EMPIRICAL RESEARCH GROUP, INC.

P.O. Box 1176 • Milton, WA 98354
(206) 631-4855

A FEW BUS WORDS ABOUT THE MC68000

Until recently there were few ways of investigating the most advanced of the 16 bit processors, Motorola's MC68000. If you were able to pay the price, you could obtain a development system from Motorola or attempt your own wire wrap board. Now there is a tool available for the large group of people owning S100 bus machines that will encourage the full exploration of the MC68000. Empirical Research Group, Inc. (ERG) model ERG68-696 CPU brings the full power of the MC68000 processor to the S100 bus. The ERG68-696 meets or exceeds all requirements of IEEE-696 specification for S100 bus interface devices. Please note that meeting the specification is not trivial and is very different from ambiguous statements of compatibility, works with, designed to, or based on. To ensure compliance requires careful design then rigorous testing using sophisticated testing equipment capable of validating a large number of simultaneous and sometimes very subtle timing and state variables, not a task traditional suppliers of S100 based equipment are generally equipped or qualified to perform. Considering the importance of standardization to the user of S100 systems it is imperative that manufacturers stating compliance with IEEE-696 be prepared to validate such claims and provide documentation (such as logic analyzer displayed results), the alternative is simple, 'don't claim it if you can't prove it.'

IEEE-696 BUS REVEALED

ADDRESS BUS A0..A23, PHANTOM*

These 25 lines are responsible for selecting system resources on the bus, each card uses these lines to determine it's participation in a given cycle of processor activity ('a bus cycle'). In a system with a single CPU (the processor) and no 'TMA' devices the address lines originate on the processor (CPU) card, otherwise the address lines originate from the 'bus master' which could be another CPU or a TMA device (see TMA control bus for further discussion of how this works). Most processors do not generate all of these lines, in particular, eight bit processors (eg. 8080,Z80,6809,etc.) usually generate only A0..A15 and where A16..A23 are implemented. Generation of these pseudo 'fake' lines is accomplished by some additional hardware typically controlled via an IO Port. Various schemes exist to generate these so-called extended address lines ranging from simple eight bit 'latches' (storage devices which hold their value after being set) to elaborate 'memory mapping' circuits which are complex in both implementation and application (such schemes frequently refer to a thing called 'windowing'), let it suffice to say that there is no simple way to increase the inherent address space of a given processor. Sixteen bit processors have address space ranging from 64Kbyte (eg. TMS9900) to 64 megabytes (eg. MC68000). Sixteen bit processors generally do not produce address line A0 directly, this is because these processors have a basic data path of two bytes (referred to as a word), if 0 is the lowest address then the next word address is 2 the next 4 and so on. Since only ODD addresses would use A0 explicitly anyway it is eliminated. The consequences of this 'missing' address line are unremarkable in that virtually all processors capable of 'byte' addressing provide some means of generating a fake 'A0,' the MC68000 has two lines called UDS* and LDS* which are used to steer byte data to the correct portion of the processors internal data bus. The IEEE-696 specification requires all address lines to be valid and stable when the status valid strobe (pSTVAL*) is asserted during pSYNC. The spec further requires at least 16 address bits but allows 8 bits for I/O addressing (genesis 8080), the ERG68-696 exceeds this requirement (1) the address lines are guaranteed stable at the leading edge of pSYNC and are stable until the end of the cycle, (2) 24 bits of address (16 megabytes) are provided plus 4 memory management lines vailable on the CPU board which further define these 24 address bits as (a) supervisory program space, (b) supervisory data space, (c) user program space or, (d) user data space, (3) The MC68000 does not inherently provide for programmed I/O (ie. address space specifically allocated to I/O processes, genesis 8080 although Z8000, and 8086 are two current 16 bit processors which continue the tradition). The ERG68-696 allocates a 16K portion (expandable to 64K) of the 16 megabyte basic address space to simulated I/O space, all IEEE-696 specification concerned with programmed I/O are faithfully met (see STATUS BUS specifically sINP and sOUT). Note that the ERG68-696 is not limited in the types of instructions which may be performed within this I/O space as are the Z8000, and the 8086 however the user should scrutinize the effects of instruction which perform read/modify/write operations (genesis PDP-11) on I/O devices. (4) 'A0' is user definable as EVEN or ODD (see DATA BUS) for maximum flexibility, the availability of this fake 'A0' eases the use of older S100 type I/O and memory products with the ERG68-696 (note byte memory may not be used as program space). The PHANTOM* signal (the 4-2-81 addendum to the IEEE-696 standard makes this signal part of the address bus) is used to remove a slave from the system address space. PHANTOM* is normally used to allow 'BOOT' ROMs and similar devices to overlay a portion of memory, these devices are sometimes called 'SHADOW' ROMs. The slave (usually a memory card) must have the PHANTOM* function implemented in order for the process to work. The general method of implementing PHANTOM* is to include it in the address decode scheme, ie. if PHANTOM* is asserted the slave is not enabled and thus does not participate in the bus cycle. The ERG68-696

PHANTOM* is asserted by power on jump, optionally by RESET*, and optionally by address greater than 64K. The reset option allows a user to change the reset vector in RAM so that RESET* will cause the supervisory stack pointer to be loaded with an appropriate value and the processor to begin execution at the address entered rather than the default ERG68-696 monitor. The greater than 64K option allows memory cards with PHANTOM* implemented which cannot decode the 24 bits of address tc used in the bottom 64K.

The ERG68-696 PHANTOM* signal meets the requirements of IEEE-696 and the 4-2-81 addendum.

STATUS BUS

The status bus consist of eight lines which identify what kind of transaction is taking place during a bus cycle. The various cards on the bus can use this information along with the address bus to determine whether they are involved in the transaction and respond accordingly. The IEEE-696 status lines are:

sMEMR	memory read.	sWO*	write cycle.
sM1	op-code fetch (see discussion below).	sINTA	interrupt acknowledge.
sINP	programmed input.	sHLTA	processor halt acknowledge.
sOUT	programmed output.	SXTRQ*	16 bit data transfer request.

Memory read (sMEMR) indicates that any card which acts as a storage device and is being addressed by the address bus should begin putting the requested data on the data bus for consumption by the processor later on in the cycle. ERG68-696 provides sMEMR no later than the leading edge of PSYNC.

Op-code fetch (sM1) has the distinction of being the only IEEE-696 signal which is not fully supported by the ERG68-696, sM1 is optional and not required for full compliance with the IEEE-696 specification never-the-less a discussion of its possible uses would seem in order. First the reason sM1 is not fully supported by the ERG68-696 is simple, the MC68000 provides no means of discriminating between a read cycle which fetches an op-code and one which fetches an operand given that both reside in program space, the ERG68-696 therefore conservatively generates sM1 for every read cycle (sMEMR=true) from program memory. What is sM1 used for? Considerable research reveals two basic uses of sM1: The first to be discussed is simple the second is not. Properly designed dynamic RAM (read/write memory) cards invariably have some sort of system which allows an operation known as 'refresh,' required by all dynamic storage devices, to be performed in such a way and at such a time as to minimize interference (ie. wait states or extended read/write cycles) with the processor. The generic name for this capability is 'transparent refresh' although it may also be called statistical refresh, there is no need to discuss the details of such schemes as all known methods are served quite well by the ERG68-696 which puts its form of sM1 on the bus earlier than most other processor (eg. 8080,8085,Z80,8086,8088,Z8000,etc.) which should actually improve the chances of a fully transparent refresh. The secc use of sM1 is rather esoteric, an example is the ZILOG series of LSI peripheral devices (SIO,PIO,CTC,etc.). These devices decode the instruction (op-code) on the data bus during an sM1 cycle in order (to service their rather elaborate interrupt prioritizing scheme, of course they only work with a Z80 in any case so the simple fact that sM1 may be generated accurately by an 8085 or 8086 or some other son of an 8080 does nothing for them. The ERG68-696 can accommodate these devices with some help from the user by (1) setting the interrupt type jumper on the card to 'ZILOG' and (2) either using one of the emulator traps (1111 or 1010), some other trap, or simply placing the appropriate bit pattern in memory and reading it when the time comes. The device won't know the difference. Reasonable efforts have failed to uncover any other uses of this status line and while there may be other devices which use it in strange and unique ways the ERG68-696 with a little help from its MC68000 should be able to deal with them. A last comment on the now famous sM1 bit. From the preceeding discussion it follows that even if the MC68000 or for that matter some lesser processor provided a means of generating a true sM1 it would be nice to be able to defeat it in favor of the more general 'early read status signal' provided by the ERG68-696. After all, any processor generating an op-code fetch would expect the data fetched during that cycle to be a native instruction which very probably would not be the same as a Z80 op-code and even if it had the correct bit pattern it would not be likely to perform the same function. The ERG68-696 asserts sM1 no later than the leading edge of PSYNC.

The sINP and sOUT lines are treated together here due to similarities in implementation, sINP identifies a cycle in which a 'programmed I/O' read will take place and sOUT identifies a cycle in which a 'programmed I/O' write will take place. What is 'programmed I/O?' Processors can be divided into two basic classes with respect to I/O operations these are (1) processors which provide separate address space for I/O, called 'programmed I/O.' The 8080 class processor (8085,Z80,8086, Z8000 etc.) is in this category (so is the TI9900 series but in a very different way). These processors also include special instructions for use within the I/O address space, the rest of the instruction set is generally precluded from participation in I/O space, (2) The second category of processors use a portion of their inherent address space to perform I/O operations, called 'memory mapped I/O' the 68000 processor is in this category (as are the PDP-11,6800,6502,WD1600,etc.), usually no special I/O instructions are included for these processors. I/O operations in this case are performed using the same instruction as those used on memory. The second category is therefore more general but not without some hazards, for instance read-modifywrite operations can do strange and unexpected things when applied to some I/O devices, such as clearing a status bit in friendly cases or trashing a di in less benign cases. In any event it is nice to have the power of the full instruction set for use in I/O operations but it is wise stick to relatively simple operations equivalent to 'in' and 'out' until the nature of more complex instructions and their effect on a specific I/O device are well understood.

The ERG68-696 simulates the category 1 devices by allocating a portion of the basic 16 megabyte space to 16K bytes of I/O space (expandable to 64K). Operations performed in this address space cause faithful generation of the IEEE-696 sOUT and sINP signals. The full instruction set of the MC68000 is available for I/O operations. A simple equivalent to the 8080 'IN reg,port' and 'OUT port, reg' for the ERG68-696 is 'MOVE.b port,reg' and 'MOVE.b reg,port' respectively (note the 8080 convention is 'inst destination,source' whereas the MC68000 is 'inst source, destination'). The ERG68-696 I/O address space is normally 00FFC000...00FFFFFF, this particular address range was not chosen casually but rather with an eye to simplifying coding in advanced applications (hint absolute short addresses are sign extended prior to assertion), in other words while the address space used for this function can be changed by the user it is not recommended that he do so because (1) ERG monitors and utilities do not support such changes and (2) programming convenience. The ERG68-696 sOUT and sINP signals fully comply with the IEEE-696 specifications.

The sWO* line identifies the cycle as a processor output (write) cycle. The ERG68-696 makes provisions for user selection of sWO* as either (1) a copy of PWR* which conforms closely to the sWO* produced by most S100 CPU's or (2) a true status write stroke and is asserted much earlier in the cycle than PWR*, an advantage to well designed dynamic memory cards. The ERG68-696 sWO* line fully complies with the IEEE-696 specifications.

The sINTA line indicates that an interrupt has been acknowledged by the processor. The complexities of the sINTA line get a bit sticky, a little grinding may clear up some of the variations on this line, but one suspects more are in the works. The problem seems to result from that old pathological connection to the world of 8080, specifically the architecture of priority interrupt controllers such as the INTEL 8259, and others generally known as 'PIC's.' The IEEE-696 specification essentially states that interrupts ought to be acknowledged in some fashion unless it is a non-maskable interrupt (NMI*) in which case it does not need to be acknowledged, the specification assumes 'that if interrupt acknowledge cycles occur, an interrupt controller somewhere in the system will respond appropriately.' That all seems simple enough, except for a couple of details such as some PIC's requiring multiple sINTA cycles and others not needing any. A few concrete examples may save this discussion from going completely off the deep end. The INTEL 8259x PIC has two methods of responding to sINTA, one is the '8080' mode which puts a call instruction on the bus during the first sINTA then requires two more sINTA's to get the address vector of the interrupt service routine, the other form is the '8086' mode which just provides the address of one of the 8086 interrupt vectors (needs two sINTA's to do this, unfortunately it's the second sINTA which puts the vector on the bus) which in turn provides the interrupt service address vector. The requirement for multiple sINTA's can create an insurmountable problem in systems using non-Intel processors, note that there are 'PIC' boards for the S100 using the 8259x which generate the second (and third if in 8080 mode) sINTA's 'on-board' for such processors as the Z80 or for that matter the ERG68-696, be advised that this scheme is extremely hardware dependent and is not reliable in most cases. The AMD AM9519 can be programmed to respond to a single sINTA or as many as four sINTA's putting a byte of data on the bus for each sINTA from it's internal response register which is loaded by the user. The AM9519 should be able to accommodate just about any processor available including ERG68-696 as well as all those Intel creatures. The last (but certainly not least) is the Zilog family of devices (SIO,PIO,CTC) which have built-in interrupt processing capabilities and are almost impossible to use (the interrupt capabilities) on anything but Z80/Z8000 and perhaps the ERG68-696 (note the letter 'Z' as well as the numbers following are claimed by Zilog to be legal trademarks). Additionally the Z80 has three interrupt modes which further complicates the use of sINTA. The purpose of this sINTA discussion was to clarify the use of this line, the opposite may well have been achieved never-the-less it does become obvious that there is no simple way to assure that any combination of processor and interrupt controller which individually meet the requirements of the IEEE-696 specification will work together on the bus (systems houses take heart you are still needed). The ERG68-696 attempts to accommodate the widest possible range of protocols as long as a single sINTA will do the job. A better solution for users of the ERG68-696 is to use the seven level auto-vectored interrupt scheme which can accommodate up to 6 maskable and 1 non-maskable interrupt lines (VIO*...VI7*,NMI*,INT*), a very convenient user jumper area is provided. The ERG68-696 sINTA fully complies (albeit somewhat academic) with the IEEE-696 specification. The sHLTA indicates that the processor is in a halt state, ie. not processing. The sHLTA line on the ERG68-696 derives from the HALT* pin on the MC68000.

The ERG68-696 sHALT fully complies with the IEEE-696 specification.

The SXTRQ* line indicates that a 16 bit (word) operation is to be performed on the bus. The ERG68-696 (requires 16 bit program space, it will not run a program in 8 bit RAM, so this line is asserted during all accesses of either user or supervisory program space. The assertion of SXTRQ* requires SIXTN* to be asserted, if SIXTN* is not asserted within the correct time frame exception processing is invoked (ie. error trap). See SIXTN* under 'control bus' as well as the discussion of word/byte addressing under 'data bus.'

The ERG68-696 SXTRQ* fully complies with the IEEE-696 specification.

The last status bus line (MWRT) is not included in the list of status lines in paragraph 2.2.3 of the IEEE-696 specification but is mentioned elsewhere and is a status line. The MWRT line indicates a memory write cycle is in progress. MWRT has historically been generated by the S100 'front panel' since the MWRT signal is generated by combining two existing bus signals and is 'not directly available' on the bus, it should be mentioned that most modern S100 based CPU's optionally generated this signal for use in systems without 'front panels.' The signal is defined to be MWRT=pWR*-sOUT thus MWRT is HI when pWR* is LOW and sOUT is LOW. The ERG68-696 produces MWRT which can optionally (jumper) be removed from the bus.

The ERG68-696 MWRT fully complies with the IEEE-696 specification.

DATA BUS

The data bus consists of 16 lines which form the primary information path for all system elements. The data direction is defined by the processor controlling the bus, that is data out is from the processor to the bus and data in is from the bus to the processor.

The IEEE-696 data bus includes the original S100 data-in (to the processor) and data-out (from the processor) unidirectional 8 bit buses but allows these buses to be combined into a single bidirectional 16 bit bus. The 16 bit bus allows efficient use of 16 bit processors on the bus (all other considerations ignored), however it may come as no surprise at this point that there are some complications. The retention of the original S100 data-in and data-out buses in order to be (potentially) upward compatible wit older S100 products is the crux of the problem. The multitudes wait with bated breath for the discovery of the S100 rosset stone which will undoubtedly reveal why the ancients chose to separate a nice bidirectional data bus (8080) into a pair of unidirectional buses which in most cases must be reconstituted to their original (bidirectional) form at their destination, additionally we look forward to a better understanding of the choice of positive logic levels for some of the signals. Oh well perhaps such knowledge would be dangerous. In any case, the effect of retaining the original data-in, data-out buses is to complicate 8 bit operations on all devices capable of 16 bit transactions. The reason is (that 16 bit devices deal with 8 bit objects (generally speaking) as onehalf of a 16 bit object, which seems simple enough. What isn't simple is agreeing (as in standardizing) on which half is being dealt with in a given instance, the terms commonly used to identify these 8 bit objects are 'high order' and 'low order' or 'even' and 'odd.' The IEEE-696 specification, true to it's 8080 roots, originally defined the data-out bus as the 'high-byte' (A0=0) and the data-in bus as the 'low-byte' the 4-2-81 addendum to the IEEE-696 spec. clarifies this very nicely. Specifically the old DI (data-in) bus which was originally the 'HIGH' byte is defined as the 'ODD' byte (A0=1) and the old DO (data-out) which was originally the 'LOW' byte is defined as the 'EVEN' byte (A0=0). The new definition improves the processor independance of the bus, unfortunately some boards are currently being marketed which conform to the old definition which means that they cannot be used on the same bus with boards designed to the new definition. The ERG68-696 has user programmable jumpers which can accommodate virtually any data path for both the internal bus (MC68000) and the external bus IEEE-696(S100). The options are; (1) A0 or inverted A0,(2) IEEE-696 DE (data even) to MC68000 Least significant byte or most significant byte,(3) IEEE-696 least significant byte at ODD address or least significant byte at EVEN address (these are shown as LO which is the default and LE in the IEEE-696 compliance level). All cards on the bus at any given time must of course be mutually compatible (ie. same LE/LO discipline). The data path control scheme is sufficiently complex to justify a picture or two (see fig 4 of the IEEE-696 spec S100 MICROSYSTEMS vol 1/no.1 and S100 MICROSYSTEMS vol 1/no.4 pg. 21).

The ERG68-696 data bus protocols fully conform to the IEEE696 specification in either the 'LO' or 'LE' format.

CONTROL OUTPUT BUS

The control output bus consists of 5 signals which provide timing information and control data movement on the bus. The 5 signals are:

1) pSYNC	start of bus cycle.
2) pSTVAL*	signals that status bus is valid.
3) pDBIN	processor is requesting data from the bus.
4) pWR*	processor has data for the bus.
5) pHLDA	signals that the previous processor has given up the bus.

The pSYNC signal (which gets its name from the SYNC line on the 8080 clock generator) is perhaps the most critical of the bus timing signals in that it is the first indication to the slaves that a bus master is alive and is about to do something. In other words the bus master is starting a bus cycle so everybody should pay attention. The pSYNC signal has the dubious distinction of being the least consistent (critical) signal among the various CPU cards for the S100 bus. A well designed CPU card should generate a pSYNC with a duration of one system clock cycle (give or take a few nano's) well centered on the low period of the system clock (see figure 11 of the proposed spec vol 1/ no.1 of S100 MICROSYSTEMS). The rising edge of pSYNC thus generated will be in the middle of the first half of a new bus cycle. A number of S100 boards use an inverted copy of pSYNC to generate a wait state, this is a highly questionable practice and usually indicates a marginal design. One reliable method of generating a wait state(s), which will meet the bus timing requirements, is to use pSYNC as the data input to a latch and clock the wait state through using the system clock.

The ERG68-696 pSYNC conforms precisely to the IEEE-696 specification.

The pSTVAL* (processor status valid) signal is asserted (falling edge) when the address bus and status bus are valid. This signal is crucial to IEEE-696 products (especially 'intelligent' cards). The pSTVAL* signal suffers from the same inconsistencies as pSYNC in some current products purporting to meet the IEEE-696 standard. The pSTVAL* signal is due to the IEEE-696 specification and replaces the old (8080) phase 1 clock, while the 8080 phase 1 clock low during pSYNC approximates the pSTVAL* signal any new designs based on the IEEE-696 spec which use this signal will expect only one pSTVAL* in any given bus cycle. One can be almost certain that old S100 cards which used the phase 1 clock and new CPU cards which generate a (phase 1 clock instead of pSTVAL* will not work properly (if at all) with cards designed to the IEEE-696 standard (with all due respect to the footnote in said standard). The ERG68-696 generates pSTVAL* for approximately one half system clock period centered on pSYNC, notably the ERG68-696 address bus and status bus are always valid at the rising edge of pSYNC which guarantees adequate timing margins.

The ERG68-696 pSTVAL* fully complies with the IEEE-696 specification.

The pDBIN signal indicates that the addressed slave should put data on the bus for consumption by the bus master. The name came from the 8080 (DBIN) and so did the timing, the name is ok but the timing may leave a bit to be desired for the new generation of processors. The IEEE-696 (para 2.7.4.2) says that pDBIN should be asserted after a minimum specified time from the assertion of pSTVAL*, the question here is whether the processor (eg. MC68000) should arbitrarily wait to signal a read from the bus even though the information is available much earlier, say prior to PSYNC. The ERG68-696 meets the specification but the user may at his option change a jumper which will allow pDBIN to be generated at the earliest possible time consistant with the rest of the bus timing. The restriction on pDBIN does not seem appropriate, it is reasonable to assume that this is an 8080 left-over.

The ERG68-696 pDBIN fully complies with and optionally excels the IEEE-696 specification.

The pWR* signals that the bus master has placed data on the bus for consumption by the addressed slave. The pWR* signal derives from the 8080 WR* and the IEEE-696 timing constraints are consistant with the 8080 timing. The situation for pWR* is similar to pDBIN except the pWR* signal must wait until after pSYNC is complete, again the ERG68-696 meets this requirement but provides for the user to allow for pWR* to be asserted as soon as possible.

The ERG68-696 pWR* fully complies with and optionally excels the IEEE-696 specification.

The pHLDA signal acknowledges that the previously asserted HOLD* signal has initiated a transfer of the bus to a temporary master. What this means is that things are well underway for a TMA (Temporary Master Access), which in the old days was called DMA (Direct Memory Access). But alas, the world is more complex now and the direct access could well be to another processsor or I/O or .... The pHLDA signal (8080 HLDA) which on the MC68000 is derived from the bus grant (BG*) line, is discussed further in the TMA control bus section. The ERG68-696 pHLDA fully complies with the IEEE-696 specification.

CONTROL INPUT BUS

The control input bus consists of 6 signals which are used to synchronize master and slaves on the bus. The signals are:

1) RDY	causes wait states when LOW.
2) XRDY	causes wait states when LOW.
3) INT*	interrupt request.
4) NMI*	non-maskable interrupt request.
6) SIXTN*	acknowledges a 16 bit request.

RDY and XRDY signals are identical in function, when either is low the bus master is placed in a wait state. RDY is the primary ready signal for bus slaves, XRDY is used only by front panel devices or special circuits such as single step. The XRDY line must not be driven by more than one device on the bus whereas the RDY line can be driven by many (one at a time of course). The XRDY line serves no useful purpose but if either it or RDY go low the ERG68-696 will begin executing wait states. Remember that the MC68000 is an asynchronous processor and therefore RDY and XRDY simply hold-off the data acknowledge signal which indicates that the bus is ready to complete the cycle.

The ERG68-696 RDY and XRDY fully comply with the IEEE-696 specification.

The INT* signal indicates that a slave requires service from the bus master, the 4-2-81 addendum to the IEEE-696 specification has significantly changed the function of this signal in what appears to be an undesirable way not to mention highly hardware dependent, so here comes the complaint. First note that the ERG68-696 meets the old and new definitions of this line so put away your clubs for beating self-serving designers. The specification (para 2.9.1.1) originally stated 'the interrupt controller is NOT required to use INT*. A vectored interrupt may occur without INT* EVER being asserted.' The addendum states 'The ... IS required to use INT*. A vectored interrupt SHALL NOT occur without INT* being asserted.' The addendum negates the original and does so at the expense of versatiltiy not to mention existing cards which might otherwise be ok. The question is, if the processor is capable of servicing a vectored interrupt without INT* being asserted why require it? A request is being made to the IEEE-696 committee to reconsider this requirement and perhaps even provide a compatibility matrix and compliance level indicator to preserve the purpose of the standard. The ERG68-696 has a jumper which when installed requires INT* before an interrupt request will be acknowledged. The original purpose of the option was to allow the user to use either a general interrupt scheme where the vector is placed on the bus by the interrupting slave or the simpler auto-vector scheme. The MC68000 is capable of servicing both forms of interrupt request.

The ERG68-696 INT* fully complies with both the original and negated versions of the IEEE-696 specification.

The NMI* signal is a special type of interrupt which can not be overriden by software instructions to the processor. NMI* is normally the highest priority interrupt available to the processor and is frequently driven by power fail circuitry which detects fatal hardware errors (see special condition lines PWRFAIL* and ERROR*). The ERG68-696 fully implements the NMI* signal, and jumper options are available to the user for a variety of configurations.

The ERG68-696 NMI* fully complies with the IEEE-696 specification.

The HOLD* signal is used by temporary bus masters to request control of the bus from the current bus master. HOLD* initiates a TMA operation on the bus and is discussed further under TMA control bus. HOLD* is copied to the bus request (BR*) line on the MC68000.

The ERG68-696 HOLD* fully complies with the IEEE-696 specification.

SIXTN* signal is asserted by slaves capable of performing 16 bit data transfers in response to the assertion of SXTRQ* by the [illegible]s master. The SIXTN* signal must meet the same timing requirements as RDY and XRDY in order to be properly recognized by the bus master. The ERG68-696 issues a bus error if SIXTN* is not asserted in response to assertion of the SXTRQ* signal. The ERG68-696 executes instructions only on 16 bit program space, while data memory and other devices on the bus may be accessed as 8 bit space. The ERG68-696 does not conduct 16 or 32 bit transfers in byte serial fashion, but rather generates an error trap if such a transfer is attempted.

The ERG68-696 SIXTN* complies with the IEEE-696 specification.

TMA CONTROL BUS

The TMA (Temporary Master Access) consists of 8 open collector lines which perform the arbitration and transfer between temporary and permanent bus masters in a consistent orderly manner. The HOLD* and PHLDA signals play an important ro in this process and will be considered part of the TMA control logic for purposes of this discussion. The TMA control bus lin are:

1) DAM0*	through DMA3* used to establish the priority of the Temporary Master (TM).
2) ADSB*	disables Permanent Master (PM) Address Bus.
3) DDSB*	disables PM Data Out Bus.
4) SDSB*	Disables PM Status Bus.
5) CDSB*	Disables PM Control Bus.

The transfer of the bus between master is probably the most hazardous event which the IEEE-696 bus has to contend with. The importance of compatibility in bus transfer protocols of all S100 products claiming to meet the standard, as temporary or permanent bus master, cannot be over emphasized. Designers of TM or PM must rigorously adhere to the provisions of the IEEE-696 specificaion. The requirements are well stated and nonambiguous. How does this transfer stuff take place and what's all this arbitration about? It all begins when a TM decides it needs the bus in order to transact its business. The TM issues an 'IWANT' signal to the arbitration logic which starts the fracas. The arbitration logic then checks HOLD* and pHLDA, if neither of these is asserted, the arbitration logic issues the 'MINE' signal and asserts HOLD*. The bus request is made without a contest. If on the other hand, pHLDA or HOLD* are already asserted the TM must wait for pHLDA to be negated prior to taking further action. When pHLDA is negated, the battle for the bus is on. The bus allows for up to 16 TM's and each one wanting the bus must begin the process of checking the priority of all other requestors until it has priority (or gives up). Assuming that the subject TM finally gets its turn the arbitration logic issues 'MINE' and asserts HOLD*. Note that even if a higher priority TM wants the bus, it must wait until the current TM is done and pHLDA is negated before contending for control. Finally by hook or crook, a bus request (HOLD*) has been issued and the transfer process begins. The bus transfer sequence starts with Transfer State One TSI), which results in disabling the PM's address (ADSB*) status (SDSB*) and (DDSB*) buses. The control bus is still alive and for good reason, the control bus signals can wreak havoc if left unattended for an instant. The control bus signals must be in their most friendly state at this time (see IEEE-696 para 2.8.2.3.4). The control bus is driven at these levels for a minimum of one half system clock cycle prior to disabling the PM control bus (CDSB*) . Once CDSB* is asserted the TM can have it's way with the bus. When the TM is done, it negates HOLD* and the transfer process is rerun in reverse, this is called transfer state two (TSI Again, both the TM and PM drive the control bus for at least ½ system clock time. CDSB* is then negated, allowing the PM t control of the control bus. Next ADSB*,SDSB* and DDSB* are negated, then pHLDA is negated. The bus arbitration and transfer process may now begin again. The IEEE-696 folks have provided a recommended design for the arbitration logic. Designers may want to quibble with the circuit and thats ok, but for fear of your life, don't get creative. The ERG68-696 is a PM and thus DMA0* thru DMA3* are simply provided with pullups. The ERG68-696 fully implements the transfer logic of the IEEE-696 specification, the logic may be disabled by the user without affecting the ability of a TM with its own logic to control the transfer (ie. the ERG68-696 has both drivers (OC) and receivers (74LS14) for ADSB*,DDSB*,SDSB* and CDSB*, pullups are provided on each of these lines).

The ERG68-696 TMA control bus fully complies with the IEEE-696 specifications, with a compliance of PM.

VECTORED INTERRUPT BUS

The VECTORED INTERRUPT BUS consists of 8 lines which may be used to provide prioritized interrupt requests to some form of interrupt controller. The lines are designated VIO (default highest priority) through V17 (default lowest priority). The VI lines are by convention (and IEEE-696 recommendation) 'level' sensitive as opposed to 'edge' sensitive. Level sensitive simply means that a request is asserted until it is serviced or the requestor gives up. Additionally, level sensitive implies that the request will be serviced repeatedly until removed, ie. if servicing the request does not result in removal of the request the controller sees it as another request and may initiate service again. The consequence of the 'repeat service while request asserted' is that logic must be included in the interrupt scheme which negates the request once service is received (eg. sINTA clears request). The IEEE-696 specification allows the interrupt controller to mask, fence out, or rotate priority of requests from the interrupting slave. The 4-2-81 addendum to the IEEE-696 requires the INT* be asserted before a vectored interrupt request can be serviced, see comments on this under discussion of INT*. The ERG68-696 is designed to accommodate the full range of interrupt capabilities of the MC68000 including the very powerful and convenient auto-vector mode.

The ERG68-696 VECTORED INTERRUPT BUS complies with the IEEE696 specification.

The remaining bus lines are pretty vanilla, although essential. The ERG68-696 meets the IEEE-696 specification for all remaining bus lines.

A final note. The ERG design team has been using the MC68000, via Motorola development products for about a year and a ha and via the ERG68-696 prototypes for the last couple of months, based on this considerable experience one has to say 't MC68000 is DYNAMITE !.' No compromise has been made in the design and production of the ERG68-696 CPU card. The goal was and is to provide a reliable vehicle, which allows the user to exploit the full power of the MC68000.