ARMY RESEARCH LABORATORY

Interrupt Driven RS-232, Pulse Width Modulation, and Control Processing on a Single 8-bit PIC Chip

by Justin Shumaker

ARL-TR-4747 **March 2009**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TR-4747 **March 2009**

Interrupt Driven RS-232, Pulse Width Modulation, and Control Processing on a Single 8-bit PIC Chip

Justin Shumaker
Vehicle Technology Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2009	Final	2 July 2007–19 July 2007

4. TITLE AND SUBTITLE
Interrupt Driven RS-232, Pulse Width Modulation, and Control Processing on a Single 8-bit PIC Chip

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Justin Shumaker

5d. PROJECT NUMBER
1L162618AH80

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRD-ARL-VT-UV
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-4747

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Autopilots utilizing a single processor allow for smaller, lighter, more cost-effective designs than those using multi-processor architecture. Presented here is a software architecture that allows multiple tasks to operate in a real-time mutually exclusive environment using a single 8-bit processor. This type of architecture will be utilized to create an autopilot using a single processor and sensors. The problem is preventing individual tasks from interfering with one another in a real-time environment. The processor selected for this design is the 8-bit microchip programmable integrated circuit (PIC) 18F2420. Hardware limitations on the microchip PIC, such as a lack of symmetric multi-processing, must be emulated in a round-robin interrupt scheduler whereby each task runs in sequence, receiving only a slice of the total processing time available.

15. SUBJECT TERMS
microprocessor, UAV, autopilot, microchip, GPS, RS-232, PIC

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Justin Shumaker
Unclassified	Unclassified	Unclassified	UU	22	19b. TELEPHONE NUMBER (*Include area code*) (410) 278-2834

Contents

List of Figures

1. Background

Several commercial autopilots, such as the Micropilot, Piccolo, and Procerus, utilize multiple chips for processing and interfacing with sensors. While this hardware architecture greatly simplifies the software, it makes the overall hardware complexity greater due to an increased number of components and traces. If all of the processing and interfaces to sensors can be performed in software, then the autopilot is reduced to a single processor and sensor chips.

Most 8-bit chips typically have no parallel processing capabilities and therefore require the use of interrupt driven programming to emulate this behavior. Alternatively, one could utilize a faster multi-core 32-bit microprocessor environment or a field-programmable gate array (FPGA). While both the 32-bit and FPGA architectures provide more processing power they are typically larger, more costly, and consume more power than 8-bit chips.

2. The Impetus

In order to reduce the size of an autopilot one must minimize the number of components used. For example, one open source autopilot utilizes five programmable integrated circuit (PIC) chips in order to operate the global positioning system (GPS) sensor, wireless communications, vehicle control, servo control, and read the inertial measurement unit (IMU) sensors. After evaluating this autopilot platform and measuring the computational load on each processor, it became apparent that all of the PIC chips on this autopilot were mostly idle. This discovery lead to the idea of migrating this 5-PIC chip design on to a single 10 million instructions per/s 8-bit PIC chip.

3. The Setup

An inexpensive prototyping board, 18F2420 PIC chip, GPS, and servo were utilized to create a circuit to interface GPS data and drive up to four servos. The objective was to see if it were possible to process GPS data, generate pulse-width modulation (PWM) signals for four servos, and process a control algorithm on a single 8-bit PIC chip. The GPS sensor received data at 5 Hz over a 9600 baud RS-232 link while driving four servos with 1% precision (1.8° per step) in their full range (180°) of motion. Figure 1 illustrates the prototyped circuit and figure 2 provides a schematic of the circuit.

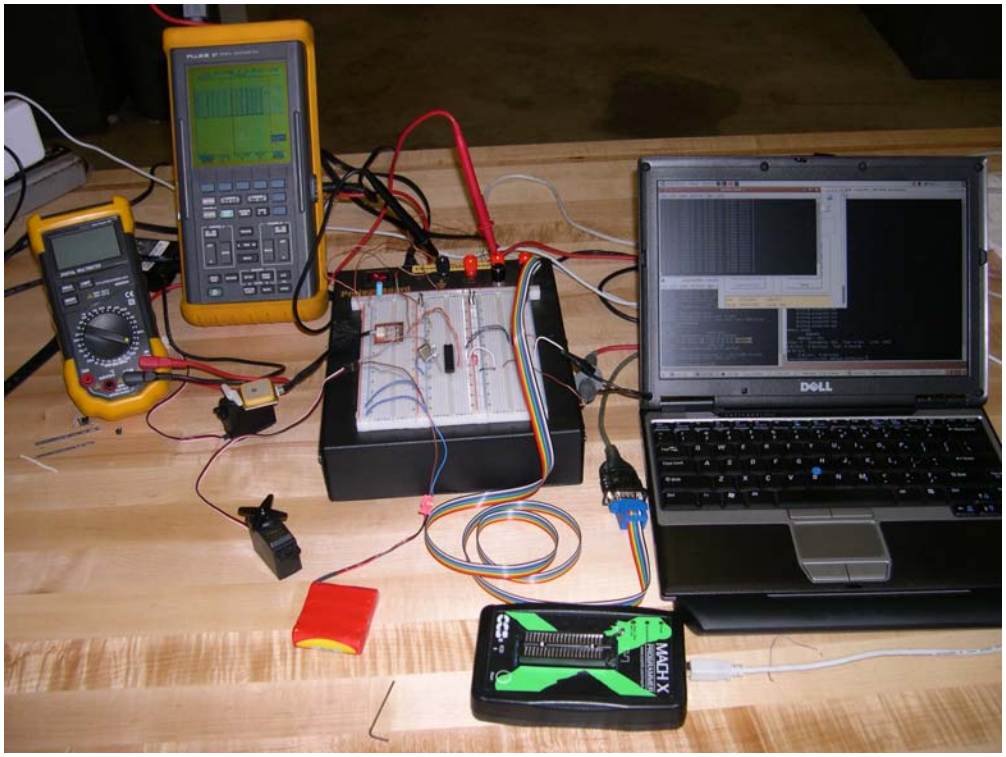

Figure 1. Picture of prototyped circuit.

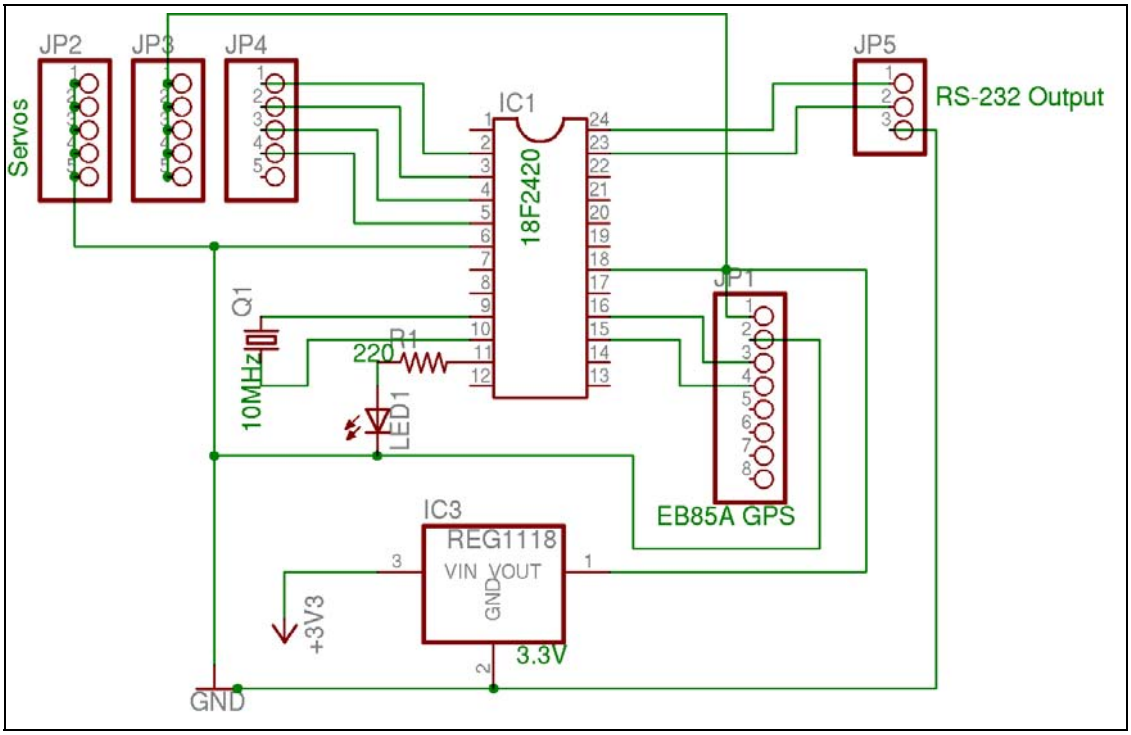

Figure 2. Schematic of the circuit.

4. Interrupt Driven Design

If both RS-232 and PWM tasks are to be handled simultaneously, then they must operate in a mutually exclusive real-time interrupt driven environment. Interrupts are a means to transfer control from one part of a program to another, provided an external trigger or timer-based event occurred. After an interrupt completes control is transferred from the interrupt back to the program or any new interrupts that may have occurred during the servicing of the current interrupt.

Both PWM and RS-232 generate pulses many times per second. More specifically, RS-232 generates 10 bits for every byte because of the start bit, 8-bits of data, and stop bit. Furthermore, an RS-232 signal is time dependent and will result in corrupt data if not read at exactly the correct time intervals. For example, a 9600 baud RS-232 link streams 9,600 bits/s, which translates to a required pulse sampling every 104.16 μs. However, if the RS-232 connection is buffered then the chip is storing the bytes in a hardware buffer until they are read. Most 8-bit PIC chips have a 2-byte universal synchronous asynchronous receiver transmitter (USART) hardware buffer. Therefore, it is possible to read the buffer once every 1.04 ms (1 byte [9600/10 bits per byte] or every 2.08 ms (2 bytes [9600/10 bits per byte]) before any data loss. The USART hardware buffer greatly simplifies support for RS-232 devices because they allow for an order of magnitude more clock cycles to occur between each servicing of the interrupt. While 2.08 ms doesn't sound like a great deal of time, there are actually 20,833 (2.08 ms * 10 million instructions per second [MIPS]) instructions that can be processed during that time. This interval leaves plenty of time for the PWM interrupt to operate between each byte transferred over the RS-232 link.

The RS-232 GPS interrupt was required to perform several comparison tests and buffer incoming data until a specific line of the National Marine Electronics Association data was received before setting a variable to indicate that the GPS buffer was ready to be processed. The GPS interrupt was set to trigger once a byte appeared in the USART hardware buffer of the PIC chip. Once the GPS buffer (an array of characters in memory) was full, the interrupt was disabled to prevent the newly available data from being overwritten. Once the control algorithm processed the buffered GPS data, it allowed the GPS interrupt to buffer new data once again.

The PWM interrupt was more sophisticated than the GPS interrupt because PWM requires a 20-ms low time followed by a variable high time for four individual servos. Most servos require a 20-ms low time followed by a 0.5–2.5 ms high time to set the position of the servo from 0° of rotation to 180° of rotation (see figure 3). It was important not to consume anywhere near 2.08 ms of time in the PWM interrupt, otherwise the risk of data loss from the GPS was imminent. The PWM interrupt was divided into three conditions, one for the 20-ms low time,

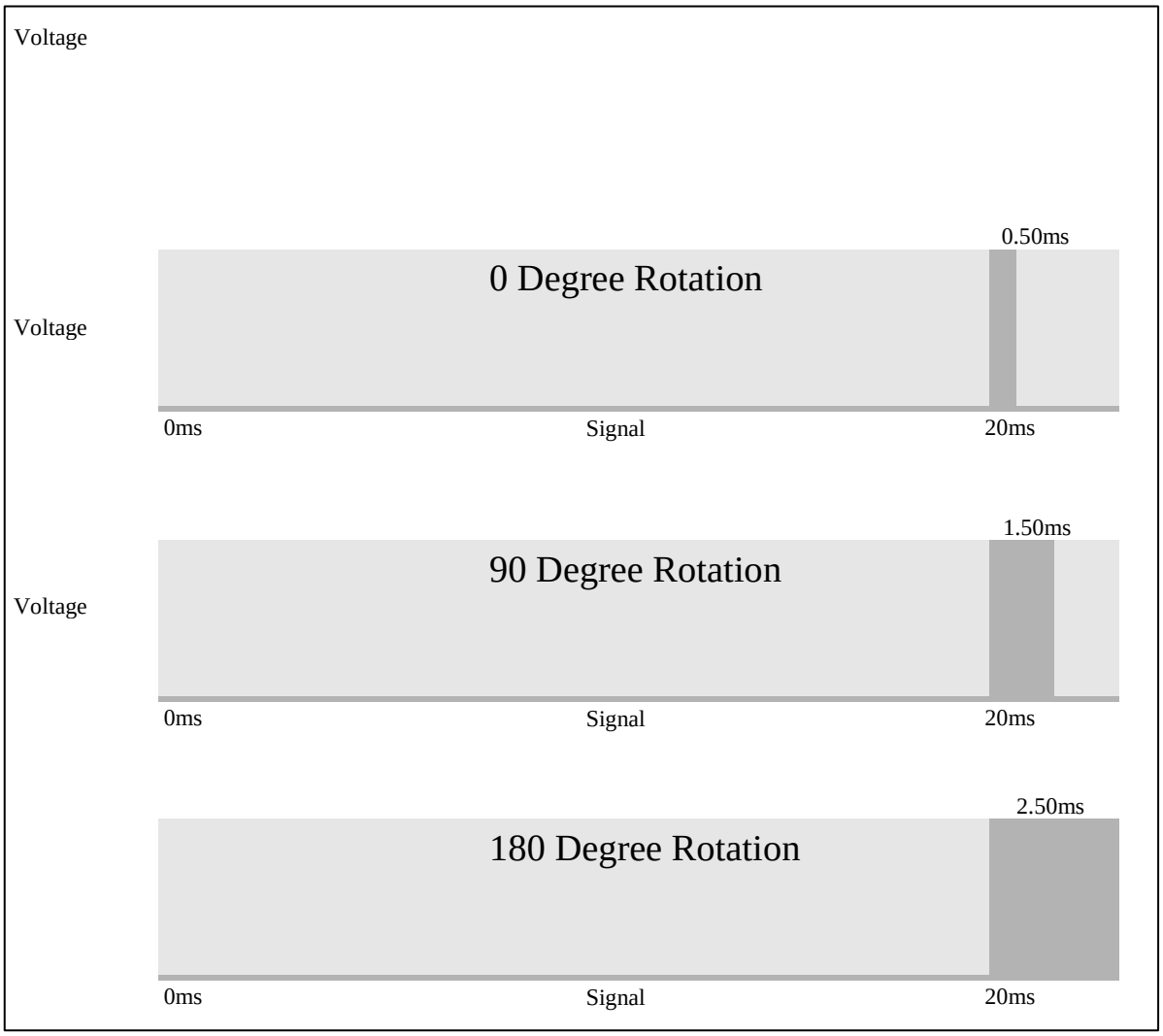

Figure 3. Servo PWM control signals.

one for the 0.5-ms fixed high time, and one for the variable high times defined by each variable corresponding to the servo position. The PWM interrupt operated off of a 16-bit timer interrupt with a post-scaling of 4, which translated to an interrupt trigger every time the 16-bit timer overflowed at $[(2^{16} * 4) / 10{,}000{,}000] = 26.2144$ ms. Because the 16-bit timer can be set to any value $[0, 2^{16}]$ while incrementing, it is possible to set the overflow to occur from [0 ms, 26.2144 ms]. Setting the timer to a value of 15,535 for the 20-ms low time condition meant there would be exactly $[(65535 – 15535) * 4) / 10{,}000{,}000] = 20$ ms before the interrupt was triggered again. Furthermore, the PWM interrupt only consumes about 10 clock cycles to do all of this first condition setup. Once the interrupt occurred it would be operating under the second condition which was responsible to set all four servos high for 0.5 ms. This second condition was done much in the same way the first condition was done except with an overflow of (65535 - 1250) = 64,285. The third condition was set to overflow every 20 μs, 100 times, for a total of 2 ms so that there were 100 different positions the servos could be toggled off at to provide 1% positional precision. Again, the timer was calculated and set to be (65535–50) = 65485. As the

interrupt occurred 100 times during the third condition, each servo value was compared with the iteration number of the interrupt and, once the iteration exceeded the value of the servo value, the servo was set low again. After the 100 iterations of the third condition, all pins on the servos are low and the condition was set back to 0 for the whole process to repeat again. During all 102 interrupt triggers for a complete PWM cycle, the PWM interrupt function provided ample time between each PWM interrupt to allow for the GPS interrupt to handle incoming data from the GPS sensor.

With both PWM and GPS interrupts constantly triggering, there is still ample time for the main/control task to be performing calculations in the background. The main/control task is not compromised if a calculation is interrupted so long as it gets completed. When the main/control task requires GPS data, it reads the data out of the GPS buffer and updates the PWM servo variables with the appropriate values.

5. Summary

The handling of multiple tasks in a real-time environment, a single 8-bit PIC chip with an interrupt driven architecture works. It is quite feasible to create an autopilot utilizing a single 8-bit PIC chip. If size is not a huge concern, then one might consider using multiple PIC chips in a voting system for redundancy. RS-232 connections require a hardware USART buffer for this architecture to work. Most PIC chips have one or two USARTs. Therefore, if GPS, IMU, and communications are to be integrated it is essential that either the GPS or IMU operate on a non-RS232 protocol such as inter integrated circuit (I2C) or serial parallel interface (SPI) because wireless communications will likely be one of the RS-232 devices requiring a hardware USART. With the GPS, PWM, and IMU interrupt tasks running in the background, the 18F2420 8-bit 10 MIPS PIC chip had between 90 and 95% of its resources (9.0–9.5 MIPS, respectively) available to perform additional control calculations. As 16-bit PIC chips bolstering speeds up to 40 MIPS become prevalent, developers will be able to create more complex control algorithms with only minor changes to the interrupt code. For the complete code listing, see the appendix.

INTENTIONALLY LEFT BLANK.

Appendix. Complete autopilot.c Code

This appendix appears in its original form, without editorial change.

```
#include <18F2420.h>
#include <stdlib.h>
#include <math.h>
//#device ICD=TRUE
#fuses H4,NOWDT,NOLVP,PUT,NOMCLR,NOBROWNOUT
#use delay (clock=40000000)
#use rs232 (baud=115200, xmit=PIN_B7, rcv=PIN_B6, stream=stream_comm, INVERT)
#use rs232 (baud=9600, UART1, stream=stream_gps)


#define PIN_LED          PIN_C2

#define CLEAR_RCSTA() { \
            if (bit_test (*0xFAB, 1)) \
            { \
              bit_clear (*0xFAB, 7); \
              bit_clear (*0xFAB, 4); \
              bit_set (*0xFAB, 4); \
              bit_set (*0xFAB, 7); \
            } }

typedef struct io_s
{
  int8 pwm[4];
  signed int8 pwm_cycle, gpgga_ind;
  char gpgga[80], uart_buf[3];
} io_t;


typedef struct gps_s
{
  float time;
  float latitude;
  float longitude;
  int8 quality;
  int8 satellites;
  float altitude;
} gps_t;


typedef struct autopilot_s
{
  int8 toggle;
} autopilot_t;


io_t io;
autopilot_t autopilot;
gps_t gps;


/****************************************
* BEGIN - INTERNAL FUNCTIONS
****************************************/
void
gps_init ()
```

```
{
  /*
  * Specific to the EB-230 GPS Chip using PMTK command set.
  * Change the GPS to output only the GPGGA string and change the
  * baud rate from 38400 to 9600.
  */
  SETUP_UART (38400, stream_gps);
  delay_ms (2500);
  fprintf (stream_gps, "$PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n"); /* GPGGA 5x a second */
  delay_ms (100);
  fprintf (stream_gps, "$PMTK251,9600*17\r\n"); /* 9600 Baud */
  delay_ms (100);
  SETUP_UART (9600, stream_gps);
  fprintf (stream_gps, "$PMTK301,2*2e\r\n"); /* Enable Differential WAAS Support */
  CLEAR_RCSTA ();
}


#int_rda
void
gps_interrupt ()
{
  io.uart_buf[0] = io.uart_buf[1];
  io.uart_buf[1] = io.uart_buf[2];
  io.uart_buf[2] = fgetc (stream_gps);

  if (io.gpgga_ind == 79)
  {
    /* Handle an overflow */
    io.gpgga_ind = -1;
  }
  if (io.gpgga_ind >= 0)
  {
    io.gpgga[io.gpgga_ind] = io.uart_buf[2];
    io.gpgga_ind++;
    io.gpgga[io.gpgga_ind] = 0;
    if (io.uart_buf[2] == '*')
      io.gpgga_ind = -1;
  }
  else if ((io.uart_buf[0] == 'G') && (io.uart_buf[1] == 'A') && (io.uart_buf[2] == ','))
  {
    io.gpgga_ind = 0;
  }
}


#int_timer1
void
pwm_driver_interrupt ()
{
  /*
  * Generate a PWM signal for 20ms low and (0.5-2.5ms) high
  * using an integer [0..100]
  */
  if (io.pwm_cycle == -1)
  {
```

```
    output_high (PIN_A0);
    output_high (PIN_A1);
    output_high (PIN_A2);
    output_high (PIN_A3);
    set_timer1 (64295); /* Hold high for 0.5ms + bias(10)[40] */
    io.pwm_cycle++;
  }
  else if (io.pwm_cycle > 100)
  {
    /* Timer1 will overflow every 20ms */
    set_timer1 (15535);
    io.pwm_cycle = -1;
  }
  else
  {
    /* Every 20us - 100x in 2.0ms - 65535 - 50[200] + bias(15)[60] */
    set_timer1 (65500);

    if (io.pwm_cycle >= io.pwm[0])
      output_low (PIN_A0);
    if (io.pwm_cycle >= io.pwm[1])
      output_low (PIN_A1);
    if (io.pwm_cycle >= io.pwm[2])
      output_low (PIN_A2);
    if (io.pwm_cycle >= io.pwm[3])
      output_low (PIN_A3);
    io.pwm_cycle++;
  }
}


void
format_gps ()
{
  char gpgga_copy[80], buf[13], buf2[13];
  int8 i, gpgga_ind, tok_ind, buf_ind;

  /*
  * This copy needs to be as quick as possible
  * so that interrupts can get going again.
  */
  disable_interrupts (GLOBAL);
  for (buf_ind = 0; buf_ind < 80; buf_ind++)
    gpgga_copy[buf_ind] = io.gpgga[buf_ind];
  io.gpgga_ind = -2;
  CLEAR_RCSTA();
  enable_interrupts (GLOBAL);

  /* Parse the GPGGA String */
  gpgga_ind = 0;
  tok_ind = 0;
  buf_ind = 0;
  while (tok_ind < 9)
  {
    if (gpgga_copy[gpgga_ind] == ',')
    {
```

```
  buf[buf_ind] = 0;
  if (tok_ind == 0)
  {
    gps.time = atof (buf);
  }
  else if (tok_ind == 1)
  {
    for (i = 0; i < 2; i++)
      buf2[i] = buf[i];
    buf2[i] = 0;
    gps.latitude = atof (buf2);
    for (i = 0; i < 11; i++)
      buf2[i] = buf[2+i];
    buf2[i] = 0;
    gps.latitude += 0.01666666 * atof (buf2); /* divide by 60 */
  }
  else if (tok_ind == 2)
  {
    if (buf[0] == 'S')
      gps.latitude *= -1.0;
  }
  else if (tok_ind == 3)
  {
    for (i = 0; i < 3; i++)
      buf2[i] = buf[i];
    buf2[i] = 0;
    gps.longitude = atof (buf2);
    for (i = 0; i < 10; i++)
      buf2[i] = buf[3+i];
    buf2[i] = 0;
    gps.longitude += 0.01666666 * atof (buf2); /* divide by 60 */
  }
  else if (tok_ind == 4)
  {
    if (buf[0] == 'W')
      gps.longitude *= -1.0;
  }
  else if (tok_ind == 5)
  {
    gps.quality = atoi (buf);
  }
  else if (tok_ind == 6)
  {
    gps.satellites = atoi (buf);
  }
  else if (tok_ind == 8)
  {
    gps.altitude = atof (buf);
  }

  tok_ind++;
  buf_ind = 0;
}
else
{
  buf[buf_ind] = gpgga_copy[gpgga_ind];
```

```
      buf_ind++;
    }
    gpgga_ind++;
  }
}
/****************************************
* END - INTERNAL FUNCTIONS
****************************************/


/****************************************
* BEGIN - USER FUNCTIONS
****************************************/
void
control ()
{
  float x, y, z;
  /*
  * PERFORM SOME MATH
  */
  x = cos (0.12345);
  y = sin (0.12345);
  z = x / y;

  /*
  * BLINK LED and ADJUST SERVO
  */
  if (autopilot.toggle)
  {
    output_high (PIN_LED);
    io.pwm[0] = 10;
  }
  else
  {
    output_low (PIN_LED);
    io.pwm[0] = 90;
  }

  /*
  * OUTPUT SOMETHING TO TERMINAL
  */
  disable_interrupts (GLOBAL);
  if (gps.quality > 0)
  {
    fprintf (stream_comm, "tog: %d, time: %f, lat: %.4f, lon: %.4f, quality: %d\r\n", autopilot.toggle, gps.time,
gps.latitude, gps.longitude, gps.quality);
  }
  else
  {
    fprintf (stream_comm, "%s\r\n", io.gpgga);
  }
  CLEAR_RCSTA();
  enable_interrupts (GLOBAL);

  autopilot.toggle = 1 - autopilot.toggle;
}
```

```
/****************************************
* END - USER FUNCTIONS
****************************************/


void
main ()
{
  fprintf (stream_comm, "Autopilot Started\r\n");
  gps_init ();

  /* Initialize Variables */
  io.pwm_cycle = -1;
  io.gpgga_ind = -2;
  autopilot.toggle = 0;

  /* Timer1 Overflow is: 26.2144ms */
  setup_timer_1 (T1_INTERNAL |  T1_DIV_BY_4);
  enable_interrupts (INT_TIMER1);
  enable_interrupts (INT_RDA);
  enable_interrupts (GLOBAL);

  /* Initialize servo values */
  io.pwm[0] = 0;
  io.pwm[1] = 0;
  io.pwm[2] = 0;
  io.pwm[3] = 0;

  while (1)
  {
    if (io.gpgga_ind == -1)
    {
      format_gps ();
      control ();
    }
  }
}
```

NO. OF COPIES	ORGANIZATION
1 (PDF only)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LAB IMNE ALC HR 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI OK TL 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI OK PE 2800 POWDER MILL RD ADELPHI MD 20783-1197

ABERDEEN PROVING GROUND

1	DIR USARL AMSRD ARL CI OK TP (BLDG 4600)

NO. OF
COPIES ORGANIZATION

ABERDEEN PROVING GROUND

1 DIR USARL
AMSRD ARL VT UV
J SHUMAKER

INTENTIONALLY LEFT BLANK.