Introduction to Digital Control Systems

- Learned about systems that collect information from the environment.
- Now, we use this information to control the environment.
- A control system is a collection of mechanical and electrical devices connected to regulate a physical plant.
- Many embedded applications are control systems.
- Control theory is a richly developed discipline covered by other courses, so we emphasize implementation issues.
Traffic Light Control System
(See Figures 13.2 and 13.3)

Traffic Light Controller (cont)
void main(void){ StatePtr *Pt; /* Current State */
Pt=NorthRed_EastGreen; /* Initial State */
DDRB=0xFF; /* Make Port B outputs */
while(1){
    PORTB=Pt->Out; /* Perform output */
    Wait(Pt->Time); /* Time to wait in this state */
    Pt=Pt->Next; /* Move to next state */
}
}

Traffic Light Controller
const struct State {
    unsigned char Out; /* Output to Port B */
    unsigned short Time; /* Time in sec to wait */
    const struct State *Next; /* Next state */
typedef const struct State StateType;
#define NorthRed_EastGreen &fsm[0]
#define NorthRed_EastYellow &fsm[1]
#define NorthGreen_EastRed &fsm[2]
#define NorthYellow_EastRed &fsm[3]
StateType fsm[4] = {
    {0x21, 180, NorthRed_EastYellow},
    {0x22, 15, NorthGreen_EastRed},
    {0x0C, 180, NorthYellow_EastRed},
    {0x14, 15, NorthRed_EastGreen}};

Open-Loop Stepper Controller
(See Figure 13.6)
Circular List for Stepper Motor

```c
const struct State {
    unsigned char Out; /* Output to Port B */
    const struct State *Next; /* Next state */
} StateType;
#define S6 &fsm[0]
#define S5 &fsm[1]
#define S9 &fsm[2]
#define S10 &fsm[3]
StateType fsm[4]=
{0x06, S5}, {0x05, S9},
{0x09, S10}, {0x0A, S6};
StatePtr *Pt; /* Current State */
unsigned short Speed;
```

Spin Stepper Motor at Constant Speed

```c
#define OC5 0x08
#pragma interrupt_handler TOC5handler()
void TOC5handler(void) {
    PORTB=Pt->Out; // output for this state
    Pt=Pt->Next;  // Move to next state
    TFLG1=OC5;   // Ack OC5F
    TOC5=TOC5+Speed; // Executed every step
    void ritual(void) {
        asm("sei"); // make atomic
        TMSK1|=OC5; // Arm output compare 5
        TFLG1=OC5;  // Initially clear OC5F
        Speed=10000; // initial speed
        Pt=S6;      // initial state
        TOC5=TCNT+2000; // First one in 1 ms
        asm("cli");
    }
}
```

Bang-Bang Temperature Controller

(See Figure 13.7 and 13.8)

Bang-Bang Temperature Controller

(See Figure 13.9)
**Bang-Bang Temperature Control Software**

```c
unsigned char Tlow,Thigh,T; int E; // units in C
void Actuator(unsigned char relay){
    PORTB=relay; // turns power on/off
    #pragma interrupt_handler TOC5handler()
    void TO5handler(void){
        T=SE(A2D(channel)); // estimated T
        E=Tstar-T; // error
        if(T<Tlow)
            Actuator(0); // too cold so off
        else if (T>Thigh)
            Actuator(1); // too hot so on
        else
            // leave as is if Tlow<T<Thigh
            TOC5=TOC5+rate; // periodic rate
            TFLG1=0x08; } // ack OC5F
    }
}
```

---

**Incremental Position Control System**

```c
unsigned char Xstar,X; int E; // in mm
#pragma interrupt_handler TOC5handler()
void TO5handler(void){ int new;
    X=SE(A2D(channel)); // estimated (mm)
    E=Xstar-X; // error (mm)
    new=PORTB; // promote to 16 bits
    if(E<-1) new--; // decrease
    else if (E>1) new++; // increase
    // leave as is if -1<E<1
    if(new<0) new=0; // underflow
    if(new>255) new=255; // overflow
    PORTB=new; // output to actuator
    TOC5=TOC5+rate; // establish periodic
    TFLG1=0x08; } // ack OC5F
```

---

**Position Control System Using Incremental Control**

(See Figures 13.10 and 13.11)

---

**Block Diagram of a Linear Control System**

(See Figure 13.12)
General Approach to a PID Controller

\[
U(t) = K_p E(t) + K_i \int_0^t E(\tau) d\tau + K_d \frac{dE(t)}{dt}
\]

\[
U(n) = P(n) + I(n) + D(n)
\]

\[
P(n) = K_p E(n)
\]

\[
I(n) = I(n-1) + K_I \cdot E(n) \cdot \Delta t
\]

\[
D(n) = K_D \cdot \frac{E(n) - E(n-1)}{\Delta t} \quad \text{Large errors}
\]

\[
D(n) = K_D \cdot \left[ \frac{1}{2} \frac{E(n) - E(n-3)}{3\Delta t} + \frac{1}{2} \frac{E(n-1) - E(n-2)}{\Delta t} \right]
\]

\[
D(n) = K_D \cdot \frac{E(n) + 3E(n-1) - 3E(n-2) - E(n-3)}{6\Delta t}
\]

Effect of Noise on Derivative Calculation

(See Figure 13.13)

Interface of a PID Velocity Controller

(See Figure 13.14)

Velocity PID Controller
Integral Controller with a PWM Actuator

(See Figure 13.15 and Table 13.1)

Integral Position Control Software

unsigned int Time; // Time in msec
unsigned int X; // Est position in cm
unsigned int Xstar; // Desired pos in cm
unsigned int U; // Actuator duty cycle
unsigned int Cnt; // once a sec
unsigned int Told; // used to meas period

Integral Position Control Software (cont)

#pragma interrupt_handler TOC5handler()

void TOC5Handler(void){ int NewU;
TFLG1=0x08; // Ack OC5F
TOC5=TOC5+2000; // every 1 ms
Time++; // used to measure period
if((Cnt++)==1000) Cnt=0; // every 1 sec
// 0<X<100, 0<Xstar<100, 100<U<19900, so
// Min when U=100,Xstar=0,X=100, min NewU = -900
// Max when U=19900,Xstar=100,X=0, max NewU = 20900
// so NewU will be a valid signed int
NewU=U+10*(Xstar-X);
if(NewU<100) NewU=100; // Constrain
if(NewU>19900) NewU=19900;
U=NewU; }

PWM Actuator Control Software

#pragma interrupt_handler TOC4handler()

void TOC4Handler(void){
TFLG1=OC4F; /* Ack */
if(TCTL1&0x04) /* OL4 bit */
TFLG1=OC4F; /* Ack */
else
TFLG1=OC4F; /* Ack */
TCTL1^=0x04; } /* Toggle OL4 */
Sensor Measurement Software

```c
#include <avr/io.h>

void TIC1Handler(void){
    unsigned int p;
    TFLG1 = IC1F; // Ack IC1F
    p = Time-Told; // period in msec
    X = p-10; // estimated position (cm)
    Told=Time;
}
```

Initialization Software

```c
void ritual(void) {
    asm("sei"); /* atomic */
    OC1M=0; OC1D=0;
    TCTL1=0x08; // Clear OC4
    TCTL2=0x10; // capture on rise of IC1
    TMSK1=OC5F+OC4F+IC1F; // Arm
    U=10000; // Initial U, half power
    Time = 0; Told=0; Cnt=0;
    TFLG1=OC5F+OC4F+IC1F; // clear flags
    TOC5=TCNT+2000; // First OC5 in 1 ms
    TOC4=TCNT+100; // First OC4 in 50us
    asm("cli");
}
```

Relative Timing of Interrupts

(See Figure 13.16)

Empirical Method to Determine Parameters

- Start with a proportional term ($k_p$) which will generate a smooth motor speed but speed is not correct.
- Try different $k_p$ values till response time is acceptable.
- Response time is time from change in $X$ till a new constant speed is achieved.
- Next, add $k_I$ term to improve steady state accuracy without adversely affecting response time.
- Steady state accuracy is average diff. b/w $X*$ and $X'$.
- Finally, if overshoot or undershoot are unacceptable, then added $k_D$ term to reduce them.
- Overshoot is maximum positive error as $X*$ is increased.