API NXP Cup ARC Board

Autor:

- Lica Robert-Mihai [mailto:licarobert21@gmail.com]

Introducere

Un API pentru a folosi placa ARC cat mai usor. La aceasta se pot connecta 2 encodere, 2 servouri, 2 motoare DC si 2 camere Pixy2. API-ul creat ofera control asupra servourilor; control direct asupra %dc motoarelor sau se poate specifica un rpm daca se doteaza cu encodere prin folosirea unui PID; API-ul pentru camera Pixy2 a fost portat pe acest uC.

Descriere generală

Diagrama software:

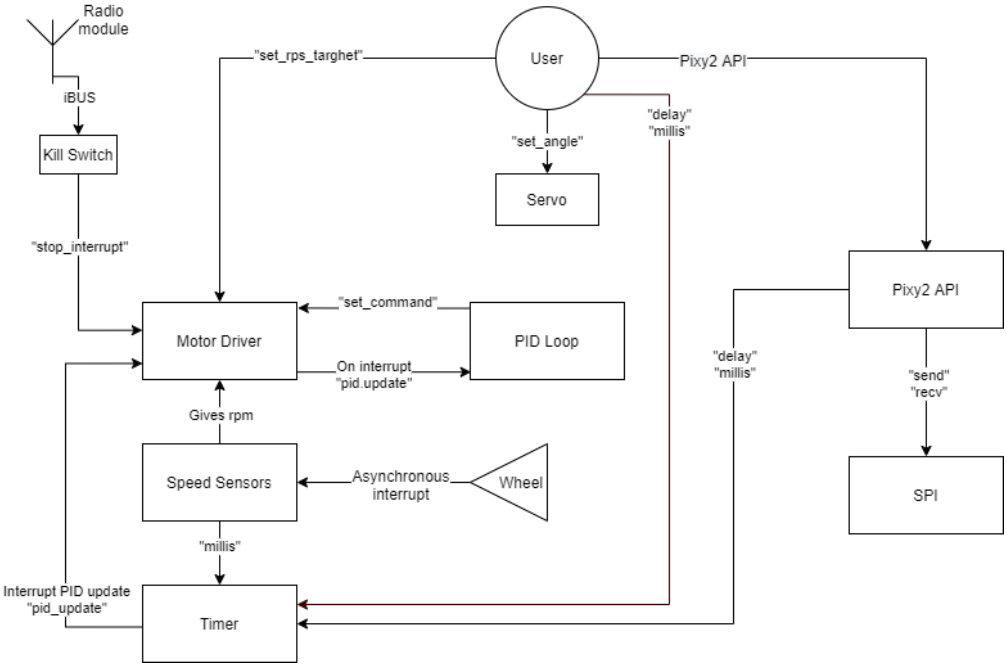
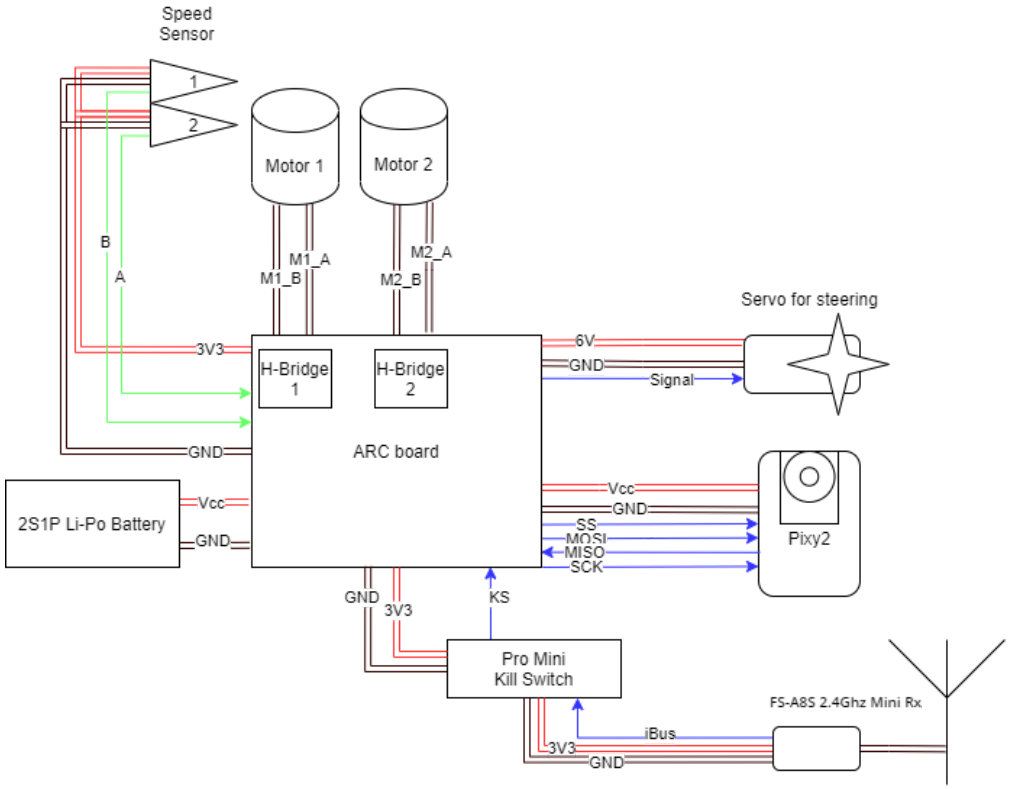


Diagrama hardware:



Hardware Design

Componente:

1. 1 ARC Board;
2. 2 motoare DC;
3. 4 senzori hall;
4. 1 servo;
5. 1 Pixy2;
6. 1 Arduino Pro Mini;
7. 1 receiver FS-A8S 2.4Ghz Mini.

1. Placa ARC a fost construita de Haute Ecole ARC [https://www.he-arc.ch] pentru competitia NXP Cup [https://community.nxp.com/t5/The-NXP-Cup-EMEA/gh-p/15351]. Microcontroller-ul folosit este un ARM Cortex-M4 MK64FN1M0VLL12 (datasheet). Schematica placii si utilizarile initiale ale pinilor pot fi gasite aici: ARC-Board schematic.
2. Pe placa ARC se afla 2 punti H, prin urmare putem connecta pana la 2 motoare cu comenzi diferite.
3. Se folosesc cate 2 senzori/roata, fiecare roata are 9 magneti atasati, prin urmare rezolutia este de 40 de grade. Cei 2 senzori de pe o roata functioneaza in cuadratura pentru a obtine si directia de rotatie.
4. Un servo montat in fata masinutei controleaza directia rotilor.
5. Pentru contextul nostru, camera video poate sa recunoasca liniile de pe podea sub forma de vectori si sa intoarca pozitiile acestora in matricea de vizualizare a camerei. Mai multe detalii se pot gasi pe wiki-ul camerei: Pixy2-wiki [https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:overview].
6. Arduino-ul este folosit pentru a decoda semnalul iBUS venit de la receiver. Am optat pentru un periferic pentru a nu incarca uC-ul principal si deoarece am gasit o biblioteca

[https://github.com/utkudarilmaz/FlySkyiBus] care decodeaza acest protocol. Din pacate nu am gasit o biblioteca de Attiny pentru acest protocol.

7. Un receiver de dimensiuni mici pe 2.4Ghz compatibil cu radio telecomenzile FlySky. Este folosit pentru a da o comanda de kill masinii daca este pe cale sa se loveasca de ceva.

Software Design

<timer.hpp>

Notes:

Biblioteca foloseste timmer-ul FTM1.

Metode:

static Timer& get_instance(void)
Returneaza o referinta catre obiect si la prima apelare este initializat obiectul.

```
Timer& instance = Timer::get_instance();
```

uint32_t micros(void)
Returneaza un unsigned int ce reprezinta timpul in microsecunde de la primul apel Timer::get_instance().

```
auto elapsed = instance.micros();
```

uint32_t millis(void)
Returneaza un unsigned int ce reprezinta timpul in milisecunde de la primul apel Timer::get_instance().

```
auto elapsed = instance.millis();
```

void delayMicroseconds(uint32_t t)
Functie blocanta ce asteapta 't' microsecunde.

```
instance.delayMicroseconds(500000U); // 0.5 sec
```

void delay(uint32_t t)
Functie blocanta ce asteapta 't' milisecunde.

```
instance.delay(1000); // 1.0 sec
```

uint32_t add_irq_handler(void (*time_IRQn_handler)(void))
Returneaza 0 pentru success si -1 in caz de eroare.
Primeste un pointer la o functie, ce intoarce void si accepta void ca parametru, si o adauga in vectorul “irq_handlers” ce vor fi apelate la intreruperea de ceas (0.2 sec).

```
instance.add_irq_handler(function);
```

void call_irq_handlers(void)
Metoda apelata de rutina de tratare a intreruperii de ceas (0.2 sec) pentru a executa toate functiile din vectorul “irq_handlers”

```
instance.call_irq_handlers();
```

<motor_driver.hpp>

Notes:

Biblioteca foloseste timmer-ul FTM0 si canalele 0, 1, 2, 3.

Metode:

static Motors& get_instance(void)
Returneaza o referinta catre obiect si la prima apelare este initializat obiectul.

```
Motors& instance = Motors::get_instance();
```

static void give_config(struct motor_config_t& conf)
Pentru viitor: mod avansat de a configura clasa inaintea primului apel catre Motors::get_instance(); momentan este folosita configuratia default.

```
Motors::give_config(custome_config);
```

void enable_motor_driver(void)
Activeaza puntile H ale placii. Pentru detalii consultati schemele placii si a puntii-H [https://www.nxp.com/docs/en/data-sheet/MC33887.pdf]

```
instance.enable_motor_driver();
```

void disable_motor_driver(void)
Dezactiveaza puntile H ale placii. Pentru detalii consultati schemele placii si a puntii-H [https://www.nxp.com/docs/en/data-sheet/MC33887.pdf]

```
instance.disable_motor_driver();
```

void enable_motor_outputs(void)
Activeaza output-urile puntilor-H. Pentru detalii consultati schemele placii si a puntii-H [https://www.nxp.com/docs/en/data-sheet/MC33887.pdf]

```
instance.enable_motor_output();
```

void disable_motor_outputs(void)
Dezactiveaza output-urile puntilor-H. Pentru detalii consultati schemele placii si a puntii-H [https://www.nxp.com/docs/en/data-sheet/MC33887.pdf].

```
instance.disable_motor_output();
```

void update_motor_status(void)
Configureaza puntile-H conform starii interne a obiectului. Aceasta metota nu ar trebuii sa fie utilizata decat daca pinii uC-ului au fost setati nu prin API-ul current si a aparut o discrepanta intre configurare si starea interna a obiectului.

```
instance.update_motor_status();
```

void set_rps_target(const float rps)
Seteaza un nr. de rotatii pe secunda (rps) dorit si bucla PID interioara se ocupa de atingerea tintei daca aceasta este activata (default activa).

```
instance.set_rps_target(15.5f);
```

void get_rps_target(void)
Returneaza nr. de rotatii pe secunda dorit, folosit de bucla PID interioara se ocupa de atingerea tintei daca aceasta este activata (default activa).

```
instance.set_rps_target(15.5f);
```

void enable_pid_control(const int8_t new_command = 0)
Daca bucla PID era dezactivata, este activat controlul prin PID, rps_targhet-ul este setat la 0, comanda la motoare este setata ca "new_command" (default 0).

```
instance.enable_pid_control();
```

void disable_pid_control(void)
Daca bucla PID era activata, este dezactivat controlul prin PID, rps_targhet-ul este setat la 0, comanda la motoare este setata la 0.

```
instance.disable_pid_control();
```

static void pid_update(void)
Functie apelata de intreruperea de ceas pentru a itera prin bucla PID de control a motoarelor daca aceasta este activata. Functia este lasata publica in caz ca user-ul doreste o iterare mai frecventa.

```
Motors::pid_update();
```

void set_command(int8_t command)
Seteaza comanda (-100 ⇐ command ⇐ 100) ce reprezinta duty cycle-ul semnalului pentru motoare. Pentru valori negative polaritatea motoarelor este inversata automat. Este folosit de bucla PID pentru a da o comanda motoarelor, dar poate fi folosita si de utilizator. Daca se seteaza o comanda in timp ce bucla PID este activa, la urmatoarea intrerupere de ceas aceasta va fi suprascrisa de rezultatul buclei PID.

```
instance.set_command(-50);
```

int8_t get_command(void)
Ar trebuii sa returneze ultima comanda data puntilor-H. Daca bucla PID este activa, returneaza ultima commanda calculata de aceasta, iar daca este inactiva returneaza ultima commanda data de utilizator manual explicit sau prin diferite functii (exemplu "enable/disable_pid_control").

```
auto command = instance.get_command();
```

void stop_interrupt(const bool state)
Pentru "state = true", opreste de urgenta puntile-H la declansarea unei intreruperi pe pinul de Signal Kill. Altfel activeaza la loc puntile-H daca acestea erau activate inaintea semnalului de kill.

```
instance.stop_interrupt(true);
```

<speed_sensor.hpp>

Notes:

Acronimul rps inseamna “rotatii pe secunda”
Biblioteca ofera suport pentru pana la 2 senzori de viteza a cate 2 canale fiecare.

Metode:

static SpS& get_instance()
Returneaza o referinta catre obiect si la prima apelare este initializat obiectul.

```
SpS& instance = SpS::get_instance();
```

float get_rps_a(void)
Returneaza nr. de rps a senzorului de viteza A daca acesta exista. Daca nu, intoarce viteza senzorului B sau 0 daca nici acesta nu exista. Functia este folosita la interarea buclei PID.

```
auto speed_a = instance.get_rps_a();
```

float get_rps_b(void)
Returneaza nr. de rps a senzorului de viteza B daca acesta exista. Daca nu, intoarce viteza senzorului A sau 0 daca nici acesta nu exista. Functia este folosita la interarea buclei PID.

```
auto speed_b = instance.get_rps_b();
```

void sensor_interrupt(void)
Functia este apelata de rutina de tratare a intreruperii pentru a verifica daca aceasta a venit pe pinii senzorilor de viteza. Este calculat noul nr. de rps per motor si se afla directia de rotatie prin cuadratura.

```
sensor_interrupt();
```

<kill_switch.hpp>

Notes:

Este folosit pinul PTA13 care deregula este folosit pentru unul din senzorii hall.
Prezenta unui modul radio pe masina este interzisa in cadrul competitiei NXP Cup. Acesta poate fi activat sau dezactivat usor doar schimband in fisierul “kill_switch.hpp”

```
#define KILL_SWITCH_ENABLE 1 // to enable the kill_switch
// #define KILL_SWITCH_ENABLE 0 // to disable the kill_switch
```

Functii:

void kill_switch_init(void)
Configureaza pinul si activeaza intreruperea pe ambele fronturi de semnal.

```
kill_switch_init();
```

void kill_switch_interrupt(void) Daca acest pin a declansat intreruperea, puntile-H vor fi dezactivate sau activate in functie de starea actualal a pinului. Aceasta functie este folosita in rutina de tratare a intreruperii.

```
kill_switch_interrupt();
```

<servo.hpp>

Notes:

Biblioteca foloseste timmer-ul FTM2 si canalele 0, 1.

Metode:

static Servo& get_instance(void)
Returneaza o referinta catre obiect si la prima apelare este initializat obiectul.

```
Servo& instance = Timer::get_instance();
```

void set_angle(ftm_chnl_t sv, uint16_t angle)
Primeste ca si parametru channel-ul (SV1_FTM_CHANNEL sau SV2_FTM_CHANNEL) la care este conectat servo-motorul si unghiul dorit (0 ⇐ angle ⇐ 180).

```
set_angle(SV1_FTM_CHANNEL , 90U);
```

void set_dutycycle(ftm_chnl_t sv, uint8_t dc)
Primeste ca si parametru channel-ul (SV1_FTM_CHANNEL sau SV2_FTM_CHANNEL) la care este conectat servo-motorul si duty cycle-ul dorit (0 ⇐ dc ⇐ 100) sa fie scris pe pinul de semnal.

```
set_dutycycle(SV1_FTM_CHANNEL , 2U);
```

<Pixy2.h>

Notes:

Pentru mai multe detalii, consultati documentatia oficiala de pe pagina [https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:porting_guide].
Pentru portare a trebuit sa implementez comunicatia pe SPI blocanta si functiile de “delay” si “millis” din <Timer.hpp>.

<SPI.h>

Notes:

Configuratia de SPI este realizata in configuratorul din MCUXpresso astfel [https://duckduckgo.com] (TO_DO: adauga imagine cu configuratia).
Biblioteca de Pixy2 se asteapta ca comunicatia de SPI sa fie blocanta. Pentru viitor comunicatia sa nu mai fie blocanta si sa se foloseasca intreruperi sau eDMA.

Metode:

static void begin(void)
Pentru a fi folosit pe viitor cand configurarea nu mai este facuta din GUI-ul din MCUXpresso.

```
SPI::begin();
```

static void send(uint8_t *buf, uint8_t len)
Trimite prin SPI un vector de bytes fara a intoarce ce citeste pe SPI. Comunicatia este blocanta.

```
SPI::send(buffer, n);
```

static void recv(uint8_t *buf, uint8_t len)
Primeste prin SPI un vector de bytes in timp ce trimite pe SPI \0. Comunicatia este blocanta.

```
SPI::recv(buffer, n);
```

Rezultate Obținute

TODO

Concluzii

TODO

Download

TODO

Jurnal

TODO

Bibliografie/Resurse

TODO