White paper: Unlimited code and data support for the ZiLOG® Z80 & Z180 family of microprocessors.

Softools' development tools provide software engineers and programmers innovative and seemless code and data support for program development. These tools, along with ZiLOG's high performance Z180 processors, extend the life of these 8-bit processors.

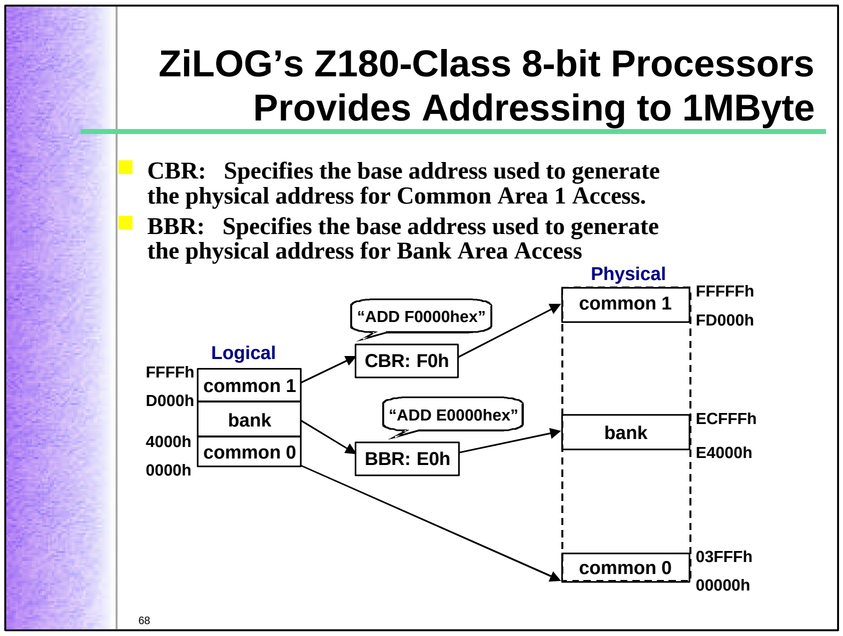
ZiLOG's Z180 Family Peripherals and Processor Performance a Platform for Users to Innovate

The Z180 is ZiLOG's second generation Z80 based processor family. Building on its world famous Z80, ZiLOG's Z180 offers several feature and improvement that have made it an attractive platform for those who require higher CPU performance as well as peripheral integration. The Z180 CPU executes Z80 more efficiently, resulting in faster code throughput compared to a Z80 based system operating at the same speed. In addition, the Z180 family integrate a number of peripherals including high speed communication ports. One of the more used peripherals, however, is the Memory Management Unit, or MMU. This peripheral allow the Z180 family to address up to 1 Mbyte of code through a method called "code banking" or "memory paging"

Introduction to Code Banking

Code banking, or memory paging, is not a new concept and has been used for decades in many hardware and software systems. Hardware systems typically switch ROM or RAM pages or regions to map in various parts of code or data normally inaccessible. Software systems often paged memory by copying parts of program or data from one area or medium to a common paging area. This method is very prominent in the Windows® operating system.

In 8-bit microcomputer designs, external hardware is often used to bank switch various RAM and ROM regions into a fixed memory address. This allows code or data to be more quickly accessed by the CPU and to extend the effective addressing space of the microprocessor. This common address space is usually fixed and referenced using logical addresses and is often called the page or bank window. Many ZiLOG® **Z80**-based systems have been doing hardware paging or banking for over 20 years.



ZiLOG's Z180-Class 8-bit Processors Provides Addressing to 1MByte

- **CBR: Specifies the base address used to generate the physical address for Common Area 1 Access.**
- **BBR: Specifies the base address used to generate the physical address for Bank Area Access**

The ZiLOG® **Z180** family of microprocessors includes a **MMU** (Memory Management Unit) which provides on-chip hardware memory mapping to map from 16-bit logical addresses to 20- or 23-bit physical addresses. The **MMU** allows for the programming of up to three different banking windows, of which two can be mapped anywhere in the 1 megabyte address space with 4 kilobyte granularity (1kilobyte in *extended* **MMU** mode).

Design Considerations with Banking

While banking offers expanded addressing, it also requires special considerations Paramount to these consideration is managing the dynamic paging from the window.. This issue is magnified because programs are getting larger, older, and therefore more complex. The programming shift in the past 5 to 10 years en masse from assembly language to **C** language has made banked systems even more cumbersome, as some Z180 compiler vendors do not support seemless MMU integration..

Many schemes have been developed over the years. Typical ones have included:

⇒ Using hand built tables of addresses. All references throughout a program to a function in another bank had to be replaced by the name in the table. Sometimes clever use of assembler or **C** macros could be used to semi-automate the process but it is still a chore. Each banked function needs a handler to change the bank and to switch to and from the address in the bank. Often overlaid segments were used to get the logical address in the bank set to the right address. Every function had to be manually handled.

⇒ Using a special function that takes one or more arguments. Then, this function could be used to switch banks and execute the “real” function. As above, the bank number had to be known and used with the call. Furthermore, if a called function needed parameters of its own, they had to somehow be passed, but this often could not be done in registers. For **C**, this was less of a problem as often times the stack is used for arguments. Source code changes were required.

Perhaps the most challenging aspect of the MMU is maintaining a crisp handoff to the next generation engineers. While the original engineer may have completely understood the MMU, the design philosophy may not have been rigorously documented   The main problems, especially for **Z180** bank switching, are as follows: New engineers faced the following challenges:

⇒ . Some knowledge of **Z180** assembly language may be required.

⇒ The user had to maintain and manage what modules were in each bank. As banks became full, the tables or mechanisms described above had to be updated to reflect the change. Miss one function or forget a table entry and that function call would crash the program.

⇒ When adding to a new program, this maintenance issue was often times a minor one. Adding to the tables and watching for a full bank was easy to do on the fly. But go in a year later and add to a function that overfills a bank and the ripple effect could cost more time than the original change. Delete code, and there is wasted code at the end of the bank unless the ripple effect again was used to slide up modules to fill the vacancy.

⇒ Few (if any) tools today support getting physical address of symbols or segments. Add to that the limitation that most tools are often unable to do adequate link-time math to build tables with calculated values. So runtime code (ROM space) and CPU time was often used to calculate **MMU** values and sometimes even table addresses. Those manual **MMU** calculations were also error prone. Make a mistake for a rarely used function and the final program is one that also crashed, but rarely as well.

Softools, Inc. releases new Z180 specific tools

The scenario for banked program development was altered dramatically in 1990. **Softools, Inc.**, a tool maker specialized and dedicated to supporting the **Z80** and **Z180** family of processors, released the **SASM180** *Advanced Assembler and Linker* package. The following year, Softools released the **SC180 C** compiler. What set them apart from other development tool packages then, and still does today, is that all of the problems above have been solved by Softools' *Advanced Linker* **SLINK**. This is not only the most powerful, feature-rich linker for the **Z80** and **Z180** families, but for any microprocessor tool package available.

Unlimited code support on the Z180

Softools' development tools allow unlimited code support on hardware banked **Z80** systems and **Z180** based systems using the **MMU**. This is because the problems outlined above are seamlessly solved for **Z80** and **Z180** banked programs for several important reasons. These include:

⇒ **SLINK** will figure out how much banked and non-banked code and data are in the system and will automatically set the **Z180 MMU** for the proper three window configuration. You need to know *nothing* about the **MMU** to make a banked switched program. You simply tell **SLINK** which modules are non-banked and which ones are banked (and this can be done at link-time, not by going to the source code to mark functions or segments manually as banked). You tell it where RAM starts physically in the target and the program is built.

⇒ **SLINK** will automatically flow from one bank to the next as each bank is filled. There is *NO* maintenance for the resulting bank size or the modules in each bank. This is true regardless of how the functions are called (i.e., direct call, through a table, array, or pointer). If you add or delete code, **SLINK** will slide all modules up or down to fill each bank, adjusting the **MMU** if necessary.

⇒ **SLINK** automatically builds the bank table which switch banks at runtime. It also redirects all banked calls to the table -- but only when it knows it must do so. A call from one module to another where each winds up in the same bank has no additional overhead! The runtime also is optimized to not switch banks if it determines the new bank is the same as the current one.

⇒ A program that is not banked switched can be made bank switched simply be relinking and adding a 6-byte initialization sequence at program startup. There are *NO* source code changes to the actual program.

⇒ Softools' tools are the only tools to automatically bank switch a program, whether it be 100% assembly code, 100% **C** code, or a mix of the two. No registers are used and pointers to functions and address tables containing banked function addresses continue to work. There are really no limitations (for programs designed by today's standards anyway).

⇒ Any expression and any operator that can be used at assembly time can be evaluated and resolved at link time if needed. This means complete address and **MMU** calculations can be done at link-time. **SLINK** even allows accessing the logical *and* physical address of any symbol or segment name. This allows for incredibly complex calculations to be resolved at link time -- but with **SLINK**'s **MMU** and banking support this is often not required.

⇒ ***Bonus***! There is a unique feature of **SLINK** that surpass all other linkers that attempt to handle bank switching. **SLINK** can optionally take all of the modules in the banked area and best fill each bank before going to the next bank. System banks can be filled such that only a handful of bytes is unused and wasted at the end of each bank. Large programs that normally would waste 20 to 30k of ROM using user-specified ordering free up this memory using this feature.

Softools has seen the results

Softools' has several customers currently running 200k, 300k, and 400+k **Z80** and **Z180** programs. Having unlimited code support allows you to add code and features to your product, allowing Softools' tools to handle the maintenance chores of building the final program.

Unlimited data support on the Z180

The Softools **WinIDE** development environment includes the latest version of the **SC180 C** *Control Cross C Compiler*. This compiler is the first and only **C** compiler for the **Z180** family that supports *far* pointers and *far* data. This allows **C** programs to have unlimited access to the 1 megabyte address space for data access. This does not impact banked programs in any way.

Far pointers are normally 24-bit addresses (stored in 32-bit locations), but a compiler option can allow storing and using all 32-bits. This can be used for encoding additional system memory in the upper bits. *far* pointers and data are not designed for general use or for computationally intensive program code using them. They can be used for tables, menus, buffers, etc., which are stored outside the 64k logical address space, freeing this space for better purposes. A design goal of Softools was to store far pointers as real physical addresses. For example:

```
far char  *farPointer = (far char *) 0x20000UL;
```

sets farPointer to physical address 0x20000. Referencing farPointer like *farPointer = 1; stores a 1 at address 20000h. **SC180** automatically handles converting logical (near) pointers to far pointers. Far pointers can be used like near pointers (of course with the added overhead of accessing 32-bits instead of 16) and far arrays and far pointers to structures are also possible. Traditional techniques of manual **MMU** mapping to access extra data is eliminated using far pointers and far data.

Conclusion

The use of the **Z80** and **Z180** microprocessors today has real potential for systems that require large programs. Having the tools to support unlimited programs sizes and to handle data outside the logical address space is paramount to successfully using these microprocessors for larger more complex programs. Softools development tools achieve this, therefore allowing continued use of these microprocessors for a much longer time than was possible without these tools.

By: William Auerbach, President, Softools, Inc.

Danny Chi, Business Line Manager, Embedded MPU, ZiLOG