Module - 5

INTERRUPT PROGRAMMING

INTERRUPT PROGRAMMING:

Nested Vectored Interrupt Controller (NVIC), external hardware interrupts, timer/counter interrupts, ADC and DAC interrupts, interrupt programming.

Text 3: Refer Ch 2, Ch 4, Ch 5
Ref: UM10360, LPC 176x/5x User Manual

Nested vector interrupt control (NVIC)

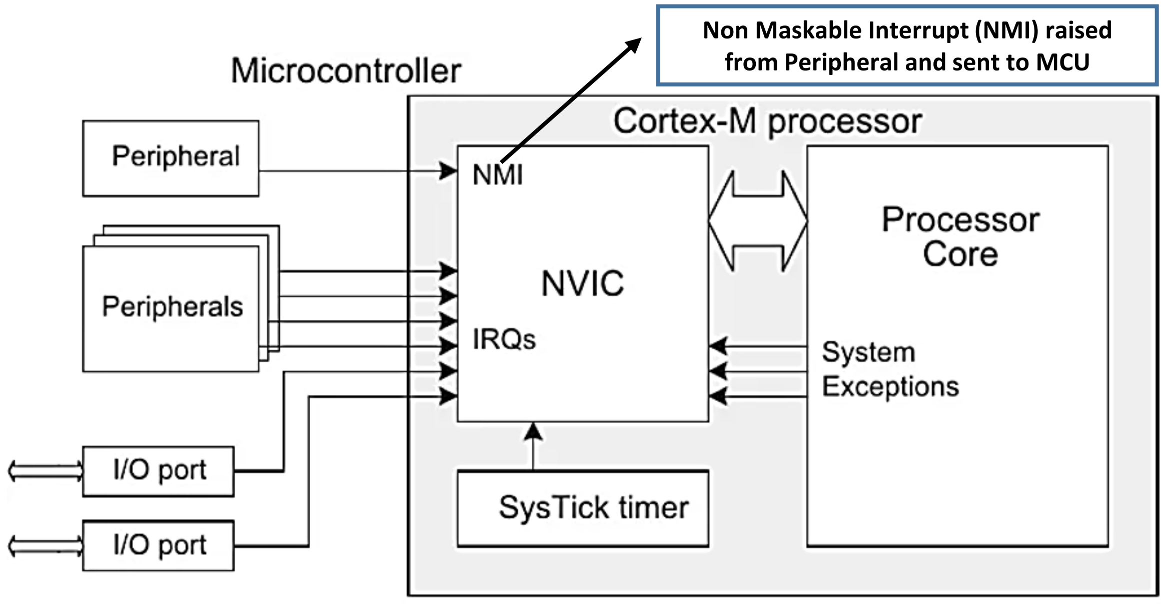
- **Nested Vector Interrupt Controller (NVIC)** is a core component of the ARM Cortex-M microcontroller architecture used for handling interrupts.
- An interrupt is an event that stops the execution of the main program and transfers the control to a specified function, called an interrupt handler, to perform a specific task.
- The NVIC manages the priority and handling of interrupts in the Cortex-M processor.
- The NVIC uses a priority-based interrupt system where each interrupt source is assigned a priority level, and the highest priority interrupt is serviced first.
- The NVIC also supports nested interrupts, which allows higher priority interrupts to preempt lower priority interrupts.
- When an interrupt occurs, the NVIC saves the current state of the interrupted program and jumps to the interrupt handler. After the interrupt handler completes its task, the NVIC restores the saved state of the interrupted program and continues its execution.

The NVIC has a set of registers that control the interrupt handling process, including

- Interrupt Set-Enable Register (ISER)
- Interrupt Clear-Enable Register (ICER)
- Interrupt Set-Pending Register (ISPR), and
- Interrupt Clear-Pending Register (ICPR)

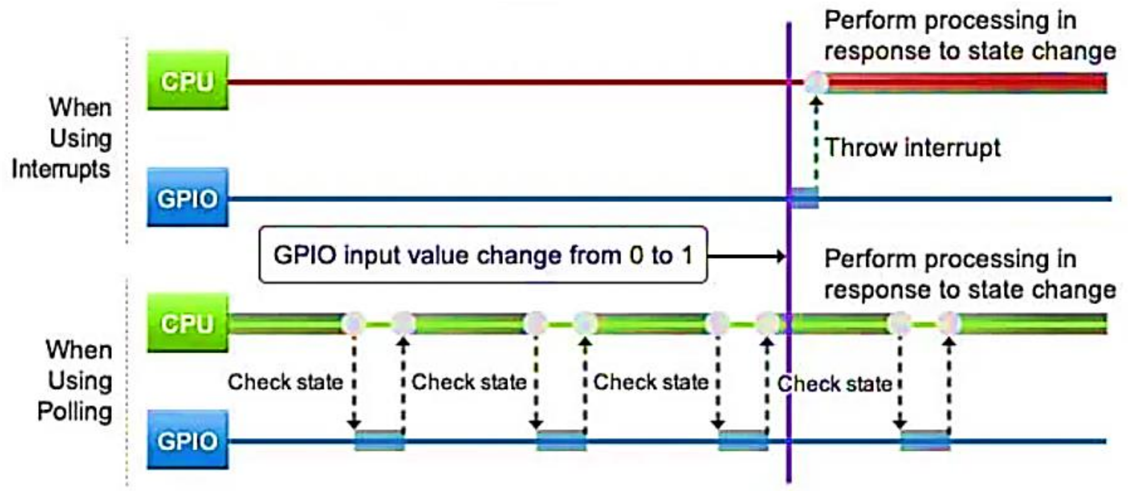
These registers allow the programmer to enable and disable interrupts, set and clear pending interrupts, and control interrupt priorities.

- **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 (a specified amount of time has elapsed or a specific time is reached, for example), a change of state, or the start or end of a process.



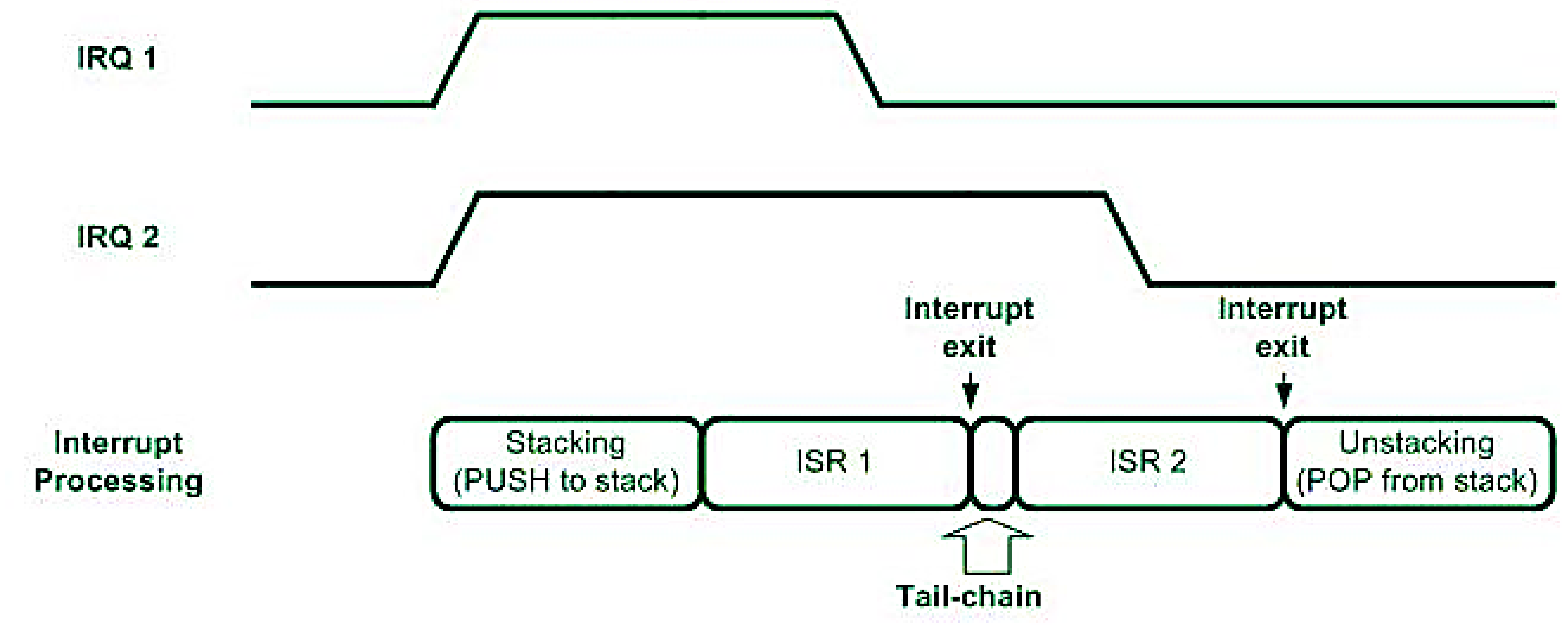
Interrupts can be triggered internally – from a timer, for example – or externally, from peripherals.

Another method of monitoring a timed event or change of state is referred to as "polling." With polling, the status of a timer or state change is periodically checked. The downsides of polling are the risk of excessive latency (delay) between the actual change and its detection, the possibility of missing a change altogether, and the increased processing time and power it requires.


Perform processing in response to state change
When Using Interrupts
CPU
Throw interrupt
GPIO
GPIO input value change from 0 to 1
Perform processing in response to state change
CPU
When Using Polling
Check state
Check state
Check state
Check state
GPIO

With polling, the CPU periodically checks for a change of state. Interrupts automatically notify the CPU when a change of state occurs, ensuring the change is not missed and its detection is not delayed.

- When an interrupt occurs, an interrupt signal is generated, which 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.
- The term “nested” refers to the fact that in NVIC, a number of interrupts (up to several hundred in some processors) 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).



With tail-chaining, if an interrupt is pending as the ISR for another, higher-priority interrupt completes, the processor will immediately begin the ISR for the next interrupt, without restoring its previous state.

- The term “vector” in nested vector interrupt control refers to the way in which the CPU finds the program, or ISR, to be executed when an interrupt occurs. Nested vector interrupt control uses a vector table that contains the addresses of the ISRs for each interrupt. When an interrupt is triggered, the processor 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.

External Hardware Interrupts

- Hardware external interrupts are interrupts generated by external hardware signals, such as buttons, switches, sensors, or other devices, that are connected to specific pins of a microcontroller or processor.
- These interrupts are triggered by a change in the state of the external signal, such as a rising or falling edge of a pulse, or a change in the voltage level of the signal.
- When a hardware external interrupt is triggered, the microcontroller stops executing the main program and jumps to a pre-defined interrupt service routine (ISR) or interrupt handler, which is a specific function in the program designed to handle the interrupt.
- The ISR typically performs a specific task or operation, such as reading sensor data or responding to user inputs, and then returns control to the main program.

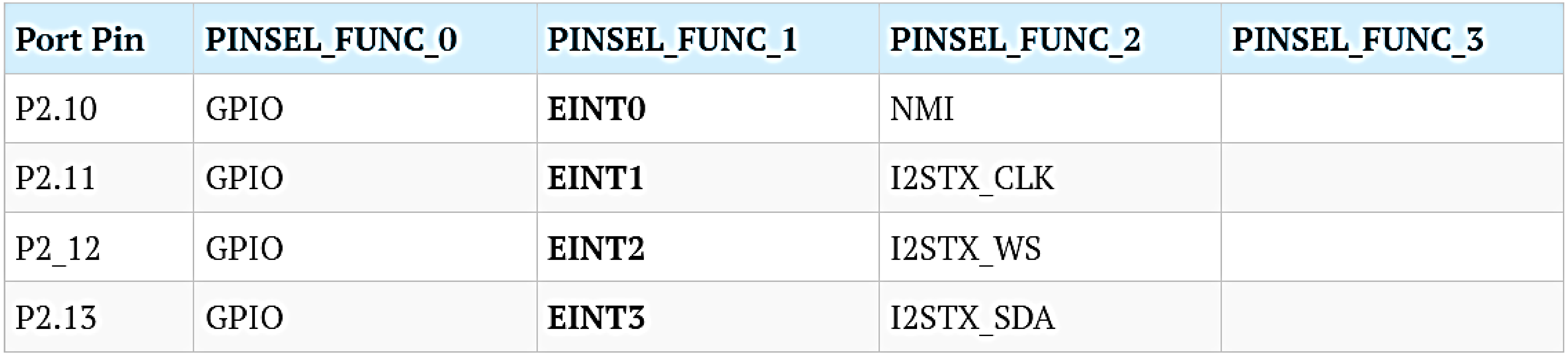
- Hardware external interrupts are often used in microcontroller-based systems to handle real-time events or time-critical operations, such as user inputs, sensor readings, or communication with external devices.
- These interrupts can be enabled or disabled by configuring specific registers in the microcontroller or processor, and can be prioritized based on their level of importance.
- Hardware external interrupts have several advantages over software interrupts, such as faster response times, lower CPU usage, and greater reliability.
- By using hardware external interrupts, developers can design more responsive and efficient systems that can quickly react to changing conditions or user inputs.

EINTx Pins

LPC1768 has four external interrupts EINT0-EINT3.

As LPC1768 pins are multi functional, these four interrupts are available on multiple pins.

Below table shows mapping of EINTx pins.



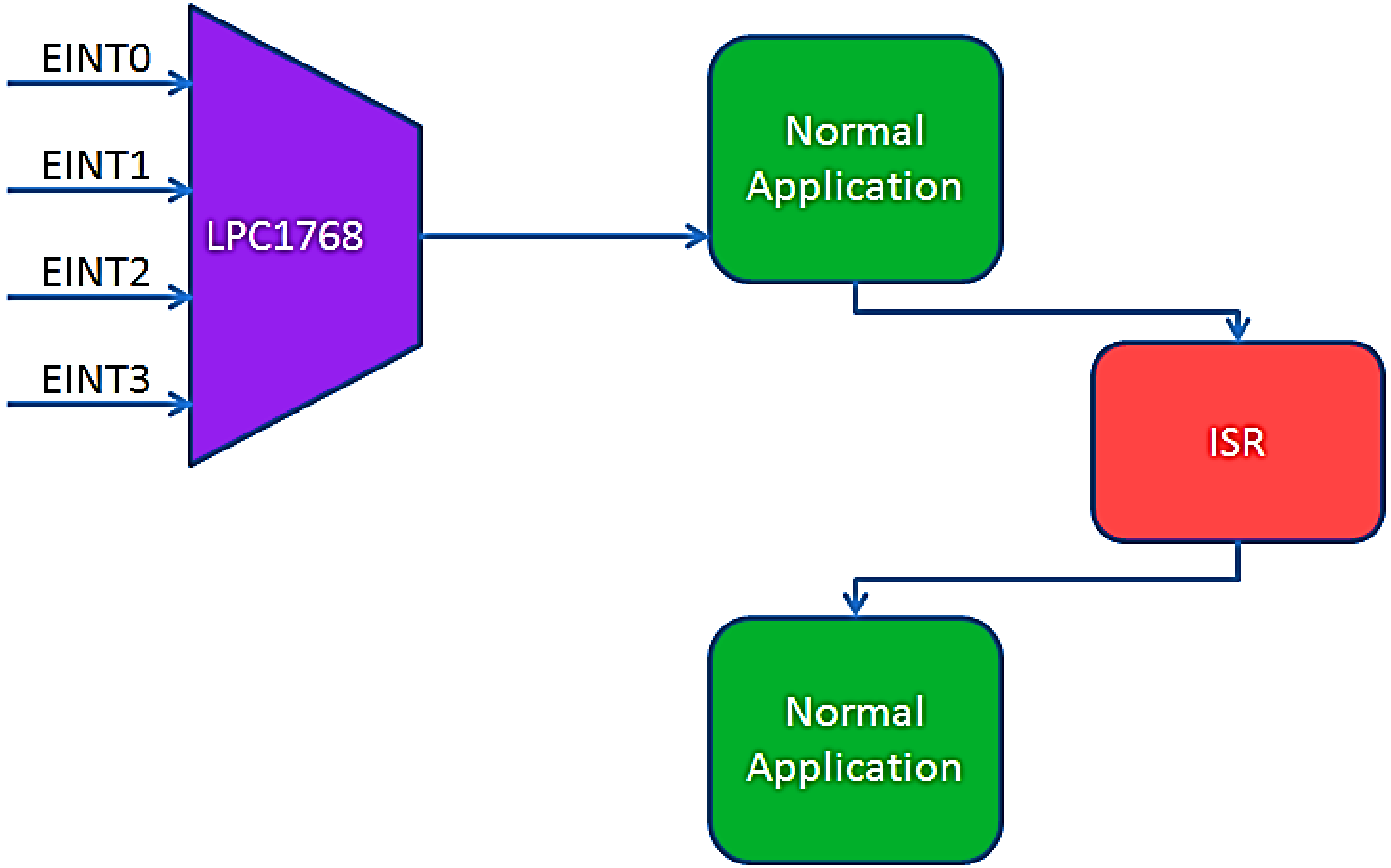
Port Pin	PINSEL_FUNC_0	PINSEL_FUNC_1	PINSEL_FUNC_2	PINSEL_FUNC_3
P2.10	GPIO	**EINT0**	NMI	
P2.11	GPIO	**EINT1**	I2STX_CLK	
P2_12	GPIO	**EINT2**	I2STX_WS	
P2.13	GPIO	**EINT3**	I2STX_SDA	

I2STX_CLK — Transmit Clock. It is driven by the master and received by the slave.

I2STX_WS — Transmit Word Select. It is driven by the master and received by the slave.

I2STX_SDA — Transmit data. It is driven by the transmitter and read by the receiver.

LPC1768 External Interrupts



EINT Registers

Register	Description
PINSELx	To configure the pins as External Interrupts
EXTINT	External Interrupt Flag Register contains interrupt flags for EINT0,EINT1, EINT2 & EINT3.
EXTMODE	External Interrupt Mode register(Level/Edge Triggered)
EXTPOLAR	External Interrupt Polarity(Falling/Rising Edge, Active Low/High)

EXTINT				
31:4	3	2	1	0
RESERVED	EINT3	EINT2	EINT1	EINT0

EINTx: Bits will be set whenever the interrupt is detected on the particular interrupt pin.
If the interrupts are enabled then the control goes to ISR.
Writing one to specific bit will clear the corresponding interrupt.

EXTMODE				
31:4	3	2	1	0
RESERVED	EXTMODE3	EXTMODE2	EXTMODE1	EXTMODE0

EXTMODEx: This bits is used to select whether the EINTx pin is level or edge Triggered
0: EINTx is Level Triggered.
1: EINTx is Edge Triggered.

EXTPOLAR				
31:4	3	2	1	0
RESERVED	EXTPOLAR3	EXTPOLAR2	EXTPOLAR1	EXTPOLAR0

EXTPOLARx: This bits is used to select polarity(LOW/HIGH, FALLING/RISING) of the EINTx interrupt depending on the EXTMODE register.
0: EINTx is Active Low or Falling Edge (depending on EXTMODEx).
1: EINTx is Active High or Rising Edge (depending on EXTMODEx).

Steps to Configure Interrupts

1. **Configure the pins as external interrupts in PINSELx register.**
2. **Clear any pending interrupts in EXTINT.**
3. **Configure the EINTx as Edge/Level triggered in EXTMODE register.**
4. **Select the polarity(Falling/Rising Edge, Active Low/High) of the interrupt in EXTPOLAR register.**
5. **Finally enable the interrputs by calling NVIC_EnableIRQ() with IRQ number.**

External Interrupt Example Programming

```
#include<LPC17xx.h>
void EINT3_IRQHandler(void);
unsigned char int3_flag=0;
int main(void)
{
          LPC_PINCON->PINSEL4 |= 0x04000000;                    //P2.13 as EINT3
          LPC_PINCON->PINSEL4 &= 0xFCFFFFFF;                    //P2.12 GPIO for LED
          LPC_GPIO2->FIODIR = 0x00001000;                       //P2.12 is assigned output
          LPC_GPIO2->FIOSET = 0x00001000;              //Initiall LED is kept on

          LPC_SC->EXTINT = 0x00000008;                          //writing 1 cleares the interrupt, get set if there is interrupt
          LPC_SC->EXTMODE = 0x00000008;                         //EINT3 is initiated as edge senitive, 0 for level sensitive
          LPC_SC->EXTPOLAR = 0x00000000;                        //EINT3 is falling edge sensitive, 1 for rising edge

                                                       //above registers, bit0-EINT0, bit1-EINT1, bit2-EINT2,bit3-EINT3
          NVIC_EnableIRQ(EINT3_IRQn);                           //core_cm3.h

          while(1) ;
}

void EINT3_IRQHandler(void)
{
                    LPC_SC->EXTINT = 0x00000008;                //cleares the interrupt

                    if(int3_flag == 0x00)              //when flag is '0' off the LED
                    {
                              LPC_GPIO2->FIOCLR = 0x00001000;
                              int3_flag = 0xff;
                    }
                    else
                                                                //when flag is FF on the LED
                    {
                              LPC_GPIO2->FIOSET = 0x00001000;
                              int3_flag = 0;
                    }
}
```

Timer/counter interrupts

- Timer/counter interrupts are a type of interrupt used in microcontrollers and processors to perform time-critical operations or measure time intervals.
- Timer/counter interrupts are generated by hardware timers and counters that are integrated into the microcontroller or processor.
- The hardware timer or counter is a device that generates a periodic signal, called a clock signal, that is used to measure time intervals or count events.
- The timer or counter is configured with a specific time interval or event count, and when the timer or counter reaches the set value, it generates a timer/counter interrupt.
- The timer/counter interrupt signals the microcontroller or processor to stop the main program and execute a specific interrupt service routine (ISR) or interrupt handler.
- Timer/counter interrupts are commonly used in microcontroller-based systems for time-critical operations, such as controlling the frequency of a PWM signal, generating accurate delays, or measuring the period or frequency of an external signal.

- The timer/counter interrupts provide a precise and accurate mechanism for handling time-sensitive operations, and they are more reliable and efficient than software-based timers.
- To use timer/counter interrupts in a microcontroller-based system, the hardware timer or counter needs to be configured with the appropriate settings, such as the time interval or event count, and the interrupt enable register needs to be set to enable the timer/counter interrupt.
- The ISR associated with the timer/counter interrupt is typically a specific function in the program that performs the desired operation, such as updating the PWM duty cycle or measuring the frequency of an external signal.

ADC and DAC interrupts

- ADC (Analog-to-Digital Converter) and DAC (Digital-to-Analog Converter) interrupts are a type of interrupt used in microcontrollers and processors to handle analog input and output operations.
- ADC interrupts are generated when an analog input signal is converted to a digital signal by the ADC, and DAC interrupts are generated when a digital signal is converted to an analog output signal by the DAC.
- ADC interrupts are commonly used in microcontroller-based systems to convert analog signals from sensors, such as temperature or pressure sensors, into digital signals that can be processed by the microcontroller.
- When an ADC interrupt is generated, the microcontroller stops executing the main program and jumps to a specific interrupt service routine (ISR) or interrupt handler that processes the converted digital signal.
- DAC interrupts are commonly used in microcontroller-based systems to generate analog output signals for controlling analog devices, such as motors or speakers.
- When a DAC interrupt is generated, the microcontroller stops executing the main program and jumps to a specific ISR or interrupt handler that processes the digital signal and generates the analog output signal.

- To use ADC or DAC interrupts in a microcontroller-based system, the ADC or DAC needs to be configured with the appropriate settings, such as the sampling rate, resolution, or output voltage range, and the interrupt enable register needs to be set to enable the ADC or DAC interrupt.
- The ISR associated with the ADC or DAC interrupt is typically a specific function in the program that performs the desired operation, such as processing the converted digital signal or generating the analog output signal.

Interrupt programming

Examples:

- ADC and
- External Interrupts