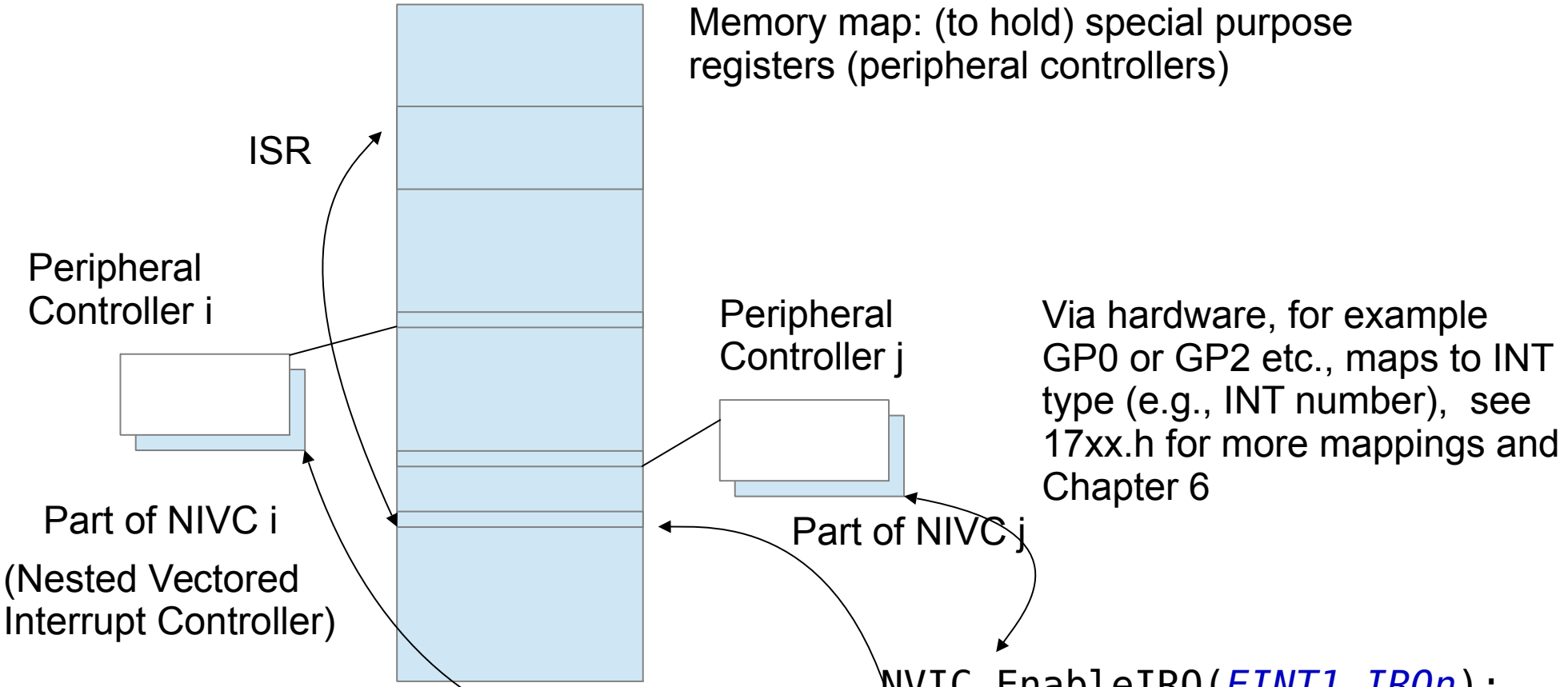
Definition of Interrupt Technique

An interrupt technique is a technique that demands CPU ***immediate*** attention, upon an interrupt request, CPU will have to ***finish*** the execution of the current instructions and then ***preserve*** the intermediate computation result by pushing the content of the general purpose registers into a stack; then jump to the interrupt service routine (ISR) to execute the ISR. Once finish the execution of the ISR, CPU will have to ***retrieve*** the information by popping up the content from the stack and ***resume*** the interrupted program.

The Big Picture of LPC INT Implementation




```
STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F));
}
```

INT Special Purpose Registers for Init

(1) function prototype: `uint32_t EINTInit( void )` the type is uint32_t, unsigned integer, right click on it to check its definition, this bring you to stdint.h header file, so you can see *typedef* unsigned *int uint32_t;*

(2) Special purpose registers (6):

PINSEL4;

PINMODE4;

IO2IntEnR; IO2IntEnF;

EXTMODE; EXTPOLAR;

```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;                              // for making pull-
up use 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11));                    /* Port2.10 is
rising edge. */
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<11));
 LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE ;      /* INT1 edge trigger */
 LPC_SC->EXTPOLAR |= 0;              /* INT0 is falling edge by default */
 NVIC_EnableIRQ(EINT1_IRQn);
 return 0;
}
```

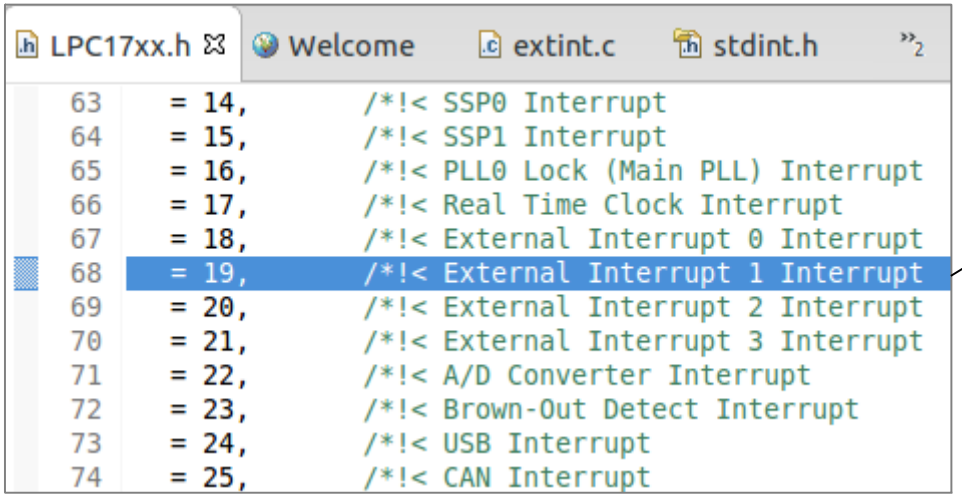
Interrupt Number Linked to ISR

Interrupt Service Routine (ISR): *NVIC_EnableIRQ(EINT1_IRQn);*
(1) where is EINT1_IRQn is declared? LPC17xx.h;
Mouse over on it (fig 1) then click on open declaration, pop-up window shows EINT1_IRQn at /CMSIS_CORE_LPC17xx/inc/LPC17xx.h

(2) Check its declaration details, click on its item on the pop-up window, see its declaration, (fig 2)

```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;                                  // for making pull-up use 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11));                   /* Port2.10 is rising edge. */
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<11));
 LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE ;      /* INT1 edge trigger */
 LPC_SC->EXTPOLAR |= 0;                  /* INT0 is falling edge by default */
 NVIC_EnableIRQ(EINT1_IRQn);
 return 0;
}
```

figure 1




```
EINT1_IRQn = 19, /*!< External Interrupt 1 Interrupt */
```

figure 2

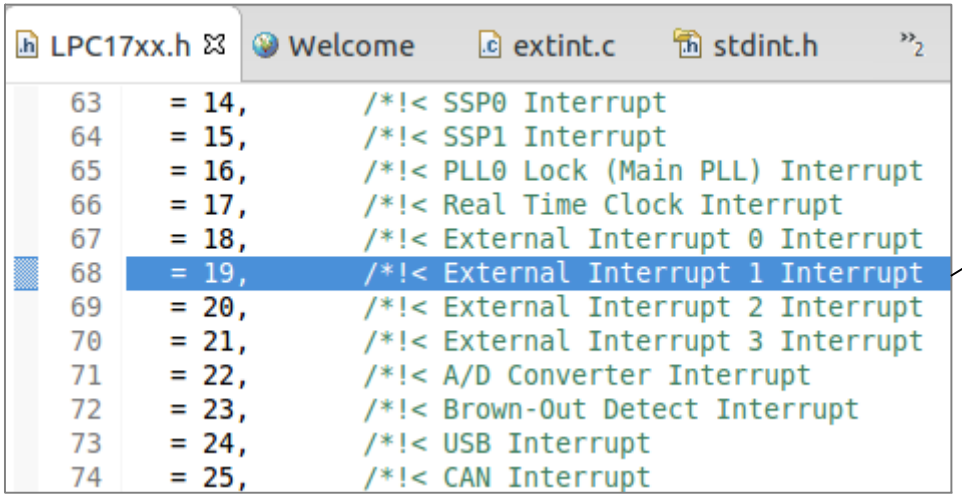
Interrupt Number Linked to ISR

Interrupt Service Routine (ISR): *NVIC_EnableIRQ(EINT1_IRQn);*
(1) where is EINT1_IRQn is declared? LPC17xx.h;
Mouse over on it (fig 1) then click on open declaration, pop-up window shows EINT1_IRQn at /CMSIS_CORE_LPC17xx/inc/LPC17xx.h

(2) Check its declaration details, click on its item on the pop-up window, see its declaration, (fig 2)

```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;                                  // for making pull-up use 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11));                   /* Port2.10 is rising edge. */
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<11));
 LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE ;      /* INT1 edge trigger */
 LPC_SC->EXTPOLAR |= 0;                  /* INT0 is falling edge by default */
 NVIC_EnableIRQ(EINT1_IRQn);
 return 0;
}
```

figure 1




```
EINT1_IRQn = 19, /*!< External Interrupt 1 Interrupt */
```

figure 2

Interrupt Set Enable Register to ISR

Interrupt Set Enable Register is is located via interrupt number (type), e.g., *NVIC_EnableIRQ(EINT1_IRQn) (defined* in core_cms3.h see the sample code below) which is in turn mapped to interrupt table in the memory map, at the corresponding location of this table holds the pointer pointing to *Service Routine (ISR)*

(1)enable device specific interrupt;

(2) the interrupt controller is *NVIC;*

```
/** \brief  Enable External Interrupt

    The function enables a device-specific interrupt in the NVIC interrupt controller.

    \param [in]      IRQn  External interrupt number. Value cannot be negative.
 */
__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F)); /* enable interrupt */
}
```

figure 1

Utilization of Int Example

After initialization, now use interrupt. Sample code touch button switch to turn on/off LED based on interrupt is given below.

```
void EINT1_IRQHandler (void)
{

LPC_SC->EXTINT = EINT1;

LPC_GPIO0->FIODIR |= (1<<3);
if(LPC_GPIO2->FIOPIN &(1<<11))
{

key_count ++;  // key_count +1 when receive external interrupt
    delayMs(0,500);  //delay used as debouncer
}

if( key_count == (key_count1 + key_count2))
{
if(LPC_GPIO0->FIOPIN & (1<<3))
{
   LPC_GPIO0->FIOCLR |= (1<<3);  //turn off led if it was on
}
else
LPC_GPIO0->FIOSET |= (1<<3);    //turn on led if it was off
key_count1 += key_count2;
key_count2 ++;                 //increment the count and calculate the number of times needed to turn on and turn off led
                               //press button 1 time to turn on LED,2 times to turn off LED, 3 times to turn on LED
                               //4 times to turn off LED ...
}

LPC_GPIOINT->IO2IntEnR = ((0x01 <<11));
LPC_GPIOINT->IO2IntClr = 0xFFFFFFFF;
LPC_GPIOINT->IO0IntClr = 0xFFFFFFFF;

}
```

figure 1

Putting INIT and ISR Together

Putting INI (initialization), and user defined ISR together .

```
int main(void)
{

LPC_GPIO0->FIODIR |= (1<<3); //set pin 0.23 as
output
LPC_GPIO0->FIODIR &= ~(1<<11); //set pin 2.11
as input
LPC_GPIO0->FIOCLR |= (1<<3); //clear pin 0.23

while(1)
{
EINTInit(); //wait for external interrupt
}
```

1

```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as
EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;    //making pull-up 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11));    //Port2.10
rising edge
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<11));
 LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE;//INT1 edge
trigger
 LPC_SC->EXTPOLAR |= 0;        // INT0 is falling edge
by default
 NVIC_EnableIRQ(EINT1_IRQn);
 return 0;
}
```

2

3

```
void EINT1_IRQHandler (void)
{
LPC_SC->EXTINT = EINT1;
LPC_GPIO0->FIODIR |= (1<<3);
if(LPC_GPIO2->FIOPIN &(1<<11))
{
key_count ++;  // key_count +1 when receive
interrupt
    delayMs(0,500);  //delay used as
debouncer
}
if( key_count == (key_count1 + key_count2))
{
if(LPC_GPIO0->FIOPIN & (1<<3))
{
   LPC_GPIO0->FIOCLR |= (1<<3);  //turn off
led if on
}
else
LPC_GPIO0->FIOSET |= (1<<3);    //turn on
led if off
key_count1 += key_count2;
key_count2 ++;                  //increment
the count
}
LPC_GPIOINT->IO2IntEnR = ((0x01 <<11));
LPC_GPIOINT->IO2IntClr = 0xFFFFFFFF;
LPC_GPIOINT->IO0IntClr = 0xFFFFFFFF;
}
```