D-MARK Script (DMS) Documentation

Overview

DMS has been designed to make the D-MARK Controller Board (DCB) run without programming, compiling, and flashing the microcontroller. Rather than existing full-featured programming languages like C and C++ et al, DMS is simply a plain text saved in a media i.e. microSD, containing configuration part and a command part including simple conditional syntax. DMS is then being interpreted and executed line by line by the firmware on the central microcontroller of the DCB. Following are the configuration, commands and a few examples of using DMS to control its on-board inputs and outputs

Configurations & Commands for the Outputs

Use tab as a delimiter to separate between assigned values and commands or configurations. Adding # in front of the configurations and commands will force D-MARK Script Interpreter firmware (DMSI) to skip that lines from the execution process.

1. MOSFET OUTPUTS (4 channels)

Configurations:

x = channel number 0 - 3 while 0 refers to FAN controlling MOSFET output (ch 0), and 1-3 are for the MOSFET output ch 1 - 3 (labelled MO1 to MO3 on board)

mosfet_x.mode [PWM | Digital | Trig]	
mosfet_x.pwm.freq [0.1-100kHz]	default = 5 kHz
mosfet_x.pwm.duty [0-100%duty]	default = 50% duty cycle
mosfet_x.trig.delay [1-60000 ms]	default = 500 ms

Commands:

mosfet_x.on | mosfet_x.off | mosfet_x.trig		perform the action according to the above configurations
mosfet_x.pwm.on	value	set duty value and turn on pwm
mosfet_x.digital	0/1	mosfet digital output
mosfet_x.trig	0/1, time	mosfet trig output with delay time

2. RELAY OUTPUTS (3 channels)

Configurations:

x = channel number 0 - 2

relay_x.mode [Digital | Trig]
relay_x.trig.delay [1-60000 ms]

Command:

relay.on | relay.off | relay.trig

SERVO OUTPUTS

Configurations:

servo.freq [50-1000] in Hz	default = 50 Hz (20 ms period) for most RC servos
servo.position_center [500-2000]	default = 1520 us
servo.position_min [0100-1000]	default = 1000 us (many servos can go down to 500 us)
servo.position_max [2000-3000]	default = 1500 us (many servos can go up to 2500 us)

Note: Futaba RC servo e.g. FT3001 will turn only -60 to +60 degree, not 180 degree like modern servos
For example: -90 deg = 1000 us, center = 1500 us, +90 deg = 2000 us

Commands:

servo.center	
servo.turn [µs]	servo turns to the position defined by the pulse width duration

servo.trim value servo position trims; negative values set ultimate left offset position

STEP MOTOR OUTPUT:

Configurations:

step_motor.pulse.frequency_min [0.1-500 kHz]	default = 1 kHz
step_motor.pulse.frequency_max [0.1-500 kHz]	default = 100 kHz
step_motor.step_length [0-60000 step/mm]	default = 800
step_motor.feedrate [0-60000]	default = 600 mm/min
step_motor.speed_max [0-60000]	default = 2400 mm/min
step_motor.accel [0-6000]	default = 2000 mm/s^2
step_motor.step_count [value]	defines motor's steps per revolution
step_motor.step_angle [value]	define motor's rotation angle per step

Commands:

step_motor.enable | step_motor.disable	enable/disable the motor output driver
step_motor.turn.left | step_motor.turn.right	turn the motor continuously
step_motor.turn.steps [steps,time(s)]	turn number of the steps within defined time
step_motor.feed.forward [0-6000] in mm	linear-feed forward
step_motor.feed.backward [0-6000] in mm	linear-feed backward
step_motor.angle.backward [angle value]	angle value to turn clockwise to
step_motor.angle.forward [angle value]	angle value to turn counterclockwise to

Note: step_length equals to (steps per revolution of the motor/mm pitch)/microstepping
For example: if a step motor with 200 steps/revolution is driven by 1/4 microstepping mode (selectable with the on-board jumper) connected with a ball screw 1 mm pitch, then the step_legth value will be 200/1/4 = 50. Now if we write a command "step_motor.feed.forward = 10", the motor will turn clockwise and advance exactly 10 mm in linear position

BUZZER OUTPUT

Configurations:

buzzer.frequency [50-15000], default = 2500 Hz

Commands:

buzzer.on [50-15000], default = 2500 Hz
buzzer.off

DISPLAY OUTPUT

D-MARK features a 6p box male header intended for connecting with small displays. Functions implemented was based on the popular 1.3" 128x64 pixels OLED module included with SH1106 driver.

Configuraiton:
None

Commands:

oled.clear	clear the display
oled.locate [x,y]	locate the lcd cursor at pixel x and y
oled.print ["string"]	print string at pixel x, y; 12 characters/line max
oled.print.analog [ch]	ch is the channel to display its value
oled.print.digital [ch]	ch is the channel to display its value
oled.print.ntc	print NTC temperature values
oled.print.ntc_unit	print NTC temperature unit
oled.print.ldr	print LDR value
oled.char [value]	print single character e.g. unit symbol

Configurations & Commands for the Inputs

D-MARK consists of 3 analog and 4 digital inputs in which LDR, NTC, and VIN labelled on the PCB refer to the analog channels while IN1, IN2, IN3, IN4 are the digital ones. LDR is for measuring light intensity (expressed in LUX), NTC is the Negative Temperature Coefficient resistance temperature sensor, VIN is a 0 - 30 VDC analog voltage measurement input. In addition 2 SenseLog USB inputs also provided for user's custom transparent single wire communication sensor

interface.

LDR Configurations:

ldr.r1 [value]	set R1 value in Ohm
ldr.i1 [value]	set I1 value in Lx (Lux)
ldr.r2 [value]	set R2 value in Ohm
ldr.i2 [value]	set I2 value in Lx (Lux)

Note: R1, I1 and R2, I2 are the resistance and light intensity value pairs in the linear slope region obtained from the LDR intensity-resistance plot (available in most LDR datasheet).

NTC Configurations:

NTC.Unit [0 or 1]	0 = deg. C, 1 = deg. F; default = 0
NTC.Constant_B [2000-5000]	default beta value = 3950
NTC.Constant_R0 [1000-100000]	default = 10000 Ohm (10k and 100k are most typical)

Command:
None

Script Conditional Syntax

waitms	value	delay time value in millisecond
goto	label	jump to the label name
:label_name	label	label_name
if.dix	label	if digital input channel x is high then jump to command label
if.!dix	label	if digital input channel x is low then jump to command label x = 1 to 4 to indicate the channel of interested
if.[ai].gt	value,label	if analog input value is greater "value" jump to command label
if.[ai].lt	value,label	if analog input value is lower "value" jump to command label
if.[ai].eq	value,label	if analog input value equals "value" jump to command label [ai] can be NTC, LDR or Voltage, omit the bracket when writing

==========

DMS EXAMPLES

==========

Example #1: Alarming through a buzzer at 1 kHz frequency and also turning on the relay channel 1 as long as there is a digital signal detected on the digital input channel 1

All we need to do is save the following text in to a provided micro SD card with the file name "DMS.txt" (script.txt was used for earlier prototyping)

```
relay_1.mode      Digital          //we configure relay ch-1 to perform a simple on-off function

:loop
  if.di1   ALARM                   // if condition met jump to ALARM
    buzzer.off                     // if not met, turn the buzzer and the relay off
    relay_1.off
goto      loop                     // check the state of the input 1 forever
  :ALARM                           // ALARM action to do
  buzzer.on       1000             // let the buzzer sounds it out at 1000 Hz (sensitive to the human ears)
  relay_1.on                       // then also turn relay channel 1 on
goto      loop
```

Fact: We wrote 10 lines of texts using 2 minutes for this, oops I forgot to compile it....

Example #2: Measuring a room temperature using NTC (beta = 3950, R0 = 10 k-Ohm) then show it on the OLED display

This one is a bit more complicate, yes we need to define some basic NTC sensor parameters namely Beta coefficient and Ro values (Ro = its resistance expressed in Ohm at 298 Kelvin)

Let's start by typing:

```
NTC.unit          1                 // we will calculate in degree F
NTC.constant_r0   10000             // we bought a popular 10 k-Ohm, B = 3950 NTC
NTC.constant_b    3950              // set the beta to 3950

oled.clear                          // clear display first

:loop                               // set a never-ending loop starting point
  oled.locate     0,12              // set display cursor to pixel x = 0, y = 12
  oled.print.ntc                    // print it out to OLED
goto     loop                       // repeat it forever
```

Fact: We wrote now 8 lines of texts using 1.5 minutes for this, oops I forgot to import that libs...

We can simply omit the first 3 lines as 10 k-Ohm, and beta = 3950 are default config values while 0 for the NTC.unit refers to the display unit in degree C, then, total lines of code become 5 and we need 59 second for it ;-), cheers!

More examples will be added in a separate text files on the GitHub repo.

Facit: The script functions are keep extending as long as there is enough flash and RAM available on the target MCU, please feel free to suggest a new function or join us developing DMSI to make the human-hardware interaction becomes a breeze for most people.

Revision 12.2020.1

Dr.-Ing. Pichanon Suwannathada