任务6-1实验指导

本任务主要学习时间函数。

1. 硬件设计

键盘硬件设计如图所示。

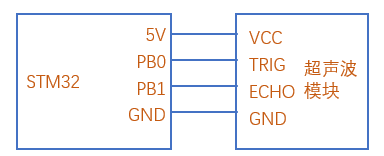
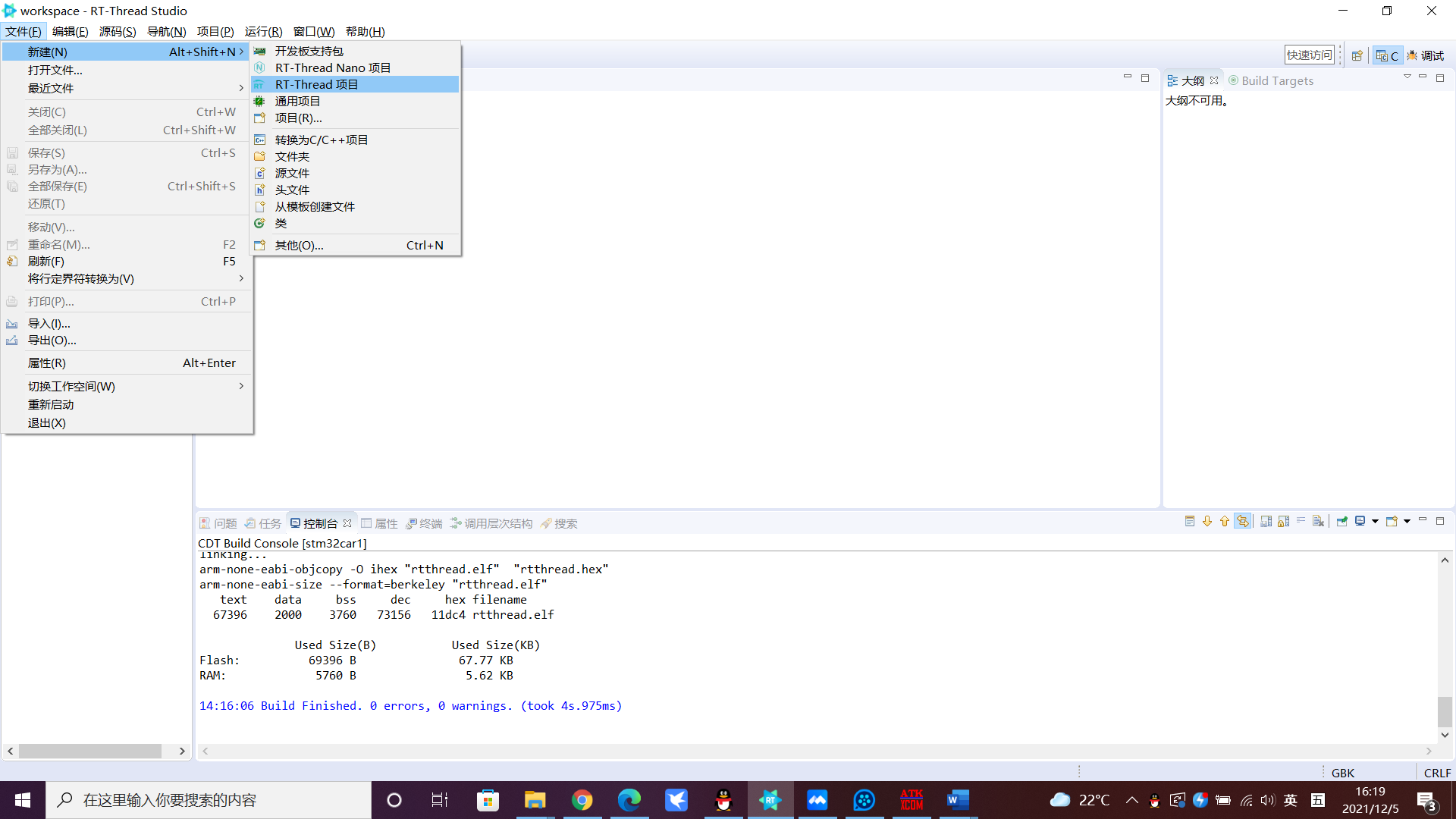


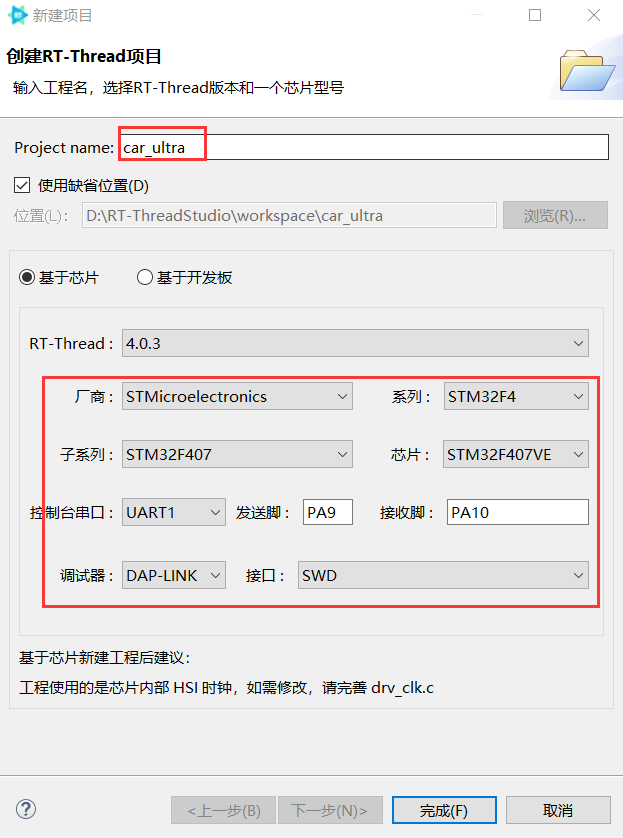
图 硬件连接图

二、新建项目

1、如下图，在RT-Thread Studio中，找到“文件->新建->RT-Thread项目”，单击“RT-Thread项目”：

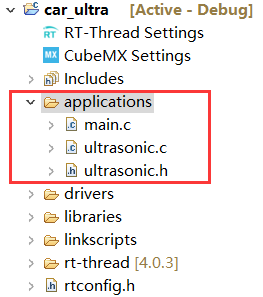


2、设置项目名称为“car\_ultra”、项目保存位置、选择项目所用芯片的厂商和型号等信息，单击“完成”。



二、编写代码

复制“操作指导\6-1”目录下的 ultrasonic.c、ultrasonic.h、main.c文件到项目中的applications目录下，结果如下：

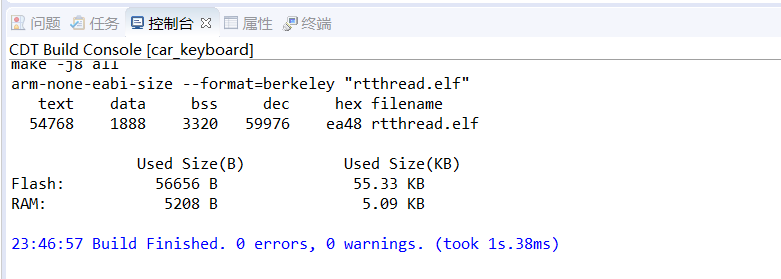


三、编译下载

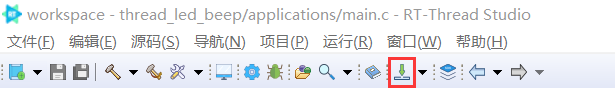
1、编译，点击下图构建按钮进行编译



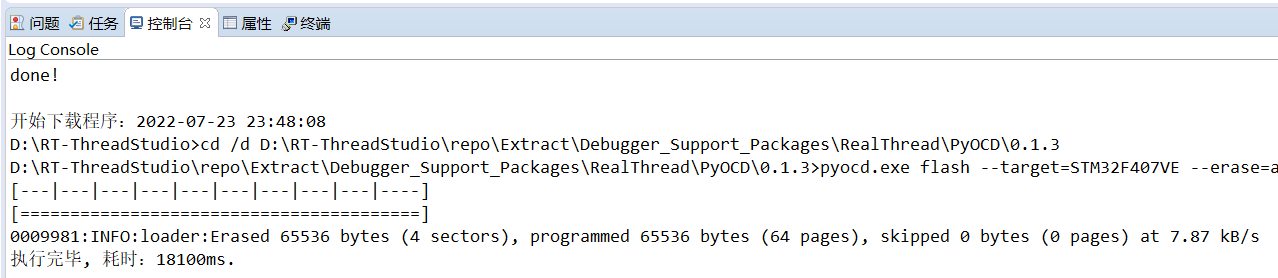
2、编译完成后，如下所示：



3、下载，点击如下图中的下载按钮进行程序下载



下载完成结果如下：



**四、程序测试**

启动系统后，可以观察到终端打印输出测量到的距离，如图6-5。

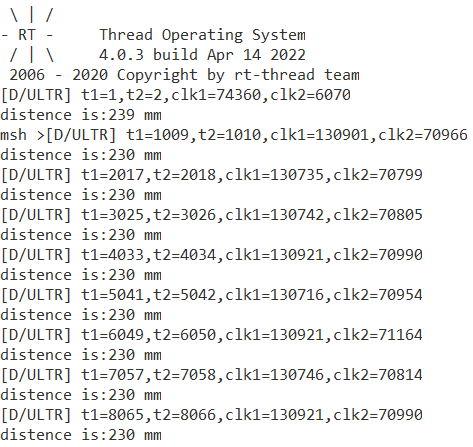


图6-5 距离测量结果

测试发现，如果超声波模块没有响应，会导致线程卡死。主要原因是因为使用查询的方法等待ECHO信号的到来，而且没有超时机制，所以如果因为硬件问题一直无法产生ECHO信号，则会导致线程在等待ECHO信号时卡死，一直占用CPU，进而导致系统低优先级线程无法得到CPU资源（这种线程长时间无法得到CPU资源的现象，在操作系统理论上称为线程饿死）。

解决这个问题的一个方法是提供一种超时机制，当线程等待信号超过一定时间时，线程主动退出等待。通常操作系统都会提供这种机制，那就是系统定时器。

任务6-2实验指导

任务描述：本任务中，我们要让两个车灯分别闪烁，其中一个周期性闪烁，另一个只闪烁一次就停止闪烁，要求采用定时器的方法实现。

**6.4.1 软件设计**

本任务主要目的是使读者了解定时器的使用方法，我们设置两个定时器，一个是周期性定时器，使用默认的硬件中断模式定时器，用于控制左车灯的闪烁；一个是单次性定时器，配置为软件线程模式定时器（回调函数在线程上下文运行），用于控制右车灯。

新建工程，在main.c文件中进行如下程序设计：

#include <rtthread.h>

#include <rtdevice.h>

#include "drv\_common.h"

#define DBG\_TAG "main"

#define DBG\_LVL DBG\_LOG

#include <rtdbg.h>

/\* 定义左右转向灯的控制引脚 \*/

#define LED\_L\_PIN GET\_PIN(D, 8)

#define LED\_R\_PIN GET\_PIN(D, 9)

/\* 静态方式定义定时器timerL \*/

static struct rt\_timer timerL;

/\* 动态方式定义定时器timerR \*/

static rt\_timer\_t timerR;

/\* 用于计算定时器的运行次数 \*/

rt\_uint32\_t countL = 0;

rt\_uint32\_t countR = 0;

/\* 定时器timerL的超时回调函数 \*/

static void timeout\_led\_l(void \*parameter)

{

LOG\_D("countL=%d",countL);

if(countL++%2 == 0)

rt\_pin\_write(LED\_L\_PIN, PIN\_LOW);//亮左灯

else {

rt\_pin\_write(LED\_L\_PIN, PIN\_HIGH);//灭左灯

}

}

/\* 定时器timerR的超时回调函数 \*/

static void timeout\_led\_r(void \*parameter)

{

LOG\_D("countR=%d",countR);

if(countR++%2 == 0)

rt\_pin\_write(LED\_R\_PIN, PIN\_LOW);//亮右灯

else {

rt\_pin\_write(LED\_R\_PIN, PIN\_HIGH);//灭右灯

}

}

int main(void)

{

/\* 把引脚设置为输出模式 \*/

rt\_pin\_mode(LED\_L\_PIN, PIN\_MODE\_OUTPUT);

rt\_pin\_mode(LED\_R\_PIN, PIN\_MODE\_OUTPUT);

rt\_pin\_write(LED\_R\_PIN, PIN\_HIGH);//灭右灯

rt\_pin\_write(LED\_L\_PIN, PIN\_HIGH);//灭左灯

/\* 初始化静态定时器，周期定时器 ，系统默认定时器回调函数在中断上下文运行\*/

rt\_timer\_init(&timerL,"timer\_l", timeout\_led\_l,

RT\_NULL, 1000,

RT\_TIMER\_FLAG\_PERIODIC);

rt\_timer\_start(&timerL);

/\* 动态创建定时器，单次定时器 ，把定时器回调函数设置为线程上下文运行\*/

timerR = rt\_timer\_create("timer\_r", timeout\_led\_r,

RT\_NULL, 2000,

RT\_TIMER\_FLAG\_ONE\_SHOT|RT\_TIMER\_FLAG\_SOFT\_TIMER);

if (timerR != RT\_NULL)

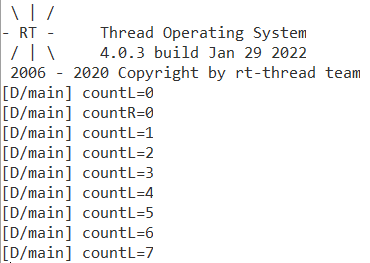
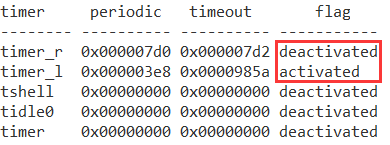
rt\_timer\_start(timerR);

return RT\_EOK;

}

**6.4.2 程序测试**

（1）启动系统，观察终端打印如图6-6（a），可以看到，右定时器只运行了一次就没有再运行了。而左定时器不停的运行；再观察车灯，右车灯一直亮着，而左车灯一直闪烁。

（a） （b）

图6-6 定时器测试结果

（2）在终端输入“list\_timer”命令，观察终端输出如图6-6（b），可以看到，右定时器处于未激活状态。左定时器处于激活状态。（注，未启动的定时器也是处于未激活状态）

任务6-3实验指导

任务描述：通过串口终端实时显示小车与前方障碍物的距离。要求硬件问题不能影响软件的执行。

**6.5.1 程序设计**

本任务只需在任务6-1基础上对ultrsonic.c进行修改，代码清单如任务6-1所示，由于任务6-1中我们已经设计了条件编译，所要在本任务中，我们只需在任务6-1的ultrsonic.c文件中修改相关预处理的宏定义，修改后宏定义如下：

/\* 序编译宏定义，如果要使用定时器，可以打开ULT\_USE\_TIMER宏 \*/

#define ULT\_USE\_TIMER

**6.5.2 程序测试**

测式方法如任务6-1。

（1）当超声波没有响应时，终端输出超时提示，且终端可以继续输入其它命令，如图6-7。

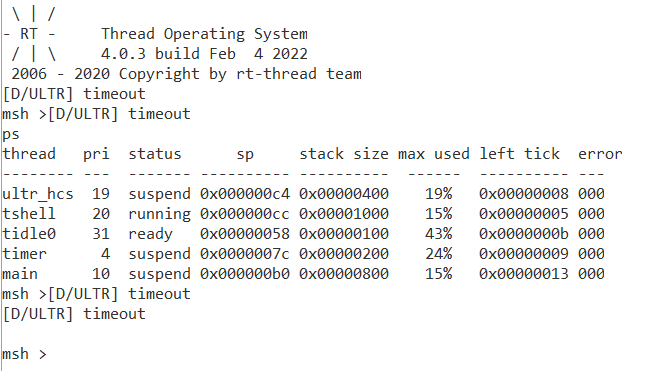


图6-7 超声波测距测试结果1

（2）当超声波正常时，终端打印出测量到的距离，如图6-8。

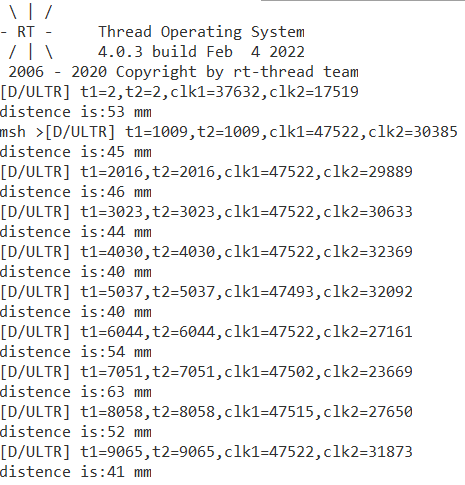


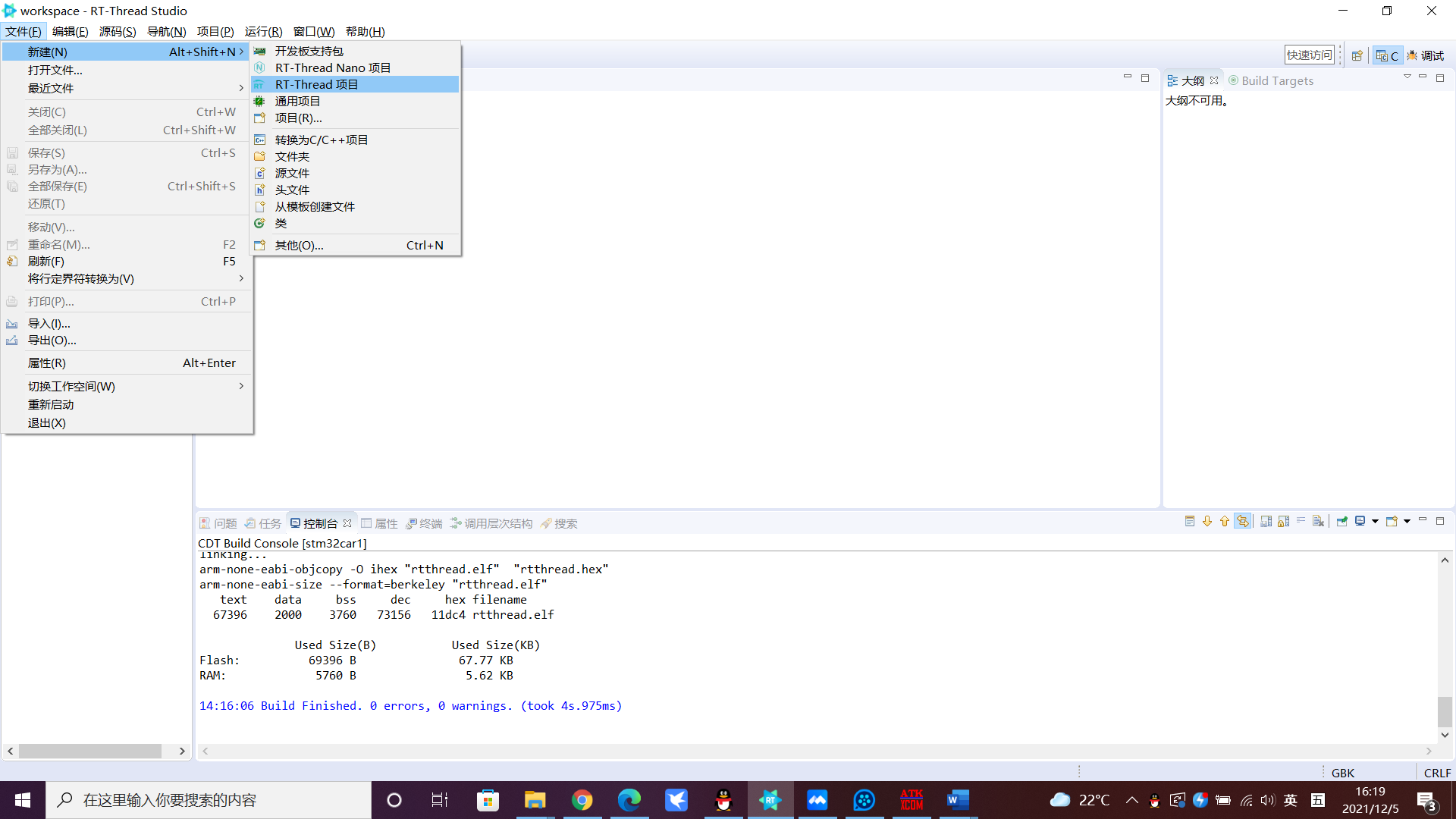
图6-8 超声波测距测试结果2

本任务我们创建了一个线程进行距离实时测量，也就是说，不管用户是否需要距离测量，线程都会以一定的频率进行距离测量。然而，在某些情况下，用户可能只需偶尔进行一次测量，这种实现方法显得有些浪费CPU资源。

任务6-4实验指导

一、新建项目

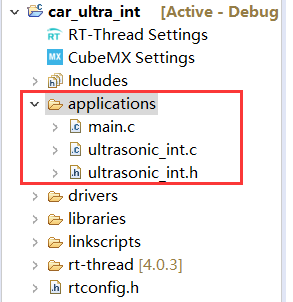
1、如下图，在RT-Thread Studio中，找到“文件->新建->RT-Thread项目”，单击“RT-Thread项目”：



2、如下图，设置项目名称为“car\_ultra\_int”、项目保存位置、选择项目所用芯片的厂商和型号等信息，单击“完成”。

二、编写代码

复制“操作指导\6-4”目录下的 ultrasonic\_int.c、ultrasonic\_int.h、main.c文件到项目中的applications目录下，结果如下：

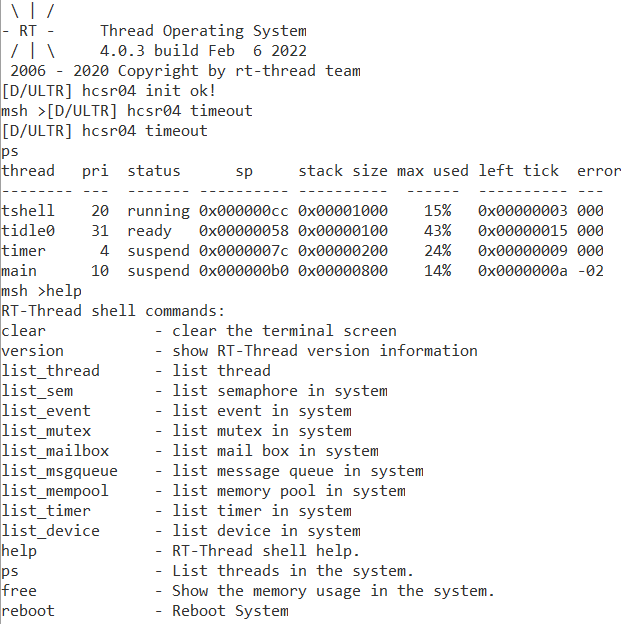
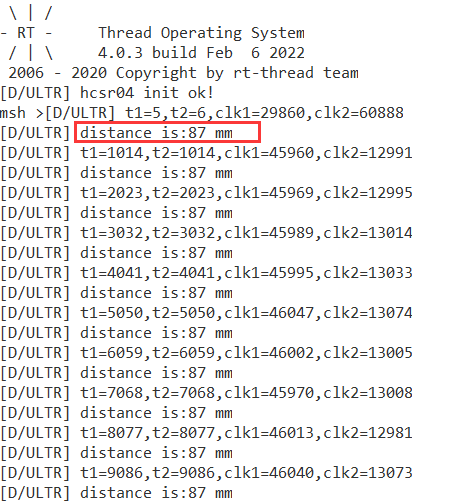


三、编译下载

**四、程序测试**

（1）接上超声波模块，启动系统，观察打印如图6-9（a）所示，可以看到，测得的距离为87mm，测试结果很稳定。

（2）把超声波模块去掉，启动系统，观察打印如图6-9（b）所示，可以看到，测试超时，且不影响系统其它线程的工作。



（a） （b）

图6-9 测试结果