# ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
# FAKULTA ELEKTROTECHNICKÁ

BAKALÁŘSKÁ PRÁCE

# MOBILNÍ PAMĚŤOVÝ OSCILOSKOP REALIZOVANÝ POMOCÍ FPGA SPARTAN 3

Studijní obor: Kybernetika a měření
Vedoucí práce: Ing. Jiří Kadlec, CSc.

ČERVEN 2006

TOMÁŠ ŠKARDA

**Poděkování:** Na tomto místě bych chtěl poděkovat především vedoucímu mé bakalářské práce Ing. Jiřímu Kadlecovi CSc. za ochotu vedení, cenné rady, připomínky a trpělivost.

Dále bych chtěl poděkovat všem přátelům, kteří mi pomohli při testování aplikací pro realizaci mobilního paměťového osciloskopu a všem blízkým, kteří mě po celou dobu studia na FEL ČVUT podporovali.

**Prohlášení:** Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

V Praze dne…… …….

podpis

# Abstrakt

Hlavní myšlenkou této bakalářské práce je vzít minimální hardware řešení osciloskopu a rozšířit ho o další hardware funkční bloky. K tomuto využijeme desku RC10 Pilot s logickým obvodem Spartan 3 XC3S1500L-4. Nejdůležitější vlastností tohoto osciloskopu, je možnost rozšíření o uživatelem požadovaný funkční blok a možnost provozu na levnější a dostupnější desce s logickým obvodem Spartan 3 XC3S400 .



# Abstract

The main idea of this Bachelor´s Project is to take a minimum hardware analys oscilloscope and extend it by additional hardware functional blocks. An RC10 Pilot board with a logic circuit Spartan 3 XC3S1500L-4 can be used for this purpose. The most important feature of this oscilloscope is the possibility to be extended by the user required operational block and the potential to operate the oscilloscope on a cheaper and more available board with the Spartan 3 XC3S400 logic circuit.

**Obsah:**


**1. Úvod ..... 1**
**2. Analýza ..... 2**
2.1. Rozdělení osciloskopů ..... 2
2.1.1. Vybrané druhy osciloskopů, jejich vlastnosti a cena ..... 3
2.2. Seznámení s deskou RC10 Pilot ..... 4
2.2.1. Funkční bloky RC10 Pilot ..... 5
2.3. Číslicově programovatelné součástky ..... 8
2.3.1. Klasické PLD ..... 8
2.3.2. Komplexní PLD ..... 9
2.3.3. Obvody FPGA ..... 10
2.4. FPGA Xilinx ..... 12
2.4.1. FPGA Spartan - 3/3L typu XC3S1500L ..... 13
2.4.2. Celkový pohled na architekturu FPGA Spartan 3/3L ..... 14
**3. Vývojové prostředí ..... 15**
3.1. Software použitý při tvorbě aplikace ..... 15
3.2. Hardware použitý při tvorbě aplikace ..... 16
**4. Popis hardware implementace a procedur ..... 17**
4.1. Minimální hardware řešení od firmy Celoxica ..... 17
4.1.1. ADC převodník použitý na RC10 Pilot ..... 17
4.1.2. Ukládání vzorků z ADC převodníků do dualport RAM pamětí ..... 18
4.2. Rozšířené hardware řešení ..... 20
4.2.1. Návrh obvodového řešení pro vykreslení uživatelského menu na obrazovku 21
4.2.2. Princip dualport RAM pamětí pro menu a fonty ..... 22
4.2.3. Princip pohybu v uživatelském menu ..... 26
4.2.4. Pomocné měřicí kurzory v ose X a Y ..... 29
4.2.5. Vykreslení hexadecimální hodnoty kurzoru v digitální podobě ..... 32
4.2.6. Zobrazení a přepočet digitální hodnoty v milivoltech ..... 33
4.2.7. Zobrazení a přepočet digitální hodnoty v dílkách ..... 35
4.2.8. Vykreslení pomocné měřicí mřížky ..... 35
4.2.9. Přenos dat do PC s vykreslením průběhu v programu matlab ..... 36
**5. Ověření funkčnosti ..... 39**
**6. Výstupní parametry osciloskopu ..... 40**
**7. Závěr ..... 41**
**8. Literatura ..... 41**
**9. Obsah přiloženého CD ..... 42**
**10. Přílohy ..... 42**

**Seznam obrázků:**

*Obrázek 1 : Digitální osciloskop* ..... 2
*Obrázek 2 : Analogový osciloskop* ..... 3
*Obrázek 3: RC10 Pilot* ..... 4
*Obrázek 4: Schéma funkčních bloků desky RC10 Pilot* ..... 7
*Obrázek 5: Vnitřní struktura klasického PLD* ..... 8
*Obrázek 6: Součinové hradlo* ..... 9
*Obrázek 7: Struktura obvodu CPLD* ..... 10
*Obrázek 8: Struktura obvodu FPGA* ..... 11
*Obrázek 9: Struktura logického bloku LB* ..... 11
*Obrázek 10: Architektura FPGA Spartan 3/3L* ..... 14
*Obrázek 11: Programové prostředí DK4.0* ..... 15
*Obrázek 12: Programové Prostředí ISE 7.1* ..... 15
*Obrázek 13: Programové Prostředí FTU* ..... 16
*Obrázek 14: Využívaný hardware* ..... 16
*Obrázek 15: Demo programu osciloskop* ..... 17
*Obrázek 16: Vnitřní zapojení ADC převodníku* ..... 18
*Obrázek 17: Vzorkování, ukládání do paměti a vykreslení na monitor* ..... 19
*Obrázek 18: Rozšířené demo programu osciloskop* ..... 21
*Obrázek 19: Vnitřní struktura dualport RAM paměti pro fonty* ..... 23
*Obrázek 20: Vnitřní struktura dualport RAM paměti pro menu* ..... 23
*Obrázek 21: Vykreslování pamětí pro menu a fonty na obrazovku* ..... 24
*Obrázek 22: Pohyb v uživatelském menu* ..... 27
*Obrázek 23: Proces pohybu v menu* ..... 27
*Obrázek 24: Vykreslené kurzory v ose X a Y* ..... 31
*Obrázek 25: Pomocná měřicí mřížka* ..... 36
*Obrázek 26: Rozšíření hardware o další dvě RAM paměti pro ukládání dat z ADC a vykreslení v programu Matlab* ..... 37

**Rejstřík:**


ADC0, 18
ADC1, 18
ADC10065, 17
Analog a zvuk, 6
Analogové osciloskopy, 2, 3
architekturu FPGA, 13
Bloky LB, 11
Digitální osciloskop, 1
Digitální osciloskopy, 3
Display, 6
DK Design Suite, 4, 5
DK41, 1
DLL, 11
dualportové RAM paměti, 23
fonty, 22
FPGA, 1, 5, 10, 11
FPGA Spartan - 3/3L typu XC3S1500L, 13
FPGA Xilinx, 11
Handel-C, 1
hexadecimální hodnoty kurzoru, 32
logické bloky, 13
logický obvod, 1
Matlab, 36
menu, 22
měřicí kurzory, 29
měřicí mřížka, 35
Mikrokontroler, 6
minimální hardware řešení, 16
osciloskop, 1
Osciloskop, 1
osciloskopu, 1
osciloskopy, 1
Osciloskopy, 1
PAL, 9
Paměť, 5
paměť RAM, 12
PDK41, 1
PixelStreams, 5
PLA, 9
Platform Developer's Kit, 1
PLD, 7
PLL, 11
proceduru „CaptureData, 18
proceduru „DisplayData“, 21
Přenosné bateriové osciloskopy, 3
přepočet na sekundy, 35
přepočet volt na pixel, 33
RC10 pilot, 1
RC10 Pilot, 1, 4
RC10Pilot, 7
Rozhraní, 5
rozhraní USB, 36
Rozšířené hardware řešení, 19
Spartan 3, 1, 40
VGA monitoru, 1
Video, 6
Xilinx ISE 7.1, 1

# 1. Úvod

Cílem této práce je naprogramovat mobilní paměťový osciloskop. K tomu využiji programovatelný logický obvod Spartan 3 XC3S1500L-4, který je součástí desky RC10 Pilot od anglické společnosti Celoxica . Výhodou tohoto osciloskopu je velmi malý rozměr, lehkost, snadná přenositelnost pro využívání při měření ve velmi ztížených podmínkách, zobrazování výstupních dat na připojeném VGA monitoru, popřípadě mobilním barevném grafickém a alfanumerickém LCD displeji 98 x 68 bodů. Oproti osciloskopům požívaných například pro laboratorní účely.

Hlavní výhodou je i jeho snadné ovládání pro uživatele a možnost rozšíření o jednotlivé snadno programovatelné bloky v jazyce Handel-C a programovém prostředí DK41 a PDK41.

V této práci jsem k programování využil demoverzi programu adc.hcc, která je součástí uživatelského balíčku Platform Developer's Kit ( PDK ) od firmy Celoxica. Úkolem mé práce bylo vzít tento program a rozšířit ho o další funkční bloky programovatelného osciloskopu pro logický obvod Spartan 3 XC3S1500L-4 na desce RC10 Pilot.

Osciloskop je zařízení, které umožňuje zobrazení periodických signálů. Tento přístroj je dnes nezbytnou součástí každého vývojového pracoviště, zabývající se výrobou elektronických zařízení. V jednoduchosti bude uživateli umožňovat:

- ✓ možnost měnit pozici signálu (Offset)
- ✓ možnost vypnutí a zapnutí signálu (Off)
- ✓ možnost zastavení a následného zapnutí signálů(ST/RU)
- ✓ měnit časovou základnu (Time Base)
- ✓ čtyři měřicí x,y kursory (Cursors), pro snadné a jednoduché měření signálů.
- ✓ ovládání těchto funkcí bude pomocí mini-joysticku, který je součástí RC10 pilot, s výběrem v uživatelském menu na připojeném monitoru.
- ✓ Přenos vstupních signálů přes rozhraní USB do programu Matlab

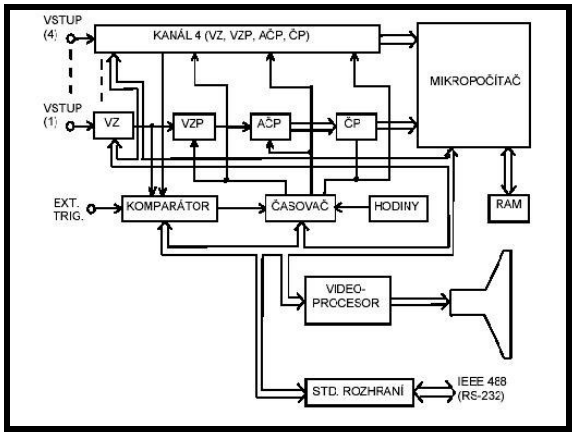
Bakalářská práce je členěna na jednotlivé kapitoly. První kapitola je úvod. Druhá kapitola se zabývá analýzou problému. Obsahuje přehled laboratorních osciloskopů a jejich vlastností, seznámení s deskou RC10 Pilot, přehled o číslicově programovatelných součástkách, jako je FPGA a architekturu čipu Spartan 3/3L. Třetí kapitola obsahuje vývojové prostředí a hardware požitý při práci. Součástí je i zmínka o programovém prostředí DK41, PDK41, Xilinx ISE 7.1. Čtvrtá kapitola se zabývá hardware implementací a popisem částí hardware v jazyce Handel-C. Pátá kapitola popisuje ověření funkčnosti osciloskopu . V šesté kapitole jsou uvedeny výstupní parametry po překladu programu. Sedmá kapitola je závěr. Osmá kapitola obsahuje seznam použité literatury a odkazy na materiály, které jsem využil při psaní této bakalářské práce.

# 2. Analýza

## 2.1. Rozdělení osciloskopů

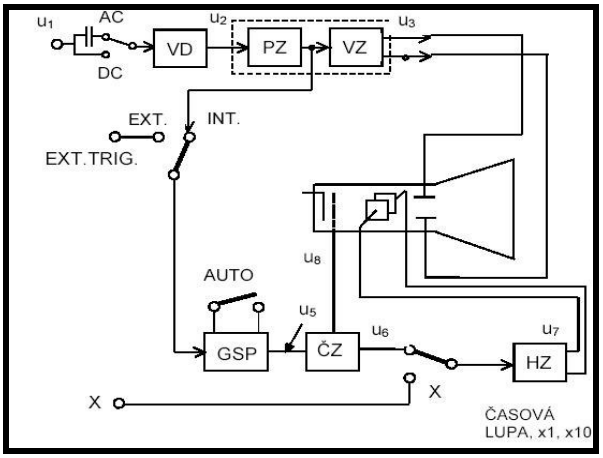
Osciloskopy se podle zpracování signálu před zobrazením mění na analogové a digitální.

Digitální osciloskop (obrázek č.1) je založen na odlišném principu než analogový osciloskop. Analogový signál je vzorkován pomocí A/D převodníku do vnitřních pamětí osciloskopu. Po zaplnění těchto pamětí měřeným signálem jsou tyto vzorky zpracovány na podobu potřebnou pro zobrazení. Upravený průběh je následně zobrazen na obrazovce. Výhodou tohoto zařízení je připojení k počítačové stanici a možnost přenosu naměřených hodnot. Digitální osciloskopy mohou provádět mnoho operací, které analogové ve svém principu neumožňují. Tímto narůstá cena digitálních osciloskopů.




***Obrázek 1 :*** *Digitální osciloskop*
Zdroj: HAASZ, V., SEDLÁČEK, M.: *Elektrická měření*

Analogové osciloskopy (obrázek č.2) pracují na principu vychylování paprsku v závislosti na dvou analogových signálech. První signál je v podstatě měřený signál a vychyluje paprsek ve vertikálním směru. Druhý signál je generován časovou základnou a vychyluje paprsek v horizontálním směru. Rozsah časové základny se mění změnou strmosti pily druhého signálu. Napěťové rozsahy se mění vstupním děličem.




***Obrázek 2 :*** *Analogový osciloskop*
Zdroj: HAASZ, V., SEDLÁČEK, M.: *Elektrická měření*

### 2.1.1. Vybrané druhy osciloskopů, jejich vlastnosti a cena

Digitální osciloskopy:

| Název | Počet kanálů | Citlivost | Kmitočtové pásmo | Cena |
|---|---|---|---|---|
| GDS 820S | dva | 2mV - 5V/díl | 150MHz | 46 000,- |
| GOS 6112 | dva | 2mV - 5V/díl | 100MHz | 38 000,- |
| GOS 658G | dva | 1mV - 5V/díl | 50MHz | 32 000,- |
| TDS 1002 | dva | 2mV - 5 V/díl | 60MHz | 48 000,- |

Analogové osciloskopy:

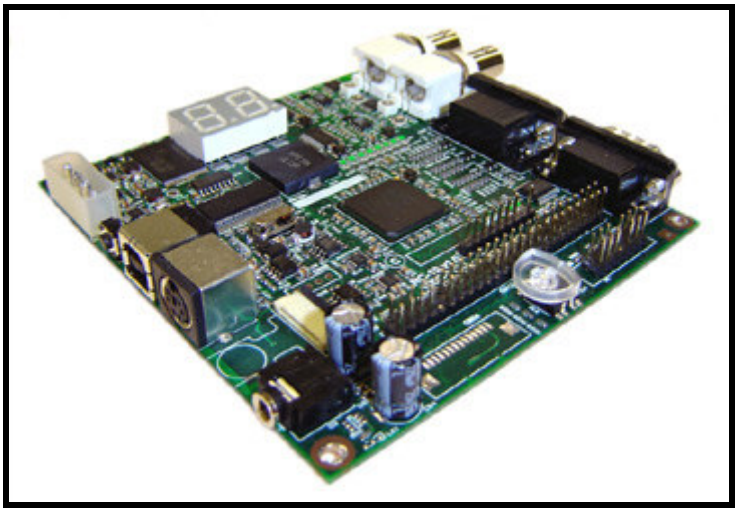
| Název | Počet kanálů | Citlivost | Kmitočtové pásmo | Cena |
|---|---|---|---|---|
| GOS 310 | jeden | 5mV | 10MHz | 11 000,- |
| GOS 622G | dva | 5mV | 20MHz | 22 000,- |
| GOS 653G | dva | 1mV | 50MHz | 33 000,- |
| OS 5060A LGP | dva | 1mV | 60MHz | 25 000,- |

Přenosné bateriové osciloskopy:

| Název | Počet kanálů | Vzorkování | Kmitočtové pásmo | Displej |
|---|---|---|---|---|
| OX 7042 - M | dva | 1 GS/s | 40MHz | ČB |
| OX 7104 - C | čtyři | 1 GS/s | 100MHz | BA 320x240 |
| Fluke 196C | dva | - | 100MHz | BA LCD |
| TPS 2000 | Dva - čtyři | až 2 GS/s | 100-200MHz | BA |

## 2.2. Seznámení s deskou RC10 Pilot

RC10 Pilot (obrázek č. 3) je pokročilý desktop. Jeho prostředí je výslovně navrženo tak, aby byl snadný a hodnotně použitelný pro vývoj, práci nebo trénink založený na principu jazyka Handel-C a pro vytváření hardware v daném procesoru.



***Obrázek 3:*** *RC10 Pilot*
Zdroj: www.celoxica.com

Deska je vytvořená z 1,5 milionů hradel ( Spartan 3 FPGA ) a je zabalená v balíčku se souborem obsahujícím kompletní knihovny určené k práci s návrhovými pomůckami ESL včetně DK Design Suite, PixelStreams obrazu a knihoven pro zpracování videa. V této době

se její cena pohybuje kolem 700 dolarů pro ziskové společnosti. Pro neziskové organizace, jako je například škola, je cena kolem 300 dolarů. RC10 Pilot je vhodný pro originální vývoj různých aplikací:

a) zpracování a práce s videem
b) využití jednotlivých aplikací, jako jsou například osciloskop, logický analyzátor, záznam dat
c) počítačové vidění a kontrola
d) šifrování dat a vstupní detekce

K RC10 Pilot je přiložen softwarový balíček, který je určen pro podporu těchto aplikací a obsahuje tyto součásti:

a) utilitu pro nahrání a stažení souboru (*.bit) do FPGA a paměti flash na desce
b) programy ovládající vysokorychlostní integraci a komunikaci větší než je 25Mbytes/s s FPGA aplikacemi
c) podpora knihoven pro DK Design Suite dovolující rychlý vývoj aplikací využívajících rozhraní API, než přímé komunikace s I/O zařízením dovolující režim "Software-like"
d) podpora knihoven k ulehčení návrhu hardware pomocí jazyka „C" pro MicroBlaze procesor, včetně ukázkových příkladů
e) utility a programy, které dovolují použití desky na různých pracovištích, např. osciloskop, inteligentní I/O zařízení, logický analyzátor

Celoxica podporuje PixelStreams, což je knihovna, pro číslicové zobrazení a zpracování videa. Ta dovoluje velmi rychlý vývoj video aplikace, buď použitím interaktivního GUI prostředí nebo použitím knihoven funkcí.

### 2.2.1. Funkční bloky RC10 Pilot

Jednotlivé funkční bloky mají svůj zásadní význam pro další vývoj a navrhování uživatelských aplikací. Tyto aplikace mohou například pracovat se zvukem, videem, zpracováním vstupního signálu, komunikaci přes sběrnici CAN, komunikací a napájením přes rozhraní USB2.0. Mohou v sobě skrývat mnoho jiných vlastností, které pracují s těmito funkčními bloky:

**FPGA**

- Xilinx Spartan 3 XC3S1500L-4

**Paměť**

- Flash paměť 16MByt

**Rozhraní**

- USB 2.0 Slave Port,
- CAN sběrnice
- RS232 port
- PS/2 porty pro připojení klávesnice a myši
- Pěti polohový mini-joystick
- JTAG konektor

**Video**

- Konektor pro CMOS kameru ( 2 Mpixel )
- Standardní 15pin D konektor pro 18bit VGA výstup

**Analog a zvuk**

- Dva ADC kanály ( 10bit, 65 MHz )
- Sériový zvukový výstup ( 1 bit DAC )

**Display**

- 8 programovatelných LED diod
- 3 sedmisegmentovky

**Mikrokontroler**

- FX2 mikrokontroler
- USB port
- FPGA konfigurace/rekonfigurace
- Flash paměť

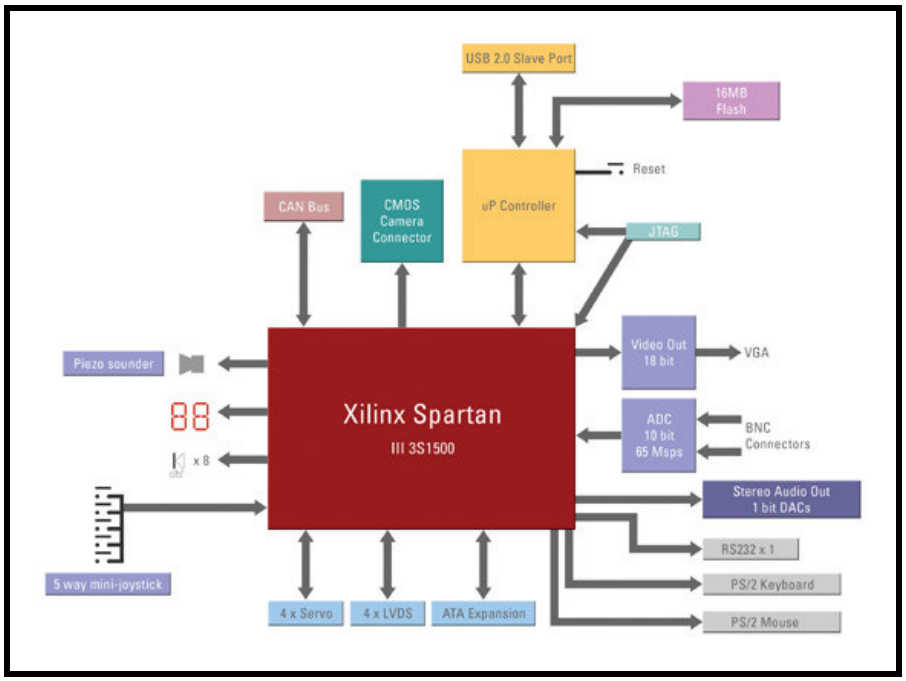
**Rozšířené rozhraní o**

- 4 LVDS konektor
- 4 konektory pro standardní servo

- 50 ATA pinů

  - 33 hlavních I/O pinů ( kompatibilními s ATA UDMA-4 nebo vyššími)
  - 3 piny pro napájení ( +12, +5, +3.3V)
  - 2 piny pro hodiny

Na (obrázku č.4) můžeme vidět blokový diagram desky RC10Pilot, který se skládá z jednotlivých výše uvedených funkčních bloků.




***Obrázek 4:*** *Schéma funkčních bloků desky RC10 Pilot*
Zdroj: www.celoxica.com

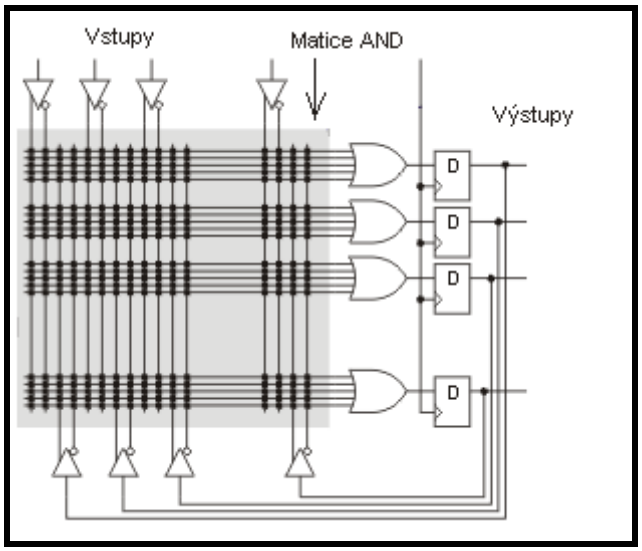
## 2.3. Číslicově programovatelné součástky

Programovatelné součástky jsou označovány jako PLD (Programmable Logic Device). Podle vnitřní struktury můžeme číslicově programovatelné součástky rozdělit do tří skupin:

a) klasické PLD
b) komplexní PLD
c) obvody typu FPGA.

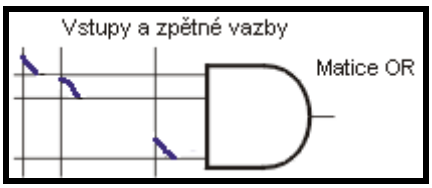
### 2.3.1. Klasické PLD

Obvody jsou charakteristické vnitřní strukturou podle (obrázku č.5).




***Obrázek 5:*** *Vnitřní struktura klasického PLD*
Zdroj: www.hw.cz

Každá vodorovná čára v programovatelné matici AND představuje jedno součinové hradlo. Na vstupy každého hradla lze připojit libovolnou kombinaci vstupních signálů, zpětných vazeb a jejich negací. Počet vstupů každého součinového hradla je však omezen. Zapojení součinového hradla je znázorněno na (obrázku č.6).




***Obrázek 6:*** *Součinové hradlo*
Zdroj: www.hw.cz

Modré spoje na tomto obrázku představují programovatelné spínače. Jejich realizace závisí na výrobní technologii obvodu. Například v technologii CMOS jsou spínače realizovány stejnými principy jako u pamětí PROM, EPROM nebo EEPROM, u bipolárních obvodů se jednalo o pojistku, která se při programování obvodu "přepálila" proudovým impulsem. Složitější obvody z kategorie FPGA mívají často spínače řízeny statickou pamětí RAM.

Do skupiny klasických PLD je možné zařadit obvody následujících typů:

a) PLA

Obvody typu PLA (Programmable Logic Array) mají obecnější strukturu než PAL na horním obrázku. Mají totiž programovatelnou nejenom matici logických součinů, ale i následující matici logických součtů.

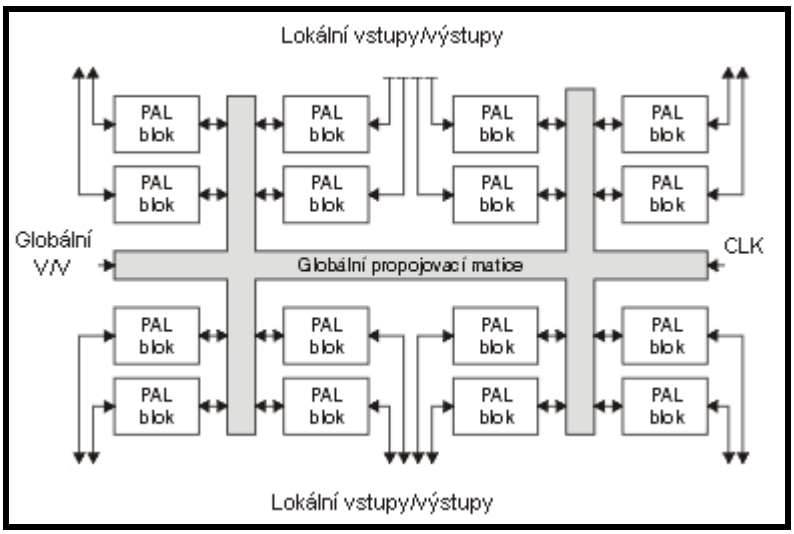
b) PAL

Obvody typu PAL (Programmable Array Logic) mají strukturu podle výše uvedených obrázků. Některé starší typy neměly například výstupní registry, takže byly vhodné spíše pro kombinační logiku.

### 2.3.2. Komplexní PLD

Klasické obvody PLD mají velmi omezené prostředky, takže umožňují realizovat pouze jednodušší funkce. Výrobci proto začali vytvářet více takovýchto obvodů na jednom čipu spolu s nutnými prostředky pro propojení. Takovéto obvody se většinou označují jako

CPLD (Complex Programmable Logic Device). Typická struktura obvodu CPLD je znázorněna na (obrázku č.7).

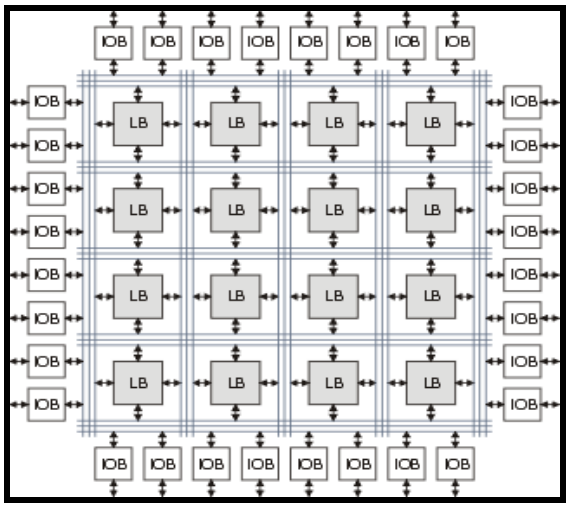



***Obrázek 7:*** *Struktura obvodu CPLD*
Zdroj: www.hw.cz

Výrobci CPLD používají trochu jinou interní strukturu obvodů, ale většinou vychází z tohoto schématu. CPLD od různých výrobců se obvykle liší v provedení bloků vlastní programovatelné logiky, i když většinou vychází z klasické struktury PAL.

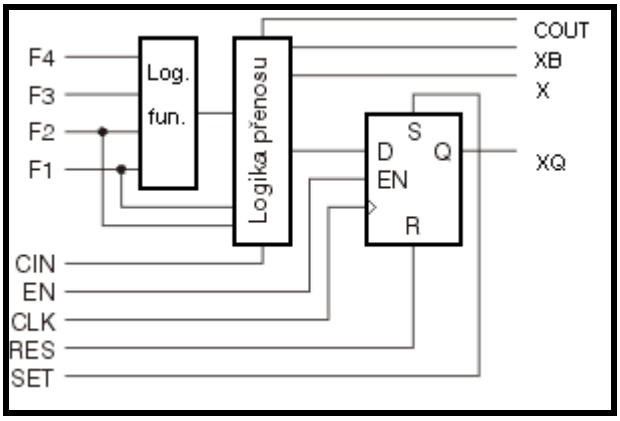
### 2.3.3. Obvody FPGA

FPGA (Field Programmable Gate Array) jsou obvody, které mají z programovatelných obvodů nejobecnější strukturu a obsahují nejvíce logiky. V současné době největší obvody FPGA obsahují až 6 milionů ekvivalentních hradel (typické dvou-vstupní hradlo NAND). Strukturu obvodu FPGA znázorňuje následující (obrázek č.8).




***Obrázek 8:*** *Struktura obvodu FPGA*
Zdroj: www.hw.cz

Bloky označené IOB (Input/Output Block) představují vstupně-výstupní obvody pro každý v-v pin FPGA. Tyto bloky obsahují registr, budič, multiplexer a ochranné obvody. Bloky LB (Logic Block) představují vlastní programovatelné logické bloky. Skupiny bloků mohou být různě propojeny globální propojovací maticí. Nejpoužívanější struktura konfigurovatelného logického bloku je znázorněna na (obrázku č.9).




***Obrázek 9:*** *Struktura logického bloku LB*
Zdroj: www.hw.cz

Obvod FPGA umožňuje propojit některé signály logických bloků přímo se sousedním, bez nutnosti využívat globální propojovací matici. Tyto spoje mají mnohem menší zpoždění a umožňují realizovat například rychlé obvody přenosu, které jsou nezbytné pro sčítačky nebo násobičky. Většina moderních FPGA obsahuje několik bloků rychlé statické paměti RAM. Velmi často obvody FPGA obsahují PLL (Phase Locked Loop) nebo DLL (Delay Locked

Loop), které jsou perspektivní pro obnovení hodinového signálu, pro násobení nebo dělení jeho frekvence.

## 2.4. FPGA Xilinx

V tomto odstavci bych rád uvedl jednotlivé typové řady programovatelných polí, které jsou podporovány firmou Xilinx.

| Název | $V_{cco}$ [V] | $V_{ccint}$ [V] | Počet hradel [tis] | Blok RAM [kbit] | Pouzdro |
|---|---|---|---|---|---|
| Virtex-4 | 1,14 | 1,14-1,26 | - | - | - |
| Virtex-II | 1,5 | 1,5-3,3 | 40-8000 | 64-832 | BGA |
| Virtex-E | 1,8 | 1,8-3,3 | 50-3200 | 32-128 | BGA |
| Virtex | 2,5 | 2,5-3,3 | 50-1000 | 32-128 | BGA |
| Spartan-3E | 1,2 | 1,2-3,3 | 1200 | 504 | BGA |
| Spartan-3/3L | 1,2 | 1,2-3,3 | 1500 | 576 | BGA |
| Spartan-IIE | 1,8 | 1,8-3,3 | 50-300 | 32-64 | BGA |
| Spartan-II | 2,5 | 2,5-3,3 | 15-200 | 16-56 | BGA |

Všechny FPGA používají konfiguraci pomocí statické paměti RAM. To znamená, že po připojení napájení musíme vždy znovu nahrát konfiguraci. Výhodou tohoto řešení je téměř nekonečná reprogramovatelnost FPGA a také rychlost. Konfigurační propojky pracující na principu paměti RAM jsou totiž rychlejší než přepínače na principu např. EEPROM.

- Distribuovaná paměť RAM je tvořena multiplexery logických buněk. Při jejím použití se připravujeme o využitelné prostředky pro ostatní logiku. Obvykle může být tato paměť využita jako dualportová nebo klasická jednoportová synchronní RAM nebo jako ROM.

- Počet systémových hradel je zavádějící. Toto číslo vyjadřuje kompletní rozsah interní logiky včetně konfiguračních prostředků, blokových pamětí apod. Mnohem důležitější údaj je počet logických buněk. Interní struktura logické buňky je vždy schématicky znázorněna v katalogu.

- Blokovou paměť RAM tvoří skutečné bloky synchronní statické RAM paměti. Bloky mají dvojnásobné adresové i datové sběrnice a lze je využít jako dualportovou paměť.

### 2.4.1. FPGA Spartan - 3/3L typu XC3S1500L

V následující tabulce jsou uvedeny jednotlivé vlastnosti dalších čipů Spartan 3/3L. V této práci je používáno zařízení typu XC3S1500, které je označeno černou šipkou. Z tabulky můžeme například vyčíst počet systémových hradel, velikost blok RAM, počet maximálně užívaných I/O zařízení.

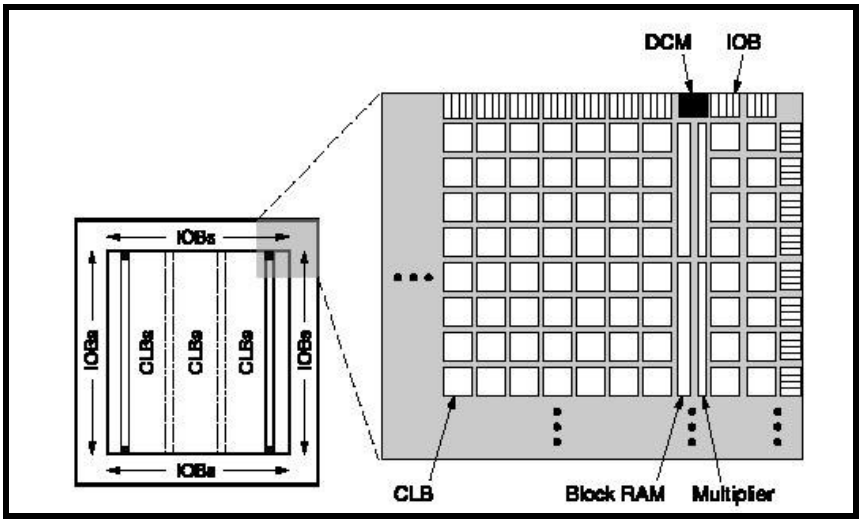
| Device | System Gates | Equivalent Logic Cells[1] | CLB Array (One CLB = Four Slices) | | | Distributed RAM Bits (K=1024) | Block RAM Bits (K=1024) | Dedicated Multipliers | DCMs | Maximum User I/O | Maximum Differential I/O Pairs |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Rows | Columns | Total CLBs | | | | | | |
| XC3S50[2] | 50K | 1,728 | 16 | 12 | 192 | 12K | 72K | 4 | 2 | 124 | 56 |
| XC3S200[2] | 200K | 4,320 | 24 | 20 | 480 | 30K | 216K | 12 | 4 | 173 | 76 |
| XC3S400[2] | 400K | 8,064 | 32 | 28 | 896 | 56K | 288K | 16 | 4 | 264 | 116 |
| XC3S1000[2, 3] | 1M | 17,280 | 48 | 40 | 1,920 | 120K | 432K | 24 | 4 | 391 | 175 |
| XC3S1500[3] | 1.5M | 29,952 | 64 | 52 | 3,328 | 208K | 576K | 32 | 4 | 487 | 221 |
| XC3S2000 | 2M | 46,080 | 80 | 64 | 5,120 | 320K | 720K | 40 | 4 | 565 | 270 |
| XC3S4000[3] | 4M | 62,208 | 96 | 72 | 6,912 | 432K | 1,728K | 96 | 4 | 712 | 312 |
| XC3S5000 | 5M | 74,880 | 104 | 80 | 8,320 | 520K | 1,872K | 104 | 4 | 784 | 344 |

### 2.4.2. Celkový pohled na architekturu FPGA Spartan 3/3L

Architektura rodiny programovatelných logických obvodů Spartan 3/3L spočívá v pěti základních programovatelných částech:

a) Konfigurovatelné logické bloky (CLBs) obsahují RAM založené na Look-Up Tables (LUTs), které tvoří logiku a ukládají části, které mohou být jako flip-flops nebo latches.
b) I/O bloky (IOB) ovládají tok dat mezi jednotlivými I/O piny a vnitřní logikou zařízení. Každý I/O blok podporuje obousměrný tok dat a třístavové operace.
c) Blok RAM poskytuje ukládání dat do paměti o velikosti 18Kbit daulport bloků.
d) Bloky pro součin přijmou dvě 18-bit binární čísla jako vstup a vypočítají výsledek 36 bitů.
e) Řídicí bloky digitálních hodin (DCM) umožňují automatickou kalibraci, řeší plně digitální rozvětvování, zpoždění, násobení, dělení a fázový posun hodinového signálu.

Tyto části jsou organizovány na (obrázku č.10).

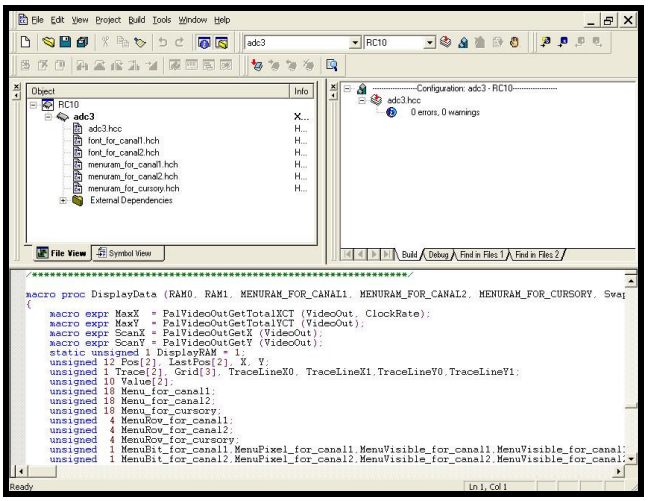



***Obrázek 10:*** *Architektura FPGA Spartan 3/3L*
Zdroj: Technická dokumentace k desce RC10 Pilot

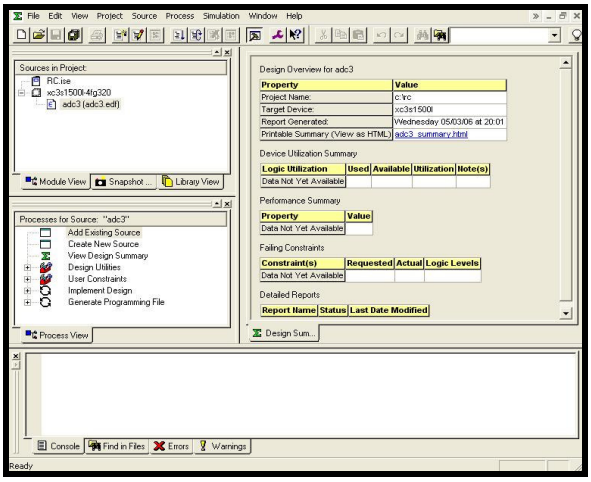
# 3. Vývojové prostředí

## 3.1. Software použitý při tvorbě aplikace

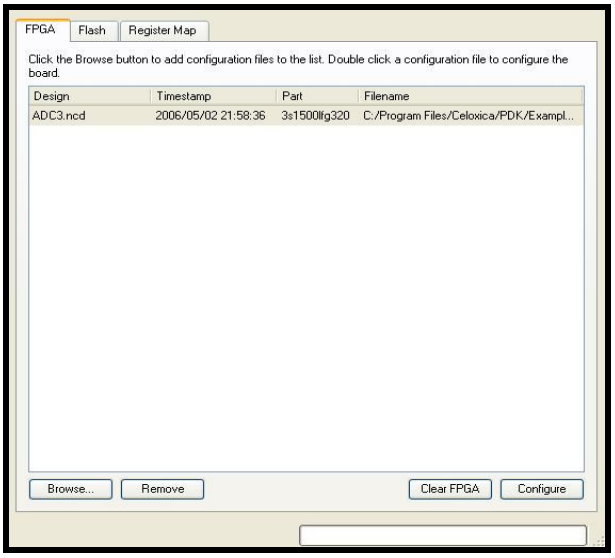
Aplikace je programována v prostředí DK4.0 (obrázek č.11), které umožňuje psát program v jazyce Handel-C. Je podporováno softwarem .NET (Visual Studio) od firmy Microsoft, což v sobě obsahuje všechny potřebné knihovny na bázi jazyka C využívané při kompilaci programu. Program je pomocí DK4.0 kompilován do souboru *.edf. Pomocí programu ISE 7.1 (obrázek č.12) od firmy Xilinx je tento soubor kompilován do souboru *.bit, který je přímo nahrán do FPGA přes FTU software (obrázek č.13). Na originálním CD je software FTU spolu s Platform Developer's Kit (PDK) a ovladačem pro RC10 Pilot pro připojení k počítači pomocí rozhraní USB.



***Obrázek 11:*** *Programové prostředí DK4.0*



***Obrázek 12:*** *Programové Prostředí ISE 7.1*




***Obrázek 13:*** *Programové Prostředí FTU*

## 3.2. Hardware použitý při tvorbě aplikace

Pro práci na této aplikaci jsem použil:

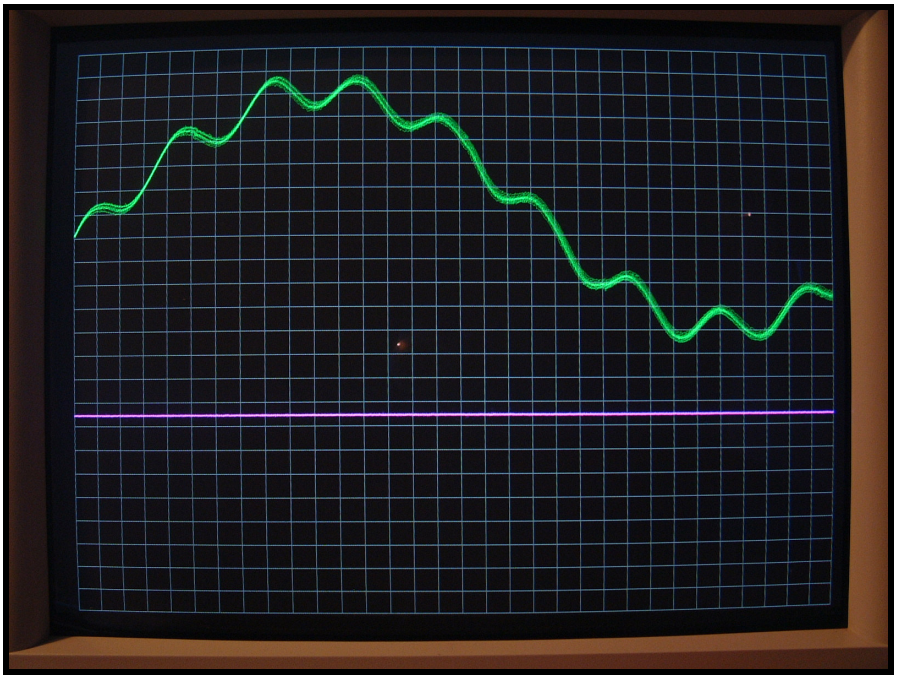
- notebook značky Acer TravelMate 4150,
- desku RC10 Pilot
- VGA monitor značky SCOT

***Obrázek 14:*** *Využívaný hardware*

# 4. Popis hardware implementace a procedur

## 4.1. Minimální hardware řešení od firmy Celoxica

Jak bylo zmíněno v úvodu, úkolem této práce bylo vzít minimální hardware řešení ‘osciloskop‘ viz (obrázek č.15) a rozšířit ho o další hardware funkční bloky.



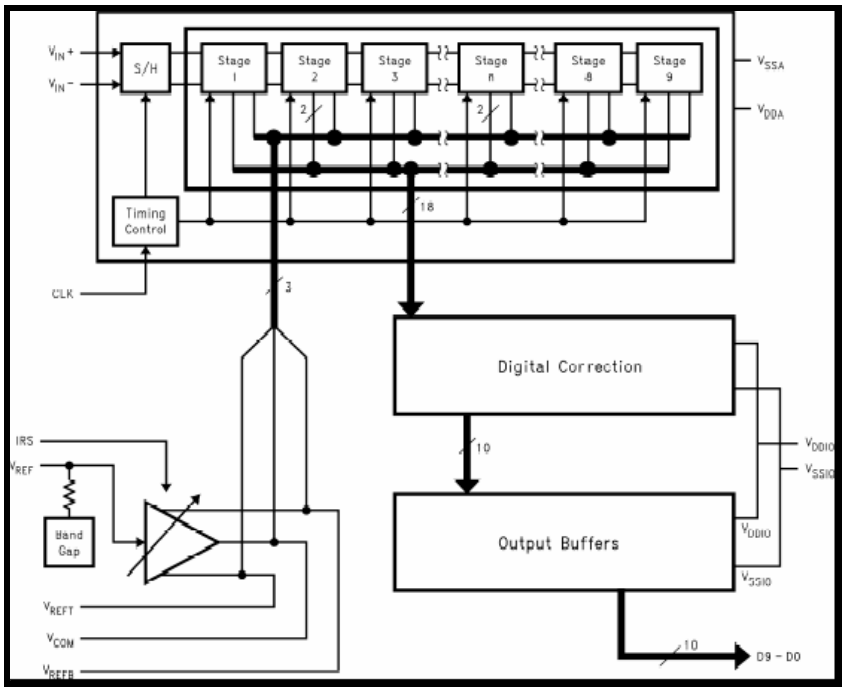
***Obrázek 15:*** *Demo programu osciloskop*

Toto řešení mělo připravené rozhraní pro ADC převodníky a stavový automat zajištující ukládání hodnot z ADC převodníků do čtyř dualport RAM pamětí. Hodnoty z pamětí jsou zpracovány a vykresleny na obrazovku v podobě vstupního signálu pomocí trvale běžícího rozhraní pro vykreslení na obrazovku. ADC převodník a ukládání navzorkovaných hodnot do pamětí z ADC převodníků je popsán v následující kapitole.

### 4.1.1. ADC převodník použitý na RC10 Pilot

Na RC10 Pilot je použit převodník typu ADC10065. Je to monolitický CMOS analog – digitální převodník schopný převodu z analogového signálu do 10 bitového slova v 65 MHz za sekundu. Tento převodník používá diferenční architekturu s digitální opravou chyb

a na čipu má umístěn vzorkovací obvod s pamětí, který poskytuje kompletní převod a minimalizuje příkon. Diferenční vstupy poskytují plný rozsah v rozmezích 1V P – P, 1,5V P-P, 2V P-P, s možností jednoduchého vstupu. Plné použití diferenčního vstupu je doporučen pro optimální výkon. Vnitřní reference 1,2V je užitá pro nastavení ADC rozsahu stupnice a také dovolí ochránit uživatele před změnou referenčního napětí pro aplikace, které požadují přesnost. Na (obrázku č.16) je vidět vnitřní zapojení ADC převodníku.



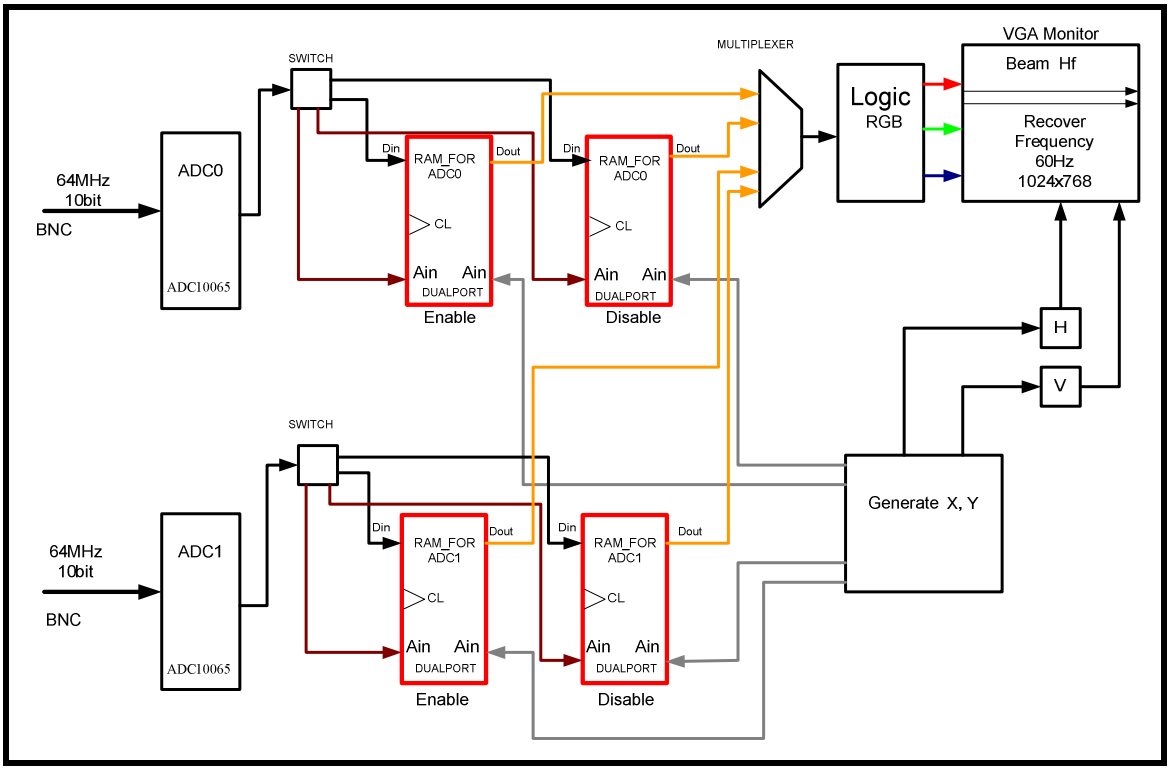

***Obrázek 16:*** *Vnitřní zapojení ADC převodníku*
Zdroj: www.national.com

### 4.1.2. Ukládání vzorků z ADC převodníků do dualport RAM pamětí

Na desce RC10 Pilot jsou umístěny dva vstupní signály 10bit, v zobrazení s periodou 64MHz, na které můžu pomocí BNC konektoru připojit vstupní měřený signál. Po připojení přivádím vstupní signály na ADC převodníky, jejichž rozsah je nastaven na 2V. Získané hodnoty ADC převodníků ukládám do 4 dualportových RAM pamětí. Pro ukládání a vzorkování je použita makro proceduru „CaptureData“. Tato procedura je volána v hardware implementaci dvakrát pro každý signál zvlášť. Na začátku této procedury je umístěn stavový automat pro nastartování ADC převodníků, který je součástí příkazu „par“. Vše, co je definováno v tomto příkazu se provádí paralelně, každý hodinový cyklus. Součástí jsou dvě nekonečné smyčky. V první smyčce se čtou data z ADC převodníků, ve druhé jsou

načtená data zapisována do příslušných pamětí. Zvlášť používám dvě paměti pro signál ADC0 (zelený signál) a ADC1 (oranžový signál). Při naplnění první paměti dojde k přepnutí na druhou paměť. Do paměti se ukládá 1024x10 bit hodnot, které po synchronizaci s displejem vykresluji v podobě vstupního signálu na monitor. Pro vykreslování na monitor je spuštěna makro procedura nazvaná jako „DisplayData“. Tato procedura je vysvětlena v některé z následujících kapitol.

Na (obrázku č.17) je naznačeno ukládání na vzorkovaných dat do paměti. Jakmile se paměť, která je 10-bitová naplní hodnotami z ADC převodníku, dojde k přepnutí na druhou paměť a původní paměť se začne vyprazdňovat. Dochází k tzv. „houpačkovému efektu“, přičemž data z první naplněné paměti jsou přivedena na multiplexor. Výstup tohoto multiplexoru je přiveden na logický obvod pro RGB barvy. Po vyhodnocení tímto obvodem jsou data na obrazovku vykreslena v podobě výstupního signálu. Tento proces ukládání do paměti se opakuje po celou dobu. Do ADC pamětí navíc ještě vstupují adresy z obvodu pro VGA rozklady. Tento obvod určuje horizontální a vertikální rozklad na obrazovce..




***Obrázek 17:*** *Vzorkování, ukládání do paměti a vykreslení na monitor*

## 4.2. Rozšířené hardware řešení

Při rozšiřování tohoto hardware řešení vznikla první otázka, jak vykreslit znak na obrazovce. Princip vykreslení znaku je vysvětlen v následující kapitole. Hardware řešení jsem rozšířil o uživatelské menu, v kterém se lze pohybovat pomocí mini-joysticku a je rozděleno na tři části:

- První řádek menu slouží pro změnu parametrů prvního signálu, který poznáme tak, že má zelenou barvu.
- Druhý řádek menu slouží pro změnu parametrů druhého signálu, který poznáme tak, že má oranžovou barvu.
- Třetí řádek menu slouží pro ovládání měřicích kurzorů a vykreslení naměřených hodnot.

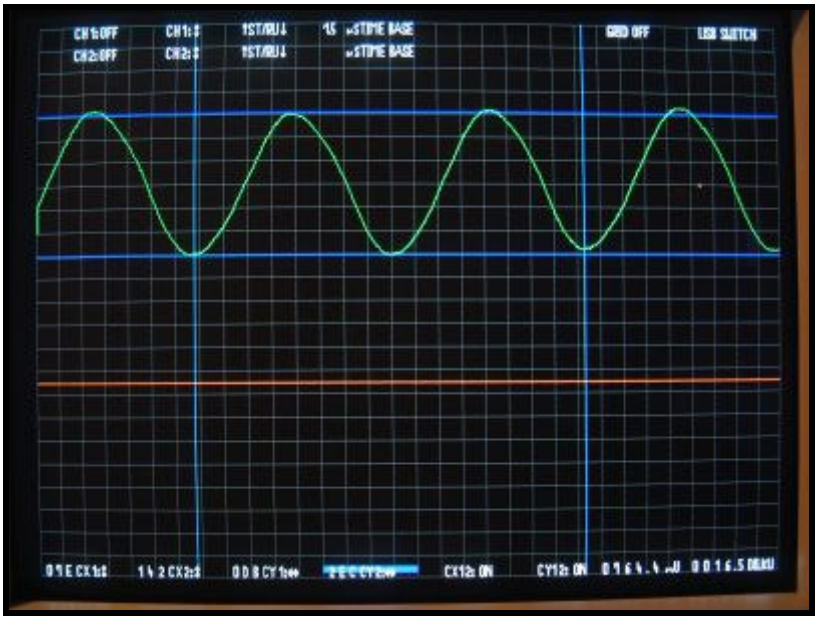
V prvním a druhém řádku menu je možno měnit následující parametry signálů:

- vypnutí a zapnutí vstupního signálu
- posun signálu nahoru a dolů
- zastavení signálu
- hodnotu časové základny od 0,5-10 µS
- vypínání a zapínání pomocné měřicí síťky

Poslední řádek menu slouží k obsluze měřicích kurzorů:

- poloha a pohyb kurzoru $X_1$ v ose x pro měření napětí, udávána v hexadecimální hodnotě
- poloha a pohyb kurzoru $X_2$ v ose x pro měření napětí, udávána v hexadecimální hodnotě
- poloha a pohyb kurzoru $Y_1$ v ose y pro měření periody, udávána v hexadecimální hodnotě
- poloha a pohyb kurzoru $Y_2$ v ose y pro měření periody, udávána v hexadecimální hodnotě
- zapnutí obou kurzorů v ose x
- zapnutí obou kurzorů v ose y
- vykreslení hodnoty v digitální podobě rozdílu kurzorů X v milivoltech
- vykreslení hodnoty rozdílu v digitální podobě kurzorů Y v dílcích

Na (obrázku č.18) je vidět, výsledné řešení po rozšíření minimálního hardware řešení.



***Obrázek 18:*** *Rozšířené demo programu osciloskop*

### 4.2.1. Návrh obvodového řešení pro vykreslení uživatelského menu na obrazovku

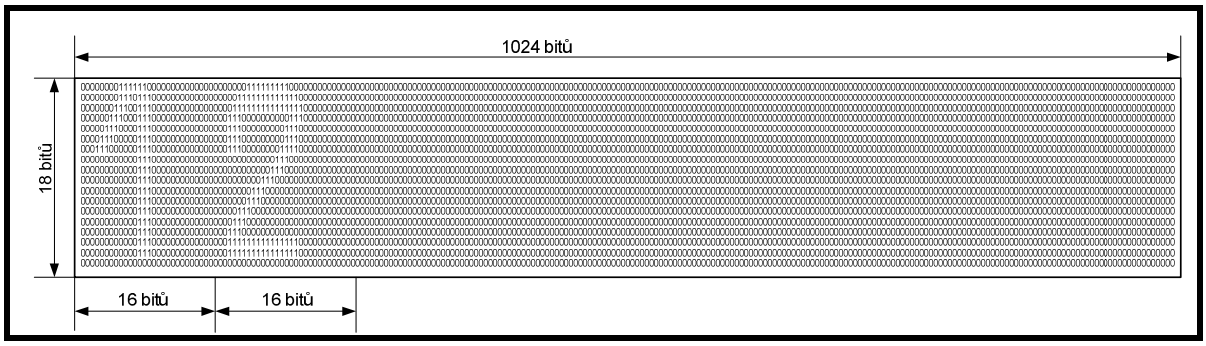
Pro vykreslení obsahu RAM pamětí na monitor v hardware návrhu používám makro proceduru „DisplayData“. V příkazu „par“ volám stavový automat pro spuštění videa a vykreslení na monitor. V této proceduře je sekvenční příkaz pro spuštění video výstupu , aby monitor nebyl černý. V nekonečné smyčce je vložen příkaz „par“, kde se čtou hodnoty z RAM pamětí a vykreslují na pevně stanovenou pozici na monitoru. Rozlišení monitoru 1024x768 je dáno vnitřními hodinami osciloskopu, které jsou stanoveny na 64 MHz. Obnovovací frekvence monitoru je 60Hz. Je zde určena pozice pro vykreslení dvou vstupních signálů. Nula prvního signálu je na pozici 256 a pro druhý signál je na 512. řádku. Tuto nulu nelze z části považovat za přesně absolutní, jelikož přesně nejde zjistit na jakém rozsahu je nastavena nula použitých převodníků. Pro nastavení nuly se musí provést kalibrace zkratováním vstupních signálů a pomocí posunu signálů nahoru a dolu v menu osciloskopu tuto absolutní nulu nastavit. Hodnoty z RAM pamětí pro oba signály dělím dvěma, aby je bylo možno vykreslit. Na osciloskopu je proto viditelné, že dojde ke zkreslení měřeného signálu, když velikost vstupního signálu dosáhne rozmezí většího než 0 až 512 řádků. To samé platí i pro druhý signál v rozmezí 256 až 768 řádků. Součástí vykreslovací procedury je i příkaz pro vykreslení měřicí mřížky, uživatelského menu a pomocných měřicích kurzorů.
Pro příklad zde uvedu jenom část hardware, jak je implementován v jazyce „Handel-C“.

```
seq
{
  PalVideoOutEnable (VideoOut);
  while (1)
   {
   par
    {
    /*
     * Read the value from RAM
     */
     Value[0] = RAM0[DisplayRAM].R[X <- 10];
     Value[1] = RAM1[DisplayRAM].R[X <- 10];

     Menu_for_canal1            =  MENURAM_FOR_CANAL1.R[X <- 10];
     MenuRow_for_canal1         =  (Y - 752)[3:0];
     Menu_for_canal2            =  MENURAM_FOR_CANAL2.R[X <- 10];
     MenuRow_for_canal2         =  (Y - 752)[3:0];
     Menu_for_cursory           =  MENURAM_FOR_CURSORY.R[X <- 10];
     MenuRow_for_cursory        =  (Y - 752)[3:0];

    /*
     * Calculate position on screen
     */

     Pos[0] = (unsigned)(511 - ((0 @ Value[0]) >> 1));
     Pos[1] = (unsigned)(767 - ((0 @ Value[1]) >> 1));
     LastPos[0] = Pos[0];
     LastPos[1] = Pos[1];
```

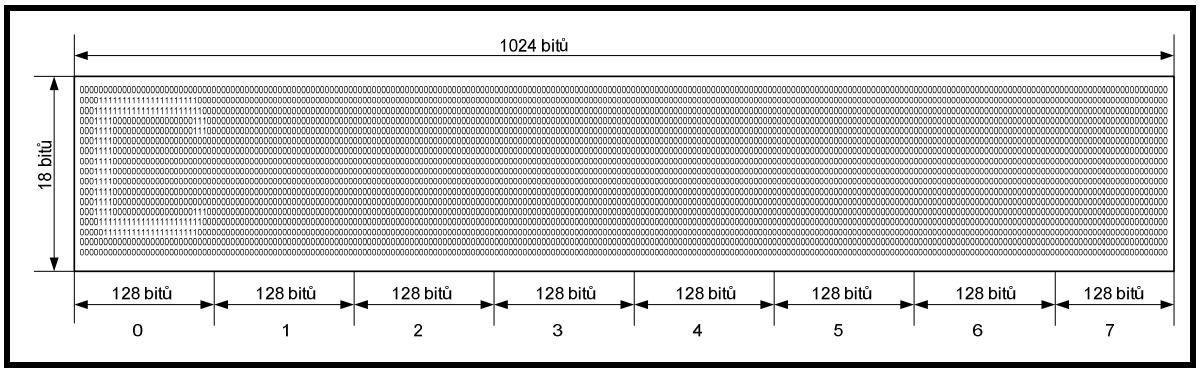
Na konci tohoto makra volám funkci ( PalVideoOutWrite ), která vytváří hardware pro vykreslování RGB barev na monitor, kde se vyhodnocuje, jakou barvou má být pixel vykreslen. V této funkci je definováno, jakou barvu z RGB budou mít ADC signály z převodníků, uživatelské menu, pomocné měřicí kurzory, pomocná měřicí síť a kurzor pro uživatele, který označuje, kde se uživatel v menu osciloskopu nachází.

### 4.2.2. Princip dualport RAM pamětí pro menu a fonty

Pro vytvoření uživatelského menu jsem použil sedm dualport RAM pamětí, které jsou definované v hardware řešení pomocí knihovny. Asi se každý zamyslí nad tím proč zrovna sedm? Pro objasnění této otázky, využívám pro vykreslení menu osciloskopu tři paměti a čtyři paměti pro vykreslení fontů. Na (obrázku č.19) je nakreslená vnitřní struktura pamětí pro vykreslování fontů a na (obrázku č.20) je vnitřní struktura pamětí pro uživatelské menu osciloskopu.




***Obrázek 19:*** *Vnitřní struktura dualport RAM paměti pro fonty*

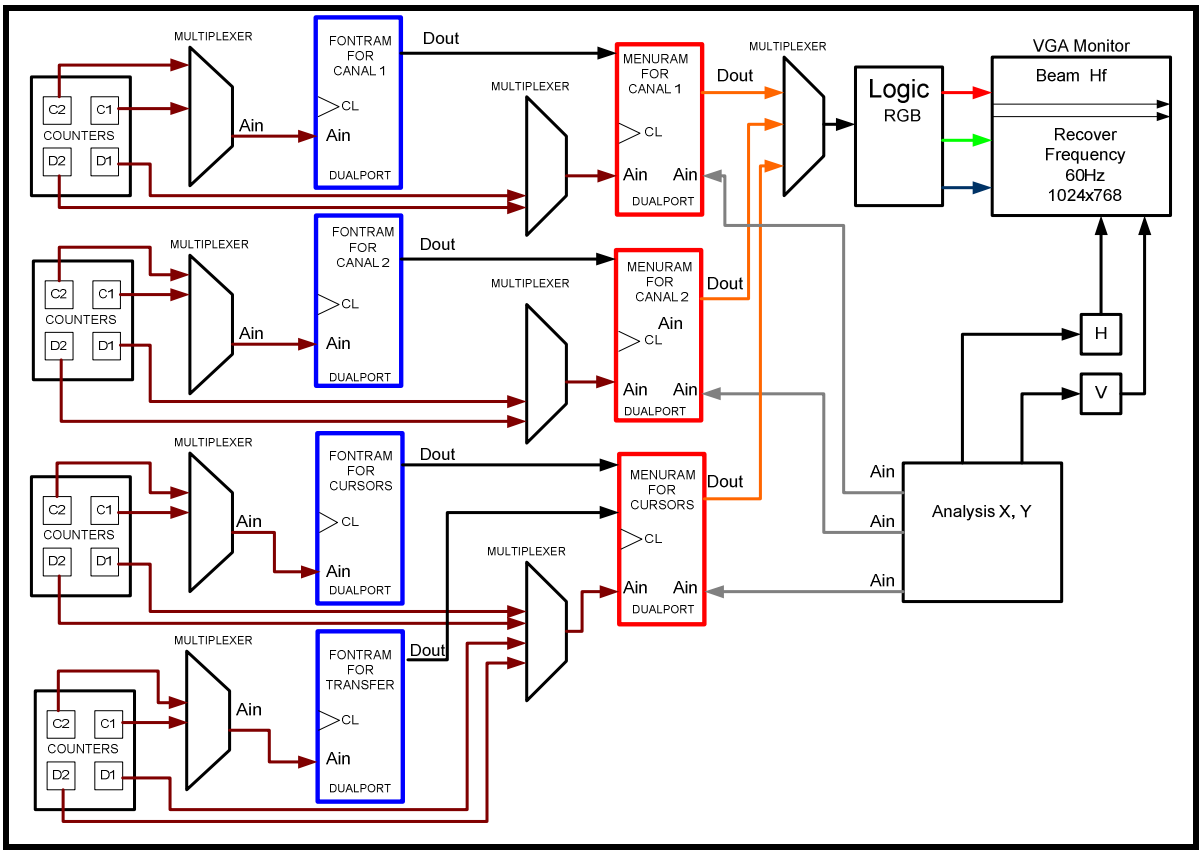



***Obrázek 20:*** *Vnitřní struktura dualport RAM paměti pro menu*

Jak je vidět na obrázku, paměti jsou 18 bitů široké, 1024 bitů dlouhé a jsou vyplněny samými nulami. Na prvních dvou pozicích v paměti pro fonty jsou vykresleny číslice. V paměti pro menu osciloskopu je vykresleno písmeno „C“. Každá paměť pro menu je rozdělena po 128 bitech, tedy k dispozici je sedm možných pozic v každém řádku uživatelského menu. Paměti pro fonty jsou vytvořeny na stejném principu s tím rozdílem, že se v paměti pohybuji o délce 16 bitů a na předem stanovené pozici v menu osciloskopu vykreslím znak.

Asi každý by se zeptal, jak je možné, že vykresluji na obrazovku paměť pro menu a v té ještě znaky z paměti fontů. Tento princip můžeme vidět na obrázku č.21. Z důvodu jednoduchosti porozumění budu popisovat pouze vykreslování prvního řádku v menu, jelikož zbylé řádky pracují na stejném principu. V hardware implementaci se využívají pro vykreslování prvního řádku menu dvě dualportové RAM paměti. Jedna pro menu (MENURAM FOR CANAL1 - obrázek č.20) a druhá pro fonty (FONTRAM FOR CANAL1 - obrázek č.19). Na vstup multiplexoru jsou přivedeny adresy z čítačů, po vyhodnocení multiplexorem se nastaví v paměti fontů vstupní adresa, ze které se budou přenášet data v podobě znaku (16 x 18 bitů) a vykreslovat na předem stanovenou pozici v paměti pro menu. Tento princip nastavení vstupní adresy vykreslení je obdobný i u paměti pro menu, do které ještě navíc vstupují adresy z obvodu pro VGA rozklady. Tento obvod určuje horizontální

a vertikální rozklad na obrazovce. Výstupní data z paměti pro menu jsou přivedeny na multiplexer. Po vyhodnocení multiplexorem jsou data přivedena na vstup logického obvodu pro RGB barvy. Tento obvod obsahuje logiku, která vyhodnocuje, v jaké barvě budou daná data vykreslena na obrazovce v délce 1024 x 18 bitů.




***Obrázek 21:*** *Blokové schéma rozšířeného řešení pro vykreslování pamětí menu a fonty na obrazovku*

Další možností, jak by mohl být znak vykreslován, je použití zvláštní paměti pro obrazovku. Při uvažování nad touto variantou jsem dospěl k závěru, že bych potřeboval sto pamětí pro vykreslování jednotlivých pixelů na obrazovku a bylo by velmi obtížné vykreslovat obsahy těchto pamětí a k tomu ještě obsahy pamětí z ADC převodníků. Proto jsem tuto variantu nemohl použít, protože se snažím o to, aby aplikace byla co nejméně hardware náročná a byla potenciálně použitelná i na desce s logickým obvodem Spartan 3 XC3S400. Tento obvod má pouze 16 blokových RAM pamětí a je podstatně levnější a více dostupnější.

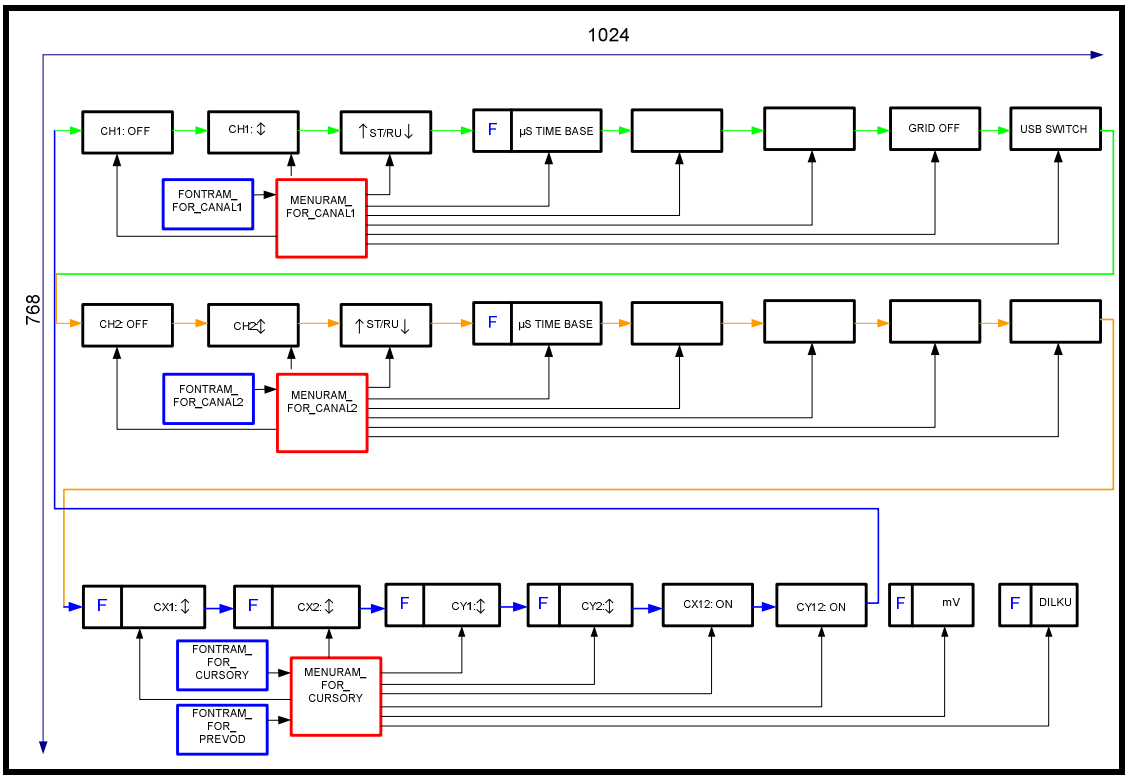
Pro názornost bych zde uvedl část hardware napsaného v jazyce „Handel-C“, kde se při pohybu mini-joystickem na čtvrté pozici v paměti pro menu kanálu jedna vykresluje hodnota časové základny z paměti fontů.

```
/*
 * Time base for canal1
 */
if (Offset[2]==3)
{
    if (RC10ButtonUpRead () || RC10ButtonDownRead ())
    {
         if (RC10ButtonUpRead ())
            if (ClockRateDiv[0]= =11)
                 delay;
            else
                 ClockRateDiv[0] ++;
            else
            {
            if (ClockRateDiv[0]= =1)
                 delay;
            else
                 ClockRateDiv[0] --;
            }
                  RC10SevenSegWritePair (ClockRateDiv[0], 0, 0);

          par
          {
           F_Addr_For_Canal1  = 0 @ ClockRateDiv[0][3:0] @ (unsigned 4) 0;
           Rw_Addr_For_Canal1 = 0 @ Offset[2][2:0] @ (unsigned 7) 0x5;
           f_cnt_for_canal1   = 0;
          }
        do
        {
        MenuHigh_for_canal1 = (Rw_Addr_For_Canal1[9:7]==Offset[2][2:0]);

         par
         {
          MENURAM_FOR_CANAL1.D[Rw_Addr_For_Canal1]=
                MenuHigh_for_canal1 @ FontValue_for_canal1[16:0];
          F_Addr_For_Canal1++;
          Rw_Addr_For_Canal1++;
          f_cnt_for_canal1++;
          Done_Rw_Addr_For_Canal1_3 = (f_cnt_for_canal1 == 15);
         }
      }
      while (~Done_Rw_Addr_For_Canal1_3);
}
else
{
    delay;
}
```

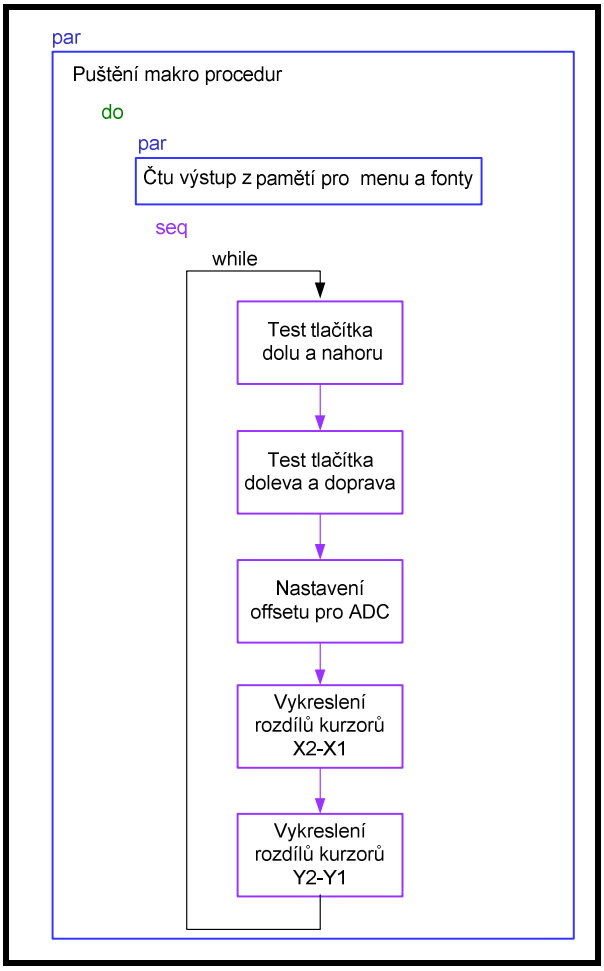
### 4.2.3. Princip pohybu v uživatelském menu

Uživatel při procházení v menu může vidět, kde se právě nachází pomocí zeleného kurzoru, který při přechodu na další řádek změní barvu na oranžovou a poté na modrou. První dva řádky menu jsou rozděleny na 8 možných položek. Třetí řádek jich má jenom 6, protože v dolní části obrazovky vykresluji hodnotu rozdílu měřicích kurzorů X a Y. Na (obrázku č.22) je blokově vidět, co přesně dualport RAM paměti pro menu vykreslují na monitor. Šipkami je znázorněno, jak se uživatel může pohybovat po menu a nastavovat si parametry osciloskopu. Tam, kde je zobrazeno písmeno F, je pozice, kde vykresluji číselné hodnoty z RAM paměti pro fonty. Prázdné pozice v menu má možnost uživatel procházet, ale neobsahují žádnou funkci pro nastavení signálů. Jak jsem uvedl již v úvodu, tyto nevyužívané pozice mohou sloužit pro další uživatelem definované funkce.

Proces pohybu v menu je součástí nekonečného hardware automatu, kde testuji pohyb tlačítek doleva (Left), doprava (Right), nahoru (Up), dolů (Down) a stisk (Fire). Testování běží sekvenčně v hlavním paralelním procesu tzv. “par“, který je součástí hlavní funkce (metody) main. Součástí tohoto procesu je další paralelní stavový automat, v kterém spouštím procedury maker pro oba signály, displej, převod voltů, sekund a dělám opět další paralelní proces, ve kterém stále čtu výstup z pamětí pro menu a fonty. Pro testování, na které pozici se uživatel nachází, vykresluji v paměti pro menu na posledním 18 bitu v délce 128 bitů samé jedničky. Na jaké pozice se uživatel v menu nachází, je zobrazeno i na sedmisegmentovce v hexadecimální podobě. Tento celý proces je pro lepší orientaci vidět na (obrázku č.23). Obrázek je kreslen z pohledu jazyka „Handel-C“. Ve skutečnosti pro představu čtenáře se za tímto obrázkem skrývá část hardware vykonávající tento proces.




***Obrázek 22:*** *Pohyb v uživatelském menu*




***Obrázek 23:*** *Proces pohybu v menu*

V jazyce „Handel-C“ je test tlačítka pro pohyb v menu osciloskopu doleva a doprava zapsán takto:

```
/*
 * Left and right controls menu highlight selection
 */

    if (RC10ButtonLeftRead ( ) || RC10ButtonRightRead ( ))
    {
            if (RC10ButtonLeftRead ( ))
            {
                  if (Offset[2] = = 0)
                  {
                        Offset[2] = 21 ;
                  }
                  else
                        Offset[2] -- ;
            }
                  else
                  {
                  if (Offset[2] = = 21)
                  {
                        Offset[2] = 0 ;
                  }
                  else
                        Offset[2] ++ ;
            }
             RC10SevenSegWritePair (Offset[2], 0, 0);
    }
    else
    {
           delay;
    }
```

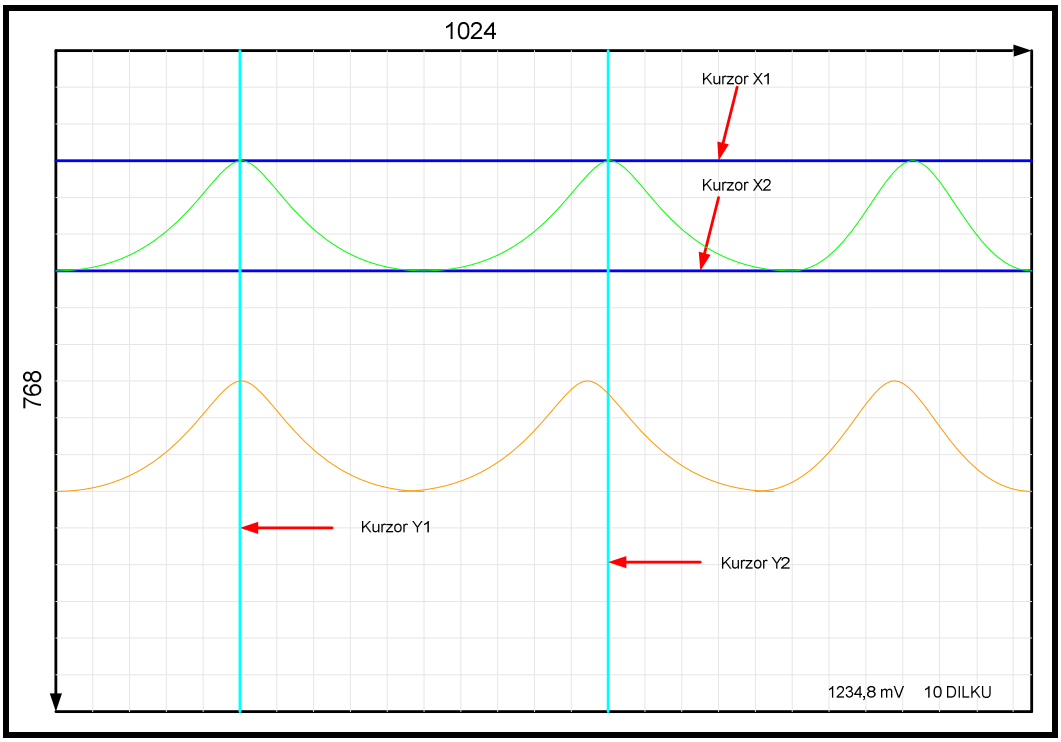
### 4.2.4. Pomocné měřicí kurzory v ose X a Y

Další částí, o kterou je osciloskop rozšířen, jsou dva měřicí kurzory v ose X pro měření amplitudy signálu v milivoltech a dva měřicí kurzory v ose Y pro měření počtu dílků. Tyto kurzory jsou vidět na (obrázku č.24). Pro ovládání kurzorů slouží třetí řádek v menu. První a druhá pozice v menu je určena pro pohyb kurzorů X1 a X2, které mají tmavě modrou barvu. Třetí a čtvrtá slouží k pohybu kurzorů Y1 a Y2, které mají světle modrou barvu. U těchto pozic je ještě vykreslována hexadecimální hodnota (pixel), na jaké se kurzor nachází. Na posledních dvou pozicích, na které se už uživatel kurzorem nedostane, je zobrazena přepočtená hodnota rozdílu kurzorů X na milivolty a rozdílu kurzorů Y na počet dílků. Vykreslení kurzorů se provádí v makro proceduře „DisplayData“. V této proceduře vytvářím hardware pro vykreslení měřicích kurzorů takto:

```
/*
 * Cursor x0,x1
 */

TraceLineX0 = (Y == (0 @ LineX0Pos)) && visible_cursor_x;
TraceLineX1 = (Y == (0 @ LineX1Pos)) && visible_cursor_x;

/*
 * Cursor y0,y1
 */

TraceLineY0 = (X == (0 @ LineY0Pos)) && visible_cursor_y;
TraceLineY1 = (X == (0 @ LineY1Pos)) && visible_cursor_y;
```

Pozice měřicích kurzorů při zapnutí je dána tak, aby kurzory nezasahovaly do uživatelského menu. Kurzory jsou vzájemně ošetřeny, nemohou se navzájem míjet. Jakmile dojde k tomu, že jeden kurzor narazí na druhý, dojde pouze k vzájemnému překrytí a jejich vzájemná hodnota je nulová. Tyto podmínky jsou stanoveny v hardware proto, aby když dochází k přepočtu rozdílu kurzorů na milivolty nebo dílky, nedošlo k zápornému výsledku. To by bylo pro uživatele osciloskopu matoucí a neodpovídalo by to skutečnému rozdílu kurzorů. Podmínky jsou v hardware umístěny tam, kde dochází k testování pohybu kurzorem nahoru nebo dolů. Pro představu bych uvedl část kódu v jazyce „Handel-C“, který představuje hardware pro testování tlačítka nahoru nebo dolů při pohybu kurzorem X1 a X2. Pro kurzory Y1 a Y2 tento kód nebudu uvádět, jelikož je založen na stejném principu.

```
if (Offset[2]==16)
{
    if (RC10ButtonUpRead () || RC10ButtonDownRead ())
    {
          if (RC10ButtonUpRead ())
          {
               if (LineX0Pos > 63)
               {
                  LineX0Pos --;
               }
             else
            {
                     delay;
            }
      }
      else
      {
          if (LineX0Pos < 736 && LineX0Pos < LineX1Pos)
          {
              LineX0Pos ++;
          }
             else
            {
                     delay;
            }
      }
      RC10SevenSegWritePair (LineX0Pos, 0, 0);
}
if (Offset[2]==17)
{
      if (RC10ButtonUpRead () || RC10ButtonDownRead ())
      {
              if (RC10ButtonUpRead ())
              {
                       if (LineX1Pos > 33 && LineX1Pos > LineX0Pos)
                       {
                           LineX1Pos --;
                       }
                      else
                       {
                        delay;
                       }
                   }
                 else
                {
                       if (LineX1Pos < 736)
                       {
                           LineX1Pos ++;
                       }
                 else
                 {
                        delay;
                 }
            }
            RC10SevenSegWritePair (LineX1Pos, 0, 0);
      }
```




***Obrázek 24:*** *Vykreslené kurzory v ose X a Y*

Kurzory v ose X je možné na páté pozici v menu osciloskopu zobrazovat či nezobrazovat. Pohybem tlačítka nahoru nebo dolů kurzory vysvítím nebo nevysvítím na obrazovku. Kurzory v ose Y je možné vysvítit či nevysvítit na pozici v menu šest. Podmínka pro zobrazení či nezobrazení kurzoru je v hardware definována takto:

```
/*
 * Visible and invisible cursorsX1,X2
 */
if (Offset[2]= =20)
{
    if (RC10ButtonUpRead ( ) || RC10ButtonDownRead ( ))
    {
         if (RC10ButtonUpRead ( ))
         {
              visible_cursor_x = 1;
         }
         else
         {
              visible_cursor_x = 0;
         }
    }
    else
    {
           delay;
    }
}
else
      delay;
```

### 4.2.5. Vykreslení hexadecimální hodnoty kurzoru v digitální podobě

Do dualport RAM paměti pro kursory na první čtyři pozice v uživatelském menu vykresluji na tři pevně stanovená místa vedle sebe fonty z jedné dualport RAM paměti určené pro fonty. V této paměti jsou uloženy hexadecimální hodnoty od 1 až do F. Proměnná, která je při pohybu kurzorem inkrementována nebo dekrementována, je 10-bitová. Z této proměnné postupně beru dolní čtyři bity, prostřední čtyři bity a poslední dva bity a hodnotu těchto bitů vykresluji v hexadecimální soustavě na obrazovku. To znamená, že když jsou první čtyři bity samé jedničky, je na obrazovce vykresleno 00F. Tento proces vykreslování je u všech měřicích kurzorů. Pro snadnou představu zde uvádím část hardware napsaného v jazyce Handel-C, který testuje na 16. pozici v menu osciloskopu stisknutí tlačítka dolů nebo nahoru pro měřicí kurzor X1. Jakmile začne uživatel pohybovat daným kurzorem, vykreslí se hodnota, na jaké se kurzor nachází. Pro jednoduchost v tomto příkladě vykresluji pouze proměnnou obsahující dolní čtyři bity, a to na místě 0x21 hexa v menu kurzorů. Pro vykreslení prostředních čtyř bitů a horních dvou bitů stačí tento proces rozšířit a zvolit pouze jiné místo v menu kurzorů, aby nedošlo k vzájemnému překreslování.

```
/*
 * Up,down with cursorX1 and value cursorX1 in hexadecimal system
 */

if (Offset[2] = = 16)
{
    if (RC10ButtonUpRead ( ) || RC10ButtonDownRead ( ))
    {
            if (RC10ButtonUpRead ( ))
            {
              if (LineX0Pos > 63)
            {
                  LineX0Pos -- ;
            }
            else
            {
                  delay;
            }
     }
          else
          {
             if (LineX0Pos < 736 && LineX0Pos < LineX1Pos)
          {
                  LineX0Pos ++ ;
          }
          else
          {
            delay;
          }
     }
          RC10SevenSegWritePair (LineX0Pos, 0, 0);
    par
    {
```

```
F_Addr_For_Cursory    = 0 @ LineX0Pos[3:0] @ (unsigned 4) 0;
Rw_Addr_For_Cursory   = 0 @ Offset[2][2:0] @ (unsigned 7) 0x21;
f_cnt_for_cursory     = 0;
}

do
{
  MenuHigh_for_cursory = (Rw_Addr_For_Cursory[9:7]==Offset[2][2:0]);

  par
  {
        MENURAM_FOR_CURSORY.D[Rw_Addr_For_Cursory] =
        MenuHigh_for_cursory @        FontValue_for_cursory[16:0];

        F_Addr_For_Cursory++;
        Rw_Addr_For_Cursory++;
        f_cnt_for_cursory++;
        Done_Rw_Addr_For_Cursory_3 = (f_cnt_for_cursory == 15);
   }
  }
  while (~Done_Rw_Addr_For_Cursory_3);
```

### 4.2.6. Zobrazení a přepočet digitální hodnoty v milivoltech

Součástí nekonečné smyčky na obrázku č.23 je část, kde se do dvaceti bitové proměnné ukládá rozdíl kurzorů X2-X1 a násobí se konstantou pro přepočet volt na pixel . Poté se zavolá makro procedura „Transfer_Volt“. Pro jednoduchost uvedu pouze část tohoto makra, kde dochází k ukládání hodnoty do proměnné „desetitisice“. V hardware je toto makro definováno takto:

```
macro proc Transfer_Volt ( )
{
unsigned 20 data;
unsigned 4  pocet1, pocet10, pocet100,pocet1000, pocet10000;

      /*
       * Repeat until we get a zero
       */
      data = LineXPos_value;

      /*
       * Initialise variable
       */

    par
    {
      desetitisice      = 0;
      tisice            = 0;
      stovky            = 0;
      desitky           = 0;
      jednotky          = 0;
    }
```

```
do
{
        if (data > 9999)
      {
         par
         {
         pocet10000 = 1;
         data       = data - 10000;
         desetitisice++;
         }
      }
   else
   {
         pocet10000 = 0;
   }
} while (pocet10000);
```

Tato procedura vezme danou proměnnou a postupně ji dělí 10000, 1000, 100, 10, 1. Výstupem této procedury je pět čtyř bitových proměnných, které v sobě obsahují počet tisíců, stovek, desítek, jednotek a desetin. V paměti pro kursory je na daném místě vykreslována desetinná čárka a jednotky v milivoltech. V paměti fontů pro převod jsou vytvořeny fonty od nuly do devíti. Jeden znak má délku 16 bitů. Fonty z této paměti vykresluji na pěti přesně stanovených místech. V paměti fontů se pohybuji po 16 bitech a vykresluji na danou pozici hodnotu, která odpovídá počtu tisícům, stovek, desítek a desetin. Pro jednoduchost: když se vykresluje například počet jednotek pět, tak v paměti fontů se odkazuji na adresu 5x16, kde je vykreslena pětka. Z této pozice vezmu 16 bitů a vykreslím je v paměti menu pro kurzory. Tento proces běží neustále, proto když dojde ke změně polohy kurzorů, celý tento proces proběhne znovu a vykreslí se odpovídající hodnota v milivoltech. Pro příklad vykreslení hodnoty v milivoltech uvedu pouze část hardware, který se stará o vykreslení proměnné pro počet hodnot desetitisíců na předem stanovené pozici v menu pro kurzory. Zbylé vykreslování proměnných je navrženo na stejném principu.

```
/*
 * Transfer from hexadecimal system into decimal and depiction on the
   screen for cursors Y
 */

   LineYPos_transfer = 0 @ (LineY1Pos - LineY0Pos);
   LineYPos_value    = ((LineYPos_transfer * 10)/32);
   Transfer_Second  ();
   sirka_fontu = 16;
   Rw_Addr_For_Cursory = 896;


   F_Addr_For_Transfer = ((unsigned 5)0 @ sirka_fontu) * ((unsigned 6)0 @
   desetitisice);
   for (i=0; i < sirka_fontu; i++) {


     MENURAM_FOR_CURSORY.D[Rw_Addr_For_Cursory] = FontValue_for_transfer;
     F_Addr_For_Transfer++;
     Rw_Addr_For_Cursory++;
   }
```

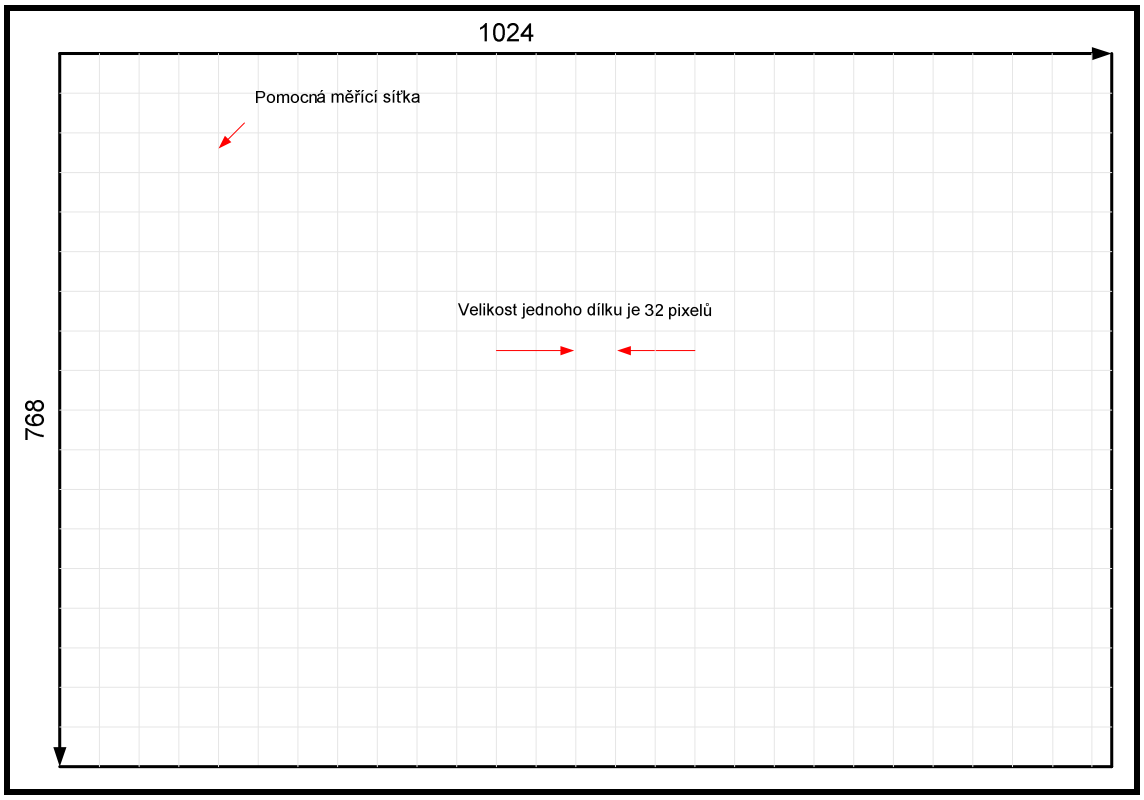
### 4.2.7. Zobrazení a přepočet digitální hodnoty v dílkách

Tento proces vykreslování funguje na stejném principu jako u zobrazování rozdílu hodnoty kurzorů X v milivoltech. S tím rozdílem, že 20-bitovou proměnnou násobím konstantu, která říká, kolik dílků jsou od sebe vzdáleny kurzory Y. Pro vykreslování fontů (číselných hodnot) používám stejnou paměť fontů jako v předchozím případě. Asi každý by se zeptal, proč když je to digitální osciloskop, nevykresluji hodnotu sekundách. Jelikož lze měnit časovou základnu vstupních signálů, nebylo by možné posunovat při změně časové základny desetinnou čárku při přepočtu na sekundy. Je to proto, že desetinnou čárku např. pro hodnotu kurzorů X vykresluji na pevně stanovenou pozici v menu pro kurzory. Z tohoto důvodu jsem se rozhodl vykreslovat hodnotu rozdílu kurzorů Y v dílcích. Uživatel tedy pro získání periody musí vzít počet dílků a vynásobit je danou časovou základnou. Hardware určený pro vykreslování hodnoty rozdílu kurzorů v ose Y zde nebudu uvádět, jelikož je stejný jako u vykreslování rozdílu kurzorů v ose X, který byl popsán v předchozí kapitole. Rozdíl je pouze v konstantě pro převod na dílky a ve změně místa, kde se bude počet dílků vykreslovat.

### 4.2.8. Vykreslení pomocné měřicí mřížky

Pro lepší odečítání velikosti amplitudy a periody signálu je na monitoru vykreslena pomocná měřicí mřížka (obrázek č.25). Tuto mřížku je možno vypnout a zapnout na šesté pozici v menu pro nastavení parametrů prvního signálu. Vypnutí a zapnutí je možné provést pomocí mini-joysticku stiskem nahoru a dolu. Velikost jednoho dílku je 32 pixlů. K vykreslování dochází v makro proceduře „DisplayData“. V této proceduře vytvářím hardware pro vykreslení pomocné měřicí mřížky takto:

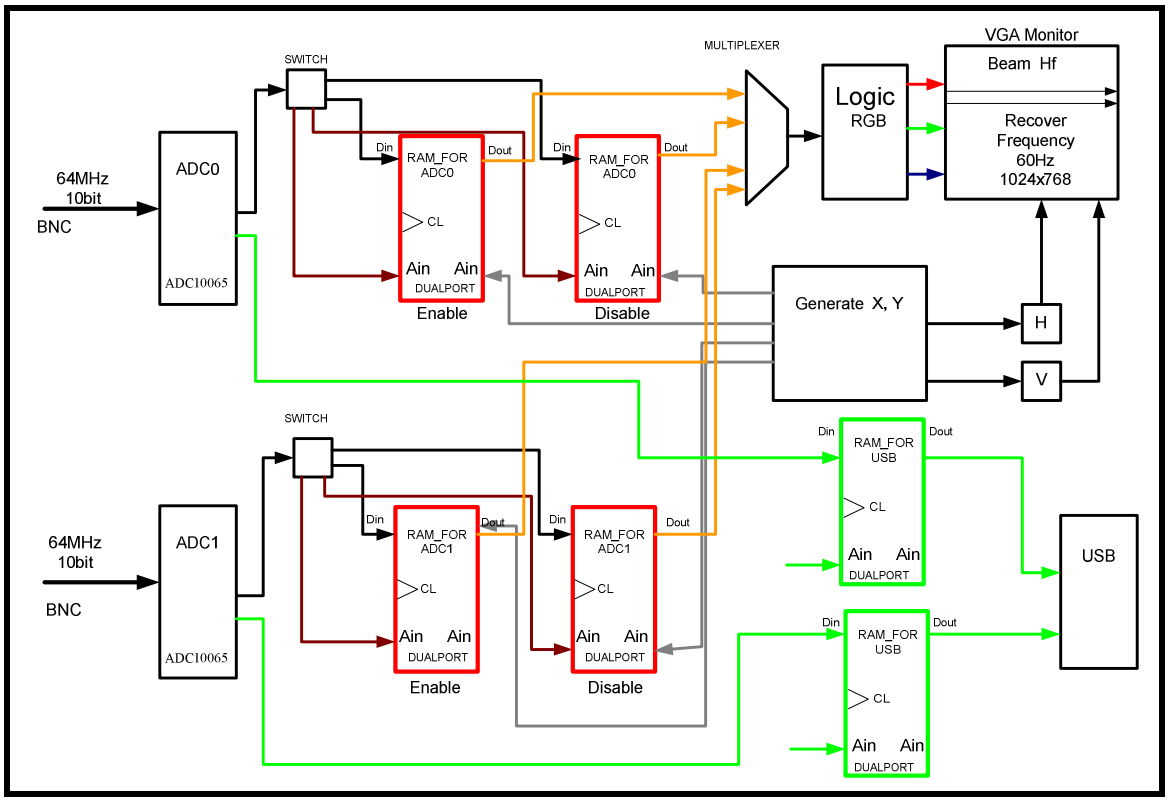
```
Grid[0]  = (((X & 0x1f) == 0) || (X == 1023) ||
            ((Y & 0x1f) == 0) || (Y == 767));
```




***Obrázek 25:*** *Pomocná měřicí mřížka*

### 4.2.9. Přenos dat do PC s vykreslením průběhu v programu matlab

Při přenosu dat z ADC převodníku přes rozhraní USB do programu Matlab nastala otázka, jak vykreslovat hodnoty z dualport RAM pamětí určených pro vstupní signály (obrázek č.17). Bylo by velmi obtížné přenášet data do programu Matlab ze dvou pamětí určených pro ukládání navzorkovaných dat z ADC převodníku, kde dochází k přepínání mezi jednotlivými paměťmi a jejich vykreslování na obrazovku. Rozšířil jsem hardware ještě o dvě paměti pro vzorky z ADC převodníků. To znamená, že vzorky z ADC převodníků ukládám do tří pamětí pro první signál a do tří pamětí pro druhý signál. (Obrázek č.26) je podobný obrázku č.17 s tím rozdílem, že je rozšířen o dvě paměti, které jsou používané pro přenos dat do programu Matlab přes rozhraní USB.




***Obrázek 26:*** *Rozšíření hardware o další dvě RAM paměti pro ukládání dat z ADC a vykreslení v programu Matlab*

V hardware je proces pro posílání dat z RAM pamětí na rozhraní USB zapsán pomocí makro procedury „USB_Transfer“. V tomto makru se obsah obou pamětí ukládá do 10-bitové proměnné, která je posléze rozdělena na osm dolních bitů a zbylé dva horní bity. Tyto proměnné jsou pak zapisovány na rozhraní USB.

```
macro proc USB_transfer(RAMPtr1,RAMPtr2)
{
static unsigned 10 usb_data_canal1;
static unsigned 10 usb_ram_adresa=0;
unsigned first_data1;
unsigned second_data1;

  while (usb_transfer_run)
   {
        if (switc_ADC1_or_ADC2)
        {
              usb_data_canal1 = RAMPtr1[2].R[usb_ram_adresa];
        }
        else
        {
              usb_data_canal1 = RAMPtr2[2].R[usb_ram_adresa];
        }
        first_data1  = usb_data_canal1[7:0];
        second_data1 = (unsigned 6)0 @ usb_data_canal1[9:8];

        RC10USBWrite (first_data1);
        RC10USBWrite (second_data1);
```

```
        if (usb_ram_adresa==1023)
        {
                usb_ram_adresa=0;
        }
        else
        {
                usb_ram_adresa++;
        }
    }
}
```

Pro navázání spojení s programem Matlab a rozhraním USB používám knihovnu, která se nahraje do programu Matlab. Tuto knihovnu vytvořil ing. Roman Bartosiński. Po nahrání knihovny do Matlabu se v simulinku objeví položka „Celoxica RC Boards“, kde je blok RC10. Po vložení modelu RC10 je třeba ho správně nastavit, aby spolu mohly obě strany komunikovat. Komunikace je blokující na obou stranách, ale na straně Matlabu je zablokovaní komunikace ošetřeno timeoutem. Blok v simulinku muže mít maximálně jeden vstupní port různé šířky a maximálně jeden výstupní port různé šířky. Podporované typy dat jsou UINT8, UINT16 a UINT32, přičemž vstup a výstup je nezávislý. Já pro svůj přenos využívám datový typ UINT16, jelikož přenáším dvě 8-bitové proměnné. Vykreslení vstupního signálu v simulinku programu Matlab je pomocí bloku „Scope“. V tomto bloku vykresluji pouze první signál. Pro přepnutí na vykreslení druhého signálu je v uživatelském menu pro kanál jedna na sedmé pozici vytvořena pozice „USB SWITCH“, kde si uživatel může přepínat pomocí tlačítka nahoru a dolů, jaký signál se mu bude do Matlabu vykreslovat. Při spuštění bloku v simulinku pro navázání komunikace je do RC10 Pilot nahrán soubor *.bit, v mém případě vytvořený osciloskop. Cesta k tomuto souboru je nastavena v bloku RC10.

# 5. Ověření funkčnosti

Mobilní paměťový osciloskop jsem po dokončení otestoval, jestli splňuje všechny požadavky. Testování provedla i nezávislá osoba. Při testování byli ověřeny tyto požadavky na funkci osciloskopu:

- pohyb v uživatelském menu
- nastavování parametrů vstupních signálů
- zapnutí a vypnutí měřicích kurzorů
- měření a zobrazení amplitudy
- měření a zobrazení počtu dílků
- přenesení vstupních signálů pomocí rozhraní USB do PC a jejich zobrazení v programu Matlab
- vypnutí a zapnutí měřicí mřížky
- hardware náročnost

Po otestování těchto funkcí osciloskopu, bylo zjištěno že osciloskop má tyto funkce plně funkční. Bylo zjištěno, že pro uživatele není moc dobře řešeno zobrazování vstupního signálu v programu Matlab s ohledem na popisování os X a Y.

# 6. Výstupní parametry osciloskopu

Výstupní parametry po překladu programu se uloží do souboru „build.log“, který je umístěn spolu v adresáři, do kterého je ukládán soubor *.bit. Pro přehlednost bych zde uvedl jenom parametry ze souboru, které jsou ve výsledku nejdůležitější. Celý tento soubor je součástí přílohy.

První dva parametry vypisují počet chyb a varování při vytváření hardware.

**Design Summary:**

| | |
|---|---|
| Number of errors: | 0 |
| Number of warnings: | 21 |

Zbylé parametry specifikují procentuální počet logických obvodů, využívaných na čipu. Je zde udaný počet používaných RAM pamětí. Pro tento hardware jich je využito 14 z 32 možných. Počet používaných násobiček je 5 z 32 možných.

**Logic Utilization:**

| | | | |
|---|---|---|---|
| Number of Slice Flip Flops: | 1,245 out of | 26,624 | 4% |
| Number of 4 input LUTs: | 2,132 out of | 26,624 | 8% |
| Number of Block RAMs: | 14 out of | 32 | 43% |
| Number of MULT18X18s: | 5 out of | 32 | 15% |

# 7. Závěr

Výsledkem této bakalářské práce je fungující mobilní paměťový osciloskop. Ten umožňuje vzorkovat vstupní signál v určitých intervalech s maximální rychlostí 64 MHz a přenést jej pomocí rozhraní USB do PC. V počítači je vstupní signál zobrazen v programu Matlab. Uživatel osciloskopu může nastavovat parametry vstupních signálů v menu osciloskopu. Dále může využívat pomocné měřicí kurzory pro měření amplitudy a periody signálu. Velikosti rozdílu kurzorů jsou vykreslovány v digitální podobě v dolním pravém rohu obrazovky. V rámci této bakalářské práce byl napsán hardware pro programovatelný logický obvod Spartan 3 XC3S1500L-4 v jazyce „Handel-C“. Při práci na této bakalářské práci jsem získal zkušenosti:

- s programovatelným logickým obvodem Spartan 3 XC3S1500L-4
- s jazykem Handel-C a jeho strukturou
- s programovým prostředím DK41 a PDK41 firmy Celoxica UK
- s návrhovým prostředím ISE7.1 firmy Xilinx
- s programem Microsoft Visio 2003 pro kreslení blokových schémat

Při vytváření hardware v podobě mobilního programovatelného osciloskopu jsem se snažil, aby počet dualport RAM pamětí nepřekročil více než 16. Toto se mi podařilo a proto je možno použít tento osciloskop na levnější a dostupnější desce s logickým programovatelným obvodem Spartan XC3S400, který má pouze 16 dualport RAM pamětí na rozdíl od desky RC10 Pilot, která jich má 32. Závěrem bych chtěl ještě říci, že tento směr v technice je značně obsáhlý, ale pro mě velice zajímavý.

# 8. Literatura


[1] MATTHEW AUBURZY, *RC 10 Platform Development Manual, Celoxica, UK, 2005*
[2] ROGER GOOK, *PDK Manual, Celoxica, UK, 2005*
[3] HAASZ, V., SEDLÁČEK, M.: *Elektrická měření, ČVUT 2000*, ISBN 80-7232-220-6
[4] HEROUT, P.: *Učebnice jazyka C, 4. vydání, Kopp*, 2004, ISBN 80-7232-220-6
[5] *RC10 Pilot* http://www.celoxica.com/products/rc10/default.asp
[6] *Programovací jazyk „C“* http://www.hw.cz/programovani/ucebnice_c/
[7] *ADC10065* http://www.national.com/pf/DC/ADC10065.html
[8] *Programovatelné součástky* http://www.hw.cz/Teorie-a-praxe/
[9] *FPGA* http://www.sweb.cz/omyl1/procesor.htm
[10] *FPGA* http://www.odbornecasopisy.cz/automa/2005/au050546.htm
[11] *Monitor CRT* http://technet.idnes.cz/hw_monitory.asp
[12] *Paměti* http://www.zsprazska.oknet.cz/navody/arch_PC/TEXTY/INTPAM.HTML
[13] *Spartan-3/3L* http://www.xilinx.com/

# 9. Obsah přiloženého CD

Obsah kořenového adresáře přiloženého CD:

| | |
|---|---|
| /log | Adresář, ve kterém jsou uloženy informace po překladu programu. |
| /bit | Soubor, určený pro nahrání do desky RC10 Pilot |
| /doc | Adresář, který obsahuje technickou zprávu na téma mobilní paměťový osciloskop |

# 10. Přílohy

1. Technická dokumentace k desce RC10 Pilot
2. Vypsaný soubor build.log