

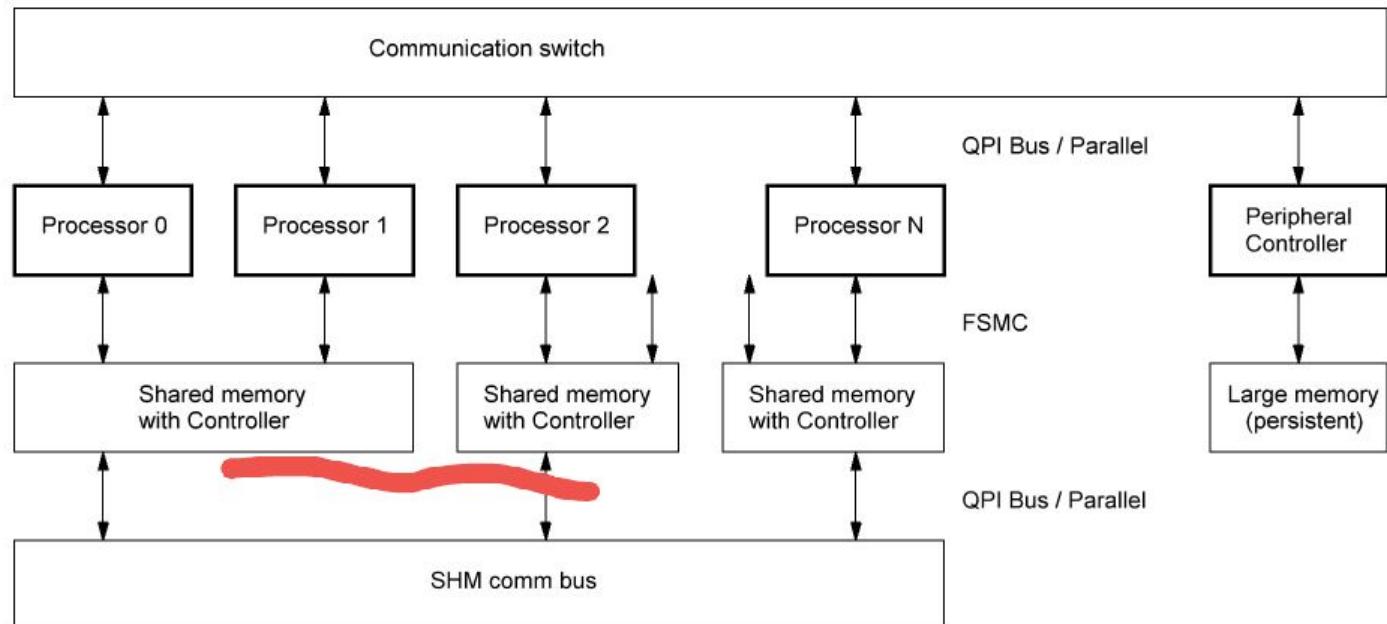
# Mid-term Report

Project name:

物联网应用中使用多个  
**STM32F101**微控制器的  
并行计算系统

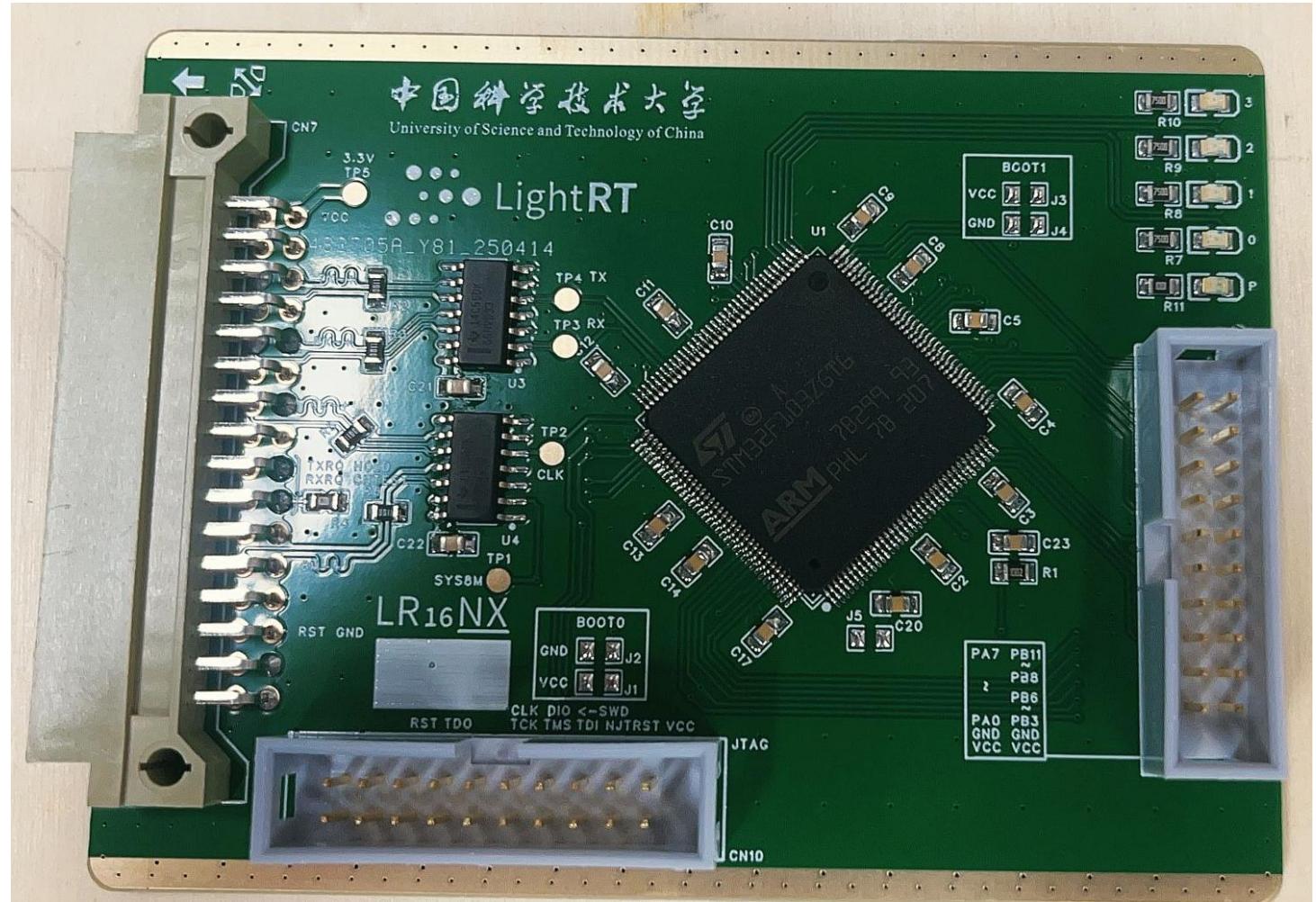
组长：位文康

组员：罗嘉宏、崔卓、郭彦禛



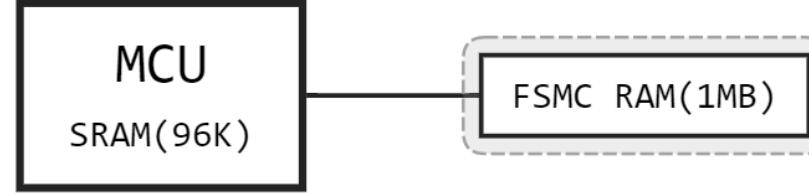
# Contents

1. Overview
2. Preliminary research
3. Feasibility analysis
4. Current progress



# Overview

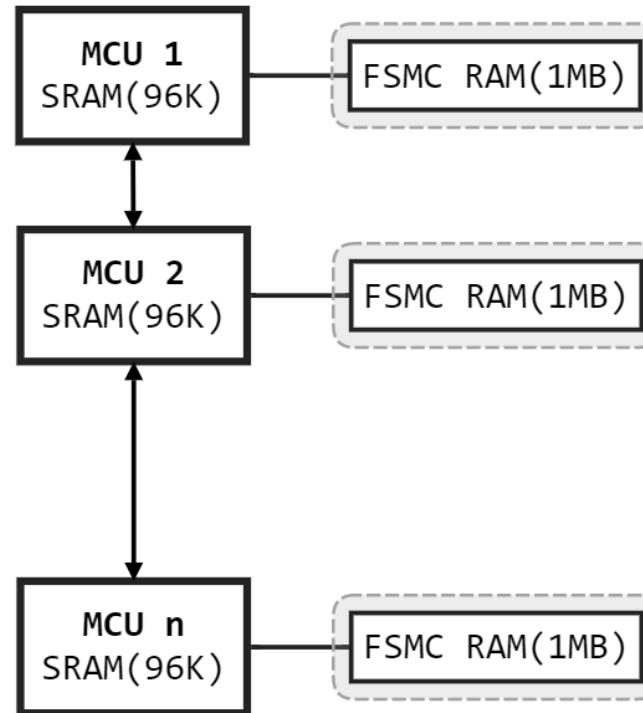
Version1: (伪) 多任务



更正：FSMC SRAM(16MB)

Target: 多个MCU上的 (真正的) 多任务

Note: FSMC RAM 直接映射进 RAM, OS 使用 MCU SRAM 96K 空间, 通过 RAM 映射来调度 FSMC 中的用户进程



# Preliminary research

STM32F103 是 STM32F1 系列单片机的一种，使用 32 位的 ARM Cortex-M3 核心。可以运行在 ~72MHz 的主频下，并具有最大 1MB 的 Flash 存储器、96KB 的内部高速 SRAM（我们使用的 STM32F103ZGT6 具有这些配置）。通过片上 FSMC 控制器，可以将地址区域的一部分映射到外部 RAM 或者 Flash，实现存储空间的扩展和多任务共用地址区间的能力。此外，该单片机还有多个 USART 接口，并有多个定时器，可以实现片间通信和多任务切换所需要的中断功能。



# Preliminary research

## OS Features:

1. 多任务
  - 默认抢占式多任务
  - 通过 FSMC 外挂 RAM 的不同区域实现内存切换
  - 提供局部实时 realtime，方便有时序需求的任务
  - 优先级控制
  - 可以主动调度
2. 通信与同步
  - 类 Socket 消息传递
  - Shared memory 传递大块数据
  - DMA 协助处理大块数据
3. 实时功能
  - 中断
4. 外设
  - 外设硬件的抽象，为用户程序提供 API
5. 调试
  - 为用户进程输出调试信息

# Feasibility analysis

我们的项目适用于需要分布式处理的中小型物联网项目，例如机器人比赛使用的机器人。

鉴于使用场景，我们不需要做真正的多核系统，也不需要重视内存保护，因为一旦烧录很难更改（除非再次烧录），而且单片机的内存是有限且宝贵的，所以我们选择相信用户程序不会滥用内存。

## Advantages:

- 模块化和可扩展设计。
- 低成本硬件。
- 通过自写操作系统和通信协议实现高度定制。

## Challenges:

- 通信和同步开销会影响性能。

# Current progress

Version1: (伪) 多任务

**System APIs**

**Symbol definition 符号定义**

- ▲ = 内部 *unsafe* API

## Return values 返回值定义

### 整数返回的 API

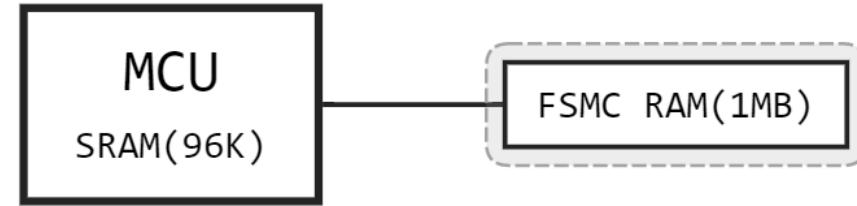
- `0` = 成功完成
- 负数 = 失败
- 正数 = 按 API 定义

### 指针返回的 API

- `NULL` = `0` = 失败

## Environment 环境

- `int cuid(int* uid)` = 获取当前处理器 ID
- `int sysinfo(sysinf* info)` = 获取系统信息
- `int tick(unsigned int* tl, unsigned int* th)` = 获取时钟计数器的值



更正: FSMC SRAM(16MB)

# Current progress

- ## System APIs
- ### Symbol definition 符号定义

  - `⚠` = 内部 *unsafe* API
- ### Return values 返回值定义

  - `0` = 成功完成
  - 负数 = 失败
  - 正数 = 按 API 定义
- ### 整数返回的 API

  - `NULL` = `0` = 失败
- ### 指针返回的 API

  - `NULL` = `0` = 失败

## Process control 任务控制

所有需要传入 PID 的都可以传入 `-1` 表示当前的 PID。

- `int exec(entryproc p, void* param, const startinf* si)` = 启动新的任务
- `int terminate(int pid)` = 中止任务
- `int cpid(int* pid)` = 获取当前 PID
- `int pinfo(int pid, procinf* pi)` = 通过 PID 得到任务信息 复制到用户区
  - `int suspend(int pid, int s)` = 挂起  $s = 1$  或者正常运行  $s = 0$  一个任务
  - `int rtime(int pid, int s, int timeout)` =  $s = 1$  在 `timeout` 或者 `realtime(pid, 0, X)` 前不要调度走
- `int pset(int pid, const procinf* pi)` = 设置 PID 对应的任务信息 从用户区
- `int ppinfo(int pid, procinf** ppi)` `⚠` = 访问系统内部的任务信息
- `int swtch(int pid)` = 调度到 PID
- `int sleep(int ms)` = 等待至少 ms 后调度, 传入 0 表示 yield, 传入 -1 表示无限等待
- `int sleepus(int us)` = 等待至少 us 后调度, 传入 0 表示 yield, 传入 -1 表示无限等待
- `int brkwait(int pid)` = 使得 PID 退出等待, 例如 `sleep()` 带来的等待
- `int yield()` = 调度走

# Current progress

## System APIs

### Symbol definition 符号定义

- ⚠ = 内部 *unsafe* API

### Return values 返回值定义

### 整数返回的 API

- 0 = 成功完成
- 负数 = 失败
- 正数 = 按 API 定义

### 指针返回的 API

- NULL = 0 = 失败

## Signal & Synchronization, Access control 信号和同步、访问控制

- `int signew(int flag, int* sig)` = 创建信号，可以创建全局所有处理器共享或者局部处理器内或者任务内
- `int sigclose(int sig)` = 关闭信号
- `int sigset(int sig, int s)` = 设置  $s = 1$  或恢复  $s = 0$  信号
- `int waitsig(int sig, int timeout)` = 等待信号  $timeout = -1$  表示一直等待
- `int waitany(const int* sigs, int nsig, int timeout)` = 等待任意一个信号  $timeout = -1$  表示一直等待
- `int readsig(int sig)` = 读取信号状态

## Communication 流式通信

- `int comopen(int uid, int pid, int flag, comm** f)` = 建立到  $uid$  = 处理器ID,  $pid$  = 过程ID 的通信连接。传入  $uid = -1$  表示本处理器
- `int comclose(comm** f)` = 关闭 f
- `int comsend(comm** f, void* buf, int len)` 发送  $[buf, buf + len]$  到目标
- `int comrecv(comm** f, void* buf, int len, int* wlen)` 阻塞式接收
- `int comcp(comm** f, void* buf, int len, int* wlen)` 复制式接收 不从缓冲区中删去数据
- `int comerase(comm** f, int len)` 从缓冲区中删去数据
- `int comavail(comm** f, int* len)` 检查接收到的未读取的多少数据
- `int comcheck(comm** f)` 检查连接是否正常

# Current progress

目前的项目结构：

已经完成了部分通信与多任务模块的工作

```
src
└── sys
    ├── comm
    │   ├── comm.c
    │   └── comm.h
    ├── signal
    │   ├── signal.c
    │   └── signal.h
    ├── task
    │   ├── task.c
    │   ├── task.h
    │   └── task.s.h
    ├── timer
    │   ├── timer.c
    │   └── timer.h
    └── global.h
uinc
└── lightrt.h
└── signal.u.h
└── task.u.h
└── timer.u.h
usrc
└── main.c
└── Makefile
└── README.md
```

Question Time

Thanks

组长：位文康

组员：罗嘉宏、崔卓、郭彦禛