

Morpheus

System Architecture

and

Developer's Guide

William Henning

<http://Mikronauts.com>

Table of Contents

Table of Contents

Introduction.....	4
A brief history of Morpheus.....	4
Morpheus System Diagram.....	6
Unique Features.....	7
Common Features.....	7
Expansion Boards.....	8
Mem+.....	8
School Board – “Icing on the Cake”.....	9
Proteus.....	10
Propteus.....	11
Morpheus Subsystems.....	12
Second Propeller.....	12
256 Color VGA.....	12
MORPHBUS.....	13
512KB RAM.....	14
Boot EEPROM.....	14
Real Time Clock.....	14
SPI RAM.....	14
SPI Flash.....	14
Expansion Connectors.....	15
EXP1.....	15
EXP2.....	15
Programming Connectors.....	16
H-COMM1.....	16
H-COMM2.....	16
I/O Map.....	16
CPU1.....	16
CPU2.....	17
Memory Map.....	18
Mem+ Addressing Limitations.....	19
Memory Mapped I/O.....	19
Morpheus Shadow RAM.....	19
SPI RAM.....	19
Developing for Morpheus.....	20
EEPROM.....	20
RTC.....	21
SPI Flash.....	22
SPI RAM.....	23
Controlling the second Propeller.....	24
Inter Processor Communications.....	25

XMM.....	26
Theory of Operation.....	26
Random Access Read/Write.....	28
Burst Mode Read/Write.....	29
VGA256 Subsystem.....	30
Theory of operations – Standard Parallax VGA.....	30
Theory of operations – Morpheus 256 Color VGA.....	31
MEM+ Subsystems.....	34
Address Decoding.....	34
512KB-2MB Static RAM.....	34
SD Card Interface.....	34
Serial Port.....	35
MCP23S17 16 bit Parallel I/O.....	35
Appendix I: Morpheus Schematic.....	36
Morpheus Schematic (Page 2 of 3).....	37
Morpheus Schematic (Page 3 of 3).....	38
Appendix II: Mem+ Schematic.....	39
Mem+ Schematic (Page 2 of 2).....	40
Appendix III: Frequently Asked Questions.....	41
Appendix IV: Acknowledgements & Recommendations.....	42
Recommended Projects.....	42
Data Sheets.....	42

Introduction

A brief history of Morpheus

Once upon a time, in a land far far away... oops, sorry, wrong topic.

What is Morpheus?

From one point of view, Morpheus a dual Propeller single board computer with 512KB of XMM, 256 color extension to the Parallax VGA model, Flash memory, RTC and an expansion bus.

From another point of view, it is the end result of almost three years of studying and playing with the Propeller, on top of a Computer Science education and over 25 years of experience in Industrial control – and several years of experience working with and in education.

I started bugging Chip back in September, 2006 – and had several e-mail exchanges with Paul Baker back then too. I was hooked from the moment I laid eyes on the Propeller specs – I mean eight cores, 32KB ram, all in an easy to use DIP 40 package? NTSC/PAL and VGA built in? After PIC's and AVR's it looked like heaven.

My demo board arrived, and I started losing sleep.

I went through the demo's, marvelling at how easy it was to generate color text and tiled graphics.

I started writing pasm code, and started dreaming of C compilers.

Then the realization hit. Cogs are limited to 496 instructions at most (ignoring shadow registers). Not a very useful target for C code. No return stack – so no recursion. No indexed addressing modes, although they can be implemented using self-modifying code.

Remember, at that time Spin was new to me, and I am an old fart when it comes to C – I've been programming in it since 1980 or so. I really wanted to get at least SmallC running.

Some head scratching later, I came up with LMM, and after a while, I could no longer hold it in, and I announced LMM on the forums. You can still find the original thread here:

<http://forums.parallax.com/forums/default.aspx?f=25&m=154421>

Ok, so now I had a machine architecture extension for the Propeller that could be targeted by C compilers. I first wanted to write an LMM assembler as my experiments with writing LMM code with the Propeller tool were exceedingly painful and frustrating. I had vague thoughts of trying to port SmallC after the assembler was done.

Then work and life intervened – between a lucrative consulting contract, and meeting my (then future) better half, LMM had to be put on the back burner.

But I could not put the Propeller totally away. I kept playing with hardware, and dreaming of getting an LMM assembler and Small C running.

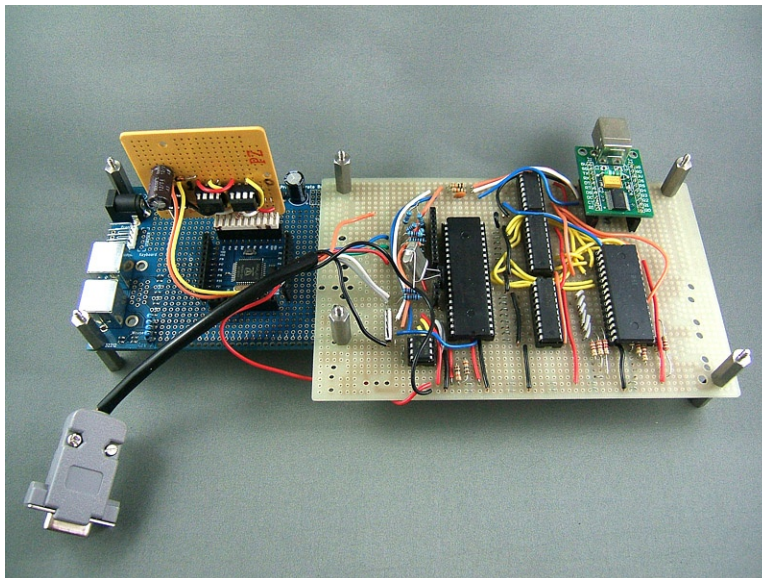
Eventually I was approached by Chip and ImageCraft, looking to make a commercial LMM C compiler. After some email discussions, Richard and I reached an agreement – as I really wanted C on the Propeller, and I wanted the Propeller to be very popular, all I asked for were C compilers and credit for coming up with LMM.

At this point I gave up my vague ideas for porting a C compiler – and started designing an operating system instead. This is when Largos was conceived – although that name did not exist yet.

A short time after that, I realized that as great as the demo board, proto boards and Hydra were, they really were not complete enough to run a full operating system... thus I started playing with designing a Propeller based single board computer.

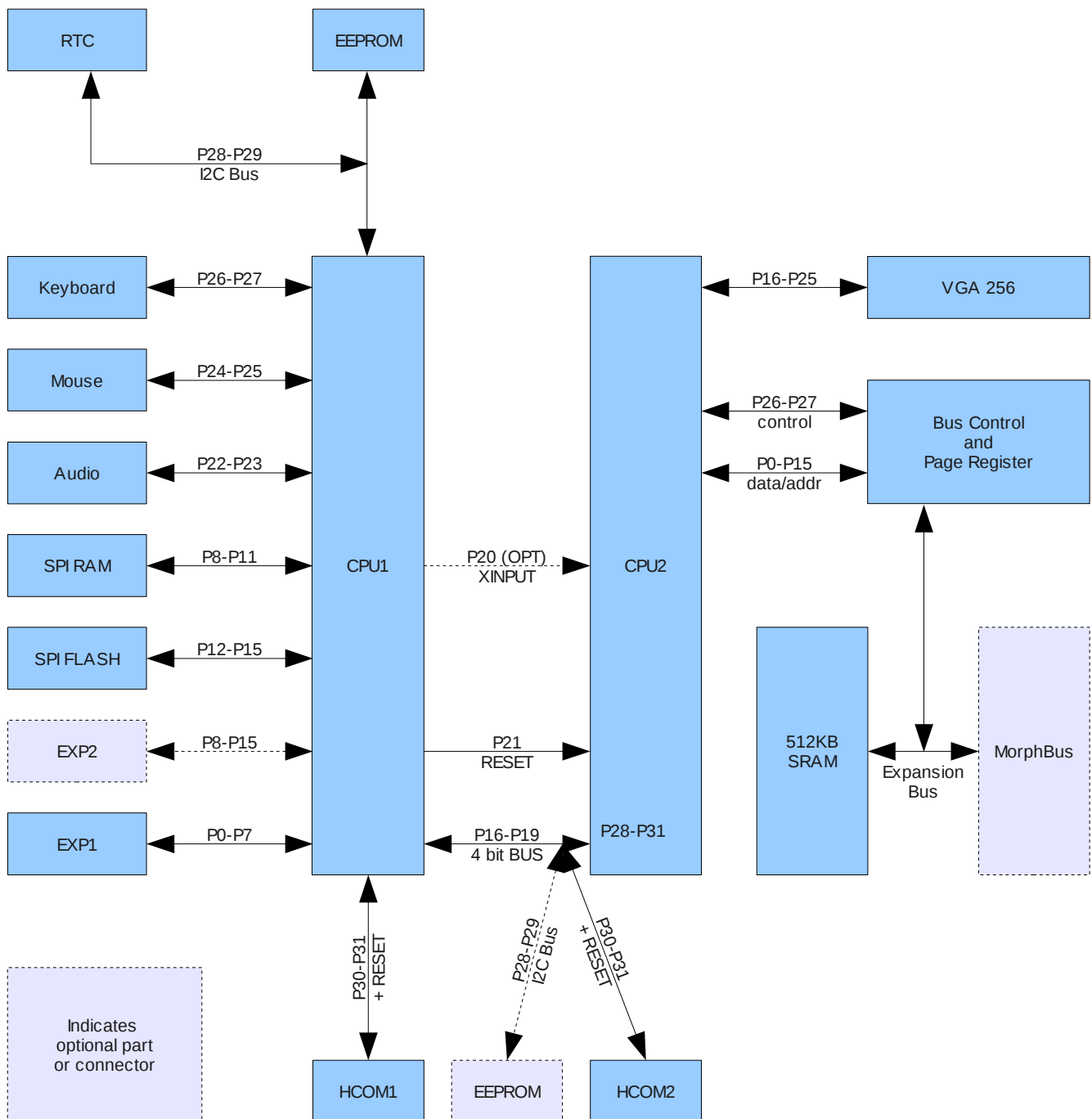
Over time, I grew to really enjoy programming in Spin, and as my LMM assembler was still buggy at that time, I started prototyping Largos as a mixture of Spin and pasm code. Ofcourse now that Las works... but I digress.

I won't bore you with all the false starts, the pitfalls of working in short “time snippets”, partial prototypes, and many experiments... instead I'll get to the crux of the matter, and introduce Morpheus – the end product of almost two years of research and development.



The first Morpheus prototype – built May 2008

Morpheus System Diagram



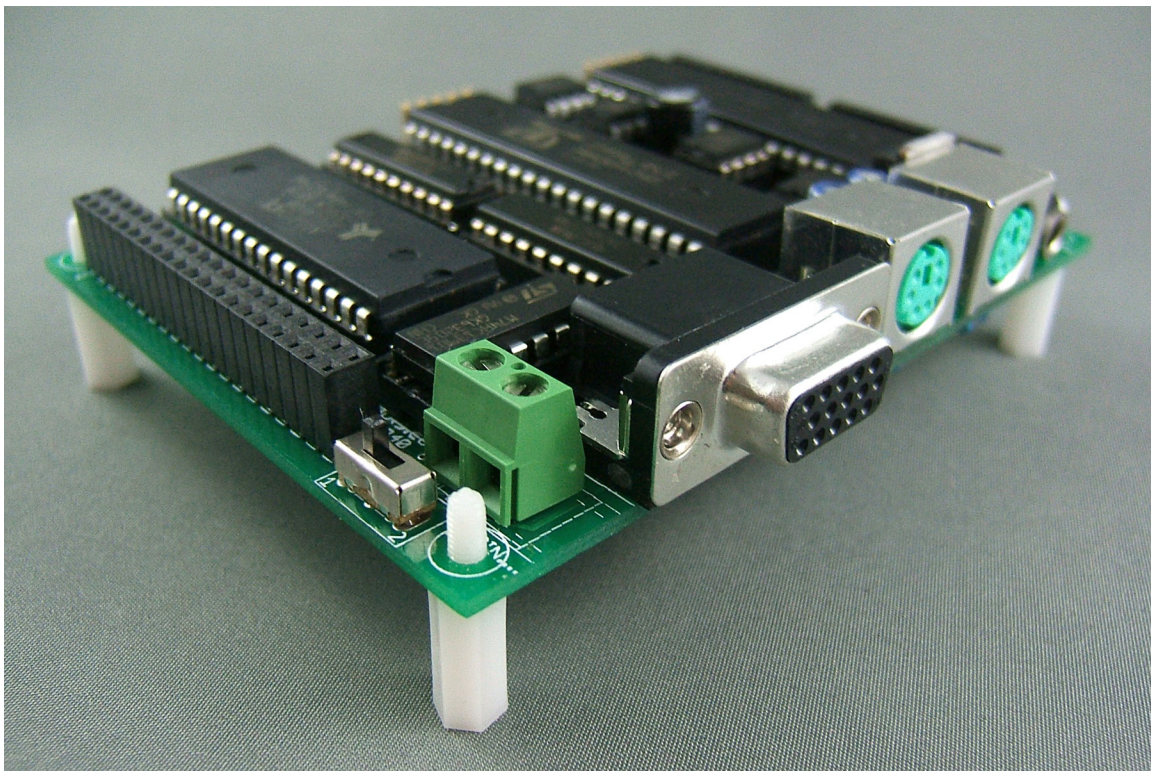
You are going to want to keep a copy of this page handy near your Morpheus board – it gives you the port mappings for all the I/O, and shows you how the subsystems are interconnected.

Unique Features

- Well-defined expansion bus with 24 address bits and 8 data bits
- 512KB reserved for memory mapped I/O devices
- 15.5MB available for memory expansion
- 8 bit VGA “true color” with 3 bits Red, 3 bits Green, 2 bits Blue
- 512KB parallel ram and 32KB SPI ram on board
- 1MB-8MB of SPI flash on board
- Real Time Clock with capacitor backup on board
- 4 bit bus connecting to P28-P31 on CPU2
- fifth pin controls CPU2's RESET line
- optional generation of clock signal for CPU2

Common Features

- P0-P7 on CPU1 available on a ProtoBoard compatible header
- Keyboard on CPU1, Demo board compatible pins
- Mouse on CPU1, Demo board compatible pins
- Stereo line level output to a 1/8th inch jack on CPU1 on pin P22 & P23
- EEPROM on CPU1
- optional EEPROM on CPU2
- Propeller Plug compatible programming headers for both CPU1 & CPU2



Here is what an assembled Morpheus looks like

Expansion Boards

*** WARNING ***

The power regulation on Morpheus is only designed to drive logic on Morpheus and expansion boards.

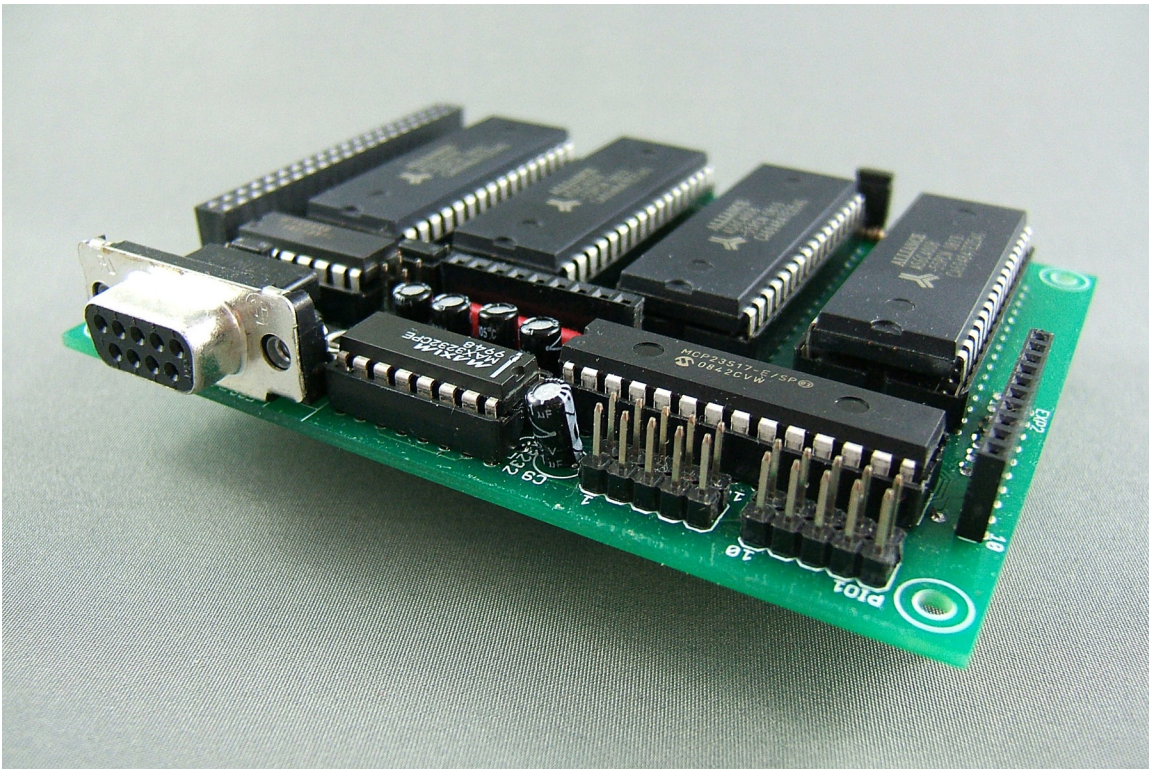
Do not power motors, servos, solenoids, or any power hungry device from MORPHBUS, if you do so it is entirely at your risk as you will be exceeding design parameters.

Only a total of 300mA may be drawn from the +5V rail by all expansion boards, and only 400mA may be drawn from the +3.3V rail.

Use an external power supply for powering servo's, motor's etc, and don't forget to tie external ground to Morpheus ground.

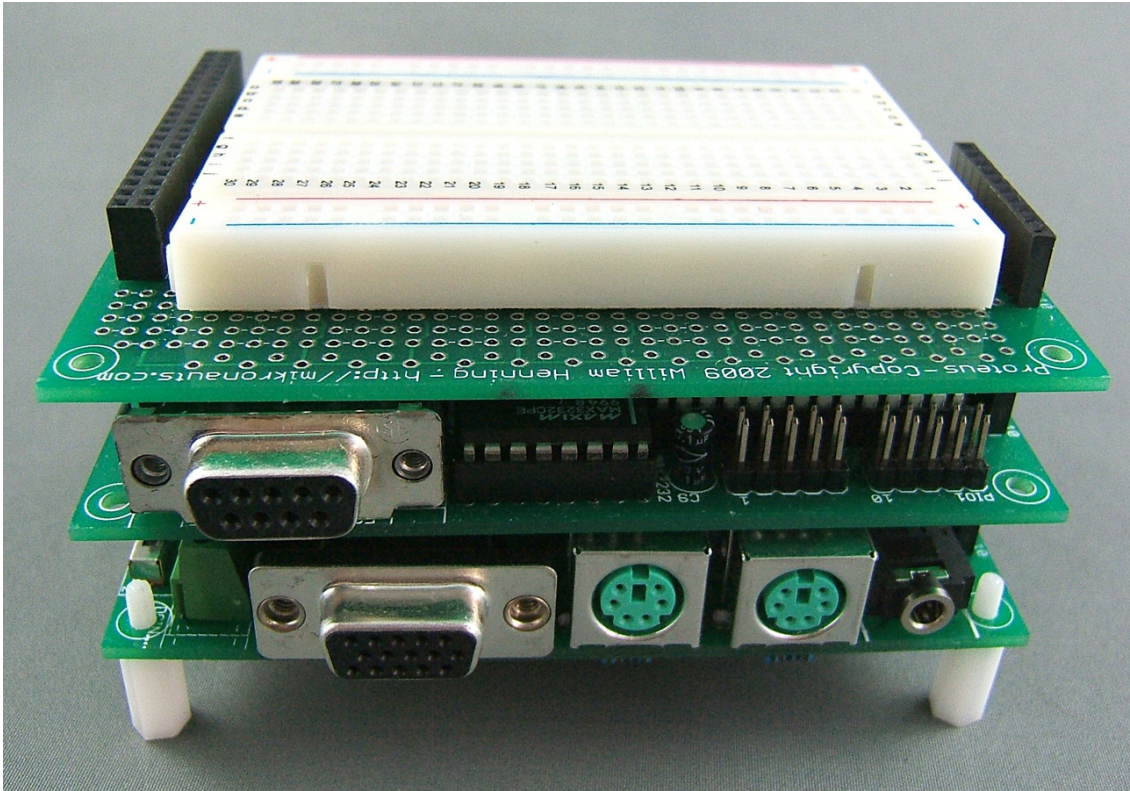
Mem+

- Up to 2MB of ram per board
- Up to four boards per system (addressable to any 2MB block in first 8MB)
- optional MAX3232 based serial port, normally connected to CPU1's programming header
- optional 16 bit SPI I/O expander, uses P0-P3 on Morpheus, only one per system
- optional SD card header, uses P4-P7 on Morpheus, only one per system



School Board – “Icing on the Cake”

- 400 pin solderless breadboard
- 40 pin connector allowing access to MORPHBUS
- 10 pin header allowing access to P0-P7 on CPU1
- optional 10 pin header allowing access to P8-P15 on CPU1



SchoolBoard mounted on a Mem+, which is mounted on Morpheus – a “Full Stack”

When you are experimenting with Propeller circuits using a SchoolBoard, and you need access to P0-P7 on CPU1, you should do one of the following:

- Mount the SchoolBoard directly on Morpheus, allowing free use of P0-P7

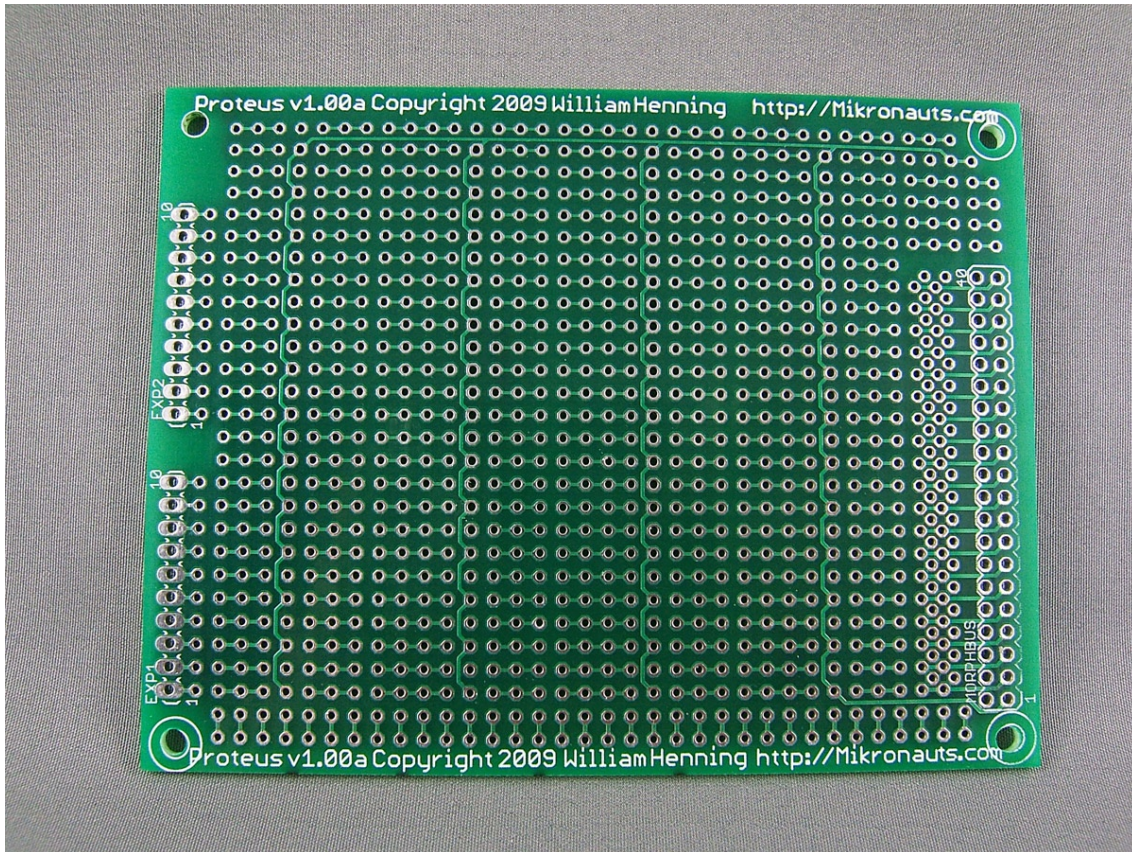
OR

- Remove any SD card and the MCP23S17 from the Mem+, allowing SchoolBoard to access P0-P7

If you are just experimenting with connecting devices to MORPHBUS and don't need to use P0-P7 there is no need to worry about sharing P0-P7 with the SD card and MCP23S17 on Mem+.

Proteus

- 40 pin connector allowing access to MORPHBUS
- Access to EXP1
- (optional) access to EXP2
- large prototyping area
- unique power grid for easy access to +3.3V and GND
- rows of three pads are bussed for easy 0.6" and 0.3" dip mounting
- up to 32 pins worth of single row headers at the bottom of the board

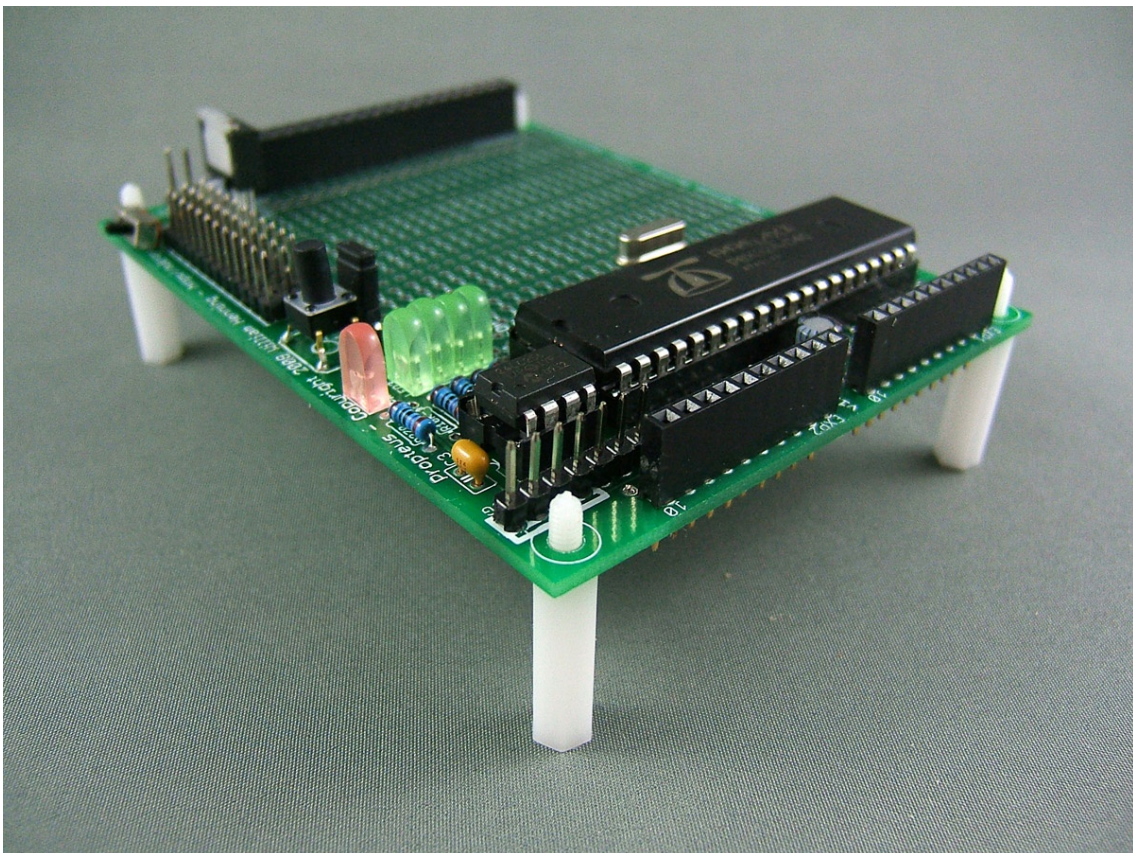


Proteus is a great blank slate. You can build anything you want for your Morpheus system on it – or use it as a stand alone prototyping board. Suggested uses:

- Parallel I/O driven by MORPHBUS
- Parallel A/D and D/A converters connected to MORPHBUS
- Parallel LCD displays
- Anything your mind can imagine!

Propteus

- 40 pin connector allowing access to MORPHBUS
- optional access to Morpheus EXP1 and EXP2
- DIP 40 Propeller
- 8 pin DIP EEPROM
- four user LED's with current limiting resistors
- two ProtoBoard compatible expansion headers
- unique power grid for easy access to +3.3V and GND
- rows of three pads are bussed for easy 0.6" and 0.3" dip mounting
- reset switch with jumper, can be used as a user switch
- large prototyping area
- PropPlug compatible programming header
- 2 pin power connector
- power switch and LED
- 3.3V regulator circuitry



There will be more boards coming... Morpheus is just the start of an extensive line of boards allowing a Lego-like “plug and play” approach for configuring semi-custom Propeller based solutions for a vast array of problems. (Ok, enough market-speak for now.)

Morpheus Subsystems

Second Propeller

In some ways CPU2, the second Propeller chip on Morpheus is a slave of CPU1 – but using it as a slave processor is not the goal of the system architecture, but a simple side effect of one processor loading the other processor – therefore it made sense to have the processor with the flash memory control the reset line of its peer.

I said peer, and I meant it – Morpheus is designed with a four bit bus connecting the processors, and once the right drivers are developed, this will allow close to 5MB/sec (burst rate) communications between the processors. Until that time, it is very easy to run FullDuplexSerial on CPU1 & CPU2 for low speed communications, and there are several projects on the forums for higher speed inter-Propeller communications that would also work on Morpheus.

Applications are meant to be distributed – spread over – both processors on Morpheus. Largos will make this easier when it becomes available. CPU1 is more I/O oriented – it has the keyboard, mouse, audio, SPI flash, SPI ram, RTC and optional SPI I/O and SD card. CPU2 has the expansion bus, lots of fast memory, and an extended VGA capability. You are already used to splitting up programs between eight cogs – using sixteen should not be too difficult.

256 Color VGA

Morpheus can be populated so that it is compatible with the standard six bit DAC the Parallax VGA drivers use (2 bits each of R, G and B, and two bits for Hsync and Vsync). I'll be writing an app note on how to do this later, but I'll bet most of you will want the extra colors that Morpheus provides.

The 256 color mode works by implementing a three bit DAC for red and green, and a two bit DAC for blue – this configuration was used by IBM earlier for a so-called “8 bit true color” mode. Morpheus uses CPU2's P16-P23 for the DAC's.

The Hsync and Vsync pins have been moved off to P24&P25 on CPU2, which is in the next eight bit group.

This does mean that the standard Parallax VGA drivers need to be modified – and it is true that synchronizing the Hsync signal between the cogs requires a bit of effort – but it is not much worse than writing two cog synchronized drivers.

The 192 extra colors are well worth the effort.

MORPHBUS

A key feature of the Morpheus line is a well defined expansion bus.

While it is true that most people will use it only for memory expansion, the bus is far more capable than that! MorphBus allows you to connect any bus oriented peripheral device – so all those lovely fast A/D and D/A converters can now easily be connected to Morpheus, and the SchoolBoard will let you easily experiment with parallel I/O devices – or other memory devices.

The bus consists of:

D0-D7	data bus, connected to P0-P7 on CPU2
A0-A7	low order address bits, connected to P8-P15 on CPU2
A8-A23	high order address bits, latched from P0-P15 on CPU2
/RD	active low read signal, decoded from P26&P27 on CPU2
/WR	active low write, decoded from P26&P27 on CPU2
+5V	power supply to boards stacked on Morpheus
+3.3V	power supply for boards stacked on Morpheus
GND	ground

Some of you may notice that the list above only defines 37 pins. The last three pins are reserved, and are not to be used at this time – they will be used to support future products.

Pin	Signal	Pin	Signal
40	/WR	39	+5.0V
38	/RD	37	reserved
36	reserved	35	reserved
34	A23	33	A22
32	A21	31	A20
30	A19	29	A18
28	A17	27	A16
26	A7	25	A6
24	A5	23	A4
22	A3	21	A2
20	A1	19	A0
18	A15	17	A14
16	A13	15	A12
14	A11	13	A10
12	A9	11	A8
10	D7	9	D6
8	D5	7	D4
6	D3	5	D2
4	D1	3	D0
2	GND	1	+3.3V

512KB RAM

After much careful deliberation, I designed the expansion bus (and thus memory subsystem) around 256 byte pages, with a sixteen bit “page select” register allowing 2^{16} (16MB) bytes to be addressed.

There was only room for one 32 pin dip chip on Morpheus limiting it to 512KB of fast memory.

Should you need more memory, you can expand up to 7.5MB right now, and 15.5MB in the future.

Why 7.5MB?

In order to fit everything on the Mem+ board without adding another chip, Mem+ boards can only be addressed in the range of \$000000-\$7FFFFFFF. Each Mem+ occupies 2MB in that address space – thus we are limited to four Mem+ boards in a stack. I reserved \$000000-\$007FFFFF for memory mapped I/O.

Should there be sufficient demand, I can manufacture an 8MB surface mounted memory board to occupy the \$800000-\$FFFFFFF address range.

Boot EEPROM

CPU1 has to boot from an EEPROM, thus Morpheus has a socket for a 32KBx8 EEPROM.

CPU2 would normally be booted from CPU1, but Morpheus has provision for a second EEPROM to be mounted on the bottom of the board.

Real Time Clock

CPU1 has a Phillips PCF8563 I2C Real Time Clock as I needed to know the time of day for Largos; and in general an RTC is an extremely useful chip to have on a single board computer.

SPI RAM

CPU1 has access to the expansion bus, and memory, connected to CPU2 over the 4 bit inter processor bus, however in some circumstances (for example, running high resolution bitmapped drivers, or timing sensitive emulations) it may be desirable to have some local memory expansion, therefore Morpheus supports a 23K256 or compatible SPI ram (or SPI fram) chip.

SPI Flash

In both industrial control and educational markets it is desirable to have the root file system on non-removable media. Kids and plant operators cannot resist removable media.

I also wanted Largos to be able to count on having a root file system available at all times, and the file system I designed for Largos takes less memory and is more featurefull than FAT – so I designed in a socket for an 8 pin SPI flash device. Currently I support W25X80 4K sector (or compatible) flash devices up to 8MB (64Mb) in size.

Expansion Connectors

EXP1

EXP1 provides access to P0-P7 of CPU1.

It is compatible with Parallax ProtoBoard I/O headers, and as such, it is also compatible with SpinStudio add-on boards from Ucontroller.com

EXP1 can be used several ways:

- on a stand-alone Morpheus board, it is available for use as general purpose Propeller I/O
- Mem+ uses P0-P3 for the optional MCP23S17 based 16 bit parallel I/O expander
- Mem+ uses P4-P7 for the optional SD card interface
- 8 bit data bus shared between multiple Morpheus and Propteus boards
- four CMOS level serial ports
- additional VGA output
- 8 PWM outputs
- and much more

EXP2

EXP2 provides access to P8-P15 of CPU2

- EXP2 is normally NOT available - it is used to provide Morpheus with SPI Flash and SPI RAM
- P8-P11 is connected to the “Flash 1” socket for a 23K256 32KB sram chip
- P12-P15 is connected to the “Flash 2” socket for a Winbond W25X80 device used for Largos

Like EXP1, EXP2 is also compatible with the standard Parallax I/O headers, and can also use SpinStudio add-ons.

Even though EXP2 is normally not available, if you are not planning to use Largos in the future, and you do not install any chips in the Flash1 and Flash2 socket, you can use EXP2 as general purpose Propeller I/O pins, implementing another VGA output, or TV out, or pretty much anything you can imagine.

Please note that if you do use EXT2 for something other than the SPI RAM and SPI Flash that normally uses those pins, Largos will NOT run on the board.

Programming Connectors

H-COMM1

The first four pins of H-COMM1 are PropPlug compatible, with Pin 1 being Vss on PropPlug. The fifth pin provides +3.3V for upcoming products.

H-COMM2

The first four pins of H-COMM2 are PropPlug compatible, with Pin 1 being Vss on PropPlug. The fifth pin provides +3.3V for upcoming products.

I/O Map

CPU1

The I/O pins of CPU1 are used as follows:

P0-P7	EXP1 expansion connector
P0-P3	optional MCP23S17 SPI I/O expander on Mem+
P4-P7	optional SD memory card interface on Mem+
P8-P15	EXP2 expansion connector (normally not available)
P8-P11	32KB SPI SRAM memory chip
P12-P15	1MB SPI FLASH memory chip
P16-P19	CPU1 to CPU2 4 bit inter processor bus
P20	optional CPU1 generated clock for CPU2
P21	/RESET signal for CPU2
P22,P23	stereo audio output
P24,P25	Mouse interface (available as I/O at PS/2 socket if not used for a mouse)
P26,P27	Keyboard interface (available as I/O at PS/2 socket if not used for a keyboard)
P28,P29	I2C interface for EEPROM and RTC
P30,P31	Serial RX&TX for H-COMM1

CPU2

The I/O pins of CPU2 are used as follows:

P0-P7 D0-D7 for MORPHBUS, also input for lower byte of address register

P8-P15 A0-A7 for MORPHBUS, also input for upper byte of address register

P16-P25 256 Color VGA interface

P16,P17 two bit Blue DAC

P18-P20 three bit Green DAC

P21-P23 three bit Red DAC

P24 HSync

P25 VSync

P26,P27 Bus Control Signal Generator

P27:P26=0:0 Latch Address Register (active low)

P27:P26=0:1 READ (active low)

P27:P26=1:0 WRITE (active low)

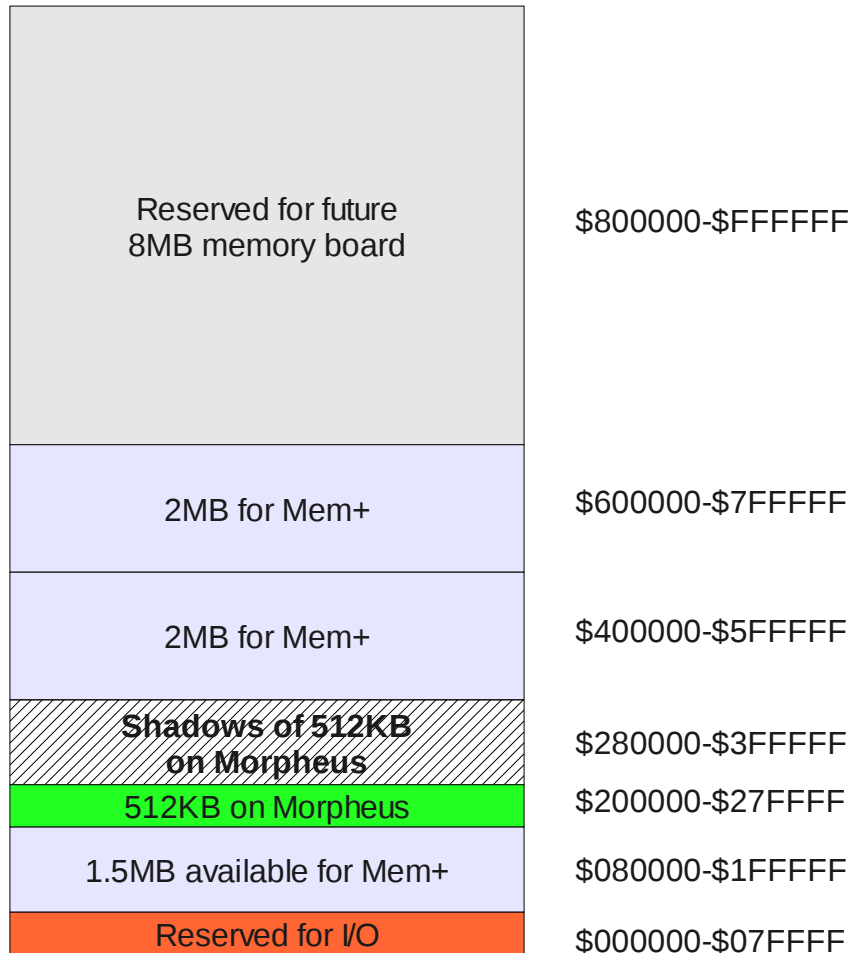
P27:P26=1:1 ALT (reserved)

P28-P31 CPU2 to CPU1 4 bit inter processor bus

P28,P29 I2C interface for optional EEPROM2

P30,P31 Serial RX&TX for H-COMM2

Memory Map



NOTE:

1. It is possible to use the full 2MB from \$200000-\$3FFFFFFF by removing the 512KB memory chip from the Morpheus board and setting the address jumper of a Mem+ board for the \$200000-\$3FFFFFFF address range
2. It is possible to map a Mem+ board into the \$000000-\$07FFFFFFF address range as long as the fourth memory socket is not populated.

If the fourth socket is populated, it may cause damage to the Propeller or memory mapped I/O devices should any be used in the system.

Mem+ Addressing Limitations

Memory Mapped I/O

In order to allow support for the extremely large variety of parallel I/O devices available on the market, Largos requires Morpheus to reserve a 512KB area of the 16MB memory address space for memory mapped I/O devices.

The first 512KB of addresses, from \$000000-\$07FFFF was chosen for memory mapped I/O in order to allow for an unbroken 15.5MB of linear address space for memory, including supporting a single 8MB high density static ram memory board in the \$800000-\$FFFFFF address range.

This design also allows for four Mem+ boards to provide 7.5MB of static ram in the low 8MB of the address space.

Morpheus Shadow RAM

The 512KB of memory on the Morpheus board occupies the entire \$200000-\$27FFFF address range due to sparse address decoding. Sparse decoding was necessary due to not having sufficient space left on the PC board for additional address decoding logic.

If the 512MB memory device is removed from Morpheus and placed on a Mem+ board strapped to select the \$200000-\$27FFFF address range then there will not be any “echoes” of the first 512KB.

SPI RAM

As you may have noticed, the SPI ram connected to CPU1 does not show up in the memory map.

This memory should be considered to be “private” memory for Largos.

Largos will use it for:

- file buffers
- text editor buffer
- temporary files

Until Largos is released, please use it as you see fit – but do realize that if you want your application to run under Largos, you will not be able to use it without asking Largos to lock it for you, and you must release it when you are done using it.

Developing for Morpheus

EEPROM

Morpheus uses standard Microchip 24L256 32KB I2C eeproms, on P28&P29.

Please see the Parallax Object Exchange for several objects that you can use to access EEPROM's.

The version of "i2c.Spin" in MorphDiag is a modified version of Mike Green's i2c driver; the I2C pins are hard wired to P28&P29.

Here is the definition of "eeprom_diag" found in MorphDiag – it uses the modified i2c.spin to display the first 256 bytes found in the EEPROM.

```
pub eeprom_diag | a
  ser.str(string(13,"EEPROM Diagnostic...",13))
  i2c.start
  i2c.initialize
  if i2c.ReadPage(i2c#EEPROM, 0, @buffer, 256)
    ser.str(string(13,"EEPROM Diagnostic...",13))
  else
    repeat a from 0 to 255
      if (a & 15) == 0
        ser.tx(13)
        ser.hex(a,4)
        ser.tx(":")
        ser.tx(" ")
        ser.hex(buffer[a],2)
```

See how easy it is to use the i2c routines?

RTC

The Phillips PCF8563 is a low cost eight pin dip Real Time Clock that is very easy to communicate with.

Morpheus provides two Spin routines:

- Date reads the RTC, updates the Largos date stam, and returns the current date/time as a string
- SetClock(year,month,day,hour,min,sec) sets the RTC to the specified date/time

For complete source code for Date and SetClock, and the RTC_Diag routine which uses Date and SetClock, please see "MorphDiag", which is available from the Downloads page of Mikronauts.com.

Here is what RTC_diag looks like:

```
pub RTC_diag
  ser.str(string(13,"RTC Time: "))
  ser.str(Date)
  ser.str(string(13,"Setting RTC to 2009/07/20 12:01:52"))
  SetClock(2009,7,20,12,1,52)
  repeat 10
    waitcnt(cnt+80_000_000)
    ser.str(string(13,"RTC Now: "))
    ser.str(Date)
```

SPI Flash

The WinBond W25X80 is an eight pin dip 8Mbit / 1MB chip that is intended to hold the root file system for Largos. It is currently possible to purchase a 16Mbit / 2MB dip device, however the 4MB and 8MB parts are not readily available in a dip format.

A sample low-level Spin driver “spin_flash.spin” for the Flash chip is provided in MorphDiag, available from the Downloads page of Mikronauts.com

A faster “pasm_flash.spin” driver will soon be available, and it will be 100% compatible with the “spin_flash” API.

Please see “flash_diag” in MorphDiag for sample code – here is a snippet to show you how easy it is to use the flash routines:

```
pub flash_diag | a,i,j,v,id
  ' use address 4096-8191 as that is normally .bad
  flash.init(12) ' initialize FLASH2
  id := flash.jdec_id
  ser.str(string(13,"  Flash JDEC ID is: "))
  ser.hex(id,8)
  ser.str(string(13,"Size in 4KB blocks: "))
  ser.dec(1<<((id&255)-12))
  ser.str(string(13,"      Size in bytes: "))
  ser.dec(1<<(id&255))
  ser.tx(13)
  if confirm(@flash_warning)
    ser.str(string(13,"Reading block..."))
    flash.read(@buffer,4096,4096)
    ser.str(string(13,"Erasing block..."))
    flash.write_en(1)
    flash.erase(4096,0)
```

SPI RAM

The Microchip 23K256 is an eight pin dip 256Kbit / 32KB static ram that can be used by CPU1 for temporary storage without worrying about the programming/erase cycle limitations of Flash or EEPROM.

A sample low-level Spin driver "spin_ram.spin" is provided in the latest version of MorphDiag.

A faster "pasm_ram.spin" driver is under development, and will soon be available – and again, it will be 100% compatible with the "spin_ram" API.

Here is some sample code from MorphDiag:

```

pub spi_ram_diag | i, j, k
  sram.init(8)
  ser.str(string(13,"SPI RAM Diagnostics - writing pattern"))
  repeat i from 0 to 32767 step 256
    repeat j from 0 to 255
      smallbuff[j]:=(i>>7)^j
      sram.write(@smallbuff,i,256)
      ser.tx(".")
  ser.str(string(13,"SPI RAM Diagnostics - reading pattern"))
  k:=0
  repeat i from 0 to 32767 step 256
    sram.read(@smallbuff,i,256)
    repeat j from 0 to 255
      if smallbuff[j]<>((i>>7)^j)
        ser.tx("x")
        k++
  if k
    ser.str(string(13,"Error: "))
    ser.dec(k)
    ser.str(string(" incorrect bytes read",13))

```

Controlling the second Propeller

CPU1 can reset CPU2 at will, and is hooked up to the pins CPU2 would otherwise use to communicate with a PropPlug and an EEPROM.

Chip's Propeller Loader can be used to boot the second processor, so while the board supports a separate EEPROM (EEPROM2) for CPU2, it is not needed for the vast majority of applications. Normally, the boot image for the second processor would be stored on the SPI Flash chip connected to CPU1.

Here is some sample code from the Largos test shell to load CPU2:

```
p := f_open(dir,argv[1],"w")
if valid_fp(p)
  ix := 0
  done := 0
  c := f_size(fp2fptr(p))
  if c>8192
    error(string("File too large for buffer"))
    f_close(p)
  else
    os.puts(string("Loading "))
    repeat while ix =< c
      f_read(p,@sector+ix,256)
      os.putc(".")
      ix+=256
    f_close(p)
    os.puts(string("Programming... "))
    c := loader.Connect(21, 19, 18, 1, loader#LoadRun, @sector)
    if c
      os.puts(string("CPU2 Loader returned error code "))
      os.dec(c)
    else
      os.puts(string("CPU2 Loaded."))
```


Inter Processor Communications

The simplest way for the two processors to communicate is FullDuplexSerial – and it is also the slowest method. Any of the various 10mbps and faster special communications routines discussed on the Parallax forums can also be used.

In the future I intend to implement a high speed four bit wide protocol for communicating between the processors – theoretically this could send small bursts at up to 10MB/sec, however as the nibbles would have to be encoded/decoded at each end, 5MB/sec is a more practical upper end for reliable communications between the processors.

For Propeller #1 start with:

```
' CPU #1
obj
    ser: "FullDuplexSerial"    ' Rx=18, Tx=19
main
    ser.start(18,19,0,115200)
```

For Propeller #2 start with:

```
' CPU #2
obj
    ser: "FullDuplexSerial"    ' Rx=31, Tx=30
main
    ser.start(31,30,0,115200)
```

After initializing the serial ports on both sides, just use the methods in FullDuplexSerial for communications between the processors.

XMM

Morpheus is designed with a 16MB memory address space of which the first 512KB reserved for future memory mapped I/O devices. This allows the use of fast parallel I/O, A/D and D/A converters as well as other more specialized I/O devices.

Theory of Operation

Please see the XGA_DEMO1 driver source files for an example of burst reading.

xmm_pasm.spin shows how to read and write individual bytes.

Selecting the right page

For random reads or writes, you first set the page register to the upper sixteen address bits of the location you want to access using the LATCH command (00 on P26 and P27).

Here is a PASM snippet showing you how to do this:

```
' compute "page" address from XMM address
mov  page, xaddr  ' move your address into "page"
shr  page, #8     ' zero's end up at p26&p27
mov  outa, page   ' set output register

' now load page register
mov  dira, bsel wz ' enable outputs  sets Z=0
muxz dira, ctrl   ' turn bits off in DIRA, pulls up
```

Please note how if you have a pre-computed page value, you can switch pages in just three instructions.

Selecting the right byte

Address lines A0-A7 are connected directly to P8-P15 on CPU2, so selecting the byte to read or write is as simple as:

```

mov  offset, xaddr
and  offset, #$ff
shl  offset, #8
mov  outa, offset
mov  dira, LADDR

```

LADDR is a constant that defines DIRA for the current cog as \$0000FF00 – that is all inputs except for P8-P15, which corresponds to A8-A15 on the bus.

In order to read a byte, simply

```

OR   dira, RSEL      ' bit mask for /READ bus signal

```

to write a byte

```

OR   dira,WSEL ' bit mask for /WRITE signal

```

If you want to be more efficient, you can save an instruction by defining two additional constants:

```

RADDR    LONG    $04000000
WADDR    LONG    $08000000

```

If instead of LADDR you used RADDR, /READ will be asserted on the bus along with the low byte of the address on A0-A8.

If you used WADDR, /WRITE would be asserted with the address.

Random Access Read/Write

Here is a code snippet showing how to read one or more bytes from a random byte address:

(note code cannot cross page boundary)

```

mov    page, xaddr      ' 002xxxxyy --> 00002xxx
shr    page, #8         ' zero's end up where p26&p27 are
mov    outa, page       ' set output register
mov    dira, bsel wz   ' enable outputs sets Z=0
muxz   dira, ctrl

' Output low byte of address

mov    offs, xaddr     ' get bus address
and    offs, #$ff      ' isolate lowest byte
shl    offs, #8        ' move it onto P8-P15
mov    outa, offs      ' output it

readn  mov    dira, rsel ' READ
       nop                    ' needs pipeline delay

lp     mov    data, ina
       wrbyte data, haddr
       add    haddr, #1
       add    outa, #$100
       djnz   xlen, #lp      ' misses optimal hub access

```

Burst Mode Read/Write

The burst read / write mode example is too long to reproduce here, however you can find an example of burst reading 256 bytes at 20MB/sec using three cogs in the following XGA_DEMO1 files:

- `xmm_addr.spin` generates addresses for the two reader cogs
- `xmm_read.spin` read data from data bus and pack it into longs, interleaved

How does it work?

`xmm_addr`:

- sets the page register
- sets a rendezvous time for the reader cogs to sync with it
- at rendezvous time, `xmm_addr` starts generating an interleaved address pattern

Currently the pattern is:

```
$00  
$80  
$01  
$81  
...  
$7F  
$FF
```

The pattern is generated by simply adding alternately adding \$000080 and \$00008100 to OUTA

`xmm_read`:

- waits for the rendezvous
- fold every second byte into a series of longs using an unrolled loop storing the first 128 bytes
- other instance of `xmm_read` is interleaved with it, reads the second 128 bytes of 256 byte burst

VGA256 Subsystem

Theory of operations – Standard Parallax VGA

The VGA video output from the Propeller assumes that one of four eight pin groups of a Propeller will generate five signals using three 2 bit DAC's.

The signals are:

- Red – P22, P23 on the demo board (P23 is the MSB)
- Green – P20, 21 on the demo board (P21 is the MSB)
- Blue – P18, 19 on the demo board (P19 is the MSB)
- Hsync – P17 on the demo board
- Vsync – P16 on the demo board

Any of P0-P7, P8-P15, P16-P23 and P24-P31 can theoretically be used for VGA, however using P24-P31 can cause issues due to serial communications and the i2c boot eeprom.

All current drivers assume that Vsync is on the least significant pin of the VGA pin group, and that Hsync is the next higher bit. Doing VGA this way has the advantage of Hsync and Vsync being locked to the same dot clock as the color data, and of only using eight pins.

Theory of operations – Morpheus 256 Color VGA

For Morpheus, I decided to use a more advanced, and more difficult scheme – which allows Morpheus to display 256 colors compared to 64 colors with the Parallax mode. I used a 3R 3G 2B configuration, with three bit DAC's for Red and Green, and an additional two pins for Hsync and Vsync.

The signals are:

- Red – P21, P22, P23 on Morpheus (P23 is the MSB)
- Green – P18, P19, P20 on Morpheus (P20 is the MSB)
- Blue – P16, P17 on Morpheus (P17 is the MSB)
- Hsync – P24 in a different pin group from the DAC inputs
- Vsync - P25 in a different pin group from the DAC inputs

Morpheus VGA drivers can take two approaches for generating the Sync signals:

A) Bit-Banged Sync signals

This is the approach taken by MorphVGA256_Diag, which implements a 640x240 two colors per tile out of 256 possible colors bitmap display with the bitmap stored in the HUB.

I first wrote this driver in April of 2008, running it on a modified Propeller board. Initially I could not get it to synchronize, as the PLL generated clock is not exactly synchronized with the Propellers clock. Later I was able to get it sync, by using WAITVID's of a single pixel length so that I would be able to manipulate Hsync and Vsync “pretty close” to the dot clock.

Unfortunately this leaves some “jitter” that is visible on the top displayed lines of most monitors.

B) Multi-cog synchronized drivers

This is the approach taken by the preliminary 1024x768 four colors out out 256 color driver that is used by XGA_DEMO1.

This type of driver uses at least two cogs – one or two RGB line display cogs that update the video DAC's, and a Sync generator cog.

All the cogs are synchronized such that their PLL driven dot clocks are synchronized to the dot clock frequency, and that the video data is shown at the appropriate time between the framing Hsync and Vsync signals.

Here is the cog usage for XGA_DEMO1:

XGA_DEMO1 – the main program

XMM_PASM - generic XMM read/write server cog, blocks during burst reads

XMM_ADDR - address generation cog

XMM_READ - burst read cog, one of two

XMM_READ - second instance, offset by four clock cycles from first one

XGA_SYNC - video HSYNC and VSYNC generator

XGA_DISP - displays video even scan line information

XGA_DISP - second instance, displays odd scan line information

I will be merging XMM_ADDR, XMM_SYNC and XMM_PASM into a single cog as tile allows, freeing up two cogs for user code.

The video synchronization code is based on Chip's VGA high resolution text driver's code, I tried to use exactly the same code to synchronize the three display generation cogs. I needed to use three as my 256 color VGA capability means that HSYNC and VSYNC have to be driven from a different video circuit as I use all eight bits to generate eight bit color (3R 3G 2B).

Currently some of the time the two XGA_DISP cogs don't synchronize - despite the counters/PLL's being programmed the same, despite allowing time for the PLL's to settle. I am at a loss - I either have had my nose buried in the problem too long, or there is some as yet undetermined reason for the issue.

I have come up with two ways of fixing it, as soon as time allows:

- disable the PLL in the video generator, and stick to 40MHz dot clocks or lower
- detect if the two DISP cogs are not sync'ed correctly to the SYNC cog, and keep retrying the sync code until they are perfectly in sync

Syncing up the memory burst engine was a piece of cake in comparison!

Driver Theory of Operation

The display engine is composed of a memory burst read engine, and an XGA display engine that displays the contents of two scan line buffers (even & odd)

- XGA_SYNC generates HSYNC, VSYNC, and requests that XMM_ADDR generate addresses for the screen scan line it specifies
- XMM_ADDR waits for the correct start time, and starts generating interleaved addresses for the two XMM_READ cogs
- XMM_READ (2 cogs) cogs read from the bus, interleaved with each other, and assemble longs for the two scan line buffers
- XMM_DISP (2 cogs) cogs flip between reading from the bus, and displaying a scan line buffer - when even cog reads, odd displays, and vica versa

Currently, XMM_DEMO1 writes bytes to the screen buffer one at a time one per Hsync. Writes are blocked whenever the ADDR/READ cogs are active.

MEM+ Subsystems

There may be up to four Mem+ boards in a Morpheus system.

- only ONE board may have its MCP23S17 PIO subsystem installed
- only ONE board may have its SD card subsystem installed
- only ONE board may have its serial subsystem connected to H-COMM1

Address Decoding

There are four possible address ranges where a Mem+ board may be placed:

- \$000000-\$1FFFFFF – **first socket must not be populated**
- \$200000-\$3FFFFFF – **remove the 512KB ram** from Morpheus if addressed here
- \$400000-\$5FFFFFF – available for first Mem+ board
- \$600000-\$7FFFFFF – available for second Mem+ board

512KB-2MB Static RAM

You do not have to populate all the memory sockets on a Mem+ board – as a matter of fact, you can start using it merely for its RS-232 programming interface! I recommend that you install memory chips from right – to – left, with the first chip being installed right next to MORPHBUS.

Remember, on a Mem+ board addressed at \$000000 you may not install the chip closest to MORPHBUS!

SD Card Interface

The SD card interface on Mem+ connects to P4-P7 on CPU1 of Morpheus, and uses 12K pullup resistors as recommended by the fsrw provided schematics.

To use it, insert a locking nine pin right angle Molex connector facing to the back of the board (connector side) into the nine pin male header on Mem+, and insert an SD card into the Molex connector.

Alternately, the SparkFun SD card connector breakout may be inserted one pin hanging off the right side of the nine pin female header.

You will need to adjust `fsrw` so that it matches the pinout used on Mem+

- P4 is the /CS signal – pin 1 on an SD card (active low chip select)
- P5 is the CLK signal – pin 5 on an SD card
- P6 is the DO signal – pin 2 (DI) on an SD card
- P7 is the DI signal – pin 7 (DO) on an SD card

Remember, only one Mem+ board in the system may have its SD card interface populated!

Serial Port

If you don't have a USB PropPlug, or you need another serial port for your system, you can populate the RS232 interface on Mem+.

The interface uses a MAX3232 3.3V RS232 transceiver, and if you use a stacking connector and install Mem+ right on top of Morpheus, plugs right into the H-COMM1 connector on the Morpheus below it.

You can disable the /RESET circuitry that is triggered by your choice of the DTR or RTS signals on the DB9 connector by removing the REN shunt near the bottom of the Mem+ board.

If you are not using the serial circuit as the Morpheus programming port, just connect pins 2 and 3 from where the header would be to any two available pins, and presto, you have another serial port!

Only one Mem+ board (that is plugged directly into Morpheus) can be used for H-COMM1.

MCP23S17 16 bit Parallel I/O

The MCP23S16 16 bit parallel I/O expander uses pins P0-P7 on the EXP1 connector on Morpheus.

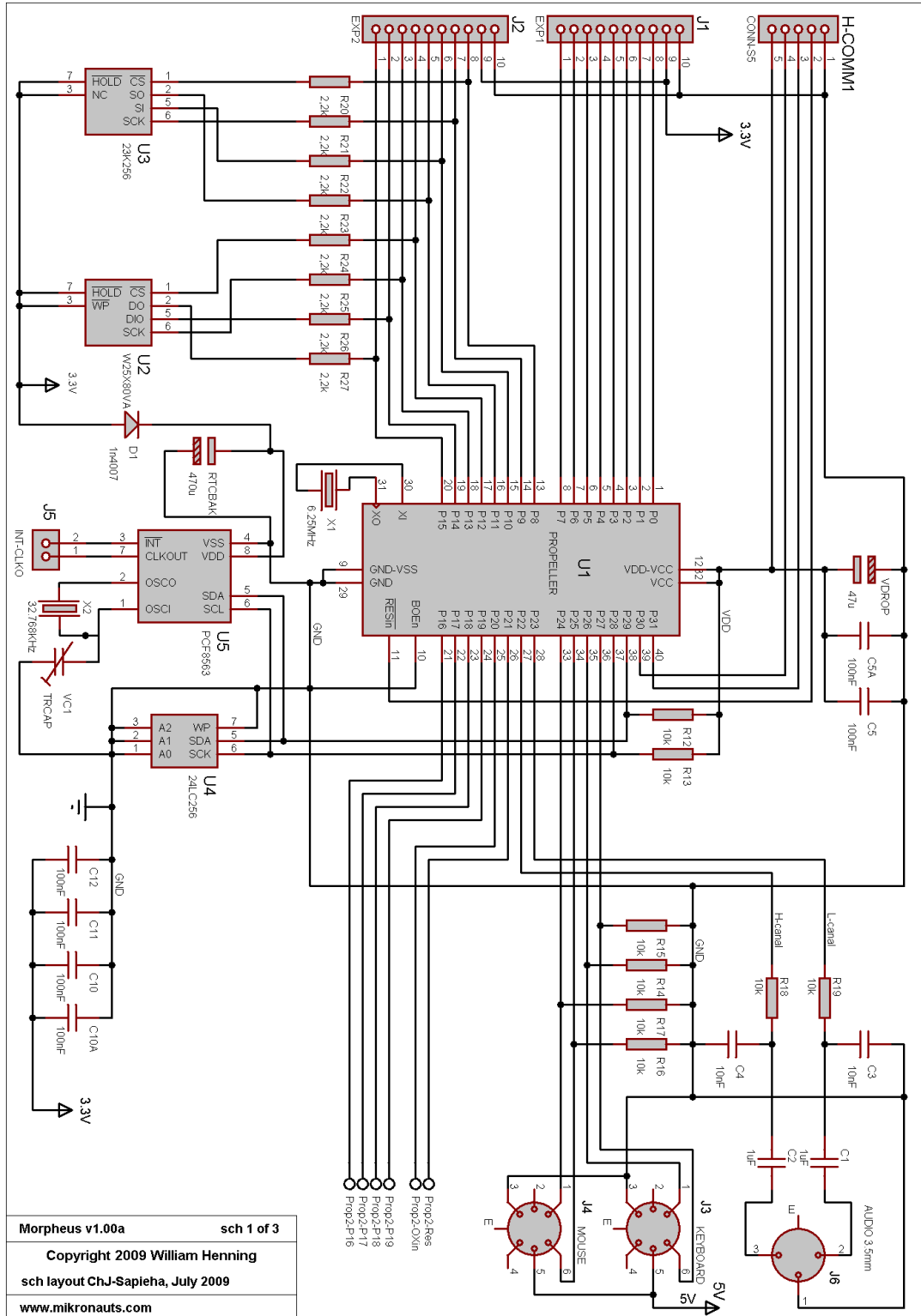
The two eight bit I/O ports are brought out to two ten pin headers with the following pinout:

Pin 1: bit 0	Pin 2: bit 1
Pin 3: bit 2	Pin 4: bit 3
Pin 5: bit 4	Pin 6: bit 5
Pin 7: bit 6	Pin 8: bit 7
Pin 9: +3.3V	Pin 10: GND

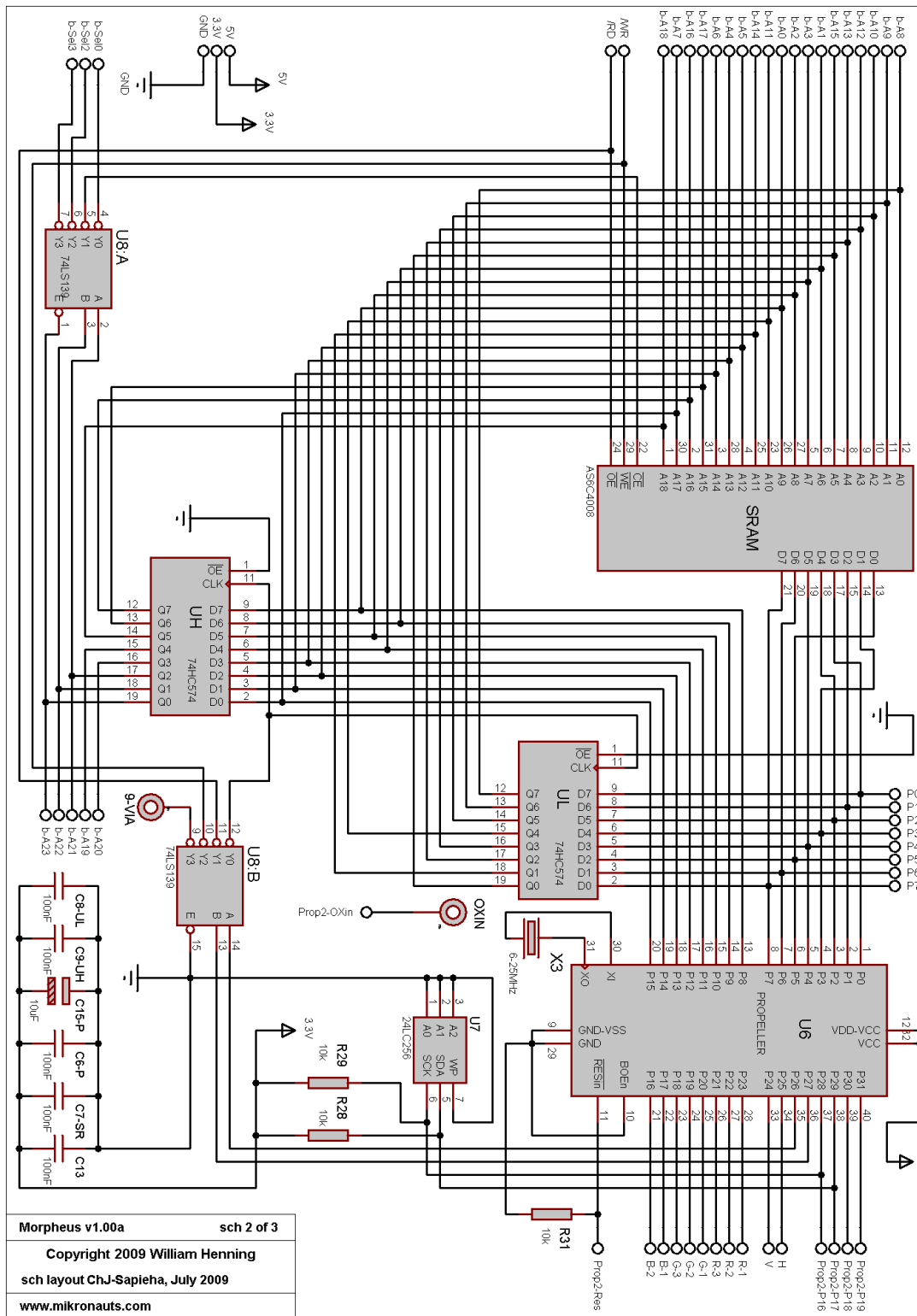
Watch out! They have pin one on different corners!

Remember, only the Mem+ plugged directly into a Morpheus may have its 16 bit I/O installed.

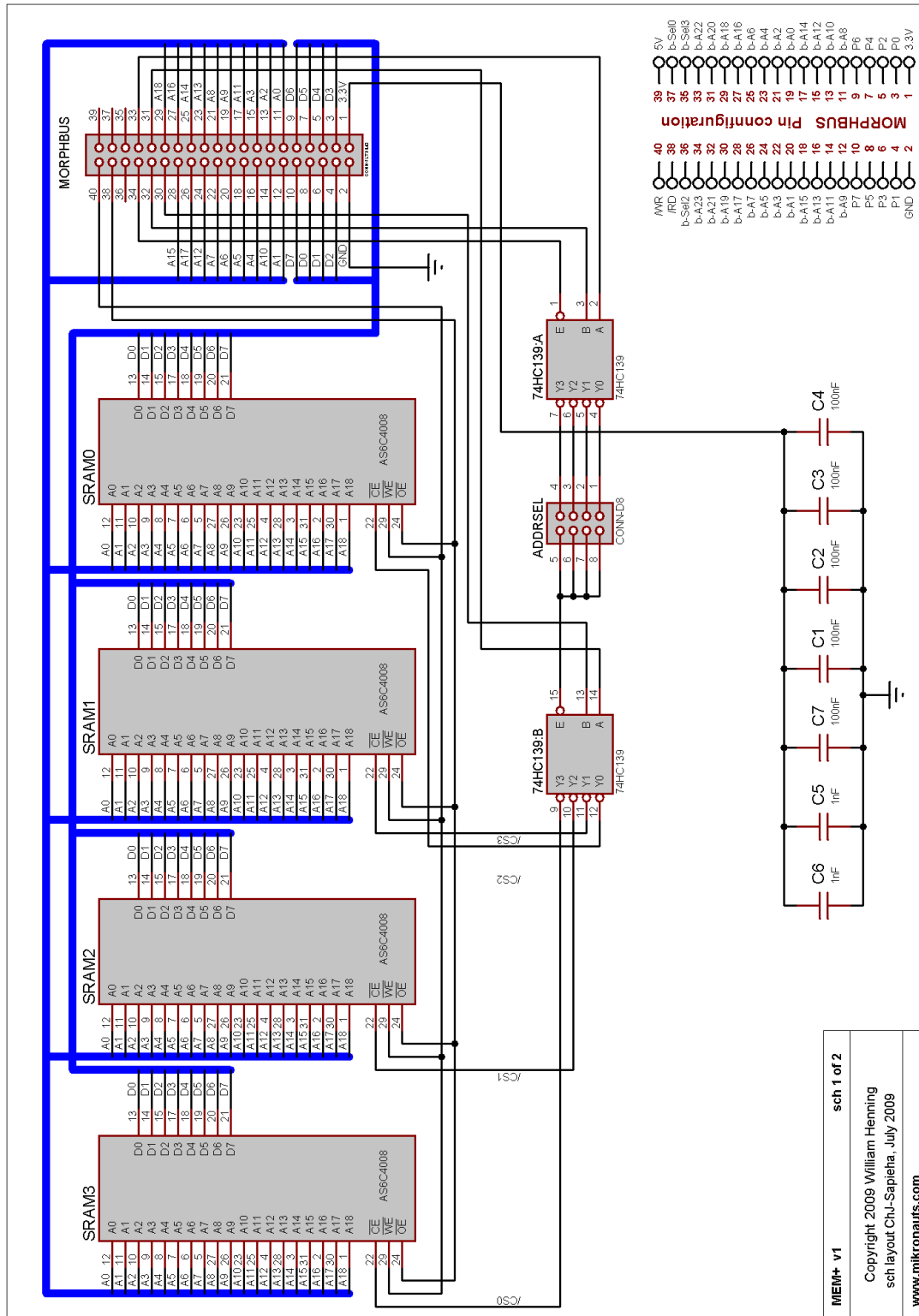
Appendix I: Morpheus Schematic



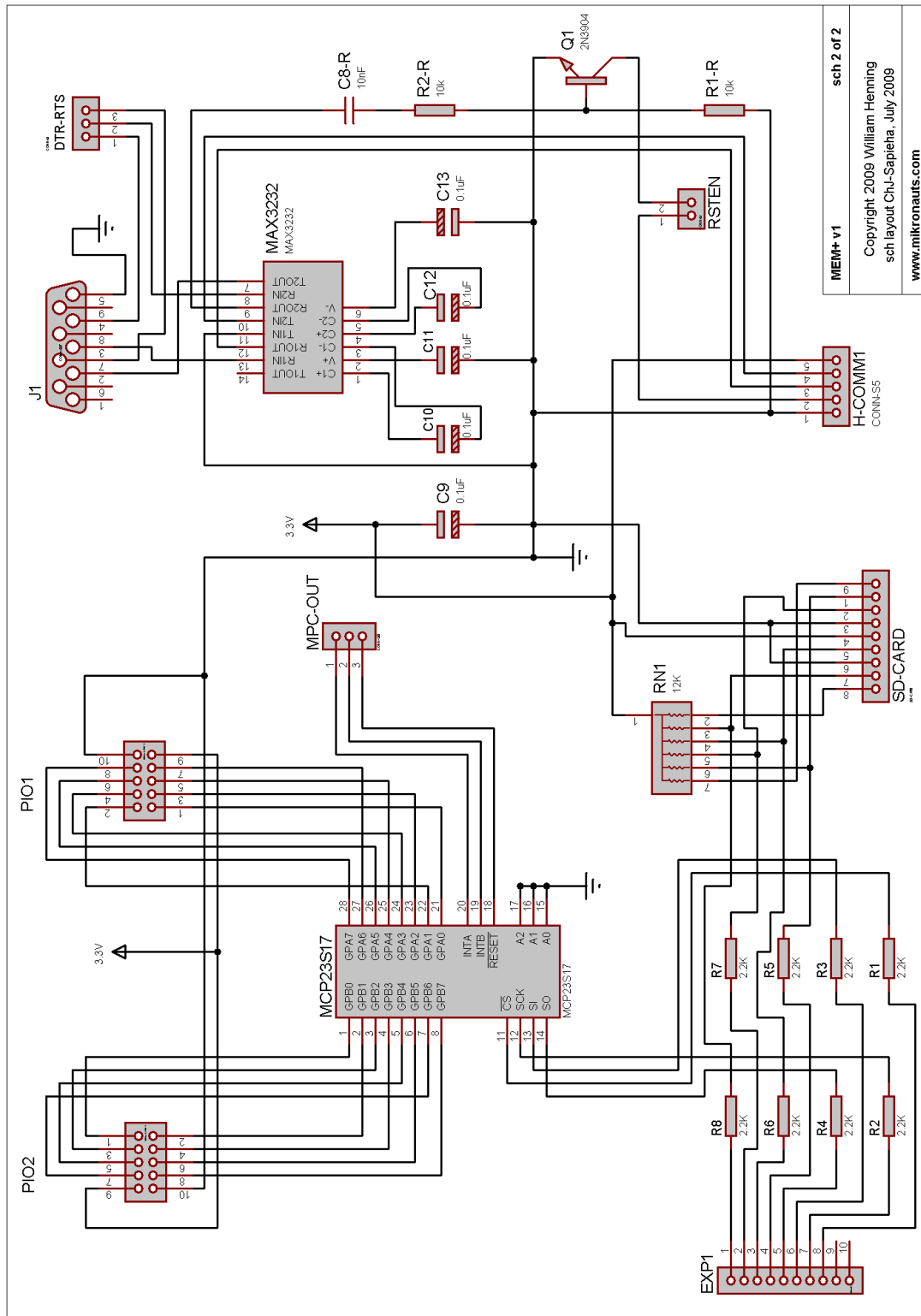
Morpheus Schematic (Page 2 of 3)



Appendix II: Mem+ Schematic



Mem+ Schematic (Page 2 of 2)



Appendix III: Frequently Asked Questions

Will there be a surface mounted version of Morpheus?

- If there is sufficient demand, I can produce assembled and tested surface mount Morpheus boards. I do not anticipate Morpheus SMT for at least a year due to NRE and MOQ costs.

Do you offer educational discounts?

- Yes we do, contact us at mikronauts@gmail.com with your required quantities.

Do you offer quantity / reseller / distributor pricing?

- Of course! Contact us at mikronauts@gmail.com to discuss your needs.

Will you be writing more VGA drivers?

- YES!

Will there be a 100MHz version of Morpheus?

- As soon as we get faster memory we will try qualify Morpheus for 100MHz operation
- We will offer “Turbo Morpheus” kits and assembled and tested systems after it is qualified and the drivers have been re-tuned

Is Morpheus an “Open Source” design?

- **No.** Morpheus is a commercial Propeller single board computer and line of expansion boards
- The Morpheus schematics, board design, bus design, pinout, layout, documentation and all software are copyrighted and may not be reproduced without permission.
- The Morpheus System Architecture is documented in this manual, and printed circuit boards are available for sale from us cheaper than you could have them made for you in small quantities.
- You are encouraged to use Morpheus boards in your applications, as long as you buy the Morpheus boards from us or one of our authorized dealers or distributors. Quantity pricing is available on request.
- If you would like to make Morpheus compatible I/O boards, please contact us so we can come to some mutually beneficial arrangement.
- You may not make Morpheus clones or variations and sell or otherwise distribute them without entering a licensing agreement with us. It took a large investment in time and money to bring Morpheus to market, an investment that we need to recoup. We are selling all Morpheus products at very reasonable prices to make them quite affordable – which also makes it fairly unprofitable to clone them. Unlicensed Morpheus derived boards will be prosecuted.
- Morpheus drivers, diagnostics and utilities will be dual licensed – they will be licensed under the GPL so any educational or personal use is fine – so is commercial use, if you also license your source code under the GPL. If you don't want to give away your source code, contact us and we'll work out a reasonable licensing deal for you.

Appendix IV: Acknowledgements & Recommendations

- Chip and Parallax for the Propeller
- Sapiha for the great schematics he made for me after I lost the originals, and much more!
- Jim for a lot great advice
- TrapperBob for a ton of great feedback
- The great fast long folding code: ariba, jazzed, PhiPi and lonesock from the forums
- fsrw Tom Rokicki, Jonathan Dummer, and all the contributors
- OBC for organizing UPEW 2009 where I unveiled Morpheus
- All the forum regulars for their contribution to the Parallax forums!

Recommended Projects

- Sphinx <http://sphinxcompiler.com>
- FemtoBasic <http://obex.parallax.com/objects/28/>
- Catalina <http://forums.parallax.com/forums/default.aspx?f=25&m=339139>

Data Sheets

Propeller:

<http://www.parallax.com/Portals/0/Downloads/docs/prod/prop/WebPM-v1.1.pdf>

<http://www.parallax.com/Portals/0/Downloads/docs/prod/prop/PropellerDatasheet-v1.2.pdf>

SRAM: <http://www.bsi.com.tw/product/BS62LV4006.pdf>

FLASH: http://www.winbond.com.tw/NR/rdonlyres/0971C40C-F202-49CA-90AF-0F0268ECF0E5/0/W25X10L_W25X20L_W25X40L_W25X80L.pdf

RTC: <http://www.standardics.nxp.com/products/pcf/datasheet/pcf8563.pdf>

EEPROM: <http://ww1.microchip.com/downloads/en/DeviceDoc/21203P.pdf>

SPI RAM: <http://ww1.microchip.com/downloads/en/DeviceDoc/22100D.pdf>

74HC139: http://www.nxp.com/acrobat_download/datasheets/74LV139_4.pdf

74HC574: http://www.nxp.com/acrobat_download/datasheets/74LV574_4.pdf

MAX3232: <http://datasheets.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf>

MCP23S17: <http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf>