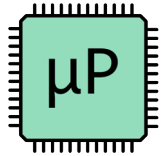
Low-power Mode

Lecture 22

Microprocessor-based Systems (E155)

Prof. Josh Brake



Outline

- Low-power mode motivation
- Low-power modes on Cortex-M4
- Low-power modes on STM32F401RE
- Demo: Blink LED with Sleep Mode

Why low power mode?

- Lots of reasons in embedded designs. Some of the most common:
  - Smaller battery size
  - Lower electromagnetic interference
  - Simpler power supply design (e.g., heat dissipation)
  - Alternative energy supply

How to measure power consumption?

- **Energy Efficiency**: work per watt (e.g., DMIPS/μW or CoreMark/μW)
- **Active current**: current per frequency μA/MHz
- **Sleep mode current**: μA as most clock signals are turned off
- **Wake-up latency**: Number of clock cycles required to go from sleep mode to resumed execution

Cortex-M4 provides many power efficiency features

- Various run/sleep modes
- Ultra low power Real-Time Clock (RTC), watchdog, and Brown-Out Detector (BOD)
- Smart peripherals that can run even in sleep mode
- Flexible clock control to disable clock for inactive parts of the design

Cortex-M4 Sleep Modes

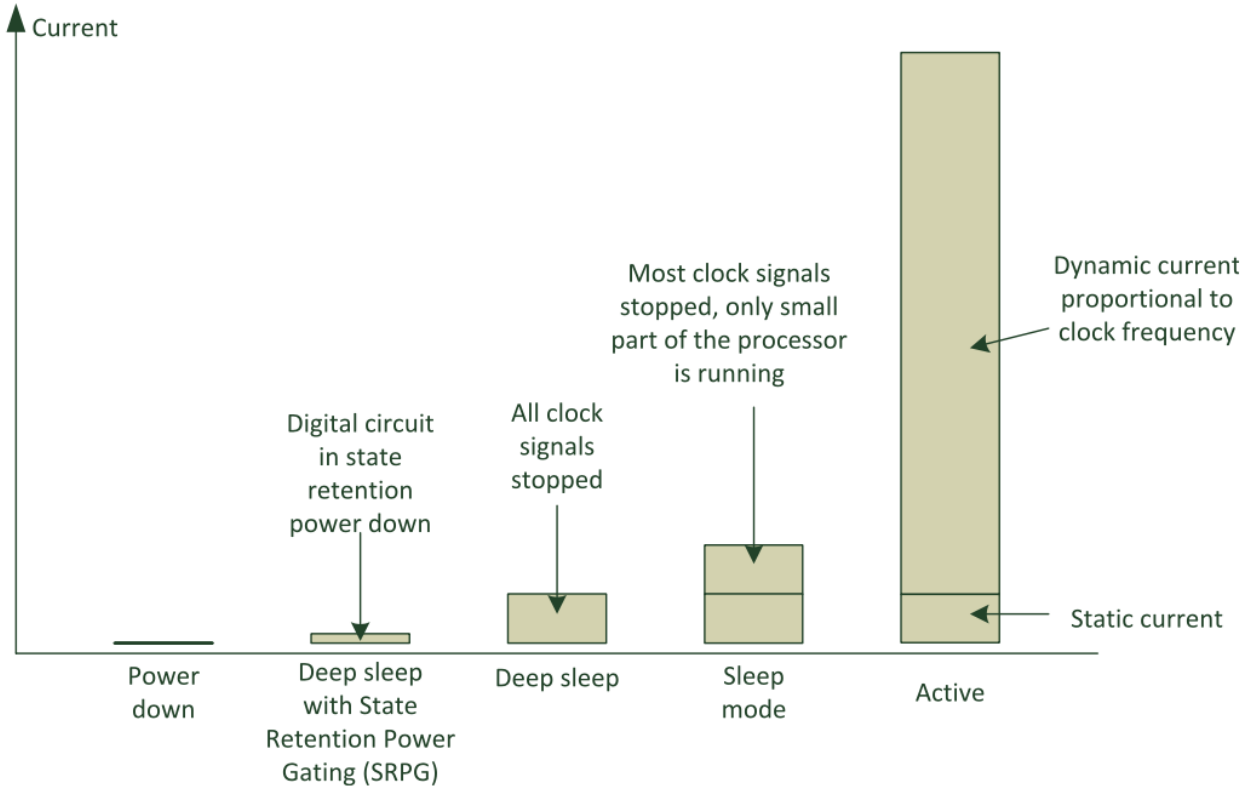


FIGURE 9.3

Various power modes including sleep modes

Figure 9.3 *The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors*

Entering Sleep Modes

Instruction	CMSIS-Core	Description
`WFI`	`void __WFI(void);`	Wait for interrupt Enter sleep mode and wake-up from interrupt request, debug request, or reset.
`WFE`	`void __WFE(void);`	Wait for Event Enter sleep mode conditionally if the event register is clear. Otherwise, clear the internal event register and continue execution. Processor can wake-up by interrupt request, **event input**, debug request, or reset.

Sleep-on-Exit Feature

- Useful for interrupt-driven applications where all operations aside from initialization take place in interrupt handlers.
- Automatically goes to sleep after returning from exception/interrupt handler and returning to Thread mode.

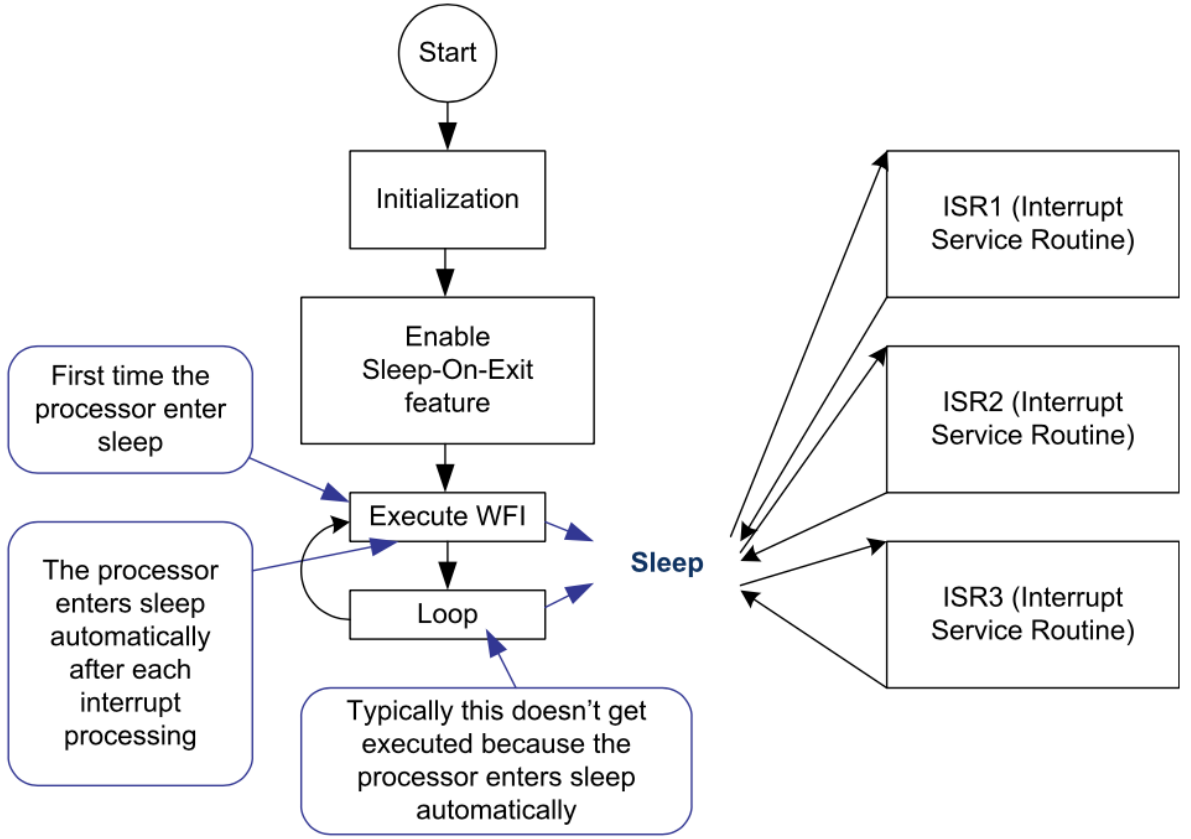


FIGURE 9.4

Sleep-on-Exit program flow

Figure 9.4 from *The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors*

Deep sleep with State Retention Power Gating (SRPG)

- Uses backup power for retaining flop states while turning off power to logic and clock buffers.

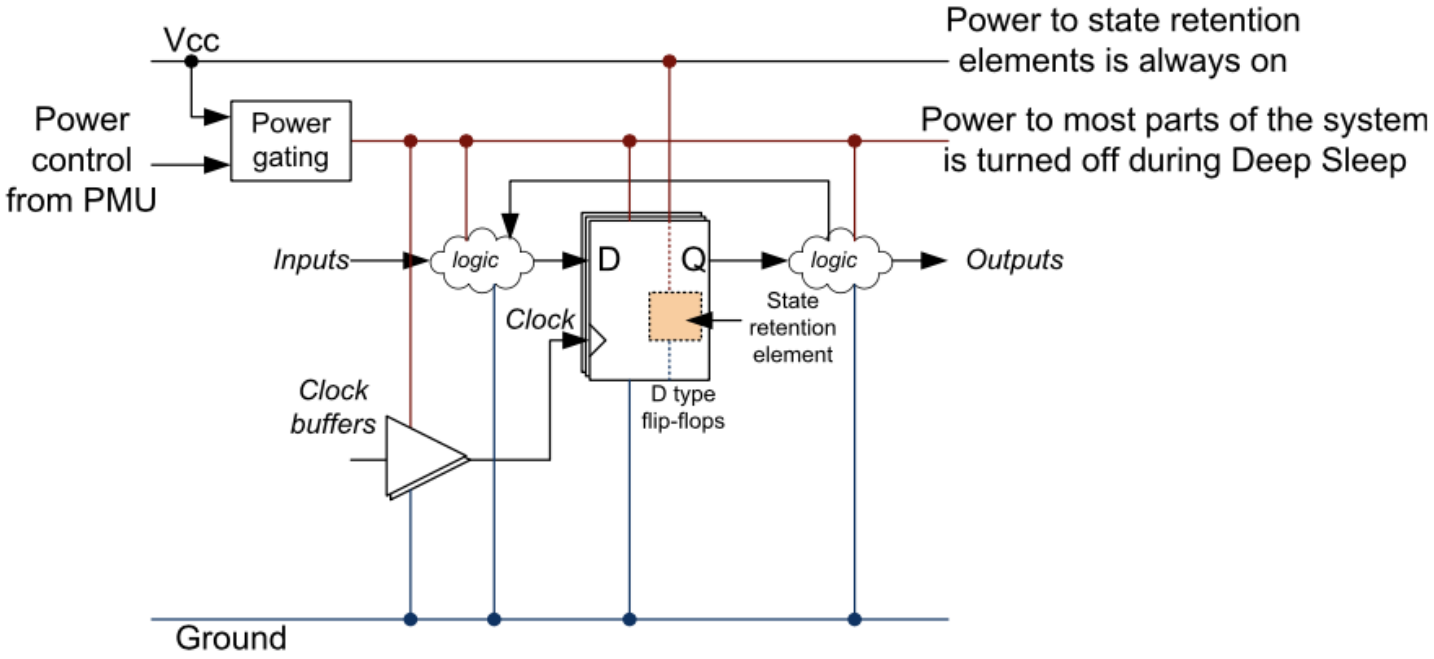


FIGURE 9.7

SRPG technology allows most parts of a digital system to be powered down without state loss

Figure 9.7 from *The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors*

Low-power Modes on the STM32F401RE

- Default mode is Run mode - CPU clocked by HCLK and program code executing
- Can reduce power consumption by:
    - Slowing down system clocks
        - Set prescaler registers in RCC
    - Gating the clocks to APBx and AHBx peripherals when they are unused
        - Turn peripheral clocks gates on in AHBxENR/APBxENR

STM32F401RE low-power modes

- **Sleep mode -** Cortex-M4 with FPU core stopped, peripherals continue running
- **Stop mode** - All clocks are stopped, PLL and oscillators disabled. Internal SRAM and registers preserved.
- **Standby mode** - 1.2 V domain powered off

STM32F401RE Low-power mode summary

- See power control registers for more detail

Table 15. Low-power mode summary

Mode name	Entry	Wakeup	Effect on 1.2 V domain clocks	Effect on V_{DD} domain clocks	Voltage regulator
Sleep (Sleep now or Sleep-on-exit)	WFI or Return from ISR	Any interrupt	CPU CLK OFF no effect on other clocks or analog clock sources	None	ON
	WFE	Wakeup event			
Stop	PDDS bit + STOP mode configuration + SLEEPDEEP bit + WFI, Return from ISR or WFE	Any EXTI line (configured in the EXTI registers, internal and external lines)	All 1.2 V domain clocks OFF	HSI and HSE oscillators OFF	Main regulator or Low-Power regulator (depends on *PWR power control register (PWR_CR)*
Standby	PDDS bit + SLEEPDEEP bit + WFI, Return from ISR or WFE	WKUP pin rising edge, RTC alarm (Alarm A or Alarm B), RTC Wakeup event, RTC tamper events, RTC time stamp event, external reset in NRST pin, IWDG reset			OFF

Using WFI

- Setup simple scenario, do setup and then enter into while loop where you execute WFI.
- Will wait for interrupt, execute ISR, and then jump back into main loop where it will execute WFI again

```
int main(void)
{

 setup_Io();
 setup_NVIC();
 …
 SCB->SCR |= 1<< 1; // Enable Sleep-on-exit feature
 while(1) {
 __WFI(); // Keep in sleep mode
 }
}
```

Danger when using WFI in this way

```
setup_timer0(); // Setup a timer to trigger a timer interrupt
NVIC_EnableIRQ(Timer0_IRQn); // Enable Timer0 interrupt at NVIC
__WFI(); // Enter sleep and wait for timer #0 interrupt
Toggle_LED();
```

- What if the interrupt takes a long time to trigger?
- What if timer is set to fire within a few cycles? Or if another interrupt occurs after the timer is configured but before WFI is executed?

Better solution

```
volatile int timer0irq_flag; // Set to 1 by timer0 ISR
 ...
 setup_timer0(); // Setup a timer to trigger a timer interrupt
 NVIC_EnableIRQ(Timer0_IRQn); // Enable Timer0 interrupt at NVIC
 if (timer0irq_flag==0) { // timer0irq_flag is set in timer0 ISR
  __WFI(); // Enter sleep and wait for timer #0 interrupt
  }
 Toggle_LED();
```

- Check a flag set in the timer ISR

Using WFE

```
volatile int timer0irq_flag;
 ...
 timer0irq_flag = 0; // Clear flag
 set_timer0();
 NVIC_EnableIRQ(Timer0_IRQn);
 while (timer0irq_flag==0) {
  __WFE(); // Enter sleep and wait for timer #0 interrupt
  };
 Toggle_LED();
```

- WFE enables conditional sleep. Here we change the sleep operation previously using WFI to an idle loop.
- WFE clears event latch

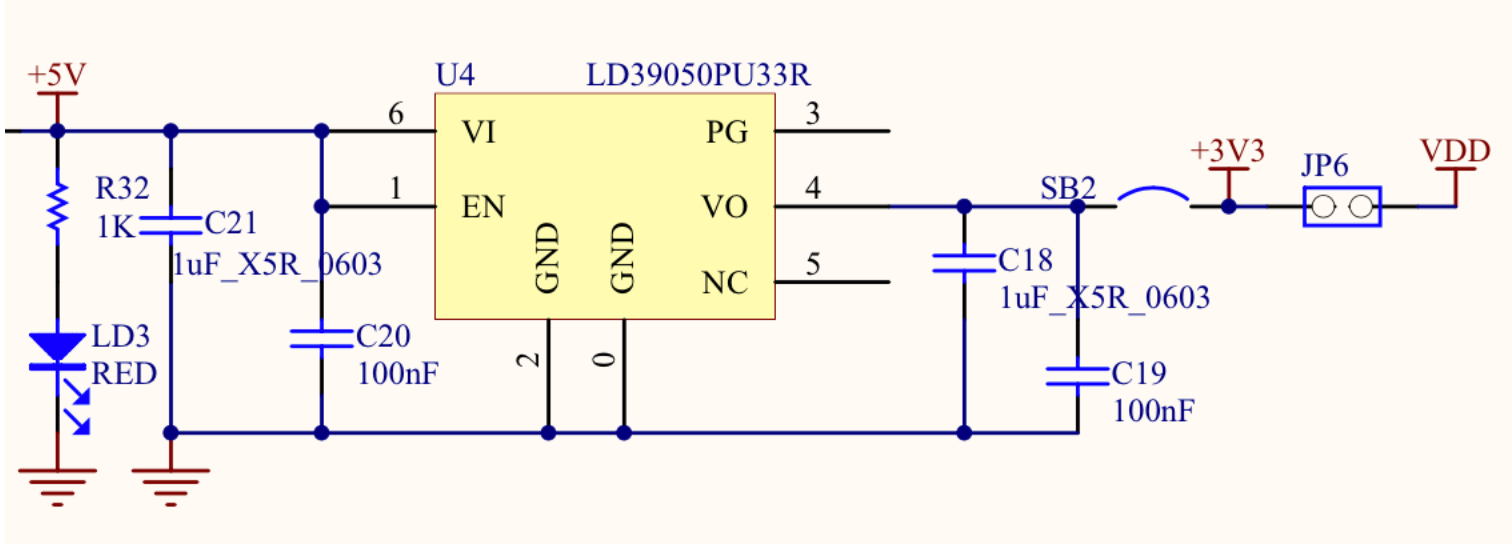
Using WFE

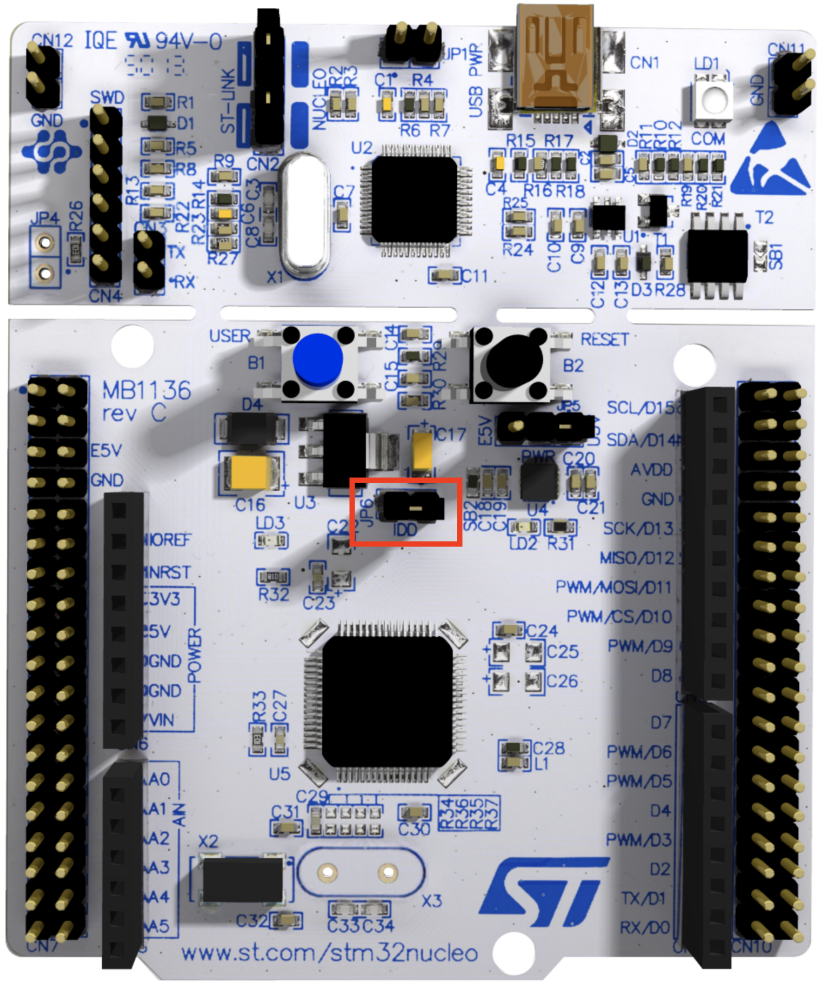
```
volatile int timer0irq_flag;
 ...
 timer0irq_flag = 0; // Clear flag
 set_timer0();
 NVIC_EnableIRQ(Timer0_IRQn);
 while (timer0irq_flag==0) {
  __WFE(); // Enter sleep and wait for timer #0 interrupt
  };
 Toggle_LED();
```

- Safer
  - If timer0 interrupt has triggered before entering the loop, we skip because software flag is set
  - If the timer0 interrupt is triggered just between the compare and WFE, the interrupt sets the internal event register and the WFE will be skipped.
  - Loop is repeated and condition checked again causing the loop to exit and toggle the LED.

Measuring Power Consumption

- Can use ammeter between IDD pins on Nucleo board





Demo

```c
// main_blink_low_power.c
// Josh Brake
// jbrake@hmc.edu
// 11/4/20

#include "stm32f4xx.h"

#define LED_PIN 5
#define DELAY_TIM TIM2


int main(void) {
    // Configure GPIO pin
    RCC->AHB1ENR |= (1 << RCC_AHB1ENR_GPIOAEN_Pos);

    GPIOA->MODER &= ~(0b11 << LED_PIN*2);
    GPIOA->MODER |= (0x01 << LED_PIN*2);

    GPIOA->BSRR = (1 << (LED_PIN + 16));

    // Initialize timer
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // TIM2EN
    uint32_t psc_div = (uint32_t) ((SystemCoreClock/1e6)-1); // Set prescaler to give 1 µs time base
    DELAY_TIM->PSC = (psc_div - 1); // Set prescaler division factor
    DELAY_TIM->EGR |= TIM_EGR_UG; // Generate an update event to update prescaler value
    DELAY_TIM->CR1 |= TIM_CR1_CEN; // Set CEN = 1
```

Demo

```
28      // Setup timer parameters
29      DELAY_TIM->ARR = 1000e3; // Set ARR to 500 ms
30      DELAY_TIM->EGR |= TIM_EGR_UG;     // Force update
31      DELAY_TIM->SR &= ~TIM_SR_UIF; // Reset UIF
32      DELAY_TIM ->CNT = 0; // Reset CNT
33
34      // Enable global interrupts
35      __enable_irq();
36
37      // Enable interrupts for TIMx
38      DELAY_TIM->DIER |= TIM_DIER_UIE;
39      NVIC_EnableIRQ(TIM2_IRQn); // IRQn 28
40
41      while(1){
42          __WFI();
43      }
44  }
45
46  void TIM2_IRQHandler(){
47      volatile int pin_val = (GPIOA->IDR >> LED_PIN) & 0x1;
48      if(pin_val) GPIOA->BSRR = (1 << (LED_PIN + 16));
49      else GPIOA->BSRR = (1 << LED_PIN);
50      DELAY_TIM->SR &= ~TIM_SR_UIF;
51  }
```

Demo

- Try commenting out the WFI line and changing the delay of the timer.
- With WFI commented out, the processor will just spin in the loop continuing to consume power.
- WFI will put it to sleep until the interrupt triggers
- Other variables to test
  - Different clock speeds
  - Different timer delays

https://github.com/joshbrake/E155_FA2020/tree/master/L22/Low-power_Interrupt_Demo

Summary

- Power consumption is an important consideration for embedded systems since we often are running on battery
- The STM32F401RE provides four main power modes: run, sleep, stop, and standby (listed in order of decreasing power consumption).
- The WFI and WFE instructions/functions provided in CMSIS enable easy use of sleep modes along with configuration registers.
- Interrupts can lead to subtle bugs due to their asynchronous nature. Watch out!

Lecture Feedback

- What is the most important thing you learned in class today?
- What point was most unclear from lecture today?

https://forms.gle/Ay6MkpZ6x3xsW2Eb8