

MicroTech

MT System Quick Start Guide

Version 1.00

www.mcu.hk

It is assumed that the reader has a basic knowledge of the C programming language. This document is not intended to provide a detailed explanation of C or how to use it – that is left to the many text books written on the subject.

Warning:

Incorrect power connection to any electronic and electrical equipment may seriously damage them or even cause a fire hazard or explosion. Users must take care to identify the correct pins and supply an acceptable voltage to operate them safely.

24th June 2007

Introduction

This purpose of this document is to provide some guidelines and examples of how to use the TinyC programming language to write programs for MT chips. For more information about the syntax of TinyC programming language and the functionality of various MT chips, please refer to documents, "TinyC.pdf" and "MT chips.pdf" for detailed explanation.

The following items are needed to conduct the examples presented in this document:

1. MT IDE
2. MT-20 DemoBoard (preferably MT ARMBBoard)
3. MT Serial LCD module
4. MT IOBoard
5. Appropriate power supply and connecting cables

The MT-20 DemoBoard is chosen because it is small and covers almost all the TinyC functions (except those specific to ARMBBoards). The serial LCD is needed to view the run-time system status and output from the DemoBoard, otherwise, it is extremely difficult to figure out what's going on within the MCU. The IOBoard contains the most common I/O components needed for the input/output operation of the DemoBoard.

Just before we are going to conduct any experiments, a few concepts and facts have to be clarified and bear in mind.

The term "MT System" is the name given to describe a suite of development software and hardware, which including the Text Editor, Compiler, Simulator, Downloader, TinyC programming language and a collection of "pre-programmed" microcontrollers called MT chips.

There is only ONE set of development software for ALL types of MT chips, therefore, from programmer's point of view, it doesn't matter for which MT chip the program he is writing for. The same piece of TinyC program will run on all types of MT chips. However, not all the TinyC functions are supported by all types of MT chips. Unsupported functions are simply ignored by the MCUs if encountered. MT chips differ from each other in the ways of how many programs they can store inside the flash memory, the amount of run-time memory available, the number of I/O pins and the TinyC functions they support.

The programming language instructs the MT chips to perform any form of electronic actions is called TinyC. The Editor is the software where you write TinyC programs in it, the Compiler is the software that translates your written programs into some form of object codes that the MT chips can understand and execute. The Downloader is a piece of software that transfer (via a serial cable) the object codes into the flash memory of a MT chip, in which the chip is going to execute it later.

The program called, "MT chips Simulator" is a piece of software that mimic a real MT chip which loads and executes the object codes produced by the Compiler. The purpose of the Simulator is to provide a simple debugging tool for viewing the results of a running TinyC program, before it get programmed into a MT chip. This is good to find out the program logic flow (i.e. execution steps) of the program. This also indirectly extends the 'life' of the MCU. Since MCUs were designed to be programmable for a specific number of times, but with the current IC manufacturing technology, most of the flash memory cells are programmable for not less than 100,000 times!

The last thing to remember is that the MT run-time system inside the MT chips is a 16-bit system, therefore the maximum signed integer it can take is 32,767 and 65,535 for unsigned, and the ADC/DAC functions in all the MT chips are in 10-bit resolution. This is also the reason why the maximum size of a TinyC program is set at 32K words, it's all because the biggest signed integer it can takes is 32,767. The run-time system uses a signed integer for the value of the PC (program counter).

About the MT Download Cable

MT chips and ARMBBoards used a 3-pin cable to communicate with a PC, as well as to download TinyC programs from it. The given MT download cable is a meter long with 3 wires twisted along it's length together. The female 9-pin D-type connector plugs into a COM port of the PC, the white wire is soldered to pin 3 (RXD), red wire is soldered to pin 2 (TXD), and the black wire soldered to pin 5 (GND).

The delay() Function

Since MCUs are running very fast, it is therefore necessary to slow it down to see the output data. The timing parameter in `delay()` is expressed in milliseconds, therefore, the maximum allowed delay per each call to `delay()` function is 32.767 seconds. However, longer delays can simply be achieved by calling `delay()` several times. The following example shows how to make a delay for 2 minutes:

```
main()
{
  int i;

  writes("Start timing", 1000);
  loop {
    delay(30000); // 30 seconds
    i++;
    if(i == 4) break;
  }
  writes("End", 1000);
}
```

All the `writeX()` output functions are equipped with a delay as part of their parameters, for convenient. If no delay is required, just put a 0 for it.

General I/O Pin Output Operation

There is no need to set beforehand the I/O pin direction, all the I/O pins in all types of MT chips can be used both for input and output, and some MT chips also provide ADC and DAC functionality.

The following experiment is to switch a LED on/off. This can simply be achieved by using the `pinhigh()`, `pinlow()`, `toggle()` and `delay()` functions.

```
main()
{
  loop {
    pinlow(0); // 0V output
    delay(1000); // 1 second delay
    pinhigh(0); // 5V output
    delay(1000); // 1 second delay
  }
}
```

The above program simply turns the output of the I/O Pin, 0 to Low (0V) and High (5V) repeatedly. The **loop** statement is used to repeat all the statements and functions within it, otherwise, all the statements in the program will execute only once and program exits. Because the MCU is running very fast, it would be impossible to see the output without putting some delays, by using the `delay()` function.

We can achieve the same programming results by using the `toggle()` function and the above program can be simplified like this:

```
main()
{
  loop {
    toggle(0); // 0V->5V, 5V->0V output
    delay(1000); // 1 second delay
  }
}
```

The `toggle()` function reverses the state of I/O Pin 0.

Sample programs: led-xx.prg

General I/O Pin Input Operation

The TinyC function `pinstate()` is used to read the state of an I/O pin. It returns a 0 to indicate that the pin is Low, otherwise it returns 1 as High. Since the design of all the I/O pins of MT chips are Low activated. Normally, all the I/O pins of the MT chip are put in the High state to receive input, users can pull a pin Low by connecting it to Ground, 0V.

The following program scans for the status of I/O Pins and displays the appropriate message:

```
main()
{
  loop {
    if(!pinstate(0)) writes("I/O Pin 0 is Low", 500);
    else
    if(!pinstate(1)) writes("I/O Pin 1 is Low", 500);
    else
    if(!pinstate(2)) writes("I/O Pin 2 is Low", 500);
    else writes("No Pin i Low", 500);
  }
}
```

The size of compiled object codes generated for the expressions of **if** statements written like above is more compact than those written as below:

```
.....
  if(pinstate(0) == 0) writes("I/O Pin 0 is Low", 500);
  else
  if(pinstate(1) == 0) writes("I/O Pin 1 is Low", 500);
  else
  if(pinstate(2) == 0) writes("I/O Pin 2 is Low", 500);
.....
```

The `getkey()` and `scankey()` functions are only applicable for the MicroBoard, which returns the ASCII values of the numeric keys ('0'-'>49, '1'-'>50, etc) being pressed, or the pre-defined values for the special PC (255), SYSTEM (254) and USER (253) keys. Function `scankey()` differs from `getkey()` in the way that it does not wait for a key to be pressed and returns 0 immediately if a key has not been pressed.

Sample programs: key.prg, piano.prg

Various Output / Input Functions

TinyC is equipped with various functions for sending characters to a MT Serial LCD module, PC COM port, serial EEPROM or just a raw byte at the I/O pins 0-7 for parallel input to other IC or circuit. By default, all outputs go to the LCD. Users can disable the output to LCD by pulling the LCD S1 pin Low (by connecting a 5K-10K resistor between S1 and Ground), then the extra 3 LCD pins (S1, S2 & S3) can be used as normal I/O pins. The followings listed out some output functions:

```
writes("Hello", 1000); // o/p to serial LCD, with delay
writes("$Hello", 1000); // o/p to serial COM port, with delay
puts("Hello"); // o/p to serial COM port, no delay
writei('d', 123, 1000); // o/p to serial LCD, with delay
writei(232, 123, 1000); // o/p to serial COM port, with delay
putint(123); // o/p to serial COM port, no delay
writec(0, 'A', 1000); // o/p to serial LCD, with delay
putchar('A'); // o/p to serial COM port, no delay
writeb(0, 0xAA, 1000); // o/p to I/O pins 0-7, bit pattern 10101010, with delay
writem(24, 100, 'A'); // o/p to serial EEPROM 24C02, storage location 101, counting from 0
writem(93, 100, 'A'); // o/p to serial EEPROM 93C56, storage location 101, counting from 0
```

The `readm()` is used to read a data byte from a specified location of EEPROM, `getchar()` is used to read a data byte from a serial port with data availability detection. For examples:

```
x = readm(24, 0); // read the 1st data byte from 24C02
x = readm(93, 0); // read the 1st data byte from 93C56
x = getchar();    // get the next available byte from a serial COM port, returns -1 if not available
```

```
// MicroBoard program
// Show the 7-Segments LED representations of both uppercase and lowercase characters
```

```
#define DELAY 1000

main()
{
    loop {
        writes("abcdefghijklmnopqrstuvwxyz", DELAY);
        writes("ABCDEFGHIJKLMNOPQRSTUVWXYZ", DELAY);
    }
}
```

```
// Demonstrate different output formats of writei()
```

```
#define DELAY 1000

main()
{
    loop {
        writei('d', 00001, DELAY); // leading 0s will be ignored!
        writei('d', 0x100, DELAY); // hex number display in signed decimal
        writei('o', 0x100, DELAY); // hex number display in octal
        writei('x', 0x100, DELAY); // hex number display in hexadecimal
        writei('u', 65535, DELAY); // maximum unsigned decimal integer
        writei('x', 65535, DELAY); // maximum unsigned hexadecimal integer
        writei('o', 65535, DELAY); // maximum unsigned octal integer
        writei('d', 32767, DELAY); // maximum +ve signed integer
        writei('d', -32768, DELAY); // maximum -ve signed integer
    }
}
```

```
// Bits complement test
// Do notice the on/off pattern of the LED bars
```

```
main()
{
    int i;

    i = 0xAA; // bits: 10101010
    loop {
        writeb(0, ~i, 1000); // unformatted byte output function, complement for common anode LEDs
        i = ~i; // bits: 01010101
    }
}
```

Sample programs: writec.prg, writei.prg, writes.prg, writeb-x.prg, key.prg, led-key.prg

Communicating with a PC

Functions `writec()`, `writei()`, `writes()`, `puts()`, `putint()`, `putchar()` and `getchar()` are used for sending and receiving data from a PC or any other devices that are compliant with the RS232 communication protocol. Before a communication can begin, a setup by a called to `setcomm()` function is necessary to specify the data transfer rate and establish the communication channel. The data format in transfer was pre-defined in

8N1 mode (i.e. 8 bits, no parity and 1 stop bit), and users cannot change it, but can set the baudrate ranging from 600 to 115200.

Note: Please refer to "TinyC.pdf" for the proper parameters and formats of writeX() functions.

Communication routines in MT-08 and MT-14 chips are implemented in software, and they are fixed at 9600 and in half-duplex mode only. In half-duplex mode means that both communicating parties can send data to each other, but not both are sending and receiving at the same time, otherwise this is called a full-duplex mode.

The getchar() function returns the next available character, otherwise it returns -1 when there is no character waiting for it in the incoming buffer. Function getchar () was implemented in this way as to avoid endless waiting for data. Users have to write programs in such a way that by checking the availability of data and continue executing.

Normally, there is only one RS232 COM port available on the MT chips, but the ARMBboards can have up to 4 COM ports (i.e. 2 normal UART ports + 2 USB COM ports). To keep the design of the TinyC language simple, we use the sys() function to switch and set the immediate following COM operations only apply to the current active COM port.

```
// COM port test, for all types of MT chips
// Action: Echo received characters back to PC.
// If there is only one COM port available on the MT chip, no need to specify the COM port number.
// By default, the MT system selects the first available COM port.

main()
{
    int d;

    // sys(0, 0);           // select COM port 0
    // sys(0, 1);           // select COM port 1
    // sys(0, 2);           // select ARMBBoard 1st virtual USB-COM port
    // sys(0, 3);           // select ARMBBoard 2nd virtual USB-COM port

    writes("9600", 500);
    setcomm(9600);          // enable COM and set baudrate at 9600,8N1
    loop {
        loop {
            d = getchar();    // wait to receive a character
            if(d != -1) break;
        }
        writec(0, d, 10);     // display it, comment this out to run faster!
        putchar(d);          // echo it back to sender
        if(d == '?') writes("$Hi!", 100); // output a message to COM if '?' is received
        if(d == '!') break;   // exit if '!' received
    }
    writes("$end", 200);
    setcomm(0);             // disable Comm
    writes("end", 2000);
}
}
```

```
// 4 x COM Ports test for MT ARMBboards
// This program opens 4 ports at the same time (optionally with different baudrates)
// NEW additional baudrates settings for MT ARMBboards are:
// 384 = 38400
// 576 = 57600
// 1152 = 115200
// Since the maximum signed integer for the 16-bit MT System is 32767, therefore, these changes are needed

main()
{
}
```

```

int i;

writes("4xCOM Test", 500);
sys(100, 0);      // Enable USB COM ports
sys(0, 0);        // select COM port 0
setcomm(9600);   // set baudrate for COM port 0, 9600,8N1
sys(0, 1);        // select COM port 1
setcomm(9600);   // set baudrate for COM port 1, 9600,8N1

// no need to set the baudrates for USB-COM ports, the ARMBBoard will detect it automatically!

loop {
  writei('u', i++, 0); // Just a counter to show it is running

  sys(0, 0); // select COM port 0
  puts("This line will appear on UART-0", 500);

  sys(0, 1); // select COM port 1
  puts("This line will appear on UART-1", 500);

  sys(0, 2); // select USB-COM port 0
  puts("This line will appear on USB-COM-0", 500);

  sys(0, 3); // select USB-COM port 1
  puts("This line will appear on USB-COM-1", 500);
}
}

```

Sample programs: comm.prg, comm2.prg, 4xcom.prg

Analogue to Digital Conversion

Some versions of MT chips come with the ADC capability with 10-bit resolution, which means that the value of the input voltage is represented by a number in range of 0 to 1023 with reference to the ADC Reference Voltage (Vref). Normally, the reference voltage for the ARMBboards is set to 3.3V, whereas, for the rest of MT chips, the Vref is set to 5V.

By referring to the MT chips Hardware Reference manual, users can find out which MT chips and I/O pins of them have the ADC function.

The TinyC function `readadc()` is used to get the voltage and converts it into a number. The following example demonstrates how to read the voltage presented on I/O pin 1 and displays it on the MT Serial LCD:

```

main()
{
  writes("ADC", 500);
  loop {
    writei('d', readadc(1), 300);
  }
}

```

Sample programs: adc.prg, adc-servo.prg

Digital to Analogue Conversion

The opposite of ADC is DAC, which converts a given number into an appropriate amount of output voltage. Amongst the MT chips, only the ARMBboards come with the DAC capability, also in 10-bit resolution, and only one I/O pin is dedicated to this function.

Before the I/O pin can perform DAC, it has to be changed (by using the `sys(50, x)` function) from normal digital I/O function into analogue DAC function. Since the output current of this pin is so low that it cannot be used to drive a speaker directly. Therefore, some sort of amplification is needed in order to drive a speaker. Handheld MP3 players are powered with this kind of IC to convert digital signals into human audible sound.

However, user can still use an oscilloscope or voltmeter to measure the output voltage from this DAC pin.

```
main()
{
  writes("DAC", 500);
  sys(50, 3001); // enable DAC
  loop {
    sys(50, 1023); writei('d', 1023, 500); // ~3.3V
    sys(50, 512); writei('d', 512, 500); // ~1.65V
    sys(50, 0); writei('d', 0, 500); // ~0V
    sys(50, 512); writei('d', 512, 500); // ~1.65V
  }
  sys(50, 3000); // disable DAC
}
```

Sample programs: dac.prg

Sound / Pulse Generation

The TinyC `sound()` function is used to generate human audible tones. It takes 3 parameters, the first one is used to indicate which I/O pin is for the output. The second parameter indicates the frequency of the square wave (50:50 duty cycles) being output, and the last parameter is used to set the duration of the output.

The program below generates 2 tones (2 seconds each) at I/O pin 12:

```
main()
{
  loop {
    sound(12, 500, 2000); // 500Hz, 2 seconds
    sound(12, 1000, 2000); // 1000Hz, 2 seconds
  }
}
```

Since the main purpose of `sound()` is for the generation of musical notes (<2000Hz), the accuracy of the output frequency get deviated when set at a higher value (>10KHz). This function can also be used to generate square waves/pulses for other circuits.

Sample programs: sound.prg, piano.prg

NOTE: Whenever the timer interrupt function is enabled in a program, the `sound()` function for the following MT chips is disabled. This is because both functions use the same hardware interrupt in these chips.

MT-40r1
 MT-40r2
 MT-20
 MT-28

Infra-Red Decoder Operation

The `ircode()` function is used to detect and decode 38KHz IR codes sent in the APEX, NEC, PIONEER and HITACHI formats. It returns a value between 0 and 254 if successful or 255 if not. The key (e.g. '1') pressed on the remote controller will not necessarily correspond to the value of 1 as decoded by the `ircode()` function, it's all depends on the manufacturer definition for the keys. But, as soon as you can see a number being shown up on the display, which means that you got the correct type of remote controller.

```
// Detect and show IR code received from an IR transmitter
// Function ircode() returns 255 if no IR code is detected

main()
{
  int x;

  writes("ir Code", 500);
  loop {
    x = ircode();           // get IR code
    if(0 <= x && x <= 254) writei('d', x, 200);
    if(x == 9) break;      // exit if key 9 is pressed
  }
  writes("end", 5000);
}
```

Sample programs: `ircode.prg`

R/C Servo Operation

The `servo()` function generates positive pulses of width from 0.5 to 2.5ms to control RC servos. The degrees of throw may vary between different types of servos. The width of pulses are designed to cater for a wide ranges of different servos. Care must be taken **NOT** to overdrive servos to their extreme limits at both ends. Start with 90° first, then work out the upper and lower limits the servo can take. Otherwise, the gears inside the servo might easily get damaged due to overdriving. This system is designed based on standard sized RC servos, like Futaba 3001 and 3003, which are capable of throws of more than 180 degrees. The smaller ones, like Futaba 3106, have smaller degrees of throws.

Since RC servos draw a lot current when they move, therefore, it is recommended a separate power source (Ni-Cad, Ni-MH or Li-Ion batteries) be supply to power the servos, with the negative wire connects to the **Ground** of the MT chip as the common signal **GND** reference.

The following program moves the servo connected to I/O pin 0 back and forth repeatedly:

```
main()
{
  loop {
    servo(0, 50);
    delay(1000);
    servo(0, 130);
    delay(1000);
  }
}
```

It is possible to move several servos simultaneously. In this case simply add 100 to the pin numbers, then issue the "move all" servo command, i.e. "`servo(200, 0);`", with 200 as its pin number.

The following program moves 2 servos connected to I/O pins 3 and 4, first individually then simultaneously:

```
main()
{
  loop {
    // move servos individually!
    servo(3, 50);
    delay(1000);
    servo(3, 130);
    servo(4, 50);
    delay(1000);
    servo(4, 130);
    delay(1000);
  }
}
```

```

// move servos simultaneously!
servo(103, 50); // set angle
servo(104, 50); // set angle
servo(200, 0); // start moving all
delay(1000);

// move servos simultaneously!
servo(103, 130); // set angle
servo(104, 130); // set angle
servo(200, 0); // start moving all
delay(1000);
}
}

```

The MT-40 series chips are able to support up to 32 servos (46 in ARMBboards) on their I/O pins, but, in order to move servos quickly and smoothly at the same time, a maximum of 10 servos is recommended.

Note: The servo() function will stop emitting pulses after it's execution. In order to send continuous controlling pulses to servos to maintain their holding power, a dedicated servo controller (like MT-SC21 or MT-SC13) must be deployed. System function sys(30, 1) is used to control the servo() to emit just a single pulse or a train of 40 pulses.

Sample programs: servo-x.prg, adc-servo.prg

Real Time Clock Operation

The ARMBboards have a built-in RTC, and by supplying the appropriate parameters to TinyC functions sys() and readsys(), user can set and read back the values of time.

```

// Real Time Clock demo on MT ARMBboards

main()
{
  int h,m,s;

  writes("RTC Demo", 1000);
  writes("~XCEOL", 0); // disable the 'clear to end of line' feature of LCD

/*
  writei('d', readsys(11), 1000); // get year
  writei('d', readsys(12), 1000); // get month
  writei('d', readsys(13), 1000); // get day
  writei('d', readsys(17), 1000); // get DOW
  writei('d', readsys(18), 1000); // get DOY
*/
/*
  sys(11, 2006); // set year, 0..4095
  sys(12, 9); // set month, 1..12
  sys(13, 15); // set day, 1..28~31
  sys(14, 16); // set hours, 0..23
  sys(15, 57); // set minutes, 0..59
  sys(16, 00); // set seconds, 0..59
  sys(17, 5); // set day-of-week, 0..6, Sunday=0
*/
  writec(2, ':', 0); // show separator
  writec(5, ':', 0); // show separator
  loop {
    h = readsys(14); // get hours
    m = readsys(15); // get minutes

```

```

s = readsys(16);           // get seconds
writec(0, h/10+'0', 0);
writec(1, h%10+'0', 0);
writec(3, m/10+'0', 0);
writec(4, m%10+'0', 0);
writec(6, s/10+'0', 0);
writec(7, s%10+'0', 0);
}
}

```

Sample programs: `rtc.prg`

System Management Functions

There are a number of system management functions in TinyC help to set and read back run-time system variables, such as the `freemem()` function is used to return the number of current available memory units, the `random()` function is a pseudo-random number generator which generates integers within the ranges between 0 and 32767. By supplying different parameters to functions `sys()` and `readsys()`, users can set and get the times of the system real time clock, enable USB COM ports, enable and disable the DAC and reset the interval for the Timer Interrupt function.

Sample programs: `sys.prg`, `sys2.prg`, `random.prg`

Using I/O Pin External Interrupts

An external interrupt function is a special kind of function which is called when it's associated I/O pin is activated i.e. pulled LOW. When the interrupt function finishes, the program resumes execution from the point where the interrupt was activated.

Syntax: `function_name() interrupt 1 { }`

This following defines `say_hello()` to be an interrupt function associated with I/O pin 1. It is activated when I/O pin 1 goes LOW.

```

say_hello() interrupt 1
{
  writes("Hello", 1000);
}

```

I/O interrupts in TinyC are not hardware interrupts. The I/O pins are checked (polled) between execution of each instruction in the program. Therefore, statements that take a while to finish e.g. `writes("hello", 5000)` will slow down the response to the interrupt. In this case the interrupt won't be acknowledged for 5 seconds.

In order to use external interrupts, the following rules must be observed:

- Only up to 5 interrupt functions can be defined in a program.
- Apart from the `main()` function, any function can be an interrupt function.
- Each interrupt function must be associated with an I/O pin.
- Interrupt functions cannot receive or return values.
- An interrupt function cannot call itself or another interrupt function but can call any other normal function.
- Interrupt functions can be called by any other normal functions anywhere within the program just like normal function calls. However this is a bad programming practice and is not encouraged. Use it mainly for testing purposes.
- Only one interrupt function is activated at any given time i.e. interrupts within interrupts are not allowed.

To disable the interrupt capability of an interrupt function, just delete the "interrupt" keyword and the associated I/O pin number. The function then reverts back to a normal function.

The interrupt function will be re-activated if the associated pin is still LOW when the interrupt function exits. If this is a problem then it is up to the user program to test and wait for the associated I/O pin to go HIGH before exiting the interrupt function.

```
// I/O Pin interrupts demo

#define DELAY 100

main()
{
  int i;

  loop {
    writei('d', i++, 200);
  }
}

ISR0() interrupt 0
{
  writes("Hello", DELAY);
}

ISR1() interrupt 1
{
  writes("Man", DELAY);
}

ISR2() interrupt 2
{
  writes("This", DELAY);
}

ISR3() interrupt 3
{
  writes("is", DELAY);
}

ISR4() interrupt 4
{
  writes("great!", DELAY);
}
```

Sample programs: `isr.prg`, `isr2.prg`, `sys.prg`, `sys2.prg`

Using the Timer Interrupt

TinyC provides a timer interrupt function. This makes tasks such as refreshing the data on a 7-segment display in the background much easier. Without a timer interrupt this would be almost impossible. The timing interval is specified in milliseconds from 0 to 32767 allowing a timed interrupt every 32.767 seconds if set to the maximum.

Syntax: `function_name() timer delay_time { }`

The following defines `message()` to be a timer function and sets the timer interrupt at 30000 milliseconds.

```
message() timer 30000
{
  writes("Hello, see you later!", 1000);
}
```

This function is called every 30000mS (30 seconds) and displays the message.

The system management function **sys()** can be used to change the timing value, as follows:

```
sys(1, 1000);      // reset to 1000ms (1 second) intervals
sys(1, 0);         // setting = 0 means stop the timer! No more timer interrupts.
                  // The first parameter refers to the timer.
                  // The second parameter sets the timing interval.
```

NOTE: Whenever the timer interrupt function is enabled in a program, the sound() function for the following MT chips is disabled. This is because both functions use the same hardware interrupt in these chips.

MT-40r1
MT-40r2
MT-20
MT-28

```
// Timer interrupt demo

// Be careful when resetting the timer's value! After a pin has been PULLED LOW,
// it has to be released (i.e. goes HIGH again) for the timer interrupt to
// continue its running. NEVER use a 'variable' to control the running of the
// timer, it's because the program and the timer interrupt function are both running
// at the same time, the value of the control variable might be non-deterministic!

int x;

main()
{
  loop {
    if(!pinstate(0)) sys(1, 0); // Stop timer!
    if(!pinstate(1)) sys(1, 10); // reset timer, goes faster
    if(!pinstate(2)) sys(1, 500); // reset timer, goes slower
    if(!pinstate(3)) sys(1, 100); // back to original setting
  };
}

time0() timer 100
{
  writei('d', x++, 0);
}
```

Sample programs: timer.prg, timer2.prg, sys.prg, sys2.prg

Summary of TinyC Functions

Note: Not all the functions are available on all types of MT chips.

Output functions

pinhigh(), pinlow(), toggle(), writei(), writec(), writeb(), writes(), writem(), sound(), servo(), puts(), putint(), putchar()

Input functions

pinstate(), readm(), readadc(), readsys(), ircode(), scankey(), getkey(), getchar()

A/D conversion

readadc() (only on MT-40-AD, MT-28, MT-20, MT-14, MT-08 and ARMBboards)

D/A conversion

By using sys(50, x) (only on ARMBboards)

Sound generation

sound()

Serial communication

setcomm(), puts(), putint(), putchar(), getchar()

Data memory (24C02 and 93C56) management

readm(), writem() (except MT-08 and MT-14, 24C02 on ARMBboards only)

Time delay function

delay()

RC servo control function

servo()

Infra-Red decoder function

ircode()

Keyboard input functions

scankey(), getkey() (MT-40r1 and MT-40r2 only)

Byte output function

writeb()

System management functions

sys(), readsys(), exit(), freemem(), random()