Rafael Khan

Embedded System als Human- Machine- Interface mit flexibler Menüstruktur und CAN Anbindung an ein digitales Schutzrelais

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Jochen Schneider
Zweitgutachter : Prof. Dr. Franz Schubert

Abgegeben am 20. August 2009

Rafael Khan

Thema der Diplomarbeit

Embedded System als Human- Machine- Interface mit flexibler Menüstruktur und CAN Anbindung an ein digitales Schutzrelais.

Stichworte

HMI, Graphic User Interface, CAN, flexible Struktur, eingebettetes System, Touchscreen, ARM7, LCD

Kurzzusammenfassung

Bei dieser Arbeit geht es um den Entwurf eines eingebetteten Systems mit grafischer Benutzeroberfläche auf Basis des LPC2478 von NXP. Der Mikrocontroller basiert auf einem 32 Bit ARM7TDMI-S. Das Interface wird nur über CAN mit dem Rest des Systems verbunden, wobei vier der darzustellenden Seiten komplett über ein externes Tool generierbar sind. Die Teile des Menüs können flexibel gehalten werden, sodass über einfache Parameter Menüpunkte und Unterpunkte variable sind. Gesteuert wird über eine Tastatur und ein berührungsemfindliches Display. Die Entwicklungsumgebung für den Software-Teil ist µVision3 der Firma Keil.

Rafael Khan

Title of the paper

Embedded System as Human- Machine- Interface with flexible menu structure and CAN connection to a digital network relay.

Keywords

HMI, Graphic User Interface, CAN, flexible Structure, Embedded System, Touch screen, ARM7. LCD

Abstract

This Paper is about the draught of an Embedded System as a graphic user interface with the LPC2478 (NXP) as basis. The microconroller has a 32 Bit ARM7DMI-S Core implemented. The interface has only a CAN connection to the rest of the system. Four of the pages to be displayed are completely gene-rateable about an external tool. The other parts of the menu have to be adaptable that its structure can be changed about a simple parameter. The Display is controlled over a keyboard and a touchscreen. The developing environment for the software part is µVision3 of the company Keil.

Inhaltsverzeichnis

I Einleitung

Basis dieser Arbeit wird eine Erweiterung der SYMAP®-Familie sein. Dieses Gerät übernimmt die Überwachung von Netzversorgungen im Nieder-, Mittel-, und Hochspannungsbereich und kann diese bei Netzschwankungen über einen Leistungsschalter abschalten. Gleichzeitig kann ein Dieselgenerator angeworfen und gesteuert werden um den möglichen Energiebedarf zu decken. Eine Übersicht ist in: Anhang B5: Präsentation Symap (CD) zu finden.

Das SYMAP® existiert in drei Serien, die Grundeinheit SYMAP®-Y, die Basiseinheit SYMAP®-X, sowie der Basiseinheit SYMAP®-BC erweitert mit LED-Anzeige, Ereignisdatenschreiber, Erweiterungskarte, Power Management und Dieselsteuerung.

Die SYMAP® Familie soll nun um eine weitere Serie, dem SYMAP®-Compact erweitert werden, wobei dies eine komplette Neuentwicklung sein wird. Der primäre Aufbau des SYMAP®-Compact wird aus drei ARM7 Controllern bestehen. Die Aufgaben der Controller werden Einlesen von Messwerten, Steuerung und Interaktion mit dem Benutzer sein.

Dieses Projekt befasst sich mit der Entwicklung und Programmierung der Displayeinheit bzw. Grafikkarte. Es gilt somit eine Benutzerschnittstelle für ein digitales Schutzrelais im Bereich Nieder-, Mittel- und Hochspannung zu entwickeln.

Basis wird die Entwicklung einer Platine sein, welche mit dem LPC2478 Mikrocontroller der Firma NXP bestückt wird. Es gilt ein TFT anzusteuern und eine Menüführung über eine Tastatur zu erstellen. Die Kommunikation zur Displayeinheit soll intern über einen CAN-Bus stattfinden, wodurch ermöglicht wird diese Einheit auch getrennt vom Rest des Systems zu verbauen.

Die grafische Benutzeroberfläche soll flexibel gehalten werden. Individuelle Seiten werden über ein externes Tool am PC generiert und über USB auf das Schutzrelais geladen. Auch soll die Menüstruktur über Parameter konfigurierbar sein, um das Menü auf einfach Weise anpassen und erweitern zu können. Menüpunkte sollen ein- und ausschaltbar sein und um weitere Punkte ergänzt werden können. Es soll möglich sein ganze Registerkarten, aber auch einzelne Elemente aus Listen, zu deaktivieren.

Das folgende Kapitel wird damit beginnen einige Grundlagen zum ARM (Acorn Risc Machine) Prozessor zu erläutern, gefolgt von einer Einführung zu dem genutzten Mikrocontroller und dem eingesetzten Evaluationboard. Es wird die Entwicklungsumgebung vorgestellt und mit der Vorstellung der eingesetzten Softwarebibliothek abgeschlossen.

Kapitel drei beschäftigt sich nun mit der eingesetzten Hardware und der Entwicklung der Platine. Es wird auf das genutzte LCD eingegangen und veranschaulicht warum dieses gewählt wurde. Es wir gezeigt wie die Peripherie konfiguriert wird.

In Kapitel vier wird die gesamte Software vorgestellt. Es wird veranschaulicht warum und wie das System Realisiert wurde und es werden verschieden Lösungsansätze diskutiert.

Die letzten Kapitel beschäftige sich mit dem eingesetzten Busprotokoll und der Speicherverwaltung.

Am Ende der Arbeit befindet sich noch eine Zusammenfassung, sowie ein Ausblick auf die Weiterführung des Projektes.

2 Grundlagen

2.1 ARM

1983 wurde das ARM Design von einem Team unter der Leitung von Roger Willson und Steve Furber gestartet. Die Entwicklung ging von einem Entwicklungsprojekt des Englischen Computerherstellers Arcon aus.

Die Zielsetzung war, nicht wie die anderen Hersteller auf Produkte der Firma Intel und Motorola zurück zu greifen, sondern einen eigenen Prozessor mit geplanten 32 Bit und 4 MHz zu entwickeln.

Der Prozessor ist dann mit 8 MHz und dem ARM2 in Serie gegangen. Die Serie wurde 1989 mit dem ARM3 fortgesetzt was dazu führte, dass im Jahre 1990 Acorn zusammen mit Apple und VLSI Technologie das Unternehmen Advanced RISC (Reduced Instruction Set Computer) Machines Ltd. In GB gründeten.

Die ARM Architektur zeichnet sich durch einen Befehlssatz aus, der eine Kompakte Umsetzung eines ASCI (American Standard Code for Information Interchange) Designs ermöglicht.

Advaced RISC Machines LTD hat an interessierte Halbleiterhersteller Lizenzen verkauft, die eine Implementierung des ARM-Kerns in ihre Chips erlaubt. Dazu gehören unter anderem: Analog Devices, Atmel, Freescale (ehemals Motorola), HP, IBM, Infineon, Intel, NEC, NXP (ehemals Philips), Samsung,Texas Instruments.

Aufgrund der geringen Leistungsaufnahme werden ARM Prozessoren in vielen eingebetteten Systemen verwendet, unter anderem in PDAs, Handys und Spielkonsolen.

Die ARM Prozessoren arbeiten mit einer RISC (Reduced Instruction Set Computing[1]) Architektur, verzichten somit auf komplexe Befehle. Als Konsequenz ergibt sich ein schnellerer Prozessor, welcher zügig nach einer Unterbrechung (z.B. Interrupt) weiterarbeiten kann. [14]

ARM7

Bei dem ARM7 handelt es sich um die siebte Generation des Prozessors. Der hier eingesetzte ARM7TDMI besitzt typischerweise eine große Anzahl an Registern. Hierbei handelt es sich um 37 32 Bit breite Register. Von den 37 Registern zur Verarbeitung der Daten sind sechs Register reine Statusregister. Im Adressregister ist die Speicheradresse enthalten auf die, die CPU zugreifen soll. Es handelt sich um ein 32 Bit System, die Speicherstruktur ist jedoch nur auf 8 Bit ausgelegt, somit wird die Adresse immer um 4 Byte erhöht. Der 32x8 *Multiplikator* ermöglicht, durch seine fest verdrahtete Logik, schnelle Multiplikationen. Der ebenso fest verdrahtete *Barrel-shifter* bietet als Schieberegister die Möglichkeit zum verschieben eines oder mehrerer Bits. Die *ALU* (Arithmetic Logical Unit) führt als Kern alle nötigen Rechenoperationen durch. In der *Instruction decoder and control logic* werden die Befehle für die Ausführung decodiert. Die Register *WDATA* und *RDATA* dienen als Schnittstelle zu den Daten (s. Bild 2-1). Es ist eine Speicherverwaltung bis 4 GB möglich.

1 engl. für *Rechnen mit reduziertem Befehlssatz.*

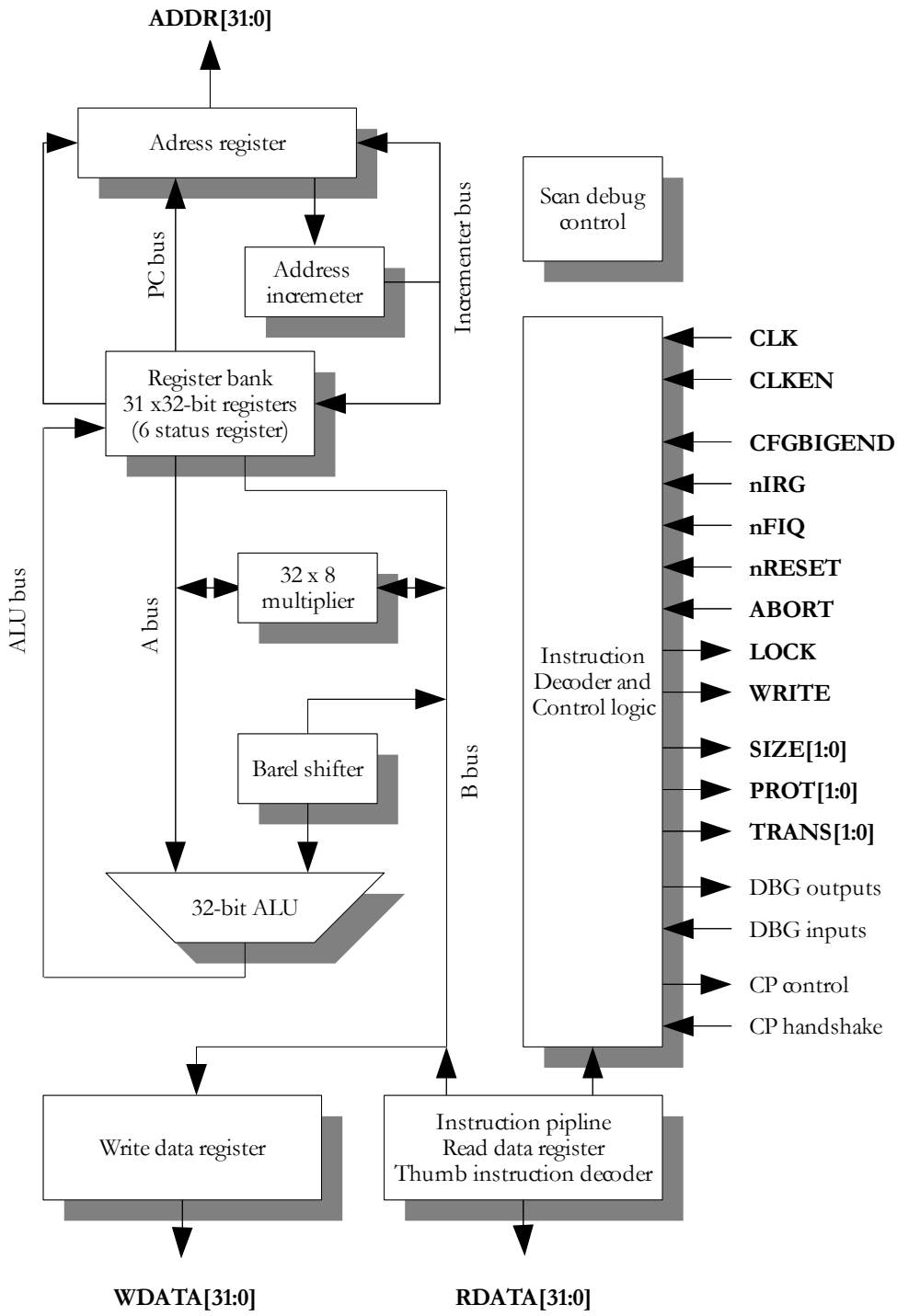


Bild 2-1 Aufbau des ARM7DMI-S. (Technische Referenz, Rev. r4p3)

Nach dem *Load and Store (*Daten-Speicherzugriffe sind nur durch spezielle Befehle erlaubt*)* Prinzip werden die Daten für die Verarbeitung vorbereitet. Somit müssen vor einer Rechenoperation alle Werte in Register umkopiert werden. Es existiert keine Adressarithmetik auf CPU-Basis. Die 32 Bit RISC Architektur unterstützt ein dreifaches Pipelining zur schnelleren Abarbeitung des Codes. Sofern die Pipeline gefüllt ist werden die Daten in einem Takt geholt, die zuvor geholten Daten dekodiert und die dekodierten Daten ausgeführt. [5]

Im ARM7TDMI-S sind zwei Befehlssätze enthalten. Ein 32 Bit ARM Befehlssatz mit 556 Befehlen, sowie einen 16 Bit *Thumb* Befehlssatz mit 48 Befehlen.

Für die Verwendung des Thumb Befehlssatzes wird ein eingebauter *Thumb instruction decoder* verwendet, welcher die Befehle in Echtzeit umwandelt. Wird der Thumb Befehlssatz benutzt, so werden die Instruktionen zunächst von dem eingebauten Thumb instruction decoder in den 32 Bit ARM Befehlssatz umgewandelt und erst danach ausgeführt. Dieses geschieht in Echtzeit, ohne Geschwindigkeitsverlust. Der Thumb mode hat so alle Vorteile einer 32 Bit CPU wie 32 Bit Adressbereich, 32 Bit Register, 32 Bit Schieberegister und ALU sowie 32 Bit Speichertransfer.

Dieser reduzierte Befehlssatz ermöglicht es nun die Codegröße um ca. 1/3 zu reduzieren und bei einem 16 Bit Speichersystem die Durchsatzrate um 160% zu erhöhen. Es besteht jedoch der Nachteil das die Anzahl der verfügbaren Register reduziert wird. Es stehen nur die Register r0 bis r7 zur Verfügung. Der Stackpointer (SP) wird in Register r13 abgelegt. Register r15 beinhaltet nun den Programm Counter (PC). [3]

2.2 LPC2478

Der LPC2478 Mikrocontroller von NXP basiert auf einem ARM7TDMI-S Kern. Der Prozessor kann mit bis zu 72 MHz getaktet werden, wofür ein 12 MHz Quarz verwendet wird. Für die RTC (Real Time Clock[2]) ist ein 32,768 kHz Quarz vorgesehen. Neben der industriellen Anwendung findet man die LPC24xx Reihe unter anderem in medizinischen Systemen, Mobiltelefonen, PDAs, Protokoll Konvertern und Routern. [7]

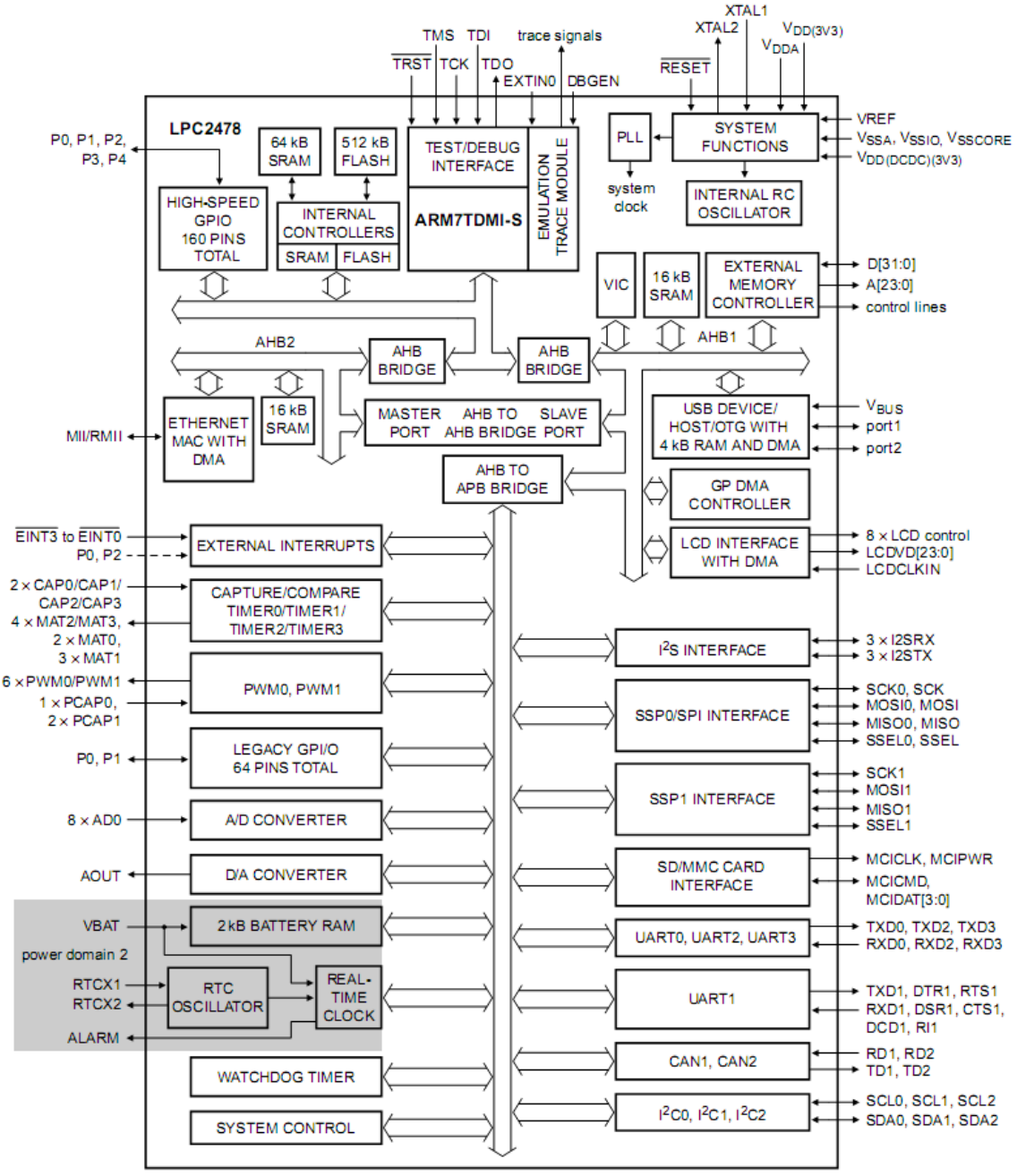


Bild 2-2 LPC2478 Blockschaltbild (Datenblatt LPC2478).

2 engl. für *Echtzeituhr*.

Bild 2-2 zeigt den gesamten Aufbau des Controllers als Blockschaltbild. Es stehen dem Prozessor insgesamt 512 kB Flash-Speicher zur Verfügung, auf den über 128 Bit zugegriffen wird. Die internen 96 kB RAM sind aufgeteilt in: 64 kB erreichbar über den lokalen Advanced High-performance Bus (AHB) zur schnellen Benutzung, 16 kB für das Ethernet Interface, 16 kB für allgemeine DMA[3] (Dircet Memory Access) Zugriffe oder die USB Schnittstelle und 2 kB für Daten die bei einem Stromausfall über eine Batteriebepufferung gesichert werden können.

Für den Betrieb des Prozessors ist nur eine einzige Spannung von 3,3 Volt (3,0 bis 3,6 Volt) notwendig. Aufgrund des Power Management Moduls, welches drei verschiedene Zustände zulässt, ist eine enorme Stromreduzierung möglich. Der Prozessor kann die Zustände Idle (Leerlauf), Sleep (Schlafmodus) und Power-down (Herunterfahren) annehmen. Aus den verschiedenen Stromsparmodi kann der Prozessor über Interrupts aktiviert werden.

Die Programmierung erfolgt über die serielle Schnittstelle (UART), oder über das Test/ Debug Interface (JTAG). Um den Prozessor über die serielle Schnittstelle zu Programmieren ist ein extra Programm notwendig (z.B. Flash Magic von Embedded Systems Academy). Hierfür muss zunächst der Boot-Loader gestartet werden. Dazu muss die Tastenkombination *RESET* und *EINT0* gedrückt werden. Nun ist der Controller empfangsbereit für Daten oder Befehle. Über das JTAG-Interface kann direkt aus der Entwicklungsumgebung programmiert werden und bietet neben einer höheren Geschwindigkeit auch die Möglichkeit des Echtzeit-Debuggings.

Der LPC 2478 wird in den Bauformen TFBGA208 (Ball Grid Array) und LQFP208 (Quad Flat Pack) geliefert. [6]

2.2.1 Peripherie

Timer

Der LPC2478 beinhaltet vier 32 Bit Zähler (Timer). Die Timer können entweder über einen System gesteuerten Takt, oder über einen externen Takt gesteuert werden. Die optionale Generierung von Interrupts kann zeitgesteuerte Ereignisse ausführen, basierend auf vier *Match* Registern. Des weiteren sind vier *Capture* Eingänge vorhanden.

GPIO

Es sind fünf Schnittstellen für universelle Ein- und Ausgaben vorhanden. Von diesen fünf Schnittstellen sind drei High-Speed GPIO Bänke und zwei normale GPIO Bänke die jedoch auf High-Speed umgeschaltet werden können. So ist es möglich bis zu 160 Ein- und Ausgänge zu benutzen, die intern auch mit Pullup oder Pulldown Widerständen versehen werden können.

Interrupt Controller

Der Prozessor hat zwei Interrupt Eingänge. Den Interrupt Request (IRQ) sowie den Fast Interrupt Request (FIQ). Der Vectored Interrupt Controller[4] (VIC) besitzt 32 Eingänge die dynamisch nach Priorität verwaltet werden können. Der FIQ besitzt immer die höchste Priorität. Der Controller besitzt die Möglichkeit vier externe einzelne Interrupts, sowie mit den Portbänken Null und Eins einen Bank-Interrupt zu generieren.

3 Direkter Zugriff auf den Speicher.

4 Verwaltet und verteilt die Interrupts.

Watchdog

Der Zweck des Watchdog ist es, den Mikrocontroller nach einer bestimmten Zeit neu zu starten, falls dieser sich in einem fehlerhaften Zustand befindet. Wenn aktiviert, wird der Reset ausgelöst falls der Watchdog nicht mehr „*gefüttert*“ wurde.

PWM (Pulsweitenmodulation)

Die PWM basiert auf dem Standard Timer und beinhaltet alle Features eines Timers. Der Unterschied besteht darin, das nur die PWM Funktionalität nach außen geführt wurde.

A/D Umsetzer

Der ADC hat eine Auflösung von 10 Bit (1024 Stufen) und acht Kanäle, die über einen Multiplexer verknüpft sind. Die Konversationszeit beträgt 2,44 µs.

D/A Umsetzer

Der digital zu analog Umsetzer hat eine Auflösung von 10 Bit. Eine Umsetzung dauert hierbei 2,5 µs.

Real-time Clock

Die RTC beinhaltet eine Reihe von Zählern für Zeitmessung und funktioniert auch wenn das System aus ist. Die RTC kann an einen 32,786 kHz Oszillator, oder über einen Prescaler (reduziert einen hohe Frequenz auf eine geringere) an den AHB Takt, angeschlossen werden. Der separate Pin VBAT für die Spannung kann an den Rest des System, oder an eine Batterie angeschlossen werden.

SPI (Serial Peripheral Interface)

Der LPC2478 enthält einen SPI Controller. SPI ist eine vollduplexfähige Schnittstelle, die dazu entworfen wurde, mehrere an einen Bus angeschlossenen Master und Slaves zu verwalten. Es kann immer nur ein einziger Master und ein einziger Slave zur selben Zeit auf dem Bus kommunizieren.

UART (Universal Asynchronous Receiver Transmitter)

Es sind drei einfache UART Schnittstellen mit Sende- und Empfangsleitungen und ein Fullmodem mit zusätzlichen Leitungen (DTR, RTS, CTS, DSR, DCD, RI) vorhanden.

CAN (Controller Area Network)

Es sind zwei CAN Controller und Busse nach Spezifikation 2.0B, ISO 11898-1 vorhanden. Datenraten bis zu 1 MBit/s sind auf jedem Bus möglich sowie ein Akzeptanzfilter für 11 Bit und 20 Bit Identifier.

weitere Peripherie

USB Interface, SD/MMC Karteninterface, Ethernet Block, SSP, I^2S, I^2C. [5]

2.2.2 CAN

Bei CAN handelt es sich um ein Multimaster-Feldbussystem, welches 1983 von Bosch entwickelt wurde. 1987 wurde zusammen mit Intel diese Projekt vorgestellt. 1993 war es dann soweit, dass das CAN-Protokoll als Internationaler Standard ISO 11898 für hohe Datenübertragungsraten von mehr als 125 kBit/s veröffentlicht wurde. Später dann auch als DIN ISO 11898. Der CAN Bus findet hauptsächlich Anwendung in der Autoindustrie und in der Industrie allgemein. [1]

Physikalische Grundlagen

Es gibt zwei Gruppen in die CAN unterteilt wird. Den hightspeed CAN mit Datenraten bis zu 1 MBit/s bei einer maximalen Buslänge von 40 m und den lowspeed CAN mit einer Datenraten von 125 kBit/s bei einer maximalen Leitungslänge von 1000 m (s. Tab. 2.1).

Tab. 2.1 CAN Baudrate.

Leitungslänge	Baudrate
0- 40 m	1 Mbit/s
40- 300 m	200 kBit/s
300- 600 m	50 kBit/s
600- 1000 m	50 kBit/s

Alle Geräte sind durch einen CAN Transceiver über zwei Leitungen mit dem BUS verbunden. Die Leitungen müssen an den Enden mit jeweils 120 Ω abgeschlossen werden. Bei den Leitungen handelt es sich idealerweise um zwei verdrillte Einzeladern mit Abschirmung (STP). Bei den Pegelzuständen auf dem BUS gilt das High-Signal mit einer Spannungsdifferenz von 3,5 V oder mehr als rezessiv und das Low- Signal mit einer Spannungsdifferenz von 1,5 V oder kleiner als dominant (s. Bild 2-3). Das ganze bezogen auf eine Spannung von 2,5 V.

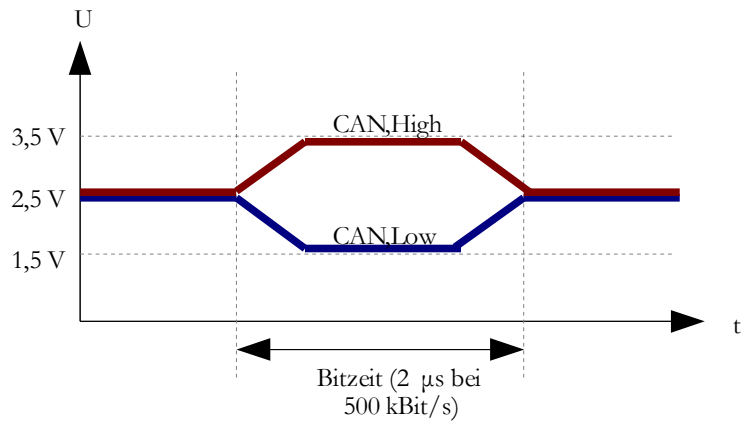


Bild 2-3 CAN Spannungspegel.

Die Priorität des Signals resultiert aus der internen Beschaltung des CAN Transceivers. Der Transceiver bestimmt auch die maximal Anzahl von Teilnehmern (32, 64, 110, mit Einschränkungen 128). Es ist jedoch ein Kopplung über Repeater möglich, was noch mehr Teilnehmer ermöglicht.

Ein CAN-Bus kann auf drei verschiedene Arten aufgebaut werden. Im Infotainmentbereich wird hauptsächlich der Ringbus eingesetzt, in der Kfz-Elektrik der Sternbus. An dieser Stelle wird ein linear aufgebauter Bus (s. Bild 2-4) gezeigt. Der linear aufgebaute Bus wird im Symap Compact verwendet. [1]

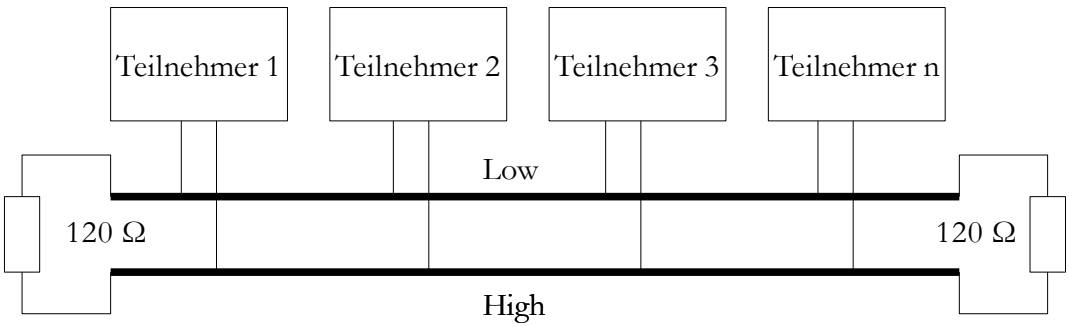


Bild 2-4 Struktur Bus – Topologie.

Datensicherung

Bei CAN sind alle Teilnehmer auf dem Bus gleichberechtigt. Folglich kann jeder Teilnehmer ohne Aufforderung auf den Bus schreiben. Die damit im Zusammenhang stehenden Kollisionen werden jedoch von der Hardware abgefangen. Eine Adressierung der Teilnehmer erfolgt nicht. Jedes Datenpaket wird mit einem eindeutigen Identifier versehen. Jeder Controller am Bus kann nun entscheiden ob das Paket weiterverarbeitet wird oder nicht. Damit die Arbitrierung funktioniert muss jeder Identifier einem Sender eindeutig zugewiesen werden, somit können keine zwei Sender mit dem selben Identifier senden.

Laut Spezifikation werden zwei unterschiedliche Formate definiert:

- 11 Bit Identifier, der auch *Base Frame Format* genannt wird (CAN 2.0A)
- 29 Bit Identifier, der auch *Extended Frame Format* genannt wird (CAN 2.0B)

Aufgrund der Dominanz des Low-Signals resultiert eine höhere Priorität der kleineren Identifier. Werden zwei Pakete gleichzeitig übertragen erfolgt ein Überschreiben mit der geringeren ID und das niedriger priorisierte Paket muss neu gesendet werden. Zur Fehlersicherheit kann des weiteren gesagt werden, dass ein Bit Monitoring, ein Messsage Frame Check, ein CRC Check und eine Hemming-Distanz von sechs existieren. Zusätzlich muss jede Nachricht quittiert werden. [2]

In diesem Projekt wird lediglich das *Base Frame Format* Anwendung finden. Die 2048 Identifier fallen zwar geringer aus, jedoch ist der damit im Zusammenhang stehende Overhead auf dem Bus auch kleiner.

Telegramm Aufbau

Die zu übertragenden Nutzdaten werden als sog. Frames verschickt. Dabei werden die Daten in 8 Byte große Anteile zerteilt und mit Identifier sowie Kontrollbits versehen. Dabei existieren vier verschiedene Arten von Frames:

- Daten- Frame : Überträgt die Nutzdaten
- Remote- Frame : Fordert ein Datenframe an
- Error- Frame : Sendet einen erkannten Fehler
- Overload- Frame : Erzeugt eine Pause zwischen Daten- und Remote- Frame

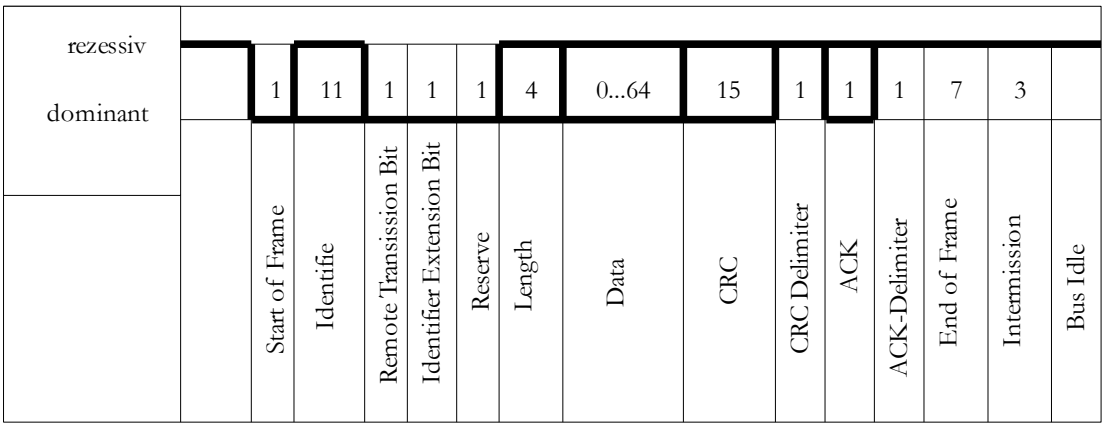


Bild 2-5 CAN Datentelegramm. – Standard.

Bild 2-5 zeigt den Aufbau eines *Standard Frames*. Hierbei ist zu beachten das das *Startbit* (Start of Frame) immer dominant ist. Das *Identifier Extension Bit* ergibt zusammen mit der Länge und dem *reservierten Bit* das Kontrollfeld. Das Datenfeld kann zwischen 0 und 8 Byte variieren. *CRC* und *CRC-Delimiter* (Trenner) bilden das Prüfsummenfeld, *ACK* (Bestätigungs-Slot) und *ACK-Delimiter* bilden das Bestätigungsfeld.

Ein *Remote-Frame* fordert von einem anderen Teilnehmer Daten an, wobei das rezessive *Remote Transmission Bit* gesetzt sein muss. Dies verhindert den Verlust der Arbitrierung bei gleichzeitigem senden eines Datentelegramms. Wichtig ist hierbei die korrekte Angabe über die Länge der Nutzdaten.

Ein *Error-Frame* besteht aus zwei Feldern, wobei das erste Feld sechs dominante oder rezessive Fehler-Flags beinhaltet und das zweite, acht rezessive Fehler-Trenner zu einer möglichen nächsten Nachricht. Nun wird zwischen einem Fehler-aktiven und einem Fehler-passiven Telegramm unterschieden, wobei das Fehler-aktive Telegramm durch das Senden von sechs dominanten Bits bewusst die Regeln des Bitstuffing verletzt um die Teilnehmer auf dem Bus neu zu initialisieren.

Das *Overload-Frame* hat den Aufbau eines Fehlertelegramms es wird gesendet wenn ein Knoten noch nicht bereit ist neue Daten zu empfangen. [2]

2.2.3 LCD Controller

Ein Grund für die Auswahl des LPC2478 war der fest integrierte LCD-Controller. Da kein interner Bildspeicher vorhanden ist, verfügt der Controller über ein AHB Interface um direkt auf den externen RAM zuzugreifen, welcher dem Controller als Bildspeicher dient. Zur Pufferung der Bilddaten stehen 16 programmierbare FIFOs mit einer Breite von 64 Bit zur Verfügung, welche die langsame Geschwindigkeit des externen Speichers ausgleichen.

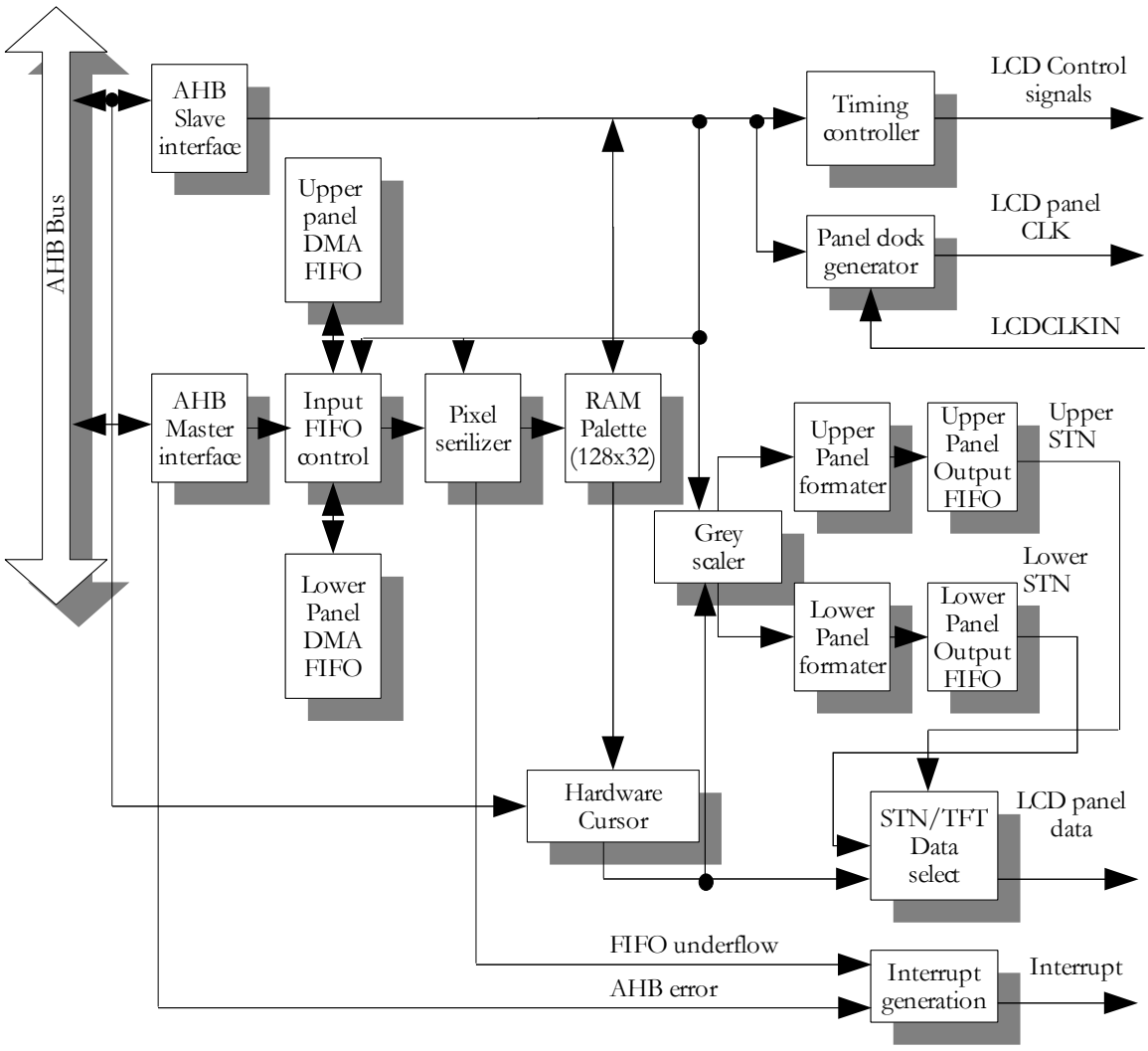


Bild 2-6 Interner Aufbau des LCD Controllers im LPC2478.

In Bild 2-6 ist der interne Aufbau zu erkennen. Der Controller unterstützt eine ganze Reihe von Displaytypen. Dazu gehören *Singel* und *Dual Panel STN* (Super Twisted Nematic) Farbdisplays mit 4 oder 8 Bit Interface. TFT (Thin Film Transistor) Farbdisplays mit Auflösungen von 320x200, 320x240, 640x200, 640x240, 640x480, 800x600 und 1024x768 Pixel. Ein Hardwarecursor steht für *Singel Panel Displays* zur Verfügung. Für den Monochrombereich kann der Entwickler mit 15 Graustufen arbeiten. Bei einem *STN Display* stehen ihm 3375

Farbschattierungen zur Verfügung. Bei einem TFT können bis zu 32786 Farbschattierungen genutzt werden.

Für die Ansteuerung lässt sich der interne Peripherietakt oder aber ein an das Takt-Eingangspin angeschlossener externen Takt nutzen. Die Ansteuerung erfolgt über eines der Register. [6]

Für die Ansteuerung stehen folgende Anschlüsse zur Verfügung :

- LCDPWR LCD panel power enable
- LCDDCLK LCD panel Clock
- LCDENA STN AC bias drive or TFT data enable output
- LCDFP STN Frame pulse or TFT vertical sync. pulse
- LCDLE Line end signal
- LCDLPS TN Line pulse or TFT horizontal sync. pulse
- LCDVD[23:0] LCD panel data

2.3 LCD

Die hier genutzten Displays gehören zur Familie der Liquid-Crystal-Displays (LCD). Diese Möglichkeit der Visualisierung hat sich unter den verschiedene Techniken der TFTs herauskristallisiert. Die sog. Flüssigkristallanzeige basiert auf den Eingenschafen von Flüssigkristallen, welche beim Anlegen einer Spannung die Polarisationsrichtung von Licht beeinflussen[5]. Da die Flüssigkristalle lediglich das Licht verändern, nicht aber welches erzeugen können, ist eine ausreichende Beleuchtung nötig. Hierbei wird in *reflektiv*, *transflektiv* und *transmassive* Displays unterschieden. Bei den *transmassiven* Displays bleibt ohne eingeschaltet Hintergrundbeleuchtung das Bild dunkel, während die *reflektiven* Displays das einfallende Tages- oder Kunstlicht als anteilige Reflexion als Lichtquelle nutzen . Die t*ransmassive* Variante vereinte beide Eigenschaften, indem bei unzureichender externen Beleuchtung eine zusätzliche Hintergrundbeleuchtung dazu geschaltet wird. [15]

TFT

Bei den TFT-Monitoren wird ein dünner mit Transistoren beschichteter Siliziumfilm über die gesamte Bildfläche gelegt. Hierbei bekommt jeder Pixel einen Transistor zugewiesen. Diese auch aktive Matrix genannte Vorgehensweise führt zu dazu, dass einzelne Pixel sehr schnell angesprochen werden und können somit Bildänderungen schneller verarbeitet werden. Dies ist der Hauptvorteil anderen Displays mit einer passiven Matrix gegenüber. Denn die Steuerspannung der Pixelpunkte sinkt bei einer passiven Matrix mit der Anzahl der Pixel, bzw. der Anzahl der Zeilen und Spalten. Dies führt dazu, dass passive Matrizen in ihrer Größe begrenzt sind. Eine Erhöhung der Spalten und Zeilenanzahl bei Verringerung der Bildpunktgröße führt zu einer Erhöhung der parasitären Kapazitäten. Wird ein lokaler Verstärker mit separater Stromversorgung eingesetzt, können dennoch schnelle Umschaltzeiten für eine Hohe Bildfrequenz garantiert werden.

Farbdisplays

Für eine farbige Anzeige ist es nötig jedes Pixel in drei Subpixel zu unterteilen (Rot, Grün, Blau). Durch die in der Informationstechnik genannte additive Farbmischung, kann durch die Ansteuerung der einzelnen Subpixel jede beliebige Farbe erzeugt werden. Hierbei ist die Intensität der einzelnen Pixel von Bedeutung. Eine Überlagerung verschiedener Pixel führt zu unterschiedlichen Farben. Während das Einschalten aller Pixel mit der größtmöglichen Intensität zu der Farbe weiß führt, führt ein Ausschalten der drei Subpixel zu einem schwarzen Punkt. Diese Art der Farbkodierung ermöglicht maximal 2^{24} unterschiedliche Farben. Als hexadezimaler Wert umgesetzt können diese von einer Grafikkarte, oder einem LCD Controller an den Bildschirm gesendet werden. [11]

Ansteuerung

Die hier genutzten TFT-Displays benötigten drei verschiedene Takte für den Betrieb. Neben dem invertierten Takt wird eine horizontale Synchronisation (H-Sync) und eine vertikale Synchronisation (V-Sync) benötigt. Wird es über eine externe Beschaltung oder Register

5 Entdeckt 1988 von dem österreichischen Botaniker Friedrich Reinizer.

nicht anders angegeben, werden für das darstellen der Daten mit jedem Takt an den 18 Farbeingängen bei 18 Bit Interfaces, bzw. 24 Farbeingänge bei einem 24 Bit Interface, die Informationen vom LCDC an das Display gesendet.

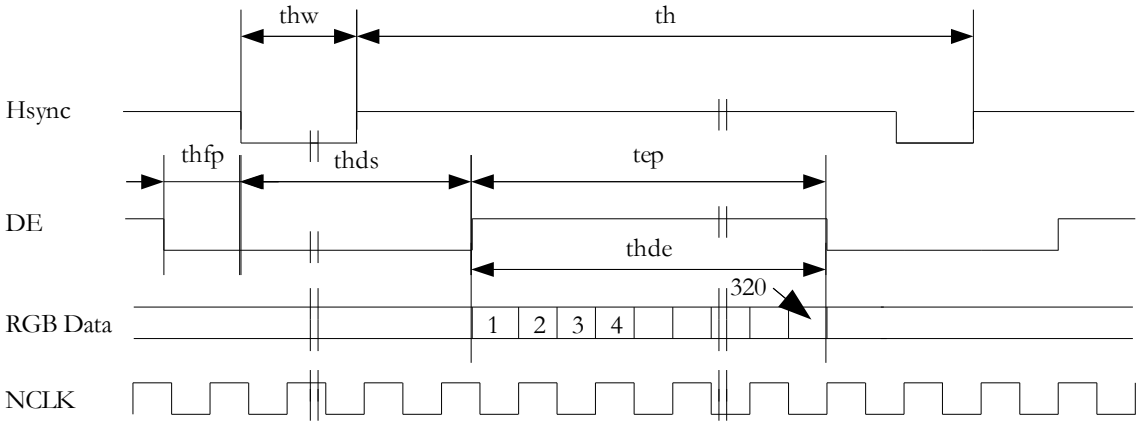


Bild 2-7 H-Sync Timingbeispiel. (Aus dem Datenblatt des Displays LTA057 von Toshiba)

Bild 2-7 zeigt das Timing der einzelnen Pixel mit der horizontalen Synchronisation. Um das Display korrekt anzusteuern ist es nötig die einzelnen Parameter im Datenblatt nach zuschlagen. Es ist zu erkennen das der H-Sync Impuls (thw) auch über mehrere Takt-Signale gehen kann. Bevor die Pixeldaten geschrieben werden ist es nötig für eine gewissen Anzahl an Taktzyklen den H-Sync Impuls zu setzen. Gleichzeitig mit dem Low setzen des H-Sync Signals muss der Data Enable Eingang (DE) für einen gewissen Zeitraum ebenfalls auf *Low* gesetzt werden. Der DE Eingang wird auf High (hier 3,3 Volt) gelegt. Nun wird mit jedem Takt-Signal ein Pixel geschrieben, bis die Zeile voll ist. Am Ende wird der DE Eingang wieder auf Low gelegt und das ganze wiederholt sich. Für den Rücksprung an die Anfangsposition ist ein Kombination von mehreren H-Sync Impulsen mit einem V-Sync Impuls notwendig. In der Regel geschieht das indem währen der vertikalen Synchronisation zwei Pulse der Horizontalen auftreten, wobei eine vom Display abhängige Anzahl von H-Sync Impulsen folgt. Die Größe des Pixel kann beeinflusst werden, indem die Pulse früher gesetzt werden als die Anzahl der Pixel pro Zeile bzw. die Zeilen , des Displays.

2.4 Touchscreen

Bei einem Touchscreen handelt es sich um einen berührungsempfindlichen Bildschirm bei dem die Befehlseingabe für ein Gerät über das direkte Tasten auf Teilen eines Bildes erfolgt. Da der Finger o.ä. dafür verwendet werden kann wird eine Maus zur Steuerung des Zeigers überflüssig. Ein Tippen mit dem Finger oder Zeiger ist analog zum Klicken mit der Maustaste. Ein System das mehrere gleichzeitige Berührungen verarbeiten kann ist als Multi-Touch bekannt.

Es existieren verschieden Technologien zur Umsetzung dieser Funktionalität:

- Kapazitiv, Akustisch, Optisch, Dispersiv, Resistiv.

Kapazitive Touchscreens

Diese Technologie nutzt die kapazitiven Eigenschaften eines mit Metalloxid beschichteten Glassubstrates. Dieses, in geringer stärke unsichtbare, Material ist nun mit Elektroden verbunden die sich an den Rändern des Bildschirms befinden. Wird diese gleichmäßig aufgeladene Schicht nun mit dem Finger berührt, springt ein Teil der Ladung über. Aus dem Entladezyklus kann nun die exakte Position der Berührung ermittelt werden. Gegenüber anderen Technologien ist die Eingabe mittels Stift nicht möglich. [10]

Akustische Touchscreens

Beim Surface-Acoustic-Wave Verfahren (SAW) basiert die Berührungsempfindlichkeit auf dem Prinzip der Oberflächenausbreitung. Hierbei werden Schwingungen in der Glasoberfläche erzeugt die sich mit einer genau definierten Geschwindigkeit ausbreiten. Wird nun das Panel berührt, verändern sich diese Schwingungen, was zu einer exakten Lokalisierung führt. Der Vorteil liegt in der potentialfreien Oberfläche. Die Berührung wird nur über das Glas registriert was zu keiner optischen Veränderung des Bildschirms führt. Auch führen Kratzer zu keinem Ausfall der Funktionalität. [10]

Infrarotlicht

Bei dieser Technologie wird von LEDs erzeugtes Licht auf der gegenüberliegenden Seite des Panels erkannt. Eine Unterbrechung des Lichtstrahls führt zu einer X-Y-Koordinate für die Berührung. [13]

DST

Die Dispersive Signal Technology basiert auf dem Ansatz, dass Berührungen Vibrationen auf dem Trägermaterial erzeugen. Wird das Glas berührt, wird die Vibrationsenergie gemessen und mittels Dispersionsanalyse ausgewertet. Diese Technologie sticht durch seine Fehlsicherheit hervor, ein Ablegen eines Gegenstandes auf dem Gerät bei gleichzeitiger Eingabe ist möglich, da ein ruhender Gegenstand keine Vibrationsenergie erzeugt. [10]

Analog-resistive Touchscreens

Bei dieser Technologie werden zwei Schichten aus einem transparenten halb leitenden Material übereinander gelegt. Die zwei Schichten werden durch Spacer-Dots[6] voneinander getrennt. An die zweite Schicht wird eine Gleichspannung angelegt. Ein Druck auf diese Schichten führt nun zu der Entstehung eines Elektrischen Kontakts. Über den Widerstand dieses Kontakts lassen sich die Koordinaten bestimmen. [10]

4 Wire Resistive Touch

Der in diesem Projekt genutzte resistive 4-Wire Touchscreen (s. Bild 2-8) basiert auf der analogen Technologie und ist die bevorzugte Lösung im *Low Cost* Bereich. Hierbei führen vier Leitungen zum Controller.

Die Vorteile liegen in der hohen Auflösung und der Drucksensitivität, was die Eingaben nicht für Schmutz, Staub, Wasser oder Licht angreifbar macht. Auch ist der Stromverbrauch relativ gering. Der Nachteil liegt in der Empfindlichkeit der Oberfläche in Bezug auf Scharfe Objekte, allerdings gibt es auch Lösungen mit Glasschichten, die weniger empfindlich sind. Alternativ gibt es auch Lösungen als 5- bzw. 8-Wire Touchscreens, die jedoch mit höheren Kosten in Verbindung stehen.

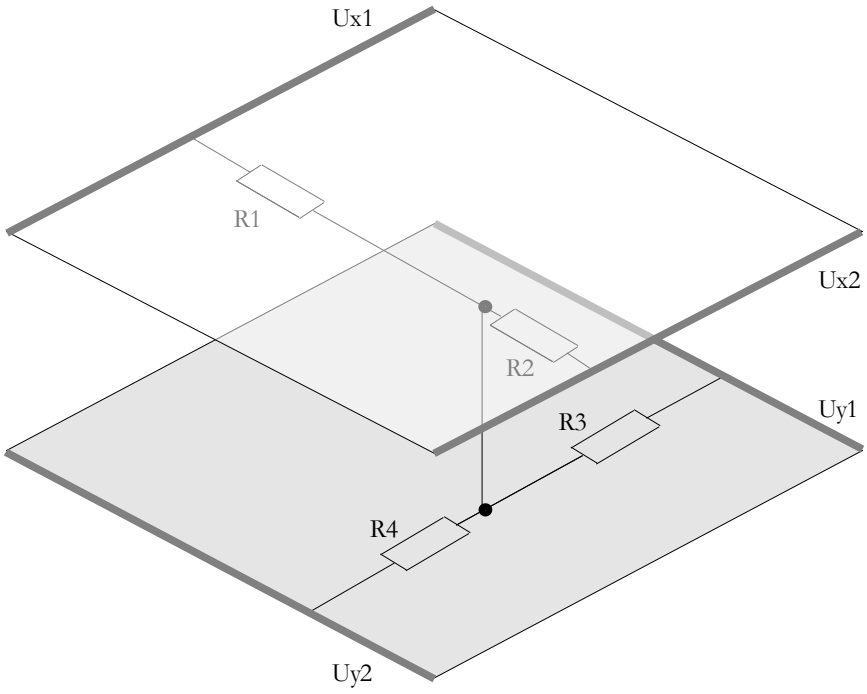


Bild 2-8 4-Wire Resistiver Touchscreen.

6 engl. für *Abstandshalter.*

2.5 LPC2478 OEM Base Board

Das Evaluationboard (s. Bild 2-9) der Firma Embedded Artists[7] hat alle für die geplante Entwicklung der CMA302 benötigten Features.

Zusätzlich zu dem integrierten Touch-Panel ist es auch möglich ein externes LCD TFT anzuschließen. Den Speicher gibt es mit 16 und 32 Bit Datenanbindung. Die notwendigen Programmierschnittstellen JTAG und UART sind vorhanden. Die UART Schnittstelle ist zusätzlich über einen USB zu UART Konverter erreichbar, da moderne Notebooks immer seltener mit einer COM Schnittstelle ausgeliefert werden. Die rote Platine beinhaltet neben dem LPC2478, 256 MBit externen RAM, sowie 32 MBit NOR FLASH und 1 GBit NAND Flash.

Bild 2-9 OEM Base Board.

Es wurden alle Pins des Mikrocontrollers nach außen geführt, was ein großes maß an Flexibilität ermöglicht. Zusätzlich wurden viele Anschlussmöglichkeiten zur direkten Nutzung herausgeführt: SD-Karten-Halterung, optionale Stromversorgung über USB, CAN und Ethernet.

Die ersten Schritte der Entwicklung wurden alle auf diesem Entwicklungsboard getätigt. Eine externe Tastatur sowie ein externes Display ermöglichen eine optimale Simulation des geplanten Gerätes. [4]

7 http://www.embeddedartists.com/

2.6 Entwicklungsumgebung

Die Entwicklungsumgebung µVision3 der Firma Keil bietet weitreichende Möglichkeiten für die ARM Programmierung. Neben der Möglichkeit zu programmieren, kann mit Hilfe des selben Programms *geflasht* und *debuggt* werden. Es ist über eine Vielzahl an Einstellungen möglich den geschrieben Code zu optimieren. In Bild 2-10 ist der auch von anderen Compilern gewohnte Aufbau zu erkennen.

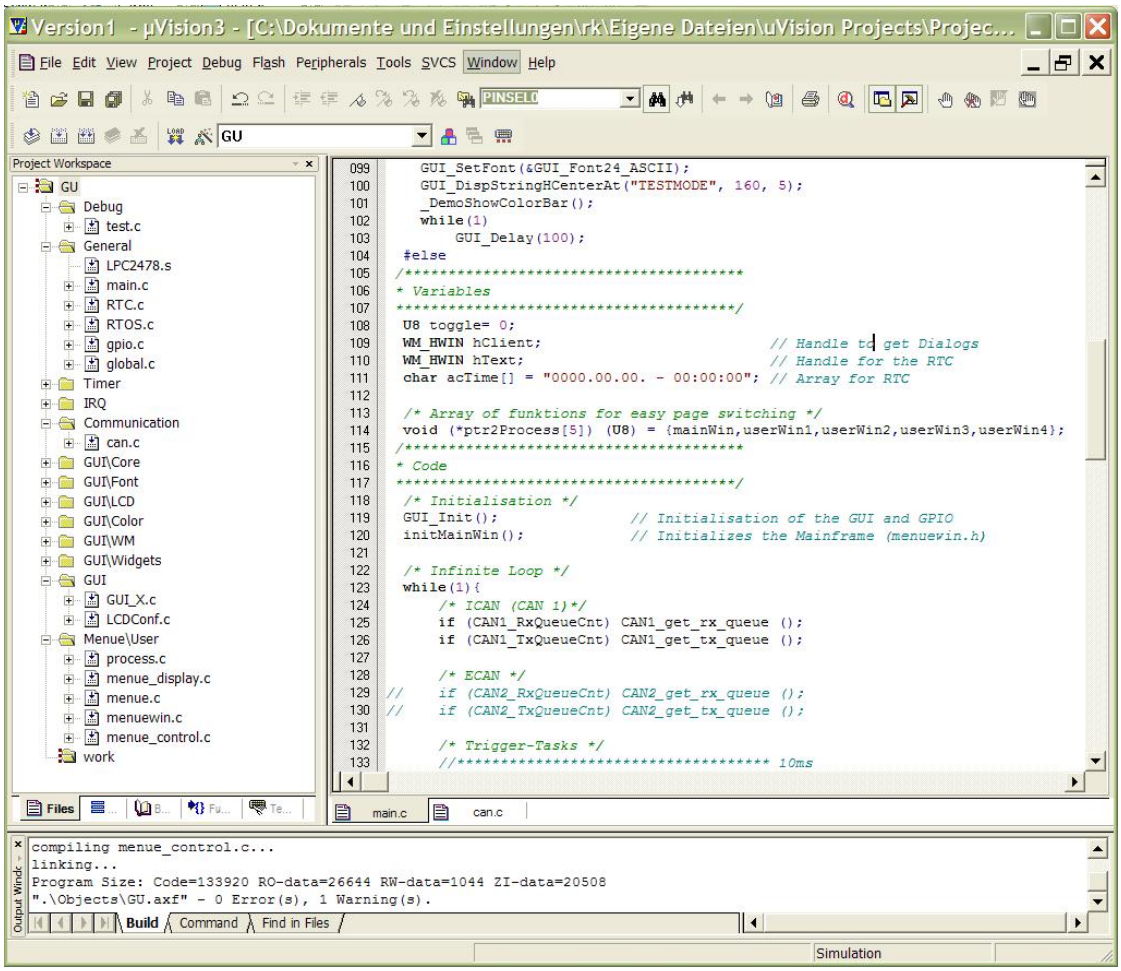


Bild 2-10 Entwicklungsumgebung Keil. - Projektbereich, Ausgabefenster, C- Datei.

Debugging

Mit Hilfe des integrierten Debuggers (s. Bild 2-11) ist die Steuerung des Programmablaufes durch Haltepunkte und zeilenweises Ausführen des Programmcodes, bis zu einem versehentlich eingebauten Fehler, möglich. Die aktuell auszuführende Programmzeile ist dabei mit einem gelben Pfeil markiert. Nach jedem Schritt kann dabei der aktuelle Zustand der einzelnen Register (mitte-links), Speicherbereiche (oben) und der Inhalt von Variablen (mitte-unten) sowie Zeigeradressen begutachtet werden. Damit ist eine schnelle Fehlerbehebung möglich.

Es besteht die Möglichkeit mit dem Debugger einen Mikrocontroller zu simulieren oder aber in Echtzeit mit der Hardware zu arbeiten. Hierbei bietet der Compiler unter dem Menüpunkt Peripherals die Möglichkeit jegliche Peripherie (z.B. SPI, VIC, AD...) zu überwachen oder in die Register zu schreiben, was die Möglichkeiten sehr umfangreich macht. Es können einzelne Flags überprüft werden oder z.B. an den Timereinstellungen gearbeitet werden.

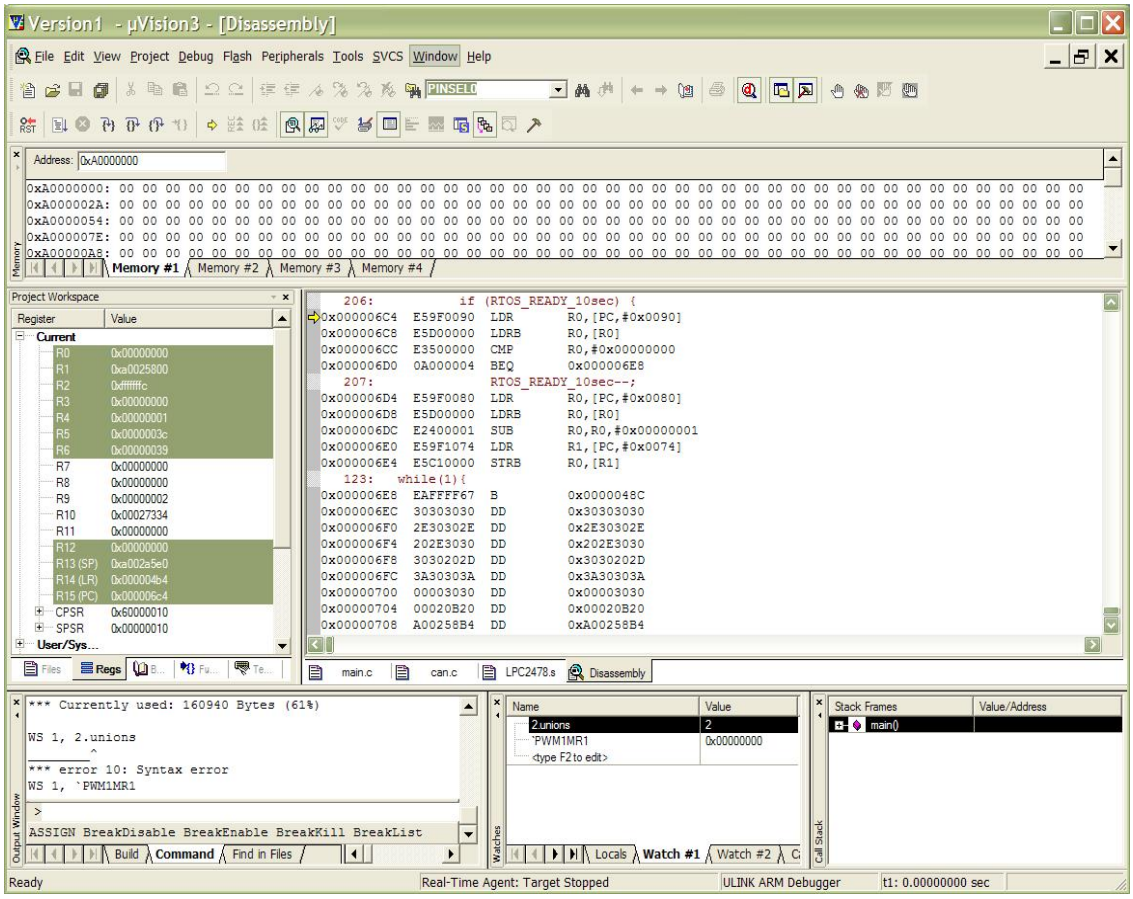


Bild 2-11 Debug Fenster. - Speicherbereich, Register, Assembler- Code, Ausgabefenster, Watch Windows.

Für das Debuggen und Flashen in der Entwicklungsumgebung ist der Programmieradapter ULINK2 von Keil notwendig. Wenn man auf das Debuggen verzichten möchte, gibt es günstigere Alternativen, wie zum Beispiel den R-Link von der Firma ST, welcher nur etwa 1/5 so viel kostet.

MicroLib

Die MicroLib ist eine von Keil optimierte Bibliothek die für ARM basierte in C geschrieben Applikationen angedacht ist. Die Bibliothek ist daraufhin optimiert den Speicherver-

brauch zu reduzieren, was jedoch nicht ohne Einschränkungen geschieht. Sind diese jedoch bekannt ist eine erhebliche effektivere Speichernutzung möglich[8].

Configuration Wizard

Der *Configuration Wizard* ermöglicht eine menügesteuerte Konfiguration einer Initialisierungsdatei. Dabei spielt es keine Rolle, ob die Datei in C oder Assembler geschrieben ist. Der Configuration Wizard nutzt dabei in Kommentare eingebettete Kommandos[9].

Tab. 2.2 zeigt die Kommandos mit denen sich der Wizard konfigurieren lässt. Der Wizard ermöglicht die Initialisierung der Peripherie über ein Menü und erleichtert somit die Übersicht im Programmcode. Variablen oder Register können vor dem Kompilieren, ohne tiefere Kenntnisse von dem Programmaufbau, verändert werden.

Tab. 2.2 Configuration Wizard. - Kommandos.

Kommando	Text	Beschreibung
<h>	ja	Überschrift für eine Optionsgruppe
<e>³	ja	Überschrift für eine Optionsgruppe mit Check-boxen,
<e.4>³	ja	Überschrift um ein bestimmtes Bit zu modifizieren (hier Bit 4)
</h> oder </e>	ja	Ende der Beschriftung
<i>	ja	Hilfetext für das vorige Element
<q>³	ja	Optionen für Bitwerte die in einer Check- Box dargestellt werden
<o>³	ja	Optionen mit einem wählbaren Nummerneintrag
<o.4..5>³	ja	Modifizieren von Bits (hier: 4 und 5)
<0.4>³	ja	Modifizieren von einem Bit
<s>³	ja	Option mit String Eintrag
<s.10>³	ja	Option mit String Eintrag, und Größen Limitierung (hier: 10 Zeichen)
Variable		
<0 - 31>	nein	Wertebereich für Optionsfeld
<0 - 100: 10>	nein	Wertebereich für Optionsfeld in 10er Schritten
<0x40 - 0x1000: 0x10>	nein	Wertebereich im hex Format
<0=>	ja	Wert und Text zu Auswahl
<#+1> <#-1> <#*8> <#/3>	nein	Modifizierung des Wertes

8 http://www.keil.com/arm/microlib.asp (09.06.2009).
9 http://www.keil.com/support/man/docs/uv3/uv3_ut_configwizard.htm (09.06.2009).

2.7 SEGGER

Die Firma SEGGER verbreitet und entwickelt Softwarekomponenten für die Industrie. Diese in ANSI C geschriebene Bibliotheken ermöglichen es einem Unternehmen Zeit bei der Entwicklung zu sparen und sich mit dem wesentlichen Kern der Programmierung zu befassen. Die Geschichte der Firma kann im Anhang A1: Geschichte Segger nachgelesen werden.

2.7.1 emWin

Bei emWin handelt es sich um eine in ANSI C geschriebenen Software Bibliothek. Der Code ist einsehbar und bietet somit die nötige Flexibilität das Programm an die Gegebenheiten anzupassen.

Die Firma SEGGER gibt an, dass ihr emWin auf jedem System mit jedem Controller und jedem Display funktionstüchtig ist[10].

Das Paket beinhaltet neben grundlegenden Funktionen wie dem Zeichnen von Linien und Polygonen, optimierte Darstellung von Werten und Schriftzeichen. Es werden diverse Eingabemöglichkeiten unterstützt.

Eine Erweiterung sind die sogenannten *Widgets*. Hierbei handelt es sich um eine Reihe von vorgefertigten Modulen wie Dialogboxen, Tasten oder Ladebalken, welche einem ein typisches vom PC gewohntes Aussehen vermitteln. Ein Vorteil der *Widgets* ist, dass sie sich selber neu zeichnen wenn dies erforderlich ist oder automatisch gelöscht werden, wenn das Fenster an dem sie hängen gelöscht wird. Dies führt zu einem geringeren Verwaltungsaufwand. Diese Eigenschaften werden im Laufe dieser Arbeit erläutert.

Wird über die Arbeit mit den grundlegenden Elementen hinausgegangen, ist es nötig objektorientiert zu denken. Für jedes darzustellende Modul, sei es ein Fenster, oder ein ganzer Dialog werden sogenannte *Handle* erstellt. Dem Handle werde nun Attributen, Verhaltensweisen oder weitere Module zugewiesen, welche in einer *Vater-Kind-Hierarchie* zu verstehen sind. Ein *Vater* kann mehrere *Kinder* besitzen, welche wiederum *Kinder* besitzen können.

Nun besteht die Möglichkeit die von SEGGER genannten *Callbackroutinen* zu verwenden. Die Philosophie dahinter ist ein Ereignis gesteuertes System. Der Kontrollmechanismus erfolgt von der Anwendung zum System und wieder zurück zur Anwendung. Hierbei ist hauptsächlich das Neuzeichen von Fenstern zu verstehen. Ohne nutzen dieser Routinen ist es sehr viel aufwendiger sich um das Aktualisieren der Fenster zu kümmern, wenn z.B. ein Bildschirmteil zeitweilig von einem Fenster überdeckt wird.

Für die Kommunikation der Module untereinander, bzw. mit den *Callbackroutinen*, existiert eine Nachrichtenstruktur. Diese beinhaltet gedrückte Tasten, Flags, oder Informationen über getätigte Eingaben. [12]

10 http://www.segger.com/cms/emwin.html

2.7.2 Widgets

Die folgenden Namen, Begriffe und Abbildungen wurden von der Firma SEGGER übernommen. Die Namen und Begriffe werden ab Kapitel 4, Software, Seite 42 Anwendung finden und von nun an als bekannt betrachtet werden. Die hier vorgestellten Widgets sind nur die bis zum jetzigen Zeitpunkt im Projekt verwendeten. Dieses Kapitel bezieht sich auf Anhang B3: Manual SEGGER: *Kapitel 15*.

Ein EDIT Widget (Bild 2-12) existiert in erster Linie für die Eingabe von Texten. Es ist aber auch möglich darin vorgegebene Zahlenwerte abzufragen, es besteht aus einem einfachen Eingabefeld. Das FRAMEWIN Widget gibt dem Benutzer ein vom PC gewohntes Erscheinungsbild, dabei sind die Farben wie bei allen *Widgets* variable. Ein Ändern der Farbe signalisiert dem Benutzer in welchem Fenster sich der Fokus befindet. Bei dem FRAMEWIN handelt es sich um zwei Fenster, dem Farbigen Balken mit einer Überschrift, genannt Frame und dem darunter liegenden Clientfenster. Eine LISTBOX gibt einem die Möglichkeit eines oder mehrere Elemente aus einer vorgegebenen Liste zu selektieren.

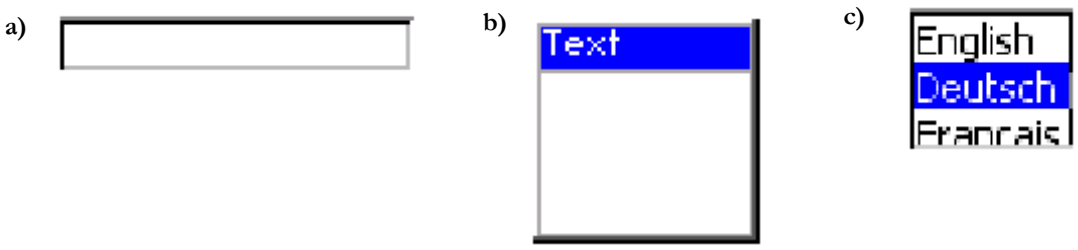


Bild 2-12 Widgets: a) Edit, b) Framewin c) Listbox.

Das LISTVIEW Widget (Bild 2-13) ermöglicht das Selektieren aus einer Liste mit verschiedenen Spalten. Zur Verwaltung der Spalten enthält das LISTVIEW Widget ein HEADER Widget über welches sich die Elemente auch sortieren lassen. Das MENU Widget bietet einem verschiedene Möglichkeiten ein Menü zu erstellen. Dabei kann das Menü horizontal, oder vertikal erscheinen und eine beliebige Anzahl von Untermenüs haben. Beim Löschen muss darauf geachtet werden, dass alle Untermenüs einzeln gelöscht werden, da beim anhängen an einen anderen Menüpunkt keine *Vater-Kind-Hierarchie* erstellt wird.

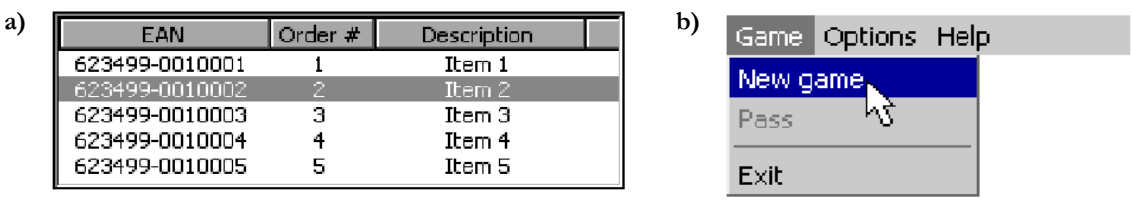


Bild 2-13 Widgets: a) Listview, b) Menu.

Über die MULTIPAGE (Bild 2-14) können mehrere, über Reiter anwählbare, Seiten in dem selben Bereich angezeigt werden. Dabei wählt ein Reiter ein Fenster an, welches nun einen beliebigen Inhalt darstellen kann. Die PROGBAR dient lediglich der Veranschaulichung von Werten. Das TEXT Widget ermöglicht es einem Text einen festen Bereich zuzuweisen. Dabei ist es möglich dieses Widget eindeutig zu identifizieren und somit den Text dynamisch zu verändern. Hierbei stehen eine Reihe von Formatierungsmöglichkeiten zur Verfügung, wie links oder rechts ausgerichteter Text oder diverse Umbrüche.

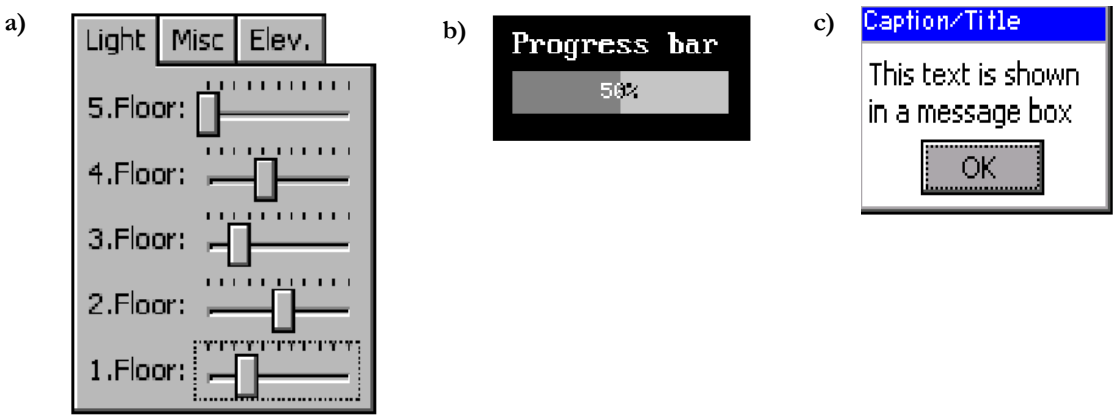


Bild 2-14 Widgets: a) Multipage, b) Progbar, c) Framwin.

Jedes dieser Elemente kann über eine große Auswahl an Funktionen angepasst und erstellt werden. Kapitel 15 der Bedienungsanleitung für emWin beschäftigt sich ausschließlich mit den Widgets und beschreibt alle zur Verfügung stehenden Funktionen. [8]

2.7.3 Callbacks

Callbackroutinen werden vom Benutzer definiert, um dem System mitzuteilen, dass eine bestimmte Funktion bei einem bestimmten Ereignis stattfinden soll. Eine solche Funktion kann an jedes beliebige Fenster angehängt werden. Und hat den folgenden Prototyp:

```
void callback(WM_MESSAGE* pMsg)
```

mit : `pMsg` – Pointer auf eine Datenstruktur vom Typ `WM_MESSAGE`.

Die Aktion die eine Callbackroutine ausführen soll hängt nun von dem Typ der Nachricht ab, die sie als Übergabeparameter bekommt. Eine Differenzierung findet hierbei typischerweise über eine switch-case Anweisung statt.

Tab. 2.3 beinhaltet die für dieses Projekt wichtigen Nachrichten. Des Weiteren gibt es noch: WM_MOVE, WM_NOTIFY_VIS_CHANGED, WM_SIZE, WM_TIMER, WM_MOUSEOVER, WM_MOUSEOVER_END. [8]

Tab. 2.3 Nachrichten der Callbackroutinen.

Message Id (MsgId)	Erklärung
System Nachrichten	
WM_CREATE	Diese Nachricht wird direkt nach dem Erstellen eines Fenster gesendet.
WM_DELETE	Wird dem Fenster kurz vor dem löschen gesendet um ggf. Speicher freizugeben.
WM_GET_ID	Wird zu einem Fenster gesendet wenn die ID abgefragt wird.
WM_INIT_DIALOG	Wird gesendet nachdem ein Dialog initialisiert wurde.
WM_KEY[a)]	Wird zu dem im Fokus stehenden Fenster gesendet wenn ein Taste gedrückt wird.
WM_NOTIFY_PARENT[b)]	Benachrichtigt den Vater, dass etwas in einem der Kinder passiert ist.
WM_PAINT[c)]	Wird zu einem Fenster gesendet falls es überdeckt wurde und neu gezeichnet werden muss.
WM_SET_FOCUS[d)]	Wird zu einem Fenster gesendet wenn es den Fokus verliert.
WM_SET_ID[e)]	Wird zu einem Fenster gesendet wenn die ID sich ändert.
Eingabegeräte	
WM_TOUCH[f)]	Wird gesendet wenn eine Berührung stattfindet.
WM_TOUCH_CHILD[g)]	Wird gesendet wenn ein Kindfenster berührt wurde.
Benachrichtigungs Code	
WM_NOTIFICATION_CHILD_DELETED[h)]	Wird gesendet wenn ein Kind gelöscht wurde
Benutzer definierte Nachricht	
WM_USER	Diese Nachricht ist frei vom Benutzer definierbar.

a) `data.p` zeigt auf eine Struktur vom Typ WM_KEY_INFO mit: int Key (gedrückte Taste), int pressedcount (Zustand).
b) `data.v` beinhaltet eine Nachricht die abhängig von dem Sendenden Widget ist.
c) `data.p` beinhaltet die Koordinaten eines Rechtecks welches neu gezeichnet werden muss.
d) `data.v` ist 1 wenn der Fokus akzeptiert wird.
e) `data.v` beinhaltet die neue ID.
f) `data.p` zeigt auf eine GUI_PID_STATE Struktur.mit int x, int y, U8 pressed (Zustand).
g) `data.p` beinhaltet die selben Informationen wie bei f) nur das die Koordinaten vom Kind kommen.
h) `hWinSrc` Handle des gelöschten Fensters.

3 Hardwareplattform

Die gesamte Hardware des SYMAP®-Compact soll aus drei Platinen bestehen, wobei evtl. eine vierte für weitere Kommunikationsschnittellen angedacht ist. Die Tabelle 3.1 zeigt die betriebsinternen Bezeichnungen der Karten.

Tab. 3.1 Platinen des Symap-Compact.

Bezeichnung der Karte	Beschreibung
CMA 300	Mess- und Kontrolleinheit
CMA 301	Umsetzerkarte
CMA 302	Grafikkarte
CMA 303	Kommunikation

Kerne dieser Platinen sind die drei Controller die sich auf der CMA 302 (GU), und CMA 300 (MU, CU) befinden. Es handelt sich hier drei Mal um den LPC2478 der Firma NXP, der auf dem ARM7 basiert. Bild 3-1 zeigt hierbei die internen und externe Kommunikationswege der einzelnen Komponenten.

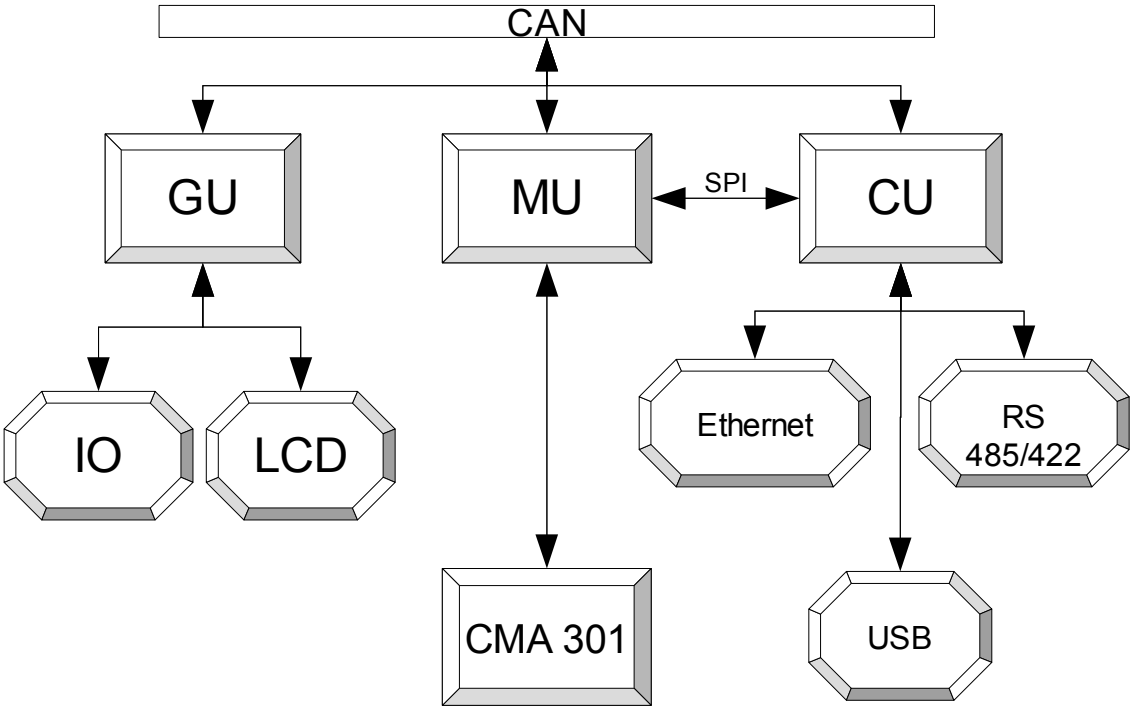


Bild 3-1 Kommunikationsverbindungen der Komponenten im Symap Compact.

3.1 CMA 302

Dieses Projekt befasst sich mit der Entwicklung der GU (Graphic Unit). Hierbei soll es sich um ein flexibles Interface handeln, welches lediglich über eine CAN Verbindung mit dem Rest des Systems kommuniziert. Dies soll ermöglichen ein Display auch außerhalb des Gerätes anzubringen. In Bild 3-2 sind die einzelnen Komponenten mit dazugehöriger Schnittstelle in einem Blockschaltbild zusammengefasst. Der 4-Wire resitive Touchscreen ist über die SSP0 angeschlossen.

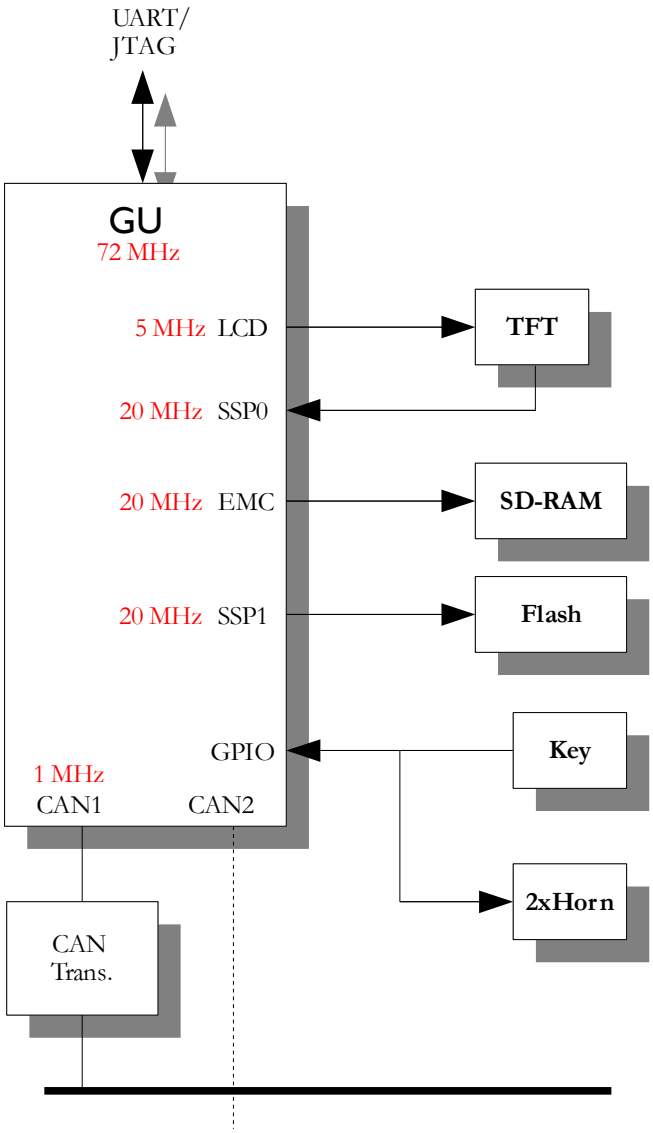


Bild 3-2 Blockschaltbild CMA302.

Ausgehend von dem SYMAP®-BCG fallen jedoch Elemente wie LEDs und sieben Segment-Anzeigen weg. Als I/O bleiben nun eine Tastatur sowie zwei Lautsprecher für Alarme übrig.

Nun besteht die Möglichkeit einen externen LCD-Controller, einen im LCD verbauten, oder einen im Mikrocontroller integrierten LCD-Controller zu verwenden. Letzterer ver-

eint die Vorteile von den beiden Ersten. Ein integrierter LCD-Controller ist platzsparender und hat keine Bindung an ein bestimmtes Display.

Da der eingebaute LCD-Controller keinen eigenen Bildspeicher hat, ist es nötig einen externen RAM anzubinden. Ein Vollbild mit 16 Bit pro Pixel ergibt bei 76800 Pixel einen Speicherbedarf von 153600 Byte.

Für das Abspeichern von ggf. vorhanden Logos ist ein Flash vorgesehen.

Als Display wird ein QVGA 5,7 Zoll RGB Farbdisplay mit 18 Bit paralleler Anbindung verwendet.

Beschaltung

Der LPC2478 wird mit seiner maximalen Taktrate von 72 MHz betrieben. Programmiert wird über JTAG wobei die nötigen Pins hierbei direkt an den Prozessor angeschlossen werden.

Um die Datenleitungen zu reduzieren wird ein 256 MBit (32 MByte) SDRAM über 16 Datenleitungen an den EMC angeschlossen. Hierbei werden die Pins direkt mit den zugehörigen Pins des Controllers verdrahtet. Verwendet wird hierbei der **MT48LC64M4A2** der Firma Micron[11]**.** Der EMC verwaltet den Speicher so, dass für 4 Byte Zugriffe zwei Mal auf den Speicher gelesen oder geschrieben wird, was somit trotz 16 Bit Speicheranbindung keine Einschränkung in der Programmierung zur folge hat. Es muss jedoch bedacht werden, dass ein 32 Bit Zugriff zwei Mal so lange dauert, wie bei einer 32 Bit Anbindung. Der EMC selber wird mit 20 MHz betrieben.

Für den Flash-Speicher wird ein 1 MB serieller Flash verwendet (**M45PE80,** s. Bild 3-3), welcher lediglich mit vier Datenleitungen auskommt und mit 20 MHz über die SPI angeschlossen wird. Das SPI Interface ermöglicht einen Zugriff von 256 Byte auf ein Mal. Da dieser Speicher nur für das einmalige Laden eines Logos o.ä. genutzt werden soll, können damit Datenleitungen im Vergleich zu einem parallelen Flash eingespart werden.

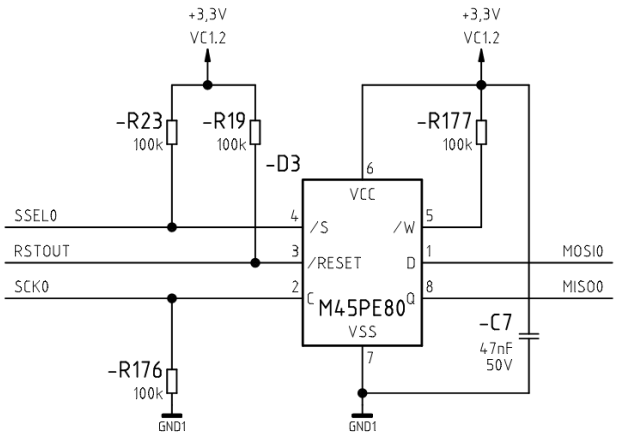


Bild 3-3 M45PE80. - Serieller Flash.

11 http://www.micron.com/

Bei der Tastatur handelt es sich um einfache Taster, welche mit Pullup Widerständen über zwei Treiberbausteine mit den interruptfähigen GPIO Eingängen des Controllers verbunden werden. Somit entfällt ein Polling der Eingänge, denn bei jedem Tastendruck wird ein Interrupt ausgelöst.

Bei den 16 Bit Treibern, welche zusätzlich auch zwischen LCDC und LCD geschaltet werden, handelt es sich um den **SN74LVC16245A** von Texas Instruments[12]. Die verwendete Beschaltung führt zu einem stabilisierten Signal von Bus A zu Bus B.

Der erste Lautsprecher ist eine Hupe, welcher sich über den Transistor (**BSP295)** ein- und ausschalten lässt. Der zweite Lautsprecher hängt an dem Verstärker **LM386M**, welcher an einem DAC Ausgang des Controllers angeschlossen ist (s. Bild 3-4 a)). Der Spannungsteiler am Eingang ermöglicht eine maximale Aussteuerung des Verstärkers über den DAC des Controllers. Pin 1 und 8 werden offen gelassen, was eine Verstärkung von 20 zur Folge hat (s. Bild 3-4 b)).

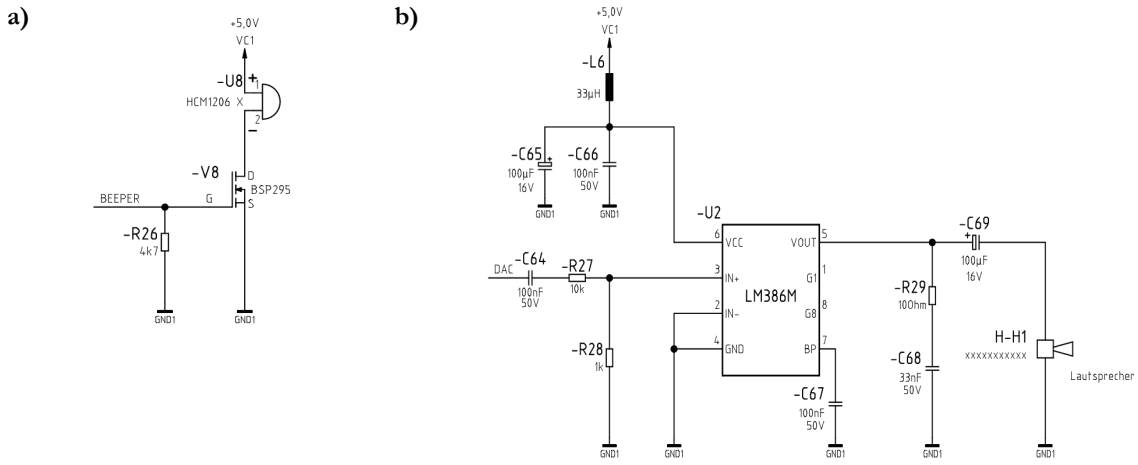


Bild 3-4 a) Hupe, b) Lautsprecher.

Der CAN Transceiver **SN65HVD251D** wird direkt an die TxD, bzw. RxD Anschlüsse des Controllers angebunden. Aufgrund der geringen Entfernungen zu den anderen Platinen wird hier eine Taktung von 1 MHz oder mehr möglich sein.

Mit dem **LM60** ist ein Temperatursensor an einen der analogen Eingänge geführt.

Die vier Ausgänge des resistive Touchscreen, werden in dem **TSC2046** (s. Bild 3-5) in Koordinaten umgewandelt und über eine weitere SPI-Schnittstelle und einem externen Interrupt mit dem Controller verbunden. Bei diesem Touchscreen Controller handelt es sich um einen ADC, welcher die über die Widerstandsmatrix gewonnene analoge Spannung in ein digitales Signal umwandelt. Der Baustein muss aufgrund eines Interruptausgangs nicht gepollt werden.

Für die Spannungserzeugung der LED Hintergrundbeleuchtung wird der LED-Treiberbaustein **TPS61165** eingesetzt, welcher über eine PWM gesteuert für die Einstellung der nötigen Hintergrundbeleuchtung sorgt. Die Beschaltung ergibt sich aus dem Datenblatt. Die zusätzlichen Kondensatoren und Spulen dienen dem Schutz des Gerätes und der Erhöhung der Störungsunempfindlichkeit (s. Bild 3-6).

12 http://www.ti.com/

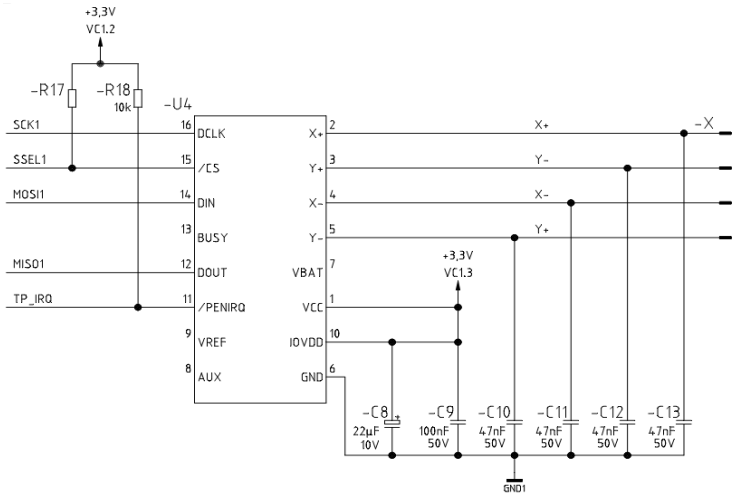


Bild 3-5 TSC2046.

Um das Testen der Bildschirme zu vereinfachen und einen leichten Umstieg zu ermöglichen, werden auf der ersten Version der CMA302 drei Anschlusstypen verbaut. Hierbei handelt es sich um einen 33 poligen Anschluss, sowie um zwei 40 polige Anschlüsse. Dabei werden die Leitungen parallel an alle Anschlusstypen geführt. Hierbei muss beachtet werden, dass der 33 polige Anschlusstyp zusätzlich zwei Pins für die vertikale und die horizontale Ausrichtung besitzt. Die Displays von Hitachi benötigen keine Synchronisation.

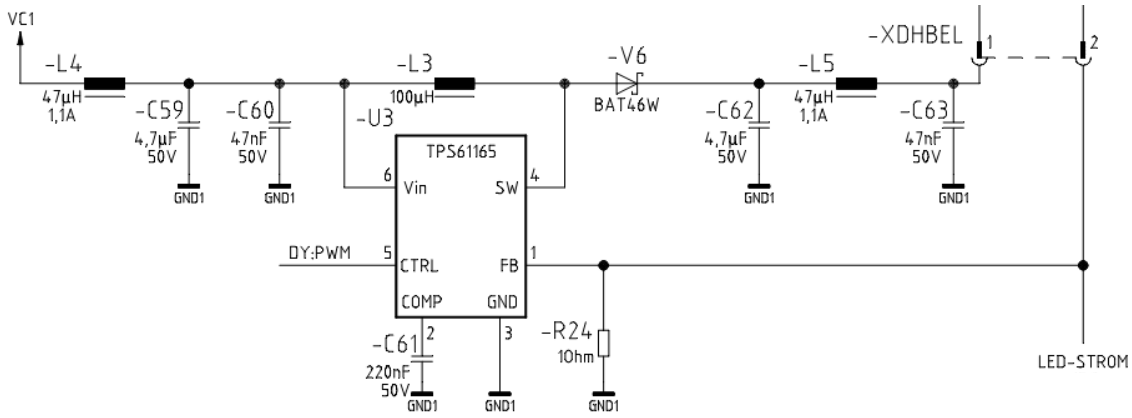


Bild 3-6 TPS61165.

Datenblätter zu den einzelnen Komponenten befinden sich in Anhand C: Datenblätter. Der komplette Schaltplan für die CMA302 befindet sich in Anhang B7: Stromlaufplan CMA302.

Das Display wird über 16 Bit RGB 565 mit dem LCDC verbunden. Da es sich in den meisten fällen um 18 Bit Bildschirme handelt werden die LSBs von Rot und Blau auf Masse gelegt, was keine Einschränkungen in der Ansteuerung zur Folge hat.

3.2 LCD Display

Kriterien

Bei dem großen Angebot an zur Verfügung stehenden Displays ist es nötig eine differenzierte Auswahl zu treffen. Hierbei sollte in erster Linie darauf geachtet werden, dass ein Displaytyp gewählt wird, der eine große Anzahl an identischen Anschlussmöglichkeiten besitzt. Dies ermöglicht es im Notfall auf einen anderen Displaytyp zu wechseln.

Außerdem ist die Helligkeit und der Kontrast sehr wichtig, wobei nach Aussage der Distributoren eine Helligkeit von 250 cd/m² genug sein sollte, da das Display ausschließlich in geschlossenen Räumen Verwendung findet.

Es gibt zwei Arten von Hintergrundbeleuchtungen, die für das Gerät in Frage kommen können:

1. LED

2. CCFL

Die Variante mit LEDs als Hintergrundbeleuchtung hat den Vorteil, dass die Helligkeit bis zur vollständigen Dunkelheit dimmbar ist. Allerdings altern die Bauelemente bei hohen und sehr tiefen Temperaturen schnell. Das heißt, die angegebene Zeit von beispielsweise 50.000 Stunden kann nicht mehr garantiert werden, da sich diese Zeitangabe meist auf eine Temperatur von ca. 25°C beläuft. Somit wäre im Schiffsbereich sicherlich eine CCFL-Hintergrundbeleuchtung angebrachter. Der Nachteil an dieser Beleuchtung ist die Häufigkeit der Ein- und Ausschaltvorgänge, welche die Lebensdauer der Röhre reduziert, sowie die Helligkeit, die nur bedingt dimmbar ist (etwa auf max. 20-30% der Helligkeit). Die Schaltspiele der Lampe können umgangen werden, indem man die Helligkeit auf das Minimum reduziert. Dadurch wird die Lebensdauer erhöht. Zusätzlich ist auch noch gewährleistet, dass auch im Standbybetrieb des Gerätes weiterhin etwas zu sehen ist, wenn kein transflektives Display verbaut wird. Der Leistungsverbrauch einer CCFL-Beleuchtung liegt mit 4 Watt etwa 2-3 mal so hoch wie der einer LED-Beleuchtung, was sich in der Wärmeentwicklung niederschlägt.

Ein OLED wäre eine alternativ gewesen, allerdings war es nicht möglich dieses in ausreichender Stückzahl sowie in der geforderten Größe zu bekommen.

Aufgrund dieser Kriterien wurde sich für die LED Variante entschieden.

Tab. 3.2 zeigt eine Auswahl der für das Projekt nicht in Frage kommenden Displays. Aussortiert wurden alle Displays ohne LED Hintergrundbeleuchtung, sowie solche mit sehr speziellen Anschlüssen.

Tab. 3.2 LCDs die den Anforderungen nicht entsprechen[a]. - Die mit * markierten Attribute sind für die Nichtauswahl des LCDs verantwortlich.

Hersteller	Hintergrund	Hintergrundbeleuchtung	Temp. (°C)	Anschluss (Pin)
Ampire	LED	126 mA / 10,5 V	-20 bis +70	24 / 34 *
Ampire	LED	140 mA / 42 V*	-20 bis +70	40
Ampire	CCFL*	6 mA / 650 V / 55 kHz	-20 bis +70	40
Ampire	LED		-20 bis +70	24 / 34 *
Ampire	LED	330 mA / 10,5 V	-20 bis +70	54 *
CTC	CCFL*	8 mA / 710 V / 35 kHz	-20 bis +70	33
CTC	LED	60 mA / 12,4 V	-20 bis +70	54 *
DEM	LED	60 mA / 13 V	-20 bis +70	60 *
Gi Far	LED		-20 bis +70	50 *
Hantronix	CCFL*		-20 bis +70	33
Hitachi	CCFL*		-20 bis +70	40
Hitachi	CCFL*		-20 bis +70	40
Hitachi	LED		-20 bis +70	40
Kyocera	CCFL*		-10 bis +70	33
Kyocera	LED		-10 bis +70	33
Kyocera	CCFL*		0 bis +60	
Newtec	LED		-20 bis +70	50 *
OSD	LED	140 mA / 10,2 V	-20 bis +70	50 *
Sharp	CCFL*		-30 bis +80	33
Sharp	CCFL*		-30 bis +80	33
Suntai	CCFL*		-10 bis +70	33
Tianma	LED	60 mA / 12,4 V	-20 bis +70	54 *

a) Die Komplette Tabelle mit Typnummer, Distributor, Preis sowie aller zur Verfügung stehenden Displays befindet sich in Anhang A4: Tabelle LCDs.

In der Tabelle 3.3 sind alle Displays aufgeführt von denen Muster zum testen vorhanden sind, dabei handelt es sich um 30 und 40 Polige Displays. Es sind ausschließlich Displays mit 3,3 V Ansteuerung für die digitale Logik und LED Hintergrundbeleuchtung vorhanden. Bis auf das Powertip PH320240T-005-I-Q welches ein 24 Bit Display ist, sind alle anderen Displays 18 Bit.

Tab. 3.3 Auswahl LCD 2.

Hersteller	Typ-Nr.
Ampire	AM320240NSTNQW-00H
Andi	YL#T320x240-17-057L2H
AUO	G057QN01
Data Image	FG050701DSSWBG01
EDT	ET057003DM6
Everbouquet	MF320240B57-BF
Gi Far	GFT057AA320240
Hantronix	HDA570S-V
Hitachi	TX14D12VM1CPC
Kyocera	TCG057QVLBA-G00
Nanya	LM740-0A
Optrex	T-55265GD057J-LW-AAN
Powertip	PH320240T-005-IC1Q
Powertip	PH320240T-005-I-Q
PrimeView	PD057VU5
Toshiba	LTA057A340F
URT	UMSH-8044MD-T

Neben Anschlussmöglichkeit und Preis wurde auf Helligkeit, Kontrast und Bildschärfe getestet. Die Einbaumöglichkeiten wurden mit den Gehäusespezifikationen des SYSMAP-Compact verglichen. Dabei fällt auf, dass die verschiedenen Displays untereinander nicht kompatibel sind (s. Anhang B6: Abmessungen LCDs (CD)).

Der erste Entwurf der Grafikkarte beinhaltete Anschluss- und Befestigungsmöglichkeiten für die 33 und 40 Poligen Displays. Hierfür wurde eine Zwischenplatine entworfen, auf der die Möglichkeit besteht die Bohrung für das verwendetet Display anzupassen. Eine Analyse der Attribute und Preise hat jedoch ergeben, dass auf die 40 Poligen Anschlüsse verzichtet werden kann, da keine signifikanten Unterschiede der Displays bestehen. Die Zwischenplatine wird jedoch beibehalten um Flexibilität zu gewährleisten.

LCD „Hantronix“

Die Auflösung und die Pixelabstände sind bei allen Modellen nahezu identisch, was zu einer gleichmäßigen Schärfe bei allen Bildschirmen führt. Der Eindruck eines Displays unterscheidet sich, wenn ein mattes oder ein glänzendes Panel verwendet wird..

Während ein glänzendes Display lediglich Geschmackssache ist, hat das Testen der Panels signifikante Unterschiede der einzelnen Displays ergeben. Bei den günstigen Varianten müssen Abstriche in verschiedenen Bereichen vorgenommen werden.

Das Display von URT welches zu den günstigsten Vertretern gehört hat zwar einen akzeptablen Kontrast, jedoch ist es nicht allzu hell, in Kombination mit einem Touch, welches das Display noch dunkler erscheinen lässt ist es nur in sehr abgedunkelten Räumen einwandfrei nutzbar. Im Gegensatz dazu steht das Display von Powertip. Dieses Display erreicht eine sehr hohe Helligkeit, ist aber auch das teuerste Modell. Die Helligkeit ergibt sich allerdings auf Kosten der Lebensdauer und des Stromverbrauches, was zusammen mit dem Preis ein Ausschluss ergibt. Das Display von Optrex hat sich als sehr gut erwiesen. Es hat einen guten Kontrast in allen Farbbereichen sowie eine hohe Helligkeit bei langer Lebensdauer.

Letztendlich wurde sich für das Hantronix entschieden, welches die selben Eigenschaften wie das Nanya hat, jedoch ein wenig günstiger ist und sich aus diesem Grund gegenüber allen anderen Displays durchsetzen konnte. Die Helligkeit ist für diese Anwendung ausreichend bei einem relativ hohen Kontrast und einem optisch guten Eindruck.

Tab. 3.4 zeigt die aus dem Datenblatt bekannten Eingenschaften des Displays.

Tab. 3.4 Spezifikationen. (Datenblatt HDA570S-V)

Nr.	Begriff	Beschreibung	Einheit
1	Größe	5.7	Zoll
2	Auflösung	320 (H) x RGB x 240 (V)	Pixel
3	Pixelabstand	0.120 (H) x 0.360 (V)	mm
4	Aktiver Bereich	115.20 (H) x 86.40 (V)	mm
5	Dimensionen	144.00 (W) x 104.60 (H) x 13.00 (T)	mm
6	Display Typ	Weiß/ Transmissiv	
7	Pixel Struktur	RGB- Streifen	
8	Eingags Interface	18 Bit parallel RGB	
9	Oberfläche	Anti- Glanz (AG)	
10	Gewicht	210	g
11	Temperaturbereich	-20 (Ta)~70(Ta)	°C
12	Blickwinkel	55/50/60/60	Grad
13	Helligkeit	380	cd/m^2
14	Lichtquelle	LED	
15	NTSC Rate	58	%

3.3 Versuchsaufbau

Da die CMA302 erst noch entwickelt werden musste war ein Testaufbau nötig um parallel zur Hardwareentwicklung die nötige Software schreiben zu können.

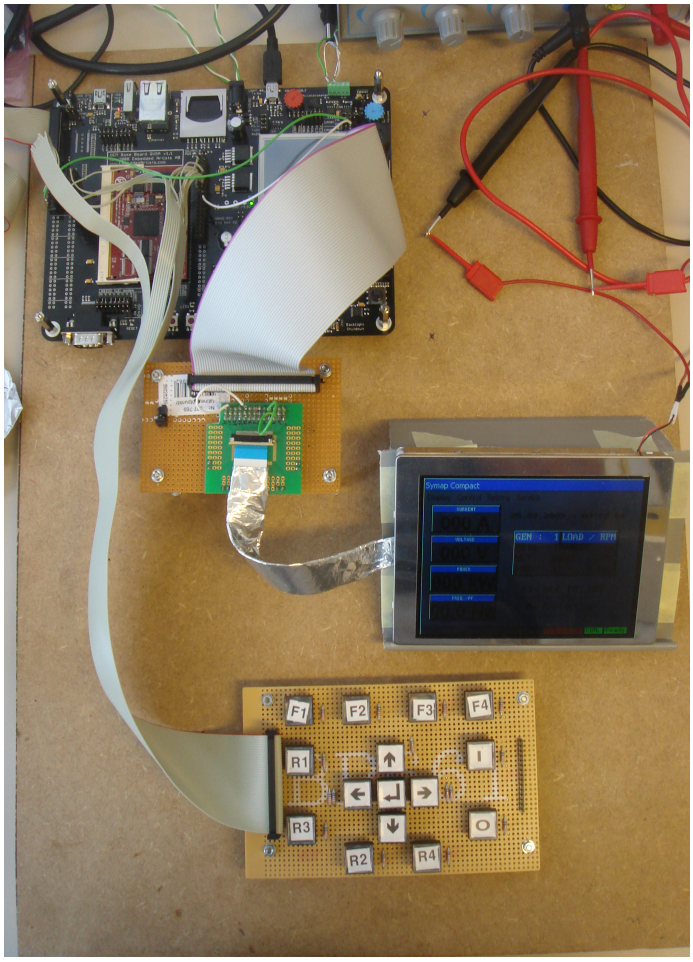


Bild 3-7 Testaufbau Gesamtbild. - Evaluationboard, Adapter, LCD, Tastatur.

Bild 3-7 zeigt den Gesamtaufbau fixiert auf einer Holzplatte. Die Spannungsversorgung der LCD-Hintergrundbeleuchtung geschieht über eine Spannungsquelle der Firma Conrad mit einem Ausgang von 0–30 Volt und 0,01–3 Ampere. Bis auf das Display wurden alle Komponenten an der Platte fixiert. Das Display steht mit einem Winkel von 35° in einer Schräge um einen Displaywechsel für spätere Tests komfortabel zu halten und einen günstigen Betrachtungswinkel zu gewährleisten. Es ist die provisorische Tastatur zu sehen. Die Taster sind über Pullup Widerstände auf High gezogen und direkt mit dem GPIO Eingängen 0.0–0.15 des Prozessors verbunden.

Bild 3-8 zeigt den Adapter, der das Flachbandkabel über den SMD-Steckverbinder auf eine dreireihige Steckleiste führt. Auf der Unterseite der Platine ist diese mit dem Anschluss für den 50 polige Pfostenstecker verlötet. Der 50 Polige Pfostenstecker ist verbunden mit dem Anschluss für ein externes Display auf dem Evaluationbord. Über die Jumper (roter Pfeil)

lässt sich die Ausrichtung für das Display ändern. Die auf der Oberseite zu sehenden Leitungen sind mit dem LSB von Rot und Blau verbunden und ziehen diese auf Low, da die Displays alle ein 18 Bit Interface besitzen, der Controller das Display aber nur mit 16 Bit ansteuert.

a) b)

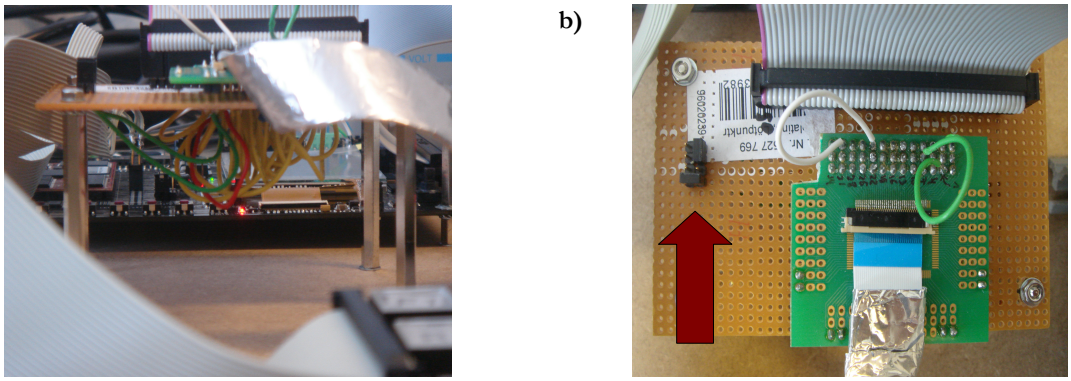
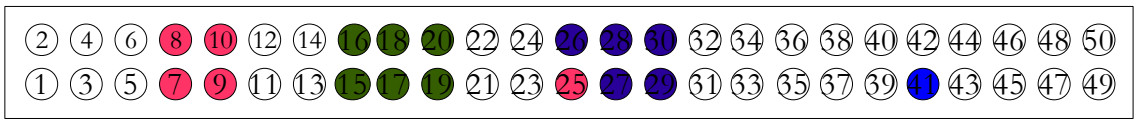


Bild 3-8 Adapter a) Seitlich, b) Oberseite.

Pin 1 vom Display ist mit dem Pfeil markierten Pin 1 des Adapters verbunden (s. Bild 3-9). Die Verkabelung erfolgt weiter in nördlicher Richtung. Die Pinbelegung des *LCD Expansion Connectors* ist in dem Schaltplan des Evaluationboard ersichtlich.

a)



b)

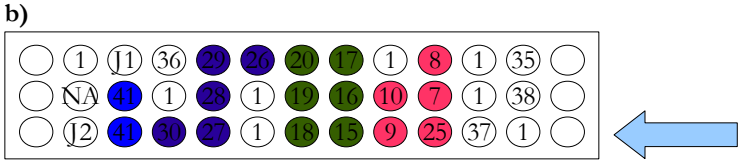


Bild 3-9 Pinbelegung des Adapters a) Expansion Connector b) LCD Connector.

Bild 3-10 zeigt das Evaluationboard. Der grüne Pfeil markiert das von der Tastatur kommende Flachbandkabel, welches auf die interruptfähigen GPIO Eingänge der Platine geht. Da die internen Pins für die CAN Verbindung schon reserviert sind, ist es nötig die Signale über zwei externe Leitungen (roter Pfeil) umzuleiten. Da kein direkter Zugriff auf die Leitungen des Tranceivers möglich ist, wurden die Leitungen an einen Jumper angeschlossen. Somit entspricht die Pinbelegung die der CMA302 und es ist keine Anpassung der Software beim Umstieg auf die Grafikkarte nötig.

Aufgrund der schlechten EMV ist eine zusätzliche Abschirmung für die Displaydaten nötig. Es hat allerdings gereicht eine Abschirmung des Kabels mit Aluminium zu verwenden.

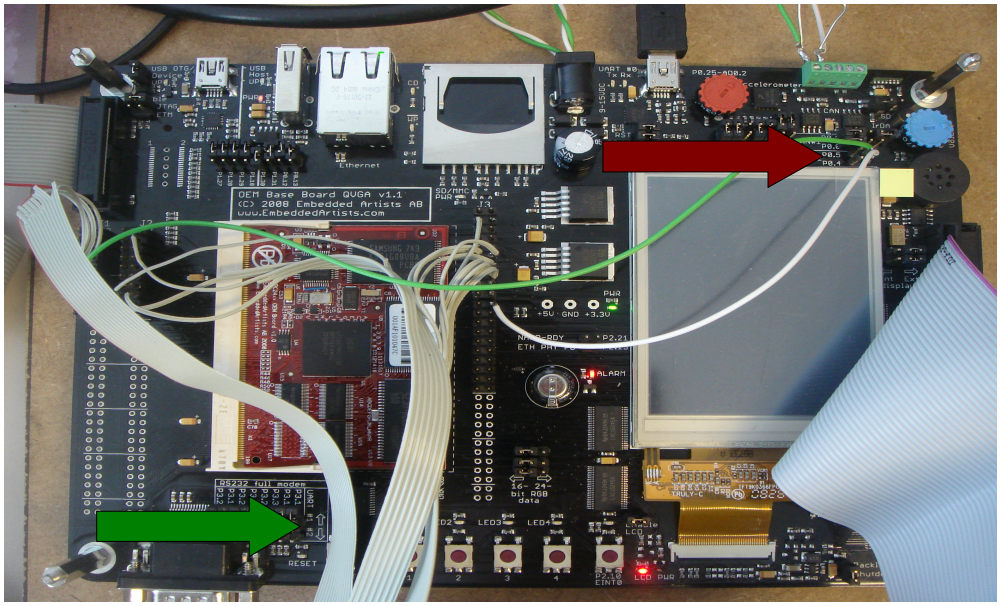


Bild 3-10 OEM Base Board mit Verkabelung.

Der Testaufbau ist in Bild 3-10 als Blockschaltbild zu erkennen. Der 50 polige Stecker zum Adapter wurde aufgrund der Bauweise des Steckers auf dem *OEM Base Board* gewählt.

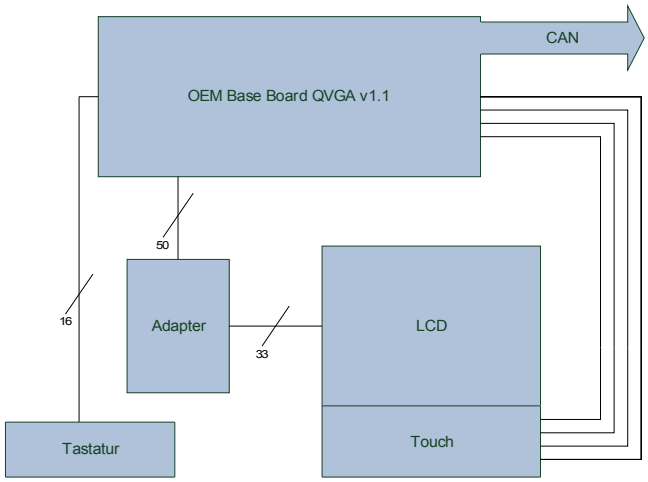


Bild 3-11 Blockschaltbild des Testaufbaus.

3.4 Konfiguration der Peripherie

Der Prozessor hat in der angedachten Konfiguration intern den im Bild 3-12 ersichtlichen Aufbau. Hierbei wurden Komponenten wie Horn, Touch oder EMC der Übersichtlichkeit halber weggelassen.

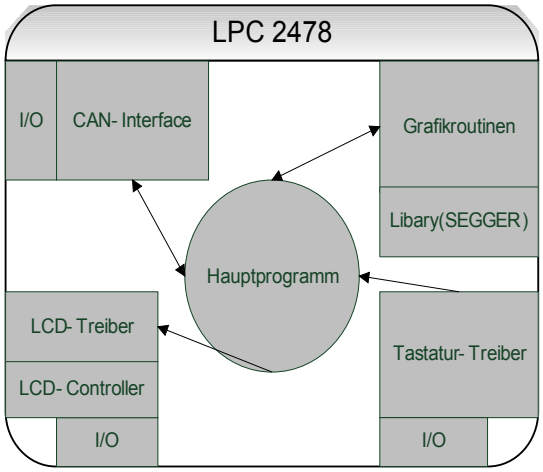


Bild 3-12 Blockschaltbild der Module im LPC2478.

Es ist also eine Reihe von hardwarespezifischen Konfigurationen des LCP2487 nötig um die Peripherie anzusteuern. Hierzu gehören folgende Komponenten:

- LCD Controller
- Tastatur
- CAN
- Horn
- EMC
- RTC
- System Funktionen (Clock Setup, Heap, Stack, Prescaler)

Ein Teil der Komponenten wie der EMC und die System Funktionen (System Funktions) lassen sich über den *Configuration Wizard* einstellen.

Es werden an dieser Stelle nur funktionelle Register erläutert. Das für die CMA302 vorgenommen Pinning befindet sich in der Tabelle in Anhang A3: Pinbelegung LPC2478.

Hierfür wurden die PINSEL, PINMODE, und IODIR gemäß *Kapitel 9.5* und *10.6* der Bedienungsanleitung beschrieben. Alle in diesem Abschnitt beschriebenen Verweise auf die Bedienungsanleitung beziehen sich auf Anhang B4: Manual LPC2478 (CD)

LCD Controller

Der Controller wird über das Schreiben einer eins in das PCONP-Register an der Stelle 20 eingeschaltet. Nach der Konfiguration der PWM1 für die Steuerung der Hintergrundbeleuchtung folgt das Einstellen der nötigen Pins.

Für die Konfiguration des Controllers sind die folgenden Informationen über das zu verwendende LCD nötig:

Höhe und Breite des Panels, horizontale und vertikale Synchronisationszeiten, Bits pro Pixel, single oder dual Panel.

Die für die Anwendung zu beschreibenden Register sind:

`LCD_CTRL, LCD_TIMEH, LCD_TIMEV, LCD_POL, LCD_LE, LCD_CFG.`

Für den Bildspeicher muss noch die Basisadresse des externen Speichers angegeben werden. In Einzelfällen ist es nötig die Kanäle für Rot und Blau zu tauschen oder das Signal komplett zu invertieren. Während das Tauschen von Rot und Blau auf Hardwareseite möglich ist, ist das Invertieren der Farbinformation nur auf Softwareseite realisierbar.

Eine detaillierte Beschreibung der Register ist in *Kapitel 12* der Bedienungsanleitung nachzulesen.[13]

Tastatur

Für die Tastatur werden die interruptfähigen PINS von *Port 0* verwendet. Hierfür müssen lediglich die Register IO0_INT_EN_F und IO0_INT_EN_R beschrieben werden, welche die steigende, bzw. fallende Flanke für die Auslösung des Interrupts einstellen. Es besteht allerdings das Problem, dass sich der LCD Controller und die Interrupt-fähigen GPIO Bänke den selben externen Interrupt teilen. LCDDVD[19] (LSB Blau) löst somit ständig Interrupts aus. Ein einmaliges setzen von EXTINT = (1 << 3) behebt diese Problem allerdings. [14]

RTC

Die Konfiguration der Real Time Clock lässt sich in *Kapitel 26* nachlesen. Über das PCLKSEL0 Register, lässt sich der im Wizard eingestellte Vortakt (Prescaler) ermitteln. Im PCONP Register an der Stelle neun lässt sich nun die RTC einschalten. Die RTC kann einen Interrupt auslösen. Dies Funktion wird in dieser Anwendung allerdings nicht benötigt. Der Interrupt ist einstellbar über das Register RTC_CIIR. Über das RTC_CCR lässt sich die RTC ein- und ausschalten.

Nun müssen nur noch die Prescaler speziell für die RTC eingestellt werden. Dies geschieht über die Register RTC_PREINT und RTC_PREFRAC. Deren Werte über eine Formel aus der Bedienungsanleitung berechnet werden müssen[15].

13 Quellcode: LCDConf.c, LCDConf.h.
14 Quellcode: gpio.c, gpio.h.
15 Quellcode: RTC.c, RTC.h.

CAN

Die CAN Initialisierung geschieht über die Register CAN1BTR, CAN1IER und CANMOD. Hierbei werden Baudrate sowie Interrupt festgelegt und ein Reset durchgeführt. Die detaillierte Beschreibung der Register ist in *Kapitel 18* nachzulesen.[16]

EMC

Die Konfiguration des EMC ist in *Kapitel 5* der Bedienungsanleitung ersichtlich. Der EMC kann über seine Register konfiguriert werden oder es kann der Wizard genutzt werden, dessen Einstellungen in der *Startup* Datei (LPC2478.s) zu finden sind. Bild 3-13 zeigt den für die EMC Konfiguration wichtigen Abschnitt im Wizard an. Die benötigten Informationen finden sich in den Datenblättern der genutzten Speicher.

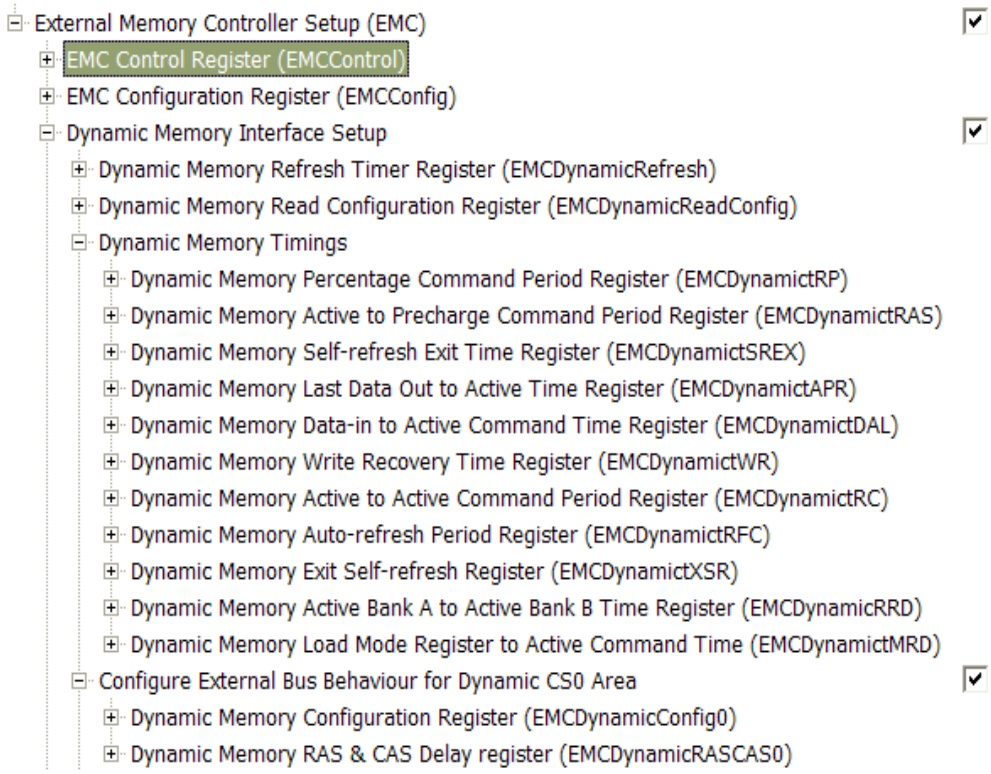


Bild 3-13 EMC Konfiguration.

Ist der externe Speicher nun durch den EMC in das System eingebunden lässt er sich auch in die Speicherverwaltung des Compilers einbinden. Hierfür muss in den *Options for Target 'GU' (µVision3→ Project-→ Options for Taget)* die Startadresse und die Größe des Speichers angegeben werden. In diesem Fall wird die Startadresse bei 0xA0025800 gesetzt, da die ersten 153600 Byte für den Bildspeicher des LCD-Controllers benötigt werden.

16 Quellcode: Can.c, can.h.

Der externe Speicher hat eine 16 Bit Datenanbindung. Der Speicher wird für Daten und zusätzlich als Bildspeicher für den LCD Controller genutzt. Dies hat zusätzliche Zugriffe zur Folge. Die Auslastung des Prozessors wird durch das DMA Interface des LCD Controllers zwar verringert, jedoch greifen beide Komponenten immernoch mit dem gleichen Bus auf den Speicher zu.

Horn

Für die Ansteuerung des *Beepers* wird lediglich ein Pin als Ausgang benötigt. Der Verstärker für den Lautsprecher wird über einen DAC angesteuert. Der Prescaler lässt sich über den Wizard einstellen. Nun müssen noch das PINSEL1 und PINMODE1 Register gemäß *Kapitel 29* der Bedienungsanleitung konfiguriert werden. Über das DACR-Register kann nun ein analoger Wert ausgegeben werden.[17]

System Funktionen

Die Prescaler, sowie Heap und Stack lassen sich über den Wizard einstellen (s. Bild 3-14). Die Timer lassen sich nach *Kapitel 24* konfigurieren. Wobei ein weiterer Prescaler je nach gewünschter Auflösung verwendet werden kann (TxPR). Insgesamt werden vier Timer benötigt. Der Timer für das RTOS (Real-Time-Operation-System[18]) löst bei erreichen des Zahlenwertes in T0MR0 einen Interrupt aus (T0MCR = 1). Diese Funktionalität wird für das RTOS benötigt. Der zweite Timer wurde so konfiguriert, dass der Übergabewert der Funktion der Anzahl an Millisekunden Wartezeit entspricht[19]. Der dritte Timer schaltet den externen Interrupt auf Port 1 bei Tastendruck für 100 ms aus. Der vierte Timer löst zyklisch einen Interrupt aus, welcher eine gedrückte Taste simuliert.

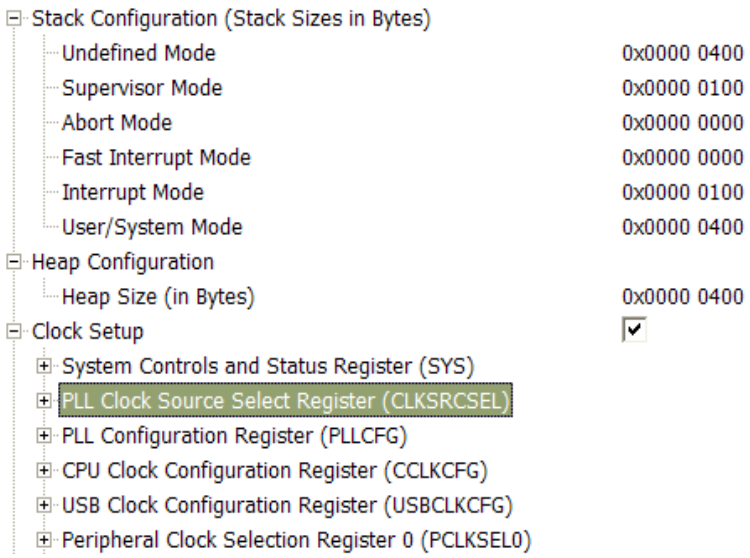


Bild 3-14 System Funktionen.

17 Quellcode: gpio.c, gpio.h.
18 engl. für *Echtzeitbetriebssystem.*
19 Quellcode: LPC2478.s, delay.c, delay,h, rtos_t.c.

Berührungsempfindlichkeit

Da der 4-Wire resistive Touchscreen über den **TSC2046** (s. 29 K. 3.1CMA 302) angeschlossen ist, erfolgt die Ansteuerung komplett über die SPI. Hierbei kann jede Konversion der analogen Daten in drei Übertragungen abgehandelt werden. Das Ansprechen des Touch-Interfaces erfolgt dabei über 1 Byte Datengröße. Im ersten Schritt wird die Konversion gestartet und das Interface konfiguriert. In den nächsten zwei Schritten (bei 12 Bit Konversion) kommen im ersten Byte die drei ersten Bit mit dem MSB beginnend an, im letzten Schritt das restliche Byte. Der **TSC2046** wurde so konfiguriert, dass mit jeder Berührung des Bildschirmes ein Interrupt ausgelöst wird. In der ISR wird nun ein Flag gesetzt, was dazu führt, dass im Hauptprogramm die Funktion GUI_TOUCH_Exec() ausgeführt wird. Diese von SEGGER vorgegebene Funktion übernimmt die Weiterverarbeitung der Berührungsdaten für die Bibliothek, wobei der Treiber für die Ansteuerung zu schreiben ist. Es sind lediglich zwei Funktionsprototypen vorgegeben, die als Rückgabewert den Digitalwert der X/Y Koordinaten erwarten.

Wird nun der Bildschirm berührt erfolgt auf der Treiberseite eine Abfrage über den Druck der Berührung:

$$Pressure = \frac{X \cdot Z_2 - Z_1}{Z_1} \tag{3.1}$$

Die interne Berechnung der Widerstände ist in Bild 3-15 ersichtlich. Überschreitet der Druck einen vorgegebenen Wert können im nächsten Schritt X- und Y-Koordinaten bestimmt werden, wobei es sich an dieser Stelle noch um die Ausgangswerte des A/D Umsetzers handelt, nicht um die Pixelkoordinaten auf dem Display.

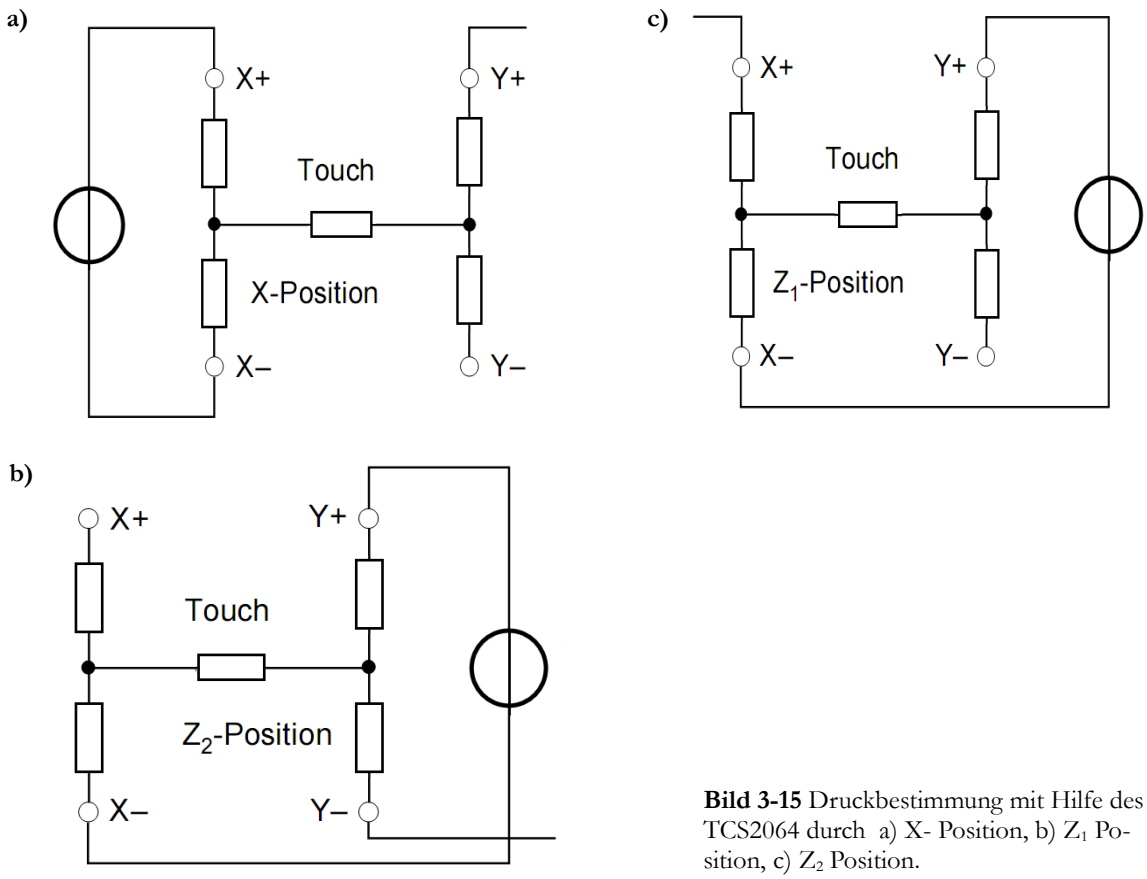


Bild 3-15 Druckbestimmung mit Hilfe des TCS2064 durch a) X- Position, b) Z_1 Position, c) Z_2 Position.

4 Software

Die Software wurde komplett in ANSI C geschrieben. Hierbei wurde speziell auf eine effektive Nutzung des Speichers geachtet. Für eine gute Performance stehen ausreichend Ressourcen zur Verfügung.

Die GUI basiert zwar auf dem Vorgänger Modell, dem SYMAP®, jedoch bietet die 16 Bit Farbtiefe und ein Touchscreen mehr Möglichkeiten in der Gestaltung, was eine komplette Neuentwicklung zur Folge hat.

Es soll die Möglichkeit geben, vier flexible Seiten mit dem *Graphic Builder Tool* am PC zu erstellen und dann auf das System zu laden. Hierbei werden die Daten auf die CU (Control Unit) geladen, welche die Parameter bei einem Reset auf den BUS legt. Bei dem alten Produkt musste für jeden neuen Kunden der zuständige Programmierer die ersten Seiten an die Applikation anpassen, war also für weitere Neuentwicklungen blockiert. Diese Problematik erledigt sich, wenn jeder Kunde sein Menü selber am PC mit einfachen Mitteln zusammenbauen kann.

Zusätzlich soll das Menü so flexibel gehalten werden, dass lediglich eine Parameterliste nötig ist um die Struktur zu ändern. In einem ersten Ansatz sollte das Menü, wie die vier Benutzerseiten über ein externes Tool zusammengesetzt werden können. Dies hat sich allerdings als zu aufwändig erwiesen. Somit sollen sich nun einzelne Menüpunkte und einzelne Parameter über Bitfelder konfigurieren lassen, bzw. soll sich der Menüaufbau anhand der verwendeten Hardware selber an das System anpassen.

Im Gegensatz zum SYMAP® soll durch eine durchdachte Fenster- und Menüstruktur das vertraute Arbeiten wie an einem Computer erzeugt werden.

Es stellen sich also zwei verschiedene Bereiche auf:

1. Der Aufbau des Menüs.
2. Die vier variablen Seiten.

Vor dem Nutzen der Grafikbibliothek muss diese mit den gegeben Parametern initialisiert werden, sowie eine Initialisierung des Controllers und des Bildschirms stattfinden.

Es werden Probleme die sich in den einzelnen Bereichen ergeben, erläutert bzw. verschiedene Lösungsansätze diskutiert. Der Rest des Kapitels wird sich mit der Lösung und Realisierung des Projektes beschäftigen.

Die *Touchfunktionalität* wurde zu Testzwecken schon komplett implementiert, wird jedoch erst in einer späteren Serie Anwendung finden. Die Menüführung ist in dieser Version noch auf die vom Vorgängermodell übernommenen Tastatur aufgebaut, kann jedoch auch mit dem Finger bedient werden.

4.1 Betriebssystem

Hauptprogramm

Die GUI basiert auf einer Art RTOS[20]. Es handelt sich hier um eine sehr einfache Art ein Echtzeitbetriebsystem zu realisieren und arbeitet mit sog. *Trigger Tasks*.

Ein Timer löst alle *10 ms* einen Interrupt aus. In dessen Interrupt Service Routine werden nun bei jedem Durchlauf, vorher initialisierte Variablen runtergezählt. Hierbei entspricht der Initialisierungswert einer Zahl, die nach der Anzahl der Durchläufe, die gewünschte Zeit ist. Die Variable für *20 ms* beinhaltet also eine zwei, was nach zwei Durchläufen dazu führt, dass der Status für *20 ms* gesetzt wird.

In der *main()-Funktion* wird nun auf diesen Status abgefragt und es ist möglich eine Task zu schreiben die alle *20 ms* ausgeführt wird. Reicht nun die Zeit zwischen zwei Tasks nicht aus, wird die entsprechende Task dennoch ausgeführt, da der Status bereits gesetzt wurde, allerdings nicht mehr exakt nach *20 ms*.

In diesem Projekt wurden Tasks für *10 ms, 20 ms, 50 ms, 100 ms, 200 ms, 500 ms, 1 s, 2s* und *10s* vorgesehen. Ergibt sich am Ende das nicht alle Tasks benötigt werden, können diese hinterher entfernt werden.

Zum jetzigen Zeitpunkt wurden die folgenden Task implementiert:

- Eine ständige Abfrage, ob CAN- Frames eingetroffen sind.
- Alle *10 ms* wird ein Flag abgefragt, ob eine Berührung des Bildschirms stattgefunden hat.
- Alle *20 ms* wird die Anzeige mit den Werten einer Berührung des Bildschirmes, oder den Informationen über ein eingetroffenes CAN-Frame gefüllt. (Dies findet nur zu Debugzwecken statt.)
- Alle *100 ms* wird abgefragt, ob eine Taste gedrückt wurde und die Anzeige aktualisiert.
- Alle *250 ms* wird die aktuelle Seite mit den aktuellen Messwerten oder Parametern aktualisiert.
- Alle *500 ms* wird der Systemstatus geprüft.
- Jede Sekunde geschieht eine Überprüfung des CAN, sowie eine Aktualisierung der dargestellten RTC. Außerdem wird der *Watchdog* gefüttert.

Echtzeituhr

Die Echtzeituhr kann mit Zeit und Datum initialisiert werden und wird hauptsächlich benötigt um Ereignissen im Verlauf einen Datumsstempel für eine spätere Diagnose zu geben. Einmal initialisiert und konfiguriert arbeitet die RTC als eigenständige Komponente.

20 Quellcode: main.c, rtos.c. rtos.h.

CAN

Tritt eine CAN Nachricht ein, wird ein Interrupt ausgeführt der nach einer Abfrage um welchen BUS es sich handelt (es sind zwei CAN Busse vorgesehen), eine Funktion aufruft die, die CAN Nachricht in eine Warteschlange schreibt. Die Warteschlange wurde vorerst mit einer Größe von 20 definiert. Spätere Tests mit dem Endgerät werden ergeben was für eine Größe angemessen sein wird.

Mit jeder Nachricht in der Warteschlange wird ein Zähler inkrementiert. Die in der *main()-Funktion* beim Erhalt einer oder mehrere Nachrichten zyklisch aufgerufene Funktion *CAN1_get_rx_queue()* ermöglicht nun einen Zugriff und eine Verarbeitung der CAN Nachrichten außerhalb der Interrupt Routine des CAN.

Ein Akzeptanzfilter ermöglicht das Filtern der eintreffenden Frames und verhindert einen Interrupt bei Nachrichten die für die anderen Komponenten des Systems vorgesehen sind. Das Filter ist in den Controller integriert und funktioniert in dem ein Speicherbereich ausgelesen wird, indem die zu filternden IDs stehen.

Neben dem Ablegen von Parametern sind auch Steuerbefehle vorgesehen wie z.B. des Resetbefehl. Die Daten treffen zyklisch ein oder werden nach Anforderung gesendet.

Tastatur

Wie auf Bild 4-3 zu erkennen existieren 15 Tasten die verschiedene Funktionen ausüben. Wird eine Taste gedrückt wird in der ISR über das entsprechende Statusregister geprüft welche der Tasten den Interrupt ausgelöst hat und ein Flag für die entsprechende Taste gesetzt. Hierfür sind nur 2 Zeilen nötig:

```
for (i = 14; i >= 0; i--)
     if (IO0_INT_STAT_F & (1 << i ))
```

Somit ist beim Eintreten der if-Anweisung immer die gedrückte Taste in der Variablen i gespeichert und ein Flag kann gesetzt werden. Laut *Segger* sind deren Verarbeitungsfunktionen für Tastatureingaben interruptfähig, dies hat sich allerdings als nicht wahr herausgestellt.

Die eigentliche Verarbeitung der Tastaturbefehle findet in der *main()-Funktion* statt. Ist das Flag für eine Taste gesetzt, wird in der Tastaturfunktion über eine *switch-case* Anweisung die geforderte Aktion ausgeführt. Da einige Tasten in Kombination mit einer weiteren Taste bedient werden können, kann in diesen fällen das *Break* weggelassen werden.

Die Tasten sind über einen Zähler entprellt der bei Tastendruck den Interrupt für 100 ms ausschaltet. Ein Scrollen für die Tasten *UP* und *DOWN* wird über einen weiteren Zähler ermöglicht, der bei gedrückter Tasten alle 30 ms einen Interrupt auslöst. Sollten bei der Weiterführung des Projektes ein Mangel an Zählern bestehen, währe es auch möglich das Scrollen als Task zu realisieren[21].

Berührungsempfindlichkeit

Um den berührungsempfindlichen Bildschirm effektiv zu nutzen sind eine Reihe von Konfigurationen notwendig. Dazu gehört die Ausrichtung des Bildschirms, sowie die genutzte Auflösung. Um die Pixelkoordinaten zu bestimmen benötigt die Bibliothek die Minimum-

21 Quellcode: gpio.c, gpio.h.

und Maximumwerte des A/D Umsetzers. Diese können über die Bestimmung von zwei Koordinaten(z.B. oben-links und unten-rechts) bestimmt werden. Die physikalischen Werte werden abgespeichert und der folgenden Funktion übergeben.

```
int GUI_TOUCH_Calibrate(int Coord, int Log0, int Log1, int
                        Phys0, int Phys1);
```

mit: `Coord` – X oder Y Koordinate
`Log0` – Soll-Minimumwert
`Log1` – Soll-Maximumwert
`Phys0` – Ist-Minimumwert
`Phys1` – Ist-Maximumwert

Der Rückgabewert ist hierbei eine Null, bei erfolgreicher Kalibrierung, und eine Eins, bei einer fehlerhaften Kalibrierung. Die Grafikbibliothek nutzt nun intern die kalibrierten Daten. Eine Berührung an der Stelle X/Y (z.B. 30/120) hat nun zur Folge dass die abgefragten Koordinaten auch X/Y (30 /120) sind und nicht mehr die Ausgangswerte des D/A Umsetzers.[22]

Bild 4-1 zeigt die Bildschirmausgabe der Kalibrierungsfunktion. Die Eingabe funktioniert allerdings auch mit „geschätzten“ Werten für den D/A Umsetzer. Nach berühren der Koordinate oben-links, erscheint ein weiterer Punkt unten-rechts. Sollte diese Genauigkeit nicht ausreichen können weitere Messpunkte oben-rechts und unten-links gewählt werden, um über eine Mittelung eine exakte Koordinatenbestimmung zu bekommen.

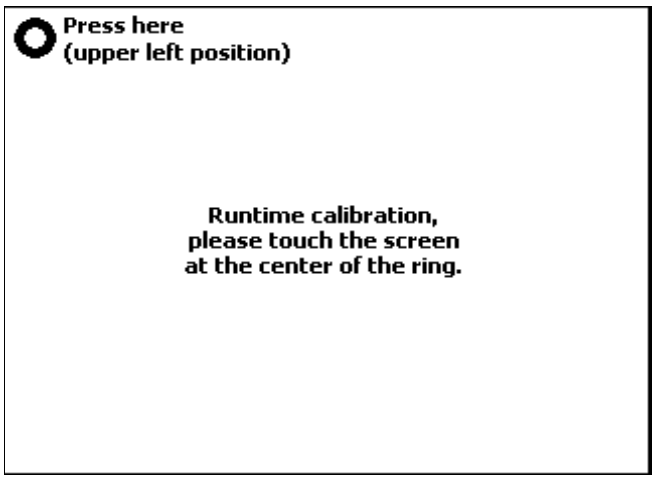


Bild 4-1 Kalibrierung des Touchscreen.

22 Quellcode: touch.c, touch.h.

4.2 Menü

Die Menüstruktur sollte von dem Vorgängermodell übernommen werden. Da es sich hierbei jedoch um ein monochromes Display handelt, welches keine Menüstruktur an sich hat, muss die Struktur von Grund auf neu überdacht werden. Das alte Menü ließ sich lediglich über vier Tasten steuern, die je nach Menüpunkt andere Funktionen bekommen haben. In Bild 4-2 sind Beispielhaft die Funktionen SCHUTZ, ALARM und ENDE aufgeführt, die sich ergeben wenn man in den Menüpunkt *Display* anwählt. Es ist somit nicht möglich einzelne Seiten direkt anzuwählen. Ein „wandern" durch die gesamte Menübaumstruktur ist nötig um an die gewünschte Stelle zu gelangen. [9]

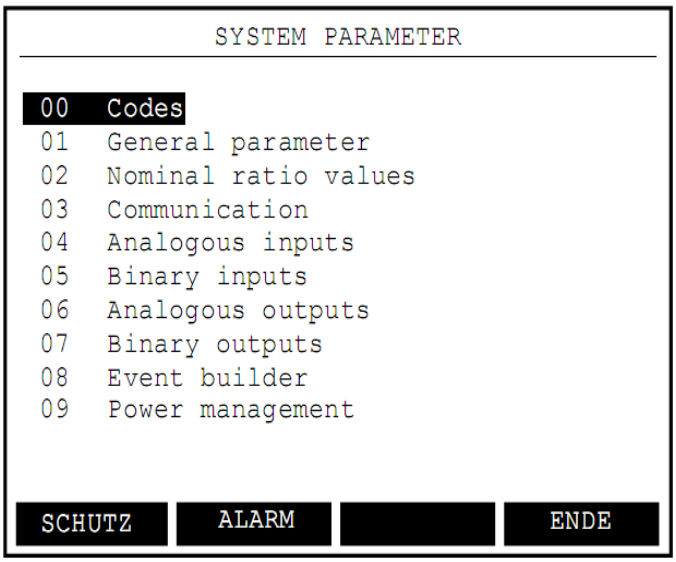


Bild 4-2 Menü des Symap-Compact Vorgängermodells.

Das SYMAP-Compact soll nun eine vom Computer bekannte Bedienungsoberfläche bekommen. Somit wäre eine Bedienung vereinfacht, da die Benutzer in der Regel mit Computern vertraut sein werden.

Angedacht war es, das Menü über ein externes Tool generieren zu können. Das Problem der Vorgängergeneration war die steife Struktur die für jede Änderung ein umprogrammieren der Software benötigte. Ein flexibles Menü, angepasst auf die genutzte Peripherie macht technisch und ökonomisch einen Sinn und erspart eine spezielle Anpassung. Erste Ansätze haben jedoch ergeben dass der Aufwand das Menü extern gestalten zu wollen zu groß werden würde.

Der zweite Ansatz war nun ein Menü mit einer festen Struktur, jedoch zusätzlich mit der Möglichkeit einzelne Seiten oder Menüpunkte über einen Parameter hinzuzufügen und entfernen zu können, bzw. ein sich selbst anpassendes Menü.

Bild 4-3 zeigt das Tastatur-Design, welches für das SYMAP-Compact vorgesehen ist, und auf dessen Basis das Menü aufgebaut werden soll. Die farbigen Tasten haben festgelegte Funktionen. Die vier F-Tasten sowie die Cursor Tasten und die Entertaste sind jedoch variable und ermöglichen das Navigieren durch das Menü und die Steuerung innerhalb des Menüs.

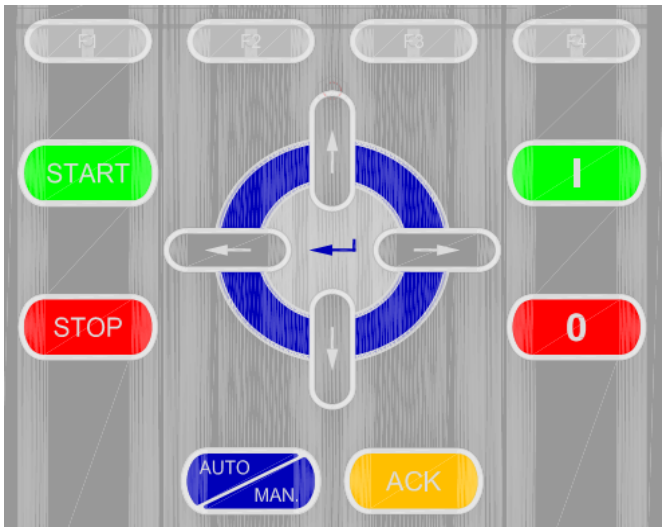


Bild 4-3 Tastaturlayout. - Symap Compact.

Der Benutzer soll über das grafische Interface die Möglichkeit bekommen alle von ihm gewünschten Messwerte darzustellen, aber auch Einstellungen am Gerät vorzunehmen.

Ein solches Menü in QVGA[23] zu zeichnen, welches zusätzlich über den RGB Farbraum verfügt, ist eine sehr zeitaufwändige Arbeit.

Eine Analyse des Marktes hat ergeben, dass es bereits vorgefertigte Bibliotheken für diese Art von Anwendungen gibt. Da die Auswahl in diesem Bereich sehr beschränkt ist, hat sich die Bibliothek von *Segger* herauskristallisiert. Das Aussehen der Objekte entsprechen den Vorstellungen für das Symap-Compact. Somit entfällt das Design der Tasten und Fenster was eine Konzentration auf das eigentliche Problem des Aufbaus, der Struktur und der Verarbeitung der Daten ermöglicht.

Nachteil ist die Abhängigkeit von der Bibliothek. Eine Einarbeitung ist nötig um mit den gegebenen Werkzeugen umzugehen. Die durch den C-Code gegebene Flexibilität der Struktur täuscht. Eine solche Struktur baut aufeinander auf und ein Eingriff können Konsequenzen für das gesamte System haben.

Für die Berührungsempfindlichkeit existieren zu diesem Zeitpunkt noch keine Vorgaben und es muss bedacht werden, dass es sich um ein Gerät für die Industrie handelt, welches bei einer solchen Steuerung sehr leicht verschmutzen könnte. Nach einer ersten Serie von Geräten mit Tastatur, ist eine weitere mit Touch-Funktionalität gewünscht, um Kosten zu reduzieren.

Der erste Schritt der Entwicklung soll nur darstellende Elemente enthalten. Eine Parametrierung erfolgt hierbei lediglich über einen extern angeschlossenen Computer. Im nächsten Schritt wird dann die Parametrierung über das Menü ermöglicht, es müssen somit ausreichend Ressourcen zur Verfügung stehen und die Struktur für die Parametrierung geschaffen werden.

23 Entspricht einer Auflösung von 320 x 240 Pixel.

4.2.1 Aufbau

Symap (Vorgänger)

Der Vorgänger des Symap-Compact hat einen linearen Aufbau. Wird die Taste *ENTER MENU* gedrückt, werden den Tasten *F1-F4* die in Bild 4-4 dargestellten Funktionen zugeteilt. Ein Drücken der entsprechenden Taste führt nun zu weiteren Untermenüs, oder ggf. auf andere spezielle Seiten.

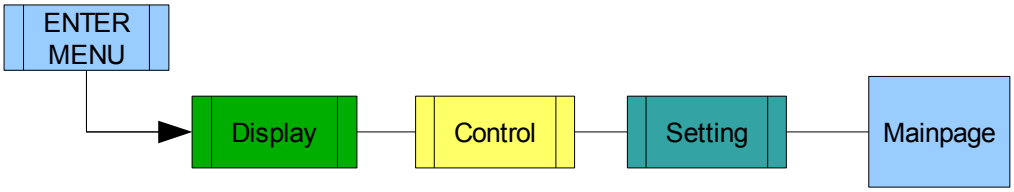


Bild 4-4 Ausschnitt aus der Menüstruktur des Vorgängermodells.

Über die Taste *F1* kommt man nun in den Unterpunkt *Display*, welcher wiederum in drei über die *F*-Tasten zu erreichenden Seiten eingeteilt ist. Wählt man nun eine dieser Seiten an bekommt man eine Auswahl an Messgrößen zur Veranschaulichung (s. Bild 4-5).

Hierbei bleibt auf der rechten Seite eine vom Benutzer gewünschte Grafik erhalten und auf der linken würden die Messwerte dargestellt werden. Es besteht in diesem Fall zusätzlich die Möglichkeit jeden der anderen beiden Menüpunkte auszuwählen.

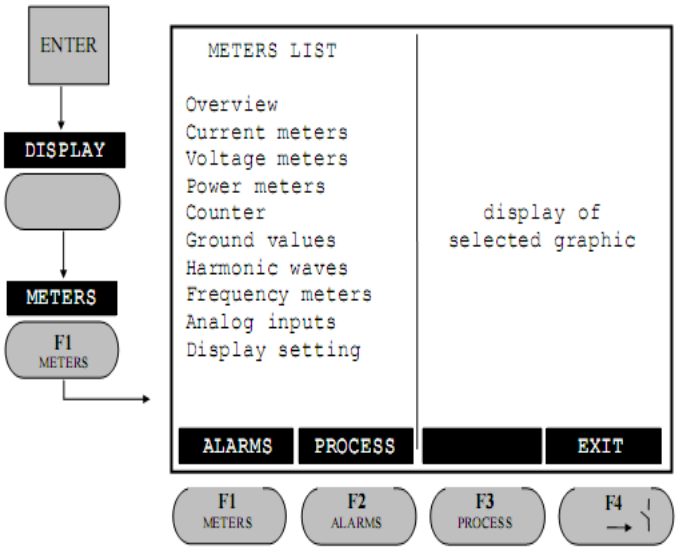


Bild 4-5 Menüstruktur des Vorgängermodells.

Die Displayseiten beinhalten auch noch den Sonderfall, dass sich diese auch direkt über die *F*-Tasten von der Hauptseite aus darstellen lassen. Sie sind somit über die Menüstruktur anwählbar, oder über die *F*-Tasten durchzuschalten.

Der Menüpunkt *Control* führt nach einer Passwortabfrage zu drei Seiten mit verschiedenen Einstellungsmöglichkeiten für das System. Bei diesen besteht allerdings nicht mehr die Möglichkeit von einer einmalig angewählten Seite zu einer anderen des Menüpunktes Control zu wechseln.

Die Menüstruktur beinhaltet somit :

1. Auswahllisten die auf Seiten führen die den halben oder ganzen Bildschirm ausfüllen.
2. Menüunterpunkte die Querverweise zu anderen Menübereichen haben.
3. Seiten mit darzustellenden Objekten.
4. Seiten mit Objekten für Interaktionen.
5. Passwortabfragen.
6. Seiten nur für den Service auf die der User keine Zugriff hat.

An dieser Stelle sollte nur der generelle Aufbau des Menüs veranschaulicht werden.

Des Weiteren gibt es vom SYMAP®-BCG ausgehend fünf LEDs für Status, Kommunikation, System, Alarm und Trip. Es gibt vier Felder bestehend aus drei 7-Segment Anzeigen mit jeweils drei LEDs. Diese stellen Spannung, Strom, Leistung und Frequenz jeweils in Prozent oder der jeweiligen Einheit dar (Bild 4-6). Die Hauptseite besteht aus einer Übersicht über die Peripherie und einer auf das System angepassten Zeichnung.

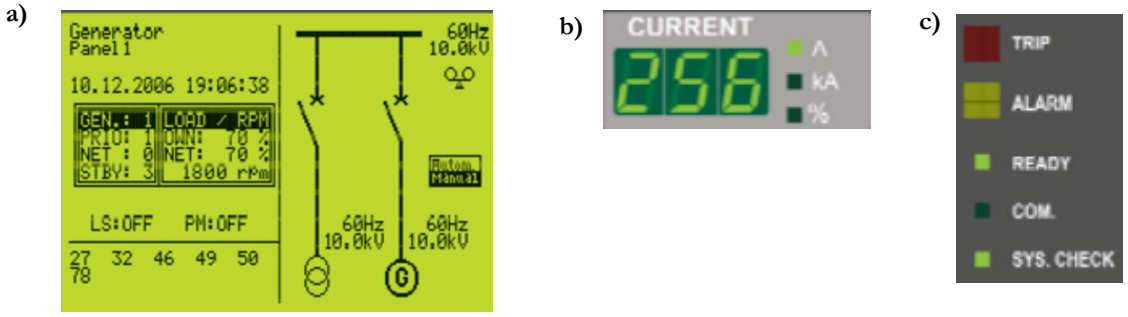


Bild 4-6 Elemente des Vorgängermodells a) Hauptseite, b) 7-Segment Anzeige, c) LEDs.

Symap Compact

Ein aus z.B *Microsoft Windows* bekanntes Menü hat den Vorteil gegenüber der aus den SYMAPS bekannten Menüführung, das jeder Menüpunk direkt angesprochen werden kann. Die Anzahl der Elemente ist nicht auf drei beschränkt (drei Untermenüs + *F4* für Zurück). Der Menüpunkt *Service* wurde aus dem Bereich *Setting* entfernt und ist nun ein eigenständi-

ger Menüpunkt, jedoch für den Endnutzer nicht direkt anwählbar. Zu einem späteren Zeitpunkt wird dieser komplett verschwinden. Bild 4-7 zeigt die Umsetzung der Menüstruktur.

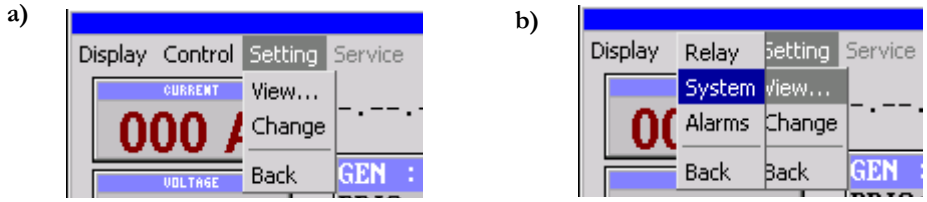


Bild 4-7 Symap Compact Menü a) Menüpunkt, b) Untermenü.

Aufgrund der Spezifikation, dass vier Seiten frei Strukturierbar sind und dadurch dass externe Elemente, wie die 7-Segment Anzeigen nun fehlen, war die Erstellung einer Haupt- oder Defaultseite nötig, welche alle wichtigen Elemente beinhaltet.

Hierbei besteht die Hauptseite aus dem Bereich unter dem Menü, bis zum Anfang der Statuszeile. Der Rest wird in jedem Fenster sichtbar sein. Diese Hauptseite, sowie die dynamischen Seiten, sind alle über die Taste F3 durchschhaltbar. Bild 4-8 zeigt wie LEDs, 7-Segment Anzeigen und Echtzeituhr realisiert wurden.

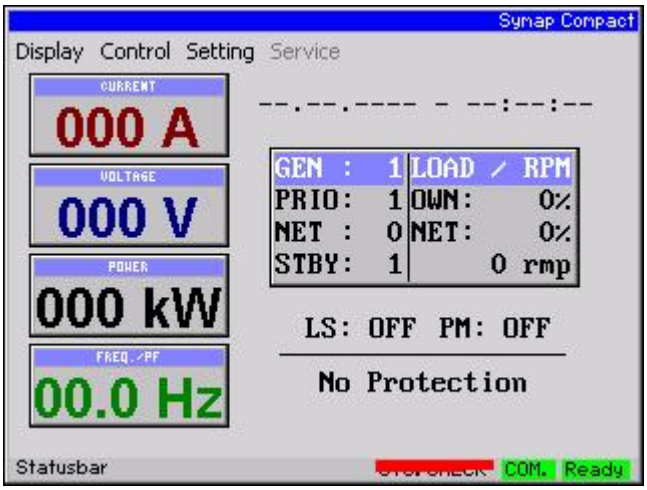


Bild 4-8. Symap Compact Hauptseite.

Seiten wie die Menüunterpunkte von *Display* und *Setting*, lassen sich über eine *Multipage* realisieren.

Das Menü an sich erscheint nur wenn ein Taste gedrückt wird und verschwindet bei der Auswahl einer Seite oder eines Menüpunktes, um eine größtmöglichen Bereich für die Darstellung zur Verfügung zu haben.

Über die Cursortasten für links und rechts lässt sich nun zwischen den Bereichen wechseln, wobei ein Druck auf die Entertaste zu einer Vorschau der darzustellenden Messwerte oder Einstellunmöglichkeiten führt (s. Bild 4-9).

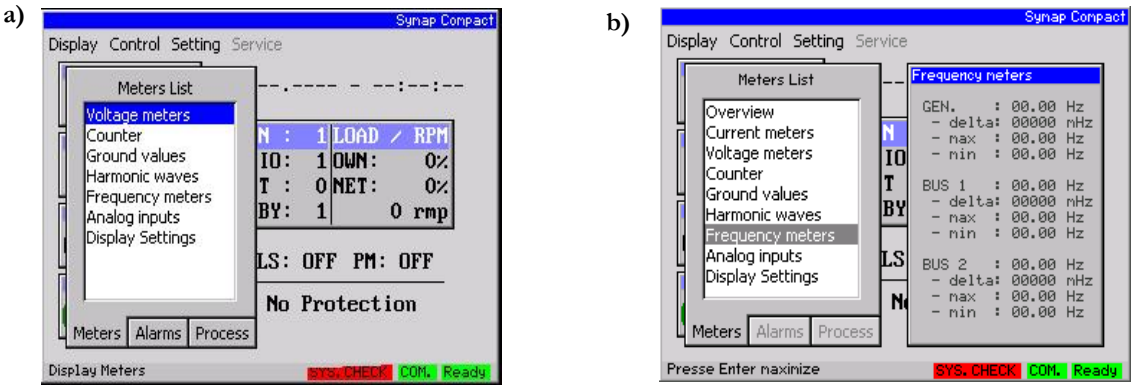


Bild 4-9 Multipage a) Menüpunkt b) Ausgewählter Menüpunkt.

Über die Cursortasten für hoch und runter lässt sich nun eine Vorschau jeder Seite aufrufen wobei im Titel und in der *Multipage* die aufgerufene Seite markiert wird. Ein weiterer Druck der Entertaste zeigt die Seite im Hauptbildschirm (Bild 4-10).

Für die Serie mit dem berührungsempfindlichen Bildschirm wird anstelle des Vorschaufensters ein Fenster mit zwei Tasten erscheinen, als Bestätigung, ob die Seite aufgerufen werden soll, evtl. mit Kurzbeschreibung.

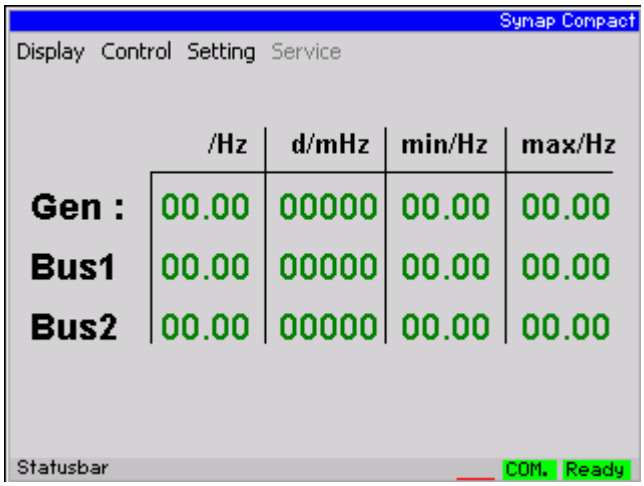


Bild 4-10 Messwertseite. - Frequenzen.

Für Bereiche in denen Einstellmöglichkeiten vorgenommen werden bietet es sich an Listviews zu verwenden. Hierbei macht eine Anwahl des Elementes eine Änderung der Einstellung möglich (s. Bild 4-11).

Bei den hier vorgestellten Elementen handelt es sich um eine Auswahl, um die Realisierung der Menüführung im Symap Compact zu demonstrieren.

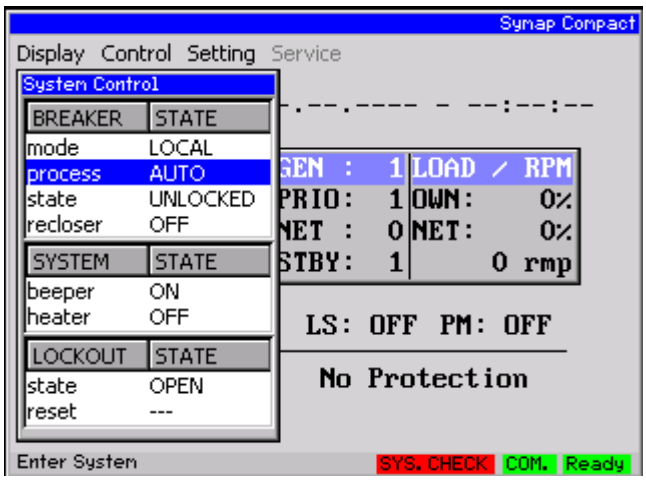


Bild 4-11 Anwendungsbeispiel. *-Listview.*

Wie aus *Anhang B1: Quellcode* ersichtlich, handelt es sich bei den Elementen nicht um Platzhalter, sondern um komplett realisierte Bereiche denen lediglich der Eingabeparameter fehlt. Es wurden weitere Elemente realisiert die nach dem hier dargestellten Prinzip erstellt wurden.

4.2.2 Realisierung

GUI

Die gesamte grafische Oberfläche basiert auf Fenstern, die wie Objekte zu behandeln sind. Da diese Fenster alle Abhängigkeiten untereinander besitzen, ist es nötig sich eine Struktur der späteren Menüführung auf Basis dieser Abhängigkeiten zu erstellen. Hierfür ist es nötig sich einen Einblick in die Grafikbibliothek zu verschaffen, da einige Objekte auf mehreren Fenstern basieren.

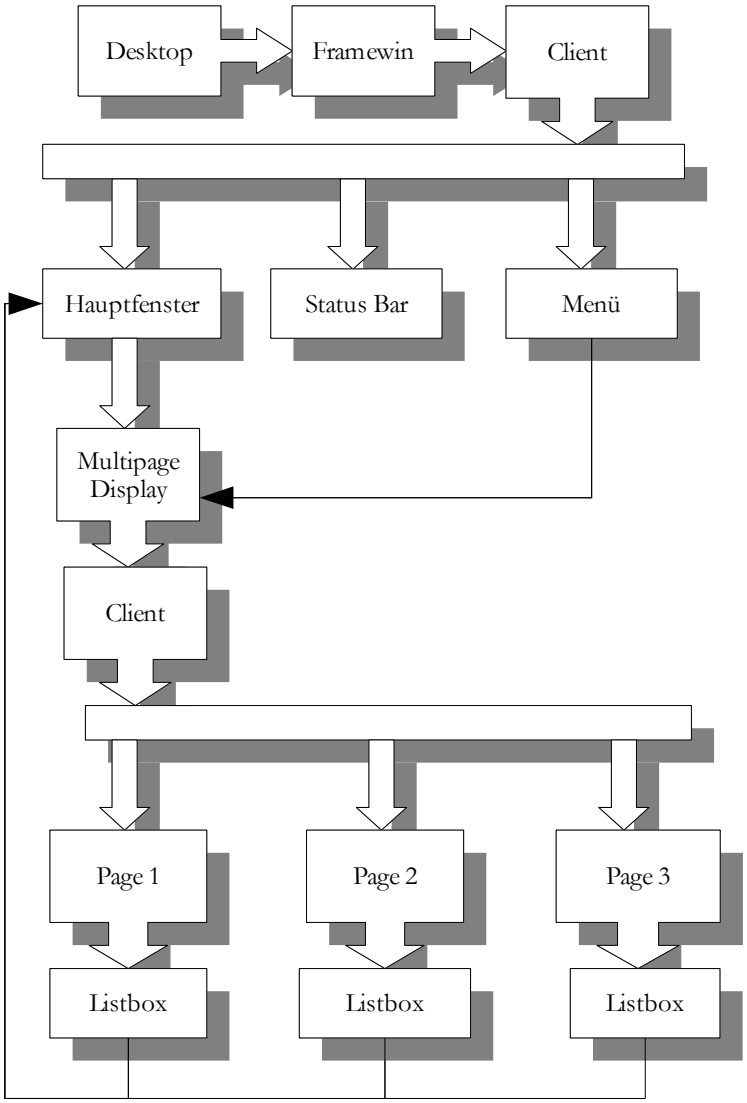


Bild 4-12 Abhängigkeit der Fenster innerhalb des Menüs.

Bild 4-12 zeigt einen Teil der Fensterbeziehungen zur Veranschaulichung. Das Desktopfenster wird hierbei immer automatisch erstellt, bildet somit den Vater für alle folgenden Fenster. Hierbei stellen die Blockpfeile eine Abhängigkeit dar und die Linienpfeile eine Ver-

knüpfung. Es ist somit möglich von einer der dargestellten *Listboxen* auf eine der Displayseiten zuzugreifen. Alle darzustellenden Seiten werden im Hauptfenster gezeichnet, alle geöffneten Fenster sind *Clientfenster* des Hauptfensters. Dies bietet den Vorteil dass das Löschen aller angehängter Fenster automatisiert ist. Wird eine neue Seite aufgerufen, wird das Hauptfenster einmal gelöscht, was auch zum löschen aller darin geöffneter Fenster führt. Die Statuszeile ist im *Clienten* des Hauptfensters verankert um ständig sichtbar zu sein. Das bei Tastendruck erstellte Menü ist am selben *Clienten* angehängt und überdeckt immer das Hauptfenster, ist somit immer sichtbar.

Der Seitenaufbau geschieht dynamisch nach Angaben in einem Bitfeld. Hierbei orientiert sich der Seitenaufbau an den Festlegungen in Tab. 4.1. Alle in der Tabelle dargestellten Seiten haben mehrere Verankerungen im System. Die Seiten existieren als Seite an sich und als anwählbares Textfeld mit Vorschauseite, welches beim Anwählen auf die eigentliche Seite springt. Die Verankerung des Fensters wurde bereits im Zusammenhang mit der Tastatur erläutert (s. 4.1). Die Verankerung in Bezug auf das Textfeld wird im Zusammenhang mit der *Multipage* erläutert werden.

Es existieren weiterhin Bitfelder für den gesamten Settingbereich, der eine komplett neuen Aufbau im Vergleich zum Vorgänger bekommt. Es sind nicht nur Zeilen auf einer Seite variable, sondern der gesamte Aufbau des Menübaums.

Tab. 4.1 Belegung des Bitfeldes für den dynamischen Menüaufbau[a)].

Bit 0 - 12	F1 Seiten	Bit 0- 9	F2 Seiten	Bit 0- 13	F3 Seite
0	Überblick	0	Aktive Alarme	0	Überblick
1	Strom	1	Alarm Gruppen	1	Benutzerseite 1
2	Spannung	2	Aktive Events	2	Benutzerseite 2
3	Zähler	3	Events Verlauf	3	Benutzerseite 3
4	Bodenrichtwert	4	Detaillierter Verlauf	4	Benutzerseite 4
5	Harmonische Wellen	5	Verriegelung Seite	5	Maschinen Überblick
6	Frequenzen	6	Systemfehler Seite	6	Leistung Management
7	Analoge Eingänge	7	Reserve	7	Ladeseite
8	Display Einstellungen	8	Reserve	8	PM Regulationen
9	Reserve	9	Reserve	9	Synchronisierer
10	Reserve	...	...	10	Unterbrecher Zähler
11	Reserve			11	Reserve
12	Reserve			12	Reserve
...	...			13	Reserve

a) Bei dem Bitfeld handelt es sich um zwei U32 Variablen, die über eine Funktion wie eine U64 Variable behandelt werden können.

Das Programm beginnt mit der Initialisierung des Haupfensters[24]. Dies ist nötig da alle anderen Fenster an dieses Hauptfenster gehängt werden. Die *Handles* für das *Framewindow*, sowie für Menü und Statuszeile, werden hierbei global angelegt, da auf diese Komponenten des Öfteren aus unterschiedlichen Bereichen zugegriffen werden muss. Zusätzlich wird ein globaler *Handle* für die darzustellenden Fenster benötigt. Es existiert zwar jedes Fenster als

24 Quellcode: menuwin.c.

eigenes Objekt, da aber immer nur eines zur Zeit dargestellt wird, ist es möglich immer wieder den selben Handle zu nutzen. Auch führt ein löschen dieses *Handles* vor der Generierung eines neuen Fensters dazu dass evtl. existierende *Kindfenster* jedes Mal vollständig gelöscht werden und nicht im Speicher verbleiben.

Menü

Die Initialisierungsfunktion erstellt den Hauptframe und Statuszeile, sowie drei Statusanzeigen. Am Ende der Funktion wird ein Fenster als Startseite aufgerufen.

In der selben Datei ist die Funktion *_AddMenuItem()* zu finden, welche über eine Reihe von Parametern ermöglicht in der Funktion

```
WM_HWIN _CreateMenu(WM_HWIN hParent, U8 menu, U8 io, U8
                    prot, U8 com)
```

mit: `WM_HWIN hParent` – Der Handle an den das Fenster gehängt wird.
`U8 menu` – Aufbau des gesamten Menüs.
`U8 io` – Aufbau des Unterpunktes IO.
`U8 prot,` – Aufbau des Unterpunktes Protect.
`U8 com` – Aufbau des Unterpunktes Communication.

das Menü nach dem *Baukastenprinzip* zusammen zubauen. Dabei entspricht der Rückgabewert dem Handle des Menüs, welcher beim Löschen wieder benötigt wird. Es ist neben der Verankerung innerhalb des Menüs wichtig den Elementen eine ID zu geben, um sie später identifizieren zu können. Das Menü wird beim Drücken der ENTER-Taste erstellt und löscht sich automatisch beim Verlassen, oder bei der Anwahl eines Menüpunktes. Für das Löschen des Menüs ist die Funktion

```
void deleteMenu(MENU_Handle hMenu)
```

nötig, da innerhalb des Menüs keine Hierarchie besteht. Ein erneutes aufrufen des Menüs würde dann zum Speicherüberlauf führen.

Die Callbackroutine des Clientwindows im Hauptfenster, übernimmt die Verwaltung des Menüs. Tritt in dieser Routine die Nachricht WM_MENU auf, kann aus der Datenstruktur *pMsg->Data.p* gelesen werden, was für eine Aktion stattgefunden hat. Die abgeleitete Struktur *pData->MsgType* stellt nun folgende Nachrichten zur Verfügung:

1. `MENU_ON_ITEMPRESSED:`

In diesem Fall wird über

```
MENU_GetItem(MENU_Handle, U16, MENU_ITEM_DATA*)
```

abgefragt, welcher Menüpunkt betätigt wurde und in welchem Zustand dieser sich befindet. Die Parameter sind in eingetragener Reihenfolge: *Menü Handle, Menüpunkt, Zustand*. Die Informationen sind in der Datenstruktur vorhanden. Dies ist von Bedeutung, da es Elemente wie die des *Control* Menüs gibt, die erst über ein Passwort zugänglich gemacht werden sollen. Die Message *MENU_ON_ITEMPRESSED* tritt aber in jedem Fall auf, auch wenn das Objekt ausgegraut wurde und bietet somit die Möglichkeit eine Passwortabfrage einzubauen indem eine Aktion stattfindet nachdem das ausgegraute Objekt angewählt wird.

2. MENU_ON_ITEMACTIVATE:

Diese Nachricht wird genutzt um in der Statuszeile eine Information über den angewählten Menüpunkt zu zeigen. Sind die IDs in der selben Reihenfolge definiert wie die dazugehörigen Statustexte, ist eine einfach Darstellung über die ID-Nummer möglich in dem in Verbindung mit einem Offset über die ID direkt die dazugehörige Nachricht ausgewählt wird.

3. MENU_ON_ITEMSELECT:

Über eine Abfrage der ID wird selektiert welcher Menüpunkt ausgewählt wurde. Die entsprechende Seite wird daraufhin geöffnet, oder eine Aktion gestartet.

Das Menü wird erst aufgerufen nachdem die ENTER Taste gedrückt wird und baut sich jedes mal in der Form auf, dass nur die für den Benutzer zugänglichen Menüpunkte sichtbar sind, wobei sich die Unterpunkte der jeweiligen Hardwarekonfiguration anpassen. Über bestimmte Passwöter lassen sich weitere Menüpunkte für Konfiguration und Service freischalten. Die Struktur des Menüpunktes *Settings* orientiert sich an der für das Symap-Compact gegebenen Struktur aus Anhang B8: Menüstruktur – Settings (CD).

MULTIPAGE

Der Menüpunkt Display führt nun auf eine Multipage[25], wie in Bild 4-9 zu sehen, indem über die Funktionen:

```
1. hMutltipageDisplay  = MultiPage();
2. MULTIPAGE_SelectPage(hMutltipageDisplay, 0);
```

die Multipage generiert wird, sowie der gewünschte Reiter ausgewählt.

Das dynamische Erstellen der Multipage geschieht, indem zuerst der Rahmen an das aktuelle Fenster gehängt wird sowie Größe, Koordinaten und ID festgelegt werden.

Nun werden nacheinander drei Dialoge, bestehend aus Fenster sowie Text- und Listbox als Seiten an das Clientfenster der Multipage gehängt. Jeder Dialog besitzt eine eigene Callbackfunktion mit der für den Dialog individuellen Initialisierung. Innerhalb dieser Initialisierung wird dynamisch der Inhalt für die Listbox generiert.

Dies geschieht mit Hilfe der Funktion:

```
GUI_ConstString* genarray( U8 len,
                           const char* strmatr[][],
                           U32 selA[] )
```

mit: `U8 len` – Die gewünschte Sprache.
`const char* strmatr[][]` – Zeiger auf die String Matrix.
`U32 selA[]` – Bitfeld für den Aufbau.

Diese Funktion generiert dem Bitfeld entsprechend ein Feld, bestehend aus Strings für die gewünschte Seite. Dabei wird der Inhalt des Arrays aus einem mehrdimensionalen String-Feld[26] geladen, welches die Einträge in einer beliebigen Anzahl von Sprachen enthalten kann. Als erstes werden die Parameter für die Schleifen und Variablen der entsprechenden Seite gesetzt. Im nächsten Schritt wird die Anzahl der im Bitfeld aktivierten Seiten festgelegt um Speicher für das Array zu reservieren. Daraufhin wird abgefragt welcher String be-

25 Quellcode: menu_display.c.
26 Quellcode: language.c.

nötigt wird, dementsprechend Speicher reserviert und der String aus dem globalen Array an die benötigte stelle kopiert. Der Rückgabewert ist ein Zeiger auf ein String-Feld . Bild 4-13 zeigt ein Flussdiagramm um den Ablauf zu veranschaulichen.

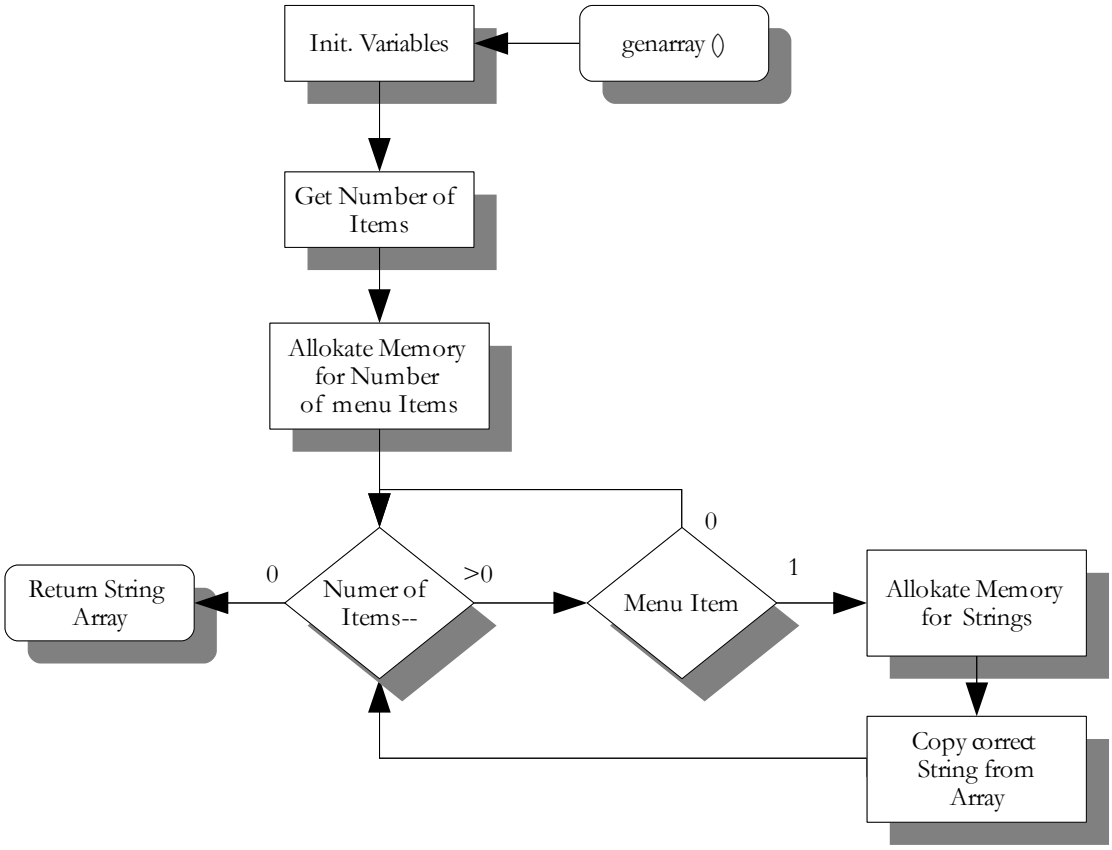


Bild 4-13 Dynamische Generierung des String-Arrays mit Hilfe der Funktion genarry().

Die Funktion *void freeStrAry(char** strAry)* gibt den Speicher wieder frei. Informationen über die Größe des Feldes, aufgrund der Programmierweise und dem Aufbau des Feldes, werden nicht benötigt.

Jede dieser *Listboxen* bekommt nun die selbe Callbackfunktion zugewiesen. Es spielt also keine Rolle von welchem Reiter aus die *Listbox* generiert wurde.

Innerhalb dieser Callbackfunktion werden die Tasten für LINKS und RECHTS auf PGUP und PGDOWN umgelegt, denn diese für das *Multipage-Widget* vorinitialisierten Tasten sind auf der SYMAP®-Compact Tastatur nicht vorhanden. Ein Drücken auf die Entertaste führt nun zu der in Bild 4-9 gezeigten Darstellung.z

Dieses Fenster, welches wiederum für alle Reiter das Selbe ist, passt sich individuell an die *Multipage* an. Über die IDs und Funktionen der Bibliothek wird abgefragt welche *Listbox* mit welchem Element das Fenster geöffnet hat. Der Inhalt des Elements wird abgefragt sowie als Titel des Fenster dargestellt. Gleichzeitig wird das vorher aktive Element ausgegraut. Die Funktion *void drawPage(U8 page, S8 select)* zeichnet nun den gewünschten Inhalt gemäß der aufrufenden Seite und dem selektierten Objekt, angepasst an das Bitfeld auf welchem die *Listbox* basiert.

Die Selektierung basiert allerdings auf dem Zusammenhang zwischen *Listbox* und der darüber aufgerufenen Dialogbox. Über die Funktion *LISTBOX_GetSel(hList)* lässt sich nur die Nummer des Elementes innerhalb der Liste feststellen. Da der Inhalt allerdings variabel ist, ist kein direkter Bezug zwischen Feld und Nummer vorhanden. Die zuvor erläuterte Funktion *genarry()* generiert die Strings immer in einer festen Reihenfolge. Dies bedeutet wiederum, fehlt ein String, haben alle darauf folgenden Strings eine kleinere Nummer als Attribut innerhalb der Liste. Somit muss nur festgestellt werden wie viele Fenster vor dem selektierten Fenster innerhalb einer Kategorie ausgeschaltet sind, um die Anzahl auf das selektierte Fenster zu addieren und die richtige Auswahl bei der Selektion zu treffen.

Das Drücken der Tasten für HOCH und RUNTER führt nun zu einem Wechsel der Seiten und gleichzeitig zu einem Wechsel des in der *Listbox* angezeigten Elementes.

MENÜ→ CONROL

Wird eines der ausgegrauten Elemente des Menüs angewählt wird daraufhin ein Dialog zur Passwortabfrage erstellt[27]. Der Dialog besteht aus einem *FRAMEWIN* mit einem *EDIT*-Feld, sowie einem *Textfeld*. Das *EDIT* Widget wird so konfiguriert, dass eine vier stellige Dezimalzahl zu sehen ist, welche sich über die Cursortasten ändern lässt. Es wurde ein Blinken der ausgewählten Zahl mit einer Zeit von 200 ms eingestellt. Bei einem Drücken der ENTER-Taste wird nun die im Fokus stehende Zahl mit einem als globale Variable deklarierten Passwort verglichen. Fällt dieser Vergleich positiv aus, werden die vorher ausgegrauten Untermenüs aktiviert. Ist der Ausgang negativ erscheint eine Meldung, dass das Passwort nicht korrekt ist. Für die gesamte Zeit der Eingabe erscheint eine Meldung zur Bedienung in der Statuszeile.

Die Auswahl eines dieser Untermenüs generiert nun ein *FRAMWIN* (s. Bild 4-11), welches seinen Inhalt dynamisch innerhalb der Callbackfunktion erstellt. Beim ersten Menüpunkt (System) werden drei *LISTVIEWs* generiert welche jeweils über zwei Spalten verfügen und jeweils eine *HEADER* Widget beinhalten. Während das Wechseln der Zeile innerhalb eines *LISTVIEW*s über dessen Eigenschaften läuft, bedarf es beim Wechsel zum nächsten LISTVIEW eines Tricks. Ziel ist es mit den Tasten für links und rechts die drei Eingabefelder zu wechseln. Die Tastatureingaben werden Bibliotheksbedingt aber nur so lange an ein übergeordnetes Fenster gesendet, bis sie benutzt werden. Da das LISTVIEW die Tasten Links und Rechts aber zum scrollen benötigt, wird bei der Tastatureingabe gleichzeitig eine selbst definierte Nachricht mitgesendet, die innerhalb der Callbackfunktion als links und rechts behandelt werden kann. Da das FRAMEWIN Widget genau drei *Kinder* hat, kann mit dem Aufruf von:

```
WM_SetFocus(WM_GetNextSibling(WM_GetFocussedWindow()));
```

der Focus auf das nächste Fenster gesetzt werden. Dabei ist der Rückgabewert der inneren Funktion der Handle des aktuell im Fokus stehenden Fensters. Der nächste Rückgabewert ist der Handle des nächsten Fensters auf gleicher Ebene, welches nun den Fokus bekommt.

Die zwei anderen Untermenüs sind auch über *Listviews* realisiert, ausgegangen von dem Vorgängermodell war hier allerdings keine Unterteilung in mehrere *Listviews* nötig.

Da die Ausgaben dieser Fenster lediglich von den Nutzereingaben abhängig sind, ist keine spezielle Aktualisierungsfunktion nötig. Die Werte verändern sich nur bei Interaktion und können somit innerhalb der Callbackfunktion abgehandelt werden.

27 Quellcode: menu_control.c.

Prozess

Die Funktion der Prozess-Taste (F3) wurde beim Symap-Compact verändert. Wurden damit vorher die Display→Prozess Seiten gewechselt, bekommt die Taste beim Compact zusätzlich die Aufgabe zwischen den benutzerspezifischen Seiten zu wechseln. Zusätzlich ist eine Standartseite vorhanden, falls der Benutzer keine spezifische Seite wünscht.

Außerdem gibt es die Möglichkeit die Seiten direkt über das Menü anzuwählen, wobei nur die im bereits erläuterten Bitfeld freigeschalteten Seiten anwählbar sind. Selbiges gilt auch für das Wechseln über die F3 Taste.

Der Aufbau einer Seite ist in diesem Fall nur von der Callbackfunktion abhängig. Jede Seite wird in dem selben Fenster gezeichnet, welches ein Client vom Clientsfenster des Hauptfensters ist. Beim erstellen eines neuen Fensters wird das Alte gelöscht, was zu einem vollständigen Löschen aller evtl. offenen Fenster führt. Es ist somit beim Aufruf einer neuen Seite nicht nötig zu wissen welche Seite vorher geöffnet wurde.

Alle Callbackfunktionen der Prozessseiten sind in einem Funktionsfeld eingetragen. Beim Aufruf einer dieser Seiten ist nun anstelle der Callbackfunktion das Array eingetragen, wobei die Stelle im Array der aufzurufenden Seite entspricht.

Die Folgende Definition zeigt das Array:

```
static void (*ptr2cbProcess[11])(WM_MESSAGE*) =
{_cbMainWin,_cbUserWin, _cbUserWin,_cbUserWin, _cbUserWin,
cbEngine, cbPower, cbLoad, cbPM, cbSync, cbBreaker};
```

Die Funktion wird als *Static* deklariert, da sie nur in dieser einen Datei bekannt sein muss. Der Typ ist ein Pointer auf eine Funktion mit dem Übergabeparameter *WM_MESSAGE**. Somit kann durch *ptr2cbProcess[page]* die gewünschte Callbackfunktion angegeben werden, dabei ist *page* die anzuzeigende Seite.

Meters, Alarms

Die Meters- und Alarms-Seiten, aufrufbar über das Menü oder der F1 und F2 Taste sind nach dem selben Prinzip aufgebaut wie die Prozess-Seiten.

IDs

Da in der gesamten Menüführung eine große Menge an Objekten erstellt wird, ist es nötig einen Überblick über die verwendeten IDs zu behalten, um ggf. Überlappungen zu vermeiden, welche zu Konflikten führen könnte. Es ist allerdings nur nötig dabei die derzeit aktuelle Seite zu beobachten, da Aufgrund der Menüstruktur keine zwei Hauptfenster gleichzeitig vorhanden sind. Es gibt jedoch die Ausnahme, dass ein paar IDs für die Statuszeile vergeben wurden, diese dürfen innerhalb einer Seite nicht nochmal verwendet werden.

4.3 Die vier variablen Seiten

Mit der geforderten Flexibilität der variablen Seiten und der Vorgabe, dass die Daten über ein externes Tool auf einen der Co-Prozessoren geladen und erst über den CAN- Bus an die GU weitergeleitet werden, sind noch weitere Anforderungen geknüpft. Die Seiten sollen über das Menü anwählbar sein und zusätzlich über die Taste F3. Die darzustellenden Elemente sollen alles beinhalten was es in der SYMAP-Serie bis jetzt gegeben hat, dazu gehören neben passiven Elementen wie Linien oder Symbolen (Generator etc.) auch aktive Elemente wie Schalter, Auswahlboxen und Messwertanzeigen. Es sollen Ladebalken, analoge Zeiger und weitere mögliche Elemente eingefügt werden und stetig ausgebaut werden. Dafür stehen vier Seiten zur Verfügung, wobei Farbe und Größe der Elemente variabel ist.

Eine Analyse der darzustellenden Elemente zeigt, dass diese in *aktive* und *passive* Elemente getrennt werden können. Unter *passiven* Elementen sind diejenigen zu verstehen die einmalig gezeichnet werden und sich nicht mehr verändern. Als Beispiel zu nennen wären Texte, Linien oder Symbole. Als *aktive* Elemente werden alle diejenigen verstanden, die sich im laufe des Betriebes verändern oder abhängig von internen oder externen Zuständen sind, wie Messwerte, Schalter, oder Auswahlboxen.

Hierbei stellen sich zwei Hauptprobleme dar. Wie werden die aktiven Elemente realisiert? Wie kann das möglichst Speicher schonend stattfinden, um den Bus und die anderen Prozessoren nicht zu belasten?

Als erstes gilt es genauer zu definieren welche Elemente es gibt und welche Eigenschaften sie haben. Es muss festgestellt werden wie groß der benötigte Speicher für ein Attribut sein könnte. Zusätzlich hat jedes Element eine Vordergrund-, sowie eine Hintergrundfarbe und ein Attribut welches die Seite angibt. Die *aktiven* Elementen benötigen zusätzlich eine Verknüpfung mit einem Parameter oder Messwert.

Wie ist nun so ein *aktives* Element zu realisieren? Die Problematik besteht darin, dass diese Elemente auf verschiedene Art und Weise gezeichnet werden. Zusätzlich sind sie abhängig von einem Parameter. Die Elemente existieren nicht als Objekte an sich, sondern müssen immer wieder anhand ihrer Attribute neu gezeichnet werden, aber auch Eingaben über die GUI an das System zurück senden. Es ist somit zwangsläufig so, dass alle Parameter eine feste Struktur benötigen, um sie später eindeutig mit einem Element verknüpfen zu können. Während der Ladebalken und die Textbox nur auf eine Art und Weise gezeichnet werden können, gibt es bei dem Schalter eine Vielzahl von Variationen. Dazu gehören z.B. die Ausrichtung sowie diverse mögliche Schalterstellungen.

Die erste Überlegung war ziemlich unflexibel. Es wurden fertige Elemente vorausgesetzt, die aufgerufen werden können. Diese speicherschonende Lösung ist allerdings wenig flexibel. Es muss jede erdenkliche Art von Schalter vorgesehen werden. Ein Schalter besteht nun nicht aus einem Element, sondern aus mehreren, wobei sich der aktive Teil lediglich auf die Wippe beschränkt. Bild 4-14 zeigt einen Schalter bestehend aus vier *passiven* Elementen, (hier Linien) und einem *aktiven,* der Wippe. Dieses Konzept wird für alle Elemente umgesetzt. Ein Messwert ist nicht ein Element, sondern besteht aus Zahl und Text als Einheit. Tab. 4.2 zeigt den Entwurf für die einzelnen Elemente.

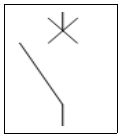


Bild 4-14 Schalter.

Tab. 4.2 Attributgrößen der Elemente. - Die Elemente beinhalten ggf. eine Vordergrund sowie Hintergrundfarbe und benötigen dafür 10 Bit Speicher. Für die Zuordnung des Bildes werden 2 Bit benötigt, sowie 4 Bit, um das Element zu identifizieren.

Element	Koordinaten[a)] (Bit)	Attribut 1 (Bit)	Attribut 2 (Bit)	Attribut 3 (Bit)
Linie	X1, Y1, X2, Y2 (2 * (10 +9))	Strichstärke (3)		
Rechteck	X1, Y1, X2, Y2 (2 * (10 +9))	Strichstärke (3)	Ausgefüllt (1)	
Dreieck	X1, Y1, X2, Y2, Rotation (2 * (10 +9) + 2)	Strichstärke (3)	Ausgefüllt (1)	
Kreis	X1, Y1, Radius (10 +9 + 10)	Strichstärke (3)	Typ[b)] (4)	
Text	X1, Y1 (10 + 9)	Schriftart (3)	Text (18)	
Auswahlbox[c)]	X1, Y1 (10 + 9)	Schriftart (3)	Typ[d)] (2)	Zustand (1)
Schalter[c)]	X1, Y1, X2, Y2 (2 * (10 +9))	Strichstärke (3)	Blinkt (1)	Zustand (1)
Ladebalken[c)]	X1, Y1 (10 + 9)	Größe (10 + 9)	Rahmentyp (2)	
Phasenwinkelanzeige[c)]	X1, Y1 (10 + 9)	Breite, Höhe (7 + 6)		
Analogmesswert[c)]	X1, Y1 (10 + 9)	Breite, Höhe (10 + 9)	Typ (2)	
Messwert[c)]	X1, Y1 (10 + 9)	Kommawert (2)	Stellen (3)	Zeit (1)

a) Die Koordinaten ergeben sich aus der Anzahl der Pixel für ein VGA Vollbild.
b) Es besteht die Möglichkeit eines Halbkreises.
c) Alle aktiven Elemente benötigen noch ein Attribut Parameter, welches die Verknüpfung zu einem Messwert o.ä. beinhaltet. Dies wurde auf 16 Bit festgelegt.
d) Es sind zwei verschiedene Auswahlboxen möglich.

Hierbei handelt es sich um einen ersten Entwurf um den Speicherverbrauch abzuschätzen. Ob alle Attribute benötigt werden oder weiter hinzukommen zeigt sich im laufe des Projektes. Außerdem werden noch weitere Elemente hinzukommen.

Die Anzahl der benötigten Elemente wurde großzügig auf 200 geschätzt. Da es sich dabei um eine Vielzahl von unterschiedlichen Elementen Handeln kann erscheint es sinnvoll diese in einem festen Format in dem Speicher abzulegen. Diese Vorgehensweise würde eine aufwendige Speicherverwaltung vermeiden.

4.3.1 Elementstruktur

Um die Verwaltung der Elemente möglichst einfach zu halten, bietet es sich an dafür feste Strukturen anzulegen. Ein Element beinhaltet neben den Koordinaten, Attribute für Farben, Strichstärke, ggf. zu koppelnde Ereignisse und Seitennummer. Eine Struktur für die Elemente könnten aussehen wie in Bild 4-15, wobei nun das Attribut ID hinzukommt welches noch genauer erläutert werden wird.

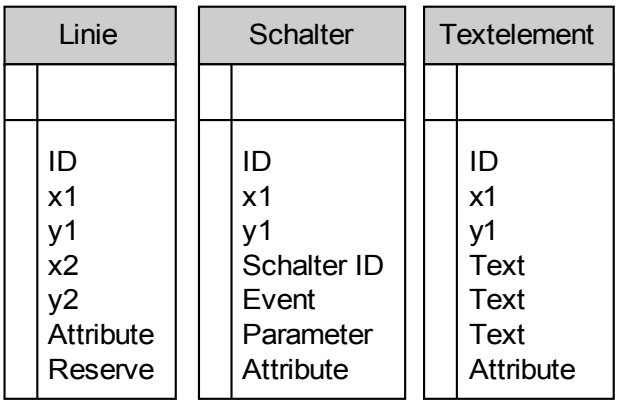


Bild 4-15 Elementstruktur ohne Anpassung. - Hierbei sollen Farbinformationen, Strichstärke etc. in dem Element Attribute untergebracht werden.

Geht man nun von einer Größe von 2 Byte pro Feld aus, ergeben sich bei 200 Elementen 200*16 Byte = 3,125 kByte. Da eine zeitaufwändige Speicherverwaltung erspart werden soll, müssen für die Elemente feste Speicherbereiche festgelegt werden. Da nicht ersichtlich ist, wie viele Elemente eines Typs existieren, müssten für jeden Elementtyp 200 Elemente angelegt werden, was zu ungenutzten Speicher führt.

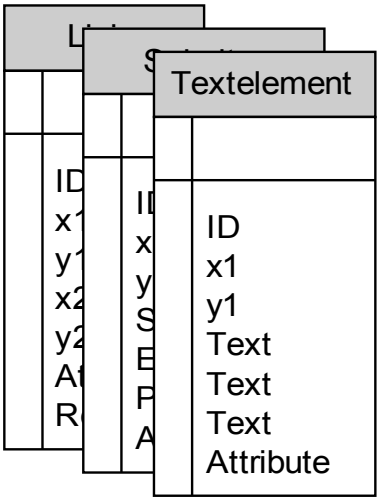
Die Lösung des Problems ist eine *UNION*. Somit belegen alle Elemente den selben Speicherbereich, nur das durch die Strukturen unterschiedlich auf sie zugegriffen werden kann.

Da die Elemente jetzt den selben Speicherbereich belegen ist nun eine Typidentifikation notwendig die darauf schließen lässt um welches Element es sich handelt. Im Verlauf der Arbeit ist noch eine Elementnummer dazu gekommen um ggf. auch ein Element eindeutig identifizieren zu können.

Wird nun zu einem späteren Zeitpunkt beschlossen den Speicher dynamisch zu verwalten ist hierfür lediglich die Gesamtzahl der genutzten Parameter notwendig.

Aufgrund der gewählten Struktureinteilung bleiben viele Bits der Struktur allerdings ungenutzt, was den Speicherverbrauch einer Struktur unnötig vergrößert. Eine Verschiebung und Größenanpassung innerhalb der Struktur ist somit notwendig. Bild 4-16 zeigt die effektive Nutzung des Speichers innerhalb einer Struktur. Ein Element der Struktur benötigt nur noch 12 Byte Speicher. Somit konnte der Speicherverbrauch von 16 Byte auf 12 Byte bei 200 Elementen somit von 3,125 kByte auf 2,4 kByte reduziert werden.

a)



b)

```
typedef struct
{
   union
   {
        struct
      {
        ...
      } linie;
      struct
      {
        ...
      } schalter;
      struct
      {
         ...
      } textelement;
   } elemente;
} ELEMENTSTRUKTUR;
```

Bild 4-16 Optimierte Elementstruktur a) Schaubild, b) Beispielcode.

Diese Form der Formatierung führt zu weiteren Vorteilen in der Handhabung. Bei der Codegenerierung über das *Graphic Builder Tool* kann jeder Parameter in der selben Form als Code erstellt werden. Zu jedem Parameter gehören drei U32 Variablen, die über die in Bild 4-18 ersichtliche Struktur eingefügt werden können, welche ebenfalls zur UNION gehört.

Das folgende Beispiel soll den Aufbau auf Bit-Ebene anhand eines Messwertelementes noch einmal veranschaulichen. Die gesamte Struktur ist in Anhang A2: Elementstruktur einsehbar.

Messwert Element:

```
1. U32    31xxxx xxxx  xxxx xxxx  xxxx xxxx  xxxx xxxx0

    Bit 29-31: Strukturidentifikation
    Bit 27-28: Bildnummer
    Bit 19-26: ID
    Bit  9-18: X1-Koordinate
    Bit  0- 8: Y1-Koordinate
2. U32   31xxxx xxxx  xxxx xxxx  xxxx xxxx  xxxx xxxx0

    Bit 27-31: Vordergrundfarbe
    Bit 22-26: Hintergrundfarbe
    Bit  8-21: Reserve
    Bit   0-7: Schriftart
3. U32   31xxxx xxxx  xxxx xxxx  xxxx xxxx  xxxx xxxx0

    Bit 16-31: Parameter
    Bit  0-15: Elementnummer
```

```
typedef struct
{
   union
   {
        struct                /* Switch                */
      {
         U32 y1        : 9;  /* y1-koord              */
         U32 x1        :10;  /* x1-koord              */
         U32 id        : 4;  /* GrafikID              */
         U32 state     : 2;  /* Every switch gets 1 drawing for each
                                position the first gets a 1 for
                                state, the 2nd a 0     */
         U32 blink     : 1;  /* Element blinking      */
         U32 screen    : 2;  /* Screen Number         */
         U32 type      : 4;  /* Typidentification :1  */


         U32 y2        : 9;  /* y2-koord              */
         U32 x2        :10;  /* x2-koord              */
         U32 strength  : 3;  /* Pen size              */
         U32 bgcolor   : 5;  /* Backgroundcolor       */
         U32 fgcolor   : 5;  /* Foregroundcolor       */

         U32 param     :16;  /* Link to Parameter     */
         U32 num       :16;  /* Elementnumber         */
      } e_switch;
```

Bild 4-17 Strukturelement eines Schalters.

Über das folgende Makro ist ein vereinfachter Zugriff möglich:

```
#define e1(i)      (*(param+i)).elemente.insert
```

Dies erleichtert an dieser Stelle das Protokoll für den Bus, da nicht auf das Datenformat geachtet werden muss.

```
struct
      {
         U32 dword1;         /* To insert         */
         U32 dword2;         /* into structure    */
         U32 dword3;         /* 3x4 Byte          */
      } insert;
```

Bild 4-18 Strukturelement zum Einsetzen der Daten.

4.3.2 Realisierung

Insbesondere bei der Darstellung muss nun beachtet werden, dass verschieden Typen von Elementen existieren die zum einen durch die Eigenschaft der Elemente zum anderen durch die Eigenschaften der Grafikbibliothek sehr unterschiedlich gehandhabt werden müssen oder können. An dieser Stelle wird nun auf die Realisierung mit den vorhandenen Mittel eingegangen.

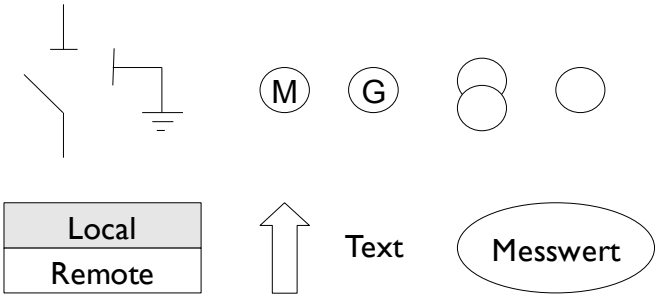


Bild 4-19 Elementauswahl.

Bei den in Bild 4-19 dargestellten Elementen handelt es sich um eine beispielhafte Auswahl, die sich am Vorgängermodell orientiert. Es ist eine beliebige Anzahl an Variationen möglich, lediglich bei den aktiven Elementen wird es Vorgaben geben.

Aufgrund der durch die Grafikbibliothek gegebenen Mittel hat es sich als effektiv erwiesen die Elemente auf zwei unterschiedliche Art und Weisen zu behandeln. Es ist nötig gewesen die Elemente zu trennen. Es müssen 2D Zeichenfunktionen oder einfache Ausgaben und Elemente die über die *emWIN Widgets* dargestellt werden.

Aufbau der Seiten

Die ersten Version basierte darauf, dass für jedes der vier variablen Seiten eine eigene Funktion erstellt wurde, die ein eigenes Fenster mit einer eigenen ID erstellt hat. Die erstellten Fenster riefen die selbe Callbackfunktion auf, wobei über die ID festgestellt werden konnte welche der Seiten gerade aktuell ist und sie dementsprechend aufbauen. Der Seitenwechsel konnte über ein Feld der Funktionen an jeder Stelle des Programms stattfinden. Dies basierte auf der Annahme, dass für jede dieser Seiten ein eigenes Fenster erstellt werden muss um später in der Callbackfunktion differenzieren zu können welches der Fenster nun angezeigt wird und dass die Prozessseiten ihren eigenen Bereich bekommen.

Eine Vertiefung mit der Thematik hat jedoch ergeben, dass die Seiten genau wie in Kapitel 4.2.2 S. 59 beschrieben aufgebaut werden können, denn die ID ist nicht speziell an ein Fenster geknüpft. Die Änderung gegenüber den den restlichen Seiten besteht nun darin, dass die variablen Seiten die selbe Callbackfunktion nutzen, jedoch das Fenster je nach aufzurufender Seiten eine eigene ID zugewiesen bekommt über die, die Seite unterschiedlich aufgebaut werden kann.

Der Aufbau der Seiten basiert nun auf zwei Funktionen die im folgenden erläutert werden. Es wird davon ausgegangen, das alle zu zeichnenden Elemente über CAN angekommen sind und in einem Feld abgelegt wurden. Hierbei spielt es keine Rolle ob das Feld dynamisch erzeugt worden ist oder nicht.

2D Elemente

Die 2D Elemente wie Linien, Kreise, Rechtecke oder auch konstanter Text lassen sich sehr einfach über die in Kap. 2.7.3 beschriebenen *Callbackroutinen* darstellen. Hierbei wird die Nachricht *WM_PAINT* genutzt, die an die zum jeweiligen Fenster gehörende Routine gesendet wird, wenn das Bild neu gezeichnet werden muss. Es reicht somit *eine* Callbackfunktion für alle vier Benutzer spezifischen Seiten aus, da immer nur eine dieser Seiten existiert und das eigentlich zeichnen in der folgenden Funktion stattfindet:

```
1. void createScreen( U32 screen )
```

wobei der Übergabeparameter *screen* das jeweils darzustellende Fenster ist. Dieses lässt sich ermitteln, indem nach der Erstellung des Fensters mit Hilfe von Funktion:

```
2. WM_SetId(hUserWin,GUI_ID_USER +  n)
```

die IDs (1 – 4) zugewiesen werden. Nach dieser Zuweisung wird die ID über die Nachricht WM_SET_ID aus der Nachrichtenstruktur *pMsg->Data.v* einer statischen Variablen zugewiesen die, die ID für den nächsten Aufruf der Callbackfunktion speichert. Der Übergabeparameter *hUserWin* ist der Handle des erstellten Fensters.

Die Funktion *createscreen()* arbeitet nun die Elementstruktur ab und führt die nötigen Zeichenoperationen gemäß der in der Struktur stehenden Eigenschaften aus. Das diese Funktion zyklisch alle 250 ms aufgerufen werden muss, wurde darauf geachtet, dass alle 2D Elemente am Anfang der Struktur sind. Somit kann die genutzte Schleife beim einmaligen Erreichen eines anderen Elementes abgebrochen werden. Bild 4-20 zeigt den generellen Ablauf der Zeichenfunktion. Jedes Element wird abgefragt, ob es zu dem zu zeichnenden Bildschirm gehört und dann gemäß seiner Eigenschaften gezeichnet. Dies geschieht für diese Art von Elementen zyklisch mit dem Aufruf der Aktualisierungsfunktion.

Da ein Schalter in der Bibliothek nicht vorgesehen ist, ist es am einfachsten diesen als zwei 2D Linien zu verstehen. Um ein Maximum an Flexibilität zu gewährleisten werden keine festen Speicherelemente genutzt, lediglich die Wippe ist ein aktives Element. Dies erhöht zwar die Anzahl der Elemente, jedoch hat der Kunde dadurch mehr Möglichkeiten den Schalter zu gestalten. Nun bekommt ein Schalter zwei Zeichnungen für zwei Schalterstellungen, wobei die erste über ein Flag die Kennzeichnung als Standartzeichnung bekommt. Es wird nun beim Zeichnen immer auf dieses Flag abgefragt um das Erste der beiden Elemente herauszufischen. Nun wird, je nach Parameter, die eine oder die andere Schalterstellung gezeichnet.

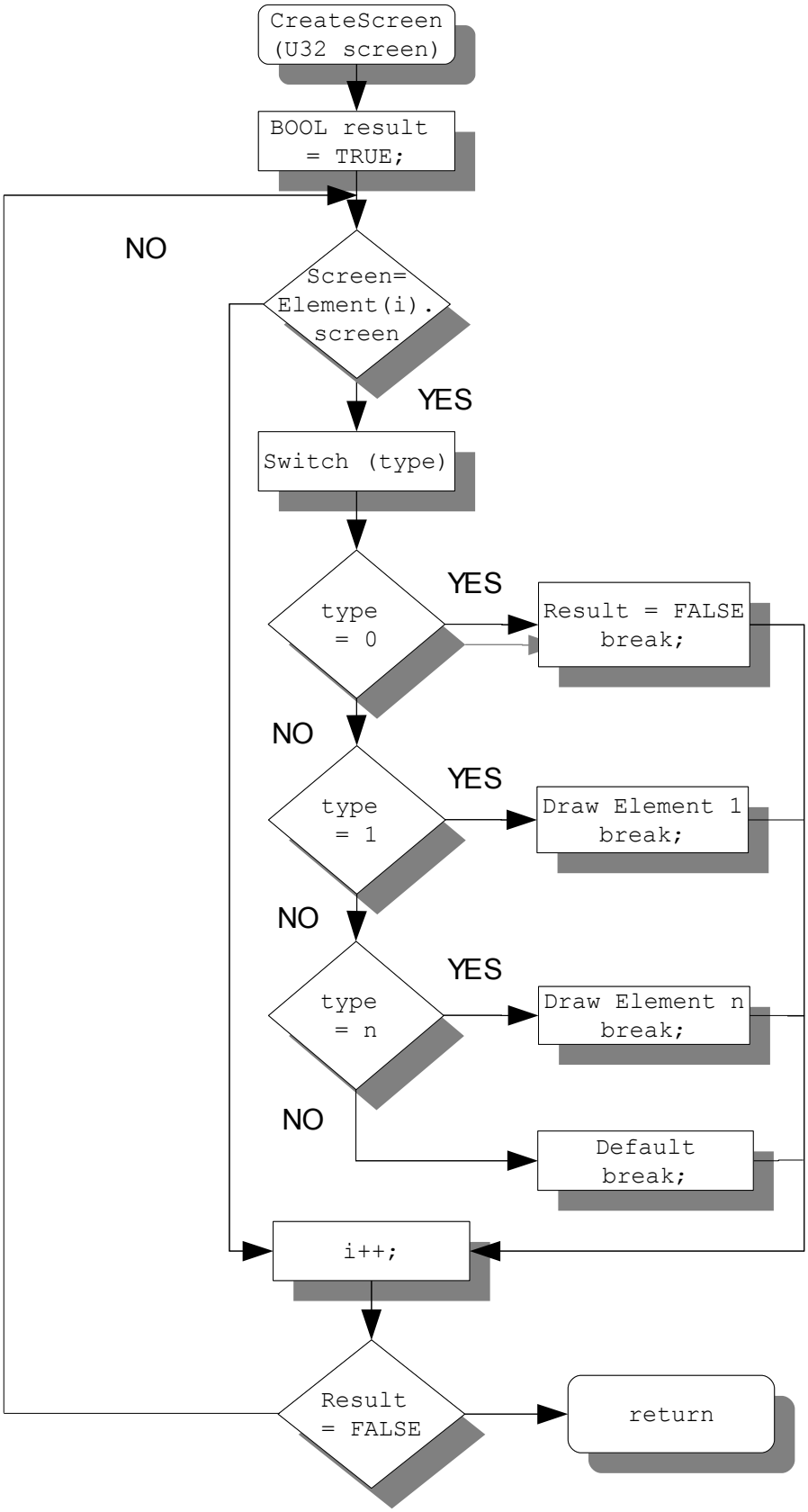


Bild 4-20 Flussdiagramm der Funktion createscreen().

Widget Elemente

Die aufgrund der gegebenen Mittel ökonomischste Lösung ist Messwerte, Auswahlboxen, oder Ladebalken einfach über die Widgets zu realisieren. Diese sollten initialisiert werden, damit eine spätere Pflege der Inhalte entfällt, indem der darzustellende Inhalt einfach über einen dereferenzierten Zeiger ausgelesen wird. Erste Versuche haben allerdings erwiesen das eine Kopplung der Widgets an einen bestimmten Wert nicht möglich ist, somit ist eine Verwaltungsfunktion nötig.

Die Nachricht *WM_CREATE* wird einmalig beim Erstellen eines Fensters aufgerufen und kann somit für die Initialisierung der Elemente genutzt werden, die mit Hilfe der Widgets dargestellt werden sollen.

An dieser Stelle wird die Funktion:

```
void createScreenWidget(U32     screen,
                        WM_HWIN hdlScreen)
```

aufgerufen. Diese enthält als Parameter den Bildschirm auf den die Elemente gezeichnet werden sollen, sowie den *Handle* für das Fenster an das, das Widget gehängt wird. Dieser steht in der Struktur *pMsg→src,* welche jeder Callbackfunktion übermittelt wird. Die Funktion *createScreenWidget()* hat schematisch den gleichen Aufbau wie Bild 4-20 mit dem Unterschied, dass die Elemente nicht direkt gezeichnet werden, sondern über Funktionen initialisiert werden.

Um eine spätere Aktualisierung möglich zu machen ist eine eindeutige Identifizierung des *Widegts* nötig. Dies geschieht indem beim Erstellen der *Widgets* ein Zähler hochgezählt wird der gleichzeitig der ID des *Widgets* entspricht. Da jeder Typ *Widget* eine eigene Funktion besitzt ist auch für jeden Typ *Widget* ein eigener Zähler nötig. Eine Aktualisierungsfunktion bezieht nun nacheinander dynamisch den *Handle* für das Widget und schickt den aktualisierten Wert ab. Es wird zuerst abgefragt, ob ein Typ *Widgets* existiert und nacheinander über alle IDs dieses *Widgets* die Werte aktualisiert.

Nun erscheint es als sinnvoll zwei Listen mit zwei Arten von Elementen zu erstellen. Die die einmalig aufgerufen werden und die, die immer wieder gezeichnet werden. Diese Datenverwaltung entfällt jedoch wenn alle *passiven* Elemente an den Anfang der Struktur geschrieben werden, und alle *Widget* Elemente an das Ende. Somit kann die *createscreen()* Funktion nach erreichen des ersten *Widgets* die Schleife abbrechen und es entsteht kein Performanceverlust. Bei dem einmaligen Initialisieren der Widgets ist dieses weniger von Bedeutung.

4.4 Aktualisierung

Für die Anzeige ist es ausreichend die Werte alle 250 ms zu aktualisieren. Es ist somit unnötig abzufragen, ob eine Veränderung bei einem Wert stattgefunden hat. Alle darzustellenden Messwerte liegen in einem Feld vor aus dem sie jederzeit ausgelesen werden können.

Da immer nur eine Seite zur Zeit dargestellt wird macht es wenig Sinn ständig alle Werte aktuell zu halten. Es reicht somit nur die Werte zu aktualisieren die sich gerade auf der Anzeige befinden. Das bestehende System besteht aus Fenstern die wie Objekte zu sehen sind, die eine Abhängigkeit voneinander haben. Es ist keine Struktur vorgesehen die Abspeichert welche Seite aktuell aufgerufen wird. Die Einführung einer solchen Struktur hätte zur Folge das bei jeder Aktualisierung eine große Menge an Abfragen getätigt werden müssen, welches Fenster nun aktualisiert werden soll. Eine alternativ angedachte Idee war es alle *aktiven* Elemente wie Objekte zu behandeln und deren Adressen in einer Liste zu speichern. Letztendliche hat sich jedoch eine andere Lösung als die effektivste erwiesen.

Zu jedem Fenster existiert eine Funktion um dieses zu aktualisieren. Diese Vorgehensweise ist nötig, da jedes Fenster anders aufgebaut ist und andere Messwerte oder Parameter bezieht. Es existiert nun ein globales Feld, welches die Zeiger auf die Funktionen zur Initialisierung enthält:

```
void(*updateF[N])(void)={Funktion1,
                         Funktion2,
                         ...,
                         FunktionN};
```

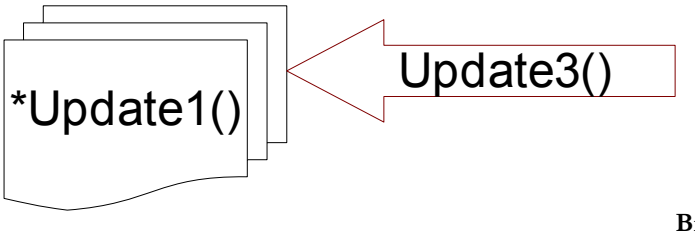


Bild 4-21 Zugriff über Funktionszeiger.

Nun existiert ein Pointer vom Typ der Aktualisierungsfunktion:

```
void (*updateFakt) (void);
```

Dieser Pointer zeigt jeweils auf die aktuelle Aktualisierungsfunktion und wird im Hauptprogram zyklisch mit

```
updateFakt();
```

aufgerufen (s. Bild 4-21). Wenn ein neues Fenster geöffnet bzw. erstellt wird, wird einmalig beim Erstellen des Fensters der entsprechende Zeiger aus dem Feld auf den Funktionszeiger vom selben Typ kopiert, welcher zyklisch die Funktion im Hauptprogramm aufruft. Es ist somit nicht nötig zu wissen welches Fenster geöffnet ist, da immer die richtige Funktion ausführt wird.

Menü

Das Aktualisieren der dargestellten Werte geschieht über zusätzliche Funktionen. Da die darzustellenden Werte alle hintereinander im Speicher abgelegt sind und auch die IDs der Elemente alle durchnummeriert wurden, ist es möglich diese teilweise über *for*-Schleifen zu aktualisieren. Ob dies möglich ist oder nicht, hängt davon ab ob die Ausgaben eine identische Formatierung haben.

Ein Beispiel wäre eine Seite mit 10 Messwerten, davon haben vier insgesamt fünf Stellen, wovon zwei Nachkommastellen sind. Die restlichen sechs sind ganzzahlig. Somit können in zwei Schleifen alle Messwerte aktualisiert werden. Die folgenden Codezeilen veranschaulichen die Funktionsweise anhand eines Wertes:

```
1. hClient = WM_GetClientWindow(hMainWin);
2. hText = WM_GetDialogItem(hClient, IDFREQ + i);
3. sprintf(buff,"%2.2f", meter[i]);
4. TEXT_SetText(hText, buff);
```

Im ersten Schritt wird hierbei der Handle für das aktuelle Fenster abgespeichert, in Schritt zwei der Handle für das zu aktualisierende Textfeld. Funktion drei schreibt nun formatiert in einen temporären String, welcher bei Funktion vier in das Textfeld geschrieben wird. Sind nun von der Formatierung her identische Werte in der Variablen *meters [n]* abgelegt und auch die IDs der Textfelder mit identischen Anzeigewerten hintereinander, können alle Felder vom selben Typ über eine Zählvariable *i* aktualisiert werden.[28]

Variable Seiten

Da die vier variablen Seiten einen nicht bekannten Aufbau haben und es Elemente wie den Schalter gibt, die eigens gezeichnet wurden, ist eine differenzierte Aktualisierung nötig. Während die *Windgets* sich selbst aktualisieren, wenn sie einen neuen Wert bekommen, werden die über die Nachricht *WM_PAINT* verwalteten Elemente nur dann neu gezeichnet, wenn das Fenster überdeckt wurde.

Die über *WM_PAINT* gezeichneten Elemente werden aktualisiert indem dem Fenster bei einer Veränderung die Nachricht *INVALID* gesendet wird, welche zu einer Neuzeichnung führt. Da dies nicht allzu oft geschieht und sehr schnell von statten geht entsteht hierbei auch kein Flimmern oder Flackern des Bildes.

Im vorigen Kapitel wurde bereits erläutert wie *Widgetelemente* für die variablen Seiten erstellt und eingebunden werden. Die Aktualisierungsfunktion dieser Elemente macht sich zu nutze, dass jeder Typ *Widget* einen eigenen ID Bereich besitzt und eine Zählvariable die, die Anzahl der Elemente dieses Typs beinhaltet. Mit Zählvariable und *Offsett* des *Widgettyps* kann somit eindeutig die ID eines *Widgets* bestimmt werden. Ist die Zählvariable größer als Null bedeutet es, dass Elemente dieses Typs existieren und es kann über die Anzahl der Elemente aktualisiert werden. Dies geschieht bei jedem *Widgettyp* anders und wird der Übersichtlichkeit halber hier aber nicht aufgeführt.[29]

28 Quellcode: meters.c, alarms.c, process.c

29 Quellcode: menu.c

4.5 Graphic Builder Tool

Das Programm zur Erstellung der grafischen Oberfläche, welche das Gerät anzeigen soll, heißt *Graphic Builder Tool* (s. Bild 4-22). Es wird die Möglichkeit bieten einfache grafische Formen und Texte auf einem virtuellen Display zu platzieren. Dabei wird Größe und Farbe des Symbols bestimmt, sowie dieses mit einem bestimmten Parameter verknüpft. Ein Beispiel wäre ein Schalter der mit einem Ereignis verknüpft ist, welches das Schließen des Schalters zur Folge hat.

Zusätzlich wird es eine Grafikbibliothek geben, bei der komplexere Symbole als Gruppe zusammengefasst und abgespeichert werden können. Diese können dann nach Bedarf in das aktuelle Bild hineingezogen und bearbeitet werden. Sämtliche Projektdateien und Bibliothekseinträge werden im XML-Format zur besseren Portierbarkeit abgespeichert.

Das *Grafic Builder Tool* ist nicht Bestandteil dieser Arbeit gewesen.

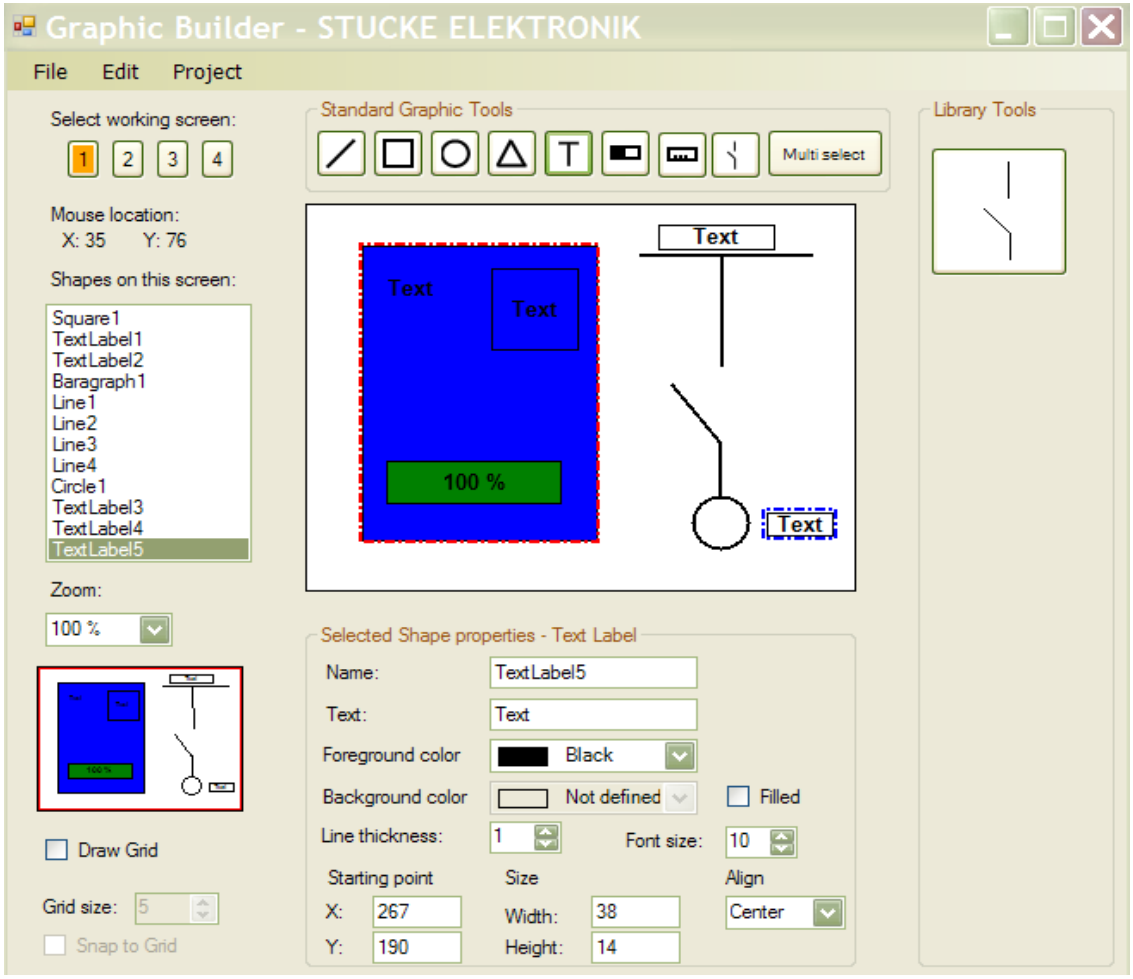


Bild 4-22 Grafic Builder Tool.

Parametersatz Beispiel

Das folgende Beispiel (Bild 4-23) zeigt, wie ein Bild mit dem dazugehörigen Parametersatz, aufgebaut werden könnte:

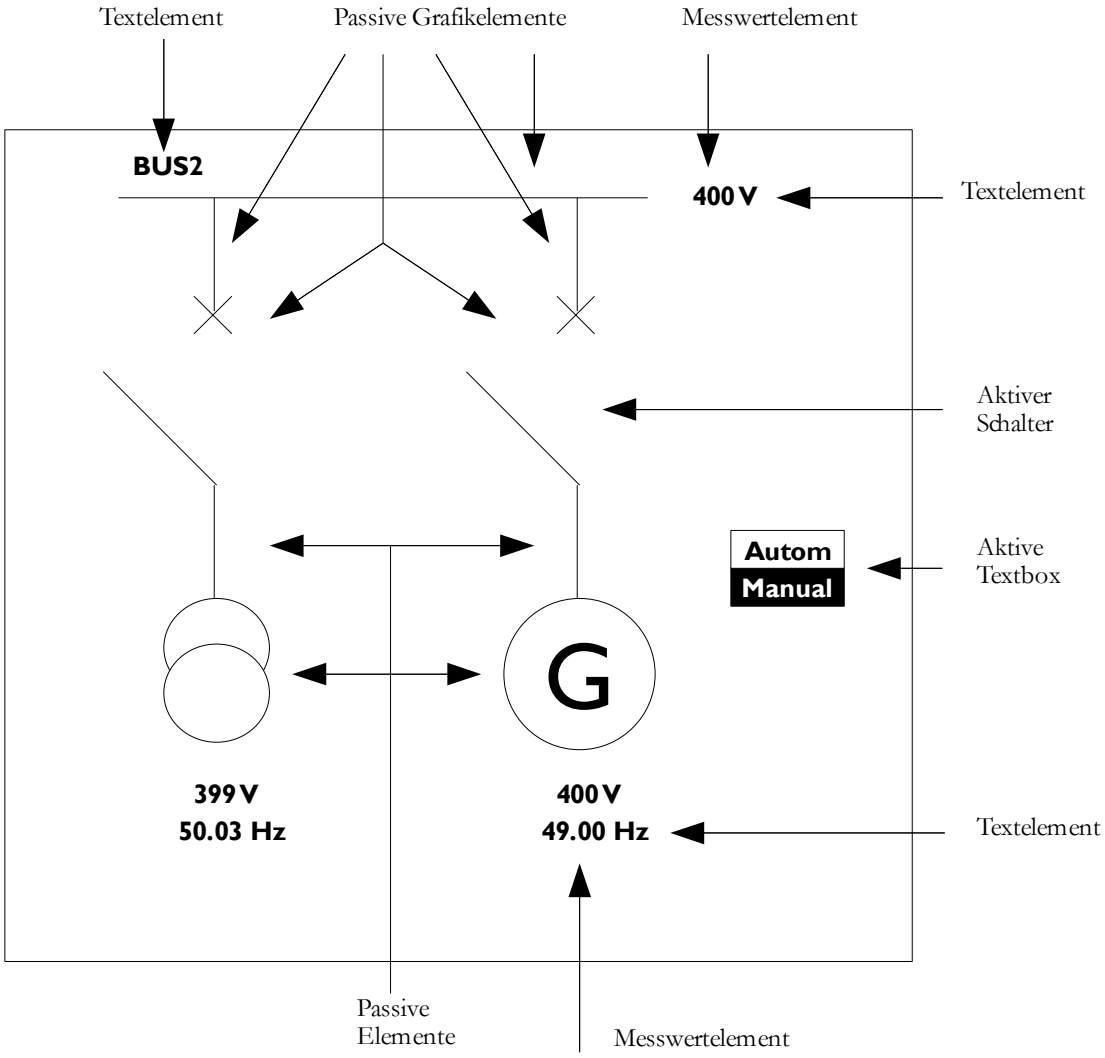


Bild 4-23 Beispiel für ein Anwendungsbild.

Der dabei generierte Code könnte folgendermaßen aussehen:

0000 0014 00B1 0470 0000 0000 (Textfeld mit den Zeichen BUS2 oben-links)
0000 0010 0018 008F 0018 0000 (Langer waagerechter Strich oben)
0000 0030 0018 0030 004A 0000 (kurzer senkrechter Strich oben-links)
0000 0070 0018 0070 004A 0000 (kurzer senkrechter Strich oben-rechts)
0000 0030 009A 0030 00A4 0000 (kurzer senkrechter Strich mitte-links)
0000 0070 009A 0070 00A4 0000 (kurzer senkrechter Strich mitte-rechts)
...
Verbrauch: ca. 348 Byte

5 Bus Protokoll

Bei den Vorgängermodellen des SYMAP®-Compact befanden sich die Prozessoren für Grafik und Kontrolleinheit auf der selben Platine.

Da nun die Kommunikation der GU mit dem Rest des Systems lediglich über den CAN Bus stattfindet, ist es nötig ein Protokoll zu entwickeln, welches die bis zu 10000 möglichen Parameter verwalten kann, ohne wichtige Kommunikation die Messungen bzw. den Schutz betrifft, zu blockieren.

Mit dem *Standard* Protokoll stehen 2048 verschiedene Identifier zur Verfügung. Jedoch ist es möglich mit jedem *Frame* bis zu 4 Byte an Daten zu verschicken. Kombiniert mit der Überlegung die anzuzeigenden Messwerte in Prozenten zu übertragen, haben diese nun eine feste Größe von 2 Byte. Somit bleiben die restlichen 2 Byte für ein Unterprotokoll zur Verfügung, welches pro Frame theoretisch 65533 Möglichkeiten bietet die Daten zu unterscheiden. Der Vorteil gegenüber dem 29 Bit Identifier besteht darin, dass der *Overhead* Pro Frame sich nicht vergrößert.

Da ein Teil der Messwerte für die Schutzfunktionen benötigt wird, macht es durchaus Sinn diese zyklisch über den Bus zu senden. Hierfür wurden 100 ms angesetzt. Für die Anzeige wären 250 ms ausreichend, ein doppeltes Senden erscheint allerdings unnötig. Die Messwerte werden direkt mit Bezug auf den Nominalwert umgerechnet und in einem float Feld abgespeichert. Die Werte habe in der Regel eine Nachkommastelle.

Auf den Großteil der Parameter wird jedoch bei Einstellungsänderungen am Gerät zugegriffen, was nach einer Installation eher selten der Fall sein wird. Diese werden somit nur auf Anforderung über den Bus geschickt. In der Regel handelt es sich hier eher um kleinere Datenmengen. Die Tabelle 5.1 zeigt einen Entwurf des Bus Protokolls, welcher in Zusammenarbeit mit den Entwicklern für die MU und CU entstanden ist.

Tab. 5.1 CAN ID Range.

CAN ID Range	Name	Amount	Bytes	CAN Objects	Cycle (ms)	Unit	Sender	Receiver
0- 20	Firmware Update	?	?	20			CU	MU, GU
100- 101	MU Event data	128 Events	16	2	Event	Bits	MU	CU, GU
110- 199	CU Event data	5000 Events	625	80	Event	Bits	CU	MU, GU
1000- 1009	MU Measure data FAST	8 x U16	16	2	100	%	MU	CU,GU
1010- 1029	MU Measure data SLOW	40 x U16	80	10	250	%	MU	CU, GU
1030- 1049	MU Calibration Data			10			MU	GU
1050- 1079	MU DEBUG Data			20			MU	GU
1100- 1199	CU Process data	8 x U16	16	100			CU	MU, GU
1200- 1299	GU User Interface data			100				
1500- 1509	Parameter (System start)			10			CU	MU. GU
1510- 1519	Parameter (Communication)			10			CU	MU, GU
1520- 1529	Parameter (GU Editor)			10			GU	CU, MU
1530- 1539	Parameter(MU Calibration)			10			MU	CU, GU

6 Speicher

Dem System steht neben dem Mikrocontroller internen Speicher, zusätzlich der externe RAM zur Verfügung. Dieser wurde mit der Startadresse 0xA0025800 und einer Größe von 153600 Byte (0x1FDA800) direkt an das System angebunden (µVision3 → Options for Target → Target). Der Speicherbereich von der Adresse 0xA0000000 bis zur Adresse 0xA0025800 wurde für den LCDC als *Frame buffer* freigehalten, befindet sich somit im Anfangsbereich des externen Speichers. Tab. 6.1 zeigt die Adressbereiche der GU.

Tab. 6.1 Speicherbereichsadressierung GU.

Adressbereich	Speicherbereich	Beschreibung
0x0000 0000 - 0x0007 FFFF	512 kB	Flash Memory
0x4000 0000 - 0x4000 FFFF	64 kB	RAM
0xA000 0000 - 0xA200 0000	32 MB	Ex- RAM
0x8000 0000 - 0x8010 0000	1 MB	Ex- Flash

Eine erste Abschätzung für den Speicherverbrauch des Codes ergab c.a. 150 kB. Die Schätzungen für die GUI wurden dem *emWin* Handbuch entnommen. Das Projekt benötigt im aktuellen Zustand bereits 154916 Byte. Davon sind 38060 RO Daten und 1496 RW Daten (Compiler). Der Speicherbereich für den LCDC ist allerdings nicht mit eingebunden und wird gesondert verwaltet. Die Abweichungen zu dem gegebenen Code lassen sich damit erklären, dass die von der Firma SEGGER angegebenen Abschätzungen sich lediglich auf den Kern des einzelnen Elementes beziehen, nicht aber auf die genutzten Funktionen. Auch sind die eigens geschriebenen Routinen nicht in der Berechnung enthalten. Wird von dem jetzigen Zustand unter Beobachtung der Softwareentwicklung ausgegangen, wird der Code vermutlich eine Größe von 250 – 300 kB erreichen was unter anderem mit einer sehr großen Menge an Strings zusammenhängt.

Die Entwicklungsumgebung bietet jedoch die Möglichkeit den Prozessor im *Thumb-Mode* (Kap. 2.1) laufen zu lassen, sowie eine von der Firma KEIL gegebene MicroLib anzuwenden.

Mit dem Optimierungslevel 1 konnte der Code auf weiterhin 60% reduziert werden, jedoch hat teilweise unvorhersehbares Verhalten dazu geführt diesen Optimierungslevel wieder ausschalten zu müssen. Da insgesamt 500 kB an Flash-Speicher zur Verfügung stehen kann evtl. auf jegliche Optimierung verzichtet werden. Viele Daten die über den Bus kommen müssen nicht im Flash abgelegt werden, sondern können im RAM gespeichert werden. Es wurde bei der Programmierung darauf geachtet, dass eine spätere Optimierung immer noch möglich ist.

7 Zusammenfassung und Ausblick

Mit der Bedienungsanleitung des Controllers und anhand einiger Beispiele von NXP war ein schneller Einstieg in die Thematik möglich. Insbesondere das Schreiben des Displaytreibers hat Einblicke in die Funktionsweise von LCD und LCD-Controller gegeben. Der Kontakt zu den Distributoren, für die Beschaffung und Selektion der Displays, hat diese Kenntnisse noch vertieft.

Nach Erarbeitung der Grundlagen war der Aufbau einer Testumgebung nötig, um parallel zur Hardwareentwicklung an der Software arbeiten zu können. Es wurde die Peripherie angebunden und getestet. Dazu gehören Tastaturtreiber, LCD-Treiber sowie CAN Treiber.

Die Vorgabe, dass vier Seiten komplett extern generierbar sein sollten, benötigte eine Reihe an Vorüberlegungen, denn Speicherverbrauch und Flexibilität stehen im Vordergrund. Es hat sich als effektiv heraus gestellt die gesamte Struktur im vornherein zu entwerfen, damit parallel am *Graphic Builder Tool* gearbeitet werden konnte. Es können damit bereits alle *passiven* Elemente wie Linien, Kreise und Text in jeglicher Form und Farbe dargestellt werden. Das Darstellen *aktiver* Elemente wie Ladebalken, Schalter und Auswahlbox ist zusätzlich möglich. Die Menüstruktur ist flexibel und kann jederzeit erweitert werden. Vorlagen für alle darstellenden Seiten sind mit Aktualisierungsfunktion implementiert.

Bei der Parametrierung der Strukturen wurde dafür gesorgt, dass ausreichend Platz für Erweiterungen vorhanden ist. Ein Umsteigen auf ein VGA Display ist möglich. Alle Elemente können mit Parametern verknüpft werden.

Probleme haben sich in verschiedenen Bereichen ergeben. Das externe Display nach Vorgabe des in das Evaluationboard integrierten Displays anzusteuern, hat sich als nicht möglich erwiesen. Es hat sich herausgestellt, dass das interne Display mit dem RGB 5:5:5:1 Format angesteuert wird und nicht mit dem gewünschten RGB 5:6:5 Format. Das Ergebnis daraus war, dass die Pinbelegung des Adapters nicht stimmte und ein *Gelbstich* im Bild zu sehen war. Das starke Prellen der Tastatur hat eine direkte Abfrage auf die Flanken erst nach dem Endprellen über einen Timer möglich gemacht. Beim Einstellen der Register hat sich herausgestellt, dass der CAN-Controller nur an einer Stelle im Code eingeschaltet werden kann, ansonsten nicht funktioniert. Ein Fehler in der Grafikbibliothek hat es nötig gemacht den LCD-Controller zweimal initialisieren zu müssen. Eine Analyse hat gezeigt, das der Speicher für die Funktionen zu spät reserviert wurde. Ein Eingriff in die Struktur konnte dieses Problem jedoch beheben. Weiterhin hat sich herausgestellt, dass die Routinen für die Eingabeverarbeitung nicht interruptfähig sind.

Die Touch-Funktionalität wurde Hardwaremäßig implementiert, und Softwaremäßig getestet. Da zu Beginn des Projektes diese jedoch noch nicht feststand, wurde drauf verzichtet die auf der Tastatureingabe bestehende Menüführung nochmal umzustellen. Die Funktionalität des Touchscreens wird zu einem späteren Zeitpunkt in das Symap Compact einfließen um kosten für die Hardware zu reduzieren. Zum jetzigen Zeitpunkt ist eine Steuerung per Finger oder Stift nur innerhalb der Menüstrukturen möglich.

Da die Platine fertig entworfen ist wird nach der Bestückung mit der CMA302 weitergearbeitet werden. Bis zu diesem Zeitpunkt wird der LED Treiber an einem Evaluationboard getestet um das Display optimal anzusteuern. Auf der Softwareseite ist das nächste Ziel, Parameter am Computer zu erstellen und via USB auf das Gerät zu laden und zu verteilen. Der gesamten grafischen Bereich soll fertig gestellt werden. Die variablen Seiten haben dabei die höchste Priorität.

Die Einstellung des Gerätes wird in der ersten Version lediglich über externe Software möglich sein. Erst im laufe der Zeit soll eine Parametrierung am Gerät selber möglich werden.

Literaturverzeichnis

[1] **Etschenberger, Konrad (2002):** *Controller Area- Network.* 3. Aufl., München/Wien: Carl Hanser Verlag

[2] **Lawrenz, Wolfgang (2000):** *CAN Grundlagen und Praxis.* 4. Aufl., Heidelberg: Hüthig GmbH

[3] **Seal, David (2001):** *ARM Architecture Referenz Manual.* 2. Aufl., Glagow: Addison-Wisley

[4] **Embedded Artsists (2008):** *OEM Base Board User Manual.*[*)]

[5] **Hitex (2009):** *Insiders Guide LPC2300/2400.*[*)]

[6] **NXP (2009):** *UM10237 LPC24XX User manual.*[*)]

[7] **NXP B.V. (2008):** *Datenblatt LPC2478.,* Rev. 01[*)]

[8] **Segger (2008):** *User's reference manual for emWin.* V4.18[*)]

[9] **Stucke Elektronik GmbH (2008):** *SYMAP® Bedienungsanleitung.*[*)]

[10] **Atterer, Richard (18.04.2007):** *Touch Screen Technologien.,* München: Ludwig-Maximilian-Unsiversität

http://www.medien.ifi.lmu.de/lehre/ws0607/mmi1/essays/Nihad-Zehic.xhtml

[11] **Boccella, Daniela / Krechel, Verena (06.07.2009):** *LCD,* Facharbeit

http://bbsw-koblenz.de/fachbereiche/klassen/fsi2003/TFT.htm

[12] **Embedded Software (06.07.2009):** *emWin.,* www.segger.com
http://www.segger.com/cms/emwin.html

[13] **Eye-Q (03.07.2009):** *Infrarot-Touch.,* Willich-Anrath

http://www.eye-q.de/infrarot_touch.html

[14] **IT-Lexikon (05.06.2009) :** *ARM (advanced RISC machine).,* DATACOM Buchverlag GmbH

http://www.itwissen.info/definition/lexikon/advanced-RISC-machine-ARM-ARM-Prozessor.html

[15] **PCMAG.COM (03.08.2009):** *transmissive LCD.,* Ziff Davis Publishing Holdings Inc.

http://www.pcmag.com/encyclopedia_term/0,2542,t=transmissive+LCD&i=55801,00.asp

Des weiteren wurde folgende Literatur für die Programmierung zur Hilfe gezogen:

Bollow, Friedrich / Homann, Matthias / Kähn, Klaus-Peter (2002): *C und C++ für Embedded Systems.* 1.Aufl., Bonn: mitp-Verlag

Erlenkötter, Helmut (2005): *C Programmieren von Anfang an.* 10. Aufl., Reinbek: Rowohlt Taschenbuch Verlag GmbH

Herrmann, Dietmar (1999): *Effektiv Programmieren in C und C++.* 4.Aufl., Braunschweig/Wiesbaden: Friedr. Vieweg & Sohn Verlagsgesellschaft mbH

*) Die Quellen sind als Anhang auf der CD zu finden

Abkürzungsverzeichnis

ACK	Acknowledge
AHB	Advanced High-performance Bus
ARM	Acorn Risc Machine
ALU	Arithmetic Logical Unit
CAN	Controller Area Network
CISC	Complex Instruction Set Computer
CRC	Cycling redundancy check
DMA	Direct Memory Access
EoF	End of Frame
EMC	Extern Memory Controller
HMI	Human Machine Interface
IFS	Intermission Frame Space
I/O	In Out
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light Emitting Diode
RTOS	Real time operating System
RISC	Reduced Instruction Set Computer
SD RAM	Synchronous Dynamic Random Access Memory
SIMD	Single Instruction Multiple Data
SMD	Surface-mounted device
STP	Shielded Twisted Pair
TFT	Thin- film Transistor
VIC	Vector Interrupt Controller
UART	Universal Asynchronous Receiver Transmitter

Bildverzeichnis

Tabellenverzeichnis

Anhang A

Anhang A1: Geschichte Segger

Tab. A.1 Geschichte SEGGER[a].

Jahr	Geschichte
1991	SEGGER Mikrocontroller Systeme wurde von Rolf Segger in Hilden gegründet. Zu diesem Zeitpunkt hatte er bereits zehn Jahre Erfahrung mit eingebetteten Systemen. Segger MCS fing an Software für verschiedene Gesellschaften in Deutschland und den Vereinigten Staaten zu entwickeln.
1993	Des erste RTOS wurde für die Mikrocontroller der NEC K0 Reihe entwickelt. Der Code wurde vollständig in Assembler geschrieben.
1994/95	Das RTOS wurde basierend auf Erfahrung mit eigenen Produkten und Feed-Backs des Kunden verbessert. Es wurde in "C" neu entworfen und verbesserte sich in Hinblick auf die maximale Vielseitigkeit, die Geschwindigkeit und den minimalen Speicherverbrauch.
1996	Es wurde die Entscheidung getroffen, embOS in einer Form zu präsentieren, die für alle unterstützten Zentraleinheiten völlig vereinbar ist. EmbOS für Mitsubishi M7700 und NECs V25/x86 Reihe wurde veröffentlicht.
1997	SEGGER fing die Entwicklung eines neuen Softwareproduktes emWin an. Eine grafische Software für monochrome Displays. Im November veröffentlichte SEGGER die erste emWin Version.
1998	Anfang der emWin/GSC Entwicklung mit enger Zusammenarbeit einer Automobilgesellschaft.
1999	In der Zusammenarbeit mit Mitsubishi fing die Entwicklung des Blinkers M16C an. Im April veröffentlichte SEGGER Blinker-Version 1. Im August präsentierte SEGGER die erste emWin/GSC Version incl. Bitmap Konverter, Schriftart Konverter und Windows Simulation.
2000/03	Im Laufe der Jahre fügte SEGGER eine breite Reihe von Eigenschaften zu ihren vorhandenen Produkten hinzu. Im August 2002 präsentierte SEGGER die erste emFile Version, die Multimediakarten unterstützt. Im November 2003 kam SEGGER mit einer neuen JTAG-Fehlersuchprogramm-Schnittstelle für ARM-Kerne heraus.
2004	Im Januar veröffentlichte SEGGER embOS Simulation, die dem Entwickler erlaubt, das Anwendungsprogramm ohne Bedürfnis nach einer funktionellen Hardware zu schreiben und zu prüfen. In der engen Zusammenarbeit mit dem IAR Schweden erweiterte SEGGER das J-Verbindungserzeugnis mit einem Fehlersuchprogramm-Emulator für den TI MSP430 Mikrocontroller.
2005	SEGGER öffnet ein amerikanisches Büro im Westminster, Massachusetts.
2006	SEGGER bringt ein neues Hardware-Produkt innerhalb des J-Verbindungserzeugnisses auf den Markt: J-Spur, ein JTAG Emulator einschließlich der Spur-Unterstützung

a) Übersetzt von http://www.segger.com/about_overview.html (23.06.2009).

Anhang A2: Elementstruktur

```
typedef struct
{
   union
   {
      struct              /* Switch */
      {
         U32 y1       : 9;  /* y1-koord             */
         U32 x1       :10;  /* x1-koord             */
         U32 id       : 4;  /* GrafikID             */
         U32 state    : 2;  /* Every switch gets 1 drawing for each
                               position the first gets a 1 for state,
                               the 2nd a 0  */
         U32 blink    : 1;  /* Element blinking     */
         U32 screen   : 2;  /* Screen Number        */
         U32 type     : 4;  /* Typidentification :1 */

         U32 y2       : 9;  /* y2-koord             */
         U32 x2       :10;  /* x2-koord             */
         U32 strength : 3;  /* Pen size             */
         U32 bgcolor  : 5;  /* Backgroundcolor      */
         U32 fgcolor  : 5;  /* Foregroundcolor      */

         U32 param    :16;  /* Link to Parameter    */
         U32 num      :16;  /* Elementnumber   */
      } e_switch;

      struct                 /* Line  */
      {
         U32 y1       : 9;  /* y1-koord             */
         U32 x1       :10;  /* x1-koord             */
         U32 reserve  : 7;  /* Reserve              */
         U32 screen   : 2;  /* Screen Number        */
         U32 type     : 4;  /* Typidentification :2 */


         U32 y2       : 9;  /* y2-koord             */
         U32 x2       :10;  /* x2-koord             */
         U32 strength : 3;  /* Pen size             */
         U32 bgcolor  : 5;  /* Backgroundcolor      */
         U32 fgcolor  : 5;  /* Foregroundcolor      */

         U32 reserve2 :16;  /* Reserve              */
         U32 Num      :16;  /* Elementnumber        */
      } e_line;

      struct           /* rect */
      {
         U32 y1       : 9;  /* y1-koord             */
         U32 x1       :10;  /* x1-koord             */
         U32 filled   : 1;  /* Filled               */
         U32 reserve  : 6;  /* Reserve              */
         U32 screen   : 2;  /* Screen Number        */
         U32 type     : 4;  /* Typidentification :3 */

         U32 y2       : 9;  /* y2-koord             */
         U32 x2       :10;  /* x2-koord             */
         U32 strength : 3;  /* Pen size             */
         U32 bgcolor  : 5;  /* Backgroundcolor      */
         U32 fgcolor  : 5;  /* Foregroundcolor      */
```

```
   U32 reserve2 :16;  /* Reserve               */
   U32 Num      :16;  /* Elementnumber         */
} e_rect;

struct                /* triangle */
{
   U32 y1       : 9;  /* y1-koord               */
   U32 x1       :10;  /* x1-koord               */
   U32 filled   : 1;  /* Filled                 */
   U32 rotation : 2;  /* Rotation 0=0, 01=90, 10=180, 11=270 */
   U32 reserve  : 4;  /* Reserve                */
   U32 screen   : 2;  /* Screen Number          */
   U32 type     : 4;  /* Typidentification :4   */

   U32 y2       : 9;  /* y2-koord               */
   U32 x2       :10;  /* x2-koord               */
   U32 strength : 3;  /* Pen size               */
   U32 bgcolor  : 5;  /* Backgroundcolor        */
   U32 fgcolor  : 5;  /* Foregroundcolor        */

   U32 reserve2 :16;  /* Reserve                */
   U32 Num      :16;  /* Elementnumber          */
} e_triangle;

struct                /* circle */
{
   U32 y1       : 9;  /* y1-koord              */
   U32 x1       :10;  /* x1-koord              */
   U32 filled   : 1;  /* Filled                */
   U32 reserve  : 6;  /* Reserve               */
   U32 screen   : 2;  /* Screen Number         */
   U32 type     : 4;  /* Typidentification :5 */

   U32 reserve2 : 9;  /* Reserve               */
   U32 x2       :10;  /* x2-koord (r= x1-x2)   */
   U32 strength : 3;  /* Pen size              */
   U32 bgcolor  : 5;  /* Backgroundcolor       */
   U32 fgcolor  : 5;  /* Foregroundcolor       */

   U32 reserve3 :16;  /* Reserve               */
   U32 num      :16;  /* Elementnumber         */

} e_cyrcle;

struct                /* Value */
{
   U32 y1       : 9;  /* y-koord               */
   U32 x1       :10;  /* x-koord               */
   U32 id       : 7;  /* GrafikID              */
   U32 screen   : 2;  /* Screen Number         */
   U32 type     : 4;  /* Typidentification :6 */

   U32 font     : 8;  /* font                  */
   U32 reserve  : 14; /* Reserve               */
   U32 bgcolor  : 5;  /* Backgroundcolor       */
   U32 fgcolor  : 5;  /* Foregroundcolor       */

   U32 param    :16;  /* Link to Parameter     */
   U32 num      :16;  /* Elementnumber         */
} e_mvalue;
```

```
      struct                  /* Textelement          */
      {
         U32 y         : 9;   /* y-koord              */
         U32 x         :10;   /* x-koord              */
         U32 reserve   : 7;   /* Reserve              */
         U32 screen    : 2;   /* Screen Number        */
         U32 type      : 4;   /* Typidentifikation :7 */

         U32 reserve2  : 6;   /* Reserve              */
         U32 bgcolor   : 5;   /* Backgroundcolor      */
         U32 fgcolor   : 5;   /* Foregroundcolor      */

         U8  text[6];         /* Text (6 Chars)       */
      } e_text;

      struct                  /* aktive textbox       */
      {
         U32 y         : 9;   /* y-koord              */
         U32 x         :10;   /* x-koord              */
         U32 id        : 7;   /* GrafikID             */
         U32 screen    : 2;   /* Screen Number        */
         U32 type      : 4;   /* Typidentifikation :8 */

         U32 reserve   :14;   /* Reserve              */
         U32 textbox   : 1;   /* Auto/Man; Local...   */
         U32 state     : 1;   /* State                */
         U32 font      : 6;   /* Font                 */
         U32 bgcolor   : 5;   /* Backgroundcolor      */
         U32 fgcolor   : 5;   /* Foregroundcolor      */

         U32 param     :16;   /* Link to Parameter    */
         U32 num     :16;     /* Elementnumber         */
      } e_textbox;

        struct                  /* Progbar              */
      {
         U32 y         : 9;   /* y-koord              */
         U32 x         :10;   /* x-koord              */
         U32 id        : 7;   /* GrafikID             */
         U32 screen    : 2;   /* Screen Number        */
         U32 type      : 4;   /* Typidentifikation :9 */

         U32 height    : 9;   /* Height               */
         U32 width     :10;   /* Width                */
         U32 strength  : 3;   /* Pen size             */
         U32 bgcolor   : 5;   /* Backgroundcolor      */
         U32 fgcolor   : 5;   /* Foregroundcolor      */

         U32 value     : 8;   /* Value                */
         U32 font      : 6;   /* Font                 */
         U32 type2     : 3;   /* V; A; kA; W; kW; ..  */
         U32 param     :15;   /* Link to Parameter    */
      } e_progbar;

      struct
      {
         U32 dword1;          /* To insert         */
         U32 dword2;          /* into structure    */
         U32 dword3;          /* 3x4 Byte          */
      } insert;
   } elemente;
} LCD_PARAMETER;
```

Anhang A3: Pinbelegung LPC2478

Tab. A.2 Pinnbelegung GU.

lfdnr	Symbol	Pin	Ball	Type	Description	Use
1	P0.0	94	U15	I/O	General purpose digital input/output pin	Tastatur 1 (AUTO/MAN)
5	P0.1	96	T14	I/O	General purpose digital input/output pin	Tastatur 2 (STOP)
9	P0.2	202	C4	I/O	General purpose digital input/output pin	Tastatur 3 (ACK)
11	P0.3	204	D6	I/O	General purpose digital input/output pin	Tastatur 4 (DOWN)
13	P0.4	168	B12	I/O	General purpose digital input/output pin	Tastatur 5 (LEFT)
17	P0.5	166	C12	I/O	General purpose digital input/output pin	Tastatur 6 (UP)
21	P0.6	164	D13	I/O	General purpose digital input/output pin	Tastatur 7 (ENTER)
25	P0.7	162	C13	I/O	General purpose digital input/output pin	Tastatur 8 (I)
29	P0.8	160	A15	I/O	General purpose digital input/output pin	Tastatur 9 (RIGHT)
33	P0.9	158	C14	I/O	General purpose digital input/output pin	Tastatur 10 (F1)
37	P0.10	98	T15	I/O	General purpose digital input/output pin	Tastatur 11 (F3)
41	P0.11	100	R14	I/O	General purpose digital input/output pin	Tastatur 12 (F4)
45	P0.12	41	R1	I/O	General purpose digital input/output pin	Tastatur 13 (F2)
49	P0.13	45	R2	I/O	General purpose digital input/output pin	Tastatur 14 (START)
53	P0.14	69	T7	I/O	General purpose digital input/output pin	Tastatur 15 (INV)
57	P0.15	128	J16	I/O	General purpose digital input/output pin	Tastatur 16
73	P0.19	122	L17	I/O	General purpose digital input/output pin	Horn
84	RD1	118	M16	I	CAN1 receiver input	CAN
88	TD1	116	N17	O	CAN1 transmitter output	CAN
89	P0.23	18	H1	I/O	General purpose digital input/output pin	LED- STROM
93	P0.24	16	G2	I/O	General purpose digital input/output pin	LCD R/L
97	P0.25	14	F1	I/O	General purpose digital input/output pin	LCD U/D
103	AOUT	12	E1	O	D/A converter output	Lautsprecher
168	PWM1.1	66	P7	O	Pulse width modulator 1, channel 1 output	DY: PWM
174	P1.20	70	U7	I/O	General purpose digital input/output pin	LCDVD[10] (G0)
178	P1.21	72	R8	I/O	General purpose digital input/output pin	LCDVD[11] (G1)
182	P1.22	74	U8	I/O	General purpose digital input/output pin	LCDVD[12] (G2)
186	P1.23	76	P9	I/O	General purpose digital input/output pin	LCDVD[13] (G3)
190	P1.24	78	T9	I/O	General purpose digital input/output pin	LCDVD[14] (G4)
194	P1.25	80	T10	I/O	General purpose digital input/output pin	LCDVD[15] (G5)
198	P1.26	82	R10	I/O	General purpose digital input/output pin	LCDVD[20] (B1)
202	P1.27	88	T12	I/O	General purpose digital input/output pin	LCDVD[21] (B2)
206	P1.28	90	T13	I/O	General purpose digital input/output pin	LCDVD[22] (B3)
210	P1.29	92	U14	I/O	General purpose digital input/output pin	LCDVD[23] (B4)
221	AD0.5	40	P1	I	A/D converter 0, input 5	Temperatur
222	P2.0	154	B17	I/O	General purpose digital input/output pin	LCDPWR
230	P2.2	150	D15	I/O	General purpose digital input/output pin	LCDDCLK

Tab. A.2 Fortsetzung.

lfdnr	Symbol	Pin	Ball	Type	Description	Use
234	P2.3	144	E16	I/O	General purpose digital input/output pin	LCDFP
238	P2.4	142	D17	I/O	General purpose digital input/output pin	LCDENAB
242	P2.5	140	F16	I/O	General purpose digital input/output pin	LCDLP
246	P2.6	138	E17	I/O	General purpose digital input/output pin	LCDVD[4] (R1)
250	P2.7	136	G16	I/O	General purpose digital input/output pin	LCDVD[5] (R2)
254	P2.8	134	H15	I/O	General purpose digital input/output pin	LCDVD[6] (R3)
258	P2.9	132	H16	I/O	General purpose digital input/output pin	LCDVD[7] (R4)
263	EINT0	110	N15	I	External interrupt 0 input (LOW during RESET --> BootLoader) (inverted!)	UART/JTAG
268	P2.12	106	N14	I/O	General purpose digital input/output pin	LCDVD[3] (R0)
272	P2.13	102	T16	I/O	General purpose digital input/output pin	LCDVD[19] (B0)
280	P2.15	99	P13	I/O	General purpose digital input/output pin	TP_IRQ
285	CAS	87	R11	O	SDRAM column address strobe (inverted!)	RAM
287	RAS	95	R13	O	SDRAM row address strobe (inverted!)	RAM
289	CLKOUT0	59	U3	O	SDRAM Clock 0	RAM
293	DYCS0	73	T8	O	SDRAM chip select 0 (inverted!)	RAM
299	SCK0	85	U12	I/O	Serial Takt for SSP0	Flash
303	SSEL0	64	U5	I/O	Slave Select for SSP0	Flash
305	CKEOUT0	53	P5	O	SDRAM Clock enable 0	RAM
315	MOSI0	47	P3	I/O	Master out slave in for SSP0	Flash
317	DQMOUT0	49	P4	O	Data mask 0 used with SDRAM and static devices	RAM
319	DQMOUT1	43	N3	O	Data mask 1 used with SDRAM and static devices	RAM
329	D0	197	B4	I/O	External Memory data line 0	RAM
331	D1	201	B3	I/O	External Memory data line 1	RAM
333	D2	207	B1	I/O	External Memory data line 2	RAM
335	D3	3	E4	I/O	External Memory data line 3	RAM
337	D4	13	F2	I/O	External Memory data line 4	RAM
339	D5	17	G1	I/O	External Memory data line 5	RAM
341	D6	23	J1	I/O	External Memory data line 6	RAM
343	D7	27	L1	I/O	External Memory data line 7	RAM
345	D8	191	D8	I/O	External Memory data line 8	RAM
347	D9	199	C5	I/O	External Memory data line 9	RAM
349	D10	205	B2	I/O	External Memory data line 10	RAM
351	D11	208	D5	I/O	External Memory data line 11	RAM
353	D12	1	D4	I/O	External Memory data line 12	RAM
355	D13	7	C1	I/O	External Memory data line 13	RAM
357	D14	21	H2	I/O	External Memory data line 14	RAM
359	D15	28	M1	I/O	External Memory data line 15	RAM

Tab. A.2 Fortsetzung.

lfdnr	Symbol	Pin	Ball	Type	Description	Use
408	P3.28	5	D2	I/O	General purpose digital input/output pin (LSB)	GeräteID 1
412	P3.29	11	F3	I/O	General purpose digital input/output pin	GeräteID 2
416	P3.30	19	H3	I/O	General purpose digital input/output pin	GeräteID 3
420	P3.31	25	J3	I/O	General purpose digital input/output pin (MSB)	GeräteID 4
424	A0	75	U9	I/O	External Memory address line 0	RAM
426	A1	79	U10	I/O	External Memory address line 1	RAM
428	A2	83	T11	I/O	External Memory address line 2	RAM
430	A3	97	U16	I/O	External Memory address line 3	RAM
432	A4	103	R15	I/O	External Memory address line 4	RAM
434	A5	107	R16	I/O	External Memory address line 5	RAM
436	A6	113	M14	I/O	External Memory address line 6	RAM
438	A7	121	L16	I/O	External Memory address line 7	RAM
440	A8	127	J17	I/O	External Memory address line 8	RAM
442	A9	131	H17	I/O	External Memory address line 9	RAM
444	A10	135	G17	I/O	External Memory address line 10	RAM
446	A11	145	F14	I/O	External Memory address line 11	RAM
448	A12	149	C16	I/O	External Memory address line 12	RAM
450	A13	155	B16	I/O	External Memory address line 13	RAM
452	A14	159	B15	I/O	External Memory address line 14	RAM
466	SCK1	109	R17	I/O	Serial Clock for SSP1	Touch
470	SSEL1	115	M15	I/O	Slave select for SSP1	Touch
474	MISO1	123	K14	I/O	Master in slave out for SSP1	Touch
478	MOSI1	129	J15	I/O	Master out slave in for SSP1	Touch
482	WE	179	B9	O	Write enable signal (inverted!)	RAM
501	DBGEN	9	F4	I	JTAG interface control signal	UART/JTAG
502	TDO	2	D3	O	Test Data out for JTAG interface	UART/JTAG
503	TDI	4	C2	I	Test Data in for JTAG interface	UART/JTAG
504	TMS	6	E3	I	Test Mode select for JTAG interface	UART/JTAG
505	TRST	8	D1	I	Test reset for JTAG interface (inverted!)	UART/JTAG
506	TCK	10	E2	I	Test Clock for JTAG interface	UART/JTAG
507	RTCK	206	C3	I/O	JTAG interface control signal	UART/JTAG
508	RSTOUT	29	K3	O	being in reset state (inverted!)	RSTOUT
509	RESET	35	M2	I	external reset input	UART/JTAG

Anhang A4: Tabelle LCDs

Tab. A.3 Vollständige Tabelle der zu Verfügung stehenden LCDs.

Hersteller	Typ-Nr.	Helligkeit (cd/m²)	Kontrast	Abmessungen	Typ	Versorgung	Temp. (°C)	Anschluss	Anschlusstyp	Lebensdauer Beleuchtung	Preis 100 Stk	Preis 1000 Stk	Anbieter	Kommentar
Ampire	AM320240N6TMQW-00H	300	350	126,0x101,8x9,7	LED	126 mA / 10,5 V	-20 bis +70	40 Pin	FA5B040HP1R3000	10.000 Std	59,80 € 63,70 €	45,50 € 51,90 €	Zettler Beck Elektronik	Interface 18 Bit
Ampire	TF320240-20-0	450	350	131,0x102.2xN/A	LED	126 mA / 10,5 V	-20 bis +70	24 / 34 Pin	mehrere	30.000 Std			Zettler AMP Display	Registeransteuerung
Ampire	AM320240NTMQW-30H	500	350	131,0x102,2x10,9	LED	126 mA / 9,6 V	-20 bis +70	40 Pin	FA5B040HP1R3000	30.000 Std			Zettler	Interface 18 Bit
Ampire	AM320240NTMQW-00H	500	350	131,0x102,2x10,9	LED	126 mA / 9,6V	-20 bis +70	40 Pin	FA5B040HP1R3000	30.000 Std		56,20 € 53,30 € 52,00 €	Beck Elektronik Mewa electronic Ultratronik	Interface 18 Bit
Ampire	AM320240NTMCW-T00H-A	260	270	131,0x102,2x12,4	CCFL	6 mA / 650 V / 55 kHz	-20 bis +70	40 Pin					Zettler	Interface 18 Bit
Ampire	AM320240NTMQW-01H	800	350	131,0x102,2x10,9	LED		-20 bis +70	40 Pin					Zettler	Interface 18 Bit
Ampire	AM320240NTMQW-W0H	500	350	131,0x102,2x10,9	LED		-20 bis +70	40 Pin					Zettler	Registeransteuerung
Ampire	AM320240NSTNQW-00H	500	170	144,0x104,6x13,0	LED	330 mA / 10,5 V	-20 bis +70	33 Pin	ELCO 08-6210-033-340-800A+	10.000 Std			Zettler	Interface 18 Bit
Ampire	AM320240N1TMQW-W0H(R)	450	350	167,0x109,0x13,1	LED		-20 bis +70	24 / 34 Pin					Zettler	Registeransteuerung
Ampire	AM320240NSTNQW-W0H	500	350	144,0x104,6x13,0	LED	330 mA / 10,5 V	-20 bis +70	54 Pin		30.000 Std			Zettler	Registeransteuerung
Andi	YL#T320x240-17-057L2H	450	350	167,0x109,0x8,9	LED	126 mA / 10,5 V	-20 bis +70	40 Pin	FA5B040HP1R3000	10.000 Std		65,85 €	Lehner Dabitros	Interface 18 Bit Sehr gute Befestigungsmöglichkeit
AUO	G057QN01	400	600	144,0x104,6x12,3	LED	15 mA	-30 bis +85	33 Pin	Hirose FH12-33S-0.5SH(55) or comp.	40.000 Std	79,50 €	65,80 €	Data-Modul Mostron	Interface 18 Bit
CTC	T2432A11VP01	400	400	144,0x104,6x13,0	CCFL	8 mA / 710 V / 35 kHz	-20 bis +70	33 Pin	ELCO 08-6210-033-340-800A+	45.000 Std			Neumüller	Interface 18 Bit
CTC	T2432C04VP01	250	250	126,1x101,5x4,8	LED	60 mA / 12,4 V	-20 bis +70	54 Pin	N/A	20.000 Std			Neumüller	Interface 24 Bit
Data Image	FG050701DSSWBG01	220	350	127,0x98,4x9,7	LED	5V	-20 bis +70	33 Pin	ELCO 08-6210-033-340-800	.		N/A 50,00 €	Ultratronik Mewa electronic	Interface 18 Bit
Data Image	FG050701DSSWBGL2	490	350	127,0x98,4x9,7	LED	5V	-20 bis +70	33 Pin	ELCO 08-6210-033-340-800			54,00 €	Ultratronik	Interface 18 Bit
Data Image	FG050700DSSWDG01	360	350	144,0x104,8x12,7	LED	k.A.	-20 bis +70	33 Pin	ELCO 08-6210-033-340-800			N/A 47,85 €	Ultratronik Mewa electronic	Interface 18 Bit
DEM	DEM320240H TMH-PW-N	250	250	126,1x101,5x6,4	LED	60 mA / 13 V	-20 bis +70	60 Pin		50.000 Std		64,80 €	WF-Elektronik	Interface 24 Bit
EDT	ET057003DM6	400	300	124,7x100,0x6,0	LED	60 mA / 30 V	-20 bis +70	33 Pin	IMSA-9637S-33A-TB	40.000 Std			Ultratronik	Interface 18 Bit
Everbouquet	MF320240B57-BF	380	250	159,4x111,0x9,0	LED	200 mA / 8,6 V	-20 bis +60	40 Pin	N/A			70,00 €	Neumüller	Interface 24 Bit Mindestabnahme 500 Stück sehr gute Befestigungsmöglichkeit
Gi Far	GFT057FA320240	N/A	250	120,3x95,7x1,6	LED	N/A	-20 bis +70	50 Pin	IRISO 9637S-50A-TB				Pohl Electronic	Braucht 4 verschiedene Spannungen Interface 18 Bit
Gi Far	GFT057AA320240	500	350	131,0x102,2x10,9	LED	126 mA / 9,6 V	-20 bis +70	40 Pin	FA5B040HF1R3000	30.000 Std	69,90 €	64,60 €	Pohl Electronic	Interface 18 Bit

Tab. A.3 Fortsetzung.

Hersteller	Typ-Nr.	Helligkeit (cd/m²)	Kontrast	Abmessungen	Typ	Versorgung	Temp. (°C)	Anschluss	Anschlusstyp	Lebensdauer Beleuchtung	Preis 100 Stk	Preis 1000 Stk	Anbieter	Kommentar
Hantronix	HDA570S-F	500	350	144,0x104,6x13,0	CCFL		-20 bis +70	33 Pin		50.000 Std	80,50 US$	75,65 US$	Ineltek	Interface 18 Bit
Hantronix	HDA570S-V	380	450	144,0x104,6x13,0	LED	75 mA / 22 V	-20 bis +70	33 Pin	FH12-33S-0.5SH	10.000 Std	80,50 US$	74,25 US$	Ineltek	Interface 18 Bit
Hitachi	TX14D12VM1CBA	350	350	131,0x102,2x10,9	CCFL		-20 bis +70	40 Pin		50.000 Std				Interface 18 Bit
Hitachi	TX14D12VM1CBB	600	350	131,0x102,2x10,9	CCFL		-20 bis +70	40 Pin		50.000 Std				Interface 18 Bit
Hitachi	TX14D12VM1CBC	350	350	131,0x102,2x10,9	LED		-20 bis +70	40 Pin		40.000 Std	119,00 € N/A	109,00 € 98,95 €	Data-Modul Glyn	Interface 18 Bit
Hitachi	TX14D12VM1CPC	280	350	131,0x102,2x12,4	LED		-20 bis +70	40 Pin		40.000 Std	133,00 €	118,00 €	Data-Modul	Interface 18 Bit (mit Touch)
Kyocera	TCG057QV1AA-G00	470	300	144,0x104,8x12,7	CCFL		-10 bis +70	33 Pin		75.000 Std		N/A N/A 91,70 €	Ultratronik Data-Modul Display Devices	Interface 18 Bit 10 Jahre Verfügbarkeitsgarantie Mechanisch identisch mit Powertip und Primeview
Kyocera	TCG057QVLAC-G00	350	450	144,0x104,8x13,0	LED		-10 bis +70	33 Pin		50.000 Std	116,00 € N/A N/A	105,00 € 90,00 € 119,00 €	Data-Modul Ultratronik Display Devices	Interface 18 Bit Mechanisch identisch mit Powertip und Primeview
Kyocera	TCG057QVLBA-G00	300	500	127,2x100,4x5,7	LED	30 mA / 22,1 V	-20 bis +70	40 Pin	IMSA-9681S-40A-GF			70,00 €	Ultratronik	Interface 18 Bit
Nanya	LM740-0A	400	250	143,7x104,4x9,7	LED	140 mA / 10,2 V	-20 bis +70	33 Pin	Hirose FH12-33S-0.5SH(55) or comp.	40.000 Std		57,20 €	Getronic	Interface 18 Bit
Newtec	NFD57A-CIW	350	200	126,0x101,6x5,7	LED		-20 bis +70	50 Pin						
Optrex	T-55265GD057J-LW-AAN	500	350	144,0x104,6x13,0	LED	180 mA / 13,2 V	-20 bis +70	33 Pin	N/A			65,00 €	Display LC	Interface 18 Bit
OSD	OSD057ACA6A0	300	200	126,0x101,6x5,7	LED	14 0mA / 10,2 V	-20 bis +70	50 Pin						Interface 24 Bit
Powertip	PH320240T-005-IC1Q	450	250	144,0x104,6x13,0	LED	25 mA / 23,5 V	-20 bis +70	33 Pin	ELCO 08-6210-033-340-800+	mind. 30k	59,00 € N/A	55,50 € 60,32 €	Display Devices Mewa-electronic	Mechanisch identisch mit Primeview und Kyocera
Powertip	PH320240T-005-I-Q	380	500	167,0x111,0x7,8	LED	60 mA / 33 V	-20 bis +70	40 Pin	IMSA-9637S-40Y905(Btm)	54.600 Std	49,00 € 52,90 € N/A	43,00 € 49,95 € 54,30 €	Data-Modul Display Devices Mewa-electronic	Interface 24 Bit Sehr gute Befestigungsmöglichkeit
PrimeView	PD057VU5	700	400	144,0x104,6x13,0	LED	240 mA / 11 V	-30 bis +70	33 Pin	ELCO 08-6210-033-340-800A+	20.000 Std	N/A 91,00 €	107,50 US$ 88,57 € 84,00 €	Ineltek Hy-Line Display Devices	Interface 18 Bit Garantie auf Hardwaremäßige Gleichheit über mehrere Jahre Mechanisch identisch mit Powertip und Kyocera
Sharp	LQ057V3DG01	400	600	144,0x104,6x12,3	CCFL		-30 bis +80	33 Pin		50.000 Std		109,00 € N/A	E-V-B Data-Modul	
Sharp	LQ057Q3D12	500	350	144,0x104,6x13,0	CCFL		-30 bis +80	33 Pin		50.000 Std		N/A 80,90 €	Data-Modul E-V-B	Interface 18 Bit
Suntai	SFD-057A	300	250	144,0x104,6x13,0	CCFL		-10 bis +70	33 Pin						
Tianma	TMT057DNAFWU	250	250	126,1x101,6x6,4	LED	60 mA / 12,4 V	-20 bis +70	54 Pin	N/A	50.000 Std		45,00 €	Display LC	Interface 24 Bit
Toshiba	LTA057A340F	400	500	144,0x104,6x7,8	LED	45 mA / 19,2 V	-10 bis +70	33 Pin	N/A	20.000 Std		68,00 €	Glyn	Interface 18 Bit
URT	UMSH-8044MD-T	300	250	144,0x104,6x13,0	LED	60 mA / 23,1 V	-20 bis +70	33 Pin	IRISO 9637S-33A-TB	50.000 Std	73 US$	66 US$	Abacus-Deltron	Interface 18 Bit
Vbest	VGG322403-6UFLWA	500	350	144,0x104,6x13,0	LED		-20 bis +70	33 Pin						
Winstar	WF57ATIACD0#	350	200	126,0x101,6x5,7	LED	140 mA / 10,2 V	-20 bis +70	50 Pin		10.000 Std				Interface 24 Bit
Winstar	WF57ATIACD0#000	350	200	126,0x101,6x7,2	LED		0 bis +50	50 Pin		10.000 Std				Interface 24 Bit (mit Touch)
Winstar	WF57ATIBCD0#	200	300	141,1x102,7x12,7	LED		0 bis +70	50 Pin		10.000 Std				16 Bit Daten / 18 Bit Adresse

Anhang B

Anhang B1: Quellcode (CD)*)

Anhang B2: Symap Compact Frontansicht (CD)

Anhang B3: Manual SEGGER (CD)

Anhang B4: Manual LPC2478 (CD)

Anhang B5: Präsentation Symap (CD)

Anhang B6: Abmessungen LCDs (CD)

Anhang B7: Stromlaufplan CMA302 (CD)*)

Anhang B8: Menüstruktur – Settings (CD)

Anhang C Datenblätter

Anhang C1: CAN TRANCEIVER SN65HVD251D (CD)

Anhang C2: LED-DRIVER TPS61165 (CD)

Anhang C3: S_FLASH M4PE89 (CD)

Anhang C4: SD-RAM MT48LC16M16A2 (CD)

Anhang C5: TFT G240320LTSW-118W-E(3.2)v1.0 (CD)

Anhang C6: TSC TSC2046 (CD)

Anhang C7: Verstärker LM386 (CD)

Anhang C8: Controller LPC2478 (CD)

Anhang C9: LCD Controller SSD1289_1.3 (CD)

*) Kann bei Prof. Dr. rer. nat. Schneider eingesehen werden.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit...

Embedded System als Human- Machine- Interface mit flexibler Menüstruktur und CAN Anbindung an ein digitales Schutzrelais

...im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht

Ort, Datum Unterschrift