

How to take advantage of the X2 feature in TS80C51 microcontrollers?

by Patrice Graziotin, C51 technical marketing

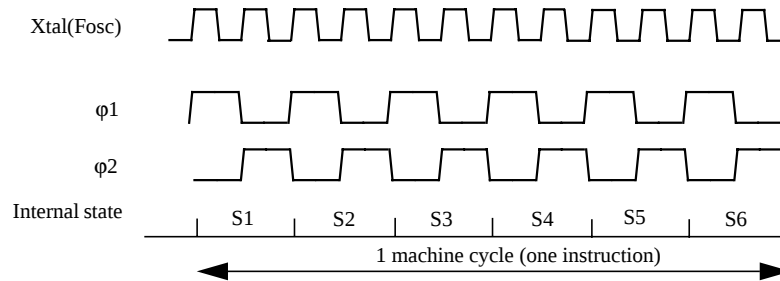
Table of contents

1. What is the X2 feature	2
2. Are the X2 core and the previous one compatible	2
3. How to activate the X2 feature	3
4. Which devices are concerned by the X2 feature	4
5. Which advantages can be taken of the X2 feature	5
5.1 Absolute CPU horsepower improvement	5
5.2 EMI reduction with same CPU performance	5
5.3 Operating Icc reduction with same CPU performance	5
5.4 Crystal cost reduction with same CPU performance	5
5.5 Switchable CPU horsepower	5
5.6 Solving customer problems	5
6. Which things must be taken into account	6
6.1 Clock duty cycle requirements	6
6.2 Time dependant peripherals set-up values	6
7. Let's test the X2 feature	7
7.1 Environment	7
7.2 Preliminary task	7
7.3 Test program	7
7.4 Program results	7
8. Appendix : X2_show.C program	8

1. What is the X2 feature ?

Basically, the X2 feature allows to run one instruction, also called one machine cycle, in 6 clock periods instead of the 12 generally required in Intel's C51 architecture based devices.

Previous C51 core used a 6 state machine, each composed of two phases, $\phi 1$ and $\phi 2$. During $\phi 1$ happened things like bus pre-charge, temporary ports pull-up activity, and during $\phi 2$, happened the evaluation of all calculation and external signals update, like ports SFR, ALE, PSEN. Thus, the previous core needed 12 clocks period per machine cycle.

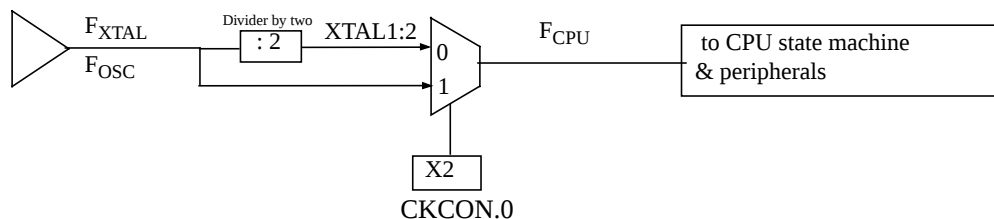


In the new X2 core, designed with VHDL with logic synthesis, everything happens on a single phase, equivalent to phase 2. The new X2 core requires then only 6 clock periods to achieve a machine cycle

2. Are the X2 core and the previous one compatible ?

YES, they are !

To keep upward compatibility, a divider by two has been inserted between the on-chip oscillator and the state machine. Like this, the X2 core requires 12 clock periods for one machine cycle. This divider by two is by default enabled, so a customer can replace an old part, or a competitor part, with an X2 one without any problem, and without changing anything in his code.



If the X2 mode is activated, the Atmel microcontroller will exactly behave as a microcontroller in standard mode at double crystal speed. *From an external point of view, the activity of the signals (ALE, PSEN, RD, WR, addresses, data, ports) will be exactly the same for the two micro-controllers.*

3. How to activate the X2 feature ?

Set bit 0 in the new CKCON (Clock CONTROL) register !

No special order is required. Every X2 device can have the X2 feature switched on or off thanks to a simple instruction. This switch can be made anytime during the code execution (not only after reset).

CKCON - Clock Control Register (8Fh)

7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	X2

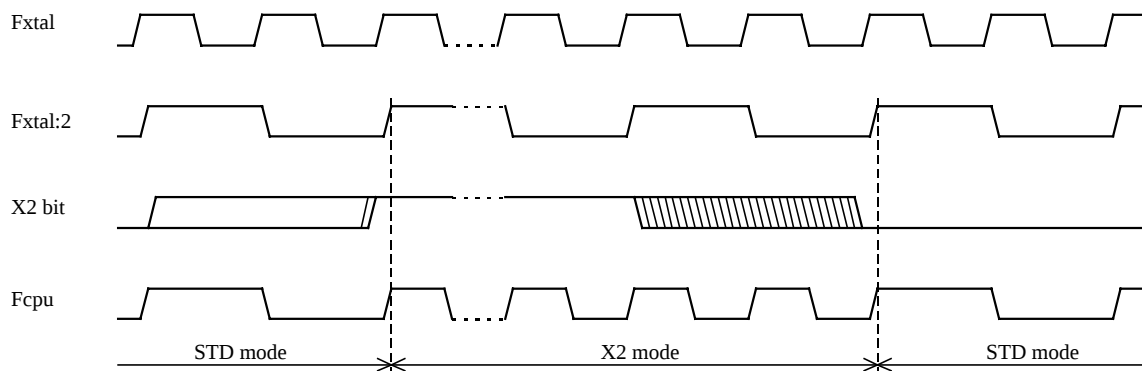
Bit Number	Bit Mnemonic	Description
7	-	Reserved The value read from this bit is indeterminate. Do not set this bit.
6	-	Reserved The value read from this bit is indeterminate. Do not set this bit.
5	-	Reserved The value read from this bit is indeterminate. Do not set this bit.
4	-	Reserved The value read from this bit is indeterminate. Do not set this bit.
3	-	Reserved The value read from this bit is indeterminate. Do not set this bit.
2	-	Reserved The value read from this bit is indeterminate. Do not set this bit.
7:1	-	Reserved The value read from this bit is indeterminate. Do not set this bit.
0	X2	CPU and peripheral clock bit Clear to select 12 clock periods per machine cycle (STD mode). Set to select 6 clock periods per machine cycle (X2 mode).

Reset Value = XXXX XXX0b

Not bit addressable

An internal system avoids glitches to occur at transition.

X2 bit is validated on XTAL₂ rising edge to avoid glitches when switching from X2 to STD mode.



4. Which devices are concerned by the X2 feature ?

Every new device built on 0.5 design rules; included are the following :

- TS80C31X2 C31 ROMless
- TS80C32X2 C32 ROMless
- TS80C51RA2 ROMless with PCA and 256 bytes of XRAM
- TS80C51RD2 ROMless with PCA and 768 bytes of XRAM
- TS80C51U2 C32 ROMless with 2 UARTs

- TS80C52X2 8K ROM
- TS80C54X2 16K ROM
- TS80C58X2 32K ROM
- TS83C51RB2 16K ROM with PCA and 256 bytes of XRAM
- TS83C51RC2 32K ROM with PCA and 256 bytes of XRAM
- TS83C51RD2 64K ROM with PCA and 768 bytes of XRAM
- TS83C51U2 16K ROM with 2 UARTs

- TS87C52X2 8K OTP
- TS87C54X2 16K OTP
- TS87C58X2 32K OTP
- TS87C51RB2 16K OTP with PCA and 256 bytes of XRAM
- TS87C51RC2 32K OTP with PCA and 256 bytes of XRAM
- TS87C51RD2 64K OTP with PCA and 768 bytes of XRAM
- TS87C51U2 16K OTP with 2 UARTs

5. Which advantages can be taken of the X2 feature ?

5.1 Absolute CPU horsepower improvement

As explained previously, the CPU core has been re-designed to work with only 6 clock periods. Furthermore, its frequency capability has been improved.

With X2 on, the combination of this two enhancements allows to clock the core up to 60 MHz, with a 30 MHz oscillator at 5V, and up to 40 MHz with a 20 MHz oscillator at 3V.

At 5V, the chip behaves the same as if it was an old one, clocked with a 60 MHz crystal ! This leads to 5 MIPS !

5.2 EMI reduction with same CPU performance

With X2 on, the customer can keep the same CPU performance with a divided by 2 crystal frequency. The lower the crystal frequency, the lower radiated emissions.

5.3 Operating Icc reduction with same CPU performance

With X2 on, the customer can keep the same CPU performance with a divided by 2 crystal frequency. This means that the on-chip oscillator runs two times slower, and its power consumption is quite proportional to its operating frequency. Furthermore, reducing operating Icc will also reduce conducted emissions.

Remarks :

in Idle mode, the CPU is frozen but the on-chip oscillator is still running. Thus, there will also be current saved.

In power-down mode, the on-chip oscillator is frozen, so there won't be any benefit in the X2 feature.

5.4 Crystal cost reduction with same CPU performance

With X2 on, the customer can keep the same CPU performance with a divided by 2 crystal frequency. The lower the crystal frequency, the cheaper it is.

5.5 Switchable CPU horsepower

The X2 feature can be switched on and off dynamically. An internal system prevents any glitches to occur at transition. This means that a customer can have a low horsepower system (and so, a low current sinking system, for example a 12 MHz clocked device), and he has a potentially two times more powerful system if he puts the X2 on, for example to serve a time critical interrupt. By simply switching a bit, he's turning his device to a 24 MHz equivalent one. When the peak calculation is no more needed, he can put the X2 off to return to his low current sinking system.

5.6 Solving customer problems...

A Atmel Wireless & Microcontrollers customer had a dilemma :

Customer Problem : his application uses RF transmission: on the one side, his micro-controller has to be clocked under 11 MHz to avoid RF disturbance. On the other side, he needs to run his micro at least at 16 MHz to fit his calculation requirements.

X2 solution : with the X2 feature enabled, the customer uses an 8 MHz crystal : his CPU performance is equivalent to a 16 MHz one without disturbing his RF line.

6. Which things must be taken into account ?

6.1 Clock duty cycle requirements

When the X2 feature is on, the crystal duty-cycle must be between 40 to 60%, which is quite always the case. This is because the divider by two is by-passed and then, the oscillator output directly drives the state machine. The state machine will behave correctly with a 40-60 duty-cycled clock drive.

If the X2 feature is not used, the divider by two will put in good shape (50-50 duty cycle) any incoming clock signal (but respect Tchcx and Tclcx, clock high and low time durations).

6.2 Time dependant peripherals set-up values

Not only the core but also the peripherals are affected by the by-pass of the divider by two when the X2 feature is enabled. This means that the peripherals will also be clocked two times faster. So, the time dependant peripherals will be affected as follows:

- Timers and PCA : counting (when configured as timers), compare match and overflow will be two time faster
- UARTs baud rates will be doubled
- Watchdog delay will be shortened by two

The implication is to re-calculate the time values if the X2 is switched on or off dynamically, or if an old design is modified to take advantage of the X2 feature.

In next micro-controllers, we will have the possibility to enable the clock doubling independently for the core, timer 0, timer 1, timer 2, PCA, UART and watchdog in the CKCON register. So this point won't be an issue any longer. The first device which will have this enhanced CKCON register is the Atmel Wireless & Microcontrollers 64K FLASH micro-controller which will be released in 1999. (64K Flash, low voltage ISP, PCA, 1K XRAM...)

7. Let's test the X2 feature !

7.1 Environment

The following has been done under Keil Software system, using vision and Dscope. We also use the TSC80251 starter kit (usable with C51 devices) which can be ordered at Atmel Wireless & Microcontrollers sales offices. This starter kit provides with 2 serial links and LEDs connected to port 1 of the micro-controller. The first serial link is used by Dscope and the monitor program to control the board, and the second serial link, which is the micro-controller one, is connected to a console to display results and catch keyboard activity.

The console has to be configured at 2400 bauds, 8 bits, no parity.

7.2 Preliminary task

The X2 Clock Control register (CKCON, @8Fh) is not declared in C51 include files from Keil at the moment (Q2 99). So we have to declare it.

Let's copy the C52 description file (reg52.h) to reg52_x2.h and let's add the following declaration lines:

```
/* X2 Extension */
```

```
sfr CKCON = 0x8F;
```

In C source file, we now just have to include reg52_x2.h to be able to control the X2 feature.

7.3 Test program

X2_show.C is given in the appendix at the end of this document

This program demonstrates the advantages of the X2 feature; basically it makes 2 things:

- sort from bottom to top 100 values which are initially sorted from top to bottom
- blink the starter kit leds 0 to 6

The X2 mode can be toggled by pushing the starter kit INT1 button.

The X2 mode can also be toggled by hitting the 'X' key on the console, and cleared by hitting any other key on the console.

7.4 Program results

If the X2 mode is off, the sorting time is 3647 ms and the LEDs blink quite slowly.

With the X2 on, the sorting time is 1810 ms and the LEDs blink two times faster.

At first sight, it appears strange that the sorting time is better than two times faster... This is because every millisecond, the timer 2 generates an interrupt to increment the variable which counts the number of milliseconds.

When the X2 mode is off, the timer 2 generates an interrupt every 1000 counts at 12 MHz (12000 clock periods). So if we suppose that it takes 1000 count (1000 machine cycles) to sort one number, the interrupt routine would be served 100 times (there are 100 numbers to sort).

When the X2 mode is on, the timer 2 generates an interrupt every 2000 counts at 12 MHz since the timer 2 input clock is two times faster (but this is always 12000 clock periods and 1 ms). So if we still suppose that it takes 1000 count to sort one number, the interrupt routine would be served 50 times.

We found here that in X2 mode, the timer 2 interrupt routine is served two times less. As serving this IT takes some overhead time, this overhead is reduced. That's why the time improvement is better than 2.

8. Appendix : X2_show.C program

```

/*-----
X2_show.C
This program demonstrates the advantages of the X2 feature; basically it makes
2 things:
sort from bottom to top 100 values which are initially sorted from top to bot-
tom
blink the starter kit leds 0 to 6
The X2 mode can be toggled by pushing the starter kit INT1 button.
The X2 mode can also be toggled by hitting the 'X' key on the console, and
cleared by hitting any other key on the console.
The console has to be configured at 2400 bauds, 8 bits, no parity.
Copyright 1999 Atmel Wireless & Microcontrollers
Written by Patrice GRAZIOTIN (patrice.graziotin@atmel-wm.com)
-----*/

#pragma DEBUG OBJECTEXTEND CODE      /* pragma lines can contain state C51 */
                                     /* command line directives */

#include <regx2.h>                    /* special function register declarations*/
                                     /* for the intended 8051 derivative */
                                     /* including the X2 CKCON register */

#include <stdio.h>                    /* prototype declarations for I/O func-
tions*/

#include <intrins.h>

/*-----
The following buffer is required if we're using the RISM monitor for debugging.
The allocated space is used by RISM for its internals.
-----*/
#ifdef RISM
const char data rism_1 [24] _at_ 0x28;
#endif

/*-----
Global variables
-----*/
unsigned int to_sort = 100;          /* number of data to sort */
unsigned int to_print = 14;          /* number of data to print */
char * ram_begin = 0x80;            /* first data address */
bit x2_mode = 0 ;                   /* if 1, x2_mode is ON */
unsigned int timer = 0; /* used to stamp time; timer variable is incremented
*/

bit exit = 0;                       /* every millisecond */
                                     /* used to exit main loop */

/*-----
Delay procedures
-----*/
void tempo () {
    unsigned int loop;
    for (loop=0; loop<10; loop++) {}
}

void delay () {
    unsigned int loop;
    for (loop=0; loop <10000; loop++) {
    }
}

```

```

}

/*-----*/
/* this procedure toggles the X2 feature and adapt timers */
/*-----*/
void deal_with_x2 () {
    if (x2_mode)
    {
        /* X2 MODE */
        CKCON=1;      /* enable X2 mode */
        TH1=230;      /* Timer 1 reload value recalculated */
        RCAP2L=0x30;
        RCAP2H=0xF8;  /* Timer 2 autoreloads every 2000 machine cycles */
    }
    else
    {
        /* STANDARD MODE */
        CKCON=0;      /* disable X2 mode */
        TH1=243;      /* Timer 1 reload value recalculated */
        RCAP2L=0x18;
        RCAP2H=0xFC;  /* Timer 2 autoreloads every 1000 machine cycles */
    }
}

/*-----*/
/* the external IT 1 toggles X2_mode */
/*-----*/
void ext_IT_1 (void) interrupt 2 using 3 {
    while (!INT1) {} /* wait for button to be released */
    exit=1;          /* to exit main loop routine */
    x2_mode=~x2_mode; /* toggles x2_mode variable */
}

/*-----*/
/* the timer 2 IT increments Timer variable */
/*-----*/
void timer2_IT (void) interrupt 5 using 3 {
    TF2 = 0;        /* clear IT flag */
    timer++;        /* increment timer variable */
}

/*-----
This routine initializes 'to_sort' bytes in RAM with decreasing values from
'to_sort' to 1
-----*/
void initialize () {
    unsigned char *i;
    unsigned int j;
    i = ram_begin;
    for (j=0; j<to_sort; j++)
    {
        *(i+j) = to_sort-j;
    }
}

```

```

/*-----
The sorting alorythm is :
for x=1 to 'to_sort'-1
  for y= x+1 to 'to_sort'
    if value(y)<value(x) then swap value(x) & value(y)
  next y
next x
/*-----*/

void sort() {
  unsigned int x,y,z ;
  unsigned char *i,*buffer ;

  i = ram_begin-1 ;
  buffer = ram_begin-1 ;      /* pointers initialized */

  for (x=0; x<26; x++)
  {
    printf("\n");              /*-----*/
                                /* effacement d'écran */
  }                             /*-----*/

  printf("Atmel Wireless & MicrocontrollersAtmel Wireless & Microcontrol-
lers\n\n");
  printf("The X2 feature\n\n\n");

  for (z=1; z<=to_print;z++)
  {
    printf("%d  ",(int)*(i+z)) ; /*-----*/
                                /* print 'to_print' first values in initial
order */
  }                             /*-----*/

  printf(" ..... \n");
  for (z=to_sort-to_print; z<=to_sort;z++)
  {
    printf("%d  ",(int)*(i+z)) ; /* print 'to_print' last values in initial
order */
  }                             /*-----*/

  printf("\n\nSORTING");
  (x2_mode?printf(" in X2 mode\n"):printf(" in Standard mode\n"));

  timer = 0;
  TL2   = RCAP2L;
  TH2   = RCAP2H;              /* timer 2 setup */
  T2CON = 0x04;                /* auto reload mode, timer 2 on */

  for (x=1; x<to_sort; x++)
  {
    for (y=0; y<4000; y++)
    {}
    for (y=x+1; y<=to_sort; y++)
    {
      if (*(i+y) < *(i+x) )
      {
        *buffer = *(i+x) ;      /*-----*/
        *(i+x) = *(i+y) ;      /* swaping procedure */
        *(i+y) = *(buffer) ;    /*-----*/
      }
    }
  }

  T2CON = 0;                  /* timer 2 off */
                                /* timer variable contents the number of milliseconds */
                                /* spent to sort */

  printf("\n");
  for (z=1; z<=to_print;z++)

```

```

    {
        printf("%d  ",(int)*(i+z)) ; /*-----*/
order */
    }
    printf(" .....\\n");
    for (z=to_sort-to_print; z<=to_sort;z++)
    {
        printf("%d  ",(int)*(i+z)) ; /* print 'to_print' last values in final
order */
    }
    printf ("\\n\\nElapsed time for sorting : %d milliseconds\\n\\n",timer);
    printf ("Push the INT1 button to toggle the mode or\\n");
    printf ("      X to activate the X2 mode\\n");
    printf ("      any other keyboard key for the standard mode");
}

/*-----
The main C function.  Program execution starts
here after stack initialization.
-----*/
void main (void) {
    unsigned int i;

    /*-----
The following code inserts 3 NOPs if we're using
the RISM monitor for debugging.  This is required
since RISM doesn't like us messing with SCON.
Three NOPs are required because LJMPs are used
to implement breakpoints.
-----*/
#ifdef RISM
    _nop_ ();
    _nop_ ();
    _nop_ ();
#endif

    /*-----
Setup the serial port for 2400 baud at 12MHz in standard mode
-----*/
    CKCON = 0;      /* standard mode */
    SCON = 0x50;    /* SCON: mode 1, 8-bit UART, enable rcvr */
    TMOD |= 0x20;   /* TMOD: timer 1, mode 2, 8-bit reload */
    TH1 = 243;     /* TH1: reload value for 2400 baud @ 12MHz, standard mode */
    TR1 = 1;       /* TR1: timer 1 run */
    TI = 1;        /* TI: set TI to send first char of UART */

    /*-----*/
    /* setup Timer 2 */
    /*-----*/
    RCAP2L= 0x18;
    RCAP2H= 0xFC;   /* auto reload each 1000 cycles */
    TL2 = RCAP2L;
    TH2 = RCAP2H; /* timer 2 setup */

    /*-----
Interrupts Configuration
-----*/
    EX1 = 1;       /* enable ext IT1 */
    ET2 = 1;       /* enable Timer 2 IT */
    IP = 0x20;     /* timer 2 = priority 1, external IT1 = priority 0 */
    EA = 1;        /* enable all ITs */

    /*-----
Note that an embedded program never exits (because
there is no operating system to return to).  It

```

must loop and execute forever.

```
----- */
P1 = 0xC0;          /* leds initialized */
while (1)
{
    initialize ();   /* goto memory initialization routine */
    sort ();         /* goto sorting procedure */
    while (!RI & !exit) /* this loop will be exited if the exit variable
is 1 */
    {
        receive /* or if the micro receives something in the UART
        */
        for (i=0; i<8; i++) /* buffer (key hit from console) */
        {
            if (P1==0xC0) /* this makes LEDs blink nicely... */
            {
                /* . */
                P1=0x81; /* . */
            }
            /* . */
            else /* . */
            {
                /* . */
                P1= 2*P1 - 128; /* . */
            }
            tempo ();
            delay ();
        }
    }
    RI = 0;          /* empty UART receive buffer */
    if (exit)
    { exit=0; }      /* reset exit variable if necessary */
    else
    { x2_mode = (SBUF == 'x'); } /* switch x2_mode variable on if the console
hit key is "x" */
    deal_with_x2 (); /* make X2 change happen */
}
}
```