

## ARMermelator Quick Start Guide

Author: Pablo Bleyer

### Summary

The ARMermelator board has been conceived as a low cost, low power, general purpose embedded board targeted for OEM development and integration. It has been designed to take advantage of the ARM7TDMI architecture and reconfigurable logic using an Atmel AT91 processor and a Xilinx FPGA.

This guide will get you started programming your ARMermelator board using RedBoot and the eCos operating system.

### Knowing your ARMermelator

The block diagram of the ARMermelator board is shown in Figure 1.

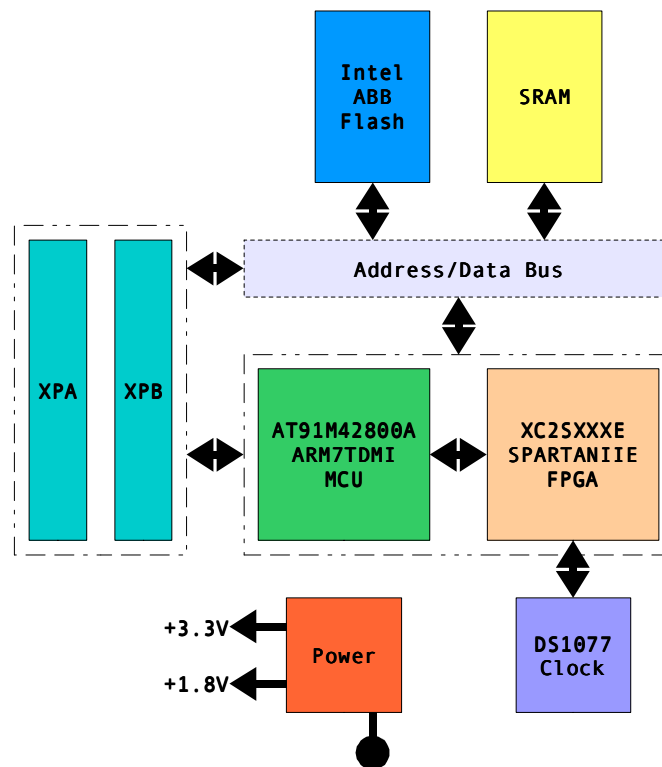


Figure 1: ARMermelator block diagram

Figure 2 shows the top view of the ARMermelator board.

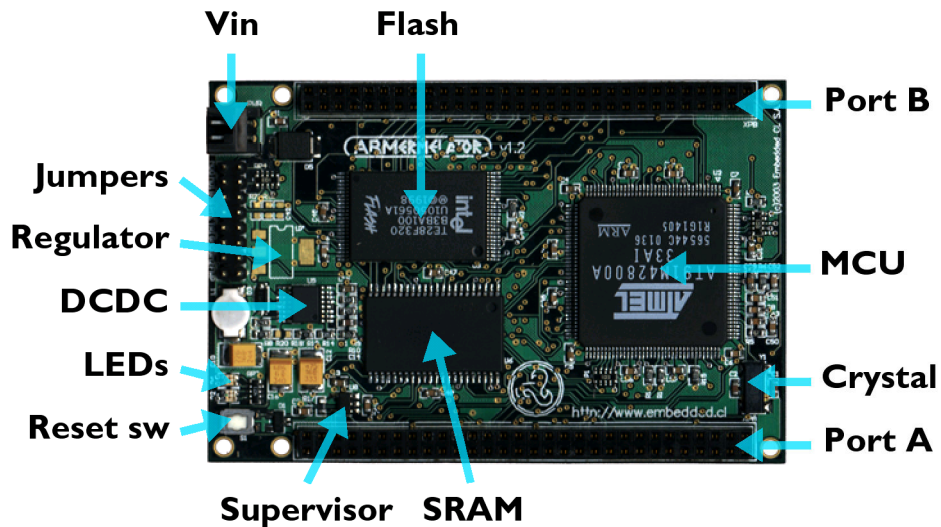


Figure 2: ARMermelator (top view)

On the top, the board contains the following main components:

1. Vin power connector
2. Configuration jumpers
3. Linear regulator (optional)
4. DC/DC switching regulator
5. LEDs
6. Manual reset switch
7. Processor voltage supervisor
8. Clock crystal
9. Atmel AT91M42800A ARM7TDMI microcontroller
10. Intel Advanced Boot Block flash memory
11. SRAM memory
12. Expansion port A (XPA)
13. Expansion port B (XPB)

Figure 3 shows the bottom view of the board.

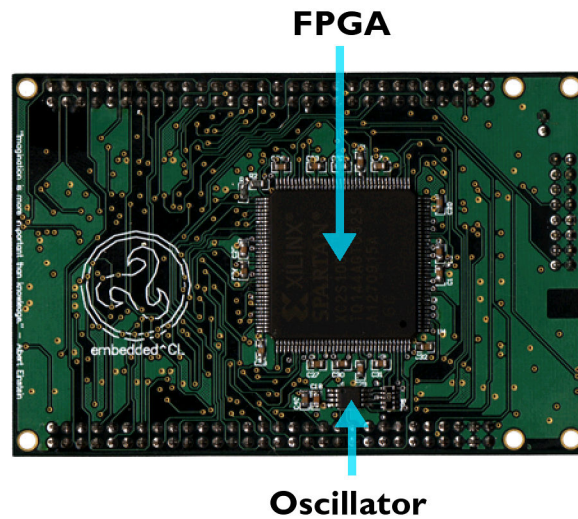


Figure 3: ARMermelator (bottom view)

Main components on the bottom of the board are:

1. Xilinx *SpartanII*E FPGA
2. Programmable oscillator

## AT91M42800A microcontroller

### Block diagram

Figure 4 shows a diagram of the Atmel AT91M42800A microcontroller. Refer to the AT91M42800A datasheet for more information about the processor and the ARM documentation for details about the ARM architecture.

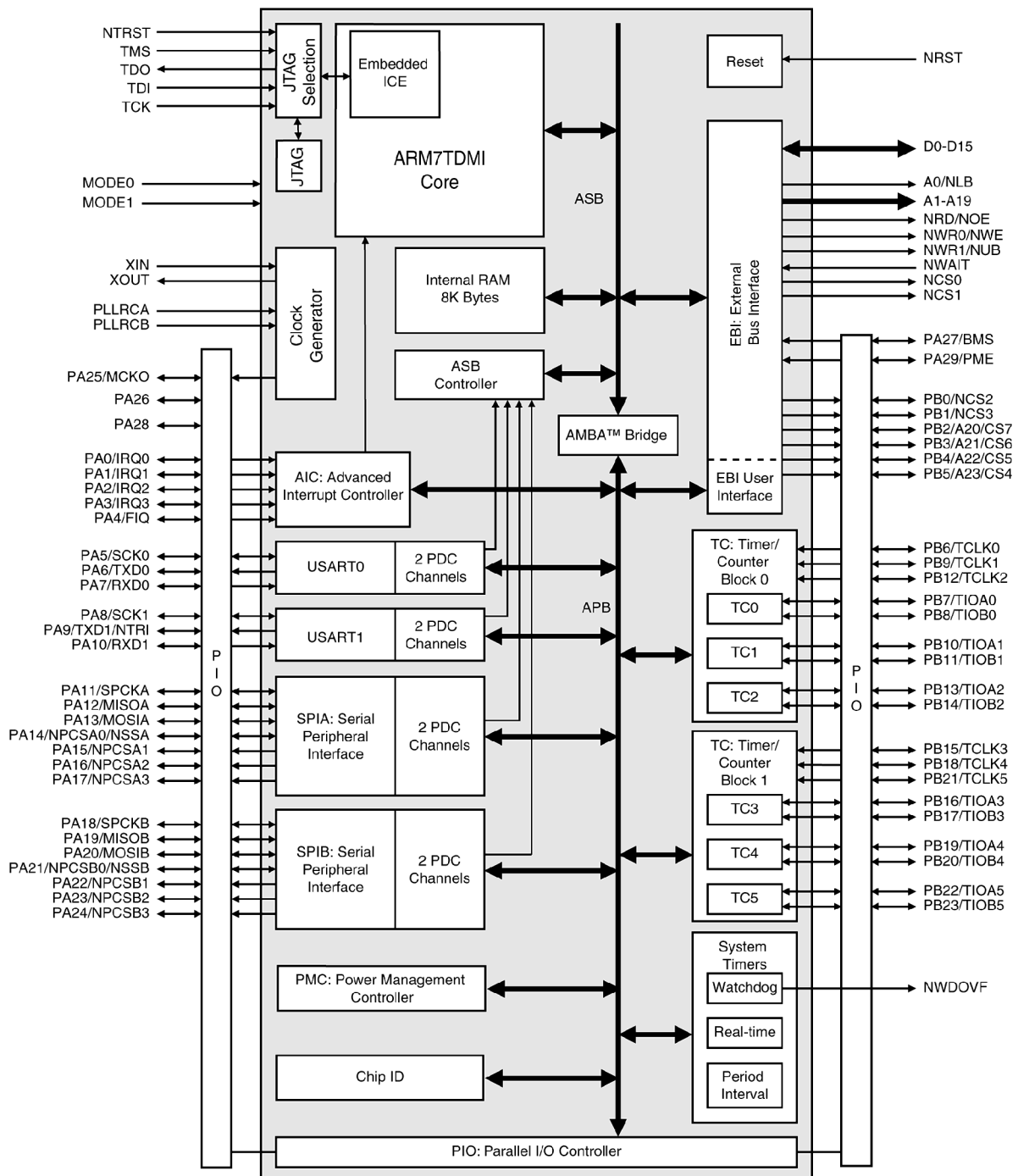


Figure 4: AT91M42800A block diagram

## Microcontroller pinout

Table 1 lists the AT91M42800A package pins and their associated signals.

P#	Signal	P#	Signal	P#	Signal
1	GND	32	D1	63	PB14/TIOB2
2	GND	33	D2	64	PB15/TCLK3
3	nLB/A0	34	D3	65	PB16/TIOA3
4	A1	35	VDDCORE	66	PB17/TIOB3
5	A2	36	VDDIO	67	PB18/TCLK4
6	A3	37	GND	68	PB19/TIOA4
7	A4	38	GND	69	PB20/TIOB4
8	A5	39	D4	70	PB21/TCLK5
9	A6	40	D5	71	VDDCORE
10	A7	41	D6	72	VDDIO
11	A8	42	D7	73	GND
12	VDDIO	43	D8	74	GND
13	GND	44	D9	75	PB22/TIOA5
14	A9	45	D10	76	PB23/TIOB5
15	A10	46	D11	77	PA0/IRQ0
16	A11	47	D12	78	PA1/IRQ1
17	A12	48	VDDIO	79	PA2/IRQ2
18	A13	49	GND	80	PA3/IRQ3
19	A14	50	D13	81	PA4/FIQ
20	A15	51	D14	82	PA5/SCK0
21	A16	52	D15	83	PA6/TXD0
22	A17	53	PB6/TCLK0	84	VDDIO
23	A18	54	PB7/TIOA0	85	GND
24	VDDIO	55	PB8/TIOB0	86	PA7/RXD0
25	GND	56	PB9/TCLK1	87	PA8/SCK1
26	A19	57	PB10/TIOA1	88	PA9/TXD1/nTRI
27	PB2/A20/CS7	58	PB11/TIOB1	89	PA10/RXD1
28	PB3/A21/CS6	59	PB12/TCLK2	90	PA11/SPCKA
29	PB4/A22/CS5	60	VDDIO	91	PA12/MISOA
30	PB5/A23/CS4	61	GND	92	PA13/MOSIA
31	D0	62	PB13/TIOA2	93	PA14/nPCSA0/nSSA

P#	Signal
94	PA15/nPCSA1
95	PA16/nPCSA2
96	VDDIO
97	GND
98	PA17/nPCSA3
99	PA18/SPCKB
100	PA19/MISOB
101	PA20/MOSIB
102	PA21/nPCSB0/nSSB
103	PA22/nPCSB1
104	PA23/nPCSB2
105	PA24/nPCSB3
106	PA25/MCKO
107	VDDCORE
108	VDDIO
109	GND
110	GND
111	PA26

P#	Signal
112	MODE0
113	XIN
114	XOUT
115	GND
116	PLLCA
117	VDDPLL
118	PLLRCB
119	VDDPLL
120	VDDIO
121	GND
122	nWDOVF
123	PA27/BMS
124	MODE1
125	TMS
126	TDI
127	TDO
128	TCK
129	nTRST

P#	Signal
130	nRST
131	PA28
132	VDDIO
133	GND
134	PA29/PME
135	nWAIT
136	nOE/nRD
137	nWE/nWR0
138	nUB/nWR1
139	nCS0
140	nCS1
141	PB0/nCS2
142	PB1/nCS3
143	VDDCORE
144	VDDIO

Table 1 : Microcontroller pinout

Some of these signals have a dedicated use in the ARMermelator board.

### Microcontroller signal description

Table 2 shows the AT91M42800A signal descriptions as stated in the processor datasheet.

Module	Name	Function	Type	Active	Comments
EBI	A[23:0]	Address Bus	Output		All valid after reset
	D[15:0]	Data Bus	I/O		
	CS[7:4]	Chip Select	Output	High	A23 - A20 after reset
	nCS[3:0]	Chip Select	Output	Low	
	nWR0	Lower Byte 0 Write Signal	Output	Low	Used in Byte Write option
	nWR1	Lower Byte 1 Write Signal	Output	Low	Used in Byte Write option
	nRD	Read Signal	Output	Low	Used in Byte Write option
	nWE	Write Enable	Output	Low	Used in Byte Select option
	nOE	Output Enable	Output	Low	Used in Byte Select option
	nUB	Upper Byte Select (16-bit SRAM)	Output	Low	Used in Byte Select option
	nLB	Lower Byte Select (16-bit SRAM)	Output	Low	Used in Byte Select option
	nWAIT	Wait Input	Input	Low	
	BMS	Boot Mode Select	Input		Sampled during reset
	PME	Protect Mode Enable	Input	High	PIO-controlled after reset
AIC	IRQ[3:0]	External Interrupt Request	Input		PIO-controlled after reset
	FIQ	Fast External Interrupt Request	Input		PIO-controlled after reset
TC	TCLK[5:0]	Timer External Clock	Input		PIO-controlled after reset
	TIOA[5:0]	Multi-purpose Timer I/O Pin A	I/O		PIO-controlled after reset
	TIOB[5:0]	Multi-purpose Timer I/O Pin B	I/O		PIO-controlled after reset
USART	SCK[1:0]	External Serial Clock	I/O		PIO-controlled after reset
	TXD[1:0]	Transmit Data Output	Output		PIO-controlled after reset
	RXD[1:0]	Receive Data Input	Input		PIO-controlled after reset
SPI	SPCKA/SPCKB	Clock	I/O		PIO-controlled after reset
	MISOA/MISOB	Master In Slave Out	I/O		PIO-controlled after reset
	MOSIA/MOSIB	Master Out Slave In	I/O		PIO-controlled after reset
	nSSA/nSSB	Slave Select	Input	Low	PIO-controlled after reset
	nPCSA[3:0]/nPCSB[3:0]	Peripheral Chip Selects	Output	Low	PIO-controlled after reset
PIO	PA[29:0]	Programmable I/O Port A	I/O		Input after reset
	PB[23:0]	Programmable I/O Port B	I/O		Input after reset
ST	nWDOVF	Watchdog Timer Overflow	Output	Low	Open drain
CLOCK	XIN	Oscillator Input or External Clock	Input		
	XOUT	Oscillator Output	Output		
	PLLRC A	RC Filter for PLL A	Input		
	PLLRC B	RC Filter for PLL B	Input		
	MCKO	Clock Output	Output		

Module	Name	Function	Type	Active	Comments
Test & reset	nRST	Hardware Reset Input	Input	Low	Schmitt trigger
	MODE[1:0]	Mode Selection	Input		
JTAG/ICE	TMS	Test Mode Select	Input		Schmitt trigger, internal pull-up
	TDI	Test Data In	Input		Schmitt trigger, internal pull-up
	TDO	Test Data Out	Output		
	TCK	Test Clock	Input		Schmitt trigger, internal pull-up
	nTRST	Test Reset Input	Input	Low	Schmitt trigger, internal pull-up
Emulation	nTRI	Tri-state Mode Enable	Input	Low	Sampled during reset
Power	VDDIO	I/O Power	Power		
	VDDCORE	Core Power	Power		
	VDDPLL	PLL Power	Power		
	GND	Ground	Ground		

Table 2: Microcontroller signal description

## Memory map

The following table displays the memory map of the AT91M42800A microcontroller.

Address	Function [BR]	Function [AR]	Size	Special
0xffff_ffff 0xffc0_0000	On-chip peripherals	On-chip peripherals	4MB	Privileged, abort control
0xffbf_ffff 0x0040_0000	Reserved	External device(s)	4MB [BR] 1/4/16/64MB [AR]	Abort control [AR]
0x003f_ffff 0x0030_0000	On-chip SRAM (8kB)	Reserved	1MB	
0x002f_ffff 0x0020_0000	Reserved on-chip device	Reserved on-chip device	1MB	
0x001f_ffff 0x0010_0000	Reserved on-chip device	Reserved on-chip device	1MB	
0x000f_ffff 0x0000_0000	External device(s) connected to nCS0	On-chip SRAM (8kB)	1MB	

Table 3: AT91M42800A memory map

Note that there are two memory spaces, one before the *cancel remap command* (BR) and one after it (AR). The AT91M42800A comes out of reset with memory spaces remapped to allow proper device initialization using a non-volatile memory at chip select 0 (connected to nCS0 signal). The ARM architecture has a hard-coded exception vector table at address 0x0000\_0000, so in order to permit dynamic exception vectors, stacks and code execution from the fast internal memory, the remap can be canceled to place the on-chip SRAM beginning at address 0x0000\_0000. The cancel remap command also enables the other chip select memory spaces if they are enabled in the external bus interface (EBI)



configuration registers.

### **Peripheral memory map**

Table 4 lists the memory map of the internal peripherals of the AT91M42800A microcontroller. As with most controller units, peripherals are configured and managed using registers at specific addresses. The ARMermelator code distribution contains C macros and definitions for register locations and values that follow the name convention used in the AT91M42800A datasheet.

Address	Peripheral	Name	Size
0xffff_ffff 0xffff_f000	AIC	Advanced Interrupt Controller	4kB
0xffff_ffff 0xffff_c000		Reserved	16kB
0xffff_bfff 0xffff_8000	ST	System Timer	16kB
0xffff_7fff 0xffff_4000	PMC	Power Management Controller	16kB
0xffff_3fff 0xffff_0000	PIOB	Parallel IO Controller B	16kB
0xfffe_ffff 0xfffe_c000	PIOA	Parallel IO Controller A	16kB
0xfffe_bfff 0xfffd_8000		Reserved	16kB
0xfffd_7fff 0xfffd_4000	TC1	Timer Counter 1	16kB
0xfffd_3fff 0xfffd_0000	TC0	Timer Counter 0	16kB
0xfffc_ffff 0xfffc_c000	SPIB	SPI B	16kB
0xfffc_bfff 0xfffc_8000	SPIA	SPI A	16kB
0xfffc_7fff 0xfffc_4000	USART1	USART 1	16kB
0xfffc_3fff 0xfffc_0000	USART0	USART 0	16kB
0xfffb_ffff 0xfff0_4000		Reserved	16kB
0xfff0_3fff 0xfff0_0000	SF	Special Function	16kB
0xffef_ffff 0xffe0_4000		Reserved	16kB
0xffe0_3fff 0xffe0_0000	EBI	External Bus Interface	16kB
0xffdf_ffff 0xffd0_0000		Reserved	16kB

Table 4: Peripheral memory map

## Jumpers

Figure 5 shows the board configuration jumpers.

<b>MD0</b>	<b>8</b>
<b>MD1</b>	<b>7</b>
<b>BMS</b>	<b>6</b>
<b>PME</b>	<b>5</b>
<b>TRI</b>	<b>4</b>
<b>GM0</b>	<b>3</b>
<b>GM2</b>	<b>2</b>
<b>FWP</b>	<b>1</b>

Figure 5: Jumpers

### 1. MD[1:0] Mode pins (NPD)

These normally pulled-down pins control the operating mode of the microcontroller. Refer to the AT91M42800A datasheet for more information.

<b>MD[1:0]</b>	<b>Operating mode</b>
00	Normal operating mode (internal crystal oscillator)
01	Normal operating mode (external XIN clock)
10	Boundary scan mode
11	Reserved

### 2. BMS – Boot Mode Select (NPD)

BMS is normally pulled-down. It selects the data width of the boot memory, connected to chip-select nCS0.

The ARMermelator board has been designed to use 16-bit wide Intel Advanced Boot Block Flash memory. This jumper could be used to boot from an external device with a different data bus width when the main Flash memory is not populated.

<b>BMS</b>	<b>Boot memory</b>
0	16-bit memory
1	8-bit memory

### 3. PME – Protected Memory Enable (NPD)

The normally pulled down PME signal controls the peripheral protect mode. Any attempt to access the peripheral space when protection is enabled will result in the ARM core aborting the bus transfer. Refer to the AT91M42800A datasheet for more information.

PME	Boot memory
0	PME disabled
1	PME enabled

#### 4. TRI – Tristate (NPU)

The TRI jumper may be used to control the nTRI signal of the microcontroller. When pulled down, the microcontroller outputs are tri-stated at reset. This operating mode is commonly used to connect an in-circuit emulator (ICE) to the board. In the ARMermelator, this signal could also be used by the FPGA to take control of the board.

This signal is shared with TXD1 and is sampled at reset.

#### 5. GM[2,0] – FPGA Configuration Mode (NPU)

These jumpers are used to select the configuration mode of the FPGA. Without installed jumpers, the default is slave serial mode without pre-configuration pull-ups.

The FPGA configuration is done by the microcontroller or an external controller connected to the shared programmable IO pins (IOB set). JTAG configuration is always possible through the FPGA JTAG pins.

GM0	Configuration mode
0	Slave parallel (SelectMAP)
1	Slave serial
GM2	Pre-configuration pull-ups
0	Enabled
1	Disabled

#### 6. FWP – Flash Write Protect (NPU)

This jumper controls the Flash write protect pin of the Flash memory. When FWP is down the Flash memory contents cannot be overwritten. FWP is normally pulled-up.

#### LEDs

The ARMermelator board has 4 LEDs (2 independent bi-color LEDs). They are used to indicate board power, board reset, and user programmable functions if SCK signals are not used.

LEDs are arranged in the following order:

LED name	Color	Details
Green LED	Green	MCU PA5, shared with SCK0
Red LED	Red	MCU PA8, shared with SCK1
Reset LED	Green	Connected to nRST line
Power-on LED	Yellow	Connected to +3.3V power

## External memories

### ROM

Intel Advanced Boot Block in TSOP48(I) package is used to store non-volatile program memory and data. This memory provides lockable 64kB blocks, finer grained 8kB boot blocks, a minimum of 100k erase cycles and device identifiers. The ARMermelator board can support up to 16MB of Flash memory at chip-select nCS0.

### RAM

The ARMermelator board has been designed to allow up to 1MB of standard asynchronous static RAM in the TSOP44(II) package. External SRAM is connected to chip-select nCS1.

### FPGA

One of the main characteristics of the ARMermelator board is that it has been conceived to take advantage of the benefits of reconfigurable logic. This provides endless possibilities for device integration, hardware acceleration and special function implementation.

Both Xilinx XC2S50E and XC2S100E SpartanII E devices in the TQFP144 package are usable. The board has been carefully designed to provide optimum architecture flexibility regarding microcontroller and FPGA usage.

### FPGA pinout

Table 5 shows how the board signals are mapped to the FPGA pins.

P#	Signal	Alternate	Location
<b>3</b>	SCK0		XPA
<b>4</b>	TXD0		XPA
<b>5</b>	RXD0		XPA
<b>6</b>	SCK1		XPA
<b>7</b>	TXD1		XPA
<b>8</b>	RXD1		XPA
<b>10</b>	D15		XPB
<b>11</b>	D14		XPB
<b>12</b>	D13		XPB
<b>13</b>	D12		XPB
<b>14</b>	D11		XPB
<b>15</b>	D10		XPB
<b>18</b>	D9		XPB
<b>20</b>	D8		XPB

P#	Signal	Alternate	Location
<b>21</b>	D7		XPB
<b>22</b>	D6		XPB
<b>23</b>	D5		XPB
<b>24</b>	D4		XPB
<b>26</b>	D3		XPB
<b>27</b>	D2		XPB
<b>28</b>	D1		XPB
<b>29</b>	D0		XPB
<b>30</b>	A23		XPB
<b>31</b>	A22		XPB
<b>32</b>	A21		XPB
<b>38</b>	A20		XPB
<b>39</b>	A19		XPB
<b>40</b>	A18		XPB

P#	Signal	Alternate	Location
41	A17		XPB
42	A16		XPB
43	A15		XPB
44	A14		XPB
47	A13		XPB
48	A12		XPB
49	A11		XPB
50	IOB2	DLL1	XPA
52	CLKX	GCK1	XPA
55	MCKO	GCK0	XPB
56	IOB3	DLL0	XPA
57	A10		XPB
58	A9		XPB
59	A8		XPB
60	A7		XPB
63	A6		XPB
64	A5		XPB
65	A4		XPB
66	A3		XPB
67	A2		XPB
68	A1		XPB
69	A0		XPB
74	IOB16	nINIT	XPA
75	IOB8	D7	XPA
76	nCS3		XPB
77	nCS2		XPB
78	nCS1		XPB
79	nCS0		XPB
80	IOB9	D6	XPA
82	IOB10	D5	XPA
83	nUB		XPB
84	nWE		XPB
85	nOE		XPB

P#	Signal	Alternate	Location
86	IOB11	D4	XPA
87	nWAIT		XPB
89	PA29/PME		Jumper
92	nRST		XPA, XPB
93	nWDOVF		XPB
94	IOB12	D3	XPA
95	IOB13		XPA
96	IOA12		XPA
97	IOA11		XPA
98	IOB13	D2	XPA
100	IOB14	D1	XPA
101	IOA10		XPA
102	IOA9		XPA
103	IOA8		XPA
104	IOB17		XPA
105	IOB15	DIN/D0	XPA
106	IOB5	DOUT/BUSY	XPA
112	IOB6	nCS	XPA
113	IOB7	nWRITE	XPA
114	IOA7		XPA
115	IOA6		XPA
116	IOA5		XPA
117	IOA4		XPA
118	IOA3		XPA
121	IOA2		XPA
122	IOA1		XPA
123	IOA0		XPA
124	IOB4		XPA
125	IOB0	DLL2	XPA
126	CLK1	GCK2	XPA
129	CLK0	GCK3	XPA
131	IOB1	DLL3	XPA
132	FIQ		XPB

P#	Signal	Alternate	Location
133	IRQ3		XPB
134	IRQ2		XPB
137	IRQ1		XPB
138	IRQ0		XPB
139	EK_SCL		GA-EK
140	EK_SDA		GA-EK
141	EK_CTRL1		GA-EK
142	EK_CTRL0		GA-EK
71	GA_DONE		GA-MC
73	GA_nPROG		GA-MC
35	GM0	M0	Jumper
33		M1	N/A

P#	Signal	Alternate	Location
37	GM2	M2	Jumper
107	(IOB17)	CCLK	XPA
111	GA_TDI		XPA
109	GA_TDO		XPA
2	GA_TMS		XPA
143	GA_TCK		XPA

Table 5: FPGA pinout

Name convention:

- GA: gate array (FPGA)
- MC: microcontroller
- XPA, XPB: expansion ports
- EK: programmable clock oscillator

Note that some signals have alternate functions. Refer to the SpartanIIIE datasheet for FPGA design, configuration and usage information.

Some board signals shared between the microcontroller, FPGA and programmable clock oscillator are not exported through the expansion ports. These are:

- DONE, nPROGRAM: control the FPGA configuration by the microcontroller (slave serial or slave parallel modes).
- SCL, SDA, CTRL[1:0]: programmable clock oscillator configuration and control signals.

Since FPGA master configuration is not possible, M1 pin has been left unconnected.

## Expansion ports

The ARMermelator contains two expansion ports A and B, labeled XPA and XPB respectively. These expansion ports use 0.079" (2mm) pitch headers separated a distance of 1.78" (45.21mm).

Expansion port A provides peripheral IO, board clocks, microcontroller JTAG/ICE signals and FPGA configuration pins. Expansion port B provides the address and data bus, bus control signals and interrupts.

### Expansion port A (XPA)

Figure 6 illustrates the pinout of the XPA connector.

	VEXT	1	2	+3.3	
	GND	3	4	GND	
	MC_nTRST	5	6	nRST	
	MC_TCK	7	8	GA_TCK	
	MC_TMS	9	10	GA_TMS	
	MC_TDO	11	12	GA_TDI	
	MC_TDI	13	14	GA_TDO	
	XB_TDO	15	16	XB_TDI	
	MODE1	17	18	MODE0	
	XIN	19	20	CLK0	
	CLKX	21	22	CLK1	
LEDG	SCK0	23	24	SCK1	LEDR
	TXD0	25	26	TXD1	
	RXD0	27	28	RXD1	
GA_DLX2	IOB0	29	30	IOB1	GA_DLX3
GA_DLX1	IOB2	31	32	IOB3	GA_DLX0
	IOB4	33	34	IOB5	GA_DOUT
GA_nCS	IOB6	35	36	IOB7	GA_nWR
GA_D7	IOB8	37	38	IOB9	GA_D6
GA_D5	IOB10	39	40	IOB11	GA_D4
GA_D3	IOB12	41	42	IOB13	GA_D2
GA_D1	IOB14	43	44	IOB15	GA_D0
GA_nINIT	IOB16	45	46	IOB17	GA_CCLK
	IOA0	47	48	IOA1	
	IOA2	49	50	IOA3	
	IOA4	51	52	IOA5	
	IOA6	53	54	IOA7	
	IOA8	55	56	IOA9	
	IOA10	57	58	IOA11	
	IOA12	59	60	IOA13	

Figure 6: Expansion port A (XPA)

The following group of signals are available on XPA:

- Power signals: VEXT (external voltage input), +3.3V (regulated +3.3V), GND (ground)
- Microcontroller JTAG/ICE: nTRST, TCK, TMS, TDO, TDI
- FPGA JTAG: TCK, TMS, TDI, TDO
- External board JTAG: TDO, TDI
- Microcontroller mode pins: MODE0, MODE1
- Board clocks: XIN (I), CLKX (I), CLK0 (O), CLK1 (O)
- USARTs: SCK0, TXD0, RXD0, SCK1, TXD1, RXD1
- PIOs: IOA[13:0], IOB[17:0] (MCU PIO A and PIO B)



- Reset: nRST

SCK0 and SCK1 are shared with the board LEDs.

The layout of microcontroller JTAG/ICE, FPGA JTAG and external board JTAG signals was designed to allow daisy chaining of devices in a JTAG boundary-scan mode fashion. Also, nTRST has been placed contiguous to nRST to ease shunting if both signals need to be asserted simultaneously.

In their alternate function, IOB signals are used to configure the FPGA in slave serial or slave parallel (SelectMAP) mode. After FPGA configuration the shared signals may be used for other purpose. FPGA clock DLL deskew signals are shared with the IOB[3:0] pins.

If the programmable oscillator is present, its clock outputs CLK0 and CLK1 are available on XPA. CLX is an external clock input connected to FPGA global clock 1 net. The XIN input has been exported to admit an external clock input to the microcontroller, in which case the internal crystal oscillator must be bypassed (using the MODE pins).

**Caution**

Care must be taken with the XIN pin. This pin is connected directly to the microcontroller input of the crystal oscillator optimized for a 32.768kHz crystal. Avoid any noise or stray capacitance on XIN when using the 32.768kHz crystal because it will affect correct oscillator operation.

---

**Expansion port B (XPB)**

Expansion port B layout is shown on Figure 7.

VEXT	1	2	+3.3
GND	3	4	GND
D15	5	6	D14
D13	7	8	D12
D11	9	10	D10
D9	11	12	D8
D7	13	14	D6
D5	15	16	D4
D3	17	18	D2
D1	19	20	D0
A23	21	22	A22
A21	23	24	A20
A19	25	26	A18
A17	27	28	A16
A15	29	30	A14
A13	31	32	A12
A11	33	34	A10
A9	35	36	A8
A7	37	38	A6
A5	39	40	A4
A3	41	42	A2
A1	43	44	A0
IRQ1	45	46	IRQ0
IRQ3	47	48	IRQ2
FIQ	49	50	nWAIT
nCS0	51	52	nOE
nCS1	53	54	nWE
nCS2	55	56	nUB
nCS3	57	58	nWDOVF
MCKO	59	60	nRST

Figure 7: Expansion port B (XPB)

The following set of signals are available on XPB:

- Power signals: VEXT (external voltage input), +3.3V (regulated +3.3V), GND (ground)
- Data bus: D[15:0]
- Address bus: A[23:0]
- Interrupt requests: IRQ[3:0] and FIQ (ARM fast interrupt request)
- Bus control: chip-selects nCS[3:0], nWAIT, nOE, nWE, nUB
- Reset: nRST and nWDOVF
- Master clock output: MCKO

The board Flash memory is connected to signal nCS0 and the external SRAM is tied to nCS1. The external

bus must be configured appropriately through the EBI module registers in the microcontroller to allow external memories and memory-mapped peripherals using chip-selects and/or address decoding. Refer to the AT91M42800A datasheet for details.

nWDOVF is the watchdog overflow output. It can be used to alert the system that the watchdog has expired and to perform a global reset.

If enabled, the master clock output signal will output the internal clock from the microcontroller.

## Board power

For integration flexibility, the ARMermelator board was designed to allow different power supply configurations. These configurations are selected during factory assembly depending on how the device will be used upon user requirement.

The power supply system consists of a linear regulator (*LM1117* or similar) and an efficient switching regulator (*MAX1765*). The latter has an integrated LDO regulator for FPGA core voltage generation.



---

### Warning

Depending on the power configuration of your board, it accepts different voltage input ranges. Always verify the DC input voltage before powering the device to avoid the risk of damaging it.

---

Possible power configurations are:

#### 1. Step-up DC/DC, +3.3V and +1.8V generation

In this configuration the system accepts a +1V to +3.5V input and generates the main +3.3V supply and FPGA core +1.8V using the switching regulator. The linear regulator is not populated.

Total power consumption is 800mA @ +3.3V. The FPGA core voltage is generated with the MAX1765 LDO regulator and can supply a maximum of 500mA in steady state. This supply is derived from the +3.3V supply.

#### 2. Step-up DC/DC +1.8V, regulated +3.3V (independent supplies)

The linear regulator is used to generate the +3.3V supply, while the switching regulator generates +1.8V efficiently. This configuration allows as much current the linear regulator can source for the +3.3V supply.

DC input voltage range for this mode is +3.5V to +5.5V.

#### 3. Regulators for +1.8V and +3.3V (chained supplies)

This mode derives the +1.8V supply from the linear regulator's +3.3V supply. The purpose of this configuration is to allow extended DC voltage input.

**Caution**

As with any power supply, care must be taken to satisfy the linear regulator's electrical and thermal requirements.

**4. Stand-alone +3.3V**

When the FPGA is not present in the board, the +1.8V supply is not needed. This low cost configuration uses the linear regulator for dropout +3.3V generation and the switching regulator is not populated.

**Connector polarity**

Figure 8 shows the voltage polarity of the power connector in the ARMermelator board.

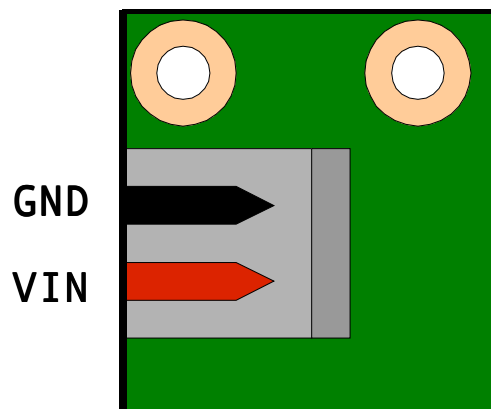


Figure 8: Connector polarity

**Reset considerations**

The processor supervisor takes care of proper board initialization holding the reset line (nRST signal) down for approximately 800ms after power-up.

The manual reset switch allows to reset the board bringing the reset line directly and momentarily to ground. The open-drain output of the voltage supervisor makes this possible. The reset switch is debounced with an RC filter.

**JTAG/ICE**

The ARM7TDMI architecture provides an embedded ICE debug interface that uses the JTAG port of the processor (EmbeddedICE/RT interface). Through this interface it is possible to halt the processor, examine its registers, step through program code, etc.

ARM suggests a standard 20-pin debug pod connection between the host system and the target system being tested, as shown in Figure 9. Since the ARMermelator was designed for a minimum integration footprint, it needs an adapter to connect it to a debug pod.

+3.3	1	2	+3.3
MC_nTRST	3	4	GND
MC_TDI	5	6	GND
MC_TMS	7	8	GND
MC_TCK	9	10	GND
	11	12	GND
MC_TDO	13	14	GND
nRST	15	16	GND
	17	18	GND
	19	20	GND

Figure 9: ARM JTAG/ICE connector

The AMLprogrammer (Figure 10) board aids in the ARMermelator debugging process adapting its expansion ports to the standard ARM debug interface.

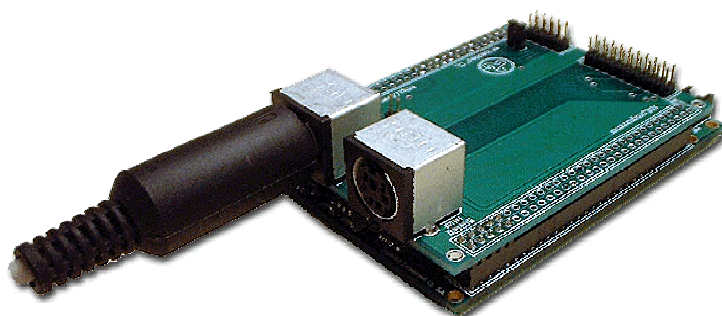


Figure 10: AMLprogrammer

It also provides two RS232 transceivers connected to the microcontroller UART signals and 6-pin MiniDIN connectors. The pinout of each connector is shown in Figure 11.

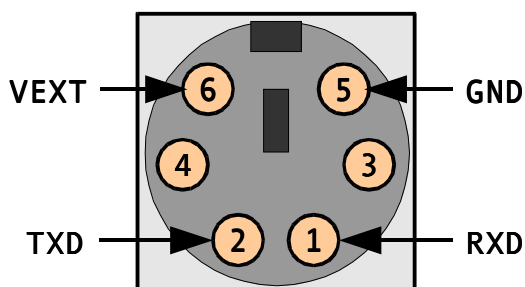


Figure 11: AMLprogrammer RS232 pinout

The AMLprogrammer features a second 10-pin header that exports the FPGA JTAG signals as illustrated in Figure 12.

+3.3	1	2	+3.3
GA_TCK	3	4	GA_TDO
GA_TDI	5	6	GA_TMS
nRST	7	8	
GND	9	10	GND

Figure 12: JTAG connector

Figure 13 illustrates the correct way to insert an ARM JTAG/ICE pod in the AMLprogrammer board. Note that the #1 pins of both JTAG and JTAG/ICE headers face outwards. The AMLprogrammer must be inserted such that the RS232 MiniDIN connectors are placed above the ARMermelator board microcontroller and the JTAG and JTAG/ICE pins are contiguous to the ARMermelator board jumpers.

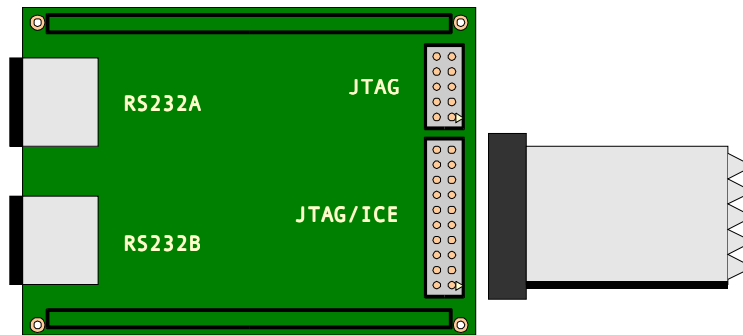


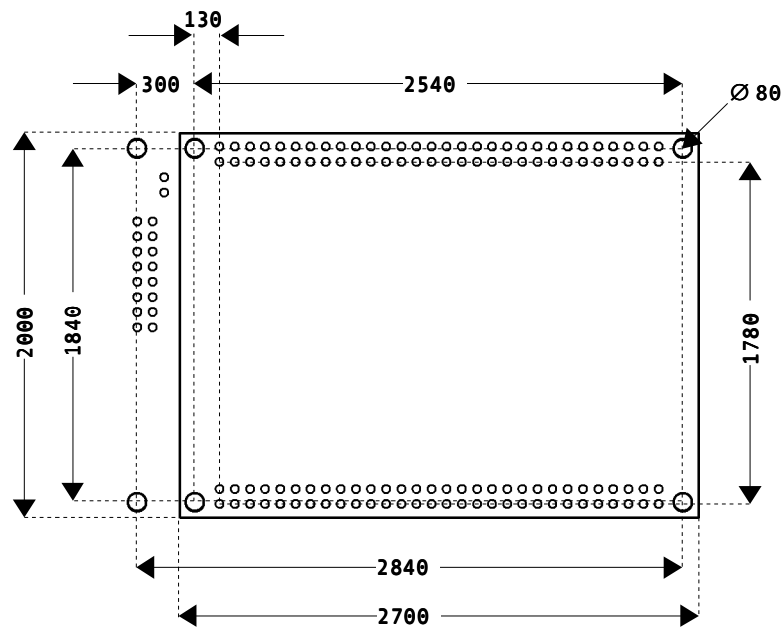
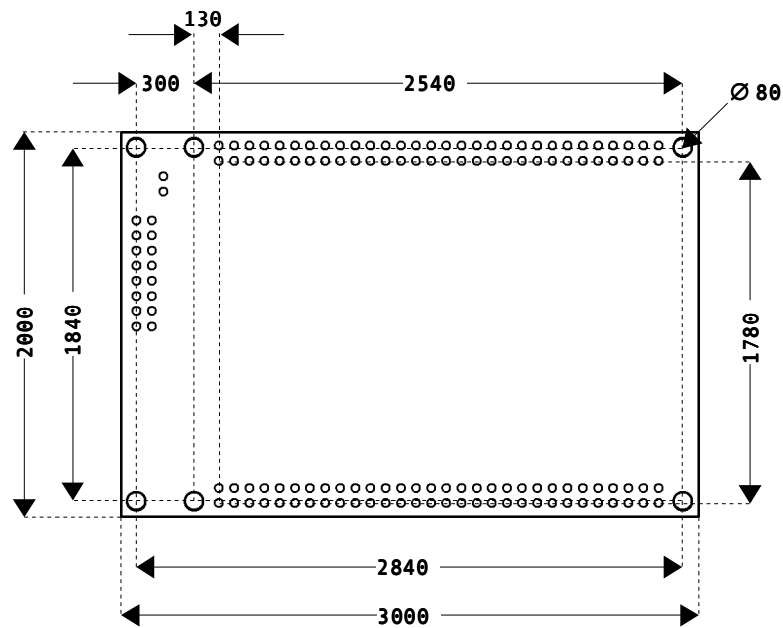
Figure 13: JTAG/ICE pod insertion

The AMLprogrammer also features two extra jumpers:

- The RSTC jumper chains the board reset signal with the nTRST JTAG/ICE reset signal. This allows a global reset to initialize and synchronize both the in-circuit emulator and the microcontroller.
- The SCEN jumper disables the RS232 transceivers tri-stating the driver inputs connected to the microcontroller RXD pins. This is useful if the AMLprogrammer is going to be used only as a JTAG/ICE adapter and other peripheral will control the RXD pins, or they will be configured as outputs by the microcontroller.

## Board dimensions

For reference, Figures 14 and 15 illustrate the ARMermelator board and module mechanical dimensions, expressed in mils (1/1000").



## Developing with the ARMermelator

### Preliminaries

The distribution CDROM contains all the necessary software tools and applications to develop with the ARMermelator. The CDROM contains:

- GNUARM toolchain (includes GNU compilers and tools like Insight/GDB debugger)
- eCos distribution + ARMermelator patches (*aml*)
- Eclipse IDE
- Macraigor's OCDLibRemote proxy
- Xilinx WebPACK ISE
- Atmel and Xilinx documentation

This guide will show you step-by-step ARMermelator development examples with RedBoot and eCos. The examples have been compiled in MS Windows under the Cygwin environment. It is assumed that the GNUARM toolchain and ARMermelator eCos distribution have been already installed in your system.

### RedBoot

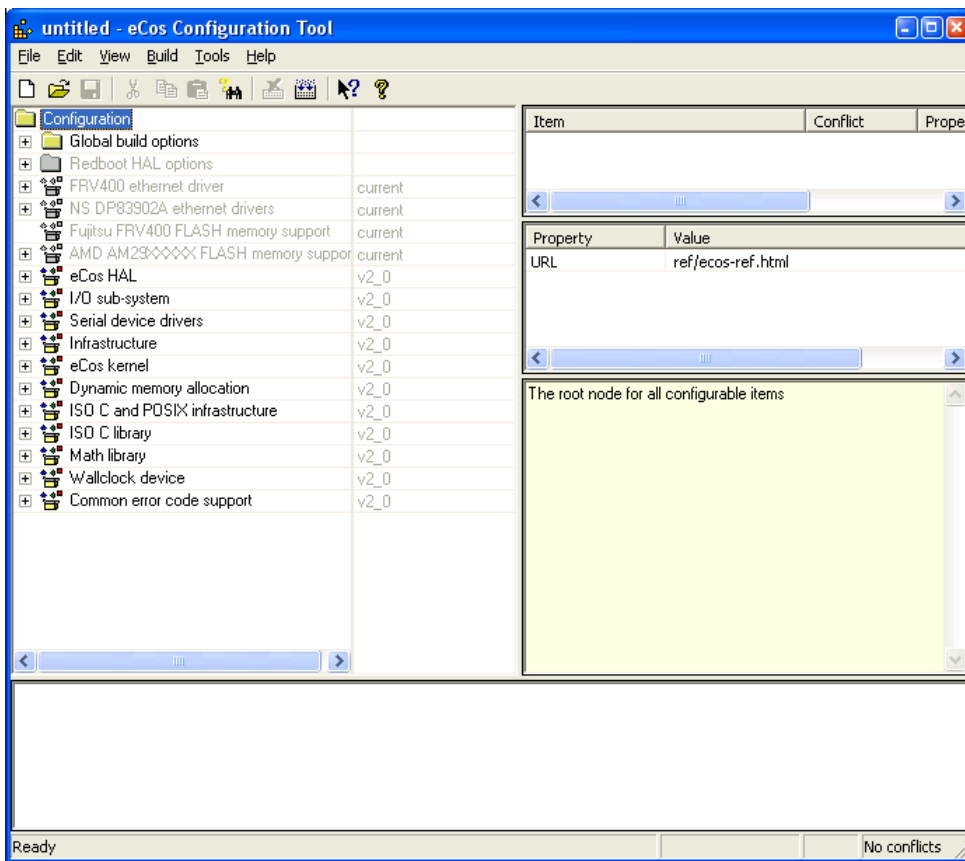
RedBoot is the RedHat bootstrap environment. RedBoot uses a minimal eCos footprint and is integrated in the eCos distribution as a standard template.

### RedBoot configuration

This guide will use the graphical *eCos Configuration Tool* for RedBoot and eCos configuration. This tool usually resides in the `tools/bin` subdirectory of the eCos distribution (the *repository*). The tool allows the user to specify eCos parameters and options in an intuitive way.

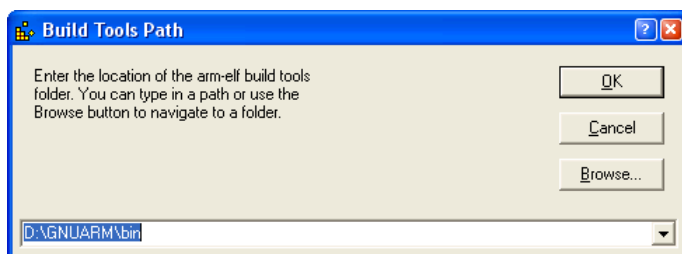
On startup, the tool window will look like in the following figure:



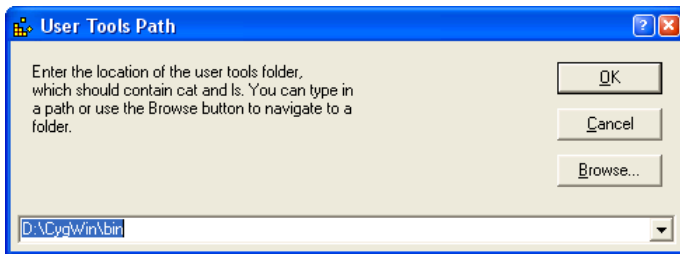


If this is your first time you work with the configuration tool, you must configure the tools path:

Select the menu Tools->Paths->Build Tools..., enter the path of your GNUARM toolchain binaries and press OK:

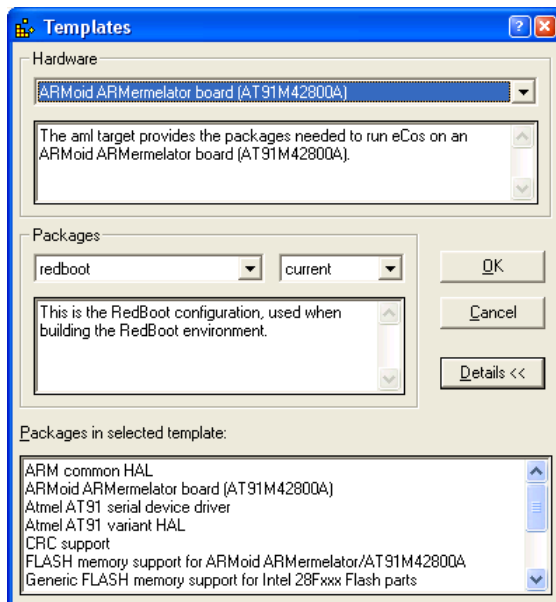


Now enter the path of your user tools selecting the menu Tools->Paths->User Tools... and press OK:



In this case we are using the Cygwin environment. The configuration tool needs to know where to find tools like make to build the eCos libraries and executables.

We will now use an eCos pre-defined template to configure a RedBoot image for the ARMermelator board. Select the menu Build->Templates... and the next pop-up window should appear:



In this pop-up select the "ARMoid ARMermelator board (AT91M42800A)" choice from the Hardware options and the "redboot" package from the Packages list.

Now click OK. The configuration tool checks for dependencies required by the package and resolves conflicts between them recursively. Click on the Continue button to accept the solutions proposed by the configuration tool.

It is time now to configure the package we selected with our custom options. First, we are trying to build a resident RedBoot, so it must be located in ROM. Open the "eCos HAL" branch in the Configuration tree, then select "ARM architecture" and "Atmel AT91 variant HAL". For the "Startup type" option choose "ROM". Again, the configuration tool will check dependencies for you. Click on Continue to resolve them.

The default options of the "redboot" package include the exec command in the "Build Redboot ROM ELF image" options of the "Redboot ROM monitor" node. This command is used to execute non-eCos applications with optional initialization parameters (like a Linux kernel). In our case this command is not

needed so it must be opted out. Uncheck the “Include exec command” option under the “Build Redboot ROM ELF image branch” node.

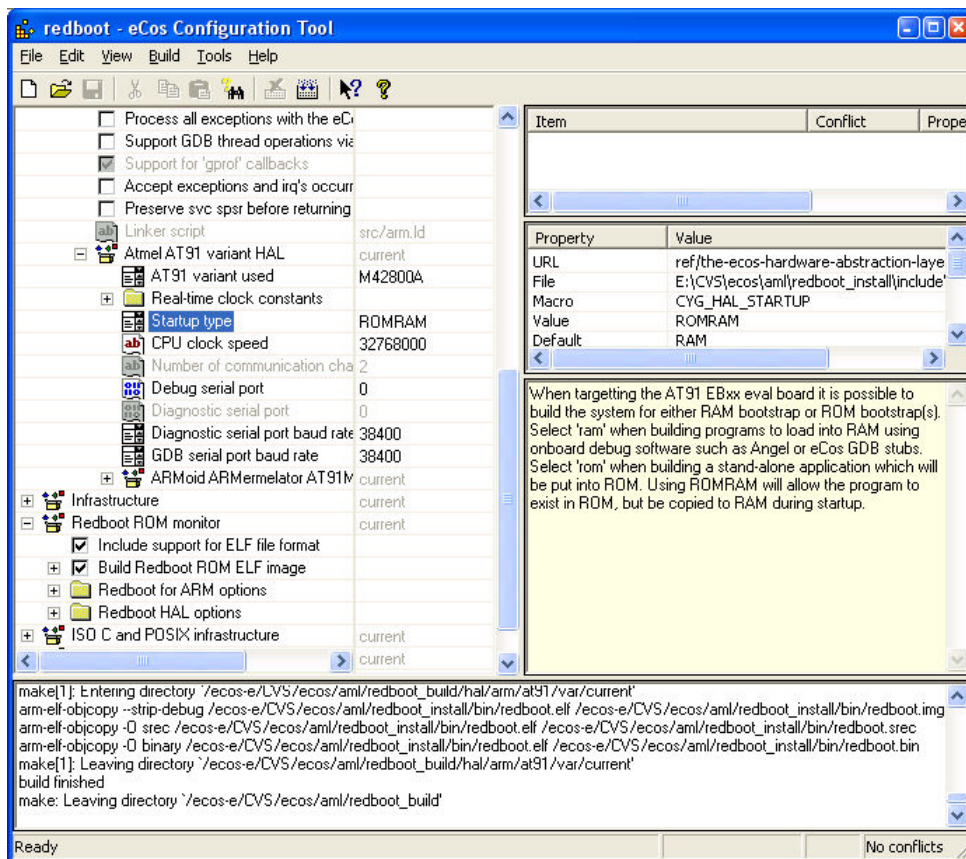
The configuration process for this example is now ready. You must save the configuration in an appropriate place. We suggest you to create an empty subdirectory in your favorite code repository since the configuration tool will build the application images relative to where you store eCos the configuration file (ECC extension). In our case, we have named our configuration “redboot”.

### Building RedBoot

We are now ready to compile our RedBoot image. From the menu bar, select the Build->Generate Build Tree option. The configuration tool will create the following subdirectories relative to the redboot.ecc ECC configuration file:

Directory	Description
redboot_build	Stores object files of the build process
redboot_install	Contains the images, libraries, include files and other necessary files to use the compiled eCos application.
redboot_mlt	Used to store memory layouts to generate targets with different memory layouts.

Now select the menu Build->Library (F7) and the compiling process will begin. Once the build process is finished, your tool window will look like the next figure:



Check out the `redboot_install/bin` directory. The files listed in this table should be there:

File name	Description
<code>redboot.bin</code>	Pure code binary image file
<code>redboot.elf</code>	ELF image file
<code>redboot.img</code>	ELF image file without debugging symbols
<code>redboot.srec</code>	Image in S-record format

Also, the next files must be located in `redboot_install/lib`:

File name	Description
<code>extras.o</code>	Required extra object files for the current package
<code>libextras.a</code>	
<code>libtarget.a</code>	Main package library file
<code>target.ld</code>	Package linker script
<code>vectors.o</code>	Exception vectors and startup code
<code>version.o</code>	Version information

The `redboot_install/include` file contains automatically generated target header files. customized with the package configuration options.

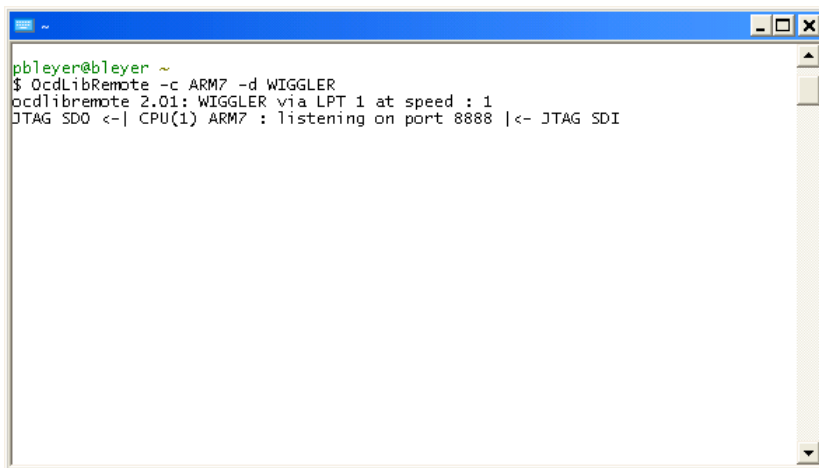
The build process is now complete. We will write the RedBoot binary image to the ARMermelator Flash memory.

## Remote debugger

This example uses Macraigor's OCDLibRemote GDB proxy debugger to connect the ARMermelator board with the host PC using a wiggler-like debug interface. Basically, the debug interface adapts the signal voltage levels of the 20-pin JTAG/ICE header in the AMLprogrammer to the voltage levels of the parallel port in the PC. The OCDLibRemote debugger also supports other debug adapters. You can also use another third party debugger supporting the EmbeddedICE/RT interface.

To start the debugger, in a shell window type:

```
OcdLibRemote -c ARM7 -d WIGGLER 
```




```
pbleyer@bleyer ~  
$ OcdLibRemote -c ARM7 -d WIGGLER  
ocdlibremote 2.01: WIGGLER via LPT 1 at speed : 1  
JTAG SDO <-| CPU(1) ARM7 : listening on port 8888 |<- JTAG SDI
```

---

### Tip



The OCDLibRemote proxy debugger exits every time GDB exits. You can instruct the shell to always restart the debugger easily with the following command:

```
while true; do OcdLibRemote -c ARM7 -d WIGGLER; done 
```

You can also put that command in a custom script. When your debugging session ends, cancel the while loop with Ctrl+C.

---

## Flasher application

We will now build the flasher application and download it to the board in order to write the RedBoot image to the Flash of the ARMermelator.

The ARMoid distribution contains a directory called `tools`. In this directory you will find board support utilities like the `amlboot` Xmodem bootloader and the `amlclear` zero-boot block eraser. These are very simple utilities that help programming and bootstrapping the board for the first time or when there is no initialization code in the Flash memory.

### ***amlboot***

When the processor comes out from reset, only the internal 8kB RAM and the memory space at chip-select `nCS0` are enabled by default. The processor has been designed to boot from this latter memory space, so the device at chip-select `nCS0` and the internal RAM are “remapped” after reset to addresses `0x0000_0000` and `0x0030_0000` respectively. In the ARMermelator board the Flash memory is connected to chip select `nCS0`. Evidently, if the Flash memory is clean there will be no initialization code executed and the board will not be able to boot.

The eCos/RedBoot environment has been designed to initialize its targets properly and to execute either from ROM or RAM. However, a typical RedBoot configuration has a code memory footprint near 50kB. The ARMermelator's AT91M42800A processor has only 8kB of internal RAM. Thus, a small middleware utility is needed to initialize the board adequately and to be able to download programs as RedBoot to the board memories.

The `amlboot` utility has been designed to run in the limited internal RAM space of the AT91M42800A. It loads and executes at address `0x0030_0000` while the stack is located at the top of the internal RAM. Its main job is to receive a program via the Xmodem protocol and write it to the Flash memory beginning at offset `0x0000_0000`.

### ***amlclear***

If there is a valid program stored in the Flash, it will commonly perform processor initialization including the remap command. This means that the internal memory is going to be relocated to absolute address `0x0000_0000` and the Flash memory will begin elsewhere. The `amlclear` utility loads at address `0x0000_0000` and clears the first block of the Flash memory, assuming it has been located at address `0x0100_0000`. When the board is reset it will no longer perform a remap command and it will be possible to load the `amlboot` utility in the internal RAM at the usual address `0x0030_0000`.




---

#### **Note**

Most probably your board was shipped from factory with RedBoot pre-programmed in its Flash memory. It is important for you to know how to debug and reprogram your board in case the RedBoot image gets corrupted by awry code, or if you wish to modify the RedBoot options or even install a completely different program at the boot location.

---

To build the `amlboot` application change to the ARMoid utilities directory and type:

```
make amlboot.elf 
```

The `libao.a` and `libarch.a` libraries and the `amlboot.elf` application will be compiled:

```


pbleyer@pbleyer /e/cvs/aml/tools
$ make amlboot.elf
cd ..; make
make[1]: Entering directory `/ecos-e/cvs/aml'
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o ao/abb.o ao/abb.c
arm-elf-ar rv libao.a ao/abb.o
arm-elf-ar: creating libao.a
a - ao/abb.o
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o ao/ga.o ao/ga.c
arm-elf-ar rv libao.a ao/ga.o
a - ao/ga.o
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o ao/snprintf.o ao/snprintf.c
arm-elf-ar rv libao.a ao/snprintf.o
a - ao/snprintf.o
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o ao/xm.o ao/xm.c
arm-elf-ar rv libao.a ao/xm.o
a - ao/xm.o
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o arch/ga.o arch/ga.c
arm-elf-ar rv libarch.a arch/ga.o
arm-elf-ar: creating libarch.a
a - arch/ga.o
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o arch/io.o arch/io.c
arm-elf-ar rv libarch.a arch/io.o
a - arch/io.o
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o arch/pmc.o arch/pmc.c
arm-elf-ar rv libarch.a arch/pmc.o
a - arch/pmc.o
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o arch/us.o arch/us.c
arm-elf-ar rv libarch.a arch/us.o
a - arch/us.o
rm arch/pmc.o ao/ga.o arch/ga.o arch/us.o arch/io.o ao/xm.o ao/abb.o ao/snprintf
.o
make[1]: Leaving directory `/ecos-e/cvs/aml'
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -o amlboot.o amlboot.c
amlboot.c: In function 'writeFlash':
amlboot.c:152: warning: cast increases required alignment of target type
amlboot.c:160: warning: cast increases required alignment of target type
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -c -Wa,--gstabs,-alhs,-L -o crt0.o crt0.S > crt0.lst
arm-elf-objdump -xdsStr crt0.o >> crt0.lst
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -Wall -Wcast-align
-fverbose-asm -I. -o amlboot.elf amlboot.o crt0.o ../libao.a ../libarch.a -nos
tartfiles -Tiram_br.ld -Wl,--cref,-Map,amlboot.map ../libao.a ../libarch.a > a
mlboot.lst # -nostdlib
arm-elf-objdump -xdsStr amlboot.elf > amlboot.lst
rm amlboot.o crt0.o
pbleyer@pbleyer /e/cvs/aml/tools
$

```

## Downloading and running

We will assume that the Flash memory boot block is clean so the processor will have its internal RAM at address 0x0030\_0000 after reset (no initialization code is executed).

Connect your ARMermelator board to the host using your debugging tool and power the device. To enter the ARM-ELF GNU debugger GDB, in a shell type:

```
arm-elf-gdb amlboot.elf 
```

We will also need an Xmodem-capable terminal utility. If you are working under MS Windows you can use the HyperTerminal application. Connect the serial cable to UART1 of the ARMermelator and open and configure your serial port for 38400 baud, 8 data bits, no parity, 1 stop bit and no flow control.

**Note**

By default, RedBoot uses the AT91M42800A USART0 for the GDB monitor and USART1 for the RedBoot console. In the AMLprogrammer board, the connectors for UART0 and UART1 are labeled RS232A and RS232B respectively.

Now type the following commands:

GDB command	Description
target remote localhost:8888	Will connect to the OCDLibRemote GDB proxy. The remote debugger listens to port 8888. You can also debug remotely to another host, in that case replace localhost appropriately.
load	Will load the code from the <code>amlboot.elf</code> image into the target's memory through the GDB proxy and the JTAG/ICE interface.
break exit	Inserts a breakpoint upon entry to the <code>exit</code> function. This function is entered when <code>main</code> exits so we know that the program has finished running
continue	Will run the program from its <code>start</code> address.

If there are no errors, you should see the following message in your GDB proxy window after issuing the target command:

```
pbleyer@pbleyer ~
$ OcdLibRemote -c ARM7 -d WIGGLER
OcdLibRemote 2.01: WIGGLER via LPT 1 at speed : 1
JTAG SDO <-| CPU(1) ARM7 : listening on port 8888 |<- JTAG SDI
CPU[1] Accepted gdb connection on port 8888.
```

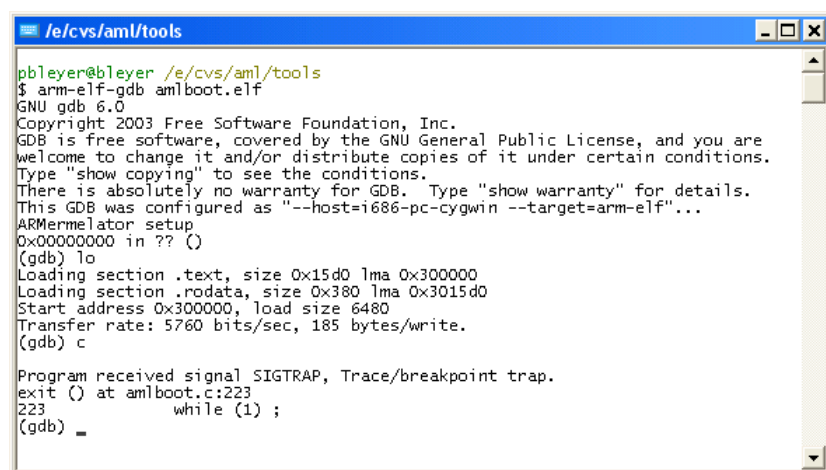
**Tip**

In GDB you can abbreviate commands. For example:

- `lo:` load
- `b:` break
- `c:` continue



The following snapshot shows how your GDB session should look:



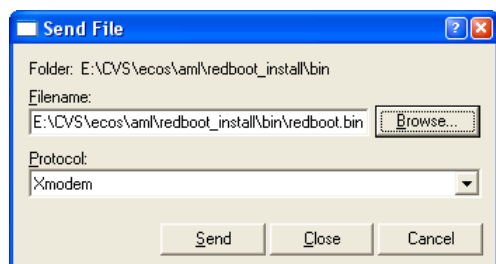
```
/e/cvs/aml/tools
pbleyer@pbleyer /e/cvs/aml/tools
$ arm-elf-gdb amlboot.elf
GNU gdb 6.0
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=arm-elf"...
ARMermelator setup
0x00000000 in ?? ()
(gdb) l0
Loading section .text, size 0x15d0 lma 0x300000
Loading section .rodata, size 0x380 lma 0x3015d0
Start address 0x300000, load size 6480
Transfer rate: 5760 bits/sec, 185 bytes/write.
(gdb) c

Program received signal SIGTRAP, Trace/breakpoint trap.
exit () at amlboot.c:223
223      while (1);
(gdb) _
```

## Using Xmodem

After beginning program execution with GDB's continue command, a "Ready" message will appear on the terminal. Following that, some "C" characters will start to show. This means that the amlboot utility is waiting for Xmodem transfers with the binary image to be flashed.

In HyperTerminal select the Transfer->Send File... menu and the next pop-up will show up:



Browse for the redboot.bin file in the Filename input (located in the redboot\_install/bin directory) and choose Xmodem as the Protocol. Click on Send and the file transfer should begin:





## Advanced

During development, the whole process of manually downloading code with GDB may become tedious. A convenient way to automate this process is to use shell scripts and GDB scripts. For example, an `amlboot.sh` script to download the `amlboot` utility and flash the board may be:

```
#!/ sh
arm-elf-gdb --nx -command=amlboot.gdb
```

This shell script calls the ARM-ELF GDB debugger with the following `amlboot.gdb` GDB script:

```
echo ARMerMelator boot sector flasher\n
file amlboot.elf
set confirm off
set debug remote 0
target remote localhost:8888
load
break exit
continue
quit
```


Then just have your HyperTerminal ready and call the `am1boot.sh` shell script.

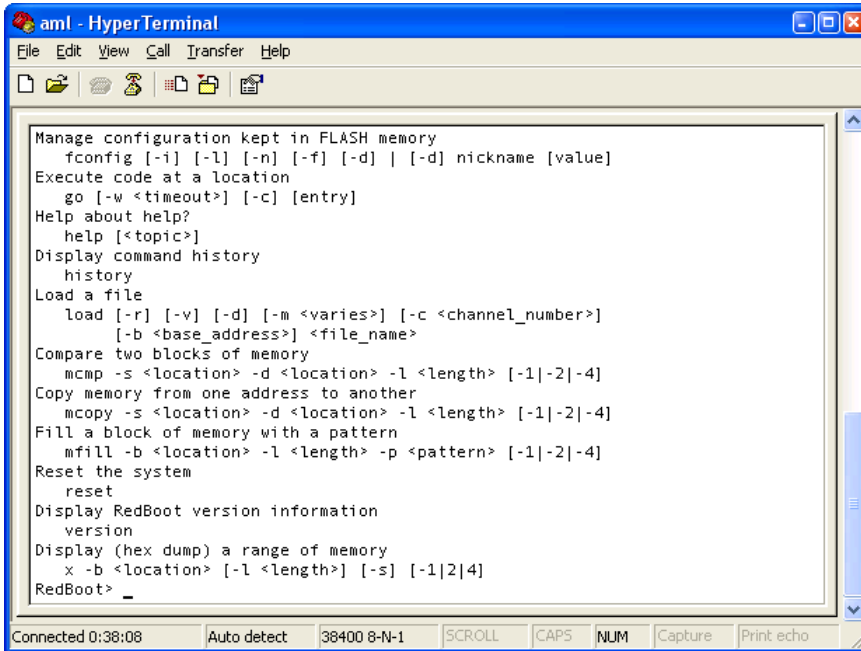
## Using RedBoot

Disconnect now your debug tool from the board and press the reset button. You should see the RedBoot command line:

[illegible]

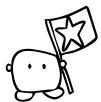
Enter the help command and feel free to try some other commands:

help 



```

aml - HyperTerminal
File Edit View Call Transfer Help
Manage configuration kept in FLASH memory
fconfig [-i] [-l] [-n] [-f] [-d] | [-d] nickname [value]
Execute code at a location
go [-w <timeout>] [-c] [entry]
Help about help?
help [<topic>]
Display command history
history
Load a file
load [-r] [-v] [-d] [-m <varies>] [-c <channel_number>]
[-b <base_address>] <file_name>
Compare two blocks of memory
mcmp -s <location> -d <location> -l <length> [-1|-2|-4]
Copy memory from one address to another
mcopy -s <location> -d <location> -l <length> [-1|-2|-4]
Fill a block of memory with a pattern
mfill -b <location> -l <length> -p <pattern> [-1|-2|-4]
Reset the system
reset
Display RedBoot version information
version
Display (hex dump) a range of memory
x -b <location> [-l <length>] [-s] [-1|2|4]
RedBoot> _
Connected 0:38:08 Auto detect 38400 8-N-1 SCROLL CAPS NUM Capture Print echo
  
```



## Congratulations!

You have just learnt how to use the eCos Configuration Tool to configure and build a RedBoot image for your ARMermelator board. You also used the ARMoid utilities and GNU tools to write binary images in the ARMermelator Flash memory.

## eCos

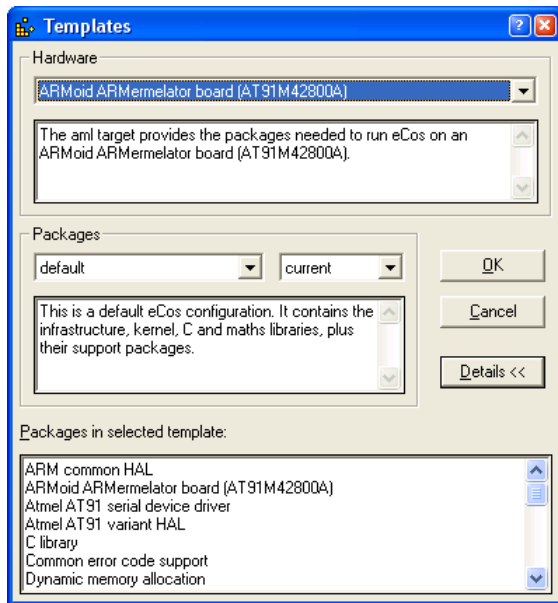
We will now build an eCos kernel and a demonstration program. Then we will download the application to the board using the Xmodem protocol with RedBoot's loader.

As with RedBoot, the eCos development philosophy is that a package is first configured and built. While RedBoot was a stand-alone eCos package, an eCos kernel will be used as the framework of an application. We will compile our eCos program and then link it against the eCos kernel.

## Configuring eCos

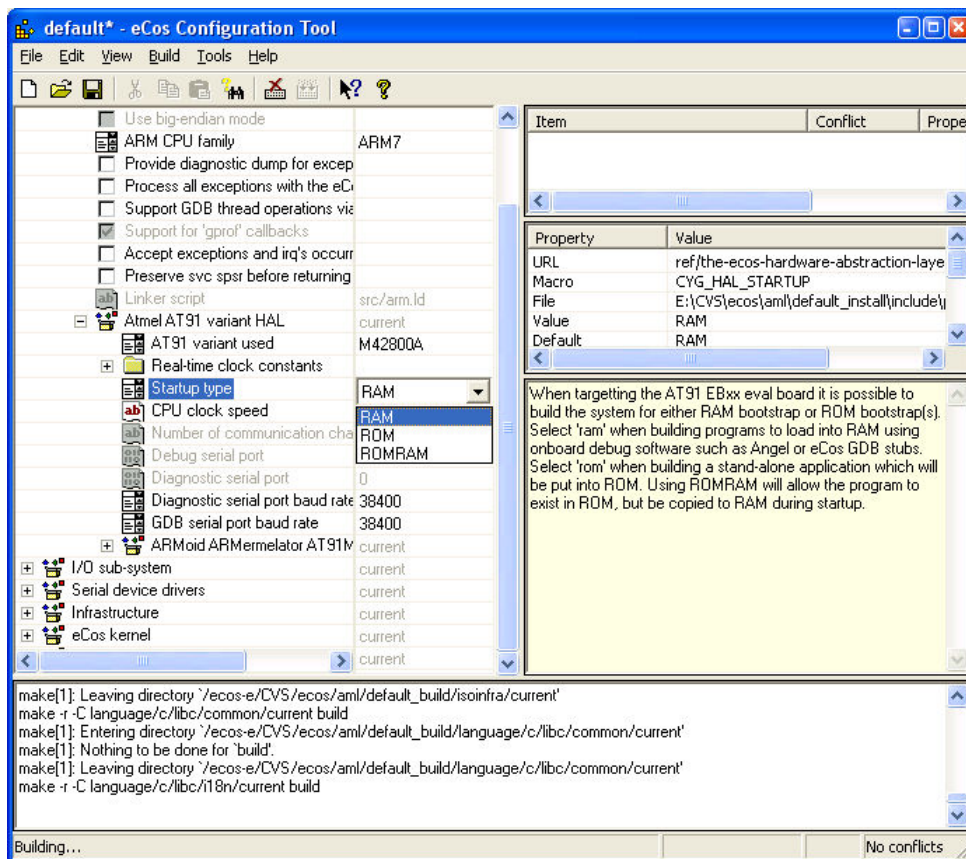
The eCos kernel configuration process follows the same steps as RedBoot's configuration.

Run the eCos Configuration Tool and select the Build->Templates... menu. Select the ARMoid ARMermelator option and the "default" package. Press Continue to resolve dependency conflicts with the proposed options.

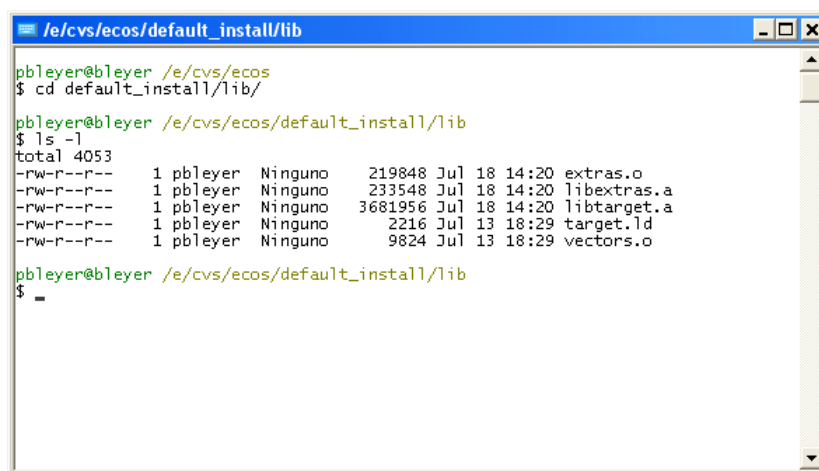


## Building eCos

Since we wish to download the application to the board's external RAM, make sure that the "Startup type" option in the Configuration->eCos HAL->ARM architecture->Atmel AT91 variant HAL branch is of type RAM.



Name your configuration as “default” and save it in an appropriate place. Select Build->Generate Build Tree followed by Build->Library. The compilation process should begin. When finished, check the files in the default\_install/lib subdirectory (relative to the default.ecc configuration file you saved). Your directory should contain the same files listed in this snapshot:



## Using eCos

We just compiled the eCos kernel. The output of this process is the kernel library and associated object files plus the `target.ld` linker script.

To use these files with our application, it is better to write a `Makefile`. Create a directory called “app” in the directory where you placed the `default.ecc` configuration file. Create a text file called “Makefile” in the app directory with the contents shown below:

*Listing 1: Makefile for eCos application*

---

```
# set this variable to point to your kernel install directory
ECOS_INSTALL = ../../ecos/default_install

ARCH = arm-elf
CPU = -mcpu=arm7tdmi
MODEL = -mthumb -mthumb-interwork

CC = $(ARCH)-gcc $(CPU) $(MODEL)
LD = $(ARCH)-ld $(CPU) $(MODEL)
OBJCOPY = $(ARCH)-objcopy
OBJDUMP = $(ARCH)-objdump

INC = -I$(ECOS_INSTALL)/include
LIB = -L$(ECOS_INSTALL)/lib -Ttarget.ld

CFLAGS = -g -O2 -Wall -Wcast-align -fverbose-asm $(INC)
LDFLAGS = -nostdlib -nostartfiles $(LIB)

%.e: %.c
    $(CC) -E -o $@ $^

%.elf: %.o
    $(CC) $^ -o $@ $(LDFLAGS) -Wl,--cref,-Map,$*.map
    $(OBJDUMP) -x $@ > $*.lst

%.srec: %.elf
    $(OBJCOPY) -O srec $^ $@

%.bin: %.elf
    $(OBJCOPY) -O binary $^ $@

clean:
    $(RM) *.e *.o *.elf *.map *.lst *.srec *.bin

distclean: clean
    $(RM) *.bak *~
```

---

In the app directory we will also put our application code. Now launch your favorite text editor and create a C file called “hello.c” with the following code:

*Listing 2: hello.c eCos example*

---

```
/*
    Small eCos test
*/

#include <cyg/kernel/kapi.h> // Kernel API
#include <cyg/kernel/diag.h> // Diagnostics

#define THREAD_STACKSIZE 1024
#define THREAD_PRIORITY 12

int thread_stack[THREAD_STACKSIZE];

cyg_thread thread;
cyg_handle_t handle;

void
thread_0(cyg_addrword_t d) {
    while (1) {
        diag_printf("Thread 0 says hello\n");
        cyg_thread_delay(100);
    }
}

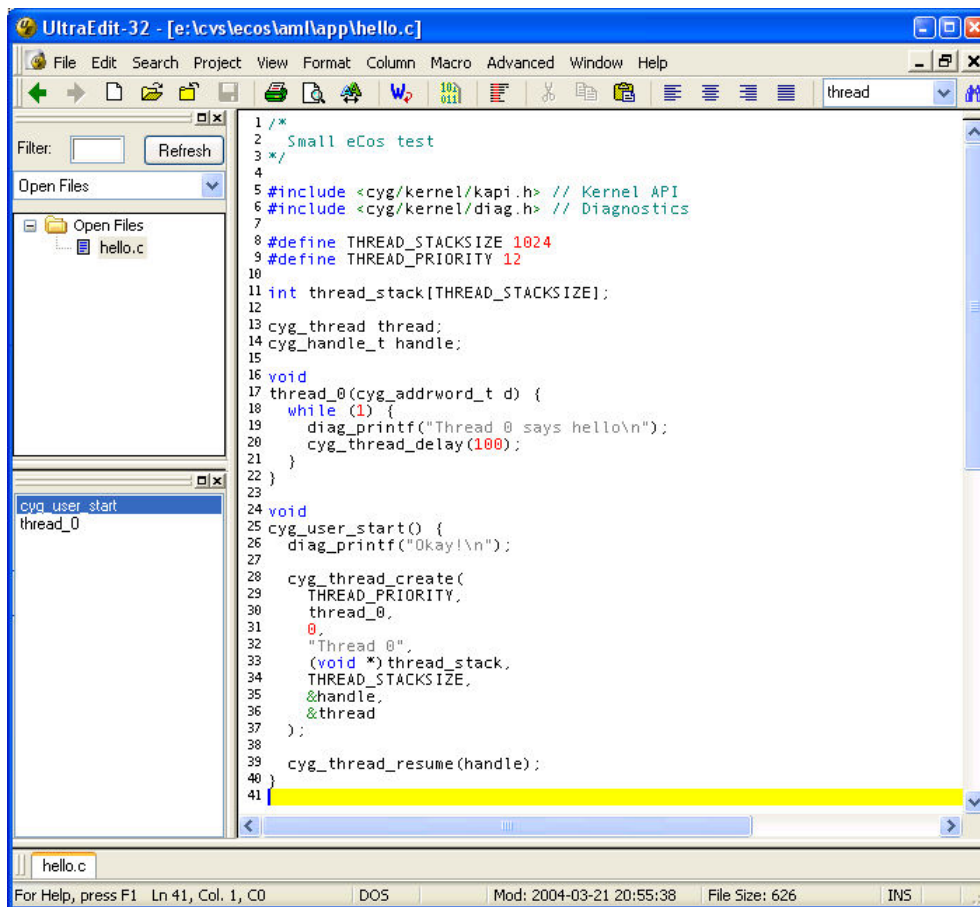
void
cyg_user_start() {
    diag_printf("Okay!\n");

    cyg_thread_create(
        THREAD_PRIORITY,
        thread_0,
        0,
        "Thread 0",
        (void *)thread_stack,
        THREAD_STACKSIZE,
        &handle,
        &thread
    );

    cyg_thread_resume(handle);
}
```

---





Now type:

```
make hello.elf 
make hello.srec 
```

These commands will compile your code into an object file using the headers of the eCos kernel you configured and built. Then it will link the object file with the eCos libraries to generate the ELF executable of your application. The SREC file is the S-record file we will download to the board.

```

/e/cvs/aml/ecos
pbleyer@bleyer /e/cvs/aml/ecos
$ ls
Makefile Makefile.bak default.ecc hello.c old testga.c testleds.c

pbleyer@bleyer /e/cvs/aml/ecos
$ make hello.elf
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork -g -O2 -I../ecos/default_install/include -I../ -c -o hello.o hello.c
arm-elf-gcc -mcpu=arm7tdmi -mthumb -mthumb-interwork hello.o -o hello.elf -nostdlib -nostartfiles -L../ecos/default_install/lib -Ttarget.ld -Wl,--cref,-Map,hello.map
arm-elf-objdump -x dStr hello.elf > hello.lst
rm hello.o

pbleyer@bleyer /e/cvs/aml/ecos
$ make hello.srec
arm-elf-objcopy -O srec hello.elf hello.srec

pbleyer@bleyer /e/cvs/aml/ecos
$ ls -l hello.*
-rw-r--r-- 1 pbleyer Ninguno 626 Apr 4 03:40 hello.c
-rw-r--r-- 1 pbleyer Ninguno 1038236 Jul 18 17:33 hello.elf
-rw-r--r-- 1 pbleyer Ninguno 2329252 Jul 18 17:33 hello.lst
-rw-r--r-- 1 pbleyer Ninguno 439972 Jul 18 17:33 hello.map
-rw-r--r-- 1 pbleyer Ninguno 413352 Jul 18 17:33 hello.srec

pbleyer@bleyer /e/cvs/aml/ecos
$

```

Boot your board and the RedBoot console should appear. Use the load command to download your application to the board's memory with RedBoot:

load -m xmodem

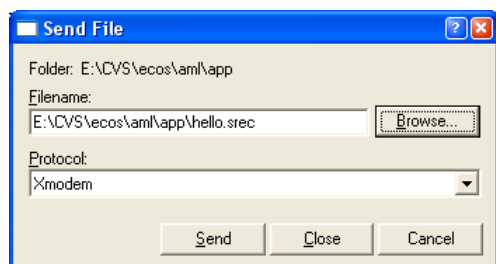
```

aml - HyperTerminal
File Edit View Call Transfer Help

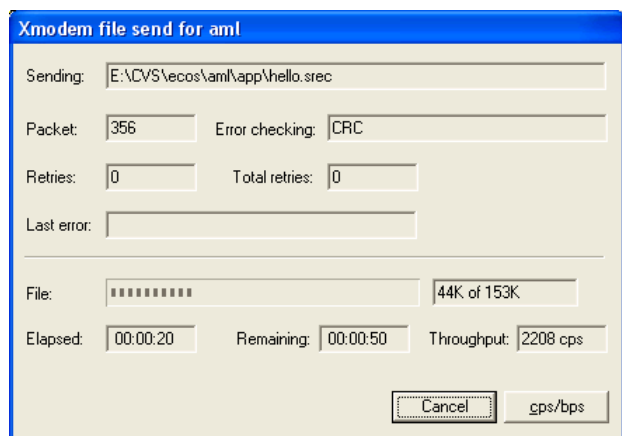
RedBoot> dump -b 0 -l 0x100
00000000: 0E 00 00 EA 18 F0 9F E5 18 F0 9F E5 18 F0 9F E5 | .....p.....|
00000010: 18 F0 9F E5 01 70 A0 E1 18 F0 9F E5 18 F0 9F E5 | .....0.4....|
00000020: 00 01 00 01 00 02 00 01 04 02 00 01 00 03 00 01 | .....3.....|
00000030: 1C 03 00 01 FC 1F 30 00 34 04 00 01 FC 03 00 01 | .....Y...Y..Y..|
00000040: 0A 33 A0 E3 F9 3B 83 E2 BF 2C E0 E3 E6 3F 83 E2 | .....Y...X...Y...|
00000050: 16 00 08 00 94 59 00 01 94 59 00 01 94 59 00 01 | .....Y...Y...Y...|
00000060: 94 59 00 01 60 41 00 02 60 41 00 02 08 5B 00 01 | .....Y...Y...Y...|
00000070: 04 5B 00 01 94 59 00 01 94 59 00 01 94 59 00 01 | .....Y...Y...Y...|
00000080: 9C 59 00 01 7C 5A 00 01 84 01 00 02 94 59 00 01 | .....Y...Z...Y...|
00000090: 7C 59 00 01 00 00 00 00 64 59 00 01 00 00 00 00 | .....dY.....|
000000A0: 94 59 00 01 D8 78 00 01 94 59 00 01 94 59 00 01 | .....X...Y...Y...|
000000B0: 94 59 00 01 94 59 00 01 94 59 00 01 94 59 00 01 | .....Y...Y...Y...|
000000C0: 94 59 00 01 94 59 00 01 94 59 00 01 94 59 00 01 | .....Y...Y...Y...|
000000D0: 94 59 00 01 94 59 00 01 94 59 00 01 AC A9 00 01 | .....Y...Y...Y...|
000000E0: 94 59 00 01 94 59 00 01 94 59 00 01 94 59 00 01 | .....Y...Y...Y...|
000000F0: 94 59 00 01 94 59 00 01 94 59 00 01 94 59 00 01 | .....Y...Y...Y...|
RedBoot> help load
Load a file
load [-r] [-v] [-h <host>] [-m <varies>] [-c <channel_number>]
      [-b <base_address>] <file_name>
RedBoot> load -m xmodem
C_

```

As before with the amlboot and the RedBoot image, select the Transfer->Send File... menu and choose the hello.srec file and Xmodem protocol:

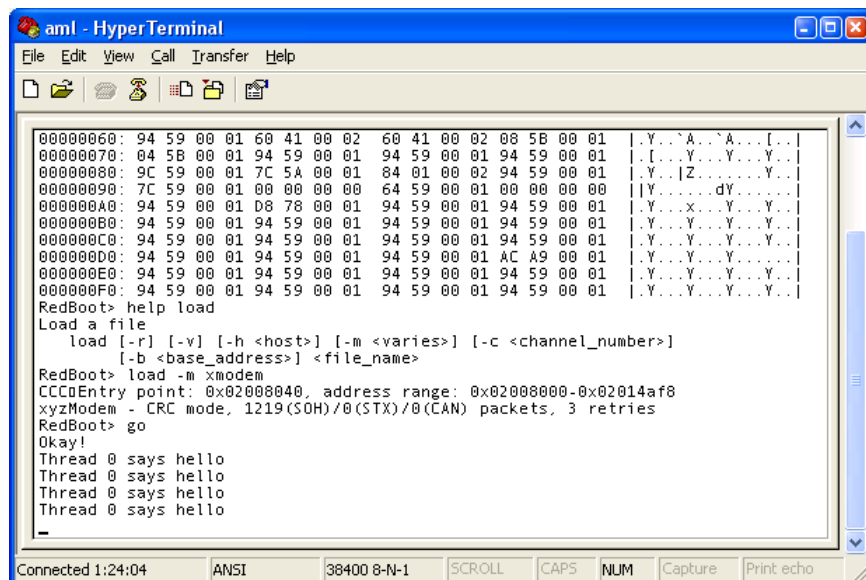


Click on the Send button and the transfer will begin:



Once the transfer is finished, the RedBoot prompt will show again. Issue the go command to instruct RedBoot to jump to your program's start address:

go



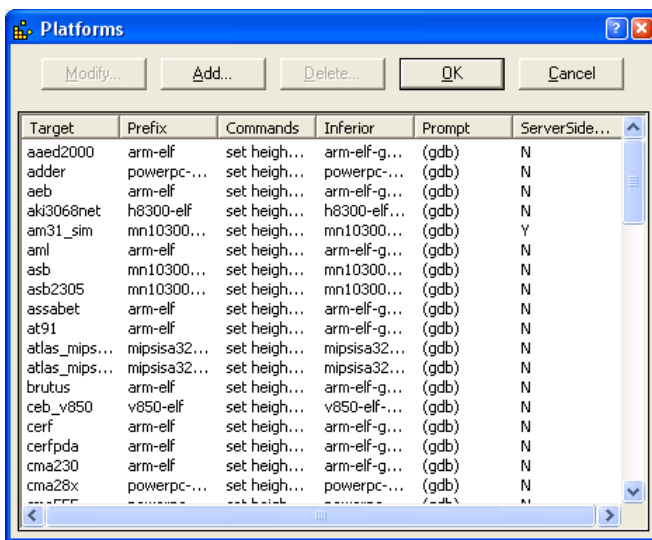
Your program will begin executing now.

## Testing

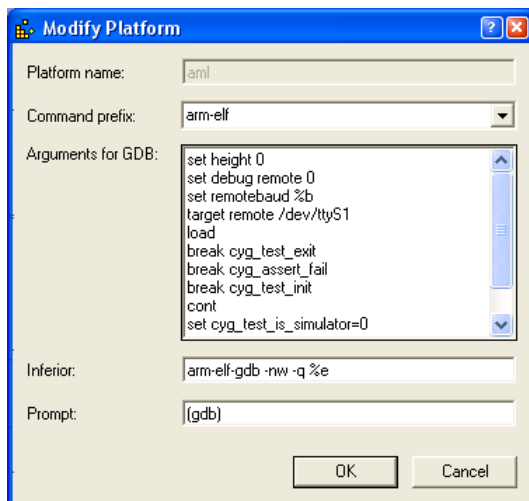
Now that we have built a default eCos kernel, we can build some eCos tests to perform real live checks in our board. As a final activity, we will use the eCos testing facilities that are incorporated in the Configuration Tool.

### Platform configuration

First of all, we need to tell the configuration tool how to communicate with our board. Choose the Tools->Platforms... menu. A pup-up window will appear:



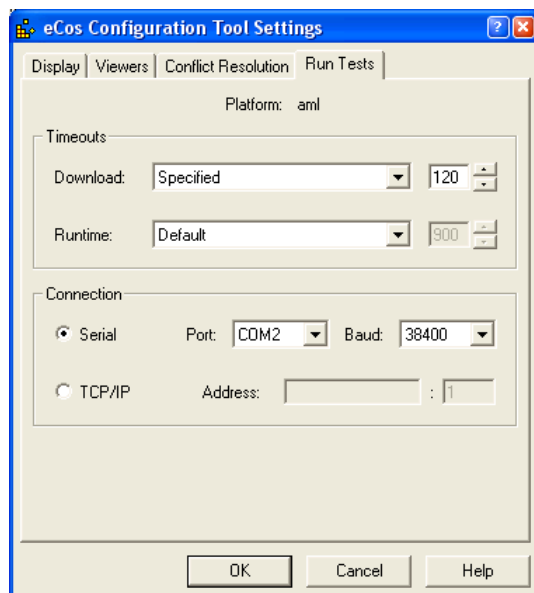
Press the Add button and a second pop-up will show up. Fill the form fields as illustrated in the next figure:



Example:

<b>Platform name</b>	aml
<b>Command prefix</b>	arm-elf
<b>Arguments for GDB</b>	set height 0 set debug remote 0 set remotebaud %b target remote /dev/ttyS0 load break cyg_test_exit break cyg_assert_fail break cyg_test_init cont set cyg_test_is_simulator=0 cont bt
<b>Inferior</b>	arm-elf-gdb -nw -q %e
<b>Prompt</b>	(gdb)

The wildcards %b, %p and %e may be used for the baud rate, port and GDB arguments. Now press OK to remove the pop-ups. Select the View->Settings... menu and check that the parameters under the "Run Tests" tab are correct for your system:

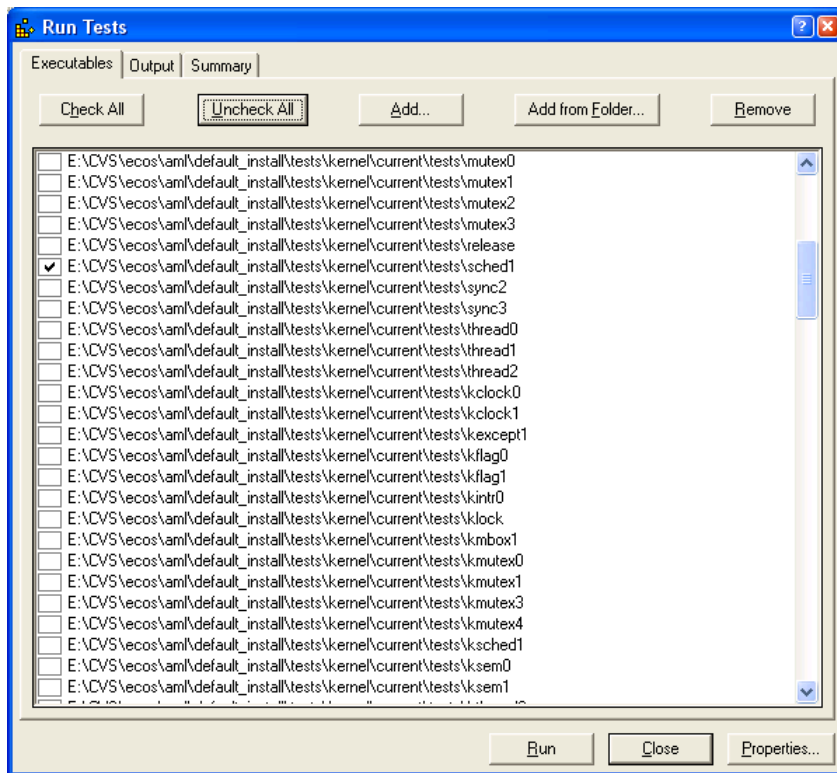


Press OK.

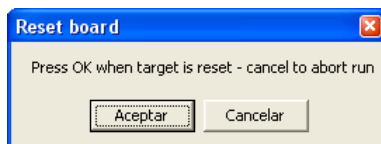
## Building and running tests

With the "default" configuration open, select the Build->Tests menu. The compilation of the test images will begin.

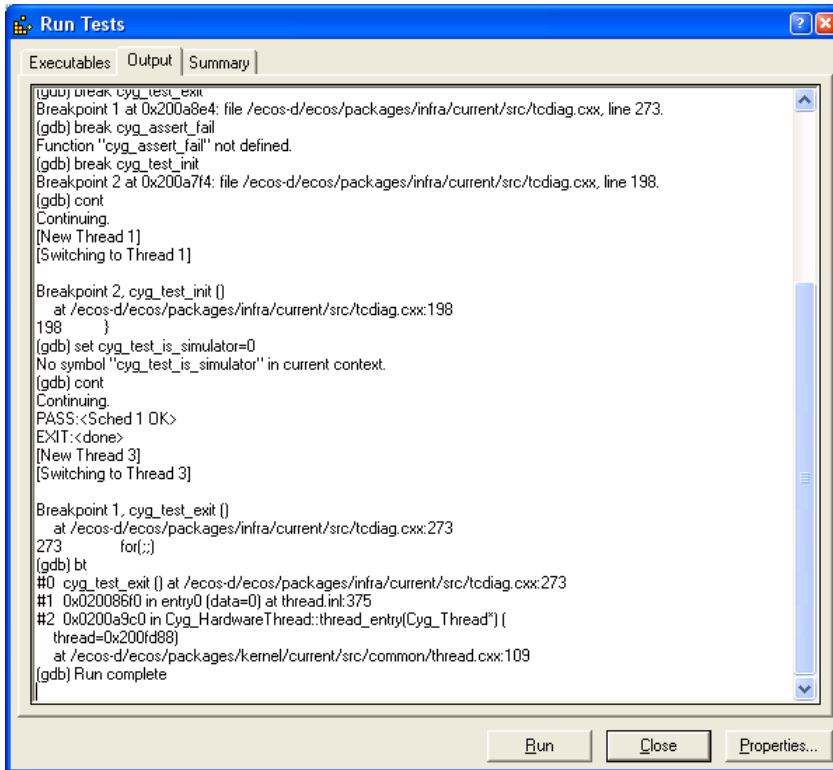
After the build process stops, select the Tools->Run Tests... menu. The following pop-up will show up:



You can now select the tests you wish to run. When you are done, press the Run button and the testing process will start:



Make sure that the serial port you have selected corresponds to the AT91M42800A USART0 (RS232A connector of the AMLprogrammer board); this is where RedBoot's GDB monitor listens remote GDB requests. Reset your board and press the OK button. The tests will be executed one-by-one. You can take a peek to what is happening selecting the "Output" tab:



The screenshot shows a window titled "Run Tests" with three tabs: "Executables", "Output", and "Summary". The "Output" tab is selected, displaying a GDB transcript. The transcript shows the setup of breakpoints for `cyg_test_exit` and `cyg_test_init`, followed by the execution of the test. It includes messages like "[gdb] break cyg\_test\_init", "Breakpoint 2 at 0x200a714: file /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx, line 198.", and "(gdb) cont". The test passes, indicated by "PASS:<Sched 1 OK>" and "EXIT:<done>". Finally, a backtrace is shown for `cyg_test_exit` at line 273, showing the call stack with the `Cyg_HardwareThread::thread_entry` function.

```
(gdb) break cyg_test_exit
Breakpoint 1 at 0x200a8e4: file /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx, line 273.
(gdb) break cyg_assert_fail
Function "cyg_assert_fail" not defined.
(gdb) break cyg_test_init
Breakpoint 2 at 0x200a714: file /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx, line 198.
(gdb) cont
Continuing.
[New Thread 1]
[Switching to Thread 1]

Breakpoint 2, cyg_test_init ()
  at /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx:198
198      }
(gdb) set cyg_test_is_simulator=0
No symbol "cyg_test_is_simulator" in current context.
(gdb) cont
Continuing.
PASS:<Sched 1 OK>
EXIT:<done>
[New Thread 3]
[Switching to Thread 3]

Breakpoint 1, cyg_test_exit ()
  at /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx:273
273      for(;;)
(gdb) bt
#0  cyg_test_exit () at /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx:273
#1  0x020086f0 in entry0 (data=0) at thread.int:375
#2  0x0200a9c0 in Cyg_HardwareThread::thread_entry(Cyg_Thread*) {
  thread=0x200fd88
  at /ecos-d/ecos/packages/kernel/current/src/common/thread.cxx:109
(gdb) Run complete
```

The transcript of a test session is shown below:

*Listing 3: eCos test session*


---

```

Run started
(gdb) set height 0
(gdb) set debug remote 0
(gdb) set remotebaud 38400
(gdb) target remote /dev/ttyS1
Remote debugging using /dev/ttyS1
0x01002dc4 in ?? ()
(gdb) load
Loading section .rom_vectors, size 0x40 lma 0x2008000
Loading section .text, size 0x62a8 lma 0x2008040
Loading section .rodata, size 0x43c lma 0x200e2e8
Loading section .data, size 0x34c lma 0x200e724
Start address 0x2008040, load size 27248
Transfer rate: 27248 bits/sec, 309 bytes/write.
(gdb) break cyg_test_exit
Breakpoint 1 at 0x200a8e4: file /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx,
line 273.
(gdb) break cyg_assert_fail
Function "cyg_assert_fail" not defined.
(gdb) break cyg_test_init
Breakpoint 2 at 0x200a7f4: file /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx,
line 198.
(gdb) cont
Continuing.
[New Thread 1]
[Switching to Thread 1]

Breakpoint 2, cyg_test_init ()
    at /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx:198
198     }
(gdb) set cyg_test_is_simulator=0
No symbol "cyg_test_is_simulator" in current context.
(gdb) cont
Continuing.
PASS:<Sched 1 OK>
EXIT:<done>
[New Thread 3]
[Switching to Thread 3]

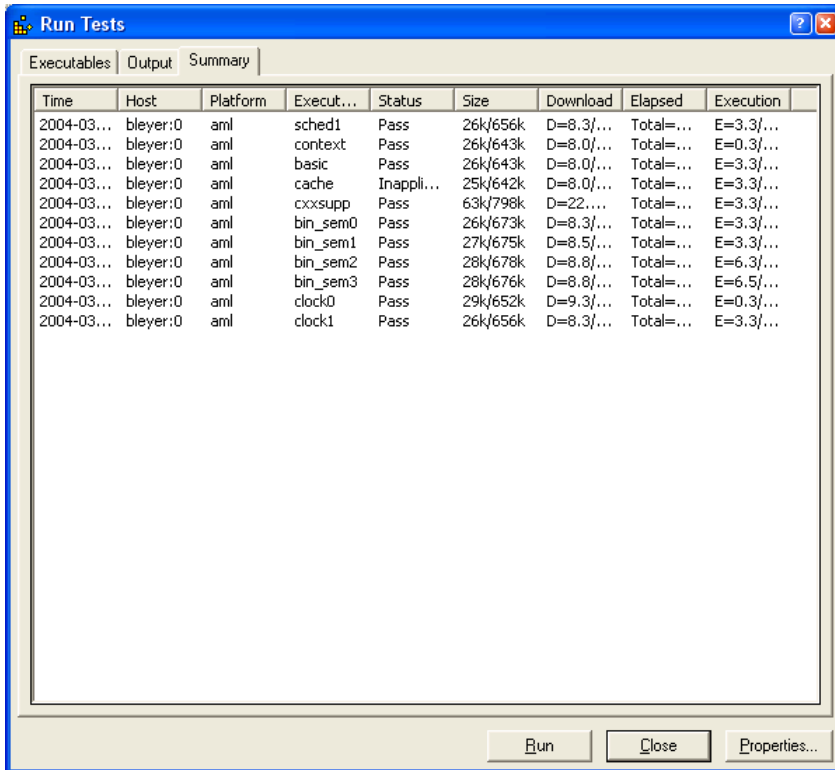
Breakpoint 1, cyg_test_exit ()
    at /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx:273
273     for(;;)
(gdb) bt
#0  cyg_test_exit () at /ecos-d/ecos/packages/infra/current/src/tcdiag.cxx:273
#1  0x020086f0 in entry0 (data=0) at thread.inl:375
#2  0x0200a9c0 in Cyg_HardwareThread::thread_entry(Cyg_Thread*) (
    thread=0x200fd88)
    at /ecos-d/ecos/packages/kernel/current/src/common/thread.cxx:109
(gdb) Run complete

```

---

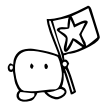
The “Summary” tab lists information and results of the eCos tests:





The screenshot shows a 'Run Tests' window with a table of test results. The table has columns for Time, Host, Platform, Execut..., Status, Size, Download, Elapsed, and Execution. The tests listed are sched1, context, basic, cache, cxxsupp, bin\_sem0, bin\_sem1, bin\_sem2, bin\_sem3, clock0, and clock1. All tests passed.

Time	Host	Platform	Execut...	Status	Size	Download	Elapsed	Execution
2004-03...	bleyer:0	aml	sched1	Pass	26k/656k	D=8.3/...	Total=...	E=3.3/...
2004-03...	bleyer:0	aml	context	Pass	26k/643k	D=8.0/...	Total=...	E=0.3/...
2004-03...	bleyer:0	aml	basic	Pass	26k/643k	D=8.0/...	Total=...	E=3.3/...
2004-03...	bleyer:0	aml	cache	Inappli...	25k/642k	D=8.0/...	Total=...	E=3.3/...
2004-03...	bleyer:0	aml	cxxsupp	Pass	63k/798k	D=22....	Total=...	E=3.3/...
2004-03...	bleyer:0	aml	bin_sem0	Pass	26k/673k	D=8.3/...	Total=...	E=3.3/...
2004-03...	bleyer:0	aml	bin_sem1	Pass	27k/675k	D=8.5/...	Total=...	E=3.3/...
2004-03...	bleyer:0	aml	bin_sem2	Pass	28k/678k	D=8.8/...	Total=...	E=6.3/...
2004-03...	bleyer:0	aml	bin_sem3	Pass	28k/676k	D=8.8/...	Total=...	E=6.5/...
2004-03...	bleyer:0	aml	clock0	Pass	29k/652k	D=9.3/...	Total=...	E=0.3/...
2004-03...	bleyer:0	aml	clock1	Pass	26k/656k	D=8.3/...	Total=...	E=3.3/...



### Congratulations!

You now know how to build eCos applications and download them to the ARMermelator board using RedBoot. You have used RedBoot's built-in GDB remote debugging agent to test eCos programs in the ARMermelator board.

## References

1. Atmel AT91M42800A datasheet [DOC1779] [<http://www.atmel.com/products/AT91>]
2. Intel Advanced+ Boot Block Flash Memory (C3) [290645] [<http://www.intel.com/design/flash>]
3. eCos Documentation [<http://ecos.sourceware.org/docs.html>]
4. OCDemon GNU Tools from Macraigor Systems [[http://www.macraigor.com/full\\_gnu.htm](http://www.macraigor.com/full_gnu.htm)]
5. GNUARM toolchain [<http://www.gnuarm.com>] [also available at <http://armoid.com/gnuarm>]
6. GNU Manuals Online [<http://www.gnu.org/manual>]
7. Cygwin GNU environment for Windows [<http://www.cygwin.com>]

## Document history

Version	Date	Comments
1.0	2004-03-26	First release.
1.1	2004-04-04	Added "Testing" section and snapshots.
1.2	2004-04-13	Added block diagrams and mechanical dimensions.
1.3	2004-06-18	Added power connector polarity and JTAG/ICE pod insertion illustrations.
1.4	2004-07-18	Added extra AMLprogrammer information. Updated tools compilation and snapshots for the latest versions.