電動鬼

Supercomputing Research and Development Project

After my order gets through, I will be designing a prototype, which is nearly a Xerox copy of the final hardware, however, with a generation old hardware. Gotta do. I also offered myself some space on the firmware ROM, as the 32KB FRAM was out of stock – I chose 64KB, ***DOUBLE*** the original chosen capacity, to give me a slightly more headroom to be stuffing few more programs other than an Operating System kernel. It was what Mouser had on hand as well. The 5MHz MEMS oscillator was also on order (SiTime MEMS oscillators were sold like hot pancakes!!!) - I can tell it's quite popular as it kept disappearing there on Mouser website – from “5,670 available to order” to “Sold out - On order” within few weeks... Luckily, I have a SiTime 5MHz MEMS oscillator on hand.

I placed order on: ISSI 16 Megabytes 42S32400D VFBGA-90 SDRAM, SGS Thomson LF33ABDT-TR 3.3 Volts LDO, FTDI FT232RQ QFN-32 converter, FTDI Vinculum II QFN-32 USB host controller, Ramtron FM24V05-G SOIC-8 FRAM and, finally a nice and big SMD filtering capacitor from Nichicon – an Aluminum-polymer 4V 2,700uF capacitor – hopefully they get through. I will be ordering some more later on as my budget allows (my budget may change in the future...) to finalize this order. I may use either address latching / FET-based analog IO switch or a CPLD to complete the SDRAM controller logic – Jazzed may know mucho more... It has to be latched directly as I won't have enough pins on Propeller for address pins on the SDRAM anymore, as there's a finite numbers of pins on Propeller – there's only 32 pins.

I'm thinking to put an Ethernet jack on the prototype board to establish the communication to the optional boards – to allow the usage of known-clean setup in a cable – there's no need for the trial and error on cabling – it will be nice enough for up to 100Mb/s. The Ethernet cables are cheap enough and plentiful, so why not?

The purpose for the prototyping hardware is to really do a hand-on probing of what could be nearly the same setup as the final hardware – compared between old and new hardware: Propeller 1 and Propeller II – SDRAM and DDR-I – it should be consistently the same so I can determine much about it. COGs between two generations of Propellers are different: The first one have simple 2-stage COGs, while the new Propeller II will have superscalar 4-stage COGs.

The important objective first, is to make a kernel capable of controlling the internal events such as branch prediction – first read the instruction / data on the COG RAM or off-die RAM, whichever comes first, then set the threads up in a fashion that the times could be reduced in branch predicting – predicting the correct answers for what could take forever for the COGs to finish – by figuring out what the codes will do, step-by-step. The microcode surgery cannot be done on SPIN, however. That was why I will have to do it in ASM, that I am familiar with (x86 assembling rules is still the same as PASM, the only difference is the type of machine code AKA ISA (Instruction Set Architecture) – which shouldn't be too hard). The main trade-off between misprediction and the completed execution is to really figure out the "sweet spot" of how the threads are to be decoded from the firmware FRAM and/or via USB controller, from the hard drive or jump drive. There is no fast and hard rule as far as I suspect.

So, with some fancy instructions under the kernel's hood, there comes the serious need for optimizing, as there is limited size of memory off-hand. I can do LMM or XMM, but the first thing first is to optimize the kernel down to 512 bytes so there is enough for the branch prediction and/or possible Out-of-Order thread scheduling (to allow OoOE processing – which I think may be doable on Propeller II – I may try it on Propeller 1 – this thing still got couples of surprises left in it.) There should be 128 bytes for branch hints/predictions and other remaining memory for extra GPRs/SPRs features as well as user's "sandbox" memory – another 512 bytes to 1KB. It's best to go by "pinch a bit of salts and ya'll know how much!" as COG's RAM is severely limited.

After software is satisfied – kernel up and running as well – with up to 70 to 90% hits on branch prediction caching (meaning Propeller will work much better) – but I won't be too surprised if it ain't "Pentium Perfect" but 70% is good enough providing 128 bytes is to be allocated there for this kind of scoreboarding. Then, I will do some sound analyzing and FFT to test (punishing the Propeller to see if the OS can hold the water) – after that, I will build few more identical boards to stack up to see how they get along with each others.

There is a reason for to use FRAM here as well – I may be updating the firmware more and more often as I go from here to there – the kernel I'll be using in Propeller II is going to be very, very sophisticated (meaning it may do Out-of-Order execution branch scheduling and other speed-boosting features) so it's kinda straightforward to do Propeller 1 machine codes and write up what I found out about its mind. This kind of updating on the I2C flash ROM is a no-no here as I would probably wear this thing out in rapid clip. FRAM is the best way to go as they're nearly made of tiny Piezoelectric ceramic cells (down to 130nm – Ramtron and TI have 130nm SOI FRAM which I am going to use) – in the place of gate capacitors in DRAM of the same logic / technology. It's supposed to give me 100 trillion rewrite cycles so it's great. Even long after Dendou Oni's built, I will still be playing with this board as well, which makes sense.

What this prototype board won't have, however, is somewhat hardware-related. The Propeller II have ADC / DAC on the pins, something that the current Propeller 1 don't have, which will allow me to try Spread-Spectrum Signaling technique on Propeller II to allow transceiving from 70 Megabytes per seconds, to several Gigabytes per seconds (by performing the ADC-orientated frequency / voltage hopping on the data to be sent) which will be necessary to do so to transfer time-sensitive data around the array.

It still will give me an excellent opportunity to explore what would be the basic foundation of Propeller II stack cards themselves. What's the difference in technology between two generations of Propeller chips and their external RAMs won't matter, because what actually matter the most is creating nearly the same environments.

I have been wondering for some time, would the 4-layer PCB be good enough for the microcontroller test board? (Keep in mind, with some carefully circuit tracing, it's possible to get away with BGA chip soldered on two-layered PCB...) I wouldn't worry way too much about the details on board as the technology used by the RAM and Propeller is quite simple – no need for trace matching. Propeller II board may need up to 6 layers, however, due to both complex BGA package and the needs to conceal extremely high speed signal (many times it will be radiated as radio noises – there will be extremely aggressive voltage / frequency fluctuation on Tx/Rx traces as necessary to modulate the data to be sent to the other Propeller II – DDR is also particularly sensitive to it) as well as the need for trace matching for DDR chip close to Propeller II chips.

Power supply noises, as far as I am concerned, isn't important with Propeller 1 chip as the only regulator that is to be used on prototype board would be a Low-Dropout regulator – 1A 3.3V LDO for the whole hardwares. The ripple from LDO is quite low compared to the wall warts (with cheap and heavy adapter – there will be unavoidable 60Hz ripples – sometimes capacitors wouldn't do great jobs – while it can conceal 50 – 170kHz ripples.) Propeller 1 and its companion chipsets (including RAM) will have to have decoupling multi-layer ceramic capacitors, as it is actually best to cover some leftover ripples than just grieving over the fact that the Propeller IDE keeps giving you the error message saying that the EEPROM write failure have occurred. It's much more critical with Propeller II-based hardware, however. From experiences and observation, to keep high-speed logics happy entails very careful attention in designing power supply rails and to place copious capacitors in decoupling rails. Why decoupling MLCC on every chips' Vdd pinout to ground? It's to keep the noises from shooting back to the +3.3V / +1.8 V logic power supply rail and messing up the other logic chips (meaning data loss will become more likely). And using higher capacitance low-ESR capacitors around power supply area and its feed-through wouldn't hurt – in fact, it will be totally clean as well! I use Nichicon low-ESR Aluminum Polymer capacitors, they're great (more consistent charge holdover – especially important in power rails) and is excellent.