DALLAS SEMICONDUCTOR

Application Note 74
Reading and Writing iButtons via Serial Interfaces

I. INTRODUCTION

An iButton™ is a chip housed in a stainless steel enclosure. The electrical interface is reduced to the absolute minimum, i.e., a single data line plus a ground reference. The energy needed for operation is either "stolen" from the data line ("parasitic power") or is taken from an embedded lithium cell. The logical functions range from a simple serial number to password–protected memory, to 64K bits and beyond of nonvolatile RAM or EPROM, to a Temperature iButton, to a real time clock plus 4K bits of nonvolatile RAM. Common to all iButtons is a globally unique registration number, the serial 1–Wire™ protocol, presence detect, and communication in discrete time slots. Table 1 gives an overview of the available devices.

For read operations all devices are satisfied with a 5kΩ pull–up resistor to supply energy and to terminate the 1–Wire bus. iButton devices based on non–volatile RAM (DS1991 to DS1996) can also be written using this same interface. Due to their different technology, EPROM based iButtons (DS1982 to DS1986) also require pulses of up to 12V for programming. Since they cannot be erased, EPROM iButtons are referred to as Add–Only Memories. Another device, the DS1920 Temperature iButton, gets its energy for temperature conversion through a low impedance active pull–up to 5V. Different requirements for writing or special functions are the reason for several types of interfaces.

iButton DEVICES Table 1

Device Type	Family Code	Serial Number	Memory Bits Type	Protected NV RAM bits	Real Time Clock	Interval Timer	Cycle Counter
DS1990A	01H	yes	—	—	—	—	—
DS1991	02H	yes	512, NVRAM	3 * 384	—	—	—
DS1992	08H	yes	1K, NVRAM	—	—	—	—
DS1993	06H	yes	4K, NVRAM	—	—	—	—
DS1994	04H	yes	4K, NVRAM	—	yes	yes	yes
DS1995	0AH	yes	16K, NVRAM	—	—	—	—
DS1996	0CH	yes	64K, NVRAM	—	—	—	—
DS1982	09H	yes	1K, EPROM	—	—	—	—
DS1985	0BH	yes	16K, EPROM	—	—	—	—
DS1986	0FH	yes	64K, EPROM	—	—	—	—
DS1920	10H	yes	16, EEPROM	TEMPERATURE iButton			

II. 1–WIRE INTERFACE

A. General Information

iButtons are self–timed silicon devices. The timing logic provides a means of measuring and generating digital pulses of various widths. Data transfers are bit–sequential and half–duplex. Data can be interpreted as commands (according to the prearranged format identified by the family code) that are compared to information already stored in the iButton to make a decision, or can simply be stored in the iButton for later retrieval. iButtons are considered slaves, while the host reader/writer is considered a master.

B. DC Requirements

iButtons operate in an open drain environment on voltage levels ranging from 2.8V (minimum pull–up voltage) to 6V (maximum pull–up voltage). All voltages greater than 2.2V are interpreted as logic 1 or HIGH, voltages less than 0.8V are considered as logic 0 or LOW. The pull–up voltage must be a minimum of 2.8V to recharge an internal storage capacitor that is used to supply power during periods when the data line is low. The size of this capacitor is about 800 pF. This capacity is seen for a short time when an iButton is contacted by a probe. After the capacitor is charged, only a very small fraction of this capacity is recognizable, according to the charge required to refill to full charge. The total time constant to charge the capacitor is defined by the capacitor itself, the internal resistances of about 1 kΩ, the resistance of the cable and contacts, the cable capacitance, and the resistor pulling up the data line.

C. AC Requirements

Timing relationships in iButtons are defined with respect to time slots. Because the falling slope is the least sensitive to capacitive loading in an open drain environment, iButtons use this edge to synchronize their internal timing circuitry. By definition the active part of a 1–Wire time slot (t_{SLOT}) is 60 µs. After the active part of the time slot, the data line needs to be inactive for a minimum of 1 µs at a voltage of 2.8V or higher to recharge the internal capacitor.

Under nominal conditions, an iButton will sample the line 30 µs after the falling edge of the start condition. The internal time base of iButton may deviate from its nominal value. The allowed tolerance band ranges from 15 µs to 60 µs. This means that the actual slave sampling may occur anywhere between 15 and 60 µs after the start condition, which is a ratio of 1 to 4. During this time frame the voltage on the data line must stay below V_{ILMAX} or above V_{IHMIN}.

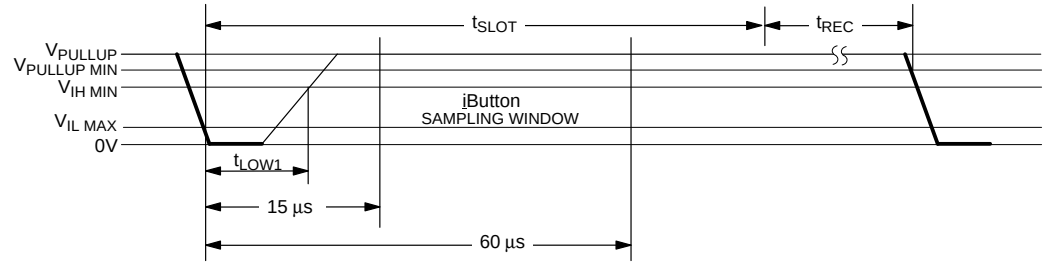
C.1. Write Time Slots

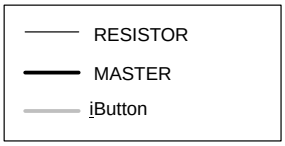
In the 1–Wire system, the logical values of 1 and 0 are represented by certain voltage levels in special waveforms. The waveforms needed to write commands or data to iButtons are called write–1 and write–0 time slots. The duration of a low pulse to write a 1 (t_{LOW1}, Figure 1) must be shorter than 15 µs. To write a 0, the duration of the low pulse (t_{LOW0}, Figure 2) must be at least 60 µs to cope with worst–case conditions.

The duration of the active part of a time slot can be extended beyond 60 µs. The maximum extension is limited by the fact that a low pulse of a duration of at least eight active time slots (480 µs) is defined as a Reset Pulse. Allowing the same worst–case tolerance ratio, a low pulse of 120 µs might be sufficient for a reset. This limits the extension of the active part of a time slot to a maximum of 120 µs to prevent misinterpretation with reset.

At the end of the active part of each time slot, iButton needs a recovery time t_{REC} of a minimum of 1 µs to prepare for the next bit. This recovery time is the inactive part of a time slot, since it must be added to the duration of the active part to obtain the time it takes to transfer one bit. The wide tolerance of the time slots and the non–critical recovery time allow even slow microprocessors to meet the timing requirements for 1–Wire communication easily.

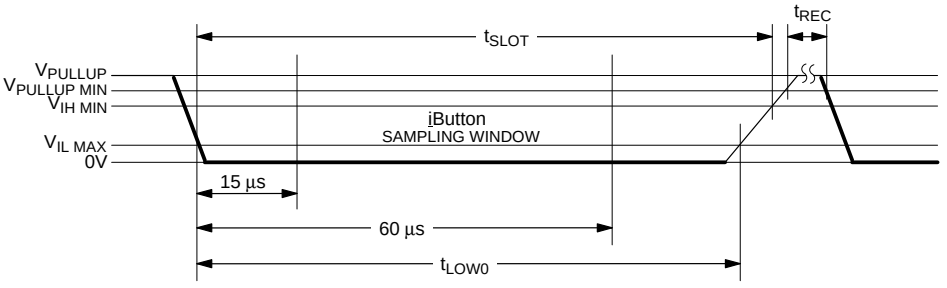
WRITE–ONE TIME SLOT Figure 1

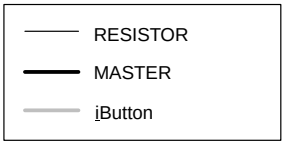




60 µs ≤ t_{SLOT} < 120 µs
1 µs ≤ t_{LOW1} < 15 µs
1 µs ≤ t_{REC} < ∞

WRITE–ZERO TIME SLOT Figure 2





60 µs ≤ t_{LOW0} < t_{SLOT} < 120 µs
1 µs ≤ t_{REC} < ∞

C.2. Read Time Slots

Commands and data are sent to iButtons by combining Write–Zero and Write–One time slots. To read data, the master has to generate Read–Data time slots to define the start condition of each bit. The Read–Data time slot looks essentially the same as the Write–One time slot from the master's point of view. Starting at the high–to–low transition, the iButton sends one bit of its addressed contents. If the data bit is a 1, the iButton leaves the pulse unchanged. If the data bit is a 0, the iButon will pull the data line low for t_{RDV} or 15 µs (Figure 3). In this time frame data is valid for reading by the master. The duration t_{LOWR} of the low pulse sent by the master should be a minimum of 1 µs with a maximum value as short as possible to maximize the master sampling window. In order to compensate for the cable capacitance of the 1–Wire line the master should sample as close to 15 µs after the synchronization edge as possible. Following t_{RDV} there is an additional time interval, $t_{RELEASE}$, after which the iButton releases the 1–Wire line so that its voltage can return to V_{PULLUP}. The duration of $t_{RELEASE}$ may vary from 0 to 45 µs; its nominal value is 15 µs.

C.3. Reset and Presence Detect

The Reset Pulse provides a clear starting condition that supersedes any time slot synchronization. It is defined as a single low pulse of minimum duration of eight time slots or 480 µs followed by a Reset–high time t_{RSTH} of another 480 µs (Figure 4). After a Reset Pulse has been sent, the iButton will wait for the time t_{PDH} and then generate a Presence Pulse of duration t_{PDL}. No other communication on the 1–Wire bus is allowed during t_{RSTH}. The Presence Pulse can be used to trigger a hardware interrupt or to automatically power up equipment like Touch Pens. If an iButton is disconnected from the probe, it will pull its data line low via an internal current source of 5 µA. This simulates a Reset Pulse of unlimited duration. As soon as the iButton detects a high level on the data line, it will generate a Presence Pulse.

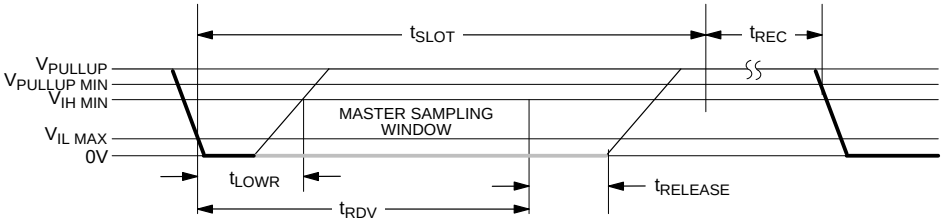
The nominal values are 30 µs for t_{PDH} and 120 µs for t_{PDL}. With the same worst–case tolerance band, the measured t_{PDH} value indicates the internal time base of the fastest device. The sum of the measured t_{PDH} and t_{PDL} values is five times the internal time base of the slowest device. If there is only one device on the line, both values will deviate in the same direction. This correlation can be used to build an adaptive system. Special care must be taken to recalibrate timing after every reset since the individual timing characteristics of the devices vary with temperature and load.

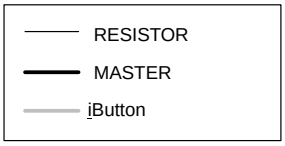
The accuracy of the time measurements required for adaptive timing is limited by the characteristics of the master's input logic, the time constant of the 1–Wire line (pullup resistor x cable capacitance) and the applied sampling rate. If the observed rise time or fall time exceeds 1 µs or the highest possible sampling rate is less than 1 MHz, adaptive timing should not be attempted.

C.4. Example Pulse Train

For illustrative purposes, a reference pulse train has been defined (Figure 5) to explain how the waveforms are generated rather than showing a complete session. It starts with a reset sequence including a Presence Pulse. Further time slots show all waveforms of reading and writing 1s and 0s. Any communication session can be constructed from the waveforms of this pulse train.

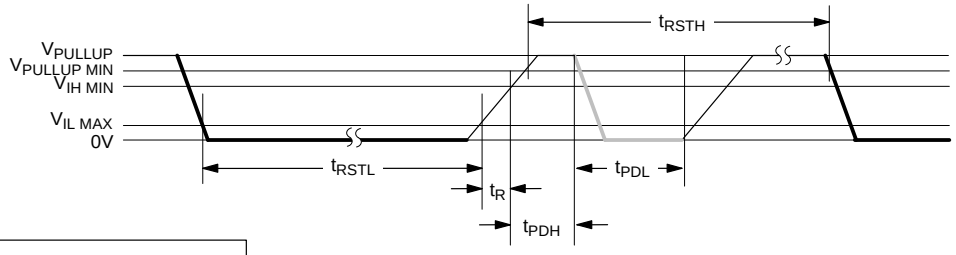
READ–DATA TIME SLOT Figure 3

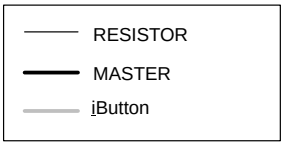




60 µs ≤ t_{SLOT} < 120 µs
1 µs ≤ t_{LOWR} < 15 µs
0 ≤ $t_{RELEASE}$ < 45 µs
1 µs ≤ t_{REC} < ∞
t_{RDV} = 15 µs

RESET AND PRESENCE PULSE Figure 4

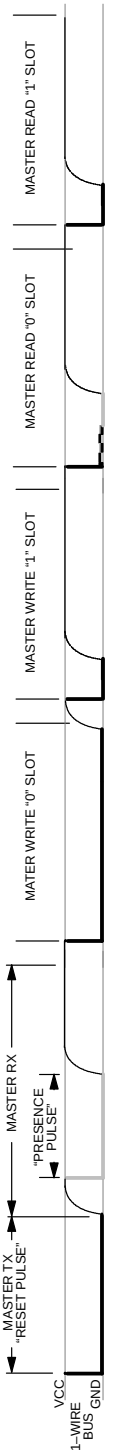




480 µs ≤ t_{RSTL} < ∞ *
480 µs ≤ t_{RSTH} < ∞ (includes recovery time)
15 µs ≤ t_{PDH} < 60 µs
60 µs ≤ t_{PDL} < 240 µs

* In order not to mask interrupt signalling by other devices on the 1–Wire bus, t_{RSTL} + t_R should always be less than 960 µs.

REFERENCE PULSE TRAIN Figure 5



III. FUNDAMENTALS

A. TTL Interface

This category includes all logic families and microprocessors that use positive logic with a maximum 0.8V for a logical 0 or LOW and a minimum of 2.2V for a logical 1 or HIGH. These voltages combined with a current source capability of at least 1 mA and a sink capability of more than 4 mA interface to a broad class of digital electronics.

Since the 1–Wire bus is an open drain system, an open drain/collector driver is required to connect the output port to the bus (Figure 6). This driver can be a general purpose NPN transistor with a resistor connected between base and output port or an n–channel MOSFET or any open drain/collector driver available in the logic family as long as the pullup voltage is equal to the driver voltage. Even a tri–state driver with its logic input tied to ground can be used, connecting the output gate to the tri–state control input. Depending on the characteristics of the driver (inverting/non–inverting), it may be required to complement the logic value of the output gate to compensate for the driver's signal inversion.

Reading from the 1–Wire bus can usually be accomplished by directly connecting the 1–Wire bus to the input port of the master. If the pullup–voltage of the 1–wire bus is too low or if the capacity of the cable produces slopes too slow for the logic family, it may be required to employ a comparator as interface and to adjust the reference voltage to optimize noise margins and timing characteristics. If the comparator inverts the signal, this inversion needs to be compensated by the software.

Generally it is recommended to test this type of interface carefully, starting with reset pulses generated by software and watching the slopes with an oscilloscope. If the timing specifications of Figure 4 are met, and the presence pulse is seen, one may proceed and test the software to generate the Write–Zero and Write–One time slots. After this works properly, the next step is reading the ROM. This is done by performing a reset cycle first, followed by 2 Write–One time slots, 2 Write–Zero time slots, 2 Write–One time slots and 2 Write–Zero time slots (this is equivalent to sending 33H, least significant bit first). After this, 64 Read Data time slots need to be generated.

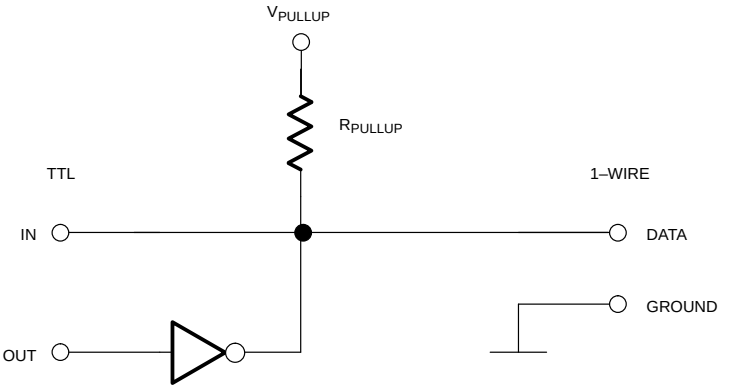
Since all timing depends on the clock frequency of the microcontroller, it is required to count the number of clock cycles for the execution of each command within the loop. The level of an output pin of a microprocessor will usually not change at the end of a command, but a few clock cycles earlier. The actual reading from any input pin also occurs some clock cycles before the end of the command. If the description of the microprocessor does not give sufficient details on this, a test series with different clock frequencies may be required. As long as the microprocessor is executing time–sensitive code, i.e., reading from the 1–Wire bus or writing to it, jumps or calls may occur only while the 1–Wire bus is idle. Interrupts from other sources than the 1–Wire bus must be disabled.

B. RS232 Interface

This section covers all interfaces that use a special controller to generate all timing and reference signals required for serial communication. The typical controller for this type of interface is the UART 8250. It relieves the microprocessor of the burden of time–critical software execution. The microprocessor simply puts the character code to be transmitted into the transmit register of the UART and the UART will do the work. A character is received by the microprocessor just by reading the UART's receive register. If the serial transmission is finished or if there is data for the microprocessor, this condition is signalled by the UART through flags that can be polled or by interrupts.

To function properly, the UART requires configuration with respect to baud rate, number of data bits per character, parity and number of start and stop bits. These terms are common for serial communication, but fit the needs of 1–Wire networks with their time slots and separate synchronization if a bit rather than a character is framed by the start condition. For 1–Wire communication the UART is set up for a high baud rate and each character delivered by the UART represents a bit on the 1–Wire bus. The microprocessor must separate the bits of a byte, least significant bit first, and write them as appropriate characters to the UART. To read data, the microprocessor has to assemble the bits received through characters back into bytes. These functions are not time–sensitive and can easily be programmed in a high level language.

TTL INTERFACING Figure 6



RS232 Conventions

Unlike TTL logic, RS232 has been established to transport data over long lines. Therefore different current drive characteristics and higher voltages are required to represent the logic levels 0 and 1.

The values to be expected are:

+3V to +15V for 0,

which is identical to the polarity of the start bit and

–3V to –15V for 1,

which is identical to the polarity of the stop bit and the idle state. The voltage range from –3V to +3V is undefined.

All voltages are measured with respect to ground. The receive channel and the transmit channel are independent wires, called RXD and TXD. Since RS232 ports are often used for communication via phone lines, several control signals are also included in the standard. Not all of these control signals need to be implemented with a communication device. For 1–Wire applications only the control signals DTR (Data Terminal Ready) and RTS (Request to send) are needed. Other signals often found with RS232 are DSR (Data Set Ready), which is the response to DTR, and CTS (Clear To Send), which is the response to RTS. How these signals are provided on a connector is detailed in Table 2. Full documentation on RS232 is beyond the scope of this application note. For a complete description please refer to other literature.

Table 2

SIGNAL	9–PIN CONNECTOR	25–PIN CONNECTOR	DESCRIPTION	FUNCTION
RXD	2	3	Receive Data	input
TXD	3	2	Transmit Data	output
DTR	4	20	Data Terminal Ready	output
RTS	7	4	Request to Send	output
GND	5	7	Ground	(reference)
DSR	6	6	Data Set Ready	input
CTS	8	5	Clear to Send	input

Hardware Simplified Model

The standard hardware of a RS232 interface is shown in Figure 7a. The UART is hooked to the system bus like an 8–bit memory device. The three address inputs A0 to A2 make the UART appear as a block of 8 read/write memory locations. On the other side there are the serial communication signals and the control signals mentioned above. Since the UART is a 5V device, special drivers and receivers for handling higher voltages are required. Circuit diagrams of bipolar integrated drivers (1488) and receivers (1489) are shown on Figures 7b and 7c. CMOS drivers and receivers are also available as standard products.

By design, the output current of an RS232 driver is limited to ±10 mA typically. Since the voltage definitions for 1 and 0 on the RS232 channel are reversed with respect to the conventions of positive logic, RS232 drivers and receivers are actually inverting devices. The inversion in the transmit channel is compensated through an inversion in the receive channel. Thus a 1 written to the transmit register will appear as a 1 at the serial output, as –15V at TXD, as a 1 at the serial input of the receiver and finally be read as a 1 in the receive register of the receiving UART.

For energy efficiency with battery operated equipment, the RS232 drivers and receivers are often replaced by simple 5V inverting drivers. This is definitely not compatible with the RS232 standard, but may be sufficient to control a modem or to transfer data through a short cable. Interfaces like this are called 5V RS232 within this application note. They run on the same software as the standard RS232, but are electrically almost the same as the TTL interface.

Programmer's Model

To write software for the 8250 UART one must know the basic address where the registers of the UART are hardwired to. This address is generally an equipment specific variable, and therefore will be referenced by the name SPA (Serial Port Address) rather than by a physical address. Of the 8 theoretically accessible addresses within the UART only 7 are really implemented, using the relative addresses 0 to 6. The names of these registers are as follows:

address:

SPA	+0	Receive (read)/Transmit (write) Data Register
	+1	Interrupt Enable Register
	+2	Interrupt Identification Register (read only)
	+3	Line Control Register
	+4	Modem Control Register
	+5	Line Status Register
	+6	Modem Status Register

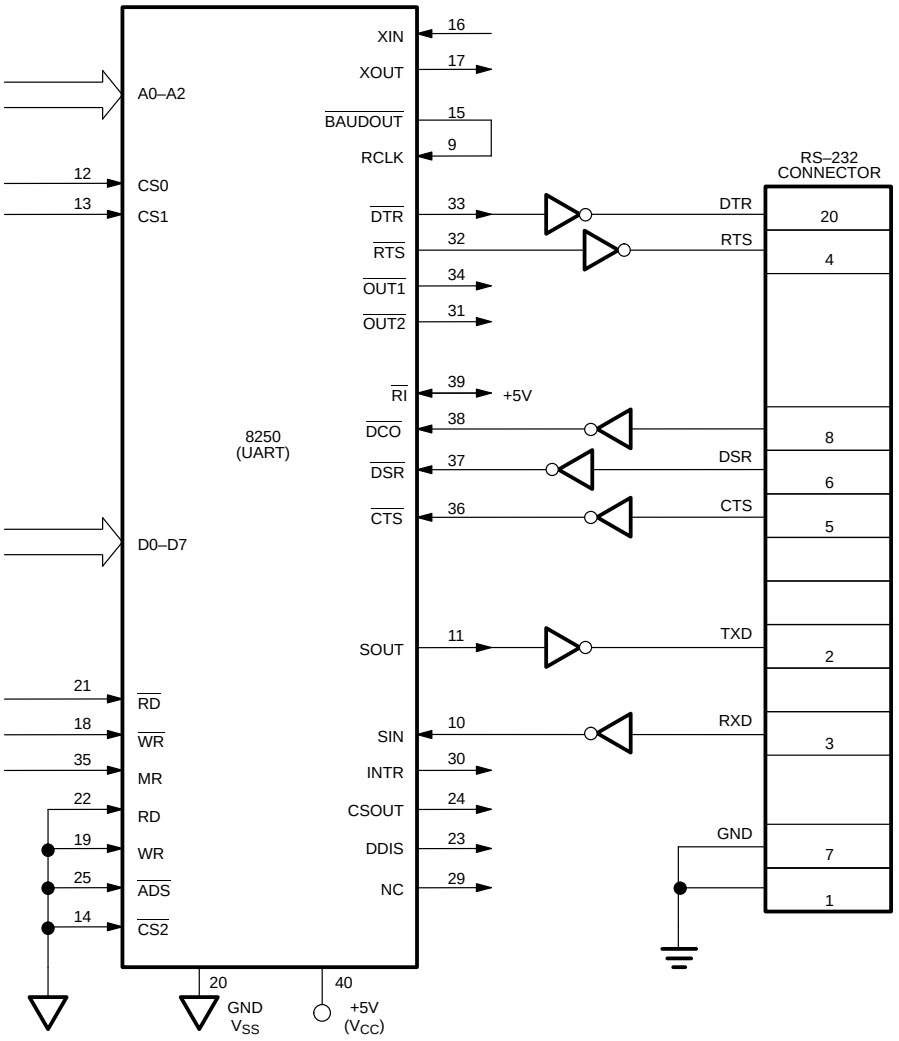
The meaning of the UART's control bits and their position within the control registers is detailed in Figure 8. For 1–Wire communication, the Interrupt Identification Register and Modem Status Register are not used.

To define the speed or baud rate of a serial communication, there is a 16 bit register, called Divisor Latch. This register is accessed as two bytes using the same addresses as the Data Register (least significant byte) and the Interrupt Enable Register (most significant byte). To access the Divisor Latch, the Divisor Latch Access Bit DLAB must be set to 1. DLAB is the most significant bit of the Line Control Register. As long as DLAB is set, the Data Register and the Interrupt Enable Register are not accessible. For the commonly used crystal frequency of 1.8432 MHz, the divisor latch must contain a number between 2304 (900hex, 50 bps) and 1 (115.2k bps).

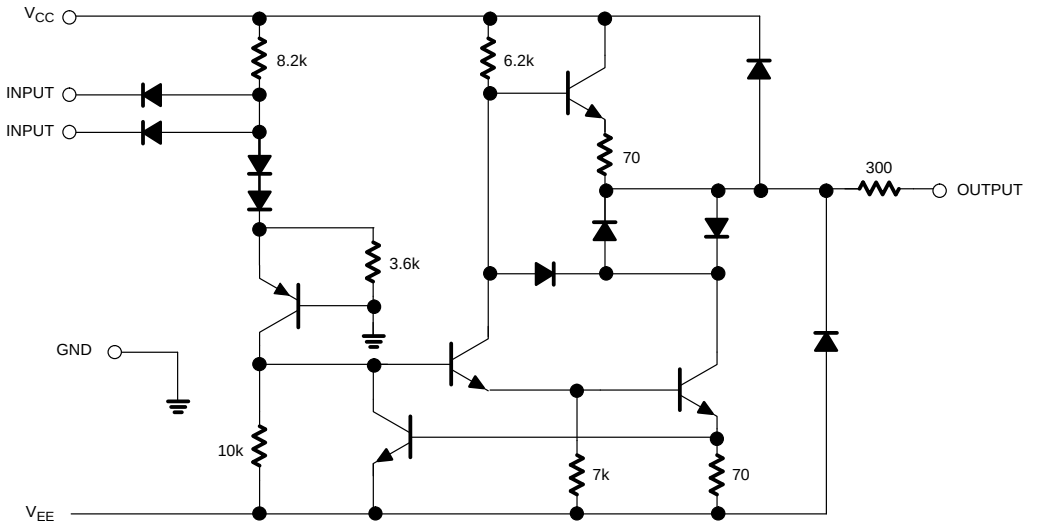
1–Wire Communication through the UART

As mentioned above, to write one bit to the 1–Wire bus, the UART is programmed to transmit one character. Since the receive and transmit channels of the UART are operating independently, but using the same communication setup, reading from and writing to the 1–Wire bus can occur at the same time. The start condition generated at the UART's serial output is fed to the 1–Wire bus and is simultaneously returned to the serial input, triggering the process of reading one character. The waveform is completely defined by the baud rate, the polarity of the start and stop bits and the bit pattern of the character. The serial output of the UART is high (~ 5V, idle) between characters, low (~0V) for the start bit and equal to the value of the data bit being transmitted. An idle (~5V) to low (~0V) transition at the UART's serial input triggers the process of receiving a character. The first bit is understood as start bit; the remaining bits are shifted into the receive register in the same polarity as they arrive at the serial input. Bits received after the receive register is full, are ignored.

HARDWARE OVERALL CONCEPT Figure 7a



TYPICAL DRIVER Figure 7b



TYPICAL RECEIVER Figure 7c

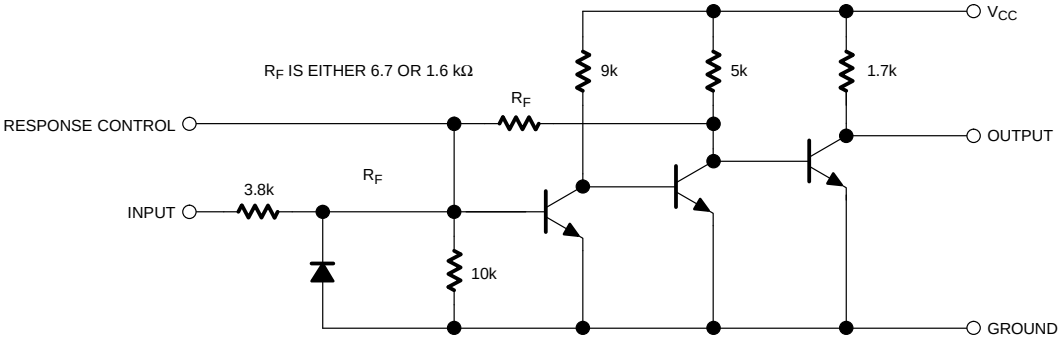


Figure 8

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	
SPA + 1	bit 15	baud rate divisor most significant byte						bit 8	DLAB=1
SPA + 0	bit 7	baud rate divisor least significant byte						bit 0	DLAB=1
SPA + 0	receive/transmit data register								DLAB=0
SPA + 1	0	0	0	0	Interrupt enable; 0=disabled				DLAB=0
SPA + 2	0	0	0	0	0	Interrupt ident. register, not used			DLAB=0
SPA + 3	DLAB, Divisor Latch Address Bit	Set Break, 0=disabled	Stick Parity, 0=disabled	Even Parity don't care	Parity Enable, 0=disabled	Nr. of stop bits, 0=1 stop bit	WLS1 Word Length	WLS0 Word Length	DLAB=0
SPA + 4	0	0	0	Loop	Out2	Out1	RTS, Request to Send	DTR, Data Terminal Ready	DLAB=0
SPA + 5	0	TSRE, transmit shift register empty	THRE, transmit hold register empty	Break Interrupt	Framing Error	Parity Error	Overrun Error	Data Ready	DLAB=0
SPA + 6	Modem Status Register, not used								DLAB=0

Reset and Presence Detect

The reset and presence detect cycle is performed by setting the baud rate to 10473 bps (Divisor Latch = 11decimal), the character length to 8 bits (WLS0=1, WLS1=1) and transmitting the character code F0. Including the start bit, this produces a pulse of 5 x 95.5 µs low (start bit plus four 0's) followed by 5 x 95.5 µs high (four 1's plus stop bit) at the serial output of the UART. If an iButton is present, then it will assert its presence pulse during the time interval where the most significant bits of the character code are transmitted. If after the transmission the receive data register reads F0, then there is no iButton. If one or more bits of the transmitted F are changed to 0s, than a presence pulse was received.

Read/Write One Bit

To generate data time slots, the UART must be set to a baud rate of 115.2k bps (Divisor Latch = 1). With a character length of 6 bits (WLS0=1, WLS1=0), any transmitted character will consist of a pulse train of 8 x 8.68 µs, beginning with a low start bit, followed by true data bits and a high stop bit. This matches the waveform for fast 1–Wire communication. If the character code is 00, than a Write–Zero time slot is generated. A character code of FF will produce a Write–One time slot.

As long as writing to the 1–Wire bus is desired, all characters received by the UART must be read and can be discarded. To read data, Write–One time slots must be generated. Bits received from the iButton are returned in their true form in the least significant bit of the character code found in the receive data register. If the iButton sends a one, all bits of the character code will read one. If the iButton sends a zero, one or more of the least significant bits of the character code will be zero, depending on the internal time base of the iButton.

Not all UARTs behave exactly the same as the 8250. Some of them do not support the character length of 6 bits. To circumvent potential problems from this restriction, the software examples in this document always use the character length of 8 bits. This extends the transmission of one bit by 17.36 µs and reduces the effective baud rate from 14.4k bps to 11.5k bps, but remains well within the specification of 1–Wire timing.

IV. CIRCUITS FOR 5V INTERFACES (TTL AND RS232)

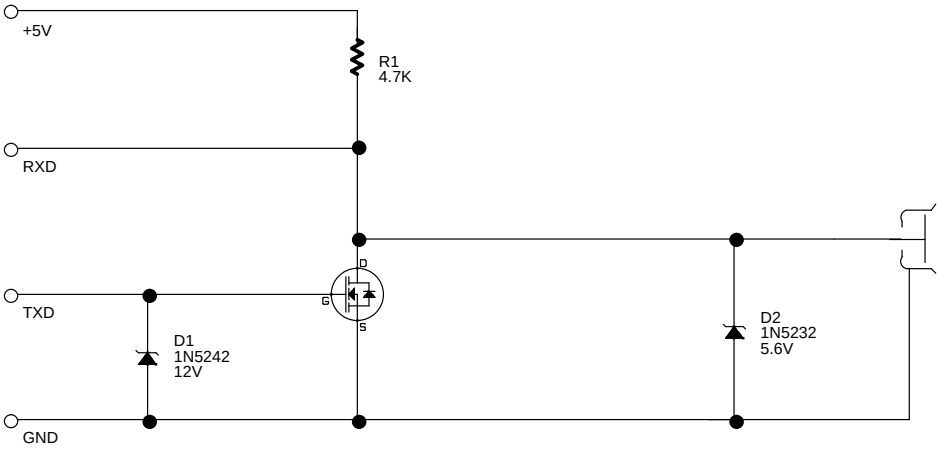
A. TTL Read All

This is the simplest interface for iButton applications. It is suitable for reading all iButtons and writing NVRAM based devices. The circuit diagram (Figure 9) conforms to the principle of Figure 6. The diodes D1 and D2 protect transistor Q1 and the input of the microprocessor, respectively, against damage from electrostatic discharge (ESD). R1 is the 1–Wire pullup resistor. If the microprocessor runs on 5V, the same supply can directly be connected to R1. If only a higher supply voltage than 5V is available, any monolithic or discrete positive 5V regulator can be used to provide the pullup voltage for the 1–Wire bus.

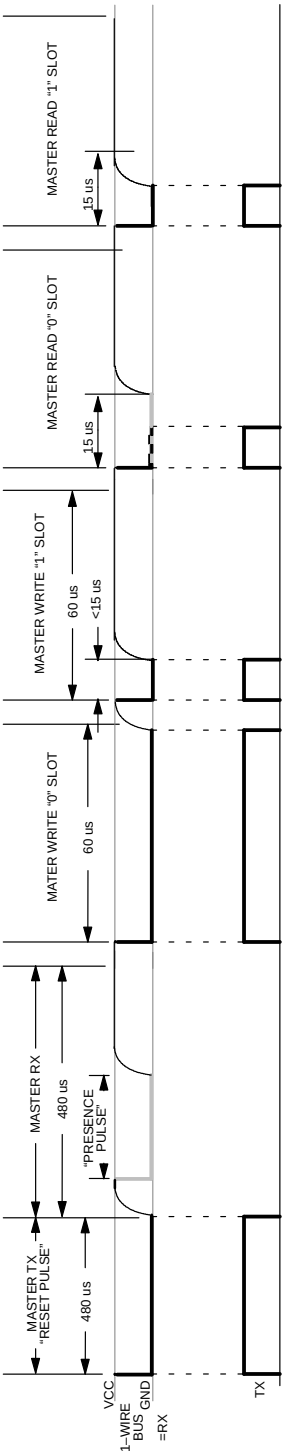
The characteristics of the components are not critical. The transistor 2N7000 has been chosen since it is a very common product and has a low threshold voltage. If desired, a small signal bipolar transistor or any available open–drain or open–collector inverting driver can be used instead of Q1. If Q1 is an npn–transistor, a resistor bypassed by a small capacitor between the TX–input and the base terminal is required to limit the input current.

The logic level high at TX will produce a low on the 1–Wire bus. To generate a Write–One or Read Data time slot, a short high pulse (1 µs < t < 15 µs) must be applied to the TX input. A Write–Zero Time Slot is formed by a 60 µs high pulse at TX. Data from iButtons is received in its true form. If idle, TX must be held at a logic low level. The reference pulse train and other relevant waveforms for this circuit are shown on Figure 9a. The timing for this type of interface is directly generated by the microprocessor. A software example for this type of interface is found later in this document.

TTL READ ALL CIRCUIT Figure 9



TTL READ ALL WAVEFORMS Figure 9a



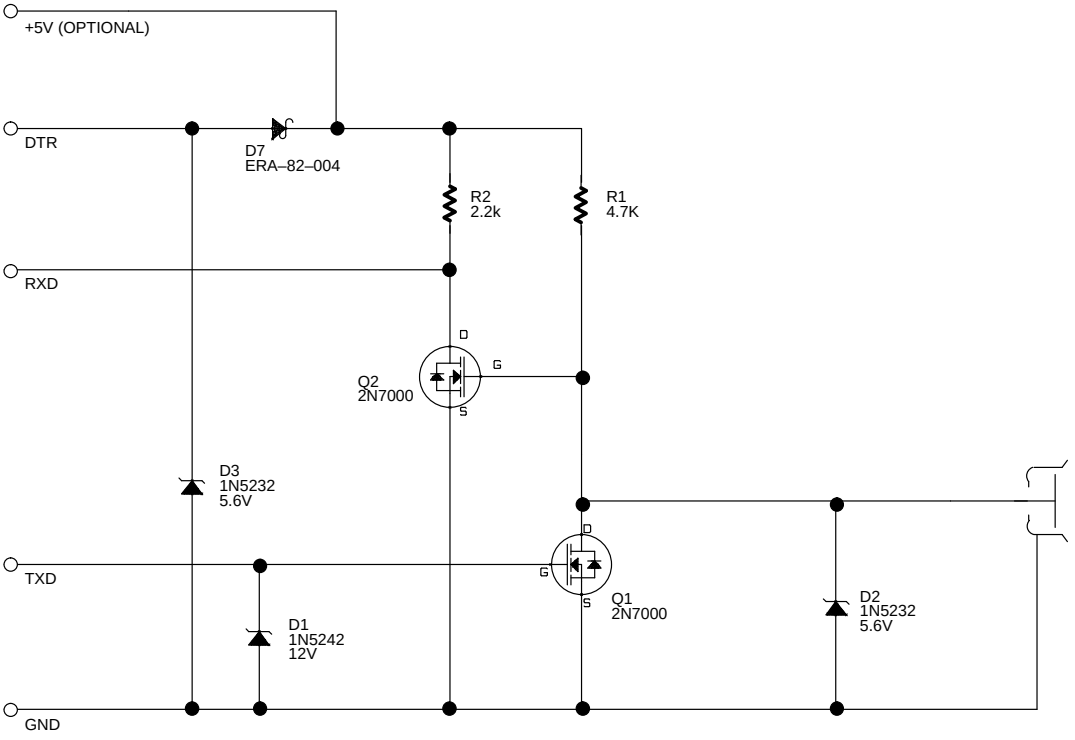
B. 5V RS232 Read All

This interface is required for equipment that uses a UART to provide all timing but does not conform to the voltage levels of the RS232 standard. The logic levels, however, are the correct polarity so that it can be operated with the same software as the standard RS232 interface.

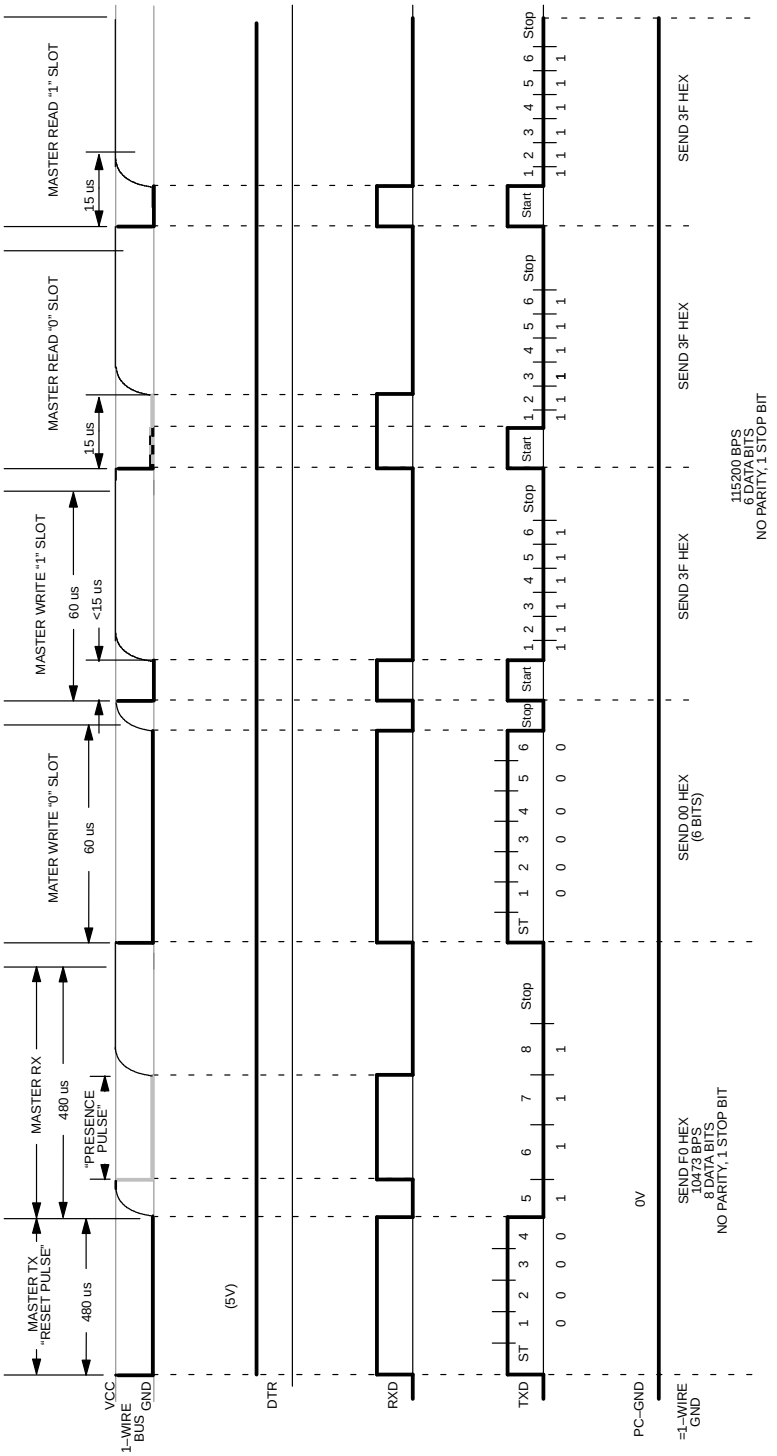
The circuit shown on Figure 10 is suitable for reading all iButtons and writing NVRAM based devices. New compared to Figure 9 are D3, Q2 and R2. D3 has the same function as D2, i.e., ESD protection. Q2 and R2 form an inverter to restore the correct polarity for the receive channel of the UART. Since there is no pin defined as a power supply with RS232, the DTR signal is used to provide the pullup voltage for the 1–Wire bus and power for the inverter in the receive channel. Everything else is identical to Figure 9.

The reference pulse train and other important waveforms for this circuit are shown on Figure 10a. Due to the double inversion in both the transmit and the receive channel (inverters between UART and RS232 connector in Figure 7, Q1 and Q2 in Figure 10) the logic levels of the 1–Wire bus are the same as those at the UART. A 1 written to the UART's transmit register will appear as a 1 on the 1–Wire bus, a zero on the 1–Wire bus will be received as a 0. No software–inversion is needed. Further details on programming the UART for 1–Wire communication are found section III.b. For software examples please refer to a later section of this document.

5V RS232 READ ALL CIRCUIT Figure 10



5V RS232 READ ALL WAVEFORMS Figure 10a

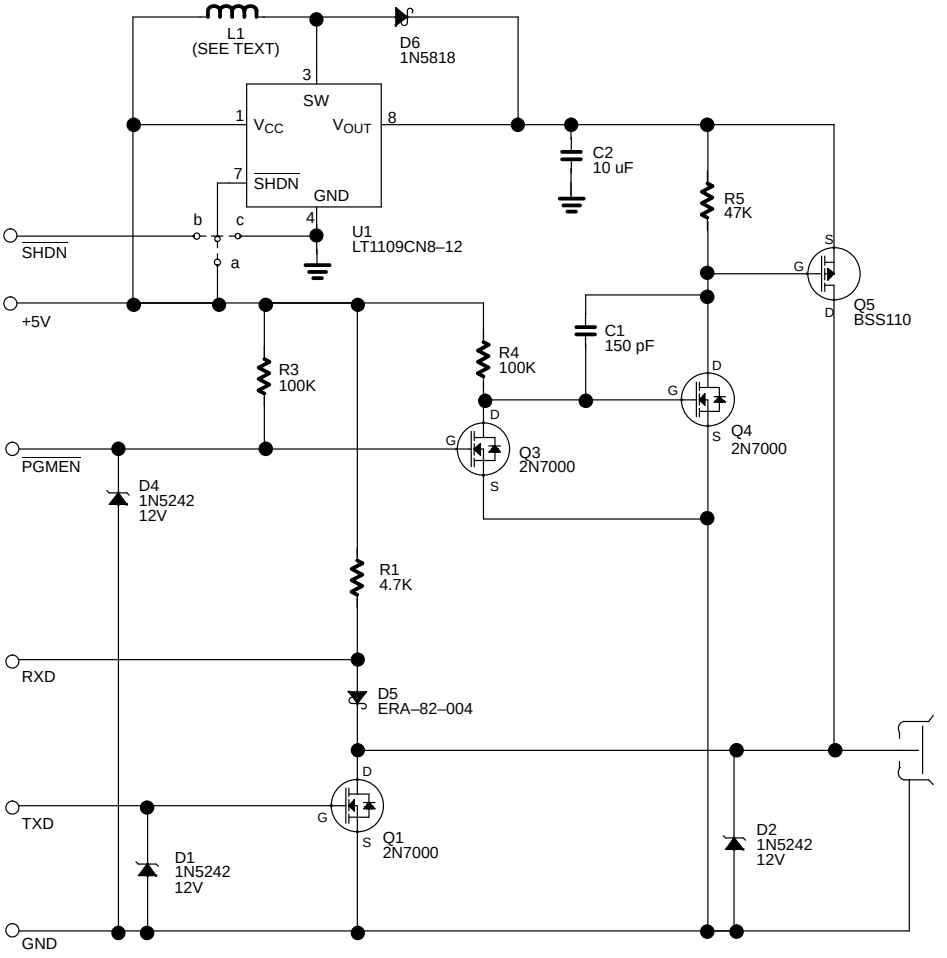


C. TTL R/W All (Voltage Converter)

The circuits described so far can read all iButtons and write NVRAM–based devices. For other technologies, however, voltage requirements other than 5V are necessary. One important group of iButtons, called Add–Only Memories, is based on EPROM technology and therefore needs a programming pulse of 12V to copy data from the scratchpad to the EPROM cells. Another device, the Temperature iButton, will operate on 5V but requires a low impedance pull–up to 5V while measuring the temperature. To fulfill the requirements of these and future devices, the circuits described above need to be upgraded. These two new functions require two more signals and the 12V programming supply can be provided by a DC–to–DC voltage converter.

The complete circuit of such a universal interface in a TTL–version is shown in Figure 11. It is a compatible superset of Figure 9. The components Q1, R1, D1, D2 and their functions are the same as before. Additional requirements include the control input $\overline{\text{PGMEN}}$ with the diode D4 for ESD protection, the diode D5, the cascaded inverters R3, Q3, R4 and Q4 & R5, C1, the pull–up switch Q5 and the controlled voltage converter IC1 with its external components L1, D6 and C2.

TTL R/W ALL CIRCUIT Figure 11



$\overline{\text{PGMEN}}$ is the active low input to activate the pull–up switch. If not connected, $\overline{\text{PGMEN}}$ will be held high through R3 to avoid unwanted activation of the pull–up switch. The voltage converter can be controlled in three ways: a) hard–wired for continuous operation, b) activated by an external signal, or c) permanently shut down. Case a) is intended for applications which never require a strong 5V pull–up. If there is no control signal available from the master and strong 5V pull–up as well as EPROM programming is required, then a mechanical switch can be used to switch between case a) and c). Case b) offers the most flexibility. For EPROM programming $\overline{\text{SHDN}}$ needs to be high, for strong 5V pull–up it should be low. If the voltage converter is shut down, L1 and D6 together with a conducting Q5 provide the required low–impedance path to 5V.

If the signal $\overline{\text{PGMEN}}$ becomes active (i.e., is low) then the voltage at the gate of Q4 rises from 0V to approximately 5V, causing Q4 to conduct. This is equivalent to feeding a low level from $\overline{\text{PGMEN}}$ to the gate of Q5. The capacitor C1 between gate and drain of Q4 slows down the rise and fall of Q4's gate–source voltage and therefore determines the ramp rate of the programming pulse. As soon as the gate voltage of Q5 changes from the quiescent state of 5 or 12V (depending on mode of operation) to 0V, the P–channel transistor becomes conducting and pulls the 1–Wire bus either to 12V (if the voltage converter is on) or to 5V, bypassing R1.

From the strictly logical point of view, the double inversion (Q3, Q4) is unnecessary. The reasons for this circuit are to convert from a TTL–level system to a 12V system (5 volts on the gate of Q5 is not sufficient to turn the transistor off if the voltage converter is running), to avoid high voltage feedback from the voltage converter through R5 to the TTL–level control input and to extend the rise and fall time of the 12V programming pulse to the required minimum of 5 µs. High voltage feedback from the 1–Wire bus to the receive input of the microprocessor is avoided by the diode D5, which becomes conducting only when the voltage on the 1–Wire bus is lower than 5V.

The monolithic voltage converter IC1 requires L1, C2 and D6 for operation. It is activated by a low level at its TTL–compatible input $\overline{\text{SHDN}}$. The right choice of L1, D6 and C2 is essential for reliable operation. D6 is a Schottky diode, recommended part number 1N5818, C2 is a low ESR tantalum capacitor of 10 µF. L1 must be a low ESR device between 20 to 100 µH, capable of withstanding current peaks of approximately 0.5 A without magnetic saturation. To avoid EMI problems, L1 should be a pot–core or toroid type; a rod core type is not recommended. For further details on the voltage converter and its external components please refer to the appropriate data sheet and application notes. The LT1109 is just one example of available parts. Other manufacturer's components or modules can be used as well.

The duration of the programming pulse (pulse width of $\overline{\text{PGMEN}}$) or the strong pull–up is determined by software. Program examples are given later in this document.

D. 5V RS232 R/W All (Voltage Converter)

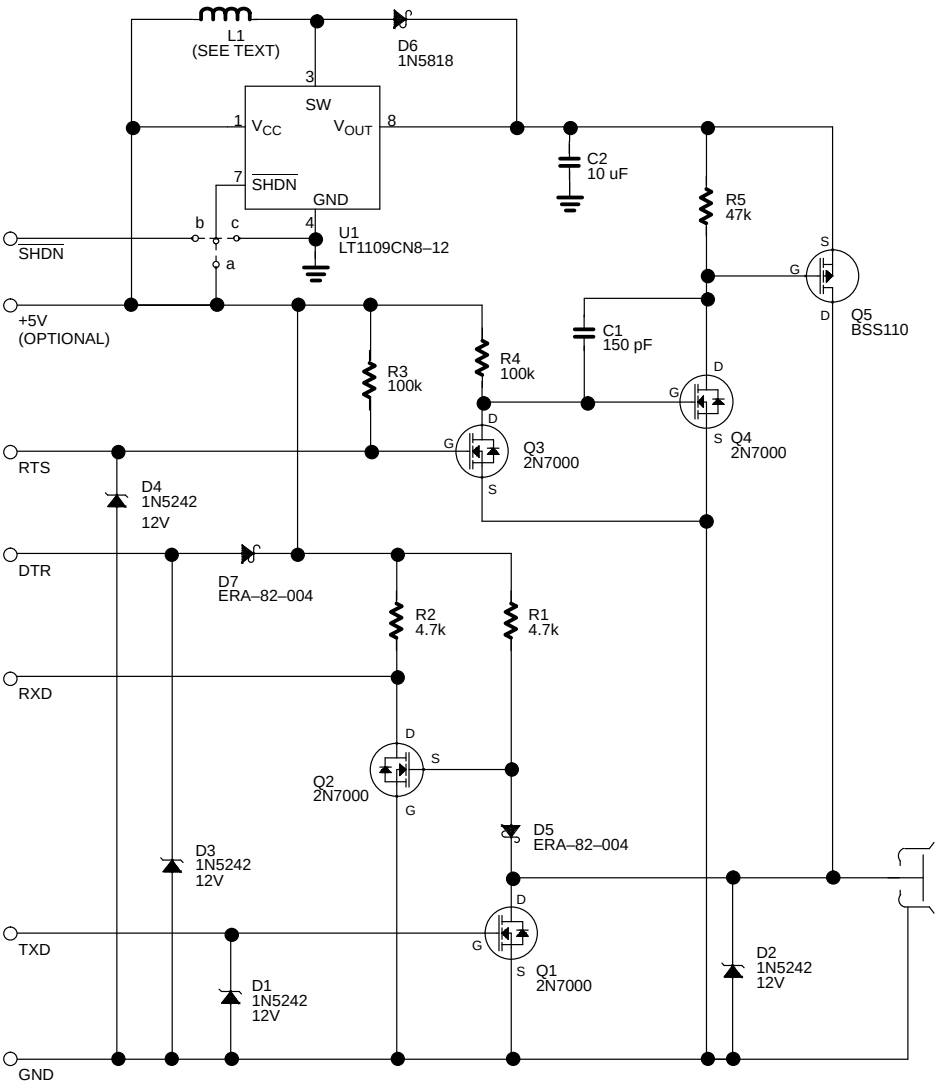
The universal upgrade of the interface of Figure 10 is shown in Figure 12. The components Q1, R1, D1, D2, Q2, R2, D3 and their functions are the same as before. Additional requirements include the control signal RTS with the diode D4 for ESD protection, the diodes D5, D7, the cascaded inverters R3, Q3, R4 and Q4 & R5, C2, the pull–up switch Q5 and the controlled voltage converter IC1 with its external components L1, D6 and C2.

RTS is used to activate the pull–up switch Q5. DTR may act as power supply, if it is able to source 5V at 25 mA. The strong pull–up is made possible by L1 and D6 if the voltage converter is shut down. If idle (i.e., the bit controlling RTS in the Modem Control Register of the UART is set to 1), RTS will be at 5V. To activate the pull–up switch, the RTS bit in the UART (see Figure 8) must be cleared to 0. This will cause a 0V level at the RTS pin of the RS232 connector. The DTR bit in the Modem Control Register of the UART must be set to 1. If power is supplied from the outside, the status of DTR becomes a don't care. In this case, D7 prevents driving the DTR line with the external power supply.

For reliable operation of this circuit, DTR must reach a high level of 5V minimum. If this is not possible, an external 5V supply must be connected as shown in Figure 12.

The function of the other components of this circuit has already been explained in the section IV.C. RTS is equivalent to $\overline{\text{PGMEN}}$ in Figure 11. D5 now prevents feedback from the 1–Wire bus to the internal power supply during the programming pulse. The duration of the programming pulse (pulse width of RTS) cannot be controlled by the UART alone. Software to provide the correct timing is found later in this document.

5V RS232 R/W ALL CIRCUIT Figure 12



V. CIRCUITS FOR 12V RS232 INTERFACES (COM PORT)

A. Read All

If equipment has a true RS232 port using current limited drivers with voltage capabilities of at least ±8V, then a simple passive circuit is sufficient to interface to the 1–Wire bus. Figure 13 shows all details; the waveforms are found on Figure 13a. This interface operates on the same software as the circuit shown in Figure 10. The waveforms at RXD and TXD with respect to the computer's ground are basically the same. The major difference is that instead of 0V, a true negative voltage will be found, representing the idle state or a logic 1. Neglecting absolute voltage levels, the waveform observed at RXD is essentially an inversion of the waveform that would be observed on a 1–Wire data line for a 0V to 5V system.

The ground potential of the computer is different from the 1–Wire ground. This allows the iButton to experience typical voltage levels (0V to 6V) on the 1–Wire data line relative to the 1–Wire ground, while the serial port generates both positive and negative voltages relative to the serial port ground. D1 clamps the data line to a constant potential of nominally 3.9V. The time slots for 1–Wire communication are generated by changing the potential of the 1–Wire ground with respect to the 1–Wire data line. The 1–Wire pullup resistor is in the path from the 1–Wire ground to TXD, which provides the voltage for the 1–Wire bus. D2 limits the voltage swing on the 1–Wire bus to a maximum value of 6.2 Volts. Since DTR is kept at 3.9V, D2 also limits the most negative voltage occurring at RXD to –2.3V. D2 is conducting only when the voltage at TXD is negative with respect to the computer's ground. D3 limits the voltage between 1–Wire ground and 1–Wire data when the voltage at TXD is positive. D4 couples TXD with RXD when TXD is positive and bypasses R1 to provide a low resistance path to initiate a time slot on the 1–Wire bus. D4 is nonconducting when the voltage at TXD is negative with respect to the computer's ground. If an iButton pulls the 1–Wire data line low (e. g. at a presence pulse or when sending out a zero data bit), it shorts DTR to RXD, resulting in a positive voltage at RXD. This positive voltage arrives as a zero in the UART's receive data register.

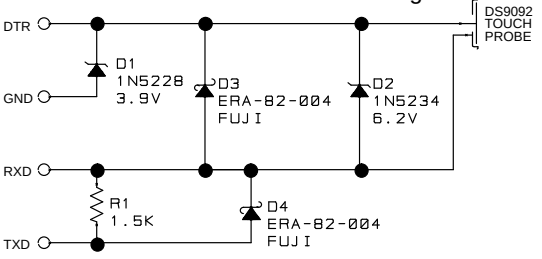
Probing the 1–Wire bus at the data contact and the 1–Wire ground with an oscilloscope, would show nothing unusual, except that the voltage swing is at the upper end of the tolerable range. Probing RXD and TXD with another oscilloscope hooked up at the computer's ground, would also reveal quite normal waveforms. The only thing one might be concerned about is the poor negative voltage of about –2.3V found at RXD. Probing all three signals (1–Wire bus, TXD, RXD) with one oscillocope hooked up at the computer's ground, would show a nearly constant voltage of 3.9V at the 1–Wire bus.

The positive voltage of approximately +4.1V at RXD is well within the RS232–specification, the negative voltage is slightly out of specification. Since the total swing on the 1–Wire bus is limited to approximately 6V by the characteristics of iButtons, a more negative voltage could be produced by replacing D1 by a zener diode of 3.2V, for example. Unfortunately, this would increase the permanent current load for DTR.

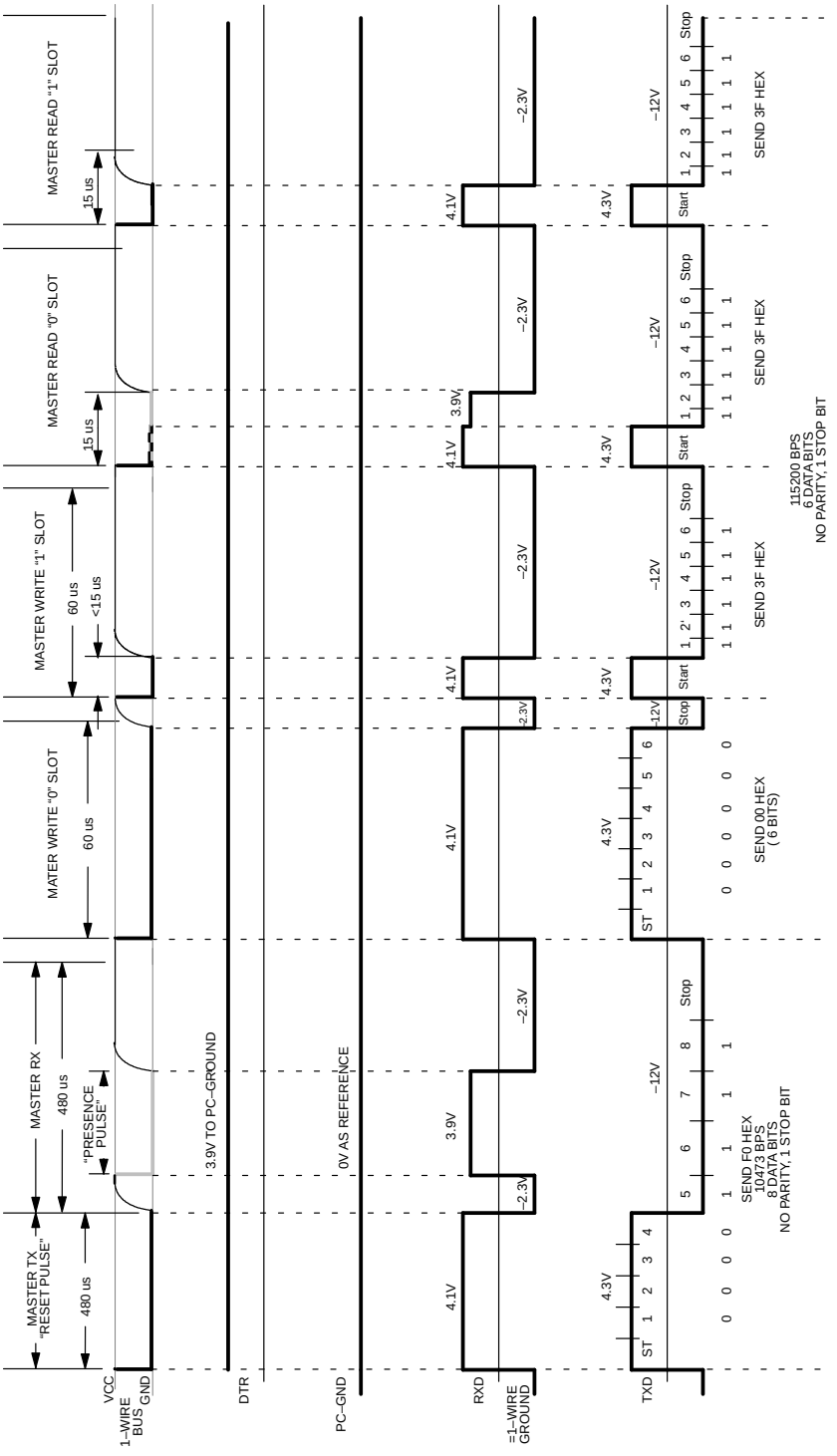
Fortunately, real RS232 receivers are more sensitive to a weak positive voltage than they are to a poor negative voltage. Although it is not completely within the specification of RS232, this interface with the components specified on Figure 13 has proven to be reliable with most desktop personal computers. Small computers, especially battery powered models, might not have the required current capability to run this interface. In case of difficulties, one should use the 5V RS232 interface instead.

This interface is sold as COM Port Adapter DS9097. It is applicable for reading all iButtons and writing to SRAM–based devices. Due to the high pullup voltage and the low pullup resistor, this interface together with the right software also allows operation of the Temperature iButton directly through the COM Port. Software to operate this interface is explained later in this document.

12V RS232 READ ALL CIRCUIT Figure 13



12V RS232 READ ALL WAVEFORMS Figure 13a



B. R/W All

The simple adapter of Figure 13 can be upgraded for programming Add–Only Memories. Figure 14 shows the details. R1 and the diodes D1 to D4 are the same in both circuits. There is a new signal in use, called RTS. When doing normal 1–Wire communication, RTS is constantly at a high positive voltage of nominally +12V. As long as the voltage at RTS remains positive, Q1 and Q2 are conducting. This allows D1 and D2 to provide the same functions as in Figure 13. Since D1 and D2 are conducting, there is not enough voltage across D5 and D6 to draw any current. The gate–source voltage of Q3 is determined by R2, R4, the voltage between DTR and RTS and the position of the contacts inside the connector for the external DC supply. D7 prevents current flow from source to drain through the substrate diode of Q3 when VDS is negative. If no external power supply is connected, R2 and R4 form a voltage divider providing a negative VGS for Q3. With an external power supply connected, R2 will hold the gate of Q3 at the same level as the source (VGS = 0V). None of these conditions will allow any current flow through Q3. With a positive voltage at RTS, the p–channel transistor Q4 remains non–conducting, regardless if an external power supply is connected or not. Thus the upgraded circuit behaves the same as the simple COM–Port adapter.

To generate a programming pulse for EPROM based devices, RTS needs to be switched to a negative voltage. This is done under software control by simply clearing the associated control bit in the UART and resetting the bit as soon as the programming pulse needs to be ended. Depending on whether an external supply is connected or not, the behavior of this circuit is slightly different.

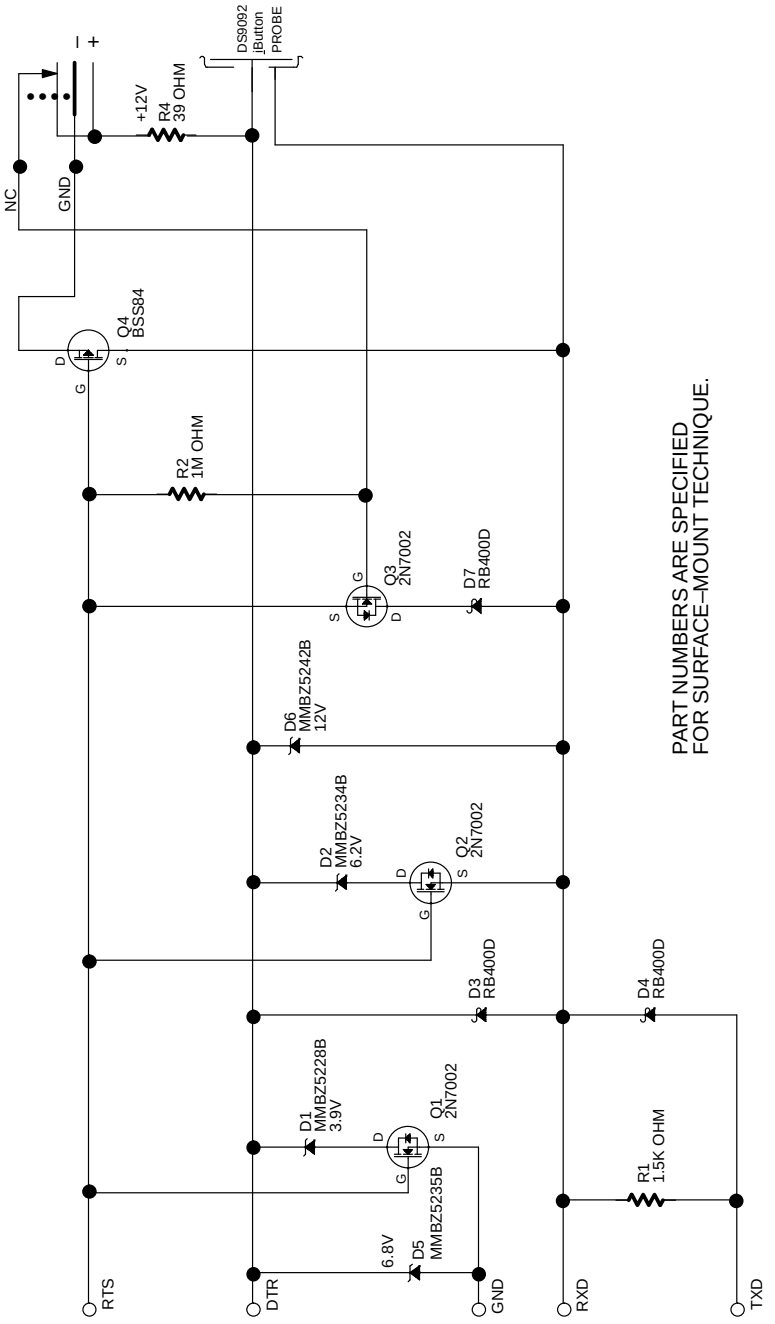
If there is no external supply, Q4 has no function and the following sequence occurs (Figure 14a): A negative voltage at RTS will switch off Q1 and Q2, activating D5 and D6 instead of D1 and D2. This increases the voltage at DTR from 3.9V to 6.8V and defines a limit of 12V for the voltage on the 1–Wire bus. Simultaneously, the negative voltage at RTS will make VGS of Q3 equal to the voltage between DTR and RTS, which is a positive value. This will switch Q3 into a conducting state, feeding the negative voltage from RTS through D7 to the 1–Wire ground. As soon as the voltage on the 1–Wire bus reaches 12V, D6 will become active as voltage limiter. Thus the voltage on RTS is limited to approximately –5.4V. During the programming pulse, the voltage at RXD will be –5.2V instead of –2.3V. This has no impact on the logic of the UART, since a positive voltage is required to trigger the reception of a character.

The COM–Port powered mode of this circuit works properly if the RS232–drivers are able to provide enough current for programming. If more than one bit of the addressed memory byte need to be altered, then more current is needed during the programming pulse. Depending on the RS232 drivers, the available energy may not be sufficient for programming all bits of a byte. There are two possible solutions to this problem. One possibility uses an adaptive programming algorithm, where multiple passes are made for each byte. For example the software may program the first four bits of every byte on the first pass, and the remaining bits on the second pass. The other possibility is to provide an auxiliary energy source, i.e., by connecting an external 12V DC supply.

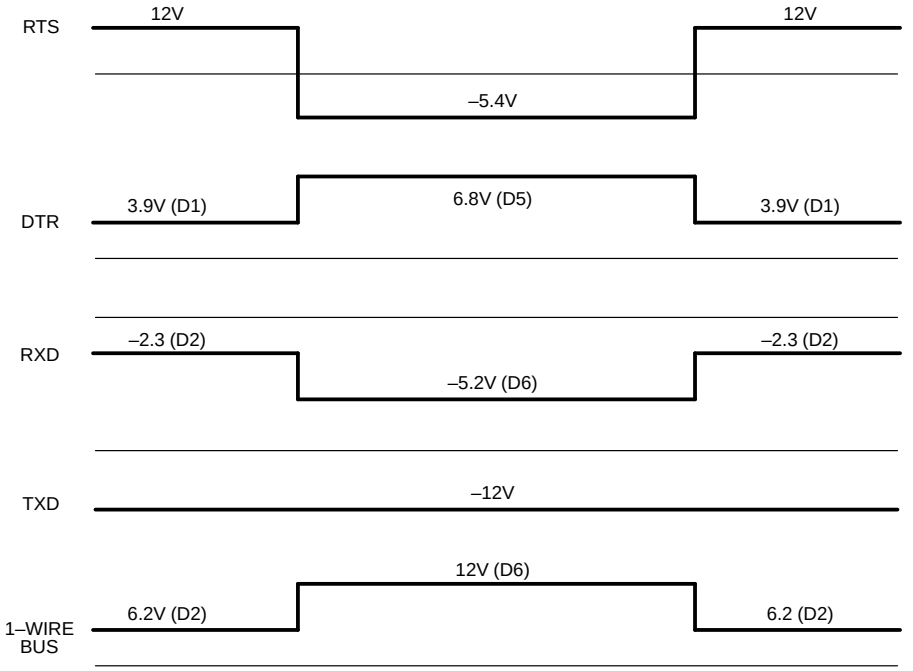
If the external supply is connected and operating, the contacts inside the connector will open. This prevents any current flow through Q3. As the voltage at RTS goes negative, Q1, Q2, D5, D6 will do the same as without the external supply. Instead of having no function, Q4 now will connect the negative end of the external voltage source with the 1–Wire ground, providing the desired low–impedance programming voltage. Since RTS has no other load than the gates of four transistors, there will be the full voltage swing. The voltages at RXD and TXD are again determined by D5 and D6 and exhibit the same waveforms as before.

This upgraded version of a COM–Port adapter is shipped with the iButton Starter Kit DS9092K and is also available as the DS9097E. It is applicable for reading and writing all iButtons. Due to the high pullup voltage and the low pullup resistor, this interface together with the right software allows operation of the Temperature iButton directly through the COM Port.

12V RS232 R/W ALL CIRCUIT Figure 14



12V RS232 R/W ALL PROGRAMMING WAVEFORMS Figure 14a



VI. INTRINSICALLY SAFE

A. Definition

iButtons satisfy a very high safety standard which makes them well suited for applications in hazardous environments. iButtons meet the UL#913 (4th Edit.) requirements as Intrinsically Safe Apparatus, Approved under the Entity Concept for use in Class I, Division 1, Group A, B, C, and D Locations. Intrinsically safe means that the probability of causing an explosion or accident in hazardous locations is not increased when using approved equipment, even if this equipment should be faulty. Since iButtons have been certified as intrinsically safe under the entity concept, they may reside in a hazardous environment but cannot be read or written in that environment unless the unit performing the reading or writing has also been certified as intrinsically safe under the same entity concept.

The iButton might be affixed to a tanker truck and contain maintenance information. The truck could enter and exit a hazardous location (fuel depot, for example) without concern over an increased potential for an explosion due to the iButton device. If no intrinsically safe reader/writer is available, any updates to the iButton would have to occur outside of the hazardous area. Should it be necessary to read or update the iButton within the hazardous area (record the fuel dispensed, for example), a certified intrinsically safe reader/writer must be used.

There are two options available to provide an intrinsically safe reader/writer. The first involves taking any piece of equipment capable of reading or writing iButtons and submitting it to an approved NRTL (Nationally Recognized Testing LAB) to be tested and certified under the same entity concept as the iButtons. The second option takes advantage of reader/writer equipment that has already been tested and certified as intrinsically safe (laptop computer, handheld reader, etc.) and uses those test results along with a specially designed iButton probe adapter to create an entire system that is intrinsically safe. One such unit utilizing this second option is the PSION Organizer II, Model LZ64. This unit is used as an example to show how an intrinsically safe system can be realized.

B. Example of an Intrinsically Safe System

To use the LZ64 as an intrinsically safe iButton Reader/Writer, an adapter is required. This adapter limits voltages and currents to safe values in the event a fault occurs. In the case of a fault, due to its internal construction, a maximum voltage of 17.22V at the 16–pin connector of the LZ64 may occur allowing a current of up to 1.55A. The adapter discussed here limits these values at the iButton Probe to a maximum voltage of 15V and a maximum current of 10mA (Values required by the iButtons in order to be certified as intrinsically safe). These values together with the maximum inductance of 18 µH and maximum capacitance of 0.2nF of an iButton fulfill the requirements for a complete system that is intrinsically safe according to UL specifications.

Figure 15 shows the complete circuit of the adapter. Essential components are the four current shunts Q1/R1 to Q4/R4, three Zener diodes D2 to D4 and one self–resetting fuse SS1. The Schottky diode D1 is optional. It protects iButtons by suppressing negative undershoots on the 1–Wire bus.

If by fault the LZ64 presents up to 17.22V at its connector and there is an open circuit at the iButton Probe, then the 12V Zener diodes D2 to D4 will limit the voltage at the iButton Probe to a value well below 15V. The intrinsically–safe regulations demand that this limitation will work correctly even if two of the protecting devices should fail. Therefore three Zener diodes are provided instead of one. If a Zener diode fails, it will either represent a short or it will be non–conducting. In either case, the voltage at the iButton Probe is limited to a safe value.

If by fault the LZ64 presents up to 17.22V at its connector and there is a short at the iButton Probe, then a current will flow through the resistors R1 to R4. The voltage drop across these resistors acts to turn on their respective transistors and causes base currents to flow. These base currents multiplied by the current gain of the transistors will direct most of the current to ground and limit the current at the iButton Probe to less than 10mA. Q1 will sink most of the current. Two of these Q/R stages are required to limit the current available at the iButton Probe under worst case conditions to less than 10 mA. The other two current shunts are redundant since again the circuit must operate correctly with up to two faults. The self–resetting fuse with a trip point below 100mA will open in less than one second and thus prevent thermal damage to the transistors. It will also serve as an indicator to the operator that a fault or malfunction has occurred.

Although this adapter is designed for use with the LZ64, it can be used with similar intrinsically safe equipment with the same or lower faulted open–circuit voltage or short–circuit current to form an iButton Reader/Writer. This adapter does not impede writing to EPROM based iButtons.

VII. COMMENTED SOFTWARE

A. Software Architectural Model

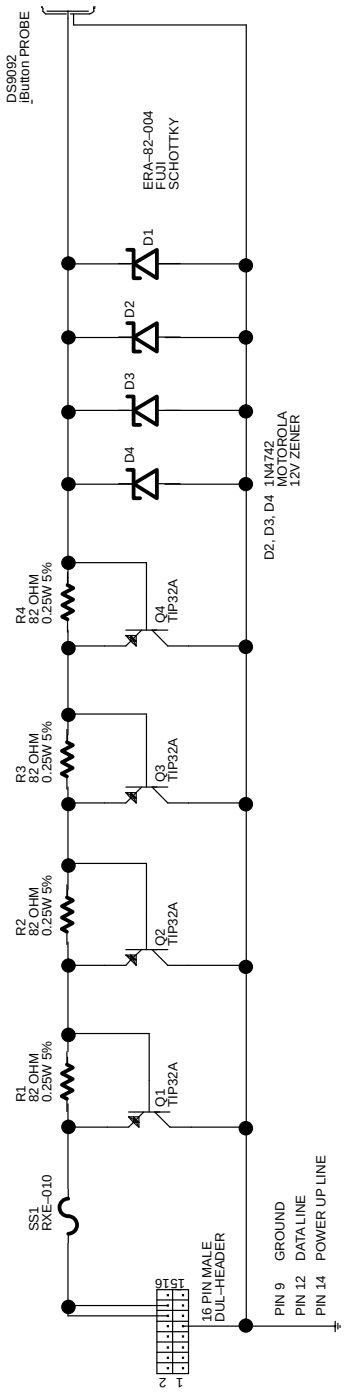
The software that manages data transfer to and from iButtons is related to the ISO reference model of Open System Interconnection (OSI). This model specifies a layered protocol having up to seven layers, denoted as Physical, Link, Network, Transport, Session, Presentation, and Application. The Application layer represents the final application designed by the customer. A Session layer may or may not be needed, depending on the environment in which the iButtons are used.

According to the ISO model, the electrical and timing requirements of iButton and the characteristics of the 1–Wire bus comprise the Physical layer. Details have already been given in section II of this document.

The Link layer defines the basic communication functions of iButtons, which are the hardware dependent functions of Reset, Presence Detect and bit transfer. Circuits for interfacing iButtons and general information on the software to operate these interfaces have already been presented in sections III, IV, and V. In this section, the software itself, specifically the functions TouchReset and TouchByte, are discussed in detail.

The Network layer provides the identification of iButtons and the associated network capabilities based on the unique lasered identification number. Software for this layer is built up using the low–level functions of the Link Layer. Since this software is independent of any particular interface, it is not within the scope of this document.

INTRINSICALLY SAFE ADAPTER (EXAMPLE) Figure 15



The Transport layer is responsible for the data transfer between the non–ROM segments of iButtons and the master, and the data transfer from the scratchpad to the final storage areas and special registers of the iButton. Due to their EPROM technology, Add–Only Memories require special attention for writing data. The Temperature iButton may require special hardware together with appropriate software to do a temperature measurement. To comply these devices, the hardware specific function PulWidth has been provided on the Transport layer. Details are given in this chapter. All other software of the transport layer is independent of the type of interface, and therefore is not discussed here.

The layers Link, Network, and Transport are the foundations of the Presentation layer. This layer provides a DOS–like file system supporting functions like Format, Directory, Type, Copy, Delete, Optimize, and integrity check. Since the Presentation layer itself is based on software of the lower layers, its software is independent of any particular interface. Full details of the Presentation layer are given in the iButton TMEX Professional Software Developer's Kit DS0621–SDK. For software examples beyond the hardware dependent functions TouchReset, TouchByte and PulWidth, please refer to the "Book of DS19xx iButton Standards" and the iButton Starter Kit DS9092K.

A matrix that indicates which software of this section matches with which hardware is given in Table 3. For the 5V TTL type interface, assembly language code for the 8051 has been provided. For the group of interfaces based on the UART 8250, code examples in Pascal and C are included. This particular software has been adapted to and verified with IBM–compatible PCs employing a 8253 timer at 2.3863633 MHz and running under DOS. The timing is practically independent of the CPU clock rate. Under WINDOWS there is a lot more software being executed around an application program. This overhead introduces a significant influence from the CPU clock rate to the desired timing with the function PulWidth. The functions TouchReset and TouchByte are timed by the UART only and therefore are independent of the operating system.

SOFTWARE/HARDWARE MATRIX Table 3

LANGUAGE	**8051 ASM**	**PASCAL AND C**
TIMING	**CPU CRYSTAL**	**8250 UART (1.8 MHz) and 8253 TIMER (2.4 MHz)**
Electric Type	**5V TTL**	**5V RS232, 12V RS232**
SRAM R/W EPROM Read	TouchReset, TouchByte 1.8 or 11 MHz	TouchReset and TouchByte Pascal or C–Language
EPROM Write	0.5 ms pulsewidth: PULWIDTH(1) at 1.8 MHz PULWIDTH(6) at 11 MHz	0.5 ms pulsewidth: PULWIDTH(1193) under DOS

B. TTL–Interface R/W All

As a representative for all microprocessor timed 1–Wire interfaces the industry–standard 8051 microcontroller has been chosen. The following pages show two versions of assembly language code to provide the functions TouchReset and TouchByte. The first example is written for an 11.0592 MHz crystal, the second one for 1.8432 MHz. The higher frequency is very common since it supports all standard baud rates with the highest accuracy. The lower frequency is the lowest that can comply with the 1–Wire timing. The port to be used as 1–Wire bus is defined in the parameter DATA_BIT. Parameter passing from the subroutines TouchReset and TouchByte is very simple: If an iButton is present on the 1–Wire bus, TouchReset will return a set carry flag; otherwise carry is cleared. To send one byte to the 1–Wire bus, the byte to be sent is loaded into the accumulator before calling TouchByte. If one intends to read, the accumulator is loaded with FFH. This generates correct Read Data Time Slots and returns data from the 1–Wire bus to the calling program through the accumulator. These conventions are valid for both versions of TouchReset and TouchByte.

The procedure to generate a programming pulse is the same for both clock frequencies. It generates a 0.5 ms LOW pulse at the port named PROGRAM. If the clock frequency is 1.8 MHz, then the accumulator needs to be

loaded with 1 before calling this procedure. For a clock frequency of 11 MHz the value of 6 loaded into the accumulator will generate a pulse of the same duration. Software considering the Temperature iButton will be published as soon as the device is available. Generally, it is not a difficult task to adapt the procedure PulWidth to a pulsewidth of 2 seconds.

8051 ASSEMBLY LANGUAGE, 11.0592 MHz

```
DATA_BIT  BIT       P0.0

; The following 8051 code uses a bi-directional port pin (specified by
; DATA_BIT) for 1-wire I/O. This code was written for an 11.0592 MHz
; crystal.
;
;         Procedure TouchReset
;
;         This procedure transmits the Reset signal to the
;         iButton and watches for a presence pulse. On return,
;         the Carry bit is set if a presence pulse was detected,
;         otherwise the Carry is cleared. The code is timed for
;         an 11.0592 MHz crystal.
;
TOUCHRESET:
          PUSH      B                   ;       Save the B register.
          PUSH      ACC                 ;       Save the accumulator.
          MOV       A,        #4        ;       Load outer loop variable.
          CLR       DATA_BIT            ;       Start the reset pulse.
          MOV       B,        #221      ;   2.  Set time interval.
          DJNZ      B,        $         ; 442.  Wait with Data low.
          SETB      DATA_BIT            ;   1.  Release Data line.
          MOV       B,        #6        ;   2.  Set time interval.
          CLR       C                   ;   1.  Clear presence flag.
WAITLOW:
          JB        DATA_BIT, WH        ;       Exit loop if line high.
          DJNZ      B,        WAITLOW   ;       Hang around for 3360
          DJNZ      ACC,      WAITLOW   ;          us if line is low.
          SJMP      SHORT               ;       Line could not go high.
WH:
          MOV       B,        #111      ;       Delay for presence detect.
HL:
          ORL       C,        /DATA_BIT ; 222.  Catch presence pulse.
          DJNZ      B,        HL        ; 222.  Wait with Data high.
SHORT:
          POP       ACC                 ;       Restore accumulator.
          POP       B                   ;       Restore B register.
          RET                           ;       Return.
;
;         Procedure TouchByte
;
;         The procedure TouchByte sends the byte in the accumulator
;         to the iButton and simulatneously returns one
;         byte from the iButton in the accumulator. Note that
;         the NOPs in the following code are intended to give the
```

```
;         optimum performance when using a 11.0592 MHz crystal.
;         Their purpose is to make the pulses as long as
;         possible consistent with the iButton timing
;         constraints. When using other crystal frequencies,
;         the delays in this code should be adjusted to conform
;         to the timing requirements of the iButton.
;
TOUCHBYTE:
          PUSH      B                   ;       Save the B register.
          MOV       B,        #8        ;       Setup for 8 bits.
BIT_LOOP:
          RRC       A                   ;   1.  Get bit in carry.
          CALL      TOUCHBIT            ;   2.  Send bit.
          DJNZ      B,        BIT_LOOP  ;   2.  Get next bit.
          RRC       A                   ;       Get final bit in ACC.
          POP       B                   ;       Restore B register.
          RET                           ;       Return to caller.
TOUCHBIT:
          CLR       DATA_BIT            ;   1.  Start the time slot.
          NOP                           ;   1.  Delay to make sure
          NOP                           ;   1.    that the iButton
          NOP                           ;   1.      sees a low for at
          NOP                           ;   1.        least 1 microsecond.
          MOV       DATA_BIT, C         ;   2.  Send out the data bit.
          NOP                           ;   1.  Delay to give the
          NOP                           ;   1.    data returned from
          NOP                           ;   1.      the iButton
          NOP                           ;   1.        time to settle
          NOP                           ;   1.          before reading
          NOP                           ;   1.            the bit.
          MOV       C,        DATA_BIT  ;   1.  Sample input data bit.
          PUSH      B                   ;   2.  Save B register.
          MOV       B,        #12H      ;   2.  Delay until the end
          DJNZ      B,        $         ;   36.   of the time slot.
          POP       B                   ;       Restore B register.
          SETB      DATA_BIT            ;       Terminate time slot.
          RET                           ;       Return to caller.
;
          END                           ; End of module._
```

8051 ASSEMBLY LANGUAGE, 1.8432 MHz

```
;         iButton I/O Procedures for use with a 1.8432 MHz crystal.

TOUCHRESET:
          CLR       DATA_BIT          ;   – Pull the data line low.
          MOV       B,        #35     ;   2 Hold the data line
          DJNZ      B,        $       ;  70    low for 481.77
          NOP                         ;   1       microseconds.
          SETB      DATA_BIT          ;   1 Release the data line.
          MOV       B,        #130    ;   2 Short circuit timeout.
```

```
          CLR       C                   ;   1 Presence pulse detector.
WAITLOW:
          JB        DATA_BIT, WAITHIGH  ; 260 Go look for Presence pulse.
          DJNZ      B,        WAITLOW   ; 260 Abort on short circuit.
          SJMP      ABORT               ;     Short circuit (3.8877 ms).
WAITHIGH:
          MOV       B,        #18       ;   2 Prepare for high period.
HL:
          ORL       C,        /DATA_BIT ;  36 Trap the Presence pulse.
          DJNZ      B,        HL        ;  36 Wait out 481.77 microsec.
ABORT:
          RET                                 Return.
TOUCHBYTE:
          MOV       B,        #8        ;    Prepare to move 8 bits.
BIT_LOOP:
          RRC       A                   ;    Move LSB to Carry.
          JC        SENDONE             ;    If Carry then send 1.
          CLR       DATA_BIT            ;    Otherwise send 0.
          SJMP      DELAYSET            ;  2 Wait out rest of time slot.
SENDONE:
          CLR       DATA_BIT            ;    Start read/write 1 slot.
          SETB      DATA_BIT            ;  1 Set data line.
          MOV       C,        DATA_BIT  ;  1 Read data line.
DELAYSET:
          NOP                           ;  1 Delay 7 more cycles
          NOP                           ;  1    to produce enough
          NOP                           ;  1       delay to complete
          NOP                           ;  1          the time slot.
          NOP                           ;  1
          NOP                           ;  1
          NOP                           ;  1
          SETB      DATA_BIT            ;  1 Done (65.1 microseconds).
          DJNZ      B,        BIT_LOOP  ;    Repeat to send 8 bits.
          RRC       A                   ;    Align final result.
          RET                           ;    Return.
```

8051 ASSEMBLY LANGUAGE PULSEWIDTH (1.8432 AND 11.0592 MHz)

```
PROGRAM   BIT       Pn.i
;
;         This procedure generates a 0.5 ms low pulse on port
;         Pn.i of an 8051 microprocessor, where 0 <= n <= 3
;         and 0 <= i <= 7.  The frequency of the crystal, in
;         multiples of the minimum frequency 1.8432 MHz, must
;         be passed in the accumulator.
;
PULWIDTH:
          MOV       B,        #38     ; Number of loops at 1.8432 MHz.
          MUL       AB                ; AB = # of loops given frequency.
          INC       B                 ; Adjust count value for
                                      ;    use with DJNZ instruction.
```

```
          PUSH      PSW             ; Preserve state of interrupts.
          CLR       EA              ; Inhibit all interrupts.
          CLR       PROGRAM         ; Bring the port pin low.
LOOP:
          DJNZ      ACC,      LOOP  ; Count while
          DJNZ      B,        LOOP  ;    pin is low.
          SETB      PROGRAM         ; Bring the port pin high.
          POP       PSW             ; Restore state of interrupts.
          RET                       ; Return.
```

C. RS232 Interface R/W All

UARTs like the 8250 can be connected to any microprocessor to implement a RS232 type interface. The software to operate the UART mainly consists of reading and writing the UART's internal registers (Figure 8). This can easily be done in any high level language. Depending on the computer, the physical address of the UART will be different, but the crystal will usually be a 1.8432 MHz type. Not regarding the UART's physical address, the software examples for TouchReset and TouchByte given on the following pages are very general. The languages C and Pascal are very common and a variety of compilers is available.

Unfortunately, the UART does not control the timing of the signals DTR and RTS. It only allows activation or deactivation of these signals by setting or clearing bits inside its control registers. The timing itself is left to the microprocessor and its peripheral timing circuits. From the software developer's point of view this is a step backwards to assembly language, where every command and its execution time at a specified clock frequency need to be counted. For this reason it is not possible to provide machine–independent software to generate the programming pulse.

The most common computer using a 8250 type UART to implement a RS232 interface is the IBM–compatible PC. These machines employ a programmable interval timer 8253 running at 2.3863633 MHz for general timing purposes. This timer is involved in controlling the timing of the software examples of PulWidth. The pulsewidth is specified by a formal parameter passed to PulWidth. For 0.5 ms the value of this parameter is 1193 decimal. Under DOS, the software examples given below will perform accurately and almost independent of the CPU clock. Due to a very different environment and use of resources under WINDOWS, the pulses will be longer and also dependent on the CPU clock. This can be compensated for experimentally by reducing the value of the parameter passed to PulWidth. Software considering the Temperature iButton will be published as soon as the device is available. For the 5V–type RS232 interface a pulsewidth of 2 seconds will be required for the strong pullup to 5V. The 12V RS232 interface has enough power available to run one Temperature iButton without extra power switching.

C LANGUAGE FOR UART 8250 SYSTEMS

```
/*
   In the following C language code, 1–wire I/O is accomplished using the
   serial port of an IBM PC or compatible. The serial port must be capable
   of a 115,200 bps data rate. Setup must be called before any of the
   touch functions to verify the existence of the specified com port and
   initialize it.

   ---------------------------------------------------------------------
   The setup function makes sure that the com port number passed to it
   is from 1 to 4 and has a valid address associated with it.
*/
uchar Setup(uchar CmPt)
{
     uint far *ptr = (uint far *) 0x00400000;
     uint SPA;
```

```
     /* check to see if it is a valid com port number and address */
     SPA = *(ptr+CmPt–1); /* get the address */
     if (CmPt < 1 || CmPt > 4 || !SPA )
        return FL;

     /* serial port initialization */
     outportb(SPA+3,0x83);  /* set DLAB */
     outportb(SPA  ,0x01);  /* bit rate is 115200 */
     outportb(SPA+1,0x00);
     outportb(SPA+3,0x03);  /* 8 dta, 1 stp, no par */
     outportb(SPA+1,0x00);  /* no interrupts */
     outportb(SPA+4,0x03);  /* RTS and DTR on */

     return TR;
}

/*--------------------------------------------------------------------------
 * Do a reset on the 1 wire port and return  0 no presence detect
 *                                           1 presence pulse no alarm
 *                                           2 alarm followed by presence
 *                                           3 short circuit to ground
 *                                           4 no com port found
 *
 * The global variable 'com_port' must be set to the com port that the
 * DS9097 COM Port Adapter is attached to before calling this routine.
 *
 */
uchar TouchReset( void )
{
     uint SPA,F,X,Y,tmp,trst=0;
     uint far *ptr = (uint far *) 0x00400000;
     ulong far *sysclk = (ulong far *) 0x0040006c;
     ulong M;

     /* get the serial port address */
     SPA = *(ptr+com_port–1);

     /* return if there is no address */
     if (!SPA) return 4;

     /* serial port initialization */
     outportb(SPA+3,0x83);  /* set DLAB */
     outportb(SPA  ,0x01);  /* bit rate is 115200 */
     outportb(SPA+1,0x00);
     outportb(SPA+3,0x03);  /* 8 dta, 1 stp, no par */
     outportb(SPA+1,0x00);  /* no interrupts */
     outportb(SPA+4,0x03);  /* RTS and DTR on */

     /* Initialize the time limit */
     M = *sysclk +1;
```

```
     /* loop to clear the buffers */
     do { tmp = inportb(SPA+5) & 0x60;  } while (tmp != 0x60);

     /* flush input */
     while (inportb(SPA+5) & 0x1) X = inportb(SPA);

     outportb(SPA+3,0x83);  /* set DLAB */
     outportb(SPA+1,0x00);  /* baud rate is 10473 */
     outportb(SPA  ,0x0B);
     outportb(SPA+3,0x03);  /* 8 dta, 1 stp, no par */
     outportb(SPA  ,0xF0);  /* send the reset pulse */

     /* wait until character back or timeout */
     do
     {
          Y = inportb(SPA+5);
          F = Y & 0x1;
     } while ( !F && (*sysclk <= M) );

     if (F) X = inportb(SPA);
     else   return 3;

     if (X != 0xF0)           /* if more bits back than sent then there */
     {                        /* is a device  if framing error or break */
          trst = TR;
          if ( (Y & 0x18) != 0 )
          {
               trst = 2;

               /* loop to clear the buffers */
               do { tmp = inportb(SPA+5) & 0x60;  } while (tmp != 0x60);

               /* wait until character back or timeout */
               do
               {
                    Y = inportb(SPA+5);
                    F = Y & 0x1;
               } while ( !F && (*sysclk <= M) );

               if (F) X = inportb(SPA);
               else return 3;
          }
     }

     outportb(SPA+3,0x83);  /* set DLAB */
     outportb(SPA  ,0x01);  /* bit rate is 115200 */
     outportb(SPA+3,0x03);  /* 8 dta, 1 stp, no par */

     return trst;
}
```

```
/*---------------------------------------------------------------------------
 * This is the 1-Wire routine 'TouchByte', sometimes called 'DataByte'.
 * It transmits 8 bits onto the 1-Wire data line and receives 8 bits
 * concurrently. The global variable 'com_port' must be set to the
 * com port that the serial brick is attached to before calling this
 * routine. This com port must also be set to 115200 baud, 8 dta, 1 stp,
 * and no parity. This routine returns the uchar 8 bit value received.
 * If it times out waiting for a character then 0xFF is returned.
 */
uchar TouchByte(uchar outch)
{
     uchar inch=0,sendbit,Mask=1;
     uint SPA;
     uint far *ptr = (uint far *) 0x00400000;
     ulong far *sysclk = (ulong far *) 0x0040006c;
     ulong M;

     /* get the serial port address */
     SPA = *(ptr+com_port-1);

     /* Initialize the time limit */
     M = *sysclk +2;

     /* wait to TBE and TSRE */
     do {} while ( (inportb(SPA+5) & 0x60) != 0x60 );

     /* flush input */
     while ( (inportb(SPA+5) & 0x1) )
          inportb(SPA);

     /* get first bit ready to go out */
     sendbit = (outch & 0x1) ? 0xFF : 0x00;

     /* loop to send and receive 8 bits */
     do
     {
          outportb(SPA,sendbit);  /* send out the bit */

          /* get next bit ready to go out */
          Mask <<= 1;
          sendbit = (outch & Mask) ? 0xFF : 0x00;

          /* shift input char over ready for next bit */
          inch >>= 1;

          /* loop to look for the incoming bit */
          for (;;)
          {
               /* return if out of time */
               if ( *sysclk > M )
                  return 0xFF;
```

```
               if ( inportb(SPA+5) & 0x01 )
               {
                  inch |= ((inportb(SPA) & 0x01) ? 0x80 : 0x00);
                  break;
               }
          }

     } while (Mask);

     return inch;   /* return the input char */
}
```

C LANGUAGE PULSEWIDTH FOR SYSTEMS USING 8253 AND 8250

```
// standard include header file
#include <dos.h>

// function prototype
void PulWidth(unsigned int);

// global variable
int SPA;

//---------------------------------------------------------------------------
//  This procedure creates a fixed pulse width for programming that is
//  approximately independent of system clock speed.  X is in units of
//  0.419 microseconds for values greater than about 1000.
//
void PulWidth(unsigned int X)
{
    unsigned int N,M;

    disable();                                  // turn off interrupts
    outportb(SPA+4,(inportb(SPA+4) & 0xFD));   // apply program pulse to rts
    outportb(0x43,0);                           // freeze value in timer
    M = inportb(0x40);                          // read value in timer
    M |= (inportb(0x40) << 8);
    do
    {
        outport(0x43,0);                        // freese value in timer
        N = inportb(0x40);                      // read value in timer
        N |= (inportb(0x40) << 8);
    }
    while (X > (M – N));

    outportb(SPA+4,(inportb(SPA+4) | 0x02));   // remove program Voltage
    enable();                                   // turn interrupts on
}
```

PASCAL LANGUAGE FOR UART 8250 SYSTEMS

```
{
   In the following pascal code 1–wire I/O is accomplished using the
   serial port of an IBM PC or compatible. The serial port must be capable
   of a 115,200 bps data rate.
}
Const
   SPA : Word = 0;           { Currently active serial port address }

Function TouchReset(N: Byte): Boolean;
{
   This function transmits the one–wire protocol reset sequence to
   the device connected to COM port number N. This sequence consists
   of a low pulse lasting a mimimum of 480 us followed by a high dead
   time lasting a mimimum of 480 us. The function returns True if a
   presence detect pulse occurs during the dead time, otherwise
   it returns False.
}
Const
   Init : Array[1..4] of Boolean = (True, True, True, True);

Var
   S : Array[1..4] of Word Absolute $40:0;
   T : LongInt Absolute $40:$6C;
   M : LongInt;
   F : Boolean;
   X, Y : Byte;

Begin
   SPA := 0; TouchReset := False;                  { Assume failure }
   If (N > 0) and (N < 5) and (S[N] > 0) then Begin  { Legal port # }
      SPA := S[N];                { Save active serial port address }
      If Init[N] then Begin   { Serial port requires initialization }
         Port[SPA +3] := $83;                       { Set the DLAB }
         Port[SPA]     :=   1;                      { Bit rate is   }
         Port[SPA +1] :=   0;                      {    115200 bps }
         Port[SPA +3] :=   3;               { 8 dta, 1 stp, no par }
         Port[SPA +1] :=   0;                       { No interrupts }
         Port[SPA +4] :=   3;                     { RTS and DTR on }
         Init[N] := False;               { Initialization completed }
      End;
      M := T +1;                        { Initialize the time limit }
      Repeat until Port[SPA +5] and $60 = $60;  { Await TBE & TSRE }
      While Odd(Port[SPA +5]) do X := Port[SPA];     { Flush input }
      Port[SPA +3] := $83;                              { Set DLAB }
      port[SPA+1] := 0;                        { Baud rate is 10473 }
      Port[SPA] := 11;
      Port[SPA +3] := 3;               { 8 data, 1 stop, no parity }
      Port[SPA] := $F0;                      { Send the reset pulse }
      Repeat                  { Wait until character back or timeout }
         Y := Port[SPA +5];
         F := Odd(Y);
```

```
      until F or (T > M);
      If F then X := Port[SPA] else X := $F0;
      If (X <> $F0) Then Begin        { If more bits back than sent }
         TouchReset := True;            {    then there is a device }
         If ((Y and $18) <> 0) Then Begin  { Framing error or break }
            Repeat until Port[SPA +5] and $60 = $60;  { TBE & TSRE }
            Repeat F := Odd(Port[SPA +5]) until F or (T > M);
            If F then X := Port[SPA];
         End;
      End;
      Port[SPA +3] := $83;                        { Set the DLAB }
      Port[SPA]     :=   1;               { Bit rate is 115200 bps }
      Port[SPA +3] :=   3;                 { 8 dta, 1 stp, no par }
   End;
End;

Function TouchByte(X: Byte): Byte;
{
   This function transmits the byte X to the device attached to the
   currently active serial port SPA, and returns a byte from the
   device as its value.
}
Var
   T    : LongInt Absolute $40:$6C;
   M    : LongInt;
   I, J : Byte;
Begin
   If SPA = 0 then TouchByte := X else Begin
      M := T +1;                       { Initialize the time limit }
      Repeat until Port[SPA +5] and $60 = $60;  { Await TBE & TSRE }
      While Odd(Port[SPA +5]) do I := Port[SPA];     { Flush input }
      I := 0; J := 0;      { Initialize output & input bit counters }
      Repeat
         If Odd(Port[SPA +5]) then Begin
            Inc(J); If Odd(Port[SPA]) then X := X or $80;
         End else If (I<=J) and (Port[SPA+5] and $20 = $20) then Begin
            If Odd(X) then Port[SPA] := $FF else Port[SPA] := 0;
            X := X shr 1; Inc(I);
         End;
      Until (J = 8) or (T > M);
      While (J < 8) do Begin
         X := X shr 1 or $80;
         Inc(J)
      End;
      TouchByte := X;
   End;
End;
```

PASCAL LANGUAGE PULSEWIDTH FOR SYSTEMS USING 8253 AND 8250

```
Procedure PulWidth(X : Word);
{
   This procedure creates a fixed pulse width for programming
   that is approximately independent of system clock speed.  When
   used in an IBM PC or compatible computer operating under MS–
   DOS, X is in units of 0.419 microseconds for values of X
   greater than about 1000.  The procedure can be used with any
   processor having an 8250 UART I/O mapped to base port address
   SPA and an 8253 timer mapped to base port address $40,
   operating with an input clock of 2.386363 MHz and with its
   count limit set to the maximum value.
}
Var
   M, N : Word;
Begin
   Inline($FA);                                  {Turn off interrupts}
   Port[SPA +4] := Port[SPA +4] and $FD;       {Apply Program Pulse on RTS}
   Port[$43] := 0;                               {Freeze value in timer}
   M := Port[$40] shl 8 or Port[$40];            {Read value in timer}
   Repeat                                        {Loop to consume real time}
      Port[$43] := 0;                            {Freeze value in timer again}
      N := Port[$40] shl 8 or Port[$40];         {Read new value in timer}
   Until M – N >= X;                             {See if "X" usec have elapsed}
   Port[SPA +4] := Port[SPA +4] or 2;          {Remove Program Voltage}
   Inline($FB);                                  {Turn interrupts on}
End;
```

VIII. SUMMARY

This application note explains the hardware of different types of 1–Wire interfaces and software examples adapted to this hardware. Depending on the types of iButtons required for a project and the type of computer to be used, the most economic interface is easily found. The hardware examples shown are basically two different types: 5V general interface and 12V RS232 interface. Within the 5V group a common printed circuit board could be used for all four circuits. The variations can be achieved by different population of components (Table 4). The same principle is used for the 12V RS232 interface. The population determines if it is a Read all or a Read/Write all type of interface (Table 5).

There are other possible circuit implementations to create a 1–Wire interface. The circuits described in this application note cover many different configurations. For a custom application, one of the described options can be adapted to meet individual needs. The circuits can be used for reading and writing SRAM based iButtons, for individually programming EPROM based iButtons and for temperature measurement with the DS1920 Temperature iButton. For programming large quantities of EPROM based iButtons with the same data (gang programming) commercial programmers are available from several independent companies. A list of vendors is available from Dallas Semiconductor on request.

PARTS LIST FOR 5V SERIAL TO 1–WIRE PORT ADAPTERS (FOUR OPTIONS) Table 4

POSITION	TTL READ ALL	5V RS232 RD ALL	TTL RW ALL	5V RS232 RW ALL
C1	empty	empty	10µ tantalum	10µ tantalum
C2	empty	empty	150p, ceramic	150p, ceramic
D1	1N5242 (12V)	1N5242 (12V)	1N5242 (12V)	1N5242 (12V)
D2	1N5232 (5.6V)	1N5232 (5.6V)	1N5242 (12V)	1N5242 (12V)
D3	empty	1N5232 (5.6V)	empty	1N5242 (12V) optional
D4	empty	empty	1N5242 (12V)	1N5242 (12V)
D5	empty	empty	ERA–82–004	ERA–82–004
D6	empty	empty	1N5818	1N5818
D7	short	short	empty	ERA–82–004 optional
IC1	empty	empty	LT1109CN8–12	LT1109CN8–12
Q1	2N7000	2N7000	2N7000	2N7000
Q2	short GD	2N7000	short GD	2N7000
Q3	empty	empty	2N7000	2N7000
Q4	empty	empty	2N7000	2N7000
Q5	empty	empty	BSS110	BSS110
R1	4.7 kΩ	4.7 kΩ	4.7 kΩ	4.7 kΩ
R2	empty	4.7 kΩ	empty	4.7 kΩ
R3	empty	empty	100 kΩ	100 kΩ
R4	empty	empty	100 kΩ	100 kΩ
R5	empty	empty	47 kΩ	47 kΩ
L1	empty	empty	see text	see text

PARTS LIST FOR 12V COM PORT TO 1–WIRE ADAPTERS (TWO OPTIONS) Table 5

POSITION	READ ALL	RW ALL
D1	empty	1N5228 (3.9V)
D2	empty	1N5234 (6.2V)
D3	ERA–82–004	ERA–82–004
D4	ERA–82–004	ERA–82–004
D5	1N5228 (3.9V)	1N5235 (6.8V)
D6	1N5234 (6.2V)	1N5242 (12V)
D7	empty	ERA–82–004
Q1	empty	2N7000
Q2	empty	2N7000
Q3	empty	2N7000
Q4	empty	BSS110
R1	1.5 kΩ	1.5 kΩ
R2	empty	1000 kΩ
R3	empty	empty
R4	empty	39 Ω