Application Note

Implementing Mixed Memory Modes on the eZ80 CPU

AN033902-0512

Abstract

This application note describes the two types of memory modes offered by the eZ80 CPU and provides a simple application to demonstrate how to properly configure the eZ80 CPU for a *mixed memory mode* of operation. As a result, you will be able to run blocks of legacy Z80 code from within the eZ80 CPU environment.

Note: The source code file associated with this application note, AN0339-SC01.zip, is available free for download on zilog.com. This source code has been tested with version 5.2.0 of ZDSII for eZ80Acclaim! MCUs. Subsequent releases of ZDSII may require you to modify the code supplied with this application note.

Discussion

The eZ80 CPU offers two types of memory modes: Z80 Mode and Address Data Long (ADL) Mode. Z80 Mode offers backward compatibility with legacy Z80 applications, while ADL Mode offers 24-bit linear addressing and 24-bit CPU registers. These multiple memory modes of the eZ80 processor allow developers to easily migrate older Z80 or Z180 codes to the new ADL Mode codes.

The eZ80 CPU's architecture is briefly discussed in this document to highlight the differences in each memory mode. For a more detailed discussion, however, please refer to the eZ80 CPU User Manual (UM0077).

An Overview of the eZ80 CPU Architecture

The eZ80 CPU is an 8-bit microcontroller that performs either 16- or 24-bit operations. In the block diagram shown in Figure 1, the separation of the data and control blocks is necessary toward understanding the eZ80 CPU's Z80 and ADL memory modes.

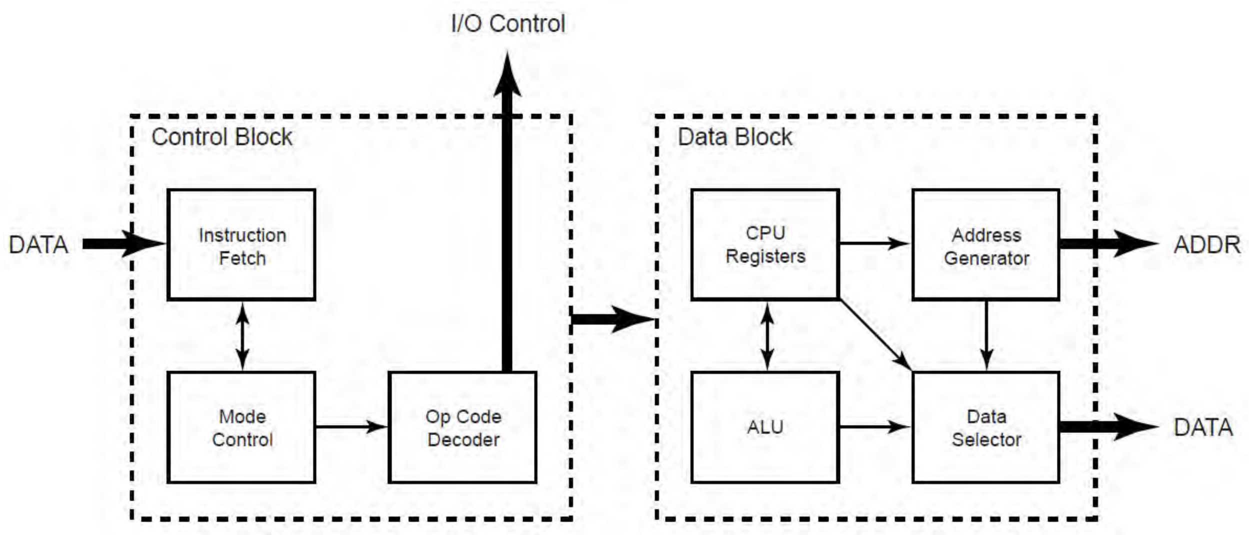


Figure 1. eZ80 CPU Block Diagram

The control block is responsible for handling the overall operation of the CPU. Generally, it acquires which code to execute and manages how data moves in and out of the CPU. It consists of the instruction fetch block, the mode control block and the op code decoder. The instruction fetch block controls the reading of op codes from memory, as well as discarding prefetched instructions when any control transfer events occur. The mode control block administers the current processor mode (halt, sleep, interrupt, debug and ADL modes). The op code decoder block is where op codes are interpreted.

The data block acts as CPU memory. The CPU registers stores relevant values, plus it stores the program counter, stack pointer and flags. The arithmetic logic unit (ALU) performs arithmetic and logic operations on the addresses and data passed over from the control block. The address generator creates addresses for all CPU memory read and write operations; it also contains the MBASE registers for handling address translations in Z80 Mode operation. The data selector controls the data path based on the instruction currently being executed.

Memory Modes

The eZ80 CPU is capable of operating in two modes: Z80 Mode and ADL Mode. Z80 Mode is provided to maintain backward compatibility with legacy Z80 codes, and uses 16-bit linear addressing and 16-bit CPU registers. ADL Mode is offered to provide code execution time that is four times faster than that of a standard Z80 CPU operating at the same clock speed. ADL Mode uses 24-bit linear addressing and 24-bit CPU registers.

Z80 Memory Mode

The Z80 CPU’s memory mode is sometimes referred to as a *non-ADL mode*. It has Z80-compatible addressing and CPU registers. Upon reset, Z80 Mode is the default operating mode of the eZ80 CPU. When operating in Z80 Mode, all registers are 16 bits, including the stack pointer and the program counter. In Z80 Mode, the 8-bit MBASE address register is always appended at the start of the 16-bit address to allow the Z80 code to access any code located within the 16MB address space of the eZ80 CPU. Also, using this kind of placement allows 256 unique code blocks within the 16MB address space. When MBASE is set to zero, the eZ80 CPU operates in similar fashion to a classic Z80 CPU, with 16-bit addressing from `0000h` to `FFFFh`. When set to another value, the 16-bit Z80-style addresses are offset to another page defined by MBASE. By maximizing MBASE functionality, multiple Z80 tasks can be placed in their own individual Z80 partitions.

Address Data Long Memory Mode

In Address Data Long (ADL) Mode, all addresses and data are 24 bits in length, including the stack pointer and the program counter. When in ADL Mode, the user application can take advantage of the CPU's 16MB linear addressing space, 24-bit CPU registers and enhanced instruction set. All read/write operations pass 3 bytes of data to/from the CPU. Thus, instructions operating in ADL Mode can require more clock cycles to complete than in Z80 Mode. As opposed to Z80 Mode, ADL Mode does not employ memory partitions; thus no pages are required.

To further illustrate these two memory modes, the sections that follow discuss how the registers and the memory addressing operations function when both memory modes operate concurrently.

CPU Registers

The eZ80 CPU's registers can be grouped into two types: working registers and control registers. The working registers can generally be used by the main application to store data for processing, while the control registers are used by the CPU to control CPU operations.

eZ80 CPU Working Registers

The BC, DE and HL registers are 24 bits long in ADL Mode; the most significant byte is denoted with a U to indicate the upper byte. These registers become 16 bits long when in Z80 Mode, making the upper bytes (denoted with a U) inaccessible. Figure 2 shows how the eZ80 CPU's working registers differ in Z80 and ADL modes.

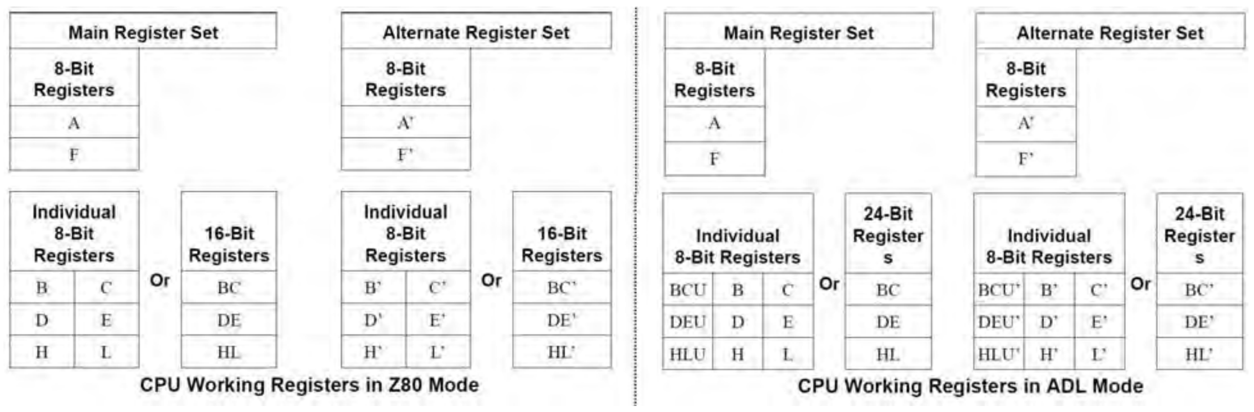


Figure 2. Z80 and ADL Mode Working Registers

The accumulator (A) and flag (F) registers each remain 8 bits long regardless of the CPU's memory mode.

eZ80 CPU Control Registers

The program counter (PC) is 24 bits long in ADL Mode; it becomes 16 bits long when in Z80 Mode. To access memory while operating in Z80 Mode, the MBASE is used by the CPU in conjunction with the program counter, resulting in a 24-bit address compatible with the eZ80 CPU's memory addressing denoted by {MBASE, PC[15:0]}. The MBASE Register is not used for address generation in ADL Mode, but this register can only be modified while in ADL Mode. This register becomes useful when in Z80 Mode.

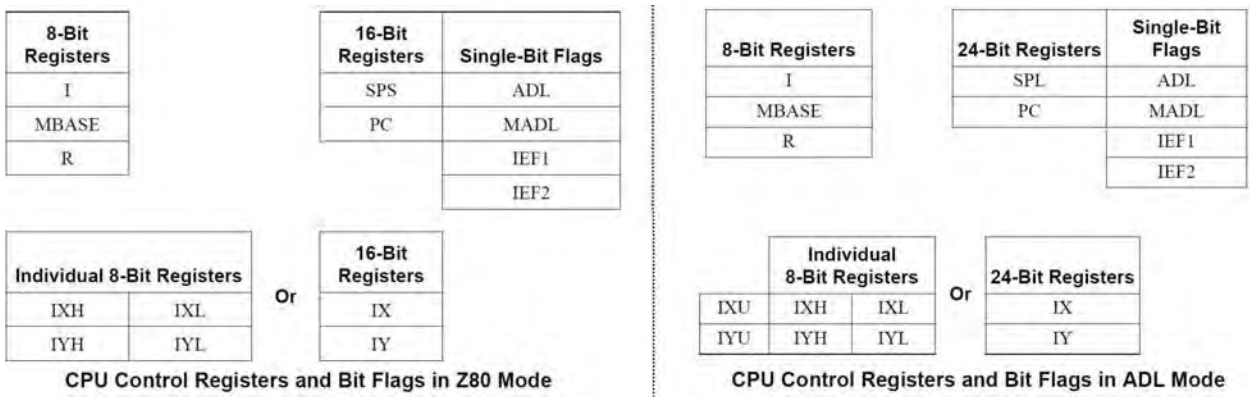


Figure 3. Z80 and ADL Mode Control Registers

In ADL Mode, the active stack pointer is the 24-bit Stack Pointer Long (SPL), which becomes inactive when memory mode switches to Z80 Mode, where the active stack pointer is the 16-bit Stack Pointer Short (SPS). Similar to the program counter, the SPS can be allocated anywhere within memory and can be accessed using the MBASE Register, effectively yielding the address {MBASE, SPS}.

The IX and IY registers, much like the working registers, are 24 bits long in ADL Mode, with the most significant byte denoted by U. In Z80 Mode, the most significant byte becomes inaccessible; these registers become 16 bits long.

The Address Data Long (ADL) bit indicates the current memory mode of the CPU. If this bit is reset to 0, the CPU will operate in Z80 Mode; if it is set to 1, this bit indicates that the CPU is running in ADL Mode. Upon reset, this bit is cleared to 0, which means that the default memory mode of the CPU is Z80 Mode. The ADL bit cannot be modified directly. To change the value of this bit, the user application must execute special control instructions such as CALL, JP, RST, RET, RETI and RETN.

The Mixed ADL (MADL) bit is used to configure the CPU to execute programs containing code that uses both ADL and Z80 memory modes. This bit can be set to 1 using the STMIX instruction, and can be cleared to 0 using the RSMIX instruction.

Memory Map

In ADL Mode, the user application can maximize use of the CPU's 16-bit linear addressing space, yielding a valid range of addresses from `000000h` to `FFFFFFh`, as shown in the right half of Figure 4.

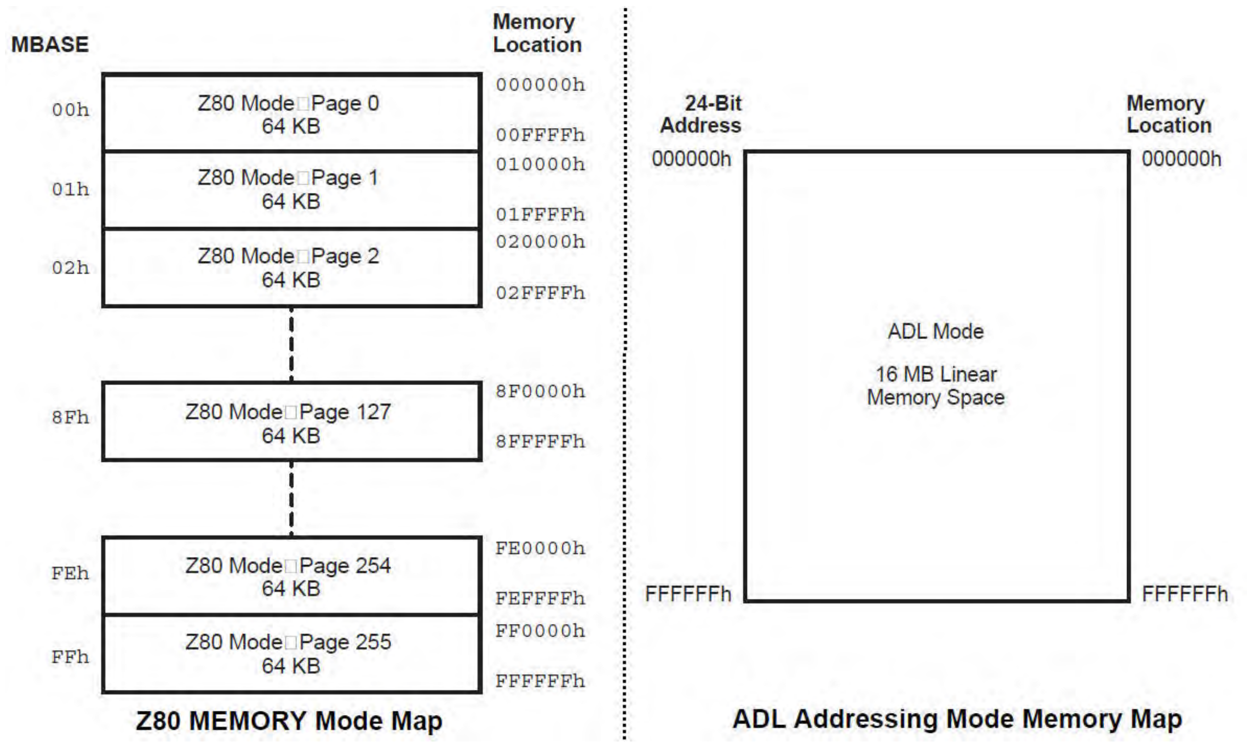


Figure 4. Z80 and ADL Mode Memory Map

In Z80 Mode, however, the eZ80 CPU employs a 16-bit linear addressing space. Because of this difference, memory access is limited only to the valid address space, which ranges from `0000h` to `FFFFh`. Nevertheless, the eZ80 CPU employs an 8-bit MBASE address register, which can be used to enable access to other areas of memory. The MBASE Address Register is always prepended to the 16-bit Z80 Mode address, thereby effectively producing a 24-bit address. This strategy also breaks the eZ80 CPU's memory into 256 pages, allowing for 256 unique Z80 code blocks to be placed anywhere within the 16MB addressing space. The left side of Figure 4 shows the memory map for Z80 Mode.

Memory Mode Switching

The eZ80 CPU offers two types of mode changes: Persistent and Single Instruction. Persistent Mode switches allow the CPU to operate indefinitely in ADL Mode, then switch to Z80 Mode to run a section of Z80 code, and then return to ADL Mode. This methodology is often used when executing a block of Z80 code from within a higher-level ADL Mode program. The CPU can only make Persistent Mode switches between the ADL and Z80 modes as part of such special control transfer instructions as CALL, JP, RST, RET, RETI and RETN, or as part of an interrupt or trap operation.

Single Instruction Mode changes allow certain instructions to operate using either addressing mode without making a persistent change to the mode. This methodology is only useful when the CPU must perform a single operation using the memory mode not currently set.

The sections that follow discuss some of the important instructions and compiler directives required by the CPU to properly execute blocks of Z80 code using the eZ80 CPU. For a more thorough understanding, however, please refer to the eZ80 CPU User Manual (UM0077) and the Zilog Developer Studio II – eZ80Acclaim! User Manual (UM0144).

The .ASSUME Compiler Directive

In the ZDS II assembler, the application code is assembled for a given state of the ADL Mode bit. This assembly is performed by issuing the `.ASSUME` compiler directive at the top of the code. This directive tells the compiler that either ADL or Z80 Mode is the memory mode that must be used for the code that is currently being compiled. The `.ASSUME` directive becomes effective from the moment the compiler encounters this directive until it encounters the next `.ASSUME` directive that changes the current memory mode, or until it encounters an end of file statement.

To set the current set of instructions to ADL Mode, place the following compiler directive at the top of the code bock:

```
.ASSUME ADL=1     ; switch to ADL Mode
```

To set the current set of instructions to Z80 Mode, place the following compiler directive at the top of the code block:

```
.ASSUME ADL=0     ; switch to Z80 Mode
```

Caution: The code developer is responsible for ensuring that this source file setting matches the state of the ADL Mode bit when the code is executed.

Special Op Code Suffixes for Memory Mode Control

In addition to the compiler directives, special op code suffixes are added to the instruction set to assist with memory mode switching operations. The code developer is required to use these suffixes to allow exceptions to the default memory mode. This requirement is important toward correctly facilitating control and data block cross-references, and can be very useful when an ADL Mode code must fetch a 16-bit address generated from a section of Z80 Mode code (and vice versa).

The four individual suffixes that can be used are `.SIS`, `.SIL`, `.LIS`, and `.LIL`. These suffixes can be appended to a number of instructions to indicate that a memory mode change or an exception to standard memory mode operation is being requested. Table 1 summarizes these op code suffixes necessary for memory mode switching operations.

Table 1. Op Code Suffixes

	S	L
IS	**.SIS** The CPU data block operates in Z80 Mode using 16-bit registers. All addresses use MBASE. The CPU control block operates in Z80 Mode. Indicates that only 2 bytes of immediate data or address must be fetched.	**.LIS** The CPU data block operates in ADL Mode using 24-bit registers. Addresses do not use MBASE. The CPU control block operates in Z80 Mode. Indicates that only 2 bytes of immediate data or address must be fetched.
IL	**.SIL** The CPU data block operates in Z80 Mode using 16-bit registers. All addresses use MBASE. The CPU control block operates in ADL Mode. Indicates that 3 bytes of immediate data or address must be fetched.	**.LIL** The CPU data block operates in ADL Mode using 24-bit registers. Addresses do not use MBASE. The CPU control block operates in ADL Mode. Indicates that 3 bytes of immediate data or address must be fetched.

As a general rule, the suffixes for memory mode control are composed of two parts that define the operation in the control block and the data block within the CPU.

Short or Long (.S/.L). Directs operations within the data block of the CPU. These suffixes control whether the overall operation of the instruction and the internal registers should use 16 or 24 bits. It also indicates if MBASE is used to define the 24-bit address.

Instruction Stream Short or Instruction Stream Long (.IS/.IL). Directs the control block within the CPU. These suffixes control whether a multibyte immediate data or address value fetched during instruction execution is 2 or 3 bytes long. In short, these suffixes indicate the length of the instruction to fetch to the CPU. If the length of the instruction is unambiguous, these suffixes yield no effect.

Guidelines for Implementing Mixed Memory Mode Applications

When programming mixed memory mode applications, it is important to take note of the default values of certain registers and flags, as the following points show.

- The CPU's default memory mode is Z80 Mode (ADL=0). The compiler directive `.ASSUME ADL=0/1` will change the CPU's current memory mode.
- The CPU's default mixed ADL Mode control bit is disabled (MADL=0). The RSMIX instruction disables mixed memory mode, while the STMIX instruction enables the CPU to run in mixed memory mode.
- Because of the first point above, the program can run exclusively in ADL Mode. However, the code can be set to permanently run in ADL Mode at startup. If a single `JP.LIL` instruction is used at or near the beginning of the code to permanently change to ADL Mode, the program is considered to operate exclusively in ADL Mode.

To be able to run legacy Z80 code on the eZ80 CPU, the eZ80 CPU must be configured to run in mixed memory mode. To configure the eZ80 CPU to run in mixed memory mode, the following procedure must be observed.

1. Execute an STMIX instruction at device initialization to ensure that interrupt service routines will begin in a consistent memory mode.
2. All interrupt service routines and code blocks that can be called from either Z80 Mode or ADL Mode must end with a `RETI.L` or `RETN.L` instruction to ensure that the interrupted code's memory is popped from the SPL stack correctly (and not the SPS stack).
3. When calling routines or code blocks, make sure to use a suffixed CALL instruction to force the CPU to access each code block in the correct memory mode into which it was assembled or compiled. This type of instruction is also important to allow the CPU to save the calling code's memory mode on the SPL stack. A suffixed `JP` instruction can also be used; however, a suffixed CALL instruction is recommended to avoid run-time errors caused by wrong assumptions as to which memory mode the CPU is currently running. The CALL instruction promotes modularity of the application code; i.e., the code developer can better monitor the memory mode for each block of code.
4. If a calling code operating in one mode must pass stack-based operands/arguments to a routine compiled or assembled for a different mode, it must use suffixed instructions to set up the operands/arguments. For PUSH, the `.S` and `.L` suffixes control whether SPS or SPL is used, and whether the operands/arguments are stored as 2- or 3-byte values.

Software Implementation

This application note provides a very simple UART application to demonstrate how to configure the eZ80F91 MCU for mixed memory mode applications. The application prompts the user to enter characters into HyperTerminal, and it will display these characters for the user in HyperTerminal. The sequence of this application is charted in Appendix A. Flowcharts on page 17.

The software is divided into two eminent blocks: the main application and the UART routines. The main application is compiled using ADL Mode, while the UART routines are compiled using Z80 Mode. At startup, the UART routines are copied to RAM so that when the CPU's control transfers to Z80 Mode, the program counter {MBASE, PC[15:0]} and the stack pointer {MBASE, SPS} will be located in one memory page. Figure 5 shows how the application is allocated within eZ80F91 memory.

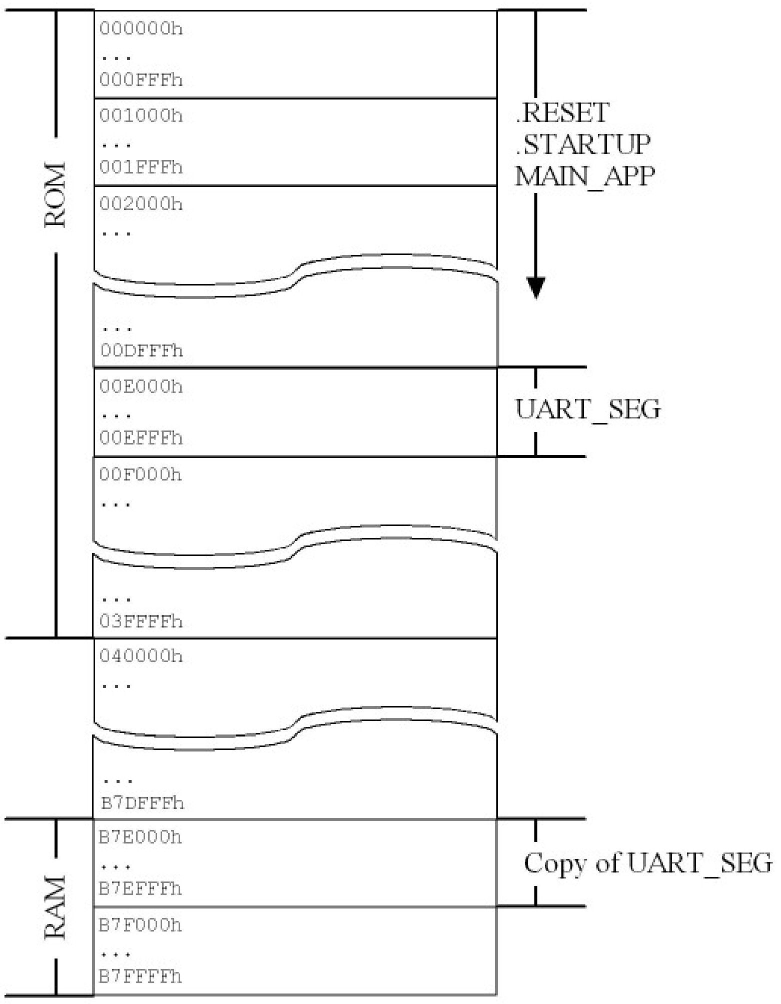


Figure 5. Application Memory Allocation in the eZ80F91 MCU

The memory setup, linker directives and start-up code (discussed below) result in the memory allocation for the UART segment in RAM shown in Figure 6.

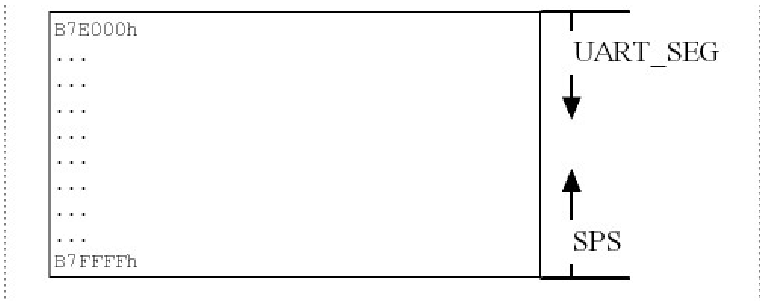


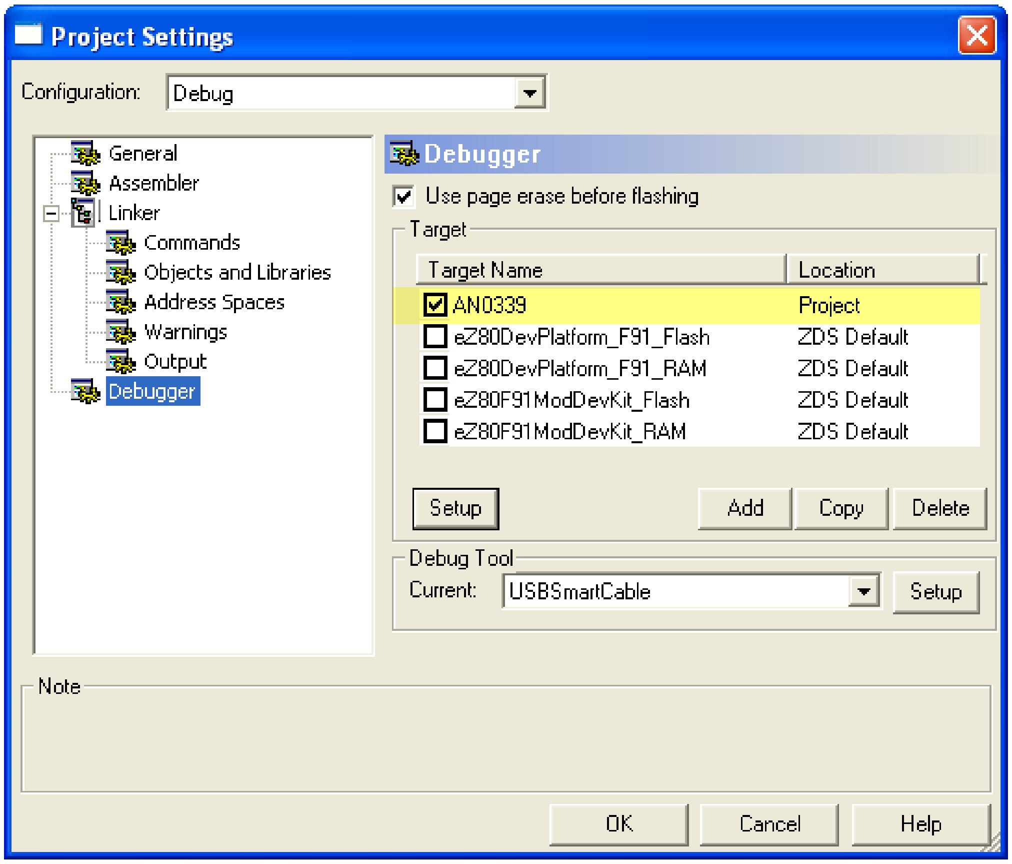
Figure 6. UART Segment in RAM

Memory Setup

Prior to developing an application, it is important that the code developer should have an idea of how the memory looks like. This application uses the eZ80F91 MCU's 256KB internal Flash and 8KB general-purpose RAM memories. Essentially, Flash is mapped to addresses in the range `000000h` to `003FFFh`, and RAM is mapped to addresses in the range `B7E000h` to `B7FFFFh`. External RAM and Flash are disabled for this application.

The ADL stack pointer (SPL) starts at `B80000h`, and the Z80 stack pointer (SPS) starts at `F000h`. The program counter (PC) remains set to its default value at address `000000h`.

The source code (AN0339-SC01.zip) includes a custom target file that includes these settings; it can be used with ZDS II by navigating via the ZDS II menu to **Project Settings** → **Debugger** → **Target Name** and selecting **AN0339**, as shown in Figure 7.


Project Settings

Configuration: Debug

General
Assembler
Linker
Commands
Objects and Libraries
Address Spaces
Warnings
Output
Debugger

Debugger

[✓] Use page erase before flashing

Target

Target Name	Location
[✓] AN0339	Project
[ ] eZ80DevPlatform_F91_Flash	ZDS Default
[ ] eZ80DevPlatform_F91_RAM	ZDS Default
[ ] eZ80F91ModDevKit_Flash	ZDS Default
[ ] eZ80F91ModDevKit_RAM	ZDS Default

Setup Add Copy Delete

Debug Tool
Current: USBSmartCable Setup

Note

OK Cancel Help

Figure 7. Selecting Target Settings

Additional Linker Directives

This section discusses linker directives and their significance in the overall functionality of the application. For more information about linker directives, please refer to the Zilog Developer Studio II – eZ80Acclaim! User Manual (UM0144).

The following linker directive ensures that the `.RESET` code segment will be the first code segment to be allocated in memory, followed by the `.STARTUP` code segment and the `MAIN_APP` code segment:

```
ORDER .RESET,.STARTUP,MAIN_APP
```

The sequence of this directive ensures that the application will start at the `.RESET` segment, which is the segment that initiates setting up of the eZ80F91 MCU memory and peripherals.

The next directive, below, creates a user-defined symbol named `__stack` to be used by the `.STARTUP` code when initializing the stack pointer:

```
DEFINE __stack = highaddr of RAM+1
```

Although the stack pointer can be set directly from the user code, it is advisable to define it through the linker to provide flexibility when increasing RAM space or when changing RAM addresses.

The following three linker directives create user-defined symbols to be used by the `.STARTUP` code when it copies the UART segment from ROM to RAM:

```
DEFINE __low_ramuart = base of RAM
DEFINE __len_uart = length of UART_SEG
DEFINE __low_romuart = base of UART_SEG
```

The `__low_ramuart` points to the base of the RAM address which, in this application, points to the address `B7E000h`. The `__low_romuart` points to the base of the UART segment in ROM which is defined in this application to be located at address `00E000h`. The `__len_uart` symbol simply identifies the actual length of the code occupied by the UART segment.

The Start-Up Code

The start-up code includes basic initialization of the eZ80F91 MCU and the copying of the UART segment from ROM to RAM. The following code segment shows how to copy the UART segment to RAM:

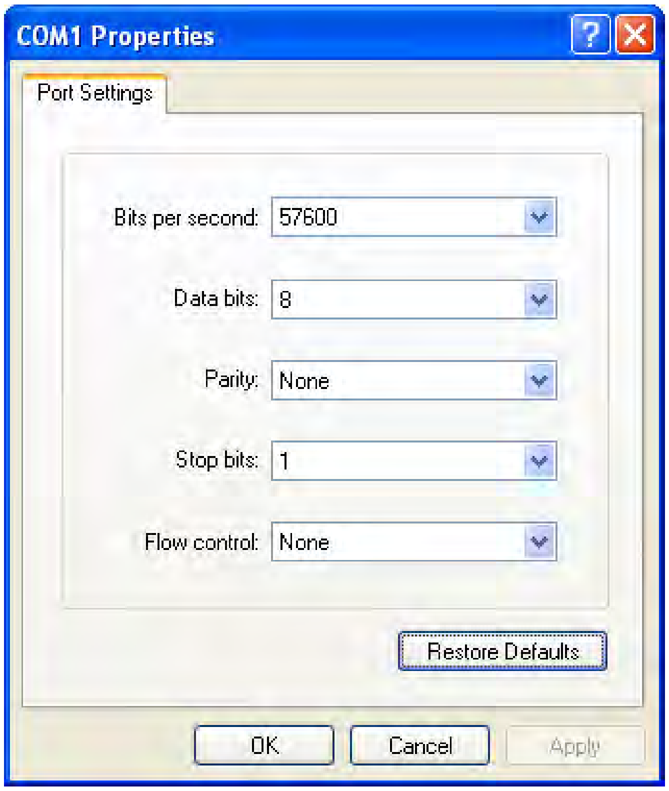
```
ld hl, __low_romuart    ; setup register values required by the
 ld de, __low_ramuart   ; ldir instruction
 ld bc, __len_uart
 ldir                   ; initiate a copy instruction
```

Prior to calling the routines included in the UART segment, the MBASE Register must be updated to reflect the memory allocation shown in Figure 6 on page 10.

Running the Application

With the eZ80F91 Modular Development Kit already loaded with the source code provided with this document (AN0339-SC01.zip), observe the following steps to run the application.

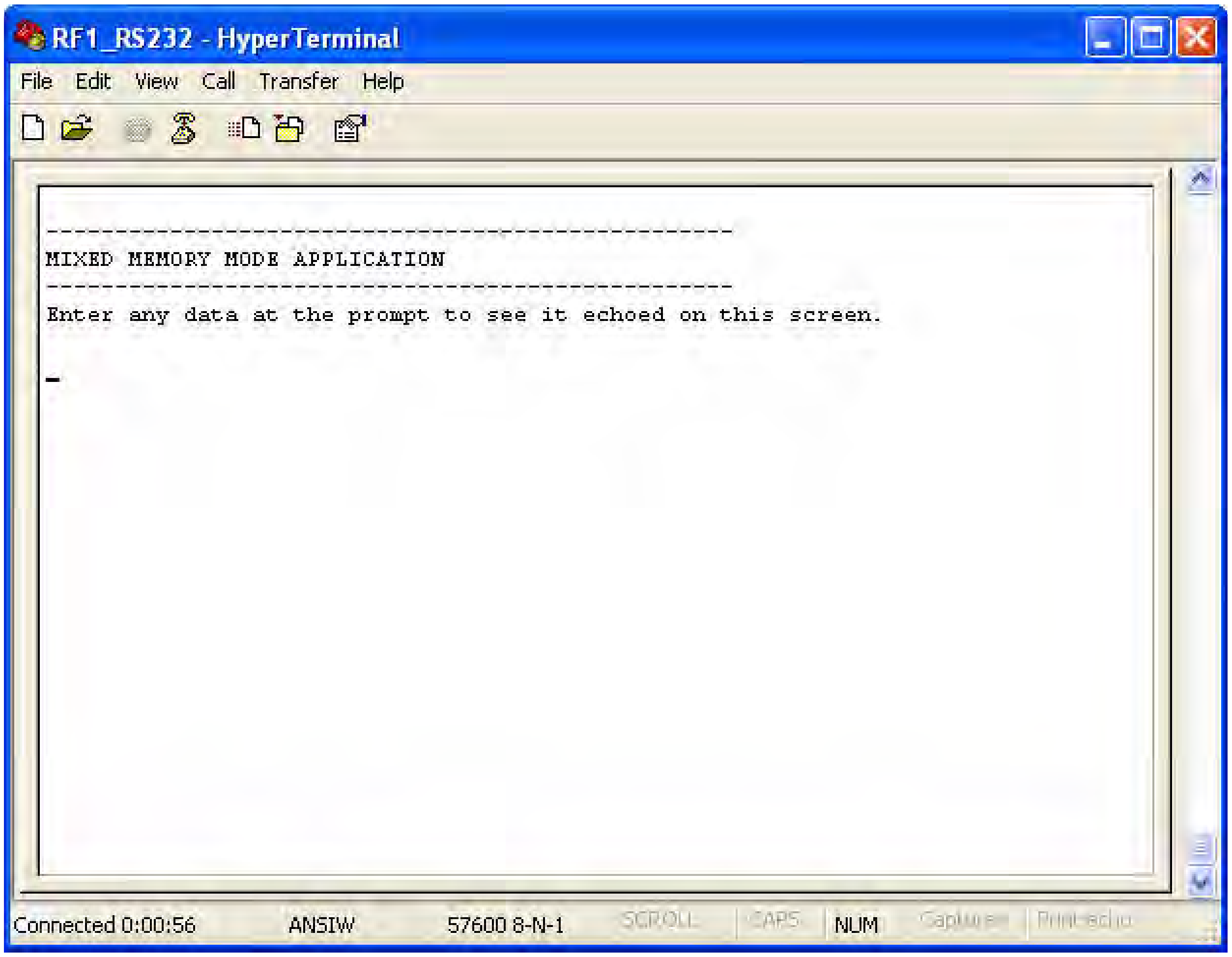
1. Connect the serial port interface (P2) of the eZ80F91 Modular Development Kit to the PC using a serial port cable or a USB-to-Serial cable.
2. On the PC, open the HyperTerminal emulation program and configure it to reflect the settings shown in Figure 8.



Bits per second: 57600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

Figure 8. HyperTerminal Settings

3. Apply power to the eZ80F91 Modular Development Kit. A welcome message should appear in the HyperTerminal, as shown in Figure 9.


RF1_RS232 - HyperTerminal
File Edit View Call Transfer Help

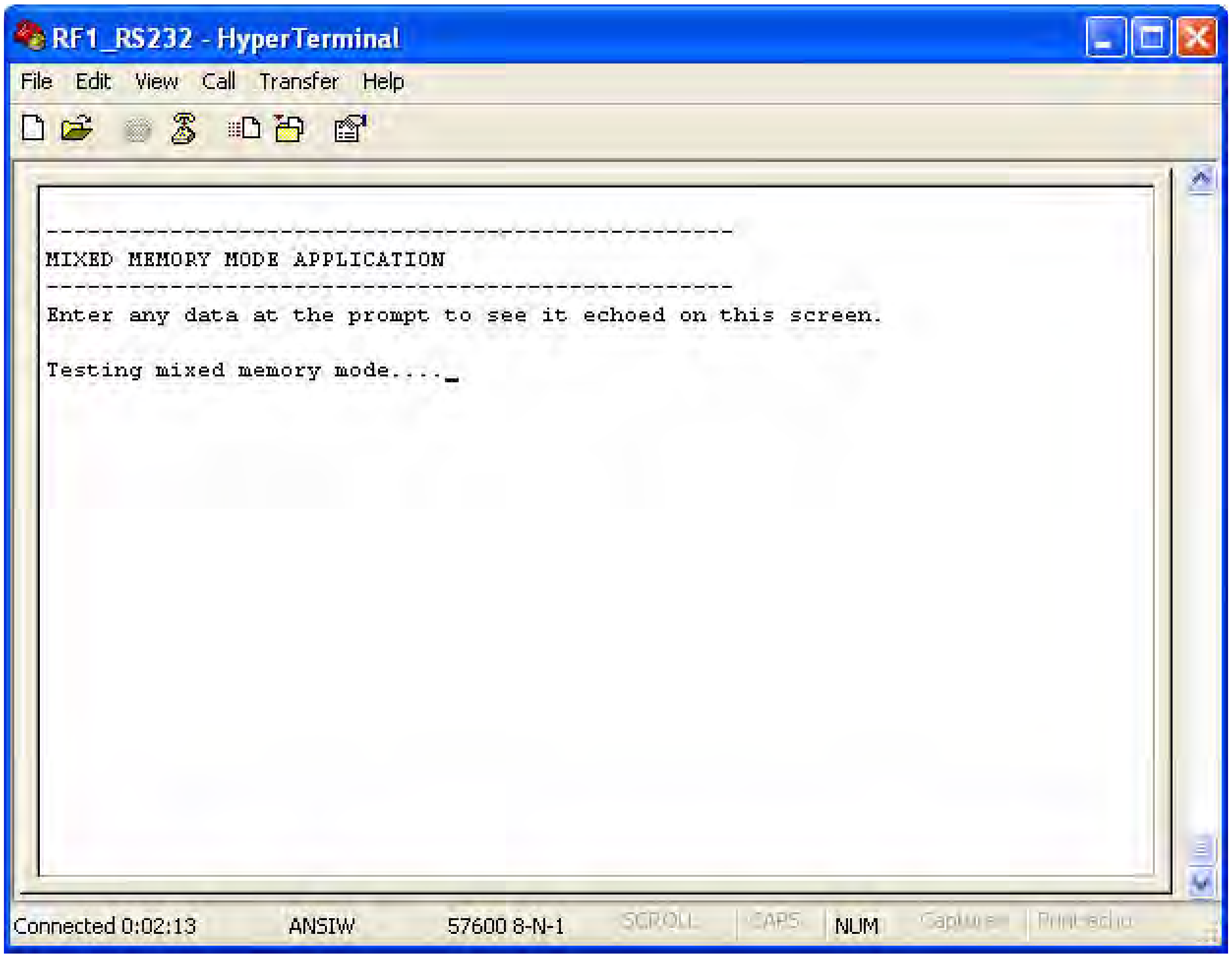
```
-------------------------------------------------
MIXED MEMORY MODE APPLICATION
-------------------------------------------------
Enter any data at the prompt to see it echoed on this screen.

_
```

Connected 0:00:56 ANSIW 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

Figure 9. Welcome Message

4. With your keyboard, enter any character string into HyperTerminal. The verbatim string should display in (i.e., be echoed back to) the HyperTerminal window, as indicated in Figure 10.


RF1_RS232 - HyperTerminal
File Edit View Call Transfer Help
Connected 0:02:13 ANSIW 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

```
--------------------------------------------------
MIXED MEMORY MODE APPLICATION
--------------------------------------------------
Enter any data at the prompt to see it echoed on this screen.

Testing mixed memory mode...._
```

Figure 10. User Input in HyperTerminal

Note: If you see two instances of each character that you entered, navigate via the HyperTerminal menu to **File** → **Properties** → **Settings** → **ASCII Setup**, and ensure that the selection labeled **Echo typed characters locally** is unchecked.

Result

With the main application code configured to run in ADL Mode and the UART code configured to run in Z80 Mode, the user-entered characters are echoed back on screen to demonstrate that the UART code is properly accessed by the main application code.

Summary

This application note demonstrates the proper configuration of Zilog's eZ80-based MCUs to easily integrate older Z80-based code blocks in these newer eZ80 devices. The demo serves as a guide to users who wish to maintain legacy Z80 codes while developing new applications with the eZ80 CPU.

References

The documents associated with the eZ80F91 MCU available on www.zilog.com are provided below:

- eZ80 CPU User Manual (UM0077)
- eZ80F91 ASSP Product Specification (PS0270)
- eZ80F91 Modular Development Kit User Manual (UM0170)
- Zilog Developer Studio II – eZ80Acclaim! User Manual (UM0144)

Appendix A. Flowcharts

Figure 11 shows the flow of the main application.

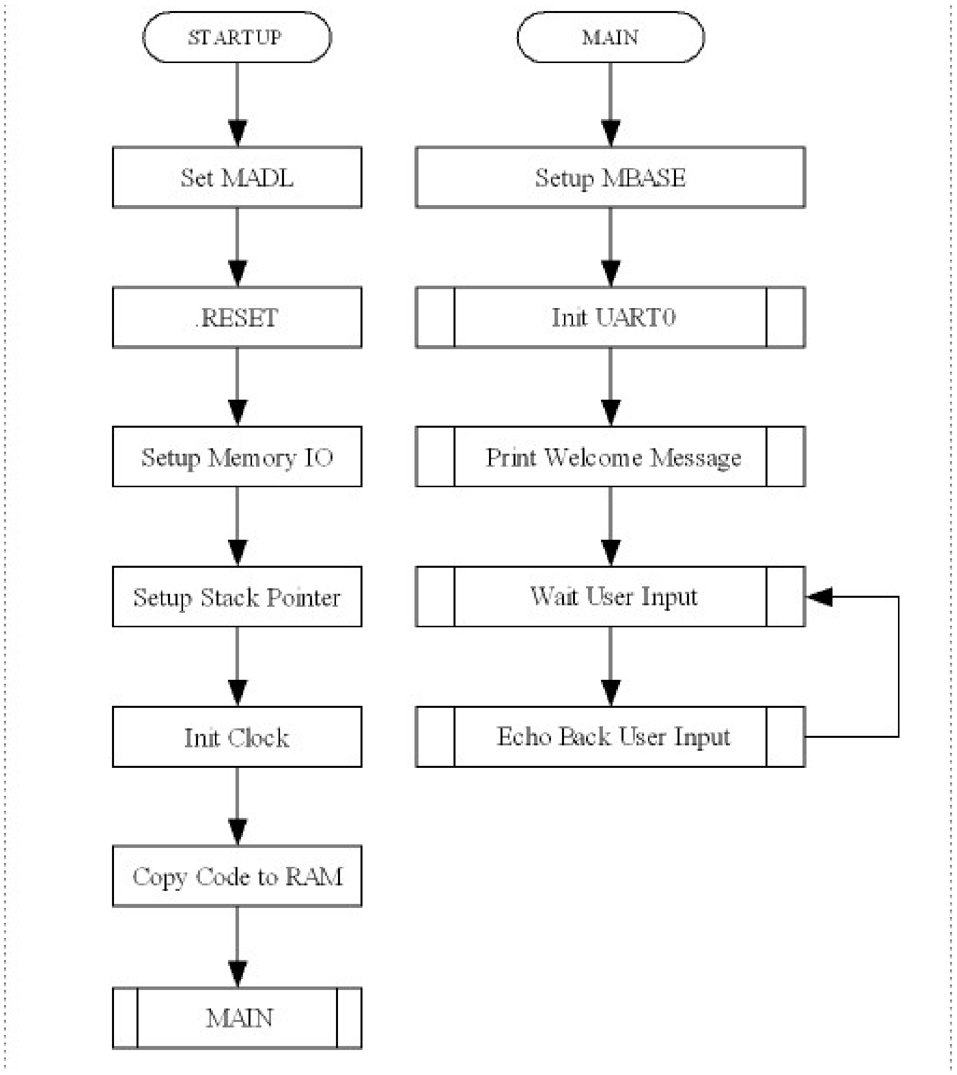


Figure 11. Application Flowchart

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.