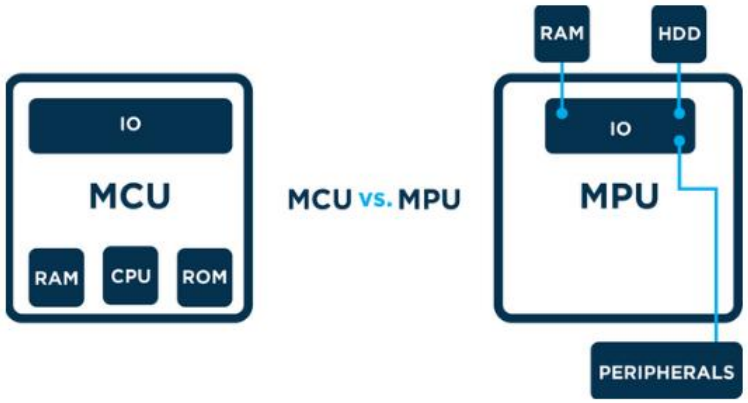
Programming on Embedded Circuits

DCE
Dept. of Computer Engineering

Micro-Controller Platform

- **Micro-Controller Unit** (MCU) contains RAM, ROM and IO
- **Micro-Processor Unit** (MPU) only contains the CPU
- **System on Chip** (SoC) refers to MCUs with a greater number of onboard peripherals and functionality



http://theairboard.cc

http://www.ti.com

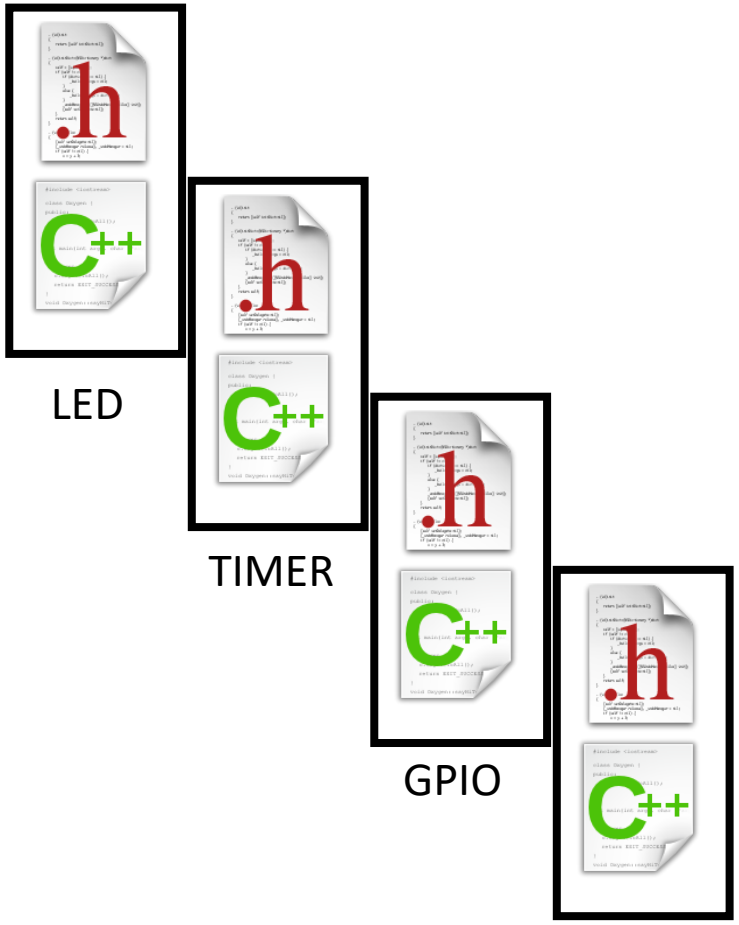
http://www.microchip.com

C Language: Header and C++ Files


```
#ifndef __LED_H_
#define __LED_H_

#include <system_lib.h>
#include "user_lib.h"
#include "UserFolder/lib.h"

extern int T_on;
extern int T_off;

void setOn(long duration);
void setoff(long duration);

#endif
```


```
#include "led.h"

int T_on;
int T_off;
int counter;

void setOn(long duration){
    //TODO: set LED on here
}
void setoff(long duration){
   //TODO: set LED off here
}
void delay(long duration){
  //TODO: set delay here
}
```

C Language: Main File




```
#include "led.h"
#include "timer.h"
#include "gpio.h"
#include "button.h"

void main(){
    initGPIO();
    initTimer();
    initButton();
    initLED();
    ....
    ....
    while(1){};
}
void timer_isr(){
}
void ext_isr(){
}
```

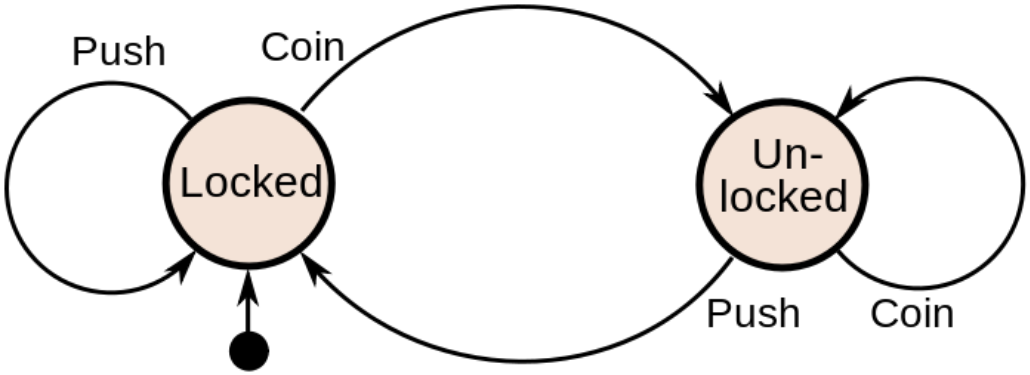
- Modules/ Libraries are included
- Modules/ Libraries are initiated
- System operations are implemented in **interrupt functions**

Finite State Machine (FSM)

- **Finite-State Machine** (**FSM**) or **Deterministic Finite Automata** (**DFA**), **finite automaton**, or simply a **state machine**, is a mathematical model of computation

Current State	Input	Next State
Locked	Coin	Unlocked
	Push	Locked
Unlocked	Coin	Unlocked
	Push	Locked

A turnstile



Finite State Machine Programming

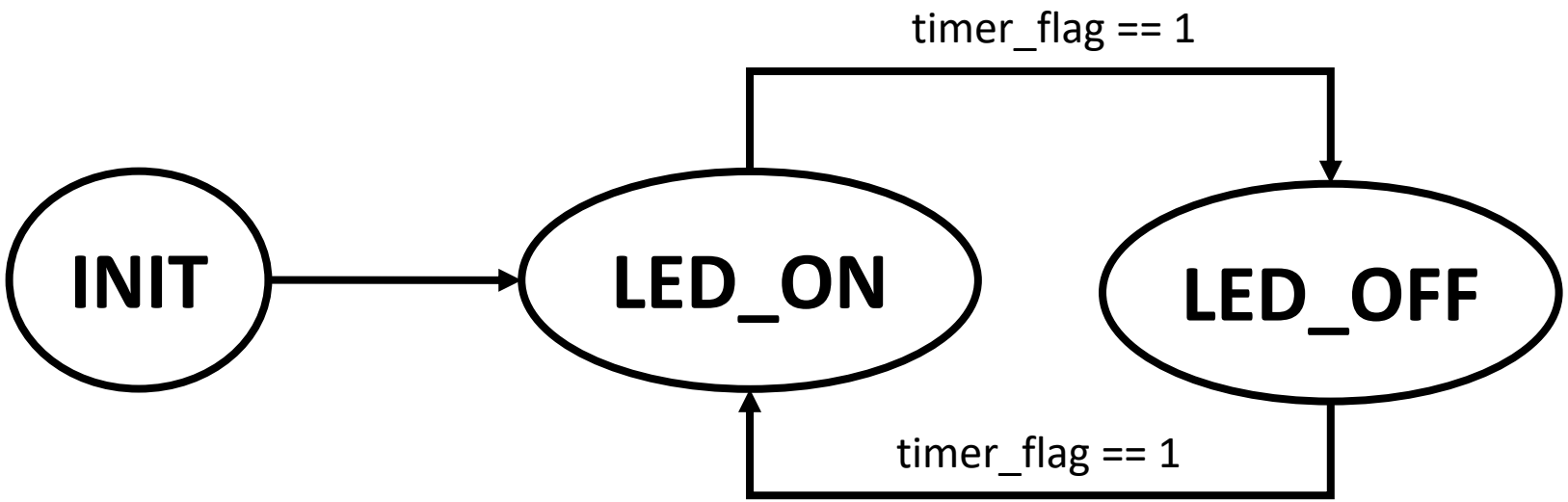
```
while(1){
  switch(status){
    case LOCKED:
      lock_turnstile(); //operation in a state
      if(Coin == true)  //transition condition
            status = UNLOCKED; //next state
      break;
    case UNLOCKED:
      unlock_turnstile(); //operation in a state
      if(Push == true)    //transition condition
            status = LOCKED; //next state
      break;
    default:
      break;
    }
}
```

Example 1

- Given an LED turns on for **T_on** and then turns off for **T_off**.
  - Design an DFA for this LED
  - Implement the DFA in Arduino

- `digitalWrite(13, HIGH)`: turn on the LED
- `digitalWrite(13, LOW)`: turn off the LED
- `setTime(duration)`: set a clock (`timer_flag = 0`), when the clock is expired, `timer_flag = 1`; duration is in mili-seconds.

Answer



- INIT: Set pin 13 to OUTPUT mode, set timer
- LED_ON: Turn on the LED
- LED_OFF: Turn off the LED

Answer (Arduino Code)

```
void loop(){
   switch(status){
      case INIT:
         pinMode(13, OUTPUT);
         setTimer(T_on);
         status = LED_ON;
         digitalWrite(13, HIGH);
         break;
      case LED_ON:
         if(timer_flag == 1){
                  status = LED_OFF;
                  setTimer(T_off);
                  digitalWrite(13, LOW);
         }
         break;
      case LED_OFF:
         if(timer_flag == 1){
                  status = LED_ON;
                  setTimer(T_on);
                  digitalWrite(13, HIGH);
         }
         break;

      default:
         break;
      }
    delay(10);
}
```

```
void timer_run(){
        if(timer_counter > 0)
                timer_counter--;
        if(counter_timer == 0)
                timer_flag =1;
}
void setTimer(long duration){
        timer_counter = duration;
        timer_flag = 0;
}
```

Example 2

- Design a smart lock which accepts 4 digits as a secrete code. However, there is a time-out for each digit (e.g. T_out = 5s). After this period, the system is reset

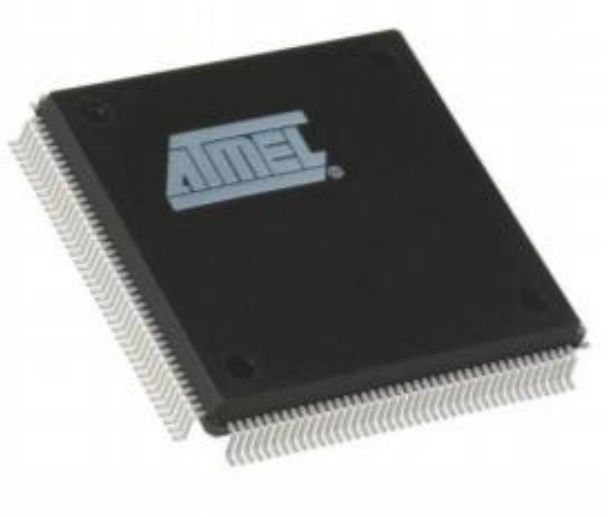
Embedded Platform based on Operating System

- Embedded means **something that is attached** to another thing.

- "Any sort of device which includes a programmable computer but itself is not intended to be a general-purpose computer" [Marilyn Wolf]

- Embedded system has three components:
  - Hardware
  - Real Time Operating System (RTOS)
  - Software

Classification of Embedded System



- Small Scale
- Medium Scale
- Sophisticated Scale

Processor

- Processor is the heart of the Embedded Syste
- General Purpose processor (GPP)
  - Microprocessor
  - Microcontroller
  - Embedded Processor
  - Digital Signal Processor
- Application Specific System Processor (ASSP)
- Multi Processor System using GPPs

Operating System: Linux and Android

- **Connectivity and UI**
  - Two powerful operating systems used in most of the embedded systems
  - Wireless connectivity and graphics interface: **Android OS**
  - Linux comes with a complex flow and it might be difficult for a beginner to understand it

- **Power management**
  - Android and Linux supports effective power management compared to real time operating systems

- **Responsiveness**
- **Cost**

Android Operating System Component

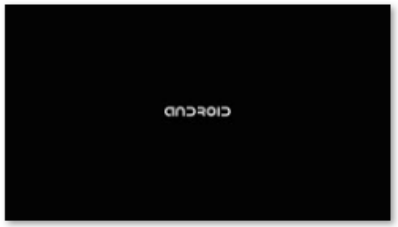
baseparamer-720P.img	07/08/2018 5:49 PM	Disc Image File	1 KB
boot.img	07/08/2018 5:50 PM	Disc Image File	10,656 KB
kernel.img	07/08/2018 5:49 PM	Disc Image File	6,661 KB
misc.img	07/08/2018 5:49 PM	Disc Image File	48 KB
recovery.img	07/08/2018 5:50 PM	Disc Image File	13,776 KB
resource.img	07/08/2018 5:49 PM	Disc Image File	2,799 KB
system.img	07/08/2018 5:51 PM	Disc Image File	778,396 KB
uboot-rk3128.img	07/08/2018 5:49 PM	Disc Image File	2,048 KB

Android Boot File

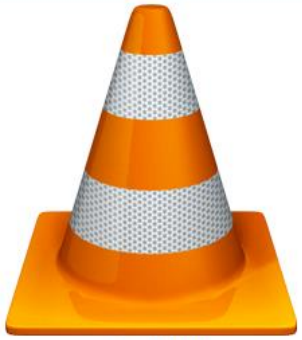
```
mkdir /system
mkdir /data 0771 system system
mkdir /cache 0770 system cache
mkdir /config 0500 root root
mkdir /metadata 0770 root root

service yunos_preinstall
/system/bin/yunos_preinstall.sh
user root
group root
class main
disabled
oneshot
```

Android Resource File

logo.bmp



logo_kernel.bmp



rk-kernel.dts

IR Remote Data

```
        ir_key1 {
            rockchip,usercode = <0xdf00>;
            rockchip,key_table = <0xe3 0x74 0xf7 0x71 0xfe 0xf9 0xa0
            0xb4 0x73 0xe7 0x8b 0xe8 0xfd 0xe5 0x67 0xb7 0x6c 0xb8 0x
            0x3 0xea 0x4 0xaf 0x5 0xed 0x6 0xee 0x7 0xb3 0x8 0xf1 0x9
        };

        ir_key2 {
            rockchip,usercode = <0xff00>;
            rockchip,key_table = <0xeb 0x74 0xe 0x3b 0xd 0x3c 0xc 0x3
            0xf9 0xf4 0x73 0xa7 0x72 0xb7 0x66 0xa3 0x9e 0xfc 0x67 0x
            0xf6 0x2 0xe2 0x3 0xe0 0x4 0xf2 0x5 0xe6 0x6 0xe4 0x7 0xe
        };

        ir_key3 {
            rockchip,usercode = <0x1dcc>;
            rockchip,key_table = <0xee 0xe8 0xf0 0x9e 0xf8 0x67 0xbb
            0xb7 0xd9 0xff 0x74 0xf3 0x71 0xbf 0x8b 0xf9 0x191 0xf5 0
            0x6 0xb1 0x7 0xfc 0x8 0xf8 0x9 0xb0 0xa 0xb6 0xb 0xb5 0xe
        };
    };
```

Android System File

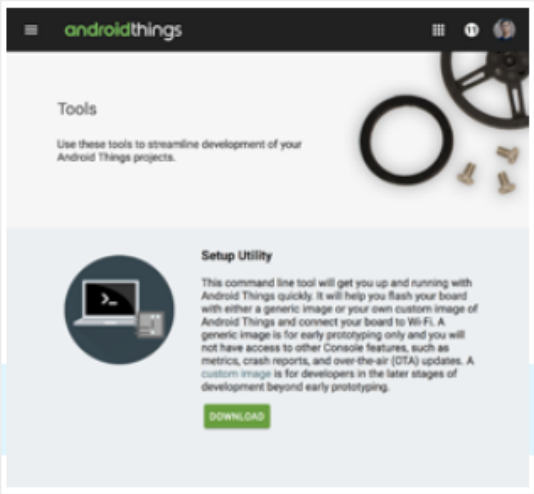
- System apps
- User apps
- Launcher apps
- Preinstall apps
- Libs
- KeyLayout
- …

app	27/07/2018 8:46 A	File folder
bin	27/07/2018 8:07 A	File folder
etc	27/07/2018 7:59 A	File folder
fake-libs	27/07/2018 7:59 A	File folder
fonts	27/07/2018 7:59 A	File folder
framework	27/07/2018 7:59 A	File folder
lib	27/07/2018 7:59 A	File folder
lib64	27/07/2018 7:59 A	File folder
lost+found	27/07/2018 7:59 A	File folder
media	27/07/2018 8:07 A	File folder
preinstall	27/07/2018 7:59 A	File folder
preinstall_del	27/07/2018 7:59 A	File folder
priv-app	27/07/2018 8:45 A	File folder
tts	27/07/2018 7:59 A	File folder
usr	27/07/2018 8:00 A	File folder
vendor	27/07/2018 8:00 A	File folder
xbin	27/07/2018 8:00 A	File folder
build.prop	27/07/2018 8:47 A	PROP File
manifest.xml	07/06/2018 10:49	XML Document

How to Port Android Things?

Step 1: Flash Android Things

Follow these steps to flash the Android Things image onto the microSD card:



1. Download the Android Things Setup Utility from the Android Things Console. You will need to sign in to your Google account and accept the licensing agreement and terms of service.
2. Unzip the downloaded archive.
3. Start the setup utility.

> ★ **Note:** You must run the setup utility as an administrator.

- On Windows, right-click on the executable file and select **Run as administrator**.
- On Mac or Linux, start the utility from the terminal. For example:

```
$ sudo ~/Downloads/android-things-setup-utility/android-things-setup-utility-linux
```

4. Select the option to install Android Things and optionally set up Wi-Fi.
   a. Select **Raspberry Pi 3** as the hardware board.
   b. Choose either a generic image or your own custom image of Android Things for flashing the board.

Create an Android Things Project

- http://developer.android.com/things/training/first-device/create-studio-project

- Prerequisites
  - Update your SDK tools to version 25.0.3 or higher The updated SDK tools enable you to build and test apps for Things.
  - Update your SDK with Android 8.1 (Oreo), API 27 or higher The updated platform version provides new APIs for Things apps.

Create an Android Things Project

- Added library

```
dependencies {
    ...
    compileOnly 'com.google.android.things:androidthings:+'
}
```

- Home Activity

```
<application>
    <uses-library android:name="com.google.android.things"/>
    <activity android:name=".HomeActivity">
        <!-- Launch activity as default from Android Studio -->
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>

        <!-- Launch activity automatically on boot, and re-launch if the app terminates. -->
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.HOME"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
    </activity>
</application>
```