Underwater Planar Take-Off and Landing (UWPVTOL) vehicle description and code operation

Zebb D. Prime
The School of Mechanical Engineering
The University of Adelaide
South Australia
zebb.prime@mecheng.adelaide.edu.au

September 28, 2006

Abstract

The Underwater Planar Vertical Take-Off and Landing (UWPVTOL) vehicle is a control rig under construction for the demonstration of advanced non-linear control theory. This document provides a brief description of the operation of the code to run inside the UWPVTOL on a MiniDRAGON+ (Motorola 9S12 microcontroller) development board.

1 PVTOL Outline

Planar Vertical Take-Off and Landing (PVTOL) systems are a group of platforms operating in three degrees of freedom, namely two translational degrees of freedom (say x and y) and a rotational degree of freedom (say roll). An example of such a system is shown in Figure 1. The problem of non-linear stabilisation and control of such systems has been the subject of much research over the past 10 years. This quantity of research may be attributed to the systems interesting dynamics. Namely, it is underactuated, non-linear, unstable and non-minimum phase.

2 UWPVTOL

The Underwater PVTOL platform is a project under construction at the University of Adelaide. It is being constructed in order to demonstrate non-linear control theory. The problem of building an airborne VTOL vehicle is being tackled at many universities across the world, consuming large amounts of money and research
time. The UWPVTOL was devised to avoid these problems and concentrate on the underlying control theory. By operating in a long and skinny water tank the problem is simplified to three degrees of freedom, and the buoyancy of the vehicle reduces the time-constants of the platform, making it easier to control. In addition operating in water allows the effects of hydrodynamic drag to be simulated at low speeds. A Computer Aided Design (CAD) image of the platform under construction is shown in Figure 2.

The main components in the UWPVTOL are:

- MiniDRAGON+, Motorola 9S12 development board (http://www.evbplus.com)
- Microstrain 3DM-GX1 Inertial Measurement Unit (http://www.microstrain.com)
- Plettenberg HP220 DC Brushless motor (http://www.plettenberg-motoren.com)
- RF 418 MHz Transmitter & 916 MHz Receiver (http://www.sparkfun.com)

The (simplified) layout of these components is shown in Figure 3. The MiniDRAGON+ is the central component to the configuration. It is connected to
Figure 2: UWPVTOL CAD Image
the 3DM-GX1 IMU via an RS232 port, the RF receiver via an RS232 port, the RF transmitter by a bit-banged serial port, the ESC and servo motor via PWM outputs and signal conditioning (diagnostics) via the ADC circuit for information such as motor temperature, speed, etc.

3 9S12 Code Operation

The operation of the code is briefly described below. The documentation generated using Doxygen begin in Section 4.

3.1 RF Receiver

The RF Receiver module is based on code written by Stephen Craig. Interrupts are generated when the serial port receives a byte. At this time the byte is processed to see if it is part of a radio transmission. The protocol for the RF transmission is:

- Header byte (0x55 or alternating high/low)
- Packet length byte
- Data bytes
• CRC8 poly checksum

It the packet is correct it is placed onto a ‘valid data’ ring-buffer.

3.2 RF Processing

Whenever the CPU is idle it checks the ‘valid data’ ring-buffer for new information. If some appears it processes it to see if it is a command byte. Currently the only commands implemented are PWM period changes. The format of this command is:

• Command byte, the character ‘s’ (for servo)
• Two period bytes, the first two bits of which define which channel to adjust, while the next 14 bits are the period to adjust it to (multiplied by the resolution of the PWM output)

3.3 RF Transmitter

The RF Transmitter operates by manually setting a pin high/low (bit-banging) depending upon the packet being sent. The Enhanced Capture Timer (ECT - timer.c) on the 9S12 is used to generate timed interrupts to make the transmission efficient. It uses approximately 5% of the CPU time while transmitting at 38400bps, and zero at idle.

3.4 IMU Communication

The 3DM-GX1 is attached to the first serial port of the 9S12 (SCI0). It has been set to continuous mode 0x31, which sends the current euler angles, acceleration and rate vectors as fast as possible. The serial port code has been written specifically for this protocol. Once a full byte is received the checksum is computed, and if this is valid the required data (roll, roll rate, X & Y acceleration) are pushed into a buffer and send immediately over the RF transmitter.

3.5 RC PWM

Radio Control (RC) equipment typically use PWM signals as control signals. However they are a little quirky as the signal does not depend upon the duty cycle, rather the on period. The PWM signals are generated using the PWM module on the 9S12, see pwm.c, with resolution ranging from 0.25µs to 1µs, with the minimum PWM frequency increasing with resolution. The 9S12 has eight eight-bit channels, however these are combined to give four channels with 16-bit accuracy.
3.6 Diagnostics

The 9S12 samples the Analogue to Digital Converter (ADC) at a rate of 10Hz. This information is send immediately over the RF transmitter. The ADC ports are connected to a breakout board which conditions information about the motor speed and temperature and battery voltage and current. This information is essentially diagnostic information, to monitor the charge of the battery and check the health of the motor.

3.7 Timer

Channel 7 of the 9S12 ECT is used to generate periodic interrupts, at 2kHz. This is used to start the ADC subsystem, generate time-outs for the receiving and processing of data and finally to generate 1Hz flashes on the LED display on the MiniDRAGON+.

4 Doxygen: UWPVTOL File List

The following sections have been automatically generated using Doxygen. They are incomplete, and more detailed information can be obtained by studying the source files themselves as they are (mostly) adequately commented.

Here is a list of all documented files with brief descriptions:

- E:/Zebb/Working/Programming/PVTOL/PVTOL_Mini-Dragon/Sources/adc.c
  - 7

- E:/Zebb/Working/Programming/PVTOL/PVTOL_Mini-Dragon/Sources/pwm.c
  - 9

- E:/Zebb/Working/Programming/PVTOL/PVTOL_Mini-Dragon/Sources/radio_tx_bb.c
  - 12

- E:/Zebb/Working/Programming/PVTOL/PVTOL_Mini-Dragon/Sources/rf_rx_process.c
  - 13

- E:/Zebb/Working/Programming/PVTOL/PVTOL_Mini-Dragon/Sources/timer.c
  - 15
#include <mc9s12dp256.h>
#include "radio_tx_bb.h"
#include "adc.h"

Functions

- void ADC_Init (void)
  *Initialise the ADC module.*

- void ADC_Start (void)
  *Start the reading of the ADC module.*

- void ADC_Send (void)
  *Send the ADC data over the serial port.*

- __interrupt void ADC_ISR (void)
  *ADC read complete interrupt service routine.*

Variables

- unsigned char ADCDataReadyFlag

5.1 Detailed Description

Reads from the ADC0 unit and sends it via the RF serial port as diagnostic information (such as battery voltage and current, motor voltage and speed - depends upon hardware configuration).

Author:

Zebb Prime

Date:

22/9/06

Version:

1.0
5.2 Function Documentation

5.2.1 __interrupt void ADC_ISR (void)

ADC read complete interrupt service routine.
Clears interrupt request flag, then reads the results of the ADC.

5.2.2 void ADC_Send (void)

Send the ADC data over the serial port.
Should be called once the ADC read is complete (after the flag is non-zero).

5.3 Variable Documentation

5.3.1 unsigned char ADCDataReadyFlag

ADC module prototypes
#include <mc9s12dp256.h>

## Defines

- #define PWM67_38400

## Functions

- void PWM_Init (void)
  
  *Initialises the PWM module.*

- void PWM_T_Set (char channel, unsigned int period)
  
  *Sets the output period for a PWM channel.*

### 6.1 Detailed Description

PWM functions to output to either a RC servo motor or speed controller. Uses 16-bit PWMs for ultra-accurate duty cycles (but limiting the number of PWM channels to 4).

### Remarks:

- Uses Port P (all channels)
  - PWM67 (RC PWM channel 3) outputs on pin 109
  - PWM45 (RC PWM channel 2) outputs on pin 111
  - PWM23 (RC PWM channel 1) outputs on pin 1
  - PWM01 (RC PWM channel 0) outputs on pin 3

### Author:

Zebb Prime

### Date:

14/9/06

### Version:

1.0 beta
6.2 Define Documentation

6.2.1 #define PWM67_38400

Macros to define the behaviour of the PWM unit.

PWM2367RES_[025][05][1] - Resolution of PWM23 and PWM67 (0.25, 0.5 or 1us)
PWM0145RES_[025][05][1] - Resolution of PWM01 and PWM45 (0.25, 0.5 or 1us)

0.25us -> min f=62Hz, 0.5us -> min f=32Hz, 1us -> min f=16Hz

PWM[01][23][45][67]FREQ - PWM frequency (Hz), see above for min, max ~400Hz
PWM[01][23][45][67]DEF - Default PWM period (us)

6.3 Function Documentation

6.3.1 void PWM_Init (void)

Initialises the PWM module.

Sets the registers for the PWM signals. The 8 PWM channels are linked together to create 4x16bit PWM channels. The last channel is set up to generate a frequency to drive bit-banging serial output. The frequency and resolution of the other channels are defined by macros.

Author:

Zebb Prime

Date:

14/9/06

6.3.2 void PWM_T_Set (char channel, unsigned int period)

Sets the output period for a PWM channel.

Sets the output PWM period (the control signal for RC PWM) to be the (channel resolution)* (input parameter). The PWM period is saturated to between 750us and 2250us. Invalid channel requests are ignored.

Parameters:

channel: The RC PWM channel to change, (0-2 Valid)
period: output period width (=period*resolution)

Author:

Zebb Prime
Date:

14/9/06
# Functions

- void `Radio_TX_BB` (unsigned char *p, unsigned char length)

## 7.1 Detailed Description

Pushes the array of characters to be sent over the RF link onto the sending ring-buffer, along with the preamble, length and crc8 as defined as the protocol in "radio_rx_sci1.c".

See also:

- `timer.c`

### Author:

Zebb Prime

### Date:

22/8/06

Version:

1.0

## 7.2 Function Documentation

### 7.2.1 void Radio_TX_BB (unsigned char *p, unsigned char length)

Pushes the character array to be sent via the RF link.

#### Parameters:

- `*p`: pointer to the array of characters.
- `length`: The length of the array (obviously limited to 256)
Functions

- void RF_Proc_Init (void)
  
  *Initialise the RF packet processing module.*

- void RF_Proc_Tick (void)
  
  *Timer Tick (for the timeout value).*

- void RF_Process (void)
  
  *Process the incoming data through the RF link.*

8.1 Detailed Description

Processes the data received over the RF link once it arrives. Operate outside of interrupts to keep the time-critical functions (such as the bit-banging serial port) running as fast as possible.

Author:

Zebb Prime

Date:

22/9/06

Version:

1.0

8.2 Function Documentation

8.2.1 void RF_Proc_Init (void)

Initialise the RF packet processing module.
Radio Receiver processing prototypes.
8.2.2  void RF_Proc_Tick (void)

Timer Tick (for the timeout value).

This function should be called at a fixed interval, probably from a timer interrupt or similar. If a timeout occurs (time without receiving any more information) the state resets.

8.2.3  void RF_Process (void)

Process the incoming data through the RF link.

Operates as a state machine, reading data from the RF ring buffer. The protocol is as follows: | 'commandType' | 'data' | 'data' | etc. The only protocol implemented thus far is for controlling the servo motor periods. It is as follows: | 's' | 'channel (2bits)' | 'period msb (6bits)' | 'period lsb' | Other commands can/will be implemented as required.

Precondition:

The radio ring buffer is non-empty (must be checked before calling this function).
9 E:/Zebb/Working/Programming/PVTOL/PVTOL_-MiniDragon/Sources/timer.c File Reference

#include <mc9s12dp256.h>
#include "imu_0x31_sci0.h"
#include "radio_rx_sci1.h"
#include "rf_rx_process.h"
#include "adc.h"
#include "rb_.h"

Functions

- __interrupt void TOC7_ISR (void)
  
  ECT channel 7 interrupt service routine.

- void TOC7_Init (void)
  
  Initialises the ECT module for 2kHz interrupts on channel 7.

- void TOC0_BBOut_Start (void)
  
  Start sending data from the ring-buffer.

- void TOC0_BB_SendBreakStart (void)
  
  Sends a break character, then begins sending data from the ring-buffer.

- void TOC0_BBOut_Init (void)
  
  Initialise the bit-banging serial output.

- __interrupt void TOC0_BBOut_ISR (void)
  
  bit-banging serial output interrupt service routine.

9.1 Detailed Description

Controls the Enhanced Capture Timer on the 9S12. Creates timed interrupts using ECT channel 7 at 2kHz. This is used to do timed tasks such as start the ADCs (@10Hz) and flash the LED (@1Hz). Also provides the tick for other software modules (for timeouts, etc.)
Channel 0 is used as a bit-banging serial port. The interrupts are timed for a certain baud rate and manually send PORT T Pin 0 high or low depending on the byte to be sent.

Remarks:

Uses Port T0 (pin 9) as the bit-banging serial output.

See also:

"imu_0x31_sci0.c"
"radio_rx_sci1.c"
"rf_rx_process.c"
"adc.c"

Author:

Zebb Prime

Date:

27/9/06

Version:

1.0

9.2 Function Documentation

9.2.1 __interrupt void TOC0_BBOut_ISR (void)

bit-banging serial output interrupt service routine.

Sets the output Port T0 to be high/low according to the bits of the byte being send. Currently consumes about 5% of the CPU time at 38400 bps while sending (0 while idle).

9.2.2 __interrupt void TOC7_ISR (void)

ECT channel 7 interrupt service routine.

Updates the modules and decrements the divider variables.