

DEBRA: Digital Emergency Brake Response Alert System

Abstract

Rear-end collisions on US highways are the single greatest type of traffic accident. This paper describes a stand alone system designed to alert drivers when other cars in close proximity decelerate quickly. The system distinguishes between cars decelerating slowly and quickly. It flashes the center high mounted stop lamp to visually alert following vehicles and also uses an RF transmission to provide audible or other alerts to other vehicles in close proximity. Simultaneously, the DEBRA system also allows for performance monitoring. Using accelerometers, it can measure 0-30, 0-60, ¼ mile, and braking times.

Charvak Karpe, Nathan Ackerman
6.111 Final Project
May 13, 2004

Introduction

Background

Currently, the single largest type of highway accident is a rear-end crash where one car does not slow down quickly enough and crashes into the rear of another car. Highway rear-ending accidents account for 23% of all accidents and cause almost a million injuries each year in the US alone. When a car applies its brakes, three brake lights illuminate in the rear of the car to alert other cars. Currently, brake lamps only inform other cars of braking and neglect to relay any information on how quickly the car is braking. Often, the cause of the collision is that the following driver does not notice how quickly the car in front is braking. Thus, we believe that rear-end collisions would decrease if there were some warning for when cars decelerate quickly.

Goals

The primary purpose of the DEBRA system is to enhance highway safety through distinguishing when cars brake quickly. The components required to accomplish the primary goal of DEBRA also allow for monitoring and measurement of car performance with minimal additional overhead. Thus, it is a secondary goal of the DEBRA system to measure car performance using a standard set of performance measuring trials.

Overview

The DEBRA system alerts other cars when the originating car decelerates quickly by pulsing the center brake lamp instead of leaving it constantly illuminated and also by sending wireless warning signals to other cars. Other drivers will visibly detect the change in the brake lamp. Additionally, other cars equipped with DEBRA will also receive the wireless warning and play an audible warning for the driver as an additional indicator. For measuring performance, cars equipped with a DEBRA system will also contain a LCD display in the cockpit of the car along with a panel for selecting performance testing. Drivers will be able to press a button to start a performance trial and read instructions and results as output from the LCD screen.

The overall block diagram of the system can be seen in Figure 1. In addition, there are also a few pieces of physical hardware. Since the main purpose of the project is to design hardware for installation in a car, we had to redesign hardware to be compact. The lab kits typically used for projects were far too large to reasonably fit in a car, thus we relocated all of the parts to a separate PCB which interfaced directly with the power supply and the FPGA.

Wireless Transmission

In order for cars equipped with DEBRA to send and receive messages, we used the Chipcon CC1010 wireless chip and supported libraries. We used this chip on the supplied CC1010IDE evaluation boards supplied from Chipcon. Using the IDE boards and software provided by Chipcon, we coded a small wireless program and uploaded it to the CC1010. The software utilizes a simple packet protocol (SPP) where all packets contain a destination and source address. In addition, we configured the SPP to send acknowledgement packets back to the sender after a packet was received. If the sender does not receive the acknowledgement packet within the specified window, the sender will continue to resend the packet up to four times before the sender stops trying. Code for the CC1010 module can be found in the appendix.

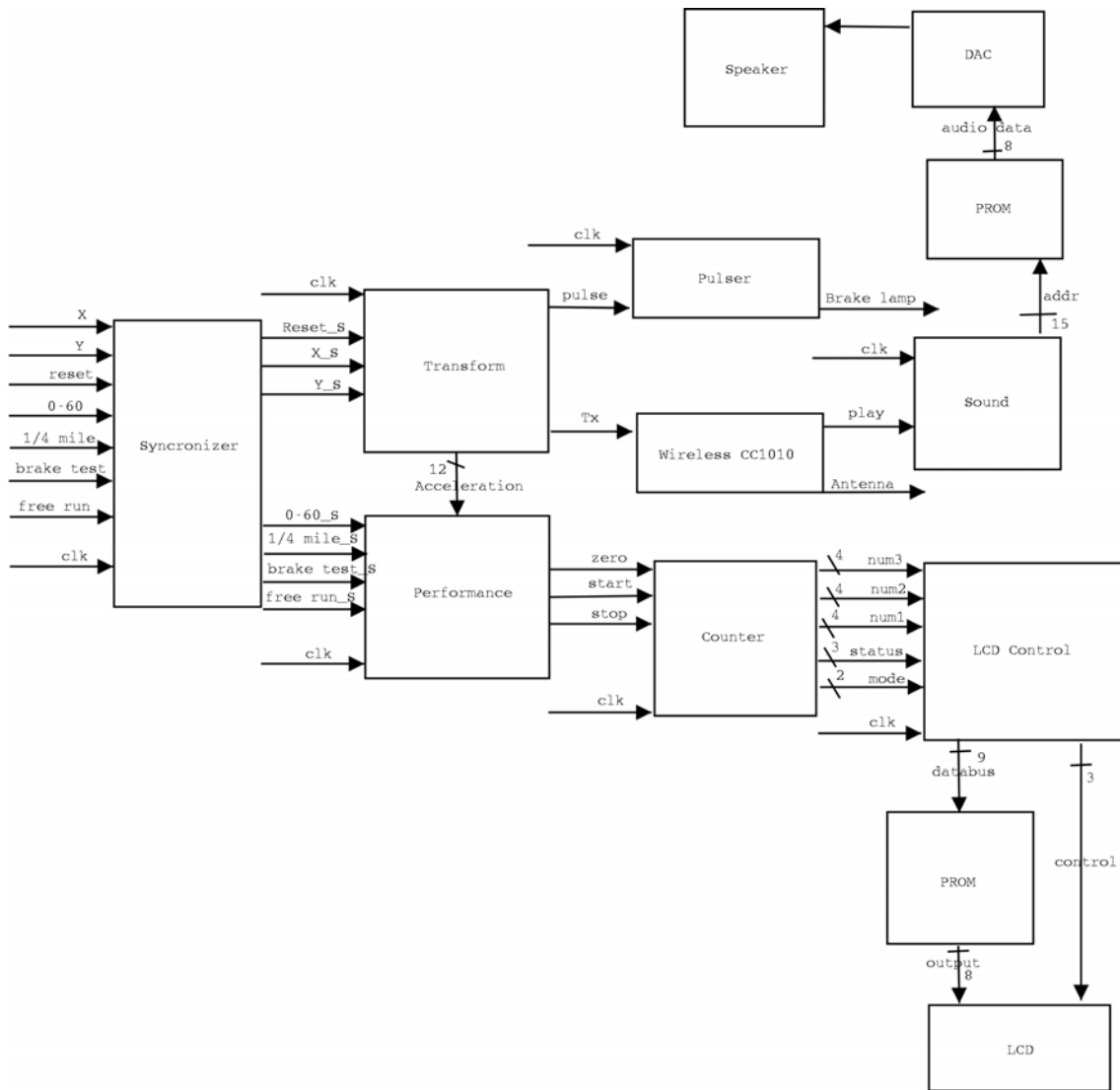


Figure 1: Block diagram of overall system

Each CC1010 module waits in receiver mode and listens for packets. When a packet is received, the module switches to transmitter mode to send an acknowledgement. After the acknowledgement is sent, the CC1010 goes back to receiver mode and listens for new packets. At the same time, the CC1010 pulls a data pin high which represents the receipt of a packet from another DEBRA system. This event occurs asynchronously and is sent to the main FSM for processing. Furthermore, when a car wishes to send a packet, a control signal is sent to the CC1010 which tells it to switch into transmitter mode and send a packet. Lastly, all packets are sent out using a broadcast command which does not specify a destination or source address. This is done so that all CC1010 modules, and subsequently, all cars in the range of the DEBRA system will receive the information.

LCD controller

Users will gauge performance and receive instructions for performance testing through an LCD screen mounted in the cockpit of the car. The LCD is controlled by a Hitachi 44780 standard LCD controller. The 44780 has 14 pins and are assigned as follows in table 1.

pins	Description
1	GND
2	Vcc
3	Contrast
4	Instruction/register sel
5	Read/write
6	Enable clk
7-14	Data pins

Table 1: Pin out for Hitachi 44780 LCD controller

In order to use the LCD, data is placed on the data pins and then clocked to the controller. When the enable clock pin goes high, the LCD controller registers the data. To display text on the device, the user must first hold the data steady for 42microseconds. After the hold time, a pulse on enable clock will register the input and if pin 4 is low, a character will be displayed in the first position of the LCD according to a map lookup for the data pins. The cursor will be incremented once so that the next time the same action occurs, a letter will be written next to it instead of replacing it.

The FPGA does not directly communicate with the LCD controller. Instead, a PROM is used in between the FPGA and the LCD controller. Since the data is input to the LCD controller in a serial fashion, a PROM is used to stream data from a range of addresses. This technique of using a PROM saves much space in the FSM required to control the LCD screen. For example, it would take a FSM 80 different states to write 40 characters to the LCD screen if a PROM were not used. In contrast, the same 40 characters could be written to the LCD in only 4 states using a PROM.

There are 5 different modes that the LCD display can be in. They are "System Ready", "0-60 Acceleration test", "Braking test", "1/4 mile test", and "free run" mode. Accordingly, each of these modes has a status associated with it. In each case, either the trial has yet to begin, has started, or has completed. Thus, the PROM stores 8 different lines of text in their entirety along with numeric characters. The PROM programming file along with the verilog code for the LCD controller can be found in the appendix as well.

Sound controller

When a car equipped with a DEBRA system receives wireless packets from another car, it means that another car is decelerating quickly and potentially presenting a dangerous situation. Accordingly, the car receiving the packet should play some auditory warning to the driver in order to alert the driver to the potentially dangerous situation. In the initial design, a sound of a human saying "warning" was used.

Sound playback is achieved through a AM28F512 PROM and a AD588 DAC. The DAC is wired to be always on with ~enable always low. Furthermore, the DAC input bus is also directly wired to the PROM output so that the FPGA simply runs through the addresses of the PROM to play a sound. The sample frequency used was 16Khz and so the FPGA running at 10MHZ must count to 625 before going to the next address. The verilog code for the Sound controller can also be found in the appendix.

Power supply

The main logic and FPGA run from a 5 volt supply while the Chipcon CC1010 and corresponding wireless module run from a 3.3 volt supply. To further complicate the problem, cars have an internal supply which ranges from 11 volts to 15 volts under normal operation. Thus, simple voltage dividers with power resistors could not be used and two regulated power supplies were built specifically for the car. The input can range from 9 volts to 27 volts. The power supply can also source 1.5 amps of current through the 3.3 volt supply and 17 amps of current through the 5 volt supply. The schematic for the power supply can be found in the appendix.

Pulser

The pulser module receives a signal from the transform module when the car is braking hard. When that happens, it pulses the brake light at a frequency of 10 Hz. It continues the pulsing for one second after the emergency braking has ceased. This is because emergency braking often happens in a very short amount of time, but the resulting difference in velocities between the leading and following cars lasts until the following car has reacted and slowed down.

Counter

The counter module functions as the timer for the different performance tests. It is controlled like a stopwatch. It has three control inputs: start, stop, and zero. Zero resets the counter. Start and stop tell the counter to start or stop counting. The counter has a 4-bit output for each decimal digit of the display. This project uses 3 digits to provide a count up to 99 seconds at tenth of a second increments.

Synchronizer

Our synchronizer does a little more than take analog user inputs and synchronize them to the clock. It also debounces the inputs from the accelerometer. Bounce happens when a one bit input is supposed to transition once from low to high or high to low, but it transitions repeatedly instead due to underdamped oscillations. The synchronizer changes state whenever an input from the accelerometer changes and does not allow it to change for 25 microseconds after that. Accelerometer inputs are not expected to change in less than 500 microseconds.

Accelerometer

The Analog Devices ADXL202EB 2-axis, +/- 2g MEMS accelerometer provides input data to the system. It is configured to have a bandwidth of 10 Hz to filter out engine and road vibrations. It is also configured to provide a 1.7 ms period signal with a duty cycle that varies depending on the acceleration. A change in the duty cycle of 12.5% corresponds to 1g of acceleration.

Transform

The transform module is the most computationally intensive part of the system. It has two 15-bit counters for each axis of the accelerometer. When the accelerometer signal transitions to high, both counters are reset to zero and begin counting up. When the accelerometer signal goes low, one counter stops counting and the other continues. The signal goes high again, the first counter is multiplied by 8192 and both are sent to the divider. After about 35 cycles, the results of the division return. Duty cycle values between 0 and 100% correspond to quotient values between 0 and 8192. This makes 1024 correspond to 1g of acceleration. 4342 is subtracted from the quotient because a 53% duty cycle corresponds to zero acceleration for the particular chips that were used. This creates a 12-bit twos complement number between -2048 and 2047 that corresponds to -2g to 2g.

When a user turns on the system, he needs to press the reset button while stationary on level ground. Reset initializes all values so the accelerometer data can be read. It waits for 0.1 second for the current values to be available and then stores them into x-reset and y-reset. This tells the system which way is down (only upward force on the car). From then on, it can calculate the backward force on the car created by braking by performing the following transform:

$$\text{braking} = x * \text{yreset} - y * \text{xreset}$$

The negative of that is the acceleration. The transform module outputs the current acceleration value and a single bit braking value if the car is braking faster than 0.3 g's.

Performance

The performance module takes the 12-bit twos complement acceleration signal from the transform and adds it to a 24-bit accumulator once every millisecond. This effectively integrates the acceleration to find velocity. The velocity accumulator is also integrated by adding the top 19 bits of it to a distance accumulator to create a 31 bit value for the distance traveled. The FPGA

cannot handle more than 32 bits at a time so we had to sacrifice the least significant bits of the velocity for the position integration. The error introduced is minimal because 31 bits is still a lot of information for a distance value.

When a user presses the 0-60 performance test button, the performance tester zeroes the counter and begins waiting for the acceleration to rise above 0.2 g's. When acceleration commences, the velocity integrator is reset and the counter is started. When the velocity integrator reaches a value in count-milliseconds that corresponds to 60 miles per hour or about 2.7 g-seconds, the counter is stopped and the test is finished.

When a user presses the 1/4 mile performance button, the same procedure as the 0-60 test is executed except this time the distance integrator is also used and the test finishes when the distance reaches a certain count.

For the braking test, the user needs to bring the vehicle up to the initial speed and press the button. Once the user begins braking, the test automatically begins when braking exceeds 0.1 g's and it continues until braking stops, corresponding to the car being brought to a complete stop. This provides the user with the time taken to perform the brake test.

The free run test functions as just a pushbutton timer for the user. It restarts every time the user presses the button.

Testing

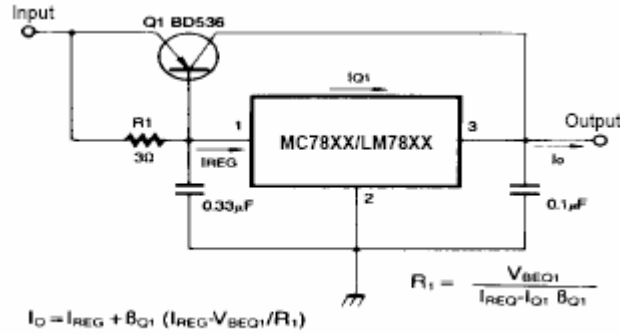
The system was tested in the laboratory before installing it in the test vehicles. It was connected to a brake-lamp spoiler using an analog inverse relay circuit made out of two resistors, a BJT, and a MOSFET. The light would turn off whenever the signal from the pulser went high. To simulate acceleration, the accelerometer was tilted manually. The performance tests provided consistent results and times that corresponded to actual numbers. For example, if the accelerometer was tilted 90 degrees, that simulated 1g of acceleration and 0-30 happened in 1.4 seconds. The wireless transmission was also demonstrated in lab by the receiving system playing a warning sound every time the sending system simulated hard braking and flashed its lights.

Once the system performance was verified in lab, it was installed in a vehicle's DIN slot and powered by an available 12-volt wire there. A wire was run back to the brake lamp and the relay was installed in a manner that does not interfere with normal operation of the brake lamp when the system is not running. The various tests were run under different conditions and they all worked. For example, the 0-30 test button was pressed at 15 mph and the test was initiated by acceleration. The car did not reach 30 mph and had to stop at a light. After turning right, the car accelerated to 45 mph, 30 more than the initial 15, and the test stopped. This demonstrates that the velocity integrator can account for negative acceleration and remain accurate for up to around a minute. Most performance tests are complete within 15 seconds so integrator drift is not an issue.

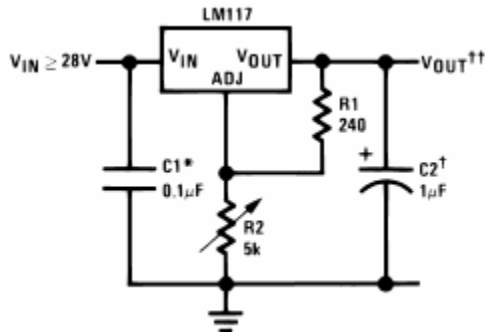
Conclusion

The DEBRA system was completed and verified successfully. For mass production, a method for calibrating different ADXL202 chips needs to be developed. The performance measurement part of the system will ideally be a low cost alternative to the G-Tech system already on the market. It mostly serves as just an added incentive for drivers to add the system as a safety modification. However, it is possible to add more features and improve the system to be a direct competitor to the G-Tech. For the brake lamp pulsing part, adapters for different brake lamp systems will need to be provided.

Appendix



5v Power Supply using LM7805



3.3v Power Supply using Lm317

```
//AM28F512 PROM data for LCD controller
//
#SET_ADDRESS=0000;
#MASK_COUNT=8;
#BASE=HEX;
```

```
FE FE FE FE FE FE FE FE FE FE FE FE 00 00 00 00
38 0C 01 80 30 31 32 33 34 35 36 37 38 39 3A 2E
53 79 73 74 65 6D FE 52 65 61 64 79 FE FE FE FE
FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FF 50 72 65 73 73 FE 61 FE
62 75 74 74 6F 6E FE 74 6F FE 73 74 61 72 74 FE
30 2D 36 30 FE 41 63 63 65 6C 65 72 61 74 69 6F
6E FE 74 69 6D 65 FE 74 72 69 61 6C FE FE FE FE
FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00
42 72 61 6B 69 6E 67 FE 74 69 6D 65 FE 74 72 69
61 6C FE FE FE FE FE FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00
51 75 61 72 74 65 72 FE 6D 69 6C 65 FE 74 69 6D
65 FE 74 72 69 61 6C FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00
46 72 65 65 FE 72 75 6E 6E 69 6E 67 FE 74 69 6D
65 FE 74 72 69 61 6C FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00
53 74 61 72 74 FE 41 63 63 65 6C 65 72 61 74 69
6F 6E FE 74 6F FE 73 74 61 72 74 FE 74 72 69 61
6C FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00
53 74 61 72 74 FE 44 65 63 65 6C 65 72 61 74 69
```

```

6F 6E FE 74 6F FE 73 74 61 72 74 FE 74 72 69 61
6C FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00
54 72 69 61 6C FE 69 6E FE 70 72 6F 67 72 65 73
73 FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00
54 72 69 61 6C FE 63 6F 6D 70 6C 65 74 65 FE FE
FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00
50 72 65 73 73 FE 61 FE 62 75 74 74 6F 6E FE 74
6F FE 73 74 61 72 74 FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE 00 00 00 00 00 00 00 00

```

Verilog Code

```

module pulser (clk, notslowingdown, outtobrakelamp);

input clk, notslowingdown;
output outtobrakelamp;
reg pulse;
reg [19:0] count;
reg [23:0] hold;
reg brakehold;

always @(posedge clk)
begin
count = count + 1;
if (count == 20'b0)
pulse = ~pulse;

if (!notslowingdown)
hold[23:0] <= 24'd10000000;

if (hold == 0) brakehold <= 0; else
begin
brakehold <= 1;
hold <= hold - 1;
end

end

assign outtobrakelamp = (brakehold && pulse);
endmodule //this is cool

//Input synchronizer. Charvak Karpe.

module synchronizer (clk, reset_raw, reset, thirty_raw, thirty,
sixty_raw, sixty, quarter_raw, quarter, xin_raw, xin,
yin_raw, yin);

input clk, reset_raw, thirty_raw, sixty_raw, quarter_raw, xin_raw,
yin_raw;

output reset, thirty, sixty, quarter, xin, yin;
reg reset, thirty, sixty, quarter, xin, yin;

```



```

reg reset_int, thirty_int, sixty_int, quarter_int, xin_int, yin_int;

reg yin2, ycheck;
reg [7:0] ycount;

always @ (posedge clk)
begin
    reset_int <= reset_raw;
    thirty_int <= thirty_raw;
    sixty_int <= sixty_raw;
    quarter_int <= quarter_raw;
    xin_int <= xin_raw;
    yin_int <= yin_raw;
    reset <= reset_int;
    thirty <= thirty_int;
    sixty <= sixty_int;
    quarter <= quarter_int;
    xin <= xin_int;
//    yin <= yin_int;
    yin2 <= yin_int;

    if (yin2 && ycheck && (ycount == 6'b0))
    begin
        yin <= 1;
        ycount <= 8'd255;
        ycheck <= 0;
    end

    if (!yin2 && !ycheck && (ycount == 6'b0))
    begin
        yin <= 0;
        ycount <= 8'd255;
        ycheck <= 1;
    end

    if (!(ycount == 8'b0))
        ycount <= ycount - 1;

end
endmodule

module top (clk, reset_raw, x, y, braking_raw, sixty_raw, quarter_raw,
freerun, outtobrakelamp, notslowingdown, en, rs, databus, addr, go);//,
display);

input clk, reset_raw, x, y, braking_raw, sixty_raw, quarter_raw,
freerun;
input go;

output outtobrakelamp, notslowingdown;
output en, rs;
output [8:0] databus;

```

```

output [14:0] addr;

wire reset, braking, sixty, quarter, xin, yin, notslowingdown;
wire [11:0] accel;
wire start, stop, zero;
wire [3:0] num3, num2, num1;
wire slowingdown;
wire [1:0] status;
wire [2:0] mode;

synchronizer synch1 (.clk(clk), .reset_raw(reset_raw), .reset(reset),
    .thirty_raw(braking_raw), .thirty(braking),
        .sixty_raw(sixty_raw), .sixty(sixty),
        .quarter_raw(quarter_raw), .quarter(quarter),
        .xin_raw(x), .xin(xin), .yin_raw(y), .yin(yin));

transform transform1 (.clk(clk), .reset(reset), .xin(xin), .yin(yin),
    .accel(accel),
        .notslowingdown(notslowingdown), .ycheck(ycheck),
        .y(yout), .ycount(ycount));

pulser pulser1 (.clk(clk), .notslowingdown(notslowingdown),
    .outtobrakelamp(outtobrakelamp));

perf perf1 (.clk(clk), .reset(reset), .braking(braking), .sixty(sixty),
    .quarter(quarter), .freerun(freerun), .accel(accel), .zero(zero),
        .start(start), .stop(stop), .status(status), .mode(mode));

counter counter1 (.clk(clk), .zero(zero), .start(start), .stop(stop),
    .num3(num3), .num2(num2), .num1(num1));

lcd lcd1 (.clk(clk), .reset(reset), .en(en), .rs(rs),
    .databus(databus), .mode(mode), .status(status), .num3(num3),
    .num2(num2), .num1(num1));

sound sound1 (.clk(clk), .go(go), .addr(addr), .reset(reset));

endmodule

module transform (clk, reset, xin, yin,
    //fsquared,
        braking, slowingdown, xquotient, yquotient, xh13, xtot,
    yh13, ytot,
        startdivx, startdivy, ycheck, xtest, ytest, y, ycount,
    ycount_high);

input clk, reset, xin, yin;
output slowingdown, startdivx, startdivy, ycheck;
input [31:0] xquotient, yquotient;
output [31:0] xh13, xtot, yh13, ytot;
output [11:0] braking;
output [3:0] xtest, ytest;

output [11:0] y;
output [14:0] ycount, ycount_high;

```

```

reg startdivx, startdivy, init, xcheck, ycheck;

reg [14:0] xcount, ycount, xcount_high, ycount_high;
reg [19:0] init_counter;
reg [5:0] divcountx, divcounty;
reg [11:0] x, xr, yr;
reg [31:0] xh13, xtot, yh13, ytot, yintermediate;

reg slowingdown;
reg [21:0] xr22, yr22, x22, y22, braking10;
reg [11:0] braking, brakingthreshold, y;//fsquared
//wire [22:0] fsquared10;
//wire [10:0] xmag, ymag;

always @(posedge clk)
begin
    if (reset)
        begin
            xcount <= 15'b0; //fifteen bits allows count up to 32767
            ycount <= 15'b0;
            xcount_high <= 15'b0;
            ycount_high <= 15'b0;
            init_counter <= 20'b1; //provides 0.1 second delay before setting
reset values
            init <= 1;
            divcountx <= 0;
            divcounty <= 0;
            xcheck <= 0;
            ycheck <= 0;
        end
    else
begin
//-----
if (startdivx)
    begin
        startdivx <= 0;
        divcountx <= 6'd35;
    end

if (!(divcountx == 0))
    divcountx <= divcountx - 1;

if (divcountx == 6'd1)
    begin
        x[11:0] <= xquotient[31:0] - 14'd4342;//1761;
    end
//-----
//-----
if (startdivy)
    begin
        startdivy <= 0;
        divcounty <= 6'd35;
    end

if (!(divcounty == 0))
    divcounty <= divcounty - 1;
end

```

```

if (divcounty == 6'd1)
  begin
    yintermediate[31:0] <= yquotient[31:0] - 14'd4342;//3867;
  end
//-----

//-----
if (init)
  begin
    if (init_counter == 20'b0)
      begin
        xr <= x;
        yr <= y;
        init <= 0;
      end
    else
      init_counter <= init_counter + 1;
    end
  end
//-----

//-----
if (xin && xcheck)
  begin
    xtot[31:0] <= {17'b0,xcount};
    xh13[31:0] <= {4'b0,xcount_high,13'b0};
    xcheck <= 0;
    xcount <= 15'b0;
    xcount_high <= 15'b0;
    startdivx <= 1;
  end
  else
    begin
      xcount <= xcount + 1;
      if (!xin && !xcheck)
        xcheck <= 1;
      if (xin && !xcheck)
        xcount_high <= xcount_high + 1;
    end
//-----
//-----
if (yin && ycheck)
  begin
    ytot[31:0] <= {17'b0,ycount[14:0]};
    yh13[31:0] <= {4'b0,ycount_high[14:0],13'b0};
    ycheck <= 0;
    ycount <= 15'b0;
    ycount_high <= 15'b0;
    startdivy <= 1;
  end
  else
    begin
      ycount <= ycount + 1;
      if (!yin && !ycheck)
        ycheck <= 1;
      if (yin && !ycheck)
        ycount_high <= ycount_high + 1;
    end

```

```

        end
//-----

end
y[11:0] <= yintermediate[11:0];

xr22[21:0] <= xr[11] ? {10'd1023,xr} : {10'd0,xr};
yr22[21:0] <= yr[11] ? {10'd1023,yr} : {10'd0,yr};
x22[21:0] <= x[11] ? {10'd1023,x} : {10'd0,x};
y22[21:0] <= y[11] ? {10'd1023,y} : {10'd0,y};
braking10 <= x22*yr22 - y22*xr22; //used to be '+'
braking[11:0] <= braking10[21:10];

brakingthreshold[11:0] <= braking[11:0] - 12'd400;
slowingdown <= ~brakingthreshold[11];

end

assign xtest[3:0] = x[9:6]; //testing
assign ytest[3:0] = y[9:6]; //testing
//assign xmag[10:0] = x[11] ? (~x + 1) : x;
//assign ymag[10:0] = y[11] ? (~y + 1) : y;
//assign fsquared10[22:0] = xmag*xmag + ymag*ymag - {1'b1,20'b0};
//assign fsquared[11:0] = fsquared10[21:10];

endmodule

module transform (clk, reset, xin, yin, accel, notslowingdown, ycheck,
y, ycount);

input clk, reset, xin, yin;
output notslowingdown, ycheck;
output [11:0] accel;

output [11:0] y;
output [14:0] ycount;

reg init, xcheck, ycheck;
reg [14:0] xcount, ycount;
reg [19:0] init_counter;
reg [11:0] x, y, xr, yr;

reg notslowingdown;
reg [21:0] xr22, yr22, x22, y22, braking10;
reg [11:0] braking, brakingthreshold, accel;

reg [19:0] xquotient6, yquotient6;
reg [13:0] xquotient, xintermediate, yquotient, yintermediate;

always @(posedge clk)
begin
    if (reset)
        begin

```

```

    xcount <= 15'b0; //fifteen bits allows count up to 32767
    ycount <= 15'b0;
    init_counter <= 20'b1; //provides 0.1 second delay before setting
reset values
    init <= 1;
    xcheck <= 0;
    ycheck <= 0;
    end
    else
begin
//-----
if (init)
begin
if (init_counter == 20'b0)
begin
xr <= x;
yr <= y;
init <= 0;
end
else
init_counter <= init_counter + 1;
end
//-----

//-----
if (xin && xcheck)
begin
xcheck <= 0;
xcount <= 15'b0;
end
else
begin
if (!xin && !xcheck)
begin
xcheck <= 1;
xquotient6[19:0] <= (xcount * 5'd31);
end
if (xin && !xcheck)
xcount <= xcount + 1;
end
//-----
//-----
if (yin && ycheck)
begin
ycheck <= 0;
ycount <= 15'b0;
end
else
begin
if (!yin && !ycheck)
begin
ycheck <= 1;
yquotient6[19:0] <= (ycount * 5'd31);
end
if (yin && !ycheck)
ycount <= ycount + 1;
end

```

```

        end
//-----

end
xquotient[13:0] <= xquotient6[19:6];
xintermediate[13:0] <= xquotient[13:0] - 14'd4342;
x[11:0] <= xintermediate[11:0];

yquotient[13:0] <= yquotient6[19:6];
yintermediate[13:0] <= yquotient[13:0] - 14'd4342;
y[11:0] <= yintermediate[11:0];

xr22[21:0] <= xr[11] ? {10'd1023,xr} : {10'd0,xr};
yr22[21:0] <= yr[11] ? {10'd1023,yr} : {10'd0,yr};
x22[21:0] <= x[11] ? {10'd1023,x} : {10'd0,x};
y22[21:0] <= y[11] ? {10'd1023,y} : {10'd0,y};
braking10 <= x22*yr22 - y22*xr22; //used to be '+'
braking[11:0] <= braking10[21:10];

brakingthreshold[11:0] <= braking[11:0] - 12'd300;
notslowingdown <= brakingthreshold[11];

accel[11:0] <= 12'b0 - braking[11:0];
end
endmodule

//DEBRA performance test module. Charvak Karpe. 5/12/04.

module perf (clk, reset, braking, sixty, quarter, freerun, accel, zero,
start, stop, status, mode);

input clk, reset, braking, sixty, quarter, freerun;
input [11:0] accel;
output zero, start, stop;
output [1:0] status;
output [2:0] mode;
reg zero, start, stop;

reg [23:0] vacc, checkbraking, checksixty;
reg [30:0] dacc, checkquarter;
reg startbrakingint, startbraking, runbraking, startsixtyint,
startsixty, runsixty,
startquarterint, startquarter, runquarter, running;

reg [1:0] status;
reg [2:0] mode;

reg freerundelay;

reg [13:0] count;

wire [23:0] accelext;
wire [30:0] vaccext;
wire [11:0] accthreshold;
wire [11:0] decel, decelthreshold;

```

```

wire brakin;
wire accelerating;

always @ (posedge clk)
begin

    if (reset)
    begin
    mode <= 0;
    zero <= 1;
    start <= 0;
    stop <= 0;
    end

    if (braking)
    begin
    mode <= 2;
    status <= 0;
    zero <= 1;
    startbrakingint <= 1;
    vacc <= 24'b0;
    end
else
    if (startbrakingint)
    begin
    zero <= 0;
    if (brakin)
    begin
    startbrakingint <= 0;
    startbraking <= 1;
    end
    end

    if (startbraking)
    begin
    status <= 2;
    startbraking <= 0;
    start <= 1;
    runbraking <= 1;
    end

    if (runbraking)
    begin
    if (!brakin) //stop the test
    begin
    status <= 3;
    runbraking <= 0;
    stop <= 1;
    end
    end

    if (sixty)
    begin
    zero <= 1;
    startsixtyint <= 1;
    vacc <= 24'b0;
    mode <= 1;
    status <= 0;
    end
end

```



```

else
  if (startsixtyint)
    begin
      zero <= 0;
      if (accelerating)
        begin
          startsixtyint <= 0;
          startsixty <= 1;
        end
      end
    if (startsixty)
      begin
        status <= 2;
        startsixty <= 0;
        start <= 1;
        runsixty <= 1;
      end
    if (runsixty)
      begin
        checksixty[23:0] <= vacc[23:0] - 24'd2773089;//subtract 60mph
        if (checksixty[23] == 0) //stop the
test
      begin
        status <= 3;
        runsixty <= 0;
        stop <= 1;
      end
    end

    if (quarter)
      begin
        mode <= 3;
        status <= 0;
        zero <= 1;
        startquarterint <= 1;
        vacc <= 24'b0;
        dacc <= 31'b0;
      end
    else
      if (startquarterint)
        begin
          zero <= 0;
          if (accelerating)
            begin
              startquarterint <= 0;
              startquarter <= 1;
            end
          end
        if (startquarter)
          begin
            status <= 2;
            startquarter <= 0;
            start <= 1;
            runquarter <= 1;
          end
        if (runquarter)
          begin

```

```

    checkquarter[30:0] <= dacc[30:0] - 31'd1312859334;//subtract 1/4
mile
    if (dacc[30:0] > 31'd1312859334)
        begin
            status <= 3;
            runquarter <= 0;
            stop <= 1;
        end
    end

    if (freerun)
        begin
            mode <= 4;
            status <= 2;
            zero <= 1;
            freerundelay <= 1;
        end
    else
        if (freerundelay)
            begin
                freerundelay <= 0;
                start <= 1;
            end

        if (start)                begin
            start <= 0;
            running <= 1;
        end

        if (stop)                begin
            stop <= 0;
            running <= 0;
        end

        if (running)
            begin
                if (count == 14'd10000)
                    begin
                        count <= 14'b1;
                        vacc[23:0] <= vacc[23:0] + accelext[23:0];
                        dacc[30:0] <= dacc[30:0] + vaccext[30:0];
                    end
                else
                    count <= count + 1;
            end
        end

    end

assign accelext[23:0] = (accel[11]) ? {12'hfff,accel[11:0]} :
{12'b0,accel[11:0]};
assign vaccext[30:0] = (vacc[23]) ? {12'hfff,vacc[23:5]} :
{12'b0,vacc[23:5]};
assign accthreshold[11:0] = accel[11:0] - 12'd200;
assign accelerating = ~accthreshold[11];

assign decel = 12'b0 - accel;

```

```
assign decelthreshold = decel - 12'd100;
assign braklin = ~decelthreshold[11];
endmodule
```

```
//Counter. Charvak Karpe. 5/11/04
```

```
module counter (clk, zero, start, stop, num3, num2, num1);
```

```
input clk, zero, start, stop;
output [3:0] num3, num2, num1;
reg [3:0] num3, num2, num1;
reg pulsenum1, pulsenum2, pulsenum3;
reg running;
reg [20:0] count;
```

```
always @(posedge clk)
begin
if (count == 21'd1000000)
begin
pulsenum1 <= 1;
count <= 1;
end
else count <= count + 1;
```

```
if (zero)
begin
num3 <= 0;
num2 <= 0;
num1 <= 0;
running <= 0;
//actualcount<=0;
end
```

```
if (start)
begin
count <= 1;
running <= 1;
end
```

```
if (stop)
running <= 0;
```

```
if (pulsenum1)
begin
pulsenum1 <= 0;
end
```

```
if (pulsenum2) pulsenum2 <= 0;
if (pulsenum3) pulsenum3 <= 0;
```

```
if (running)
begin
if (pulsenum1)
begin
```



```
    quotient_temp = quotient_temp << 1;

    if( !diff[63] )
    begin
        dividend_copy = diff;
        quotient_temp[0] = 1;
    end

    quotient = quotient_temp;

    divider_copy = divider_copy >> 1;
    bit = bit - 1;

end
end
endmodule
```