RMV ELECTRONICS INC. **Application Note:**	**Application #:** 00003
	Date: Setember 1994
Description: Using the ITC232-A for frequency measurement.	**Status:** Draft version

The basic operating principle of this frequency counter is to generate a precisely timed window in which to gate incoming pulses. A counter then keeps track of the number of pulses received during the gate time. When the window has timed out and the gate has closed, the counter is read, and the frequency can be calculated. For example, if a window of 100 milliseconds is generated and the counter subsequently reads 832, the frequency can be calculated by:

Pulse Counter Value / Window Time (in seconds) = Pulses per Second (Hz)

In our example, the result is 832 / 0.1 seconds = 8320 Hz

The longer the window (or gate) is open, the more counts are received and the higher the effective resolution of the frequency counter is. The limitations are the size of the counter and the desired result update rate. If the gate is open for too long a period of time, the counter will overflow and the results will be incorrect. The gate timing must also relate to the incoming frequency. A high frequency will need only a short gate time to produce high enough counts for good resolution. However, this same short gate time will be too short for good resolution as the frequency drops. At lower frequencies, the gate time required for high resolution may get quite long (on the order of one second or longer). The rate at which the final result is calculated can be no shorter that the gate and the results can be slow in coming.

The ideal situation would be to monitor the count value and modify the gate time to produce the optimum results. This application note describes a circuit suitable for the task.

OPERATIONAL DESCRIPTION

Please refer to the schematic diagram and the functional software listing. Also refer to the appropriate manufacturers' data sheets and the ITC232-A user's manual for further detail.

The circuit is implemented using timer/counter combinations. In this example we have used the Intel (or equivalent) 82C53 Programmable Interval Timer (PIT), the LSI Computer Systems LS7166 MultiMode counter (MMC), and a crystal based oscillator module. The 82C53 contains three independent 16 bit counters, each capable of running in one of several modes. The LS7166 is similar in nature and contains a single counter. The LS7166 was used because of its enhanced operation and 24 bit counter. If a reduced counter size (and therefore a reduced resolution) and slower operation is adequate, the LS7166 can be removed and the third counter within the 82C53 can be used instead. This would reduce the chip count and lessen the circuit cost.

The interface between these devices and the ITC232-A is through all of Port C, 6 bits of Port B, and both interrupt lines. Port C is used as a bidirectional data bus, while the Port B lines act as address and bus control lines. A sequence of port writes and reads effectively emulates the operation of a microprocessor data bus, which both the PIT and MMC were designed for.

Counter 0 in the PIT is used as a programmable rate generator, dividing the frequency of the oscillator module by the value programmed into the counter, i.e. there will be one output pulse produced every time the counter reaches the preset number. The output of this counter feeds PIT counter 1 which is set up as programmable one shot. This mode will produce a low output signal for the number of pulses programmed. The combination of these counters produces a gate signal of fixed duration under user control. This signal, in turn, drives the MMC counter gate (ABGT) pin. The MMC counter will increment only while the ABGT signal line is low. The input to the MMC counter is the input pulse stream that the frequency determination is to be made on.

The entire circuit is controlled by a single signal from an ITC232-A port pin and monitored by the interrupts.

From an initial low state, the control line (Port B bit 7) is brought high to enable the PIT one shot. The timed gate is started on the next PIT 0 output pulse and the counter begins to count the input pulse stream. After the gate has timed out, the counter is frozen and the ITC232-A interrupt (IRQH) is asserted. The enable line is returned to a low state and the MMC counter is read, then reset in readiness for the next cycle. The carry output of the MMC is connected to the ITC232-A interrupt (IRQL) to allow for the immediate detection of a counter overflow. This condition would occur when the input frequency is too high for the time the gate is open. The user could modify the value in either of the PIT counters to reduce the gate time. Conversely, if the value read from the counter value is low, the gate time can be increased.

As described above, the final value would then be calculated by dividing the number of counts by the number of seconds the gate is on.

When using the LS7166 as the counter, the input frequency is limited to 20 Mhz. Replacing the LS7166 with the 82C53 will reduce the upper frequency to 8 Mhz. If higher frequency operation is desired, a prescaler can be inserted before the counter and the results calculations modified accordingly. For example, an ECL divide by 10 prescaler would allow it to operate at 200 Mhz.

The effective resolution of the circuit will depend on the gate time used. If a full 1 second gate is chosen, the resolution will be to 1 Hz, but the update time will be also be at most 1 second. Reducing the gate time to 100 msec (1/10 sec) cuts the resolution to 10 Hz (a factor of 10) but increases the update rate by the same factor.

Circuit accuracy is based primarily on the accuracy of the crystal oscillator, which is usually quite good. Other sources of error are the clock input to gate output delays in PIT 0 and the ABGT response time in the MMC. All these delays are in the tens of nanosecond range and should contribute no significant error (on the order of 1 or 2 counts at 10 Mhz, an error of 1 or 2 ppm with a 100 msec. gate).

```
/*
Functional code listing for the frequency counter application
circuit.

ITC232 port assignments are as follows (refer to schematic
diagram)
Port C is a bidirectional 8 bit data bus
Port B, bits 0 and 1 are address lines used to select particular
registers
on the PIT and MMC
Port B, bits 2 and 3 are the active low read and write lines
respectively
Port B, bit 4 is the active low chip select for the PIT
Port B, bit 5 is the active low chip select for the MMC
Port B, bit 6 is an input for monitoring the frequency counter
status
Port B, bit 7 is an output for controlling the frequency counter
operation
IRQH is used to indicate the count cycle has completed
IRQL is used to indicate an overflow condition
/*

/* Constant definitions */
#define OSC_FREQ        8000000
#define PIT_CTR_0       0xec
#define PIT_CTR_1       0xed
#define PIT_CTR_2       0xee
#define PIT_CONTROL     0xef
#define MMC_DATA        0xdc
#define MMC_CONTROL     0xdd
#define BAUD_9600       0x0c
#define COMM_1          0x01
#define COMM_2          0x02
#define COMM_3          0x03
#define COMM_4          0x04
#define NULL            0x00

/*function declarations*/
/* function for sending data thru the ITC232 to peripherals */
void    out_data(unsigned char a, unsigned char b);
/* function for receiving data from ITC232 peripherals */
char    in_data(unsigned char a);
long    count_total;
float   result, gate_time;
int     rate_gen, one_shot;

/*Initialize the PC serial port as for 9600 baud, no parity, 8 bits,
void    ITC_send(char *ptr); /* function to send a string to
ITC232 */
void    ITC_send_no_wait(char *ptr); /* function to send a string
to ITC232 */
char    ITC_data(void); /*function that returns the data value in
response
                                to a port read operation */

/* external function for setting up serial ports */
extern void far Setup(unsigned int Rate, unsigned char Parms,
unsigned char SelBase);
/* Setup(Divisor, Parms, SelBase) */
/* Divisor = 115200/desired baud rate */
/* Parms   = 0 | a | bcd | e | fg     */
/*           a   = 0 for break */
/*               = 1 for space */
/*           bcd = 000 none parity */
/*               = 001 odd parity  */
/*               = 011 even parity */
/*           e   = 0 for 1 stop bit */
/*               = 1 for 2 stop bits */
/*           fg  = 10 for 7 bits per character */
/*               = 11 for 8 bits per character */
/* SelBase = 1 for COM1 */
/*           2 for COM2 */
/*           3 for COM3 */
/*           4 for COM4 */

/* external functions for installing an comm interrupt, removing
same,
   and restoring previous port operation */
extern void far Restore();
extern void far SetInt();
extern void far RemInt();
extern unsigned int far Receive( unsigned char Rfl, unsigned
char Rvalue);
extern unsigned int far SendChar ( unsigned char Sfl, unsigned
char Tvalue);

main()
{
char    count_hi,count_mid,count_lo, input;
1 stop*/
Setup(BAUD_9600, 3, COMM_1);

/*Initialize the ITC232 for Configure Results Ascii Program*/
ITC_send("CRAP\r");
```

```
/*Set Port B outputs bit 6 thru 0 high*/
ITC_send("PWB$7F\r");

/*Set Port B direction as bits 7 and 5 thru 0 as outputs, bit 6 as
input*/
ITC_send("PCB$BF\r");

/*Set PIT counter 0 mode to 2 (rate generator), two byte xfer,
binary counting*/
out_data(PIT_CONTROL,0x34);
/*Set PIT counter 1 mode to 1 (one shot), two byte xfer, binary
counting*/
out_data(PIT_CONTROL,0x72);
/*Setup PIT counter 0 divide rate to 80 and PIT counter 1 to
10,000.
  this example assumes a 8 Mhz oscillator for a 100 msec gate.
*/
rate_gen = 80; /* 0x0050 */
one_shot = 10000; /* 0x2710 */
out_data(PIT_CTR_0,0x50); /*counter 0 low byte*/
out_data(PIT_CTR_0,0x00); /*counter 0 hi byte*/
out_data(PIT_CTR_1,0x10); /*counter 1 low byte*/
out_data(PIT_CTR_1,0x27); /*counter 1 high byte*/

/*Reset MMC, then setup for counting up, normal mode, carry
enabled,
count gate enabled, binary count */
/* send master reset */
out_data(MMC_CONTROL, 0x20);
/* set input control register */
out_data(MMC_CONTROL, 0x78);
/* set output control register */
out_data(MMC_CONTROL, 0x80);
/* set quadrature control register */
out_data(MMC_CONTROL, 0xc0);

/* main loop is here, the process is stopped by hitting any key */
while( !kbhit() )
{
/* enable the PIT one shot to generate the timed gate and start
counting */
ITC_send("PWA$FF\r");
/* now wait for either an IRQL or IRQH interrupt from the
ITC232 */
do
   {
   input = Receive(0,0);
   }
   while(input == 0);  /* if nothing received, just wait */
if((char)input  == 'H')
   {
   /* cycle has ended normally, disable gate and get counts */
   /* first disable PIT 1 gate */
   ITC_send("PWB$7F\r");
   /* request a transfer of the counts to the output latches */
   out_data(MMC_CONTROL, 0x03);
   /* start getting count values */
   count_lo = in_data(MMC_DATA);
   count_mid = in_data(MMC_DATA);
   count_hi = in_data(MMC_DATA);
   /* reset MMC counters for the next cycle */
   out_data(MMC_CONTROL, 0x04);
   /* combine the three byte wide counter reads into a single
value */
            count_total = (count_hi << 16) + (count_mid << 8) +
count_lo;
   gate_time = 1 / OSC_FREQ / rate_gen / one_shot;
            result = count_total / gate_time;
   printf("\nInput frequency = %6.1f Hz.", result);
   printf("\nGate time = %2.6f seconds.\n", gate_time);
   /* code can be entered here to evaluate the results and see if
the
   count value is too low. If so, the gate time can be increased
*/
   }
else
   {
   /* cycle has caused a counter overflow, first wait for gate to
   complete, disable the gate, then print message */


/* then return to idle state */
ITC_send("PWB$7F\r");
}

/***************************************/
/*   in_data                           */
/*  sets up 'bus' for a read operation */
   do
      {
      input = Receive(0,0);
      }
   while(input == 0);  /* if nothing received, just wait */
   /* disable PIT 1 gate */
   ITC_send("PWB$7F\r");
   printf("\nOverflow\n");
   /* code can be entered here to reduce the gate time */
   }

} /*end of while(kbhit... loop */
printf("\nFrequency Counter function halted\n");

} /*end of main*/

/***************************************/
/*   out_data                          */
/* sets up 'bus' for a write operation */
/***************************************/
void out_data(a,b)
unsigned char a,b;
{
char string[8], value[3];

/* set string ends to NULLS */
string[7] = value[2] = NULL;

/* set port C to output */
ITC_send("PCC$FF\r");

/*setup data value on Port C*/
strcpy(string,"PWC$");
itoa(b, value, 16); /* convert the number to be sent to an ascii
hex string */
if(b < 0x10)
            strcat(string,"0");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/*setup register address */
itoa((a | 0x7c), value, 16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/*bring chip select low */
itoa((a & 0x7f), value, 16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/*bring write low */
itoa((a & 0x77), value, 16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/*bring write high*/
itoa((a & 0x7f),value,16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/*bring chip select high */
itoa((a | 0x7c), value, 16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);
/***************************************/
char in_data(a)
unsigned char a;
{
char string[8], value[3], read_data;

/* set string ends to NULLS */
string[7] = value[2] = NULL;
```

```
/* set port C to inputs */
ITC_send("PCC$00\r");

/* setup register address */
itoa((a | 0x7c & 0x7f), value, 16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/* bring chip select low */
itoa((a & 0x7f), value, 16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/* bring read low*/
itoa((a & 0x7b),value,16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/* read Port C */
ITC_send_no_wait("PRC$\r");
read_data = ITC_data();

/*bring read high*/
itoa((a & 0x7f),value,16);
strcpy(string,"PWB$");
strcat(string,value);
strcat(string,"\r");
ITC_send(string);


/* then return to idle state */
ITC_send("PWB$7F\r");

/*return value read*/
return read_data;
}

/****************************************/
/*   ITC_send                           */
/*    sends a string out the comm port  */
/*    then waits for the ITC '>' prompt */
/****************************************/
void ITC_send(str)
char *str;
{
int rval, tval;

while(*str != 0)    /* loop until end of string (NULL) */
            {
    do
        {
        tval = SendChar(0, *str);   /* wait until tx buffer empty */
        }
    while(tval == 0xff00);

    str++;      /* after sending character, go send the next one */
            }
do              /* after final character sent, went until '>' sent back
*/
    {
    do
        {
        rval = Receive(0,0);    /* go get receive buffer status
and/or data */
        }
    while(rval == 0);   /* loop until a character received */
    }
while((char)rval != '>');    /* continue looping until prompt
character rx'd */
}

/****************************************/
/*   ITC_send_no_wait                   */
/*    sends a string out the comm port */
/*    does not wait for any response   */
/****************************************/
void ITC_send_no_wait(str)
char *str;
{
int rval, tval;

while(*str != 0)    /* loop until end of string (NULL) */
            {
    do
        {
        tval = SendChar(0, *str);   /* wait until tx buffer empty */
        }
    while(tval == 0xff00);

    str++;      /* after sending character, go send the next one */
            }
}

/****************************************/
/*   ITC_data                           */
/*  retrieves the data recv'd from a   */
/*  port read operation                */
/****************************************/
char ITC_data()
{
char    string[3], value, *ptr, result;
int rval=0;

ptr = string;
string[2] =  NULL;
do
    {
    do
        {
        rval = Receive(0,0);
        }
    while(rval == 0);
    switch(value = (char)rval)
        {
        case 'O':       /* if characters are O, K, $, or >, ignore */
            break;
        case 'K':
            break;
        case '$':
            break;
        case '>':
            break;
        case 'L':
            printf("\nOverflow or Underflow has occured\n");
            break;
        default:        /* otherwise characters should be results */
            *(ptr++) = value; /* put results in string */
        }
    }
while(value != '>');    /* continue looping until prompt character
rx'd */

/* string should now contain ascii hex value of result */
/* convert from hex string to a number */
if(isdigit(string[0]))
    result = (string[0] - 0x30) * 0x10;
else
    result = (string[0] - 0x37) * 0x10;
if(isdigit(string[1]))
    result = result + (string[1] - 0x30);
else
    result = result + (string[1] - 0x37);
return result;    /* return results converted to a number */
}
```