12/23/10

Dendou Oni

Supercomputing Research and Development Project

Since I got basic needs figured out, I got hardware lists on my hands (still need the best PCI-express switcher, though – needs to be intelligent like Lucidlogix Hydra 100 / 200 chipsets so that way the seed packets are handled properly) – now I gotta start small. To start out small, I already have DIP-40 version of Propeller, and DIP style crystal oscillator (I know it's lousy, but I am starting out small, remember? Eventually, I may try to use SiTime MEMS oscillator on QFN-4 to DIP-4 oscillator style board to test the kernel.) and the 128 kilobytes EEPROM chip from Microchip (leftover from sampling) to test out the boot firmware using some leftover kernel codeset from my GreenOmega development day, so I may use the script methods to do all the jobs faster and efficiently.

But, how do I start out as I expand its circuitry? For up to 4 Propellers (will order more DIP-40 Propeller chips and Ramtron's 32 kilobytes FRAM soldered on SOIC-8 to DIP-8 board, and probably 5 MHz SiTime MEMS (SiT8003) clock chips – all together for exploratory orders), I may start with Totem stacking logic to test how much I can push all of the data into each strands of wires without exceeding both the kernel interrupting / polling limits and to be able to handle doing higher throughput FOR each microcontrollers (I shall say I can only max out at 20 Megabits per seconds (3 MB/s) for all FOUR chips, due to their CPU clock speeds and the limitation of 350nm transistors inside the pin drivers... Propeller II may do much faster, though.)

Why Totem stacking? It was the most common way to get more horsepowers from the x86 CPU. It was to, well, stack another x86 CPU of the same model and speeds atop the main CPU (like AMD Athlon MP, each in two Socket A (FC-PGA 423) sockets all filled up, for example) – and it shouldn't use too much wires. Therefore, I intend to use this topology to test Propeller chips (Once it outnumbers the requirements of Totem stacking topology, I will convert to hybrid supercomputing topology, both Common Bus and Hypercube.) - and it offers the opportunity to best fine-tune the firmware's Operating System (OS) kernel's responses and reactions so it doesn't overwhelms the whole system. And with Totem Stacking, it also may allow me to use the SDRAM memory chip (which I don't think I will need to use early in the development, but with final design, it will strictly require them to be used, to reduce the chance of Propeller chips crashing. I won't have to worry about storage for now, as I am going to use real simple codeset and instruction – as simple as possible, to see how well they respond.)

If you guys have already objected about why I said that the crystal oscillator's lousy – you will see in a bit: The synchrony between IRQ requests and send-outs should be as precise as possible. We have very small headroom here as the Propeller II chips on the hypercube processor card exceed their bandwidth to few Gb/s, the window allowable for errors becomes very small. (Ever wonder why Gigabit Ethernet is so complicated then even the quintessential Fast Ethernet?)

And, thermal drift in resonators should be kept as small as possible, and it's not in our controls. Even if you can cool the quartz oscillators, some of them still get hot and it can and actually affect the ceramic disc in it, causing drifts. And, I do not want any drifts: Once it drifts in wrong spot, it will cause Propeller II to send out packets at the wrong time (Familiar with so-called “Packet Crash”? It's caused by packets from different processors being sent at the same time. Just like in Ethernet frameworks.)

And, yet again, even with SiTime MEMS, Propeller II chips will still require synchronization between each others so it's ensured that everything stays accurate and predictable (even with COGs executing in unpredictable manners as is necessary to reduce the chance of pipeline stalling...) so we won't have to deal with packet “Crash and Burn” - and with MEMS clock, fortunately, we won't have to issue the recalibration packets to resynchronize the whole affairs more often as is likely with programmable Quartz oscillators (the clock with the worst ripples compared to fixed-frequency crystals and MEMS clock chips altogether), as well as for to deal with temperature compensation. I picked out MEMS oscillator chip is because it stays stable even with heats coming from Propeller II chip itself, and they don't drift. So, less re-syncs, more works.

Okay, about the memory... Why do I use non-violate RAM for to hold the boot firmware? I intend to make as much adjustments as possible before everything's REALLY perfect, and I am as sure as heck would wear the EEPROMs out quickly. Sure, it's cheap. But it isn't so cost- and design-efficient when soldered onto the board, as it will require the headache-inducing downtime repairs, replacing the dead EEPROMs once it gets shot after several user-setting updates and BIOS updates. (Flash don't hold their charges well either. That's why all flash chips made under 50nm are made with High-Dielectric Metal Gate (HkMG) process. Still, even with Beryllia insulating layer, they still will lose their charges over time. Effectively, the flash chip starts dying very slowly after being written. Everything comes with trade-off, anyways.)

One trade-off with FRAM (Ferroelectric RAM), they will have to be rewritten as it's being readed, fortunately, the controller in there is intelligent enough to be able to refresh the whole memory without wiping the OS off the FRAM cleanly. Another one, which I will not worry about, the capacity of holding its data after being hammered with X-rays. Ferroelectric material apparently do have photoelectric response (which is why

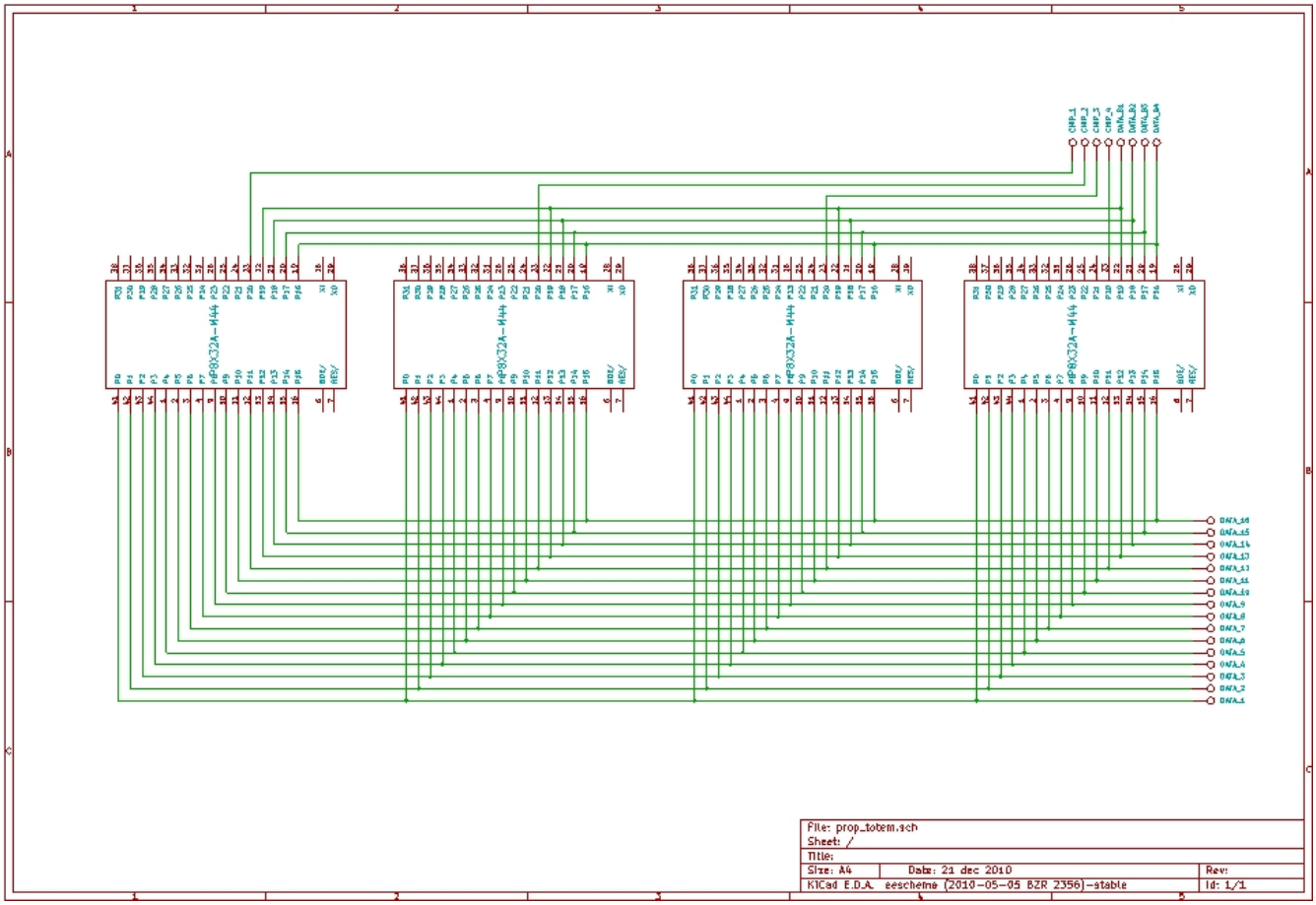
the clock resonators are covered in the darkest spots ever convinced – in the metal package, and with some, black plastic DIP/QFN package – the older quartz oscillators had nasty tendency to drift quickly when shone upon, as it used to be packed in glass tube, like the vacuum tubes. But, X-rays still will pass through the oscillators' package as if it's not there, messing everything up.) Why I am not worried about it? Because, I am going to use it in some place totally X-ray-safe so I don't risk cooking the chips with the ultra-shortwave lights hammering it. With some equipments, like cell phone, I would definitely be suspicious as the airport luggage X-ray machines would act like the degaussers for that FRAM chips in the phones or PDA passing through that thing... (Imagine yourself standing near a X-ray scanner and after you pick it up, you check your phone: “Uh? NOOOO!!!! My important documents are wiped off!!!!”)

Why FRAM? How about using MRAM? If it's true that Propeller II chips do use SPI bus for the boot firmware, I would definitely think about using Everspin's SPI MRAM chips. The worry of having X-rays accidentally wiping the FRAM clean would at least be removed, but the RAMs in Propeller II chips still would be affected as well.

And, to put it that way, I just wanted to make sure my research project is as reliable as possible so that way it won't be a waste of money, effort, and time. (and it would ensure that the down-times won't occur too often, except during the informed OS update, which the users would be informed that the OS update is proceeding then is open to everyone again after reboot and made sure everything works fine afterward.)

Also, the Totem stacking is a bit simple, but is already indispensable as it is excellent in testing the OS into seeing if the chips and the associated running firmware OS kernels are working as desired. The electrical setup may look like it's designed to violate the IO contention rules, but let's break it down for a bit: When Propeller chip is sending out a voltage signal on a 16-bit data bus, the 4-bit host (DATA_Bx) bus will be toggled up with data IRQ packet sent to particular requested Propeller chip from host lines so that way 16-bit contention won't occur (Remember the “Packet Crash”?) Oftentimes, even the unwanted Propeller chips may have the data that the requesting Propeller may want, so they may ask for a bit more details, then send back the results, then ALL propeller chips will be given IRQ_ACK: FLAG_1 messages (except for the “master” chip sending the acknowledge), letting them know that the asked conditions are met and is seriously satisfied with what it wanted. On Propeller II setup, however, the 16-bit bus here may become 32-bit bus so the Propeller II chips may be able to move their data lightning fast (maybe up to 4 Gb/s) so they can really get their jobs done swiftly. Outside the Propeller II chip, the interconnection transactions may occur out-of-order, even though they have to still be synchronized (Remember the narrow window for the mistakes at “supersonic” data throughput?) so that way the data can be moved around more efficiently.

Totem stacking setup (EEPROM (or FRAM) and MEMS/crystal oscillators omitted):



Sorry about crappy picture quality... I was using KiCAD – it was great, but unfortunately, it got severely limited library, and is kinda of limited values in professional works, but HEY!!! It's totally free! And, it got PCB author too, so it's not so bad... So, here in this diagram, I mentioned 4-bit downlink bus (as well as chip selector bus for the PC or the other microcontrollers to force the manual selection of packet delivery), and as well as the sixteen-bit bus to actually move high-speed packet (20 Mb/s Max. chip limit) – If the packet transactions occurs simultaneously (In-Order) on 16-bit bus here, it can get as slow as 256 Kb/s or worse - and it is not doing that simultaneously, it would essentially be transacting out-of-order (although electrical rule for the IO pins will still apply – the chips has to use free pins to “zip” their packets through) – up to 20Mb/s or hopefully faster.

With 32-bit buses from all 16 TFBGA-132 Propeller II chips together, I can actually have 64-bit NRZ pipeline here, which is screaming fast in its right. But for now, pokey 3 Megabytes / sec. bandwidth gotta do... What this Totem stacking topology here will not have, each Propeller II will have their own individual 8-bit buses to the XMOS XS1-G4 chips (or if I am lucky, cheap TI Quad-core VLIW floating-point DSP chips) to

provide for the floating point processing and DSP functions needed to lighten the Propeller II chips' loads (to keep everything balanced), and the 32-bit buses will be tied to a FPGA to talk to the seed computer via PCI-express 1x bus, transferring the data and instruction back and forth.

While I am at it, I will be placing the exploratory orders from Mouser Electronics, Inc. website, www.mouser.com, which is handy since I can still order them in Japan. Here, I am going to order few parts to be testing the firmwares for reals. And, also to test the feasibility of floating point processing using C++ - based FPU emulation software (it can be slow, but we don't have much choice...) - Here's the part list:

Exploratory order (from mouser.com):

619-P8X32A-D40 - Parallax, Inc. Propeller (DIP-40) - $7.99 (x4: $31.96)
877-FM24C256-G - Ramtron 256Kb (32KB) I2C FRAM (SOIC-8) - $6.69 (x4: $26.76)
604-DD12GWB - Kingbright Green LED bar graphic indicator - $2.34
788-8002AI233E-5.0T - SiTime MEMS oscillator* (3250) - $3.39 (x4: $13.56)
535-08-350000-10 - Aries Electronics, Inc. SOIC-8 to DIP-8 adapter - $5.96 (x4: $23.84

- Totals would come at $98.46

- If 5 MHz frequency selection isn't available from Mouser, ask either SiTime or Mouser, they will send you a programmed oscillator set at 5 MHz. However, if you got a blank MEMS oscillator and a programmer from SiTime, you're lucky - just set it at 3.3V and 5 MHz (PPM and pin enable is your decision, however, I prefer it to be of lower PPM jitters and at OE so I don't bother everything.) - Plus, if I can't find an oscillator SMD to DIP-4 half-height adapter, I will cheat and use 7050 (or 5032) MEMS oscillator and put in disposable DIP-8 socket so I can plug it in without bending the soldered-on leads or breaking the oscillator, only the damages will be placed AT the DIP-8 socket.

So, the exploratory order within $100 budget, not too bad. I already have prototyping breadboard from Radio Shack from last year, so it isn't in the total. I will get bigger or multiple snap-in (Lego-like) prototyping breadboard if necessary.

More to come: Software development coverage and / or exploratory order coverage - I will be doing a bit more after within few months I get my grubby hands on the exploratory order and is working. I will be also covering a bit detail of the operating system kernel / firmware frameworks that will be used in boot image for both first- and second-generation Propeller chips.