ADC Programming

- LPC1768 has an inbuilt 12-bit Successive Approximation ADC which is multiplexed among 8 input pins.
- The ADC reference voltage is measured across VREFN to VREFP,

  ➔The analog voltage b/w this range (VREFN to VREFP) is converted into its equivalent digital value.
- Generally, VREFP is connected to VDD and VREFN is connected to GND. As LPC1768 works on 3.3V, here the VDD is 3.3V.
- **ADC Resolution:** It is the smallest incremental voltage that can be recognized and thus causes a change in the digital output.
- Now the resolution of 12-bit ADC with VDD of 3.3V is,

  $$ADC = 3.3/(2^{12}) = 3.3/4096 = 0.000805 = 0.8mV$$
- ➔ B/W 3.3V to 0, whenever a 0.8mV change ADC can recognize it and produces its equivalent digital value

ADC output formula:

Digital value = (Analog input voltage)/ VAREF) * 2^n

D_{val} = (V_A)/ VaREF) * 2^n

Where n = number of bits of ADC digital output.

For example, for 12-bit ADC, VIN=1V, VAREF=3.3 V

Digital value = (1 V/3.3 V) *4096 = 1240d = 4D8h

Pin configuration for ADC

Adc Channel	Port Pin	Pin Functions	Associated PINSEL Register
AD0	P0.23	0-GPIO, 1-**AD0[0]**, 2-I2SRX_CLK, 3-CAP3[0]	14,15 bits of PINSEL1
AD1	P0.24	0-GPIO, 1-**AD0[1]**, 2-I2SRX_WS, 3-CAP3[1]	16,17 bits of PINSEL1
AD2	P0.25	0-GPIO, 1-**AD0[2]**, 2-I2SRX_SDA, 3-TXD3	18,19 bits of PINSEL1
AD3	P0.26	0-GPIO, 1-**AD0[3]**, 2-AOUT, 3-RXD3	20,21 bits of PINSEL1
AD4	P1.30	0-GPIO, 1-VBUS, 2- , 3-**AD0[4]**	28,29 bits of PINSEL3
AD5	P1.31	0-GPIO, 1-SCK1, 2- , 3-**AD0[5]**	30,31 bits of PINSEL3
AD6	P0.3	0-GPIO, 1-RXD0, 2-**AD0[6]**, 3-	6,7 bits of PINSEL0
AD7	P0.2	0-GPIO, 1-TXD0, 2-**AD0[7]**, 3-	4,5 bits of PINSEL0

ADC Registers

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

ADCR

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

ADCR – A/D Control Register :

- **Bits[7:0] – SEL:** Used to select a particular channel for ADC conversion. Setting the Bit-0 will make the ADC to sample AD0 for conversion.

- **Bits[15:8] – CLKDIV:** ADC Peripheral clock i.e. PCLK_ADC0 is divided by CLKDIV+1 to get the ADC clock. Note that ADC clock speed must be <= 13Mhz.

- **Bit[16] – BURST:** Set this to 1 for doing repeated conversions, else 0 for software controlled conversions.

- **Bit[21] – PDN :** Set it to 1 for powering up the ADC and making it operational. Set it to 0 for bringing it in power down mode.

- **Bits[26:24] – START:** When the BURST bit is 0, these bits control whether and when an A/D conversion should start:
- 000 = No start
- 001 = Start the conversion now
- The remaining cases (010 to 111) are about starting conversion on occurrence of edge on a particular CAP or MAT pin.

- **Bit[27] – EDGE:** This bit is used only when the START field contains 010-111.
  - 1= Start the conversion on falling edge of the selected CAP/MAT signal
  - 0= Start the conversion on rising edge of the selected signal.
- Other bits are reserved

ADGDR – A/D Global Data Register : Contains the ADC's flags and the result of the most recent A/D conversion.

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

- **Bits[15:4] – RESULT:** When DONE bit = 1, these bits indicate the binary equivalent of the analog voltage that was applied to D0.x. This analog voltage is b/w VREFP to VREFN.
  - Value 0x0 ➔ i/p voltage on the given pin was less than, equal to or greater than VREFN.
  - Value 0xFFF ➔ i/p voltage was close to, equal to or greater than the VREFP.
- **Bits[26:24] – CHN:** Represents the channel from which RESULT bits were obtained. 000= channel 0, 001= channel 1 and so on.

- **Bit[27] – OVERRUN:** In burst mode this bit is 1 in case of an Overrun i.e. the result of previous conversion being lost(overwritten). This bit will be cleared after reading this register (ADGDR).

- **Bit[31] – DONE:** This bit is set to 1 when an A/D conversion completes. It is cleared when this register(ADGDR) 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.

A/D Interrupt Enable register (ADINTEN)

Upon the completion of A/D conversion, this register allows the control of interrupt over a channel

- **Bits[7:0]** : Enable or disable the generation of interrupt upon the completion of a conversion on ADC channel [7:0].
  - Bit = 1, enables the interrupt
  - Bit = 0, disables the interrupt
- **Bit 8: ADGINTEN:**
  - When set to 0, only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts.
    - Remark: This bit must be set to 0 in burst mode
  - When set to 1, only the global DONE flag in ADDR is enabled to generate an interrupt.
- Other bits: Reserved

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.

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.
- It is important to use one method consistently because the DONE and OVERRUN flags can otherwise get out of synch between the ADGDR and the A/D Channel Data Registers, potentially causing erroneous interrupts or DMA activity.

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.

- **Bits[7:0] – DONE[7 to 0]:** Here xth bit mirrors DONEx status flag from the result register for A/D channel x.
- **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
- **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 register.
- Other bits are reserved.

ADC modes in LPC1768:

1.Software controlled mode : only one channel must be made active during a conversion. And hence, at a time only one conversion is possible. To convert again, you have to repeat the process.

2. Burst or Hardware mode :

- In Hardware Scan Mode or Burst Mode, the conversions will happen continuously on any number of channels starting from LSB (AD0.0) then progressing towards MSB (AD0.7) in round-robin fashion.
- Here, the conversions cannot be controlled by software, so overrun may occur in this mode. Overrun is the case when a previous conversion result is replaced by new conversion result without previous result being read i.e., the conversion is lost.
- Usually, an interrupt is used in Burst mode to get the latest conversion results. This interrupt is triggered when conversion in one of the selected channel ends.

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.

Setting up and configuring ADC Module for software controlled mode

- Configure the appropriate PINSEL register for the desired function depending on the channel used.
- To power up the ADC, first the corresponding bit in the PCONP register must be set and then the PDN bit in the ADCR must be set.
- Also the bit corresponding to the channel and the START bit must be set

```
    LPC_SC->PCONP |= (1<<12);
    LPC_ADC->ADCR = (1<<4)|(1<<21)|(1<<24) // channel ADC0.4
```

Fetching the conversion result in software controlled mode :

In software controlled mode we continuously monitor bit 31 in the corresponding channel data register ADDR. If bit 31 changes to 1 from 0, it means that current conversion has been completed and the result is ready. For example, if we are using channel 4 of AD0 then we monitor for changes in bit 31 as follows :

```
while(!(LPC_ADC->ADDR4 & 1<<31));
```

Then the result is extracted as follows:

```
adc_temp = (LPC_ADC->ADDR4>>4) & 0xFFF;
```

Setting up and configuring ADC Module for Burst mode

In burst mode the START bit in ADCR is not used. In this case we can select multiple channels for conversion when setting up ADCR. The conversions start as soon the ADCR is setup.

First the PINSEL register must be configured. Then configure the ADCR and ADINTEN as follows:

```
LPC_ADC->ADCR = (1 << 4 | 1 << 5 | 1 << 16 | 1 << 21);    //AD0.4.
AD0.5, Burst, PDN
LPC_ADC->ADINTEN = (1 << 4 | 1 << 5);   //Enable int
```

In Burst mode we use an ISR which triggers at the completion of a conversion in any one of the channel. Now, we just need to find the Channel for which the conversion was done. For this we fetch the channel number from ADGDR which also stores the conversion result. This can be read as follows:

```
int channel= (LPC_ADC->ADGDR>>24)  & 0x7; //Extract Channel Number
```

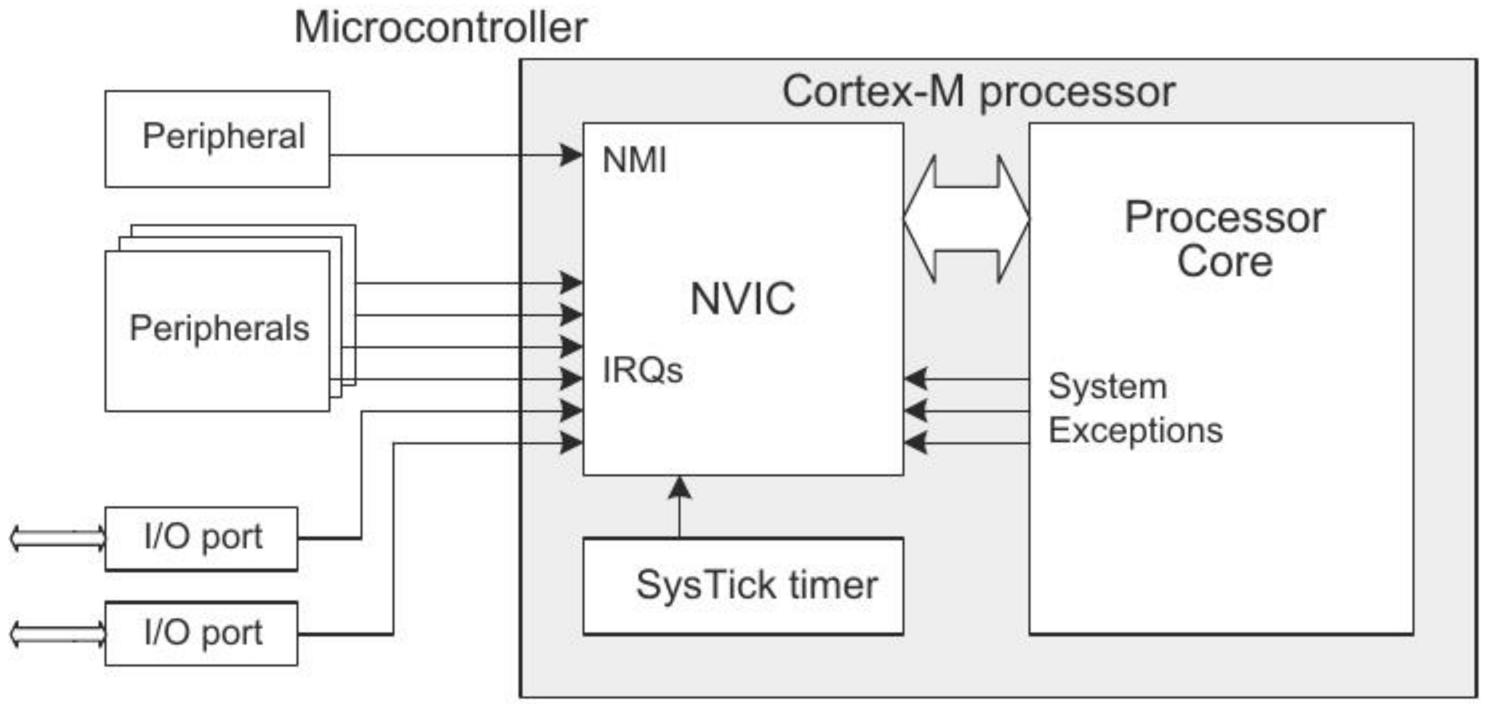
After knowing the Channel number, we have 2 options to fetch the conversion result from. Either we can fetch it from ADGDR or from ADDRx of the corresponding channel.

```
unsigned long t = LPC_ADC->ADGDR
int currentResult = (t>>4) & 0xFFF; //Extract Conversion Result
```

Ref:
http://www.ocfreaks.com/lpc1768-adc-programming-tutorial/

Nested Vector Interrupt Control (NVIC)

- In a microcontroller, interrupts serve as a way to immediately divert the central processing unit (CPU) from its current task to another, more important task.
- An interrupt can be triggered internally from the microcontroller (MCU) or externally, by a peripheral.
- The interrupt alerts the central processing unit (CPU) to an occurrence such as a time-based event, or the start or end of a process.



Interrupts can be triggered internally – from a timer,– or externally, from peripherals.

What happens when interrupt occurs

- When an interrupt occurs, an interrupt signal is generated.
- This causes the CPU to stop its current operation, save its current state, and begin the processing program — referred to as an interrupt service routine (ISR) or interrupt handler — associated with the interrupt.
- When the interrupt processing is complete, the CPU restores its previous state and resumes where it left off.

- Nested vector interrupt control (NVIC) ➔ prioritizes interrupts, thereby improves the MCU’s performance and reducing interrupt latency.
- NVIC also provides implementation schemes for handling interrupts that occur when other interrupts are being executed or when the CPU is in the process of restoring its previous state and resuming its suspended process.
- The term “nested” refers to the fact that in NVIC, a number of interrupts (up to several hundreds) can be defined, and each interrupt is assigned a priority, with “0” being the highest priority. In addition, the most critical interrupt can be made non-maskable, meaning it cannot be disabled (masked).

- NVIC is to ensure that higher priority interrupts are completed before lower-priority interrupts, even if the lower-priority interrupt is triggered first.
- For example, if a lower-priority interrupt is being registered or executed and a higher-priority interrupt occurs, the CPU will stop the lower-priority interrupt and process the higher-priority one first.

- The term “vector” in NVIC refers to the way in which the CPU ISR, to be executed.
- NVIC uses a vector table that contains the addresses of the ISRs for each interrupt.
- When an interrupt is triggered, the CPU gets the address from the vector table.
- The prioritization and handling schemes of nested vector interrupt control reduce the latency and overhead that interrupts typically introduce and ensure low power consumption, even with high interrupt loading on the controller.

- The Cortex-M3 processor uses a re-locatable vector table that contains the address of the function to be executed for a particular interrupt handler.
- On accepting an interrupt, the processor fetches the address from the vector table through the instruction bus interface.
- The vector table is located at address zero at reset, But can be relocated by programming a control register.

Ref: https://www.motioncontroltips.com/what-is-nested-vector-interrupt-control-nvic/