DATA SHEET 6500 Public Key Processor

Hi/fn® supplies two of the Internet's most important raw materials: compression and encryption. Hi/fn is also the world's first company to put both on a single chip, creating a processor that performs compression and encryption at a faster speed than a conventional CPU alone could handle, and for much less than the cost of a Pentium or comparable processor.

Hi/fn, Inc.
750 University Avenue
Los Gatos, CA 95032
info@hifn.com
http://www.hifn.com
Tel: 408-399-3500
Fax: 408-399-3501

Hi/fn Applications Support Hotline:
408-399-3544

Table of Contents

Figures

1 Product Description

Hi/fn's 6500 is a hardware accelerator for public-key cryptography applications such as virtual private networks, secure Web access, secure electronic commerce, and digital certificate services. By implementing 1024-bit modular arithmetic in hardware, the 6500 outperforms even the fastest general-purpose processors, while using only a fraction of the board space and power.

The 6500's modular arithmetic unit is compatible with a variety of public-key algorithms, including DSA, Diffie-Hellman key exchange, and RSA.

The chip also includes a hardware-based true-random-number generator.

Features

- 1024-bit modular arithmetic processor with support for 2048-bit operations
- 200 private-key 1024-bit RSA computations/second
- More than 300 IKE connections/sec
- More than 115 IKE RSA-based X.509 authenticated connections/sec
- True random number generator
- Universal PCI (3V or 5V) and direct 32-bit slave interfaces
- Supports large data bursts
- 160-pin quad flat pack; 3 volt operation
- Internal PLL clock multiplier for single-clock operation

Supported Network Security Protocols

The 6500 processor supports the public-key cryptographic algorithm requirements of the following network security protocols:

- IP Security (IPSec)
- Secure Electronic Transaction (SET)
- Secure Socket Layer/Transport Layer Security (SSL/TLS)
- Public-Key Infrastructure (PKIX)

Part Number	***Speed, MHz***	***Package***
6500-50 PH8	*50*	*160-pin HQFP*
6500-100 PH8	*100*	*160-pin HQFP*

Performance

IKE Handshake (180-bit exponent)	***6500 (100 MHz CLK)***
Handshake: Two 1024-bit Diffie-Hellman operations	*>300 connections/s*
Handshake with authentication: Two 1024-bit Diffie-Hellman operations, 1 RSA sign, 2 RSA verifies	*>115 connections/s*

Figure 1. IKE Performance

Operation	*Time to Complete vs. Key Size*			
	2048	*1024*	*768*	*512*
RSA Private Key	*18.3 ms*	*5.24 ms*	*3.19 ms*	*1.65 ms*
RSA Public Key (exp. = 3)	*0.32 ms*	*0.018 ms*	*0.015 ms*	*0.011 ms*
Diffie-Hellman (exp. = 180 bits)		*1.60 ms*	*1.25 ms*	*0.46 ms*
Diffie-Hellman (exp. = key size)		*9.15 ms*	*7.18 ms*	*2.61 ms*
DSA Sign		*1.83 ms*	*1.52 ms*	*1.22 ms*
DSA Verify		*3.26 ms*	*2.65 ms*	*2.03 ms*

Notes:

1 Performance numbers assume a uniform distribution of ones and zeros in the exponent. Actual performance varies with the Hamming weight of the exponent.

2 Because FIPS 186, the Digital Signature Standard, specifies a maximum modulus of 1024 bits, DSA values are not given for 2048 bits.

Figure 2. Performance vs. algorithm and key size

1.1 System block diagrams

The 6500 can be configured to use either a PCI interface or a direct-access interface. The PCI interface gives the system designer the option of creating a 6500-based PCI card for host systems with available PCI slots, making the 6500 an optional or upgrade feature for the product line. In new host system designs, the 6500 can be soldered directly to the host's main board, in which case either interface may be used.

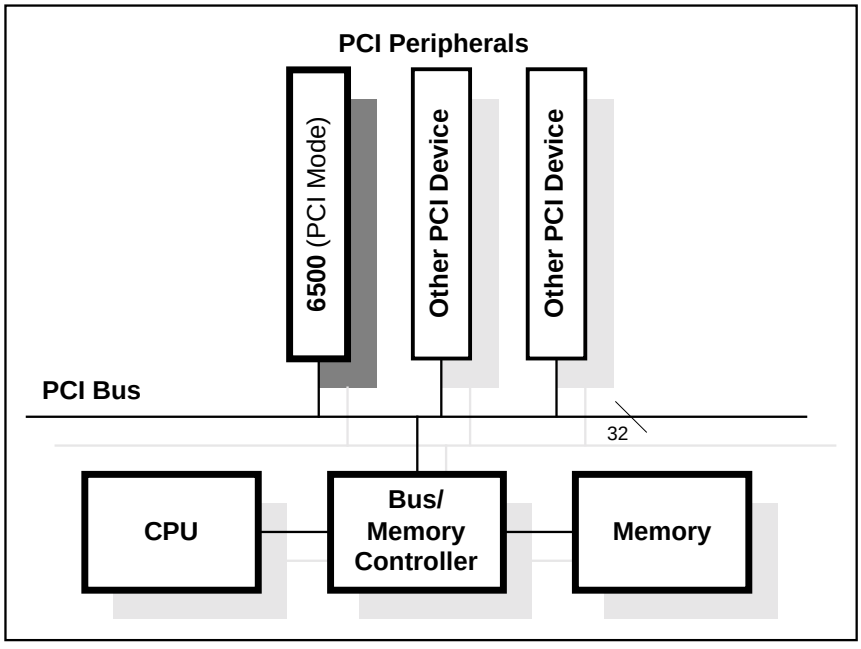


Figure 3. System block diagram, PCI mode

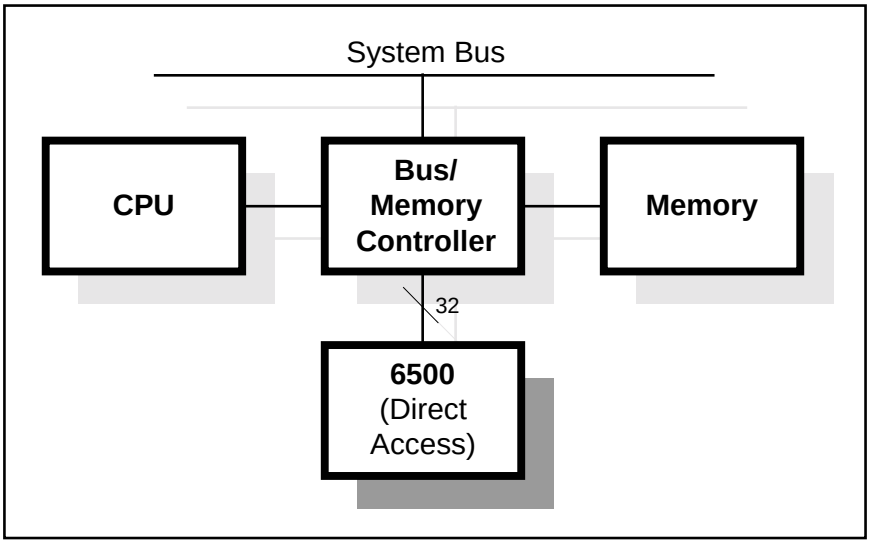


Figure 4. System block diagram, Direct-Access mode

2 Architecture

2.1 Detailed block diagram

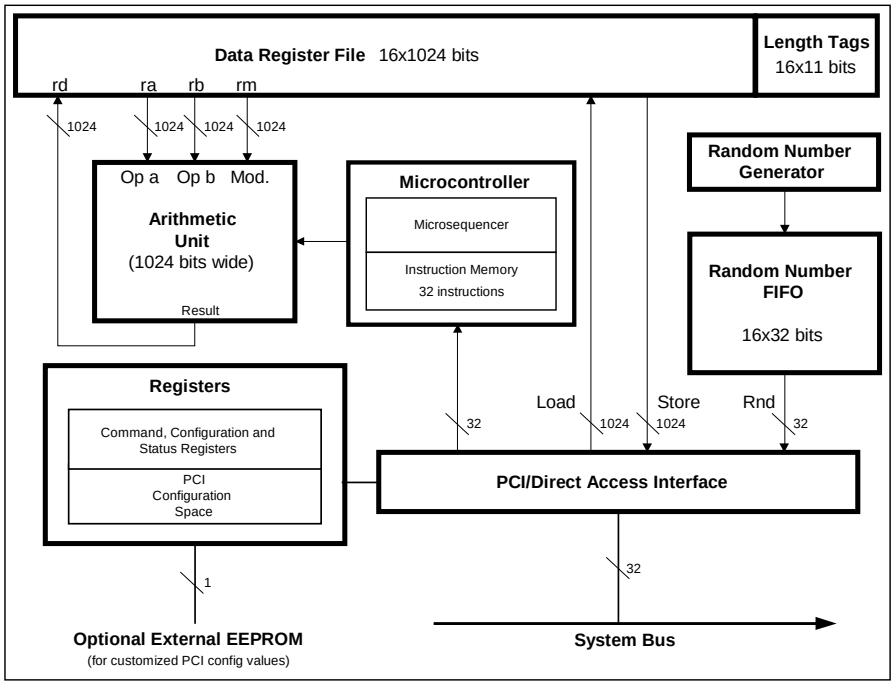


Figure 5. Internal block diagram

2.2 Operation

The 6500 implements modular arithmetic and random number functions, leaving the other tasks involved in public-key encryption (primarily control protocols) to the local host's general-purpose processor. This simplifies operation on both sides of the interface. Adding the 6500 to an existing system requires that only a handful of function calls be rewritten on the host side. Programming the 6500 itself is simple, and a typical application will use only a few 6500 instructions.

(For the purposes of this document, the "host processor" is whatever processor sends commands to the 6500, and isn't necessarily the main processor of the entire system.)

The host loads the 6500's registers, then launches a command, relinquishing the bus. The 6500 will interrupt the host when the command is complete (alternatively, the host can poll a status register). When the command is complete, the host reads the results.

The basic operation of the 6500 is as follows:

Chip initialization

1. Set modes and enable the desired interrupts by writing to the 6500's internal registers.

Program initialization

2. Load the desired sequence(s) of arithmetic operations (microprograms) to instruction memory.

Program Operation:

3. Write the operands to the data register file and the operand lengths to the length tags.
4. Launch a command by writing a microprogram starting address to the Command register.
5. Wait for completion (via polling or interrupts). This is indicated by the Done bit in the Status register. The instruction memory, data registers, and length tags must not be accessed during operation.
6. Read the result from the data register file.
7. Repeat (go to step 3).

2.3 Functional Units

2.3.1 Data Register File

The general-purpose data register file consists of sixteen 1024-bit registers, named r0 through r15. Each register also has an associated *length tag,* indicating the length of the data in the register. Both the registers and the length tags are memory-mapped to a block of consecutive addresses to facilitate block data transfers to and from the host.

The *length tags* are associated with the data register file. The sixteen 11-bit length tags give the size (in bits) of the data in each of the data registers. In general, the shorter the operands, the more quickly the operation completes.

The contents of the register file and length tags are unchanged on a hard or soft reset. They are undefined at power-up.

2.3.2 Other Registers

In addition to the data register file, the 6500 has a set of configuration and status registers. These are described in chapter 5.

2.3.3 Arithmetic Unit

The *arithmetic unit* operates on integers of up to 1024 bits in width. It reads operands from the data register file, operates on them, and writes the results back to the data register file. Thus, operations take the form:

$$rd = ra \,\mathrm{op}\, rb \bmod rm,$$

where *rd, ra, rb,* and *rm* are all data registers, and op is the operation specified by the opcode.

The arithmetic unit has the following modular arithmetic functions: exponentiation, multiplication, reduction, addition, and subtraction. In addition, it performs non-modular two's-complement addition, subtraction, and multiplication, logical left and right shifts, increment, and decrement functions. It can set the length tag of a general-purpose register to any 11-bit value to initialize the length of an operand or override the length of a result.

The arithmetic unit also has a carry flag that is set when there is a carry out of bit 1023 for non-modular addition or when there is a borrow for non-modular subtraction.

On both hard and soft reset, the arithmetic unit is halted and disabled. It must be re-enabled (by setting the Processor Enable bit in the Config2 register) before commands can be executed.

2.3.4 Microcontroller

The *microcontroller* is the 6500's instruction sequencer. It controls the operation of the arithmetic unit. Its instruction set consists solely of arithmetic operations. The lone control operation is the *Done* bit in the instruction word, which tells the microcontroller to stop execution after the current instruction.

The microcontroller is itself divided into two sections: the *microsequencer* and the *instruction memory*. The microsequencer contains the control logic, while the instruction memory consists of thirty-two 32-bit instruction words.

See chapter 6 for more information on 6500 programming.

On both hard and soft reset, the microcontroller is halted and disabled. It must be re-enabled (by setting the Processor Enable bit in the Config2 register) before commands can be executed.

2.3.5 Random Number Generator

The 6500 also contains a hardware random number generator based on internal free-running oscillators whose frequencies drift relative to each other and to the 6500's internal clock. The phase relation of these signals is unpredictable, and

this is used to provide a random bit stream. The random bits are mixed cryptographically with internal state derived from previous random bits, updating the internal state. The output mixing function uses this internal state to produce 32-bit random numbers at a programmable rate. At 100 MHz, the 6500 can generate random data at greater than 3 Mb/s.

The results are pushed into the *random number FIFO*, which is a 16-element FIFO, 32 bits wide. The FIFO is mapped to an array of sixteen 32-bit words. Reading any of these words reads only the top word from the FIFO; the FIFO's internal elements are not memory-mapped. This 16-way replication allows multiple random numbers to be read in a block data transfer.

It is possible to use random numbers more quickly or slowly than they can be generated (hence the FIFO). The Random Number Ready bit in the Status register is set when the random number generator contains at least eight random numbers. An interrupt can also be asserted when the Random Number Ready bit is set. If the host underflows the FIFO, the RNG Underflow bit will be set.

On reset, the random number generator is disabled. It must be re-enabled through the RNG Enable bit in the Config2 register before operation. If the random number FIFO is read while the random number generator is disabled, the result is undefined.

Operation

The random number generator is independent of the arithmetic unit and is enabled and accessed separately.

Setup

1. Enable the random number generator.
2. Wait for the Random Number Ready bit to be set.
3. Read and discard the first sixteen random numbers produced by the generator.
4. Enable the Random Number interrupt, if desired.

Operation

5. Wait for the Random Number Ready bit to be set, or for the associated interrupt to be asserted. This will occur when the Random Number FIFO contains eight or more 32-bit random numbers.
6. Read up to eight random numbers from the FIFO.
7. Repeat (go to step 5).

3 Memory Map and Addressing

3.1 Introduction

Figure 6 shows the memory map for the 6500. The map is the same for both PCI and direct-access modes. The 6500 occupies 32 KB of memory. This space is divided into four windows of 8 KB each. Except for the data register file, the four mappings are equivalent. For the register file, the windows correspond to different combinations of normal/reversed ordering of the bytes within a 32-bit

word and normal/reversed address ordering of the words within a 1024-bit register.

3.1.1 Byte vs. Word Addressing

All addresses are presented here as byte addresses for consistency with PCI, but the 6500 only supports 32-bit accesses, aligned to a 32-bit boundary. Thus, address bits [1:0] should always be set to 0b00 on PCI. This is enforced in direct-access mode, as there are no ADDR [1:0] bits.

3.1.2 Address Notation

The notation for 6500 memory addresses is:

BaseAddress + Window + Offset,

where *BaseAddress* is the PCI BaseAddress if the PCI interface is being used, or zero if the direct-access interface is used,

Window is the offset from the base address, defining the window used. It is 0x0000, 0x2000, 0x4000, or 0x6000 for windows 0, 1, 2, and 3, respectively, and

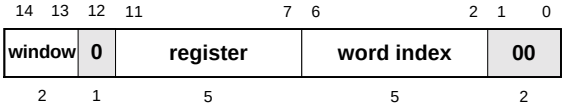
Offset is the offset into the desired window.

Window	*Offset*	*Full Address*	*Description*	*Size, Bytes*	*Organization*	*See Section*
0	*0000-07FF*	*0000-07FF*	*Data Registers (reversed byte, normal word ordering)*	*2048*	*16x1024 bits*	*3.2*
	0800-0FFF	*0800-0FFF*	*Reserved*	*2048*		
	1000-103F	*1000-103F*	*Length Tags*	*64*	*16x32 bits*	*3.3*
	1040-107F	*1040-107F*	*Reserved*	*64*		
	1080-10BF	*1080-10BF*	*Random Number FIFO*	*64*	*16x32 bits*	*3.4*
	10C0-10FF	*10C0-10FF*	*Reserved*	*64*		
	1100-117F	*1100-117F*	*Instruction Memory*	*128*	*32x32 bits*	*3.5*
	1180-1FBF	*1180-1FBF*	*Reserved*	*3648*		
	1FC0-1FFF	*1FC0-1FFF*	*Control/Status Registers*	*64*	*16x32 bits*	*3.6*
1	*0000-07FF*	*2000-27FF*	*Data Registers (reversed byte, reversed word ordering)*	*2048*	*16x1024 bits*	*3.2*
	0800-0FFF	*2800-2FFF*	*Reserved*	*2048*		
	1000-103F	*3000-303F*	*Length Tags*	*64*	*16x32 bits*	*3.3*
	1040-107F	*3040-307F*	*Reserved*	*64*		
	1080-10BF	*3080-30BF*	*Random Number FIFO*	*64*	*16x32 bits*	*3.4*
	10C0-10FF	*30C0-30FF*	*Reserved*	*64*		
	1100-117F	*3100-317F*	*Instruction Memory*	*128*	*32x32 bits*	*3.5*
	1180-1FBF	*3180-3FBF*	*Reserved*	*3648*		
	1FC0-1FFF	*3FC0-3FFF*	*Control/Status Registers*	*64*	*16x32 bits*	*3.6*
2	*0000-07FF*	*4000-47FF*	*Data Registers (normal byte, normal word ordering)*	*2048*	*16x1024 bits*	*3.2*
	0800-0FFF	*4800-4FFF*	*Reserved*	*2048*		
	1000-103F	*5000-503F*	*Length Tags*	*64*	*16x32 bits*	*3.3*
	1040-107F	*5040-507F*	*Reserved*	*64*		
	1080-10BF	*5080-50BF*	*Random Number FIFO*	*64*	*16x32 bits*	*3.4*
	10C0-10FF	*50C0-50FF*	*Reserved*	*64*		
	1100-117F	*5100-517F*	*Instruction Memory*	*128*	*32x32 bits*	*3.5*
	1180-1FBF	*5180-5FBF*	*Reserved*	*3648*		
	1FC0-1FFF	*5FC0-5FFF*	*Control/Status Registers*	*64*	*16x32 bits*	*3.6*
3	*0000-07FF*	*6000-67FF*	*Data Registers (normal byte, reversed word ordering)*	*2048*	*16x1024 bits*	*3.2*
	0800-0FFF	*6800-6FFF*	*Reserved*	*2048*		
	1000-103F	*7000-703F*	*Length Tags*	*64*	*16x32 bits*	*3.3*
	1040-107F	*7040-707F*	*Reserved*	*64*		
	1080-10BF	*7080-70BF*	*Random Number FIFO*	*64*	*16x32 bits*	*3.4*
	10C0-10FF	*70C0-70FF*	*Reserved*	*64*		
	1100-117F	*7100-717F*	*Instruction Memory*	*128*	*32x32 bits*	*3.5*
	1180-1FBF	*7180-7FBF*	*Reserved*	*3648*		
	1FC0-1FFF	*7FC0-7FFF*	*Control/Status Registers*	*64*	*16x32 bits*	*3.6*

Note: *Selecting a particular window affects byte ordering and normal/reversed address bit ordering for the 1024-bit data registers only. For the rest of the memory map, the choice of window has no effect on the 6500.*

Figure 6. Memory map

3.2 Addressing the Data Registers



Bit Field	*Description*
window	*Window selector. The window selects the byte order within a word, and whether the word index is inverted or not.* *00 = reversed byte ordering/normal word index* *01 = reversed byte ordering/reversed word index* *10 = normal byte ordering/normal word index* *11 = normal byte ordering /reversed word index*
register	*Register number, in the range of 0-15. Each data register is 1024 bits wide*
word index	*Selects a 32-bit word within the 1024-bit register. If the window field is 0b01 or 0b11, the 6500 uses the one's complement of the word index, rather than the index itself. This allows registers that are stored on the host with their least-significant word at the highest-numbered address to be transferred as a block with ascending bus accesses. When the window field is 0b00 or 0b10, the word index is not complemented, and registers stored on the host with their least-significant word at the lowest-numbered address to be transferred as a block with ascending addresses.*
0	*These bits are not used and should always be written as 0. Address pins ADDR [1:0] do not exist on the direct-access interface.*

Figure 7. Address format for the data registers

Register	*Offset*
r0	*0000*
r1	*0080*
r2	*0100*
r3	*0180*
r4	*0200*
r5	*0280*
r6	*0300*
r7	*0380*
r8	*0400*
r9	*0480*
r10	*0500*
r11	*0580*
r12	*0600*
r13	*0680*
r14	*0700*
r15	*0780*

Figure 8. Starting address for each data register, by window

3.2.1 Normal vs. Reversed Byte Ordering

Normal vs. reversed byte ordering affects the order in which byte strings are packed into 32-bit arrays. For example, the twelve-byte string, “Hello, world” would be packed into three 32-bit words as follows:

Normal byte order:

Bit Field				***Ascending address***
31:24	***23:16***	***15:8***	***7:0***	
H	*e*	*l*	*l*	*X*
o	*,*		*W*	*X + 4*
o	*r*	*l*	*d*	*X + 8*

Reversed byte order:

Bit Field				***Descending address***
31:24	***23:16***	***15:8***	***7:0***	
l	*l*	*e*	*H*	*X + 8*
W		*,*	*o*	*X + 4*
D	*l*	*r*	*o*	*X*

The 6500 operates in normal byte order internally, but supports both reversed and normal data transfers to the data register file. Only strings and byte arrays are affected by byte ordering. 32-bit quantities are defined unambiguously and are not affected.

Because 32-bit integers are byte order-independent on a 32-bit system, only the 6500’s 1024-bit data registers are affected by byte ordering. (The 11-bit length tags, which would seem to be an exception, are in fact defined as 32-bit quantities with 11 valid bits and 21 reserved bits.)

The data windows in the 6500 allow the byte order to be selected on a per-transfer basis. Windows 0 and 1 support reversed byte order transfers to the data registers, while windows 2 and 3 support normal byte order transfers. The choice of window depends on how operands and results transferred between the host and the 6500 are stored in host memory.

3.2.2 Normal vs. Reversed Word Ordering

Long integers are stored in host memory as an array of 32-bit words. Depending on how the host software allocates arrays, the least-significant 32 bits of a long integer may be at either the bottom of the array, or at the top. For our purposes, we say that data has *normal word ordering* when bits [31:0] of the long integer are at the lowest-numbered address of the array. The data has *reversed word ordering* when bits [31:0] of the long integer are at the highest-numbered address of the array.

The 6500 uses address windowing to support both types. Windows 0 and 2 support normal word ordering. Windows 1 and 3 support reversed word ordering. When a data register is transferred through Windows 1 or 3, the Word Index field of the address is inverted inside the 6500. This reverses the order in which register data is read from or written to the 6500.

3.2.3 Operand Alignment

The modular arithmetic operations are left-aligned, which means that the most-significant word of the operand is in bits [1023:992]. The byte-offset range used by a left-aligned operand is:

Register + (1024 – *Length)*/8

to

Register + 127,

Where *Register* is the starting address of the desired register (see Figure 8) and *Length* is the length of the operand in bits. Thus, transferring a 1024-bit quantity to register r0 involves a byte offset of 0 to 127, while a 160-bit quantity uses offsets of 108 to 127.

The two's complement operations are right-aligned, using the range of

Register

to

Register + *(Length* – 1)/8.

A 160-bit quantity written to register r0 would use offsets of 0 to 19.

Lengths of zero are invalid in either case.

3.3 Addressing the Length Registers

The length registers give the length in bits, of the number stored in each of the data registers. The length is stored as a 32-bit unsigned binary number, with eleven valid bits and 21 reserved bits. The registers are addressed linearly, with address 0x1000 addressing length register 0, and address 0x103c addressing length register 15:

Length Register	***Offset***
L0	*0x1000*
L1	*0x1004*
L2	*0x1008*
L3	*0x100c*
L4	*0x1010*
L5	*0x1014*
L6	*0x1018*
L7	*0x101c*
L8	*0x1020*
L9	*0x1024*
L10	*0x1028*
L11	*0x102c*
L12	*0x1030*
L13	*0x1034*
L14	*0x1038*
L15	*0x103c*

Figure 9. Address of each length register

3.4 Addressing the Random Number FIFO

The random number generator has a 16-element FIFO, 32 bits wide. This FIFO can be accessed from offsets 0x1080-0x10BF. Reading any word in this range will return the top entry of the FIFO. This multiple mapping allows burst transfers of the FIFO contents.

3.5 Addressing the Instruction Memory

The instruction memory consists of thirty-two 32-bit instructions. Execution starts at the address given in the command register and increments until the end of the program is reached. The instruction memory is mapped to offsets 0x1100-0x117F, with instruction 0 at offset 0x1100.

Instruction	***Offset***
i0	*0x1100*
i1	*0x1104*
i2	*0x1108*
i3	*0x110c*
i4	*0x1110*
i5	*0x1114*
i6	*0x1118*
i7	*0x111c*
i8	*0x1120*
i9	*0x1124*
i10	*0x1128*
i11	*0x112c*
i12	*0x1130*
i13	*0x1134*
i14	*0x1138*
i15	*0x113c*
i16	*0x1140*
i17	*0x1144*
i18	*0x1148*
i19	*0x114c*
i20	*0x1150*
i21	*0x1154*
i22	*0x1158*
i23	*0x115c*
i24	*0x1160*
i25	*0x1164*
i26	*0x1168*
i27	*0x116c*
i28	*0x1170*
i29	*0x1174*
i30	*0x1178*
i31	*0x117c*

Figure 10. Address of each instruction register

3.6 Addressing the Control and Status Registers

The control registers are mapped to offsets 0x1FC0-0x1FFF. Each register is mapped to its own 32-bit word.

Offset	*Description*
0x1FC0-0x1FD3	*Reserved*
0x1FD4	*Command Register*
0x1FD8	*Status Register*
0x1FDC	*Interrupt Enable Register*
0x1FE0	*Random Number Configuration*
0x1FE4	*Config1 Register*
0x1FE8	*Config2 Register*
0x1FEC	*Chip ID Register*
0x1FF0-0x1FFF	*Reserved*

Figure 11. Address map for status and configuration registers

4 Signals and I/O

The 6500 operates in one of two bus modes: direct access or PCI. Direct access is a simple burst-oriented bus interface. PCI is an industry standard interface. The 6500 is a 32-bit PCI target device or a 32-bit direct-access slave device. Direct-access would be the interface of choice when integrating the 6500 into a system that doesn't have a PCI bus.

Both interfaces map all accesses to a 32 KB block of memory. See Figure 6 for the memory map.

4.1 Clocks

The 6500 has two clocks, the bus clock and AUX_CLK. The name of the bus clock signal changes with the bus type – it is called PCI_CLK when the PCI interface is enabled, and CLK when the direct-access interface is enabled.

The bus clock is used for I/O. The internal engines of the 6500 (the random number generator, arithmetic unit, and microsequencer) are driven by an internal clock, called ECLK, which can be derived from either the bus clock or AUX_CLK. The bus clock is used by default, but AUX_CLK can be selected by writing a one to the Clock Select bit of the Config1 register (see section 5.5).

The selected clock can then be multiplied by a factor of 1-4x by an internal PLL, or the PLL can be bypassed. The resulting signal is ECLK. Refer to the Config1 register for PLL clock multiple (section 5.5). Irrespective of the whether the PLL is enabled or not, the maximum ECLK clock rate is 100 MHz for a 6500-

100 device and 50 MHz for a 6500-50 device.

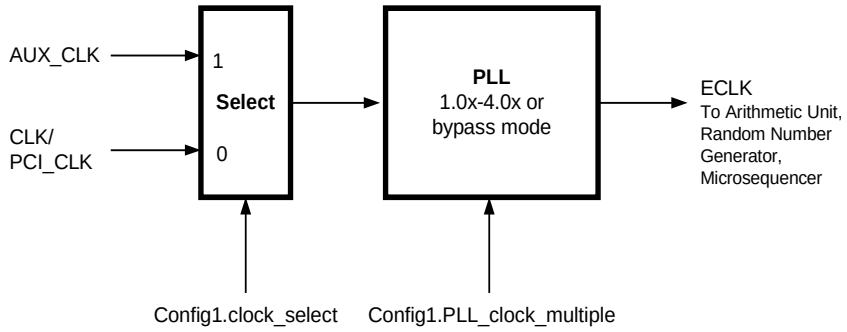


Figure 12. ECLK generation

4.2 Direct-Access I/O

Direct-access I/O is a fast, efficient interface to the 6500. It is pipelined, resembling a synchronous SRAM interface. Direct-access supports single-cycle transfers on a 32-bit bus at speeds up to 66 MHz.

4.2.1 Direct-Access Signal Summary

Pin	***Signal***	***Type***	***Description***
88	*BMODE*	*Input**	*Bus mode = 1 for direct access*
57	*AUX_CLK*	*Input**	*Optional engine clock. (See section 4.1.)*
133, 7, 25, 26, 28, 29, 32, 33, 35, 36, 38, 144,	*ADDR[14:2]*	*Input*	*Register address*
136	*AS#*	*Input*	*Address strobe*
93	*CLK*	*Input*	*Bus clock*
139	*CS#*	*Input*	*Chip select*
98, 99, 101, 103, 104, 107, 108, 112, 118, 119, 122, 124, 125, 127, 128, 132, 152, 153, 155, 158, 159, 3, 4, 6, 10, 11, 13, 14, 17, 18, 21, 22	*D[31:0]*	*I/O*	*Data bus*
96	*IRQ#*	*Open drain output*	*Interrupt request*
143	*OE#*	*Input*	*Output enable*
91	*RST#*	*Input*	*Reset*
140	*R/W#*	*Input*	*Read/write*
** Low-voltage (3.3V) input. Other inputs and I/O signals are 5V-tolerant Please refer to Figure 32 or Figure 33 for Vdd, Vss and NC pins.*			

Figure 13. Direct-access signals

Figure 13 gives a summary of the 6500 signals. Pin configuration is listed in section 7.5. AC specifications are given in section 7.4.

4.2.2 Direct-Access Operation

Reads from the 6500

Figure 14 shows direct-access read timing. On reads, the host asserts CS#, AS#, and R/W#, and places the address in ADDR[14:2]. These signals are synchronous to CLK.

CLK is the bus clock. Inputs are sampled on the rising edge of CLK. The D[31:0] bus has an output delay measured from the rising edge of CLK (see section 7.4.1 for the AC specs of the direct-access interface).

CS# (chip select) enables the data transfer. Data is driven by the 6500 five cycles after any cycle in which CS# is asserted (provided that the OE# is asserted).

AS# (address strobe) clocks in a new address from the ADDR[14:2] bus. CS# and AS# are independent of one another. There is an internal address register that increments after each bus transfer (that is, after each cycle in which CS# is asserted). Asserting AS# overwrites this register with a new address.

OE# is an asynchronous output enable that tri-states the D[31:0] bus when deasserted.

R/W# sets the bus direction. When high, data is read from the 6500; when low, data is written to the 6500.

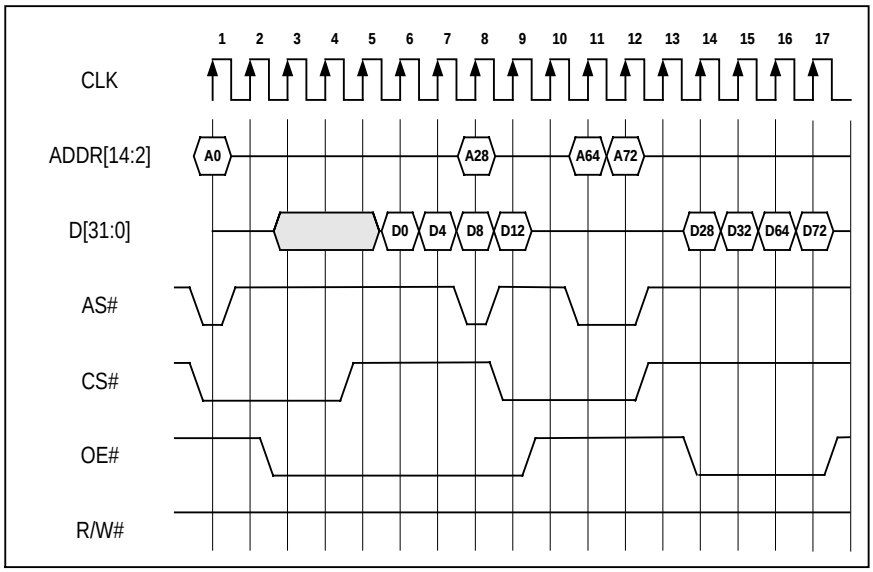


Figure 14. Direct-access read timing

Writes to the 6500

On writes, the pipelining is internal, so there are no cycles between the host's assertion of CS#, R/W#, and D[31:0]. As with reads, AS# and CS# can be asserted together or separately, and the internal address counter increments after every bus cycle, being overwritten on a cycle where AS# is asserted. Data can be written to the chip at one transfer per bus cycle. There is no handshaking on writes. As with reads, one transfer takes place for every cycle in which CS# is asserted.

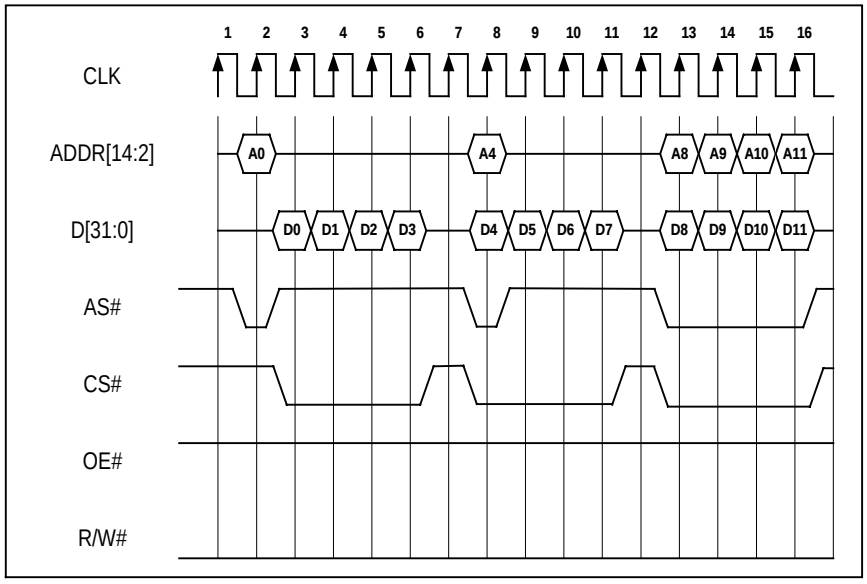


Figure 15. Direct-access write timing

Bus Turnaround

A read-to-write transition requires no dead cycles. The write can start immediately after the cycle that data is read. From the point of view of the CS# signal, this means that CS# must remain deasserted for five cycles between the last read and the first write, while the host memory controller waits for the read to complete. See Figure 16.

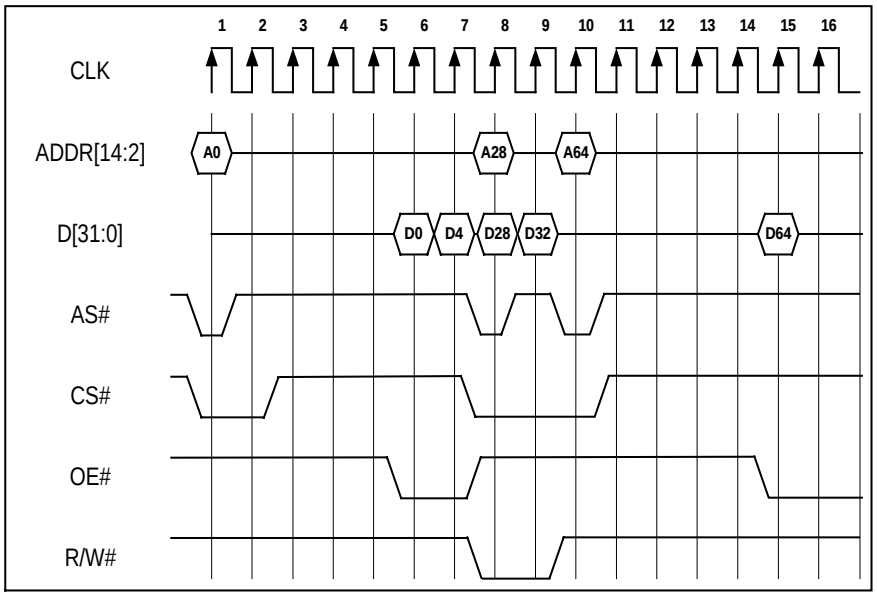


Figure 16. Bus turnaround

4.3 PCI I/O

The 6500 also has a PCI interface that complies with version 2.1 of the PCI specification. The 6500 is a PCI slave device with a 32-bit interface capable of

running at 33 MHz. Figure 17 describes the PCI bus interface. All accesses are mapped to a 32 KB block of memory. See Figure 11 for the memory map.

4.3.1 PCI Signal Summary

Pin	*Signal*	*Type*	*Description*
88	*BMODE*	*Input**	*Bus mode = 0 for PCI*
57	*AUX_CLK*	*Input**	*Optional modular arithmetic and random number generator clock. See section 4.1.*
98, 99, 101, 103, 104, 107, 108, 112, 118, 119, 122, 124, 125, 127, 128, 132, 152, 153, 155, 158, 159, 3, 4, 6, 10, 11, 13, 14, 17, 18, 21, 22	*PCI_AD[31:0]*	*I/O*	*PCI multiplexed address/data bus. During the first clock of the transaction, PCI_AD[31:0] contains a physical address. In later cycles, PCI_AD[31:0] contains data.*
113, 133, 148, 7	*PCI_CBE[3:0]#*	*I/O*	*PCI command/byte enables. During the address phase, they provide the PCI bus command. During the data phase, they are the byte enables.*
93	*PCI_CLK*	*Input*	*PCI system clock. This clock can run at up to 33 MHz.*
141	*PCI_DEVSEL#*	*Output*	*PCI device select, asserted by target when address is decoded.*
136	*PCI_FRAME#*	*Input*	*PCI cycle frame, indicating start and duration of transaction.*
115	*PCI_IDSEL*	*Input*	*PCI initialization device select. Used as chip select during access to one of the devices' configuration registers.*
96	*PCI_INTA#*	*Open-Drain Output*	*PCI interrupt request. Asserted when an enabled interrupt condition occurs. Open-drain.*
139	*PCI_IRDY#*	*Input*	*PCI initiator ready. During a write if asserted, indicates the initiator is driving valid data onto the bus. During a read, assertion indicates the initiator is ready to accept data from the 6500.*
147	*PCI_PAR*	*I/O*	*PCI generated bus parity bit signal that could be generated by either the initiator or target.*
144	*PCI_PERR#*	*Output*	*PCI parity error signal that is asserted on checking PCI_PAR.*
91	*PCI_RST#*	*Input*	*PCI system reset. When in the reset state, all 6500 output pins are put into tri-state and all open-drain signals are floated.*
143	*PCI_STOP#*	*Output*	*PCI bus transfer interruption that indicates to the host processor to abort the transaction in progress.*
140	*PCI_TRDY*	*Output*	*PCI target ready indicating the 6500 is ready to complete the current data transfer.*
44	*EEPROM_CS*	*Output*	*EEPROM chip select*
47	*EEPROM_DI*	*Input**	*EEPROM serial data in*
43	*EEPROM_DO*	*Output*	*EEPROM serial data out*
46	*EEPROM_SK*	*Input**	*EEPROM serial clock*
49	*NO_EEPROM*	*Input**	*EEPROM not present*

** Low-voltage (3.3V) input. Other inputs and I/O signals are 5V-tolerant.*

Please refer to Figure 34 or Figure 35 for Vdd, Vss and NC pins.

Figure 17. PCI signal summary

4.3.2 EEPROM Support

The PCI interface includes a five-pin EEPROM interface that can be connected to an optional 93C48 EEPROM (or equivalent) containing PCI configuration data. If the EEPROM is present, its values override the 6500's built-in defaults. The EEPROM interface consists of the four 93C48 signals (EEPROM_CS, EEPROM_DI, EEPROM_DO, and EEPROM_SK) plus NO_EEPROM. The NO_EEPROM is connected, through a resistor, to either VDD or VSS. When pulled up, it indicates that there is no EEPROM on the board.

The EEPROM contents are shown in .

EEPROM Address	***Value***
00	*Device ID][15:0]*
01	*Not used*
02	*Vendor ID[15:0]*
03 to 07	*Not used*
08	*Class Code[24:8]*
09	*Not used*
0A	*Class Code[7:0], Revision ID[7:0]*
0B	*Not used*
0C	*Not used[7:0],Header Type[7:0]*
0D to 2B	*Not used*
2C	*Subsystem ID[15:0]*
2D	*Not used*
2E	*Subsystem Vendor ID[15:0]*
2F to 3B	*Not used*
3C	*Not used*
3D	*Not used*
3E	*Interrupt Pin[7:0], Not used[7:0]*
3F	*Not used*

Figure 18. EEPROM address map

4.3.3 PCI Configuration Space

The 6500 supports a standard PCI configuration block, which provides the first 64 bytes of Type 0, version 2.1, Configuration Space Header (CSH) to support software-driven "Plug-and-Play" initialization and configuration. The configuration space includes Command, Status, and Base Address Registers (BAR). These registers give the starting address for the 6500's memory-mapped address space, which occupies 8 KB of memory.

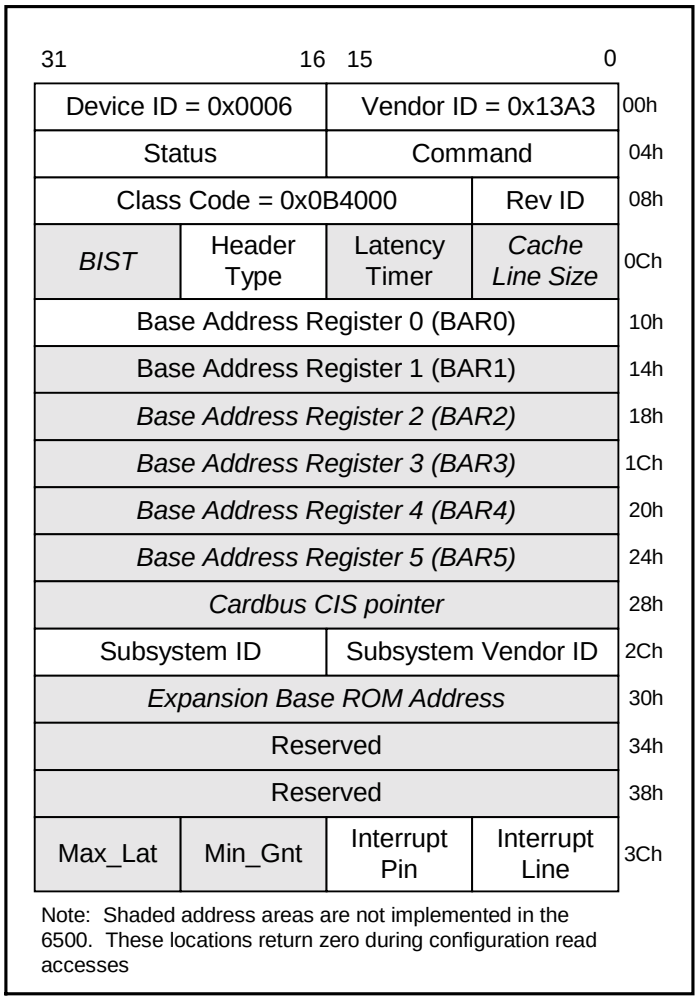


Figure 19. PCI configuration space map

PCI Configuration Space Field	*Default Value (Hex)*	*Read/Write Access*	*PCI Configuration Address (Hex)*	*EEPROM Loadable*
Device ID	*0006*	*Read*	*02-03*	*Yes*
Vendor ID	*13A3*	*Read*	*00-01*	*Yes*
Status	*0280*	*Read*	*06-07*	*No*
Command	*00A8*	*Read/Write*	*04-05*	*No*
Class Code	*0B4000*	*Read*	*09-0B*	*Yes*
Revision ID	*01*	*Read*	*08*	*Yes*
Header Type	*00*	*Read*	*0E*	*Yes*
BAR0	*00000000*	*Read/Write*	*10-13*	*No*
Subsystem ID	*0000*	*Read*	*2E-2F*	*Yes*
Subsys Vendor ID	*0000*	*Read*	*2C-2D*	*Yes*
Interrupt Pin	*01*	*Read*	*3D*	*Yes*
Interrupt Line	*00*	*Read/Write*	*3C*	*No*

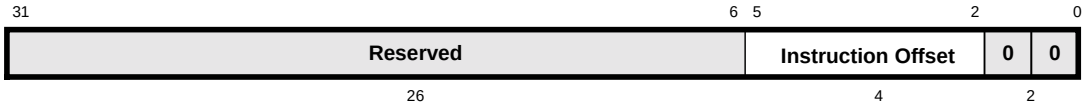
Figure 20. Default PCI Configuration Register Values

The PCI Configuration Register values provided in the table are the default values present in the configuration registers in the absence of an EEPROM.

5 Register Descriptions

5.1 Command Register

Register 5: Command (Read/Write)



Bit Field	*Description*
31:6, 1:0	*Reserved. Must be written as zero.*
5:2	*Instruction Offset.*

Register Description

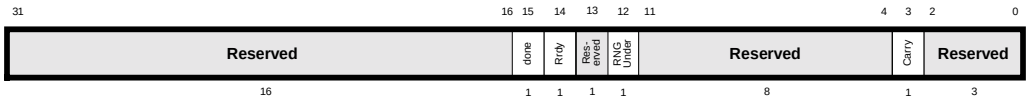
Writing to the Command register causes the microprogram in the instruction registers to begin executing at the register specified in the instruction offset. The offset is a linear value, corresponding to the register number (0-15) where the microprogram starts. Execution will continue until an instruction with a Done bit set is encountered, or the PC wraps around from all 1's to zero.

Reading the Command register returns the offset of the last instruction that has been executed. This is useful in debugging.

On both hard and soft reset, the Command register is set to zero.

5.2 Status Register

Register 6: Status (Read Only)



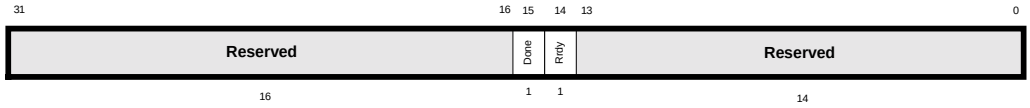
Bit Field	*Description*
31:16	***Reserved.***
15	***Done****. When set, indicates that the 6500 is idle. If clear, 6500 is executing a microprogram. The read-only Status register reports the state of the microsequencer and the random number generator. If the Done bit is not set, the data register files, length tags, instruction memory, command register, and config registers should not be accessed. If this rule is violated, the results are undefined. (See also the Reset bit in the Config1 register.)*
14	***Random Number Ready (Rrdy).*** *When set, the random number FIFO contains at least eight 32-bit random numbers.*
13	***Reserved.***
12	***RNG Underflow.*** *This bit is set when an attempt is made to read an empty Random Number FIFO. When set, this bit is persistent. Disabling the random number generator (by clearing the RNG Enable bit in the Config2 register) will clear this bit.*
11:4	***Reserved.***
3	***Carry.*** *The carry bit from two's-complement instructions. Set or cleared according to the carry of the result.*
2:0	***Reserved****.*

Register Description

This register is set to 0x00008000 (the Done bit is set) on a hard or soft reset.

5.3 Interrupt Enable Register

Register 7: Interrupt Enable (Read/Write)



Bit Field	*Description*
31:16	***Reserved.*** *Must be written as zero.*
15	***Done****. The 6500 will interrupt the host when the Status registers' Done bit is set by one of the engines.*
14	***Rrdy (Random number ready).*** *the 6500 will interrupt the host when the Status register Rrdy bit is set.*
13:0	***Reserved.*** *Must be written as zero.*

Register Description

The Interrupt Enable register controls which conditions in the Status register will cause an interrupt when set. Interrupts can be enabled for the Done and Random Number Ready conditions.

Interrupts can be enabled or disabled while the arithmetic unit and random number generator are running; the host does not have to wait for them to be idle.

This register is set to zero on both hard and soft reset.

5.4 Rngconfig Register

Register 8: Rngconfig (Read/Write)

Bit Field	*Description*
31:12	*Reserved. Must be written as zero.*
11:8	*First prescaler value. See Figure 21.*
7	*Output prescaler value:* *0 = 1024 (default)* *1 = 512*
6:0	*Reserved. Must be written as zero.*

Register Description

*The two prescalers determine the speed at which the random number generator produces output. A new 32-bit random number is generated every (**'first prescaler' * 'output prescaler')** engine cycles. The 'first prescalar' ranges from '2' to '32K'. The 'output prescalar' can take only two different values, i.e. 512 or 1024. This gives the 'first prescaler' and 'output prescaler' product a range of '1K' to '32M' cycles.*

*At its maximum speed, the random number generator opearates at about (100 Mhz * 32 bits) / 1K ≅ 3.0 Mb/s at a 100 Mhz clock.*

*Both registers are set to zero on hard and soft resets, giving a speed of one random number per 32 M cycles, or about, (100 Mhz * 32 bits) / 32M ≅ 95 b/s at a 100 MHz clock.*

First Prescalar	*Description*
000x	*32K (Default)*
0010	*16K*
0011	*8K*
0100	*4K*
0101	*2K*
0110	*1K*
0111	*512*
1000	*256*
1001	*128*
1010	*64*
1011	*32*
1100	*16*
1101	*8*
1110	*4*
1111	*2*

Figure 21. Decoding of the 'first prescaler' field

5.5 Config1 Register

Register 9: Config1 (Read/Write)

Bit Field	*Description*
31:7	***Reserved**. Must be written as zero.*
6	***Clock Select.** If set to one, AUX_CLK is used as the source for ECLK, which drives the microsequencer, arithmetic unit, and random number generator. If set to zero, the bus clock (PCI_CLK or CLK, depending on bus mode) is used. The selected clock is optionally multiplied by 1-4x, as determined by the PLL Clock Multiple field. See section 4.1.*
5:3	***PLL Clock Multiple**. Sets the frequency multiplier between the input clock (selected by the Clock Select field) and ECLK. The PLL should be given time to lock (see PLL Lock Time in section 7.4) before the arithmetic unit and random number generator are enabled. See section 4.1. Under no condition must ECLK be greater than 50Mhz for a 6500-50 device and 100Mhz for a 6500-100 device.* *000: PLL bypass. Bus clock drives arithmetic unit and random number generator directly.* *001: 1.0x CLK* *010: 1.5x CLK* *011: 2.0x CLK* *100: 2.5x CLK* *101: 3.0x CLK* *110: 4.0x CLK* *111: Reserved*
2:1	***Reserved.** Must be written as zero.*
0	***Reset.** If this bit is set when read, it means that the chip is performing a reset operation and should not be accessed, except for reading the Configuration register to monitor this bit.*

Register Description

Writing to the Config1 register has the side effect of causing a soft reset of the chip. This soft reset does not affect the register contents; that is, the write completes normally. On a hard reset, the register is set to 0x00.

5.6 Config2 Register

Register 10: Config2 (Read/Write)

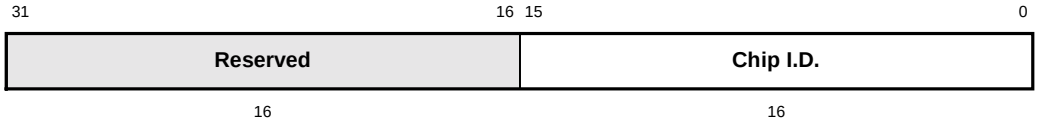
Bit Field	*Description*
31:2	***Reserved**. Must be written as zero.*
1	***Processor enable**. When set, the 6500's arithmetic unit is enabled. When clear, it is disabled. The default on reset is zero. Note that the clock must be stable when the arithmetic unit is enabled. See the PLL Clock Multiple field in section 5.5. When disabled, the arithmetic unit consumes less power.*
0	***RNG Enable**. When set, the random number generator is enabled. When clear, it is shut down. This bit would be set during normal operation, after the PLL had locked. Disabling the random number generator reduces its power consumption.*

Register Description

On hard and soft reset, this register defaults to a value of 0x0.

5.7 Chip ID Register

Register 11: Chip ID (Read Only)



Bit Field	*Description*
31:16	*Reserved*
15:0	*Chip ID*

Register Description

The read-only Chip ID register contains information about the identity and revision level of the 6500. Its value is 0x60xx (the lower eight bits are undefined).

6 Commands and Programming

6.1 Introduction

Before the 6500 can execute commands, the host writes a microprogram to the 6500's instruction memory. To execute a microprogram, the host writes the starting address to the Command register. This write causes the microsequencer to jump to the indicated address and execute instructions until an instruction is encountered that has its Done bit set. When the microsequencer halts in response to the Done bit, the Status register's Done bit is set, and, if enabled, an interrupt is generated. The microsequencer will remain idle until the next write to the Command register.

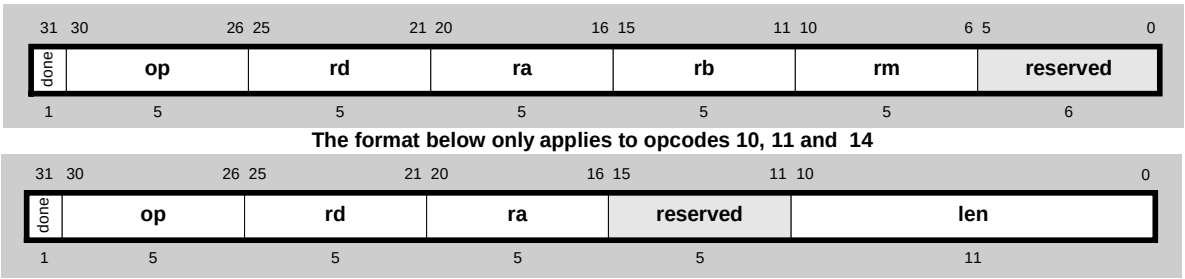
Most registers are not accessible when commands are in progress. This restriction simplifies the programming model, has a negligible effect on performance, and makes register interlocking unnecessary. The Status and Interrupt Enable registers are accessible at all times.

The instruction memory can be reloaded whenever the microsequencer is idle, making it possible for the device to be reprogrammed for different algorithms as needed.

6.1.1 Reserved Fields

Reserved fields in instructions and registers should be written as zeroes. This includes fields which are not used by specific opcodes. On reads, reserved fields should be masked or otherwise ignored.

6.2 Instruction Formats



Bit Field	*Description*
Done	*Indicates the end of the microprogram. This bit is set on the last instruction of the program. The microprogram terminates after the instruction is executed.*
Op	*Instruction opcode. See Figure 23 for opcode assignments. See section 6.4 for a detailed description of each operation.*
ra	*Register number containing operand A.*
rb	*Register number containing operand B.*
rd	*Register number in which to write the result.*
rm	*Register number containing modulus. In the case of 2048-bit multiplication,* rm *holds the high-order product bits*
len	*A literal field containing the length operand for the 'shift' and 'set length' instructions.*
Reserved	*Reserved. This field must be written as 0.*

Figure 22. Instruction Formats

6.2.1 Opcode Summary

Opcode	*Function*	*Description*
0	$rd = ra^{rb} \bmod rm$	*Modular exponentiation*
1	$rd = ra \times rb \bmod rm$	*Modular multiplication*
2	$rd = ra \bmod rm$	*Modular reduction*
3	$rd = (ra + rb) \bmod rm$	*Modular addition*
4	$rd = (ra - rb) \bmod rm$	*Modular subtraction*
5	$rd = ra + rb$	*Addition*
6	$rd = ra - rb$	*Subtraction*
7	$\{carry, rd\} = ra + rb + carry$	*Addition with carry*
8	$\{carry, rd\} = ra - rb - carry$	*Subtraction with borrow*
9	$\{rm, rd\} = ra \times rb$	*2048-bit multiplication*
10	$rd = ra >> len$	*Shift right*
11	$rd = ra << len$	*Shift left*
12	$\{carry, rd\} = ra + 1$	*Increment*
13	$\{carry, rd\} = ra - 1$	*Decrement*
14	$tag[rd] = len$	*Set length tag*
15-30	*Reserved*	*Reserved*
31	*(Math unit reset)*	*No op*

Figure 23. Instruction opcodes

6.2.2 Programming Example

Diffie-Hellman key agreement consists of the operation:

$key = y^x \bmod p$

This would require a single modular exponentiation instruction:

$rd = ra^{rb} \bmod rm$.

By also setting the Done bit on this instruction, we have a complete, one-line 6500 microprogram.

If we selected registers 0, 1, 2, and 3 for *ra*, *rb*, *rd*, and *rm*, our one-instruction program would look like this:

Field	*Value*
Done	*0b1*
Opcode	*0b00000*
ra	*0b00000*
rb	*0b00001*
rd	*0b00010*
rm	*0b00011*

Concatenating these into a control word of:

0b10000000010000000000100011000000,

we store this value into the address of our choice in the instruction memory; for example, address 0. To execute the microprogram, we simply write the operands

to the data register file, write the microprogram start address to the Command register, and wait for the microprogram to complete (by testing the Status register Done bit or waiting for an interrupt). We then read the result from register 2 of the data register file.

More complex calculations (such as RSA with the Chinese Remainder Theorem) can be performed by using longer microprograms and more registers.

6.3 Data Format

Modular arithmetic uses unsigned integers. The other operations use two's-complement integers. In general, modular arithmetic uses *left-aligned* numbers. That is, the most-significant bit of the operand or result will be in bit 1023 of the register. The increment and decrement instructions also use left-aligned numbers. In addition to being left aligned, the MSB of the modulus must always be a one.

Non-modular arithmetic uses *right-aligned* numbers. Bit 0 of the operand or result will be in bit 0 of the register. An operand that is 1024 bits wide is both left- and right-aligned at the same time. For both modular and non-modular arithmetic, operands less than 1024 bits must have the unused bits set to zero.

The length tags have the same meaning in all cases, indicating the length of the operand in bits. The arithmetic unit updates the length tags when it writes results to the data register file.

The 6500 uses normal byte order data formats internally. Thus, the least-significant 8 bits of a 32-bit word and the least-significant 32 bits of a 1024-bit register are at the lowest address in that register's memory space. The 6500's windowed addressing mechanism allows data registers stored under different conventions to be transferred efficiently. (See section 3.2.)

In addition to large integers, a special-case data format is used in non-modular arithmetic, where the carry bit is maintained in the status register.

6.4 Instruction Descriptions

6.4.1 Modular Exponentiation

$$rd = ra^{rb} \bmod rm$$

The data is assumed to be left-aligned. The exponent length must be at least one. The modulus length must be at least five.

The length tag of the result is set to the length tag of the modulus.

6.4.2 Modular Multiplication

$$rd = ra \times rb \bmod rm$$

The data is assumed to be left-aligned. The multiplier (*rb*) length must be at least five. As a special case, a modulus of zero is permissible in this instruction. If a zero modulus is used in a 1024 x 1024 mod 1024 operation, the result is the lower 1024 bits of the 2048-bit product. In another special case, a 512 x 512 mod 512 operation produces the 1024-bit unreduced product if the multiplicand and multiplier are both right-aligned.

The length tag of the result is set to the length tag of the modulus.

6.4.3 Modular Reduction

$rd = ra \bmod rm$

The data is assumed to be left-aligned, and must be at least two bits longer in length than the modulus (tag [*ra*] - tag [*rm*] > 1). For example, to perform this operation with a 128-bit modulus, *ra* would have to be at least 130 bits long. The length of the modulus must be at least one.

The length tag of the result is set to the length tag of the modulus.

6.4.4 Modular Addition

$rd = (ra + rb) \bmod rm$

The data is assumed to be left-aligned. Both addends are assumed to be unsigned integers. The length tag of the modulus must be at least one. The length tags of *ra*, *rb* and *rm* must be equal.

The length tag of the result is set to the length tag of the modulus.

6.4.5 Modular Subtraction

$rd = (ra - rb) \bmod rm$

The data is assumed to be left-aligned. The addend and subtrahend are both assumed to be unsigned integers. The length tag of the modulus must be at least one. The length tags of *ra*, *rb* and *rm* must be equal.

The length tag of the result is set to the length tag of the modulus.

6.4.6 Addition

$\{carry, rd\} = ra + rb$

The addends are assumed to be right-aligned two's-complement integers. The addends must have a length of at least one. The carry is not used in the addition, but is set by the result.

The length tag of the result is set to that of the larger of the two operands.

6.4.7 Subtraction

$\{carry, rd\} = ra - rb$

The addend and subtrahend are assumed to be right-aligned two's-complement integers. The operands must have a length of at least one. The carry is not used in the subtraction, but is set by the result.

The length tag of the result is set to that of the larger of the two operands.

6.4.8 Addition with Carry

$\{carry, rd\} = ra + rb + carry$

As addition, but the Status register carry bit is used as a carry-in bit, and then updated with the carry bit of the result.

The length tag of the result is set to that of the larger of the two operands.

6.4.9 Subtraction with Borrow

$\{carry, rd\} = ra - rb - carry$

As subtraction, but the Status register carry bit is used as a borrow-in bit, and then updated with the borrow bit of the result.

The length tag of the result is set to that of the larger of the two operands.

6.4.10 Multiplication

$\{rm, rd\} = ra \times rb$

This operation multiplies two two's-complement numbers, giving the full product; for example, multiplying two 1024-bit numbers gives the 2048-bit product. The result is written to two registers, *rd* (least-significant 1024 bits) and *rm* (most-significant 1024 bits). The length tag of *rb* must be a multiple of 16.

The data is assumed to be right-aligned.

The operation completes more quickly if the smaller operand is the multiplier *(rb).*

The length tags of both *rm* and *rd* are set to 1024. This choice is somewhat arbitrary.

6.4.11 Shift Right

$rd = ra >> len$

The data in *ra* is shifted right by the number of bits given in *len*. The length of *ra* must be at least one, and *len* must be at least one but less than 1024.

Regardless of the operand length, the entire 1024-bit register is shifted. The length of the result is set to:

$tag[ra] + len$, if $tag[ra] + len < 1024$, or

$tag[ra] + len - 1024$ otherwise.

6.4.12 Shift Left

$rd = ra << len$

The data in *ra* is shifted left by the number of bits given in *len*. The length of *ra* must be at least one, and *len* must be at least one but less than 1024.

Regardless of the operand length, the entire 1024-bit register is shifted. The length of the result is set to:

$tag[ra]+len$, if $tag[ra]+len < 1024$, or

$tag[ra]+len-1024$ otherwise.

6.4.13 Increment

$carry, rd = ra+1$

This operand returns the value of the left-aligned *ra* plus one. Since the operation is left-aligned, the LSB moves with the size of the operand, and the length of the constant "1" must vary with the length of *ra*. This variation is taken care of by the microcontroller. The carry is affected by the addition. The length of the result is the same as that of *ra*. The length of *ra* must be at least one.

6.4.14 Decrement

$carry, rd = ra-1$

This operand returns the value of the left-aligned *ra* minus one. Since the operation is left-aligned, the LSB moves with the size of the operand, and the length of the constant "1" must vary with the length of *ra*. This variation is taken care of by the microcontroller. The carry is affected by the subtraction. The length of the result is the same as that of *ra*. The length of *ra* must be at least one.

6.4.15 Set Length Tag

$tag[rd] = len$

The length tag of register *rd* is set to the value in the *len* field of the instruction word.

6.4.16 No Op

This operation resets the arithmetic unit.

7 Specifications

7.1 Absolute Maximum Ratings

DC Supply Voltage (VDD)	*-0.3 V to +7.0 V*
DC Input Voltage	*-0.3 V to +7.0 V*
DC Input Current	*±10 mA*
Operating Temperature	*0°C to +70°C*
Storage Temperature	*-40°C to +125°C*

Caution: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions may affect device reliability.

7.2 Recommended Operating Conditions

DC Supply Voltage	*+3.0V to +3.6V*
Operating Temperature	*0°C to +70°C*

Figure 24. Recommended operating conditions

7.3 DC Specifications

Symbol	*Parameter*	*Conditions*	*Min*	*Typ*	*Max*	*Units*
VIL	*Low level input voltage (I, SI)*				*0.8*	*V*
	Clock Input (CI)				*0.8*	*V*
VIH	*High level input voltage (I,SI)*		*2.0*			*V*
	Clock Input (CI)		*2.4*			*V*
VH	*Schmitt hysteresis*			*0.8*		*V*
IIL	*Low level input current (I,SI)*	*VIN = VSS*	*-10*			*µA*
	With Pull-up (PI)	*VDD = 3.6V*	*-40*			*µA*
IIH	*High level input current*	*VIN = VDD VDD = 3.3V*			*10*	*µA*
VOL	*Low level output voltage*	*VDD = 3.0V*				
	(02)	*IOL = 2mA*			*0.4*	*V*
	(04)	*IOL = 4mA*			*0.4*	*V*
	(06)	*IOL = 6mA*			*0.4*	*V*
	(08)	*IOL = 8mA*			*0.4*	*V*
VOH	*High level output voltage*	*VDD = 3.0V*				
	(02)	*IOL = -2mA*	*2.4*			*V*
	(04)	*IOL = -4mA*	*2.4*			*V*
	(06)	*IOL = -6mA*	*2.4*			*V*
	(08)	*IOL = -8mA*	*2.4*			*V*
IOZ	*High impedance leakage current*	*VO = VSS VDD = 3.6V*	*-10*			*µA*
IDD	*Quiescent supply current*				*10*	*µA*
CIN	*Input capacitance*	*VDD = 3.3V*		*2.4*		*pF*
COUT	*Output capacitance*	*VDD = 3.3V*		*5.6*		*pF*
CI/O	*Input/Output capacitance*	*VDD = 3.3V*		*6.6*		*pF*
PA	*Power dissipation*	*VDD = 3.3V* *6500-100*		*3.0*		*W*
PA	*Power dissipation*	*VDD = 3.3V* *6500-50*		*1.5*		*W*

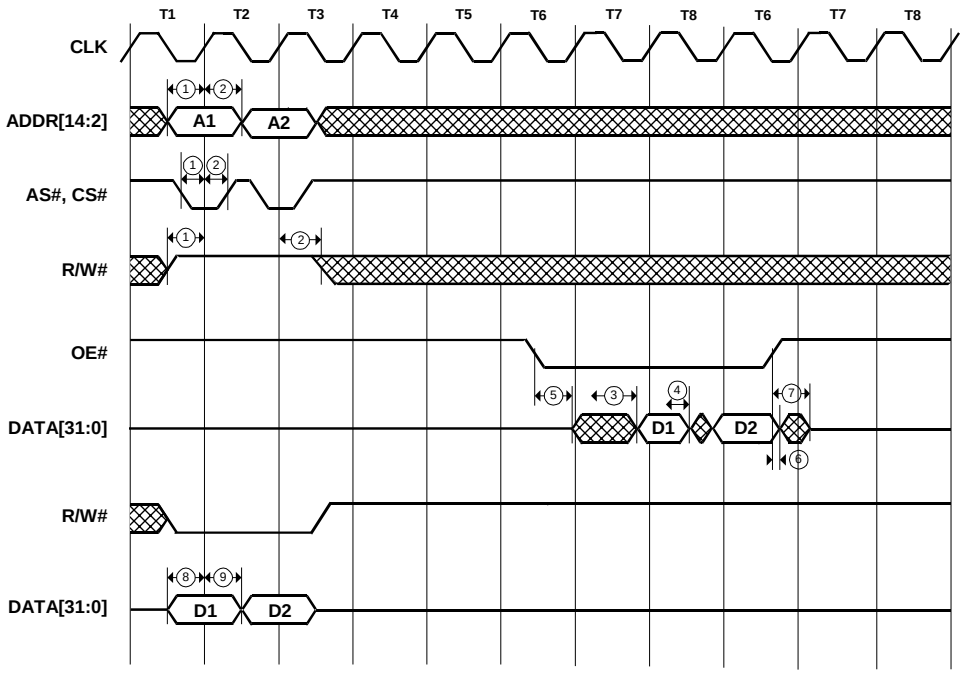
Figure 25. DC electrical characteristics

Symbol	*Parameter*	*Conditions*
CL2	*Load on bus*	*50 pF*
VDD	*Supply voltage*	*3.3V ± 0.3V*
VSS	*Ground potential*	*0V*
TA	*Ambient operating temperature*	*0°C to +70°C*

Figure 26. Test conditions

7.4 AC Specifications

7.4.1 Direct-Access Timing



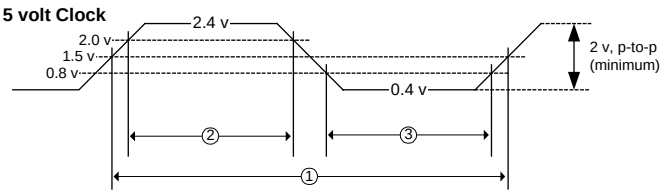
Number	*Description*	*Min (ns)*	*Max (ns)*
1	*ADDR[14:2], AS#, CS#, R/W# setup*	*2.5*	
2	*ADDR[14:2], AS#, CS#, R/W# hold*	*1.5*	
3	*DATA[31:0] output delay (read)*[1]		*15*
4	*DATA[31:0] hold (read)*	*1.0*	
5	*OE# to DATA[31:0] valid (Tzo)*[2]		*22*
6	*OE# to DATA[31:0] hold*	*1.0*	
7	*Toz*		*9*
8	*DATA[31:0] input setup (write)*	*2.0*	
9	*DATA[31:0] input hold (write)*	*2.5*	

Figure 27. Direct-access specifications

[1] Read output delay is actually slightly under 15ns, so that 66MHz system implementations with minimal required setup time may be able to sample the data during the clock in which it is presented.

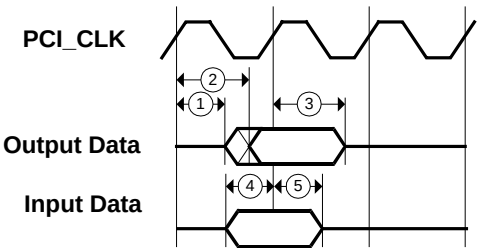
[2] OE# can be asserted earlier so that Tzo delay becomes transparent. An example of this is shown in Figure 14, section 4.2.2.

7.4.2 PCI Timing



Number	*Description*	*Min*	*Max*	*Units*
t_{cyc}	*PCI_CLK Cycle Time*	*30*	∞	*ns*
t_{high}	*PCI_CLK High Time*	*11*		*ns*
t_{low}	*PCK_CLK Low Time*	*11*		*ns*
-	*PCI_CLK Transition Time*	*1*	*4*	*ns*
-	*PCI_RST# Transition Time*	*50*	-	*ns*
-	*PLL Lock Time*	*100,000*		*ns*

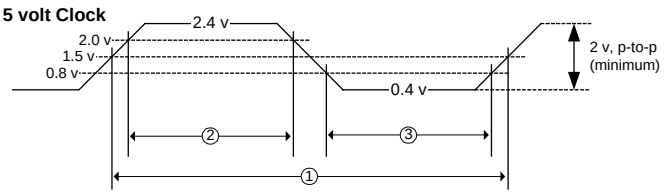
Figure 28. PCI clock and reset parameters



Number	*Description*	*Min*	*Max*	*Units*
1a	*PCI_CLK to Signal Valid Delay – bused signals*	*2*	*11*	*ns*
1b	*PCI_CLK to Signal Valid Delay – point*	*2*	*12*	*ns*
2	*Float to Active Delay*	*2*		*ns*
3	*Active to Float Delay*		*28*	*ns*
4	*Input Setup Time to PCI_CLK – bused signals*	*7*		*ns*
4	*Input Setup Time to PCI_CLK – point to point*	*10,12*		*ns*
5	*Input Hold Time from PCI_CLK*	*0*		*ns*

Figure 29. PCI timing parameters

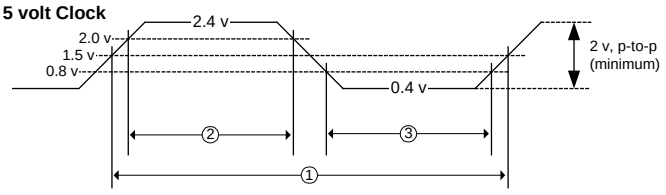
7.4.3 Auxiliary Clock Timing (PCI and Direct Access Modes)



Number	*Description*	*Min*	*Max*	*Units*
t_{cyc}	*AUX_CLK Cycle Time*	*10*	*∞*	*ns*
t_{high}	*AUX_CLK High Time*	*4.5*		*ns*
t_{low}	*AUX_CLK Low Time*	*4.5*		*ns*
-	*AUX_CLK Transition Time*	*1*	*4*	*ns*
-	*PLL Lock Time*	*100,000*		*ns*

Figure 30. Auxiliary clock parameters

7.4.4 CLK Timing (Direct Access Mode)



Number	*Description*	*Min*	*Max*	*Units*
t_{cyc}	*CLK Cycle Time*	*15*	*∞*	*ns*
t_{high}	*CLK High Time*	*7*		*ns*
t_{low}	*CLK Low Time*	*7*		*ns*
-	*CLK Transition Time*	*1*	*3*	*ns*
-	*RST# Transition Time*	*50*		*ns*
-	*PLL Lock Time*	*100,000*		*ns*

Figure 31. Bus clock parameters

7.5 Direct Access Pin Configuration, by Pin Number

Pin	Name	Type	Pin	Name	Type	Pin	Name	Type	Pin	Name	Type
1	NC		41	NC		81	VSS	Ground	121	VDD	Power
2	VSS	Ground	42	VSS	Ground	82	VSS	Ground	122	D21	I/O
3	D10	I/O	43	NC		83	VDD	Power	123	VSS	Ground
4	D9	I/O	44	NC		84	VSS	Ground	124	D20	I/O
5	VDD	Power	45	VDD	Power	85	VSS	Ground	125	D19	I/O
6	D8	I/O	46	NC		86	VSS	Ground	126	VDD	Power
7	ADDR13	Input	47	VSS	Ground	87	NC		127	D18	I/O
8	VSS	Ground	48	VSS	Ground	88	BMODE	(Tie High)	128	D17	I/O
9	VSS	Ground	49	VSS	Ground	89	VSS	Ground	129	VSS	Ground
10	D7	I/O	50	VSS	Ground	90	VSS	Ground	130	VSS	Ground
11	D6	I/O	51	NC		91	RST#	Input	131	VSS	Ground
12	VDD	Power	52	VSS	Ground	92	VSS	Ground	132	D16	I/O
13	D5	I/O	53	VDD	Power	93	CLK	Input	133	ADDR14	Input
14	D4	I/O	54	NC		94	VDD	Power	134	VDD	Power
15	VSS	Ground	55	NC		95	VDD	Power	135	VDD	Power
16	VSS	Ground	56	VSS	Ground	96	IRQ#	Output	136	AS#	Input
17	D3	I/O	57	AUX_CLK	Input	97	VSS	Ground	137	VSS	Ground
18	D2	I/O	58	NC		98	D31	I/O	138	VSS	Ground
19	VDD	Power	59	VDD	Power	99	D30	I/O	139	CS#	Input
20	VDD	Power	60	VDD	Power	100	VDD	Power	140	R/W#	Input
21	D1	I/O	61	VDD	Power	101	D29	I/O	141	VSS	Ground
22	D0	I/O	62	AVDD	Power	102	VSS	Ground	142	VSS	Ground
23	VSS	Ground	63	NC		103	D28	I/O	143	OE#	Input
24	VSS	Ground	64	AVSS	Ground	104	D27	I/O	144	ADDR3	Input
25	ADDR12	Input	65	VSS	Ground	105	VDD	Power	145	VDD	Power
26	ADDR11	Input	66	VSS	Ground	106	VDD	Power	146	VDD	Power
27	VDD	Power	67	VSS	Ground	107	D26	I/O	147	ADDR2	Input
28	ADDR10	Input	68	NC		108	D25	I/O	148	VSS	Ground
29	ADDR9	Input	69	VSS	Ground	109	VSS	Ground	149	VSS	Ground
30	VSS	Ground	70	VSS	Ground	110	VSS	Ground	150	VSS	Ground
31	VSS	Ground	71	VSS	Ground	111	VSS	Ground	151	VSS	Ground
32	ADDR8	Input	72	VDD	Power	112	D24	I/O	152	D15	I/O
33	ADDR7	Input	73	VSS	Ground	113	VSS	Ground	153	D14	I/O
34	VDD	Power	74	VSS	Ground	114	VDD	Power	154	VDD	Power
35	ADDR6	Input	75	VDD	Power	115	VSS	Ground	155	D13	I/O
36	ADDR5	Input	76	NC		116	VSS	Ground	156	VSS	Ground
37	VSS	Ground	77	NC		117	VSS	Ground	157	VSS	Ground
38	ADDR4	Input	78	VSS	Ground	118	D23	I/O	158	D12	I/O
39	VDD	Power	79	NC		119	D22	I/O	159	D11	I/O
40	NC		80	NC		120	NC		160	NC	

Figure 32. Direct access pin configuration, by pin number

7.6 Direct Access Pin Configuration, Alphabetical

Pin	Name	Type	Pin	Name	Type	Pin	Name	Type	Pin	Name	Type
28	ADDR10	Input	**104**	D27	I/O	**34**	VDD	Power	**65**	VSS	Ground
26	ADDR11	Input	**103**	D28	I/O	**39**	VDD	Power	**66**	VSS	Ground
25	ADDR12	Input	**101**	D29	I/O	**45**	VDD	Power	**67**	VSS	Ground
7	ADDR13	Input	**17**	D3	I/O	**53**	VDD	Power	**69**	VSS	Ground
133	ADDR14	Input	**99**	D30	I/O	**59**	VDD	Power	**70**	VSS	Ground
147	ADDR2	Input	**98**	D31	I/O	**60**	VDD	Power	**71**	VSS	Ground
144	ADDR3	Input	**14**	D4	I/O	**61**	VDD	Power	**73**	VSS	Ground
38	ADDR4	Input	**13**	D5	I/O	**72**	VDD	Power	**74**	VSS	Ground
36	ADDR5	Input	**11**	D6	I/O	**75**	VDD	Power	**78**	VSS	Ground
35	ADDR6	Input	**10**	D7	I/O	**83**	VDD	Power	**81**	VSS	Ground
33	ADDR7	Input	**6**	D8	I/O	**94**	VDD	Power	**82**	VSS	Ground
32	ADDR8	Input	**4**	D9	I/O	**95**	VDD	Power	**84**	VSS	Ground
29	ADDR9	Input	**96**	IRQ#	Output	**100**	VDD	Power	**85**	VSS	Ground
136	AS#	Input	**1**	NC		**105**	VDD	Power	**86**	VSS	Ground
57	AUX_CLK	Input	**40**	NC		**106**	VDD	Power	**89**	VSS	Ground
62	AVDD	Power	**41**	NC		**114**	VDD	Power	**90**	VSS	Ground
64	AVSS	Ground	**43**	NC		**121**	VDD	Power	**92**	VSS	Ground
88	BMODE	(Tie High)	**44**	NC		**126**	VDD	Power	**97**	VSS	Ground
93	CLK	Input	**46**	NC		**134**	VDD	Power	**102**	VSS	Ground
139	CS#	Input	**51**	NC		**135**	VDD	Power	**109**	VSS	Ground
22	D0	I/O	**54**	NC		**145**	VDD	Power	**110**	VSS	Ground
21	D1	I/O	**55**	NC		**146**	VDD	Power	**111**	VSS	Ground
3	D10	I/O	**58**	NC		**154**	VDD	Power	**113**	VSS	Ground
159	D11	I/O	**63**	NC		**2**	VSS	Ground	**115**	VSS	Ground
158	D12	I/O	**68**	NC		**8**	VSS	Ground	**116**	VSS	Ground
155	D13	I/O	**76**	NC		**9**	VSS	Ground	**117**	VSS	Ground
153	D14	I/O	**77**	NC		**15**	VSS	Ground	**123**	VSS	Ground
152	D15	I/O	**79**	NC		**16**	VSS	Ground	**129**	VSS	Ground
132	D16	I/O	**80**	NC		**23**	VSS	Ground	**130**	VSS	Ground
128	D17	I/O	**87**	NC		**24**	VSS	Ground	**131**	VSS	Ground
127	D18	I/O	**120**	NC		**30**	VSS	Ground	**137**	VSS	Ground
125	D19	I/O	**160**	NC		**31**	VSS	Ground	**138**	VSS	Ground
18	D2	I/O	**143**	OE#		**37**	VSS	Ground	**141**	VSS	Ground
124	D20	I/O	**91**	RST#	Input	**42**	VSS	Ground	**142**	VSS	Ground
122	D21	I/O	**140**	R/W#	Input	**47**	VSS	Ground	**148**	VSS	Ground
119	D22	I/O	**5**	VDD	Input	**48**	VSS	Ground	**149**	VSS	Ground
118	D23	I/O	**12**	VDD	Power	**49**	VSS	Ground	**150**	VSS	Ground
112	D24	I/O	**19**	VDD	Power	**50**	VSS	Ground	**151**	VSS	Ground
108	D25	I/O	**20**	VDD	Power	**52**	VSS	Ground	**156**	VSS	Ground
107	D26	I/O	**27**	VDD	Power	**56**	VSS	Ground	**157**	VSS	Ground

Figure 33. Direct access pin configuration, alphabetical

7.7 PCI Pin Configuration, by Pin Number

Pin	Name	Type	Pin	Name	Type	Pin	Name	Type	Pin	Name	Type
1	NC		41	NC		81	VSS	Ground	121	VDD	Power
2	VSS	Ground	42	VSS	Ground	82	VSS	Ground	122	PCI_AD21	I/O
3	PCI_AD10	I/O	43	EEPROM_DO	Output	83	VDD	Power	123	VSS	Ground
4	PCI_AD9	I/O	44	EEPROM_CS	Output	84	VSS	Ground	124	PCI_AD20	I/O
5	VDD	Power	45	VDD	Power	85	VSS	Ground	125	PCI_AD19	I/O
6	PCI_AD8	I/O	46	EEPROM_SK	Output	86	VSS	Ground	126	VDD	Power
7	PCI_CBE0#	I/O	47	EEPROM_DI	Input	87	NC		127	PCI_AD18	I/O
8	VSS	Ground	48	VSS	Ground	88	BMODE	(Tie Low)	128	PCI_AD17	I/O
9	VSS	Ground	49	NO_EEPROM	Input	89	VSS	Ground	129	VSS	Ground
10	PCI_AD7	I/O	50	VSS	Ground	90	VSS	Ground	130	VSS	Ground
11	PCI_AD6	I/O	51	NC		91	PCI_RST#	Input	131	VSS	Ground
12	VDD	Power	52	VSS	Ground	92	VSS	Ground	132	PCI_AD16	I/O
13	PCI_AD5	I/O	53	VDD	Power	93	PCI_CLK	Input	133	PCI_CBE2#	I/O
14	PCI_AD4	I/O	54	NC		94	VDD	Power	134	VDD	Power
15	VSS	Ground	55	NC		95	VDD	Power	135	VDD	Power
16	VSS	Ground	56	VSS	Ground	96	PCI_INTA#	Output	136	PCI_FRAME#	Input
17	PCI_AD3	I/O	57	AUX_CLK	Input	97	VSS	Ground	137	VSS	Ground
18	PCI_AD2	I/O	58	NC		98	PCI_AD31	I/O	138	VSS	Ground
19	VDD	Power	59	VDD	Power	99	PCI_AD30	I/O	139	PCI_IRDY#	I/O
20	VDD	Power	60	VDD	Power	100	VDD	Power	140	PCI_TRDY#	I/O
21	PCI_AD1	I/O	61	VDD	Power	101	PCI_AD29	I/O	141	PCI_DEVSEL#	I/O
22	PCI_AD0	I/O	62	AVDD	Power	102	VSS	Ground	142	VSS	Ground
23	VSS	Ground	63	NC		103	PCI_AD28	I/O	143	PCI_STOP#	I/O
24	VSS	Ground	64	AVSS	Ground	104	PCI_AD27	I/O	144	PCI_PERR#	I/O
25	VSS	Ground	65	VSS	Ground	105	VDD	Power	145	VDD	Power
26	VSS	Ground	66	VSS	Ground	106	VDD	Power	146	VDD	Power
27	VDD	Power	67	VSS	Ground	107	PCI_AD26	I/O	147	PCI_PAR	I/O
28	VSS	Ground	68	NC		108	PCI_AD25	I/O	148	PCI_CBE1#	I/O
29	VSS	Ground	69	VSS	Ground	109	VSS	Ground	149	VSS	Ground
30	VSS	Ground	70	VSS	Ground	110	VSS	Ground	150	VSS	Ground
31	VSS	Ground	71	VSS	Ground	111	VSS	Ground	151	VSS	Ground
32	VSS	Ground	72	VDD	Power	112	PCI_AD24	I/O	152	PCI_AD15	I/O
33	VSS	Ground	73	VSS	Ground	113	PCI_CBE3#	I/O	153	PCI_AD14	I/O
34	VDD	Power	74	VSS	Ground	114	VDD	Power	154	VDD	Power
35	VSS	Ground	75	VDD	Power	115	PCI_IDSEL	Input	155	PCI_AD13	I/O
36	VSS	Ground	76	NC		116	VSS	Ground	156	VSS	Ground
37	VSS	Ground	77	NC		117	VSS	Ground	157	VSS	Ground
38	VSS	Ground	78	VSS	Ground	118	PCI_AD23	I/O	158	PCI_AD12	I/O
39	VDD	Power	79	NC		119	PCI_AD22	I/O	159	PCI_AD11	I/O
40	NC		80	NC		120	NC		160	NC	

Figure 34. PCI pin configuration, by pin number

7.8 PCI Pin Configuration, Alphabetical

Pin	Name	Type	Pin	Name	Type	Pin	Name	Type	Pin	Name	Type
57	AUX_CLK	Input	**119**	PCI_AD22	I/O	**53**	VDD	Power	**48**	VSS	Ground
62	AVDD	Power	**118**	PCI_AD23	I/O	**59**	VDD	Power	**50**	VSS	Ground
64	AVSS	Ground	**112**	PCI_AD24	I/O	**60**	VDD	Power	**52**	VSS	Ground
88	BMODE	(Tie Low)	**108**	PCI_AD25	I/O	**61**	VDD	Power	**56**	VSS	Ground
44	EEPROM_CS	Output	**107**	PCI_AD26	I/O	**72**	VDD	Power	**65**	VSS	Ground
47	EEPROM_DI	Input	**104**	PCI_AD27	I/O	**75**	VDD	Power	**66**	VSS	Ground
43	EEPROM_DO	Output	**103**	PCI_AD28	I/O	**83**	VDD	Power	**67**	VSS	Ground
46	EEPROM_SK	Output	**101**	PCI_AD29	I/O	**94**	VDD	Power	**69**	VSS	Ground
1	NC		**17**	PCI_AD3	I/O	**95**	VDD	Power	**70**	VSS	Ground
40	NC		**99**	PCI_AD30	I/O	**100**	VDD	Power	**71**	VSS	Ground
41	NC		**98**	PCI_AD31	I/O	**105**	VDD	Power	**73**	VSS	Ground
51	NC		**14**	PCI_AD4	I/O	**106**	VDD	Power	**74**	VSS	Ground
54	NC		**13**	PCI_AD5	I/O	**114**	VDD	Power	**78**	VSS	Ground
55	NC		**11**	PCI_AD6	I/O	**121**	VDD	Power	**81**	VSS	Ground
58	NC		**10**	PCI_AD7	I/O	**126**	VDD	Power	**82**	VSS	Ground
63	NC		**6**	PCI_AD8	I/O	**134**	VDD	Power	**84**	VSS	Ground
68	NC		**4**	PCI_AD9	I/O	**135**	VDD	Power	**85**	VSS	Ground
76	NC		**7**	PCI_CBE0#	I/O	**145**	VDD	Power	**86**	VSS	Ground
77	NC		**148**	PCI_CBE1#	I/O	**146**	VDD	Power	**89**	VSS	Ground
79	NC		**133**	PCI_CBE2#	I/O	**154**	VDD	Power	**90**	VSS	Ground
80	NC		**113**	PCI_CBE3#	I/O	**2**	VSS	Ground	**92**	VSS	Ground
87	NC		**93**	PCI_CLK	Input	**8**	VSS	Ground	**97**	VSS	Ground
120	NC		**141**	PCI_DEVSEL#	I/O	**9**	VSS	Ground	**102**	VSS	Ground
160	NC		**136**	PCI_FRAME#	Input	**15**	VSS	Ground	**109**	VSS	Ground
49	NO_EEPROM	Input	**115**	PCI_IDSEL	Input	**16**	VSS	Ground	**110**	VSS	Ground
22	PCI_AD0	I/O	**96**	PCI_INTA#	Output	**23**	VSS	Ground	**111**	VSS	Ground
21	PCI_AD1	I/O	**139**	PCI_IRDY#	I/O	**24**	VSS	Ground	**116**	VSS	Ground
3	PCI_AD10	I/O	**147**	PCI_PAR	I/O	**25**	VSS	Ground	**117**	VSS	Ground
159	PCI_AD11	I/O	**144**	PCI_PERR#	I/O	**26**	VSS	Ground	**123**	VSS	Ground
158	PCI_AD12	I/O	**91**	PCI_RST#	Input	**28**	VSS	Ground	**129**	VSS	Ground
155	PCI_AD13	I/O	**143**	PCI_STOP#	I/O	**29**	VSS	Ground	**130**	VSS	Ground
153	PCI_AD14	I/O	**140**	PCI_TRDY#	I/O	**30**	VSS	Ground	**131**	VSS	Ground
152	PCI_AD15	I/O	**5**	VDD	Power	**31**	VSS	Ground	**137**	VSS	Ground
132	PCI_AD16	I/O	**12**	VDD	Power	**32**	VSS	Ground	**138**	VSS	Ground
128	PCI_AD17	I/O	**19**	VDD	Power	**33**	VSS	Ground	**142**	VSS	Ground
127	PCI_AD18	I/O	**20**	VDD	Power	**35**	VSS	Ground	**149**	VSS	Ground
125	PCI_AD19	I/O	**27**	VDD	Power	**36**	VSS	Ground	**150**	VSS	Ground
18	PCI_AD2	I/O	**34**	VDD	Power	**37**	VSS	Ground	**151**	VSS	Ground
124	PCI_AD20	I/O	**39**	VDD	Power	**38**	VSS	Ground	**156**	VSS	Ground
122	PCI_AD21	I/O	**45**	VDD	Power	**42**	VSS	Ground	**157**	VSS	Ground

Figure 35. PCI pin configuration, alphabetical

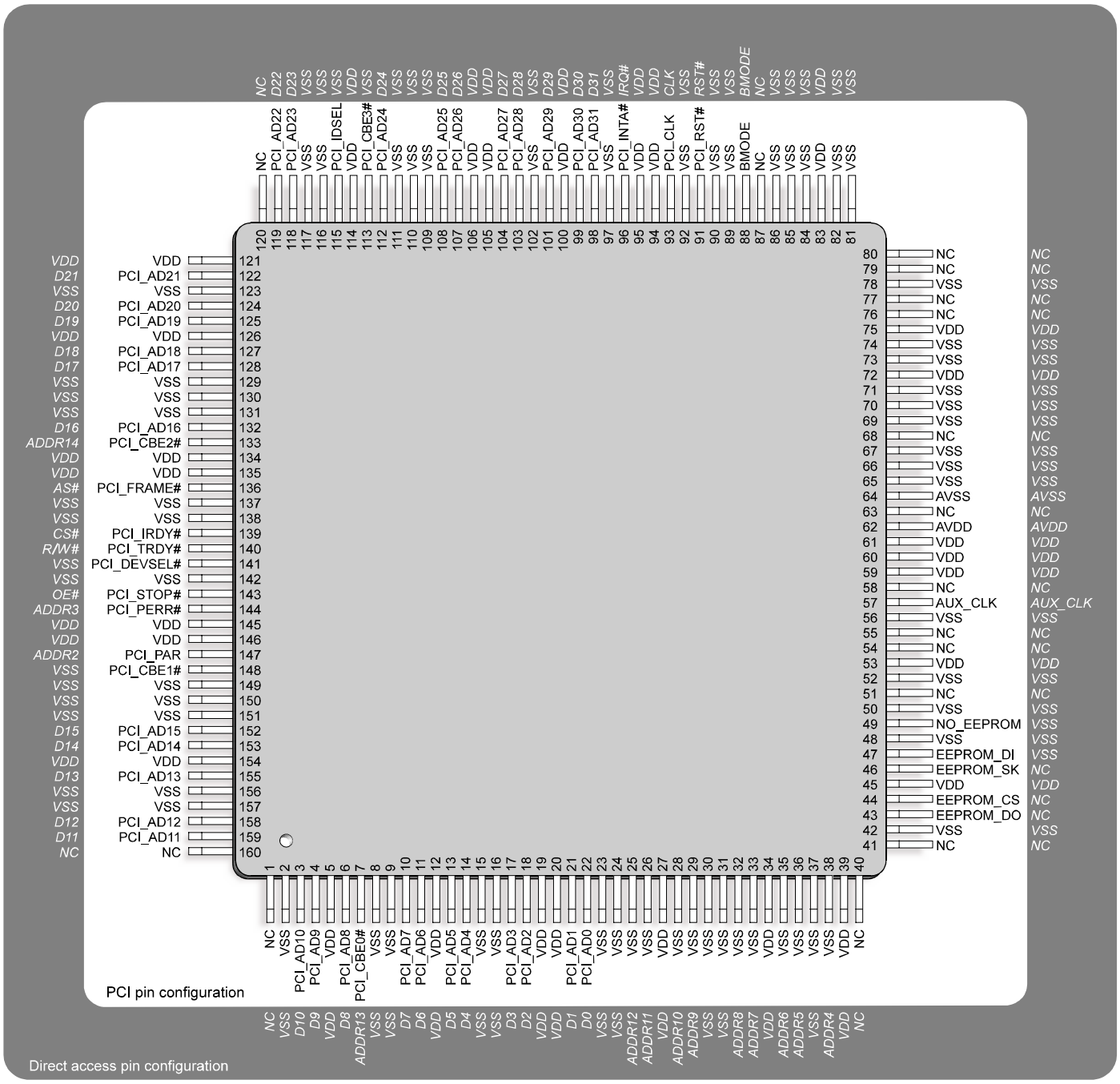
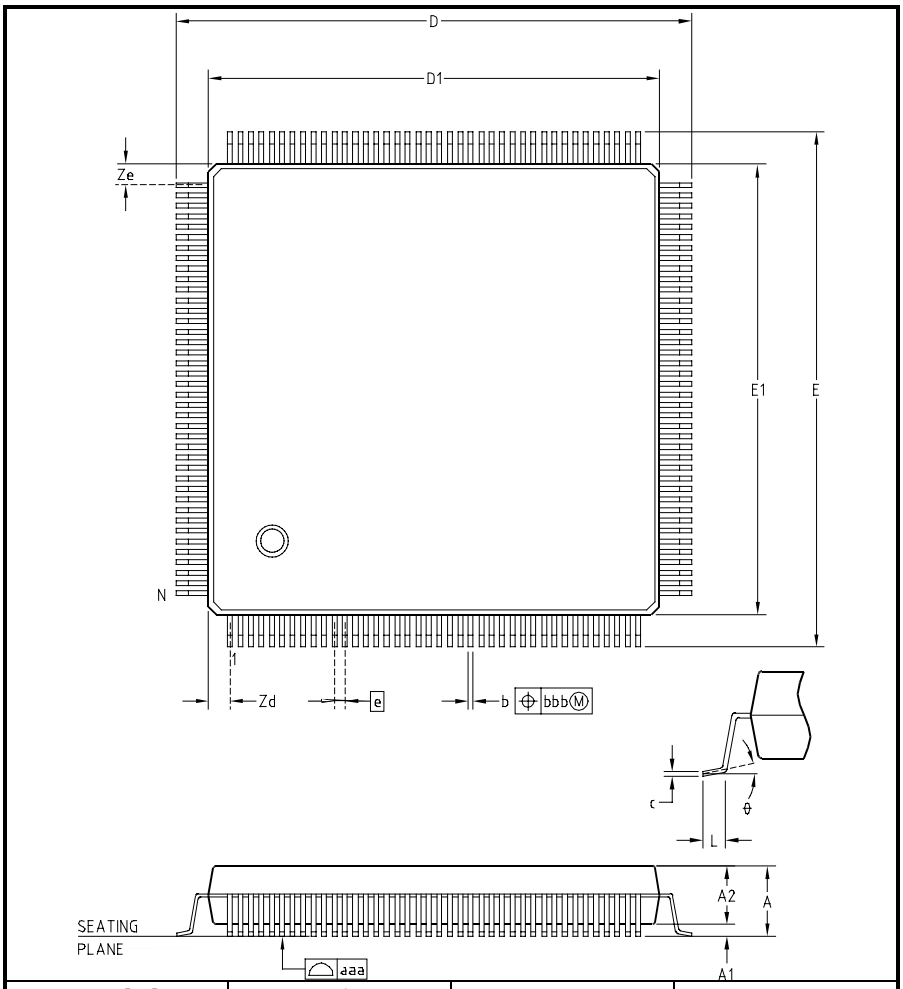


Figure 36. 160-pin HQFP configuration

7.9 Package Dimensions



Symbol	*Min*	*Nom*	*Max*
A			*4.45*
A1	*0.35*	*0.45*	*0.65*
A2	*3.4*	*3.6*	*3.8*
b	*0.2*	*0.3*	*0.4*
c	*0.10*	*0.15*	*0.25*
D	*31.6*	*32.0*	*32.4*
D1	*27.8*	*28.0*	*28.2*
E	*31.6*	*32.0*	*32.4*
E1	*27.8*	*28.0*	*28.2*
e		*0.65*	
L	*0.6*	*0.8*	*1.0*
Zd=Ze		*1.325 TYP*	
θ	*0°*		*10°*
aaa		*0.1*	
bbb	*0.13*	*0.13*	*0.08*
N		*160*	

Figure 37. Package dimensions (in millimeters)