Serial Compact Hopper Mk 3

Combi

Product Manual

Version 1.3

Revision History

Issue	Date	Comments
1.0	10/03/05	First issue
1.1	12/04/05	Appendix E, further notes
1.2	19/04/05	Change of name from Accounter to Combi
	11/05/05	Added encryption enabled flag to ‘Test hopper’ response
1.3	18/05/05	TSP number added and released.

Contents

1. Introduction

The Serial Compact Hopper Mk3 **Combi** or SCH3A is a serially controlled accumulator hopper that can dispense more than 1 coin type. The host machine requests that the hopper dispense a value of coins rather than a number of coins. As which coin is dispensed is determined when the coin leaves the hopper, the hopper may stop the motor before the requested coin value is reached to prevent over-payout. It is then up to the host machine to dispense the remaining coins from another hopper. In other words, an accumulator hopper must be used in conjunction with a standard hopper - it cannot operate in isolation.

The serial interface is **cctalk**, firmly established as the leading, low-speed, device control protocol in the money-transaction industry. A key feature of cctalk is its optimal balance between simplicity and security.

2. Product Features

The following features are available on SCH3A…

- Accumulator operation. For example, a single hopper can contain both 1 and 2 Euros and dispense both.
- Coin dispensing security. The serial command to pay out a coin can use a 64-bit encryption key which changes randomly after each operation. This makes serial 'cloning' whereby a handheld terminal listens to the serial bus and plays back dispense commands a fruitless exercise. A typical dispense cycle will see a total of 192 bits of random information transferred across the bus.
- Sophisticated optical lightguide for coin exit sensing uses transmissive as well as reflective paths to combat fraud. An ejected coin must block light as well as reflect light for the hopper to continue dispensing coins.
- Slider sensor. A feedback signal on the mechanical slider used to eject coins ensures the correct sequence of events is seen for each coin dispensed. Any discrepancy stops the motor and an error condition is reported back to the host machine.
- PIN number security. There is an option to protect the hopper with a PIN number so that if it is stolen from a machine and plugged into another one it cannot be made to work. It is the equivalent of a mechanical lock with the machine manufacturer generating and keeping the key.
- Anti-jam operation. If the hopper experiences a coin jam during a payout sequence it will automatically reverse in order to clear the jam.
- Software fuse. If an absolute maximum current threshold is exceeded then the hopper aborts payout with an error code.
- Power fail protection. A non-volatile memory keeps track of coins paid out. If power is lost during a payout sequence then the residual value of coins to pay can be read back after the machine re-initialises.
- Unique serial number. Each hopper is manufactured with a unique 24-bit serial number which cannot be modified by external means.

- Coin counting. Counters for each coin type record the number of coins paid out of the hopper. Counters are implemented in NV Memory and can be reset by host software.
- Level sensor support. There is an option on the hopper to fit high or low level sensor plates and the status of these can be read back by the host machine on cctalk to indicate ‘nearly empty’ or ‘nearly full’.
- Remote configuration. Motor parameters such as reversing current and payout timeout can be changed with serial commands. No changes to electronic components are required.
- Multi-drop operation. A number of serial hoppers can be connected to the same serial bus. Device addresses default to those determined by the wiring harness but they can be changed in software to any 8-bit value.
- User memory. 16 bytes of non-volatile memory are available for unrestricted use by the host machine. There are various security and auditing tasks which could be accomplished with this feature.
- Diagnostic and error reporting. Full access to diagnostic and error codes are made available over serial.
- Code protection. The software is protected with an internal, independent, watchdog circuit. A ‘crash’ in the software will result in a clean reset of code.

3. cctalk Design Parameters

Refer to issue **4.4** of the ‘cctalk Serial Communication Protocol / Generic Specification’ for an explanation of the protocol and its implementation on any platform.

The latest published version of the cctalk specification is available at…
http://www.cctalk.org

This product is configured as…
cctalk b96.p0.v24.a5.d0.c8.m0.x8.e0.i4.r4

In other words…

- 9600 baud (1 start bit, 1 stop bit, no parity)
- Open-collector interface
- Nominal supply voltage +24V
- Serial data pull-up voltage +5V
- Supply sink
- Connector type 8 (10-way pin header)
- Slave device only
- 8-bit addition checksum
- No encryption (no encrypted protocol layer)
- Minor release 4
- Major release 4

The hopper can **only** operate at 9600 baud.

A +12V supply version of the SCH3A is available.
cctalk b96.p0.v12.a5.d0.c8.m0.x8.e0.i4.r4

Although the hopper does not use an encrypted protocol layer as used currently on bill validators, the dispense command can be protected with a 64-bit encryption key.

For details of the encryption algorithm on Combi, please request a copy of the following document from the Technical Services department at Money Controls UK. You may be required to sign a NDA.

Serial Compact Hopper Mk3 - Combi - Encryption Standard - CMF2-1

Filename : SCH3 Combi Encryption v2.1.doc

Email Request : techsupport@moneycontrols.com

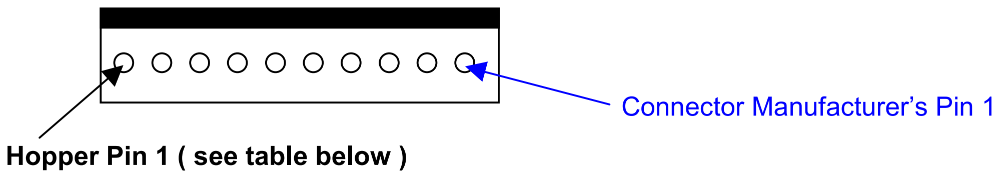
4. Connectors

4.1. *Serial Connector*

PCB Connector
2.54mm pitch with locking wall

Part No. : CviLux CI3110-P1V00
(equivalent parts : Molex type 6410 or AMP 1-640456-0)

Keying Information


Connector Manufacturer's Pin 1
Hopper Pin 1 (see table below)

View of Connector from Top

4.1.1. Serial Connector Pinout

Pin	Function
1	Address select 3 - MSB
2	Address select 2
3	Address select 1 - LSB
4	+Vs
5	+Vs
6	0V
7	0V
8	/DATA (cctalk)
9	N/C
10	N/C

Operation can be achieved with just 3 wires…
- +Vs to pin 4
- 0V to pin 6
- Bi-directional serial data line to pin 8

+Vs is either +12V or +24V depending on the build of the hopper.

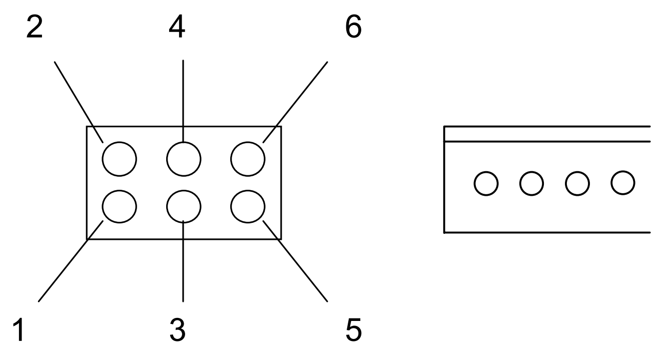
Pins 4 and 5, and pins 6 and 7, are linked internally. The provision of extra pins is to simplify the manufacture of a multi-drop cable using thicker wire for the power leads. There can be a 'power-in' and a 'power-out' pin, and the hoppers daisy-chained..

Recommended wire gauge : AWG 22

4.2. Level Sensor Connector

PCB Connector
2.54mm pitch

4.2.1. Level Sensor Connector Pinout



Pin	Function
1	High Level Plate
3	Low Level Plate
5	Plate Common
2	High Level Link
4	Low Level Link
6	Link Common

4.2.2. Level Sensor Connector Wiring

To notify the hopper software that level plate sensors are fitted, the link pins should be connected as follows…

Mode	Connections	Logical
High level plates only	pin 2 to pin 4	High to Low
Low level plates only	pin 4 to pin 6	Low to Common
High & low level plates	pin 2 to pin 4 to pin 6	High to Low to Common

Otherwise, no connections should be made.

The level plates themselves should be connected through the corresponding plate pin (pin 1 for high level, pin 3 for low level) and the plate common (pin 5).

5. Address Selection

The default cctalk bus address for a 'Payout' device is 3. This is the address of the Serial Compact Hopper if no connections are made to the address select pins (pins 1 to 3) on the connector.

For applications requiring more than one hopper on the serial bus, one or more of the address select lines may be connected to +Vs. A total of 8 unique bus address may be generated in this way, in the range 3 to 10 inclusive.

X = Connect to +Vs (Pins 4, 5)			Serial Address
Address select 3	Address select 2	Address select 1	
			3
		X	4
	X		5
	X	X	6
X			7
X		X	8
X	X		9
X	X	X	10

A number of mating connectors on a multi-drop bus cable may each be wired uniquely to allow operation of multiple hoppers. Since address selection is done externally, any Serial Compact Hopper may be plugged into any position on the bus and the host machine will know which one is paying out a particular coin.

☞Address determination from the connector is only done at power-up or reset. Changing the address select lines afterwards has no effect.

Note : Addresses may be changed in software to values other than those in the above table. Refer to the 'Address change' and 'Address random' serial commands. These values are lost at power-down or reset.

6. Optional Dispense Encryption

A 64-bit encryption mechanism is used to ensure that an illegal attempt to dispense coins from SCH3A is detected and prevented. The details of the algorithm used are not published in this document. If you wish to operate with encryption enabled then please request a copy of the encryption documentation for the SCH3 Combi from the Technical Services department of Money Controls UK. A copy will be sent to you after certain approval procedures have been gone through.

To show the principal for dispensing encrypted coin values, an example is shown here with a 'trivial' encryption mechanism but the overall procedure is the same. Byte values between brackets are shown in hex.

First of all we pump the random number generator of the hopper by sending 8 bytes of random data to it…

Command = Pump RNG
Transmitted data : [34] [A2] [D7] [0F] [35] [17] [55] [94]
Received data : ACK

This is not an essential step but is useful to broaden the spectrum of cipher keys that are transmitted along the serial bus and which may be 'recorded' by a hacker. As the host machine is likely to be an AWP machine with a sophisticated random number generator, way beyond the capability of the hopper microcontroller, we may as well make use of it. Note that the pump value does not pre-set or 'seed' the RNG as that would defeat the security mechanism, but only scrambles it further. The exact details of the scrambling algorithm will not be documented.

Then we request a key cipher key…

Command = Request cipher key
Transmitted data : <none>
Received data : [E5] [88] [13] [07] [46] [FE] [29] [05]

A new cipher key must always be requested prior to dispensing coins. There is no point using an old copy as it changes after every dispense command. The 'Request cipher key' command itself can be repeated in the event of a communication error and the cipher key will be re-transmitted rather than regenerated.

Now we combine the cipher key with the value of coins to pay out (in this example 2 Euros = 200 cents = 00 C8 hex) by tagging it onto the end of the data block…

Non-encrypted data = [E5] [88] [13] [07] [46] [FE] [29] [05] [C8] [00]

In this case we will assume that the CMF (Cryptographic Mapping Function) is simply inverting all the bytes (new data = FF - old data).

Performing this calculation on each of the cipher key bytes we obtain…

Encrypted data = [1A] [77] [EC] [F8] [B9] [01] [D6] [FA] [C8] [00]

Note that the value of coins to pay out is unencrypted but it is used in the real CMF to ensure the message packet cannot be intercepted and the value changed.

Now we send that data to the hopper to pay out a coin…

Command = Dispense hopper coins
Transmitted data : [1A] [77] [EC] [F8] [B9] [01] [D6] [FA] [C8] [00]
Received data : [5] - example event counter

The next time we pay out a coin the cipher key will have changed, so unless the algorithm is known, simple command 'cloning' will not work.

In practice, the CMF will be far more complex than this example. Firing random data at the hopper will also prove fruitless as there are astronomical odds against a successful code match.

7. PIN Number Mechanism

A PIN number is provided on SCH3A as an **optional** security feature. By default, units are shipped without the PIN number mechanism enabled. If this feature is not required or its use is too restrictive then it can simply be ignored.

By programming a PIN number into the device, if the hopper device is subsequently powered down or removed to another location then unless the PIN number is known, no coins can be dispensed. This is another layer of defence against the determined hacker who wishes to experiment with the encryption mechanism. However it does require the host machine keeping track of the PIN numbers of any hoppers used in that machine.

Various possibilities include…

1. Don't use a PIN number
Nice and easy that one.
2. Fix the PIN number to the same value always
This can be done but is not very secure. Once the PIN number is known then there is effectively no PIN number protection on any of the hoppers. It is simple to manage though and the 'master' PIN number is unlikely to be forgotten.
3. Scramble the PIN number and store in the user memory
This is quite a clever idea because it means you can randomise the PIN number on each hopper and as long as you know how you scrambled it, it can be recovered, unscrambled and sent to the hopper during the initialisation routine. Security relies on keeping this scrambling algorithm secret.
4. Log the PIN number versus serial number
As each hopper has a unique serial number then this gives a convenient method of storing the serial number against a random PIN number in a central database which all the machines have access too on a network. This is the most secure method because unless the PIN number transaction is captured on the bus at just the right moment in time, and for that particular hopper, the only way to obtain the PIN would be by exhaustive searching.

If you are unfortunate enough to have a hopper for which you have forgotten the PIN number then contact Money Controls for details of any possible recovery mechanism that we may have in place at the time.

7.1. PIN Number Example

This is how to set the PIN number on a new hopper to 1-2-3-4 which does not already have one…

Command = Enter new PIN number
Transmitted data : [1] [2] [3] [4]
Received data : ACK

Subsequent use of this command will still return an ACK but will not actually change the PIN number to any other value. This is a 'use once' command.

As soon as a PIN number is programmed, the 'Dispense hopper coins' command will fail until this PIN number is re-entered with the command below.

Likewise, after powering up SCH3A with the PIN number mechanism enabled, it must be entered prior to paying out coins.

Command = Enter PIN number
Transmitted data : [1] [2] [3] [4]
Received data : ACK

Note that an ACK is always returned, even if the PIN number is incorrect. This increases security.

8. What happens after Power Up ?

Parameter	cctalk Header	Action
event counter	133	Cleared to zero
payout value remaining	133	Cleared to zero
last payout : coin value paid	133	Restored from NV memory
last payout : coin value unpaid	133	Restored from NV memory
current limit	247	Initialised to CURLMT
motor stop delay	247	Initialised to STOPDLY
payout timeout	247	Initialised to PAYTIM
maximum current measured	247	Cleared to zero
connector address	247	Value read from connector
cctalk address	All	Initialised to connector address + 3
indexed hopper dispense count	130	Restored from NV memory
PIN number mechanism	218	PIN number must be re-entered to dispense coins
rx timeouts	002	Cleared to zero
rx bytes ignored	002	Cleared to zero
rx bad checksums	002	Cleared to zero

For the action on status flags refer to Table 1 in the ‘Test hopper’ command.

The ‘last payout’ values refer to the last payout performed by the hopper, whether it completed successfully or not. The ‘Power-down during payout’ flag returned by the ‘Test hopper’ command can identify whether power was lost during the payout such that the remainder of coins can be paid out by the host.

9. What happens after Software Reset ?

Parameter	cctalk Header	Action
event counter	133	Cleared to zero
payout value remaining	133	Cleared to zero
last payout : coin value paid	133	No change
last payout : coin value unpaid	133	No change
current limit	247	Initialised to CURLMT
motor stop delay	247	Initialised to STOPDLY
payout timeout	247	Initialised to PAYTIM
maximum current measured	247	Cleared to zero
connector address	247	Value read from connector
cctalk address	All	Initialised to connector address + 3
indexed hopper dispense count	130	No change
PIN number mechanism	218	PIN number must be re-entered to dispense coins
rx timeouts	002	Cleared to zero
rx bytes ignored	002	Cleared to zero
rx bad checksums	002	Cleared to zero

For the action on status flags refer to Table 1 in the ‘Test hopper’ command.

The ‘last payout’ values refer to the last payout performed by the hopper, whether it completed successfully or not. The ‘Power-down during payout’ flag returned by the

‘Test hopper’ command can identify whether power was lost during the payout such that the remainder of coins can be paid out by the host.

10. Power Fail Recovery

If power is lost during a payout operation then when power is re-applied the ‘Power-down during payout’ flag in hopper status register 3 will be set. This can be checked by the host software with the ‘Test hopper’ command before a ‘Reset device’ command is sent. The ‘Request hopper polling value’ will then display values of
[last payout : coin value paid] and [last payout : coin value unpaid] as they were just before power was lost. The host can then resume dispensing the remainder of coins.

The accuracy of the last payout counters is ±1 coin as there could be a coin over the exit sensor when power is lost. In the case of an accumulator hopper this coin is of unknown type i.e. low value or high value.

11. Example Hopper Payout (with encryption)

The hopper is assumed to be on address 3.

cctalk header bytes shown in blue
cctalk data bytes shown in red

All numbers in decimal.

Step 1 : Clear error flags

```
Command 'Reset device'
TX = 003 000 001 001 251
RX = 001 000 003 000 252 = ACK
```

Any pending error conditions are now cleared.

Step 2 : Enable hopper

```
Command 'Enable hopper'
TX = 003 001 001 164 165 178
RX = 001 000 003 000 252 = ACK
```

The hopper is now enabled and can dispense coins.

Step 3 : Pump random number generator

```
Command 'Pump RNG'
TX = 003 008 001 161 180 136 148 074 077 198 003 194 097
RX = 001 000 003 000 252 = ACK
```

Step 4 : Request encryption key

```
Command 'Request cipher key'
TX = 003 000 001 160 092
RX = 001 008 003 000 095 192 137 000 132 087 188 070 111
```

These numbers now need to be ‘scrambled’ by the encryption algorithm which is not shown here.

Step 5 : Dispense coins

```
Command 'Dispense hopper value'
TX = 003 010 001 134 212 027 199 251 234 176 142 078 044 001 024
RX = 001 001 003 000 001 250
```

The last 2 data bytes sent by the host = ‘044 001’ are for a coin dispense value of 300.
The returned event counter = 1

Step 6 : Poll hopper

The first poll is…

```
TX = 003 000 001 133 119
RX = 001 007 003 000 001 044 001 000 000 000 000 199
```

The return message shows event 1, value 300 remaining, value 0 paid, value 0 unpaid.

Example of final poll… (depends on sequence of coins dispensed)

```
Command 'Request hopper polling value'
TX = 003 000 001 133 119
RX = 001 007 003 000 001 000 000 200 000 100 000 200
```

The return message shows event 1, value 0 remaining, value 200 paid, value 100 unpaid.

Step 7 : Test flags

```
Command 'Test hopper'
TX = 003 000 001 163 089
RX = 001 003 003 000 000 004 000 245
```

The return message has bit 2 set of flag register 2 = 'Use other hopper for change'

12. Example Hopper Payout (without encryption)

The hopper is assumed to be on address 3.

cctalk header bytes shown in blue
cctalk data bytes shown in red

All numbers in decimal.

Step 1 : Clear error flags

```
Command 'Reset device'
TX = 003 000 001 001 251
RX = 001 000 003 000 252 = ACK
```

Any pending error conditions are now cleared.

Step 2 : Enable hopper

```
Command 'Enable hopper'
TX = 003 001 001 164 165 178
RX = 001 000 003 000 252 = ACK
```

The hopper is now enabled and can dispense coins.

Step 3 : Dispense coins

```
Command 'Dispense hopper value'
TX = 003 010 001 134 000 000 000 000 000 000 000 000 044 001 063
RX = 001 001 003 000 001 250
```

The encryption key is sent as 8 x zeros.
The last 2 data bytes sent by the host = ‘044 001’ are for a coin dispense value of 300.
The returned event counter = 1

Step 4 : Poll hopper

The first poll is…

```
TX = 003 000 001 133 119
RX = 001 007 003 000 001 044 001 000 000 000 000 199
```

The return message shows event 1, value 300 remaining, value 0 paid, value 0 unpaid.

Example of final poll… (depends on sequence of coins dispensed)

```
Command 'Request hopper polling value'
TX = 003 000 001 133 119
RX = 001 007 003 000 001 000 000 200 000 100 000 200
```

The return message shows event 1, value 0 remaining, value 200 paid, value 100 unpaid.

Step 5 : Test flags

```
Command 'Test hopper'
TX = 003 000 001 163 089
RX = 001 003 003 000 000 004 000 245
```

The return message has bit 2 set of flag register 2 = ‘Use other hopper for change’

13. Example Hopper Payout (without encryption / refused)

The hopper is assumed to be on address 3.

cctalk header bytes shown in blue
cctalk data bytes shown in red

All numbers in decimal.

Step 1 : Clear error flags

```
Command 'Reset device'
TX = 003 000 001 001 251
RX = 001 000 003 000 252 = ACK
```

Any pending error conditions are now cleared.

Step 2 : Enable hopper

```
Command 'Enable hopper'
TX = 003 001 001 164 165 178
RX = 001 000 003 000 252 = ACK
```

The hopper is now enabled and can dispense coins.

Step 3 : Dispense coins

```
Command 'Dispense hopper value'
TX = 003 010 001 134 000 000 000 000 000 000 000 000 044 001 063
RX = 001 000 003 005 247 = NAK
```

The encryption key is sent as 8 x zeros.
The last 2 data bytes sent by the host = '044 001' are for a coin dispense value of 300.

Step 4 : Test flags

```
Command 'Test hopper'
TX = 003 000 001 163 089
RX = 001 003 003 000 008 000 000 241
```

The return message has bit 3 set of flag register 1 = 'Opto fraud attempt, path blocked during idle'

The hopper refused to dispense coins because there is (or was) something obstructing the exit optos.

14. Command List

36 commands are supported on SCH3A as listed below.

Refer to Appendix E to compare commands with SCH2 and SCH1.

Header 254 :	Simple poll
Header 253 :	Address poll
Header 252 :	Address clash
Header 251 :	Address change
Header 250 :	Address random
Header 247 :	Request variable set
Header 246 :	Request manufacturer id
Header 245 :	Request equipment category id
Header 244 :	Request product code
Header 242 :	Request serial number
Header 241 :	Request software revision
Header 236 :	Read opto states
Header 219 :	Enter new PIN number
Header 218 :	Enter PIN number
Header 217 :	Request payout high / low status
Header 216 :	Request data storage availability
Header 215 :	Read data block
Header 214 :	Write data block
Header 196 :	Request creation date
Header 195 :	Request last modification date
Header 192 :	Request build code
Header 169 :	Request address mode
Header 165 :	Modify variable set
Header 164 :	Enable hopper
Header 163 :	Test hopper
Header 161 :	Pump RNG
Header 160 :	Request cipher key
Header 134 :	Dispense hopper value
Header 133 :	Request hopper polling value
Header 132 :	Emergency stop value
Header 131 :	Request hopper coin value
Header 130 :	Request indexed hopper dispense count
Header 004 :	Request comms revision
Header 003 :	Clear comms status variables
Header 002 :	Request comms status variables
Header 001 :	Reset device

15. Commands in Detail

All byte values shown in [decimal] unless otherwise stated.

15.1. Header 254 : Simple poll

Transmitted data : <none>
Received data : ACK

This is a good command to use to confirm that a device is plugged into the expected address, powered-up and operating correctly. A total of 5 bytes are sent to the hopper which then replies with 5 bytes.

15.2. Header 253 : Address poll

Transmitted data : <none>
Received message : {variable delay} <slave address byte>

Only a single byte is returned by the hopper rather than a full cctalk message packet.

See the generic specification for more details.

15.3. Header 252 : Address clash

Transmitted data : <none>
Received message : {variable delay} <slave address byte>

Only a single byte is returned by the hopper rather than a full cctalk message packet.

See the generic specification for more details.

15.4. Header 251 : Address change

Transmitted data : [address]
Received data : ACK

The address specified overrides that determined by the connector wiring loom. The new value is lost at power-down or reset.

15.5. Header 250 : Address random

Transmitted data : <none>
Received data : ACK

The address is randomly set to a value between 3 and 255. The broadcast address 0, the default bus master address 1, and the default coin acceptor address 2, are automatically avoided for your convenience. The new value is lost at power-down or reset.

15.6. Header 247 : Request variable set

Transmitted data : <none>
Received data : [current limit] [motor stop delay] [payout timeout]
[maximum current measured] [supply voltage]
[connector address]

15.6.1. current limit

This is the current threshold at which the motor reverses in order to clear jams.

Refer to CURLMT in Appendix B for details of scaling and its default value.

15.6.2. motor stop delay

This is the time the motor is allowed to run on for after detecting the last coin in a payout sequence and should be sufficient for the coin to exit cleanly.

Refer to STOPDLY in Appendix B for details of scaling and its default value.

15.6.3. payout timeout

This is the total amount of time allowed for **each** coin to be paid out, allowing for some reversing in the event of a jam. After this time, the motor is stopped.

Refer to PAYTIM in Appendix B for details of scaling and its default value.

15.6.4. maximum current measured

Measured with the same units as [current limit]. The current is sampled and averaged as the motor is running and should be used as an approximate guide only.

This measurement can be cleared to zero with the ‘Reset device’ command.

15.6.5. supply voltage

Not supported on SCH3A. Returns zero always.

15.6.6. connector address

Range 0 to 7.

This is the number specified by the address select pins on the connector. The device address on the cctalk bus is this value plus 3 if it hasn't been changed subsequently using serial commands.

device address = connector address + 3

Note that this value is continuously updated unlike the actual comms address.

15.7. Header 246 : Request manufacturer id

Transmitted data : <none>
Received data : "MCI"

This is the 3 letter abbreviation for Money Controls International. Refer to Table 6 in the cctalk generic specification for 'cctalk Standard Manufacturer Strings'.

15.8. Header 245 : Request equipment category id

Transmitted data : <none>
Received data : "Payout"

All hoppers return 'Payout' for equipment category.

15.9. Header 244 : Request product code

Transmitted data : <none>
Received data : "SCH3A"

SCH3A = Serial Compact Hopper Mk3 Accumulator.

15.10. Header 242 : Request serial number

Transmitted data : <none>
Received data : [serial 1 - LSB] [serial 2] [serial 3 - MSB]

This is a 24-bit binary serial number.

Decimal serial number = [serial 1] + 256 * [serial 2] + 65536 * [serial 3]

e.g. [78] [97] [188] = 78 + 256 * 97 + 65536 * 188 = 12,345,678

15.11. Header 241 : Request software revision

Transmitted data : <none>
Received data : “SCH3A-Vx.y”

x, y = 0, 1, 2… depending on the revision level of the software.

Note that prototype units return “SCH3A-Px.y” (P replaces V)

15.12. Header 236 : Read opto states

Transmitted data : <none>
Received data : [payout opto]

[payout opto]
Bit mask :
B0 - payout opto TX (0 = path clear, 1 = path blocked)
B1 - payout opto TX (copy of B0)
B2 - payout opto RX (0 = path clear, 1 = light reflected)
B3 - payout opto Slider Sensor (0 = slider clear, 1 = slider active)
B4 - B6 - not used, 0 returned
B7 - payout opto TX * RX

The coin exit sensor consists of independent transmissive and reflective light paths. A valid payout coin must break the transmissive beam as well as form a reflective beam. The optos are continuously sampled in the background and the current state is reported by this command.

Holding a coin in the exit sensor should return…
MSB 10000111 LSB = 135 decimal

Bit 1 returns the same information as Bit 0.

Bit 7 returns the transmissive state ANDed with the reflective state.

Note that this command is designed for test purposes only (checking the opto-electronics are working correctly) and not for counting coins during a payout sequence ! Counting coins during payout is handled automatically by the hopper firmware and is performed a lot faster than polling serially.

15.13. Header 219 : Enter new PIN number

Transmitted data : [PIN1] [PIN2] [PIN3] [PIN4]
Received data : ACK

A factory-fresh hopper has the PIN number mechanism disabled.

A manufacturer can subsequently program the PIN number to any chosen value using this command and after that the number cannot be changed, even if the existing PIN number has been entered correctly. It is a 'once-only' lockout mechanism - turning the power off and on does not clear the PIN number.

A PIN number of [0] [0] [0] [0] is legal and would have to be entered.

The 'Dispense hopper value' command is the only one which is 'blocked' by the PIN number mechanism - all other commands operate as normal.

Entering a new PIN number after one has already been programmed will still result in the return of an ACK even though the PIN number remains unchanged.

The 'Test hopper' command can be used to see if a PIN number has been programmed - refer to bit 7 of 'hopper status register 2'.

If you program and lose a PIN number then see Appendix I.

15.14. Header 218 : Enter PIN number

Transmitted data : [PIN1] [PIN2] [PIN3] [PIN4]
Received data : ACK

If the PIN number mechanism is enabled then the 'Dispense hopper coins' command will not work unless the correct PIN number has been entered. The current PIN number is lost after a…

- power-down
- software reset
- emergency stop with the motor running

Incorrect PIN numbers are always ACKed as if they had worked.

15.15. Header 217 : Request payout high / low status

Transmitted data : <none>
Received data : [level status]

This command returns the status of the level sensor. Level sensors may or may not be fitted, depending on the build of the hopper. If fitted then either low, high or both level sensors will be supported in software.

[level status]
Bit mask :
Bit 0 - Low level sensor status (1 = lower than low level trigger)
Bit 1 - High level sensor status (1 = higher than or equal to high level trigger)
Bit 2 - not used
Bit 3 - not used
Bit 4 - Low level sensor support (1 = feature supported and fitted)
Bit 5 - High level sensor support (1 = feature supported and fitted)
Bit 6 - not used
Bit 7 - not used

For the operator…
Bit 0 is set if the hopper is NEARLY EMPTY.
Bit 1 is set if the hopper is NEARLY FULL.

The normal operating condition of the hopper is bits 0 & 1 clear.

If the raw level sensor inputs are reading NEARLY EMPTY as well as NEARLY FULL then this is an illegal condition (it can only be one or the other) and bits 0 & 1 are both left clear.

The level sensor inputs are debounced with a time Tlevdeb in Appendix C to remove the effects of shifting coins.

The level sensor status does not affect operation of the hopper in any way. Coins can still be paid out of a nearly empty hopper. You can also attempt to dispense coins from a completely empty hopper.

If more accurate coin levels are required by the host machine then this must be done by coin-in and coin-out counting. There is no method to obtain an absolute coin count from the hopper.

15.16. Header 216 : Request data storage availability

Transmitted data : <none>
Received data : [memory type] [read blocks] [read bytes per block]
[write blocks] [write bytes per block]

[memory type]
= 2, permanent (limited use)
[read blocks]
= 3
[read bytes per block]
= 8
[write blocks]
= 3
[write bytes per block]
= 8

In other words, 24 bytes of NV Memory are available for reading and 24 bytes for writing, accessed 8 bytes at a time.

Refer to Appendix D for details of the memory map.

15.17. Header 215 - Read data block

Transmitted data : [block number]
Received data : [data 1] [data 2]… [data 8]

[block number]
0 to 2

Provides read access to the NV Memory.

Refer to Appendix D for details of the memory map.

15.18. Header 214 - Write data block

Transmitted data : [block number] [data 1] [data 2]… [data 8]
Received data : ACK

[block number]
0 to 2

Provides write access to the NV Memory.

Refer to Appendix D for details of the memory map.

15.19. Header 196 : Request creation date

Transmitted data : <none>
Received data : [date code LSB] [date code MSB]

Refer to generic specification for date format.

15.20. Header 195 : Request last modification date

Transmitted data : <none>
Received data : [date code LSB] [date code MSB]

Refer to generic specification for date format.

Header 192 : Request build code

Transmitted data : <none>
Received data : 8 x ASCII chars

The build code is currently used to return the level sensor options on the hopper as an ASCII string. The same information can be obtained by examining bits 4 and 5 of the level status return byte to the 'Request payout high / low status' command.

The build code is determined automatically by the hopper during initialisation. It is assumed any build options do not change while power remains on the unit, otherwise a software reset needs to be issued.

`'Lev HiLo'`	for high and low level sensor fitted
`'Lev Hi  '`	for high level sensor only fitted
`'Lev   Lo'`	for low level sensor only fitted
`'Standard'`	for standard model

15.21. Header 169 : Request address mode

Transmitted data :	<none>
Received data :	[address mode]

The address selection method is determined by the product and in this case the value **4A hex** is always returned.

[address mode]
Bit mask :
B0 - Address is stored in ROM
B1 - Address is stored in RAM
B2 - Address is stored in EEPROM or battery-backed RAM
B3 - Address selection via interface connector
B4 - Address selection via PCB links
B5 - Address selection via switch
B6 - Address may be changed with serial commands (volatile)
B7 - Address may be changed with serial commands (non-volatile)

4A hex = Address stored in RAM
Address selection via interface connector
Address may be changed with serial commands (volatile)

The address of the hopper defaults to that in the connector wiring after a power-up or software reset.

15.22. Header 165 : Modify variable set

Transmitted data :	[current limit] [motor stop delay] [payout timeout] [single coin mode]
Received data :	ACK

Refer to 'Request variable set' for more details - the format of data is the same.

[single coin mode]
Not used in accumulator mode - value is ignored

This command allows some of the motor control variables to be modified but does not necessarily have to be sent before using the hopper. The default values listed in Appendix B are normally optimal.

Variable set changes are volatile. Any custom values are lost at power-down or reset.

15.23. Header 164 : Enable hopper

Transmitted data : [enable code]
Received data : ACK

[enable code]
165 - enable hopper payout
not 165 - disable hopper payout

The hopper must be enabled before coins can be paid out. Only the decimal value 165 (A5 hex, 10100101 binary) enables it, all other values disable it.

If a hopper is disabled prior to a 'Dispense hopper coins' command, a NAK is returned.

A 'Reset device' command or a power-down cycle will cause the hopper to be disabled by default.

15.24. Header 163 : Test hopper

Transmitted data : <none>
Received data : [hopper status register 1] [hopper status register 2]
[hopper status register 3]

This command reports back various operating and error flags from the hopper and is the equivalent of the 'Perform self-check' command in coin acceptors and bill validators.

[hopper status register 1]
Bit mask :

B0 - Absolute maximum current exceeded	1 = current exceeded
B1 - Payout timeout occurred	1 = timeout occurred
B2 - Motor reversed during last payout to clear a jam	1 = motor reversed
B3 - Opto fraud attempt, path blocked during idle	1 = fraud attempt
B4 - Opto fraud attempt, short-circuit during idle	1 = fraud attempt
B5 - Opto blocked permanently during payout	1 = blocked
B6 - Power-up detected	1 = power-up detected
B7 - Payout disabled	1 = payout disabled

[hopper status register 2]
Bit mask :

B0 - Opto fraud attempt, short-circuit during payout	1 = fraud attempt
B1 - Single coin payout mode	1 = single coin mode
B2 - Use other hopper for change	1 = other hopper required
B3 - Opto fraud attempt, finger / slider mis-match	1 = fraud attempt
B4 - Motor reverse limit reached	1 = reverse error
B5 - Inductive coil fault	1 = coil fault
B6 - NV memory checksum error	1 = checksum error
B7 - PIN number mechanism	1 = PIN enabled

[hopper status register 3]
Bit mask :

B0 - Power-down during payout	1 = power-down detected
B1 - Unknown coin type paid	1 = unknown coin
B2 - PIN number incorrect	1 = incorrect PIN
B3 - Incorrect cipher key	1 = incorrect key
B4 - Encryption enabled	1 = encrypted
B5 - Not used	
B6 - Not used	
B7 - Not used	

15.24.1. Flag Explanation

15.24.1.1.Absolute maximum current exceeded

The maximum operating current for the product was exceeded which may be as a result of a coin jam that cannot be cleared by reversing the motor or a fault.

15.24.1.2.Payout timeout occurred

The motor stopped as no coins were seen on the exit optos. This is most likely due to the hopper being empty.

15.24.1.3.Motor reversed during last payout to clear a jam

Motor reversal is the normal operation of the hopper to clear coin jams, especially when the bowl is more than half full. This flag is for information only and does not indicate a problem.

15.24.1.4.Opto fraud attempt, path blocked during idle

This flag reports that the exit optos are blocked even when the hopper is not paying out coins. This may be due to a fraud attempt (opaque object inserted) or a coin stuck in the exit channel. See Appendix H.

15.24.1.5. Opto fraud attempt, short-circuit during idle

This flag reports that the exit optos are picking up light even though switched off during idle. This will happen if somebody is deliberately shining light into the optos to ‘fool’ the hopper into paying out more coins, or if somebody is short-circuiting the electronics with a piece of wire etc. See Appendix H.

15.24.1.6. Opto blocked permanently during payout

If a coin cannot clear the exit sensors as the slider pushes it out, the hopper will stop and this error reported. See Appendix H.

15.24.1.7. Power-up detected

This flag is set when power is applied to the hopper and can be cleared with a software reset. It can be used to detect an unexpected power fail.

15.24.1.8. Payout disabled

The enable / disable state of the hopper is shown by this flag. The hopper always changes to a disabled state after a power-up or software reset. To dispense coins, the hopper must be enabled with an ‘Enable hopper’ command.

15.24.1.9. Opto fraud attempt, short-circuit during payout

This flag reports that the exit optos are picking up light even though switched off during a payout operation. This will happen if somebody is deliberately shining light into the optos to ‘fool’ the hopper into paying out more coins, or if somebody is short-circuiting the electronics with a piece of wire etc. See Appendix H.

15.24.1.10. Single coin payout mode

This flag is shown for compatibility with SCH2 but there is no single coin payout mode on the accumulator hopper.

15.24.1.11. Use other hopper for change

After a payout operation is completed, this flag indicates whether the host needs to dispense the remainder of coins from the other non-accumulating hopper. It may not be set if an error condition caused the hopper to stop prematurely.

15.24.1.12. Opto fraud attempt, finger / slider mis-match

This flag indicates that a coin seen at the exit optos did not trigger a slider sensor feedback signal. This could be indicative of a fraud attempt.

15.24.1.13. Motor reverse limit reached

There is a limit on how many times the motor can reverse to clear a coin jam and if this limit is reached the motor is stopped.

15.24.1.14. Inductive coil fault

The accumulator hopper uses an inductive measurement system to decide which coin has been paid out. If a fault is detected with the loop coil then this error flag is set.

15.24.1.15. NV memory checksum error

A checksum error was detected in the NV (non-volatile) memory of the hopper. Contact Money Controls for possible recovery mechanisms.

15.24.1.16. PIN number mechanism

This indicates a PIN number has been entered into the hopper. If it has, the PIN number needs to be sent to the hopper after a power-up or software reset to allow coins to be dispensed. Hoppers leave the factory without a PIN number.

15.24.1.17. Power-down during payout

If power was lost during a dispense operation then this flag will be set. The host machine can therefore check this flag after a power-up and prior to a software reset to decide whether to resume the payout of coins.

15.24.1.18. Unknown coin type paid

If the accumulator cannot decide what the last coin to exit the hopper was then the motor is stopped and this flag set. It may be that a slug or fraud coin was routed to the hopper bowl in error.

No monetary value is assigned to the unknown coin type and so it will not register on the ‘paid’ value counter.

15.24.1.19. PIN number incorrect

A dispense operation was refused and a NAK message returned to the host either because a PIN number had not been entered or the wrong PIN number had been entered. This will only happen if the PIN number mechanism is enabled.

15.24.1.20. Incorrect cipher key

A dispense operation was refused and a NAK message returned to the host because the cipher key was incorrect. This only applies to encrypted hoppers.

15.24.1.21. Encryption enabled

This flag indicates whether the hopper uses encryption to dispense coins. This option can only be changed with authorised re-programming equipment.

15.24.2. Table 1 - Flag Logic

Bit Position	Short Label	Power-up State	Software Reset	Dispense Requirement	On Dispense NAK	State after Dispense
0.0	Max. Current	0	0	YES	FLAG	Set
0.1	Timeout	0	0	no	-	Set
0.2	Reverse	0	0	no	-	Set
0.3	Idle block	0 α	0 α	YES	FLAG	Set
0.4	Idle s/c	0 α	0 α	YES	FLAG	-
0.5	Pay block	0	0	YES	FLAG	Set
0.6	Power-up	1	0	no	-	-
0.7	Disabled	1	1	YES	FLAG	-
1.0	Pay s/c	0	0	YES	FLAG	Set
1.1	Single coin	0	0	no	-	-
1.2	Other hopper	0	0	no	-	Set
1.3	Slider error	0	0	YES	FLAG	Set
1.4	Reverse limit	0	0	YES	FLAG	Set
1.5	Coil fault	0 α	0 α	YES	FLAG	Set
1.6	Checksum	0 α	0 α	YES	FLAG	-
1.7	PIN enabled	Set	Set	no	-	-
2.0	Power-down	Set	0	YES	FLAG	Set
2.1	Unknown coin	0	0	YES	FLAG	Set
2.2	PIN error	0	0	YES	FLAG Set	-
2.3	Cipher error	0	0	YES	FLAG Set	-
2.4	Encryption	Set	Set	no	-	-

15.24.2.1.Power-up State

This indicates the state of the flag register after the supply voltage has been removed and applied to the hopper.

Set = 0 or 1 depending on a previously stored state

Note α = set if there is permanent fault or error condition

15.24.2.2.Software Reset

This indicates the state of the hopper after the host sends a software reset command, cctalk header 1.

Set = 0 or 1 depending on a previously stored state

Note α = set if there is permanent fault or error condition

15.24.2.3. Dispense Requirement

This indicates if the flag needs to be clear for a dispense operation. If the flag is set and it needs to be cleared then a software reset command must be issued first followed by a hopper enable.

15.24.2.4. On Dispense NAK

If the hopper refuses a host request to dispense coins then the reason is indicated by one of the FLAGS.

FLAG Set = hopper may set this flag before sending a NAK back.

15.24.2.5. State after Dispense

This indicates any flags that may be set after the dispense operation terminates and the motor stops.

15.25. Header 161 : Pump RNG

Transmitted data :	[random 1] [random 2] [random 3] [random 4]
	[random 5] [random 6] [random 7] [random 8]
Received data :	ACK

This command pumps the random number generator of the hopper with extra random variables to make prediction of the next random number in the sequence a lot harder. Its use is optional but recommended where security is paramount.

15.26. Header 160 : Request cipher key

Transmitted data :	<none>
Received data :	[key 1] [key 2] [key 3] [key 4]
	[key 5] [key 6] [key 7] [key 8]

This command requests the encryption key required for coin payout. It may be requested repeatedly in the event of a comms error since it only changes after a…

- ‘Dispense hopper value’ command
- ‘Pump RNG’ command
- Power-down or software reset

15.27. Header 134 : Dispense hopper value

Transmitted data :	[sec 1] [sec 2] [sec 3] [sec 4]
	[sec 5] [sec 6] [sec 7] [sec 8]
	[coin value LSB] [coin value MSB]
Received data :	[event counter] or NAK

[coin value]
Range 0 to 65,535. The units are always the lowest denomination in that currency e.g. cents or pence rather than Euros, dollars or pounds. To dispense 2 Euros send 200.

This command instructs the hopper to dispense the value of coins requested.

Encrypted hoppers need to have the security bytes set to exactly the right values for each dispense operation. The security bytes will be different each time the dispense command is used.

Unencrypted hoppers still require the security bytes to be sent but the values are ignored. It is recommended that they are all set to zero.

The hopper will either return the newly incremented event counter or a cctalk NAK message. An incremented event counter indicates the command was successful and the hopper is going to pay out (or at least *attempt* to pay out) coins. A NAK indicates the dispense command was refused. See table 1 for the 'Test hopper' command to see why a dispense command might be refused.

A request to dispense a coin value of zero will be accepted but the paid and unpaid registers will be zero.

A request to dispense a coin value less than the highest value coin will not operate the motor as the hopper cannot guarantee to dispense the lowest value coin - this is the nature of an accumulator hopper. In this case the command is accepted but the unpaid register is returned with the same value requested.

So on a SCH3A '2 Euro / 1 Euro' hopper, a request to dispense [150] cents will give…
[payout value remaining] = 0
[last payout : coin value paid] = 0
[last payout : coin value unpaid] = 150
and no payout will be attempted. The host machine would then dispense 1.50 Euros from the second 'change' hopper e.g. 3 x 50 cent coins.

The maximum value that can be paid out of the hopper in one go is determined by the currency scaling factor which is usually 100. This scaling factor is not stored by the hopper and so is arbitrary in the sense that the hopper would work just as well with coin values of 1 and 2 rather than 100 and 200. The convention is however to use 100. As the paid and unpaid counters are 2 bytes each, the maximum what can be paid in one go is 65535. So for a Euro hopper this is equivalent to €655.35. To dispense a larger value than this the host machine must send 2 or more dispense commands, adding in the remainder each time.

For instance, to dispense 750 Euros, the host could pay 600 Euros + 150 Euros. If the status after the first dispense was paid 599 Euros, unpaid 1 Euro, then the second dispense should be for 151 Euros. Any remainder after that needs to be paid from the change hopper.

15.28. Header 133 : Request hopper polling value

Transmitted data :	<none>
Received data :	[event counter] [payout value remaining LSB] [payout value remaining MSB] [last payout : coin value paid LSB] [last payout : coin value paid MSB] [last payout : coin value unpaid LSB] [last payout : coin value unpaid MSB]

The dispensing status of the hopper can be checked at any time with this command. If the [payout value remaining] is zero then the hopper is either not paying out or has finished paying out.

On receiving a valid dispense command the status registers are update as followed.

[event counter]
The event counter is incremented. If the event counter is 255 then the value wraps to 1. The value of 0 is only obtained after a power-up or software reset.

[payout value remaining]
Set to the coin value to be dispensed and counts down (by the appropriate amount) as each coin is paid out.

[last payout : coin value paid]
Cleared to zero and then counts up (by the appropriate amount) as each coin is paid out.

[last payout : coin value unpaid]
Cleared to zero for the duration of payout.

When payout is complete the [payout value remaining] will be zero and both the [last payout] registers updated with the correct values.

There are various reasons for the [payout value remaining] becoming zero.

a) The correct value of coins was dispensed
b) The nearest value of coins to that requested was paid and another hopper is needed to dispense the remainder.
c) The hopper timed out as it had run out of coins
d) An attempted fraud was detected
e) A fault or error occurred during payout
f) Power was lost

Which one of these conditions occurred can be discovered by using the 'Test hopper' command immediately after payout stops. The returned status flags indicate the reason why the full value of coins was not dispensed if that was the case.

15.29. Header 132 : Emergency stop value

Transmitted data : <none>
Received data : [payout value remaining LSB] [payout value remaining MSB]

This command stops the hopper motor immediately and returns the payout value remaining.

15.30. Header 131 : Request hopper coin value

Transmitted data : [coin type]
Received data : 6 x ASCII chars
[coin value LSB] [coin value MSB]

[coin type]
Value = 1 or 2
Coin 1 is always the lower value coin.

The return data consists of 8 bytes. The first 6 bytes are the ASCII coin identifier in standard cctalk format and the last 2 bytes represent the coin value in binary.

Coin value = [coin value LSB] + 256 * [coin value MSB]

<u>Example</u>

Coin	Identifier	LSB	MSB
2 Euros	EU200A	C8	00
1 Euro	EU100A	64	00

15.31. Header 130 : Request indexed hopper dispense count

Transmitted data : [coin type]
Received data : [no. of coins 1 - LSB] [no. of coins 2] [no. of coins 3 - MSB]

[coin type]
Value = 1 or 2
Coin 1 is always the lower value coin.

The hopper keeps track of how many of each coin type have been dispensed since the counter was last cleared.

Coins paid = [no. of coins 1] + 256 * [no. of coins 2] + 65536 * [no. of coins 3]

To clear the counter, zeros need to be written into the user memory in the correct location.

15.32. Header 004 : Request comms revision

Transmitted data : <none>
Received data : [cctalk level] [major revision] [minor revision]

[cctalk level]
= 1
[major revision]
= 4
[minor revision]
= 4

In other words, the first issue level of cctalk specification 4.4

15.33. Header 003 : Clear comms status variables

Transmitted data : <none>
Received data : ACK

Clears the counters returned by the ‘Request comms status variables’ command.

15.34. Header 002 : Request comms status variables

Transmitted data : <none>
Received data : [rx timeouts] [rx bytes ignored] [rx bad checksums]

[rx timeouts]
Number of receive message timeouts recorded by the processor.

There is an inter-byte timeout value of Trxout - see Appendix C. The cctalk protocol has a variable length packet structure but data timeouts can be detected and counted.

[rx bytes ignored]
Number of receive bytes ignored by the processor (due to receive buffer overflow). The receive buffer is 20 bytes in size so sending a cctalk command with 100 data bytes (packet size = 105) would give 105 - 20 = 85 bytes ignored.

[rx bad checksums]
Number of messages received by the processor with bad checksums.

All these counters are cumulative and wrap around to 0 after 255.

When testing a new software driver, it is worth checking these counters after a series of message transactions to confirm all is well.

15.35. Header 001 : Reset device

Transmitted data : <none>
Received data : ACK

This is the command required for a 'software reset'.

An ACK is returned **prior** to resetting.

Host software should allow a delay after the ACK before sending the next command to allow the hopper initialisation code to complete - see Tsinit in Appendix C.

After a 'Reset device' command, various status flags are cleared (see the 'Test hopper' command).

Any motor parameters sent with a 'Modify variable set' command will return to their default values. Default values are given in Appendix B.

16. Power Distribution on a Multi-Drop Bus

The multi-drop bus for Serial Compact Hoppers consists of a power, ground and serial data line. When more than one hopper is attached to the bus, all power is transferred along a single cable and significant ground potential shifts can occur.

The following recommendations are made…

- Only operate one serial hopper at a time. Never initiate a second payout sequence when one is already in progress.
- Consider the use of signal conditioning on the serial data line receiver. Perhaps a high-frequency filter and voltage comparitor input with a mid-rail (2.5V) threshold.
- Consider running separate power cables to the hoppers to alleviate the ground potential problem.
- If communication errors still occur, consider changing the topology of the multi-drop bus network. A star network will distribute power more evenly than a ring, tree or daisy-chain network.

17. Electrical Noise - Physical Measures

The cctalk protocol is not designed for long distance transfer but for local hook-up of various peripherals within a machine cabinet. Typical cable lengths are likely to be of the order of a few metres.

Various measures can be taken to minimise the effects of radiated and conducted noise on the cctalk bus.

- Use a good quality regulated power supply with mains filtering. The power rating should be sufficient to handle a Serial Compact Hopper at maximum surge current.
- Do not run the multi-drop bus cables directly next to noisy electrical components if at all possible. These are typically motors, relays, VDU's, fluorescent strip lights etc. If problems are experienced consider the use of screened cable.
- Keep cable runs as short as possible.
- Make sure the cctalk data line has an appropriate load resistor at the host end (typically a 10K pull-up to +5V).
- Do not place too many peripherals on the bus - consider the loading effects of each cctalk interface circuit. The maximum number allowed will depend on the host transceiver circuit.

18. Electrical Noise - Software Measures

There is a big difference in security and reliability terms between a good software implementation of cctalk and that of a poor one.

The following design points should be noted carefully…

Check each cctalk reply packet for errors.

- Was there a low level byte framing error ?
- Were there the correct number of data bytes in the message ?
- Was the return destination address correct ?
- Was the checksum correct ?
- Was the peripheral source address the one expected ?
- Was the return header zero ?

If a reply is returned with an error then the cctalk command can be re-transmitted as many times as deemed appropriate - this is a key feature of cctalk.

Note that the [rx bad checksums] byte returned by the 'Request comms status variables' command is useful for monitoring noise.

19. Appendix A - Physical Specification

19.1. Current

19.1.1. +24 volt motor version

Voltage	19 to 26V dc	
Current	when empty	0.35A typical
	when full	1.0A typical
	at switch-on	3.0A peak
	on reversing	3.5A peak

19.1.2. +12 volt motor version

Voltage	10 to 14V dc	
Current	when empty	0.45A typical
	when full	1.2A typical
	at switch-on	3.5A peak
	on reversing	4.5A peak

19.2. Payout Rate

Rate, multi-coin payout mode	8 to 10 coins/s typical

19.3. Environment

Operating temperature	0 to 60°C
Storage temperature	-20 to 70°C
Operating humidity	10 to 75% RH
Storage humidity	10 to 95% RH, non-condensing

19.4. Maintenance Schedule

Every 100,000 coins	clean light guide with a damp cloth
Every 500,000 coins	replace slider

19.5. Life

Expected product lifetime	3 million coins with routine maintenance

20. Appendix B - Conversion Equations & Default Values

Label	Scaling & Units	Default Byte Value	Default Physical Value
CURLMT	N / 17.0 Amps	27 (α) 39 (β)	1.6A (α) 2.3A (β)
STOPDLY	N ms	0	0ms
PAYTIM	N s	10	10s

Note (α) : +24V motor version Note (β) : +12V motor version

Software Reset

A software reset will force the motor parameters to return to their default values. They are not stored in NV memory. Host software must change any non-default values in the initialisation routine.

21. Appendix C - Timing Parameters etc.

Label	Description	Value	Units
Xtal	PIC Resonator	4.0	MHz
Ifuse24	Absolute maximum trip current (+24V motor)	5.0	A
Ifuse12	Absolute maximum trip current (+12V motor)	7.5	A
Vtrip24	Power-fail trip threshold (+24V motor)	16	V
Vtrip12	Power-fail trip threshold (+12V motor)	9	V
Tpinit	Power-up initialisation time	50	ms
Tsinit	Software reset initialisation time	50	ms
Tlevdeb	Level sensor debounce time	2	s
Trxout	Receive data timeout	50	ms
Header 254	Comms reply delay : Simple poll	< 4	ms
Header 218	Comms reply delay : Enter PIN number (good)	< 4	ms
Header 218	Comms reply delay : Enter PIN number (bad)	< 110	ms
Header 214	Comms reply delay : Write data block	< 60	ms
Header 134	Comms reply delay : Dispense hopper value	< 15	ms
Header 133	Comms reply delay : Request hopper polling value	< 4	ms
Header 161	Comms reply delay : Pump RNG	< 4	ms
Header 160	Comms reply delay : Request cipher key	< 4	ms
Header 001	Comms reply delay : Reset device	< 170	ms

The ‘Comms reply delay’ is measured from the stop bit of the last host transmit byte to the first start bit of the hopper reply.

22. Appendix D - User Memory Map Description

Block No.	Length Bytes	Description	Read / Write Permission
0	8	User data 1	R / W
1	8	User data 2	R / W
2	3	Hopper dispense count 1	R / W
2	1	Checksum	R / W
2	3	Hopper dispense count 2	R / W
2	1	Checksum	R / W

22.1. User data

16 bytes of user data are available to the host machine for any kind of storage requirement. The data can be ASCII test or binary data - there is no restriction on format.

22.2. Hopper dispense count

The total no. of coins dispensed since the counter was reset. Separate counters are provided for each coin type. Each counter is followed by a checksum. The checksum is calculated such that the addition of all counter bytes plus the checksum should be zero.

☞ Note that the cctalk command 'Request indexed hopper dispense count' should be used to read the value of each counter rather than directly out of the user memory area as the counter value may not be immediately stored in EEPROM.

Technology : NV memory is implemented in EEPROM with a working lifetime of at least 100,000 write cycles.

22.2.1. Clearing the Hopper dispense counters

This can be achieved by writing zeros to block 2 as follows…

Command 'Write data block'

```
TX = 003 009 001 214 002 000 000 000 000 000 000 000 000 027
RX = 001 000 003 000 252 = ACK
```

Each block is 8 bytes in size so after the first data byte of 2, for a write to block 2, there should follow 8 zero bytes.

The checksums are set to zero which is correct when the counters are zero.

The counters may be initialised to values other than zero if the checksums are calculated correctly.

23. Appendix E - Mk3 versus Mk2 versus Mk1 Compatibility

This is a quick command comparison between the serial hoppers.

cctalk Command	**Supported on SCH3A ?**	**Supported on SCH2 ?**	**Supported on SCH1 ?**
Header 254 : Simple poll	X	X	X
Header 253 : Address poll	X	X	X
Header 252 : Address clash	X	X	X
Header 251 : Address change	X	X	X
Header 250 : Address random	X	X	X
Header 247 : Request variable set	X	X	X
Header 246 : Request manufacturer id	X	X	X
Header 245 : Request equipment category id	X	X	X
Header 244 : Request product code	X	X	X
Header 242 : Request serial number	X	X	X
Header 241 : Request software revision	X	X	X
Header 236 : Read opto states	Note β	Note α	X
Header 219 : Enter new PIN number	X	X	-
Header 218 : Enter PIN number	X	X	-
Header 217 : Request payout high / low status	X	X	X
Header 216 : Request data storage availability	X	X	X
Header 215 : Read data block	X	X	-
Header 214 : Write data block	X	X	-
Header 192 : Request build code	X	X	X
Header 172 : Emergency stop	-	X	X
Header 171 : Request hopper coin	-	X	X
Header 169 : Request address mode	X	X	X
Header 168 : Request hopper dispense count	-	X	X
Header 167 : Dispense hopper coins	-	Note α	X
Header 166 : Request hopper status	-	X	X
Header 165 : Modify variable set	X	Note α	X
Header 164 : Enable hopper	X	X	X
Header 163 : Test hopper	Note β	Note α	X
Header 161 : Pump RNG	X	X	-
Header 160 : Request cipher key	X	X	-
Header 134 : Dispense hopper value	X	-	-
Header 133 : Request hopper polling value	X	-	-
Header 132 : Emergency stop value	X	-	-
Header 131 : Request hopper coin value	X	-	-
Header 130 : Request indexed hopper dispense count	X	-	-
Header 004 : Request comms revision	X	X	X
Header 003 : Clear comms status variables	X	X	X
Header 002 : Request comms status variables	X	X	X
Header 001 : Reset device	X	X	X

Note α : The data packets have been modified on these commands from Mk1 format to Mk2 format.

Note β : The data packets have been modified on these commands from Mk2 format to Mk3 format.

Remember that just because a command is marked as the same doesn't mean the data returned from the hopper is. For example, SCH2 returns 'Money Controls' for manufacturer id but SCH3A returns 'MCI'.

These are some important operating differences between SCH3A and SCH2.

SCH3A uses the standard cctalk interface circuit as shown in Circuit 1 of the generic specification whereas SCH2 used Circuit 3. This means when no power is applied to the SCH3A hopper the cctalk bus is no longer clamped low and other peripherals can still be communicated with.

Pin 10 of the connector on SCH2 was for a hardware reset. This function is no longer available on SCH3A in order to make the product more robust to static discharge and electrical noise. The software reset function should be used instead. In the event of all communication being 'dead' power should be removed from the hopper. A hardware reset is generated automatically on power being applied.

The unencrypted SCH3A does not require a 'Request cipher key' command to be sent by the host prior to dispensing coins unlike SCH2. The SCH2 hopper required the same sequence of commands for encrypted and unencrypted payout.

24. Appendix F - Coin Dispense Probability Table

24.1. Permutations

The listing below shows the probability of certain coin permutations being dispensed from a 2 Euro / 1 Euro SCH3A for dispense values between 1 and 10 Euros. It is assumed that the SCH3A contains an equal number of each coin type.
R = coin remainder when hopper stops in Euros
2 = No. of 2 Euro coins dispensed
1 = No. of 1 Euro coins dispensed
The ratio of 2 Euro to 1 Euro payout is calculated. If unity then the mix of 2 Euros to 1 Euro coins in the hopper will tend to remain the same.
The overall probability of the other hopper being used to dispense the remainder of coins for each payout request is shown as well.

```
Coin 1 Frequency = 50.0%, Value = 100
Coin 2 Frequency = 50.0%, Value = 200

Dispense     1 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
1.00000000 =                   1 0 0
Probability sum = 1.000
Prob. of using other hopper = 100.0%

Dispense     2 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.500000000 = 1                 1 0 1
.500000000 = 2                 0 1 0
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 50.0%

Dispense     3 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.500000000 = 2                 1 1 0
.250000000 = 1,1               1 0 2
.250000000 = 1,2               0 1 1
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 75.0%

Dispense     4 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.250000000 = 1,2               1 1 1
.250000000 = 2,1               1 1 1
.250000000 = 2,2               0 2 0
.125000000 = 1,1,1             1 0 3
.125000000 = 1,1,2             0 1 2
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 62.5%

Dispense     5 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.250000000 = 2,2               1 2 0
.125000000 = 1,1,2             1 1 2
.125000000 = 1,2,1             1 1 2
.125000000 = 1,2,2             0 2 1
.125000000 = 2,1,1             1 1 2
.125000000 = 2,1,2             0 2 1
.062500000 = 1,1,1,1           1 0 4
.062500000 = 1,1,1,2           0 1 3
```

```
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 68.8%

Dispense     6 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.125000000 = 1,2,2             1 2 1
.125000000 = 2,1,2             1 2 1
.125000000 = 2,2,1             1 2 1
.125000000 = 2,2,2             0 3 0
.062500000 = 1,1,1,2           1 1 3
.062500000 = 1,1,2,1           1 1 3
.062500000 = 1,1,2,2           0 2 2
.062500000 = 1,2,1,1           1 1 3
.062500000 = 1,2,1,2           0 2 2
.062500000 = 2,1,1,1           1 1 3
.062500000 = 2,1,1,2           0 2 2
.031250000 = 1,1,1,1,1         1 0 5
.031250000 = 1,1,1,1,2         0 1 4
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 65.6%

Dispense     7 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.125000000 = 2,2,2             1 3 0
.062500000 = 1,1,2,2           1 2 2
.062500000 = 1,2,1,2           1 2 2
.062500000 = 1,2,2,1           1 2 2
.062500000 = 1,2,2,2           0 3 1
.062500000 = 2,1,1,2           1 2 2
.062500000 = 2,1,2,1           1 2 2
.062500000 = 2,1,2,2           0 3 1
.062500000 = 2,2,1,1           1 2 2
.062500000 = 2,2,1,2           0 3 1
.031250000 = 1,1,1,1,2         1 1 4
.031250000 = 1,1,1,2,1         1 1 4
.031250000 = 1,1,1,2,2         0 2 3
.031250000 = 1,1,2,1,1         1 1 4
.031250000 = 1,1,2,1,2         0 2 3
.031250000 = 1,2,1,1,1         1 1 4
.031250000 = 1,2,1,1,2         0 2 3
.031250000 = 2,1,1,1,1         1 1 4
.031250000 = 2,1,1,1,2         0 2 3
.015625000 = 1,1,1,1,1,1       1 0 6
.015625000 = 1,1,1,1,1,2       0 1 5
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 67.2%

Dispense     8 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.062500000 = 1,2,2,2           1 3 1
.062500000 = 2,1,2,2           1 3 1
.062500000 = 2,2,1,2           1 3 1
.062500000 = 2,2,2,1           1 3 1
.062500000 = 2,2,2,2           0 4 0
.031250000 = 1,1,1,2,2         1 2 3
.031250000 = 1,1,2,1,2         1 2 3
.031250000 = 1,1,2,2,1         1 2 3
.031250000 = 1,1,2,2,2         0 3 2
.031250000 = 1,2,1,1,2         1 2 3
.031250000 = 1,2,1,2,1         1 2 3
.031250000 = 1,2,1,2,2         0 3 2
.031250000 = 1,2,2,1,1         1 2 3
.031250000 = 1,2,2,1,2         0 3 2
.031250000 = 2,1,1,1,2         1 2 3
.031250000 = 2,1,1,2,1         1 2 3
.031250000 = 2,1,1,2,2         0 3 2
.031250000 = 2,1,2,1,1         1 2 3
.031250000 = 2,1,2,1,2         0 3 2
.031250000 = 2,2,1,1,1         1 2 3
.031250000 = 2,2,1,1,2         0 3 2
.015625000 = 1,1,1,1,1,2       1 1 5
```

```
.015625000 = 1,1,1,1,2,1       1 1 5
.015625000 = 1,1,1,1,2,2       0 2 4
.015625000 = 1,1,1,2,1,1       1 1 5
.015625000 = 1,1,1,2,1,2       0 2 4
.015625000 = 1,1,2,1,1,1       1 1 5
.015625000 = 1,1,2,1,1,2       0 2 4
.015625000 = 1,2,1,1,1,1       1 1 5
.015625000 = 1,2,1,1,1,2       0 2 4
.015625000 = 2,1,1,1,1,1       1 1 5
.015625000 = 2,1,1,1,1,2       0 2 4
.007812500 = 1,1,1,1,1,1,1     1 0 7
.007812500 = 1,1,1,1,1,1,2     0 1 6
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 66.4%

Dispense     9 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.062500000 = 2,2,2,2           1 4 0
.031250000 = 1,1,2,2,2         1 3 2
.031250000 = 1,2,1,2,2         1 3 2
.031250000 = 1,2,2,1,2         1 3 2
.031250000 = 1,2,2,2,1         1 3 2
.031250000 = 1,2,2,2,2         0 4 1
.031250000 = 2,1,1,2,2         1 3 2
.031250000 = 2,1,2,1,2         1 3 2
.031250000 = 2,1,2,2,1         1 3 2
.031250000 = 2,1,2,2,2         0 4 1
.031250000 = 2,2,1,1,2         1 3 2
.031250000 = 2,2,1,2,1         1 3 2
.031250000 = 2,2,1,2,2         0 4 1
.031250000 = 2,2,2,1,1         1 3 2
.031250000 = 2,2,2,1,2         0 4 1
.015625000 = 1,1,1,1,2,2       1 2 4
.015625000 = 1,1,1,2,1,2       1 2 4
.015625000 = 1,1,1,2,2,1       1 2 4
.015625000 = 1,1,1,2,2,2       0 3 3
.015625000 = 1,1,2,1,1,2       1 2 4
.015625000 = 1,1,2,1,2,1       1 2 4
.015625000 = 1,1,2,1,2,2       0 3 3
.015625000 = 1,1,2,2,1,1       1 2 4
.015625000 = 1,1,2,2,1,2       0 3 3
.015625000 = 1,2,1,1,1,2       1 2 4
.015625000 = 1,2,1,1,2,1       1 2 4
.015625000 = 1,2,1,1,2,2       0 3 3
.015625000 = 1,2,1,2,1,1       1 2 4
.015625000 = 1,2,1,2,1,2       0 3 3
.015625000 = 1,2,2,1,1,1       1 2 4
.015625000 = 1,2,2,1,1,2       0 3 3
.015625000 = 2,1,1,1,1,2       1 2 4
.015625000 = 2,1,1,1,2,1       1 2 4
.015625000 = 2,1,1,1,2,2       0 3 3
.015625000 = 2,1,1,2,1,1       1 2 4
.015625000 = 2,1,1,2,1,2       0 3 3
.015625000 = 2,1,2,1,1,1       1 2 4
.015625000 = 2,1,2,1,1,2       0 3 3
.015625000 = 2,2,1,1,1,1       1 2 4
.015625000 = 2,2,1,1,1,2       0 3 3
.007812500 = 1,1,1,1,1,1,2     1 1 6
.007812500 = 1,1,1,1,1,2,1     1 1 6
.007812500 = 1,1,1,1,1,2,2     0 2 5
.007812500 = 1,1,1,1,2,1,1     1 1 6
.007812500 = 1,1,1,1,2,1,2     0 2 5
.007812500 = 1,1,1,2,1,1,1     1 1 6
.007812500 = 1,1,1,2,1,1,2     0 2 5
.007812500 = 1,1,2,1,1,1,1     1 1 6
.007812500 = 1,1,2,1,1,1,2     0 2 5
.007812500 = 1,2,1,1,1,1,1     1 1 6
.007812500 = 1,2,1,1,1,1,2     0 2 5
.007812500 = 2,1,1,1,1,1,1     1 1 6
.007812500 = 2,1,1,1,1,1,2     0 2 5
.003906250 = 1,1,1,1,1,1,1,1   1 0 8
.003906250 = 1,1,1,1,1,1,1,2   0 1 7
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 66.8%
```

```
Dispense     10 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.031250000 = 1,2,2,2,2         1 4 1
.031250000 = 2,1,2,2,2         1 4 1
.031250000 = 2,2,1,2,2         1 4 1
.031250000 = 2,2,2,1,2         1 4 1
.031250000 = 2,2,2,2,1         1 4 1
.031250000 = 2,2,2,2,2         0 5 0
.015625000 = 1,1,1,2,2,2       1 3 3
.015625000 = 1,1,2,1,2,2       1 3 3
.015625000 = 1,1,2,2,1,2       1 3 3
.015625000 = 1,1,2,2,2,1       1 3 3
.015625000 = 1,1,2,2,2,2       0 4 2
.015625000 = 1,2,1,1,2,2       1 3 3
.015625000 = 1,2,1,2,1,2       1 3 3
.015625000 = 1,2,1,2,2,1       1 3 3
.015625000 = 1,2,1,2,2,2       0 4 2
.015625000 = 1,2,2,1,1,2       1 3 3
.015625000 = 1,2,2,1,2,1       1 3 3
.015625000 = 1,2,2,1,2,2       0 4 2
.015625000 = 1,2,2,2,1,1       1 3 3
.015625000 = 1,2,2,2,1,2       0 4 2
.015625000 = 2,1,1,1,2,2       1 3 3
.015625000 = 2,1,1,2,1,2       1 3 3
.015625000 = 2,1,1,2,2,1       1 3 3
.015625000 = 2,1,1,2,2,2       0 4 2
.015625000 = 2,1,2,1,1,2       1 3 3
.015625000 = 2,1,2,1,2,1       1 3 3
.015625000 = 2,1,2,1,2,2       0 4 2
.015625000 = 2,1,2,2,1,1       1 3 3
.015625000 = 2,1,2,2,1,2       0 4 2
.015625000 = 2,2,1,1,1,2       1 3 3
.015625000 = 2,2,1,1,2,1       1 3 3
.015625000 = 2,2,1,1,2,2       0 4 2
.015625000 = 2,2,1,2,1,1       1 3 3
.015625000 = 2,2,1,2,1,2       0 4 2
.015625000 = 2,2,2,1,1,1       1 3 3
.015625000 = 2,2,2,1,1,2       0 4 2
.007812500 = 1,1,1,1,1,2,2     1 2 5
.007812500 = 1,1,1,1,2,1,2     1 2 5
.007812500 = 1,1,1,1,2,2,1     1 2 5
.007812500 = 1,1,1,1,2,2,2     0 3 4
.007812500 = 1,1,1,2,1,1,2     1 2 5
.007812500 = 1,1,1,2,1,2,1     1 2 5
.007812500 = 1,1,1,2,1,2,2     0 3 4
.007812500 = 1,1,1,2,2,1,1     1 2 5
.007812500 = 1,1,1,2,2,1,2     0 3 4
.007812500 = 1,1,2,1,1,1,2     1 2 5
.007812500 = 1,1,2,1,1,2,1     1 2 5
.007812500 = 1,1,2,1,1,2,2     0 3 4
.007812500 = 1,1,2,1,2,1,1     1 2 5
.007812500 = 1,1,2,1,2,1,2     0 3 4
.007812500 = 1,1,2,2,1,1,1     1 2 5
.007812500 = 1,1,2,2,1,1,2     0 3 4
.007812500 = 1,2,1,1,1,1,2     1 2 5
.007812500 = 1,2,1,1,1,2,1     1 2 5
.007812500 = 1,2,1,1,1,2,2     0 3 4
.007812500 = 1,2,1,1,2,1,1     1 2 5
.007812500 = 1,2,1,1,2,1,2     0 3 4
.007812500 = 1,2,1,2,1,1,1     1 2 5
.007812500 = 1,2,1,2,1,1,2     0 3 4
.007812500 = 1,2,2,1,1,1,1     1 2 5
.007812500 = 1,2,2,1,1,1,2     0 3 4
.007812500 = 2,1,1,1,1,1,2     1 2 5
.007812500 = 2,1,1,1,1,2,1     1 2 5
.007812500 = 2,1,1,1,1,2,2     0 3 4
.007812500 = 2,1,1,1,2,1,1     1 2 5
.007812500 = 2,1,1,1,2,1,2     0 3 4
.007812500 = 2,1,1,2,1,1,1     1 2 5
.007812500 = 2,1,1,2,1,1,2     0 3 4
.007812500 = 2,1,2,1,1,1,1     1 2 5
.007812500 = 2,1,2,1,1,1,2     0 3 4
.007812500 = 2,2,1,1,1,1,1     1 2 5
.007812500 = 2,2,1,1,1,1,2     0 3 4
.003906250 = 1,1,1,1,1,1,1,2   1 1 7
```

```
.003906250 = 1,1,1,1,1,1,2,1   1 1 7
.003906250 = 1,1,1,1,1,1,2,2   0 2 6
.003906250 = 1,1,1,1,1,2,1,1   1 1 7
.003906250 = 1,1,1,1,1,2,1,2   0 2 6
.003906250 = 1,1,1,1,2,1,1,1   1 1 7
.003906250 = 1,1,1,1,2,1,1,2   0 2 6
.003906250 = 1,1,1,2,1,1,1,1   1 1 7
.003906250 = 1,1,1,2,1,1,1,2   0 2 6
.003906250 = 1,1,2,1,1,1,1,1   1 1 7
.003906250 = 1,1,2,1,1,1,1,2   0 2 6
.003906250 = 1,2,1,1,1,1,1,1   1 1 7
.003906250 = 1,2,1,1,1,1,1,2   0 2 6
.003906250 = 2,1,1,1,1,1,1,1   1 1 7
.003906250 = 2,1,1,1,1,1,1,2   0 2 6
.001953125 = 1,1,1,1,1,1,1,1,1 1 0 9
.001953125 = 1,1,1,1,1,1,1,1,2 0 1 8
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
Prob. of using other hopper = 66.6%
```

24.2. Combinations

The listing below shows the probability of certain coin combinations being dispensed from a 2 Euro / 1 Euro SCH3A for dispense values between 1 and 10 Euros. It is assumed that the SCH3A contains an equal number of each coin type.

2 = No. of 2 Euro coins dispensed

1 = No. of 1 Euro coins dispensed

The ratio of 2 Euro to 1 Euro payout is calculated. If unity then the mix of 2 Euros to 1 Euro coins in the hopper will tend to remain the same.

```
Coin 1 Frequency = 50.0%, Value = 100
Coin 2 Frequency = 50.0%, Value = 200

Dispense     1 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
1.00000000 =                   x 0 0
Probability sum = 1.000

Dispense     2 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.500000000 = (1E)x1            x 0 1
.500000000 = (2E)x1            x 1 0
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     3 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.500000000 = (2E)x1            x 1 0
.250000000 = (1E)x2            x 0 2
.250000000 = (2E)x1 (1E)x1     x 1 1
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     4 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.500000000 = (2E)x1 (1E)x1     x 1 1
.250000000 = (2E)x2            x 2 0
.125000000 = (1E)x3            x 0 3
.125000000 = (2E)x1 (1E)x2     x 1 2
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     5 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.375000000 = (2E)x1 (1E)x2     x 1 2
```

```
.250000000 = (2E)x2 (1E)x1     x 2 1
.250000000 = (2E)x2            x 2 0
.062500000 = (1E)x4            x 0 4
.062500000 = (2E)x1 (1E)x3     x 1 3
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     6 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.375000000 = (2E)x2 (1E)x1     x 2 1
.250000000 = (2E)x1 (1E)x3     x 1 3
.187500000 = (2E)x2 (1E)x2     x 2 2
.125000000 = (2E)x3            x 3 0
.031250000 = (1E)x5            x 0 5
.031250000 = (2E)x1 (1E)x4     x 1 4
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     7 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.375000000 = (2E)x2 (1E)x2     x 2 2
.187500000 = (2E)x3 (1E)x1     x 3 1
.156250000 = (2E)x1 (1E)x4     x 1 4
.125000000 = (2E)x2 (1E)x3     x 2 3
.125000000 = (2E)x3            x 3 0
.015625000 = (1E)x6            x 0 6
.015625000 = (2E)x1 (1E)x5     x 1 5
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     8 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.312500000 = (2E)x2 (1E)x3     x 2 3
.250000000 = (2E)x3 (1E)x1     x 3 1
.187500000 = (2E)x3 (1E)x2     x 3 2
.093750000 = (2E)x1 (1E)x5     x 1 5
.078125000 = (2E)x2 (1E)x4     x 2 4
.062500000 = (2E)x4            x 4 0
.007812500 = (1E)x7            x 0 7
.007812500 = (2E)x1 (1E)x6     x 1 6
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     9 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.312500000 = (2E)x3 (1E)x2     x 3 2
.234375000 = (2E)x2 (1E)x4     x 2 4
.156250000 = (2E)x3 (1E)x3     x 3 3
.125000000 = (2E)x4 (1E)x1     x 4 1
.062500000 = (2E)x4            x 4 0
.054687500 = (2E)x1 (1E)x6     x 1 6
.046875000 = (2E)x2 (1E)x5     x 2 5
.003906250 = (1E)x8            x 0 8
.003906250 = (2E)x1 (1E)x7     x 1 7
Probability sum = 1.000
Ratio of 2/1 payout = 1.000

Dispense     10 Euro
Prob.        Sequence          R 2 1
----------   ----------------- - - -
.312500000 = (2E)x3 (1E)x3     x 3 3
.164062500 = (2E)x2 (1E)x5     x 2 5
.156250000 = (2E)x4 (1E)x2     x 4 2
.156250000 = (2E)x4 (1E)x1     x 4 1
.117187500 = (2E)x3 (1E)x4     x 3 4
.031250000 = (2E)x1 (1E)x7     x 1 7
.031250000 = (2E)x5            x 5 0
.027343750 = (2E)x2 (1E)x6     x 2 6
.001953125 = (1E)x9            x 0 9
.001953125 = (2E)x1 (1E)x8     x 1 8
Probability sum = 1.000
Ratio of 2/1 payout = 1.000
```

25. Appendix G - Firmware Upgrading

Firmware upgrades are performed through a separate flash programming connector rather than with cctalk.

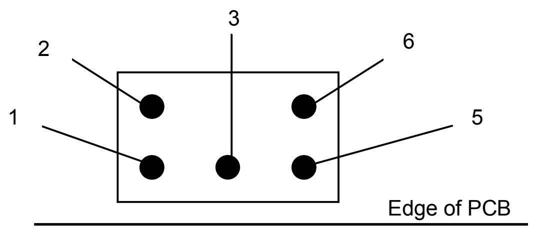
The SCH3A microcontroller is a Microchip PIC16F876A and the 8K program memory can be programmed with a standard ICSP interface.

25.1. Flash Connector Pinout

Pin	Label	Connect to…
1	PGD	ICSP PGD
2	PGC	ICSP PGC
3	Vpp	+13V
4	-	-
5	GND	0V
6	Vcc	+5V

The connector is located on the lower PCB.

25.2. Flash Connector Polarity



26. Appendix H - Opto Logic Table

This table shows the possible opto responses in each hopper state (idle and payout) together with their associated error codes.

In Idle Loop

Action	Transmit Sensor (LED on)	Reflective Sensor (LED on)	Transmit Sensor (LED off)	Reflective Sensor (LED off)
Block exit with shiny object	Fraud α	Fraud β	-	-
Block exit with dull object	Fraud α	-	-	-
Shine light into exit sensor	-	Fraud β	Fraud β	Fraud β

Fraud α : Opto fraud attempt, path blocked during idle
Fraud β : Opto fraud attempt, short-circuit during idle

During Payout

Action	Transmit Sensor (LED on)	Reflective Sensor (LED on)	Transmit Sensor (LED off)	Reflective Sensor (LED off)
Block exit with shiny object	Coin (Timeout)	Coin (Timeout)	-	-
Block exit with dull object	Coin (Timeout)	-	-	-
Shine light into exit sensor	-	Coin (Timeout)	Fraud γ	Fraud γ

Timeout : Opto blocked permanently during payout
Fraud γ : Opto fraud attempt, short-circuit during payout

27. Appendix I - PIN Number Recovery

If you program a PIN number with the ‘Enter new PIN number’ command and subsequently forget the PIN number, there is no way to dispense coins.

There is a hardware recovery mechanism built into the hopper which allows the PIN number to be disabled which is the factory default condition. To do this the PCB cover must be removed (held in place by a single screw) and the top PCB unplugged and turned over. There is a ‘zero ohm link’ resistor which must be removed from the circuit (or slid off one of the pads) so that when power is re-applied the hopper eneters a special state and clears the PIN number. To do this you will need a soldering iron. In this special state the hopper will NAK all serial commands.

To return the hopper to the normal operating state the zero ohm link must be re-soldered across the pads. When power is re-applied the ‘PIN number mechanism flag’ should be clear.

See **Figure 1** for link position.

Figure 1 - Link Position