LAN I2C Adapter VM

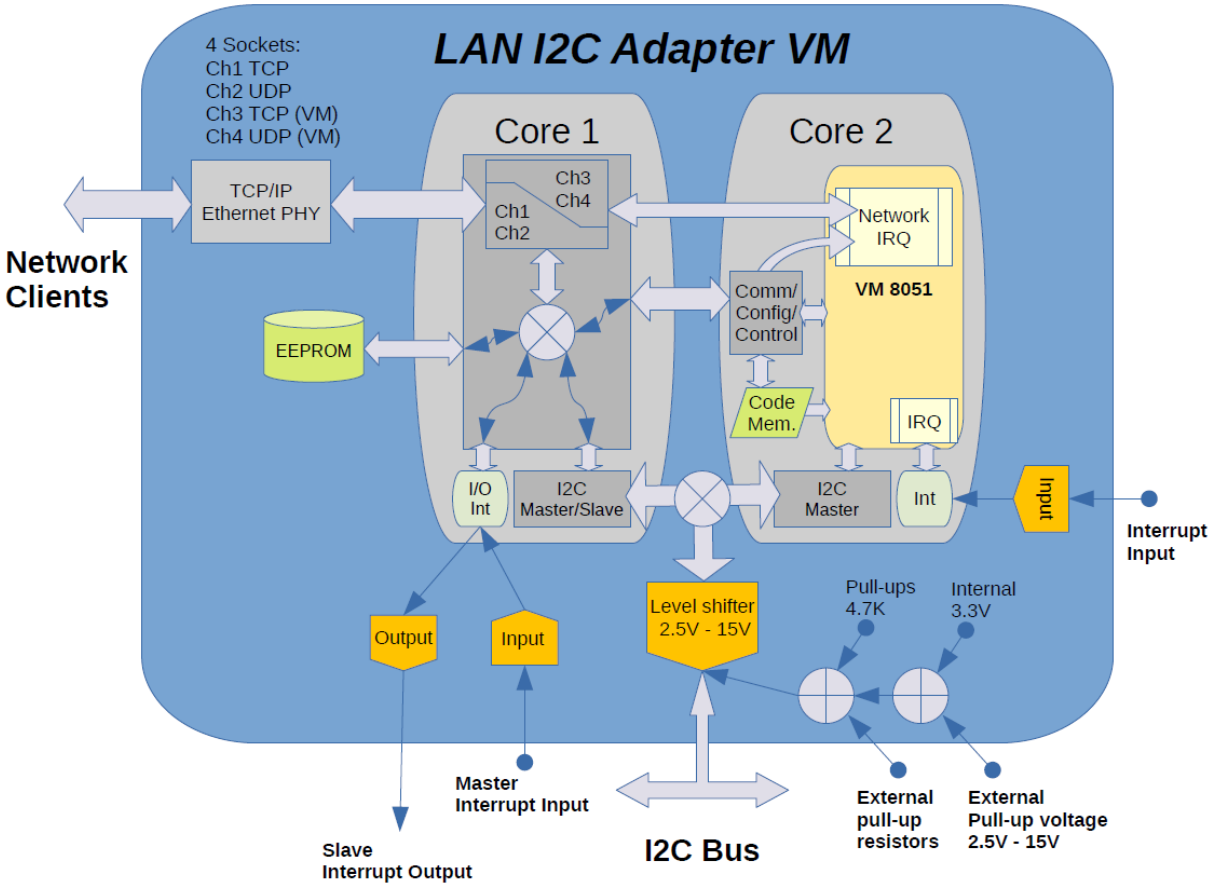
Data sheet

(Rev 1.0)

COPTONIX

Falkentaler Steig 9
D – 13467 Berlin
Phone: +49 30 617 412 48
Fax: +49 30 617 412 47
www.coptonix.com

1. General description



The Lan I2C Adapter VM is a versatile programmable Ethernet to I2C adapter with an adjustable I2C frequency up to 400 kHz. The adapter provides system designers a quick and easy way to add Ethernet networking capabilities to any device with an I2C interface. Implementing this adapter in a system allows Ethernet connectivity and standard protocol processing to be completely offloaded from the system, significantly reducing hardware, firmware and software development. It is therefore the first choice for Machine and Equipment Builders.

The core of the adapter is a dual core ARM 32 Bit microcontroller. Each core has an independent I2C interface, however both interfaces are connected to the same I2C bus. While the I2C interface in Core 1 can be I2C master or slave, the I2C interface in Core 2 is I2C master only.

Core 1 serves as a classic out-of-the-box Ethernet-to-I2C adapter with I2C master and slave functionality. The disclosed interface documentation allows the adapter to be implemented in target applications independent of the operating system. Whether Windows, Linux, real-time operating systems or others, the adapter is completely independent of the operating system and requires no drivers. However, a Windows DLL (Dynamic Link Library) is available to speed up the implementation of the adapter in applications running on Windows.

A virtual machine emulating an 8051 microcontroller is implemented in Core 2. Real-time, time-sensitive and repetitive processes can be run in the virtual machine. This also reduces network traffic, among many other benefits.

A simple case example, when it comes to monitoring a large number of I2C devices (e.g. flow, temperature and pressure sensors) then it is important to read the values in real time and to react to them as fast as possible. If required, status messages can be forwarded to the PC.

As a second case example, some I2C devices require certain registers to be constantly updated at short time intervals, even if the values are not changing. Such repetitive processes cause a lot of unnecessary network traffic. Running such processes in the virtual machine reduces network traffic to a minimum.

Each Core has an additional interrupt input needed to respond to external events without constantly polling connected I2C devices. For example, an IO expander uses an interrupt output to inform the master as soon as the status of the IOs has changed. The interrupt input can be activated (rising or falling edge) or deactivated by software. In I2C slave mode, the interrupt output is used. In this case, an external I2C master is notified as soon as the I2C slave output buffer is filled with new data. Switching between master mode and slave mode is done by software. Since the I2C interface in Core 2 can only be operated in I2C master mode, no interrupt output is available.

4.7k pull-up resistors are already on board. However, these can be disabled to use external resistors. The pull-up voltage can be set to either the internal 3.3V or to an external voltage in the range from 2.5V to 15V.

The lowest 256 bytes of the onboard 32K bytes EEPROM are reserved for storing internal settings. The remaining memory is available to the user. This memory area can be used to store any user specific information and/or VM 8051 application code.

The "IIC Device Control" software supports all I2C devices that comply with the I2C specification. Ready-to-use GUI for a variety of different I2C devices, such as Temperature, Pressure, Flow, ADC, DAC, IO-Expander, PWM, Audio, Digital Potentiometer, Fan, LED Blinker / Dimmer, Magnetometer, Multiplexers, Switchers, RTC / Calendar, Stepper Motor, Accelerometer and Gyroscope. The software supports EEPROMS of 1Kbit (128 bytes) to 1Mbit (128K bytes).

For more information on the disclosed software interface, please refer to the "*TCP/UDP I2C Interface Manual*". When using the Windows Dynamic Link Library (DLL) please refer to the "Interface Manual LANIICVM.DLL"

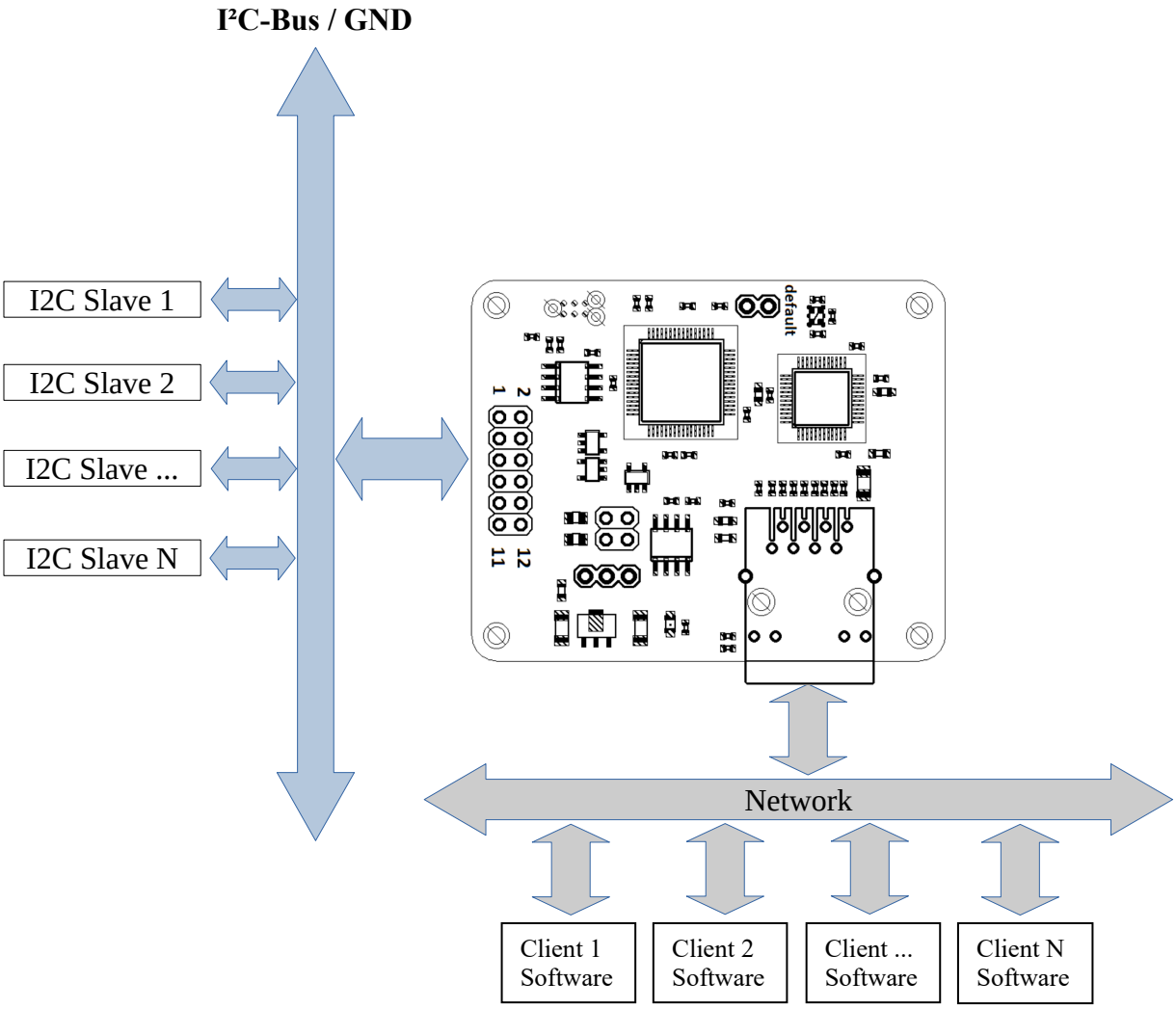
2. Features

- +5V supply voltage
- 4x independent Sockets: 2x TCP and 2x UDP
- Supports KEEP ALIVE packets for TCP sockets
- Supports network cable physical connection detection
- On board I2C level shifter, I2C levels from 2.5V to 15V
- Dual Core Microcontroller
  Core 1:
  → Ready-to-use, Plug & Play Ethernet to I2C adapter
  → 2x independent sockets: 1x TCP and 1x UDP
  → 1K bytes TCP and UDP packets
  → 1x interrupt input (external Slave → Master)
  → 1x interrupt output (Slave → external Master)
  → 1x independent I2C interface
  → I2C frequency 500Hz to 400kHz
  → Supports multi-master
  → Master transmit & receive
  → Slave transmit & receive
  → Supports clock streching
  → 7 bit addressing
  → 1K bytes per I2C WRITE and READ transaction

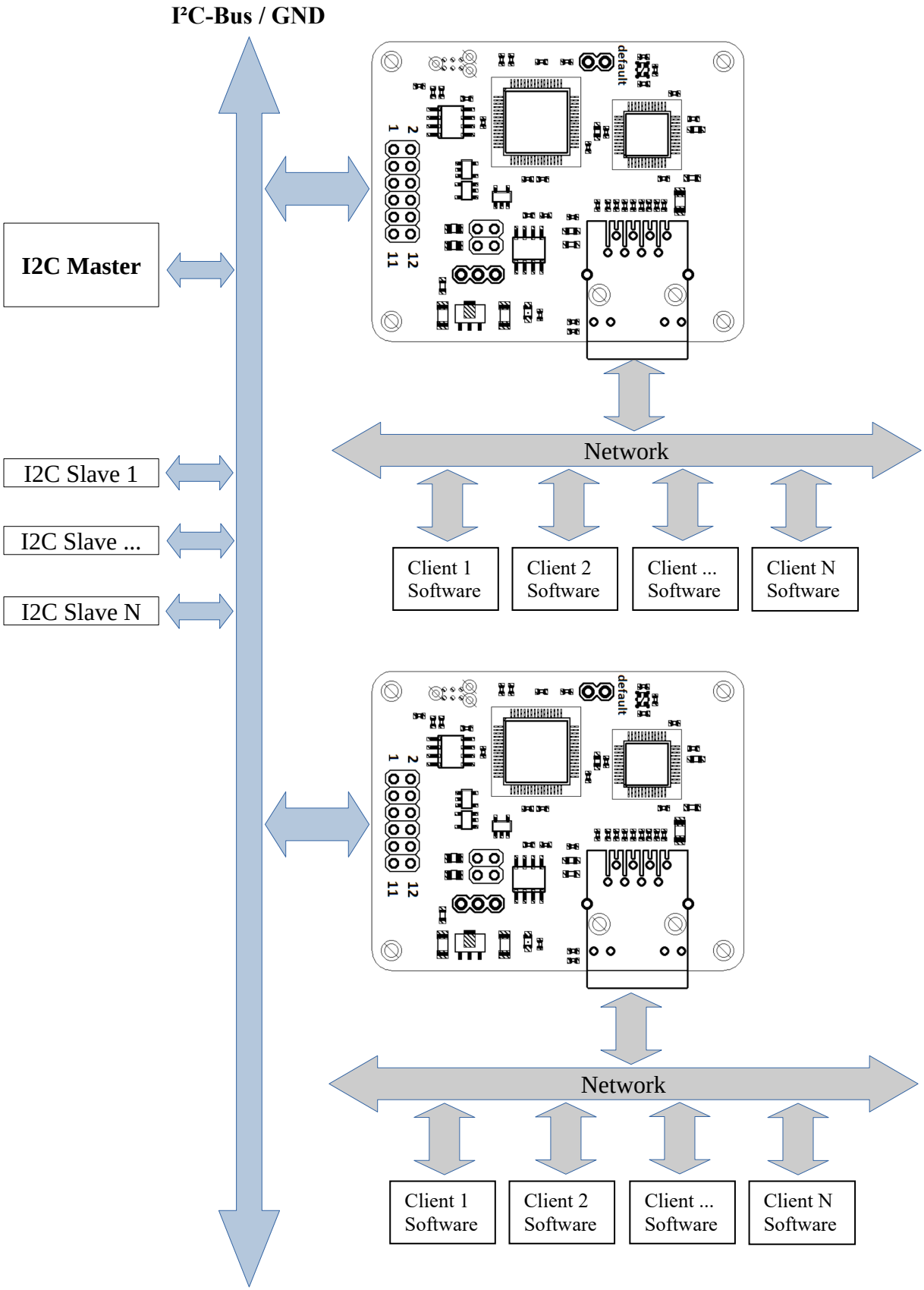
  Core 2:
  → Programmable Ethernet to I2C Adapter
  → 8051 Virtual Machine
  → 16K bytes program code memory
  → 256 bytes internal RAM
  → 3K bytes external RAM
  → 1x Timer
  → 2x sockets of Core 1
  → 2x additional independent sockets: 1x TCP and 1x UDP
  → 255 bytes TCP and UDP packets
  → 1x interrupt input (external Slave → Master)
  → 1x independent I2C interface
  → I2C frequency 5kHz to 400kHz
  → Supports multi-master
  → Master transmit & receive
  → Supports clock streching
  → 7 bit addressing
  → 255 bytes per I2C WRITE or READ transaction

- 32K bytes EEPROM
- Supports firmware updates
- 45x56 mm^2

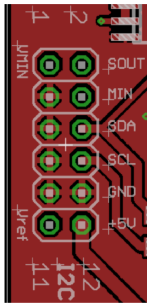
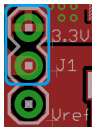
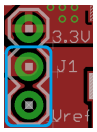
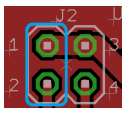
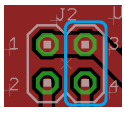
2.1 LAN I2C Adapter VM as Master



2.2 LAN I2C Adapter VM as Slave



3. Interfaces

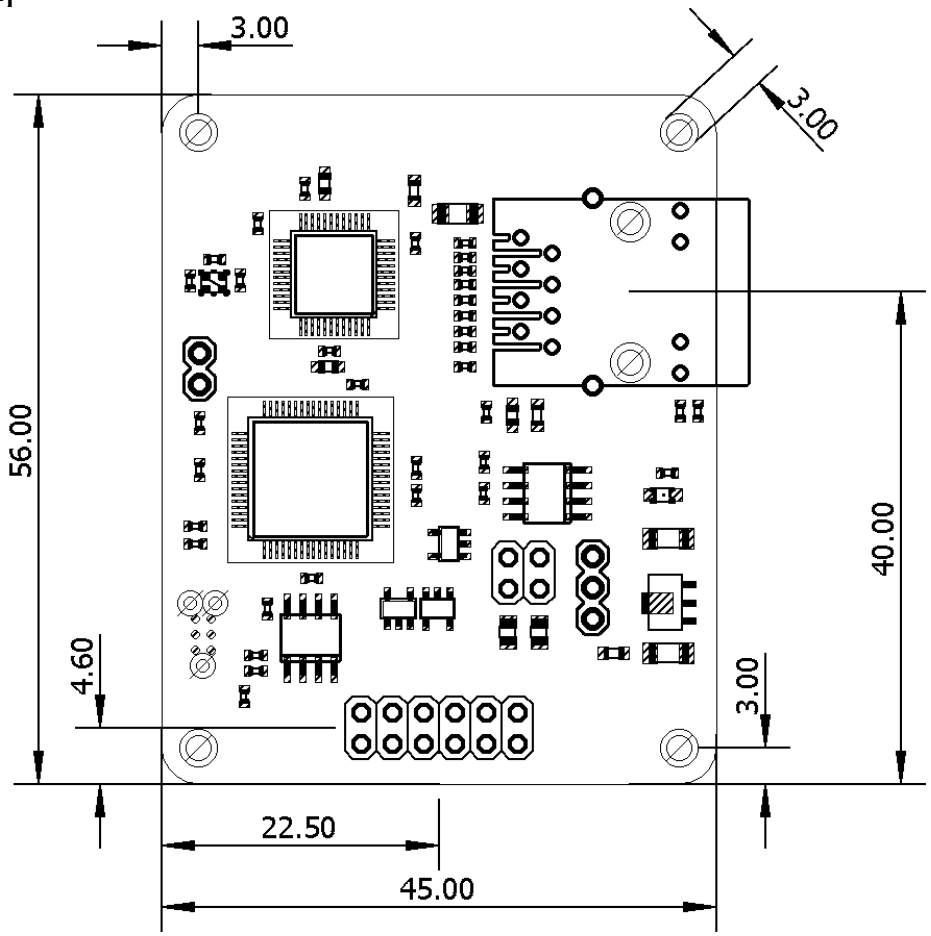
1 – MagJack		
Mag Jack (Transformer and RJ45)		10/100 Base-T Ethernet (Auto Detection) Protocol: TCP, UDP
2 – I2C Interface (12 pin header)		
3, 5, 7, 9, 10		Ground
1		VMIN: Virtual Machine Master Interrupt Input
2		SOUT: Slave Interrupt Output
4		MIN: Master Interrupt Input
6		I2C SDA
8		I2C SCL
11		Vref: Optional external pull-up voltage (2.5V – 15V)
12		+5V supply voltage
3 – J1 Pull-up voltage		
Position 1 - 2		Pull-up voltage set to internal voltage 3.3V
Position 2 - 3		Pull-up voltage set to external voltage 2.5V to 15V (Pin 14)
4 – J2 Pull-up resistors		
Position 1 – 2		Remove the jumper to disable internal pull-up resistor (4.7k) of SCL
Position 3 – 4		Remove the jumper to disable internal pull-up resistor (4.7k) of SDA
5 – J3 Network default settings		
Position 1 – 2		If set, default network settings are loaded on power on, else network settings are loaded from EEPROM.

4. Characteristics

	Min.	Typ	Max.	Unit
Power-Supply				
Supply Voltage		5.0		V
Supply Current		140		mA
I2C-Bus pins (SCL, SDA)				
V_{ext} External Pull-up Voltage	2.5	-	15	V
V_{IH} High-level Input Voltage	$0.55V_{pull-up}$	-	-	V
V_{IL} Low-level Input Voltage	-	-	$0.4V_{pull-up}$	V
Limiting values				
Interrupt pin				
Input Voltage	0	-	5.5	V
Output Voltage	0	-	$V_{DD(3,3V)}$	V
Power-Supply				
Supply Voltage	4.0	5.0	6.0	V
Temperature				
Operating temperature	0	-	+70	°C

5. Dimensions

Units [mm]



6. Virtual Machine 8051

6.1 Restrictions

The VM 8051 is compatible with MCS-51 architecture, and can be programmed using the MCS-51 instruction set. However, the standard MCS-51 peripherals such as Ports (P0, P1, P2, P3), standard Timers (TCON, TMOD) and Serial Port (SCON, SBUF) are not available. The VM 8051 is designed as a programmable Ethernet to I2C adapter, so new features have been added to allow easy programming of the I2C interface and network communication. Resource management and flow control for I2C and TCP/IP and UDP are already implemented, so that very in-depth programming knowledge of the I2C bus and Ethernet is not required.

6.2 Memory Organization

The VM 8051 has separate address spaces for program and data memory. The program memory has a regular linear address space with support for up to 16K bytes of directly addressable application code. The data memory has 256 bytes of internal RAM and 3K bytes of external RAM. The upper 1K bytes (4x 256 bytes) of external RAM are reserved for the I2C interface and network communication as input/output buffers, so that only 2K bytes of external RAM are available to the programmer.

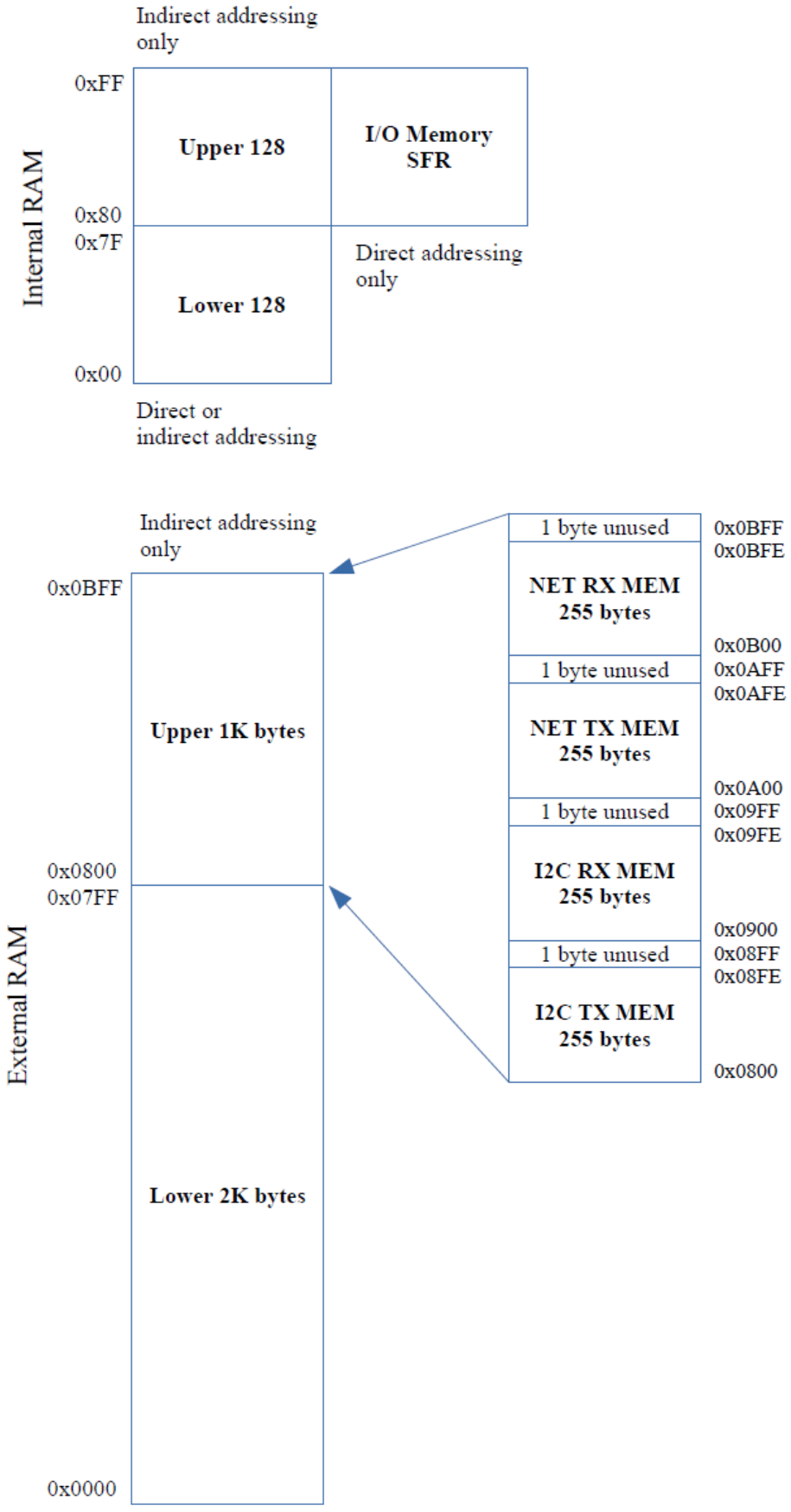
6.2.1 Program Memory

The program memory contains 16K bytes of RAM, which emulates the Flash Memory. Programming the program memory is done by software using the API (VM LOAD CODE). Optionally, the application code can be loaded from the on board EEPROM into program memory (VM LOAD FROM EEPROM). The LAN I2C Adapter VM has 32K bytes of on-board EEPROM memory for program storage. The reset and interrupt vectors are located within the first 67 bytes of program memory.

6.2.2 Data Memory

The VM 8051 contains 256 bytes of general internal RAM, 128 bytes of I/O memory and 3K bytes external RAM. The lower 128 bytes of internal RAM may be accessed through both direct and indirect addressing. The upper 128 bytes of internal RAM and the 128 bytes of I/O memory share the same address space. The upper 128 bytes of internal RAM may only be accessed using indirect addressing. The I/O memory can only be accessed through direct accessing and contains the Special Function Registers (SFRs). The lowest 32 bytes of data memory are grouped into 4 banks of 8 registers each. The RS0 and RS1 bits (PSW.3 and PSW.4) select which register bank is in use. Instructions using register addressing will only access the currently specified bank.

6.2.3 Data Memory Map



6.3 Interrupts

The VM 8051 provides 3 interrupt sources: one external interrupt, one timer interrupt and one TCP/IP and UDP interrupt. The standard 8051 interrupts are listed in the interrupt vector address table, but these vector addresses are not supported and are reserved for future use.

Interrupt priority levels are not supported. An internal polling service determines which request is serviced. The polling service is based on the vector address. An interrupt with a lower vector address has higher priority than an interrupt with a higher vector address.

When an interrupt is generated, the VM clears the flag when the interrupt service routine is vectored to. If no interrupt service routine is implemented, then the interrupt flag mus be cleared by software.

Interrupt	**Source**	**Vector Address**
System Reset	API CMD: VM START REQUEST	0x0000
External Interrupt 0	*Not used (reserved)*	*0x0003*
Timer 0 Overflow	*Not used (reserved)*	*0x000B*
External Interrupt 1	*Not used (reserved)*	*0x0013*
Timer 1 Overflow	*Not used (reserved)*	*0x001B*
Serial Port	*Not used (reserved)*	*0x0023*
TCP/IP and UDP Sockets	FNET	0x002B
VM Timer	FTMR	0x0033
VM External Interrupt	FEXT	0x003B

6.4 Special Function Registers

0xF8									0xFF
0xF0	B * 0x00								0xF7
0xE8									0xEF
0xE0	ACC * 0x00								0xE7
0xD8									0xDF
0xD0	PSW * 0x00								0xD7
0xC8	IFLAG * 0x00	TMRCON 0x00	TMRL 0x00	TMRH 0x00					0xCF
0xC0	IE2 * 0x00	I2CCON 0x00	I2CADDR 0x00	I2CFREQ 0x14	I2CTXLEN 0x00	I2CRXLEN 0x00	I2CSTAT 0x00	EXTCON 0x00	0xC7
0xB8	IP # 0x00	NETCON 0x00	NETTXLEN 0x00	NETRXLEN 0x00	NETSTAT 0x00	NETTOUT 0x0A			0xBF
0xB0	P3 # 0x00	RXIP0 0x00	RXIP1 0x00	RXIP2 0x00	RXIP3 0x00	RXPORTL 0x00	RXPORTH 0x00	RXCH 0x00	0xB7
0xA8	IE # 0x00	TXIP0 0x00	TXIP1 0x00	TXIP2 0x00	TXIP3 0x00	TXPORTL 0x00	TXPORTH 0x00	TXCH 0x00	0xAF
0xA0	P2 # 0x00								0xA7
0x98	SCON # 0x00	SBUF # 0x00							0x9F
0x90	P1 # 0x00								0x97
0x88	TCON # 0x00	TMOD # 0x00	TL0 # 0x00	TL1 # 0x00	TH0 # 0x00	TH1 # 0x00			0x8F
0x80	P0 # 0x00	SP 0x07	DPL 0x00	DPH 0x00				PCON # 0x00	0x87

* Bit addressable registers
# Grayed 8051 standard registers are not supported by the VM 8051 and must not be used.
Unused SFR locations are reserved and must not be used.

IE2 – Interrupt Enable Register 2

Writing 1 enables the interrupt, and writing 0 disables the interrupt. Reading 1 indicates that the interrupt is enabled, and reading 0 indicates that the interrupt is disabled.

IE2 = 0xC0 - Bit addressable

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	-	EEXT	ETMR	ENET
Access	-	-	-	-	-	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
0	ENET	Enable Ethernet Receive Data Available Interrupt	0
1	ETMR	Enable Timer Interrupt	0
2	EEXT	Enable External Interrupt	0
-		Reserved, user software should not write ones to reserved bits.	0

IFLAG – Interrupt Flag Register

The IFLAG register allows reading the flag of peripheral interrupts, or for clearing the flag of those interrupts. The interrupt flag is cleared by the VM, when the VM vectors to the interrupt service routine (ISR). If no ISR is implemented, then interrupt flag mus be cleared by software.

IFLAG = 0xC8 - Bit addressable

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	-	FEXT	FTMR	FNET
Access	-	-	-	-	-	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
0	FNET	Receive Data Available Interrupt Flag 0: Interrupt is not pending 1: Interrupt is pending	0
1	FTMR	Timer Interrupt Flag 0: Interrupt is not pending 1: Interrupt is pending	0
2	FEXT	External Interrupt Flag 0: Interrupt is not pending 1: Interrupt is pending	0
-		Reserved, user software should not write to reserved bits.	0

I2CCON – I2C Control Register

I2CCON = 0xC1

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	-	I2CRS	I2CRD	I2CWR
Access	-	-	-	-	-	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
0	I2CWR	Writing 1 to this register performs I2C WRITE to a device on the I2C bus	0
1	I2CRD	Writing 1 to this register performs I2C READ from a device on the I2C bus	0
2	I2CRS	If set to 1, then a REPEATED START is used between I2C WRITE and I2C READ.	0
-		Reserved, user software should not write ones to reserved bits.	0

This register is cleared after performing a WRITE or READ. Check register I2CSTAT to read the result of the last I2C transaction.

Example 1:
Write 2 bytes (e.g. 0xAA and 0xBB) to the slave address 0x40:

```
I2CADDR = 0x40;          //load I2C Slave address
I2CTXBUF[0] = 0xAA;      //load first data byte
I2CTXBUF[1] = 0xBB;      //load second data byte
I2CTXLEN = 0x02;         //load number of bytes to write
I2CCON = 0x01;           //I2CWR = 0x01=> perform I2C WRITE
```

Example 2:
Read 1 byte from the slave address 0x40:

```
I2CADDR = 0x40;          //load I2C Slave address
I2CRXLEN = 0x01;         //load number of bytes to read
I2CCON = 0x02;           //I2CRD = 0x02=> perform I2C READ.
X = I2CRXBUF[0];         //store byte read to variable X.
```

Example 3:
Write 2 bytes (e.g. 0xAA and 0xBB) and read 1 byte to/from slave address 0x40:

```
I2CADDR = 0x40;          //load I2C Slave address
I2CTXBUF[0] = 0xAA;      //load first data byte
I2CTXBUF[1] = 0xBB;      //load second data byte
I2CTXLEN = 0x02;         //load number of bytes to write
I2CRXLEN = 0x01;         //load number of bytes to read
I2CCON = 0x03;           //I2CWR + I2CRD = 0x03 => perform I2C WRITE followed
                         //by I2C STOP and then I2C READ.
                         //REPEATED START is not used.
```

Example 4:
Write 2 bytes (e.g. 0xAA and 0xBB) and read 3 bytes to/from the slave address 0x40:

```
I2CADDR = 0x40;          //load I2C Slave address
I2CTXBUF[0] = 0xAA;      //load first data byte
I2CTXBUF[1] = 0xBB;      //load second data byte
I2CTXLEN = 0x02;         //load number of bytes to write
I2CRXLEN = 0x03;         //load number of bytes to read
I2CCON = 0x07;           //I2CWR + I2CRD + I2CRS = 0x07 => perform I2C WRITE
                         //followed by REPEATED START and I2C READ.
```

SLVADDR – I2C Slave Address Register

SLVADDR = 0xC2

Bit	7	6	5	4	3	2	1	0
Symbol	A7	A6	A5	A4	A3	A2	A1	A0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	A7:A0	This register is the slave address to write to or to read from. The LSB (Bit 0) is the direction Bit. Whether writing or reading and thus the direction is determined by the I2CCON register, the LSB has no effect and is always set to 0.	0x00

I2CFREQ – I2C SCL Frequency Register

I2CFREQ = 0xC3

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	This register contains the SCL frequency. The frequency is specified in 5000 Hz units. A value of 0x14 = 20 corresponds to 100 kHz. (20 x 5000 Hz = 100000 Hz) The frequency range is between 5 kHz (0x01) and 400 kHz (0x50)	0x14

I2CTXLEN – I2C WRITE Data Length Register

I2CTXLEN = 0xC4

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The number of bytes to be written to an I2C device must be written to this register before an I2C WRITE is performed. The minimum length is 1 byte and the maximum length is 255 bytes. After an I2C WRITE has been performed, this register contains the actual number of bytes written.	0x00

I2CRXLEN – I2C READ Data Length Register

I2CRXLEN = 0xC5

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The number of bytes to be read from an I2C device must be written to this register before an I2C READ is performed. The minimum length is 1 byte and the maximum length is 255 bytes. After an I2C READ has been performed, this register contains the actual number of bytes read.	0x00

I2CSTAT – I2C Status Register

I2CSTAT = 0xC6

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R	R	R	R	R	R	R	R

After an I2C WRITE, READ or WRITE and READ has been performed the I2C Status register contains the status of the last I2C transaction.

Bit	Symbol	Description		Reset Value
7:0	D7:D0	RESULT_OK	0x00	0x00
		ERROR_I2C_ERROR	0x01	
		ERROR_I2C_ARBLOST	0x02	
		ERROR_I2C_BUS_ERROR	0x03	
		ERROR_I2C_INVALID_WLEN	0x04	
		ERROR_I2C_INVALID_RLEN	0x05	
		ERROR_I2C_NAK_WADR	0x06	
		ERROR_I2C_NAK_RADR	0x07	
		ERROR_I2C_NAK_DAT	0x08	
		ERROR_I2C_TIMEOUT	0x09	
		ERROR_I2C_BUSY	0x0A	
		ERROR_I2C_UNKNOWN	0xFF	

NETCON – Network Control Register

NETCON = 0xB9

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	FEST3	FEST1	FPHY	NETNH	NETWR
Access	-	-	-	R	R	R	R/W	R/W

Bit	Symbol	Description	Reset Value
0	NETWR	Writing 1 to this register transfers the data available in the output buffer NETTXBUF to the socket specified in register TXCH. This bit is cleared by the VM after transmission.	0
1	NETNH	NO HEADER Bit is valid only when TXCH is equal to 0. Data sent over sockets 1 (TCP/IP) and 2 (UDP) is prepended **with** a header to be processed by the API. Data sent over sockets 3 (VM TCP/IP) and 4 (VM UDP) is transmitted **without** a header.	0
2	FPHY	PHYLINK FLAG 0 : Physical network cable is unplugged 1 : Physical network cable is plugged in Read only Bit	0
3	FEST1	Socket 1 (TCP/IP) Established Flag. 0 : Connection is closed 1 : Connection is established Read only Bit	0
4	FEST3	Socket 3 (TCP/IP) Established Flag. 0 : Connection is closed 1 : Connection is established Read only Bit	0
-		Reserved, user software should not write to reserved bits.	0

Example 1:
Send 3 bytes (e.g. 0xAA, 0xBB and 0xCC) to the client connected to socket 1:

```
TXCH = 0x01;              //select socket 1 (TCP/IP)
NETTXBUF[0] = 0xAA;       //load first data byte
NETTXBUF[1] = 0xBB;       //load second data byte
NETTXBUF[2] = 0xCC;       //load third data byte
NETTXLEN = 0x03;          //load number of bytes to send
NETCON = 0x01;            //NETWR = 0x01=> start data transmission
```

Example 2:
Send 3 bytes (e.g. 0xAA, 0xBB and 0xCC) to the IP address 1952.168.1.200 and port number 5000 using UDP:

```
TXCH = 0;                 //no socket is selected, send to the IP address below
TXIP0 = 200;              //low byte of the IP address
TXIP1 = 1;
TXIP2 = 168;
TXIP3 = 192;              //high byte of the IP address
TXPORT0 = 0x88;           //low byte of port number
TXPORT1 = 0x13;           //high byte of port number
NETTXBUF[0] = 0xAA;       //load first data byte
NETTXBUF[1] = 0xBB;       //load second data byte
NETTXBUF[2] = 0xCC;       //load third data byte
NETTXLEN = 0x03;          //load number of bytes to send
NETCON = 0x01;            //NETWR = 0x01=> start data transmission
```

Example 3:
Send 3 bytes (e.g. 0xAA, 0xBB and 0xCC) without header bytes to the IP address 1952.168.1.200 and port number 5000 using UDP:

```
TXCH = 0;                 //no socket is selected, send to the IP address below
TXIP0 = 200;              //low byte of the IP address
TXIP1 = 1;
TXIP2 = 168;
TXIP3 = 192;              //high byte of the IP address
TXPORT0 = 0x88;           //low byte of port number
TXPORT1 = 0x13;           //high byte of port number
NETTXBUF[0] = 0xAA;       //load first data byte
NETTXBUF[1] = 0xBB;       //load second data byte
NETTXBUF[2] = 0xCC;       //load third data byte
NETTXLEN = 0x03;          //load number of bytes to send
NETCON = 0x03;            //NETWR + NETNH = 0x03=> start data transmission
                          //do not prepend and send header bytes
```

NETTOUT – Transmit Timeout Register

NETTOUT = 0xBD

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	Data transfer is asynchronous, so confirmation is required after successful transfer. In order not to have to wait endlessly for a confirmation because of an unexpected error, a timeout is necessary. The Transmit Timeout is specified in 10 ms units. A value of 0x0A = 10 corresponds to 100 ms. A value of 0x00 means an unlimited timeout.	0x0A

NETTXLEN – Transmit Data Length Register

NETTXLEN = 0xBA

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The number of bytes to be transmitted to the socket specified in register TXCH. The minimum length is 1 byte and the maximum length is 255 bytes.	0x00

NETRXLEN – Receive Data Length Register

NETRXLEN = 0xBB

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The number of bytes received from the socket specified in register RXCH. The minimum length is 1 byte and the maximum length is 255 bytes.	0x00

NETSTAT – Network Status Register

NETSTAT = 0xBC

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R	R	R	R	R	R	R	R

Bit	Symbol	Description		Reset Value
7:0	D7:D0	RESULT_OK	0x00	0x00
		TRANSMIT_PENDING	0x01	
		ERROR_NET_SOCKNUM	0x02	
		ERROR_NET_SOCKINIT	0x03	
		ERROR_NET_SOCKCLOSED	0x04	
		ERROR_NET_SOCKMODE	0x05	
		ERROR_NET_SOCKSTATUS	0x06	
		ERROR_NET_PORTZERO	0x07	
		ERROR_NET_IPINVALID	0x08	
		ERROR_NET_TIMEOUT	0x09	
		ERROR_NET_DATALEN	0x0A	
		ERROR_NET_BUFFER	0x0B	
		ERROR_NET_BUSY	0x0C	
		ERROR_NET_ACT_TIMEOUT	0x0D	
		ERROR_NET_UNKNOWN	0x0FF	

RXIPx – Source IP Address Register

RXIP0 = 0xB1 ; RXIP1 = 0xB2 ; RXIP2 = 0xB3 ; RXIP3 = 0xB4

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The IP address of the client from which last data was received. e.g. 192.168.1.101 RXIP0 = 101 RXIP1 = 1 RXIP2 = 168 RXIP3 = 192	0x00

RXPORTx – Source Port Number Register

RXPORTL = 0xB5 ; RXPORTH = 0xB6

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The port number of the client from which last data was received. e.g. 5000 = 0x1388 RXPORTL = 0x88 (Low Byte) RXPORTH = 0x13 (High Byte)	0x00

RXCH – Source Socket Number

RXCH = 0xB7

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The socket number of the client from which last data was received. 0x01 : TCP/IP (Client → API → Core 1 → Core 2) 0x02 : UDP (Client → API → Core 1 → Core 2) 0x03: TCP/IP (Client → Core 2) 0x04: UDP (Client → Core 2)	0x00

TXIPx – Destination IP Address Register

TXIP0 = 0xA1 ; TXIP1 = 0xA2 ; TXIP2 = 0xA3 ; TXIP3 = 0xT4

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The IP address of the client to send data to. The destination IP address is used only when TXCH is set to 0. UDP protocol is used. e.g. 192.168.1.101 TXIP0 = 101 TXIP1 = 1 TXIP2 = 168 TXIP3 = 192	0x00

TXPORTx – Destination Port Number Register

TXPORTL = 0xA5 ; TXPORTH = 0xA6

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The port number of the client to send data to. The destination port number is used only when TXCH is set to 0. UDP protocol is used. e.g. 5000 = 0x1388 TXPORTL = 0x88 (Low Byte) TXPORTH = 0x13 (High Byte)	0x00

TXCH – Destination Socket Number

RXCH = 0xA7

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The socket number of the client to send data to. 0x00 : UDP (Core 2 → Client) 0x01 : TCP/IP (Core 2 → Core 1 → API → Client) 0x02 : UDP (Core 2 → Core 1 → API → Client) 0x03: TCP/IP (Core 2 → Client) 0x04: UDP (Core 2 → Client) **Socket Number 0:** If TXCH is set to 0, the UDP protocol is used to send data to the IP address and port number specified in the TXIPx and TXPORTx registers. When NETNH is set to 0 in the NETCON register, data is prepended **with** header bytes and sent to a client via Core 1 and API. When NETNH is set to 1 in the NETCON register, data is sent to a client **without** header bytes. **TCP/IP Sockets 1 and 3:** TCP/IP sockets send data to the connected clients. **UDP Sockets 2 and 4:** To send data over UDP socket **2**, Core 1 must have received data from a client at least once. To send data over UDP socket **4**, Core 2 must have received data from a client at least once. Data is sent to the last IP address and port number from which data was received.	0x00

EXTCON – External Interrupt Control Register

EXTCON = 0xC7

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	-	-	EXTEDG	EXTEN
Access	-	-	-	-	-	-	R/W	R/W

Bit	Symbol	Description	Reset Value
0	EXTEN	0 : External Interrupt Input Pin is disabled. 1 : External Interrupt Input Pin is enabled.	0
1	EXTEDG	0 : External interrupt is falling edge sensitive. 1 : External interrupt is rising edge sensitive.	0
-		Reserved, user software should not write ones to reserved bits.	0

TMRCON – Timer Control Register

EXTCON = 0xC9

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	-	TMRRS	TMRBL	TMRRN
Access	-	-	-	-	-	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
0	EXTRN	0 : Timer is disabled. 1 : Timer is enabled.	0
1	EXTBL	0 : Regular non-blocking timer. 1 : Special blocking timer is enabled. VM 8051 stops executing the bytecode until the time set in the TMRx registers has elapsed.	0
2	EXTRS	0 : The timer stops after the time set in the TMRx registers has elapsed. 1 : The timer restarts after the time set in the TMRx registers has elapsed. Not valid for blocking timer	0
-		Reserved, user software should not write ones to reserved bits.	0

TMRx – Time Interval Register

TMRL = 0xCA ; TMRH = 0xCB

Bit	7	6	5	4	3	2	1	0
Symbol	D7	D6	D5	D4	D3	D2	D1	D0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Symbol	Description	Reset Value
7:0	D7:D0	The Time Interval is the number of milliseconds set as countdown. e.g. 1000 = 0x03E8 TMRL = 0xE8 (Low Byte) TMRH = 0x03 (High Byte)	0x00

Revision history

Rev.	Date	Description
1.0	14 JULY 2022	LAN I2C Adapter VM – Product data sheet