Software Design of the Stiquito Micro Robot

Andrew McClain and James M. Conrad
University of North Carolina at Charlotte
jmconrad@uncc.edu

Abstract

The Stiquito robot is a small, six legged robot that walks by utilizing a special material called Flexinol. Flexinol acts as a muscle for each of the legs, providing a means of walking for the robot. Stiquito was originally designed as a manually controlled device, where the operator would manually control the six legs in sets of three. A self-controlled version of the Stiquito was later developed. This version provides more versatility in possible applications by allowing it to travel alone, without being tethered to an operator.

This self-controlled version of the Stiquito robot uses a microprocessor to coordinate the operation of the legs to produce a forward motion. To accomplish this, an algorithm and resulting software must be developed to satisfy several operating requirements. This paper discusses these requirements and the development of the C/C++ software designed to implement them.

1. Introduction

The Stiquito robot was designed as a 6-legged robot which uses a special metal alloy as a "muscle" to provide means of locomotion. The first two Stiquito books describe the design criteria and procedures for creating this robot [1, 2]. One of the driving forces behind this robot, however, is for use as an instructional device for engineering students. For this purpose, continual improvement and expansion of the robot's design and capabilities are always being researched.

One of the largest leaps in capabilities of the robot was the inclusion of a printed circuit board on the back of the Stiquito robot body. This PCB is connected to the Flexinol wire attached to the legs of the robot and contains a microcontroller, LEDs, a transistor array, and the passive components required to operate these components.

With the addition of this board, it was necessary that software be designed and written, and then programmed into the microcontroller. This code must coordinate all of the controller's output pins corresponding to the legs of the robot in such a way that the robot can move by itself. Due to the motion of the legs and the desire to keep power usage at a minimum, several requirements must be designed into the software.

2. The Original Stiquito Robot

The original Stiquito robot is quite simple in design when compared to its newest generation. Developed by Jonathan Mills of Indiana University, the original Stiquito consists of a plastic body, piano wire legs, and a single strand of Flexinol attached to each leg [1, 2]. These Flexinol strands act as a muscle to pull the leg toward the back of the Stiquito body, propelling the robot forward. When power is taken away from the leg, the Flexinol slowly expands, and the natural position of the piano wire leg pulls the leg back toward its forward position. Because there is no method of lifting the leg off of the walking surface, the "feet" of the robot are bent toward the rear of the robot. This creates a ratcheting effect between the foot and the walking surface, due to the back-and-forth motion of the leg.

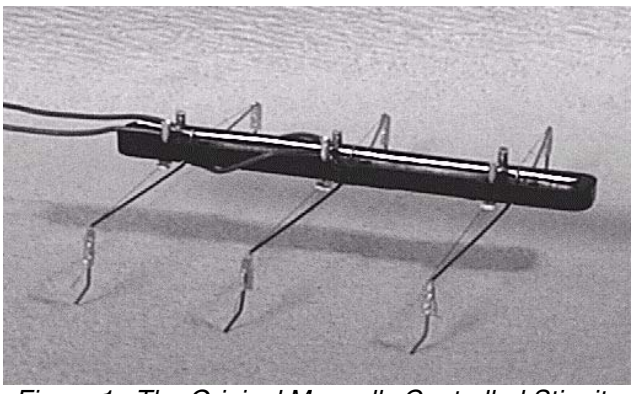

Figure 1: The Original Manually Controlled Stiquito

This design is quite capable of moving, but is also open to a great deal of improvement. One of the major disadvantages of this design is that the legs have constant contact with the walking surface. This is not a problem when a leg is being pulled back, but when the Flexinol is relaxing, the leg's return motion can pull the robot backwards, negating the walking motion of the other legs. Although the ratcheting effect greatly reduces this problem, it is still very tedious to attempt control of the robot on a very sticky surface, such as a tablecloth.

Another disadvantage of this original design, and perhaps the one most focused upon with the addition of the microprocessor, is that the robot must be manually controlled. This is done in the original design by a small wire device. With this device, the operator manually

0-7803-8865-8/05/$20.00 ©2005 IEEE.

applies electric power to alternating sets of the Stiquito's tripods. This limits the applications for Stiquito, as it requires that an operator be in close proximity to operate it. It is much more desirable to have the Stiquito fully automated, so that it can travel into tight or hazardous spaces where a human could not go.

3. A New Generation of Stiquito

The problems of the original Stiquito design were addressed in the design of a newer Stiquito. To solve the problem of the leg never lifting from the walking surface, a second strand of Flexinol was attached to the leg on the forward side. The anchor for this strand was mounted on the top of the Stiquito body. When this strand is contracted, it pulls the end of the leg up and forward. This lifts the leg off of the surface and pulls it forward, instead of allowing it to drag forward on the walking surface. This motion is referred to as two-degree of freedom, due to the back and forward control.

To make the robot autonomous, a printed circuit board was designed to fit on top of the robot. With the addition of brass screws instead of the crimps used in the original design, the design of the circuit board and its attachment to the robot is simplified. With some simple wiring and a few solder joints, the circuit board can be attached to either a standard Stiquito or a two degree-of-freedom robot.

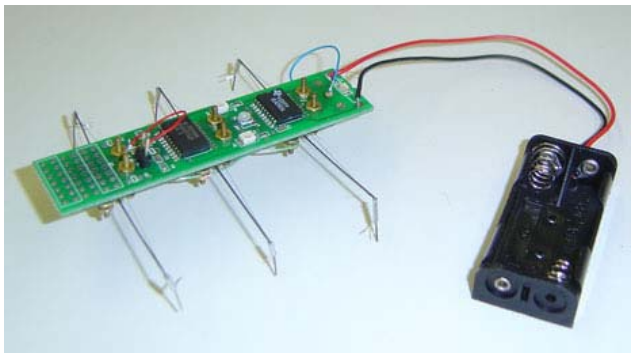

Figure 2: A Two Degrees-of-Freedom Stiquito

These critical additions were designed and implemented by Dr. James Conrad with the help of his embedded systems students at both the University of North Carolina at Charlotte and North Carolina State University [3].

4. Stiquito Operation

The six legs of the Stiquito robot are divided into two tripods, with each tripod including two legs on one side of the robot, and one on the other. This division provides for a more smooth motion without the complexity of having to control each leg individually.

The single degree-of-freedom Stiquito walks by alternatively activating each of these tripods. While the first tripod is being activated, the second is in a "relaxing" state, where it is returning to the position where the leg is perpendicular to the body. The second tripod is then activated, and the first is allowed to relax. Figure 3 describes the motion of the Stiquito in one degree-of-freedom mode.

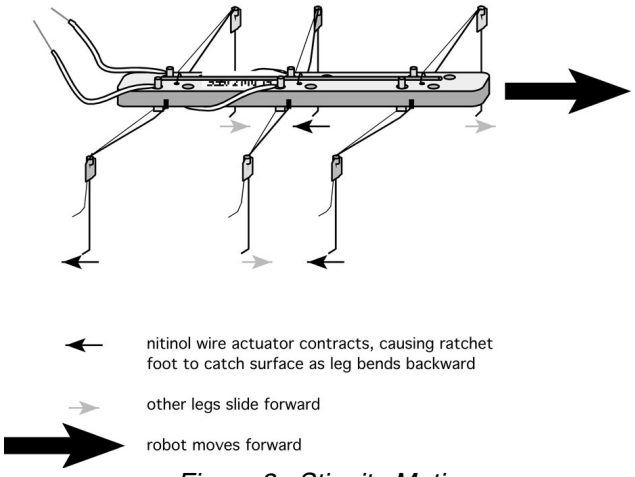

Figure 3: Stiquito Motion

For the two degrees-of-freedom Stiquito, the same procedure is performed. The difference is that a second set of Flexinol is activated at the completion of a tripod's activation period. When one tripod's contraction is nearly complete, the secondary Flexinol for that tripod is activated, lifting those legs off the surface before the leg has a chance to move forward to the relaxed position.

5. The TI MSP430 Microcontroller

The Texas Instruments MSP430F1122 microprocessor was chosen for use with Stiquito due to its small size, versatile features, and low power usage. Many of the useful features of this microcontroller are:

- Low Power Consumption
  - Active Mode ~200 µA
  - Standby Mode 0.7 µA
- 16-Bit Timer
- 10-Bit, 200-ksps A/D Converter
- Serial Onboard Programming (JTAG)
- Supply Voltage Brownout Protection
- 4KB + 256B Flash Memory (256B RAM)

Each of these features is important to the operation of the microcontrolled Stiquito robot. The Stiquito is designed to operate on battery power, so power consumption is an important consideration. The low power modes of the MSP430 allow the processor to sit in an idle state as much as possible.

The 10-Bit Analog-to-Digital converter is another very important feature. This allows the speed to be controlled by reading a voltage from the on-board potentiometer. The A/D converter is only active when a speed measurement is being performed. This maximizes the time that the microcontroller can remain in low-power standby mode.

The 4 Kilobytes of flash memory is more than sufficient for the code required for movement of the Stiquito. In fact, nearly 70% of this memory is free after programming of the standard Stiquito software, leaving plenty of room for new features and functionality to be added.

The on-board JTAG capability is an important tool for the development of software for Stiquito. Using the free software provided by Texas Instruments, a developer can control the operation of the microprocessor by stepping though code and watching variables.

One of the most important features of this chip is the 16-Bit Timer. This timer triggers the interrupt that "wakes" the microprocessor from standby mode whenever an output needs to be changed. The rate at which this interrupt occurs is determined by the value measured from the potentiometer by the A/D converter. The Timer Interrupt Service Routine contains the bulk of the operational code, where the microprocessor decides which tripod outputs should be moving.

6. Software Requirements

With the addition of the microprocessor to the Stiquito robot, software is required to coordinate the movement of the robot. As with any project, requirements were created to ensure the desired operation. Primarily, these requirements relate to power consumption and the leg movement attributes. The software requirements are as follows:

1. To reduce power consumption, the MSP430F1122 microprocessor should operate in low power standby mode with interrupt support.
2. Also for power consumption, the processor should operate at the slowest possible speed.
3. Software must check the state of the degrees-of-freedom jumper and alter the mode of operation to reflect the state of this pin.
4. The two degrees-of-freedom leg muscles (legs that pull the feet up and forward) will be powered alternatively from the primary leg muscles so that no more than two sets of legs are active at any given time.
5. Timers will be used to determine activation of the leg pairs instead of "wait loops."
6. The software must also check the output from the on-board potentiometer to calculate and change the speed of operation.
7. The period of one complete cycle should range from 3 seconds to 11 seconds in 32 possible increments.
8. For each activation cycle, the output signal must first be pulsed at double frequency. This activates the Flexinol and begins its contracting action.

7. Software Design

With these requirements in mind, the design of the software could begin. The first requirement was satisfied by the following line of code:

```
_BIS_SR(LPM0_bits + GIE);
```

This command is executed after initialization, and instructs the processor to power down and wait for interrupts. The processor automatically "wakes up" whenever the timer interrupt occurs, and automatically returns to standby when the execution of that interrupt service routine is completed.

The second requirement is satisfied with the following pair of instructions:

```
BCSCTL1 &= ~(RSEL0 + RSEL1 + RSEL2);
DCOCTL  &= ~(DCO0 + DCO1 + DCO2);
```

These commands set different internal clocks to their slowest respective speeds. Both of these instructions and the one for the standby mode can be found in the sample programs provided with the IAR Kickstart software provided by Texas Instruments.

Requirement 3 is satisfied in the "set_new_speed()" function. This function is called during the initialization phase and between each stage of operation. The function will check the state of the pin by masking the port and checking the result. The degree-of-freedom is then recorded in the global variable "degree". This variable is then checked in the timer ISR to determine which legs to toggle.

Satisfying requirement 4 was a great deal more complex. A separate switch statement was first created to include operation of the secondary tripods. To alternate activations, the secondary tripod is delayed by one step at the beginning of the stage in which it is first activated. Operation is then continued normally until the stage following the last stage in which that tripod is activated. During the first step of this stage, the leg is then toggled once more. If this remaining toggle is not included, the leg will be "stuck" in the activated position, where it would remain until the next stage in which it is activated. This is obviously not acceptable, as it would most likely damage the Flexinol wire, excessively drain the battery, and would severely disrupt the locomotion of the robot.

Due to the ability of the robot to dynamically change between one and two degrees-of-freedom motion, a check must also be included in the "set_new_speed()" function. When the robot is changing degrees-of-freedom, the appropriate legs are manually shut off or activated.

Requirement 5 is met by simply locating the leg movement instructions in the timer interrupt service routine. Having one and two degrees-of-freedom capability means that separate switch functions are included in the software. To increase the readability of the code, these switch functions are located in separate functions, which are then called by the timer ISR based on the current degree-of-freedom.

The speed control defined by requirements 6 and 7 are met by the instructions in the "set_new_speed()" and "set_speed()" functions. The "set_new_speed()" function is called at the end of every stage, or 16 times per complete cycle of the legs' motion. This function in turn calls the "set_speed()" function, which reads the value of the built-in 10 bit analog-to-digital converter. The A/D converter is set to read input port 2, pin 0 by default in the microprocessor's hardware. Therefore, no software is needed for this in the initialization procedures.

The "set_speed()" function then performs some integer arithmetic to determine the timer register value needed to produce the corresponding cycle time. Because the Stiquito checks the potentiometer value at the end of every stage, or 16 times per cycle, the total cycle time will change several times during a cycle if the potentiometer is being adjusted. The important aspect of this function is the calculation of the register value. To be exact, floating point arithmetic is needed. This, however, forces the compiler to include floating point functions, which drastically increases the size of the program. Because of this problem, an accurate integer-only formula was derived to calculate the timer register value. The accuracy lost by not using floating point arithmetic is negligible, especially when considering the accuracy of the potentiometer.

Finally, requirement 8 is implemented by running the timer at twice the speed needed to toggle the outputs in the normal output stage. At the first stage of activation, a tripod is toggled with every timer interrupt. After the first stage is complete, it then toggles every other interrupt occurrence.

8. Results of Implementation

The resulting code is a complete program that satisfies all of the requirements set forth at the beginning of the development process. Figure 4 shows one complete walking cycle in two degree-of-freedom operation.

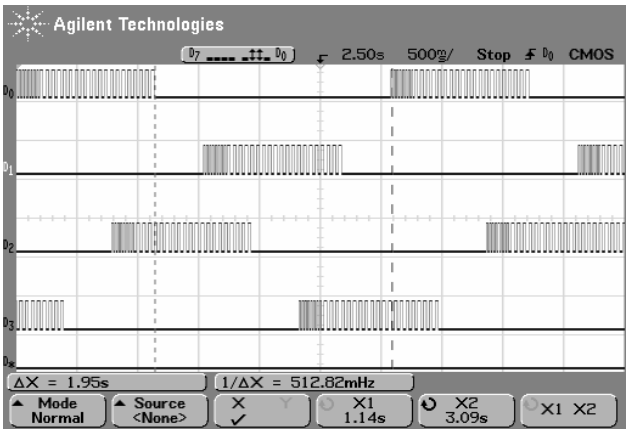

Figure 4: Stiquito Tripod Outputs

To obtain this image, the digital inputs of a mixed signal oscilloscope were attached to the output pins of the processor on a functioning, but not assembled Stiquito PCB (no Flexinol legs attached.) The speed for this test was arbitrarily set to a 3 second cycle.

In this screen capture, the first two signals (D0 and D1) represent the two main tripods of the Stiquito robot. The second two signals (D2 and D3) represent the secondary tripods which pull the legs up and forward. The D2 signal is the secondary tripod which is linked to the D0 primary tripod. It can be seen from this graph that the secondary tripod begins lifting the leg before the leg has been fully pulled back. This prevents any relaxation of the primary "muscle" while the foot is on the ground, causing the robot to pull itself backward.

Figure 5 shows a smaller time frame of the same cycle. In this screen capture, it is easy to see the alternating activations of the primary and secondary leg muscles. This shows the resolution of requirement 4.

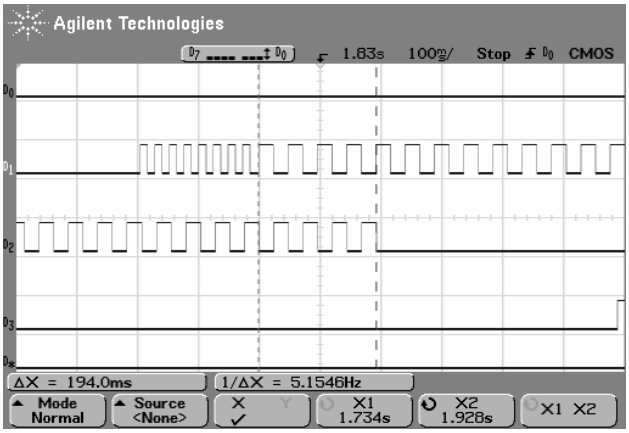

Figure 5: Single Tripod Activation

The single degree-of-freedom motion is slightly different from the two degree-of-freedom motion shown above. Besides not having secondary tripods, the time between activations of the primary tripods is extended.

This is necessary because the leg must return to its resting position before activated again. This provides a full range of motion for the leg. This takes longer than with the two degree-of-freedom operation because there is no secondary muscle forcing the leg back to its rest position. This functionality can be seen below in Figure 6.

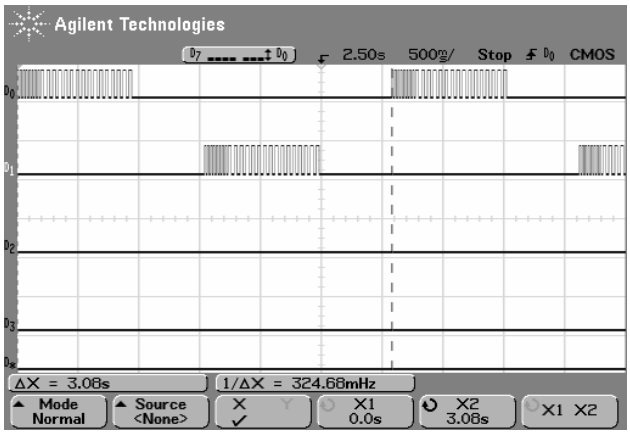

Figure 6: One Degree-of-Freedom Operation

These screen captures also show the implementation of requirement 8. The quick pulsing at the beginning of an activation stage can be easily seen in Figure 5. Due to the double toggling of a newly activated tripod, there will be some small periods where two tripods are active at the same time. Unfortunately, this cannot be avoided. The intention of this requirement is still upheld; however, as the amount of time that two tripods will simultaneously be active is very small.

9. Conclusion

The development of the Stiquito software shows many of the obstacles present in the development of software for an embedded system. With careful planning, the required development time can be reduced, and errors can be prevented.

One of the most important aspects of the Stiquito robot is its expandability. Just as the original design was expanded to a micro-controlled version, this new design can be enhanced. There are open ports that can be used for additional sensors or output devices. Examples of additional devices would be a proximity sensor to stop the walking motion when a wall is reached, or a radio transmitter to pass information to other Stiquitos or a base station.

Different versions of the software were produced, with each varying in the amount of functionality supported. These versions range from the fully functional code described here to single degree-of-freedom only versions with no speed control. The compiled versions of these packages range from 1,216 bytes to only 544 bytes, respectively. These "stripped" versions of the code would particularly be useful to a developer adding functionality that required a large amount of code. These code versions are publicly available for developers to add other functionality for Stiquito using this microprocessor [5].

10. References

[1] James M. Conrad and Jonathan W. Mills, *Stiquito™ for Beginners: An Introduction to Robotics*, IEEE Computer Society Press (Wiley), Los Alamitos, CA, 1999.

[2] James M. Conrad and Jonathan W. Mills, *Stiquito™: Advanced Experiments with a Simple and Inexpensive Robot*, IEEE Computer Society Press (Wiley), Los Alamitos, CA, 1997.

[3] James M. Conrad and Jonathan W. Mills, *Stiquito™ Controlled!*, IEEE Computer Society Press (Wiley), Los Alamitos, CA, 2005.

[4]focus.ti.com/docs/prod/folders/print/msp430f1122.html

[5] www.coe.uncc.edu/~jmconrad/StiquitoControlled.htm