1 Screens und Grafiklayout

Voraussetzungen

Ziel ist eine weitestgehend flackerfreie Darstellung aller Grafikelemente auf dem LCD. Die Dauer für die Darstellung eines Grafikelements ist weitestgehend proportional zur unterbrechungsfrei beschreibbaren rechteckigen Grundfläche plus eine konstante Zeit für die Vorbereitung dieser Fläche. So sind zusammenhängende Rechtecke (z.B. vorherberechenbare Pixelmuster aus Vorder- und Hintergrundfarbe) schneller zu füllen als Rechtecke mit transparentem Hintergrund, da letztere pixelweise geschrieben werden müssen.

Damit muß das Bestreben in die Richtung gehen, immer nur die Grafikelemente zu aktualisieren, die benötigt werden.

Es ist möglich, Grafikelemente pixelweise zu überschreiben, aber nicht zu löschen. Werden verdeckende Grafikelemente durch Überschreiben mit der Hintergrundfarbe entfernt, müssen evtl. dahinterliegende Elemente wieder separat rekonstruiert werden.

Grafik-Layout

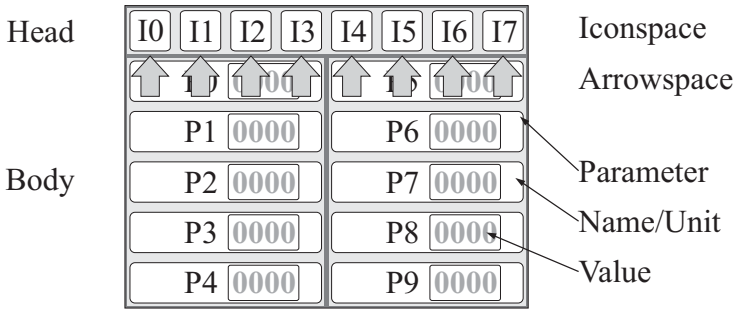

Abbildung 1: Grafik-Layout: Für Piktogramme sind 8, für Texte und Werte 18 Positionen (*slots*) vorhanden. Die oberste Parameter-Reihe (Slots P0 und P5) kann durch animierte Pfeile verdeckt werden.

Das (vorläufige) Grafik-Layout ist in Abbildung 1 dargestellt. Der Kopf (*head*) bietet Platz für acht Piktogramme. Piktogramme sind sehr unterschiedlich, bestehen aus mehreren halbtransparenten Layern und benötigen daher für ihre Größe verhältnismäßig viel Zeit zur Darstellung. Sie stellen jeweils wenige diskrete Werte (z.B. Status) dar.

Der Körper (*body*) bietet Platz für zehn tabellarisch dargestellte Werte, bestehend aus Name, Wert und Einheit. Name und Einheit ändern sich praktisch nie, der Wert oft. Die oberste Zeile des Körpers kann durch Pfeile verdeckt werden.

Animierte Pfeile können wichtige Piktogramme gesondert hervorheben.

Insgesamt existiert damit eine begrenzte Zahl an Grafikelementen (siehe Tabelle 1 auf der nächsten Seite).

Die Anzahl der Piktogramme kann noch zu- und die Anzahl der Textfelder abnehmen. Ebenfalls sind noch wenig preudoproportionale Anzeigen möglich. Die Anzahl der Grafikelemente wird allerdings unter 100 bleiben.

Hintergrund	*background*	1
Hintergrund Kopf	*head background*	1
Hintergrund Körper	*body background*	1
Piktogramme	*icons*	8
Animierte Pfeile	*animated arrows*	8
Name + Einheit	*name, unit*	18
Wert	*value*	18
Insgesamt	*total*	55

Tabelle 1: Auflistung der Grafikelemente

1.1 IST-Zustand

Abbildung 2: *Screenshot* des IST-Zustands

Die Werte entstehen aus unterschiedlichen Datentypen, die meisten sind jedoch vom Typ `float32` (Body-Bereich) oder `enum` (Piktogramme). Jede Darstellung resultiert aus einem Wert in einer großen (lokalen) *struct*-Variable. Einzige Ausnahme ist die globale Millisekundenzählvariable.

Pro Grafikelement existiert eine eigene Darstellungsfunktion `drawXX()`. Das ist deshalb notwendig, weil sich die Piktogramme in der Anzahl der Layer und Farben massiv unterscheidet und die Zahlenwerte unterschiedliche Quelldatentypen haben und Rundungswerte benötigen.

Eine große Funktion wird mit aktuellen Werten aufgerufen, speichert die alten Werte, vergleicht Änderungen und ruft die einzelnen Grafik-Zeichenfunktionen nach Bedarf auf. In den Grafik-Zeichenfunktionen wird zusätzlich entschieden, ob ein Update nötig ist.

1.2 Entwurf 1

① Gegeben: Alle Daten liegen in einem großen lokalen *struct* einer aufrufenden Funktion vor. Die aufrufende Funktion kann zudem entscheiden, welche Daten überhaupt dargestellt werden.

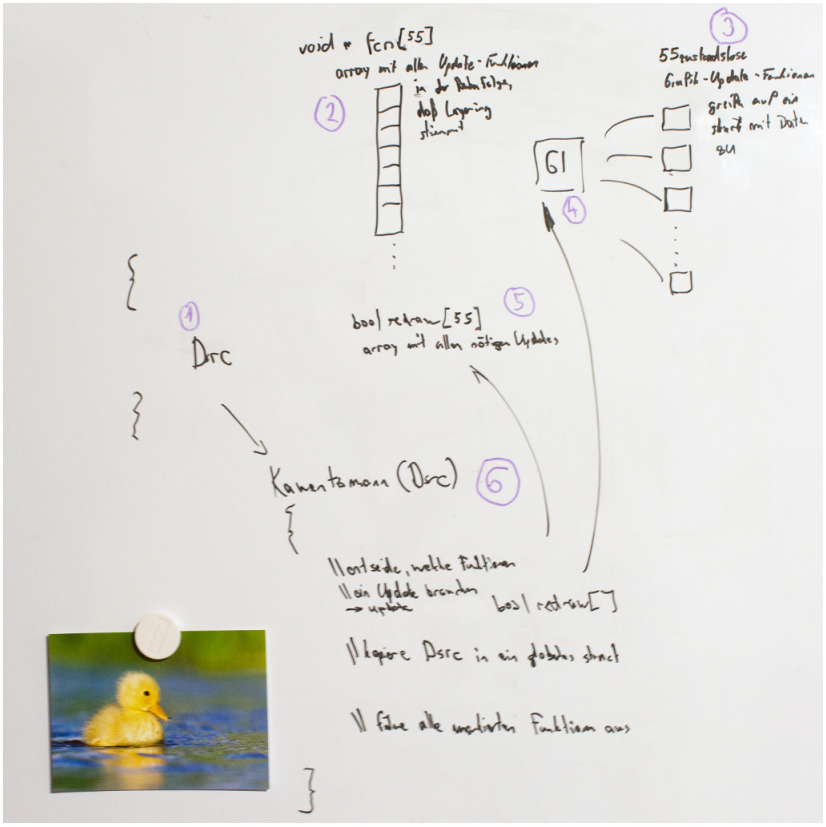

Abbildung 3: Entwurf 1 des Grafiksystems

② Eine konstantes *array* Funktionszeigern aller *update*-Funktionen in der Reihenfolge, daß das *layering* stimmt.

③ 55 zustandslose Grafik-Update-Funktionen (*void–void*), greifen auf ein globales *struct* mit Daten zu

④ Globales *struct* mit darzustellenden Daten.

⑤ `bool redraw[55]` Globales *array* mit der Information, welche Grafik-*updated* vollzogen werden müssen.

⑥ `void Kaventzmann(struct Dsrc_s Dsrc}` Dicke Funktion, der alle darzustellenden Daten übergeben werden.

- Entscheidet, welche Funktionen ein Update benötigen → `bool redraw[55]`
- kopiert `Dsrc` in ein globales *struct*
- Führt alle markierten Grafik-*update*-Funktionen aus

Variante 1

Anstelle `bool redraw[55]` wird eine 64 Bit breite Maske verwendet. Das erleichtert die Markierung mehrere aktiver oder inaktiver Elemente. (Kann man 64 Bit breite `enum`s verwenden, oder muß man hier mit `define`s hantieren?)