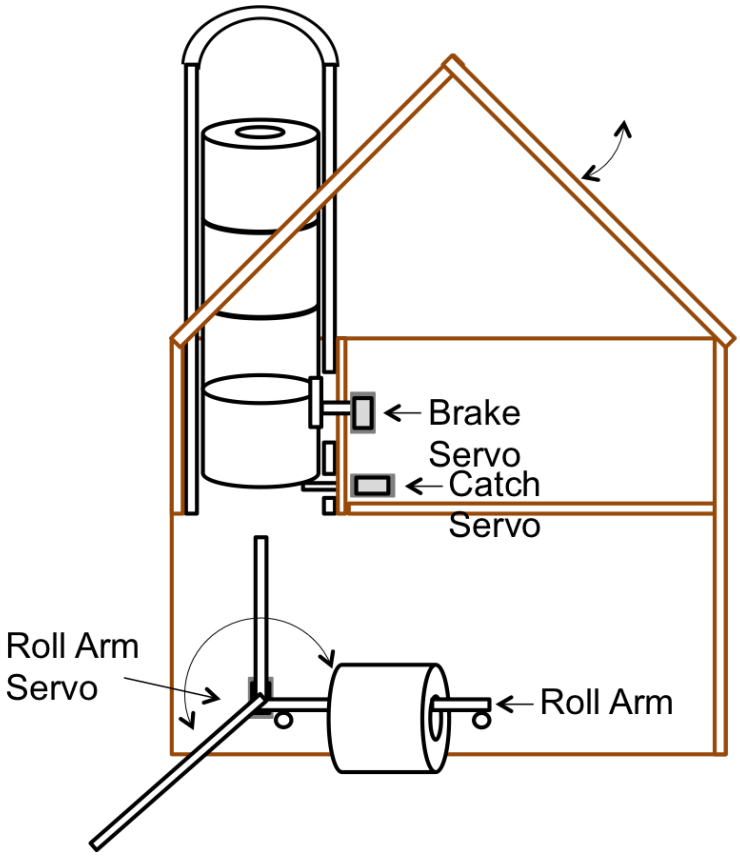
Toilet-Roll-O-Matic

John Dunn & Joshua Crane

May 5, 2015



1 Introduction

Its frustrating, time consuming, and physically challenging to change a roll of toilet paper. Usually the chore presents itself after nature has taken its course, and you have finished the last of the current roll. This forces you to awkwardly waddle to the sink vanity where you keep fresh rolls.

In a society where you can access anything from your phone, why havent inventors reached the humble toilet paper roll holder? Of the multiple profitable inventions we had conjured up, the toilet paper roll manager seemed like a winner.

Enter the Toilet-Roll-O-Matic. A simple, yet elegant solution to the problem posed, it soon exceeded our initial expectations of what a toilet paper roll manager could be. With persistence, 3 servo motors, LPCxpresso Cortex M0 board, and 5V power supply, we were able to construct the beauty seen below.

Figure 1

2 Background

The LPCxpresso board we purchased for the class worked just fine for our application. It contained a basic Cortex M0 processor that had enough features to reliably run our code. To run code, we needed to use the LPCxpresso software in conjunction with the board.

The servos we purchased required a pulse width modulated (PWM) signal to operate bidirectionally. To create a PWM signal for each of the three servos, we utilized three timers on the board, along with code from the PWM example found on the NXPs website.

We also developed an Android application in Android Studio to control the Toilet-Roll-O-Matic. Using Android bluetooth pairing procedures and a BlueSMiRF module connected to the Cortex M0 processor through a breakout board, we were able to create a reliable data signal between the two devices. To send data in this signal, we utilized UARTs (universal asynchronous receiver/transmitter) in both the Android phone and the Cortex M0. Sending the data was as simple as UARTSend(Your message here, NumOfChars). Receiving and processing data was trickier, and required knowledge of interrupts and interrupt request routines.

3 Design and Testing Methodology

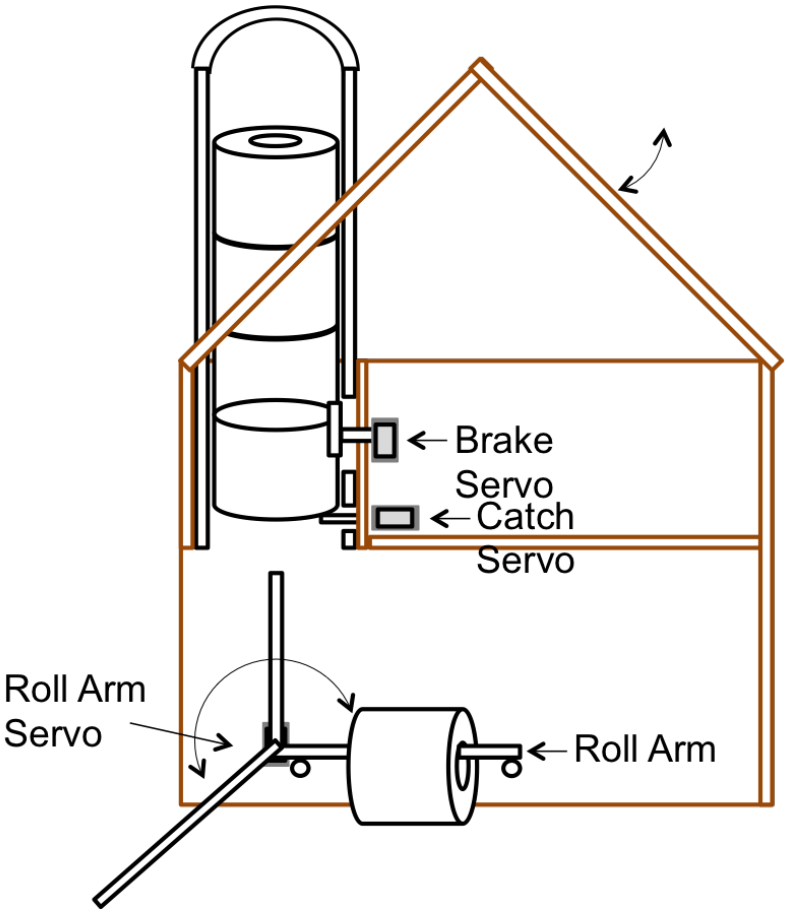


Figure 2

Foremost, the system needed to operate as a toilet paper dispenser similar to those found in bathrooms already. To achieve this, a simple roll arm was made hang rolls on. This bar was then attached to a servo in order to facilitate the reloading process. We selected a servo with more torque for the roll arm servo because it would have to operate reliably under load.

To ensure only one roll was dispensed from the feed tube at a time, we designed a brake and catch servo system. Another option that we considered was using one servo with a ratcheting arm to control the dispensing of rolls. Due to time constraints and limited access to mechanical engineering students, we went in favor of the two servo system. The catch servo holds the rolls inside the feed tube from the bottom edge of the tube, while the brake servo holds all but one roll during the reload process. This ensures that only one new roll is loaded on the roll arm at a time. The final three servo designed proved to be surprisingly efficient.

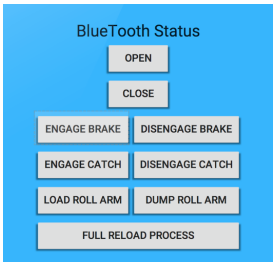


Figure 3: The Toilet-Roll-O-Matic Android Application

The operation of these servos was controlled by the LPCxpresso. When the user presses the full reload process button on the Android application, an onClickListener in the application sends a byte of information to the Bluetooth module connected to the LPCxpresso board via a UART connection. When interrupted by this byte through the UART, an interrupt service routine is called on the LPCxpresso. Our code intercepts this interrupt and uses a function called UART_New_Byte to determine which servo process to execute. UART_New_Byte chooses one, or in the case of Full Reload Process, a combination of these processes, to execute.

```
int main (void)
{ ...
}


void UART_New_Byte (uint8_t byte){ ...
}

void Print_Menu(uint8_t menu){ ...
}

void Brake_Servo_On(){ ...
}
void Catch_Servo_On(){ ...
}
void Roll_Arm_Servo_On(){ ...
}

void Brake_Servo_Off(){ ...
}
void Catch_Servo_Off(){ ...
}
void Roll_Arm_Servo_Off(){ ...
}

void Set_Servo_Direction_Brake(uint8_t rot){ ...
}
void Set_Servo_Direction_Catch(uint8_t rot){ ...
}
void Set_Servo_Direction_Roll_Arm(uint8_t rot){ ...
}


void Engage_Brake_Servo(){ ...
}
void Disengage_Brake_Servo(){ ...
}

void Engage_Catch_Servo(){ ...
}
void Disengage_Catch_Servo(){ ...
}

void Dump_Roll_Arm_Servo(){ ...
}
void Load_Roll_Arm_Servo(){ ...
}

void Delay(uint32_t delay){ ...
}
```

Figure 4

Some of the functions in Figure 4 (from Brake_Servo_On to Set_Servo_Direction_Roll_Arm) manipulate match registers of each of the three timers to change the PWM sent to each servo. The lower six func-

tions (from Engage_Brake_Servo to Load_Roll_Arm_Servo) use the manipulation functions to perform their processes. A more detailed explanation of this can be found in the Code section.

With all of this completed the full system was tested to ensure that it met the requirements of the original proposal. The reload process was first tested and tuned step by step to ensure that each of the servos worked consistently as expected. The entire reload process was then ran multiple times with a varying number of rolls loaded (this changes the weight on the catch servo and thus could have affect the system.) It consistently preformed the desired actions without any complications. It was also tested under slightly varying roll arm starting locations. This simulated possible unwanted user error. Again, it preformed the desired role consistently.

4 Results

As proven from the testing described above the project met all of the requirements. It was able to consistently and accurately reload toilet paper rolls. It is worth noting that this process occurs faster than a human user would be able to complete the same objective. Furthermore, as a final user product it was aesthetically designed to be appealing to use. One aspect that could be added to the project would be a button that activated the reload process. This would be easy to add and the Bluetooth was demonstrated as that was more complicated. The entire system cost approximately $120 to construct. This is high for a consumer product of this nature. This cost would be greatly reduced through large scale manufacturing to make it a more viable product.

5 Conclusions

The goal was to create a user ready product that automatically reloaded toilet paper rolls. This was achieved through the use of an LPCXpresso board, 3 servos, a bluetooth module, and plywood framing. Our design was able to quickly and accurately reload toilet paper rolls.

6 Code

```
#include "LPC11Uxx.h"
#include "timer16.h"
#include "clkconfig.h"
#include "gpio.h"
#include "nmi.h"
#include "uart.h"

extern volatile uint32_t UARTCount;
extern volatile uint8_t UARTBuffer[BUFSIZE];
extern volatile uint32_t timer32_0_counter[2];

uint8_t menu;
uint8_t printMenus = 0;
float DutyCycle_CT32B0;
float DutyCycle_CT32B1;
float DutyCycle_CT16B0;
uint32_t sysClkInt32;
uint16_t sysClkInt16;
uint16_t Prescalar16 = 10;

/* Main Program */

int main(void) {
  SystemCoreClockUpdate();

  /* Timer Setup */
  CLKOUT_Setup(CLKOUTCLK_SRC_MAIN_CLK);
  LPC_IOCON->PIO0_1 &= 0x01; /* CLK OUT */

  /* Enable AHB clock to the GPIO domain. */
  /* Piping clock to GPIO (we need it for PIO1_25) */
  LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 6);
  /* Piping clock to The CT32B0, CT32B1, and CT16B0 */
  LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 7) | (1 << 9) | (1 << 10);
  /* Piping clock to the IOCON */
  LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 16);

  /* Sets up the timer interval. SystemCoreClock is 12-Mhz*/
  /* Timer Interval for 32-bit Counters */
  sysClkInt32 = (SystemCoreClock / 500);
  /* Timer Interval for 16-bit Counter */
  sysClkInt16 = (SystemCoreClock / 500) / Prescalar16;
  LPC_CT16B0->PR = Prescalar16;

  if (sysClkInt32 > 0xFFFFFFFF) {
    sysClkInt32 = 0xFFFFFFFF;
  }
  if (sysClkInt16 > 0xFFFF) {
    sysClkInt16 = 0xFFFF;
  }
  /* Brake Servo (P1_25) */
  /* Output pin is P1_25, Connected to CT32B0->MR1 */
  LPC_IOCON->PIO1_25 = (0x1);

  /* Catch Servo (P0_14) */
  /* Output pin is P0_14, Connected to CT32B1->MR1 */
```

```
57    LPC_IOCON->TRST_PIO0_14 = (0x3);

59    /* Roll Arm Servo (P0_9) */
      /* Output pin is P0_9, Connected to CT16B0->MR1 */
61    LPC_IOCON->PIO0_9 = (0x2);

63    /* CT32BO Enable PWM for MAT1 and MAT3 */
      LPC_CT32B0->PWMC |= (1 << 1) | (1 << 3);
65    /* CT32B1 Enable PWM for MAT1 and MAT3 */
      LPC_CT32B1->PWMC |= (1 << 1) | (1 << 3);
67    /* CT16BO Enable PWM for MAT1 and MAT3 */
      LPC_CT16B0->PWMC |= (1 << 1) | (1 << 3);
69
      /* Enables EM1(clears output on match) and EM3(Toggles output on match) */
71    LPC_CT32B0->EMR |= (0x3 << 10) | (1 << 6) | (1 << 3) | (1 << 1);
       /* Frequency */
73    LPC_CT32B0->MR3 = sysClkInt32;
      /* Enables EM1(clears output on match) and EM3(Toggles output on match) */
75    LPC_CT32B1->EMR |= (0x3 << 10) | (1 << 6) | (1 << 3) | (1 << 1);
       /* Frequency */
77    LPC_CT32B1->MR3 = sysClkInt32;
      /* Enables EM1(clears output on match) and EM3(Toggles output on match) */
79    LPC_CT16B0->EMR |= (0x3 << 10) | (1 << 6) | (1 << 3) | (1 << 1);
       /* Frequency */
81    LPC_CT16B0->MR3 = sysClkInt16;

83    LPC_CT32B0->MCR = (1 << 10);
      LPC_CT32B1->MCR = (1 << 10);
85    LPC_CT16B0->MCR = (1 << 10);

87    /* NVIC is installed inside UARTInit file. */
      UARTInit(9600);
89    UARTSend("AT+NAME\r\nTiger\r\n", 16);
      if (printMenus) {
91      Print_Menu(0);
      }
93
      while (1) {
95    }
}
97
void UART_New_Byte(uint8_t byte) {
99    switch (byte) {
      /* Engage Brake Servo*/
101   case '1':
        Engage_Brake_Servo();
103     break;

105     /* Disengage Brake Servo*/
      case '2':
107     Disengage_Brake_Servo();
        break;
109
        /* Engage Catch Servo */
111   case '3':
        Engage_Catch_Servo();
113     break;

115     /* Disengage Catch Servo */
```

```
    case '4':
117   Disengage_Catch_Servo();
      break;
119
      /* Dump Roll Arm Servo*/
121 case '5':
      Dump_Roll_Arm_Servo();
123   break;

125   /* Load Roll Arm Servo */
    case '6':
127   Load_Roll_Arm_Servo();
      break;
129
      /* Full Reload Process */
131 case '7':
      Dump_Roll_Arm_Servo();
133   Delay(1000000);
      Engage_Brake_Servo();
135   Delay(1000000);
      Disengage_Catch_Servo();
137   Delay(2500000); /*Change Delay Later */
      Engage_Catch_Servo();
139   Delay(1000000);
      Disengage_Brake_Servo();
141   Delay(1000000);
      Load_Roll_Arm_Servo();
143   break;
    }
145 }

147 void Print_Menu(uint8_t menu) {
    switch (menu) {
149 case 0:
      UARTSend(
151       (uint8_t *) "\n\r|----Servo-Control-Menu----------------------|\n\r",
          47);
153   UARTSend((uint8_t *) "| 1. Engage Brake Servo                    |\n\r",
          45);
155   UARTSend((uint8_t *) "| 2. Disengage Brake Servo                 |\n\r",
          45);
157   UARTSend((uint8_t *) "| 3. Engage Catch Servo                    |\n\r",
          45);
159   UARTSend((uint8_t *) "| 4. Disengage Catch Servo                 |\n\r",
          45);
161   UARTSend((uint8_t *) "| 5. Dump Roll Arm Servo                   |\n\r",
          45);
163   UARTSend((uint8_t *) "| 6. Load Roll Arm Servo                   |\n\r",
          45);
165   UARTSend((uint8_t *) "| 7. Full Reload Process                   |\n\r",
          45);
167   UARTSend(
          (uint8_t *) "|-----------------------------------------|\n\n\r",
169       46);
      break;
171 case 1:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
173       46);
      UARTSend((uint8_t *) "| Engaged Brake Servo                      |\n\r",
```

```
175         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
177         46);
      break;
179   case 2:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
181         46);
      UARTSend((uint8_t *) "| Disengaged Brake Servo                  |\n\r",
183         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
185         46);
      break;
187   case 3:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
189         46);
      UARTSend((uint8_t *) "| Engaged Catch Servo                     |\n\r",
191         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
193         46);
      break;
195   case 4:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
197         46);
      UARTSend((uint8_t *) "| Disengaged Catch Servo                  |\n\r",
199         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
201         46);
      break;
203   case 5:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
205         46);
      UARTSend((uint8_t *) "| Dump Roll Arm Servo Completed           |\n\r",
207         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
209         46);
      break;
211   case 6:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
213         46);
      UARTSend((uint8_t *) "| Load Roll Arm Servo Completed           |\n\r",
215         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
217         46);
      break;
219   case 7:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
221         46);
      UARTSend((uint8_t *) "| Full Reload Process Completed           |\n\r",
223         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
225         46);
      break;
227   default:
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
229         46);
      UARTSend((uint8_t *) "| Nothing Changed                         |\n\r",
231         46);
      UARTSend((uint8_t *) "|-----------------------------------------|\n\r",
233         46);
```

```
    break;
  }
}

void Brake_Servo_On() {
  if (!(LPC_CT32B0->TCR &= 1)) {
    LPC_CT32B0->TCR |= (1 << 0);
  }
}
void Catch_Servo_On() {
  if (!(LPC_CT32B1->TCR &= 1)) {
    LPC_CT32B1->TCR |= (1 << 0);
  }
}
void Roll_Arm_Servo_On() {
  if (!(LPC_CT16B0->TCR &= 1)) {
    LPC_CT16B0->TCR |= (1 << 0);
  }
}

void Brake_Servo_Off() {
  if ((LPC_CT32B0->TCR &= 1)) {
    LPC_CT32B0->TCR &= (0 << 0);
  }
}
void Catch_Servo_Off() {
  if ((LPC_CT32B1->TCR &= 1)) {
    LPC_CT32B1->TCR &= (0 << 0);
  }
}
void Roll_Arm_Servo_Off() {
  if ((LPC_CT16B0->TCR &= 1)) {
    LPC_CT16B0->TCR &= (0 << 0);
  }
}

void Set_Servo_Direction_Brake(uint8_t rot) {
  switch (rot) {
  case 1:
    DutyCycle_CT32B0 = 1.335;
    break;
  case 2:
    DutyCycle_CT32B0 = 2;
    break;
  case 3:
    DutyCycle_CT32B0 = 4;
    break;
  }

  LPC_CT32B0->MR1 = (uint32_t) ((((float) sysClkInt32) / DutyCycle_CT32B0)); /* Duty Cycle
    */
}
void Set_Servo_Direction_Catch(uint8_t rot) {
  switch (rot) {
  case 1:
    DutyCycle_CT32B1 = 1.335;
    break;
  case 2:
    DutyCycle_CT32B1 = 2;
```

```
    break;
  case 3:
    DutyCycle_CT32B1 = 4;
    break;
  }

  /* Duty Cycle */
  LPC_CT32B1->MR1 = (uint32_t) ((((float) sysClkInt32) / DutyCycle_CT32B1));
}
void Set_Servo_Direction_Roll_Arm(uint8_t rot) {
  switch (rot) {
  case 1:
    DutyCycle_CT16B0 = 1.335;
    break;
  case 2:
    DutyCycle_CT16B0 = 2;
    break;
  case 3:
    DutyCycle_CT16B0 = 4;
    break;
  }

  /* Duty Cycle */
  LPC_CT16B0->MR1 = (uint16_t) ((((float) sysClkInt16) / DutyCycle_CT16B0));
}

void Engage_Brake_Servo() {
  Set_Servo_Direction_Brake(1);
  Brake_Servo_On();
  Delay(4000000);
  Brake_Servo_Off();
  if (printMenus) {
    Print_Menu(1);
    Print_Menu(0);
  }
}
void Disengage_Brake_Servo() {
  Set_Servo_Direction_Brake(3);
  Brake_Servo_On();
  Delay(1000000);
  Brake_Servo_Off();
  if (printMenus) {
    Print_Menu(2);
    Print_Menu(0);
  }
}

void Engage_Catch_Servo() {
  Set_Servo_Direction_Catch(3);
  Catch_Servo_On();
  Delay(900000);
  Catch_Servo_Off();
  if (printMenus) {
    Print_Menu(3);
    Print_Menu(0);
  }
}
void Disengage_Catch_Servo() {
  Set_Servo_Direction_Catch(2);
```

```
351   Catch_Servo_On();
      Delay(700000);
353   Catch_Servo_Off();
      if (printMenus) {
355     Print_Menu(4);
        Print_Menu(0);
357   }
}
359
void Dump_Roll_Arm_Servo() {
361   Set_Servo_Direction_Roll_Arm(1);
      Roll_Arm_Servo_On();
363   Delay(2000000);
      Set_Servo_Direction_Roll_Arm(2);
365   Delay(2000000);
      Set_Servo_Direction_Roll_Arm(3);
367   Delay(1250000);
      Set_Servo_Direction_Roll_Arm(2);
369
      if (printMenus) {
371     Print_Menu(5);
        Print_Menu(0);
373   }
}
375 void Load_Roll_Arm_Servo() {
      Set_Servo_Direction_Roll_Arm(3);
377   Roll_Arm_Servo_On();
      Delay(300000);
379   Roll_Arm_Servo_Off();
      if (printMenus) {
381     Print_Menu(6);
        Print_Menu(0);
383   }
}
385
void Delay(uint32_t delay) {
387   int i;
      for (i = 0; i < delay; i++) {
389   }
}
```

ToiletRollOMatic.c