Software description

Litan Virgil

1) The timer

The system needs a timer to determine the current speed of the vehicle. This timer is composed of an interrupt routine that increases a register each millisecond.

1.1) Initialization of the interrupt

The interrupt vector (TIMER1 COMPA) and the desired time interval that the interrupt occurs is initialized using the registers: TCCR1B, OCR1AH, OCR1AL, TIMSK1;

1.2) Using the interrupt

The interrupt service routine TIMER1_COMPA_vect will increase the value of a general purpose register that will be accessed from the main code loop.

2) Analog input

The system needs an analog input to read the value from a potentiometer. This value will determine the desired speed of the vehicle. In order to do that the internal ADC is used.

2.1) ADC setup

The ADMUX register is used to set the voltage reference to be AVcc. The ADCSRA register is used to set the prescaler of the ADC to 128 in order to set the sampling frequency and ADCSRA is used to enable the ADC.

2.2) Using the ADC

ADMUX is used to set the resolution and the security mask. After the conversion is finished the result is found in ADCH.

3) Reading comparators outputs.

As the outputs of the comparators are digital, the signals are read using general purpose I/O registers.

3.1) Register setup

DDRD is set using a bitmask to use the desired pins as inputs.

3.2) Using the inputs

The inputs can be read from the pins using the PORTD register.

4) Motors control

In order to control the motors generating PWM signals is necessary. This is done by using 2 8-bit timers that will each control 2 channels with separate comparators.

4.1) PWM timer setup

The timer must be initialized choosing the prescaler (using TCCR0B on timer ne and TCCR2B on timer 2), operation mode (fast and phase correct) using TCCR0A, TCCR2A respectively. Also, inverting or non-inverting mode is set using this register.

4.2) Match values

The match values for the counter (that will toggle the output of the pin, generating the PWM) are set in OCR0A for channel A on timer 0, OCR0B for channel B on timer 0, OCR2A for channel A on timer 2 and OCR2B for channel B on timer 2. Those values determine the width of the pules.

4.3) Motor control

In order to control one motor one pin will have to be 0 and the other one generates the PWM controlling the speed of the motor. To reverse the direction of the motor the first pin generates PWM and the second one is 0.

The speeds and direction for the motors will be stored in 2 pairs of general purpose registers and updated accordingly.

5) Main flow of the program

5.1) Setup

Initialize all hardware described at points 1), 2) ,3) and 4).

5.2) Infinite loop

5.2.1) Read the desired speed using 2)

5.2.2) Compute the current speed using 1) and 3).

5.2.2.1) Save previous value for each input of the comparator and compare it with the current value. If the two are different then a falling / rising edge is detected.

5.2.2.2) If there was a rising edge: Get the current time by reading the value of the register holding the number of milliseconds elapsed from the beginning of the program run and save the previous value.

5.2.2.3) Get the time interval between the two rising / falling edges.

5.2.2.4) Compute the instantaneous speed by dividing a constant (number of cm corresponding to one segment of the encoder) with the time interval.

5.2.3) Compute the difference between the desired speed and the actual speed for each motor.

5.2.4) Adjust the values in the PWM dedicated registers (increase / decrease by an amount) accordingly with the difference

5.2.5) Apply the new values to the motors

6) Additional improvements (in case the project is finished earlier)

The proportional control system used (5.2.2.4) could be improved by adding an integral component (summing the previous errors) and a derivative component (using the difference between two consecutive errors).

6) Toolchain and IDE

The AVR assembler is formerly known as AVRASM2. The .hex file generated after the assembling process is done is loaded with avrdude into the flash memory of the microcontroller using and programmer (USBASP). The programmer communicates with the microcontroller via SPI and the programing is done using ISP. The whole toolchain is linked and used with Atmel Studio 7 IDE.