MATRIX

MIAC

Module communications

www.matrixmultimedia.com

MIAC System module communications

This document outlines how a MIAC System communicates between the MIAC Master and the peripheral expansion modules using the CAN bus. Hence this document is essential to system builders not intending to use Flowcode, which masks the complexity, or those wishing to integrate or amend the system.

The following section details the standard factory default CAN configuration. For those wishing to use a MIAC System on an existing CAN bus, the CAN parameters can be re-configured. Please see the document "MIAC System CAN Customisation" for further details.

Module CAN ID

11 Bit Standard Identifier

B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
Ordinal		Module type								
O1	O0	T4	T3	T2	T1	T0	0	0	0	0

The base ID of each module is calculated from the module type and the ordinal (0-3) of the corresponding module component in the Flowcode flowchart.

Module type values

1 = Basic
2 = Advanced
3 = Serial
4 = Industrial
5 = Bluetooth
6 = GPS
7 = (future use)
8 = GSM
9 = ZigBee coordinator
10 = ZigBee router
11 = Base
12 = MIAC Slave
13 = (future use)
14 = (future use)
15 = MIAC Master

Base ID calculation

Base ID = (ordinal <<9) + (type <<4)

Example

A system consisting of a Basic module (ordinal = 0) and two serial modules (ordinals = 0, 1)

Basic(0) = (0<<9) + (1<<4) = 0x010
Serial(0) = (0<<9) + (3<<4) = 0x030
Serial(1) = (1<<9) + (3<<4) = 0x230

Hardware address

The address jumpers on the motherboard of each expansion module (ADD1:ADD0) must be set to match the ordinal of the corresponding module component in the Flowcode flowchart (default value is 0).

When a MIAC is used as a slave, its ordinal address is setup by holding the Menu (Green) key whilst the unit powers up, or resets. The value set is persisted in internal EEPROM.

MIAC System module communications

CAN Communications

Messages
Each individual system CAN messge consists of 2 to 8 data bytes.
The first two bytes represent the command ID (11-bit) and status/control flags.
Up to 6 bytes are available are available for data.

Byte 0: Cmd_H + Flag
Byte 1: Cmd_L
Byte 2: Data 0 <optional>
Byte 3: Data 1 <optional>
Byte 4: Data 2 <optional>
Byte 5: Data 3 <optional>
Byte 6: Data 4 <optional>
Byte 7: Data 5 <optional>

Transactions
A transaction consists of 1 or more messages, allowing for commands with more than 6 data bytes.

Command messages sent by the master MIAC have the following format:

Byte 0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	MORE	START	Cmd: 10	Cmd: 9	Cmd: 8

Byte 1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Cmd: 7	Cmd: 6	Cmd: 5	Cmd: 4	Cmd: 3	Cmd: 2	Cmd: 1	Cmd: 0

START Flag indicating the first message in a command transaction.
MORE Flag indicating further data to be transmitted in following message.
Cmd: 10-0 11-bit command ID

Acknowledge messages sent by the modules have the same format as the command messages.

Byte 0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	MORE	START	Cmd: 10	Cmd: 9	Cmd: 8

Byte 1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Cmd: 7	Cmd: 6	Cmd: 5	Cmd: 4	Cmd: 3	Cmd: 2	Cmd: 1	Cmd: 0

START Flag indicating the first message in a command transaction.
MORE Flag indicating further data to be transmitted in following message.
Cmd: 10-0 11-bit command ID (used to confirm received ID)

MIAC System module communications

Each individual CAN message transmitted by the master MIAC should be matched by an acknowledge message from the appropriate module.

Acknowledge messages in the body of a multi-message command transaction (MORE flag = 1) contain only a copy of the message ID. This is used to confirm reception and maintain timing. The final message in the transaction (MORE flag = 0) causes the command to be executed. Data may be returned in the resulting, and any subsequent, acknowledge messages. All the data associated with a transaction (single or multi-message) is buffered in a 255 byte array in the receiving module until the transaction is complete and the command can be executed.

If multiple acknowledge messages are required after execution of a command (more than 6 bytes of return data). The master MIAC must request each message, by repeatedly transmitting the command message, with no data, based on the status of the MORE flag in the previous acknowledge message.

MIAC system functions
Several functions are included in the PIC_CAL_MIAC_SYSTEM.c file.

Functions derived from the standard Flowcode CAN component (some modifications to improve buffering)

```
void SPI_Init(void);
char SPI_SendByte(char);
char CAN_MaskFromBit(char);
void CAN_Reset(void);
void CAN_Write(char, char);
char CAN_Read(char);
void CAN_Config(char);
char CAN_ReadStatus(void);
char CAN_ReadRXStatus(void);
void CAN_BitModify(char, char, char);
char CAN_ReadRX_ID_and_Data(char);
void CAN_Init(void);
void CAN_SendBuffer(char);
char CAN_CheckRx(char);
char CAN_GetRxDataCount(char);
char CAN_GetRxData(char, char);
void CAN_SetTxData(char, char, char, char, char, char, char, char, char, char);
void CAN_SetTxID(char, unsigned short);
char CAN_GetRxIDLo(char);
char CAN_GetRxIDHi(char);
```

New CAN functions to clear blocked transmit messages and re-program the reception filters

```
void CAN_CancelTx(char);
void CAN_SetFilter (unsigned short);
```

MIAC system communications functions

```
void MIAC_Module_WR(unsigned short, unsigned short, char, char, char);
char MIAC_Module_RD(unsigned short, unsigned short);
char MIAC_Module_GetAck(unsigned short, unsigned short, unsigned short);
```

Copy of MIAC functions (allows calls from other components - original MIAC functions not always loaded)

```
void MIAC_Master_RelayON(char);
void MIAC_Master_RelayOFF(char);
void MIAC_Master_OutputON(char);
void MIAC_Master_OutputOFF(char);
char MIAC_Master_InputDIGITAL(char);
char MIAC_Master_InputANA(char);
short MIAC_Master_InputANA10(char);
```

MIAC System module communications

MIAC_Module_WR() and MIAC_Module_GetAck() are the functions used by the Flowcode components to access the system modules.

void **MIAC_Module_WR**(unsigned short, unsigned short, char, char, char);

unsigned short	Module ID
unsigned short	Command ID
char	Number of data bytes (0-6 : excluding 2-byte command ID)
char	START flag
char	MORE flag

Compiles and transmits an individual Command message

---

char **MIAC_Module_RD**(unsigned short, unsigned short);

unsigned short	Module ID
unsigned short	Command ID

Checks for an incoming acknowledge message to match the transmitted CAN and command IDs.

Return value (Bits7:5)

111 = no message
110 = incorrect CAN ID
101 = insufficient data (<2 bytes – command ID)
100 = incorrect command ID

000 = message received
Bit 4 = MORE flag
Bit 3 = START flag
Bit 2:0 = Number of data bytes (0-6 : excluding 2-byte command ID)

---

char **MIAC_Module_GetAck(**unsigned short, unsigned short, unsigned short);

unsigned short	Module ID
unsigned short	Command ID
unsigned short	Time-out period (x200us)

Repeats calls to MIAC_Module_RD until a message is received, or the time-out period has elapsed.

Return value (Bits7:5)

111 = no message
110 = incorrect CAN ID
101 = insufficient data (<2 bytes – command ID)
100 = incorrect command ID

000 = message received
Bit 4 = MORE flag
Bit 3 = START flag
Bit 2:0 = Number of data bytes (0-6 : excluding 2-byte command ID)

MIAC System module communications

Buffering and conversion
Two 6-byte buffers are declared in PIC_CAL_MIAC_SYSTEM.c to handle the transmit and receive data:

```
char MIAC_CAN_TX[6];
char MIAC_CAN_RX[6];
```

Each module reserves a 255 byte array to allow multi-message transactions (transmit or receive) to be buffered independent of the CAN messaging.

A union is also declared to allow floating-point values to be transmitted and received as 4-btyes:

```
union Float4Bytes
{
        float FloatVal;
        char ByteArray[4];
};
union Float4Bytes FloatConv;
```

Module functions/macros

The expansion modules themselves have function macros that are called from a Flowcode flowchart and these interface with the module hardware via the CAN bus.

The following two function macros are available for all modules:

void Initialise(void)
This macro uses the corresponding module's interrupt CAN ID and forces a software reset (Command = 0).

void LED_Control(char State)
This macro is used to control the module's status LED.

Some modules have additional function macros, particularly where they have features that are not otherwise available via the attachment standard Flowcode component directory:

void Write_DAC_1Channel(char Channel, short Value)
Write a 12-bit value to the selected DAC channel

void Write_DAC_2Channel(short ChannelA, short ChannelB)
Write 12-bit values to both DAC channels simultaneously

short Sensor_Read(char Sensor, char HiRes)
Read the selected sensor channel with either 8-bit or 10-bit resolution

void RTC_Get_Data(void)
Read 6 bytes (Hr, Min, Sec, Day, Mth, Yr) from the RTC into an internal buffer

char RTC_Read_Time(char HMS)
Read the selected time value from the buffer (0=Hr, 1= Min, 2=Sec)

char RTC_Read_Date(char DMY)
Read the selected date value from the buffer (0=Day, 1= Mth, 2=Yr)

void RTC_Set_Time(char Hour, char Min, char Sec)
Set RTC time

void RTC_Set_Date(char Day, char Month, char Year)
Set RTC date

char RTC_Reg_Read(char Reg)
Direct read of selected RTC register (additional functions - expert mode)

void RTC_Reg_Write(char Reg, char Value)
Direct write to selected RTC register (additional functions - expert mode)

Note: The RTC is fixed in 24hr mode. All values in binary (bcd conversion done in the module)

Command IDs, parameters and returns

As discussed earlier, the functionality of each expansion module is accessed via CAN messages. Within the message data field is a unique code indicating the function to be implemented. This code is the Command ID. Not all modules support all Command IDs.

This section summarises the data sent with each Command, and the data that is returned as a reply to the Command by the addressed Expansion Module. For detailed examples of use please see the MIAC System components source files, FC5_MIAC_xxxx.c and FC5_PIC_xxxx_MS.c

The Command IDs are grouped by relevant title and list the Command ID, followed by the Command name which briefly describes the command. Parameters sent with the command are listed within brackets as a comma separated list and any returned data is identified after "Return".

For example:

Sensor (Advanced only)

160	Read channel	(channel) : Return (value_H, value_L)

160 is the Command ID for reading the Sensor inputs of the Advanced Module.

The command requires one data byte to be sent which defines the Sensor to be read (1 or 2), this is sent as the first byte of the transmit data buffer, namely, MIAC_CAN_TX[0]

The reply received will contain two relevant bytes in the receive buffer, the first byte MIAC_CAN_RX[0] will contain the high byte of the result, and the second byte MIAC_CAN_RX[1] will contain the low byte of the result, from which the full 10 bit conversion value can be re-constructed.

MIAC Slave

64	InputDIGITAL	(input) : Return (state)
65	InputANALOG	(input) : Return (value)
66	GetKeypad	Return (key)
67	RelayON	(relay)
68	RelayOFF	(relay)
69	OutputON	(output)
70	OutputOFF	(output)
71	Start(LCD)	
72	Clear(LCD)	
73	Cursor	(x, y)
74	PrintASCII	(char)
75	PrintNumber	(number_H, number_L)
76	PrintString	Potential multi-message transaction
77	Command(LCD)	
78	InputANALOG_10bit	(input) : Return (value_H, value_L)

System (Applicable to all Modules)

128	Set LED state	(state)
129	Respond	No action, ping function, generates ack message
131	Read Firmware Version	Return (value_H, value_L)
132	Software reset	
133	Read module type	Return (Type, Ordinal)
134	Customise CAN	(see CAN customisation document)
135	Factory defaults	

Sensor (Advanced only)

160	Read channel	(channel) : Return (value_H, value_L)

RTC (Advanced only)

192	Set time	(H, M, S)
193	Set date	(D, M, Y)
194	Read data	Return (H, M, S, D, M, Y)
195	Read register	(register) : Return value – access to extra functions
196	Write register	(register, value) – access to extra functions

DACs (Advanced only)

224	Write 1 channel	(channel, value_H, value_L)
225	Write both channels	(ch1_H, ch1_L, ch2_H, ch2_L) – synchronised updates

I/O

256	LED/Write	(terminal, state, polarity) nb. Polarity 0 inverts state
257	Switch/Read	(terminal) : Return state
258	Wait until high	(terminal) – waits indefinitely for ack
259	Wait until low	(terminal) – waits indefinitely for ack

ADC

289	Read terminal	(terminal) : Return (value_H, value_L)

PWM

320	Enable	(channel= 1,2, period= 0-255, prescaler=4,5,6 equates to 1,4,16)
321	Disable	(channel= 1,2)
322	Set duty cycle	(channel= 1,2, duty= 0-255)
323	Change period	([0]period= 0-255, [1]prescaler= 1,4,16, [2]not used)
324	Set duty cycle 10-bit	(channel= 1,2, duty_H, duty_L)

SPI1 (SPI1 – External)

352	Init	(prescaler, ckp, cke, smp)
353	Uninit	
354	Send char	(char)
355	Get char	Return (char)

SPI2 (SPI2 – Internal) GPS and Serial only (SD CARD)

360	Init	(prescaler, ckp, cke, smp)
361	Chip Select	(state)
362	Rotate Byte	(byte in): Return (byte out)

I2C

384	Init	(smbus, slew, baud_H, baud_L)
385	Start	
386	Re-start	
387	Stop	
388	Tx byte	(byte) : Return (ack)
389	Rx byte	(last) : Return (byte)
390	Tx transaction	(device ID, addr_H, addr_L, data)
391	Rx transaction	(device ID, addr_H, addr_L) : Return (data)

UART1

416	Init	(txsta, spbrg, data size, config flags, rts terminal, cts terminal)
417	Send byte	(byte)
418	Send string	Potential multi-message transaction
419	Receive byte	(timeout) : Return (Hi byte, Lo byte)
420	Receive string	Return potential multi-message transaction

UART2

448	Init	(txsta, spbrg, data size, config flags, rts terminal, cts terminal)
449	Send byte	(byte)
450	Send string	Potential multi-message transaction
451	Receive wide char	(timeout) : Return (Hi byte, Lo byte)
452	Receive string	Return - potential multi-message transaction

Nb. Following implemented on Industrial Module only, for 9 bit RS485

453	Send wide char	(Hi byte, Lo byte)

Zigbee

480	Init	(hw flow control)
481	Send char	(char)
482	Receive char	(timeout) : Return (char) nb. char = 255 if timed out
483	Send AT command	Potential multi-message transaction
484	Sleep	
485	Wake	
486	Set RTS	(0=RTS low)
487	Get CTS	Return (byte) CTS state
488	Initialise Interrupt Rx Buffer	(0=Off, else On)
489	Get Count	Return (byte) Rx buffer character count
490	Get Char	Return (byte) character from FIFO buffer

Stepper

512	Output	(coil1, coil2, coil3, coil4, pattern)
513	Disable	(coil1, coil2, coil3, coil4)

Keypad

576	Init	(col1, col2, col3, col4, row1, row2) + (row3, row4, columns, rows)
577	Get number	Return (keynumber) – final conversion in master MIAC

Servo

640	Init	(ch0, ch1, ch2, ch3, ch4, ch5) (ch6, ch7, trim0, trim1, trim2, trim3) (trim4, trim5, trim6, trim7, channels)
641	Enable	(channel)
642	Disable	(channel)
643	Set position	(channel, position)
644	Set trim	(channel, trim)
645	Move to pos.	(channel, position)
646	Auto move to pos	(channel, position)

Bluetooth

672	Init	(hw flow control)
673	Send char	(char)
674	Read char	(time-out) : Return (char) + update %a_BLU_RX_STATUS
675	Rx Interrupt Buffer	(enable/disable)
676	Read Count	Return (count of chars in rx buffer)
677	Read Char	Return (char from buffer and decrement count)

GPS

704	Init	(spbrg, echo)
705	Is data valid	Return (valid)
706	Read long.	Return (longitude - 4 byte array as float union)
707	Read lat.	Return (latitude - 4 byte array as float union)
708	Read alt.	Return (altitude - 4 byte array as float union)
709	Read speed	Return (speed - 4 byte array as float union)
710	Read course	Return (course - 4 byte array as float union)
711	Read sats.	Return (satellites)
712	Read UTC time	(unit) : Return (time unit)
713	Read UTC date	(unit) : Return (date unit
714	Read DOP	Return (DOP - 4 byte array as float union)
715	Read HDOP	Return (HDOP - 4 byte array as float union)
716	Read VDOP	Return (VDOP - 4 byte array as float union)

FAT16

736	Init	
737	Open file	Potential multi-message transaction : Return (byte)
738	Read file length	Return (length_H, length_L)
739	Read file sector	Return (byte)
740	To next sector	(force) : Return (byte)
741	Append string	Potential multi-message transaction : Return (byte)
742	Write file sector	Return (byte)
743	Create file	Potential multi-message transaction : Return (byte)
744	Delete file	Potential multi-message transaction : Return (byte)
745	Write to buffer	(addr_H, addr_L, byte)
746	Read from buffer	(addr_H, addr_L) : Return (byte)
747	Open folder	Potential multi-message transaction : Return (byte)
748	Scan folder	(idx_H, idx_L, scantype) : Return (byte)
749	Read from scan	(idx) : Return (byte)
750	Component init	

GSM

780	Initialise	
781	Set START (G8)	(byte) 0=low, else high
782	Set RTS (G14)	(byte) 0=low, else high
783	Set RESET (G7)	(byte) 0=low, else high
784	Get PORTG	Return (high byte, low byte)
785	Get Count	Return (byte) Rx buffer character count
786	Get Char	Return (byte) character from FIFO buffer
787	Send Char	(byte)

INTERNET INDUSTRIAL

800	Initialise	(BRG prescaler, CKP, CKE, SMP)
801	Write Byte	(add_h, add_l, data byte)
802	Read Byte	(add_h, add_l) Return byte
803	Write Short Word	(add_h, add_l, data_h, data_l)
804	Read Short Word	(add_h, add_l) Return data_h, data_l
805	Read Interrupt (G12)	Return State

Matrix Multimedia Ltd.
23 Emscote Street South
Halifax
HX1 3AN

t: +44 (0)1422 252380
e: sales@matrixmultimedia.co.uk

www.matrixmultimedia.com