Subject .... Date ....

Embedded Systems

Dr. Pedram

اسلایدهای آموزشی شرکت ARM — برد KL25Z

کتاب Lee & Seshia انتشارات: MIT نسخه 2017

ARM (M0+) processor
M2
M3
M4

تمرین - پروژه (آزمایشگاه) - میان ترم اول - 2 - پایان ترم

② سیستم نهفته چیست؟ پردازنده‌ی خاص منظوره که برای کاربردی خاص طراحی شده و درون یک سیستم دیگر (بزرگتر) قرار می‌گیرد.

مثال: وسایل خانه: لباسشویی، یخچال ... ، دستگاه‌های صنعتی، هواپیما، اتومبیل ← عملکرد پیچیده، چندین پردازنده، نیاز به شبکه ارتباطی

پس تعداد پردازنده‌های بکار رفته در سیستم‌های نهفته بسیار زیاد است.

③ مزایای اضافه کردن یک میکروپروسسور به یک سیستم دیگر چیست؟

- Flexibility
- performance & efficiency
- more functions & features → بخاطر نرم افزار
- Lower cost systems
- more dependability → قابلیت اتکاپذیری = طول عمر بالاتر

نه چراکه سخت‌افزار مکانیکی ساده شده و پیچیدگی به سمت نرم‌افزار منتقل شده پس استهلاک کمتر است

روش‌های ساخت Embedded Systems

Discrete Logic → توان (کار در واحد زمان) و انرژی (تعداد Instruction/program انجام شده)، هزینه بالا

ASIC (Application Specific Integrated Circuit) → هزینه پایه غیر معقول برای تیراژ پایین

Programmable Logic → مدار program می‌شود (low level): processor به آن اضافه نمی‌شود

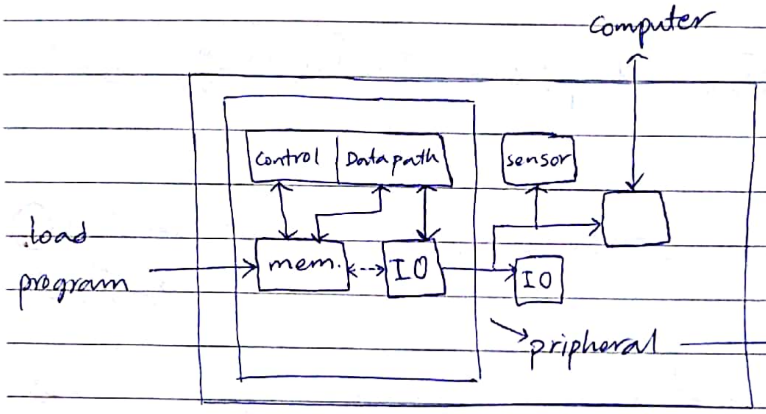
FPGA, PLD

MP + Mem. + Pripherals → همه در یک IC
↓
Microcontrollers (Interal mem. & pripherals)

Scanned by CamScanner

Subject .... Date ....

* فاکتور توان مهم تر از انرژی است چون با خنک‌سازی خطرات سیستم مورد توجه است.
دستگاه‌هایی که با باتری کار می‌کنند انرژی محدود دارند پس فاکتور انرژی برای آن‌ها مهم است.
دستگاه‌هایی که انرژی بسیار بالا نیاز دارند، گرمای زیادی تولید می‌کنند و تهویه آن مشکل است پس انرژی در آن‌ها و تامین هزینه قابل توجه است.



prototype or evalnation board

pripheral → ارتباط با دنیای خارج

← برای هر طراحی‌ای مصرف دارد ← اضافه کردن friendly IO، تعریف، Accelerator، ...
ولی برای تیراژ بالا امکانات اضافه دارد که باید حذف شود پس در این مورد به سراغ ASIC می‌رویم

جلسه دوم - 96/6/28

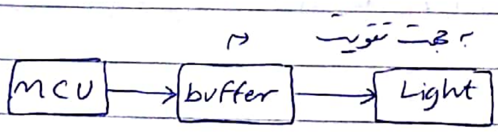
example: A Green House

→ Inputs : Temprature
Humidity
Light

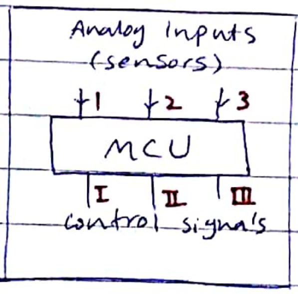
→ Outputs : Heater
Humidifier
Light Switch
Display

یکی از مشکلات این طراحی این است که توزیع پارامترهای مورد نظر در کل فضای گلخانه یکنواخت نیست و سنسورها تنها در یک نقطه یا نقاط مشخصی قرار دارند.

→ Functions : Keep the parameters in a predefined rang; within constraints



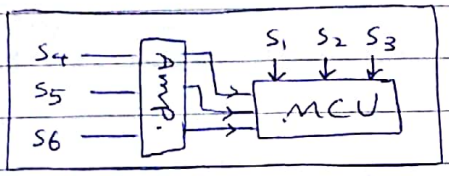
MCU هم ورودی دیجیتال می‌پذیرد ولی سیگنال‌های موجود در محیط آنالوگ هستند پس نیاز به مبدل ADC داریم.



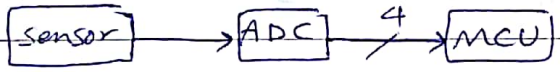
1 Light sensor 2 Temp. Sensor 3 Humidity Sensor

I Light on/off II Temp. on/off III Humidifier on/off

Scanned by CamScanner



کمیت آنالوگ در سیر طولانی تا رسیدن به MCU تضعیف می‌شود. با تقویت کننده‌ها تا حدی مشکل حل می‌شود ولی نه بطور کامل. برای حل کامل مشکل سگنال را بصورت دیجیتال به MCU می‌رسانیم. همچنین تمامین می‌کنیم که on و off بودن وسیله مشخص می‌شود نیازی به دقت بالا نداریم. برای ارسال داده آنالوگ به یک سیم نیاز داریم ولی برای ارسال داده دیجیتال بصورت موازی به چند سیم نیاز است که در مسافت بالا باز به مشکل می‌خورد.



جلسه دوم - 96/7/2

Microcontroller

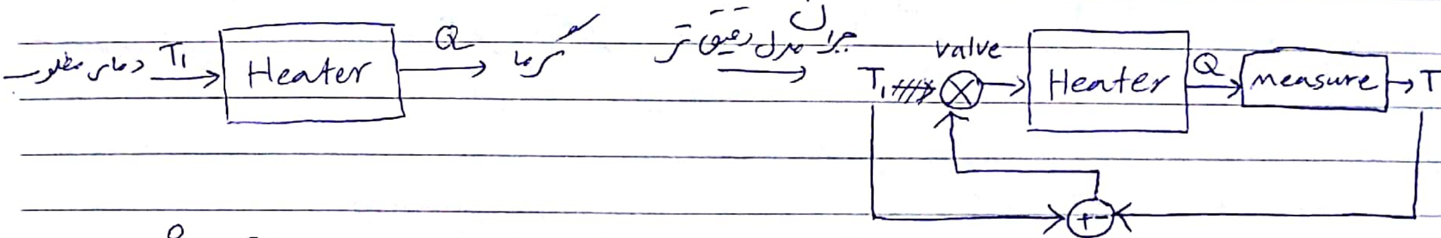
Microprocessor + Memory + IO + ADC + Communication means
+ Timer + MMU (memory management unit) I, SPI, I^2C, CAN, UART, LAN

Microprocessor Applications 1٪ → تنها در وسایل محاسباتی
Microcontroller Applications 99٪

→ need to add:
power supply
mechanical enclosure
software → requirements
design
development cycle
testing

Embedded System Functions

→ closed-loop control system → no accurate model



مقایسه‌گر ← در صورت برابری T و T_1 خروجی صفر می‌شود پس valve بسته می‌شود

عملیات ممکن است به سادگی یک مقایسه نباشد، مثلاً نیاز به انتگرال‌گیری باشد که اگر کار را به جای سخت افزار به نرم‌افزار بسپاریم کار ساده‌تر است.

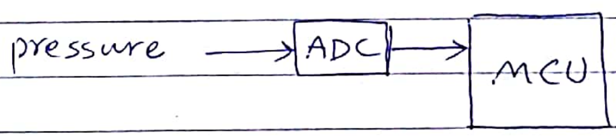
→ Sequencing → از استیتی به استیت دیگر رفتن بسته به شرایط

→ Signal processing

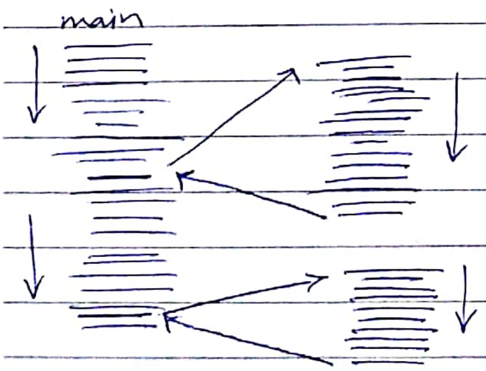
→ Communication & Networking

Attributes of Embedded Systems

→ concurrent reactive behaviour



با دوره تناوب مشخص فشار اندازه‌گیری می‌شود و از طریق Interrupt داده به MCU داده می‌شود و خود MCU سرکشی نمی‌کند



که برنامه‌ها به صورت concurrent انجام می‌شوند (parallel نیست)

Constraints

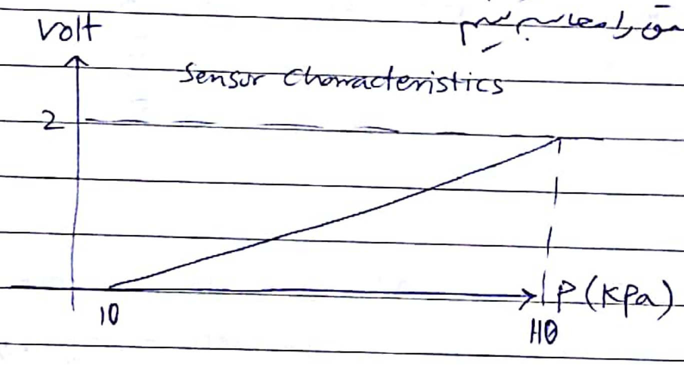
→ Cost

→ Size & weight limits

→ Power & Energy limits

→ Environment — دمای محیطی که پرنده در آن کار می‌کند ممکن است بسیار متغیر باشد

می‌خواهیم با وارد کردن فشار آب به MCU، عمق را محاسبه کنیم



$$P = 10 + \frac{V_{out}}{0.02} = 10 + 50 * V_{sensor}$$

عمق $Z = \frac{1}{\rho g}(P - P_0)$

$$Z = \frac{1000}{9.8}(P - 101.3)$$

voltage range : $+V_{ref}$ - $+V_{ref}$. فرض کنیم خروجی سنسور سیگنال دیجیتال ۱۰ بیتی است

دقت = step = $\frac{V_{ref} - (-V_{ref})}{2^{10}} = \frac{5}{1024} = 4.9\ mV$

measured voltage = ADC output * step = V_{sensor}

بدون استفاده از نرم‌افزار، روشی که استفاده می‌شد مثلاً این بود که فشار اندازه‌گیری شده به یک عقربه وصل می‌شود که عقربه خطا دارد
- خواندن عقربه توسط انسان خطا دارد
- عدد مورد نظر جایی ذخیره شده تا استفاده‌های آتی را داشته باشیم

Subject ..... Date .....

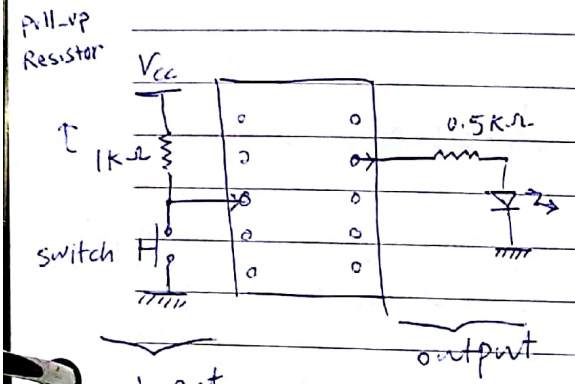
mbed.com

96/7/4 - فایل مطلب

↳ write and compile your code → .bin file in Download folder

connect the board and copy .bin file in it and reset the board

board independent program



ports

32 pins ← A

32 pins ← B

← C

D

E

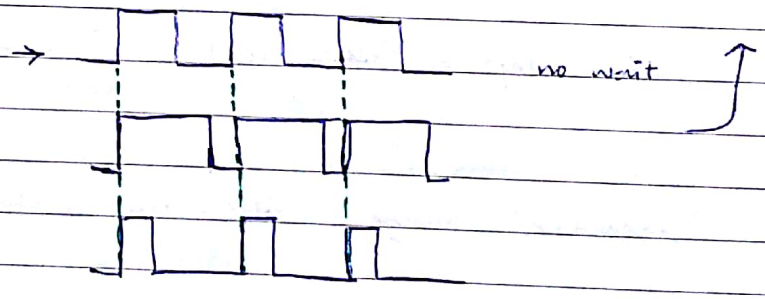
not in our board :)

blinking LED (hello world)

```
#include "mbed.h"
DigitalOut Ledpin(PTD2);
int main(){
    Ledpin = 1;
    while (true){
        Ledpin = 0;
        wait(0.5);
        Ledpin = 1;
        wait(0.5);
    }
}
```

PWM (pulse width modulation)

Simulate analog signal with digital signal, by changing duty cycle




```
#include "mbed.h"
PwmOut Ledpwm(PTD2);
int main(){
    Ledpwm.period(0.01);
    Ledpwm = 0.0;
    while (true){
        for (float val = 0.0; val < 1.0; val += 0.05){
            Ledpwm = val;
            wait(0.05);
        }
        for (float val = 1; val > 0; val -= 0.05){
            Ledpwm = val;
            wait(0.05);
        }
    }
}
```

HW1 → 7/10 → mail

Subject ..... Date .....

Reading Input

use internal pullup

```
#include "mbed.h"
DigitalIn SwitchOn(PTD3,PullUp);
DigitalOut Ledpin(PTD2);
int main(){
    Ledpin = 0;
    while(true){
        if(switchOn == true)
            Ledpin = 1;
        else
            Ledpin = 0;
    }
}
```

Project 0 (96/7/16) with documentation → mail

Each time you press the switch, led light increases 20%.

a) with pwm

b) without pwm

96/7/11 جلسه نهم

PTC1 — A5, PWM, I2C } multiplex { PTE30 — Analog out

Digital to Analog → 12 bit Analog to Digital → 16 bit

مثال: یک DAC دهبیتی طراحی کنید. اگر $V_{ref} = 2.7V$ و input = 260 باشد، output آن چقدر است؟

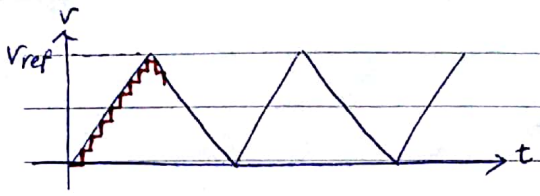
$$V_{out} = V_{ref} \times \frac{n}{2^{10}-1} = 2.7 \times \frac{260}{1023} = 0.68\,V$$

مثال: یک ADC طراحی کنید که 12 بیتی است و $V_{ref} = 3.3V$ باشد. اگر input=0.92 باشد خروجی چیست؟

$$\left\lceil \frac{V_{in}}{V_{ref}} \times (2^N - 1) \right\rceil = \left\lceil \frac{0.92}{3.3} \times (2^{12} - 1) \right\rceil = 1142$$

Subject ..... Date .....

فرکانس نوت های مختلف



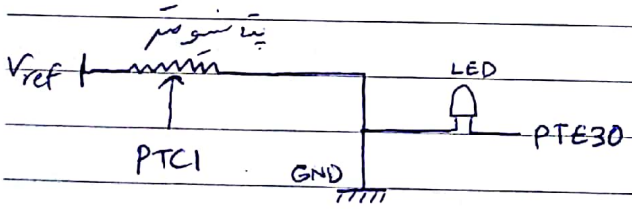
← خروجی را به شکل زیر می خواهیم تولید کنیم :

```
#include "mbed.h"
AnalogOut dac(PTE30);
int main(){
    while(1){                    Vref معادل
        for(float i=0.0; i<=1.0; i+=0.0001){
            dac = i;
            wait(0.0001); → تعیین کننده شیب نمودار  => تنظیم فرکانس
        }
        for(float i=1.0; i>=0.0; i-=0.0001){
            dac = i;
            wait(0.0001);
        }
    }
}
```

Project 1

چند نوت اول سمفونی نهم بتهوون را بنوازید :

← Analog output ، Analog Input



با تغییر مقاومت پتانسیومتر می خواهیم نور LED را تنظیم کنیم .

```
#include "mbed.h"
AnalogOut dac(PTE30);
AnalogIn adc(PTC1);
int main(){
    while(1){
        float A_val = adc;
        dac = A_val;
    }
}
```

Subject Embedded Sys. Date 96/7/17

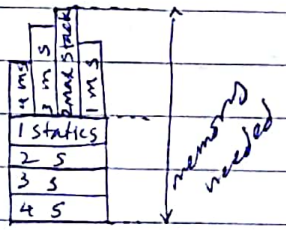
RTOS (Real Time Operating System)

↳ garantee the maximum response time for each task

in the RTOS:

- Task schedular → Support Interrupt
- Core Integrated RTOS services
  - ↳ Safe data sharing
  - ↳ Time management
- Optional Integrated RTOS Services
  - ↳ I/O abstraction → high level
  - ↳ Mem. management
  - ↳ File System
  - ↳ Network support
  - ↳ GUI

non-preemptive = run to completetion



→ memory needed for Non-preemptive statics model

↳ just one task is running at time

Interrupt code:

```
#include "mbed.h"
InterruptIn button(PTD3); Ticker time_up;
DigitalOut LED(PTD2);
DigitalOut flash(PTB2);
void flip() { LED = !LED; } → ISR (Interrupt Service Routine)
int main() {
  button.rise(&flip); time_up.attach(&flip, 1.0); // interrupt by time
  while(1) {
    flash = !flash;
    wait(0.25);
  }
}
```

08

Subject ..... Date .....

C as Implemented in Assembly Language

Programmer's world → Land of chocolate :)
processors' world → ?

→ Memory Requirements
in a program :
- Code
- Read only static Data
- Writable static Data
- Stack
- Heap

What goes where?
Can the information change?
How long the data need to exist? → reuse the memory

- statically allocated
- Automatically allocated → in stack → not reusable
- Dynamically allocated → scoped variables → usually short-live
  → runtime variables → usually short-live
  → saved in RAM → reuse

=> program memory use

variable { volatile → a variable may change from different sources
non volatile

```
A = B + C;
D = A + E;
```

variables should be loaded from memory to Registers and then after operation, memory data should be updated

if compilers don't update variables instantly, and it can be changed from two sources, they might not be synched.

```
A = B + C;
  B → R1 ✓
  C → R2 ✓
  R1 + R2 → R3
  R3 → A ✓

D = A + E;
  A → R4 ✓
  E → R5 ✓
  R3 + R5 → R6
  R6 → D ✓
```

↳ compiler will ommit these two steps for optimization

Communication ✓ with memory

→ Solution :
volatile int A;

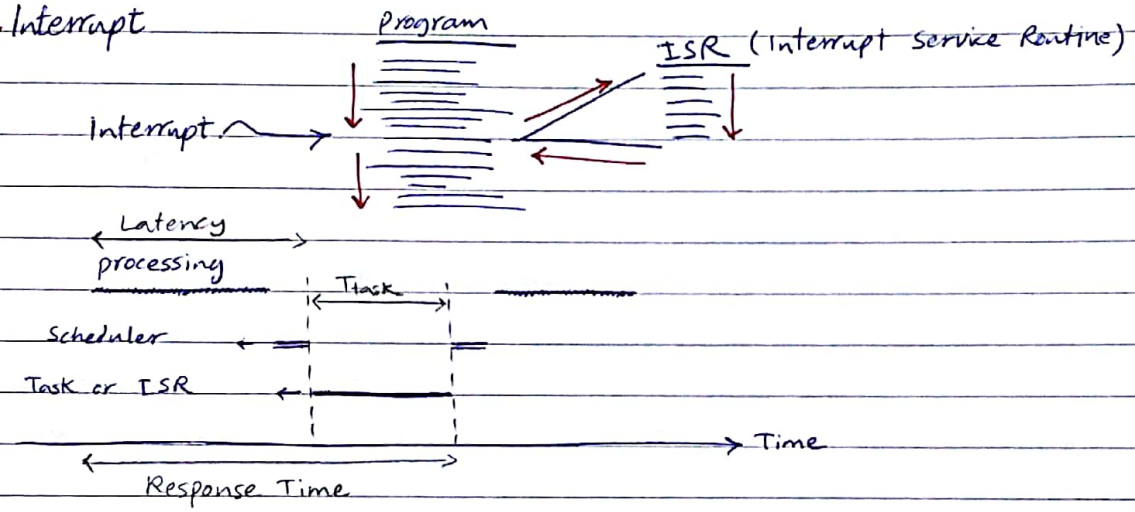
Software Design Basics

→ Concurreney ≠ parallelism parallel ⇒ concurrent

→ Software Engineering for Embedded Systems

Concurrency

→ Interrupt



96/7/17

So, if an interrupt or something change the value of A in memory befor the second operation (D=A+B) So with the optimization compiler did for us, the result would be Incorrect

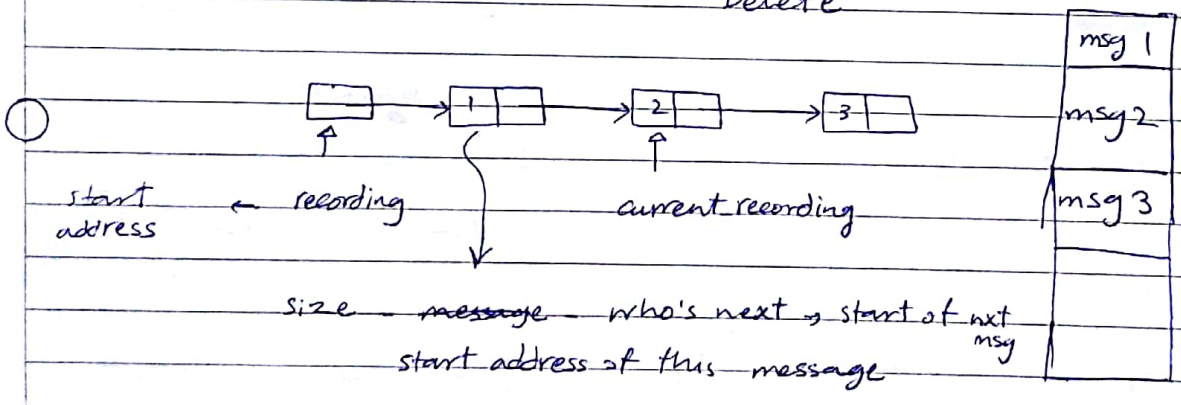
to protect critical section → semaphor, test & set → Atomic running of C.S.

↳ Interrupts will be saved to be serviced later.

Slide 11 → find the problems & solutions (HW)

↳ Time is updated in each second and a function may read the time data at any time

Example: Voice Recorder → Record, Forward, Delete — Slide 36-40



all messages are stored in memory consicutively

96/7/25

Cortex-M0

Core Architecture → load/store → store, load

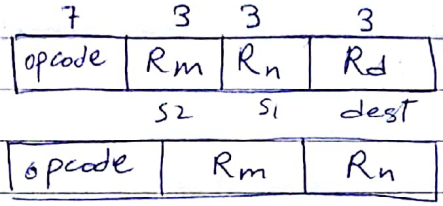
⇐ عجیب‌ترین بخش‌ها از لحاظ سرعت، بعد از حافظه است.

general purpose تا 13 ، (تا 16) ← تعداد رجیستر

Subject ..... Date .....

Link Register — function call — نگهداری آدرس بازگشت

instruction set :



ADD R3, R1, R2 ADD R3, R1

ADDS R3, R1

↳ updates flages (sets Flages)

* 128KB Flash , 16KB SRAM

4GIG → فضای آدرس دهی

Peripheral → برای آدرس دهی وسایل جانبی مثل DAC، timer ...

port Data Output Register → PDOR

port Data Input Register → PDIR

LD R2, R3, R4 (base ↰ ↱ Index)

R2 ← [R3+R4]

(memory map) : نواحی پورت مثل نواحی حافظه است چون آدرس دهی هایی است.

→ Addressing modes:

offset (displacement) LD R2, R3, #40 : R2 ← [R3+40]

index

→ Flags : Negative , Zero, Carry , oVerflow

Load literal value into register LDR Rd, value

- Decimal
- Hex
- character
- string

(پشته stack) Subject Embedded sys. Date 96/7/25

→ Stack Access

mem[SP-4] ← Ra

Rb ← mem[SP+4]

PUSH {Ra, Rb, ...} PUSH {R1-R4, LR}

POP {Ra, Rb, ...}

LSR (Logical Shift Right)

LSL Rd, Rn ; Rd ← Rd << Rn

ASL (Arithmatic) ~→ علامت را در نظر می‌گیرد برخلاف Logical

→ Program Flow Control Register

(branch) B <label> ±2046 (relative) فقط برای دستورات پرش

BX Rn → function call استفاده می‌شود

B <Cond> <label>

(branch & link) BL <label> → آدرس دستور بعدی در LR ذخیره می‌شود

Subject Embedded Sys. Date 7/30

Project 2 — 96/8/19

Task1 & Task2 → Concurrent

Task3 & Task4 → Interrupt Service

HW3 → 96/8/7

```
#include "mbed.h"
DigitalOut LedPin(PTD2);
int main(){
    LedPin = 1;
    while(1){
        LedPin = 0
        wait(0.5);
        LedPin = 1;
        wait(0.5);
    }
}
```

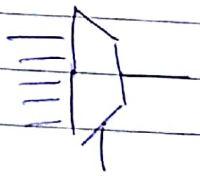
کامپایلر mbed این کد را برای برد مخصوص کد مخصوص می‌کند و کامپایل می‌کند.

کامپایلر زبان C است (عمومی) KEIL

↓ without mbed Compiler

1 activate Clock for Port D
2 Define Port D as GPIO (general purpose digital)
3 Define Port D as output
4 output 0, 1
5 program wait or delay

SCGC5 → bit 12 → Port D Clock




```
#define SIM_SCGC5_PORTD_MASK 12
① SIM→SCGC5 |= 1 << SIM_SCGC5_PORTD_MASK
```

Pin Control Register → برای کنترل عملکرد پورت، کنترل وقفه‌ها و ...

Interrupt Status Flag Register (ISFR)(32bit) نشان می‌دهد وقفه از کدام پین آمده

② PORTD→PCR[LedPin] |= PORT_PCR_MUX(1) → برای MUX_SEL یک قرار می‌دهد

PTOR → Toggle output Register

Subject Embedded Sys. Date 30

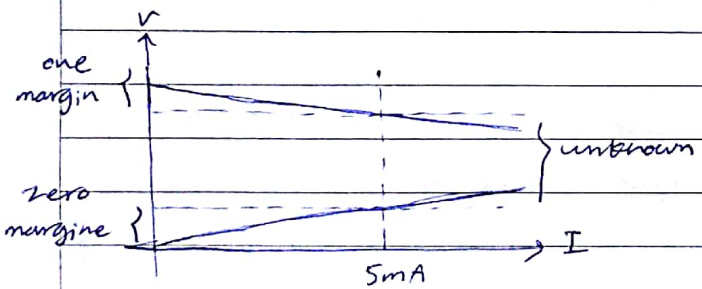
```
/* C program to have a blinking LED */

#include "MKL25Z4.h"
#define LedPin 2
// enable clock
SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK;
// select GPIO
PORTD->PCR[LedPin] &= ~PORT_PCR_MUX_MASK;
PORTD->PCR[LedPin] |= PORT_PCR_MUX(1);
// LedPin output
PTD->PDDR |= MASK(LedPin); // Data Direction Register
// LED on
PTD->PDOR = MASK(LedPin);
// Delay
while(1){
  delay(cycles);
  PTD->PDOR = ~MASK(LedPin);
  delay(cycles);
  PTD->PDOR = MASK(LedPin);
}
void Delay(int cycles){ while(cycles--); }
```

ابتدا MUX را صفر می‌کنیم و بقیه بدون تغییر

0 input
1 output

MASK(n) → 31 ... n ... 0 → 00...010...0 (32 bit)



جریان

برای راه‌اندازی جریانی از Pin می‌گیریم

pull enable → pull select → up or down
→ set to 1 to drive more current
→ available for some pins

Subject .... Date ....

Interrupt

سیگنالی که یک رویداد خارجی را اطلاع می‌دهد
باعث می‌شود برنامه اصلی قطع شود و ISR اجرا شود سپس به برنامه اصلی باز گردد.

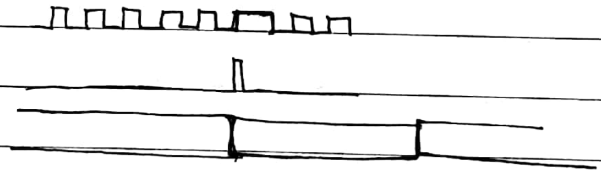
~~disable_Interr~~ } __disable_irq → کل وقفه ها را غیرفعال می کند
mbed در کامپایلر ← { __enable_irq

```
                    → PTA5
InterruptIn   int_pin(SW2);
int_pin.rise(&ISR);      →  برای rising edge ، ISR صدا می شود
int_pin.rise(NULL);      →  غیرفعال کردن وقفه (بصورت انتخابی)

NVIC_SetPriority(PORTD_IRQn, (128));  → اولویت وقفه های پورت D را تعیین می کند
Nested Vector Interrupt Control          چهار حالت دارد
0x00   0x40   0x80   0xC0   ←   0   64   128   192
```

همه ی پورت ها در KL25Z نمی توانند وقفه تولید کنند. فقط پورت D، A از وقفه پشتیبانی می کنند.

```
PTB1 → DBG_Main
PTB0 → DBG_ISR
PTD6 → Swith
```



falling edge ← وقتی کلید فشرده شود به زمین وصل می کند ~~...~~ قرمز روشن است

```
int main() {
    init_switch();        → clock, GPIO, input
    init_RGB_LEDs();      ↑        output
    init_debug_signals(); ↑        output
    __enable_irq();
    while (1) {                Toggle
                                 ↑
        DEBUG_PORT->PTOR = MASK(DBG_MAIN_POS);
        Control_RGB_LEDs(Count&1, Count&2, Count&4);
    }
}
```

↳ counter : _ _ _ → Control LEDs

Subject: سیستم‌های نهفته و بی‌درنگ Date: ۲/ آبان

init_switch()   PTD6
  // enable clock   GPFO   pull up
  PORTD->PCR[SW_POS] |= PORT_PCR_MUX(1) | PORT_PCR_PS_MASK
                        | PORT_PCR_PE_MASK → pull enable
                        | PORT_PCR_IRQC(0x0a);
              interrupt on falling edge ~~1010~~ (1010) لـ

  // set priority
  // clear pending Interrupt

}

PORTD_IRHandler() {   وقفه آیا حاصل از پورت D پین 6 بوده یا نه چک می‌کنیم

        ⋮   Interrupt ~~service~~ Status Flag Register

    if(PORTD->ISFR & MASK(SW_POS)) {
        counter ++;
    }

    // clear statuse flag
    PORTD_ISFR = 0xFFFFFFFF;

project 3 → برای دو هفته بعد → Keil development environment   ۷ آبان

→ critical section

تمرین دوم : همزمانی با ورود سانس و نشان دادن زمان → در واقع یک داده مشترک داریم که باید حفاظت شود

راه ① : غیرفعال کردن وقفه تایمر در ابتدای تابع نشان دادن و فعال کردن آن در انتهای آن

راه ② :

lock mutex   lock mutex

Display ←

unlock mutex   unlock mutex

راه ③ : test & set

راه ④ : swap instruction

این راه حل ها پشتیبانی سخت افزاری می‌خواهد

Subject ..... Date .....

LDREX → Load Register Exclusive
STREX → Store Register Exclusive
→ می‌توانیم مجموعه این دو دستور را اتمیک بگیریم.

وقفه هایی که در حالت عادی توانایی درخواست وقفه دارند Vector Int emupt

push {r4, r1} ≠ آدرس برگشت Interrupt با آدرس برگشت

زمان اجرای وقفه زمان مفیدی محسوب نمی‌شود اگرچه لازم است مثل زمان اجرای کرنل عمل

utilization = زمان اجرای برنامه‌ها / کل زمان
مثل OS و وقفه و برنامه

State Machine

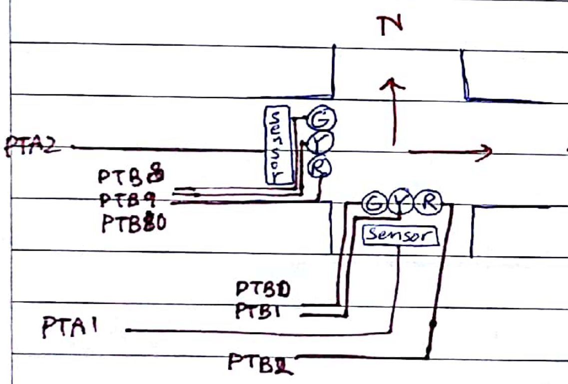
→ Finite state machine (FSM) → Abstraction for system Description

inputs → [System / logic / states] → outputs

→ 5-Tuples:
1. set of inputs
2. set of outputs
3. set of states → Graph (STG)
4. state transition → Table / matrix
5. Output Description (Determination)
6. Initial state

- Moore FSM → next state = f_1(current state, Input)
- Moore FSM → Output = f_2(current state) ⇒ رابطه یک به یک بین حالات و خروجی
- Mealy FSM → next state = f_1(current state, Input)
- Mealy FSM → Output = f_3(current state, Input)

مثال: کنترل عبور و مرور چهارراه



PTA1	PTA2	Status
0	0	no cars
0	1	cars on E road
1	0	cars on N road
1	1	cars on both roads

در وضعیت چراغ زرد و سبز باید مدتی بمانیم ← 5s برای زرد و 30s برای سبز

set of states: goN , waitN , goE , waitE

	goN	waitN	goE	waitE	
PTB 10:8	100	100	001	010	outputs
PTB 2:0	001	010	100	100	
Time	30	5	30	5	

Inputs (N E)

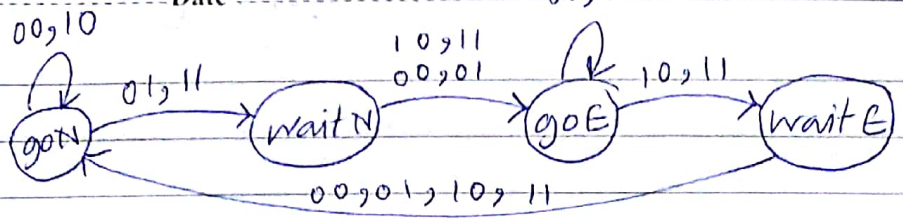
Inputs	goN	waitN	goE	waitE
0 0	goN	goE	goE	goN
0 1	waitN	goE	goE	goN
1 0	goN	goE	waitE	goN
1 1	waitN	goE	waitE	goN

State Transition Table → مستقیماً می‌تواند به کد تبدیل شود

Starvation تغییر وضعیت برای پیشگیری از

10 9 8
0 0 1

Subject ..... Date .....



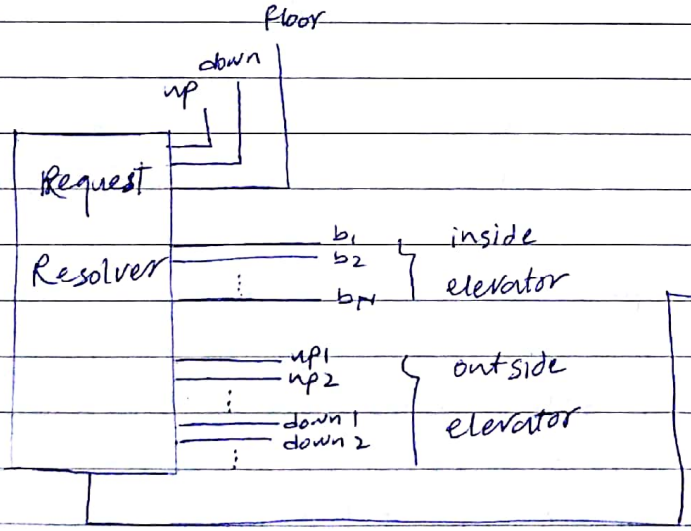
State Transition Graph

PTB 10:8 PTB 2:0

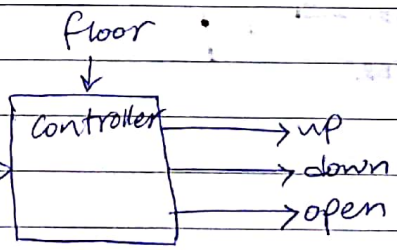
100 001 → 0x401

```
SType FSM[4] = {{0x401, 30, {goN, waitN, goN, waitN}},
                {0x402, 5, {goE, goE, goE, goE}},
                {},
                {}};

CS = goN; // initialize current state
while(1){
    PTB->PDOR |= FSM[CS].out;
    delay(FSM[CS].Time);
    Input = (PTA->PDIR & 0x06) >> 1;
    CS = FSM[CS].Next(Input);
}
```



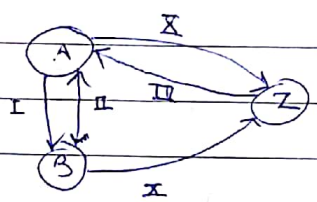
elevator Control



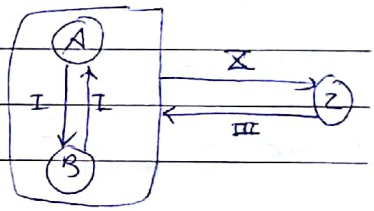
↳ Sequential program model
↳ Finite state machine model

State set: Idle, goingup, goingdown, dooropen
outputs: up, down, timer, open
inputs: uprequest, down_request, timer reaches end time or not

→ Hierarchical FSM
→ Concurrent FSM



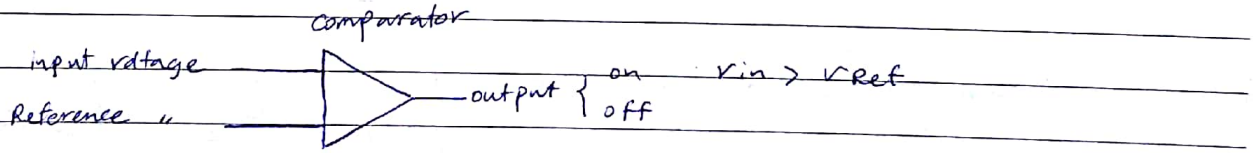
CFSM

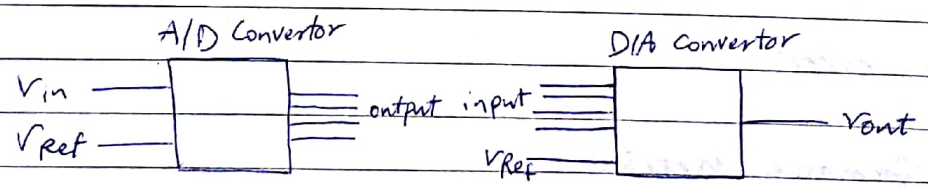


HFSM

16 آبان

Converting between Analog and Digital Values





Example 1. 10 bit DAC ($V_{Ref} = 2.7v$)
input code = 0x104 => $V_{out} = ?$

$$V_{out} = V_{Ref} \times \frac{n}{2^N - 1} = 2.7 \times \frac{0x104}{0x3FF} = 2.7 \times \frac{260}{1023} = 0.68\ v$$

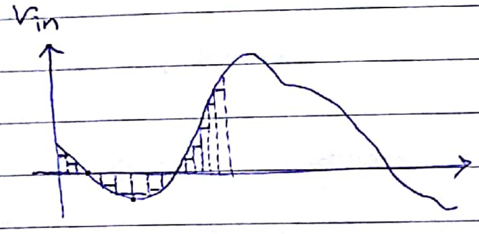
Example 2. 12 bit DAC ($V_{Ref} = 3.3v$)
$V_{out} = 1.43v$ => input code = ? $= \frac{1.43}{3.3} \times (2^{12} - 1) \simeq 1779 = 0x6E$

Subject .... Date ....

Example 3. 8 bit DAC ($V_{Ref} = 3\,V$) resolution ?

$$\text{resolution} = \frac{V_{Ref}}{2^N - 1} = \frac{3.0}{255} = 0.0117\,V$$

ADC

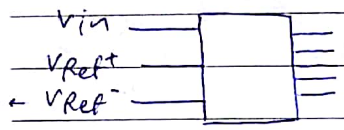


→ هر نمونه یک کد N بیتی تبدیل می شود

نرخ نمونه برداری از سیگنال مشخص است

↳ بالاترین فرکانس = f_m Sampling Frequency $= 2 f_m$ ↲

↳ فاصله دو نمونه متوالی: Interval between samples $= \frac{1}{2 f_m}$

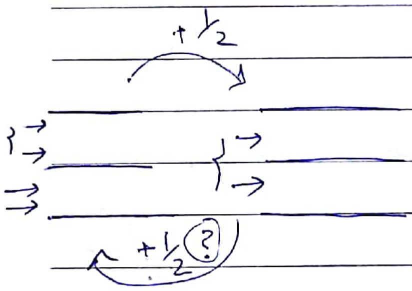


نصف واحد دیجیتال

$$n = \left\lfloor \frac{(V_{in} - V_{Ref}^-)(2^N - 1)}{V_{Ref}^+ - V_{Ref}^-} + \frac{1}{2} \right\rfloor$$

$V_{in} = V_{Ref}^+ \Rightarrow n = 111\cdots1$

$V_{in} = V_{Ref}^- \Rightarrow n = 000\cdots0$



$$V_{in\_min} = \left(\frac{V_{Ref}^+ - V_{Ref}^-}{2^N - 1}\right) \times \left(n - \frac{1}{2}\right)$$

نصف واحد آنالوگ

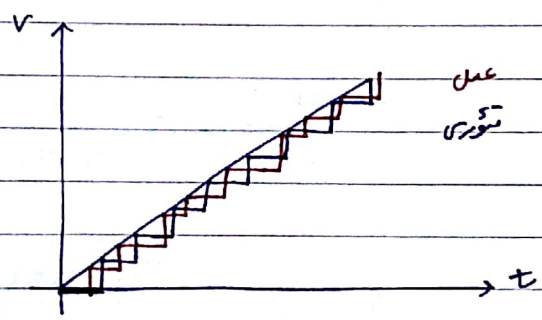
$$V_{in\_max} = \left(\frac{V_{Ref}^+ - V_{Ref}^-}{2^N - 1}\right) \times \left(n + \frac{1}{2}\right)$$

وقتی n (Digital code) مشخص باشد می توانیم تعیین کنیم ولتاژ ورودی در چه بازه ای بوده است و مقدار دقیق آن را نمی دانیم

quintization error

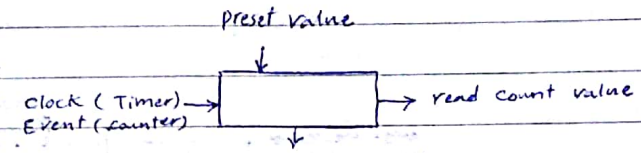
ADC Performance Metric

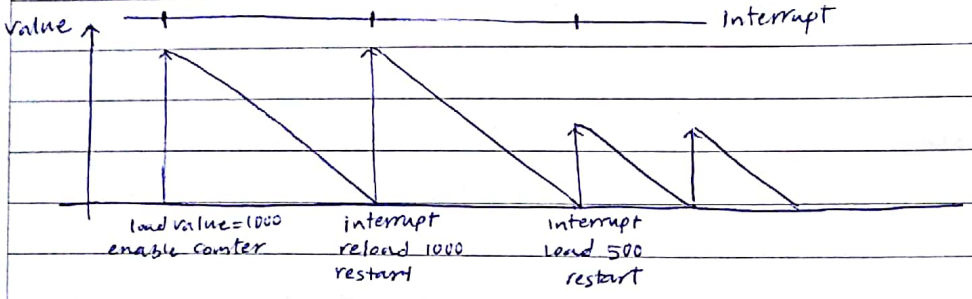
- INL (Integral NonLinear) → تفاوت بین تئوری و عمل (جمع شونده)
- DNL (Differential NonLinear) → هر نمونه را جداگانه بررسی می کند



Timer

Counter/Timer :





down counter

PIT Module

Generate interrupt every T seconds → load value = round ($T \times f_{count}$)

↳ Interrupt every 137.41 ms → load value = $137.41^{ms} \times 24^{MHz} = 3297840$

↳ Interrupt with a freq of 91 Hz → load value = $\frac{1^{s}}{91} \times 24^{MHz} = 263735$

Example : write a program to generate interrupt with a frequency of 315 Hz (watch video)

Example : Stopwatch → Resolution = 100 ms , update LCD screen every 10 ms

$\frac{10ms}{100ms} = 100$ → every 100 tik , call update screen

ISR

counter++;

if (counter == 100){

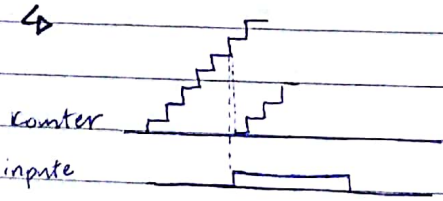
setUpdateFlag();

counter = 0; }

load value = $10^{ms} \times 24^{MH} = 24000$

→ TPM (Timer pwm Module)

- time measurement → capture time



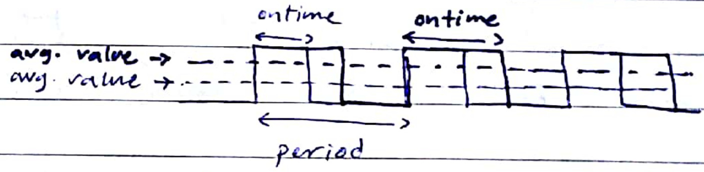
- Counting event or keeping time → input clock / event

↳ ~~generate~~ pw

Subject .... Date ....

Generate PWM Signal → motor speed control, light dimmer, ...

PWM _ modulation frequency



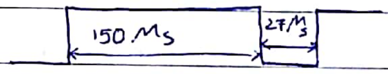
Time control → amplitude control

period

on-Time = pulse width

duty cycle = $\frac{\text{on-Time}}{\text{period}}$

Example:



periods = (150+27) Ms x 24 MH = 4248 cycle

pulse width = 150 Ms x 24 MH = 3600 cycle

* * *

مقدمه:

اکثر embedded system ها سیستم عامل ندارند چراکه معمولاً یک تسک انجام می‌دهند که خیلی هم پیچیده نیستند. ولی سیستم های پیچیده تر مثل دستگاه های صنعتی، اتوبوس ها، ماهواره ها و ... کارهای پیچیده انجام می‌دهند که بیش از یک تسک انجام می‌دهند و Operating system نیاز دارند ولی OS آنها سبک است و توانایی های چندانی ندارند.

RTOS (Real Time Operating System) _ RTX

- RTX Kernel
- Task
- Scheduling
- Task Switching
- Starting RTX
- Creating Tasks
- Task management
- Time management
- Communication between tasks
- Synchronization

Application programming Interface (API)

Subject: سیستم‌های نهفته و بی‌درنگ   Date: ۲۱ آبان

objects		services
Tasks		Task management
events		Time management
semaphore		memory management
mutex	(Schedular)	ISR support
message		⋮
mail box		

Schedular runs program code as tasks. Communication is accomplished by objects.

procedure

```
unsigned int procedure(void){
    ═══════
    return(value);
}
```

Task

```
_task void Task(void){
    for(;;){
        ═══      ← بدنه اصلی تسک
    }
}
```

a function is expected to return.
a RTX task contains an endless loop

Scheduling

- Round Robin → default (preemptive without priority) { → Time slicing, → first come first served }
- priority scheduling, preemptive
- Cooperative scheduling → در این زمان‌بندی تعدادی تسک با اولویت یکسان داریم و هستند؛ یک تسک می‌تواند در انتها تسک دیگری را بیدار (ready) کند یا در زمان دیگر Run to complete نوبت را به بعدی بدهد (داوطلبانه نوبت را از دست می‌دهد) زمان

۲۳ آبان

→ Launching RTX

```
void main(){
    // Do any C code you want
    OS_Sys_init(task1);
}
```

OS_Sys_init(task1) ↓ ایجاد اولین تسک و راه‌اندازی سیستم عامل

هر process، process دیگری بوجود می‌آورد جز اولی. ↓ برنامه در حال اجرا

↳ Launch RTX, but only starts the first task. After RTX initialization, Control is passed to task1. task1 has a default priority.

Subject ..... Date ۲۳ آبان

→ Creating tasks

the first task started is used to create additional tasks.

```
__task void task1() {                    priority
    tskID2 = OS_tsk_create(task2, 0x10);
    tskID3 = os_tsk_create(task3, 0x10);
    os_tsk_delete_self(); ~~> api for system call
}
```

هر تسک یک ① stack و یک ② Task control block دارد.

هر process یک بخش کد برنامه، پشته، ... دارد و یک process control block که وقتی سیستم عامل می‌خواهد process را اجرا کند برایش PCB ایجاد می‌کند برایش state آن را شامل رجیسترها program counter، stack pointer، اطلاعات resource های مورد استفاده، اولویت و ... نگه می‌دارد. وقتی process ای می‌خواهد اجرا شود این اطلاعات باید بود شود وقتی می‌خواهد سوئیچ شود این اطلاعات باید update شوند.

وقتی مثلا برنامه word را باز می‌کنیم، در واقع یک process در حال اجرا بوده و command ها را گرفته است و یک برنامه دیگر را بالا آورده است.

reset کردن وضعیت را به یک حالت default، یک state معلوم می‌برد که از آن تعدادی از رجیسترها مقدار مشخص می‌گیرند و از آن به بعد process های دیگر می‌توانند اجرا شوند.

Task control block هم اطلاعات مشابه اطلاعات PCB نگه می‌دارد برای thread ها هم دلایل مشابهی داریم

Subject .... Date ....

اگر تسک ها روی منابع مشترک بخواهند عملیات انجام دهند نیز به Synchronization داریم.

↲ مثلاً در یک فایل می‌خوانند و می‌نویسند یا مثال producer consumer

↲ منتظر وقفه شویم
با بافر محدود کار کنیم
دسترسی انحصاری بدهیم.

تبادل داده بین تسک ها هم معمولاً با صورت message passing و گرد روی از shared mem. هم استفاده می‌شود.

↲ روی کدام flag های می‌خواهیم صبر کنیم.

```
OS_RESULT os_evt_wait_or(unsigned short wait_flags, unsigned short timeout);
```
(and — بالای or)

↳ event management

یک تسک در یک loop انتظار است تا تعدادی فلگ ست یا ریست شوند (بلاک شده) تا ← کار را ادامه دهد.

```
void os_evt_set(unsigned short event_flags, OS_TID task);
```
(Task ID — بالای OS_TID)

↳ تسک منتظر مانده را بیدار می‌کند.

مثال

(0 --- 0011) یعنی flag اول و دوم (=3، 16 bit)

```
result = os_evt_wait_and(0x0003, 500)
```
→ timeout شد.

```
if(result == OS_R_TMO)
    ~~~~
else
    ~~~~

__task void task1(){
    ~~~
    os_evt_set(0x0003, tsk2);
    ~~
}
```