VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Mikroprocesorové a vestavěné systémy – projekt

Měření vzdálenosti ultrazvukovým senzorem

17. prosince 2021 Ondřej Ondryáš (xondry02)

Popis projektu

Cílem projektu byla implementace vestavné aplikace pro mikrokontrolér z řady Kinetis K60, která pomocí ultrazvukového senzoru HY-SRF05 měří vzdálenost mezi senzorem a nejbližším objektem a zobrazuje ji na sedmisegmentovém LED displeji.

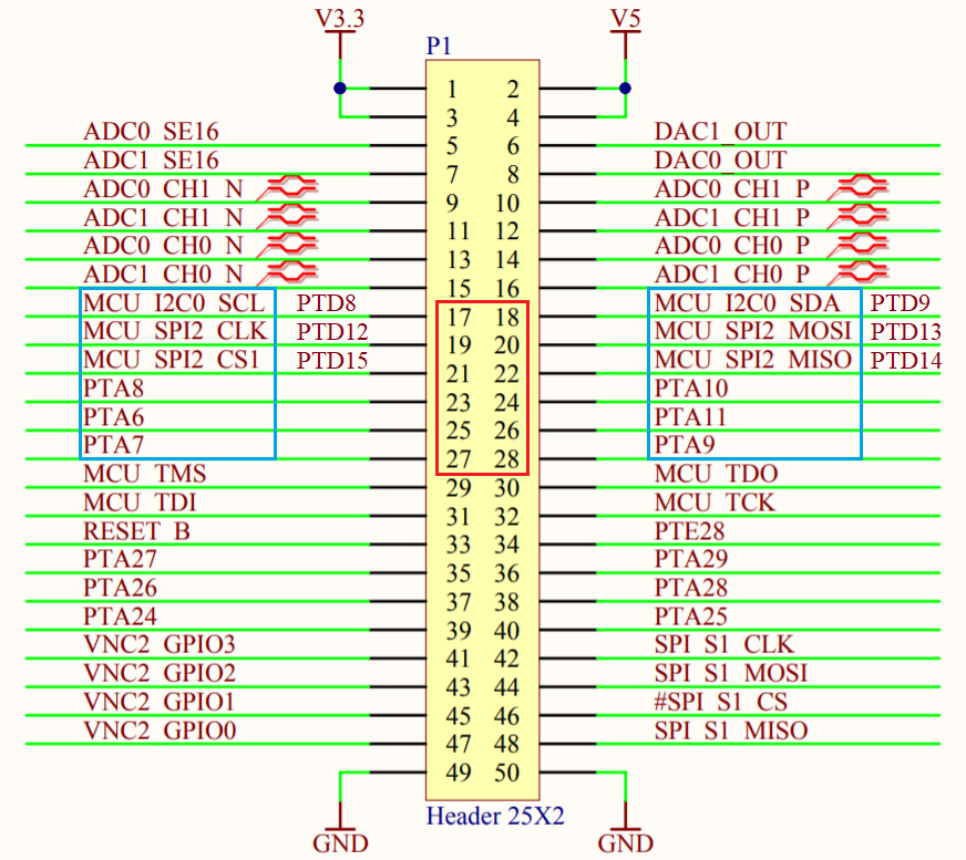
Aplikace byla realizována na výukovém kitu Minerva (FITkit 3), v rámci experimentování se senzorem vzdálenosti jsem obdobnou aplikaci implementoval i pro platformu Arduino Uno.

Video s ukázkou funkce aplikace a s vysvětlením struktury zdrojového kódu je k dispozici zde: `https://www.youtube.com/watch?v=jJUQ7xVLiK8`.

Zapojení

Displejový modul obsahuje čtyři číslice, které se skládají z osmi samostatně řiditelných segmentů (sedm segmentů číslice a desetinná tečka). Každý segment jedné číslice je tvořen jednou diodou LED. Na piny modulu jsou vyvedeny paralelně spojené anody diod jednotlivých číslic (celkem 8 vývodů) a paralelně spojené katody segmentů v každé jedné číslici (celkem 4 vývody). Zobrazení jedné číslice je tedy provedeno vytvořením napěťového rozdílu mezi vývody anod požadovaných segmentů a jednom vývodu odpovídajícím katodám v požadované číslici (přivedením log. 0). Zobrazování více číslic „najednou“ je realizováno rychlým přepínáním jedné aktivní číslice.

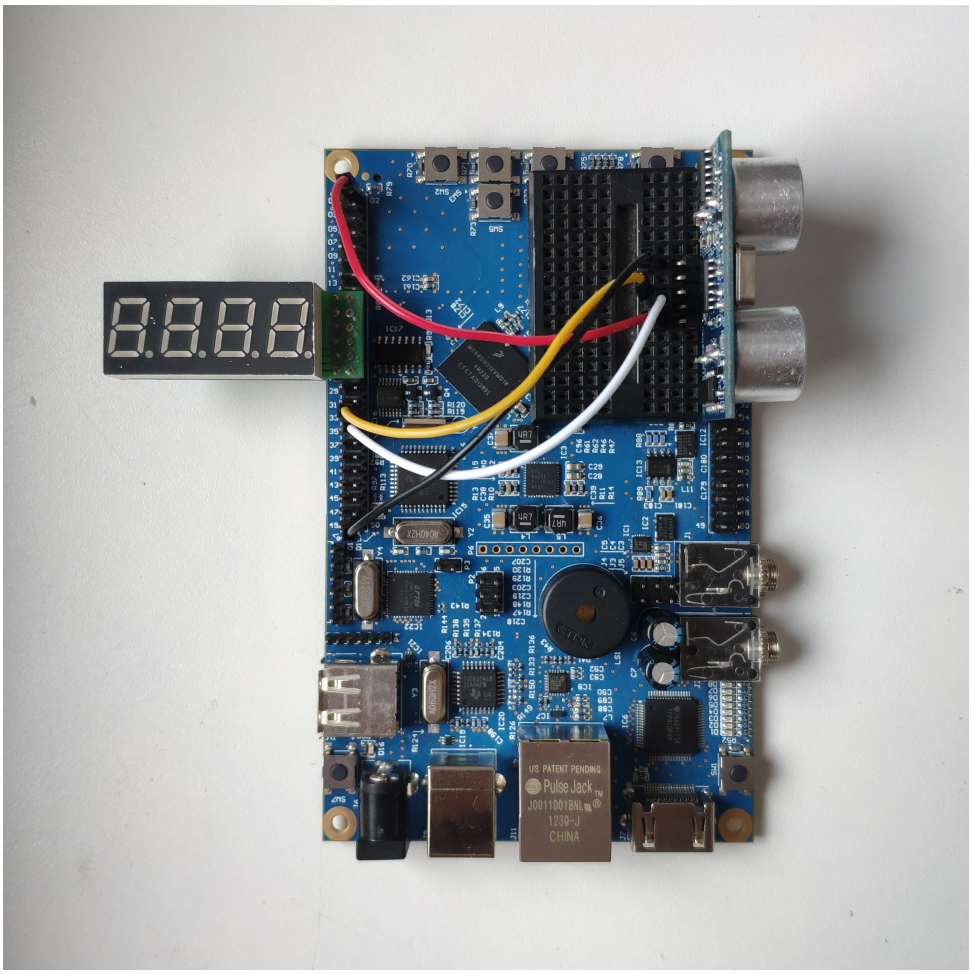
Zapojení displejového modulu vyžaduje celkem 12 GPIO pinů. Displej je zapojen přímo do propojovacího konektoru P1, jak je naznačeno na obr. 1.



Obrázek 1: Zapojení displejového modulu do konektoru P1 na desce kitu Minerva.

Senzor HY-SRF05 využívá kromě 5V napájení a země (použit pin 2 a 50) dva datové signály, jeden vstupní (*trigger*) a jeden výstupní (*echo*). Signál *trigger* je přiveden na pin 36 (PTA29), signál *echo* je přiveden na pin 34 (PTE28).

Celkové zapojení je ukázáno na obrázku 2.



Obrázek 2: Fotografie zapojení displeje a senzoru do desky kitu Minerva.

Způsob řešení

Při implementaci bylo nutné řešit řízení segmentového displeje a obsluhu senzoru. Řízení displeje obnáší dostatečně rychle přepínat mezi číslicemi (v jednom momentu může být aktivní pouze jedna číslice – logická nula může být pouze na jedné ze čtyř sdružených katod). Obsluha senzoru pak spočívá v periodickém generování signálu *trigger*, zachycení pulzu na signálu *echo*, výpočet jeho délky (v čase) a přepočet této délky na vzdálenost. Při implementaci jsem využil různé časovače (FTM, LPTMR) a přerušení při detekci hran.

Implementaci jsem provedl s využitím IDE MCUXpresso. Použil jsem *bare-metal* přístup k čipu, ze speciálních funkcí SDK, resp. MCUXpresso Config Tools, jsem využil pouze nastavování hodinových signálů, to ale spíše v rámci experimentování, jejich použití není nutné.

Řízení displeje

Pro řízení displeje jsem použil časovač mikrokontroléru `LPTMR0` (*low-power timer*). Pro využití tohoto konkrétního časovače není žádný význačný důvod[1], chtěl jsem si především vyzkoušet jeho použití. Užitečné však je, že zdrojem hodinového signálu jsou 1kHz hodiny, což je celkem vhodná frekvence pro obnovování displeje.

V mé implementaci je časovač inkrementován s každým tikem těchto 1kHz hodin a jeho registr *compare* je nastaven na 3. V každém čtvrtém tiku, tedy s frekvencí 250 Hz, je vygenerováno přerušení, které zavolá

[1] tento časovač se hodí zejména pro úsporu energie v aplikacích, kde může být většina čipu po delší dobu vypnuta, přičemž tento časovač stále běží při minimální spotřebě

funkci `HandleDisplay`. Ta nastaví výstupy příslušných GPIO pinů takovým způsobem, který rozsvítí vhodné segmenty aktuální číslice a inkrementuje index aktuální číslice. Každá číslice je tedy obnovována s frekvencí cca 62,5 Hz, což plně dostačuje k tomu, aby lidské oko nevnímalo blikání jednotlivých číslic na displeji.

Obsluha displeje je implementována v modulu `display.c`. Funkce `SetDisplayData` nastavuje aktuálně zobrazované číslo, které je uloženo jako vnitřní stav tohoto modulu. Při nastavování čísla jsou dočasně vypnuta přerušení z LPTMR, aby nedocházelo k nekonzistentnímu zobrazování.

Obsluha senzoru

Komunikace se senzorem HY-SRF05 je vcelku jednoduchá: Na jeho pin *trigger* je přiveden puls o šířce min. 10 µs. Senzor provede měření a zvýší logickou úroveň na svém pinu *echo* na dobu, která odpovídá naměřené vzdálenosti. Pro zjištění vzdálenosti je tedy nutné reagovat na nástupné a sestupné hrany signálu *echo* a počítat jejich délku.

Obsluha senzoru je implementována v modulu `sensor.c`. Odesílání pulzu je řešeno aktivním čekáním. Po dobu odesílání pulzu je deaktivováno přerušení z LPTMR, aby nedocházelo k problémům s časováním. Příjem pulzu je poté řešen nastaveným přerušením na pinu, na který je přiveden signál *echo*. Toto přerušení je vyvoláno na náběžné i sestupné hraně. V rámci obsluhy přerušení je na nástupné hraně uložena aktuální hodnota počítadla FTM0 (viz dále), na sestupné hraně je poté vypočten rozdíl mezi aktuální hodnotou počítadla FTM0 a dříve uloženou. Tento rozdíl odpovídá délce přijatého pulzu. Ta je poté (už v rámci hlavní smyčky programu) převedena na celé centimetry převedena pomocí vztahu:

$$d = value \cdot \frac{34\,000}{1\,310\,720} \cdot \frac{1}{2}$$

kde *value* je rozdíl hodnot počítadla, 34 000 je rychlost zvuku v cm/s a 1 310 720 je hodnota, o kterou je počítadlo zvýšeno během 1 sekundy (viz dále). Výsledek je vydělen dvěma, protože význam hodnoty je čas od vyslání ultrazvukového signálu senzorem do jeho zaznamenání, zvuk musí urazit vzdálenost k předmětu a pak zpět.

Počítání času pulzu

Pro určení času mezi náběžnou a sestupnou hranou přijímaného signálu ze senzoru je využit čítač/časovač `FTM0` (*FlexTimer*). Ten jen nakonfigurován velmi jednoduše – jako zdroj hodinového signálu je použit hlavní systémový hodinový signál (*system clock*), který je nastaven na hodnotu 20,971 52 MHz. FTM má nastaveno dělení 16, hodnota jeho čítače se tedy mění s frekvencí 1,310 72 MHz. Z toho vyplývá, že je možné rozeznat pulzy o minimální délce $\frac{1}{1{,}310\,72}$ µs, což je pro potřeby použitého senzoru zcela dostačující[2].

Zhodnocení řešení

Implementovaná aplikace plní svůj záměr – měří vzdálenost a zobrazuje ji na displeji. Implementace je uspokojivě spolehlivá.

Použitý senzor nepovažuji za velmi přesný. Zejména při delších vzdálenostech začíná vracet poněkud nahodilé, rozkmitané hodnoty. Nejsem si jistý, čím je toto chování způsobeno. Abych (alespoň částečně) vyloučil chybu ve svém programu, napsal jsem také primitivní implementaci této aplikace pro platformu Arduino, jejíž chování odpovídá chování mé implementace pro MCU Kinetis.

Jako významnější nedostatek vidím použití aktivního čekání při generování *trigger* pulzů: aktivní čekání zbytečně vytěžuje procesor a spotřebovává energii, navíc je značně náchylné na způsob, jakým je kód čekání přeložen. (Na druhou stranu, v tomto případě není délka *trigger* pulzu příliš důležitá, musí být pouze delší než 10 µs.) Pro generování pulzů by bylo vhodnější použít časovač.

[2] jako minimální hodnota vzdálenosti, kterou senzor dokáže změřit, jsou uvedeny 2 cm, což odpovídá délce pulzu cca 154 cyklů čítače

Způsob počítání délky pulzu není je vyhovující, ovšem existují lepší řešení. Mým původním plánem bylo pro počítání délky pulzu použít přímo *FlexTimer*, který je možné nastavit do režimu, ve kterém na jednom ze svých kanálů sám zaznamenává hrany vstupního signálu a spouští přerušení. V tomto případě není vůbec nutné pro počítání délky pulzu využívat modul GPIO. Toto řešení jsem nepoužil, protože na konektoru P1 nebyl po zapojení displeje volný žádný pin, který by se dal napojit na kanál FTM. Použité řešení tedy využívá přímo čtení hodnoty z hlavního čítače modulu `FTM0`, v případě potřeby využít tento modul i jiným způsobem by muselo dojít k výraznějším modifikacím kódu. Dochází zde také k problémům s přetečením počítadla, které je nutné speciálně ošetřovat.

Zajímavé by také mohlo být pro samotnou obsluhu displeje využít hradlové pole, které se na desce výukového kitu také nachází, a komunikovat s ním z mikrokontroléru pomocí vhodné sběrnice, např. SPI.