PropDAQ Firmware

Table Of Contents

Introduction

This describes the firmware (software) developed for the Propeller MCU used in our PropDAQ-Module. It provides the necessary communication with the PC through our "PropMeter PC Data Acquisition Software", and control the various input and output signals.

Functionality

The PropDAQ Firmware needs to provide the following functionality:

- Communication link with PC
- Read Analog Input values (QTY=4)
  This is accomplished by communicating with the ADC chip MCP3204 from Microchip.
- Read Digital Input (QTY=4) (reading state of pins)
- Setting state of Digital Output (QTY=4) (setting output state of pins)
- Setting value of Analog Output (QTY=2)

Propeller Specific Hardware

The Propeller chip has 8 Cogs which can operate simultaneously, either independently or cooperatively, sharing common resources through a central hub. The developer has full control over how and when each cog is employed; there is no compiler-driven or operating system-driven splitting of tasks among multiple cogs. A shared system clock keeps each cog on the same time reference, allowing for true deterministic timing and synchronization.

For optimum speed, we have elected to run each PWM Analog Output in a dedicated Cog (we have developed our own PWM object in Prop Assembly that can run at more than 20kHz). For the same reason, we have also elected to run the ADC communication in a dedicated Cog, and the same for the PC communication protocol. As it is, the Cogs usage is as follows:

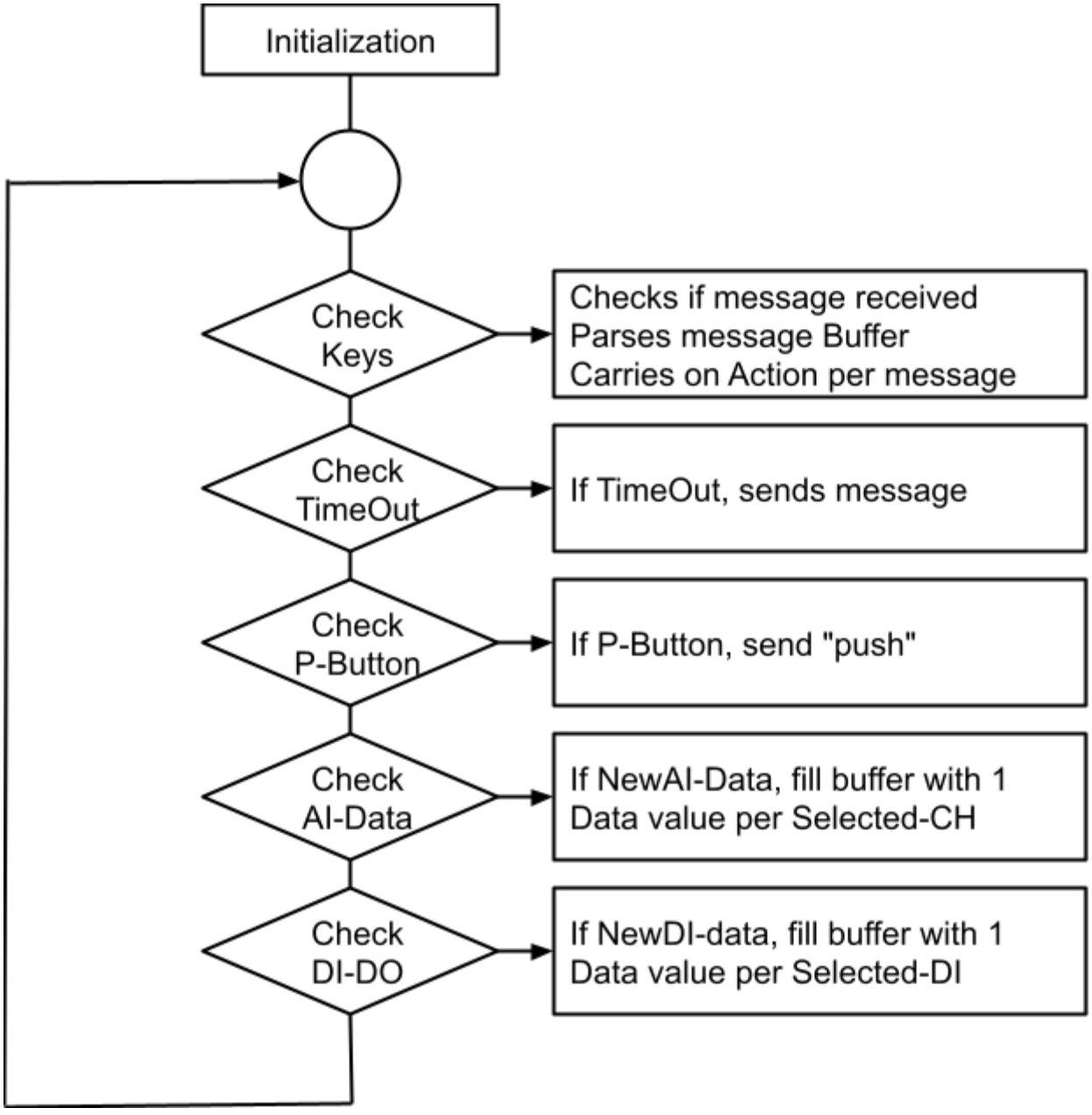
1. cog 0 - initial boot-up
2. main program
3. PC com
4. ADC read (Analog Inputs)
5. DI / DO
6. PWM Analog Output 0 (AO-0)
7. PWM Analog Output 1 (AO-1)
8. NA

For the digital Inputs or Outputs, the Propeller chip has 32 I/O pins. QTY=2 are used for serial communication with PC, QTY=2 for EEPROM reading, QTY=3 pins for ADC reading Analog Inputs, QTY=2 for PWM Analog Outputs, QTY=4 for Digital Inputs and QTY=4 for Digital Outputs. The pin assignment is as follow:

- P31 (pin 38) - RX (receive from PC)
- P30 (pin 37) - TX (transmits to PC)
- P29 (pin 36) - EEPROM SDA
- P28 (pin 35) - EEPROM SCL
- P2 (pin 43) - PWM AO-0
- P3 (pin 44) - PWM AO-1
- P4 (pin 1) - ADC Csn
- P5 (pin 2) - ADC DI-DO
- P6 (pin 3) - ADC SCLK
- P16 (pin 19) - DO-0
- P17 (pin 20) - DO-1
- P18 (pin 21) - DO-2
- P19 (pin 22) - DO-3
- P20 (pin 23) - DI-0
- P21 (pin 24) - DI-1
- P22 (pin 25) - DI-2
- P23 (pin 26) - DI-3

Overall Architecture

The overall functionality of the firmware is organized around a main perpetual loop defined in the PropDAQ-PWM.spin file. The functionality of this loop is illustrated below.



Block diagram illustrating the functionality of the main loop.

The actual code is written in Propeller Spin and Propeller Assembly language in four different program files:

- PropDAQ-PWM.spin
  Contains the Main control loop.

Includes the following objects

```
  Msg       : "Messager"
  Stk       : "Stack Length"
  pwm1      : "AsmPwm"
  pwm2      : "AsmPwm"
```

Defines commands

```
        NameTable       byte
"version",0,"model",0,"nchannels",0,"push",0,"set",0,"start",0,"stop",0,"dir",0,"type",0
```

Starts the various Cogs

```
  pwm1.start( %010, 1015)
  pwm1.changePwmAsm(000)
  pwm2.start( %001, 1015)
  pwm2.changePwmAsm(000)
  ADCCOG:=cognew( @DACLoop, 0 )+1                    ' start ADC
  DIGCOG:=cognew( @DigitalLoop, 0)+1                 ' start Digital Cog
  RDCOG:= cognew( ReadLoop, @Stack )+1                ' start readloop
```

‘ This starts the ReadLoop in a new cog.

Contains the Assembly code for DigitalLoop and DACLoop

```
pub ReadLoop | n, m, mask, curVal, curTime, pushed, ldbg1, ldbg2, ldbg3, ldbg4, duty, inc
  duty:=0
  inc:=1

  repeat                    ‘main loop
    Msg.checkKeys                                          ' check buffer
    Msg.checkTimeout                                       ' check timeouts

    if not ina & btnMask and not pushed                    ' check push-button
                pushed~~
                Msg.sendKey(String("push"),4,@ExData,0,1)
    if pushed and ina & btnMask
                pushed~
```

```
  repeat until not lockSet(LockID)

  n:=0
  mask:=1

  repeat nAnalogI                                   Send Analog Values if needed
    if lastVal[n]
          sendVal( tstamp[n], n, lastVal[n]&$FFFF )
          lastVal[n]:=0
    n++
    mask<<=1

  lockClr(LockID)

  curVal:=DigIn~
  if curVal                                         Send Digital Values if needed
          sendDig(cnt,curVal&$7FFFFFFF)
```

- Messager.spin

Defines the methods to send, retrieve and interpret messages to and from PC.
Makes use of the Queue buffer.
Specifies BAUD rate value.
Instantiates 8 different “Queue”

```
obj
  Com   :      "FullDuplexSerial_rr004"
  Q     :      "Queue"
  Name  :      "Queue"
  Val   :      "Queue"
  Tmp   :      "Queue"
  Tmp2  :      "Queue"
  Waiting    :  "Queue"
  WaitingMsg :  "Queue"
  eIDQ  :      "Queue"


pub checkKeys
   if Read > 0                                      '(reads in the RxBuffer from Ser. Com)
     Parse                                          ' parses the buffer for keys.

pub Parse | beg,end,numChars,c,state, exec,  m, n, msgsum
```

{{ Parses the buffer for keys. }}

```
pub main
  ComCog:=Com.Start(31,30,%0000,BAUD)
  if  ComCog== 0
    repeat
      Com.str(String ("MCOG:"))
      Com.dec(ComCog)
    return 0
  repeat
    Com.str(String (" BadDownload "))

pub Start(callbackaddr, nametableaddr, nkeys)
```

- Queue.spin

Defines the functions for a rotating Queue buffer.

- AsmPwm.spin

Defines functions to manage, update and generate PWM Analog Output.
Contains the Assembly code to generate the PWM.

```
loop          ' main pwm loop
              mov       T,      onT             wz  '1 instruction = 4 cycles
              or        outa,   Mask                '1

:on     if_nz djnz      T,      #:on                '2

              mov       T,      offT            wz
              andn      outa,   Mask

:off    if_nz djnz      T,      #:off

              rdlong    Change, pChange         wz
          if_nz call    #reload

              jmp       #loop
```

Functions Implementation

This section describes how the main functions of the Propeller are implemented.

PC Communication

The communication between the device and the host PC is established through a USB link using the FT232RL USB to serial UART interface chip, which essentially translates the USB communication protocol into a classic 2-wires serial communication protocol and vice-versa. When connected to the PC, the PropDAQ will be recognised by the PC as a FTDI USB-to-serial-converter.

The communication protocol between the Propeller and an external computer is described in our PC Data Acquisition document. In short, every piece of information is wrapped in small packets in both directions. Each packet is structured like so:

<*name*:*values*> or **<*name*>**

where ***name*** is any arrangement of ASCII characters [excluding <:[!]>]

The following “names” or “commands” have been defined:

- version
  <version> Returns the version number of current firmware
- model
  <model> Returns the model number of current hardware (not used yet)
- nchannels
  <nchannels> Returns the total number of all channels (A/D/I/O)
- push
  <push> Identifies that the push-button has been activated (may become obsolete)
- set
  <set:#channel,value> Set the parameter value for the channel specified. Used to set sampling rate of AI or level (duty-factor) of AO (PWM).
- start
  <start:#channel> Start acquisition of specified channel
- stop
  <stop:#channel> Stop acquisition of specified channel
- dir
  <dir:#channel, value> Set the direction (0 input, 1 output) of specified digital channel
  Alternate: <dir:00000000> The 8-bit is used as a bit-mask to specify the direction of all digital channels at once.

- type

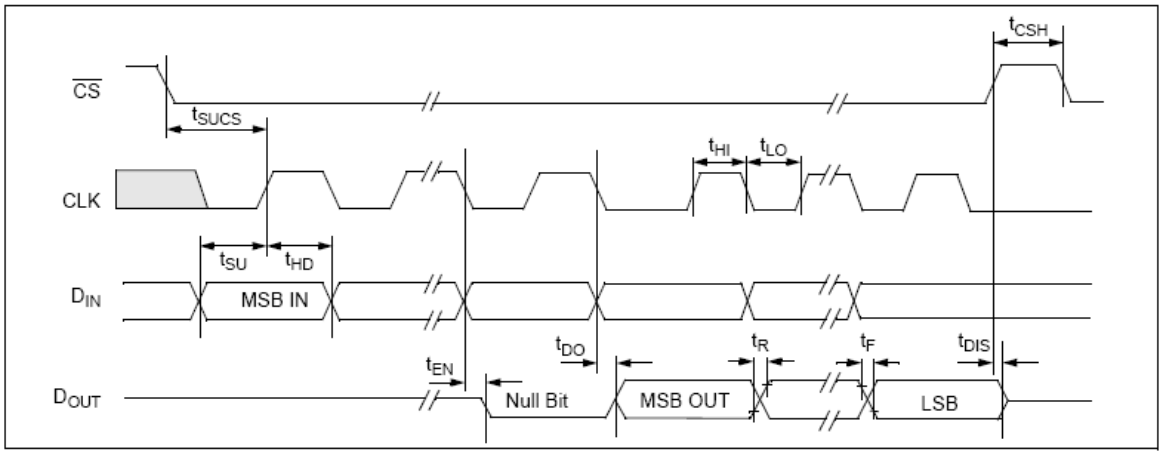
#channel refers to the channel number, from 0 to 13. The channels are as follow:

0 AI-0
1 AI-1
2 AI-2
3 AI-3
4 AO-0
5 AO-1
6 DI-0
7 DI-1
8 DI-2
9 DI-3
10 DO-0
11 DO-1
12 DO-2
13 DO-3

The communication speed is set at 115200 BAUDS, which is the maximum speed that the Propeller can handle with the FT232R chip.

ADC Communication

The second critical function is to read the value from the ADC chip. We are using the MCP3204 chip. The critical feature is the serial interface timing protocol form the ADC, illustrated below.



Serial interface timing from MCP3204.

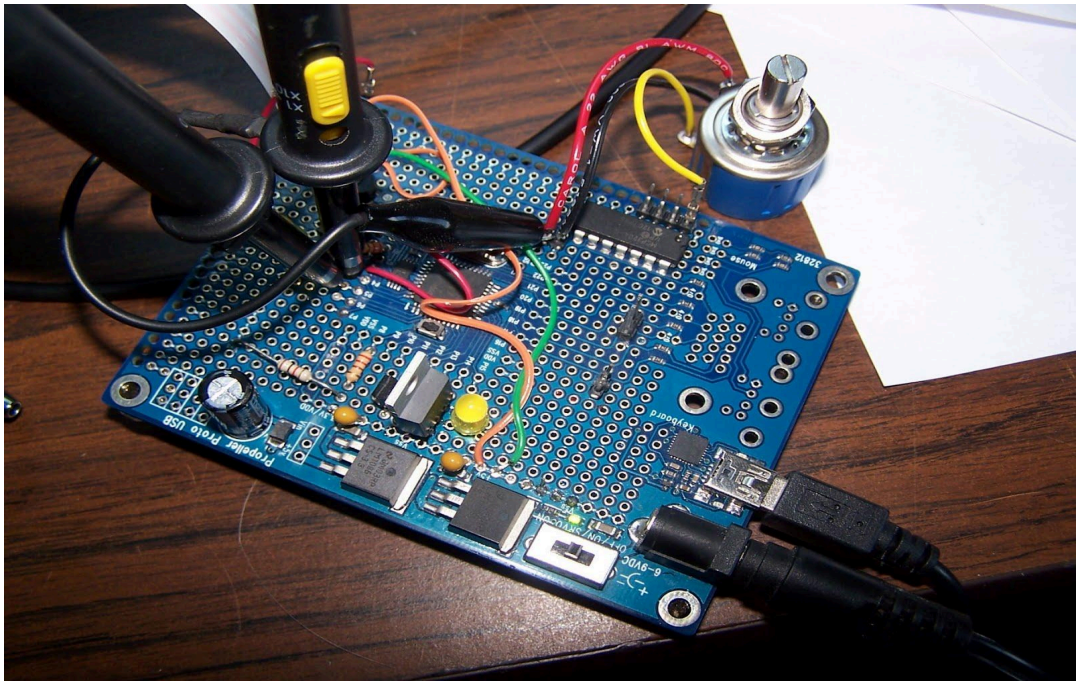
The MCP3204 is a 12-bit ADC capable of conversion rates of up to 100 ksps. Clock frequency can be up to 2.0MHz under 5V, and minimum Clock-High and Clock-Low time is 250 nsec each. The effective clock frequency needs to be at least 10 kHz to avoid linearity errors into the

conversion.

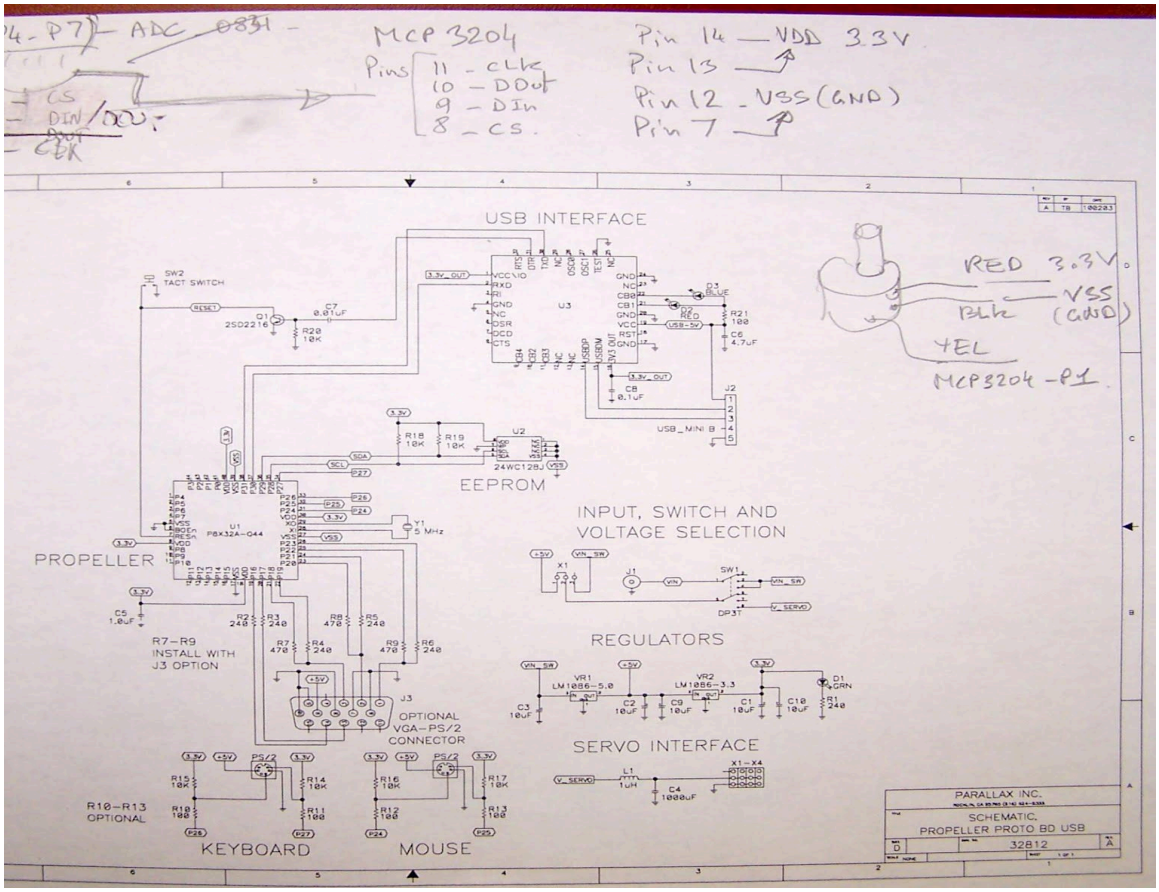
We first tested an object shared on a forum. It worked well, but could not be incorporated into our larger program due to conflicts between spin and assembly code.

We have developed our own Propeller code to read the values from the MCP3204 ADC chip. The code was written using Propeller Assembly language and a dedicated Cog. We can read values at a rate of approximately 55 kSample/sec (about 18-microsecond between samples).

The picture below illustrates our proto board with a Propeller chip and a MCP3204 chip. A simple potentiometer was used to send a variable voltage signal to one of the input channel of the MCP3204 chip.

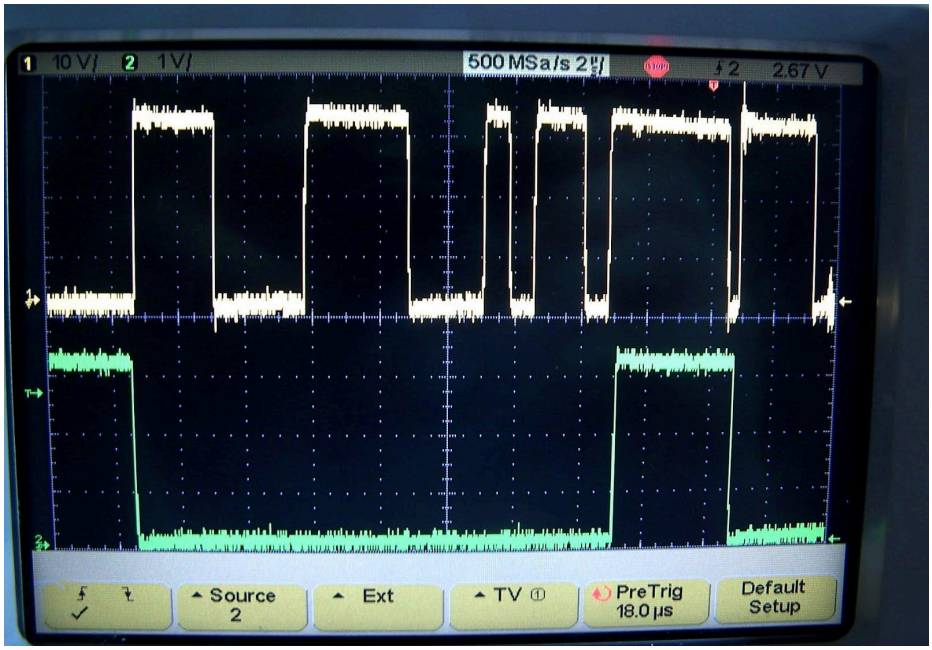


Picture of set-up with Propeller Proto Board and a MCP3204 connected to a simple potentiometer on CH0.



Schematic of Propeller Proto Board, with notes for connecting to MCP3204 and potentiometer.

ADC Testing



Oscilloscope traces of CS signal to MCP3204 (green) and DO signal to Propeller (yellow). Duration between consecutive falling edges of the green signal (time between successive samples) is about 18-microsecond, corresponding to a sampling rate of 55,555 samples per seconds, or about 55 ksps.

PWM Analog Output

We have developed our own code to provide a PWM output on one pin. The code was written using Propeller Assembly language and a dedicated Cog.