
# Introduction to Timer_A and LCD Peripherals

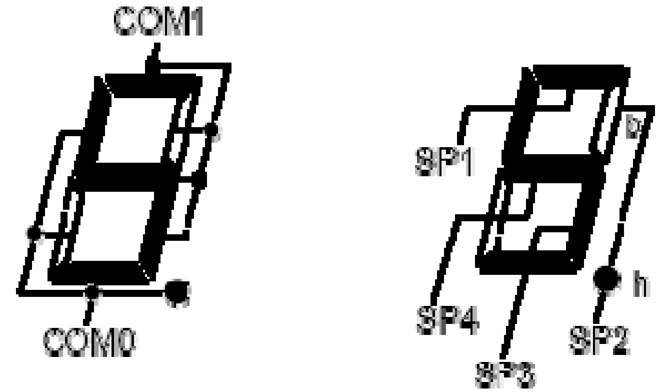
Peter Forstner
MSP430 FAE Europe
Texas Instruments

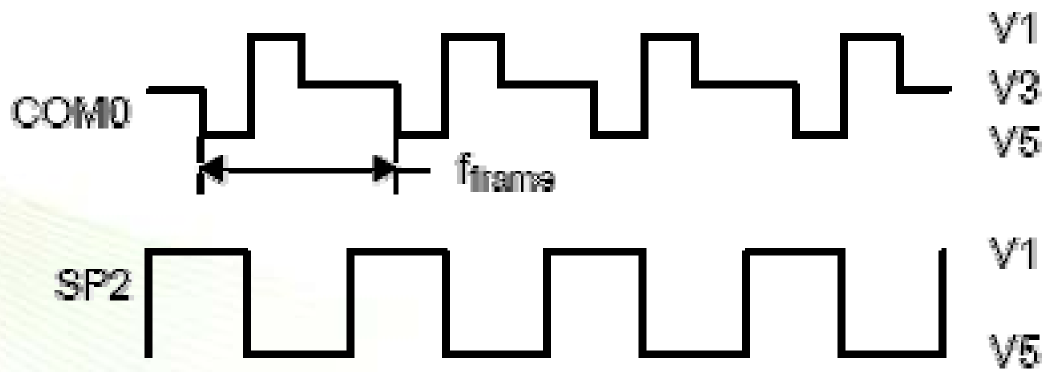
# Agenda

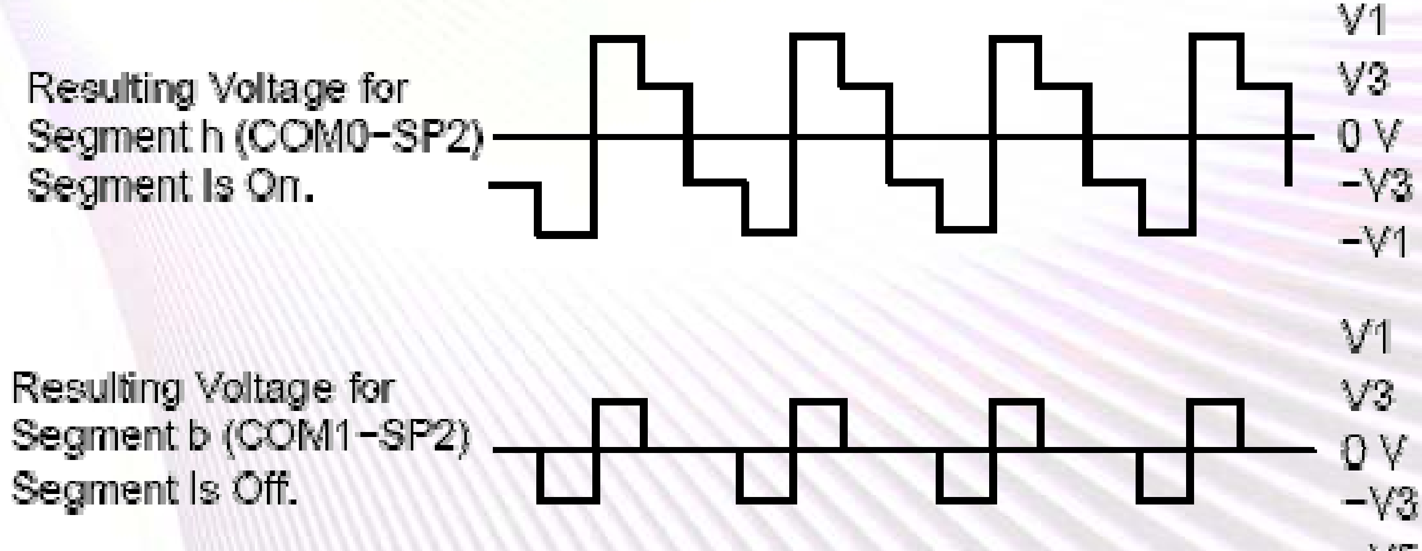
- LCD Modules
- Timer_A
- Timer_A Lab

# LCD Drive Basics

- AC signals
- Automatic signal generation
- RMS voltage determines if segment is on/off
- < 50mV DC voltage allowed



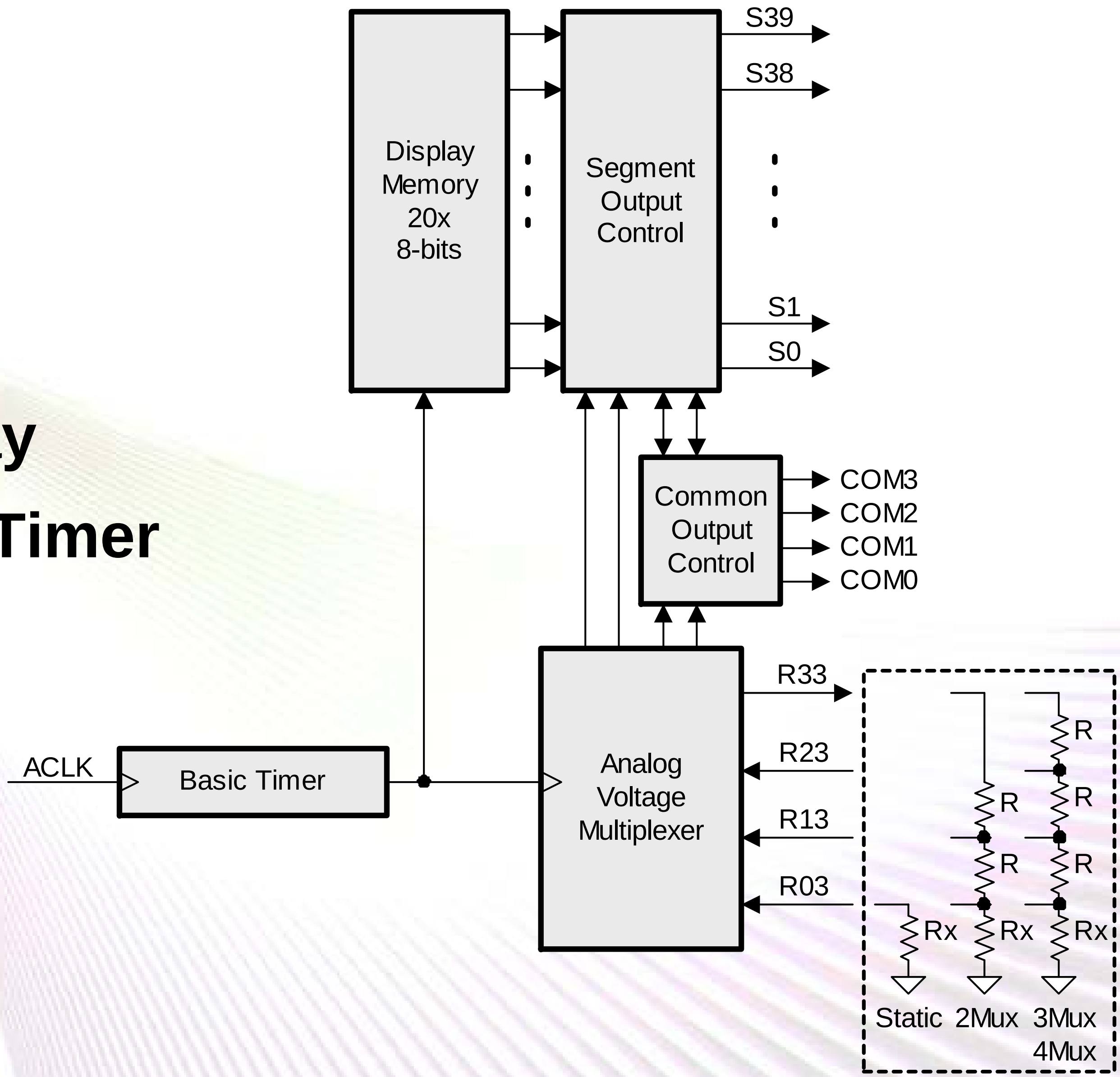
# LCDx Memory Map

- **Each bit corresponds to one LCD segment**
- **All bits may not be used depending on muxing**
- **‘1’ = segment on**
  **‘0’ = segment off**

| Associated Common Pins | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Address | 7 | | | | | | | 0 | n | Associated Segment Pins |
| 0A4h | -- | -- | -- | -- | -- | -- | -- | -- | 38 | 39, 38 |
| 0A3h | -- | -- | -- | -- | -- | -- | -- | -- | 36 | 37, 36 |
| 0A2h | -- | -- | -- | -- | -- | -- | -- | -- | 34 | 35, 34 |
| 0A1h | -- | -- | -- | -- | -- | -- | -- | -- | 32 | 33, 32 |
| 0A0h | -- | -- | -- | -- | -- | -- | -- | -- | 30 | 31, 30 |
| 09Fh | -- | -- | -- | -- | -- | -- | -- | -- | 28 | 29, 28 |
| 09Eh | -- | -- | -- | -- | -- | -- | -- | -- | 26 | 27, 26 |
| 09Dh | -- | -- | -- | -- | -- | -- | -- | -- | 24 | 25, 24 |
| 09Ch | -- | -- | -- | -- | -- | -- | -- | -- | 22 | 23, 22 |
| 09Bh | -- | -- | -- | -- | -- | -- | -- | -- | 20 | 21, 20 |
| 09Ah | -- | -- | -- | -- | -- | -- | -- | -- | 18 | 19, 18 |
| 099h | -- | -- | -- | -- | -- | -- | -- | -- | 16 | 17, 16 |
| 098h | -- | -- | -- | -- | -- | -- | -- | -- | 14 | 15, 14 |
| 097h | -- | -- | -- | -- | -- | -- | -- | -- | 12 | 13, 12 |
| 096h | -- | -- | -- | -- | -- | -- | -- | -- | 10 | 11, 10 |
| 095h | -- | -- | -- | -- | -- | -- | -- | -- | 8 | 9, 8 |
| 094h | -- | -- | -- | -- | -- | -- | -- | -- | 6 | 7, 6 |
| 093h | -- | -- | -- | -- | -- | -- | -- | -- | 4 | 5, 4 |
| 092h | -- | -- | -- | -- | -- | -- | -- | -- | 2 | 3, 2 |
| 091h | -- | -- | -- | -- | -- | -- | -- | -- | 0 | 1, 0 |
| | Sn+1 | | | | Sn | | | | | |

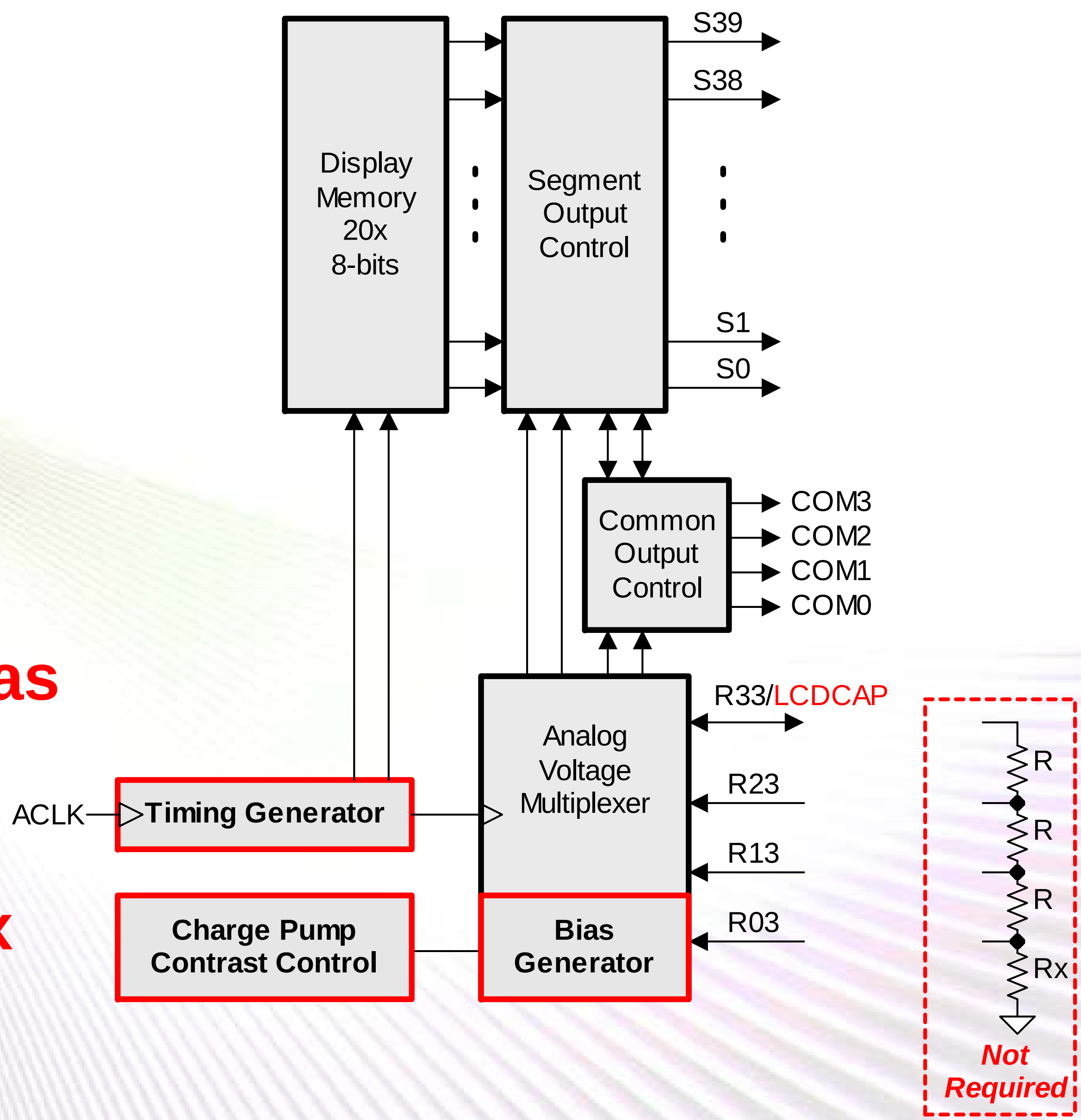
# LCD Module

- Ultra-low power
- Fully automatic
- 4/3/2/1 mux
- Up to 160-bit display
- Timing from Basic Timer

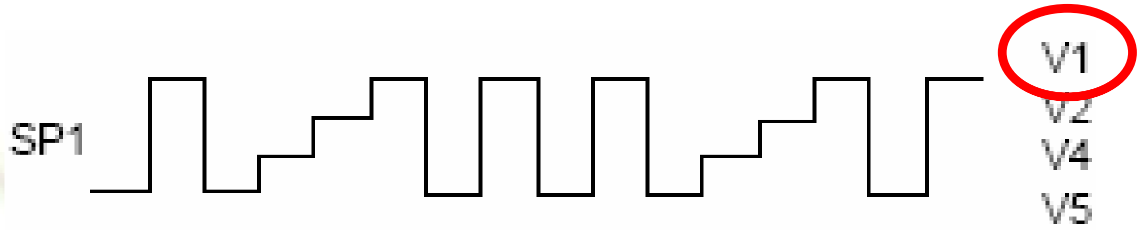


# LCD_A Module

- Ultra-low power
- Fully automatic
- 4/3/2/1 mux
- Up to 160-bit display
- Internal regulated voltage generator
- Internal or external bias generation
- Contrast control
- 1/2 bias for 3 or 4 mux
- Internal clock generation
- ’42x0 and ‘461x devices



# LCDx Voltage Options



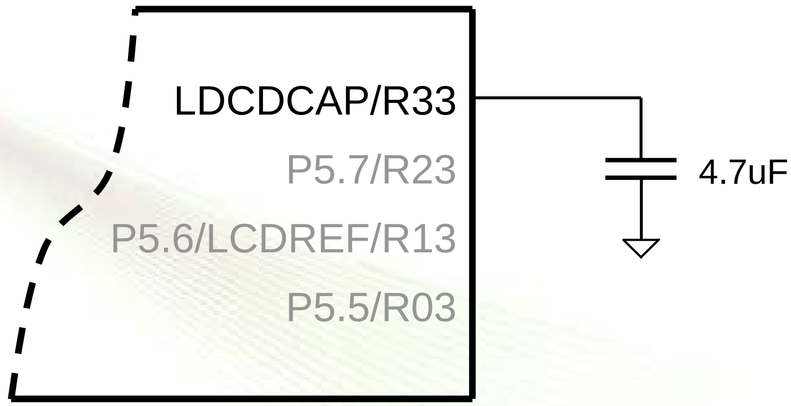

## LCD Module:

- **LCD sources $V_{LCD}$ from AVcc only**

## LCD_A Module:

- **LCD_A can source $V_{LCD}$ from AVcc, internal charge pump or externally (must be ≤ AVcc)**
- **Charge pump output independent of Vcc**
- **Charge pump provides ~2.6V to ~3.44V**

# LCD_A Charge Pump Capacitor




- **When LCD_A charge pump is on:**
  - External 4.7uF capacitor or greater *MUST* be on LCDCAP pin *Otherwise device damage can occur*
  - S0-S3 disabled

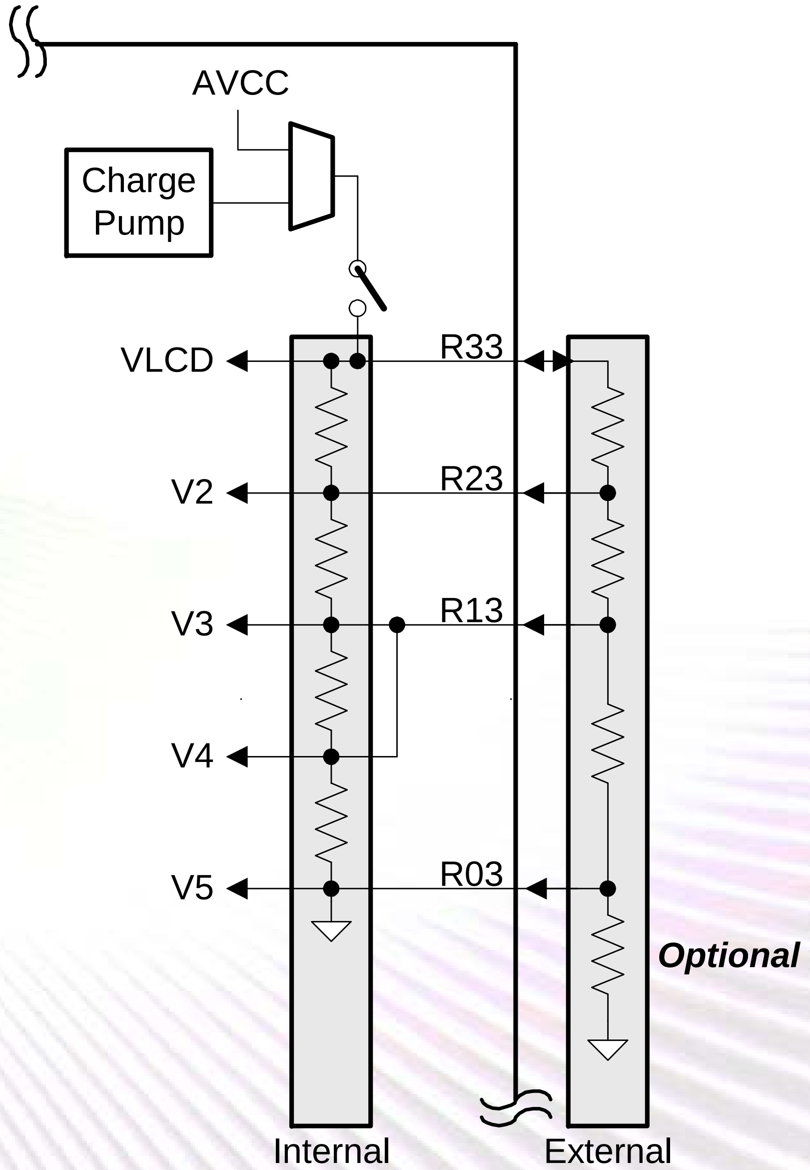
# LCDx Biasing Options

## LCD Module:

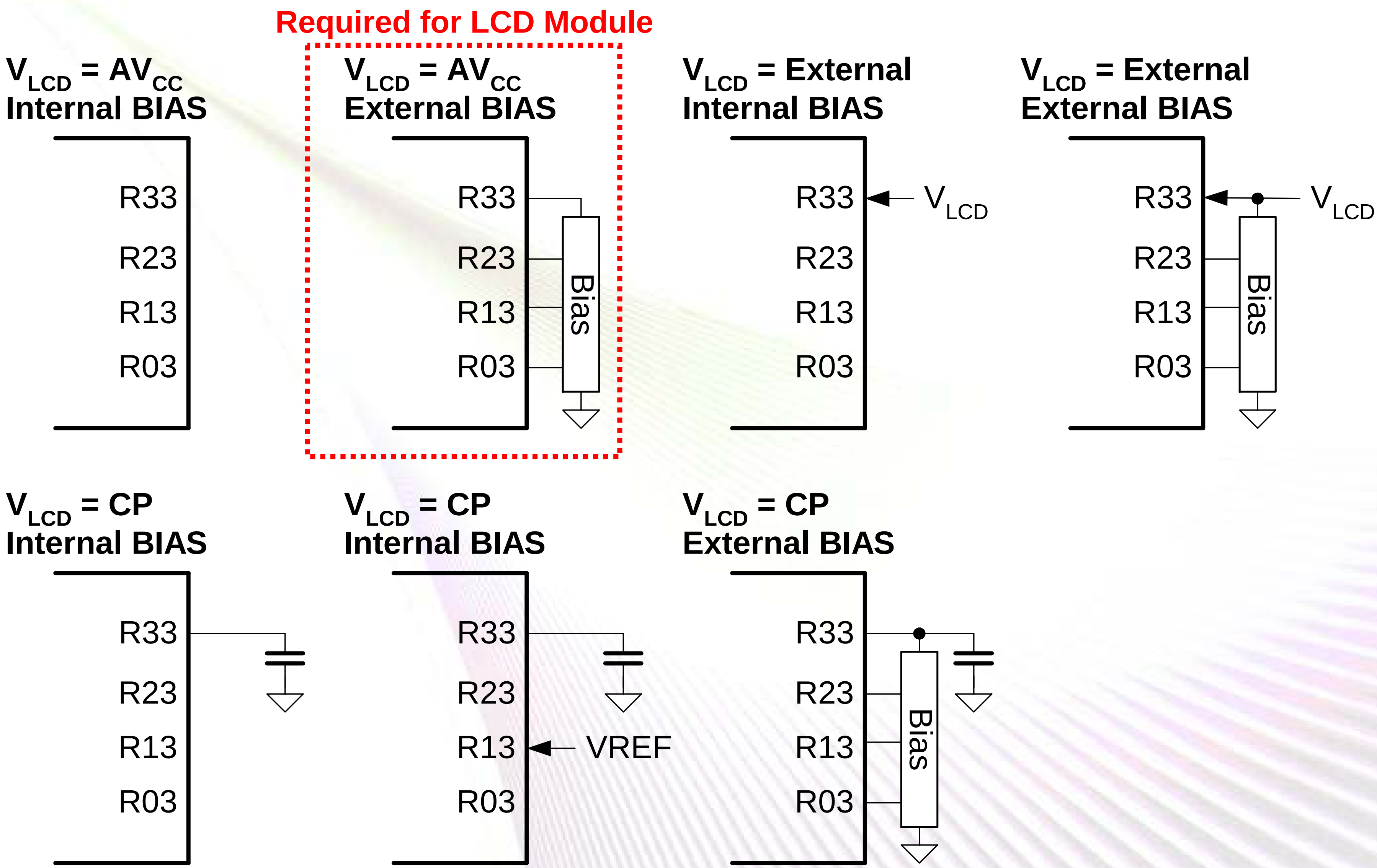
- External bias resistors required

## LCD_A Module:

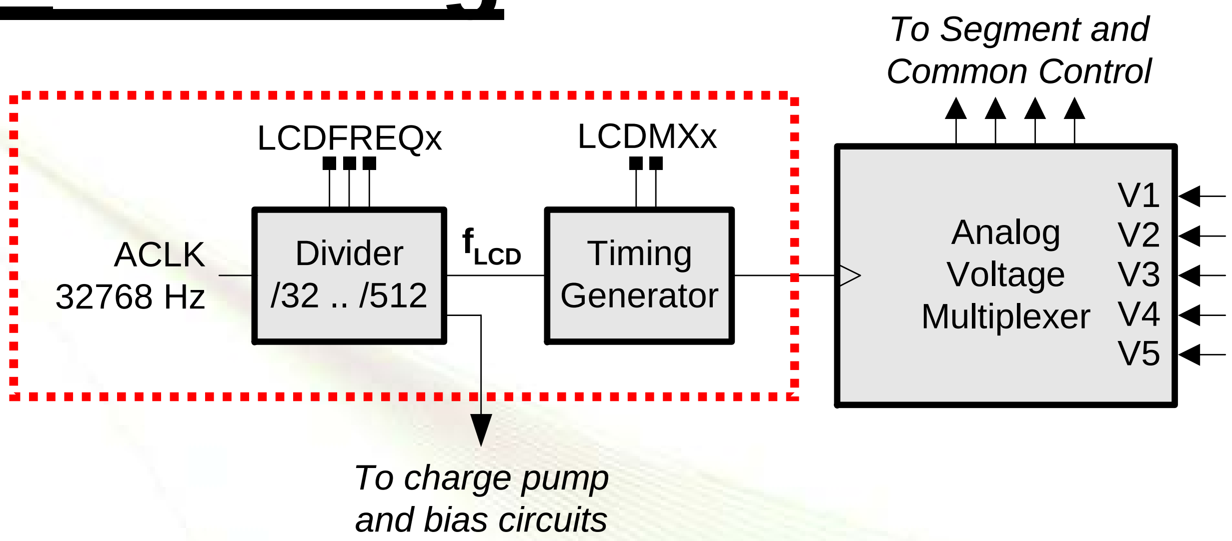
- Internal or external bias voltage generation
- Independent of $V_{LCD}$ source
- Internal = simpler
- External = lower power



# LCD_A Voltage And Biasing Options



# LCD_A Timing




$$f_{LCD} = 2 \times mux \times f_{FRAME}$$

- **Internal timing generation: Basic_Timer not required**
- **Divider assumes 32768 Hz**
- **$f_{LCD}$ based on $f_{FRAME}$ of LCD**
- **Lowest frequency = lowest current**
- **Highest frequency = lowest flicker**

# LCD_A Contrast Control

LCDAVCTL1, LCD_A Voltage Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Unused | | | VLCDx | | | | Unused |

| | | |
|---|---|---|
| VLCDx | Bits 4-1 | 0000 Charge pump disabled. |
| | | 0001 $V_{LCD}$ = 2.60V |
| | | 0010 $V_{LCD}$ = 2.66V |
| | | 0011 $V_{LCD}$ = 2.72V |
| | | 0100 $V_{LCD}$ = 2.78V |
| | | 0101 $V_{LCD}$ = 2.84V |
| | | 0110 $V_{LCD}$ = 2.90V |
| | | 0111 $V_{LCD}$ = 2.96V |
| | | 1000 $V_{LCD}$ = 3.02V |
| | | 1001 $V_{LCD}$ = 3.08V |
| | | 1010 $V_{LCD}$ = 3.14V |
| | | 1011 $V_{LCD}$ = 3.20V |
| | | 1100 $V_{LCD}$ = 3.26V |
| | | 1101 $V_{LCD}$ = 3.32V |
| | | 1110 $V_{LCD}$ = 3.38V |
| | | 1111 $V_{LCD}$ = 3.44V |

- **Charge pump gives software-selectable $V_{LCD}$**
- **Changing $V_{LCD}$ automatically adjusts other LCD voltages regardless of location of bias resistors**
- **LCD_A maintains original external contrast control option as LCD module**

# LCD_A Current Consumption

| | |
|---|---|
| $V_{LCD}$ = $AV_{CC}$, 1/3 internal bias | ~4.5 µA |
| $V_{LCD}$ = $AV_{CC}$, 1/3 external bias | ~2.3 µA |
| $V_{LCD}$ = CP, 1/3 internal bias | ~7.5 µA |
| $V_{LCD}$ = CP, 1/3 external bias | ~5 µA |

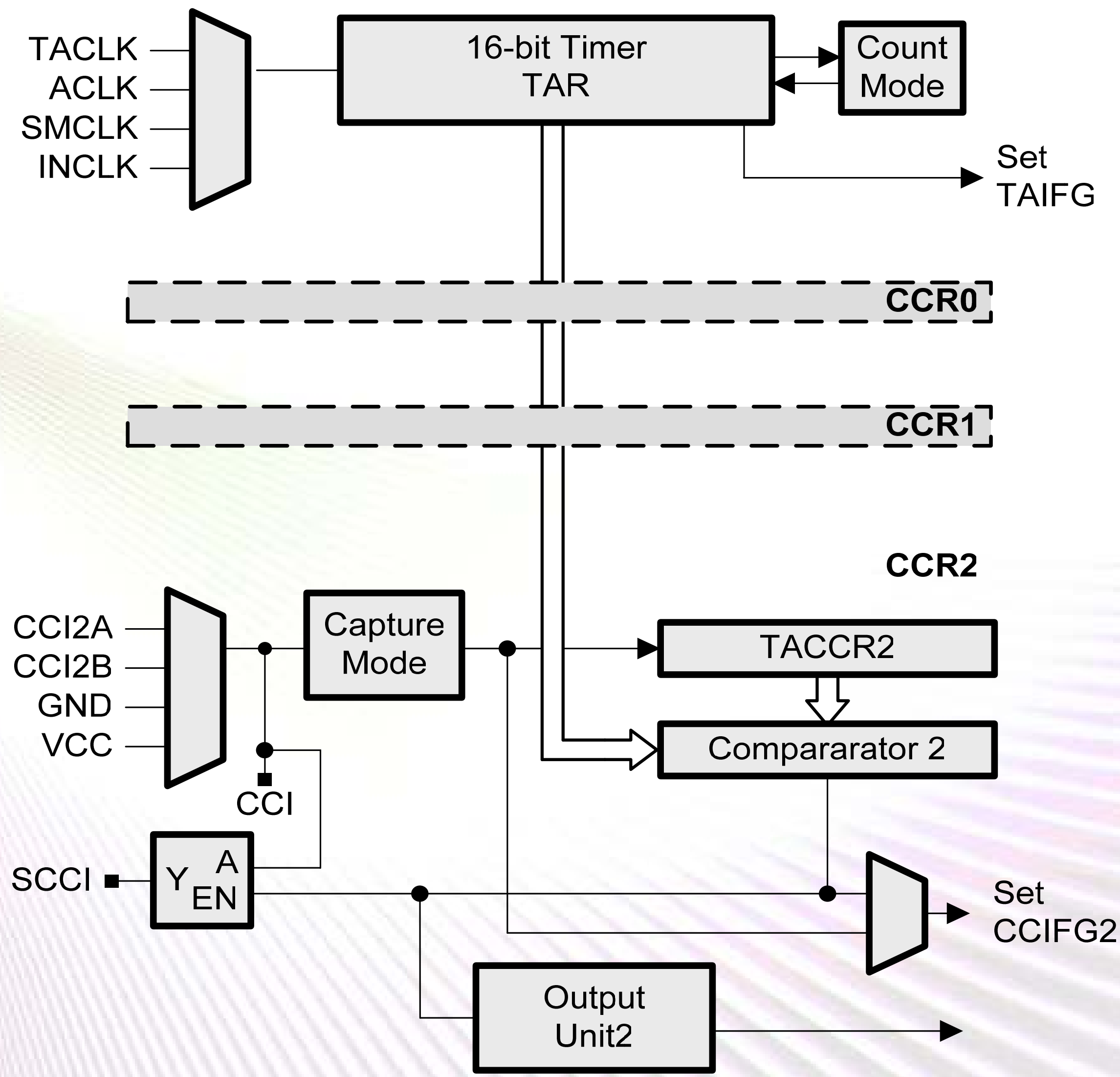
- Vcc with external bias is lowest current option
- Using internal bias adds 1 to 2 µA
- Enabling CP adds 2 to 4 µA

# Agenda

- LCD Modules
- Timer_A
- Timer_A Lab

# Overview of Timer_A

- **Asynchronous 16-Bit timer/counter**
- **Continuous up-down up**
- **Asynchronous input latch**
- **Interrupt vector register for fast decoding**
- **DMA enabled**
- **On ALL MSP430s!**



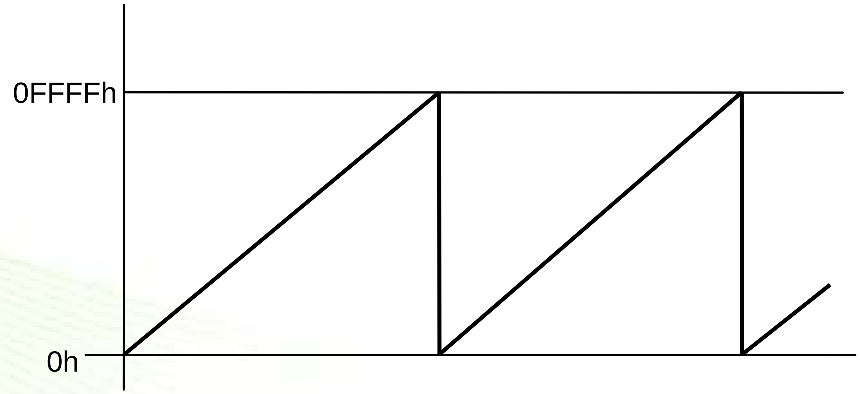
# Timer_A Counting Modes

**Stop/Halt**

Timer is halted

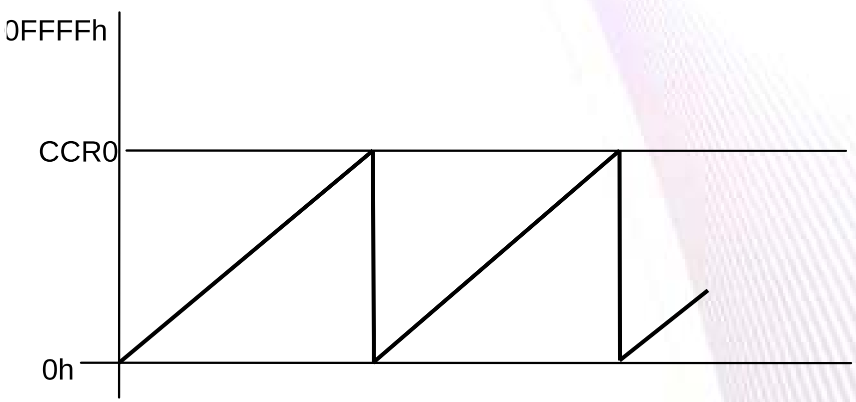
**Continuous**

Timer continuously counts up



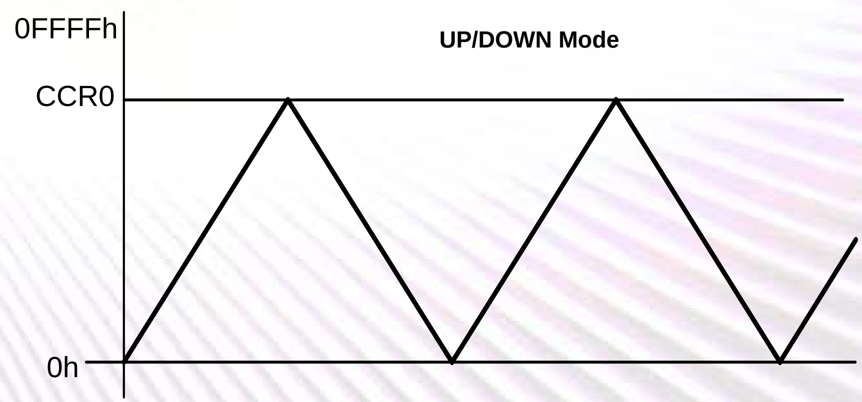

**Up**

Timer counts between 0 and CCR0

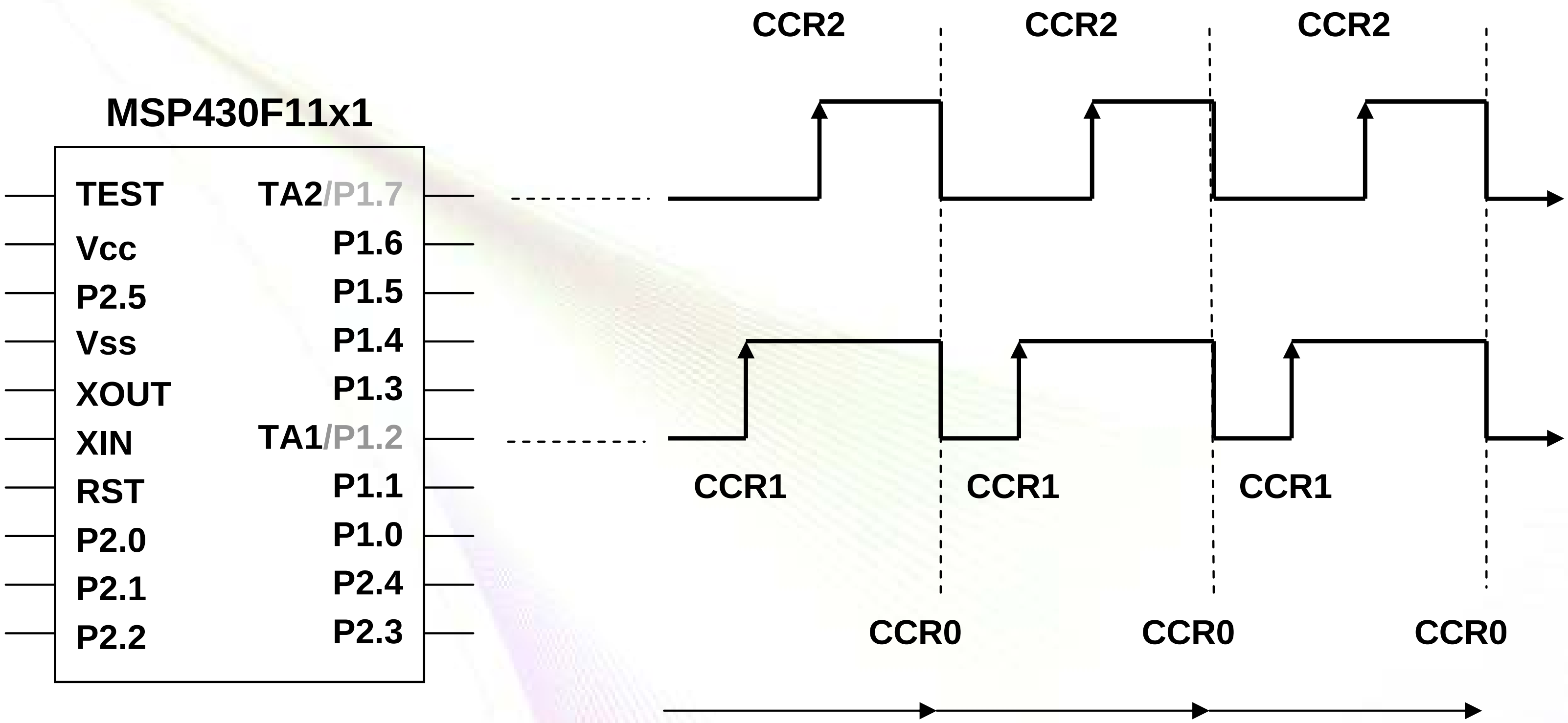



**Up/Down**

Timer counts between 0 and CCR0 and 0

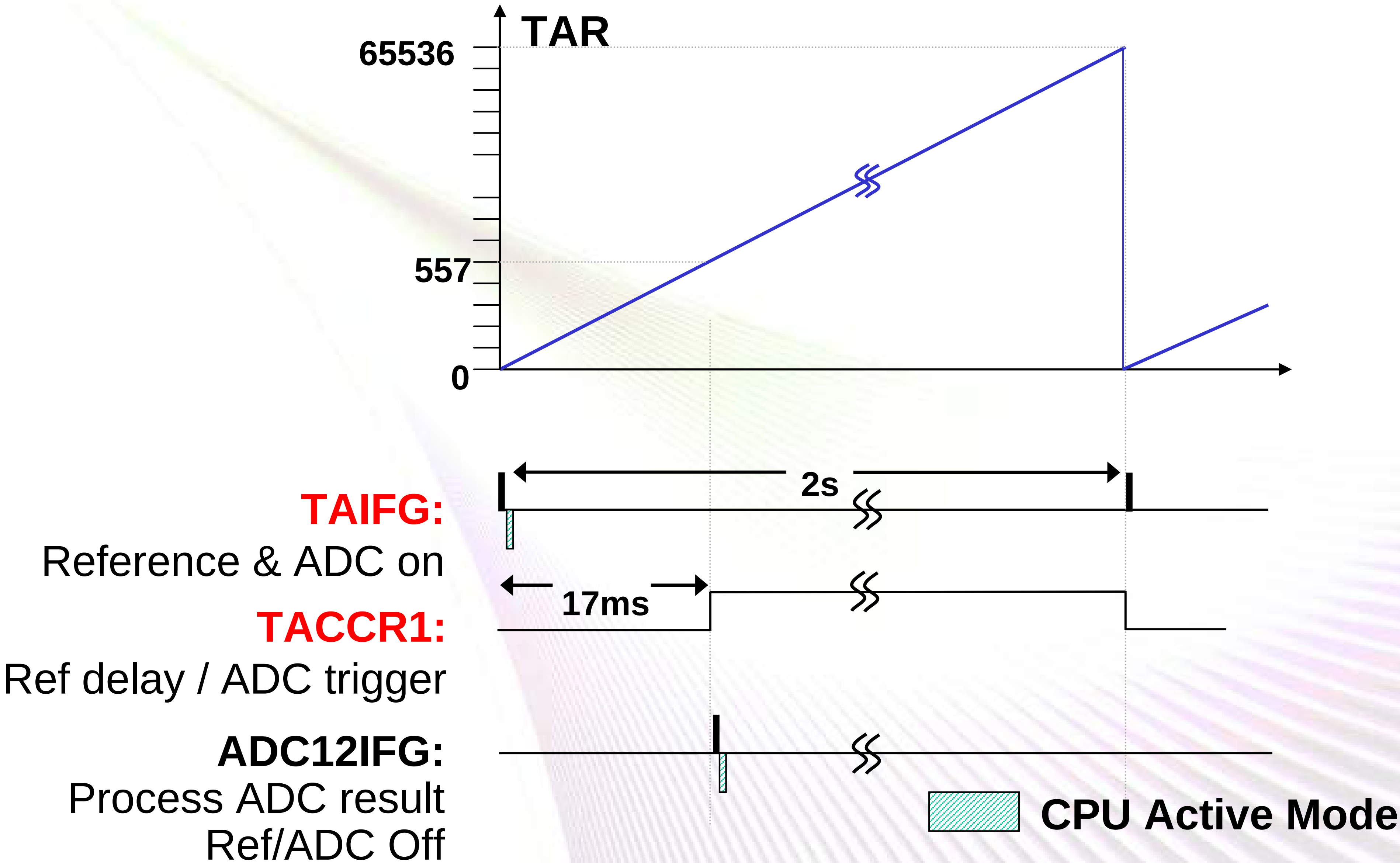


# Timer_A PWM Example

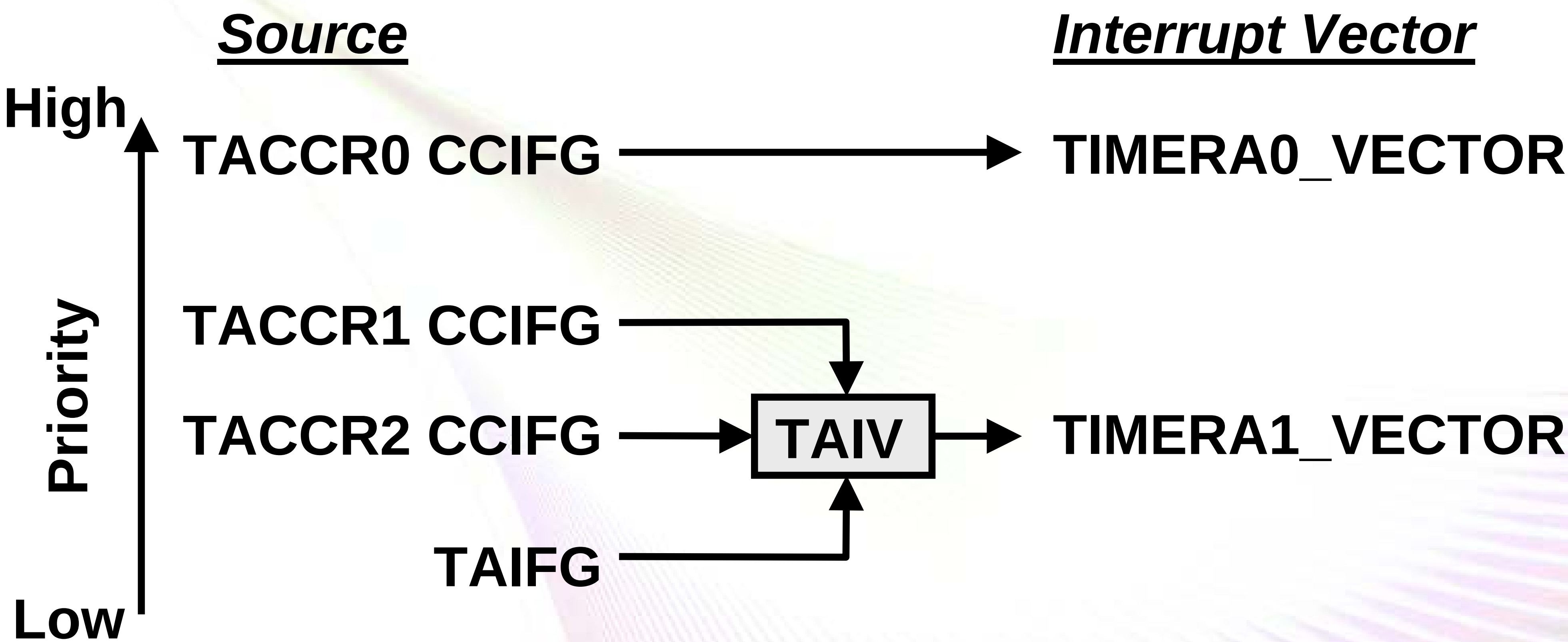



**Completely automatic**

# Direct H/W Control With Timer_A



# Interrupt Sources and Flags




- **Use interrupt vector generator TAIV**
  ***for efficient interrupt processing***

# Interrupt Vector Generator TAIV

| 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | 0 |

| Source | TAIV Contents |
|---|---|
| No interrupt pending | 0 |
| TACCR1 CCIFG | 02h |
| TACCR2 CCIFG | 04h |
| Reserved | 06h |
| Reserved | 08h |
| TAIFG | 0Ah |
| Reserved | 0Ch |
| Reserved | 0Eh |

- Shared interrupt vector for CC1/2 IFGs / overflow
- Fast decoding reduces code size / CPU load
- In assembly, add TAIV to PC!

# TAIV Example

```
#pragma vector = TIMERA1_VECTOR
__interrupt void TIMERA1_ISR(void)
{
  switch(__even_in_range(TAIV,10))
  {
    case  2 :       // TACCR1 CCIFG
      P1OUT ^= 0x04; break;
    case  4 :       // TACCR2 CCIFG
      P1OUT ^= 0x02; break;
    case 10 :       // TAIFG
      P1OUT ^= 0x01; break;
   }
}
```

```
0xF814  add.w   &TAIV,PC
0xF818  reti
0xF81A  jmp     0xF824
0xF81C  jmp     0xF82A
0xF81E  reti
0xF820  reti
0xF822  jmp     0xF830
0xF824  xor.b   #0x4,&P1OUT
0xF828  reti
0xF82A  xor.b   #0x2,&P1OUT
0xF82E  reti
0xF830  xor.b   #0x1,&P1OUT
0xF834  reti
```

- Example shows IAR C code vs. disassembly
- Low software-overhead assembly code can be seen
- Use IAR intrinsic for vectored INT sources as well!

# 1 Timer - Different PWM Frequencies

- **With 3 CC blocks, 3 independent frequencies with 3 different duty cycles can be generated**
- **Timer_A in continuous mode**
- **CC ISR is used to update CCRx for next transition**
- **Two ISR calls per PWM period**
- **ISRs as short as possible**
- **Limits of this approach:**
  - CCR update overhead limits min and max duty cycle
  - Stopping program execution interrupts PWM generation
- ***Always*, Timer_A can also be used to generate same-frequency PWM signals entirely in hardware**

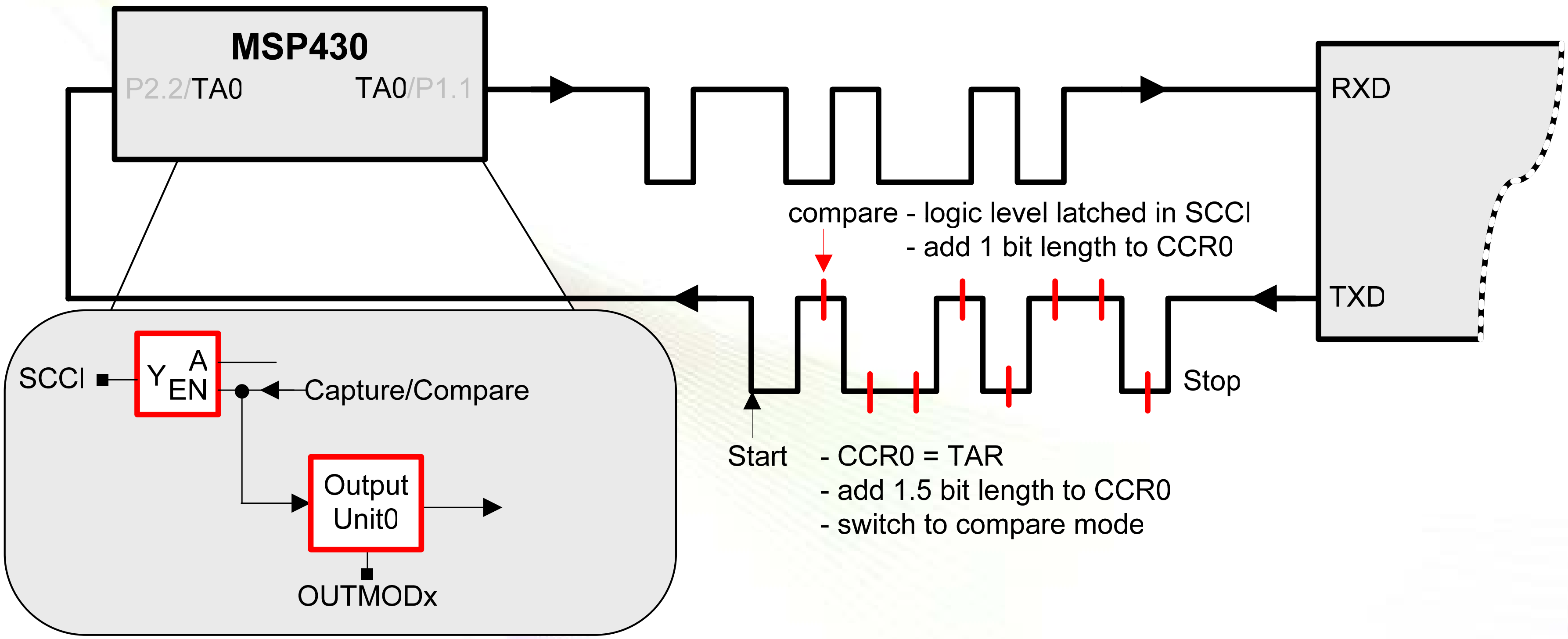
# 1 Timer - Different PWM Frequencies

```
void PWM0Upd(unsigned int Period, unsigned int DutyCycle)
{
  TACCTL0 &= ~CCIE;                // Protect against int.
  PWM0Low = Period - DutyCycle;    // Sync update of PWM low
  PWM0HighTmp = DutyCycle;         // and high periods
  TACCTL0 |= CCIE;                 // Allow interrupts
}

#pragma vector = TIMERA0_VECTOR
__interrupt void TIMERA0_ISR(void)
{
  if (TACCTL0 & OUTMOD2) {         // Was output reset?
    PWM0High = PWM0HighTmp;        // Load new duty cycle
    TACCR0 += PWM0Low; }           // Cycles until set...
  else                             // Output was set
    TACCR0 += PWM0High;            // Cycles until reset...
  TACCTL0 ^= OUTMOD2;              // Toggle set/reset
}
```

- **TIMERA1_VECTOR updates PWM1 & PWM2 outputs**

# Low-Overhead UART Implementation




- **100% hardware bit latching and output**
- **Full speed from LPM3 and LPM4**
- **Low CPU Overhead**
- **App Note SLAA078**

# Timer_A: 1000 And 1 Uses

- **Different function for each CC block can coexist:**
  - Peripheral trigger (ADC, DAC, DMA, …)
  - Interval timer
  - RTC
  - Periodic wakeup
  - Slope A/D conversion for R and V
  - Serial communication, UART
  - Hardware PWM
  - Event capture
  - Interval measurement
  - Frequency measurement
  - Software FLL (devices without FLL)

- ***Select Timer input frequency for best fit***

# Timer_B Differences

- 8,10,12 or 16-bit timer or counter
- Up to 7 CCRx units available
- Outputs double-buffered for simultaneous loading
- CCRx registers can be grouped for simultaneous updates
- SCCI latch not implemented (no UART function)
- Tri-state function from external pin

**Default Function is identical to Timer_A**

# Agenda

- LCD Modules
- Timer_A
- Timer_A Lab

# Timer_A Lab - Overview

- **Configure Timer_A for a 1-second interrupt**
  - Set applicable TACTL bits to configure clock source and clear timer
  - Set applicable bits in TACCTL0 for compare mode, interrupt enabled
  - Load TACCR0 with #of counts to give 1s interrupt
  - Start timer by setting mode bits in TACTL

- **Toggle P2.1 inside of 1-second ISR**

# Timer_A Lab – Code

```
  void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop WDT
  FLL_CTL0 |= XCAP14PF;     // Configure load caps
  P2DIR |= BIT1;            // Set P2.1 to output
  TACTL = ________________; // Clock = ACLK (32768), clear
  TACCTL0 = ____;           // CCR0 interrupt enabled
  TACCR0 = _______;         // #counts for 1s
  TACTL |= ____;            // Setting mode bits starts timer

  _BIS_SR(LPM3_bits + GIE); // Enter LPM3 w/ interrupt
}
```

# Timer_A Lab – Solution

```
  void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop WDT
  FLL_CTL0 |= XCAP14PF;     // Configure load caps
  P2DIR |= BIT1;            // Set P2.1 to output
  TACTL = TASSEL_1 + TACLR; // Clock = ACLK (32768), clear
  TACCTL0 = CCIE;           // CCR0 interrupt enabled
  TACCR0 = 32768-1;         // #counts for 1s
  TACTL |= MC_1;            // Setting mode bits starts timer

  _BIS_SR(LPM3_bits + GIE); // Enter LPM3 w/ interrupt
}
```

- **Why was TAIE not set in TACTL?**
- **Why was MCx bits not set initially when TACTL was configured?**
- **Why was TACCR0 loaded with 32768-1?**

## IMPORTANT NOTICE



Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DSP | dsp.ti.com | Broadband | www.ti.com/broadband |
| Interface | interface.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Logic | logic.ti.com | Military | www.ti.com/military |
| Power Mgmt | power.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| RFID | www.ti-rfid.com | Telephony | www.ti.com/telephony |
| Low Power Wireless | www.ti.com/lpw | Video & Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless |


Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265