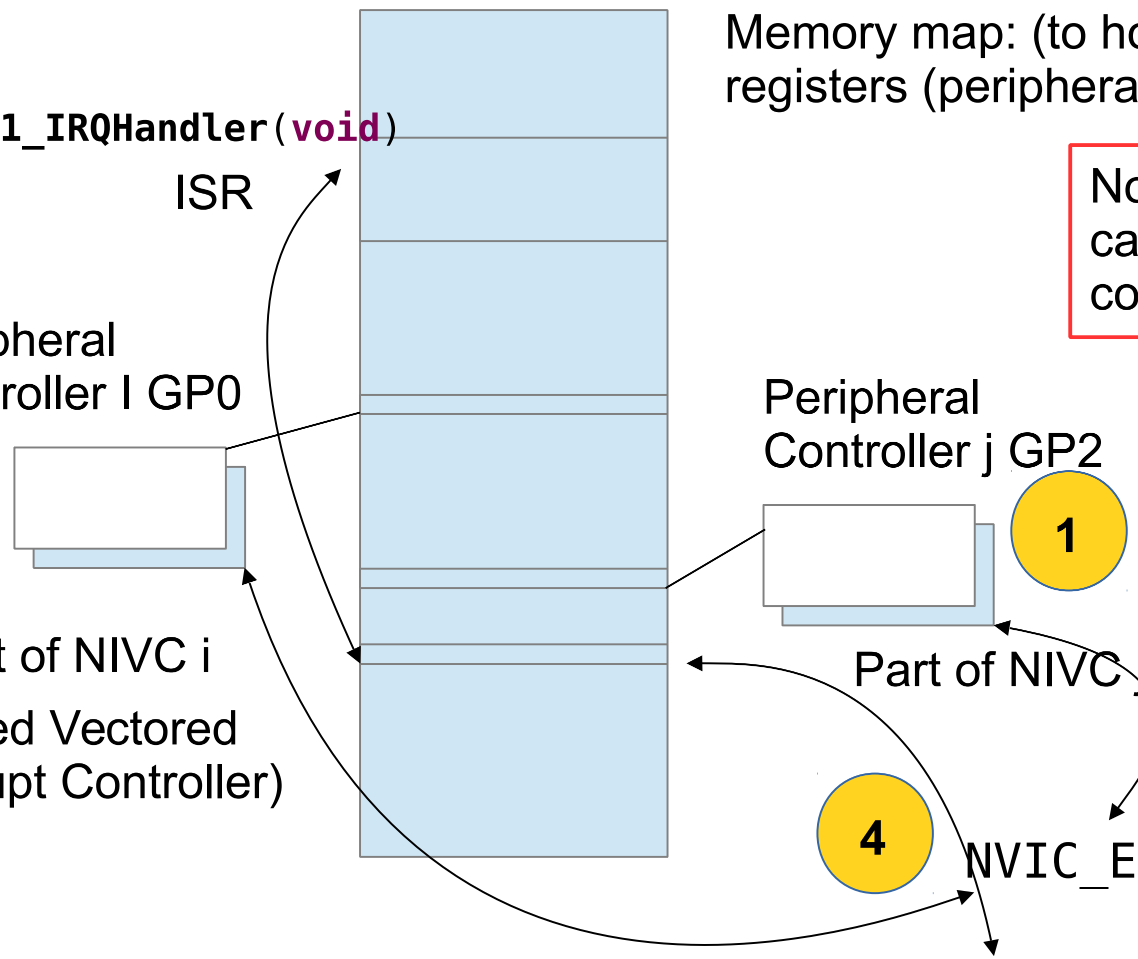
Definition of Interrupt Technique

An interrupt technique is a technique that demands CPU 1
immediate attention, upon an interrupt request, CPU will have
2 to ***finish*** the execution of the current instructions and then
3 ***preserve*** the intermediate computation result by pushing the
content of the general purpose registers into a stack; then
4 **jump** to the interrupt service routine (ISR) to execute the ISR.
Once finish the execution of the ISR, CPU will have to
5 ***retrieve*** the information by popping up the content from the
stack and ***resume*** the interrupted program.

The Big Picture of LPC INT Implementation



Memory map: (to hold) special purpose registers (peripheral controllers)

Note step 2 and 3, 5 are taken care of OS or CPU ExINT controller plus microprogramming

`void EINT1_IRQHandler(void)`

Peripheral Controller I GP0

Peripheral Controller j GP2

Via hardware, for example GP0 or GP2 etc., maps to INT type (e.g., INT number), see 17xx.h for more mappings and Chapter 6

Part of NIVC i

(Nested Vectored Interrupt Controller)

Part of NIVC j

Link to ISR `void EINT1_IRQHandler`

```
NVIC_EnableIRQ(EINT1_IRQn);
```

```
EINT1_IRQn = 19
```

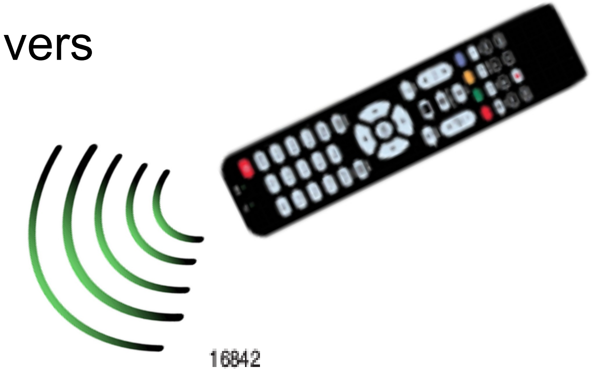
(Interrupt Set-Enable Register, in init & config)

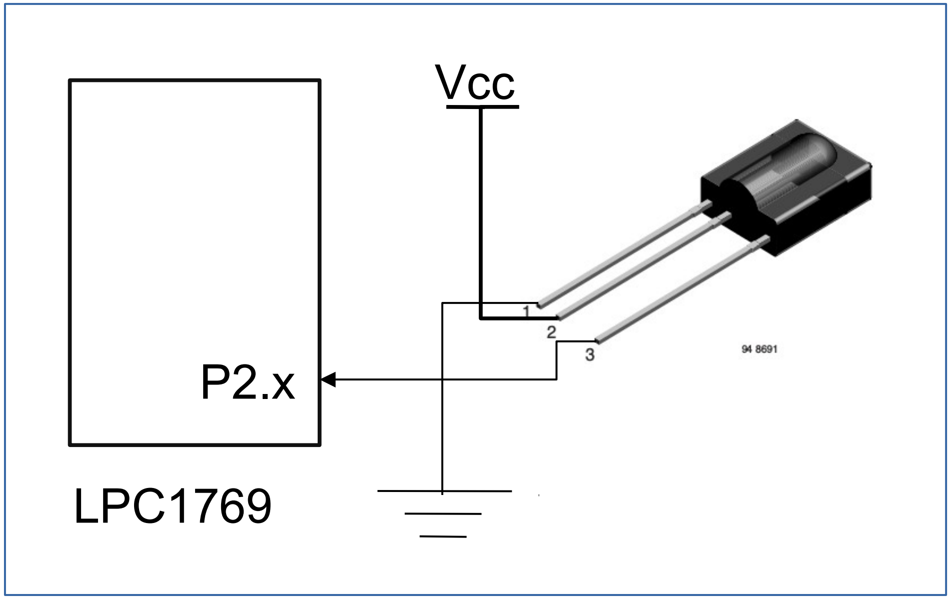
```
STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F));
}
```

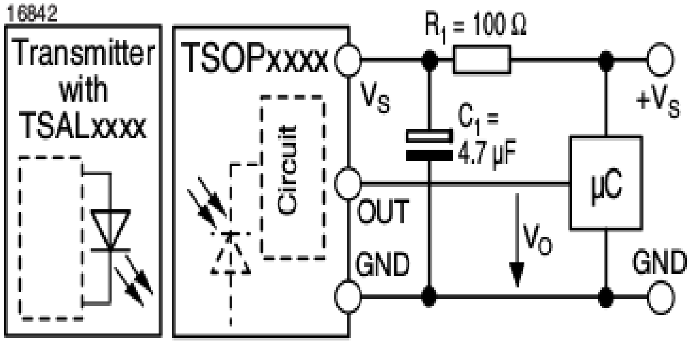
IrDA IC Triggers the ExINT

IR Receiver Modules for Remote Control Systems

The TSOP12xx - series are miniaturized receivers for infrared remote control systems.

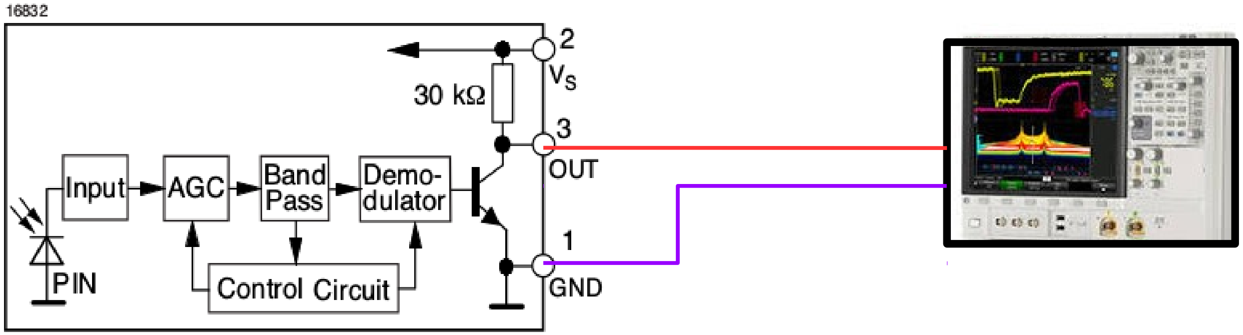






Reference: TSOP1236 datasheet

Example: Measure TSOP1236 response to the color remote controller's signal



(c) Harry Li, Ph.D.

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; Table 116. Interrupt Enable for port 2 Rising Edge (IO2IntEnR)

EXTMODE; EXTPOLAR; Table 11. External Interrupt Mode register (EXTMODE)

Table 12. External Interrupt Polarity register EXTPOLAR

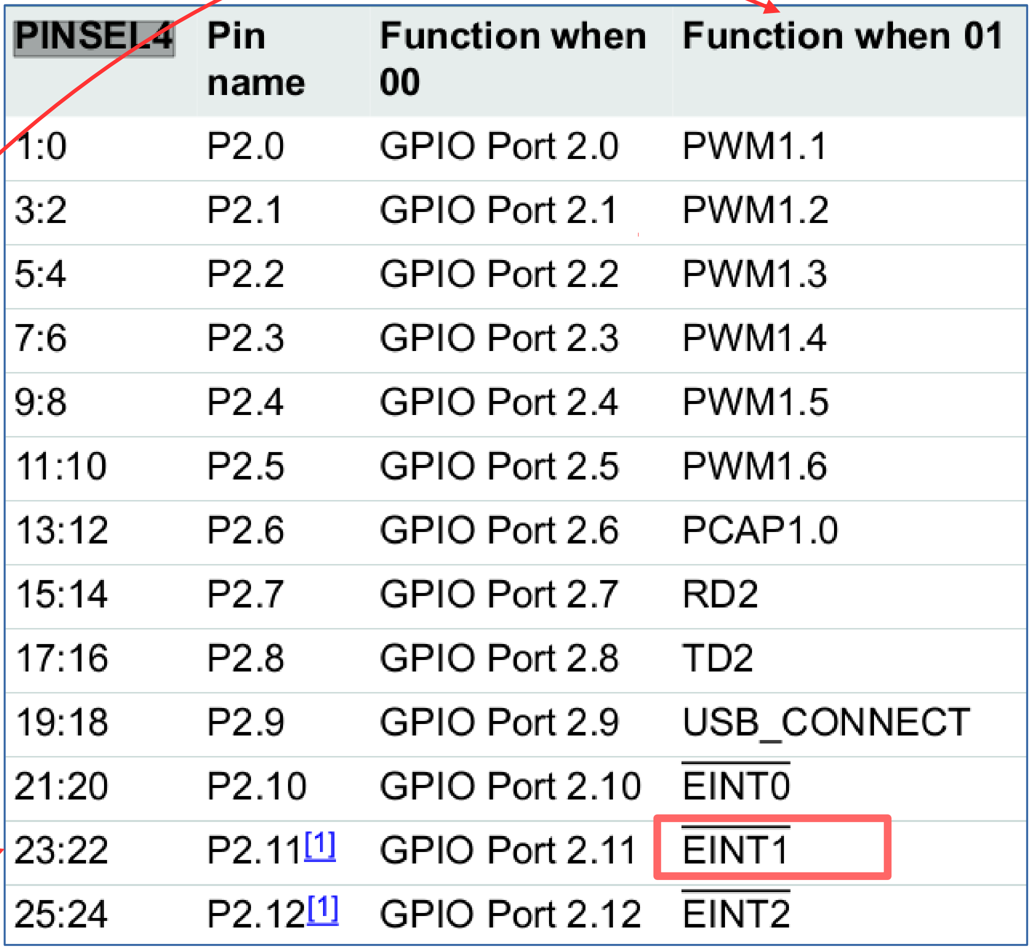
```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;            // for making pull-up use 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11)); /* Port2.11 is rising edge. */
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<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;
}
```

Pin Function Select Register PINSEL4

Example:

```
LPC_PINCON->PINSEL4 &= ~(3 << 22 );
//set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
```

Table 84: Pin function select register 4 (PINSEL4), pp 119.



PINSEL4	Pin name	Function when 00	Function when 01
1:0	P2.0	GPIO Port 2.0	PWM1.1
3:2	P2.1	GPIO Port 2.1	PWM1.2
5:4	P2.2	GPIO Port 2.2	PWM1.3
7:6	P2.3	GPIO Port 2.3	PWM1.4
9:8	P2.4	GPIO Port 2.4	PWM1.5
11:10	P2.5	GPIO Port 2.5	PWM1.6
13:12	P2.6	GPIO Port 2.6	PCAP1.0
15:14	P2.7	GPIO Port 2.7	RD2
17:16	P2.8	GPIO Port 2.8	TD2
19:18	P2.9	GPIO Port 2.9	USB_CONNECT
21:20	P2.10	GPIO Port 2.10	$\overline{\text{EINT0}}$
23:22	P2.11[1]	GPIO Port 2.11	$\overline{\text{EINT1}}$
25:24	P2.12[1]	GPIO Port 2.12	$\overline{\text{EINT2}}$

(c) Harry Li, Ph.D.

Pin Mode Select Register PINMODE

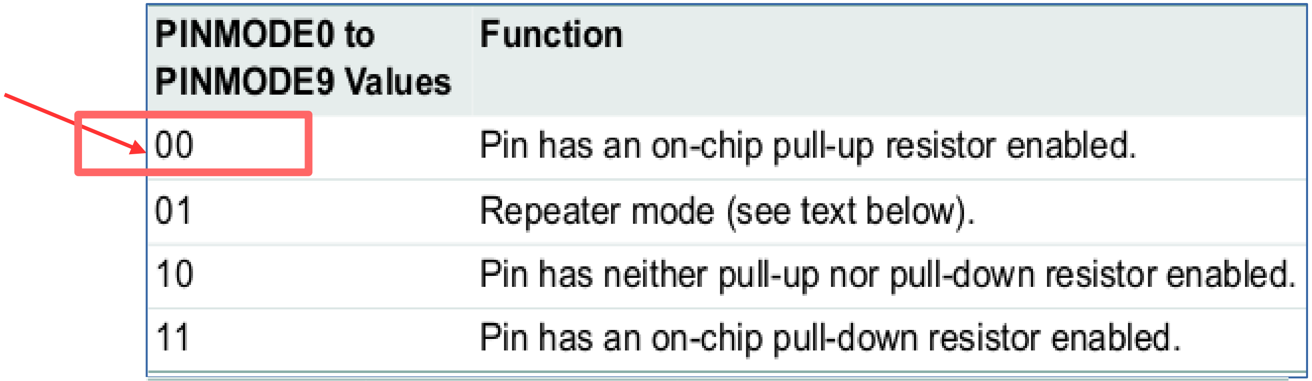
The PINMODE registers control the input mode of all ports. This includes the use of the on-chip pull-up/pull-down resistor feature and a special open drain mode. pp. 114

i, Ph.D.

Example:

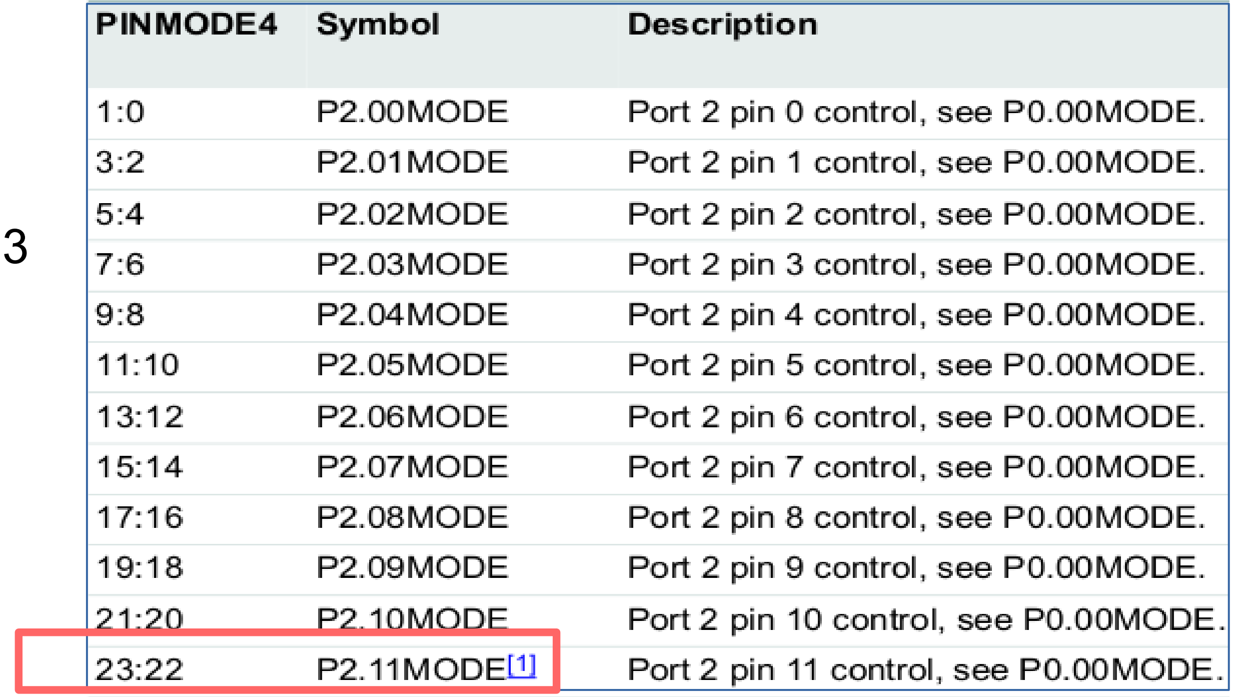
```
LPC_PINCON->PINMODE4 = 0;
```

Table 77. Pin Mode Select register Bits, pp. 114



PINMODE0 to PINMODE9 Values	Function
00	Pin has an on-chip pull-up resistor enabled.
01	Repeater mode (see text below).
10	Pin has neither pull-up nor pull-down resistor enabled.
11	Pin has an on-chip pull-down resistor enabled.

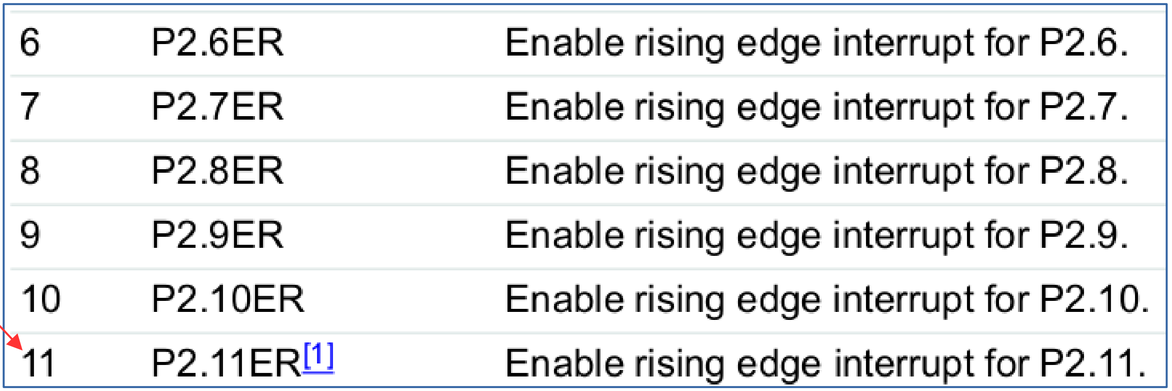
Table 92. Pin Mode select register 4: PINMODE4, pp. 123



PINMODE4	Symbol	Description
1:0	P2.00MODE	Port 2 pin 0 control, see P0.00MODE.
3:2	P2.01MODE	Port 2 pin 1 control, see P0.00MODE.
5:4	P2.02MODE	Port 2 pin 2 control, see P0.00MODE.
7:6	P2.03MODE	Port 2 pin 3 control, see P0.00MODE.
9:8	P2.04MODE	Port 2 pin 4 control, see P0.00MODE.
11:10	P2.05MODE	Port 2 pin 5 control, see P0.00MODE.
13:12	P2.06MODE	Port 2 pin 6 control, see P0.00MODE.
15:14	P2.07MODE	Port 2 pin 7 control, see P0.00MODE.
17:16	P2.08MODE	Port 2 pin 8 control, see P0.00MODE.
19:18	P2.09MODE	Port 2 pin 9 control, see P0.00MODE.
21:20	P2.10MODE	Port 2 pin 10 control, see P0.00MODE.
23:22	P2.11MODE[1]	Port 2 pin 11 control, see P0.00MODE.

Interrupt Enable for P2 Rising Edge IO2IntEnR

Table 116. GPIO Interrupt Enable for port 2 Rising Edge (IO2IntEnR), chapter 9, pp. 141

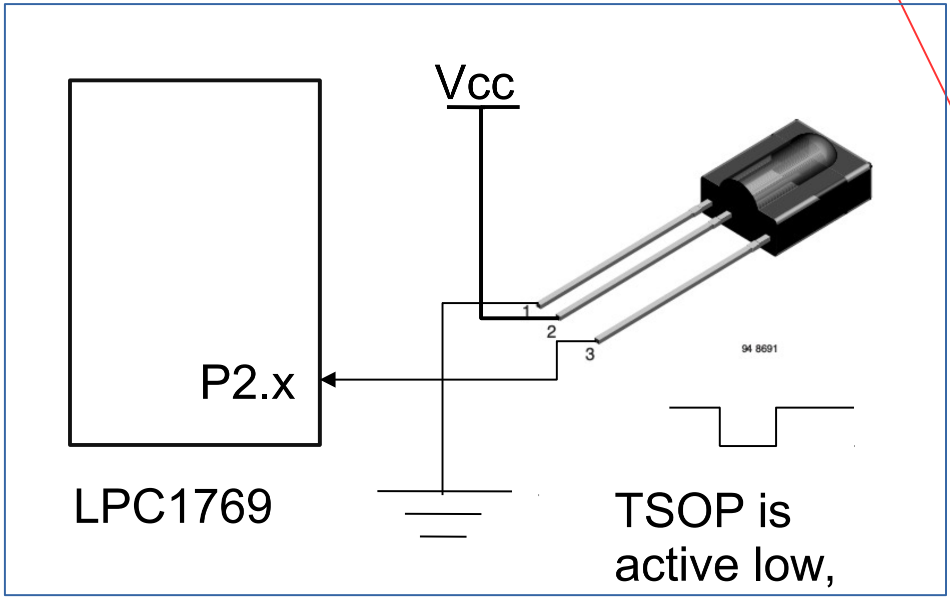


6	P2.6ER	Enable rising edge interrupt for P2.6.
7	P2.7ER	Enable rising edge interrupt for P2.7.
8	P2.8ER	Enable rising edge interrupt for P2.8.
9	P2.9ER	Enable rising edge interrupt for P2.9.
10	P2.10ER	Enable rising edge interrupt for P2.10.
11	P2.11ER[1]	Enable rising edge interrupt for P2.11.

Example:

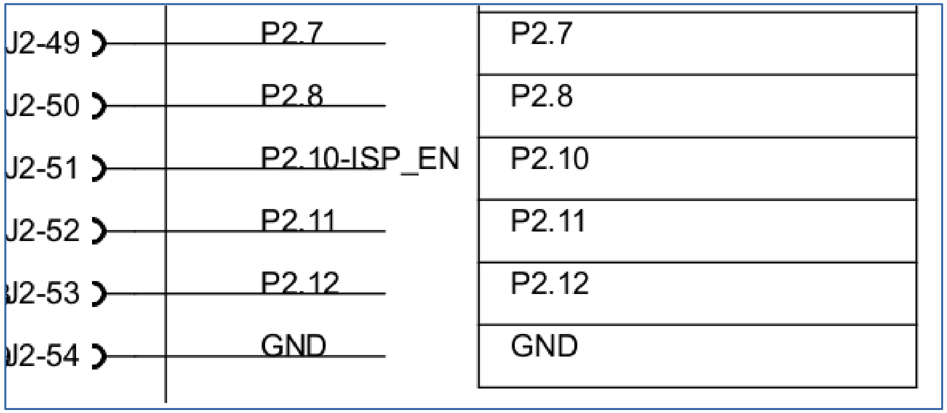
```
LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11));
```

For the falling edge interrupt, see 9.5.6.5 GPIO Interrupt Enable for port 2 Falling Edge (IO2IntEnF), pp. 143.



TSOP is active low, so use IO2IntEnF

Pin assignment for P2, from schematics of the LPC1769 module



(c) Harry Li, Ph.D.

EXTMODE: The External Interrupt Mode Register

The External Interrupt Mode Register controls whether each pin is edge- or level-sensitive. pp. 35 from CPU data sheet.

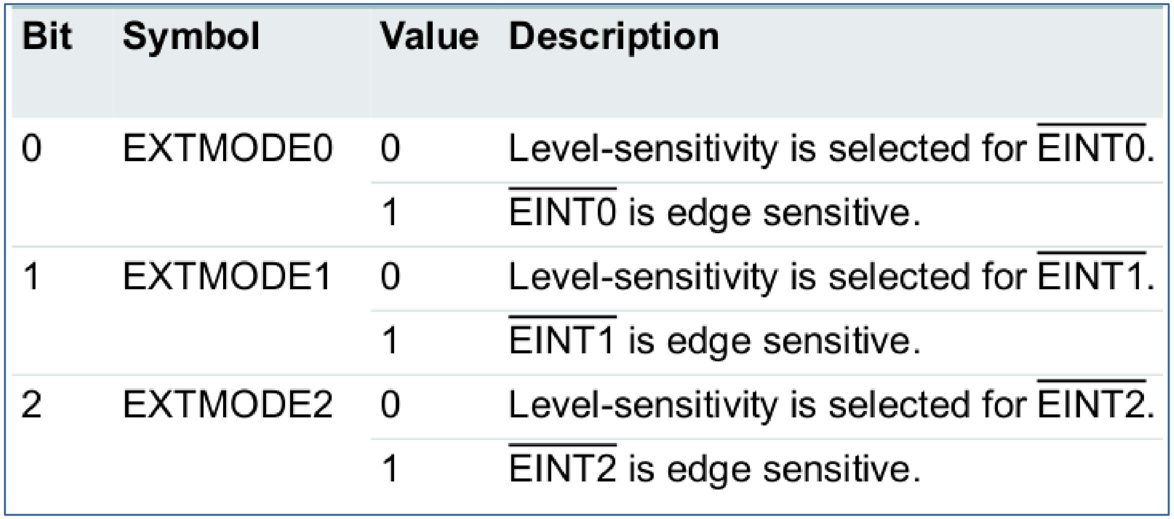
Example:

```
LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE
```

Note:

Edge triggered (sensitive) or level triggered (sensitive) detect the interrupt signal based on its level or its (raising or falling) edge. Edge triggered are more timing sensitive to capture the sudden change of the signal, however, random noise can easily mis-triggered the interrupt then level triggered interrupt.

Table 11. External Interrupt Mode register (EXTMODE)



Bit	Symbol	Value	Description
0	EXTMODE0	0	Level-sensitivity is selected for $\overline{\text{EINT0}}$.
		1	$\overline{\text{EINT0}}$ is edge sensitive.
1	EXTMODE1	0	Level-sensitivity is selected for $\overline{\text{EINT1}}$.
		1	$\overline{\text{EINT1}}$ is edge sensitive.
2	EXTMODE2	0	Level-sensitivity is selected for $\overline{\text{EINT2}}$.
		1	$\overline{\text{EINT2}}$ is edge sensitive.

External Interrupt Polarity Register EXTPOLAR

In level-sensitive mode, the bits in this register select whether the corresponding pin is high- or low-active. In edge-sensitive mode, they select whether the pin is rising- or falling-edge sensitive. pp.27 data sheet.

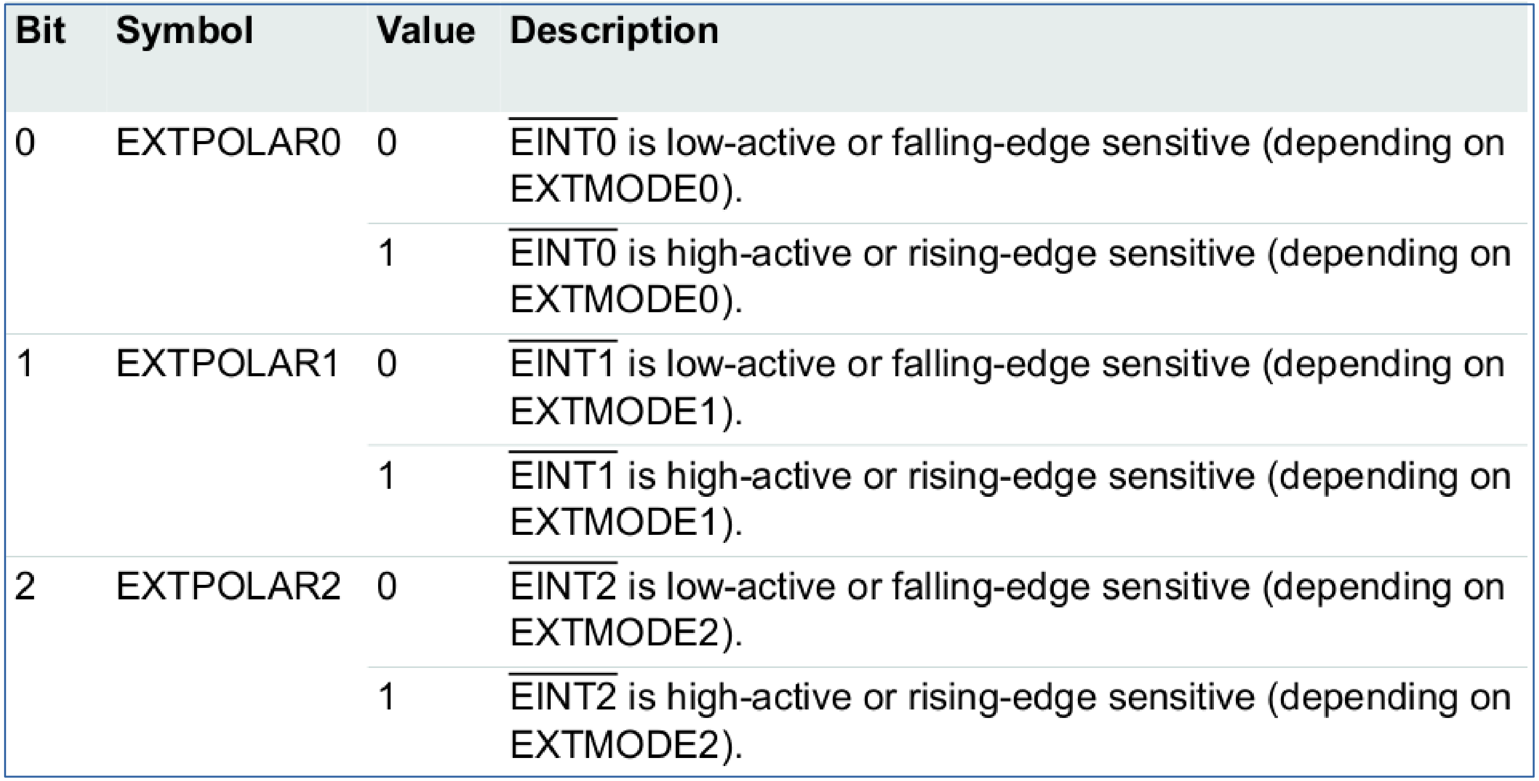
Example:

```
LPC_SC->EXTPOLAR |= 0;
```

Note the above code sets falling edge or level interrupt from NXP sample code, but the following could be better

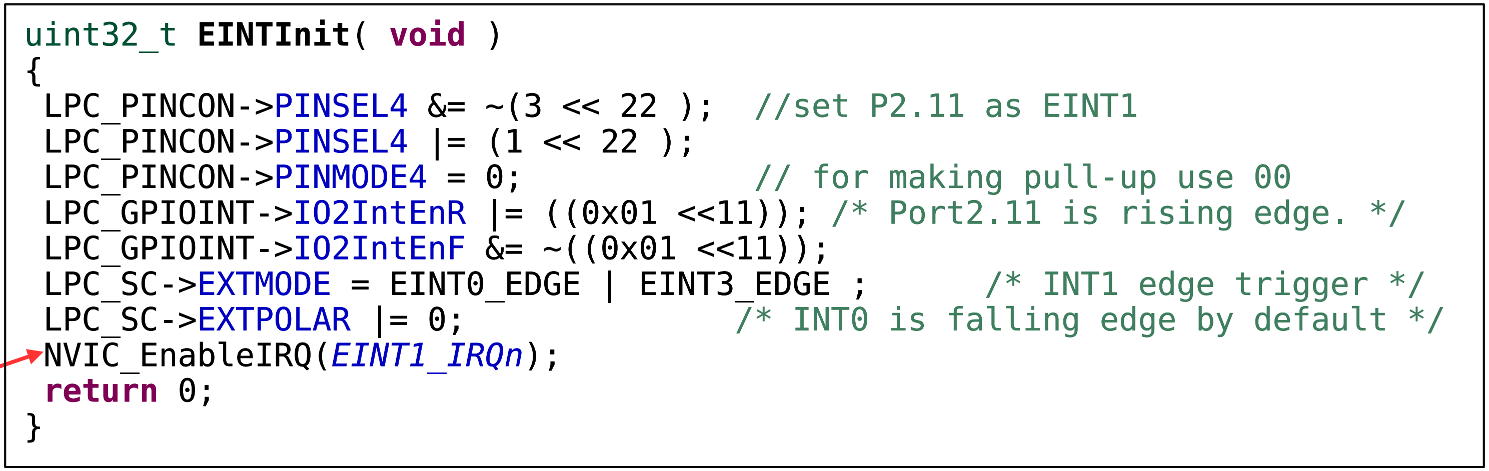
```
LPC_SC->EXTPOLAR &=
~(1<<2);
```

Table 12. External Interrupt Polarity register EXTPOLAR



Bit	Symbol	Value	Description
0	EXTPOLAR0	0	$\overline{\text{EINT0}}$ is low-active or falling-edge sensitive (depending on EXTMODE0).
		1	$\overline{\text{EINT0}}$ is high-active or rising-edge sensitive (depending on EXTMODE0).
1	EXTPOLAR1	0	$\overline{\text{EINT1}}$ is low-active or falling-edge sensitive (depending on EXTMODE1).
		1	$\overline{\text{EINT1}}$ is high-active or rising-edge sensitive (depending on EXTMODE1).
2	EXTPOLAR2	0	$\overline{\text{EINT2}}$ is low-active or falling-edge sensitive (depending on EXTMODE2).
		1	$\overline{\text{EINT2}}$ is high-active or rising-edge sensitive (depending on EXTMODE2).

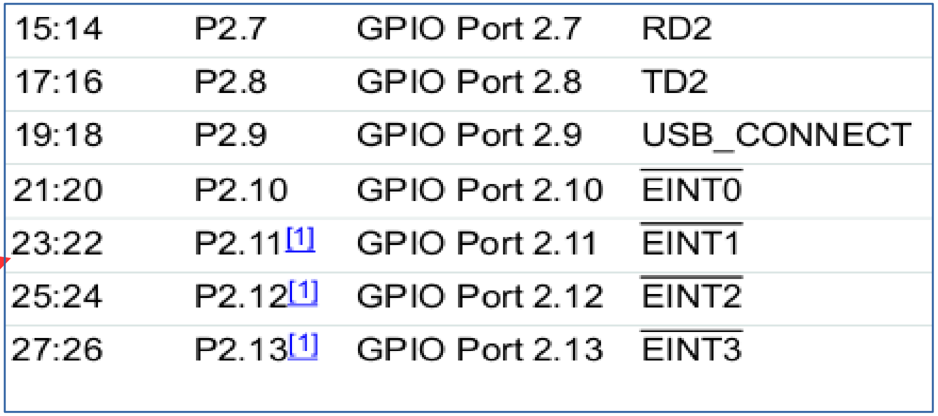
NVIC_EnableIRQ(*EINT1_IRQn*)



```
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)); /* Port2.11 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;
}
```

Need to use ExINT in the design, from CPU data sheet, the ExINT maps to GPP port 2, e.g., P2.x

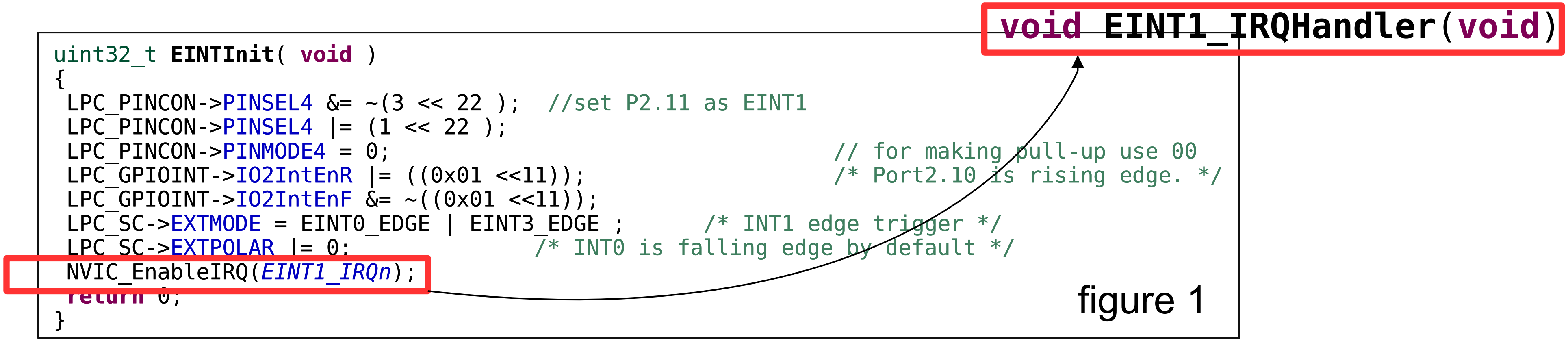
Then to find the pins of P2.x, from the CPU data sheet, table 84, Pin Function Select Register 4 (PINSEL4), pp. 119.



15:14	P2.7	GPIO Port 2.7	RD2
17:16	P2.8	GPIO Port 2.8	TD2
19:18	P2.9	GPIO Port 2.9	USB_CONNECT
21:20	P2.10	GPIO Port 2.10	$\overline{\text{EINT0}}$
23:22	P2.11[1]	GPIO Port 2.11	$\overline{\text{EINT1}}$
25:24	P2.12[1]	GPIO Port 2.12	$\overline{\text{EINT2}}$
27:26	P2.13[1]	GPIO Port 2.13	$\overline{\text{EINT3}}$

Interrupt Number Linked to ISR

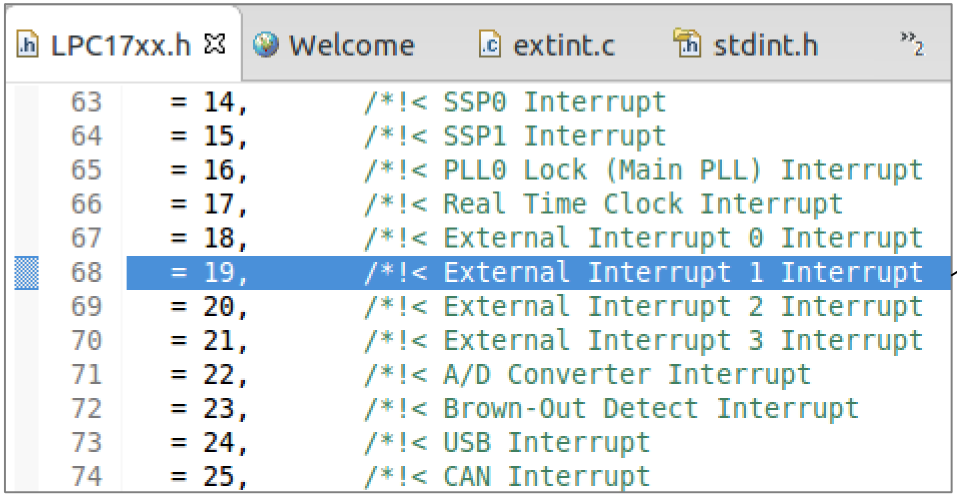
Interrupt Service Routine (ISR): *NVIC_EnableIRQ(EINT1_IRQn);*
(1) where is EINT1_IRQn is declared? LPC17xx.h;
Mouse over on it (fig 1) then click on open declaration, pop-up window shows EINT1_IRQn at /CMSIS_CORE_LPC17xx/inc/LPC17xx.h
Check its declaration details, click on its item on the pop-up window, see its declaration, (fig 2); (2) How is ISR connected?



```
void EINT1_IRQHandler(void)
```

```
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



```
        = 14,       /*!< SSP0 Interrupt
        = 15,       /*!< SSP1 Interrupt
        = 16,       /*!< PLL0 Lock (Main PLL) Interrupt
        = 17,       /*!< Real Time Clock Interrupt
        = 18,       /*!< External Interrupt 0 Interrupt
        = 19,       /*!< External Interrupt 1 Interrupt
        = 20,       /*!< External Interrupt 2 Interrupt
        = 21,       /*!< External Interrupt 3 Interrupt
        = 22,       /*!< A/D Converter Interrupt
        = 23,       /*!< Brown-Out Detect Interrupt
        = 24,       /*!< USB Interrupt
        = 25,       /*!< CAN Interrupt
```

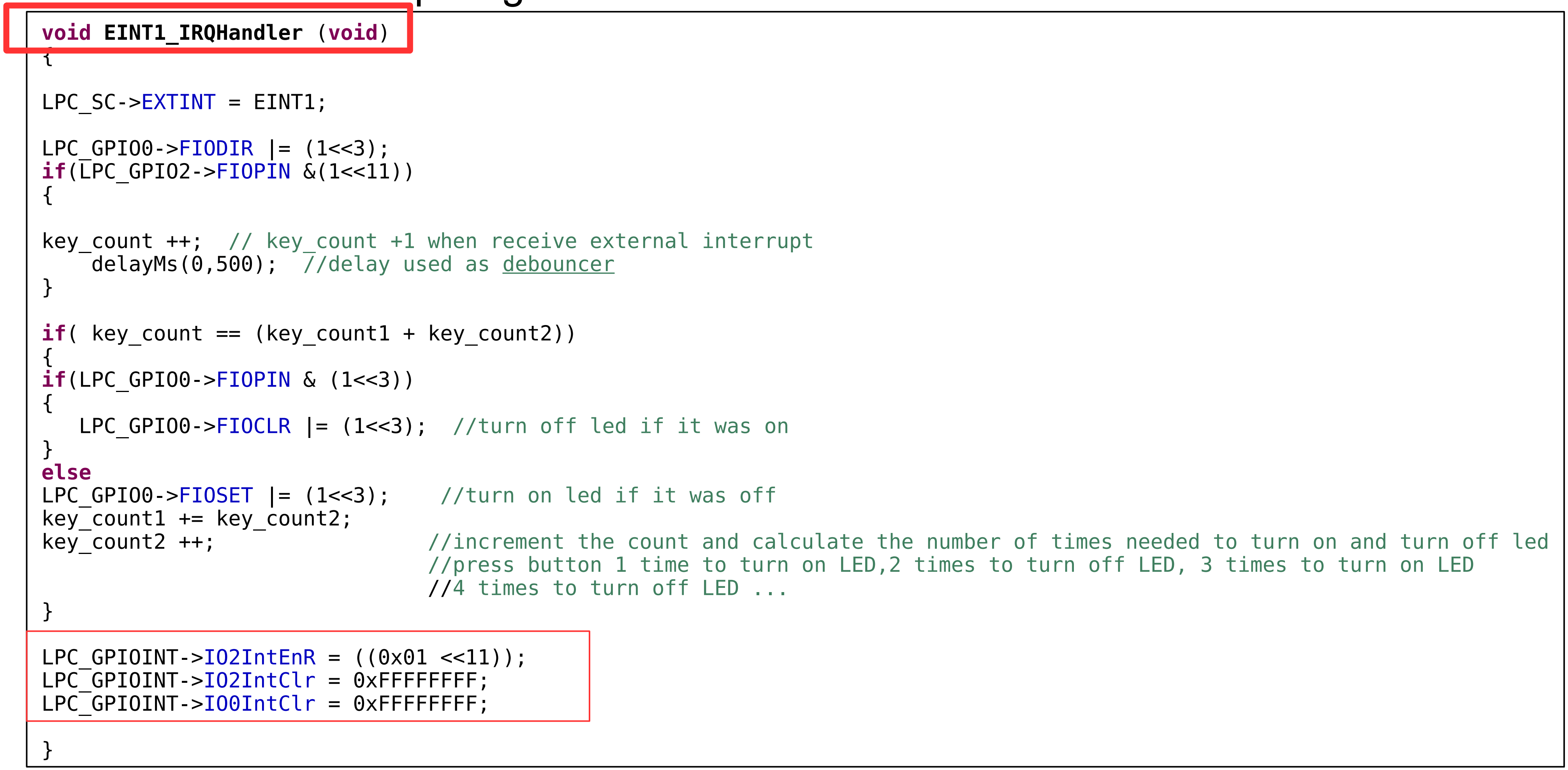
```
EINT1_IRQn = 19, /*!< External Interrupt 1 Interrupt */
```

figure 2

(c) Harry Li, Ph.D. SJSU, CMPE127

ISR 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;

}
```

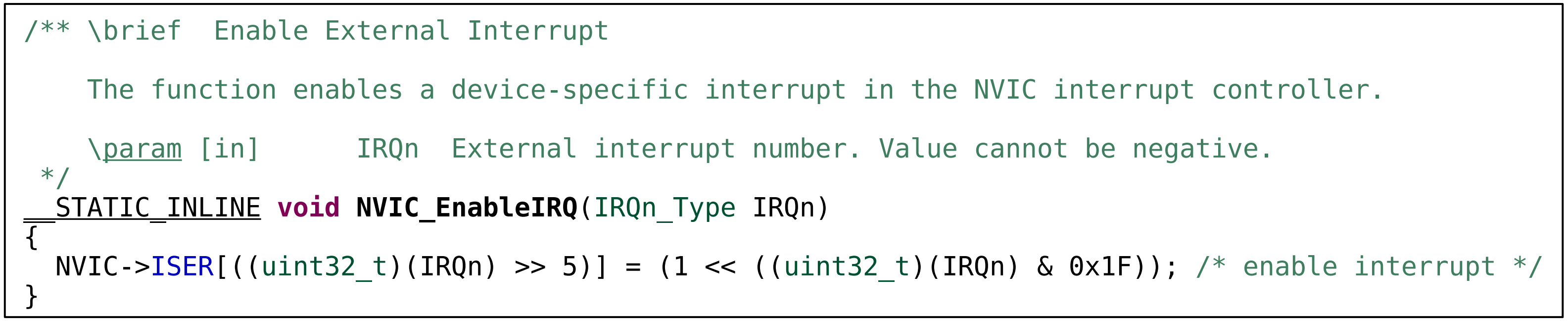
figure 1

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;*

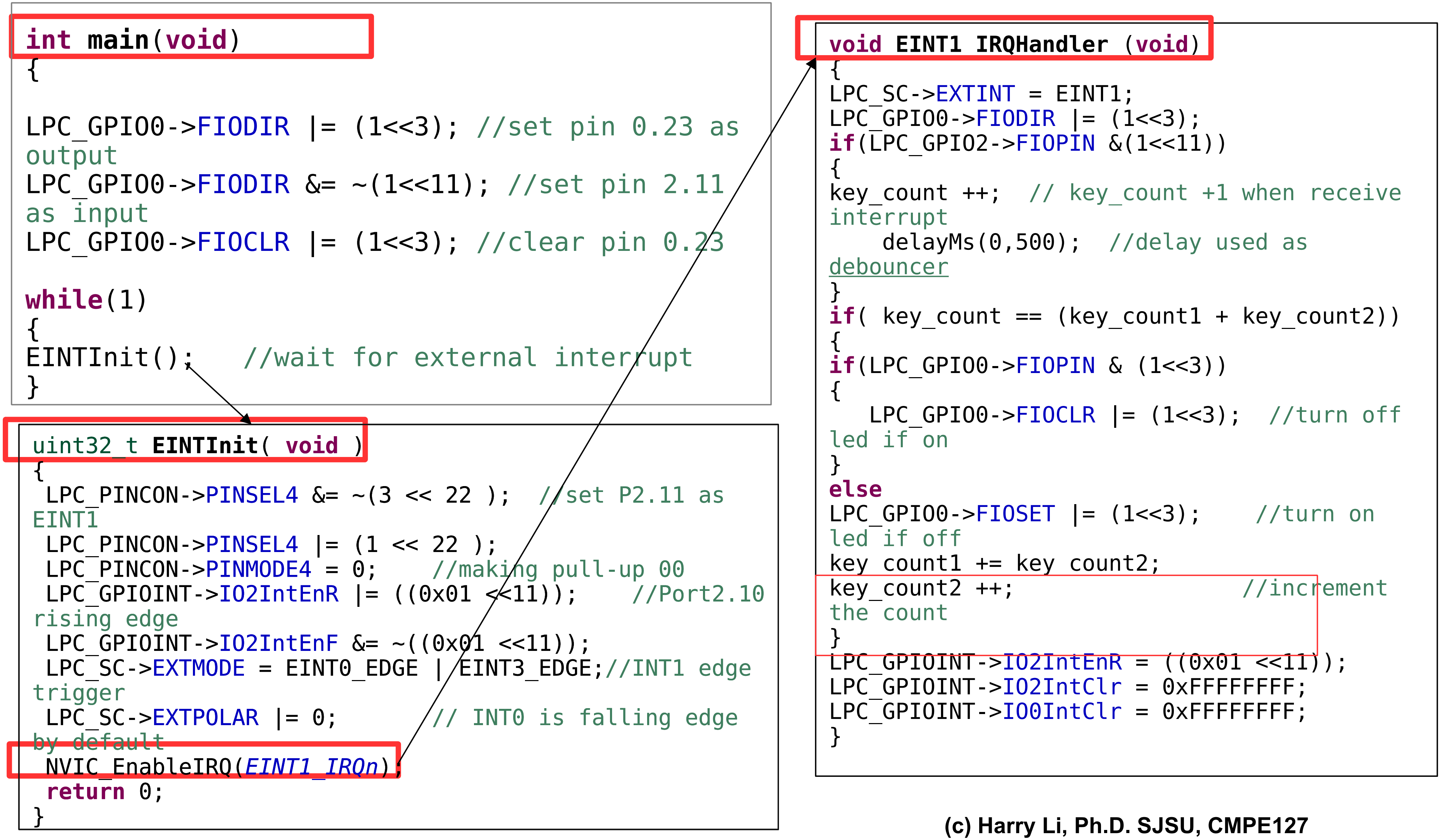


```
/** \brief  Enable External Interrupt

    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

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
}
```

```
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;
}
```

```
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;
}
```