0

Board Architecture

The most important decision was to retain the Spartan-6 LX9 in the TQ-144 footprint from an earlier design. The footprint has limited the number of available IO pins. This in turn made it necessray to share some pins among several on-board subsystems.

Shared Signals

1. I2C bus (SCL, SDA). Also used for the MAC EEPROM 24AA02E48T-I/OT, temperature sensor TMP100NA, current sensors INA219AIDCN (three pieces), the real time clock M41T62LC6, and the HDMI monitor. The latter connection can be selectively enabled, so you do not need to worry with the unknown monitor doing weird things to the I2C bus.

2. Four GPIO pins are routed to all three PMODs. They have 2k pullups to 3.3V. They can be used to implement your own UART or I2C on the PMODs or the expansion board.

3. SPI select 3, 4, 5, and 7. They are used to select three PMODs and the Nordic wireless. They are yours to grab if you are not using them for their primary uses.

4. Radio IRQ and CE. Their primary use is for the Nordic radio. They are also connected to the Sparkfun WiFi socket, where they are used for UART.

5. The main SPI bus labeled SPI_West. There is only one primary SPI bus on the entire board. It is connected to the FPGA configuration flash, the W5500 Ethernet chip, the microSD card, all three PMODs, and the Nordic radio. Do not expect a very high frequency from this bus which is heavily loaded. A few MHz is probably feasible. This needs to be tested with real hardware.

Video

The increase of the avaiilable memory will allow to increase the color depth to 256 colors, using 1 MB for the 1024*768 8-bit color pixel map. Alternatively, we can use an even larger display size with 16 colors.

The original VGA connecor was replaced with the HDMI connector to decrease the number of interface wires. This will necessitate using the VGA-to-HDMI firmware, kindly contributed by Magnus Karlsson. The LX9 FPGA resource utilization has grown from about 50% up to 76%, still leaving room for future firmware development.

Memory

The memory has been increased from 1 MB on the original 2013 FPGA Oberon System to 4 MB. This will allow to run significant applications in the 3 MB available after devoting 1 MB to the display framebuffer.

The memory type was canged from ASRAM to ZBT RAM. The nominal speed of the ZBT chips is twice that of ASRAM. Even more important, ASRAM relies on timing margins rather than on the clock. The effective ASRAM speed was only 25 MHz with the original board, despite using the nominaly 100 MHz chip. ZBT RAM does not impose such limitations. (A separate page is devoted to discussing the ZBT.)

Two SD cards

Two SD cards are motivated by the desire to provide a removable medium readable by a host PC. There are a few obstacles to overcome: (1) The original proprietary Oberon file system is hardly compatible with PC hosts; (2) the original file system is small; (3) when the Oberon System SD card is removed, the Oberon System cannot work anymore. (The latter would be true with Linux or Windows as well.)

Since the present Oberon System disk is sufficient to run the System, a conservative solution is to leave it intact even though it does not utilize the entire SD. This misfeature is not critical, though it points at the Oberon System design deficiency. Providing a separate SD can address the need for removable media. We call it the Data Disk, as opposed to the System Disk which is meant to stay put while the System is running. Since the System Disk is not meant to be regularly removed, we used the microSD footprint, whose replacement requires some dexterity. This will not impact the functionality, if the System Disk moslty stays in place.

Due to the pin shortage, the System Disk is connected to the main SPI bus. It will allow to utilize the original software drivers without any change.

The Data Disk is a full size SD which is easy to handle. The dedicated QSPI interface will help achieve high speed. We envision that the Data Disk will employ a FAT filesystem utilizing the entire card.

It will be also possible to use the full size SD for the System Disk and to disregard the microSD until the FAT file system gets developed.

Single-Core versus Multi-Core Design

RiskZero will most likely remain a single core system due to the limited FPGA capacity. Multicore designs have to wait for an FPGA upgrade. The following projects will benefit from the multiple CPUs.

In each case the auxiliary CPU can execute a simple program from the embedded BRAM. In most cases a few kilobytes would suffice. Such programs can be compiled, using the MODULE* notation. (Standalone software modules.)

1. Data Disk can be connected to a dedicated slave RISC5 core. The data can be then formatted by the slave RISC5.

2. ADC chips can be handled by a dedicated CPU, able to detect data patterns such as pulses, bursts of noise, or abnormalities indicative of monitored equipment failures.

3. Communication channels can be served with their own dedicated CPUs.

4. Display framebuffer can be handled with its own dedicated CPU using a command interface.

1

Flash Chip Notes

Since the bitstream file size is below 512 kB, half a meg flash would be sufficient. It is quite hard to find such small flashes anymore. A large flash (32 MB or more) can also be used for Oberon System disk beyond the first half a meg. Implementing the Oberon disk in a NOR flash may be attractive because the flash chip must always be present to boot the FPGA. A flash serving both roles would save both the board space and cost.

This idea will be practical if the efficient file sharing utility is available (such as the PClink module), because flash cannot be brought to a PC for reprogramming.

Example flash chips: S25FL256LAGNFV010 (32Mx8) costs less than $3, which is less than the SD card. S25HL512TDPNHI010 (64Mx8) costs $8, which is still below the SD.

HDMI Notes

Magnus Karlsson kindly provided the first draft of the Oberon SoC with the HDMI output. The LX9 utilization increased from 58% for the original SoC to 76% for the SoC with HDMI. It is a very optimistic result. Thank you Magnus!

LCD to HDMI converter is also available from Open Cores. It is a parametrized IP that does not use any setup. Default timing setups are provided for different display resolutions. It is not clear whether or not this IP can fit into the LX9.

https://opencores.org/projects/lcd_to_hdmi_output_ip

Deeply Embedded Notes

The minimum number of FPGA pins can be reached when the system disk is implemented in the free space remaining in the FPGA boot flash, which needs to be present anyway. The memory can be either (1) Serial RAM or (2) HyperRAM.

Less than 1 MB will suffice if there is no GUI, what is OK for deeply embedded control.

1) Serial SRAM IS62WVS5128FALL-16NLI (512k * 8) will require six pins (CS#, SCK, D0, D1, D2, D3),. With some FW effort it probably can share the SPI bus with the flash, but this may be tricky. The serial clock speed of 20 MHz will be OK for control applications.

2) HyperRAM is much faster. Being a DRAM it will benefit from a cache, which has already been developed by Joerg and Magnus. HyperRAM does not require the DRAM controller FW because it has one on chip. It is using 12 pins @1.8V, or 11 pins @3.3V.

3) Cache takes lots of BRAM. It is OK if no other FW needs BRAM. It is not OK if other FW needs it, what is mostly true with our instruments.

FTDI Notes

DONT use FT_Prog from the FTDI web site on the original Digilent cable, as it can trash the cable firmware! The Digilent EEPROM contains private data that cannot be handled correctly by FT_Prog.

The Digilent JTag cable uses FT2232. Its configuration EEPROM contains private data to be recoginzed by Xilinx ISE/Vivado. The following page provides step-by-step instructions how to restore a trashed Digilent cable.

The instructions can also be used to patch the on-board FT2232H to behave like a Digilent cable under ISE/Vivado.

Search the web for "FT2232 to Digilent JTag for Xilinx FPGAs (ISE/Vivado)". The direct link to the page is below.

https://gist.github.com/rikka0w0/24b58b54473227502fa0334bbe75c3c1

Networking Notes

It would be good to operate the board remotely and at the same time not open the door to hacking. Here is the idea. The buffers received from the remote host can be passed to a resident command interpreter which will perform the same actions as the on-screen interpreter would do. Namely, if the command has the form Module.Proc, then it will be called the usual way. The command interpreter will use the module loader the same way as the GUI would do. If the Module.Proc does not exist then the command will fail.

In addition, the network interpreter can use a white list of permitted modules. If the invoked module is not on the white list then the Module.Proc will not be passed to the module loader.

The workstation can be thus operated remotely in a restricted and safe way. This would be ideal for a deeply embedded System w/o a GUI.

SkuTek Instrumentation - 150 Lucius Gordon Drive, W. Henrietta, NY 14586-9687		
Title: Notes and suggestions		
Size: A	Document Number: RiskZero LX9 Single Board Workstation Computer	Rev: 0
Date: Monday, September 28, 2020	Sheet 25 of 40	

The Oberon SoC

Currently, the Oberon SoC exists as a complete reference design occupying the entire Spartan-3 FPGA. There are two possible paths to use this design to implement our own SoCs.

1. Adopt the original framework. Add own peripherals according to the framework rules.

2. Encapsulate the Oberon SoC into a component, resembling the Microblaze Controller System by Xilinx, used and discussed by Pong P. Chu in his textbooks.

Path 1. Adopt the Original Oberon SoC Framework

This path looks tempting. What is more natural than adding new peripherals to the existing framework? But it has substantial drawbacks.

1. This framework was not designed for handling substantial peripherals, which were hardly possible with the original Spartan-3 board. Only the simple controllers were implemented (SPI, PS/2, RS-232, LEDs, a few GPIOs), which needed a few registers each. As few as 16 bytes were provided per peripheral. This little space is too restrictive for more complex devices.

2. While it is possible to change the register memory map to provide more space per peripheral, it is not clear if the emulators will follow the suit. It is thus the emulators rather than the SoC itself which are holding off any substantial modifications

Path 2. Turn The Oberon SoC Into a Component

The alternative path is to encapsulate the SoC "as is" and turn it into the RISC5 Controller System (RCS) resembling the Microblaze Controller System (MCS) by Xilinx. Internally the RCS will stay very close to the present SoC, providing for good compatibility with the emulators.

We can provide an RCS interface either identical or highly similar to the MCS interface. The entire Fpro framework by P.P.Chu will then become available to help implement substantial projects.

Coding Style of The Oberon SoC

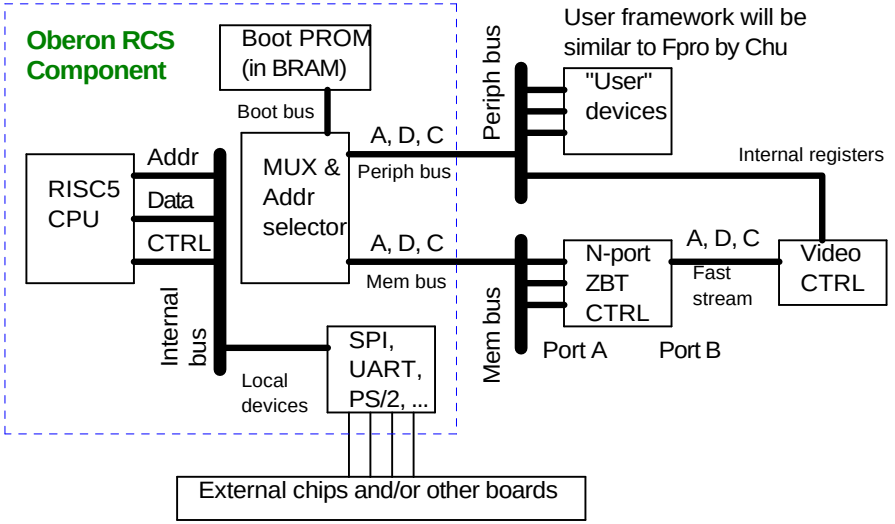
The original Oberon SoC was written in a terse Verilog using short (mostly single letter) signal names and few comments. The SoC is much more cryptic than it could have been. On the positive side, the SoC is of a small size. Restructuring the code is entirely feasible.

The Problem with Oberon SoC Emulators

The emulators implement the official Oberon SoC with all its hardwired addresses. The emulators do not seem to envision a framework supporting a modified SoC. Restructuring the SoC into a modular firmware framework is unlikely to be followed by the emulators.

This being the case, emulators are still going to be of some help if the MCS is internally close to what the emulators are providing.

High Level View of The RISC5 Controller System



Internally, the RCS will be a close follow up of the original Oberon SoC with the same internal addresses. It will stay highly compatible with the legacy Oberon SoC and with the emulators. The SW may require some tweaks, but otherwise it will be largely portable between the emulators and the RCS Component.

Both the video and the ZBT memory controller will be moved outside the RCS to avoid RCS modification when either of these is modified.

The Interface Buses will be active only for the addresses of their respective ranges. There will be at least FOUR buses.

1. The Internal SoC Bus for accessing local devices (SPI, UART, PS/2, ...) provided for backward compatibility. Local devices will be enclosed within the RCS.

2. The External Peripheral Bus accessing the non-RCS devices. We will adopt the Fpro bus design by P.P.Chu either "as is", or with only a few modifications.

3. The RAM Memory Bus connected to Port A of the Multiport ZBT Controller. The Video controller will access the video data through a separate Port B of the Multiport Controller. The Internal Registers of the Video Controller will be mapped to the External Peripheral Bus.

4. The Bootloader Bus for accessing the Bootloader code in the BRAM (named PROM in Oberon documentation). Both the bus and the PROM will be enclosed in the RCS.

SkuTek Instrumentation - 150 Lucius Gordon Drive, W. Henrietta, NY 14586-9687

Title: Oberon SoC Notes

Size: A | Document Number: RiskZero LX9 Single Board Workstation Computer | Rev: 0.1

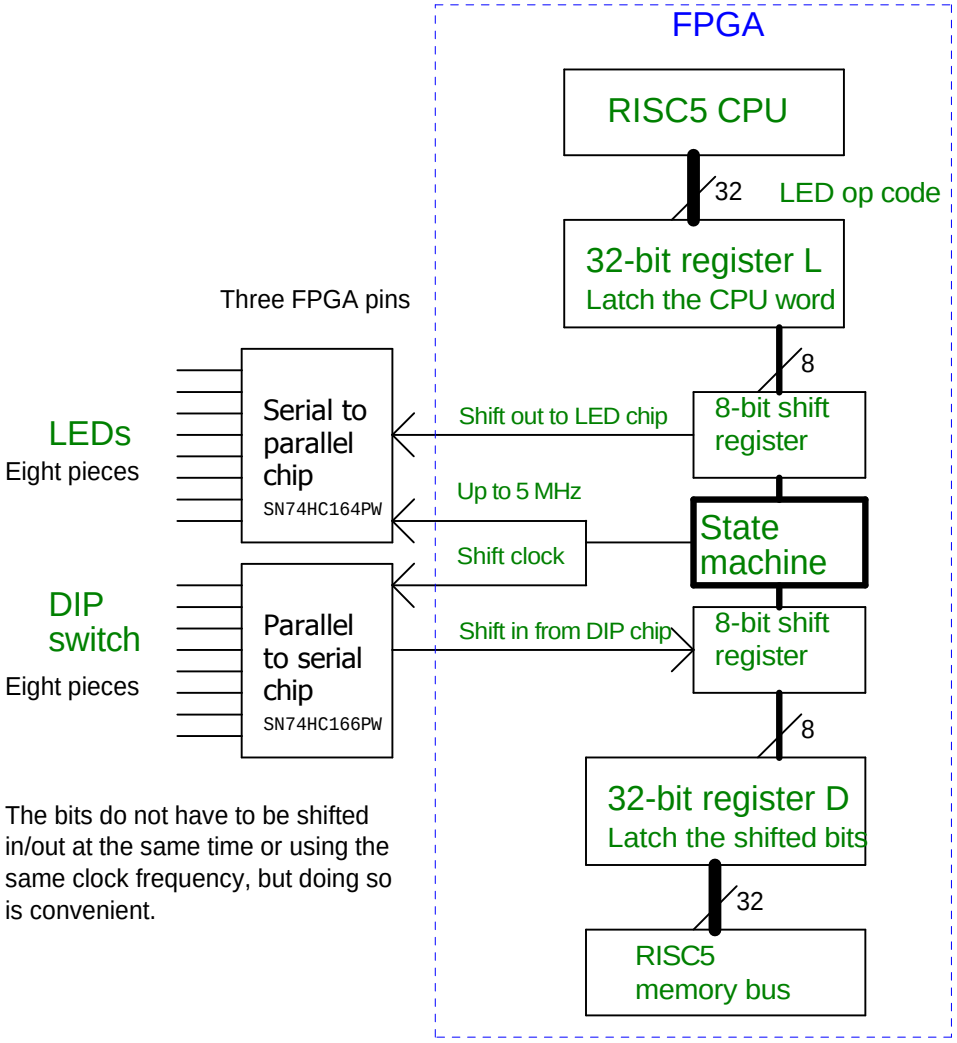
Date: Monday, September 28, 2020 | Sheet 26 of 40

3

Shift Regiser Notes

The parallel / serial chips provide additional pins which are needed for the LEDs and the DIP switch. The LEDs are important for the software diagnostics. RISC5 even uses a special CPU instruction to show the software status on the LEDs.

The DIP switch is useful for setting options in either the FW or SW. For example, Pepino uses one of the switches to choose the display color scheme either white text on black background, or the opposite.



The bits do not have to be shifted in/out at the same time or using the same clock frequency, but doing so is convenient.

Shift Firmware Notes

If we had 16 free pins, then each LED could be driven with a pin connected to one of the Register L bits. Likewise, each DIP circuit would set one of the bits in the Register D. The connections would be hardwired, direct, and asynchronous.

Since we do not have these sixteen pins, we use three FPGA pins to drive the parallel / serial chips. The firmware needs to serialize the bit transfer in both the IN and OUT directions.

The parallel / serial chips provide a very simple version of SPI, but w/o the full SPI complexity. The bit shift portion of the FW can be borrowed from a SPI controller. The chip select part of SPI is not needed because we are serving a single chip in the IN or OUT directions.

Periodic State Machine

Here I am proposing a Periodic State Machine, which will shift eight bits IN, and eight bits OUT every millisecond or so. Running more often than a millisecond is not warranted because these are human interface devices. The shift clock can run up to about 5 MHz.

The state machine will run periodically and automatically. Both the IN and OUT will be shifted at the same time. The memory-mapped Registers D and L will mirror the content of the physical devices with a latency of about a millisecond. The software needs neither to control nor to customize the shift operations.

Automatic mirroring of the chips will let the DIP affect either the FW or the SW settings with a latency of about a millisecond.

SkuTek Instrumentation - 150 Lucius Gordon Drive, W. Henrietta, NY 14586-9687		
Title: Shift register notes: LEDs and DIP switch		
Size: A	Document Number: RiskZero LX9 Single Board Workstation Computer	Rev: 0
Date: Monday, September 28, 2020	Sheet 27 of 40	

Overview of the Problem by Joerg Straube

Joerg <joerg.straube@iaeth.ch> wrote on Sept/25: The task of an OS is to provide a rather universal API of the Wifi functionality to the upper client layers. So, the API of the Oberon System to provide WiFi should be carefully crafted. „reduce it to the max to be useful“

The task of the driver SW is to map this general, universal, OS Wifi API to the chip's specifics.

You can not assume that the different chips offer the same low layer interface.

As an example: For the TCP/IP functionality in the Unix OS the „sockets“ API seems to be a common ground. Whenever there is a new Ethernet/IP chip, the chip manufacturer often provides his „sockets“ implementation to ease the OS integration.

As the Oberon system is NOT Unix, we have two tasks:
1) invent a Wifi API for the upper layers
2) map this API to the chosen chip's lower layer

Use Example #1: AX88180 by ASIX www.asix.com.tw

AX88180 is an Non-PCI 32-bit 10/100/1000M Gigabit Ethernet Controller with a 32-bit SRAM-like interface. For the host CPU it is either a piece of addressable RAM, or a FIFO. (It supports both addressing modes.) It is a traditional MAC chip, with all the traditional features like CRC calculation, autonegotiation, etc. It does not provide the on-chip TCP stack. So it is an old style "dumb" interface chip. (Pardon ASIX.) It does not interpret the data buffers. It just passes them both the Tx and Rx ways.

All the SW is implemented on the host. The low-level driver is responsible for setting up and controlling the interface: duplex vs. half duplex, wire speed, enabling the CRC, enabling jumbo packets, etc. The driver is also setting up the PHY connected to the wire side of the MAC.

The SW driver is taking care of the "chip specifics". The upper layers do not deal with the specific chip. They delegate the specifics to the driver. For those layers, all the drivers and all chips look the same. The upper layers deal with the content of the buffers, but not with the specifics of sending or receiving those buffers.
This scheme provides the chip independence because, lets be honest, all MAC chips are similarly dumb. The chips do not even attempt to help in protocol handling. All the protocol handling is implemented in the same upper level SW.

This architecture has a very important benefit: There is only one instance (per operating system) of the protocol SW. It can be thoroughly debugged and optimized. There is no danger that the "helpful driver" will screw the protocol, because the driver is not even trying to help.

There is also a drawback: Protocol handling is very complex. Protocol SW is taking memory and also burning lots of CPU power. Development of this SW stack requires lots of work. In practice, all this work comes from the OS developers. Linux or FreeRTOS are well funded and provide the ready-to-use protocol SW to the application developers.

This comfortable situation breaks down in case of the Oberon OS, or other niche OS's, where the core developement labor is scarce and hardly available.

Use Example #2: Wiznet W5300 & W5500, and some WiFi chips

The niche OS's are helped by shifting all this labor intense and complex SW away from the OS and into the interface chips. Both Wiznet chips are good examples. (See other pages for more such examples in the WiFi domain.) Internally, both chips are running some sort of OS which is not specified. (Some WiFi chips run FreeRTOS, some other run LWIP.)

So now you get the encapsulated TCP/IP stack and you do not need to develop or maintain this SW yourself. You only need to "talk to the chip". This is great as long as you talk to one kind of chip. It would be great if all chips offered the same interface. But they do not.

The interfaces of W5300 and W5500 are not the same. But they are sufficiently similar so they can likely be handled by their drivers. Internally, their TCP/IP engines come from the same vendor, so we expect them being highly similar. The remaining details can be then encapsulated and hidden in the drivers.

There are a few similar chips in the WiFi domain. For example, the TI SimpleLink Wi-Fi CC3135MOD, or MicroChip ATWINC15x0. All these chips use BSD socket interface. It means that the chip specifics can be hidden inside the driver. (Setting up the link parameters, for example.) The buffers exchanged with the chip will not depend on these specifics. So the architecture described in the left column can be realized with these chips.

The modules mentioned above encapsulate a great portion of the SW which previously was running on the host CPU. The trick of using those chips is that their interfaces look similar to each other, so they hopefully can be wrapped into a common host driver.

Use Example #3: A problem with Helpful ASCII Protocols

Some other modules go a step further. They provide an ASCII command interface: (1) the AT commands, (2) the BGScript, or (3) LANCIS. The official blurb is "user friendly" and "easy". We should not be mislead by this. "User friendly" means less effort up front to achieve initial connections. (See [7] in the WRL-13678 section.) However, while simple applications will be easy to achieve, building any sophisticated application with AT commands will likely become a huge mess. "Simple result really fast" is the quintesscence of "hobby quality".

Wrapping the AT commands into a driver is of course possible. We can imagine that the driver will provide BSD sockets to the user, while using the AT commands with the WiFi module. (Or any other ASCII script which the module is providing.) However, one has to doubt the point. The WRL-13678 was built around ESP8266, which can also implement other internal SW, with or without an RTOS. So perhaps a better approach is to invest time up front in ESP8266 using its SDK, and then work with the BSD socket interface.

Reprogramming WRL-13678 is barely possible because the module does not provide enough pins for any serious interface. If one really wants to use the ESP8266, then one should rather take the WROOM module which provides access to more pins. WROOM will not directly plug into RiskZero, but it can be put on an expansion board and plugged into the expansion connector.

Conclusion and recommendations

RiskZero is not really meant for heavy duty WiFi development. The light duty "hobby quality" WiFi can be pursued with the WRL-13678 which plugs into the provided socket. The mid-duty can be pursued with a more serious module on the expansion board. A more powerful motherboard will be needed for heavy duty networking.

5

Overview of the WiFi modules

Many modules provide the on-chip TCP/IP. Sometimes they also provide the webserver and other servers on chip, plus a flash for storing the web pages.

Some modules seem to be pretty bare. Such modules rely on the user to develop the software.

Programming-wise, the WiFi modules can be roughly divided into three categories.

1. Module which uses not just a radio Network Processor, but also a general purpose MCU which can run a standalone application.

2. Modules which provide just a newtork CPU and are not capable of a standalone application. These modules are further subdivided into:

2.1 Modules implement an ASCII command language, for example the AT modem commands. (AT = "attention".) ASCII command strings can also implement a custom set of keywords other than AT strings.

2.2 Modules which require a full development system.

Electrical Interface of WiFi modules

The Host Interface of the WiFi module can be SPI, enhanced SPI (dual or quad), SDIO, or UART. I have not seen the memory-mapped modules yet.

Some modules also provide GPIO pins or ADC inputs. Such modules can implement a standalone application. They also provide a programmable CPU and internal flash. These are hardly "modules" anymore, but rather Single Board Computers.

Ready-to-use Commands, versus using a Development System

Wiznet or SparkFun modules are operated over UART with AT commands. The AT commands let you *use* the modules rather than *develop* for them. The path to initial success is much shorter. The performance will be very limited, because these are UART modules.

The modules with the BSD socket interface will provide higher performance, but the development requires more up front effort.

Leveraging the W5500 Oberon SW

The WiFi modules with BSD sockets are close to W5300 (with RAM interface) and even more to W5500 (with SPI interface). The Oberon SW developed for W5500 can be ported to the BSD-style modules. With some effort one can even develop a common API.

Assessment of the WiFi Modules

There are two defining characteristics of the WiFi modules: (1) The electrical interface and (2) the software interface.

1. Electrical Interface

Electrical interface is serial: either UART or plain SPI. The high performance SPI (QSPI or SDIO) is rare. I have not seen any parallel interface, like a SRAM bus similar to Wiznet W5300. This is because the module vendors can count on either UART or plain SPI always being offered by any CPU. Moreover, the lean interface is easier to break off to an add-on module with a cable. The module can be thus added retroactively. Multiwire memory bus would require a dedicated motherboard.

Occasionally I have seen a USB interface, but it was marked "future". I do not recall seeing I2C. Apparently SPI is better serving the role than I2C, and it is probably better supported by the CPU vendors.

UART speed can range from 115 kbauds to about 1 Mbaud, and rarely higher. This is probably due to scarcity of fast UARTs at the CPU side. SPI speed can range to 48 MHz, but usually it is up to about 25 MHz. A multiwire SPI would help, but it is rare.

2. Software Interface at the Module side

Since the module is linked to the CPU, software Interface has two parts. 1) software running on the module, and 2) software running on the CPU.

2.1.1 The lowest level software running on the module is taking care of controlling the WiFi, formatting, sending, and receiving the packets. It is always provided.

2.1.2 The "communication layer" can take one of three forms, which can be named "communication protocol".

2.1.2.a Low performance modules using UART often provide AT commands. These seem pretty similar across the UART modules, though I am not sure how universal they are.

2.1.2.b Some higher performance modules (Silicon Labs) provide ASCII BGscript, which is much better readable, but hardly universal. Another example is Lantronix. They provide LACIS, which is again a script interface. It is quite heavy weight.

2..1.2.c The third solution is to expose the module's registers and FIFOs in the form of the BSD sockets, which the host CPU is writing and reading. This is similar to the copper Wiznet W5500 or W5300.

3. Software Interface at the Host side

The SW at the host side must use the module's services. If the module is operated with AT commands, then the host must send such commands and receive the responses. If the module uses the BSD sockets, then the host must service these, using the API for sending and receiving the needed values.

It means that the Host side application depends on the module. It would be nice, though probably difficult, to define an API to "any module".

6

WiFi over UART: Two Wiznet modules

Two WizNet WiFi modules use UART rather than SPI, even though SPI is pinned out.

Both Wiznet modules have two UARTs. UART1 for communication, UART0 for debugging and upgrading the internal firmware. Communication uses AT Commands.

Both modules are SMT. Breakup boards and shields are available, for example for Arduino.

Documentation is much below the acceptable levels. The AT command reference is available, but w/o much discussion. The web server example for Arduino (built with AT commands) is sketchy and barely readable. They did not even bother to avoid Chinese characters in the screen shots.

The programming examples in C are missing. There is no SDK, even though their AT firmware must have been developed under an SDK. The data sheets are short and incomplete.

WizFi310 with UART up to 921 kbps

WizFi310. Internal TCP / IP stack: ARP, IP, ICMP, TCP, UDP, DHCP CLIENT, DHCP SERVER, DNS and other etc. It supports AP mode and Station mode. It provides AT commands. The baud rate of module serial port up to 921,600 bps. The data sheet says that MQTT and SSL are also provided. However, I cannot find any documents on these high level features.
Price: ~$14 with PCB antenna, ~$15 with antenna connector.
WizFi310 Programmer Guide is available as a Wiki page. it describes the AT commands, but in a murky and incomplete way.

http://wizwiki.net/wiki/doku.php?id=products:wizfi310:wizfi310pg:start

WizFi360 with UART up to 2 Mbps

WizFi360 = similar to WizFi310, but UART rate is up to 2 Mbps. Unlike the WizFi310, this module has several GPIO pins which can be controlled over UART with AT commands. These can be useful for debugging.
Same price with or w/o antenna: ~$3.30/ea.

WizFi360 Programmer Guide is NOT available as a Wiki page. The PDF is provided instead, similar to WizFi310. The document is pretty murky. It is not even hobby quality. It is much below.

Seeed Studio WiFi, LinkIt MT5931

This WiFi module has something to do with Raspberry Pi. It is offered from many sources (Seeed, Amazon, Arrow). Price varies from $0.99 promotion by Seed, to about ~$20 regular. It is hard to find any info besides the 1-page infomercial.

The original outlet seems to be MediaTek. Their Mediatek LINKIT ASSIST 2502 is available from DigiKey, part number 102990179-ND, where I found a 75-page manual. It is a regular development platform with the embedded OS, Eclipse, compilers, etc. Nothing seems to be ready for the end user. It looks like heavy development. Stay away.

MicroChip ATWINC15x0, WiFi (SPI up to 48 MHz)

$8 @DigiKey. MicroChip ATWINC15x0 (four different modules). Host interface is SPI up to 48 MHz. UART is used for debugging. WiFi and TCP/IPv4 are embedded on chip. It is possible to bypass these and use a TCP/IP stack running on the host.

MicroChip ATWINC3400, WiFi and BLE (SPI up to 48 MHz)

$11 @DigiKey. Very similar to ATWINC15x0. Good documentation. ATWINC3400 Wi-Fi® Network Controller Software Design Guide is very well written. It is most likely the same as the other ATWINC.

There is now a separate page devoted to ATWINC

TI SimpleLink Wi-Fi CC3135MOD Dual-Band Network Processor (SPI)

This module provides integrated Wi-Fi® and internet protocols.

$15 @DigiKey. The module provides the Network Processor but not the Application Processor. Control SW lives on the Host. It is supported with SimpleLink Developers Ecosystem for Host SW development. (I wonder which hosts are supported.)
– The antenna must be implemented on the motherboard. (Chip antenna is OK.)
– 802.11a/b/g/n: 2.4 GHz and 5 GHz
– HTTPs server, mDNS, DNS-SD, DHCP
– IPv4 and IPv6 TCP/IP stack
– 16 BSD sockets (fully secured TLS v1.2 and SSL 3.0)
– UDP: 16 Mbps; TCP: 13 Mbps

TI SimpleLink Wi-Fi CC3235MODxx Dual-Band WiFi Microcontrollers

These modules can be standalone. They provide the on-chip ARM MCU. Quite complex.

$20 @DigiKey. Network Processor AND the 80 MHz ARM M4 Application Processor with 256 kB RAM. All SW lives on the Module.The WiFi details are as above. There are four combinations of "A" and "F":
The "A" variants provide an integrated antenna.
The "F" variants provide 1 MB flash for ARM code storage.
Interfaces: SD/MMC, camera, SPI, I2C, I2S, UART, 4-channel ADC.

Murata LBWA1KL1FX

WiFi w/o TCP/IP stack on module. SDIO Host interface @25 MHz (quad SPI). Uses Cypress CYW43364. Requires Cypress SDK. High data rate 65 Mbps PHY data rate on WiFi. Module w/o antenna.. $9.50 @ DigiKey.

https://wireless.murata.com/products/rf-modules-1/embedded-wi-fi-1/type-1fx.html

Assessment: No embedded TCP stack. Stay away.

7

Silicon Labs WF111

$10 @ DigiKey. WF111 provides a low cost and simple Wi-Fi solution for devices that run an operating system and a TCP/IP stack on-board... . Bluegiga also provides WF111 drivers for the Linux operating system.

Assessment: I prefer a module which provides TCP/IP stack on module. Stay away.

Silicon Labs WF121

$18 @ DigiKey. WF121 is a self-contained Wi-Fi module providing a fully integrated 2.4GHz 802.11 b/g/n radio and a 32-bit microcontroller (MCU) platform. WF121 allows end user applications to be embedded onto the on-board 32-bit microcontroller either using a simple BGScriptTM scripting language or for more sophisticated applications; ANSI C-language.

. Embedded PIC32 Microchip PIC32MX695H.
. Host interfaces:
. 20 Mbps UART
. SPI up to 40 MHz (data sheet page 13 and 21)
. USB on-the-go (useless for Oberon System, but can be used for keyboard / mouse)
. Peripheral interfaces:
. GPIO, AIO and timers
. I2C, SPI and UART
. Ethernet with RMII interface to external PHY
. Embedded TCP/IP and 802.11 MAC stacks: IP, TCP, UDP, DHCP and DNS protocols
. BGAPI host protocol for modem like usage
. BGScript scripting language or native C-development for self-contained applications

Three ways to use the module: Host (p. 21), embedded script (p. 21), or native application.

Assessment: BGScript looks promising. All the other features will require C development which can be labor intense

Lantronix xPico Arduino and Raspberry Pi Test Boards

https://www.lantronix.com/products/xpico-wi-fi/#tab-docs-downloads

The Arduino eval board is named a "shield". the Rpi test board is named a "plate". There are example programs and User Guides in the above directory.

Arduino is probably the best way to evaluate and develop the xPico.

Lantronix xPico XPCW100x with SPI (25 MHz) or UART (921 kbaud)

https://www.lantronix.com/products/xpico-wi-fi/

About $30/ea @ DigiKey. They claim "there is virtually no need to write a single line of code, translating to a much lower development cost and faster time-to-market."

Host interface: master SPI up to 30 MHz, slave SPI up to 25 MHz, or UART up to 921 kbps.

Tx current 380 mA (integration Guide, page 40).

Software (data sheet p. 32): web server (for setting up), DHCP client or server, many others but marked "future" on p.32.

Can access the WiFi as client, and be accessed by up to four clients at the same time. It means it can be hub for up to four other WiFi devices.

Page 34: AT commands are provided. Another option: "transparent tunnel" where the host just writes the data to the module.

Page 35: Web pages are stored on the unit. Custom web pages are supported with data being sent or received to a connected host device.

Page 36: Lantronix Application Toolbox for IoT solutions (LATIS) enables customization of the end user product without the need to use an SDK or write software. LATIS is described in the User Guide Rev. L on page 86.

Page 104 of User Guide Rev. L: Customizing the embedded web site.

Command reference starts on UG p.109 and ends on p.144. It looks very heavy weight.

Lantronix xPico Documentation

https://www.lantronix.com/products/xpico-wi-fi/#tab-docs-downloads

There are two versions of the xPico Embedded Device Server User Guide, "Revision E December 2017" or "Revision L March 2018". They are quite different. The later one provides more user-friendliness. The embedded setup web pages are pretty impressive.

The Integration Guide xPico_IG.pdf provides details on the PCB layout. There seem to be the SMT modules and the 40-pin connector modules. The pin assignment is provided in the IG.

Lantronix pin compatible WiFi and copper

Lantronix xPico plug-in modules can be interchanged in the product. Product Brief says: "All members of the xPico product family use the same pin compatible interface, providing unmatched flexibility whether it is Wi-Fi or Ethernet ."

The wired Ethernet xPico uses only the UART for host interface. SPI is not provided. It will be sluggish. There is little benefit in looking in this direction.

8

MicroChip ATWINC15x0, WiFi (SPI up to 48 MHz)

https://www.microchip.com/wwwproducts/en/ATWINC1500

$8 @DigiKey. MicroChip ATWINC15x0 (four different modules). Host interface is SPI up to 48 MHz. UART is used for debugging. WiFi and TCP/IPv4 are embedded on chip. It is possible to bypass these and use the TCP/IP stack running on the host.

Documentation is available from Microchip (link above). It looks good. The app note AN2907 explains how to program the host in C to drive the ATWINC15x0.

The AN5305 Software Programming Guide looks a bit more accessible than AN2907.

AN5305 ATWINC15x0/ATWINC3400 Wi-Fi® Network Controller Software Programming Guide dated 2020 pertains to BOTH chips, the 15x0 and 3400. They say on page 1:

"All the listed examples and folder structure/files are similar for both WINC15x0 and ATWINC3400. The diagrams and console logs in this document refer to ATWINC15x0."

The architecture is similar to Wiznet W5500. Host writes to BSD-like sockets using SPI. The W5500 software in Oberon exists. I can be customized for WINC modules.

The user must write the callback functions which the driver is calling. See AN2907 page 7. Looks complex. I counted 14 header files on page 10, and eight C-source files on page 12.

Advanced ATWINC Programming Examples

Software Programming Guide dated 2020 shows basic operations (connecting, opening sockets, etc.). After that, it shows more complex examples.

Advanced Examples on Page 39 of Software Programming Guide dated 2020. Most examples are supported only on ATWINC15x0.

- Exosite Cloud Application
- Growl client – example using RESTful API over SSL (essential for IoT application)
- HTTP client file downloader application
- IoT temperature and Qtouch® sensor demo application using an Android app
- MQTT chat client – send and receive IoT information using the MQTT protocol
- OTA firmware upgrade – ATWINC15x0/ATWINC3400 firmware upgrade via OTA server
- PubNub Cloud application
- SSL client connection – Set up an SSL client connection
- Weather client – get the current weather information from the network provider and utilize the IO1 sensor device
- Wi-Fi serial – useful for chatting or controlling a remote device
- Enterprise security – for details, refer to the ATWINC Enterprise Security Application Note
- Simple Roaming and ALPN – for details, refer to the ATWINC3400 Wi-Fi/BLE Network Controller - Software Design Guide
- ALPN connect example – demonstrates TLS ALPN extension to negotiate secure protocol (HTTP/2(preferred) or HTTP/1.1) between client and server

MicroChip ATWINC3400, WiFi and BLE (SPI up to 48 MHz)

https://www.microchip.com/wwwproducts/en/ATWINC3400

$11 @DigiKey. Very similar to ATWINC15x0. Good documentation (link above).

ATWINC3400 Wi-Fi® Network Controller Software Design Guide is dated 2019. It most likely was superseded by the other document dated 2020.

Asssessment: Both ATWINs provide WiFi. Example applications are all running on the 15x0, but not quite on 3400 (probably too new). So it is better to focus on the 15x0.

Assessment of the ATWINC Programming Examples

These applications are not providing a general purpose Web interface on the host (HTTP server, web browser, telnet, etc.).

The examples are customized for doing just one thing, Some of these applications are like an instrument, which gathers data and sends to the cloud or to Android phone. This is basically the essence of a dedicated instrument.

For example, Wi-Fi Serial reads input from the serial interface and sends it via a Wi-Fi connection and the terminal window will display the messages which you typed or received. It can be useful for chatting or controlling a remote device.

Microchip MRF24WG0MA, WiFi (SPI up to 54 MHz)

$30 from Digilent. This module is available from Digilent. It is a PMOD which you can plug into any of the three RiskZero PMOD sockets. The module can be ordered directly from Digilent, or from the distributors. The technical data and the inventory per distributor is on the following page. I am not discussing it any further because the data sheet, schematics, and libraries are available from the link below.

https://store.digilentinc.com/pmodwifi-wifi-interface-802-11g/

9

Espressif ESP8266EX Processor and Radio-on-chip

Espressif ESP8266 is a processor with a radio on chip It requires an SDK. Some vendors developed the FW and provide it in the on-chip in the flash. There is also a github page for the ESP8266 in general. Documentation is patchy because it is supported by the community.

Espressif website and documents

https://www.espressif.com/en/products/socs/esp8266

https://www.espressif.com/en/support/documents/technical-documents

These documents look good. The AT interface document dated 2020.08.25 is comprehensive. I hope that the Sparkfun module is actually using these commands rather than some old version.

They provide two SDK variants, the non-OS and the RTOS. In the RTOS they are using FreeRTOS and LWIP. The NON-OS API reference reads excellent, but it is huge.

https://github.com/espressif/esp8266-nonos-sample-code
https://github.com/espressif/esp8266-rtos-sample-code

Espressif ESP-WROOM-02D and ...02U with ESP8266EX

Modules with and w/o antenna are $3/ea. Network protocols IPv4, TCP/UDP/HTTP/FTP. User configuration AT Instruction Set, Cloud Server, Android/iOS app.

Note that there is the ESP8266 WROOM and the ESP-32 WROOM. These are different. The AT firmware exists for both. Full blown development systems are also available.

SparkFun WiFi for Arduino with ESP8266 (UART 11520 baud)

SparkFun WRL-13678, DigiKey 1568-1235-ND, $7/ea. The board looks like Nordic NRF24L01 with 8-pin connector, but the pin arrangement is different. It uses AT Commands over UART.

The wiki [5] makes it clear that AT commands were provided by Espressif as a demo, what makes it clear why some commands hardly work or provide strange results [4].

The Instructables [3] describe how to start connecting to the local WiFi network at home. The author warns to use the supply with more than 500 mA.

The MQTT example [7] makes it clear that this chip has a potential if used with C compiler. It is a processor after all. But it is not a complete application where data would be passed between the Host and the ESP chip. So it is no more than the 1st step along a very long path.

1. https://www.sparkfun.com/products/13678 - data sheet and links to documents
2. https://nurdspace.nl/ESP8266 - an overview written in very poor English.
3. https://www.instructables.com/id/Using-the-ESP8266-module/ - step-by-step in good English.
4. https://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference/ - AT commands.
5. https://github.com/esp8266/esp8266-wiki/wiki - lots of info, hobby quality.
6. https://github.com/esp8266/source-code-examples - a few undocumented code examples.
7. https://github.com/tuanpmt/esp_mqtt - MQTT example ported from Contiki.

ESP8266 assessment

The chip seems to be popular in the community. The quality of community documentation and examples [1-7] is hobby at best.

Spending effort on community examples is questionable, There is not much substance there except for Instructables [7] which is pretty helpful.

Other WiFi modules will probably be accompanied by equally large SDKs (Lantronix, Microchip, or Texas Instruments). If the module contains the CPU (it has to), and if the CPU can be programmed by the user, then the tools will be equally complex.

The complexity can be avoided if the module presents a simple interface. It can be AT commands, a script language (Silicon Labs), or the register-based interface (Wiznet wired Ethernet chips W5500 or W5300).

AT commands are implemented with a particular firmware. It means that the same module can be reprogrammed for using some other interface (register-based SPI, for example)..However, this is not possible when only the UART is pinned out to a connector.

AT commands seem to imply the UART, what means "low performance". Higher performance may be achieved when UART is used for control, and some other interface for data. I doubt whether such modules exist, but who knows?

Sparkfun WRL-13678 WiFi assessment

The Sparkfun module may be worth using with the AT commands to develop a simple control application. For example, setup the DAQ device, let it start and stop the acquisition, and retrieve histogrammed values once every few seconds. This is as far as this tool can go.

Espressif ESP32 Processor and ESP32 Modules

https://www.espressif.com/en/ecosystem/iot-college/books
https://www.espressif.com/en/products/modules
https://www.espressif.com/en/products/socs

The ESP32 and ESP32-S2 are two processors with WiFi and BLE. The ESP32-S2 also has the USB interface. They seem more powerful than ESP8266. The documents are of good quality. There are also some books which look very inviting. IMHO the Espressif website makes much better impression than the WRL-13678 would suggest. Go ahead and explore Espressif. Do not get discouraged by the Sparkfun and other hobby stuff.

The books are worth exploring. Espressif is apparently addressing the collega audience, which is one step above the hobby addressed by the Sparkfun.
Especially interesting:
Neil Cameron, Electronics Projects with the ESP8266 and ESP32. Building Web Pages, Applications, and WiFi Enabled Devices. To be released November 2020.

https://www.apress.com/gb/book/9781484263358

Nordic NRF24L01+ Radio Module (SPI)

The RiskZero radio socket pinout is compatible with the NRF24L01 wireless module used in 2013 FPGA Oberon. Remember: This is just a socket! Another SPI application is possible as well.

Search Amazon for "NRF24L01 transceiver module" and you will find plenty of these.

1) Aideepen 2PCS NRF24L01 Wireless Transceiver Module+2.4GHz Antenna for Arduino
2) DEVMO 2PCS Compatible with Arduino NRF24L01+ 2.4GHz Wireless Transceiver
3) MakerFocus 2pcs NRF24L01+PA+LNA Wireless Transceiver RF Transceiver Module

Details of Nordic NRF24xxxx (not recommended for new designs)

http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P tries to discourage from using the nRF24 Series. It says "not recommended for new designs".

NRF24L01P_Product_Specification_1_0.pdf available from sparkfun.com. The document is not available from Nordic. It tells you something about the company.

See also:
http://randomnerdtutorials.com/nrf24l01-2-4ghz-rf-transceiver-module-with-arduino/
https://gadgetperfect.wordpress.com/nrf24l001-rf-module2-4ghzinterfacing/

Should we use Nordic NRF24L01+ Radio Module?

We want to use the already existing design, which is fully developed. So the reservation is not a serious show stopper. The advantage of nRF24 is low power.

- 250 kbps, 1 Mbps and 2 Mbps on air data rates
- 11.3 mA TX at 0 dBm output power
- 13.5 mA RX at 2 Mbps air data rate
- 900 nA in power down
- 26 microA in standby.
- Host Interface

.. 4-pin hardware SPI
.. Max 10 Mbps
.. 3 separate 32 bytes TX and RX FIFOs

The chip operates in Tx, Rx, or standby modes. It is not duplex. Not possible to perform Tx and Rx at the same time. You must rather switch between the modes. It takes 130 microseconds.

C/C++ Software for Nordic NRF24L01+

Oven Edwards posted a C++ library to Mbed OS forum, with very useful API documentation.
https://os.mbed.com/users/Owen/code/nRF24L01P/

A lot of tutorials and app notes on nRF24xx

https://github.com/fffaraz/Introduction-to-Microprocessors/tree/master/material/nordic

Mouse/keyboard dongle with NRF24xx

The Nordic App Note nAN24-07 "Frequency Agility Protocol for nRF24XX" mentions the mouse/keyboard dongle with NRF24xxx. After some exploration it turns out an unsupported mess..

NRF24L01 Programming example from MIT

The pages are incomplete and seem abandoned, but whatever remains is pretty interesting.
Testing round trip over the radio network
https://pub.pages.cba.mit.edu/ring/
In particular, the test of the NRF24L01 and the C code to operate the radio.
https://pub.pages.cba.mit.edu/ring/rf/nrf24L01/

Old Legacy NRF24L01 Library for Arduino

This library will no longer be maintained or updated, but we will continue to publish it for the benefit of the the community, and you may continue to use it within the terms of the license. Nevertheless we recommend upgrading to RadioHead where possible.

http://airspayce.com/mikem/arduino/NRF24/

This legacy NRF24 library seems to be much simpler than the new RadioHead library. Also, it can use the low level auto-acknowledgement feature supported by this chip. RadioHead can not.

New RadioHead Library for Arduino

This C++ library is complicated. The list of supported radio chips is pretty impressive. It also supports message passing over serial UART wires.

This is the RadioHead Packet Radio library for embedded microprocessors. It provides a complete object-oriented library for sending and receiving packetized messages via a variety of common data radios and other transports on a range of embedded microprocessors.

http://www.airspayce.com/mikem/arduino/RadioHead

RadioHead and its RH_NRF24 driver provides all the features supported by NRF24, and much more besides, including Reliable Datagrams, Addressing, Routing and Meshes. All the platforms that NRF24 supported are also supported by RadioHead.

Oberon System Software for Nordic NRF24L01+

Discussed on a separate page.

Oberon System Software for Nordic NRF24L01+

Oberon NRF24L01+ software consists of two modules:

SCC.Mod is a low level SPI communication with the NRF24L01+. The name means Serial communications controller" (PO.System Section 9.5), which is misleading because "serial" usually means RS-232 rather than network.

Net.Mod uses SCC to implement the network, with special focus on file transfer. (Discussion: PO.Applications Chapter 10).

Remarks on Oberon System Software for Nordic NRF24L01+

1. I would expect that SCC.Mod would use a lower level SPI library, but it does not. It implements the entire SPI communication in the module.

2. Module Net.Mod defines numerical constants (see below) w/o explaining the particular values. If they are specific to NRF24L01+ then they should be defined in SCC. If they are legacy values from some other source, then it could be explained.

Examples of Net constants: ACK = 10H; NAK = 25H; NPR = 26H;

4. Module Net uses undocumented numerics in the code. E.g., in procedure SendMsg there are calls "reply(3)" or "reply(1)" w/o a word of explanation. It is a common malady of all Wirth software. It needs to get fixed.

5. Section 10.2 of PO.Applications (page 2) refers to Section 9.3, which however is devoted to an entirely different topic.

6. It could be beneficial to redesign the Net such that it works on top of NRF24L01+, WiFi, and Wiznet at the same time. Only then the name Net will be justified. Note that the scope of Net.Mod is much more restricted than the scope of Ethernet. So it could be possible to implement just the Net services upon the Ethernet.

7. Net.Mod does not provide services for machine to machine requests for actions. E.g., "turn an LED on" seems outside the scope of Net.Mod. The only focus of Net.Mod seems to be file transfer. It means that either Net.Mod needs to be extended, or another module needs to be written using SCC services.

8. PO.Applications Section 10.3 "Station addressing" (page 3) says "address is obtained from a dip switch". It does not say which DIP switch positions are used. How many bits and which bits?

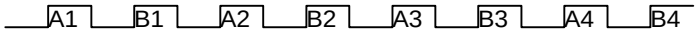
9. Station address should be rather obtained from the ID EEPROM which is present on the RiskXXX boards.

10. PO.Applications Section 11.5 discusses Core.Mod on page 12. No such module is provided.

12

ZBT Timing Diagram

The pipelined ZBT works with 2 clock cycle latency in both read and write cycles. In the write direction, the address is presented at A1. The data is expected by the ZBT chip at A2. The cycle B1 should be skipped. The same is true with reading from the chip.



The figure shows that the interleaved A and B cycles effectively define a dual-port operation, when the ZBT runs at twice the clock frequency of the bus host. There are three possible use cases.

1. The cycles A are used and B are discarded. The ZBT becomes a single port memory much like ASRAM. The ASRAM memory controller can be used with little modification.

2. Two bus hosts can access memory at A or B cycles without disrupting each other. The ZBT is operated as a dual port memory, with ports A and B operating at the alternating clocks.

3. The controller is using a single port at the full bandwidth w/o the regard to two-clock latency. The controller is internally pipelining the data to generate the needed two-clock "misalignment". Both the Xilinx and Altera cores are doing just that (see below).

OpenCores SSRAM Dual-Port ZBT Controller

https://opencores.org/projects/ssram

Open Cores dual-port design is available in VHDL. A quote from the project page:

The entity cs_ssram uses the standard interface to turn the ssram into a cycle shared memory. Because ZBTs feature zero bus latency there is no impact on throughput. Thus providing a low-cost alternative to dual-ported rams.

Official Xilinx and Altera ZBT Controllers

Xilinx App Note XAPP136 (v2.0) January 10, 2000 titled "Synthesizable 200 MHz ZBT SRAM Interface" describes ZBT and its controller. This App Note is marked "obsolete", but I see no reason to disregard the implementation.

A similar note AN-329-1.0 is available from Altera, titled "ZBT SRAM Controller Reference Design for Stratix & Stratix GX Devices".

Neither Xilinx nor Altera discuss the dual-port use of the ZBT chip. They are rather maximizing the throughput by internally "misaligning" the address and the data. These controllers would be needed if the soft CPU could run at 200 MHz.

Note that Spartan-6 on-chip hardened memory interface does not support ZBT. It is thus not clear why XAPP136 was marked "obsolete".

Wish List: Multi-Port ZBT Controller

The A-B-A-B-... scheme can possibly be generalized to A-B-C... or A-B-C-D....

A four-port memory would run at 50 MHz on each port. It can be used for two RISC5 cores, a video, and a hardwired DSP block such as a Histogram incrementer.

In my previous work I implemented Histograms in BRAM which is dual ported without interleaving the clocks. Making the ZBT effectively multiported will provide a few MB of such memory to a low end FPGA, rather than reaching for a larger, more expensive FPGA.

SSRAM: Candidate Multi-Port ZBT Controller

The Open Cores SSRAM project mentioned the triple-port and quad port on their Wish List. This work has not been finished. The controller is well structured but poorly documented (only the web page and the comments in the code). It is not even clear which kind of SSRAM it is serving, the pipelined or flow-through? The readme says "pipelined", but the comment in the code says "three cycles", which looks more like flow-through. The code needs to be analyzed to make it clear.

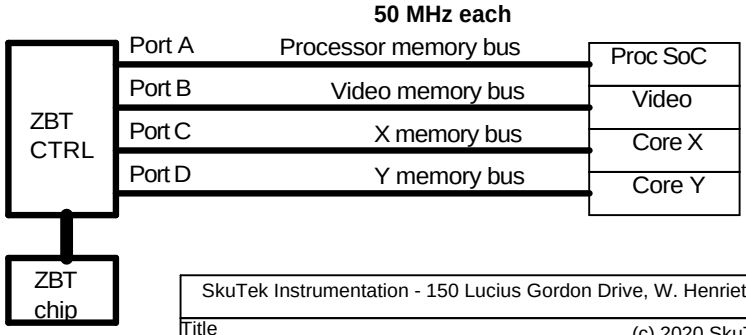
Firmware Structure With Multi-Port RAM Controllers

There are two possible design patterns of the Oberon SoC design.

1. The RAM controller is contained within the SOC component. Memory is not accessible from outside the SoC. However, the bus (address, data, and strobes) sticks out from the SoC in order to reach the peripherals. The Microblaze Controller System (MCS) discussed by Pong P. Chu shows an example of this pattern.

2. The RAM controller is outside. It is a component on the bus. The SoC is its client.

The latter approach is preferred with the Multi-Port Controller. One of the RAM ports will be connected to the processor SoC. Other ports will be connected to other RAM clients.



SkuTek Instrumentation - 150 Lucius Gordon Drive, W. Henrietta, NY 14586-9687

Title: ZBT Notes and Plans

Size: A | Document Number: RiskZero LX9 Single Board Workstation Computer | Rev: 0.1

Date: Monday, September 28, 2020 | Sheet 36 of 40

Sources of PMOD information

Digilent Pmod Interface Specification 1.2.0 is dated October 5, 2017. Note that some PMODs do not quite follow the specs. For example, the I2S2: Stereo Audio Input and Output is in violation of the specs. You cannot use it with RiskZero. (It is regrettable.)

1. https://reference.digilentinc.com/reference/pmod/specification
2. https://reference.digilentinc.com/reference/pmod/start

The list of modules available from Digilent is here. You can also buy them from DigiKey.

3. https://store.digilentinc.com/pmod-modules-connectors/

What is a PMOD?

PMOD is a small PCB board with either a six-pin or 12-pin, 0.1" right angle connector. It plugs into the mating socket at the edge of the main PCB. If you follow the specs of how wide is the PMOD board, you can stack them horizontally next to each other. You will be able to plug three PMODs into three RiskZero sockets, and they should not bump into one another.

The PMOD connector provides ground, power, and GPIO pins to talk to the PMOD add-on. The GPIO usually means SPI. Many PMOD boards are just break outs for SPI-based ADC's, DAC's, temperature sensors, etc., You can add their functionality to your main board without providing the footprints for those sensor chips on the main board.

Simple PMODs are priced at a few $$ each. There are also more complex PMODs, like a Blootooth module, a WiFi module, or a GPS unit. These are priced aroud $30 each.

You can build your own PMODs. You do not need to design a PCB and have it manufactured. Since the PMOD connector is 0.1" pin header, you can solder it onto a pre-perforated board and add through hole parts on the 0.1" grid. The opamps and other such chips still exist in 0.1" packaging. Just be aware that these are not high performance parts. Doing audio with 0.1" parts is perfectly viable. You can reach into a few MHz transfer over PMOD, while tens of MHz is iffy, and hundreds will not be possible.

If you design a PMOD board which is wider than the proscribed width, then you can connect it to the PMOD socket on the main board with an extension cable.

Things to keep in mind when using PMODs with the RiskZero board

RiskZero is providing three PMOD sockets, sharing the SPI bus and four GPIOs. Each socket has its own individual CS# line (chip select, active LOW). These are the only signals which are not shared. It means that the PMODs have to be accessed sequentially, one SPI transaction at a time. There is no restriction on the transaction order. Concurrent access is not possible.

Four GPIOs can implement extra signals, like muxing or interrupts. Most PMODs do not use any extra GPIOs. (Such PMODs can be implemented with only one row of six pins.) Some PMODs do use the extra pins. This depends on the PMOD.

Be aware that the four extra GPIOs are bussed across all three sockets. Consider the possible contention. It is advised to dedicate the GPIO to only one PMOD at a time. You can also make them input at the PMOD side. Avoid driving the GPIOs from more than one source.

What can you do with PMOD and the RiskZero board?

The most freedom is afforded if you develop your own PMOD boards. You can use either the PMOD BB (the breadboard with the PMOD connector) or a prototyping board of any size and connect them to the RiskZero with a PMOD Cable Kit from Digilent ($7).

Planning these experiments, keep in mind that the SPI signals are shared with all the SPI chips on the RiskZero board. This includes the W5500 Ethernet chip, Nordic radio, and the microSD card. You can decrease the load on this common SPI bus by using the full size SD card for the System Disk, because the full size SD has its own dedicated connections to the FPGA. It is not sharing the common SPI bus.

Since the main SPI bus is shared, the traffic is not predictable because it depends on all the SPI activities such as networking. If you need communication at regular time intervals, then you can use the GPIOs. The six-pin PMODs can be plugged into either the upper or the lower pin rows. Such PMODs will connect to either the common SPI bus or the GPIOs. The DA2 digital-to-analog PMOD is one example.

Here are some ideas, using some selected Digilent PMODs.

1. Using PMOD USB to UART Interface with FTDI FT232R implement the mouse and keyboard interface superseding the legacy PS/2. This PMOD will use all four GPIO pins. It will not use SPI.

2. Using PMOD WiFi with Microchip MRF24WG0MA, implement WiFi networking in Oberon.

3. Using either BT2 or BLE Bluetooth PMOD Interfaces, implement Blootooth under Oberon, communicatinng with RN42 over UART.

4. Using PMOD KYPD 16-button Keypad and PMOD OLEDrgb: 96 x 64 RGB OLED Display with 16-bit color resolution, implement a User Input keypad with a feedback on the OLED display. Both these PMODs are larger than the standard PMOD size, so you will need to use PMOD extender cables to connect them to RiskZero.

5. Using PMOD AD1 with two A/D channels, implement a dual channel digital scope. Use the main Oberon screen to display the waveforms, because the add on display from Project 4 may be too crude for this purpose. Use the GPIOs for the interface to avoid loading the main SPI bus. This 6-pin PMOD plugs into either the upper or the lower pin row.

6. Using either PMOD AD2 or AD5 with four A/D channels, implement a quad channel digital scope. Use the main Oberon screen to display the waveforms. Note that AD2 uses I2C interface which you will have to implement with two GPIO pins in the lower pin row.

7. Using the PMOD DA2 with two 12-bit D/A Outputs, implement a dual channel arbitrary signal generator running with up to 30 MHz SPI clock. Use the GPIO pins for the interface, because the main SPI bus may not be able to run at regular time intervals. Note that this PMOD can be plugged into either row of the PMOD connector, what makes the GPIO pins a viable option to drive the D/A chips.

Expansion Connector

The 0.1" pin expansion connector is located in the South-West of the board. It provides ground, 3.3V, and 5V. BEWARE! Five volts can damage the chips, when it is accidentally shorted to any chip on either the expansion board or the main board. If you feel uncomfortable with that, then you can clip the 5V pins off.

You can use individual jumper wires to pick off the signals. Use the Digilent 6-pin MTE Cable, or a set of individual wires with rectangular female sockets. You can also use oscilloscope probes to examine the signals with a scope.

All the expansion signals are also used elsewhere on the board. None of them is dedicated. You need to determine whether or not your planned use conflicts with other uses of the same signal. For example, if you want to use the four GPIOs, then make sure they are not needed for any PMOD module that might be attached. The following signals are provided on the expansion, and also used by the main board. All signals are 3.3V logic levels.

List of expansion signals

1. I2C bus (SCL, SDA). Also used for the MAC EEPROM 24AA02E48T-I/OT, temperature sensor TMP100NA, current sensors INA219AIDCN (three pieces), the real time clock M41T62LC6, and the HDMI monitor. The latter connection can be selectively enabled, so you do not need to worry with the unknown monitor doing weird things to the I2C bus.

   I2C is wired following the Digilent recommendations from their PMOD specification.

2. Four GPIO pins. Also routed to all three PMODs. They have 2k pullups to 3.3V. They can be used to implement your own UART or I2C on the PMODs or the expansion board.

   Be aware that the four extra GPIOs are bussed across all three PMOD sockets. Consider the possible contention. It is advised to dedicate the GPIO to only one PMOD at a time. You can also make them input at the PMOD side. Avoid driving the GPIOs from more than one source.

3. SPI select 3, 4, 5, and 7. They are used to select three PMODs and the Nordic wireless. They are yours to grab if you are not using them for their primary uses (for example., the primary modules are not plugged in).

4. Radio IRQ and CE. Their primary use is for the Nordic radio. They are also connected to the Sparkfun WiFi socket, where they are used for UART. (This will necessitate new firmware, sorry for that!). You can use them, if the primary modules are not plugged in.

5. The main SPI bus labeled SPI_West. There is only one primary SPI bus on the entire board. It is connected to the FPGA configuration flash, the W5500 Ethernet chip, the microSD card, all three PMODs, and the Nordic radio. Do not expect a very high frequency from this bus which is heavily loaded. A few MHz is probably feasible. This needs to be tested with real hardware.

6. The FTDI-to-FPGA UART. It should not be used for anything else, unless you are sure that the FTDI cable was unplugged and the FTDI signals are not active. If you need your own UART then use GPIO pins instead.

7. FPGA_DONE is a master reset for the entire board, active LOW. It is asserted LOW when the FPGA is not configured. It goes HIGH after the FPGA firmware was loaded. You cannot drive this signal from within the firmware.

Part Numbers

Connector pin headers for the main board, pins facing up.

CONN HEADER VERT 34POS 2.54MM
34 pos TSW-117-07-G-D Samtec
CONN HEADER VERT 32POS 2.54MM
32 pos TSW-116-07-G-D Samtec

Connector pin sockets for the expansion board.

CONN RCPT 32POS 0.1 TIN PCB
32 pos SSW-116-01-T-D Samtec
CONN RCPT 32POS 0.1 GOLD PCB
32 pos SSW-116-02-G-D Samtec

24 positions 6-146256-2
16 positions 77313-818-16LF