A Novel Racetrack Platform For Teaching Microcontroller System Design Concepts

Brinkley Sprunt

Electrical Engineering, Bucknell University

1 Abstract

The primary goal of the sophomore microcontroller system design course [7, 8] offered by Bucknell University's Electrical Engineering Department is for the students to gain experience with the key concepts of microcontroller-based system design. However, because this is a sophomore-level course, the students typically do not have substantial prior experience with assembly language programming, digital devices, or analog devices. As such, the first half of the course has traditionally been devoted to the development of assembly language programming skills and interfacing concepts for digital and analog devices. Consequently, many of the more complex, high-level concepts such as polling, interrupts, state machines, and control algorithms are not introduced until late in the course. This late introduction limits the students' exposure to these concepts and often prevents the students from employing these concepts in their final term projects. This paper describes changes being implemented for the next offering of this course that are intended introduce these more complex topics earlier. The motivation behind these changes is to improve the students' understanding of these more advanced topics by giving the students more experience with these topics earlier in the course.

To enable the introduction and use of the more complex microcontroller topics earlier in this course, a sophisticated hardware platform is being developed that employs memory-mapped I/O, LED displays, photo sensors, switches, analog-to-digital converters, timers, and interrupts. This platform is an electronic race car and racetrack set that supports two cars, variable speed controls, car-position sensors, as well as lap-time and total-time displays for each car. By providing the students access to this type of platform for the mid-term project, the students gain significant experience with both high-level microcontroller system concepts (e.g., interrupts) as well as low-level concepts (e.g., assembly language programming, LED interfacing) prior to beginning their final project for the course. To implement the software design for the racetrack system, the students will be grouped into several teams. The goal for each team is to create racetrack system software that is able to host a race between two cars while concurrently controlling one of the cars in the race. This paper describes the overall plans for this new course [8] and details how the features of the racetrack platform will be used to teach both high-level and low-level microcontroller system design concepts.

2 Background: Bucknell's Microcontroller System Design Course

Bucknell's Electrical Engineering Department has required a microcontroller system design course for EE majors since the mid-1990s. As originally offered [7], this course was a half-credit

course for sophomores in the fall semester. The goal for this course was to introduce the students to basic computer architecture concepts, assembly language programming, and microcontroller interfacing. Although this course was successful, it had a number of problems:

1. Although it was a half-credit course (meeting only once per week for three hours), the students often complained that they spent more time on this course than their full-credit courses.
2. Most students taking the course had little or no prior programming experience. Additionally, assembly language programming is probably the most difficult programming language to learn as a first programming language. As a result, students often wrestled with the simultaneous introduction of programming concepts and the precise requirements and low-level limitations of assembly language programming.
3. This course was the second EE course after the introductory EE course. As such, the students had only a brief exposure to digital logic prior to this course.
4. Because of the lack of programming experience, the first half of the course was traditionally devoted to assembly language programming skills and basic interfacing with digital and analog devices. Consequently, many of the more complex, high-level concepts such as polling, interrupts, state machines, and control algorithms were not introduced until late in the course. This late introduction limited the students' exposure to these concepts and often prevented the students from employing these concepts in their final term projects. The final term project is a significant microcontroller system design to be specified and implemented by the students.

To address the first three problems described above, the following changes were made to the sophomore EE curriculum:

- The microcontroller system design course was changed from a half-credit fall course to a full-credit spring course. The change to a full-credit course alleviates some of the time pressure for the students and provides much more in-class time. Previously, the students would only meet for one three-hour combined lecture and lab section per week. The switch to a full-credit course provides three hours of lecture and three hours of lab per week.
- A high-level programming course in C++ was added to the fall semester. By introducing high-level programming concepts in a separate, fall course, the students will have programming experience prior to taking the microcontroller course in the spring. This allows the simultaneous introduction of both programming concepts and assembly-language programming to be avoided in the microcontroller course.
- The spring-semester circuits course was split into two half-credit courses, one in the fall and one in the spring. This provides the students more exposure to circuits as well as lab techniques and equipment prior to and concurrent with the microcontroller course.

To address the problem of the late introduction of complex, high-level microcontroller system concepts (the fourth problem listed above) the early lab assignments for the course will be updated and a new hardware platform will be introduced for the mid-term project.

Previously, the early lab assignments focussed entirely upon assembly language programming. Lab assignments that required the use of polling, interrupts, or the the interfacing of the microcontroller to other devices did not occur until five or six weeks into the semester. With the additional lab time, the new version of the microcontroller course introduces these techniques and concepts along with assembly language programming in the initial lab assignments.

The cornerstone of the changes to the microcontroller course (and the primary topic of this paper) is the introduction of a new hardware platform for the mid-term project. Previously, the mid-term project was primarily a large, assembly-language programming project that required only simple interfacing and made no use of interrupts. For example, the previous mid-term project was to create a program that played the *Pong* video game on a terminal emulator [6]. The goal of the mid-term project was to give the students a substantial and interesting assembly language programming task to solidify the assembly language programming concepts introduced in the early lab assignments. The new hardware platform for the mid-term project employs memory-mapped I/O, LED displays, photo sensors, switches, analog-to-digital converters, timers, polling, and interrupts. This platform is an electronic race car and racetrack set that supports two cars, variable speed controls, car position sensors, as well as lap-time and total-time displays for each car. By providing the students access to this type of platform for the mid-term project, the students will gain significant experience with both high-level microcontroller system concepts (e.g., interrupts) as well as low-level concepts (e.g., assembly language programming, LED interfacing) prior to beginning their final project for the course.

To implement the software design for the racetrack system, the students will be grouped into several teams. The goal of each team will be to develop racetrack software to host a race (track the lap times for each car) and compete in a race (control the speed of one car in a race). Upon completion of the racetrack system software, the teams will compete against each other in a series of race competitions. The hardware and software for the racetrack project lend themselves well to dividing the racetrack software development into two sub-projects. The tracking and display of the lap times for each race car is one sub-project and the tracking and speed control of a race car is the other sub-project. Also, since two microcontrollers are needed for each racetrack platform, the two groups working on each of these sub-projects can work concurrently with minimal interference using the same racetrack platform. As such, each racetrack team will be encouraged to split the racetrack software development into these sub-projects and merge the sub-projects to complete the racetrack system. This will give the students experience working as members of a team on separate components while still being responsible to the team for the overall completion of the project.

The outline of the remainder of this paper is as follows. First, the racetrack platform and its capabilities are described. This is then followed by a description of the early lab assignments that lead up to the mid-term racetrack project. The racetrack project is then described. The paper closes with a summary section.

3 Racetrack Platform

This section describes the objectives of the racetrack platform along with the track, the microcontroller development board, and the microcontroller's inputs and outputs.

3.1 Racetrack Objectives

The racetrack platform was conceived to meet the following objectives for a microcontroller-based system:

- The high-level function of the system should:
  - be easily understood without a technical background.
  - be engaging.
  - require a significant level of baseline functionality.
  - provide opportunities for several different software implementation approaches.
  - provide opportunities for optimizations beyond the baseline requirements.
  - allow for competition between different teams.
- The system to be controlled should be composed of various hardware devices, not a terminal emulator as had previously been used for the mid-term projects [6].
- The devices being controlled should generate interrupts that require acknowledgement.
- The use of timers and the tracking of elapsed time should be required.
- A variety of input and output devices should be used (LEDs, digital-to-analog converters, motors, photo sensors, switches, etc.).

A system for an electronic racetrack meets the above objectives well. An electric race car set is a familiar, engaging system. A microcontroller-based design that is able to host a race between two cars as well as control one of the cars in the race requires a significant level of baseline functionality and provides opportunities for different solution approaches and optimizations (e.g., completing a three-lap race quickly without flying off the track). The racetrack system also allows different teams to compete against one another. Furthermore, a variety of devices are used to implement the system's inputs and outputs. To use these devices effectively, two interrupt sources are necessary: timer interrupts to track the race times for each car and photo sensors to track the position of the race cars.

An electronic racetrack system also has another significant advantage over other possible projects such as those that involve robots [2]: the entire functionality of the racetrack system can be demonstrated and tested in a short amount of time. A good, three-lap race using the racetrack layout described below takes less than ten seconds total (about three seconds per lap). As such, demonstrations of a team's racetrack system and competitions between teams can be completed quickly.

3.2 Racetrack Layout

The layout of the racetrack is shown in Figure 1. The track is a modified "figure-8" configuration. The intersection of the "figure-8" has a vertical loop. One of the "figure-8" loops is larger than the other, providing two long straightaways to the "figure-8" intersection. Also, the larger

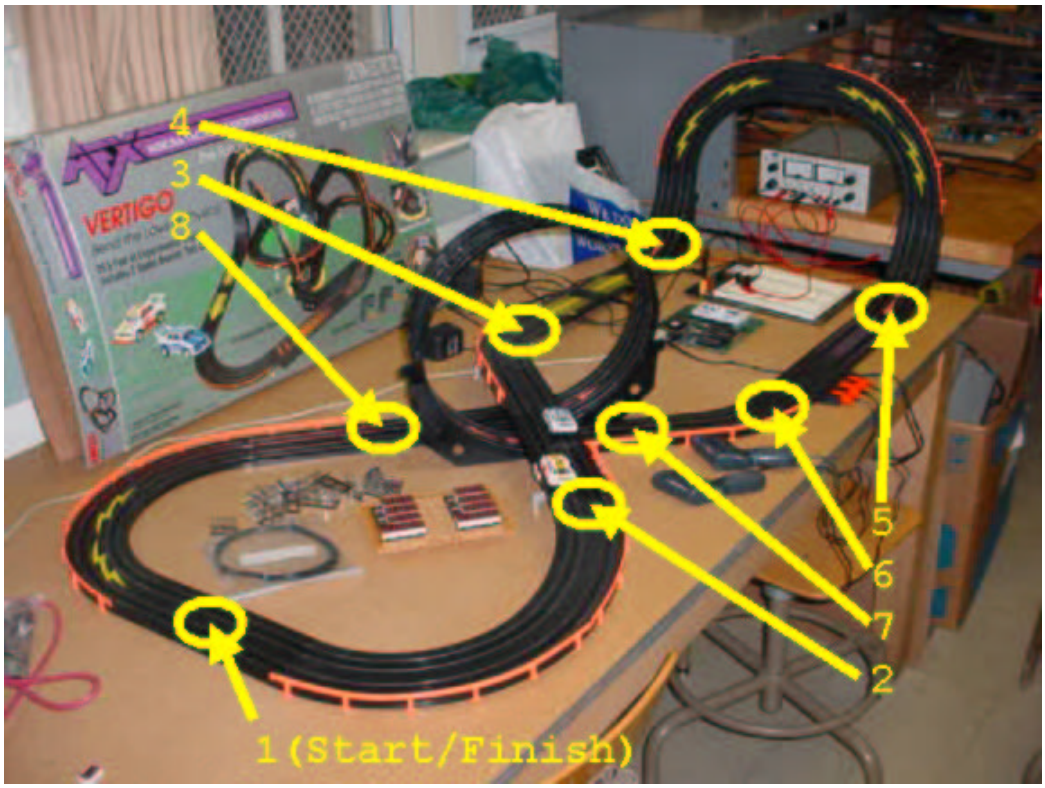

Figure 1: Racetrack Layout.

"figure-8" loop has a vertical curve. Each major section of this layout has different characteristics and, therefore, different maximum car speeds (if the car speed is too high, the car easily flies off the track). However, there is a constant car speed, slower than the maximum speeds for each section, that allows one to successfully complete the entire track without crashing.

Figure 1 also highlights the locations of the car position sensors placed along the track. Each location noted in Figure 1 represents two sensors, one for the left track and one for the right track. Eight pairs of sensors are present around the track at points indicating the beginning of a different type of track section, allowing the microcontroller to set the car speed as a function of track location. It is also handy having only eight sensors per track such that the status of all eight sensors can be read using an 8-bit I/O port on the microcontroller, with each bit representing a different sensor.

This track was constructed from the TOMY AFX Vertigo Race Set [4] (see Figure 2) obtained from JLP Merchandising's Model-Auto Slot Car Racing [3].

3.3 Microcontroller and Development Board

Two Axiom CME-11E9-EVBU boards [1] with Motorola's M68HC11 microcontroller [5] and 32KB of memory are used for the racetrack system: a host board and a guest board. The host board is responsible for starting and stopping the race and tracking the lap times and total times for each

Figure 2: TOMY AFX Vertigo Race Set.

race car. The host board is also responsible for "driving" one of the race cars. The guest board is only responsible for "driving" the other race car.

One might think that the computing load for the host board would put it at a disadvantage compared to the guest board when considering the computing cycles needed to manage the "driving" of its race car. However, the "overhead" for managing the race is low for the host board and the computing cycles needed to "drive" a car are also low. As such, this system provides a demonstration to the students of how capable a 2MHz, 8-bit microcontroller can be.

Another advantage of this two-board setup for the racetrack system is that a team comprised of two groups can be simultaneously working on the racetrack system. For example, the group developing the host functions for tracking lap times and total times can be working on one board and one racetrack lane and the group developing the driving functions can be using the other board and the other racetrack lane. As such, one racetrack system is sufficient to keep a team of four to five students busy.

3.4 Microcontroller Inputs and Outputs

To support the features of the racetrack system, numerous inputs and outputs are supported for the microcontroller boards. The microcontroller inputs from the racetrack are as follows:

- The host and guest boards each have a set of eight DIP switches for configuration purposes. The function of some of these switches will be specified (e.g., to indicate host or guest mode, driving the left or right track, etc.). However, several switches will be left unspecified and available to the racetrack software developers (e.g., to specify a "safe" or "aggressive" race mode).
- The host board is connected to a race reset/start button. When the reset/start button is first pressed, the host resets the system and prepares to start a new race (e.g., all the time displays are turned off). When the reset/start button is pressed a second time, the host begins a countdown light sequence to start the race.

- Both the guest and host board have eight car-position sensors each. When any car-position sensor is activated, both the guest and host boards will be interrupted. These boards can then poll a host-position port and/or guest-position port to determine which car or cars just tripped a position sensor and which sensor or sensor(s) were tripped. In order to "drive" one car, only the position sensors for that car need to be monitored. However, the position sensors for both cars are hooked to each microcontroller to support the implementation of adaptive racing algorithms that adjust the speed of a car based upon the opponent's position.
- The speed control for a car is created using a DAC (digital-to-analog converter) to control a power transistor that provides current for the car (see the discussion of the microcontroller outputs below). The output voltage from each car's speed-control DAC is also fed into the M68HC11 A/D converter on the guest and host boards, allowing each microcontroller to ascertain how *hard* its opponent's car is being driven. As with the ability to monitor the position sensors of the opponent's car, this capability is not necessary to "drive" a car, but it does support the implementation of adaptive racing algorithms that adjust the speed of a car depending upon how hard the opponent's car is being driven.

The microcontroller outputs to the racetrack are as follows:

- The racetrack supports 26 7-segment displays as shown in Figure 3. These displays are used for the lap and total times for each track. As the race starts, the Lap 1 display and the total time display begin counting in increments of 1/100 second. As each car passes the starting position sensor, the microcontroller freezes the current lap time and begins updating the next lap time display while always updating the total time display. When one car completes three laps, its displays are frozen. When both cars have completed three laps, the race is over and the winning total time is flashed on the winning car's total time display.

  These displays are also used for the light sequence to start a race. This light sequence begins when the reset/start button is pressed after the racetrack system has been reset. For the light sequence, the right-hand segments of the right-most 7-segment displays for each track are lighted in sequence starting with the upper right segments until all the right-hand segments are illuminated. At this point, the race begins and the host tracks the lap and total times for each race car.
- Two 8-bit speed-control ports are implemented, one for the host and one for the guest. Each of these ports is connected to an 8-bit DAC which is used to control a power transistor that supplies the current for a race car. By writing different 8-bit values to a speed-control port, one can control the speed of a race car. This configuration allows either continuous or pulse-width modulation control of each race car's speed. As mentioned previously, each car's DAC output voltage is also provided as an input to the M68HC11 A/D converter on the opponent's microcontroller board.

  As an aside, the racetrack circuit is designed to allow the original speed-control guns to be used to control a race car, instead of a microcontroller board. This enables races between a human and a microcontroller as well as between two microcontrollers.
- Each microcontroller has an output port that is used to acknowledge the car-position sensor interrupts generated as the cars race around the track. A write to the interrupt-acknowledge port causes the car-position sensor interrupt to be de-asserted.

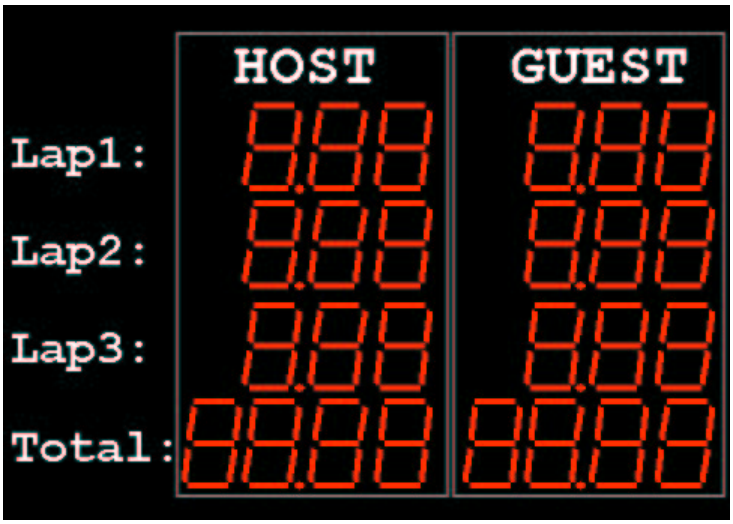

Figure 3: Lap Time Displays.

To support the inputs and outputs described above for the racetrack system, a set of I/O boards are being developed for use with the CME-11E9-EVBU microcontroller boards. Each of these I/O boards provides 16 output ports and 4 input ports. 13 of the output ports (corresponding to the lap and total time displays for one race car) are connected to 7-segment displays. The undedicated output ports and the input ports are used to implement the other input and output functions described above. Thus, the host microcontroller uses two of these I/O boards, one for the host displays and one for the guest displays. Another I/O board is also used to provide the guest microcontroller board the non-display input and output ports described above (even though the guest microcontroller board is not required to drive any 7-segment displays).

4 Introductory Lab Assignments

The first five lab assignments for the microcontroller course provide the foundation for the mid-term racetrack project. These lab assignments combine the introduction to assembly language programming and debugging with the use of output ports, input ports, LED displays, switches, polling, interrupts, and timers. The topics covered by these lab assignments are as follows:

1. **Introduction and Output Ports.** In this lab, students learn how to:
   - Configure the microcontroller board and connect it to the host PC.
   - Edit, build, and execute simple assembly language programs.
   - Interface a 7-segment LED display to an I/O port and control it via an assembly language program.
2. **Addressing Modes and Input Ports.** In this lab, students learn how to:

- Use each of the M68HC11 addressing modes with a series of programming exercises that manipulate simple arrays.
- Interface several single-pole, double-throw switches to an M68HC11 input port and use the data read from the input port to control a 7-segment LED display.

3. **Debugging, Revision Control, and Simple Interrupts.** In this lab, students learn how to:

   - Use an assembly program listing and the M68HC11's monitor commands to debug assembly language programs.
   - Use a revision control system to record and recover copies of revisions made to an assembly language program during its development.
   - Use a simple interrupt to count the number of times a switch is thrown and display the count using a 7-segment LED display. The students are also introduced to the problem of switch bounce and several techniques to de-bounce switches.

4. **Conditional Branches and Subroutines.** In this lab, students learn how to:

   - Use conditional branches to control the flow of an assembly language program.
   - Use subroutines to provide structure to a program and encapsulate commonly used procedures.
   - Use conditional branches and subroutines to count the number of asserted switches on an input port and display the count on a 7-segment LED display.

5. **Timer Interrupts and Digital-to-Analog Converters.** In this lab, students learn how to:

   - Configure the M68HC11 timers to provide interrupts at regular, specified intervals.
   - Interface the M68HC11 to a DAC.
   - Use the timer interrupts and DAC to produce sine, ramp, and saw tooth waves of various frequencies on an oscilloscope.

5 Racetrack Project

By the completion of the introductory lab assignments discussed above, the students have been introduced to the majority of the assembly language constructs, microcontroller interfaces, and the high-level microcontroller system design concepts (i.e., interrupts and polling) that will be needed to implement the racetrack system. The mid-term racetrack project is a substantial project that requires the students to pull together and employ the concepts from these lab assignments. Four lab periods (and five calendar weeks) are allocated to the mid-term project. Upon successful completion of the mid-term project, the students should be able to specify and implement a microcontroller-based system of their own choice for the course's final project.

The starting point for the racetrack project consists of the following:

- The racetrack platform described previously in Section 3.

- A set of suggested steps for implementing the software for the racetrack system. These steps allow for incremental development and testing of the racetrack functionality.
- A skeleton assembly program for implementing the racetrack software. This skeleton program presents the students with the high-level structure of the racetrack system (the bones) and leaves the low-level details (the meat) to be implemented by the students. This skeleton structure also serves as the starting point for the suggested implementation steps mentioned above.

Each racetrack development team consists of four or five students. The suggested implementation steps recommend that the team be split into two groups: the timer group and the race group. The timer group is charged with implementing the host support for tracking the lap and total times for each race car and displaying them using the 7-segment displays. The race group is charged with implementing the support for tracking the car's position on the track and controlling the car's speed. As mentioned previously, the work of both of these groups can proceed concurrently using one racetrack platform. As the groups complete their implementations, the whole team is responsible for merging the two implementations and producing a working racetrack system.

The requirements for a successful racetrack system are as follows:

- To be able to host multiple races from start to finish, accurately displaying the lap and total times for each race car while safely controlling the speed of the host car such that it completes races without crashing.
- To be able to participate in a race as a guest car and safely control the guest car such that it can complete races without crashing.

As an option, each team that implements a successful racetrack system can participate in a competition to determine the fastest racing team. Each member of the winning team will receive a small trophy. Also, a picture of the winning team will be archived on the web page for the microcontroller course. Participation in the competition is not required and is not graded.

6 Summary

Bucknell's microcontroller system design course is undergoing a number of changes to improve the quality and depth of the course. The overall goal of these changes is to introduce the high-level microcontroller system design concepts (e.g., interrupts, polling, state machines) and interfacing techniques earlier in the course by combining their introduction with the introduction of assembly language programming. Chief among the changes being made is the incorporation of a sophisticated hardware platform for the mid-term project. This platform is a microcontroller-based system for an electronic race car and racetrack set. This racetrack system supports two cars, variable speed controls, car position sensors, as well as lap-time and total-time displays for each car. The mid-term project and racetrack system serve to solidify the assembly language programming, microcontroller interfacing, and high-level microcontroller system concepts that are introduced in

the initial lab assignments. For the mid-term project, the students must create an assembly language program to host a race between two cars, track and display the lap and total times for each car, and "drive" one of the cars during the race. The successful completion of the mid-term project prepares the students well for their final project, a significant microcontroller system design that is specified and implemented by the students.

References

[1] Axiom Manufacturing.
CME-11E9-EVBU MC68HC11 Low Cost Development System.
http://www.axman.com/Pages/neuprod8_cme11e9.html.

[2] Trinity College.
Trinity College Fire-Fighting Home Robot Contest.
http://www.trincoll.edu/events/robot/.

[3] Model-Auto Slot Car Racing. *JLP Merchandising's Model-Auto Slot Car Racing Home Page*.
http://www.mascr.com/.

[4] Model-Auto Slot Car Racing. *TOMY AFX Vertigo Race Set*.
http://www.mascr.com/108/884.htm?827.

[5] Motorola.
Motorola M68HC11 Reference Manual.
http://e-www.motorola.com/brdata/PDFDB/docs/M68HC11RM.pdf.

[6] Brinkley Sprunt.
Elec 246: Develop a program that plays pong.
http://www.eg.bucknell.edu/~bsprunt/classes/elec_246/Labs/lab6/lab6.htm.

[7] Brinkley Sprunt.
Elec 246: Microcontroller System Design.
http://www.eg.bucknell.edu/~bsprunt/classes/elec_246/elec_246.htm.

[8] Brinkley Sprunt.
Elec 247: Microcontroller System Design.
http://www.eg.bucknell.edu/~bsprunt/classes/elec_247/elec_247.htm.

Brinkley Sprunt

Brinkley Sprunt is an assistant professor of Electrical Engineering at Bucknell University. He received his B.S.E.E. degree from Rice University and his M.S.E.E. and Ph.D. degrees from Carnegie Mellon University. Prior to joining Bucknell, he worked as a computer architect at Intel Corporation for nine years on the development of several Pentium-family processors.