第2章 S800硬件系统与嵌入式开发

本章节参考资料：

A. 《The Definitive Guide to Arm Cortex M3 and Cortex M4 Processors》（中译本《ARM Cortex-M3与Cortex-M4权威指南》）

B. Tiva™ TM4C1294NCPDT Microcontroller Data Sheet

C. S800板介绍V0.65

第2章 S800硬件系统与嵌入式开发

2.1 TM4C1294NCPDT微控制器（ref.B-chapter1）

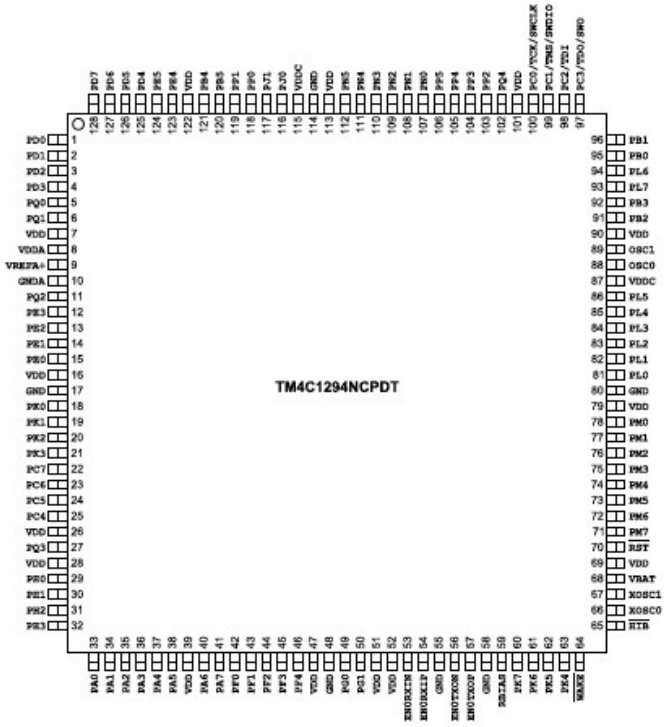
2.2 基于TM4C1294 MCU的S800实验板（ref.C）

2.3 嵌入式软件开发（ref.A-chapter2，20）

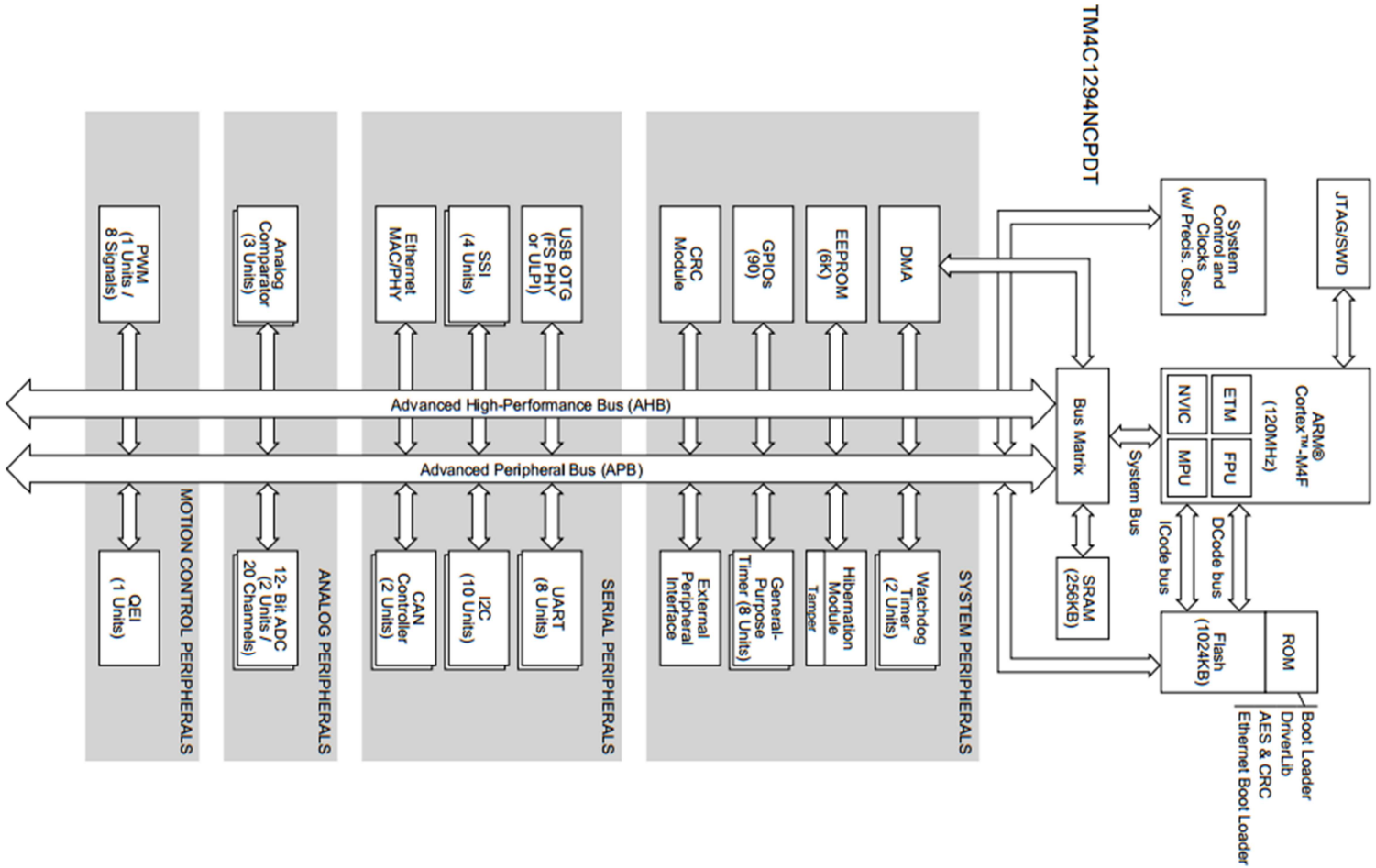
2.4 S800的嵌入式软件开发（ref.A-chapter15）

2.1 TM4C1294NCPDT 微控制器

- TM4C1294NCPDT 微控制器内部结构
  - 128-Pin TQFP Package

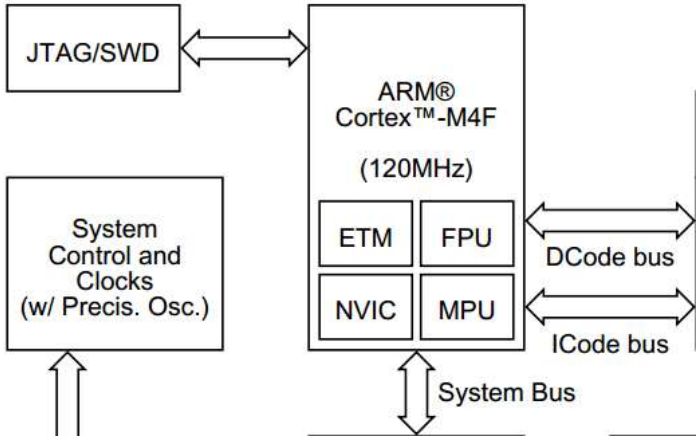


- TM4C1294NCPDT MCU内部结构

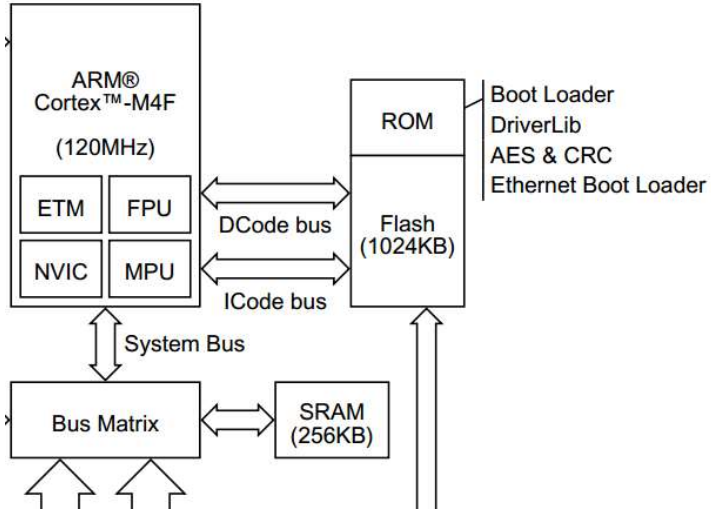


TM4C1294NCPDT MCU特性

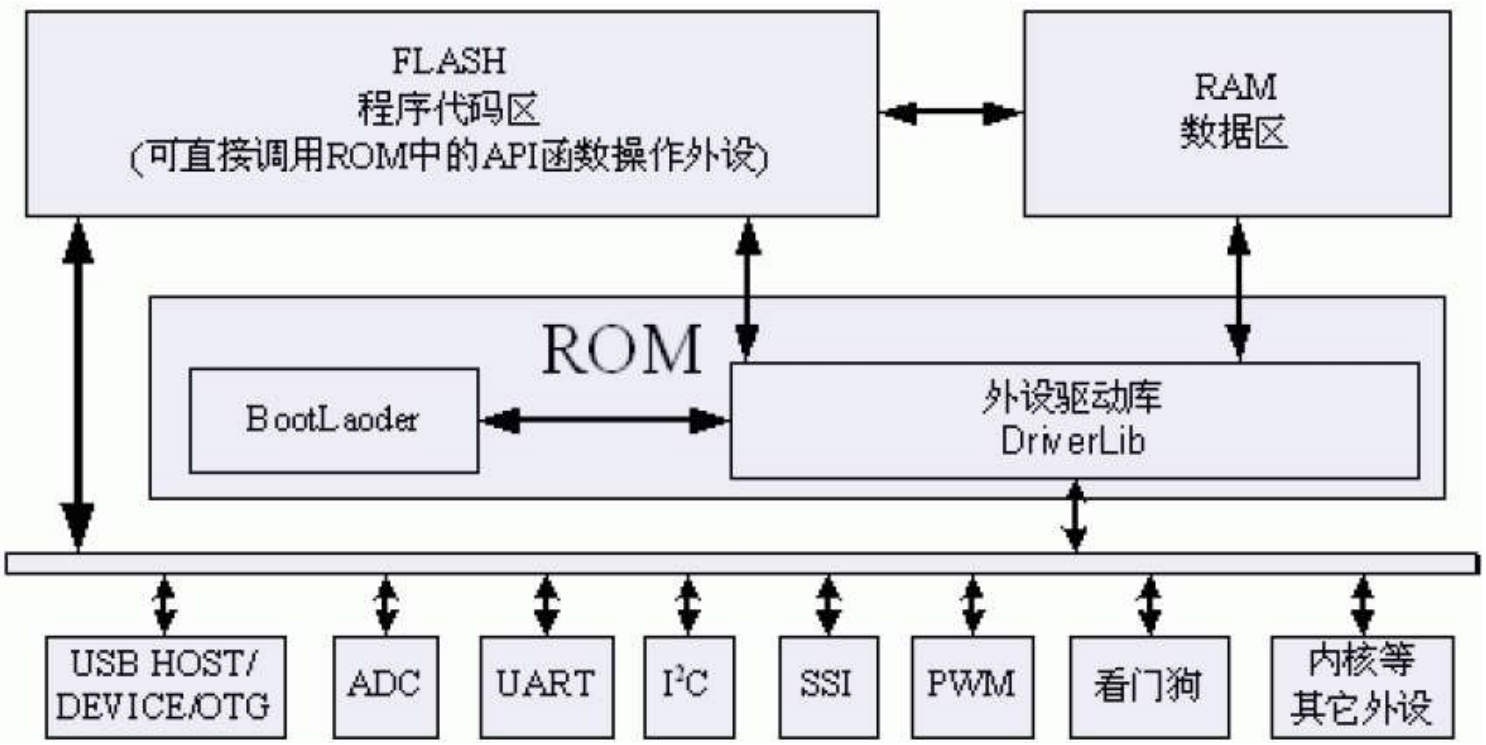
- 处理器内核
  - 32-bit ARM Cortex-M4F 内核
  - 主频可达120MHz，150DMIPS速度
  - IEEE754兼容的单精度浮点单元FPU
  - ARM Cortex SysTick 24-bit定时器
  - NVIC中断管理
  - MPU存储保护单元
  - 完全硬件调试（JTAG或SWD）



- 片上存储器（On-Chip Memory）
  - 256KB 单周期 SRAM
    - SRAM存储器的映射地址为**0x2000 0000**，可执行位带操作
  - 1024KB Flash存储器，最高速度50M
    - Flash存储器的映射地址为**0x0000 0000**，可应用Flash存储器保护
  - 6KB EEPROM
  - 16MB 内部ROM预装TivaWare软件
    - ROM存储器的映射地址为**0x0200 0000**
    - 预装TivaWare引导装载程序（Boot Loader）
    - 预装TivaWare外设驱动库
    - AES高级加密标准，内嵌密码表
    - CRC校验算法



- 片上存储器结构示意


FLASH
程序代码区
(可直接调用ROM中的API函数操作外设)
RAM
数据区
ROM
BootLaoder
外设驱动库
DriverLib
USB HOST/
DEVICE/OTG
ADC
UART
I²C
SSI
PWM
看门狗
内核等
其它外设

- SRAM中的位带别名
  - 在位带使能的处理器中，存储器映射的特定区域（SRAM和外设空间）能够使用地址别名，在单个原子操作中访问位带位

    SRAM位带基址：0x2200 0000
  - 位带别名的地址计算：

    位带别名地址 = 位带基址+ (字节偏移量* 32) + (位偏移* 4)

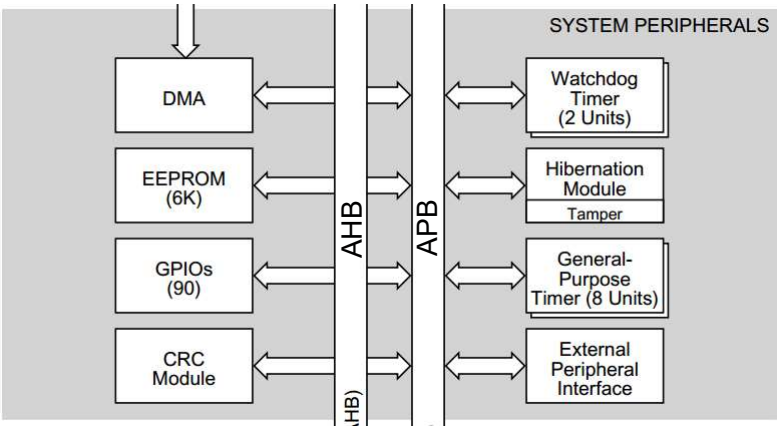
例：地址0x20001000的第3位对应当位带别名地址为：

0x2200 0000 + (0x1000 * 32) + (3 * 4) = 0x2202 000C

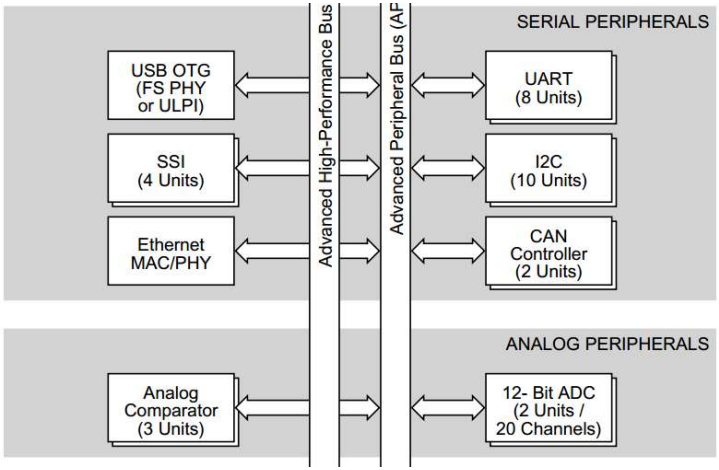
- FLASH程序代码区
  - Flash存储器配置为四组16K x 128位（总共4*256KB）
  - Flash内存块可以标记为只读或只执行，可以提供不同的级别的代码保护。
  - TM4C1294NCPDT微控制器提供两组指令预取缓冲区，每组为2×256位，组成 4×256位的指令预取缓冲区，以提高执行性能

- ROM区的引导装载程序
  - TivaWare 引导装载程序（Boot Loader）作为初始程序加载器和初始化应用程序固件的更新机制，可用来将代码下载到设备的Flash存储器中
  - Reset启动配置
    - 如果 Flash 地址0x0000.0004的数据是有效的，并且BOOTCFG 寄存器的EN位被设置，那么堆栈指针 (MSP) 装载 0x0000.0000的数据，程序计数器(PC) 装载地址0x0000.0004的数据，用户应用程序开始执行
    - 否则 MSP 和 PC 从ROM获取，执行ROM的引导装载
- ROM区的TivaWare外设驱动库
  - TivaWare外设驱动库包含初始化和控制片上外设的API函数库，可以供应用程序调用。与ROM库函数调用相关的文件是driverlib/rom.h
  - 外设驱动库的功能描述详见 “TivaWare™ Peripheral Driver Library User's Guide”

- 系统外设
  - 系统控制和时钟，内置16M高精度振荡源
  - μDMA控制器
  - 2路看门狗定时器
  - 低功耗休眠模块
  - 8个32个定时器，实时时钟兼容
  - 至多90个GPIO脚，可设为2, 4, 8, 10,或12-mA驱动力
  - 外部外设接口EPI
    - 8/16/32位可共享并行总线用于与外部设备相连
    - 支持SDRAM，SRAM，NOR FLASH
    - 可以以并行接口形式与FPGAs、CPLDs相连

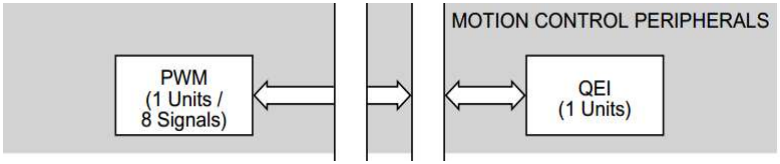


- 高级串行接口设备
  - 10/100M 以太网MAC和PHY接口，IEEE1588 PTP硬件支持
  - 2路CAN2.0 A/B控制器
  - USB2.0 OTG/Host/Device
  - 8路UARTs，支持IrDA，9-bit和ISO7816
  - 10路I2C模块四种传输速度
  - 4路同步串行接口QSSI



- 模拟量设备模块
  - 三个模拟比较器，片上电压调节器
  - 两路12位精度ADC，最多20个模拟量输入加1个内部温度传感器

- 高级运动控制设备
  - 8路高级PWM输出
  - 1个正交编码器输入（QEI）



- 其他
  - JTAG (IEEE标准调试口) 和SWD (Serial Wire Debug) 调试接口
  - 128-PIN TQFP封装
  - 工业应用温度范围-40°C ~85°C

TM4C1294NCPDT MCU存储器映射

Start	End	Description
Memory		
0x0000.0000	0x000F.FFFF	On-chip Flash
0x0010.0000	0x01FF.FFFF	Reserved
0x0200.0000	0x02FF.FFFF	On-chip ROM (16 MB)
0x0300.0000	0x1FFF.FFFF	Reserved
0x2000.0000	0x2006.FFFF	Bit-banded on-chip SRAM
0x2007.0000	0x21FF.FFFF	Reserved
0x2200.0000	0x2234.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000
0x2235.0000	0x3FFF.FFFF	Reserved
Peripherals		
0x4000.0000	0x4000.0FFF	Watchdog timer 0
0x4000.1000	0x4000.1FFF	Watchdog timer 1
0x4000.2000	0x4000.3FFF	Reserved

- 存储器映射定义（inc/hw_memmap.h）

```
#define FLASH_BASE                0x00000000  // FLASH memory
#define SRAM_BASE                 0x20000000  // SRAM memory

#define WATCHDOG0_BASE            0x40000000  // Watchdog0
#define WATCHDOG1_BASE            0x40001000  // Watchdog1
#define GPIO_PORTA_BASE           0x40004000  // GPIO Port A
#define GPIO_PORTB_BASE           0x40005000  // GPIO Port B
#define GPIO_PORTC_BASE           0x40006000  // GPIO Port C
#define GPIO_PORTD_BASE           0x40007000  // GPIO Port D
#define SSI0_BASE                 0x40008000  // SSI0
#define SSI1_BASE                 0x40009000  // SSI1
#define SSI2_BASE                 0x4000A000  // SSI2
#define SSI3_BASE                 0x4000B000  // SSI3
#define UART0_BASE                0x4000C000  // UART0
#define UART1_BASE                0x4000D000  // UART1
```

第2章 S800硬件系统与嵌入式开发

2.1 TM4C1294NCPDT微控制器（ref.B-chapter1）

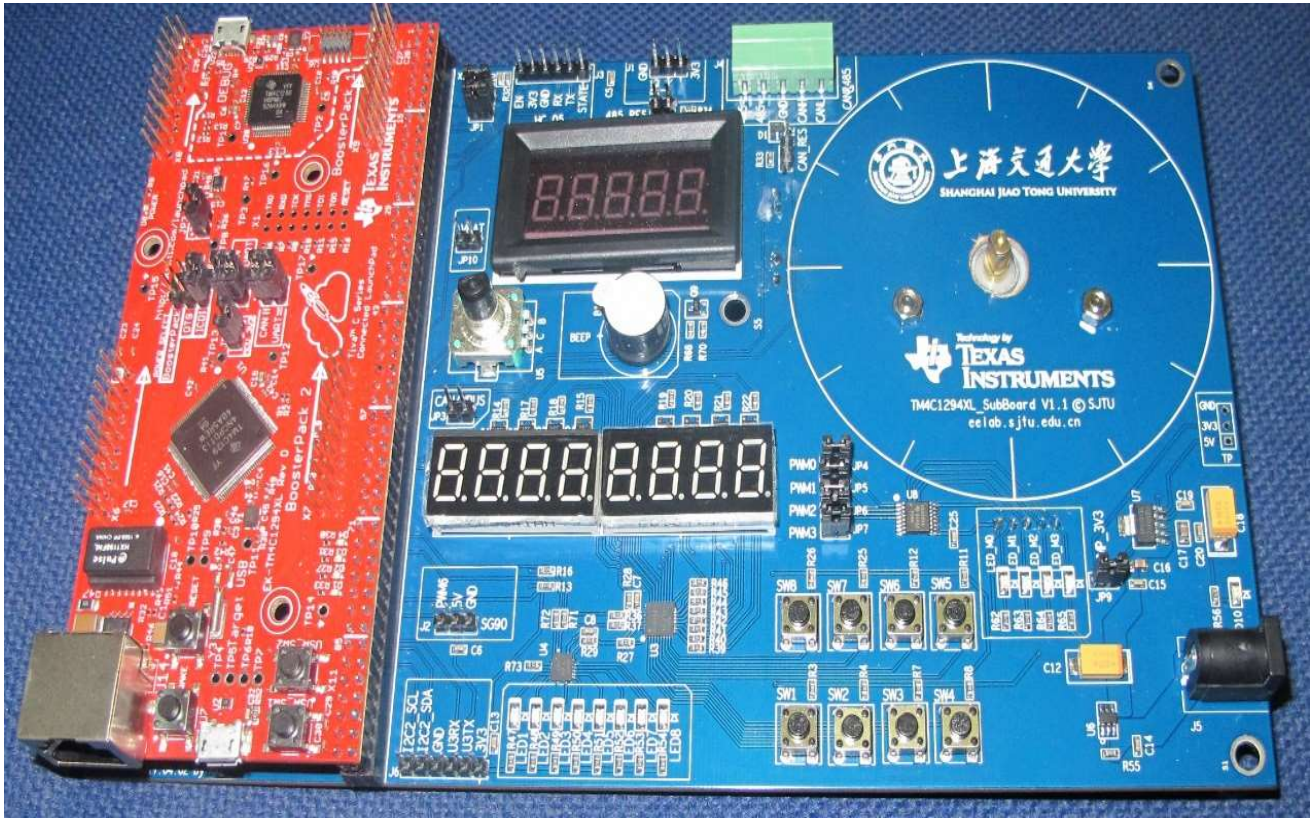
2.2 基于TM4C1294 MCU的S800实验板（ref.C）

2.3 嵌入式软件开发（ref.A-chapter2，20）

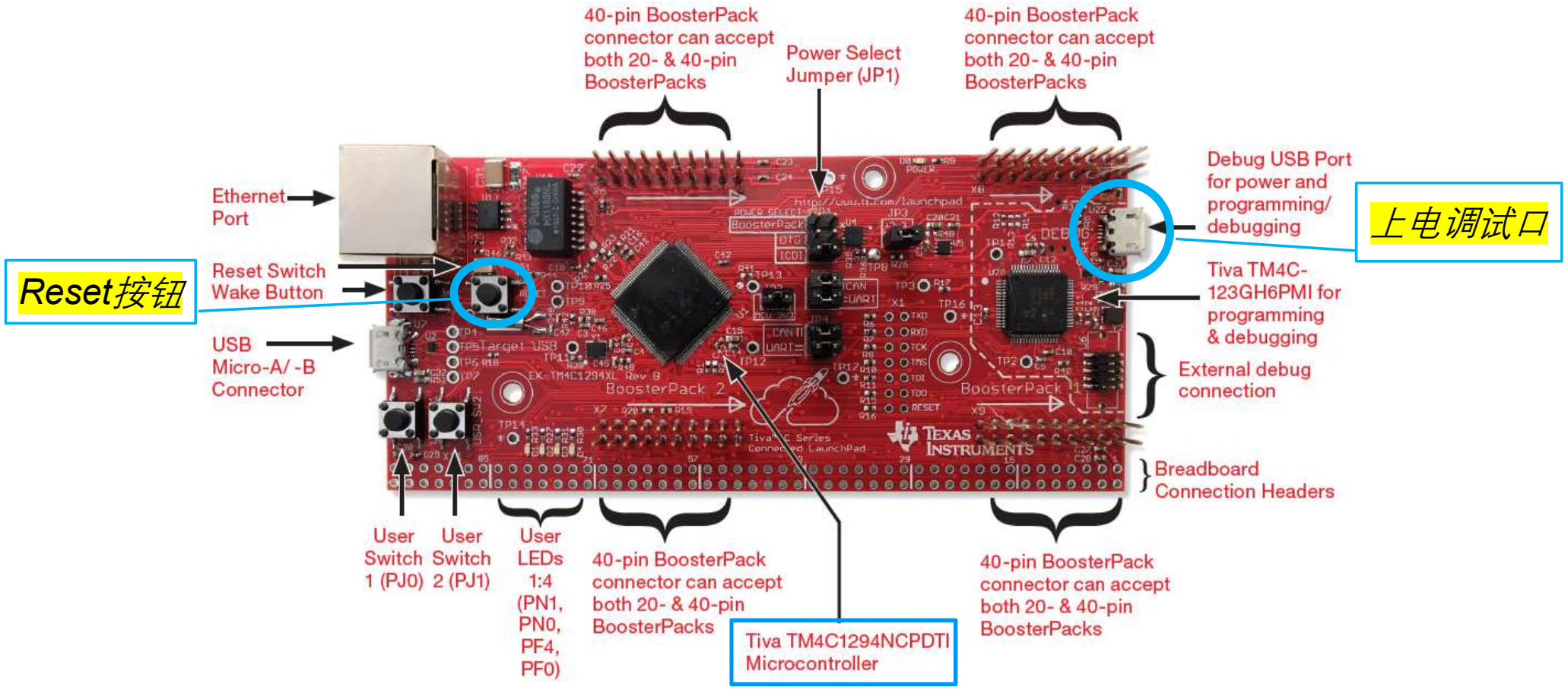
2.4 S800的嵌入式软件开发（ref.A-chapter15）

2.2 基于TM4C1294 MCU的S800实验板

- S800实验板 = TM4C1294XL(红) + TM4C1294XL_SUB(蓝)



TM4C1294XL评估板



TM4C1294XL_SUB扩展功能板

- 静电ESD保护及过电流负载保护
- 电流显示
- 多路可选电源输入，包括
  - DC5V
  - MICRO-USB5V输入
  - MICRO-USB OTG 5V输入
- I2C扩展GPIO
  - 8位共阴数码管
  - 8位输入按键
  - 8位共阴LED
- USART- RS485总线串行接口
- CAN总线串行接口
- 5V直流有刷电机或步进电机接口
- PWM输出
- DAC输出
- 外部模拟量输入
- 可调电位器模拟量输入
- 蜂鸣器
- SD卡接口
- QEI数字电位器接口
- 蓝牙模块接口
- 舵机控制接口
- RTC备用电池

第2章 S800硬件系统与嵌入式开发

2.1 TM4C1294NCPDT微控制器（ref.B-chapter1）

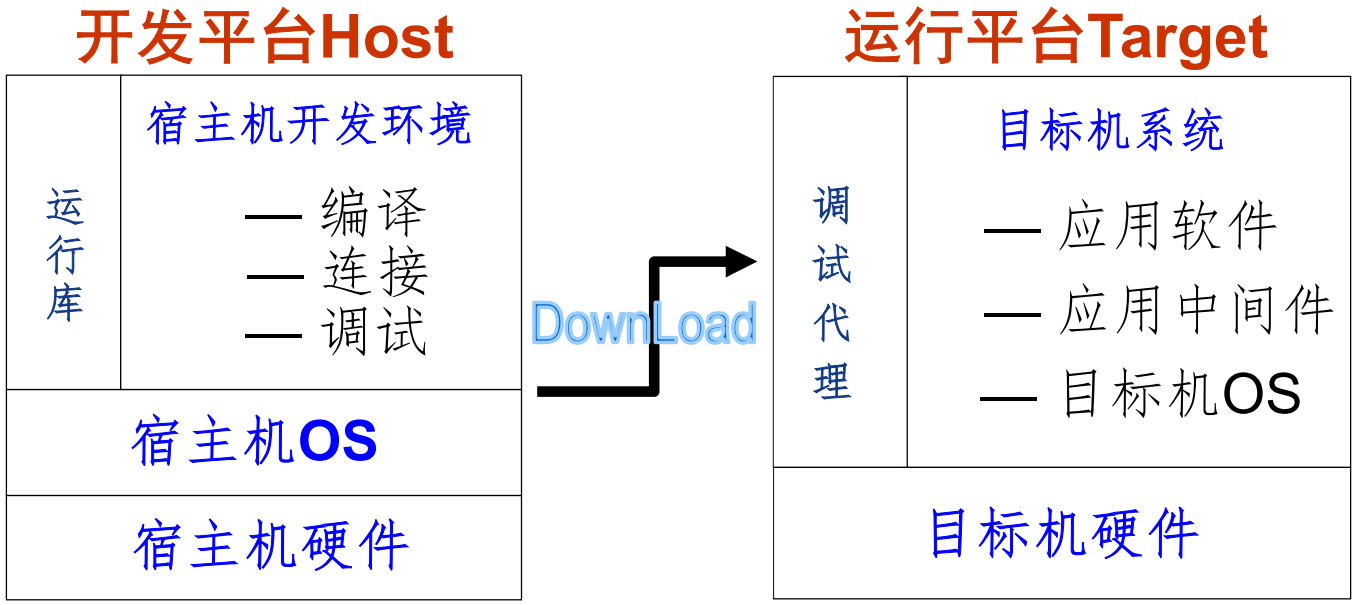
2.2 基于TM4C1294 MCU的S800实验板（ref.C）

2.3 嵌入式软件开发（ref.A-chapter2，20）

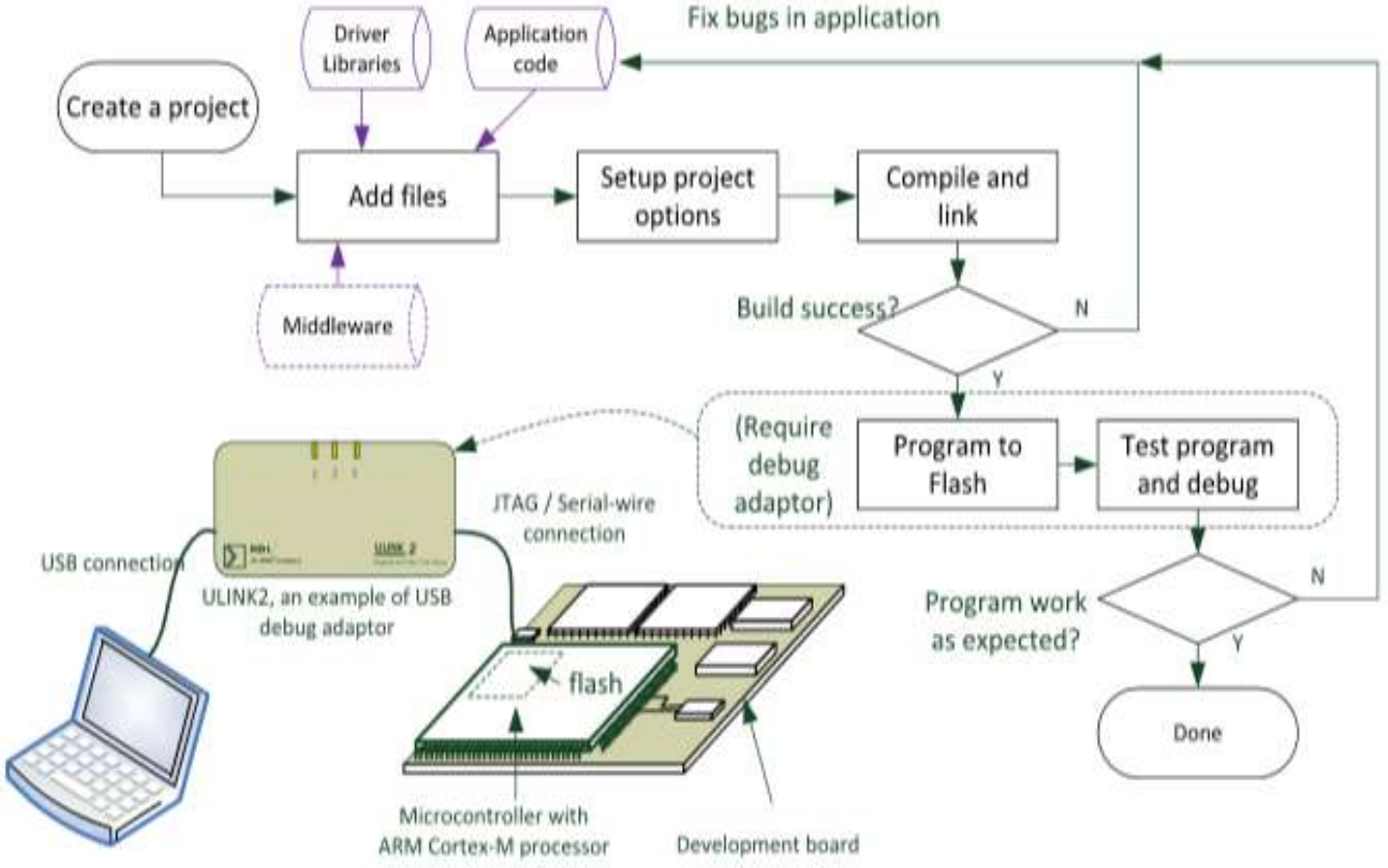
2.4 S800的嵌入式软件开发（ref.A-chapter15）

2.3 嵌入式软件开发

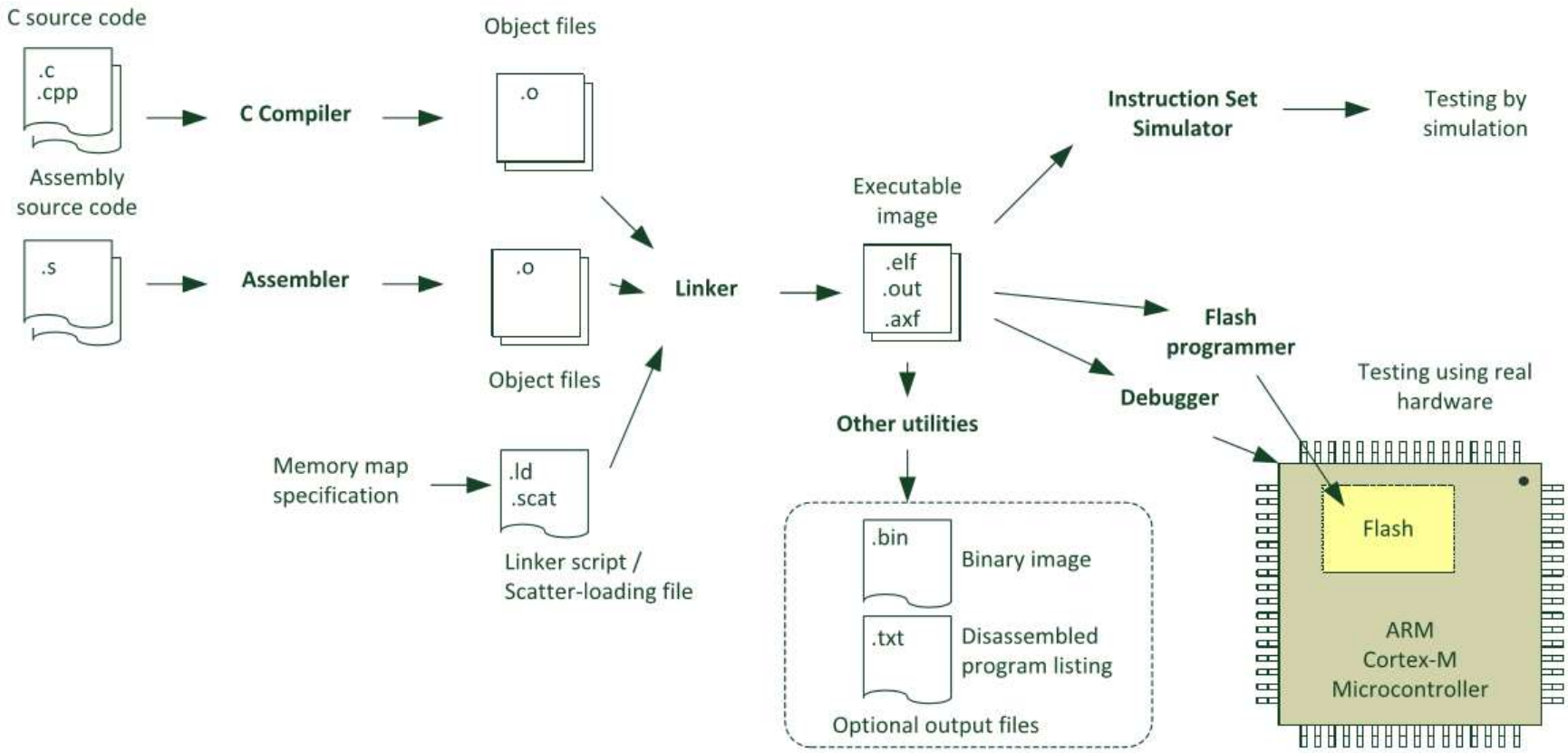
- 宿主机/目标机的交叉开发模式
  - 嵌入式软件属跨平台开发，即开发平台的处理器和运行平台的处理器不是同一类型，因此需要一个交叉开发环境，进行编译、链接和调试



嵌入式软件开发流程


Driver Libraries
Application code
Fix bugs in application
Create a project
Add files
Setup project options
Compile and link
Middleware
Build success?
N
Y
(Require debug adaptor)
Program to Flash
Test program and debug
JTAG / Serial-wire connection
USB connection
ULINK2, an example of USB debug adaptor
Program work as expected?
N
Y
Done
flash
Microcontroller with ARM Cortex-M processor
Development board

使用ARM工具链的程序编译过程

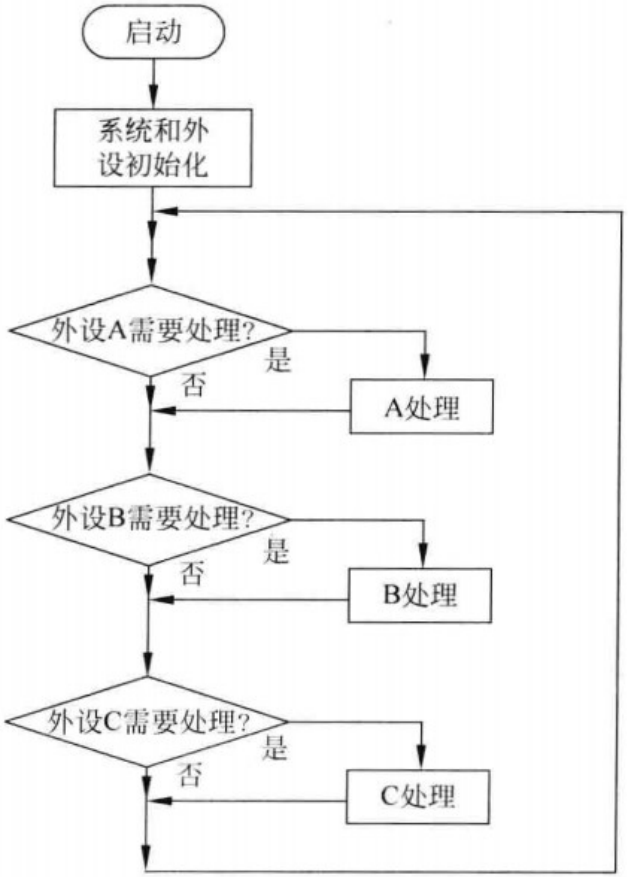


ARM嵌入式程序框架

- ARM体系结构支持汇编语言与C/C++/其他高级语言的混合编程
  - 一个ARM项目（project）由多个文件组成
    - 扩展名为.s的汇编语言源文件、.c/.cpp的C/C++语言源文件、.h的头文件等
  - 启动代码用汇编语言完成
    - 硬件初始化，如设定CPU工作状态、中断向量、RAM控制参数等
  - 主要的编程任务用C/C++完成
    - 用户程序的主函数名字必须为main

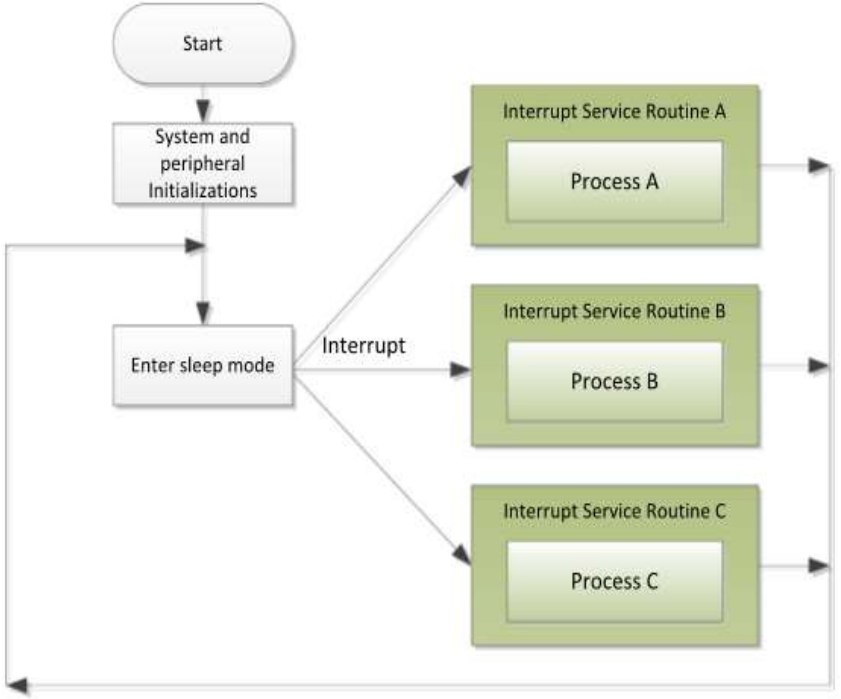
C程序结构

- 轮询方式：主程序+循环轮询
  - 主程序中除了一些初始化设置外，还包含一个顺序执行的无限循环程序段，用于检测和调用各个功能模块的处理



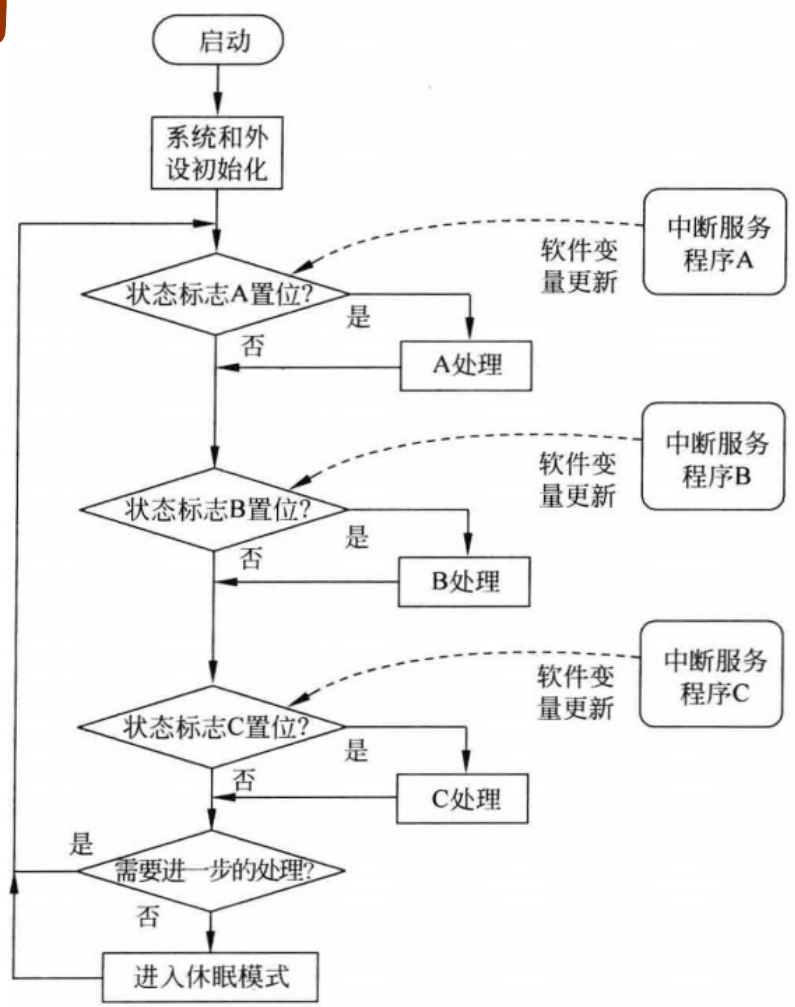
C程序结构

- 中断驱动：主程序+中断服务程序
  - 主程序进行初始化设置后进入睡眠状态，按中断的优先级实时响应各处理模块的中断请求
  - 各模块的处理在中断中完成



C程序结构

- 前后台系统：中断和轮询结合
  - 后台程序：主程序初始化设置后进入一个无限循环程序段，按顺序检查并调用各个功能模块的处理；
  - 前台程序：各模块的中断服务程序；
  - 通常对实时性要求较高的系统，中断服务程序中仅仅标记事件，事件的处理则由后台程序调度完成



前后台程序框架

- 后台程序：

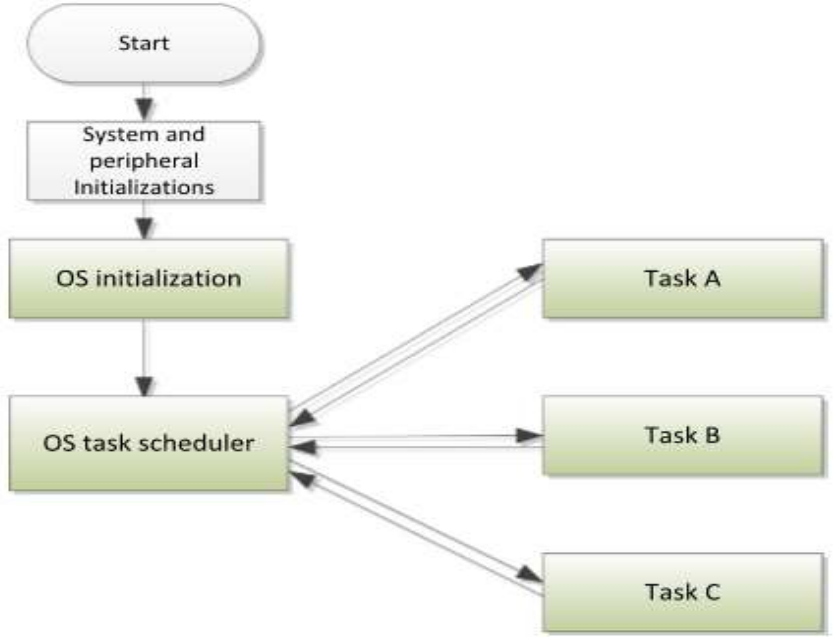
```
main()
{  Init(); //初始化
   FlagA = FlagB = FlagC = 0;
   while (1) {
      if (FlagA)  taskA();
      else if (FlagB)  taskB();
      else if (FlagC) taskC();
      …
   }
}
```

- 前台程序：

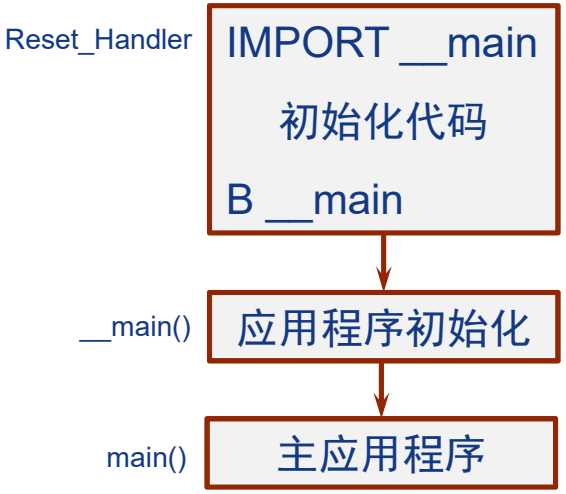
```
void taskA_Handler(void)
{     清中断请求标志;
      FlagA = 1;
}
void taskB_Handler(void)
{     清中断请求标志;
      FlagB = 1;
}
void taskC_Handler(void)
{     清中断请求标志;
      FlagC = 1;
}
```

C程序结构

- 实时多任务系统：任务调度主程序+子任务
  - 主程序中进行系统初始化设置，创建各个子任务，并启动任务调度器管理子任务的运行
  - 实时操作系统RTOS可用于处理任务调度



C/C++与汇编语言的混合编程

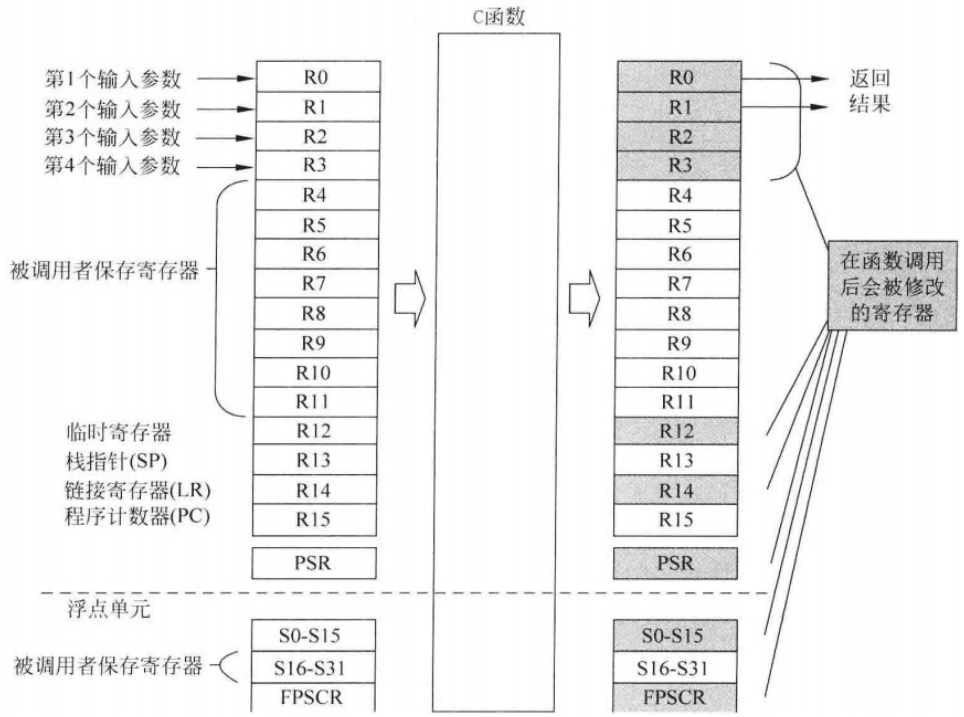


汇编程序中使用IMPORT伪操作声明被调用的C/C++函数，并通过B或BX指令实现C函数的调用

- C/C++和汇编程序间相互调用
  - C/C++中内嵌汇编
  - C/C++程序调用汇编程序
  - 汇编程序调用C/C++程序
- C/C++程序和汇编程序调用必须遵守AAPCS规则

AAPCS规范

- AAPCS：ARM Architecture Produce Call Standard
  - 函数调用和返回中寄存器的使用
    - 前4个参数通过R0～R3传递，
    - 其他参数通过数据堆栈传递
    - 32位的返回值通过R0返回；
    - 64位的返回值通过R0和R1返回



C语言程序中使用内嵌汇编

- 内联汇编

  __asm ("ARM指令[；ARM指令]");　　或者

  __asm volatile (“ARM指令[；ARM指令]”);　加volatile可禁止编译器做优化

  如：__asm volatile(“cpsie i”); /* 使能 IRQ 中断, PRIMASK置0*/

- 嵌入汇编

```
__asm void SetFaultMask (unsigned int new_value)
{
    MSR     FAULTMASK, R0      // 参数 new_value 通过R0传递
    BLX     LR      // 返回调用程序
}
```

C语言程序中调用汇编程序

- 在C程序中使用extern关键词声明要调用的汇编子程序，调用方式如同调用普通C函数
  - C源程序

```
extern  int  Sum(int num); /*声明函数Sum为外部函数*/
int  main  ()
{ int s;
   s = Sum(10);          //调用汇编Sum, 参数由R0传送
                         //返回值也通过R0传回
}
```

C语言程序中调用汇编程序

- 在汇编程序中用EXPORT声明可被调用的子程序
  - 汇编源程序

```
        AREA    |.text|, CODE, READONLY    ;代码段
        THUMB
        EXPORT Sum      ; 声明子程序可被外部引用
Sum
        MOV     R1, #0    ; 初始化运算结果的值，参数在R0中
loop
        ADD     R1, R0    ; R1 = R1+ R0
        SUBS    R0, #1    ; R0 = R0 − 1，并且根据结果更新标志
        BNE     loop      ; if (R0≠0) 转 loop;
        MOV     R0, R1    ; 运算结果送R0
        BLX     LR        ; 返回
        END               ;汇编程序结束
```

汇编程序中调用C语言程序

- 汇编程序调用C程序需要做到：
  - 汇编语言程序中使用IMPORT或EXTERN伪操作声明要调用的C程序函数
  - 汇编程序通过B或BL指令来调用该C函数
    - 如果有参数，则调用时需正确设置入口参数

如：

```
IMPORT  __main
B   __main
```

汇编程序中调用C语言程序

- C语言函数

```
int  maxnum(int a, int b)
{
    return (a>b ? a : b);
}
```

- 汇编源程序

```
    AREA    |.text|, CODE, READONLY          ;代码段
    IMPORT  maxnum      ; 引入外部函数maxnum
    ; 初始化参数寄存器 R0->a, R1->b
    MOV     R0, #10
    MOV     R1, #20
    BL      maxnum              ; 调用C函数，返回的结果在R0中
    END
```

附：C语言回顾

- 无名字空间
- 无 bool类型、类类型class
- 变量定义必须放在语句之前
- volatile关键字
  - 告诉编译器该变量无持久性是随时可能发生变化的，每次使用时必须从它的地址中读取，禁止编译器做优化
  - volatile unsigned char sysflag;

附：C语言位运算

作用于整型、字符型数据的每个二进制位

- 位逻辑运算： & | ^ ~
- 位移运算： << 和 >>
  - 左移相当于乘法运算。左移一位相当于乘 2
  - 高位左移后溢出被舍弃，低位补以0
  - 右移相当于除法运算。右移一位相当于除 2
  - 无符号数右移时高位补0，有符号数高位补符号位

如：x <<= 2; ⇒ x=x<<2; ⇒ x=x*4;

运算符	含义
&	按位与
|	按位或
^	按位异或
~	取反
<<	左移
>>	右移

附：C语言常用字符串处理函数

- 头文件：#include <string.h>
- strcat()函数：连接字符串，用法：strcat(str1, str2);
- strcpy()：字符串拷贝，用法：strcpy(str1, str2);
- strlen()：求字符串的长度，用法：int strlen(str);
- strchr()：字符查找，用法：char * (const char *s, int c);
- strcmp()：字符串比较，用法：int strcmp(str1, str2);
- strncmp()：比较前n个字符，用法：int strcmp(str1, str2, n);

附：C语言常用字符处理函数

- toupper(), tolower()：大小写转换
  - 头文件：#include <ctype.h>
  - int toupper(int c)；把字符c转换成大写

    如：char c1; c1 = toupper('a');　　//c1为'A'
  - int tolower(int c)；把字符c转换成小写

    如：char c2; c2 = tolower(c1);　　//c2为'a'

附：C语言格式化字符串输入/输出函数

- 头文件：#include <stdio.h>
- sprintf()：格式化字符串输出
  - int sprintf(char *buffer, const char *format [, arg] ...);
  - 功能：把格式化的数据写入某个字符串buffer
  - char str[20];
  - sprintf(str, “%d”, 123); //str为"123"
  - sprintf(str, “%-4d%d”, 123, 4567); //str为"123 4567"
  - 若 min=2, sec=15;
  - sprintf(str, “#%04d# “,min*100+sec); //str为"#0215#"

附：C语言格式化字符串输入/输出函数

- 头文件：#include <stdio.h>
- sscanf()：格式化字符串输入
  - int sscanf(char *buffer, const char *format, arg ...);
  - 功能：把字符串buffer中的数据按格式写入参数arg
  - //假设str存放的字符串为"Tel:1234-567"
  - sscanf(str, “%3s:%4d%c%3d”, s,&n1,&ig,&n2);
  - 则s为“Tel”，‘:’被忽略，n1=1234，ig=‘-’，n2=567

附：C语言格式控制符

- 格式控制符
  - %d：以带符号的十进制形式输出整数
  - %o：以八进制无符号形式输出整数
  - %x：以十六进制无符号形式输出整
  - %u：以无符号十进制形式输出整数
  - %c：以字符形式输出，只输出一个字符
  - %s：输出字符串
  - %f：以小数形式输出浮点数，隐含输出六位小数
  - %e：以指数形式输出实数

附：C语言格式控制符

- 格式控制符的修饰符
  - l：加在格式符d，o，x，u前面，用于长整型整数，如%ld。加在格式符f前，用于双精度浮点型，如%lf
  - m.n（m和n代表正整数）：m表示数据最小宽度，对实数n表示输出的小数位数；对字符串n表示截取的字符个数，如%4d，%5.3f，%4.2s
    - 0m：m含义同上，0表示不足位用0填充，如%04d
  - -：输出的数字或字符在域内向左靠，如%-5s

第2章 S800硬件系统与嵌入式开发

2.1 TM4C1294NCPDT微控制器（ref.B-chapter1）

2.2 基于TM4C1294 MCU的S800实验板（ref.C）

2.3 嵌入式软件开发（ref.A-chapter2，20）

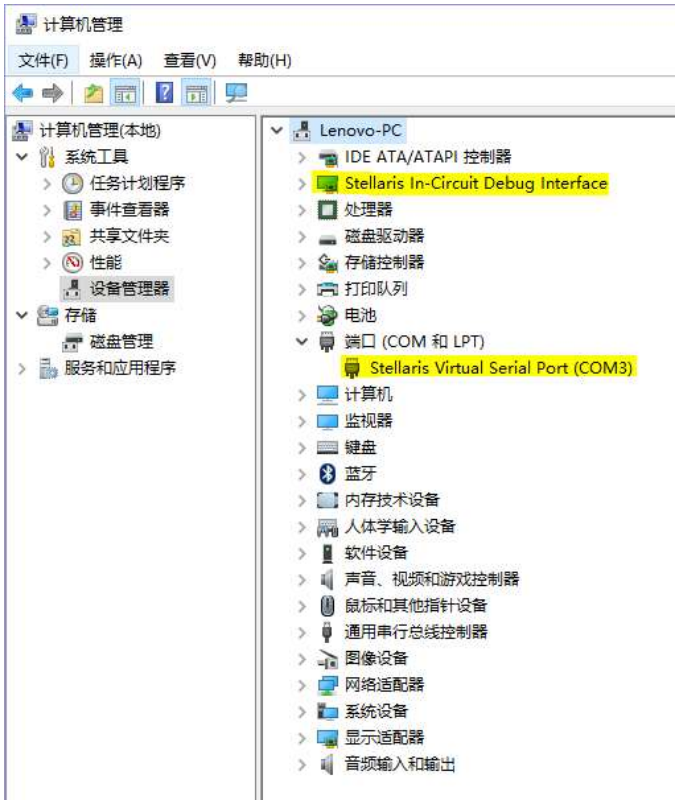
2.4 S800的嵌入式软件开发（ref.A-chapter15）

Keil RealView MDK简介

- KEIL RealView Microcontroller Development Kit
  - 简称RealView MDK 或RVMDK
  - 源自德国Keil公司，现被ARM公司收购，是目前ARM公司最新一款针对各种嵌入式处理器的软件开发工具
- Keil RealView MDK开发工具包包括：
  - μVision集成开发环境(IDE)、调试器、仿真器、编译工具
  - RTX Real-Time Kernel
  - 针对各种微控制器的详细启动代码
  - Flash 编程算法等

Keil和驱动程序的安装

- 安装Keil：MDK522.exe
- 安装LEGACY：MDKCM522.exe
- 安装DFP包：Keil.TM4C_DFP.1.1.0
- 安装TivaWare驱动及样例程序：SW-EK-TM4C1294XL-2.1.4.178.exe
- 安装调试驱动：解压Stellaris ICDI Drivers，将S800开发板通过MICRO-USB线与电脑连接。进入电脑的设备管理器，更新驱动程序
  - ICDI在线调试工具和虚拟串口（如图）



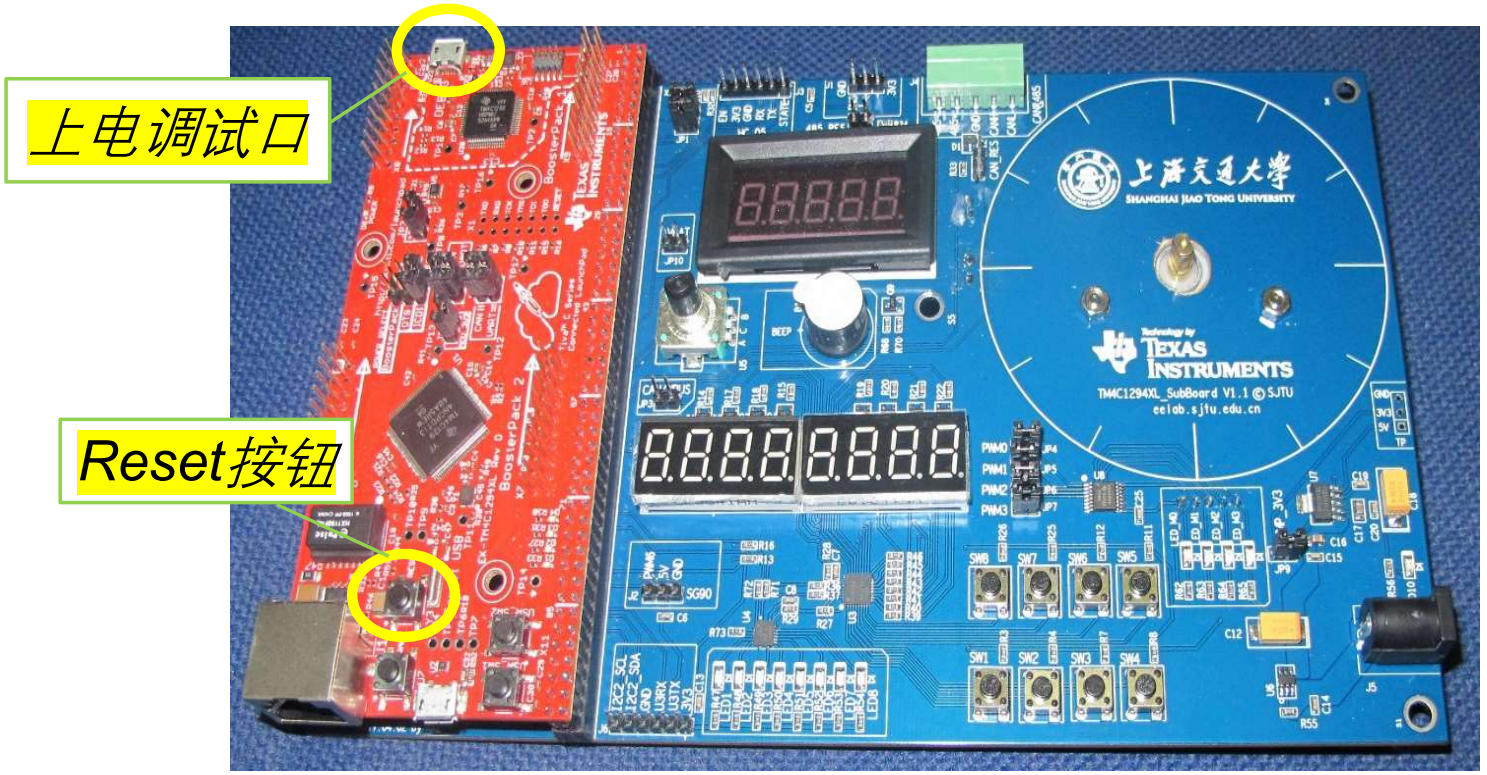
KEIL入门教程

本章所给示例基于Keil RealView MDK V5.22

- 连接实验板
- 新建一个项目
- 添加带有main函数的C文件
- 添加driverlib.lib文件
- 项目配置
- 编译和下载程序
- 调试和运行程序

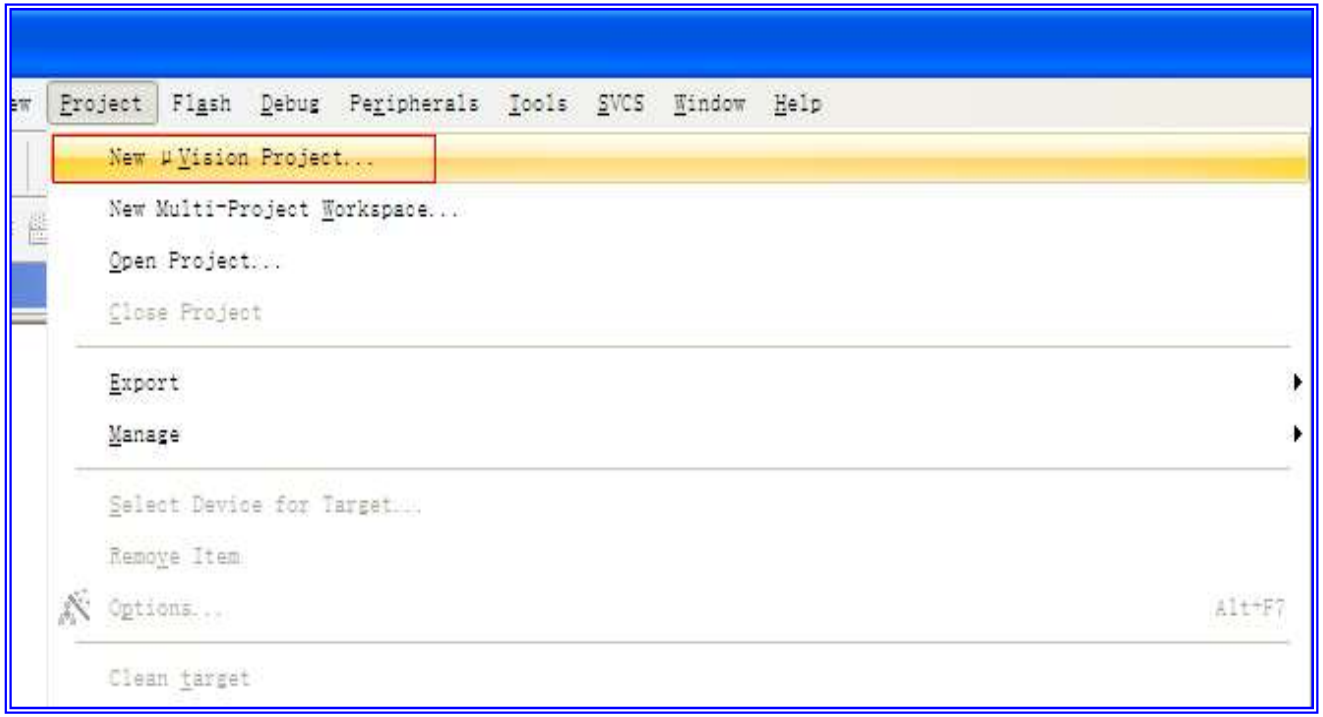
步骤一：连接实验板

- 将MICRO-USB数据线一端接电脑，一端接TM4C1294XL的上电调试口



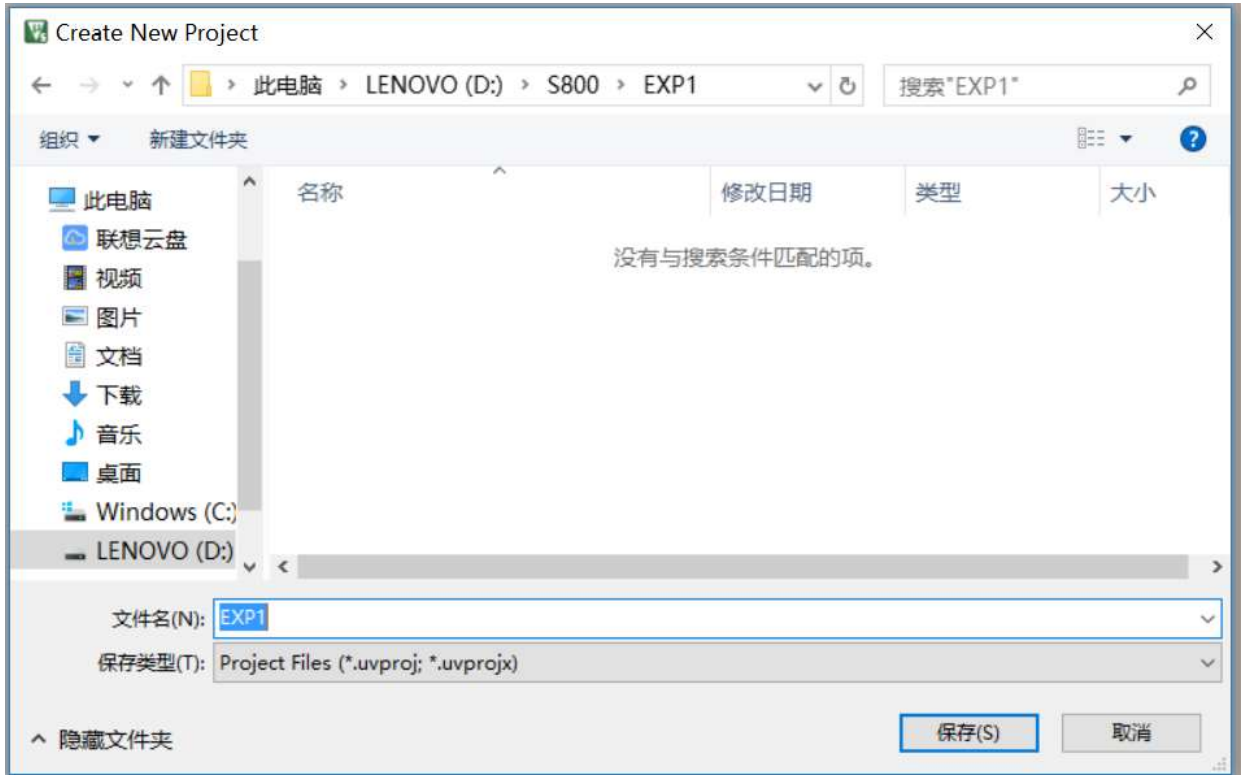
步骤二：新建一个项目

- 启动Keil uVision5，关闭打开的其它项目。选择Project→New uVision Project...



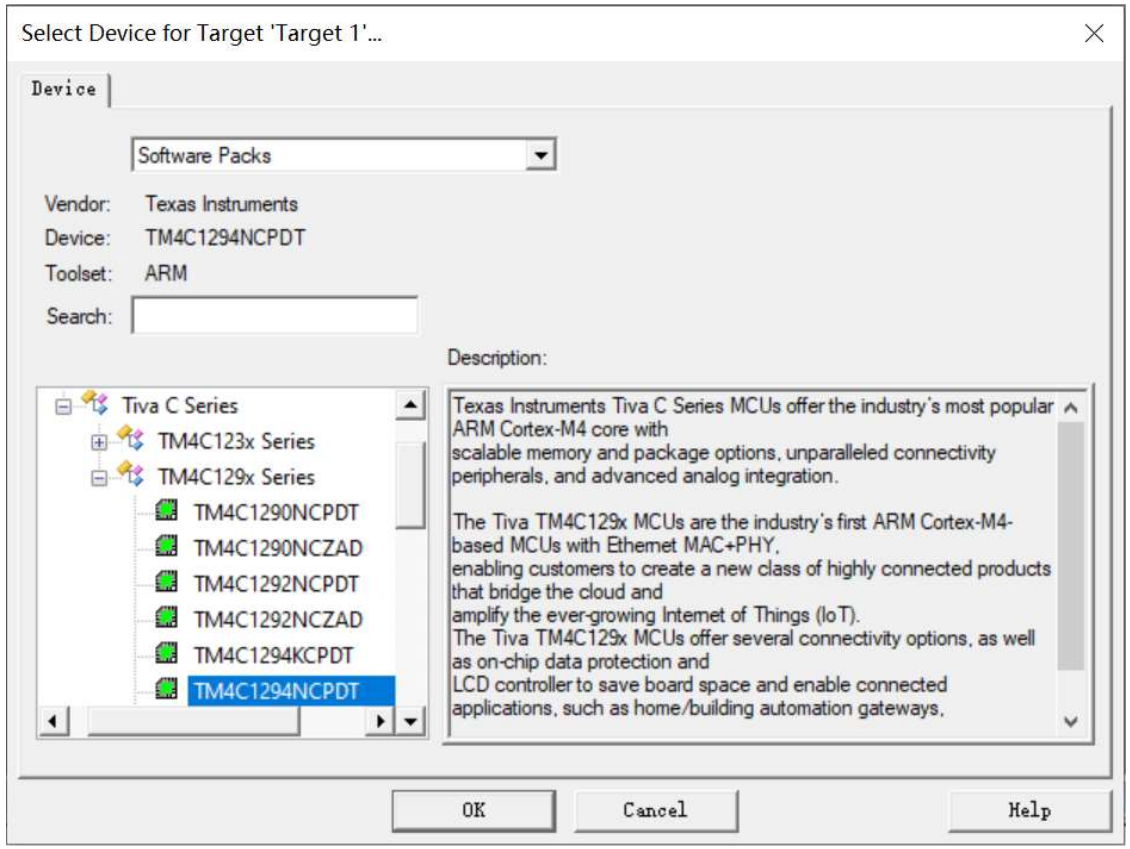
确定项目保存路径和项目名

- 选择文件夹D:\S800\EXP1，并把新建项目保存在该文件夹中，项目名设为EXP1。点击“保存”



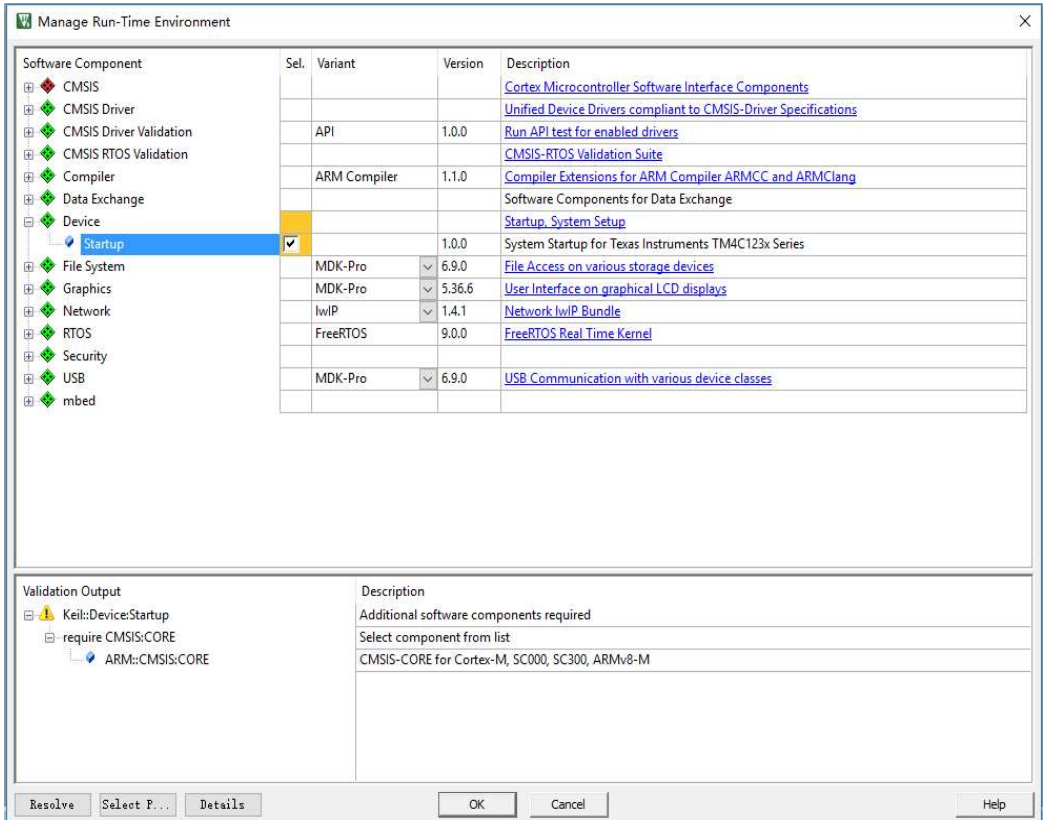
为项目选择对应的目标处理器

- 选择芯片：TM4C1294NCPDT →OK



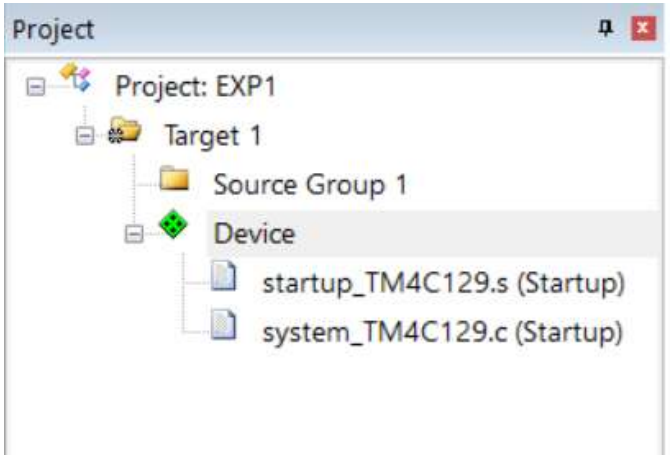
添加启动代码

- 生成启动代码：展开Device选项，勾选Startup → OK



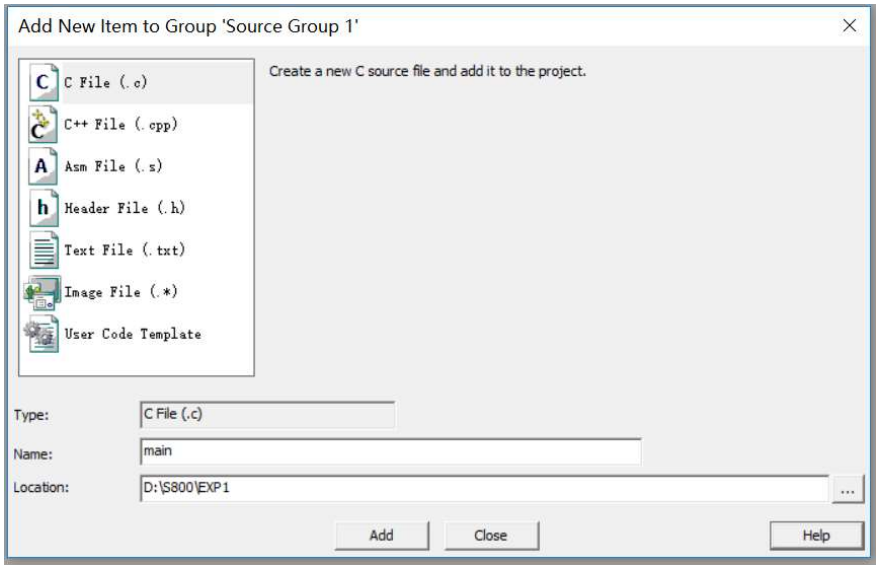
新项目的结构

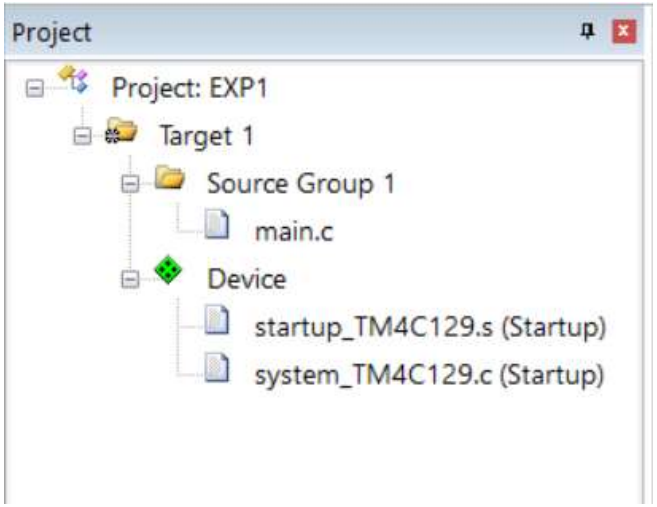
- 现在我们有了一个项目EXP1，含有
  - Source Group 1，用来放置用户的源文件，目前为空；
  - 设备目录Device，含Startup_TM4C129.s和System_TM4C129.c两个文件
    - Startup_TM4C129.s：配置了堆栈和中断函数名称以及从复位到main函数前的处理过程
    - System_TM4C129.c：系统初始化，配置了默认时钟



步骤三：添加带main函数的C文件

- 右击Source Group 1，选择Add New Item to Group ‘Source Group 1’，取名 main.c，添加。（也可以在当前目录下先建一个main.c文件，再右击Source Group 1，选择Add Existing Files to Group ‘Source Group 1’，找到文件添加）





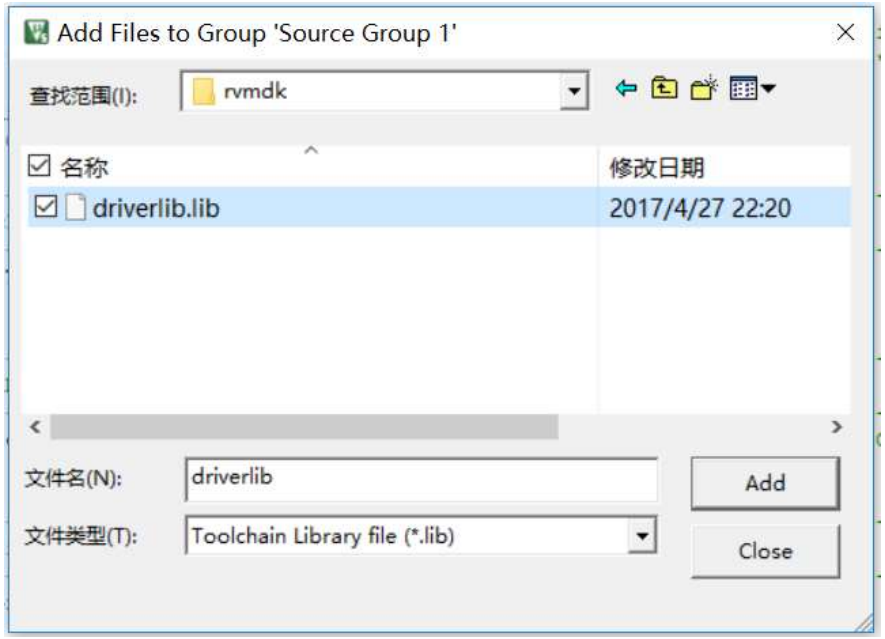
编辑main.C文件

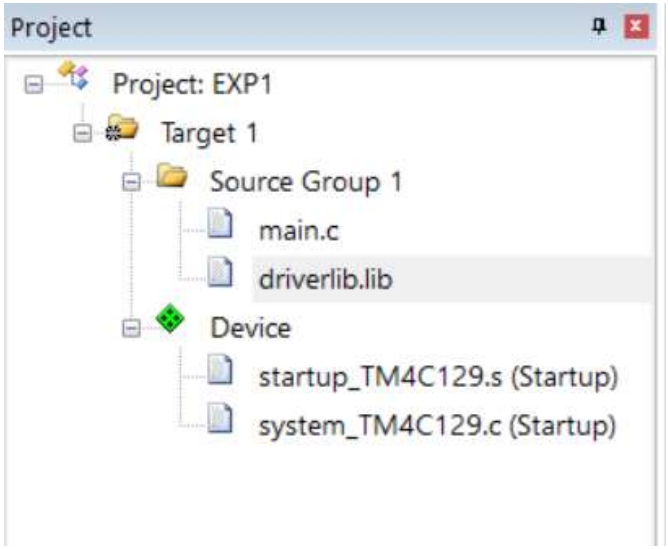
- 双击main.c，在文件窗口键入程序代码：

```
#include <stdint.h>
#include <stdbool.h>
#include "hw_memmap.h"
#include "debug.h"
#include "gpio.h"
#include "hw_types.h"
#include "pin_map.h"
#include "sysctl.h"

int main(void)
{
    uint32_t ui32Loop;

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //Enable PortF
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0); //Set PF0 as Output pin
    while(1)  {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_PIN_0); // Turn on the LED
            for(ui32Loop = 0; ui32Loop < 800000; ui32Loop++){}; // Delay
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0x0); // Turn off the LED.
            for(ui32Loop = 0; ui32Loop < 800000; ui32Loop++){}; // Delay
   }
}
```

步骤四：添加driverlib.lib文件

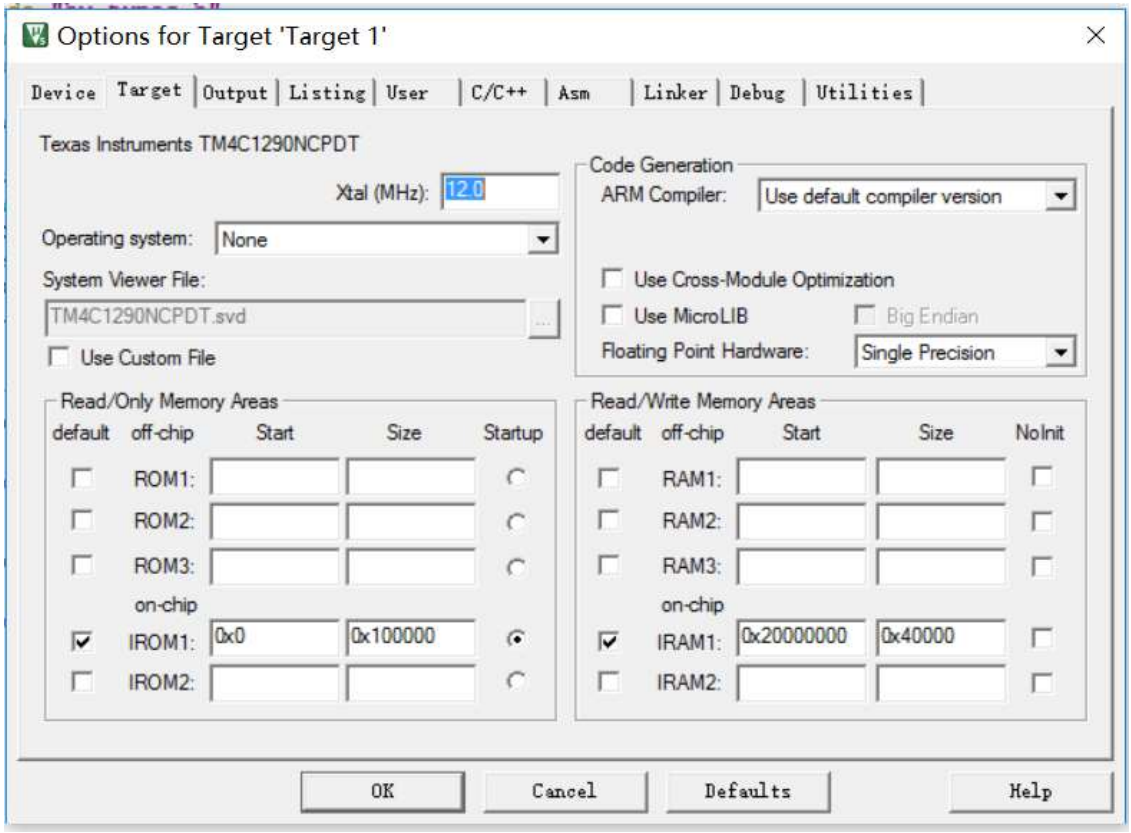
- 为了使用 TivaWare 驱动程序，需要在项目中添加库文件driverlib.lib
- 右键Source Group 1文件夹并选择Add Existing Files to Group ‘Source Group 1’，浏览 C:\ti\TivaWare_C_Series-2.1.4.178\driverlib\rvmdk，选中 driverlib.lib文件。





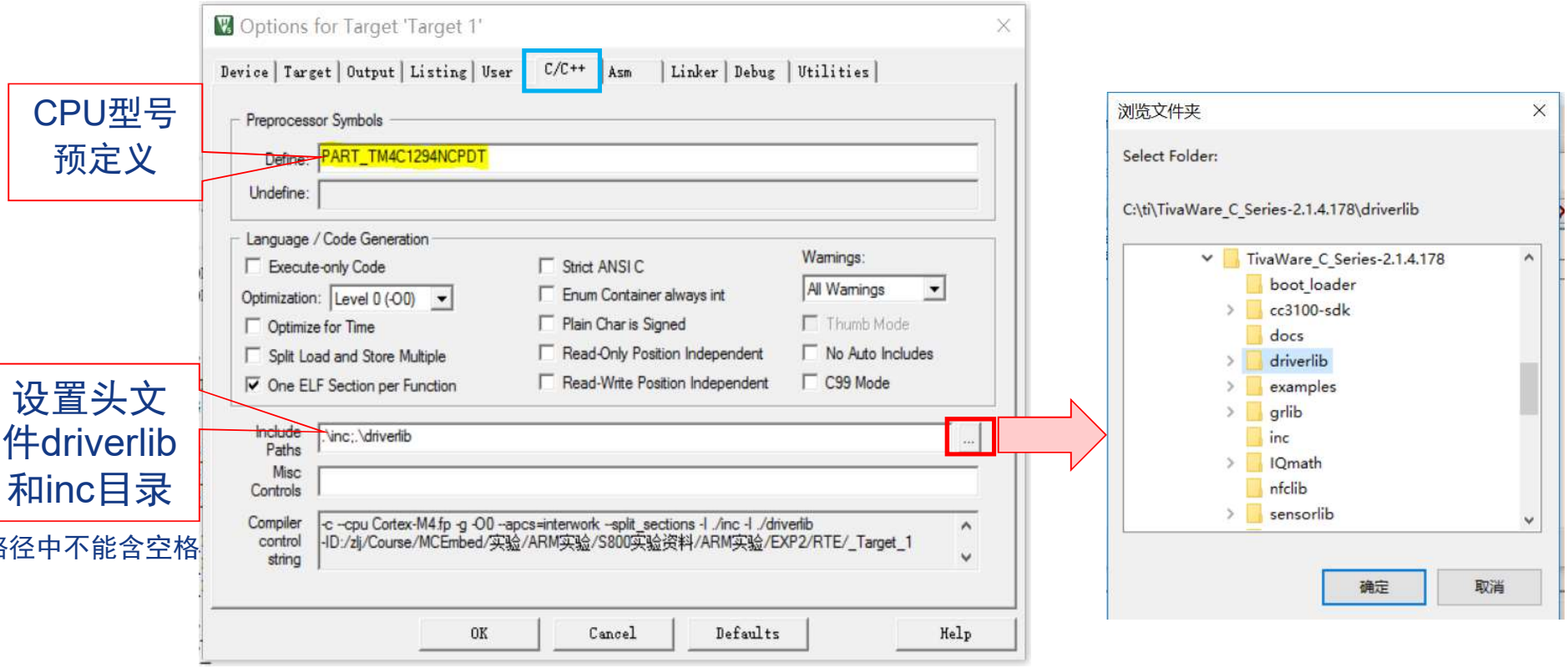
步骤五：项目配置

- 选择Project→Options for Target ‘Target 1.’ （快捷方式 ）



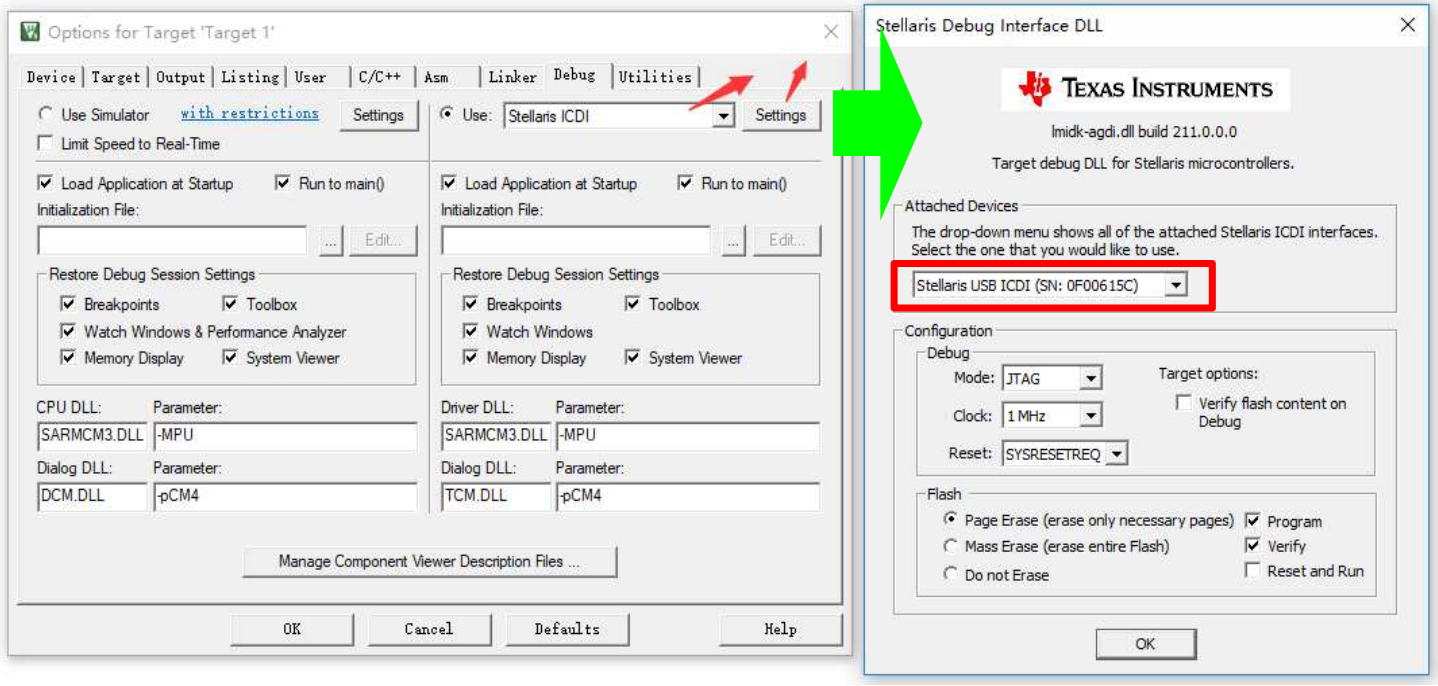
设置头文件搜索路径

- 在C/C++选项卡中，按照下图所示添加预定义CPU型号和头文件搜索路径



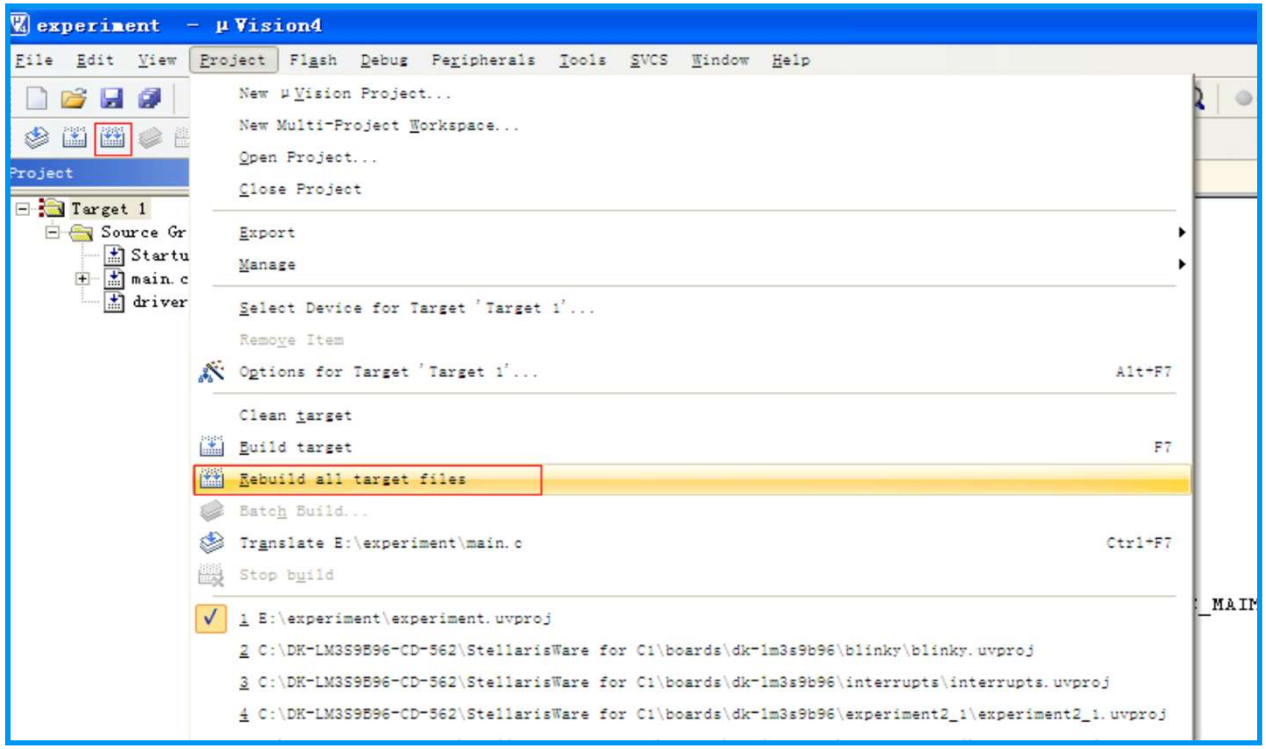
设置调试适配器

- 在Debug选项卡中，选中“Use”，在下拉列表中选择“Stellaris ICDI”选项，点击Settings，在弹出的对话框进行JTAG时钟和复位方式等设置，并在Run to main()选项前打勾



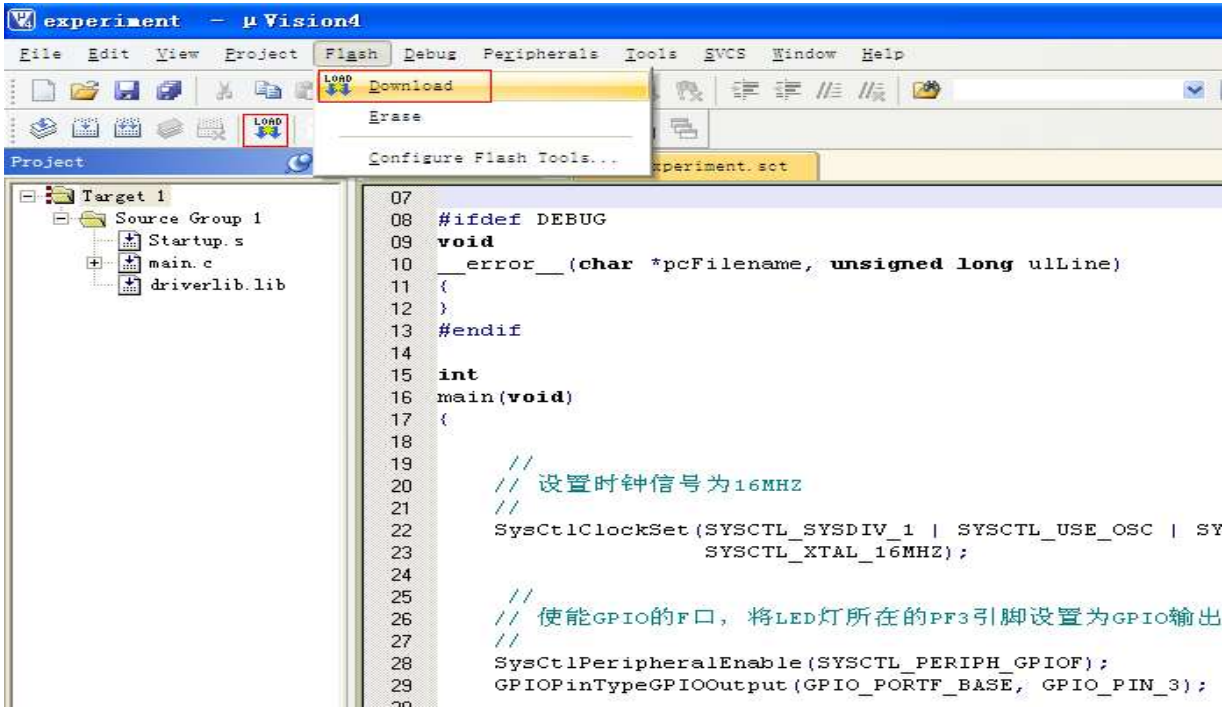
步骤六：编译和下载程序

- 编译：选择“Project→Build Target或Rebuild all target files”编译链接该项目。编译成功后，编译输出窗口显示信息，并生成目标文件(.axf)



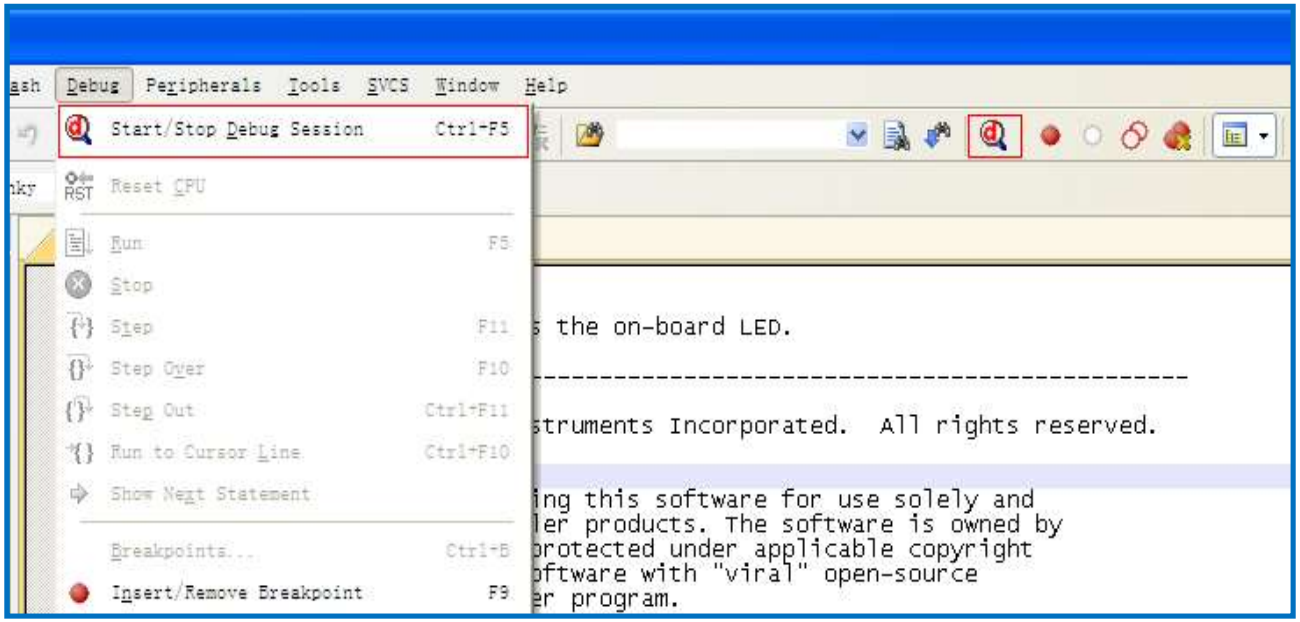
编译和下载程序

- 下载：连上数据线，选择“Flash→Download”或者单击“LOAD”快捷按钮下载程序到Flash存储器。IDE窗口的底部可以看到进度条



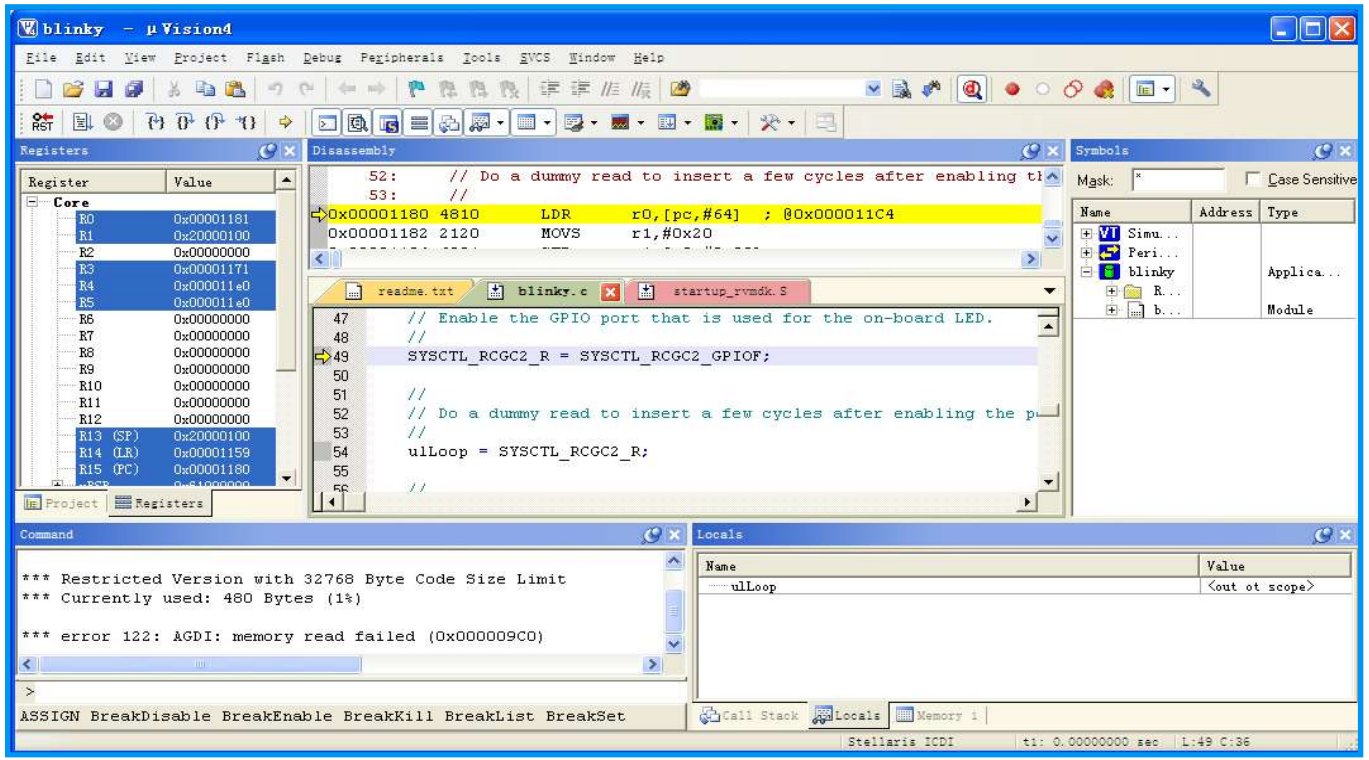
步骤七：调试和运行程序

- 运行：直接按实验板上的 Reset 按钮全速运行程序
- 调试：选择“Debug→Start/Stop Debug Session”, 或者单击“Debug”快捷按钮调试运行



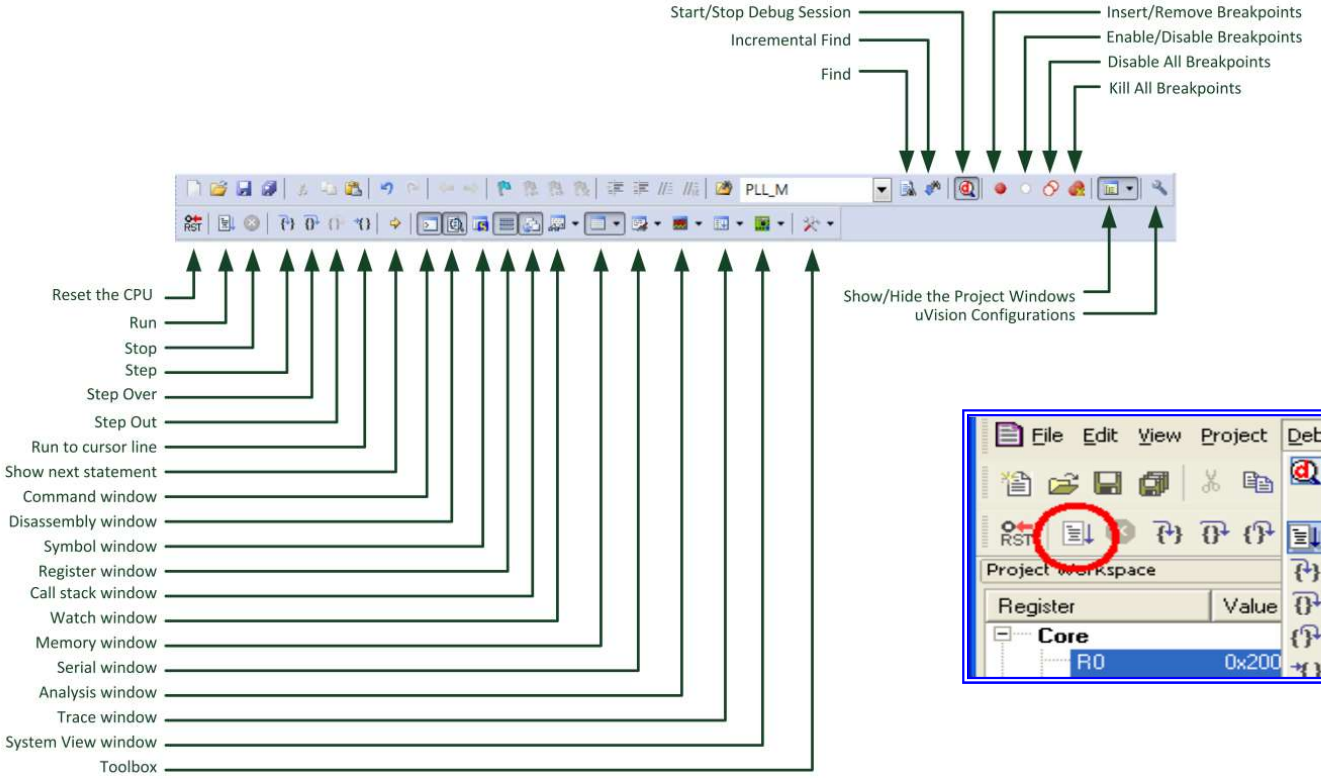
调试运行

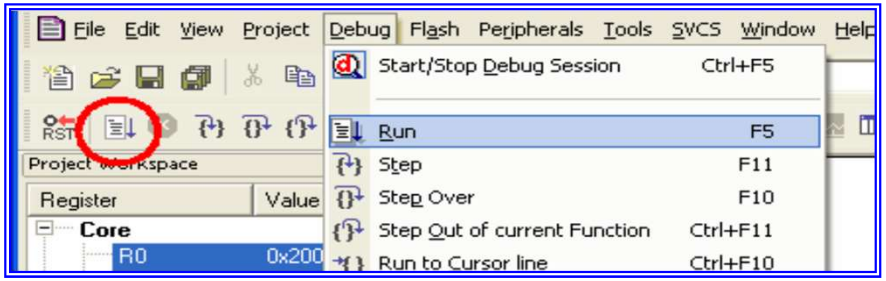
- 进入调试模式。在左边的窗口中显示处理器的各寄存器，在底部可看见调试命令窗口，主窗口显示正在调试的源代码。调试器自动在main函数处停止。



调试和运行程序

- 现在开始，您可以修改存储器，程序变量和处理器寄存器，设置断点，单步运行以及所有其它调试方法。要运行程序，在Debug目录里选择“Run”，或者F5。





小组讨论：ARM C预备实验

- 完成Keil MDK及其相应软件的安装
- 熟悉Keil C开发环境，掌握系统设置和编译、下载、调试方法

1. 新建一个工程项目，选择CPU型号、启动代码，理解Startup程序的功能和结构；
2. 添加驱动库driverlib.lib、main.c、led.c和sum.s源文件，设置Include路径（Inc/DriverLib目录，C/C++选项）；
3. 编译、下载程序，调试运行，观察并解释ARM寄存器的变化；
4.* 了解C与ARM汇编之间子程序调用的方式。
5. 试着运行安装目录下的例程
   ti\TivaWare_C_Series-2.1.4.178\examples\boards\ek-tm4c1294xl

开始你的第一个项目