VE373 Recitation Class

Week 3

2022.05.28

L4 — Timers and IO

1. **GPI/O Ports**

   Each port has 4 associated registers for its operation:

   - `TRISx` register: Data Direction register, or Tri-State Control register
     - `1` for input
     - `0` for output
     - all port I/O pins are defined as inputs after a device Reset. Certain I/O pins are shared with analog peripherals and default to analog inputs after a device Reset.
   - `LATx` register: output latch
     - used to write date to the port I/O pins. The `LATx` Latch register holds the data written to either the `LATx` or `PORTx` registers.
     - reading the LATx Latch register reads the last value written to the corresponding PORT or Latch register.
   - `PORTx` register: reads the levels on the pins of the device
     - used to read the current state of the signal applied to the port I/O pins
     - writing to a `PORTx` register performs a write to the port's latch, `LATx` register, latching the data to the port's I/O pins
   - `ODCx` register: Open-drain control

     Pins are configured as digital outputs by setting the corresponding `TRIS` register bits = 0. When configured as digital outputs, these pins are CMOS drivers or can be configured as open-drain outputs by setting the corresponding bits in the Open-Drain Configuration (`ODCx`) register.

     The open-drain feature allows generation of outputs higher than V_{DD} (e.g., 5V) on any desired 5V tolerant pins by using external pull-up resistors. The maximum open-drain voltage allowed is the same as the maximum V_{IH} specification.

2. **`CLR`, `SET` And `INV` Registers**

   Provide fast atomic bit manipulations, perform operation in hardware atomically.

   Reading SET, CLR and INV registers returns undefined values.

   E.g. `T2CONSET = 0x8000` in homework 1.

L5 — Interrupts

1. **Detect events**

   Dealing with asynchronous events:

   - Exceptions caused by program execution
   - Peripheral needs attention or has completed a requested action
   - Application makes a system call

2. **Software polling**

   - We can repeatedly poll the application for peripherals.
   - When an event occurs, detect this via a poll and take action
   - Common in small or low-performance real-time embedded systems

    - Predictable timing
    - Low hardware cost
  - In other systems, wastes CPU time
    - Affect processor's responsiveness and efficiency

```c
/* Timer control */
T2CON = 0x0;        // Stop Timer2 and clear control register
                    // prescale 1:1, internal clock source
TMR2 = 0x0;         // Clear Timer
PR2 = 0xFFFF;
T2CONSET = 0x8000; //Start Timer2

/* polling */
while (TMR2 != 1000);

/* move on */
...
```

3. **Interrupt**

   - Let the application for peripheral notify us automatically.
   - Take action (respond to the event) when such a notification occurs (or shortly later).

   When a special event or error occurs

   - I/O controller interrupts CPU – by a call from the I/O hardware
   - CPU stops executing the current program
   - CPU goes to run an **interrupt handler** (a special piece of program) or **interrupt service routine** (ISR)
   - After finish executing the ISR, CPU returns back to the interrupted program and resume
   - Not synchronized to instruction execution, may happen between any two instructions

4. **Typical Types of Interrupts**

   - Internal (exception)
     - By CPU errors
     - By bad instructions
   - External (interrupt)
     - External I/O device • Timer
     - Reset
     - System failure

   Interrupt comes as a high-level signal or a rising edge depending on processors

5. **SFRs for Interrupt**

   - `INTCON` – Interrupt Control Register
   - `INTSTAT` – Interrupt Status Register
   - `TPTMR` – Temporal Proximity Timer Register
   - `IFSx` – Interrupt Flag Status Registers
   - `IECx` – Interrupt Enable Control Registers
   - `IPCx` – Interrupt Priority Control Registers

6. **Interrupt Control**

   1. Enable interrupt globally
   2. Config `INTCON` to select multi vector mode.
   3. Enable individual interrupt, set priority

   4. When interrupt event happens, flag is set regardless of the enable bit
   5. If enabled, interrupt service routine (ISR)
   6. In ISR, service interrupt, clear interrupt flag (for most interrupt requests)
   7. Main program resumes

7. **Enable/disable all interrupt sources globally**
   - `asm("ei"); // inline MIPS32 assembly`
   - `asm("di");`
   - Corresponding to enable/disable IE bit (`STATUS<0>`)
   - Must be enabled before individual interrupt can be used

8. **Operation Mode**
   - Controlled by `MVEC` bit in `INTCON`
   - **Single vector mode:**
     - All interrupt requests will be serviced at one vector address (default upon reset)
     - One location for all different interrupts
     - have to determine what exception/interrupt has just happened, by checking status registers
     - `if...else...` structure
   - **Multi vector mode:** Interrupt requests will be serviced at the calculated vector address
     - One location for one (or a small number of) interrupt
     - Take actions or handle interrupts directly

9. **Interrupt Vector Table**

TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION

Interrupt Source(1)	IRQ Number	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
Highest Natural Order Priority						
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>
INT0 – External Interrupt 0	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>
IC1 – Input Capture 1	5	5	IFS0<5>	IEC0<5>	IPC1<12:10>	IPC1<9:8>
OC1 – Output Compare 1	6	6	IFS0<6>	IEC0<6>	IPC1<20:18>	IPC1<17:16>
INT1 – External Interrupt 1	7	7	IFS0<7>	IEC0<7>	IPC1<28:26>	IPC1<25:24>
T2 – Timer2	8	8	IFS0<8>	IEC0<8>	IPC2<4:2>	IPC2<1:0>
IC2 – Input Capture 2	9	9	IFS0<9>	IEC0<9>	IPC2<12:10>	IPC2<9:8>
OC2 – Output Compare 2	10	10	IFS0<10>	IEC0<10>	IPC2<20:18>	IPC2<17:16>
INT2 – External Interrupt 2	11	11	IFS0<11>	IEC0<11>	IPC2<28:26>	IPC2<25:24>
T3 – Timer3	12	12	IFS0<12>	IEC0<12>	IPC3<4:2>	IPC3<1:0>
IC3 – Input Capture 3	13	13	IFS0<13>	IEC0<13>	IPC3<12:10>	IPC3<9:8>
OC3 – Output Compare 3	14	14	IFS0<14>	IEC0<14>	IPC3<20:18>	IPC3<17:16>
INT3 – External Interrupt 3	15	15	IFS0<15>	IEC0<15>	IPC3<28:26>	IPC3<25:24>
T4 – Timer4	16	16	IFS0<16>	IEC0<16>	IPC4<4:2>	IPC4<1:0>
IC4 – Input Capture 4	17	17	IFS0<17>	IEC0<17>	IPC4<12:10>	IPC4<9:8>

10. **ISR – Interrupt Service Routine**

    Discussed in the next week's RC.

Tips for Homework

Define a new 32-bit SFR named `MCUCON` as a C structure. The SFR contains four fields:

- `ON`: in `MCUCON[31]`
- `MODE`: in `MCUCON[24:22]`
- `STATUS`: in `MCUCON[15:8]`
- `IPL`: in `MCUCON[7:6]`

You can refer to the header file of the PIC32 board we use. You have two ways to find it:

1. Access it on github.
2. You can create a test.c file, include header file `p32xxxx.h`, and write a SFR in `main()` (e.g. `T1CON`). Then by holding ctrl and clicking on it, you can navigate to its definition through the built-in function of IDE. You may want to do that in future labs.

The header file defines `T1CON` as shown below:

```
#define T1CON T1CON
extern volatile unsigned int   T1CON __attribute__((section("sfrs")));
typedef union {
  struct {
    unsigned :1;
    unsigned TCS:1;
    unsigned TSYNC:1;
    unsigned :1;
    unsigned TCKPS:2;
    unsigned :1;
    unsigned TGATE:1;
    unsigned :3;
    unsigned TWIP:1;
    unsigned TWDIS:1;
    unsigned SIDL:1;
    unsigned :1;
    unsigned ON:1;
  };
  struct {
    unsigned :4;
    unsigned TCKPS0:1;
    unsigned TCKPS1:1;
  };
  struct {
    unsigned :13;
    unsigned TSIDL:1;
    unsigned :1;
    unsigned TON:1;
  };
  struct {
    unsigned w:32;
  };
} __T1CONbits_t;
extern volatile __T1CONbits_t T1CONbits __asm__ ("T1CON") __attribute__((section("sfrs")));
extern volatile unsigned int        T1CONCLR __attribute__((section("sfrs")));
extern volatile unsigned int        T1CONSET __attribute__((section("sfrs")));
extern volatile unsigned int        T1CONINV __attribute__((section("sfrs")));
```

You only need to define the union as shown above.