Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Praktikum Systemprogrammierung

Versuch 6

Touch-LCD

Lehrstuhl Informatik 11 - RWTH Aachen

16. Juni 2023

Commit: c3b830de

Inhaltsverzeichnis

6 Touch-LCD **3**
6.1 Versuchsinhalte . . . 3
6.2 Lernziel . . . 3
6.3 Grundlagen . . . 3
6.3.1 Callbacks . . . 4
6.3.2 Serial Peripheral Interface . . . 4
6.3.3 Small Protokoll . . . 8
6.4 Hausaufgaben . . . 13
6.4.1 Bitübertragungs- und Sicherungsschicht . . . 14
6.4.2 Aufbereiten der empfangenen Daten . . . 17
6.4.3 Implementierung der grafischen Steuerung . . . 18
6.4.4 Benutzerinteraktion mittels Schaltflächen . . . 19
6.4.5 Implementierung einer Paint Anwendung . . . 20
6.4.6 Zusammenfassung . . . 22
6.5 Pinbelegungen . . . 24

Dieses Dokument ist Teil der begleitenden Unterlagen zum *Praktikum Systemprogrammierung*. Alle zu diesem Praktikum benötigten Unterlagen stehen im Moodle-Lernraum unter https://moodle.rwth-aachen.de zum Download bereit. Folgende E-Mail-Adresse ist für Kritik, Anregungen oder Verbesserungsvorschläge verfügbar:

support.psp@embedded.rwth-aachen.de

6 Touch-LCD

Als Erweiterung zum aktuell verwendeten 2x16 character LCD bietet sich ein berührungsempfindliches Display an, dessen Konzept unter anderem durch Smartphones den Weg in unseren Alltag gefunden hat. Diese Technologie ermöglicht die Verwendung einer einzigen Oberfläche sowohl als Eingabe- als auch als Ausgabemedium. Neben der gesteigerten Platzeffizienz werden gleichzeitig neue Möglichkeiten zur Interaktion zwischen Mensch und Maschine möglich. So kann ein Anwender beispielsweise in derselben Art und Weise auf einem TLCD schreiben oder zeichnen, wie er es mit einem Kugelschreiber auf einem Blatt Papier tun würde.

6.1 Versuchsinhalte

In diesem Versuch wird die Ansteuerung des TLCDs EA eDIPTFT43-A mithilfe des ATmega 644 realisiert. Die Kommunikation erfolgt über das *Serial Peripheral Interface* (SPI) und dem darüber übertragenen *Small Protokoll* des TLCDs. Darauf aufbauend wird eine Paint Anwendung implementiert, die das farbige Malen auf dem Display ermöglicht.

6.2 Lernziel

Das Lernziel dieses Versuchs ist das Verständnis der folgenden Zusammenhänge:

- Bitübertragung über das Serial Peripheral Interface
- Sicherung der Datenkonsistenz durch das Small Protokoll
- Empfangen und Verarbeiten von Touch-Events

6.3 Grundlagen

Zur Lösung der Aufgaben werden einige Grundlagen benötigt. Im Folgenden wird zunächst die Kommunikation auf Ebene der Bitübertragungsschicht mittels SPI erklärt. Das darauf folgende Kapitel beschreibt die vom TLCD vorgegebene Sicherungsschicht *Small Protokoll*.

6.3.1 Callbacks

Callbacks sind ein Konzept der Informatik, das erlaubt sogenannte Callback-Funktionen zum späteren Aufruf zu registrieren. Diese Funktionen werden beim Eintreten eines Ereignisses aufgerufen und haben als Argument Informationen zu diesem Ereignis. Damit lässt sich CPU-Zeit intensives Warten auf das Eintreten von Ereignissen vermeiden.

6.3.2 Serial Peripheral Interface

SPI ist ein synchroner, serieller Datenbus, durch den Komponenten nach dem Master-Slave-Prinzip miteinander verbunden werden. Wir betrachten zunächst wie die Bitübertragungsschicht in SPI realisiert ist. Anschließend werden die SPI-Schnittstellen des Mikrocontrollers und TLCDs erklärt.

Bitübertragungsschicht

Die Bitübertragungsschicht behandelt den Abschnitt der Kommunikation, der die semantische De- und Enkodierung einzelner Bits auf einem physikalischen Medium beschreibt. Die synchrone Datenübertragung im Bus erfolgt über die folgenden drei Leitungen, an denen jede Buskomponente angeschlossen wird:

- MISO (Master In, Slave Out)
- MOSI (Master Out, Slave In)
- CLK (Clock)

Die Zugriffsverwaltung auf den SPI-Bus erfolgt nach dem sogenannten Master-Slave-Prinzip. Dem Master steht neben den obigen drei Leitungen zur Datenübertragung für jeden angeschlossenen Slave eine Chip Select (`CS`) Leitung zur Verfügung, um einen Slave zur Kommunikation auszuwählen. Hierzu wechselt der Master die an der entsprechenden `CS`-Leitung anliegende Spannung von high nach low. Im Takt des vom Master generierten Clock-Signals, das an der CLK-Leitung anliegt, sendet der Master Daten an den selektierten Slave über die MOSI-Leitung und empfängt Daten vom Slave über die MISO-Leitung. Dabei wird der Pegel einer Datenleitung zu den Zeitpunkten von fallenden oder steigenden Clock-Flanken (Flanken im Signalverlauf der CLK-Leitung) als übertragenes Bit interpretiert. Wie diese Interpretation im Detail erfolgt, wird durch den konfigurierten SPI-Modus festgelegt.
In SPI wird zwischen vier verschiedenen Modi zur Datenübertragung unterschieden, die durch die booleschen Parameter Clock Polarität (`CPOL`) und Clock Phase (`CPHA`) des ATmega 644 definiert werden. Die resultierenden Modi aus den entsprechenden Konstellationen von `CPOL` und `CPHA` sind in Abbildung 6.1 aufgelistet.

Abbildung 6.1 veranschaulicht exemplarisch die Bedeutung der verschiedenen SPI-Modi. `CPOL=0` bedeutet, dass das Clock-Signal mit einem Low-Pegel startet. Folglich ist die erste Flanke des Clock-Signals steigend. Entsprechend bedeutet `CPOL=1`, dass das Clock-Signal mit einem High-Pegel beginnt und die erste Flanke fallend ist.

Modus	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Tabelle 6.1: SPI-Modi

CPHA definiert, ob bei einer geraden oder ungeraden Clock-Flanke ein an einer Datenleitung anliegender Pegel als Bit interpretiert wird. Ist CPHA=0, dann wird zum Zeitpunkt der ersten und somit bei jeder ungeraden Clock-Flanke der Wert des aktuell zu übertragenden Bits entsprechend des Pegels der Datenleitung übernommen. Analog gilt für CPHA=1, dass bei der zweiten und somit jeder geraden Flanke der Wert der Datenleitung als übertragenes Bit interpretiert wird.
Dabei ist zu beachten, dass es von CPOL abhängig ist, ob die geraden und ungeraden Flanken fallend bzw. steigend sind.

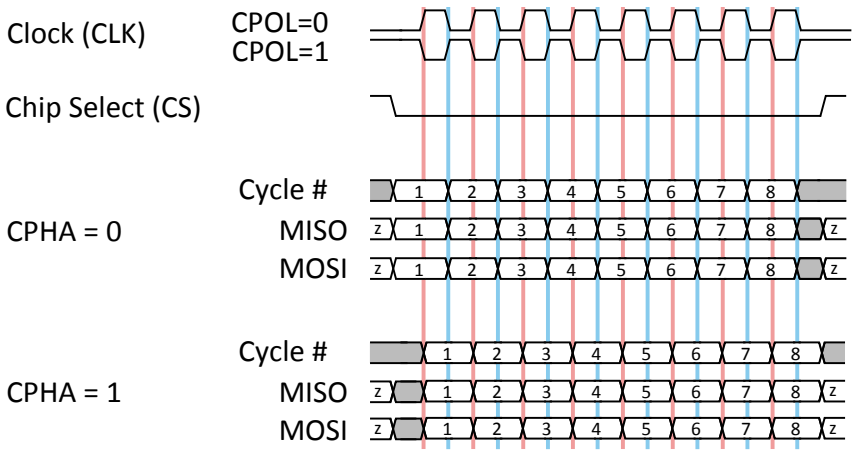

Abbildung 6.1: Semantik der SPI-Modi[1]

Die Initialisierung des SPI-Busses muss vor dem Selektieren eines Slaves durch das Ziehen der entsprechenden CS-Leitung von high nach low beendet sein. Zu diesem Zeitpunkt muss das Clock-Signal, abhängig vom Parameter CPOL, bereits auf low bzw. high gezogen worden sein.
Oftmals wird zusätzlich ein dritter Paramater Data Order (DORD) angegeben, der die Reihenfolge der Bitübertragung definiert. DORD gibt an, ob zuerst das „Most Significant Bit“ (MSB) oder das „Least Significant Bit“ (LSB) eines Datums gesendet wird. Analog wird beim Empfang eines Datums gemäß DORD entschieden, wie die Wertigkeiten der

[1] nach Burnett, Colin M.L. (2010)

empfangenen Bits bezüglich ihrer Reihenfolge zu interpretieren sind. Dabei bedeutet DORD=0, dass zuerst das MSB gesendet wird. Entsprechend gilt für DORD=1, dass zuerst das LSB gesendet wird.

SPI-Schnittstelle des ATmega 644s

Der ATmega 644 verfügt über ein internes Modul, welches die SPI-Kommunikation ermöglicht. Dieses Modul ist in der Lage, einen Großteil der zuvor vorgestellten Aspekte der SPI-Kommunikation selbstständig durchzuführen. Über verschiedene Konfigurationsregister können die im vorherigen Abschnitt erwähnten Parameter festgelegt werden. Abbildung 6.2 zeigt die Aufteilung des SPI Control Registers SPCR und deutet mit den entsprechenden Flag-Bezeichnern deren Funktion an. Durch Setzen der Bits SPE und MSTR wird die SPI-Kommunikation als Master aktiviert. Die Bedeutung der Bits DORD, CPOL und CPHA ist analog zu den gleichnamigen, oben beschriebenen Kommunikationsparametern. Durch die Bits SPR1 und SPR0 wird entsprechend Abbildung 6.2 ein Prescaler zur Clockfrequenz f_{osc} des Mikrocontrollers ausgewählt. Aus der Division von f_{osc} und dem Prescaler ergibt sich die Frequenz $f_{\texttt{CLK}}$ des Clock-Signals.

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

Abbildung 6.2: SPI Control Register (SPCR)

SPR1	SPR0	$f_{\texttt{CLK}}$
0	0	$f_{osc}/4$
0	1	$f_{osc}/16$
1	0	$f_{osc}/64$
1	1	$f_{osc}/128$

Tabelle 6.2: Semantik der Prescaler-Bits SPR1 und SPR0

Durch Schreiben eines Bytes in das SPI Data Register SPDR wird das Byte sofort vom Mikrocontroller an den selektierten Slave gesendet. Ein im Austausch zum ersten Byte vom Slave an den Mikrocontroller gesendetes Byte befindet sich nach Abschluss der Übertragung im SPDR Register.
Die in 6.3.2 erwähnten Leitungen MOSI, MISO und CLK werden vom Mikrocontroller internen SPI-Modul automatisch beschaltet und verwendet, sofern dieses korrekt konfiguriert wurde. Sobald die beidseitige Datenübertragung abgeschlossen ist, wird das SPI Interrupt Flag SPIF des SPI Status Registers SPSR gesetzt. Somit kann über das SPIF Bit überprüft werden, ob die Übertragung oder der Austausch eines Bytes abgeschlossen wurde.

Ein wichtiges Detail des SPI ist, dass eine Antwort des Slaves im Kontext immer einen Sendevorgang zeitlich nach hinten versetzt übertragen wird. Wird zum Beispiel vom Master eine Anfrage an den Slave gesendet, die einer Antwort bedarf, so befindet sich diese nicht direkt nach dem Sendevorgang des Befehls im `SPDR` Register. Es soll bei langen Verarbeitungszeiten von Befehlen auf Seiten des Slaves nicht auf die Bereitstellung der Antwort gewartet werden, sondern nach einer entsprechenden Zeit ein zweites Mal angefragt werden. Somit ist die Antwort des Slaves erst nach der erfolgreichen Übertragung eines weiteren Bytes im `SPDR` Registers zu finden. Der Unterschied zwischen einem Sendevorgang und einem Empfangsvorgang aus Sicht des Mikrocontrollers besteht darin, ob ein Byte gesendet und die Antwort des TLCD ignoriert oder eine Art Dummy-Byte (z.B. `0xFF`) übertragen und die Antwort des TLCD entsprechend weiterverarbeitet wird. Diese Ähnlichkeit der beiden Abläufe kann bei der Implementierung der entsprechenden Funktionalität ausgenutzt werden.

SPI-Schnittstelle des TLCDs

Das TLCD verfügt über ein SPI-Modul, welches alle vier SPI-Modi unterstützt. Um den SPI-Modus auf Seiten des Displays festzulegen, können die Pins `DORD`, `CPOL` und `CPHA` des TLCDs beschaltet werden. Diese Pins sind mit einem Pullup-Widerstand an VCC beschaltet. Deshalb ist nur der Low-Pegel, das heißt der Wert 0, aktiv anzulegen. Ist ein Konfigurationspin unbeschaltet, so ist der entsprechende Parameter mit dem Wert 1 konfiguriert. Um eine überflüssige Verwendung der Pins des Mikrocontrollers zu vermeiden, werden die Pins `DORD`, `CPOL` und `CPHA` des TLCDs nicht beschaltet.
Ferner signalisiert das TLCD über den Ausgang `SBUF`, dass Daten zur Abholung bereitstehen, indem der Pegel von high nach low gezogen wird.

Achtung

Da das TLCD eine bestimmte Zeit benötigt um Daten bereitzustellen, muss nach jeder Selektion des Slaves 6 µs gewartet werden.

Lernerfolgsfragen

- Welcher SPI-Modus ergibt sich aus der Pinbelegung aus Kapitel 6.5?

6.3.3 Small Protokoll

Im Folgenden wird das Datenübertragungsprotokoll *Small Protokoll* beschrieben. Dafür werden verschiedene Protokollbytes benötigt, deren konkrete Werte in 6.3 aufgelistet sind. Der Datenaustausch erfolgt in Form von Paketen, in denen die Nutzdaten in einem festen Rahmen (Frame) eingebettet sind. Ein Paket endet stets mit der Prüfsumme `bcc`. Diese berechnet sich aus der Summe der Werte aller Bytes Modulo 256, wobei auch das Start- und das `len`-Byte in der Summe enthalten sind. Das TLCD quittiert ein Paket mit dem Byte `<ACK>` bei erfolgreichem Empfang oder mit `<NAK>`, falls die Prüfsumme fehlerhaft oder der Empfangspuffer des TLCD übergelaufen ist. Wurde ein Paket mit `<NAK>` quittiert, so muss das komplette Paket erneut gesendet werden. Das Small Protokoll unterscheidet zwischen sechs verschiedenen Framevarianten, von denen zwei für diesen Versuch relevant sind. Entsprechend dieser zwei Framevarianten wird nachfolgend zwischen zwei Arten von Paketen unterschieden: Pakete zum Senden von Befehlen und Pakete zum Anfragen von Daten. Im Folgenden sind Pakete, die zum TLCD gesendet werden, durch einen Pfeil von links nach rechts (→) und Antworten des TLCDs durch einen Pfeil von rechts nach links (←) dargestellt.

Protokollbyte	Wert
<DC1>	0x11
<DC2>	0x12
<ACK>	0x06
<NAK>	0x15
<NUL>	0x00
<ESC>	0x1B
<A>	0x41
<C>	0x43
<D>	0x44
<E>	0x45

Protokollbyte	Wert
<F>	0x46
<G>	0x47
<H>	0x48
<L>	0x4C
<P>	0x50
<R>	0x52
<S>	0x53
<T>	0x54
<Z>	0x5A

Tabelle 6.3: Small Protokoll Protokollbytes

Befehle/Daten zum TLCD senden

Wie in Abbildung 6.3 dargestellt, beginnt ein Paket zum Senden von Befehlen oder Daten mit dem Byte `<DC1>`. Darauf folgt die Anzahl der in den Nutzdaten `data` enthaltenen Byte als numerischer Wert des Bytes `len`. Die Größe der darauf folgenden Nutzdaten ist folglich auf 255 Byte beschränkt.

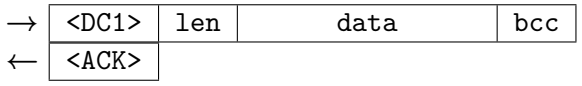

Abbildung 6.3: Small Protokoll Data-Frame

Inhalt des Sendepuffers des TLCDs anfordern

Das TLCD speichert in einem internen Sendepuffer Datenpakete, die an den Mikrocontroller versendet werden. Das Versenden der im Sendepuffer enthaltenen Pakete muss erst durch einen vom Mikrocontroller versendeten Befehl initiiert werden und erfolgt nicht automatisch. In diesem Zusammenhang verfügt das TLCD über den Ausgang `SBUF` (Sendbuffer-Indicator). An diesem Ausgang signalisiert das TLCD durch Anlegen eines Low-Pegels, dass im internen Sendepuffer des TLCDs Daten zur Abholung bereit stehen. Um die bereitstehenden Daten abzurufen, erwartet das TLCD ein Anfragepaket, wie es in Abbildung 6.4 dargestellt ist. Anstelle der Nutzdaten enthält ein Anfragepaket nur das Byte `<S>` und beginnt mit dem Byte `<DC2>`, um eine Verwechselung mit einem entsprechenden Befehls-/Datenpaket auszuschließen. Das TLCD antwortet nach erfolgreicher Quittierung mit einem wie im vorherigen Abschnitt beschrieben Datenpaket.

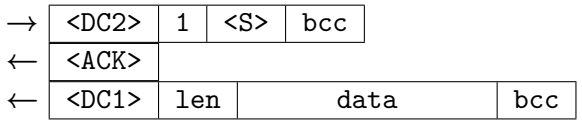

Abbildung 6.4: Small Protokoll Request-Frame

Befehle und Antworten des TLCDs

Nachdem im vorherigen Abschnitt Frames vorgestellt wurden, wird im Folgenden der Aufbau von Nutzdaten `data` behandelt. Nutzdaten können aus mehreren Segmenten gleicher Art bestehen. Bei Segmenten unterscheidet man zwischen zwei Arten: Befehle und Antworten. Einzelne Segmente beginnen grundsätzlich mit dem `<ESC>`-Byte. Befehle sind solche Nutzdaten, die vom Mikrocontroller an das TLCD gesendet werden (→), und Antworten sind Nutzdaten, die vom TLCD an den Mikrocontroller gesendet werden (←). Bei Befehlen folgen auf `<ESC>` zwei Byte, um den Typ des Befehls zu spezifizieren. Bei Antworten folgt nur ein Byte, um den Typ der Antwort zu spezifizieren. Das zweite Byte gibt hierbei die Größe der folgenden Parameter in Byte an. Je nach Art der Nutzdaten folgen unterschiedlich viele Bytes für Parameter.

Im Folgenden werden die in diesem Versuch verwendeten Arten von Nutzdaten vorgestellt. Da die maximale Auflösung des Displays 480x272 Pixel beträgt, wäre ein Byte zur Darstellung eines Punktes sowohl auf der X- als auch auf der Y-Achse zu gering. Daher haben X- und Y-Koordinaten, die im Folgenden mit `x` und `y` bezeichnet werden, jeweils eine Größe von zwei Byte. Die jeweils zwei Byte großen Koordinaten werden so gesendet und empfangen, dass das Low- vor dem High-Byte übertragen wird.

Touchbereich definieren

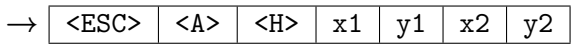

Abbildung 6.5: Touchbereich definieren

Um einen Touchbereich zu definieren, erwartet das TLCD, wie in Abbildung 6.5 dargestellt, den Befehlstyp `<A> <H>`. Die darauf folgenden Parameter definieren die Eck-Koordinaten eines Rechtecks, das dem gewünschten Touchbereich entspricht. Hierbei ist zu beachten, dass die Koordinaten `x1`, `y1`, `x2`, `y2` aus dem oben beschrieben Grund jeweils zwei Byte groß sind. Interaktionen mit dem TLCD (sog. Touchaktionen) innerhalb des definierten Bereichs werden fortan vom TLCD in Form eines Touch-Events als Antwort gesendet, dessen Format im nächsten Abschnitt beschrieben wird.

Touch-Event

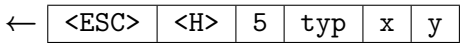

Abbildung 6.6: Touch-Event Antwort

Sobald das TLCD in einem zuvor definierten Bereich eine Touchaktion erkennt, generiert das TLCD eine Antwort vom Typ `<H>`, wie sie in Abbildung 6.6 dargestellt ist. Der Parameter `typ` hat beim Loslassen (`up`) den Wert `0`, beim Berühren (`down`) den Wert `1` und beim Wischen über das Display (`drag`) den Wert `2`. Darauf folgen die Koordinaten, an denen das Touch-Event ausgelöst wurde.

Punkt/Gerade Zeichnen

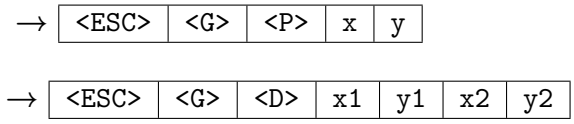

Abbildung 6.7: Punkt/Gerade Zeichnen

Die Befehle zum Zeichnen eines Punktes und einer Geraden werden durch das Zeichen `<G>` spezifiziert. Um einen Punkt zu zeichnen, folgt das Zeichen `<P>` und um eine Gerade

zu zeichnen, das Zeichen <D>. Wie in Abbildung 6.7 dargestellt, besteht die jeweilige Parameterliste aus den entsprechenden Koordinaten.

DrawBox

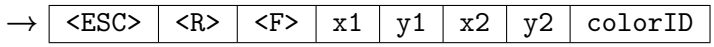

Abbildung 6.8: Box zeichnen

Abbildung 6.8 stellt den Befehl zum Zeichnen einer farbigen Box dar. Diese Box wird definiert durch Koordinaten (x1,y1) der oberen linken Ecke und Koordinaten (x2,y2) der unteren rechten Ecke. Das Paket wird mit einem Byte für die Farbauswahl abgeschlossen, siehe 6.3.3.

ChangePenSize

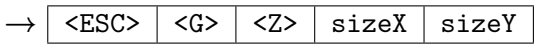

Abbildung 6.9: Stiftdicke ändern

Abbildung 6.9 stellt den Befehl zum Ändern der Stiftdicke beim Zeichnen z.B. von Linien dar. Die Argumente sizeX und sizeY geben die Breite des gemalten Punktes in x- und y-Richtung an. Es ist zu beachten, dass die Stiftdicke in jeder Dimension auf 15 begrenzt ist.

DefineColor

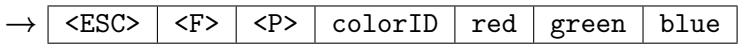

Abbildung 6.10: Farbe definieren

Abbildung 6.10 stellt den Befehl zum Definieren einer Farbe dar. Diese Farbe wird definiert durch ihren Rotanteil red, Grünanteil green und Blauanteil blue. Das Argument colorID dient zur späteren Referenz der definierten Farbe. Zum Beispiel kann beim Zeichnen einer farbigen Box die colorID einer definierten Farbe angegeben und somit als Hintergrundfarbe der Box verwendet werden. Es ist zu beachten, dass insgesamt 32 colorIDs zur Verfügung stehen, wobei 1 bis 16 bereits vordefinierte Farben enthalten. Alle colorIDs können mit neuen Farben belegt werden, ohne dass sich die Farbe von bereits Gezeichnetem verändert.

ChangeDrawColor

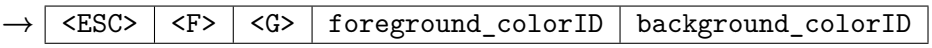

Abbildung 6.11: Stiftfarbe ändern

Abbildung 6.11 stellt den Befehl zum Ändern der Stiftfarbe beim Zeichnen z.B. von Linien dar. Das Argument colorID dient dem Auswählen einer zuvor definierten Farbe, siehe 6.3.3. Beim Zeichnen von z.B. Linien spielt die Hintergrundfarbe keine Rolle und kann mit dem Wert 255 unverändert gelassen werden.

DrawChar

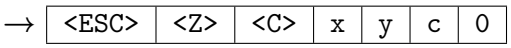

Abbildung 6.12: Character Zeichnen

Abbildung 6.12 stellt den Befehl zum Zeichnen eines Characters dar. Die Position des Characters wird mit den Koordinaten (`x1`,`y1`) festgelegt. Darauf folgt der eigentliche zu zeichnende Character und abgeschlossen wird das Paket mit einem Nullbyte.

Zeichenkette ausgeben

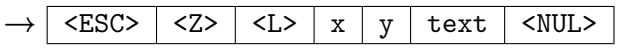

Abbildung 6.13: Zeichenkette ausgeben

Abbildung 6.13 stellt den Befehl zum Schreiben einer Zeichenkette `text` beginnend ab den Koordinaten (`x`,`y`) auf dem Display dar. Dieser Befehl beginnt mit der Kennung `<Z>` `<L>`. Die darauffolgende Zeichenkette wird mit dem Byte `<NUL>` abgeschlossen.

SetCursor

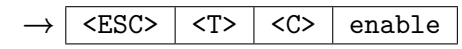

Abbildung 6.14: Cursor de-/aktivieren

Abbildung 6.14 stellt den Befehl zum Aktivieren und Deaktivieren des Cursors dar. Das Argument enable sorgt mit einem Wert von 0 für die Deaktivierung und mit einem Wert von 1 für die Aktivierung des Cursors.

ClearDisplay

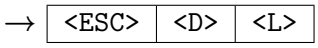

Abbildung 6.15: Display Löschen

Abbildung 6.15 stellt den Befehl zum Löschen des Displays dar.

Beispiel eines vollständigen Datenpakets

In Abbildung 6.16 ist beispielhaft ein komplettes Befehlspaket dargestellt, das zwei Touch-Events an den Koordinaten (120,170) und (121,171) enthält.

<DC1>	16	<ESC>	<H>	5	1	120	170	<ESC>	<H>	5	2	120	171	57

Abbildung 6.16: Datenpaket, das zwei Touch-Events enthält

Lernerfolgsfragen

- Warum ist in dem in 6.16 abgebildeten Datenpaket eine Länge von 16 Bytes hinterlegt, obwohl die Nutzdaten in der Abbildung nur 12 zu übertragene Elemente enthalten?
- Wie errechnet sich die Prüfsumme `bcc`?
- Ermöglicht die Prüfsumme `bcc` Verfahren zur Fehlerkorrektur?
- Wieso sendet das TLCD nicht selbstständig ein Datenpaket bei einem Touch-Event?
- Warum ist die Anzahl der Nutzdaten eines Pakets auf 255 Byte beschränkt?

6.4 Hausaufgaben

Implementieren Sie die in den nächsten Abschnitten beschriebenen Funktionalitäten. Halten Sie sich an die hier verwendeten Namen und Bezeichnungen für Variablen, Funktionen und Definitionen.
Lösen Sie alle hier vorgestellten Aufgaben zu Hause mithilfe von Microchip Studio 7 und schicken Sie die dabei erstellte und funktionsfähige Implementierung über Moodle ein. Ihre Abgabe soll dabei die `.atsln`-Datei, das Makefile, sowie den Unterordner mit den `.c/.h`-Dateien inklusive der `.xml/.cproj`-Dateien enthalten. Beachten Sie bei der

Bearbeitung der Aufgaben die angegebenen Hinweise zur Implementierung! Ihr Code muss ohne Fehler und ohne Warnungen kompilieren sowie die Testtasks mit aktivierten Compileroptimierungen bestehen. Wie Sie die Optimierungen einschalten ist dem begleitenden Dokument in Abschnitt *6.3.2 Probleme bei der Speicherüberwachung* zu entnehmen.

Achtung

Verwenden Sie zur Prüfung auf Warnungen den Befehl „Rebuild Solution“ im „Build“-Menü des Microchip Studio 7. Die übrigen in der grafischen Oberfläche angezeigten Buttons führen nur ein inkrementelles Kompilieren aus, d.h. es werden nur geänderte Dateien neu kompiliert. Warnungen und Fehlermeldungen in unveränderten Dateien werden dabei nicht ausgegeben.

Hinweis

Da die Ports B und D für andere Zwecke benötigt werden, muss in diesem Versuch von der Benutzung des externen SRAMs abgesehen werden. Somit ist es ratsam, den externen SRAM wie auch den dazugehörigen Heap aus der Liste Ihrer Speichermedien bzw. Heaps zu entfernen.

Die Implementierung dieses Versuchs erstreckt sich über die folgenden Dateien. In `tlcd_core.c` werden die im Grundlagenkapitel 6.3 beschriebenen Bitübertragungs- und Sicherungsschicht implementiert. Anschließend wird in der Datei `tlcd_parser.c` die Aufbereitung der vom TLCD empfangenen Daten realisiert. Funktionalität zur Verwaltung von Schaltflächen wird von der Datei `tlcd_button.c` bereitgestellt. Die benötigten Grafikbefehle werden in der Datei `tlcd_graphic.c` implementiert.

6.4.1 Bitübertragungs- und Sicherungsschicht

Initialisierung der SPI-Schnittstelle

Wie in Abschnitt 6.1 beschrieben, kommuniziert der Mikrocontroller als Master eines SPI-Busses mit dem TLCD. Die Initialisierung des Mikrocontrollers als Master soll in der Funktion `tlcd_init` implementiert werden. Es empfiehlt sich die in `tlcd_core.h` vordefinierten Konstanten und Makros zu benutzen. Passen Sie dazu zunächst die Register `DDRB` und `PORTB` entsprechend der Pinbelegung aus Kapitel 6.5 an. Konfigurieren Sie anschließend das Register `SPCR` entsprechend Abschnitt 6.3.2 so, dass beide Komponenten in einem einheitlichen SPI-Modus kommunizieren. Beachten Sie dabei den durch die Pinbelegung spezifizierten SPI-Modus des TLCDs.

Überlegen Sie sich dazu, welche Pins als Aus- und welche als Eingangspins genutzt werden und schalten Sie gegebenenfalls für Eingangspins die Pullup-Widerstände ein. Stellen Sie zudem sicher, dass nach Aufruf dieser Funktion das TLCD entsprechend Abschnitt 6.3.2 nicht als Slave selektiert ist. Dazu können Sie das vordefinierte Makro `tlcd_spi_disable` benutzen.
Das Register `SPCR` muss wie im Folgenden beschrieben konfiguriert werden. Aktivieren Sie die SPI-Kommunikation und konfigurieren Sie den Mikrocontroller als Master des SPI-Busses. Wählen Sie anschließend einen geeigneten Prescaler für die vom Master generierte Clockfrequenz $f_{\texttt{CLK}}$. Wie im Grundlagenkapitel 6.3.2 beschrieben, sind die Pins `DORD`, `CPOL` und `CPHA` des TLCDs nicht aktiv beschaltet. Überlegen Sie sich, welche Auswirkungen das für die Reihenfolge der Bitübertragung und den vom TLCD erwarteten SPI-Modus hat und konfigurieren Sie die entsprechenden Bits im Register `SPCR`.

Achtung

Das TLCD unterstützt eine durchgehende Datenübertragung bis zu einer **maximalen** Clockfrequenz von 200 kHz. Beachten Sie, dass der Mikrocontroller mit einer Frequenz f_{osc} von 20 MHz arbeitet. Bei der Wahl des Prescalers ist dieser Umstand zu berücksichtigen, sodass der Mikrocontroller als Master eine Clockfrequenz $f_{\texttt{CLK}}$ vorgibt, die vom TLCD unterstützt wird.

TLCD zurücksetzen

Implementieren Sie unter Berücksichtigung der Pinbelegung 6.5 die Funktion `tlcd_reset`. Diese Funktion soll das TLCD neustarten, indem für 10 ms ein Low-Pegel am `RESET`-Pin des TLCDs angelegt wird. Achten Sie darauf, dass nach Beenden der Funktion kein Low-Pegel mehr an dem `RESET`-Pin des TLCDs anliegen darf.

Senden und Empfangen von Bytes

Implementieren Sie die Funktionen `tlcd_writeByte` und `tlcd_readByte` zum Senden und Empfangen einzelner Bytes, wie im Grundlagenkapitel 6.3.2 beschrieben. Stellen Sie sicher, dass diese Funktionen nicht nur im Kontext des Betriebssystems SPOS atomar sind. Beachten Sie weiter, dass Sie zunächst das TLCD mithilfe des Makros `tlcd_spi_enable` als Slave selektieren und 6 µs warten müssen, bevor Sie ein Byte durch Schreiben in das `SPDR`-Register senden. Das Warten einer bestimmten Zeit in Mikrosekunden kann mit der Funktion `void _delay_us(double us)` aus `<util/delay.h>` realisiert werden. Ferner muss das TLCD vor Beendigung der beiden Funktionen jeweils wieder freigegeben werden.

Verwendung eines Eingabepuffers

Bevor die empfangenen Daten verarbeitet werden, sollen diese zunächst in einem intern verwalteten Eingabepuffer gespeichert werden. Legen Sie dazu in der Datei `tlcd_core.c` eine Variable vom Typ `tlcdBuffer` an.
`tlcdBuffer` ist eine Typdefinition für eine Struktur, die einen Ringpuffer auf einem Heap in SPOS verwaltet. Das Feld `data` speichert die Adresse und `size` die Größe des allozierten Speicherbereiches. Die Felder `head` und `tail` dienen als relative Zeiger in diesem Speicherbereich. Dabei soll jeweils beim Schreiben in den Puffer `head` und beim Lesen aus dem Puffer `tail` inkrementiert werden. Daten, die sich in Speicheradressen zwischen `head` und `tail` befinden, wurden demnach noch nicht gelesen und verarbeitet.

Puffer anlegen Initialisieren Sie in der Funktion `tlcd_initBuffer` den Eingabepuffer. Allozieren Sie dazu im internen Heap einen Speicherbereich in einer ausreichenden Größe, um Nutzdaten zu speichern. Speichern Sie die Adresse des allozierten Speicherbereichs im `data`-Feld und behandeln Sie auch mögliche Fehlerfälle. Speichern Sie anschließend die Größe des Puffers im Feld `size` und initialisieren Sie die Felder `head` und `tail` mit 0.

Puffer zurücksetzen Setzen Sie in der Funktion `tlcd_resetBuffer` den Eingabepuffer zurück, indem Sie `head` und `tail` auf 0 setzen.

Lesen und Schreiben Implementieren Sie zum Lesen und Schreiben auf den Eingabepuffer die Funktionen `tlcd_readNextBufferElement` und `tlcd_writeNextBufferElement`. Überlegen Sie welche Art von Sicherheitsüberprüfungen beim Lesen und Schreiben auf den Ringpuffer normalerweise notwendig sein könnten und ob diese in diesem Fall benötig werden.

Pufferzeiger abfragen Überprüfen Sie in der Funktion `tlcd_hasNextBufferElement`, ob sich noch nicht gelesene Daten im Puffer befinden.

Anfordern von Daten

Durch Aufrufen der Funktion `tlcd_requestData` soll der Sendepuffer des TLCDs, wie im Grundlagenkapitel 6.3.3 beschrieben, angefordert werden. Implementieren Sie das Senden eines Anfragepakets gemäß des Smart Protokolls mithilfe der Funktion `tlcd_writeByte`. Beachten Sie, dass sich die zu sendende Prüfsumme `bcc` auch auf das Start- und `len`-Byte bezieht. Wiederholen Sie das Senden des Anfragepakets solange, bis eine gültige Quittierung vom TLCD empfangen wurde.

Daten vom TLCD lesen

Die Funktion `tlcd_readData` soll vom TLCD angeforderte Datenpakete lesen, auf Konsistenz überprüfen und in den Eingabepuffer speichern. Dabei soll nicht das Frame in

den Eingabepuffer gespeichert werden, sondern nur die Nutzdaten. Treten Inkonsistenzen wie ein falsches Startbyte oder eine falsche Checksumme auf, soll der Eingabepuffer mittels `tlcd_resetBuffer` zurückgesetzt werden. Daher ist es wichtig, dass Sie beim Lesen der Daten mithilfe von `tlcd_readByte` die Checksumme berechnen und mit der übertragenen Checksumme vergleichen.

Senden von Befehlen

Implementieren Sie in der Funktion `tlcd_sendCommand` das Senden von Befehlen gemäß dem in Abschnitt 6.3.3 vorgestellten Small Protokoll. Der Sendevorgang muss atomar erfolgen, darf somit beispielsweise nicht durch Interrupts unterbrochen werden.
Nachdem Sie das Start- und `len`-Byte gesendet haben, durchlaufen Sie das Bytearray `cmd` und übertragen Sie die einzelnen Bytes, gefolgt von der Prüfsumme `bcc` des Pakets. Wiederholen Sie den kompletten Sendevorgang des Pakets solange, bis eine gültige Quittierung vom TLCD empfangen wurde.

6.4.2 Aufbereiten der empfangenen Daten

Die zu implementierenden Funktionen in diesem Kapitel beziehen sich auf die Datei `tlcd_parser.c`.

Pin-Change Interrupt Service Routine

Wie im Grundlagenkapitel 6.3.2 beschrieben, signalisiert das TLCD am Ausgang `SBUF` durch einen Pegelwechsel von high nach low, dass Daten zur Abholung bereitstehen. In der Funktion `tlcd_init` ist bereits die Konfiguration eines Pin-Change Interrupts für den `SBUF`-Ausgang des TLCDs enthalten. Dieser Pin-Change Interrupt ist so konfiguriert, dass bei einem Pegelwechsel an `SBUF` die zu implementierende Interrupt Service Routine `ISR(PCINT1_vect)` aufgerufen wird.
Fordern Sie solange den Sendepuffer des TLCDs an, bis am `SBUF`-Pin wieder ein High-Pegel anliegt. Nach jedem Anfordern des Sendepuffers sollen die empfangenen Daten durch die Funktionen `tlcd_readData` gelesen und durch `tlcd_parseInputBuffer` verarbeitet werden, deren Implementierung im folgenden Abschnitt behandelt wird.

Aufbereitung nach Small Protokoll

Die Funktion `tlcd_parseInputBuffer` soll die in 6.3.3 beschriebenen und zuvor gespeicherten Datenpakete verarbeiten. Beachten Sie, dass ein Paket grundsätzlich mehrere Nutzdaten-Pakete – in diesem Fall Antworten vom TLCD – enthalten kann. Durchlaufen Sie die Nutzdaten paketweise, indem Sie die folgenden Schritte solange wiederholen, wie Daten im Eingabepuffer zur Verfügung stehen. Verwenden Sie dazu die Funktion `tlcd_hasNextBufferElement`. Lesen Sie mithilfe der Funktion `tlcd_readNextBufferElement` für jedes Paket das Escape- und Typ-Byte. Anhand des Typ-Bytes rufen Sie eine entsprechende Funktion zum Verarbeiten

des Pakets auf: `tlcd_parseTouchEvent`, wenn ein `<H>`-Byte gesendet wurde, und `tlcd_parseUnknownEvent`, wenn ein anderes Typ-Byte gesendet wurde.

Aufbereitung eines Touch-Events Der in `tlcd_parser.h` vorgegebene Datentyp `TouchEvent` dient zur Verwaltung einer Touch-Event Antwort vom TLCD. Entsprechend Abschnitt 6.3.3 umfasst dieses `struct` neben den X- und Y-Koordinaten ein `enum TouchEventType` zur Speicherung des Typs des Touch-Events.
Extrahieren Sie in der Funktion `tlcd_parseTouchEvent` aus den Nutzdaten die Parameter des Touch-Events und speichern Sie diese in einer Variable vom Typ `TouchEvent` ab. Beachten Sie, dass die Koordinaten jeweils zwei Byte groß sind und jeweils das Low- vor dem High-Byte gesendet wird. Lassen Sie außerdem an geeigneter Stelle die Koordinaten und den Typ des Touch-Events auf dem LCD des Evaluationsboards ausgeben.

Aufbereitung eines unbekannten Events In der Funktion `tlcd_parseUnknownEvent` soll ein Paket übersprungen werden. Lesen Sie dazu die Länge des Pakets aus und verwerfen Sie entsprechend viele Elemente aus dem Puffer.

Interface für Anwenderprogramme

Erfolgreich empfangene Touch-Events sollen mithilfe eines Callbacks einem Anwendungsprogramm zur Verfügung gestellt werden können. Definieren Sie dafür einen Funktionstyp `EventCallback`, der ein Event vom Typ `TouchEvent` übergeben bekommt und keinen Rückgabewert besitzt.
Implementieren Sie außerdem eine Funktion `tlcd_setEventCallback`, welche die übergebene Callback-Funktion in einer globalen Variable speichert. Diese Funktion wird vom Anwenderprogramm benutzt, um den Callback für den späteren, automatischen Aufruf zu registrieren.
Die aktuell registrierte Callback-Funktion soll nun an geeigneter Stelle in `tlcd_parseTouchEvent` mit dem aufbereiteten Touch-Event als Argument aufgerufen werden. Bedenken Sie, dass das Zurücksetzen der aktuell vom TLCD Modul verwendeten Callback-Funktion durch einen entsprechenden Aufruf von `tlcd_setEventCallback` möglich sein soll und deshalb der Wert der globalen `EventCallback` Variable vor Dereferenzierung überprüft werden muss.

6.4.3 Implementierung der grafischen Steuerung

Implementieren Sie in der Datei `tlcd_graphic.c` die Befehle zur Definition eines Touchbereichs und zur Steuerung der grafischen Ausgabe auf dem TLCD. Verwenden Sie dazu die in Abschnitt 6.4.1 behandelte Funktion `tlcd_sendCommand`, und übergeben Sie dieser den zu sendenden Befehl als Byte-Array und dessen Länge. Die Länge eines Byte-Arrays kann mithilfe der Funktion `sizeof` ermittelt werden.
Implementiert werden sollen folgende Funktionen entsprechend ihres Namens wie im Grundlagenkapitel 6.3.3 beschrieben:

- `tlcd_defineTouchArea`

- `tlcd_drawPoint`
- `tlcd_drawLine`
- `tlcd_drawBox`
- `tlcd_changePenSize`
- `tlcd_defineColor`
- `tlcd_changeDrawColor`
- `tlcd_drawChar`
- `tlcd_setCursor`
- `tlcd_clearDisplay`

6.4.4 Benutzerinteraktion mittels Schaltflächen

Die zu implementierenden Funktionen in diesem Kapitel beziehen sich auf die Datei `tlcd_button.c` und `tlcd_button.h`.
Um Schaltflächen auf das TLCD zeichnen und auf eine Berührung dieser reagieren zu können, müssen die Koordinaten aller Schaltflächen mitgehalten und eine Kollisionsberechnung durchgeführt werden. Erstellen Sie dazu einen Datentyp `Button`, der alle wichtigen Merkmale einer Schaltfläche kapselt. Dazu gehört die Position, die Größe, die ColorID, ein Character als Aufschrift und ein Button-Code, der dem Anwendungsprogramm bei Berührung der Schaltfläche zwecks Identifikation übergeben wird.
Legen Sie außerdem einen globalen Speicher für Buttons von der Größe des vorgegebenen Defines `MAX_BUTTONS` an.
Auch bei Schaltflächen bietet sich die Verwendung von Callbacks an. Implementieren Sie dazu die Funktion `tlcd_setButtonCallback` und den Funktionstyp `ButtonCallback`. `tlcd_setButtonCallback` funktioniert analog zu `tlcd_setEventCallback`, der Funktionstyp `ButtonCallback` soll jedoch einen Button-Code und eine Position auf dem TLCD übergeben bekommen. Buttons müssen allerdings noch hinzugefügt, gezeichnet und bei Berührung ihr Down-Code dem Button-Callback übergeben werden. Dazu sollen folgende Funktionen umgesetzt werden:

- `void tlcd_addButtonWithChar(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t color, uint8_t downCode, char c)`
- `void tlcd_drawButtons(void)`
- `uint8_t tlcd_handleButtons(TouchEvent event)`

`tlcd_addButtonWithChar` legt einen Button mit den übergebenen Merkmalen im globalen Speicher für Buttons ab.
Die Funktion `tlcd_drawButtons` durchläuft alle hinzugefügten Buttons und malt diese inklusive der Aufschrift und Farbe mithilfe der Funktionen `tlcd_drawBox` und

`tlcd_drawChar` auf das TLCD. Bedenken Sie, dass Buttons keine Aufschrift oder Farbe haben müssen und es möglich sein soll, Buttons ohne diese zu definieren.
`tlcd_handleButtons` ist für die Kollisionsberechnung und das Aufrufen des registrierten Button-Callbacks zuständig. Dazu muss `tlcd_handleButtons` in der Funktion `tlcd_parseTouchEvent` an geeigneter Stelle mit dem aktuellen Touch-Event als Argument aufgerufen werden. In `tlcd_handleButtons` muss überprüft werden, ob die Koordinaten des übergebenen Touch-Events innerhalb einer hinzugefügten Schaltfläche liegen und es sich um ein Drücken auf die Oberfläche des TLCDs handelt. Wenn das der Fall ist, soll der registrierte Button-Callback mit entsprechenden Argumenten aufgerufen werden. Über den booleschen Rückgabewert signalisiert die Funktion, ob das Touch-Event innerhalb einer Schaltfläche liegt und behandelt wurde oder nicht.

6.4.5 Implementierung einer Paint Anwendung

Softwareinterpolation von Touch-Eingaben

Eine triviale Methode auf das TLCD zu malen ist das unmittelbare Zeichnen eines Punkts an die dem EventCallback übergebenen Koordinaten. Allerdings fällt bei diesem Ansatz insbesondere beim schnellen Wischen über das Display auf, dass Abstände zwischen den Punkten auftreten. Dies lässt sich auf die begrenzte Geschwindigkeit der Hardware zurückführen. Um dieses Problem zu umgehen, soll eine verbesserte Methode verwendet werden, die die Koordinaten zweier Touch-Events durch eine Linie verbindet. So werden die nicht erfassten Pixel zwischen zwei Punkten durch eine Linie approximiert. Wie in Abschnitt 6.3.3 beschrieben, liefert ein Touch-Event neben den Koordinaten die Typ-Information, ob das Display losgelassen (`up`), berührt (`down`) oder gestreift (`drag`) wird. Es ist sinnvoll bei jedem Touch-Event den Typ zu untersuchen und nur beim Wischen über das Display eine Linie zu zeichnen. Andernfalls würde ein Punkt beim Loslassen mit dem ersten Punkt der darauffolgenden Berührung und beispielsweise zwei parallele Linien immer durch eine Diagonale verbunden werden.

Anwendungsprogramm

Als Anwendungsprogramm soll eine Paint Anwendung implementiert werden, die das farbige Malen von Punkten und Linien verschiedener Größe auf dem TLCD ermöglicht. Für die Auswahl der Stiftfarbe soll auf der Oberfläche des TLCDs ein Farbverlauf dargestellt werden, der den kompletten unteren Rand des Displays einnimmt. Des Weiteren müssen Buttons für das Erhöhen und Verringern der Stiftdicke erstellt werden. Zusätzlich soll die Funktionalität eines „Radiergummis“ zur Verfügung gestellt werden. Dabei wird die Stiftfarbe auf die Hintergrundfarbe des TLCDs gesetzt und die Stiftgröße auf 15 festgelegt. Beachten Sie, dass die Hintergrundfarbe standardmäßig die ColorID `1` hat. Der Rest der Displayoberfläche dient zum eigentlichen Malen. Platzieren Sie die Buttons in der oberen rechten Ecke des TLCDs.

Abbildung 6.17 zeigt schematisch, wie ein Farbverlauf durch RGB-Werten realisiert werden kann. Es bietet sich an, den Farbverlauf mithilfe eines einzelnen Buttons zu reali-

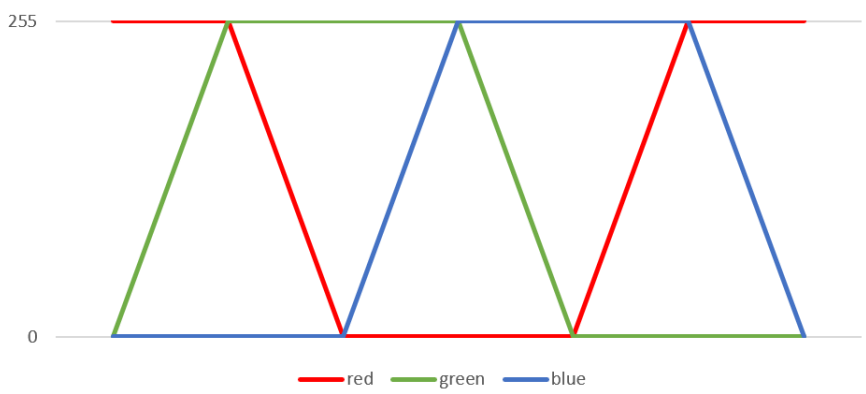

Abbildung 6.17: Beispielhafte Funktion der Farbanteile eines Farbverlaufs

sieren, der den gesamten Farbverlauf abdeckt. Dünne, vertikale, farbige Linien erzeugen den Eindruck eines Farbverlaufs. Wird der Button über dem Farbverlauf gedrückt, kann über den dem Button-Callback übergebenen Down-Code und die Koordinaten entschieden werden, welche Farbe vom Benutzer ausgewählt wurde.

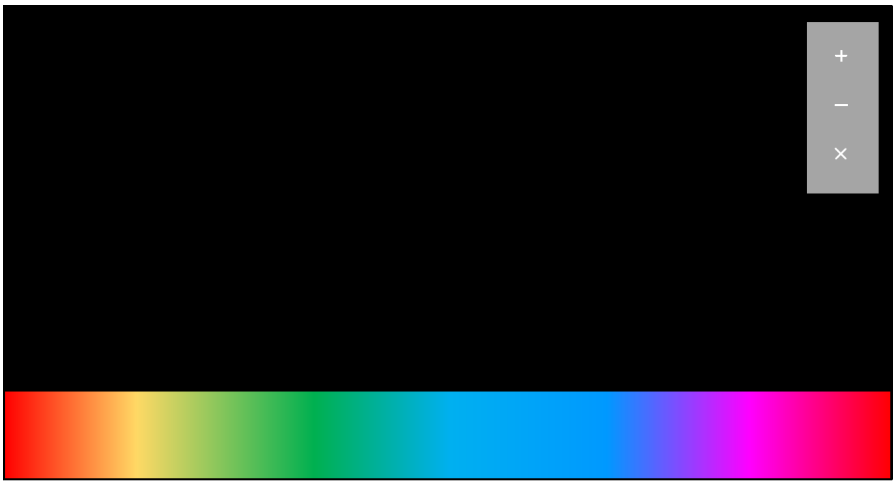

Abbildung 6.18: Paint Anwendung

Denken Sie daran das TLCD zu initialisieren, einen Touchbereich zu definieren und Callbacks für Buttons und Events zu implementieren. Das Programm soll in einer Endlosschleife laufen, auf Touch-Events reagieren und beim Drücken eines auf dem TLCD-Shield befindlichen Hardware Buttons den Inhalt des TLCDs löschen. Abbildung 6.18 zeigt eine beispielhafte Implementierung der Paint Anwendung.

6.4.6 Zusammenfassung

Die zu spezifizierenden Typen und Funktionen im Überblick:

- ☐ `tlcd_core:`
  - ☐ `void tlcd_init()`
  - ☐ `void tlcd_reset()`
  - ☐ `uint8_t tlcd_writeByte(uint8_t byte)`
  - ☐ `uint8_t tlcd_readByte()`
  - ☐ `inputBuffer`
  - ☐ `void tlcd_initBuffer()`
  - ☐ `void tlcd_resetBuffer()`
  - ☐ `void tlcd_writeNextBufferElement(uint8_t byte)`
  - ☐ `uint8_t tlcd_readNextBufferElement()`
  - ☐ `uint8_t tlcd_hasNextBufferElement()`
  - ☐ `void tlcd_requestData()`
  - ☐ `void tlcd_readData()`
  - ☐ `void tlcd_sendCommand(const unsigned char* cmd, uint8_t len)`
- ☐ `tlcd_parser:`
  - ☐ `ISR(PCINT1_vect)`
  - ☐ `void tlcd_parseInputBuffer()`
  - ☐ `void tlcd_parseTouchEvent()`
  - ☐ `void tlcd_parseUnknownEvent()`
  - ☐ `typedef TouchEvent`
  - ☐ `typedef EventCallback`
  - ☐ `void setEventCallback(EventCallback* callback)`
- ☐ `tlcd_graphic:`
  - ☐ `void tlcd_defineTouchArea(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)`
  - ☐ `void tlcd_drawPoint(uint16_t x1, uint16_t y1)`
  - ☐ `void tlcd_drawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)`
  - ☐ `void tlcd_drawBox(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t color)`
  - ☐ `void tlcd_changePenSize(uint8_t size)`

- ☐ `void tlcd_defineColor(uint8_t colorID, Color color)`
- ☐ `void tlcd_changeDrawColor(uint8_t color)`
- ☐ `void tlcd_drawChar(uint16_t x1, uint16_t y1, char c)`
- ☐ `void tlcd_setCursor(uint8_t enabled)`
- ☐ `void tlcd_clearDisplay()`

☐ `tlcd_button:`

- ☐ `typedef Button`
- ☐ `typedef ButtonCallback`
- ☐ `buttonBuffer`
- ☐ `void tlcd_setButtonCallback(ButtonCallback* callback)`
- ☐ `void tlcd_addButtonWithChar(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t color, uint8_t downCode, char c)`
- ☐ `void tlcd_drawButtons()`
- ☐ `uint8_t tlcd_handleButtons(TouchEvent event)`

☐ `Paint Anwendung Funktionalitäten:`

- ☐ Stiftdicke erhöhen, verringern
- ☐ Radiergummi
- ☐ Farbverlauf
- ☐ Kombination oben genannter Elemente zum farbigen Malen verschieden breiter Punkte und Linien auf dem Rest der TLCD Oberfläche

6.5 Pinbelegungen

Port	Pin	Belegung
Port A	A0	LCD Pin 1 (D4)
	A1	LCD Pin 2 (D5)
	A2	LCD Pin 3 (D6)
	A3	LCD Pin 4 (D7)
	A4	LCD Pin 5 (RS)
	A5	LCD Pin 6 (EN)
	A6	LCD Pin 7 (RW)
	A7	frei
Port B	B0	frei
	B1	frei
	B2	TLCD Pin 20: Sendbuffer Indicator
	B3	TLCD Pin 5: Reset
	B4	TLCD Pin 6: \CS
	B5	TLCD Pin 7: MOSI
	B6	TLCD Pin 8: MISO
	B7	TLCD Pin 9: CLK
Port C	C0	Button 1: Enter
	C1	Button 2: Down
	C2	Reserviert für JTAG
	C3	Reserviert für JTAG
	C4	Reserviert für JTAG
	C5	Reserviert für JTAG
	C6	Button 3: Up
	C7	Button 4: ESC
Port D	D0	frei
	D1	frei
	D2	frei
	D3	frei
	D4	frei
	D5	frei
	D6	frei
	D7	frei
	VCC	TLCD Pin 2: VCC
	GND	TLCD Pin 1: GND
		TLCD Pin 11: SPIMO
		TLCD Pin 13: DPOM

Pinbelegung für Versuch 6 (*Touch-LCD*).