Inhaltsangabe

Temperaturmessung mit DS1620

1.) Zusammenfassung des Projektergebnisses:

1.1 Kurzbeschreibung in Deutsch:

Es sollen über die parallele Schnittstelle die Daten eines Temperatursensors in den PC eingelesen werden. Über die Software werden diese Daten als gemessene Temperatur am Bildschirm ausgegeben. Weiters soll es möglich sein, mit Hilfe der Software, die gemessene Temperatur auf ein LED – Display auszugeben, welches ebenfalls über die parallele Schnittstelle gesteuert wird. Die LED - Anzeige und der Temperatursensor werden zweckmäßig außerhalb der Hauptplatine angeordnet, um auch höhere Temperaturen erfassen zu können, und um die LED - Anzeige an einer anderen Position anzuordnen.

1.2 Kurzbeschreibung in Englisch:

We will read the data from the temperature sensor over the parallel port into the PC. The measured temperature should be displayed on the PC monitor, and on a LED - display wich is also controlled by the parallel port. The LED - display and the temperature sensor should be connected to the main board by wires, to measure or to display the temperature some meters away from the PC and the main board.

1.3 Erreichte Spezifikationen:

Es wurden alle im Pflichtenheft genannten Spezifikationen erreicht.

1.4 Abweichungen vom Pflichtenheft:

Unser endgültiges Projektergebnis weicht nicht von der Aufgabenstellung im Pflichtenheft ab.

2.) Pflichtenheft:

2.1 Aufgabenstellung:

Die Schaltung soll Temperaturänderungen in 0.5° Schritten erkennen und ausgeben können. Obwohl der Temperatursensor von -55° ... +125° messen kann, werden wir den Bereich auf Temperaturen von -55° ... +99° einschränken, da wir mit unserer LED - Anzeige nur Zahlen bis +99.5° ausgeben können. Die Kommunikation zwischen PC und der Hauptplatine (mit dem Temperatursensor und der LED - Anzeige) erfolgt über die parallele Schnittstelle des PC.

2.2 Technische Daten:

Zur Temperaturmessung verwenden wir den IC DS1620 der Firma DALLAS SEMICONDUCTOR. Dieser IC schickt die Daten seriell als Binärzahl an die parallele Schnittstelle. Zwischen dem Eingang der Schnittstelle und dem Temperatursensor verwenden wir einen 74HCT367 Tristate Buffer, um den Eingang der Schnittstelle, beim senden von Daten an den Sensor, hochohmig zu schalten.
Die eingelesenen Daten werden von der Software am PC Bildschirm ausgegeben, und in einen BCD Kode umgewandelt, welcher über die Schnittstelle wieder ausgegeben wird. Dieser Kode wird dann von zwei 74HCT4511 BCD / 7-Segment Wandlern, welche für jede Stelle ein- / ausschaltbar sind, umgesetzt, und damit die LED - Anzeigen vom Typ HDSP5503 angesteuert.
Die Spannungsversorgung der LED´s und der IC´s übernimmt ein externes Speisegerät. Bei den IC´s verwenden wir noch zusätzliche Stützkondensatoren.

2.3 Ausführungsform:

Für unser Projekt haben wir einstweilen kein Gehäuse vorgesehen. Die Printplatte werden wir doppelseitig ausführen, da dies bei einer digitalen Schaltung leichter zu realisieren ist.

Weiters ist vorgesehen, den Temperatursensor extern auf einer kleinen Platine unterzubringen, um die restlichen Bauteile thermisch nicht zu überlasten.

3.) Projektplanung:

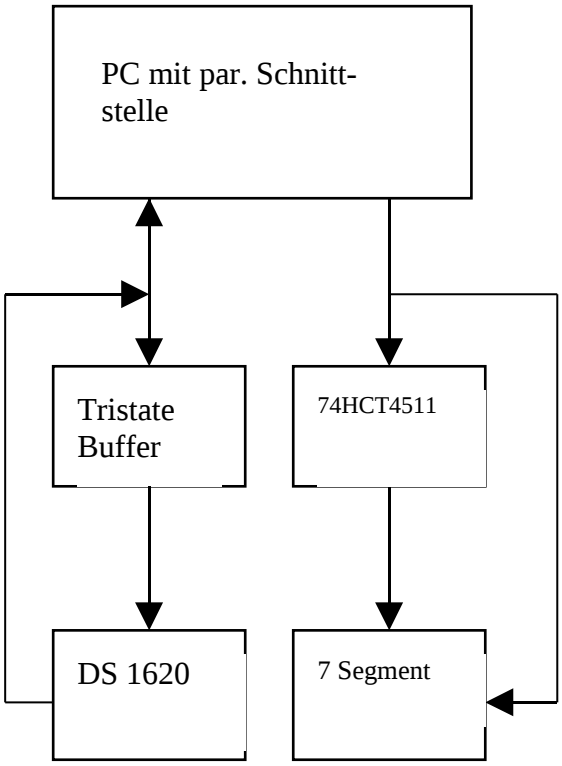
3.1 Auswahl und Begründung des Lösungswegs:

Als erstes wurde uns die Aufgabe gestellt, eine Schaltung zu entwerfen, die mit dem PC über die parallele Schnittstelle kommuniziert.
Unsere erste Idee war, eine komplette Heizungssteuerung zu entwerfen. Auf der Suche nach einem geeigneten Temperatur-sensor stießen wir schließlich auf den DS1620 von Dallas Semiconductors.
Dieser Baustein erweckte unser Interesse, weshalb wir beschlossen, uns näher damit zu beschäftigen, was letztendlich zu unserer Temperaturmessschaltung führte. Die Anzeige über 7 Segment Anzeigen entstand, weil wir uns auch näher mit Schnittstellen-programmierung befassen wollten.

3.2 Funktion der Hardware:

Über die parallele Schnittstelle wird eine 9bit Binärzahl seriell vom DS1620 in den PC eingelesen. Die eingelesene Binärzahl wird dann von der Software umgerechnet und in einen 4bit BCD Kode umgewandelt (z.B.: 19°C = 0001(Kode für 1.LED) & 0101(Kode für zweite LED)), über die parallele Schnittstelle an die BCD / 7-Segment Wandler ausgegeben. An welcher Stelle der Anzeige aktualisiert wird, hängt von einer eigenen
Steuerleitung für die 7-Segment Wandler ab. Mit log. 0 ist der eine, mit log. 1 der andere der 7-Segment Wandler aktiviert. Der Zustand des deaktivierten Wandlers bleibt über einen eingebauten Buffer erhalten.

3.3 Blockschaltbild der Hardware:

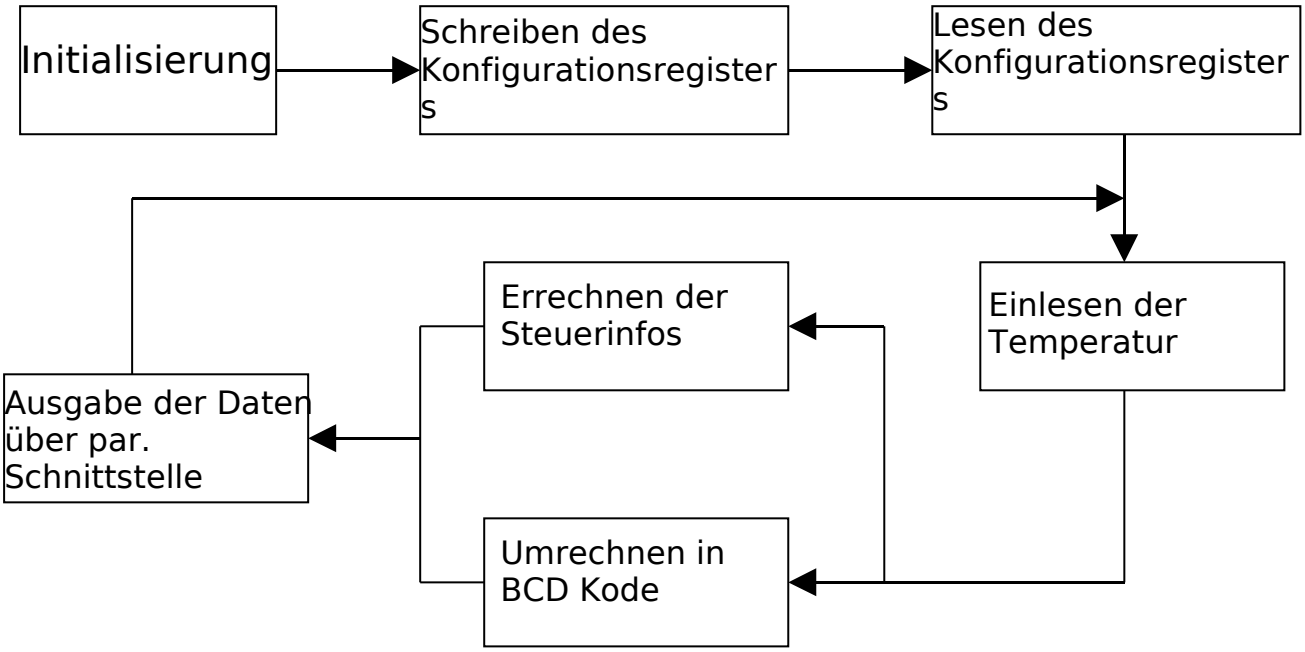


9.) Zeitplan:

	Oktober		November		Dezember		Jänner	
1	Fr		1	Mo		1	Mi	Bauteile bestellen	1	Sa	
2	Sa		2	Di		2	Do		2	So	
3	So		3	Mi	Schaltungsentwurf	3	Fr		3	Mo	
4	Mo		4	Do		4	Sa	Print bestellen	4	Di	
5	Di		5	Fr		5	So		5	Mi	
6	Mi		6	Sa	Simulation	6	Mo		6	Do	
7	Do		7	So		7	Di		7	Fr	
8	Fr		8	Mo		8	Mi		8	Sa	
9	Sa		9	Di		9	Do		9	So	
10	So		10	Mi	Schaltungsentwurf	10	Fr		10	Mo	
11	Mo		11	Do		11	Sa	Dokumentation	11	Di	
12	Di		12	Fr		12	So		12	Mi	Test des Prints
13	Mi	Abgabe Zeitplan	13	Sa	Simulation	13	Mo		13	Do	
14	Do		14	So		14	Di		14	Fr	
15	Fr		15	Mo		15	Mi	Dokumentation	15	Sa	Softwareentwickl.
16	Sa	Info par. Schnittst.	16	Di		16	Do		16	So	
17	So		17	Mi	Schaltungsentwurf	17	Fr		17	Mo	
18	Mo		18	Do		18	Sa	Print bestücken	18	Di	
19	Di		19	Fr		19	So		19	Mi	Software
20	Mi	Datenblätter	20	Sa	Layout in Eagle	20	Mo		20	Do	
21	Do		21	So		21	Di		21	Fr	
22	Fr		22	Mo		22	Mi	Fertigst. des Prints	22	Sa	Test des Projektes
23	Sa	Datenblätter	23	Di		23	Do		23	So	
24	So		24	Mi	Layout in Eagle	24	Fr		24	Mo	
25	Mo		25	Do		25	Sa		25	Di	
26	Di		26	Fr		26	So		26	Mi	Test des Projektes
27	Mi	Datenblätter	27	Sa	Testprint	27	Mo		27	Do	
28	Do		28	So		28	Di		28	Fr	
29	Fr		29	Mo		29	Mi		29	Sa	Abgabe
30	Sa	Datenblätter	30	Di		30	Do		30	So	
31	So					31	Fr		31	Mo	

4.) Softwareentwicklung:

4.1 Struktogramm:



4.2 Verwendete Protokolle:

-Konfiguration schreiben: 0x0C
-Konfiguration: 0x8A
-Konfiguration lesen: 0xAC
-Temperatur lesen: 0xAA
-8 Bit Temp. + 1 Bit Vorzeichen
-Ausgabe von 4 Bit BCD Kode

Serielle Ein- / Ausgabe aller Daten außer BCD Kode

5.) Fertigungsunterlagen:

5.1 Kaufmännische Stückliste:

Anz.	VE	Typ	Best. Nr.	Einzelpreis	Gesamtpreis
1	1	IC DS1620	218-3810	92,00	92,00
2	1	IC 74HCT4511	634-401	10,00	20,00
1	1	IC 74HCT367	652-207	10,00	10,00
1	1	IC SN74LS00N	643-821	6,00	6,00
4	1	7 Segment LED	587-951	19,00	76,00
1	1	D Buchse 25pol.	446-608	56,00	56,00
1	10	Wid. 330R 0.25W	135-825	6,00	6,00
1	10	Wid. 2k2 0.25W	135-881	6,00	6,00
1	5	Schraubklemme 2pol.	426-042	36,00	36,00
1	10	Bannanenstecker rot	444-208	57,00	57,00
1	10	Bannanenstecker blau	444-185	57,00	57,00
1	1	2m Kabel 5 adrig		25,00	25,00
		Versandkostenanteil			40,00

Warenwert	487,00
mit 20% MWST	584,40
Endbetrag	584,40

5.2 Technische Stückliste:

Nr.	Anz.	Typ	Anmerkungen
1	1	IC DS1620	DIL - 8
2	2	IC 74HCT4511	DIL - 16
3	1	IC 74HCT367	DIL - 16
4	1	IC SN74LS00N	DIL - 14
5	4	HDSP5503	12.5x17x8
6	1	ITT 118 D-Buchse 25pol.	25pol. abgewinkelt
7	22	Wid. 330R 0.25W	
8	1	Wid. 2k2 0.25W	
9	3	Schraubklemme 2pol.	
10	1	Bananenstecker rot	4 mm
11	1	Bananenstecket blau	4mm
12	4	Kondensator 22nF	
13	1	Anschlußkabel + Buchse	2 adrig
14	1	Verbindungskabel parallel	5 adrig

-

5.3 Bedienungsanleitung:

1.) Stecken Sie das mitgelieferte 25pol. Verbindungskabel an die parallele Schnittstelle ihres Rechners an (25pol. Buchse Rechnerseitig)

2.) Schließen Sie das Verbindungskabel nun an die 25pol. Buchse, die sich auf der Hauptplatine befindet an.

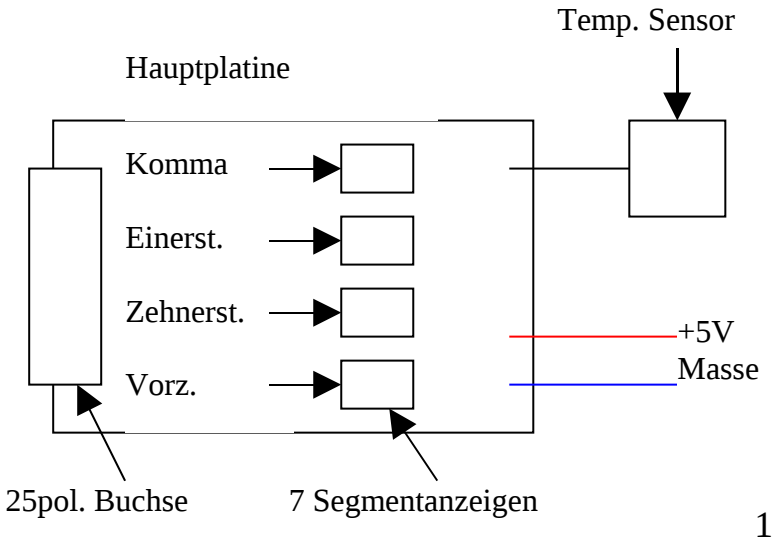
3.) Kopieren Sie nun die Datei TMS2000.EXE in ein von Ihnen gewähltes Verzeichnis.
Erstellen des Verzeichnisses: md <Verzeichnisname>
Kopieren: copy a:\tms2000.exe <Verzeichnis>
Die angegebenen Befehle gelten nur für den MS – DOS Modus.

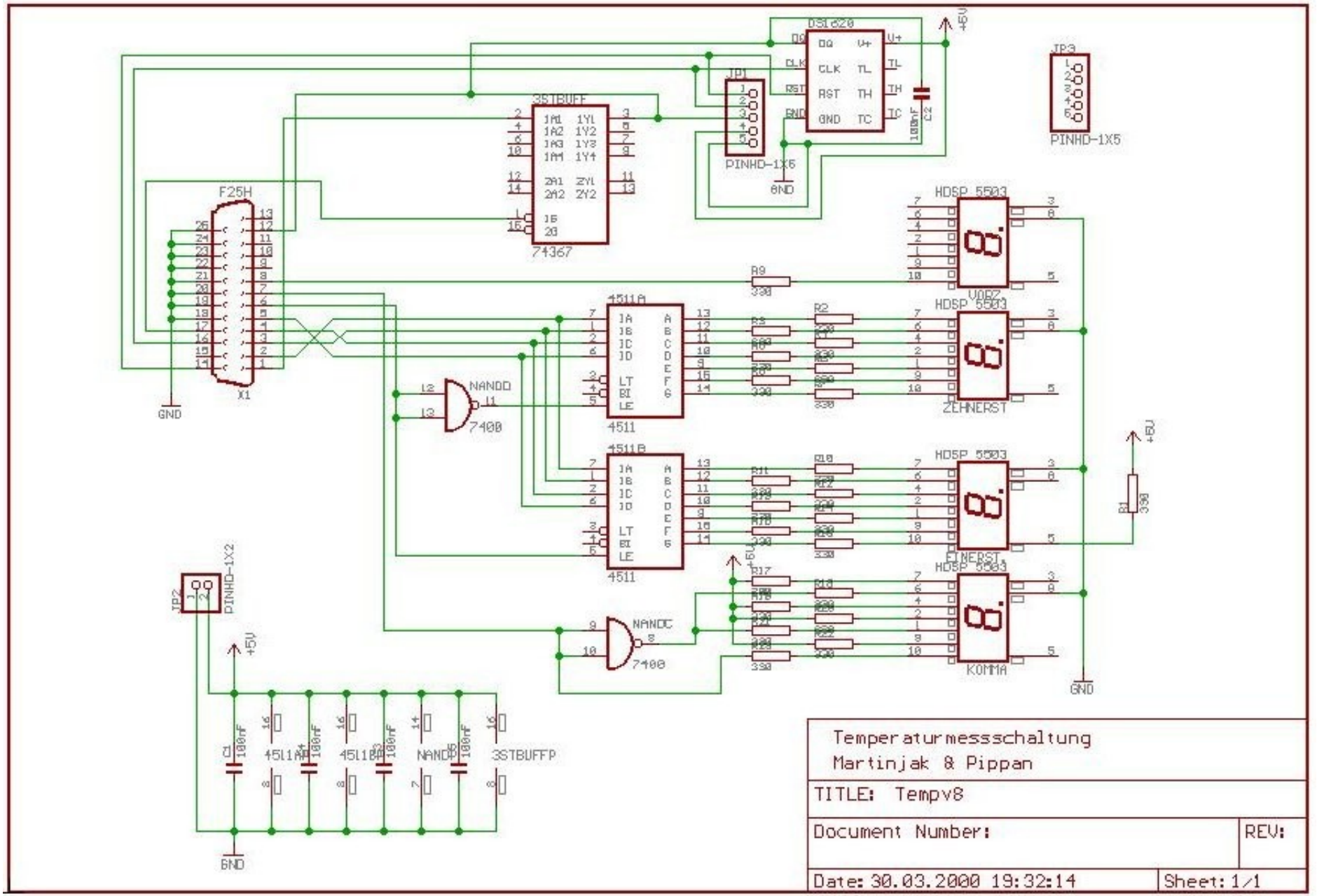
4.) Wechseln Sie in das von ihnen gewählte Verzeichnis und starten Sie das Programm TMS2000 mit der Eingabe von <tms2000.exe> .

5.) Legen Sie die Versorgungsspannung von 5V an.
Roter Bananenstecker: +5V
Blauer Bananenstecker: Masse

6.) Wählen Sie nun im Hauptbildschirm des Programms den Menüpunkt <Temperaturmessung starten>.
Um das Programm zu beenden folgen Sie den Anweisungen am Programmbildschirm.
Falls Sie den Temperatursensor Reseten müssen, wählen Sie den Menüpunkt <Configuration Register schreiben>. Um den Wert des Konfigurationsregisters zu prüfen, wählen Sie den Menüpunkt <Configuration Register lesen>.

7.) Für technischen Support wenden Sie sich bitte an vincent.pippan@i-one.at oder tinaeeeeeeeeeeee@lion.cc .




DS1620
DQ
CLK
RST
GND
V+
TL
TH
TC
+5V
C2
100nF
JP1
PINHD-1X5
JP3
PINHD-1X5
3STBUFF
74367
F25H
X1
GND
HDSP 5503
VORZ
ZEHNERST
EINERST
KOMMA
R9
330
4511A
4511B
4511
NANDD
NANDC
7400
JP2
PINHD-1X2
C1
100nF
45l1A
45l1B
NANDC
3STBUFFP
Temperaturmessschaltung
Martinjak & Pippan
TITLE: Tempv8
Document Number:
REV:
Date: 30.03.2000 19:32:14
Sheet: 1/1

6.) Diskussion:

6.1 Besondere Probleme bei der Realisierung:

Beim Entwurf der Hardware selbst gab es keine Probleme.
Als wir den Testaufbau fertig hatten, mussten wir feststellen, dass unsere Hardware nicht funktioniert, obwohl der Schaltungsentwurf korrekt war.
Bei unseren Messungen stellten wir dann durch Zufall fest, dass die Schaltung korrekt funktioniert, aber nur wenn das Oszilloskop daran angeschlossen ist. Nach einer Reihe von Versuchen stellten wir fest, dass die Schaltung nur funktioniert, wenn wir zwischen irgendeinen Datenpin des DS1620 und GND einen Kondensator schalten. Dies wurde im endgültigen Schaltplan berücksichtigt. Den Grund, weshalb wir diesen Kondensator brauchen, konnten wir leider nicht feststellen.
Ein weiteres kleines Problem ergab sich beim Aufbau der Schaltung auf die Printplatte. Unser Layout stimmte leider nicht mit dem Schaltplan überein, weshalb die Printplatte nachträglich mit Drahtbrücken und unterbrochenen Leiterbahnen korrigiert wurde.

6.2 Verbesserungsmöglichkeit:

Erweiterung der Anzeigesoftware so, dass auch Temperaturen über +99°C angezeigt werden, um den gesamten Messbereich des Temperatursensors auszunützen.

6.3 Vorschläge für weitere Entwicklung:

- Erstellen eines CGI Programms, mit dessen Hilfe die Temperatur auch über das Internet abgerufen werden kann.
  Diese Erweiterung eignet sich besonders gut zur Fernüberwachung der Temperatur von verschiedenen Orten aus, denn man benötigt lediglich einen PC oder Laptop mit Internetanbindung.

- Entwicklung einer Stand Alone Version, bei welcher ein Mikrokontroller die Aufgabe des PC’s übernimmt.

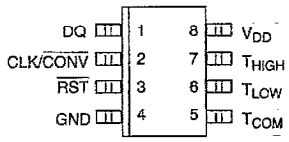
DALLAS SEMICONDUCTOR

DS1620
Digital Thermometer and Thermostat

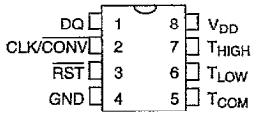
FEATURES

- Requires no external components
- Measures temperatures from –55°C to +125°C in 0.5°C increments. Fahrenheit equivalent is –67°F to 257°F in 0.9°F increments
- Temperature is read as a 9–bit value
- Converts temperature to digital word in 200 ms, typical
- Thermostatic settings are user–definable and non–volatile
- Data is read from/written via a 3–wire serial interface (CLK, DQ, $\overline{RST}$)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system
- 8–pin DIP or SOIC package

PIN ASSIGNMENT



DS1620S
8–PIN SOIC (208 MIL)
See Mech. Drawings Section



DS1620
8–PIN DIP
See Mech. Drawings Section

PIN DESCRIPTION

DQ	– 3–Wire Input/Output
CLK/$\overline{CONV}$	– 3–Wire Clock Input and Standalone Convert Input
$\overline{RST}$	– 3–Wire Reset Input
GND	– Ground
T_{HIGH}	– High Temperature Trigger
T_{LOW}	– Low Temperature Trigger
T_{COM}	– High/Low Combination Trigger
V_{DD}	– Power Supply Voltage (+5V)

DESCRIPTION

The DS1620 Digital Thermometer and Thermostat provides 9–bit temperature readings which indicate the temperature of the device. With three thermal alarm outputs, the DS1620 can also act as a thermostat. T_{HIGH} is driven high if the DS1620's temperature is greater than or equal to a user–defined temperature TH. T_{LOW} is driven high if the DS1620's temperature is less than or equal to a user–defined temperature TL. T_{COM} is driven high when the temperature exceeds TH and stays high until the temperature falls below that of TL.

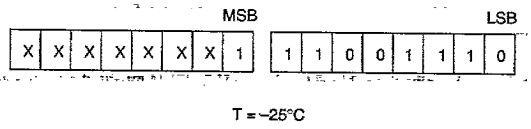
User–defined temperature settings are stored in non–volatile memory, so parts can be programmed prior to insertion in a system, as well as used in standalone applications without a CPU. Temperature settings and temperature readings are all communicated to/from the DS1620 over a simple 3–wire interface.

OPERATION–READING TEMPERATURE

The DS1620 measures temperatures through the use of an onboard, proprietary temperature measurement technique. The temperature reading is provided in a 9–bit, two's complement format. Table 1 describes the exact relationship of output data to measured temperature. The data is transmitted serially through the 3–wire serial interface, LSB first. The DS1620 can measure temperature over the range of –55°C to +125°C in 0.5°C increments. For Fahrenheit usage, a lookup table or conversion factor must be used.

Since data is transmitted over the 3–wire bus LSB first, temperature data can be written to/read from the DS1620 as either a 9–bit word (taking $\overline{RST}$ low after the 9th (MSB) bit), or as two transfers of 8–bit words, with the most significant 7 bits being ignored or set to zero, as illustrated in Table 1. After the MSB, the DS1620 will output 0's.

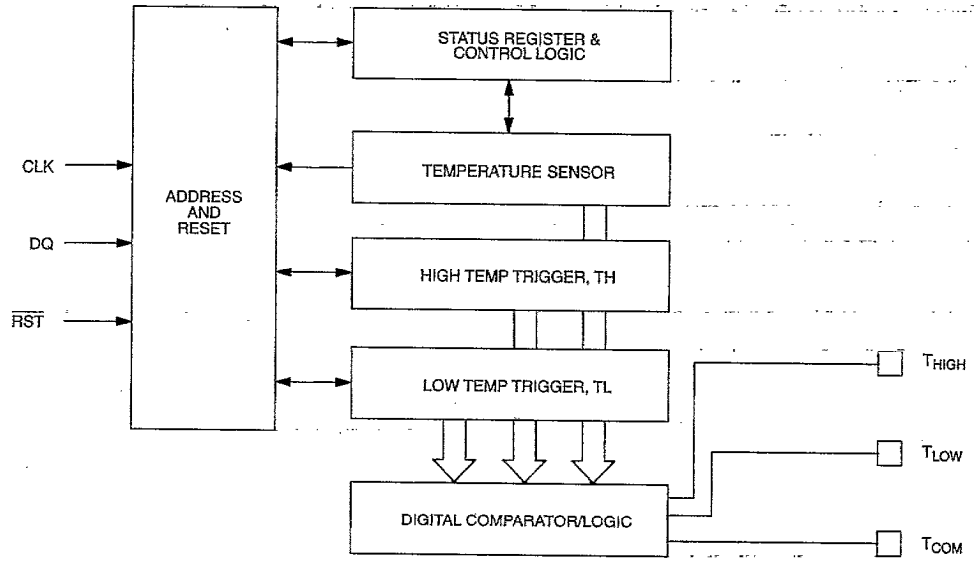
Note that temperature is represented in the DS1620 in terms of a $^1/_2$°C LSB, yielding the following 9–bit format:



TEMPERATURE/DATA RELATIONSHIPS Table 1

TEMP	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0 11111010	00FA
+25°C	0 00110010	0032h
$^1/_2$°C	0 00000001	0001h
0°C	0 00000000	0000h
–$^1/_2$°C	1 11111111	01FFh
–25°C	1 11001110	01CEh
–55°C	1 10010010	0192h

DS1620 FUNCTIONAL BLOCK DIAGRAM Figure 1



10

DETAILED PIN DESCRIPTION Table 2

PIN	SYMBOL	DESCRIPTION
1	DQ	**Data Input/Output pin** for 3–wire communication port.
2	CLK/$\overline{CONV}$	**Clock input pin** for 3–wire communication port. When the DS1620 is used in a standalone application with no 3–wire port, this pin can be used as a convert pin. Temperature conversion will begin on the falling edge of $\overline{CONV}$.
3	$\overline{RST}$	**Reset input pin** for 3–wire communication port.
4	GND	**Ground pin.**
5	T_{COM}	**High/Low Combination Trigger.** Goes high when temperature exceeds TH; will reset to low when temperature falls below TL.
6	T_{LOW}	**Low Temperature Trigger.** Goes high when temperature falls below TL.
7	T_{HIGH}	**High Temperature Trigger.** Goes high when temperature exceeds TH.
8	V_{DD}	**Supply Voltage.** 5V input power pin.

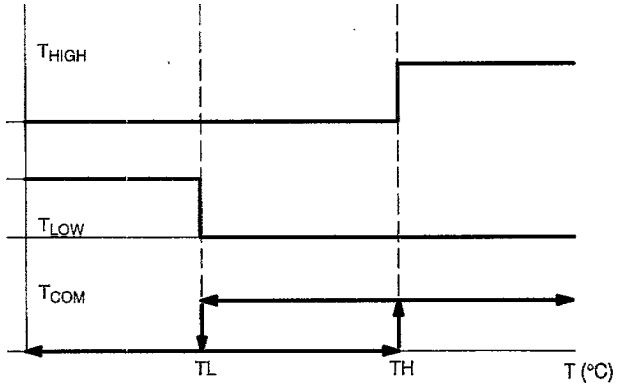
OPERATION–THERMOSTAT CONTROLS

Three thermally triggered outputs, T_{HIGH}, T_{LOW}, and T_{COM}, are provided to allow the DS1620 to be used as a thermostat, as shown in Figure 2. When the DS1620's temperature meets or exceeds the value stored in the high temperature trip register, the output T_{HIGH} becomes active (high) and remains active until the DS1620's measured temperature becomes less than the stored value in the high temperature register, TH. The T_{HIGH} output can be used to indicate that a high temperature tolerance boundary has been met or exceeded, or as part of a closed loop system can be used to activate a cooling system and to deactivate it when the system temperature returns to tolerance.

The T_{LOW} output functions similarly to the T_{HIGH} output. When the DS1620's measured temperature equals or falls below the value stored in the low temperature register, the T_{LOW} output becomes active. T_{LOW} remains active until the DS1620's temperature becomes greater than the value stored in the low temperature register, TL. The T_{LOW} output can be used to indicate that a low temperature tolerance boundary has been met or exceeded, or as part of a closed loop system, can be used to activate a heating system and to deactivate it when the system temperature returns to tolerance.

The T_{COM} output goes high when the measured temperature meets or exceeds TH, and will stay high until the temperature equals or falls below TL. In this way, any amount of hysteresis can be obtained.

THERMOSTAT OUTPUT OPERATION Figure 2



OPERATION AND CONTROL

The DS1620 must have temperature settings resident in the TH and TL registers for thermostatic operation. A configuration/status register is also used to determine the method of operation that the DS1620 will use in a particular application, as well as indicating the status of the temperature conversion operation. The configuration register is defined as follows:

CONFIGURATION/STATUS REGISTER

DONE	THF	TLF	NVB	1	0	CPU	1SHOT

where

DONE = Conversion Done bit. 1=conversion complete, 0=conversion in progress.

THF = Temperature High Flag. This bit will be set to 1 when the temperature is greater than or equal to the value of TH. It will remain 1 until reset by writing 0 into this location or by removing power from the device. This feature provides a method of determining if the DS1620 has ever been subjected to temperatures above TH while power has been applied.

TLF = Temperature Low Flag. This bit will be set to 1 when the temperature is less than or equal to the value of TL. It will remain 1 until reset by writing 0 into this location or by removing power from the device. This feature provides a method of determining if the DS1620 has ever been subjected to temperatures below TL while power has been applied.

NVB = Nonvolatile Memory Busy Flag. 1=write to an E^2 memory cell in progress. 0=nonvolatile memory is not busy. A write to E^2 may take up to 10 ms.

CPU = CPU use bit. If CPU=0, the CLK/$\overline{CONV}$ pin acts as a conversion start control, when $\overline{RST}$ is low. If CPU is 1, the DS1620 will be used with a CPU communicating to it over the 3–wire port, and the operation of the CLK/$\overline{CONV}$ pin is as a normal clock in concert with DQ and $\overline{RST}$. This bit is stored in nonvolatile E^2 memory, capable of at least 50,000 writes.

1SHOT = One–Shot Mode. If 1SHOT is 1, the DS1620 will perform one temperature conversion upon reception of the Start Convert T protocol. If 1SHOT is 0, the DS1620 will continuously perform temperature conversion. This bit is stored in nonvolatile E^2 memory, capable of at least 50,000 writes.

For typical thermostat operation, the DS1620 will operate in continuous mode. However, for applications where only one reading is needed at certain times, and to conserve power, the one–shot mode may be used. Note that the thermostat outputs (T_{HIGH}, T_{LOW}, T_{COM}) will remain in the state they were in after the last valid temperature conversion cycle when operating in one–shot mode.

OPERATION IN STANDALONE MODE

In applications where the DS1620 is used as a simple thermostat, no CPU is required. Since the temperature limits are nonvolatile, the DS1620 can be programmed prior to insertion in the system. In order to facilitate operation without a CPU, the CLK/$\overline{CONV}$ pin (pin 2) can be used to initiate conversions. Note that the CPU bit must be set to 0 in the configuration register to use this mode of operation.

To use the CLK/$\overline{CONV}$ pin to initiate conversions, $\overline{RST}$ must be low and CLK/$\overline{CONV}$ must be high. If CLK/$\overline{CONV}$ is driven low and then brought high in less than 10 ms, one temperature conversion will be performed and then the DS1620 will return to an idle state. If CLK/$\overline{CONV}$ is driven low and remains low, continuous conversions will take place until CLK/$\overline{CONV}$ is brought high again. With the CPU bit set to 0, the CLK/$\overline{CONV}$ will override the 1–shot bit if it is equal to 1. This means that even if the part is set for one–shot mode, driving CLK/$\overline{CONV}$ low will initiate conversions.

3–WIRE COMMUNICATIONS

The 3–wire bus is comprised of three signals. These are the $\overline{RST}$ (reset) signal, the CLK (clock) signal, and the DQ (data) signal. All data transfers are initiated by driving the $\overline{RST}$ input high. Driving the $\overline{RST}$ input low terminates communication. (See Figures 3 and 4.) A clock cycle is a sequence of a falling edge followed by a rising edge. For data inputs, the data must be valid during the rising edge of a clock cycle. Data bits are output on the falling edge of the clock, and remain valid through the rising edge.

10

When reading data from the DS1620, the DQ pin goes to a high impedance state while the clock is high. Taking $\overline{RST}$ low will terminate any communication and cause the DQ pin to go to a high impedance state.

Data over the 3–wire interface is communicated LSB first. The command set for the 3–wire interface as shown in Table 3 is as follows; only these protocols should be written to the DS1620, as writing other protocols to the device may result in permanent damage to the part.

Read Temperature [AAh]
This command reads the contents of the register which contains the last temperature conversion result. The next nine clock cycles will output the contents of this register.

Write TH [01h]
This command writes to the TH (HIGH TEMPERATURE) register. After issuing this command, the next nine clock cycles clock in the 9–bit temperature limit which will set the threshold for operation of the T_{HIGH} output.

Write TL [02h]
This command writes to the TL (LOW TEMPERATURE) register. After issuing this command, the next nine clock cycles clock in the 9–bit temperature limit which will set the threshold for operation of the T_{LOW} output.

Read TH [A1h]
This command reads the value of the TH (HIGH TEMPERATURE) register. After issuing this command, the next nine clock cycles clock out the 9–bit temperature limit which sets the threshold for operation of the T_{HIGH} output.

Read TL [A2h]
This command reads the value of the TL (LOW TEMPERATURE) register. After issuing this command, the next nine clock cycles clock out the 9–bit temperature limit which sets the threshold for operation of the T_{LOW} output.

Start Convert T [EEh]
This command begins a temperature conversion. No further data is required. In one–shot mode, the temperature conversion will be performed and then the DS1620 will remain idle. In continuous mode, this command will initiate continuous conversions.

Stop Convert T [22h]
This command stops temperature conversion. No further data is required. This command may be used to halt a DS1620 in continuous conversion mode. After issuing this command, the current temperature measurement will be completed, and then the DS1620 will remain idle until a Start Convert T is issued to resume continuous operation.

Write Config [0Ch]
This command writes to the configuration register. After issuing this command, the next eight clock cycles clock in the value of the configuration register.

Read Config [ACh]
This command reads the value in the configuration register. After issuing this command, the next eight clock cycles output the value of the configuration register.

DS1620 COMMAND SET Table 3

INSTRUCTION	DESCRIPTION	PROTOCOL	3–WIRE BUS DATA AFTER ISSUING PROTOCOL	NOTES
TEMPERATURE CONVERSION COMMANDS				
Read Temperature	Reads last converted temperature value from temperature register.	AAh	<read data>	
Start Convert T	Initiates temperature conversion.	EEh	idle	1
Stop Convert T	Halts temperature conversion.	22h	idle	1
THERMOSTAT COMMANDS				
Write TH	Writes high temperature limit value into TH register.	01h	<write data>	2
Write TL	Writes low temperature limit value into TL register.	02h	<write data>	2
Read TH	Reads stored value of high temperature limit from TH register.	A1h	<read data>	2
Read TL	Reads stored value of low temperature limit from TL register.	A2h	<read data>	2
Write Config	Writes configuration data to configuration register.	0Ch	<write data>	2
Read Config	Reads configuration data from configuration register.	ACh	<read data>	2

NOTES:

1. In continuous conversion mode, a Stop Convert T command will halt continuous conversion. To restart, the Start Convert T command must be issued. In one–shot mode, a Start Convert T command must be issued for every temperature reading desired.
2. Writing to the E^2 typically requires 10 ms at room temperature. After issuing a write command, no further writes should be requested for at least 10 ms.

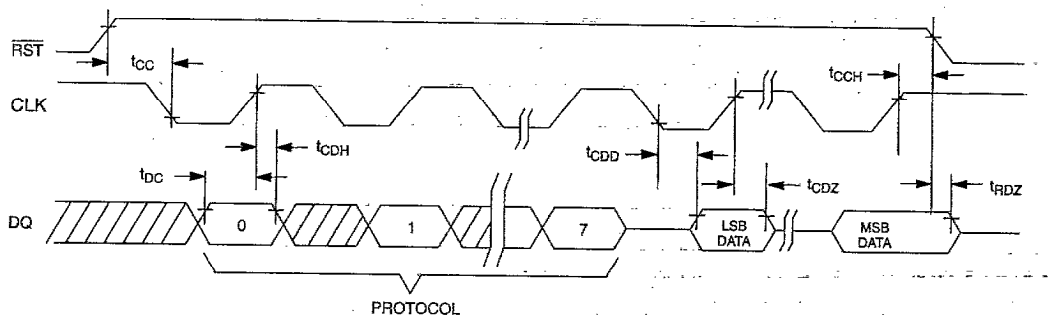
10

FUNCTION EXAMPLE

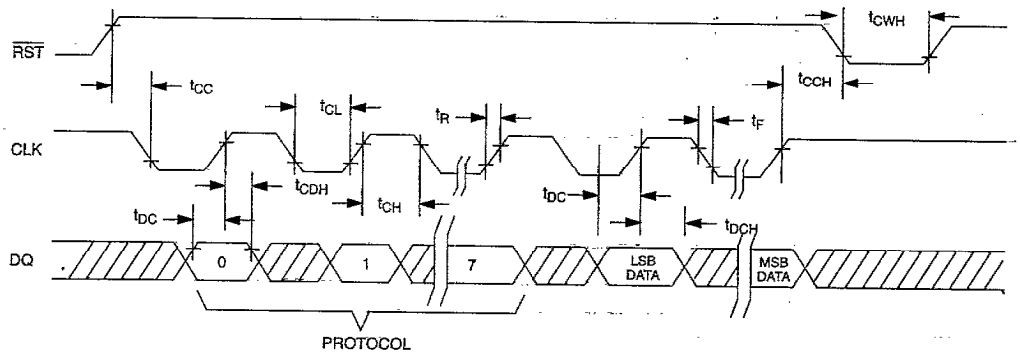
Example: CPU sets up DS1620 for continuous conversion and thermostatic function.

CPU MODE	DS1620 MODE (3–WIRE)	DATA (LSB FIRST)	COMMENTS
TX	RX	0Ch	CPU issues Write Config command.
TX	RX	02h	CPU sets DS1620 up for continuous conversion.
TX	RX	Toggle $\overline{RST}$	CPU issues Reset to DS1620.
TX	RX	01h	CPU issues Write TH command.
TX	RX	0050h	CPU sends data for TH limit of +40°C.
TX	RX	Toggle $\overline{RST}$	CPU issues Reset to DS1620.
TX	RX	02h	CPU issues Write TL command.
TX	RX	0014h	CPU sends data for TL limit of +10°C.
TX	RX	Toggle $\overline{RST}$	CPU issues Reset to DS1620.
TX	RX	A1h	CPU issues Read TH command.
RX	TX	0050h	DS1620 sends back stored value of TH for CPU to verify.
TX	RX	Toggle $\overline{RST}$	CPU issues Reset to DS1620.
TX	RX	A2h	CPU issues Read TL command.
RX	TX	0014h	DS1620 sends back stored value of TL for CPU to verify.
TX	RX	Toggle $\overline{RST}$	CPU issues Reset to DS1620.
TX	RX	EEh	CPU issues Start Convert T command.
TX	RX	Toggle $\overline{RST}$	CPU issues Reset to DS1620.

READ DATA TRANSFER Figure 3



WRITE DATA TRANSFER Figure 4



NOTE: t_{CL}, t_{CH}, t_R, and t_F apply to both read and write data transfer.

10

ABSOLUTE MAXIMUM RATINGS*

Voltage on Any Pin Relative to Ground	–0.5V to +7.0V
Operating Temperature	–55°C to +125°C
Storage Temperature	–55°C to +125°C
Soldering Temperature	260°C for 10 seconds

* This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

RECOMMENDED DC OPERATING CONDITIONS

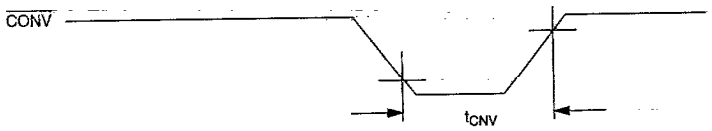
PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Supply	V_{DD}	4.5	5.0	5.5	V	1
Logic 1	V_{IH}	2.0		V_{CC}+0.3	V	1
Logic 0	V_{IL}	–0.3		+0.8	V	1

DC ELECTRICAL CHARACTERISTICS

(–55°C to +125°C; V_{DD}=4.5V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Thermometer Error	T_{ERR}	0°C to +70°C			±0.5	°C	10, 11
		–55°C to +0°C and 70°C to 125°C	See Typical Curve				
Long-term Stability		105°C, 1000 hours		±0.1		°C	
Logic 0 Output	V_{OL}				0.4	V	3
Logic 1 Output	V_{OH}		2.4			V	2
Input Resistance	R_I	DQ,$\overline{RST}$ to GND, CLK to V_{DD}	1 1			MΩ MΩ	
Active Supply Current	I_{CC}	0°C to +70°C			1	mA	4, 5
Standby Supply Current	I_{STBY}	0°C to +70°C			1	µA	4, 5

SINGLE CONVERT TIMING DIAGRAM (STAND–ALONE MODE)



AC ELECTRICAL CHARACTERISTICS

(–55°C to +125°C; V_{DD}=4.5V to 5.5V)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	T_{TC}		200	500	ms	
Data to CLK Setup	t_{DC}	35			ns	6
CLK to Data Hold	t_{CDH}	40			ns	6
CLK to Data Delay	t_{CDD}			100	ns	6, 7, 8
CLK Low Time	t_{CL}	285			ns	6
CLK High Time	t_{CH}	285			ns	6
CLK Frequency	f_{CLK}	DC		1.75	MHz	6
CLK Rise and Fall	t_R, t_F			500	ns	
$\overline{RST}$ to CLK Setup	t_{CC}	100			ns	6
CLK to $\overline{RST}$ Hold	t_{CCH}	40			ns	6
$\overline{RST}$ Inactive Time	t_{CWH}	125			ns	6, 9
CLK High to I/O High Z	t_{CDZ}			50	ns	6
$\overline{RST}$ Low to I/O High Z	t_{RDZ}			50	ns	6
Convert Pulse Width	t_{CNV}	250 ns		500 ms		
NV Write Cycle Time	t_{WR}		10	50	ms	

AC ELECTRICAL CHARACTERISTICS

(–55°C to +125°C; V_{DD}=4.5V to 5.5V)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Input Capacitance	C_I		5		pF	
I/O Capacitance	$C_{I/O}$		10		pF	

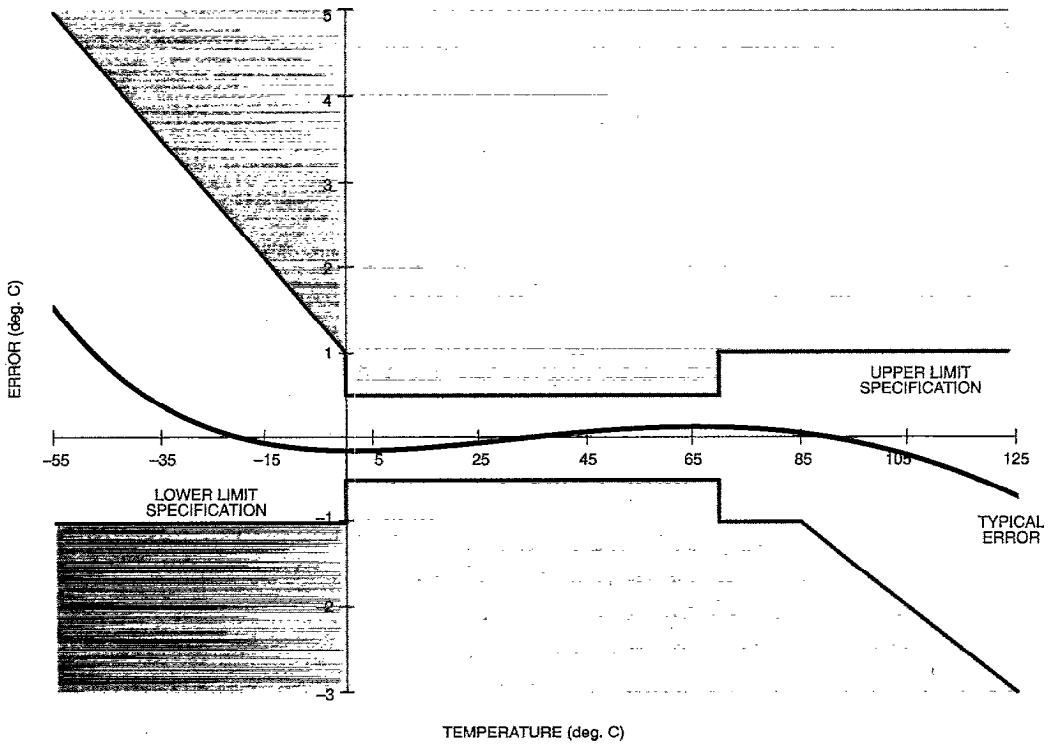
NOTES:

1. All voltages are referenced to ground.
2. Logic one voltages are specified at a source current of 1 mA.
3. Logic zero voltages are specified at a sink current of 4 mA.
4. I_{CC} specified with DQ pin open.
5. I_{CC} specified with V_{CC} at 5.0V and $\overline{RST}$=GND.
6. Measured at V_{IH} = 2.0V or V_{IL} = 0.8V.
7. Measured at V_{OH} = 2.4V or V_{OL} = 0.4V.
8. Load capacitance = 50 pF.
9. t_{CWH} must be 10 ms minimum following any write command that involves the E^2 memory.
10. See Figure 5 for specification limits outside 0°C to 70°C range.
11. Thermometer error reflects temperature accuracy as tested during calibration.

10

TYPICAL PERFORMANCE CURVE Figure 5

DS1620 DIGITAL THERMOMETER AND THERMOSTAT
TEMPERATURE READING ERROR



```
/*

 ********************************************************************************
 ********************************************************************************
 ***************                                                ***************
 ***************      Testprogramm zur Tenperaturmessung        ***************
 ***************       (Z)opyright by Martinjak & Pippan        ***************
 ***************         Version 2005 erweitert für Siebens.    ***************
 ***************         23.02.2000 (funktioniert)              ***************
 ***************                                                ***************
 ********************************************************************************
 ********************************************************************************

*/

 #include <stdio.h>
 #include <conio.h>
 #include <dos.h>
 #include <math.h>
 #include <graphics.h>
 #include <stdlib.h>

 #define    ws              8  // ms Waitstate
 #define    basis        0x378  // I/O Portadress
 #define    Data       basis+0
 #define    Status     basis+1
 #define    Control    basis+2

 #define    o               8
 #define    n               9

class INITDS
{
 protected:
        unsigned char out;
        unsigned int bin;
        float  in;
        int a,Vorzeichen;

 public:
        INITDS ();
        ~INITDS();

        void Rahmen( int, int, int, int );
        void SetKastl( int, int, unsigned char, int);
        void Kastl( int, int, int );

        void Import_Temperatur ( int, int, int, int );
        void Temperatur_Ausgabe( int, int );
        void Feld_Bin ( unsigned char );
        void Send_Protocol ( int, int, unsigned char, int, int );
        void Write_Port ( unsigned int, unsigned char );
        void LCD_Ausgabe( float );
        void Menu(int , int , char*, int, int );
        float int_to_double( unsigned int );
        void Minusgrade( unsigned int );
};


void main()
{
```

```
INITDS t;
char wahl;
char zeichen[80];
int x=100, y=60;

//************************* MENU ****************************
do
{
 outportb(Data,0xFF);  //INIT Data Port
 outportb(Data,0x00);
 outportb(Status,0x00);//INIT Status Port

/******  Auswahlmenü (Startbildschirm)  ********/

 cleardevice();
 setcolor( BLUE );
 sprintf  ( zeichen, "Testprogramm zur Temperaturmessung");
 outtextxy( x, y, zeichen );

 sprintf  ( zeichen, "TMS 2000 Version 1.x");
 outtextxy( x+20, y+10, zeichen );

 sprintf  ( zeichen, "(Z)opyright by Martinjak und Pippan");
 outtextxy( x, y+20, zeichen );

 sprintf  ( zeichen, "[R] Lesen der Temperatur");
 t.Menu( x+20, y+100, zeichen, 350, 20);

 sprintf  ( zeichen, "[C] schreiben des Configuration Register");
 t.Menu( x+20, y+120, zeichen, 350, 20);

 sprintf  ( zeichen, "[A] Lesen des Configuration Register");
 t.Menu( x+20, y+140, zeichen, 350, 20);

 sprintf  ( zeichen, "[O] Schreiben auf LCD und messen");
 t.Menu( x+20, y+160, zeichen, 350, 20);

 sprintf  ( zeichen,"Beenden mit [ESC] " );
 t.Menu( x+20, y+180, zeichen, 350, 20);

 setcolor(RED);
 sprintf  ( zeichen, "Thanks for using our Temperature Messurement");
 outtextxy( x, y+300, zeichen);

 //************************* CASES ****************************

 wahl=getch();
 switch(wahl)
 {
  case 'r': {     //************* Case Read Temperature *****************
                do
                {
                 cleardevice();
                 setcolor( RED );

                 outtextxy(150,30,"Temperaturmessung mit dem PC ");
                 outtextxy(170,40,"Temperatur einlesen");
                 outtextxy(150,50,"(Z)opyright by Martinjak & Pippan ");

                 t.Write_Port(Control,0x0F); //CLK auf HIGH Der REST auf NULL
                 delay( ws );
```

```
                t.Write_Port(Control,0x0D); //drive RST high

                //**************** Start Convert ******************

                t.Send_Protocol( 180, 120, 0xEE, o, 1 );

                //**************** Toggle RST **********************

                t.Write_Port( Control,0x0D );
                outportb( Control,0x0F );
                delay( 12 );
                t.Write_Port( Control,0x0D );

                //************ Send Read Temperature Protocol ********

                t.Send_Protocol( 180, 220, 0xAA, o, 1); //sends the Protocol 0xAA

                //*************** Import Temperature ****************

                t.Import_Temperatur( 180, 320, n, 1);   //Import the Temperature from
the DS1620
                setcolor( BLUE );

                //*************** Temperature Output ****************

                t.Temperatur_Ausgabe( 200, 400);

                t.Write_Port( Control,0x0D );
                outportb( Control,0x0F );
                delay( 12 );
                t.Write_Port( Control,0x0D );

                outtextxy( 150, 450, "Programm beenden mit [ESC]");

               }while(getch() != 27 );
               break;
               }

        case 'o':{
                char buf[40];

                cleardevice();
                //************** Case Read Temperature *****************
                do
                {
                 setcolor( RED );

                 outtextxy(150,30,"Temperaturmessung mit dem PC ");
                 outtextxy(170,40,"Temperatur einlesen");
                 outtextxy(150,50,"(Z)opyright by Martinjak & Pippan ");

                 t.Write_Port( Control,0x0F); //CLK auf HIGH
                 delay( ws );
                 t.Write_Port(Control,0x0D); //drive RST high

                //**************** Start Convert ******************

                 t.Send_Protocol( 400, 120, 0xEE, o, 1 ); //
(x,y,Protocol,length,on/off);18i?#?#

                //**************** Toggle RST **********************
```

```
                  t.Write_Port( Control, 0x0D );
                  outportb( Control,0x0F );
                  delay( 12 );
                  t.Write_Port( Control,0x0D );

                 //************ Send Read Temperature Protocol ********

                  t.Send_Protocol( 400, 220, 0xAA, o, 1);

                 //*************** Import Temperature ****************

                  t.Import_Temperatur( 400, 320, n, 1);   //Import the Temperature from
the DS1620

                  setcolor( YELLOW );

                  t.Temperatur_Ausgabe(100,100);//+i*20);

                 //*************** Temperature Output ****************

                  t.Write_Port( Control,0x0D );
                  outportb( Control,0x0F );
                  delay( 12 );
                  t.Write_Port( Control,0x0D );

                  outtextxy( 150, 450, "Programm beenden mit [Q]uit");
                  }while(!kbhit());
                  break;
                }

   case 'c': {   //************** Case Configuration Register **********

               cleardevice();
               setcolor( RED );

               outtextxy(180,30,"Temperaturmessung mit dem PC ");
               outtextxy(200,40,"Configurationsregister schreiben");
               outtextxy(180,50,"(Z)opyright by Martinjak & Pippan ");

               t.Write_Port( Control,0x0F); //CLK auf HIGH Der REST auf NULL
               delay( ws );
               t.Write_Port(Control,0x0D); //drive RST high

               t.Send_Protocol( 180, 120, 0x0C, o, 1 );
               t.Send_Protocol( 180, 220, 0x8A, o, 1 );

               t.Write_Port( Control,0x0D );
               outportb( Control,0x0F );
               delay( 15 );
               t.Write_Port( Control,0x0D );

               outtextxy( 150, 450, "Zurück durch Tastendruck");
               getch();

               break;
            }

   case 'a': {     //************* Case Read Temperature *****************
                do
                {
```

```
                    cleardevice();
                    setcolor( RED );

                    outtextxy(150,30,"Temperaturmessung mit dem PC ");
                    outtextxy(170,40,"Configuration Register einlesen");
                    outtextxy(150,50,"(Z)opyright by Martinjak & Pippan ");

                    t.Write_Port( Control,0x0F); //CLK auf HIGH Der REST auf NULL
                    delay( ws );
                    t.Write_Port(Control,0x0D); //drive RST high

                    //************ Send Read Configuration Protocol ********

                    t.Send_Protocol( 180, 120, 0xAC, o, 1);

                    //*************** Import Temperature ****************

                    t.Import_Temperatur( 180, 220, n, 1);   //Import the Temperature from
the DS1620
                    setcolor( BLUE );

                    t.Write_Port( Control,0x0D );
                    outportb( Control,0x0F );
                    delay( 12 );
                    t.Write_Port( Control,0x0D );

                    outtextxy( 150, 450, "Programm beenden mit [ESC]");

                   }while(getch() != 27 );
                   break;
              }
   }
 }while(getch() != 27);
}

 //********************** Kon- Destruktor *************************

INITDS::INITDS()
{
 int modus, karte = DETECT;
 initgraph( &karte, &modus,"g:\\dosprog\\borlandc\\bgi"); //"c:\\tc\\bgi" );

 cleardevice();
 setbkcolor( 7 );
 setcolor(RED);
}

INITDS::~INITDS()
{
 getch();
 closegraph();
}

//******************* Send Protocol Function ************************

void INITDS::Send_Protocol( int x, int y, unsigned char z, int d, int set)
{
 //***************************** set BIT 0-7 **************************
 unsigned char p, u, r;
 char buf[80];
```

```
 setcolor( BLUE );
 if(set==1)
 {
  sprintf( buf, "Send Protocol 0x%x ", z);
  outtextxy( x-40 ,y-30,buf );
  Rahmen( x, y, WHITE ,d );
 }

 Write_Port( Control,0x09 ); // drive CLK low

 for( int i=0; i<o; i++ )
        {
         u=(~((z>>i) & 0x01))&0x01;

         p = ( 0x08 |  u  );  // CLK low
         Write_Port( Control, p );        // writes Protocol

         if(set==1) SetKastl( x, y,(~p) & 0x01, i);  // put a Kastl
         Write_Port( Control, u );

         Write_Port( Control,0x0D ); // drive CLK high
         Write_Port( Control,0x09 ); // drive CLK low

        }
  //********************** PROTOCOL END ***************************

}

//************************** Import Temperature ***********************

void INITDS::Import_Temperatur(int x, int y, int d, int set)
{
 int i;
 unsigned int p=0,r=0,u;

 setcolor( BLUE );

 if(set==1)
 {
  Rahmen( x, y, WHITE, d);
  setcolor(BLUE);
  outtextxy(x-40 ,y-30,"Import Temperatur" );
 }
 Write_Port( Control, 0x01 );//0x01

 if(set==1) Rahmen( x, y, WHITE, d);

 for( i=0; i<n; i++ )
        {
          p = inportb( Status ) & 0x20;  //reads the contents of the Port

          Write_Port( Control, 0x01 );   //0x01 clk low

          u = (( p >> 5 ) & 0x01);

          if(set==1)SetKastl( x, y, ( u ), i );    // put a Kastl

          Write_Port( Control, 0x05 );   //clk high
          Write_Port( Control, 0x05 );   // drive CLK high
          Write_Port( Control, 0x01 );   // drive CLK low
```

```
          r = r | (u<<i);
        }

 Minusgrade(r);
 }

void INITDS::Menu(int x, int y, char *zeichen, int xm, int ym)
{
 setcolor(BLACK);
 setfillstyle( 1, GREEN );
 bar(x+4,y+4,x+4+xm,y+ym+4);

 setcolor(WHITE);
 setfillstyle( 1, WHITE );
 bar(x,y,x+xm,y+ym);
 setcolor( GREEN );
 outtextxy(x+6,y+6,zeichen);
}

void INITDS::Minusgrade( unsigned int r)
{
 unsigned int b;
 int t;
 b=(r >> 8) & 0x01;
 if(b==1)
 {
  r=(((~r)+1) & 0x00FF);
  in=int_to_double( r );
  in*=-1;
  Vorzeichen=0x40;
 }
 else
 {
  in=int_to_double( r );   //unsigned int to float
  Vorzeichen=0x00;
 }
 bin=in;
}

float INITDS::int_to_double( unsigned int Value)
{
 float Tmp;
 Tmp=(( float )( Value & 0x00FF ))/2;
 return Tmp;
}

void INITDS::Write_Port( unsigned int Port ,unsigned char i )
{
 outportb( Port, i );//| 0x2);
 delay( ws );
}

void INITDS::Temperatur_Ausgabe( int x, int y)
{
 char buf[40];

 setcolor( 7 );
 setfillstyle( 1, 7 );
 bar(x-80,y+40,x+220,y+250);

 setcolor(YELLOW);
```

```
 settextstyle( 10, 0, 5 );
 sprintf( buf, "%3.1f?C", in );
 outtextxy( x-25, y+50, buf );

 LCD_Ausgabe( in );

 settextstyle( 7, 0, 3 );
 sprintf( buf, "gemessene Temperatur:", in );
 outtextxy( x-70, y, buf );

 settextstyle( 0, 0, 1);

}

void INITDS::Rahmen(int x, int y, int c, int e)  //macht eine schönen weisen Rahmen
{
 setcolor( c );
 setfillstyle( 1, c );
 bar( x, y, x+e*20+5, y+40 );

 outtextxy( x, y-15, "LSB" );
 outtextxy( x+e*20-20, y-15, "MSB" );
 outtextxy( x-40, y+10, "WERT" );
 outtextxy( x-40, y+23, "BIT" );
}

void INITDS::SetKastl( int x, int y, unsigned char p, int i)
{//setzt die Bits
 int c;
 char zeichen[80];

 if( p==1 )
 {
  c=GREEN;

  Kastl( x+5+i*20, y+5, c );

  setcolor( WHITE );
  outtextxy( x+10+i*20, y+10, "1" );

  sprintf( zeichen,"%i",i );
  outtextxy( x+10+i*20, y+23, zeichen );
 }

 if( p==0 )
 {
  c=RED;

  Kastl( x+5+i*20, y+5, c );

  setcolor( WHITE );
  outtextxy( x+10+i*20, y+10, "0" );

  sprintf( zeichen,"%i",i );
  outtextxy( x+10+i*20, y+23, zeichen );

 }
}

void INITDS::Kastl( int x, int y, int c)
```

```
{
 setfillstyle( 1, c );
 bar(x,y,x+15,y+30);
}

void INITDS::LCD_Ausgabe( float u )
{
 int links=0,rechts=0,Vorzeichen=0,Komma=0;
 float i;

 if(u<0)Vorzeichen =0x80;
 if(u>=0)Vorzeichen=0x00;

 links = u/10;
 rechts = u-links*10;
 rechts=(int)rechts;
 i=u-(int)u;


 if(i==0)Komma=0x00;
 if(i==0.5)Komma=0x20;

 links = links | 0x10;
 outportb( Data, ( rechts | Vorzeichen | Komma ));
 delay(3);
 outportb( Data, ( links  | Vorzeichen | Komma ));
 delay(3);
}
```