Xilinx Zynq FPGA, TI DSP, MCU 기반의 회로 설계 및 임베디드 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – Hyungjoo Kim(김형주)
mihaelkel@naver.com

엔코더 분석

- 엔코더 스펙

엔코더부 사양 ENCODER SPECIFICATION

항목	사양	항목	사양
전원	5V ± 5% 0.075 A 이하	실용 회전수	0 ~ 2,000 RPM
부호	INCREMENTEL	선재	자기 소화성을 가질 것
분해능	432 P/R	비금속부품 재질	UL94V 이상의 것을 사용할 것
출력 파형	단파형	온도	작동시 0 ~ +60 ℃
출력 전압	L=0.4V / 10mA 이하 H=4V 이상	습도	작동시 0 ~ 85 %
기동 시간	2 usec 이하	진동	작동시 0.25 G
응답주파수	36 KHz 이하		

분해능(PR)= $\frac{432}{2\pi} pulse/rad$

1/분해능 = $\frac{2\pi}{432} rad/pulse$

주기 = $T_{encoder} \frac{sec}{pulse}$

모터 각속도 $\omega = \frac{1}{T_{encoder} * PR}$

$= \frac{2\pi}{T_{encoder} * 432}$

- Ex)

엔코더 pulse 주기 : 0.639ms

모터 각속도 $\omega = \frac{2\pi}{0.639 * 10^{-3} * 432}$

$= 22.7612\ rad/sec$

엔코더 분석

- 엔코더->모터속도 변환 코드

$$\omega = \frac{2\pi}{T_{encoder} * 432}$$

```
180 void set_omega()
181 {
182     /*unexpected over values ocurr in case the sampling time is too much shot or long*/
183     if(2*M_PI/(period*432) * 1000000 > MAX_OMEGA)
184         goto err;
185 
186     /*encoder's resolution : 432 P/R, period has ns, but omega should be second, so multiple 10^9*/
187     omega = 2*M_PI/(period*432) * 1000000;  //period : ns
188     omega *= 1000;                          // omega : rad/s
189 
190 err:
191 }
```

모터 속도 제어

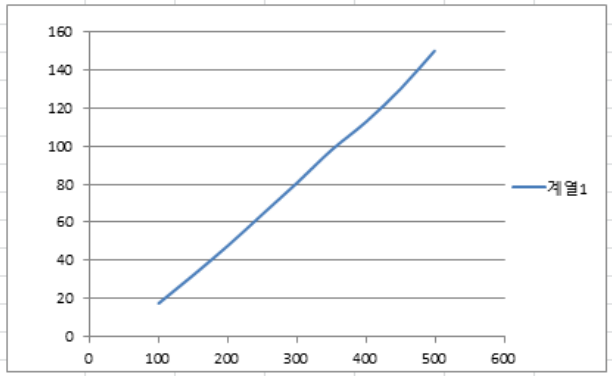
- 모터 드라이버를 통해 속도제어
- Pulse <->속도 관계 : 실험값 사용
- $\omega = 17.46073 + (duty\ - 100) * 0.331994$

  $duty = 100 + \dfrac{omega\ - 0.1746073}{0.331994}$
- 선형성이 어느정도 보장되었기 때문에, Linear approximation을 사용하지 않고 수식 하나로 해결.
- 코드

```
165 float64 get_omeaga(int duty)
166 {
167     /*w = 17.46073 + (duty-100)*0.331994*/
168     float64 res = 17.46073;
169     res += 0.331994*(duty-100);
170     return res;
171 }
172 int get_duty(float64 omega)
173 {
174     /*duty = (omega - 17.46073)/0.331994 + 100*/
175     int res = 100;
176     res += (int)((omega - 17.46073)/0.331994);
177     return res;
178 }
```

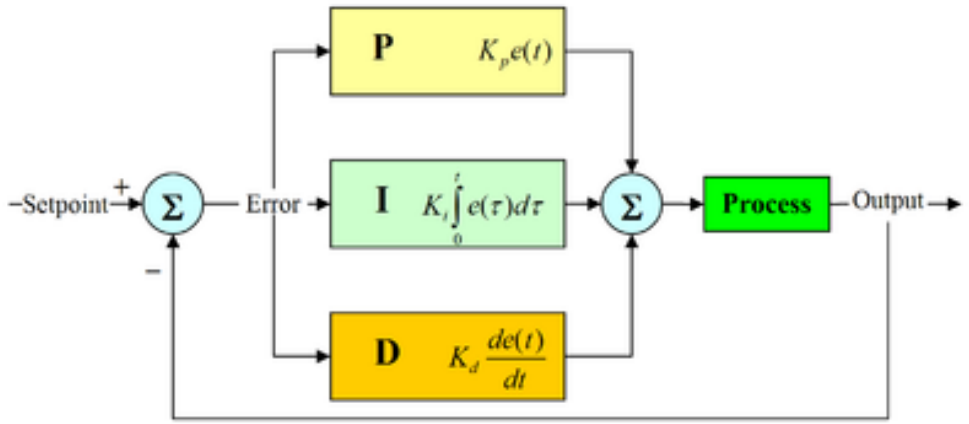
O1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	150.459	130.748	112.969	97.187	80.558	63.937	31.196	16.337	46.132		주기	듀티	499	149.9264			
2	150.855	130.248	113.145	97.613	80.226	63.52	30.552	16.561	46.553		500		450	130.099			
3	149.449	130.326	112.457	97.413	80.748	64.208	32.167	17.051	46.04				400	112.7399			
4	149.286	129.784	112.55	97.413	80.713	64.542	32.254	16.5	46.597				350	97.94244			
5	150.397	130.078	112.006	97.335	80.784	64.638	31.76	15.759	46.367				300	80.68561			
6	150.397	130.28	112.887	97.422	80.85	64.772	32.034	15.778	46.369				250	64.1932			
7	150.294	130.576	113.028	97.213	81.515	64.577	31.08	17.039	46.849				200	47.69378			
8	148.98	130.404	112.969	97.239	81.417	64.356	31.553	18.082	47.075				150	32.30334			
9	150.439	129.215	112.526	97.789	80.952	63.888	32.567	18.222	47.31				100	17.46073			
10	150.128	130.435	112.018	97.517	81	63.244	31.712	17.982	46.095								
11	149.245	130.67	112.422	97.317	80.629	64.261	32.07	17.287	46.472				기울기	0.331994			
12	148.919	129.969	113.133	97.326	80.445	64.212	31.607	17.621	45.768				초기값	100	17.46073		
13	149.942	129.599	113.274	97.239	80.096	63.694	31.362	17.791	46.347				수식	w = 17.46073 + (duty-100)*0.331994			
14	150.087	130.031	112.272	97.032	80.463	63.769	32.342	17.404	45.706				비교	100	17.46073		
15	149.634	130.717	112.724	97.3	80.874	64.284	31.906	16.815	46.974					150	34.06043		
16	148.736	130.373	111.903	97.824	81.339	65.225	32.003	16.661	46.731					200	50.66013		
17	149.819	130.326	113.262	97.448	80.665	64.954	32.106	17.075	47.122					250	67.25983		
18	150.356	129.506	112.864	97.309	80.516	63.724	31.247	17.987	46.849					300	83.85953		
19	149.819	129.876	113.004	97.239	80.244	64.065	31.904	18.287	47.004					350	100.4592		
20	149.021	130.685	112.295	97.771	80.748	64.306	31.908	17.535	47.265					400	117.0589		
21	150.169	130.968	112.272	97.066	80.671	63.817	31.224	17.451	46.583					450	133.6586		
22	150.563	129.522	112.029	97.43	80.802	63.78	31.871	17.715	47.883					499	149.9263		
23	149.984	129.215	113.298	97.727	80.91	63.933	30.96	17.842	47.228								
24	149.286	130.389	112.724	97.701	81.625	64.001	31.248	17.614	47.284								
25	150.149	130.701	112.654	97.161	81.527	63.941	32.166	16.601	46.547								
26	150.834	129.892	111.743	96.954	81.266	63.817	31.596	17.502	46.837								
27	149.963	129.768	112.133	96.765	81.236	65.027	32.161	18.805	46.681								
28	149.102	129.954	112.958	97.023	80.892	64.919	31.863	18.653	46.669								
29	150.439	130.389	113.028	97.283	80.802	63.806	30.913	18.748	47.314								
30	150.542	130.404	112.689	97.797	80.451	63.565	32.21	17.548	47.372								
31	150.169	130.342	111.949	97.771	80.635	63.87	32.481	17.516	47.624								
32	149.163	129.429	112.122	97.257	80.904	64.276	32.063	18.155	46.559								
33	150.314	129.583	112.736	96.808	81.308	63.754	32.017	17.543	47.257								
34	150.73	130.701	113.298	96.731	80.778	63.657	30.851	17.267	46.897								
35	149.901	131.015	112.981	96.868	80.474	64.182	31.917	16.408	47.202								
36	149.265	129.923	112.887	97.43	80.084	64.432	32.389	16.967	47.049								



PID 제어기

- PID Block diagram



K_p: 응답속도 제어
K_i : 정상상태 오차 제어
K_d : 안정성 제어

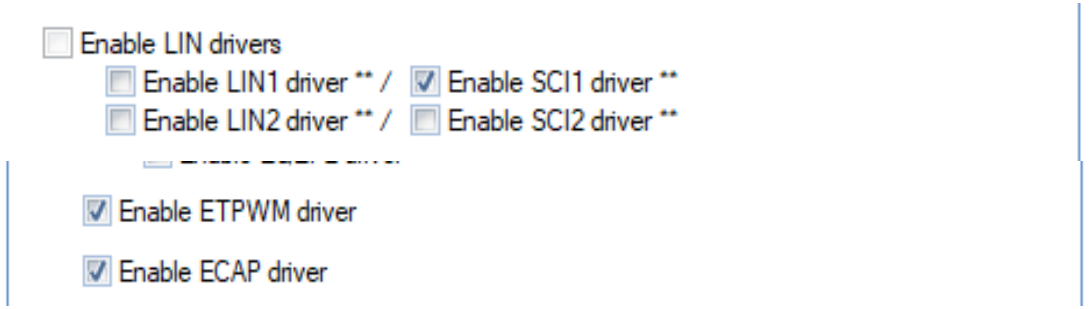
- $u(t) = \omega_{res} - \omega_{err} K_p + \int \omega_{err} * K_i + \frac{d}{dt} \omega_{err}$
- $\omega_{err} = \omega_{res} - \omega_{goal}$
- 코드 적용

```
105         /*pid controler*/
106         u = omega - Kp*err_omega + Ki*sum_err + Kd*diff_err;
```

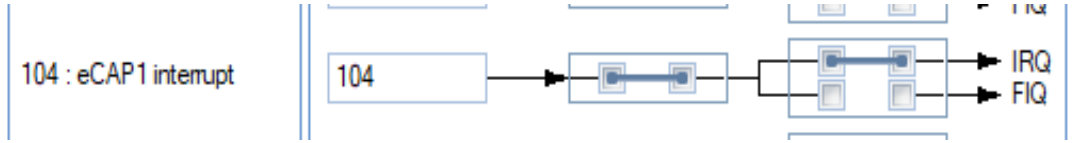
```
137     /*error detection, */
138     err_omega = omega - goal_omega;
139     cur_err = err_omega;
140
141     /*integral err_omega*/
142     /*period has ns unit. so, divide 10^9*/
143     sum_err -= cur_err*period/1000000000;
144
145     /*differential err_omega*/
146     diff_err = (prev_err - cur_err);
147
148     /*update previous error*/
149     prev_err = cur_err;
```

HalCoGen 설정

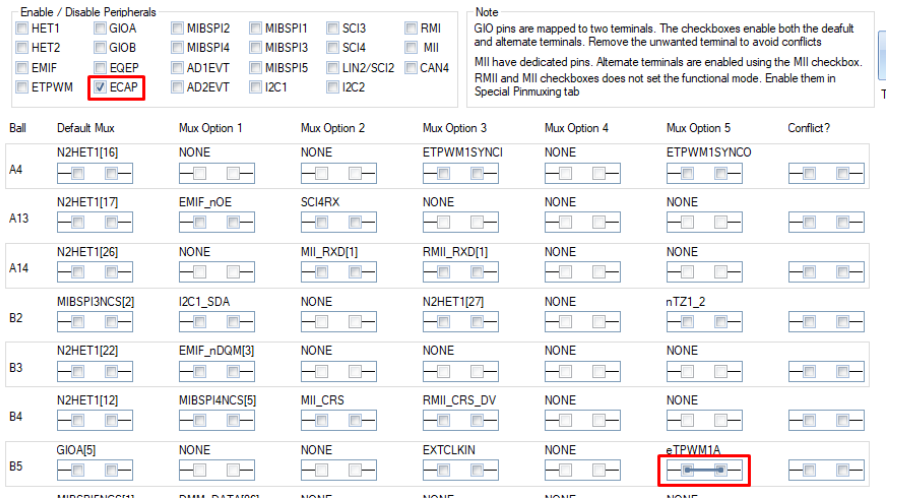
- Driver Enable



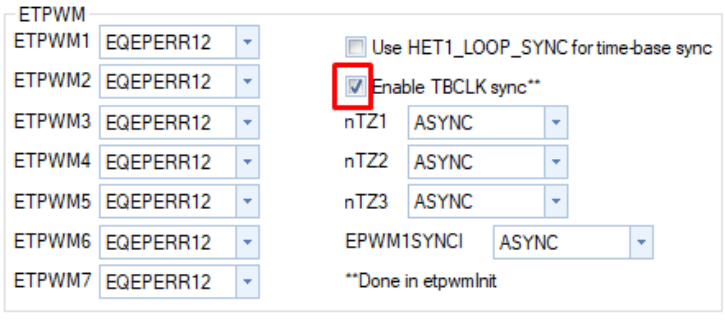
- Vim channel



- PINMUX

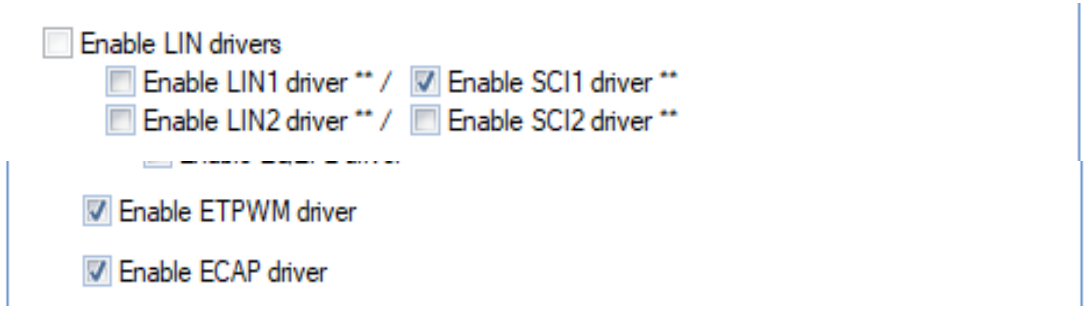


- Special Pin Muxing

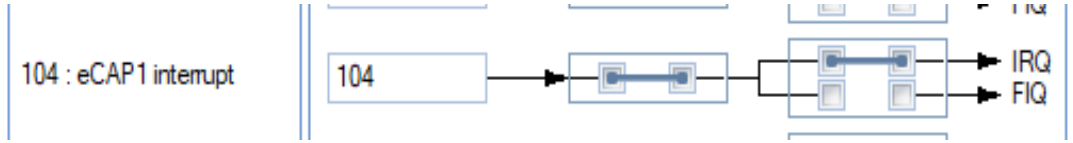


HalCoGen 설정

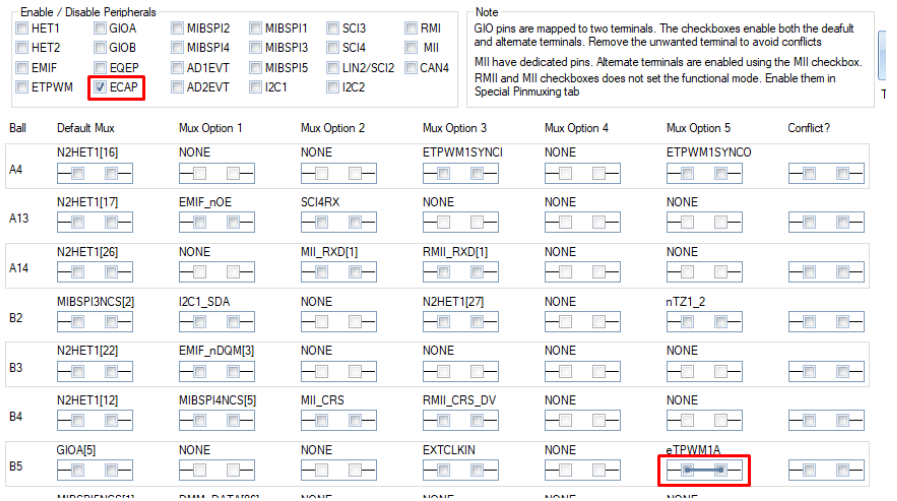
- Driver Enable



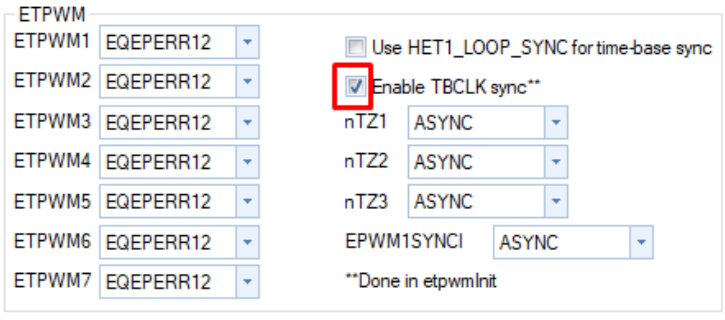
- Vim channel



- PINMUX

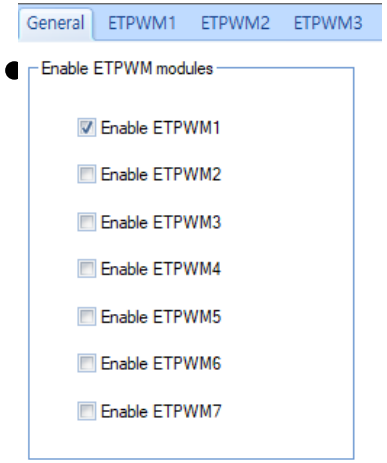


- Special Pin Muxing

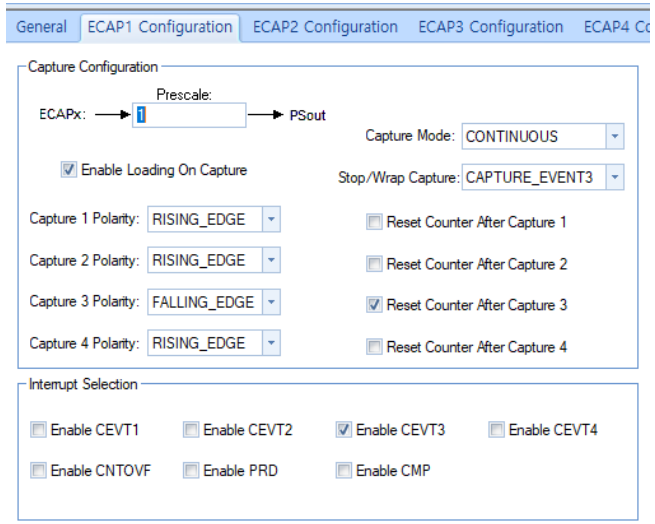


HalCoGen 설정

- etPWM



eCAP1



CODE

Getting Started | HL_sys_main.c | HL_sys_main.c | HL_etpwm.c | HL_etpwm.h | HL_rti.h | HL_rti.c

```c
#include <HL_ecap.h>
#include <HL_etpwm.h>
#include <HL_hal_stdtypes.h>
#include <HL_reg_ecap.h>
#include <HL_reg_etpwm.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <HL_sys_core.h>
#include <HL_system.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define MAX_OMEGA   1000

void send_data(sciBASE_t* sci, uint8* msg, int length);
void set_period(int period, int duty);
float64 get_omeaga(int duty);
int get_duty(float64 omega);
void set_omega();

float64 omega = 0.0, goal_omega = 30.0;
float64 duty = 0.0, period = 0.0, prev_duty = 0.0;
float64 sum_err = 0.0, diff_err = 0.0;
float64 err_omega = 0.0, u = 0.0;
volatile float prev_err = 0.0f;
volatile float cur_err = 0.0f;
float64 prev_omega = 0.0, cur_omega = 0.0;
int goal_period = 500, goal_duty = 100;

void main(void)
{
/* USER CODE BEGIN (3) */

    uint8 tx_msg[64] = {0,};

    float64 Kp = 1.3, Kd = 0.05, Ki = 0.1;  //Ki = 0.4 Kd = 0.02

    _enable_interrupt_();

    /* Initialise EPWM and ECAP with GUI configuration */
    etpwmInit();
    ecapInit();
    sciInit();
    /* Alternate code for configuring ETPWM and ECAP */
        /* Configure ETPWM1 */
    /* Set the TBCLK frequency =  VCLK3 frequency = 75MHz */
    etpwmSetClkDiv(etpwmREG1, ClkDiv_by_1, HspClkDiv_by_1);

    /* Set the time period as 1000 ns (Divider value = (1000ns * 75MHz) - 1 = 74)*/
    etpwmSetTimebasePeriod(etpwmREG1, 74);

    /* Configure Compare A value as half the time period */
    etpwmSetCmpA(etpwmREG1, 37);

    /* Configure mthe module to set PWMA value as 1 when CTR=0 and as 0 when CTR=CmpA  */
    etpwmActionQualConfig_t configPWMA;
    configPWMA.CtrEqZero_Action = ActionQual_Set;
    configPWMA.CtrEqCmpAUp_Action = ActionQual_Clear;
    configPWMA.CtrEqPeriod_Action = ActionQual_Disabled;
    configPWMA.CtrEqCmpADown_Action = ActionQual_Disabled;
    configPWMA.CtrEqCmpBUp_Action = ActionQual_Disabled;
    configPWMA.CtrEqCmpBDown_Action = ActionQual_Disabled;
    etpwmSetActionQualPwmA(etpwmREG1, configPWMA);

    /* Start counter in CountUp mode */
    etpwmSetCount(etpwmREG1, 0);
    etpwmSetCounterMode(etpwmREG1, CounterMode_Up);
    etpwmStartTBCLK();

    /* Configure ECAP1 */
    /* Configure Event 1 to Capture the rising edge */
    ecapSetCaptureEvent1(ecapREG1, RISING_EDGE, RESET_DISABLE);

    /* Configure Event 2 to Capture the falling edge */
    ecapSetCaptureEvent2(ecapREG1, FALLING_EDGE, RESET_DISABLE);
```

CODE

```
81      /* Configure Event 3 to Capture the rising edge with reset counter enable */
82      ecapSetCaptureEvent3(ecapREG1, RISING_EDGE, RESET_ENABLE);
83
84      /* Set Capure mode as Continuous and Wrap event as CAP3  */
85      ecapSetCaptureMode(ecapREG1, CONTINUOUS, CAPTURE_EVENT3);
86
87      /* Start counter */
88      ecapStartCounter(ecapREG1);
89
90      /* Enable Loading on Capture */
91      ecapEnableCapture(ecapREG1);
92
93      /* Enable Interrupt for CAP3 event */
94      ecapEnableInterrupt(ecapREG1, ecapInt_CEVT3);
95
96
97      /*  ... run forever  */
98      while(1)
99      {
100         /*pid controler*/
101         u = omega - Kp*err_omega + Ki*sum_err + Kd*diff_err;
102
103         /*conversion from omega to duty cycle, to set the PWM signal's duty cycle*/
104         goal_duty = get_duty(u);
105         set_period(goal_period, goal_duty);
106
107         /*send the data to SCI*/
108         sprintf(tx_msg,"omega = %.3f, duty = %d%, err = %.3f, u = %.3f, int = %.3f, diff = %.10f\r\n",
109                 omega, goal_duty, err_omega ,u, sum_err, diff_err); // rad/s
110         send_data(sciREG1, tx_msg, strlen(tx_msg));
111     }
112
113 }
114
```

```
114
115 void ecapNotification(ecapBASE_t *ecap,uint16 flags)
116 {
117     uint32 cap1, cap2, cap3;
118
119     cap1 = ecapGetCAP1(ecapREG1);
120     cap2 = ecapGetCAP2(ecapREG1);
121     cap3 = ecapGetCAP3(ecapREG1);
122     duty = (cap2 - cap1)*1000/VCLK3_FREQ;
123     period = (cap3 - cap1)*1000/VCLK3_FREQ;
124
125
126     /*calc with Encoder's specification*/
127     set_omega();
128
129     /*unexpected over values ocurr in case the sampling time is too much shot or long*/
130     if(omega > MAX_OMEGA)
131         goto err;
132
133     /*error detection, */
134     err_omega = omega - goal_omega;
135     cur_err = err_omega;
136
137     /*integral err_omega*/
138     /*period has ns unit. so, divide 10^9*/
139     sum_err -= cur_err*period/1000000000;
140
141     /*differential err_omega*/
142     diff_err = (prev_err - cur_err);
143
144     /*update previous error*/
145     prev_err = cur_err;
146 err:
147 }
```

CODE

```
148
149 void send_data(sciBASE_t* sci, uint8* msg, int length)
150 {
151     int i;
152     for(i=0;i<length;i++)
153         sciSendByte(sci, msg[i]);
154 }
155
156 void set_period(int period,int duty)
157 {
158     etpwmREG1->TBPRD = period;
159     etpwmREG1->CMPA = duty;
160 }
161 float64 get_omeaga(int duty)
162 {
163     /*w = 17.46073 + (duty-100)*0.331994*/
164     float64 res = 17.46073;
165     res += 0.331994*(duty-100);
166     return res;
167 }
168 int get_duty(float64 omega)
169 {
170     /*duty = (omega - 17.46073)/0.331994 + 100*/
171     int res = 100;
172     res += (int)((omega - 17.46073)/0.331994);
173     return res;
174 }
175 void set_omega()
176 {
177     /*unexpected over values ocurr in case the sampling time is too much shot or long*/
178     if(2*M_PI/(period*432) * 1000000 > MAX_OMEGA)
179         goto err;
180
181     /*encoder's resolution : 432 P/R, period has ns, but omega should be second, so divide 10^9*/
182     omega = 2*M_PI/(period*432) * 1000000;  //period : ns
183     omega *= 1000;                          // omega : rad/s
184
185 err:
186 }
```