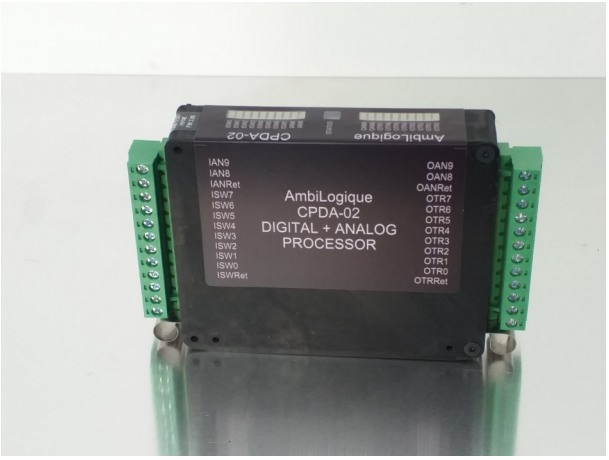
CPDA-02 Processor

Advanced Programmer's Guide

A Much Better Idea : Logical : Unique

AmbiLogique

Electronic Controllers



AmbiLogique CPDA-02 PLC Processor Module

Contents

AmbiLogique Ltd

347 Broadway, Stratford 4332, New Zealand

+64 6 928 4942 ph www.ambilogique.com sales@ambilogique.com

1. Outline

This guide is for users who need to go beyond the standard control diagram, stand-alone method of applying these processors.

Some of the advanced facilities and methods are accessible via the control diagram, some via the serial port, and some through both.

For serial port access, it is vital to master the ARTPC serial protocol. The protocol definition is available on the AmbiLogique website as a free download. In addition we can provide a 'C' library and a LabVIEW driver package.

2. sUbslots

Having used the CPDA-02 in various applications, you will have become familiar with the physical inputs and outputs on sUbslot 0, and probably the K-Factors in sUbslot 6.

Most of the advanced facilities are available via sUbslots other than these, and this guide will cover the other sUbslots, their purposes, and methods of access.

a) sUbslot 1: Internal Signals (Control Diagram Variables)

The assignment of Control Diagram Variables is made by the AmbiLogique compiler, and is set out in the wire list 'wires.txt'

This document lists both the constants (at the beginning of the list) and the variables (in the latter part). SUbslot 1 accesses purely the variables, and is able to read them but not write to them.

The following page shows an extract from a wire list, showing the end of the constants and the beginning of the variables. You can see that each part of the list has its own set of reference numbers (refnums) starting from 0.

Register 0 of sUbslot 1 is a read/write register which is used to set and read back the refnum of the first variable to be accessed.

Register 1 reads the value of the variable addressed by R0.

If R1 is repeatedly read, it will return the current value of the variable in real time, and the refnum in R0 will not change.

If R2 or higher registers are read, the refnum is incremented before each read. This means that a block of registers will return the values of a corresponding block of variables. For example, suppose we want to read back variables 10 through 15 as a block, we would set R0 to 10, then read R1 through R6 as a single block.

We could also read R0 through R6 as a single block and achieve the same result, except that the refnum of the first variable is returned as the first item, followed by the 6 variables that we requested.

This block access method is used in some of the other sUbslots.

```
-------- REFNUM 41  Const  0.500000 ---------------------------
Sources:
  -none-
Sinks:
  F0808.2
-------- REFNUM 42  Const  160.000000 -------------------------
Sources:
  -none-
Sinks:
  F0916.1

-------- REFNUM 0  XY Status ----------------------------------
Sources:
  XY Status.0
Sinks:
  F0110.1, F0118.1, F0103.1, F0104.1

-------- REFNUM 1  __BIX0S1U0R1 -------------------------------
Sources:
  __BIX0S1U0R1.0
Sinks:
  XY Status.1

-------- REFNUM 2  X PosAct -----------------------------------
Sources:
  X PosAct.0
Sinks:
  X Position.1, F0514.1
```

Figure 1: Part of a wire list 'wires.lst'

b) sUbslot 2: Operating Mode and Communications Stats

Reg 0 Mask 1 Read/write Operating Mode

0 = Normal Run; 1 = Stop.

Defaults to 0 (Run) on power up. Note that starting and stopping does not cause a reset: on starting, the process continues from where it left off when stopped.

Reg 0 Mask 128 Write Software Initiated Restart

When set, this causes a Processor Reset. All variables are set to 0 and the control diagram scan starts from Function Block 0.

This is used by the AmbiL_PLC software to trigger a restart when a new Control Diagram has been uploaded.

Reg 1 Mask 255 Read Time Used

This reads a value from 0 to 255 which indicates how much of the cycle time (62.5 ms) has been taken up with the execution of the Control Diagram.

Time Used varies with the mix of Function Blocks in the Control Diagram, but normally allows for up to 800 Function Blocks before a cycle overflow occurs.

Note that a cycle overflow is not a disaster: the CPDA-02 will always complete the scan of the full Control Diagram. However a cycle overflow will halve the speed at which the CPDA-02 operates because 2 cycles are used for each scan instead of 1.

Reg 2 Mask 0 Read Millions of backplane transactions

Reg 3 Mask 0 Read Backplane transactions up to 1 million

These 2 registers combine to read back the number of backplane transactions carried out by the processor since the last reset.

Reg 4 Mask 0 Read No. of backplane transactions in error

This is the total number of backplane transactions where the dialog did not complete successfully. The actual errors are categorised in the following registers:

Reg 5 Mask 0 Read NoRep errors

This presents a count of No-Replies, where there was no response to a BRQ or BCM packet. A large number of these errors usually indicates a missing or faulty slave module on the bus.

Reg 6 Mask 0 Read Timeout errors

This is where a dialog did not complete within the allocated time. A partial response was obtained, otherwise the error would have been flagged as a NoRep.

Reg 7 Mask 0 Read Sentinel errors

An incorrect packet sentinel was received. A BRQ should get a BAS in reply; a BCM should get a BAK. Any other packet sentinel is an error.

Reg 8 Mask 0 Read Format errors

ARTPC is very precise about the format of the packet: field sentinels need to be in the right place, and the field widths need to be correct according to the sentinels.

The ARTPC document sets out these rules. Sentinels appearing in the wrong place, or field widths in error will trigger these errors.

Reg 9 Mask 0 **Read** **Checkword errors**

If the checkword of a received packet did not compute correctly, the packet is rejected and this error count is incremented. Note that old-style ARTP (without the checkword) packets can be processed without error, but obviously the protection of the checkword is not available.

c) sUbslot 3: Control Diagram Constants

This sUbslot operates in a similar way to sUbslot 1, except that the constants accessed here can be written as well as read.

The AmbiLogique compiler uses this sUbslot in order to write the constants during an Upload.

Register 0 of sUbslot 3 is a read/write register which is used to set and read back the refnum of the first constant to be accessed.

Register 1 reads or writes the value of the constant addressed by R0.

If R2 or higher registers are accessed, the refnum is incremented before each read or write. This means that a block of registers will read or write the values of a corresponding block of constants. For example, suppose we want to read back constants 10 through 15 as a block, we would set R0 to 10, then read R1 through R6 as a single block.

We could also read R0 through R6 as a single block and achieve the same result, except that the refnum of the first constant is returned as the first item, followed by the 6 constants that we requested.

Writing constants is not straightforward, due to their being stored in flash memory.

Whilst a bit which is '1' can be written to '0', the reverse is not true. This means that before a block of memory can be written to, it needs to be bulk erased in order to set the memory to all '1'.

This bulk erase operation is programmed into the CPDA-02 operating system, and is triggered when the first constant in a block is written to. Blocks are 128 constants long, so an update to constant 0 will erase all of the constants 0..127. All of these will need to be re-written (or as many as are used in the particular Control Diagram).

Random-access editing of constants is not an option, and it is vital that any revision of constants follows the block rules above: write the entire block, starting at refnum 0 or 128 or 256 as appropriate.

d) sUbslot 4: Function Block (User Program) area

Reg 0 Mask 0 Read/write Execution Number

This register holds the refnum of the Function Block accessed by the following registers:

Reg 1 Mask 0 Read/Write Function Block Type

Reg 2..16 Masks 0 Read/write Pin Descriptors

The signal refnum to which the pin connects is Mask 4095 (hex 0x0FFF) and the pin type is mask 28672 (hex 0x7000).

Registers 1 and upwards are read-write only when the processor is stopped; when in Run mode these registers are read-only.

```
          FUNCTION BLOCK LIST

-------- EXEC NUM 0  TYPE  PREVSCAN  ID F0710 ------------------
Pin Name        Type    Signum  Name/Value

  0 prev(x)     analog   78     Z Ready
  1 x           analog  136     F0712

-------- EXEC NUM 1  TYPE  PREVSCAN  ID F0604 ------------------
Pin Name        Type    Signum  Name/Value

  0 prev(x)     analog   77     Y Ready
  1 x           analog  123     F0612

-------- EXEC NUM 2  TYPE  PREVSCAN  ID F0505 ------------------
Pin Name        Type    Signum  Name/Value

  0 prev(x)     analog   76     X Ready
  1 x           analog  110     F0512
```

Figure 2: Part of Function Block List fblock.lst

Again, the Function Blocks are stored in flash memory, so the rules about bulk erase and complete write apply here as with constants. Each flash memory block contains 15 Function Blocks, and the auto erase function applies here also. So, starting at 0 a block erase will occur before this block is written, then before blocks 15, 30 etc.

e) sUbslot 5: Direct Memory Access

This provides direct access to the entire memory area of the processor chip. In order for this to be useful, the source code for the firmware needs to be at hand.

This facility is used only by AmbiLogique's technical staff, and is not available for general use.

f) sUbslot 6: K-Factors

Reg 0 Mask 0 Read/write K-Factor Reference Number (refnum)

This is written to access the K-Factor required, as with sUbslot 1.

Reg 1 Mask 0 Read/write K-Factor Value

As before, a block of K-Factors can be written by writing a block to Registers 1 and upwards. The refnum is incremented before the values in Registers 2 and upwards are written to.

Again, the K-Factors are stored in flash memory, 128 to a block. An auto-erase is carried out on each block before the first value is written to it.

g) sUbslot 255: Firmware and Serial

Reg 0 Mask 0 Read Firmware Revision

(Major_revision * 256) + minor_revision.

Example: Rev 3.20 => (3 * 256) + 20 = 788.

Easily decoded as a hexadecimal number:

788d = 314h => 300h + 14h => 3.20

Reg 1 Mask 0 Read Serial No.

This is a decimal number which is composed of:-

- 2-digit year
- 2-digit month
- 3-digit sequence number

so, for example 2409053 is a module manufactured in September 2024, and is the 53rd in that month.

3. Backplane Communications

The backplane communication system is extensible over suitable screened cables so that peripheral modules can be situated anywhere up to a couple of hundreds of metres from the Processor Module.

The system operates using RS-485 serial asynchronous communications at 115200 bits/sec, 8 bits, no parity, 1 stop bit (115200, 8, N, 1). The protocol employed is the AmbiLogique Register Transfer Protocol with Checkword, or ARTPC for short.

The ARTPC definition document is available free of charge in the Downloads section of the AmbiLogique website.

3.1. *The ARTPC Protocol*

An outline of the protocol follows: a sample printout of a set of backplane transactions is provided below with a breakdown and description of each of the fields.

```
-jo03jno04
          B5A2
!jo03jno04jq8E6D9C1q8E6AC7Bo20
                              D70D
-jnjo02o07
          7EA2
!jnjo02o07jjjp0168jp0168j
                         0075
+jnjo08nr
         29B8
*jnjo09j
        442E
```

Figure 3: Snapshot of backplane communications

This snapshot was taken by connecting a computer, fitted with a USB-to-RS485 adaptor, to the backplane. The computer therefore sees packets sent by both the processor and its slaves.

Each transaction is a dialog initiated by the bus master (the Processor Module) and responded to by the addressed slave.

Note that every other line is dropped below the end of the previous line. This is the correct representation of a line-feed character which marks the start of the Checkword.

a) Block Request

The first packet is (we have artifically inserted spaces to separate the fields, and added [lf] for a linefeed and [cr] for carriage return):

- j o03 j n o04 [lf] B5A2 [cr]

- is the unique packet sentinel for a Block Request (BRQ)

j is both a numeric sentinel and the boX address. Numeric sentinels are lower case letters in the range 'j' to 'y'. They are decoded by subtracting 'j' and expressing the result as a 4-bit field. The bits are:

8	M	minus flag	0	the number is non-negative;
4	Y	'yes' flag	0	the number is zero;
3	SS	size flags	0	no numeric values follow.

Thus we interpret the boX address as 0.

o03 is the Slot address: 'o' – 'j' = 5 or 0101 in binary:

8	M	minus flag	0	the number is non-negative;
4	Y`	'yes' flag	1	the number is non -zero;
3	SS	size flags	1	2 hex characters follow;

03 is the 8-bit value: gives a Slot address of 3.

j another 0 for sUbslot address.

n base register address: 'n' – 'j' = 4 or 0100 in binary:

8	M	minus flag	0	the number is non-negative;
4	Y	'yes' flag	1	the number is non-zero;
3	SS	size flags	0	no numeric values follow.

The base register requested is 1

o04 Similar to the Slot address analysis, this gives a register count of 4.

B5A2 This is the Checkword calculated in accordance with the ARTPC rules.

b) Block Assert

The slave module in Slot 3 responds with:

! j o03 j n o04 j q8E6D9C1 q8E6AC7B o20 [lf] D70D [cr]

The first character '!' is the unique packet sentinel for a Block Assert (BAS) packet.

The next part of the packet is a repeat of the boX, Slot, sUbslot, Base register and Number of registers embodied in the BRQ. An exception can occur where a BRQ requests registers which are outside the range of the slave peripheral device. In this case, the slave will correct the Number of registers and construct its packet accordingly.

The contents of R1 are reported as 'j', i.e. 0.

The next 2 fields have the numberic sentinel 'q', 'q' – 'j' = 7: binary 0111.

8	M	minus flag	0	the number is non-negative;
4	Y`	'yes' flag	1	the number is non -zero;
3	SS	size flags	3	7 hex characters follow;

The first character is another set of flags, expressed as a hexadecimal digit, in this case 8:

8	F	float flag	1	this is a floating-point number;
4	O	overflow	0	within range of a float;
2	X	previous Y	0	used only within the PLC firmware;
1		not used.		

As this is a floating point number, the next 2 characters are the signed 8-bit exponent E6 = -26 (this is 2 to the power of -26);

The final 4 characters are the unsigned 16-bit mantissa D9C1 = 55745.

The number represented is +830.66E-6 or just under 1/1000.

c) Block Command

Further down the data log we see a Block Command packet. So far we have looked at the way the Processor requests data from its slaves: now we see how it sends commands.

+ j n j o08 n r [lf] 29B8 [cr]

The packet sentinel is '+' which indicates a Block Command (BCM) packet.

This is directed at boX 0, Slot 1, sUbslot 0, base register 8, one register.

The value sent is 'r' (which is unusual): 'r' – 'j' = 8:

8	M	minus flag	1	the number is negative;
4	Y	'yes' flag	0	the number is zero;
3	SS	size flags	0	no numeric values follow.

So weirdly, the register is loaded with -0.

d) Block Acknowledge

The slave responds with a Block Acknowledge:

* j n j o09 j [lf] 442E [cr]

'*' is the sentinel for Block Acknowledge. The following 3 values tell us that this came from boX 0, Slot 1, sUbslot 0.

The next value o09 is the next register above the last register affected by the Block Command.

The final value before the Checkword is the error code: in this case 'j' signifying 0 for no error (even though the encoding of the data was slightly odd).

3.2. CPDA-02 Backplane Management

Backplane management is set up and carried out automatically by the CPDA-02. This information is provided for understanding: there is no programming activity involved.

a) Control Diagram Analysis

When a new Control Diagram is loaded to a CPDA-02, one of the important tasks it carries out before starting to execute the diagram is to organise its management strategy for the backplane.

It is perfectly permissible and normal to place TERMIN and TERMOUT blocks at appropriate places throughout the various diagram sheets, and these will be in a random order.

TERMIN and TERMOUT blocks which address peripherals outside of Slot 0 (the Processor slot) are always consolidated into BUSIN and BUSOUT blocks together with mask functions by the AmbiLogique compiler.

After loading a Control Diagram, the CPDA-02 makes a list of all of these BUSIN and BUSOUT blocks and sorts them into order.

It then amalgamates all the blocks that share boX, Slot and sUbslot addresses into combined blocks, expanding the Base-Register and Number-of-Registers parameters as required.

The CPDA-02 can then construct a set of BRQ commands which will read back every slave register that is accessed by that particular Control Diagram.

At the same time, the CPDA-02 sets up a memory area which will receive and hold all of these register images, and creates an index so that the images can be accessed via the appropriate XSUR addresses.

b) Run-time Backplane Operation

When the CPDA-02 is running its control diagram, it also runs the bus scan in the background. A drop-through state machine is employed so that the time taken away from Control Diagram execution is minimised.

Normally this continuously runs through the BRQ list, interrogates the slaves, and lodges the newest version of their data (including command registers) in the allocated memory area. This strategy makes the most of the Block capability of ARTPC.

Whenever the Control Diagram calls up slave data as an input, it accesses the image area, thereby getting the most up-to-date information.

BUSOUT blocks operate in a different way. Data pushed into the BUSOUT block by the Control Diagram is compared with the image read back as part of the bus scan process.

If the data matches, no action is needed. On the other hand, the data needs to be sent to the slave: a single BCM packet is threaded into the head of the backplane transaction queue so that it is enacted as quickly as possible.

A Much Better Idea : Logical : Unique
AmbiLogique
Electronic Controllers

WARNING
SAFETY-CRITICAL SYSTEMS

A Safety-Critical system is a system whose failure or malfunction could cause death, significant injury or loss of property.

AmbiLogique products incorporate electronic hardware and software, both of which carry a remote but real possibility of failure. AMBILOGIQUE DOES NOT WARRANT, CLAIM OR REPRESENT THAT ITS PRODUCTS ARE INFALLIBLE.

It is therefore THE RESPONSIBILITY OF THE DESIGNER of any safety-critical system which incorporates AmbiLogique products to ensure that:-

1. The system is designed so that any failure of an AmbiLogique component will not cause death, injury or loss of property.
2. The system incorporates independent monitoring means which detect the failure of any of the electronic control elements.
3. The system has alternative and independent means of control which enable it to be controlled and shut down in an orderly manner.
4. Any and all other industry-specific safety requirements are fully implemented.

Revision History:

R 0.0 2025-08-04 Initial issue