This is a description of COED's time-of-day clock.

There are three major difficulties in the maintenance of a time-of-day clock within COED. First, the clock must be initialized when the system is started, and when the system resumes operation after a power failure. Second, all of the CPU's in the system should keep the same time, and finally, each of the CPU's must be able to operate a time-of-day clock in the absence of one or more of the other CPU's or of the hardware time-of-day clock, the CHRONO-LOG.

Therefore, it is clear that there must be a system within COED that transmits the correct time-of-day from one of the CHRONO-LOG clocks, but not both, when the systems starts, resumes, and periodically. There is also a need to pick a particular CPU and instruct it to maintain the time-of-day for all of COED, starting from a particular value (the SDATE command). For "fail-soft" reasons, this alternate source should probably be any of the CPU's in COED, choosen at will. Notice that the time may not be transmitted at system start-up and be expected to remain accurate for the long periods that COED is supposed to remain operational, since the internal, real-time clocks are accurate to no more than one part in 10 to the fifth.

There are perhaps two schemes for transmitting the time among the CPU's. The first is a simple system wherein the "EXEC" process within each CPU sends "packets" over the inter-CPU links containing the time. These processes would have to decide among themselves which one has the "correct" time, and would have to periodically reassure each other that they are all on time. There are several difficulties with this scheme. First, because the time must be communicated over the inter-CPU links, competing with all other inter-CPU, inter-process traffic, it is difficult to ensure that the time is still valid when it reaches the destination process and is used to set the time-of-day in that CPU. Further, it may require more than one transmission over a CPU-link for the time to move from the CPU keeping the true time to the destination CPU. The accuracy of the time reaching a CPU will thus vary, reaching a matter of seconds. Worse, the CPU's will not have the same time-of-day. The time can be accurately propagated by stopping all inter-CPU traffic, and all processes, except the "EXEC's", in both the sending and the receiving CPU's, but that is at best undesirable at system start-up and power-restoration, and is probably unacceptable for maintaining the time under normal circumstances.

The other scheme is largely the same, except that the task of communicating the correct time is moved down from "user" code to "interrupt" code. Because it takes very little time to transmit the 32-48 bits required to specify the time-of-day, and because the time may (and probably should) be transmitted from the most accurate, available time-base on a very low duty cycle (perhaps once a minute or once an hour), moving the function to "interrupt" code incurs no significant cost. I claim that the total code involved with this scheme is no more, and probably less, than the code associated with the first scheme. Further, this scheme over comes the difficulties of the other one.

This second scheme is based on the transmission of a few (32-48) bits of data over an interrupt line to all CPU's simultaneously. The data is to be transmitted as a single, bit-serial character, asynchronously over the interrupt line. The bits are transmitted by causing an interrupt within a particular time

(15 milliseconds) to signify a "0", and not causing an interrupt within the same time to signify a "1". This long character (which is the time itself) is preceeded by a preamble of at least 1.5 seconds of quiet (no interrupts) and a single "start-bit" interrupt, just as 8 bit characters are transmitted over a standard, low speed asynchronous communications line. The time is received by all of the CPU's accurate to within plus or minus 1.5 five millisecond clock "ticks". The time is transmitted periodically (perhaps once a minute) by the current "master" CPU. If the master fails to transmit the time in a timely fashion (say within 2 minutes after the last time the time-of-day was transmitted) a second CPU may take over. If the second CPU fails to take over, a third may, and failing that, the fourth, lonely CPU can maintain his own clock. Which CPU is which is determined from the "CPU ID", available at system start-up. If any CPU starts transmitting out of turn (that is, starts transmitting before the current master is scheduled to start), all the the CPU's including the current master listen to the up-start's time. If the time is valid, the impolite CPU becomes the "master", and is expected by all of the others to shoulder the burden of transmitting the time. If the time is invalid, the current master is expected to immediately transmit the correct time. Thus any CPU may trivially execute an "SDATE" command, or, upon belatedly joining the other COED CPU's, may request the current time-of-day, without waiting for the "master" to get around to transmitting it.

Implementation

All of this is accomplished using the Multi-processor Interrupt, RMI or level 3, and the normal 200 hz. real-time clock interrupt, CLK or level 6. A minor modification of the hardware is also required to permit all eight RMI lines (one from each CPU and one to each CPU) to be tied to a common, active-low, open-collector bus.

The code for RMI does little more than count interrupts. When the count gets too big, indicating that something is wrong with the RMI bus, it disables itself. It also sets the "bit-time" (see below) to be two 5 msec. clock ticks short of expiration for self-synchronize with the bits being received from another CPU or to prematurely terminate a quiet preamble. It ignores interrupts requested by its own CPU.

The code for the normal clock level is substantially more complicated. Besides maintaining the current time-of-day, running watchdog timers for various drivers, and stimulating the scheduler, it must decide what and when to transmit to the other CPU's. On CPU's with A CHRONO-LOG digital, time-of-day clock on a "MODAC", it must also decide whether and when to see what time they are keeping. The time-of-day keeping functions are accomplished with a simple abstract machine. This abstract machine is actually a more intuitive realization of the many required state changes. On each five millisecond CLK interrupt, a counter of the current "bit-time" is counted up toward zero. At zero, the post-processing code for the current abstract machine instruction is executed. This code can decide to reset the bit-time counter and wait some more (e.g. when transmitting/receiving a quiet preamble or when transmitting/receiving a bit of the time-of-day), or the code may decide that the instruction is finished, and take an exit. Since each abstract machine instruction consists of an op-code, a 2-bit argument, and two abstract machine "exit" addresses, the post-processing code may choose an appropriate state transition, depending, for example, on whether some other CPU has requested the time or whether the "master" CPU has sent the time in time. When an abstract instruction exits, a new instruction is fetched from the indicated abstract machine word and is executed. The new instruction may perform some function and immediately exit (e.g. change one of the auxiliary state bits) or it may pause, so that its post-processing code will be executed when the new "bit-time" expires.

A third interrupt routine is also involved in the time-of-day clock system. It is the 1 interrupt/second signal provided by a CHRONO-LOG clock. Certain abstract machine instructions may specify that the CHRONO-LOG clock interrupt should be armed. It will be armed, and should occur within a second, provided that the auxiliary abstract machine state bits permit it (i.e., this CPU is the master, not a slave, and our CHRONO-LOG clock is healthy). When the interrupt occurs, the time-of-day is read and checked for validity. If it's good, it is converted to COED internal format and forced into this CPU's time-of-day counter. The interrupt then disarms itself.

On the RMI bus, the time-of-day is sent as a string of 49 bits. A zero is sent as an interrupt during a 15 millisecond period, and a one is sent as the absence of an interrupt. The first 48 bits are sent most significant first, just as they come from the current time-of-day clock in memory (words CLK$0, CLK$1, and CLK$2). The final, fourty-nineth bit is a stop bit, saying essentially that the transmitting CPU was able to finish without being interrupted. Because the first word of the time-of-day, containing the year and month, is always positive, the time-of-day always starts with an interrupt, and so all other CPU's can know which bit is which. If a CPU is only trying to change the year, and not the current time, then the middle word (seventeenth through thirty-second bits) are zero. There is no valid time-of-day of day with a zero second word, so a listening CPU can unambiguously decide whether it is receiving the complete time-of-day or only the year.

The time itself is the time that the transmission ends. That is, at the end of the fourty-nineth bit time, the current time-of-day is that indicated by the first 48 bits of the transmitted/received time.

The basic time-of-day clock routines are E3$TIM (the level 3, RMI, interrupt) E6$CLK (the level 6 interrupt) and SI$MCO (the CHRONO-LOG interrupt). One other routine is CLK$OP which causes the clock system to perform operations such as requesting the time and causing a CPU to rebel and assume responsibility for keep COED's time-of-day. Another useful routine is CLKCN$ which returns the number of even minutes in the current month of the current year.

The following words are used by the time-of-day clock system:

CLK$0,CLK$1,CLK$2 contain the current time-of-day in the standard, COED internal format, which is:
word 0, bits 0-7=current year, A.D., modulo 100,
bits 8-15=current month
word 1=minus the number of even minutes until the end of the month, and
word 2=minus the number of 5 msec. clock ticks until the end of the current even minute.

TIMBIT containing number of level 3 (RMI) interrupts received during this "bit time". Note that when transmitting, this word is adjusted so that a CPU will ignore its own interrupts.

TIMWD1,TIMWD2,TIMWD3 are a shift-register for transmitting or receiving the time-of-day.

TIMAST contains auxiliary status bits for the abstract machine
bit 12=1 kills the CHRONO-LOG clock interrupt at the end of an abstract machine instruction,
bit 13=1 indicates that the CHRONO-LOG interrupt has been killed by the occurrance of a successful interrupt during a recent abstact machine instruction,
bit 14=1 means the CHRONO-LOG clock is dead, and that we are to ignore and never arm it, and
bit 15=1 means this CPU is a "slave" to some other CPU's time-of-day, and therefore has killed its own CHRONO-LOG clock forever (bit 14). When 0, this CPU is the "master" of all of COED.

TIMBTM is a counter (up to zero) of the 5 msec. ticks remaining in the current "bit time".

TIMCNT is a bit (up-to-zero) counter for counting the 49 bits sent or received with the time-of-day.

TMSTAT is the current abstract machine instruction. It is used primarily for the "0" and "1" exit addresses that it contains.

TIMPST contains the address of the post-processing code for the current abstract machine instruction.

The following arguments for the routine CLK$OP specify the actions that it can perform:

ABSREB forces this CPU to rebel and start transmitting its version of the current time-of-day

ABSYER sends this CPU's year to all other CPU's, but doesn't change its master or slave status.

ABSREQ transmits a request for the current time-of-day from the current master.

ABSINI initializes this CPU after a power failure, including requesting the correct time.

ABSCLK is the same as ABSREB, except that it also makes this CPU revive its CHRONO-LOG clock.

The Abstract Machine Instructions

Each abstract machine instruction is 16 bits long and its four fields are arranged so that the first 2 bits are the "augment" field, the next 4 bits are the opcode field, and the next pair of 5 bit fields are first the "0" exit abstract machine address followed by the "1" exit. The "0" exit is so called because the code which implements the instruction exits uses R4=0 to decide to execute a "0" exit.

The following are the abstract machine instructions:

Instruction 0--Transmit a preamble after arming CHRONO-LOG clock. This instruction is essentially the same as instruction 1, but first arms the actual, hardware CHRONO-LOG clock interrupt and marks the fact in the word TIMAST. If this particular CPU does not have a CHRONO-LOG clock, no attempt is made to arm the non-existent hardware, but word TIMAST is marked as if an interrupt was expected. When it fails to arrive, the abstract machine program must act just as if the CHRONO-LOG clock were present, but failing. After arming the CHRONO-LOG clock, this instruction becomes identical to instruction 1.

Instruction 1--Transmit a preamble.
A preamble consists of a period of time without any interrupts on the RMI bus. There are four different preambles available, and which one is "transmitted" depends on the augment field of the abstract instruction. This instruction essentially uses the augment to pick a negative number of 5 msec. ticks. The number is then put in the word TIMBTM, the "bit time" counter. After the long "bit time" constituting the preamble, this instruction checks to see that an even minute is not going to end in the next few ticks. If we are near the end of an even minute, we start another, relatively short preamble that will last until after the start of the next even minute. This is done so that instruction 3, which transmits the time, does not need to worry about having the first or second words of the time-of-day change while it is being transmitted. If no RMI interrupts are detected during the explicit preamble, or during the short, extra, automatic preamble, the instruction takes the "zero" exit. An RMI interrupt will, by its self-synchronizing action, end the preamble prematurely and cause the instruction to immediately take its "one" exit.

The preamble may be any one of four different lengths. It may be a "calm-preamble", which is the interval that the current "master" (i.e. the CPU currently responsible for the time-of-day) waits between broadcasts of the time-of-day. It may be a "master-preamble", which is the time the master CPU waits after its broadcast of the time is interrupted before trying to retransmit the time. The preamble may the a "rebel-preamble", which is the time that a "slave" CPU (i.e. any CPU other than the current master) waits before transmitting a time-of-day that it wants the others to accept. Finally, it may be a "slave-preamble", which is the time that a slave CPU waits before deciding that the master has died. If the master is quiet for a "slave-preamble", it must be dead. The last two preambles vary for each CPU, and depend on the priority/reliability of time-keeping ability of the CPU, which is defined by the parameter MODAC$. This variation keeps two or more CPU's from trying to transmit over the RMI bus simultaneously, and it provides for an orderly transition when a master CPU dies.

Instruction 2--Receive the time-of-day.
This instruction receives the time-of-day. It assumes that the first interrupt (zero bit) of the 49 bits of time has already been received. It simply waits until it has accumulated all 48 bits of time and the stop bit. If the stop bit was bad (i.e. the interrupt did not occur), this instruction takes the "1" exit. If the stop bit was valid, but the second of the three words of time-of-day was zero, it sets the received year in this CPU's clock (word CLK$0) and takes the "1" exit. Otherwise, it forces the entire, received time-of-day into this CPU's clock (words CLK$0, CLK$1, and CLK$2) and takes the "0" exit.

Instruction 3--Transmit the time-of-day.
This instruction transmits the time-of-day to the other CPU's. If the augment for the instruction is odd, it zeros the second word to indicate that only the year is valid. If another CPU causes any interrupts on the RMI bus while this instruction is running, it immediately stops and takes the "1" exit. Every time it needs to cause an RMI interrupt, it decrements the word TIMBIT. Then, at the start of the following bit time, it takes the "1" exit just in case TIMBIT is not zero. Therefore, this instruction will always immediately take the "1" exit if RMI is broken.

This instruction must always be preceeded by a preamble (instructions 0 or 1). The instruction simply adds 49 to the third word of this CPU's current time-of-day and assumes that never affects the first or second words. The assumption is valid provided a preamble has just ended because a preamble never ends near the end of an even minute.

Instruction 4--Request the time-of-day.
This instruction is essentially the same as sending a preamble with instruction 0 and then sending an invalid time-of-day. It takes an augment which is used just as in instructions 0 and 1. It takes the "1" exit if any RMI interrupts occur during the preamble. The only difference between the first part of this instruction and instruction 0 is that if an even minute is about to end, no extra, automatic preamble is added.

After the preamble is transmitted, a single RMI interrupt is transmitted. The instruction then sends 48 silences (i.e. it transmitts 48 ones). Thus, the time-of-day transmitted does not have a valid stop bit. Further, the initial interrupt interrupts the master, if the master happens to have already started sending the time-of-day. The abstract machine program in the master CPU detects the invalid time-of-day and then sends the correct time.

Instruction 5--Set MASTER/SLAVE bit.
This instruction is used to set or reset the MASTER/-SLAVE bit and to set the CHRONO-LOG-disable-bit (bits 14 and 15 of word TIMAST). The bottom two bits of the augment are used, with the bottom bit replacing the MASTER/SLAVE bit and the next bit getting "OR'ed" with the disable bit. It always takes the "0" exit.

Instruction 6--Test the CHONO-LOG-was-here bit.
This instruction takes the "0" exit just in case this CPU is the master, the CHRONO-LOG clock is not dead, and we have not received a CHRONO-LOG interrupt since the last time we requested one with an instruction 0. That is, bits 13, 14, and 15 of word TIMAST are all zero. In any other case, the "1" exit is taken.

This instruction is used to determine if our CHRONO-LOG clock has just now died.

CLOCK (CR) STATE DIAGRAM