NEWWOOD SOLUTIONS Ltd
Derby Office: 15 Kings Croft, Allestree, Derby, DE22 2FP.
Tel +44 (0)1332 721326

Reading Office: 13 Highfield Road, Tilehurst, Reading RG31 6YR.
Tel +44 (0) 118 9012298

Email: sales@newwoodsolutions.co.uk Web: www.newwoodsolutions.co.uk

9010 IOC User Manual

1U Rack Mounted Input Output Controller
6 Industry Pack and 1 PMC Carrier

Document Reference:	**IOC9010/UM/2.3**
Issue:	**Version 2.3**
Original authors:	**D. Nineham, P. Marshall**
Update authors:	**G. Cross, J. Chen. M. Newman**
Date:	**November 2017**

Issuing Organisation : Hytec Electronics Ltd
Classification : Unclassified

Distribution List

COPY	REGISTERED HOLDER
MASTER	Newwood Solutions Ltd

Version control

VERSION	DATE	Description	Author
Draft	Oct 2006	Original draft sent for approval	D Nineham
V1.0	Nov 2006	Approved version	D Nineham
V1.1	Jan 2007	Corrections and additions	D Nineham
V1.2	May 2009	Further corrections	P Marshall
V2.0	Mar 2014	Revised and updated	G Cross/J Chen/M Newman
V2.3	28/11/17	Change from Hytec to Newwood Solutions for contact details	M Newman

CRITICAL APPLICATIONS DISCLAIMER

THIS PRODUCT FROM NEWWOOD SOLUTIONS LTD USES COMPONENTS THAT ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS IN LIFE-SUPPORT OR SAFETY DEVICES OR SYSTEMS, CLASS III MEDICAL DEVICES, NUCLEAR FACILITIES, APPLICATIONS RELATED TO THE DEPLOYMENT OF AIRBAGS, OR ANY OTHER APPLICATIONS THAT COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE (INDIVIDUALLY AND COLLECTIVELY, “CRITICAL APPLICATIONS”).

FURTHERMORE, SOME COMPONENTS USED IN THIS NEWWOOD SOLUTIONS LTD PRODUCTS ARE NOT DESIGNED OR INTENDED FOR USE IN ANY APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE OR AIRCRAFT, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR.

THE CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF NEWWOOD SOLUTIONS LTD PRODUCT IN CRITICAL APPLICATIONS.

CONTENTS

1. INTRODUCTION

The 9010 IOC is a 1U high (1.75”), 19” Rack Mounted Input / Output controller, designed to carry up to 6 single width Industry Pack cards (IP), each with its own signal conditioning card (SCC) slot and a single PCI Mezzanine Card (PMC) card. The CPU is an Intel ATOM 1.6GHz processor with a PCI bus interface. The CPU has 2GB ram and a 120GB SATA hard drive. It has VGA, keyboard, mouse and USB 2 interface connections. Each industry pack card has a rear panel high density 50 way scsi 2 connector to connect the external I/O signals.

The IOC is designed to integrate with our existing range of 50 way scsi connected DIN rail mounting terminal blocks.

Hytec can configure the INTEL ATOM CPU to run with Windows 7, versions of Linux and Rtems real time executive.

The logic on the 9010 IOC motherboard includes a PLX PCI bridge chip and Spartan 2 FPGA chip programmed to interface the Industry Pack address space to the PCI bus address space. Additionally, support is provided for the IOC front panel 2 line character display and the various switches.

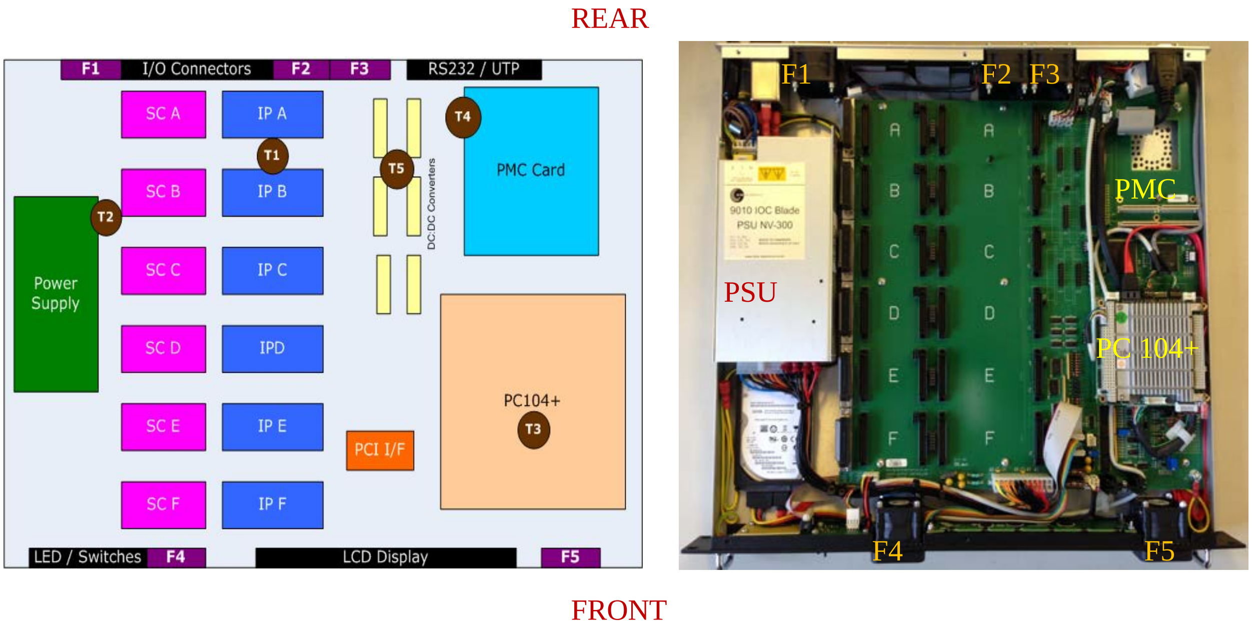
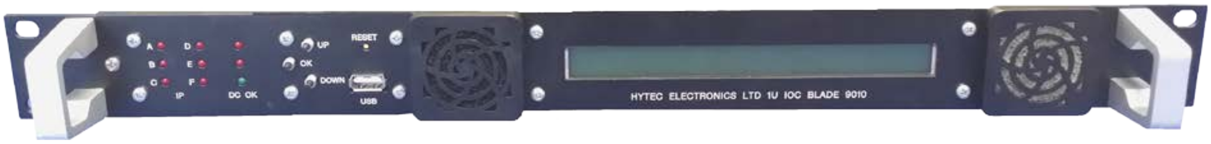


Figure 1: IOC Blade 9010 System Block Diagram

Key:
F1 to F5 - Temperature controlled cooling fans
T1 to T5 – Temperature sensors for fan speed control

Fan speed control is managed by the on board FPGA switching the fans from off to low, medium or high speed as the local area temperature increases. An application program interface (API) is available to allow remote monitoring and control of fan speeds. The API also includes routines for temperature monitoring, writing the two 40 character text lines on the front panel display and read back of the front panel and internal switches.

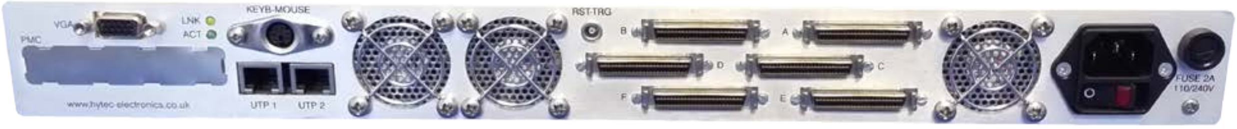
1.1 9010 IOC Front panel



The 9010 IOC front panel has the following features:

- Two aluminium carry handles. Handles to help install and remove the IOC.
- Six red IP slot activity LEDs – A, B, C, D, E, F. Each flash for approx 0.25 second when the associated IP slot is addressed correctly.
- One red addresses LED. Flashes once for approx 0.25 second for any IP slot addressed correctly.
- One red ‘fault’ led. Flashes once for approx 0.25 second for any IP slot where the addressed acknowledge is not detected (empty IP slot or IP card fault).
- One green DC OK led. Indicates the all the DC lines required are at OK voltage levels.
- Three push buttons (UP, OK, DOWN) that can be used by applications programs, typically to drive a menu system in association the two line 40 character display. Can be read by the 9010 IOC API library routines.
- Recessed RESET button. Can be used to reset the CPU when internally linked (not always linked).
- USB port. Internally connected to the PC104+ USB port. This is USB 2 with the ATOM CPU.
- Two air in-taking cooling fans with push-on filter covers. These filter covers can be removed to clean or replace the filters.
- One rear illuminated LCD display. This is a 2 line 40 character display that can be written to by the 9010 IOC API library routines. It displays the power on self-test message by default. Should say “Test OK”. The level of illumination can be adjusted on an internal potentiometer – VR1

1.2 9010 IOC rear panel



The 9010 IOC front panel has the following features:

- One IEC switched mains connector with power on neon lamp. Will accept 110 – 230V at 50 – 60 Hz ac.
- One 2A mains fuse.
- Six off 50 way SCSI connector sockets. One socket for each internal Industry Pack card slot, labelled A to F.
- Three air extracting cooling fans.
- One PCI Mezzanine (PMC) removable blank plate.
- Two UTP connector ports.
  - UTP 1 is connected to the Ethernet port on the PC104+. The ATOM CPU sets this to a 10 / 100Mhz Ethernet connection.
  - UTP-2 not normally used. UTP-2 can be configured as a second Ethernet port if the CPU allows it or as an RS232 / RS422 port.
- Ethernet LINK established LED – yellow.
- Ethernet LINK activity LED – green.
- One combined PS2 keyboard and mouse port (requires the special Y splitter cable to give separate mouse and keyboard connections).
- One standard 15 pin VGA socket connector
- One LEMO connector for either timing sync / inhibit inputs or watchdog outputs.

2. OPERATING MODES

There are several basic operating systems / software / protocols to access hardware inpu / output.modes that the 9010 IOC Blade will support…

Linux / EPICS

Hytec can configure and provide software drivers for all Hytec IP cards for various flavours of Linux (RedHat, Scientific, Debian, others). Hytec also supply device and asyndriver support for EPICS and will pre-configure the EPICS environment. This will allow EPICS users via straight CA (Channel Access), EDM, MEDM, Control System Studio (CSS), BOY and other EPICS utilities to access the IOC’s interfaces.

Linux or Windows / VSYSTEM (from VISTA Control Systems Inc.)

Hytec can configure the 9010 IOC with Vista Control Systems Inc’s Vsystem real time SCADA software. This can be configure on either a Windows or Linux operating system platform. Hytec provide device drivers and an applications API to communicate with our Industry Packs. Hytec can also provide Vsystem readers and handlers to support the system I/O into the Vsystem database.

RTEMS / EPICS

RTEMS is an open source real time executive offering the maximum I/O performance on our 9010 IOC. RTEMS is a similar architecture to the popular VxWorks real time system but without the burden of license costs. RTEMS is well supported by the RTEMS community. Hytec have built the Board Support Package (BSP) for the Intel ATOM PC104 CPU that is required to operate RTEMS and the RTEMS support for our Industry Pack cards. The 9010 IOC Blade runs in the same way as a VxWorks IOC would, i.e. it boots up, connects to a host and runs off a start up script.

Windows / OPC Server

The 9010 IOC Blade also supports an OPC UA Server running on a Windows operating system. The Industry Pack (IP) card I/Os appear as OPC tags that can be discovered by an OPC UA client.

3. The PCI bus register interface

The PLX PCI 9030 bridge device requests resources from the PC104+ processor as follows:

- An I/O area for access to the configuration registers (not used).
- A MEMORY area for access to the configuration registers (not used).
- An **I/O area** for access to the internal registers of the Xilinx (Carrier Board registers).
- A MEMORY area for access to the Industry Packs.

This **I/O area** is 64 bytes wide, organised as 32 16-bit words, starting at offset zero as follows:

Offset	Name	Description
0	CSR-CB	Carrier board Control and Status Register.
2	CONFIG	Carrier Board Switches and settings.
4	DISP_CONT	Read/write access to the LCD display control register
6	DISP_DATA	Read/write access to the LCD display data register
8	INTS_LO	Read only access to the IP IRQ Status Register (12 bits)
A	INTS_HI	Read only access to the IP Error Status Register (7 bits)
C	MASK_LO	Read/write access to a mask register for IP IRQ sources
E	MASK_HI	Read/write access to a mask register for IP Error sources
10	IP_CLK	Read/write access the IP CLOCK SELECT register (6 bits)
12	FAN_1_2	Read only register showing the speeds of fans 1 & 2 in RPS.
14	FAN_3_4	Read only register showing the speeds of fans 3 & 4 in RPS.
16	FAN_5_6	Read only register showing the speeds of fans 5 & 6 in RPS.
18	FAN_CONT	Read/Write access to Control bits for each fan
1A	TEMP_FLAG	Read only access to temperature sensor output flags.
1C	CONFIG_2	Read only access to the two 8-bit configuration switch packs.

4. PRODUCT SPECIFICATIONS

Size:	1U 19” Rack 400mm Deep Approx
Operating temp:	0 to 45 deg C ambient
Number of input/outputs:	6 SCSI Style Connectors providing connection to the IP Cards.
Power:	+100 to 240VAC at 47-63Hz 20W Maximum Unpopulated.

5. APPLICATION REGISTERS

The following is a description of the application registers that are available for low level programming users. It should be noted that Hytec provide a device application program interface (API) that provides higher level support to access the 9010 IOC functions. A descrition of this API can be found in the Hytec document xxxyyyzzz.

Carrier Board Control/Status Register CSR-CB Address 0 (Read/Write)

Address: Base + 0x0

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	FCON	0	TP16	TP15	FAN_6	FAN_5	FAN_4	FAN_3	FAN_2	FAN_1	0	TIMO	PMC

PMC a ‘1’ indicates a valid PMC card detected.
TIMO a ‘1’ indicates the last attempted access to an IP card timed out.
FAN_1-6 a ‘1’ indicates detected fan rotation (see also fan control monitoring registers).
TP15-16 from on-board test points (with pull-ups) inverted, so normally ‘0’.
FCON one writeable and readable bit to select the cooling fan control method. ‘0’ = local automatic control, ‘1’ = remote control through the register at offset 18 HEX.

Carrier Board Config and Switches Register IMR IP Address 2 (Read Only)

Address: Base + 0x2

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
								SP_4	SP_3	SP_2	SP_1	RESET	DOWN	OK	UP

UP a ‘1’ indicates the UP Button on the Front Panel is pressed.
OK a ‘1’ indicates the OK Button on the Front Panel is pressed.
DOWN a ‘1’ indicates the DOWN Button on the Front Panel is pressed.
RESET a ‘1’ indicates the recessed RESET Button on the Front Panel is pressed.
SP_1-4 a ‘1’ indicates the on-board switch point is closed. (4 Spare / Extra Switch Points).

DISP CONT/DATA. (Offsets 4, 6)

Registers for direct R/W access to the front panel LCD display. The Control register is at offset 4, and the Data register at offset 6.

INTS LO. Register Address 8 (Read Only)

Address: Base + 0x8

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	X	X	X	INT REQ F1	INT REQ F0	INT REQ E1	INT REQ E0	INT REQ D1	INT REQ D0	INT REQ C1	INT REQ C0	INT REQ B1	INT REQ B0	INT REQ A1	INT REQ A0

INT REQ A0 a ‘1’ indicates the Interrupt Source was Input 0 on Industry Pack A.
INT REQ A1 a ‘1’ indicates the Interrupt Source was Input 1 on Industry Pack A.
INT REQ B0 a ‘1’ indicates the Interrupt Source was Input 0 on Industry Pack B.
INT REQ B1 a ‘1’ indicates the Interrupt Source was Input 1 on Industry Pack B.
INT REQ C0 a ‘1’ indicates the Interrupt Source was Input 0 on Industry Pack C.
INT REQ C1 a ‘1’ indicates the Interrupt Source was Input 1 on Industry Pack C.
INT REQ D0 a ‘1’ indicates the Interrupt Source was Input 0 on Industry Pack D.
INT REQ D1 a ‘1’ indicates the Interrupt Source was Input 1 on Industry Pack D.
INT REQ E0 a ‘1’ indicates the Interrupt Source was Input 0 on Industry Pack E.
INT REQ E1 a ‘1’ indicates the Interrupt Source was Input 1 on Industry Pack E.
INT REQ F0 a ‘1’ indicates the Interrupt Source was Input 0 on Industry Pack F.
INT REQ F1 a ‘1’ indicates the Interrupt Source was Input 1 on Industry Pack F.

INTS HI Register Address A (Read Only)

Address: Base + 0xA

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	X	X	X	X	X	X	TIMO	X	X	ERR_ F	ERR_E	ERR_D	ERR_C	ERR_B	ERR_A

ERR_A a ‘1’ indicates the Interrupt Source was the Error Output From Industry Pack A.
ERR_B a ‘1’ indicates the Interrupt Source was the Error Output From Industry Pack B.
ERR_C a ‘1’ indicates the Interrupt Source was the Error Output From Industry Pack C.
ERR_D a ‘1’ indicates the Interrupt Source was the Error Output From Industry Pack D.
ERR_E a ‘1’ indicates the Interrupt Source was the Error Output From Industry Pack E.
ERR_F a ‘1’ indicates the Interrupt Source was the Error Output From Industry Pack F.
TIMO a ‘1’ indicates the Interrupt Source was Timeout for accessing any Industry Pack.

MASK LO/HI (Offsets Ch, Eh)

Registers corresponding to Interrupt and Error flag bits in INTS LO & INTS HI above, to select which, if any, are permitted to produce a PC104+ processor interrupt. This interrupt is presented to the PC104+ card on PCI interrupt INTA#.

IP CLOCK Register Address 10 (Read/Write)

Address: Base + 0x10

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	X	X	X	X	X	X	X	X	X	CKS_F	CKS_E	CKS_D	CKS_C	CKS_B	CKS_A

CKS_A a ‘1’ sets the clock for Industry Pack A to 32MHz and ‘0’ sets it 8MHz.
CKS_B a ‘1’ sets the clock for Industry Pack B to 32MHz and ‘0’ sets it 8MHz.
CKS_C a ‘1’ sets the clock for Industry Pack C to 32MHz and ‘0’ sets it 8MHz.
CKS_D a ‘1’ sets the clock for Industry Pack D to 32MHz and ‘0’ sets it 8MHz.
CKS_E a ‘1’ sets the clock for Industry Pack E to 32MHz and ‘0’ sets it 8MHz.
CKS_F a ‘1’ sets the clock for Industry Pack F to 32MHz and ‘0’ sets it 8MHz.

All bits default to ‘0’ (i.e. 8Mhz) on power-up.

FANS_1_2 Register Address 12 (Read Only)

Address: Base + 0x12

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
F2_7	F2_6	F2_5	F2_4	F2_3	F2_2	F2_1	F2_0	F1_7	F1_6	F1_5	F1_4	F1_3	F1_2	F1_1	F1_0

F1_0-7 These 8 bits form a value which is Fan 1’s Speed in Revolutions per Second.
F2_0-7 These 8 bits form a value which is Fan 2’s Speed in Revolutions per Second.

FANS_3_4 Register Address 14 (Read Only)

Address: Base + 0x14

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
F4_7	F4_6	F4_5	F4_4	F4_3	F4_2	F4_1	F4_0	F3_7	F3_6	F3_5	F3_4	F3_3	F3_2	F3_1	F3_0

F3_0-7 These 8 bits form a value which is Fan 3’s Speed in Revolutions per Second.
F4_0-7 These 8 bits form a value which is Fan 4’s Speed in Revolutions per Second.

FANS_5_6 Register Address 16 (Read Only)

Address: Base + 0x16

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
F6_7	F6_6	F6_5	F6_4	F6_3	F6_2	F6_1	F6_0	F5_7	F5_6	F5_5	F5_4	F5_3	F5_2	F5_1	F5_0

F5_0-7 These 8 bits form a value which is Fan 5’s Speed in Revolutions per Second.
F6_0-7 These 8 bits form a value which is Fan 6’s Speed in Revolutions per Second.

FAN_CONT Register Address 18 (Read/Write Only)

Address: Base + 0x18

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
x	x	x	x	x	x	F5_A	F5_B	F4_A	F4_B	F3_A	F3_B	F2_A	F2_B	F1_A	F1_B

F1_A Writing a ‘1’ sets Fan 1 to Low Speed.
F1_B Writing a ‘1’ sets Fan 1 to High Speed.
F2_A Writing a ‘1’ sets Fan 2 to Low Speed.
F2_B Writing a ‘1’ sets Fan 2 to High Speed.
F3_A Writing a ‘1’ sets Fan 3 to Low Speed.
F3_B Writing a ‘1’ sets Fan 3 to High Speed.
F4_A Writing a ‘1’ sets Fan 4 to Low Speed.
F4_B Writing a ‘1’ sets Fan 4 to High Speed.
F5_A Writing a ‘1’ sets Fan 5 to Low Speed.
F5_B Writing a ‘1’ sets Fan 5 to High Speed.

Cooling Fan Control and Status Register. The unit has five cooling fans which can be controlled to be either off, half speed or full speed. This register has ten active bits, two for each fan. The ‘A’ bit controls the low speed option: ‘0’ = OFF; ‘1’ = ON. The ‘B’ bit controls the high speed option: ‘0’ = OFF; ‘1’ = ON. Note that when the high speed bit is set for a fan, the low speed bit becomes ‘don’t care’. Thus bit 4, F3_B controls the high speed option of fan number 3. This register is controlled by the FAN MODE bit of the CSR_CB; when this bit is zero, fan control is automatic, based on the signals from the temperature sensors, and reading this register will show how the fans are being operated. Writing to this register in this mode has no effect. When the FAN MODE bit in CSR_CB is written as ‘1’, this register is used to control the fans and will read back what is written. However, when in this mode, if any of the temperature sensors indicates that the unit is overheating, remote mode is overridden and local automatic control resumes.

TEMP_FLAG RegisterAddress 1A (Read Only)

Address: Base + 0x1A

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
x	TE_HI	TE_MD	TE_LO	TD_HI	TD_MD	TD_LO	TC_HI	TC_MD	TC_LO	TB_HI	TB_MD	TB_LO	TA_HI	TA_MD	TA_LO

TA_LO a ‘1’ indicates Temperature Sensor A is above low temp setting (e.g. 20ºC).
TA_MD a ‘1’ indicates Temperature Sensor A is above medium temp setting (e.g. 30ºC).
TA_HI a ‘1’ indicates Temperature Sensor A is above high temp setting (e.g. 40ºC).
TB_LO a ‘1’ indicates Temperature Sensor B is above low temp setting (e.g. 20ºC).
TB_MD a ‘1’ indicates Temperature Sensor B is above medium temp setting (e.g. 30ºC).
TB_HI a ‘1’ indicates Temperature Sensor B is above high temp setting (e.g. 40ºC).
TC_LO a ‘1’ indicates Temperature Sensor C is above low temp setting (e.g. 20ºC).
TC_MD a ‘1’ indicates Temperature Sensor C is above medium temp setting (e.g. 30ºC).
TC_HI a ‘1’ indicates Temperature Sensor C is above high temp setting (e.g. 40ºC).
TD_LO a ‘1’ indicates Temperature Sensor D is above low temp setting (e.g. 20ºC).
TD_MD a ‘1’ indicates Temperature Sensor D is above medium temp setting (e.g. 30ºC).
TD_HI a ‘1’ indicates Temperature Sensor D is above high temp setting (e.g. 40ºC).
TE_LO a ‘1’ indicates Temperature Sensor E is above low temp setting (e.g. 20ºC).
TE_MD a ‘1’ indicates Temperature Sensor E is above medium temp setting (e.g. 30ºC).
TE_HI a ‘1’ indicates Temperature Sensor E is above high temp setting (e.g. 40ºC).

This register shows the state of the temperature sensors in the unit. Each of the five sensors has three output flags for low, middle and high alarm states. The sensors are referred to as TA-TE and the flags as LO (low) MD (mid) and HI (high).

In automatic mode (see Fan Control register above) the LO bit of each of these sensors is used to turn the associated fan on in low speed mode. The MD bit controls the high speed mode of that fan and any of the HI bits appearing will cause all five fans to go into high speed mode.

CONFIG_2 RegisterAddress 1C (Read Only)

Address: Base + 0x1C

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
SW2/7	SW2/6	SW2/5	SW2/4	SW2/3	SW2/2	SW2/1	SW2/0	SW1/7	SW1/6	SW1/5	SW1/4	SW1/3	SW1/2	SW1/1	SW1/0

SW1/0-7 a ‘1’ indicates the relevant switch from SW1 8-way switch pack is closed / ‘ON’.
SW2/0-7 a ‘1’ indicates the relevant switch from SW2 8-way switch pack is closed / ‘ON’.

INDUSTRY PACK MEMORY AREA.

This area is 16 Mbytes wide, organised as 16-bit words. All accesses to Industry Pack resources are through this area, organised as follows:

Offset	Contents
0000000h-01FFFFEh	Memory area of Industry Pack A, 2Mbytes.
0200000h-03FFFFEh	Memory area of Industry Pack B, 2Mbytes.
0400000h-05FFFFEh	Memory area of Industry Pack C, 2Mbytes.
0600000h-07FFFFEh	Memory area of Industry Pack D, 2Mbytes.
0800000h-09FFFFEh	Memory area of Industry Pack E, 2Mbytes.
0A00000h-0BFFFFEh	Memory area of Industry Pack F, 2Mbytes.
0C00000h-0DFFFFEh	Not used (spare).
0E00000h-0E00FFEh	Access to Industry Packs A-F I/O, ID and INT areas.

This last area is sub-divided as follows:

Offset	Contents
000h-07Eh	Industry Pack A I/O registers (64 bytes).
080h-0FEh	Industry Pack A ID registers (64 bytes).
100h-17Eh	Industry Pack B I/O registers (64 bytes).
180h-1FEh	Industry Pack B ID registers (64 bytes).
200h-27Eh	Industry Pack C I/O registers (64 bytes).
280h-2FEh	Industry Pack C ID registers (64 bytes).
300h-37Eh	Industry Pack D I/O registers (64 bytes).
380h-3FEh	Industry Pack D ID registers (64 bytes).
400h-47Eh	Industry Pack E I/O registers (64 bytes).
480h-4FEh	Industry Pack E ID registers (64 bytes).
500h-57Eh	Industry Pack F I/O registers (64 bytes).
580h-5FEh	Industry Pack F ID registers (64 bytes).
600h-7FEh	Not used
800h-8FEh	Industry Pack A INT registers (128 bytes, only 2 words used).
900h-9FEh	Industry Pack B INT registers (128 bytes, only 2 words used).
A00h-AFEh	Industry Pack C INT registers (128 bytes, only 2 words used).
B00h-BFEh	Industry Pack D INT registers (128 bytes, only 2 words used).
C00h-CFEh	Industry Pack E INT registers (128 bytes, only 2 words used).
D00h-DFEh	Industry Pack F INT registers (128 bytes, only 2 words used).
E00h-FFEh	Not used.

Note: for this last set of registers, an access at the base address requests the vector for IP Interrupt 0, and at base address plus two, the vector for Interrupt 1. All other addresses are not used.

6. ID PROM

Should the 9010 IOC Blade should have an ID PROM ?????

If so here’s the probable contents ????

The ID configuration information held in the PROM is as detailed below.
The byte addresses of the ID PROM are as below:-

Base+80	ASCII ‘VI’	5649h	
Base+82	ASCII ‘TA’	5441h	
Base+84	ASCII ‘4 ‘	3420h	
Base+86	Hytec ID high byte	0080h	
Base+88	Hytec ID low word	0300h	
Base+8A	Model number	9010h	
Base+8C	Revision	2204h	(This shows PCB Issue 2 and Xilinx at issue 4)
Base+8E	Reserved	0000h	
Base+90	Driver ID	0000h	
Base+92	Driver ID	0000h	
Base+94	Flags	0002h	
Base+96	No of bytes used	001Ah	
Base+98	Not used	0000h	
Base+9A	Serial Number	xxxxd	

7. I/O Connections

Pin	Signal	Pin	Signal

8. Physical Hardware Configuration (Jumpers, Pots etc)

JUMPERS.

J1 Connects the strobe line from the Carrier Board to Industry Pack A Logic Connector pin 46.
J2 Connects the strobe line from the Carrier Board to Industry Pack B Logic Connector pin 46.
J3 Connects the strobe line from the Carrier Board to Industry Pack C Logic Connector pin 46.
J4 Connects the strobe line from the Carrier Board to Industry Pack D Logic Connector pin 46.
J5 Connects the strobe line from the Carrier Board to Industry Pack E Logic Connector pin 46.
J6 Connects the strobe line from the Carrier Board to Industry Pack F Logic Connector pin 46.
J7 Xlinx FPGA startup mode select Factory Set.
J8 Set temperature control mode – IN = lower thresholds.
J9 Xlinx FPGA startup mode select Factory Set.
J10 – J15 Select connection mode for Industry Pack A external Clock, Trigger etc signals – normally all IN.
J16 – J21 Select connection mode for Industry Pack A external Clock, Trigger etc signals – normally all IN.
J22 – J27 Select connection mode for Industry Pack A external Clock, Trigger etc signals – normally all IN.
J28 – J33 Select connection mode for Industry Pack A external Clock, Trigger etc signals – normally all IN.
J34 – J39 Select connection mode for Industry Pack A external Clock, Trigger etc signals – normally all IN.
J40 – J45 Select connection mode for Industry Pack A external Clock, Trigger etc signals – normally all IN.
J46 Connects the common strobe line to the rear panel LEMO (option).
J47 Rear panel LEMO input to PC104+ module RESET line.
J48 Wiring points for PC104+ module RESET line.
J49 Rear panel LEMO input to common strobe line.

Other Jumper Selections:

There is a set of three pins next to HP7 labelled ‘DCOK’, 11 and 22. This is factory set to DCOK-22.

SWITCHES.

SW1 8-way switch pack for ‘Configuration 1’ settings, labelled ‘8-15’.
SW2-5 Reserved for future use.
SW6 8-way switch pack for ‘Configuration 2’ settings, labelled ‘0-7’.

Note: Configuration 1 and 2 are used in software to provide mode or location settings for a system.

VARIABLE RESISTORS.

VR1 Sets the contrast for the front panel LCD display and is factory set.
VR2 Sets the threshold for the temperature sensors and is factory set.
VR3 Sets the voltage regulator for low fan speed and is factory set.

8. HYTEC TRANSITION BOARDS

Pin	Signal	Pin	Signal

APPENDIX A

Guide to Installing Linux on the 9010 IOC Blade

1. Install Linux, we usually use Scientific Linux 4, the base SL distribution of which is basically Red Hat Enterprise Linux, recompiled from source. For more information and copies of the operating system please go to https://www.scientificlinux.org/. Choose the version of Linux you want to use, it MUST have a 2.6 Kernel if you want to do any work with the Hytec IOC Blade 9010. As its PCI interface code is presently only supported with this Kernel.

2. Select Installation Language (e.g. English(English)).
3. Select Keyboard (e.g. United Kingdom).
4. Select Mouse (it will indicate what you are presently using, so normally just click on next).
5. Partition Disk (Automatic is OK in most cases). If you are installing onto some sort of restricted system, such as a Flash Disk on a PC104, you need…

- /boot – 76Mb on ext3 – Force to be Primary Partition.
- 250Mb on swap. Can be less but get it as close to this as possible.
- / - Rest of the Disk (i.e. Fill all available space).

6. (Linux Text install) Select use GRUB Boot Loader.
7. (Linux Text install) Enter Boot Loader special options, normally leaving blank is usually fine.
8. (Linux Text install) Enter Boot Loader Password, normally leaving blank is usually fine.
9. (Linux Text install) Add other operating system to Boot Loader if required.
10. (Linux Text install) Select installing Boot Loader on to Master Boot Record (MBR).
11. Network Configuration (Automatically by DHCP is OK in most cases).
12. Firewall Configuration, either have No Firewall or preferably Enable Firewall under 'Other ports' entry box list the EPICS ports i.e. '5064:tcp,5065:tcp,5064:udp,5065:udp'.
13. Add any language support required.
14. Select Time Zone.
15. Enter root password and confirm.
16. (Linux Text install) If you are installing onto some sort of restricted system, such as a Flash Disk on a PC104, you need to use 'customize software selection' and at least include (as a minimum for EPICS)…

- Development Tools.
- Kernel Development.
- Legacy Software Development.

17. Insert CD 2 to 4 as requested.
18. Graphical Interface Configuration (it will indicate what you are presently using, so normally just click on next).
19. Monitor Configuration (it will indicate what you are presently using, so normally just click on next).
20. Customize Graphics Configuration (it will indicate what you are presently using, so normally just click on next, restriction for PC104).
21. Agree to License.
22. Set Date and Time.
23. Add user.
24. Skip additional CD Installation.

APPENDIX B

Guide to Producing Stand Alone Auto Booting 9010 IOC Blade

Graphical Version

It is envisaged that the development will be done under root whereas the auto boot will be a normal user. This setup allows easy kernel code modification and offers better security to the source code, since the automatic log in will not give write access. To produce this setup…

1. Disable the "Halt on Keyboard Error" in the bios. Various PC104+ modules support different bios, but normally under the Standard Bios Heading there is 'Halt On' section. This need to be set to 'All, But Keyboard'. This will allow the 9010 to boot normally even without a keyboard fitted.
2. Scientific Linux 4, supports both the Gnome and KDE environments. KDE allows automatic log in and can restore the last session, which can be used to allow a Graphical Installation of Linux to automatically log on and run up the EPICS IOC application immediately after power up.
3. To enable automatic logon, click on K -> System Settings -> Login Screen. Select the 'Login a user automatically on first boot up' and then select a user from the 'Automatic login username' list. Select close.
4. Only a non-root user can be enabled to automatically logon, but to install the kernel you need root privileges. To give the user such privileges you need to…

   i) In the file /etc/sudoers you need to add the line…

   ```
   test    ALL=(ALL)     NOPASSWD:ALL
   ```

   This allows the user 'test' to usurp the powers of root without the need to enter a password. To edit /etc/sudoers you will need change the privileges (i.e. chmod 777 /etc/sudeors) you must ensure that the privileges are restored to the original settings (i.e. chmod 0440 /etc/sudeors).

   ii) Add the non-root user to the root group…

   Click on K -> System Settings -> Users and Groups.

   Click on the desired user to auto login from the presented list and then the 'properties' button, then click on the 'groups' tag and add (tick) the 'root' group.

5. Right Click on Desktop -> Create New -> File -> Link to Application. Change to the 'Application' Tab and use the 'Browse' button to your start up shell or application (i.e. /root/startup.sh). Click on the 'Advanced Options' button and select the 'Run in terminal' option. Change to the 'permissions' tab and change the ownership selection to root (you may need to re-login for all the options to appear). Now there exists a link to the EPICS IOC application. The startup.sh needs a slight modification the command line installing the kernel must be preceeded by 'sudo', i.e.

   ```
   sudo /root/IOCBlade9010/pci/IOC9010_load
   ```

6. To automatically restore the last session, click on K -> Control Centre -> KDE Component -> Session Manager. Select the ‘Restore previous session’ under the ‘On Login’. Select ‘Apply’.
7. Now if the link is used to run the EPICS IOC application, and KDE is logged out, it will now automatically restore.

We have not found a satisfactory “automatic log in / restore the last session” method under Gnome but for those developers who prefer to work under Gnome. You can add a command to install the 9010 kernel driver in the file /etc/profile, i.e. simply add the line ‘/root/IOCBlade9010/pci/IOC9010_load’.

Command Line Version

If the application needs to be run on Compact Flash then it is envisaged that the installation will be run under a minimized command-line Linux. To produce an automatic login under a command line only installation…

1. Disable the “Halt on Keyboard Error” in the bios. Various PC104+ modules support different bios, but normally under the Standard Bios Heading there is ‘Halt On’ section. This need to be set to ‘All, But Keyboard’. This will allow the 9010 to boot normally even without a keyboard fitted.
2. The login commands can simply be added to bash shell. To edit the bash shell (Linux default shell), you need to edit the hidden file .bash_profile (can be seen with ***ls –al*** from /root) and modify to something like below . The important additions are highlighted….

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
      . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"
EPICS_HOST_ARCH=linux-x86
EPICS_BASE=/usr/local/EPICS/base-3.14.8.2

export USERNAME BASH_ENV PATH EPICS_HOST_ARCH EPICS_BASE
```

APPENDIX C

Guide to Installing EPICS on the 9010 IOC Blade

All previously released versions of Base are now publicly available, including a nightly snapshot of the R3-14 branch of the EPICS CVS repository. To discover the size of the download files in advance, visit the Base Download area. The tar file linked below contains source code only (no binaries), and was compressed using gnuzip. baseR3.14.8.2.tar.gz

1. Gotohttp://www.aps.anl.gov/epics/base/R3-14/6.php
2. load ‘baseR3.14.8.2.tar.gz’ to /usr/local/EPICS. Make the directory if necessary(i.e. ***mkdir***).
3. Unzip it (i.e. ***gunzip baseR3.14.8.2.tar.gz***).
4. Expand it (i.e. ***tar xvf baseR3.14.8.2.tar***).
5. Before you can build or use EPICS, you must set a couple of environment variables This can be done by either simply typing the two commands (i.e.
   - ***export EPICS_HOST_ARCH=linux-x86***
   - ***export EPICS_BASE=/usr/local/EPICS/base-3.14.8.2***)

or by modifying the bash shell. To edit the bash shell (Linux default shell), you need to edit the hidden file .bash_profile (can be seen with ***ls –al*** from /root) and modify to something like below . The important additions are highlighted….

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
      . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"
EPICS_HOST_ARCH=linux-x86
EPICS_BASE=/usr/local/EPICS/base-3.14.8.2

export USERNAME BASH_ENV PATH EPICS_HOST_ARCH EPICS_BASE
```

6. Move to the build directory containing the EPICS make file (i.e ***cd /usr/local/EPICS/base-3.14.8.2***).
7. make EPICS (i.e. ***make***).

APPENDIX D

Guide to Building an EPICS Example

1. Make a directory under your root, suitably named for the example you wish to build (i.e. ***mkdir IOCBlade9010***).
2. Change to this newly created directory (i.e. ***cd IOCBlade9010***)..
3. Call a script to build an example, in the following command '-t example' is telling it to build an example type script and IOC9010 is its name (i.e. ***/usr/local/EPICS/base-3.14.8.2/bin/linux-x86/makeBaseApp.pl -t example IOC9010***)
4. Run a second script (i.e. ***/usr/local/EPICS/base-3.14.8.2/bin/linux-x86/makeBaseApp.pl -i -t example IOC9010***).
5. make (i.e. ***make***)
6. Under your example directory, the directory structure /iocBoot/'ioc + 'Example Name' will have been added. Change to this sub-directory (i.e. ***cd iocBoot/iocIOC9010***).
7. This directory contains a script (st.cmd) this files need to be made executable (i.e. ***chmod 777 st.cmd***).
8. run it (i.e. ***./st.cmd***).
9. Typing the command '**dbl**' will list the

Quick EPICS Test

1. ***cd/usr/local/EPICS/base-3.14.8.2/bin/linux-x86***
2. ***./caRepeater&***
3. ***./caget IOC:aiChannel1***

Adding Device Support to an EPICS Example

1. Copy the Device Support source file (c code file) to the source directory. The source ('src') directory is in the 'App' directory (which is named 'Example Name' + 'App') under the top example directory (i.e. in the above example it is /***IOCBlade9010/IOC9010App/src***).
2. Include the source file in the source directory's makefile. The ***makefile*** is also in the source directory and needs a line added under the '# Add locally compiled object code' heading (e.g. something like ***IOC9010_SRCS+=devHy8601.c***).
3. Modify the build to include the EPICS records added by the new device support. Again in the source directory there is the file xxxSupport.dbd, you will need to add a line for each EPICS record to be supported (e.g. of the format ***device(ao,VME_IO,devAoHy8601,"Hy8601")***).
4. Change to the top level of the example directory (i.e. ***cd / IOCBlade9010***).
5. re-make (i.e. ***make***).
6. Assuming the source has been successfully built, you will need to add your db file. The db file is usually contained in the 'db' directory under the top example directory (i.e. in the above example it is /***IOCBlade9010/db***). Copy your db file into this directory.
7. The file (st.cmd) needs to call the newly included db file. Under your example directory, in the directory structure /iocBoot/'ioc + 'Example Name' you will find the st.cmd. Change to this sub-directory (e.g. in the above example ***cd iocBoot/iocIOC9010***) and then add this db file (e.g. ***dbLoadRecords("db/Hy8601-ao.db","device=IPCard")***)
8. You should now be able to re-run it (i.e. ***./st.cmd***) and access the new variables.

APPENDIX E

Guide to Installing and Running Medm

1. Go to web site http://www.aps.anl.gov/epics/download/extensions/index.php
2. Download the following three files and copy them into base directory ***/usr/local/EPICS/base-3.14.8.2***

   medm3_0_3.tar.gz
   extensionsConfig_20040406.tar.gz
   extensionsConfigure_20040406.tar.gz

3. Extract all three files from there. This will build the extension structures.
4. Find the file RELEASE in the… ***/usr/local/EPICS/base-3.14.8.2/extensions/config*** directory and open it by using a text editor. Change the line EPICS_BASE= to the current base directory i.e.

   EPICS_BASE=***/usr/local/EPICS/base-3.14.8.2***

   And save it. Do the same change to the RELEASE file in the ***/usr/local/EPICS/base-3.14.8.2/extensions/configure*** directory and save it.

5. Go to medm subdirectory by typing:

   cd /usr/local/EPICS/base-3.14.8.2/extensions/src/medm

6. Build medm by doing a make, i.e. simply type…

   Make

   This will take possibly half an hour to build the medm depending on the speed of your machine.

7. Once the build is complete, to run medm simply change to directory….

   cd /usr/local/EPICS/base-3.14.8.2/extensions/src/medm/medm/0.linux-x86

8. Then Run it, i.e. simply type…

   ./medm

9. If you have an example to run, simply use the mouse to click on FILE -> OPEN and navigate for the example display file (*.adl is the normal extension).

10. To run it, click on the 'Execute' button.

Installing the Hytec IOC Blade Linux Kernel 2.6 Driver

1. As you are installing / running some kernel level code, you will find it it much easier to log on as root. Also this Driver is for a Linux Kernel 2.6 Version only (such as found with Scientific Linux 4.0).
2. Go to web site http://www.newwoodsolutions.co.uk
3. Download the file IOCBlade9010.tar.gzip and copy it into the root directory… ***/root***
4. Extract it in this location.
5. This is an EPICS example application and also includes a copy of the Hytec IOC Blade Linux Kernel 2.6 Driver under the Directory…   ***/root/IOCBlade9010/pci***
6. To install the driver simply type.. ***/root/IOCBlade9010/pci/IOC9010_load***

APPENDIX F

Vsystem Support

The Hytec IOC Blade 9010 Linux Kernel Driver can also be used with Vsystem and Hytec have produced example drivers for all its IP card range. They also contain simple examples (including source and build files). The development directory structure is as follows....

Development Directory Structure

In the directory **/usr/local /IOC9010/HYTECIPS/Release/** you will find these sub-directories and files…

example/demo_use.c - This has various blocks of example code used to test the Hytec IP Cards.

include/*.h - This contains the header files for individual Hytec IP Cards.

Src/HytecIPAPIs.cpp – Source Code for Hytec IP Card Classes.

In the directory **/usr/local /IOC9010/IOCGeneric/src** is the file...

IOCGeneric.cpp – Source Code for Hytec Generic IOC Access Classes.

System Setup

The following things MUST be done !

1. To be able to build and run your example code “**/usr/local/lib”** MUST be on the path, the easiest thing to do is to add it to your shell command, for instance if you use bash, simply add the line “export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib” to .bash_profile file.

2. All the Vsystem support is via IOC Blade 9010 Kernel Driver, so this needs to be loaded, the easiest thing is to simply type “***/root/IOCBlade9010/pci/IOC9010_load***” from inside your development shell.

Build Sequence

1. Build the Generic Functions (i.e**. ./usr/local /IOC9010/IOCGeneric/src/makescript**).

2. Build the Specific Library Functions (i.e**. ./usr/local /IOC9010/HYTECIPS/Release/src/makescript**).

3. Build the Test Example (i.e**. ./usr/local /IOC9010/HYTECIPS/Release/example/make**).

Distribution and Installation

The previous discussion aside usually a Hytec Vsystem distribution is usually just the Library, Header and Example files. A simple pack script generate a tar of tars of these files, which is distributed. The user simply has to unpack them to create / overwrite the directories. Implement the system setup described above and then the user is ready to edit, build and try example code.

Usually our Vsystem directories are set up under /usr/local/IOC9010 and are as follows....

```
1. Under usr/local/IOC9010/HYTECIPS…
  Release
    example
      demo use files
    include
      api function header files
```

Directly under /usr/local/IOC9010 is the HYTECIPS directory which has the example directory with some demonstration code and the include directory which contains IP Cards header files.

```
2. Under usr/local/IOC9010/IOCGeneric…
  example
     more demo use files
  include
     driver c source and header files
```

Directly under /usr/local/IOC9010 is the IOCGeneric directory which has the include directory for the Generic (i.e. can be used on 9010 and 5331/3331) access functions.

3. usr/local/lib…

When the Vsystem library is built it generates two library files which have to be transferred to the following usual library file location (i.e. usr/local/lib). The library files are libHytecIPAPIs.so.1.O (the IP Cards API) and libIOCGeneric.so.1.0 (the generic kernel calls).

APPENDIX G

Writing Your Own IP Card Drivers

The Hytec IOC Blade 9010 Linux Kernel Driver is contained in the Directory ***/root/IOCBlade9010/pci***. This Directory includes the driver itself, load / unload scripts and a header file to allow you to access the IP Cards and the actual IOC Blade 9010 Registers. It also contains a simple example (including source and makefiles). The files in this directory are....

9010LinuxDriver.c / 9010LinuxDriver.h

Source Code of the Kernel Driver. Do NOT Edit ! The 9010LinuxDriver.ko IS built from these files, only edit these if you wish to change the kernel level operation and always backup the originals.

IOC9010_load

A script to load / install the Kernel Driver (i.e. ./IOC9010_load). The 9010LinuxDriver.ko MUST have been built before this script is run. An IOC Blade 9010 is always shipped with an operational copy of 9010LinuxDriver.ko.

IOC9010_unload

A script to unload / remove the Kernel Driver (i.e. ./IOC9010_unload). The 9010LinuxDriver.ko MUST have been built before this script is run. An IOC Blade 9010 is always shipped with an operational copy of 9010LinuxDriver.ko.

main.c

Example Source Code for the User to Edit ! This is normally matched to the system configuration brought. Once built it can be run.

Makemain

Simple script to build the Source Code main.c to use (i.e. ./makemain) to make the executable main. To run type ./makemain.

makefile

Simple makefile to build the Kernel Driver object (9010LinuxDriver.ko) from the Source Code (9010LinuxDriver.c and 9010LinuxDriver.h) for the course attendee to use (i.e. make) to make the Kernel Driver.

9010LinuxDriver.ko MUST be built before the scripts (IOC9010_unload and IOC9010_load) are run.

The driver provides functions that use stream like operations, so you for instance read the ID PROM of the IP Card in Site A…

```
/* Open the Stream */
IOCHandle = open("/dev/IOC9010",0);
if (IOCHandle = = -1) printf("9010: Error Opening Device !\n");

/* Set up the Data Structure */
ioctl_buf.lAddress = IP_A_ID_BASE_ADDR + ID_MODEL_NUMBER;
ioctl_buf.lLength  = 1;
ioctl_buf.sData  = (unsigned long)(&data);

/* Read IP Card Type from ID PROM */
*val = ioctl(IOCHandle, OP_GENERAL_READ, &ioctl_buf);

printf("IP Slot %c = %4X\n", 'A', *val);

/* Close the Stream */
close(IOCHandle);
```

or reading the first 5 registers on the 8505 IP Card in Slot C…

```
IOCTL_BUF   ioctl_buf;
unsigned short   data = value;
unsigned short   readbuffer[5];
int      IOCHandle;

IOCHandle = open("/dev/IOC9010",0);
if (IOCHandle == -1) printf("8505: Error Opening Device !\n");

/* 8505 Basic Digital Output Setup */
ioctl_buf.lAddress = IP_C_IO_BASE_ADDR;
ioctl_buf.lLength  = 5;
ioctl_buf.sData    = (unsigned long)(readbuffer);
ioctl(IOCHandle, OP_GENERAL_READ_BLOCK, &ioctl_buf);

close(IOCHandle);
```

Developing and Running Your Own IP Card Drivers

Assuming the most basic of systems and Linux installations the beginner can develop even inside a terminal window…

1. Log in ***root*** user (default password for IOC Blade 9010 is ***password***). Root privileges are required for kernel driver installation.
2. Open a terminal.
3. Change to the Directory type ***/root/IOCBlade9010/pci***.
4. Type ./***IOC9010_load*** to install the Hytec Kernel Driver.
5. Type ***vi main.c*** – to edit the source code. Obviously you can use any text editor, I am sure your version of Linux will provide a useful graphical editor. Obviously a knowledge of C is required.
6. Type ***./makemain -*** to make the executable main.
7. To run type ***./main.***

Type ./***IOC9010_unload*** to remove the Hytec Kernel Driver when you have finished.

APPENDIX H

EPICS Variables Pre-Installed on IOC Blade 9010 Demo

EPICS Variable Name	Description	Range
Hy9010:ai-Fan1-PSU	Speed in rpm of Fan 1 – Rear Fan nearest the PSU	Limits 0-10,000 rpm. Slow – 5,000 High - 7,000
Hy9010:ai-Fan2-IPCards	Speed in rpm of Fan 2 – Rear Fan near the IP Cards	Limits 0-10,000 rpm. Slow – 5,000 High - 7,000
Hy9010:ai-Fan3-Invertors	Speed in rpm of Fan 3 – Rear Fan near Invertors	Limits 0-10,000 rpm. Slow – 5,000 High - 7,000
Hy9010:ai-Fan4-Trans	Speed in rpm of Fan 4 – Front Fan near Transition	Limits 0-10,000 rpm. Slow – 5,000 High - 7,000
Hy9010:ai-Fan5-PC104+	Speed in rpm of Fan 5 – Front Fan for PC104+	Limits 0-10,000 rpm. Slow – 5,000 High - 7,000
Hy9010:ai-Fan6-PMC	Speed in rpm of Fan 6 – Fan under PMC	NOT FITTED
Hy9010:ai-temp1-IP	Temperature of sensor near the IP Cards	Limits 0-40 ºC in 10 ºC steps.
Hy9010:ai-temp2-PSU	Temperature of sensor near the PSU	Limits 0-40 ºC in 10 ºC steps.
Hy9010:ai-temp3-PC104+	Temperature of sensor under the PC104+	Limits 0-40 ºC in 10 ºC steps.
Hy9010:ai-temp4-PMC	Temperature of sensor near the PMC	Limits 0-40 ºC in 10 ºC steps.
Hy9010:ai-temp5-Trans	Temperature of sensor near the transition cards	Limits 0-40 ºC in 10 ºC steps.
Hy9010:ai-IP-Card-A	The ID from the ID PROM in Site A	Hytec Electronics Ltd IP Cards are encoded in Hex.
Hy9010:ai-IP-Card-B	The ID from the ID PROM in Site B	Hytec Electronics Ltd IP Cards are encoded in Hex.
Hy9010:ai-IP-Card-C	The ID from the ID PROM in Site C	Hytec Electronics Ltd IP Cards are encoded in Hex.
Hy9010:ai-IP-Card-D	The ID from the ID PROM in Site D	Hytec Electronics Ltd IP Cards are encoded in Hex.
Hy9010:ai-IP-Card-E	The ID from the ID PROM in Site E	Hytec Electronics Ltd IP Cards are encoded in Hex.
Hy9010:ai-IP-Card-F	The ID from the ID PROM in Site F	Hytec Electronics Ltd IP Cards are encoded in Hex.

EPICS Variable Name	Description	Range
IPCard:mbboDirect	An 8505 Card in Slot C outputs is driven via this.	0-65535
IPCard:ao00	An 8402 Card in Slot B output 0 is driven via this.	-10V to +10V
IPCard:ao01	An 8402 Card in Slot B output 1 is driven via this.	-10V to +10V
IPCard:ao02	An 8402 Card in Slot B output 2 is driven via this.	-10V to +10V
IPCard:ao03	An 8402 Card in Slot B output 3 is driven via this.	-10V to +10V
IPCard:ao04	An 8402 Card in Slot B output 4 is driven via this.	-10V to +10V
IPCard:ao05	An 8402 Card in Slot B output 5 is driven via this.	-10V to +10V
IPCard:ao06	An 8402 Card in Slot B output 6 is driven via this.	-10V to +10V
IPCard:ao07	An 8402 Card in Slot B output 7 is driven via this.	-10V to +10V
IPCard:ao08	An 8402 Card in Slot B output 8 is driven via this.	-10V to +10V
IPCard:ao09	An 8402 Card in Slot B output 9 is driven via this.	-10V to +10V
IPCard:ao10	An 8402 Card in Slot B output 10 is driven via this.	-10V to +10V
IPCard:ao11	An 8402 Card in Slot B output 11 is driven via this.	-10V to +10V
IPCard:ao12	An 8402 Card in Slot B output 12 is driven via this.	-10V to +10V
IPCard:ao13	An 8402 Card in Slot B output 13 is driven via this.	-10V to +10V
IPCard:ao14	An 8402 Card in Slot B output 14 is driven via this.	-10V to +10V
IPCard:ao15	An 8402 Card in Slot B output 15 is driven via this.	-10V to +10V

APPENDIX I

IOC Blade 9010 API Commands

IOC9010Open

Syntax

```
FUNCTION IOC9010Open(void) : int;
```

Parameter

None.

Result

If successful the return value will be positive and will be the Identifier.
A return value of -1 indicates the open has failed.

Description

This function is called to obtain an identifier for all future calls to the API Functions.
This function MUST be called before any other.

Example

```
int iIOC9010Handle;
iIOC9010Handle = IOC9010Open();
```

IOC9010Close

Syntax

```
FUNCTION IOC9010Close(int i9010Handler) : void;
```

Parameter

`i9010Handler` : The handler to be closed.

Result

0 - is returned if the API is successfully closed.
-1 - indicates the close has failed.

Description

This function is used to close the API, it is last action of the program before closing the application.

Example

```
IOC9010Close(i9010Handle);
```

IOC9010GetConfig

Syntax

```
FUNCTION IOC9010GetConfig(struct *PresentConfig) : int;
```

Parameter

`*PresentConfig` - A pointer to a structure to write the present configuration into..

Result

0 - is returned if the structure is successfully written.
-1 - indicates the structure update has failed.

Description

This function populates a passed structure with details of the IOC Blade 9010’s present configuration.

`*PresentConfig` - The structure is as follows…

Example

```
IOC9010GetConfig();
```

IOC9010SetConfig

Syntax

```
FUNCTION IOC9010SetConfig( ) : int;
```

Parameter

Result

0 - is returned if the structure is successfully written.
-1 - indicates the structure update has failed.

Description

This function is used to update IOC Blade 9010's present configuration.

Example

IOC9010FPC

Syntax

```
FUNCTION IOC9010FPC(int iMMIOnOFF) : int;
```

Parameter

`iMMIOnOFF` - '1' Enable Automatic Front Panel Man-Machine Interface.
- '0' Disable Automatic Front Panel Man-Machine Interface.

Result

If successful the return value is 0, the automatic front panel control man-machine interface (MMI) is disabled.
A return value of -1 indicates the action has failed.

Description

This function is used to disable or enable the automatic front panel control man-machine interface (MMI) of the IOC Blade 9010. Once the MMI is disabled the display can be updated via the API.

Example

```
IOC9010FPC(0); /* Disable MMI, so can now over write Display */

IOC9010FPC(1); /* Enable MMI */
```

IOC9010LCDWrite

Syntax

```
FUNCTION IOC9010LCDWrite(U16 position, U8 *string) : int;
```

Parameter

`position` – The start position of the string on the LCD. Values of…

0 – 39   is the number characters in from the left on the top line.
64 - 103 is the number characters in from the left on the bottom line.

To ease use the header file includes the following defines which can be OR ed or added to produce the desired positioning. There is also the define API_LCD_CENTRE_ON which will automatically centre the passed string.

API_LCD_LINE_1	(0)
API_LCD_LINE_2	(64)
API_LCD_CENTRE_ON	(0x8000)

`*string –` a pointer to a null-terminated ASCII string (up to 40 characters in length) containing the string to be written to the display.

Result

0 - is returned if the display is successfully written.
-1 - indicates the display update has failed.

Description

This function updates the front panel LCD with the passed strings at the requested position.
This function is can ONLY be used when the automatic front panel man-machine interface is disabled.

Example

```
Char sTopString[] = “Top Line”;
Char sBottomString[] = “Bottom Line”;

/* Write String on Top Line Automatically Centred */
IOC9010LCDWrite(API_LCD_LINE_1 | API_LCD_CENTRE_ON, sTopString);

/* Write String on Bottom Line 10 Character in from Left */
IOC9010LCDWrite(API_LCD_LINE_2 + 10, sBottomString);
```

IOC9010LCDRead

Syntax

```
FUNCTION IOC9010LCDRead(U8 *LCDString) : int;
```

Parameter

`*LCDString` - a pointer to a null-terminated string, 80 characters in length, containing the ASCII characters presently on the display.

Result

0 - is returned if the display is successfully read.
-1 - indicates the reading of the display has failed.

Description

This function is used to obtain the present message on the front panel display. Characters 0-39 is the top line and characters 40-79 is the bottom line.

Example

```
char cLCDString[80];
if (IOC9010LCDRead(cLCDString) == 0)
{
   printf(“The LCD is presently displaying %s\n”, cLCDString);
}
```

IOC9010LCDClear

Syntax

```
FUNCTION IOC9010LCDClear(void) : int;
```

Parameter

None.

Result

0 - is returned if the display is successfully cleared.
-1 - indicates the display update has failed.

Description

This function clears the front panel LCD i.e. sets all characters to space (ASCII 0x20).
This function is can ONLY be used when the automatic front panel man-machine interface is disabled.

Example

```
IOC9010LCDClear();
```

IOC9010ConnectFunctToKey

Syntax

```
FUNCTION IOC9010ConnectFunctToKey(void *function, int switch) :
int;
```

Parameter

`*function` – A pointer to the function to be called.

`switch` – The switch to monitor.

To ease use the header file include defines for all switches.

Result

0 - is returned if the function is successfully connected to the requested switch.
-1 - indicates the action failed.

Description

This function is used to connect the passed function is to the requested switch.

The function will automatically perform all the hardware debounce etc.

This function is can ONLY be used when the automatic front panel man-machine interface is disabled.

Example

```
void vOKPressed(void)
{
   printf("OK Key Pressed\n");
}


IOC9010ConnectFunctToKey(&vOKPressed(void), API_SWITCH_OK)
```

IOC9010ConnectFunctToInt

Syntax

```
FUNCTION IOC9010ConnectFunctToInt(void *function, int Int) : int;
```

Parameter

`*function` – A pointer to the function to be called.

`Int` – The switch to monitor.

To ease use the header file include defines for all IP Cards Interrupts.

Result

0 - is returned if the function is successfully connected to the requested Interrupt.
-1 - indicates the action failed.

Description

This function is used to the connect passed function to the requested interrupt source.

Example

```
void vIPSlotAISR(void)
{
   printf(“IP Slot A Interrupt Detected\n”);
}


IOC9010ConnectFunctToInt(&vIPSlotAISR(void), API_INT_SLOT_A_0)
```

IOC9010CarrierRead

Syntax

```
FUNCTION IOC9010CarrierRead(U16 add, U16 len, U16 *data) : int;
```

Parameter

`Add` – The position of the first register to read.

To ease use the header file include defines of Register Addresses.

`Len` – The number of registers to read.

`*data` – A pointer to an array or U16 (for single register access) to store the registers contents.

Result

0 - is returned if the carrier board register is successfully read.
-1 - indicates the carrier board register read has failed.

Description

This function is used to read the Control and Configurations Registers on the IOC Blade 9010 Carrier Board itself.

Example

```
U16 U16CarrierRegs[9];

/* Read all the 9010 Carrier Board Registers */
IOC9010CarrierRead(API_REG_CSR, 9, U16CarrierRegs);
```

IOC9010CarrierWrite

Syntax

```
FUNCTION IOC9010CarrierWrite(U16 add, U16 len, U16 *data) : int;
```

Parameter

`Add` – The position of the first register to write.

To ease use the header file include defines of Register Addresses.

`Len` – The number of registers to write.

`*data` – A pointer to an array or U16 (for single register access) to store the register contents.

Result

0 - is returned if the carrier board register is successfully written.
-1 - indicates the carrier board register write has failed.

Description

This function is used to write the Control and Configurations Registers on the IOC Blade 9010 Carrier Board itself.

Example

```
U16 U16CarrierReg = 0x0001;

/* Write the IOC9010 Carrier Board CSR Register */
IOC9010CarrierWrite(API_REG_CSR, 1, &U16CarrierReg);
```

IOC9010IPRead

Syntax

```
FUNCTION IOC9010IPRead(U16 add, U16 len, U16 *data) : int;
```

Parameter

`Add` – The position of the first IP register / memory to read.

To ease use the header file include defines of Addresses.

`Len` – The number to read.

`*data` – A pointer to an array or U16 (for single register access) to store the read contents.

Result

0 - is returned if the read is successful.
-1 - indicates the read has failed.

Description

This function is to read data from any of the IP Cards contents.

Example

```
U16 U16IPRegs[5];
U16 U16_IP_ID;
U16 U16_ADC;

/* Read the I/D Register from IP Card in Site A */
IOC9010IPRead( API_ IP_A_ID_BASE_ADDR + API_ID_MODEL_NUMBER,
               1,
               & U16_IP_ID);

/* If its an Hytec Electronics Ltd 8401 8 X 16 Bit ADC */
if (U16_IP_ID == 0x8401)
{
    /* Read the first 5 I/O Registers from IP Card in Site A */
    IOC9010IPRead(API_ IP_A_IO_BASE_ADDR, 5, U16IPRegs);

    /* Read All the ADC Values from IP Card in Site A */
    IOC9010IPRead(API_ IP_A_IO_BASE_ADDR + 16, 8, U16_ADC);
}
```

IOC9010IPWrite

Syntax

```
FUNCTION IOC9010IPWrite(U16 add, U16 len, U16 *data) : int;
```

Parameter

`Add` – The position of the first register to write.

To ease use the header file include defines of Addresses.

`Len` – The number to write.

`*data -` – A pointer to an array or U16 (for single register access) to store the register contents.

Result

0 - is returned if the display is successfully cleared.
-1 - indicates the display update has failed.

Description

This function is to write data to any of the IP Cards contents.

Example

```
U16 U16IPReg = 0x0001;

U16 U16SetUp = {0x0C01,0x0078,0x0000,0x0000};

/* Write the 3rd I/O Registers on the IP Card in Site F */
IOC9010IPWrite(API_ IP_F_IO_BASE_ADDR + 2, 1, &U16IPReg);

/* Set up the Hytec Electronics Ltd 8505 in Site B */
IOC9010IPWrite(API_ IP_B_IO_BASE_ADDR + 1, 4, U16SetUp);
```

APPENDIX J

IOC Blade 9010 HTML Web Interface

Introduction

The HTML support is NOT intended for normal everyday control use (perhaps with the exception of the Remote Reset / Reboot) but is intended only for the following activities…

1. Out of the Box Testing – Since the HTML Webpage can be accessed by any internet browser, it is envisaged that one of its primary uses would be "Out of the Box" testing. The IOC Blade 9010 can be taken out of the box, be fitted with the desired selection of IP Cards. If the 9010 is then connected to a network (or straight to a PC with an Ethernet Crossover Lead), any internet browser can access the Web interface and confirm the unit settings and the IP Cards fitted.

2. Remote Testing – The HTML Webpage allows IP Card Registers to be overwritten as well as read. With Hytec Electronics Ltd IP Cards this will allow the HTML interface to directly modify outputs or read inputs. This effectively gives you an instant, network controlled, with flexible outputs / inputs piece of test equipment. Which you put anywhere on your network and test and monitor any system signals without the need to develop any software.

3. Configuration – The first tab of the Webpage will allow elements such as the 9010's IP Address, Subnet Mask etc allowing the 9010's general set up to be done relatively easily and quickly.

4. Remote Reset / Reboot – There is a button on the first tab of the Webpage, which will run a script which will shutdown and restart the IOC Blade 9010. This may allow a remote recovery if an EPICS or OPC application has crashed.

Using the HTML Webpage Interface

To access the Webpage simply start your favourite internet / web browser and where you normally enter the address, enter the IP Address of IOC Blade 9010 in dot-decimal notation (e.g. 172.23.81.192). As long as there is no problem with sub-net, firewall or some other access problem, you should immediately see the IOC Blade 9010 Webpage.

The webpage is made up of 7 tabbed sub pages, one for each IP Card and the front page which is specific to the 9010.

Letting the Webpage Know the IP Card “Personality”

The webpage reads the list of registers to show for each IP Card from the configuration file, ***ip_types.db***, which is a simple comma delimited text file. Each IP Card’s “personality” is a single line entry in the file and is of the format…

Vendor ID, Product ID, No Reg, Reg Name 1 … Reg Name N , No Mem, Name Mem 1…Name Mem N

Vender ID The Manufactor / Vendor ID of the IP Card as in the ID PROM.
Product ID The Product ID of the IP Card as in the ID PROM.
No Reg The Number of registers from start of the I/O Space to be named in the following data.
Reg Name 1 The Name to be displayed on the Web page of the first register in the I/O Space.
…
Reg Name N The Name to be displayed on the Web page of the Nth register in the I/O Space.
No Mem The Number of registers from the start of Memory Space to be named in following data.
Mem Name 1 The Name to be displayed on the Web page of the first register in the Memory Space.
…
Mem Name N The Name to be displayed on the Web page of the Nth register in the Memory Space.

Notes:-

3. Vendor and Product ID Combination used to uniquely identify the IP Card.
4. Reg Name 1…N is a contiguous list of names for contiguous expecting to name a contig

Typical Examples of Hytec Electronics Ltd’s IP Cards are shown below…

```
0x00800300, 0x8505, 6,LKC, CSR, IMR , DBR, PSR, PPR
0x00800300, 0x8513, 4,CSR, ARM, IRQ STATUS, IRQ MASK, 8, COUNT 0 LSB, ………
```

The IOC blade 9010 will be supplied with a default ***ip_types.db***, which will support the Hytec Electronics Ltd range of IP Cards presently available. If you wish to support other manufacturers, you can simply add additional entries / lines to the file.

APPENDIX K

IOC Blade 9010 TCP/IP Interface

Introduction

This document defines the proprietary TCP/IP commands supported by Hytec Electronics Ltd equipment. The TCP/IP will support a single socket.

Command Set

All the TCP/IP Instructions are made up of a Command and Response, both have the same identifier, but the Response has the Most Significant Bit set (0x80).

Status

This command is used to obtain a list of IP (Industry Pack) cards installed in an IOC. This is assuming that the IP Cards fitted are VITA4 compliant and have their identification stored in an ID PROM.

Status Command

Byte	Byte Description	Range
1	Command Identifier	0x01
2	Command Length (bytes to follow)	0x00

Figure 2: Format of Status Command

Status Command Response

Byte	Byte Description	Range
1	Command Identifier	0x81
2	Command Length (bytes to follow)	IP Cards x 4
3*	Slot Number	1-255
4*	IP Card 0=A, 1=B, 2=C, 3=D, 4=E and 5=F	0-5
5*	Top 2 Nibbles of IP Card encoded as BCD	0x00-0xFF
6*	Bottom 2 Nibbles of IP Card encoded as BCD	0x00-0xFF

Note * These bytes are repeated for each and every IP Card Slot.

Figure 3: Format of Status Command Response

Read Command

This command is used to read data from any IP (Industry Pack) card installed in an IOC. The format of the data in the response is dependant on the IP card type.

Read Command

Byte	Byte Description	Range
1	Command Identifier	0x03
2	Command Length (bytes to follow)	2
3	Slot Number	1-255
4	IP Card 0=A, 1=B, 2=C, 3=D, 4=E and 5=F	0-5

Figure 4: Format of Read Command

Read Command Response

Byte	Byte Description	Range
1	Command Identifier	0x83
2	Command Length (bytes to follow)	Length of Response + 2
3	Slot Number	1-255
4	IP Card 0=A, 1=B, 2=C, 3=D, 4=E and 5=F	0-5
5*	Response	0x00-0xFF

Note * These bytes are repeated for each and every relevant value on IP Card.

Figure 5: Format of Read Command Response

Write Command

This command is used to write data to any IP (Industry Pack) card installed in an IOC. The format of the data in the response is dependant on the IP card type. The response is simply an acknowledgement of receipt of the command.

Write Command

Byte	Byte Description	Range
1	Command Identifier	0x04
2	Command Length (bytes to follow)	Length of Configure String + 2
3	Slot Number	1-255
4	IP Card 0=A, 1=B, 2=C, 3=D, 4=E and 5=F	0-5
5*	Control Data	0x00-0xFF

Note * These bytes are repeated for each and every relevant value on IP Card.

Figure 6: Format of Write Command

Write Command Response

Byte	Byte Description	Range
1	Command Identifier	0x84
2	Command Length (bytes to follow)	0

Figure 7: Format of Write Command Response

APPENDIX L

Producing Disk Images for the IOC Blade 9010

Introduction

The recommended / supported Disk Image Making Software is the shareware G4L (Ghost for Linux)

This can be obtained from http://sourceforge.net/projects/g4l , there are various options we normally prefer to download a CD Image (e.g. g4l-v0.22.iso) and produce a bootable CD. The default bios setting on the IOC Blade 9010 is to try to boot from USB CD-ROM first, so normally simply attaching one will allow you to boot straight into G4L. Obviously if your PC104+ bios is set to something different you may need to alter you boot sequence.

When the IOC Blade 9010 boots off the G4L CD-ROM, do not panic its takes quite a long time as it has to load the image etc, if it boots the following messages appear…

```
Loading bzImage6…………………………………………………………………………………………
Loading ramdisk.gz………………………………………………………………………………………..
```

At the command prompt type g4l <CR>.

Select Raw Mode.

We store our disk image on the network so you need to select “Network Use”.

A – Select eth0
B – Config Device
C – Config with DHCP
D – Config FTP (172.23.81.3)
E – Config useridpass (username : password)
F – Config filename (MOPS-PM600.gzip)
G – Compression (.gzip)

H – Backup (hda)
I – Restore (hda)

APPENDIX L

Idiots Guide to RTEMS

Introduction

RTEMS (Real-Time Executive for Multiprocessor Systems) is designed for real-time, embedded systems. Due to its license free, open source nature it has rapidly become popular in markets previously dominated by VxWorks, particularly the EPICS Community. It has been ported to various target processor architectures, but we use it with i386 Family processors and have specifically produced a PC104+ processor for the IOC Blade 9010 to support RTEMS.   This PC104+ is a variant of the MSM800 family with hardware changes

It is a free open source operating system and was originally designed for missile systems hence initially the acronym stood for *Real-Time Executive for Missile Systems*, then became *Real-Time Executive for Military Systems* before changing to its current meaning.

Its development began in the late 1980s with early versions of RTEMS available via ftp as early as 1993. OAR Corporation is currently managing the RTEMS project in cooperation with a Steering Committee which includes user representatives.

There should be at least 300 Mbytes of space available on the drive where these directories are located. I used /usr/local/rtems/rtems4.7 as the installation target directory. The location of the RTEMS source is not critical. This document assumes that the root of the RTEMs source tree is /usr/local/rtems/source.

Create the RTEMS source and installation directories

There should be at least 300 Mbytes of space available on the drive where these directories are located. I used /usr/local/rtems/rtems4.7 as the installation target directory. The location of the RTEMS source is not critical. This document assumes that the root of the RTEMs source tree is /usr/local/rtems/source.

Create the directories where the source will be placed and the results of the build installed:

/usr/local/rtems/source
/usr/local/rtems/source/tools
/usr/local/rtems/rtems4.7

Add the directory containing the tools to your shell search path

The following sections assume that the directory into which you will install the cross-development tools (/usr/local/rtems/rtems4.7/is on your shell search path. For shells like sh, bash, zsh and ksh you can to this with

```
PATH="$PATH:/usr/local/rtems/rtems4.7/bin"
```

For shells like csh and tcsh you can

```
set path = ( $path /usr/local/rtems/rtems4.7/bin )
```

Get and build the development tools

RTEMS uses the GNU toolchain to build the executive and libraries. Information about the GNU tools can be found on the GNU home page. If you’re feeling brave you can skip the following sections and turn loose the script included in appendixA. In either case, if you’re building on Solaris you’ll need to ensure that you have GNU make (gmake) installed on your system and also set a couple of environment variables for things to build properly:

```
MAKE=gmake
INTLLIBS=-lintl
```

The script attempts to download, unpack, configure, build and install the GNU cross-development tools and libraries
for one or more target architectures. To use the script, set the ARCHS environment variable to the architectures you
wish to support, then

```
sh getAndBuildTools.sh
```

Set the MAKE environment variable to the name of whatever make program you need for your system.

Download the tool source files

The source for the GNU tools should be obtained from the On-line Applications Research (OAR) FTP server since
that server provides any RTEMS-specific patches that may have to be applied before the tools can be built.
The files in the OAR FTP server directory ftp://www.rtems.com/pub/rtems/SOURCES should be downloaded to the
RTEMS/tools directory created above. The files can be downloaded using a web browser or a command-line program
such as curl or wget. (note that the command examples have been split to help them fit on the page):

```
curl --remote-name
ftp://www.rtems.com/pub/rtems/SOURCES/binutils-2.16.1.tar.bz2
curl --remote-name
ftp://www.rtems.com/pub/rtems/SOURCES/gcc-4.1.0.tar.bz2
curl --remote-name
ftp://www.rtems.com/pub/rtems/SOURCES/newlib-1.14.0.tar.gz
curl --remote-name
ftp://www.rtems.com/pub/rtems/SOURCES/binutils-2.16.1-rtems-20050816.diff
```

or

```
wget --passive-ftp --no-directories --retr-symlinks
ftp://www.rtems.com/pub/rtems/SOURCES/binutils-2.16.1.tar.bz2
```

```
wget --passive-ftp --no-directories --retr-symlinks
ftp://www.rtems.com/pub/rtems/SOURCES/gcc-4.1.0.tar.bz2
...
```

Depending on the type of firewall between your machine and the OAR FTP server you may need to remove the
--passive-ftp option from the wget commands.

Unpack the source archives:

The following commands will extract the GNU tool sources from the downloaded tar archive files.

```
bzcat binutils-2.16.1.tar.bz2 | tar xf -
bzcat gcc-4.1.0.tar.bz2 | tar xf -
zcat newlib-1.14.0.tar.gz | tar xf -
```

To build the newlib libraries needed by RTEMS you must make a symbolic link to the newlib source directory from
the gcc source directory.

```
cd gcc-4.1.0
rm -rf newlib
ln -s ../newlib-1.14.0/newlib newlib
cd ..
```

Apply any RTEMS-specific patches

If any patch files (those with a .diff suffix) were downloaded from the OAR FTP server the patches in those files
must be applied before the tools can be compiled.
Here is how the patches can be applied to the binutils sources:

```
cd binutils-2.16.1
patch -p1 <../binutils-2.16.1-rtems-20050816.diff
cd ..
```

2.3.4 Configure, build and install the ‘binutils’:

The commands in this section must be repeated for each desired target architecture. The examples shown build the
tools for Motorola Power PC targets.

1. Create a directory in which the tools will be built and change to that directory.

```
rm -rf build
mkdir build
cd build
```

2. Configure the tools.

```
../binutils-2.16.1/configure --target=powerpc-rtems4.7 \
--prefix=/usr/local/rtems/rtems4.7
```

You should replace the ‘powerpc’ with the name of the architecture for which you’re building the tools. Common
alternatives are ‘m68k’ and ‘i386’ for the Motorola M68k and Intel x86 family of processors, respectively.

3. Build and install the tools.

```
make -w all install
```

In this and all subsequent cases the use of a GNU make program is required. On some hosts you’ll have to use
gmake instead of make.

4. Return to the directory containing the tool and library sources.

```
cd ..
```

2.3.5 Configure, build and install the cross-compiler and libraries

1. Create a directory in which the tools will be built and change to that directory.

```
rm -rf build
mkdir build
cd build
```

Configure the compiler and libraries.

```
../gcc-4.1.0/configure --target=powerpc-rtems4.7 \
--prefix=/usr/local/rtems/rtems4.7 \
--with-gnu-as --with-gnu-ld --with-newlib --verbose \
--with-system-zlib --disable-nls \
--enable-version-specific-runtime-libs \
--enable-threads=rtems \
--enable-languages=c,c++
```

You should again replace the ’powerpc’ with the name of the architecture for which you’re building the crosscompiler
and libraries.

3. Build and install the cross-compiler and libraries by.

```
make -w all install
```

4. Return to the directory containing the tool and library sources.

```
cd ..
```

2.4 Get, build and install RTEMS

2.4.1 Download the RTEMS source from the OAR web server.

At the time of writing no release of RTEMS was capable of supporting EPICS so you had to, and may still have to,
work with a developer’s snapshot. The latest snapshot is available in
http://www.rtems.com/ftp/pub/rtems/4.6.99.2/rtems-4.6.99.2.tar.bz2
The compressed tar archive in this directory can be downloaded using a web browser or a command-line program
such as curl or wget:

```
curl --remote-name
’http://www.rtems.com/ftp/pub/rtems/4.6.99.2/rtems-4.6.99.2.tar.bz2’
```

or

```
wget --passive-ftp --no-directories --retr-symlinks
’http://www.rtems.com/ftp/pub/rtems/4.6.99.2/rtems-4.6.99.2.tar.bz2’
```

Depending on the type of firewall between your machine and the OAR FTP server you may need to remove the
--passive-ftp option from the wget command.
When you are done you should have the compressed archive with a name something like
rtems-4.7.tar.bz2

2.4.2 Unpack the RTEMS sources

Change to your RTEMS source directory and unpack the RTEMS sources by:

```
bzcat rtems-4.7.tar.bz2 | tar xf -
```

This will create the directory rtems-4.7 and unpack all the RTEMS source into that directory.

2.4.3 Make changes to the RTEMS source to reflect your local conditions.

Some of the board-support-packages distributed with RTEMS may require modifications to match the hardware in use at your site. The following sections describe changes commonly made to two of these board-support-packages.

PC-x86

A change I like to make to the RTEMS pc386 source is to increase the number of lines on the console display from 25
to 50 since I find that the output from some EPICS commands scrolls off the display when only 25 lines are present.
To make this change, add the ‘#define’ line shown below

```
rtems-4.7/c/src/lib/libbsp/i386/pc386/start/start16.S
+------------------------------------------------------*/
#include <bspopts.h>
#define RTEMS_VIDEO_80x50
/*--------------------------------------------------------+ | Constants
```

Another change I make is to automatically fall back to using COM2: as a serial-line console (9600-8N1) if no video adapter is present. This allows the pc386 BSP to be used on conventional PCs with video adapters as well as with embedded PCs (PC-104) which have no video adapters. To make this change, add the ‘#define’ line shown below

6

```
rtems-4.7/c/src/lib/libbsp/i386/pc386/console/console.c
*/
rtems_termios_initialize ();
#define RTEMS_RUNTIME_CONSOLE_SELECT
#ifdef RTEMS_RUNTIME_CONSOLE_SELECT
/*
* If no video card, fall back to serial port console
```

2.4.4 Build and install RTEMS

1. It is best to start with a clean slate. Make very sure you’re in the right directory before running the rm command!

```
cd /usr/local/rtems/source
rm -rf build
mkdir build
cd build
```

2. Configure RTEMS for your target architecture:

```
cd /usr/local/rtems/source/build
../rtems-4.7/configure --target=powerpc-rtems4.7 \
--prefix=/usr/local/rtems/rtems4.7 \
--enable-cxx --enable-rdbg --disable-tests --enable-networking \
--enable-posix --enable-rtemsbsp=mvme2100 \
```

You should replace the ‘powerpc’ with the name of the architecture for which you’re building RTEMS. Common alternatives are ‘m68k’ and ‘i386’ for the Motorola M68k and Intel x86 family of processors, respectively.

Your should replace the ‘mvme2100’ with the board-support packages for your particular hardware

You can build for more than one board-support package by specifying more names on the command line. For example, you could build for a Arcturus uCDIMM ColdFire 5282 system and an MVME-167 system by:

```
cd /usr/local/rtems/source/build
../rtems-4.7/configure --target=m68k-rtems4.7 \
--prefix=/usr/local/rtems/rtems4.7 \
--enable-cxx --enable-rdbg --disable-tests --enable-networking \
--enable-posix --enable-rtemsbsp="uC5282 mvme167" \
```

2.5 Get, build and install some RTEMS add-on packages

The EPICS IOC shell uses the the libtecla or GNU readline package to provide command-line editing and command history. While the IOC shell can be compiled without these capabilities I think they’re important enough to warrant making the effort to download and install the extra packages. GNU readline is more well-tested, but libtecla does not bring along the problems associated with the GNU Public License.

EPICS can use either TFTP or NFS for remote file access. NFS provides considerably more flexibility so it’s a good idea to install this add-on package as well.

7

2.5.1 Download the add-on package sources

The latest versions of these files are in
ftp://ftp.rtems.com/pub/rtems/snapshots/contrib/add_on_packages/
The compressed tar archive in this directory can be downloaded using a web browser or a command-line program such a wget (note that the wget command example has been split to make it fit on this page):
wget --passive-ftp --no-directories --retr-symlinks
"ftp://ftp.rtems.com/pub/rtems/snapshots/contrib/add_on_packages/rtems-addon-packages-20041017.tar.Depending on the type of firewall between your machine and the OAR FTP server you may need to remove the --passive-ftp option from the wget command.
When you are done you should have the compressed archive with a name something like
rtems-addon-packages-20041017.tar.bz2

2.5.2 Unpack the add-on package sources

Change to your RTEMS source directory and unpack the RTEMS sources by:

```
cd /usr/local/rtems/source
bzcat rtems-addon-packages-20041017.tar.bz2 | tar xf -
```

This will unpack the source for all the RTEMS packages into a directory named
rtems-addon-packages-20041017

2.5.3 Set the RTEMS MAKEFILE PATH environment variable

The makefiles in the RTEMS packages use the RTEMS_MAKEFILE_PATH environment variable to determine the target architecture and board-support package. For example,

```
export RTEMS_MAKEFILE_PATH=/usr/local/rtems/rtems4.7/powerpc-rtems4.7/mvme2100
```

will select the Motorola Power PC architecture and the RTEMS mvme2100 board-support package.

2.5.4 Build and install the add-on packages

The bit script in the packages source directory builds and installs all the add-on packages. To run the script change directories to the add-on packages directory and execute:

```
sh bit
```

If you are building for more than one architecture or board-support package, you must run the bit script once for each variation with RTEMS_MAKEFILE_PATH set to the different architecture and board-support package.

2.6 Try running some RTEMS sample applications (optional)

The RTEMS build process creates some sample applications. If you’re just getting started with RTEMS it’s probably a good idea to verify that you can actually run a simple RTEMS application on your target hardware before trying to run a full-blown EPICS IOC application.
The actual method of loading an application into a target processor is hardware-dependent. Section 4.1.3 describes a method which may be used with RTEMS mvme2100 targets.
8

Chapter 3

EPICS Base

The first step in building an EPICS application is to download the EPICS base source from the APS server and unpack it. The details on how to perform these operations are described on the APS web pages and will not be repeated here.
Make sure you get the R3.14.7 or later release of EPICS.

3.1 Specify the location of RTEMS tools and libraries

You must first let the EPICS Makefiles know where you’ve installed the RTEMS development tools and libraries. The default location is
/usr/local/rtems/rtems4.7

If you’ve installed the RTEMS tools and libraries in a different location and have not created a symbolic link from

/usr/local/rtems/rtems4.7

to wherever you’ve installed RTEMS you need to edit the EPICS configuration file

configure/RELEASE

In this file you’ll find the lines

RTEMS_BASE=/usr/local/rtems/rtems4.7

RTEMS_VERSION=4.7

while will have to be changed to reflect the directory where you installed RTEMS.

If you installed the RTEMS readline or tecla add-on packages you can edit your EPICS local configuration file

(configure/os/CONFIG_SITE.Common.RTEMS) and change the EPICSCOMMANDLINE_LIBRARY definition

from EPICS to READLINE or LIBTECLA, respectively. If you don’t want to use NFS to access remote files you

can add

OP_SYS_CFLAGS += -DOMIT_NFS

to this file.

3.2 Specify the network domain

If you’re using neither DHCP/BOOTP not non-volatile RAM to provide network configuration information to your

RTEMS IOCs you should edit your EPICS local configuration file

(configure/CONFIG_SITE.Common.RTEMS)

and specify your Internet Domain Name as:

9

OP_SYS_CFLAGS += -DRTEMS_NETWORK_CONFIG_DNS_DOMAINNAME=your.dnsname.here

3.3 Specify the network interface

Some RTEMS board support packages support more than one type of network interface. The pc386 BSP, for example,

can be configured to use several different network interface cards. By default the EPICS network configuration for

the pc386 BSP loads network drivers for all network interfaces which support run-time probing so if you’ve got one

of these network interfaces you don’t need to make any changes to the EPICS network configuration. If not, see the

comments in

src/RTEMS/base/rtems_netconfig.c

for instructions on selecting a network interface card when building your EPICS application.

3.4 Specify the target architectures

The configure/CONFIG_SITE file specifies the target architectures and board support packages to be supported.

For example, I regularly build for a single target:

CROSS_COMPILER_TARGET_ARCHS=RTEMS-mvme2100

If you want to build for multiple RTEMS targets you would change this line to something like

CROSS_COMPILER_TARGET_ARCHS=RTEMS-mvme2100 RTEMS-uC5282 RTEMS-pc386

The format of the target architecture names is RTEMS-bspname, where RTEMS- indicates that the RTEMS development

tools and libraries should be used, and bspname is the name of the RTEMS target architecture and board support

package used back in section 2.4.4.

If you build EPICS base on multiple host architectures and want to build RTEMS target support on only one of those

host architectures the target specification can instead be performed in

configure/os/CONFIG_SITE.<hostArch>.Common/CONFIG_SITE

3.5 Build EPICS base

This step is very simple. Just change directories to the EPICS base directory and run

```
make
```

After a while you'll end up with a working set of EPICS base libraries and tools.

10

Chapter 4

EPICS Applications

Now that you've built the EPICS base libraries you're ready to build and run your first EPICS application. Once you've
got this application running you can forget about this tutorial and revert to using the standard EPICS documentation.

You can start with your own special application or you can start with the example application that is provided with the
EPICS distribution. The following sections describe the procedure to create, build and run this example application.

4.1 The EPICS example application

4.1.1 Build the example application

1. Create a new directory to hold the application source and then 'cd' to that directory.
2. Run the makeBaseApp.pl program to create the example application:

```
makeBaseApp.pl -t example test
makeBaseApp.pl -i -t example -a RTEMS-mvme2100 test
```

If you get complaints about not being able to run these commands you've probably forgotten to put the 'bin' directory created in the previous section on your shell executable search path.

The 'test' in the two makeBaseApp.pl commands can be replaced with whatever name you want to give your example application. The 'RTEMS-mvme2100' can be replaced with whatever target architecture you plan to use to run the example application.

3. Build the example application by running

```
make
```

4.1.2 Install the EPICS IOC files on the TFTP/NFS server

The application build process creates db and dbd directories in the top-level application directory and produces a
set of IOC shell commands in the st.cmd file in the iocBoot/ioctest directory. If you chose an application name different than test in the previous step, the directory name will change accordingly. These directories and their
contents must be copied to your TFTP/NFS server. The actual location depends upon the remote file access technique
being used as described in the following section.

11

4.1.3 Run the example application on an RTEMS IOC

Everything's now ready to go. The only item left is arranging some way of loading the RTEMS/EPICS application
executable image into the IOC. There are many ways of doing this (floppy disks, PROM images, etc.), but I find
using a BOOTP/DHCP/TFTP server to be the most convenient. The remainder of this section describes how I load
executables into my RTEMS-mvme2100 and RTEMS-pc386 IOCs. If you're running a different type of IOC you'll
have to figure out the required steps on your own. The RTEMS documentation may provide the required information
since an EPICS IOC application is an RTEMS application like any other.

Some RTEMS board-support packages require an NTP server on the network. If such an IOC doesn't receive a timesynchronization
packet from an NTP server the IOC time will be set to 00:00:00, January 1, 2001.

4.1.4 Location of EPICS startup script

If you're using BOOTP/DHCP to provide network configuration information to your IOC you should use DHCP sitespecific option 129 to specify the path to the IOC startup script. If you’re using PPCBUG you should set the NIOT ‘Argument File Name’ parameter to the IOC startup script path.

If you’re using NFS for remote file access the EPICS initialization uses the startup script pathname to determine the parameters for the initial NFS mount. If the startup script pathname begins with a ‘/’ the first component of the pathname is used as both the server path and the local mount point. If the startup script pathname does not begin with a ‘/’ the first component of the pathname is used as the local mount point and the server path is “/tftpboot/” followed by the first component of the pathname. This allows the NFS and TFTP clients to have a similar view of the remote filesystem.

If you’re using TFTP for remote file access the RTEMS startup code first changes directories to /epics/hostname/ within the TFTP server, where hostname is the Internet host name of the IOC. The startup code then reads IOC shell commands from the st.cmd script in that directory. The name (st.cmd) and location of the startup script are fixed from the IOCs point of view so it must be installed in the corresponding location on the TFTP server. Many sites run the TFTP server with an option which changes its root directory. On this type of system you’ll have to copy the startup script to the /epics/hostname/ directory within the TFTP server’s root directory. On a system whose TFTP server runs with its root directory set to /tftpboot the startup script for the IOC whose name is norumx1 would be placed in

```
/tftpboot/epics/norumx1/st.cmd
```

The application build process creates db and dbd directories in the top-level application directory. These directories and their contents must be copied to the IOC’s directory on the TFTP server. For the example described above the command to install the files for the norumx1 IOC is

```
cp -r db dbd /tftpboot/epics/norumx1
```

MVME2100 Using PPCBUG

1. Use the PPCBUG ENV command to set the ‘Network PReP-Boot Mode Enable’ parameter to ‘Y’.
2. Use the PPCBUG NIOT command to set the network parameters. Here are the parameters for a test IOC I use:

```
Controller LUN =00
Device LUN =00
Node Control Memory Address =FFE10000
Client IP Address =192.168.8.8
Server IP Address =192.168.8.131
Subnet IP Address Mask =255.255.252.0
Broadcast IP Address =192.168.11.255
12
Gateway IP Address =0.0.0.0
Boot File Name ("NULL" for None) =/epics/test/bin/RTEMS-mvme2100/example.boot
Argument File Name ("NULL" for None) =/epics/test/iocBoot/iocexample/st.cmd
Boot File Load Address =001F0000
Boot File Execution Address =001F0000
Boot File Execution Delay =00000000
Boot File Length =00000000
Boot File Byte Offset =00000000
BOOTP/RARP Request Retry =00
TFTP/ARP Request Retry =00
Trace Character Buffer Address =00000000
BOOTP/RARP Request Control: Always/When-Needed (A/W)=W
```

BOOTP/RARP Reply Update Control: Yes/No (Y/N) =Y

3. Set up your TFTP/NFS servers. PPCBUG uses TFTP to load the executable image then the EPICS initialization uses NFS to read the EPICS startup script (the ‘Argument File Name’ in the NIOT parameters). I set the TFTP server root to /tftpboot and arrange for the NFS server to export /tftpboot/epics to the IOCs. This arrangement lets me simply copy the application tree, beginning at the <top> directory to the TFTP/NFS server area.

PC386

1. Install an EtherBoot bootstrap PROM image obtained from the ‘ROM-o-matic’ server (http://www.rom-omatic.net/) on the IOC network interface cards.

2. Set up your BOOTP/DHCP server to provide the network configuration parameters to the IOC.

3. The TFTP and NFS servers can be configured as noted above.

Arcturus uCDIMM ColdFire 5282

1. Use the bootstrap setenv command to set the EPICS and network configuration parameters:

```
IPADDR0=192.168.8.27
HOSTNAME=ioccoldfire
BOOTFILE=epics/ucdimm/bin/RTEMS-uC5282/ucdimm.boot
NAMESERVER=192.168.8.167
NETMASK=255.255.252.0
CMDLINE=epics/i2c/iocBoot/ioci2c/st.cmd
SERVER=192.168.8.161
NFSMOUNT=106.74@nfssrv:/export/nfssrv:/home/nfssrv
```

2. Use the bootstrap tftp command to load the IOC application image (which may include a tar image of the in-memory filesystem contents in which case the CMDLINE will likely look something like /st.cmd and the NFSMOUNT need not be present).

3. Use the bootstrap goram command to start the application or the program command to burn the image into the on-board flash memory. In the latter case you may want to also use the setenv command to set the AUTOBOOT environment variable.

The cexp package can be used to incrementally load your application components. This package can not be distributed with RTEMS or EPICS since it uses components covered by the GNU Public License.

13

Appendix A

Script to get and build the cross-development tools

If you’re feeling brave you can turn loose the following script. It attempts to download, unpack, configure, build and install the GNU cross-development tools and libraries for one or more target architectures. To use the script, set the ARCHS environment variable to the architectures you wish to support, then

```
sh getAndBuildTools.sh
```

```
#!/bin/sh
#
# Get, build and install the latest cross-development tools and libraries
#
#
# Specify the architectures for which the tools are to be built
# To build for single target: ARCHS="m68k"
#
```

```
ARCHS="${ARCHS:-m68k i386 powerpc}"
#
# Specify the versions
#
GCC=gcc-4.1.0
BINUTILS=binutils-2.16.1
NEWLIB=newlib-1.14.0
BINUTILSDIFF=20050816
GCCDIFF=
NEWLIBDIFF=
RTEMS_VERSION=4.7
#
# Where to install
#
PREFIX="${PREFIX:-/usr/local/rtems/rtems-${RTEMS_VERSION}}"
14
#
# Where to get the GNU tools
#
RTEMS_SOURCES_URL=ftp://www.rtems.com/pub/rtems/SOURCES
RTEMS_BINUTILS_URL=${RTEMS_SOURCES_URL}/${BINUTILS}.tar.bz2
RTEMS_GCC_URL=${RTEMS_SOURCES_URL}/${GCC}.tar.bz2
RTEMS_NEWLIB_URL=${RTEMS_SOURCES_URL}/${NEWLIB}.tar.gz
RTEMS_BINUTILS_DIFF_URL=${RTEMS_SOURCES_URL}/${BINUTILS}-rtems-${BINUTILSDIFF}.diff
RTEMS_GCC_DIFF_URL=${RTEMS_SOURCES_URL}/${GCC}-rtems-${GCCDIFF}.diff
RTEMS_NEWLIB_DIFF_URL=${RTEMS_SOURCES_URL}/${NEWLIB}-rtems-${NEWLIBDIFF}.diff
#
# Uncomment one of the following depending upon which your system provides
#
#GET_COMMAND="curl --remote-name"
GET_COMMAND="wget --passive-ftp --no-directories --retr-symlinks "
GET_COMMAND="wget --no-directories --retr-symlinks "
#
# Allow environment to override some programs
#
MAKE="${MAKE:-make}"
export MAKE
SHELL="${SHELL:-/bin/sh}"
export SHELL
#
# Get the source
# If you don’t have curl on your machine, try using
# wget --passive-ftp --no-directories --retr-symlinks <<url>>
# If that doesn’t work, try without the --passive-ftp option.
#
getSource() {
${GET_COMMAND} "${RTEMS_BINUTILS_URL}"
if [ -n "$BINUTILSDIFF" ]
then
${GET_COMMAND} "${RTEMS_BINUTILS_DIFF_URL}"
fi
${GET_COMMAND} "${RTEMS_GCC_URL}"
if [ -n "$GCCDIFF" ]
then
${GET_COMMAND} "${RTEMS_GCC_DIFF_URL}"
fi
${GET_COMMAND} "${RTEMS_NEWLIB_URL}"
if [ -n "$NEWLIBDIFF" ]
```

```
then
${GET_COMMAND} "${RTEMS_NEWLIB_DIFF_URL}"
fi
}
#
15
# Unpack the source
#
unpackSource() {
rm -rf "{$BINUTILS}"
bzcat "${BINUTILS}.tar.bz2" | tar xf -
for d in "${BINUTILS}"*.diff
do
if [ -r "$d" ]
then
cat "$d" | (cd "${BINUTILS}" ; patch -p1)
fi
done
rm -rf "${GCC}"
bzcat "${GCC}.tar.bz2" | tar xf -
for d in "${GCC}"*.diff
do
if [ -r "$d" ]
then
cat "$d" | (cd "${GCC}" ; patch -p1)
fi
done
rm -rf "${NEWLIB}"
zcat "${NEWLIB}.tar.gz" | tar xf -
for d in "${NEWLIB}"*.diff
do
if [ -r "$d" ]
then
cat "$d" | (cd "${NEWLIB}" ; patch -p1)
fi
done
(cd "${GCC}" ; ln -s "../${NEWLIB}/newlib" newlib)
}
#
# Build
#
build() {
PATH="${PREFIX}/bin:$PATH"
for arch in $ARCHS
do
rm -rf build
mkdir build
cd build
"${SHELL}" "../${BINUTILS}/configure" "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${${MAKE}
-w all install
cd ..
rm -rf build
mkdir build
16
cd build
"${SHELL}" "../${GCC}/configure" "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${--with-gnu-as -
-with-gnu-ld --with-newlib --verbose \
--with-system-zlib --disable-nls \
--enable-version-specific-runtime-libs \
--enable-threads=rtems \
--enable-languages=c,c++
${MAKE} -w all
${MAKE} -w install
cd ..
done
}
#
# Do everything
#
# Comment out any activities you wish to omit
#
set -ex
getSource
unpackSource
build
```

17