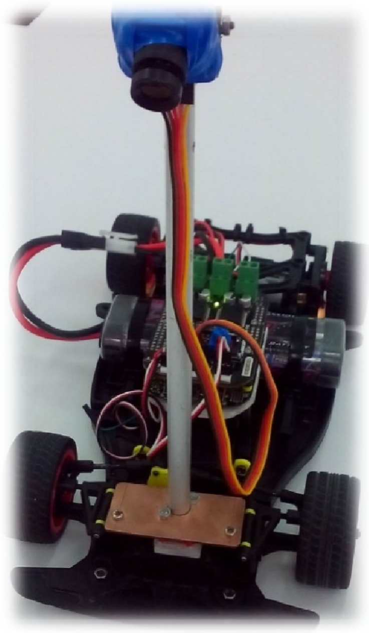
Romania
University of Craiova



Tehnical Report

The Freescale Cup
Intelligent Car Racing

“FasTech_UCV” team:

Teodor Nichiteanu

Horatiu Roibu

Laurentiu Păstae

Team Coordinator:

Prof. Cosmin Ionete

-2015 -

Table of content

1. Description of mechanical design of car model

1.1 Mechanical components and tools

The main goal was to achieve an intelligent car, able to follow a white track bordered by two black lines in the shortest time. To build the car we started from Freescale TFC kit and several other components.:

- **Car Chassis: Rugged 1/18th scale car - includes frame and tires**

- **2 x Standard Motor "rn 260 c" winding 18130**

- **Steering servo Futaba S3010**

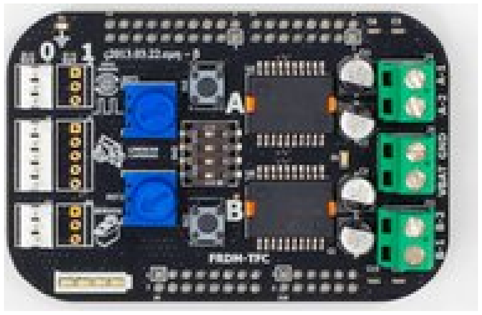
- **Line Scan Camera-**Texas Advanced Optical Solutions (TAOS) Linear Scan Sensor and **Freescale Light sensor**

- **2 x Digital IR Line Sensor - QRE1113**

- **Small components as: nuts, screws and plastic parts**

- **Freescale Freedom development platform -** FRDM-KL25Z

- **FRDM-TFC Freescale Cup Shield**



- **7.2V 2400mAh Ni-MH Battery Pack -** UNI Plug byVenom Group International (VNR1531)

- **Venom Sport Charger 4 to 8 Cell AC NiMH Battery Charger**

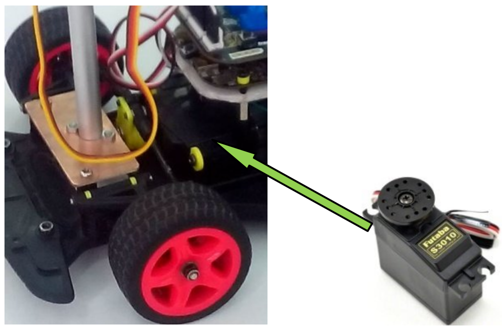
For this chapter we used information from:

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TFC-KIT

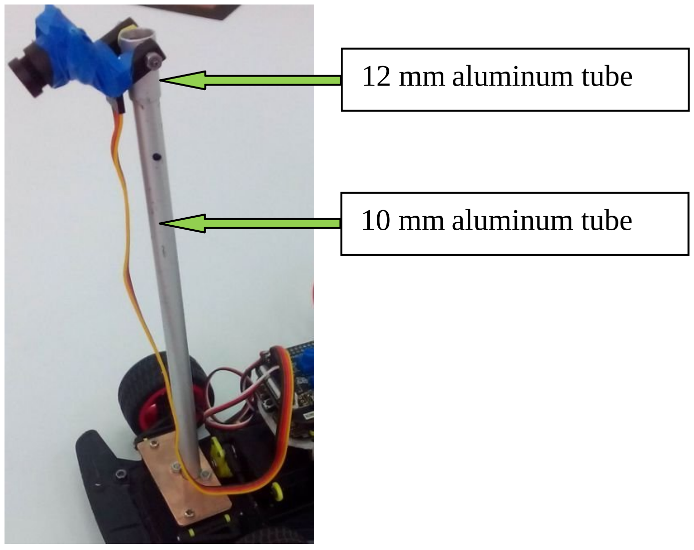
1.2 Mechanical architecture

Achieving mechanical architecture of the car began by assembling component parts of the Freescale kit and adding other components. Next we detail the assembly of several major components:

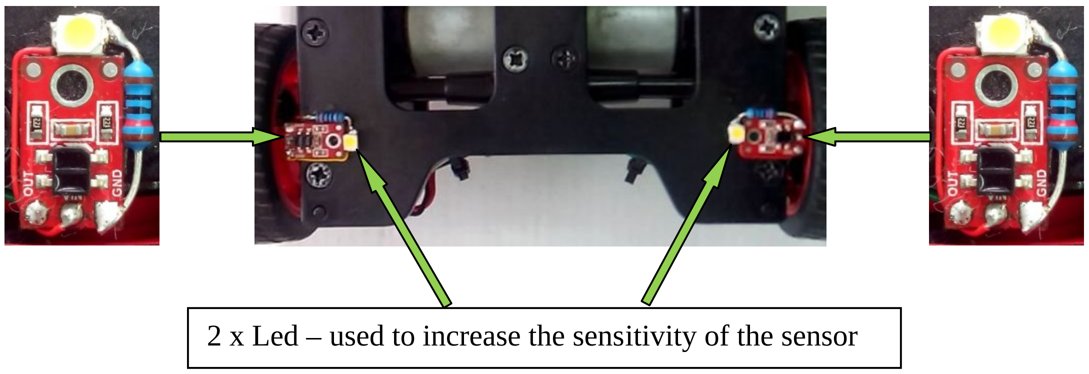
- ✓ the steering hardware implementation:



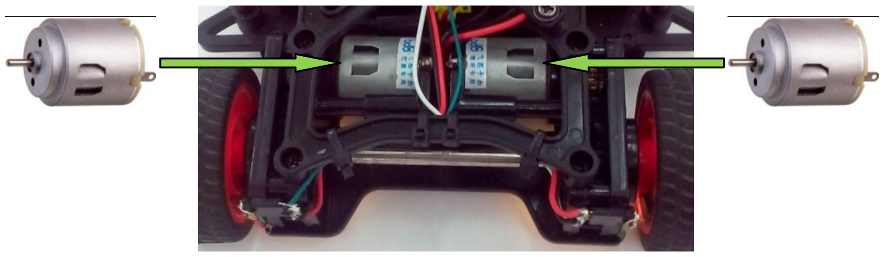
- ✓ The Line Scan Camera - hardware implementation: We used an aluminum tube with a diameter of 12 mm which we mounted the camera using screws that allows us to adjust the vertical position of the camera. This assembly was placed on an aluminum tube with a diameter of 10 mm which was fixed to the chassis through some components made by us.



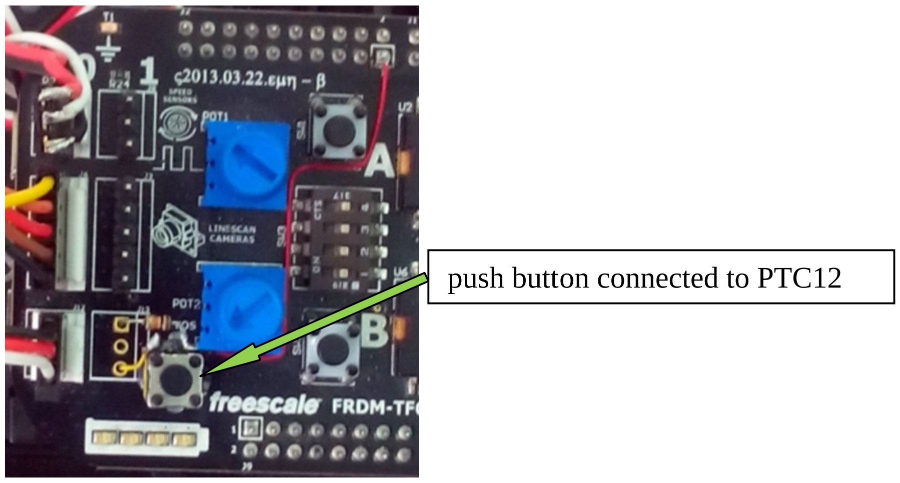
- ✓ The Digital IR Line Sensor - hardware implementation:



- ✓ The rear part of the vehicle implements the hardware traction of the vehicle:

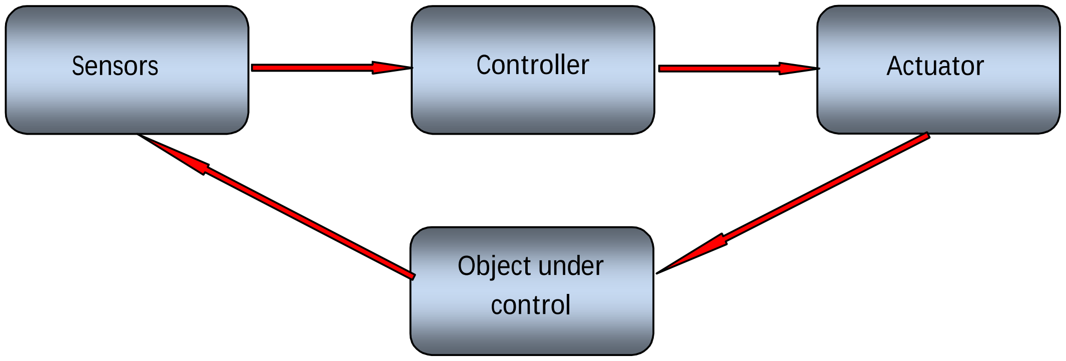


- ✓ The push button - hardware implementation: To select parameters we needed another extra push button.

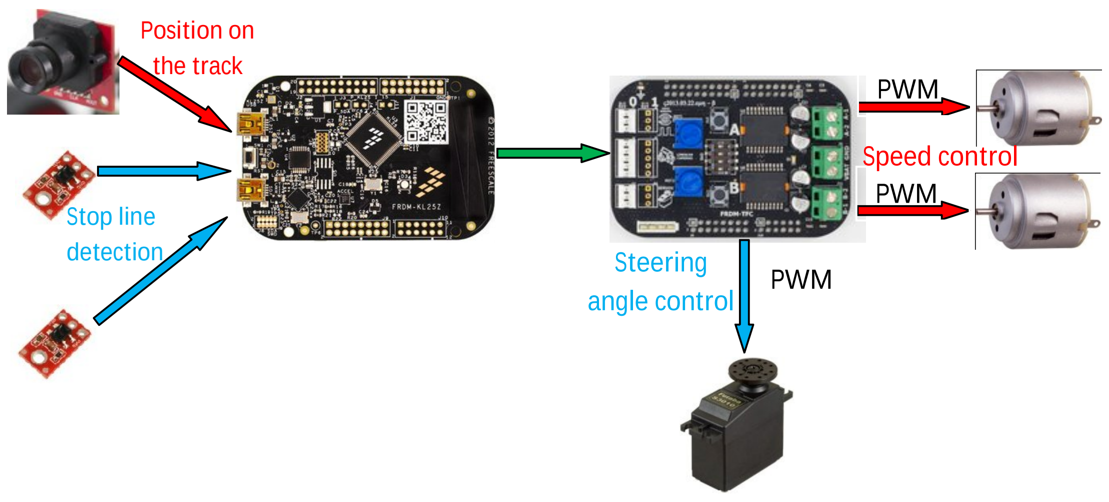


2. Description of control circuit design

The design of the control circuit is the image of a classical control loop:



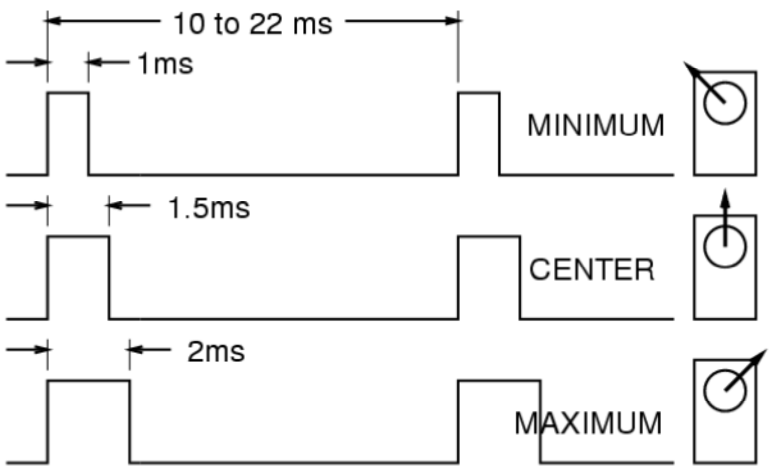
For our specific application, the control circuit devices are represented bellow:



All above control circuits and components are fully described on Freescale community portal except the line sensors and additional push button.

For the control algorithm point of view, is important to know the time behavior of all control devices included in the control loop.

2.1 Steering servo: Futaba S3010



PWM command

Basic Information

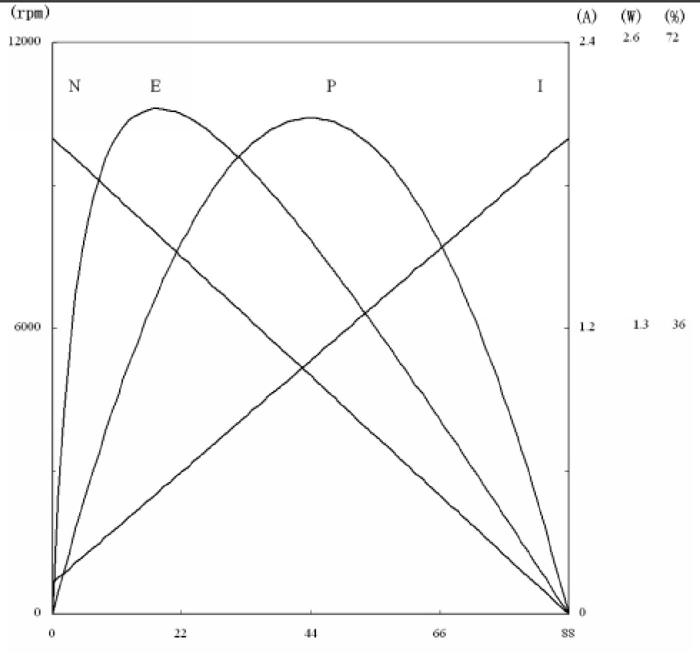
Modulation:	Analog
Torque:	4.8V:72.0 oz-in (5.18 kg-cm) 6.0V:90.0 oz-in (6.48 kg-cm)
Speed:	4.8V:0.20 sec/60° 6.0V:0.16 sec/60°

Weight:	1.45 oz(41.0 g)
Dimensions:	Length:1.57 in (39.9 mm) Width:0.79 in (20.1 mm) Height:1.50 in (38.1 mm)
Motor Type:	3-pole
Gear Type:	Plastic
Rotation/Support:	Single Bearing
Additional Specifications	
Rotational Range:	$180^o - 270^o$
Pulse Cycle:	Min: 10 ms/ Max: 22 ms
Pulse Width:	1 ms (left)/ 1.5 ms center/ 2 ms right
Connector Type:	Universal

From the above specifications (see [1]) we conclude that best Pulse Cycle from the control point of view is 10 ms (100 Hz). We can control the servo steering motor 100 times per seconds, hoping to obtain better results for control performances.

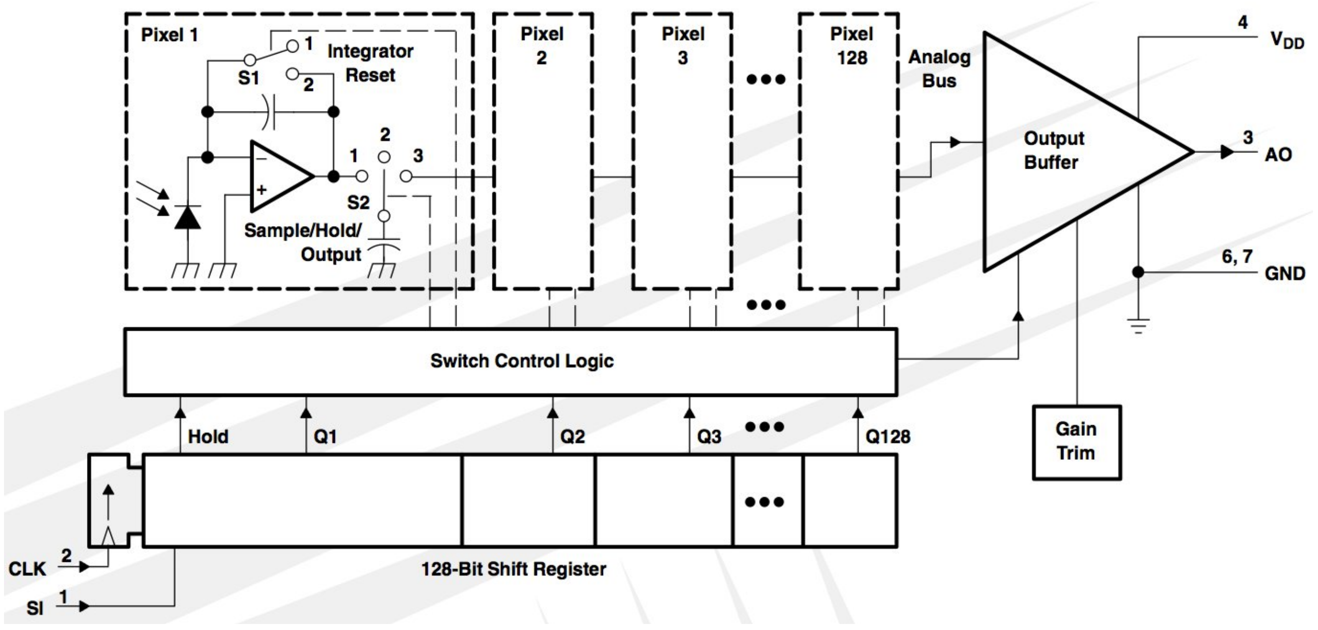
2.2Data sheet: Standard Motor "rn 260 c" winding 18130

No Load		Max Efficiency			Max Output			Stall	
Current	Speed	Current	Speed	Torque	Current	Speed	Torque	Current	Torque
A	rpm	A	rpm	g.cm	A	rpm	g.cm	A	g.cm
0.13	10000	0.51	7950	18	1.07	5000	44	2	88



From the control point of view, the PWM commands to the traction DC motors have no specific limitations. We decided to control the motor at a frequency of 600 Hz

2.3 Line Scan Camera: TSL1401CL



The sensor consists of 128 photodiodes arranged in a linear array. Light energy impinging on a photodiode generates photocurrent, which is integrated by the active integration circuitry associated with that pixel. During the integration period, a sampling capacitor connects to the output of the integrator through an analog switch. The amount of charge accumulated at each pixel is directly proportional to the light intensity and the integration time."

Integration Time: T

T = (1/fmax)*(n-18)pixels + 20us,

where n is the number of pixels

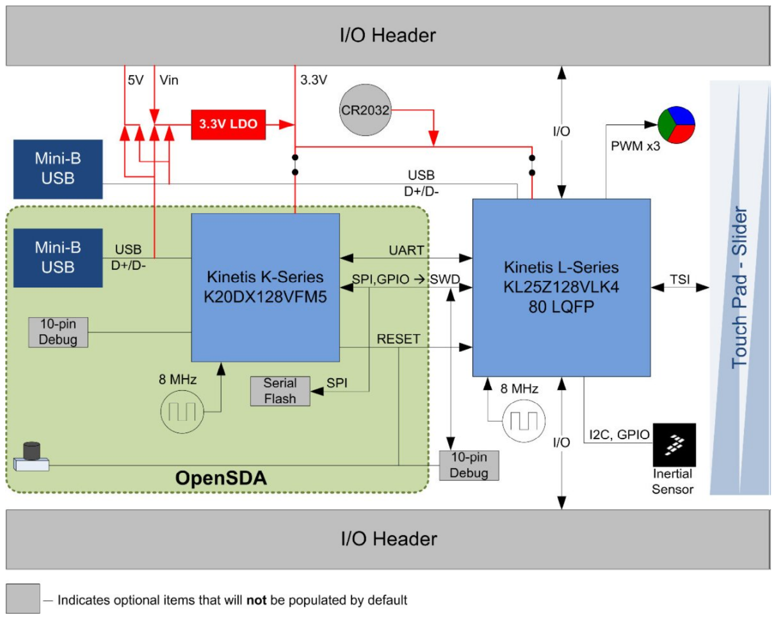
Minimum integration time: 33.75us
Maximum integration time: capacitors will saturate if exceeding 100ms

3. Description of the electronics design

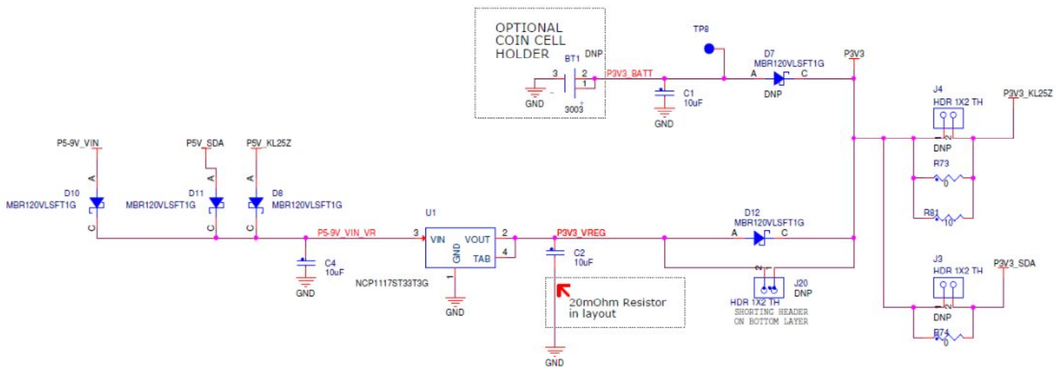
The electronics consists of Freescale Freedom development platform FRDM-KL25Z and the FRDM -TFC Freescale Cup Shield.

3.1 FRDM-KL25Z schematics and Functional architecture

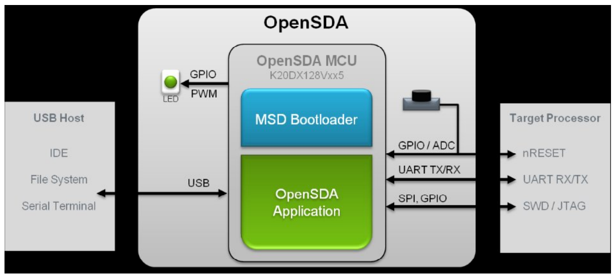
I/O Header



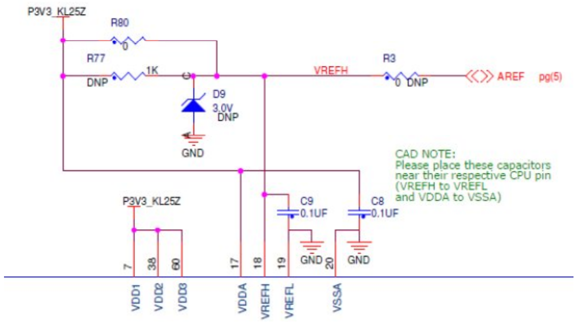
3.1.1 FRDM-KL25Z power supply schematics



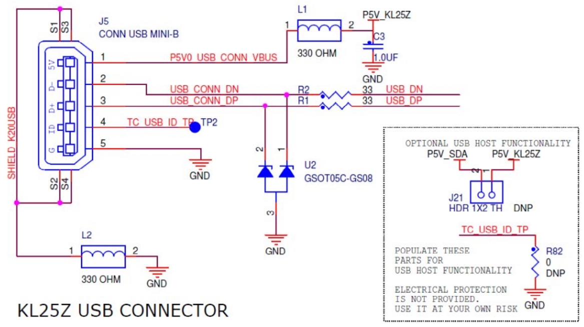
3.1.2 OpenSDA High-Level Block Diagram



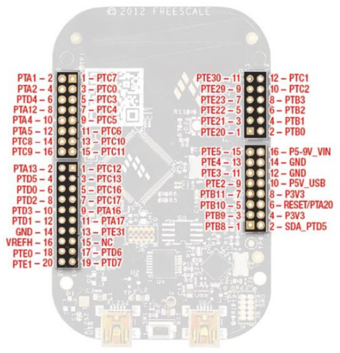
3.1.3 FRDM-KL25Z VREFH Circuit Schematic



3.1.4 FRDM-KL25Z USB Conector



3.1.5 FRDM-KL25Z Pinouts



3.1.6 KL25Z Microcontroller features:

- ✓ **32-bit ARM Cortex-M0+ core**
- up to 48 MHz operation
- Single-cycle fast I/O access port
- ✓ **Memories**
- 128 KB flash
- 16 KB SRAM
- ✓ **System integration**
- Power management and mode controllers
- Low-leakage wakeup unit
- Bit manipulation engine for read-modify-write peripheral operations
- Direct memory access (DMA) controller
- Computer operating properly (COP) Watchdog timer
- ✓ **Clocks**
- Clock generation module with FLL and PLL for system and CPU clock generation
- 4 MHz and 32 kHz internal reference clock

- System oscillator supporting external crystal or resonator
- Low-power 1kHz RC oscillator for RTC and COP watchdog

✓ **Analog peripherals**
- 16-bit SAR ADC w/ DMA support
- 12-bit DAC w/ DMA support
- High speed comparator

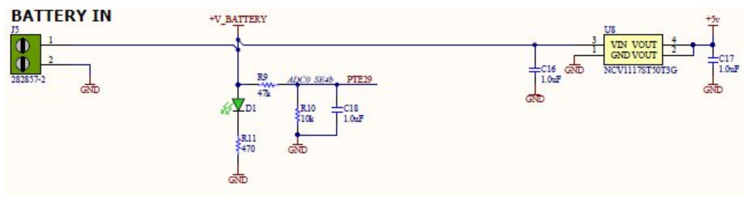
✓ **Communication peripherals**
- Two 8-bit Serial Peripheral Interfaces (SPI)
- USB dual-role controller with built-in FS/LS transceiver
- USB voltage regulator
- Two I2C modules
- One low-power UART and two standard UART modules

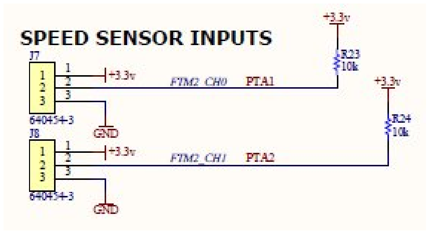
✓ **Timers**
- One 6-channel Timer/PWM module
- Two 2-channel Timer/PWM modules
- 2-channel Periodic Interrupt Timer (PIT)
- Real time clock (RTC)
- Low-power Timer (LPTMR)
- System tick timer

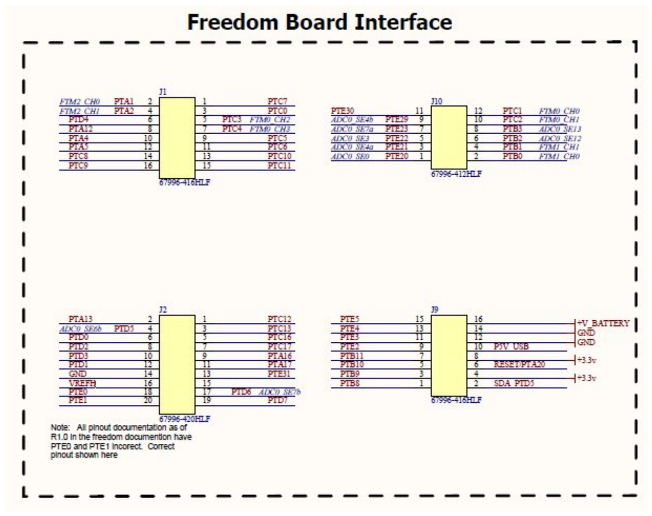
✓ **Human-Machine Interfaces (HMI)**
- General purpose input/output controller
- Capacitive touch sense input interface hardware module

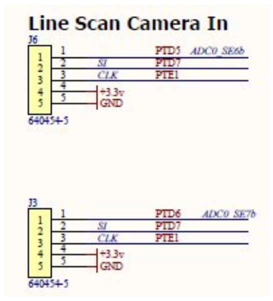
3.2 FRDM-TFC Freescale Cup Shield

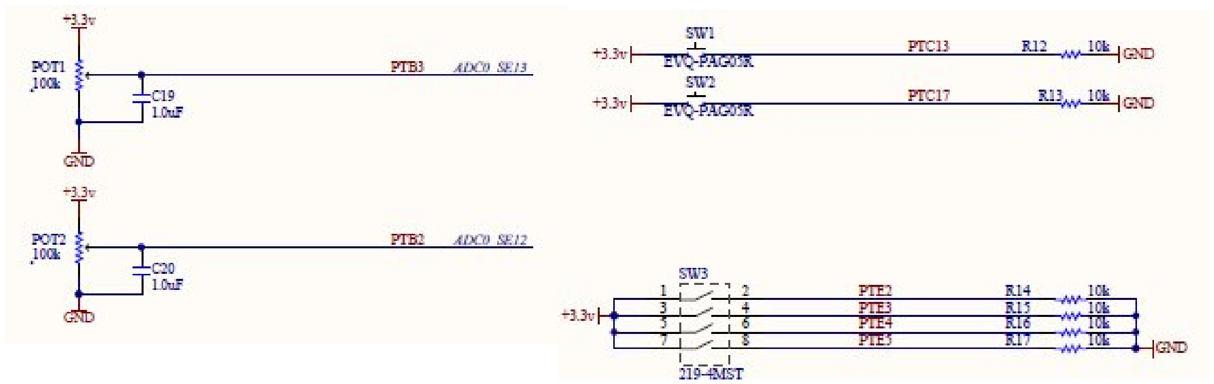
3.2.1 FRDM-TFC Freescale Cup Shield schematics



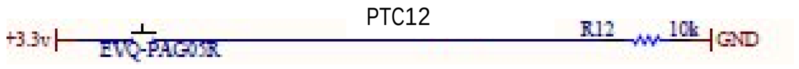


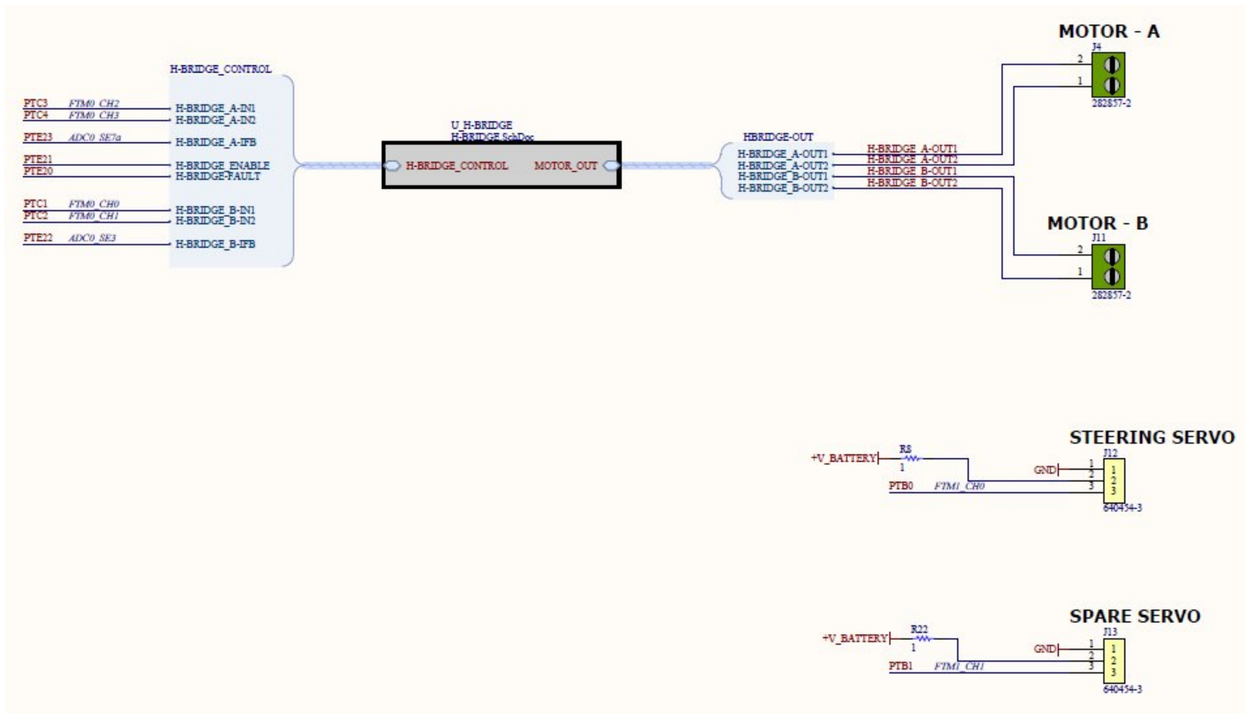

Freedom Board Interface
J1
67996-416HLF
J10
67996-412HLF
J2
67996-420HLF
J9
67996-416HLF
+V_BATTERY
P5V_USB
RESET/PTA20
SDA_PTD5
Note: All pinout documentation as of R1.0 in the freedom documention have PTE0 and PTE1 incorect. Correct pinout shown here



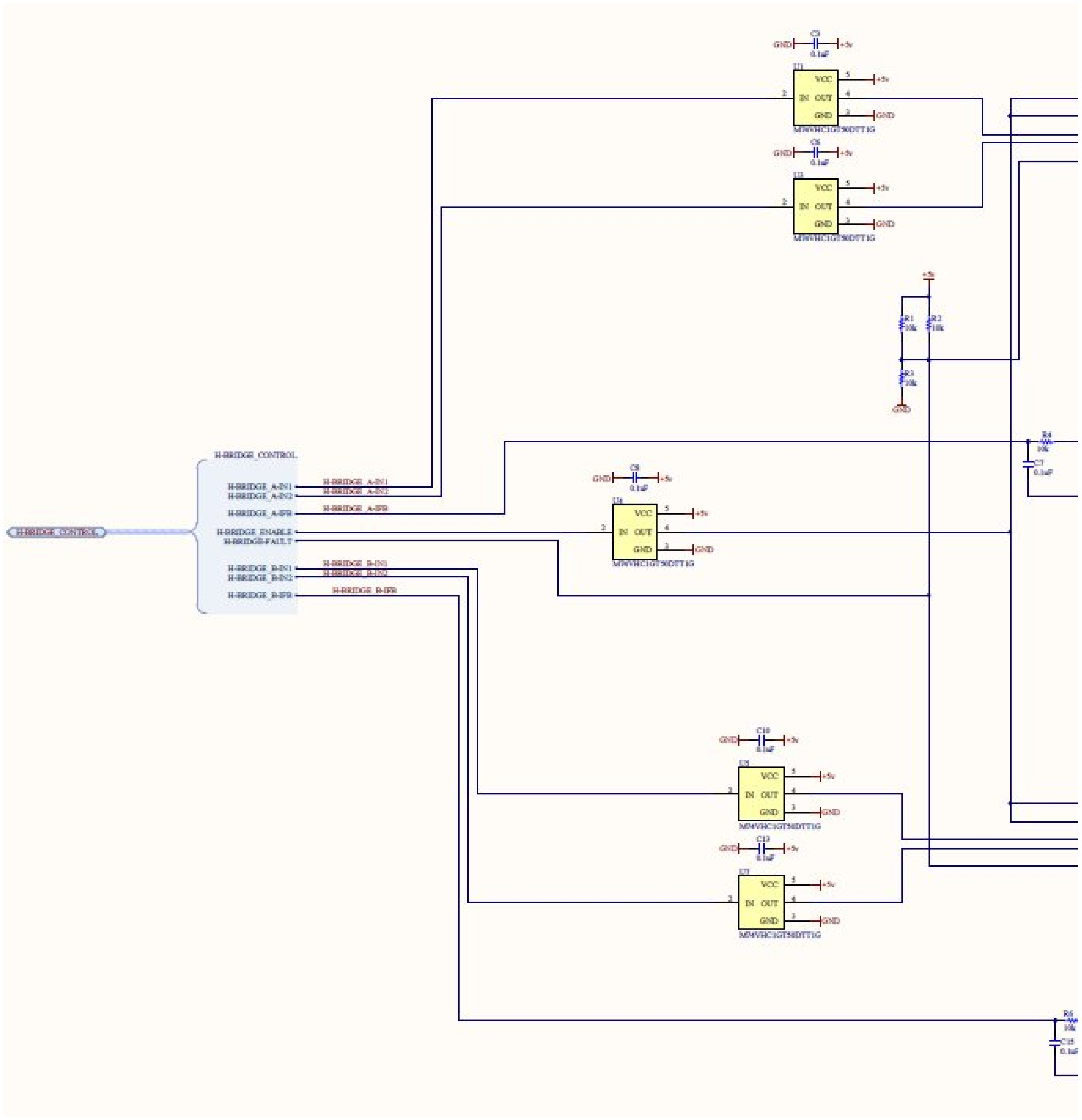


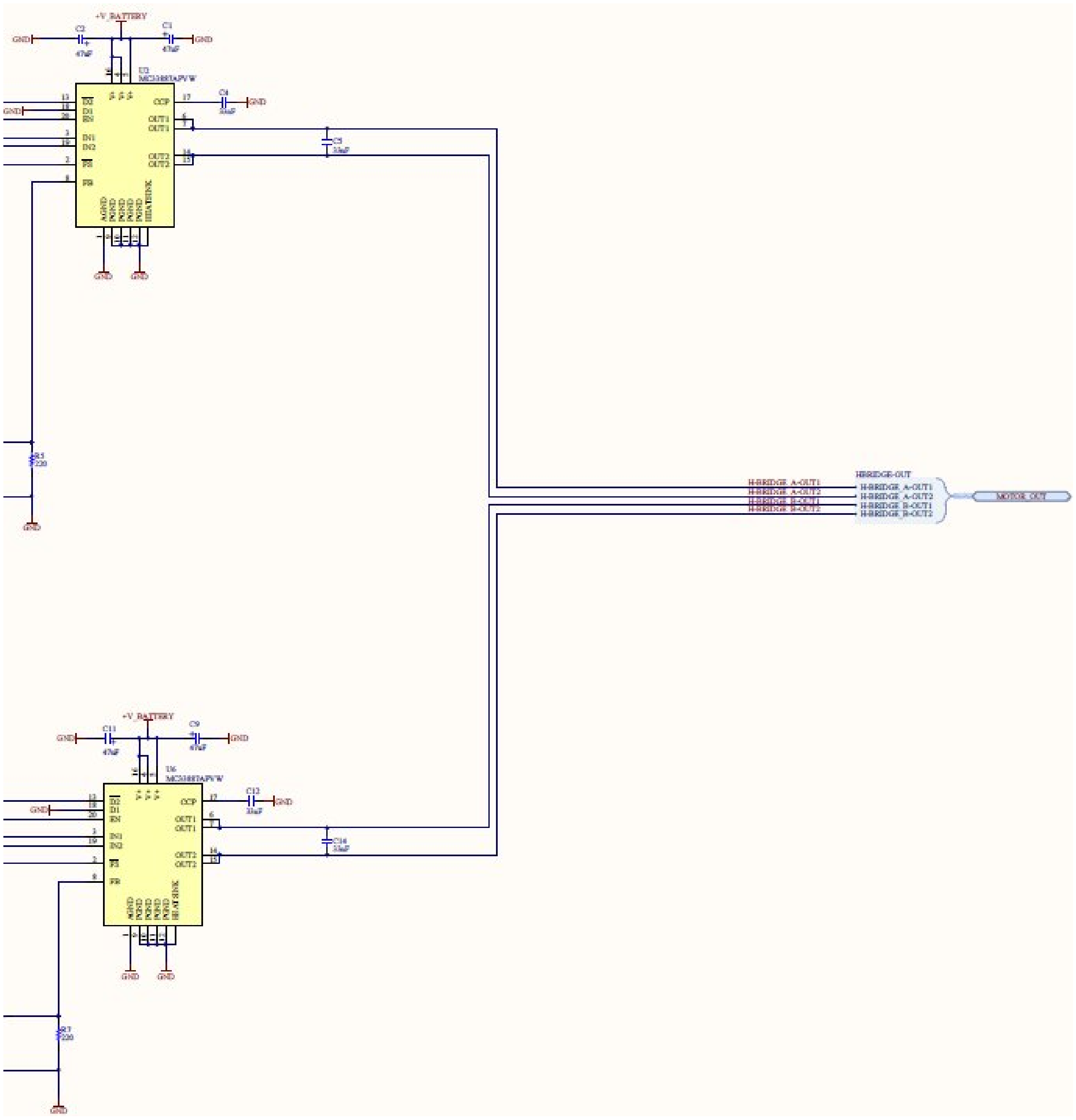
Additional button mounted by us





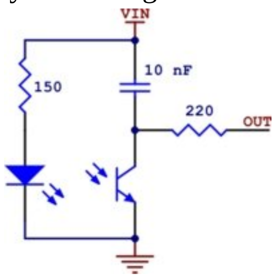
H-Bridge


H-BRIDGE_CONTROL
H-BRIDGE_A-IN1
H-BRIDGE_A-IN2
H-BRIDGE_A-IFB
H-BRIDGE_ENABLE
H-BRIDGE-FAULT
H-BRIDGE_B-IN1
H-BRIDGE_B-IN2
H-BRIDGE_B-IFB
U1
U3
U4
U5
U7
MM74VHC1GT50DTT1G
VCC
IN
OUT
GND
+5v
C3
C6
C8
C10
C13
C7
C15
0.1uF
R1
R2
R3
R4
R6
10k



3.3 Digital IR Line Sensor - QRE1113

The QRE1113 reflectance sensor carries a single infrared LED and phototransistor pair. The phototransistor uses a capacitor discharge circuit that allows a digital I/O line on a microcontroller to take an analog reading of reflected IR by measuring the discharge time of the capacitor.



4. Description of control software design

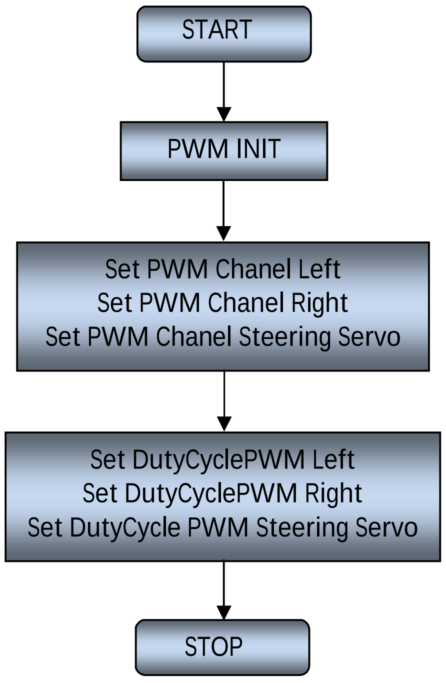
4.1 Drivers

- **Port Driver:** Performs port configuration
- **DIO Driver (Digital I/O):**Implements Init/Get/Set APIs for correct use of Digital I/Os.
- **PWM Driver:**Implements Init/SetDuty/SetFrequency/GetTrigger APIs for correct use of PWM channels
- **ADC Driver:** Implements Init/GetValue APIs for correct use of ADC channel.

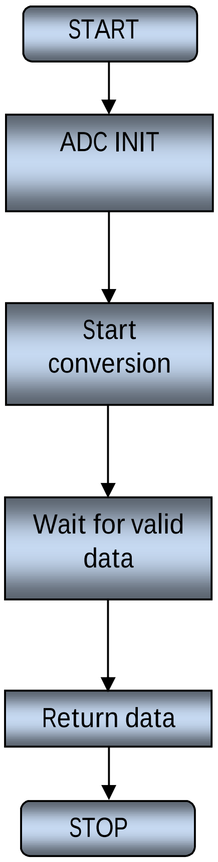
DIO driver and Port driver were not realized as separate modules but they are integrated in PWM and ADC drivers.

Example of drivers design

✓ *PWM Driver*



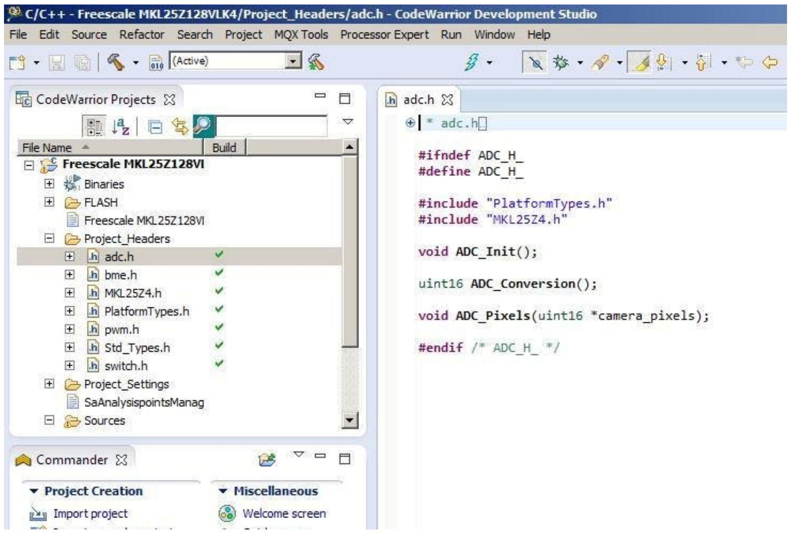
- ✓ *ADC Driver:*



4.2 Headers

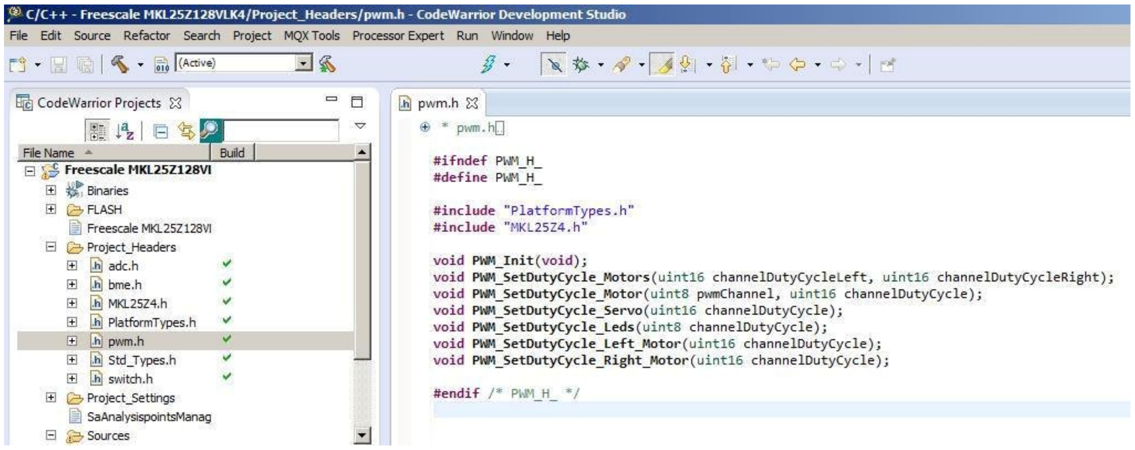
Example of headers

- ✓ *ADC.h:*



```
* adc.h

#ifndef ADC_H_
#define ADC_H_

#include "PlatformTypes.h"
#include "MKL25Z4.h"

void ADC_Init();

uint16 ADC_Conversion();

void ADC_Pixels(uint16 *camera_pixels);

#endif /* ADC_H_ */
```

✓ *PWM.h*



```
#ifndef PWM_H_
#define PWM_H_

#include "PlatformTypes.h"
#include "MKL25Z4.h"

void PWM_Init(void);
void PWM_SetDutyCycle_Motors(uint16 channelDutyCycleLeft, uint16 channelDutyCycleRight);
void PWM_SetDutyCycle_Motor(uint8 pwmChannel, uint16 channelDutyCycle);
void PWM_SetDutyCycle_Servo(uint16 channelDutyCycle);
void PWM_SetDutyCycle_Leds(uint8 channelDutyCycle);
void PWM_SetDutyCycle_Left_Motor(uint16 channelDutyCycle);
void PWM_SetDutyCycle_Right_Motor(uint16 channelDutyCycle);

#endif /* PWM_H_ */
```

4.3 Control algorithm

Control algorithm contains**:**

- **Input Application (Sensor):** Read data from Line camera and performs basic data processing for the Controller Application
- **Controller:** Implement two different controllers: one for the Traction DC motors (Left and Right) and one for the Steering Servo.
- **Output Application:** Implements the transfer of data commands (2 PWM commands for the Traction DC motors (Left and Right) and one PWM command for the Steering Servo) from controller to drivers from the Basic Software.

Control algorithm implements two main functionalities:

- the PID controller for setting the reference speed for and a "differential" command, in order to help the car to keep the track on difficult curves or in sudden track changes
- PID control for Servo Steering and Traction DC Motors

For implementing the control algorithm, we have chosen the classical PID control algorithm, always an optimal choice for the control algorithm of the intelligent car. Short theory on this control algorithm is presented in this module description.

If we consider the continuous PID control low:

$$H_C(s) = \left(K_P + K_I \cdot \frac{1}{s} + K_D \cdot \frac{s}{T_g s + 1}\right)$$

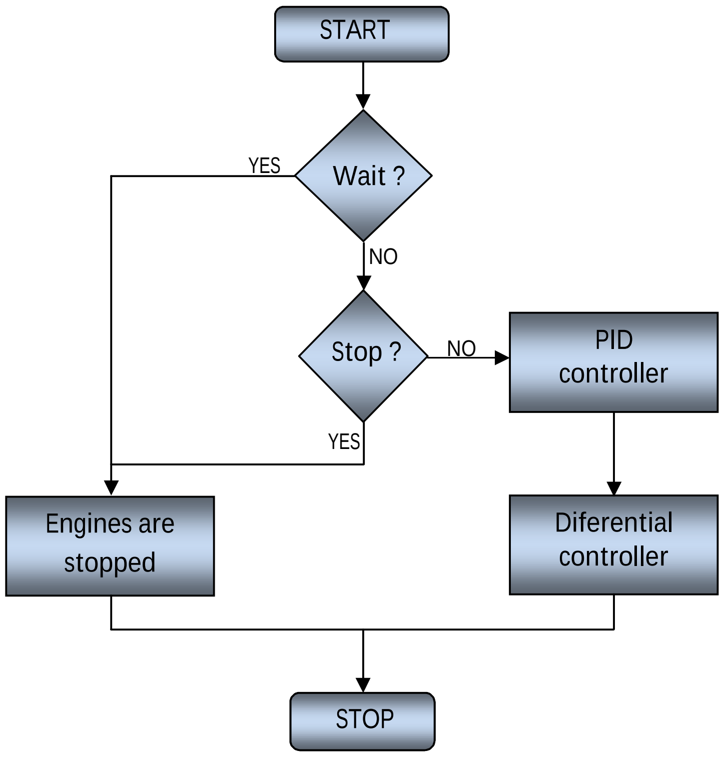
For discretizing this continual control laws, we can use different bilinear methods:

a. *Tustin substitution*: $H_d(z) = H_c(s)\Big|_{s = \frac{2}{T_s} \cdot \frac{1-z^{-1}}{1+z^{-1}}}$

b. *Forward Rectangular*: $H_d(z) = H_c(s)\Big|_{s = \frac{1}{T_s}\cdot(z-1)}$

c. *Backward Rectangular :* $H_d(z) = H_c(s)\Big|_{s = \frac{1}{T_s}\cdot\frac{z-1}{z}}$

Ts represents the sample time of the discretized system. In our case, Ts is 2 ms for the traction motors and 10 ms for steering servo.



Reference speed is maximum controllable speed by our algorithms, and is determinate in experiments. Usually the *Reference speed* is a certain percentage from the maximum Speed (PWM 100% to Traction DC motors)

StartTime and **StopTime** are time moments when the car 'starts running' and 'stops moving'.

T**rack_error** are input for our algorithm.

EngineReqStop stop engines if Start-Stop line was see.

We use for computing the '`SetDutyCycle_Motor`' a simple PID-algorithm:

SetDutyCycle_Motor (k) = $\mathbf{K_p}$* *track_error*(k) + $\mathbf{K_{i*}}$ [Integral+ *track_error*(k)* $\mathbf{D_t}$]+
$\mathbf{K_d}$ * [*track_error*(k)- *track_error*(k-1)]

The output from the PID-algorithm is subtracted from the "Reference_speed", the maximum speed imposed for the race.

Further, the ‘differential’ algorithm generates the commands for the Left/ Right DC motors in a differential way:

Left_duty_cycle(k) = [**Refference_speed** - *SetDutyCycle_Motor (k)*] + sign(*track_error*(k)) * $\mathbf{K_{p\_diff}}$ * *track_error*(k)

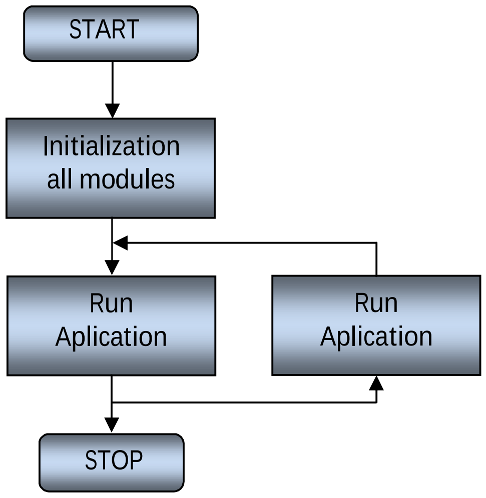
Right_duty_cycle(k) = [**Refference_speed** - *SetDutyCycle_Motor (k)*] - sign(*track_error*(k)) * $\mathbf{K_{p\_diff}}$ * *track_error*(k)

- ✓ *ServoApp*

SetDutyCycle_Servo calculation algorithm is similar to that for *SetDutyCycle_Motor*

4.4 Main module

This module work as a pseudo-OS. His main role is to initialize all modules and correct run of applications.

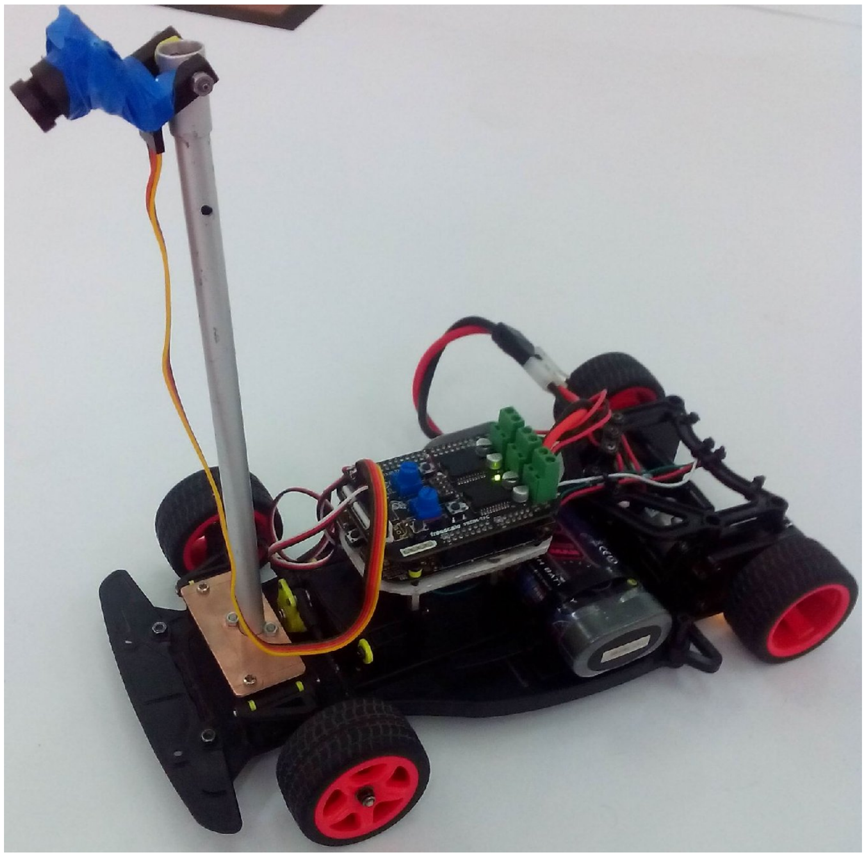


First we initialize all drivers, handlers and applications before being used. After that we run all applications in correct order (Sensors → Controllers → Actuators). When we finish running all applications we will give control to RTC trigger to rerun applications again at a specific moment.

5. Race car characteristics

5.1. Total weight and dimensions of the reengineered car

Characteristics	Value
Total weight	0.98 Kg
Max height	287 mm
Max length	285 mm
Max width	160 mm



5.2 Power consumption

- no load running power consumption = 0.7A (~5.04W) when command to motors is full speed
- consumption on the race track = 1.8A (~12.96W)
- max consumption = 3.2A (~23.04W)

5.3 Count and type of sensors used

- 1 x original Freescale Line Camera
- 2 x Digital IR Line Sensor - QRE1113

5.4 Number of servo motors besides the existing driving motors and rudder motors of the car model

No other servo motors was been used.

6 References

[1] LILLI, GIORDANO (2013) *HARDWARE E SOFTWARE DI CONTROLLO DEL VEICOLO "FREESCALE CUP"*.http://tesi.cab.unipd.it/42394/1/Tesi.pdf

[2] T. N. Blalock and R. C. Jaeger, *Microelectronic Circuit Design*, McGraw-Hill, 3rd ed., 2008

[3] FRDM-KL25Z User's manual

[4] FRDM-KL25Z_SCH_REV_E

[5] FRDM-TFC_Schematics

[6] Taos, *TSL1401CL 128x1 linear sensor array with hold*, July 2011

[7] Freescale University Programs (https://community.freescale.com/community/uvp)

[8] Futaba (http://www.futabarc.com)

[9] Servo motor tutorial (http://www.seattlerobotics.org)

[10] Standard Motors (http://www.standardmotor.net/)

[11] http://www.autosar.org/download/R3.1/AUTOSAR_LayeredSoftwareArchitecture.pdf

[12] EMMA team (University of Craiova) technical report– Freescale Cup 2014