VE373 Recitation Class

Week 4

2022.06.04

L5 — Interrupts

1. **ISR — Interrupt Service Routine**

   Interrupt flag should be cleared in ISR. Otherwise interrupt will be triggered over and over again.

   In ISR, you need to

   - Clear interrupt flag
   - Perhaps, stop peripheral
   - Perhaps, disable interrupt (if in critical section)
   - Do whatever you need to do, fix problem, respond to the peripheral
   - Perhaps, some configuration
   - Perhaps, restart peripheral for next interrupt
   - Perhaps, re-enable interrupt
   - Perhaps, start another peripheral for another interrupt
   - Return to the interrupted instruction

2. **Critical section/region**

   Should NOT be interrupted

   Should disable interrupts while executing those instructions

3. **Interrupt priority**

   Devices needing more urgent attention get higher priority

   Higher priority interrupt can interrupt execution of a lower priority interrupt

   Three priority levels

   - Group level: 0 - 7 (highest)
   - Subgroup level: 0 - 3 (highest)
   - Natural level: 0 - 63 (lowest)

4. **Preemption**

   Higher priority interrupt request (IRQ) preempts (overrides) any lower priority interrupts (on group level).

   No preemption on subgroup and natural levels.

5. **IPL and RIPL**

   - Requested interrupt priority level (RIPL)
     - In field `RIPL` (`CAUSE<12:10>`)
     - 0. . . 7, encode requested interrupt priority
   - Current CPU priority (IPL)
     - In field `IPL` (`STATUS<12:10>`)
     - Usually default 0 (no interrupt), changes to IRQ priority when servicing IRQ
     - 0. . . 7, encode the MIPS interrupt priority hierarchy
   - Preemption happens only when `RIPL` > `IPL`

6. **How to write ISR**

```c
#pragma interrupt foo ipl4 vector @23
void foo(void){
  ...
}
// foo will be located at the address of interrupt vector 23

#pragma interrupt bar ipl5 vector 23
void bar(void){
  ...
}
// bar will be located in general purpose program memory
// A dispatch function targeting bar will be created at exception vector address
    ↪ 23
```

`iplx`, `x` = 0...7, 0 means interrupt disabled. `x` must match the group interrupt priority level specified in `IPCx`.

Each interrupt vector has 8 words, usually holding dispatch function that associates the vector address with the real interrupt handler function.

7. **Change Notice (CN)**

   Generate interrupt upon change of state on selected CN pins

   - CN pins must be configured as inputs
   - Two adjacent reads (delayed by `SYSCLK`) of CN PORT are compared, mismatch generated if different
   - Any mismatch provides an interrupt signal

8. **CN configuration**

   1. Disable all interrupts
   2. Set selected CN pin as input
   3. Enable CN module (`CNCON.ON = 1`)
   4. Enable individual CN inputs, and pull ups (optional)
   5. Read corresponding PORT registers to clear pre-existing mismatch condition
   6. Configure the CN interrupt priority
   7. Clear CN interrupt flag
   8. Enable CN interrupt
   9. Enable all interrupts

9. **CN ISR**

   1. Temporarily disable CN interrupt
   2. PORT should be read to clear mismatch condition
   3. Interrupt flag should be cleared by user program
   4. Current PORT value should be compared with the last PORT value of the same pin to determine which CN pin changed
   5. Re-enable CN interrupt

   Note:

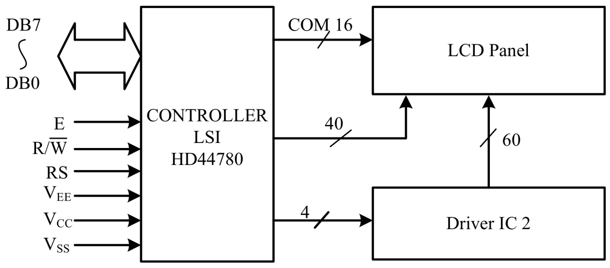
   1. CN interrupt is a persistent interrupt: only clearing IF flag is not enough, must clear the corresponding condition.
   2. CN interrupts of all pins go into the same vector address (vector number 26). Therefore in ISR we should check which IO pin is changed.

L6 — Liquid-Crystal Display (LCD) Driver

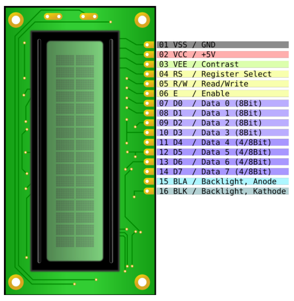
1. **Make good use of datasheet**

   The datasheet of LCD is file `lcd1602a_398762.pdf` in `./Reference_Materials/PIC32 Starter Kit/`.

2. **LCD module**



(a) LCD module



(b) LCD pins definition

3. **Display Data RAM (DDRAM)**

   - Can hold up to 80 bytes (characters)
   - 40 locations mapped to Line1: `0x00 - 0x27`
   - 40 locations mapped to Line2: `0x40 - 0x67`
   - 2 line × 16 display window is aligned with DDRAM from the head

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

2-Line by 16-Character Display

   - Display window can be shifted left or right

4. **Code Generate ROM (CGROM)**

   It can be understood as the 8-bit encoding of actual character patterns. Almost the same as ASCII chart except some characters.

5. **Control Inputs**

   - RS, R/W

RS	R/W	Operation
0	0	Write data as a configuration instruction
0	1	Read data as busy flag (DB7) and address counter (DB6 to DB0)
1	0	Write data as a character to be displayed
1	1	Read data from the internal memory

   - E – data transfer enable

     a High to Low transition on E enables the data transfer

6. **Instruction table**

   Defines the inputs to perform specific instruction. Refer to the datasheet.

7. **Timing requirements**

   - Important, must be satisfied.

- All instruction execution time must be satisfied, except
  - For 4-bit interface, two consecutive writes, one for high nibble, one for low nibble, need no delay in between, but need 40 μsec after
  - read instruction needs no time
- Each write operation is enabled by a high to low transition on E input including the two consecutive nibble write

8. **Initialization**

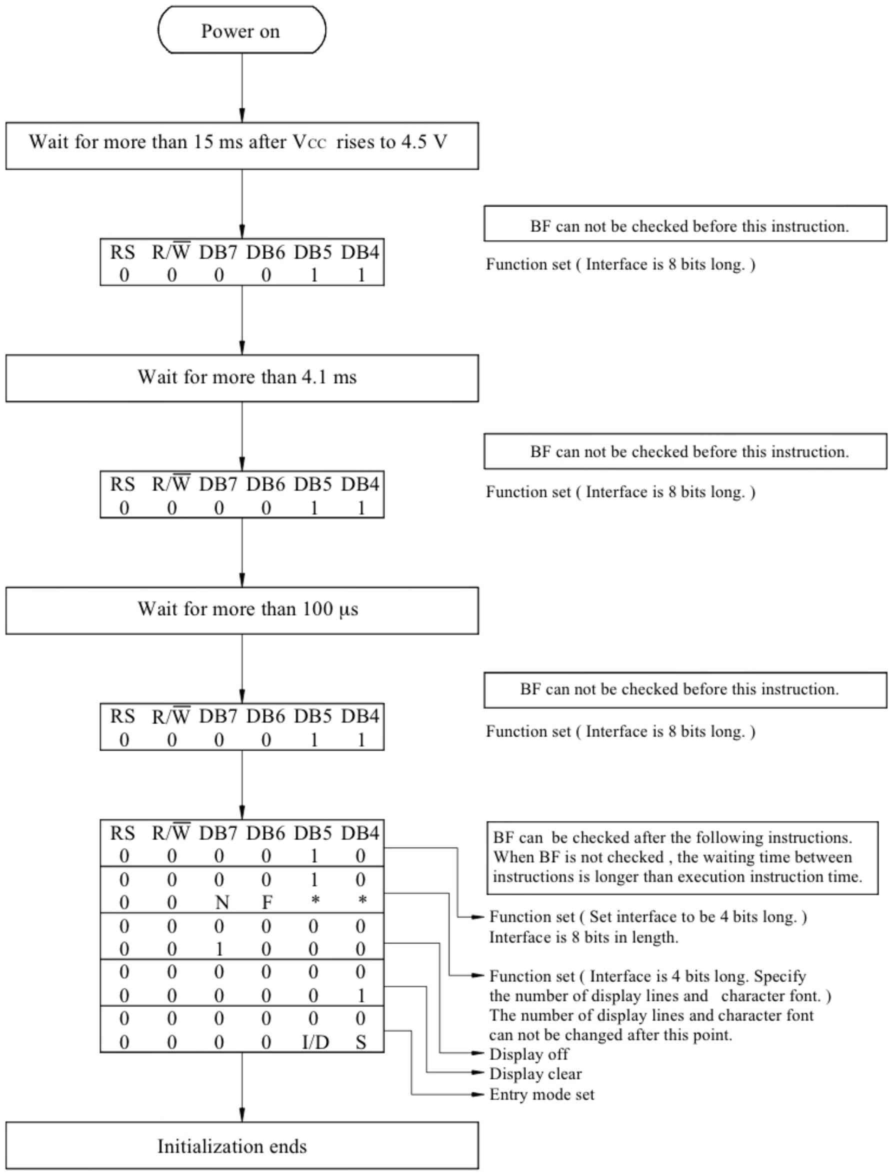

Power on
Wait for more than 15 ms after Vcc rises to 4.5 V
RS R/W̄ DB7 DB6 DB5 DB4
0 0 0 0 1 1
BF can not be checked before this instruction.
Function set (Interface is 8 bits long.)
Wait for more than 4.1 ms
Wait for more than 100 μs
BF can be checked after the following instructions. When BF is not checked , the waiting time between instructions is longer than execution instruction time.
Function set (Set interface to be 4 bits long.) Interface is 8 bits in length.
Function set (Interface is 4 bits long. Specify the number of display lines and character font.) The number of display lines and character font can not be changed after this point.
Display off
Display clear
Entry mode set
Initialization ends

Figure 2: LCD 4-bit interface initialization

9. **Tips**

- This chapter mainly focuses on the usage of LCD. Not much concepts.
- The usage of LCD will cover using timer and interrupt to generate delays, and using GPIO to communicate with LCD.
- The corresponding lab will be the first hardcore lab you meet in this course.
- You may have to debug your program for quite loooooong time. Start early.

L7 — Power-saving Operations

1. **Static/Dynamic power**

   Dynamic power is comprised of switching and short-circuit power.

   Static power is comprised of leakage, or current that flows through the transistor when there is no activity.

   $$P_{\text{dynamic}} = CV^2 f$$
   $$P_{\text{static}} = VI_{\text{static}}$$

2. **Reduce power consumption**

   - Reduce operation frequency (lower frequency, lower power consumption)
   - Halt CPU or disable peripheral modules

3. **9 power saving modes**

   - CPU running
     - FRC RUN mode: CPU uses `FRC` clock source
     - LPRC RUN mode: CPU uses `LPRC` clock source
     - SOSC RUN mode: CPU uses `SOSC` clock source
     - Peripheral bus scaling mode: `PBCLK` is fraction of `SYSCLK`
   - CPU halted
     - SLEEP mode: anything using `SYSCLK` (CPU and peripherals) halted, peripheral using other clock source are operating – lowest power consumption
     - POSC IDLE mode: Primary Oscillator
     - FRC IDLE mode: Fast RC Oscillator (8 MHz)
     - LPRC IDLE mode: Low-Power RC Oscillator (32 KHz)
     - SOSC IDLE mode: Secondary Oscillator (32.768 KHz)

4. **SLEEP mode**

   - Characteristics
     CPU and most peripherals are halted
     System clock source is shut down
     Lowest power consumption
     Several peripherals alive to wake up CPU
   - Enter SLEEP mode
     `SLPEN` (`OSCCON<4>`) bit set, followed by "wait" assembly instruction
   - Exit sleep mode (wake up)
     By these events
     - Interrupt from operating peripheral
     - Watch dog timer
     - Reset

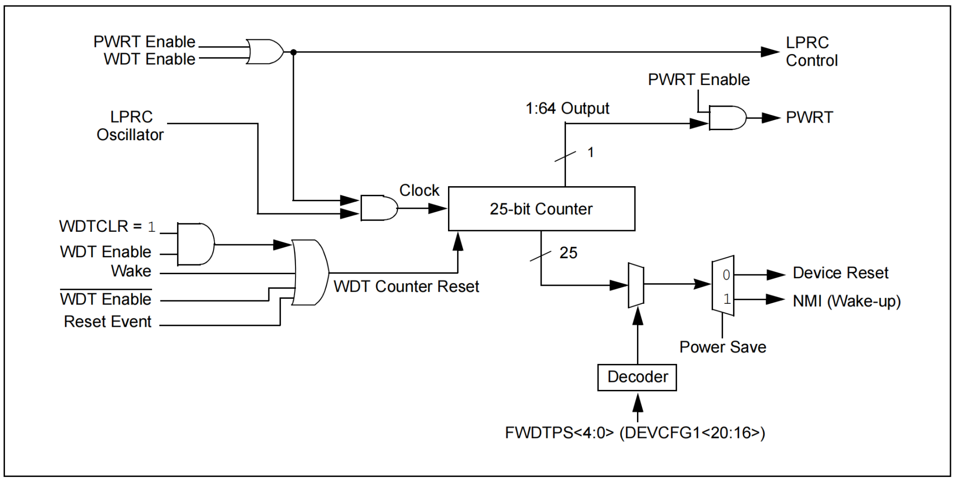
     May need start-up delay
     - Program not start running until clock signal detected stable
     - May report fail if delay is too long
   - Example

```
SYSKEY = 0x0;              // Write invalid key to force lock
SYSKEY=0xAA996655;         // write Key1 to SYSKEY
SYSKEY=0x556699AA;         // Write Key2 to SYSKEY
OSCCONSET = 0x10;          // Set power-saving mode as SLEEP
SYSKEY = 0x0;              // Write invalid key to force lock

asm volatile ("wait");     // Put device in selected power-saving mode
                           // code execution will resume here after
```

```
                         // wake and the ISR is complete.
                         // "volatile" forces instruction location in
                         // the program

...user code after wake-up...
```

5. **IDLE mode**
   - CPU is halted
     - System clock source is still running
     - Peripherals alive, unless individually configured to halt when in IDLE
     - Low start-up latency
     - Operating clock frequency may be slowed down to reduce power consumption
   - Enter IDLE mode
     `SLPEN` (`OSCCON<4>`) bit cleared followed by "wait" assembly instruction, e.g.
     `OSCCONCLR = 0x10; asm ("wait");`
   - Exit from IDLE mode
     - Interrupt from operating peripheral
     - Watch dog timer
     - Reset
6. **Wake-up CPU**
   - Peripheral interrupt
     - If current CPU priority is greater (IPL > RIPL), CPU remains halted
       System remains IDLE, or system enters IDLE from SLEEP
     - If requested peripheral IRQ priority is greater (RIPL > IPL)
       CPU services IRQ, then continues executing instruction following "wait"
   - WDT (watch dog timer) Non-Maskable Interrupt (NMI)

FIGURE 28-1: WATCHDOG TIMER AND POWER-UP TIMER BLOCK DIAGRAM



Also used to check software malfunction. Error if software not periodically clear WDT