Final Report: Energy Management Subsystem

LA-SUB--93-244

Submitted in compliance with Task No. 4,
Ref. 9-X63-0402J-1 to the Los Alamos National Laboratory

by
The New Mexico Institute of Mining and Technology
Socorro, New Mexico 87801

RECEIVED
FEB 14 1997
OSTI

Principal Investigator:
Colin W. Wightman
Assistant Professor of Electrical Engineering
Telephone: 505-835-5708
FAX: 505-835-5707

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Dr. Colin Wightman:

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DLC

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

1 Background

In today's environment-conscious world, increasing levels of automotive emissions have been recognized as a major source of pollutants and greenhouse gases. Despite increasingly stringent tailpipe emission standards, the increased use of the automobile has more than offset the lowered per-vehicle emissions. Consequently, there is a great deal of interest in so-called "zero-emission vehicles", such as electric and hybrid-electric automobiles. Although very attractive in terms of emissions, these vehicles present some design challenges which are not generally part of conventional automotive design.

One such challenge is the development of an effective energy management strategy for the vehicle. While a conventional automobile has an engine whose power output far exceeds the average vehicle needs, hybrid electric vehicles generally have very limited energy reserves and efficiency in the use of these reserves is paramount if acceptable overall performance is to be achieved. Many aspects of the vehicle design (such as aerodynamics, powertrain design, gross weight, etc.) strongly influence the overall vehicle efficiency. However, the actual performance achieved by any given driver is strongly dependent on his or her driving skills. One way to reduce the effect of differences in driving skills is to provide for automatic accelerator control, permitting the vehicle to be driven in an efficient manner without necessitating extensive driver training.

This report describes the accelerator/brake control system developed as part of the energy management strategy used on the *Zia Roadrunner*, New Mexico Tech's entry in the 1993 Sunrayce for solar-electric hybrid vehicles. In this report, we describe the theoretical aspects of the accelerator control algorithm, and overview the hardware and software design used to implement the control algorithms.

2 General Description

The system developed provides for automatic control of the accelerator setting and an anti-lock brake function. The accelerator control can operate in one of two different modes: constant vehicle speed, or constant battery current. The system is implemented digitally using a Motorola 68HC11 microprocessor. Sensors built into the vehicle provide measurements of power consumption, vehicle and wheel speeds. A simple control panel allows the driver to interact with the system, and fail-safe circuitry ensures that the driver can regain normal control of the vehicle in the event of any malfunction.

3 Theory of Operation

A series of experiments were conducted to estimate the parameters for a first-order model of the vehicle dynamics. Although there are a number of higher-order effects, the vehicle's inertia and drag forces completely dominate them at any legal speed. Of course, the aerodynamic drag is not well described by a first-order model but, when considering a small range

of speeds around a setpoint, the drag can be linearized with a fair degree of accuracy. By choosing to model the system by the linearized drag forces in the range of 10 to 20 MPH, the effect of higher speeds will be to increase the damping and thus increase the response time of the system. We determined that the increased response time was more than offset by the simplicity of the resulting controller and, furthermore, that the slower response might actually be desirable at higher speeds.

The open-loop transfer function between the motor input voltage and the vehicle's speed was determined to be

$$H(s) = \frac{0.9157}{2.189s + 1.04}.$$

The control implemented is a simple proportional control with a gain of 1.09. In actuality, the motor torque is limited so the proportional control is possible only when the actual speed is within 7 mph of the desired speed. When the difference between the actual and desired speeds is greater than this, the full motor torque is used.

4 Hardware Description

In this section, we present a high-level description of the system architecture. We begin with a brief overview of the relevant aspects of electric vehicle design, and then provide a system description followed by a review of the principal functional blocks.

4.1 Vehicle Design

Virtually all electric (and hybrid electric) vehicles share a common design for the motor control connections. Having the accelerator and brake pedal directly control the flow of electricity to and from the motor would present a number of problems, foremost of which is the safety hazard associated with having high power components directly connected to the driver. A more intractable problem occurs when brushless or AC motors are used requiring sophisticated automatic switching systems. To avoid these problems, a power controller is used to modulate flow of power to the motor, and a low-power control voltage is used to signal the desired speed and brake force to the controller. Thus, the drivers pedals need only generate control the low-power control signals, a much simpler and safer task.

Because of this basic design approach, installing an automatic control system for the accelerator and brake is rather straight-forward. When operating, such a system need only intercept the control signals from the pedals, and substitute its own outputs for them. Moreover, since the control signals are low-power voltages, the automatic controller need not utilize any high-power components, which reduces cost and weight.

4.2 System Overview

In addition to the motor power controller, the accelerator control system involves additional signal processing and control (See Figure 1) From the accelerator and brake pedals, the

signal voltages are digitized, and processed. The output control voltages are then passed to the motor controller. In case of a failure, the pedal controls will be directly connected to the motor power controller by the fail-safe system.

With the power on but speed control disengaged, the microprocessor simply passes the pedal voltage through to the motor controller. If control is engaged, the microprocessor will ignore the accelerator pedal and produce a control voltage based on the control algorithms.

4.3 Human Interface

In order to run the accelerator control system, the operator must have the following controls:

- Power (on/off)
- Control (engage/disengage)
- Control mode (speed/current)
- Setpoint entry (up/down)

A display panel mounted in the cockpit of the vehicle houses the required controls (See Figure 2) A toggle switch controls power to the system, and a display light indicates power status. A push button is used to toggle the speed control on and off, and a second display light is used to indicate when the controller is engaged. A second toggle switch selects constant speed or constant current control. Two pairs of seven-segment displays display the actual value and the set value (Either speed or current, depending on the mode selected). The setpoint value can be changed by incrementing or decrementing it using the Up/Down pushbutton. Holding these buttons down will cause the setpoint to count up or down as long as the button is held.

4.4 Fail-Safe Design

The fail-safe system uses a mechanical relay to return motor control to the pedals in case of a system failure (See Figure 3) The reed relay receives input from one of two places: the pedal voltage, or the D/A controlled by the microprocessor. As long as the relay is triggered at intervals of less than 100 msec, the D/A output is connected to the motor controller. The trigger signal is sent from the main loop of the microprocessor program. Therefore, if the program ever fails to return to the main loop within 100msec, the pedal be reconnected directly to the motor controller.

4.5 Brake Control

In three-wheel vehicles, such as the New Mexico Tech solar car, the drive wheel is typically in the rear. Therefore, the rear wheel is the one to receive regenerative braking. In order to recover as much energy as possible with the regenerative braking, maximum regenerative

braking is applied to the rear wheel before the front hydraulic brakes are engaged. Because of this, three-wheel vehicles are extremely susceptible to rear wheel skids. This skidding can destabilize the vehicle potentially resulting in a spinout. To avoid this, an anti-lock brake system releases the regenerative brake if a rear wheel skid occurs. This is easily accomplished as the regenerative brake force is controlled by low-power voltage signal from the brake pedal. The system utilizes optical speed sensors on both the front and rear wheels. As long as wheels speeds are fairly close to each other (some tolerance is needed because the wheels will turn at different speeds as the vehicle travels around turns), the control signal from the brake pedal is passed to the motor power controller. If the rear wheels rotation is substantially less than that of the front wheel, however, the brake signal voltage is set to zero (no brake force) until the wheel speeds match again.

5 Software Description

In this section, we describe the structure of the microprocessor program and briefly describe the function of each interrupt and subroutine. The source code for the program is attached to this report.

5.1 Interrupts

The program has three interrupts. Two are input capture interrupts that are generated by a pulses from the wheel sensors. Each interrupt is associated with one of the wheels. On receiving a rising edge from the wheel sensor, the corresponding interrupt is triggered and the handler increments a memory location corresponding to the appropriate wheel.

The third interrupt is a real time interrupt that is generated every 33msec. The corresponding interrupt handler performs two functions: First, it compares the pulses from the front wheel and the pulses from the back wheel. stores the difference, and clears the pulse counts. Second, it collects and averages the motor speed (as signaled by the motor power controller) over a 1 second period, and sets a flag when the new motor speed average has been completed..

5.2 Main Loop

The first function in the main loop is to trigger the relay involved in the fail-safe system (refer to Figure 3) Following this command are seven subroutine calls. The subroutines in the main loop are as follows: cntl_chk, read_ad, get_dta, accel, brake, set_dac, and display. Following execution of these subroutines the loop will is repeated.

5.3 Subroutines

The cntl_chk subroutine checks the various controls from the display panel and sets corresponding flags. In cntl_chk the program jumps to various flag checks. On_chk first checks a

brake flag. If it is set the brake is on and controller is considered off so the control system engaged flag is ignored. If the brake flag is not set, the status of the engage button is checked and a flag is toggled with each push. In addition, a display panel light is turned on when the control system is engaged. Up_chk and dwn_chk read and accumulate the up and down button pushes and stores the set value in 'set_pnt.' Spd_chk sets a flag when constant speed is the selected mode and clears it when constant current is the set mode.

The read_ad subroutine simply uses the microprocessors analog-to-digital function to digitize the accelerator and brake pedal inputs and store them in 'acl_set' and 'brk_set' respectively.

The get_dta subroutine takes the average motor speed obtained in the real time interrupt and averages these 1 second averages over a set time period. Currently the program is set to average every two values from the interrupt. Therefore the motor speed is updated every two seconds.

The accel subroutine implements the accelerator control of the system. If the control system is engaged, the program will check the spd_on flag to see whether to control by constant speed or constant current. After subtracting either 'act_spd' or 'amps' from the set value, the program will do one of two things. If the difference is negative (If the actual value is higher than the set value) the program will send 0 volts to the motor controller to slow the speed down. if the difference is positive the difference will be checked. If the difference is large, maximum voltage will be sent to the motor controller (this is like flooring the pedal.) If the difference is small, the set value will be sent to the controller.

The brake subroutine checks to see if the brake voltage is above a certain threshold. If it is the 'brk_flg' is set, the control system is turned off, and the anti-lock function is invoked. The abs subroutine simply checks the value of the difference between wheel speeds obtained in the real time interrupt and sends 0 volts to the motor controller if the difference is above a set threshold.

The set_dac subroutine sends the control voltages to the digital to analog interface. From there they are sent to the motor power controller.

The display subroutine sends the set value and the actual value to the seven segment displays of the display panel using the serial output function of the microprocessor.

6 Conclusion

The accelerator and brake control functions provided by the system developed here are a small, but important part of any hybrid electric vehicle. Although the system is rather simple, its effects on the overall performance of the vehicle can be quite substantial. In the future, we hope to embed these functions within a larger instrumentation and control system so that factors such as battery charge state and solar insolation can be considered as inputs to a more sophisticated algorithm.

Figures

Figure 1

Figure 2

Figure 3

INPUT SIGNAL
CONDITIONING

$V_{DD} = 5V$ $u1: AD7528$ port₂
 $R_1 = 10k$ $u2, u3: 741$
 $R_2 = 1k$ $u7 - u11: 7411$
 $R_3 = 20k$ $u5, u6: read$
 $C1 = 10 - 15pF$ $relays$
 $C2 = 10\mu F$

Source Code

Software to Implement Speed Control

-----Speed Control-----

```

get_dta    ldaa    spd_flg                ;checks for data ready flag set
           anda    #%00000001
           beq     end2
           ldy     #bfr_bse
           ldab    pntr
           aby
           ldaa    new_dta
           staa    0,y
           ldy     #bfr_bse
           clr     sum
           clr     {sum+1}
loop       clra
           ldab    0,y
           addd    sum
           std     sum
           iny
           cpy     #{bfr_bse+1}
           ble     loop
           ldd     sum
           lsr     lsr
           ldaa    #vlt_rpm
           mul
           ldab    #!100
           mul
           std     act_spd
           ldaa    pntr
           inca
           anda    #%00000001
           staa    pntr
           clr     spd_flg                ;clears data ready flag
end2       rts

```

```

accel      pshx
           ldaa    ctl_on
           anda    #%00000001
           beq     done
           ldaa    spd_on
           anda    #%00000001
           beq     pwr_ctl
           ldd     set_pnt
           subd    act_spd
           bra     control
pwr_ctl    ldd     set_pnt
           subd    amps
control    bmi     slw_dwn
           ldx     #gain
           idiv
           xgdx
           cmpa    #!0
           beq     sm_err
           ldab    #$ff
sm_err     stab    acl_set

```

```

slw_dwn    bra    done
           clr    acl_set
           bra    done
done        pulx
           rts
*
*-----Controls braking mechanism-----
*
brake      jsr    read_ad
           ldab   #!13                ;check if braking above threshold
           cmpb   brk_set
           bhi    end3
           inc    brk_flg            ;if braking turn cruise cntl off
           bset   porta,x,%00100000
           jsr    abs                ;do anti-lock
end3        rts
*
*-----Anti-Lock Brake Subroutine-----
*
abs         ldaa   dif
           ldab   maxdif              ;specified max. dif. betw. wheels
           cba
           bls    end4                ;if dif </= maxdif do nothing
           ldaa   #$00
           staa   brk_set            ;turns off brake
end4        rts
*
*-----Outputs data to D/A-----
*
set_dac     bclr   portc,x,$01        ;accelerate - DAC A
           ldab   acl_set
           stab   portb,x
*           bset   portc,x,$01        ;decelerate - DAC B
*           ldab   brk_set
*           stab   portb,x
           rts
*
*-----Output to Hex Displays-----
*
display     ldaa   #%00111000        ;configure serial output
           staa   ddrd,x
           ldaa   #%01011100
           staa   spcr,x
*
           pshx
           pshy
           ldy    #basreg
           ldaa   spd_on
           cmpa   #!0
           beq    shw_amp
           ldd    act_spd
           ldx    mph_rpm
           idiv
           xgdx
           jsr    hex2bcd
           ldab   lsb
           jsr    flip_b
           stab   spdr,y
           brclr  spsr,y,$80,*

```

```

        ldaa    spsr,y          ;two loads to clear
        ldaa    spdr,y

*

        ldd     set_pnt
        ldx     mph_rpm
        idiv
        xgdx
        jsr     hex2bcd
        ldab    lsb
        jsr     flip_b
        stab    spdr,y
        brclr   spsr,y,$80,*
        ldaa    spsr,y
        ldaa    spdr,y
        bset    portd,y,%00100000
        bclr    portd,y,%00100000
        bra     fini
shw_amp  ldd     amps
        jsr     hex2bcd
        ldab    lsb
        jsr     flip_b
        stab    spdr,y
        brclr   spsr,y,$80,*
        ldaa    spsr,y          ;two loads to clear
        ldaa    spdr,y

*

        ldd     set_pnt
        jsr     hex2bcd
        ldab    lsb
        jsr     flip_b
        stab    spdr,y
        brclr   spsr,y,$80,*
        ldaa    spsr,y
        ldaa    spdr,y
        bset    portd,y,%00100000
        bclr    portd,y,%00100000
fini     puly
        pulx
        rts

*
*-----Performs A/D conversion-----
*
read_ad  sei                      ;disables interrupts to avoid conflict
        ldab    #$10
        stab    adctl,x
        brclr   adctl,x,$80,*    ;branches to itself
        ldaa    ctl_on
        anda    #%00000001
        bne     no_acl
        ldab    adr1,x          ;A/D off PE1
        stab    acl_set
no_acl   ldab    adr2,x          ;A/D off PE2
        stab    brk_set
        cli
        rts

*
*-----flips bits MSB->LSB in accum a
*
flip_b:

```



```

    pshx
    psha
    ldx    #$0008
    clra
flip_loop:
    lsll
    lsrll
    bcc    is_zero
    oraa    #$01
is_zero:
    dex
    bne    flip_loop
    tab

    pula
    pulx
    rts

```

```

*
*-----converts 16-bit binary in accum d to BCD
*
hex2bcd:

```

```

    pshx

    ldx    #!10
    idiv
    stx    msb
    andb    #$0f
    std    lsb

    ldaa    {msb + 1}
    lsll
    lsll
    lsll
    lsll
    anda    #$f0
    oraa    {lsb + 1}
    staa    lsb

    pulx
    rts

```

```

*
*-----Implements anti-lock braking system-----
*

```

```

tic1_ser  ldx    #basreg
           inc    fwct                ;counts pulses from front wheel
           bset    tflg1,x,$fb        ;clears ic1 flag
           rti

tic2_ser  ldx    #basreg
           inc    rwct                ;counts pulses from rear wheel
           bset    tflg1,x,$fd        ;clears ic2 flag
           rti

rti_ser   pshy

```

```

ldx    #basreg
ldaa   fwct
suba   rwct           ;subtracts back pulses from front pulses
staa   dif           ;stores value in memory
ldaa   #$00
staa   fwct         ;clears pulse counts
staa   rwct
bset   tflg2,x,$40   ;clears rti flag
ldab   #$10         ;the rest measures motor speed
stab   adctl,x
brclr  adctl,x,$80,* ;branches to itself
ldab   adr4,x
clra
addd   tmp_spd
std    tmp_spd
inc    count         ;when count = # samples
ldab   count
cmpb   #samples      ;samples=32
blt    wait
ldd    tmp_spd
lsrd
lsrd
lsrd
lsrd
lsrd
stab   new_dta
inc    spd_flg
clr    count
clr    tmp_spd
clr    {tmp_spd+1}
wait   puly
      rti

```

===== ram storage =====

```

*
acl_set   rmb    1           ;set accelerate output for D/A
brk_set   rmb    1           ;set brake output for D/A
brk_off   rmb    1
fwct      rmb    1
rwct      rmb    1
dif       rmb    1
spd_on    rmb    1
pwr_on    rmb    1
ctl_on    rmb    1
set_pnt   rmb    2
mtr_spd   rmb    1
tmp_spd   rmb    2
count     rmb    1
spd_flg   rmb    1
bfr_bse   rmb    4
pntr      rmb    1
new_dta   rmb    1
avrg      rmb    1
hi_flg    rmb    1
sum       rmb    2
act_spd   rmb    2
err_sgnl  rmb    2
brk_flg   rmb    1
nxt_up    rmb    1
lst_up    rmb    1

```

nxt_dwn	rmb	1
lst_dwn	rmb	1
mph_rpm	rmb	2
lsb	rmb	2
msb	rmb	2
ad_tmp	rmb	2
amps	rmb	2
amp_set	rmb	2
rpm_set	rmb	2

*****Debbie Crane - Sunrayce Motor Control System*****

#include 'template.asm'

evbram equ \$c000

* set constant values

gain equ !2

maxdif equ !2

brk_lim equ !25

samples equ !32

vlt_rpm equ !55 ;voltage to rpm - must be multiplied by 100

*

=====Configuration of Registers=====

*

org tic1_vec

jmp tic1_ser

org tic2_vec

jmp tic2_ser

org rti_vec

jmp rti_ser

org evbram

ldx #basreg

ldy #\$00

ldaa #%01010101

;configure output pins of portc

staa ddrc,x

clr portb,x

clr portc,x

ldaa #%10101111

staa porta,x

ldaa #\$80

;set up A/D conversion

staa option,x

clr tctl1,x

ldaa #%00000110

;enables input capture interrupts

staa tmsk1,x

;clears input capture flags

staa tflg1,x

;configure input captures to rising edge

ldaa #%00010100

staa tctl2,x

ldaa #%01000000

;enables rti interrupt

staa tmsk2,x

ldaa #%01000000

;clears rti flag

staa tflg2,x

ldaa #%00000011

;sets rti interrupt rate

staa pact1,x

ldd #\$0053

std mph_rpm

;clear all memory locations initially

clr dif

clr rwct

clr fwct

clr spd_on

clr ctl_on

```

clr    brk_set
clr    hi_flg
clr    brk_flg
clr    spd_flg
clr    lst_up
clr    tmp_spd
clr    {tmp_spd+1}
clr    count
clr    amp_set
clr    {amp_set+1}
clr    rpm_set
clr    {rpm_set+1}
cli

```

```

*
*=====main program=====
*

```

```

main    bset    portc,x,$40
        jsr     cntl_chk
        jsr     read_ad
        jsr     get_dta
        jsr     accel
*        jsr     brake
        jsr     set_dac
        jsr     display
bclr    portc,x,$40
bra     main

```

```

*
*-----Check Driver Controls-----
*

```

```

cntl_chk  jsr     on_chk
          jsr     up_chk
          jsr     dwn_chk
          jsr     spd_chk
          clr     brk_flg
skip      rts

```

```

*-----
on_chk    ldaa    brk_flg
          bne     off
          ldaa    portc,x           ;loads value of portc
          anda    #%00000010
          bne     toggle
          clr     hi_flg
off      rts

```

```

toggle    ldaa    hi_flg
          bne     no_tog
          ldaa    porta,x
          eora    #%00100000
          staa    porta,x
          ldaa    ctl_on
          eora    #%00000001
          staa    ctl_on
          inc     hi_flg
no_tog    rts

```

```

*-----
up_chk    ldaa    portc,x
          anda    #%00001000

```

```

      staa     nxt_up
      beq      no_up
      ldaa     lst_up
      bne      end_up
      ldaa     spd_on
      cmpa     #$0
      beq      set_amp
      ldd      rpm_set
      addd     mph_rpm
      std      rpm_set
      bra      shft_it
set_amp ldy      amp_set
      iny
      sty      amp_set
shft_it ldaa     nxt_up
      staa     lst_up
      bra      end_up
no_up   clr      lst_up
end_up  rts

```

```

; if present edge is zero no rising e
; if lst_edg and nxt_edg both high
; then no edge

```

*-----

```

dwn_chk ldaa     portc,x
      anda     #%00100000
      staa     nxt_dwn
      beq      no_dwn
      ldaa     lst_dwn
      bne      end_dwn
      ldaa     spd_on
      cmpa     #!0
      beq      set_amp2
      ldd      rpm_set
      subd     mph_rpm
      std      rpm_set
      bra      shft_it2
set_amp2 ldy      amp_set
      dey
      sty      amp_set
shft_it2 ldaa     nxt_dwn
      staa     lst_dwn
      bra      end_dwn
no_dwn   clr      lst_dwn
end_dwn  rts

```

```

; if present edge is zero no rising e
; if lst_edg and nxt_edg both high
; then no edge

```

*-----

```

spd_chk ldaa     portc,x
      anda     #%10000000
      beq      not_spd
      ldaa     #$01
      staa     spd_on
      ldd      rpm_set
      std      set_pnt
      bra      end_spd
not_spd  clr      spd_on
      ldd      amp_set
      std      set_pnt
end_spd  rts

```

*