

Thiết kế hệ thống nhúng-EE4251

GV: ĐÀO ĐỨC THỊNH

KHOA TỰ ĐỘNG HÓA-TRƯỜNG ĐIỆN ĐIỆN TỬ-ĐHBK HN

Lập trình C cho hệ thống nhúng

Tại sao lại là C ??

Nó là một ngôn ngữ trung gian, có cả đặc điểm của ngôn ngữ bậc cao và ngôn ngữ bậc thấp.

Rất hiệu quả

Thông dụng, dễ hiểu.

Có các chương trình dịch cho các uP hệ nhúng 8 bit đến 32 bit.

Có nhiều nhân viên lành nghề.

Sách, các khoá học, ví dụ và các trang web cho ngôn ngữ này rất nhiều.

Yêu cầu lập trình C cho MCS51

Đã biết ngôn ngữ lập trình: Java, C++, Basic..

Có cơ sở về ngôn ngữ C.

Sử dụng Keil C Compiler.

Phần mềm với “Super loop”

Vấn đề:

Một môi trường phần mềm tối thiểu cần để tạo ra một chương trình C cho hệ nhúng là gì?

Phần mềm với “Super loop”

Giải pháp:

```
void main(void)
{
    /* Prepare for task X */
    X_Init();
    while(1) /* 'for ever' (Super Loop) */
    {
        X(); /* Perform the task */
    }
}
```

Phần mềm với “Super loop”

Ưu điểm:

- Ưu điểm chính của “Super loop” là đơn giản. Dễ viết, gỡ rối, kiểm tra, duy trì hoạt động.
- Hiệu quả cao, thực hiện với phần cứng tối thiểu.
- “Super loop” rất nhỏ gọn.

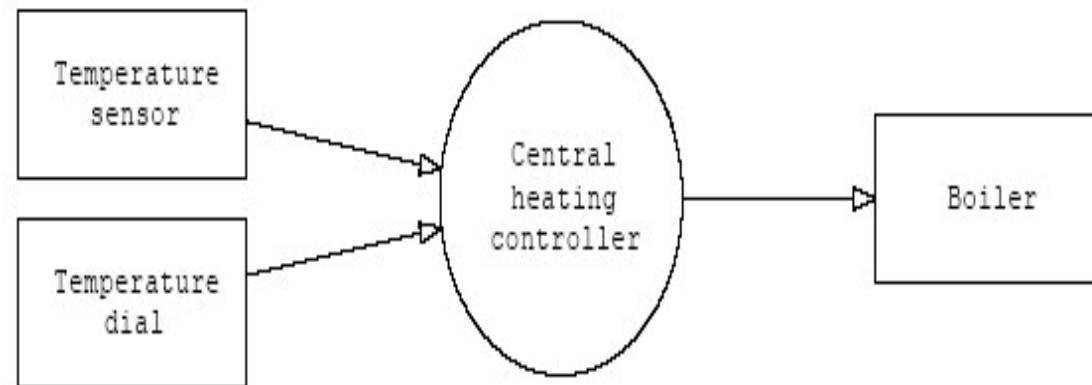
Phần mềm với “Super loop”

Nhược điểm:

- Nếu ứng dụng yêu cầu độ chính xác thời gian thì nó không đáp ứng được và không mềm dẻo.
- Lúc nào cũng hoạt động ở công suất tối đa. Tiêu thụ nhiều điện.

"Hello, Embedded World"

Example: Central-heating controller



"Hello, Embedded World"

```
void main(void)
{
    /* Init the system */
    C_HEAT_Init();
    while(1) /* 'for ever' (Super Loop) */
    {
        /* Find out what temperature the user requires (via the user interface) */
        C_HEAT_Get_Required_Temperature();
        /* Find out what the current room temperature is(via temperature sensor) */
        C_HEAT_Get_Actual_Temperature();
        /* Adjust the gas burner, as required */
        C_HEAT_Control_Boiler();
    }
}
```

Keil C compiler

Development Tools	Support Microcontrollers, Description
C51 Compiler A51 Macro Assembler BL51 Linker/Locator	Development Tools for classic 8051. Includes support for 32 x 64KB code banks.
C51 Compiler (with OMF2 Output) AX51 Macro Assembler LX51 Linker/Locator	Development Tools for classic 8051 and extended 8051 variants (like the Dallas 390). Includes support for code banking and up to 16MB code and xdata memory.
CX51 Compiler AX51 Macro Assembler LX51 Extended Linker/Locator	Development Tools for the Phillips 80C51MX Supports up to 16MB code and xdata memory.

Keil C compiler

Tạo ra một Project với Keil C

Keil C compiler

Vùng nhớ:

- Bộ nhớ chương trình: code
- Bộ nhớ dữ liệu trong:
 - data.
 - idata
 - bdata
- Bộ nhớ dữ liệu ngoài:
 - xdata
 - pdata
- Bộ nhớ xa (far memory): far

Keil C compiler

- Bộ nhớ xa (far memory): far
- Bộ nhớ thanh ghi chức năng đặc biệt.

Keil C compiler

Mô hình tổ chức bộ nhớ:

- Small: các biến chứa trong bộ nhớ trong (data).
- Compact: các biến có thể chứa trong 1 trang 256 byte của bộ nhớ ngoài (pdata)
- Large: các biến chứa trong 64kbyte bộ nhớ ngoài.

Keil C compiler

Kiểu của bộ nhớ:

Memory Type	Description
code	Program memory (64 KBytes); accessed by opcode MOVC @A+DPTR.
data	Directly addressable Internal data memory; fastest access to variables (128 bytes).
idata	Indirectly addressable Internal data memory; accessed across the full Internal address space (256 bytes).
bdata	Bit-addressable Internal data memory; supports mixed bit and byte access (16 bytes).
xdata	External data memory (64 KBytes); accessed by opcode MOVX @DPTR.
far	Extended RAM and ROM memory spaces (up to 16MB); accessed by user defined routines or specific chip extensions (Philips 80C51MX, Dallas 390).
pdata	Paged (256 bytes) external data memory; accessed by opcode MOVX @Rn.

Keil C compiler

Example:

```
char data var1;  
char code text[] = "ENTER PARAMETER:";  
unsigned long xdata array[100];  
float idata x,y,z;  
unsigned int pdata dimension;  
unsigned char xdata vector[10][4][4];  
char bdata flags;
```

Keil C compiler

Kiểu của dữ liệu:

Data Types	Bits	Bytes	Value Range
bit †	1		0 to 1
signed char	8	1	-128 to +127
unsigned char	8	1	0 to 255
enum	8 / 16	1 or 2	-128 to +127 or -32768 to +32767
signed short	16	2	-32768 to +32767
unsigned short	16	2	0 to 65535
signed int	16	2	-32768 to +32767
unsigned int	16	2	0 to 65535
signed long	32	4	-2147483648 to +2147483647
unsigned long	32	4	0 to 4294967295
float	32	4	±1.175494E-38 to ±3.402823E+38
abit †	1		0 or 1
afir †	8	1	0 to 255
afir16 †	16	2	0 to 65535

Keil C compiler

Đặt biến vào vị trí bộ nhớ:

type memory_space variable_name _at_ constant;

Keil C compiler

```
struct link
{
    struct link idata *next;
    char code *test;
};

Struct link list idata _at_ 0x40; /* list at idata 0x40 */
char xdata text[256] _at_ 0xE000; /* array at xdata 0xE000 */
int xdata i1 _at_ 0x8000; /* int at xdata 0x8000 */

void main ( void ) {
    link.next = (void *) 0;
    i1 = 0x1234;
    text [0] = 'a';
}
```

Keil C compiler

Con trỏ:

```
char *s; /* string ptr */  
int *numptr; /* int ptr */  
long *state; /* Texas */
```

Keil C compiler

```
char * xdata strptr; /* generic ptr stored in xdata */  
int * data numptr; /* generic ptr stored in data */  
long * idata varptr; /* generic ptr stored in idata */
```

Keil C compiler

```
char data *str; /* ptr to string in data */  
int xdata *numtab; /* ptr to int(s) in xdata */  
long code *powtab; /* ptr to long(s) in code */
```

Keil C compiler

```
char data * xdata str; /* ptr in xdata to data char */  
int xdata * data numtab; /* ptr in data to xdata int */  
long code * idata powtab; /* ptr in idata to code long */
```

Keil C compiler

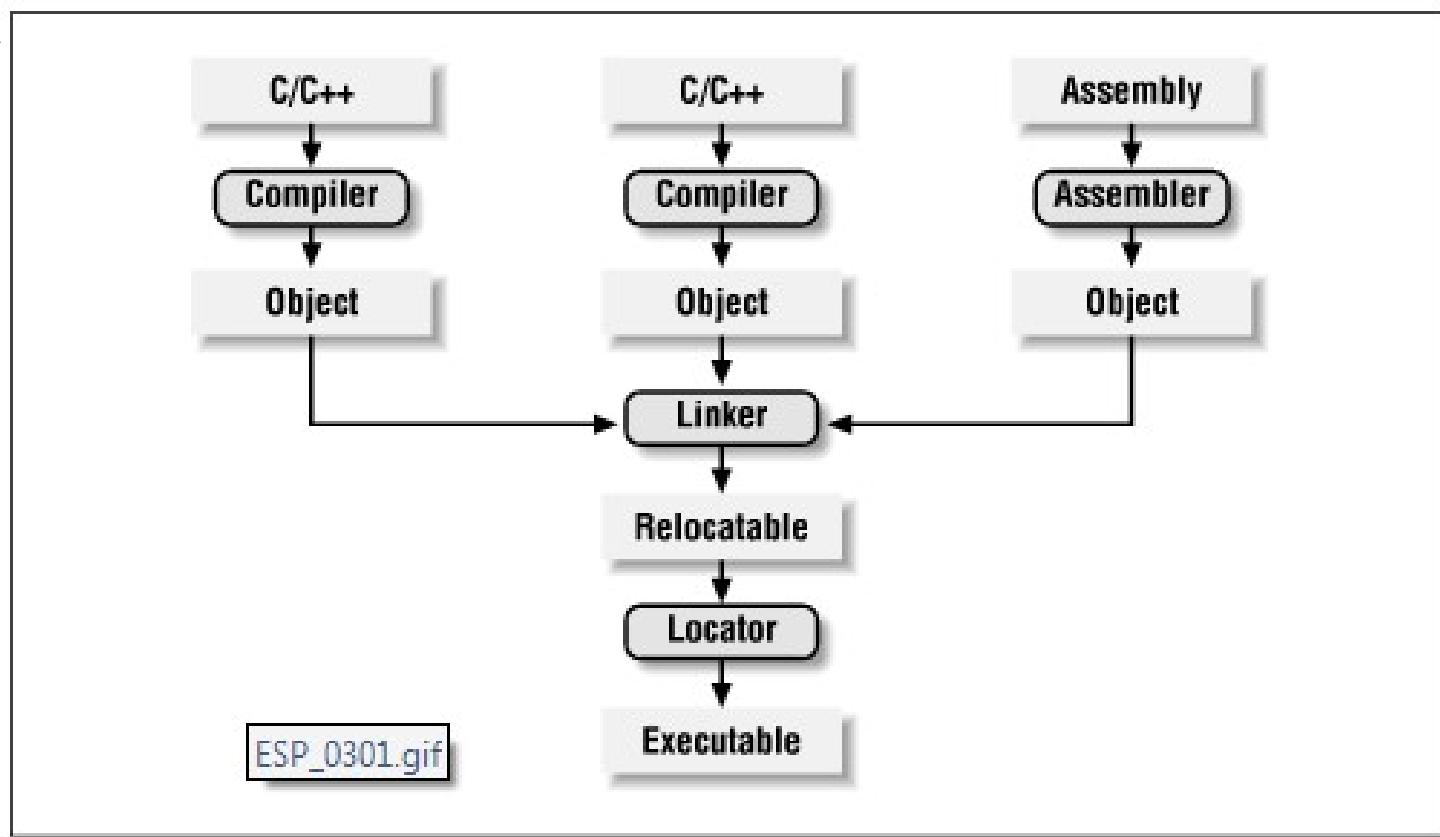
Hàm: C cho MCS51 có một số mở rộng cho hàm như sau:

- Chỉ ra một hàm sử dụng như ngắn.
- Chọn bank thanh ghi.
- Chọn mô hình bộ nhớ.
-

Keil C compiler

return_type funcname (args) {small | compact | large} reentrant interrupt n using n

Keil C compiler



Hệ thời gian thực

Hệ thời gian thực

Trong phần này ta xem xét hệ nhúng yêu cầu độ chính xác về thời gian.

Hệ thống hoạt động Real-time.

Hệ thời gian thực

```
bit SWITCH_Get_Input(const tByte DEBOUNCE_PERIOD)
{
    tByte Return_value = SWITCH_NOT_PRESSED;

    if (Switch_pin == 0)
    {
        /* Switch is pressed */

        /* Debounce - just wait... */
        DELAY_LOOP_Wait(DEBOUNCE_PERIOD); /* POTENTIAL PROBLEM */

        /* Check switch again */
        if (Switch_pin == 0)
        {
            /* Wait until the switch is released. */
            while (Switch_pin == 0); /* POTENTIAL CATASTROPHE */
            Return_value = SWITCH_PRESSED;
        }
    }

    /* Now (finally) return switch value */
    return Return_value;
}
```

Hệ thời gian thực

Có hai vấn đề:

- Thời gian trễ không chính xác.
- Hệ thống chạy nhiều ứng dụng

Ta giải quyết bằng trễ phần cứng và ngắn thời gian thực

Hệ thời gian thực

```
/* Configure Timer 0 as a 16-bit timer */
TMOD &= 0xF0; /* Clear all T0 bits (T1 left unchanged) */
TMOD |= 0x01; /* Set required T0 bits (T1 left unchanged) */

ET0 = 0; /* No interrupts */

/* Values for 50 ms delay */
TH0 = 0x3C; /* Timer 0 initial value (High Byte) */
TL0 = 0xB0; /* Timer 0 initial value (Low Byte) */

TF0 = 0; /* Clear overflow flag */
TR0 = 1; /* Start Timer 0 */

while (TF0 == 0); /* Loop until Timer 0 overflows (TF0 == 1) */

TR0 = 0; /* Stop Timer 0 */
```

Hệ thời gian thực

```
#include <reg52.h>

sbit LED_pin = P1^5;
bit LED_state_G;

void LED_FLASH_Init(void);
void LED_FLASH_Change_State(void);

void DELAY_HARDWARE_One_Second(void);
void DELAY_HARDWARE_50ms(void);

/*.....
void main(void)
{
    LED_FLASH_Init();

    while(1)
    {
        /* Change the LED state (OFF to ON, or vice versa) */
        LED_FLASH_Change_State();

        /* Delay for approx 1000 ms */
        DELAY_HARDWARE_One_Second();
    }
}
```

Hệ thời gian thực

```
void LED_FLASH_Init(void)
{
    LED_state_G = 0;
}
```

Hệ thời gian thực

```
void LED_FLASH_Change_State(void)
{
    /* Change the LED from OFF to ON (or vice versa) */
    if (LED_state_G == 1)
    {
        LED_state_G = 0;
        LED_pin = 0;
    }
    else
    {
        LED_state_G = 1;
        LED_pin = 1;
    }
}
```

Hệ thời gian thực

```
void DELAY_HARDWARE_One_Second(void)
{
    unsigned char d;

    /* Call DELAY_HARDWARE_50ms() twenty times */
    for (d = 0; d < 20; d++)
    {
        DELAY_HARDWARE_50ms();
    }
}
```

Hệ thời gian thực

```
void DELAY_HARDWARE_50ms(void)
{
    /* Configure Timer 0 as a 16-bit timer */
    TMOD &= 0xF0; /* Clear all T0 bits (T1 left unchanged) */
    TMOD |= 0x01; /* Set required T0 bits (T1 left unchanged) */

    ET0 = 0;          /* No interrupts */

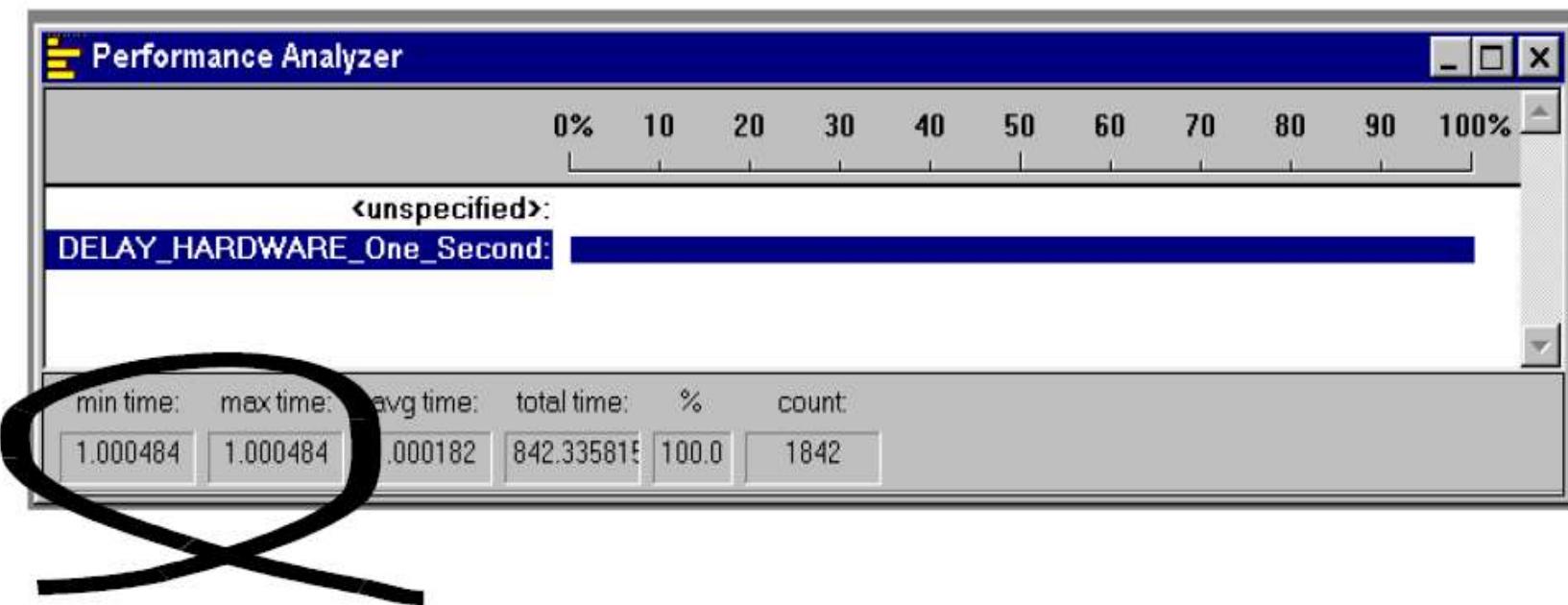
    /* Values for 50 ms delay */
    TH0 = 0x3C;      /* Timer 0 initial value (High Byte) */
    TL0 = 0xB0;      /* Timer 0 initial value (Low Byte) */

    TF0 = 0;          /* Clear overflow flag */
    TR0 = 1;          /* Start timer 0 */

    while (TF0 == 0); /* Loop until Timer 0 overflows (TF0 == 1) */

    TR0 = 0;          /* Stop Timer 0 */
}
```

Hệ thời gian thực



Hệ thời gian thực

Cơ chế time-out:

```
while (((ADCON & ADCI) == 0) && (Timeout_loop != 0))
{
    Timeout_loop++; /* Disable for use in hardware simulator */
}
```

Hệ thời gian thực

```
#include <reg52.H>
#include "TimeoutL.H"

/* Typedefs (see Chap 5) */
typedef unsigned char tByte;
typedef unsigned int tWord;
typedef unsigned long tLong;

/* Function prototypes */
void Test_Timeout(void);

/*-----*/
void main(void)
{
    while(1)
    {
        Test_Timeout();
    }
}
```

Hệ thời gian thực

```
/*-----*/  
void Test_Timeout(void)  
{  
    tLong Timeout_loop = LOOP_TIMEOUT_INIT_10000ms;  
  
    /* Simple loop timeout... */  
    while (++Timeout_loop != 0);  
}
```

Hệ thời gian thực

```
/* Configure Timer 0 as a 16-bit timer */
TMOD &= 0xF0; /* Clear all T0 bits (T1 left unchanged) */
TMOD |= 0x01; /* Set required T0 bits (T1 left unchanged) */

ET0 = 0;          /* No interrupts */

/* Simple timeout feature - approx 10 ms */
TH0 = PRELOAD_10ms_H; /* See Timeout.H for PRELOAD details */
TL0 = PRELOAD_10ms_L;
TF0 = 0; /* Clear flag */
TR0 = 1; /* Start timer */

while (((ADCON & ADCI) == 0) && !TF0);
```

Hệ điều hành cho hệ nhúng

Một chương trình đơn giản nhất

```
void main(void)
{
    /* Prepare run function X */
    X_Init();

    while(1) /* 'for ever' (Super Loop) */
    {
        X(); /* Run function X */
    }
}
```

Một chương trình đơn giản nhất

Chỉ thực hiện X() với một thời gian lấy mẫu nào đó.

Yêu cầu cho hệ: Ô-tô

Đo tốc độ 0.5s một lần.

Hiển thị quét 40 lần trên 1 s.

Tính toán độ mở van tiết lưu 0.5s một lần.

Biến đổi t-f 20 lần trên s

Lấy mẫu độ rung máy 1000 lần trên 1s.

Quét bàn phím 200ms

Sensor lấy mẫu 1 lần trên s

.....

Yêu cầu cho hệ: Ô-tô

Hệ điều hành.

Chương trình

```
void main(void)
{
    Init_System();

    while(1) /* 'for ever' (Super Loop) */
    {
        X();           /* Call the function (10 ms duration) */
        Delay_50ms(); /* Delay for 50 ms */
    }
}
```

Chương trình:

Độ chính xác thời gian.

Thời gian không thay đổi.

Ngắt thời gian (lõi của hệ điều hành cho HN)

```
#define INTERRUPT_Timer_2_Overflow 5
...
void main(void)
{
    Timer_2_Init(); /* Set up Timer 2 */
    EA = 1;          /* Globally enable interrupts */
    while(1);        /* An empty Super Loop */
}
```

Ngắt thời gian (lõi của hệ điều hành cho HN`

```
void Timer_2_Init(void)
{
    /* Timer 2 is configured as a 16-bit timer,
       which is automatically reloaded when it overflows

       This code (generic 8051/52) assumes a 12 MHz system osc.
       The Timer 2 resolution is then 1.000 µs

       Reload value is FC18 (hex) = 64536 (decimal)
       Timer (16-bit) overflows when it reaches 65536 (decimal)
       Thus, with these setting, timer will overflow every 1 ms */

    T2CON = 0x04; /* Load T2 control register */

    TH2    = 0xFC; /* Load T2 high byte */
    RCAP2H = 0xFC; /* Load T2 reload capt. reg. high byte */
    TL2    = 0x18; /* Load T2 low byte */
    RCAP2L = 0x18; /* Load T2 reload capt. reg. low byte */

    /* Timer 2 interrupt is enabled, and ISR will be called
       whenever the timer overflows - see below. */
    ET2    = 1;

    /* Start Timer 2 running */
    TR2    = 1;
}
```

Ngắt thời gian (lõi của hệ điều hành cho HN)

```
void X(void) interrupt INTERRUPT_Timer_2_Overflow
{
    /* This ISR is called every 1 ms */
    /* Place required code here... */
}
```

Interrupt source	Address	IE Index
Power On Reset	0x00	-
External Interrupt 0	0x03	0
Timer 0 Overflow	0x0B	1
External Interrupt 1	0x13	2
Timer 1 Overflow	0x1B	3
UART Receive/Transmit	0x23	4
Timer 2 Overflow	0x2B	5

Hệ điều hành

Mạch thời gian tự động nạp lại (auto reload)

Hệ điều hành

Có các yêu cầu:

- Chu kỳ thời gian dài.
- Không đòi hỏi xử lý từ CPU
- -> Timer 2

```
#include "Main.H"
#include "Port.H"
#include "Simple_EOS.H"

#include "X.H"

/* ----- */

void main(void)
{
    /* Prepare for dummy task */
    X_Init();

    /* Set up simple EOS (60 ms tick interval) */
    sEOS_Init_Timer2(60);

    while(1) /* Super Loop */
    {
        /* Enter idle mode to save power */
        sEOS_Go_To_Sleep();
    }
}
```

```

void sEOS_Init_Timer2(const tByte TICK_MS)
{
    tLong Inc;
    tWord Reload_16;
    tByte Reload_08H, Reload_08L;

    /* Timer 2 is configured as a 16-bit timer,
       which is automatically reloaded when it overflows */
    T2CON = 0x04; /* Load T2 control register */

    /* Number of timer increments required (max 65536) */
    Inc = ((tLong)TICK_MS * (OSC_FREQ/1000)) /
          (tLong)OSC_PER_INST;

    /* 16-bit reload value */
    Reload_16 = (tWord) (65536UL - Inc);

    /* 8-bit reload values (High & Low) */
    Reload_08H = (tByte) (Reload_16 / 256);
    Reload_08L = (tByte) (Reload_16 % 256);

    /* Used for manually checking timing (in simulator) */
    /*P2 = Reload_08H; */
    /*P3 = Reload_08L; */

    TH2 = Reload_08H; /* Load T2 high byte */
    RCAP2H = Reload_08H; /* Load T2 reload capt. reg h byte */
    TL2 = Reload_08L; /* Load T2 low byte */
    RCAP2L = Reload_08L; /* Load T2 reload capt. reg l byte */

    /* Timer 2 interrupt is enabled, and ISR will be called
       whenever the timer overflows. */
    ET2 = 1;

    /* Start Timer 2 running */
    TR2 = 1;

    EA = 1;           /* Globally enable interrupts */
}

```

```
sEOS_ISR() interrupt INTERRUPT_Timer_2_Overflow
{
    /* Must manually reset the T2 flag */
    TF2 = 0;

    /*===== USER CODE - Begin =====*/
    /* Call dummy task here */
    X();

    /*===== USER CODE - End =====*/
}
```

```
void sEOS_Go_To_Sleep(void)
{
    PCON |= 0x01;      /* Enter idle mode (generic 8051 version) */
}

/*-----*
 *----- END OF FILE -----*
 */
```

Tasks, functions and scheduling

Trong hệ thống nhúng ta hay đề cập đến Task, thời gian thực hiện các Task và hệ multi tasking.

Task thường là một hàm chức năng thực hiện theo chu kỳ.

Trong hệ điều hành đơn giản Task là các hàm được thực hiện bởi gọi từ các ngắt thời gian.

Tasks, functions and scheduling

Đặt khoảng thời gian có thể được đặt một cách tự động.

Các giá trị tính toán có thể thực hiện trong Main();

Tasks, functions and scheduling

(Main.H):

```
/* Oscillator / resonator frequency (in Hz) e.g. (11059200UL) */
#define OSC_FREQ (12000000UL)

/* Number of oscillations per instruction (12, etc) */
...
#define OSC_PER_INST (12)
```

Tasks, functions and scheduling

```
/* Number of timer increments required (max 65536) */
Inc = ((tLong)TICK_MS * (OSC_FREQ/1000)) / (tLong)OSC_PER_INST;

/* 16-bit reload value */
Reload_16 = (tWord) (65536UL - Inc);

/* 8-bit reload values (High & Low) */
Reload_08H = (tByte)(Reload_16 / 256);
Reload_08L = (tByte)(Reload_16 % 256);

TH2      = Reload_08H;      /* Load T2 high byte */
RCAP2H   = Reload_08H;      /* Load T2 reload capt. reg h byte */
TL2      = Reload_08L;      /* Load T2 low byte */
RCAP2L   = Reload_08L;      /* Load T2 reload capt. reg l byte */
```

Tasks, functions and scheduling

Nếu thạch anh 12MHz thì chu kỳ thời gian có thể đạt từ 1ms đến 60 ms.

Nếu dùng thạch anh khác (ví dụ 11,0592MHz) độ chính xác của khoảng thời gian và khoảng có thể đạt được tới một mức nào đó.

Nếu yêu cầu chính xác cao-> kiểm tra lại bằng tay.

Tasks, functions and scheduling

Khi sử dụng hệ điều hành đơn giản ta có thể tiết kiệm nguồn bằng cách đưa uP về chế độ Idle.

Ví dụ với 89C51

C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C	10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz	20	mA
		Idle Mode, 12 MHz	5	mA
	Power Down Mode ⁽²⁾	$V_{CC} = 6V$	100	μA
		$V_{CC} = 3V$	40	μA

Notes: 1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:

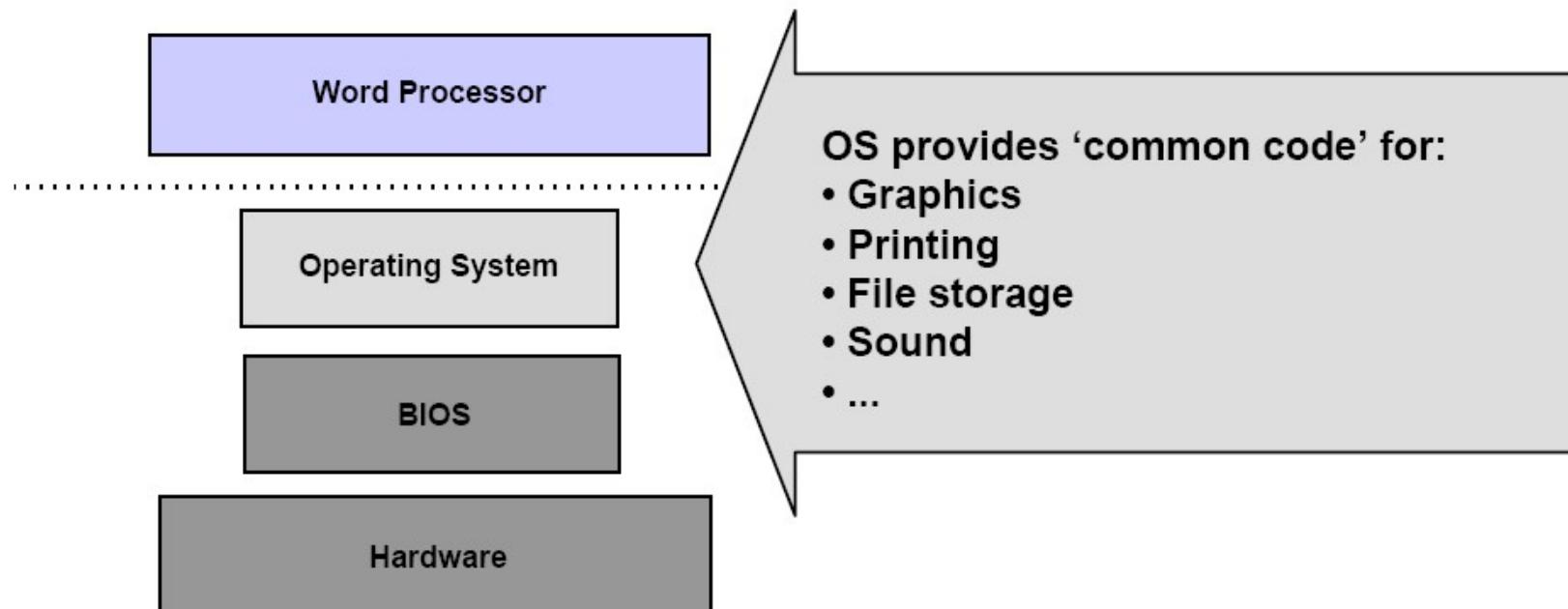
Hệ điều hành thời gian thực

Cấu trúc “Super loop”

```
void main(void)
{
    /* Prepare for Task X */
    X_Init();

    while(1) /* 'for ever' (Super Loop) */
    {
        X(); /* Perform the task */
    }
}
```

Hệ thống:



Task theo cho kỳ
Task thực hiện một lần.

```
void main(void)
{
    Timer_2_Init(); /* Set up Timer 2 */

    EA = 1;          /* Globally enable interrupts */

    while(1);        /* An empty Super Loop */
}
```

```
void Timer_2_Init(void)
{
    /* Timer 2 is configured as a 16-bit timer,
       which is automatically reloaded when it overflows
       With these setting, timer will overflow every 1 ms */
    T2CON    = 0x04;      /* Load T2 control register */
    T2MOD    = 0x00;      /* Load T2 mode register */

    TH2      = 0xFC;      /* Load T2 high byte */
    RCAP2H   = 0xFC;      /* Load T2 reload capt. reg. high byte */
    TL2      = 0x18;      /* Load T2 low byte */
    RCAP2L   = 0x18;      /* Load T2 reload capt. reg. low byte */

    /* Timer 2 interrupt is enabled, and ISR will be called
       whenever the timer overflows - see below. */
    ET2      = 1;

    /* Start Timer 2 running */
    TR2      = 1;
}
```

```
void X(void) interrupt INTERRUPT_Timer_2_Overflow
{
    /* This ISR is called every 1 ms */
    /* Place required code here... */
}
```

Single task

THE CO-OPERATIVE SCHEDULER

- A **co-operative scheduler** provides a **single-tasking system** architecture

Operation:

- Tasks are scheduled to run at specific times (either on a one-shot or regular basis)
- When a task is scheduled to run it is added to the waiting list
- When the CPU is free, the next waiting task (if any) is executed
- The task runs to completion, then returns control to the scheduler

Single task

Implementation:

- The scheduler is simple, and can be implemented in a small amount of code.
- The scheduler must allocate memory for only a single task at a time.
- The scheduler will generally be written entirely in a high-level language (such as 'C').
- The scheduler is not a separate application; it becomes part of the developer's code

Performance:

- Obtain rapid responses to external events requires care at the design stage.

Reliability and safety:

- Co-operate scheduling is simple, predictable, reliable and safe.

Hệ điều hành thời gian thực

Thế nào là hệ điều hành thời gian thực.

Thực hiện nó với 8051

Hệ điều hành thời gian thực

RTX-51 hệ điều hành thời gian thực cho 8051

RTS full và RTX tiny.

Hệ điều hành thời gian thực

Simple C Program using RTX51

```
#include <rtx51tny.h>                                /* Definitions for RTX51 Tiny */
int counter0;
int counter1;

job0 () task 0 {
    os_create_task (1);                                /* Mark task 1 as "ready"      */
    while (1) {                                         /* Endless loop               */
        counter0++;                                     /* Increment counter 0       */
    }
}

job1 () task 1 {
    while (1) {                                         /* Endless loop               */
        counter1++;                                     /* Increment counter 1       */
    }
}
```

Hệ điều hành thời gian thực

Program with os_wait Function

```
#include <rtx51tny.h>           /* Definitions for RTX51 Tiny */

int counter0;
int counter1;

job0 () task 0 {
    os_create_task (1);

    while (1) {
        counter0++;
        /* Increment counter 0 */ /* */
        os_wait (K_TMO, 3, 0);   /* Wait 3 timer ticks */ */
    }
}

job1 () task 1 {
    while (1) {
        counter1++;
        /* Increment counter 1 */ /* */
        os_wait (K_TMO, 5, 0);   /* Wait 5 timer ticks */ */
    }
}
```

Hệ điều hành thời gian thực

Program with Wait for Signal.

```
#include <rtx51tny.h>           /* Definitions for RTX51 Tiny */

int counter0;
int counter1;

job0 () task 0 {
    os_create_task (1);

    while (1) {
        if (++counter0 == 0) {      /* On counter 0 overflow */
            os_send_signal (1);   /* Send signal to task 1 */
        }
    }
}

job1 () task 1 {
    while (1) {
        os_wait (K_SIG, 0, 0);  /* Wait for signal; no timeout */
        counter1++;             /* Increment counter 1 */
    }
}
```

Hệ điều hành thời gian thực

Description	RTX-51 Full	RTX-51 Tiny
Number of tasks	256; max. 19 tasks active	16
RAM requirements	40 .. 46 bytes DATA 20 .. 200 bytes IDATA (user stack) min. 650 bytes XDATA	7 bytes DATA 3 * <task count> IDATA
Code requirements	6KB .. 8KB	900 bytes
Hardware requirements	timer 0 or timer 1	timer 0
System clock	1000 .. 40000 cycles	1000 .. 65535 cycles
Interrupt latency	< 50 cycles	< 20 cycles
Context switch time	70 .. 100 cycles (fast task) 180 .. 700 cycles (standard task) depends on stack load	100 .. 700 cycles depends on stack load
Mailbox system	8 mailboxes with 8 integer entries each	not available
Memory pool system	up to 16 memory pools	not available
Semaphores	8 * 1 bit	not available

Hệ điều hành thời gian thực

Function	Description	CPU Cycles
<code>isr_recv_message</code> †	Receive a message (call from interrupt).	71 (with message)
<code>isr_send_message</code> †	Send a message (call from interrupt).	53
<code>isr_send_signal</code>	Send a signal to a task (call from interrupt).	46
<code>os_attach_interrupt</code> †	Assign task to interrupt source.	119
<code>os_clear_signal</code>	Delete a previously sent signal.	57
<code>os_create_task</code>	Move a task to execution queue.	302
<code>os_create_pool</code> †	Define a memory pool.	644 (size 20 * 10 bytes)
<code>os_delete_task</code>	Remove a task from execution queue.	172
<code>os_detach_interrupt</code> †	Remove interrupt assignment.	96
<code>os_disable_isr</code> †	Disable 8051 hardware interrupts.	81
<code>os_enable_isr</code> †	Enable 8051 hardware interrupts.	80
<code>os_free_block</code> †	Return a block to a memory pool.	160
<code>os_get_block</code> †	Get a block from a memory pool.	148
<code>os_send_message</code> †	Send a message (call from task).	443 with task switch
<code>os_send_signal</code>	Send a signal to a task (call from tasks).	408 with task switch 316 with fast task switch 71 without task switch
<code>os_send_token</code> †	Set a semaphore (call from task).	343 with fast task switch 94 without task switch
<code>os_set_slice</code> †	Set the RTX-51 system clock time slice.	67
<code>os_wait</code>	Wait for an event.	68 for pending signal 160 for pending message

Hệ điều hành thời gian thực

Additional debug and support functions in RTX-51 Full include the following:

Function	Description
<code>oi_reset_int_mask</code>	Disables interrupt sources external to RTX-51.
<code>oi_set_int_mask</code>	Enables interrupt sources external to RTX-51.
<code>os_check_mailbox</code>	Returns information about the state of a specific mailbox.
<code>os_check_mailboxes</code>	Returns information about the state of all mailboxes in the system.
<code>os_check_pool</code>	Returns information about the blocks in a memory pool.
<code>os_check_semaphore</code>	Returns information about the state of a specific semaphore.
<code>os_check_semaphores</code>	Returns information about the state of all semaphores in the system.
<code>os_check_task</code>	Returns information about a specific task.

Hệ điều hành thời gian thực

Function	Description
<code>os_check_tasks</code>	Returns information about all tasks in the system.

Hệ điều hành thời gian thực

CAN Function	Description
<code>can_bind_obj</code>	Bind an object to a task; task is started when object is received.
<code>can_def_obj</code>	Define communication objects.
<code>can_get_status</code>	Get CAN controller status.
<code>can_hw_init</code>	Initialize CAN controller hardware.
<code>can_read</code>	Directly read an object's data.
<code>can_receive</code>	Receive all unbound objects.
<code>can_request</code>	Send a remote frame for the specified object.
<code>can_send</code>	Send an object over the CAN bus.
<code>can_start</code>	Start CAN communications.
<code>can_stop</code>	Stop CAN communications.
<code>can_task_create</code>	Create the CAN communication task.
<code>can_unbind_obj</code>	Disconnect the binding between a task and an object.
<code>can_wait</code>	Wait for reception of a bound object.
<code>can_write</code>	Write new data to an object without sending it.

Ví dụ:

Đèn giao thông.

Banking

Hệ Multi-state và điều khiển tuần tự

Hệ multi-state

Hệ Multi-state (theo thời gian): chuyển giữa các trạng thái chỉ phụ thuộc vào thời gian (đèn giao thông).

Hệ multi-state (theo thời gian/đầu vào): chuyển giữa các trạng thái phụ thuộc vào cả thời gian và đầu vào của hệ (Máy giặt, báo trộm)

Hệ multi-state (theo đầu vào): chuyển trạng thái chỉ phụ thuộc vào đầu vào

Hệ multi-state (theo thời gian)

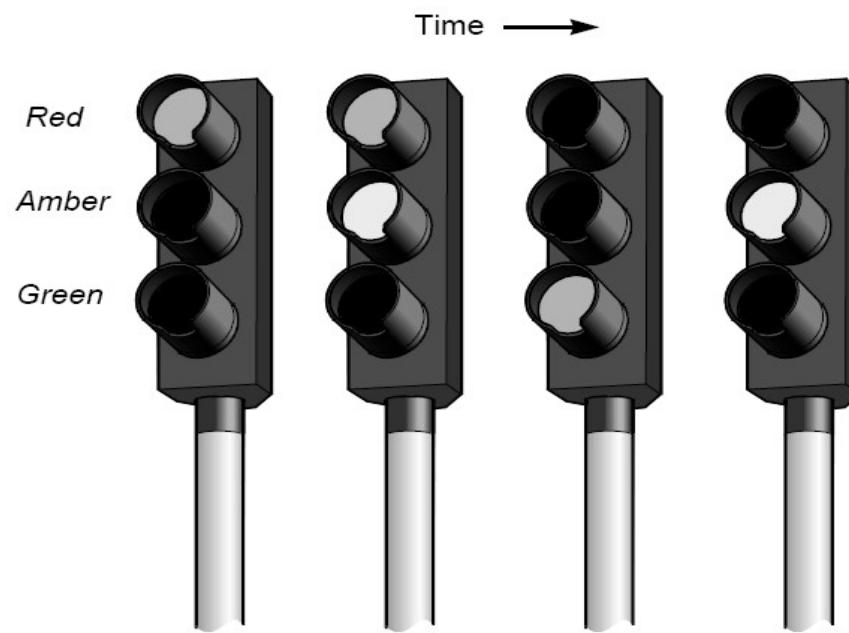
Hệ hoạt động ở hai hay nhiều trạng thái.

Mỗi một trạng thái thực hiện bởi gọi một hay nhiều hàm.

Chuyển trạng thái được điều khiển bởi thời gian.

Chuyển trạng thái có thể thực hiện bằng gọi hàm.

Hệ multi-state



Note:
European sequence!

H₂ multi-state

In this case, the various states are easily identified:

- **Red**
- **Red-Amber**
- **Green**
- **Amber**

Hệ multi-state

```
#include "Main.H"
#include "Port.H"

#include "T_lights.H"

/* ----- Private constants ----- */

/* Easy to change logic here */
#define ON 0
#define OFF 1

/* Times in each of the (four) possible light states
   (Times are in seconds) */
#define RED_DURATION 20
#define RED_AND_AMBER_DURATION 5
#define GREEN_DURATION 30
#define AMBER_DURATION 5

/* ----- Private variables ----- */

/* The state of the system */
static eLight_State Light_state_G;

/* The time in that state */
static tLong Time_in_state;

/* Used by sEOS */
static tByte Call_count_G = 0;
```

Hệ multi-state

```
#include "Main.H"
#include "Port.H"
#include "Simple_EOS.H"

#include "T_Lights.H"

/* ----- */

void main(void)
{
    /* Prepare to run traffic sequence */
    TRAFFIC_LIGHTS_Init(RED);

    /* Set up simple EOS (50 ms ticks) */
    sEOS_Init_Timer2(50);

    while(1) /* Super Loop */
    {
        /* Enter idle mode to save power */
        sEOS_Go_To_Sleep();
    }
}
```

Hệ multi-state

```
/*-----*
   T_Lights.H (v1.00)
-----*

- See T_Lights.C for details.

-*-----*/
#ifndef _T_LIGHTS_H
#define _T_LIGHTS_H

/* ----- Public data type declarations ----- */
/* Possible system states */
typedef enum {RED, RED_AND_AMBER, GREEN, AMBER} eLight_State;

/* ----- Public function prototypes ----- */

void TRAFFIC_LIGHTS_Init(const eLight_State);
void TRAFFIC_LIGHTS_Update(void);

#endif
```

Hệ multi-state

```
void TRAFFIC_LIGHTS_Init(const eLight_State START_STATE)
{
    Light_state_G = START_STATE; /* Decide on initial state */
}
```

Hệ multi-state

```
void TRAFFIC_LIGHTS_Update(void)
{
    switch (Light_state_G)
    {
        case RED:
            {
                Red_light = ON;
                Amber_light = OFF;
                Green_light = OFF;

                if (++Time_in_state == RED_DURATION)
                {
                    Light_state_G = RED_AND_AMBER;
                    Time_in_state = 0;
                }
            }

            break;
    }
}
```

Hệ multi-state

```
case RED_AND_AMBER:  
{  
    Red_light = ON;  
    Amber_light = ON;  
    Green_light = OFF;  
  
    if (++Time_in_state == RED_AND_AMBER_DURATION)  
    {  
        Light_state_G = GREEN;  
        Time_in_state = 0;  
    }  
  
    break;  
}
```

Hệ multi-state

```
case GREEN:  
{  
    Red_light = OFF;  
    Amber_light = OFF;  
    Green_light = ON;  
  
    if (++Time_in_state == GREEN_DURATION)  
    {  
        Light_state_G = AMBER;  
        Time_in_state = 0;  
    }  
  
    break;  
}
```

Hệ multi-state

```
case AMBER:  
    {  
        Red_light = OFF;  
        Amber_light = ON;  
        Green_light = OFF;  
  
        if (++Time_in_state == AMBER_DURATION)  
        {  
            Light_state_G = RED;  
            Time_in_state = 0;  
        }  
  
        break;  
    }  
}
```

Hệ multi-state(Timed/Input)

Hệ hoạt động với hay hay nhiều trạng thái.

Mỗi một trạng thái thực hiện một hay nhiều chức năng.

Chuyển trạng thái bằng đầu vào hay thời gian hoặc tổ hợp cả hai thứ.

Chuyển trạng thái thực hiện nhờ hàm.

Hệ multi-state

Bơm nước trong 10s nếu không thấy nước thì tắt bơm và nước cạn.

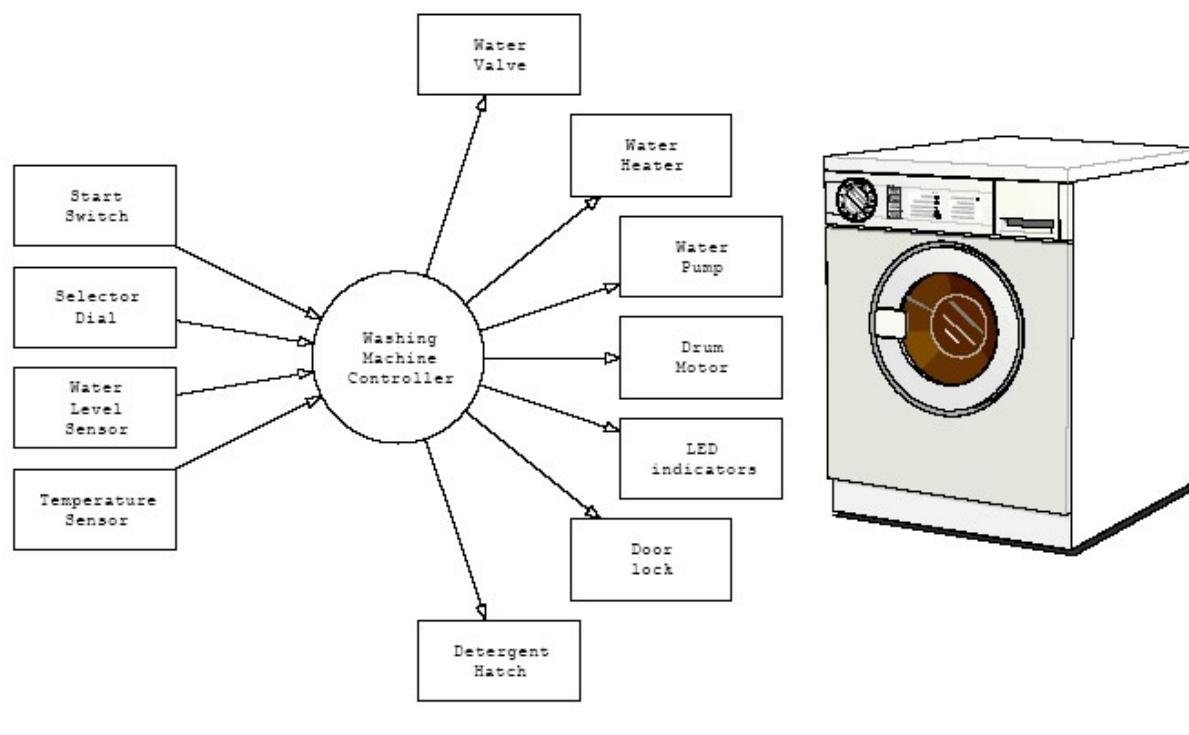
Nếu có nước thì bơm trong 45s hay khi bể đầy

Hệ multi-state

Nếu cửa mở nhảy password trong 30s nếu không báo động.

Hệ multi-state

Ví dụ: Máy giặt



Hệ multi-state

Người dùng lựa chọn chế độ giặt

Ấn phím start.

Cửa đã đóng.

Mở van để nước chảy vào thùng giặt.

Mở->đóng van của bột giặt.

Khi nước đầy đóng van nước.

Nếu có giặt nóng bật đun nước, đạt nhiệt độ tắt đi

Hệ multi-state

Bật motor quay lồng giặt. Kết thúc chu kỳ motor dừng.

Bật bơm xả nước, khi hết nước thì dừng.

Hệ multi-state

```
#include "Main.H"
#include "Port.H"

#include "Washer.H"

/* ----- Private data type declarations ----- */

/* Possible system states */
typedef enum {INIT, START, FILL_DRUM, HEAT_WATER,
              WASH_01, WASH_02, ERROR} eSystem_state;

/* ----- Private function prototypes ----- */

tByte WASHER_Read_Selector_Dial(void);
bit   WASHER_Read_Start_Switch(void);
bit   WASHER_Read_Water_Level(void);
bit   WASHER_Read_Water_Temperature(void);

void WASHER_Control_Detergent_Hatch(bit);
void WASHER_Control_Door_Lock(bit);
void WASHER_Control_Motor(bit);
void WASHER_Control_Pump(bit);
void WASHER_Control_Water_Heater(bit);
void WASHER_Control_Water_Valve(bit);
```

Hệ multi-state

```
/* ----- Private constants ----- */  
  
#define OFF 0  
#define ON 1  
  
#define MAX_FILL_DURATION (tLong) 1000  
#define MAX_WATER_HEAT_DURATION (tLong) 1000  
  
#define WASH_01_DURATION 30000
```

Hệ multi-state

```
/* ----- Private variables ----- */

static eSystem_state System_state_G;

static tWord Time_in_state_G;

static tByte Program_G;

/* Ten different programs are supported
   Each one may or may not use detergent */
static tByte Detergent_G[10] = {1,1,1,0,0,1,0,1,1,0};

/* Each one may or may not use hot water */
static tByte Hot_Water_G[10] = {1,1,1,0,0,1,0,1,1,0};

/* ----- */
void WASHER_Init(void)
{
    System_state_G = INIT;
}
```

Hệ multi-state

```
/* ----- */
void WASHER_Update(void)
{
    /* Call once per second */
    switch (System_state_G)
    {
        case INIT:
        {
            /* For demo purposes only */
            Debug_port = (tByte) System_state_G;

            /* Set up initial state */
            /* Motor is off */
            WASHER_Control_Motor(OFF);

            /* Pump is off */
            WASHER_Control_Pump(OFF);

            /* Heater is off */
            WASHER_Control_Water_Heater(OFF);
        }
    }
}
```

Hệ multi-state

```
/* Valve is closed */
WASHER_Control_Water_Valve(OFF);

/* Wait (indefinitely) until START is pressed */
if (WASHER_Read_Start_Switch() != 1)
{
    return;
}

/* Start switch pressed
   -> read the selector dial */
Program_G = WASHER_Read_Selector_Dial();

/* Change state */
System_state_G = START;
break;
}
```

Hệ multi-state

```
case START:  
{  
    /* For demo purposes only */  
    Debug_port = (tByte) System_state_G;  
  
    /* Lock the door */  
    WASHER_Control_Door_Lock(ON);  
  
    /* Start filling the drum */  
    WASHER_Control_Water_Valve(ON);  
  
    /* Release the detergent (if any) */  
    if (Detergent_G[Program_G] == 1)  
    {  
        WASHER_Control_Detergent_Hatch(ON);  
    }  
  
    /* Ready to go to next state */  
    System_state_G = FILL_DRUM;  
    Time_in_state_G = 0;  
  
    break;  
}
```

Hệ multi-state

```
case FILL_DRUM:  
{  
    /* For demo purposes only */  
    Debug_port = (tByte) System_state_G;  
  
    /* Remain in this state until drum is full  
     * NOTE: Timeout facility included here */  
    if (++Time_in_state_G >= MAX_FILL_DURATION)  
    {  
        /* Should have filled the drum by now... */  
        System_state_G = ERROR;  
    }  
  
    /* Check the water level */  
    if (WASHER_Read_Water_Level() == 1)  
    {  
        /* Drum is full */
```

Hệ multi-state

```
/* Does the program require hot water? */
if (Hot_Water_G[Program_G] == 1)
{
    WASHER_Control_Water_Heater(ON);

    /* Ready to go to next state */
    System_state_G = HEAT_WATER;
    Time_in_state_G = 0;
}
else
{
    /* Using cold water only */
    /* Ready to go to next state */
    System_state_G = WASH_01;
    Time_in_state_G = 0;
}
break;
}
```

Hệ multi-state

```
case HEAT_WATER:  
{  
    /* For demo purposes only */  
    Debug_port = (tByte) System_state_G;  
  
    /* Remain in this state until water is hot  
     * NOTE: Timeout facility included here */  
    if (++Time_in_state_G >= MAX_WATER_HEAT_DURATION)  
    {  
        /* Should have warmed the water by now... */  
        System_state_G = ERROR;  
    }  
  
    /* Check the water temperature */  
    if (WASHER_Read_Water_Temperature() == 1)  
    {  
        /* Water is at required temperature */  
        /* Ready to go to next state */  
        System_state_G = WASH_01;  
        Time_in_state_G = 0;  
    }  
  
    break;  
}
```

Hệ multi-state

```
case WASH_01:  
{  
    /* For demo purposes only */  
    Debug_port = (tByte) System_state_G;  
  
    /* All wash program involve WASH_01  
       Drum is slowly rotated to ensure clothes are fully wet */  
    WASHER_Control_Motor(ON);  
  
    if (++Time_in_state_G >= WASH_01_DURATION)  
    {  
        System_state_G = WASH_02;  
        Time_in_state_G = 0;  
    }  
  
    break;  
}
```

Hệ multi-state

```
case WASH_02:  
{  
    /* For demo purposes only */  
    Debug_port = (tByte) System_state_G;  
  
    break;  
}  
  
case ERROR:  
{  
    /* For demo purposes only */  
    Debug_port = (tByte) System_state_G;  
  
    break;  
}  
}
```

Hệ multi-state

```
/* ----- */
tByte WASHER_Read_Selector_Dial(void)
{
    /* User code here... */

    return 0;
}

/* ----- */
bit WASHER_Read_Start_Switch(void)
{
    /* Simplified for demo ... */

    if (start_pin == 0)
    {
        /* start switch pressed */
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Hệ multi-state

```
/* ----- */
bit WASHER_Read_Water_Level(void)
{
    /* User code here... */

    return 1;
}

/* ----- */
bit WASHER_Read_Water_Temperature(void)
{
    /* User code here... */

    return 1;
}

/* ----- */
void WASHER_Control_Detergent_Hatch(bit State)
{
    bit Tmp = State;
    /* User code here... */
}
```

Watchdog

Watchdog

Watchdog cần có hai đặc điểm sau:

- Chương trình thực hiện theo chu kỳ, nếu quá chu kỳ đó thì gây ra reset.
- Khi khởi động uC cần phải định rõ nguyên nhân gây ra reset (reset thường hay watchdog).

Watchdog

Watchdog có ba vấn đề chính :

- Lựa chọn phần cứng.
- Watchdog với mạch reset.
- Quá trình phục hồi.

Watchdog

Lựa chọn phần cứng: ta có thể chọn trên chíp hay mạch ngoài. Chọn trên chip có một số ưu điểm sau:

- Giảm độ phức tạp của phần cứng -> hệ ổn định.
- Giảm giá thành.
- Giảm kích thước.
- Chỉ có nó mới cho phép phân biệt nguyên nhân gây reset.

Watchdog

Giám sát lỗi chu kỳ thời gian: trong các ứng dụng theo chu kỳ, watchdog timer cho phép phát hiện tràn thời gian như sau:

- Đặt thời gian tràn của watchdog ở chu kỳ lớn nhất
- Các chu kỳ ứng dụng sẽ xoá watchdog trước khi tràn.
- Cho watchdog làm việc.

Watchdog

Nếu thời gian tràn có thể cho nó phát ra tín hiệu reset để khắc phục lỗi.

Reset có thể do một số nguyên nhân....

Watchdog

Thao tác phục hồi thực hiện khi hệ thống bị reset

Nếu reset thường thì hệ thống sẽ chạy với cấu hình chuẩn.

Nếu reset bởi watchdog có 3 cách sau:

- Hệ thực hiện như reset thông thường.
- Giữ hệ thống ở trạng thái reset, có thể hiểu như là “fail-silent recovery”.
- Chạy hệ với thuật toán khác, “limp home recovery”.

Watchdog

Lập thời gian cho Watchdog

- Chu kỳ cố định.
- Chu kỳ thay đổi.

Watchdog

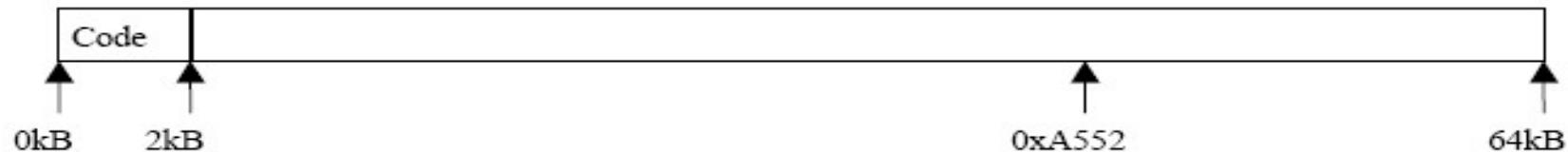
Program-Flow Watchdog cải thiện ổn định hệ thống

Có hai cách thực hiện:

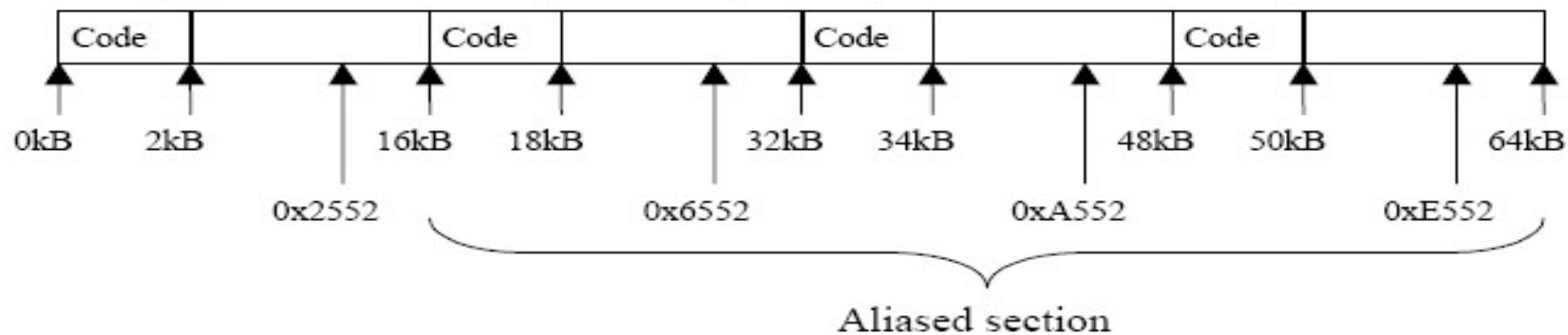
- Điền đầy vị trí không sử dụng bằng lệnh NOP.
- Đặt “PC Error Handler” vào phần cuối của bộ nhớ

Watchdog

64kB physical code memory – no memory aliasing



16kB physical code memory – memory overlap 4 times due to aliasing



Watchdog

Tăng tốc độ đáp ứng:

- Sử dụng lệnh long jump.
- Đặt ct xử lý lỗi tại địa chỉ đó.

Hệ đa vi xử lý

Multi uC

Tại sao sử dụng nhiều uC:

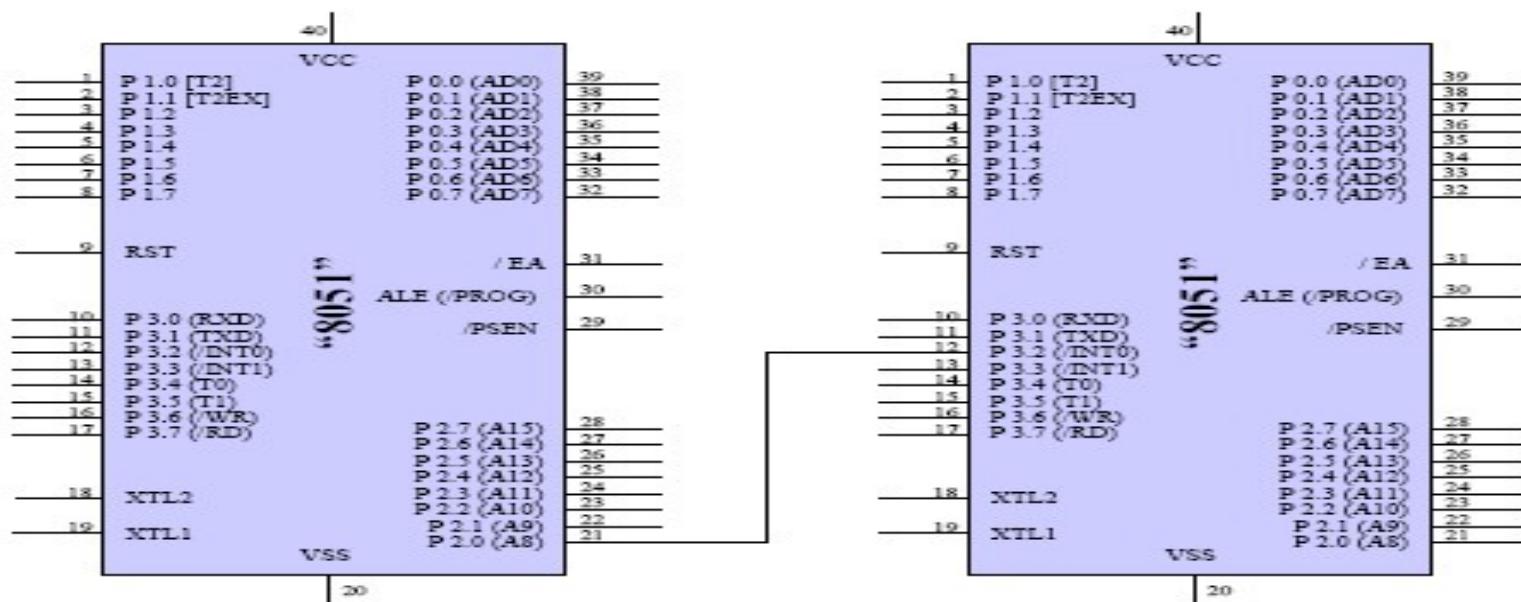
- Thêm hiệu năng cho CPU, các chức năng phần cứng.
- Thiết kế theo module.

Multi uC

Thêm hiệu năng cho CPU và các chức năng phần cứng:
uC có thông số sau:

- 60 chân vào ra.
- 6 bộ thời gian.
- 2 UART.
- 128 kROM
- 512 RAM
- Giá VNĐ

Multi uC



Multi uC

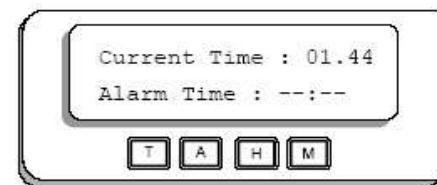
62 chân tín hiệu, 4 bộ timer, 2 UART....

Thông tin = 1 dây đảm bảo các Task đồng bộ.

Multi task có thể thực hiện

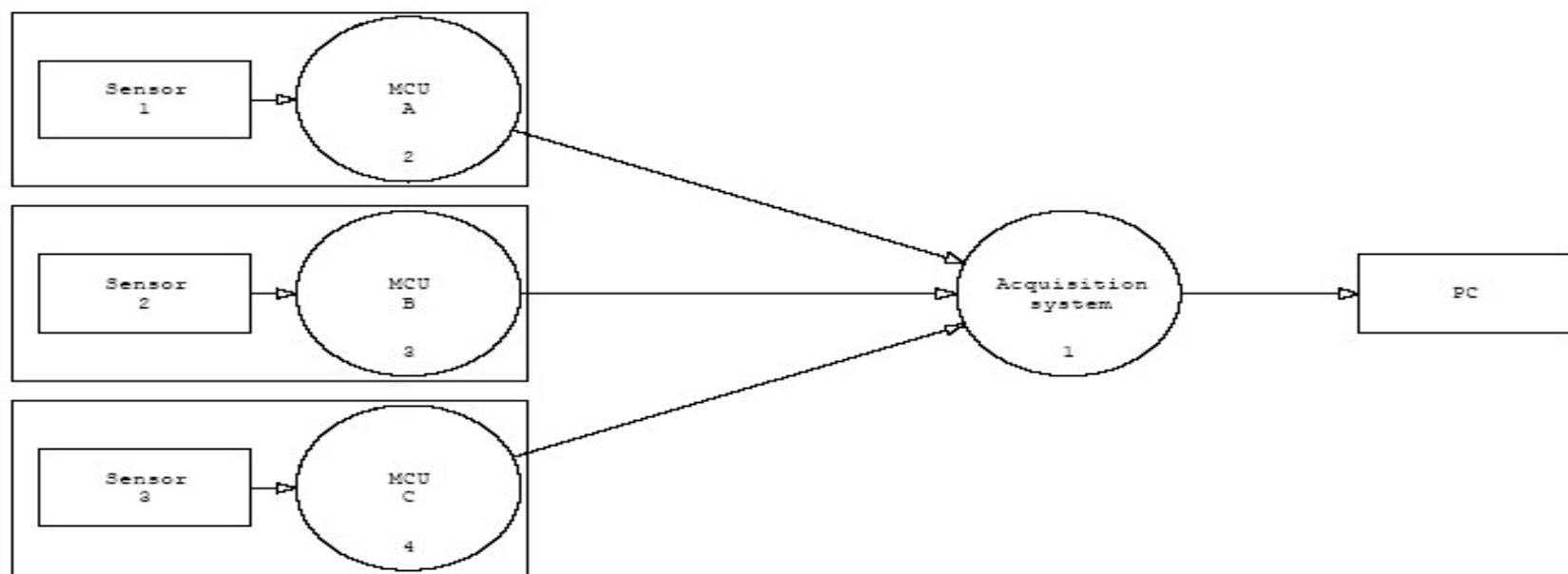
Multi uC

Suppose we want to build a range of clocks...



We can split the design into ‘display’ and ‘time-keeping’ modules.

Multi uC



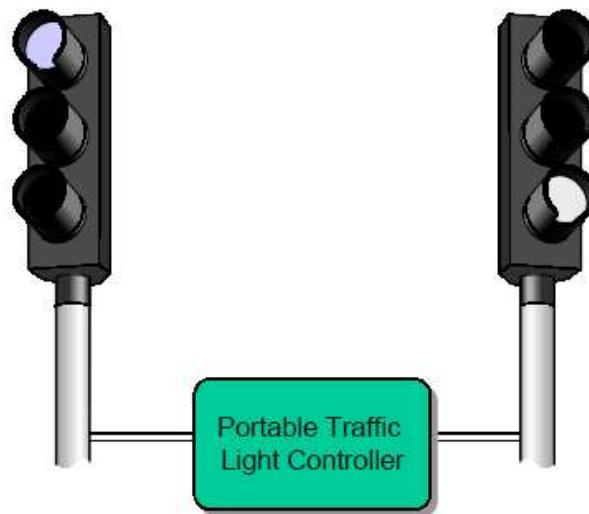
Multi uC

Để thực hiện nhiều uC thì phải giải được các bài toán:

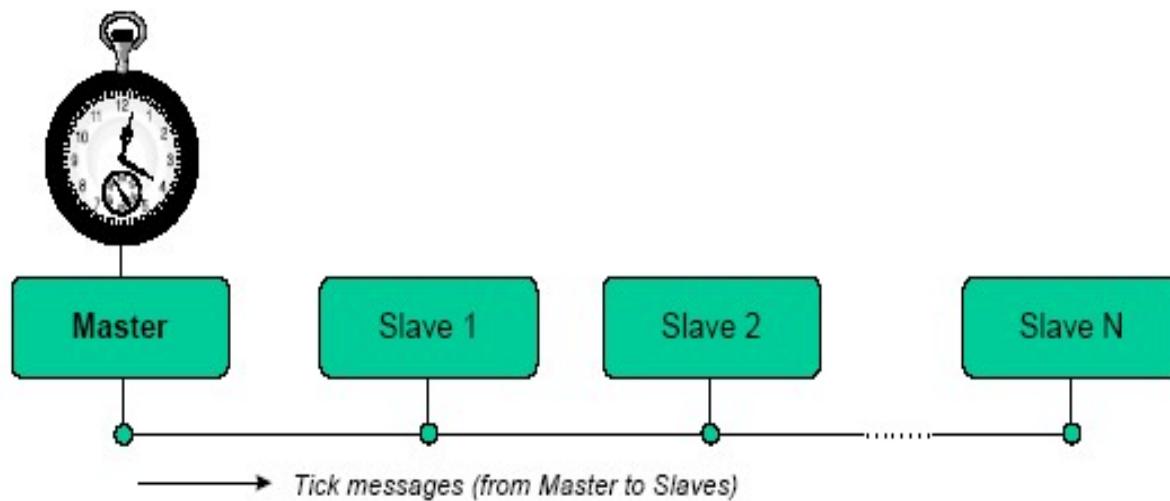
- Đồng bộ đồng hồ.
- Thực hiện truyền số liệu giữa các nút.
- Làm thế nào để một nút có thể đọc lỗi từ nút khác.

Multi uC

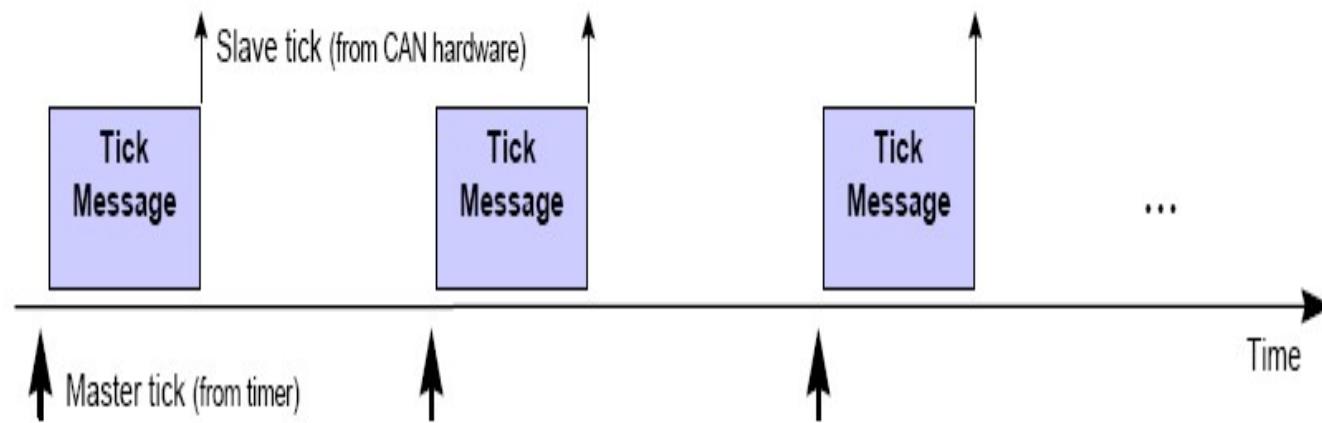
Đồng bộ đồng hồ:



Multi uC

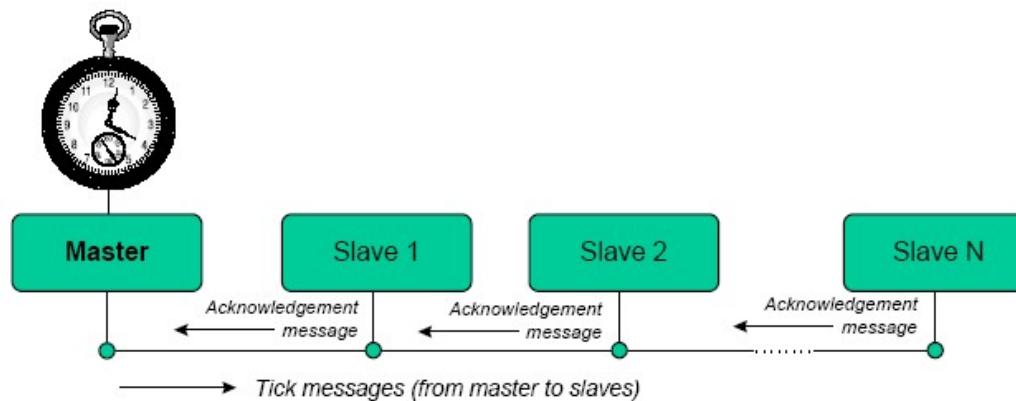


Multi uC

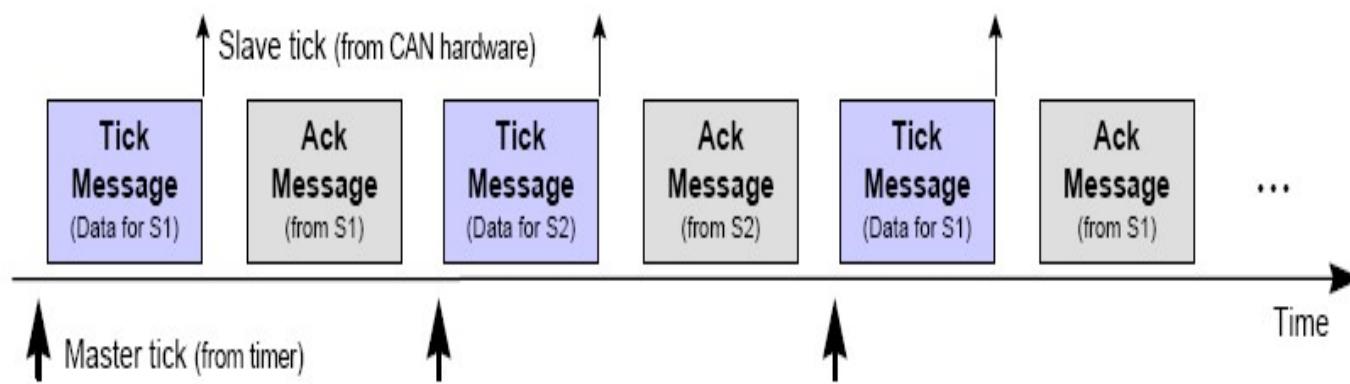


Multi uC

Truyền số liệu: các Task cần truyền số liệu cho nhau

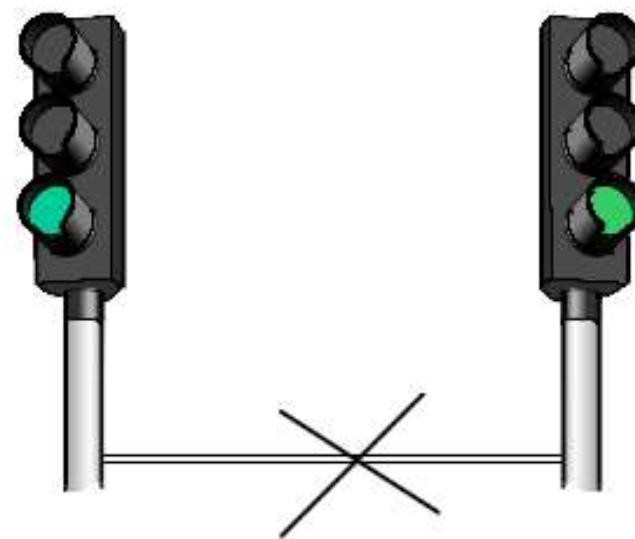


Multi uC



Multi uC

Phát hiện mạng hay nút lối:



Multi uC

Trong Slave:

- Slave cần nhận stick trong khoảng thời gian chính xác cho trước.
- Đo thời gian này, nếu nó lớn hơn thời gian đặt trước thì đã có lỗi xảy ra.
- Đặt watchdog.

Multi uC

Trong master:

- Master yêu cầu các slave trả lời trong thời gian định trước.

Multi uC

Khi lỗi trong slave: nó duy trì trạng thái an toàn cho đến khi nhận được tín hiệu start từ master.

Lỗi master:

- Chuyển sang chế độ an toàn.
- Khởi động lại mạng.
- Khởi động lại các slave

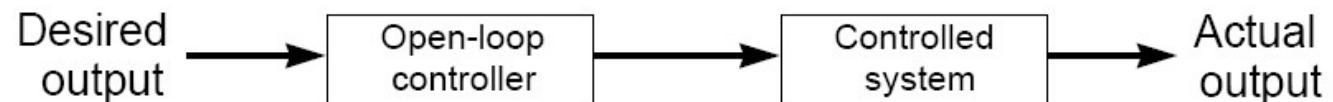
Multi uC

Nhiều uC cho ứng dụng có độ an toàn cao.

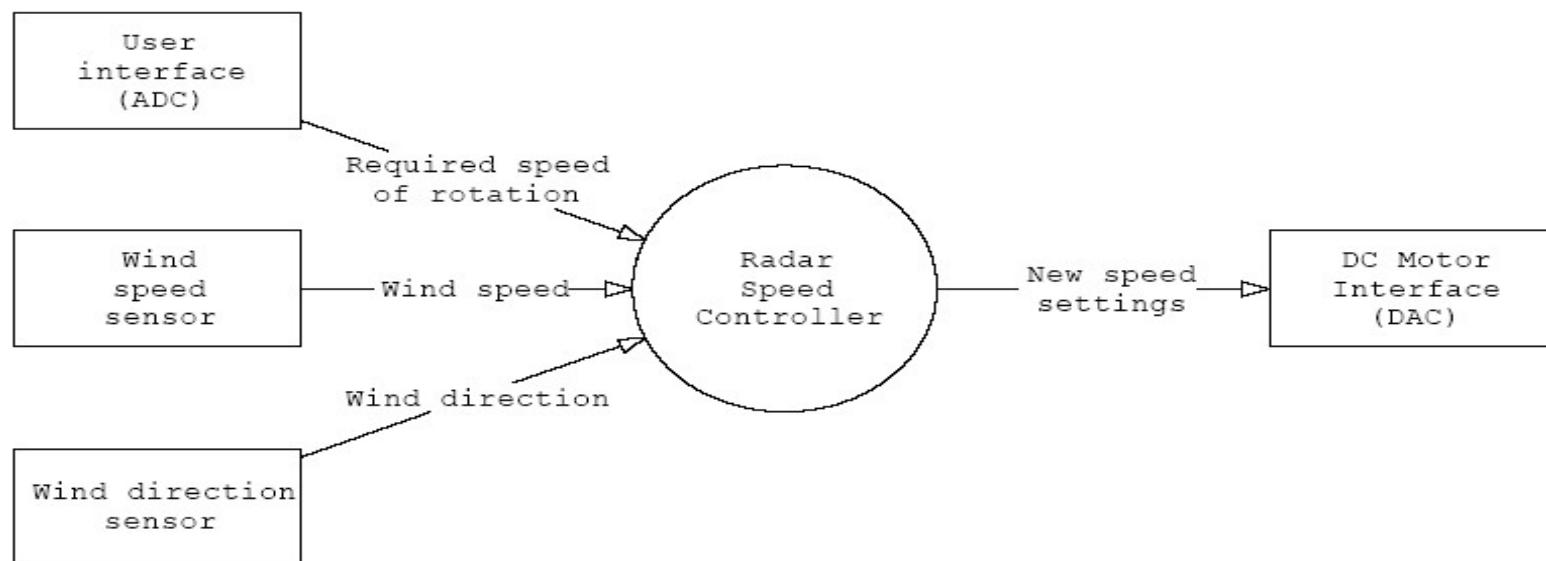
Điều khiển PID

Điều khiển PID

Điều khiển vòng hở:

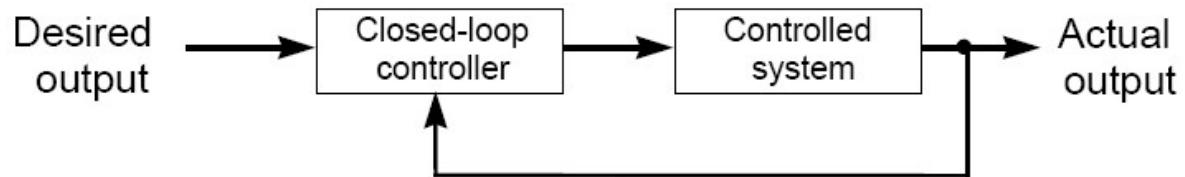


Điều khiển PID

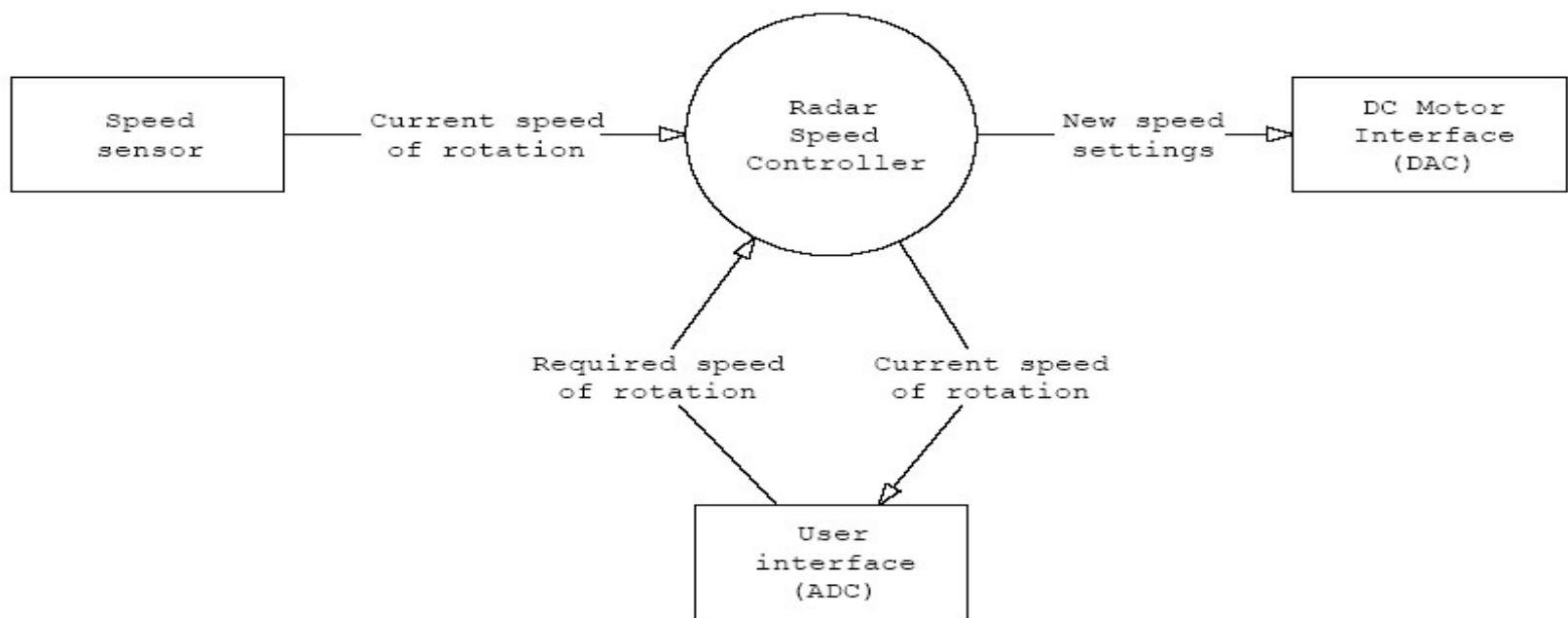


Điều khiển PID

Điều khiển vòng kín:



Điều khiển PID



Điều khiển PID

Thuật toán xử dụng cho điều khiển vòng kín là thuật toán PID.

PID có một số thông số sau:

- $U(k)$ setpoint.
- $E(k)$ sai lệch tại thời điểm lấy mẫu k
- K hệ số tỉ lệ
- $1/T_i$ Hệ số tích phân.
- T_d hệ số vi phân.

Điều khiển PID

```
/* Proportional term      */
Change_in_controller_output = PID_KP * Error;

/* Integral term */
Sum += Error;
Change_in_controller_output += PID_KI * Sum;

/* Differential term */
Change_in_controller_output += (PID_KD * SAMPLE_RATE * (Error -
Old_error));
```

Điều khiển PID

Another version

```
float PID_Control(float Error, float Control_old)
{
    /* Proportional term */
    float Control_new = Control_old + (PID_KP * Error);

    /* Integral term */
    Sum_G += Error;
    Control_new += PID_KI * Sum_G;

    /* Differential term */
    Control_new += (PID_KD * SAMPLE_RATE * (Error - Old_error_G));
}
```

Điều khiển PID

```
/* Control_new cannot exceed PID_MAX or fall below PID_MIN */
if (Control_new > PID_MAX)
{
    Control_new = PID_MAX;
}
else
{
    if (Control_new < PID_MIN)
    {
        Control_new = PID_MIN;
    }
}

/* Store error value */
Old_error_G = Error;

return Control_new;
}
```

Điều khiển PID

Dealing with ‘windup’

```
float PID_Control(float Error, float Control_old)
{
    /* Proportional term      */
    float Control_new = Control_old + (PID_KP * Error);

    /* Integral term */
    Sum_G += Error;
    Control_new += PID_KI * Sum_G;

    /* Differential term */
    Control_new += (PID_KD * SAMPLE_RATE * (Error - Old_error_G));

    /* Optional windup protection - see text */
    if (PID_WINDUP_PROTECTION)
    {
        if ((Control_new > PID_MAX) || (Control_new < PID_MIN))
        {
            Sum_G -= Error; /* Don't increase Sum... */
        }
    }
}
```

Điều khiển PID

```
/* Control_new cannot exceed PID_MAX or fall below PID_MIN */
if (Control_new > PID_MAX)
{
    Control_new = PID_MAX;
}
else
{
    if (Control_new < PID_MIN)
    {
        Control_new = PID_MIN;
    }
}

/* Store error value */
Old_error_G = Error;

return Control_new;
}
```

Điều khiển PID

Lựa chọn thông số cho PID:

- Đặt Ki và Kd về 0.
- Tăng Kp đến khi hệ bắt đầu dao động.
- Giảm Kp bằng 1 nửa giá trị trên.
- Thủ nghiệm với giá trị Kd nhỏ để giảm thay đổi đầu ra.
- Thay đổi Ki giá trị nhỏ để giảm sai lệch tĩnh.

Điều khiển PID

Tần số lấy mẫu:

- Tần số lấy mẫu phụ thuộc vào tốc độ thay đổi của tín hiệu.

Điều khiển PID

Phần cứng để thực hiện PID:

- Thực hiện một vài phép tính số nguyên và số thực.
- Thời gian thực hiện thay đổi thông thường nó thực hiện 4 phép nhân, 3 phép cộng và 2 phép trừ
- Với số thực và với Keil C nó thực hiện 2000 lệnh
- Nó có thể thực hiện cỡ ms với 8051 chạy ở Fosc=24MHz

Điều khiển PID

- 1 ms đáp ứng phần lớn nhu cầu điều khiển (đòi hỏi vài trăm mS).
- Nếu yêu cầu cao hơn thì có Infineon 517 and 509 có phần cứng cho thực hiện phép toán nhanh hơn.

Điều khiển PID

PID phù hợp với hệ nhiều bộ ĐK 1vào-1 ra

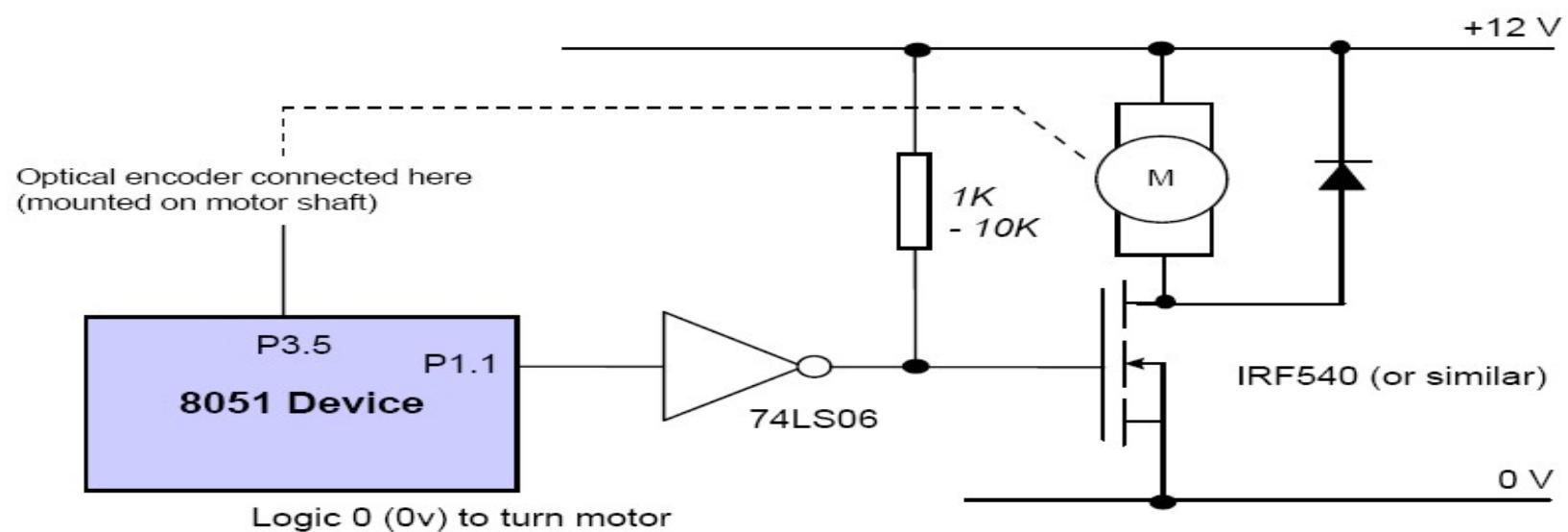
Không cho phép ĐK nhiều đầu vào/ra

Dùng thông dụng, hiệu quả

dễ thực hiện.

Thông số phải điều chỉnh và có thể rất mất thời gian.

Điều khiển PID



Điều khiển PID

```
void main(void)
{
    SCH_Init_T1(); /* Set up the scheduler */
    PID_MOTOR_Init();

    /* Set baud rate to 9600, using internal baud rate generator */
    /* Generic 8051 version */
    PC_LINK_Init_Internal(9600);

    /* Add a 'pulse count poll' task */
    /* TIMING IS IN TICKS (1ms interval) */
    /* Every 5 milliseconds (200 times per second) */
    SCH_Add_Task(PID_MOTOR_Poll_Speed_Pulse, 1, 1);

    SCH_Add_Task(PID_MOTOR_Control_Motor, 300, 1000);

    /* Sending data to serial port */
    SCH_Add_Task(PC_LINK_Update, 3, 1);
```

Điều khiển PID

```
/* All tasks added: start running the scheduler */
SCH_Start();

while(1)
{
    SCH_Dispatch_Tasks();
}

...
#define PULSE_HIGH (0)
#define PULSE_LOW (1)

#define PID_PROPORTIONAL (5)
#define PID_INTEGRAL      (50)
#define PID_DIFFERENTIAL (50)
```

Điều khiển PID

```
void PID_MOTOR_Control_Motor(void)
{
    int Error, Control_new;

    Speed_measured_G = PID_MOTOR_Read_Current_Speed();
    Speed_required_G = PID_MOTOR_Get_Required_Speed();

    /* Difference between required and actual speed (0-255) */
    Error = Speed_required_G - Speed_measured_G;

    /* Proportional term */
    Control_new = Controller_output_G + (Error / PID_PROPORIONAL);

    /* Integral term [SET TO 0 IF NOT REQUIRED] */
    if (PID_INTEGRAL)
    {
        Sum_G += Error;
        Control_new += (Sum_G / (1 + PID_INTEGRAL));
    }
}
```

Điều khiển PID

```
/* Differential term [SET TO 0 IF NOT REQUIRED] */
if (PID_DIFFERENTIAL)
{
    Control_new += (Error - Old_error_G) / (1 + PID_DIFFERENTIAL);

    /* Store error value */
    Old_error_G = Error;
}

/* Adjust to 8-bit range */
if (Control_new > 255)
{
    Control_new = 255;
    Sum_G -= Error; /* Windup protection */
}

if (Control_new < 0)
{
    Control_new = 0;
    Sum_G -= Error; /* Windup protection */
}
```

Điều khiển PID

```
/* Convert to required 8-bit format */
Controller_output_G = (tByte) Control_new;

/* Update the PWM setting */
PID_MOTOR_Set_New_PWM_Output(Controller_output_G);
...
}
```