Die HPIB, GPIB, IEEE488, IEC-625 – Schnittstelle

Dieser Artikel zeigt, wie man mit geringstem Hardwareaufwand die HPIB-Schnittstelle von Geräten nützen kann. Dies geschieht nach vorhergehender Einführung in die Schnittstelle am Beispiel von fünf Projekten:

1. **Die Steuerung des SMS (Digitaler Signalgenerator von Rohde & Schwarz)**
2. **Ein beliebiges Listener/Talker-Gerät am Beispiel des Keithley 199 bzw. Keithley 2000 Multimeters, am HPIB-Bus betreiben.**
3. **Übertragen der Messungen des Spektrumanalysators HP-8559A bzw. HP-8569A an den PC**
4. **Mehrere Geräte gleichzeitig am HPIB-Bus betreiben.**
5. **Der Messstellenumschalter**

Außerdem wird gezeigt, wie die Messwerte direkt vom Controller in eine beliebige Software (Editor, Textverarbeitung, Tabellenkalkulation etc) geladen werden können und somit die Auswertung wesentlich vereinfacht wird.

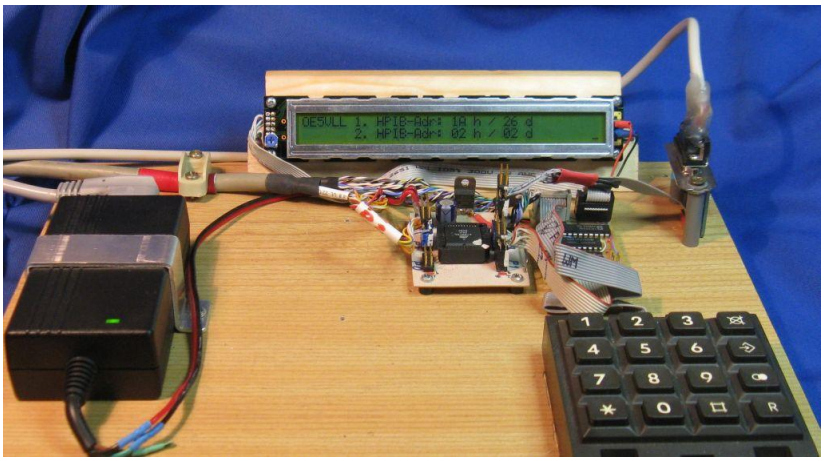


Bild_01: Foto des Testaufbaus mit Mikroprozessorplatine, Zusatzplatine, LCD 2x40 Zeichen, 16er-Tastatur und Netzteil.

Vorwort:

Vorweg: HPIB, GPIB, IEEE488 und IEC625 sind eigentlich nur unterschiedliche Bezeichnungen für ein und das Selbe. Der Einfachheit halber verwende ich immer die Bezeichnung HPIB, weil die meisten der von mir verwendeten Messgeräte mit diesem Bus von HP sind.

Dieser Bericht soll keine hochwissenschaftliche Angelegenheit sein, sondern Hinweise aus der Praxis liefern, wie man mit minimalstem Hardware-Aufwand und geringsten Kosten Messgeräte etc., welche mit so einer Schnittstelle ausgerüstet sind, steuern bzw. deren Messergebnisse direkt z.B. in eine Tabellenkalkulation einfließen lassen kann. Deshalb sind Beschreibungen nicht komplett, sondern immer nur auf das unbedingt Benötigte beschränkt. Für weitere Informationen wird auf die Handbücher dementsprechender Geräte und auf das Internet verwiesen. Die Zielgruppe sind somit jene „Bastler“, welche Interesse haben, so etwas ebenfalls selbst aufzubauen ohne vorher in all die Fallstricke zu gelangen oder aber auch diejenigen, welche sich einfach ein bisschen darüber informieren wollen, wie diese Schnittstelle funktioniert und was man damit so machen kann. Als Eigenbauer sollte man zumindest einen Mikroprozessor programmieren können und mit den dementsprechenden Grundla-

gen vertraut sein. Es ist keine spezielle Programmiersprache erforderlich. Mir ist natürlich auch klar, dass selten jemand exakt die gleichen Geräte zur Verfügung hat. Deshalb bemühe ich mich auch alles so zu beschreiben, dass man möglichst einfach die jeweiligen Vorschläge auch auf andere Messgeräte anwenden kann.

Wie es dazu kam:

Die ganze Geschichte begann damit, dass ich mich beim Wobbeln einer 70-cm-Antenne nicht erst das erste mal darüber ärgerte, dass ich eigentlich keine bessere Möglichkeit fand, als mit dem Finger minutenlang die +10-kHz-Taste des SMS (Digitaler HP Signalgenerator) zu drücken. Der analoge Wavetek-Wobbler war auch nicht gerade das ideale Gerät, weil bis man bei dem die Startfrequenz, die Stopfrequenz und die Wobbelgeschwindigkeit dort hat, wo man sie bräuchte, wollte man eigentlich schon längst fertig sein.

Da dachte ich daran, dass der SMS ja eine HPIB-Schnittstelle hat, über die er sich steuern lässt. Somit sollte es doch möglich sein, ihn über diese Schnittstelle zum „durchfahren“ eines bestimmten Frequenzbereiches zu bewegen.

Nun gibt es kommerzielle HPIB-Bus-Schnittstellen für den PC. Die haben allerdings einige „Haken“. Für die „billigen“ Karten (ca. 50,- bis 100,- Euro) gibt es in den modernen PCs keine Steckplätze mehr. Somit müsste man für diese extra einen alten PC betreiben. Eine Angelegenheit, welche mir nicht zusagte. Bei den moderneren HPIB-Bus-Karten liegt der Preis bei einigen hundert Euro, was mir auch nicht zusagte – frei nach dem Motto: Können muss es viel, aber kosten darf es nur wenig.

Ein anderer Grund war, dass ich des öfteren für Dokumentationszwecke „Bildschirmabzüge“ vom Spectrumanalyzer (HP 8559A im HP853A-Grundgerät mit digitalem Readout) benötige. Die bisherige Möglichkeit, mit der Digitalkamera Fotos vom Bildschirm zu machen war auch nicht gerade das Gelbe von Ei. Außerdem wäre es eine tolle Sache, auch die detaillierten Messwerte in den PC zu bekommen, so ließen sich dann z.B. diverse Messkurven direkt miteinander vergleichen oder z.B. Differenzen berechnen.

Dies alles war schon Grund genug, mich mit der Idee zu befassen, den HPIB-Bus mittels eines selber programmierten Mikroprozessors zu bedienen. Also mal drauf los, diverse Infos aus Messgerätemanuals und dem Internet zusammentragen und mit der Arbeit beginnen. Jedoch begannen damit schon die ersten Schwierigkeiten und brauchbare Ergebnisse waren alles andere als in greifbarer Nähe.

Ein guter Freund borgte mir für Messzwecke einen kommerziellen HPIB-zu-USB-Wandler, damit ich durch Messungen an dieser Schnittstelle mal sehen konnte, wie der tatsächliche Signalverlauf auf diesem Bus ausschaut. Es stellte sich heraus, dass diese Signale eine ziemlich bösartige Angelegenheit sind. Mit meinem Digitalspeicheroszilloskop, welches immerhin 250 Megasampels (250 Millionen Abtastungen pro Sekunde) schafft, war es nicht mehr möglich, die zum Teil extrem schnellen Signalverläufe auf dem Bus ordentlich darzustellen. Nur leichte „Einbuchtungen“ des Signals ließen den Verdacht auf ein LOW-Signal zu. Bei dieser Gelegenheit sei gesagt, dass sowohl die 8 Datenbits, als auch die 5 Steuer- und die 3 Handshake-Signale alle LOW-aktiv sind, also mit negativer Logik arbeiten. Außerdem ist es ziemlich

mühselig, die ganzen Signale mit nur zwei Kanälen zu erforschen – solche Arbeiten sind nicht mehr in wenigen Stunden zu erledigen.

Da kam natürlich die Idee auf, den Logik-Analysator einzusetzen. Ein Jämmerlicher Misserfolg – mit seiner Abtastrate von 20 Nanosekunden war der viel zu langsam. Somit doch wieder zurück zu den Oszi-Messungen.

Nach und nach reiften bei mir die Erkenntnisse und nach einigen Tagen gelang es mir zum Ersten mal, die Remote-LED am SMS aufleuchten zu lassen. Ein zwar kleiner aber nicht unwesentlicher Erfolg – zumal ich davor schon manchmal daran zweifelte, ob ich dieses Projekt nicht lieber doch vergessen sollte.

Nach weiteren drei, vier Tagen hatte ich dann mein Programm so weit, dem SMS eine Frequenz oder auch einen Pegel über die Schnittstelle zu übermitteln, welche er dann einstellt. Das Programm dann so zu erweitern, dass man eine Startfrequenz, eine Stoppfrequenz und ein Intervall (Schrittweite), alles in MHz, eingeben kann und mit diesen Werten den SMS zum dauernden durchlaufen des gewünschten Frequenzbereiches zu bewegen, war dann eigentlich nur mehr eine Kleinigkeit.

Nun zu den wesentlichen technischen Details.

Was ist der HPIB-Bus

Dabei handelt es sich um einen Messgerätebus, welcher ca. 1970 von HP vorgeschlagen wurde und der Datenübertragung zwischen bis zu 15 Messgeräten und einem (in Ausnahmefällen auch mehreren) Controller dient.
Es handelt sich um bidirektionale, bitparallele und byteserielle asynchrone Datenübertragung . Das bedeutet, dass in jeder Richtung Byte für Byte übertragen werden kann. Vorzugsweise werden ASCII-Zeichen, mit 48-mA-Tristate-Treibern mit TTL-Pegel und negativer Logik übertragen. Maximale Leitungslänge 10 m (20m bei 500 kByte/s).
Die Datenübertragung kann beliebig „verlangsamt“ werden.

Beschreibung der Schnittstelle

Diese besteht aus insgesamt 16 Leitungen, welche alle Low-Aktiv sind, was bedeutet dass sie mit negativer Logik arbeiten.
Insgesamt kann man sie in 3 Gruppen einteilen:

Daten-Bus:
Bestehend aus den Datenleitungen 1 - 8

Steuer-Bus:
ATN (ATtentioN)
REN (Remote ENable)
SRQ (Service ReQuest)
IFC (InterFace Clear)
EOI (End Or Identify)

Handshake-Bus:
NRFD (Not Ready For Data)
DAV (Data VAlid)
NDAC (Not Data ACcepted)

In manchen Beschreibungen werden die Signale etwas unterschiedlich bezeichnet, was aber keine weiteren Schwierigkeiten bedeutet.

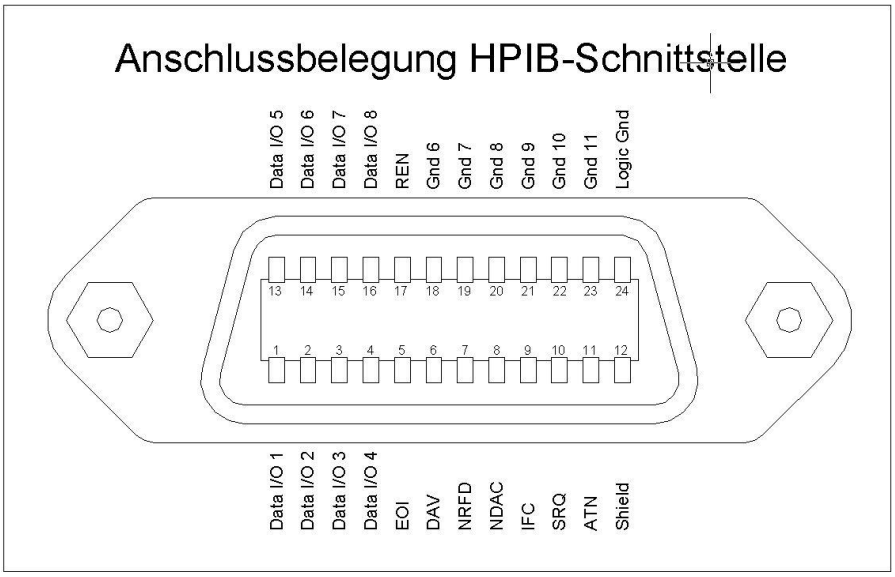


Bild_02: Die Anschlussbelegung der HPIB-Buchse

Die Adressierung der Geräte

Da am HPIB-Bus mehrere Geräte gleichzeitig betrieben werden können, müssen diese bis auf wenige Ausnahmen adressiert werden, damit mit ihnen kommuniziert werden kann. Etwas verwirrend ist am Anfang, dass mehrere Adressen pro Gerät existieren.
Es muss an jedem Gerät eine jeweils andere Adresse eingestellt werden. Auch wenn nur eines am Bus angeschlossen ist, muss dieses eine Adresse haben. Grundsätzlich haben die Geräte üblicherweise einen „Dip-Switch“ oder eine andere Möglichkeit, um diese einstellen zu können. Viele Geräte haben auch eine sogenannte Vorzugsadresse, welche bereits voreingestellt ist.

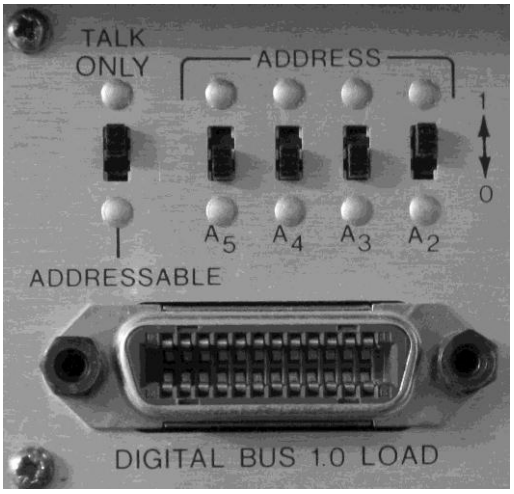


Bild_03: Auf diesem Foto der HPIB-Schnittselle des Frequenzzählers HP5345A kann man sehr gut die 24-polige Buchse, die Kodierschalter für die Adresseinstellung und den Umschalter für den Talk-Only-Betrieb erkennen. Als Besonderheit ist hier zu erwähnen, dass der Schalter A1 „fehlt“ und daher nur „gerade“ Adressen eingestellt werden können. Hier ist die Adresse „2“ eingestellt.

Die eigentliche Adresse besteht aus den niederwertigsten 5 Bit des Adressbytes. Damit ergeben sich 32 Möglichkeiten. Die erste Adresse (00000 Binär) ist jedoch dem Controller vorbehalten, die letzte Adresse (11111 binär) ist „Unlisten“ bzw. „Untalk“ und dient dazu, alle Geräte „abzumelden“.
Je nachdem, ob ein Gerät als „Listener“ (wenn Daten an das Gerät gesendet werden) oder als „Talker“ (wenn Daten vom Gerät empfangen werden) angesprochen wird, ist die an das Gerät gesandte Adresse eine andere. Wird das Gerät als „Listener“ angesprochen, wird zusätzlich zu den 5 Adressbits das 6. Bit gesetzt und das 7. Bit nicht gesetzt. Wird es als „Talker“ angesprochen, ist es umgekehrt, Bit 7 wird gesetzt und Bit 6 wird nicht gesetzt. Das 8. Bit wird nie gesetzt.
ACHTUNG: Wegen der negativen Logik bedeutet „Bit gesetzt“ Low (0 Volt) am Bus, „Bit nicht gesetzt“ bedeutet High (+5 Volt) am Bus. Für meine Zwecke habe ich diese Angelegenheit derart gelöst, dass ich generell mit normaler positiver Logik arbeite, und die Signale jeweils erst unmittelbar vor dem Senden bzw. nach dem Empfang mit dem „Complement“-Befehl in bzw. von negativer Logik umwandle.
Dazu kommt noch, dass in den Manuals die Adressangaben meistens Dezimal oder gar als druckbare Zeichen angegeben werden. Zwei Beispiele sollen das veranschaulichen:

Beispiel 1:
Eingestellte Geräteadresse: 00110 Binär = 06 Hexadezimal = 6 Dezimal
Listener-Adresse: 00100110 Binär = 26 Hexadezimal = 38 Dezimal = „&“
Talker-Adresse: 01000110 Binär = 46 Hexadezimal = 70 Dezimal = „F“

Beispiel 2:
Eingestellte Geräteadresse: 11101 Binär = 1D Hexadezimal = 29 Dezimal
Listener-Adresse: 00111101 Binär = 3D Hexadezimal = 61 Dezimal = „=“
Talker-Adresse: 01011101 Binär = 5D Hexadezimal = 93 Dezimal = „]“

Misst man direkt an den Busleitungen, ist wiederum die negative Logik zu beachten, 1 ist dann 0 Volt und 0 ist dann ca. 5 Volt !

Wie man sieht, braucht man hexadezimal nur jeweils 20 Hex zur Grundadresse zu addieren, um die Listener-Adresse zu erhalten, bzw. 40 Hex um die Talker-Adresse zu erhalten.

Ganz wesentlich ist noch, dass während einer Adressübertragung das ATN-Signal (Attention) vom Controller am Bus LOW gesetzt ist – damit können alle angeschlossenen Geräte eine Adresse als solche erkennen.

Manche Geräte haben mehrere Grundadressen, dies soll aber hier nicht speziell behandelt werden, gegebenenfalls ist deren Bedeutung dann den jeweiligen Manuals zu entnehmen.

Kleiner Tipp am Rande: Wer schnell mal zwischen Dezimal-, Hex-, Oktal- oder Binärwerten umrechnen will, kann dazu den Windows-Rechner in der Ansicht „wissenschaftlich“ verwenden.

Die Datenübertragung – das „handshaking“

In fast jedem Manual findet man ein Timing-Diagramm für den HPIB-Bus, schematisch sind sie meistens ziemlich gleich. Sie zeigen aber immer nur die Signale des Handshake-Bus, nicht aber die Signale des Steuer-Bus.

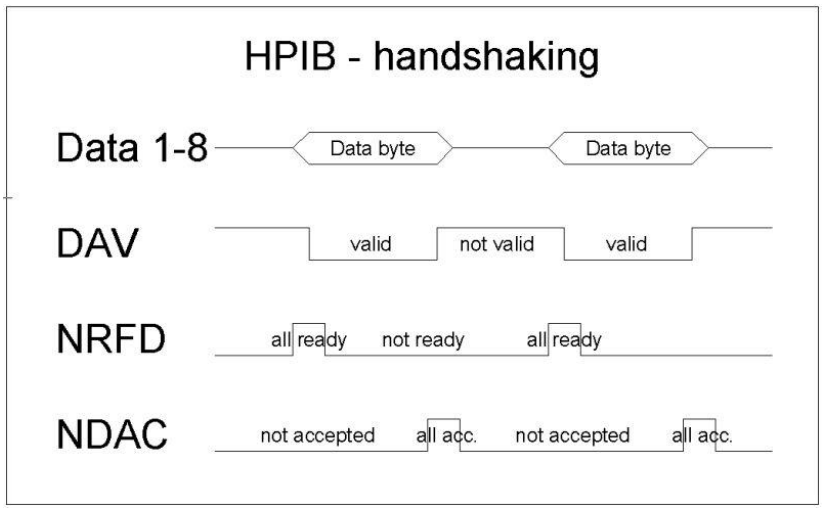


Bild_04: Das HPIB-handshaking wie es üblicherweise dargestellt wird

Wesentlich ist: Datenleitungen und DAV-Signal werden vom „Talker“ gesteuert, NRFD und NDAC vom „Listener“. Je nach Erfordernis des Datenflusses tauschen Messgerät und Controller die Funktionen. Gerade hierin liegt eine nicht unwesentliche Fehlerquelle, deshalb muss dies genau erläutert werden.

Ein entscheidender Punkt betrifft die Festlegung, wo ein Übertragungszyklus, egal in welcher Richtung, beginnt. Eigentlich mit dem Setzen des DAV-Signals (DAV wird Low!). Das setzen von DAV bedingt aber, dass der vorhergehende Zyklus abge-

schlossen ist, was man daran erkennt, dass das NRFD-Signal High wird. Das bedeutet, dass nach jedem Bus-Zugriffszyklus dieser auch wieder eindeutig beendet und alle Signale in Ausgangsstellung sein müssen. Somit kann man sagen, dass zwar theoretisch ein neuer Zyklus damit beginnt, dass DAV Low wird, praktisch aber mit dem Schritt davor, wo gewartet wird, bis NRFD High wird (in den meisten Fällen wird NRFD bereits High sein, dies muss aber von der Software überprüft werden).

Eine Datenübertragung wird immer durch eine Adressierung eingeleitet. Davor werden generell alle am Bus angeschlossenen Geräte durch senden von „Unlisten“ (3F hex) und „Untalk“ (5F hex) abgemeldet, anschließend wird die Listener-Adresse des gewünschten Gerätes gesendet. Während dieser ganzen Prozedur ist das ATN-Signal gesetzt (0 Volt am Bus).

Controller sendet Byte an Gerät:

a.) Controller wartet auf NRFD = High (All Listener are ready for Data)
b.) Datenbyte wird vom Controller auf den Datenbus gelegt
c.) Mindestens 2 µsec Wartezeit
d.) DAV wird vom Controller auf Low (0 Volt am Bus) gesetzt
e.) NRFD wird vom Gerät auf Low gesetzt
f.) Controller wartet bis NDAC auf High (+5Volt am Bus). Bei mehreren Geräten bestimmt das langsamste den Zeitpunkt.
g.) DAV vom Controller wieder auf High setzen
h.) NDAC wird vom Gerät wieder auf Low gesetzt
i.) NRFD wird vom Gerät wieder auf High gesetzt (entspricht dem Punkt a.), damit ist Zyklus beendet

Diese Prozedur ist peinlichst genau einzuhalten. Wird das nicht beachtet, können Fehler entstehen, welche sich erst viel später an ganz anderer Stelle auswirken, und man erkennt dann unter Umständen nur sehr schwer die eigentliche Ursache des Fehlers.

Ist die Adressierung abgeschlossen, können an das adressierte Gerät ein oder auch mehrere Befehle übertragen werden.

Werden anschließend Daten vom Gerät erwartet, wird davor wieder ein Adressierungszyklus gestartet, allerdings mit dem Unterschied, dass jetzt die Talker-Adresse gesendet wird.

Geräte am HPIB-Bus

Jedes Gerät mit HPIB-Schnittstelle ist gesondert zu behandeln und hat seine eigenen Besonderheiten.

Prinzipiell lassen sich diese Geräte aber in drei Gruppen einteilen:

1. Geräte welche nur Befehle entgegennehmen aber keine Daten senden (Listener). Dazu zählt z.B. der SMS Signalgenerator von Rohde & Schwarz. Dessen Antworten sind an der HF-Buchse zu „sehen“, aber nicht am HPIB-Bus.
2. Geräte welche nur Daten senden aber keine Befehle entgegennehmen. Dazu zählt z.B. der Spectrumanalyzer HP8559A, welcher Daten auf Knopfdruck sendet, aber keine Befehle über den HPIB-Bus entgegennehmen kann.
3. Geräte, welche sowohl Listener als auch Talker sein können, z.B. das Multimeter Keithley 199.

Es gibt auch Sonderfälle, wie z.B. den Frequenzzähler HP5345A, welcher durch einen Umschalter einerseits als reiner Talker, andererseits als Listener/Talker betrieben werden kann.

Die Befehle, welche an die Listener gesendet werden können, sind praktisch immer systemspezifisch und unterscheiden sich von Gerät zu Gerät. Bis jetzt sind mir aber nur Befehle aus ASCII-Zeichen (Grossbuchstaben, Ziffern und einige Sonderzeichen) untergekommen. Hier ist genaues Studium der jeweiligen Handbücher angesagt. Auf einen Stolperstein sei aber bereits hier verwiesen: Obwohl es ein eigenes Bussignal (EOI) zur Ende-Erkennung gibt (dieses wird während der Übertragung des letzten Bytes auf Low gesetzt), wollen diverse Geräte ein bestimmtes Zeichen als Ende-Erkennung einer Befehlssequenz übertragen haben. Das Keithley 199 DMM will ein „X“, der R&S SMS will ein „Komma“, andere wiederum benötigen kein Ende-Zeichen. Wenn solche benötigt werden, so geht das nicht immer klar aus den Unterlagen hervor, man muss unter Umständen schon danach suchen. Beim Keithley-DMM 199 kann man das Verhalten des Gerätes in Bezug auf das EOI-Signal übrigens per HPIB-Bus programmieren.

Hat man keine Unterlagen, wie es bei einem Siemens-Multimeter war, bleibt nur noch die Möglichkeit, durch Versuche und Vergleiche mit den Befehlen anderer Multimeter die Codes herauszufinden, was aber auch innerhalb kurzer Zeit gute Erfolge brachte. Also sind auch solche Fälle nicht hoffnungslos.

Das 1. Projekt:
Die Steuerung des SMS (Digitaler Signalgenerator von Rohde & Schwarz)

Wer mit dem SMS Frequenzabhängige Signalverläufe z.B. am Spectrumanalyzer darstellen will, kann dies tun, indem er mit dem Finger auf der +10kHz-Taste bleibt und mittels der automatischen Tastenwiederholung den gewünschten Frequenzbereich überstreichen. Dies ist aber doch etwas mühselig und wenn man das öfter braucht, wünscht man sich sehnlichst eine Automatik, welches das selbsttätig erledigt. Somit ein ideales HPIB-Projekt. Damit niemand eine unangenehme Überraschung beim Kauf eines SMS erlebt: Dieser ist nicht generell mit einem HPIB-Bus ausgestattet, es gibt also auch Geräte ohne diese Schnittstelle. Bei einem eventuellen Kauf unbedingt vorher abklären, ob diese Option mit eingebaut ist, speziell bei Einkauf via ebay.

Bild_05: Der SMS Signalgenerator

Für diesen Zweck wünschte ich ein Gerät, welches eigenständig direkt in der Nähe des SMS betrieben werden kann und keinen eigenen PC benötigt. Es sollten Startfrequenz, Stoppfrequenz und Schrittweite eingegeben und anschließend der automatische Ablauf gestartet werden können.

Der SMS ist ein reiner Listener. Das bedeutet, er kann nur Befehle empfangen, selber aber keine Antworten geben. Das erleichtert den Anfang, da hier noch kein Wechsel der Datenrichtung vorgenommen werden muss, es werden nur Daten vom Prozessor zum SMS gesendet.

Der von mir verwendete Prozessor AT89C55 hat 4 Ports zu je 8 Bit. Ursprünglich verwendete ich je einen Port für Tastatur (16 Tasten) und einen für das LCD (2 x 40 Zeichen). Der dritte Port wird für den HPIB-Datenbus und der vierte Port sowohl für die HPIB-Steuer- und Handshake-Leitungen als auch (später) für die serielle Schnittstelle verwendet. Da insgesamt somit zuwenig Busleitungen zur Verfügung stehen, wurden nur die unbedingt benötigten Steuerleitungen verwendet – die Handshake-Leitungen werden übrigens alle drei unbedingt benötigt.

Da es sich hier um keinen Mikroprozessor-Kurs handelt, wird hier auch nicht der Betrieb einer 16-Tasten-Tastatur und eines LCD beschrieben. Es gibt aber auch die Möglichkeit, dass die Software derart gestaltet wird, dass ohne LCD und Tastatur, dafür aber mit einem PC (speziell Notebook) als Ein-/Ausgabe-Gerät gearbeitet werden kann.
.

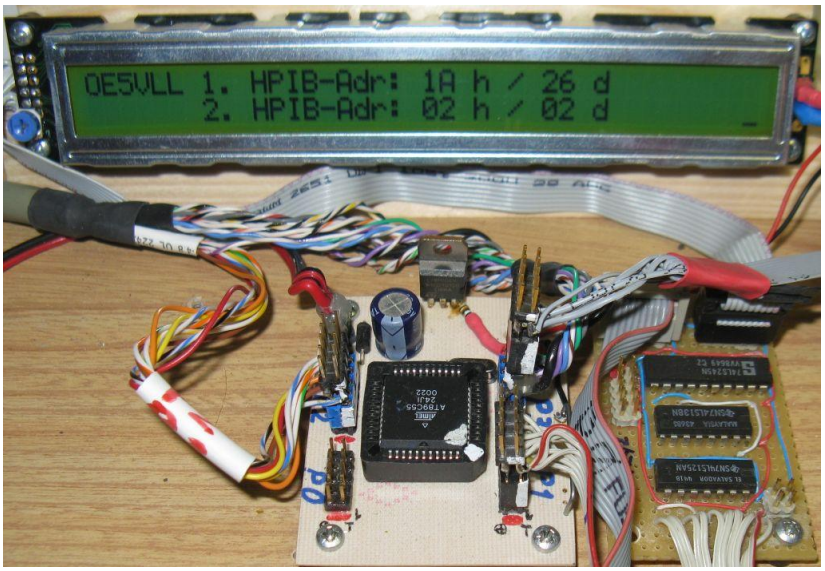


Bild_06: Ein Foto von Prozessorplatine, Zusatzplatine und dem LCD im Testaufbau

Die von mir verwendeten Leitungen des HPIB-Bus sind:

1. Die 8 Datenleitungen
2. Die 3 Leitungen des Handshake-Bus
3. Vom Steuerbus: ATN, REN und EOI

Die Leitungen IFC und SRQ wurden von mir sozusagen links liegengelassen, da ich die restlichen 2 Pins des Ports für die später hinzukommende serielle Schnittstelle reserviert hatte. Diese werden hier auch nicht benötigt.

Ist nur ein Gerät am HPIB-Bus angeschlossen, kann man sich einige Signale bzw. Daten einsparen. Dies ist aber nicht anzuraten, denn früher oder später legt man sich womöglich doch ein oder mehrere weitere Geräte mit dieser Schnittstelle zu, und dann ist es nicht schlecht, vorhandene Routinen ohne großartiges umprogrammieren weiterverwenden zu können.

Als erstes muss die Hardware komplett sein. Dazu zählt ganz speziell auch ein Bus-Kabel. Auf Amateurfunk-Flohmärkten aber auch im Internet werden dementsprechende angeboten. Sie sollten nicht zu kurz sein, empfehle jene mit rund 2 Metern Länge oder mehr.
Ein Unterschied bei den Kabeln besteht auch darin, ob die Stecker zerlegbar oder ob es vergossene Ausführungen sind. Eines meiner Kabel (leider mit vergossenen Steckern) hatte bei einer Datenbus-Ader eine Unterbrechung und war somit eigentlich unbrauchbar. Doch ich hatte eine Idee. Mit einem Kapazitätsmessgerät stellte ich fest, an welchem Ende die Unterbrechung war (dort ist die geringere Kapazität zu benachbarten Adern zu messen). Dort schnitt ich das Kabel einfach ab und montierte zwei 10-polige Gegenstücke für Pfostenstecker, welche ich dann direkt an das Prozessorboard stecken konnte. Somit hatte ich wieder ein zweites Kabel, welches ich später unbedingt brauchte. Wenn jemand einen einzelnen HPIB-Stecker sein Eigen nennt, gibt es einen weiteren Tipp: Aus alten Druckerkabeln lassen sich dann Eigenbau-HPIB-Kabel herstellen, diese haben genügend Adern, sind auch geschirmt, und werden in den meisten Fällen wegen der moderneren Schnittstellen nicht mehr benö-

tigt – habe es selber getestet und funktioniert recht gut. Falls jemand sehr lange Kabel (z.B. 10 m) verwenden will, dem sei angeraten dann qualitativ sehr hochwertige Kabel (z.B. von HP) zu verwenden. Billige oder Eigenbau-Kabel können da schon mal zu Problemen führen.

Außer dem speziellen Schnittstellenkabel braucht man eigentlich nur noch ein Miroprozessorsystem mit dementsprechend vielen freien Port-Leitungen, welches man zu programmieren in der Lage ist. Falls die Ports nicht TTL-Pegel entsprechen, sind noch dementsprechende Bustreiber-ICs vorzusehen. Welcher Typ Mikroprozessor verwendet wird ist eigentlich egal, sofern er die Mindestanforderungen speziell in Bezug auf die Anzahl der Ports erfüllt – sogar hier ließe sich mit geeigneter Beschaltung nachhelfen. Später wird von mir noch eine Schaltung mit 3 TTL-ICs beschrieben, mit welcher Tastatur und LCD an nur einem Port betrieben werden können.

Als erstes wird die Adressierung programmiert. Für die ersten Tests reicht es, eine fixe Adresse zu vergeben. Davor ist allerdings festzustellen, auf welche Adresse der SMS eingestellt ist. Des weiteren kann beim SMS am Anfang auf die „Untalk“- und „Unlisten“-Bytes verzichtet werden.

Nun zur Programmierung. Nach der eigentlichen Prozessorinitialisierung werden die Ports in Grundstellung gebracht. Danach erfolgt die Adressierung des Gerätes.

a.) Ursprünglich alle HPIB-Pins auf High (+5 Volt).
b.) Als erstes wird REN vom Controller auf Low (0 Volt) gesetzt.
c.) ATN wird vom Controller auf Low gesetzt (weil eine Adresse übertragen wird).
d.) Warten bis NRFD = High (ist in diesem Fall üblicherweise schon High).
e.) Adressbyte (invertierte Talker-Adresse) vom Controller auf den Bus legen.
f.) Minimal 2 µsec warten.
g.) DAV vom Controller auf Low legen.
h.) Warten bis NDAC = High
i.) DAV vom Controller wieder auf High setzen.
j.) ATN vom Controller wieder auf High setzen.

Damit sollte am SMS die Remote-LED aufleuchten. Geschieht dies, hat man die erste Hürde geschafft, denn damit gibt der SMS bekannt, dass er in die Fernsteuer-Betriebsart gewechselt hat.

Dabei sei darauf hingewiesen, dass ich zwischen den einzelnen Programmschritten an vielen Stellen sogenannte NOP´s (Befehle welche nur ein bisschen verzögern – sogenannte Leer-Befehle) eingefügt habe. Damit lässt sich eine eventuelle Fehlersuche mit dem Oszilloskop leichter durchführen. Der Bus lässt sich beliebig verzögern.

Anschließend kann zur Übertragung des ersten Befehls geschritten werden. Eine der einfachsten Befehlsfolgen ist „S1,“. S steht für „Pegel in dBm einstellen“, 1 steht für „+1 dBm“ und das Komma muss als Abschluss der Befehlsfolge gesendet werden.

Das EOI-Signal wird vom SMS nicht verwendet und kann somit für diesen weggelassen werden. Im Hinblick auf weitere Geräte sollte dieses Signal aber mit verwendet werden.

Im Prinzip folgt drei mal der selbe Ablauf wie bei der Adressierung von Punkt c.) bis Punkt i.). REN bleibt generell auf Low, ATN bleibt jetzt aber auf High. Statt der Adresse wird nacheinander „S“ (53 hex), „1“ (31 hex) und das „Komma“ (2C hex) übertragen. Nur während des letzten Bytes (in diesem Fall das Komma) wird dann als Start das EOI-Signal auf Low und nach der Übertragung wieder auf High gesetzt (so wie bei der Adressübertragung das ATN-Signal). Wobei hier noch einmal erwähnt wird, dass der SMS dieses Signal nicht verwendet. Nach Übertragung dieser 3 Bytes sollte der SMS auf einen Pegel von +1 dBm eingestellt sein. Hat man dies erfolgreich geschafft, ist man eigentlich schon auf der sicheren Seite, denn dann kann man im Prinzip schon alle Einstellungen programmieren. Wesentlich für später ist noch, dass der SMS ca. 40 ms benötigt, um Einstellungen auszuführen. So lange sollte dann mit dem nächsten Befehl gewartet werden.

Die diversen Befehle für den SMS bestehen aus jeweils einem Grossbuchstaben (Ausnahme ist der Klammeraffe für eine Pause). Anschließend werden die numerischen Parameter (nicht bei allen Befehlen) und am Ende immer das Komma gesendet.

Hier einige beispielhafte Befehlsfolgen für den SMS:

„A145.625,“ = Frequenz 145,625 MHz einstellen
„B50,“ = AM-Modulation mit Modulationsgrad 50 Prozent einstellen
„C,“ = Keine Modulation (unmoduliert) einstellen
„J,“ = Modulationsfrequenz intern 1 kHz einstellen
„S-111,5,“ = Pegel –111,5 dBm einstellen
„P10,“ = Pegel 10 µV einstellen

Hat man hier ein paar Programmiertests durchgeführt, kann man sich die Programmierung der Wobbelroutine überlegen. Hier sind der Fantasie des Programmierers keine Grenzen gesetzt. Ich habe 3 Eingaberoutinen geschrieben, je eine für die Startfrequenz, die Stoppfrequenz und die Schrittweite, jeweils in MHz (damit es schön einheitlich bleibt). Eine weitere Routine startet die eigentliche Wobbelroutine. Als erstes wird der SMS auf die Startfrequenz gesetzt, dann die Schrittweite aufaddiert und geschaut ob die Stoppfrequenz bereits überschritten. Ist dies nicht der Fall wird die neu berechnete Frequenz übertragen und so lange die Addition und die Frequenzeinstellung wiederholt, bis die Stoppfrequenz überschritten ist. Dieser Wert wird dann nicht mehr übertragen, sondern wieder die Startfrequenz eingestellt und das ganze beginnt von vorne bis das Programm abgebrochen wird. Der SMS führt dieses mit einer Geschwindigkeit von ca. 13 Frequenzeinstellungen pro Sekunde durch. Ist somit sogar etwas schneller als mit der „Finger-auf-der-Taste-Methode“. Auf eine Plausibilitätsüberprüfung der Frequenzen (z.B. dass die Stoppfrequenz höher als die Startfrequenz sein muss) wurde bewusst verzicht, denn erstens passiert bei einer Falsch-Eingabe nichts und zweitens muss man ja sowieso bei solchen Sachen ein bisschen denken und überlegen was man tut.

Wie man sieht ist nichts Großartiges dabei und die erste Hürde kann schnell genommen sein.

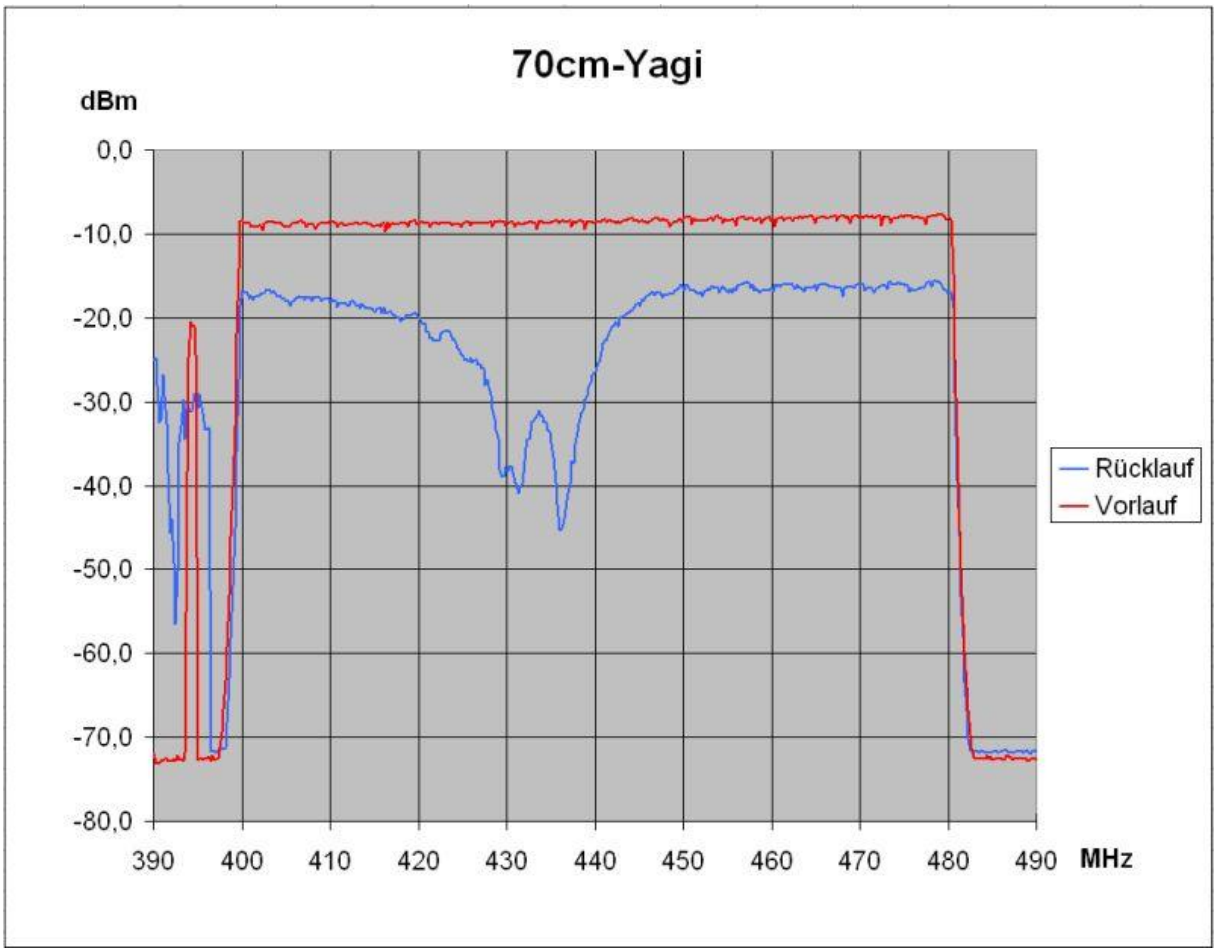


Bild_07: Das Diagramm zeigt als Beispiel Vor- und Rücklauf einer 70cm-Yagi via 35 m RG213-Koaxkabel. Als Richtkoppler wurde ein HP-775D mit einer Auskoppeldämpfung von 20 dB verwendet. Gewobbelt wurde von 400 bis 480 MHz mit 100 kHz Schrittweite. Die unregelmäßigen Schwingungen ganz links im Bild (390 bis 400 MHz) zeigen Einschwingvorgänge des SMS und wurden bewusst nicht entfernt. Normalerweise stellt man den Wobbelbereich so ein, dass diese nicht sichtbar sind. Die kleinen unregelmäßigen Zacken am oberen Rand der Kurve können durch mehrere Wobbeldurchläufe oder eine kleinere Schrittweite verringert werden. Auch diese wurden bewusst dargestellt, damit man auch sieht, wo hier Grenzen und Nachteile sind.

Das 2. Projekt:
Ein beliebiges Listener/Talker-Gerät am Beispiel des Keithley 199 bzw. Keithley 2000 Multimeters, am HPIB-Bus betreiben.

Bild_08:
Das Digitalmultimeter Keithley 199

Das Keithley 2000 kann im „Keithley 199 Modus“ betrieben werden. Dadurch sind die verwendeten 199er-Befehle auch eins zu eins für das 2000er verwendbar.

Bild_09:
Das Digitalmultimeter Keithley 2000

Man stelle sich eine einfache Problemstellung vor: Die Beobachtung der Netzspannung über einige Stunden oder länger. Ohne Messgerät, welches die Daten an einen Computer übergibt, würde dies eine ziemlich aussichtslose Angelegenheit werden.

Wie an ein Gerät über den HPIB-Bus Befehle gesendet werden, wurde schon beim ersten Projekt beschrieben. Nun geht es darum, auch Daten vom Messgerät in Empfang zu nehmen und an einen PC zu übermitteln.

Somit wird als erstes das Mikroprozessorsystem um eine serielle Schnittstelle erweitert, da ja nun von einem daran angeschlossenen PC sowohl diverse Befehle an das HPIB-Gerät übermittelt als auch Daten von diesem Gerät an den PC übergeben werden sollen.

Nun zur Softwareroutine, welche diesen Datentransfer steuert.

Als erstes wurde eine Interrupt-Routine für die RS232-Schnittstelle geschrieben, welche, wenn Daten vom PC einlangen, diese in einen Zwischenspeicher ladet. Wird „CR“ (Carriage Return) oder „LF“ (Line Feed) erkannt, wird dies als Ende der Übertragung gewertet und ein Merker-Bit gesetzt, welches auf an den HPIB-Bus zu übertragende Daten hinweist.

Nun der eigentliche Programmablauf:

a.) NRFD vom Controller auf High (freigeben des Signals)
b.) NDAC vom Controller auf Low setzen
c.) Check, ob DAV = Low, wenn ja, Sprung nach e), sonst weiter

d.) Check, ob Merker-Bit gesetzt – wenn ja, Sprung nach o.), sonst Rücksprung nach b.)
e.) DAV = Low, vom Gerät sind Daten abzuholen
f.) NRFD vom Controller auf Low setzen
g.) Byte vom HPIB-Port übernehmen
h.) NDAC vom Controller auf High setzen
i.) Warten, bis DAV = High
j.) NDAC vom Controller auf Low setzen
k.) NRFD vom Controller auf High setzen
l.) Byte vom HPIB-Port invertieren
m.) Das Byte via RS232 an den PC senden
n.) Rücksprung nach b.)
o.) Merker-Bit ist gesetzt, Daten von RS232 an den HPIB-Bus übermitteln
p.) Senden von Unlisten, Untalk und Listener-Adresse
q.) Byte von Zwischenspeicher holen
r.) Wenn letztes Zeichen, dann EOI auf Low setzen
s.) Warten bis NRFD = High
t.) Datenbyte invertieren und an HPIB-Port übergeben
u.) DAV vom Controller auf Low setzen
v.) Warten bis NDAC = High
w.) DAV vom Controller wieder auf High setzen
x.) Wenn letztes Zeichen, dann EOI wieder auf High setzen und Merker-Bit rücksetzen
y.) Wenn nicht letztes Zeichen, dann nächstes Zeichen von Zwischenspeicher holen und Rücksprung nach r.)
z.) Rücksprung nach b.)

Des weiteren sei noch erwähnt, dass ich die Daten, welche, egal in welcher Richtung sie übertragen werden, zu Kontrollzwecken auch am LCD ausgebe. Speziell wenn etwas nicht funktioniert kann das bei der Fehlersuche sehr hilfreich sein.

Nun zur PC-Seite:

PC-seitig kann im Prinzip jedes beliebige Terminalprogramm verwendet werden, welches ASCII-Daten über eine serielle Schnittstelle ausgeben bzw. entgegennehmen kann. Es funktionieren natürlich auch die Adapter Seriell-auf-USB, speziell für Rechner, welche gar keine RS232-Schnittstelle mehr haben.

Persönlich verwende ich Comtool in der Version 2.02, welches von OE5GWM geschrieben wurde und einige Vorzüge bietet. Man kann Einstellungen für unterschiedliche Geräte abspeichern, die Sende- bzw. Empfangsdaten sowohl Hexadezimal als auch als ASCII-Zeichen darstellen und vieles andere mehr.

Dieses Programm steht kostenlos zur Verfügung und kann von www.marking.at, unter Downloads als 251 KByte großes Zip-File heruntergeladen werden. Nach dem „Entzippen“ hat das File als Comtool.Exe 513 KByte. Für den Betrieb benötigt es keine Installation, es muss nur „gestartet“ werden. Im Betrieb legt das Programm noch das Textfile ComTool.Ini im selben Verzeichnis an. Hierin werden die Einstellungen gespeichert.

Bezüglich der Baudrate ist noch etwas zu erwähnen: Da ich, wie bereits beschrieben, mit der Anzahl der I/O-Pins sehr beschränkt war, habe ich für die serielle Schnittstelle nur die Tx- und die Rx-Leitung vorgesehen, nicht aber die Handshake-Leitungen DSR, DTR, CTS und RTS. Das funktionierte bei mir so lange, bis ich ein etwas langsameres älteres Notebook einsetzte, welches die Zeichen nicht schnell genug verarbeitete. Deshalb habe ich dann die ursprünglich verwendete Baudrate von 9600 auf 4800 Baud umgestellt und nun funktioniert das auch auf dem langsamen Rechner einwandfrei.

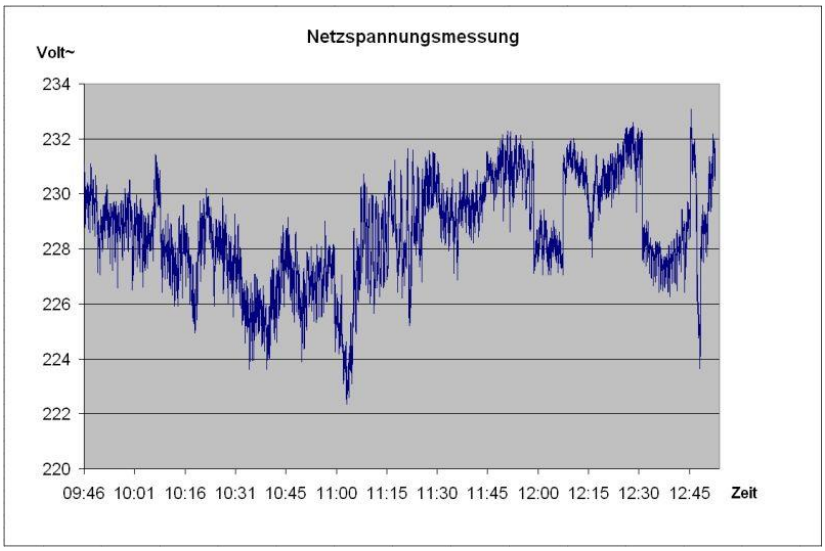


Bild_10: Hier als Beispiel eine Aufzeichnung der Netzspannung über ca. 3 Stunden und als Excel-Diagramm dargestellt.

Das 3. Projekt:
Übertragen der Messungen des Spectrumanalysators HP-8559A bzw. HP-8569A an den PC

Bild_11:
Spectrumanalyzer HP-8559 im Grundgerät HP-853A

Der Spectrumanalyzer HP-8559A im Grundgerät HP-853A ist ein sogenannter Talker. Man kann per Taste die Ausgabe des Bildschirminhaltes über den HPIB-Port veranlassen. Eine Taste gibt „das Raster“ aus, die zweite Taste die Kennlinien der beiden Kanäle.
Bei HP-8569 ist es im Prinzip gleich, man kann nur zusätzlich auch noch die Einstellwerte des Gerätes ausgeben. Befehle an den Analyzer können nicht übermittelt werden. Das bedeutet auch, dass der Analyzer NICHT ADRESSIERT werden muss (kann). Es brauchen bloß die Daten in Empfang genommen werden.

Nun wurde dies ursprünglich vorgesehen, um mit diesen Daten direkt einen Plotter im HPGL-Format anzusteuern und die Kennlinien mitsamt Raster auf Papier zu bringen. Das bedeutet, dass die Daten vom Analyzer zwar als ASCII-Zeichen eintrudeln, aber so nicht direkt zu gebrauchen sind. Habe diese Daten dann mal mit einem HPGL-Viewer am PC dargestellt. Das funktioniert zwar, hat aber den Nachteil, dass man bei den beiden Kennlinien nie weiß, ob sie sich nur berühren oder ob sie sich kreuzen. Konnte die Daten dann zwar ins Autocad (CAD-Programm) exportieren und dort den beiden Kennlinien mühselig unterschiedliche Farben verpassen, aber das ganze war doch ziemlich aufwendig und deshalb bei mir nicht sehr beliebt.
Da ich ja einen Mikroprozessor zwischen PC und Analyzer habe, dachte ich, der könnte das nötige umwandeln des Datenstromes übernehmen und analysierte mal die gesendeten Files. Es stellte sich heraus, dass eigentlich nur an 5 Stellen Daten entfernt werden müssen, damit man die Daten für die beiden Kennlinien erhält. Diese sind dann nach folgendem Schema aufgebaut:

,001,049,002,047,003,047 bis ,479,032,480,048;
,001,042,002,039,003,036 bis ,479,022,480,024;

Die erste Zeile ist Kennlinie 1, die zweite Zeile Kennlinie 2. Jede Kennlinie besteht aus 480 Werte-Paaren, wovon jeweils der erste Wert die laufende Nummer für die Horizontal-Position (X-Wert) und der Zweite den Vertikal-Wert (Y-Wert, Amplitude) darstellt.
Horizontal werden somit 48 Punkte pro „cm“ dargestellt.
Der Vertikalwert entspricht 100 Einheiten pro „cm“ auf dem Bildschirm, kann jedoch nicht nur die dem Bildschirm entsprechenden Werte 000 bis 800, sondern bis 1000 annehmen. Die Werte über 800 sind dann aber als „Überlauf“ anzusehen.

Somit entstand die Routine „Daten vom Spex“. Diese sorgt dafür dass sämtliche nicht benötigten Teile der Daten entfernt und die Kommas nach den Vertikal-Werten durch „LineFeed“ (= 0A hex) ersetzt werden. Das ersetzen der Kommas ist für die Eingabe

in Excel nötig, damit jeweils nach einem Werte-Paar in eine neu Zeile gesprungen wird.

Hier der Programmablauf für die Extrahierung der eigentlichen Datenbytes. Zu bemerken ist, dass bei jedem Zeichen vorher der Reihe nach die Merkerbits abgefragt werden, bis das erste nicht gesetzte Bit gefunden wird, dann wird die zugehörige Routine durchlaufen.

a.) Empfangene Bytes nicht an RS232 ausgeben
b.) Check ob „A“, Bytes weiterhin nicht ausgeben, Merkerbit 60 setzen
c.) Check ob „1“,wenn ja, ab hier Bytes an RS232 ausgeben, Merkerbit 61 setzen
d.) Check ob „Strichpunkt“ (3B hex), Bytes ab hier nicht ausgeben, Merkerbit 62 setzen
e.) Check ob „0“, wenn ja, Bytes ab hier an RS232 ausgeben, Merkerbit 63 setzen
f.) Check ob „Strichpunkt“ (3B hex), Bytes ab hier nicht ausgeben, Merkerbit 64 setzen
g.) Check ob „1“, wenn ja, ab hier Bytes an RS232 ausgeben, Merkerbit 65 setzen
h.) Check ob „Strichpunkt“ (3B hex), Bytes ab hier nicht ausgeben, Merkerbit 66 setzen
i.) Check ob „0“, wenn ja, Bytes ab hier an RS232 ausgeben, Merkerbit 67 setzen
j.) Check ob „Strichpunkt“ (3B hex), Bytes ab hier nicht ausgeben, Merkerbit 68 setzen
k.) Check ob LineFeed (0A hex), (= Ende erreicht). Alle Merkerbits rücksetzen

Die Daten direkt in eine Tabellenkalkulation oder andere Windows-Software laden

Vor einigen Monaten stieß ich auf einer Software-CD der Computerzeitschrift c´t auf das Programm „ewCaptSer“ von Dr. Erwin Wolf Labordatenverarbeitung.. Diese Software ermöglicht, Daten an der seriellen Schnittstelle entgegenzunehmen und via Software-Emulation einer Tastatur die Daten in praktisch beliebige Software wie Editor, Textverarbeitung, Tabellenkalkulation etc. einzugeben. Wollte dies damals sofort für mein Multimeter mit serieller Schnittstelle verwenden. Letztlich scheiterte ich aber daran, dass mein Multimeter auf jeweils ein Zeichen wartet, um anschließend einen Messwert zu übermitteln. Da mittlerweile mein HPIB-Projekt entstanden war und auch andere Messgeräte zur Verfügung standen, erinnerte ich mich dieser Software und dachte daran, diese einzusetzen.
EwCaptSer beherrscht die üblichen Einstellmöglichkeiten für die serielle Schnittstelle. Des weiteren können via Konfiguration / Filter/Zulässig jene Zeichen eingegeben werden, welche von der Software überhaupt weitergeleitet werden sollen. Das bedeutet, dass jene Zeichen, welche nicht in dieser Liste stehen, auch nicht durchgereicht werde. Dies ist sehr praktisch zum Extrahieren von überflüssigen Zeichen, kann aber auch Anlass zur Fehlersuche sein, wenn man benötigte Zeichen vergisst in der Liste einzutragen.
Wer im Internet nach dieser Software sucht, wird schnell fündig werden. Sie ist Freeware und kostenlos herunterzuladen.

Somit hat man die Möglichkeit, sich z.B. eine Excel-Tabelle inklusive Diagramm einzurichten, von dieser jeweils vorher ein Kopie abspeichern und in diese dann die Daten direkt reinzuladen. Wenn man es richtig macht, sieht man dann sogar in Echtzeit die Diagrammlinien wachsen (allerdings nur so lange, bis durch die Eingaben das Diagramm aus dem Bildschirmbereich geschoben wird).
Dies hat dann den Vorteil, dass man von vornherein den beiden Kennlinien getrennte Farben zuweisen kann, Auf diese Art hat man sehr schnell super Dokumentationsunterlagen beisammen. Des weiteren lassen sich so auch zu unterschiedlichen Zeiten erfasste Messwerte miteinander vergleichen bzw. auch verrechnen (Stichwort: Differenzbildung).

Das Programm ewCaptSer lässt sich selbstverständlich auch für andere Geräte am HPIB-Bus einsetzen und ich schätze diese Software mittlerweile sehr.

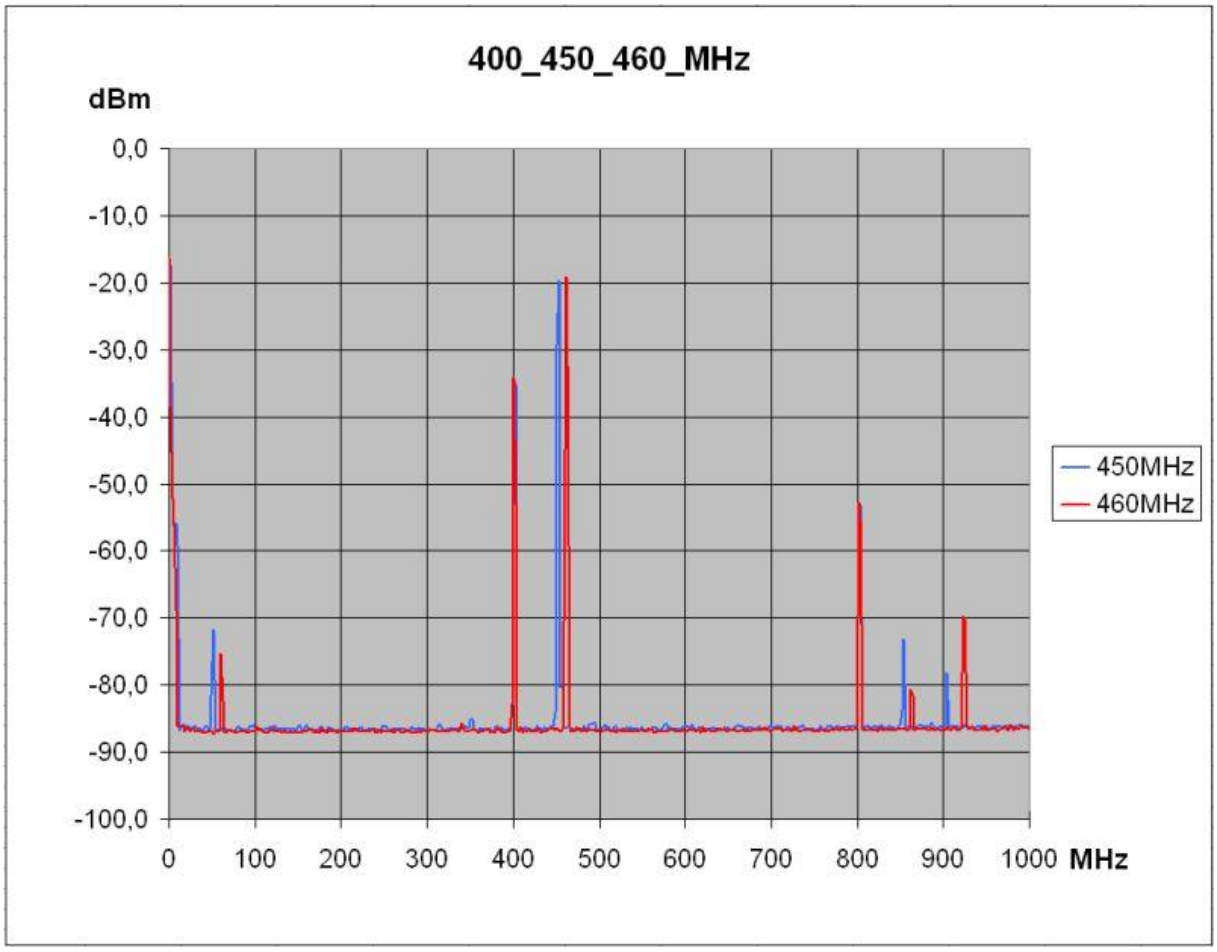


Bild_12: Ein Beispiel soll dies veranschaulichen. Auf diesem Diagramm werden ein 400- und ein 450-MHz-Signal (blau) bzw. im zweiten Kanal ein 400- und ein 460-MHz-Signal (rot) dargestellt.
Man kann deutlich die Mischprodukte und ersten Oberwellen der beiden Signale erkennen. Die Differenzfrequenzen 50- bzw. 60-Mhz, die Summenfrequenzen von 850- bzw. 860-MHz und die Oberwellen von 800-, 900- und 920 MHz. Die unregelmäßigen Linien von der Grundlinie nach unten stellen etwas schlecht das Rauschen dar, was aber hier keine weitere Bedeutung hat.

Das 4. Projekt:
Mehrere Geräte gleichzeitig am HPIB-Bus betreiben.

Dabei muss man unterscheiden, ob man z.B. zwei gleichartige Geräte (z.B. zwei Multimeter, um Strom und Spannung gleichzeitig zu erfassen) oder zwei unterschiedliche Geräte am Bus betreibt.

Bei Ersterem ist es am einfachsten. Man schreibt eine Routine, welche nach jedem gelieferten Datensatz einfach die nächste gewünschte HPIB-Adresse verwendet und somit vom nächsten Gerät die Daten empfängt.
Hier ist man aber spätestens an dem Punkt eingelangt, wo auch Routinen für die Speicherung der benötigten HPIB-Adressen benötigt werden. Für meine Zwecke hat es sich als Sinnvoll erwiesen, die Adressen jeweils sowohl hexadezimal als auch dezimal anzeigt, die Eingabe aber auf dezimal beschränkt. Man benötigt dann nämlich nur die zehn Ziffern und die Verwendung einer Tastatur mit nur 12 Tasten ist auch möglich. Da ich bis jetzt maximal 2 Geräte gleichzeitig betrieben habe, habe ich mich auch auf 2 HPIB-Adressen beschränken können.
Zum auseinanderhalten der Daten von den beiden Messgeräten hat sich bewährt, jeweils vor einem Datensatz (Messwert) z.B. ein Kennzeichen für das jeweilige Messgerät zu senden, z.B. in der Form „MA, Messwert vom Gerät A, MB, Messwert vom Gerät B, LineFeed“ liefert. Damit lassen sich die Messwerte dann bei der Auswertung eindeutig zuordnen.
Wenn benötigt, lassen sich auch Pausen einfügen, da man nicht immer die höchstmögliche Anzahl an Messungen pro Zeiteinheit wünscht – speziell bei Langzeitaufzeichnungen über mehr als 24 Stunden. Man beachte, dass z.B. Excel-2000 zwar rund 65000 Zeilen verwalten kann, bei der Darstellung eines Diagramms aber nur halb so viele Punkte verwenden kann!

Hat man zwei unterschiedliche Geräte am Bus (z.B. ein Multimeter und einen Frequenzzähler) so sind einige Besonderheiten zu beachten. Erstens werden diese dann auch mit unterschiedlichen Befehlen angesteuert – somit muss auch für jedes der Geräte nicht nur eine eigene Adressierung sondern auch eine eigene Befehlssequenz geschrieben werden. Des weiteren sind die gelieferten Datensätze unterschiedlich aufgebaut, was unter Umständen auch berücksichtigt werden muss.
Es gibt aber auch noch andere Komplikationsmöglichkeiten. Z.B. der Frequenzzähler HP-5345A. Dieser lässt sich nämlich (per Schalter an der Rückseite umschaltbar) auf zwei unterschiedliche Arten betreiben: Einerseits als normales Talker/Listener-Gerät, andererseits als reiner Talker. Als solcher sendet er immer (auch unaufgefordert!) Messwerte über die Schnittstelle und ist damit softwaremäßig nicht zu unterbrechen – das bedeutet, dass er in dieser Betriebsart mit keinem weiteren Gerät am Bus betrieben werden kann, es sei denn das weitere Gerät ist ausgeschaltet.

Bild_13:
Frequenzzähler HP-5345A
mit
4-GHz-Einschub HP-5354A

Es sei hier auch noch besonders darauf hingewiesen, dass man mittels des Mikroprozessors die Möglichkeit hat, beliebige Zeichen aus den Datensätzen herauszufi-

schen, sie durch andere ersetzen oder beliebige Zeichen einfügen kann. Dies kann in manchen Anwendungen die Arbeit gut unterstützen und sehr hilfreich sein.

Als Beispiel sei hier eine kombinierte Messung der Phasendifferenz zweier 10-MHz-Signale, geteilt durch 16, dargestellt. Die Teilung durch 16 mittels zweier 4-Bit-Synchronteiler hatte den Zweck, dass die Phasenverschiebung über einen Bereich von 16 Vollwellen des 10-MHz-Signals (=1600 nsec) gemessen werden konnte.

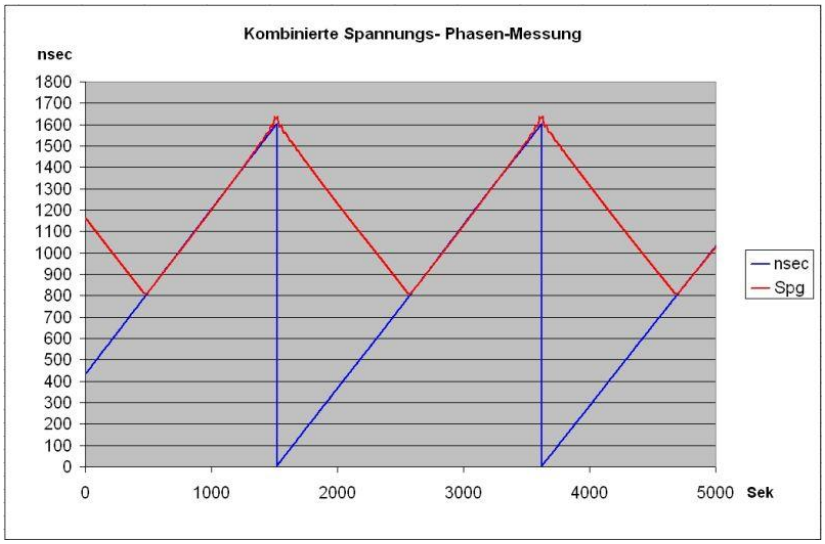
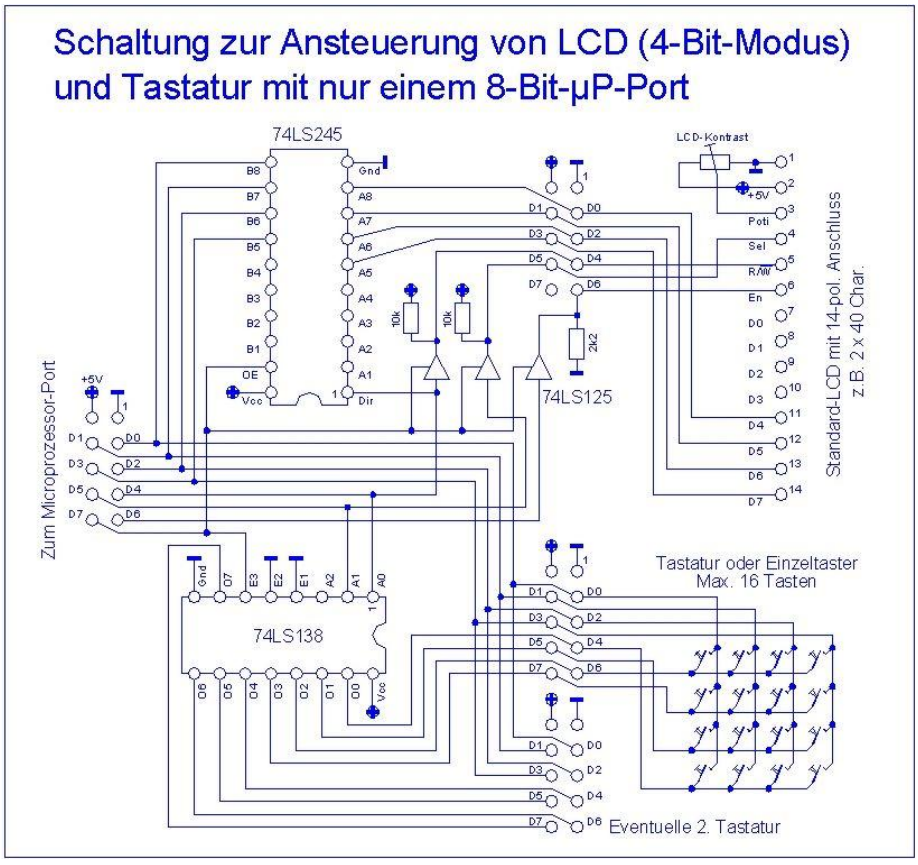


Bild_14: Die sägezahnförmige Kurve stellt die direkte Messung der Phasendifferenz mittels Frequenzzähler dar. Dieser hat allerdings „nur“ eine Auflösung von 2 nsec. Außerdem kann man erkennen, dass diese Kurve, wenn der Wert von 1600 nsec überschritten wird, wieder bei 0 nsec beginnt, da prinzipbedingt nicht mehr als diese 1600 nsec dargestellt werden können.

Die dreieckförmige Kurve stellt den Spannungsmittelwert des Ausgangs eines EX-OR-Gatters dar, an dessen beiden Eingängen wiederum die beiden 625-kHz-Signale anlagen. Allerdings wurden die Spannungswerte in Bezug auf Offset und Amplitude an die erste Kurve angepasst (deswegen auch keine Einheitenangabe zur Spannung), damit man den Zusammenhang in Bezug auf die Linearität der beiden Kurven zueinander erkennen kann. Wie man sieht, gibt es am oberen Ende der Kurven merkliche Abweichungen, aber der Grossteil der Kurven stimmt sehr gut überein. Außerdem kann man sehr deutlich erkennen, dass mittels der Spannungsmessung am EXOR prinzipbedingt nur die Hälfte des möglichen Bereiches erfasst werden kann.

Das 5. Projekt:
Der Messstellenumschalter

Für eine spezielle Anwendung im Zusammenhang mit Frequenznormalen wollte ich die Phasenbeziehung mehrerer 10-MHz-Quellen untereinander messen. Es sollte z.B. alle 5 Sekunden die Phasendifferenz von bis zu sechs weiteren Quellen zur ersten Quelle (hier Alpha-Quelle genannt) erfasst werden.
Somit wurde ein Messstellenumschalter benötigt, welcher direkt durch den Mikroprozessor angesteuert werden kann. Dies hatte jedoch einen entscheidenden Haken. Die dafür benötigten I/O-Ports standen nicht zur Verfügung und ein zweiter Prozessor war mir zu viel Aufwand. Die Lösung bestand in folgender Möglichkeit: Für die 16er-Tastatur und das LCD wurde jeweils ein Port belegt. Wenn es gelang, diese beiden an einem Port zu betreiben, wäre ein ganzer 8-Bit-Port für den Messstellenumschalter frei. Dies führte zur folgenden Schaltung.



Bill_15: Schaltplan der Zusatzplatine zur Ansteuerung von LCD und Tastatur an einem Port des Prozessors.

Zu bemerken ist, dass das Bit 7 vom Prozessorport die eigentliche Umschaltung zwischen Tastatur und LCD erledigt. Die Steuereingänge des 74LS125 und der Enable-Eingang des 74LS245 sind Low-aktiv, der E3-Eingang des 74LS138 ist High-aktiv. Somit ist bei Low an Bit 7 das LCD angesteuert, bei High die Tastatur. Diese kann auch aus Einzeltasten bestehen. Wenn benötigt, kann sogar eine zweite Tastatur angeschlossen werden (jede mit jeweils maximal 16 Tasten). Für manche Aufbauten ist somit z.B. eine 12er-Tastatur und zusätzlich einige einzelne Tasten möglich.

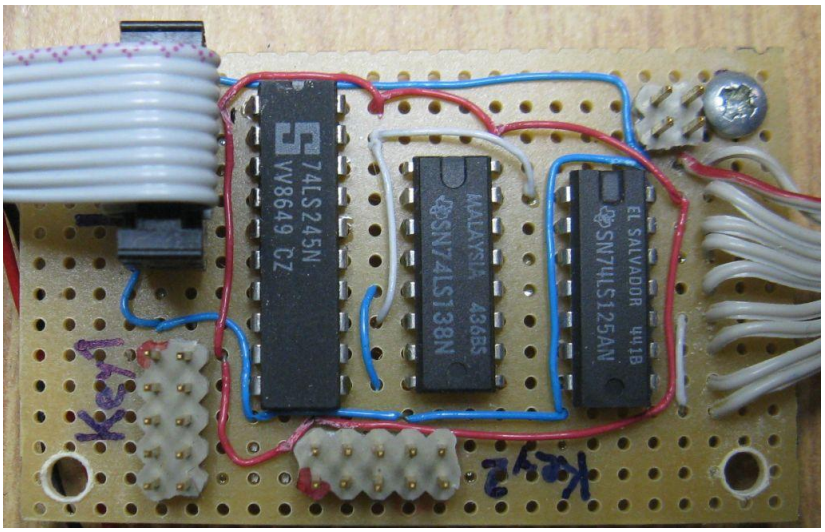


Bild_16: Foto der auf einer Lochrasterplatine aufgebauten Schaltung

Da nun der benötigte Prozessor-Port frei war, konnte der Messstellenumschalter aufgebaut werden. Dieser ist sehr einfach und besteht aus nur zwei TTL-IC´s und 6 Reed-Relais. Diese Relais-Typen haben den Vorteil, dass die Kontakte in gasgefüllten Glaskolben eingeschmolzen sind und deshalb auch bei kleinsten Signalströmen hohe Kontaktsicherheit aufweisen. Umschalter in CMOS-Technologie hätten sowohl Vorteile als auch Nachteile. Die Vorteile liegen hauptsächlich in der Geschwindigkeit und es gibt kein Kontaktprellen. Dafür nimmt man aber den Nachteil in Kauf, dass es keine getrennte „Masse“ für die Messsignale gibt. Habe mich deshalb für die Relais entschieden. Das Kontaktprellen kann durch softwaremäßige Verzögerung der Messungen egalisiert werden und die Schaltgeschwindigkeit der Relais hat für mich keinen nennenswerten Einfluss.

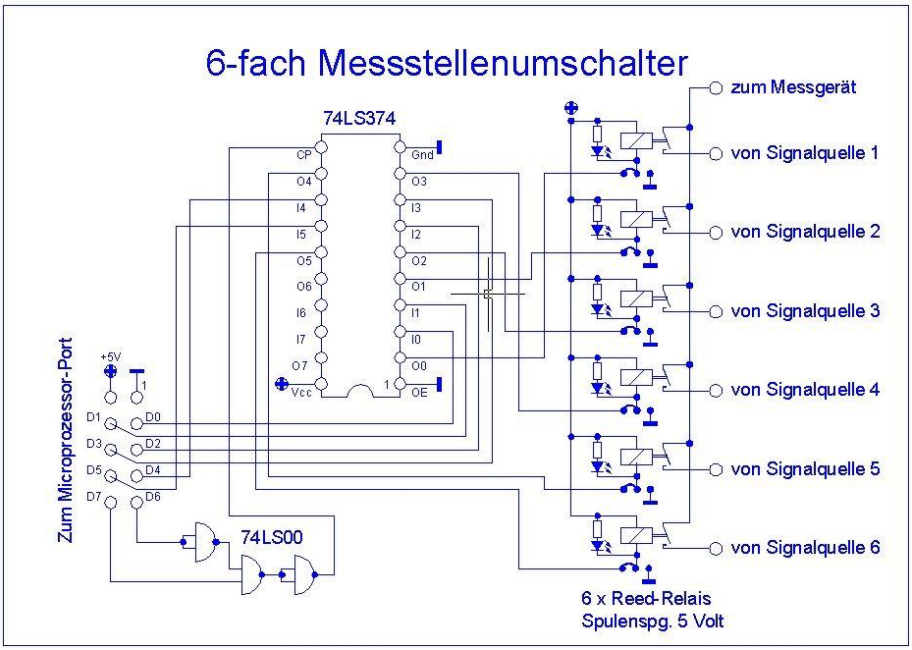


Bild_17: Schaltplan des Messstellenumschalters.

Der 74LS00 dient nur dazu, dem Umschalter eine von zwei möglichen „Adressen“ zuzuweisen (Bit 7 muss High sein, damit er angesprochen werden kann). Das ist nur

für eventuelle spätere Erweiterungen gedacht, da kann dann mit „Bit 7 = High“ eine weitere Schaltung adressiert werden.
Bit 6 des Prozessor-Ports ist normal High und wird nur zur Übernahme einer neuen Einstellung (Ansteuerung eines anderen Relais) kurz auf Low und dann wieder auf High gelegt. In dieser Zeit muss dann natürlich Bit 7 High sein. Erst dies ermöglicht die Verwendung des Ports auch für Anderes.
Die parallel zu den Relais geschalteten LED´s mit den zugehörigen Widerständen und die im Schaltbild unterhalb der Relais angeordneten Jumper haben sich in der Praxis sehr bewährt. An Hand der LED´s sieht man sofort, welches Relais durchgeschaltet ist, speziell zur Kontrolle und beim Schreiben der zugehörigen Software hat das große Vorteile. Nicht zu vergessen, dass es auch sehr hilfreich ist, wenn man „sieht“, dass die Relais schalten, denn hören tut man diese fast nicht. Die Jumper dienen dazu, beliebige Relais zwangsweise durchzuschalten oder auch für Tests kurzfristig mehrere Relais parallel durchschalten zu lassen. Müsste man für solche Fälle jedes Mal die Verkabelung ändern wäre dies ein ungleich höherer Aufwand.

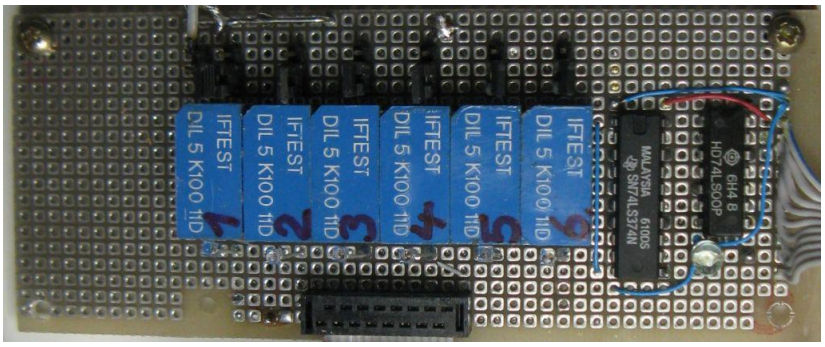


Bild_18:
Foto des auf einer Lochrasteplatine in Fädeltechnik aufgebauten Messstellenumschalters

Softwaremäßig habe ich den Messstellenumschalter bis jetzt nur für den Frequenzzähler HP5345A benötigt. Dieser hat, wie schon beschrieben, auch den „Dauer-Messwertausgabe-Modus“, welchen ich hier nütze.

MSU-Port = Jener Port des Mikroprozessors, an welchem der Messstellenumschalter angeschlossen ist.

Programmablauf:

a.) Merkerbit für „erstes LineFeed“ rücksetzen
b.) NRFD auf High
c.) Messstelle 1 auswählen vorbereiten („FE“ hex an MSU-Port ausgeben)
d.) Bit 6 von MSU-Port auf Low setzen (Damit wird Messstelle 1 ausgewählt)
e.) Bit 6 von MSU-Port auf High setzen
f.) Verzögerungszeit 10 ms einfügen (Eventuelles Kontaktprellen übergehen)
g.) Unterroutine „Messwert von Messgerät holen“ aufrufen
h.) Zeichen „M“ für „Messung“ an RS232 ausgeben
i.) Zeichen „A“ für „Messstelle 1“ an RS232 ausgeben
j.) Messwert an RS232 ausgeben

Ab hier wiederholt sich das Ganze mit den Messstellen zwei bis sechs. Nach dem „M“ werden dann aber die Buchstaben „B“ bis „F“, je nach Messstelle ausgegeben.

Dies erleichtert die Auswertung am PC. Abschließend wird noch eine Pause von knapp 5 Sekunden eingefügt. Dann wird wieder mit Messstelle 1 begonnen.

Unterroutine „Messwert von Messgerät holen“:

a.) RAM-Buffer auf „Start-Adresse“ setzen
b.) Warten bis DAV = Low
c.) Wenn Merkerbit für „erstes LineFeed“ gesetzt, springe nach g.)
d.) Wenn nicht „LineFeed“, springe nach b.)
e.) Merkerbit für „erstes LineFeed“ setzen (das erste LineFeed zeigt das Ende einer gerade laufenden Messwertausgabe an)
f.) Sprung nach b.)
g.) Byte in den RAM-Buffer laden
h.) Wenn nicht „LineFeed“ dann Sprung nach b.)
i.) Rücksprung aus Unterroutine (das zweite LineFeed zeigt das Ende der aufgezeichneten Messwertausgabe an)

Damit habe ich die Möglichkeit, Messungen mit einer Dauer von bis knapp über 45 Stunden durchzuführen und z.B. als Excel-Diagramm darzustellen. Die 5-Sekunden-Wartezeit sollte natürlich veränderbar sein, da bei anderen Einsätzen eine andere Zykluszeit nötig sein kann.

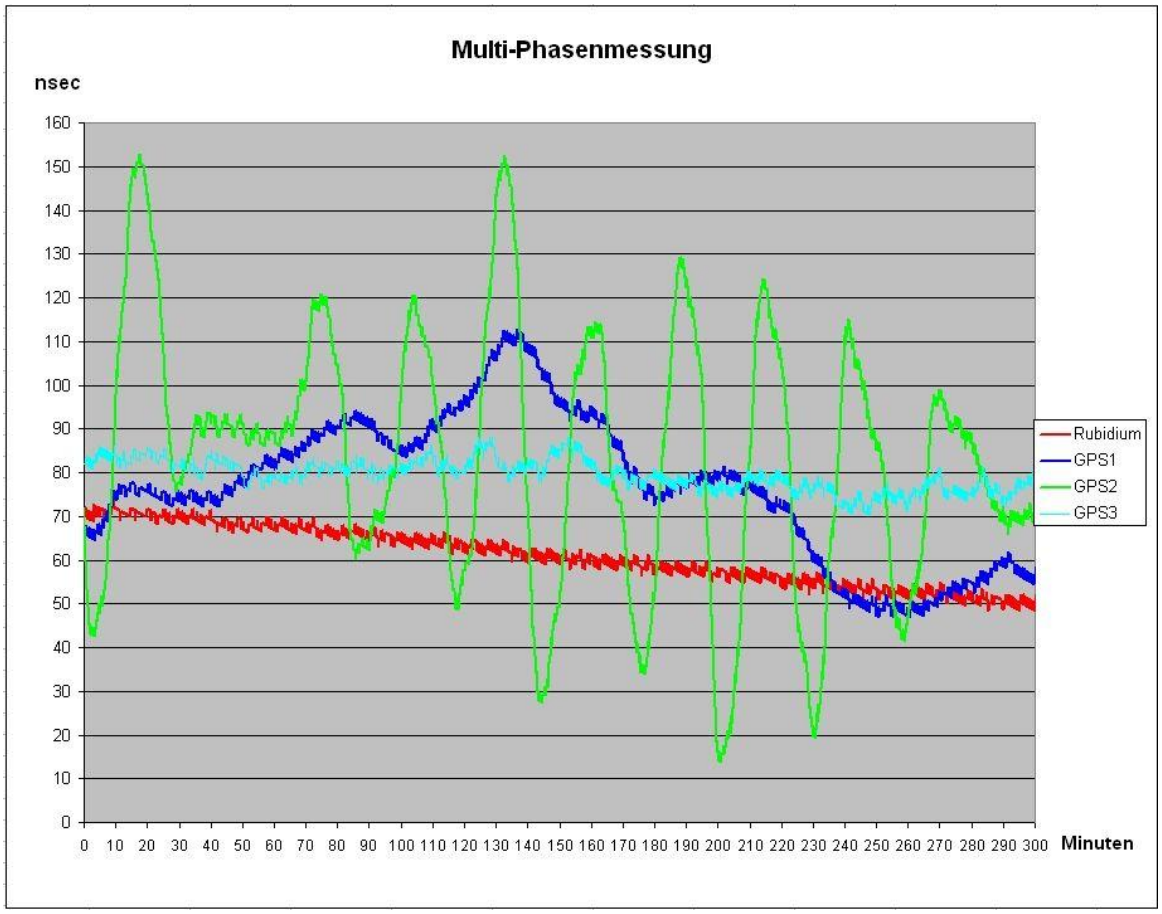


Bild_19: Beispiel eines Diagramms mit 4 aufgezeichneten Kanälen über 5 Stunden Dauer