# ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

# FAKULTA ELEKTROTECHNICKÁ

Bakalářská práce

# Miniaturní procesorový systém s dotykovým LCD displejem pro monitoring a řízení

Studijní program: Elektrotechnika a informatika (bakalářský)
Obor: Kybernetika a měření

Vedoucí práce: Ing. Pavel Píša

Praha, 2008

Autor: Radek Pupák
pupakr1@fel.cvut.cz

**Prohlášení**

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze, dne 10. 7. 2008

Pupík Radek
podpis

# Anotace

Práce je zaměřena na tvorbu Linuxového ovladače pro desku PiMX1. Deska je osazena procesorem Freescale DragonBall M9328 MX1. Tento procesor má analogově-digitální převodník, který je navržený přímo pro analogový dotykový displej. Ovladač je psán pro jádro Linuxu 2.6.24, ale měl by být použitelný i pro novější jádra Linuxu. Práce se zabývá také teorií. Popisuje technologie dotykových panelů, provedení a možnosti desky PiMX1, některé vlastnosti použitého procesoru, rozebírá možnosti A/D převodníku v použitém procesoru. V neposlední řadě přibližuje operační systém Linux na ARM platformě a nastavení kompilace jádra, pro použitou desku. Dále se práce zabírá teorií při psaní Linuxového modulu ovladače a popisuje jeho funkci. V závěru práce nastiňuje možnosti propojení ovladače dotykového displeje s grafickým uživatelským prostředím Nano-X prostřednictvím knihovny Tslib.

# Annotation

This work purpose is driver of analog touchscreen for Linux on desk PiMX1. Desk is build-up on the processor Freescale DragonBall M9328 MX1. This proccesor contains analog/digital conventor designed for the analog touchscreen. Driver is writen for Linux 2.6.24 kernel, but it could be also usable in the latest version of kernel. The bachelor thesis also deal with theory, describing technologies of the touchscreen panels, desing and potentials of PiMX1 desk, some of properties of used processor and also describing oportunities of A/D convertor. The work also introduse operating system Linux on ARM platform and setting of kernel compilation for used desk. The work describing theory of writing Linux drivers as module and describe its function. The work also describing oportunities of driver integration in user graphics environment Nano-X trought Tslib project.

# Obsah

Obsah....4
1 Úvod....6
2 Dotykový displej [1, 2]....7
2.1 Historie a využití dotykového displeje....7
2.2 Odporový dotykový displej....8
2.3 Ostatní typy dotykových displejů....9
2.3.1 Kapacitní princip....9
2.3.2 Snímání povrchovou akustickou vlnou....10
2.3.3 Méně využívané metody snímání....10
2.3.3.1 Infračervené snímání....10
2.3.3.2 Metoda měření náklonu....10
2.3.3.3 Technologie rozptýleného mechanického signálu....10
2.3.3.4 Rozpoznání akustického pulsu....11
2.3.3.5 Nedopravený vnitřní odraz....11
3 Deska PiMX1 s mikroprocesorem Freescale....12
3.1 Specifikace desky [4]....12
3.2 Freescale [5,6]....16
3.3 Specifikace procesoru ARM920T [7,8]....16
3.4 ASP jednotka [9]....17
3.4.1 Seznámení s jednotkou analogového zpracování signálů....17
3.4.1.1 Auto Zero funkce....20
3.4.1.2 Detekce přitlačení pera....20
3.4.1.3 Detekce oddálení pera....20
3.4.1.4 Kompenzace teplotní chyby....20
3.4.2 Registry ASP....21
4 Arm Linux na PiMX1....22
4.1 Linux [10,11]....22
4.2 ARM architektura [12,13]....23
4.3 Specifické vlastnosti ARM Linuxu [13,14]....24
5 Oživení desky....25
5.1 Kompilace jádra [15, 16, 17, 18]....25
5.1.1 Získání zdrojových kódů....25
5.1.2 Úprava zdrojových kódů....26
5.1.3 Konfigurace parametrů kernelu....28
5.1.4 Kompilace kernelu....28
5.1.5 Přenos do výsledného zařízení....29
5.2 mx1-load Bootloader....29
5.3 U-Boot [19, 21]....30
5.4 Podpůrné programy....31
5.4.1 GNU Compiler Collection - GCC [20,21]....31
5.4.2 NetworkNetwork File System - NFS [22]....31
5.4.3 Trivial File Transfer Protocol - TFTP [23]....32
5.4.4 Minicom [24]....32
5.4.5 GIT [25]....33
6 Vývoj ovladače pro Linux [26, 15]....34
6.1 Možnosti integrace s jádrem....34
6.2 Kompilace....35
6.3 Práce s pamětí....36

6.3.1 Dynamicky alokovaná paměť....36
6.3.2 I/O paměť....37
6.4 Přerušení....38
6.5 Debugování....39
7 Ovladač dotykového displeje....40
7.1 Schéma ovladače dotykového displeje....40
7.2 Nastavení registrů ASP pro práci s displejem....41
7.3 Registrace platform driveru....42
7.4 Funkce ovladače pro obsluhu jaderných volání....43
7.5 Registrace vstupního zařízení kernelu....44
7.6 Zpracování příchozích přerušení....44
7.7 Časové schéma funkce ovladače....46
8 Grafické prostředí....47
8.1 Důvod grafického prostředí....47
8.2 Nano-X (microwindows) [27]....47
8.3 Knihovna Tslib [28]....47
9 Závěr....49
10 Seznam použité literatury....50
11 Seznam obrázků použitých v textu....52
12 Seznam příloh na CD....52

# 1 Úvod

Při komunikaci s počítači lidé využívají různá vstupní a výstupní rozhraní, která jim usnadňují interakci s technikou. Mezi standardní vstupní zařízení patří klávesnice a myš. Jako výstupní zařízení je většinou použit monitor nebo LCD displej. Technologie dotykového displeje spojuje vstupní i výstupní zařízení do jednoho a tím vytváří více intuitivní prostředí pro uživatele, který má možnost reagovat přímo dotykem na zobrazované objekty.

Tématem mé bakalářské práce je „Miniaturní procesorový systém s dotykovým LCD displejem pro monitoring a řízení“. Toto zadání jsem si vybral, protože jsem chtěl využít možnosti podrobněji se seznámit s jádrem operačního systému Linux a s procesem vývoje ovladačů.

Cílem této práce je tedy vytvoření ovladače pro dotykový displej připojený k desce PiMX1 a ukázka možnosti jeho využití v aplikacích pro monitoring a řízení.

Pro řešení zadaného úkolu jsem si vybral operační systém ARM Linux, který byl na desce otestován. Z tohoto důvodu byly patche kernelu již hotové. Nicméně pro práci je nutné zkompilovat nové jádro s upraveným nastavením pro konkrétní desku, tak aby vyhovovalo konkrétní hardwarové implementaci desky s barevným dotykovým displejem.

Pro seznámení s hardwarem bylo možné využít pouze dokumentaci firmy PiKRON a dokumentaci k procesoru od firmy Freescale.

Při psaní ovladače dotykového displeje bylo možné vycházet ze vzorového „továrního“ ovladače napsaného firmou Freescale pro jádro Linuxu verze 2.4 a také.je možné nahlížet do nově napsaných ovladačů jiného hardwaru pro jádro Linux 2.6, které jsou umístěny ve zdrojových kódech Linuxu v adresáři /drivers/input/touchscreen. O vývoji linuxových ovladačů bylo napsáno několik publikací. Ty, ze kterých jsem čerpal, jsou uvedené v seznamu literatury.

Pro implementaci grafických obrazovek bych chtěl využít existující koncept grafického uživatelského prostředí Nano-X (dříve nazývaný Microwindows). Uživatelské prostředí je dlouho vyvíjeno a na ovládání pomocí dotykového displeje je uživatelské prostředí také připraveno. Podle mého názoru je vhodnější začlenit dotykový displej do fungujícího konceptu grafického prostředí s vysokou pravděpodobností budoucího vývoje než implementovat jednoúčelové prostředí, které by v průběhu času zastaralo a stalo se nepoužitelným.

Vzhledem k tomu, že nyní jsem pouze běžným uživatelem operačního systému Linux, tak předpokládám, že práce bude také shrnutím potřebných znalostí k vývoji Linuxového ovladače.

V případě, že vývoj bude postupovat bez komplikací je mým cílem vytvořit jednoduchou demonstrační aplikaci, na které budou patrné možnosti dotykového displeje.

Každý oddíl mé je vystavěn na literatuře, kterou uvádím v nadpisu příslušné kapitoly v hranatých závorkách (např. [2]). Uvedené číslo odpovídá příslušné položce v seznamu literatury.

# 2 Dotykový displej [1, 2]

Dotykový displej je zobrazovací zařízení, které je také vybaveno technologií pro detekci dotyku. Slouží nejen jako výstupní, ale také jako vstupní zařízení systému. Může být náhradou myši, nebo v případě, že je používán společně se softwarem pro něj navrženým může být jediným vstupním zařízením. Název dotykový 'touch' je dán schopností zařízení snímat dotyk na ploše obrazovky. Dotyk může být realizován prsty nebo dotykovým perem – stylusem. Seznámíme se nejprve s možnostmi využití dotykových displejů, jejich typy a poté se zaměříme na konkrétní typ použitý při realizaci dotykového displeje s deskou PiMX1.

*Obr 1. : Příklad dotykového displeje*

## *2.1 Historie a využití dotykového displeje*

První elektronické zařízení pro zpracování dotyku bylo vynalezeno v roce 1971 Dr. Samuelem C. Hurstem. Dnes je již dotykový displej poměrně rozšířený na mnoha místech. Prvním komerčním počítačem využívajícím touchscreen byl zřejmě HP-150 (obr. 2) z roku 1983. Dotyková nebyla přímo obrazovka, ale v prostoru před obrazovkou bylo několik infračervených vysílačů a přijímačů, čímž bylo možno určit pozici neprůhledného předmětu v prostoru obrazovky.

*Obr 2. : Počítač HP-150 vybavený dotykovým displejem*

Dotykový displej je vhodný pro mnoho aplikací. Uplatnění nalézá také v průmyslových aplikacích, bankomatech, terminálech městské hromadné dopravy, v muzeích, přenosných počítačích, PDA, tzv. chytrých telefonech, některých herních konzolích (například Nintendo), na informačních kioscích. Výhodou dotykového displeje oproti standardním vstupním rozhraním (myš a klávesnice) je intuitivní a rychlé ovládání, kdy uživatel přímo reaguje na informace zobrazené na monitoru. Hlavní překážkou masivnějšího rozšíření dotykových displejů je zatím jejich vyšší cena v porovnání s myší a klávesnicí, která je však pro některá využití stále výhodnější. Mezi aplikace, kde se se dotykový displej nehodí využít patří například terminál vyžadující zadávání dlouhých textů, kde je rychlejší a pohodlnější psaní na klávesnici.

Dotykové displeje se dostávají do stádia, kdy si výrobci uvědomili uživatelsky přívětivé vlastnosti technologie a dotykové displeje se začínají více objevovat v běžných zařízeních – terminály, pokladny, mobilní telefony. Budoucnost vývoje dotykových displejů se jistě zaměří na zařízení umožnující registraci více dotyků zároveň (jako využívá Apple iPhone), protože více dotyků nabízí více možností interakce uživatele s přístrojem a také je více intuitivní a v neposlední řadě také umožňuje spolupráci několika uživatelů na jednom dotykovém displeji. Vzhledem k větší výrobě dotykových displejů lze očekávat jejich zlevnění a využití v aplikacích od letadel, automobilů, průmyslových ovládacích terminálů, herních konzolí až k mobilním telefonům a jiným přenosným zařízením.

Nevýhodou touchscreenů je problém namáhání lidských prstů při používáním delším než několik minut, protože většina technologií vyžaduje vyvíjení určitého tlaku proti pevné podložce. Tato nevýhoda je zmírněna možností používání dotykového pera, nebo jiného ukazatele, který svým malým hrotem snižuje sílu potřebnou k zaregistrování dotyku. Použití stylusů však není v některých podmínkách možné – například ve veřejných kioscích nebo bankovních automatech.

## *2.2 Odporový dotykový displej*

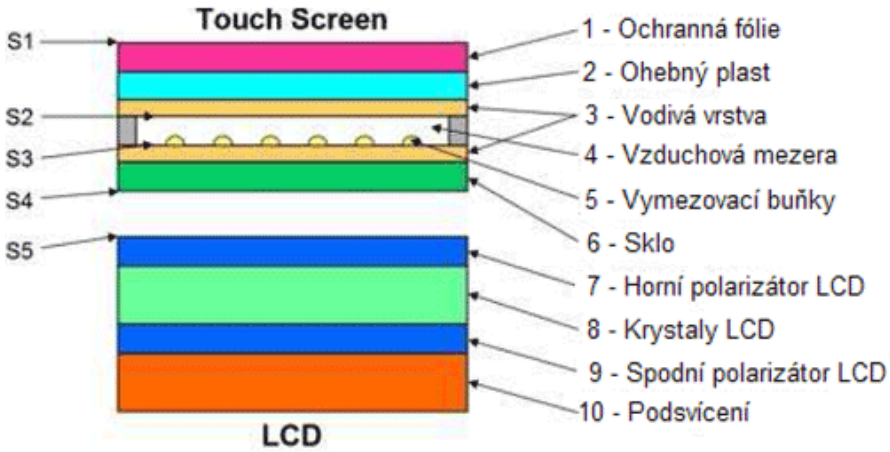
Odporový dotykový displej, jenž používám ve své práci, je složen z několika vrstev. Nejdůležitějšími z nich jsou dvě tenké vodivé odporové kovové vrstvy, mezi nimiž je tenká mezera.

Při dotyku se vrstvy v bodě dotyku propojí. Tím se z vrstev stane odporový napěťový dělič. Proud procházející odporovým děličem je vyhodnocen jako dotyk a přeposlán kontroléru ke zpracování.

Při požadavku měření velikosti tlaku dotyku je nutné přidat mezi vodivé vrstvy odpor závislý na tlaku. Výstup odporového dotykového displeje mohou tvořit 4 až 8 drátů. Pozice vodivých kontaktů ve vodivých vrstvách se mohou lišit, což je určující pro počet vodičů. Pokud jsou vyvedeny 4 vodiče, tak jsou kontakty umístěny na hranách displeje – nahoře, vlevo, vpravo a na spodní hraně. V případě pěti-vodičové technologie jsou kontakty umístěny v rozích na jedné desce.

Poloha dotyku se počítá z protékajících proudů. Světelná propustnost odporové vrstvy je asi 75% a vrstva může být poškozena ostrými objekty. Snímání polohy není ovlivněno vnějšími vlivy jako jsou prach nebo voda a jedná se v současnosti o nejpoužívanější technologii.

Dotykové displeje založené na odporové technologii se používají v restauracích, nemocnicích, továrnách a ve většině přenosných zařízení. Lze je obsluhovat i v rukavicích. Rychlost reakce je 10-20 ms. Životnost je u 4 vodičové technologie asi milion dotyků u 5 vodičové 35 milionů dotyků.




*Obr 3. : Znázornění vrstev odporového dotykového displeje*

Jak můžeme vidět z řezu touchscreenem na obrázku č. 3 vodivé vrstvy jsou nanesené pouze na skle a na ohebném plastu. Ve vzduchové mezeře mezi vodivými vrstvami jsou ještě vymezovací buňky, které brání dotyku vodivých vrstev bez vnějšího tlaku.

## *2.3 Ostatní typy dotykových displejů*

Snímání pozice dotyku na displeji může být řešeno i jinými technologiemi, ty nejvíce rozšířené jsou rozebrány níže.

### 2.3.1 Kapacitní princip

Kapacitní snímání polohy dotyku je založené na pokrytí monitoru tenkou vrstvou metalického vodivého materiálu (oxidu india a stříbra), která vede elektrický proud. Vodivou plochou v klidovém stavu protéká malý proud, který je měřen v každém rohu monitoru. Lidské tělo má určitou elektrickou kapacitu, a proto se při dotyku změní kapacita plochy, což ovlivní velikost proudu protékajícího vodivou vrstvou. Vychýlení proudu z referenčního stavu je snímáno elektronickými obvody v rozích panelu. Ze změřených rozdílů kapacit se matematicky spočítá poloha dotyku. Kapacitní senzory jsou citlivé na kontakt prstu, nebo vodivého předmětu drženého v holé ruce. Touchscreen je necitlivý vůči dotykům nevodivými předměty a v rukavicích. Kapacitní dotykové obrazovky bývají hermeticky uzavřené, aby se zabránilo poškození vodou a špínou. Typicky je kapacitní vrstva světelně propustnější než odporová, proto displeje mívají vyšší jasnost než touchscreeny založené na odporovém principu. Díky citlivosti na nečistoty je třeba displej překalibrovávat, také jsou citlivé na poškrábání. Rychlost odezvy je mezi 15 až 24 ms, propustnost světla kapacitní vrstvou je 75%, typicky je možné rozlišit 50 dotyků na palec plochy. Kapacitní vrstva by měla bez problémů přežít více než 20 milionů dotyků. Typické využití kapacitních senzorů je ve veřejných informačních a prodejních kioscích.

### 2.3.2 Snímání povrchovou akustickou vlnou

Povrchová akustická vlna – SAW (Surface acoustic wave) je technologie využívající ultrazvukové vlny, které jsou vysílány nad skleněným povrchem obrazovky. Většinou jsou používány piezoelektrické vysílače a přijímače doplněné o odrazové pruhy. Využívá se pulsů o frekvenci 5 MHz, které vybudí vysílače k vysílání vlny. Ta se šíří po mechanickém materiálu, kterým je v tomto případě rovné sklo kryjící obrazovku. Při dotyku je část energie vysílané vlny absorbována prstem. Při zpracování přijaté vlny je kontrolér schopen určit souřadnice X,Y dotyku. Výhodou této technologie je, že funkce dotykového rozhraní nemůže být poškozena neopatrným zacházením, jelikož vnějším vlivům je vystaveno pouze sklo. Touchscreeny jsou jasné, stabilní, odolné vůči změnám teploty a vlhkosti. Poškrábání skla ani jeho chemické znečištění nemá na funkci akustické vlny vliv. Doba odezvy technologie SAW je 20 ms, propouští 92% světla vyzářeného monitorem, technologie by měla být schopna rozlišit 30 poloh dotyků na jeden čtvereční palec. Životnost dotykového displeje fungujícího na principu povrchové akustické vlny je v podstatě neomezená (obrazovky jsou testovány na 50 miliónů dotyků).

### 2.3.3 Méně využívané metody snímání

#### *2.3.3.1 Infračervené snímání*

První metodou infračerveného snímání je snímání teplotních změn na odporovém povrchu. Tato metoda je pomalá a vyžaduje teplé ruce.

Druhá metoda snímání je založená na polích infračervených senzorů umístěných vertikálně a horizontálně. Senzory detekují přerušení infračerveného světelného paprsku – registrují stín. Infračervené dotykové panely mají velmi odolné povrchy a bývají používány ve vojenských aplikacích.

#### *2.3.3.2 Metoda měření náklonu*

Při použití této technologie je zobrazovací panel uchycen v rozích takovým způsobem, aby se mohl mírně ve směru osy Z pohybovat. Při stlačení panelu se změří posunutí jednotlivých rohů. Z těchto údajů je možné vypočítat polohu a tlak dotyku. Tato technologie je velmi odolná vůči působení vandalů, proto je vhodná pro veřejně přístupné terminály.

#### *2.3.3.3 Technologie rozptýleného mechanického signálu*

Tato technika měření polohy dotyku byla představena v roce 2002. Senzory měří mechanickou energii ve skle, která je předána dotykem. Technologie by měla být odolná vůči prachu a vnějším vlivům včetně mechanického poškození (poškrábání). Jelikož kromě skla není nutné obrazovku ničím překrývat, měla by být zaručena také dobrá propustnost světla. Dotyk může být zprostředkován předmětem přenášejícím mechanickou energii. Nevýhodou technologie je neschopnost rozlišit délku dotyku.

#### *2.3.3.4 Rozpoznání akustického pulsu*

Metoda snímání je založená na piezoelektrických přijímačích, které přemění mechanickou energii dotyku (vibrace) na elektrický signál, jenž je převeden na audio signál, který je porovnáván s předdefinovanými signály pro každou pozici obrazovky. Stejně jako u metod SAW, měření náklonu a technologie rozptýleného mechanického signálu není pro snímání potřeba žádných prvků překrývajících displej než běžného skla. Proto metoda umožňuje dobrou propustnost světla a je necitlivá na znečištění prachem a poškrábání. Nevýhoda metody je stejná jako u technologie rozptýleného mechanického signálu – po prvotním dotyku není registrován nepohybující se prst.

#### *2.3.3.5 Nedopravený vnitřní odraz*

Metoda s anglickým názvem Frustrated total internal reflection používá odraz světla ve vrstvě mezi obrazovkou a citlivou vrstvou k detekci pozice prstu. Když se měkká vrstva stlačí, tak vnitřní cesta světla je přerušená a dojde k pohlcení paprsku a pozice dotyku je pohlceným světlem zvýrazněna. Kamera pak zachytí takto vzniklé zvýraznění měkké vrstvy. Metoda byla zlepšena na metodu Diffused laser imaging, která používá stejný softwarový a zobrazovací systém, ale jako světelný zdroj slouží rozptýlené laserové světlo. Výhodou tohoto vylepšení je možnost použití jen dvou zdrojů světla místo nutnosti použití pole světelných zdrojů.

# 3 Deska PiMX1 s mikroprocesorem Freescale

## *3.1 Specifikace desky [4]*

Deska PiMX1 byla navržena firmou PiKRON s.r.o. pro injekční pumpy využívané ve zdravotnictví.

*Obr 4 : Praktická aplikace desky PiMX1 - injekční pumpa*

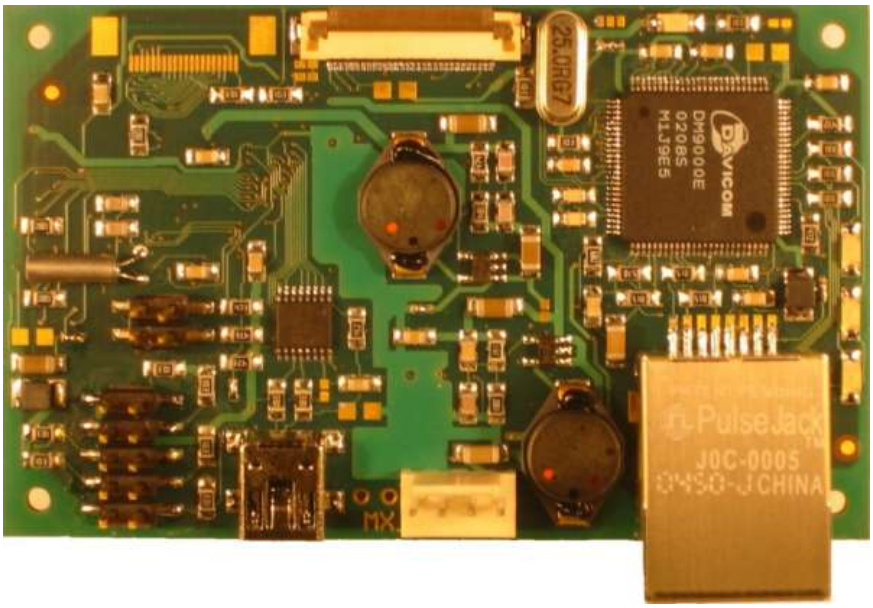
Společnost PiKRON s.r.o. byla založena jako pokračovatel „Czech producers of laboratory instruments“. Hlavním cílem společnosti je pokračovat ve vývoji a výrobě laboratorních nástrojů pro české nemocnice, laboratoře, akademické instituce a průmysl. Firma při výrobě injekční pumpy sopulupracovala s firmou Technic I. PiKRON je zodpovědná za mechanický, elektronický a softwarový návrh a vývoj instrumentů, ale většina výroby je poté realizována firmou Technic I.

*Obr 5.: Deska PiMX1, pohled ze strany procesoru*

Deska PiMX1 je vybudována na architektuře ARM i.MX. V práci je rozebírána verze desky s označením 'MX_CPU2 Board'. Je navržena jako 'universal controller board', to znamená, že má široké možnosti použití v různých aplikacích. Je vybavena řadou rozhraní, která zvyšují možnosti jejího začlenění do stávajícího systému a usnadňují použití v nových aplikacích.

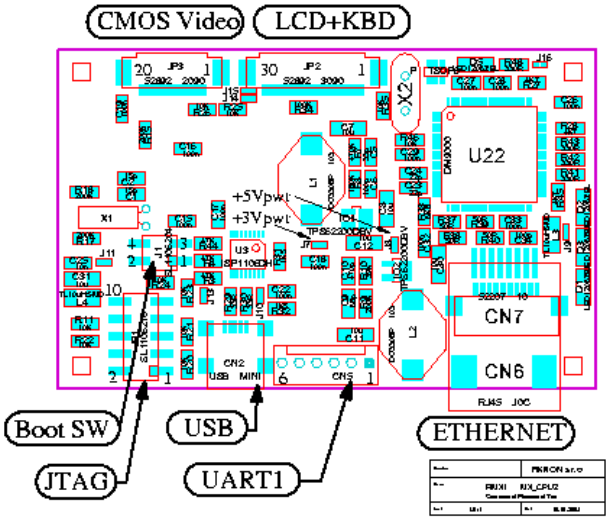
Základem desky je procesor od americké firmy FreeScale DragonBall M9328 MX1 vystavěný na architektuře ARTM920T. Tento procesor může pracovat na maximální frekvenci 92 nebo 192 MHz (dle verze procesoru). Procesor má 16kB instrukční a 16 kB datovou paměť cache. Správu paměti zajišťuje jednotka managementu virtuální paměti (VMMU) s 64 + 64 translation lookaside bufferem (TLB). Deska je vybavena 128 kB pamětí eSRAM, 32 MB pamětí SDRAM pracující na frekvenci 100 MHz, která tvoří operační paměť desky. Paměť SDRAM je připojena jen 16-ti bitovou sběrnicí z důvodu zachování malých rozměrů desky. Pomalejší 16-ti bitová sběrnice je kompenzována rychlostí cache procesoru.

Deska může být vybavena rozšiřující FLASH pamětí typu StrataFlash o velikosti od 4 MB do 32 MB. Osazení desky FLASH pamětí je ovlivněno předpokládaným využitím desky.




*Obr 6.: Deska PiMX1, pohled ze strany součástek*

Je možnost osazení různých rozhraní. Nejvýznamnějším z nich pro běžné použití desky je zřejmě USB rozhraní, které může pracovat v rychlosti Full speed (12Mbit/s). Rozhraní je osazeno Mini-USB konektorem. Je použitý ISP1106DH USB transceiver, který zajišťuje softwarovou kontrolu připojení. Deska MX_CPU2 může být připojena přímo k hostitelskému počítači a může z něj být prostřednictvím USB rozhraní také napájena. USB rozhraní desky je pouze typu slave, nelze jej tedy bohužel využít k rozšíření desky o USB příslušenství.
Tři rozhraní UART jsou přístupné prostřednictvím TTL signálů o velikosti 3.1 V. Rozhraní UART1 je připojené k rozšiřujícímu konektoru desky. Ostatní UART rozhraní (UART2 a UART3) jsou připojena k vstupně/výstupním rozhraním nebo ke konektoru sběrnice umožňující rozšíření desky. Signály úrovně 3.1 V mohou být převedeny na signály RS-232 ve standardním DB9 konektorovém rozhraní.

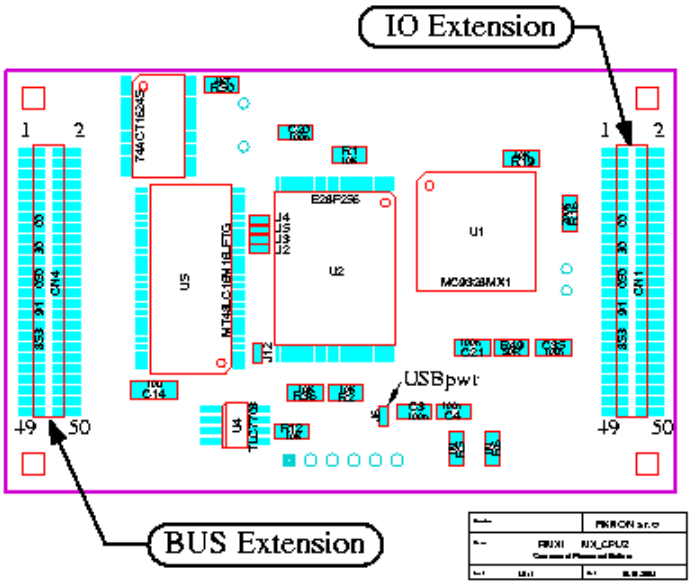



*Obr 7.: Schématické rozmístění součástek na desce*

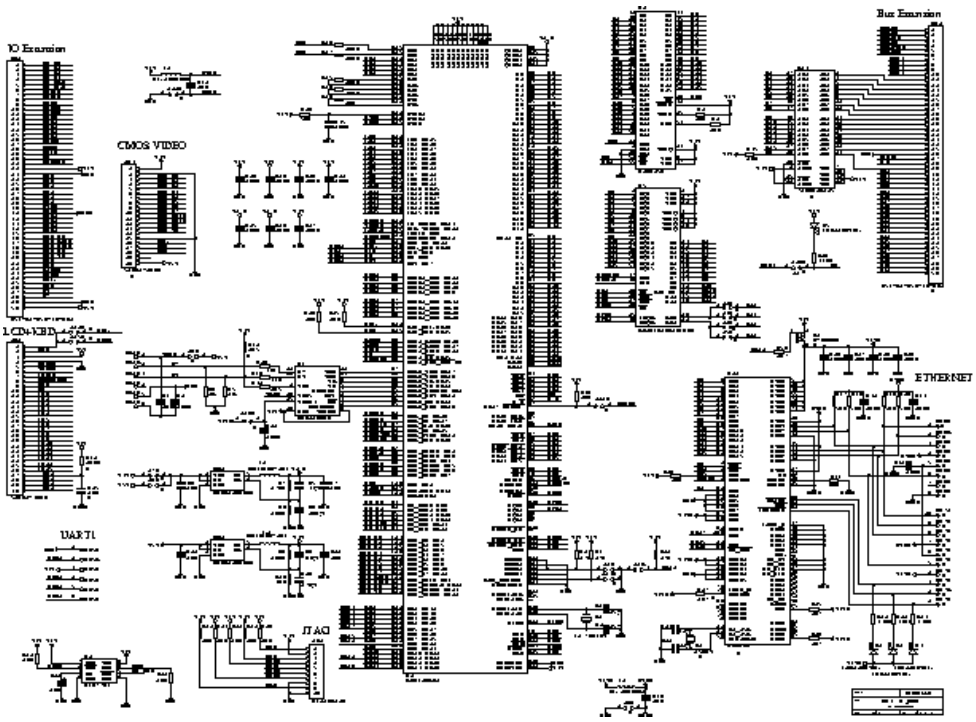
Deska je dále vybavena kontrolérem rozhraní ETHERNET. Je realizováno čipem DM9000 připojeným k procesoru 16-ti bitovým datovým rozhraním. Deska může být vybavená rozhraním 10Base-T/100Base-TX a čtyřmi signálními LED diodami, nebo přímo modulem integrovaného konektoru (ICM) pro přímé připojení ke standardní Ethernetové infrastruktuře.

Dále na desce nalezneme 30-ti pinový Molex FFC/FPC-to-Board konektor pro připojení modulu LCD displeje. Umožňuje přímé připojení většiny pasivních a TFT LCD panelů. Maximální podporované rozlišení je 640x512 pixelů a je podporováno až 64 tisíc barev pro TFT panely a až 16 odstínů šedé nebo 4096 barev pro pasivní LCD displeje. Dále konektor obsahuje analogové vstupy pro připojení analogového dotykového displeje. Piny, které nejsou použité pro připojení LCD displeje, mohou být použité pro připojení maticové klávesnice.

CMOS video vstup je vyveden pomocí 20-pinového Molex FFC/FPC-to-Board konektoru. Procesor MC9328 umožňuje přímé připojení CMOS kamery o nízkém rozlišení.




*Obr 8.: Schématické rozmístění součástek ze strany procesoru*




*Obr 9.: Schématický diagram desky*

Dalšími rozhraními desky jsou I/O Extension Connector a BUS Extension Connector. První zmiňovaný prostřednictvím dvouřadého ultra-miniaturního 50-ti pinového konektoru umožňuje připojení většiny vstupně/výstupních signálů – SecureDigital interface, UART1 a UART2, SPI a I2C, analogových vstupů, časovačů, PWM a některých dalších vstupně-výstupních signálů pro obecné použití. Druhý jmenovaný je osazený stejným typem konektoru a slouží k možnosti rozšíření desky o další příslušenství. 16-bitová sběrnice je připojena k vyhrazeným signálům procesoru MX1, také je dostupná adresová sběrnice a signály chip-select poskytující tři 64kB paměťové oblasti.
Deska je osazena také dvouvodičovou I2C sběrnicí, která umožňuje její další rozšíření. SDA a SCL signály jsou přístupné pro všechny podsystémy na IO Extension konektoru, LCD+KBD konektoru a vstupního CMOS video konetoru. 10 pinový JTAG konektor je určen pro usnadnění debugingu.
Praktickou vlastností desky je osazení dvou jumperů, které umožňují přepínání módu bootování. Jeden vybírá MX1 serial bootstrap mód a druhý slouží k zablokování bootování z paměti FLASH. Deska obsahuje 4 diody, jedna z nich je použitelná softwarově pro diagnostické účely a ostatní 3 jsou určeny pro zobrazení statutu připojení a trafiku na ETHERNET rozhraní.

Deska může být napájena přímo prostřednictvím USB rozhraní o napětí 5 V, nebo pomocí vyhrazených pinů na IO Extension konektoru. Na desce jsou umístěny dva spínací zdroje, proto je požadováno napájecí napětí 5 V nebo alternativně 3.1 V. Deska má proudový odběr mezi 20 a 30 mA při napětí 5 V s vypnutým napájením Ethernetového rozhraní. Pro tento účel je deska vybavena softwarově ovládaným MOSFET spínačem. Zapnutý EHTERNET odebírá relativně vysoký proud okolo 100 mA.

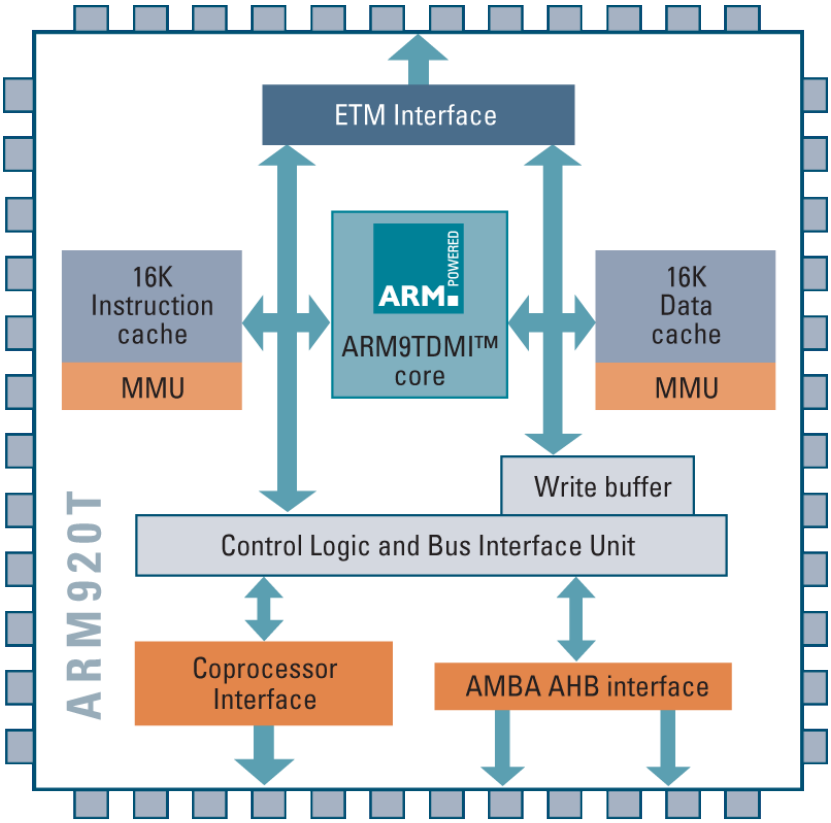
Rozměry desky jsou 50 x 79 mm. Na platformě jsou otestované operační systémy ARM LINUX 2.6.x a RTEMS 4.7.
Hardwarová část desky byla navržena Petrem Porazilem a nápady a softwarová část práce byla provedena Pavlem Píšou. Vývoj byl financován a návrh je vlastněn firmou PiKRON s.r.o.

## 3.2 Freescale [5,6]

Freescale Semiconductor, Inc. (http://www.frescale.com) je americkou firmou vyrábějící polovodiče. Byla založena odštěpením divize polovodičů od mateřské firmy Motorola v červenci 2004. Firma s více než 50-ti letou historií v mikroelektronice nabízí široké portfolio polovodičů. V jejích produktech nalezneme čipy pro mnoho odvětví elektroniky např. automobilový průmysl, počítačové sítě, bezdrátové a mobilní komunikace, embedded systémy... Centrála společnosti sídlí v Austinu, Texasu, ale vývojová, výrobní a prodejní centra jsou ve více než 30. zemích. Firma má v České republice své obchodní zastoupení a v Rožnově pod Radhoštěm sídlí její vývojové centrum.

## 3.3 Specifikace procesoru ARM920T [7,8]

Platforma procesoru ARM920T je použitelná pro široké možnosti aplikací s různými operačními systémy. Je založen na výkonném ARM9TDMI 32-bitovém RISC procesoru. Procesor obsahuje instrukční a datovou paměť cache (16 a 16 KB), memory management unit. Z hlediska integrace podporuje procesor AMBA sběrnicový interface a ARM real-time trace technology, jenž umožňuje debugování procesoru při zachování jeho plného výkonu.




*Obr 10 : Blokový diagram architektury procesoru ARM920T*

Procesor je použitelný pro aplikace v komunikátorech, chytrých telefonech, PDA, pro zachytávání obrazu, audio a video dekódování, automobilové informační prostředí, pro set-top boxy nebo pro použití v náročnějších aplikacích s operačním systémem. Procesor využívá 32-bitovou sadu ARM instrukcí, ale je možné také využívat 'Thumbmb instruction set', která obsahuje jen nejpoužívanější

instrukce a 16-bitovou délkou instrukcí snižuje velikost výsledného kódu. Memory management unit podporuje různé operační systémy například Symbian OS, Windows CE, Linux a Palm OS.

Procesory pracují na maximálních frekvencích 190 – 200 MHz, pro 0,18µm výrobní proces a mají spotřebu 0,8mW/MHz, při výrobě 0,13µm technologií mají spotřebu 0,25mW/MHz a dosahují frekvencí 230 – 250 MHz.

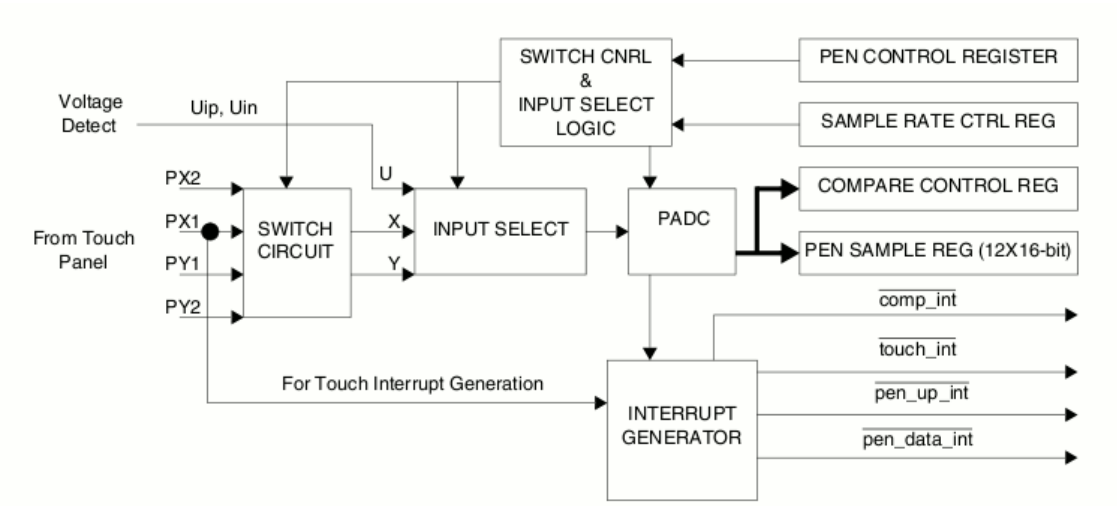
## *3.4 ASP jednotka [9]*

### 3.4.1 Seznámení s jednotkou analogového zpracování signálů

'Analog Signal Processor' (ASP) je modul procesoru MC9328MX1, který obsahuje 9-ti bitový analogově/digitální převodník pro dotykový displej. Výhodou je, že čip procesoru obsahuje referenční napětí pro převodník. Pro uchovávání navzorkovaných hodnot je implementována fronta (zásobník typu FIFO), která umožňuje uchovat 12 vzorků o velikosti 16 bitů. Jsou vestavěné obvody pro dotykový displej, což zjednodušuje práci při zpracování dotykových událostí. Převodník podporuje automatický i manuální mód vzorkování. Generovaná přerušení 'PEN DOWN' a 'PEN UP' lze programovat. Další přerušení, která převodník poskytuje jsou "DATA-READY", po načtení vzorku, a FIFO-full, pokud dojde k zaplnění vzorkovacího zásobníku. Podporuje také softwarovou kompenzaci teplotní chyby.

Jasnější přehled o funkci ASP jednotky poskytuje její blokový diagram. Z diagramu je patrné, že jednotka zpracovává 3 signály – Uin, což je hodnota externího napětí, tuto část převodníku je vhodné využít v aplikacích, ve kterých je deska s procesorem MC9328MX1 napájena z bateriového zdroje, ke vzorkování napětí na baterii. Zbylé dva vstupy X a Y slouží pro detekci polohy dotyku na dotykovém displeji. Jsou vzorkovány celkem 4 signály PX1, PX2, PY1 a PY2. Ty jsou pomocí interní logiky převedeny na hodnoty X a Y.

Softwarovému ovladači je k dispozici jednak ovládání logiky převodu signálů PX1, PX2, PY1 a PY2 na signály X a Y, které vstupují do analogově/digitálního převodníku. Také přepínání konkrétního signálu, který vstupuje do převodníku je softwarově ovladatelné.




*Obr 11 : Blokový diagram ASP systému*

**Jednotka dokáže generovat 4 typy přerušení.**

'Pen Data Compare Interrupt (comp_int)', kterému je přiřazena hodnota přerušení 9, je volitelně generováno, pokud právě odečtený vzorek splňuje podmínky definované v 'compare control register'. Zde lze uložit referenční hodnotu vzorku a podmínky generace přerušení - zda bude vyvoláno vzorkem nižším nebo vyšším než je uložená referenční hodnota.

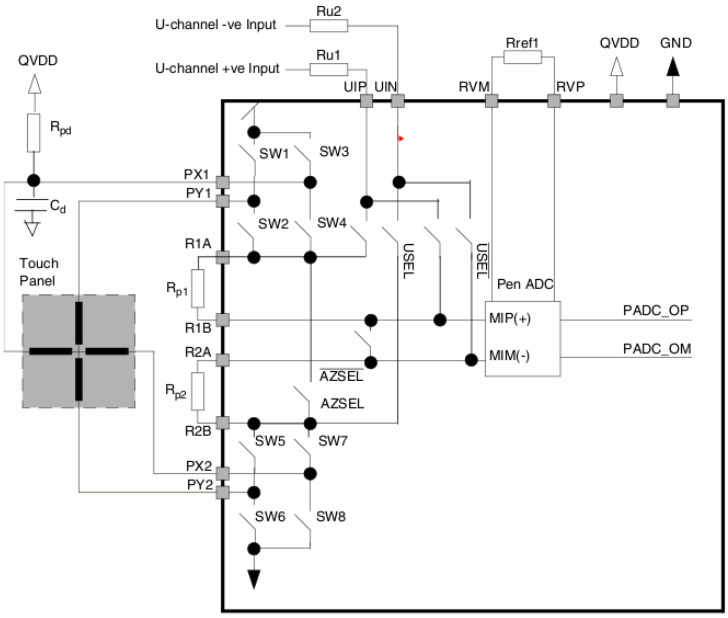
'Pen Touch (touch_int)' má přiřazenou hodnotu přerušení 46 a je generováno při události dotyku perem. Je možné definovat spouštění hranou, nebo nastavitelnou hodnotou.

'Pen-up Interrupt (pen_up_int)' má přiřazené přerušení číslo 5. Přerušení je generováno pouze při automatickém vzorkovacím módu (vstupní signály do převodníku se periodicky přepínají) typu XYU nebo ZXYU s nastaveným PUIE bitem (bude níže podrobněji vysvětleno), když je detekováno zvednutí dotykového pera.

'Pen FIFO Data Ready Interrupt' a 'Pen FIFO Full Interrupt' (pen_data_int) používají hodnotu přerušení 33 a indikují změnu stavu na zásobníku vzorků. Přerušení při plném zásobníku je od nové hodnoty na zásobníku odlišeno bitem 'Pen sample data overflow' v registru 'Interupt/Error status register'.

ASP jednotka umožňuje auto-vzorkovací mód. Při tomto nastavení je vzorkovací frekvence volitelná až do 9,6 kHz při vzorkování jen U – chanel (signálu malého napětí), nebo až do 1,2 kHz při vzorkování signálů XYU a auto kalibračního vstupu.

Pro výběr vzorkovaného signálu slouží 8 spínačů pro signály X a Y dotykové obrazovky. V manuálním módu jsou přepínače vypínány a zapínány příkazy prostřednictvím 'Switch Control and Input Select Logic block', který je ovládán prostřednictvím ASP registrů . V automatickém módu jsou přepínače ovládány automaticky podle vzorkovaných signálů. Jasnější popis je patrný z Obr. č. 12.




*Obr 12 : Zjednodušený diagram přepínacích obvodů ASP jednotky*

Nastavení spínačů pro čtení signálů (1 – spínač sepnutý, 0 – spínač vypnutý):

```
Read X: SW[8:1] = 1100 0110
Read Y: SW[8:1] = 0011 1001
Auto zero: SW[8:1] = 0000 0000
Auto Calibration X SW[8:1] = 1100 1100
Auto Calibration Y SW[8:1] = 0011 0011
Default: SW [8:1] = 0010 000
```

výrobcem doporučené hodnoty součástek a napětí na vstupech převodníku:

| Parameter | Description | Typical Value |
|---|---|---|
| Rp1, Rp2, Ru1, Ru2 | Pen ADC Input Resister | 200kOhm |
| Cd | Noise Decoupling Cap | 0.01uF |
| Rpd | Pen-Down Detect Resister | 100kOhm |
| Rref | Analog Reference Resister | 40kOhm |
| QVDD | ASP Power Supply | 1.8V |
| GND | ASP Circuit Ground | 0V |

Navzorkované hodnoty jsou uloženy v 'Pen Sample FIFO', jenž má formát 16 bitových celých čísel bez znaménkového bitu. FIFO je navržen jako kruhový buffer, proto data, jenž nejsou dostatečně rychle čtena jsou přepisována od nejstarších. Při přetečení zásobníku je nastaven bit POV v registru 'Interrupt/Error Status Register'.

ASP jednotka umožňuje režim automatického vzorkování a mód manuálního odečtu vzorků. Přehled dostupných variant vzorkování uvádí ilustrace č. 13 níže.

| AUTO | PADE | MOD | AZE | Data Format in the Pen Sample FIFO | Notes |
|---|---|---|---|---|---|
| 1 | 1 | 00 | Don't Care | ADC idle | No A/D sample |
| | | 01 | 0 | X,Y,X,Y.... | Only sample pen input, Disable U input with auto-zero disabled |
| | | 01 | 1 | AZ,X,Y,AZ,X,Y.... | Only sample pen input, Disable U input with auto-zero enabled |
| | | 10 | 0 | X,Y,U,X,Y,U... | Sample pen input and U input with auto-zero disabled |
| | | 10 | 1 | AZ,X,Y,U,AZ,X,Y,U,... | Sample pen input and U input with auto-zero enabled |
| | | 11 | Don't Care | U,U,U... | Only sample U input |
| Don't Care | 0 | Don't Care | Don't Care | No valid data | Pen idle state |
| 0 | 1 | 00 | Don't Care | ADC idle | No A/D sample |
| | | 01 | 0 | X | In this manual mode, only sample when PADE is toggled from 0 to 1 by software. |
| | | 01 | 1 | AZ, X | |
| | | 10 | 0 | Y | |
| | | 10 | 1 | AZAZ, Y | |
| | | 11 | Don't Care | U | |

*Obr 13 : Tabulka uvádějící možnosti nastavení automatického vzorkování ASP jednotky*

Pro použití s dotykovým displejem se jeví nejvíce výhodná možnost vzorkování typu AZ (auto zero), X, Y, U nebo AZ, X, Y při nevyužitém napěťovém vstupu. Při automatickém vzorkování se spínače automaticky přepínají tak, aby vzorkované hodnoty přicházely v definovaném pořadí. Po digitalizaci hodnoty signálu je vzorek uložen do zásobníku a generováno přerušení 'Data ready'. V manuálním módu vzorkování (bit AUTO je nastaven na 0) jsou hodnoty odečítány pouze při softwarové změně PADE bitu z 0 na 1.

Analogově/digitální převodník pracuje v proudovém režimu. To znamená, že výstupní vzorek je rozdílem vstupních proudů. Vstupní napětí jsou převedena na vstupní proudy na rezistorech R1, R2.

Proto platí následující vztahy:

- ip = (Vp - V1a) / R1 a zároveň je velikost proudu omezená $-2{,}5\mu A \leq ip \leq +9{,}5\mu A$
- im = (Vm - V2b) / R2 $-2{,}5\mu A \leq im \leq +9{,}5\mu A$

$-12\mu A \leq \Delta i \leq +12\mu A$

To znamená, že vstupní rozsah převodníku lze upravit změnou rezistorů.

Vstupní napětí na vstupních pinech procesoru (včetně X, Y a U signálů) nesmí překročit velikost 1,8 V.

#### *3.4.1.1 Auto Zero funkce*

Vzhledem ke snižování přesnosti A/D převodníku šumy vznikajícími v MOS součástkách je implementováno odečítání kalibrační hodnoty 'auto zero', kdy se změří chyby převodníku na začátku každého měření. Tuto hodnotu je možné odečíst od ostatních vzorků a tak zlepšit přesnost výsledných vzorků. Výrobce doporučuje funkci měření nuly převodníku při odečtu vzorků vždy využít. Při odečtu nulové hodnoty je možné nastavit, zda se má nula měřit i s 200kΩ rezistory, nebo bez nich (AZ_SEL=1, AZ_SEL=0).

#### *3.4.1.2 Detekce přitlačení pera*

Přerušení přitlačení pera je poskytováno při nastavení v idle módu. Vstupní pin PX1 je použit jako trigrovací pin pro generaci přerušení. Za normálního stavu je PX1 přes odpor Rpd připojen k VDD, zatímco PY2 je připojen k GND spínačem SW6. Když dojde k dotyku, tak se PX1 zkratuje s PY1 a je připojen na zem. Sestupná hrana, která je výsledkem dotyku, spustí obvod generující přerušení.

#### *3.4.1.3 Detekce oddálení pera*

– Komparační metoda

Oddálení pera je detekováno porovnáváním hodnoty vzorků. Protože při oddálení pera je hodnota vzorku vždy větší než při přitlačeném peru. Porovnává se každý vzorek s hodnotou uloženou v registru, při nižší hodnotě je generováno přerušení. Nastavení hodnoty pro porovnávání je nutné zjistit experimentálně.

– Metoda detekce rostoucí hrany

Při použití této metody funguje detekce na stejném principu jako detekce přitlačení pera. Trigger hlídá rostoucí hranu na pinu PX1. Tato možnost detekce je dostupná jen při automatickém módu vzorkování signálů X, Y, U nebo AZ, X, Y, U. Vypínače se pro detekci oddálení pera nastavují v průběhu vzorkování napětí U. Nastavení vzorkování touto metodou je možné prostřednictvím PUIS bitu v registru stavu přerušení.

#### *3.4.1.4 Kompenzace teplotní chyby*

Charakteristika A/D převodníku se mění se změnami teploty. Pro zajištění stability je nutné použít kompenzační softwarový algoritmus. Auto-kalibrační mód poskytuje nutné nastavení vypínačů tak, aby poskytl data pro teplotní kompenzaci. Auto-kalibrační mód se aktivuje nastavením ACAL bitu

a je možné jej použít pouze při vzorkování v auto AZ, X, Y módu. Při kalibraci dojde k propojení vstupu + analogově-digitálního převodníku k napájecímu napětí jednotky a – vstup je uzemněn.

Softwarová kalibrační smyčka obsahuje následující 3 kroky:

1) aktivace auto AZ, X, Y a auto-kalibračního módu, získání vzorků pro X, Y a AZ, deaktivace auto kalibračního módu a návrat do normálního vzorkovacího módu
2) v pravidelných intervalech, nebo při detekci položení pera zopakovat krok 1) k získání aktualizovaných hodnot X a Y. Při porovnání s navzorkovanými hodnotami se určí relativní změna hodnot vzorků.
3) V průběhu normálního vzorkování je nutné korigovat relativní odchylku na AZ vzorky. Tím bude kompenzovat teplotní posun zesílení A/D převodníku.

### 3.4.2 Registry ASP

ASP jednotka obsahuje osm 32. bitových registrů. Přehled registrů je na obrázku č. 14. níže. Pod stejným názvem, jsou registry používané v hlavičkovém souboru ovladače.

| Description | Name | Address |
|---|---|---|
| ASP Control Register | ASP_ACNTLCR | 0x00215010 |
| Pen A/D Sample Rate Control Register | ASP_PSMPLRG | 0x00215014 |
| Compare Control Register | ASP_CMPCNTL | 0x00215030 |
| Interrupt Control Register | ASP_ICNTLR | 0x00215018 |
| Interrupt/Error Status Register | ASP_ISTATR | 0x0021501C |
| Pen Sample FIFO | ASP_PADFIFO | 0x00215000 |
| Clock Divide Register | ASP_CLKDIV | 0x0021502C |
| ASP FIFO Pointer Register | ASP_FIFO_PTR | 0x00215034 |

*Obr 14 : Registry ASP jednotky*

Nejdůležitějším registrem je 'Control register' obr. 15. Ten slouží k nastavení ASP jednotky.

**ASP_ACNTLCR** Control Register **Addr 0x00215010**

| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | ASWB | ACAL | CLKEN | | SWRST | | U_SEL | AZ_SEL | | | | |
| TYPE | r | r | r | r | rw | rw | rw | r | rw | r | rw | rw | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | 0x0200 | | | | | | | |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AZE | AUTO | MOD | | SW8 | SW7 | SW6 | SW5 | SW4 | SW3 | SW2 | SW1 | | | PADE | BGE |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | r | r | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | 0x0000 | | | | | | | |

*Obr 15 : ASP Kontrolní registr*

Funkce jednotlivých bitů registrů jsou přehledně a podrobně popsány v literatuře [9].

# 4 Arm Linux na PiMX1

## *4.1 Linux [10,11]*

Linux je operační systém typu UNIX, který je volně šiřitelný. Původním autorem je Linux Torvalds, ale na současné podobě Linuxu má svůj podíl obrovské množství programátorů. Jádro Linuxu je volně šiřitelné podle pravidel GNU General Public License. To znamená, že je šířen nejen zadarmo, ale také i se svými zdrojovými kódy jako Open Source. General Public Licence zajišťuje nikoliv jen to, že operační systém je šířen volně, ale také, že i všechen budoucí software, který vznikne z jeho zdrojových kódů bude šířen podle GPL. Přestože byl původně Linux psán pro architekturu i386, tak v současné době existuje pro mnoho různých platforem. Jako příklady můžeme jmenovat architektury ALPHA, MOTOROLA 6800, Sun SPARCS, PowerPC, MIPS, ARM a další. Jeho hlavními výhodami jsou nulová cena, dostupnost zdrojových kódů, relativní bezpečnost a stabilita, dnes již poměrně široké portfolium volně šiřitelných programů a také podpora zajištěná rozsáhlou linuxovou komunitou. API Linuxu je plně kompatibilní s POSIX normou.

Linux kernel (v práci používám také český překlad 'jádro Linuxu') je část operačního systému, která zajišťuje kontrolu hardwaru, spravuje soubory, běžící procesy . Společně s ostatními částmi operačního systému Linux tvoří kernel distribuce, které jsou plnohodnotnými operačními systémy. Distribuce často obsahují mnoho vestavěných uživatelských programů a také grafické prostředí.

14. března 1994 je datum, kdy vyšla verze 1.0, čímž bylo ukončeno období beta testování. V systému jsou samozřejmě stále chyby, ale díky 'Open development model' jsou všechny nové verze uvolněny bez ohledu na to, v jaké fázi kvality se kód nachází. Aby bylo veřejnosti jasné v jakém stavu vývoje se kód nachází, je platné dané schéma zveřejňování. Stabilní a testovací (developerské) verze jsou rozlišeny číslováním. Verze Linuxu jsou označeny číslem ve tvaru n.x.y, pokud číslo x je sudé, pak se jedná o stabilní verzi. Při opravách chyb se inkrementuje číslo y. Pokud číslo x je liché, tak se jedná o 'beta-quality' vydání, která jsou určena jen pro developery, protože mohou být nestabilní a může docházet k pádům systému. Nové vlastnosti, rysy (features) jsou přidávány během vývoje stále. Jednou za čas, po otestování a odladění stability, je vývojová verze podle své stability, zmrazena jako nová 'stable' verze. Vývoj pak pokračuje na nové vývojové verzi jádra. Je nutné zmínit, že i vývojové verze jádra jsou relativně robustní. Stabilní verze se liší hlavně tím, že se vlastnosti jádra nemění tak často jako u vývojové verze.

Současná stabilní verze je 2.6.25 (ke dni 9. května 2008), zdrojové kódy nejnovějších aktuálních verzí – stabilní i vývojové větve je možné stáhnout z http://www.kernel.org.

Ve zdrojových kódech jsou obsaženy také soubory, které mají pomoci vývojářům a nováčkům s konfigurací a kompilací jádra, také obsahuje soubor se změnami, které je třeba reflektovat při upgradu na novou verzi jádra.

Vývoj linuxového jádra, na rozdíl od vývoje většiny komerčních softwarů, probíhá otevřeně a distribuovaně, ne uzavřeně a centralizovaně. To znamená, že mnoho vývojářů přispívá do zdrojového kódu jádra. Aktuální vývojová verze jádra je vždy veřejná a kdokoli ji může používat. Výhodou tohoto modelu je, že pokud je přidána nová funkcionalita, jenž obsahuje chybu, tak je díky velkému množství vývojářů v komunitě chyba rychle nalezena a opravena (často jsou chyby opravené v intervalech hodin).

Naopak uzavřený centralizovaný model funguje tak, že určitou část kódu píše jeden člověk či tým a finální vydání kódu je uskutečněno jakmile se vývojový tým domnívá, že splňuje dané předpoklady. Tento model často vede k dlouhým intervalům mezi vydáním, způsobuje dlouhé doby na opravy chyb a pomalý vývoj produktu.

Z hlavních vlastností operačního systému uvádím jen ty, které jsou dle mého názoru, důležité:

- podpora multitaskingu – vlastnost umožňující běh více programů zároveň z pohledu uživatele (na jednoprocesorových strojích je multitasking vyřešen rychlým přepínáním kontextu)
- multiuser (víceuživatelský) – více uživatelů může pracovat na stejném stroji ve stejném čase (lokálně či vzdáleně)
- multiplatformní – systém funguje na různých platformách
- multiprocesorový – SMP podpora je dostupná zatím pro architekturu Intel a SPARC
- multithreading – nativní podpora kernelu pro vícevláknové procesy
- ochrana paměti mezi jednotlivými procesy
- sdílené 'copy-on-write' stránky paměti. Více procesů může používat stejnou paměť pro svůj běh. Až v okamžiku, kdy se jeden z procesů pokusí zapsat do paměti, dochází ke zkopírování stránky paměti. Tato vlastnost zrychluje systém při vytváření nových procesů.
- Používání virtuální paměti pro swappování na pevný disk
- podpora dynamicky linkovaných sdílených knihoven (DLL)
- dostupný zdrojový kód
- více virtuálních konzolí – je možné nezávislé přihlášení na různých (až 64.) konzolích
- podpora síťových rozhraní a protokolů přímo v jádře Linuxu

## *4.2 ARM architektura [12,13]*

ARM architektura vznikla v roce 1987, kdy firma Acorn RISC Machines vyvinula RISC (*Reduced Instruction Set Computer*) procesor s názvem ARM. Arm je zkratka pro Avanced RISC Machines, což je název divize firmy Acorn, do současnosti je nevyšší řadou procesorů ARM11. Arm procesory se využívají hlavně v aplikacích pro PDA a embedded systémy. Nejznámější verzí jsou procesory StrongARM.

Mezi vlastnosti ARM architektury patří hlavně nízká spotřeba, díky které nalezl uplatnění v mobilních zařízeních. ARM procesory se staly zajímavé díky jejich výběru firmou Microsoft jako základní platformy pro mobilní operační systém a také díky licencování ARM platformy firmou Intel, který pokračuje ve vývoji platformy.

Největší výhoda procesoru spočívá v jeho RISC architektuře. Redukované instrukce jsou zpracovávány přímo hardwarem a není třeba zpracovávat mikrokód. Díky jednoduší instrukční sadě mohou být obvody jednodušší, tedy zabírají méně plochy a celková spotřeba mikroprocesoru je nižší. Malá instrukční sada je také výhodou pro kompilátor.

Díky vlastnostem procesoru je také do značné míry omezen problém predikce skoků, což zjednodušuje návrh procesoru i kompilátorů. RISC procesory mají také větší počet univerzálně použitelných registrů.

Pro ARM existuje množství specializovaných jednoúčelových operačních systémů, které slouží pro různé průmyslové aplikace. Z hlediska běžného uživatele jsou nejznámější operační systémy pro mobilní zařízení např. Palm OS, Symbian OS, Windows CE, ARM Linux.

## *4.3 Specifické vlastnosti ARM Linuxu [13,14]*

ARM Linux je portovaný systém na stroje založené na procesoru ARM. Vývojová větev je vedená Russellem Kingem s přispěním mnoha ostatních. Linux byl pro tuto architekturu adaptován v létě 1994, kdy byl portován pro Acorn A5000. ARM jádro Linuxu již bylo portované na více než 500 odlišných přístrojů od kompletních počítačů přes síťové počítače, příručních přístrojů až po vyhodnocovací desky.

Většina Linuxových distribucí pro ARM má kořeny v Debianu pro ARM. Distribuce se často zaměřují na konkrétní typ ARM zařízení (Familiar, jLiMe, OpenZaurus). My se v práci oprostíme od distribucí, protože chceme zachovat výsledný systém co nejmenší a nejuniverzálnější a za základ si vezmeme přímo na zkompilované čisté jádro Linuxu pro architekturu ARM.

V počátcích vývoje ARM Linuxu byla kompilace a konfigurace noční můrou (neexistoval cross - kompilátor pro ARM). Musel být přepsán Linux memory management systém a ovladače pro ARM zařízení. V dnešní době již Linux na platformě ARM funguje takřka plnohodnotně jako na platformě x86 a má plnou podporu v mainlainu .

# 5 Oživení desky

Na desce byl otestován systém Linux verze 2.6., ale pro účely seznámení se s deskou a pro vývoj ovladače dotykového displeje bylo nutné zkompilovat pro desku nové jádro (2.6.24). Na desce fungoval zavaděč mx1-load, který bylo možné použít k přenesení zkompilovaného jádra a root-file systému do paměti desky. Komunikace s deskou probíhala prostřednictvím rozhraní RS232 a samotný přenos dat přes USB rozhraní. Protože tato metoda bootování desky byla časově náročná, tak se nejprve přestal přenášet root filesystém. Nyní se pracuje se soubory uloženými na hostitelském počítači. Pomocí NFS (Network File System) se přenáší pouze soubory, které se aktuálně používají. Deska s počítačem komunikuje přes síťové rozhraní. Protože i přenášení kernelu Linuxu trvalo nějakou dobu, k velkému zrychlení došlo po přehrání bootloaderu na U-Boot. Ten obsahuje jednoduché prostředí, které je možné nastavit na bootování jádra, které je přeneseno síťovým rozhraním prostřednictvím protokolu TFTP. V této kapitole se pokusím objasnit postup při práci s programy, s deskou i nastavení hostitelského počítače.

## *5.1 Kompilace jádra [15, 16, 17, 18]*

Při oživování Linuxu na embleded zařízení se nabízejí dvě možnosti řešení. První je volba existující distribuce a její upravení pro potřeby konkrétního zařízení. Výhodou tohoto postupu je, že vycházíme z distribuce, jenž většinou obsahuje všechny programy, které bychom mohli potřebovat. Nevýhod je však více, distribuce jsou často navrženy pro stolní počítače, nebo notebooky, tedy i386 platformu a proto je výběr balíčků přizpůsoben těmto počítačům. Druhou nevýhodou je, že distribuce obsahují spoustu pro embleded systémy nepotřebných balíčků – spoustu věcí z distribuce bychom nevyužili a zbytečně zvyšují velikost výsledného systému a také konfigurační skripty jsou příliš velké a zbytečně složité. Třetí nevýhodou je nutnost znalosti balíčků distribuce, aby je bylo možné vyřadit, a vzhledem k tomu, že bychom pravděpodobně museli vyřazovat většinu aplikací, tak by byla nutná velmi dobrá znalost distribuce.

Proto se častěji volí druhá metoda přizpůsobení Linuxu. Ta vychází z čistého linuxového jádra, které postupně obohacujeme o programy, jenž jsou pro náš systém důležité. Tím docílíme malé velikosti výsledného systému, lepší přehlednosti a porozumění systému a také malých a jednoduchých konfiguračních skriptů.

Samotná kompilace kernelu Linuxu se skládá z několika hlavních kroků:

1) Získání zdrojových kódů
2) Úprava zdrojových kódů danému hardwaru (pokud není jeho podpora přímo v mainlainu kernelu)
3) Konfigurace parametrů kernelu
4) Kompilace kernelu
5) Přenos výsledného image kernelu do embleded zařízení

### 5.1.1 Získání zdrojových kódů

Zdrojové kódy čistého jádra (nepozměněného žádnou distribucí) je možné získat ze serveru http://kernel.org. Zdrojové kódy je možné stáhnout buď zabalené kompletní zdrojové kódy o velikosti zhruba 50 MB, nebo, jsme-li majiteli zdrojových kódů předchozí verze jádra, tak je

možné stáhnout pouze záplatu – patch. Patche se vydávají vždy z předchozí na aktuální verzi, takže například pokud jsme 2 verze pozadu, tak je nutné ve správném pořadí aplikovat 2 patche.

Zdrojové kódy kernelu rozbalíme například pro verzi jádra 2.6.11.12 příkazem

**tar jxf linux-2.6.11.12.tar.gz**

Tar je archivní program, který se používá pro komprimaci a dekomprimaci souborů. Je použitých několik parametrů: j – soubory jsou zabalené kompresí bzip2, x – extract naším cílem je rozbalit soubory, f – přejeme si rozbalovat ze souboru, který uvádíme, pokud bychom parametr f nezadali, tak tar bude očekávat komprimovaná data na standardním vstupu.

Pokud stáhneme pouze patch, tak v adresáři se zdrojovými kódy jádra jádro patchujeme příkazem **bzcat .. /patch-2.6.11.12.bz2 | patch -p1**

Příkaz bzcat rozbalí zadaný soubor s kompresí bzip2 na vstup programu patch, který přepíše obsah změněných souborů. Parametr 'p' udává kolik se může lišit cesta k souboru, který je v patch souboru uvedený. Všechny patche kernelu jsou dělané tak, aby fungovaly pro p1, tzn. je možný rozdíl o max 1 adresář.

### 5.1.2 Úprava zdrojových kódů

Patche se nepoužívají jen k upgradu zdrojových kódů na aktuální verzi jádra, ale mohou být použity také pro jinou úpravu zdrojových kódů. V mém případě se jednalo o použití patchů z darcs repositáře Ing. Pavla Píši - http://rtime.felk.cvut.cz/repos/ppisa-linux-devel/. Soubory je ideální stáhnout přímo nástrojem darcs příkazem

**darcs get http://rtime.felk.cvut.cz/repos/ppisa-linux-devel/**.

Patche upravují zdrojové kódy kernelu pro desku PiMX1. Protože se jedná o specifický hardware a některé patche jsou konkrétní přímo pro implementaci desky firmou PiKRON, tak není možné, aby byly v mainstreamu kernelu, protože úprava se hodí jen pro malé množství desek.

Jedná se o patche, které jsou nutné pro správnou funkci hardwaru PiMX1. Jelikož to není zaměřením mé práce, tak nebudu jednotlivé patche rozebírat. Pouze se stručně zmíním o tom, jak se patch tvoří a jak vypadá.

Patch soubor vytvoříme příkazem „**diff -Nru old new > zmena.patch“**, příkaz diff porovnává obsah zadaných souborů. Parametr **N** říká, že pokud se jedná o nový soubor, tak se program diff chová jako by nový soubor existoval i ve starém adresáři, ale byl prázdný. Parametr 'r' v případě porovnávání adresářů zajistí rekurzivní porovnávání podadresářů a parametr u zajistí unifikovaný formát výstupu – výstup bude mít podobu patche (s označením +, - tak jak je popsáno níže). Stručný návod pro tvůrce patchů můžeme najít v manuálových stránkách příkazu patch (**man patch**).

Ukázka ze souboru pimx1-board-nand.patch:

```
17 --- linux-2.6.24.orig/drivers/mtd/nand/Kconfig 2008-02-11 06:51:11.000000000 +0100
18 +++ linux-2.6.24/drivers/mtd/nand/Kconfig 2008-03-02 22:09:28.000000000 +0100
19 @@ -272,6 +272,12 @@
20
21 If you say "m", the module will be called "cs553x_nand.ko".
22
23 +konfigurace MTD_NAND_PIMX1
24 + trestati "NAND Flasar on PiMX1 board"
25 + tristateon MTD_NAND && MACH_PIMX1
26 + help
27 + Enable NAND Flash suppoert for PiMX1 equipped by K9K8G08U0A
28 +
29 config MTD_NAND_AT91
30 bool "Support for NAND Flash / SmartMedia on AT91"
31 depends on ARCH_AT91
```

Každý patch soubor obsahuje vždy názvy souborů, které se mění, zde například vidíme na řádku 17 a 18, jaké dva soubory jsou porovnávány. Původní soubor je označen --- a nový upravený soubor je označen +++. Dále patch obsahuje informaci o tom, které řádky souboru se mění, to je označeno @@ - <první řádek změny ve starém souboru>, <počet upravovaných starých řádek> +<první řádek v novém souboru>, <počet řádek v novém souboru> @@

Samotná změna je ohraničena třemi řádky před změnou a třemi řádky po změně (pokud existují). To umožňuje identifikovat změnu i pokud se změní čísla řádek v souboru. Samotná změna je označena - pro odstraněný řádek, + pro přidávaný řádek. Změní-li se například jen část řádku, pak je z důvodu přehlednosti a jednoduchosti starý řádek odstraněn a vložen nový upravený (standardní chování příkazu diff).

Pro konkrétní typ desky, kterou mám k dispozici, bylo nutné ještě ve zdrojových kódech jádra udělat menší úpravu z důvodu použití barevného displeje. Jednalo se o nastavení registru PCDR zajištující nastavení systémové frekvenční děličky. Před změnou měl displej špatnou obnovovací frekvenci, což se projevovalo vodorovnými pruhy. Druhou změnou byla úprava velikosti paměti pro data displeje.

**Pimx1.board-lcd-pcdr.patch :**

```
---
 arch/arm/mach-imx/pimx1-lcd.c    |    2 ++
 include/asm-arm/arch-imx/pimx1.h |    2 +-
 2 files changed, 3 insertions(+), 1 deletion(-)

Index: linux-2.6.24/arch/arm/mach-imx/pimx1-lcd.c
===================================================================
--- linux-2.6.24.orig/arch/arm/mach-imx/pimx1-lcd.c
+++ linux-2.6.24/arch/arm/mach-imx/pimx1-lcd.c
@@ -325,12 +325,14 @@ void __init pimx1_init_lcd(void)
 {
          printk(KERN_INFO "PiMX1 display registered for type %s\n", pimx1_lcd_type);
          if(!strncmp(pimx1_lcd_type, "tft", 3)) {
+                   PCDR &= ~0x000000f0;
                    set_imx_fb_info(&pimx1_tft_16bpp_fb_info);
                    pimx1_lcdbufen_pin = PIMX1_LCDBUFEN_p;
                    pimx1_lcdbufen_pol = 0;
                    /* Disable display negative power source to prevent damage */
                    imx_gpio_direction_output(pimx1_lcdbufen_pin, 1);
          } else if(!strncmp(pimx1_lcd_type, "col", 3)) {
+                   PCDR &= ~0x000000f0;
                    set_imx_fb_info(&pimx1_color_8bpp_fb_info);
                    pimx1_lcdbufen_pin = PIMX1_LCDBUFEN_p;
                    pimx1_lcdbufen_pol = 0;
Index: linux-2.6.24/include/asm-arm/arch-imx/pimx1.h
===================================================================
--- linux-2.6.24.orig/include/asm-arm/arch-imx/pimx1.h
+++ linux-2.6.24/include/asm-arm/arch-imx/pimx1.h
@@ -39,7 +39,7 @@
 # define PIMX1_FLASH_SIZE             0x02000000
 #endif

-#define IMX_FB_PHYS                   (0x00300000 + 0x10000)
+#define IMX_FB_PHYS                   (0x00300000 + 0x08000)

 #define CLK32 32768
```

### 5.1.3 Konfigurace parametrů kernelu

Jádro Linuxu je možné ze stejných zdrojových kódů zkompilovat na mnoho různých platforem s různými hardwarovými konfiguracemi, proto je nutné ještě před kompilací jádro nakonfigurovat. Konfiguruje se pro příslušný hardware a vybírají se služby, pro které si přejeme mít v jádře podporu. Konfigurace kompilace se provádí pomocí souboru .config. Zde jsou volby s s prefixem CONFIG_ například (CONFIG_ARM=y ), volba může být 'y' ... yes - danou vlastnost chceme, aby výsledné jádro mělo, 'm' ... module – vlastnost si přejeme kompilovat samostatně jako modul (je možné jen u některých položek) nebo 'n' .. no – vlastnost, nebo ovladač si nepřejeme kompilovat.

Možných nastavení je, vzhledem ke komplexnosti Linuxového jádra, velmi mnoho. Naštěstí existují nástroje, jenž konfiguraci kompilace velmi usnadňují. Po zadání příkazu „**make config“** se spustí průvodce. Ten s námi projde všechny možnosti nastavení prostřednictvím jednoduchých dotazů, na které se nám nabízí možné odpovědi např. „BSD Process Accounting (BSD_PROCESS_ACCT) [N/y/?]“. První zvýrazněná odpověď je doporučená. Pokud místo odpovědi pouze potvrdíme klávesou Enter, pak vybereme doporučenou možnost.

Další možností je použít „**make menuconfig“**, což je základní grafické menu, které běží i bez grafického prostředí X windows. „**make xconfig**“ spustí konfigurační nástroj postavený na X windows a grafické knihovně Qt, **„make gconfig“** využívá knihovnu Gtk.

Pak je ještě několik možností automatického nastavování, „**make oldconfig**“ nakonfiguruje kompilaci podle starého, nebo ručně editovaného souboru .config a ptá se jen na položky nové nebo v souboru .config neobsažené. Stejnou funkci jako oldconfig má „make silentoldconfig“, který však nezobrazuje již zodpovězené otázky v souboru .config. „make defconfig“ vytvoří .config podle defaultní konfigurace pro danou architekturu. „make allyesconfig“ vyplní tolik možností 'Y', kolik je pro danou architekturu možné. „make allmodconfig“ a „make allnoconfig“ se snaží naopak vyplnit co nejvíce možností M, nebo N. Pro testovací účely existuje ještě příkaz **„make randconfig“**, jenž vyplní položky souboru .config náhodně. Pokud chceme vytvořené jádro běžně používat, pak se ale rozhodně doporučuje vyhnout náhodnému výběru a zvolit variantu, kdy máme možnost jednotlivé položky projít.

Pro desku PiMX1 je nejdůležitější zvolit architekturu ARM (ARCH=arm). Poté prostřednictvím některého nástroje např. „make menuconfig“ vybrat System type -> ARM system type – IMX, IMX implementations – pimx1 a support ARM920T procesor. Ostatní výběr volíme podle potřebné hardwarové podpory.

### 5.1.4 Kompilace kernelu

Po konfiguraci se kernel zkompiluje příkazem **„make“**. Pokud však chceme zdrojové kódy používat pro více různých zařízení, nebo si chceme zdrojové kódy zachovat čisté pro budoucí použití, tak je vhodné si vytvořit v adresáři se zdrojovými kódy vlastní adresář, např. _build s podadresářem pro naši kompilaci, v našem případě to vytvoříme strukturu /linux-2.6.24/_build/arm. Jako nejlepší možnost kompilace kernelu do vlastního adresáře bylo vytvořit si v daném adresáři GNUmakefile, který má přednost před Makefile, ale zároveň není kompilátorem jádra automaticky upravován. Zde je důležité udat cestu ke zdrojovým kódům kernelu, výstupní adresář, architekturu a cros_compile parametry. Zde je ukázka funkčního GNUmakefile, tímto je zkompilované linuxové jádro pro desku PiMX1.

```
VERSION = 2
PATCHLEVEL = 6

KERNELSRC    := ~/linux-2.6.24
KERNELOUTPUT := ~/linux-2.6.24/_build/test

MAKEFLAGS += --no-print-directory

.PHONY: all $(MAKECMDGOALS)

ARCH            = arm
#CROSS_COMPILE  ?= arm-unknown-linux-gnu-
CROSS_COMPILE   = arm-linux-

all:
<------>$(MAKE) ARCH=$(ARCH) $(if $(CROSS_COMPILE),CROSS_COMPILE=$(CROSS_COMPILE)) -C $(KERNELSRC) O=$(KERNELOUTPUT)

Makefile:;

$(filter-out all Makefile,$(MAKECMDGOALS)) %/:
<------>$(MAKE) ARCH=$(ARCH) $(if $(CROSS_COMPILE),CROSS_COMPILE=$(CROSS_COMPILE)) -C $(KERNELSRC) O=$(KERNELOUTPUT) $@
```

Výsledkem kompilace je několik souborů, které najdeme v adresáři /arch/<platforma>/boot v případě architektury arm je to /arch/arm/boot. Pro arm platformu jsou zde dva soubory image jádra, je to Image a zImage. zImage je komprimovaná varianta binárního souboru jádra Linuxu. Pro případ bootování pomocí bootloaderu U-Boot je výhodnější jádro zkompilovat do podoby uImage, který je velmi podobný zImage, ale U-Boot jej podporuje jako primární linuxový obraz. uImage vytvoříme příkazem „**make uImage**“.

### 5.1.5 Přenos do výsledného zařízení

Přenos jádra do výsledného zařízení je možný několika způsoby. V případě desky PiMX1 jsem nejdříve k přenosu image kernelu používal mx1-load, který přenášel jádro po USB rozhraní. Postupem byl však nahrazen U-Bootem, který umožňuje přenos kernelu přes ethernetové rozhraní, čímž se velmi urychluje bootování systému. O obou zavaděčích se podrobněji zmíním. Bootloader je spouštěn hardwarem, většinou je uložen v paměti, kterou hardware po své inicializaci začne provádět, čímž se spustí bootloader. Ten nahraje image kernelu do paměti RAM a poté jej spustí. Kernel se sám již rozbalí a začne svou inicializaci.

Při svém spouštění kernel akceptuje parametry, kterými můžeme ovlivnit chování spuštěného jádra.

Bootovacími parametry můžeme nastavit velké množství parametrů jádra od nastavení chování ACPI, přes nastavení PCI sběrnice až po nastavení chování bezpečnostního SELINUXu. Úplný výčet parametrů nalezneme v dokumentaci Linuxu v souboru Documentation/kernel-parameters.txt.

Pro naše použití je nedůležitější možnost nastavení načtení root-filesystému po síti přes NFS (root=/dev/nfs, ip=host:server:gateway:netmask:hostname:device, ntsroot=NFS_server:adresář na serveru), potom je možnost měnit hlavní konzoli systému. Tou může být buď sériový terminál (console=ttySMX0), nebo LCD displej při neuvedení tohoto parametru.

## *5.2 mx1-load Bootloader*

MX1-load je program, který byl v roce 2004 napsán Ing. Pavlem Píšou. Ke komunikaci s deskou používá rozhraní RS232. Přenos image kernelu a root filesystému následně probíhá přes USB rozhraní. Root filesystém musel být zkompilován jako 'cramfs'. Po přenesení všech dat se provedl skok na místo v paměti, kde začínal kód jádra a systém začal bootovat. Přenos jádra a filesystému trval však příliš dlouho, proto se nejprve eliminoval přenos root filesystému. To bylo možné díky exportu filesystému z hostitelského počítače přes NFS. Následně byl zavaděč nahrazen zavaděčem 'Das U-Boot'.

## *5.3 U-Boot [19, 21]*

The universal bootloader 'Das U-Boot' je bootloader vyvíjený pod licencí GPL. Je schopen nahrát boot Image do paměti (RAM, flash) a poté příkazem **'bootm'** spustit tento Image. Mezi schopnosti U-Boot loaderu patří inicializační otestování architektury a OS, testování cheksumu, rozbalování komprimovaného Image, zkopírování do paměti na určené místo, zadání bootovacích argumentů. Do paměti může být operační systém zkopírován mnoha způsoby (přes sériový port, Ethernet, z hardisku či CDROM, z paměťové karty, z USB Mass Storage Device, NAND paměti s filesystémem JFFS2 či s PCI sběrnici).

Ovládání je dostupné ze sériového rozhraní, kde lze U-Boot ovládat pomocí CLI příkazů (simple command line interface) nebo HUSH Shellu (z balíku Busybox). Konfigurační parametry je možné uložit do prostředí programu (enviroment), které je možné ukládat do paměti hardwarového zařízení. Příkazem '**printenv**' se vypíše aktuální nastavení pracovního prostředí, jednotlivé položky můžeme upravovat příkazem '**setenv <proměnná> <hodnota>**' a příkazem '**saveenv**' se nastavení uloží do paměti zařízení.

Zde uvádím nastavení pracovního prostředí U-Bootu pro desku PiMX1 z interní dokumentace firmy PiKRON

```
setenv ethaddr 80:81:82:83:84:85
setenv hostname mx1ant
setenv ipaddr 192.168.2.31
setenv serverip 192.168.2.51
setenv netmask 255.255.255.0
setenv gatewayip 192.168.2.35
setenv eval_bootfile 'setenv bootfile ${hostname}/uImage'
setenv eval_rootpath 'setenv rootpath ${serverip}:/tftpboot/${hostname}'
setenv ipautoconf none
setenv baudrate 19200

setenv kern_ramaddr 08001000
setenv kern_flashaddr 10040000

setenv nfs_bootargs 'setenv bootargs ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:${eth0}:${ipautoconf}
          rw nfsroot=${rootpath} ${kernelparams}'
setenv boot_tftp_nfs 'run eval_rootpath; run nfs_bootargs; tftp ${kern_ramaddr}; bootm ${kern_ramaddr}'
setenv boot_flash 'setenv bootargs console=ttySMX0,${baudrate}n8 root=31:4 rootfstype=jffs2; bootm 10040000'

setenv bootcmd 'dmpower 1; run eval_bootfile; run boot_tftp_nfs'
```

Postupně se nastavují jednoduché proměnné nutné pro vytvoření síťového spojení, nastavení sériové konzole, místa v paměti pro uložení načteného Image kernelu. Ty se později vyhodnocují ve složitějších výrazech. Na posledním řádku je již nastavení bootovacího příkazu, který zapne síťovou kartu a spustí vyhodnocení dvou výrazů, z nichž druhý nakonec spustí bootování jádra, které zkopíruje přes TFTP službu ze serveru, jehož IP adresa je uvedena v proměnné 'serverip' .

U-Boot je užitečným nástrojem, protože umožňuje širokou škálu nastavení a různé způsoby zavedení jádra. Také umožňuje základní otestování hardwaru a zaváděného obrazu, před samotným bootováním. Přenos jádra pomocí UDP/IP protokolu je také velmi rychlý.

## *5.4 Podpůrné programy*

Tato kapitola je stručným přehledem použitých programů. Pokud bylo u programu nutné nějaké netriviální nastavení, tak jej uvedu.

### 5.4.1 GNU Compiler Collection - GCC [20,21]

GCC je sada kompilačních nástrojů, která umožňuje překlad různých programovacích jazyků. Původní název projektu představeného v roce 1987 byl pouze GNU C Compiler, ale postupem času se překladač rozšiřoval o další jazyky. Nejprve C++, později Fortran, Pascal, Java a další. Jeho vývoj je zaštiťován projektem GNU, kterému také slouží jako oficiální kompilační nástroj. Je také standardním kompilačním nástrojem pro mnoho moderních operačních systémů, jako je Linux, BSD, Mac OS X.

GCC obsahuje podporu pro mnoho druhů zařízení, proto kód, který je odladěn pro GCC kompilátor, má mnohem vyšší pravděpodobnost přenositelnosti, než kód odladění pro nativní kompilátor daného procesoru.

Překlad je prováděn v několika krocích, nejprve se spustí program gcc, který vyhodnotí své vstupní parametry, pro každý soubor zvolí jazykový kompilátor, kterým bude soubor přeložen. Po kompilaci kódů spustí assembler a linkovací program, čímž se vytvoří spustitelný binární soubor.

Nástroj GCC je možné použít i pro 'cross-kompilaci', která byla nutná i v našem případě. Jedná se o techniku, kdy cílový počítač kompilace je výpočetně velmi slabý, a proto by kompilace na něm trvala velmi dlouho, nebo by nebyla z důvodu hardwarových nedostatků vůbec možná (například díky malé paměti zařízení). Při cross-kompilaci se provádí přeložení programu na výpočetně silnějším počítači pro hardware a knihovny, které budou dostupné na cílovém zařízení. Pro tento účel je nutné mít nainstalovanou fungující zkompilovanou kopii 'binutils' pro cílovou platformu. (Binutils je sbírkou základních nástrojů pro kompilaci. Mezi dva nejdůležitější nástroje, které binutils obsahuje patří asembler kompilátor a linkovací nástroj.) Další potřebnou komponentou je standardní C knihovna pro cílovou platformu. Při cross-kompilaci Linuxových ovladačů je na developerském počítači také nutné mít zdrojové kódy Linuxového jádra, pro které ovladač vyvíjíme.

Při nastavování cross-kompilace je nutné rozlišovat několik pojmů, které je třeba vždy správně nastavit. 'Build platform' je platforma, na které budeme kód kompilovat. 'Host platform' nebo 'target platform' je cílová platforma, na které zkompilovaný kód má být spustitelný.

### 5.4.2 NetworkNetwork File System - NFS [22]

'Network File System' je přenosový protokol, který umožňuje sdílení dat prostřednictvím síťového spojení. Distribuce dat je realizována jako klient-server transakce, kdy jeden z počítačů data sdílí (server) a druhý počítač tyto data používá (klient).

V našem případě jsme NFS použili pro export root-filesystému pro desku PiMX1. Jelikož je sdílení realizováno po Ethernetové síti a navíc se přenášejí jen soubory, které jsou aktuálně potřeba, tak je použití NFS mnohem rychlejší, než transport celého root-filesystému do flash paměti desky.

Při požadavku na akceptování NFS root-filesystému kernelem Linuxu, je nutné při jeho kompilaci zapnout podporu síťových služeb a NFS systému. Potřebnou podporu NFS jako root-filesystému zajistíme prostřednictvím těchto konfiguračních parametrů:

```
CONFIG_NFS_FS = Y
CONFIG_IP_PNP = Y
CONFIG_ROOT_NFS = Y
```

Dále je nutné cestu k souborovému systému zadat jako bootovací parametry kernelu. Potřebné nastavení je uvedeno výše v kapitole U-Boot, kde jsou nastavovány bootovací parametry v proměnné s názvem 'bootargs'.

Na hostitelském serveru je třeba mít nainstalovaný a aktivní NFS server. Konfigurace serveru se provádí prostřednictvím souboru /etc/exports. Zde pro náš účel stačí přidat jeden řádek s udáním exportovaného adresáře, IP adresou (nebo IP rozsahem), jenž si může tento adresář připojit a volitelnými parametry:

```
/tftpboot/mx1ant 192.168.2.31 (rw, no_root_squash)
```

Po dokončení nastavení restartujeme NFS server příkazem '**/etc/init.d/nfs-kernel-server restart'.**

### 5.4.3 Trivial File Transfer Protocol - TFTP [23]

'Trivial File Transfer Protocol' je jednoduchý protokol, který umožňuje přenos souborů prostřednictvím velmi jednoduchého protokolu. Je velmi jednoduchou formou protokolu FTP a definován je od roku 1980. Protože je velmi jednoduchý, tak jeho implementace využívá jen malé množství paměti. Časté využití je pro přenos bootovacích souborů do aktivních síťových prvků, malých embedded zařízení, nebo například pro komunikaci vzdáleného terminálu X Window System.

Po povolení UDP portu číslo 69 ve firewallu hostitelského počítače stačí pouze spustit TFTP daemona s parametrem adresáře, který chceme sdílet.

**'/usr/sbin/in.tftpd -l -r blksize -s /tftpboot'**

### 5.4.4 Minicom [24]

Minicom je komunikační program, který používám při komunikaci s deskou prostřednictvím sériového rozhraní. Vzhledem k tomu, že hostitelský počítač není vybaven sériovým rozhraním, komunikace probíhala přes USB-RS232 převodník, s jehož ovladači však nebyl v Linuxu žádný problém. Nastavení programu minicom je třeba provádět jako superuživatel. Konfiguraci si můžeme uložit pod vlastním názvem a poté se připojovat z běžného uživatelského konta. Po spuštění příkazu **'minicom -s mx1'** je nutné upravit nastavení sériového portu – v mém případě na /dev/ttyUSB0. Protože se jedná primárně o program spolupracující s modemem, je ještě nutné smazat nastavení v oddílu 'Modem a vytáčení' - odstranit veškeré posílané znaky po inicializaci – položka A.

S tímto nastavením je již minicom připravený zastoupit roli sériové systémové konzole. Po spuštění programu minicom '**minicom mx1**' (uživatel, pod kterým minicom běží, musí mít přístup k zařízení ttyUSB0) a připojení převodníku k sériovému rozhraní desky se nám po zapnutí napájení zobrazí inicializační výpis U-Bootu.

### 5.4.5 GIT [25]

Git je distribuovaný systém správy verzí vyvíjený pod licencí GPL. Původním autorem je Linus Tovalds. Je navržený pro správu velkých projektů, ale také je jej možné využít pro malé osobní repositáře. Z velkých projektů, které jsou spravovány v Gitu, je možné jmenovat Linuxové jádro, WINE nebo X.org.

Výhodou Gitu je možnost distribuovaného vývoje aplikací. Každá kopie repositáře je samostatně fungujícím repositářem. Tzn., že je možné dělat revize, prohledávat historii i bez přístupu k centrálnímu serveru. Dalšími výhodami jsou jeho rychlost i při práci s velkými projekty, podpora šifrovaných podpisů a navržení podle Unixové tradice. Git je souborem jednoduchých nástrojů psaných v jazyce C a množství skriptů.

Pro naše použití využijeme pouze některé z bohaté nabídky příkazů

'**git add'** <soubor> **-** přidá soubor do systému správy verzí

'**git checkout'** – zkontroluje aktuálnost verze

'**git commit'** – uloží změny do repositáře

'**git diff'** – ukáže porovnání mezi verzemi

'**git fetch'** – stáhne repositář s zadaného umístění

'**git log'** – vypíše logy commitů

'**git merge'** – spojí různé vývojové verze

'**git pull** – upgraduje vzdálené úložiště lokálním repositářem

'**git revert**' – vrátí provedený commit

# 6 Vývoj ovladače pro Linux [26, 15]

## *6.1 Možnosti integrace s jádrem*

Při psaní linuxového ovladače existují dvě možnosti začlenění ovladače do jádra Linuxu. První možností je ovladač zakompilovat přímo do monolitického jádra. Tato metoda má však několik nevýhod. Nejmarkantnější z nich je zvýšení velikosti kernelu i v situaci, kdy daný ovladač nepotřebujeme, nebo nechceme využívat. Nevýhodou z vývojářského pohledu je nutnost při změně kódu ovladače překompilovat celé jádro a znovu nabootovat zařízení. Vzhledem k tomu, že kompilace složitějšího jádra (nastaveného tak, aby podporovalo větší množství služeb a ovladačů) může trvat i několik hodin, tak je to metoda pro vývoj ovladače přímo nedoporučená.

Druhou možností je ovladač vyvíjet jako modul, který se při své potřebě začlení do jádra a pak je možné jej opět odpojit. Tento postup je možné využít pro všechny ovladače, které nejsou pro systém nezbytné – tzn. je možné nabootovat i bez jejich přítomnosti v jádře. Vzhledem k tomu, že ovladač dotykového displeje je využitelný až jako zařízení určené pro lidskou interakci, tak není problém ovladač nahrát až po startu systému. Kromě toho, že můžeme ovladač k jádru připojit/odpojit podle aktuální potřeby, má vývoj ovladače jako modulu ještě další výhody. Ovladače ve formě modulů pomáhají udržovat minimální velikost jádra – je možné šetřit pamět, pokud není zařízení zrovna využívané a přitom ovladač neztrácí žádnou z výhod přístupu k celému kernelu. Vývoj modulu je usnadněn také tím, že jeho kompilace je možná samostatně – je jen nutno uvést při kompilaci cestu ke zdrojovým kódům kernelu, pro který je modul určen. Také je možné mezi moduly vytvářet závislosti. To znamená, že například modul USB kamery může požadovat nahrání modulu ovladače USB sběrnice. Tím je možné zajistit, že není nezbytné mít některé moduly nahrány stále, ale při odebrání posledního modulu, který jej využíval, mohou být také odebrány.

K připojení modulu k jádru slouží příkaz '**insmod <modul>**', pro odebrání '**rmmod <název modulu>**'. Výpis aktuálně připojených modulů k jádru získáme příkazem '**lsmod**'.

Pokud má ovladač fungovat jako modul, musí obsahovat inicializační funkci - '__init' a funkci, jenž je volána při odebírání modulu - '__exit'.

Zde uvádím příklad jednoduchého modulu, který pouze vypisuje informace o svém připojení a odpojení. Kromě dvou zmiňovaných funkcí jsou zde ještě potřebné importy a také makra, zajišťující připojení informací o autoru, licencování modulu a základní popis modulu.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static int __init mujmodul_init(void)
{
          printk(KERN_ERR "Modul byl pripojen k jadru Linuxu.\n");
          return 0;
}

static void __exit mujmodul_exit(void)
{
          printk(KERN_ERR "Modul bude odpojen.\n");
          return;
}

module_init(mujmodul_init);
module_exit(mujmodul_exit);

MODULE_DESCRIPTION("Jednoduchý modul, vypisujici informace o svem stavu.");
MODULE_AUTHOR("Autor Modulu");
MODULE_LICENSE("GPL");
```

## *6.2 Kompilace*

Při kompilaci modulu je nutné zadat adresář, ve kterém se nachází zdrojové kódy kernelu, se kterým bude modul pracovat. To je nutné z důvodu slinkování s použitými kernelovými funkcemi.

Nejvýhodnější metodou vývoje je dle mého názoru použití vývojářského prostředí, ve které můžeme aplikaci rovnou kompilovat. Většinou je vývojové prostředí schopno zvýraznit základní chyby, případně při chybách či varováních kompilace nám rovnou zvýrazní příslušný chybový řádek.

Pro můj vývoj jsem se po konzultaci s mým vedoucím práce rozhodl používat prostředí KDevelop, které umožňuje vývoj aplikací v mnoha programovacích jazycích, mimo jiné také v C/C++.

Nejjednoduší možnost kompilace souboru v operačním systému Linux je vytvoření Makefile. To je soubor, který při spuštění příkazu **'make'** v daném adresáři určuje co se bude kompilovat a jakým způsobem.

Zde je ukázka z mého Makefile souboru, který byl vytvořen aplikací KDevelop. Vzhledem k automatické generaci je soubor mírně komplikovanější, než by mohl být, ale zase je snadněji upravovatelný. Například je tu navíc proměnná TARGET, ve které se uchovává název souboru, při přejmenování pak stačí změnit pouze tento jeden řádek a nemusí se upravovat všechny řádky, na kterých je uveden výraz $(TARGET), kde by při ruční tvorbě byl napsaný název souboru. V proměnné KDIR je uložena cesta ke kernelu se kterým modul bude funkční. Makefile také umožnuje spouštět příkazy shellu, takže v mém případě, pokud ve vývojovém prostředí spustím kompilaci s parametrem test, tak jak vidíme z příkladu Makefile, nejprve se zkompiluje soubor podle default pravidel a poté se zkopíruje do adresáře, ze kterého se provádí export NFS systému. Ovladač je proto po zkompilování s parametrem test připraven rovnou na testování na desce PiMX1.

```
TARGET = mx1ts-driver
OBJS = mx1ts-driver.o
MDIR = drivers/misc

EXTRA_CFLAGS = -DEXPORT_SYMTAB
CURRENT = $(shell uname -r)
KDIR = /home/Radek/bakalarka/linux-2.6.24/_build/arm
PWD = $(shell pwd)

obj-m      := $(TARGET).o

default:
          make -C $(KDIR) SUBDIRS=$(PWD) modules

$(TARGET).o: $(OBJS)
          $(LD) $(LD_RFLAG) -r -o $@ $(OBJS)

test:     default
          -cp -v $(TARGET).ko /srv/nfs/mx1ant/root/$(TARGET).ko

clean:
          -rm -f *.o *.ko .*.cmd .*.flags *.mod.c

-include $(KDIR)/Rules.make
```

## *6.3 Práce s pamětí*

Vzhledem k faktu, že ovladač pracuje jako modul – v jádře Linuxu, není možné používat knihovní funkce jazyka C. O možnost dynamické alokace paměti však nepřicházíme. Kernel Linuxu má vlastní funkce, které nám přístup k paměti umožňují. Kapitolu rozdělím na práci s dynamicky alokovanou pamětí a I/O paměť.

### 6.3.1 Dynamicky alokovaná paměť

Dynamicky alokovaná paměť je pro ovladač, který by neměl používat statické proměnné a struktury, aby nezabíral paměť, v případě nepřítomnosti daného hardware v systému, velmi praktická. Pro alokaci paměti je možné použít funkci kmalloc, která je velmi podobná knihovní funkci malloc.

```
static inline void *kmalloc(size_t velikost, int flags)
```

kde, velikost je počet bytů, které si přejeme přiřadit a flags jsou vlastnosti, které si přejeme pro dané využití paměti vyhradit. Standardně používaným flagem je GFP_KERNEL který naznačuje, že se jedná o běžnou paměť, která může být blokována. Parametr GFP_ATOMIC slouží pro vyhrazení paměti, kterou není možné blokovat. Takovou pamět je třeba používat například v handlerech přerušení a kritických sekcích chráněných semafory či mutexy. GFP_USER slouží pro označení uživatelské paměti s nejnižší prioritou, může být blokována. K vybranému flagu můžeme ještě klasickým 'OR operátorem - | ' připojit doplňkové flagy, například __GFP_DMA, pokud požadujeme aby naše paměť byla v oblasti umožnující DMA přenosy, __GFP_ZERO pro požadavek vynulované paměti. Je možné také doplnit flag - požadavak na chování v případě neúspěšnosti operace. __GFP_NORETRY se pokusí o přiřazení paměti, pokud z nějakého důvodu selže, vrací se ihned s chybovou návratovou hodnotou do programu. __GFP_NOFAIL se naopak pokouší přiřadit pamět, dokud není přiřazení úspěšné.

Funkce kmalloc vrací ukazatel na získanou pamět. Při další nepotřebě paměti je nutné ji dealokovat, to je možné funkcí kfree, která má jediný parametr a tím je ukazatel vrácený funkcí kmalloc.

```
void kfree (const void *ukazatel);
```

Stejně jako u knihovní funkce malloc i kernelových funkcí existuje několik modifikací – kzalloc, který vrací buffer paměti vynulovaný. kcalloc( size_t n, size_t size, gfp_t flags), jehož parametr **n** určuje počet elementů pole, které budou vyhrazené standardním kmalloc.

Velmi běžné použití kmalloc je alokace určité struktury, tento trik nám umožnuje funkce sizeof(), která sama určí potřebný prostor pro strukturu. Uvádím praktickou ukázku použití kzalloc z mého kódu.

```
struct mx1ts *mts;
mts = kzalloc(sizeof(struct mx1ts), GFP_KERNEL);
kfree(mts);
```

Funkce kmalloc umožnuje alokovat pro ovladač až 128 KB paměti o velikosti $2^n$ .Pro alokování větších oblastí pamětí je nutné použít funkce alokující celé paměťové stránky jako například __get_free_page, __get_free_pages nebo vmalloc. Tak rozsáhlá oblast paměti však v případě modulu dotykového ovladače nebyla potřebná.

### 6.3.2 I/O paměť

I/O porty a I/O paměť jsou jednou z metod, která umožňuje přístup k periferiím. Při zápisu, nebo čtení I/O paměti vykonávají instrukce procesoru také přístup k reálnému hardwaru. Výhodou tohoto přístupu je možnost přistupovat k hardwaru přímo prostřednictvím operací čtení a zápisu v paměti. Při jiném řešení - navržení obsluhy hardwaru po přístupu přes speciální sběrnici se sice zrychlí práce paměti, která není zpomalována rychlostí práce periferií, ale přístup k hardwaru je složitější. U procesoru Dragonball M9328 MX1 výrobce k přístupu k ASP (Analog Signal Processing) jednotce zvolil metodu I/O paměti. Registry ASP jednotky jsou umístěny od paměti 0x00215000 a zabírají oblast paměti 0x38 bitů.

K registraci I/O paměti slouží funkce request_mem_region, podobná funkce request_region slouží k registraci I/O portu.

```
struct resource * request_mem_region( unsigned long start, unsigned long len, char *name);
```

Po zaregistrování se o úspěšnosti operace můžeme přesvědčit prostřednictvím výpisu z adresářové struktury proc, kde se požadovaná paměť objeví ve výpise i s naším požadovaným jménem.

```
~ # cat /proc/iomem
00205000-002050ff : imx-fb.0
  00205000-002050ff : IMXFB
00206000-002060ff : imx-uart.0
00207000-002070ff : imx-uart.1
00214000-002140ff : imx-mmc.0
  00214000-002140ff : IMXMCI
00215000-00215038 : mx1ts.0
  00215000-00215037 : mx1ts
08000000-09ffffff : System RAM
  08025000-082f8fff : Kernel text
  082fa000-083545ab : Kernel data
16000000-16000003 : dm9000.0
  16000000-16000003 : dm9000
16000004-16000007 : dm9000.0
  16000004-16000007 : dm9000
e0006000-e00060ff : imx-uart
e0007000-e00070ff : imx-uart
```

K opětovnému uvolnění paměti existuje funkce release_mem_region, jejíž parametry jsou stejně jako u registrace začátek a délka I/O paměti.

```
void release_mem_region( unsigned long start, unsigned long len);
```

Registrace I/O paměti je však pro její používání ještě nedostatečná. Pro umožnění přístupu ovladačů musí ovladač používat virtuální paměť, kterou může procesor obsluhovat. Proto je nutné použít ještě funkci ioremap definovanou v knihovně asm/io.h (pro použití funkce je nutné knihovnu připojit k ovladači '#include <asm/io.h>'). Tato funkce nám pro fyzickou adresu phys_addr o délce size vrátí použitelnou paměť pro operace čtení a zápisu.

```
void *ioremap(unsigned long phys_addr, unsigned long size);
```

Komplementární funkcí je iounmap, která zajistí odmapování virtuální paměti.

```
void iounmap(void *address);
```

Pro čtení virtuální paměti můžeme použít mnoho různých funkcí unsigned int ioread8(void *addr), unsigned int ioread16(void *addr), unsigned int ioread21(void *addr), unsigned int _raw_readl(void *addr) pro zápis pak funkce void iowrite8(u8 value, void *addr) (16,32) nebo void _raw_writel(u8 value, void *addr).

## *6.4 Přerušení*

Pro reakci na asynchronní události uživatele je ještě nutné zaregistrovat přerušení a vytvořit jejich handlery – funkce, které příchozí přerušení zpracovávají. Přerušení je většinou dáno hardwarovým propojením. Přerušení může být jednoúčelové, určené pro obsluhu určité události, nebo může být sdílené a sloužit například celé sběrnici.

Pro registraci přerušení slouží funkce request_irq:

```
int request_irq( unsigned int irq, irqreturn_t handler, unsigned long irq_flags, const char * devname, void *dev_id);
```

parametry jsou irq číslo daného přerušení, ukazatel na obsluhující funkci handler, možné vlastnosti irq_flagy, které mohou být IRQF_DISABLED (při příchodu přerušení jsou vypnutá ostatní přerušení na příslušném CPU, handler by měl být velmi rychlý, aby neblokoval ostatní přerušení), IRQF_SHARED (pro sdílené přerušení různými zařízeními) a IRQF_SAMPLE_RANDOM (jedná se o náhodně přicházející přerušení, které může být použito pro /dev/random a /dev/urandom). Dalšími parametry jsou jméno určené pro registraci přerušení a dev_id jednoznačný identifikátor zařízení v případě sdíleného přerušení.

O úspěšnosti registrace přerušení se můžeme také přesvědčit výpisem z filesystému /proc, kde je kromě čísla přerušení a jeho názvu uveden také počet příchozích přerušení od startu systému.

```
~ # cat /proc/interrupts
           CPU0
 26:          8         MPU  IMX-uart
 29:         38         MPU  IMX-uart
 30:          0         MPU  IMX-uart
 33:        202         MPU  mx1-ts
 35:         11         MPU  imx-mmc
 46:         30         MPU  mx1-ts
 59:       6008         MPU  i.MX Timer Tick
 60:          0         MPU  DMA
 61:          0         MPU  DMA
 84:          0        GPIO  imx_nand
155:       4006        GPIO  eth0

Err:          0
```

K odregistrování přerušení slouží funkce

```
void free_irq( unsigned int irq, void *dev_id);
```

s parametry irq čísla přerušení a jednoznačným identifikátorem dev_id sloužícím pro případ odregistrování zařízení ze sdíleného přerušení.

Ještě si dovolím zdůraznit vlastnosti handleru, mezi které by měla patřit co nejrychlejší kód, nevyužívání mutexů, semaforů a jiných funkcí, které by mohli způsobit pozastavení či spánek handleru.

Hlavička funkce handleru by měla mít tvar

```
static irqreturn_t jmeno_funkce(int irq, void *dev_id),
```

kde irq je číslo příchozího přerušení a dev_id jednoznačný identifikátor zařízení.

Jsou možné dvě návratové hodnoty IRQ_HANDLED, pokud přerušení bylo zpracováno, nebo IRQ_NONE, které oznamuje scheduleru operačního systému, že přerušení nebylo pro daný ovladač a má hledat jiný handler pro zpracování přerušení.

## *6.5 Debugování*

Při vývoji Linuxového ovladače je nutné si zvolit nástroj pro ladění chování zařízení. Jako nejjednodušší a v případě jednoduchého ovladače, kde po odladění již nebudou výpisy dále potřeba, je použít k výpisu linuxovou funkci printk, která funguje velmi podobně jako knihovní funkce printf. Výpis může mít například následující tvar:

```
printk(KERN_DEBUG "mx1 touchscreen: Touch down interrupt \n");
```

kde makro KERN_DEBUG se rozbalí do příslušné úrovně výpisu v tomto případě <7>. Úrovně systémových výpisů zajišťují jejich zobrazení podle jejich důležitosti. Loglevel, který je ještě vypisovaný na terminály, nebo systémovou konzoli, je možné nastavit přes souborový systém **proc** /proc/sys/kernel/printk.

Makra, která se rozbalují do hodnoty příslušného loglevelu jsou definována v souboru 'include/linux/kernel.h'.

```
include/linux/kernel.h

#define   KERN_EMERG          "<0>"     /* system is unusable                  */
#define   KERN_ALERT          "<1>"     /* action must be taken immediately    */
#define   KERN_CRIT           "<2>"     /* critical conditions                 */
#define   KERN_ERR            "<3>"     /* error conditions                    */
#define   KERN_WARNING        "<4>"     /* warning conditions                  */
#define   KERN_NOTICE         "<5>"     /* normal but significant condition    */
#define   KERN_INFO           "<6>"     /* informational                       */
#define   KERN_DEBUG          "<7>"     /* debug-level messages                */
```

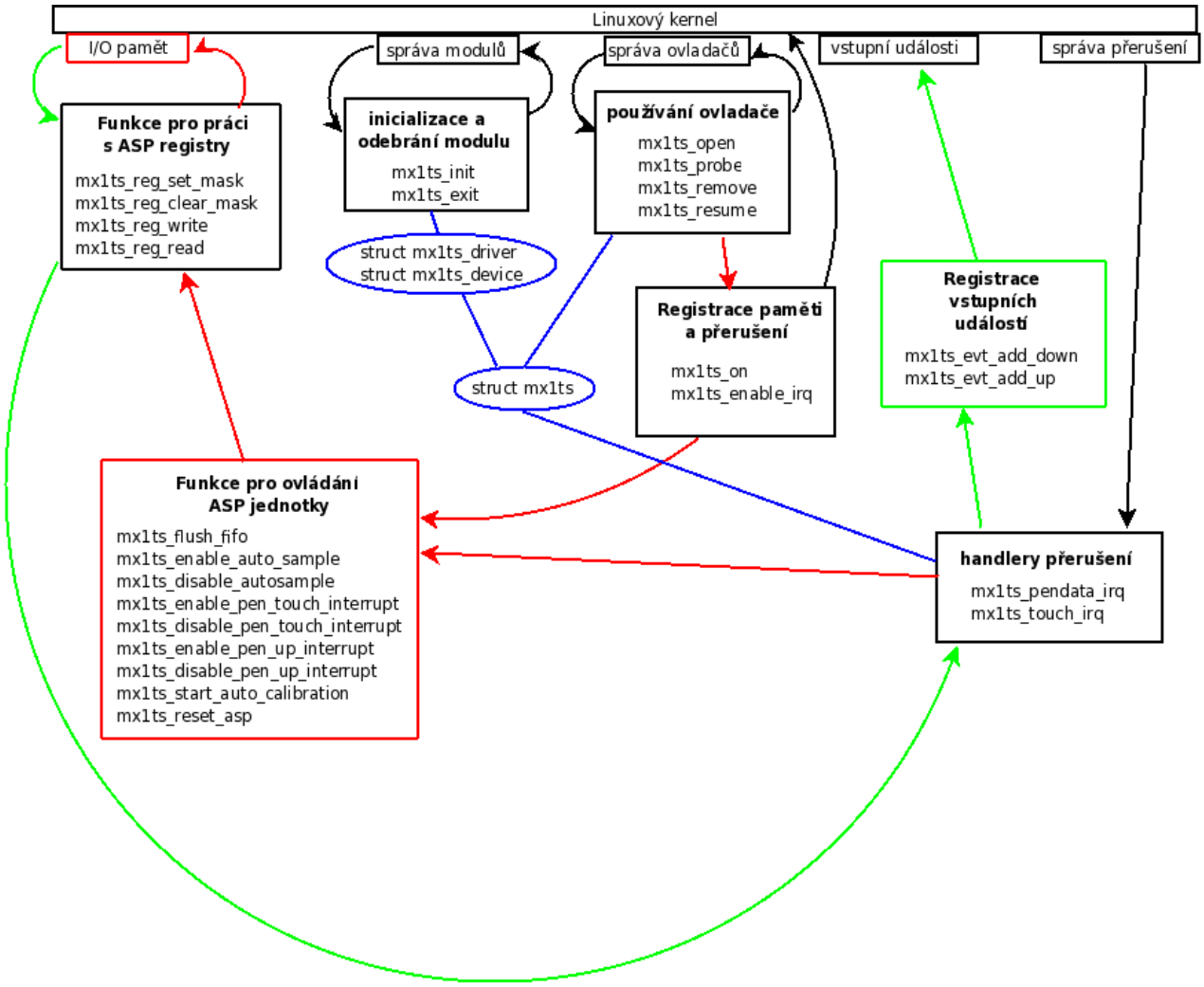
Pokud jsme některé výpisy na konzoli přehlédli, není problém si příkazem **'dmesg'** vypsat celý systémový log se všemi loglevely.

# 7 Ovladač dotykového displeje

## *7.1 Schéma ovladače dotykového displeje*

Dotykový ovladač spolupracuje s jádrem Linuxu. Funkci ovladače jsem se pokusil znázornit na jednoduchém diagramu.

Pro řídící příkazy jsem použil červenou barvu, toky datových informací o dotyku zeleně a vnitřní datové struktury ovladače jsou znázorněny modře.




*Obr 16: Schéma vzájemné interakce funkcí ovladače a jejich spolupráce s jádrem Linux*

## *7.2 Nastavení registrů ASP pro práci s displejem*

Všechna komunikace s hardwarem analogově digitálního převodníku, přes který je připojený displej, probíhá přes I/O paměť. V ovladači zajišťují komunikaci s I/O pamětí funkce :

```
static inline void mx1ts_reg_set_mask(struct mx1ts *mts , unsigned int reg, u32 mask)
static inline void mx1ts_reg_clear_mask(struct mx1ts *mts , unsigned int reg, u32 mask)
```

Funkce slouží k nastavení a smazání masky, jako vstupní parametry se funkcím předává ukazatel na strukturu mx1ts z důvodu uchovávání ukazatele na přemapovanou virtuální pamět, potom registr ASP, jehož hodnotu chceme měnit a ovlivňované bity jako masku

```
static inline void mx1ts_reg_write(struct mx1ts *mts, unsigned int reg, unsigned int val)
static inline unsigned int mx1ts_reg_read(struct mx1ts *mts, unsigned int reg)
```

Zapisují, nebo čtou 32 bitová čísla. Opět se předává ukazatel na strukturu, mx1ts, registr, který si přejeme číst, nebo zapisovat můžeme předávat také prostřednictvím definic v první části zdrojového kódu ovladače.

```
#define ASP_ACNTLCR       (0x10)     /* Control register */
#define ASP_PSMPLRG       (0x14)     /* Pen A/D sampe rate control */
#define ASP_CMPCNTL       (0x30)     /* Compare control register */
#define ASP_ICNTLR        (0x18)     /* Interrupt control register */
#define ASP_ISTATR        (0x1C)     /* Interrupt status register */
#define ASP_PADFIFO       (0x00)     /* Pen sample FIFO */
#define ASP_CLKDIV        (0x2C)     /* Clock divide register */
```

Pro jednotlivé registry nalezneme také definice všech důležitých bitů. Jednotlivé definice jsou stručně, ale výstižně komentované, tak není problém najít potřebnou funkci.

Na tyto funkce pak navazují pokročilejší funkce, které zajišťují komplexnější nastavení registrů vždy pro určitou příležitost. Všechny zmiňované funkce mají jako parametr ukazatel na strukturu mx1ts, ve které jsou uchovávána všechna potřebná data pro funkci ovladače.

```
static inline void mx1ts_flush_fifo(struct mx1ts *mts)
```

Slouží k vyprázdnění cyklického bufferu vzorků A/D převodníku. Funkce je nezbytná hlavně pro vyprázdnění zásobníku před spuštění auto-kalibrace, protože se může stát, že i po vypnutí vzorkování vlivem přepínání kontextu schedulerem systému dorazí ještě nějaký 'zpožděný' vzorek, který by pak vyvolal neplatnost dat autokalibrace.

```
static void mx1ts_enable_auto_sample(struct mx1ts *mts)
static void mx1ts_disable_auto_sample(struct mx1ts *mts)
```

Zapínají a vypínají auto-vzorkování. Automatické vzorkování je používáno v módu ASP_MODE_ONLY_Y, který v auto režimu vzorkuje kanály X, Y, U. Při zapínání auto_sample je aktivované také vzorkování 'Auto-Zero'. Hodnotou Auto-Zero se korigují měřené hodnoty. Vzorkování tedy probíhá v pořadí AZ, X, Y, U, AZ, X, Y, U ...

```
static void mx1ts_enable_pen_touch_interrupt(struct mx1ts *mts)
static void mx1ts_disable_pen_touch_interrupt(struct mx1ts *mts)
static void mx1ts_enable_pen_up_interrupt(struct mx1ts *mts)
static void mx1ts_disable_pen_up_interrupt(struct mx1ts *mts)
```

Ovládají nastavení příslušného přerušení. Přerušení 'pen up' se v ovladači neregistruje a jeho handler není potřeba, protože k události zvednutí pera může dojít jen při příchodu datových vzorků. Přítomnost přerušení se tedy kontroluje přítomností bitu ASP_PUIS v registru 'Interrupt/Error Status Register' (ASP_ISTATR) při příchodu nového datového vzorku v handleru přerušení mx1ts_pendata_irq.

```
static void mx1ts_start_auto_calibration(struct mx1ts *mts)
```

Funkce nastaví ASP registry do stavu auto-kalibrace, při které se odečtou referenční hodnoty, pomocí kterých se přepočítávají standardní vzorky dotyků. Auto-kalibrace odečte hodnoty call_auto_zero, call_range_x, call_range_y. Kalibrační hodnota auto-zero se použije pro korekci vzorků – její hodnota se odečte od hodnot call_range_x a call_range_y. Kalibrační hodnoty X a Y se používají k vydělení hodnot naměřených vzorků. Kalibrace slouží hlavně k eliminaci vlivů teplotních změn.

```
static void mx1ts_reset_asp(struct mx1ts *mts)
```

Provede reset ASP do stavu ve kterém se jednotka nachází po zapnutí napájení.

## *7.3 Registrace platform driveru*

Při konzultaci bylo rozhodnuto, že se ovladač bude psát jako platform_device. Důvodem je, že touchscreen je připojen přímo k procesoru a tudíž zde není žádná sběrnice, kterou by bylo možné využít jako základ pro ovladač. Pro úspěšnou funkci je nutné zaregistrovat jak platform_device, tak platform_driver, teprve po přítomnosti obou struktur v jádru je možné ovladač používat. Driver i device jsem registroval v inicializační funkci modulu. mx1ts_device a mx1ts_driver jsou struktrury, které odpovídají jejich definici v hlavičkových souborech jádra.

```
static int __init mx1ts_init(void)
{

	int error;
	error = platform_device_register(&mx1ts_device);
	if(error) {
		printk(KERN_ERR "mx1 touchscreen: device registration failed\n");
		return error;
	}


	error = platform_driver_register(&mx1ts_driver);
	if (error) {
		printk(KERN_ERR "mx1_touchscreen: failed to register touchscreen driver,
error: %d\n", error);
		platform_device_unregister(&mx1ts_device);
		return error;
	}


	return 0;

}
```

Komplementárními funkcemi k registracím jsou funkce s postfixem _unregistrer. Ty jsem použil při odebírání modulu.

```
static void __exit mx1ts_exit(void)
{
	platform_driver_unregister(&mx1ts_driver);
	platform_device_unregister(&mx1ts_device);

	return;
}
```

## 7.4 Funkce ovladače pro obsluhu jaderných volání

Struktura platform_driver vyžaduje zaregistrování několika obslužných funkcí ovladače. Jak je patrné ze struktury mx1ts_driver

```
static struct platform_driver mx1ts_driver = {
          .probe = mx1ts_probe,
          .remove   = mx1ts_remove,
          .resume   = mx1ts_resume,
          .driver = {
                    .name              = "mx1ts",
                    .owner             = THIS_MODULE,
                    },
};
```

je bylo třeba registrovat funkce probe, remove a resume.

```
static int mx1ts_probe(struct platform_device *dev)
```

Funkce mx1ts_probe je volána při zjištění hardwaru, to je v našem případě po spuštění funkce mx1ts_init, která zaregistruje platform_driver a platform_device. Po zjištění, že je přítomný jak device tak příslušný driver. Kernel zavolá funkci .probe danou ve struktuře platform_driver.

Ta v našem případě provede alokaci struktury mx1ts, ve které se uchovávají data potřebná pro další funkce ovladače. Také provede alokaci struktury idev – input_device. Dále do struktury mx1ts zkopíruje všechny informace dostupné z platform_device->resource (zde jsou uloženy hodnoty přerušení ASP jednotky a umístění a rozsah I/O paměti displeje). Zavolá funkci mx1ts_on, která se postará o vyžádání potřebné oblasti I/O paměti a provede její přemapování do použitelné virtuální paměti. Poté funkce mx1ts_on zavolá funkci mx1ts_enable_irqs, která zaregistruje handlery přerušení 'pen data' a 'pen touch'. Posledním úkolem mx1ts_on je reset asp jednotky, poté předá běh programu zpět funkci probe. Ta provede další nezbytné kroky pro správnou funkci ovladače. Nastaví proměné strukturu input_dev – vstupního zařízení a zajistí jeho registraci o které se podrobněji zmíním v další kapitole. Důvod, proč funkce volá pomocné funkce mx1ts_on a mx1ts_enable_irq, je hlavně jednodušší a přehlednější ošetření chyb. Funkce se tím rozdělí na několik částí přičemž v každé jsou samostatně ošetřeny chyby, které mohou vzniknout při neúspěšných registracích.

```
static int mx1ts_remove(struct platform_device *dev)
```

Funkce ošetřuje volání kernelu při odebrání ovladače. To znamená, že v opačném pořadí uvolňuje systémové zdroje, které byli obsazeny funkcí probe. Uvolní registrovaná přerušení, odregistruje vstupní zařízení, odmapuje virtuální pamět, uvolní I/O pamět a odalokuje paměť zabranou pro strukturu mx1ts.

```
static int mx1ts_resume(struct platform_device *dev)
```

Funkce resume by měla sloužit k obnovení funkce ovladače po režimu úspory energie. Vzhledem k tomu, že se ovladač vyvíjel pro touchscreen, který měl sloužit jako primární vstupní zařízení desky, tak uspání ovladače dotykového displeje není možné z důvodu jeho možnosti probuzení desky z úsporného stavu.

## 7.5 Registrace vstupního zařízení kernelu

Klíčovým prvkem funkce ovladače je registrace struktury input_dev, která je rozhraním kernelu pro hlášení vstupních událostí.

Registrace input_dev se provádí ve funkci mx1ts_probe. V první části ukázky vidíme nastavování struktury idev, dále pak volání funkce input_set_abs_params, pomocí které se registrují události, které se budou v rámci zařízení idev reportovat. Posledním důležitým voláním je input_register_device, které provede registraci celého input_dev.

```
        idev->dev.parent    = &dev->dev;
        idev->name          = "MX1 touchscreen interface";
        idev->phys          = "mx1ts/input0";
        idev->id.vendor     = (unsigned int) 345;
        idev->id.product    = (unsigned int) 354;
        idev->id.version    = 0x0100;
        idev->id.bustype    = BUS_HOST;
        idev->open          = mx1ts_open;
        idev->close         = mx1ts_close;
        idev->evbit[0]      = BIT_MASK(EV_KEY) | BIT_MASK(EV_ABS);
        idev->keybit[BIT_WORD(BTN_TOUCH)] = BIT_MASK(BTN_TOUCH);
        printk(KERN_DEBUG "mx1ts: setting idev struct \tOK\n");

        mts->x_res = 60000;
        mts->y_res = 60000;
        printk(KERN_DEBUG "mx1ts: x/y = %d/%d\n", mts->x_res, mts->y_res);

        input_set_abs_params(idev, ABS_X, 0, mts->x_res, 0, 0);
        input_set_abs_params(idev, ABS_Y, 0, mts->y_res, 0, 0);
        input_set_abs_params(idev, ABS_PRESSURE, 0, 1, 0, 0);

        error = input_register_device(idev);
        if (error) {
                printk(KERN_ERR "mx1 touchscreen: error while register input device\n");
                goto err_free_devs;
                }

        printk(KERN_DEBUG "mx1ts: input device registered \tOK\n");
```

Po registraci je možné začít dotykové události systému hlásit. To se provádí pomocí funkce input_report_abs a input_report_key. Praktickou ukázku vidíme na funkcích mx1ts_evt_add_touch a mx1ts_evt_add_up, které zajištují report událostí dotyku a funkce mx1ts_evt_add_up uvolnění dotyku.

```
static void mx1ts_evt_add_touch(struct input_dev *idev, u8 p , u16 x, u16 y)
{
        struct mx1ts *mts = dev_get_drvdata(idev->dev.parent);
        mts->x_akt = x;
        mts->y_akt = y;

        input_report_key(idev, BTN_TOUCH, 1);
        input_report_abs(idev, ABS_X, x);
        input_report_abs(idev, ABS_Y, y);
        input_report_abs(idev, ABS_PRESSURE, 1);
        input_sync(idev);
}

static void mx1ts_evt_add_up(struct input_dev *idev)
{
        struct mx1ts *mts = dev_get_drvdata(idev->dev.parent);

        input_report_abs(idev, ABS_X, mts->x_akt);
        input_report_abs(idev, ABS_Y, mts->y_akt);
        input_report_abs(idev, ABS_PRESSURE, 0);
        input_sync(idev);
}
```

## 7.6 Zpracování příchozích přerušení

Po otevření zařízení nějakým programem je v ovladači zavolána funkce mx1ts_open, která pouze vyprázdní fifo buffer a zapne generaci 'pen touch' přerušení v ASP registrech. Opakem otevření je funkce mx1ts_close, která vypne generaci přerušení a provede reset ASP jednotky.

Samotná funkce čtení vzorků z dotykového displeje je zajišťována při příchodu příslušného přerušení. Jsou zaregistrovány dva handlery přerušení.

```
static irqreturn_t mx1ts_touch_irq(int irq, void *dev_id)
```

Je registrován pro obsluhu události dotyku. Při volání funkce vypne další generaci 'pen touch' přerušení, potvrdí vyřízení přerušení nastavením bitu ASP_PEN v registru 'Interrupt/Error Status Register' a aktivuje auto-kalibraci obvodů analogově-digitálního převodníku. Auto-kalibrace zajistí načtení kalibračních vzorků, čímž dojde ke generaci přerušení 'pen data'.

```
static irqreturn_t mx1ts_pendata_irq(int irq, void *dev_id)
```

Obsluhuje událost při příchodu nových dat do cyklického datového bufferu ASP jednotky. Po svém zavolání funkce zkontroluje hodnotu bitu ASP_PUIS v registru stavu přerušení. Nenulový bit ASP_PUIS indikuje příchozí 'pen up' přerušení. V tom případě dojde k zapsání informace o zpracování přerušení, vypnutí auto-vzorkování desky, zapnutí generace 'pen touch' přerušení a nahlášení události zvednutí pera prostřednictvím funkce mx1ts_evt_add_up.

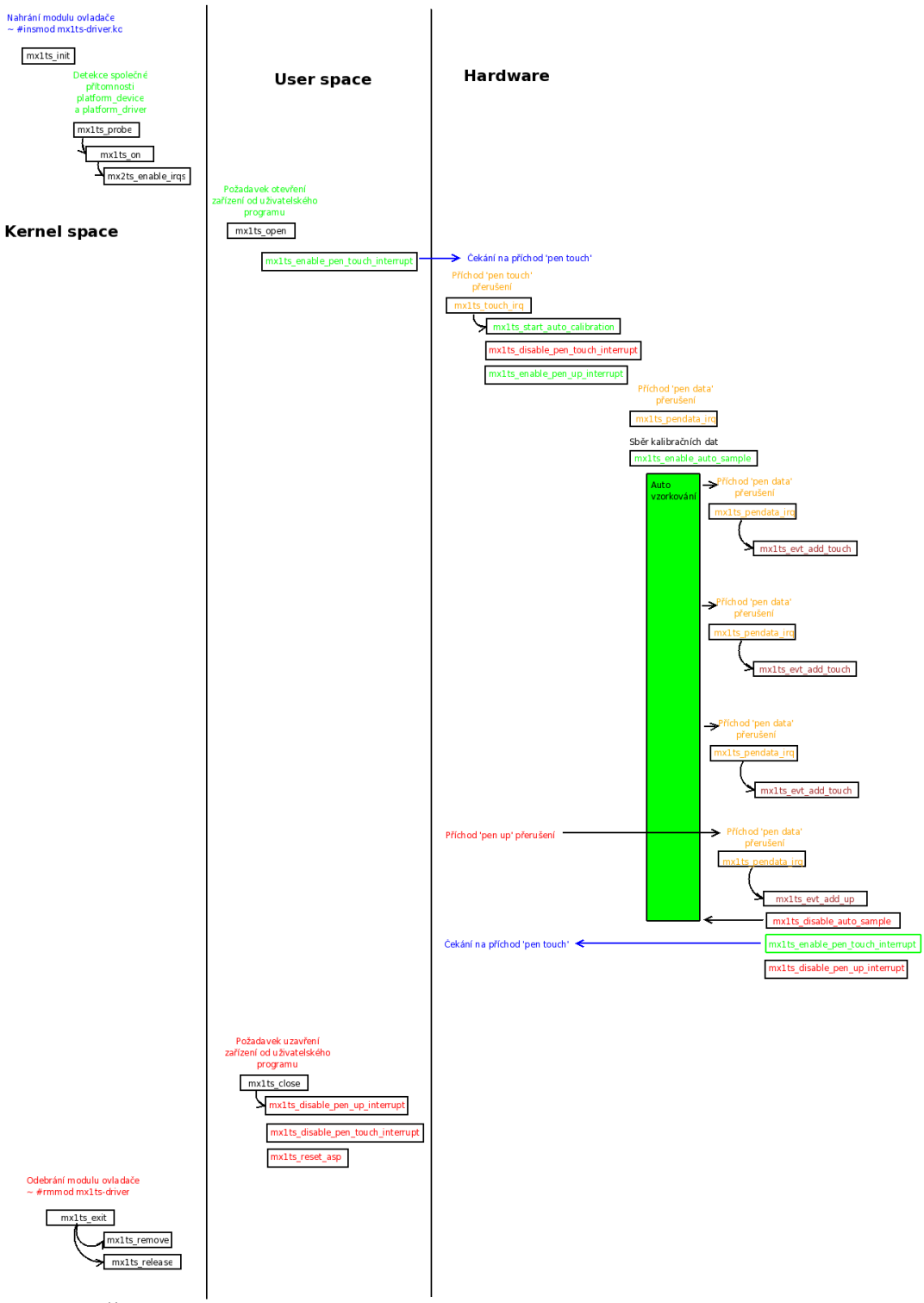
Pokud není aktivní příchozí 'pen up' přerušení, tak funkce podle stavu vnitřní struktury ovladače mx1ts, zkontroluje, zda není zapnutá auto-kalibrace. Když autokalibrace probíhá, tak dojde k odečtu kalibračních a jejich uložení do struktury mx1ts. Poté je aktivována auto vzorkovací funkce ASP jednotky.

Jestliže nebyla splněna žádná z předchozích podmínek, pak se jedná o vzorky dat probíhajícího dotyku, proto se provede přečtení navzorkovaných hodnot pro jednotlivé osy displeje, korekce pomocí dat uložených z poslední kalibrace a jejich nahlášení pomocí funkce mx1ts_evt_add_touch.

Handlery přerušení zaručují celou funkčnost ovladače displeje tím, že dochází k aktivaci generace 'pen touch' přerušení při nepřítomnosti dotyku, nebo naopak aktivaci 'pen up' přerušení pokud dotyk probíhá. Pomocí těchto dvou přerušení je událost dotyku jednoznačně identifikovatelná.

## 7.7 Časové schéma funkce ovladače

Jelikož jsem si vědom komplikovanější struktury ovladače dotykového displeje, tak kapitolu ještě doplňuji velmi zjednodušeným časovým diagramem předpokládaného běžného provozu ovladače mx1ts_driver.




*Obr 17: Časové schéma volání funkcí v ovladači mx1ts_driver*

# 8 Grafické prostředí

## *8.1 Důvod grafického prostředí*

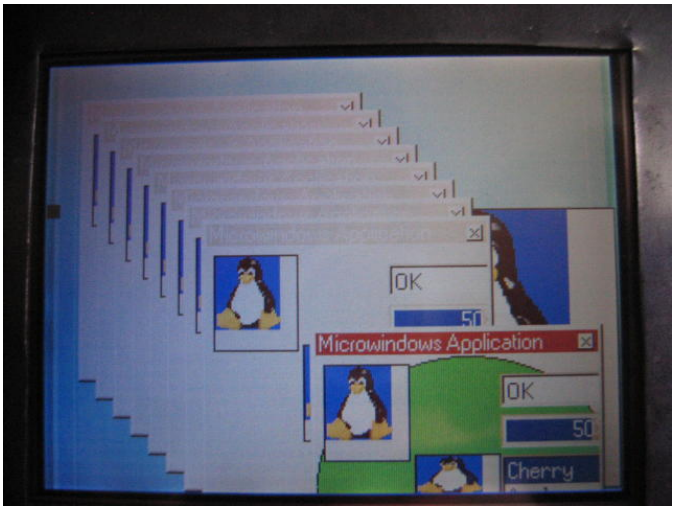
Pro použití existujícího grafického uživatelského prostředí je z hlediska budoucí podpory tvůrců aplikací mnohem vhodnější než implementace vlastního grafického prostředí, které v průběhu času zastarává a chybí k němu funkční e-mailová konference s podporou nových vývojářů.

## *8.2 Nano-X (microwindows) [27]*

Nano-X je grafické uživatelské prostředí vyvíjené jako Open source. Projekt s původním názvem microwindows slouží hlavně pro malá zařízení a platformy. Při vývoji aplikace je možné používat dvě různá API a to Win32 API a API podobné Xlib. Aktuální zdrojové kódy projektu je možné získat na `cvs.microwindows.org`. Zdrojové kódy obsahují základní elementy potřebné pro funkci grafického prostředí od enginu přes potřebné fonty a ovladače. Také je zde několik demo aplikací pro ověření správné funkčnosti Nano-X a jako inspirace pro vývojáře aplikací založených na prostředí Nano-X.

Pro nastavení kompilace systému Nano-X slouží konfigurační soubor config. Zde je možné nastavit různé parametry od architektury až po vstupní a výstupní zařízení grafického prostředí. Možností je velmi mnoho a konfigurační soubor je vcelku přehledný, proto jej zájemcům o podrobnější informace o možnostech konfigurace doporučuji k nahlédnutí.

Po kompilaci se vytvoří několik demo aplikací sloužících k ověření funkčnosti prostředí.



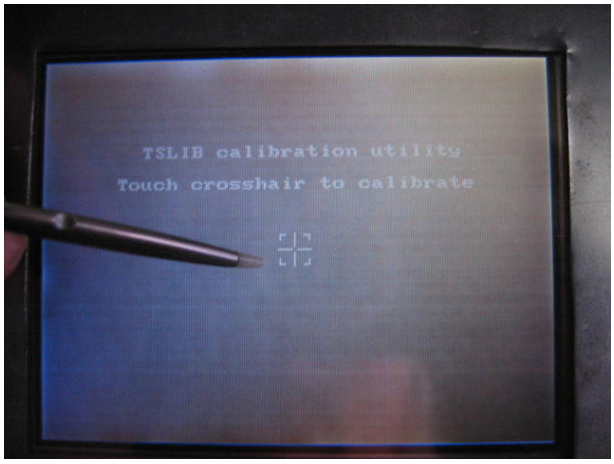

*Obr 18 : Demo aplikace Nano-X ukazuje možnosti zobrazení oken*

## *8.3 Knihovna Tslib [28]*

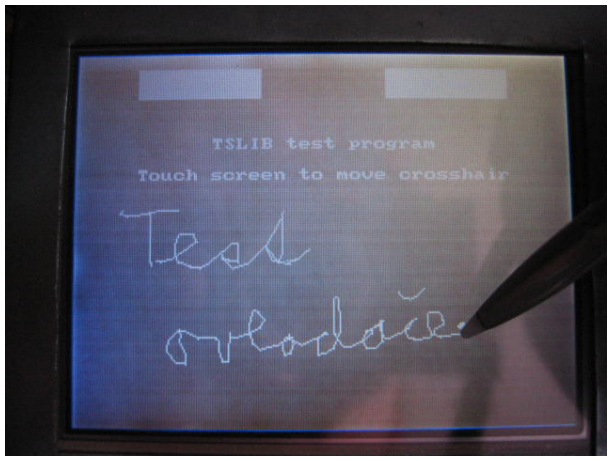
Tslib je knihovna, jenž vytváří abstrakční mezivrstvu mezi ovladačem dotykového panelu a uživatelskou aplikací. Byla vytvořena Russelem Kingem. Obsahuje také základní filtry pro zpracování dotykových událostí, například eliminace šumu způsobeného chvěním nebo odchylkami způsobenými nestálosti nejnižšího bitu A/D převodníku. Knihovna také obsahuje nástroj pro kalibraci dotykového displeje a filtr pro výpočet kalibrační transformace při běžném používání touchscreenu.

Knihovna Tslib nevyžaduje žádné zvláštní nastavení. Vytvořený ovladač používá hlášení událostí jako 'input events', proto lze ke čtení výstupu dotykových hodnot použít modul knihovny 'raw input' a není třeba nijak upravovat kód knihovny. Je pouze nutné ji zkompilovat. Před samotnou kompilací je třeba spustit 'configure' s parametrem --host=arm-linux, který vytvoří soubory Makefile podle možností kompilátoru, který je určen parametrem host. Nejaktuálnější verze zdrojových kódů knihovny Tslib definuje v souboru config.h použití funkce rpl_malloc místo standardní funkce malloc, což je pro náš případ chyba, proto je nutné řádek '#define malloc rpl_malloc' v souboru config.h zakomentovat. Poté je se zkompilované knihovny a testovací soubory nakopírují do příslušných adresářů. Knihovny do adresáře /lib v exportovaném rootfile systému, binární soubory ts_calibrate, ts_harvest, ts_print, ts_print_raw a ts_test do adresáře /bin, soubor ts.conf do /etc a soubor tslib.h do /include.

Konfigurace se provádí prostřednictvím souboru ts.conf. Zde je třeba odkomentovat řádek 'module_raw input ', čímž nastavíme jako vstupní zařízení /dev/input/event. Dále je v souboru možné zakomentovat filtry, které nebudeme používat. Lineární filtr je výhodné nechat, protože se používá kalibrace dotykového displeje pomocí knihovny Tslib. Pro kalibraci slouží binární soubor ts_calibrate, pro otestování nastavení existuje ts_test. Spustitelné soubory ts_print a ts_print_raw slouží pro otevření zařízení a výpis hlášených dotykových událostí.




*Obr 19 : Ukázka kalibrace dotykového displeje pomocí Tslib*




*Obr 20 : Ukázka funkce ovladače pomocí testovací utility knihovny Tslib*

# 9 Závěr

Při práci jsem byl velmi překvapen rozsahem nutných znalostí pro vytvoření jednoduchého Linuxového ovladače. Proto také rozsáhlou část práce zabírá kompilace vědomostí z existujících zdrojů. Práce se snaží být pro čtenáře i návodem, který jej zasvětí do základů tvorby Linuxového ovladače bez nutnosti procházení množství různých cizojazyčných webových materiálů. Jelikož však práce omezená svým rozsahem může obsáhnout jen základy této problematiky, tak zároveň čtenáři nechává dostatek prostoru pro pokročilejší studium anglických zdrojů. Optimistickým zjištěním pro mne je to, že po absolvované cestě studia problematiky jsem získal mnoho dovedností, díky kterým pro mne není orientace ve zdrojových kódech Linuxu problémem. Jak se práce blížila ke konci, tak jsem si také více uvědomoval důvod malého množství literatury zaměřené na vývoj Linuxu a jeho součástí. Tímto důvodem, je snaha psát zdrojové kódy natolik přehledně, aby návody nebyly potřebné a také velmi rychlý vývoj jaderných funkcí, který není schopen žádný autor komplexně sledovat. Nejlepším postupem pro tvorbu ovladače je poučit se z již napsaných podobných ovladačů, jaké struktury je nejvýhodnější použít, definici těchto struktur si najít přímo ve zdrojových kódech Linuxu a pak následuje nejobtížnější fáze práce – podle specifikací výrobce hardwaru napsat a odladit ovladač. Cesta je velmi jednoduchá, jen je nutné úvodní ponoření do velmi komplexní problematiky. Doufám, že tato cesta bude díky této práci pro čtenáře jednodušší.

Ovladač dotykového displeje byl úspěšně otestován s knihovnou Tslib Russella Kinga. Ta zajišťuje základní interpretaci dotykových událostí hlášených ovladačem. S grafickým prostředím Nano-X se mi naneštěstí do uzavření práce nepodařilo dotykový displej úspěšně otestovat. To je dáno zčásti nehodou, která mne postihla v průběhu práce – poškození pevného disku na notebooku, na kterém jsem pracoval. Bohužel vzhledem ke komplexnosti nutného nastavení systému a cross-compilace byly mé zálohy nedostatečné, opětovnou konfigurací jsem pak ztratil příliš velké množství času.

Dalším problémem, který mne v práci potkal bylo neosazení odporu na vstupu analogově digitálního převodníku. Tento malý chybějící odpor mne při testování ovladače dlouhou dobu trápil, jelikož jeho nepřítomnost vedla ke generaci falešných přerušení, které zabraňovaly použitelnosti dotykových událostí. Řešením tohoto problému jsem strávil mnoho dní i nocí, proto bych chtěl na tomto místě poděkovat odborníkům firmy PiKRON, že se jim podařilo záhadu odhalit a tím vyřešit můj problém.

Zčásti sehrála svou roli také složitost systémů Nano-X a TSLIB, ve kterých je nutná alespoň částečná orientace pro zajištění jejich bezchybné spolupráce.

Nicméně i přes nedokončení vývoje programu v grafickém prostředím si myslím, že práce je velmi obsáhlá a komplexní. Linuxový ovladač dotykového displeje otestovaný a odladěný. Také znalosti a zkušenosti, které jsem při práci získal, jsou pro mne nesmírně hodnotné a pevně věřím, že řešení podobných úkolů bude pro mne v budoucnu mnohem jednodušší.

Za pomoc při zpracování bakalářské práce děkuji Ing. Pavlu Píšovi.

# 10 Seznam použité literatury


1) *Triggs, A.: TouchScreen FAQ* [online]. 1997, poslední revize 2005-09-25 [cit. 2008-07-04]. Dostupné z WWW: <http://www.repairfaq.org/filipg/LINK/F_touchscreen.html>

2) *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikimedia Foundation, 2001- , poslední revize 2008-07-03 [cit. 2008-07-04]. Wikipedia: Touchscreen. Dostupné z WWW: < http://en.wikipedia.org/wiki/Touchscreen>

3) *Hewlett-Packard Development Company, L.P.: HP-150 touchscreen personal computer with HP 9121 dual drives, 1983* [online]. 2008, [cit. 2008-07-04]. Dostupné z WWW: <http://www.hewlettpackard.com/hpinfo/abouthp/histnfacts/museum/personalsystems/0031/index.html>

4) PiKRON: PiMX1 [online]. 2007, [cit. 2007-12-29]. Dostupné z WWW: <http://www.pikron.com/pages/products/cpu_boards/pimx1.html>

5) Freescale Semiconductor: *Freescale Polovodiče* [online]. 2004 , [cit. 2007-12-29]. Dostupné z WWW:

   <http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=06784331630520>

6) *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikimedia Foundation, 2001- , poslední revize 2008-07-03 [cit. 2008-01-16]. Wikipedia: Freescale. Dostupný z WWW: <http://en.wikipedia.org/wiki/Freescale>

7) ARM: *ARM920T* [online]. [cit. 2008-01-30]. Dostupné z WWW: <http://www.arm.com/products/CPUs/ARM920T.html>

8) Freescale Semiconductor: *i.MX1: MC9328MX1 Applications Processor* [online]. 2004, poslední revize [cit. 2008-01-31]. Dostupné z WWW: <http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX1>

9) Freescale Semiconductor, *MC9328MX1 i.MX Integrated Portable System Processor, Reference Manual* [online], Rev. 6.1 , poslední revize 06/2007, [cit. 2008-03-12]. Dostupné z WWW: <http://www.freescale.com/files/32bit/doc/ref_manual/MC9328MX1RM.pdf>

10) Přibyl, A.: G*NU/Linux* [online]. 2007, [cit. 2008-03-20]. Dostupné z WWW: <http://www.linux.cz>

11) Johnsons M. K.: *Linux Information Sheet* [online]. 1998, [cit. 2008-03-20]. Dostupné z WWW: <http://linuxdoc.linux.cz/HOWTO/INFO-SHEET.html>

12) ARM: *About ARM* [online]. [cit 2008-05-10]. Dostupné z WWW:< http://www.arm.com>

13) Přibyl A.: *ARM – Linux není jen x86* [online]. 2005-09-01 [cit 2008-05-10] Dostupné z WWW: <http://www.root.cz/clanky/arm-linux-neni-jen-x86/>

14) King Russell: *The ARM Linux project* [online]. Poslední revize 2008 [cit 2008-05-10]. Dostupné z WWW: <http://www.arm.linux.org.uk/>

15) Free Electrons: *Embedded Linux Training* [online]. Poslední revize 2008-04-10 [cit. 2008-05-15]. Dostupné z WWW: <http://free-electrons.com/training>

16) manuálové stránky jmenovaných příkazů (man <příkaz>) ve verzi jádra 2.6.24

17) dokumentace linuxu dostupná ve zdrojových kódech kernelu v adresáři Documentation, verze jádra 2.6.24

18) soubor README, návod ke kompilaci kernelu obsažený ve zdrojových kódech jádra 2.6.24

19) DENX: *U-Boot* [online]. 2002 – 2008 poslední revize 2008-05-13 [cit. 2008-05-20]. Dostupné z WWW: <http://www.denx.de/wiki/U-Boot/WebHome>

20) GNU: *GCC, the GNU Compiler Collection* [online]. poslední revize 2008-06-06 [cit. 2008-05-20]. Dostupné z WWW: <http://gcc.gnu.org>

21) *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikimedia Foundation, 2001- , poslední revize 2008-05-07 [cit. 2008-05-21]. Wikipedia: GNU Compiler Collection. Dostupný z WWW: <http://en.wikipedia.org/wiki/GNU_Compiler_Collection>

22) Smith Christopher M.: Linux NFS Overview, FAQ and HOWTO Documents [online]. [cit. 2008-05-21]. Dostupný z WWW: <http://nfs.sourceforge.net>

23) Wikipedia: the free encyclopedia [online]. St. Petersburg (Florida): Wikimedia Foundation, 2001- , poslední revize 2008-05-14 [cit. 2008-05-21]. Wikipedia: Trivial_File_Transfer_Protocol.
Dostupný z WWW: <http://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol>

24) M. van Smoorenburg, J. Lahtinen, - *Manuálové stránky programu minicom.* (man minicom) [cit. 2008-05-23], verze jádra 2.6.24

25) Baudis P.: Git – *Fast Version Control System* [online]. [cit. 2008-05-23] Dostupné z WWW: <http://git.or.cz/>

26) J. Corbet, A. Rubini, G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly Media, Inc.; 3 edition (February 7, 2005), ISBN-10: 0596005903 ISBN-13: 978-0596005900

27) Gregory Haerr: *The Nano-X Window System* [online]. Poslední revize 2005-01-30 [cit. 2008-05-25]. Dostupné z WWW: <http://www.microwindows.org>

28) Tslib [online]. [cit. 2008-05-26] Dostupné z WWW: <http://tslib.berlios.de/>

# 11 Seznam obrázků použitých v textu

Obr 1. : Příklad dotykového displeje....7
Obr 2. : Počítač HP-150 vybavený dotykovým displejem....7
Obr 3. : Znázornění vrstev odporového dotykového displeje....9
Obr 4 : Praktická aplikace desky PiMX1 - injekční pumpa....12
Obr 5.: Deska PiMX1, pohled ze strany procesoru....12
Obr 6.: Deska PiMX1, pohled ze strany součástek....13
Obr 7.: Schématické rozmístění součástek na desce....14
Obr 8.: Schématické rozmístění součástek ze strany procesoru....14
Obr 9.: Schématický diagram desky....15
Obr 10 : Blokový diagram architektury procesoru ARM920T....16
Obr 11 : Blokový diagram ASP systému....17
Obr 12 : Zjednodušený diagram přepínacích obvodů ASP jednotky....18
Obr 13 : Tabulka uvádějící možnosti nastavení automatického vzorkování ASP jednotky....19
Obr 14 : Registry ASP jednotky....21
Obr 15 : ASP Kontrolní registr....21
Obr 16: Schéma vzájemné interakce funkcí ovladače a jejich spolupráce s jádrem Linux....40
Obr 17: Časové schéma volání funkcí v ovladači mx1ts_driver....46
Obr 18 : Demo aplikace Nano-X ukazuje možnosti zobrazení oken....47
Obr 19 : Ukázka kalibrace dotykového displeje pomocí Tslib....48
Obr 20 : Ukázka funkce ovladače pomocí testovací utility knihovny Tslib....48

# 12 Seznam příloh na CD

1) mx1ts_driver.c – zdrojové kódy modulu ovladače displeje
2) Makefile pro kompilaci ovladače
3) linux-2.6.24.tar.gz – zabalený adresář se zdrojovými kódy použitého jádra s aplikovanými patchy
4) microwin.tar.gz – zabalený adresář se zdrojovými kódy použité verze Nano-X
5) tslib.tar.gz – zdrojové kódy a zkompilovaná verze knihovny Tslib