1. 介紹

閱讀完本章後，讀者將：
1.了解我們如何設計機電系統
2.知道此類系統的設計階段是什麼
3.對如何處理設計的每個階段都有清晰的想法機電一體化系統

我們在上一屆電子學中看到的進步和小型化數十年來，工程師們提出了新產品和新工程學科。在十八世紀初期，我們已經看到了新產品的推出結合了機械零件和電子零件。另一個因素機電一體化應用的蓬勃發展是價格的不斷下降電子零件以及設計非常小的系統的挑戰。今天，高性能的實例微處理器變得非常便宜，鼓勵在計算機控制系統中使用它們。

微處理器是一種集成電路，其中包含整個中央過程，單芯片上的計算機單元。微處理器是我們的主要組成部分如今的計算機。它執行所有必要的計算並處理數據。的微處理器具有以下組件：

- 控制單元
- 算術和邏輯單元

•輸入/輸出（I / O）數據總線

•地址總線

•內部寄存器

•時鐘

為了構造計算機，將其他外圍設備和組件添加到了計算機中。

主要部分是微處理器。屏幕，硬盤，軟盤，內存等我們可以在計算機中使用的此類外圍設備的示例。對於 com-在推桿控制的系統中，我們需要適當的卡，稱為數據採集卡。

這些設備帶有模數（ADC）和數模（DAC）連接，轉換器和其他必要組件的實時控制應用程序。對於一些機電系統，計算機和數據採集卡的使用不被允許更重要的是，我們通常使用圍繞微控制器構建的電子電路可以認為是帶有自己的外圍設備的小型微處理器。微控制器就是集成電路，就像微處理器一樣，包括：

•相對簡單的中央處理器（CPU）

•記憶

•晶體振盪器

•計時器

•看門狗，

•串行和模擬 I / O

•脈寬調製（PWM）模塊

微控制器專為小型應用而設計，而微處理器用於高性能應用程序和個人計算機。英特爾 mi-在筆記本電腦中運行的 croprocessor 是這些微處理器的示例，Microchip1 的 PIC 是微控制器的示例。這些機器被使用在我們日常生活中使用的幾乎所有產品中。作為使用的例子微控制器，引用：

• 汽車

•飛機

•手機

• 數碼相機

如今，大多數係統都是計算機控制的，些機電系統是在微控制器中實現的。該學科處理此類系統的是機電一體化，我們將其定義為協同組合-機械工程，電子工程和軟件工

程。

這個跨學科工程領域的目的是建立和控制組合通過提供硬件和軟件解決方案來實現 plex 系統。工程師在工作在該領域中，必須掌握電子，控制和編程方面的概念。前在航空航天等工業領域中可以找到這種系統的樣品

汽車工業。
機電一體化系統的設計是一項任務，需要來自不同領域的工程師
機械工程，電氣工程，控制工程等學科，

計算機工程等方面的知識。
是最好的機電系統。這些機電一體化系統大多數是由的：
•機械零件，包括執行器和傳感器
•圍繞微控制器或一組微控制器構建的電子電路
微控制器
•代表系統智能的實時實現

作為機電系統的例子，讓我們考慮作為機電系統的例子，讓我們考慮一個實驗室設置實時執行控制算法。此設置必須具有所有允許學習實時控制的功能。更具體地說，是針對實時執行控制算法。此設置必須具有所有允許學習實時控制的功能。進一步來說，

•機械部分必須允許用戶檢查控件的輸出 算法

•電子電路必須簡單易用，以防用戶複製

•實施必須易於執行且有據可查。

在本章的其餘部分，我們將簡要介紹設計的每個階段。整個機電系統。

1.1 機械零件設計

機械部分是機電系統中的基本部分。在階段設計此零件，我們將構思和製造構成零件的零件機電系統。我們還將選擇我們將使用的執行器和傳感器該機電一體化系統。機械零件的設計或零件的選擇

執行器和傳感器是通過遵守一些預先設計的規則來完成的。

在本卷的下一章中發送。記住一點也很重要機電系統的回收一旦過時，就必須尊重我們的環境是我們在設計階段必須考慮的重要問題。用於維護或其他任何目的的系統的組裝和拆卸在設計階段也應考慮目的。對於機械零件的設計，機械設計的步驟例如問題的定義，使用頭腦風暴法或其他等效方法研究解決方案

方法，實用性研究，原型製作等。執行器的選擇

並且傳感器也應遵循其中的準則和規範

採用。例如，如果機電一體化系統旨在在礦山中運行，避免使用電動執行器，因為它們可能引起火災，而對於食品行業

液壓執行器也不包括在內。

對於我們正在考慮作為示例的實時實施的設置，在這種情況下，機械部分只是一個小刻度盤（以度為單位）牢固地連接到執行器的軸上。機械零件是由

鋁。執行器是直流電動機，配有齒輪箱和編碼器。變速箱的作用是降低機械零件的速度，也要施加高扭矩。編碼器

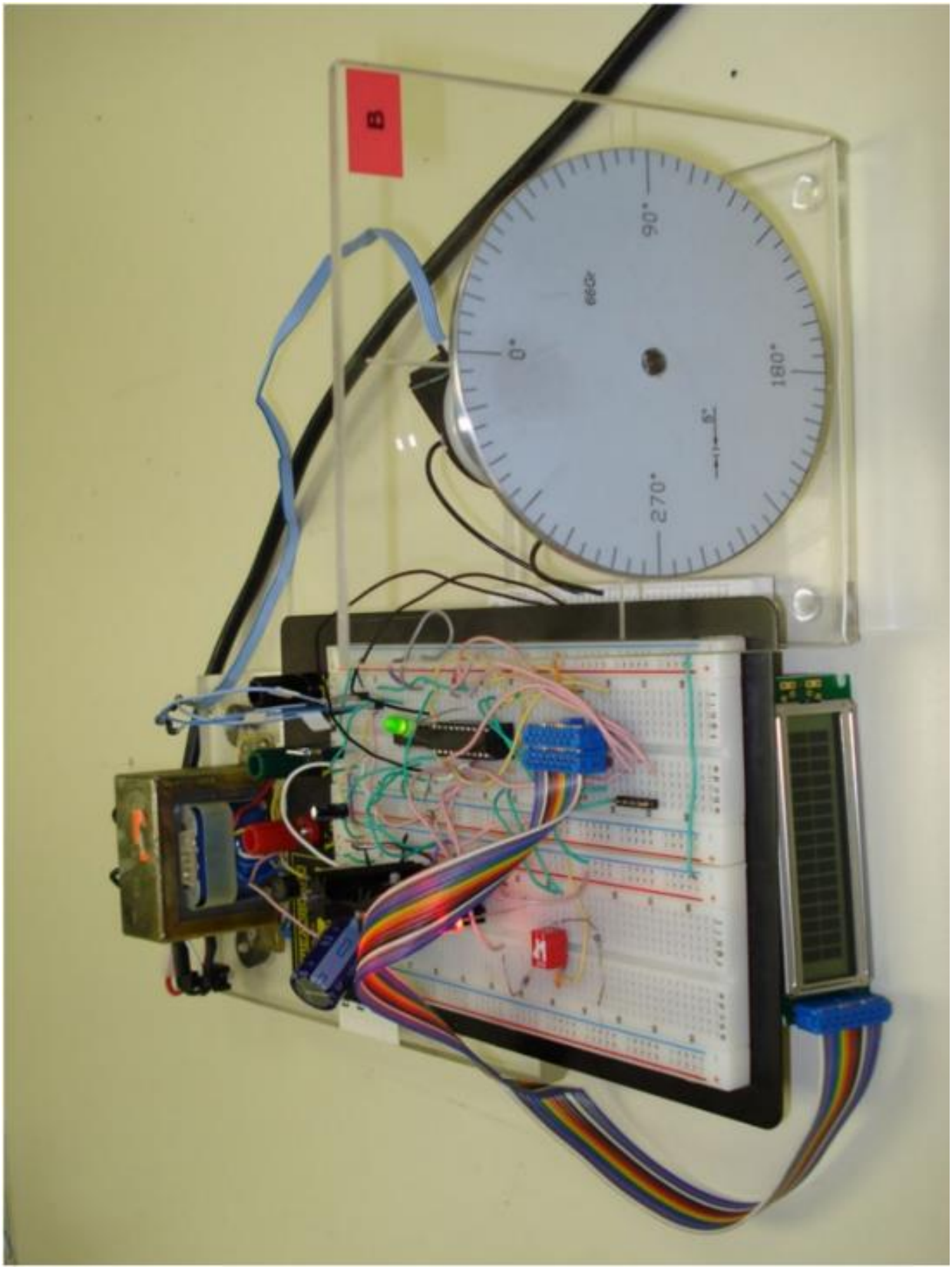
用於測量磁盤位置和
因此，請將此信息用於反饋。整體安裝在有機玻璃上
如圖 1.1 所示。有關此機械零件概念的更多詳細信息
將在本卷的下一章中給出。

1.2 電子電路設計

在電子部分，工程師必須設計電路，以確保機電一體化系統的功能。它涵蓋了所需的集成
電子零件，例如電阻器，電容器，集成電路以及所選的微型 crocontroller 或微控制器。不同的所需的穩壓電壓

組件也是此步驟的一部分。電子電路的主要部分是微控制器或一組微控制器。在本卷中，我們決定使用一個微控制器的一種類型，即 Microchip 生產的 dsPIC30F4011。我們不能給出真正的理由，而我們的願望是採用一種微控制器，我們將在本卷中介紹所有示例。此選擇將也使讀者可以輕鬆實現實時實施，因為我們將使用所有示例的結構相同。
調節電壓將取決於我們將在

要求其數據手冊中的電壓介於 2.5 V 至 5V。由於大多數示例使用直流致動器並驅動它們，因此我們需要一個我們可以使用 DAC 或僅 PWM 以及集成的模擬信號電路名 L293D（H 橋）。該集成電路需要穩壓 5 V，它將提供信號輸出，該信號將為 0 V 之間的直流電動機供電和 24V。我們還使用了許多需要調節電壓才能工作的傳感器正確地。這些設備中的大多數都需要為加速度計和需要較少調節電壓的陀螺儀（請參見兩輪機器人）。



直流電動機套件（1.1）-（1.2）給出了直流電動機的電子電路的概念

我們將在本卷中使用的工具包。

控制機械部件，可以採用兩種結構。這些結構是

由圖 1 和 2 所示。 1.3-1.4。

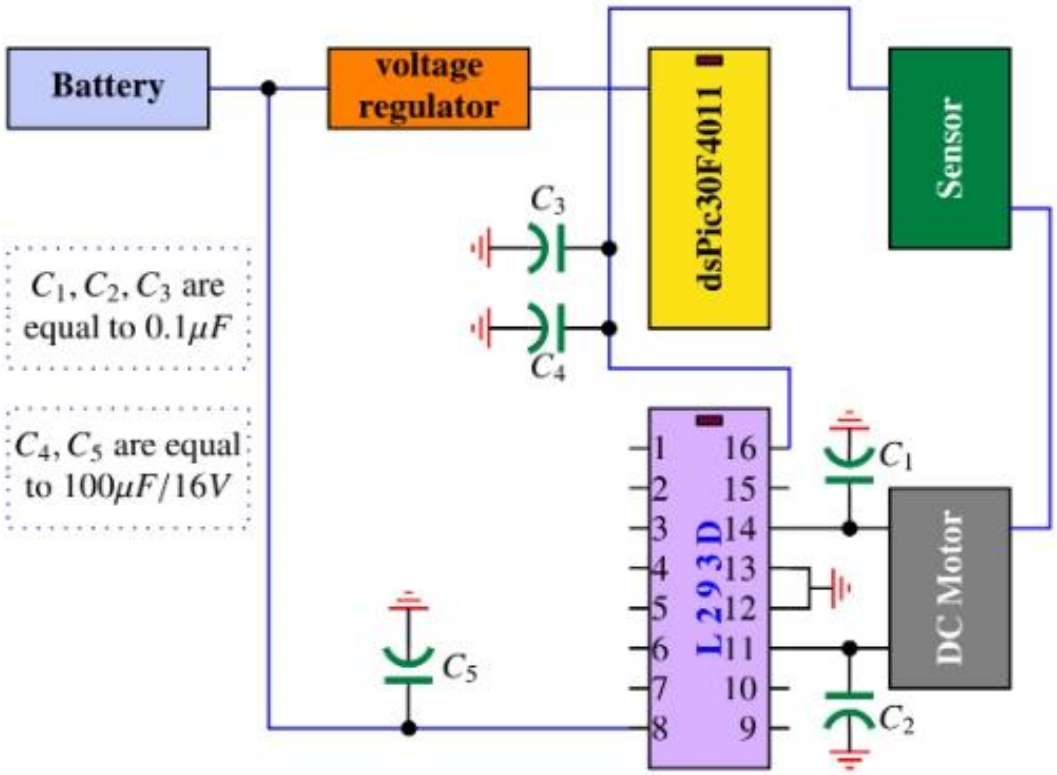


Fig. 1.2 Electronic circuit of the dc motor kit

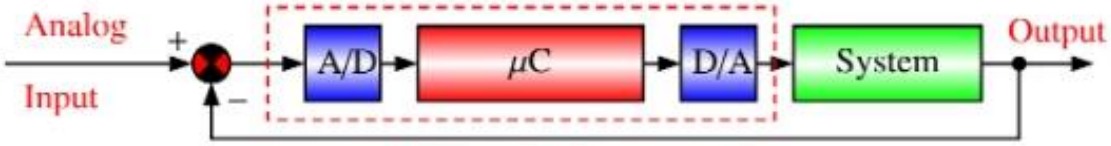


Fig. 1.3 Signal conversion made in the forward path

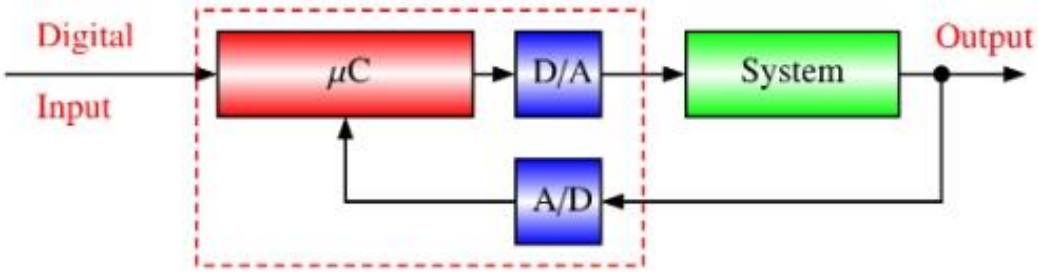


Fig. 1.4 Signal conversion made in the feedback path

如果比較這兩種結構，在第一個中指出是模擬的，而第二個是數字的。第二種結構具有可以消除噪音的優勢。在本

卷的其餘部分中，我們將採用這種結構。

該結構的功能很簡單，可以解釋如下。

微控制器無限循環運行，在每個中斷處，微控制器使用傳感器和 ADC 讀取輸出值，然後使用控件讀取該算法計算出控制動作，並通過 DAC 發送到系統。所有這些步驟在中斷例程中完成。避免錯誤計算和錯誤量化微控制器或 ADC 的位數選擇是一個重要的問題。對於 micrcontroller，可以選擇 16 位，提供了很好的精度，而對於 ADC，所有示例都將使用 10 位我們正在介紹。這不會帶來很好的精度，但是結果是可以接受的。如果我們返回到實時實施設置，則其電子電路已建立 dsPIC30F4011 周圍。 PWM 模塊用於將電壓提供給 L293D 集成電路依次提供必要的功率來驅動執行器。編碼器用於測量小磁盤的位置以及通過簡單的計算速度。

1.3 實時實施

在控制部分，工程師必須分析正在研究和設計的系統適當的控制器以獲得所需的性能。在分析部分，首先應該建立

一個可以接受的模型，輸入和輸出。一旦掌握了動態，採樣週期就是

選擇模型，然後將其轉換為離散時間形式，並在適當的條件下拖曳器可以從經典比例積分和微分（PID）中選擇控制器或狀態反饋控制器或任何其他可以提供理想的表演。為了響應控制規範，控制器結構併計算其參數，然後為我們必須在每個採樣週期內將控制措施終止發送給系統。

在編程部分，工程師輸入所選算法的算法微控制器內存中的文件。許多語言可以用於這個目的。在本卷的其餘部分中，將使用 C 語言來實現開發的算法。同樣，如果我們回到實時實施設置並考慮這種情況 PID 控制器和狀態反饋控制器這兩種簡單算法中的一種。對於這些控制器的控制動作是通過測量來計算的，在所有情況下，控制律的表達都很簡單，應該不要花費超過採樣週期的時間（見圖 1.5）。實施使用中斷概念來完成。以下示例顯示了位置負載的控制。

//

```
// A C program for the dsPic4011 for control the position of a
// dc motor driving a small disk
//
//
// Includes and defines
//
#include <p30f4011.h>
#include <pwm.h>
#include <stdio.h>
#include <stdlib.h>
#include "xlcd.h"
#define ENABLETRIS TRISEbits.TRISE2
#define ENABLE LATEbits.LATE2
#define ENCODER_PRIORITY 7
#define CONTROLLER_PRIORITY 5
#define DISPLAY_PRIORITY 2
#define Ts 0.005; // 1.0/200;
#define Fs 200.0;
typedef struct {
float KP; // Proportional gain
float KI; // Integral gain
float KD; // Derivative gain
} PIDstruct;
PIDstruct thePID;
typedef struct {
long Position; // Shaft position
long error[3]; // the errors
long ref; // the reference
double u[2]; // control (actual and past)
}motorData;
motorData themotorData;
//
// dsPic configuration
//
_FOSC(CSW_FSCM_OFF & FRC_PLL16);;
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & MCLR_DIS);
_FGS(CODE_PROT_OFF);
```

```
_FICD( ICS_NONE );
//
// Variables
//
typedef enum _BOOL { FALSE = 0, TRUE } BOOL;
BOOL A;
BOOL B;
BOOL prevA;
BOOL prevB;
unsigned int dutycycle;
//
// Functions
//
// Initialization function
void Initialize(void);
// Interrupt functions
void __attribute__((interrupt, auto_psv)) _CNInterrupt(void);
void __attribute__((__interrupt__)) _T1Interrupt(void);
//
// Main function
//
int main(void)
{
Initialize();
themotorData.ref = 600; // (90 deg)
while(1);
}
//
// Initialize function
//
void Initialize(void)
{
// variables initialization
thePID.KA = 70.14;
thePID.KI = -128.62;
thePID.KD = 58.54;
themotorData.u[0] = 0.0;
themotorData.u[1] = 0.0;
```

```
themotorData.error[0] = 0;
themotorData.error[1] = 0;
themotorData.error[2] = 0;
// Activation of the interrupts priority
INTCON1bits.NSTDIS = 0;
// Digital pins
ADPCFG = 0b11111111;
// I/O
TRISEbits.TRISE0 = 0; // PWM1H
TRISEbits.TRISE1 = 0; // PWM1L
TRISBbits.TRISB2 = 1; // Encoder Chanal A : RB2 -- CN4
TRISBbits.TRISB3 = 1; // Encoder Chanal B : RB3 -- CN5
ENABLETRIS = 0;
/* start-up LCD */
OpenXLCD(FOUR_BIT & LINES_5X7);
//
// initialize variables for the encoder
//
prevA = PORTBbits.RB2;
prevB = PORTBbits.RB3;
//
// Initialize CN interrupts *
//
CNEN1bits.CN0IE=0; // CN0 interrupt disable
CNEN1bits.CN1IE=0; // CN1 interrupt disable
CNEN1bits.CN2IE=0; // CN2 interrupt ENABLE
CNEN1bits.CN3IE=0; // CN3 interrupt ENABLE
CNEN1bits.CN4IE=1; // CN4 interrupt disable
CNEN1bits.CN5IE=1; // CN5 interrupt disable
CNEN1bits.CN6IE=0; // CN6 interrupt disable
CNEN1bits.CN7IE=0; // CN7 interrupt disable
CNEN2bits.CN17IE=0; // CN17 interrupt disable
CNEN2bits.CN18IE=0; // CN18 interrupt disable
IFS0bits.CNIF = 0; // clear CN interrupt flag
IPC3bits.CNIP = ENCODER_PRIORITY; // CN interrupt max priority (7)
IEC0bits.CNIE = 1; // CN interrupt enable
//
// Configure PWM
```

```
//
ConfigIntMCPWM(PWM_INT_DIS & PWM_FLTA_DIS_INT);
SetDCMCPWM(1, 1024, 0);
OpenMCPWM (0x3FF, 0x0, PWM_EN & PWM_IDLE_CON & PWM_OP_SCALE1
& PWM_IPCLK_SCALE1 & PWM_MOD_FREE,
PWM_MOD1_COMP & PWM_PDIS3H & PWM_PDIS2H & PWM_PEN1H
& PWM_PDIS3L & PWM_PDIS2L & PWM_PEN1L,
PWM_SEVOPS1 & PWM_OSYNC_TCY & PWM_UEN);

//
// Initialize Timer 1 interrupt
//
T1CONbits.TON=1; // turn timer 1 on
T1CONbits.TGATE=0;
T1CONbits.TSIDL=0; // stop timer in idle mode (0=non)
T1CONbits.TCKPS=1; // prescaler (0=1:1, 1=1:8, 2=1:64)
T1CONbits.TCS=0; // clock source (0=FOSC/4)
PR1 = 18424; // 200Hz
IFS0bits.T1IF = 0; // clear timer 1 interrupt flag
IPC0bits.T1IP = CONTROLLER_PRIORITY;
IEC0bits.T1IE=1; // enable timer 1 interrupt
//
// Initialize Timer 2 interrupt
//
T2CONbits.TON=1; // turn timer 2 on
T2CONbits.TGATE=0;
T2CONbits.TSIDL=1; // stop timer in idle mode (0=non)
T2CONbits.TCKPS=2; // prescaler (0=1:1, 1=1:8, 2=1:64)
T2CONbits.TCS=0; // clock source (0=FOSC/4)
PR2 = 0xFFFF; // slower possible
IFS0bits.T2IF = 0; // clear timer 2 interrupt flag
IPC1bits.T2IP = DISPLAY_PRIORITY;
IEC0bits.T2IE = 1; // timer 2 interrupt enable
}
//
// C N Interrupt routine
//
// Decode of the position
```

```
void __attribute__((interrupt, auto_psv)) _CNInterrupt(void)
{
if(IFS0bits.CNIF)
{
CNLED = !CNLED;
// Get current Encoder signals
// Must read port before clearing flag!!
A = PORTBbits.RB2;
B = PORTBbits.RB3;
// Compare current signals with previous ones to see which
// one has changed
// Change occurs on A
if(A != prevA){
if(A == B){

themotorData.Position++;

}else{

themotorData.Position--;

}
// Change occurs on B
}else if(B != prevB){
if(A == B){

themotorData.Position--;

}else{

themotorData.Position++;

}
}
// Save current signals for next time
prevA = A;
prevB = B;
IFS0bits.CNIF=0; // clear interrupt flag
```

```
}
} //end of CN_interrupt function
//
// T 1 Interrupt service routine
//
// Sampling period
void __attribute__((__interrupt__)) _T1Interrupt(void)
{
if (IFS0bits.T1IF)
{
// Error
themotorData.error[0] = themotorData.ref - themotorData.Position;
// Control equation
themotorData.u[0] = themotorData.u[1] + thePID.KA*themotorData.error[0];
themotorData.u[0] += thePID.KI*themotorData.error[1];
themotorData.u[0] += tthePID.KD*themotorData.error[2];
// send control
SetDCMCPWM(1, 1024 + (int)(themotorData.u[0]), 0);
// save the actual data
themotorData.u[1] = themotorData.u[0];
themotorData.error[2] = themotorData.error[1];
themotorData.error[1] = themotorData.error[0];
IFS0bits.T1IF = 0; // Clear Timer interrupt flag
}
}
//
// T 2 I N T E R R U P T service routine
//
// LCD
void __attribute__((interrupt, auto_psv)) _T2Interrupt(void)
{
if (IFS0bits.T2IF)
{
while(BusyXLCD());
XLCDLine1();
printf("e: %ld", themotorData.error[0]);
while(BusyXLCD());
XLCDLine2();
```

```
printf("u: %8.3f ", themotorData.u[0]);
IFS0bits.T2IF = 0;
}
}
```

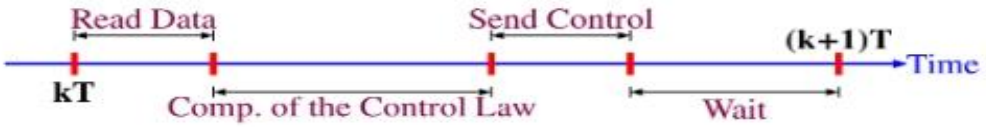


Fig. 1.5 Partition of the sampling period T

例 1.3.1 作為機電系統的第二個例子，讓我們考慮交通信號燈控制系統的設計。我們假設我們有兩條街道，一條主要街道一個擁有 80%的流量，而另一個擁有 20%的流量。圖 1.6 說明了我們正在處理並應為其設計的交通信號燈系統機電一體化系統。我們的目標是設計一種可控制這兩條街道的交通流量。更具體地說，我們必須控制燈光（紅色，黃色和綠色）在每條街道中。周圍大多數常見的交通信號燈

世界由紅色，黃色和綠色三盞燈組成。圖 1.7 給出了在我們的交通系統中使用的燈光。在交通系統的每個角落，我們都放了一個燈為了使行人和駕駛員能夠看到燈光並採取適當的措施行動。

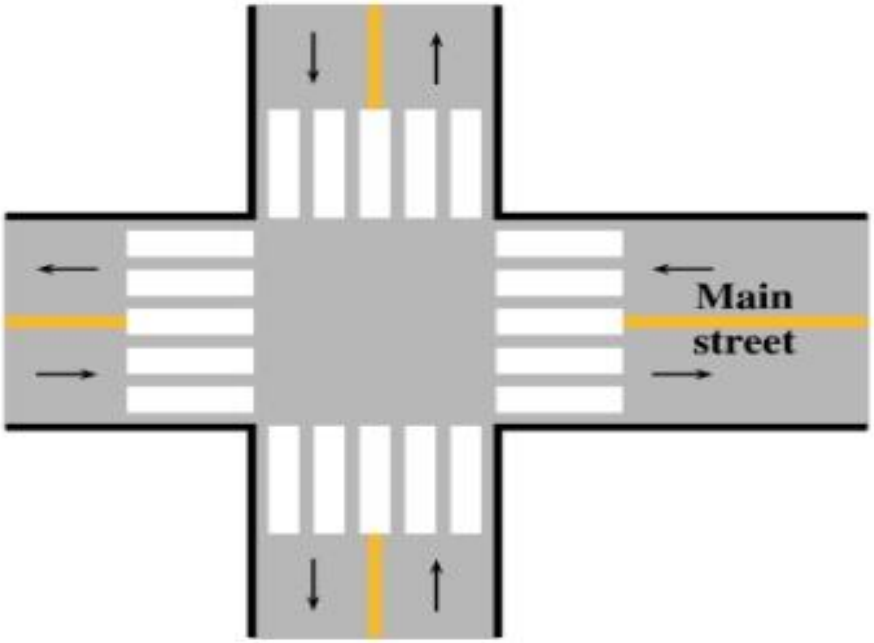


Fig. 1.6 Traffic system

當指示燈變為紅色時，駕駛員必須停下汽車，而當指示燈變為紅色時為綠色，駕駛員有權移動自己的汽車。黃燈用作謹慎的步驟，指示燈即將變成綠色或紅色，並且駕駛員必須採取適當的措施移動或停車。更多當燈光即將從綠色切換為紅色時，通常使用黃色耗時短的中間步驟。每條街道分為兩個方向的兩個方向，每個方向都有兩個車道。賽車可以直行，也可以左右轉彎。我們也有在每個路口控制行人的要求。這些要求是隨機性，必須在短時間內以一定優先級予以考慮。交通信號燈的機電一體化系統是一個簡單的系統，由以下部分組成的：

•位於街道各個角落的照明燈，帶有一些按鈕，用於

行人要求允許過馬路

- 圍繞 dsPIC30F4011 構建的電子電路
- C 語言的控制算法

控制交通的燈位於街道的每個拐角處。這些燈中的一個如圖 1.7 所示。推底也可以幫助行人在安全需要時過馬路。為了模擬我們的交通燈，我們用彩色發光二極管表示燈光（LED）使用與交通燈控制系統相同的顏色。對於行人，使用藍色。將用於控制流量的算法非常簡單，並且除行人的要求外，它以順序方式執行被視為中斷例程。如果我們用 Gmain，Ymain，Rmain，Gsec 表示，

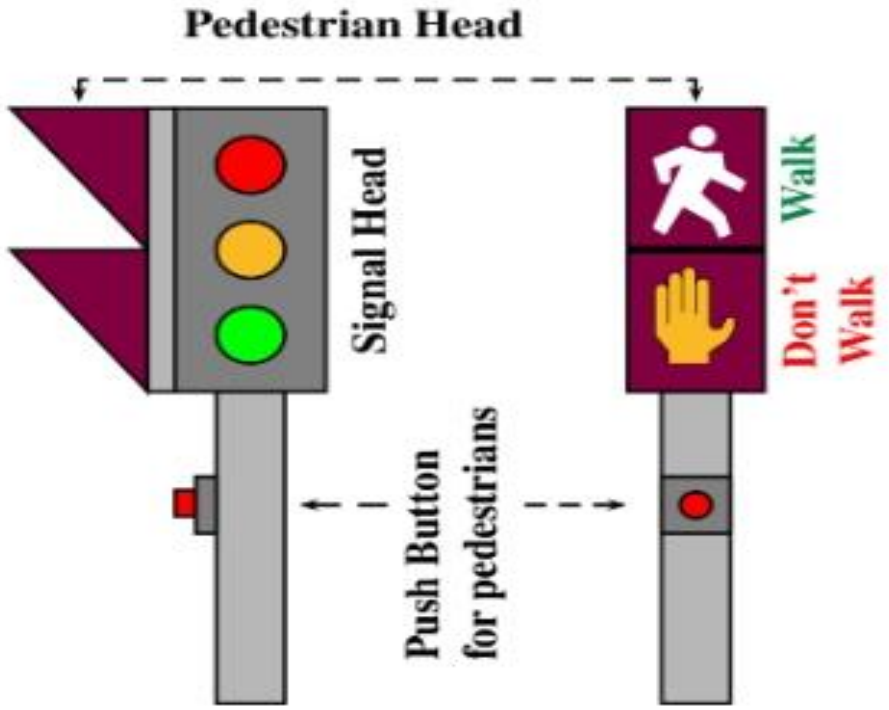


Fig. 1.7 Type of light used in the traffic light system

Ysec，Rsec 分別為主要街道和主要街道的淺綠色，黃色和紅色

次要街道。算法如下：

開始循環

–開啟 Gmain，關閉 Ymain，關閉 Rmain，關閉 Gsec，關閉 Ysec 和 Rsec，以及

等待一段時間

–關閉 Gmain，Ymain 開啟，Rmain 關閉，Gsec 關閉，Ysec 關閉和 Rsec 開啟，以及

等待時間 tswitch

–將 Gmain 關閉，將 Ymain 關閉，將 Rmain 打開，將 Gsec 打開，將 Ysec 關閉和將 Rsec 關閉，以及

等待時間 tsec

–關閉 Gmain，關閉 Ymain，關閉 Rmain，關閉 Gsec，關閉 Ysec 和關閉 Rsec，以及

等待時間 tswitch

結束循環

當發生中斷時，我們確定行人推到了哪個角落按鈕，並通過阻止車輛通行來採取行動，以允許行人以安全的方式過馬路。用於控制燈系統的程序的結構如下：

```
// Include here the headers
#include <dspic30f4011.h>
// Define variables
unsigned int i;
unsigned int Tmax = 65535;
unsigned int tmain = 8;
unsigned int tsec = 4;
unsigned int tswitch = 1;
#define delaytmain() {for i=0;i<tmain*Tmax;i++) Nop(); }
#define delaytsec() {for i=0;i<tsec*Tmax;;i++) Nop(); }
#define delaytswitch() {for i=0;i<tswitch*Tmax;;i++) Nop(); }
#define greenMain RE0 // green light of the main street
#define yellowMain RE1 // yellow light of the main street
#define redMain RE2 // red light of the main street
#define greenSecondary RE3 // green light of the secondary street
#define yellowSecondary RE4 // yellow light of the secondary street
#define redSecondary RE5 // red light of the secondary street
```

```
typedef enum _BOOL { FALSE = 0, TRUE } BOOL;
BOOL A;
BOOL prevA;
// Initialization of the streets
// Main street
MainStreet.green = TRUE;
MainStreet.orange = FALSE;
MainStreet.rouge = FALSE;
// Secondary street
SecondaryStreet.green = FALSE;
SecondaryStreet.orange = FALSE;
SecondaryStreet.rouge = TRUE;
// Assign the dsPic ports to the lights
//
// Functions
//
void Initialize(void);
void __attribute__((interrupt, auto_psv)) _CNInterrupt(void)
//
// main function
//
int main (void)
{
Initialize();
while (1)
{
// tmain
// Main Street during the tmain
greenMain = 1;
yellowMain = 0;
redMain = 0;
// Secondary street during the tmain
greenSecondary = 0;
yellowSecondary = 0;
redSecondary = 1;
delaytmin();
// tswitch
// Main Street during the tswitch
```

```
greenMain = 0;
yellowMain = 1;
redMain = 0;
// Secondary street during the tswitch
greenSecondary = 0;
yellowSecondary = 0;
redSecondary = 1;
delaytswitch();
// tsec
// Main Street during the tsec
greenMain = 0;
yellowMain = 0;
redMain = 1
;
// Secondary street during the tsec
greenSecondary = 1;
yellowSecondary = 0;
redSecondary = 0;
delaytsec();
// tswitch
// Main Street during the tswitch
greenMain = 0;
yellowMain = 0;
redMain = 1;
// Secondary street during the tswitch
greenSecondary = 0;
yellowSecondary = 1;
redSecondary = 0;
delaytswitch();
}
}
void Initialize(void)
{
TRISE = 0x00 // configure the port E as output
//
// initialize variables for the encoder
//
prevA = PORTBbits.RB2;
```

```
//
// Initialize CN interrupts *
//
CNEN1bits.CN0IE=0; // CN0 interrupt disable
CNEN1bits.CN1IE=0; // CN1 interrupt disable
CNEN1bits.CN2IE=0; // CN2 interrupt ENABLE
CNEN1bits.CN3IE=0; // CN3 interrupt ENABLE
CNEN1bits.CN4IE=1; // CN4 interrupt disable
CNEN1bits.CN5IE=1; // CN5 interrupt disable
CNEN1bits.CN6IE=0; // CN6 interrupt disable
CNEN1bits.CN7IE=0; // CN7 interrupt disable
CNEN2bits.CN17IE=0; // CN17 interrupt disable
CNEN2bits.CN18IE=0; // CN18 interrupt disable
IFS0bits.CNIF = 0; // clear CN interrupt flag
IPC3bits.CNIP = ENCODER_PRIORITY; // CN interrupt max priority (7)
IEC0bits.CNIE = 1; // CN interrupt enable
}
//
// C N Interrupt routine
//
// Pedestrian ask to cross
void __attribute__((interrupt, auto_psv)) _CNInterrupt(void)
{
if(IFS0bits.CNIF)
{
CNLED = !CNLED;
// Get the switch signal
// Must read port before clearing flag!!
A = PORTBbits.RB2;
// Compare the current signal with the previous signal to see the change
// Change occurs on A
if(A != prevA){
// put all the red lights on
}
// Save current signal for next time
prevA = A;
IFS0bits.CNIF=0; // clear interrupt flag
}
```

} //end of CN_interrupt function

該程序首先初始化所有變量，然後配置輸入以及 dsPIC30F4011 的輸出。之後，程序進入不確定循環在其中我們執行控制交叉路口燈光的序列。如果一個行人要求允許過馬路，我們縮短了實際時間活動，因為我們不能突然停止活動以防止發生事故。給定的時間被分配給行人過馬路。完成此時間後，循環中的順序恢復。

備註 1.3.1 對於行人，也有可能包括過馬路的權利我們必須在程序中執行的序列中的街道。也遲到了夜晚，因為可能性很小，我們可以消除行人的權利一個行人將在拐角處，他將過馬路。但是隨著中斷解決方案，有可能一直保持相同的算法，不必更改它。

我們可以通過添加適當的代碼來改進算法，使其更加智能記憶每條街道上的隊列並通過調整來適當動作的傳感器每條街道上的燈光時間，以減少駕駛員在街上的等待時間紅綠燈。

這兩個示例提供了有關機電系統以及它們如何實現的想法設計困難且複雜。重要的是要注意給定的解決方案機電一體化系統不是唯一的，它會隨著設計知識的變化而變化

球隊。同樣重要的是要記住，在獲得競爭體系的各個階段。

1.4 本書的組織

這本書可以視為機電一體化課程中的第二門課程，學生應該參加必修課程，其中的結構和已經介紹了機電一體化系統上的不同組件。它僅聚焦連續時間控制系統的分析，設計與實現由微控制器使用高級算法來獲得所需的性能。在建模部分，開發了描述系統行為的模型使用傳遞函數或狀態空間表示。在傳遞函數逼近部分中，連續時間系統的模型 tems 轉換為離散時間系統，並使用不同的技術進行分析和控制器的綜合以保證某些期望的性能得以開發。在狀態空間逼近部分中，連續時間系統的模型為轉換為離散時間狀態空間表示和不同的技術控制器的分析和綜合，以保證某些所需的性能發達。

實施部分將重點介紹如何實施控制我們開發的算法要么是關於傳遞函數的方法，要么是基於狀態空間。硬件和軟件部分都將涵蓋在內讀者對如何處理此類問題的想法。

在高級控制部分，可以使用一些算法來控制系統提出具有不確定性和/或外部干擾的項目以賦予風味

向讀者介紹魯棒控制理論，並向他介紹本研究領域。
在案例研究部分，提供了一些實際示例展示如何實現我們前面介紹的概念以獲得功能機電系統。