APPLICATION NOTE

ABSTRACT
This procedure is suggested to verify the signal, bus, power connections and timing between the host (control processor), UART and printed circuit board. The following procedures may be executed and the results evaluated without recourse to any exotic test equipment (logic or protocol analyzers, oscilloscopes, etc). The assumption on which this is based requires that the processor must be able to read data from and write data to the UART and have some means of presenting those results to the human operator. If one can not be absolutely certain that simple reads and writes are properly executed then any other means of evaluating the UART connections will be suspicious.

AN462
Hardware and software verification procedure

author: *Peter Narvaez*

1998 Oct 07

Philips Semiconductors

PHILIPS

Hardware and software verification procedure AN462

Summary: This procedure is suggested to verify the signal, bus, power connections and timing between the host (control processor), UART and printed circuit board. The following procedures may be executed and the results evaluated without recourse to any exotic test equipment (logic or protocol analyzers, oscilloscopes, etc). The assumption on which this is based requires that the processor must be able to read data from and write data to the UART and have some means of presenting those results to the human operator. If one can not be absolutely certain that simple reads and writes are properly executed then any other means of evaluating the UART connections will be suspicious.

General procedure: To perform writes and reads of the several registers where the only " clock " involved is the chip select, read and write pins. Secondly to observe the results of writes to several of the control registers and observe the results by reading the status registers. The several procedures suggested below verify bus data flow and use the "Local Loop back" mode to verify the receiver and transmitter operation. The local loop back mode (where in all data transmission and reception occurs within the UART) will be used to setup processor interrupt or polling conditions. Successful completion of these procedures will show correct operation of all internal registers, bus interface, clock generation, counter timer and oscillator. The only thing not verified is the oscillator frequency and the connection of the TxD and RxD to external ports and general purpose input and output pins.

General Comment: When in the hardware or software verification mode it is very advantageous to read the status register often. This is also true when testing for some "random" or "infrequent" fault that appears after the hardware and software have been "verified". The read of the status may be performed before and after every access to the device! The content of this register will indicate when and where unexpected conditions occurred. This will be indicative of conditions internal to the UART as well as external connection, timing, and software. An example would be after the hardware reset is issued a read of the status shows that the transmitter empty bit set. THIS MEANS THE TRANSMITTER IS ENABLED! (One would expect to see 0x00, which MUST be returned immediately after hardware reset.) However it is known that software DID NOT enable the transmitter. Therefore one would reason "some very special" kind of noise, very slow reset fall time, etc. must have looked like a transmitter enable command. If this situation did exist it would be pointless to go farther in the verification procedures until correcting this fault. Another example: After a reset and several accesses to the UART and before the receiver is enabled one notes that the some of the receiver data status bits (a parity error for example) is/are set. This could mean that the receiver is enabled (although an overt software receiver enable command was not issued) and has received something. It could also mean that timing violations (contention on the address bus maybe) moved the receiver FIFO read pointer such that the status of random power up conditions of the FIFO are reported. Often it is found that different parts of the software have control of the UART and they are activated independently of each other. There are many more conditions!

Procedures	Expected Results and Comments
1. Do a hardware reset. (In systems where hardware reset is not used, temporarily enable it by disconnecting the reset pin such that a manual reset may be performed).	Hardware reset is not required for proper UART action. For these procedures, however, it is highly recommend even if it is artificially implemented. Software resetting is always available. However knowing that software reset worked implies, at least to some extent, that these procedures are not required. It is difficult to understand how one would verify the efficacy of the software reset without going through a good part of what is described below.
2. Do two writes of different data (0xAA, 0x55) to each of the sets of MR register(s). Do not worry about MR pointers—just do the writes!	This will always result in the second byte being written to MR2—the x'55 in this case. This will show (in step 4 below) that you can at least read and write to the device and return data to the controlling system. No other exotic measuring devices are required—only the system itself. I believe this test will work even without V_{SS} and V_{DD} applied although logical low levels would be near +0.7 volt.
3. Read the status register of each UART before and after the accesses to the MR registers. They must always return x'00.	The read of the status must return x'00. If not x'00 then the write to the MR register was corrupted to do some other action. The condition of the status register indicates what that may be.
4. Perform a single read of the MR register. Again, don't worry about the MR pointer. Just do the read! The read will return the second byte written to the MR register; A x'55 would be returned for the data above. Caution: This is a very important step. It shows the validity of fundamental control. However it is so simple from the internal logic of the chip that it will most likely work even if V_{SS} and V_{DD} are not connected!	After the previous two writes (in step 2) the MR pointer must be at MR2. Then a read of MR will return the value of the second byte written to the MR in step 2. This step indicates that at least some of the address lines are connected and operating correctly and that the CEN, RDN, and WRN signals are correct. It also shows that the data bus is operating but it does not show that the data bus is correctly wired. Most CMOS devices of nominal design can be powered through input pins assuming at least one is at V_{CC} and another is at V_{SS}. The internal logic will then be supplied with approximately 3.6 volts (a diode drop above Vss and a diode drop below V_{CC}). They will most likely not operate a specified speed however. This is a good way (V_{CC} and V_{SS} pins not connected) to generate the famous CMOS latch up which can destroy CMOS devices. Modern design of CMOS circuits considers this potential for latch up and consequently circuit design and process procedures have been improved to discourage the latch up "feature." However the "old timers" will say that if you try hard enough one can latch up any CMOS integrated circuit. I haven't tried.
5. Write 0x10 to the command register (CR) at chip address 0x2. Repeat the writes to the MR registers of step 2. Set the MR pointer back to MR1 and do two reads of the MR register. Do this for all sets of the MR registers. The two bytes previously written MUST be returned. Note: It would be convenient, for step 8, to leave the MR1 and MR2 set to 00 and 87 respectively. However for the purpose of this step all MR registers should be written with different data.	This shows that one more address line is operating, one data line is correct and that the oscillators operating and the UART is not in he power down mode and V_{SS} and V_{DD} are connected. Through this entire sequence a read of the status register must return x'00.
6. Write to the command register again and enable the transmitter. Write x'15 to CR of each UART	Enabling the transmitters will cause the transmitter status bits in the status register (SR) to set immediately. It also sets the MR pointer to MR1
7. Read the status register. You must see the transmitter ready and empty bits set in the status register. The status register read should return a 0x0C. Now to regress a bit: If the MR1 and MR2 values are at 00 and 87 respectively the receivers and transmitters will be put into the "local loop back" mode. Set them to those values now. (The MR pointer was set to MR1 in previous step.)	
8. Write to the clock select registers and select a clock, say code BB – 9600 baud, for each receiver and transmitter.	

Procedures	Expected Results and Comments
9. Write to the command register and enable the receivers and transmitters. (0x05) ??? already done step 8 ??	
10. Read status. It should still be 0C.	
11. Write a byte to the transmitter. Now several events will occur which will be shown through a read of the status register(s)	
12. Loop on a read of the status registers. The important events to read are the status conditions of 0x04 and 0x0D.	The status register will report read this sequence of values starting with 0x0C then 0x00, 0x04, 0x0C, 0x0D. Due to the asynchronous nature of the read timing with respect to the 9600 baud clock you may not see all or the above values; some exist only for 1/16 bit time and some for only 270 ns. However you will certainly see the 0x04 and the 0x0D.
a. Immediately after the write to the transmitter the status will return a x'00.	This shows the transmitter has accepted the byte, is loading it to the shift register and is starting to send the start bit.
b. Status now changes to x'04 showing the start bit is completed and the TxFIFO is ready for another byte	b) The time for the x'04 to occur is a variable that depends on the relative positions of the Tx 16x clock, the Tx 1x clock and the bus cycle. It may be from 1/16 to 17/16-bit time.
c. Next the status will read 0x05. This state will exist for less than 7/16-bit time. This is usually not seen by the bus cycle.	c) It shows receiver having received the byte (The receiver loads the byte to the Rx FIFO shortly after the center of the stop bit time.) and the transmitter has not yet finished the stop bit.
d. The status will now go to 0x0D.	d) The status shows the receiver has loaded the byte to the RxFIFO and the transmitter is empty.
13. Read the receiver FIFO.	You must read the byte loaded to the transmitter in step 8.
14. Read the status. It should show 0x0C.	It should show 0C meaning that the receiver FIFO is now empty; transmitter is empty and FIFO still ready to be loaded.
15. This finishes the basic verification of the bus interface and expected values from the status register.	All of the above MUST function as described. Any unexpected results will be caused by timing violations, incorrect wiring, or the oscillator not starting. Of course a damaged chip must also be considered. FAILURES OF THE ABOVE PROCEDURE WILL BE CAUSED BY "FIRST ORDER " EFFECTS!
If the MR2 register is now set to 03 you should be able to connect the transmitter output pin to the receiver input pin and repeat the steps 8 to 10. This shows the transmitter and receiver can converse through an "external" loop back.	In most cases this may not be done easily since board connections to the UART are in place. However if it can be done such that only a "wire type" connection exists between the transmitter and the receiver then it proves that the transmitter output port and the receiver input port have not been damaged in the fracas of getting the first prototype board built. **The antithesis of this procedure is attempting to verify the function of the UART by "having it talk to" some other UART or a dumb terminal. This implies that you already can program he UART for the "other device" and that the "other device" is correctly programmed and all the intervening hardware is operating! It is suggested that this is a rather large assumption; that an error in this assumption has extremely high probability of leading one to a debug path that has no relation to the real problem!** At this point connection to real external transmitters and receivers is meaningful. Any malfunction can now be properly partitioned to the areas of external hardware, differences between the setups of the communications devices and misunderstandings of the system protocols. System protocols include baud rates, character lengths, RTS/CTS handshake, simplex, or duplex etc.

Procedures	Expected Results and Comments
PLEASE NOTE THAT ALL OF THIS IS DONE WITH THE SYSTEM ITSELF! No external logic analyzers, transmitters, receivers, etc. are required. The only three entities are used: the UART, its interface to the control device and the control/display device itself. Now the problem will be to determine that the UART can communicate with external devices. For example, in this list the ACR (Auxiliary Control Register) was not used nor was MR0 programmed. This could cause the baud rate to be other than 9600 but the transmitters and receivers would still see the same baud rate and operate together. While in the local loop back mode all UART to processor controls and interrupt configurations may be verified. This seems like a good thing to do since it eliminates any external influences from clouding the analysis of what the UART is doing and how the host interprets its actions.	

Preliminary code for setting up the C100 devices for basic 9600-baud receive and transmit Initialization for the 28L198, 26C198, 26C194 and 28L194.

For this code, use “Local Loop back” or connect transmitter A to Receiver A. If this is not done then the status bits and interrupt associated with the receiver will not go active. See steps 19 to 23.

Step	Action	Reg	Addr	Data	Comments
1	WR	GCCR	F8	04	Async bus cycle, Interrupt vector modified to show channel code in lower 3 bits of vector
2	WR	CRa	81	F8	Reset chip
3	WR	MR0a	00	C0	No flow control, no Xon/Xoff, Tx Int on FIFO empty
4	WR	MR1a	01	13	No RTS/CTS, ISR unmasked, character mode, no parity, 8 data bits
5	WR	MR2a	10	00	Normal mode, Receiver interrupt on ready, 1 stop bit
6	WR	Rx CSR	0C	1E	9600 baud
7	WR	Tx CSR	0E	1E	9600 Baud
8	WR	CRa	81	28	Reset Break Change, Disable Tx & Rx (Redundant at this point but good restart w/o power cycle
9	RD	SRa	81	00	Should read a status of 00—ANYTHING ELSE SHOWS A FUNDAMENTAL PROBLEM!
10	RD	ISRa	82	00	“
11	WR	IMRa	82	03	Enable Tx and Rx interrupts for channel a
12	WR	ICR	1B	00	set interrupt threshold at 00
13	WR	IVR	1F	00	Interrupt vector set to 00. (See step 1—lower three bits modified)
14	WR	CRa	81	03	Enable Rx, Tx. This will generate an immediate interrupt. This interrupt will show transmitter ”A” interrupting. You may see this by reading the SR 9 Status Register) or by letting the interrupt vector steer the service routine.
15	RD	SRa	81	0C	transmitter ready & empty
16	WR	TXa	83	nun	Write any data you wish. Tx interrupt will go off
17	WR	TXa	83	tt	write any value you wish
18	RD	SRa	81	04	04—transmitter not empty and ready for more data. Interrupt is off because Tx fifo is not empty
—					After a character time (10.4ms) the receiver interrupt is active. Decide on service type per step 14
19	RD	SRa	81	05	Receiver has one byte and Tx is still busy
—					Wait one more character time
20	RD	SRa	81	0B	Tx ready and empty, Receiver FIFO ready
21	RD	RXa	83	nn	Read First data byte sent by transmitter
22	RD	RXa	83	tt	second byte sent by the transmitter
23	RD	SRa	81	0C	Receiver is empty; transmitter is empty, and ready. See step 15.

The programming below is a basic setup for all SCCxxx, SCxxx, and SCNxxx UART devices. Setup is for one start bit, 8 data bits, no parity and one stop bit. Recall that for the devices under consideration the access to the MR registers is by the MR pointer. TxA may be externally connected to RxA or Local Loop back may be used.

Step	Action	Reg	Addr	Data	Comments
1					Execute hardware reset.
2					
3	WR	MR0a	0	00	No flow control, no Xon/Xoff, Tx Int on FIFO empty (only SCxxx devices have MR0)
4	WR	MR1a	0	13	No RTS/CTS, character mode, no parity, 8 data bits
5	WR	MR2a	0	07	Normal mode, Receiver interrupts on ready, 1 stop bit. (External Tx to Rx connection is used)
6	WR	CSRa	1	BB	9600 baud Rx, 9600 baud Tx
7	WR	CRa	2	28	Reset error bits, disable Rx and Tx (Redundant at this point but good restart w/o power cycle)
8	WR	ACR	4	00	Baud rate normal, C/T to counter mode, External C/T clock, disable change of state.
9	WR	IMR	5	03	Enable interrupts for RxA and TxA
10	RD	SRa	1	00	Status register should read 0x00—ANYTHING ELSE SHOWS A FUNDAMENTAL PROBLEM!
11	WR	CRa	2	15	Reset MR pointer to MR1, Enable Rx, and Tx. Interrupt will occur immediately!
12	RD	SRa	2	0C	Shows transmitter is empty and ready
13	WR	TXa	3	55	Load first byte to transmitter. Interrupt sets inactive for approximately 1 to 2 bit times.
14	WR	TxA	3	AA	Load next byte to TXa. Interrupt will extinguish for about one character time.
15	RD	ISR	5	00	Loop on this read operation or wait for the interrupt. The next interrupt will be the transmitter followed by a receiver interrupt
16	RD	SRa	1	0D	Transmitter is empty and ready again, receiver has one byte in its FIFO.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

Date of release: 10-98

Document order number: 9397 750 04631