CHAPTER 7:
EMBEDDED PERIPHERALS

M. Jiménez, R. Palomera, & I. Couvertier

INTRODUCTION TO EMBEDDED SYSTEMS:
Using Microcontrollers and the MSP430

© M. Jiménez et al. 2014
OUTLINE

- Fundamental Interrupt Concepts
- Interrupt Handling in the MSP430
- Interrupt Software Design
- Timers and Event Counters
- Embedded Memory Technologies
- Bus Arbitration and DMA Transfers
7.1 FUNDAMENTAL INTERRUPT CONCEPTS
WHAT IS AN INTERRUPT?

- A signal indicating the occurrence of an event that needs immediate CPU attention
- Provides for a more efficient event handling than using polling
  - Less CPU cycles wasted
- Advantages
  - Compact & Modular Code
  - Allow for Reduced Energy Consumption
  - Faster Multi-event Response Time
INTERRUPT TRIGGERS

- **Hardware Triggers**
  - Caused by external hardware components
  - **Sensitivity Level**
    - Edge Triggered
      - Rising edge
      - Falling edge
    - Level Triggered
      - Caused a logic level (High or Low)

- **Software Triggers**
  - Caused by a software event in user’s program

- **CPU Exception**
  - Internal CPU state
Maskable Interrupts

- Can be blocked through a flag
  - Global Interrupt Flag (GIE)
  - Local Flags in Peripheral Interfaces
- Most common type of interrupt
- Disabled upon RESET
- By default disabled upon entrance into an ISR

Non-maskable Interrupts (NMI)

- Cannot be masked, thus are always served
- Reserved for system critical events
INTRODUCTION TO EMBEDDED SYSTEMS: Using Microcontrollers and the MSP430

INTERRUPT SERVICE SEQUENCE

1. Interrupt request
2. CPU saves PC, PSW and clears GIE
3. PC loaded with ISR address
4. ISR executed
5. Executing IRET restores PSW and PC
6. Interrupted program resumes

Fig. 7.1: Sequence of events in the CPU upon accepting an interrupt request.

© 2014 by M. Jiménez, R. Palomera, & I. Couvertier

Chapter 7: 7
**Non-vectored Systems**

- Single, multidrop interrupt request line
- Single ISR for all devices
- CPU identifies source by polling service request (SRQ) flags

---

**Fig. 7.2:** Device interfaces in a system managing non-vectored interrupts
 Vectored Interrupts
  - Require an Interrupt Acknowledgment (INTA) cycle
  - Interfaces generate an ID number (vector) upon INTA
  - ID number allows calculating ISR location

 Auto-vectored Interrupts
  - Each device has a fixed vector or fixed ISR address
  - No INTA cycle or vector issuing required
  - CPU loads direct ISR address into PC to execute ISR
  - Method used in MSP430 MCUs
INTRODUCTION TO EMBEDDED SYSTEMS: Using Microcontrollers and the MSP430

PRIORITY HANDLING (1/3)

- Interrupt Priority Management
  - Strategy to resolve multiple, simultaneous interrupt requests
  - Priority scheme decides which one is served first

- Non-vectored Systems
  - Polling order of SRQ flags decides priority

- Vectored & Auto-vectored Systems
  - Hardware supported
  - Daisy Chain-based
  - Interrupt Controller-based
PRIORITY HANDLING (2/3)

- Daisy Chain-based Arbitration
  - Devices linked by a daisy-chain element
    - Simple to implement
  - Hardwired priorities
    - The closer the device to the CPU the higher the priority

![Diagram of Daisy Chain-based Arbitration](image)

**Fig. 7.3**: Device interfaces connected in a daisy chain

**Fig. 7.4**: Daisy chain logic
Interrupt Controller-based Arbitration
- Uses central arbiter for resolving priorities
- Reduces interface overhead for vectored systems
- Allows for configuring the priority scheme

Fig. 7.5: Interrupt management using a programmable interrupt controller
7.2 INTERRUPT HANDLING IN THE MSP430

- System Resets
- (Non)-Maskable Interrupts
- Maskable Interrupts
All Internal Peripherals are Interrupt Capable
- Use an internal daisy-chain scheme
- Support low-power operation

Auto-vectored Approach
- Fixed interrupt table
- Vector addresses depend on MCU family member

Types of MSP430 Interrupts
- System Resets
- Non-maskable
- Maskable
SYSTEM RESETS

- Have the highest system priority
- Truly non-maskable
- Triggered by different events (multi-sourced)
  - Power-up,
  - External Reset,
  - Watchdog Timer,
  - FLASH key violation, etc.
- Vector at address 0FFFEh
- Saves no PC and no PSW
  - Non returning ISR
  - Calls bootstrap sequence

Fig. 6.28 Basic reset module in MSP430 devices
(NON)-MASKABLE INTERRUPTS

- Pseudo NMI operation
  - Cannot be masked through GIE flag
  - CAN be masked through their respective flags
- Vector address at 0FFCh
  - Multi-sourced Vector
- NMI Trigger Sources
  - An edge on the RST/NMI pin
    - When NMI mode
  - An oscillator fault
  - A FLASH key access violation

Fig. 7.6 Flowchart for an NMI handler
(Courtesy of Texas Instruments, Inc.)
MASKABLE INTERRUPTS

- Most common user type interrupt
  - All sources other than Reset & NMI
- Masked by GIE flag
  - All triggers also have individual enable bits in source peripherals
  - Both GIE and local must be enabled
- Fixed priorities
  - See device interrupt table for details
- External IRQ pins provided via GPIO Ports
  - Most devices support interrupts via P1 & P2
  - Single vector per port
  - Individual status and enable bits per pin
7.3 INTERRUPT SOFTWARE DESIGN

- Interrupt Programming
- Examples of Interrupt-Based Programs
- Multi-Source Interrupt Handlers
- Dealing with False Triggers
- Interrupt Latency and Nesting
- Interrupts and Low-Power Modes
- Working with MSP430 Low-Power Modes
1) Stack Allocation
   - Is where CPU saves the SR and PC
   - Automatically allocated by C-compiler

2) Vector Entry Setup
   - Specify entry in vector table

3) Provide the Actual ISR Code
   - Short and quick
   - Register transparent (in ASM)
   - Do not use parameter passing or value return
   - In ASM, always end with RETI

4) Enable Interrupt Flags
   - Both GIE and local enable
interrupts in assembly language

; Code assumes push-button in P1.3 is hardware debounced and wired to
; produce a high-to-low transition when depressed.
                         ----------------------------------(1)
#include "msp430g2231.h"
                         ----------------------------------(1)
RSEG CSTACK ; Stack declaration
RSEG CODE ; Executable code begins
                         ----------------------------------
Init MOV.W #SFE(CSTACK),SP ; Initialize stack pointer
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop the WDT
                         ---------------------------------- Port1 Setup ----------------------------------
BIS.B #0F7h,&P1DIR ; All P1 pins but P1.3 as output
BIS.B #BIT3,&P1REN ; P1.3 Resistor enabled
BIS.B #BIT3,&P1OUT ; Set P1.3 resistor as pull-up
BIS.B #BIT3,&P1IES ; Edge sensitivity now H->L
BIC.B #BIT3,&P1IFG ; Clears any P1.3 pending IRQ
Port1IE BIS.B #BIT3,&P1IE ; Enable P1.3 interrupt
                         ---------------------------------- Main ----------------------------------
Main BIS.W #CPUOFF+GIE,SR ; CPU off and set GIE
NOP ; Debugger breakpoint
                         ---------------------------------- PORT1_ISR ; Begin ISR ----------------------------------
BIC.C #BIT3,&P1IFG ; Reset P1.3 Interrupt Flag
XOR.B #BIT2,&P1OUT ; Toggle LED in P1.2
RETI ; Return from interrupt
                         ---------------------------------- Reset and Interrupt Vector Allocation ----------------------------------
ORG RESET_VECTOR ; MSP430 Reset Vector
DW Init ;
ORG PORT1_VECTOR ; Port.1 Interrupt Vector
DW PORT1_ISR ; -------(2)
END
#include <msp430g2231.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    P1DIR |= 0xF7; // All P1 pins as out but P1.3
    P1REN |= 0x08; // P1.3 Resistor enabled
    P1OUT |= 0x08; // P1.3 Resistor as pull-up
    P1IES |= 0x08; // P1.3 Hi->Lo edge selected
    P1IFG &= 0x08; // P1.3 Clear any pending P1.3 IRQ
    P1IE |= 0x08; // P1.3 interrupt enabled

    _bis_SR_register(LPM4_bits + GIE); // Enter LPM4 w/GIE enabled
}

#pragma vector = PORT1_VECTOR // Port 1 vector configured

__interrupt void Port_1(void) // The ISR code
{
    P1OUT ^= 0x04; // Toggle LED in P1.2
    P1IFG &= 0x08; // Clear P1.3 IFG
}
Multiple Events Served by a Single Vector

- Single ISR for all events
- Specific event flags need explicit clear
- ISR code must identify actual trigger source

Source Identification

- Polling ALL interrupt request flags
- Via calculated branching

Fig. 7.7: ISR for a multi-source interrupt vector. Identification via polling all flag sources
USING CALCULATED BRANCHING

; Pseudo code for Multi source Interrupt Service Routine for four events
; Assumes FlagReg contains prioritized, encoded IRQ flags organized as:
; 0000 - No IRQ  0004 - Event2  0008 - Event4
; 0002 - Event 1  0006 - Event3

#include "headers.h"    ; Header file for target MCU
...
Preliminary declarations and code

MultiSrcISR
ADD &FlagReg,PC       ; Adds Interrupt Flag Register contents to PC
JMP Exit             ; Flags = 0: No interrupt
JMP Event1           ; Flags = 2: Event1
JMP Event2           ; Flags = 4: Event2
JMP Event3           ; Flags = 6: Event3
Event4 Task 4 starts here ; Flags = 8: Event4
...
JMP Exit             ; Exit ISR

Event1 Task 1 starts here ; Vector 2
...
JMP Exit             ; Exit ISR

Event2 Task 2 starts here ; Vector 4
...
JMP Exit             ; Exit ISR

Event2 Task 3 starts here ; Vector 4
...
JMP Exit             ; Exit ISR

Exit RETI; Return

- Calculation Methods
  - Look-up table-based
  - Via encoded flags (shown)
DEALING WITH FALSE TRIGGERS

- Interrupt Signals Triggered by Unwanted Events
  - Cause undesirable effects

- Causes of False Interrupt Triggers
  - Power glitches
  - Electromagnetic Interference (EMI)
  - Electrostatic Discharges
  - Other noise manifestations

- Mitigation Mechanisms
  - Provide dummy ISR for unused sources
  - Write a “False ISR Handler”
  - Use Watchdog Timers
  - Consider the use of polling
Interrupt Latency
- Amount of time from IRQ to fetch of first ISR instruction
- Negligible in most applications
  - An issue in fast, time sensitive real-time systems
  - Affected by Hardware & Software factors

Hardware Factors
- Propagation delays in IRQ and acknowledgment paths
- Source identification method

Software Factors
- Scheduling & Priority Scheme
- ISR coding style

Recommendations
- Minimize number of stages in IRQ & ACK path (if possible)
- Use vectored schemes with in-service hardware tracking
- Keep ISRs short & quick
- Avoid service monopolization (prioritization Vs. nesting)
INTERRUPT NESTING

- Achieved by re-enabling the GIE within an ISR
- Use only when strictly necessary
  - Most applications don’t need it

Recommendations

- Establish a strict prioritization
- Exert SW stack depth control
- Whenever possible, avoid re-entrancy
- Avoid static variables – self-modification risk
- Do not use self-modifying code
- Re-entrant code shall only call re-entrant functions
INTRODUCTION TO EMBEDDED SYSTEMS: Using Microcontrollers and the MSP430

INTERRUPTS AND LOW-POWER MODES

- Programming for Low-power Consumption
  - Initialize the system
  - Activate interrupts
  - Enable a low-power mode

- Interrupts wake CPU
  - No energy wasted waiting for events
  - System active only when needed
  - All tasks performed within ISRs
  - LPM automatically restored upon IRET

Fig. 7.8 Flowchart of a main program using a low-power mode and a single event ISR.
COPING WITH COMPLEX EVENTS

- Main Suspended by LPM Activation
- ISRs only Activate Service Indicators
  - Kill LPM from within
- After LPM, Insert Main Code to Detect Triggering Source
  - Clear event flags
  - Render event service
- Ensure no Event Goes un-served
  - Observe event priorities
  - Reactivate LPM
**INTRODUCTION TO EMBEDDED SYSTEMS: Using Microcontrollers and the MSP430**

### MSP430 LOW-POWER MODES

**Fig. 7.9:** Supply current for MSP430F21x1 devices in their different operating modes *(Courtesy of Texas Instruments, Inc.)*

![Supply current graph](image)

**Table 7.2: MSP430 Operating Modes** *(Courtesy of Texas Instruments, Inc.)*

<table>
<thead>
<tr>
<th>SCG1</th>
<th>SGGO</th>
<th>OSC OFF</th>
<th>CPU OFF</th>
<th>Mode</th>
<th>CPU &amp; Clock Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Active</td>
<td>CPU and all enabled clocks are active. Default mode upon power-up</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LPM0</td>
<td>CPU &amp; MCLK disabled, SMCLK &amp; ACLK active</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>LPM1</td>
<td>CPU &amp; MCLK disabled. DCO &amp; DC generator off if DCO is not used for SMCLK. ACLK active</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LPM2</td>
<td>CPU, MCLK, SMCLK, &amp; DCO disabled DC generator enabled &amp; ACLK active</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>LPM3</td>
<td>CPU, MCLK, SMCLK, &amp; DCO disabled DC generator disabled, ACLK active</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>LPM4</td>
<td>CPU and all clocks disabled</td>
</tr>
</tbody>
</table>

© 2014 by M. Jiménez, R. Palomera, & I. Couvertier

Chapter 7: 29
7.4 TIMERS AND EVENT COUNTERS
A Binary Counter Driven by a Periodic Signal

- Mux: Clock source selector
- Prescaler: Clock frequency divider
- Counter: $n$-bit binary counter
- Comparator: compares counter output Vs. compare register

Fig. 7.11: Components of a base timer
OVERFLOW VS. OUTPUT COMPARE

**Overflow Output Operation**

Fig. 7.12 Overflow signal obtained from a 16-bit timer.

**Output Compare Operation**

Fig. 7.13 Output compare signal obtained when loading the compare register with a value of three (3)
**Interval Timer**
- Measures the time elapsed after $k$ clock cycles
- As the clock period $T$ is known, the time interval is $kT$

**Event Counter**
- Counts the occurrence of $k$ external events
- The clock is driven by the signal marking the external event

![Diagram](image)

**Fig. 7.14:** Periodic interval of 4 clock Vs. counting four aperiodic events
EXTENDING THE TIMER COUNT

- Cascading Multiple Timers

- Using Software Variable
Watchdog Timers (WDT)
- Monitor events within an expiration intervals

Real-time Clocks (RTC)
- Measure time in seconds, minutes, hours and days

Baud Rate Generators (BRG)
- Provide periodic signal for serial channels

Pulse-width Modulation (PWM)
- Duty cycle control in periodic signals
Expect the occurrence of an event within a certain interval
- On time event arrival restarts or cancels WDT (event ISR)

If WDT expires before event, a default action is executed
- Default action performed by WDT ISR
- **Basic Features**
  - 16-bit timer
  - Can be used as WDT or as interval timer
    - Sourced from ACLK or SMCLK
  - Password protected
    - All accesses must have key 05Ah in upper byte
    - Violations cause a PUC

- **WDT Operation**
  - Default mode upon Reset
    - User programs either use it for reliable SW control or cancel it upon reset
  - Restarts system upon a software problem
    - Max count: 32768 cycles

- **Interval Timer Operation**
  - Configured via WDTTMSSEL
  - Provides programmable periodic interrupts
MSP430 WDT STRUCTURE

Fig. 7.21: MSP430x2xx Watchdog Timer block diagram (Courtesy of Texas Instruments, Inc.)
# MSP430 WDT Variations

Table 7.3: MSP430 watchdog timer generational variations

<table>
<thead>
<tr>
<th>WATCHDOG TIMER</th>
<th>x3xx</th>
<th>x4xx</th>
<th>x1xx</th>
<th>x2xx</th>
<th>x5xx x6xx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software selectable time intervals</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Watchdog mode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interval mode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Password protected write</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Selectable clock source</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Power conservation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Clock fail-safe feature</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RST/NMI pin function control</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
STA: REAL-TIME CLOCKS (RTC)

- Timer to Measure Seconds, Minutes, Hours, Etc.
  - Includes registers for each time unit

- Chronometer Type
  - Measure fractions of a second

- Calendar type (RTCC)
  - Measure days, weeks, months, and years

- Clock Source Dependence
  - Accuracy depends on CLK frequency
  - Sweet frequencies preferred (32768KHz)
MSP430 REAL-TIME CLOCK MODULE

- **Basic Features**
  - 32-bit timer with selectable clock sources
  - Can be used as calendar or counter
  - Full featured register set
    - Seconds, Minutes, Hours, Day of the week, day of the month, & year
  - Hex or BCD output format

- **Calendar Mode**
  - Features full set of registers
  - Leap year capable from 1901 to 2099
  - Timer 1 (x4xx) used as pre-divider of ACLK source

- **Counter Mode**
  - Four 8-bit counters provide a 32-bit timer operation
  - Software-defined functionality
MSP430 RTC MODULE STRUCTURE

Fig. 7.18: MSP430x4xx Real-time Clock Module block diagram (Courtesy of Texas Instruments, Inc.)
Timers Can Provide Clock Base for Baud Rate
- Directly impacts bit time
- Main baud clock for simple UART modules
- Dedicated Timer Channel Usage

Baud Rate Accuracy Dependence
- Sweet frequencies preferred

 STA: BAUD RATE GENERATION

\[
\text{baud rate} = \frac{f_{\text{clk}}}{PS \times TopCount}
\]

- Avoid \( f_{\text{clk}} \) values producing large fractional quotients

\[ PS = \text{Prescaler Value} \]
\[ TopCount = \text{Compare Value} \]
\[ f_{\text{clk}} = \text{Clock Frequency} \]
STA: PULSE WIDTH MODULATION (PWM)

- **Timer Application for Controlling**
  - Duty Cycle and
  - Frequency of a Periodic Signal

- **Applications**
  - Data Encoding
  - Voltage Regulation
  - Motor Control
  - Tone Generation
  - Power & Energy Delivery Control

- **Control Parameters**
  - Top Count (Duty Cycle)
  - Counter Timer (Resolution)
  - Frequency (System Response Time)
PWM STRUCTURE & WAVEFORMS

Fig. 7.19: Pulse-width modulation module: (a) hardware structure, (b) signal output

(a)

(b)

© 2014 by M. Jiménez, R. Palomera, & I. Couvertier
Example 7.9: Controlling the brightness of a 240lm, 720mA LED using 8 different PWM levels. LED brightness is a function of the average current intensity passing through the LED junction.

Fig. 7.20: LED connection diagram and required duty cycle values

<table>
<thead>
<tr>
<th>Brightness (lm)</th>
<th>Duty Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>30</td>
<td>12.5%</td>
</tr>
<tr>
<td>60</td>
<td>25.0%</td>
</tr>
<tr>
<td>90</td>
<td>37.5%</td>
</tr>
<tr>
<td>120</td>
<td>50.0%</td>
</tr>
<tr>
<td>150</td>
<td>62.5%</td>
</tr>
<tr>
<td>180</td>
<td>75.0%</td>
</tr>
<tr>
<td>210</td>
<td>87.5%</td>
</tr>
<tr>
<td>240</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

*High count values assuming an 8-bit counter: 0, 32, 64, 96, 128, 160, 192, or 224*
16-bit Timer/Counter
3-bit Prescaler
3 Capture/Compare Registers
Four Modes
  - Stop, Up, Continuous, Up/Down
Selectable Clock Source
Configurable Outputs
PWM Capable

Fig. 7.22: MSP430 Timer_A block diagram
(Courtesy of Texas Instruments, Inc.)
INTRODUCTION TO EMBEDDED SYSTEMS: Using Microcontrollers and the MSP430

**TIMER_A OPERATING MODES**

- **Stop**: Timer Halted
- **Up**: Repeatedly Counts from 0 to TACCR0
- **Continuous**: Repeatedly Counts from 0 to 0FFFFh
- **Up/Down**: Repeatedly Counts from 0 to TACCR0 and back to 0
Common Applications
- Measuring time intervals
- Computing the speed or frequency of external events

Capture Inputs (CCIxA & CCIxB) Triggers
- Internal or external signals, or by software
- Edge sensitivity configurable to rise, fall, or both

Trigger Effects
- Timer value copied into TACCRx register
- Interrupt flag is set

Fig. 7.27: Timer_A operating in capture mode (Courtesy of Texas Instruments, Inc.)
Common Applications

- Generating PWM signals
- Producing interrupts at specific time intervals

Events Triggered by TAR Reaching TACCRx Value

- Interrupt flag CCIFG is set and $EQU_x = 1$
- $EQU_x$ affects the output according to the output mode
- The input signal CCI is latched into SCCI

Output Unit (OU)

- Makes timed signal available on I/O pins
- One output unit per capture/compare block
- Eight configurable output modes per output
## OUTPUT UNIT MODES

Table 7.5: Output unit modes *(Courtesy of Texas Instruments, Inc.)*

<table>
<thead>
<tr>
<th>MODx</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Output</td>
<td>The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated</td>
</tr>
<tr>
<td>001</td>
<td>Set</td>
<td>Output set when timer reaches the TACCRx value. It remains set until a timer reset, or until another mode affecting the output is selected</td>
</tr>
<tr>
<td>010</td>
<td>Toggle/Reset</td>
<td>The output is toggled when the timer counts to the TACCRx value. It is reset when the timer counts to the TACCO value</td>
</tr>
<tr>
<td>011</td>
<td>Set/Reset</td>
<td>The output is set when the timer counts to the TACCRx value. It is reset when the timer counts to the TACCO value</td>
</tr>
<tr>
<td>100</td>
<td>Toggle</td>
<td>The output is toggled when the timer counts to the TACCRx value. The output period is double the timer period</td>
</tr>
<tr>
<td>101</td>
<td>Reset</td>
<td>Output reset when timer reaches the TACCRx value. It remains reset until another output mode is selected and affects the output</td>
</tr>
<tr>
<td>110</td>
<td>Toggle/Set</td>
<td>The output is toggled when the timer counts to the TACCRx value. It is set when the timer counts to the TACCO value</td>
</tr>
<tr>
<td>111</td>
<td>Reset/Set</td>
<td>The output is reset when the timer counts to the TACCRx value. It is set when the timer counts to the TACCO value</td>
</tr>
</tbody>
</table>
OUTPUT WAVEFORM EXAMPLES

Timer_A in UP Mode

0FFFFh
TACCR0
TACCR1

0h

EQU0
TAIFG
EQU1
EQU0
TAIFG
EQU1
EQU0
TAIFG

Output Mode 1: Set
Output Mode 2: Toggle/Reset
Output Mode 3: Set/Reset
Output Mode 4: Toggle
Output Mode 5: Reset
Output Mode 6: Toggle/Set
Output Mode 7: Reset/Set

Interrupt Events

Timer_A in Up/Down Mode

0FFFFh
TACCR0
TACCR2

0h

TAIFG
EQU2
EQU0
TAIFG
EQU2
EQU0

Output Mode 1: Set
Output Mode 2: Toggle/Reset
Output Mode 3: Set/Reset
Output Mode 4: Toggle
Output Mode 5: Reset
Output Mode 6: Toggle/Set
Output Mode 7: Reset/Set

Interrupt Events

Figure 7.28: Output unit waveforms (Courtesy of Texas Instruments, Inc.)
PWM CODING EXAMPLE

Example 7.13: This program generates one PWM output on P1.2 using Timer_A configured for up mode. The value in CCR0, 512-1, defines the PWM period and the value in CCR1 the PWM duty cycles. A 75% duty cycle is on P1.2. ACLK = n/a, SMCLK = MCLK = T ACLK = def ault DCO

```c
;-------------------------------------------------------------------------------
;MSP430G2xx1 Demo - Timer_A, PWM TA1, Up Mode, DCO SMCLK
;By D. Dang - (c) 2010 Texas Instruments, Inc.
;-------------------------------------------------------------------------------
#include <msp430.h>

ORG 0F800h ; Program Reset

RESET mov.w #0280h,SP ; Initialize stack pointer
    StopWDT mov.w #WDTPW+WDTHOLD,&WDTCTL ; Stop WDT
    SetupP1 bis.b #00Ch,&P1DIR ; P1.2 and P1.3 output
    bis.b #00Ch,&P1SEL ; P1.2 and P1.3 TA1/2 options
    SetupC0 mov.w #512-1,&CCR0 ; PWM Period
    SetupC1 mov.w #OUTMOD_7,&CCTL1 ; CCR1 reset/set
    mov.w #384,&CCR1 ; CCR1 PWM Duty Cycle
    SetupTA mov.w #TASSEL_2+MC_1,&TACTL ; SMCLK, up mode

Mainloop bis.w #CPUOFF,SR ; CPU off
    nop ; Required only for debugger

; Interrupt Vectors
;-------------------------------------------------------------------------------
ORG 0FFFEh ; MSP430 RESET Vector
DW RESET
END
```

© 2014 by M. Jiménez, R. Palomera, & I. Couvertier
## TIMER_A VARIATIONS

Table 7.3: MSP430 Timer_A generational variations

<table>
<thead>
<tr>
<th>TIMERR_A</th>
<th>x3xx</th>
<th>x4xx</th>
<th>x1xx</th>
<th>x2xx</th>
<th>x5xx x6xx</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit timer/counter w/4 operating modes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Selectable, configurable clock source</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Independently configurable CCRs</td>
<td>5</td>
<td>3 or 5</td>
<td>3</td>
<td>2 or 3</td>
<td>up to 7</td>
</tr>
<tr>
<td>PWM output capability</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Asynchronous I/O latching</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interrupt vector register</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Second Timer_A</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
OTHER MSP430 TIMER RESOURCES

- **Basic Timer**
  - Available in legacy x3xx and x4xx devices
  - Two independent, cascadable, interrupt-driven 8-bit timers

- **Timer_B**
  - Same as Timer_A with seven CCRs

- **Watchdog Timer +**
  - Similar to WDT discussed earlier

- **Real-time Clock**
  - Available in x4xx, x5xx, and x6xx devices
  - 32-bit counter with calendar function
  - Seconds, minutes, hours, DOW, DOM, month, year (w/leap)
  - Selectable BCD output
7.5 EMBEDDED MEMORY TECHNOLOGIES

- A Classification of Memory Technologies
- Flash Memory: General Principles
- MSP430 Flash Memory and Controller
### Embedded Memory Characteristics

<table>
<thead>
<tr>
<th>Control-dominated Applications</th>
<th>High-performance Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Program Memory</td>
<td>- Program Memory</td>
</tr>
<tr>
<td>- Non-volatile</td>
<td>- Large amount</td>
</tr>
<tr>
<td>- Small to moderate amount</td>
<td>- Non-volatile</td>
</tr>
<tr>
<td>- Data Memory</td>
<td>- Re-writable for firmware upgrade</td>
</tr>
<tr>
<td>- Run-time Variables</td>
<td>- Data memory</td>
</tr>
<tr>
<td>- Volatile OK</td>
<td>- Large amount</td>
</tr>
<tr>
<td>- Small amount</td>
<td>- Non-volatile</td>
</tr>
<tr>
<td>- System Parameters</td>
<td>- Media processing applications</td>
</tr>
<tr>
<td>- Non-volatile</td>
<td>- Volatile OK</td>
</tr>
<tr>
<td>- Very small</td>
<td></td>
</tr>
<tr>
<td>- Data Storage (rarely used)</td>
<td>- Data Storage</td>
</tr>
<tr>
<td>- Large</td>
<td>- Very Large</td>
</tr>
<tr>
<td>- Non-volatile</td>
<td>- Non-volatile</td>
</tr>
</tbody>
</table>

© 2014 by M. Jiménez, R. Palomera, & I. Couvertier

Chapter 7: 57
## MEMORY TECHNOLOGIES CLASSIFICATION

<table>
<thead>
<tr>
<th>Memory Technology</th>
<th>Cell Structure</th>
<th>Volatility</th>
<th>Write-ability</th>
<th>Storage Permanence</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROM</td>
<td>Mask Programmed</td>
<td>Non Volatile ROM</td>
<td>None Factory Programmed</td>
<td>Highest</td>
</tr>
<tr>
<td>OTPROM</td>
<td>Fuse-based</td>
<td>Non Volatile ROM</td>
<td>One time only blowing fuses</td>
<td>Highest</td>
</tr>
<tr>
<td>EPROM</td>
<td>Floating-Gate MOSFET</td>
<td>Non Volatile ROM</td>
<td>Erase-Program Cycle UV Erasable</td>
<td>Very High 20 Years</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Floating-Gate MOSFET</td>
<td>Non Volatile ROM</td>
<td>Erase-Program Cycle Electrically Erasable</td>
<td>Very High 20 Years</td>
</tr>
<tr>
<td>FLASH</td>
<td>Floating-Gate MOSFET</td>
<td>Non Volatile ROM</td>
<td>Erase-Program Cycle Electrically Erasable</td>
<td>Very High 20 Years</td>
</tr>
<tr>
<td>Static RAM</td>
<td>Flip-Flop</td>
<td>Volatile RAM</td>
<td>Highest Read-write</td>
<td>Low Lost w/power down</td>
</tr>
<tr>
<td>Dynamic RAM</td>
<td>Capacitor Charge</td>
<td>Volatile RAM</td>
<td>Highest Read-write</td>
<td>Low Lost w/power down</td>
</tr>
<tr>
<td>FRAM</td>
<td>Ferroelectric Cell</td>
<td>Non Volatile RAM</td>
<td>Very High Read-write</td>
<td>High Magnetic</td>
</tr>
</tbody>
</table>
Based on Floating-gate MOSFET (FGMOS)
- Reads like a ROM cell
- Charge trapping/removal required for modification
- Finite number of erase/re-program cycles

Content Modification
- Erased by pages
- Reprogram cycle

Fig. 7.30: Floating Gate MOSFET (FGMOS) (a) Basic structure; (b) Erasing process; (c) Programming; (d) Symbol
FLASH MEMORY TOPOLOGIES

- NAND Flash
  - Blocks erasable
  - Fast writes
  - Slower reads
  - Data storage

- NOR Flash
  - Single word erasable
  - Fast reads
  - Slower writes
  - Code storage
  - Endures larger number of erase/write cycles

- FLASH Management
  - Both require worn-out prevention strategies
  - Flash Controller

Fig. 7.31: Basic flash architectures: a) NAND Flash, and b) NOR Flash
**Internal MCU Flash**
- Provides for data and program storage
- In-system erasable & programmable
  - No external hardware necessary
- Password protected

**Features**
- Integrated Flash controller
- Internal programming voltage generator
- Bit, byte, word, and double-word (x5xx) addressable
- Segment, bank, or mass erasable

---

Fig. 7.32a: Flash memory block diagram
(Courtesy of Texas Instruments, Inc.)
FLASH MEMORY ORGANIZATION

- Information Section
  - Intended for non-volatile data
    - Can contain data or code
  - Composed of multiple segments
    - Segment number and size varies with device & MCU family

- Main Section
  - Intended for user programs
    - Can contain code or data
  - Device dependent partitioning
    - Two or more segments
    - May have banks and/or blocks depending on device & family

- Bootstrap Loader
  - Present only in x5xx/x6xx devices
  - Contains 4 segments (A ... D)
  - Intended for booting code
    - Can contain code or data
a) 32KB organization in MSP430x2xx

b) 256KB organization in MSP430x5xx

Fig. 7.32b: Sample Flash organization schemes in MSP430 devices
(Courtesy of Texas Instruments, Inc.)
**Four Control Registers**
- All are 16-bit wide
- Device dependent

**All are Password Protected**
- Key = 0A5h
- Loaded as high byte control word

**Flash Memory Security Violation**
- Caused by a Flash access w/o key
- Causes a PUC (system reset)

---

**Flash Control Registers**

<table>
<thead>
<tr>
<th>FCCTL1</th>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKWRT</td>
<td>WRT</td>
<td>Reserved*</td>
<td>EEIEX**</td>
<td>EEI**</td>
<td>MERAS</td>
<td>ERASE</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

* : SWR in Series ‘5xx/’6xx
** : Reserved in series ‘1xx/’5xx/’6xx

<table>
<thead>
<tr>
<th>FCCTL2</th>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSSELx</td>
<td>FNx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FCCTL3</th>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAIL**</td>
<td>LOCKA*</td>
<td>EMEX**</td>
<td>LOCK</td>
<td>WAIT</td>
<td>ACCVIFG</td>
<td>KEYV</td>
<td>BUSY</td>
<td></td>
</tr>
</tbody>
</table>

* : SWR in Series ‘1xx’
** : Reserved in series ‘1xx/’5xx/’6xx

<table>
<thead>
<tr>
<th>FCCTL4</th>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKINFO*</td>
<td>MRG1</td>
<td>MRG0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VPE*</td>
</tr>
</tbody>
</table>

* : Series ‘5xx/’6xx only

Fig. 7.32 MSP430 low bytes of flash memory control registers
## Introduction to Embedded Systems: Using Microcontrollers and the MSP430

### Flash Programming

#### Table 7.8 Flash control registers in MSP430 models

<table>
<thead>
<tr>
<th>Series</th>
<th>FCCTL1</th>
<th>FCCTL2</th>
<th>FCCTL3</th>
<th>FCCTL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1xx</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>x2xx</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes1</td>
</tr>
<tr>
<td>x4x</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes1, 2</td>
</tr>
<tr>
<td>x5/xx</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>x6/xx</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1: Not available in all devices. Must consult the specific device data sheet. 2: The higher byte is 00h on reset for this series.

#### Table 7.9 Flash memory writing mode

<table>
<thead>
<tr>
<th>BLKWRT</th>
<th>WRT</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No write</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>byte/word write</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>long word (32 bits) write¹</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Mass erase (</td>
</tr>
</tbody>
</table>

#### Table 7.10 Flash memory erasing mode¹

<table>
<thead>
<tr>
<th>MERAS</th>
<th>ERASE</th>
<th>'x1xx, 'x2xx, and 'x4xx</th>
<th>'x5xx/x6xx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No erase</td>
<td>No erase</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Individual segment</td>
<td>Individual Segment</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Main memory segments</td>
<td>One bank erase</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Erase all flash memory²</td>
<td>Mass erase all banks</td>
</tr>
</tbody>
</table>

1: Series ’5xx/’6xx only

1: MSP430FG461x have another bit, GMERAS, for more options. 2: Information Segment A is not erased when locked with LOCKA = 1.
FLASH MEMORY OPERATION

- Read Mode: Default Upon Reset
  - Contents modification requires Erase-Write sequence

- Erasing the Flash
  - Minimum erase is one segment
    - Segments, Banks, and the whole Flash (mass) can be erased
  - Can be done from several places
    - From another flash segment, from RAM, or from BSL

- Writing the Flash
  - Can write bytes, words, long words (x5xx), segments, or blocks
  - Write access achieved by several means
    - From another flash segment, from RAM, or from BSL

- Programming the Flash
  - Programs can be downloaded via JTAG, BSL, or Custom Solution

- Operation Sequence
  - Specific details change with device capacities and family
  - Refer to specific device’s User’s Manual & Data Sheet for details
BUS ARBITRATION & DMA TRANSFERS

- Fundamental Concepts in Bus Arbitration
- Direct Memory Access Controllers
- MSP430 DMA Support
**Bus Master**

- Device controlling Address, Data, & Control buses
- CPU is default bus master

**Bus Arbitration Process**

- Allows bus master capable devices request and gain control of system buses
- Sample bus master capable devices (other than CPU)
  - DMA controllers
  - GPUs
  - Math Co-processors
  - Other CPUs (multiprocessors)

**Fig. 3.1** General architecture of a microcomputer system
**SIMPLE BUS ARBITRATION PROTOCOL**

### Basic Steps

1. Request
2. Grant
3. Release

![Simple scenario for a bus arbitration transaction](image)

**Fig. 7.33**: Simple scenario for a bus arbitration transaction

---

**Fig. 7.34** Timing of a bus request and granting process

- **BRQ**: CPU Busy
- **BGT**: CPU in control
- **PBM in control**
**When is it Needed?**
- When multiple potential masters place bus requests

**How is it Handled?**
- Assigning priorities to bus requests
  - Similar to Interrupt Priority Handling

**Priority Management Schemes**
- Serial Arbitration
  - Daisy chain schemes
- Central Arbitration
  - Dedicated arbiter
  - Configurable controller
DMA: A Bus Master Capable Peripheral to Accelerate

- I/O-to-memory transfers
- Memory-to-memory transfers

Why a DMA Controller?

- To overcome the CPU transfer limitations

Fig. 7.35: Sequence of events in a conventional I/O to memory data transfer
Connected as a Bus Master Capable Peripheral
- Accessed as a regular peripheral to be configured
- Replaces the CPU as bus master
  - After a successful bus arbitration transaction

Enables Direct Transfers to/from Memory without CPU Intervention

Fig. 7.36: Connecting a DMA controller to an MPU system and I/O device
DMA CONTROLLER STRUCTURE (1/2)

Fig. 7.37: Minimal structure of a one-channel DMA controller
- **Status and Control Registers**
  - Like any other peripheral

- **An Address Register**
  - Specifies the initial address of a transfer
  - A second address register if supporting memory-to-memory transfers

- **A Word Count Register**
  - Specifies how many words will be transferred

- **A Data Register**
  - Temporarily holds data in transit
### DMA Transfer Modes

- **Based on the Way the DMA Uses the System Buses**
  - Data block transfer assumed

- **Burst Mode Transfers**
  - All words in a block transferred with a single bus arbitration transaction
  - Used with fast peripherals

- **Cycle Stealing Transfers**
  - A bus arbitration transaction mediates per word
  - Used for slower peripherals

- **Transparent Transfers**
  - DMA detects CPU inactivity to perform transfers
  - Requires additional hardware
DMA TRANSFER TYPES

- Based on the Number of Bus Cycles to Complete a Transfer
  - Involves different data paths
- Two-cycle DMA Transfer
  - Two bus cycles needed to complete the transfer
  - Data passes through DMA data register
- One-cycle DMA Transfer
  - Whole transfer completed in a single bus cycle
  - Direct I/O – Memory path
1. DMA request
2. Bus arbitration
3. First cycle: store datum in DMA
4. Second cycle: datum to destination
5. Address & count update
6. Redo from step 3 or exit
7. Exit: release buses

By far the most common type of DMA transfers

Fig. 7.38: Sequence of events in a two-cycle DMA transfer
ONE-CYCLE DMA TRANSFER

1. DMA request
2. Bus arbitration
3. Simultaneous address & DMAACK strobe. Data transferred
4. Address & count update
5. Redo from step 3 or exit
6. Exit: release buses

Fig. 7.39: Sequence of events in a one-cycle DMA transfer

Requires support from DMAACK signal and simultaneous memory & I/O strobes

© 2014 by M. Jiménez, R. Palomera, & I. Couvertier

Chapter 7: 78
DMA PROGRAMMING HINTS

- **Designer Establishes DMA Usage**
  - Establishes which peripherals
  - Decides how the DMA will be used

- **User Program Configures DMA**
  - Provide starting address and block size
  - Decides transfer type and mode

- **Application Dictates the Rules**
  - Based on speed transfer needs and peripheral type
MSP430 DMA SUPPORT

- Provided by many MSP430 Models

- Features
  - Several independent transfer channels
  - Configurable DMA channel priorities
  - Supports only two-cycle transfers
    - byte-to-byte
    - word-to-word, and
    - mixed byte/word transfers
  - Maximum block sizes up to 65,535 bytes or words
  - Configurable transfer trigger selections
  - Selectable edge or level-triggered DMA transfer requests
  - Support for single, block, or burst-block transfer modes
Characteristics and Number of Channels are Device Dependent
- Refer to User’s Manual for specific details

Single Multi-sourced Interrupt Vector
- Encoded for jump table

Enabled for Low-power Modes support

Fig. 7.40: MSP430x5xx DMA controller block diagram
Four Addressing Modes
- Address to address
- Block to address

Six Transfer Modes

<table>
<thead>
<tr>
<th>DMADTx</th>
<th>Transfer Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Single transfer</td>
<td>Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.</td>
</tr>
<tr>
<td>001</td>
<td>Block transfer</td>
<td>A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.</td>
</tr>
<tr>
<td>010, 011</td>
<td>Burst-block transfer</td>
<td>CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.</td>
</tr>
<tr>
<td>100</td>
<td>Repeated single transfer</td>
<td>Each transfer requires a trigger. DMAEN remains enabled.</td>
</tr>
<tr>
<td>101</td>
<td>Repeated block transfer</td>
<td>A complete block is transferred with one trigger. DMAEN remains enabled.</td>
</tr>
<tr>
<td>110, 111</td>
<td>Repeated burst-block transfer</td>
<td>CPU activity is interleaved with a block transfer. DMAEN remains enabled.</td>
</tr>
</tbody>
</table>
Trigger Conditions

- Several internal peripheral modules can trigger and initiate a DMA transfer
- Source list is device dependent. Check device’s User’s manual & data sheet for details

<table>
<thead>
<tr>
<th>Module</th>
<th>Trigger Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA</td>
<td>When DMAREQ bit is set.</td>
</tr>
<tr>
<td>Timer_A</td>
<td>When TAxCCRO CCIFG flag is set.</td>
</tr>
<tr>
<td>Timer_B</td>
<td>When TBxCCRO CCIFG flag is set.</td>
</tr>
<tr>
<td>USCI_Ax</td>
<td>When USCI_Ax receives new data.</td>
</tr>
<tr>
<td>USCI_Bx</td>
<td>When USCI_Bx receives new data.</td>
</tr>
<tr>
<td>DAC12_A</td>
<td>When DAC12_xCTL0 DAC12IFG flag is set.</td>
</tr>
<tr>
<td>ADC12_A</td>
<td>When ADC12IFG flag is set.</td>
</tr>
<tr>
<td>MPY</td>
<td>When the hardware multiplier is ready for a new operand.</td>
</tr>
<tr>
<td>Reserved</td>
<td>No transfer is triggered.</td>
</tr>
</tbody>
</table>
END OF CHAPTER 7 SLIDES