

MỤC LỤC

MỞ ĐẦU: Khái quát giới thiệu đề tài	2
CHƯƠNG I: Khái niệm Encoder và thuật toán PID	3
CHƯƠNG II: Giới thiệu phần cứng	8
CHƯƠNG III: Sơ đồ nguyên lý mạch và phần mềm	13
KẾT LUẬN: Vấn đề giải quyết được và những hạn chế. Mở rộng	30
TÀI LIỆU THAM KHẢO	32

I. MỞ ĐẦU

Điều khiển động cơ DC (DC Motor) là một ứng dụng thuộc dạng cơ bản nhất của điều khiển tự động vì DC Motor là cơ cấu chấp hành (actuator) được dùng nhiều nhất trong các hệ thống tự động (ví dụ robot). Do đó, tôi chọn đề tài điều khiển tốc độ động cơ để làm nền tảng phát triển cho các đề tài sau.

Đề tài ứng dụng kiến thức trong các môn học Cơ sở tự động, Lý thuyết điều khiển nâng cao vào áp dụng thực tế để điều khiển tốc độ động cơ.

Thông thường động cơ được chia làm 2 loại:

- **Động cơ bước:** loại động cơ điện có nguyên lý và ứng dụng khác biệt với đa số các động cơ điện thông thường. Chúng thực chất là một động cơ đồng bộ dùng để biến đổi các tín hiệu điều khiển dưới dạng các xung điện rời rạc kế tiếp nhau thành các chuyển động góc quay hoặc các chuyển động của rôto có khả năng cố định rôto vào các vị trí cần thiết.
- **Động cơ servo:** là loại động có gắn thêm bộ encoder dùng để xác định vị trí quay của động cơ cũng như số vòng trong khoảng thời gian xác định.

Động cơ Servo được chọn trong đề tài này, bởi đáp ứng được theo yêu cầu đề tài.

Có khá nhiều thuật toán điều khiển được tốc độ động cơ, em xin trình bày một phương pháp khá dễ được sử dụng phổ biến trong lĩnh vực điều khiển, đó là **phương pháp PID (Proportional , Integral, Derivative)**.

Việc điều khiển được động cơ theo ý muốn của người sử dụng giúp chúng ta mở rộng hơn trong các lĩnh vực điều khiển tự động: điều khiển robot, mô hình CNC, điều khiển tự chỉnh.....

Phương pháp điều khiển đơn giản gồm các bước:

- Nhập các hệ số: KI, KP, KD và vận tốc mong muốn (vòng/s) từ máy tính.
- Điều chỉnh giá trị Output từ việc đọc số xung, tính ra sai số... giá trị Output chính là giá trị PWM xuất ra cho động cơ.
- Sau mỗi thời gian cố định, ta lại cho Vi Xử Lý xuất lên máy tính.
- Việc xuất từ máy tính lên VXL là liên tục để có thể chỉnh được tốc độ trên máy tính.
- Mục tiêu đề ra là đặt được tốc độ mong muốn với các yêu cầu: chính xác (accurate), nhanh (fast response), ổn định (small overshoot).

Để thực hiện cách bước trên, ta cần triển khai

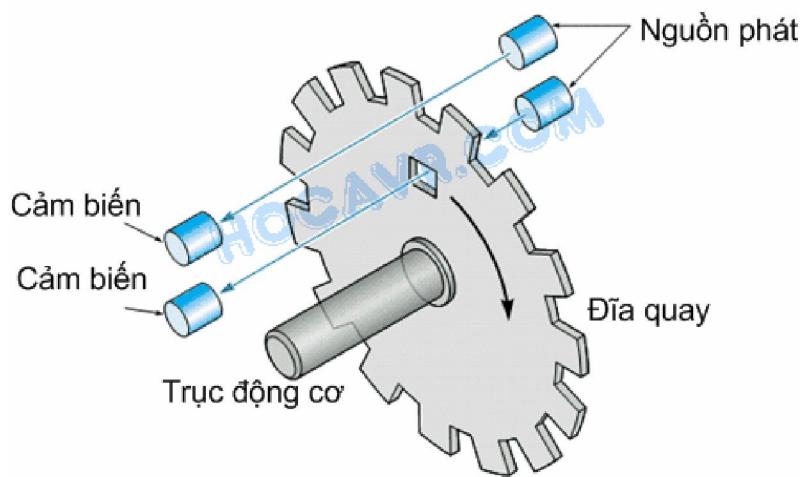
- Mạch cầu H, ta dùng IC LM298N (thiết kế cho một động cơ)

- Một bộ nguồn 0 – 24V, dòng 3A
- Kit LaunchPad MSP430 G2
- Động cơ Servo

II. CHƯƠNG 1

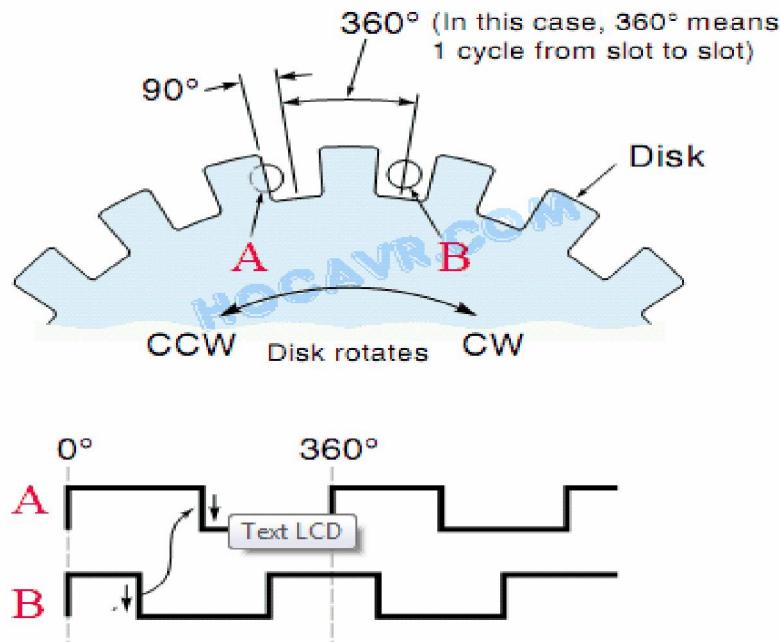
1. KHÁI NIỆM ENCODER

- Có 2 Encoder thông dụng: **Encoder tuyệt đối** và **Encoder tương đối**.
- Tuy nhiên ta chỉ xét đến Encoder tương đối bởi giá thành rẻ, đáp ứng được khả năng điều khiển tốt
- Hình mô tả Encoder loại này ở phía dưới



Encoder thường có 3 kênh (3 ngõ ra) bao gồm kênh A, kênh B và kênh I (Index). Trong ta thấy hãy chú ý một lỗ nhỏ bên phía trong của đĩa quay và một cặp phat-thu dành riêng cho lỗ nhỏ này. Đó là kênh I của encoder. Cứ mỗi lần motor quay được một vòng, lỗ nhỏ xuất hiện tại vị trí của cặp phat-thu, hỏng ngoại từ nguồn phát sẽ xuyên qua lỗ nhỏ đến cảm biến quang, một tín hiệu xuất hiện trên cảm biến. Như thế kênh I xuất hiện một “xung” mỗi vòng quay của motor. Bên ngoài đĩa quay được chia thành các rãnh nhỏ và một cặp thu-phát khác dành cho các rãnh này. Đây là kênh A của encoder, hoạt động của kênh A cũng tương tự kênh I, điểm khác nhau là trong 1 vòng quay của motor, có N “xung” xuất hiện trên kênh A. N là số rãnh trên đĩa và được gọi là độ phân giải (resolution) của encoder. Mỗi loại encoder có độ phân giải khác nhau, có khi trên mỗi đĩa chỉ có vài rãnh nhưng cũng có trường hợp đến hàng nghìn rãnh được chia. Để điều khiển động cơ, bạn phải biết độ phân giải của encoder đang dùng. Độ phân giải ảnh hưởng đến độ chính xác điều khiển và cả phương pháp điều khiển. Không được vẽ trong hình 2, tuy nhiên trên các encoder còn có một cặp thu phát khác được đặt trên cùng đường tròn với

kênh A nhưng lệch một chút (lệch M+0,5 rãnh), đây là kênh B của encoder. Tín hiệu xung từ kênh B có cùng tần số với kênh A nhưng lệch pha 90o. Bằng cách phối hợp kênh A và B người đọc sẽ biết chiều quay của động cơ.



Hình trên thể hiện sự bộ trí của 2 cảm biến kênh A và B lệch pha nhau. Khi cảm biến A bắt đầu bị che thì cảm biến B hoàn toàn nhận được hồng ngoại xuyên qua, và ngược lại. Hình thấp là dạng xung ngõ ra trên 2 kênh. Xét trường hợp motor quay cùng chiều kim đồng hồ, tín hiệu “đi” từ trái sang phải. Hãy quan sát lúc tín hiệu A chuyển từ mức cao xuống thấp (cạnh xuống) thì kênh B đang ở mức thấp. Ngược lại, nếu động cơ quay ngược chiều kim đồng hồ, tín hiệu “đi” từ phải qua trái. Lúc này, tại cạnh xuống của kênh A thì kênh B đang ở mức cao. Như vậy, bằng cách phối hợp 2 kênh A và B chúng ta không những xác định được góc quay (thông qua số xung) mà còn biết được chiều quay của động cơ (thông qua mức của kênh B ở cạnh xuống của kênh A).

Cách đọc xung Encoder:

- **Dùng Timer – Capture:** ta sẽ ghi nhận thời gian của một xung khi Encoder kích cạnh lên hay cạnh xuống từ đó suy ra được tốc độ xung/s. Tuy nhiên cách này khá phức tạp trong việc Config thanh ghi cũng như tốn thêm một bộ Timer
- **Sử dụng ngắt ngoài:** chỉ đơn giản là nối kênh A của encoder với 1 ngắt ngoài (P1.5), mỗi lần vào ngắt ta sẽ tăng hay giảm biến đếm xung tùy thuộc đang quay cùng chiều hay ngược chiều kim đồng hồ. Tuy nhiên ta cũng nên rất chú trọng đến việc xác định tần số ngắt ngoài. Ví dụ: một động cơ có độ phân giải 2000 xung/vòng và động cơ quay tốc độ 100 vòng/s => tần số ngắt của động cơ là

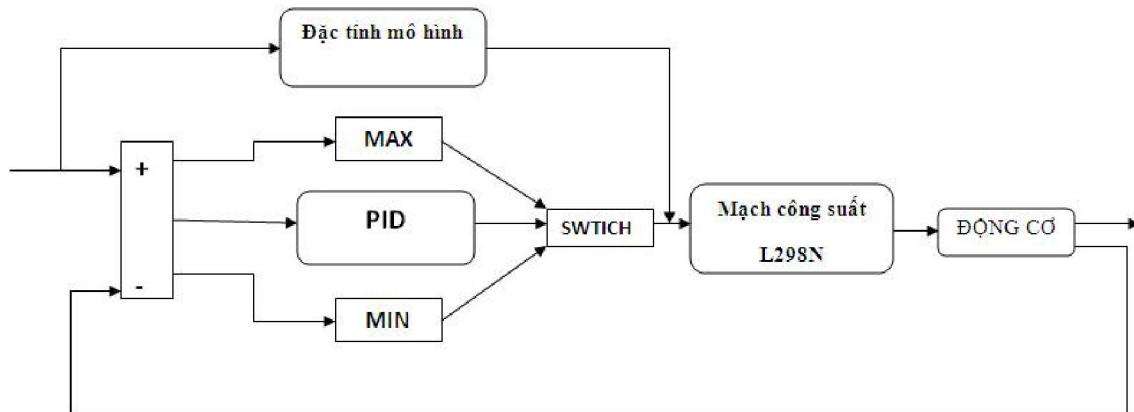
200KHZ, tương đương cứ 5us xảy ra một ngắt, thời gian ngắt quá nhanh nên MCU sẽ không kịp xử lý các lệnh khác. Ở đây ta sử dụng động cơ động phân giải 400 xung/vòng với động cơ DC 15 vòng/s => thời gian ngắt là 170us, phù hợp MCU hoạt động với tần số 1MHZ.

2. KHÁI NIỆM PID (Propotional , Integral, Derivative)

- Tuy xuất hiện từ lâu nhưng đến nay PID vẫn là giải thuật điều khiển được dùng nhiều trong các ứng dụng điều khiển tự động.
- Mục tiêu đề ra là đặt được tốc độ mong muốn với các yêu cầu: chính xác (accurate), nhanh (fast response), ổn định (small overshoot).

$$u = K_p * e + K_d * (de/dt) + K_i * \int e dt .$$

với: - e là sai số giữa tốc độ thực tế với tốc độ đặt.
u là tín hiệu điều khiển.



Mô hình điều khiển động cơ

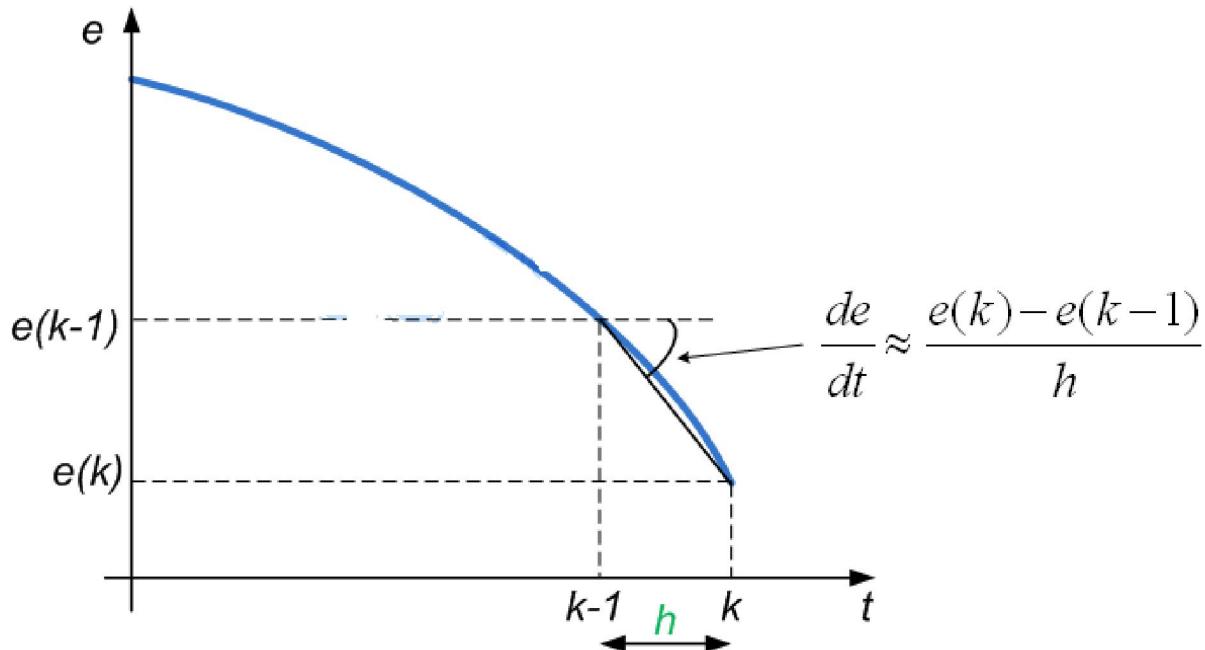
2.1 Điều khiển PID số:

- Công thức của bộ điều khiển PID trình bày ở trên là dạng hàm liên tục của biến e, trong đó có cả thành phần tuyến tính, đạo hàm và tích phân. Tuy nhiên, hệ thống máy tính và vi điều khiển lại là hệ thống số. Muốn xây dựng bộ điều khiển PID trên máy tính hay trên vi điều khiển chúng ta phải biết cách xấp xỉ phương trình liên tục thành dạng rời rạc.

- Rõ ràng thời gian lấy mẫu càng nhỏ (tần số cao) thì việc hiệu chỉnh càng tiến gần đến sự “liên tục” và chất lượng điều khiển sẽ tốt hơn. Trong các bộ điều khiển số, thời gian lấy mẫu là một yếu tố rất quan trọng. Cần tính toán để thời gian này không quá lớn nhưng cũng đừng quá nhỏ, vì như thế sẽ hao phí thời gian thực thi.
- Vì bộ điều khiển PID xây dựng trong MSP430 sẽ là bộ điều khiển số, chúng ta cần xấp xỉ công thức của bộ điều khiển này theo các khoảng thời gian rời rạc.
- Trước hết, thành phần P tương đối đơn giản vì đó là quan hệ tuyến tính $K_p * e$, chúng ta chỉ cần áp dụng trực tiếp công thức này mà không cần bất kỳ xấp xỉ nào. Tiếp đến là xấp xỉ cho đạo hàm của biến e . Vì thời gian lấy mẫu cho các bộ điều khiển thường rất bé nên có thể xấp xỉ đạo hàm bằng sự thay đổi của e trong 2 lần lấy mẫu liên tiếp:

$$de/dt = (e(k) - e(k-1))/h.$$

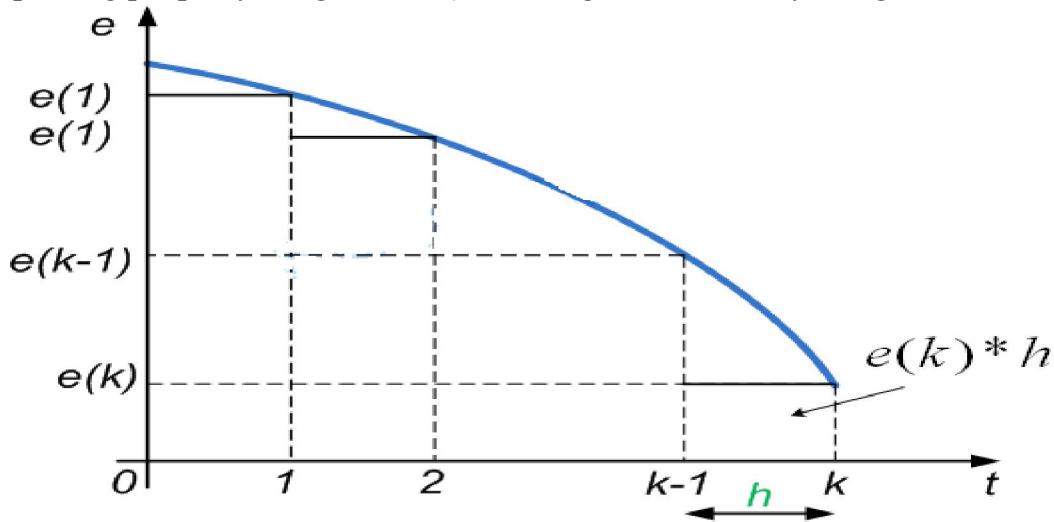
Trong đó $e(k)$ là giá trị hiện tại của e , $e(k-1)$ là giá trị của e trong lần lấy mẫu trước đó và h là khoảng thời gian lấy mẫu (h là hằng số).



Hình 8. Xấp xỉ đạo hàm của biến sai số e

Thành phần tích phân được xấp xỉ bằng diện tích vùng giới hạn bởi hàm đường biểu diễn của e và trục thời gian. Do việc tính toán tích phân không cần quá chính xác,

chúng ta có thể dùng phương pháp xấp xỉ đơn giản nhất là xấp xỉ hình chữ nhật (sai số của phương pháp này cũng lớn nhất). Ý tưởng được trình bày trong hình.



Hình 9. Xấp xỉ tích phân của biến sai số e

Tích phân của biến e được tính bằng tổng diện tích các hình chữ nhật tại mỗi thời điểm đang xét. Mỗi hình chữ nhật có chiều rộng bằng thời gian lấy mẫu h và chiều cao là giá trị sai số e tại thời điểm đang xét.

Tổng quát:

$$\int_0^t e dt = \sum_0^k e(k) * h \quad (4)$$

Tổng hợp các xấp xỉ, công thức của bộ điều khiển PID số được trình bày trong (5):

$$u = K_p * e + \frac{(e(k) - e(k-1))}{h} + \sum_0^k e(k) * h \quad (5)$$

Trong đó u là đại lượng output từ bộ điều khiển. Để đơn giản hóa việc tính thành phần tích phân, chúng ta nên dùng phương pháp “cộng dồn” (hay đệ quy):

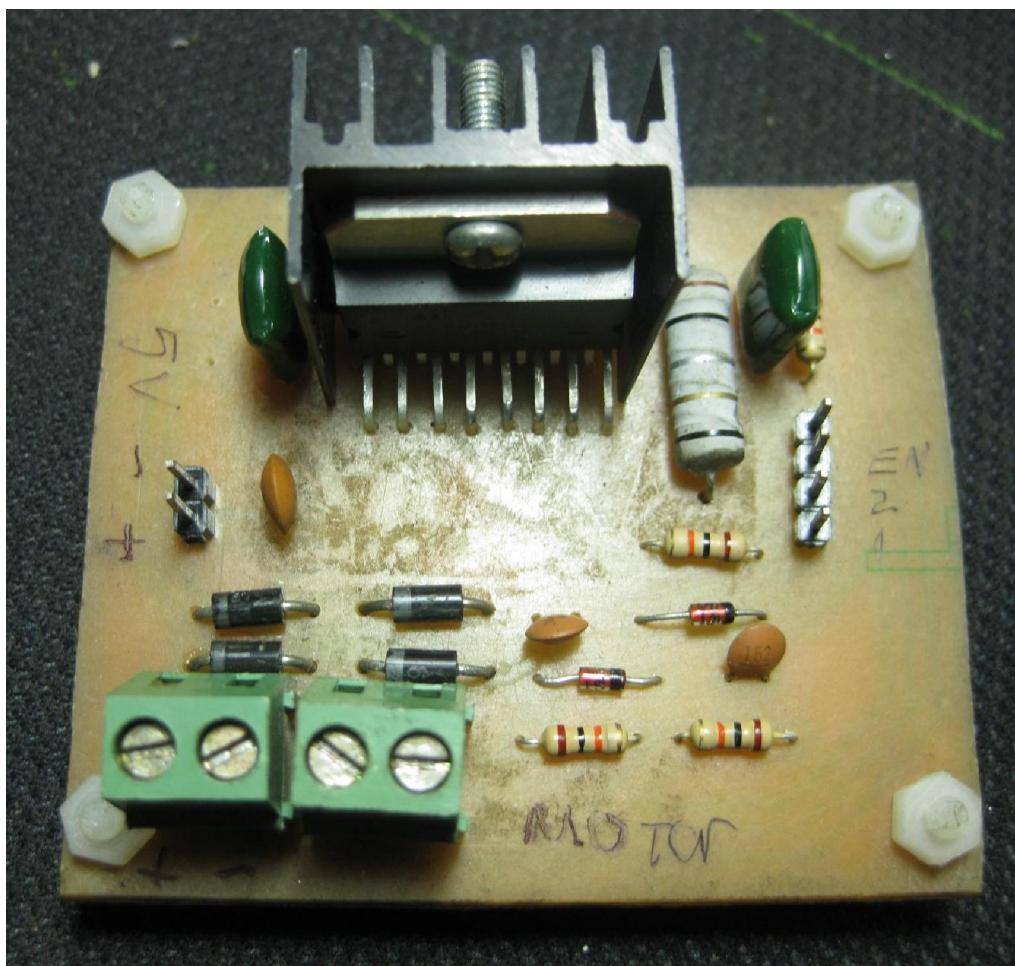
$$I(k) = I(k-1) + e(k) * h$$

(6)

Với $I(k)$ là thành phần tích phân hiện tại và $I(k-1)$ là thành phần tích phân trước đó. Các công thức (5) và (6) rất dễ dàng để thực hiện bằng MSP430. Do đó, đến lúc này chúng ta đã sẵn sàng để đưa ý tưởng vào lập trình cho chip.

III. CHƯƠNG 2

1. GIẢI TÍCH CHỨC NĂNG TỪNG THÀNH PHẦN LINHKIỆN TRONG BOARD MẠCH MẠCH CẦU H



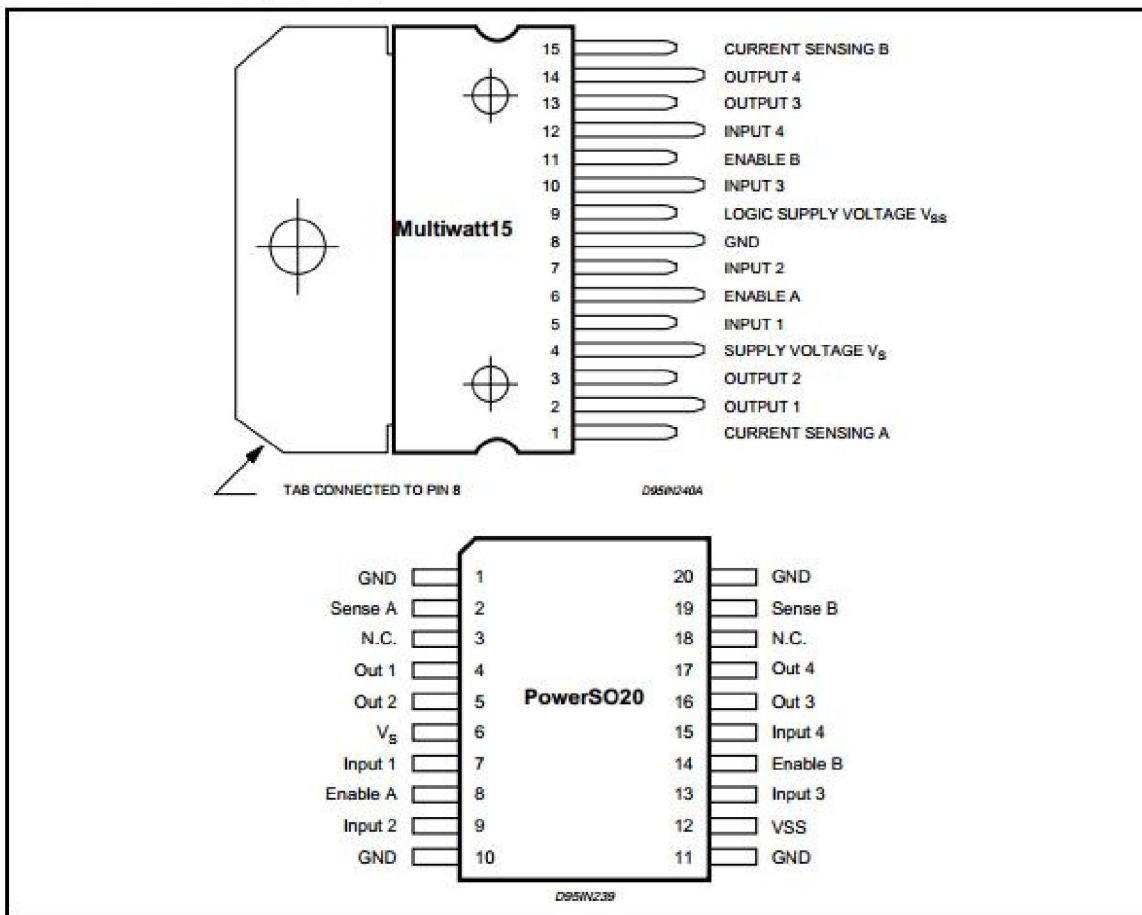
Gồm IC L298N

- Nguồn cung cấp tối đa 50V
- Với dòng tối đa 4A
- Độ bão hòa áp thấp
- Khả năng chịu được nhiệt độ cao
- Điều khiển tối đa được 4 động cơ DC



Sơ đồ chân IC L298N

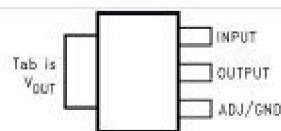
PIN CONNECTIONS (top view)



2. MẠCH NGUỒN



- Áp vào: nguồn AC xoay chiều 24V
- Áp ra: nguồn DC 3.3V, 5V, 12V và 24V
Dòng ra có thể tới 3A
- Gồm các IC nguồn:
 - LM1117:



TOP View

- Áp vào 2.8V, 2.5V, 2.85V, 3.3V, 5V
- Áp ra 3.3V

- **HQ LM78XX:**

- **LM7805:** tạo áp ra 5V
- **LM7812:** tạo áp ra 12V
- **LM7824:** tạo áp ra 24V
- Áp vào: **24V**

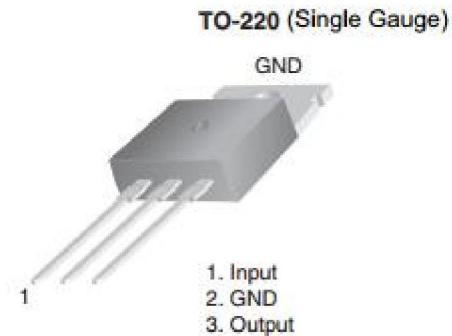
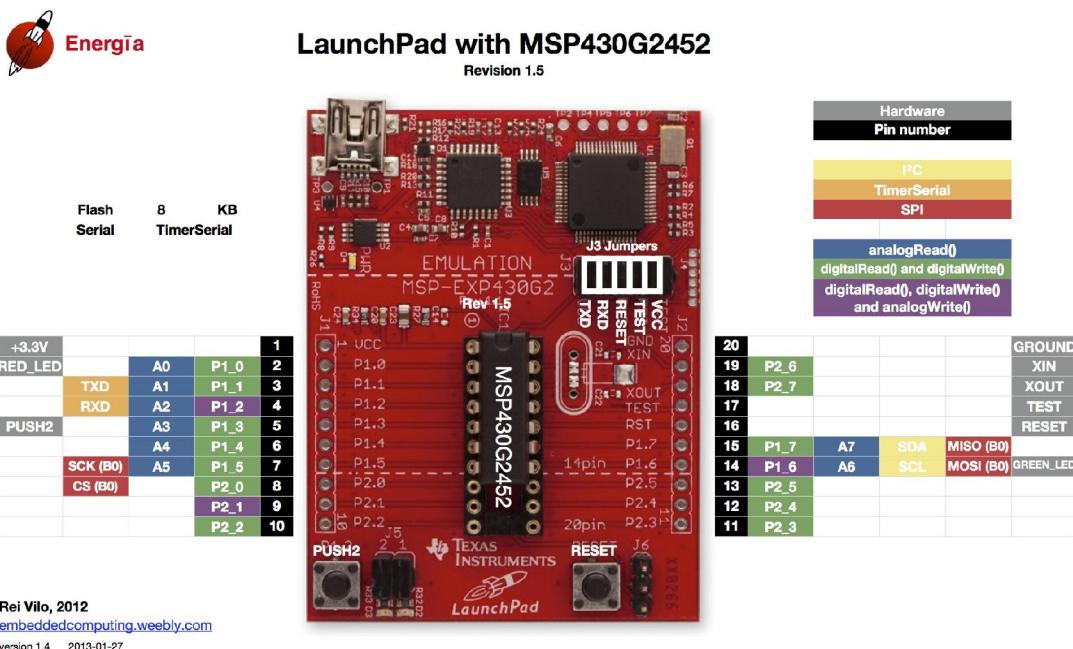


Figure 2.

3. KIT LAUCHPAD MSP430 G2



- MCU: **MSP430G2553**
- Nguồn vào tâm **1.8V – 3.6V**
- Xung Clock gồm các chế độ:
 - Tần số nội bộ lên tới **16MHz** với 4 hệ số hiệu chỉnh
 - **32kHz** thạch anh
 - Nguồn clock từ ngoại vi
- Hai timer 16bit: **TimerA0** và **TimerA1** với **3** thanh ghi Capture/Compare
- Uart
- **TXD P1.1**

- RXD P1.2
- 3 PORT PWM: **P1.2 - P1.6 – P2.1**

4. ĐỘNG CƠ DC

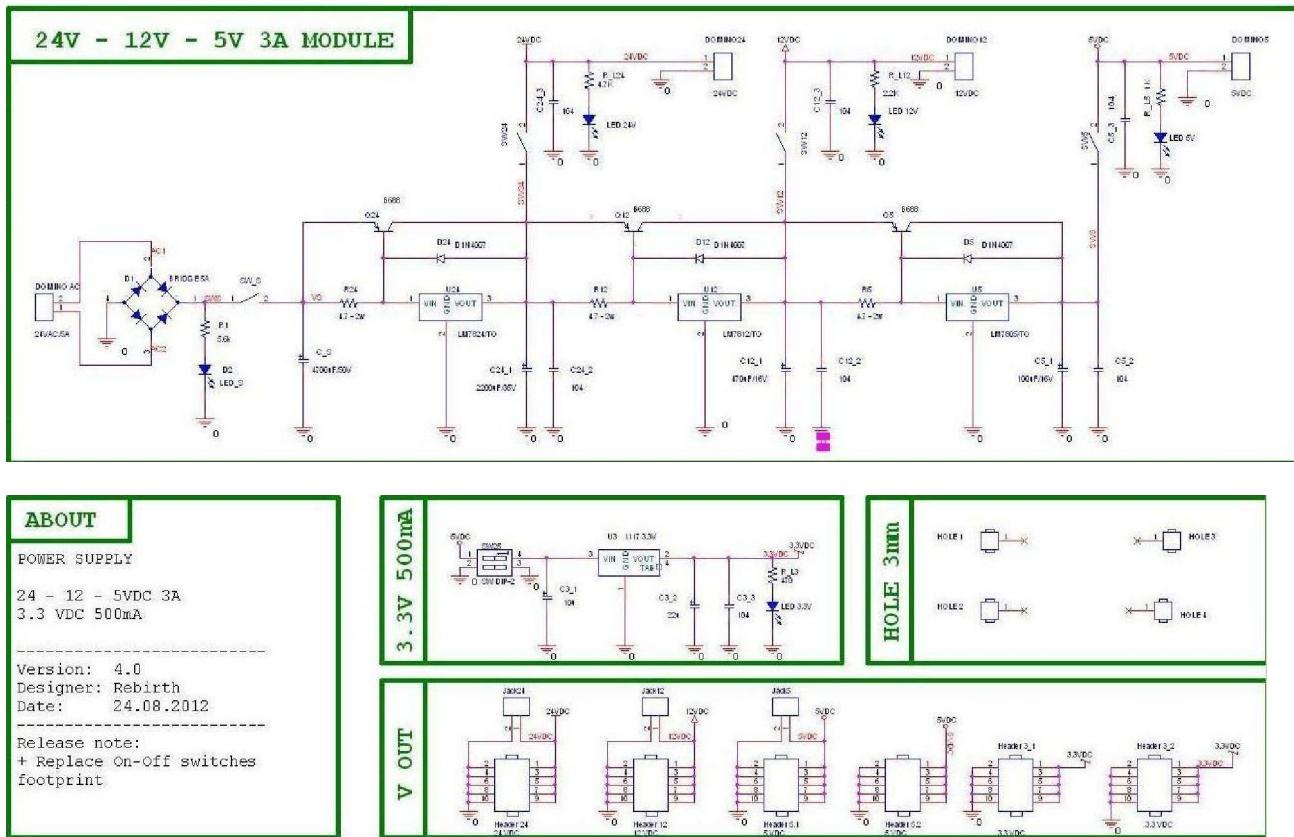
- ❖ Encoder **400 xung/ vòng**
- ❖ Nguồn cấp tối đa **24V**
- ❖ Không có số hiệu động cơ nên không tra được tốc độ động cơ. Trong lúc thực hành đó được động cơ quay tối đa **16 vòng/s** với áp vào **12V**



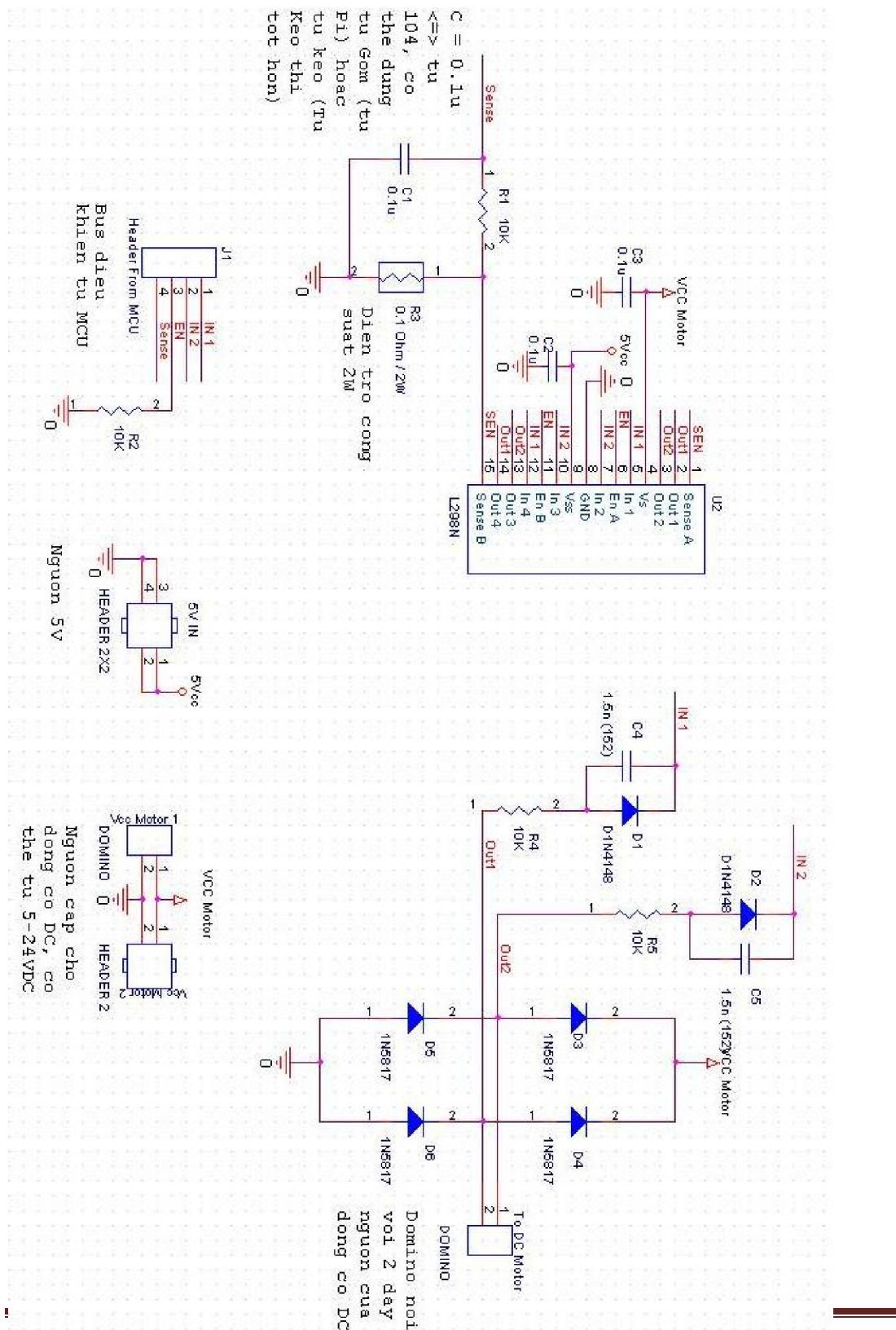
IV. CHƯƠNG 3

THIẾT KẾ SƠ ĐỒ MẠCH DÙNG LAYOUT VÀ CAPTURE CỦA ORCAD 9.2

1. SƠ ĐỒ NGUỒN: (THAM KHẢO TỪ CLB PAYITFOWARD)

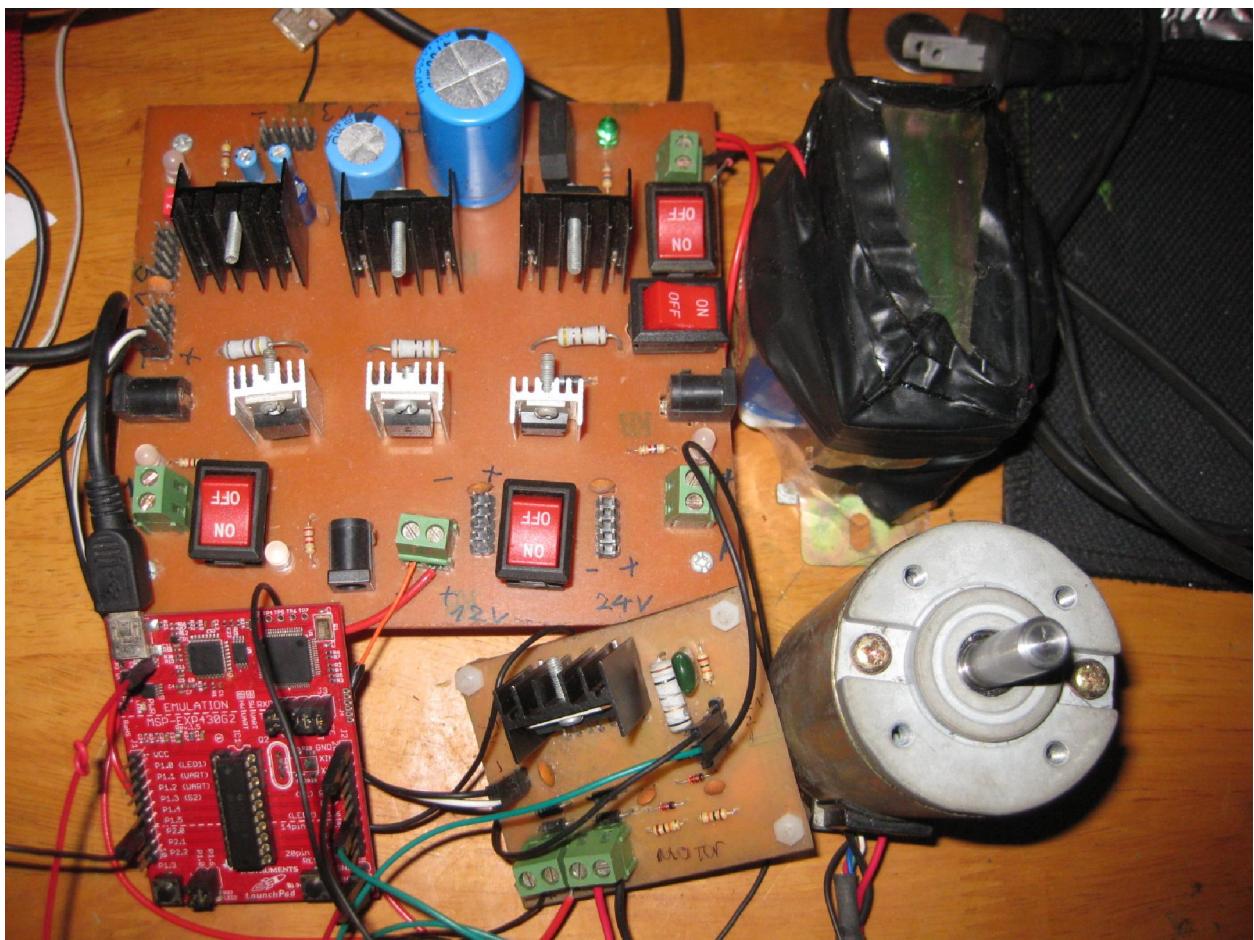


2. SƠ ĐỒ MẠCH CẦU H – L298N



- ❖ Do chỉ chỉ điều khiển một động cơ DC nên **OUT1** nối chung **OUT4** và **OUT2** nối chung **OUT3**, EN được nối chungENA và ENB
- ❖ Các bước để dùng mạch:
 - Cáp nguồn DC 5V cho IC
 - Cáp nguồn DC 5 – 24V cho động cơ
 - Chân điều khiển:
 - *IN1, IN2* cấp xung PWM (tối đa 3.3V)
 - *EN*: tích cực cao, cho phép IC hoạt động
 - *Sense*

Từ đó, ta được bộ mạch hoàn chỉnh sau:



3. THIẾT KẾ PHẦN MỀM:

 **COMPOSE CODE STUDIO v5.2 (CCS)**

 **VISUAL STUDIO (CSHARP)**

- ❖ Như phần trình bay lý thuyết PID ở trên, ta sẽ điều khiển động DC theo cách sau:
- ❖ Gửi dữ liệu liên tục từ máy tính lên Vi xử lý: các hệ số Kp, Ki, Kd và vận tốc mong muốn (vòng/ s)
- ❖ Đếm số xung quay được trong 25ms (dùng ngắt Port để đếm xung và ngắt Timer trong 25ms)
- ❖ Tính toán các số pPart , iPart, dPart => Xung PWM cấp cho động cơ
- ❖ Cứ 250ms ta xuất ra kết quả lên máy tính qua UART
- ❖ Lưu ý: do cổng COM sẽ xảy ra trường hợp vừa xuất và vừa nhận đồng thời, để dễ xử lý, ta dùng thêm một VECTOR ngắt PORT tránh hiện tượng xung đột cổng COM

A. PHẦN VI XỬ LÝ (MSP430G2553) - CCS

```

1 #include <msp430.h>
2 #include "UART.h"
3 volatile long int Sample_count,Pulse,Prepulse,Err,Ctrl_speed,pPart,iPart,dPart,PreErr,Output;
4 volatile long int Kp,Kd,Ki,Rspeed,Sample_time,inv_Sample_time;
5 char b[9];
6 void config_init(){
7     //PWM Config 0 -> 10000
8     P1DIR |= BIT6;
9     P1SEL |= BIT6;
10    TAOCCR0 = 8000;
11    TAOCCR1 = 0;
12    TAOCCCTL1 = OUTMOD_7;
13    TAOCTL = TASSEL_2 + MC_1;
14    //SET FREQUENCY MCU 1MHZ
15    BCSCTL1 = CALBC1_1MHZ;
16    DCOCTL = CALDCO_1MHZ;
17    WDTCTL = WDTPW + WDTHOLD;
18    // INTERRUPT PORT
19    P1IE |= BITS;
20    P1IES &= ~BITS;
21    P1IFG &= ~BITS;
22    //SET TIMER 25MS
23    TA1CCTL0 = CCIE;// CCRO interrupt enabled
24    TA1CCR0 = 25000;
25    TA1CTL = TASSEL_2 + MC_1;
26}
27 void config_value(){
28     Kp=0;Ki=0;Kd=0;
29     dPart=0;iPart=0;pPart=0;
30     Err = 0;PreErr=0;
31     Sample_time=25;
32     inv_Sample_time=40;
33     Ctrl_speed = 0 ;
34     Sample_count = 0;
35}

```

```

42 void getuart(){
43     int i;
44     for (i=0;i<10;i++) (b[i]=uart_getc());}
45     for (i=0;i<9;i++) (b[i]=b[i]-48);}
46     Kd = b[0]*10+b[1];
47     Ki = b[2]*10+b[3];
48     Kp = b[4]*10+b[5];
49     Ctrl_speed = b[6]*100+b[7]*10+b[8];
50     Ctrl_speed= Ctrl_speed*5/2;
51 }
52
53 void main(){
54     config_init();
55     config_value();
56     uart_init();
57
58     __enable_interrupt();
59     _BIS_SR(GIE);
60     while (1){
61         if (Sample_count >=10){
62             Rspeed=Rspeed/5*2;
63             uart_put_num(Rspeed,0,0);
64             b[9]=10;
65             uart_putc(b[9]);
66             Sample_count=0;
67         }
68     }
69 }
70

```

```

73 #pragma vector=TIMER1_A0_VECTOR
74 __interrupt void Timer_A (void)
75 {
76
77     Rspeed = Pulse - Prepulse;
78     Prepulse = Pulse;
79     Err = Ctrl_speed - abs(Rspeed);
80     PreErr = Err;
81     // PID
82     pPart = Kp*Err;
83     dPart = Kd * (Err - PreErr) * inv_Sample_time ;
84     iPart += Ki * Err * Sample_time/1000 ;
85
86     Output+= pPart + dPart + iPart;
87
88     if (Output >= 8000) Output= 8000 - 1;
89     if (Output <= 0) Output= 1;
90
91     TAOCCR1 = Output;
92     Sample_count++;
93 }
94
95 #pragma vector = PORT1_VECTOR
96 __interrupt void P1_ISR(void){
97     Pulse++;
98     P1IFG &= ~BIT5;// Clear P1.5 interrupt flag
99
100 }
101
102 #pragma vector=USCIAB0RX_VECTOR
103 __interrupt void USCIAB0RX_ISRR(void)
104 {
105     getuart();
106 }

```

Trước hết ta set tốc độ cho MCU: tần số **1MHZ** tương đương thời gian **1us** cho một chu kì máy và **tắt WatchDog Timer** để chạy chương trình ở **dòng 15 – 17**

Sau đó **Config Port P1.6** là chân *PWM* (dùng **TIMERA0** thanh ghi **CCR1**) và dùng **TIMERA1** thanh ghi **CCR0**.

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS / SIGNALS ⁽¹⁾					
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x=1 ⁽²⁾	JTAG Mode	CAPD.y
P1.6/ TA0.1/ UCB0SOMI/ UCB0SCL/ A8 ⁽²⁾ / CA6 TDI/TCLK/ Pin Osc	6	P1.x (I/O)	I: 0; O: 1	0	0	0	0	0
		TA0.1	1	1	0	0	0	0
		UCB0SOMI	from USCI	1	1	0	0	0
		UCB0SCL	from USCI	1	1	0	0	0
		A8	X	X	X	1 (y = 6)	0	0
		CA6	X	X	X	0	0	1 (y = 6)
		TDI/TCLK	X	X	X	0	1	0
		Capacitive sensing	X	0	1	0	0	0

P1.6 có rất nhiều chức năng, để khởi động chân P1.6 là chân PWM, ta chọn theo **dòng 8,9** dùng **TimerA0 CCR1**

Timer gồm:

TACTL: thanh ghi điều khiển

12.3.1 TACTL, Timer_A Control Register

	15	14	13	12	11	10	9	8
	Unused						TASSELx	
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
	7	6	5	4	3	2	1	0
	IDx		MCx		Unused	TACLR	TAIE	TAIFG
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
Unused	Bits 15-10	Unused						
TASSELx	Bits 9-8	Timer_A clock source select						
		00 TACLK						
		01 ACLK						
		10 SMCLK						
		11 INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)						
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock.						
		00 /1						
		01 /2						
		10 /4						
		11 /8						
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.						
		00 Stop mode: the timer is halted.						
		01 Up mode: the timer counts up to TACCR0.						
		10 Continuous mode: the timer counts up to 0FFFFh.						
		11 Up/down mode: the timer counts up to TACCR0 then down to 0000h.						
Unused	Bit 3	Unused						
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.						
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request.						
		0 Interrupt disabled						
		1 Interrupt enabled						
TAIFG	Bit 0	Timer_A interrupt flag						
		0 No interrupt pending						
		1 Interrupt pending						

Dòng 13, 25 chọn cả 2 Timer chạy theo Mode 1: Mode up và nguồn xung nội

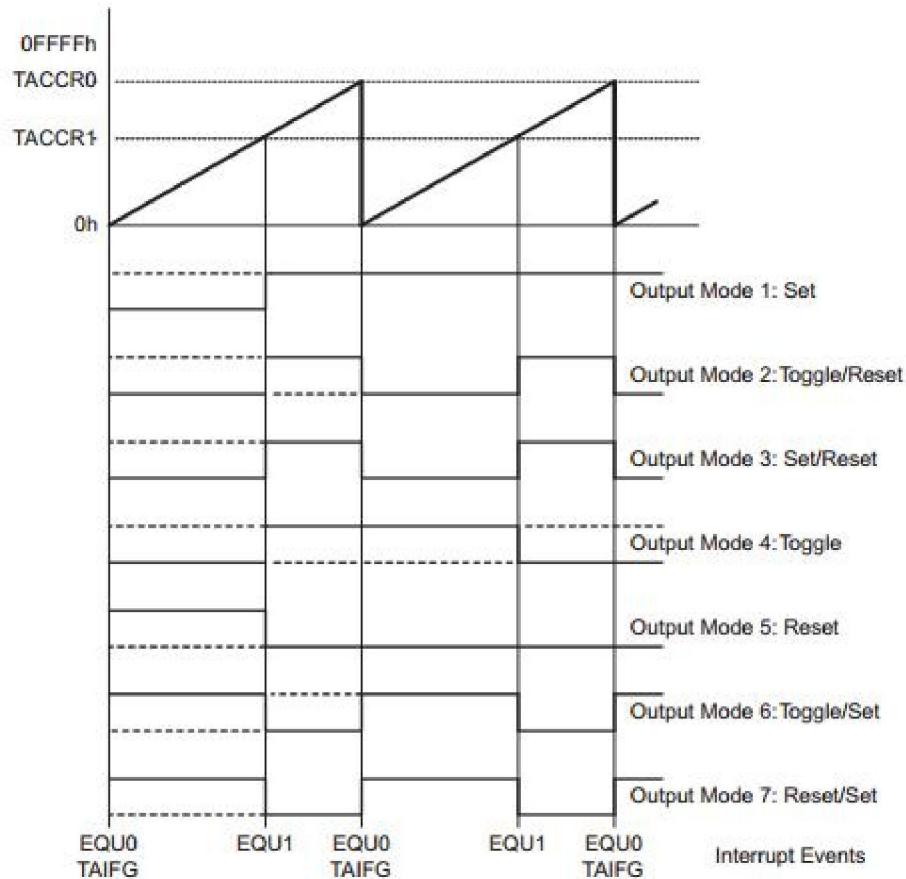
Table 13-1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of compare register TBCL0.
10	Continuous	The timer repeatedly counts from zero to the value selected by the CNTLx bits.
11	Up/down	The timer repeatedly counts from zero up to the value of TBCL0 and then back down to zero.

Do TimerA0 CCR1 xuất PWM nên ta sẽ config cho thanh ghi TACCTL với chế độ MOD_7

111	Reset/Set	The output is reset when the timer counts to the TACCRx value. It is set when the timer counts to the TACCR0 value.
-----	-----------	---

Tức là Timer này sẽ đọc tới giá trị CCR1 và CCR0 thì sẽ đảo giá trị



Dòng 10,11 set giá trị thanh ghi **CCR0=8000** và **CCR1= 0**(PWM 0%) **cho P1.6**

Dòng 24, set giá trị cho thanh ghi **TA1_0 25000** (1us * 25000 = 25ms) tức cứ 25ms xảy ra một lần ngắt và bật ngắt ở **dòng 23**

Tiếp đến ta cần config cho **P1.5** để tạo ngắt khi **Chân A của động cơ** có điện áp thay đổi **dòng 19 – 21**, tạo cò ngắt Port.

Để đơn giản hóa chương trình, ta viết một file riêng UART và gọi nó thông qua dòng 2 CODE

```

1 #ifndef UART_H_
2 #define UART_H_
3 #ifndef SMCLK_F
4 #define SMCLK_F 1000000 // frequency of Sub-System Master Clock in Hz
5 #endif
6 #define BAUDRATE 9600 // may be ... 1200, 2400, 4800, 9600, 19200, ...
7 #define UART_RX_INT_EN 0
8
9 void uart_init();
10 void uart_putc(char c);
11 void uart_puts(const char *s);
12 void uart_put_num(unsigned long val, char dec, unsigned char neg);
13 char uart_data_ready();
14 char uart_getc();
15 void uart_gets(char *s);
16#endif

```

Đặt tốc độ xung **UART 1MHZ** (lưu ý: đặt cùng tốc độ xung MCU nếu không dữ liệu Send/Receive sẽ sai)

Tốc độ Baud 9600

Đặt tên các hàm qua **dòng 9 – 15**

Sau đó ta Config thanh ghi cho Uart qua file **UART.C**

```

1 P1SEL |= BIT1 + BIT2;                                // Set P1.1 as RXD
2 P1SEL2 |= BIT1 + BIT2;                               // Set P1.2 as TXD
3 UCA0CTL0 |= UCMODE_0;
4 UCA0CTL1 |= UCSSEL_2;
5 tempfactor = SMCLK_F/BAUDRATE;
6 UCA0BRO = (unsigned char) tempfactor&0x00FF;
7 tempfactor >>= 8;
8 UCA0BR1 = (unsigned char) (tempfactor&0x00FF);
9 UCA0MCTL |= UCBRF_0 + UCBRS_0;
10 UCA0CTL1 &= ~UCSRST;
11 if (UART_RX_INT_EN)                                // Enable USCI_A0 RX interrupt
12 IE2 |= UCA0RXIE;

```

Dòng 1,2: quy định chức năng Uart cho P1.1 và P1.2

Dòng 3, chon Mode 0: Mode Uart với xung clock nội **dòng 4**

Dòng 11,12: *bật ngắt* cho Uart lúc Receive dữ liệu

Quay trở lại chương trình chính

Dòng 27-35, ta khởi tạo các giá trị ban đầu trước khi bắt đầu chương trình

Sampe_time - thời gian lấy mẫu và **inv_Sample_time** là ngịch đảo thời gian lấy mẫu ($1/(25\text{ms})$) ta tính trước để **giảm** mức độ tính toán cho MCU

Dòng 42 – 51, thực hiện chức năng nhận dữ liệu từ máy tính lấy theo kiểu kí tự char. Dữ liệu nhận được gồm 10 bit:

- 2 bit đầu là hệ số KD
- 2 bit tiếp theo là hệ số KI
- 2 bit sau là hệ số KP
- Vận tốc mong muốn 3 bit còn lại
- **Bit cuối cùng** có giá trị 10: cho biết kết thúc 1 dòng (xem trong bảng mã ASCII)

Dòng 45, chuyển kí tự sang số bằng cách trừ đi 48

Trước khi vào hàm Main ta xem trước các VECTOR NGẮT

a) **NGẮT PORT**

- Khi có sự thay đổi P1.5, cờ IFG của Port sẽ được set lên 1, và đi vào vùng ngắt, ta tăng Pulse lên 1, sau đó phải set lại cờ ngắt IFG ở dòng 95 – 100;

b) **NGẮT UART RECEIVE (dòng 102 – 106)**

- Ở đây ta chỉ cần cập nhật các thông số từ máy tính gửi lên

c) **NGẮT TIMER**

- Cứ 25ms, ta sẽ tính toán lại giá trị **Output** cho PWM như sau:
- Tính số xung trong 25ms: **Rspeed = Pulse – Prepulse** (Prepulse là giá trị xung trước đó) - **dòng 77**
- Cập nhật lại giá trị **Prepulse** - **dòng 78**
- Ta tính lại **sai số vận tốc** giữa **vận tốc mong muốn** và **vận tốc hiện tại** để tính giá trị thành phần P, I, D
- Do số xung **Rspeed** có thể âm nên ta lấy **độ lớn** của nó – **dòng 79**
- **Cập nhật** lại giá trị sai số trước đó - **dòng 80**
- **Dòng 82 – 86**, tính các thành phần P, I, D và giá trị **Output**.

Lưu ý:

- ❖ Dòng 84, ta cộng dồn để quy nén **iPart += Ki * Err * Sample_time/1000** (chia 1000 vì ta tính theo chuẩn giây trong khi đó thời gian lấy mẫu là 25ms)
- ❖ Ngoài ra, **Output += pPart + dPart + iPart**, do khi đạt tới tốc độ mong muốn thì **Err = 0**, trong khi đó ta cần đặt một giá trị cố định PWM nên giá trị **Output** phải bằng giá trị trước đó, nên ta dùng phương pháp cộng dồn trong việc điều khiển vận tốc
- ❖ **Dòng 88 – 89**, xét trường hợp bão hòa (saturation) khi **Output** vượt quá giới hạn cho phép của PWM (xén 2 đầu)

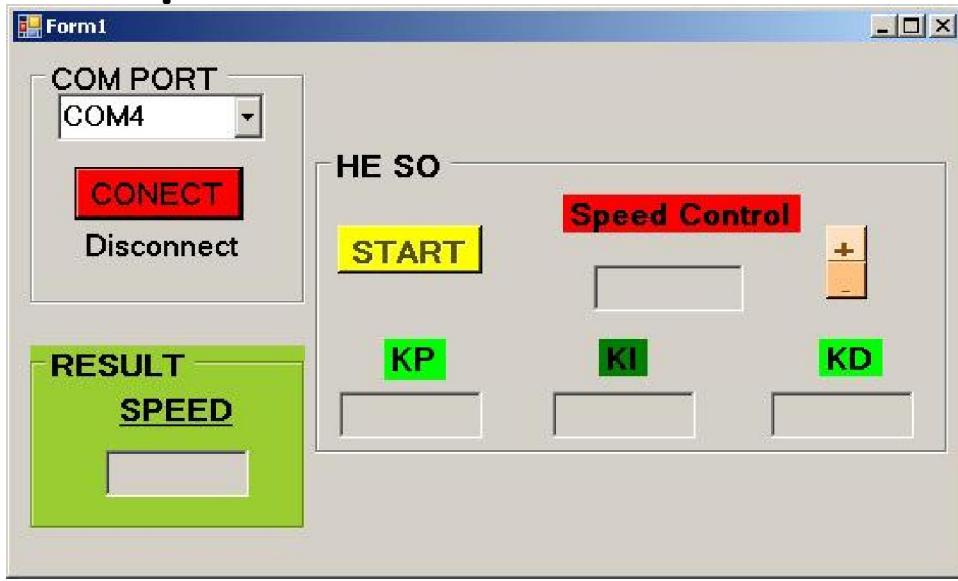
Bây giờ ta quay lại với hàm main() **dòng 53 – 69**

- ❖ Khởi tạo giá trị thanh ghi và giá trị **dòng 54 – 56**
- ❖ Cho phép MCU bật ngắt **dòng 58 – 59**
- ❖ Do tần số xuất kết quả lên máy tính rất cao **40 lần** trong 1s (1/25ms) khiến việc nhận dữ liệu có khả năng sai nên ta quy định mỗi lần **Sample_count = 10** (tương đương $10 \times 25\text{ms} = 0.25\text{s}$) xuất ra kết quả.

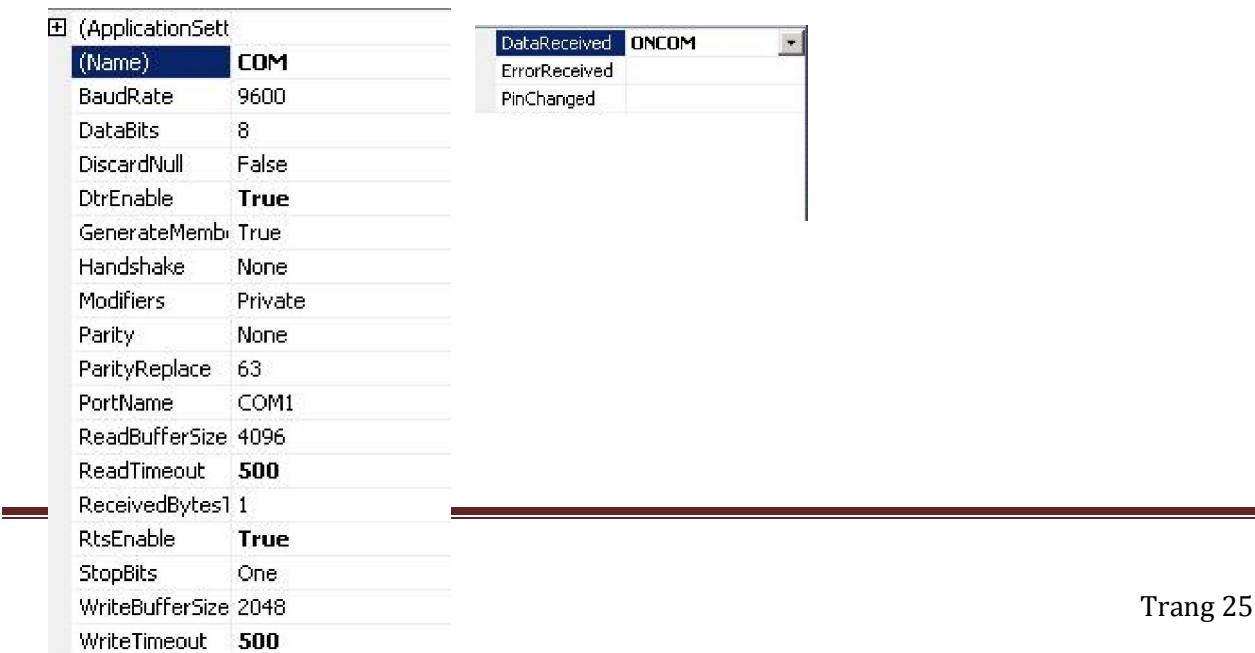
- ❖ Cách thức truyền giữa máy tính và VXL là **Writeline**, nên kết thúc bằng kí tự kết thúc dòng giá trị **10 – 0AH** để máy tính biết kết thúc dữ liệu truyền ; **dòng 64 – 65**.

Kết thúc phần code cho **MSP430G2553**, bây giờ ta xử lý phần C# (VS2010)

Giao diện:



Xác lập giá trị cổng COM, thay vì chọn các giá trị, ta chọn giá trị mặc định cho cổng COM và bật cổng **ngắt** cho COM với tên **ONCOM**



Dùng 2 timer: **Timer1** và **Timer2**. **Timer1**: dùng cập nhật cổng COM, bật sẵn và **Timer2** dùng truyền dữ. Tuy nhiên việc truyền và nhận song song như vậy sẽ khiến dữ liệu khó quản lý được trong lúc truyền nhận => Ta sẽ cho phép **Writeline** khi cập nhật giá trị vận tốc và bỏ **Timer2**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
namespace PID_CONTROL
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Khoa het cac Texbox va Button khi chua Connect cong COM
            txtKD.Enabled = false;
            txtKI.Enabled = false;
            txtKP.Enabled = false;
            PbStart.Enabled = false;
            Pbup.Enabled = false;
            Ptdown.Enabled = false;
            txtResult.Enabled = false;
            txtSpeed.Enabled = false;
            txtKD.Text = "";
            txtKI.Text = "";
            txtKP.Text = "";
            txtResult.Text = "";
            txtSpeed.Text = "";
        }

        private void PbConect_Click(object sender, EventArgs e)
        {


---


```

```

if (Lbss.Text == "Disconnect")
{
    COM.PortName = Cbcom.Text; //Chon cong COM ket voi voi VXL
    COM.Open();
    Lbss.Text = "Connect";
    PbConect.Text = "Disconnect";
    // Sau khi nhan Connect cho phep nhap du lieu de gui len VXL
    txtKD.Enabled = true;
    txtKI.Enabled = true;
    txtKP.Enabled = true;
    PbStart.Enabled = true;
    Pbup.Enabled = true;
    Pbdown.Enabled = true;
    txtResult.Enabled = true;
    txtSpeed.Enabled = true;
}
else
{
    // Khi tat cong Ket noi ta set lai gia tri ban dau nhu khi Form Load
    COM.Close();
    Lbss.Text = "Disconnect";
    PbConect.Text = "Connect";
    txtKD.Enabled = false;
    txtKI.Enabled = false;
    txtKP.Enabled = false;
    PbStart.Enabled = false;
    Pbup.Enabled = false;
    Pbdown.Enabled = false;
    txtResult.Enabled = false;
    txtSpeed.Enabled = false;
    txtKD.Text = "";
    txtKI.Text = "";
    txtKP.Text = "";
    txtResult.Text = "";
    txtSpeed.Text = "";
}
}

// Su dung Timer 1 de cap nhat ten cac cong COM khi dang ket noi
int intlen = 0;
private void timer1_Tick(object sender, EventArgs e)
{
    string[] ports = SerialPort.GetPortNames();
    if (intlen != ports.Length)
    {
        intlen = ports.Length;
        Cbcom.Items.Clear();
        for (int j = 0; j < intlen; j++)
        {
            Cbcom.Items.Add(ports[j]);
        }
        Cbcom.Text = ports[0];
    }
    timer1.Enabled = false;
}

```

```

        }

        string s;
        string s1, s2, s3, s4;

    private void PbStart_Click(object sender, EventArgs e)
    {
        int a, b, c, d;

        if ((txtKD.Text == "") | (txtKI.Text == "") | (txtKP.Text == "") | (txtSpeed.Text == ""))
            MessageBox.Show("Chua nhap cac he so");
        else
        {
            a = Convert.ToInt32(txtKD.Text);
            b = Convert.ToInt32(txtKI.Text);
            c = Convert.ToInt32(txtKP.Text);
            d = Convert.ToInt32(txtSpeed.Text);
            if ((a > 99) | (b > 99) | (c > 99))
            {
                MessageBox.Show("Cac he so <100");
            }
            else
            {
                if (PbStart.Text == "START")
                {
                    PbStart.Text = "STOP";
                    // Xu ly cac he so KI KP KD truoc khi gui du lieu
                    if (txtKD.TextLength == 1) s1 = "0" + txtKD.Text;
                    if (txtKI.TextLength == 1) s2 = "0" + txtKI.Text;
                    if (txtKP.TextLength == 1) s3 = "0" + txtKP.Text;
                    if (txtSpeed.TextLength == 1) s4 = "00" + txtSpeed.Text;
                    if (txtSpeed.TextLength == 2) s4 = "0" + txtSpeed.Text;
                    txtKD.Enabled = false;
                    txtKI.Enabled = false;
                    txtKP.Enabled = false;
                    // Chuoi sau khi du lieu gom 9 bit: KI, KD, KP moi hieu so 2 bit,
                    Van toc mong muon 3 bit
                    s = s1 + s2 + s3 + s4;
                    txtResult.Text = s;
                    COM.WriteLine(s);
                }
                else
                {
                    PbStart.Text = "START";
                    txtKD.Enabled = true;
                    txtKI.Enabled = true;
                    txtKP.Enabled = true;
                    txtSpeed.Enabled = true;
                    // Muon dung Dong co ta chi viec gui 00000000
                    s = "00000000";
                    COM.WriteLine(s);
                }
            }
        }
    }

```

```

        }

    }

    private delegate void Dldisplay(string Value);
    private void display(string Value)
    {
        if (txtResult.InvokeRequired )
        {
            Dldisplay sd = new Dldisplay(display);
            txtResult.Invoke(sd, new object[] { Value });
        }
        else
        {
            txtResult.Text = Value;
        }
    }
    string data;
    private void ONCOM(object sender, SerialDataReceivedEventArgs e)
    {
        // Vector ngat khi nhan du lieu
        if ((PbConect.Text == "Disconnect") )
        {
            data = (string)COM.ReadLine();
        }
        display(data);

    }

    private void Pbup_Click(object sender, EventArgs e)
    {
        double a;
        Double.TryParse(s4,out a); // Chuyen string sang so
        a = a + 1; // tang bien len 1 khi nhan nut UP
        s4=Convert.ToString(a); // Chuyen nguoc sang sang chuoi
        txtSpeed.Text = s4;
        // Xu ly chuoi truoc khi gui du lieu len VXL
        if (txtSpeed.TextLength == 1) s4 = "00" + txtSpeed.Text;
        if (txtSpeed.TextLength == 2) s4 = "0" + txtSpeed.Text;
        s = s1 + s2 + s3 + s4;
        COM.WriteLine(s);
    }

    private void Pbdown_Click(object sender, EventArgs e)
    {
        double a;
        Double.TryParse(s4, out a); // Chuyen string sang so
        a = a - 1; // giam bien len 1 khi nhan nut UP
        s4 = Convert.ToString(a); // Chuyen nguoc sang sang chuoi
        txtSpeed.Text = s4;
        // Xu ly chuoi truoc khi gui du lieu len VXL
    }

```

```

        if (txtSpeed.TextLength == 1) s4 = "00" + txtSpeed.Text;
        if (txtSpeed.TextLength == 2) s4 = "0" + txtSpeed.Text;
        s = s1 + s2 + s3 + s4;
        COM.WriteLine(s);
    }
}
}

```

V. KẾT LUẬN

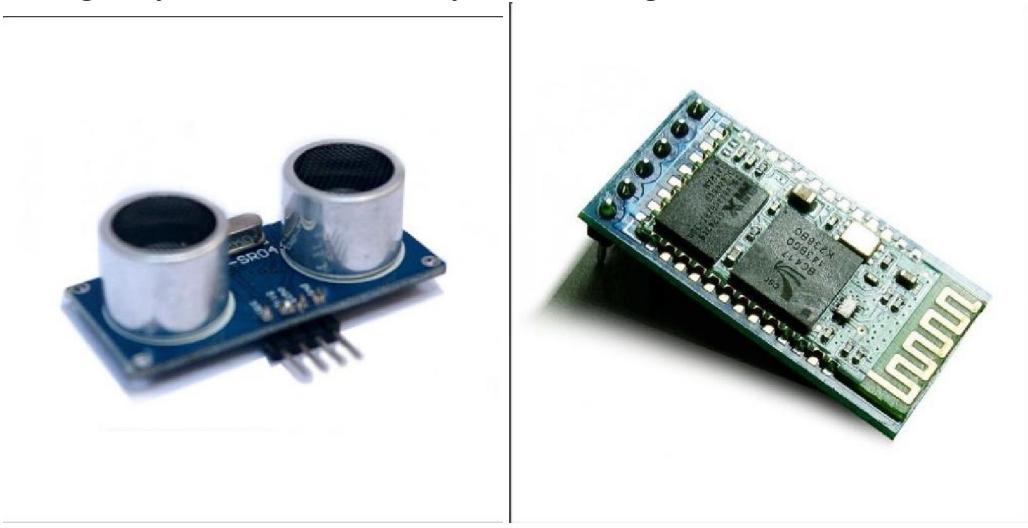
- ❖ Mạch điều khiển, mạch nguồn, mạch driver hoạt động ổn định.
- ❖ Có thể điều khiển động cơ sơ bộ ở khoảng tốc từ 0(vòng/ giây) đến 15 (vòng/giây) với một chiều quay ở 12V.
- ❖ Giao tiếp thành công giữa máy tính và LaunchPad bằng cổng nối tiếp, có thể truyền nhận các thông số cài đặt trên máy tính xuống Board, và Board gửi dữ liệu lại lên máy tính để so sánh vận tốc đặt.
- ❖ Vẫn chưa xử lý được Floating (dấu chấm động) do MSP430G2553 không hỗ trợ nên việc thời gian đáp ứng với sai số xác lập khó đối với động cơ DC có tốc độ quay không lớn.
- ❖ Hiệu chỉnh hệ số PID rất khó để đạt được vận tốc mong với thời gian đáp ứng nhanh và sai số ít

PHẦN MỞ RỘNG:

Việc điều khiển được động cơ theo ý muốn giúp ta phát triển hơn trong lĩnh vực điều khiển, tăng độ chính xác cao. Phát triển ứng dụng khi thêm mô hình xe:



Cùng với **cảm biến siêu âm** có thể phát triển lên trong việc Robot bám tường, hay thêm dò line chạy theo đường chỉ dẫn



Cùng với **Module Bluetooth** ta có thể điều khiển động cơ bằng tay giao tiếp qua Bluetooth điện thoại với ngôn ngữ Adroid.

TÀI LIỆU THAM KHẢO

- <http://www.ti.com/>
- <http://www.wikipedia.org/>
- <http://www.payitforward.edu.vn>
- <http://www.diendanti.com/forum.php>
- <http://www.alldatasheet.com/>
- <http://www.hocavr.com/index.php/app/dcservo>
- <https://www.google.com.vn/>