Fennec 6800

A Motorola 6800 Based Expandable Single Board Computer

Designed By: Chartreuse
2022-2023

Dedicated to the Retro Dreams Discord Server for all the help and discussions over the years.

Introduction

The Fennec 6800 is a Motorola 6800 based single board computer (SBC) created with the inspiration of targeting 1977 components and features. It features an original Motorola 6800 processor running at whopping 921.6kHz, 8kB of the latest 2114 SRAM, 2kB of ROM using the brand new 2716 EPROM, and dual serial ports. An optional 40 pin expansion bus allows for connectivity to other modules, such as the Fennec VTI video terminal interface board, or to create your own attachments.

Specifications

- Motorola 6800 @ 921.6kHz
- 1-8kB of onboard SRAM using pairs of 2114 (1k x 4 bit) SRAM chips. *Using the optional RAM adapter board up to 32kB of RAM can be fitted with minimal modifications*
- 2kB, 2716 EPROM containing the Fennec Monitor. Optional support for a 4kB 2732 EPROM
- Dual 6850 ACIA, RS-232 UART connected to dual 25 pin EIA-232 ports in DTE wiring.
- Selectable serial baud rates via DIP switches from 600-9600 baud + 115,200 in default /16 mode, 150-2400+28,800 in /64 mode when configured. Non-standard baud rates are possible using a different master oscillator, but CPU speed will be affected.
- Single rail design. Requires only a 5v regulated source of power.
- 40 pin expansion bus with pre-decoded expansion board select signals, allowing for quite prototyping of custom expansions.
- Beautiful blue PCB with exposed traces for a refined look.
- Compact Board design, only 8.6×5.2 inches!

The Fennec 6800 includes a capable and expandable ROM monitor built in, requiring no tedious toggling of front panel switches to enter and begin executing code. Only a terminal is required to begin making use of the system, simply hook the terminal up to the bottom serial port (furthest from the power jack). This serial port is the local console port in the default system, all traffic is normally routed through this port, though inside the monitor certain commands may be redirected to the auxiliary port located above it.

The monitor provides facilities for examining memory, depositing values into memory, jumping to code in memory, receiving a binary over the XMODEM protocol into RAM, and dumping a block of RAM over the XMODEM protocol. With these facilities and an appropriate host system, program development can be done remotely and loaded into the Fennec to speed up initial system bring-up and testing, or development can be done in the traditional way on paper then loaded in in hexadecimal into the program memory and tested out using the computer.

The monitor also provides support for option ROMs. These option ROM images can be mapped into any of the expansion regions in upper memory starting at a multiple of 0x1000. If detected they can hook into the monitor to provide future expandability and extra features, such as the Fennec VTI board which uses an option ROM to redirect the default terminal from the first serial port to a video display and asci keyboard interface, allowing for standalone use. Future cards may also use this facility to add additional commands such as providing for a rudimentary disk operation system for example.

Design Goals

My goal with the Fennec 6800 was the idea of making a computer mostly within the constraints of the very early days of the personal computer movement. I originally planned to target around 1975 to 1976, however a major difficulty of that era is the available types of EPROMS. The venerable 1702A EPROM introduced in 1971 only provides 256 bytes of storage, but more importantly requires a +5v and -9v rail to operate, and +10v,+47v, and +59v to program. Without a suitable programmer this chip was right out. Next option was the 2704 and 2708 EPROMs, introduced in 1975 at 512 bytes and 1kB respectively these would be fairly suitable to implement a monitor. However like the 1502 these require split rail power supplies to even operate, +5, -5v, and 12v for both, and 26v to program. Not wanting to require a split rail power system I needed to look forward again. Enter the 2716 EPROM, introduced in 1977 and being one of, if not the first, EPROM to work with a single +5v supply, and giving us a generous 2kB of space.

This set the target date for my system, 1977, two years after the venerable Altair 8800, one year after the Sol-20, and the same year that the personal computer trio launched; the TRS-80, Commodore PET, and the first Apple][launching the same year. Now this is a loose goal as I will soon demonstrate, but gives me a focus onto what I want the capabilities to be and roughly how those should be aimed at. One major cheat to this on the board was the use of MAX232 serial level shifters, a chip from 1988. This was more for convenience and not needing to introduce split rail power for the more period MC1488/MC1489 shifters, or discrete transistor circuits also common.

In keeping with the target date, RAM needed to be limited. Now it’s tempting with a modern homebrew to just slap 64kB or more on these 8-bit systems and be done with it, however in fitting with the era I wanted something a bit more humble. I figured that 8kB was a suitable amount of RAM, with the system being capable of working with as little as 1kB fitted. For RAM I went with the 2114 SRAM chip, also from 1977. While it would not have been the most economical RAM chip of that era, it was the most simple to interface with. It requires only a single rail 5v supply, and no refresh or multiplexing circuitry. Another reason for the choice was that I was able to source twenty of these chips relatively cheaply from an eBay seller and it was part of the inspiration for the project.

Now you may think that only 8kB of RAM seems rather limited, and really it’s not that bad for the era. Sure bigger systems were around, but many of the home computers and kits available at that time would be lucky to have 8kB. The Altair 8800 in bare form shipped with only 256 bytes, though if you were running BASIC you’d likely have at least one 8kB ram expansion for it. Even for 1977, the lowest spec Apple][came with a whopping 4kB of RAM, and as RAM was quite expensive, not many people would have opted to max the Apple][out with 48kB. The TRS-80 Model 1 also only came with 4kB as standard, same with the PET][. So 8kB on the Fennec is plenty in line with the competition and other computers of the era, even if RAM prices started falling quickly afterwards.

Part of design was this would be my first homebrew computer created on a PCB in KiCAD without prototyping it on a protoboard or breadboard. Care was put into the schematics and overall design, built on experience with my prior homebrew computer projects and lessons learned from them. This resulted in a computer that required only a small handful of bodge wires to get it working. A resistor was pulled the wrong voltage rail on the ROM. One of the serial baud rates was connected to the wrong point, resulting in the wrong indicated baud, and one bus buffer was connected in a way that made the expansion port unable to drive the bus. The third error being corrected by using a diode and a spare pin on the BUS to create a bus enable input.

In fitting with the design period goals, my aesthetic intentions were to make the board look like some period boards of the time, specifically the blue FR4 with exposed traces look of some Solid State Music and Vector protoboards of the era. Blue dyed FR4 is not really an option at any of the major board houses, especially the cheap ones, however I had learned of a trick from Glitch, who used the concept on his recreation of a Solid State Music board. The trick being to copy the copper layers into the Gerber files of the solder mask layers. This would result in the copper being entirely exposed, with solder mask everywhere but the copper. This is due to the solder mask Gerber files being an inverse mask. Using this with blue solder mask results in the board with some of the appearance of blue dyed FR4, though with a slightly more shiny finish and more translucency of the board. However the end result is simply stunning in appearance.

One downside of course is that all traces are exposed, resulting in more chances of metal contact resulting in a short, or solder splashes shorting some traces. However this can be avoided by being extra careful during soldering, and double checking for shorts after assembling. Shorts can then simply be wicked away with solder braid. As for the former, I'd recommend placing the board in a protective case, perhaps with a clear top to allow the board design to be clearly seen.

Theory of Operation

Clock Generator

The MC6800 requires two two clock signals, ϕ1 and ϕ2, to be rail-to-rail, non-overlapping inverse clock signals, ie. 180 degrees out of phase. The tricky part here being the non-overlapping part, preventing the use of just a simple inverter to put the clocks out of phase. A simple inverter would satisfy the 180 degrees out of phase part, mostly, but the second clock would be overlapping with the first by the propagation of a 74LS04 gate delay, approximately 7-15ns.

Motorola recommends the use of the MC6875 clock generator to provide the required ϕ1 and ϕ2 clock inputs. However this chip is difficult to track down nowadays and would add another obstacle to building the Fennec by requiring another vintage chip. Furthermore, even in it's time the chip was rather expensive, so designers would be apt to avoid it if they didn't need the additional functionality it offered besides driving the clock signal. For these reasons I chose to forgo this chip, and instead settled on driving the clock myself.

To generate these non-overlapping clocks without the MC6875 we need just a simple circuit to generate a non-overlapping inverted signal. A pair of NOR gates are wired to form a simple RS flip flop, with a simple square wave clock signal going into the set input. The reset input is connected via an inverter to the clock signal as well. This inverter between the set and reset inputs helps to insure the non-overlapping condition through it's own gate delay. With this the output of the NOR gate connected to the set input will always change it's state before that of the one on the reset input. Thus ensuring that both clock inputs are never high at the same time, the ~10ns gate delay of the 74LS04 providing a period where both clocks will be low before any rising edge.

UART decoding

In the name of future expandability, as the /EXP0 is brought out to the bus, the two UARTs are fairly tightly decoded on the system board, taking up only 16-bytes of the entire address space. While the upper address space is decoded into 8 equal 4kB regions called EXP6 to EXP0 as well as the ROM region, it feels appropriate to not waste address space on partial decoding of the UARTs.

The UARTs are thus decoded with a combination of signals. First A11 to A4 are NAND-ed together to create a signal limiting it to the top 16 bytes of any given 4kB segment, this signal is then inverted before going to the active-high CS1 chip selects of the 6850 UARTs. The active-low chip select CS2 is fed by the /EXP0 selection signal, thus now limiting both UARTs to the $EFFx address space. Finally to differentiate the two UARTs, one is fed A1 on its final CS0 active high chip select, and the other is fed the inverse of A1 on its CS0 chip select.

This results in a slight partial decoding as the UARTs only require 2 bytes of address space each, but will be mirrored 4 times within the 16-byte space that they are decoded.

UART Clock Generation

The UART clock rates were an important factor in deciding the overall clock speed of the system. The CPU speed was desired to be as close to the maximum 1MHz permitted by a plain MC6800 as possible, however I only wished to use a single oscillator for the entire system. The standard 1.8432MHz baud rate oscillator was thus chosen to be master oscillator for the system. When divided in half we get a 921.6kHz clock, very close to the maximum 1MHz for the CPU.

1.8432MHz also is extremely useful as a baud rate clock, with no external division and just relying on the 6850 ACIAs internal /16 mode for asynchronous serial we get 115,200 baud. Too fast for our time period and liking, but a useful modern serial baud rate. The magic however happens when we further divide the clock down. First we divide the 921.6kHz CPU clock down by 6. To accomplish this we use a 74LS73 dual-JK flip flop in a divide by three configuration. The output of this configuration is not a 50% duty cycle however so we divide that again by 2 using a 74LS74 D flip flop to return the duty cycle to 50%. This brings us to a total of 1.8432MHz / 12 or 153.6kHz. When this baud rate is divided by 16 in the UART that results in our first usable standard baud rate of 9600.

From 9600 it’s then just a matter of using a 4-bit binary counter to further divide this down into usable baud rates. 4800, 2400, 1200, and 600 are derived this way. We can also take advantage of the /64 mode of the 6850 UART to get an additional set of 6 baud rates to select form. 115,200 becomes 28,800. 9600 becomes 2400, 4800 becomes 1200, 2400 becomes 600, 1200 becomes 300, and 600 becomes 150. This gives us the standard set of serial baud rates from 150 to 9600 baud to chose from, in addition to the faster standards of 38,400 and 115,200.

19,200 baud is unfortunately not possible with this clock choice and division circuit as it would require 1.8432MHz / 3 / 16. However we cannot produce the divide by 3 with a 50% duty cycle as required. It could be generated with a 3.6864MHz oscillator, however this would require another D-flip flop to divide down the clock to create our 921.6kHz CPU clock.

ROM decoding

Upon coming out of reset, the Motorola 6800 first fetches its 16-bit reset vector from address $FFFE/$FFFF it then jumps to the address contained in that vector to begin execution. Because of this we require ROM to be in the upper most address of our address space. As such the Fennec 6800’s 2kB ROM is put into the upper most address space. The upper 32kB of address space on the Fennec is decoded by a 74LS138 3-8 decoder, resulting in 8 equally sized 4kB regions being decoded. The 2716 ROM is assigned the upper most of these, labelled by the select line /ROM.

As the 2716 ROM is only 2kB, it is allowed to mirror itself an additional time in this 4kB space. First it is visible from $F000 to $F7FF. Secondly an additional identical copy is visible from $F800 to $FFFF. For the code in the ROM its canonical address is assumed to be the upper $F800-$FFFF mirror.

If a 2732 ROM is fitted, and jumper JP1 fitted, then it will not be mirrored at all, instead spanning the entire 4kB address space assigned to the ROM. In this case a special version of the monitor ROM will need to be compiled, either to put the standard version only in the upper 2kB of the ROM, or else modified to know that it is now 4kB long so that the reset and interrupt vectors arrive in the correct places. If the later option is chosen note that code that relies on the monitor jump table, or any other fixed entry point into the ROM will need to be patched to work.

Expansion Bus

The Fennec 6800 is equipped with a custom 40-pin expansion bus. This size was chosen to correspond with easily available 40-pin IDC connectors and ribbon cables. These ribbon cables are commonly found in old IDE interfaces for IBM-compatible personal computers. Later 80-pin varieties are NOT compatible as they assume certain pins to be ground. **Do not attempt to use an 80-pin IDE cable as a bus cable.**

The bus provides access to a buffered set of 8 data lines. 12 buffered address lines A0 through A11, corresponding to the size of each expansion area in the memory map. 7 pre-decoded expansion signals /EXP0 through /EXP6 to allow for rapid development and prototyping of expansion cards. An assortment of 6800 control signals: BA indicates that the bus is available if the processor is in /HALT or WAIT. /VPA is a decoded signal where VMA is asserted and PHI2 is high, indicating addresses are stable and peripherals can drive or read the bus. /HALT can stop the processor and free the bus. /RESET is bidirectional (open collector) to either reset peripherals from the master reset circuitry on the CPU board, or to drive reset to reset the processor. /NMI and /INT are open collector inputs to signal either a non-maskable or maskable interrupt to the processor. E_PHI2 is a buffered TTL level CPU clock at 921.6kHz, allowing peripherals to synchronize more flexibly than just /VPA or derive synchronous clocks to the CPU. R_W2 is a buffered R/W signal where a high level indicates read and a low level a write operation. /BEN is an open collector input to the CPU board indicating that an external board wishes to have the data buffer enabled. This signal should be driven by a peripheral board when it senses it has been decoded, it is required to be enabled for both reads and writes to work to that peripheral. Power and ground are self explanatory, a regulated 5v source and ground reference.

Finally two lines on the buffer are not-connected and reserved for future/user use and expansion to send signals between multiple peripheral cards, or for adding additional features to the system board.

[INSERT BUS PINOUT DIAGRAM]

Assembly

Assembly is rather straight forward but should be accompanied by the schematic or bill of materials to know where to place the various resistors. I've corrected this for my future boards, but this board only has the reference designators and not the component values marked.

Like other throughole kits I would recommend starting from the shortest components to the tallest components, and I would personally recommend sockets for the ICs, especially the vintage ones, though they are not required. My personal assembly order would be to do the IC sockets first, even if they're taller than the resistors, but either approach has it's merits. For the tantalum capacitors to the side of the RAM, and the two electrolytic capacitors, make sure to get their polarity correct. Both have their positive side marked on the board, for tantalum capacitors their **positive** lead is normally marked with a + sign or stripe on the capacitor, and for electrolytics their **negative** side is normally marked with a stripe. If cost is a factor, feel free to omit some of the tantalum capacitors as the number is likely overkill as I miscalculated the current dram of the SRAM chips (mainly due to only a single pair being active at any one time).

Before installing R8 near the 2716 RAM please check the bodges section below if you have a REV. 0 board, if you have a REV. 1 board disregard this information.

If you are opting to not install a power switch, connect a decently thick jumper wire from the centre pin of the switch footprint to the top pin of the 3. Do not connect the wire up to the two mounting holes near the top edge of the board as this would result in a short circuit.

The CPU must be a Motorola 6800 or pin compatible clone. The later Motorola 6802 is not compatible as it requires a different clock generation circuit and includes some built in RAM, though with some bodges it may be possible to adapt, but this is left as an exercise to the reader. The 6801/6803/6805 micro-controller variants are likewise incompatible as they use a different pin-out and multiplexed bus structure. The Motorola 6809 is also incompatible being a completely different CPU. The rather rare MOS 6501 CPU would work in the board, however it would require an entirely different ROM to be made for it. While it's pin compatible with the Motorola 6800 it is NOT software compatible.

Care should be taken when installing the DIP18 sockets for the 2114 SRAM chips as there are a few instances where multiple traces are fairly close the pins and I found that this was where I made the only solder bridge when assembling my original, but this I caught immediately.

Resistors R1 and R2, the pair of 680 ohm resistors to the left of the CPU, are used to help drive the outputs of the clock generator U5 rail-to-rail to meet the 6800's specifications. If using a 74HCT32 instead of a 74LS32 these resistors may be omitted entirely. Other logic family substitutions may require slight changes from the values depending on the current sinking strength of the IC in question. The value of the resistors should be kept as low as feasible to improve the rise times, up to what is required by the data-sheet, but need to be high enough to present a voltage below the 6800's V_IL for the clock input pins. For the clock lines, the Motorola 6800 data-sheet calls for a rise and fall time of not greater than 100ns, an input low voltage of not greater than 0.4v (above Vss/GND), and an input high voltage no less than 0.6v below Vcc, a voltage of 4.4v in our case.

If using polarized capacitors, such as electrolytic or tantalum, for the MAX232 ICs charge pump capacitors, their polarity is not marked on the board and must be installed as following: For C44, C46, C48, C49, C50, and C52 the positive lead should be installed towards the "top" of the board, if looking at the board with the Fennec 6800 text upright, ie. with the positive leads going on the side towards pin 1 of their respective IC. C47 and C51 should have the positive lead going to the right side of the board if viewed the same way, ie with the positive lead going towards pin 16 of their respective MAX232 IC.

The two DB-25 ports are wired for male DB-25 ports such that the device is configured as a DTE device. If female ports are fitted instead then the device will stick act as a DTE, just with the wrong polarity connector, for the Fennec to act as a DCE a null-modem adapter is required to swap the Rx and Tx lines, as well as the CTS and RTS lines. DTS, DCD, and DSR and not connected and thus limited modem compatibility is available. The 6850 UARTs fitted do have provisions for a DCD line to be fitted, however no free buffers are available with the MAX232 chips. If desired an additional MAX232 or similar could be dead bugged into the circuit, and the DCD lines hooked up, however no space is provided for the chip or it's associated capacitors.

Required Bodges (REV. 0)

If assembling a REV. 0 version of the board a number of bodges need to be applied for correct operation. The baud rate bodge is optional as it only affects the 115200 baud rate which is honestly too fast for the computer in most circumstances. For wiring I recommend 26 or 30 AWG wire wrap wire (0.4 to 0.25mm wire) for doing the neatest bodges. Wire-wrap wire has thin insulation of Kynar (PVDF) or sometimes Teflon (PTFE) and is silver plated copper wire, and is ideal for bodges like these.

VPP Resistor on the 2716 EPROM

The original board layout had the 10k resistor from the VPP pin on the 2716 going to ground, however for correct operation of the ROM it should be going to the 5v rail. Instead of putting the resistor between the two holes designated for it. Solder one lead to the hole nearest the jumper, and for the other, bend it 90 degrees and solder it to the top side of the nearby 5v rail. The rail is just to the right of the ground rail hole that it was originally indicated to be soldered into.

Bus Transceiver for Expansion

This one is a little more involved and requires two additional components. A 1n4148 or similar fast signal diode, and an additional 10k ohm resistor. Solder the resistor from pin 40 to pin 32 of the expansion connector. When looking on the back side of the board with the edge connector on the edge closest to you, pin 40 is the top left most pin, and pin 32 is the pin four positions to the right of it.

Next you need to cut two traces on the board. First is the trace going in to pin 19 of IC U10 (74LS245) to disconnect the output enable line.

Next solder a wire from pin 19 of IC U10 to pin 32 of the expansion connector, where we attached the resistor to previously.

Next we need to solder a diode to pin 19 of IC U10, keep the leads very short as the other end of this diode will be otherwise loose and we don't want it to short to anything else. Solder the cathode (black stripe) side of the diode to pin 19, and leave the other end in the air. Then solder a wire from the anode end of the diode that we left floating, to pin 9 of IC U7 (the lower 6850 UART). This connects the /EXP0 signal to the diode, allowing it to enable the bus buffer again.

High Speed Baud Fix

The 115,200 baud rate selection switch was accidentally connected to CLK rather than FULLCLK, resulting in the baud rate actually being 57,600. To fix this and match the speed to the label cut the trace leading to pin 7 of the lower dip switch, leaving the connection to the upper dip switch alone.

Then run a wire from pin 3 of IC U6 (74LS74) to the now isolated pin 7 of the lower dip switch. This will connect the full 1.8432 MHz clock to the dip switch. This clock rate when divided by 16 in the UART results in 115,200 baud.

MAX232 Bypass Caps (Optional)

The general Vcc to GND bypass caps for the MAX232 were left out of the rev. 0 board by mistake, however due to nearly capacitors this has not shown to be an issue at all. However to properly bypass them a 1uF ceramic capacitor should be installed between Vcc and GND for each the ICs. A 1uF capacitor should be soldered between pins 15 and 16 on each IC on the bottom of the board.

Software

Memory Map

The memory map of the Fennec 6800 is rather simple. The lower 32kB of address space from $0000 to $7FFF is reserved for system RAM, with $0000-$1FFF actually being populated with RAM on a full 8kB system with 16x 2114 SRAMs. When using the optional Fennec 62256 RAM replacement board, the full 32kB can be mapped with the addition of two bodge wires from the replacement board down to the CPU socket.

System ROM is mapped from $F000-$FFFF, if the normal 2kB 2716 EPROM is being used then this comprises of two mirrors at $F000-$F7FF and $F800-$FFFF with the later being used.

The remaining address space is divided into 7 expansion zones labelled EXP0 to EXP6. EXP6 is mapped at the lowest memory address of $8000, and each expansion section is given 4kB of address space to make use of. EXP6 goes from $8000-$8FFF, EXP5 from $9000-$9FFF, EXP4 from $A000-

$AFFF, EXP3 from $B000-$BFFF, EXP2 from $C000-CFFF, EXP1 from $D000-$DFFF, and finally EXP0 from $E000-$EFFF.

EXP0 is partially used by the onboard two 6850 ACIA UARTs. These are mapped at $EFF0-$EFFF, with with 4 mirrors of each being present, leaving the rest of the address space still free. The canonical base addresses of each are $EFF0 and $EFF2, each being composed of two addresses for data and control. See the 6850 ACIA data-sheet for further information on programming the UARTs.

Option ROMs are tested for every 1kB starting from $8000 to $DC00. A valid expansion ROM begins with the two byte sequence of $F1 $0F the "FLOoF" signature, following this is a 10 byte human readable name of the option ROM that will be printed to the console when found, followed by 4 reserved bytes. The code of the option ROM when found is executed started immediately after this signature at offset $10 (16-bytes) into the ROM.

ROM Monitor

The stock ROM monitor is currently at revision 3.0 and includes support for a number of commands to make programming possible and allow for ease of transfer of programs to other systems. There are facilities to both load and retrieve hexadecimal data from RAM, to jump to a specified address, and to transfer memory contents over XMODEM from either the host to the Fennec, or from the Fennec to the host. The later two commands are intended to support rapid development on the Fennec when using a more powerful host system acting as a serial terminal emulator. They could also be used with a modem line to transfer data from one host to another. The 3.0 revision adds various memory hooks to the BIOS to allow option ROM code on expansion parts to hook and expand the BIOS allowing for greatly increased and flexible capabilities. Option ROMs can redirect I/O and also introduce additional commands into the monitor, and even replace existing commands.

Commands

- **E** - (E)xamine RAM – Performs a hex dump from between two specified addresses. The dash between addresses is NOT entered by the user but automatically inserted by the monitor
  - eg: * **E0000-0010**
    0000: 00 20 08 20 00 00 DF 46 2C 0E 6F 36 7E FA 08 7E
    >
- **D –** (D)eposit RAM – Allows the user to enter hex data in memory starting at the specified address. After entering the address the starting address will be displayed and the user can then start entering bytes. After every byte (2 digits) entered a space will be returned showing the byte has been entered into RAM. Every 16 bytes a newline and the next address will be displayed similar to the examine command. Press return or ESC after you are done with entering bytes.
  - Eg: * **D0100**
    0100:**12 34*[esc]***
    *

- **G –** (G)oto address – Jumps to the specified address and resumes execution at that point. This is performed as a subroutine jump, so if a program preserves the monitors stack, it can RTS back into the monitor. Otherwise it should jump back to the cold start address at $F800 so that the monitor can resume.
    - Eg: * **G0100**
      Program executed at address $0100 starts
- **R –** (R)ecieve XMODEM – Receives a program using the XMODEM (checksum) protocol starting at the specified address. After specifying address the user will be prompted to start the XMODEM transfer on the remote computer. At this point start the transfer and when completed you will be returned to the monitor prompt. If the prompt is not visible, try hitting ESC to get it to reprint. The Fennec will send a NAK approximately every 3 seconds waiting for the other end to begin it's transfer. It also includes retries, attempting to contact the sender 10 times before giving up after an error or lack of acknowledgement. If 10 retries pass it will output *TIMEOUT* to the console and return to the monitor to indicate the error.
    - Eg: * **R0100**
      SEND FILE NOW
      User starts XMODEM transfer in their serial terminal emulator program
      *
- **S –** (S)end XMODEM – Sends a block of memory using XMODEM (checksum) from between the two specified addresses. After specifying the second address, the user will be prompted to start an XMODEM receive on their terminal. At this point, start the transfer and when completed you will be returned to the monitor prompt. If the prompt is not visible, try hitting ESC to get it to reprint. The send command currently does not feature any timeout protection and instead relies on the user and the status of the other end to detect that something has gone wrong in the transfer, if the transfer is locked up the user must press the reset button to return to the monitor.
    - Eg: * **S0100-0200**
      RECV FILE NAME
      User starts XMODEM receive in their serial terminal emulator program
      *

Callable Monitor Routines

The monitor program for the Fennec is ever changing, and proper code should not rely on addresses within it of its routines to stay at fixed addresses. However a jump table of useful functions is kept near the beginning of the ROM at a fixed address for user programs to call. This table is used to initialize the RAM controllable routines by the monitor and thus contains the serial port 0 and 1 input and output functions, as well as a non-blocking version of the serial in routines for both. If the user program wishes to use the controllable devices of CON and AUX, they should instead use the RAM resident jump table of the controllable routines, see the section **Controllable Routines** for more information.

- S0IN - $F803 – Serial Port 0 (lower port) blocking read character. Returns the value received in the A register. Corrupts the B register.
- S0OUT - $FB06 – Serial Port 0 (lower port) write character routine, blocks until character is sent. Value to send is passed in the A register. Corrupts the B register
- S0IN_NB - $FB09 – Serial Port 0 non-blocking read character. If a character is available in the UART, returns the character in A with the zero flag cleared, otherwise returns 0 in A with the zero flag set. Corrupts the B register.
- S1IN - $F80C – Serial Port 1 (upper port) blocking read character. Returns the value received in the A register. Corrupts the B register.
- S1OUT - $FB0F – Serial Port 1 write character routine, blocks until character is sent. Value to send is passed in the A register. Corrupts the B register
- S1IN_NB - $FB12 – Serial Port 1 non-blocking read character. If a character is available in the UART, returns the character in A with the zero flag cleared, otherwise returns 0 in A with the zero flag set. Corrupts the B register.
- DDEVIN - $FB15 – Not recommended, use controllable routine DEVIN instead
- DDEVOUT - $FB18 – Not recommended, use controllable routine DEVOUT instead
- DDEVINNB - $FB1B – Not recommended, use controllable routine DEVINNB instead

Controllable Routines

In order to allow for expandability of the monitor ROM by option ROMs the Fennec makes use of so-called controllable routines. This is essentially a jump table of I/O routines used by the monitor that are stored in volatile RAM. Using these an option ROM or user program can change where the routines point to such that the monitor will use a different I/O device for user interactions.

The controllable routines define two virtual devices, the CON and the AUX device. By default the CON device is set to serial port 0, and the AUX device is set to serial port 1.

The CON device is designed to be for the user facing communication device, it is where user input is expected and data to the user is output. An example of this is the monitor prompt itself, the * prompt is written to this device, user input is received from it and echoed back to it, and the command results are output to it. If connected, the Fennec VTI module will take over this CON function, changing the device so that output is written to the video display (emulating a serial terminal) and input is from a connected parallel ASCII keyboard.

The AUX device is designed for supplemental functions not strictly dealing with user input. By default this device is used for the XMODEM commands. However user software may use this device as seen fit, for example it may be used as a modem line for connecting to other systems without having to resort to direct serial port access. It may also be used for a serial printer; a parallel printer could also be used with a suitable expansion card.

Besides the AUX and CON devices and their functions a routine is also provided to address arbitrary devices by number. These routines take a device number as an argument and route the output to the appropriate device. The first 4 devices are specified by the monitor and should be respected by all option ROMS implementing this function. These devices are the CON device, the AUX device, serial port 0, and serial port 1 in order. Further devices greater than this may be implemented by either option ROMs or user programs.

For all controllable routines it is expected that registers A and B may be changed by the routine (caller saved), while IX and SP are to be preserved by the routine (callee saved). Arguments are passed in A, and the return value is also in A. Flags may be set by the routines on return, such is the case with the non-blocking variants of the various IN functions.

- CONIN - $000C – Blocking get character routine for the CON device. Result in A
- CONOUT - $000F – Blocking put character routine for the CON device. Character to send in A
- CONINNB - $0012 – Non-blocking get character routine for the CON device. If a character is available from the device, returns the character in A with the zero flag cleared, otherwise returns 0 in A with the zero flag set.
- AUXIN - $00015 – Blocking get character routine for the AUX device. Result in A
- AUXOUT - $0018 – Blocking put character routine for the AUX device. Character to send in A
- AUXINNB - $001B – Non-blocking get character routine for the AUX device. If a character is available from the device, returns the character in A with the zero flag cleared, otherwise returns 0 in A with the zero flag set.
- DEVIN - $0001E – Blocking get character routine for the device selected by B. Result in A
- DEVOUT - $0021 – Blocking put character routine for the device selected by B. Character to send in A
- DEVINNB - $0024 – Non-blocking get character routine for the device selected by B. If a character is available from the device, returns the character in A with the zero flag cleared, otherwise returns 0 in A with the zero flag set.

For the three DEV calls the following devices are defined by the monitor ROM and must be respected by replacement functions:

- 0 – The CON device
- 1 – The AUX device
- 2 – Serial Port 0

- 3 – Serial Port 1

Devices 4 and up may be defined by option ROMs or user programs at their will. See the manual for a given board to know which devices it defines.

Bill of Materials

Ceramic Capacitors

- 35x 0.1uF 50v X7R with 0.1” (2.54mm) lead spacing
- 11x 1uF 50v X7R with 0.1” (2.54mm) lead spacing. 10 for MAX232 charge-pumps, 1 for 555 reset circuit.

Tantalum Capacitors/Low ESR Electrolytics

- 8x 10uF 16v with 0.1” (2.54mm) lead spacing. Max diameter/width of 5mm.

Aluminum Electrolytics

- 1x 470uF 16v with 5mm lead spacing, max diameter of 13mm. *Value not critical, can be larger, my original prototype used a rather tall 2200uF capacitor in this place.*
- 1x 100uF 16v with 2.5mm lead spacing, max diameter of 6.3mm. *Don’t use too high of diameter of capacitor or it may interfere with 62256 ram adapter board or card rails when mounting.*

Sockets

- 16x DIP18 sockets (for 2114 SRAM). If using the 62256 RAM adapter only 3 are required.
- 6x DIP14 sockets
- 5x DIP16 sockets
- 5x DIP20 sockets (0.3” wide)
- 3x DIP24 wide socket (0.6” wide)
- 1x DIP40 wide socket (0.6” wide) for CPU
- 1x DIP8 socket for 555
- 1x DIP14 crystal socket (only 4 corner pins fitted) for main oscillator

Connectors

- 2x D-sub DB-25 male board mount sockets with 7.7mm pin edge offset and 9.12mm mounting hole offset, or compatible.
- 1x 2.1mm x 5.5mm barrel jack. May be substituted to match your desired regulated 5v power supply
- 1x 2x20 0.1” male pin header. *I recommend getting a full 2x40 and cutting it in half.*
- 1x 1x2 0.1” male pin header. *I recommend getting a full 1x40 pin header and cutting to size, it’ll end up cheaper.*

Resistors

- 8x 10k 1/4W carbon film resistors.
- 2x 620 ohm 1/4W carbon film resistors
- 2x 470k ohm 1/4W carbon film resistors
- 1x 470 ohm 1/4W carbon film resistor (for power LED, use higher value if using a high brightness led)

Diodes

- 1x 1n4148 or similar small signal high speed diode
- 1x 5mm LED. Colour of your own preference or choosing but a deep red low brightness LED recommended.

Transistors

- 1x 2n3904 or p2n222 in TO-92 package with a staggered centre pin.

Switches

- 1x 6mm through hole, 4 pin push button switch for reset
- 1x ESWITCH 100SP1T2B4M7QE right angle toggle switch or compatible for power
- 2x 6 pin DIP switches (DIP-12 0.3” spacing) for baud rate selection. A wire link may be used instead for fixed baud rate operation.

ICs

- 1x Motorola 6800 processor, any speed grade 1MHz or higher
- 1x 1.8432MHz DIP-14 (full can) oscillator
- 1x 2716 EPROM or 28C16 EEPROM or compatible, a 2732 will also work, 2764 may be used with a 28 to 24 pin EPROM adapter.
- 2x 6850 UARTs. For 115,200 baud operation (not recommended) a 68B50 or better is required, and even a 68B50 is slightly out of spec running at 1.8432MHz input clock over its rated 1.5MHz, a 2MHz rated Hitachi 63B50 or similar is required to be fully in spec at that speed.
- 2x MAX232 UART line drivers
- 16x 2114 1k x 4-bit SRAM. Installed in pairs, up to 8 banks.
- 4x 74LS244 octal-bus drivers
- 2x 74LS245 octal-bus transceivers
- 1x 74LS193 4-bit counter

- 1x 74LS74 dual D-flip flop
- 1x 74LS73 dual JK-flip flop
- 1x 74LS02 quad NOR gate
- 1x 74LS04 hex inverter
- 1x 74LS00 quad NAND gate
- 1x 74LS30 8-input NAND gate
- 2x 74LS138 3-8 decoder
- 1x 555 timer

BOM Table

Reference Designators	Qty.	Part	Digikey Part #	Alternative Digikey Part # (ICs are HCT instead of LS)
C1, C2, C3, C4, C5, C6, C8, C10, C12, C14, C16, C17, C19, C20, C21, C22, C23, C24, C25, C26, C27, C28, C29, C30, C31, C32, C33, C34, C35, C36, C37, C38, C39, C45, C53	35	0.1uF 50v Ceramic Capacitor, .1” pin pitch	399-14036-1-ND	
C7, C9, C11, C13, C40, C41, C42, C43	8	10uF 16v Tantalum Capacitor, .1” pin pitch	478-TAP106K016CRSCT-ND	
C15	1	100uF 16v Aluminum Electrolytic. .1” pin pitch, diameter < 6.3mm	P19632CT-ND	
C18	1	470uF 16v Aluminum Electrolytic. 5mm pin pitch, diameter <13mm	P12375-ND	
C44, C46, C47, C48, C49, C50, C51, C52, C56, C55, C56	11	1.0uF 50v X5R Ceramic Capacitor, .1” pin pitch	445-173257-1-ND	478-11561-1-ND
D1	1	T1-3/4 (5mm) low-brightness red LED	160-1707-ND	
D2	1	1N4148 DO-3 diode	1N4148FS-ND	
J1	1	Barrel_Jack, 2.1mm x 5.5mm	CP-202A-ND	CP-102A-ND
J2, J4	2	DB25 Male, right angle board mount. 7.7mm pin edge offset, 9.12mm mounting hole offset	2057-DB25-PL-24-ND	
J3	1	2x20 0.1” male pin header	10129381-940002BLF-ND	S2011EC-40-ND
JP1	1	1x2 0.1” male pin header	2057-PH1-02-UA-ND	S1011EC-40-ND
Q1	1	2N3904-AP TO-92 package with staggered centre pin	2N3904BU	2N3904-AP
R1, R2	2	620 ohm 1/4W 5% Carbon Film	620QBK-ND	
R3, R4, R5, R6, R7, R8, R13	8	10k ohm 1/4W 5% Carbon Film	CF14JT10K0CT-ND	
R9, R10	2	470k ohm 1/4W 5% Carbon Film	CF14JT470KCT-ND	
R12	1	470 ohm 1/4W 5% Carbon Film	CF14JT470RCT-ND	
SW1, SW2	2	DIP-12, 6 pole DIP switch, 0.3” spacing	2449-KG06E-ND	
SW3	1	6mm through hole push button	679-2428-ND	
SW4	1	ESWITCH 100SP1T2B4M7QE right angle toggle switch or compatible, SPDT switch	EG5419-ND	
U1	1	74LS00 DIP	296-1626-ND	296-1603-5-ND
U2	1	74LS04 DIP	296-1629-5-ND	296-1605-5-ND
U3	1	74LS193 DIP	296-1605-5-ND	296-2142-5-ND
U4	1	74LS73 DIP	296-26520-5-ND	296-3541-5-ND
U5	1	74LS02 DIP	296-1627-5-ND	296-3541-5-ND
U6	1	74LS74 DIP	296-1668-5-ND	296-1605-5-ND
U7, U8	2	MC6850 (68A50/68B50/6350/63A50/63B50) DIP	Vintage, eBay	
U9, U10	2	74LS245 DIP	296-1655-5-ND	296-2105-5-ND
U11	1	MC6800 Microprocessor	Vintage, eBay	
U12, U13, U14	3	74LS244 DIP	296-1653-5-ND	296-1610-5-ND
U15, U16	2	74LS138 DIP	296-1639-5-ND	296-1608-5-ND
U17, U18, U19, U20, U22, U23, U24, U25, U26, U27, U28, U29, U30, U31, U32, U33	16	P2114 SRAM DIP-18	Vintage, eBay	
U21	1	2716 EPROM or 28C16 EEPROM	Vintage, eBay	
U34, U35	2	MAX232ECN or similar DIP	296-26139-5-ND	
U36	1	74LS30 DIP	296-3693-5-ND	296-2111-5-ND
U37	1	NE555P DIP	296-NE555P-ND	ICM7555IPAZ-ND
X1	1	1.8432MHz Full-can (DIP-14) oscillator	X939-ND	110-MXO45-3C-1M843200-ND

Reference Designators	**Qty.**	**Part**	**Digikey Part #**	**Alternative Digikey Part #** **(ICs are HCT instead of LS)**
X1 Socket	1	DIP-14 (0.3”) crystal socket	ED90428-ND	
U7, U8, U21 Socket	3	DIP-24 (0.6”) socket	1175-2554-ND	
U9, U10, U12, U13, U14 Socket	5	DIP-20 (0.3”) socket	AE9998-ND	
U11 Socket	1	DIP-40 (0.6”) socket	ED3033-ND	
U1, U2, U4, U5, U6, U36 Socket	6	DIP-14 (0.3”) socket	2057-ICS-314-T-ND	
U3, U15, U16, U34, U45	5	DIP-16 (0.3”) socket	AE9992-ND	
U17, U18, U19, U20, U22, U23, U24, U25, U26, U27, U28, U29, U30, U31, U32, U33 Socket	16	DIP-18 (0.3”) socket	2057-ICS-318-T-ND	