RMV ELECTRONICS INC. **Application Note:**	**Application #:** 00004
	Date: September 1994
Description: Using the ITC232-A for Period Measurement.	**Status:** Draft version

The basic operating principle to measure period is to generate a known reference frequency, and then use the input pulse stream to gate this frequency into a counter. The counter is then read and the period calculated.

For example, the reference frequency is 1 Mhz (i.e. the counter is being incremented every 1 microsecond). For one period of the input pulse stream, the period gates the reference into a counter. If, at the end of this cycle, the counter reads 11432, the input period can be calculated by:

Counter Value / Reference Frequency = Period in seconds

In our example, the result is 11432 / 1,000,000 = 0.011432 (or about 11 msec.)

The higher the reference frequency, the more counts are received and the higher the effective resolution of the period measurement. The limitation is the size of the counter. If the input period is too long, the counter may overflow and the results will be incorrect.

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 another counter within the 82C53 can be used instead. This would reduce the chip count and lessen the circuit cost. The circuit also includes a 74HC73 dual JK flip flop. The input pulses pass through one of JK flip flops to divide the input frequency by two. This results in an output pulse width that corresponds to the input period.

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 is the reference frequency and directly feeds the MMC counter input. The MMC is initialized as a standard up counter with carry (CY) output. The entire circuit is controlled by a single signal from an ITC232-A port pin (Port B, bit 7) and monitored by the ITC232-A interrupt lines.

With the control signal in a low state, the flip flops are cleared. The counter gate (ABGT) signal is high, disabling counts. To start the sequence, the control line is brought high. This releases the clear signal to the flip flops. The first flip flop U4A will now change output states (the Q output going from low to high) on the first rising edge on the input.
At the same time, the ABGT signal goes from high to low, allowing the MMC counter to begin counting the reference frequency. The falling edge of the input signal has no effect. The next rising edge will again toggle the outputs of U4A. The falling edge of the Q output will clock U4B. The Q output of U4B will then go low, and since this line feeds the J and K inputs of U4A, U4A will be stopped from any further toggling. At the same time U4A Q output goes low, U4A not Q goes high, freezing the MMC counter. The IRQH line will be set, causing the ITC232-A to notify the PC that the cycle has been completed. This state will remain in effect until the ITC232-A once again asserts and releases the flip flop clear lines. The counter can be read and the pulse width calculated. If the MMC counter overflows, the carry output signals the ITC232-A through the IRQL line, notifying the PC.

As described above, the final value would then be calculated by dividing the number of counts by the reference frequency.

The input pulse stream frequency (and thereby the minimum period measurable) is limited by the flip flops. Using ICs such as 74F73 instead of the HC series would increase the maximum input.

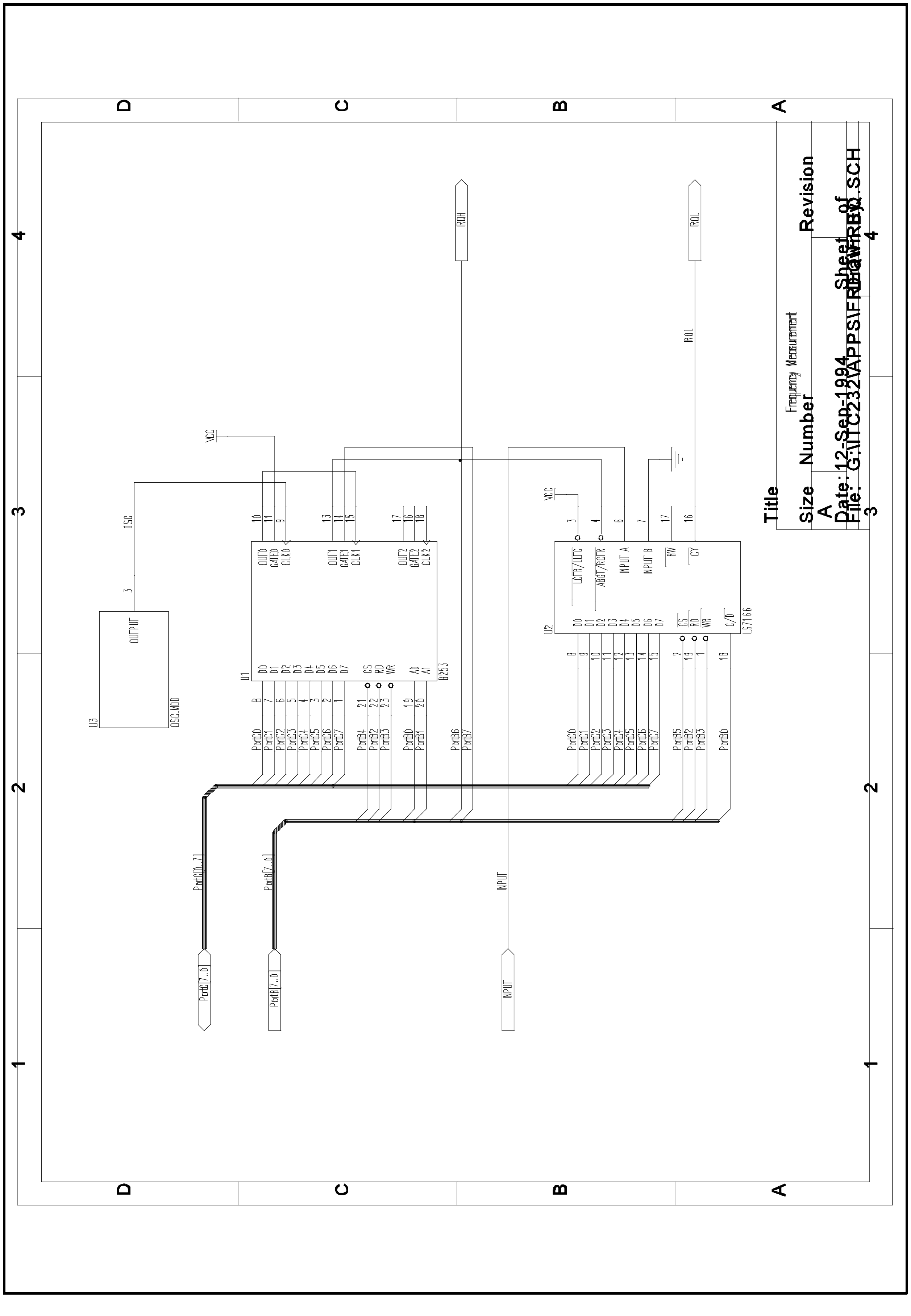
The resolution of the system is limited to the frequency of the PIT 0 output clock. The input to the PIT is 8 Mhz maximum and the minimum divide rate is two. The maximum reference frequency fed to the MMC is therefore 4 Mhz (a time base of 250 nsec.). The maximum resolution in this case would be one count or 250 nsec. Maximum pulse width at this resolution would be 24 bits binary (~16 million counts) or slightly over 4 seconds. For period measurement in excess of 4 seconds, the reference frequency would be reduced by increasing the divide rate in PIT counter 0. By increasing the divider to maximum, periods in the order of hours could be measured.

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 the flip flops 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 count or less at high reference frequency rates).

```
/*
Functional code listing for the input period measurement
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 count cycle status
Port B, bit 7 is an output for controlling the count cycle
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 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);
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;
long    count_total;
float   result, ref_freq;
int     rate_gen;

/*Initialize the PC serial port as for 9600 baud, no parity, 8 bits,
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);

/*Setup PIT counter 0 divide rate to 8.
  This example assumes a 8 Mhz oscillator, resulting a 1
microsecond
  time base. */
```

```
rate_gen = 8; /* 0x0008 */
out_data(PIT_CTR_0,0x08); /*counter 0 low byte*/
out_data(PIT_CTR_0,0x00); /*counter 0 hi 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 flip flop one shot to enable counting */
ITC_send("PWB$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 the flip flops */
             ITC_send("PWB$7F");
   /* 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;
   ref_freq = OSC_FREQ / rate_gen;
             result = count_total / ref_freq;
   printf("\nInput Period = %2.6f seconds.", result);

   /* code can be entered here to evaluate the results and see if
the
   count value is too low. If so, the reference frequency can be
   increased */
   }
else
   {
   /* cycle has caused a counter overflow, first wait for gate to
   complete, disable counting, then print message */
do
   {
   input = Receive(0,0);
   }
   while(input == 0);  /* if nothing received, just wait */
   /* disable the flip flops */
   ITC_send("PWB$7F\r");
   printf("\nOverflow\n");

   /* code can be entered here to reduce the reference
frequency */
   }

} /*end of while(kbhit... loop */
printf("\nInput Period Measurement 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);


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

/****************************************/
/*   in_data                           */
/*  sets up 'bus' for a read operation */
/****************************************/
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 */
}
```


Title
Frequency Measurement
Size A
Number
Revision
Date: 12-Sep-1994
Sheet of
File: G:\ITC232\APPS\FRQ\FREQ.SCH
Drawn By:
U1 8253
U2 LS7166
U3 OSC.MOD
OUTPUT
OSC
VCC
RQH
RQL
INPUT
PortC[7..0]
PortB[7..0]
OUT0 GATE0 CLK0
OUT1 GATE1 CLK1
OUT2 GATE2 CLK2
CS RD WR A0 A1
LCTR/LLTC
ABGT/RCTR
INPUT A
INPUT B
BW
CY
C/D