

# Arithmetic Calculations

- Avoid using **multiplications** and **divisions** whenever possible. These complex mathematic calculations need a lot processing power and (what's even worse) a huge library to be compiled at all.
- If you need to multiply with and divide through even numbers like 2, 4, 8, 16, 32 ... you can make use of the bitshifting operators "»" and "«"

```
char c;
c = 12 >> 1; // c is 6 (division through 2)
c = 12 >> 2; // c is 3 (division through 4)
c = 12 << 1; // c is 24 (multiplication by 2)
c = 12 << 2; // c is 48 (multiplication by 4)
c = 1023 >> 3; // c is 127 (10bit to 7bit ;)
```

- There's an excellent thread in the forum that discusses bitoperations:  
<http://www.midibox.org/forum/index.php?topic=6981.0>
- If this is not enough, you could search for ASM optimized custom functions. You'll find some in code examples of TK, the [ACSensorizer](#) and a lot of PIC-Specialized Webpages - or of course the forum.
- If that still is not enough or you have no time and a lot of processing power / space available on your PIC, you can include the **libsdcc library**:

*if multiplications, divisions, pointer operations, etc. are used in the .c code, the linker may fail due to missing functions, which are part of the libsdcc.lib library. The common library for pic16 derivatives is not compatible to MIOS, therefore I've created a special one which can be downloaded from [here](#). Read the README.txt file for further details. [TK on the C-Page](#)*

## C Functions

- MIOS\_\*\_SRSet and \_SRGet Functions refer to the pins in [Little-Endian](#) order, so for example:

MIOS\_DOUT\_SRSet(1, 00000001) Will set the 1st pin (aka Pin 0).... or  
MIOS\_DOUT\_SRSet(1, 01000000) Will set the 7th pin (aka Pin 6)

## C Optimizations

- [How to mix C and ASM](#)
- [Compiled C Code Size](#)

# C Variables

- Declaring a [variable](#) as type 'const' will cause the [compiler](#) to store the variable in the PIC's program [flash memory](#), not the [SRAM](#).
- Adding the keyword '[volatile](#)' to a [variable](#) is a good idea when this variable can be changed or altered outside the sourcefile that declared this variable.

## SDCC Bugs/Workarounds

Some of these bugs have first been described in a [german thread in the forum](#).

### Array Access

Sometimes the transfer of an array between modules does not work properly, e.g. file 1:

```
unsigned char MIDIValues[8];
```

file 2:

```
MIOS_MIDI_TxBufferPut(MIDIValues[1]);
```

Instead, you need to do something like

```
unsigend char value = MIDIValues[1]; //explicite temp variable  
MIOS_MIDI_TxBufferPut(value);
```

### Large Arrays

Arrays with more than 256 elements will produce compile (in fact linker) errors:

```
unsigned char myArray[256]; // will work  
unsigned char myArray[257]; // will not be linked!  
  
unsigned char myArray[64][4]; // will work
```

```
unsigned char myArray[64][5]; // will not be linked!
```

Thanks to Thomas for [testing some workarounds with multiple single-dimensional arrays](#).

## Bit Copy Operations

There is potential trouble with bit copy operations (See [this posting](#)). Instead of

```
app_flags.SRAM_CARD_STATUS = PORTEbits.RE2;
```

you should use

```
if( PORTEbits.RE2 ){  
    app_flags.SRAM_CARD_STATUS = 1;  
}else{  
    app_flags.SRAM_CARD_STATUS = ;  
}
```

It is less elegant, but it works safely.

## Parenthesis

Always use parenthesis around expressions like

```
myarray[a+b];
```

instead use

```
myarray[(a+b)];
```

## Preprocessor #ifs

Avoid `#ifdef` and `#if` preprocessor-statements wrapped around declarations and function prototypes. Even if the preprocessor's `#if` statement is true (eg defined as '1'), any access to it's vars and functions from outside these wrapped statements produce a compile-warning:

```
#define TEST 1  
  
#if TEST
```

```
    unsigned char testvar;  
#endif /* TEST */  
  
void testfunction(void) {  
    unsigned char c = testvar + 1; // access to testvar produces compiler  
error!  
}
```

## Zero Compare

Avoid comparisons of unsigned char with , e.g.

```
unsigned char i;  
for (i = ; i < ; i++) {  
    //body  
}
```

0 could be a constant that was defined using #define, e.g. the number of motorized faders. But you have no motorized faders... The main problem consists in the fact that your code depends on what else is done around the comparison or in the body. This provokes completely erratic behaviour.

From:  
<http://wiki.midibox.org/> - **MIDIbox**

Permanent link:  
[http://wiki.midibox.org/doku.php?id=c\\_tips\\_and\\_tricks\\_for\\_pic\\_programming&rev=1163512530](http://wiki.midibox.org/doku.php?id=c_tips_and_tricks_for_pic_programming&rev=1163512530)

Last update: **2006/11/19 05:07**

