**První cvičení – ADC**

1. Sestavte program, který bude pomocí A/D převodníku měřit napětí na některém pinu. Převedené číslo si zobrazte na nějakém portu, na LED si dejte nejvyšší bity převedeného čísla a předveďte, že převodník funguje. Vzorkovací kmitočet zvolte 8 kHz.

až sem za 3

1. Sestavte program, který změří napětí na Band Gap diodě v procesoru .

Až sem za 2

1. Ukažte, že napětí na Band Gap diodě je stále stejné. K tomu účelu napájejte procesor napětím 4.5V 5V 5.5V

až sem za 1

Ad1: To je základní úloha. Nastavíte A/D převodník, dále si uděláme přerušení od čítače s kmitočtem 8kHz. V tomto přerušení vždy „odstartujeme“ převod. Dále uděláme přerušení od A/D převodníku, v něm přečteme ADCBUF a hodnotu uložíme na vhodný port. Na správné bity dáme LED a díváme se, co převodník dělá. Vstupní proměnné napětí získejte např. pomocí potenciometru. Pokud někdo řekne, že neví, jak se to dělá ( měnitelné napětí pomocí potenciometru ), tak rovnou jděte k panu řediteli a odhlaste se ze školy.

Ad2: A/D převodník máme z prvního příkladu. Prostě nastavíme ten vstup, na kterém je Band Gap dioda . Referenční napětí pro převodník je ….. V, to odpovídá číslu …… , takže napětí na BandGap snadno spočítáte.

Ad3: Měníme napájecí napětí, to znamená, že měníme referenční napětí pro převodník. Z LED si odečtete číslo, které převodník dodává, a vypočítáte napětí na BandGApDiodě. Mělo by být stále stejné. Pro měnitelné napětí využijete napájecí vstup na šroubečkové svorkovnici, **odpojíte PicKit2** Desku budete napájet z vhodného měnitelného zdroje. Vezměte si voltmetr**. Pozor ! Nepřekračujte napětí 5.5V. Při napětích nad 6V procesor spolehlivě zničíte.**

**Druhé cvičení – DAC**

1. Sestavte program, který vytvoří na jednom výstupu DAC pilu o kmitočtu asi tak 100 Hz. Napětí bude stoupat od 0 do maxima, pak opět klesne na 0, a tak stále dokolečka. Vzorkovací kmitočet udělejte 8 kHz.

až sem za 3

1. Sestavte program, který na jednom výstupu DAC bude vysílat sinusovku o kmitočtu 80 Hz. Vzorky sinusovky generujte pomocí DSP algoritmu.

Až sem za 2

1. Sestavte program, který bude na jednom výstupu DAC vysílat vámi vymyšlený fantastický průběh. Jeho vzorky si uložte do pole .

až sem za 1

Ad 1: DA převodník jsme probírali loni, vše potřebné víte a máte naprogramováno v Assembleru. Musíte zprovoznit SPI, to znamená také pomocí PPS nastavit správně piny procesoru. PPS umíme z OX, tam jste ho potřebovali na UART. Pokud ne, tak studujte ! <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/progr_a_vysv/PPS/> <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/progr_a_vysv/SPI/>

U SPI nezapomeneme zamaskovat vysílané číslo tak, abychom nastavili správně řídící registry převodníku MCP4822 . Jeho datasheet je na <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/asm_progr_a_vysv/SPI/MCP4822.pdf>

Nejlépe je si celé kompletní vysílání udělat jako podprogram, který dá CS do 0 , zamaskuje vysílané číslo, přečte SPIBUF, vyšle číslo. V přerušení od SPI pak vracíme CS na hodnotu 1 . Název přerušení máte ve vzorovém programu na SPI .

Pokud hodláte brečet, že je toho moc, tak připomínám, že všechno jsme dělali ve druhém ročníku, akorát pomocí jiného jazyka. Jste studenti, tak studujte.

A potom prostě v přerušení od čítače vyrábíme pilu, to znamená, že čísla stoupají od …. Do …. Po …. , a v každém zavolání přerušení se zvětší. Potřebujete si do dalšího zavolání přerušení pamatovat hodnotu té pily. To znamená, že proměnná pro pilu musí být lokální/globální (nehodící se deletujte).

Doufám, že už vám také došlo, že přerušení je funkce. Pokud ne, tak to tady píšu: přerušení je funkce.

Ad 2 : sinusovku pomocí DSP jsme dělali letos v prvním pololetí, program máte, takže uděláte CntrlC+CntrlV . Umíme udělat jakoukoli amplitudu. Náš DAC převodník vyžaduje na vstupu čísla v rozsahu od … do ….. . To znamená, že amplitudu naší sinusovky musíme udělat poloviční a celou sinusovku stejnosměrně posunout o ………... V programu nezapomeňte na přetypování, předpokládám, že váš program na výpočet sinusovky bude pracovat s typy …… , ale převodník si žádá typ ………….. . Opět doporučuji udělat si výpočet dalšího vzorku sinusovky v přerušení od čítače a následně zavolat podprogram pro vyslání čísla do převodníku , abyste tak zajistili správné časování. Je nutno, aby si program pamatoval hodnoty proměnných do dalšího zavolání přerušení, ta znamená, že proměnné, které počítají sinusovku, musí být lokální/globální (nehodící se deletujte) .

Nebo – a to by bylo ještě lepší – si celý výpočet vzorku sinusovky udělejte jako funkci. A ta funkce na každé zavolání dodá další hodnotu vzorku sinusovky. Výhodou je, že pak můžete tuhle funkci přenést do dalších programů.

Ad 3: Uděláme si nějaký fantastický průběh, bude mít třeba 200 vzorků, rozvášníte se a uděláte nějakou hezkou věc v Excelu – třeba nějak takhle:

Hodnoty vzorků si uložíte do pole. To pole bude samozřejmě uloženo v paměti EEPROM, takže použijeme mechanismus PSV. To se udělá tak, že pole definujete pomocí direktivy const

const unsigned int pole[MAXPOLE+1] ;

A Cčko umí do toho pole rovnou nacpat hodnoty, které si v něm přejeme mít

const unsigned int pole[MAXPOLE+1] = { 2, 8, 14, 93, 1, 6 } ;

Tohle samozřejmě uloží čísla jenom do prvních šesti prvků pole, ostatní prvky pole budou náhodné hodnoty.

Protože odřádkování a mezery jsou „bílé znaky“ , tedy se nepřekládají, může to vypadat i takto:

const unsigned int pole[MAXPOLE+1] = { // tady je složená závorka

2,

3,

8,

26,

69,

1548,

45,

638,

123

) ; // tohle je složená závorka a středník

Tohle je užitečné při vkládání hodnot z Excelu nebo nějakého jiného programu, který vám je spočítal.

Za poslední } je středník, nezapomeňte . A za posledním číslem není čárka. Čárka slouží jako oddělovač čísel.

Krásné, že ?

Pokud by vám vadilo, že hned na začátku program je takováhle šílenost se spoustou čísel, můžete si pole udělat do samostatného file a includnout ho.

Čísla, která jsem já napsal, jsou náhodná, jen pro inspiraci. Vy si je udělajte tak, abyste využili maximální roszah převodníku.

**Třetí cvičení – „drát z PICa“**

1. Sestavte program, který udělá „drát z PICa“ . To znamená, že naprogramujete na vstupu AD převodník, na výstupu DA převodník a předvedete, že napětí, které pustíme do A/D převodníku, se objeví na D/A převodníku. Dále předveďte aliasing. Vzorkovací kmitočet udělejte 8 kHz.

až sem za 3

1. Sestavte program, který bude fungovat jako derivační člen. Jeho funkci předveďte na osciloskopu. Vzorkovací kmitočet zvolte 8 kHz.

Až sem za 2

1. Sestavte program, který bude fungovat jako analogová zpožďovací linka s co největším zpožděním. Vzorkovací kmitočet ponechte 8kHz. Funkci předveďte na osciloskopu. až sem až sem za 1

Ad1: Konečně jsme se dostali k legendárnímu drátu z PICa !

Znamená to spojit první a druhé cvičení, jinak je to hračka. Prostě hodnotu, kterou přečetl AD převodník, vyšleme na DA převodník. AD převodník nastavte nejlépe do formátu „unsigned integer“, abyste nemuseli přečtené číslo upravovat.

Do AD převodníku budete pravděpodobně pouštět střídavé napětí z „normálního“ generátoru. To je ale kladné i záporné. Náš převodník převádí napětí v rozsahu 0V – 5V . Znamená to, že napětí z generátoru musíme posunout. K tomu máte na desce obvod C9-R2-R3 nebo C10-R4-R5 , který slouží k posunutí stejnosměrné složky. Ostatně jsme o tom mluvili na teoretické hodině.

Na osciloskopu tedy zapojíte dva kanály, na jeden vstupní signál a na druhý výstupní, a ukážete, že „drát z PICa“ funguje.

Dále vše upravíte tak, aby vzniknul aliasing, a předvedete ho. Aliasing jste měli s Ing. Novotným, takže není co řešit. Bez předvedení aliasingu nebude tento bod uznán.

Ad 2: Derivační člen je hračka, podívejte se do sešitu z teorie. Nezapomeňte, že na výstupu budou kladná i záporná čísla ( pokud vstupní napětí stoupá, pak kladná, pokud vstupná napětí klesá, záporná). Do D/A převodníku ovšem pouštíte pouze kladná čísla ! Musíte tedy výstup z derivačního členu před odesláním do D/A převodníku stejnosměrně posunout, to znamená, že s čísly uděláte ...... Výstupní čísla vám možná vyjdou moc velká nebo moc malá . V tom případě je vynásobte vhodným číslem tak, abyste dostali použitelný výsledek.

Na osciloskopu předvedete, že obvod derivuje. Vy budete vědět, co má obvod dělat, a předvedete to. Nachystáte si amplitudovou i fázovou charakteristiku derivačního členu a budete předvádět, co se v obvodu děje. Bez předvedení funkce nebude tento bod uznán.

Ad 3: opět hračka. Uděláme si pole o maximální možné velikosti. Z pole přečteme číslo a vyšleme do D/A. Následně do stejného místa uložíme číslo, které jsme přečetli z A/D. Inkrementujeme index pole. Až index dojde na konec, nastavíme ho na začátek pole, tedy do něj uložíme číslo ........

Předpokládám, že mnohým se bude zdát divné, že máme nejdříve přečíst číslo z pole a potom do stejného místa jiné uložit. Jak si jistě pamatujete, program se píše odprostředka. To znamená, že tohle všechno už dávno běží a běží, takže v poli máte nějaká čísla z minulých cyklů. Na začátku programu jste samozřejmě do pole uložili nějaká vhodná čísla, takže i v prvním průchodu D/A něco vysílá. Pokud jste tak neučinili, jsou v poli čísla ......... , takže převodník v prvním průchodu cyklem vysílá ..................... . To vadí/nevadí (nehodící se deletujte)

Ideální varianta programu vypadá takhle:

1. Přečíst číslo z pole
2. Odstartovat vysílání přečteného čísla do DA převodníku
3. Přečíst číslo z ADCBUF a uložit do pole
4. Odstartovat převod AD převodníku
5. Inkrementovat ukazovátko pole

A tohle všechno se udělá 8000 krát za sekundu, je to napsáno v přerušení od čítače

Možná ve vás ještě trochu vzbuzují otázku body 3 a 4 . Opět si uvědomíme, že program už dávno běží. To znamená, že v bodu 3 čteme číslo z minulého převodu. U AD převodníku nepoužíváme přerušení. Vše jsme naprogramovali tak, aby převodník stihnul převést napětí dříve, než přijde další přerušení od čítače.

**Čtvrté cvičení**

1. Sestavte program, který přepne kmitočet vnitřního oscilátoru na 140 MH nebo trochu méně. Přepnutí vhodným způsobem dokažte. Dále s tímto kmitočtem realizujte „drát z PICa“

až sem za 4

1. Sestavte program, který bude fungovat jako pásmová propusť. Kmitočty si zvolte libovolně, ale tak, aby propouštěné pásmo bylo někde v ¼ pásma, které je schopen náš PIC zpracovat, a propouštěné pásmo bylo co možná nejužší. Použijte filtr FIR, tak, aby měl řekněme 20-25 koeficientů b, více ne . Vzorkovací kmitočet ponechte 8 kHz. Funkci filtru předveďte na osciloskopu. Dále do filtru pusťte obdélníkový signál s první harmonickou v propustném pásmu filtru a předveďte signál na výstupu. Zhodnoťte, zda se obvod chová správně.

**!!!!! Filtr budete mít navržený již při příchodu do laboratoře a dostanete za něj další známku. Každý žák bude mít svůj filtr FIR, i když pak dvojice realizuje jenom jeden.**

až sem až sem za 1

ad 1: je to hračka, nastavíme si PLL a je to. Dále si vyvedeme vydělený kmitočet na vhodný pin procesoru ( použijeme vhodný čítač ). No a pokud vám bude výstupní kmitočet souhlasit s výpočty, je asi vše v pořádku. Pro „drát z PICa“ musíte přepočítat hodnoty děliček u hodin pro SPI a ADC, abyste zachovali přibližně stejné kmitočty jako ve třetím cvičení. Pokud to neuděláte, drát z PICa vám fakt opravdu fungovat nemůže.

Použití PLL známe z druhého ročníku. Podobně jako u PPS existuje i tady Cčkový nástroj, jak snadno lock- a unlock- nout příslušná registrz. Vše je popsáno v <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/progr_a_vysv/oscilator/>

<http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/progr_a_vysv/oscilator/oscilator.doc> , na straně 3 . K tomu samozřejmě zdrojové file \*.c v tomto adresáři .

ad 2:

navrhneme si filtr FIR. Můžete zkusit např. naši oblíbenou <http://t-filter.engineerjs.com/> . S návrhem si musíte trochu pohrát. 20 koeficientů jsem zvolil proto, aby procesor byl schopen „upočítat“ další hodnotu v čase mezi jednotlivými vzorky. Nemůžete udělat příliš strmé boky té propusti. Až budete mít sestavený program pro výpočet dalšího vzorku , podívejte se v simulátoru, jak to dlouho trvá. Nastavte si kmitočet 140MHz. Při 20-ti koeficientech b by čas na výpočet měl být menší než čas mezi dvěma vzorky. Pokud to opravdu nepůjde, snižte vzorkovací kmitočet a ve stejném poměru snižte všechny kmitočty u filtru

Dále doporučuji zvolit si nějaký výstupní pin, který pro nic nepoužíváte. Ten nastavíte před začátkem výpočtu filtru do 1 a po skončení výpočtu filtru do 0. Tak budete moci změřit, jak dlouho opravdu filtr počítáte. Pro úvodní pokusy si nastavte hodně malý vzorkovací kmitočet, abyste mohli délku výpočtu snadno změřit.

Program udělejte nejlépe následovně:

čítačem generujte vzorkovací kmitočet

v přerušení od čítače přečtěte hodnotu z AD převodníku

odstartujte další AD převod

minulou spočtenou hodnotu z výstupu filtru FIR vyšlete na D/A převodník

vypočtěte další hodnotu výstupního vzorku pro filtr FIR

tím dosáhneme toho, že AD převod, vysílání a počítání běží paralelně. Dále běží vysílání na DA převodník, zase „samo“ , nezávisle na výpočtech. Na konci vysílání do DA převodníku vracíme v přerušení CS zpátky do 1. Je možné, že se tohle bude dít až v době, kdy přijde další přerušení od čítače. **Přerušení od SPI proto dejte vyšší prioritu než přerušení od čítače.**

Dále nezapomeňte na to, že musíte přepočítat hodiny pro AD převodník a hodiny pro SPI, protože pracujete s kmitočtem 140 MHz na procesoru.

Doporučuji nastavit AD převodník do režimu „signed integer“ , dále se podívejte na amplitudovou charakteristiku vaší propusti – je možné, že filtr bude zesilovat. V tom případě vstupní čísla vhodně zmenšete. Na výstupu nezapomeňte čísla upravit tak, aby byla kladná, stejně jako u derivačního členu, výsledky výpočtů budou kladné i záporné.

Při předvádění pustíte do bastldesky z generátoru nějaký kmitočet a podíváme se, co vychází na výstupu. Podíváme se na charakteristiku filtru a zhodnotíme, zda to tak má nebo nemá být. To uděláme pro více kmitočtů, abychom tak zdokumentovali celou přenosovou charakteristiku našeho filtru FIR .

**Z domova budete mít připravené koeficienty b filtru FIR a amplitudovou charakteristiku filtru. Obojí předvedete na začátku hodiny.**

Pole koeficientů b filtru FIR si okopírujete z návrhového programu. Nezapomeňte na direktivu const při definici pole, aby se pole uložilo do paměti EEPROM. Typ pro výslednou sumu doporučuji float, pak to číslo samozřejmě převedeme na integer a stejnosměrně posuneme, stejně jako jsme to dělali u derivačního členu.

Na vaší bastldesce <http://ozeas.sdb.cz/panska/mikroproc/33EV32/bastldeska/33EV32_TRTUSEK_2019/cad/Schema.pdf>

máte obvod R4 R5 C10 a obvod R2 R3 C9 , který lze použít pro posuv vstupního střídavého napětí .

**Páté cvičení – DSP engine**

1. Sestavte filtr FIR, který bude realizovat pásmovou propust nebo zádrž s alespoň třemi pásmy ( tedy propust bude mít tři pásma , ve kterých propouští, a čtyři pásma, ve kterých nepropouští , zádrž analogicky ). Naprogramujte do PICa pomocí DSP-engine. Činnost filtru předveďte na osciloskopu, vzorkovací kmitočet je 8 kHz.

**Stejně jako ve čtvrtém cvičení budete mít filtr navržen již při příchodu do učebny a dostanete z něj další známku.**

až sem za 1

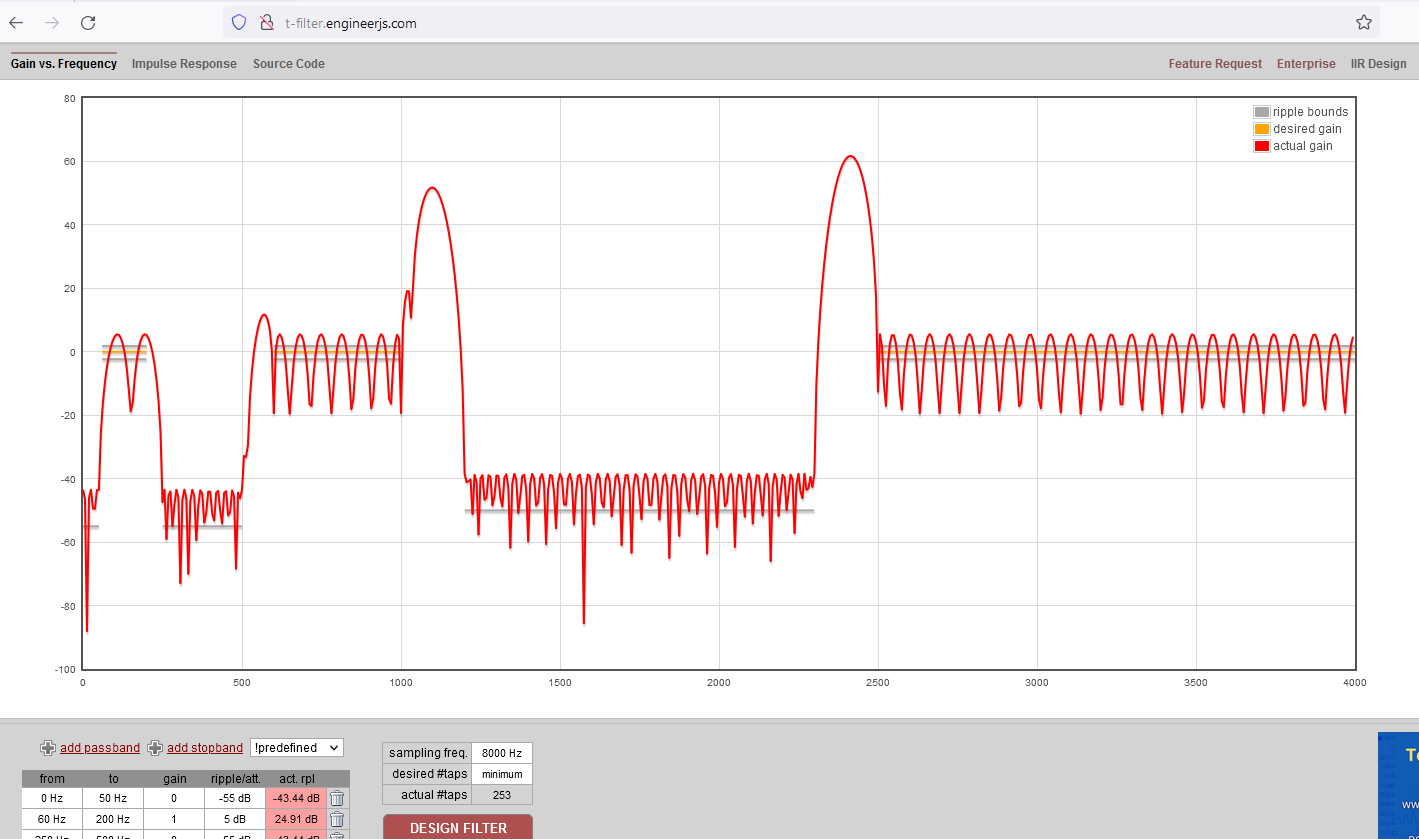
Poznámky: AD převodník nastavíme tak, aby dodával čísla signed fractional. Pro vlastní výpočet použijeme postup podle <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/progr_a_vysv/fixed_point_simplex/fixed_point_simplex.docx> . A dál už není co řešit !

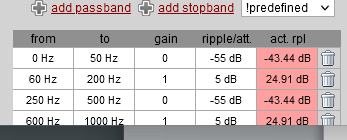
Při provádění výpočtů vás asi nebude omezovat čas, samozřejmě si přetaktujeme procesor na 140 MHz. Omezení bude z hlediska paměti, navrhněte si takový filtr, aby se vám koeficienty vešly do paměti.

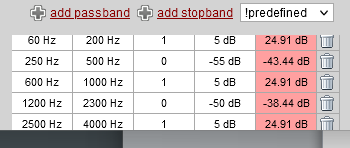
Já jsem navrhnul tohle, vy si samozřejmě uděláte něco jiného. Vidíme, že filtr na některých kmitočtech zesiluje, ten náš návrhový nástroj není příliš dokonalý. Musíte si pohrát se strmostí sestupných a vzestupných hran filtru, když je uděláte moc strmé, návrh filtru se zhroutí.

**Pokud máte někoho známého na FELu nebo obdobné škole, poproste ho, ať vám filtr navrhne v programu MATLAB, ten se na to krásně hodí a funguje úplně dokonale.**

Nezapomeňte, že náš návrhový nástroj má osu kmitočtů lineární, vy si to udělejte logaritmicky, tedy u nižších kmitočtů jsou hrany blíže k sobě, u vyšších dál, ostatně můj obrázek je toho dokladem.









**Šesté cvičení – DSP engine a cyklický buffer**

1. Páté cvičení upravte tak, abyste použili místo „posouvátek“ cyklický buffer.

Známka: 1

**Sedmé cvičení – Goertzelův algoritmus**

1. Sestavte program, který bude sloužit jako detektor vámi zvoleného kmitočtu. Zařízení má na vstupu napětí sinusového průběhu o nějakém kmitočtu. Dále je na procesor připojena LED. LED svítí, pokud na vstupu je určený kmitočet.

Známka: 1

Poznámky: Vzorkovací kmitočet ponechte 8kHz, jak jsme zvyklí. Dále si zvolte kmitočet, který chcete detekovat – navrhuji třeba 100 – 500 Hz, pro vyšší kmitočty se už příliš blížíme k vzorkovacímu, a to by asi pro první pokusy nebylo vhodné. Dále si zvolíte čas, po který budete vstupní signál zkoumat, zkuste třeba 0.5 sec ? Z toho pak vyjde potřebný počet vzorků - N, které je nutno spočítat.

V přerušení od čítače čteme hodnoty z AD převodníku a počítáme a počítáme. Pro výpočty potřebujete pouze:

Vstupní vzorek x

Výstupní vzorek y

Výstupní vzorek zpožděný yzp

Výstupní vzorek dvakrát zpožděný y2zp

Pokud sem patláte nějaké pole, je to opravdu špatně.

Přicházejí vstupní vzorky, počítáte dokolečka ten algoritmus. Až přijde vzorek [N-1], uložíte si někam výstupní vzorek. To je samozřejmě vzorek y[N-1]. Až přijde vzorek [N], uložíte si někam jinam výstupní vzorek. To je samozřejmě vzorek y[N]. Dále nastavíte nějakou proměnnou, podle které se pozná, že vám přišlo N vzorků. Jo, doufám, že vám došlo, že musíte přicházející vzorky počítat. V tomto okamžiku vynulujete y, yzp, y2zp a začínáte úplně od nuly znovu hledat sinusovku.

Následující akci doporučuji provést v hlavním programu, protože trvá dost dlouho. V přerušení od čítače se mezitím počítá další rozpoznávání sinusovky.

Pro hlavní program máte uloženo y[N-1] a y[N]. V nějaké proměnné máte nastaveno, že už přišlo N vzorků. Tím říkáte hlavnímu programu „počítej!“ . Spočtete to šílené číslo a podle jeho velikosti rozsvítíte nebo nerozsvítíte diodu. Protože tady už musíme dost experimentovat, doporučuji udělat několik rozhodovacích úrovní pro několik LED.

Nějak takhle:

Když je číslo větší než …. , svítí dioda ….. , jinak zhasne

Když je číslo větší než …. , svítí dioda ….. , jinak zhasne

Když je číslo větší než …. , svítí dioda ….. , jinak zhasne

Když je číslo větší než …. , svítí dioda ….. , jinak zhasne

Když je číslo větší než …. , svítí dioda ….. , jinak zhasne

Nezapomeňte, že ty LED musíte také pozhasínat.

Úrovně doporučuji udělat exponenciálně , abyste pokryli velký rozsah spočítaných čísel. Ostatně jsem to dělali na Ozeášovi, dejte si do svého programu čísla, která odpovídají tomuto zadání ( tomuto == “to, co právě teď děláte”) a podívejte se, co vám vychází .

A v hlavním programu musíte také shodit tu proměnnou, která říkala, že přišlo N vzorků .