Buzz Wire Game

# Quick Overview

You get a timed lap of the Snetterton Racetrack with this game! The game will speak your player number and lap time and you may get on the leaderboard or make a new lap record.

When the game first powers up it checks the battery capacity and tells you to place the car at the left or right start point, if it’s not already there.

The game then says it’s initialised and suggests you press the yellow Help Button, which speaks some instructions. To start racing, you press the green Race Button, wait for the green light, lift the car clear of the start point and place it at the other end of the track.

The race is bidirectional, so leave the car at the end and this becomes the start point for the next player. During the game you’ll be told if you’re going too fast or too slow based on your crashes and time. At the end of the game your lap time is spoken and you may get some advice if you crashed too often or went too slow! If you get on the leaderboard of top 20 places, you’ll get some praise and if you get the lap record you’ll get a fanfare too.

If you start the race too soon, you’ll get a 10 second time penalty. You’ll also get a 3 second time penalty if you skid off course by touching the buzz wire. Correct your course quickly because the crash penalty is added every 0.5 seconds. “When” you crash you’ll hear a buzz tone and some car crashing sounds too. If your crash penalties add up to more than your driving time you’ll also get an extra 1 minute penalty for “stock car racing”.

If you’re not crashing too often during a race, you’ll hear some background music or sound effects played. During idle times different sounds are played, mostly with a car theme, which are interspersed with adverts every 90 seconds to encourage game play.

If you want a practice, just lift the handle and return it to an endpoint when finished. Unless you press the Race Button you won’t get a player number or lap time though.

Pressing the Leaderboard Button will speak out the top 4 player numbers and their lap times. The leaderboard records were cleared for today and the results are not lost when the game is switched off. The fastest lap of the day will receive the Piston Cup Trophy.

The battery lasts about 10 hours when the volume is high, or about a day if the volume is low.

# Technical Summary

The game uses an MCU (Microcontroller Unit), which is a low power computer chip as used in something like a washing machine. It has 4 Kbytes of program memory, 56 bytes of general purpose RAM and 256 bytes of permanent storage where the leaderboard is stored. The game almost filled of all these memory locations. The program is about 1,000 lines of code and is written in BASIC because it’s simple, fun and I’m old. The MCU is a PICAXE 20X2.

An audio module stores sound on an SD card, consisting of songs, effects and downloaded AI generated words or phrases. These sounds are strung together and controlled by a communication link to the microcontroller. The microcontroller has a timer and the electrical input and output pins for the buttons, buzz wire connections, buzz sound, LEDs (Light Emitting Diodes). The inputs use transient voltage suppression to protect the electronics against static electricity, up to 30 kV.

# Photos

|  |  |
| --- | --- |
| **Working Prototype** | **The Workbench** |
|  |  |
| **Handle V1 (Final was brazed)** | **1st Semblance** |
|  |  |
| **Some 30 Year Old Varnish** | **The Underneath** |
|  |  |

|  |  |
| --- | --- |
| **Button Panel** | **Assembled Components** |
|  |  |
| **Testing Australian Style** | **The Piston Cup Trophy** |
|  |  |
| **A Look From The Top** | **A Look Underneath** |
|  |  |
|  |  |

|  |
| --- |
| **It’s Finished!**  **(Until another 22 tracks were added!)** |
|  |

# The Circuit Diagram

The Electronics Schematic was produced in KiCad and the connectivity of all components was exported as a “netlist”, so that the circuit board design could be produced in VeroRoute.

A computer diagram with text and symbols

Description automatically generated with medium confidence

# Circuit Board

The game uses a type of circuit board, which is referred to as “Stripboard” these days. It has holes 0.1” apart, which early electronic chips used and this size spacing is still common today. The board has copper strips, which are easy to visualise and connect components together. Components are passed through the holes and soldered to the copper strips. Where you want to isolate components a hole is cut in the copper strip.

Some of the components I used are over 40 years old, from when I first played with electronics. These days it’s increasingly hard to use this type of layout because components are miniaturised to tiny sizes and then surface mounted on printed circuit boards.

A diagram of a computer

Description automatically generated

# Detailed Features

The game was going to be used in a charity event with a prize attached. So, for the first time ever, I created a list of detailed features for testing. This helped to perform a final code review in addition to identifying test cases. It’s challenging to test all the program states of the microcontroller, so both steps proved worthwhile. As a bonus the list helps to describe the game’s features too:

1. Initialisation, Battery Checks and Program Updates:
   1. The game tells you when it’s powered on or restarted.
   2. The battery voltage is checked at restart and between idle tracks.
   3. If either of the 2 battery cells are nearly exhausted (below 3,000mV) a warning will be spoken and the game will shut down into a low power state to prevent over discharge. If either cell is low (below 3,300mV) then a warning will be spoken between sound tracks when the game is idle, or directly after a game has finished, then the game continues as normal.
   4. The Reset Button only works during a restart and it clears the games stats.
   5. Program updates do not clear the game stats.
   6. After a restart the previous number of games is spoken and if games have been played the lap record is spoken.
   7. After a restart the handle must be placed at a starting position, if not it will speak a warning every 3 seconds before proceeding.
2. When the game is waiting to be played:
   1. Background music with a driving or fun theme is always played. There are several soundtracks played in a loop.
   2. Every 90 seconds an advert is played to encourage game play. These adverts play in a loop:
      1. “Roll-up, roll-up! It's only 50p a game. Have some fun and a chance to win the coveted Piston Cup Trophy!”
      2. “Most racing car legends start their careers with BUZZ WIRE RACING!  It's not too late to start now, so give it a try!”
      3. “Here is an important announcement: It's championship race day here at Snetterton. Get the fastest lap of the day and get that darn tooting trophy!”
      4. “Please, please, PLEASE have a game. I'm processing 4 million lines of code a second, so I'm kind of getting bored over here! “
      5. “Hey you! Yes you! Do you want some fun? It's a mere 50p for a chance to become a racing car legend!”
      6. “Come on, have a game! I promise you won't EVEN kill yourself if you crash.”
      7. “This game has a special 2 for 1 offer! You have some FUN! And your money will go to restoring the Village Sign TOO!”
      8. “Is there no one else! Is there no one else!”, a Brad Pitt extract from Troy.
3. Starting a practise:
   1. If the handle is raised from either start point without pressing the Race Button first, then a practise session is assumed. “Having a practise hey! There's no lap time for you.” is spoken, but other sound effects of a real game are still played.
   2. When the handle is returned to either endpoint, “Practise over and ready to race! Press the green button next time!” is played.
4. Starting a race:
   1. Pressing the Race Button only functions when the game is in idle mode, or while the help information is being spoken.
   2. The game is bidirectional and can be started at either start point, even if there’s been a practise and the previous race endpoint is changed.
   3. After pressing the Race Button a coin drop effect is played, then the player number is incremented and spoken, then starting beeps and the LED sequence (Red 1, Red 1 and 2, Red 1, 2 and 3, and then Green only) are synced to run in parallel.
   4. To allow for an accidental wiggle of the handle, the handle can lose contact with the start point for up to 100mS before a premature start is determined.
   5. If the player starts before the green light a 10 second penalty is created and a message “Time penalty! Started too soon!” is played.
5. During a race:
   1. Race themed background music starts with the game. There are several soundtracks in a loop and the next track is played if a game is started, or if a track finishes during a game.
   2. The player gets some feedback:
      1. The audio clip “Taking it slow huh. I can respect that” is played after 10 seconds if crash count is zero, or after 30 seconds if crash count is less than 2.
      2. The audio clip “Slow down!” is played after 10 seconds if crash count is more than 2, or after 30 seconds if crash count is more than 5.
      3. “You've been playing a while now, if there's someone waiting let them have a go to” is spoken after 90 seconds.
   3. Whenever the handle touches the buzz wire:
      1. There’s a buzz tone.
      2. The background sound is interrupted with a crash sound effect, the sound effect changes each time and ultimately loops.
      3. There’s a 3 second penalty, which is repeated in 0.5 seconds unless the handle is free from the buzz wire.
6. Cancelling the race:
   1. If the handle is returned to the start point during a race, then the race is cancelled and the next race will use the same player number. To debounce any handle wobbles during the race start phase, this event won’t be acknowledged until at least 2 seconds after the normal race start, or a premature race start.
   2. Then “Going back huh. I would have done the same” and “Press the Race Button to start again!” are spoken.
7. Finishing the race:
   1. If the handle touches the finish point:
      1. The green race light goes out and the log of player numbers is incremented and stored permanently.
      2. “You’ve finished!” is spoken.
      3. If the total crash penalty time is higher than the driving time an audio clip “Slow down stop the insanity!” is played and a cartoon voice states “You spent more time crashing than racing!” and a normal voice states “A 1-minute penalty is added for stock car racing!”.
      4. If the player’s adjusted lap time is not quick enough for a leaderboard entry then “Aarh you just missed out on the leaderboard! Have another try!” is spoken.
      5. If the player’s adjusted lap time achieves a leaderboard ranking their player number and lap time is inserted in the permanent memory of the leaderboard. All lower positions are shuffled downwards and eventually out of the leaderboard, which is limited to 20 places.
      6. If the player is ranked 1st the audio clip “A new record!” is played. If this isn’t the 1st player, then 1 of 4 random victory audio clips is played.
      7. If the player achieves a leaderboard ranking that is less than half the number of games played, the announcement “Here's a musical tribute just for you!” is spoken. A short fanfare is then played from a loop of tracks. Then a voice states their position on the leaderboard.
      8. If the crash count is zero then “No crashes. That's amazing for this crowd!” is spoken. If it’s not a lap record the audio clip “Wow all I can say is wow!” is played.
      9. If the crash count is greater than 9 “Unfortunately the car is a right off” is spoken.
      10. If there were any crashes, a voice states the number of crashes and how much time each crash cost.
      11. Then the lap time is spoken in seconds to 1 decimal place.
      12. Then “Waiting for next player!” is spoken and the game returns to idle/waiting mode.
8. Help Button:
   1. The help button only functions when the game is in idle mode, waiting to be played.
   2. This soundtrack is spoken: “This is a souped-up buzz wire game! The idea of this game is to get your race car around a lap of the racetrack as quickly as possible. If the car touches the metal tube you'll hear the car crashing, your parents worrying about the repair bill and most importantly time will be added to your lap of the race! To start a race, press the green button and wait for the green light. All you have to do is lift your car up and place it down at the other end of the track. The game will tell you your lap time and whether you've set a record for the day. Take it in turns to play. Have fun and Good luck!”.
   3. As an admin function, if the Help Button is held down and the Leaderboard Button pressed then the battery cell voltages will be spoken.
9. Speaking the Leaderboard:
   1. The Leaderboard Button only functions when the game is in idle mode, or while the help information is being spoken.
   2. If any games have been played an announcement starts with “The top N positions are”, where N is the number of games played up to the maximum of 4.
   3. Then from 1st to 4th place: the place, player number and lap time in tenths of a second are spoken.
   4. If the number of games is more than 4 the number of games played is spoken.
   5. Then “Press the green race button to beat them!” is played.

# The Program

Written in PICAXE BASIC, the program occupies 4,072 bytes of the 4,096 available on the 20X2 microcontroller. As more functionality, like the Leaderboard, was added some memory had to be saved by reducing the debug logging.

The program is organized into the following regions to improve quality:

1. Program Notes
2. Compiler Directives
3. Resources
   1. Pins
   2. Variables
   3. Constants
   4. Utilities
4. Initialisation
5. Main:
6. Interrupt:
7. Subroutines

Four software modules were written for the game, these are intended to be re-useable utilities for future projects:

1. "Debug\_Terminal.basinc" helps to easily display status information to a computer screen while the program is running.
2. "Timing.basinc" sets the processor clock rate and provides timing functions regardless of the clock rate.
3. "DF\_Player\_Mini.basinc" provides functions to easily control sound output from a DF Player Mini audio module mounted on the circuit board.
4. "Voltage.basinc" calculates the chip supply voltage and the input voltage levels of the battery cells.

# Program Listing – “Buzz Wire 2024-08-30.bas”

#region "Program Notes"

#rem

Buzz Wire Game - Snetterton Races, Created by Alan Hunt 2024

Date/Version: 2024-08-30

Author: Alan Hunt

Microcontroller Input and Output Pins:

Pin Usage PICAXE 20X2 Pinout & Functions Pin Usage

-------------------------------------------------------------------

+5V Supply |1 Vdd Gnd 20| 0V Supply

Programming Input |2 Serial In Serial out 19| Debugging Output

BATTERY (ADC3 VoltageBatt) |3 C.7(ADC3/Out/In) B.0(In/Out/ADC1/hint1) 18| AUDIO\_TX (to DF Player @T9600)

HelpButton |4 C.6(In) B.1(In/Out/ADC2/hint1/SRQ) 17| CELL (ADC2 for Cell 1 voltage)

LeaderboardButton |5 C.5(hpwmA/pvmC.5/Out/In) B.2(In/Out/ADC4/Comp2+) 16| AUDIO\_STATUS (DF Player low)

RaceButton |6 C.4(hwpmB/SRNQ/Out/In) B.3(In/Out/ADC5/Comp2-) 15| LED\_RED1 (1st "Get Ready")

InputPosL (Left Start Point) |7 C.3(hwpmC/ADC7/Out/In) B.4(In/Out/ADC6/hpwmD/Comp1-) 14| PWM\_BUZZ (Tone for Buzz Wire)

InputPosR (Right Start Point) |8 C.2(kbclk/ADC8/Out/In) B.5(In/Out/ADC10/hi2csda/hspisdi) 13| LED\_RED2 (2nd "Get Ready")

InputBuzz (Buzz Wire) |9 C.1(hspisdo/kbdata/ADC9/Out/In) B.6(In/Out/ADC11/hserin) 12| LED\_RED3 (3rd "Get Ready")

RESET\_BUTTON |10 C.0(hserout/Out/In) B.7(In/Out/hi2cscl/hspisck) 11| LED\_Green(Race "Go" light)

-------------------------------------------------------------------

The game has 7 digital inputs monitored on Port C.

5 of the inputs are monitored by software interrupts with "setint" (Only c.1 to c.5 permitted for 20M2 and 20X2).

The symbol "getInputs" (pinsC) is used to read PortC simultaneously and place it in the symbol "GameInputs" (variable b3).

Then bits of b3 are used to check specific inputs, like InputBuzz(bit25).

PICAXE 20X2 Notes:

\* Chip hardware is the Microchip PIC18F14K22 MCU. Power supply 1.8V to 5.5V. Default clock 8MHz.

- Pullup and pulldown input currents measured respectively as 1.5nA and 300pA on B.1 when set as a digital input.

- When an ADC channel is enabled it draws approximately 280 uA and the recommended source impedance is 20 kOhm.

- Input over/under voltage protection has a maximum 20mA clamp current.

- Serial comms is fixed to N, 8, 1 (No parity, 8 data bits and 1 stop bit).

- The ADC range is rail to rail, Vdd to 0V. In 10 bit mode, 1023 = Vdd, or more precisely Vin is greater than 1023/1024 of Vdd.

calibadc10 reads a 10 bit value for the 1.024V internal reference voltage. With Vdd=5.00V, vRef should be 210.

Vdd = 1024 \* 1.024 / calibadc10 = 1049 /calibadc10; or scaling tenfold for accuracy 10486 / calibadc10.

With Vdd=5.00V: 3Vin(5% capacity) is read as 629; 3.25V(15% battery) is 666 and 3.7V(50% battery) is 758.

\* Reminders:

- If you power down Vdd and have an ADC input connected to a voltage source, the ADC pin will try to power up the circuit again!

- Some functions may temporarily revert to default clock rate and multitasking also overides setfreq.

\* Game Settings:

- Clock rate is set to 16MHz to speed up programming but it helps with speeding up interupts and obtaining better timing accuracy too.

- Timer3 word value is set to increment every 131 mS (Prescale8:1 \* 65536 \* 4 / Clock), so very roughly divide 8 to get seconds.

Audio Module Notes:

\* The "DF Player Mini" supply is 3.2V to 5.0V. TTL is 3.3v level so a 1kOhm series must be placed on its Rx pin from 5V circuits.

\* Pause up to 4 seconds before use so that module has time to read file list.

\* See www.picaxe.com/docs/spe035.pdf and www.picaxe.com/docs/spe033.pdf

\* Command packet format is: $7E(Start), $FF(Version), $06(Length), $xx(Command), $00(No feedback), $xx(Param1), $xx(Param2), {Optional 2 byte checksum},$EF(End)

- The checksum is zero minus all byte values in message.

\* Various commands take time to respond, or need wait periods before another command is supplied.

\* The Busy signal only indicates audio output, there is a delay after requesting a track to be played.

Battery Notes:

The battery is 2 Li-ion cells in series.

The battery cells are Samsung INR18650-30Q, 3000mAhr, min cut-off 2.5V, planned cut-off 3.0V(about 5%).

OCV(OpenCircuitVoltage) can vary by as much as 500uV per degrees centigrade, so insignificant +-5mV from 15C to 35C dependent of SOC(StateOfCharge).

The circuit drain is highly dependant on volume, it runs for over a day with low to moderate volume. Maximum volume probably lasts about 8 hours.

#endrem

#endregion

#Region "Main Compiler Directives"

'Compiler Directives

#picaxe 20X2

#no\_data

'If "#no\_table" directive is defined then the eeprom/data command does not work. I'd expect this behaviour only with "#no\_data".

#no\_table

#no\_debug

#define DebugTerminal

#Endregion

#Region "Resources"

'Pins

symbol AUDIO\_TX = B.0 '@DF\_Player\_Mini.basinc: Serial Tx to audio module.

symbol CELL1 = pinB.1 '@Voltage.basinc: Cell 1 voltage monitoring is ADC2(B.1).

symbol AUDIO\_STATUS = pinB.2 '@DF\_Player\_Mini.basinc: Busy is active low after time lag (AUDIO\_IDLE and AUDIO\_BUSY).

symbol LED\_RED1 = B.3 '1st track light.

symbol PWM\_BUZZ = B.4 'Tone for Buzz Wire.

symbol LED\_RED2 = B.5 '2nd track light.

symbol LED\_RED3 = B.6 '3rd track light.

symbol LED\_GREEN = B.7 '4th "go" and running track light.

symbol RESET\_BUTTON = pinC.0 'Circuit board button to clear game stats if pressed during initialisation.

symbol HELP\_BUTTON = pinC.6 'Button to explain game.

symbol BATTERY = pinC.7 '@Voltage.basinc: Battery voltage monitoring is ADC3(C.7).

'Variables

symbol GameTime = timer 'Alias for the system "timer", measures elapsed game/idle time time in tenths of a second.

symbol UpTime = timer3 'Alias for "timer3", used for random number generation and uptime in 8th's of a second.

symbol \_DebugTerminalBinary = b0 '"Debug\_Terminal.basinc" utility is limited using b0 for Byte to ASCII bitstream conversion.

symbol InterruptSetting = b1 'Interrupt setting for input pins.

symbol IntSetBuzz = bit9

symbol IntSetPosR = bit10

symbol IntSetPosL = bit11

symbol IntSetRace = bit12

symbol IntSetLeaderboard = bit13

symbol InterruptMask = b2 'Interrupt mask for input pins.

symbol IntMaskBuzz = bit17

symbol IntMaskPosR = bit18

symbol IntMaskPosL = bit19

symbol IntMaskRace = bit20

symbol IntMaskLeaderBoard = bit21

symbol GameInputs = b3 'Game inputs on PortC with quick byte getter "getInputs" and bit checkers in b3.

symbol getInputs = pinsC

symbol InputBuzz = bit25

symbol InputPosR = bit26

symbol InputPosL = bit27

symbol RaceButton = bit28

symbol LeaderboardButton = bit29

symbol HelpButton = bit30

symbol GameStatus = b4 'Game Status has enumerated state constants

symbol GameDirection = b5

symbol SoundLoopIdle = b6 'Pointer for a list of tracks played when no game is running.

symbol SoundLoopBackground = b7 'Pointer for a list of tracks played as background music during the game.

symbol SoundLoopCrash = b8 'Pointer for a list of crash sounds that interupt the background music.

symbol SoundLoopFanfare = b9 'Pointer for a list of tracks played at the end of a game.

symbol SoundLoopAdvert = b10 'Pointer for a list of adverts played when the game is idle for too long.

symbol CrashCount = b11

symbol LapTime = w6 'Time in tenths of a second.

symbol PenaltyTime = w7 'Time added, in tenths, for a premature race start.

symbol LapRecord = w8

symbol PlayerNumber = w9

symbol PlayerRanking = b20

symbol GameTimerLog = b21

symbol GameTimer1 = w11

symbol GameTimer2 = w12

symbol GameTimer3 = w13

symbol VoltsCell2 = w14

symbol VoltsCell1 = w15

symbol BatteryCapacity = b32

'Internal variables for gosubs (Try and keep these unique due to ease Interrupt handling)

symbol \_LeaderBoard\_ResultPTR = b33 'For Gosub PlayerRank only.

symbol \_LeaderBoard\_TimePTR = b34 'For Gosubs PlayerRank and SayLeaderboard.

symbol \_LeaderBoard\_PlayerPTR = b35 'For Gosubs PlayerRank and SayLeaderboard.

symbol \_GameEndW = w18 'For Gosub GameEnd only.

symbol \_LeaderBoardW = w19 'For Gosubs PlayerRank and SayLeaderboard.

symbol \_LeaderBoardB = b40 'For Gosubs PlayerRank and SayLeaderboard.

'Only spare word is w21

'@DF\_Player\_Mini.basinc Variables

symbol \_Audio\_Digit = b41 'Assign symbol to any spare Byte.

symbol \_Audio\_Integer = w22 'Assign symbol to any spare Word.

symbol \_Audio\_Fraction = w23 'Assign symbol to any spare Word.

symbol \_Audio\_Track = b48 'Assign symbol to any spare Byte.

symbol \_Audio\_Byte = b49 'Assign symbol to any spare Byte.

'@Voltage.basinc Variables

symbol Voltage\_Vdd = w25 'Assign symbol to any spare Word. Stores power supply voltage to Picaxe chip in mV.

symbol \_ADCval = w26 'Assign symbol to any spare Word.

symbol \_VoltageW1 = w27 'Assign symbol to any spare Word.

'Constants

'For interrupts

symbol VAL\_DEFAULT = **%00000000** 'Interrupt value default

symbol MASK\_DEFAULT = **%00111110** 'Interrupt mask default

symbol NOT\_MASK\_DEFAULT = **%11000001**

symbol CONTACT = **1** 'Logic level for contact with buzz wire or end points.

symbol NO\_CONTACT = **0**

symbol PRESSED = **1** 'Logic level for button press.

symbol NOT\_PRESSED = **0**

'For GameStatus variable

symbol GAME\_NOT\_SET = **0** 'Not at a valid start point.

symbol GAME\_WAITING = **1** 'At a valid start point and waiting for Race or practise.

symbol GAME\_BEGINNING = **2** 'Race button pressed.

symbol GAME\_NOT\_STARTED = **3** 'Race Go signal was given but not moved off yet.

symbol GAME\_PLAYING = **4**

symbol GAME\_ENDING = **5** 'Reached end point and stating performance.

symbol GAME\_PRACTISE = **6** 'Moved off without pressing Race button.

'For GameDirection variable

symbol R\_TO\_L = **1**

symbol L\_TO\_R = **2**

'For Penalty time (tenths of a second)

symbol PENALTY\_PREMATURE\_START = **100**

symbol PENALTY\_CRASH = **30**

'Initialise Variables

symbol SOUND\_IDLE\_START = **139** 'MP3 140 to 176 play backgound music when no game is running.

symbol SOUND\_IDLE\_END = **177**

symbol SOUND\_BACKGROUND\_START = **198** 'MP3 198 to 218 play background music during the game.

symbol SOUND\_BACKGROUND\_END = **218**

symbol SOUND\_CRASH\_START = **1** 'ADVERT 1 to 9 play crash sounds that interupt the background music.

symbol SOUND\_CRASH\_END = **8**

symbol SOUND\_FANFARE\_START = **240** 'MP3 240 to 244 play different game endings for leaderboard entries.

symbol SOUND\_FANFARE\_END = **244**

symbol SOUND\_ADVERT\_START = **15** 'ADVERT 15 to 22 interuppt idle music to encourage game playing.

symbol SOUND\_ADVERT\_END = **22**

'Timer Values (tenths of a second)

symbol GAME\_TIME1 = **100** 'Some encouragement after 10S, or quit a failed race start.

symbol GAME\_TIME2 = **300** 'Some encouragement after 30S.

symbol GAME\_TIME3 = **900** 'Suggest you let someone else play after 90S.

symbol IDLE\_TIMER = **900** 'Play an advert to encourage playing every 90S.

'For EEPROM storage locations

symbol aGAME\_COUNT = **2**

symbol LEADERBOARD\_SIZE = **20** 'Limited to 20 to make speaking nth's easier.

symbol aLEADERBOARD\_PLAYER = **30** 'The Leaderboard player numbers occupy EEPROM addresses from here.

symbol aLEADERBOARD\_PLAYER\_END = **68** 'Player numbers are stored as words, the last byte used is this plus 1.

symbol aLEADERBOARD\_TIME = **70** 'Laptimes are stored as words from here.

symbol aLEADERBOARD\_TIME\_END = **108** 'The last byte is this +1.

symbol LEADERBOARD\_SPOKEN\_SIZE = **4** 'Number of entries spoken after LEADERBOARD\_BUTTON pressed (limited to 4 by track 120).

'For Battery Monitoring

symbol VOLTS\_CELL\_GOOD = **3300** 'Voltage in mV

symbol VOLTS\_CELL\_EXHAUSTED = **3000**

'@Timing.basinc Constants

#define CLOCK\_SET\_KHZ **16000**

#define CLOCK\_ARG M16

#define TIMER\_IN\_HUNDRETHS **10**

'@DF\_Player\_Mini.basinc Constants

symbol AUDIO\_VOLUME = **31** '0 to 31(Maximum).

symbol AUDIO\_BAUD = T9600\_16 'Tx logic is idle high (T) and 9600 bps, the last characters you adjust to your clock speed.

symbol AUDIO\_ACTIVE = **0** 'The AUDIO\_STATUS pin is active low, indicating audio playing.

symbol AUDIO\_IDLE = **1** 'AUDIO\_STATUS high indicates the module is idle, or seeking to play a track on the disk.

'@Voltage.basinc Constants

symbol V2C\_ADDR = **130** 'EEPROM Address for start of Voltage to Capacity Table.

symbol V2C\_MIN = **3000** 'Lowest voltage in Voltage to Capacity Table, below this zero is returned.

symbol V2C\_MAX = **4200** 'Highest voltage in Voltage to Capacity Table, after this 100 is returned.

symbol V2C\_INC = **100** 'Voltage difference to next address. NB: The table length is (Max-Min)/Inc.

'Utilities

#include "Debug\_Terminal.basinc"

#include "Timing.basinc"

#include "DF\_Player\_Mini.basinc"

#include "Voltage.basinc"

#Endregion

#Region "Programme Initialisation"

Init:

if PenaltyTime = PENALTY\_PREMATURE\_START then goto Main 'A premature start causes a Run 0

Pause300mS 'Allows PICAXE Editor time to open the terminal window.

DebugLine("@Initialising") 'Display values in PICAXE Editor Terminal if "TerminalDebug" is defined.

'Setup Pins

dirsB = **%11111001** 'Set port B7 to B0 pins (1=output).

dirsC = **%00000000** 'Ensure all C pins are inputs.

adcsetup = **%0000000000000110** 'Use ADC3(Pin 3, C.7) and ADC2 (Pin 17, B.1), this disables digital interface circuitry.

hpwm PWMDIV16, **0**, **0**, **%1000**, **255**, **511** 'Use PWM to produce lowest tone possible for a potential buzz wire contact sound.

'Setup Timers

tmr3setup **%10110001** 'Enables timer3 to generate randomness and for approx timing (8 per Sec, see notes).

'Initialise Variables

SoundLoopIdle = SOUND\_IDLE\_START 'MP3 150 to 172 play backgound music when no game is running.

SoundLoopBackground = SOUND\_BACKGROUND\_START 'MP3 200 to 212 play background music during the game.

SoundLoopCrash = SOUND\_CRASH\_START 'ADVERT 1 to 9 play crash sounds that interupt the background music.

SoundLoopFanfare = SOUND\_FANFARE\_START 'MP3 240 to 245 play different game endings.

SoundLoopAdvert = SOUND\_ADVERT\_START 'ADVERT 15 to 21 to encourage play when idle too long.

GameTimer1 = GAME\_TIME1 'Some encouragement after 10S, or quit a failed race start.

GameTimer2 = GAME\_TIME2 'Some encouragement after 30S.

GameTimer3 = GAME\_TIME3 'Suggest you let someone else play after 90S.

'Initialise Audio Module and check battery

gosub AudioInitialise

PlayMP3(**0062**,"Restarted") 'Play "The system has restarted"

gosub CheckBattery

'Reset game records during startup if necessary

if RESET\_BUTTON = PRESSED then

DebugLine("@Reset Stats")

PlayerNumber = **0**

write aGAME\_COUNT, WORD PlayerNumber

LapRecord = **0**

for PlayerRanking = aLEADERBOARD\_Player to aLEADERBOARD\_TIME\_END step **2**

write PlayerRanking, WORD PlayerNumber

next PlayerRanking

PlayMP3(**0091**,"game stats reset")

gosub AudioWaitForIdle

endif

'Speak the game stats

gosub SayGamesPlayed

gosub SayLapRecord

'Ensure the car is at one end of the track and prepare interrupt settings

gosub GameGetReady

InterruptSetting = VAL\_DEFAULT

InterruptMask = MASK\_DEFAULT

'Confirm Initialisation

PlayMP3(**0100**," Hello the game is ready")

PlayMP3(**0067**," Press Help or start now")

DebugLine("@Initialised")

debug 'Display values in PICAXE Editor if "no\_debug" is not defined.

#Endregion

#Region "Main"

Main:

'Contiunally loop around Main and see if GameTimers expire

setint NOT InterruptSetting, InterruptMask 'Set interupts for low state on C1 to C5 (BuzzWire, PosR, PosL, Race and Leaderboard)

if AUDIO\_STATUS = **1** then 'Check to see if nothing is playing

DebugText("@Main Without Audio - ")

gosub DebugGameStatus

select case GameStatus

case GAME\_NOT\_SET

PlayNow(**68**," Place car at 1 end of the track")

Pause10S

case GAME\_WAITING

gosub CheckBattery

gosub AudioWaitForIdle

DebugLineWithDecimal("Play Idle ", SoundLoopIdle, NoText)

PlayNow(SoundLoopIdle," IdleTrack") 'Play the next rotating idle track

inc SoundLoopIdle

if SoundLoopIdle > SOUND\_IDLE\_END then

SoundLoopIdle = SOUND\_IDLE\_START

endif

case GAME\_NOT\_STARTED, GAME\_PLAYING, GAME\_PRACTISE

DebugLineWithDecimal("Play Background ", SoundLoopBackground, NoText)

PlayNow(SoundLoopBackground," GameTrack") 'Play the next rotating game track

inc SoundLoopBackground

if SoundLoopBackground > SOUND\_BACKGROUND\_END then

SoundLoopBackground = SOUND\_BACKGROUND\_START

end if

else

'Other values GAME\_BEGINNING and GAME\_ENDING are temporary and should not occur.

endselect

endif

select case GameStatus

case GAME\_WAITING

if GameTime > IDLE\_TIMER then

PlayAdvert(SoundLoopAdvert,"Roll-up etc")

Gametime = **0**

inc SoundLoopAdvert

if SoundLoopAdvert > SOUND\_ADVERT\_END then

SoundLoopAdvert = SOUND\_ADVERT\_START

endif

endif

if HELP\_BUTTON = PRESSED then

DebugLine("Help Pressed")

GameTime = **0** 'Reset the timer for encouragement adverts

PlayNow(**0066**," GameIntro")

gosub AudioWaitForIdle

endif

case GAME\_BEGINNING

gosub GameBegin

case GAME\_NOT\_STARTED

'DebugTextWithDecimal("GameTime: ",GameTime)

'DebugLineWithDecimal(", GameTimer1: ",GameTimer1, NoText)

if GameTime > GameTimer1 then

DebugLine("Not started after 10S")

gosub GameQuit

PlayNow(**0102**, "Oh no it's nap time")

PlayMP3(**0101**, "Press the race button to start again")

gosub AudioWaitForIdle

endif

case GAME\_PLAYING

select case GameTime

case > GameTimer1

if CrashCount = **0** then

PlayAdvert(**11**,"Too slow")

elseif CrashCount > **2** then

PlayAdvert(**14**, "Slow down")

endif

GameTimer1 = **65535**

case > GameTimer2

if CrashCount < **2** then

PlayAdvert(**11**,"Too slow")

elseif CrashCount > **5** then

PlayAdvert(**14**, "Slow down")

endif

GameTimer2 = **65535**

case > GameTimer3

PlayNow(**0069**, "You've been playing a while, let someone else too")

GameTimer3 = **65535**

endselect

endselect

#Endregion

Pause10mS

goto Main

#Region "Interrupts"

**Interrupt:**

GameInputs = getInputs

DebugLine("@Interupt")

gosub DebugGameStatus

gosub DebugGameInputs

select case GameStatus

case GAME\_NOT\_SET

if InputPosL = CONTACT OR InputPosR = CONTACT then 'Car now placed correctly at one end of the track.

GameStatus = GAME\_WAITING

if InputPosL = CONTACT then

GameDirection = L\_TO\_R

else

GameDirection = R\_TO\_L

endif

PlayNow(**0065**," Thank you")

endif

case GAME\_WAITING

if InputPosL = NO\_CONTACT AND InputPosR = NO\_CONTACT then 'Car moving without Race button, so it's a practise.

GameStatus = GAME\_PRACTISE

SoundLoopCrash = SOUND\_CRASH\_START

CrashCount = **0**

PlayNow(**0094**,"having a practise hey. There's no laptime for you")

endif

if LeaderboardButton = PRESSED then

if HelpButton = PRESSED then

gosub AudioClear

gosub CheckBattery 'If both pressed then state battery cell voltages

PlayNow(**0112**, "The cell voltages are")

AudioSpeakNumber(VoltsCell1)

PlayMP3(**0130**, "millivolts")

PlayMP3(**0060**, "and")

AudioSpeakNumber(VoltsCell2)

PlayMP3(**0130**, "millivolts")

else

gosub SayLeaderboard

endif

GameTime = **0** 'Reset the timer for encouragement adverts

endif

if RaceButton = PRESSED then

GameStatus = GAME\_BEGINNING

endif

case GAME\_BEGINNING

if InputPosL = NO\_CONTACT AND InputPosR = NO\_CONTACT then 'Car moved too early while game still begining (before green light).

Pause100ms 'Pause a while to debounce a player wiggling the handle.

GameInputs = getInputs

if InputPosL = NO\_CONTACT AND InputPosR = NO\_CONTACT then 'Car is really moving too early, so start the game and log the penalty.

GameStatus = GAME\_PLAYING

'DebugLine("Started too Soon")

PlayNow(**0097**,"Time penalty. Started too soon.")

low LED\_RED1, LED\_RED2, LED\_RED3

high LED\_GREEN

GameStatus = GAME\_PLAYING

GameTimer1 = GAME\_TIME1

GameTimer2 = GAME\_TIME2

GameTimer3 = GAME\_TIME3

SoundLoopCrash = SOUND\_CRASH\_START

CrashCount = **0**

PenaltyTime = PENALTY\_PREMATURE\_START

GameTime = **0**

IntSetPosL = InputPosL

IntSetPosR = InputPosR

IntSetBuzz = InputBuzz

run **0**

endif

endif

case GAME\_NOT\_STARTED

if InputPosL = NO\_CONTACT AND InputPosR = NO\_CONTACT then 'Car moving correctly after given the green light.

DebugLine("Started OK")

GameStatus = GAME\_PLAYING

endif

case GAME\_PLAYING

if InputPosL = CONTACT then 'Car touching left end of the track.

if GameDirection = R\_TO\_L then

gosub GameEnd

else

if GameTime > **20** then 'debounce a return to start

PlayNow(**0090**," Going back huh")

PlayMP3(**0101**," Press the race button to start again")

gosub AudioWaitForIdle

gosub GameQuit

endif

endif

endif

if InputPosR = CONTACT then 'Car touching right end of the track.

if GameDirection = L\_TO\_R then

gosub GameEnd

else

if GameTime > **20** then 'debounce a return to start

PlayNow(**0090**," Going back huh")

PlayMP3(**0101**," Press the race button to start again")

gosub AudioWaitForIdle

gosub GameQuit

endif

endif

endif

if InputBuzz = CONTACT then 'Buzz wire contact

gosub GameCrash

endif

case GAME\_PRACTISE

if InputPosL = CONTACT or InputPosR = CONTACT then 'Car finished practising on the track.

GameStatus = GAME\_WAITING

if InputPosL = CONTACT then

GameDirection = L\_TO\_R

else

GameDirection = R\_TO\_L

endif

PlayNow(**0099**," Practise over, press green button")

endif

if InputBuzz = CONTACT then 'Buzz wire contact

gosub GameCrash

endif

endselect

IntSetPosL = InputPosL

IntSetPosR = InputPosR

DebugText("@Interupt End - ")

gosub DebugGameStatus

setint NOT InterruptSetting, InterruptMask

return

#Endregion

#Region "Subroutine GameGetReady"

GameGetReady:

DebugLine("@GameGetReady")

GameInputs = getInputs

gosub DebugGameInputs

if InputPosL = NO\_CONTACT and InputPosR = NO\_CONTACT then

do until InputPosL = CONTACT or InputPosR = CONTACT 'Ensure car is at 1 end of the track

PlayMP3(**68**,"Place car at 1 end of the track")

Pause3S

GameInputs = getInputs

loop

PlayNow(**0065**," Thank you")

endif

if InputPosL = CONTACT then

GameDirection = L\_TO\_R

else

GameDirection = R\_TO\_L

end if

IntSetPosL = InputPosL

IntSetPosR = InputPosR

GameStatus = GAME\_WAITING

return

#Endregion

#Region "Subroutine GameBegin"

GameBegin:

DebugLine("@GameBegin")

GameStatus = GAME\_BEGINNING

read aGAME\_COUNT, WORD PlayerNumber

inc PlayerNumber 'Don't flash the PlayerNumber increase until a successful end.

PlayNow(**180**," Arcade token")

PlayMP3(**108**," Player")

AudioSpeakNumber(PlayerNumber)

PlayMP3(**109**," Wait for the green light")

PlayMP3(**186**," Mario Kart beeps")

high LED\_RED1

Pause1S

high LED\_RED2

Pause1S

high LED\_RED3

Pause1S

low LED\_RED1, LED\_RED2, LED\_RED3

high LED\_GREEN

GameStatus = GAME\_NOT\_STARTED

GameTimer1 = GAME\_TIME1

GameTimer2 = GAME\_TIME2

GameTimer3 = GAME\_TIME3

SoundLoopCrash = SOUND\_CRASH\_START

CrashCount = **0**

PenaltyTime = **0**

GameTime = **0**

return

#Endregion

#Region "Subroutine GameCrash"

GameCrash:

DebugLine("@GameCrash")

PlayAdvert(SoundLoopCrash, "Crash") 'There is a timed delay in PlayAdvert, which helps as a crash rate limiter.

if CrashCount < **255** then

inc CrashCount

inc SoundLoopCrash

if SoundLoopCrash > SOUND\_CRASH\_END then

SoundLoopCrash = SOUND\_CRASH\_START

endif

endif

DebugLineWithDecimal("CrashCount=", CrashCount, NoText)

return

#Endregion

#Region "Subroutine GameQuit"

GameQuit:

DebugLine("@GameQuit")

low LED\_RED1, LED\_RED2, LED\_RED3, LED\_GREEN

gosub GameGetReady

GameTime = **0**

return

#Endregion

#Region "Subroutine GameEnd"

GameEnd:

DebugLine("@GameEnd")

'Record Game and notify of GameEnd

GameStatus = GAME\_ENDING

LapTime = GameTime

low LED\_GREEN

gosub AudioClear

write aGAME\_COUNT, WORD PlayerNumber

PlayNow(**71**," You finished!")

'Add 1 minute penalty if crash time is greater than laptime.

\_GameEndW = PENALTY\_CRASH \* CrashCount

if \_GameEndW > Laptime then

PenaltyTime = PenaltyTime + **600**

PlayMP3(**231**," Slow down stop the insanity!")

PlayMP3(**105**," 1 minute penalty because crashtime > laptime")

endif

'Calculate laptime and penalty time (measured in tenths of a second)

DebugTextWithDecimal("GameTime=", Laptime)

DebugTextWithDecimal(", CrashTime=", \_GameEndW)

PenaltyTime = \_GameEndW + PenaltyTime

LapTime = LapTime + PenaltyTime

DebugLineWithDecimal(", Adjusted=", LapTime, NoText)

'Leaderboard ranking and associated messages.

gosub PlayerRank

select case PlayerRanking

case **0**

PlayMP3(**131**," Ah you just missed out on the leaderboard, have another try!")

case **1**

LapRecord = LapTime

PlayMP3(**0098**," New lap record")

\_GameEndW = UpTime % **4**

\_GameEndW = \_GameEndW + **220**

PlayMP3(\_GameEndW," Winners track")

gosub PlayFanfare

else

gosub PlayFanfare

endselect

'Speak about crashes

if CrashCount = **0** then

PlayMP3(**88**," No crashes!")

if LapTime <> LapRecord then

PlayMP3(**232**," Wow all I can say is wow!")

endif

PlayMP3(**73**," Your time was")

else

if CrashCount > **9** then

PlayMP3(**89**," The car is a right off.")

endif

PlayMP3(**86**," You crashed")

AudioSpeakNumber(CrashCount)

PlayMP3(**87**," times & each cost")'

AudioSpeakNumber(PENALTY\_CRASH/**10**)

PlayMP3(**75**," seconds")

PlayMP3(**103**," Your adjusted time was")

endif

'Speak the laptime seconds

AudioSpeakTenths(LapTime)

PlayMP3(**75**," seconds")

If PlayerRanking <> **1** then gosub SayLapRecord

gosub GameQuit

PlayMP3(**0107**," Waiting for next player!")

return

#Endregion

#Region "Subroutine SayGamesPlayed"

SayGamesPlayed:

read aGAME\_COUNT, WORD PlayerNumber

DebugLineWithDecimal("@SayGamesPlayed: ", PlayerNumber, NoText)

PlayMP3(**0084**," There have been")

AudioSpeakNumber(PlayerNumber)

PlayMP3(**0085**," games")

return

#Endregion

#Region "Subroutine SayLeaderboard"

SayLeaderBoard:

read aGAME\_COUNT, WORD PlayerNumber

DebugLine("@SayLeaderBoard")

if PlayerNumber = **0** then

'@No games yet

gosub AudioClear

gosub sayGamesPlayed

else

'@Speak Leaderboard

PlayNow(**0127**, "The top")

If PlayerNumber > LEADERBOARD\_SPOKEN\_SIZE then

PlayMP3(LEADERBOARD\_SPOKEN\_SIZE, "n(spoken) places")

else

PlayMP3(PlayerNumber, "n(games)places")

endif

PlayMP3(**123**," leaderboard positions are:")

\_LeaderBoard\_PlayerPTR = aLEADERBOARD\_PLAYER

\_LeaderBoard\_TimePTR = aLEADERBOARD\_TIME

\_LeaderBoard\_ResultPTR = **0**

do

read \_LeaderBoard\_PlayerPTR, WORD \_LeaderBoardW

if \_LeaderBoardW = **0** then exit 'Quit leaderboard if the entry isn't filled yet.

\_LeaderBoardB = **117** + \_LeaderBoard\_ResultPTR 'Track 0117 to 0120 is "1st place" to "4th place".

PlayMP3(\_LeaderBoardB," xx place.")

PlayMP3(**0108**," Player")

AudioSpeakNumber(\_LeaderBoardW)

read \_LeaderBoard\_TimePTR, WORD \_LeaderBoardW

PlayMP3(**0122**," with laptime")

AudioSpeakTenths(\_LeaderBoardW)

PlayMP3(**75**," seconds")

inc \_LeaderBoard\_ResultPTR

\_LeaderBoard\_PlayerPTR = \_LeaderBoard\_PlayerPTR + **2**

\_LeaderBoard\_TimePTR = \_LeaderBoard\_TimePTR + **2**

loop until \_LeaderBoard\_ResultPTR = LEADERBOARD\_SPOKEN\_SIZE 'Only speak the first n entries.

if PlayerNumber > LEADERBOARD\_SPOKEN\_SIZE then gosub sayGamesPlayed 'Speak the game count if more than the spoken records.

PlayMP3(**129**, "Press the green button and try to beat them")

endif

GameTime = **0**

return

#Endregion

#Region "Subroutine SayLapRecord"

SayLapRecord:

read aLEADERBOARD\_TIME, WORD LapRecord

DebugLineWithDecimal("@SayLapRecord: ",LapRecord, NoText)

if LapRecord > **0** then

PlayMP3(**0093**," The lap record is")

AudioSpeakTenths(LapRecord)

PlayMP3(**75**," seconds")

endif

return

#Endregion

#Region "Subroutine PlayFanfare"

PlayFanfare:

PlayMP3(**125**, "Congratulations you are")

\_GameEndW = **32** + PlayerRanking

PlayMP3(\_GameEndW, "nth")

PlayMP3(**124**, "on the leaderboard")

\_GameEndW = PlayerNumber / **2**

if PlayerRanking <= \_GameEndW then

PlayMP3(**115**," Musical tribute just for you.")

PlayMP3(SoundLoopFanfare," Fanfare")

inc SoundLoopFanfare

if SoundLoopFanfare > SOUND\_FANFARE\_END then

SoundLoopFanfare = SOUND\_FANFARE\_START

endif

endif

return

#Endregion

#Region "Subroutine PlayerRank"

PlayerRank:

'DebugLine("@PlayerRank")

'@Work down the leaderboard to determine ranking from the 1st beaten or blank time entry.

\_LeaderBoard\_ResultPTR = aLEADERBOARD\_TIME

do

read \_LeaderBoard\_ResultPTR, WORD \_LeaderBoardW

if \_LeaderBoardW = **0** OR LapTime < \_LeaderBoardW then exit

inc \_LeaderBoard\_ResultPTR

inc \_LeaderBoard\_ResultPTR

loop until \_LeaderBoard\_ResultPTR > aLEADERBOARD\_TIME\_END

'@Chose action based on ranking.

select case \_LeaderBoard\_ResultPTR

case = aLEADERBOARD\_TIME\_END

'@Last entry on the leaderboard.

PlayerRanking = aLEADERBOARD\_TIME\_END - aLEADERBOARD\_TIME / **2** + **1**

write aLEADERBOARD\_PLAYER\_END, WORD PlayerNumber

write aLEADERBOARD\_TIME\_END, WORD LapTime

'DebugLineWithDecimal("RankLast=",PlayerRanking,NoText)

case > aLEADERBOARD\_TIME\_END

'@Not on the leaderboard.

PlayerRanking = **0**

'DebugLine("Rank=None")

return

else

'@Other placing on the leaderboard.

PlayerRanking = \_LeaderBoard\_ResultPTR - aLEADERBOARD\_TIME / **2** + **1**

'DebugLineWithDecimal("RankElse=",PlayerRanking,NoText)

'@Work up the leaderboard to shuffle slower entries down.

\_LeaderBoard\_PlayerPTR = aLEADERBOARD\_PLAYER\_END - **2**

\_LeaderBoard\_TimePTR = aLEADERBOARD\_TIME\_END - **2**

do

'Move time entry down

read \_LeaderBoard\_TimePTR, WORD \_LeaderBoardW

\_LeaderBoard\_TimePTR = \_LeaderBoard\_TimePTR + **2**

write \_LeaderBoard\_TimePTR, WORD \_LeaderBoardW

\_LeaderBoard\_TimePTR = \_LeaderBoard\_TimePTR - **4**

'Move player entry down

read \_LeaderBoard\_PlayerPTR, WORD \_LeaderBoardW

\_LeaderBoard\_PlayerPTR = \_LeaderBoard\_PlayerPTR + **2**

write \_LeaderBoard\_PlayerPTR, WORD \_LeaderBoardW

\_LeaderBoard\_PlayerPTR = \_LeaderBoard\_PlayerPTR - **4**

loop until \_LeaderBoard\_TimePTR < \_LeaderBoard\_ResultPTR

'@Insert new leaderboard entry

\_LeaderBoard\_PlayerPTR = \_LeaderBoard\_PlayerPTR + **2**

write \_LeaderBoard\_PlayerPTR, WORD PlayerNumber

\_LeaderBoard\_TimePTR = \_LeaderBoard\_TimePTR + **2**

write \_LeaderBoard\_TimePTR, WORD LapTime

endselect

'DebugLineWithDecimal("Rank=", PlayerRanking, NoText)

#rem'For Debug output only:

for \_LeaderBoard\_PlayerPTR = aLEADERBOARD\_PLAYER\_END to aLEADERBOARD\_PLAYER STEP -2

'DebugTextWithDecimal("PlayerRankPTR=",\_LeaderBoard\_PlayerPTR)

read \_LeaderBoard\_PlayerPTR, WORD \_LeaderBoardW

DebugTextWithDecimal("Player=",\_LeaderBoardW)

\_LeaderBoard\_TimePTR = \_LeaderBoard\_PlayerPTR + aLEADERBOARD\_TIME - aLEADERBOARD\_PLAYER

read \_LeaderBoard\_TimePTR, WORD \_LeaderBoardW

DebugLineWithDecimal(", Time=",\_LeaderBoardW,NoText)

next \_LeaderBoard\_PlayerPTR

#endrem

return

#EndRegion

#Region "Subroutine CheckBattery"

CheckBattery:

GetVdd

VoltsByVdd(VoltsCell2,**3**) 'Get half battery voltage on ADC3 and store in VoltsCell2 as mV.

VoltsByVdd(VoltsCell1,**2**) 'Get cell voltage on ADC2 and store in VoltsCell1 as mV.

VoltsCell2 = VoltsCell2 \* **2** - VoltsCell1 'Convert ADC3 input to Cell2 voltage.

'DebugLineWithDecimal("Vdd: ", Voltage\_Vdd,"mV")

'DebugLineWithDecimal("Cell 2: ", VoltsCell2,"mV")

'DebugLineWithDecimal("Cell 1: ", VoltsCell1,"mV")

if VoltsCell2 > VOLTS\_CELL\_GOOD AND VoltsCell1 > VOLTS\_CELL\_GOOD then

'DebugLine("Battery cells good")

else

if VoltsCell2 < VOLTS\_CELL\_EXHAUSTED OR VoltsCell1 < VOLTS\_CELL\_EXHAUSTED then

'DebugLine("Battery cells exhausted")

PlayMP3(**111**,"System shutting down due to low battery")

sleep **0** 'Enter low power state, essentially off.

else

'DebugLine("Battery cells low")

PlayMP3(**110**,"Warning, battery is below 20%")

endif

endif

return

#Endregion

#Region "Subroutine DebugGameStatus"

#ifndef DebugTerminal

DebugGameStatus:

Return

#else

DebugGameStatus:

DebugText("GameStatus: ")

select case GameStatus

case GAME\_NOT\_SET

DebugText("NotSet")

case GAME\_WAITING

DebugText("Waiting")

case GAME\_BEGINNING

DebugText("Beginning")

case GAME\_NOT\_STARTED

DebugText("NotStarted")

case GAME\_PLAYING

DebugText("Playing")

case GAME\_ENDING

DebugText("Ending")

case GAME\_PRACTISE

DebugText("Practising")

endselect

DebugText(", Direction: ")

select case GameDirection

case L\_TO\_R

DebugLine("Left to Right")

case R\_TO\_L

DebugLine("Right to Left")

else

DebugLine("Not Set")

endselect

return

#endif

#Endregion

#Region "Subroutine DebugGameInputs"

#ifndef DebugTerminal

DebugGameInputs:

Return

#else

DebugGameInputs:

DebugText("GameInputs: ")

if GameInputs = **0** then

DebugLine("Nothing(Playing)")

else

if InputBuzz = CONTACT then

DebugText("BuzzWire ")

endif

if InputPosR = CONTACT then

DebugText("Right ")

endif

if InputPosL = CONTACT then

DebugText("Left ")

endif

if RaceButton = PRESSED then

DebugText("RaceButton ")

endif

if HelpButton = PRESSED then

DebugText("HelpButton")

endif

DebugLine(NoText)

endif

return

#endif

#Endregion

# Program Listing – “Debug\_Terminal.basinc”

#REM

File: "Debug Terminal.basinc", By Alan Hunt 2024

Date/Version: 2024-07-03

Author: Alan Hunt

This file provides easy to use functions based on sertxd. The functions can show detailed debugging information with descriptive text and any specific values, or just position information like 'DebugLine("@Interrupt")'. The debugging can also be turned on or off easily and when turned off there is no programe space used and no serial transmissions, so no performance delays and the temporary disabling of software interuppts.

Note, Picaxe debugging is typically performed with the sertxd command, which is used to send messages and values to the Picaxe Editor Terminal window, using the same connection as the programming interface. The debug command can also be used to copy all Picaxe status, but this is a large transfer without human readable status, or even program position.

The utilities are dependent on defining a compiler directive and a symbol before the inclusion of this file. The utilities are turned on or off with "#define DebugTerminal" and the symbol \_DebugTerminalBinary must be defined as variable b0. The following program structure is recommended:

1) Program Notes

2) Compiler Directives

3) Resources

3a) Pins

3b) Variables

3c) Constants

3d) Utilities

4) Initialisation

5) Main:

6) Interrupt:

7) Subroutines

An example configuration is shown below:

#Region "Compiler Directives"

#picaxe 20X2

#no\_data

#no\_table

#no\_debug

#define DebugTerminal 'Comment out this line to disable Debug Termnial.

#Endregion

#Region "Resources"

'Pins

'Variables

symbol \_DebugTerminalBinary = b0 '"Debug\_Terminal.basinc" utility is limited using b0 for Byte to ASCII bitstream conversion.

'Constants

'Utilities

#include "Debug Terminal.basinc"

#Endregion

#EndREM

#Region "Terminal Debug Utilities"

'Check DebugTerminal is defined

#ifndef DebugTerminal

'If "DebugTerminal" not defined then utilities become comments to save serial delays and program space.

#define DebugLine(msg) '"DebugLine" includes CR,LF

#define DebugLineWithDecimal(msg,val,msg2) 'Byte or Word accepted here for val

#define DebugLineWithBinary(msg,val,msg2) 'Byte shown as 2 nibbles, so that input values are clear

#define DebugText(msg) 'msg and msg2 can be a string in quotes, or NoText

#define DebugTextWithDecimal(msg,val) '

#define DebugTextWithBinary(msg,val) '

#else

#define DebugLine(msg) sertxd (msg,cr,lf)

#define DebugLineWithDecimal(msg,val,msg2) sertxd (msg,#val,msg2,cr,lf)

#define DebugLineWithBinary(msg,val,msg2) \_DebugTerminalBinary=val : sertxd (msg,#bit7,#bit6,#bit5,#bit4," ",#bit3,#bit2,#bit1,#bit0,msg2,cr,lf)

#define DebugText(msg) sertxd (msg)

#define DebugTextWithDecimal(msg,val) sertxd (msg,#val)

#define DebugTextWithBinary(msg,val) \_DebugTerminalBinary=val : sertxd (#bit7,#bit6,#bit5,#bit4," ",#bit3,#bit2,#bit1,#bit0)

symbol NoText = **3** 'In PICAXE Serial Terminal untick the option "Display non-ASCII as Hex"

#endif

#Endregion

# Program Listing – “Timing.basinc”

#REM

File: "Debug Terminal.basinc"

Date/Version: 2024-07-09a

Author: Alan Hunt

Description:

This file provides 3 core features:

1) It provides functions like "Pause100mS" that work consistently regardless of clock rate.

2) It sets the processor clock rate and adjusts the terminal rate accordingly.

3) For X2 chips it sets the chip "timer" variable to increment at the rate specified in hundreths of seconds.

Limitations

Parameters for CLOCK\_SET\_KHZ and CLOCK\_ARG have chipset limitations as defined in PICAXE documentation, see Manual 2, setfreq. For reference, common values for CLOCK\_ARG are K31, K250, K500, M1, M2, M4, M8, M16, M32 and M64, where K=kHz and M=MHz.

Due to integer maths, the timer rate for X2 chips will become more inaccurate as clock speed is reduced.

The utility was briefly used with the 20M2 chip and almost always used with a 20X2 running at 16MHz. Please treat other configurations with caution and let me have improvements that don't distract from the relative simplicity.

Programming Guide:

The "#define" pre-processor directives shown below are defined as constants before the inclusion of this file. An example configuration is shown below:

#Region "Resources"

'Constants

'@Timing.basinc Constants

#define CLOCK\_SET\_KHZ 16000

#define CLOCK\_ARG M16

#define TIMER\_IN\_HUNDRETHS 10

'Utilities

#include "Timing.basinc"

#Endregion

#EndREM

#Region "Timing Utilities"

'Determine Default Clock Rate in kHz

'===================================

#ifdef \_20X2

#define CLOCK\_DEFAULT\_KHZ **8000**

#elseifdef \_28X2

#define CLOCK\_DEFAULT\_KHZ **8000**

#elseifdef \_40X2

#define CLOCK\_DEFAULT\_KHZ **8000**

#else

#define CLOCK\_DEFAULT\_KHZ **4000**

#endif

'Determine TerminalRate, TickRate and ClockRatio

'===============================================

'NB: TerminalRate is the bit rate for the Serial Out pin, as used for debugging and the sertxd command.

'NB: TickRate is the incrementing rate for the system "timer" variable.

'NB: ClockRatio is a pre-processor symbol used as a multiplier/divisor in timing statements.

#if CLOCK\_SET\_KHZ >= CLOCK\_DEFAULT\_KHZ

'Determine Clock Multiplier, set Terminal Rate and define minor tick rate for settimer.

symbol ClockMultiplier = CLOCK\_SET\_KHZ / CLOCK\_DEFAULT\_KHZ

symbol TerminalRate = **9600** \* ClockMultiplier

symbol MinorTickuS = **32** / ClockMultiplier 'Minor ticks are 256/Clock (8Mhz=32uS, 16MHz=16uS).

symbol MinorTicksIn\_10mS = **10000** / MinorTickuS

symbol TicksRequired = TIMER\_IN\_HUNDRETHS \* MinorTicksIn\_10mS

symbol TickToSet = **65536** - TicksRequired

#define ClockRatio \* ClockMultiplier

#else

symbol ClockDivider = CLOCK\_DEFAULT\_KHZ / CLOCK\_SET\_KHZ

symbol TerminalRate = **9600** / ClockDivider

symbol minorTickuS = **32** \* ClockDivider 'Minor ticks are 256/Clock (8Mhz=32uS).

symbol MinorTicksIn\_10mS = **10000** / MinorTickuS

symbol TicksRequired = TIMER\_IN\_HUNDRETHS \* MinorTicksIn\_10mS

symbol TickToSet = **65536** - TicksRequired

#define ClockRatio / ClockDivider

#endif

'Set Clock speed and terminal rate

'=================================

setfreq CLOCK\_ARG

#terminal TerminalRate

'If an X2 chip then set the timer rate

'=====================================

#if CLOCK\_DEFAULT\_KHZ = **8000**

settimer TickToSet 'Sets incrementing rate for the system "timer" word variable.

'For example, at 16MHz the "timer" word value increments every 0.1S with a setting of 59,286 (65,536-(100,000/minorTickuS).

#endif

'Pre-Processor Symbols for standard pause statements

'===================================================

'NB: These should work regardless of clockrate, just enter statement like "Pause10mS" in your program.

'Pause Statements using "pauseus", which is normally 10uS units.

symbol ustime1mS = **100** ClockRatio

#define Pause1mS pauseus ustime1mS

symbol ustime3mS = **300** ClockRatio

#define Pause3mS pauseus ustime3mS

symbol ustime10mS = **1000** ClockRatio

#define Pause10mS pauseus ustime10mS

symbol ustime30mS = **3000** ClockRatio

#define Pause30mS pauseus ustime30mS

symbol ustime100mS = **10000** ClockRatio

#define Pause100mS pauseus ustime100mS

'Pause Statements using "pause", which is normally 1mS units.

symbol time300mS = **300** ClockRatio

#define Pause300mS pause time300mS

symbol time1S = **1000** ClockRatio

#define Pause1S pause time1S

symbol time3S = **3000** ClockRatio

#define Pause3S pause time3S

symbol time10S = **10000** ClockRatio

#define Pause10S pause time10S

#Endregion

# Program Listing – “DF\_Player\_Mini.basinc”

#REM

File: "DF\_Player\_Mini.basinc"

Date/Version: 2024-08-21

Author: Alan Hunt

Description

============

This PICAXE Library eases the use of the DF Player Mini audio module. It provides macros and subroutines to assist in speech, sound and music output.

The main 3 macros include a Title parameter, which allows short descriptive text to be passed. The Title can greatly improve readability of your program and it consumes no program memory if debugging is commented out, e.g. if track 30 is configured to say "thousand" the program statement would simply be PlayMP3(30,"thousand"). The Title output can also be very helpful when debugging is enabled, but program memory will diminish very quickly, so the text should be short.

The DF Player Mini allows an SD card to be loaded with .mp3 or .wav tracks that can easily be selected from the root MP3 folder. The filename simply must begin with a number 000 to 255. Advert tracks can also be played that temporarily suspend the main playing track. These Adverts have uses such as playing a crash sound during a game and they are placed in the root ADVERTS folder.

The DF Player module does not behave entirely as expected, such as the busy pin taking time to become active. Testing has found minimum pause times required to acheive reliability. These utilities provide these timings and simple methods to interact with the audio module, such as the macro "AudioSpeakNumber(Val)".

The list of macros include:

PlayMP3(Track, Title) Plays the Track after the previous track is finished. Title is for debugging/information.

PlayNow(Track, Title) Stops anything currently played or queued and plays the selected Track.

PlayAdvert(Track, Title) Plays the Advert and temporarily suspends anything currently playing.

AudioSpeakNumber(Val) Speaks the number Val, which is a Word value from 0 to 65,535.

AudioSpeakTenths(Val) Speaks the number Val divided by 10

AudioSpeakHundreths(Val) Speaks the number Val divided by 100

AudioSpeakThousandths(Val) Speaks the number Val divided by 1000

The list of subroutines include:

AudioInitialise To be included in program initialisation.

AudioWaitForIdle Wait for sound to stop.

AudioClear Stop sound.

AudioPlayMP3 Play the track defined by \_Audio\_MP3Track.

AudioPlayAdvert Play the advert defined by \_Audio\_AdvertTrack.

AudioSpeakInteger Speak the number defined by \_Audio\_Integer.

AudioSpeakDecimalPlaces Speak a number as individual digits, as spoken after a decimal place.

Dependencies

=============

Pause statements must be defined for 3mS, 30mS, 100mS, 300mS and 1S. In your main program you can either have pre-processor directives like '#define Pause3mS pause 3'; or you can '#include "Timing.basinc"' before this file, which will provide consistent timing with different clock rates. Certain symbols must also be defined for pins, variables and constants, see the Programming Guide section.

Sourcing the hardware

=====================

The DF Player Mini can be obtained from many sources and in many variants. Some of these are unreliable or appeared faulty. Here's a summary of types I compared:

1) Good: The PICAXE SPE035, labelled "DF Player Mini", with processor "YX5200-24SS, OE64 1607" provided good results and this utility is based on its timings. Like other units, completion of a track is reported twice from its serial output and the track number appears to be a random number, which is perhaps the file position in the SD card FAT.

2) Good if it works: From a pack of 3 modules marked "DF Player Mini, VO5.1, HW-247A" with processor "TD5580A" only 1 worked. Differences from (1) are that Busy went active earlier but stayed on later, typically 160mS after the modules track completion message. Also, there's a 10ms gap in the double reporting of track completion.

3) Bad: A batch labelled "MP-TF-16P-V3.0" with processor "MH2024K-24SS, 240113" were not usable. They were very slow, only reported track completion occasionally and did not reliably accept track requests.

4) Based on the findings above it's probably best to buy from an electronics store and avoid cheap packs from eBay, Amazon, AliExpress etc.

Programming Guide

==================

This utility is dependent on resources, such as Pins, Variables and Constants, that must be declared before the inclusion of this file. My preferred program structure is:

1) Program Notes

2) Compiler Directives

3) Resources

3a) Pins

3b) Variables

3c) Constants

3d) Utilities

4) Initialisation

5) Main:

6) Interrupt:

7) Subroutines

Based on the program structure above, here is an example configuration:

#Region "Resources"

'Pins

symbol AUDIO\_TX = B.0 '@DF\_Player\_Mini.basinc: Serial Tx to audio module.

symbol AUDIO\_STATUS = pinB.2 '@DF\_Player\_Mini.basinc: Busy is active low after about 230mS (check with constants).

'Variables

'@DF\_Player\_Mini.basinc Variables

symbol \_Audio\_Digit = b43 'Assign symbol to any spare Byte.

symbol \_Audio\_Integer = w22 'Assign symbol to any spare Word.

symbol \_Audio\_Fraction = w23 'Assign symbol to any spare Word.

symbol \_Audio\_Track = b48 'Assign symbol to any spare Byte.

symbol \_Audio\_Byte = b49 'Assign symbol to any spare Byte.

'Constants

'@DF\_Player\_Mini.basinc Constants

symbol AUDIO\_VOLUME = 31 '0 to 31(Maximum).

symbol AUDIO\_BAUD = T9600\_16 'Tx logic is idle high (T) and 9600 bps, the last characters you adjust to your clock speed.

symbol AUDIO\_ACTIVE = 0 'The AUDIO\_STATUS pin is active low, indicating audio playing.

symbol AUDIO\_IDLE = 1 'AUDIO\_STATUS high indicates the module is idle or seeking to play a track on the disk.

'Utilities

#include "Debug\_Terminal.basinc"

#include "Timing.basinc"

#include "DF\_Player\_Mini.basinc"

#Endregion

#EndREM

#Region "DF Player Audio Module Utilities"

'Some Aliases for subroutine readability

'=======================================

symbol \_Audio\_0to99 = \_Audio\_Byte

symbol \_Audio\_DP = \_Audio\_Byte

'Audio Macros

'============

#macro PlayMP3(Track, Title)

'DebugLineWithDecimal("@PlayMP3 ", Track, Title)

gosub AudioWaitForIdle

\_Audio\_Track = Track

gosub AudioPlayMP3

#endmacro

#macro PlayNow(Track, Title)

'DebugLineWithDecimal("@PlayNow ", Track, Title)

gosub AudioClear

\_Audio\_Track = Track

gosub AudioPlayMP3

#endmacro

#macro PlayAdvert(Track, Title)

\_Audio\_Track = Track

gosub AudioPlayAdvert

#endmacro

#macro AudioSpeakNumber(Val)

\_Audio\_Integer = Val

gosub AudioSpeakInteger

#endmacro

#macro AudioSpeakTenths(Val)

\_Audio\_Integer = Val / **10**

\_Audio\_Fraction = Val % **10**

gosub AudioSpeakInteger

PlayMP3(**61**,"point")

\_Audio\_DP = **1**

gosub AudioSpeakDecimalPlaces

#endmacro

#macro AudioSpeakHundreths(Val)

\_Audio\_Integer = Val / **100**

\_Audio\_Fraction = Val % **100**

gosub AudioSpeakInteger

PlayMP3(**61**,"point")

\_Audio\_DP = **2**

gosub AudioSpeakDecimalPlaces

#endmacro

#macro AudioSpeakThousandths(Val)

\_Audio\_Integer = Val / **1000**

\_Audio\_Fraction = Val % **1000**

gosub AudioSpeakInteger

PlayMP3(**61**,"point")

\_Audio\_DP = **3**

gosub AudioSpeakDecimalPlaces

#endmacro

goto AudioEnd

'Audio Subroutines

'=================

AudioInitialise:

'DebugLine("@AudioInitialise")

high AUDIO\_TX 'Output pins are default low, so this prepares communication.

Pause100mS

serOut AUDIO\_TX, AUDIO\_BAUD, \_

(**$7E**, **$FF**, **$06**, **$0C**, **$0**, **$0**, **$0**, **$EF**) 'Command $0C (Reset)

Pause1S 'AUDIO\_ACTIVE after 220mS and module messages going to sleep at 900mS.

serOut AUDIO\_TX, AUDIO\_BAUD, \_

(**$7E**, **$FF**, **$06**, **$09**, **$0**, **$0**, **$2**, **$EF**) 'Command $09 (Playback source), Arg $0002 (SD card)

Pause1S 'Pause to read SD card

serOut AUDIO\_TX, AUDIO\_BAUD, \_

(**$7E**, **$FF**, **$06**, **$06**, **$0**, **$0**, AUDIO\_VOLUME, **$EF**) 'Command $06 (Set Audio volume)

Pause100mS 'Pause to ensure volume set

return

AudioWaitForIdle:

'DebugLine("@AudioWaitForIdle")

do while AUDIO\_STATUS = AUDIO\_ACTIVE

Pause3mS

loop

return

AudioClear:

'DebugLine("@AudioClear")

serOut AUDIO\_TX, AUDIO\_BAUD, \_

(**$7E**, **$FF**, **$06**, **$15**, **$0**, **$0**, **$0**, **$EF**) 'Command $15. Stop Advert(Crash), if playing.

Pause100mS '75mS required for settling time.

'NB: Contrary to manual, command $16 did not stop everything.

serOut AUDIO\_TX, AUDIO\_BAUD, \_

(**$7E**, **$FF**, **$06**, **$16**, **$0**, **$0**, **$0**, **$EF**) 'Command $16. Stop main track, if playing

Pause30mS '14mS required before requesting the new track

return

AudioPlayMP3:

'DebugLineWithDecimal("@AudioPlayMP3: ", \_Audio\_MP3Track, NoText)

serOut AUDIO\_TX, AUDIO\_BAUD, \_

(**$7E**, **$FF**, **$06**, **$12**, **$0**, **$0**, \_Audio\_Track, **$EF**) 'Command $12. Play from MP3 folder.

do while AUDIO\_STATUS = AUDIO\_IDLE 'Ensures AUDIO\_STATUS caught up (seen delays up to 290mS)

Pause3mS

loop

return

AudioPlayAdvert:

'DebugLineWithDecimal("@AudioPlayAdvert: ", \_Audio\_AdvertTrack, NoText)

serOut AUDIO\_TX, AUDIO\_BAUD, \_

(**$7E**, **$FF**, **$06**, **$13**, **$0**, **$0**, \_Audio\_Track, **$EF**) 'Command $13. Play ADVERT (suspends current track while playing).

Pause300mS '170mS required because AUDIO\_STATUS goes Idle before main track resumed.

return

AudioSpeakInteger:

'DebugLineWithDecimal("@AudioSpeakInteger: ", \_Audio\_Integer, NoText)

'Thousands

\_Audio\_0to99 = \_Audio\_Integer / **1000**

if \_Audio\_0to99 > **0** then

gosub \_AudioSpeak1to99

PlayMP3(**30**,"thousand")

endif

'Hundreds

\_Audio\_Digit = \_Audio\_Integer dig **2**

if \_Audio\_Digit > **0** then

PlayMP3(\_Audio\_Digit,"aDigit[0to9]")

PlayMP3(**29**,"hundred")

endif

'Units

\_Audio\_0to99 = \_Audio\_Integer % **100**

if \_Audio\_Integer > **99** AND \_Audio\_0to99 > **0** then

PlayMP3(**0060**,"and")

endif

gosub \_AudioSpeak1to99

if \_Audio\_Integer = **0** then

PlayMP3(**0**,"zero")

endif

return

AudioSpeakDecimalPlaces:

'DebugTextWithDecimal("@AudioSpeakDecimalPlaces: ", \_Audio\_DP)

'DebugLineWithDecimal(" places for ", \_Audio\_Integer, NoText)

dec \_Audio\_DP

for \_Audio\_Digit = \_Audio\_DP to **0** step **-1**

\_Audio\_Track = \_Audio\_Fraction dig \_Audio\_Digit

PlayMP3(\_Audio\_Track,"aDigit[0to9]")

next \_Audio\_Digit

return

\_AudioSpeak1to99:

'Speak numbers 1 to 99 (zero is not sounded here). NB: "dig" operator needs to be the first or only argument.

\_Audio\_Digit = \_Audio\_0to99 dig **1**

if \_Audio\_Digit >= **2** then

\_Audio\_Track = \_Audio\_Digit + **18**

PlayMP3(\_Audio\_Track,"aDecade[20to90]") 'Twenty=20, Thirty=21 etc

\_Audio\_Track = \_Audio\_0to99 dig **0**

if \_Audio\_Track > **0** then

PlayMP3(\_Audio\_Track,"[0to9]")

endif

else

if \_Audio\_Digit = **1** then

\_Audio\_Track = \_Audio\_0to99 dig **0** + **10**

PlayMP3(\_Audio\_Track,"[10to19]")

else

\_Audio\_Track = \_Audio\_0to99 dig **0**

if \_Audio\_Track <> **0** then

PlayMP3(\_Audio\_Track,"[0to9]")

endif

endif

endif

return

AudioEnd:

#Endregion

# Program Listing – “Voltage.basinc”

#REM

File: "Voltage.basinc"

Date/Version: 2024-07-09a

Author: Alan Hunt

This file provides macros and subroutines to assist power supply and battery monitoring using the PICAXE 10bit ADC channels.

When calculating the PICAXE supply voltage, Vdd, the internal 1.024V reference is used and the reading is scaled up to calculate Vdd. The reading will be between 581 (1023\*1.024V/1.8Vdd) and 201 (1023\*1.024V/5.2Vdd).

Please note, the CapacityByVolts macro would ideally use an OCV(Open Circuit Voltage) and consider current drain history, resting time and temperature. It will provide better results with low drain circuits, below 0.1C, and small temperature ranges, like household use. A calibrated lookup table needs to be produced that is placed in EEPROM memory to avoid draining available program space.

The list of macros included:

GetVdd Power supply in mV, Voltage\_Vdd, calculated by scaling up internal 1.024V reference.

SetVdd If the power supply is accurately known and stable, the Voltage\_Vdd can be set manually.

VoltsByVdd(Volts, Channel) Provides a Word variable with the voltage in mV for the specified Channel.

CapacityByVolts(Capacity, Volts) Provides the battery capacity based on voltage to capacity data held in EEPROM memory.

An example configuration is shown below:

#Region "Resources"

'Variables

'@Voltage.basinc Variables

symbol Voltage\_Vdd = w25 'Assign symbol to any spare Word. Stores power supply voltage to PICAXE chip in mV.

symbol \_ADCval = w26 'Assign symbol to any spare Word.

symbol \_VoltageW1 = w27 'Assign symbol to any spare Word.

'Constants

'@Voltage.basinc Constants

symbol V2C\_ADDR = 130 'EEPROM Address for start of Voltage to Capacity Table.

symbol V2C\_MIN = 3000 'Lowest voltage in Voltage to Capacity Table, below this zero is returned.

symbol V2C\_MAX = 4200 'Highest voltage in Voltage to Capacity Table, after this 100 is returned.

symbol V2C\_INC = 100 'Voltage difference to next address. NB: The table length is (Max-Min)/Inc+1.

'Utilities

#include "Voltage.basinc"

#Endregion

#EndREM

#Region "Voltage Monitoring Utilities"

'==== Voltage Monitoring Utility ====

goto VoltageEnd

#macro GetVdd

calibadc10 \_ADCval 'Read reference 1.024v (Value = 1023 \* 1.024V / Vdd)

Voltage\_Vdd = **10450** / \_ADCval \* **100** 'Obtain first 2 digits, nnxx. Max=5200(5211), Min=1800(1803)

\_VoltageW1 = **10450** // \_ADCval 'Find remainder. Max=580, Min=0

Voltage\_Vdd = **100** \* \_VoltageW1 / \_ADCval + Voltage\_Vdd 'Obtain last 2 digits, xxnn. Max=99(=100\*580/581), Min=0

'DebugLineWithDecimal(" 1.024Vref = ",\_ADCval,NoText)

'DebugLineWithDecimal(" Vdd = ",Voltage\_Vdd,"mV")

#endmacro

#macro SetVdd(VoltsW)

Voltage\_Vdd = VoltsW

#endmacro

#macro VoltsByVdd(VoltsW, Channel)

'Voltage = Vdd(1800 to 5200)mV \* ADC10(0 to 1023) / 1023(ADC range). NB: ADC range is approximately 1024

readadc10 Channel, \_ADCval 'Read ADC Channel, the range is 0 to 1023 (Gnd to Vdd)

VoltsW = Voltage\_Vdd \*/ \_ADCval 'Get middle word of Vdd \* ADC10 (= result divided by 256)

VoltsW = VoltsW / **4** 'Divide by 4 (now we have approximately Vdd \* ADC10 / 1023)

'DebugLineWithDecimal("ADC Value = ",\_ADCval,NoText)

'DebugLineWithDecimal(" Voltage = ",VoltsW,"mV")

#endmacro

#macro CapacityByVolts(Capacity, Volts)

'This function is not tested and would never be very accurate.

'The idea is to use a battery voltage to battery capacity table in EEPROM, but typically batteries maintain their voltage well until exhaustion.

'A table needs to be produced based on testing a specific battery with a representative load and temperature and it would need re-producing with age.

'The table would produce rough capacity values but would be very dependent on the battery type and a fairly small or stable load, along with battery age and temperature.

select case Volts

case < V2C\_MIN

Capacity = **0**

case >= V2C\_MAX

Capacity = **100**

else

\_VoltageW1 = Volts - V2C\_MIN / V2C\_INC + V2C\_ADDR

read \_VoltageW1, Capacity

endselect

'DebugLineWithDecimal(" Volts = ",Volts, "mV")

'DebugLineWithDecimal(" Addr = ",\_VoltageW1, NoText)

'DebugLineWithDecimal("Capacity = ",Capacity,"%")

#endmacro