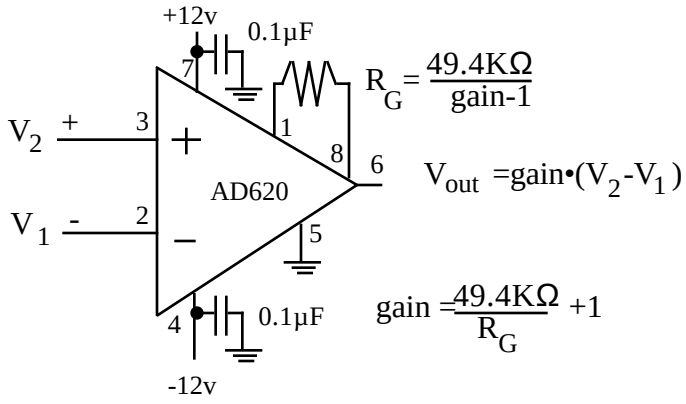
Jonathan W. Valvano December 11, 2002, 9 am to 12 noon

(15) Question 1. The key is to use an instrumentation amp like the AD620. The gain resistor is 49.4K/499 = 99 Ω. This is figure 11.38 on page 615 from the book.



(15) Question 2. Consider 128K by 8 bit static RAM DS1230.

Part a) Read Data Available = (later (AdV+t_{ACC} , ↓ $\overline{CE}$ + t_{CO} , ↓ $\overline{OE}$ + t_{OE}) ,
earlier (AdN+t_{OH} , ↑ $\overline{CE}$, ↑ $\overline{OE}$))

Part b) Write Data Required = (↑ $\overline{CE}$ - t_{DS} , ↑ $\overline{CE}$ + t_{DH2})

(55) Question 3. The objective of this problem is to implement a PI motor controller.

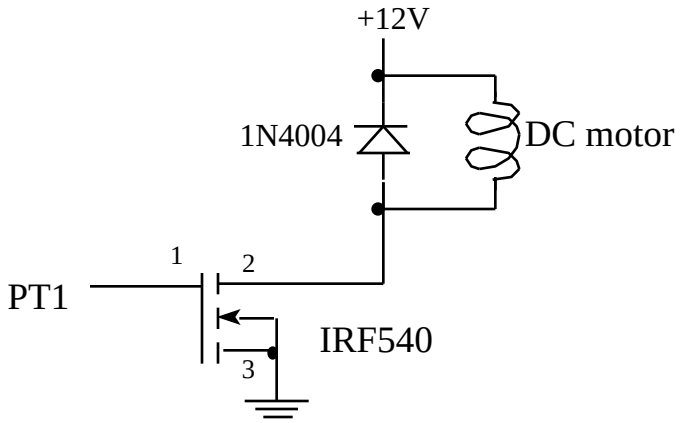
(10) Part a) At 500 rpm, the Period= 200000/500=400µsec
ΔSpeed = 200000/400-200000/401 = 1.25 rpm

(10) Part b) measure period on input PT0, and change the resolution to 1 µsec.

```
unsigned short Period;  // units of 1us
unsigned short First;   // TCNT first edge
unsigned short Count;   // number of TOF interrupts since last IC
void IC_Ritual(void){
  asm(" sei");    // make atomic
  TIOS &= 0xFE;   // PT0 input capture
  DDRT &= 0xFE;   // PT0 is input
  TSCR = 0x80;    // enable TCNT
  TMSK2= 0x33;    // 1us clock
  TCTL4 = (TCTL4&0xFC)|0x01; // rising
  First = TCNT;   // first will be wrong
  TFLG1 = 0x01;   // Clear C0F
  TMSK1 |= 0x01 ;  // Arm IC0
  asm(" cli");}
#pragma interrupt_handler IChandler()
void IChandler(void){
  Period = TC0-First;
  Count = 0;     // signal to TOF all is OK
  First = TC0;   // Setup for next
  TFLG1 = 0x01;  // ack by clearing C0F
  Speed = 200000L/(long)Period;}
#pragma abs_address:0xffee
void (*ICvector[])() = {IChandler};
#pragma end_abs_address
// Program 4.51, page 259
unsigned short Time;
void TOF_Ritual(void){
  asm(" sei");      // Make ritual atomic
  TMSK2 |= 0x80 ;   // Arm, Set PR to 011, 15.2585Hz
  TSCR = 0x80;      // enable counter
  Time = 0;         // Initialize global data structures
  Count = 0;
```

```
  asm(" cli");}
#pragma interrupt_handler TOFHan()
void TOFHan(void){
  TFLG2 = 0x80;       // Acknowledge by clearing TOF
  Count++;
  if(Count>1) Speed=0;}
#pragma abs_address:0xffde
void (*TOFvector[])() = {TOFHan};
#pragma end_abs_address
```

(10) Part c) The DC motor must be interfaced to PT1. Interface needs a snubber diode and a way to sink 0.5 amp of current. Other chips like ULN2074, TIP120 and L293 are OK.



(10) Part d) The output is on PT1 and the basic period is 50 ms (20Hz).

```
unsigned short High; // Num of Cycles High
unsigned short Low;  // Num of Cycles Low
// Period is High+Low Cycles
void OC_Ritual(void){
  asm(" sei");    // make atomic
  TIOS |= 0x02;   // enable OC1
  DDRT |= 0x02;   // PT1 is output
  TSCR = 0x80;    // enable
  TMSK2 = 0x32;   // 500 ns clock
  TMSK1 |= 0x02;  // Arm output compare 1
  TFLG1 = 0x02;   // Initially clear C1F
  TCTL2 |= 0x0C;  // PT1 set on next int
  TC1 = TCNT+50;  // first right away
  asm(" cli"); }
#pragma interrupt_handler OChandler()
void OChandler(void){
  TFLG1 = 0x02;       // ack C1F
  if(TCTL2&0x04){     // PT1 is now high
    Controller();
    TC1 = TC1+High;   // 1 for High cyc
    TCTL2 &= 0xFB;    // clear on next
  }
  else {              // PT1 is now low
    TC1 = TC1+Low;    // 0 for Low cycles
    TCTL2 |= 0x04;    // set on next int
}}
#pragma abs_address:0xffec
void (*OCvector[])() = {OChandler};
#pragma end_abs_address
```

(15) Part e) The digital controller is executed every $\Delta t = 0.05$ s.

```
long Ui; // integral term
void Controller(void){ long e,Up,U;
  e = 500 – Speed;    // in rpm
```

```
  Up = (269*e)/2;     // proportional term      134.5=269/2
  Ui = Ui+(789*e)/2;  // integral term 7890*0.05=394.5=789*/2
  if(Ui<-1000) Ui = -1000; // anti-reset windup
  if(Ui>1000)  Ui = 1000;  // anti-reset windup
  U = Up + Ui;
  if(U<200) U = 200;      // minimum duty cycle
  if(U>49800) U = 49800;  // maximum duty cycle
  High = U;               // demote
  Low = 50000-High;          // maintain High+Low always equal to 50000
}
```

(15) Question 4. The answer is described in the book on page 281

```
#define FifoSize 10    /* Number of 8 bit data in the Fifo */
unsigned char PutI;    /* Index of where to put next */
unsigned char GetI;    /* Index of where to get next */
unsigned char Size;    /* Number of elements currently in the FIFO */
/* FIFO is empty if Size=0     FIFO is full if Size=FifoSize */
char Fifo[FifoSize];      /* The statically allocated fifo data */
void InitFifo(void){char SaveSP;
  asm(" tpa\n staa %SaveSP\n sei");   /* make atomic, entering critical*/
  PutI=GetI=Size=0;                   /* Empty when Size==0 */
  OS_InitSemaphore(&RoomLeft,FifoSize); /* Number of empty places */
  OS_InitSemaphore(&DataAvailable,0);   /* Number of elements stored */
  asm(" ldaa %SaveSP\n tap");         /* end critical section */
}
```

```
int PutFifo(char data){ char SaveSP;
  if (Size == FifoSize )
    return(0);    // Failed, was full
  else{
    asm(" tpa\n staa %SaveSP\n sei");
    Size++;
    OS_Wait(&RoomLeft);
    Fifo[PutI++]=data;  //  put data
    if (PutI == FifoSize) PutI = 0;
    asm(" ldaa %SaveSP\n tap");
    OS_Signal(&DataAvailable);
    return(-1);    /* Successful */
  }
}
```

```
int GetFifo(char *datapt){char SaveSP;
  if (Size == 0 )
    return(0);     // Empty if Size=0
  else{
    asm(" tpa\n staa %SaveSP\n sei");
    OS_Wait(&DataAvailable);
    *datapt=Fifo[GetI++];
    Size--;
    if (GetI == FifoSize) GetI = 0;
    OS_Signal(&RoomLeft);
    asm(" ldaa %SaveSP\n tap");
    return(-1);
  }
}
```

Extra Credit (bonus points)

(1) Bonus 1. Spinlock binary semaphore without disabling interrupts uses `minm`

(1) Bonus 2. The C compiler uses `bclr`

(1) Bonus 3. The defuzzification step in a fuzzy logic controller uses `wav` and `ediv`

(1) Bonus 4. Optimization uses `emacs` and `edivs`

(1) Bonus 5. Linear interpolation uses `tbl or etbl`

(1) Bonus 6. Switch the control back to the foreground is performed by `rti`