8/30/96

ROM Board Designer's Guide

This chapter provides you with the information that you need to design a ROM board for the N2, to either provide additional functionality to the Newton OS, using ROM extensions or to add additional FLASH devices to increase user storage. Both the hardware and firmware design considerations are described in detail.

Newton OS ROM Extension Overview

The N2's logical ROM consists of a Base ROM image and up to four ROM extensions.The Base ROM image and the four ROM extensions do not correspond to a physical devices, but logical partions that can span multiple physical ROM devices. A single physical ROM device may contain several logical ROM extensions, and a single logical ROM extension may span multiple physical devices.

The Base ROM contains the basic N2 Operating system. ROM extensions (REXs) are binary blocks that can contain packages, device drivers, diagnostics, and other configuration information. The Apple-supplied ROM extension contains the standard N2 applications, device drivers, diagnostic code, and configuration information. You can add up to three additional ROM extensions of your own.

Through the use of REXs, you can add or replace configuration information and diagnostics present in the Apple ROM image. You can also add additional device drivers and built-in applications.

REX Scanning

The Newton OS scans for REXs in the physical address space. The first place that the Newton OS checks is the physical address immediately following the end of base ROM image. This is the location where the Newton OS expects to find the Apple REX. Once the Apple REX is found, the Newton OS then begins the search for Licensee REXs. The first location searched is the physical location immediately following the Appple REX.

The Newton OS will continue to search at the physical address location after each found REX until no more REXs are found.

If the last address location where a valid REX was found offset by the size of that REX is less than ROM_CS_0 + 0x800000, then the Newton OS will begin to search for additional REXs at ROM_CS_0 + 0x800000. If a valid REX is found at that location, the Newton OS will continue to search at the physical address location immediately after each found REX until no more REXs are found.

The next location the Newton OS scans for REXs is at ROM_CS_1. If a valid REX is found at that location, the Newton OS will continue to search at the physical address location immediately after each found REX until no more REXs are found.

If the last address location where a valid REX was found offset by the size of that REX is less than ROM_CS_1 + 0x800000, then the Newton OS will begin to search for additional REXs at ROM_CS_1 + 0x800000.

If more than 4 REXs were found during the scan, the Newton OS will generate a fatal error and won't boot.

The process is shown in Figure 6-1.

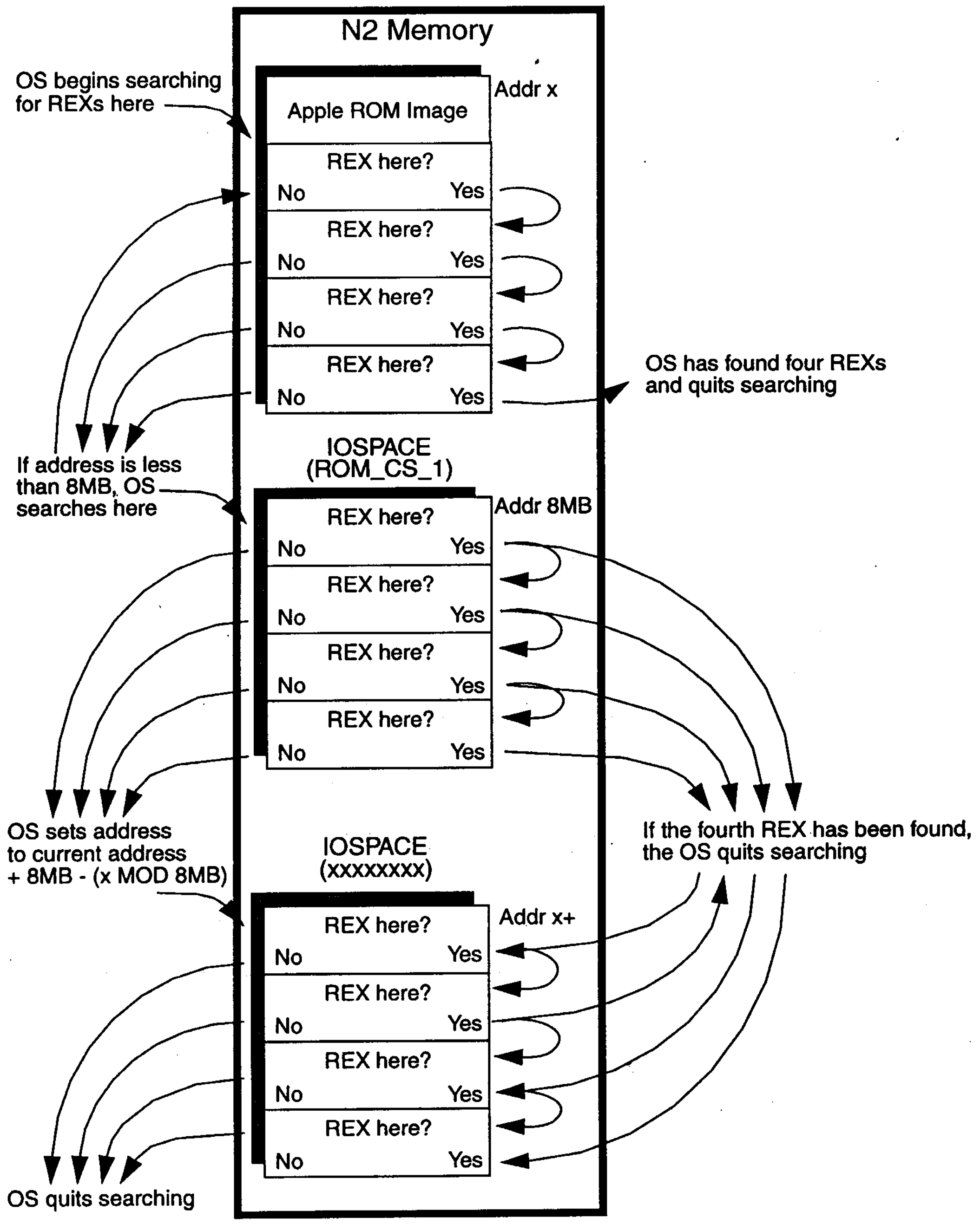
N2 Memory
OS begins searching for REXs here
Apple ROM Image
Addr x
REX here?
No
Yes
OS has found four REXs and quits searching
If address is less than 8MB, OS searches here
IOSPACE (ROM_CS_1)
Addr 8MB
OS sets address to current address + 8MB - (x MOD 8MB)
IOSPACE (xxxxxxxx)
If the fourth REX has been found, the OS quits searching
Addr x+
OS quits searching

Figure 1-1 REX Scan Process

Newton OS ROM_CS_1 Flash Memory Overview

If the address location selected by ROM_CS_1 is not needed for the permanent storage of REXs, it can be used to select additional FLASH devices to be used as User Storage. The Newton OS currently does not support both FLASH devices as User Storage and ROM devices for REX storage in ROM_CS_1 space. Therefore, once a valid REX is found in ROM_CS_1 space, the Newton OS will not scan for FLASH to be used as additional User Storage. This permits the use of FLASH devices as ROM during development or product shipment.

Hardware Design

This section describes how to design and interface a ROM card to the N2's ROM socket. Signal descriptions, Signal locations andSignal timing is provided.

Additional REX Support

If you add additional REXs, with different physical ROM devices, extra address decoding may be necessary to support them. The following subsections explain how to add devices.

Selecting the Device Location

Additional Physical ROM using ROM_CS_0

If the amount of ROM that the Newton OS that you have licensed fits in 8 Mbytes or less, you can add additional chipselect decode logic to select the physical ROM devices containing your REXs. This additional decode logic should ensure that the Apple ROM devices are only selected when ROM_CS_0 is asserted and the address presented on the address bus is in the range between 0 and 0x7FFFFF.

Additionally, the added ROM devices containing your REXs must be 32 bits in width and must only be selected when ROM_CS_0 is asserted and the address presented on the address bus is in the range between 0x800000 and 0xFFFFFF.

Timing should be recalculated to ensure that the modified timing parameters with the addition of the new chipselect logic meet the specifications of the Apple ROM devices.

Timing for the entire bank of devices selected by ROM_CS_0 must be the same. This requires that additional ROM devices that are added must meet or exceed the same timing parameters as the Apple Newton OS ROMs. If slower devices are selected, all devices must be run at the slower speed,which could adversely affect system performance. The Figure 2 shows the addtitional functional logic required to place the new ROM devices in the ROM_CS_0 bank.

Additional Physical ROM using ROM_CS_1

If you wish to use additional devices, ROM_CS_1 can be used to select the devices. The bank of devices must be 32 bits in width. The timing for the bank of ROM selected by ROM_CS_1 is independent from timing of the devices selected by ROM_CS_0, allowing for devices with differnent timing requirements than the Apple ROM's containing the Newton OS. Selecting this location for ROM prevents increasing additional User Storage using the ROM board. The Figure 3 shows the block diagram for this configuration.

Manufacturing a single ROM device

If the Newton OS image that you have licensed and the REX that you generated will fit into a single bank of devices, you can combine the files and manufacture a single set of ROM devices. This bank must be 32 bits in width.

Using this method, you must obtain the Apple OS image and Apple REX. The Figure 4 shows the block diagram for this configuration.

ROM Access Cycles

All ROM devices selected by ROM_CS_0 and ROM_CS_1 are controlled by the CPU bus. Devices selected by ROM_CS_0 have the capability to support single transactions and burst transactions. ROM_CS_1 space only supports single transactions, and does not support burst transactions. ROM_CS_1 also has a external ready line that can beused to dynamically insert waitstates into access cycles.

All accesses begin with a valid address followed by a valid ROM_CS_x signal. The address setup time to ROM_CS_x is programmable in system clock increments. Accesses to ROM_CS_0 space can be programmed to be read only. In this mode, the ROM_IO_RD and ROM_IO_WR signals will not be generated and all accesses are assumed to be Read accesses.

ROM_CS_1 space accesses will always assert either ROM_IO_RD or ROM_IO_WR signals, depending on the access type. Read and Write cycles are differentiated by the signals ROM_IO_WR and ROM_IO_RD. ROM_IO_RD asserted low signifies a Read transaction and ROM_IO_WR asserted low signifies a Write transaction.

Although ROM_CS_1 space can not be programmed to be read only, ROM_IO_WR should never occur if a valid REX has been found in ROM_CS_1 space.

Single Read Transaction

During a single transaction read cycle, a valid address is placed on Addr_24-Addr_2 and the Strong ARM will assert nMREQ. This means the next falling edge of MCLK will be a sequential cycle. All timing is programmable from this falling edge. A programmable number of FCLK cycles from the start of the sequential cycle ROM_CS_x is asserted.This parameter is tAC. If ROM_CS_0 is selected. If Writable ROMs is configured or ROM_CS_1 is selected, ROM_IO_RD is asserted a programmable number of FCLK cycles after the start of the sequential cycle. This parameter tASR. Data is returned by the ROM devices a programmable number of FCLK cycles after the start of the Strong

ARM sequential cycle and is placed on the bus and the cycle terminates. This parameter is tCYC. The figure below shows the timing for a single transaction read. The Strong Arm and digital system Asic signals and timing parameters are shown to help the reader understand the transaction. However, these signals are not available at the ROM Board Connector.

Burst Read Transactions

The ROM_CS_0 bank can also be programmed to support burst read transactions. Read burst transactions occur when the Strong Arm CPU is performing a cache line fill. All other reads will be single read transactions.

A Burst Read Transaction begins like a single read transaction: a valid address is placed on Addr_24-Addr_2 and ROM_CS_0 is asserted a programmable number of FCLK cycles from the begining of the first sequential cycle. The number of FCLKs before ROM_CS_0 is asserted is tAC, the same programmable number of FCLK cycles as the single Single Read Transaction. If Writable ROMs are selected, ROM_IO_RD is asserted a programmable number of clocks after the start of the first sequential cycle. The number of FCLK cycles before ROM_IO_RD is asserted is tASR, the same programmable number of FCLK cycles as as the single Read Transaction. The first data item must be placed on the bus a number of programmable FCLK cycles after the start of the first sequential cycle. This parameter is tCYC, the same number of FCLK cycles as the single Read Transaction.

ROM_CS_0 will then remain asserted. The address will be incremented by 4 and will be placed on Addr_24-Addr_2. Because the lower address bits are not visible of the ROM Board connector, the address will appear to increment by 1. The next data item will be sampled a programmable number of clocks from the falling edge of MCLK that sampled the data item from the last data sampling. This parameter is called tPAC.

Because a cache line fill is 8 words long, there will be 7 tPAC cycles that follow the first data item sampling. The timing for the Read Burst transaction is shown in the figure below.

CALCULATING AND CHANGING TIMING

The Digital System ASIC allows complete flexibilty in changing device timing parameters to accomodate different devices. Timing equations are show in the above timing diagrams (need pointer. do you mean chapter 1?). The following timing parameters can be programmed as multiples of Fclk. All default register parameters can be modified with the REX configuration block entry. ROM_CS_0 is configured differently than ROM_CS_1 space and both entries are shown below:

ROM_CS_0 Configuration Long Word

This section describes the fields in the ROM_CS_0 configuration long word.

Bit	Field	Description
Bits 31:16	Reserved	Set to TBD
Bit 14	Burst Enable	Determines if Burst Cycles are to be enabled.
		1 - Enable Burst Cycles for ROM_CS_0
		0 - Disable Burst Cycles for ROM_CS_0
Bits 13:12	tPAC	The number of FCLK cycles from the sampling edge of MCLK until the next sampling edge of MCLK during a Read burst cycle.
		00 - 5 Clocks
		01 - 2 Clocks
		10 - 3 Clocks
		11 - 4 Clocks
Bits 11:10	tASR	Number of FCLK cycles from the start of a Sequential Cycle until ROM_IO_RD is asserted.
		00 - 0 Clocks
		01 - 1 Clock
		10 - 2 Clocks
		11 - 3 Clocks
Bits 9:8	tASW	The number of FCLK cycles from the start of a sequential cycle before ROM_IO_WR is asserted.
		00 - 0 Clocks
		01 - 1 Clock
		10 - 2 Clocks
		11 - 3 Clocks
Bit 7	Reserved	Set to TBD
Bit 6	Enable RW	Enables the IORD and IOWR strobes during ROM accesses to ROM_CS_0 space
Bits 5:4	tAC	The number of FCLK cycles from the start of the sequential cycle before ROM_CS_0 is asserted.
		00 - 0 Clocks
		01 - 1 Clock
		10 - 2 Clocks
		11 - 3 Clocks

Bit	Field	Description
Bits 3:0	tCYC	The number of FCLK cycles from the start of the sequential cycle until the sampling edge of MCLK during a single Read Transaction or the first sequential access during a burst cycle. 0000 - 16 Clocks 0001 - 1 Clock 0010 - 2 Clocks 0011- 3 Clocks 0100 - 4 Clocks 0101 - 5 Clocks 0110 - 6 Clocks 0111 - 7 Clocks 1000 - 8 Clocks 1001 - 9 Clocks 1010 - 10 Clocks 1011 - 11 Clocks 1100 - 12 Clocks 1101 - 13 Clocks 1110 - 14 Clocks 1111 - 15 Clocks

ROM_CS_0 Bank Default Timing

The Newton OS configures the ROM_CS_0 timing with the following values for the above parameters: Burst Enable = TBD

- tPAC = TBD
- tASR = TBD
- tASW = TBD
- tAC = TBD
- tCYC = TBD

ROM_CS_1 Configuration Bits

This section describes the fields in the ROM_CS_0 configuration long word.

BIT	Field	Description
Bits 31:14	Reserved	Set to TBD
Bits 13:12	tASR	Number of FCLK cycles from the start of a Sequential Cycle until ROM_IO_RD is asserted. 00 - 0 Clocks 01 - 1 Clock 10 - 2 Clocks 11 - 3 Clocks
Bits 11:10	tASW	The number of FCLK cycles from the start of a sequential cycle before ROM_IO_WR is asserted. 00 - 0 Clocks 01 - 1 Clock 10 - 2 Clocks 11 - 3 Clocks
Bits 9:6	Reserved	Set to TBD
Bits 5:4	tAC	The number of FCLK cycles from the start of the sequential cycle before ROM_CS_0 is asserted. 00 - 0 Clocks 01 - 1 Clock 10 - 2 Clocks 11 - 3 Clocks
Bits 3:0	tCYC	The number of FCLK cycles from the start of the sequential cycle until the sampling edge of MCLK during a single Read or Write Transaction. 0000 - 17 Clocks 0001 - 2 Clock 0010 - 3 Clocks 0011- 4 Clocks 0100 - 5 Clocks 0101 - 6 Clocks 0110 - 7 Clocks 0111 - 8 Clocks 1000 - 9 Clocks 1001 - 10 Clocks 1010 - 11 Clocks

BIT	Field	Description
		1011 - 12 Clocks 1100 - 13 Clocks 1101 - 14 Clocks 1110 - 15 Clocks 1111 - 16 Clocks

ROM_CS_1 Bank Default Timing

The Newton OS configures the ROM_CS_1 timing with the following values for the above parameters:

Additional User Storage Support

This section describes adding additional FLASH devices for user storage.

Flash Location

Any FLASH devices added for User Storage, must be placed in the address locations decoded by ROM_CS_1. The memory locations selected by ROM_CS_1 require a 32 bit data path.

Device Requirements

Additionally, the Newton OS supports only Intel Compatible 28f016sa devices for Additional User Storage. These devices can be used in either 8 bitor 16 bit mode. When the FLASH devices are used in 16bit mode, two devices are required, providing an increase of 4 Megabytes of User Storage. When the FLASH devices are used in 8 bit mode, four devices are required, providing an increase of 8 Megabytes of User Storage.

Timing

The Newton OS default timing for the ROM_CS_1 signal allows for the following timing:

Electrical Specifications

Power Requirements

The ROM Card shares power supplies with other add-on devices in the N2 system, so the total power requirement for the entire system should be calculated. The ROM slot uses three power supplies: a 3.3V Digital Supply, a 5V Digital Supply and a 12V Digital Power supply. The amount of power used by each device in the system must be deducted from the total power available to ensure proper operation.

The 3.3V Power Supply Budget

The 3.3V Digital power supply can source a total of 700ma. This supply provides power to the digital circuitry on the Main Logic Board, to the two PCMCIA slots, as well as to the ROM card. The Main Logic Board uses approximately 100ma, so the ROM card must share the remaining 600ma with the two PCMCIA slots.

The 5.0V Power Supply Budget

The 5.0 Digital power supply can source a total of TBD ma. This supply provides power to the TBD.

The 12V Power Supply Budget

The 12Volt power supply can source a total of 65ma. In addition to the ROM card slot, this supply provides power to the Main Logic Board for Flash writes, to the Backlight, and to the two PCMCIA slots. The Main Logic Board uses approximately 5ma, and the Backlight requires 10ma. The remaining 50ma are shared between the ROM card and the two PCMCIA slots.

Power Specifications

This section specifies the DC operating conditions of the Power Sources to the ROM Board.

Supply	Min Voltage	Maximum Voltage
DVCC	3.3V - 5%	3.3V + 10%
D5VCC	5.0 - 10%	5.0 + 10%
D12VCC	12.0 - 5%	12.0 + 5%

ROM Board Signal Descriptions

The signal descriptions are listed in the following tables. The I/O direction is referenced from the ROM Board's view. A "O" specifies that the signal is a output from the card. A "I" specifies that the signal is a input into the card. A "B" specifies that the signal is Bi Directional.

Control Signals

Signal Name	Pin	I/O	Description
ROM_IO_RDY	1	O	This signal is used to add wait states to the access cycle of the ROM.
ROM_IO_INT	3	O	Interrupt Signal
ROM_CS_0	5	I	ROM Bank Chip Select 0
ROM_CS_1	7	I	ROM Bank Chip Select 1
RESET	23	I	Reset
ROM_IO_RD	40	I	ROM READ
ROM_IO_WR	41	I	ROM WRITE
PowerEnable	72	I	This signal is asserted ????

Clock Signals

Signal Name	Pin	IO	Description
SCLK	5	I	3.68 Mhz Clock

Address Signals

Signal Name	Pin	IO	Description
Addr_2	9	I	Address line 2
Addr_3	11	I	Address line 3
Addr_4	13	I	Address line 4
Addr_5	15	I	Address line 5
Addr_6	25	I	Address line 6
Addr_7	27	I	Address line 7
Addr_8	29	I	Address line 8
Addr_9	31	I	Address line 9
Addr_10	33	I	Address line 10
Addr_11	37	I	Address line 11
Addr_12	43	I	Address line 12
Addr_13	45	I	Address line 13
Addr_14	47	I	Address line 14
Addr_15	49	I	Address line 15
Addr_16	51	I	Address line 16
Addr_17	59	I	Address line 17
Addr_18	61	I	Address line 18
Addr_19	63	I	Address line 19
Addr_20	65	I	Address line 20
Addr_21	67	I	Address line 21
Addr_22	69	I	Address line 22
Addr_23	71	I	Address line 23
Addr_24	47	I	Address line 24

Data Signals

Signal Name	Pin	IO	Description
Data_0	2	B	Data bit 0
Data_1	4	B	Data bit 1
Data_2	6	B	Data bit 2
Data_3	8	B	Data bit 3
Data_4	10	B	Data bit 4
Data_5	12	B	Data bit 5
Data_6	14	B	Data bit 6
Data_7	16	B	Data bit 7
Data_8	20	B	Data bit 8

Signal Name	Pin	IO	Description
Data_9	22	B	Data bit 9
Data_10	24	B	Data bit 10
Data_11	26	B	Data bit 11
Data_12	28	B	Data bit 12
Data_13	30	B	Data bit 13
Data_14	32	B	Data bit 14
Data_15	34	B	Data bit 15
Data_16	38	B	Data bit 16
Data_17	40	B	Data bit 17
Data_18	42	B	Data bit 18
Data_19	44	B	Data bit 19
Data_20	46	B	Data bit 20
Data_21	48	B	Data bit 21
Data_22	50	B	Data bit 22
Data_23	52	B	Data bit 23
Data_24	56	B	Data bit 24
Data_25	58	B	Data bit 25
Data_26	60	B	Data bit 26
Data_27	62	B	Data bit 27
Data_28	64	B	Data bit 28
Data_29	66	B	Data bit 29
Data_30	68	B	Data bit 30
Data_31	70	B	Data bit 31

Power & Gnd

Signal Name	Pin	IO	Description
DVCC	17	I	Digital 3.3 Volt VCC Supply
DVCC	35	I	Digital 3.3 Volt VCC Supply
DVCC	53	I	Digital 3.3 Volt VCC Supply
D5VCC	19	I	Digital 5 Volt Supply
D12VCC	21	I	12 Volt VCC

Signal Name	Pin	IO	Description
DGND	18	O	Digital Ground
DGND	36	O	Digital Ground
DGND	54	O	Digital Ground

ROM SIMM Connector Pinout

Pin	Description
1	ROM IO RDY
2	Data_0
3	ROM IO INT
4	Data_1
5	~ROM_CS_0
6	Data_2
7	~ROM_CS_1
8	Data_3
9	Addr_2
10	Data_4
11	Addr_3
12	Data_5
13	Addr_4
14	Data_6
15	Addr_5
16	Data_7
17	DVCC
18	DGND
19	D5VCC
20	Data_8
21	D12VCC
22	Data_9
23	~Reset
24	Data_10
25	Addr_6
26	Data_11
27	Addr_7
28	Data_12

Pin	Description
29	Addr_8
30	Data_13
31	Addr_9
32	Data_14
33	Addr_10
34	Data_15
35	DVCC
36	DGND
37	Addr_11
38	Data_16
39	ROM_IO_RD
40	Data_17
41	ROM_IO_WR
42	Data_18
43	Addr_12
44	Data_19
45	Addr_13
46	Data_20
47	Addr_14
48	Data_21
49	Addr_15
50	Data_22
51	Addr_16
52	Data_23
53	DVCC
54	DGND
55	SCLK
56	Data_24
57	Addr_24
58	Data_25
59	Addr_17
60	Data_26
61	Addr_18
62	Data_27
63	Addr_19
64	Data_28

Pin	Description
65	Addr_20
66	Data_29
67	Addr_21
68	Data_30
69	Addr_22
70	Data_31
71	Addr_23
72	PowerEnable

DC Absolute Maximum Ratings

Address and Data

Signal

I/O Voltage
MinMaxUnits
Data_31-Data_0DVGND - 0.5DGND + 3.6 Volts

Control Signals

SignalI/O Voltage
MinMaxUnits
ROM_IO_RDYTBDTBDVolts
ROM_IO_INTTBDTBDVolts

DC Signal Operating Voltages

Address and Data

SignalInput High VoltageInput Low Voltage
MinMaxMinMaxUnits
Data_31-Data_0.9*DVCCDVCC0.0.1*DVCCVolts

SignalOutput High VoltageOutput Low Voltage
MinMaxMinMaxUnits
Addr_24-Addr_2.8DVCCDVCC0.0.2*DVCCVolts
Data_31-Data_0.8DVCCDVCC0.0.2*DVCCVolts

Control Signals

SignalInput High VoltageInput Low Voltage
MinMaxMinMaxUnits
ROM_IO_RDYTBDTBDTBDTBDVolts
ROM_IO_INTTBDTBDTBDTBDVolts

SignalOutput High VoltageOutput Low Voltage
MinMaxMinMaxUnits
ROM_CS_0TBDTBDTBDTBDVolts
ROM_CS_1TBDTBDTBDTBDVolts
RESETTBDTBDTBDTBDVolts
ROM_IO_RDTBDTBDTBDTBDVolts
ROM_IO_WRTBDTBDTBDTBDVolts
PowerEnableTBDTBDTBDTBDVolts

Clock Signals

SignalOutput High VoltageOutput Low Voltage
MinMaxMinMaxUnits
SCLKTBDTBDTBDTBDVolts

DC Characteristics

Loading Specifications

This section specifies the amount of capacitive load that the is present on Output signals from the board. This section also specifies the maximum capacitive loading that can be present on Input signals to the Rom Board to ensure that the timing will be met. Both specifications are listed for Bidirectional signals.

Address and Data

Addr_24- Addr_2I50pf - (Milton - Connector
Data_31-Data_0B50pf - (Milton - Connector (Output)
5pf + Milton Connector(Input)

Control Signals
ROM_IO_RDYO10pf + Milton Connector
ROM_IO_INTO10pf + Milton Connector
ROM_CS_0I50pf - (Milton Connector
ROM_CS_1I50pf- (Milton Connector
RESETI
ROM_IO_RDI 50pf - (Milton Connector

ROM_IO_WRI 50pf - (Milton Connector
PowerEnableI

Clock Signals
SCLKI

Output Current Current
This section specifies the output current supplied from the drivers of the signals from the Main Logic Board into the ROM Board.

Address and Data

Signal Name	Output high Current	Output Low Current	Units
Addr_24- Addr_2	TBD	TBD	mA
Data_31-Data_0	TBD	TBD	mA

Control Signals

Signal Name	Output high Current	Output Low Current	Units
ROM_CS_0	TBD	TBD	mA
ROM_CS_1	TBD	TBD	mA
RESET	TBD	TBD	mA
ROM_IO_RD	TBD	TBD	mA
ROM_IO_WR	TBD	TBD	mA
PowerEnable	TBD	TBD	mA

Clock Signals

Signal Name	Output high Current	Output Low Current	
SCLK	TBD	TBD	mA

MECHANICAL SPECIFICATIONS

This section provides mechanical requirements for the design of the Rom Board.

ROM Board Mechanical Drawing
This section provides information of the Mechanical specifications for the ROM Board. The specifications provided assume that the card will be placed into a completed N2 system. Some of the dimmensions may change if the card is to be placed in a unit with a different enclosure case.

ROM Board Mechanical Receptical Specification
The receptical used on the N2 device is manufactured by AMP. Please refer to specification #xxxx-xxxxx-xxxxx for further information on the receptical.

Newton ROM Configuration

The Newton ROM consists of a base ROM and from one to four ROM extensions. The BASE ROM contains the ROM code for the base Newton Operating System. Apple supplies a ROM extension that contains the standard Newton applications, device drivers, diagnostic boot code, and RAM allocation information.
You can replace all or part of the Apple ROM extension, and you can also add up to three additional ROM extensions.
This document describes the use, construction, organization and contents of these ROM extensions.

Terminology

Base ROMThe basic Newton system (always present)
ROM ExtensionsBinary blocks that contain packages, diagnostics, and other information.
RexTool for creating ROM extensions.
DumpRex A tool for Dumping the Contents of a REX.

Overview

The generic Newton platform device consists of one base ROM, provided by Apple, and up to four extension blocks.
These extension blocks can contain:
- one or more packages
- one diagnostics code block
- one or more raw blocks
- a RAM allocation table

The only required piece is a RAM allocation table.

Each ROM extension must have a ID field, which is used to determine precedence. The four possible ROM extension blocks have ID numbers 0, 1, 2, and 3. Extension ID 3 has the highest precedence, while extension ID 0 has the lowest.
Apple supplies an extension with ID 0 that contains packages with the standard Newton applications (calendar, names, and so on), the standard Newton diagnostic code, and the standard RAM allocation table. You can replace this extension block entirely, replace parts of it, or supersede parts of it by giving replacement information in an extension with a higher ID number.

How the Newton OS Finds Extensions

The Rex Scanning Algorythm is described in the section above. (GREG- You may want to reformat the above section and add the Scanning Section here to make it flow better. Your Call.)

The Parts of a ROM Extension

A ROM extension can contain any or all of the parts in the following sections.

ROM Extension Header

A ROM extension header looks like this:

```
struct RExHeader {
   ULong signatureA;
   ULong signatureB;
   ULong checksum;
   ULong headerVersion;
   ULong manufacturer;
   Fixed version;
   ULong length;
   ULong id;
   VAddr start;
   ULong count;
   ConfigEntry table[1];// really "count" entries
};
```

signatureA The long value 'RExB'.
signatureB The long value 'lock'.
checksum A checksum for the extension starting with the headerVersion field and extending to the end of the extension. This has the value 0xFFFFFFFF is there is no checksum.

headerVersionA simple long whose value describe the format of the header. This allows Apple to change the header layout in the future. The one shown in this section is headerVersion 1.
manufacturer and version Used by the patching code in the same way they are used for the base ROM.
length The size in bytes of this extension including the header.
idThe ROM extension identifier, used to distinguish each ROM extension. The id is used both as a precedence value, and to associate patch tables and magic pointer tables to specific virtual address slots. All REX's must have unique Id's. See the patching sections later in this document for more information.
startThis gives the ROM extension's Virtual Address.

countThe number of configuration entry tuples immediately following.

ConfigEntry A struct of the form:

```
struct ConfigEntry {
 ULong tag;
 ULong offset;
 ULong length;
};
```

It consists of a configuration entry, containing a tag, an offset, and a length. The tag identifies what kind of configuration block can be found at the specified offset. (There may be additional types of configuration blocks in the future.) The length is the length of that configuration block. All configuration blocks must start on a longword boundary.

Newton OS Supported Configuration Blocks
This section describes the Configuration Blocks that are currently recognized by the Newton Operating System that can be included in a Licensee REX. This list does not contain all values currently supported by the OS. Only Configuration Blocks that may be added to Licensee REXs.

Ram Allocation Configuration Block.

This is used by the system to divide RAM among various system resources. RAM Allocation Configuration Entries are identified by the Configuration Entry Tag:

```
const ULong kRAMAllocationTag = (ULong) 'ralc';
```

The Apple ROM extension supplies a default table, so if you don't replace ROM extension 0, you can use the default table.

The RAM allocation structures look like the following.

```
const ULong kRAMAllocTableVersion = 1;
struct RAMAllocEntry {
ULong min;
ULong max;
ULong pct;
};
struct RAMAllocTable {
ULong version;
ULong count; // Must =3
RAMAllocEntry table[3];
};
```

The current Newton OS supports three entries. These entries represent:
1.RAM user store
2.Frames heap
3.ROM Domain Manager working set

For each entry, the OS computes the amount of RAM to give out by multiplying the total system RAM by the pct field. It then forces the result to be at least min bytes. (If min is zero this step is skipped.) It then forces the result to be no more than max bytes. If max is zero this step is skipped.) The result is given to the client as the amount of space in bytes to allocate. Only a small number of configurations are supported. These values will be described at a later date.

Package Lists

Each ROM extension may contain a list of packages. Package List Configuration Entries are identified by the Configuration Entry Tag:

```
const Ulong kPackageListTag = (Ulong) 'pkgl';
```

Each package follows the previous package immediately (that is, there are no alignment restrictions). The OS looks for new packages until it reaches the end of a package and it isn't followed by a valid package header, or it has reached the end of the package list block.
All places in the system that scan the package list automatically scan all the ROM extensions for packages. Two or more packages can have the same name, but only one package with a given name is loaded. The loaded package is the one found in the ROM extension with the highest ID. That is, if a Names package is found in ROM extension 0 and another Names package is found in ROM extension 3, the Names package from ROM extension 3 is loaded. This allows you to, for example, replace standard applications without modifying the Apple ROM extension.

Diagnostic Code

Each ROM Extension may contain one Diagnostic Routine. A Diagnostic Routine Configuration Entry is identified by the Configuration Entry Tag:

```
const Ulong kDiagnosticsTag = (Ulong) 'diag';
```

Diagnostic code is C/C++ code linked to its actual physical address. The top of the code must be the entry point into a single routine. The OS jumps into this routine on cold booting. Note that the MMU has not been enabled at that time so the diagnostic code must be linked to its actual physical address, and the OS has not sized or otherwise configured RAM so the diagnostic code must run without RAM. The routine should return immediately if diagnostic mode is disabled (a condition the diagnostic code must somehow determine without using RAM or the MMU). Otherwise, the diagnostic code can do as it pleases (including setting up RAM or MMU code).
There can be more than one diagnostic block, but the OS invokes only the one found in the ROM extension with the highest ID. That is, if a diagnostic block is found in ROM extension 0 and another diagnostic block is found in ROM extension 3, the diagnostic block from ROM extension 3 is loaded. This allows you to, for example, replace the standard diagnostic block without modifying the Apple ROM extension.
You build a diagnostic block using the Diagnostic DDK. See the documentation for the Diagnostic DDK for more information.

Raw Blocks

A raw block contains licensee-specific data. The OS does not attempt to use or interpret the contents of a raw block. If any Raw Blocks are to be used, Licensing Support needs to ensure that the tag value is not used by any other system resource.

API Description

Here are descriptions of the ROM routines you can use in your code.

ROM Routines

These are all declared in VirtualMemory.h. The layout for ROM extensions is in ROMExtensions.h.

VAddr GetPackageList(ULong id);

Returns a VAddr of the package list contained in the id'th ROM extension, or nil if there is none available (package list or id'th ROM extension).

VAddr GetRExConfigEntry(ULong id, ULong tag, ULong* length);

Returns the pointer to the specified ROM extension configuration block, or nil if there is none (configuration block or ROM extension). length is set to the length or the block, or zero.

VAddr GetLastRExConfigEntry(ULong tag, ULong* length);

Returns the pointer to the last specified ROM extension configuration block, or nil if there is none. length is set to the length or the block, or zero. The last block is defined as the configuration block residing in the ROM extension block with the highest ID.
RExHeader* GetRExPtr(ULong id);

Returns a pointer to the id'th ROM extension's header, or nil if there is none.

DumpRex

DumpRex is a MPW tool that dumps the header information of a REX. The tool takes as input a REX file and dumps the header file contents in ASCII readable format to stdout.

DumpRex Command Line Syntax
Here is the command-line syntax.

DumpRex [Rex-File]
Rex-FileSet the name of the REX file that you wish to display the header information.

Sample Output
Shown below is a sample of the output from Dumping a Sample Apple REX is shown below.

```
### DumpREx of "mac Drive:English:Senior CirrusNoDebug high"
#   signature: 'RExBlock' (0x524578420x6C6F636B)
#    checksum: 0xCDD5 (unverified)
#  hdrVersion: 1
# manufacturer: 0x1000000
#     version: 0x20000
#      length: 821524 (0xC8914)
#          id: 0
#       start: 0x703978
#       count: 9
# tag 'dio ', addr 0x703A0C, length 584, offset 148
# tag 'gpio', addr 0x703C54, length 648, offset 732
# tag 'ralc', addr 0x703EDC, length 44, offset 1380
# tag 'pkgl', addr 0x703F08, length 814760, offset 1424
# tag 'pad ', addr 0x7CADB0, length 592, offset 816184
# tag 'ptpt', addr 0x7CB000, length 1024, offset 816776
# tag 'glpt', addr 0x7CB400, length 1024, offset 817800
# tag 'pad ', addr 0x7CB800, length 2048, offset 818824
# tag 'fexp', addr 0x7CC000, length 652, offset 820872
```

Rex

Rex is a MPW tool used to build ROM extensions. Most of its input is specified in the configuration file that may be provided on the command line or through stdin. The output is a ROM image file. You can specify the name of the output file on the command line or the file can be emitted from stdout.
The configuration file consists of a series of clauses. Each clause starts with an identifier. The identifier is followed by configuration data appropriate to the identifier. Data consists of one of the following:
- integers, which are specified as decimal or hex [0x] numbers or 'xxxx' 4-char constants
- strings
- raw data blocks, which are specified as hex digits surrounded by angle brackets
- option lists, which are identifier/data pairs surrounded by braces

Identifiers are case-insensitive.
Comments and white space are allowed anywhere, including between bytes of hex data. As with C++ code, you precede comments with // or surround them with /* and */.
Rex takes all the input configuration information, processes as necessary, then concatenates it all and builds the ROM extension header.

Rex Command Line Syntax

Here is the Rex command-line syntax.

```
rex [-help] [-o output-file] [config-file]
```

-help print this message

-o output-file set the name of the output file
config-fileset the name of the configuration file

Configuration File Syntax

Here is the current set of clauses Rex accepts in the configuration file.

id integer ROM extension ID (0-3) All Rom Extensions must have Unique Identifiers. This field is Mandatory. The Apple Rom Extension, will always have ID 0.

start integer Virtual address of the start of the ROM extension. Depending on where the ROM extension exists

manufacturer integer Manufacturer ID. These ID's must be unique and are obtained from Apple Newton Licensing Support.

version integer Version field of header

package string Add a package file
string (OPTIONAL) Patch info file for the package
diagnostics string Add the diagnostics file (this currently just does a raw copy of the file.)

RAMAllocationAdd a RAM allocation block
option list:min integer
max integer
percent integer (0-100)
block integer Specify a tag (usually a 'xxxx'-style integer)
block stringRead data from the file with the specified name
block raw-dataAdd the given raw data

Sample Configuration File

Here is a sample configuration file.

```
/*
A test Rex file (this is a comment)

rex -o testout test
*/
// double slashes work as comments too

manufacturer 0x100// Integers can be hex...[Required Field]
version 1// or decimal.
// or 'xyzt' form (see "block" below)
id 2// REx ID [Required Field]
start 0x1000000// Virtual Start Address [Required Field]

// Multiple package clauses are allowed.
// Each just appends another package.
package "::magicfolder:ntest.mp"

// An optional additional string specifies the patch info file
package "::magicfolder:killpass.mp" "::magicfolder:killpass.pt"

diagnostics "test"// Only one diagnostics file is allowed per REx.

// RAMAllocations' percent is 0-100 (translated into 0-1023 for you)
RAMAllocation { min 10 max 50 percent 25 }
RAMAllocation { min 10 max 50 percent 25 }
block 'test' "test"// "block" lets you specify arbitrary
block 'frob' <123456>// blocks as a file or raw data.
block 'frob' <555121>// Additional ones get appended.
```