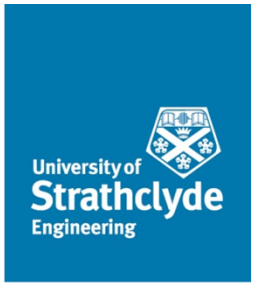


Switch Debouncing and the ADC

James Irvine

DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING

Switch Debouncing

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= BIT0;                            // P1.0 output
  P1OUT |= BIT3;                            // Select pull up resistor on P1.3
  P1REN |= BIT3;                            // and enable it

  while (1)
  {
    if (!(P1IN & BIT3))                     // Is switched pressed (==0)?
    {
      P1OUT ^= BIT0;                        // Toggle P1.0 to switch LED
      while (!(P1IN & BIT3))                // Wait for switch to be released
        /* do nothing */ ;
    }
  }
}
```

Switch Debouncing

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P1DIR |= BIT0;                            // Set P1.0 to output direction
  P1OUT |= BIT3;                            // Select pull up resistor on P1.3
  P1REN |= BIT3;                            // and enable it
  P1IES |= BIT3;                            // P1.3 high to low edge
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
  P1IE |=  BIT3;                            // P1.3 interrupt enabled

  __bis_SR_register(LPM4_bits + GIE);       // Enter LPM4 w/interrupt
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
  P1OUT ^= BIT0;                            // P1.0 = toggle
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P1DIR |= BIT0;                            // Set P1.0 to output direction
  P1OUT |= BIT3;                            // Select pull up resistor on P1.3
  P1REN |= BIT3;                            // and enable it
  P1IES |= BIT3;                            // P1.3 high to low edge
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
  P1IE |=  BIT3;                            // P1.3 interrupt enabled

  __bis_SR_register(LPM4_bits + GIE);       // Enter LPM4 w/interrupt
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
  P1OUT ^= BIT0;                            // P1.0 = toggle
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P1DIR |= BIT0;                            // Set P1.0 to output direction
  P1OUT |= BIT3;                            // Select pull up resistor on P1.3
  P1REN |= BIT3;                            // and enable it
  P1IES |= BIT3;                            // P1.3 high to low edge
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
  P1IE |=  BIT3;                            // P1.3 interrupt enabled

  __bis_SR_register(LPM4_bits + GIE);       // Enter LPM4 w/interrupt
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
  P1OUT ^= BIT0;                            // P1.0 = toggle
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P1DIR |= BIT0;                            // Set P1.0 to output direction
  P1OUT |= BIT3;                            // Select pull up resistor on P1.3
  P1REN |= BIT3;                            // and enable it
  P1IES |= BIT3;                            // P1.3 high to low edge
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
  P1IE |=  BIT3;                            // P1.3 interrupt enabled

  __bis_SR_register(LPM4_bits + GIE);       // Enter LPM4 w/interrupt
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
  P1OUT ^= BIT0;                            // P1.0 = toggle
  P1IFG &= ~BIT3;                           // P1.3 IFG cleared
}
```

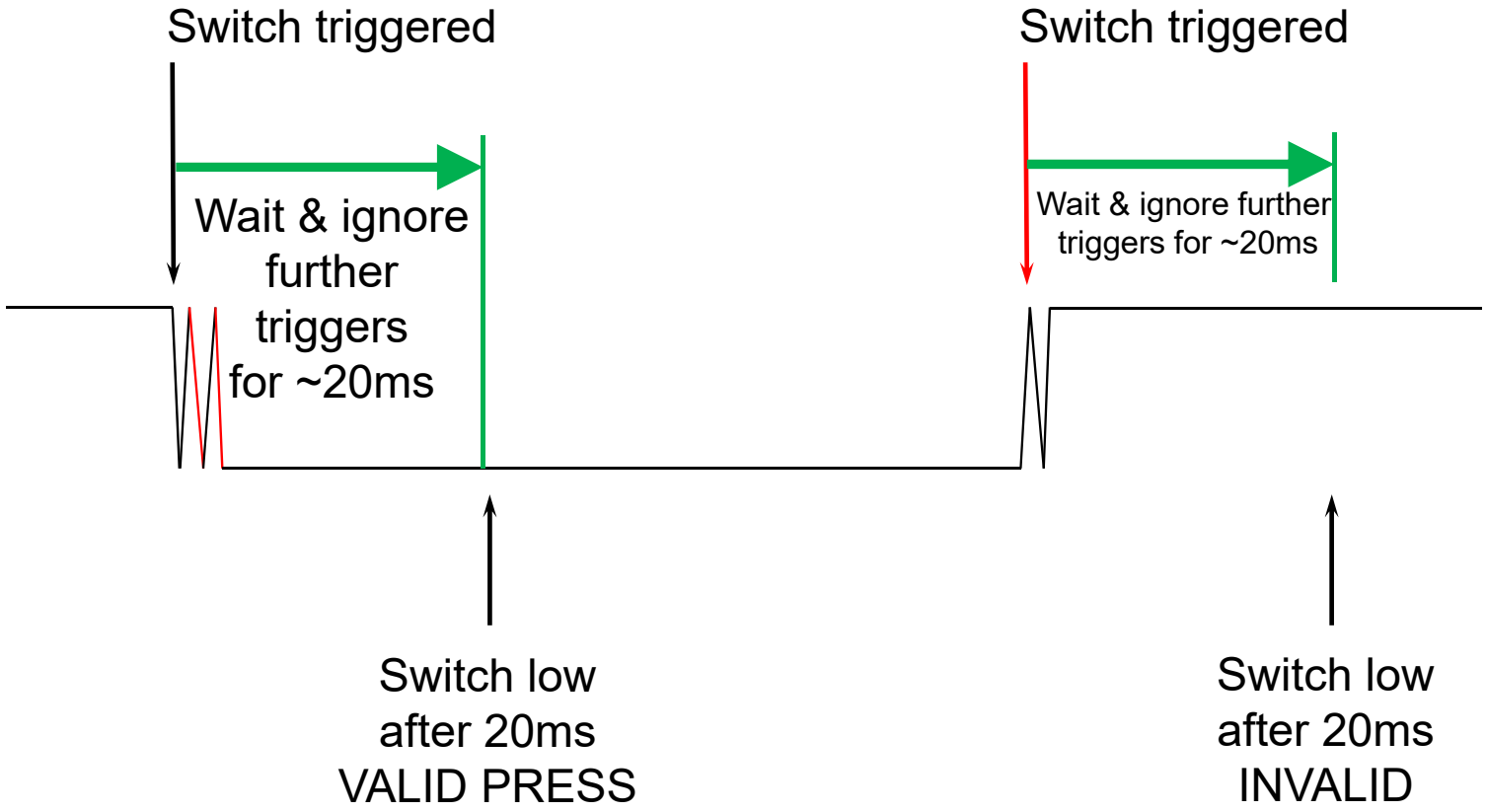
Switch Debouncing

Switch Debouncing

False count on release

Debouncing alone doesn't solve problem

Check **After** Debouncing



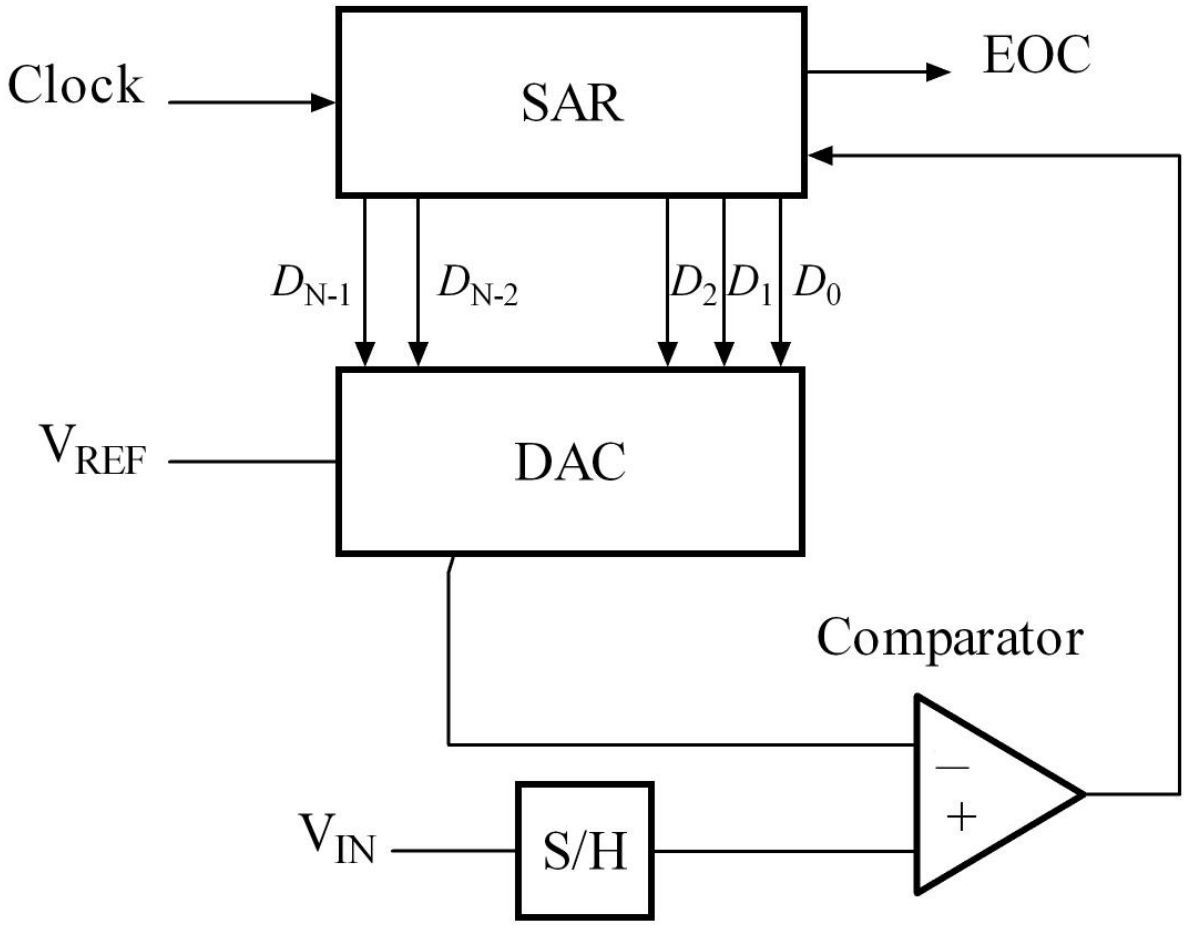
Switch Debouncing

University of Strathclyde Engineering

- Trigger starts debounce period
- Switch press only registered if valid at end of debounce
  - Delay of debounce period, but only a few ms
- Can have a wait loop, but better to have a timer
  - Using timer and interrupts allows you to switch to LPM
  - Start a 20ms timer from the switch interrupt, don't leave LPM
  - On timer interrupt process switch
    - If more than a few operations, set flag and leave LPM for processing

ADC

- MSP430G2xxx has one ADC10
- Resolution is 10 bits (can also be set to 8 bits)
  - Other MSP devices have 12 or 14 bit ADCs
- Multiple input pins can be connected to ADC
- Sample and hold is available
- Device can be programmed for one shot or continuous
- Continuous operation can be
  - on one port
  - round robin sweep between a number of ports
  - prioritised sweep between a number of ports


Clock
SAR
EOC
D_{N-1}
D_{N-2}
D_2
D_1
D_0
V_{REF}
DAC
Comparator
V_{IN}
S/H

Successive Approximation

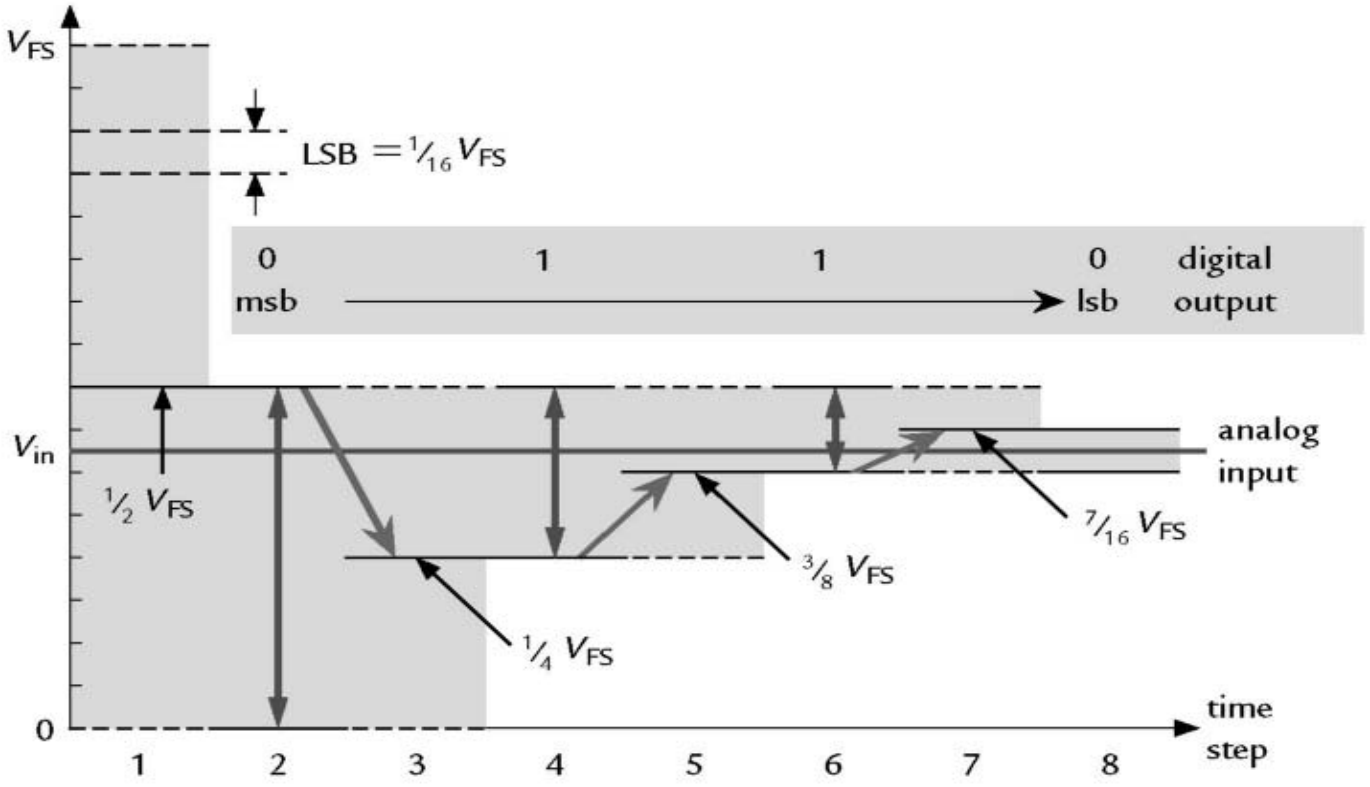
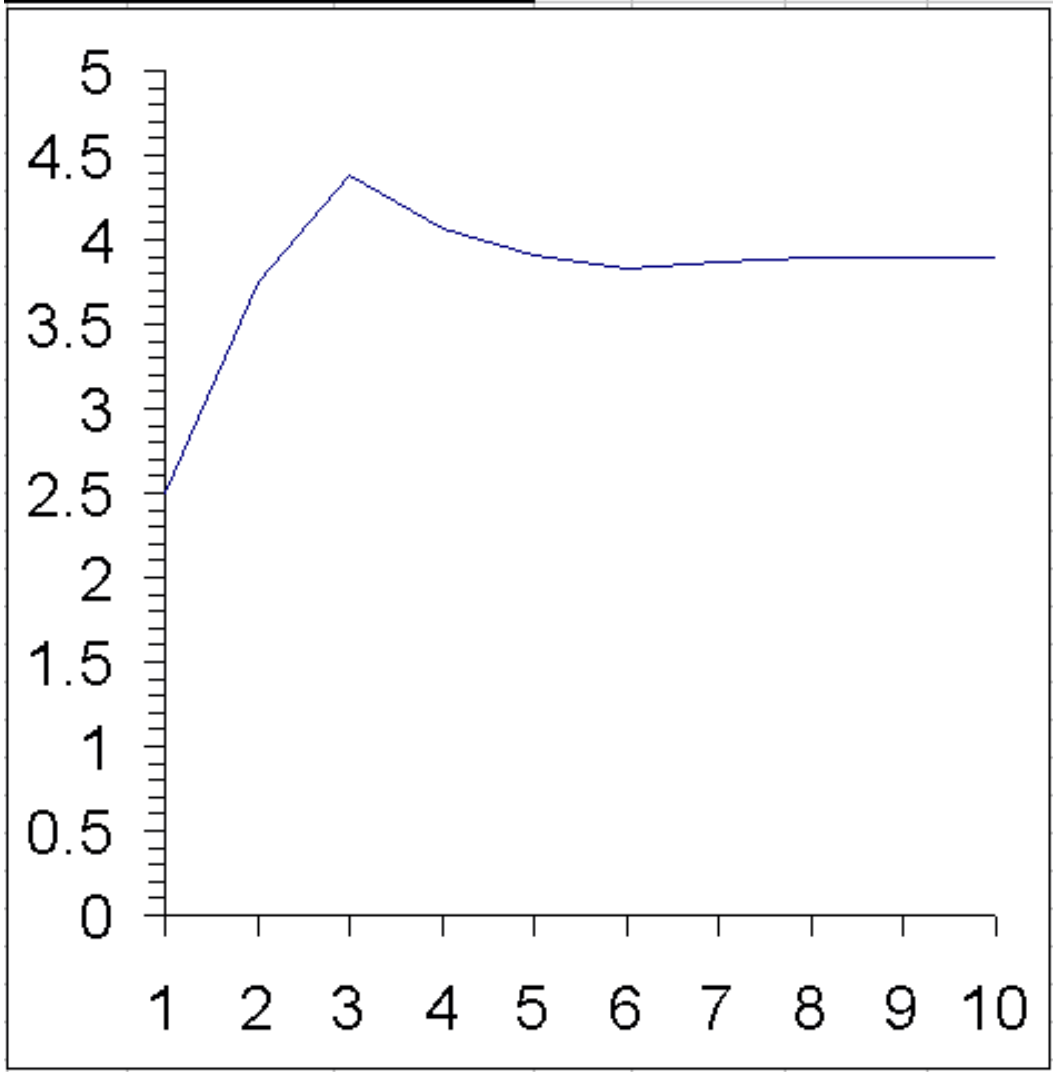


Figure 9.13: Operation of a 4-bit successive-operation ADC with an input of Vin = 0.4 V_{FS}.

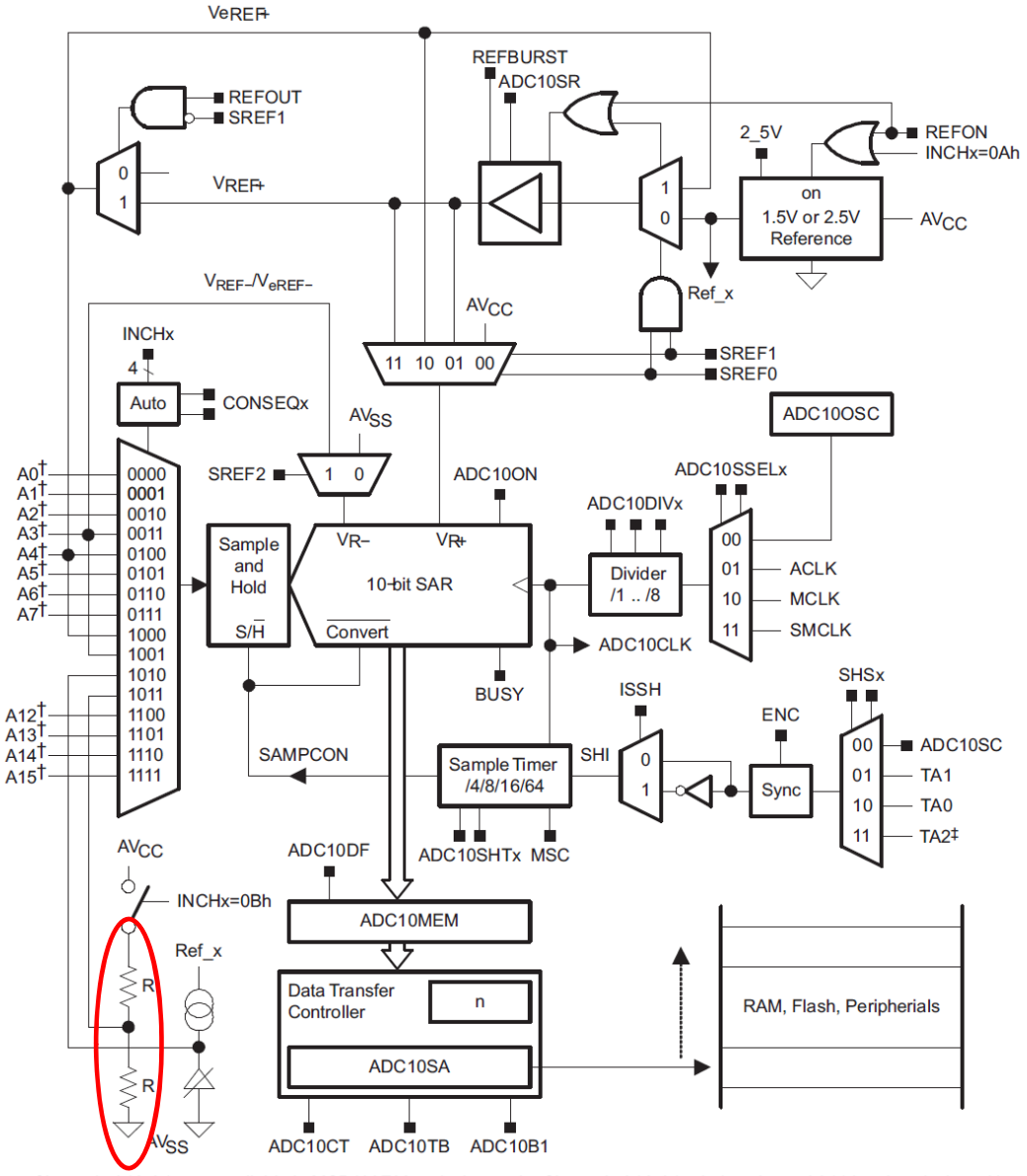
V_{REF} =	5	Volts
V_{in} =	3.9	Volts

1100011101



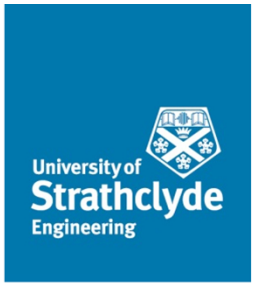
Russ Puskarcik

MSP430G2553
ADC10



†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

Using the MSP430 ADC



- Configure the pin
- Configure clock source, operation mode
- Configure ADC mux and positive and negative references
- If necessary, configure the interrupt

Using the ADC

Configure clock source, operation mode

Need to set ADCCTL0, ADCCTL1 and ADCCTL2

ADCCTL0 – bits 15 - 2

Sample and hold time 4 – 1024 cycles

- Can be calculated using an equivalent circuit *(textbook section 9.5.1 Single Conversion with the ADC10 Triggered by Software)* or set empirically.

- Turn the ADC on

Table 13-3. ADCCTL0 Register Description

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	ADCSHTx	RW	0h	ADC sample-and-hold time. These bits define the number of ADCCLK cycles in the sampling period for the ADC. Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0000b = 4 ADCCLK cycles 0001b = 8 ADCCLK cycles 0010b = 16 ADCCLK cycles 0011b = 32 ADCCLK cycles 0100b = 64 ADCCLK cycles 0101b = 96 ADCCLK cycles 0110b = 128 ADCCLK cycles 0111b = 192 ADCCLK cycles 1000b = 256 ADCCLK cycles 1001b = 384 ADCCLK cycles 1010b = 512 ADCCLK cycles 1011b = 768 ADCCLK cycles 1100b = 1024 ADCCLK cycles 1101b = 1024 ADCCLK cycles 1110b = 1024 ADCCLK cycles 1111b = 1024 ADCCLK cycles
7	ADCMSC	RW	0h	ADC multiple sample-and-conversion. Valid only for sequence or repeated modes. Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-convert. 1b = The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed.
6-5	Reserved	R	0h	Reserved. Always reads as 0.
4	ADCON	RW	0h	ADC on Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = ADC off 1b = ADC on
3-2	Reserved	R	0h	Reserved. Always reads as 0.

ADCTL1 bits 15 - 5

- Set sample & hold trigger source.
- Divide ADC clock

Table 13-4. ADCCTL1 Register Description

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-10	ADCSHSx	RW	0h	ADC sample-and-hold source select Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 00b = ADCSC bit 01b = Timer trigger 0 (see device-specific data sheet) 10b = Timer trigger 1 (see device-specific data sheet) 11b = Timer trigger 2 (see device-specific data sheet)
9	ADCSHP	RW	0h	ADC sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = SAMPCON signal is sourced from the sample input signal. 1b = SAMPCON signal is sourced from the sampling timer.
8	ADCISSH	RW	0h	ADC invert signal sample-and-hold Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = The sample input signal is not inverted. 1b = The sample input signal is inverted.
7-5	ADCDIVx	RW	0h	ADC clock divider Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 8

ADCTL1 bits 4-0

- ADC clock select
- Sampling mode

Table 13-4. ADCCTL1 Register Description (continued)

Bit	Field	Type	Reset	Description
4-3	ADCSSELx	RW	0h	ADC clock source select Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 00b = MODCLK 01b = ACLK 10b = SMCLK 11b = SMCLK
2-1	ADCCONSEQx	RW	0h	ADC conversion sequence mode select Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 00b = Single-channel single-conversion 01b = Sequence-of-channels 10b = Repeat-single-channel 11b = Repeat-sequence-of-channels
0	ADCBUSY	R	0h	ADC busy. This bit indicates an active sample or conversion operation. 0b = No operation is active. 1b = A sequence, sample, or conversion is active.

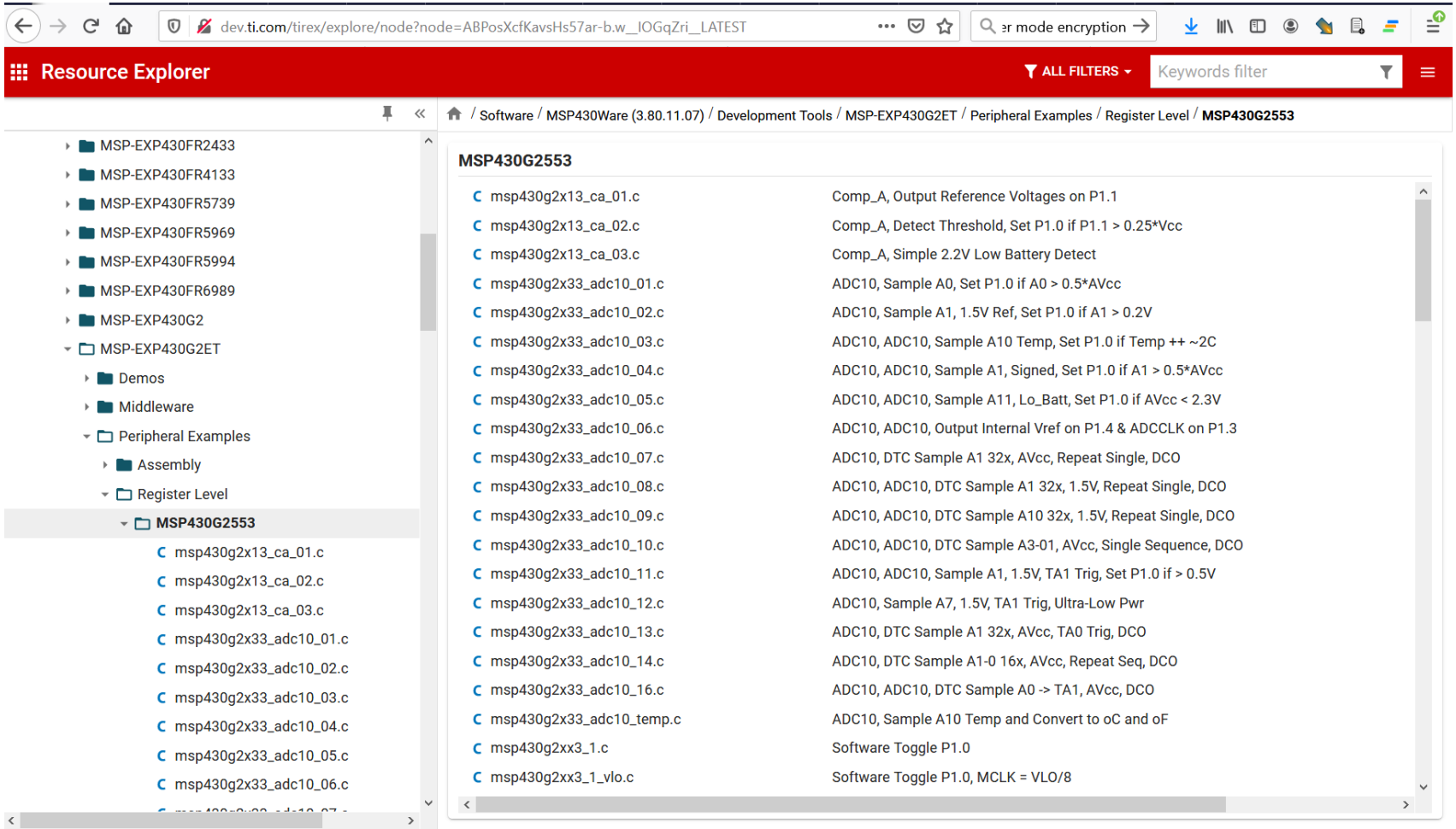
ADCTL2

- ADC clock configuration
- Resolution
- Buffering

Table 13-5. ADCCTL2 Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-8	ADCPDIVx	RW	0h	ADC predivider. This bit predivides the selected ADC clock source before it gets divided again using ADCDIVx. 00b = Predivide by 1 01b = Predivide by 4 10b = Predivide by 64 11b = Reserved
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4	ADCRES	RW	1h	ADC resolution. This bit defines the conversion result resolution. 0b = 8 bit (10 clock cycle conversion time) 1b = 10 bit (12 clock cycle conversion time)
3	ADCDF	RW	0h	ADC data read-back format. Data is always stored in the binary unsigned format. 0b = Binary unsigned. Theoretically the analog input voltage $-V_{REF}$ results in 0000h, the analog input voltage $+V_{REF}$ results in 03FFh. 1b = Signed binary (2s complement), left aligned. Theoretically the analog input voltage $-V_{REF}$ results in 8000h, the analog input voltage $+V_{REF}$ results in 7FC0h.
2	ADCSR	RW	0h	ADC sampling rate. This bit selects drive capability of the ADC reference buffer for the maximum sampling rate. Setting ADCSR reduces the current consumption of this buffer. 0b = ADC buffer supports up to approximately 200 ksps 1b = ADC buffer supports up to approximately 50 ksps
1	Reserved	R	0h	Reserved. Always reads as 0.
0	Reserved	RW	0h	Reserved. Must be written as 0.

TI Resources – Link on MyPlace



```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exit
    if (ADC10MEM < 0x1FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on
  }
}

// ADC10 interrupt service routine
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(ADC10_VECTOR))) ADC10_ISR (void)
#else
#error Compiler not supported!
#endif
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exit
    if (ADC10MEM < 0x1FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on
  }
}

// ADC10 interrupt service routine
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(ADC10_VECTOR))) ADC10_ISR (void)
#else
#error Compiler not supported!
#endif
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exit
    if (ADC10MEM < 0x1FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on
  }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exit
    if (ADC10MEM < 0x1FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on
  }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exit
    if (ADC10MEM < 0x1FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on
  }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

The Tricks!


```
__bis_SR_register(CPUOFF + GIE);
```

- Set (BIt Set) CPUOFF flag to switch off CPU
- Set GIE flag to enable interrupts

```
__bic_SR_register_on_exit(CPUOFF);
```

- Set (BIt Clear) CPUOFF flag to switch on CPU on return from the interrupt

Power Efficient Code

University of Strathclyde Engineering

- Effective use of low power modes can very significantly reduce power consumption – one 3000th in LPM4
- Most of the time system is not actually doing that much
- Generally,
  - Keep in low power mode, using only the necessary clocks
  - Wake using interrupts
  - If necessary, come out of LPM on return from interrupt, do processing, then return to LPM