Nov. 17 Lesson Plan – Using Easy C to Program a Vex Robot

The "C" Programming Language

- Developed by Dennis Ritchie from 1969 to 1973 at AT&T Bell Labs.
- One of the most widely used programming languages of all time.
- Computer Code is "Compiled" by software into Assembly Language Code (Human readable) *(machine readable, 0"s and 1") and finally into machine language code. (by software called as "Assembler").

*written "Hex" for Efficiency ⇒ hexadecimal

(Ex) 97 = 61 = 01100001

Base 16 Hexadecimal Base-16 | Base 2 Machine Code

"CODE"

EASY C ⇒ • FUNCTION BLOCKS
• BLOCK AND C PROGRAMMING
• PROJECT EXPLORER
↳ * CONTROLLER CONFIGURATION

CONTROLLER CONFIGURATION ⇒
- SENSES ENVIRONMENT
- CONTROLS THINGS ON ROBOT

THE DESCRIPTION SHOWS UP IN DROP-DOWN CHOICES WHEN ADDING FUNCTION BLOCKS TO BUILD CODE.

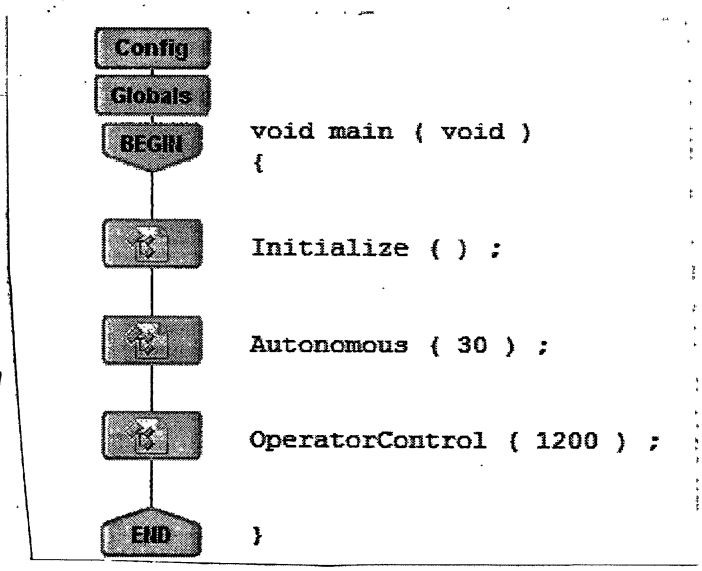
- 8 ANALOG PORTS (0 to 1023)
  - INPUT ONLY (EX POTENTIOMETER) ≐ 1024 VALUES
- 12 DIGITAL PORTS (0 AND 1)
  - ≐ 2 VALUES
  - INPUT or OUTPUT (EX LIMIT SWITCH, EX LED)
  - SP = SPEAKER PORT
  - MUST CONFIGURE: ●→○ INPUT; ●←○ OUTPUT
- MOTORS
  - 10 NORMAL PORTS } MUST BE CONFIGURED
  - 10 IME'S ⇒ 1 for each NORMAL } MUST BE ASSOCIATED

Overview of a Program

→ File/Open
- Standalone
- Competition
  - Field *
    - Control
    - Timed

- Code blocks
- Functions ⇒ Tabs

"Subroutines"
"Tasks"



VOID ⇒ DOES NOT "RETURN" any updated or new variables to calling program

(↑)
Argument ⇒ WHAT "calling" code sends to the function

Local vs. Global variables

Local ⇓ WITHIN EACH function

Global ⇓ FOR ALL functions

ARM Cortex M3 MCU ~ 90 million instructions Per second!

Write A new Field Competion Program.

Initialize( ): . Runs before both the Autonomous and Operator Control Functions
- optional function
- not timed

Lets Flash an LED:

Program Flow:

(1) Comment "LIGHT GREEN LED"

Output:

(2) Controller Configuration ← Digital — OUTPUT: [6] SET: [0] TURN LED ON

Digital Port 6 [0]

(3) Prog. Flow: WAit [1000] (1000 millisec = 1 sec)

(4) Output: OUTPUT [6] SET: [1] TURN LED OFF

See any problem with this code?

It only runs once. We need a loop. Use a while loop. When you want a command to "always be 'waiting'", put it in a

```
while(1){ loop
    ↑
    Condition always true
}
```

* Braces open and close loop.

used for JOYSTICK commands for example. Loop is thousands to millions of times per second!

But an infinite loop will mean you are stuck in the initialize function. We need an option.

Typical "while" loop conditions are "while (some sensor is <, >, = ?)"

or "while (A counter "counts" the number of loops),

Lets use A counter.

---

(5) Add While Loop ⇒ Ctr-F6, define "count_loops" "INT"

* NO SPACES IN NAME

Common Types of Variable		
	INT	Integer ± 32,000
	long	integer ± 2billion
	float	decimal } 10^{38} (6 decimal places)
	char	any keyboard "key"

LOOK AT HELP

⇒ CODE `while (count_loops < 5)` 0, 1, 2, 3, 4

(6) Move CODE INTO While loop

*(7) INCREMENT COUNTER! (8) Add wait after "OFF"

↳ ASSIGN count_loop = count_loops + 1

Set Times in "Main" to Stay in Autonomous (robot follows code commands only) and Operation Control (robot follows commands and can be controlled by Joystick) → time in seconds

---

set to 20 sec (Auto) and 10 min (600 sec) (OperCtrl)

Go to Autonomous ⇒ move forward and backward

Configuration Control ⇒

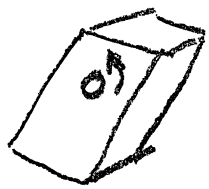
Motor 1 ⇒ Left Side Drive
10 ⇒ Right Side Drive

Type [STANDARD] 1 ⇒ Left, 10 ⇒ Right

AUTONOMOUS CODE – No Loop – JUST ONCE

① SMART TASKS – MultiMotor Time Control

Power



LOOKING AT MOTOR, SHAFT ROTATION IN CCW direction is +, 0 to 127

CW is – 0 to 127

#1 | 1 // LEFT | #2 | 10 // RIGHT |

☑ Invert ☐

FWD ⇒ Power = +30 for 2000 milli-sec

② WAIT (1000)

③ Reverse ⇒ "-30" for power for 2 sec.

④ WAIT (1000)

AUTO SHAFT TO OPERATOR CONTROL FUNCTION

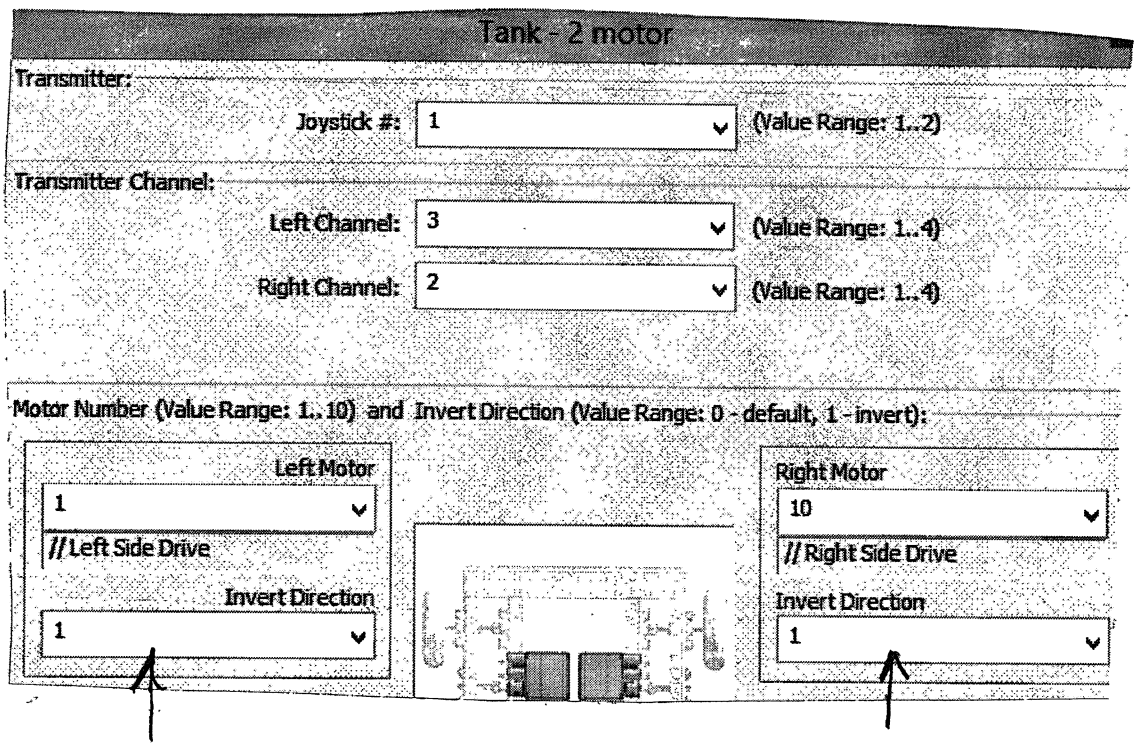
OPERATOR CONTROL

- Just the two drive motors to start with

---

(1) Joystick

↳ "TANK - 2 motor" ... CODE



NEEDED TO MAKE BOTH JOYSTICKS WORK

↑ FORWARD

↓ REVERSE

} WIRING? "RED WIRE TOWARD CENTER?

CAN USE "JOYSTICK - DIGITAL TO MOTOR" TO ADD CONTROLS FOR ARM & CLAW MOTORS.

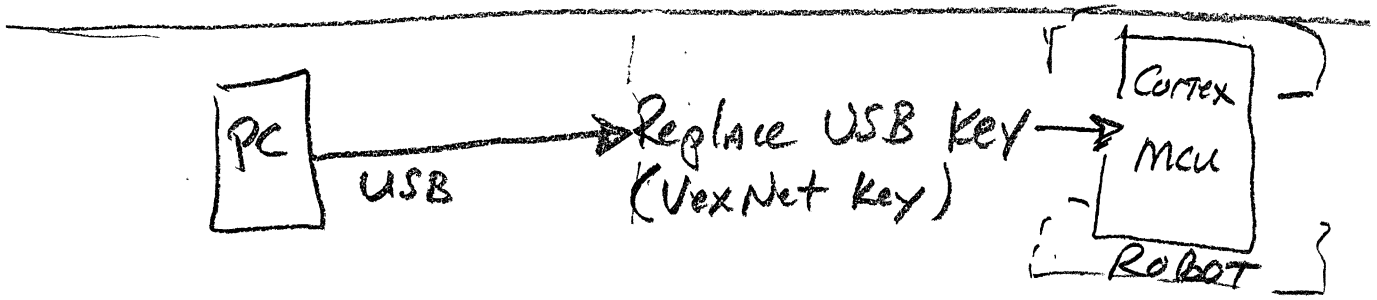
Lets Install & Test The Program

"NOV_17_lesson_plan_pcb_part_a"

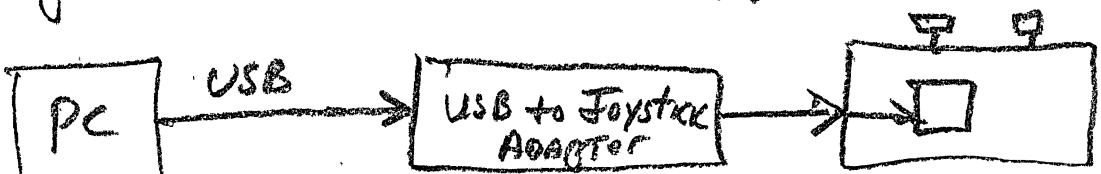
① * Two steps ⇒ Compile (fix errors)
(recommended) ↓
Build and Download

OR

One Step ⇒ Build and Download



② Replace VexNet USB Key, connect



PLUG INTO "phone" JACK LABELED "Program" ON Joystick (leave Joystick OFF)

*PUT ROBOT ON TEST STAND

③ Turn on Robot – wait for VexNet SYNCH.
LAUNCH "TOOLS – "Terminal window" – Press Button (any) on Joystick

Program ⇒ Flashes LED ⇒ "initialise code"

Energizes Drive Motors Fwd/Rev ⇒ Autonomous mode

Flashes LED ⇒ Initialise code

Allows Joystick to control drive motors Fwd/Rev ⇒ operator control mode

---

* Can comment out comms programmer task

Lets add a few "Print to screens", very useful for debugging — don't want in competition, may slow processing —

- Turn everything off — rehook USB to Cortex

Save "Program As"

NOV_17_lesson_plan_per_1_part_6"

Add "Program Flow — Print to Screen" Commands to:

initialize ⇒ just in front of loop counter

"PrintToScreen ("message", count_loops)
↑ variable

AUTONOMOUS ⇒ first line

"Print to Screen (Autonomous Mode Started)"

oper. control ⇒ first line

* "Print to Screen (Operator Control Mode Started)

* in front of while loop — why?

SAVE FILE — BUILD / DOWNLOAD

Reinstall Vex Key — hook up USB to Joystick — turn on Robot — Start Terminal Window — hit any Joystick Button — voila!

- Demo "On-line Window" Output

- Review Other Menu Options

Plus mini-reviews of code commands.

Install latest EasyC

Program that exercises all of the installed "temporary" sensors.

* User Programs – benefits, especially for team programming

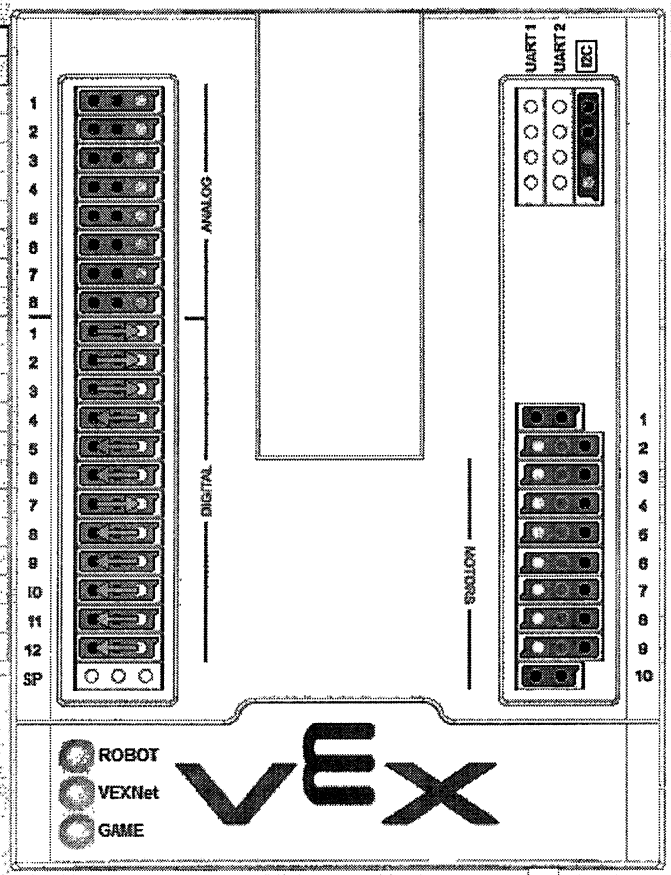
- Questions

- Time Permitting
  - Intro. to PID Video
    (the difference between "move east" and "move to downtown Atlanta")

Complete Controller Configuration as of Nov. 16, 2014

Controller Configuration

ANALOG & DIGITAL	
#	Description
1	PotentiometerA
2	
3	
4	
5	
6	
7	
8	
1	
2	
3	Echo From Ultra RF
4	Trigger Ultra RF
5	Red LED
6	Green LED
7	Arm Limit Switch
8	Yellow LED
9	
10	
11	
12	



INTEGRATED MOTOR ENCODERS		
I2C #	Motor Port #	Description
1	7	Arm Drive Motor393 with IME
2	2	Test Motor 393 with IME
3	3	
4	4	
5	5	
6	6	
7	1	
8	8	
9	9	
10	10	

MOTORS		
#	Motor Type	Description
1	Standard	Left Side Drive Motor - 2 Wire 393
2	Big IME	Test Motor - 2 Wire 393 with IME
3	n/a	
4	n/a	
5	n/a	
6	Standard	Claw Drive Motor - 2 Wire 393
7	Big IME	Arm Drive Motor - 2 Wire 393 with IME
8	n/a	
9	n/a	
10	Standard	Right Side Drive Motor - 2 Wire 393

Motor Type Information
n/a - Motor Type is not provided
Standard - Motor Module without Integrated Encoder
Small IME - 269 with Integrated Encoder
Big IME - 393 with Integrated Encoder
Big IME HS - 393 High Speed Gearing with Integrated Encoder

Left-Click to set Digital I/O | Restore Defaults | OK | Cancel | Help

Port Configuration – Vex Robot – Period 7
As of Nov. 16, 2014

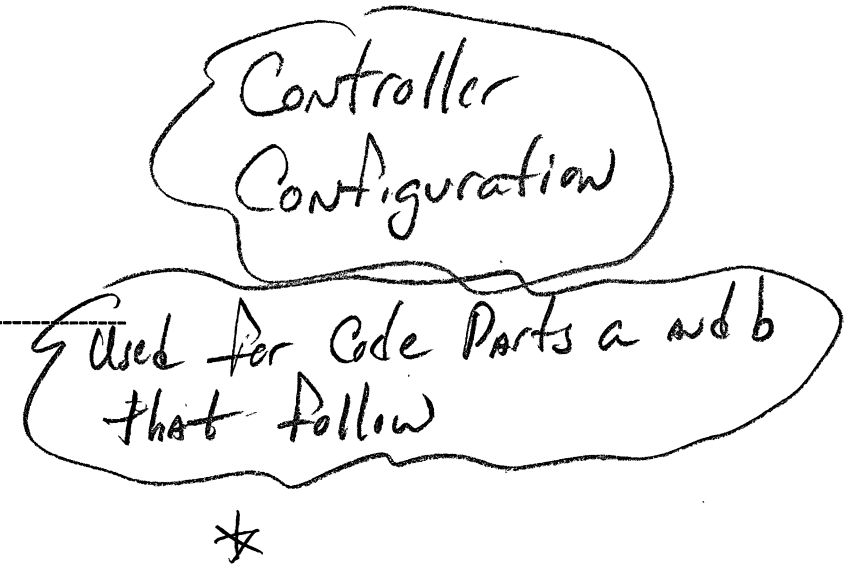
Controller configuration for project: nov_17_lesson_plan_per_1_part_a
Date: 11/16/2014 Time: 15:55

ANALOG INPUTS:
Input #1
Input #2
Input #3
Input #4
Input #5
Input #6
Input #7
Input #8

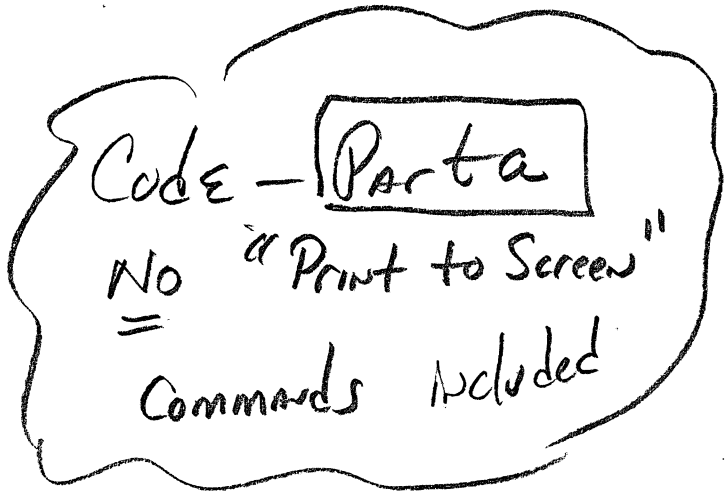
DIGITAL IN/OUT:
Input #1
Input #2
Input #3
Input #4
Input #5
Output #6 Green LED
Input #7
Input #8
Output #9
Output #10
Output #11
Output #12

MOTORS:
Motor #1 Type: Standard Description: Left Side Drive
Motor #2 Type: n/a Description:
Motor #3 Type: n/a Description:
Motor #4 Type: n/a Description:
Motor #5 Type: n/a Description:
Motor #6 Type: n/a Description:
Motor #7 Type: n/a Description:
Motor #8 Type: n/a Description:
Motor #9 Type: n/a Description:
Motor #10 Type: Standard Description: Right Side Drive

INTEGRATED MOTOR ENCODERS:
IME #1 Motor# 1 Description:
IME #2 Motor# 2 Description:
IME #3 Motor# 3 Description:
IME #4 Motor# 4 Description:
IME #5 Motor# 5 Description:
IME #6 Motor# 6 Description:
IME #7 Motor# 7 Description:
IME #8 Motor# 8 Description:
IME #9 Motor# 9 Description:
IME #10 Motor# 10 Description:



```
#include "Main.h"
void main ( void )
{
    Initialize ( ) ;
    Autonomous ( 20 ) ;
    OperatorControl ( 600 ) ;
}
```



```
#include "Main.h"
void Initialize ( void )
{
    int count_loops = 0; // flash led loop counter
    // Do NOT place infinite loops inside of your Initialize Function.
    // If your Initialize Function never terminates, your program
    // will never progress to Autonomous and Operator Control.
    // See Help for more information about this function.
    // Flash Green LED
    while ( count_loops < 5 )
    {
        SetDigitalOutput ( 6 , 0 ) ; // Turn LED ON
        Wait ( 1000 ) ;
        SetDigitalOutput ( 6 , 1 ) ; // Turn LED off.
        Wait ( 1000 ) ;
        count_loops = count_loops + 1 ;
    }
}
```

```
#include "Main.h"
void Autonomous ( unsigned long ulTime )
{
    MultiMotorTimeControl ( 2 , 1 , 10 , 8 , 9 , 1 , 0 , 0 , 0 , 30 , 2000 ) ; // Both drive motors forward.
    Wait ( 1000 ) ;
    MultiMotorTimeControl ( 2 , 1 , 10 , 8 , 9 , 1 , 0 , 0 , 0 , -30 , 2000 ) ; // Both drive motors reverse.
    Wait ( 1000 ) ;
}
```

```
#include "Main.h"
void OperatorControl ( unsigned long ulTime )
{
    while ( 1 ) // This loop is ALWAYS true - put your Joystick code here.
    {
        Tank2 ( 1 , 3 , 2 , 1 , 10 , 1 , 1 ) ;
    }
}
```

```
#include "Main.h"
void main ( void )
{
    Initialize ( ) ;
    Autonomous ( 20 ) ;
    OperatorControl ( 600 ) ;
}
```

Code Part b

Includes "Print to Screen" Commands — Use "Terminal Mode" to monitor screen output.

```
#include "Main.h"
void Initialize ( void )
{
    int count_loops = 0; // flash led loop counter
    // Do NOT place infinite loops inside of your Initialize Function.
    // If your Initialize Function never terminates, your program
    // will never progress to Autonomous and Operator Control.
    // See Help for more information about this function.
    // Flash Green LED
    while ( count_loops < 5 )
    {
        SetDigitalOutput ( 6 , 0 ) ; // Turn LED ON
        Wait ( 1000 ) ;
        SetDigitalOutput ( 6 , 1 ) ; // Turn LED off.
        Wait ( 1000 ) ;
        PrintToScreen ( "Flash LED Loop Counter%d\n" , count_loops ) ; // Flash LED Loop Counter
        count_loops = count_loops + 1 ;
    }
}
```

```
#include "Main.h"
void Autonomous ( unsigned long ulTime )
{
    PrintToScreen ( "Autonomous Mode Started\n" ) ;
    MultiMotorTimeControl ( 2 , 1 , 10 , 8 , 9 , 1 , 0 , 0 , 0 , 30 , 2000 ) ; // Both drive motors forward.
    Wait ( 1000 ) ;
    MultiMotorTimeControl ( 2 , 1 , 10 , 8 , 9 , 1 , 0 , 0 , 0 , -30 , 2000 ) ; // Both drive motors reverse.
    Wait ( 1000 ) ;
}
```

```
#include "Main.h"
void OperatorControl ( unsigned long ulTime )
{
    PrintToScreen ( "Operator Control Mode Started\n" ) ;
    while ( 1 ) // This loop is ALWAYS true - put your Joystick code here.
    {
        Tank2 ( 1 , 3 , 2 , 1 , 10 , 1 , 1 ) ;
    }
}
```