Embedded Systems
3.3 - Interrupts (Clicker 2)

Enrico Simetti

ISME - Interuniversity Research Center on Integrated Systems for Marine Enviroment
DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering
University of Genova, Italy
enrico.simetti@unige.it

interrupt: reprogrammable pins, we must configure where to find the correct peripheral???

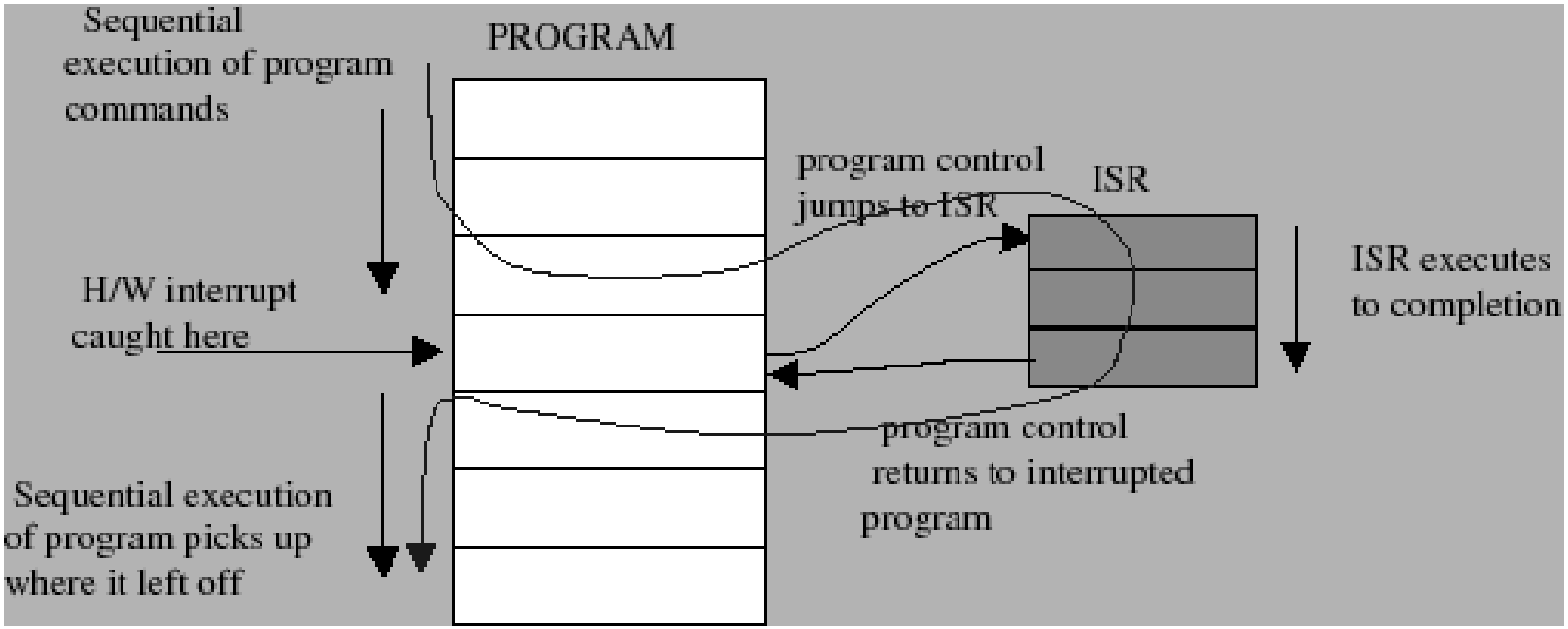
What is an interrupt I

Interrupt: an asynchronous signal from hardware indicating the need for attention (hardware interrupt) or a synchronous event in software indicating the need for a change in execution (software interrupt or software trap)

What is an interrupt II

- Synchronous and asynchronous with respect to what?
- To program execution!
- Synchronous means that the interrupt depends on the instruction we are currently executing (e.g. a divide by zero trap)
- Asynchronous means that it does have any temporal relation to which part of the code the microcontroller is executing (the timer expires independently of which point the program has currently reached)

What is an interrupt III

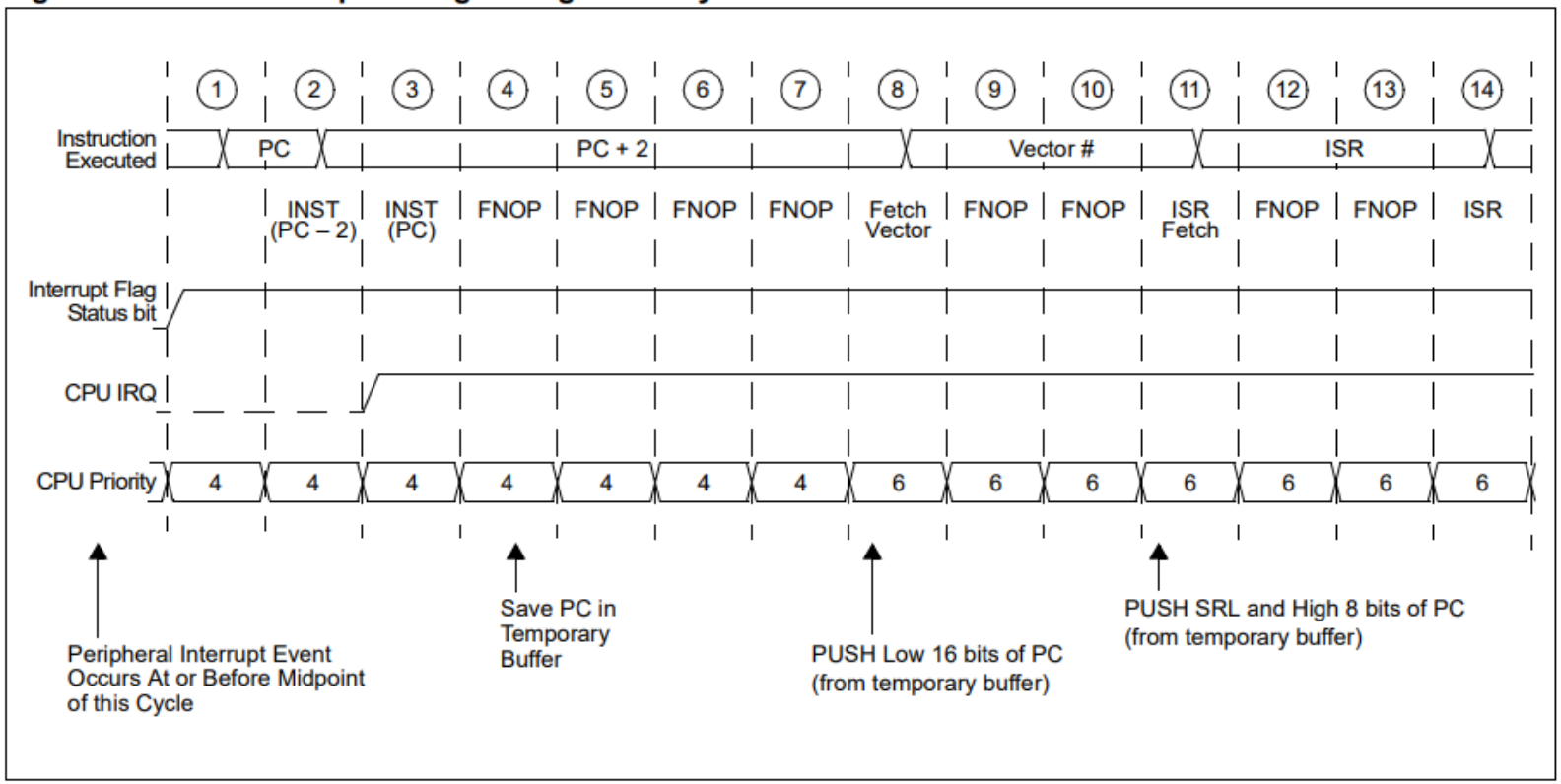


What is an interrupt IV

- All the peripherals have one or more interrupt flags which are set by hardware when a special event occurs
- All interrupt flags are sampled at the beginning of each instruction cycle
- If there is a pending request and if the corresponding interrupt generation is enabled an interrupt will be presented to the processor.
- The processor then saves the following information on the software stack (interrupt context switch):
  - the current PC value
  - the Processor Status register
- The CPU sets its priority level equal to the one of the pending interrupt
- The Interrupt Service Routine corresponding to the event is executed
- After the ISR, the previous context is recovered and the program execution continues

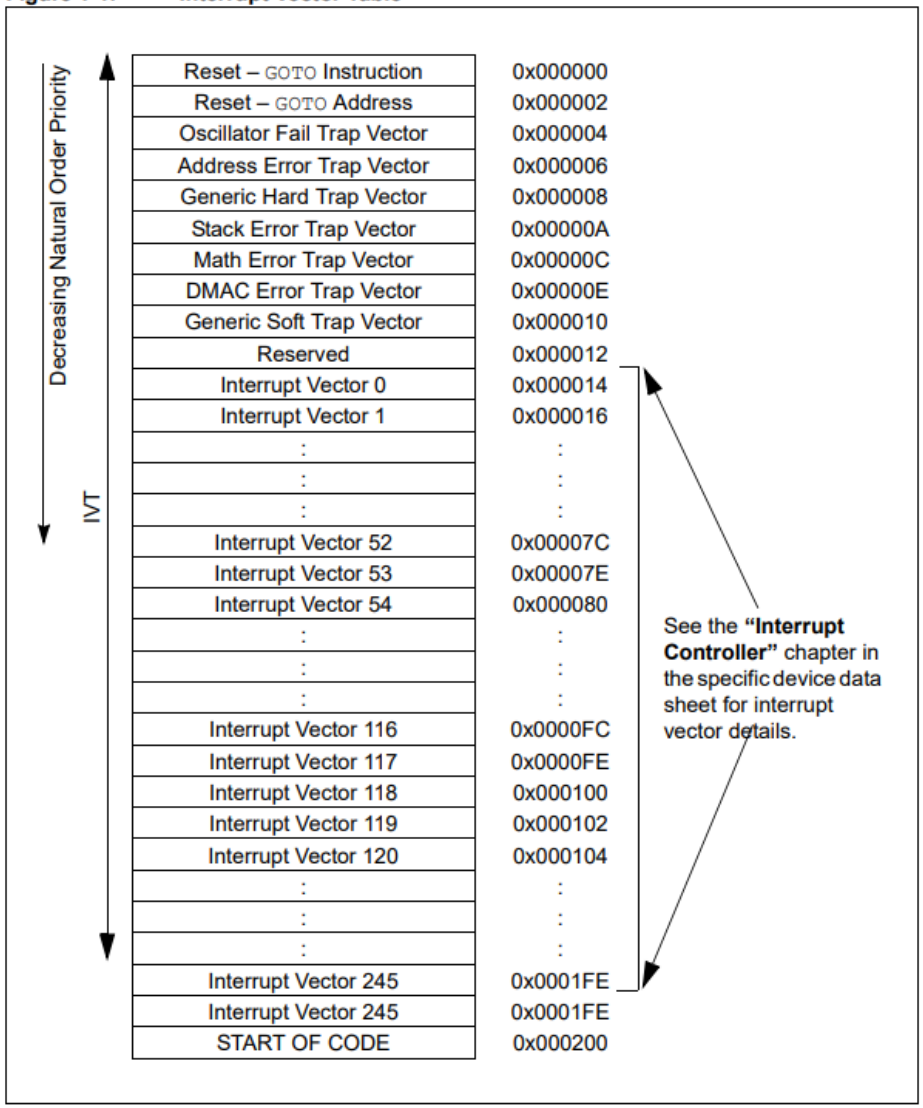
Interrupt Latency

Figure 3-1: Interrupt Timing During a One-Cycle Instruction



Interrupt Vector Table

Figure 1-1: Interrupt Vector Table



Interrupt Service Routine I

An ISR is like any other C function with some special rules:

- each ISR has a special name which is used for linking its code to the IVT
- no parameters can be passed to the ISR
- no parameters can be returned by an ISR (void return type is mandatory)
- the flag generating the interrupt should be generally cleared in the ISR
- ISR code should not contain blocking sections
- ISRs should not be called by main line code
- Care must be taken with shared data

```
void __attribute__((__interrupt__, __auto_psv__)) _INT0Interrupt() {
  IFS0bits.INT0IF = 0; // reset interrupt flag

  [...] // do some action
}
```

Interrupt Service Routine II

ISR names for dsPIC33 can be found in the dsPIC C compiler document, Chapter 8, page 117

Table 6-1: Interrupt Vector Details

IRQ #	IVT Address	AIVT Address	Interrupt Source
Highest Natural Order Priority			
0	0x000004	0x000084	Reserved
1	0x000006	0x000086	Oscillator Failure
2	0x000008	0x000088	Address Error
3	0x00000A	0x00008A	Stack Error
4	0x00000C	0x00008C	Math Error
5	0x00000E	0x00008E	DMAC Error
6	0x000010	0x000091	Reserved
7	0x000012	0x000092	Reserved
8	0x000014	0x000114	INT0 – External Interrupt 0
9	0x000016	0x000116	IC1 – Input Compare 1
10	0x000018	0x000118	OC1 – Output Compare 1
11	0x00001A	0x00011A	T1 – Timer1
12	0x00001C	0x00011C	DMA0 – DMA Channel 0
13	0x00001E	0x00011E	IC2 – Input Capture 2
14	0x000020	0x000120	OC2 – Output Compare 2
15	0x000022	0x000122	T2 – Timer2
16	0x000024	0x000124	T3 – Timer3
17	0x000026	0x000126	SPI1E – SPI1 Fault
18	0x000028	0x000128	SPI1 – SPI1 Transfer Done
19	0x00002A	0x00012A	U1RX – UART1 Receiver
20	0x00002C	0x00012C	U1TX – UART1 Transmitter
21	0x00002E	0x00012E	AD1 – ADC1 Convert Done
22	0x000030	0x000130	DMA1 – DMA Channel 1
23	0x000032	0x000132	Reserved
24	0x000034	0x000134	SI2C1 – I^2C1 Slave Event
25	0x000036	0x000136	MI2C1 – I^2C1 Master Event
26	0x000038	0x000138	Reserved
27	0x00003A	0x00013A	CN – Input Change Interrupt
28	0x00003C	0x00013C	INT1 – External Interrupt 1

This is just a snippet!

Interrupt related registers I

Register 6-5: IFS0: Interrupt Flag Status Register 0

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIF	MI2CIF	SI2CIF	NVMIF	ADIF	U1TXIF	U1RXIF	SPI1IF
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T3IF	T2IF	OC2IF	IC2IF	T1IF	OC1IF	IC1IF	INT0IF
bit 7							bit 0

bit 15 **CNIF:** Input Change Notification Flag Status bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 14 **MI2CIF:** I^2C Bus Collision Flag Status bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

IFSx: Interrupt Flag Status registers, contains the flags that notify the program of certain events. For example IFS0bits.T1IF tells if the Timer 1 has expired. These flags are set to '1' automatically, but must be set to '0' by the user.

see Chapter 6, pag. 141 of the reference manual

Interrupt related registers II

Register 6-8: IEC0: Interrupt Enable Control Register 0

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIE	MI2CIE	SI2CIE	NVMIE	ADIE	U1TXIE	U1RXIE	SPI1IE
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T3IE	T2IE	OC2IE	IC2IE	T1IE	OC1IE	IC1IE	INT0IE
bit 7							bit 0

bit 15 **CNIE:** Input Change Notification Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

bit 14 **MI2CIE:** I^2C Bus Collision Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

IECx: Interrupt Enable Control registers, it is used to enable an interrupt. For example IEC0bits.T1IE = 1 tells that we want an interrupt every time that IFS0bits.T1IF becomes '1'.

see Chapter 6, pag. 148 of the reference manual

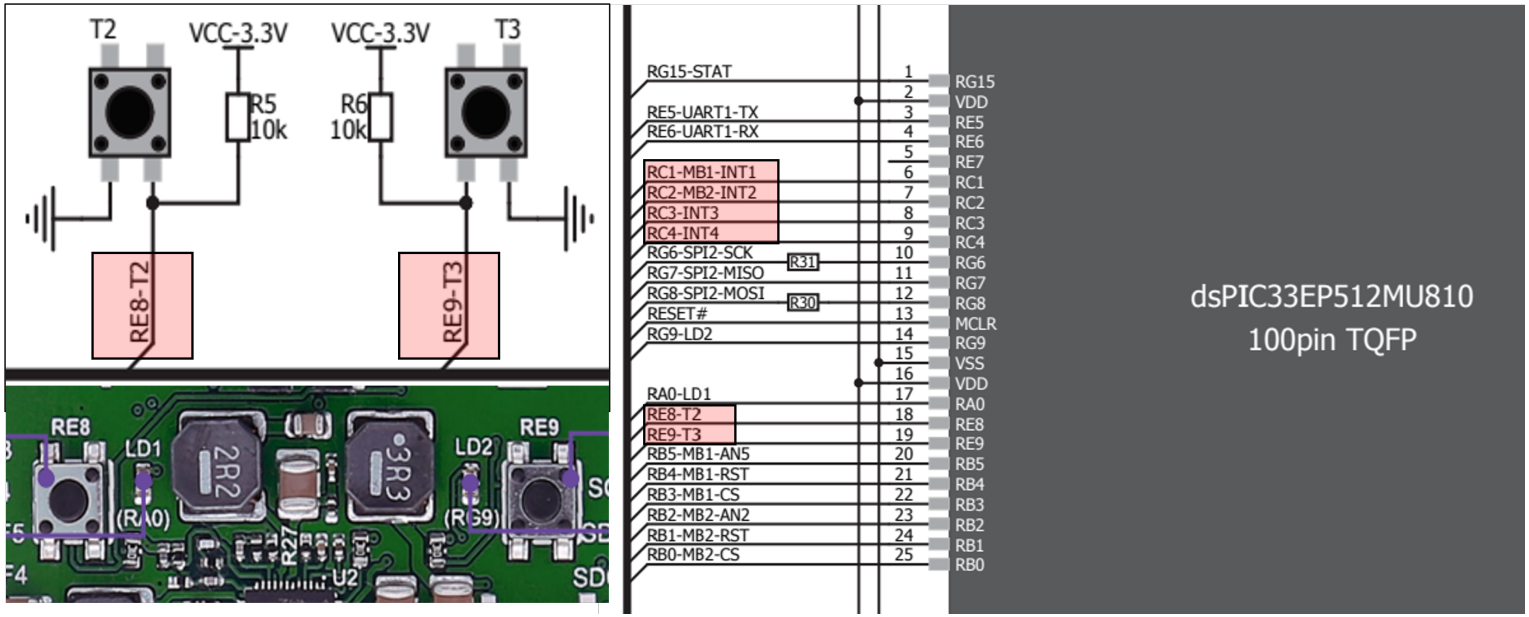
Example interrupt programming

Example: enabling INT0 interrupt

```
void __attribute__((__interrupt__, __auto_psv__)) _INT0Interrupt
    () {
IFS0bits.INT0IF = 0; // reset interrupt flag

[...] // do some action
}

int main() {
    IEC0bits.INT0IE = 1; // enable INT0 interrupt!

    [...] // rest of the code.
}
```

Clicker 2 Board Button/Interrupt I

Unlike DEM2 board, here buttons are not linked to a specific interrupt by default. You need to ***assign pins*** with the ***Peripheral Pin Select***.



E.g. assign External Interrupt 1 (INT1) to RE8 Pin (left button)

Clicker 2 Board Button - PPS Example I

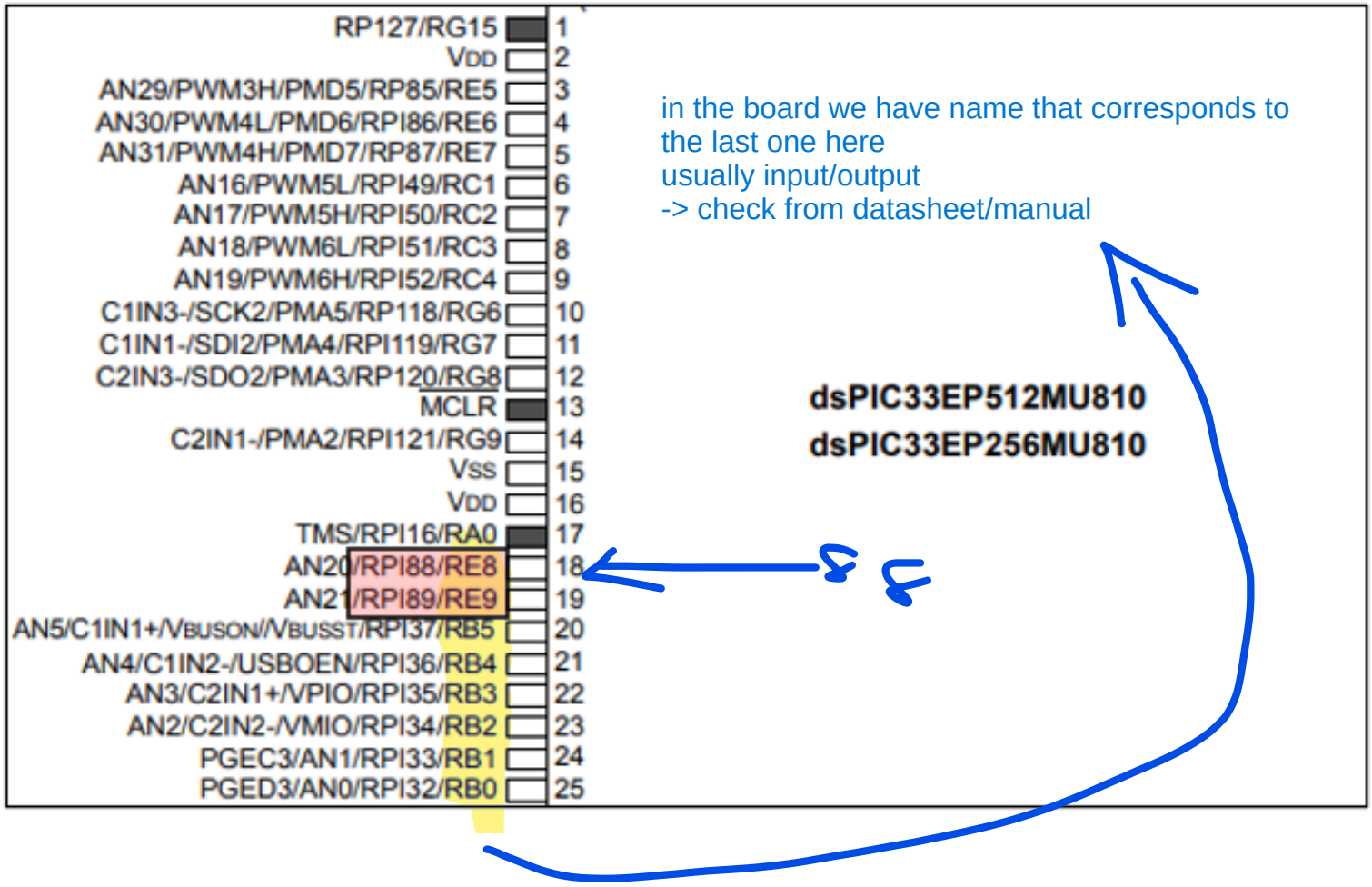
The following table shows ***input mapping*** resources:

Table 4-1: Selectable Input Sources (Maps Input to Function)

Input Name[1]	Function Name	Register Bits	Configuration Bits
External Interrupt 1	INT1	RPINR0<13:8>	INT1R<5:0>
External Interrupt 2	INT2	RPINR1<5:0>	INT2R<5:0>
External Interrupt 3	INT3	RPINR1<13:8>	INT3R<5:0>
External Interrupt 4	INT4	RPINR2<5:0>	INT4R<5:0>
Timer2 External Clock	T2CK	RPINR3<5:0>	T2CKR<5:0>
Timer3 External Clock	T3CK	RPINR3<13:8>	T3CKR<5:0>
Timer4 External Clock	T4CK	RPINR4<5:0>	T4CKR<5:0>
Timer5 External Clock	T5CK	RPINR4<13:8>	T5CKR<5:0>
Input Capture 1	IC1	RPINR7<5:0>	IC1R<5:0>
Input Capture 2	IC2	RPINR7<13:8>	IC2R<5:0>
Input Capture 3	IC3	RPINR8<5:0>	IC3R<5:0>
Input Capture 4	IC4	RPINR8<13:8>	IC4R<5:0>
Input Capture 5	IC5	RPINR9<5:0>	IC5R<5:0>
Output Compare Fault A	OCFA	RPINR11<5:0>	OCFAR<5:0>
Output Compare Fault B	OCFB	RPINR11<13:8>	OCFBR<5:0>
UART1 Receive	U1RX	RPINR18<5:0>	U1RXR<5:0>
UART1 Clear-to-Send	$\overline{\text{U1CTS}}$	RPINR18<13:8>	U1CTSR<5:0>
UART2 Receive	U2RX	RPINR19<5:0>	U2RXR<5:0>
UART2 Clear-to-Send	$\overline{\text{U2CTS}}$	RPINR19<13:8>	U2CTSR<5:0>
SPI1 Data Input	SDI1	RPINR20<5:0>	SDI1R<5:0>
SPI1 Clock Input	SCK1	RPINR20<13:8>	SCK1R<5:0>
SPI1 Slave Select Input	$\overline{\text{SS1}}$	RPINR21<5:0>	SS1R<5:0>
SPI2 Data Input	SDI2	RPINR22<5:0>	SDI2R<5:0>
SPI2 Clock Input	SCK2	RPINR22<13:8>	SCK2R<5:0>
SPI2 Slave Select Input	$\overline{\text{SS2}}$	RPINR23<5:0>	SS2R<5:0>

Note 1: The device may have more or less number of input functions. For actual details, please refer to the specific device data sheet.

You also need to check the ***Remappable Pin Name***:



Clicker 2 Board Button - PPS Example III

Example: Mapping INT1 to RE8 pin of left button

```
// RE8 is RPI88
RPINR0bits.INT1R = 0x59; // 0x59 is 88 in hex
INTCON2bits.GIE = 1;     // set global interrupt enable
INTCON2bits.INT1EP = 1;  // interrupt on negative edge
IFS1bits.INT1IF = 0;     // clear interrupt flag
IEC1bits.INT1IE = 1;     // enable interrupt
```

1st instruction tells where the signal/peripheral is

Assignment: Interrupts

1. Blink D1 led at 1 Hz frequency (500ms time on, 500ms off) without using interrupts; then make LD2 blink at 2 Hz frequency using interrupts.
2. Blink D2 led at 1 Hz frequency (500ms time on, 500ms off) without using interrupts; every time the button E8 is pressed, toggle the led D2 using interrupts.