ZBasic Application Note

AN-102 Migrating from the ZX-24 to the ZX-24a or ZX-24p

Introduction

The recently announced discontinuation of the ZX devices based on the ATmega32 processor (e.g. ZX-24, ZX-40 and ZX-44) will require that some applications be migrated to an alternate device. This application note describes the differences between the ATmega32-based ZX devices and those based on the ATmega644 and ATmega644P processors. Changes to your application's source code that might be required are also described.

Overview of Differences

The table below gives an overview of the differences between ZX devices due to differences in the underlying processor. When migrating from left to right, none of these differences should require ZBasic source code changes in most applications. One exception to this would be the unlikely (and inadvisable) situation where your application uses absolute RAM addresses directly as opposed to using the ZBasic `.DataAddress` property.

	ATmega32	**ATmega644**	**ATmega644P**
ZX Devices	ZX-24, ZX-40, ZX-44	ZX-24a, ZX-40a, ZX-44a	ZX-24p, ZX-40p, ZX-44p
User RAM	1.5K bytes	3.5K bytes	3.5K bytes
User RAM Start Address	&H00a0 (160)	&H0140 (320)	&H0140 (320)
Persistent Memory	992 bytes	2016 bytes	2016 bytes
External Interrupts	INT0, INT1, INT2[1]	INT0, INT1, INT2	INT0, INT1, INT2
Pin Change Interrupts	None	All I/O lines	All I/O lines
Hardware UARTs	Com1	Com1	Com1, Com2

Notes: 1) For INT2, only rising edge and falling edge modes are available on the ATmega32.

Another type of change between the ATmega32 and the ATmega644/ATmega644P processors is that the addresses of some of the I/O registers, their names and/or the bits within them have changed. Unless your application refers to registers using an absolute address (unlikely and inadvisable), the address changes are of no consequence since the ZBasic compiler will handle that automatically. The name changes and bit assignments within the registers will require some attention if your application uses direct I/O register access. The table below enumerates the register names that have changed. Note that the bits in some registers have been split out to multiple registers so you'll have to consult the processor datasheet to determine which register name you should use. The entries with grey background are unlikely to be used in your application but are included for completeness.

Register Name Changes

ATmega32	**ATmega644 or ATmega644P**
GIFR	EIFR
GIMSK	EIMSK
MCUCR	SMCR or EICRA
MCUCSR	MCUSR
OCR0	OCR0A
OCR2	OCR2A
SFIOR	ADCSRB
SPMCR	SPMCSR
TCCR0	TCCR0A
TIMSK	TIMSK0, TIMSK1, TIMSK2
TIFR	TIFR0, TIFR1, TIFR2

Published April 2008

Migration Procedure

Step 1) If your application does not refer to absolute memory addresses and does not refer to any I/O registers either by name or by address, you should be able to simply recompile for the new target device. No other source code changes should be required.

Step 2) If your application directly manipulates any I/O registers whose names have changed (listed in the table above), make the appropriate changes consulting the processor datasheet as necessary. It is highly recommended to use the conditional compilation capability of the ZBasic compiler to retain the original code while adding new code to handle the new target processor. An example of this is shown below. Note the use of the `#error` directive to automatically notify you in the future if you retarget for a different underlying processor.

```
' define the value of the analog comparator multiplexer enable bit
 #if Option.CPUType = "mega32"
Private Const ACME_BIT as Byte = &H08
#elseif Option.CPUType = "mega644" Or Option.CPUType = "mega644p"
Private Const ACME_BIT as Byte = &H40
#else
#error Need constant definition for target CPU
#endif

' enable the analog comparator multiplexer
 #if Option.CPUType = "mega32"
   Call SetBits(Register.SFIOR, ACME_BIT, ACME_BIT)
#elseif Option.CPUType = "mega644" Or Option.CPUType = "mega644p"
   Call SetBits(Register.ADCSRB, ACME_BIT, ACME_BIT)
#else
#error Need code for target CPU
#endif
```

Step 3) If your application directly manipulates any I/O registers, e.g. Register.MCUCR, you must consult the datasheet for the underlying processor to determine if the bits of interest are in the same or different positions in the new target processor. It is highly recommended to use ZBasic constants for setting bits and masking a result (as shown in the example above) rather than using explicit values. Not only does this have the potential to make the intent of the code more clear, it also makes it easier to re-target your application by using the conditional compilation capability of the ZBasic compiler.

Step 4) If your application would benefit by having larger queues or buffers and the size of those elements was necessarily limited in your mega32-based application, consider increasing the sizes and testing the effect on the application's performance.

Author

Don Kinzer is the founder and CEO of Elba Corporation. He has extensive experience in both hardware and software aspects of microprocessors, microcontrollers and general-purpose computers. Don can be contacted via email at dkinzer@zbasic.net.

e-mail: support@zbasic.net **Web Site:** http://www.zbasic.net