



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of "Quality Parts,Customers Priority,Honest Operation,and Considerate Service",our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



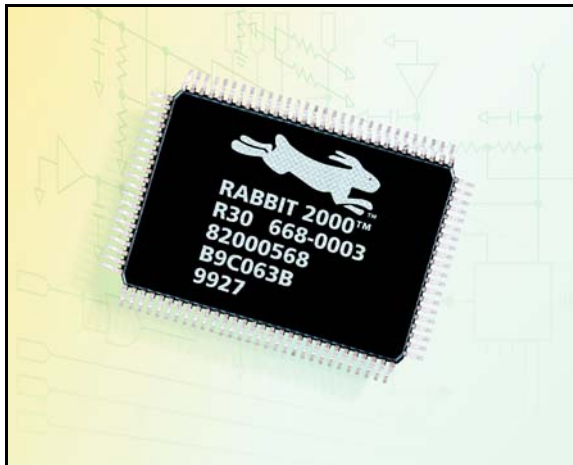
## Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China





# Rabbit 2000<sup>®</sup> Microprocessor

## User's Manual

019-0069 • 070831-P

# Rabbit 2000 Microprocessor User's Manual

Part Number 019-0069 • 070831-P • Printed in U.S.A.

©2002–2007 Rabbit Semiconductor Inc. • All rights reserved.

No part of the contents of this manual may be reproduced or transmitted in any form or by any means without the express written permission of Rabbit Semiconductor.

Permission is granted to make one or more copies as long as the copyright page contained therein is included. These copies of the manuals may not be let or sold for any reason without the express written permission of Rabbit Semiconductor.

Rabbit Semiconductor reserves the right to make changes and improvements to its products without providing notice.

## Trademarks

Rabbit and Dynamic C are registered trademarks of Rabbit Semiconductor Inc.

Rabbit 2000 is a trademark of Rabbit Semiconductor Inc.

The latest revision of this manual is available on the Rabbit Semiconductor Web site, [www.rabbit.com](http://www.rabbit.com), for free, unregistered download.

**Rabbit Semiconductor Inc.**

[www.rabbit.com](http://www.rabbit.com)

# TABLE OF CONTENTS

<b>Chapter 1. Introduction</b>	<b>1</b>
1.1 Features and Specifications .....	1
1.2 Summary of Rabbit Advantages .....	5
<b>Chapter 2. Rabbit Design Features</b>	<b>7</b>
2.1 The Rabbit 8-bit Processor vs. 16-bit and 32-bit Processors .....	8
2.2 Overview of On-Chip Peripherals.....	8
2.2.1 Serial Ports .....	8
2.2.2 System Clock .....	8
2.2.3 Time/Date Oscillator .....	9
2.2.4 Parallel I/O .....	9
2.2.5 Slave Port .....	10
2.2.6 Timers .....	10
2.3 Design Standards .....	12
2.3.1 Programming Port .....	12
2.3.2 Standard BIOS .....	12
2.4 Dynamic C Support for the Rabbit.....	12
<b>Chapter 3. Details on Rabbit</b>	
<b>Microprocessor Features</b>	<b>13</b>
3.1 Processor Registers .....	13
3.2 Memory Mapping .....	15
3.2.1 Extended Code Space .....	18
3.2.2 Extending Data Memory .....	19
3.2.3 Practical Memory Considerations .....	21
3.3 Instruction Set Outline .....	22
3.3.1 Load Immediate Data To a Register .....	23
3.3.2 Load or Store Data from or to a Constant Address .....	23
3.3.3 Load or Store Data Using an Index Register .....	24
3.3.4 Register to Register Move .....	25
3.3.5 Register Exchanges .....	25
3.3.6 Push and Pop Instructions .....	26
3.3.7 16-bit Arithmetic and Logical Ops .....	26
3.3.8 Input/Output Instructions .....	29
3.4 How to Do It in Assembly Language—Tips and Tricks .....	31
3.4.1 Zero HL in 4 Clocks .....	31
3.4.2 Exchanges Not Directly Implemented .....	31
3.4.3 Manipulation of Boolean Variables .....	31
3.4.4 Comparisons of Integers .....	32
3.4.5 Atomic Moves from Memory to I/O Space .....	34
3.5 Interrupt Structure .....	35
3.5.1 Interrupt Priority .....	35
3.5.2 Multiple External Interrupting Devices .....	37
3.5.3 Privileged Instructions, Critical Sections and Semaphores .....	37
3.5.4 Critical Sections .....	38
3.5.5 Semaphores Using Bit B,(HL) .....	38
3.5.6 Computed Long Calls and Jumps .....	39

<b>Chapter 4. Rabbit Capabilities</b>	<b>41</b>
4.1 Precisely Timed Output Pulses .....	41
4.1.1 Pulse Width Modulation to Reduce Relay Power .....	43
4.2 Open-Drain Outputs Used for Key Scan .....	44
4.3 Cold Boot.....	45
4.4 The Slave Port.....	46
4.4.1 Slave Rabbit As A Protocol UART .....	47
<b>Chapter 5. Pin Assignments and Functions</b>	<b>49</b>
5.1 Package Schematic and Pinout.....	49
5.2 Package Mechanical Dimensions .....	50
5.3 Rabbit Pin Descriptions.....	52
5.4 Bus Timing.....	58
5.5 Description of Pins with Alternate Functions .....	59
5.6 DC Characteristics .....	61
5.6.1 5.0 Volts .....	62
5.6.2 3.3 Volts .....	63
5.7 I/O Buffer Sourcing and Sinking Limit.....	64
<b>Chapter 6. Rabbit Internal I/O Registers</b>	<b>65</b>
6.1 Default Values for all the Peripheral Control Registers .....	65
<b>Chapter 7. Miscellaneous I/O Functions</b>	<b>71</b>
7.1 Processor Identification.....	71
7.2 Rabbit Oscillators and Clocks.....	72
7.3 Clock Doubler.....	74
7.4 Controlling Power Consumption .....	76
7.5 Output Pins CLK, STATUS, /WDTOUT, /BUFEN.....	77
7.6 Time/Date Clock (Real-Time Clock) .....	78
7.7 Watchdog Timer .....	80
7.8 System Reset.....	82
7.9 Rabbit Interrupt Structure .....	84
7.9.1 External Interrupts .....	86
7.9.2 Interrupt Vectors: INT0 - EIR,0x00/INT1 - EIR,0x08 .....	87
7.10 Bootstrap Operation .....	88
<b>Chapter 8. Memory Mapping and Interface</b>	<b>91</b>
8.1 Memory-Mapping Unit .....	91
8.2 Memory Interface Unit.....	93
8.3 Memory Control Unit Registers .....	94
8.3.1 Memory Bank Control Registers .....	94
8.3.2 MMU Instruction/Data Register .....	95
8.3.3 Memory Timing Control Register .....	95
8.4 Allocation of Extended Code and Data.....	96
8.5 How Compiler Compiles to Memory .....	97
<b>Chapter 9. Parallel Ports</b>	<b>99</b>
9.1 Parallel Port A.....	100
9.2 Parallel Port B .....	101
9.3 Parallel Port C .....	102
9.4 Parallel Port D.....	103
9.5 Parallel Port E .....	106
<b>Chapter 10. I/O Bank Control Registers</b>	<b>109</b>



<b>Chapter 11. Timers</b>	<b>111</b>
11.1 Timer A.....	112
11.1.1 Timer A I/O Registers .....	113
11.1.2 Practical Use of Timer A .....	114
11.2 Timer B.....	115
11.2.1 Using Timer B .....	117
<b>Chapter 12. Rabbit Serial Ports</b>	<b>119</b>
12.1 Serial Port Register Layout.....	120
12.2 Serial Port Interrupt .....	123
12.3 Transmit Serial Data Timing.....	124
12.4 Receive Serial Data Timing.....	124
12.5 Clocked Serial Ports.....	125
12.6 Clocked Serial Timing.....	128
12.6.1 Clocked Serial Timing With Internal Clock .....	128
12.6.2 Clocked Serial Timing with External Clock .....	128
12.7 Serial Port Software Suggestions .....	129
12.7.1 Controlling an RS-485 Driver and Receiver .....	131
12.7.2 Transmitting Dummy Characters .....	131
12.7.3 Transmitting and Detecting a Break .....	131
12.7.4 Using A Serial Port to Generate a Periodic Interrupt .....	131
12.7.5 Extra Stop Bits, Sending Parity, 9th Bit Communication Schemes .....	132
12.7.6 Supporting 9th Bit Communication Protocols .....	134
12.7.7 Rabbit-Only Master/Slave Protocol .....	135
12.7.8 Data Framing/Modbus .....	135
<b>Chapter 13. Rabbit Slave Port</b>	<b>137</b>
13.1 Hardware Design of Slave Port Interconnection.....	143
13.2 Slave Port Registers .....	143
13.3 Applications and Communications Protocols for Slaves .....	145
13.3.1 Slave Applications .....	145
13.3.2 Master-Slave Messaging Protocol .....	146
<b>Chapter 14. Rabbit 2000 Clocks</b>	<b>149</b>
14.1 Low-Power Design .....	150
14.2 Clock Spectrum Spreader Module.....	150
<b>Chapter 15. AC Timing Specifications</b>	<b>151</b>
15.1 Memory Access and I/O Read/Write Times .....	154
15.2 Current Consumption .....	162
<b>Chapter 16. Rabbit BIOS and Virtual Driver</b>	<b>165</b>
16.1 The BIOS .....	165
16.1.1 BIOS Services .....	165
16.1.2 BIOS Assumptions .....	166
16.2 Virtual Driver .....	166
16.2.1 Periodic Interrupt .....	166
16.2.2 Watchdog Timer Support .....	166
<b>Chapter 17. Other Rabbit Software</b>	<b>169</b>
17.1 Power Management Support.....	169
17.2 Reading and Writing I/O Registers.....	170
17.2.1 Using Assembly Language .....	170
17.2.2 Using Library Functions .....	170
17.3 Shadow Registers .....	171
17.3.1 Updating Shadow Registers .....	171

17.3.2	Interrupt While Updating Registers .....	171
17.3.3	Write-only Registers Without Shadow Registers .....	172
17.4	Timer and Clock Usage .....	172
<b>Chapter 18.</b>	<b>Rabbit Instructions</b>	<b>175</b>
18.1	Load Immediate Data .....	178
18.2	Load & Store to Immediate Address.....	178
18.3	8-bit Indexed Load and Store .....	178
18.4	16-bit Indexed Loads and Stores.....	178
18.5	16-bit Load and Store 20-bit Address .....	179
18.6	Register to Register Moves.....	179
18.7	Exchange Instructions .....	180
18.8	Stack Manipulation Instructions .....	180
18.9	16-bit Arithmetic and Logical Ops.....	180
18.10	8-bit Arithmetic and Logical Ops.....	181
18.11	8-bit Bit Set, Reset and Test.....	182
18.12	8-bit Increment and Decrement.....	182
18.13	8-bit Fast A register Operations .....	183
18.14	8-bit Shifts and Rotates .....	183
18.15	Instruction Prefixes .....	184
18.16	Block Move Instructions .....	184
18.17	Control Instructions - Jumps and Calls .....	185
18.18	Miscellaneous Instructions .....	185
18.19	Privileged Instructions .....	186
<b>Chapter 19.</b>	<b>Differences Rabbit vs. Z80/Z180 Instructions</b>	<b>187</b>
<b>Chapter 20.</b>	<b>Instructions in Alphabetical Order With Binary Encoding</b>	<b>189</b>
<b>Appendix A.</b>	<b>The Rabbit Programming Port</b>	<b>197</b>
A.1	The Rabbit Programming Port .....	197
A.2	Use of the Programming Port as a Diagnostic/Setup Port.....	198
A.3	Alternate Programming Port.....	198
A.4	Suggested Rabbit Crystal Frequencies .....	199
<b>Appendix B.</b>	<b>Rabbit 2000 Revisions</b>	<b>201</b>
B.1	Rabbit 2000 Revisions.....	201
B.2	Discussion of Fixes and Improvements .....	203
B.2.1	Rabbit Internal I/O Registers .....	204
B.2.2	Revision-Level ID Register .....	205
B.2.3	Serial Port Changes .....	207
B.2.4	Improved Battery-Backup Circuit .....	209
B.2.5	Added Support for Instruction/Data Split .....	211
B.2.6	Write Inhibit (/WE0) After Reset .....	213
B.2.7	Chip Selects Inactive During Internal I/O .....	213
B.2.8	External Interrupt Input Bug Fix .....	213
B.2.9	IOI/IOE Prefix Bug Fix .....	213
B.2.10	DDCB/FDCB Instruction Page and Wait State Bug Fixes .....	214
B.2.11	LDIR/LDDR Instruction/Data Split Bug Fix .....	214
B.2.12	Clock Spectrum Spreader Module .....	215
B.2.13	Early Memory Output-Enable Feature .....	218
<b>Index</b>		<b>219</b>

# 1. INTRODUCTION

Rabbit Semiconductor was formed expressly to design a better microprocessor for use in small and medium-scale controllers. The first product is the *Rabbit 2000* microprocessor. The Rabbit 2000 designers have had years of experience using Z80, Z180 and HD64180 microprocessors in small controllers. The Rabbit shares a similar architecture and a high degree of compatibility with these microprocessors, but it is a vast improvement.

The Rabbit has been designed in close cooperation with Z-World, Inc., a long-time manufacturer of low-cost single-board computers. Z-World and Rabbit Semiconductor products are supported by an innovative C-language development system (Dynamic C).

The Rabbit 2000 is easy to use. Hardware and software interfaces are as uncluttered and are as foolproof as possible. The Rabbit 2000 has outstanding computation speed for a microprocessor with an 8-bit bus. This is because the Z80-derived instruction set is very compact and the design of the memory interface allows maximum utilization of the memory bandwidth. The Rabbit races through instructions.

Traditional microprocessor hardware and software development is simplified for Rabbit users. In-circuit emulators are not needed and will not be missed by the Rabbit developer. Software development is accomplished by connecting a simple interface cable from a PC serial port to the Rabbit-based target system.

## 1.1 Features and Specifications

- 100-pin PQFP package. Operating voltage 2.7 V to 5 V. Clock speed to 30 MHz. All specifications are given for both industrial and commercial temperature and voltage ranges. Rabbit microprocessors cost under \$10 in moderate quantities.
- Industrial specifications are for a voltage variation of 10% and a temperature range from  $-40^{\circ}\text{C}$  to  $+85^{\circ}\text{C}$ . Commercial specifications are for a voltage variation of 5% and a temperature range from  $0^{\circ}\text{C}$  to  $70^{\circ}\text{C}$ .
- 1-megabyte code space allows C programs with up to 50,000+ lines of code. The extended Z80-style instruction set is C-friendly, with short and fast instructions for most common C operations.
- Four levels of interrupt priority make a fast interrupt response practical for critical applications. The maximum time to the first instruction of an interrupt routine is about 1  $\mu\text{s}$  at a clock speed of 25 MHz.



- Access to I/O devices is accomplished by using memory access instructions with an I/O prefix. Access to I/O devices is thus faster and easier compared to processors with a restricted I/O instruction set.
- The hardware design rules are simple. Up to six static memory chips (such as RAM and flash EPROM) connect directly to the microprocessor with no glue logic. Even larger amounts of memory can be handled by using parallel I/O lines as high-order address lines. The Rabbit runs with no wait states at 24 MHz with a memory having an access time of 70 ns. There are two clocks per memory access. Most I/O devices may be connected without glue logic.

The memory cycle is two clocks long. A clean memory and I/O cycle completely avoid the possibility of tri-state fights. Peripheral I/O devices can usually be interfaced in a glueless fashion using pins programmable as I/O chip selects, I/O read strobes or I/O write strobe pins. A built-in clock doubler allows 1/2-frequency crystals to be used to reduce radiated emissions.

- The Rabbit may be cold-booted via a serial port or the parallel access slave port. This means that flash program memory may be soldered in unprogrammed, and can be reprogrammed at any time without any assumption of an existing program or BIOS. A Rabbit that is slaved to a master processor can operate entirely with volatile RAM, depending on the master for a cold program boot.
- There are 40 parallel I/O lines (shared with serial ports). Some I/O lines are timer synchronized, which permits precisely timed edges and pulses to be generated under combined hardware and software control.
- There are four serial ports. All four serial ports can operate asynchronously in a variety of customary operating modes; two of the ports can also be operated synchronously to interface with serial I/O devices. The baud rates can be very high—1/32 the clock speed for asynchronous operation, and 1/6 the clock speed externally or 1/4 the clock speed internally in synchronous mode. In asynchronous mode, the Rabbit, like the Z180, supports sending flagged bytes to mark the start of a message frame. The flagged bytes have 9 data bits rather than 8 data bits; the extra bit is located after the first 8 bits, where the stop bit is normally located, and marks the start of a message frame.
- A slave port allows the Rabbit to be used as an intelligent peripheral device slaved to a master processor. The 8-bit slave port has six 8-bit registers, 3 for each direction of communication. Independent strobes and interrupts are used to control the slave port in both directions. Only a Rabbit and a RAM chip are needed to construct a complete slave system if the clock and reset are shared with the master processor
- The built-in battery-backable time/date clock uses an external 32.768 kHz crystal. The time/date clock can also be used to provide periodic interrupts every 488  $\mu$ s. Typical battery current consumption is 25  $\mu$ A with the suggested battery circuit. An alternative circuit provides means for substantially reducing this current.
- Numerous timers and counters (six all together) can be used to generate interrupts, baud rate clocks, and timing for pulse generation.

- The built-in main clock oscillator uses an external crystal or more usually a ceramic resonator. Typical resonator frequencies are in the range of 1.8 MHz to 29.5 MHz. Since precision timing is available from the separate 32.768 kHz oscillator, a low-cost ceramic resonator with ½ percent error is generally satisfactory. The clock can be doubled or divided by 8 to modify speed and power dynamically. The I/O clock, which clocks the serial ports, is divided separately so as not to affect baud rates and timers when the processor clock is divided or multiplied. For ultra low power operation, the processor clock can be driven from the separate 32.768 kHz oscillator and the main oscillator can be powered down. This allows the processor to operate at approximately 100 µA and still execute instructions at the rate of approximately 10,000 instructions per second. This is a powerful alternative to sleep modes of operation used by other processors. The current is approximately 65 mA at 25 MHz and 5 V. The current is proportional to voltage and clock speed—at 3.3 V and 7.68 MHz the current would be 13 mA, and at 1 MHz the current is reduced to less than 2 mA. Flash memory with automatic power down (from AMD) should be used for operation at the lowest power.
- The excellent floating-point performance is due to a tightly coded library and powerful processing capability. For example, a 25 MHz clock takes 14 µs for a floating add, 13 µs for a multiply, and 40 µs for a square root. In comparison, a 386EX processor running with an 8-bit bus at 25 MHz and using Borland C is about 10 times slower.
- There is a built-in watchdog timer.
- The standard 10-pin programming port eliminates the need for in-circuit emulators. A very simple 10 pin connector can be used to download and debug software using Rabbit Semiconductor's Dynamic C and a simple connection to a PC serial port. The incremental cost of the programming port is extremely small.

Figure 1-1 shows a block diagram of the Rabbit.

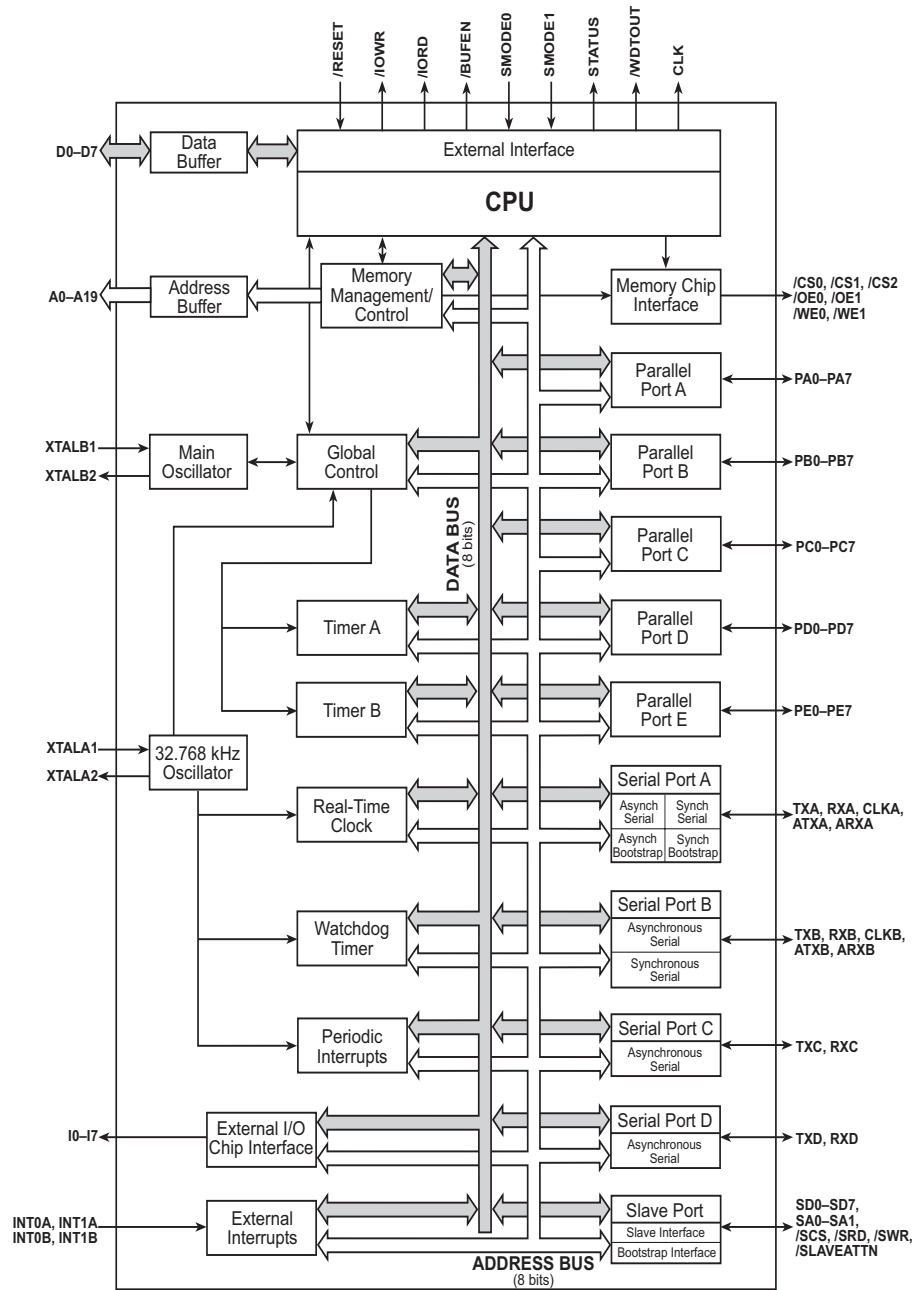


Figure 1-1. Block Diagram of the Rabbit Microprocessor

## 1.2 Summary of Rabbit Advantages

- The glueless architecture makes it is easy to design the hardware system.
- There are a lot of serial ports and they can communicate very fast.
- Precision pulse and edge generation is a standard feature.
- Interrupts can have multiple priorities.
- Processor speed and power consumption are under program control.
- The ultra low power mode can perform computations and execute logical tests since the processor continues to execute, albeit at 32 kHz.
- The Rabbit may be used to create an intelligent peripheral or a slave processor. For example, protocol stacks can be off loaded to a Rabbit slave. The master can be any processor.
- The Rabbit can be cold booted so unprogrammed flash memory can be soldered in place.
- You can write serious software, be it 1,000 or 50,000 lines of C code. The tools are there and they are low in cost.
- If you know the Z80 or Z180, you know most of the Rabbit.
- A simple 10-pin programming interface replaces in-circuit emulators and PROM programmers.
- The battery backable time/date clock is included.
- The standard Rabbit chip is made to industrial temperature and voltage specifications.





## 2. RABBIT DESIGN FEATURES

The Rabbit is an evolutionary design. The instruction set and the register layout is that of the Z80 and Z180. The instruction set has been augmented by a substantial number of new instructions. Some obsolete or redundant Z180 instructions have been dropped to make available efficient 1-byte opcodes for important new instructions. (see “Differences Rabbit vs. Z80/Z180 Instructions” on page 187.) The advantage of this evolutionary approach is that users familiar with the Z80 or Z180 can immediately understand the Rabbit. Existing source code can be assembled or compiled for the Rabbit with minimal changes.

Changing technology has made some features of the Z80/Z180 family obsolete, and these have been dropped. For example, the Rabbit has no special support for dynamic RAM but it has extensive support for static memory. This is because the price of static memory has decreased to the point that it has become the preferred choice for medium-scale embedded systems. The Rabbit has no support for DMA (direct memory access) because most of the uses for which DMA is traditionally used do not apply to embedded systems, or they can be accomplished better in other ways, such as fast interrupt routines, external state machines or slave processors.

Our experience in writing C compilers has revealed the shortcomings of the Z80 instruction set for executing the C language. The main problem is the lack of instructions for handling 16-bit words and for accessing data at a computed address, especially when the stack contains that data. New instructions correct these problems.

Another problem with many 8-bit processors is their slow execution and a lack of number-crunching ability. Good floating-point arithmetic is an important productivity feature in smaller systems. It is easy to solve many programming problems if an adequate floating-point capability is available. The Rabbit’s improved instruction set provides fast floating-point and fast integer math capabilities.

The Rabbit supports four levels of interrupt priorities. This is an important feature that allows the effective use of super fast interrupt routines for real-time tasks.



## 2.1 The Rabbit 8-bit Processor vs. 16-bit and 32-bit Processors

The Rabbit is an 8-bit processor with an 8-bit external data bus and an 8-bit internal data bus. Because the Rabbit makes the most of its external 8-bit bus and because it has a compact instruction set, its performance is as good as many 16-bit processors. Thus the Rabbit can handle many 16-bit operations.

We hesitate to compare the Rabbit to 32-bit processors, but there are undoubtedly occasions where the user can use a Rabbit instead of a 32-bit processor and save a vast amount of money. Many Rabbit instructions are 1 byte long. In contrast, the minimum instruction length on most 32-bit RISC processors is 32 bits.

## 2.2 Overview of On-Chip Peripherals

The on-chip peripherals were chosen based on our experience as to what types of peripheral devices are most useful in small embedded systems. The major on-chip peripherals are the serial ports, system clock, time/date oscillator, parallel I/O, slave port, and timers. These are described below.

### 2.2.1 Serial Ports

There are four serial ports designated ports A, B, C, and D. All four serial ports can operate in an asynchronous mode up to the baud rate of the system clock divided by 32. The asynchronous ports can handle 7 or 8 data bits. A 9th bit address scheme, where an additional bit is sent to mark the first byte of a message, is also supported. The software can tell when the last byte of a message has finished transmitting from the output shift register - correcting an important defect of the Z180. This is important for RS-485 communication because the line driver cannot have the direction of transmission reversed until the last bit has been sent. In many UARTs, including those on the Z180, it is difficult to generate an interrupt after the last bit is sent. Parity bits and multiple stop bits are not supported directly by the Rabbit, but can be accomplished with appropriate driving software.

Serial ports A and B can be operated alternately in the clocked serial mode. In this mode, a clock line synchronously clocks the data in or out. Either device of the two devices communicating can supply the clock. When the Rabbit provides the clock, the baud rate can be up to 1/4 of the system clock frequency, or more than 7,375,000 bps for a 29.5 MHz clock speed.

Serial port A has special features. It can be used to cold boot the system after reset. Serial port A is the normal port that is used for software development under Dynamic C.

### 2.2.2 System Clock

The main oscillator uses an external crystal with a frequency typically in the range from 1.8 MHz to 29.5 MHz. The processor clock is derived from the oscillator output by either doubling the frequency, using the frequency directly, or dividing the frequency by 8. The processor clock can also be driven by the 32.768 kHz oscillator for very low power operation, in which case the main oscillator can be shut down under software control.

Table 2-1 provides estimates of the operating power for selected clock speeds.

**Table 2-1. Operating Power Estimates at Selected Clock Speeds**

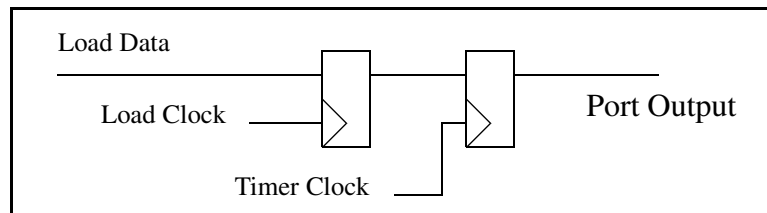
Clock Speed (MHz)	Voltage (V)	Current (mA)	Power (mW)	Clock Speed (MHz)	Voltage (V)	Current (mA)	Power (mW)
25.0	5.0	80	400	6.0	2.5	10	25
12.5	5.0	40	200	3.0	2.5	5	12
12.5	3.3	26	87	1.5	2.5	2.5	6
6.0	3.3	13	42	0.032	2.5	0.054	0.135

### 2.2.3 Time/Date Oscillator

The 32.768 kHz oscillator drives an external 32.768 kHz quartz crystal. The 32.768 kHz clock is used to drive a battery-backable (there is a separate power pin) internal 48-bit counter that serves as a real-time clock (RTC). The counter can be set and read by software and is intended for keeping the date and time. There are enough bits to keep the date for more than 100 years. The 32.768 kHz oscillator is also used to drive the watchdog timer and to generate the baud clock for serial port A during the cold boot sequence.

### 2.2.4 Parallel I/O

There are 40 parallel input/output lines divided among five 8-bit ports designated A through E. Most of the port lines have alternate functions, such as serial data or chip select strobes. Parallel ports D and E have the capability of timer-synchronized outputs. The output registers are cascaded.



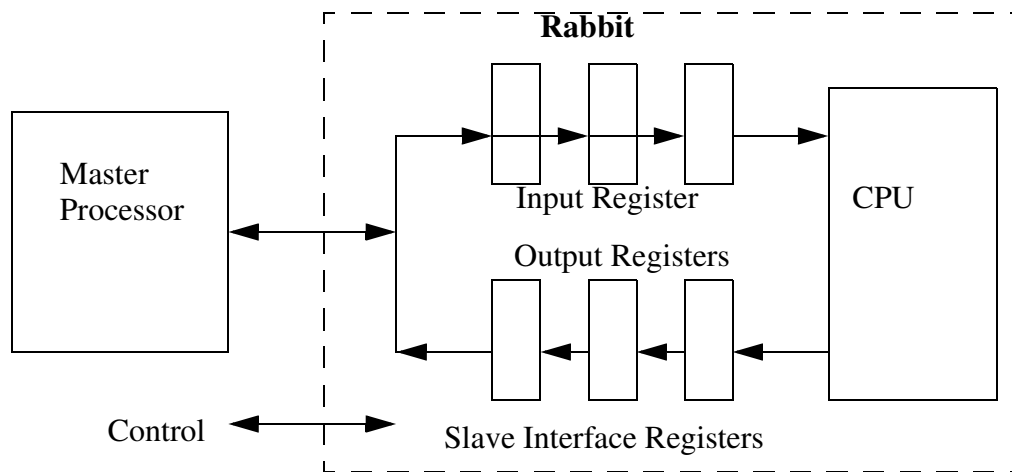
**Figure 2-1. Cascaded Output Registers for Parallel Ports D and E**

Stores to the port are loaded in the first-level register. That register in turn is transferred to the output register on a selected timer signal. The timer signal can also cause an interrupt that can be used to set up the next bit to be output on the next timer pulse. This feature can be used to generate precisely controlled pulses whose edges are positioned with high accuracy in time. Applications include communications signaling, pulse width modulation and driving stepper motors.

## 2.2.5 Slave Port

The slave port is designed to allow the Rabbit to be a slave to another processor, which could be another Rabbit. The port is shared with parallel port A and is a bidirectional data port. The master can read any of three registers selected via two select lines that form the register address and a read strobe that causes the register contents to be output by the port. These same registers can be written as I/O registers by the Rabbit slave. Three additional registers transmit data in the opposite direction. They are written by the master by means of the two select lines and a write strobe.

Figure 2-2 shows the data paths in the slave port.



**Figure 2-2. Slave-Port Data Paths**

The slave Rabbit can read the same registers as I/O registers. When incoming data bits are written into one of the registers, status bits indicate which registers have been written, and an optional interrupt can be programmed to take place when the write occurs. When the slave writes to one of the registers carrying data bits outward, an attention line is enabled so that the master can detect the data change and be interrupted if desired. One line tells the master that the slave has read all the incoming data. Another line tells the master that new outgoing data bits are available and have not yet been read by the master. The slave port can be used to direct the master to perform tasks using a variety of communication protocols over the slave port.

## 2.2.6 Timers

The Rabbit has several timer systems. The periodic interrupt is driven by the 32.768 kHz oscillator divided by 16, giving an interrupt every 488  $\mu$ s if enabled. This is intended to be used as a general-purpose clock interrupt. Timer A consists of five 8-bit countdown and reload registers that can be cascaded up to two levels deep. Each countdown register can be set to divide by any number between 1 and 256. The output of four of the timers is used to provide baud clocks for the serial ports. Any of these registers can also cause interrupts and clock the timer-synchronized parallel output ports. Timer B consists of a 10-bit

counter that can be read but not written. There are two 10-bit match registers and comparators. If the match register matches the counter, a pulse is output. Thus the timer can be programmed to output a pulse at a predetermined count in the future. This pulse can be used to clock the timer-synchronized parallel-port output registers as well as cause an interrupt. Timer B is convenient for creating an event at a precise time in the future under program control.

Figure 2-3 illustrates the Rabbit timers.

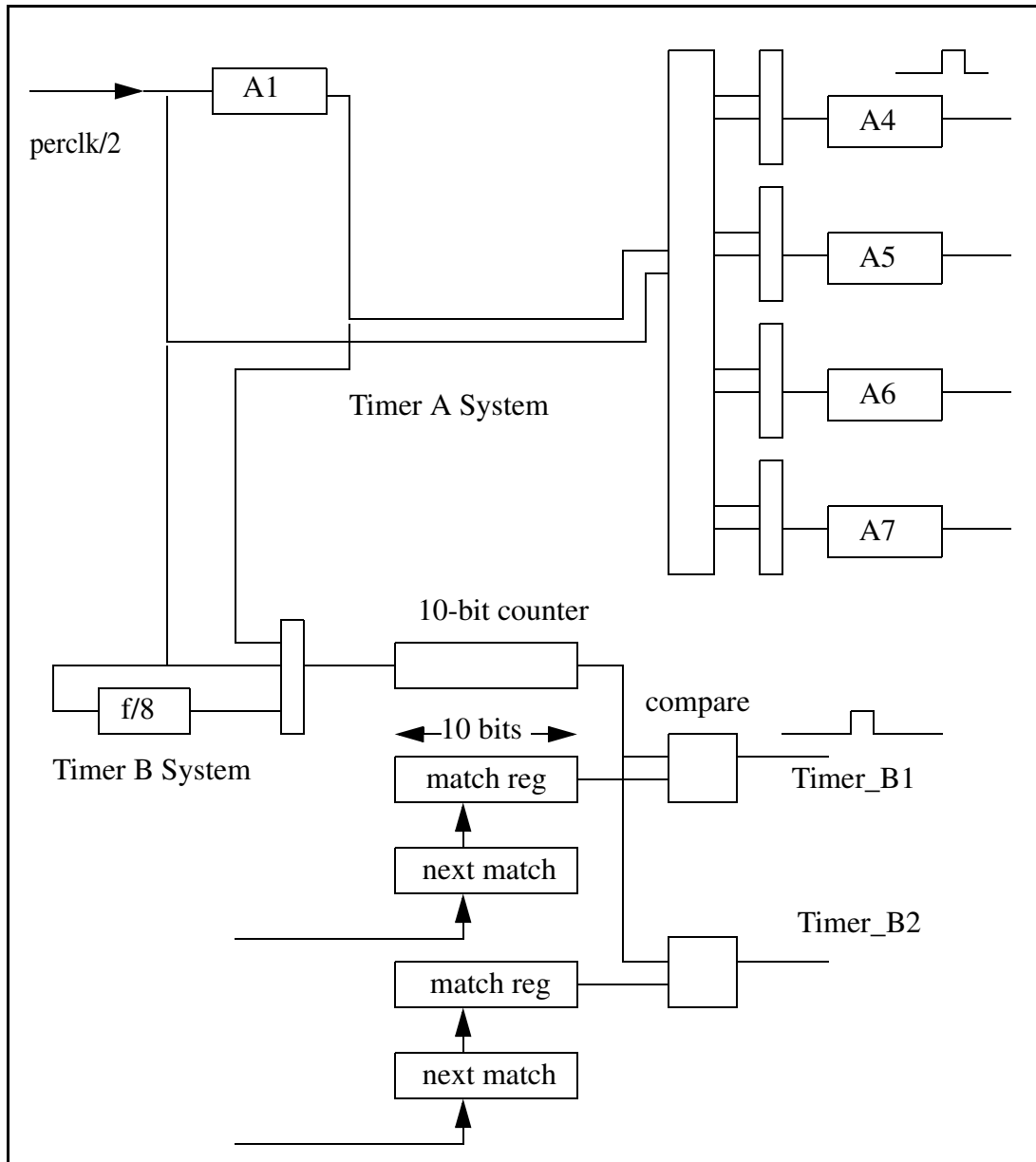


Figure 2-3. Rabbit Timers

## **2.3 Design Standards**

The same functionality can be accomplished in many ways using the Rabbit. By publishing design standards, or standard ways to accomplish common objectives, software and hardware support become easier.

### **2.3.1 Programming Port**

Rabbit Semiconductor publishes a specification for a standard programming port (see Appendix A.1, “The Rabbit Programming Port,”) and provides a converter cable that may be used to connect a PC serial port to the standard programming interface. The interface is implemented using a 10-pin connector with two rows of pins on 2 mm centers. The port is connected to Rabbit serial port A, to the startup mode pins on the Rabbit, to the Rabbit reset pin, and to a programmable output pin that is used to signal the PC that attention is needed. With proper precautions in design and software, it is possible to use serial port A as both a programming port and as a user-defined serial port, although this will not be necessary in most cases.

Rabbit Semiconductor supports the use of the standard programming port and the standard programming cable as a diagnostic and setup port to diagnosis problems or set up systems in the field.

### **2.3.2 Standard BIOS**

Rabbit Semiconductor provides a standard BIOS for the Rabbit. The BIOS is a software program that manages startup and shutdown, and provides basic services for software running on the Rabbit.

## **2.4 Dynamic C Support for the Rabbit**

Dynamic C is Rabbit Semiconductor’s interactive C language development system. Dynamic C runs on a PC under Windows 95/98/Me/XP or Windows NT. It provides a combined compiler, editor and debugger. The usual method for debugging a target system based on the Rabbit is to implement the 10-pin programming connector that connects to the PC serial port via a standard converter cable. Dynamic C libraries contain highly perfected software to control the Rabbit. These includes drivers, utility and math routines and the debugging BIOS for Dynamic C.

In addition, the internationally-known real-time operating system, uC/OS-II, has been ported to the Rabbit and is available starting with Dynamic C Premier v. 6.50.

# 3. DETAILS ON RABBIT MICROPROCESSOR FEATURES

## 3.1 Processor Registers

The Rabbit's registers are nearly identical to those of the Z180 or the Z80. The figure below shows the register layout. The XPC and IP registers are new. The EIR register is the same as the Z80 I register, and is used to point to a table of interrupt vectors for the externally generated interrupts. The IIR register occupies the same logical position in the instruction set as the Z80 R register, but its function is to point to an interrupt vector table for internally generated interrupts.

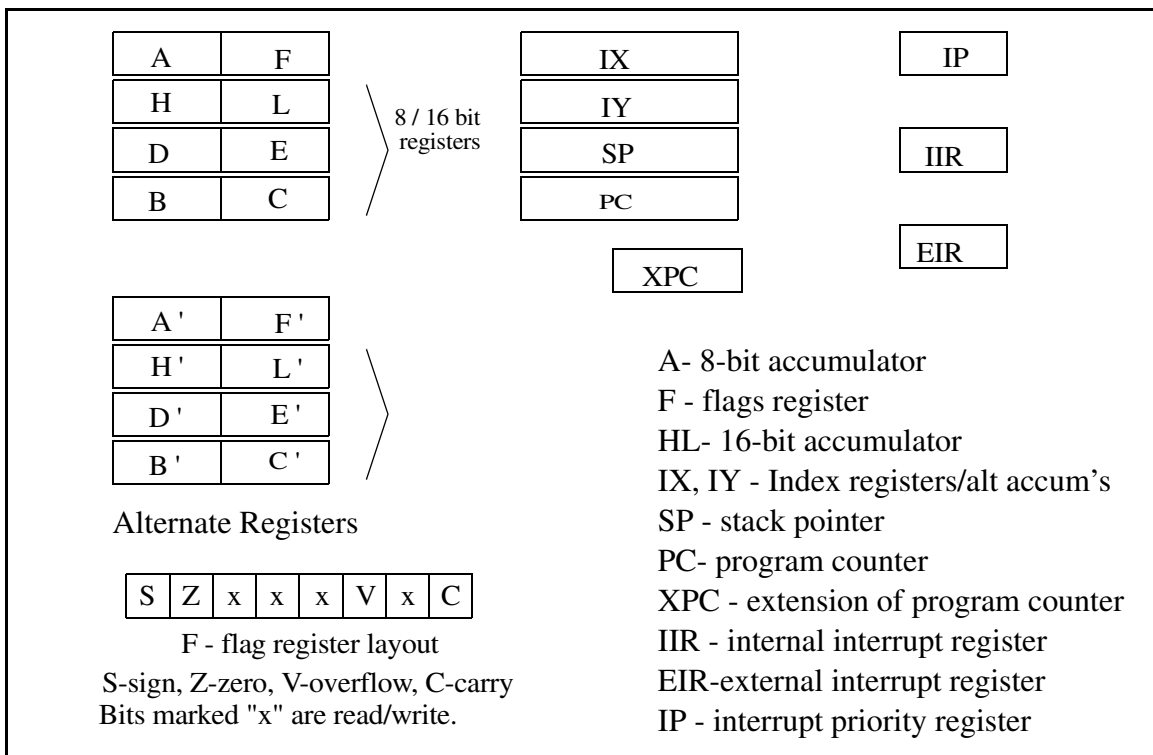


Figure 3-1. Rabbit Registers



The Rabbit (and the Z80/Z180) processor has two accumulators—the A register serves as an 8-bit accumulator for 8-bit operations such as **ADD** or *and*. The 16-bit register HL register serves as an accumulator for 16-bit operations such as **ADD HL, DE**, which adds the 16-bit register DE to the 16-bit accumulator HL. For many operations IX or IY can substitute for **HL** as accumulators.

The register marked F is the flags register or status register. It holds a number of flags that provide information about the last operation performed. The flag register cannot be accessed directly except by using the **POP AF** and **PUSH AF** instructions. Normally the flags are tested by conditional jump instructions. The flags are set to mark the results of arithmetic and logic operations according to rules that are specified for each instruction. There are four unused read/write bits in the flag register that are available to the user via the **PUSH AF** and **POP AF** instructions. These bits should be used with caution since new-generation Rabbit processors could use these bits for new purposes.

The registers IX, IY and HL can also serve as index registers. They point to memory addresses from which data bits are fetched or stored. Although the Rabbit can address a megabyte or more of memory, the index registers can only directly address 64K of memory (except for certain extended addressing LDP instructions). The addressing range is expanded by means of the memory mapping hardware (see “Memory Mapping” on page 15) and by special instructions. For most embedded applications, 64K of *data* memory (as opposed to *code* memory) is sufficient. The Rabbit can efficiently handle a megabyte of code space.

The register SP points to the stack that is used for subroutine and interrupt linkage as well as general-purpose storage.

A feature of the Rabbit (and the Z80/Z180) is the *alternate register set*. Two special instructions swap the alternate registers with the regular registers. The instruction **EX AF, AF'** exchanges the contents of AF with AF'. The instruction **EXX** exchanges HL, DE, and BC with HL', DE', and BC'. Communication between the regular and alternate register set in the original Z80 architecture was difficult because the exchange instructions provided the only means of communication between the regular and alternate register sets. The Rabbit has new instructions that greatly improve communication between the regular and alternate register set. This effectively doubles the number of registers that are easily available for the programmer's use. It is not intended that the alternate register set be used to provide a separate set of registers for an interrupt routine, and Dynamic C does not support this usage because it uses both registers sets freely.

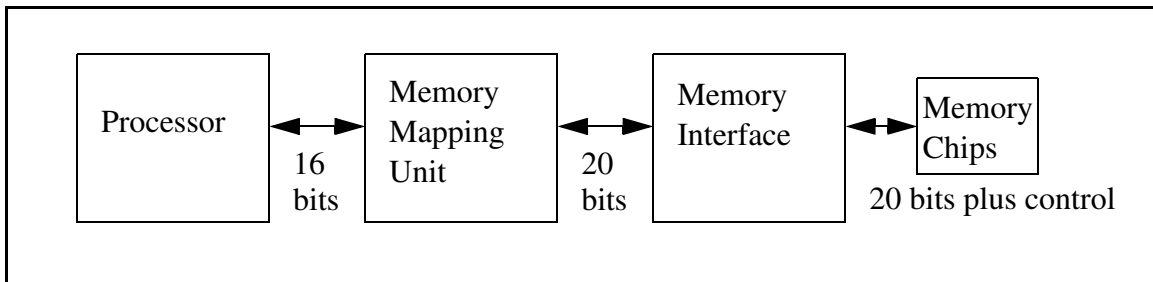
The IP register is the interrupt priority register. It contains four 2-bit fields that hold a history of the processor's interrupt priority. The Rabbit supports four levels of processor priority, something that exists only in a very restricted form in the Z80 or Z180.

## 3.2 Memory Mapping

Except for a handful of special instructions (see Section 18.5, “16-bit Load and Store 20-bit Address”), the Rabbit instructions directly address a 64K data memory space. This means that the address fields in the instructions are 16 bits long and that the registers that may be used as pointers to memory addresses (index registers (**IX**, **IY**), program counter and stack pointer (**SP**)) are also 16 bits long.

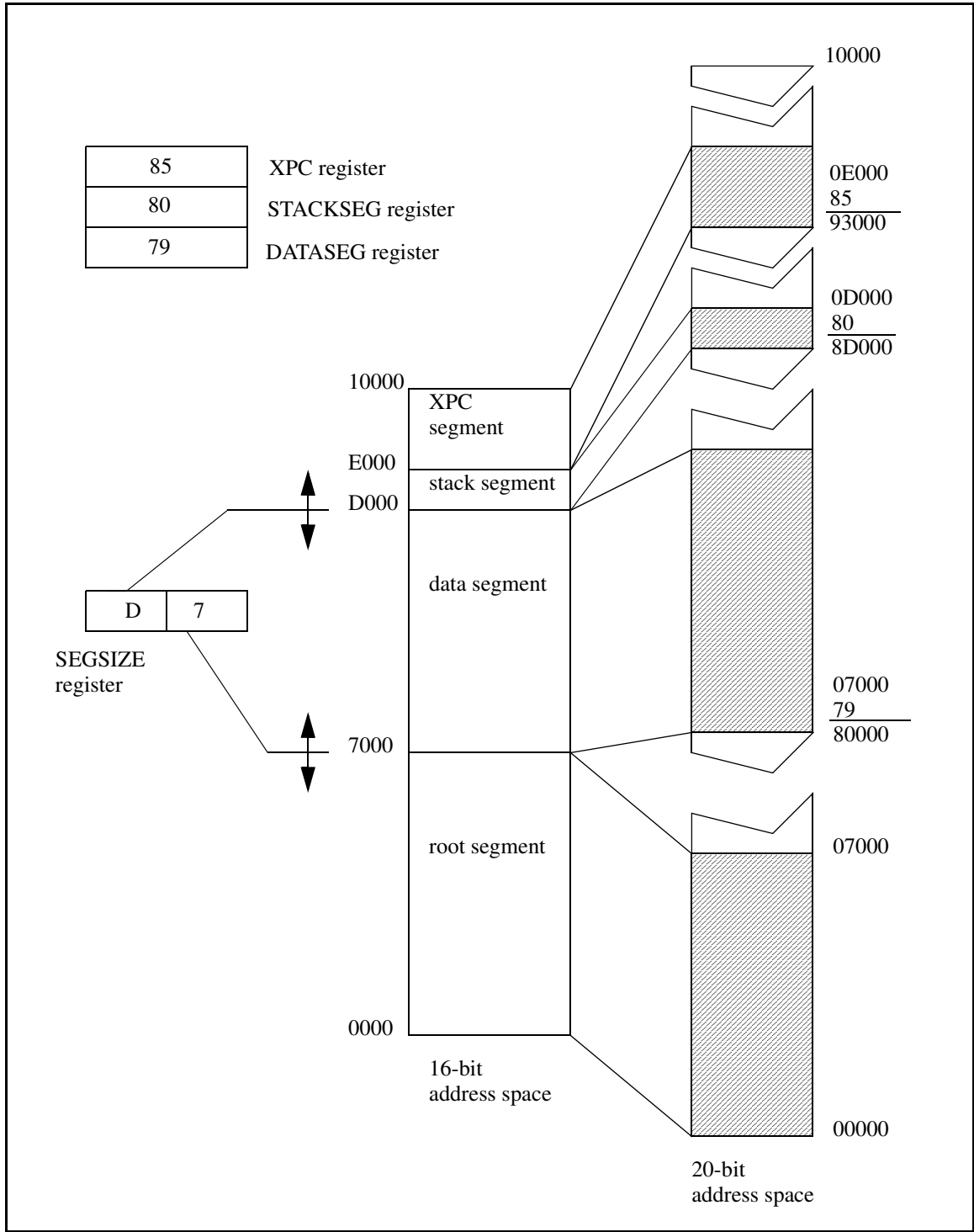
Because Rabbit instructions use 16-bit addresses, the instructions are shorter and can execute much faster than, for example, 32-bit addresses. The executable code is also very compact. Even though these 16-bit addresses are a valuable asset, they do create some complications because a memory-mapping unit is needed in order to access a reasonable amount of memory for modern C programs.

The Rabbit memory-mapping unit is similar to, but more powerful than, the Z180 memory-mapping unit. Figure 3-2 illustrates the relationship among the major components related to addressing memory.



**Figure 3-2. Addressing Memory Components**

The memory-mapping unit receives 16-bit addresses as input and outputs 20-bit addresses. The processor (except for certain LDP instructions) sees only a 16-bit address space. That is, it sees 65536 distinctly addressable bytes that its instructions can manipulate. Three segment registers are used to map this 16-bit space into a 1-megabyte space. The 16-bit space is divided into four separate zones. Each zone, except the first or root zone, has a segment register that is added to the 16-bit address within the zone to create a 20-bit address. The segment register has eight bits and those eight bits are added to the upper four bits of the 16-bit address, creating a 20-bit address. Thus, each separate zone in the 16-bit memory becomes a window to a segment of memory in the 20-bit address space. The relative size of the four segments in the 16-bit space is controlled by the **SEGSIZE** register. This is an 8-bit register that contains two 4-bit registers. This controls the boundary between the first and the second segment and the boundary between the second and the third segment. The location of the two movable segment boundaries is determined by a 4-bit value that specifies the upper four bits of the address where the boundary is located. These relationships are illustrated in Figure 3-3.



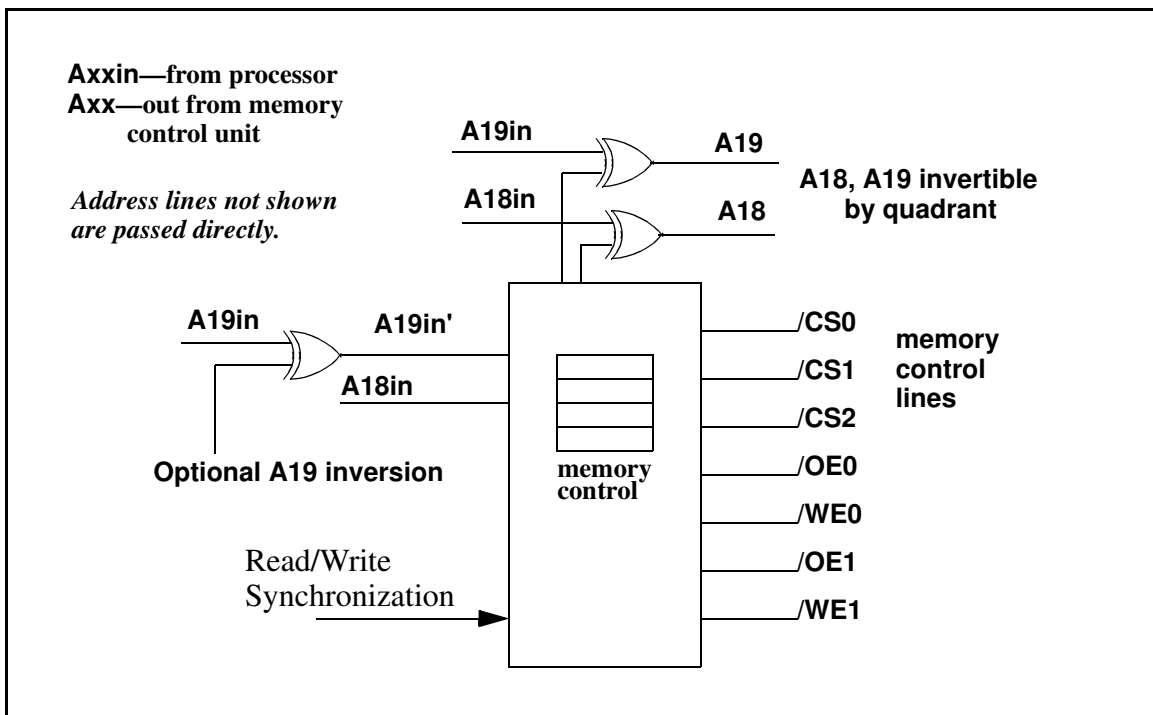
**Figure 3-3. Example of Memory Mapping Operation**

The names given to the segments in the figure are evocative of the common uses for each segment. The *root segment* is mapped to the base of flash memory and contains the startup code as well as other code that may happen to be stored there. The *data segment* usage varies depending on the overall strategy for setting up memory. It may be an extension of

the root segment or it may contain data variables. The *stack segment* is normally 4K long and it holds the system stack. The *XPC segment* is normally used to execute code that is not stored in the root segment or the data segment. Special instructions support executing code that is visible in the XPC segment.

The memory interface unit receives the 20-bit addresses generated by the memory-mapping unit. The memory interface unit conditionally modifies address lines A16, A18 and A19. The other address lines of the 20-bit address are passed unconditionally. The memory interface unit provides control signals for external memory chips. These interface signals are chip selects (*/CS0*, */CS1*, */CS2*), output enables (*/OE0*, */OE1*), and write enables (*/WE0*, */WE1*). These signals correspond to the normal control lines found on static memory chips (chip select or */CS*, output enable or */OE*, and write enable or */WE*). In order to generate these memory control signals, the 20-bit address space is divided into four quadrants of 256K each. A *bank control register* for each quadrant determines which of the chip selects and which pair of output enables, and write enables (if any) is enabled when a memory read or write to that quadrant takes place. For example, if a 512K x 8 flash memory is to be accessed in the first 512K of the 20-bit address space, then */CS0*, */WE0*, */OE0* could be enabled in both quadrants.

Figure 3-4 shows a memory interface unit.



**Figure 3-4. Memory Interface Unit**

### 3.2.1 Extended Code Space

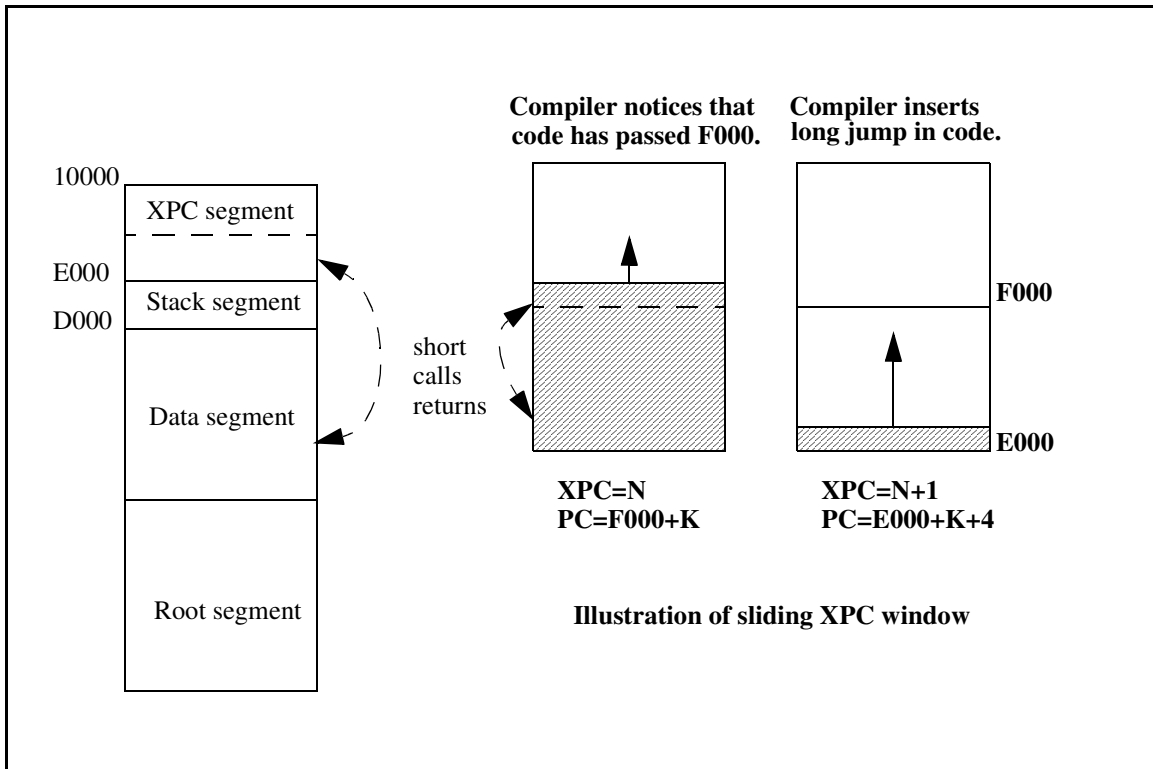
A crucial element of the Rabbit memory mapping scheme is the ability to execute programs containing up to a megabyte of code in an efficient manner. This ability is absent in a pure 16-bit address processor, and it is poorly supported by the Z180 through its memory mapping unit. On paged processors, such as the 8086, this capability is provided by paging the code space so that the code is stored in many separate pages. On the 8086 the page size is 64K, so all the code within a given page is accessible using 16-bit addressing for jumps, calls and returns. When paging is used, a separate register (CS on the 8086) is used to determine where the active page currently resides in the total memory space. Special instructions make it possible to jump, call or return from one page to another. These special instructions are called long calls, long jumps and long returns to distinguish them from the same operations that only operate on 16-bit variables.

The Rabbit also uses a paging scheme to expand the code space beyond the reach of a 16-bit address. The Rabbit paging scheme uses the concept of a sliding page, which is 8K long. This is the XPC segment. The 8-bit XPC register serves as a page register to specify the part of memory where the window points. When a program is executed in the XPC segment, normal 16-bit jumps, calls and returns are used for most jumps within the window. Normal 16-bit jumps, calls and returns may also be used to access code in the other three segments in the 16-bit address space. If a transfer of control to code outside the window is required, then a long jump, long call or long return is used. These instructions modify both the program counter (PC) and the XPC register, causing the XPC window to point to a different part of memory where the target of the long jump, call or return is located. The XPC segment is always 8K long. The granularity with which the XPC segment can be positioned in memory is 4K. Because the window can be slid by one-half of its size, it is possible to compile continuously without unused gaps in memory.

As the compiler generates code resident in the XPC window, the window is slid down by 4K when the code goes beyond F000. This is accomplished by a long jump that repositions the window 4K lower. This is illustrated by Figure 3-5. The compiler is not presented with a sharp boundary at the end of the page because the window does not run out of space when code passes F000 unless 4K more of code is added before the window is slid down. All code compiled for the XPC window has a 24-bit address consisting of the 8-bit XPC and the 16-bit address. Short jumps and calls can be used, provided that the source and target instructions both have the same XPC address. Generally this means that each instruction belongs to a window that is approximately 4K long and has a 16-bit address between E000+n and F000+m, where n and m are on the order of a few dozen bytes, but can be up to 4096 bytes in length. Since the window is limited to no more than 8K, the compiler is unable to compile a single expression that requires more than 8K or so of code space. This is not a practical consideration since expressions longer than a few hundred bytes are in the nature of stunts rather than practical programs.

Program code can reside in the root segment or the XPC segment. Program code may also be resident in the data segment. Code can be executed in the stack segment, but this is usually restricted to special situations. Code in the root, meaning any of the segments other

than the XPC segment, can call other code in the root using short jumps and calls. Code in the XPC segment can also call code in the root using short jumps and calls. However, a long call must be used when code in the XPC segment is called. Functions located in the root have an efficiency advantage because a long call and a long return require 32 clocks to execute, but a short call and a short return require only 20 clocks to execute. The difference is small, but significant for short subroutines.



**Figure 3-5. Use of XPC Segment**

### 3.2.2 Extending Data Memory

In the normal memory model, the data space must share a 64K space with root code, the stack, and the XPC window. Typically, this leaves a potential data space of 40K or less. The XPC requires 8K, the stack requires 4K, and most systems will require at least 12K of root code. This amount of data space is more than sufficient for most embedded applications.

One approach to getting more data space is to place data in RAM or in flash memory that is not mapped into the 64K space, and then access this data using function calls or in assembly language using the LDP instructions that can access memory using a 20-bit address. This is satisfactory for accessing simple data structures or buffers.

Another approach to extending data memory is to use the stack segment to access data, placing the stack in the data segment so as to free up the stack segment. This approach works well for a software system that uses data groupings that are self-contained and are accessed one at a time rather than randomly between all the groupings. An example would