

Institut für Informatik
Lehrstuhl für Theoretische Informatik
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



Praktikumsbericht

**Nibbles auf dem AURIX
Application Kit TC234 TFT**

Automotive Praktikum

Clemens Niemeyer

Informatik Master

Aufgabensteller: Dr. Gordon Cichon
Abgabetermin: 23. März 2015

Vom AURIX zur Handheld-Konsole

Dieser Praktikumsbericht dokumentiert die Entstehung einer Handheld-Konsole, die im Rahmen des Automotive Blockpraktikums an der LMU im Wintersemester 2014/15 entstanden ist. Inhalt des Praktikums war die Einführung in die Programmierung eines AURIX Application Kit TC234 TFT von Infineon (AURIX). Auf diesem Developer Board ist ein Infineon Tricore verbaut, der hauptsächlich im Automotive Umfeld gebraucht wird. Ziel des Praktikums war es, ein eigenes Projekt mit dem Aurix zu realisieren, das Ergebnis ist in Abbildung 1 zu sehen. Auf der Handheld-Konsole läuft ein Klon des MS-DOS Klassikers Nibbles [Rad91]. Die Konsole kann mithilfe eines an der Seite befestigten An-Knopfes gestartet werden, gespielt werden kann allerdings nur, wenn sich ein Spiel im dafür vorgesehenen Schlitz befindet. Die Steuerung der Spielfigur – Sammy – erfolgt über zwei angeschlossene Potentiometer. Sammy ist ein Computer-Wurm und ernährt sich von Zahlen. Jedesmal, wenn er eine Zahl verspeist, wächst und wird er schneller. Das Spiel bietet fünf abwechslungsreiche Level und eine eingehende Spielmelodie. Ist das Spiel gewonnen oder verloren, wird die erreichte Punktzahl angezeigt. Eine neue Partie kann über den gelben Knopf gestartet werden.



Abbildung 1: Infineon Handheld-Konsole

Inhaltsverzeichnis

1 Komponenten	1
1.1 Druckknopf - IO Pin	1
1.2 Potentiometer - A/D-Wandler	2
1.3 Lautsprecher - PWM	3
1.4 Darstellung - TFT Bildschirm	6
2 Zusammensetzung	7
2.1 Spezifikationen	7
2.2 Gehäuse	8
2.3 Fehlstart	8
2.4 Steuerung	9
2.5 Musik	9
3 AURIX Handheld 2.0	11
Abbildungsverzeichnis	13
Tabellenverzeichnis	13
Literaturverzeichnis	13

Kapitel 1

Komponenten

In diesem Kapitel werden die Verwendeten Komponenten vorgestellt und deren Funktion erläutert. Zudem wird Anhand von ausgewählten Programmabschnitten gezeigt, wie das AURIX mit den Bauteilen kommuniziert und diese initialisiert werden.

1.1 Druckknopf - IO Pin

Bei einem Druckknopf handelt es sich um ein Bauteil mit zwei Eingängen. Während der Knopf gedrückt wird, sind beide Eingänge miteinander verbunden. Um einen Druckknopf zu verwenden benötigt man einen IO Pin und eine positive Versorgungsspannung (VCC), hier 3.3 V. Der Knopf wird mit dem Pin und der VCC verbunden. Wird der Knopf betätigt, liegt am Pin eine Spannung an und es kann eine logische Eins gelesen werden. Wichtig dabei ist, dass der Pin an einen Pull-Down-Resistor angeschlossen ist. Dieser stellt sicher, dass bei geringeren Spannungen als der verwendeten VCC der Pin nicht auf logisch Eins gesetzt wird.

Das AURIX stellt 60 IO Pins zur Verfügung [AG14, S. 5-1]. In Abbildung 1.1 ist dargestellt, wie auf den Pin 00.0 zugegriffen werden kann. Ein externer Pull-Down-Resistor ist nicht nötig, es muss lediglich der intern verbaute Widerstand aktiviert werden.

```
#include <tc23xa/IfxPort_reg.h>
#include <tc23xa/IfxPort_bf.h>

static Ifx_P * const port00 = (Ifx_P *)&MODULE_P00;

int main() {
    while(1){
        // init Pull-Down-Resistor
        port00->IOCR0.B.PC0 = 0x1;
        // read input of Pin 00.0
        if(P00_IN.B.P0){
            LED_TOGGLE(0);
        }
    }
    return 0;
}
```

Abbildung 1.1: Der Knopf ist mit Pin 00.0 und VCC verbunden und kann dazu verwendet werden, LED0 an und auszuschalten.

1.2 Potentiometer - A/D-Wandler

Eine weitere Möglichkeit mit dem AURIX zu kommunizieren, kann mit einem Potentiometer realisiert werden. Ein Potentiometer ist ein variabler Widerstand, der eine angelegte Spannung stufenlos teilen kann. Die eingestellte Spannung kann mithilfe eines Pins mit angeschlossenem Analog/Digital-Wandler (AD-Wandler) eingelesen werden. Das AURIX stellt zwölf analoge Eingänge zur Verfügung, AN[0-17] [AG14, S. 5-1], die das Eingangssignal in eine 12, 10 oder 8-Bit Binärzahl umwandeln [AG13, S. 27-95].

Im Praktikum wurde eine ADCDemo zur Verfügung gestellt. In dieser Demo wird ein Analoger Pin in Betrieb genommen und der Wert des Potentiometers ausgelesen. In Abbildung 1.2 wird dargestellt wie der Wert eines weiteren Potentiometer ausgelesen werden kann. Es werden nur die Änderungen in der *ADCDe-mo/src/main.c* angegeben wobei die zu ersetzenen Zeilen als Kommentare mit *--* gekennzeichnet sind. Weitere Erläuterungen zum untenstehenden Programmcode:

(1) Im Register BRSSEL0 werden die Kanäle der Gruppe 0 definiert, die eingelesen werden sollen. Da Kanal 0 und 1 benutzt werden sollen, muss das Register auf 11_B gesetzt werden [AG13, S. 27-78].

(2) Setzen des RESTBS auf 0_B hat zur Folge, dass das Ergebnis in einem lokalen Register gespeichert wird. Ist RESTBS auf 1_B gesetzt, wird das Ergebnis in das globale Register der Gruppe geschrieben. Schreiben beide Kanäle ins globale Register, werden die Werte des zuerst ausgelesenen Kanals überschrieben [AG13, S. 27-92].

(3) In (2) wurde festgelegt, das ein lokales Register verwendet werden soll. Der Standard-Wert ist dabei das lokale Register 0. Um ein Überschreiben im lokalen

Register zu verhindern, wird Kanal 1 dem lokalen Register 1 zugewiesen [AG13, S. 27-38]. Die Werte des Kanals 0 werden weiterhin im lokalen Register 0 gespeichert.

(4) Am Schluss wird die Umwandlungsanfrage für alle definierten Kanäle aktiviert [AG13, S. 27-76].

(5) Das Abfragen der Werte erfolgt Analog zur ADCDemo mit dem Unterschied, dass die entsprechend (3) festgelegten lokalen Register ausgelesen werden.

```
int main() {
    [ . . . ]
    //—VADC_BRSSEL0.U = 0x1;           ///(1)
    VADC_BRSSEL0.U = 0b11;

    //—VADC_G0CHCTR0.B.RESTBS = 1;     ///(2)
    VADC_G0CHCTR0.B.RESTBS = 0;
    VADC_G0CHCTR1.B.RESTBS = 0;

    VADC_G0CHCTR1.B.RESREG = 0x1;      ///(3)

    //—VADC_BRSMR.U = 0x1;            ///(4)
    VADC_BRSMR.U = 0x17;

    [ . . . ]
    while(1){
        VADC_BRSMR.B.LDEV = 1;          ///(5)
        result0 = VADC_G0RES0.B.RESULT;
        result1 = VADC_G0RES1.B.RESULT;
        [ . . . ]
    }
    return 0;
}
```

Abbildung 1.2: Die Änderungen von einem auf zwei analoge Eingangs-signale

1.3 Lautsprecher - PWM

Ein Lautsprecher besteht - stark vereinfacht - aus einer Membran, einem Ringmagneten und einem Aktuator. Der Aktuator besteht aus einer um einen Kolben gewickelten Spule. Er ist an der Membran befestigt und ragt ins Innere des Magneten. Wird die Spule von Strom durchflossen, entsteht ein weiteres Magnetfeld im Ringmagneten. Die daraus resultierende Lorentzkraft schiebt den Kolben, inklusive Membran, nach außen. Hört der Strom auf zu fließen, springt die Membran wieder in die Ausgangsposition zurück [Gio09, S. 673]. Um einen Ton zu erzeugen, wird ein im hörbaren Frequenzbereich (20–20 000 Hz) getaktetes Signal benötigt [Gia10, S. 562]. Solche Signale können mit der sogenannten Puls-Breite-

Modulation (PWM) erzeugt werden. Ein hardware gesteuertes PWM-Signal mit Hilfe eines GTM-TOM-Timers zu erstellen, war nicht möglich, da die benötigten Low-Level-Driver (iLLD) nicht unter der Hightec Eclipse umgebung zum laufen gebracht werden konnten. Stattdessen wurde der schon vorhandene Timer genutzt und ein Software gesteuertes PWM-Signal erzeugt. In Abbildung 1.3 ist der entsprechende Programmabschnitt aufgeführt.

```

#define HZ 1600 /* timer event rate [Hz] */
[...] // setup timer as in provided demo

Melody melody = {
    .length = 9,
    .freqs = {8,4,3,6,6,8,5,7,5},
    .durations = {800,400,200,200,200,800,400,400,100}
}

int freq;
int duration = 0;
int toneCount = 0;
int freqCount = 0;

void playMusic(){
    ++freqCount;
    if(duration > 0){
        --duration;
        if(freqCount >= freq){
            freqCount = 0;
            PORT33_TOGGLE(1);
        }
    } else{
        freqCount = 0;
        if(toneCount < melody.length){
            duration = melody.durations[toneCount];
            freq = melody.freqs[toneCount];
            toneCount++;
        } else{
            toneCount = 0;
        }
    }
}

int main() {
    while(1){
        if(event_flag){
            playMusic();
        }
    } return 0;
}

```

Abbildung 1.3: Implementierung eines Software PWM Signals an Pin33.1

1.4 Darstellung - TFT Bildschirm

Für den integrierten 320x240px TFT Bildschirm, wurden einige Hilfsfunktionen zur Erleichterung der Darstellung des Spiels geschrieben. Diese verwenden jedoch ausschließlich Funktionen der schon vorhandenen Bibliothek *display.h*. Die meistbenutzte Funktion ist die in Abbildung 1.4 gezeigte *writeSquare(...)*. Da alle Bausteine des Spielfelds Quadrate mit Seitenlänge fünf sind.

```
#include "display.h"

void
writeSquare(int w, int h, int x, int y,
            uint8_t color) {
    int i, k;
    if(y+h>239) y= 240-h;
    if(x+w>319) x= 320-w;
    for (i=0; i<h; i++) {
        for (k=0; k<w; k++) {
            writePixel(x+k, y+i, color);
        }
    }
}
```

Abbildung 1.4: Funktion zum schreiben eines Vierecks mit Breite *w*, Höhe *h*, Position (*x,y*) und Farbe *color* auf dem TFT Bildschirm des AURIX

Eine weitere Nennenswerte Funktion aus dem Bereich der Visualisierung ist die Funktion *drawWall()*. Die Funktion zeichnet für jedes Level die Wände. Der Trick dabei ist, dass nur die Wände im linken unteren Quadranten definiert werden und um die Mitte punktgespiegelt. Dadurch haben die verwendeten Arrays zur definition eines Levels nur ein viertel so groß wie ohne Punktspiegelung.

Kapitel 2

Zusammensetzung

In diesem Kapitel wird beschrieben wie die Komponenten aus Kapitel 1 zu einer Handheld-Konsole zusammengesetzt wurden auf der Nibbles gespielt werden kann. Dabei werden zuerst die Spezifikationen der Konsole aufgezählt und anschließend die daraus resultierenden Probleme und deren Lösungen genauer beleuchtet. Die Komponenten sind wie in Abbildung 2.1 angebracht.

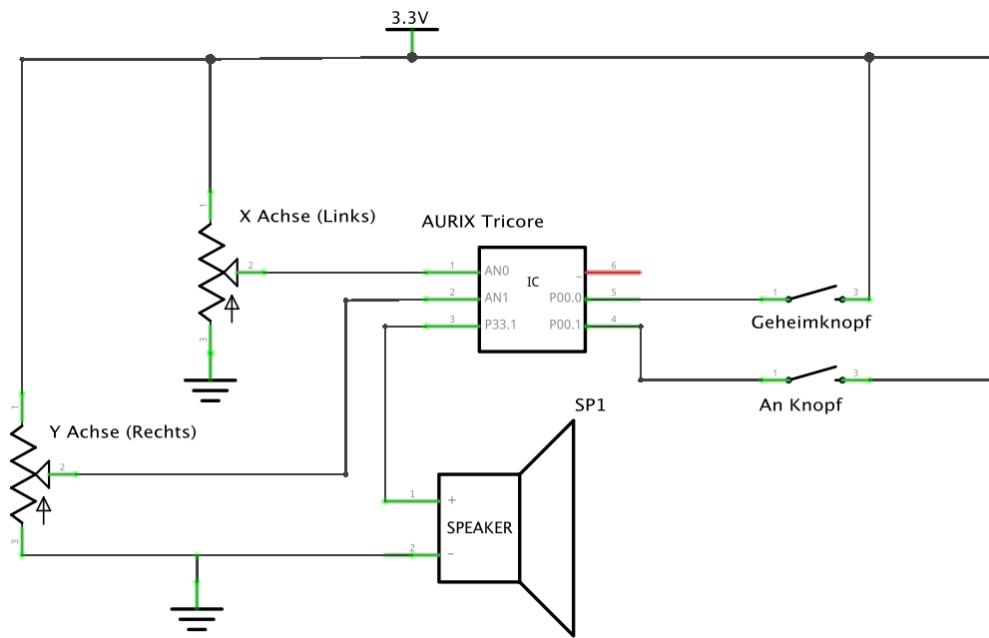


Abbildung 2.1: Schaltplan der Infineon Handheld-Konsole

2.1 Spezifikationen

Das Gehäuse wird mithilfe eines 3D-Druckers angefertigt. Das AURIX, die Kabel sowie der Lautsprecher befinden sich im inneren des Gehäuses. Der Bildschirm kann durch ein Loch auf der oberen Seite betrachtet werden. Die zwei Potentiometer sind links und rechts vom Bildschirm angebracht. An der rechten Seite

befindet sich ein An-Knopf. Unter dem Bildschirm ist ein Schlitz für das, ebenfalls 3D-gedruckte, Spielmodul. Befindet sich kein Spielmodul im Schlitz, kann das Spiel nicht korrekt gestartet werden. Es ist weiterhin möglich den Reset-Knopf zu drücken. USB- sowie Strom-Anschluss sind durch Löcher in der Rückwand noch gut zu erreichen. Auf der Unterseite sind Lüftungsschlitz angebracht, damit das AURIX nicht überhitzen kann.

Das Spiel Nibbles ist ein Vertreter der Snake-Spiele, d.h. es wird eine Schlange aus der Vogelperspektive gesteuert um Beute zu fangen. Beute – hier Zahlen – wird zufällig auf dem Spielfeld platziert. Wird eine Zahl verspeist wächst die Schlange und die nächsthöhere Zahl erscheint. Dabei ist zu beachten, nicht gegen Wände oder den eigenen Schwanz zu kriechen. Tritt dieser Fall ein, wird ein Leben abgezogen. Wurden erfolgreich alle Zahlen von 0 bis 9 verspeist, startet der nächste Level. Insgesamt gibt es fünf Level. Am Ende erscheint die insgesamt erreichte Punktzahl.

2.2 Gehäuse

Das Gehäuse wurde in Rhino [McN] modelliert, mit Cura [Ult] gesliced und auf einem PrintrBot [Pri] gedruckt. Eine Schwierigkeit dabei war, dass kein öffentlich zugängliches CAD-Modell des AURIX existiert und die Abmessungen manuell erfolgten.

2.3 Fehlstart

Da das Spielmodul keine Daten enthält die vom AURIX ausgelesen werden, wird ein Fehlstart nur simuliert. Ein am Ende des Schlitzes angebrachter „geheimer“ Knopf wird ausgelesen. Ist er gedrückt startet das Spiel normal. Ist er nicht gedrückt startet das Spiel mit einem für Modul Konsolen typischen Fehlstart indem der Startbildschirm mit einer eigens dafür geschriebenen Funktion verpixelt wird siehe Abbildung 2.2.

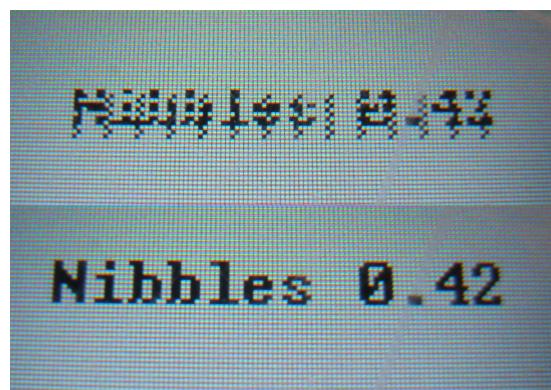


Abbildung 2.2: Oben: Fehlstart,
Unten: Normaler Start

2.4 Steuerung

Die Spiellogik ist in *nibbles.c* ausgelagert. In der *main()*-Methode wird die Funktion *tick(...)* aufgerufen. Die Schlange – Sammy – bewegt sich pro Tick nur einen Schritt weiter. Gesteuert wird Sammy mithilfe der zwei Potentiometer. Mit dem linken Potentiometer wird die horizontale Richtung (Links/Rechts), mit dem rechten die vertikale Richtung (Oben/Unten) geändert. Dabei wird immer nur die Differenz zum vorherigen Wert ermittelt und dementsprechend die Richtung geändert. Nachdem beide Potentiometer angeschlossen waren wie unter 1.2 gab es keine Probleme mehr. Ein Akkumulieren des Wertes durch setzen des Registers *VADC_G0RCR0.B.DRCTR*, war nicht nötig [AG13, S. 27-125]. Beim festkleben mit der Heißklebepistole wurde zuerst nicht beachtet, dass der rotierende Teil des Potentiometers nach unten hin offen ist, d.h. der Rotor wurde auch verklebt. Dies konnte durch das Befestigen eines Stücks Malerkrepps an der Unterseite beim zweiten Potentiometer verhindert werden. Der Unterschied ist deutlich beim Drehen der Potentiometer zu bemerken.

2.5 Musik

Die Spielmelodie wird mithilfe der unter 1.3 beschriebenen Funktion generiert, da es nicht möglich war die iLLD unter der Hightec Eclipse IDE zum laufen zu bringen. Ein daraus resultierendes Problem ist, dass der Ton stottert. Das kommt daher, dass die Funktion *printScrn()* aus der Bibliothek *display.h* sehr rechenintensiv ist und währenddessen kein PWM-Signal an den Lautsprecher gesendet werden kann. Ein weiteres Problem, ist das Fehlen eines Signalverstärkers, wodurch die Melodie nur sehr leise ist.

Kapitel 3

AURIX Handheld 2.0

Das Projekt AURIX Handheld hat großes Potenzial, da die Leistung des Tricore noch lange nicht ausgereizt ist. Mögliche Verbesserungen sind neben einem passgenauerem Gehäuse:

HighScore Liste: Am Ende wird der aktuelle HighScore mit einer im Flash-EEPROM gespeicherten Allzeit-Bestenliste verglichen, eingeordnet und angezeigt.

Musik: Einen echten GTM-TOM-Timer zum laufen bekommen und das PWM-Signal verstärken, sodass die Musik ohne Unterbrechungen und besser zu hören ist.

Multiplayer: Mithilfe des CAN-Bus. Sobald die Konsolen miteinander verbunden sind, müssen nur die Werte der jeweiligen Potentiometer Differenzen ausgetauscht werden. Pro Spieler wäre das nur ein Byte an Daten. Die Spiele selbst werden dann auf jedem AURIX getrennt berechnet.

”Echte“ Spielmodule: RFID-Chips. und Lesegeräte sind weit verbreitet und erschwinglich. Es könnten weitere Spiele programmiert werden und auf einer SD-Karte gespeichert werden. Auf dem RFID-Chip wird lediglich eine ID gespeichert die dann ausgelesen wird und daraufhin das entsprechende Spiel von der Karte geladen und gestartet.

Dieser AURIX Handheld 2.0 vereint viele Grundkonzepte der Mikrocontroller in minimaler Ausführung (Ein Byte über CAN-Bus senden, schreiben/lesen aus dem EEPROM, anschliessen eines RFID-Lesegeräts über i2c etc.) und ist dadurch ein ideales Beispielprojekt um von diesem aus weit größere Projekte zu realisieren.

Abbildungsverzeichnis

1	Infineon Handheld-Konsole	3
1.1	Der Knopf ist mit Pin 00.0 und VCC verbunden und kann dazu verwendet werden, LED0 an und auszuschalten.	2
1.2	Die Änderungen von einem auf zwei analoge Eingangssignale . . .	3
1.3	Implementierung eines Software PWM Signals an Pin33.1	5
1.4	Funktion zum schreiben eines Vierecks mit Breite w, Höhe h, Position (x,y) und Farbe color auf dem TFT Bildschirm des AURIX	6
2.1	Schaltplan der Infineon Handheld-Konsole	7
2.2	Fehlstart	8

Literaturverzeichnis

- [AG13] AG, Infineon T.: *AURIX TC22x/TC23x Family 32-Bit Single-Chip Microcontroller, Users Manual*. Infineon Technologies AG, 2013
- [AG14] AG, Infineon T.: *Application Kit TC2X4 Hardware: APPLICATION KIT TC2X4 V1.0 Hardware Manual*. Infineon Technologies AG, 2014
- [Gia10] GIANCOLI, Douglas C.: *Physik - Lehr- und Übungsbuch*. 3. erweiterte Auflage. München : Pearson Deutschland GmbH, 2010. – ISBN 978–3–868–94023–7
- [Gio09] GIORDANO, Nicholas J.: *College Physics: Reasoning and Relationships* -. 1. Aufl. Clifton Park, NY : Cengage Learning, 2009. – ISBN 053–4–424–716–
- [McN] MCNEEL, Robert: *Rhinoceros*. <https://www.rhino3d.com/de/>, . – (Visited on 03/22/2015)
- [Pri] PRINTRBOT: *Printrbot — Affordable high resolution 3D printers*. <http://printrbot.com/>, . – (Visited on 03/22/2015)
- [Rad91] RADDATZ, Rick: *Nibbles*. Microsoft Corporation. http://en.wikipedia.org/wiki/Nibbles_%28video_game%29. Version: 1991. – (Visited on 03/22/2015)
- [Ult] ULTIMAKER: *Ultimaker*. <https://ultimaker.com/en/products/software>, . – (Visited on 03/22/2015)