Analog To Digital Converter (ADC)

- Analog to Digital Conversion is used when we want to interface an external analog signal or when interfacing analog sensors, like for example a temperature sensor.
- The ADC block in LPC1768 Microcontroller is based on Successive Approximation Register(SAR) conversion method.
- LPC1768 ADC Module uses 12-bit SAR.
- The Measurement range is from VREFN to VREFP, or commonly from 0V to 3.3 V.
- Maximum of 8 multiplexed inputs can be used for ADC.

Analog To Digital Converter (ADC)

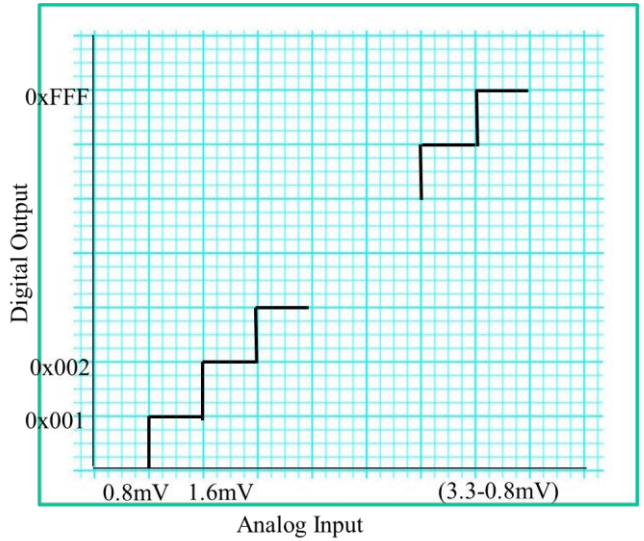
Analog voltage and Digital value of the ADC are related as follows:

$V_A = (V_{REFP} / 2^N)$ (Decimal Equivalent of Digital value)

V_A is the Input analog voltage
V_{REFP} Is the Reference voltage of ADC
N is the number of bits

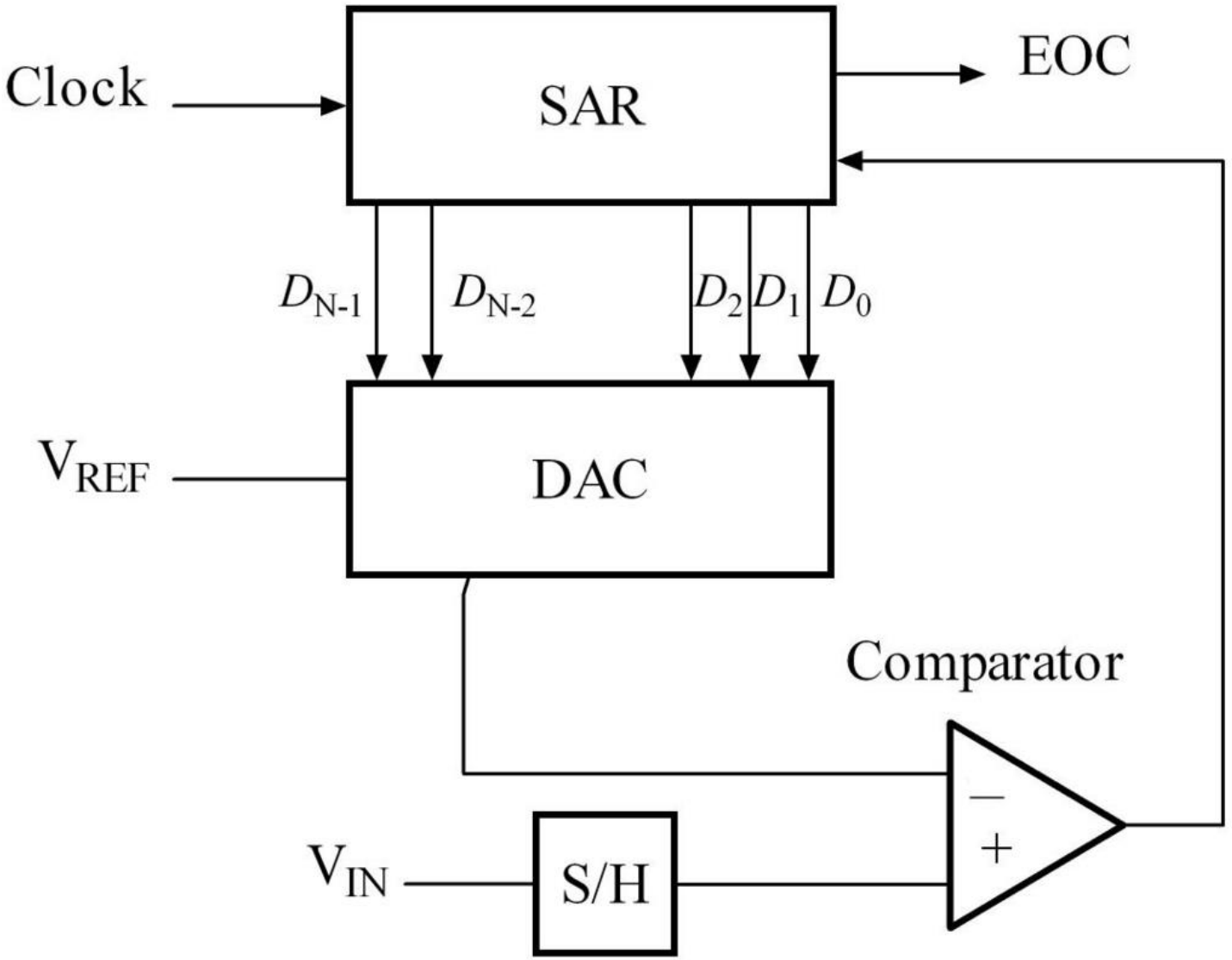
$(V_{REFP} / 2^N)$ is Resolution; It is a constant for given value of N and V_{REFP}

Digital output is directly proportional to Analog input voltage

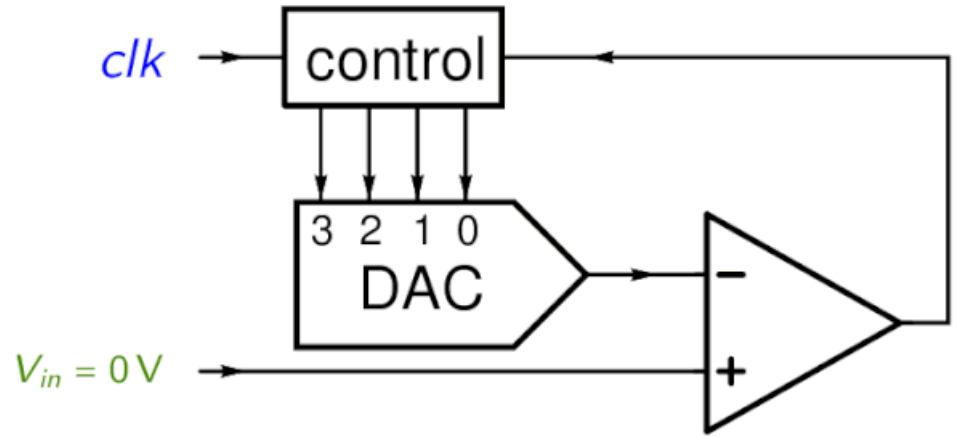


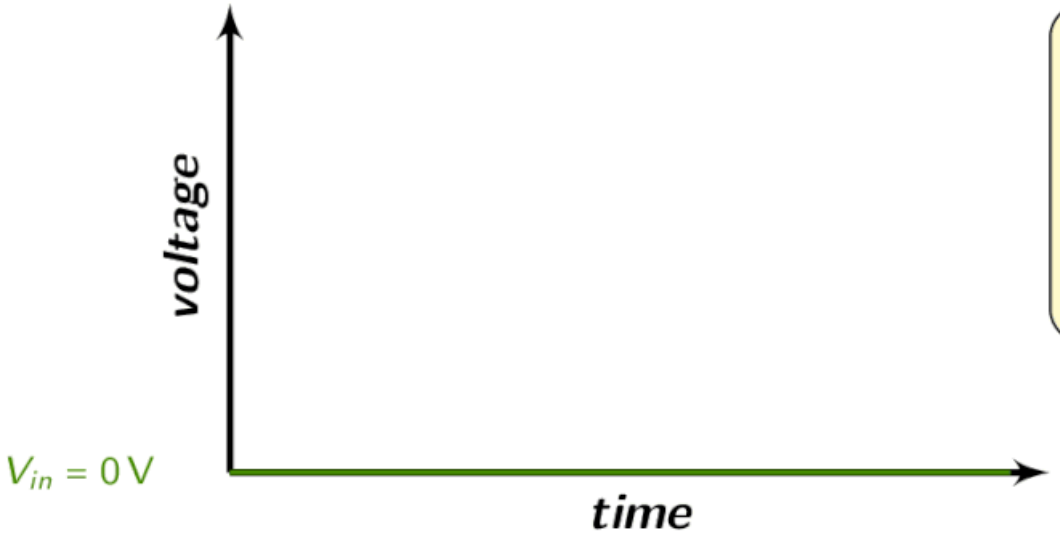
For 3.3 V, N=12, Resolution is 0.805 mV. i.e 0.805 mV change at the input creates ±1 change at the digital output.

When analog voltage is 0 mV output decimal value is 0
When analog voltage is 0.805 mV output decimal value is 1
When analog voltage is (0.805x2 =1.6) mV output decimal value is 2
When analog voltage is (0.805x3 =2.4) mV output decimal value is 3
When analog voltage is (0.805x4095 = 3.3-0.805) mV output decimal value is 4095


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 – example of a 4-bit ADC





Resolution:	
5 V × 1/2	2.5000 V
5 V × 1/4	1.2500 V
5 V × 1/8	0.6250 V
5 V × 1/16	0.3125 V
...	
5 V × 1/1024	0.0049 V

Register	Description
ADCR	A/D COntrol Register: Used for Configuring the ADC
ADGDR	A/D Global Data Register: This register contains the ADC's DONE bit and the result of the most recent A/D conversion
ADINTEN	A/D Interrupt Enable Register
ADDR0 - ADDR7	A/D Channel Data Register: Contains the recent ADC value for respective channel
ADSTAT	A/D Status Register: Contains DONE & OVERRUN flag for all the ADC channels

ADC Register Configuration

Now lets see how to configure the individual registers for ADC conversion.

ADCR								
31:28	27	26:24	23:22	21	20:17	16	15:8	7:0
Reserved	EDGE	START	Reserved	PDN	Reserved	BURST	CLCKDIV	SEL

ADGDR (ADC Global Data Register)

ADGDR					
31	27	26:24	23:16	15:4	3:0
DONE	OVERRUN	CHN	Reserved	RESULT	Reserved

Bit 15:4 - RESULT
This field contains the 12bit A/D conversion value for the selected channel in **ADCR.SEL**
The vale for this register should be read oncve the conversion is completed ie DONE bit is set.

Bit 26:24 - CHN : Channel
These bits contain the channel number for which the A/D conversion is done and the converted value is available in RESULT bits(e.g. 000 identifies channel 0, 011 channel 3...).

Bit 27 - OVERRUN
This bit is set during the BURST mode where the previous conversion data is overwritten by the new A/D conversion value.

Bit 31 - DONE
This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started.

4) **ADDR0 to ADDR7 – A/D Data registers :** This register contains the result of the most recent conversion completed on the corresponding channel [0 to 7]. Its structure is same as ADGDR except Bits[26:24] are not used/reserved.

5) **ADSTAT – A/D Status register :** This register contains DONE and OVERRUN flags for all of the A/D channels along with A/D interrupt flag.

1. **Bits[7:0] – DONE[7 to 0]:** Here xth bit mirrors DONEx status flag from the result register for A/D channel x.
2. **Bits[15:8] – OVERRUN[7 to 0]:** Even here the xth bit mirrors OVERRUNx status flag from the result register for A/D channel x
3. **Bit 16 – ADINT:** This bit represents the A/D interrupt flag. It is 1 when any of the individual A/D channel DONE flags is asserted and enabled to contribute to the A/D interrupt via the ADINTEN(given below) register.
4. Other bits are reserved.

Analog To Digital Converter (ADC)

Pin	Type	Description
AD0.7 to AD0.0	Input	**Analog Inputs.** The ADC cell can measure the voltage on any of these input signals. Digital signals are disconnected from the ADC input pins when the ADC function is selected on that pin in the Pin Select register.

ADCR – A/D Control Register

Bit	Symbol	Value	Description
7:0	SEL		Selects which of the AD0.7:0 pins is (are) to be sampled and converted. For AD0, bit 0 selects Pin AD0.0, and bit 7 selects pin AD0.7. In software-controlled mode, only one of these bits should be 1. In hardware scan mode, any value containing 1 to 8 ones is allowed. All zeroes is equivalent to 0x01.
15:8	CLKDIV		The APB clock (PCLK_ADC0) is divided by (this value plus one) to produce the clock for the A/D converter, which should be less than or equal to 13 MHz. Typically, software should program the smallest value in this field that yields a clock of 13 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable.
16	BURST	1	The AD converter does repeated conversions at up to 200 kHz, scanning (if necessary) through the pins selected by bits set to ones in the SEL field. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1-bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion that's in progress when this bit is cleared will be completed. **Remark:** START bits must be 000 when BURST = 1 or conversions will not start. If BURST is set to 1, the ADGINTEN bit in the AD0INTEN register (Table 534) must be set to 0.
		0	Conversions are software controlled and require 65 clocks.
20:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

AD0.0 –P0.23 FN 01
AD0.1-P0.24 FN 01
AD0.2-P0.25 FN 01
AD0.3-P0.26 FN 01
AD0.4-P0.30 FN 03
AD0.5-P0.31 FN 03
AD0.6-P0.3 FN 02
AD0.7-P0.2 FN 02

Analog To Digital Converter (ADC)

ADCR – A/D Control Register

21	PDN	1	The A/D converter is operational.
		0	The A/D converter is in power-down mode.
23:22	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
26:24	START		When the BURST bit is 0, these bits control whether and when an A/D conversion is started:
		000	No start (this value should be used when clearing PDN to 0).
		001	Start conversion now.
		010	Start conversion when the edge selected by bit 27 occurs on the P2.10 / EINT0 / NMI pin.
		011	Start conversion when the edge selected by bit 27 occurs on the P1.27 / CLKOUT / USB_OVRCRn / CAP0.1 pin.
		100	Start conversion when the edge selected by bit 27 occurs on MAT0.1. Note that this does not require that the MAT0.1 function appear on a device pin.
		101	Start conversion when the edge selected by bit 27 occurs on MAT0.3. Note that it is not possible to cause the MAT0.3 function to appear on a device pin.
		110	Start conversion when the edge selected by bit 27 occurs on MAT1.0. Note that this does not require that the MAT1.0 function appear on a device pin.
		111	Start conversion when the edge selected by bit 27 occurs on MAT1.1. Note that this does not require that the MAT1.1 function appear on a device pin.
27	EDGE		This bit is significant only when the START field contains 010-111. In these cases:
		1	Start conversion on a falling edge on the selected CAP/MAT signal.
		0	Start conversion on a rising edge on the selected CAP/MAT signal.
31:28	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

Analog To Digital Converter (ADC)

A/D Global Data Register (ADGDR) :

The A/D Global Data Register holds the result of the most recent A/D conversion that has completed, and also includes copies of the status flags that go with that conversion. Results of ADC conversion can be read in one of two ways. One is to use the A/D Global Data Register to read all data from the ADC. Another is to use the A/D Channel Data Registers.

Bit	Symbol	Description
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin selected by the SEL field, as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}.
23:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
26:24	CHN	These bits contain the channel from which the RESULT bits were converted (e.g. 000 identifies channel 0, 001 channel 1...).
29:27	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits. This bit is cleared by reading this register.
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

Analog To Digital Converter (ADC)

A/D Data Registers (ADDR0 to ADDR7)

The A/D Data Registers hold the result of the last conversion for each A/D channel, when an A/D conversion is complete. They also include the flags that indicate when a conversion has been completed and when a conversion overrun has occurred.

Bit	Symbol	Description
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin, as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}.
29:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits.This bit is cleared by reading this register.
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

Analog To Digital Converter (ADC)

A/D Interrupt Enable register (ADINTEN) : This register allows control over which A/D channels generate an interrupt when a conversion is complete.

Bit	Symbol	Value	Description
0	ADINTEN0	0	Completion of a conversion on ADC channel 0 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 0 will generate an interrupt.
1	ADINTEN1	0	Completion of a conversion on ADC channel 1 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 1 will generate an interrupt.
2	ADINTEN2	0	Completion of a conversion on ADC channel 2 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 2 will generate an interrupt.
3	ADINTEN3	0	Completion of a conversion on ADC channel 3 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 3 will generate an interrupt.
4	ADINTEN4	0	Completion of a conversion on ADC channel 4 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 4 will generate an interrupt.
5	ADINTEN5	0	Completion of a conversion on ADC channel 5 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 5 will generate an interrupt.
6	ADINTEN6	0	Completion of a conversion on ADC channel 6 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 6 will generate an interrupt.
7	ADINTEN7	0	Completion of a conversion on ADC channel 7 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 7 will generate an interrupt.
8	ADGINTEN	0	Only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts. **Remark:** This bit must be set to 0 in burst mode (BURST = 1 in the AD0CR register).
		1	Only the global DONE flag in ADDR is enabled to generate an interrupt.
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

A/D Status register (ADSTAT) :

The A/D Status register allows checking the status of all A/D channels simultaneously. The DONE and OVERRUN flags appearing in the ADDRn register for each A/D channel are mirrored in ADSTAT. The interrupt flag (the logical OR of all DONE flags) is also found in ADSTAT.

Bit	Symbol	Description
0	DONE0	This bit mirrors the DONE status flag from the result register for A/D channel 0.
1	DONE1	This bit mirrors the DONE status flag from the result register for A/D channel 1.
2	DONE2	This bit mirrors the DONE status flag from the result register for A/D channel 2.
3	DONE3	This bit mirrors the DONE status flag from the result register for A/D channel 3.
4	DONE4	This bit mirrors the DONE status flag from the result register for A/D channel 4.
5	DONE5	This bit mirrors the DONE status flag from the result register for A/D channel 5.
6	DONE6	This bit mirrors the DONE status flag from the result register for A/D channel 6.
7	DONE7	This bit mirrors the DONE status flag from the result register for A/D channel 7.
8	OVERRUN0	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 0.
9	OVERRUN1	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 1.
10	OVERRUN2	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 2.
11	OVERRUN3	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 3.
12	OVERRUN4	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 4.
13	OVERRUN5	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 5.
14	OVERRUN6	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 6.
15	OVERRUN7	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 7.
16	ADINT	This bit is the A/D interrupt flag. It is one when any of the individual A/D channel Done flags is asserted and enabled to contribute to the A/D interrupt via the ADINTEN register.

Steps for Configuring ADC

Below are the steps for configuring the LPC1768 ADC.

1. Configure the GPIO pin for ADC function using PINSEL register.
2. Enable the CLock to ADC module.
3. Deselect all the channels and Power on the internal ADC module by setting ADCR.PDN bit.
4. Select the Particular channel for A/D conversion by setting the corresponding bits in ADCR.SEL
5. Set the ADCR.START bit for starting the A/D conversion for selected channel.
6. Wait for the conversion to complete, ADGR.DONE bit will be set once conversion is over.
7. Read the 12-bit A/D value from ADGR.RESULT.
8. Use it for further processing or just display on LCD.

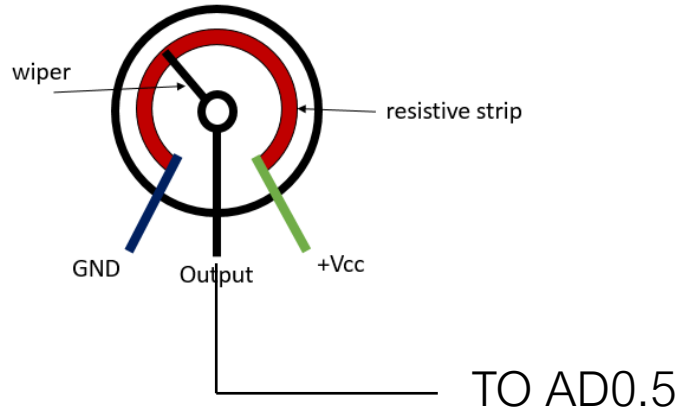
ADC software mode for 2-channel concurrent conversion

```
#include<LPC17xx.h>
 #include<stdio.h>
int main(void)
{
        unsigned long temp4, temp5;
        unsigned int i;

        SystemInit();
        SystemCoreClockUpdate();
        LPC_PINCON->PINSEL3 = (3<<28) | (3<<30);          //P1.30 as AD0.4 and P1.31 as AD0.5
        LPC_ADC->ADINTEN = 0;
        while(1)
        {
                LPC_ADC->ADCR = (1<<4)|(1<<21)|(1<<24);//;          //ADC0.4, start conversion and operational
                for(i=0;i<2000;i++);
                while(((temp4=LPC_ADC->ADDR4) & (1<<31)) == 0);   //wait till 'done' bit is 1, indicates conversion complete
                temp4 = LPC_ADC->ADDR4;
                temp4 >>= 4;
                temp4  &= 0x00000FFF;                             //12 bit ADC

                LPC_ADC->ADCR = (1<<5)|(1<<21)|(1<<24);//          //ADC0.5, start conversion and operational
                for(i=0;i<2000;i++);                              //delay for conversion
                while(((temp5=LPC_ADC->ADDR5) & (1<<31)) == 0);   //wait till 'done' bit is 1, indicates conversion complete
                temp5 = LPC_ADC->ADDR5;
                temp5 >>= 4;
                temp5 &= 0x00000FFF;                              //12 bit ADC
//Now you can use temp4 and temp5 for further processing based on your requirement
        }
}
```

Input Analog voltage and display its digital equivalent on LCD

```
include<LPC17xx.h>
 #include<stdio.h>
 #define      Ref_Vtg                    3.300
 #define      Full_Scale   0xFFF//12 bit ADC
unsigned long int init_command[] = {0x30,0x30,0x30,0x20,0x28,0x0c,0x06,0x01,0x80};
int main(void)
{
flag1 =0;//Command
                unsigned long adc_temp;
                unsigned int i;
                float in_vtg;
                unsigned char vtg[7], dval[7];
                unsigned char Msg3[] = {"ANALOG IP:"};
                unsigned char Msg4[] = {"ADC OUTPUT:"};
                SystemInit();
                SystemCoreClockUpdate();
                 lcd_init();//Initialize LCD
                  LPC_PINCON->PINSEL3 |= 3<<30;              //P1.31 as AD0.5
                LPC_SC->PCONP |= (1<<12);//enable the peripheral ADC
                 for (i=0; i<9;i++)
                      { temp1 = init_command[i];
                                lcd_write(); //send Init commands to LCD   }
                flag1=1;//Data
                  i =0;
                 while (Msg3[i++] != '\0')
                    {  temp1 = Msg3[i];
                    lcd_write();//Send data bytes}
```

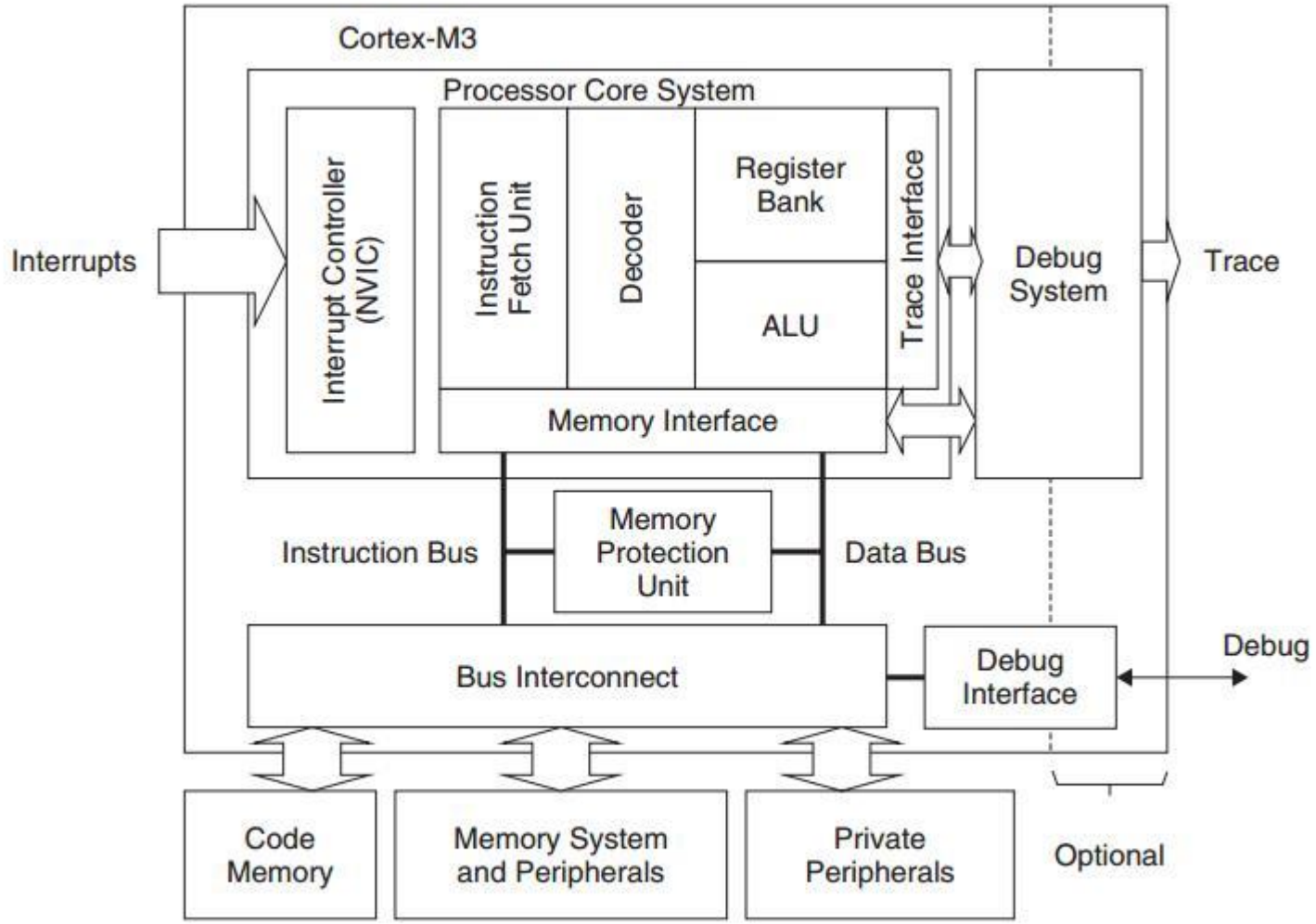
```
ANALOG INPUT 1.1V
ADC OUTPUT 555
```



```
                flag1=0; //Command
                    temp1 = 0xC0;//Cursor at beginning of second line
                  lcd_write();
                  flag1=1;
                    i =0;
                  while (Msg4[i++] != '\0')
                    {
                     temp1 = Msg4[i];
                     lcd_write();//Send data bytes
                      }
while(1)
{
              LPC_ADC->ADCR = (1<<5)|(1<<21)|(1<<24);//ADC0.5, start conversion and operational
                 while(((adc_temp=LPC_ADC->ADGDR) & (1<<31)) == 0);
              adc_temp = LPC_ADC->ADGDR;
              adc_temp >>= 4;
              adc_temp &= 0x00000FFF;          //12 bit ADC
              in_vtg = (((float)adc_temp * (float)Ref_Vtg))/((float)Full_Scale);          //calculating input analog voltage
              sprintf(vtg,"%3.2fV",in_vtg);          //convert the readings into string to display on LCD
              sprintf(dval,"%x",adc_temp);
              flag1=0;;
              temp1 = 0x8A;
              lcd_write();
              flag1=1;
                    i =0;
                  while (vtg[i++] != '\0')
                    {
                     temp1 = vtg[i];
                     lcd_write();//Send data bytes
                      }
```

```
flag1=0;
temp1 = 0xCB;
lcd_write();
flag1=1;
  i =0;
 while (dval[i++] != '\0')
   {
     temp1 = dval[i];
     lcd_write();//Send data bytes
     }
    for(i=0;i<7;i++)
    vtg[i] = dval[i] = 0;
}
}
```

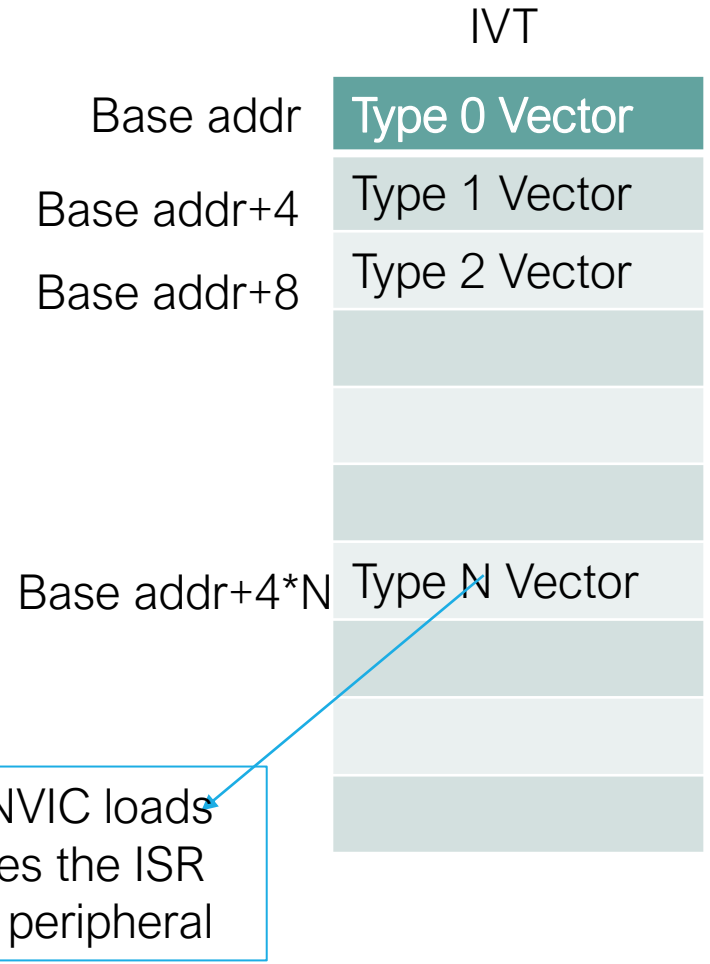
Nested Vectored Interrupt Controller(NVIC)



Nested Vectored Interrupt Controller(NVIC)

- Controls system exceptions and peripheral interrupts
- In the LPC176x, the NVIC supports 35 vectored interrupts
- Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller.
- Interrupt numbers relate to where entries are stored in the Interrupt vector table.

- Interrupt Vector – Is the address of Interrupt Service Subroutine (ISR)
- Interrupt vector of Interrupt Type N is stored at an offset N*4 from the base address of Interrupt Vector Table(IVT)
- If the peripheral device is enabled to generate Interrupt when some event



Nested Vectored Interrupt Controller(NVIC)

- If the peripheral device is enabled to generate Interrupt when some event occurs, the INTR request is sent to NVIC
- If NVIC is enabled to service the INTR request from the peripheral, it services the INTR request by executing ISR pertaining to the peripheral.(i.e Save the return address, Get the Interrupt Vector from IVT and load that address to PC. Upon completion of ISR execution resumes the calling function)

Nested Vectored Interrupt Controller(NVIC)

- In case of Timer, there are 6 INTR enable flags (4 in MCR and 2 in CCR. i.e 4 Match events and 2 Capture events can generate the Interrupt when the event occurs)
- When the event occurs the corresponding bit is set automatically in the IR register. This indicates the NVIC about the event
- If the NVIC is enabled to service the Timer Interrupt, it executes the corresponding ISR and gives the desired service to the Timer.
- In the ISR, clear the corresponding bit, by writing back 1.

Timer/Counter Interrupt Programming

Interrupt Register (IR)

The Interrupt Register consists of 4 bits for the match interrupts and 2 bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low.

Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Bit	Symbol	Description
0	MR0 Interrupt	Interrupt flag for match channel 0.
1	MR1 Interrupt	Interrupt flag for match channel 1.
2	MR2 Interrupt	Interrupt flag for match channel 2.
3	MR3 Interrupt	Interrupt flag for match channel 3.
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.

Timer/Counter Interrupt Programming

Toggle LED connected to p0.2 every second while displaying the status of switch connected to P1.0 on the LED connected to P2.0

```
#include<stdio.h>
#include<LPC17xx.h>
unsigned int ticks=0,x;
void TIMER0_IRQHandler(void)
{
LPC_TIM0->IR = 1;
                ticks++;
                if(ticks==1000)
                {
                                ticks=0;
                                LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
                }
                                }
void init_timer0(void)
{
                LPC_TIM0->TCR = 0x00000002;     // Timer0 Reset
                LPC_TIM0->CTCR =0x00;//Timer
                LPC_TIM0->MR0 = 2999; //  For 1ms
                LPC_TIM0->EMR = 0X00;//Do nothing for EM0
                LPC_TIM0->PR = 0;
                LPC_TIM0->MCR = 0x00000003;   //Reset TC upon Match-0 and generate INTR
                LPC_TIM0->TCR = 0x00000001;     // Timer0 Enable

                return;
```

Timer/Counter Interrupt Programming

```
int main(void)
{
        LPC_GPIO0->FIODIR=0x00000004;
        LPC_GPIO2->FIODIR=0x00000001;
        init_timer0();
        NVIC_EnableIRQ(TIMER0_IRQn);//timer 0 intr enabled in NVIC
                while(1)
                {
                        LPC_GPIO2->FIOPIN=(LPC_GPIO1->FIOPIN & 0x01) ;
                }

}
```

Timer/Counter Interrupt Programming

Toggle P0.2 whenever counter value reaches 3. I. e for every 4 edges using the counter interrupt.

```
#include<stdio.h>
#include<LPC17xx.h>
void TIMER0_IRQHandler(void)
{
                    LPC_TIM0->IR = 1;
                    LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
}
void init_timer0(void)
{       LPC_TIM0->TCR = 0x00000002;     // Timer0 Reset
        LPC_TIM0->CTCR =0x05; // Counter at +ve edge of CAP0.1
        LPC_TIM0->MR0 = 3;
        LPC_TIM0->EMR = 0X00;
        LPC_TIM0->PR = 0;
        LPC_TIM0->MCR = 0x00000003;
        LPC_TIM0->TCR = 0x00000001;     // Timer0 Enable
        return;
        }

 int main(void)
 {
          LPC_GPIO0->FIODIR=0x00000004;
          LPC_PINCON->PINSEL3 |=((3<<22)|(3<<24));
          init_timer0();
          NVIC_EnableIRQ(TIMER0_IRQn);
          while(1);
 }
```

43	P1.27	CLKOUT	USB_OVRCR	CAP0.1
44	P1.28	MC2A	PCAP1.0	MAT0.0

Timer/Counter Interrupt Programming

Timer interrupt for rectangular waveform generation (1.5 second HIGH and 0.5 second LOW)

```
include<stdio.h>
#include<LPC17xx.h>
unsigned char flag=1;
void TIMER0_IRQHandler(void)
{
LPC_TIM0->IR = 1;

if(flag)
        {
                flag=0;
                LPC_TIM0->TCR = 0x00000002;    // Timer0 Reset
                LPC_GPIO0->FIOCLR=0x00000004;
                LPC_TIM0->MR0 = 500;
                LPC_TIM0->TCR = 0x00000001;    // Timer0 Enable
        }
        else
                {
                flag=1;
                LPC_TIM0->TCR = 0x00000002;    // Timer0 Reset
                LPC_GPIO0->FIOSET=0x00000004;
                LPC_TIM0->MR0 = 1500;
                LPC_TIM0->TCR = 0x00000001;    // Timer0 Enable
                }

}
```

Timer/Counter Interrupt Programming

```
void init_timer0(void)
{
		LPC_TIM0->TCR = 0x00000002;	// Timer0 Reset
		LPC_TIM0->CTCR =0x00;
		LPC_TIM0->MR0 = 1500;
		LPC_TIM0->EMR = 0X00;
		LPC_TIM0->PR = 3000;
		LPC_TIM0->MCR = 0x00000005;
		LPC_TIM0->TCR = 0x00000001;	// Timer0 Enable
		LPC_GPIO0->FIOSET=0x00000004;
		return;
}
int main(void)
{
		LPC_GPIO0->FIODIR=0x00000004;
		init_timer0();
		NVIC_EnableIRQ(TIMER0_IRQn);
		while(1);
}
```