Proceedings of the ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2014
August 17-20, 2014, Buffalo, New York, USA

DETC2014-35355

A Matlab-Based Toolkit to program Microcontrollers for use in Teaching Mechanisms and Robotics

Stephen L. Canfield, Sheikh Ghafoor
Tennessee Technological University
Depts. of Mechanical Engineering, Computer Science
Cookeville, Tennessee, 38505
United States of America
SCanfield@tntech.edu

Abstract

Programming is an essential skill for engineering students, particularly in areas of mechanisms and robotics, mechatronics and design. Students receive formal programming training early on in the typical engineering curriculum, but generally demonstrate difficulty in implementing programming skills to solve engineering problems in later courses. This is due to a number of factors including a lack of cohesion in programming practice in the curriculum and improper context for introducing programming to engineering students. The authors have developed a hands-on programming toolkit to allow engineers to learn programming on a Microcontroller and associated hardware (sensors, motors, output devices) using the Matlab environment. This toolkit is applicable to all levels of students, from freshman in their introductory programming course through senior and graduate students. In this hands-on toolkit, the MCU becomes the target for the program. Once programmed, the hardware runs independently and can readily be implemented outside of class or the lab. The primary method of programming an MCU is with C or assembly. One of the unique offerings of this toolkit is that it provides a way to program an MCU directly using Matlab. The premise is that adding an MCU as a programming target, rather than simply a PC, may provide a more appropriate context for engineers to learn programming. In addition, the MCU target will offer a greater number of options for incorporating programming into the engineering curriculum. This paper will present an overview of the programming toolkit and environment. The paper will also present a series of programming exercises related to dynamics of machinery, robotics, mechatronics and controls.

1. Introduction

Programming is an essential skill for engineering students. The traditional entry-level programming course for engineers is based on learning C, Fortran or Matlab [1] to solve numerical algorithms associated with common engineering models. Students receive formal programming training early on in a typical engineering curriculum but often demonstrate difficulty in implementing programming skills to solve engineering problems in later courses. To reinforce students' ability to use programming tools, educators are looking for opportunities to implement more programming activities in the curriculum. The mechanisms and robotics type courses provide one such opportunity, through closed loop control systems operating on an embedded processor (i.e., a microcontroller that is part of the machine). In order to use microcontrollers in machines, historically this has required a student to know C or assembly and thorough understanding microcontroller's processor architecture. This is now changing. Two of the primary examples are the Raspberry Pi [2] and the Arduino [3] movements with a low cost microcontroller unit (MCU) and available code snippets. However, these programs often do not relate readily to other programs written in classes, in particular if the class has more emphasis in using Matlab. This paper will present another tool to aid in rapid or ready implementation of embedded control systems in mechanisms and robotics courses. This will allow students to create, compile, and download their program written as Matlab script files directly from the Matlab environment to the MCU board. There are a few other efforts using Matlab as an embedded processor. Mathworks provides Matlab support for Arduino microcontroller board through Simulink [3]. Alternatively, several examples of using Matlab scripts to create serial objects for data acquisition and communication with MCU Boards are [5, 6] but rely on the programming remaining on the desktop. A large variety of a microcontroller-based educational classroom/lab projects exists in different academic institutes [7-9]. This paper will present an alternative tool that allows structured programming via Matlab script files for standalone or embedded operation on a microcontroller.

The remainder of this paper will proceed as follows. The description of the toolkit is provided in section 2. A curriculum for mechanism and robotics class using the tool kit is described in section 3. The paper ends with concluding remarks in section 4.

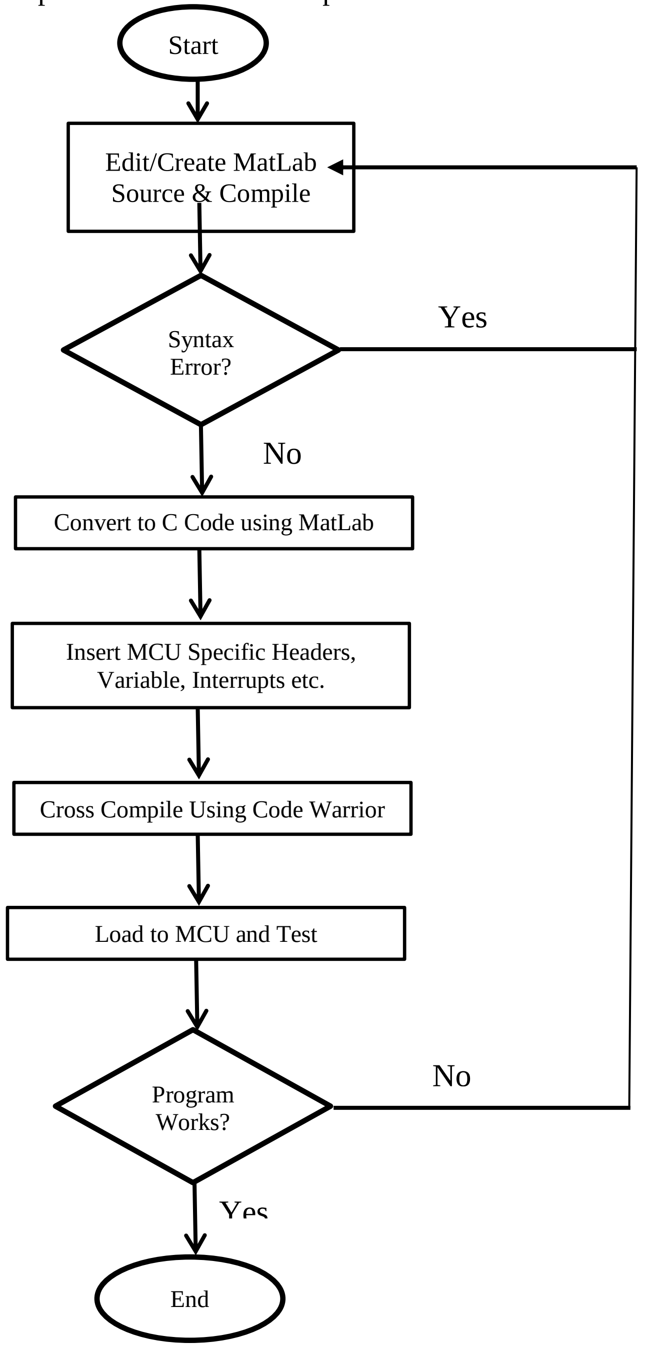
2. Approach

The toolkit consists of a combination of software package and MCU board. The software package exists as a toolbox for Matlab and is used to compile and download matlab scripts or programs to the target MCU. The software package also includes a variety of functions written in Matlab (or C) to help with common programming tasks. The target MCU is selected as the Dragon12 plus evaluation board based on the Motorola MC9S12 processor, because it contains a wide range of input/output devices suitable for use in labs and historically it has been used in educational institutes. The software and hardware are discussed further in the following sections.

2.1 Software

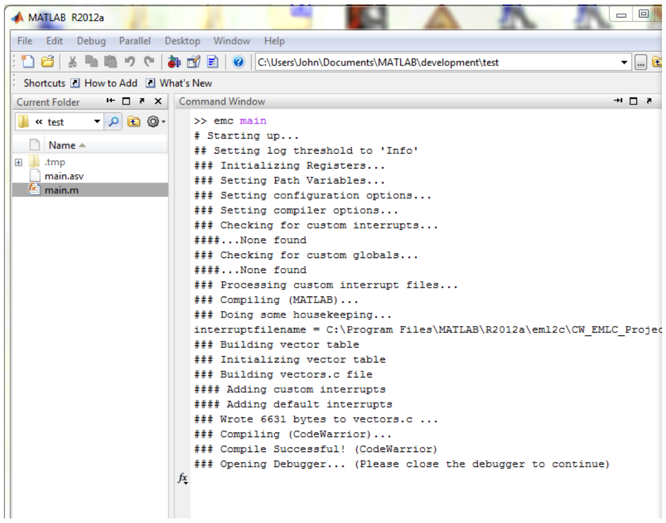
The software package of the toolkit consists of five components: 1) Matlab MCU function Library, 2) Matlab to C Translator, 3) MCU Code Converter, 4) MCU Cross Compiler, and 5) Loader. For component 2, the Matlab Coder has been used and free scale code warrior [10] has been used for components 4 and 5. Components 1 and 3 were developed by the authors and are packaged into a single toolkit.

2.1.1 Overview: Figure 1 shows the flow diagram of compilation process of Matlab program for the MCU. The user creates a Matlab script (.m file) in Matlab environment. To control the I/O devices/Port on the MCU the user calls the Matlab to MCU functions. The user then issues a single command from Matlab window to the invoke toolkit which will compile the Matlab script and upload it onto the MCU board (Fig. 2 shows a screen shot of the command window for this operation while Fig. 3 shows the hardware interface window once download is complete). A brief description of the components of the toolkit is provided below



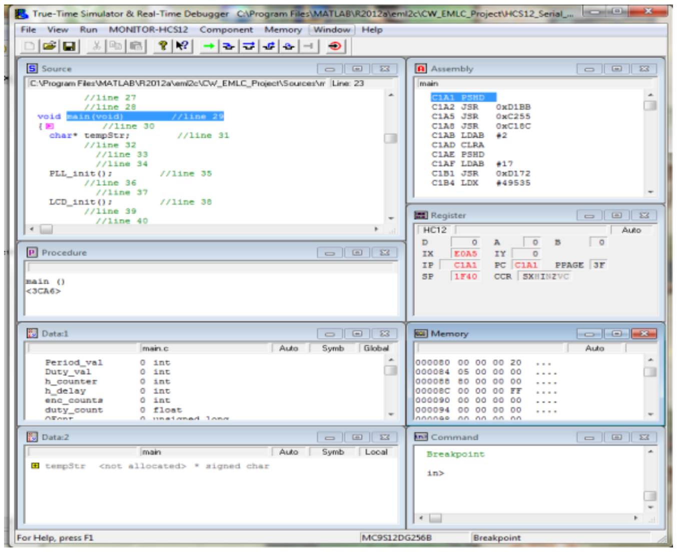
Start
Edit/Create MatLab Source & Compile
Syntax Error?
Yes
No
Convert to C Code using MatLab
Insert MCU Specific Headers, Variable, Interrupts etc.
Cross Compile Using Code Warrior
Load to MCU and Test
Program Works?
No
Yes
End

Fig 1: Flow chart of Generating MCU Code for Matlab.


MATLAB R2012a
File Edit Debug Parallel Desktop Window Help
C:\Users\John\Documents\MATLAB\development\test
Shortcuts How to Add What's New
Current Folder
« test
Name
.tmp
main.asv
main.m
Command Window

```
>> emc main
# Starting up...
## Setting log threshold to 'Info'
### Initializing Registers...
### Setting Path Variables...
### Setting configuration options...
### Setting compiler options...
### Checking for custom interrupts...
####...None found
### Checking for custom globals...
####...None found
### Processing custom interrupt files...
### Compiling (MATLAB)...
### Doing some housekeeping...
interruptfilename = C:\Program Files\MATLAB\R2012a\eml2c\CW_EMLC_Projec
### Building vector table
### Initializing vector table
### Building vectors.c file
#### Adding custom interrupts
#### Adding default interrupts
### Wrote 6631 bytes to vectors.c ...
### Compiling (CodeWarrior)...
### Compile Successful! (CodeWarrior)
### Opening Debugger... (Please close the debugger to continue)
```

Fig. 2: Compiling program in Matlab Command window


True-Time Simulator & Real-Time Debugger C:\Program Files\MATLAB\R2012a\eml2c\CW_EMLC_Project\HCS12_Serial_...
File View Run MONITOR-HCS12 Component Memory Window Help
Source
C:\Program Files\MATLAB\R2012a\eml2c\CW_EMLC_Project\Sources\m
Line: 23
Procedure
Data:1
main.c
Auto
Symb
Global
Assembly
main
Register
HC12
Auto
Memory
Auto
Data:2
main
Auto
Symb
Local
Command
For Help, press F1
MC9S12DG256B
Breakpoint

```
            //line 27
            //line 28
void main(void)           //line 29
{             //line 30
  char* tempStr;          //line 31
            //line 32
              //line 33
              //line 34
  PLL_init();           //line 35
            //line 36
              //line 37
  LCD_init();           //line 38
            //line 39
              //line 40
```

```
main ()
<3CA6>
```

```
Period_val    0 int
Duty_val      0 int
h_counter     0 int
h_delay       0 int
enc_counts    0 int
duty_count    0 float
```

```
tempStr  <not allocated> * signed char
```

```
C1A1 PSHD
C1A2 JSR    0xD1BB
C1A5 JSR    0xC255
C1A8 JSR    0xC18C
C1AB LDAB   #2
C1AD CLRA
C1AE PSHD
C1AF LDAB   #17
C1B1 JSR    0xD172
C1B4 LDX    #49535
```

```
D      0     A     0     B     0
IX  E0A5     IY    0
IP  C1A1     PC C1A1     PPAGE 3F
SP  1F40     CCR SXHINZVC
```

```
000080  00 00 00 20   ...
000084  05 00 00 00   ....
000088  80 00 00 00   ....
00008C  00 00 00 FF   ....
000090  00 00 00 00   ....
000094  00 00 00 00   ....
```

```
Breakpoint

in>
```

Fig. 3: Hardware Interface Window once program download is complete

2.1.2 Matlab MCU Function Library: To assist in programming the Input/Output (I/O) devices, a library of functions has been created specifically for the MC9S12 and Dragon12 board. These functions are written in Matlab or C. The functions work in the same way all Matlab functions work and will show up in the command line help window when the function name is specified. For example to turn on the 1st LED The user needs to write MCU_SET_LED(1) in the Matlab script. The appendix contains a summary table of the functions that are currently available for the Matlab to MCU toolkit.

2.1.3 Matlab to C Translator: This component converts a Matlab code to equivalent C code. The Matlab Coder module that comes with Matlab has been used for code conversion.

2.1.4 MCU Code Converter: Matlab Coder translates a Matlab script to produce equivalent ANSI C code which can be compiled for a target PC. The converted C code produced by the Matlab Coder cannot be cross compiled for MCU without modification. A C program for MCU requires several libraries, header files, variable declaration, interrupt setting etc. which are not part of ANSI C. In addition in the translation process Matlab Coder deletes variables and arrays if they are not used or initialized in Matlab scripts. A user would be required to modify the translated code manually before it can be compiled and used on the MCU. This requires expertise in C language as well as a deeper understanding of CPU architecture and MCU I/O devices. The Code Converter performs this function by inserting required code and processor directives to the translated code in order to make compilable for the MCU without user intervention.

2.1.5 MCU Cross Compiler: Code Warrior [10] cross compiler has been used to compile the C code produced by the Code Converter.

2.1.6 Loader: The code warrior comes with a loader which has been used to upload the compiled MCU object code from PC to the MCU board through a USB cable. Once the program is uploaded, the user can test the program.

All these components are packaged into a single program and the whole process is transparent to the user. The user writes a program using native Matlab scripts and the MCU functions developed as part of the toolkit. Then the user issues a single command to invoke the toolkit which will translate, convert, compile, and upload the program to the MCU.

2.2 Hardware

The Matlab to microcontroller (MCU) toolkit discussed in this paper makes use of the Motorola MC9S12 processor implemented on an evaluation board through the Dragon12 plus (Dragon12 plus, www.evbplus.com accessed 12/20/11.) This processor is commonly used for engineering practice and has a large number of inputs and outputs making it suitable to control systems of two or more degrees of freedom. The Dragon12 evaluation board, shown in figure 4 below, has numerous input / output functions integrated directly with the MCU and ready to use in projects. Table I provides a summary of the primary features of the Dragon12. It should be noted that this particular selection of microcontroller and corresponding evaluation board is not a unique selection for this model, but is rather representative.

Table I: Summary of MHCS12/Dragon12 features

Product	Capability
Processor: MC9S12	16bit CPU, 24Mhz 256K Flash EEPROM, 12K RAM Serial communication, 10-bit ATD, timer channels, PWM, discrete I/O, interrupt I/O
Evaluation Board: Dragon12, www.wytec.com	Output Devices: 2x16 digit LCD, single-row LEDs, 4 – 7 segment LEDS, Piezo speaker
	Input Devices: 8 dip switches, 4 momentary switches, 16-key keypad, IR proximity sensor, Photoresister
	Other: Motor driver (H-bridge)

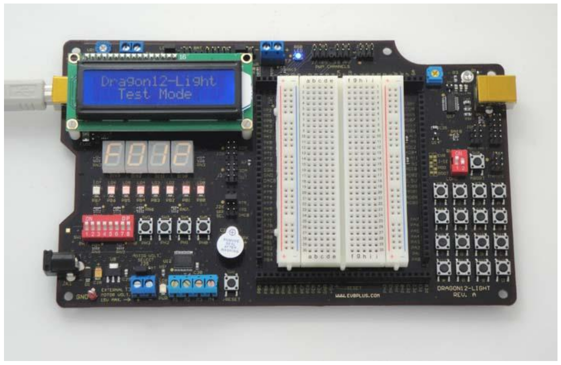


Figure 4: Dragon12 Evaluation board.

3. Curriculum Examples: Components useful in Mechanisms and Robotics Courses

This section will present an example of how the proposed toolkit could be used in a mechanisms or robotics course, to address course material and to introduce simple hands-on applications with common components. The overall example is first presented, following a breakdown of implementation in Matlab, transfer to an embedded processor through the Matlab toolbox, and incorporation of sensors and actuators through a function created in Matlab or direct control of the processor registers. Functions for each of these examples can be found at [11]

3.1 Inverse Kinematic control of a Serial Arm Robot:

In this activity, students can build, program and control a multi degree of freedom (dof) serial robot arm. This example will demonstrate the construction of a simplistic robot arm attached to a clipboard and capable of drawing shapes on paper. The robot is constructed from aluminum links driven by RC servo motors or stepper motors and controlled using the Dragon12 board. The example is presented in the following steps: 1) construction of the hardware, 2) describing tool-space motion, 3) generating joint-space motion, 4) implementing actuators, and 5) results.

3.2 Construction of the hardware.

The following figures outline a simple construction approach for the multi-link robot arm. This example assumes that standard RC motors are used. Figure 5 shows a clipboard with a plastic mount attached to hold a first RC servo motor. Figure 6 shows a standard link from ¼ x ½ x *l* aluminum bar with holes drilled to attach to the servo “horns” (standard hubs that attach to RC servomotors), as well as an access hole to allow attachment of the servo horn. Figure 7 shows a link attached to the horn of the RC servo. Figure 8 shows a link modified to hold a pencil. Finally, figure 9 shows the entire assembly.

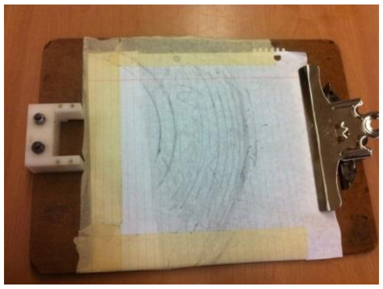


Figure 5: Clipboard with plastic mount

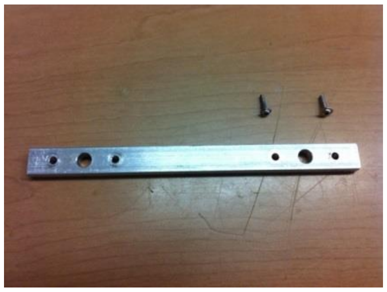


Figure 6: Aluminum link

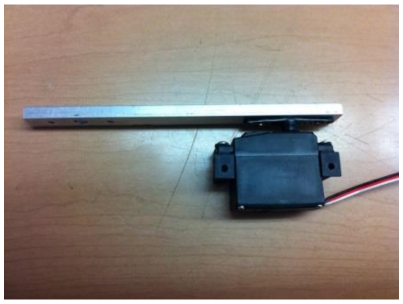


Figure 7: Aluminum link with Motor Horn

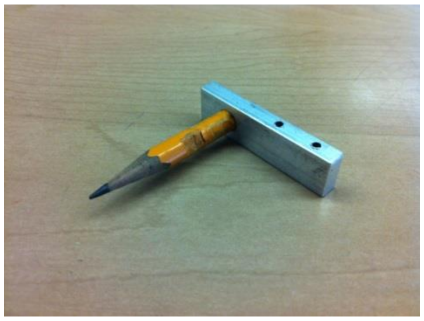


Figure 8: Modified link to hold pen

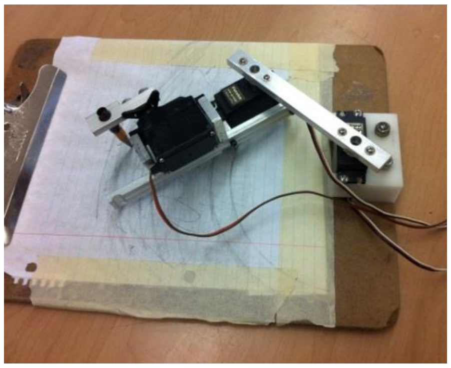


Figure 9: Serial Arm Assembly

3.3: Path planning:

A path description can be constructed in the Matlab programming language to describe desired tool-space locations. The example below constructs the path as a circle of radius *r* with center at (*h*,*k*), evaluating these positions separated by an angle of π/50 radians, prints these coordinates to the second line of the screen on the Dragon12 with a one second delay between each position.

```
%#codegen
% The above line is required to tell Matlab
C compiler
%*** find coordinates on a circle ********%
% define circle
r = 3; h = 4; k = 15; % define circle
radius and center positions
% Create a for loop from 0 to 2*pi in steps
of 2*pi/100
for angle = 0:2*pi/100:2*pi
   %*** define the path in tool space ***%
   x = r*cos(angle) + h;  % define tool x
                          %coordinate
   y = r*sin(angle) + k;  % define tool y
                          % coordinate
   %*** Print path position to the screen
***%
   mc_lcd_println('x= %2.1f,y= %2.1f',
x,y,1); % display current
        %position on the screen
   mc_timer_wait(1000); %delay for 1000
                        %mili-seconds
end
```

3.4 Inverse Kinematics

The joint space path information is generated from the tool space path points through the inverse kinematics of the mechanism. This example assumes that the robot consists of a two link planar arm with no offsets. This example presents the solution as a Matlab function that can be called to return the two joint angles of the robot arm when passed a they *x*,*y* coordinates of a desired tool space position. This function, when incorporated with other code portions, can be compiled and downloaded to the S12 processor.

```
% *** mc_2link_ik ***%
  % The Inverse kinematic solution for two-
link robot arm
  % call as [theta1, theta2] =
mc_2link_ik(x, y);
  % send in x,y values in cm, return theta1,
theta2 in degrees
function [theta1, theta2] = mc_2link_ik(x,
y)
      %*** initialize constants ***%
      l1 = 8.5; l2 = 10.25; % hard code
          %actual robot link lengths  in cm
      dtr = pi/180; % dtr = degrees to
                    % radians
% *** Begin inverse kinematic solution ***%
      % Calculate the second joint angle,
      % theta_2
      theta_2 =acos((x^2+y^2-l1^2-
l2^2)/(2*l1*l2));
      % Calculate the first joint angle,
      % theta_1
      theta_1 = atan2(y,x) -
atan2(l2*sin(theta_2),(l1+l2*cos(theta_2));
      % convert from rads to degs
      theta2 = theta_2/dtr;
      theta1 = theta_1/dtr;
end
```

3.5 Implementing actuators

The desired joint angles are now passed to the robot arm actuators. In general, three approaches exist for implementing these; 1) closed-loop position control through a commercial or hobby motor system (for example RC servo motors), 2) open-loop position control with stepper motors, 3) closed loop control from an external sensor and local motor control implementation. The first two cases are discussed next.

3.5.1 Stepper motor control: A stepper motor consists of a permanent magnet rotor with a series of alternating poles, and a stator consisting of two phases that, through bi-directional control, can provide alternating electromagnetic poles. The servo motor can provide open-loop position control with 360/200 degree resolution in full-stepping mode (both stator phases energized). The Dragon12 board has a built-in double

H-bridge driver with inputs connected to the four least significant bits of PortA. The stepper is driven in a step-wise fashion for a desired number of steps as demonstrated in the following code segment. This code segment demonstrates driving the stepper motor for one full revolution (200 steps). This code segment also demonstrates direct control of an MC9S12 output register (PORTA and DDRA). In many cases, it would be useful to have this capability embedded in a function (see the appendix) so students do not need the register-level information for this example. This is demonstrated in the next example.

```
%#codegen
% Program Stepper motor control - full step
function [ ] = main(  )
      DDRA = uint8(255);  % Set PORTA as
outputs
% define steps - full step
      step = double(ones(4,1));  % Initialize
      % step as a 4x1 ray of doubles
      % define the four unique steps for full
      % stepping mode
      % note: the Matlab function, bin2dec
      % converts a binary string to a decimal
      % number
      step(1) = bin2dec('00001010');
      step(2) = bin2dec('00000110');
      step(3) = bin2dec('00000101');
      step(4) = bin2dec('00001001');
      num_step = double(200);  % variable
       %containing the number of steps
% A for loop runs the stepper through
% num_step steps
      for i = 1:1:num_step
        PORTA = step(mod(i,4)); %mod(i,4) is
        % a Matlab function for modulo
        %divide, it returns the remainder
        %from integer division
        mc_timer_wait(100);  % delay 100 ms
        % to allow stepper to respond
      end
end
```

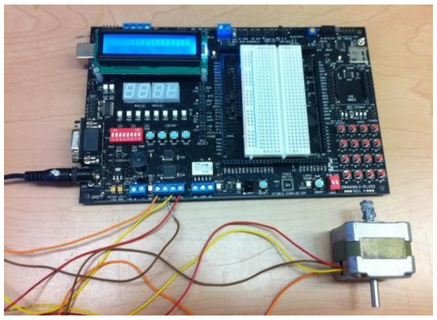


Figure 10: Stepper Motor Setup

3.5.2 RC Servo Motor Control: The RC servo is a common and inexpensive servo motor that incorporates feedback control to track a desired position or velocity. The RC servo takes as its input a Pulse Width Modulated (PWM) signal with a period of approximately 25 milliseconds and a high time varying from approximately one to two milliseconds. The PWM output on the MCS12 consists of a clock and two counters, one counter for the period and a second counter for the duty cycle. On the S12 this can be arranged into eight, 8-bit counters or four 16-bit counters. Generally speaking, the 16-bit counter would be required to use the resolution available on a typical hobby RC servo motor. Initialization of the PWM registers on the MC9S12 set up the clocks, define the PWM channel behavior including concatenation (16 bit counters rather than 8) and polarity. An example of this initialization in a Matlab function can be found http://matlab-nsfwiki.csc.tntech.edu/index.php/Servo. An example code segment that uses these functions and sends PWM signal to an RC servomotor connected to PortP4 (PP4) is shown below.

```
%#codegen
function [ ] = main()
mc_servo_pos_init();  % initialize the PWM
                      % function
% Send motor 5 to 0 degrees:
mc_servo_pos_set(5,0);
mc_timer_wait(1000);  % delay 1000 ms
% Send motor 5 to 90 degrees:
mc_servo_pos_set(5,90);
mc_timer_wait(1000);  % delay 1000 ms
% Send motor 5 to 180 degrees:
mc_servo_pos_set(5,180);
mc_timer_wait(1000);  % delay 1000 ms
```

3.6 Example course activities

A variety of hands-on activities can be constructed using the Matlab toolbox and embedded processor such as the MC9S12 on the Dragon 12. Several example activities are summarized in table II.

4. Discussion and Conclusions

This paper has presented a toolkit for programming microcontrollers using Matlab scripting language, with the Motorola MC9S12 MCU and hardware found on the Dragon12 evaluation board. This toolkit can be used to emphasize hands-on applications in mechanisms, robotics and other engineering courses. The toolkit allows students to implement embedded controllers in mechanical engineering applications with a basic understanding of Matlab programming. The toolkit can make use of a variety of functions that students commonly use when writing Matlab programs (examples include min, max, sort, polyfit). The toolkit allows students to interact with the microcontroller at a high level through a variety of functions for controlling input and output devices (see Appendix) or at a lower level through direct access to the microcontroller registers.

These examples presented in the paper have been implemented by the authors over a variety of courses at

TTU. Informal assessments of the student interaction with the project showed positive student response to having hands-on exposure to control of machines. Students that engaged in the hands-on, hardware-based programming activities reported a more improved experience with understanding of programming involving practical control and applications of machines. When opportunities exist in the engineering curriculum, these activities provide hands-on programming experiences to increase student engagement and understanding of engineering topics. The increased engagement appears to be due to the hands-on involvement, while the better understanding may in part stem from 1) increased engagement, 2) building on an existing framework of knowledge, and 3) seeing programming in multiple contexts (both hands on and desktop).

Table II: Example course activities

Class Example	**Task**	**Brief Description**
Pendulum dynamics	Calculate the value of gravity with a pendulu m	Construct a simple pendulum with a light rigid bar fixed on a pivot supporting a larger pendulum mass. Place an incremental encoder at the pivot to track pendulum motion. The pendulum motion can be measured over time and from this the pendulum period, an estimate of gravity can be determined from the pendulum dynamic model. http://workshop-nsfwiki.csc.tntech.edu/index.php/Pendulum
Security system	Design and operate a mechanic al latch for a toolbox	Design a simple mechanism to create a latch for a tool box, driven by an RC servo motor or stepper motor. Allow the latch to be opened through a key code entry on the dragon12 or other input. http://workshop-nsfwiki.csc.tntech.edu/index.php/Security_System
Robot Arm	Control of a simple robot arm	Construct a planar, 2dof robot arm using RC servo motors. Use kinematic models to operate the robot in tool space to draw shapes or letters. http://workshop-nsfwiki.csc.tntech.edu/index.php/Robot_arm
Mecanum-drive robot	Control of a mecanum-drive robot	Construct the mecanum drive robot with RC servo motors modified for full rotation and mecanum or omni wheels. Use kinematic models to drive the robot along desired paths. http://workshop-nsfwiki.csc.tntech.edu/index.php/Omni_Robot

Bibliography:

1) http://www.mathworks.com/products/matlab/, website of The Math Works, Inc. developer and distributor of technical computing software Matlab.
2) Raspberry Pi, http://www.raspberrypi.org
3) **Arduino, http://www.arduino.cc**
4) Arduino Support from MATLAB, http://www.mathworks.com/hardware-support/arduino-matlab.html
5) Yan-Fang Li, Saul Harari, Hong Wong, and Vikram Kapila, "Matlab-Based Graphical User Interface Development for Basic Stamp 2 Microcontroller Projects", in the Proceedings of American Control Conference, Boston MA, June 03- July 02, 2004.
6) Sang-Hoon Lee, Yan-Fang Li, and Vikram Kapila, "Development of a Matlab-Based Graphical User Interface Environment for PIC Microcontroller Projects", in the Proceedings of American Society for Eng0ineering Education Annual Conference & Exposition, 2005.
7) Benjamin J. Engle, John Watkins, "MATLAB Based DC Motor Control Lab" , Wichita State University December 4, 2007, http://engineering.wichita.edu/esawan/engle.pdf (accessed January 27, 2014)
8) Interfacing with matlab, http://mbed.org/cookbook/Interfacing-with-Matlab. (accessed January 27, 2014)
9) Introduction to Data Acquisition and Processing, Department of Electrical and Systems Engineering, University of Pennsylvania. http://www.seas.upenn.edu/~ese111/Fall2012labs/Lab5.pdf ((accessed January 27, 2014)
10) Freescale Codewarrior, www.freescale.com
11) Matlab to MCU programming wiki, http://matlab-nsfwiki.csc.tntech.edu/index.php/Main_Page

Appendix Summary of MHCS12/Dragon12 features

Function	Parameters /return value	Description
mc_atd_init	none	Initializes the ATD hardware. WARNING! This will overwrite any ATD settings set by 'mc_light_init'.
mc_atd_get	channelID (8 – 15) return level (0 – 255)	reads and returns the signal level on the selected ATD channel
mc_buzzer_beep	duration (ms)	Beeps the speaker if the SPK SEL jumper is set to PT5 (not normally used)
mc_encoder_init	none	Initializes the timer channels for use with the our encoders
mc_encoder_get	return encodercounts	Get the step count of the encoder. The will need to be calibrated and converted to determine actual angle. Reading is relative to the position of the encoder when the 'mc_encoder_init' function is called.
mc_ir_trigger	channel (8 – 15) threshold (int) return triggertime	Waits for the value on the specified ATD channel to exceed the given threshold. Then waits for the value to go back below the threshold. Returns the time when the threshold was triggered.
mc_lcd_init	none	Initializes the LCD screen
mc_lcd_println	text, linenum	Prints the specific text to the lcd on the specified line.
mc_led_init	none	Initializes the LED display
mc_led_set	led code (byte)	Sets the LED display. The low order byte of the led code input determines which LEDs with each bit representing the state of one of the LEDs.
mc_led_get	return led code (byte)	Read the current state of the LEDs.
mc_light_init	none	Initializes the ATD channel for the light sensor. WARNING! This will overwrite any ATD settings set by 'mc_atd_init'.
mc_light_get	return light value (0 – 255)	Reads the ATD channel associated with the light sensor and returns the result.
mc_math_rand	return random double	Returns a random value of type double.
mc_math_srand	none	Seeds the random number generator with the reading from the light sensor.
mc_pll_init	none	Initializes the PLL and sets the MCU clock to use it
mc_pwm_init	none	Initializes the default (5) PWM channel for general use
mc_pwm_duty	duty (0 – 1)	Sets the PWM duty for the default (5) PWM channel.
mc_serial_init	port, baudrate	Initializes the specified serial port to the given baud rate.
mc_servo_init	none	Initializes the PWM channels for use with a position servo. Deprecated in favor of 'mc_servo_pos_init'
mc_servo_pos_init	none	Initializes the PWM channels for use with a position servo. 16 bit PWM mode is used, so there are only the odd servo channels are

		valid for use (1, 3, 5, 7)
mc_servo_pos_set	servoID, position	Sets the position (in degrees) of the specified (1, 3, 5, 7) position servo. Calibrated for the servos on the omnirobot.
mc_servo_vel_init	none	Initializes the PWM channels for use with a velocity servo.
mc_servo_vel_set	servoID, velocity	Set the velocity (range = [-1, 1]) of the specified velocity servo. Calibrated for the servos on the omnirobot.
mc_speaker	frequency	Sets the current speaker frequency to the specified value.
mc_speaker_init	none	Initializes the speaker hardware
mc_speaker_set_freq	frequency	Sets the current speaker frequency to the specified value. Calls 'mc_speaker'
mc_speaker_set_pitch	hightime	Sets the current speaker pitch (specifically the half-cycle time of the output square wave = 750000/frequency)
mc_stop	none	Halts the MCU (actually, calls 'mc_timer_wait(100)')
mc_switch_init	none	Initializes the switches
mc_switch_get	return switchID (2-5)	Waits for a switch to be pressed and then returns the ID of the switch pressed.
mc_timer_init	none	Initializes the timer system
mc_timer_get	return time (double)	Gets the current elapsed time since the board was reset.
mc_timer_wait	none	Waits for the specified number of milliseconds
RealTimeInterrupt		Used by the encoder system
TC0Interrupt		Used by the encoder system
TC1Interrupt		Used by the encoder system