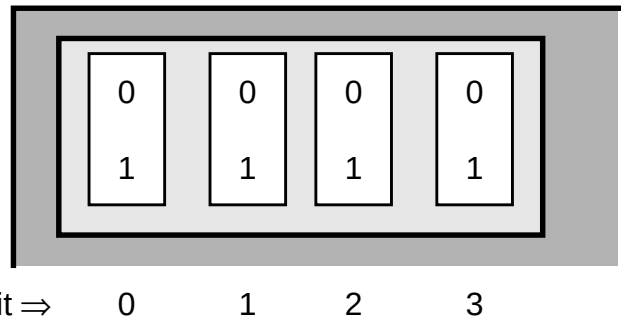
Boot Code

On reset the system goes through a short initialisation sequence and then despatches control to one of up to sixteen applications. The application is selected using the DIP switch in the rear, right corner of the PCB, as shown below. (This may be read more easily by turning the board around.)



The application entered is governed by a control block in the ROM. This is located at address &004000 with entries which are each &100 bytes long. The format of each entry is:

Offset	Function	Comments
00	“CODE” Magic number	
04	Flags	See below
08	RAM image start address in ROM	“Address” is relative to ROM start
0C	RAM image length	In bytes but whole words copied
10	ROM image start address	“Address” is relative to ROM start
14	ROM image length	Used only for mapping/checksum
18	ROM execution offset	Offset is from image start
1C	ROM execution CPSR	Defines execution mode
20	Spartan definition address	Points to definition block
24	Spartan definition length	Used only for mapping
28	Virtex definition address	Points to definition block
2C	Virtex definition length	Used only for mapping
30 +	LCD start-up message	Zero terminated string Understands control characters: LF (0A) – change lines FF (0C) – clear screen CR (0D) – start of line

The remaining space (until the next used entry) is available for user descriptions, version numbers etc.
In the event of the selected boot not being valid a message is output to the LCD (if present) and the LEDs are flashed.
Boot #0 is normally used for the on-board Flash loader which allows the ROM to be configured via a serial line. This does not allow the boot code to be overwritten but does allow the boot table to be changed. Boot #0 can therefore be redefined. In the event of a data table corruption the on-board loader can still be reached by resetting with boot #0 selected whilst holding down the right-hand button on the PCB. (This can also allow seventeen boot options, if desperate!)
The user application is entered at the address specified by the start of the ROM (irrelevant here) plus the address within the ROM plus the execution offset. The mode is specified by the CPSR. At this point the RAM image will have been copied into the internal RAM from address &00000000 (the length is in bytes but will be rounded up to a whole number of words; a length of zero indicates no RAM image is required). If the FPGA pointers indicate an appropriate configuration block the FPGAs will have been initialised. If an LCD module is present it will have been reset, initialised to an eight-bit interface and cleared. If start-up string is required (see flags) and the LCD is present it will be printed to the LCD module. The initial state of the LCD backlight cab also be set here.

The memory has been remapped and all the areas are programmed to be useable. The PIO has been set into a 'safe' configuration. The serial ports are reset, enabled and initialised to 115.2 kbaud[1], 8-bits + 1 stop bit, no parity. The baud rate from serial port #0 is offered to the SCK0 output (port bit 13) but not set as an output by the PIO.
The external memory is cleared following a power-up reset but is unaffected by other resets (unless specified otherwise). The internal memory is not changed except that the RAM image is reloaded (as required) and a few (~8) words at the top of the memory will be corrupted by the start-up process.
The bits within the flag word (0 = don't, 1 = do) are:

Bit	Contents
31 - 20	Reserved
19	Enable data cache (rev. 3+)
18	Enable Instruction cache (rev. 3+)
17	Do not disable watchdog (rev. 3+)
16	Disable reset button (rev. 3+)
15 - 10	Reserved
9	Zero external RAM
8	Zero internal RAM
7 - 5	Reserved
4	Checksum ROM (Not yet implemented)
3	Start in RAM – absolute jump to "ROM execution offset" (rev. 2+)
2	LEDs enabled
1	LCD backlight enabled
0	Print LCD start-up message

Rev. 2 addition: Bit 3 allows an image to be copied into internal RAM (from address 00000000) and jumped to without entering the ROM image.

On entry to the user's application the following registers are meaningful:

Register	Contents
R0	Pointer to own boot table (true address)
R1	Size of internal RAM
R2	Base address of external RAM (if any)
R3	Length of external RAM
R4	Pointer to Spartan definition block (true address)
R5	Pointer to Virtex definition block (true address)
R6	Boot code version/date 8,8,5,4,7 bits (Maker, version, day, month, year)
R7	Flags: Bit 0 set if LCD module present Bit 8 set if power up reset Bit 9 set if watchdog reset
R8	Board serial number (Values 00000000 and FFFFFFFF are invalid codes)
R9	Clock speed in MHz: format <16bits>.<16bits> (Note: added in rev. 2)

1. Typically 32 MHz/(16*17) or 117.6 kbaud on ARM7 boards

In addition the various stack pointers are initialised to point within (or just beyond) the internal RAM space:. In the following table the internal RAM length is N (and its start address is always zero).

Register	Address
SP_svc	N
SP_FIQ	N - 200
SP_IRQ	N - 280
SP_abort	N - 300
SP_undef	N - 380
SP_user	N - 400

Additional setup on AT91SAM9261 boards

The default boot code starts both PLLs on the microcontroller. PLLA is used to clock the processor and is set to ~240MHz, the maximum speed of the part. The internal bus is set at half this speed.
PLLB is set to ~48MHz and is intended to be used for the USB interface. A ~24MHz clock, derived from PLLB, is exported to the FPGA via PC31.
[All SAM9 boards to date are fitted with an 18.432MHz crystal, which yields a processor speed of 239.616MHz and a USB clock of 48.055MHz.]

Both FPGA regions of the address map are initialised to timings identical to the Flash memory but with the bus width set to 8-bit; this maintains compatibility with earlier systems.

The peripheral clocks are enabled for all three PIOs. PIO C is set up as appropriate for on-board use; the other PIOs are not changed from their initial (input) state.

UARTs 0 & 1 have their clocks enabled and are initialised to 115k2 baud, 8 bits, no parity, one stop bit.

Memory Map

All the external devices reside at addresses programmed into the EBI. The memory maps set up by the boot programme for the ARM7 devices is given below.

Start Address	End Address	Chip select	Device	Actual Size	Width	Clocks/cycle
00000000	000FFFFF	—	On chip RAM	8KB/256KB	32	1
10000000	003FFFFF	—	Reserved	—	—	—
00400000	07FFFFFF	—	—	—	—	—
08000000	0BFFFFFF	CS0	Flash ROM	2MB	16	4
0C000000	0FFFFFFF	—	—	—	—	—
10000000	13FFFFFF	CS1	RAM	0-4MB	16	3
14000000	1FFFFFFF	—	—	—	—	—
20000000	23FFFFFF	CS2	Virtex	64	8/16	???
24000000	2FFFFFFF	—	—	—	—	—
30000000	300FFFFF	CS3	Ethernet	???	8	6
30100000	3FFFFFFF	—	—	—	—	—
40000000	400FFFFF	CS4	Spartan	32	8	3
40100000	FFBFFFFF	—	—	—	—	—
FFC00000	FFFFFFFF	—	On-chip peripherals	Assorted	32	1

The AT91SAM9261 does not have programmable chip selects and the MMU is not normally set up; the exception is if the data cache is enabled for the application[1], in which case the MMU is necessary for indicating cacheable areas. In this last case a translation table at offset 8000 in the ROM is used; this will typically be a flat mapping with (only) the ROM and RAM pages marked as cacheable. All the on-chip RAM is remapped to address 00000000 in both data and instruction spaces. The on-board Flash ROM is connected to NCS0 and both NCS1 and NCS2 are fed to the FPGA, allowing two different logical ‘devices’ to be integrated here. The figure below shows the memory map.

Start Address	End Address	Chip select	Device	Actual Size	Width	Clocks/cycle
00000000	000FFFFF	—	On chip RAM	160KB	32	1 ??
00100000	001FFFFF	—	ITCM	0	32	1
00200000	002FFFFF	—	DTCM	0	32	1
00300000	003FFFFF	—	On chip RAM	160KB	32	1??
00400000	004FFFFF	—	On chip ROM	32KB	32	1??
00500000	005FFFFF	—	UHP interface	0	32	1??
00600000	006FFFFF	—	LCD interface	0	32	1??
00700000	0FFFFFFF	—	—	0	32	1??
10000000	1FFFFFFF	NCS0	Flash ROM	8MB	16	22 (x 2)
20000000	2FFFFFFF	NCS1	Spartan-3	128bytes	8/16/32	???
30000000	3FFFFFFF	NCS2	Spartan-3	128bytes	8/16/32	???
40000000	EFFFFFFF	—	—	—	—	—
F0000000	FFFFFFFF	—	On-chip peripherals	Assorted	32	??

1. In which case all domains are set to ‘manager’ permissions.

Flash Programmer

A protocol is defined so that the on-board Flash ROM can be erased and reprogrammed via the serial interface. Two versions of the back-end are available.

On-board flash programming allows the board's own ROM to be reprogrammed. The lowest 64 Kbytes is not writeable to ensure the software cannot commit suicide.

Off-board flash programming uses a Spartan device and a cable to reprogramme another board via the second boards expansion bus connector. This allows any part of the Flash ROM to be written to.

Both versions of the code use the same protocols and can use the same front-end software.

Before allowing any changes the front- and back-end software establish bone fides by exchanging a simple message. The front end should send the bytes "FE", "A5", "1B", "1E" to which the back-end will respond "FE", "E1", "90", "0D".

The following commands are then available:

Command	Function	Parameter(s)	Returns
"C"	Check lock	W: Address	B: "N" if segment not locked B: "L" if segment locked
"E"	Erase	W: Address	B: "A" okay B: "N" if error occurred
"I"	ROM ID	None	B: Manufacturer code B: Part code
"L"	Lockout	W:Address (Not yet implemented)	B: "A"
"P"	Ping	None	B: "A"
"R"	Read	W: Start address W: Length (B: Acknowledgements)	B: "Length" bytes post-incrementing from "Start address"
"W"	Write	W: Start address W: Length B: "Length" bytes post-incrementing from "Start address"	(B: Acknowledgements)

Any other byte sent at 'command time' will simply be echoed back.

Both the read and write byte streams are punctuated every sixteen (or fewer if the remaining length is smaller) bytes by the receiver sending a single byte acknowledgement. The transmitter should wait for this before continuing to avoid overruns. An acknowledgement of "A" means "proceed"; anything else (typically "N") means terminate the command.

Note: a segment should normally be erased before it is written to.

Komodo/Monitor Communications Protocol

All communication is controlled by the front-end controller which sends commands to the back-end and expects the appropriate response. In the case of a response being detectably wrong or timing out it is the front-end's job to re-establish/resynchronise the protocol and recover.
Commands sent are single bytes with a variable number of parameters following.
In the following descriptions these definitions apply:
Byte: 8 bits, **H**alfword: 16 bits, **W**ord: 32 bits, **D**ouble word 64 bits, **P**ointer: 32 bits (ARM).
Commands are divided into sets on the most significant two bits as follows:

Command	Use
00xx xxxx	General operations
01xx xxxx	Memory transfers
10xx xxxx	Programme execution
11xx xxxx	Auxiliary functions

General operations are functions which establish and maintain communications with the target unit, setting breakpoints et al. They are also used for downloading configuration files to board "features" such as FPGAs.
Memory transfers are used for up- and down-loading the states in the address spaces of the target unit. All accessible state (memories, registers, etc.) is mapped into one of these addressable spaces.
Programme execution is used to start execution and set various options such as single stepping or breakpoints.
Auxiliary functions are reserved for future expansion.

General operations:

Command	Meaning
0000 xxxx	Board level communications
0001 xxxx	Board 'feature' communications
0010 xxxx	Programme execution control extras
0011 xxxx	Breakpoint/watchpoint manipulation

Board level communications:

Command	Function	Parameters	Return values
0000 0000	No operation	None	None
0000 0001	"Ping" – resynchronise	None	W: "OK??" ?? is the software version - initially "00"
0000 0010	Board definition	None	H: message length B: processor type H: processor subtype B: feature count {B: feature ID H: feature sub-ID} B: memory segment count {W: memory segment address W: memory segment length}
0000 0011	Reserved		
0000 0100	Reset	None	None
0000 0101+	Reserved		

Board feature/subfeature definitions

Feature type	Feature	Subfeature number	Subfeature
00	Terminal	bit 0	indicates activated by default
01 - 07	Not defined	—	—
08	Cycle counter/performance monitor	—	—
09 - 10	Not defined	—	—
11	Xilinx Spartan XL	05xx 0Axx 14xx 1Exx 28xx	XCS05— XCS10— XCS20— XCS30— XCS40—
12	Xilinx Virtex	05xx 0Axx 0Fxx 14xx 1Exx 28xx 3Cxx 50xx 64xx	XCV50— XCV100— XCV150— XCV200— XCV300— XCV400— XCV600— XCV800— XCV1000—
13	Xilinx Virtex-E	as Virtex (#12)	as Virtex (#12)
14	Xilinx Spartan-3	00xx 02xx 04xx 0Axx 0Fxx 14xx 28xx 32xx xx05	XC3S50— XC3S200— XC3S400— XC3S1000— XC3S1500— XC3S2000— XC3S4000— XC3S5000— —FG320 (???? FIX)
—	FPGA package options	xx00 xx01 xx02 xx03 xx04 xx05 xx06 xx07 xx08 xx09 xx0A xx0B xx0C xx0D xx0E xx0F xx10 xx11 xx12 xx13	—PC84 —VQ100 —CS144 —TQ144 —PQ208 —PQ240 —HQ240 —BG256 —FG256 —CS280 —BG352 —BG432 —FG456 —BG560 —FG676 —FG680 —CP132 —FT256 —FG900 —FG1156
15 - FF	Not defined	—	—

Board ‘feature’ communications

Command	Function	Parameters	Return values
0001 0000	Get Status	B: Feature number	W: Status Currently always 0
0001 0001	Set Status	B: Feature number W: Status Currently ignored	None
0001 0010	Send message	B: Feature number B: Length (0-255) Specified number of bytes.	B: Number of bytes accepted
0001 0011	Get message	B: Feature number B: Max. length (0-255)	B: Actual length (0-255) Specified number of bytes.
0001 0100	Download Header	B: Feature number W: Length of file	B: “A” if successful/“N” if feature unrecognised
0001 0101	Download Packet	B: Feature number B: Packet length (0⇒256) Specified number of bytes.	B: “A” if successful/“N” if unsuccessful
0001 0110+	Reserved	B: Feature number	

Device download is instigated by sending a header to the appropriate feature. In the case of an FPGA this initialises the device and readies it to receive a new programme. Subsequently a number of ‘packets’ should be sent with a total length equal to the specified file length. (Packets beyond this length will be ignored.)

Programme execution control extras

Command	Function	Parameters	Return values
0010 0000	What is executing?	None	B: Execution status W: Number of steps ‘remaining’ W: Number of steps since reset
0010 0001	Stop execution	None	None
0010 0010	Pause execution	None	None
0010 0011	Continue execution	None	None
0010 0100	Set runtime flags	B: xxxx xxIF	None
0010 0101	Get runtime flags	None	B: xxxx xxIF
0010 0110+	Reserved		

Execution status	Meaning
00	Reset
01	Busy - go away!
40	Stopped
41	Stopped due to breakpoint
42	Stopped due to watchpoint
43	Stopped due to memory fault
44	Stopped by programme request
80	Running normally
81	Servicing a SWI
8?	Servicing ????
C0-FF	Error codes

Trap manipulation

Command	Function	Parameters	Return values
0011 bb00	Define trap	B: trap number B: trap conditions B: trap sizes W: trap address A W: trap address B D: trap data A D: trap data B	None
0011 bb01	Read trap definition	B: trap number	B: trap conditions B: trap sizes W: trap address A W: trap address B D: trap data A D: trap data B
0011 bb10	Set trap status	W: bitmask1 W: bitmask0	None
0011 bb11	Read trap status	None	W: bitmask1 W: bitmask0

The actual class of trap is defined by the two bits “bb”:

bb	Meaning
00	Breakpoint (instruction fetch)
01	Watchpoint (data transfer)
10	Register value
11	Reserved

The trap conditions are set up as follows:

Trap condition	Meaning
Uxxx xxxx	0: do not trap if in user mode 1: may trap if in user mode
xPxx xxxx	0: do not trap if in privileged mode 1: may trap if in privileged mode
xxRW xxxx	00: do not trap 01: trap only on write accesses 10: trap only on read accesses 11: may trap on any transfer
xxxx AAxx	0x: reserved 10: may trap if addrA <= addr <= addrB 11: may trap if addr AND addrB = addrA
xxxx xxDD	0x: reserved 10: may trap if dataA <= data <= dataB 11: may trap if data AND dataB = dataA

These allow a flexible range of conditions to be set up. Note that not all conditions may be supported in all cases.

Transfer size mask	Meaning
xxxx Sxxx	Trap on 64-bit accesses
xxxx xSxx	Trap on 32-bit accesses
xxxx xxSx	Trap on 16-bit accesses
xxxx xxxS	Trap on 8-bit accesses

When activating of deactivating traps all the traps (of a given type) can be manipulated with a single command. The trap is defined by the two bits with the appropriate bit numbers.

bitmask1:bitmask0	Meaning when written	Meaning when read
0:0	No operation	Not implemented
0:1	Delete definition	Implemented but not defined
1:0	Inactivate	Inactive
1:1	Activate	Active

Thus the parameters 00000005:00000003 would inactivate trap #2, delete trap #1 and activate trap #0; an undefined trap will not be activated.

Memory transfers

Memory transfers are specified by the command word as follows:
01 mm d sss
mm indicates the address space used:

mm	Meaning
00	Memory address space
01	Registers
1x	Reserved

d indicates the direction:

d	Meaning
0	Write (0utput from user)
1	Read (1nput to user)

sss indicates the transfer element size:

sss	Meaning
000	8 bit
001	16 bit
010	32 bit
011	64 bit
1xx	Reserved

All memory transaction commands are followed by two parameters:
Address: word
Number of elements: halfword

These are then followed by the designated number of elements of the designated size in the appropriate direction. The address will be incremented to the next element remotely.
All transfers are little endian, so that a serial line will assume least significant byte first.

Programme execution

Run is a single byte command used to start execution on the remote processor. It is encoded as follows, where each bit 0/1 means disabled/enabled respectively:
10 w b m s p x
w enables halting when an active watchpoint is encountered
b enables halting when an active breakpoint is encountered
m enables halting when a memory abort occurs
s enables the treatment of a SWI as a single instruction (when single stepping)
p enables the treatment of a BL (procedure) as a single instruction (when single stepping)
x enables breakpoints on the first instruction executed (to allow ‘walk’ to break successfully)
Run commands are always followed by a single, word parameter which indicates the maximum number of steps to execute; 00000000 is a special case which indicates unlimited steps.

Auxiliary functions

None of these is yet defined.

ARM Software Emulator

The virtual machine is currently an ARM7 (no Thumb extension yet). All the external RAM (typically 512 Kbytes) is available to the user and begins at address 00000000.

The following I/O regions are available:

Start address	End address	Size (bytes)	Width	Use
10000000	1FFFFFFF	32	8	Internal I/O
20000000	2FFFFFFF	32	8	Spartan FPGA
?0000000	?0000000	64	16 (?)	Virtex FPGA

The internal I/O region map is:

Offset	Direction	Register	Remarks
00	R/W	PIO_A	Bidirectional data for LCD, LEDs etc.
04	R/W	PIO_B	Sundry bits plus PIO_A direction control
08	R/W	Timer	8-bit free running, incrementing at 1 kHz
0C	R/W	Timer compare	Interrupt asserted when timer equals this register
10	RO	Serial RxD	Each read removes one byte from the input stream
10	WO	Serial TxD	Each write puts one byte into the output stream
14	R/W	Serial status	Bit 1 = Tx ready; Bit 0 = Rx ready
18	R/W	Interrupt requests	Raw interrupt requests, active high
1C	R/W	Interrupt enables	Active high
20	WO	Stop execution	Writing any value causes execution to stop
20	RO	Serial Number	Byte #0 (LSB)
24	RO	Serial Number	Byte #1
28	RO	Serial Number	Byte #2
2C	RO	Serial Number	Byte #3 (MSB)
30-3C	–	Reserved	TBC

The bit maps for the I/O ports are as follows:

PIO_B:

Bit	Use		
7	Button (left)	1 = pressed	Read only
6	Button (right)	1 = pressed	Read only
5	LCD backlight	1 = on	
4	LED enable	1 = on	
3	Not used		
2	LCD R/$\overline{W}$	1 = read	also PIO_A direction
1	LCD RS	1 = data register	
0	LCD E	1 = active	

Interrupt Request/Enable:

Bit	Use
7	Button (left)
6	Button (right)
5	Serial transmitter ready
4	Serial receiver ready
3	Ethernet IRQ
2	Virtex IRQ
1	Spartan IRQ
0	Timer compare

The interrupt request bits are set or cleared by external events, but may also be written to by software.

Angel

Angel is the back-end software produced by ARM Ltd for their monitor and development environments. It should work with ARM tools; it is also supported by GDB. The reader is referred to ARM's documentation for the protocols, etc.

The relevant version here is 1.04, which was obtained from Atmel's WWW site and subsequently modified to make the code position independent. Note: the static data is still mapped at absolute addresses, beginning at 10000000 (i.e. the start of the RAM, as normally mapped).

The startup sequence copies the Angel code into RAM. If there is space (AT91R40008) the internal RAM is used, otherwise the code is moved to the top of the external RAM. The stacks are then initialised to the top of the (free) external RAM.