MSP430FR5969: ADC 2 analog input read

sadasivam arumu... Intellectual 940 points
Community Member

Part Number: MSP430FR5969

Hi,

I am using MSP430FR5969.

1. For 12-bit ADC. I want to read analog data from A11 and A13. But, its not the sequencial input. So, how to configure the CONSEQ bit in ADC12CTL1 register. Can we use single channel mode

2. Does the single channel-single read mode reads the data continosly?

3. If i want to read independent timing, how do configurations?

over 5 years ago

Johnson He over 5 years ago TI__Mastermind 28496 points

Hi sadasivam,

MSP430FR5969 ADC_12 module support 4 mode:(you can set via *ADC12CONSEQx* register)

00b = Single-channel, single-conversion
01b = Sequence-of-channels
10b = Repeat-single-channel
11b = Repeat-sequence-of-channels

For sequence channels mode, ADC module will automatically convert the values of A0 -> Ax channels in sequence.

If you would like to read A11 and A13 channel value, you need to modify the channel manually.

I write a example code for A11 and A13 as your reference:

```
#include <msp430.h>


unsigned int ADC11_Value = 0;
unsigned int ADC13_Value = 0;

int main(void)
{
  WDTCTL = WDTPW | WDTHOLD;                 // Stop WDT

  P4SEL1 |= BIT3;                           // Configure P4.3 for ADC11
  P4SEL0 |= BIT3;

  P3SEL1 |= BIT1;                           // Configure P3.1 for ADC13
  P3SEL0 |= BIT1;

  // Disable the GPIO power-on default high-impedance mode to activate
  // previously configured port settings
  PM5CTL0 &= ~LOCKLPM5;

  // Configure ADC12
  ADC12CTL0 = ADC12SHT0_2 | ADC12ON;        // Sampling time, S&H=16, ADC12 on
  ADC12CTL1 = ADC12SHP;                     // Use sampling timer
  ADC12CTL2 |= ADC12RES_2;                  // 12-bit conversion results
  ADC12MCTL0 |= ADC12INCH_11;               // A11 ADC input select; Vref=AVCC
  ADC12IER0 |= ADC12IE0;                    // Enable ADC conv complete interrupt

  while (1)
  {
    __delay_cycles(5000);
    ADC12CTL0 |= ADC12ENC | ADC12SC;        // Start sampling/conversion

    __bis_SR_register(LPM0_bits | GIE);     // LPM0, ADC12_ISR will force exit
    __no_operation();                       // For debugger
  }
}

#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(ADC12_VECTOR))) ADC12_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(ADC12IV, ADC12IV_ADC12RDYIFG))
  {
    case ADC12IV_NONE:        break;        // Vector  0:  No interrupt
    case ADC12IV_ADC12OVIFG:  break;        // Vector  2:  ADC12MEMx Overflow
    case ADC12IV_ADC12TOVIFG: break;        // Vector  4:  Conversion time overflow
    case ADC12IV_ADC12HIIFG:  break;        // Vector  6:  ADC12BHI
    case ADC12IV_ADC12LOIFG:  break;        // Vector  8:  ADC12BLO
    case ADC12IV_ADC12INIFG:  break;        // Vector 10:  ADC12BIN
    case ADC12IV_ADC12IFG0:                 // Vector 12:  ADC12MEM0 Interrupt
      if((ADC12MCTL0 & 0x001f) == ADC12INCH_11)
      {
          ADC11_Value = ADC12MEM0;
          ADC12CTL0 &= ~ADC12ENC;           // Disable ADC
          ADC12MCTL0 &= 0xffe0;             // clear ADC channel
          ADC12MCTL0 |= ADC12INCH_13;       // set ADC channel to A13

      }
      else if((ADC12MCTL0 & 0x001f) == ADC12INCH_13)
      {
          ADC13_Value = ADC12MEM0;
          ADC12CTL0 &= ~ADC12ENC;           // Disable ADC
          ADC12MCTL0 &= 0xffe0;             // clear ADC channel
          ADC12MCTL0 |= ADC12INCH_11;       // set ADC channel to A11

      }
      __bic_SR_register_on_exit(LPM0_bits); // Exit active CPU
      break;                                // Clear CPUOFF bit from 0(SR)
    case ADC12IV_ADC12IFG1:   break;        // Vector 14:  ADC12MEM1
    case ADC12IV_ADC12IFG2:   break;        // Vector 16:  ADC12MEM2
    case ADC12IV_ADC12IFG3:   break;        // Vector 18:  ADC12MEM3
    case ADC12IV_ADC12IFG4:   break;        // Vector 20:  ADC12MEM4
    case ADC12IV_ADC12IFG5:   break;        // Vector 22:  ADC12MEM5
    case ADC12IV_ADC12IFG6:   break;        // Vector 24:  ADC12MEM6
    case ADC12IV_ADC12IFG7:   break;        // Vector 26:  ADC12MEM7
    case ADC12IV_ADC12IFG8:   break;        // Vector 28:  ADC12MEM8
    case ADC12IV_ADC12IFG9:   break;        // Vector 30:  ADC12MEM9
    case ADC12IV_ADC12IFG10:  break;        // Vector 32:  ADC12MEM10
    case ADC12IV_ADC12IFG11:  break;        // Vector 34:  ADC12MEM11
    case ADC12IV_ADC12IFG12:  break;        // Vector 36:  ADC12MEM12
    case ADC12IV_ADC12IFG13:  break;        // Vector 38:  ADC12MEM13
    case ADC12IV_ADC12IFG14:  break;        // Vector 40:  ADC12MEM14
    case ADC12IV_ADC12IFG15:  break;        // Vector 42:  ADC12MEM15
    case ADC12IV_ADC12IFG16:  break;        // Vector 44:  ADC12MEM16
    case ADC12IV_ADC12IFG17:  break;        // Vector 46:  ADC12MEM17
    case ADC12IV_ADC12IFG18:  break;        // Vector 48:  ADC12MEM18
    case ADC12IV_ADC12IFG19:  break;        // Vector 50:  ADC12MEM19
    case ADC12IV_ADC12IFG20:  break;        // Vector 52:  ADC12MEM20
    case ADC12IV_ADC12IFG21:  break;        // Vector 54:  ADC12MEM21
    case ADC12IV_ADC12IFG22:  break;        // Vector 56:  ADC12MEM22
    case ADC12IV_ADC12IFG23:  break;        // Vector 58:  ADC12MEM23
    case ADC12IV_ADC12IFG24:  break;        // Vector 60:  ADC12MEM24
    case ADC12IV_ADC12IFG25:  break;        // Vector 62:  ADC12MEM25
    case ADC12IV_ADC12IFG26:  break;        // Vector 64:  ADC12MEM26
    case ADC12IV_ADC12IFG27:  break;        // Vector 66:  ADC12MEM27
    case ADC12IV_ADC12IFG28:  break;        // Vector 68:  ADC12MEM28
    case ADC12IV_ADC12IFG29:  break;        // Vector 70:  ADC12MEM29
```

```
      case ADC12IV_ADC12IFG30:  break;        // Vector 72:  ADC12MEM30
      case ADC12IV_ADC12IFG31:  break;        // Vector 74:  ADC12MEM31
      case ADC12IV_ADC12RDYIFG: break;        // Vector 76:  ADC12RDY
      default: break;
    }
  }
```

Best Regards

Johnson

sadasivam arumugam *over 5 years ago in reply to Johnson He* Intellectual 940 points

It's works for me. Thanks.

efficiently manage power, accurately sense and transmit data and provide the core control or processing in their designs.