嵌入式计算机系统与实验 | 第一次作业

Shuiyuan@Noroshi

1 Keil 软件使用入门

1.1

在安装软件的时候，我并没有在网上找到除了官方提供的最新版以外的版本，所以只能下载最新版的 µVision 5.38 的版本。但出现了一个问题就是此处在创建项目的时候此处“Software Packs”的下拉菜单一开始是灰色的。在网上查询需要额外下载旧版包，在官网上下载旧版包后就可以成功选择 Legacy Device Database.

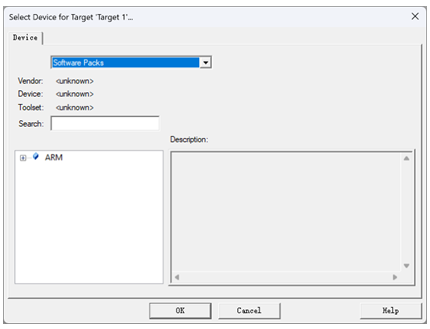

Figure 1: 一开始的时候此处选择框内无文字, 下拉菜单也是灰色的

1.2

安装完成后我按照 PPT 内容写入了如下示例代码，并成功编译:

```
ORG   000h              ;以下部分程序位于000h
    LJMP  START         ;跳转至START标签处
ORG   040H              ;以下部分程序位于040h
START:  MOV   SP, #5FH  ;将立即数5F赋值给SP寄存器
    MOV   A,  #05H      ;将立即数5赋值给A寄存器
    MOV   B,  #03H      ;将立即数3赋值给B寄存器
    ADD   A,  B         ;A,B相加并将值赋给A寄存器
LOOP1:  NOP               ;无操作
    LJMP  LOOP1         ;跳转至LOOP标签处(死循环)
END
```

进入调试模式后，已跳转至 START 标签处，同时可见上部分的反汇编代码片段中的操作数（75815F）与 memory 中记录的实际数值也是对应的，观察汇编程序可知其 START 部分的内容是分别给 SP，A，B 三个寄存器分别赋立即数 5F，5，3，并将 A，B 相加（存于 A）。这个在运行结束后可以在 Register 栏得到验证。

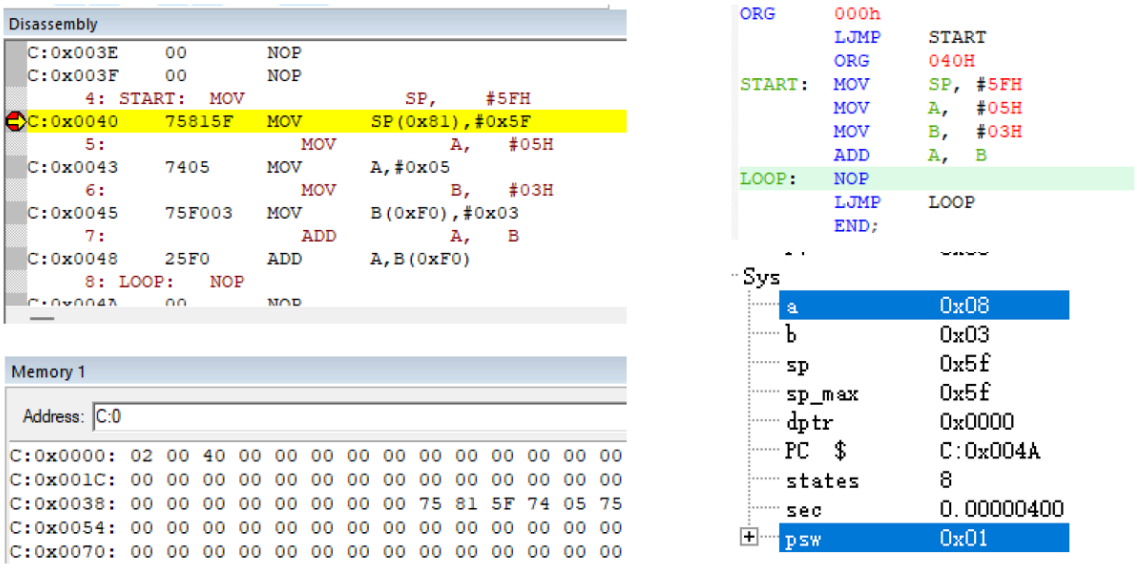

Figure 2:

左二图: 可见反汇编代码操作数与内存中对应地址上的操作数相同;
右二图:可见寄存器中存储的数值(A 原为 5, B 原为 3, 执行 ADD 指令后 A 变为 8, 与 START 部分的程序指令相同)

1.3

为了更进一步了解调试流程我又尝试了更多代码（注：本段代码由 ai 生成）

```
ORG 0H              ;以下部分程序位于000h
START:
    MOV P1, #0xFF   ;将立即数FF赋值给P1端口(即8个端口全输出高电位)
    MOV R0, #0x05   ;将立即数5赋值给R0寄存器
DELAY1:
    DJNZ R0, DELAY1    ;将R0值--, 若非0则跳转至DELAY1标签处
    MOV P1, #0x00      ;将立即数0赋值给P1端口(即8个端口全输出低电位)
    MOV R0, #0x05      ;将立即数5赋值给R0寄存器
DELAY2:
    DJNZ R0, DELAY2    ;将R0值--, 若非0则跳转至DELAY2标签处
    SJMP START         ;跳转至START标签处(无限循环)
END
```

`DJNZ Reg, Tag` 的意思则表示将 Reg 中的数递减 1，若非零则跳转至 Tag. 这段代码的作用是反复亮灯。其中，P1 即为输出串口，其数据可在调试时在 Peripherals>I/O Ports>Port 1 中查看（左即为全高电位，右即为全低电位）

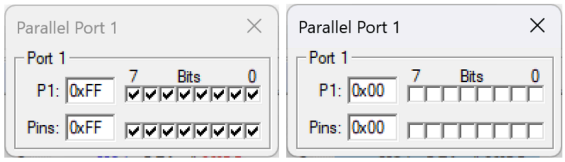

Figure 3: Debug 时打开 Peripherals>I/O Ports>Port 1 监视窗口可以观察到 P1 的电位变化符合预期

2 Keil, Proteus 的联调

2.1

依照 PPT 中的安装方法，我顺利完成了 Proteus 的安装，并在下载额外 ddl 后成功实现了 Keil 与 Proteus 的基础的联调例程

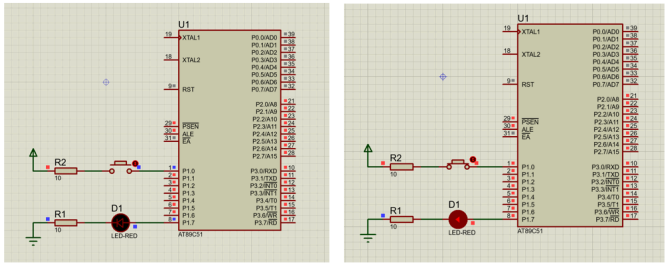

Figure 4: 按钮按下灯亮, 按钮松开灯灭,符合预期

2.2

此外在之前 1.3 的基础上我手动修改了一下, 以尝试更多代码的联调流程. 修改后的代码实现了 P1 端口所连八个灯的循环点亮步骤:

```
ORG 0H                  ;以下部分程序位于000h
START:
  DJNZ R0, START        ;若R0非0则跳转至START标签处
  MOV P1, #00000001B    ;将二进制00000001赋值给P1(即仅P1.0端口为高电位)
  MOV R0, #0XFF         ;将立即数FF赋值给R0端口
DELAY1:
  DJNZ R0, DELAY1       ;R0--, 若非0则跳转至DELAY1标签处
  MOV P1, #00000010B    ;将二进制00000010赋值给P1(即仅P1.1端口为高电位)
  MOV R0, #0XFF         ;将立即数FF赋值给R0端口
DELAY2:
  DJNZ R0, DELAY2       ;以此类推
  MOV P1, #00000100B
  MOV R0, #0XFF
; ...
; 此处省略了部分代码
; ...
DELAY7:
  DJNZ R0, DELAY7
  MOV P1, #10000000B
  MOV R0, #0XFF
  LJMP START
END
```

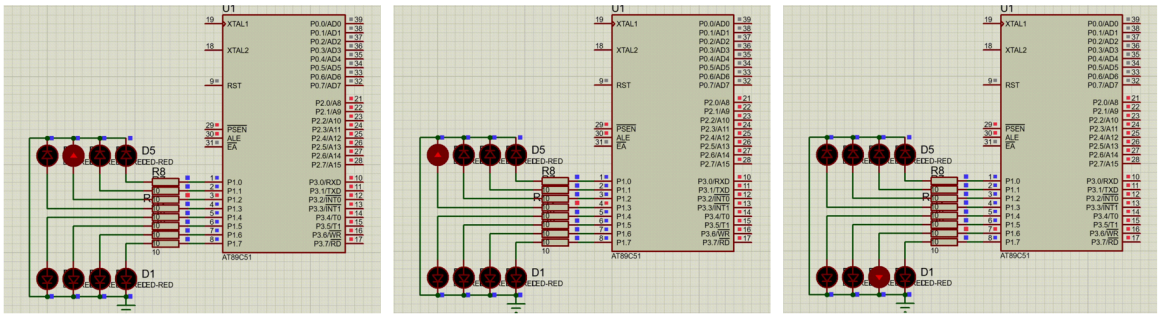

Figure 5: 运行流程截图, 从端口电位和亮灯情况可见符合预期

3 MCS-51 的最小系统

MCS-51 的最小系统包含 1.电源电路, 2.复位电路, 3.时钟电路, 4.外部 Memory 等. 在 Proteus 里似乎 51 单片机电源电路是默认启用且隐藏的, 而外部 Memory 在较小的程序时也并非必须. 因此目前我们就先着眼于复位电路和时钟电路上.

3.1 时钟电路

最基础的时钟电路(的外部晶体连线部分)由一个石英晶体与两个电容构成. 其与片内的对应电路构成振荡器, 成为 CPU 的工作时钟. 振荡周期就是时钟信号的周期, 其由振荡器的晶振或者其他时钟源提供; 机器周期是单片机执行一条指令所需(最短)的时间, 对 a51 单片机来说通常是振荡周期的 12 倍. 也有很多命令需要不止 1 个机器周期来执行.

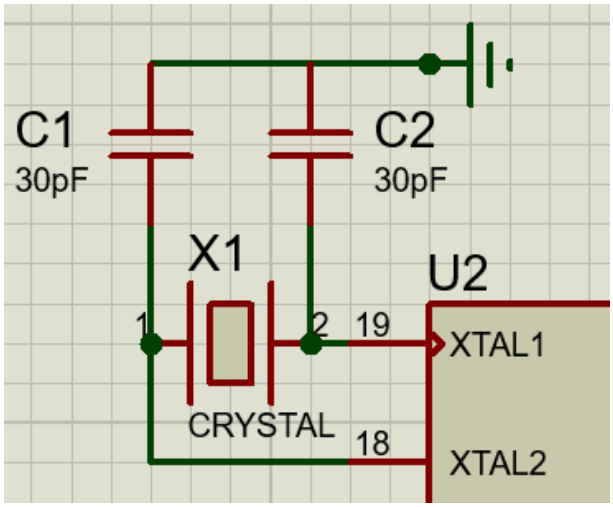

Figure 6: 最基础的时钟电路

当然也有很多其他的时钟电路, 如 HMOS/CHMOS 外部震荡源等.

3.2 复位电路

基本复位电路包括上电自动复位和手动自动复位, 其最终目的都是使 RESET 端口进入>2 个机器周期的高电平. 就可以实现复位.

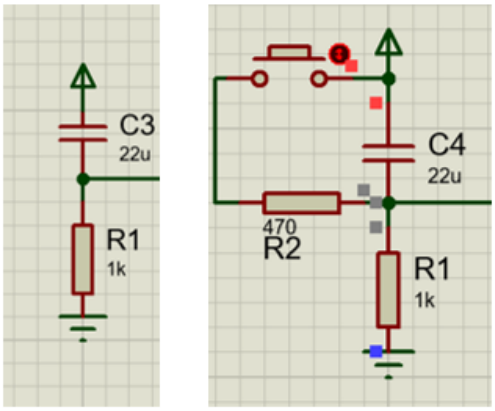

Figure 7: 左:上电自动复位, 右:手动(按钮)复位

上电复位的原理即是在电源打开瞬间, 电容在充电时所分电压较小, 使 RESET 端口处在高电位状态, 触发复位, 充满电之后 RESET 端口进入低电平状态.

3.2.1 上电自动复位

当使用一般的电容和示波器的时候, 处于软件的限制, 当示波器开始反馈波形的时候已经进入稳态了, 难以观测. 需要使用一些其他办法来更准确地模拟与采集数据.

1. 与普通的电容(CAP)相比, 另一种电容能更加准确地模拟电容(REALCAP), 使用这种电容可以反映出电容在达到稳态前的变化
2. 使用 ANALOGUE ANALYSIS 来更准确地采集上电内短时间的电压波形, 使用方法:
    1. 选择最左侧边栏中的“探针模式”, 选择“VOLTAGE”电压探针选项.
    2. 将电压探针放置于 RESET 端口的连线上
    3. 选择最左侧边栏中的“图形模式”, 选择“ANALOGUE”图形模式, 拉一个大小合适的框框出来
    4. 右键菜单“添加曲线”, 选择对应的探针
    5. 点击“仿真图表”(似乎这个仿真图标要求单片机必须有一个程序, 不过实际内容无所谓)绘制结果

最后我们就可以得到准确的上电复位时的电压图. 观察电压图可以看出, 电压>2v(高电位)的时间大约有 80ms 左右, 远高于两次机器周期的时间, 说明这个上电复位的电路是正常的.

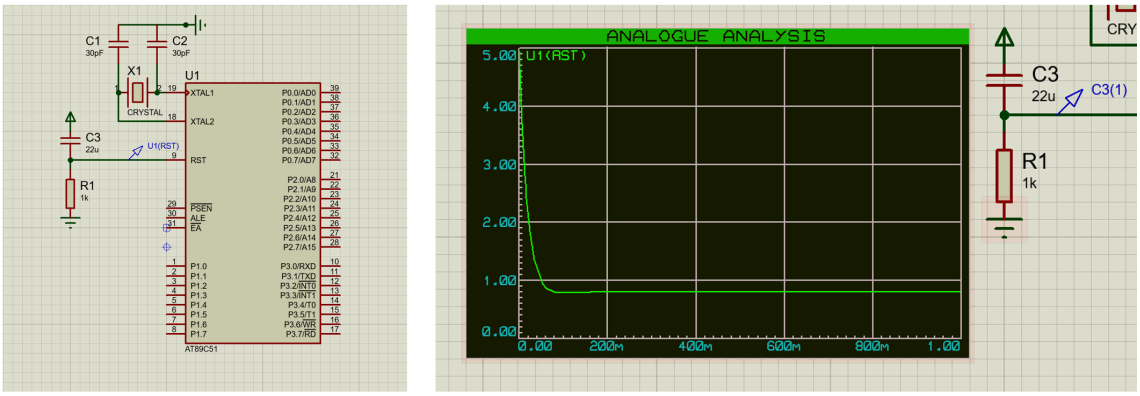

Figure 8: 左: 包含上电复位电路的模拟图, 右: 绘制的电压与时间的关系图

3.2.2 手动复位电路

手动复位电路就可以正常用示波器来测量了. 虽然大学物理实验和电路实验已经用过无数次示波器了, 但在模拟环境下, 也要注意将接受 CHANNEL 设置为 DC 模式, 同时, 调整好水平和竖直坐标轴的便宜与坐标大小, 使结果可以被清晰而准确地观测. 最终结果如下图所示. 可以看见, 即使在短按的情况下, RESET 接口仍然接收到了大约 170ms 左右的高电平信号, 足以触发复位.

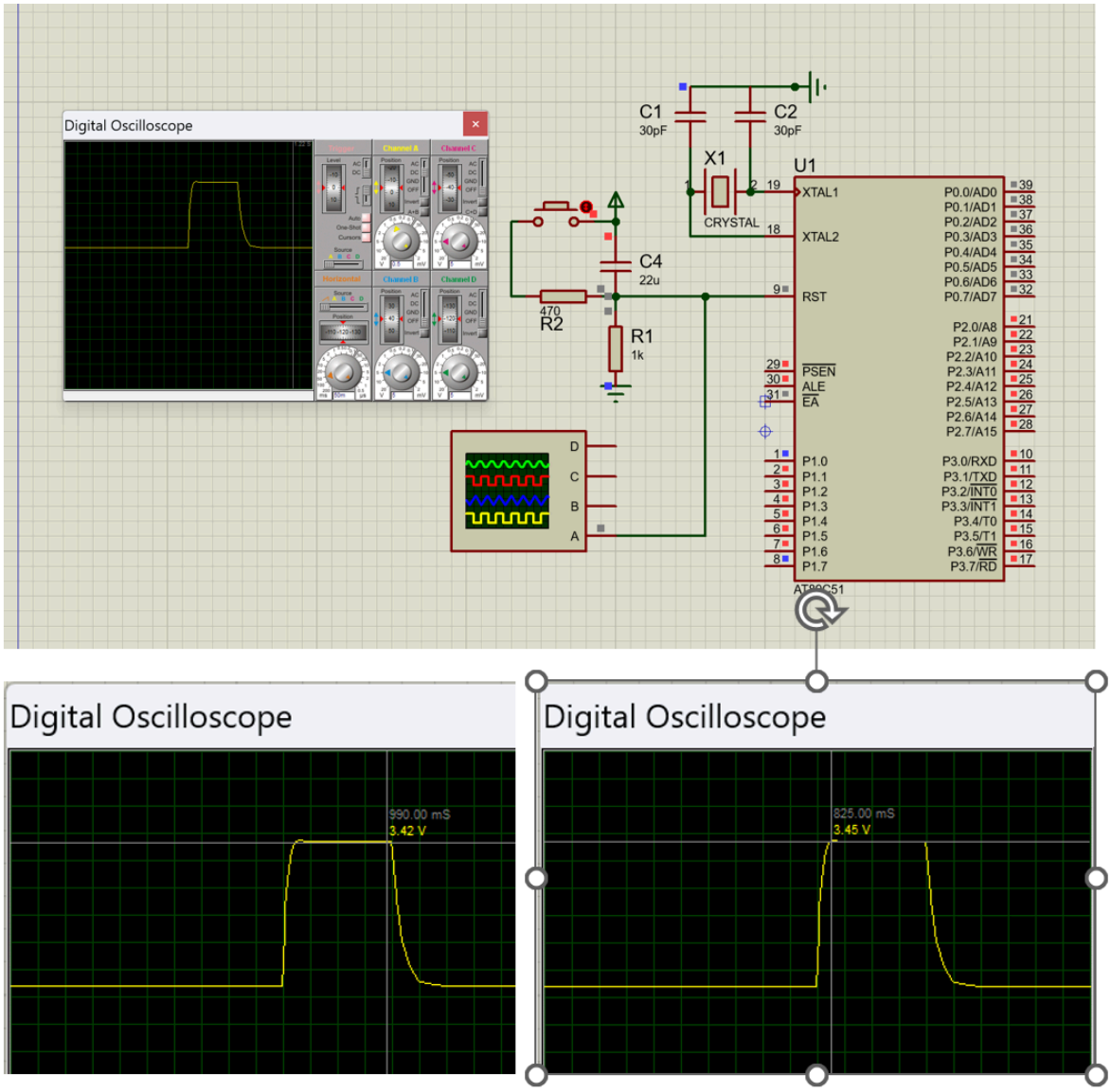

Figure 9: 上: 手动复位电路, 下: 按下复位键时的点位变化

3.3 外部 memory 拓展电路

当内部的 ROM 和 RAM 不够用 1 时候, 通常我们可以进行外扩的程序储存器, 其中, 74L373 是锁存器, 而 2784 是 EPROM, 一般用来存储烧录的程序.

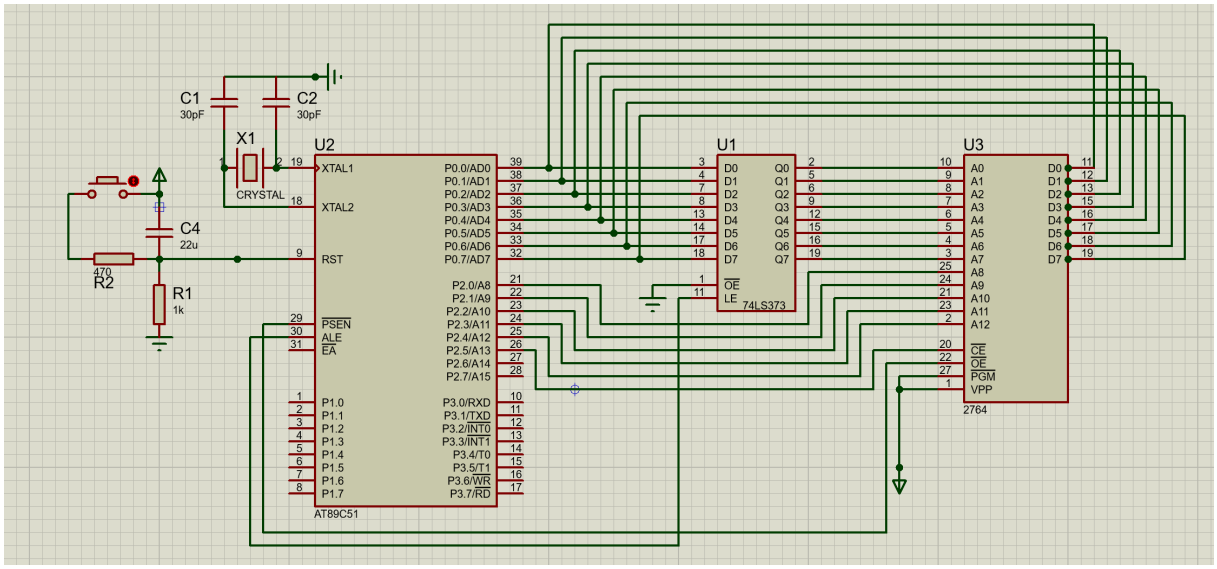

Figure 10: 完整的包含电源电路(已隐藏), 时钟电路, 复位电路, 外部 memory 的 MCS-51 最小系统