer voy a STANDBY
  - Ventajas & Desventajas
- Fixed time-out (temporizado fijo)
  - Usado ampliamente.
  - Problema de elección T_{th} (umbral)
- Predictive (predictivo)
  - Shut-down
  - Wake-up
- Observación:
  - Cuando por la aplicación se conoce el “futuro” se puede decidir si pasar a STANDBY.

Nivel sistema: bajo duty-cycle

- Potencia promedio: $\bar{P} = P_0 + \sum_{i=1} \left(P_i - P_0 \right) d_i$
  - P_i potencia consumida en modo i
  - $d_i = t_i/T$, $\Sigma(d_i)=1$
- Ejemplo: MSP430 con P_{AM} y P_{LPM3} (1 MHz y 3 V)

$$\bar{P} = \left(P_{AM} - P_{LPM3} \right) d + P_{LPM3}$$

  - $I_{AM} \sim 330\ \mu A$
  - $I_{LPM3} \sim 0.7\ \mu A$
  - Ejercicio: calcular d para que: $\bar{I}_{AM} \sim \bar{I}_{LPM3}$

Deberes

- Evaluación de DVFS en MSP430G2553
  - Objetivo:
    1) Calcular el consumo promedio de una aplicación periódica, que realiza un procesamiento cada segundo que le insume N = 160 mil ciclos de reloj (f_{CLK}= 16 MHz y V_{CC} = 3.3 V).
    2) Repetir para V_{CC} = 2.2 V y estimar el ahorro de potencia consumida (promedio).
  - Nota: considerar que tiempo activo es $\tau = N \cdot T_{CLK} = N/f_{CLK}$
  - Material:
    - MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J), págs. 21 a 23.

Bibliografía

- Libro:
  - Pedram, M. and Rabaey, J. (2002) *Power Aware Design Methodologies*. Kluwer Academic Publishers.
    - Chap. 1. Introduction (Massoud Pedram & Jan Rabaey)
    - Chap. 13. Circuit and System Level Power Management (Farzan Fallah & Massoud Pedram)
- Artículos:
  - Unsal, O.S.; Koren, I., "System-level power-aware design techniques in real-time systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1055-1069, July 2003.
  - Pedram, M., "Power optimization and management in embedded systems," *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001*, pp. 239-244, 2001.