US 20030114075A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2003/0114075 A1**

Moll et al. (43) **Pub. Date: Jun. 19, 2003**

(54) **TOY VEHICLE WIRELESS CONTROL SYSTEM**

(76) Inventors: **Joseph T. Moll**, Prospect Park, PA (US); **James M. Dickinson**, Haddon Township, NJ (US); **Frank W. Winkler**, Mickleton, NJ (US); **David V. Helmlinger**, Mt. Laurel, NJ (US); **Charles S. McCall**, San Francisco, CA (US); **Stephen N. Weiss**, Philadelphia, PA (US)

Correspondence Address:
AKIN GUMP STRAUSS HAUER & FELD L.L.P.
ONE COMMERCE SQUARE
2005 MARKET STREET, SUITE 2200
PHILADELPHIA, PA 19103-7013 (US)

(21) Appl. No.: **10/284,046**

(22) Filed: **Oct. 30, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/340,591, filed on Oct. 30, 2001.

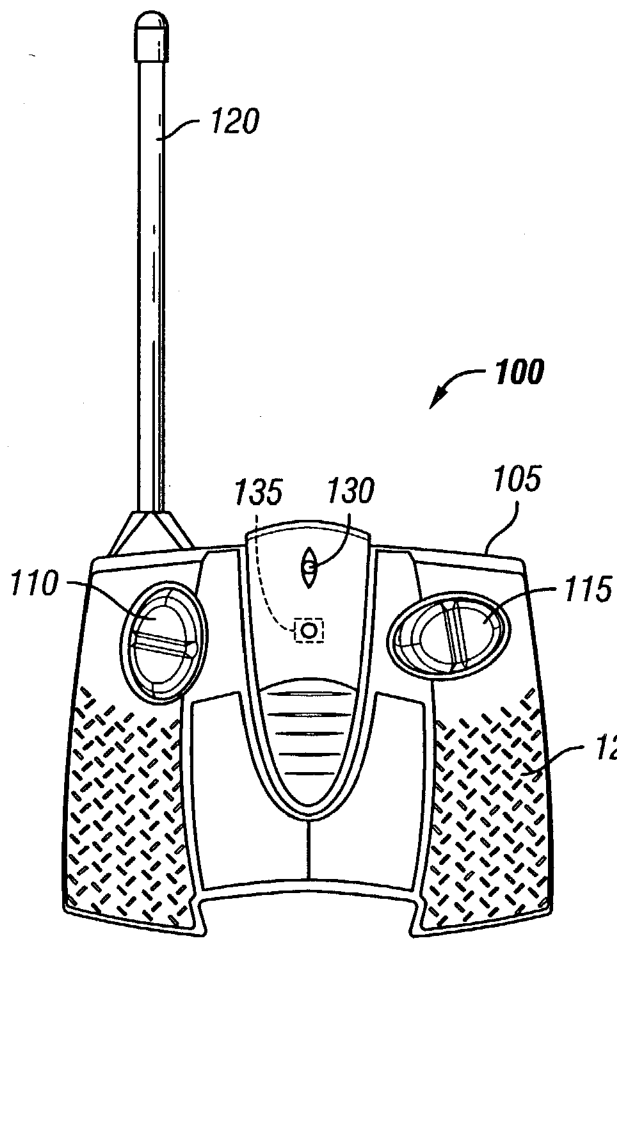
Publication Classification

(51) **Int. Cl.**^{7} **A63H 30/04**

(52) **U.S. Cl.** **446/456**

(57) **ABSTRACT**

A toy vehicle remote control transmitter unit wirelessly controls the movements of a programmable toy vehicle. The toy vehicle includes a motive chassis having a plurality of steering positions. A microprocessor in the transmitter unit emulates manual transmission operation of the toy vehicle by being in any one of a plurality of different gear states selected by an operation of manual input elements on the transmitter unit. Forward propulsion control signals representing different toy vehicle speed ratios associated with each of the gear states are transmitted from the transmitter unit to the toy vehicle. The motive chassis has a steering feedback sensor with a plurality of defined steering positions to vary rate of steering position change to avoid overshoot.



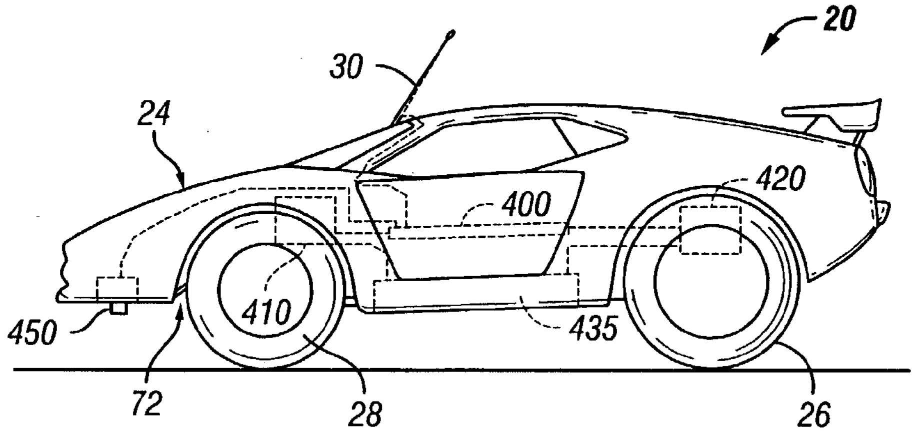

20
30
24
400
420
450
410
435
72
28
26

FIG. 1B

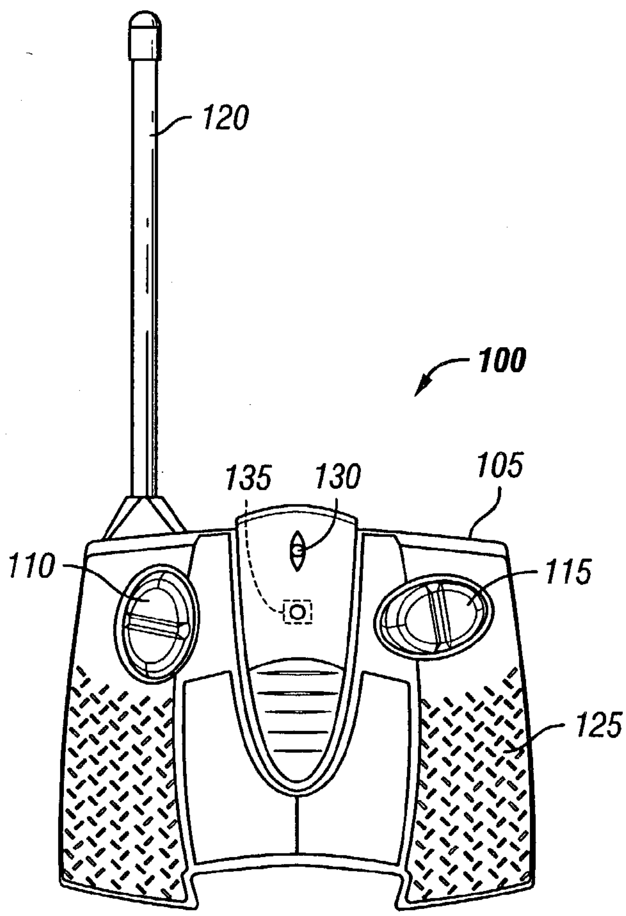

120
100
135
130
105
110
115
125

FIG. 1A

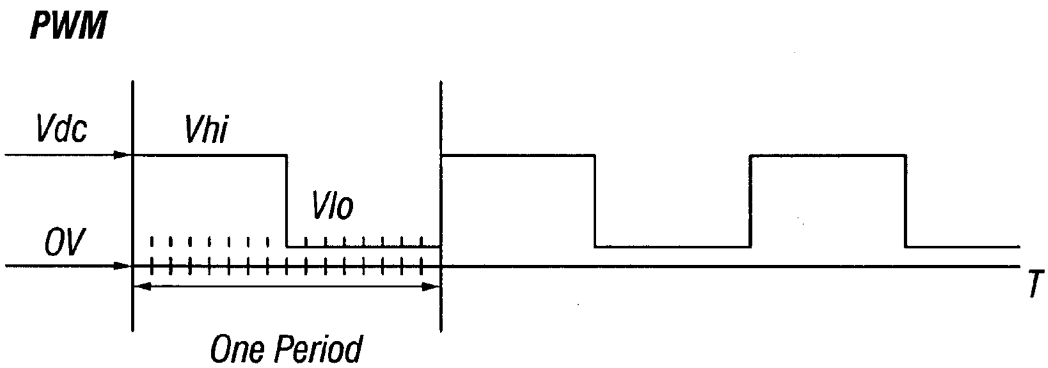


FIG.2

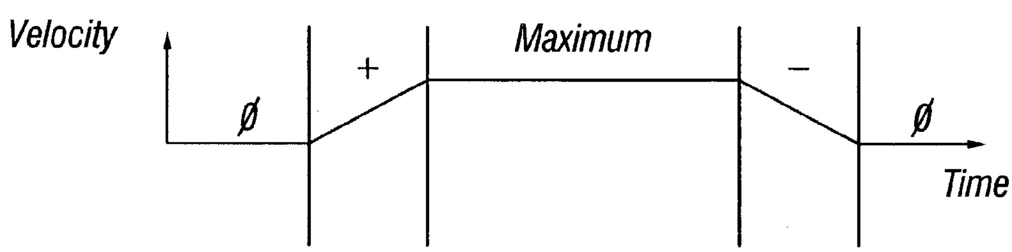


FIG.3

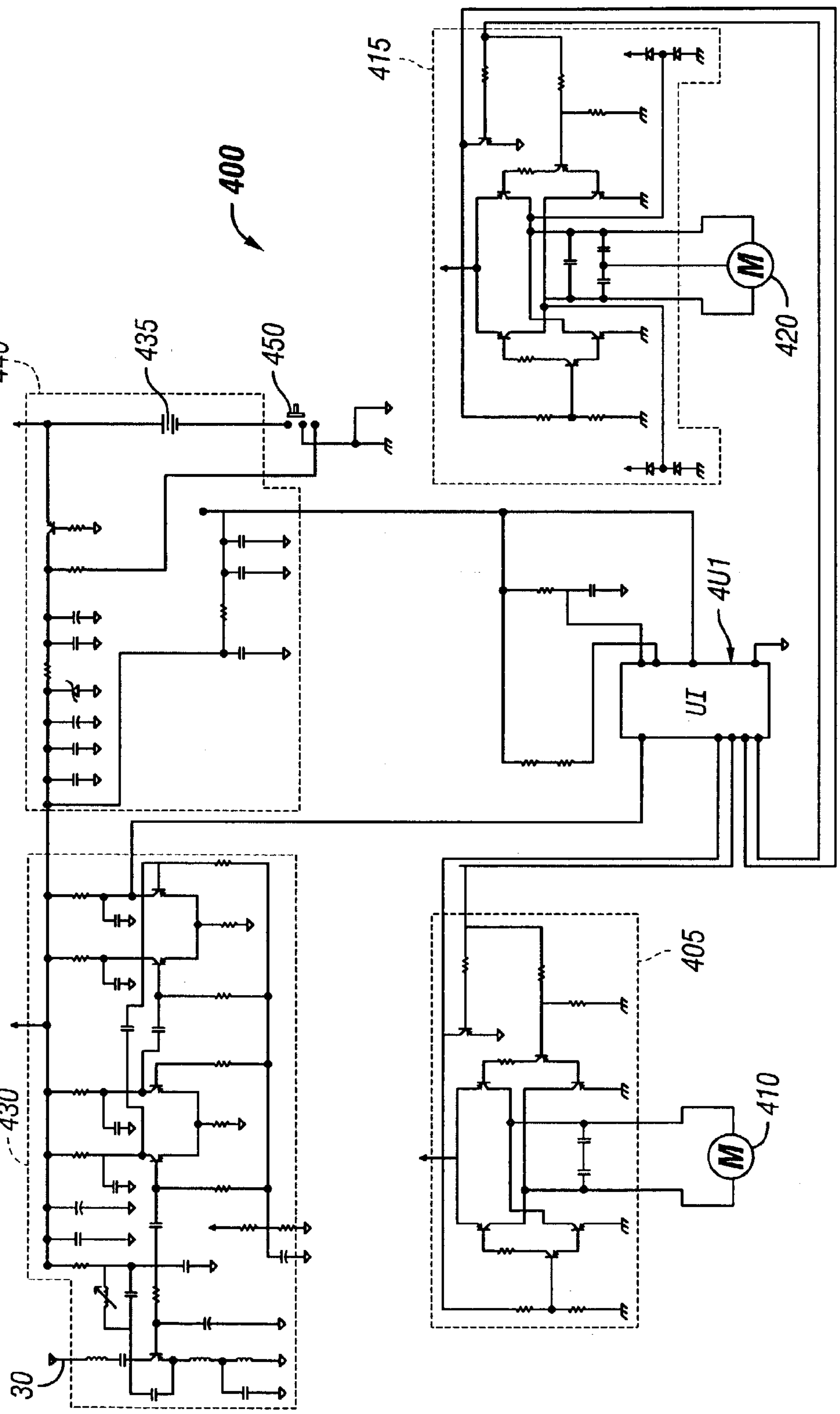


FIG.4

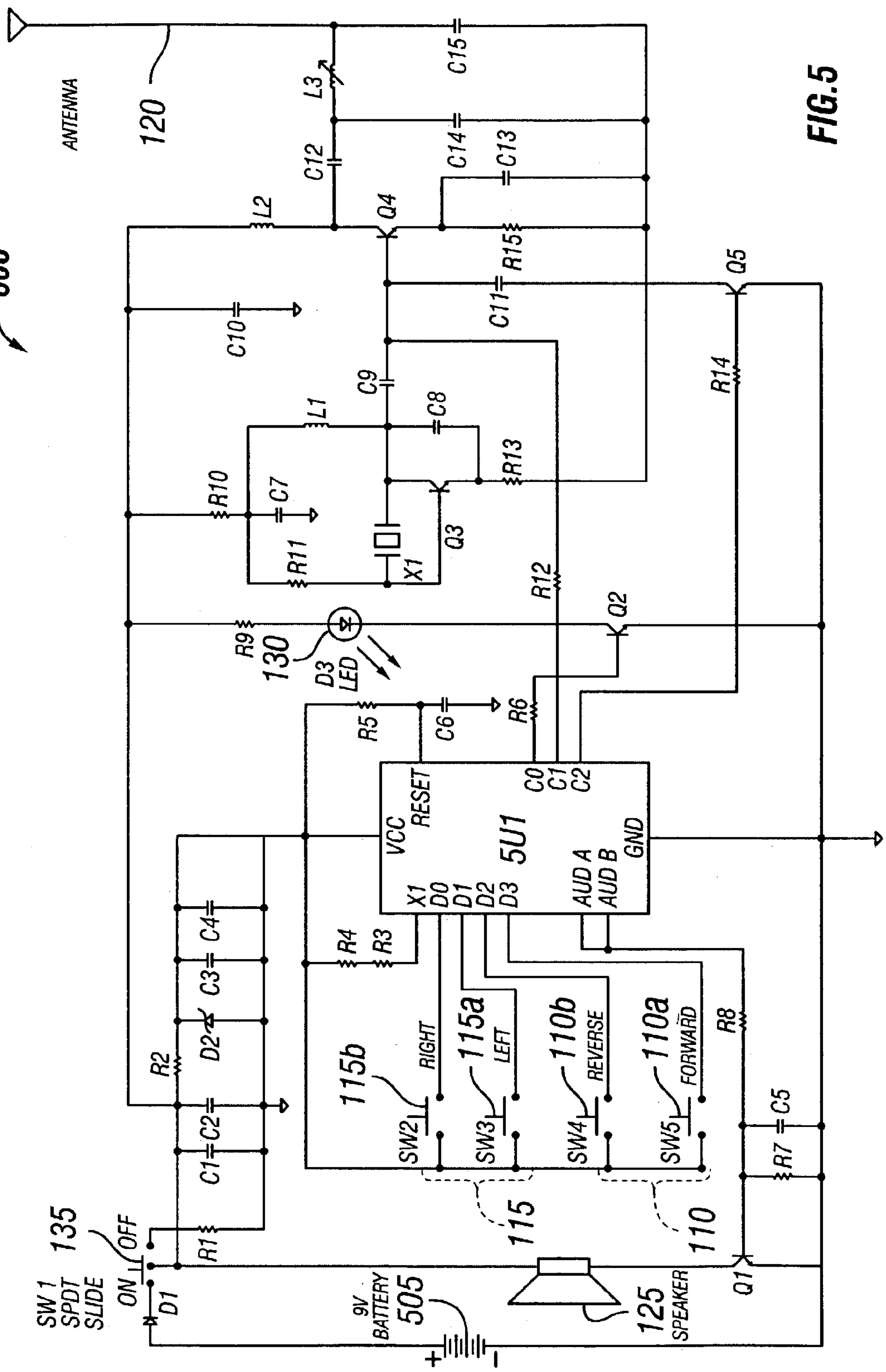

500
ANTENNA
120
SW1 SPDT SLIDE
135
ON
OFF
D1
9V BATTERY
505
125
SPEAKER
5U1
VCC
RESET
X1
D0
D1
D2
D3
AUD A
AUD B
GND
C0
C1
C2
115b
RIGHT
115a
LEFT
110b
REVERSE
110a
FORWARD
115
110
D3 LED
130

FIG.5

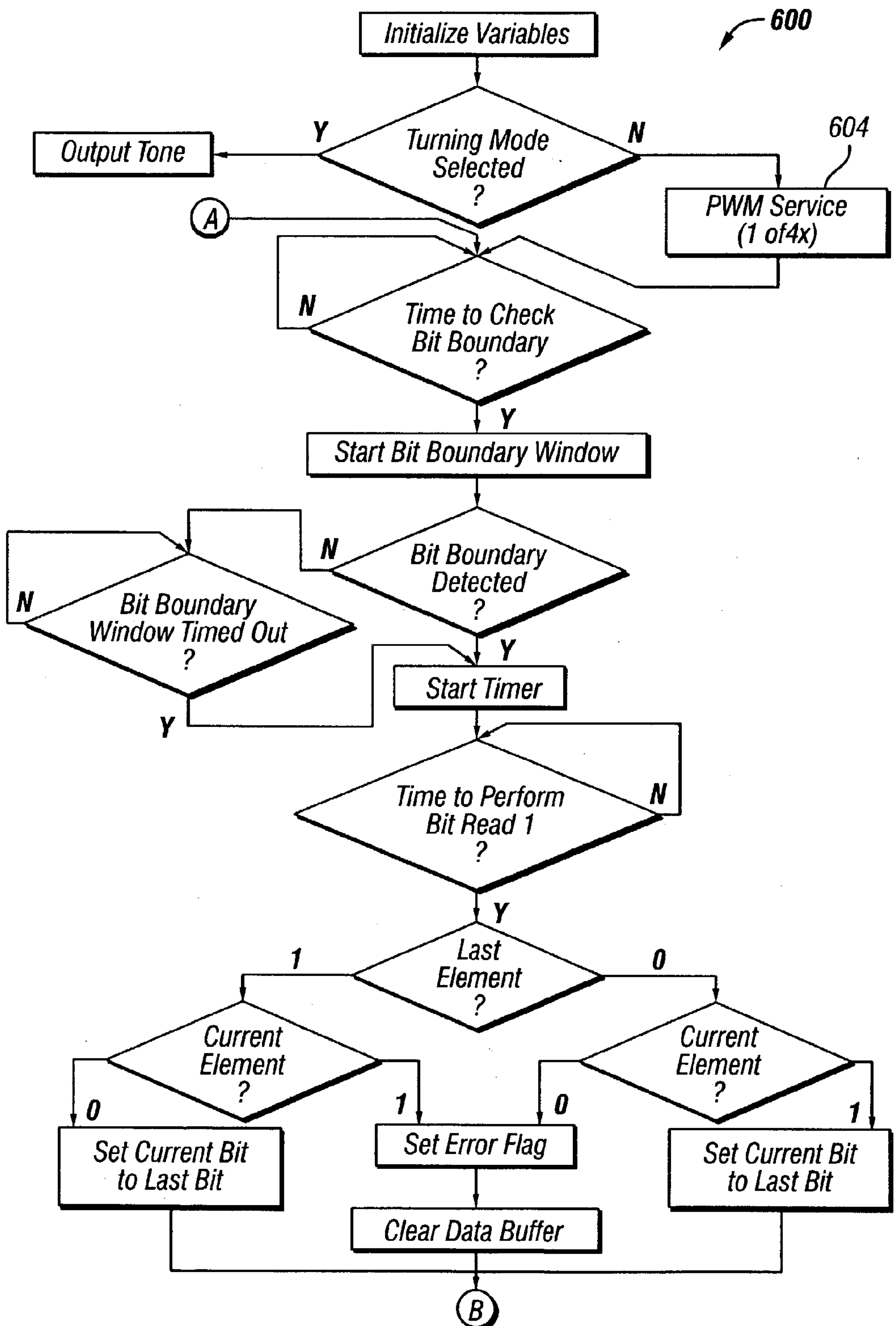

600
Initialize Variables
Turning Mode Selected ?
Y
N
Output Tone
604
PWM Service (1 of4x)
A
N
Time to Check Bit Boundary ?
Y
Start Bit Boundary Window
N
Bit Boundary Detected ?
N
Bit Boundary Window Timed Out ?
Y
Y
Start Timer
Time to Perform Bit Read 1 ?
N
Y
Last Element ?
1
0
Current Element ?
Current Element ?
0
1
0
1
Set Current Bit to Last Bit
Set Error Flag
Set Current Bit to Last Bit
Clear Data Buffer
B

FIG.6A

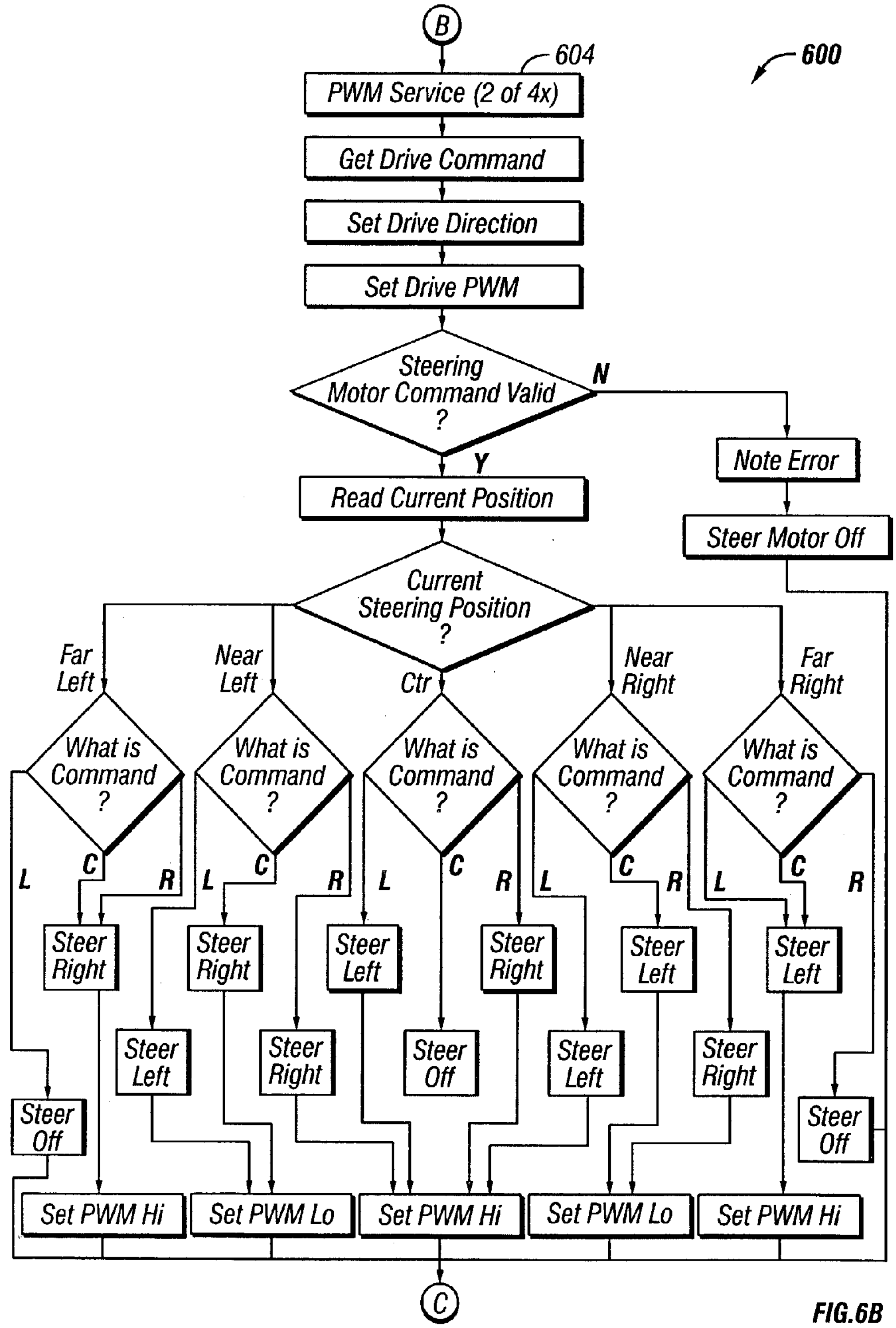

B
604
600
PWM Service (2 of 4x)
Get Drive Command
Set Drive Direction
Set Drive PWM
Steering Motor Command Valid ?
N
Y
Note Error
Steer Motor Off
Read Current Position
Current Steering Position ?
Far Left
Near Left
Ctr
Near Right
Far Right
What is Command ?
L
C
R
Steer Right
Steer Left
Steer Off
Set PWM Hi
Set PWM Lo
C

FIG.6B

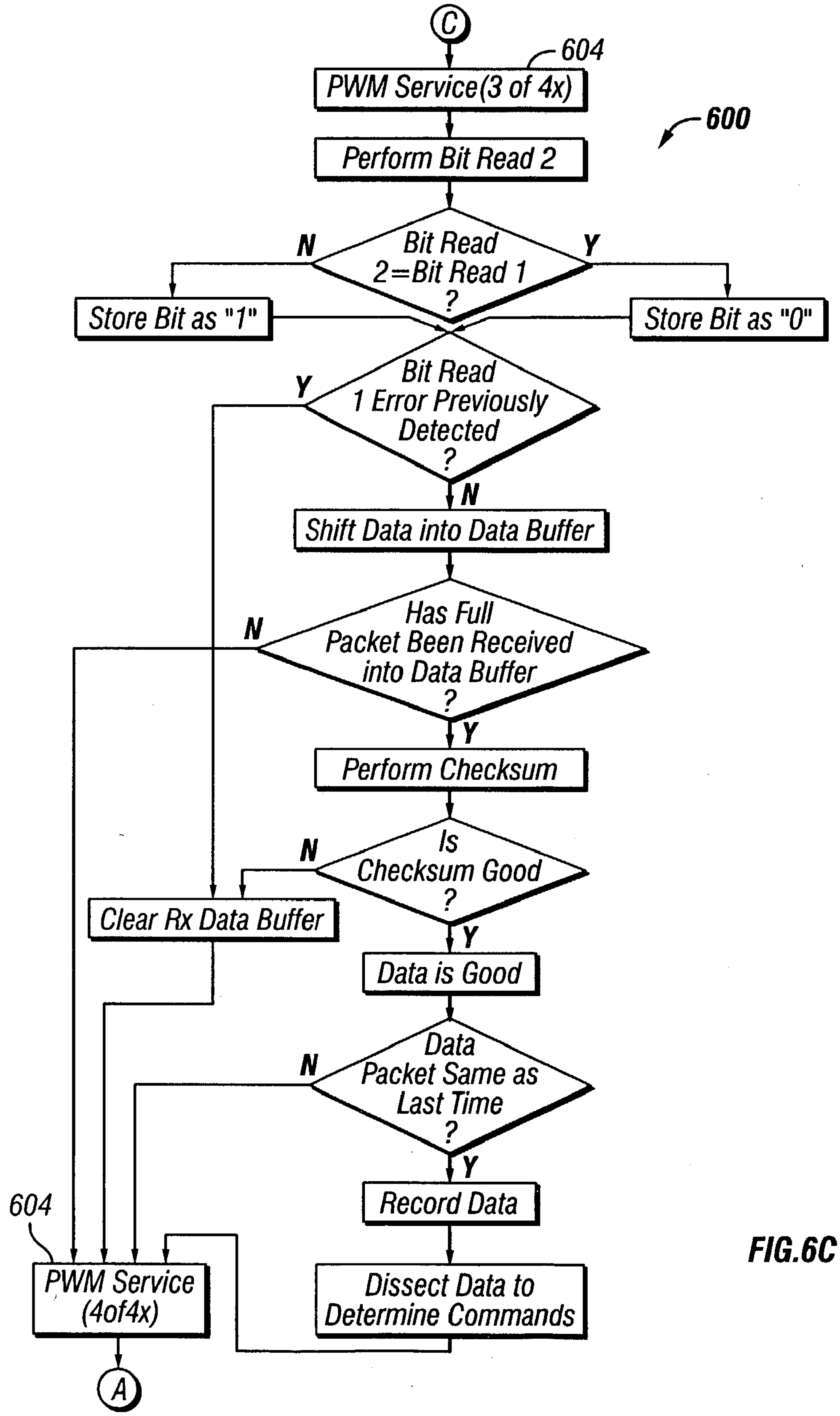

C
604
PWM Service(3 of 4x)
600
Perform Bit Read 2
Bit Read 2=Bit Read 1 ?
N
Y
Store Bit as "1"
Store Bit as "0"
Bit Read 1 Error Previously Detected ?
Y
N
Shift Data into Data Buffer
Has Full Packet Been Received into Data Buffer ?
N
Y
Perform Checksum
Is Checksum Good ?
N
Y
Clear Rx Data Buffer
Data is Good
Data Packet Same as Last Time ?
N
Y
Record Data
604
PWM Service (4of4x)
Dissect Data to Determine Commands
A

FIG.6C

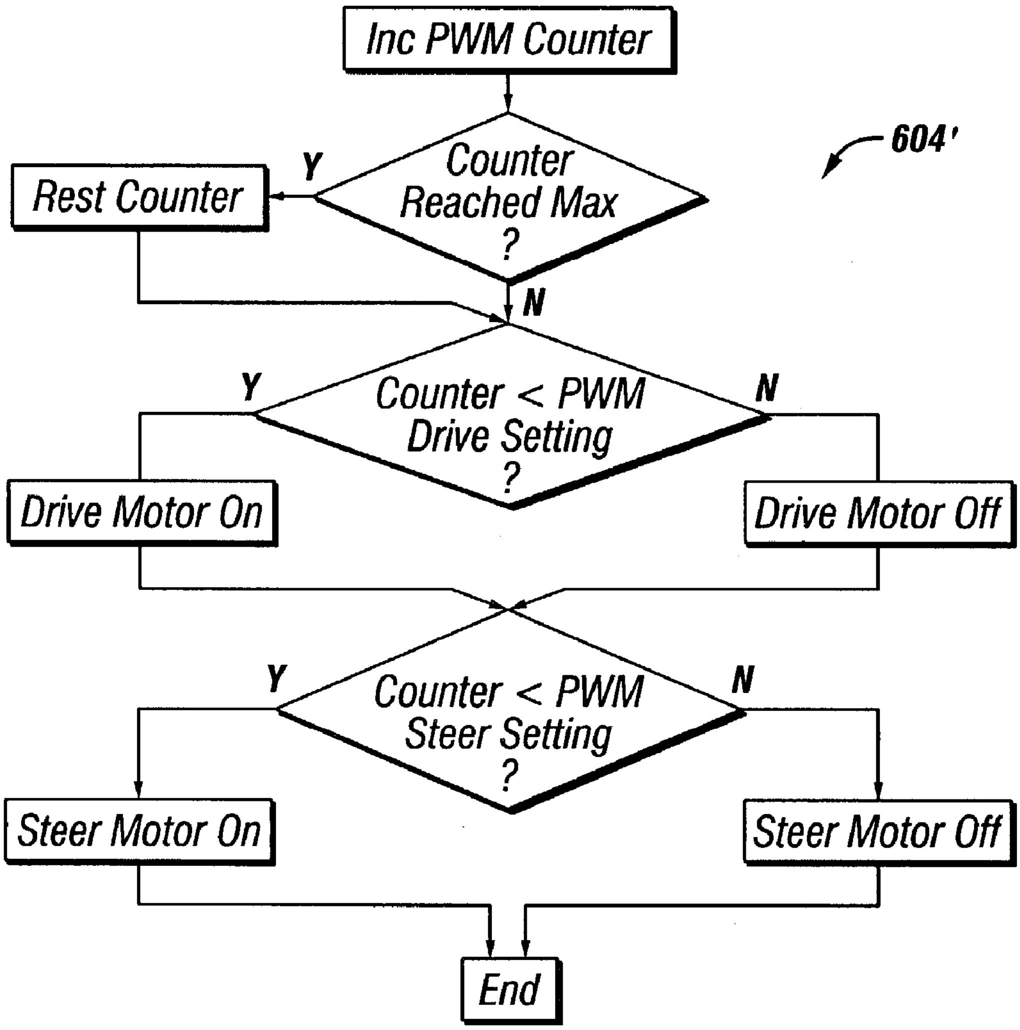

604'
Inc PWM Counter
Counter Reached Max ?
Y
N
Rest Counter
Counter < PWM Drive Setting ?
Y
N
Drive Motor On
Drive Motor Off
Counter < PWM Steer Setting ?
Y
N
Steer Motor On
Steer Motor Off
End

FIG.6D

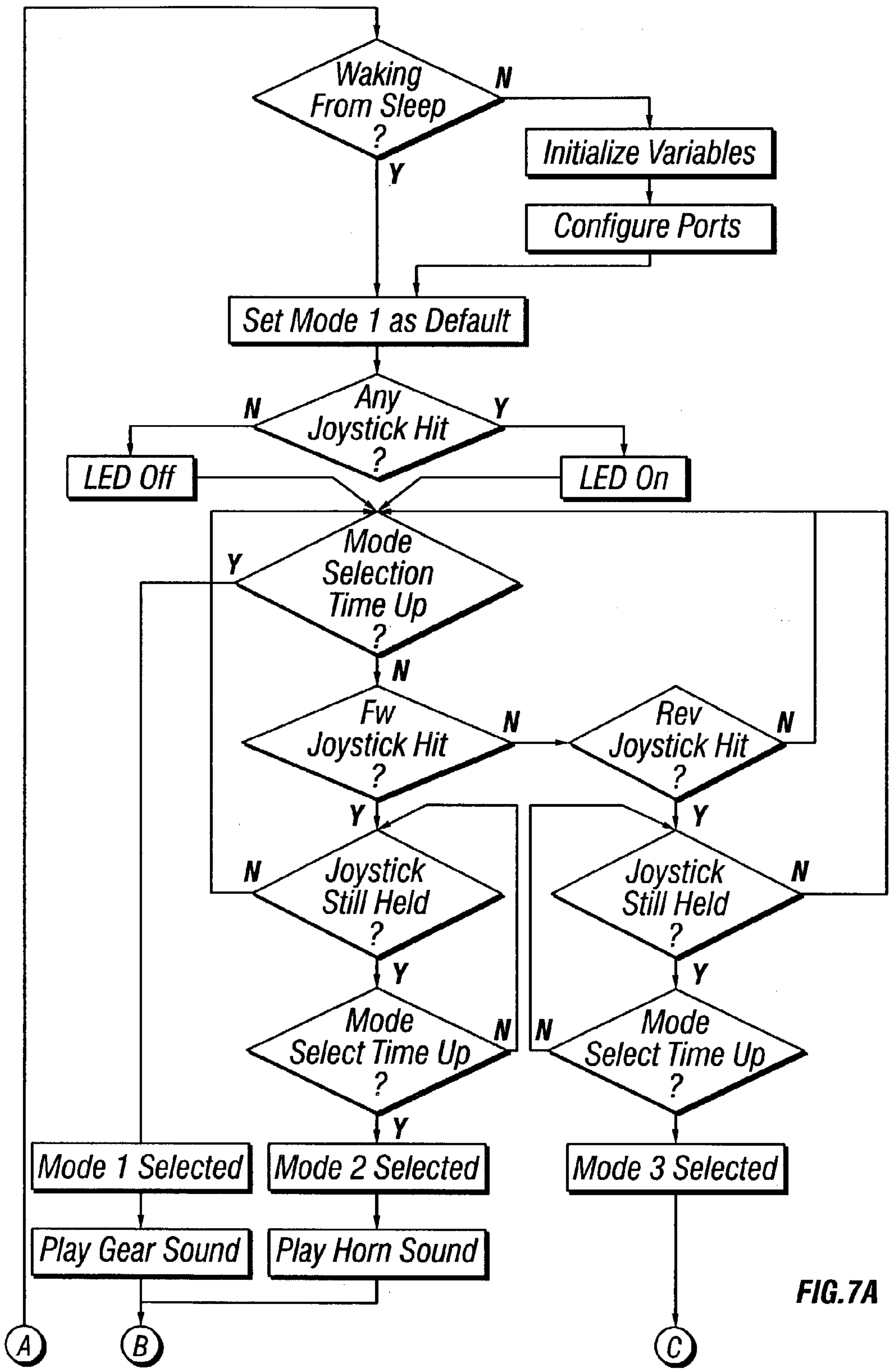

Waking From Sleep ?
N
Y
Initialize Variables
Configure Ports
Set Mode 1 as Default
Any Joystick Hit ?
N
Y
LED Off
LED On
Mode Selection Time Up ?
Y
N
Fw Joystick Hit ?
N
Rev Joystick Hit ?
N
Y
Y
Joystick Still Held ?
N
Joystick Still Held ?
N
Y
Y
Mode Select Time Up ?
N
N
Mode Select Time Up ?
Y
Mode 1 Selected
Mode 2 Selected
Mode 3 Selected
Play Gear Sound
Play Horn Sound
A
B
C

FIG.7A

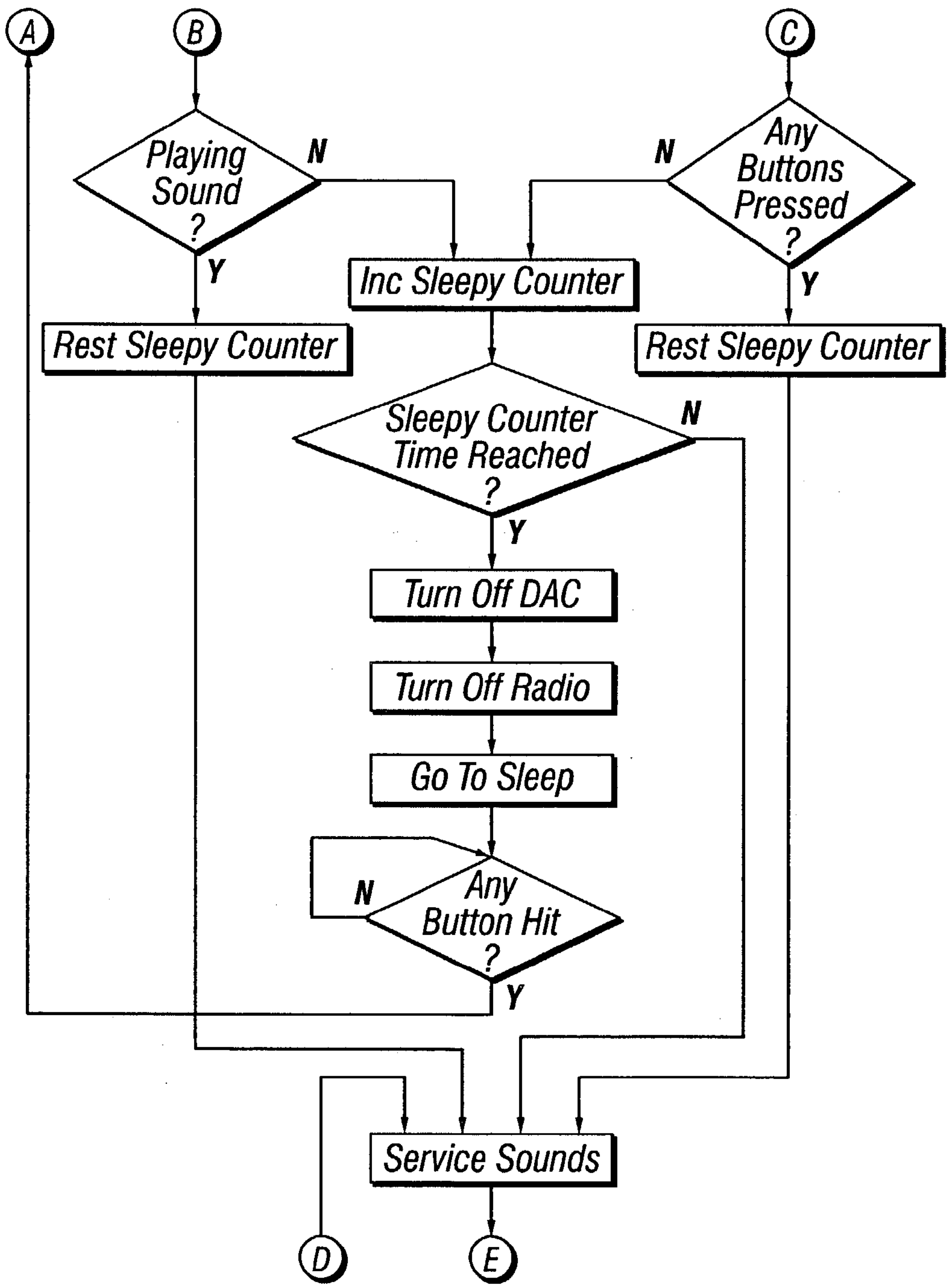

A
B
C
Playing Sound ?
N
Y
Any Buttons Pressed ?
N
Y
Inc Sleepy Counter
Rest Sleepy Counter
Rest Sleepy Counter
Sleepy Counter Time Reached ?
N
Y
Turn Off DAC
Turn Off Radio
Go To Sleep
Any Button Hit ?
N
Y
Service Sounds
D
E

FIG.7B

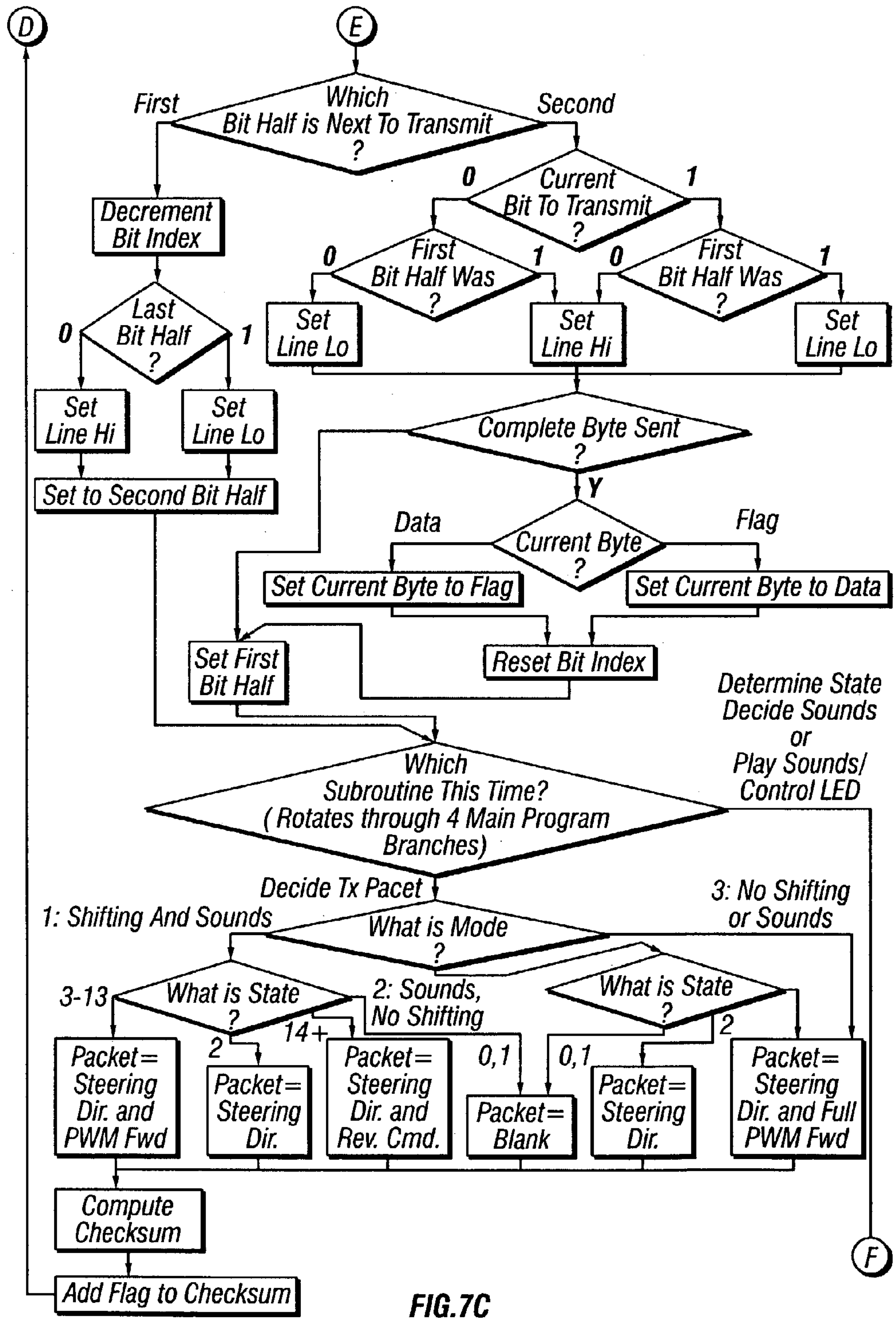

D
E
Which
Bit Half is Next To Transmit
?
First
Second
Decrement
Bit Index
Current
Bit To Transmit
?
0
1
First
Bit Half Was
?
0
1
First
Bit Half Was
?
0
1
Last
Bit Half
?
0
1
Set
Line Lo
Set
Line Hi
Set
Line Lo
Set
Line Hi
Set
Line Lo
Set to Second Bit Half
Complete Byte Sent
?
Y
Current Byte
?
Data
Flag
Set Current Byte to Flag
Set Current Byte to Data
Reset Bit Index
Set First
Bit Half
Determine State
Decide Sounds
or
Play Sounds/
Control LED
Which
Subroutine This Time?
(Rotates through 4 Main Program
Branches)
Decide Tx Pacet
1: Shifting And Sounds
What is Mode
?
3: No Shifting
or Sounds
What is State
?
3-13
2: Sounds,
No Shifting
What is State
?
2
14+
0,1
0,1
2
Packet=
Steering
Dir. and
PWM Fwd
Packet=
Steering
Dir.
Packet=
Steering
Dir. and
Rev. Cmd.
Packet=
Blank
Packet=
Steering
Dir.
Packet=
Steering
Dir. and Full
PWM Fwd
Compute
Checksum
Add Flag to Checksum
F

FIG.7C

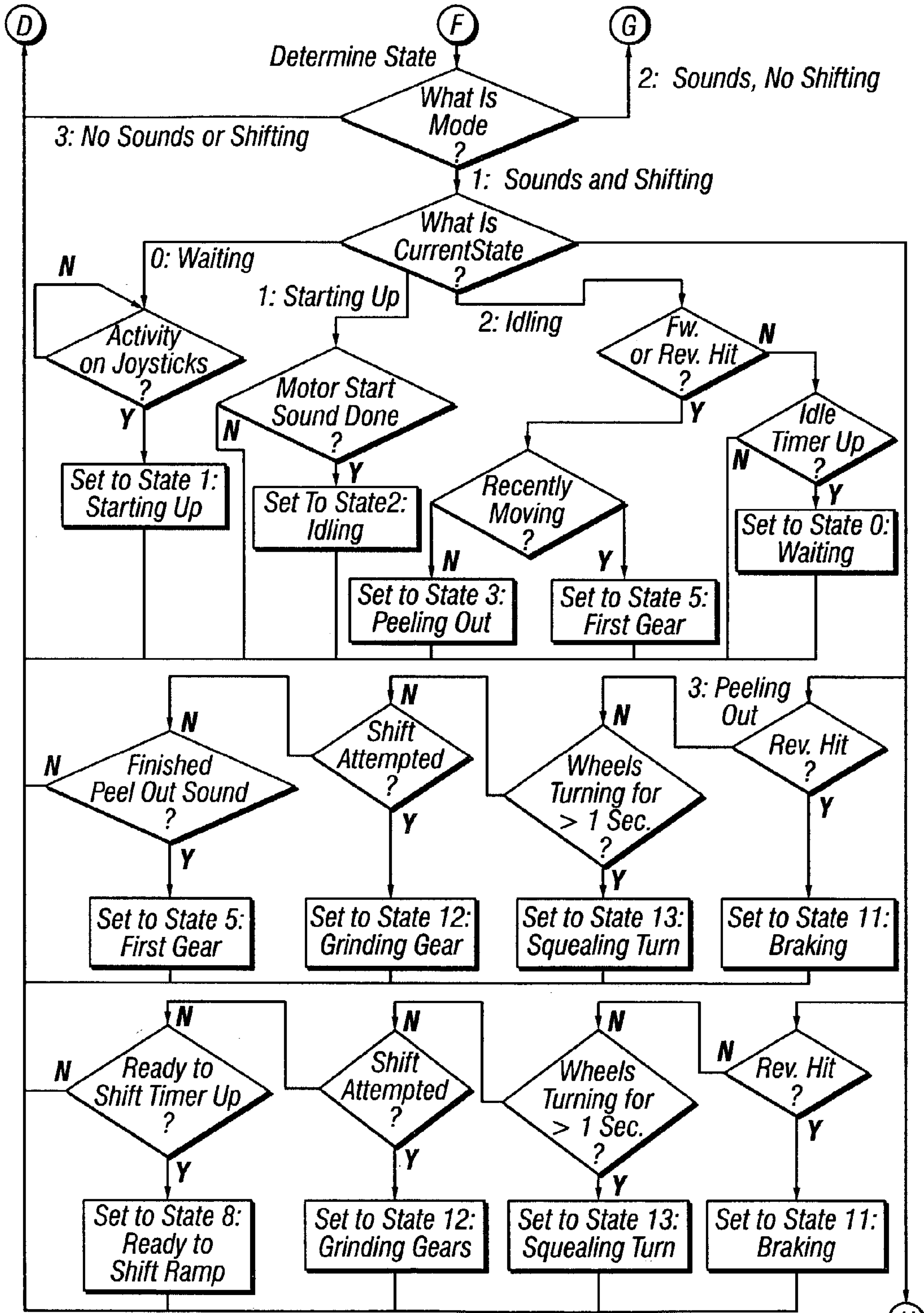

D
F
G
Determine State
What Is Mode ?
2: Sounds, No Shifting
3: No Sounds or Shifting
1: Sounds and Shifting
What Is CurrentState ?
0: Waiting
1: Starting Up
2: Idling
N
Activity on Joysticks ?
Y
Set to State 1: Starting Up
Motor Start Sound Done ?
N
Y
Set To State2: Idling
Fw. or Rev. Hit ?
N
Y
Recently Moving ?
N
Y
Set to State 3: Peeling Out
Set to State 5: First Gear
Idle Timer Up ?
N
Y
Set to State 0: Waiting
N
Finished Peel Out Sound ?
Y
Set to State 5: First Gear
N
Shift Attempted ?
Y
Set to State 12: Grinding Gear
N
Wheels Turning for > 1 Sec. ?
Y
Set to State 13: Squealing Turn
3: Peeling Out
Rev. Hit ?
Y
Set to State 11: Braking
N
Ready to Shift Timer Up ?
Y
Set to State 8: Ready to Shift Ramp
N
Shift Attempted ?
Y
Set to State 12: Grinding Gears
N
Wheels Turning for > 1 Sec. ?
Y
Set to State 13: Squealing Turn
N
Rev. Hit ?
Y
Set to State 11: Braking
H

FIG.7D

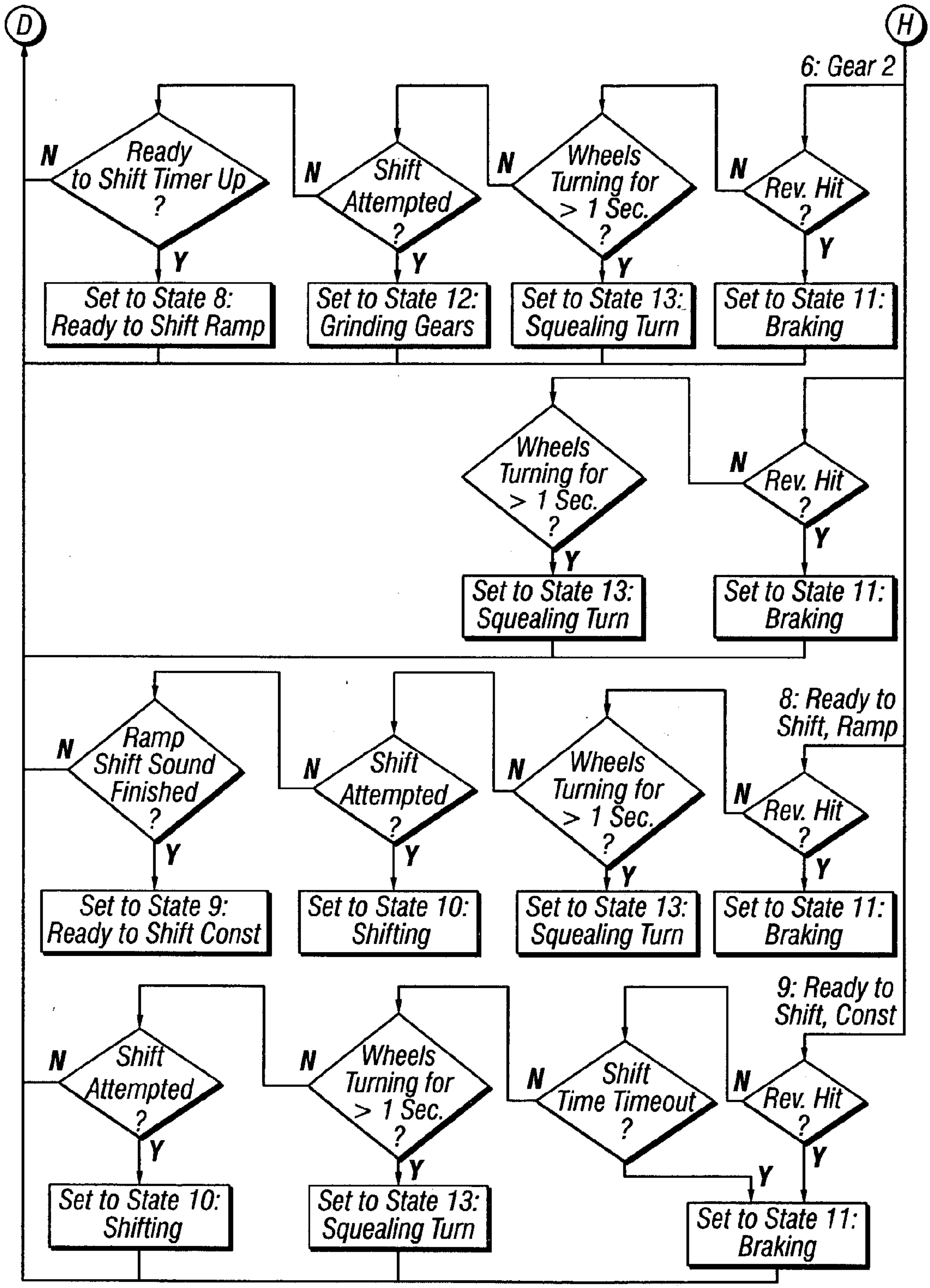

D
H
6: Gear 2
Ready to Shift Timer Up ?
N
Y
Set to State 8: Ready to Shift Ramp
Shift Attempted ?
N
Y
Set to State 12: Grinding Gears
Wheels Turning for > 1 Sec. ?
N
Y
Set to State 13: Squealing Turn
Rev. Hit ?
N
Y
Set to State 11: Braking
Wheels Turning for > 1 Sec. ?
Y
Set to State 13: Squealing Turn
Rev. Hit ?
N
Y
Set to State 11: Braking
8: Ready to Shift, Ramp
Ramp Shift Sound Finished ?
N
Y
Set to State 9: Ready to Shift Const
Shift Attempted ?
N
Y
Set to State 10: Shifting
Wheels Turning for > 1 Sec. ?
N
Y
Set to State 13: Squealing Turn
Rev. Hit ?
N
Y
Set to State 11: Braking
9: Ready to Shift, Const
Shift Attempted ?
N
Y
Set to State 10: Shifting
Wheels Turning for > 1 Sec. ?
N
Y
Set to State 13: Squealing Turn
Shift Time Timeout ?
N
Y
Rev. Hit ?
N
Y
Set to State 11: Braking

FIG. 7E

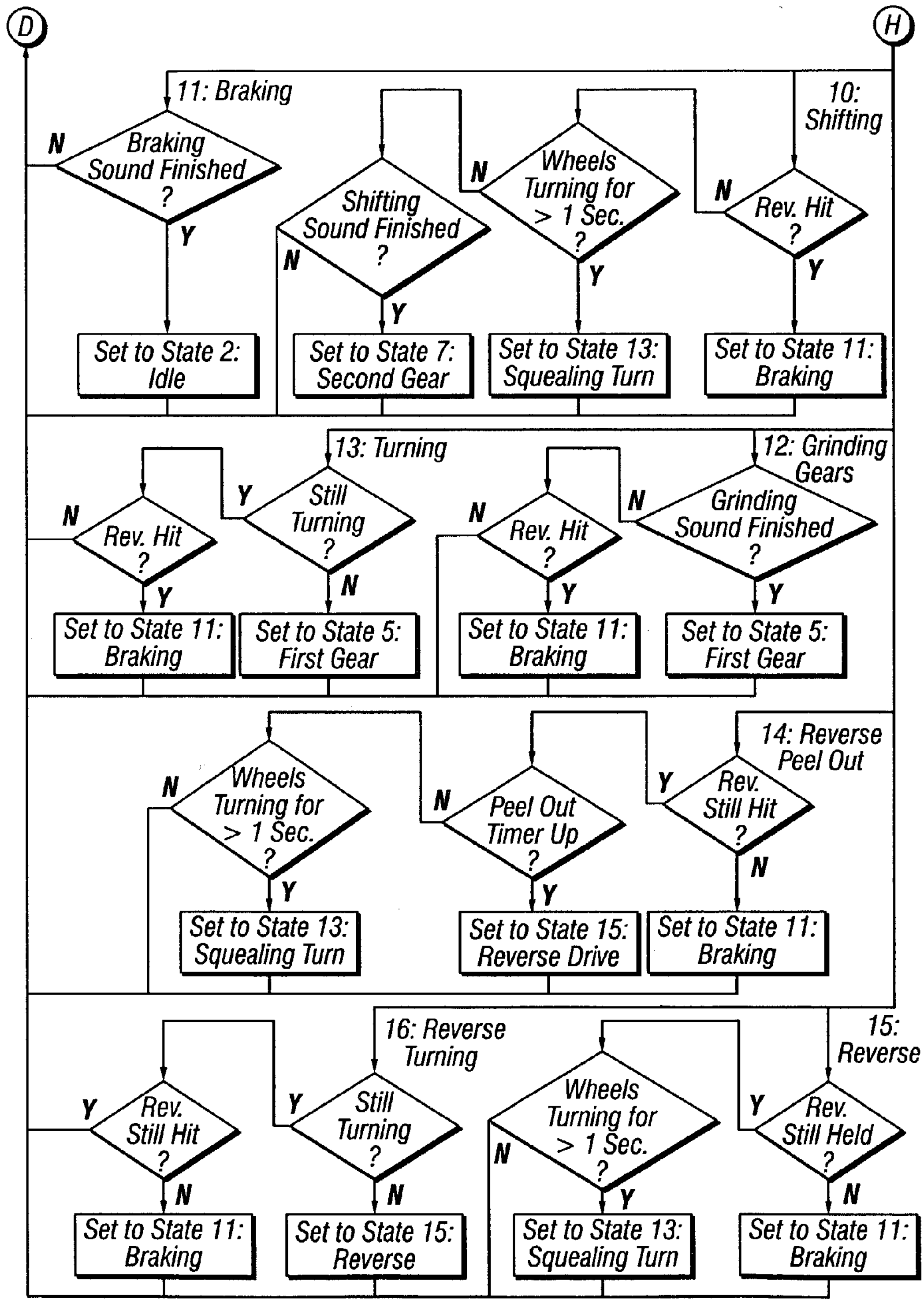

D
H
11: Braking
Braking Sound Finished ?
N
Y
Set to State 2: Idle
Shifting Sound Finished ?
N
Y
Set to State 7: Second Gear
Wheels Turning for > 1 Sec. ?
N
Y
Set to State 13: Squealing Turn
10: Shifting
Rev. Hit ?
N
Y
Set to State 11: Braking
13: Turning
Still Turning ?
Y
N
Set to State 5: First Gear
Rev. Hit ?
N
Y
Set to State 11: Braking
12: Grinding Gears
Grinding Sound Finished ?
N
Y
Set to State 5: First Gear
Rev. Hit ?
N
Y
Set to State 11: Braking
Wheels Turning for > 1 Sec. ?
N
Y
Set to State 13: Squealing Turn
Peel Out Timer Up ?
N
Y
Set to State 15: Reverse Drive
14: Reverse Peel Out
Rev. Still Hit ?
Y
N
Set to State 11: Braking
16: Reverse Turning
Rev. Still Hit ?
Y
N
Set to State 11: Braking
Still Turning ?
Y
N
Set to State 15: Reverse
Wheels Turning for > 1 Sec. ?
N
Y
Set to State 13: Squealing Turn
15: Reverse
Rev. Still Held ?
Y
N
Set to State 11: Braking

FIG.7F

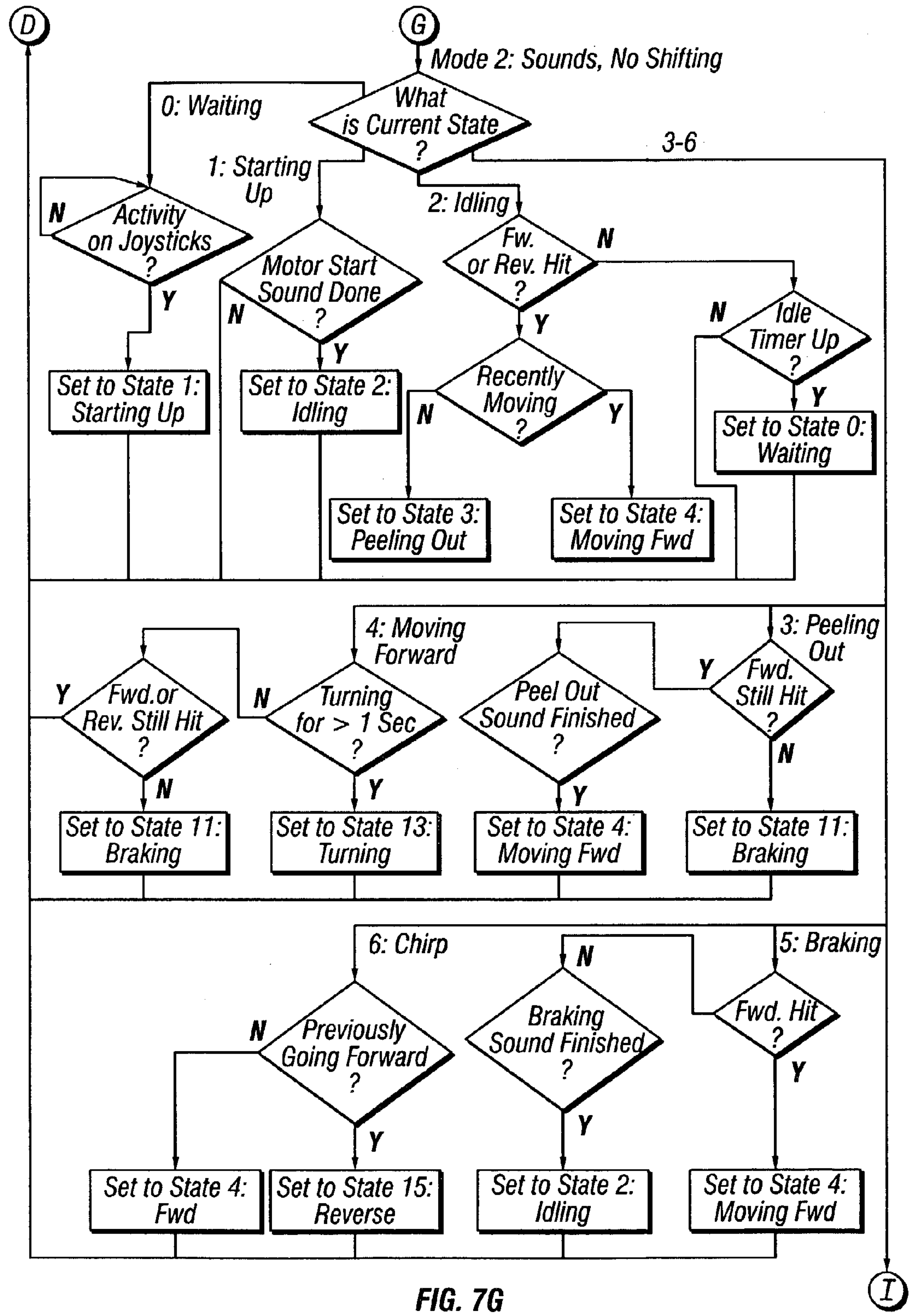

D
G
Mode 2: Sounds, No Shifting
What is Current State ?
0: Waiting
1: Starting Up
2: Idling
3-6
Activity on Joysticks ?
N
Y
Motor Start Sound Done ?
Fw. or Rev. Hit ?
Idle Timer Up ?
Set to State 1: Starting Up
Set to State 2: Idling
Recently Moving ?
Set to State 0: Waiting
Set to State 3: Peeling Out
Set to State 4: Moving Fwd
4: Moving Forward
3: Peeling Out
Fwd.or Rev. Still Hit ?
Turning for > 1 Sec ?
Peel Out Sound Finished ?
Fwd. Still Hit ?
Set to State 11: Braking
Set to State 13: Turning
Set to State 4: Moving Fwd
Set to State 11: Braking
6: Chirp
5: Braking
Previously Going Forward ?
Braking Sound Finished ?
Fwd. Hit ?
Set to State 4: Fwd
Set to State 15: Reverse
Set to State 2: Idling
Set to State 4: Moving Fwd
I

FIG. 7G

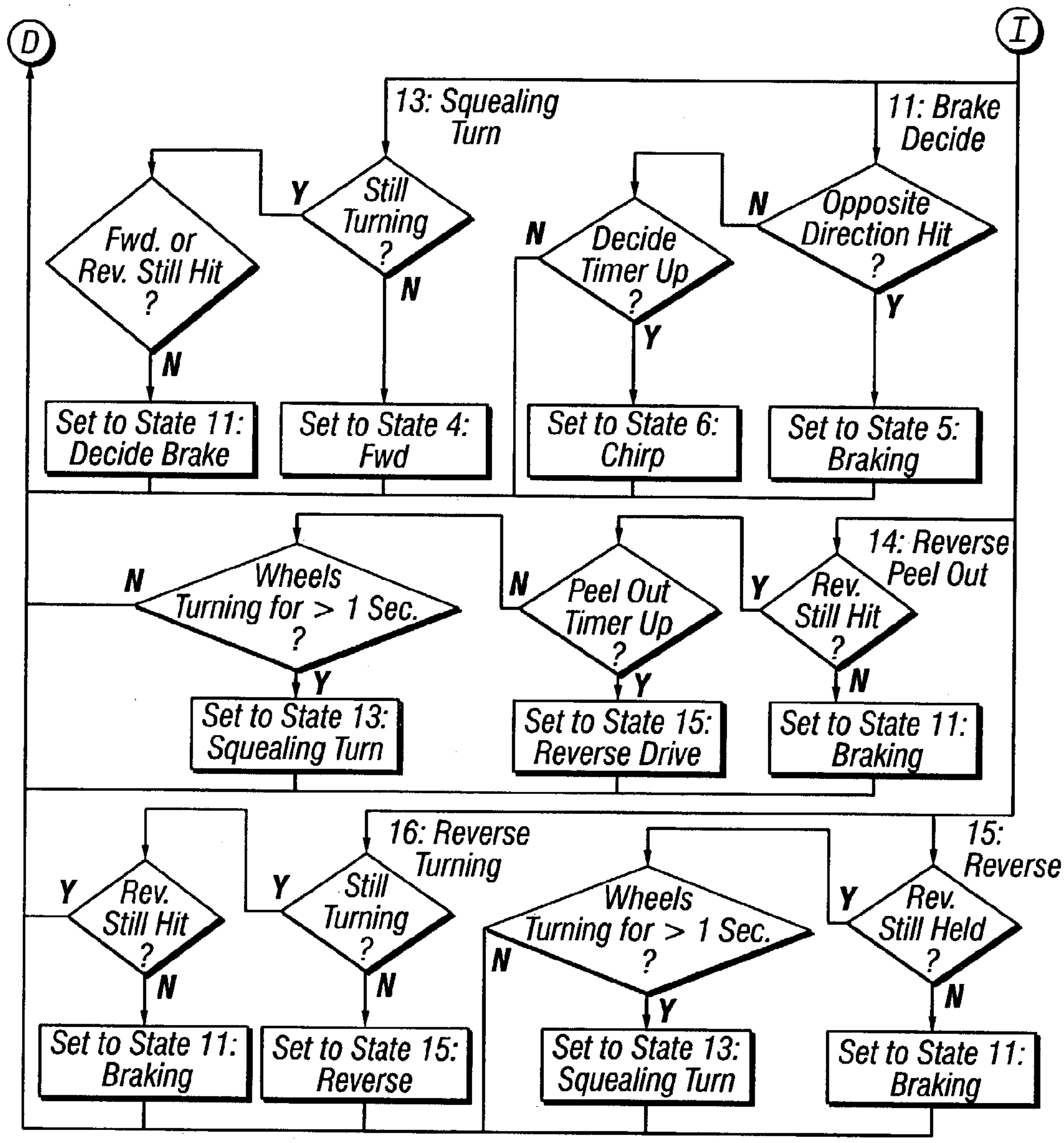

D
I
13: Squealing Turn
11: Brake Decide
Fwd. or Rev. Still Hit ?
Still Turning ?
Decide Timer Up ?
Opposite Direction Hit ?
Y
N
N
N
Y
Y
Set to State 11: Decide Brake
Set to State 4: Fwd
Set to State 6: Chirp
Set to State 5: Braking
14: Reverse Peel Out
Wheels Turning for > 1 Sec. ?
Peel Out Timer Up ?
Rev. Still Hit ?
Set to State 13: Squealing Turn
Set to State 15: Reverse Drive
Set to State 11: Braking
16: Reverse Turning
15: Reverse
Rev. Still Hit ?
Still Turning ?
Wheels Turning for > 1 Sec. ?
Rev. Still Held ?
Set to State 11: Braking
Set to State 15: Reverse
Set to State 13: Squealing Turn
Set to State 11: Braking

FIG.7H

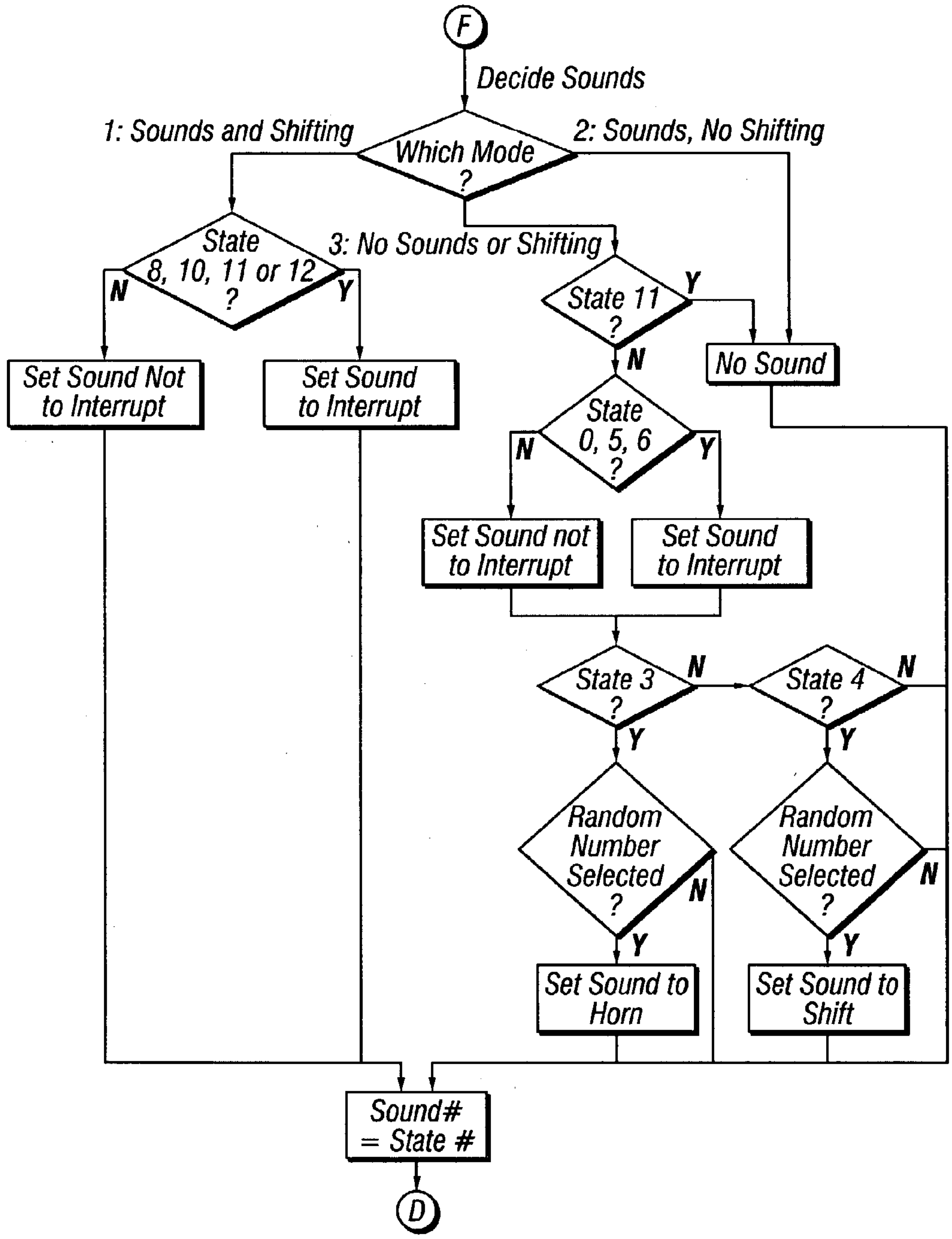

F
Decide Sounds
1: Sounds and Shifting
Which Mode ?
2: Sounds, No Shifting
State 8, 10, 11 or 12 ?
3: No Sounds or Shifting
N
Y
State 11 ?
Y
N
No Sound
Set Sound Not to Interrupt
Set Sound to Interrupt
State 0, 5, 6 ?
N
Y
Set Sound not to Interrupt
Set Sound to Interrupt
State 3 ?
N
State 4 ?
N
Y
Y
Random Number Selected ?
N
Random Number Selected ?
N
Y
Y
Set Sound to Horn
Set Sound to Shift
Sound# = State #
D

FIG.7I

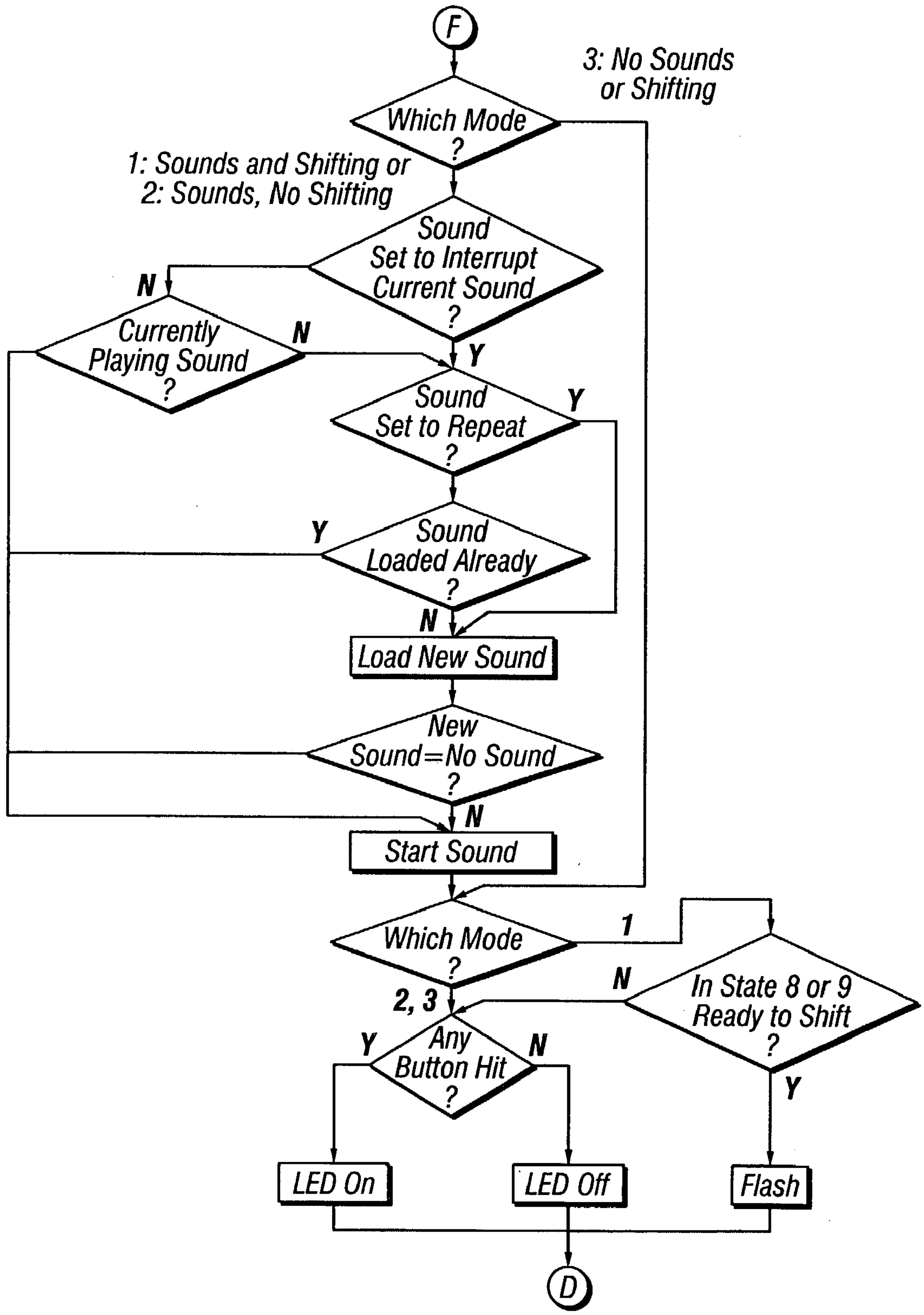

F
Which Mode ?
3: No Sounds or Shifting
1: Sounds and Shifting or
2: Sounds, No Shifting
Sound Set to Interrupt Current Sound ?
N
Currently Playing Sound ?
N
Y
Sound Set to Repeat ?
Y
Y
Sound Loaded Already ?
N
Load New Sound
New Sound=No Sound ?
N
Start Sound
Which Mode ?
1
2, 3
In State 8 or 9 Ready to Shift ?
N
Y
Any Button Hit ?
Y
N
LED On
LED Off
Flash
D

FIG.7J

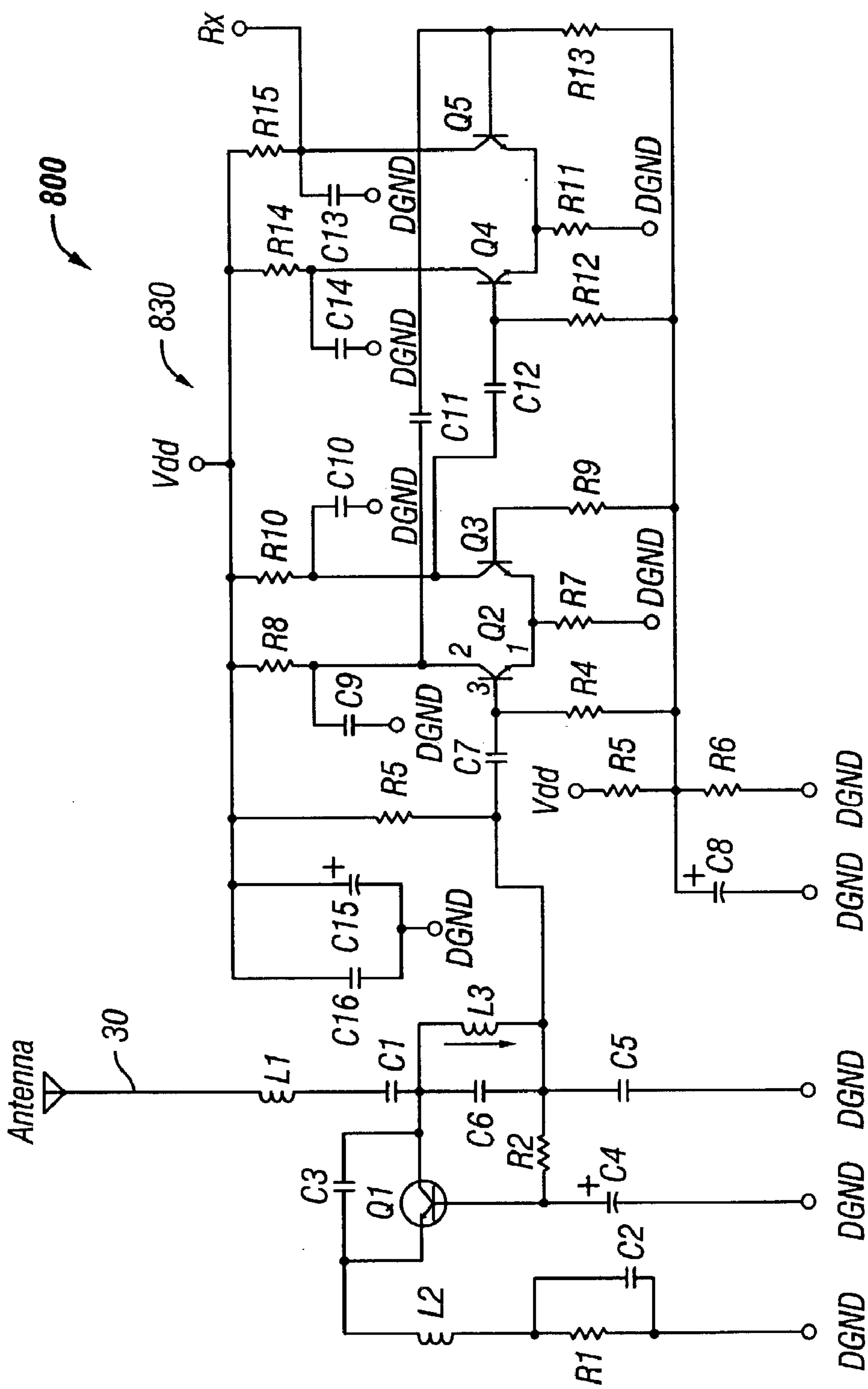

800
830
Antenna
30
L1
L2
L3
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
C16
Q1
Q2
Q3
Q4
Q5
R1
R2
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
Vdd
Rx
DGND

FIG. 8A

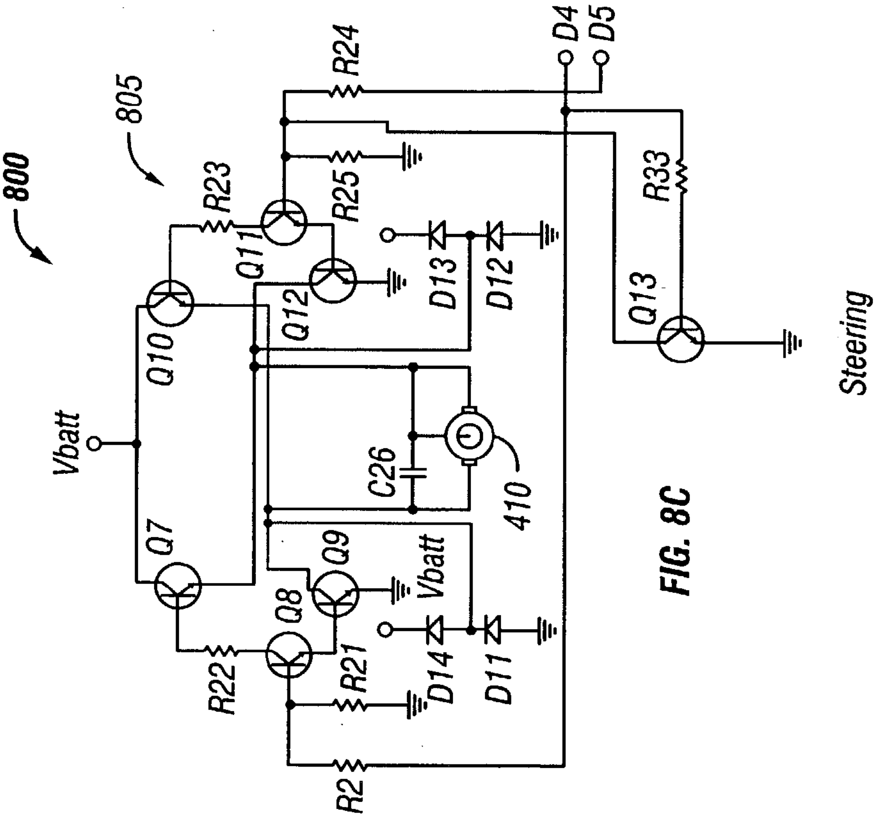

800
805
Vbatt
Q7
Q8
Q9
Q10
Q11
Q12
Q13
R2
R21
R22
R23
R24
R25
R33
C26
D11
D12
D13
D14
410
D4
D5
Steering

FIG. 8C

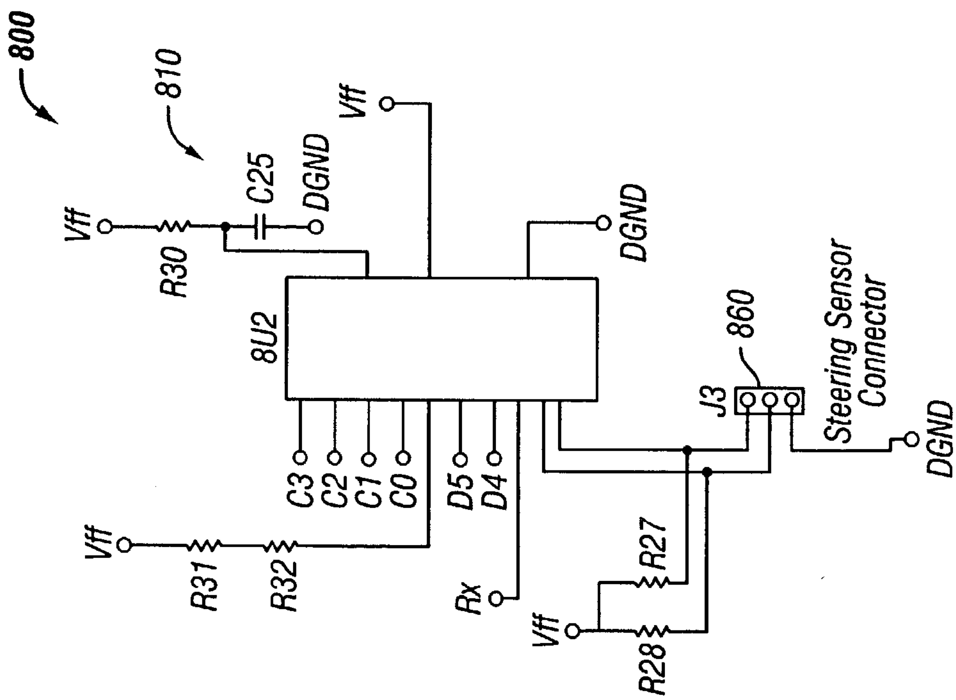

800
810
Vff
R30
C25
DGND
8U2
R31
R32
C3
C2
C1
C0
D5
D4
Rx
R27
R28
J3
860
Steering Sensor Connector

FIG. 8B

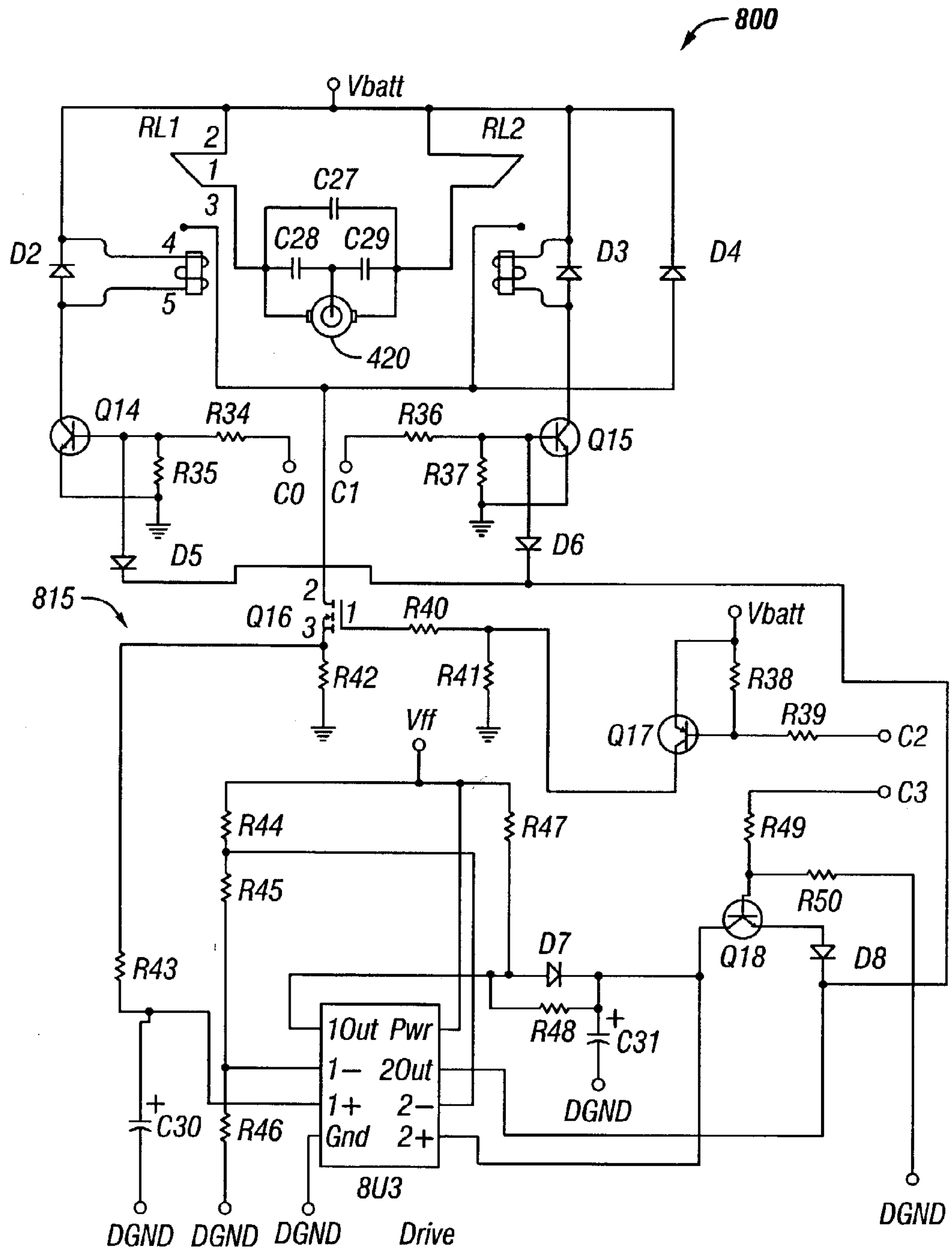

800
Vbatt
RL1
2
1
3
RL2
C27
C28
C29
4
5
D2
D3
D4
420
Q14
R34
R35
C0
R36
C1
R37
Q15
D5
D6
815
2
Q16
3
1
R40
R42
R41
Vbatt
R38
R39
C2
Q17
Vff
C3
R44
R47
R49
R45
R50
Q18
D7
D8
R43
R48
C31
1Out Pwr
1− 2Out
1+ 2−
Gnd 2+
8U3
C30
R46
DGND
DGND
DGND
DGND
Drive
DGND

FIG. 8D

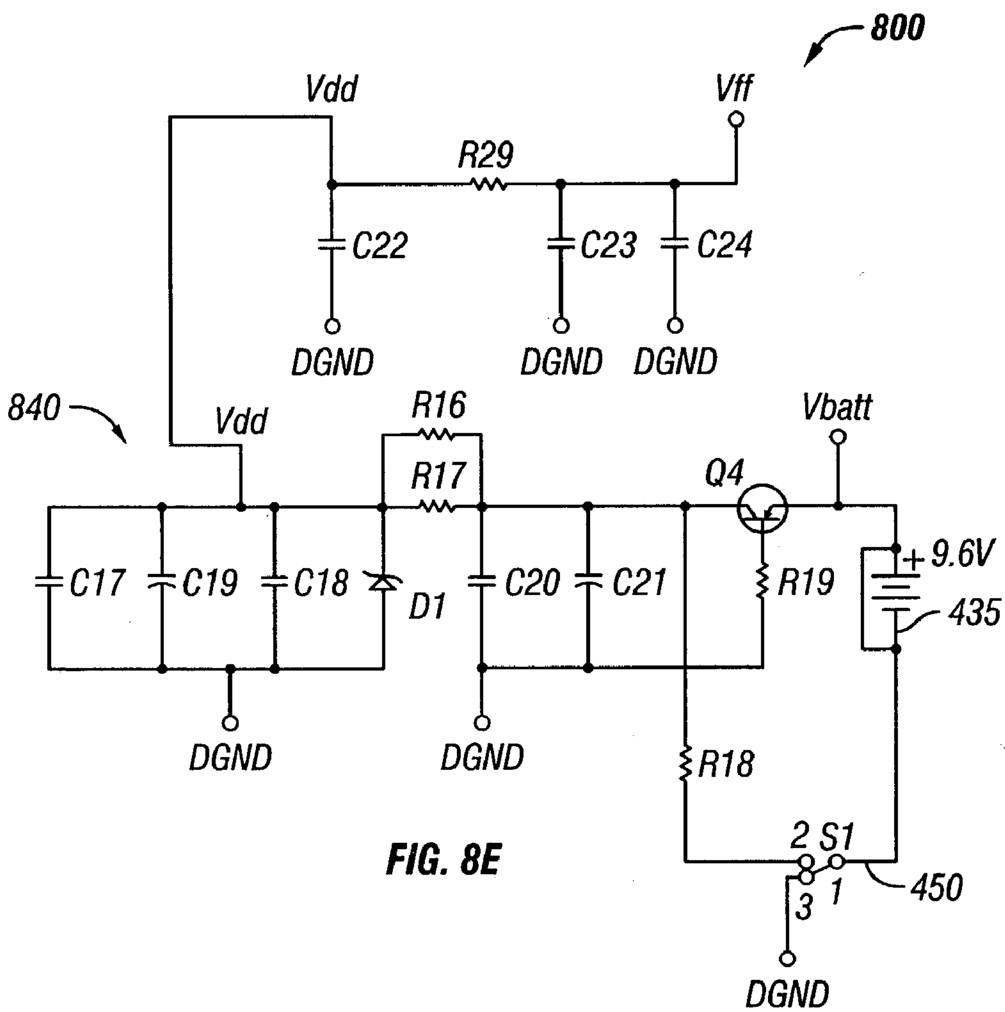

800
Vdd
Vff
R29
C22
C23
C24
DGND
DGND
DGND
840
Vdd
R16
R17
Vbatt
Q4
C17
C19
C18
D1
C20
C21
R19
+ 9.6V
435
DGND
DGND
R18
2 S1
3
1
450
DGND

FIG. 8E

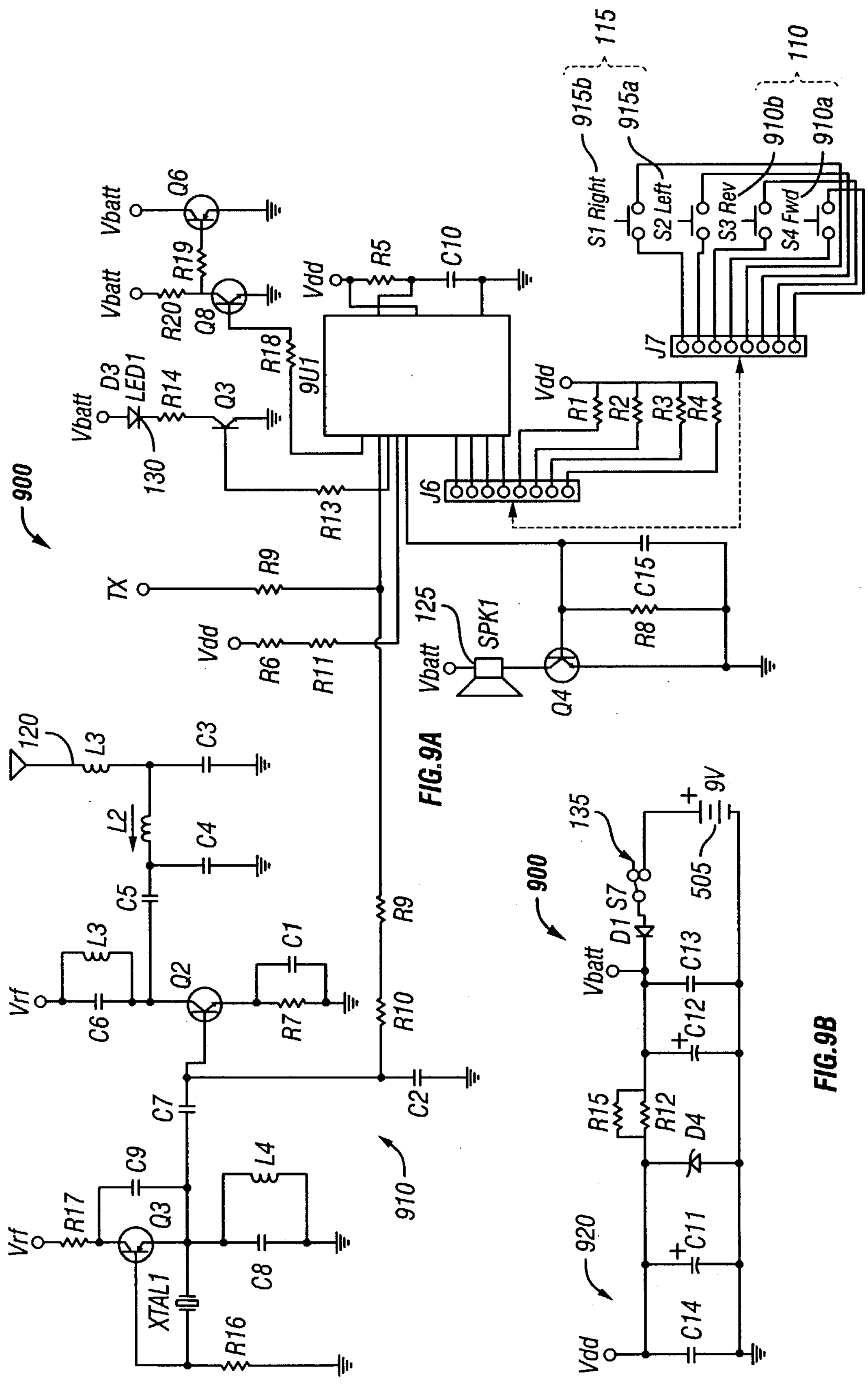

900
Vbatt
Q6
R19
R20
Q8
D3
LED1
R14
Q3
130
R18
9U1
Vdd
R5
C10
R1
R2
R3
R4
J6
J7
S1 Right
S2 Left
S3 Rev
S4 Fwd
915b
915a
115
910b
910a
110
TX
R9
R13
R6
R11
125
SPK1
Q4
C15
R8
120
L3
C3
L2
C4
C5
Vrf
C6
Q2
C1
R7
R10
C2
C7
R17
C9
L4
C8
XTAL1
R16
910
135
9V
505
D1
S7
C13
C12
R15
R12
D4
C11
C14
920

FIG.9A

FIG.9B

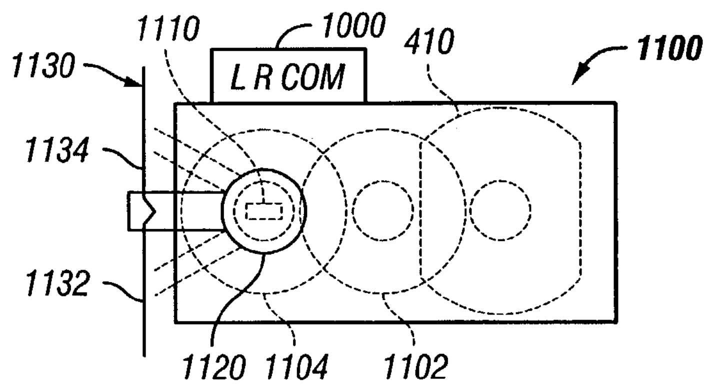


FIG. 10A

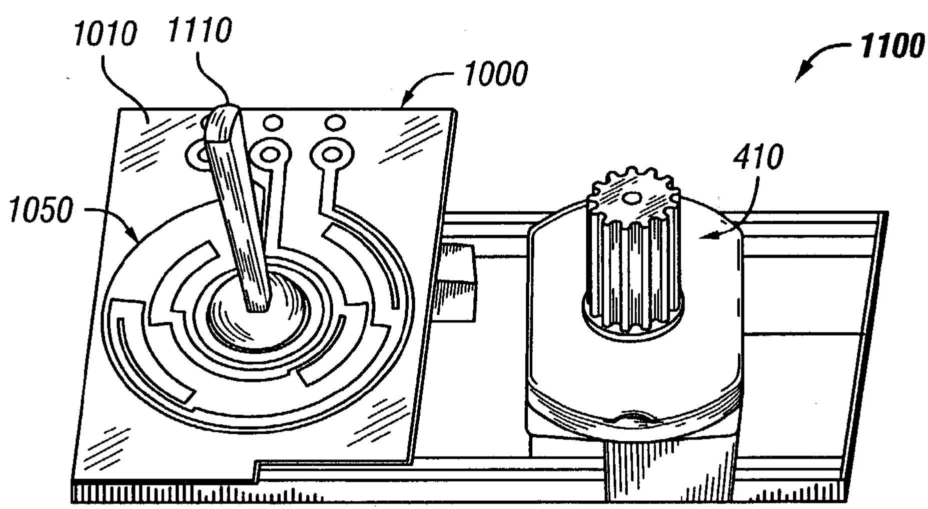


FIG. 10B

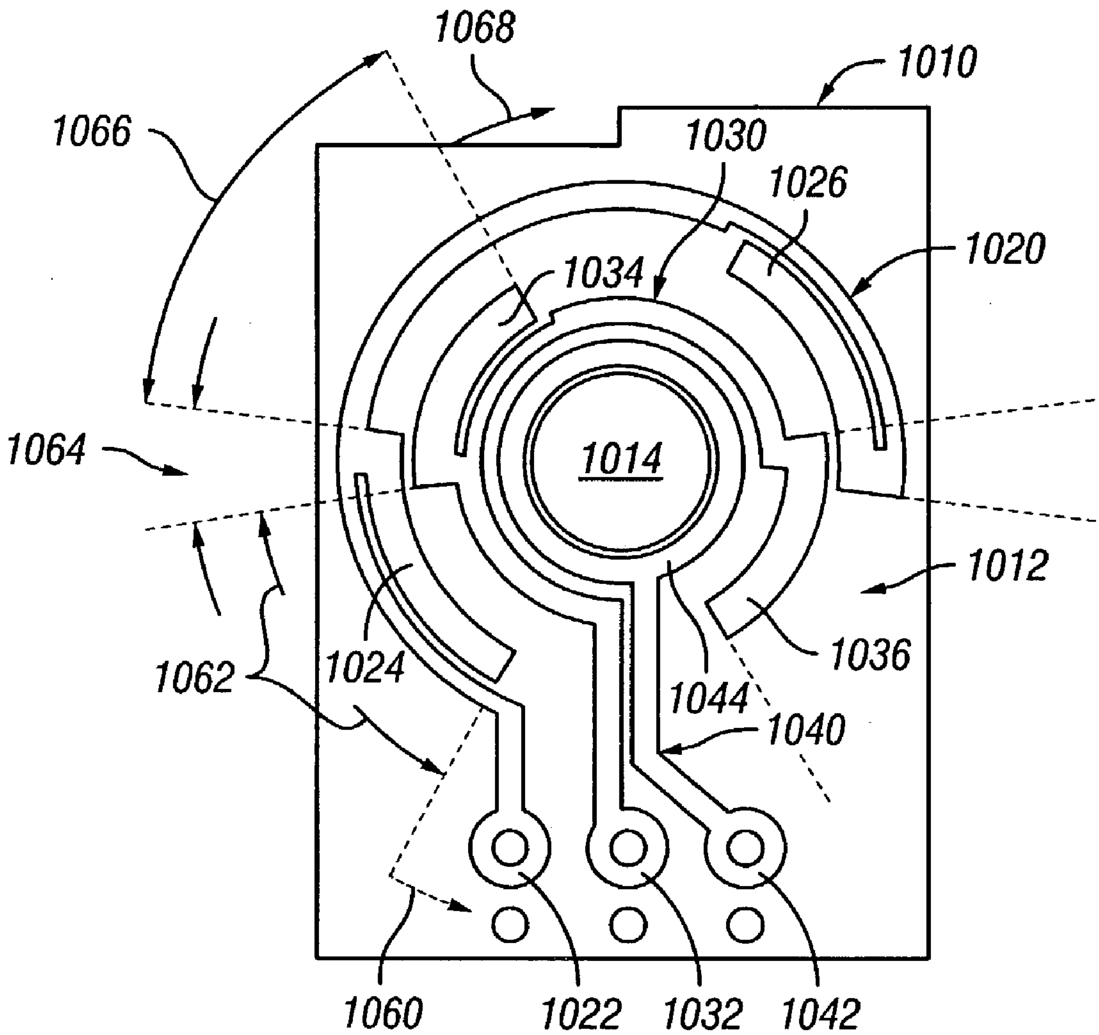

1068
1010
1066
1030
1026
1020
1034
1064
1014
1012
1062
1024
1036
1044
1040
1060
1022
1032
1042

FIG. 11

TOY VEHICLE WIRELESS CONTROL SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 60/340,591, filed Oct. 30, 2001, entitled "Toy Vehicle Wireless Control System," which is incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

[0002] This invention relates to toy vehicles and, in particular, to remotely controlled, motorized toy vehicles.

SUMMARY OF THE INVENTION

[0003] The invention is in a toy vehicle remote control transmitter unit including a housing, a plurality of manual input elements mounted on the housing for manual movement, a microprocessor in the housing operably coupled with each manual input element on the housing, and a signal transmitter operably coupled with the microprocessor to transmit wireless control signals generated by the microprocessor to a toy vehicle. The invention is characterized in that the microprocessor is configured for at least two different modes of operation. One of the modes emulates manual transmission operation of the toy vehicle by being in any one of a plurality of different gear states and transmitting through the transmitter forward propulsion control signals representing different speed ratios for each of the plurality of different gear states. The microprocessor is further configured to consecutively advance through the different gear states in response to successive manual operations of at least one of the manual input devices.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0004] The following detailed description of preferred embodiments of the invention, will be better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings embodiments which are presently preferred. It should be understood, however, that the invention is not limited to the precise arrangements and instrumentalities shown. In the drawings:

[0005] **FIG. 1A** is a top plan view of an exemplary remote control/transmitter used in accordance with the present invention;

[0006] **FIG. 1B** is an exemplary toy vehicle remotely controlled by the remote control/transmitter of **FIG. 1A**;

[0007] **FIG. 2** is a timing diagram showing an analog output of a control circuit used to drive different motor speeds of the toy vehicle of **FIG. 1B** in accordance with a preferred embodiment of the present invention;

[0008] **FIG. 3** is a diagram showing a trapezoidal velocity profile of a steering finction of the toy vehicle of **FIG. 1B**;

[0009] **FIG. 4** is a schematic diagram of a control circuit in the toy vehicle of **FIG. 1B**, which is directly responsive to steering commands received in accordance with the present invention;

[0010] **FIG. 5** is a schematic diagram of a speed shifter remote control/transmitter circuit which sends steering commands to the control circuit of **FIG. 4**;

[0011] **FIGS. 6A, 6B**, 6C and 6D, taken together, is a flow chart illustrating the operation of the vehicle control circuit of **FIG. 4**;

[0012] **FIGS. 7A, 7B**, 7C, 7D, 7E, 7F, 7G, 7H, 7I and 7J, taken together, is a flow chart illustrating the operation of the speed shifter remote control/transmitter circuit of **FIG. 5**;

[0013] **FIGS. 8A, 8B**, 8C, 8D and 8E, taken together, is a schematic diagram of a toy vehicle control circuit which processes received steering commands based on current steering position of the toy vehicle in accordance with an alternate embodiment of the present invention;

[0014] **FIGS. 9A and 9B**, taken together, is a schematic diagram of a speed shifter remote control/transmitter circuit in accordance with an alternate embodiment of the present invention;

[0015] **FIG. 10A** depicts a steering output assembly;

[0016] **FIG. 10B** depicts the assembly of **FIG. 10A** with the output member and reduction gearing removed; and

[0017] **FIG. 11** depicts the stationary portion or contact member of a steering sensor.

DETAILED DESCRIPTION OF THE INVENTION

[0018] Related U.S. Application No. 60/340,591 filed Oct. 30, 2001 is incorporated by reference herein. The present invention is a toy vehicle wireless control system which includes a remote control/transmitter **100** (**FIG. 1A**) with a speed shifter remote control/transmitter circuit **500** (see **FIG. 5**) or **900** (see **FIGS. 9A, 9B**), and a remotely controlled toy vehicle **20** (**FIG. 1B**) with a receiver/microprocessor based toy vehicle control circuit **400** (see **FIG. 4**) or **900** (see FIGS. 9A-9E), also hereinafter referred to as a speed shifter receiver circuit.

[0019] The remote control/transmitter **100** depicted in **FIG. 1A** includes a housing **105** and a plurality of manual input elements **110, 115** mounted on housing **105** and used for controlling the manual movement of a toy vehicle **20**. The manual input elements **110, 115** are conventionally used to supply propulsion or movement commands and steering commands, respectively. They also enable selection among three different modes of operation or usage (hereinafter referred to as "Mode 1,""Mode 2," and "Mode 3"), each having a different play pattern. Power is selectively provided to circuitry in the remote control/transmitter **100** via ON/OFF switch **135** (in phantom in **FIG. 1A**).

[0020] Car **20** is shown in **FIG.** 1B and includes a chassis **22**, body **24**, rear drive wheels **26** operably coupled to drive/propulsion motor **420** (phantom) and front free rotating wheels **28** operably coupled with steering motor **410** (phantom). An antenna **30** receives command signals from remote control/transmitter **10** and carries those signals to the vehicle control circuit **400** (phantom) or **800** (not shown in **FIG. 1B**). An on-off switch **450** turns the circuit **400** on and off, and a battery power supply **435** provides power to the circuit **400** and motors **410, 420**.

[0021] FIG. 4 shows a schematic diagram of a vehicle control circuit 400 in the toy vehicle 20. The vehicle control circuit 400 includes a steering motor control circuit 405 which controls steering motor 410, and a propulsion motor control circuit 415 which controls drive motor 420. Microprocessor 4U1 is in communication with steering motor and drive motor control circuits 405, 415, and controls all other functions executed within the toy vehicle 20. A vehicle receiver circuit 430 receives control signals sent by remote control/transmitter 100 and amplifies and sends the control signals to microprocessor 4U1 for processing. A power supply circuit 440 powers the vehicle control circuit 400 in toy vehicle 20 and the steering and propulsion motors 410, 420, respectively.

[0022] FIG. 5 shows a transmitter circuit 500 in the remote control/transmitter 100 (see FIG. 1A) that is powered by a battery 505 in communication with a two-position switch 135 that is used to turn the device 100 on and off and for selecting one of the modes. The transmitter circuit 500 also includes a microprocessor 5U1. The microprocessor 5U1 is operably coupled with each of the manual input elements 110, 115. The remote control/transmitter 100 must first be turned off via switch 135 to change the mode used. Manual input element 110 is preferably a center biased rocker button operating momentary contact switches 110*a* and 110*b*, as shown in FIG. 5. When pressed, the manual input element 110 causes one of contact switches 110*a* and 110*b* to change states. This is sensed by the microprocessor 5U1 which responds by transmitting a signal via antenna 120 to cause remotely controlled toy vehicle 20, which includes receiver/microprocessor 4U1, to move forward or backward. Manual input element 115 is also preferably a center biased rocker button operating momentary contact switches 115*a* and 115*b* in FIG. 5 which, when pressed, causes the remote control/transmitter 100 to transmit via antenna 120 a command to receiver/microprocessor 4U1 causing the toy vehicle 20 to steer to the left or to the right. When manual input element 115 is not pressed (i.e. in center position), the toy vehicle 20 travels in a straight path. When the manual input element 110 is not pressed, the vehicle 20 stops.

[0023] Mode 1, a first mode of operation or usage, is the default mode achieved when the remote control/transmitter 100 is activated from a deactivated state by moving on-off switch 135 in FIG. 5 from an "off" position to an "on" position. This mode has a multiple-speed (3-speed in the present embodiment) manual gear-shifting play pattern in which the microprocessor 5U1 emulates a manual transmission operation of the toy vehicle 20 and in which corresponding sounds are generated by the microprocessor 5U1 and played on a speaker 125 in the remote control/transmitter 100. Mode 1 has the following features and characteristics:

[0024] (1) The motionless toy vehicle 20 is put into motion by pressing manual input element 110 to a "forward" button position, closing or otherwise changing the nominal state of switch 110*a* on the remote control/transmitter 100. The microprocessor 5U1 is configured (i.e., programmed) to respond to the depressions of manual input element 110 by entering a first gear state of operation and generating a first forward movement command signal transmitted to the toy vehicle 20. Initially, the toy vehicle 20 responds to the first signal and moves forward at a first top speed which is less than a maximum speed the toy vehicle 20 is capable of running. The microprocessor 5U1 generates a first sound, which is outputted by speaker 125, to simulate first gear operation of the toy vehicle 20.

[0025] (2) Once the toy vehicle 20 is moving forward for a while in a first gear state (as timed by microprocessor 5U1), a visual indication (e.g., red flashing LED 130) and/or an audible sound (e.g., single horn beep) can be outputted by the microprocessor 5U1 from the remote control/transmitter 100 to signal to a user that it is OK to shift to the second gear. Shifting into a higher gear is performed by momentarily releasing and re-engaging the forward button position of manual input element 110, which closes switch 110*a* within a predetermined time window. If the time window elapses, the toy vehicle 20 will return to first gear state when the forward button position of manual input element 110 is activated (i.e., switch 110*a* is closed). Once in the second gear state, the microprocessor 4U1 commands the vehicle 20 to move forward at a second top speed that is faster than the first top speed but less than maximum speed, and preferably the microprocessor 5U1 generates a second sound which is outputted by speaker 125 to simulate second gear operation of the toy vehicle 20. Once the toy vehicle 20 is moving forward for a while in a second gear state, a visual indication (e.g., red flashing LED 130) and/or an audible sound (e.g., single horn beep) can be outputted by microprocessor 5U1 from speaker 125 of the remote control/transmitter 100 to signal to a user that it is OK to shift to the third gear. The forward button position of input element 110 closing switch 110*a* is again momentarily released and re-engaged within a predetermined time window. If the time window elapses, the toy vehicle 20 will return to first gear when the forward button position of manual input element 110 is activated. Once in the third gear state, the toy vehicle 20 moves forward at a third top speed that is faster than the second top speed, and preferably the microprocessor 5U1 generates a third sound that is outputted by speaker 125 to simulate third gear operation of the toy vehicle 20. The movement of the toy vehicle 20 is terminated by releasing the forward button position of manual input element 110 closing switch 110*a* or by pressing and then releasing reverse button position of manual input element 110 closing switch 110*b*.

[0026] (3) In the three-speed embodiment, preferably the top speed of the toy vehicle 20 may be 62.5% of maximum speed when in the first gear state, 75% of maximum speed when in the second gear state, and 100% of maximum speed when in the third gear state. Other ratios and/or additional ratios to provide four, five, six or more speeds can be used to simulate other car and truck shifting.

[0027] (4) If the gear state of the toy vehicle 20 is changed before the toy vehicle 20 reaches its top speed for the previous gear by momentarily releasing and re-engaging the forward button position of manual input element 110, before the microprocessor 5U1 opens the predetermined time window to shift, the microprocessor 5U1 generates a different audible sound (e.g., grinding noise), which is preferably outputted by the speaker 125 of the remote control/transmitter 100, to signal that the user shifted too early. Top speed is not increased.

[0028] (5) Various audible sounds (e.g., peel out, squealing tire, hard braking, accelerating motor, etc.) are preferably outputted by the remote control/transmitter 100 in response to activating the manual input elements **110, 115** on the remote control/transmitter **100**. For example, transmitting a steering command by causing manual input element **115** to close switch **115*a*** while the toy vehicle **20** is moving (e.g., forward position of manual input element **110** being pressed changing the state of switch **110*a***) causes the microprocessor **5U1** to output an audible sound (e.g., the squealing of tires) through speaker **125**. There is a small delay in producing the audible sound so that small steering corrections do not cause the audible sound to be outputted by speaker **125**. Releasing either the forward and reverse position of manual input element **110** preferably causes the microprocessor **5U1** to output an audible sound (e.g., hard breaking, tire screeching) through speaker **125**. An "idling" sound is then preferably outputted by microprocessor **5U1** through speaker **125** until a next propulsion/drive command is transmitted.

[**0029**] (6) Speed of the toy vehicle **20** is controlled by the remote control/transmitter **100** outputting propulsion control signals having PWM (Pulse Width Modulation) characteristics with duty cycles approximate for the speed ratios selected, e.g., 56%, 75%, and 100% (see **FIG. 2**). Preferably, the remote control/transmitter **100** outputs a binary signal with two or more values allocated to propulsion commands. Two binary bits can be used to identify stop and three forward speed values (e.g., first, second and third speeds). The vehicle microprocessor **4U1** is preferably programmed to power each motor **410, 420** according to a duty cycle identified by the binary bits. Referring to **FIG. 2, a** fixed time period (e.g. sixteen milliseconds) can be broken up into fractions (e.g., sixteen, one millisecond parts) and power (V hi) supplied to the motor for the fraction of the time period (e.g., $\frac{0}{16}$, $\frac{10}{16}$, $\frac{12}{16}$, $\frac{16}{16}$) commanded by the two binary bits. An $\frac{8}{16}$ duty cycle is depicted, with V hi provided for eight parts and V low (i.e. 0 Volts) provided for the remaining eight parts of the period constituting the cycle. If three bits are allocated to propulsion commands, a stop command and seven different forward and reverse speed commands can be encoded. Preferably, reverse speed is at a ratio of less than 100% for ease of vehicle control and realism.

[**0030**] Mode 2 is achieved by turning on switch **135** of the remote control/transmitter **100** while holding manual input element **110** in a "forward" movement position (changing the state of switch **110*a***) on the remote control/transmitter **100** until the microprocessor **5U1** acknowledges the command by causing the speaker **125** to output an audible sound (e.g., horn beeps) and/or the red LED **130** to flash. This mode allows the user to maneuver the toy vehicle **20** in the usual manner with sounds being generated but no gear shifting operation. The microprocessor **5U1** is preferably preprogrammed for a desired default speed, e.g., 100% forward and 50% or 100% reverse.

[**0031**] Mode 3 is achieved by turning on switch **135** of the remote control/transmitter **100** while holding manual input element **110** in a "reverse" movement position (i.e. changing state of the switch **110*b***) on the remote control/transmitter **100** until the microprocessor **5U1** causes speaker **125** to output an audible sound (e.g., horn beeps) and/or the red LED **130** to flash. This mode allows the user to maneuver the toy vehicle **20** in the usual manner with no sound generation by microprocessor **5U1** or gear shifting operation. The microprocessor **5U1** is preprogrammed for a desired default speed, e.g., 100% forward and 50% or 100% reverse.

[**0032**] A "Try Me Mode" may be provided, if desired, allowing only sound effects of the remote control/transmitter **100** to be produced while still in its packaging. Sound effects are generated by pressing any button on the transmitter. Pushing the manual input element **110** to the "forward" position can cause the start-up sound to play followed by a peel-out sound with both motor and shifting sounds. Pushing the manual input element **110** to the "reverse" position can cause the horn sound to play with the motor running sound. Pushing the manual input element **15**"left" and "right" can activate the squealing tire sound accompanied by the engine downshift sound. The "Try Me Mode" preferably is deactivated automatically when the toy is taken out of its packaging and a pull-tab is removed from the remote control/transmitter **100**, allowing the transmitter **100** and toy vehicle **20** to be operated in one of the three modes described above.

[**0033**] FIGS. **7**A-**7**J depict the various steps of an operating program **700** contained by the transmitter circuit **500**, such as by firmware or software in the microprocessor **5U1**, to operate the remote control/transmitter **100** in the multiple modes of operation and in the different shift states in the first mode of operation. Again, the microprocessor **5U1** is preferably configured to transmit commands in binary form with propulsion and/or steering commands encoded as binary bits or sets of such bits.

[**0034**] FIGS. **6**A-**6**C depict the various steps of an operating program **600** contained by the vehicle control circuit **400**, such as by firmware or software in the microprocessor **4U1**, to operate the toy vehicle **20** in the multiple modes and in the different shift states in the first mode of operation. **FIG. 6D** depicts the steps of a subroutine **604**' which is entered four different times at steps **604** in the main program **600** (FIGS. **6**A-**6**C) to increment and test the state of a pulse width modulator (PWM) timer (i.e. counter) to power or turn off power to either motor **410, 420**. The operating program **600** must be cycled through four times to increment the PWM counter a total of sixteen times to complete one PWM power cycle (sixteen parts) for either motor **410, 420**.

[**0035**] FIGS. **8**A-**8**E collectively represent a schematic diagram for a second embodiment toy vehicle control circuit indicated generally at **800** in the Figure in which **FIG. 8A** depicts a vehicle receiver circuit **830** which receives control signals sent by the remote control/transmitter **100** and amplifies and sends those signals to microprocessor **8U2** in **FIG. 8B**. Outputs D**4** and D**5** from the microprocessor **8U2** are sent to a steering motor control circuit **805** depicted in **FIG. 8C** while outputs C**0**-C**3** are transmitted from the microprocessor **8U2** to a propulsion motor control circuit **815** depicted in **FIG. 8D**. Circuit element **8U3** is a dual operating amplifier chip. Power is supplied to both the steering motor **410** in **FIG. 8**C and drive motor **420** in **FIG. 8D** as well as the other components of circuit **800** via a power supply sub circuit **430** depicted in **FIG. 8E** which include both the ON/OFF switch and a battery powered supply **435**. One difference between circuit **800** and circuit **400** is the provision of a steering feedback through connector **860** in **FIG. 8B** to the vehicle microprocessor **8U2**. The purpose of this will be described shortly.

[**0036**] **FIGS. 9A and 9B** collectively depict a second embodiment remote control/transmitter circuit indicated generally at **900** which is shown essentially in **FIG. 9**A and indicated at **910**. The only missing element is a power supply circuit **920** shown in **FIG. 9B** which provides two outputs Vdd and Vbatt. Again, manual input elements **110** and **115** control momentary contacts switches **910***a,* **910***b* and **915***a,* **915***b* respectively. These switches are located on a board separate from the board supporting a microprocessor **9U1** and are mechanically and electrically coupled together through connectors J6 and J7.

[**0037**] **FIG. 10A** depicts part of a steering sensor indicated generally at **1000** in a steering output assembly indicated generally at **1100**. Output assembly **110** includes a housing **1102** containing steering motor **410**, a plurality of compound reduction gears indicated in phantom generally at **1102, 1104** driving a shaft **1110** (phantom) keyed with a rotary output member **1120** on the housing **1102**. Output member **1120** rotates in an arc, moving from side to side a wire member **1130** defining a pair of steering arms **1132, 1134** operably coupled with separate ones of the pair of front wheels **28** of the vehicle **20** to pivot those wheels side to side about vertical axes in a conventional manner to steer wheel **20**. **FIG. 10B** shows the output assembly **1100** with the gears **1102, 1104** and a top cover carrying the rotary output member **1120** removed. The left side of assembly **1100** includes steering sensor **1000** while the right side includes steering motor **420**. Sensor **1000** includes a stationary member or portion, which is indicated generally at **1010** and seen separately in **FIG. 11**, and a rotary member or rotating portion indicated generally at **1050**. The rotary member **1050** includes a plurality of connected concentric ring portions **1052, 1054, 1056** each containing one or more dimples **1052***a,* **1054***a* and **1056***a,* **1056***b* for the innermost ring. These dimples ride over the upper surface of the stationary portion **1010**. Referring to **FIG. 11**, the stationary portion **1010** includes a circuit board **1012** on which are mounted three electrically conductive, generally concentric tracks **1020, 1030** and **1040**. Each track includes an output terminal **1022, 1032, 1042**, respectively on one edge of the board **1012**. These three terminals connect via a suitable electrical connection (e.g. connector **860** in **FIG. 8B**) to microprocessor **8U2**. Each track **1020, 1030, 1040** is continuous around a central opening **1014** in the circuit board **1012** through which the output shaft **1110** extends. Rotating portion **1050** is keyed with shaft **1110** to rotate with the shaft. Rotating portion **1050** is a continuous piece of electrically conductive material such as metal and electrically couples one or more of the two outer tracks **1020** and **1030** with the innermost track **1040**. A high level voltage is applied by the microprocessor **8U2** through the connecter **860** to the terminals **1022** and **1032**. Terminal **1042** is connected to common or ground. The contacting dimples **1056***a* **1056***b* are in constant contact with the ring portion **1044** of innermost track **1040**. In contrast, dimples **1054**a of ring portion **1054** only contact wiper portions **1034** and **1036** of central track **1030** at certain angular positions of rotating portion **1050**. Similarly, dimples **1052***a* of ring **1052** only contact wiper portions **1024** and **1026** of the outermost track **1020**.

[**0038**] Referring to **FIG. 1**, dimples **1052***a,* **1054***a,* **1056***a,* **1046***b* of rotating contact member **1050** come in contact with the tracks **1020, 1030, 1040** in five different steering positions (far left indicated at **1060**, near left **1062**, center **1064**, near right **1066**, far right **1068**) on printed circuit board **1010** as member **1050** turns clockwise from far left to far right. When the rotating member **1050** is turned fully left or right, dimples **1052***a,* **1054***a* loose contact with tracks **1020, 1030** and logic bits "1,1" are outputted from electrical contacts **1022, 1032**. When the rotating member **1050** is turned clockwise from far left to left of center **1062**, logic bits "0,1" are outputted from electrical contacts **1022, 1032**. When the rotating member is in the center position **1064**, logic bits "0,0" are outputted from electrical contacts **1022, 1032**. When the rotating member is turned to the right of center but not fully right, logic bits "1,0" are outputted from electrical contacts **1022, 1032**. When fully right, logic bits "1, 1" are again output from contacts **1022, 1032**.

[**0039**] The states of electrical contacts **1022, 1032** are monitored by processor **8U2** and the speed of steering motor **410** is preferably controlled based on the outputted logic bits (i, i) which indicate the position of the front wheels **28**. Normally the steering motor **410** operates at top speed (100%). However, with feedback provided by sensor **1000**, the motor **410** can be operated to prevent overshoot. **FIG. 3** shows a trapezoidal velocity profile of speed versus time for the steering function of a toy vehicle **20** according to a preferred embodiment of the present invention. Steering motor **410** may be controlled like propulsion motor **420** by a PWM duty cycle to prevent overshoot of the steering system. For example, the steering motor **410** may be driven by microprocessor **8U2** (or **4U1**) at a higher duty cycle when going from a left or right turn to a turn in the other direction (e.g., from far left to far right) and at a lesser duty cycle when going from a center position to right or left and vice versa. When logic bits "0, 1" are detected as the rotating member **1120** turns from center position (0, 0) to the left and passes the near left wipers **1024, 1026**, or when logic bits "1, 0" are detected as the output member **1120** and rotary member **1050** turn to the right and pass the near right wipers **1034, 1036**, the rate of the steering motor and front wheel rotation is reduced to 50% to avoid overshooting its destination (far left or far right). Preferably too, the speed of the propulsion motor **420** can further be reduced automatically by the processor **8U2** when the processor **8U2** detects that a turn of the toy vehicle **20** is in progress to automatically slow the vehicle to a speed less than maximum while making the turn.

[**0040**] With a start and end point considered in a closed loop system, speed of the steering motor **410** in the toy vehicle **20** can be varied so that steering follows a trapezoidal profile as shown in **FIG. 3, i**.e. start from zero and reach a maximum turning rate, and then slowed to reduce its rate of rotation so that steering system momentum is dissipated and the steering system does not overshoot its target. When the command to steer to a new position is given, firmware operating in conjunction with microprocessor **8U2** (or **4U1**) will identify the current steering position and move at a higher rate and duty cycle (e.g., 100% duty cycle) when the commanded steering position is more than one steering position away from (i.e., other than adjacent to) its current position. For example, in going from a left turn to a right turn through consecutive outputs (1, 1), (0, 1), (1, 1), (1, 0) to (1, 1), the motor **410** may be driven at high speed (100% duty cycle) until center position (0, 0) or near right (1, 0) is encountered and the motor **410** then driven at a lower speed (e.g., 50% duty cycle) until far right (1, 1) is sensed.

[**0041**] Steering control can be further refined if the steering function is spring centered, i.e. a single torsion spring or pair of compression or tension springs (none depicted) used to drive the rotary output member **1120** to the straight forward position. Then the microprocessor **8U2** (or **4U1**) can be configured by programming to account for action of the spring(s). For example, turning from left to right, the microprocessor **8U2** may drive at high level and low level in moving more than one steering position (e.g. left-right) or only one steering position (e.g. center left/right), respectively, from the present position and at different speeds if moving with or against a spring. For example, movement left to right or vice versa can begin at full speed (100% duty cycle) and transfer to first low speed (e.g. 50% duty cycle) from the center position (0, 0) to the far right position to drive against the centering spring in the latter part of the movement. In going from right or left to center with spring assistance, the motor **410** is operated at a second, lower speed (e.g., 37.5% duty cycle), whereas, while going from center to left or right against a spring, the motor **410** is operated at the first low speed (e.g., 50%).

[**0042**] A spring loaded steering function of the toy vehicle **20** may also incorporate a target pad timeout period which monitors the time it takes for the sensor **1000** to reach a particular steering position (center, near left, far left, near right, far right). If the position is not reached within a predetermined period of time, the power to the motor **410** is turned off and the spring(s) will return the steering output number **1120** to the center position. If the steering position does not return to the center position, the microprocessor **8U2** (or **4U1**) is alerted that the steering is misaligned and electromechanically re-centers the steering.

[**0043**] Preferred transmitter code used in a remote control/transmitter **100** operating in accordance with the present invention is located on pages A-1 through A-53 of the attached Appendix incorporated by reference herein. Preferred receiver code used in a toy vehicle **20** operating in accordance with the present invention is located on pages A-54 through A-77 of the Appendix.

[**0044**] In addition to duty cycle control in the vehicle **20**, speed control of the vehicle **20** could be performed by the remote control/transmitter **100** by duty cycle transmission of a propulsion or steering signal (i.e. transmit the signal(s) several times followed by a period with no signal) or by varying the rate at which the propulsion signal is transmitted (e.g., every 10, 15 or 20 millisecond). Of course, the microprocessor of the toy vehicle **20** would also have to be appropriately configured to operate with such a duty cycle arrangement.

[**0045**] It will be appreciated by those skilled in the art that changes could be made to the embodiments described above without departing from the broad inventive concept thereof. It is understood, therefore, that this invention is not limited to the particular embodiments disclosed, but it is intended to cover modifications within the spirit and scope of the present invention.

APPENDIX

Transmitter Code

```
        .LINKLIST
        .SYMBOLS
        .CODE

; /***** System parameters *****

SystemClock:            EQU     2000000

SPC41A:                 EQU     1       ; select body (hardware.inh)

; /***** sound details (Ver3.42a or later)
ADPCM_TABLE_65:         EQU     1       ;If use ADPCM65 or later
_ADPCM_H_:              EQU     1       ;If no limit

 .Include       Hardware.Inh


; ***** Addresses SunPlus forgot *****

P_MultiPhase            EQU     $37     ; register that controls Multi Phase
settings on 81A

; ***** CONSTANTS/DEFINES *****
; sound stuff
D_RampDownValue:        EQU     00H     ;If CurrentDAC->00H, PWM->80H
D_MaxWord:              EQU     16      ;number of speech pieces
D_MaxMelody:            EQU     1       ;number of melodies
D_MaxRhythm:            EQU     0       ;number of rhythms
D_SamplePreload:        EQU     00H     ;should be 6 for 8KHz or 0 for 6 Khz

; my sounds
;D_Snd_Accel            EQU     0
D_Snd_Braking           EQU     1
D_Snd_Chirp             EQU     2
D_Snd_Dnshift           EQU     3
D_Snd_Eng_Strt          EQU     4
D_Snd_Gear              EQU     5
D_Snd_Grind             EQU     8
D_Snd_Horn              EQU     9
D_Snd_Idle              EQU     10
D_Snd_Peelout           EQU     11
D_Snd_Squeel            EQU     12
D_Snd_Upshift           EQU     13
D_Snd_Whine_C           EQU     14
D_Snd_Whine_R           EQU     15
D_Snd_None              EQU     ffh

; within loop timiers
; Sound Service Timers
D_TmBH_SS0              EQU     00h
```

```
                                          Shft96Tx.ASM
D_TmBL_SS0                    EQU      96h       ; 75 uS

D_TmBH_SS1                    EQU      01h
D_TmBL_SS1                    EQU      d0h       ; 232 uS

D_TmBH_SS2                    EQU      02h
D_TmBL_SS2                    EQU      f6h       ; 379uS

D_TmBH_SS3                    EQU      04h
D_TmBL_SS3                    EQU      4ah       ; 549uS

D_TmBH_SS4                    EQU      05h
D_TmBL_SS4                    EQU      82h       ; 705uS

D_TmBH_SS5                    EQU      06h
D_TmBL_SS5                    EQU      bch       ; 862uS

D_TmBH_SS6                    EQU      07h
D_TmBL_SS6                    EQU      f8h       ; 1020uS

D_TmBH_SS7                    EQU      09h       ; 1177uS
D_TmBL_SS7                    EQU      32h

D_TmBH_SS8                    EQU      0Fh       ; only for standby horn sound
D_TmBL_SS8                    EQU      D5H       ; 2 ms

D_TmBH_Tx0                    EQU      09h
D_TmBL_Tx0                    EQU      D8h       ; 1260 uS

D_TmBH_Tx1                    EQU      02h
D_TmBL_Tx1                    EQU      76h       ; 315 uS

D_TmBH_Tx2                    EQU      04h       ; 630 uS
D_TmBL_Tx2                    EQU      ech

D_TmBH_Tx3                    EQU      07h       ; 945 uS
D_TmBL_Tx3                    EQU      62h


; large timers-small timer ticks each loop. (1260 uS). Large timer ticks every .32
S.
D_Small_Squeel_Timer_Preload        EQU     #8ch
D_Large_Squeel_Timer_Preload        EQU     #01h      ; 018ch =0.5 seconds

; for prototyping a short medium or long shift time can be selected
D_Small_Shift_Timer_Preload         EQU     #33h      ; 0633=2.0 seconds
D_Large_Shift_Timer_Preload         EQU     #06h      ;

D_Small_Fwd_Release_Timer_Preload           EQU     #19h
D_Large_Fwd_Release_Timer_Preload           EQU     #03h      ; 0319h = 1 second

D_Small_Sound_Check_Timer_Preload           EQU     #8ch
D_Large_Sound_Check_Timer_Preload           EQU     #03h

D_Small_Idle_Timer_Preload          EQU     #80h
D_Large_Idle_Timer_Preload          EQU     #2eh      ; 2e80h=15 seconds

D_Small_Chirp_Timer_Preload         EQU     #c6h      00c6h =0.25 seconds

D_Small_LED_Timer_Preload           EQU     #ffh      ; ffh =.3 seconds

D_Peelout_Time                      EQU     #28h      ; when idle timer has gotten here
```

Shft96Tx.ASM

```
                                         ; 2eh-28h=2 seconds

; Timers which run off of interrupts (interrupts used only during mode selection)
; both run off of clock/65536 interrupt.  For 2 Mhz clock this is 31 hz
D_Mode_Check_Timeout      EQU      #93      ; 93=3 seconds
D_Mode_Select_Time        EQU      #62      ; 62=seconds
D_LED_Flash_Timer         EQU      #8
; Inputs

; buttons
D_Pin_Fwd:                EQU      00001000b        ; PortD bit of motor pin output
D_Pin_Rev:                EQU      00000100b        ;
D_Pin_Left:               EQU      00000010b        ;
D_Pin_Right:              EQU      00000001b        ;

; outputs

D_Pin_Tx:                 EQU      00000010b        ; Port C

D_Pin_LED:                EQU      00000001b        ;
D_Pin_Tx_Enable           EQU      00000100b


; PACKET BITS
D_Fwd_Bits                EQU      00110000b        ; 2 bit pwm level
D_Rev_Bits                EQU      00001100b        ; 2 bit pwm level
D_Left_Bit                EQU      00000010b
D_Right_Bit               EQU      00000001b
D_Turns_Bit               EQU      00000011b

D_PWM_Lo                  EQU      00001000b
D_PWM_Med                 EQU      00010000b
D_PWM_Hi                  EQU      00011000b

D_TX_Flag                 EQU      01111100b        ; first six bits are the flag
; mode selection
D_Mode_2_Command          EQU      00000111b        ; fw  pull down
D_Mode_3_Command          EQU      00001011b        ; rev  pull down

; ***** VARIABLES *****
        .PAGE0
        .ORG    D_RamTop

; variables used in all modes


                                        ; interrupts related
R_IntFlags:      DS        1
R_IntTemps:      DS        1
R_TempA:         DS        1
R_TempX:         DS        1

                                        ; sound related
R_SongNo:        DS        1
R_Volume:        DS        1
R_Temp1:         DS        1

R_Next_Sound     DS        1            ; sound to be played next
R_Current_Sound  DS        1            ; sound being played
R_Sounds_Array   DS        17
R_Sound_Interrupt          DS      1
```

Shft96Tx.ASM

```
R_Sound_Repeat  DS          1

                                    ; mode related
R_Mode                      DS      1           ; 3 modes- shifting, sounds, no sounds
R_Mode_Timer                DS      1
R_Mode_Check_Timer          DS      1
R_Command_To_Check          DS      1
R_Mode_To_Check             DS      1

R_State                     DS      1

                                    ; tx related
R_Tx_Flag_And_Ck            DS      1
R_Tx_Commands               DS      1

R_Tx_Data_Current           DS      1
R_Current_Tx_Byte_Num       DS      1
R_Tx_Bit_Index              DS      1

R_Second_Tx_Bit_Half        DS      1
R_First_Tx_Bit_Half         DS      1
R_Bit_Half                  DS      1


; Variables Specific to Mode 1
R_Gear                      DS      1

R_Shift_Legal               DS      1

; Variables Specific to Mode 2

R_Turning                   DS      1
R_Dir                       DS      1
R_Peelout_Enable            DS      1
R_Large_Shift_Timer         DS      1
R_Small_Shift_Timer         DS      1
R_Large_Gear_Timer          DS      1
R_Small_Gear_Timer          DS      1
R_Large_Idle_Timer          DS      1
R_Small_Idle_Timer          DS      1
R_Small_Chirp_Timer         DS      1
R_Large_Squeel_Timer        DS      1
R_Small_Squeel_Timer        DS      1
R_Small_LED_Timer           DS      1
R_Small_Fwd_Release_Timer           DS      1
R_Large_Fwd_Release_Timer           DS      1

R_Fwd_Ack_Ok                DS      1
R_Rev_Ack_Ok                DS      1

R_Gear_Bits                 DS      1
R_First_Start               DS      1
R_SS_Time_H                 DS      8
R_SS_Time_L                 DS      8
R_Tx_Time_H                 DS      4
R_Tx_Time_L                 DS      4
R_Wait_Time_H               DS      1
R_Wait_Time_L               DS      1
R_Horn_Plays                DS      1
R_Small_Sound_Check_Timer           DS      1
R_Large_Sound_Check_Timer           DS      1
R_Sound_Check_Delay_Complete        DS      1
```

Shft96Tx.ASM

```
R_Peeled_Out            DS      1
R_Shifted               DS      1
R_Sleepy_Counter        DS      1
R_Standby_Counter       DS      1

R_MS_Timer_Hi           DS      1
R_MS_Timer_Lo           DS      1

R_Sound_Wait_Index      DS      1

        .PAGE0
        .Include        Channel.Inh


        .CODE
        .ORG    000H
        DB      FFH
        .ORG    600H

; ********************* Begin Main Code *****************

; V_ is by convention a vector.  The reset vector is where the code goes when the
micro is reset
V_Reset:

        SEI
        ;(From Demo Code)
        LDX     #FFH                    ; load ff into the x reg (H means hex)
        TXS                             ; transfer x reg contents to stack

 ;wake from sleep stuff
        ;lda    #C0h                    ; turn off and clear all interrupt
        ;sta    $0D                     ; disable watchdog
        ;lda    $08                     ; Acc <- Wake up status
        ;ldx    #00
        ;stx    $08
        ;and    #01
        ;beq    L_Init_Variables  ; if starting from power up
        ;jmp    L_Wake_Up         ; if starting from sleep

                ; ***** Initialize variables
L_Init_Variables:

        lda     #0
        sta     R_IntFlags
        sta     R_IntTemps
        sta     R_TempA
        sta     R_TempX
        sta     R_SongNo
        sta     R_Volume
        sta     R_Temp1
        sta     R_Sounds_Array
        sta     R_Sound_Interrupt
        sta     R_Sound_Repeat
        sta     R_Mode
        sta     R_Mode_Timer
        sta     R_Mode_Check_Timer
        sta     R_Command_To_Check
        sta     R_Mode_To_Check
        sta     R_State
        sta     R_Tx_Flag_And_Ck
```

Shft96Tx.ASM

```
sta     R_Tx_Commands
sta     R_Tx_Data_Current
sta     R_Current_Tx_Byte_Num
sta     R_Tx_Bit_Index
sta     R_Second_Tx_Bit_Half
sta     R_First_Tx_Bit_Half
sta     R_Bit_Half
sta     R_Gear
sta     R_Shift_Legal
sta     R_Turning
sta     R_Dir
sta     R_Horn_Plays

sta     R_Large_Shift_Timer
sta     R_Small_Shift_Timer

sta     R_Large_Gear_Timer
sta     R_Small_Gear_Timer
sta     R_Large_Idle_Timer
sta     R_Small_Idle_Timer
sta     R_Small_Chirp_Timer
sta     R_Large_Squeel_Timer
sta     R_Small_Squeel_Timer
sta     R_Gear_Bits
sta     R_First_Start
sta     R_Shifted
sta     R_Peeled_Out
sta     R_Sleepy_Counter
sta     R_Standby_Counter
sta     R_Fwd_Ack_Ok
sta     R_Rev_Ack_Ok


lda     #1
sta     R_Peelout_Enable

ldx     #0
lda     #D_TmBH_SS0
sta     R_SS_Time_H,X
lda     #D_TmBL_SS0
sta     R_SS_Time_L,X

ldx     #1
lda     #D_TmBH_SS1
sta     R_SS_Time_H,X
lda     #D_TmBL_SS1
sta     R_SS_Time_L,X

ldx     #2
lda     #D_TmBH_SS2
sta     R_SS_Time_H,X
lda     #D_TmBL_SS2
sta     R_SS_Time_L,X

ldx     #3
lda     #D_TmBH_SS3
sta     R_SS_Time_H,X
lda     #D_TmBL_SS3
sta     R_SS_Time_L,X

ldx     #4
lda     #D_TmBH_SS4
sta     R_SS_Time_H,X
```

Shft96Tx.ASM

```
        lda     #D_TmBL_SS4
        sta     R_SS_Time_L,X

        ldx     #5
        lda     #D_TmBH_SS5
        sta     R_SS_Time_H,X
        lda     #D_TmBL_SS5
        sta     R_SS_Time_L,X

        ldx     #6
        lda     #D_TmBH_SS6
        sta     R_SS_Time_H,X
        lda     #D_TmBL_SS6
        sta     R_SS_Time_L,X

        ldx     #7
        lda     #D_TmBH_SS7
        sta     R_SS_Time_H,X
        lda     #D_TmBL_SS7
        sta     R_SS_Time_L,X


        ldx     #0
        lda     #D_TmBH_Tx0
        sta     R_Tx_Time_H,X
        lda     #D_TmBL_Tx0
        sta     R_Tx_Time_L,X

        ldx     #1
        lda     #D_TmBH_Tx1
        sta     R_Tx_Time_H,X
        lda     #D_TmBL_Tx1
        sta     R_Tx_Time_L,X

        ldx     #2
        lda     #D_TmBH_Tx2
        sta     R_Tx_Time_H,X
        lda     #D_TmBL_Tx2
        sta     R_Tx_Time_L,X

        ldx     #3
        lda     #D_TmBH_Tx3
        sta     R_Tx_Time_H,X
        lda     #D_TmBL_Tx3
        sta     R_Tx_Time_L,X

        lda     #D_Snd_None
        sta     R_Current_Sound
        sta     R_Next_Sound



; ***** External initializations
%ChannelPlayerInitial           ; in Channel.inh

        lda     P_Stop          ; set volume
        and     #%11110000
        nop
        nop
        nop
        nop
        sta     P_Stop
```

Shft96Tx.ASM

```
        LDA     #00000000b              ; all off (low)
        STA     P_PortC                 ;

        lda     #0
        sta     P_PortB
        sta     P_PortA

; ***** Port configuration

        LDA     #10111111b              ; D-C-B-A, high-low, 1=output
        STA     P_PortIO_Ctrl
        LDA     #00000000b              ; outputs buffer;   DL pull down
        STA     P_Port_Attrib


        LDA     #00000010b              ;
        STA     P_MultiPhase            ; turn off multi-phase on A2, but set 1/3
duty
                                        ; in case it does turn on for diagnosis

        ; ***** Configure interrupts
        lda     #%11000010              ; disable watchdog
                                        ; disable nmi
                                        ; enable TimerA interrupt
                                        ; enable TimerB interrupt
                                        ; disable 4 khz interrupt
                                        ; disable 500 Hz and
                                        ; enable 62.5 Hz interrupts
                                        ; disable external interupt
        STA     P_Ints                  ; store interrupt settings
        STA     R_IntFlags              ; store interrupt settings here, too

        SEI                             ; disable interrupts
; ***** Preload Timers

        LDA     #00h                    ;
        STA     P_TmAL                  ; preload: D58h = 315 us
        LDA     #00h                    ;
        STA     P_TmAH                  ; above and mode bits

        LDA     #00h                    ;
        STA     P_TmBL                  ;
        LDA     #00h                    ;
        STA     P_TmBH                  ;


L_Main:

        JSR     F_Get_Mode
        jsr     F_Init_Mode

L_Wake_Up:

        lda     P_PortC                 ; enable radio
        and     #.NOT.D_Pin_Tx_Enable
        sta     P_PortC


L_Main_Loop:

        sei
```

Shft96Tx.ASM

```
; tx line should be serviced every 315 us, sound every 157 uS

        LDA     #00h     ; preload timers
        STA     P_TmBL                  ;
        LDA     #00h                    ;
        STA     P_TmBH                  ;

        jsr     F_Check_Time_To_Standby

L_Set_Tx_Line:

        JSR     F_Set_Tx_Line           ;

        JSR     F_IntCh1Service
        JSR     F_Decide_Packet
        ldx     #1
        jsr     F_Wait_Sound_Service
        JSR     F_IntCh1Service

        ldx     #1
        jsr     F_Wait_Tx_Line

        JSR     F_Set_Tx_Line           ;
        ldx     #2
        jsr     F_Wait_Sound_Service

        JSR     F_IntCh1Service

        jsr     F_Determine_State                ; state determines packet, sounds

        ldx     #3
        jsr     F_Wait_Sound_Service

        JSR     F_IntCh1Service

        ldx     #2
        jsr     F_Wait_Tx_Line
        JSR     F_Set_Tx_Line           ;
        ldx     #4
        jsr     F_Wait_Sound_Service

        JSR     F_IntCh1Service
        JSR     F_ServiceChannelPlayer
        JSR     F_Decide_Sounds
        ldx     #5
        jsr     F_Wait_Sound_Service

        JSR     F_IntCh1Service

        ldx     #3
        jsr     F_Wait_Tx_Line
        JSR     F_Set_Tx_Line           ;
        ldx     #6
        jsr     F_Wait_Sound_Service

        JSR     F_IntCh1Service
```

Shft96Tx.ASM

```
        JSR     F_Play_Sounds
        jsr     F_Control_LED
        ldx     #7
        jsr     F_Wait_Sound_Service

        JSR     F_IntCh1Service

        ldx     #0
        jsr     F_Wait_Tx_Line
        JMP     L_Main_Loop

;*********************Functions**********************************

; as an alternate to sleeping, the micro remains on, but shuts off radio
transmission
; and the DAC when nothing is doing

F_Check_Time_To_Standby:

        lda     R_Mode
        cmp     #3
        beq     L_CSt_Mode_3

        ; in modes 1 or 2 (these modes have sound)
        %TestSpeechCh1          ; sets carry if playing
        bcs     L_CSt_Clear_Standby_Counter


L_CSt_Mode_3:

        lda     P_PortD
        and     #0Fh
        cmp     #0Fh
        bne     L_CSt_Clear_Standby_Counter     ; button pressed

        inc     R_Standby_Counter               ; getting sleepier

        lda     R_Standby_Counter
        cmp     #9
        bne     L_CSt_Done


L_CSt_Prepare_To_Standby:

        lda     #0                      ; clear standby counter
        sta     R_Standby_Counter

        lda     P_PortC                 ; turn off radio transmission
        and     #.NOT.D_Pin_Tx_Enable
        sta     P_PortC

        lda     #0                      ;turn off dac
        sta     R_DacCh1
        sta     P_DacCh1

        lda     #0
        sta     R_Sound_Wait_Index

        jsr     F_Standby


L_CSt_Clear_Standby_Counter:
```

Shft96Tx.ASM

```
        lda     #0
        sta     R_Standby_Counter

L_CSt_Done:
        rts

; F_Standby
        ; wait until a button is hit.  In the meantime play a sound occasionally
        ; to suggest to user that he turn off device.
F_Standby:

L_S_Start:

        lda     #0                      ; clear timer A
        sta     P_TmBH
        sta     P_TmBL
        sta     R_MS_Timer_Hi           ; clear
        sta     R_MS_Timer_Lo
        sta     R_Sound_Wait_Index

L_S_Loop:                                       ; checks to see if any button hit or

        jsr     F_ServiceChannelPlayer  ; if waiting time has expired.

        ldx     R_Sound_Wait_Index
        jsr     F_Wait_Sound_Service
        jsr     F_IntChlService

        lda     P_PortD                         ; check button
        and     #0fh
        cmp     #0fh
        bne     L_S_Done                        ; button hit

                ; no button hit
        inc     R_Sound_Wait_Index
        lda     R_Sound_Wait_Index
        cmp     #09
        bne     L_S_Loop

                ; finished hw timer loop
        lda     #0              ; clear hw timers for next time
        sta     P_TmBH
        sta     P_TmBL

        lda     #0              ; reset index for next time
        sta     R_Sound_Wait_Index

        inc     R_MS_Timer_Lo
        lda     R_MS_Timer_Lo
        cmp     #ffh
        beq     L_S_Inc_MS_Timer_Hi
;
        jmp     L_S_Loop        ; MS_Timer_Lo not at max, keep looping

L_S_Inc_MS_Timer_Hi:

        lda     #0
        sta     R_MS_Timer_Lo

        inc     R_MS_Timer_Hi
        lda     R_MS_Timer_Hi
        cmp     #ffh
```

Shft96Tx.ASM

```
        bne     L_S_Loop        ; MS_Timer_Hi not at max, keep looping

L_S_Set_Sound:

        lda     #0
        sta     R_Sound_Interrupt
        lda     #0
        sta     R_Sound_Repeat

L_Playing_Sound:

        lda     #3
        sta     R_Current_Sound
        lda     #9
        sta     R_Next_Sound

        jsr     F_Play_Sounds

        jmp     L_S_Start

L_S_Done:
        rts

;/**********************************************
F_Blip_B0:
        lda     P_PortB
        ora     #00000001b
        sta     P_PortB

        lda     P_PortB
        and     #11111110b
        sta     P_PortB

        rts

F_Blip_B1:
        lda     P_PortB
        ora     #00000010b
        sta     P_PortB

        lda     P_PortB
        and     #11111101b
        sta     P_PortB

        rts



F_Get_Mode:

        lda     #1
        sta     R_Mode                  ; set mode 1 as new default

        sei                             ; disable ints momentarily since were
fooling
                                        ; with a variable that's changed in the int
        lda     #0
        sta     R_Mode_Timer
        cli

L_GM_Check_Modes_Loop:
```

Shft96Tx.ASM

```
        lda     P_PortD
        and     #0fh
        cmp     #0fh
        beq     L_GM_CK_LED_Off

        lda     P_PortC
        ora     #D_Pin_LED
        sta     P_PortC
        jmp     L_GM_Check_Timer

L_GM_CK_LED_Off:
        lda     P_PortC
        and     #.NOT.D_Pin_LED
        sta     P_PortC

L_GM_Check_Timer:
        sei
        lda     R_Mode_Timer
        cli
        cmp     #D_Mode_Check_Timeout                   ; 3 seconds
        bcs     L_GM_Store_Mode_1; use default

        lda     P_PortD
        and     #0fh
        cmp     #D_Mode_2_Command
        bne     L_GM_Check_Mode_3

;mode 2 selected initially--keys need to be held

        lda     P_PortC
        ora     #D_Pin_LED
        sta     P_PortC

        lda     #D_Mode_2_Command
        sta     R_Command_To_Check
        lda     #2
        sta     R_Mode_To_Check
        jmp     L_GM_Wait_Modes

L_GM_Check_Mode_3:

        cmp     #D_Mode_3_Command
        bne     L_GM_Check_Modes_Loop

; mode 3 selected--keys need to be held

        lda     P_PortC
        ora     #D_Pin_LED
        sta     P_PortC

        lda     #D_Mode_3_Command
        sta     R_Command_To_Check
        lda     #3
        sta     R_Mode_To_Check
        jmp     L_GM_Wait_Modes


; wait to see if the buttons are pressed long enough for either mode 2 or 3 to be
selected
L_GM_Wait_Modes:

        sei                             ; start timer
```

Shft96Tx.ASM

```
        lda     #0
        sta     R_Mode_Check_Timer
        cli

L_GM_Wait_Mode_Loop:

        lda     P_PortD                 ; see if buttons still pressed
        and     #0fh
        cmp     R_Command_To_Check
        beq     L_GM_Still_Pressed

        lda     P_PortC                 ; turn led off
        and     #.NOT.D_Pin_LED
        sta     P_PortC

        jmp     L_GM_Check_Modes_Loop   ; button no longer pressed

L_GM_Still_Pressed:
        lda     R_Mode_Check_Timer
        cli
        cmp     #D_Mode_Select_Time
        bcc     L_GM_Wait_Mode_Loop
        jmp     L_GM_Store_Mode

L_GM_Store_Mode_1:
        lda     #1
        sta     R_Mode_To_Check

        ; mode selected
L_GM_Store_Mode:

        lda     R_Mode_To_Check
        sta     R_Mode
        cmp     #1
        beq     L_GM_Setup_Shift_Sound

        cmp     #2
        beq     L_GM_Setup_Horn_Sound

        jmp     L_GM_Wait_For_Joystick_Release

L_GM_Setup_Shift_Sound:

        lda     #D_Snd_Upshift
        jmp     L_GM_Setup_Sound

L_GM_Setup_Horn_Sound:

        lda     #D_Snd_Horn
        jmp     L_GM_Setup_Sound

L_GM_Setup_Sound:

        sta     R_Current_Sound
        sta     R_Next_Sound

        LDX     #D_SamplePreload        ; the setting for sample frequency
        JSR     F_PlaySpeechCh1         ; play

L_GM_Wait_For_Joystick_Release:

        sei
        lda     #0
```

Shft96Tx.ASM

```
        sta     R_Mode_Timer
        cli


L_GM_Wait_For_Release:

        sei
        lda     R_Mode_Timer
        cmp     #D_LED_Flash_Timer
        bcc     L_GM_Check_Release

; flash led
        lda     P_PortC
        eor     #D_Pin_LED
        sta     P_PortC
        lda     #0
        sta     R_Mode_Timer
        cli

L_GM_Check_Release:
        cli

        lda     P_PortD
        and     #0fh
        cmp     #0fh
        bne     L_GM_Wait_For_Release

        jmp     L_GM_Done


L_GM_Done:
        sei                             ; disable interrupts
        lda     #%11000000              ; disable clk/65536 interrupt
        STA     P_Ints                  ; store interrupt settings
        STA     R_IntFlags              ; store interrupt settings here, too
        rts

F_Init_Mode:

        lda     R_Mode

        cmp     #1
        beq     L_IM_1

        cmp     #2
        beq     L_IM_2

        jmp     L_IM_3

L_IM_1:

        jsr     F_Load_Shared_Sounds    ; several sounds are shared by modes 1 and 2


        ; setup the sound array

L_LD_SND:
        ldx     #4                      ; acceleration mode no longer exists
        lda     #D_Snd_Gear
        sta     R_Sounds_Array,X

        ldx     #5
        lda     #D_Snd_Gear
```

Shft96Tx.ASM

```
        sta     R_Sounds_Array,X

        ldx     #6
        lda     #D_Snd_Gear
        sta     R_Sounds_Array,X

        ldx     #7
        lda     #D_Snd_Gear
        sta     R_Sounds_Array,X

        ldx     #8
        lda     #D_Snd_Horn
        sta     R_Sounds_Array,X

        ldx     #9
        lda     #D_Snd_Gear
        sta     R_Sounds_Array,X

        ldx     #11
        lda     #D_Snd_Braking
        sta     R_Sounds_Array,X

        ldx     #12
        lda     #D_Snd_Grind
        sta     R_Sounds_Array,X

        rts

L_IM_2:
        ;setup the sound array
        jsr     F_Load_Shared_Sounds    ; several sounds are shared by modes 1 and 2

        ldx     #4
        lda     #D_Snd_Gear
        sta     R_Sounds_Array,X

        ldx     #5
        lda     #D_Snd_Braking
        sta     R_Sounds_Array,X

        ldx     #6
        lda     #D_Snd_Chirp
        sta     R_Sounds_Array,X

        ldx     #9
        lda     #D_Snd_Horn
        sta     R_Sounds_Array,X

        ldx     #11
        lda     #D_Snd_Gear
        sta     R_Sounds_Array,X

        rts

L_IM_3:

        lda     #0                      ;turn off dac so we're not leaking current
        sta     R_DacCh1
        sta     P_DacCh1
        rts
```

Shft96Tx.ASM

```
F_Load_Shared_Sounds:

        ldx     #0
        lda     #D_Snd_None
        sta     R_Sounds_Array,X

        ldx     #1
        lda     #D_Snd_Eng_Strt
        sta     R_Sounds_Array,X

        ldx     #2
        lda     #D_Snd_Idle
        sta     R_Sounds_Array,X

        ldx     #3
        lda     #D_Snd_Peelout
        sta     R_Sounds_Array,X

        ldx     #10                     ; shift plays occasional when in mode2,
state 4
        lda     #D_Snd_Upshift
        sta     R_Sounds_Array,X

        ldx     #13
        lda     #D_Snd_Squeel
        sta     R_Sounds_Array,X

        ldx     #14
        lda     #D_Snd_Peelout
        sta     R_Sounds_Array,X

        ldx     #0fh
        lda     #D_Snd_Gear
        sta     R_Sounds_Array,X

        ldx     #10h
        lda     #D_Snd_Squeel
        sta     R_Sounds_Array,X

        rts
;*********************************************************************************;

; set the tx line
; 2 Bytes are sent.  The first contains the flag and the checksum.  The second
; contains the actual data.
; every bit has 2 halves they will either be the same (0) or different (1).
; The first bit half will always be different than the previous second bit half
; (that is the tx line state always changes at the Bit boundry

;*********************************************************************************
F_Set_Tx_Line:

;       lda     P_PortB
;       ora     #02h
;       sta     P_PortB

;       lda     P_PortB
;       and     #.NOT.02h
;       sta     P_PortB

        lda     R_Bit_Half
        bne     L_ST_Second_Half
```

Shft96TX.ASM

```
; first bit half

        dec     R_Tx_Bit_Index

        ; check to see start of packet (for debugging)
        lda     R_Current_Tx_Byte_Num
        bne     L_ST_Check_Last_Second_Half

        ; doing flag/checksum byte currently
        lda     R_Tx_Bit_Index
        cmp     #7
        bne     L_ST_Check_Last_Second_Half


        ; blip A0 to show start of packet

        lda     P_PortA
        ora     #00000001b
        sta     P_PortA

        lda     P_PortA
        and     #11111110b
        sta     P_PortA

L_ST_Check_Last_Second_Half:

        lda     R_Second_Tx_Bit_Half
        bne     L_ST_Lo_First_Half      ; last was hi, we'll set next to lo s.t.
                                        ; there's change across the bit boundary

; last was no, we'll set to hi
        lda     P_PortC
        ora     #D_Pin_Tx
        sta     P_PortC

        lda     #1
        sta     R_First_Tx_Bit_Half
        jmp     L_ST_Done_First_Half

L_ST_Lo_First_Half:
        lda     P_PortC
        and     #.NOT.D_Pin_Tx
        sta     P_PortC

        lda     #0
        sta     R_First_Tx_Bit_Half

L_ST_Done_First_Half:

        lda     #1                      ; set flag to do second half of bit next
time f_ is called
        sta     R_Bit_Half
        jmp     L_ST_Done

; Second half of bit
L_ST_Second_Half:

        rol     R_Tx_Data_Current       ; shifts bit of interest into carry bit
        bcc     L_ST_Tx_Zero

; We want to transmit a "1"
        lda     R_First_Tx_Bit_Half
```

Shft96Tx.ASM

```
        beq     L_ST_Tx_Hi_Second_Half  ; since first half was lo, making second
half hi will make a "1"
        jmp     L_ST_Tx_Lo_Second_Half  ; since first half was hi, making second
half lo will make a "1"

; We want to transmit a "0"
L_ST_Tx_Zero:
        lda     R_First_Tx_Bit_Half
        beq     L_ST_Tx_Lo_Second_Half ; since first half was lo, making second half
lo will make a "0"

L_ST_Tx_Hi_Second_Half: ; since first half was hi, making second half hi will make a
"1"

        lda     P_PortC
        ora     #D_Pin_Tx
        sta     P_PortC

        lda     #1                              ;record
        sta     R_Second_Tx_Bit_Half

        jmp     L_ST_Check_Finished_Byte

L_ST_Tx_Lo_Second_Half:

        lda     P_PortC
        and     #.NOT.D_Pin_Tx
        sta     P_PortC

        lda     #0
        sta     R_Second_Tx_Bit_Half

        ; check to see if all bits of current byte have been sent
L_ST_Check_Finished_Byte:
        lda     R_Tx_Bit_Index
        bne     L_ST_Done_Second_Half


        ; index is zero...all bits have been sent in current data byte
        lda     #8
        sta     R_Tx_Bit_Index

        lda     R_Current_Tx_Byte_Num
        bne     L_ST_Set_Tx_Flag_And_Ck

        ; current byte is flag+ck --> set to commands

        lda     R_Tx_Commands
        sta     R_Tx_Data_Current
        lda     #1
        sta     R_Current_Tx_Byte_Num
        jmp     L_ST_Done_Second_Half

; long name, I know.  It sets the byte to the flag and ck byte
L_ST_Set_Tx_Flag_And_Ck:

        lda     R_Tx_Flag_And_Ck
        sta     R_Tx_Data_Current
        lda     #0
        sta     R_Current_Tx_Byte_Num


L_ST_Done_Second_Half:
```

Shft96Tx.ASM

```
        lda     #0
        sta     R_Bit_Half

L_ST_Done:
        rts

; ********************************************
; **** Determine State *******
;*********************************************

F_Determine_State:

        lda     R_Mode
        beq     L_Get_State_M0

        cmp     #1
        beq     L_Get_State_M1

        cmp     #2
        beq     L_Get_State_M2_Dummy

        jmp     L_Get_State_M3

L_Get_State_M2_Dummy:
        jmp     L_Get_State_M2


L_Get_State_M0:
        rts
;*************************************
;Determine state for Mode 1
;*************************************
L_Get_State_M1:

        lda     R_State
        beq     L_GS1_State_0_Dummy     ; same for modes 1 and 2

        cmp     #1
        beq     L_GS1_State_1_Dummy     ; same for modes 1 and 2

        cmp     #2
        beq     L_GS1_State_2_Dummy     ; same for modes 1 and 2

        cmp     #3
        beq     L_GS1_State_3

        cmp     #4
        beq     L_GS1_State_4_Dummy

        cmp     #5
        beq     L_GS1_State_5_Dummy

        cmp     #6
        beq     L_GS1_State_6_Dummy

        cmp     #7
        beq     L_GS1_State_7_Dummy

        cmp     #8
        beq     L_GS1_State_8_Dummy

        cmp     #9
        beq     L_GS1_State_9_Dummy
```

Shft96Tx.ASM

```
        cmp     #10
        beq     L_GS1_State_10_Dummy

        cmp     #11
        beq     L_GS1_State_11_Dummy

        cmp     #12
        beq     L_GS1_State_12_Dummy

        cmp     #13
        beq     L_GS1_State_13_Dummy

        cmp     #14
        beq     L_GS1_State_14_Dummy

        cmp     #15
        beq     L_GS1_State_15_Dummy

        jmp     L_GS_State_16


L_GS1_State_0_Dummy:
        jmp     L_GS_State_0

L_GS1_State_1_Dummy:
        jmp     L_GS_State_1

L_GS1_State_2_Dummy:
        jmp     L_GS_State_2

L_GS1_State_4_Dummy:
        jmp     L_GS1_State_4

L_GS1_State_5_Dummy:
        jmp     L_GS1_State_5

L_GS1_State_6_Dummy:
        jmp     L_GS1_State_6

L_GS1_State_7_Dummy:
        jmp     L_GS1_State_7

L_GS1_State_8_Dummy:
        jmp     L_GS1_State_8

L_GS1_State_9_Dummy:
        jmp     L_GS1_State_9

L_GS1_State_10_Dummy:
        jmp     L_GS1_State_10

L_GS1_State_11_Dummy:
        jmp     L_GS1_State_11

L_GS1_State_12_Dummy:
        jmp     L_GS1_State_12

L_GS1_State_13_Dummy:
        jmp     L_GS1_State_13

L_GS1_State_14_Dummy:
        jmp     L_GS_State_14
```

Shft96Tx.ASM

```
L_GS1_State_15_Dummy:
        jmp     L_GS_State_15

;******Model State 3***********************************************
;General:               Peelin' out
;Gear                   1
;Sound:                 Peel Out
;To Change State:       State 4 (Accel): Peel Out sound complete
;                       State 11(Braking): Rev Hit
;                       State 12(Grind Gears): trying to shift too soon (Fwd Only)
;                       State 13 (Squeeling): Turning for a long time
;

L_GS1_State_3:



        jsr     F_GS1_Check_Fw_Still_Pressed

        jsr     F_GS1_Check_Fwd_Ack_Ok

        lda     #1
        sta     R_Peeled_Out

        jsr     F_GS1_Check_Shift_Legal

        lda     R_Sound_Check_Delay_Complete
        bne     L_GS1_3_Sound_Check

        ; decrement timer
        dec     R_Small_Sound_Check_Timer
        lda     R_Small_Sound_Check_Timer
        beq     L_GS1_3_Dec_Big_Sound_Check_Timer      ; if small timer has run to
0
        jmp     L_GS1_3_Check_Squeel

L_GS1_3_Dec_Big_Sound_Check_Timer:
        lda     #ffh            ; reload small timer
        sta     R_Small_Sound_Check_Timer

        Dec     R_Large_Sound_Check_Timer
        LDA     R_Large_Sound_Check_Timer       ;
        Bne     L_GS1_3_Check_Squeel            ; not  yet


        lda     #1
        sta     R_Sound_Check_Delay_Complete

L_GS1_3_Sound_Check:

        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        bcs     L_GS1_3_Check_Squeel    ; still playing

        jsr     F_GS1_Set_State_5
        jmp     L_GS_Done


L_GS1_3_Check_Squeel:
        jsr     F_GS_Check_Squeel
```

Shft96Tx.ASM

```
        ;jsr    F_GS1_Check_Premature_Shift
        jsr     F_GS1_Check_Braking

        jmp     L_GS_Done

;******Model State 4**********************************************
;General:               Just jumps to state 5.  Used to be accellerating state, but
it was eliminated
;                       It is kept to avoid excessive rework in the code.  Also,
seems possible
;                       that client might ask for it back.
;Gear:                  -
;Sound:                 -
;To Change State:       State 5 (gear1). instantly, more or less


; going fwd. or reverse plays motor running sound, or occasionally gear shift-allow
car to move.

L_GS1_State_4:

;       %TestSpeechCh1
;       bcs     L_GS1_4_Done

        jsr     F_GS1_Set_State_5

L_GS1_4_Done:
        jmp     L_GS_Done

;******Model State 5**********************************************
;General:               Gear 1
;Movement:              Fwd possibly turning
;Sound:                 Gear 1
;To Change State:       State 8 (ready to shift) Timeout  (Fwd Only)
;                       State 11(Braking): Rev Hit if going fwd, or rev released if
going rev
;                       State 12(Grind Gears): trying to shift too soon
;                       State 13 (Squeeling): Turning for a long time (Fwd Only)


L_GS1_State_5:

        jsr     F_GS1_Check_Fw_Still_Pressed
        jsr     F_GS1_Check_Fwd_Ack_Ok
        jsr     F_GS_Check_Squeel
        jsr     F_GS1_Check_Premature_Shift
        jsr     F_GS1_Check_Shift_Legal
        jsr     F_GS1_Check_Braking

        jmp     L_GS_Done
;******Model State 6**********************************************
;General:               Gear 2
;Movement:              Fwd, possibly turning
;Sound:                 Gear 2
;To Change State:       State 8 (ready to shift) Timeout
;                       State 11(Braking): Rev Hit
;                       State 12 (Grind Gears): trying to shift too soon
;                       State 13 (Squeeling): Turning for a long time
;
L_GS1_State_6:
```

Shft96Tx.ASM

```
; traveling fwd currently

        jsr     F_GS1_Check_Fw_Still_Pressed
        jsr     F_GS1_Check_Fwd_Ack_Ok
        jsr     F_GS_Check_Squeel
        jsr     F_GS1_Check_Premature_Shift
        jsr     F_GS1_Check_Shift_Legal
        jsr     F_GS1_Check_Braking
        jmp     L_GS_Done
;*************************************************************

;******Model State 7*********************************************
;General:               Gear 3
;Movement:              Fwd, possibly turning
;Sound:                 Gear 3
;To Change State:
;                       State 11(Braking): Rev Hit
;                       State 13 (Squeeling): Turning for a long time
;
L_GS1_State_7

; traveling fwd currently
        jsr     F_GS1_Check_Fw_Still_Pressed
        jsr     F_GS_Check_Squeel
        jsr     F_GS1_Check_Braking
        jmp     L_GS_Done
;*************************************************************
;******Model State 8*********************************************
;General:               ready to shift ramp whining noise
;Movement:              Fwd
;Sound:                 ramp whining noise
;To Change State:
;                       State 9 (Ready to shift, const sound),
;                       State 10 (Shifting) Fwd hit legally
;                       State 11(Braking): Rev Hit or timeout
;                       State 13 (Squeeling): Turning for a long time

L_GS1_State_8:

        jsr     F_GS1_Check_Fw_Still_Pressed
        jsr     F_GS1_Check_Fwd_Ack_Ok

; a delay in the sound check timer is apparently necessary.  If I try to check the
; sound right away, it doesn't think a sound is playing.  So you wait some arbitrary
; amount of time (maybe half a second), then check to see if the sound is playing.
; at the time of writing this, I don't understand why this is necessary.

        lda     R_Sound_Check_Delay_Complete
        bne     L_GS1_8_Sound_Check

; decrement timer
        dec     R_Small_Sound_Check_Timer
        lda     R_Small_Sound_Check_Timer
        beq     L_GS1_8_Dec_Big_Sound_Check_Timer       ; if small timer has run to
0
        jmp     L_GS1_8_Check_Squeel

L_GS1_8_Dec_Big_Sound_Check_Timer:
        lda     #ffh            ; reload small timer
        sta     R_Small_Sound_Check_Timer

        Dec     R_Large_Sound_Check_Timer
        LDA     R_Large_Sound_Check_Timer       ;
```

Shft96Tx.ASM

```
        Bne     L_GS1_8_Check_Squeel            ; not  yet

        lda     #1
        sta     R_Sound_Check_Delay_Complete

L_GS1_8_Sound_Check:

        %TestSpeechCh1
        BCS     L_GS1_8_Check_Squeel

        ; done playing ramping whining noise
        jsr     F_GS1_Set_State_9
        jmp     L_GS_Done

L_GS1_8_Check_Squeel:

        jsr     F_GS_Check_Squeel
        jsr     F_GS1_Check_Shift
        jsr     F_GS1_Check_Braking
        jmp     L_GS_Done


;******Model State 9************************************************
;General:               ready to shift
;Movement:              Fwd
;Sound:                 constant whining noise
;To Change State:
;                       State 10 (Shifting) Fwd hit legally
;                       State 11(Braking): Rev Hit or timeout
;                       State 13 (Squeeling): Turning for a long time
;
L_GS1_State_9:

        jsr     F_GS1_Check_Fw_Still_Pressed
        jsr     F_GS1_Check_Fwd_Ack_Ok

        jsr     F_GS_Check_Squeel
        jsr     F_GS1_Check_Shift
        jsr     F_GS1_Check_Braking
        jmp     L_GS_Done

;******Model State 10***********************************************
;General:               shifting
;Movement:              Fwd
;Sound:                 shifting
;To Change State:
;                       State  6, 7 (Gear 2,3)  Sound finishes
;                       State 11(Braking): Rev Hit
;                       State 13 (Squeeling): Turning for a long time


L_GS1_State_10:

        jsr     F_GS1_Check_Fw_Still_Pressed
        jsr     F_GS1_Check_Fwd_Ack_Ok
        jsr     F_GS1_Check_Shift_Legal

        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        BCS     L_GS1_10_Check_Squeel           ; still playing shift

        lda     R_Gear
        cmp     #3                      ; shifting into gear 3
```

Shft96Tx.ASM

```
        beq     L_GS1_10_Set_State_7

        jsr     F_GS1_Set_State_6       ; shifting from gear 1 to 2
        jmp     L_GS_Done

L_GS1_10_Set_State_7:
        jsr     F_GS1_Set_State_7       ; shifting from gear 2 to 3
        jmp     L_GS_Done

L_GS1_10_Check_Squeel:
        jsr     F_GS_Check_Squeel
        jsr     F_GS1_Check_Braking
        jmp     L_GS_Done

;******Model State 11*********************************************
;General:               braking
;Movement:              none
;Sound:                 braking
;To Change State:       State  2 (idle)  braking sound finishes
;
L_GS1_State_11:

        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        bcs     L_GS1_11_Done           ; still playing braking

        jsr     F_GS_Set_State_2
L_GS1_11_Done:
        jmp     L_GS_Done

;******Model State 12*********************************************
;General:               grinding gears
;Movement:              unchanged
;Sound:                 grinding gears
;To Change State:       State  5 (gear 1)  braking sound finishes
;                       State 11 (braking) hit rev

L_GS1_State_12:

        jsr     F_GS1_Check_Fw_Still_Pressed

        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        BCS     L_GS1_12_Check_Braking  ; still playing squeeling

        jsr     F_GS1_Set_State_5
        jmp     L_GS_Done

L_GS1_12_Check_Braking:
        jsr     F_GS1_Check_Braking
        jmp     L_GS_Done


;******Model State 13*********************************************
;General:               squeel
;Movement:              turning, and going forward
;Sound:                 squeel
;To Change State:       State 11 (braking) hit rev
;                       State 5 (gear 1) no longer turning
L_GS1_State_13:

        lda     #0
        sta     R_Shifted
```

Shft96Tx.ASM

```
        jsr     F_GS1_Check_Fw_Still_Pressed

        ; check to see if turning
        lda     P_PortD
        and     #D_Pin_Left
        beq     L_GS1_13_Check_Braking  ; still turning

; check right turn
        lda     P_PortD
        and     #D_Pin_Right
        beq     L_GS1_13_Check_Braking  ; still turning

        ; no longer turning

        jsr     F_GS1_Set_State_5
        jmp     L_GS_Done

L_GS1_13_Check_Braking:
        jsr     F_GS1_Check_Braking
        jmp     L_GS_Done


;*******************************************
F_GS1_Check_Braking:

; checks braking when moving fwd

; see if rev hit (Causes braking)
        lda     P_PortD
        and     #D_Pin_Rev
        bne     L_GS1CB_Done

        jsr     F_GS_Set_State_11       ; braking
;       jmp     L_GS_Done

L_GS1CB_Done:
        rts

;********************************************8
F_GS1_Check_Premature_Shift:

        lda     R_Fwd_Ack_Ok
        beq     L_GS1PS_Done

        ; has returned to neutral

        lda     P_PortD
        and     #D_Pin_Fwd
        bne     L_GS1PS_Done    ; fwd not hit

        jsr     F_GS1_Set_State_12
        jmp     L_GS_Done

L_GS1PS_Done:
        rts

;*********************************
F_GS1_Check_Shift_Legal:

; decrement timer
        dec     R_Small_Shift_Timer
```

Shft96Tx.ASM

```
        lda     R_Small_Shift_Timer
        beq     L_DS_Dec_Big_Shift_Timer        ; if small timer has run to 0
        jmp     L_CSL_Done

L_DS_Dec_Big_Shift_Timer:
        lda     #ffh            ; reload small timer
        sta     R_Small_Shift_Timer

        Dec     R_Large_Shift_Timer
        LDA     R_Large_Shift_Timer     ;
        Bne     L_CSL_Done                      ; not legal yet

; shift is legal
        jsr     F_GS1_Set_State_8
        jmp     L_GS_Done

L_CSL_Done:
        rts
;************************************************
F_GS1_Check_Shift:
        lda     R_Fwd_Ack_Ok
        beq     L_GS1CS_Done

        lda     P_PortD
        and     #D_Pin_Fwd
        bne     L_GS1CS_Done    ; i.e. fwd pin is hi

        ; shift
        jsr     F_GS1_Set_State_10

L_GS1CS_Done:
        rts


;********************************
; checks if joystick has rtned to neutral position.  It must
; return here before a new fwd (or rev.) is acknowledged

F_GS1_Check_Fwd_Ack_Ok:

        lda     R_Fwd_Ack_Ok
        bne     L_GS1CF_Done    ;already okay

        lda     P_PortD
        and     #D_Pin_Fwd
        beq     L_GS1CF_Done    ; button pressed

; not pressed

        lda     #1
        sta     R_Fwd_Ack_Ok

L_GS1CF_Done:
        rts

F_GS1_Check_Rev_Ack_Ok:

        lda     R_Rev_Ack_Ok
        bne     L_GS1CR_Done    ;already okay

        lda     P_PortD
        and     #D_Pin_Rev
        beq     L_GS1CR_Done    ; button pressed
```

Shft96Tx.ASM

```
; not pressed

        lda     #1
        sta     R_Rev_Ack_Ok

L_GS1CR_Done:
        rts



;*********************************
; checks to see if fwd is still pressed, or if or it has only recently released
; directs a state change (braking) if fwd is no longer pressed, and if it has not
been
; pressed for more than .5 seconds

F_GS1_Check_Fw_Still_Pressed:

        lda     P_PortD
        and     #D_Pin_Fwd
        beq     L_GS1CFS_Init_Fwd_Release_Timer ; fwd still pressed

; decrement timer
        dec     R_Small_Fwd_Release_Timer
        lda     R_Small_Fwd_Release_Timer
        beq     L_GS1CFS_Dec_Big_Fwd_Release_Timer      ; if small timer has run to
0
        jmp     L_GS1CFS_Done

L_GS1CFS_Dec_Big_Fwd_Release_Timer:
        lda     #ffh            ; reload small timer
        sta     R_Small_Fwd_Release_Timer

        Dec     R_Large_Fwd_Release_Timer
        LDA     R_Large_Fwd_Release_Timer       ;
        Bne     L_GS1CFS_Done                   ; time not up yet
        ; time up

        jsr     F_GS_Set_State_11
        jmp     L_GS1CF_Done

L_GS1CFS_Init_Fwd_Release_Timer:

        lda     #D_Small_Fwd_Release_Timer_Preload
        sta     R_Small_Fwd_Release_Timer
        lda     #D_Large_Fwd_Release_Timer_Preload
        sta     R_Large_Fwd_Release_Timer

L_GS1CFS_Done:
        rts




;************************************
; some set state functions are shared by modes 1 and 2, some are not.

F_GS1_Set_State_5:
```

Shft96Tx.ASM

```
        lda     #0
        sta     R_Fwd_Ack_Ok

        lda     #1
        sta     R_Sound_Repeat

        lda     #1
        sta     R_Gear

        lda     #5
        sta     R_State

        lda     R_Peeled_Out
        bne     L_GS15_Clear_P_And_S            ; get it? P_and_S

        lda     R_Shifted
        bne     L_GS15_Clear_P_And_S

        jsr     F_GS_Preload_Shift_Timer        ; didn't peel out or shift (both
would
                                                ; (have already started timer),
start shift timer

                                                ; peeled out, so shift timer already
started
L_GS15_Clear_P_And_S:

        lda     #0
        sta     R_Peeled_Out
        sta     R_Shifted

        rts

F_GS1_Set_State_6:

        lda     #0
        sta     R_Fwd_Ack_Ok

        lda     #1
        sta     R_Sound_Repeat

        lda     #2
        sta     R_Gear

        lda     #6
        sta     R_State

        rts

F_GS1_Set_State_7:

        lda     #1
        sta     R_Sound_Repeat

        lda     #3
        sta     R_Gear

        lda     #7
        sta     R_State
        rts

F_GS1_Set_State_8:
```

Shft96TX.ASM

```
        lda     #0
        sta     R_Sound_Repeat
        sta     R_Sound_Check_Delay_Complete

        lda     #D_Small_Sound_Check_Timer_Preload
        sta     R_Small_Sound_Check_Timer
        lda     #D_Large_Sound_Check_Timer_Preload
        sta     R_Large_Sound_Check_Timer


        ; gear unchanged

        lda     #8
        sta     R_State
        rts

F_GS1_Set_State_9:
        lda     #1
        sta     R_Sound_Repeat

        ; gear unchanged

        lda     #9
        sta     R_State
        rts

F_GS1_Set_State_10:

        lda     #0
        sta     R_Sound_Repeat

        lda     #10
        sta     R_State

        lda     #1
        sta     R_Shifted
        jsr     F_GS_Preload_Shift_Timer

        lda     R_Gear
        cmp     #1
        beq     L_SS10_G2

        lda     #3
        sta     R_Gear

        jmp     L_SS10_Done

L_SS10_G2:
        lda     #2
        sta     R_Gear

L_SS10_Done:
        rts


F_GS1_Set_State_12:

        lda     #0
        sta     R_Sound_Repeat

        ; gear unchanged
        lda     #12
```

```
        sta     R_State
        rts
;****************************************************************
F_GS_Preload_Shift_Timer:

        lda     #D_Small_Shift_Timer_Preload
        sta     R_Small_Shift_Timer
        lda     #D_Large_Shift_Timer_Preload
        sta     R_Large_Shift_Timer

        rts

;****************************************************************
;****************************************
;Determine state for Mode 2
;****************************************
L_Get_State_M2:

        lda     R_State
        beq     L_GS2_State_0_Dummy

        cmp     #1
        beq     L_GS2_State_1_Dummy

        cmp     #2
        beq     L_GS2_State_2_Dummy

        cmp     #3
        beq     L_GS2_State_3

        cmp     #4
        beq     L_GS2_State_4

        cmp     #5
        beq     L_GS2_State_5_Dummy

        cmp     #6
        beq     L_GS2_State_6_Dummy

        cmp     #11
        beq     L_GS2_State_11_Dummy

        cmp     #13
        beq     L_GS2_State_13_Dummy

        cmp     #14
        beq     L_GS2_State_14_Dummy

        cmp     #15
        beq     L_GS2_State_15_Dummy

        jmp     L_GS_State_16


L_GS2_State_0_Dummy:
        jmp     L_GS_State_0    ; same for modes 1 and 2

L_GS2_State_1_Dummy:
        jmp     L_GS_State_1    ; same for modes 1 and 2

L_GS2_State_2_Dummy:
```

Shft96Tx.ASM

```
        jmp     L_GS_State_2     ; same for modes 1 and 2

L_GS2_State_5_Dummy:
        jmp     L_GS2_State_5

L_GS2_State_6_Dummy:
        jmp     L_GS2_State_6

L_GS2_State_11_Dummy:
        jmp     L_GS2_State_11

L_GS2_State_13_Dummy:
        jmp     L_GS2_State_13

L_GS2_State_14_Dummy:
        jmp     L_GS_State_14

L_GS2_State_15_Dummy:
        jmp     L_GS_State_15


;******Mode2 State 3***********************************************
;General:               Peelin' out
;Movement:              Fwd
;Sound:                 Peel Out
;To Change State:       State 4: Peel Out sound complete
;                       State 11: Fwd let go

; plays peelout-allow car to move
L_GS2_State_3:
        lda     #0
        sta     R_Sound_Repeat

; see if fwd still pressed
        lda     P_PortD
        and     #D_Pin_Fwd
        beq     L_GS2_3_Check_Sound_Finished

        lda     #D_Pin_Fwd
        sta     R_Dir
        jsr     F_GS_Set_State_11
        jmp     L_GS_Done

L_GS2_3_Check_Sound_Finished:    ; see if sound finished
        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        BCS     L_GS2_3_Done            ; still playing peelout

        jsr F_GS_Set_State_4

L_GS2_3_Done:
        jmp     L_GS_Done

;******Mode2 State 4***********************************************
;General:               Cruisin'
;Movement:              Fwd, possibly turning
;Sound:                 Motor Running
;To Change State:       State 5: Turning for some time
;                       State 11: Fwd no longer pressed
```

Shft96Tx.ASM

```
; going fwd. plays motor running sound, or occasionally gear shift-allow car to
move.
L_GS2_State_4:

        lda     #D_Pin_Fwd
        sta     R_Dir

        jsr     F_GS_Check_Squeel

        lda     P_PortD
        and     #D_Pin_Fwd
        beq     L_GS2_4_Done    ; fwd still pressed

        lda     #D_Pin_Fwd
        sta     R_Dir
        jsr     F_GS_Set_State_11       ; no longer moving

L_GS2_4_Done:

        jmp     L_GS_Done



;******Mode2 State 5***********************************************
;General:                Hard Braking
;Movement:               Neutral
;Sound:                  Hard Braking Sound
;To Change State:        State 4: Movement re-initiated before sound ends
;                        State 2: No Movement re-initiated before sound ends

; see if fwd and rev commanded

L_GS2_State_5:

        lda     P_PortD
        and     #D_Pin_Fwd
        beq     L_GS2_5_Set_State_4

L_GS2_5_Check_Reverse:
        lda     P_PortD
        and     #D_Pin_Rev
        bne     L_GS2_5_Check_Sound_Finished

        jsr     F_GS_Set_State_15
        jmp     L_GS_Done

L_GS2_5_Set_State_4:
        jsr     F_GS_Set_State_4
        jmp     L_GS_Done

L_GS2_5_Check_Sound_Finished:

        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        BCc     L_GS2_5_Set_State_2             ; still playing sound

        jmp     L_GS_Done

L_GS2_5_Set_State_2             ; Done playing sound
        jsr     F_GS_Set_State_2
        jmp     L_GS_Done
```

```
                                     Shft96Tx.ASM
;******Mode2 State 6*************************************************
;General:                Chirp
;Movement:               Fwd or Rev, can be turning
;Sound:                  Chirp
;To Change State:        State 4: At timeout if previously going reverse
;                        State 15: At timeout if previously going forward


L_GS2_State_6:


        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        BCC     L_GS_6_Check_Dir        ; still playing sound

        jmp     L_GS_Done
L_GS_6_Check_Dir:
        lda     P_PortD
        and     #D_Pin_Fwd
        bne     L_GS2_6_Rev

        jsr     F_GS_Set_State_4
        jmp     L_GS_Done

L_GS2_6_Rev:
        jsr     F_GS_Set_State_15
        jmp     L_GS_Done

;******Mode2 State 11************************************************
;General:                Fwd or Rev just released-wait to happens next
;Movement:               Neutral
;Sound:                  None
;To Change State:        State 5: If not slammed into opposite direction
;                        State 6: Slammed into reverse

L_GS2_State_11:

        ; check timer

        dec     R_Small_Chirp_Timer
        lda     R_Small_Chirp_Timer
        beq     L_GS2_11_Chirp_Timeout  ; if small timer has run to 0
        jmp     L_GS2_11_Check_Slam     ; timer not run out yet

L_GS2_11_Chirp_Timeout          ; timer has run out
        lda     #D_Small_Chirp_Timer_Preload; reload small timer
        sta     R_Small_Chirp_Timer
        jsr     F_GS2_Set_State_5
        jmp     L_GS_Done

L_GS2_11_Check_Slam:

        lda     R_Dir
        cmp     #D_Pin_Fwd
        beq     L_GS2_11_Check_FR_Slam

; check slam from reverse into fwd
        lda     P_PortD
        and     #D_Pin_Fwd
        bne     L_GS_Done               ; reverse not pressed

        jmp     L_GS_2_Slam
```

Shft96Tx.ASM

```
L_GS2_11_Check_FR_Slam:
        lda     P_PortD
        and     #D_Pin_Rev
        bne     L_GS_Done

L_GS_2_Slam:
        jsr     F_GS2_Set_State_6
        jmp     L_GS_Done


;******Mode2 State 13*************************************************
;General:               Squeeling
;Movement:              Fwd turning
;Sound:                 Squeeling like a stuck pig
;To Change State:       State 4: No Longer turning
;                       State 11: Fwd or Rev let go

L_GS2_State_13:

        lda     P_PortD
        and     #D_Pin_Fwd
        beq     L_GS2_13_Check_Turning

        lda     P_PortD
        and     #D_Pin_Rev
        beq     L_GS2_13_Check_Turning

        lda     #D_Pin_Fwd
        sta     R_Dir
        jsr     F_GS_Set_State_11 ;no longer going either fwd or rev-no squeel if
not moving
        jmp     L_GS_Done


L_GS2_13_Check_Turning:
        ; check to see if turning
        lda     P_PortD
        and     #D_Pin_Left
        bne     L_GS2_13_Check_Right_Turn

        jmp     L_GS_Done       ; still turning

L_GS2_13_Check_Right_Turn:

        lda     P_PortD
        and     #D_Pin_Right
        bne     L_GS2_13_Set_State_4

        jmp     L_GS_Done       ;still turning

L_GS2_13_Set_State_4:
        jsr     F_GS_Set_State_4
        jmp     L_GS_Done




F_GS2_Set_State_5:
        lda     #0
        sta     R_Sound_Repeat

        lda     #5
```

Shft96Tx.ASM

```
        sta     R_State
        rts

F_GS2_Set_State_6:
        lda     #0
        sta     R_Sound_Repeat

        lda     #6
        sta     R_State
        rts

L_GS_Done:
        rts

L_Get_State_M3:
        rts

;*************************************************
; General Get State Stuff
;*************************************************

;***Modes 1 or 2 State 0*******************************************
;General:               Waiting to be played with
;Gear                   0
;Sound:                 None
;To Change State:       State 1: Move any joystick

L_GS_State_0:

        lda     P_PortD         ; check activity on joysticks
        and     #0fh
        cmp     #0fh
        beq     L_GS0_Done

        jsr     F_GS_Set_State_1

L_GS0_Done:
        jmp     L_GS_Done


;******Modes 1 or 2 State 1*********************************************
;General:               Starting up
;Gear                   0
;Sound:                 Motor Starting
;To Change State:       State 2: Finished startup sound


L_GS_State_1:

        jsr     F_GS1_Check_Fwd_Ack_Ok
        jsr     F_GS1_Check_Rev_Ack_Ok

        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        bcs     L_GS_Done       ; still playing motor start sound

        ; sound finished
        jsr     F_GS_Set_State_2
        jmp     L_GS_Done


;******Modes 1 or 2 State 2**********************************************
;General:               Idling
```

Shft96Tx.ASM

```
;Movement:                 Neutral
;Sound:                    Motor Idling
;To Change State:          State 0: Idle Timer Runs Out
;                          State 3: Fwd or Rev Pressed (after sitting idle for a few
seconds)
;                          State 4: Fwd or Rev Pressed (after sitting idle for less
than a few seconds)

L_GS_State_2:

        lda     #0
        sta     R_Shifted

        jsr     F_GS1_Check_Fwd_Ack_Ok
        jsr     F_GS1_Check_Rev_Ack_Ok

        dec     R_Small_Idle_Timer
        lda     R_Small_Idle_Timer
        beq     L_GS_2_Dec_Big_Idle_Timer       ; if small timer has run to 0
        jmp     L_GS_2_Check_Fwd
L_GS_2_Dec_Big_Idle_Timer:
        lda     #ffh            ; reload small timer
        sta     R_Small_Idle_Timer

        Dec     R_Large_Idle_Timer
        LDA     R_Large_Idle_Timer      ;
        Bne     L_GS_2_Check_Peel_Out_Timer

; timer run to 0--back to mode 0
        jsr     F_GS_Set_State_0
        jmp     L_GS_Done

; peel out only sounds enabled if we've been in state 2 for a couple seconds

L_GS_2_Check_Peel_Out_Timer:
        lda     R_Peelout_Enable
        bne     L_GS_2_Check_Fwd

        lda     R_Large_Idle_Timer
        cmp     #D_Peelout_Time
        bcs     L_GS_2_Check_Fwd        ; idle timer greater or equal to peelout
time

        ; a couple of seconds have passed--enable peel out sound
        lda     #1
        sta     R_Peelout_Enable

L_GS_2_Check_Fwd:
        lda     P_PortD
        and     #D_Pin_Fwd
        bne     L_GS_2_Check_Reverse

; fwd hit
        lda     R_Fwd_Ack_Ok
        beq     L_GS_Done       ; joystick has not returned to center yet

        lda     R_Peelout_Enable
        beq     L_GS_2_Set_State_4

        jsr     F_GS_Set_State_3
        jmp     L_GS_Done

L_GS_2_Set_State_4:
```

Shft96Tx.ASM

```
        jsr     F_GS_Set_State_4
        jmp     L_GS_Done

L_GS_2_Check_Reverse:

        lda     R_Rev_Ack_Ok
        beq     L_GS_Done

        lda     P_PortD
        and     #D_Pin_Rev
        beq     L_GS_2_Check_Peelout

        jmp     L_GS_Done

L_GS_2_Check_Peelout:

        lda     R_Peelout_Enable
        beq     L_GS_2_Set_State_15

        jsr     F_GS_Set_State_14
        jmp     L_GS_Done

L_GS_2_Set_State_15:
        jsr     F_GS_Set_State_15
        jmp     L_DS_Done

;********************************************

;******Mode 1 or 2 State 14***********************************************
;General:                Reverse Peel Out
;Movement:               Reverse
;Sound:                  squeel
;To Change State:        State 11 (braking) hit rev
;                        State 15 (Rev, normal)  Peelout timer times out
;                        State 16 (rev peel out) turns for some time
L_GS_State_14:


        %TestSpeechCh1                  ; which will set the Carry if actively
playing
        BCS     L_GS_14_Check_Squeel    ; still playing peel out

        jsr     F_GS_Set_State_15       ; peel out timer timed out
        jmp     L_GS_Done

L_GS_14_Check_Squeel:
        jsr     F_GS_Check_Squeel

        lda     P_PortD
        and     #D_Pin_Rev
        beq     L_GS_14_Done            ; rev pin still pressed

        lda     #D_Pin_Rev
        sta     R_Dir
        jsr     F_GS_Set_State_11

L_GS_14_Done:
        jmp     L_GS_Done


;******Model or 2 State 15**********************************************
```

Shft96Tx.ASM

```
;General:                 Reverse
;Movement:                Reverse
;Sound:                   gear 1
;To Change State:         State 11 (Brake) let go of of reverse
;                         State 16 (Squeeling) turn for some time
;
L_GS_State_15:

        jsr     F_GS_Check_Squeel

        lda     P_PortD
        and     #D_Pin_Rev
        beq     L_GS_15_Done     ; still going in reverse

        lda     #D_Pin_Rev
        sta     R_Dir
        jsr     F_GS_Set_State_11

L_GS_15_Done:
        jmp     L_GS_Done

;******Model or 2 State 16**********************************************
;General:                 Turning in Reverse
;Movement:                Turning in Reverse
;Sound:                   squeeling
;To Change State:         pull joystick out of reverse
;                         brake
;
L_GS_State_16:
        ; check to see if turning
        lda     P_PortD
        and     #D_Pin_Left
        beq     L_GS_16_Check_Braking    ; still turning

; check right turn
        lda     P_PortD
        and     #D_Pin_Right
        beq     L_GS_16_Check_Braking    ; still turning

        ; no longer turning

        jsr     F_GS_Set_State_15
        jmp     L_GS_Done

L_GS_16_Check_Braking:

        lda     P_PortD
        and     #D_Pin_Rev
        beq     L_GS_16_Done

        lda     #D_Pin_Rev
        sta     R_Dir
        jsr     F_GS_Set_State_11

L_GS_16_Done:
        jmp     L_GS_Done

;***********************************************8
```

Shft96Tx.ASM

```
;*****************************************
F_GS_Set_State_0:

        lda     #0
        sta     R_Sound_Repeat

        sta     R_Gear

        lda     #0
        sta     R_State
        rts

F_GS_Set_State_1:

        lda     #0
        sta     R_Sound_Repeat
        sta     R_Gear
        sta     R_Fwd_Ack_Ok
        sta     R_Rev_Ack_Ok

        lda     #1
        sta     R_First_Start
        sta     R_State
        rts
F_GS_Set_State_2:

        lda     #1
        sta     R_Sound_Repeat

        lda     #0
        sta     R_Gear

        lda     R_First_Start           ; flag if 1st start since being off
        beq     L_GS2_Disable_Peelout
        lda     #1
        sta     R_Peelout_Enable
        lda     #0
        sta     R_First_Start
        jmp     L_GS_2_Preload_Idle_Timer

L_GS2_Disable_Peelout:
        lda     #0
        sta     R_Peelout_Enable

L_GS_2_Preload_Idle_Timer:      ; preload idle timer
        lda     #D_Small_Idle_Timer_Preload
        sta     R_Small_Idle_Timer
        lda     #D_Large_Idle_Timer_Preload
        sta     R_Large_Idle_Timer


        lda     #2
        sta     R_State
        rts

F_GS_Set_State_3:

        lda     #0
        sta     R_Fwd_Ack_Ok
```

Shft96Tx.ASM

```
        lda     #D_Small_Sound_Check_Timer_Preload
        sta     R_Small_Sound_Check_Timer
        lda     #D_Large_Sound_Check_Timer_Preload
        sta     R_Large_Sound_Check_Timer

        lda     #0
        sta     R_Sound_Check_Delay_Complete


        lda     #0
        sta     R_Sound_Repeat

        lda     #1
        sta     R_Gear

        jsr     F_GS_Preload_Shift_Timer

        lda     #3
        sta     R_State
        rts

F_GS_Set_State_4:

        lda     #0
        sta     R_Fwd_Ack_Ok
        sta     R_Rev_Ack_Ok

        lda     R_Mode
        cmp     #2
        beq     L_GSSS_4_Repeat

        lda     #0
        sta     R_Sound_Repeat
        jmp     L_GSSS_4_Store_Gear

L_GSSS_4_Repeat:
        lda     #1
        sta     R_Sound_Repeat

L_GSSS_4_Store_Gear:
        lda     #1
        sta     R_Gear

        lda     #4
        sta     R_State
        rts


F_GS_Set_State_11:

        lda     #0
        sta     R_Sound_Repeat

        ; preload chirp timer
        lda     #D_Small_Chirp_Timer_Preload
        sta     R_Small_Chirp_Timer

        lda     #11
        sta     R_State
        rts


F_GS_Set_State_13:
```

Shft96Tx.ASM

```
        lda     #1
        sta     R_Sound_Repeat

        lda     #1
        sta     R_Gear

        lda     #13
        sta     R_State
        rts

F_GS_Set_State_14:

        lda     #0
        sta     R_Sound_Repeat

;       lda     #ffh
;       sta     R_Gear

        lda     #14
        sta     R_State
        rts

F_GS_Set_State_15:

        lda     #1
        sta     R_Sound_Repeat

;       lda     #feh
;       sta     R_Gear

        lda     #15
        sta     R_State
        rts

F_GS_Set_State_16:

        lda     #1
        sta     R_Sound_Repeat

;       lda     #feh
;       sta     R_Gear

        lda     #16
        sta     R_State
        rts
;****************************************

F_GS_Check_Squeel:

        lda     P_PortD
        and     #D_Pin_Left
        beq     L_GSCS_Turning

        lda     P_PortD
        and     #D_Pin_Right
        beq     L_GSCS_Turning

        lda     #0              ; not turning
        sta     R_Turning

        rts
```

Shft96Tx.ASM

```
L_GSCS_Turning:

        lda     R_Turning
        bne     L_GSCS_Still_Turning

        ; not previously turning preload timer

        ; preload squeel timer
        lda     #D_Small_Squeel_Timer_Preload
        sta     R_Small_Squeel_Timer
        lda     #D_Large_Squeel_Timer_Preload
        sta     R_Large_Squeel_Timer

        lda     #1
        sta     R_Turning

        rts

L_GSCS_Still_Turning:

        dec     R_Small_Squeel_Timer
        lda     R_Small_Squeel_Timer
        beq     L_GS_Dec_Big_Squeel_Timer        ; if small timer has run to 0
        rts

L_GS_Dec_Big_Squeel_Timer:
        lda     #ffh            ; reload small timer
        sta     R_Small_Squeel_Timer

        Dec     R_Large_Squeel_Timer
        LDA     R_Large_Squeel_Timer    ;
        Beq     L_GS_Set_Squeel

        rts

L_GS_Set_Squeel:

        lda     P_PortD
        and     #D_Pin_Rev
        beq     L_GSSS_Reverse  ; only going in reverse if reverse button is pressed

        jsr     F_GS_Set_State_13
        jmp     L_GSCS_Done

L_GSSS_Reverse:
        jsr     F_GS_Set_State_16
        jmp     L_GSCS_Done


L_GSCS_Done:
        rts
;*****************************


;**************************************************************

;********************************
;********************************
```

Shft96Tx.ASM

```
;****decides which 2 byte packet to send*************

F_Decide_Packet:

        lda     R_Mode
        cmp     #1
        beq     L_DP_Mode_1
        cmp     #2
        beq     L_DP_Mode_2
        jmp     L_DP_Mode_3

; ****Mode 1 ***************************************
; Tx Command Depends on state

L_DP_Mode_1:

        lda     R_State
        cmp     #2
        bcc     L_DP_Blank_Packet       ; state is 0 1

        beq     L_DP_Pass_Steer_Only    ; state 2

        cmp     #11
        beq     L_DP_Set_Brake

        cmp     #14                     ; in gear 0 1 2 3
        bcc     L_DP1_Set_Gear_Bits_Fwd

        ; in reverse

        jmp     L_DP1_Set_Gear_Bits_Rev

L_DP1_Set_Gear_Bits_Fwd:

        ; moving the gear bits to the left puts em in the the
        ; pwm position of the tx packet
        lda     R_Gear
        clc
        rol     a
        rol     a
        rol     a
        rol     a
        sta     R_Gear_Bits
        jmp     L_DP_Set_Tx

L_DP1_Set_Gear_Bits_Rev:        ; reverse normal or peel out

        lda     #00001000b
        sta     R_Gear_Bits
        jmp     L_DP_Set_Tx


; ****Mode 2 ***************************************
; Tx Command Depends on state
; In some modes sends nothing, others passes through
; commands while making pwm all the way 1 or all the way 0

L_DP_Mode_2:

        lda     R_State
```

Shft96Tx.ASM

```
        cmp     #2                      ; state is 0 1
        bcc     L_DP_Blank_Packet

        beq     L_DP_Pass_Steer_Only

        cmp     #5
        beq     L_DP_Set_Brake

        cmp     #6
        beq     L_DP_Set_Brake

        cmp     #11
        beq     L_DP_Set_Brake

        ; allow movement
        jsr     F_DP_Set_Data_Full_Pwm

        jmp     L_DP_Done_Command

;**************************
; Shared by modes 1 and 2
;***************************
L_DP_Blank_Packet:
        lda     #0
        sta     R_Tx_Commands

        jmp     L_DP_Done_Command

L_DP_Set_Brake:

        lda     #00100100b      ; set brake secret code
        sta     R_Gear_Bits

        jmp     L_DP_Set_Tx

L_DP_Pass_Steer_Only:
        lda     #0
        sta     R_Gear_Bits

L_DP_Set_Tx:
        lda     P_PortD
        eor     #00000011b
        and     #00000011b
        ora     R_Gear_Bits
        sta     R_Tx_Commands
        jmp     L_DP_Done_Command


; ****Mode 3 ****************************************
; Passes through commands while making pwm all the way 1 or all the way 0
L_DP_Mode_3:

        jsr     F_DP_Set_Data_Full_Pwm
        ; keep going to done command
;*****************************************************8
L_DP_Done_Command:
        jsr     F_DP_Flag_And_Cksum

L_DP_Done:
        ; debug--put tx commands to led's
;       lda     R_Tx_Commands
;       sta     P_PortB
        rts
```

Shft96Tx.ASM

```
;**** Transmit Subroutines ************

F_DP_Set_Data_Full_Pwm:
        lda     P_PortD
        eor     #03h
        and     #03h
        sta     R_Tx_Commands

        lda     P_PortD
        and     #D_Pin_Fwd
        bne     L_DPSD_Check_Rev_Command

        ; if fwd button is down, set pwm to max
        lda     R_Tx_Commands
        ora     #D_Fwd_Bits
        sta     R_Tx_Commands
        JMP     L_DPSD_Tx_Commands_End

L_DPSD_Check_Rev_Command:
        lda     P_PortD
        and     #D_Pin_Rev
        bne     L_DPSD_Tx_Commands_End

        lda     R_Tx_Commands
        ora     #D_Rev_Bits
        sta     R_Tx_Commands

L_DPSD_Tx_Commands_End:
        rts

F_DP_Flag_And_Cksum:    ; Loads RX_data2 with the flag and checksum in the
                        ; bottom two bits.

        ; Checksum: counts the number of 1 bits in data

        LDX     #8                      ; X will be the loop counter

        LDA     #0                      ; clear
        STA     R_Tx_Flag_And_Ck                ;

        LDA     R_Tx_Commands           ; use to compute checksum

L_FB_Compute_Checksum:
; compute checksum

        ROL     A                       ; shift out MSB
        BCC     L_FB_CS_LoopEnd         ; and don't add one if that bit is zero

        INC     R_Tx_Flag_And_Ck                ; but if it's a 1, increment the
checksum

L_FB_CS_LoopEnd:

        DEX                             ; loop
        BNE     L_FB_Compute_Checksum           ;

        LDA     R_Tx_Flag_And_Ck                ; which has the checksum in low
three bits
        AND     #00000011b              ; so strip it down to just the bottom two
bits
        ORA     #D_TX_Flag              ; paste in the Flag
        STA     R_Tx_Flag_And_Ck        ;
```

```
        RTS

; ***************************************************
; F_Wait functions
;
; These all are versions of "while (TmB < Limit);"
; where the Limit is different for each one. It's
; faster with separate functions, each using #def'ed
; numbers instead of variables.
;
; Note that you shouldn't do the second loop, the lower
; byte, by itself. If you do, it can get stuck if it's
; waiting for P_TmBL to exceed FDh, for example.

; -------------------------------------------------



F_Wait_Sound_Service:

        lda     R_SS_Time_H,X
        sta     R_Wait_Time_H

        lda     R_SS_Time_L,X
        sta     R_Wait_Time_L

L_WS_Loop:
        LDA     P_TmBH                  ;
        AND     #0Fh                    ; strip away top nibble
        CMP     R_Wait_Time_H           ; and see if we're still within time limit
        BCC     L_WS_Loop               ; loop if still within time limit
        BNE     L_WS_Sound_Done         ; but if above limit, get out

        LDA     P_TmBL                  ; and if we're at the right TmBH, check TmBL
        CMP     R_Wait_Time_L           ;
        BCC     L_WS_Loop               ; loop if still within time limit

L_WS_Sound_Done:
        RTS                             ; and now the time has elapsed

;************************************************************
F_Wait_Tx_Line:

        lda     R_Tx_Time_H,X
        sta     R_Wait_Time_H

        lda     R_Tx_Time_L,X
        sta     R_Wait_Time_L

L_WT_Loop:
        LDA     P_TmBH                  ;
        AND     #0Fh                    ; strip away top nibble
        CMP     R_Wait_Time_H           ; and see if we're still within time limit
        BCC     L_WT_Loop               ; loop if still within time limit
        BNE     L_WT_Done               ; but if above limit, get out

        LDA     P_TmBL                  ; and if we're at the right TmBH, check TmBL
        CMP     R_Wait_Time_L           ;
        BCC     L_WT_Loop               ; loop if still within time limit
```

Shft96Tx.ASM

```
L_WT_Done:
        RTS                              ; and now the time has elapsed



; **************************************************
; F_DecideSounds
;
; Chooses what value to load into R_NextSound, which
; will be looked at by F_PlaySounds.
;
; **************************************************

F_Decide_Sounds:

        lda     R_Mode

        cmp     #1
        beq     L_DS_Mode_1_Sounds

        cmp     #2
        beq     L_DS_Mode_2_Sounds

        jmp     L_DS_Done       ; No sounds for mode 3

;********************Decide Sounds for Mode 1 *********************
L_DS_Mode_1_Sounds:

        lda     R_State
        cmp     #11
        beq     L_DS1_Set_Sound_Interrupt       ;brake
        cmp     #12
        beq     L_DS1_Set_Sound_Interrupt       ; grind
        cmp     #8                              ; ready to shift
        beq     L_DS1_Set_Sound_Interrupt
        cmp     #10                             ; upshift
        beq     L_DS1_Set_Sound_Interrupt

        lda     #0
        sta     R_Sound_Interrupt
        jmp     L_DS1_Load_Sound

L_DS1_Set_Sound_Interrupt:
        lda     #1
        sta     R_Sound_Interrupt
        lda     #0                       ; new code 9.26.01
        sta     R_Sound_Repeat

L_DS1_Load_Sound:
        ldx     R_State
        lda     R_Sounds_Array,X
        sta     R_Next_Sound

        jmp     L_DS_Done
;**************Decide Sounds for Mode 2 *************************
L_DS_Mode_2_Sounds:

        lda     R_State
        cmp     #11
```

Shft96Tx.ASM

```
        beq     L_DS_Done        ; m2 s11 is a short "deciding state"  --don't change
motor running
                                 ; sound until decided
        cmp     #5
        beq     L_DS2_Set_Sound_Interrupt       ;brake
        cmp     #6
        beq     L_DS2_Set_Sound_Interrupt       ; chirp

        lda     #0
        sta     R_Sound_Interrupt
        jmp     L_DS2_Random_Horn

L_DS2_Set_Sound_Interrupt:
        lda     #1
        sta     R_Sound_Interrupt


L_DS2_Random_Horn:

        lda     R_State
        cmp     #3
        beq     L_DS2_Check_Random_Horn_3

        cmp     #4
        beq     L_DS2_Check_Random_Shift_4

        jmp     L_DS2_Set_Sound_Index

L_DS2_Check_Random_Horn_3:
        lda     P_TmBL
        and     #0Fh
        bne     L_DS2_Set_Sound_Index

        ldx     #9
        jmp     L_DS2_Set_Sound

L_DS2_Check_Random_Shift_4:
        lda     P_TmBL
        and     #0Fh
        bne     L_DS2_Set_Sound_Index

        ldx     #10
        jmp     L_DS2_Set_Sound

L_DS2_Set_Sound_Index:
        ldx     R_State
L_DS2_Set_Sound:
        lda     R_Sounds_Array,X
        sta     R_Next_Sound

L_DS_Done:
        rts


; ***************************************************
; F_PlaySounds
;
; Looks at R_NextSound and handles the starting
; and repeating of sounds.
;
; ***************************************************
```

Shft96Tx.ASM

```
F_Play_Sounds:

        lda     R_Mode
        cmp     #3
        beq     L_PS_Done                   ; no sounds for mode 3

        lda     R_Sound_Interrupt           ; see if flag for immediate interruption of
sounds is set
        bne     L_PS_Check_Repeat           ; don't wait til sound is finished

        %TestSpeechCh1                      ; which will set the Carry if actively
playing
        BCS     L_PS_Done                   ; don't interrupt the sound

L_PS_Check_Repeat:

        lda     R_Sound_Repeat
        bne     L_PS_Start_New_Sound

        lda     R_Current_Sound
        cmp     R_Next_Sound
        beq     L_PS_Done                   ; sound loaded already

L_PS_Start_New_Sound:
        lda     R_Next_Sound
        sta     R_Current_Sound

        CMP     #D_Snd_None                 ;
        BEQ     L_PS_NoSound                ;

        LDX     #D_SamplePreload            ; the setting for sample frequency
        JSR     F_PlaySpeechCh1             ; play

        JMP     L_PS_Done                   ;

L_PS_NoSound:

        LDA     #D_Snd_None                 ;
        STA     R_Current_Sound             ; set NoSound as current
                                            ; just do nothing to let the sound run out
L_PS_Done:
        RTS

;*******************************************************************
F_Control_LED:

        lda     R_Mode

        cmp     #1
        bne     L_CL_Normal

; mode 1
        lda     R_State
        cmp     #8              ; flashes when ready to shift, whining
        beq     L_CL_Flash
        cmp     #9              ; and when ready to shift, constant
        bne     L_CL_Normal

        L_CL_Flash:

; decrement timer
        dec     R_Small_LED_Timer
```

Shft96Tx.ASM

```
        lda     R_Small_LED_Timer
        beq     L_DS_Flip_LED   ; if small timer has run to 0
        jmp     L_CL_Done

L_DS_Flip_LED:

        lda     P_PortC
        eor     #D_Pin_LED
        sta     P_PortC
; note that since we want a relatively short time cycle, we deal only with the
; small timer.
        ; preload timer
        lda     #D_Small_LED_Timer_Preload
        sta     R_Small_LED_Timer

        jmp     L_CL_Done

L_CL_Normal:

        lda     P_PortD
        and     #0Fh
        cmp     #0Fh
        beq     L_CL_Off

L_CL_On:
        lda     P_PortC
        ora     #D_Pin_LED
        sta     P_PortC
        jmp     L_CL_Done

L_CL_Off:
        lda     P_PortC
        and     #.NOT.D_Pin_LED
        sta     P_PortC

L_CL_Done:
        rts

; ********************** Interrupt Service Routine
******************************************

V_Irq:


        STA     R_TempA                 ; save accumulator value
        STX     R_TempX                 ; save x value

        LDA     P_Ints                  ; read the interrupt register and store
        STA     R_IntTemps              ; this variable is our working copy of the
interrupt register
        lda     #C0H                    ; ?
        STA     P_Ints
        LDA     R_IntFlags              ; load original interrupt settings and store
        STA     P_Ints

        LDA     R_IntTemps              ; check to see if timer A is the cause of
the interrupt
        AND     #TimeBase62_5Hz
        BEQ     L_Done_Int

        ; clk/655536 Service
```

Shft96Tx.ASM

```
        inc     R_Mode_Timer
        inc     R_Mode_Check_Timer

        L_Done_Int:

V_Nmi:                          ; non maskable interrupt--Sunplus does not support
this code
                                ; very well, and we have been warned not to use any
or mess
                                ; with it
        LDA     R_TempA
        LDX     R_TempX
        RTI

        .Include        Channel.Asm
        .DB             'PEND',0                ; no idea what this is

        ; Vectors settings - do not change  (from Sunplus Demo Code)

        .ORG    7FFAH
        DW      V_Nmi
        DW      V_Reset
        DW      V_Irq

        .ORG    FFFAH
        DW      V_Nmi
        DW      V_Reset
        DW      V_Irq
        END
```

Receiver Code

```
        .LINKLIST
        .SYMBOLS
        .CODE

; /***** System parameters *****

SystemClock:            EQU     2000000
SPC21A:                 EQU     1       ; select body (hardware.inh)

 .Include       Hardware.Inh

; ***** Addresses SunPlus forgot *****

P_MultiPhase            EQU     $37     ; register that controls Multi Phase
settings on 81A

ADPCM_TABLE_65:         EQU     1       ; If use ADPCM65 or later
_ADPCM_H_:              EQU     1       ; If no limit

; ***** CONSTANTS/DEFINES *****
; sound stuff
D_RampDownValue:        EQU     00H     ;If CurrentDAC->00H, PWM->80H
D_MaxWord:              EQU     6       ;number of speech pieces
D_MaxMelody:            EQU     1       ;number of melodies
D_MaxRhythm:            EQU     0       ;number of rhythms
D_SamplePreload:        EQU     00H     ;should be 6 for 8KHz or 0 for 6 Khz

; RX SETTINGS

D_RX_Flag:              EQU     01111100b ; Flag, in first six bits
D_RXbitCount_Limit      EQU     250     ; limit for bits received w/o good packet
before command erased

                                        ; 128 = 50 ms

                                        ; The TmB constants are used for "wait
until" timers

                                        ; 1 us = 2 timer ticks at 2Mhz osc

; wait times for different functions.

D_Wait_DPLL             EQU     0

D_Wait_BR1              EQU     1
D_Wait_BR2              EQU     2
D_Wait_PWM3             EQU     3
D_Wait_Tune             EQU     4

D_TmBH_Bit_Read1:       EQU     01h     ;
D_TmBL_Bit_Read1:       EQU     3Bh     ; 157 uS
D_TmBH_Bit_Read2:       EQU     03h     ;
D_TmBL_Bit_Read2:       EQU     B0h     ; 472 uS
```

Shft96Rx.ASM

```
D_TmBH_DPLLmin:          EQU     04h        ;
D_TmBL_DPLLmin:          EQU     24h        ; 424h = 1060 = 530 us

D_TmBH_DPLLmax:          EQU     05h        ;
D_TmBL_DPLLmax:          EQU     B4h        ; 5b4h = 730

D_TmBH_PWM3:             EQU     02h        ;
D_TmBL_PWM3:             EQU     8Ch        ; 2BCh =        350 us

D_TmBH_Tune:             EQU     04h        ;
D_TmBL_Tune:             EQU     ECh        ; 4eCh =        630 us

;D_Far_L_R_Motor_Timeout         EQU     4         ; 4*255*.0006= 06 seconds
D_Startup_Motor_Counts   EQU     2
; OUTPUTS

D_Pin_Tune_Out           EQU     00000100b          ; PortC
D_Pin_Forward:           EQU     00000001b          ; PortC
D_Pin_Reverse:           EQU     00000010b          ; PortC
D_Pins_Drive             EQU     00000011b          ; PortC
D_Pin_Drive_Enable       EQU     00000100b          ; PortC
D_Pin_Overcurrent        EQU     00001000b          ; PortC
D_Pin_Left:              EQU     00010000b          ; PortD
D_Pin_Right:             EQU     00100000b          ; PortD
D_Pins_Steer             EQU     00110000b          ; PortD

; INPUTS

D_Pin_RX:                EQU     00000100b          ; R/C RX pin, D2
D_Pin_Tune_Switch        EQU     00001000b          ; d3

; PACKET BITS
D_Fw_Bits                EQU     00110000b
D_Rev_Bits               EQU     00001100b
D_Left_Bit               EQU     00000010b
D_Right_Bit              EQU     00000001b

; pwm

D_PWM_Max:               EQU     16;     normally 16     ; numbers below are a
fraction of this

D_Drive_PWM_Low:         EQU     10      ; 10 for initial release, 10 for release
2.18.02
D_Drive_PWM_Medium:      EQU     12      ; 13 for initial release, 12 for release
2.18.02
D_Drive_PWM_High:        EQU     14      ; 16 for initial release, 14 for release
2.18.02

D_Steer_PWM_Hi_W_Spring  EQU     16      ; 12 for initial release, 16 for release
2.18.02
D_Steer_PWM_Lo_W_Spring  EQU     6
D_Steer_PWM_Hi_A_Spring  EQU     16      ; 5 for initial release, 12 for release
2.18.02
D_Steer_PWM_Lo_A_Spring  EQU     12

D_PWM_On_Delay_Time      EQU     15      ; 15*.00063=10ms

; steering control

D_Steer_Pos_Bits         EQU     00000011b
```

Shft96Rx.ASM

```
D_Steer_Left_Cmd        EQU     00000010b
D_Steer_Right_Cmd       EQU     00000001b

D_Steer_Near_Left_Pos   EQU     00000010b
D_Steer_Near_Right_Pos  EQU     00000001b
D_Steer_Ctr_Pos         EQU     00000000b
D_Steer_Far_Pos         EQU     00000011b

; Drive

D_Relay_Off_Delay_Hi    EQU     04h     ; =seconds/(630uS)
D_Relay_Off_Delay_Lo    EQU     A6h     ; 04A6h=1193.  1193*.00063=0.75 seconds
                                        ; this needs to be kept longer than the fwd
release timer
                                        ; in the tx code.  The fwd release timer
dictates how long
                                        ; the joystick may be in neutral when a kid
is shifting the
                                        ; vehcile.  When it is in this state we want
the vehicle to
                                        ; coast.  Making the Relay Off Timer shorter
than the fwd
                                        ; release timer would mean that the vehicle
would start to
                                        ; brake during the window in which it still
allowable to
                                        ; shift.  This isn't what we want.

; ***** VARIABLES *****
        .PAGE0
        .ORG    D_RamTop

R_IntFlags:     DS      1
R_IntTemps:     DS      1
R_TempA:        DS      1
R_TempX:        DS      1
R_Temp1:        DS      1
R_temp:         DS      1
R_RXdata:       DS      1       ; data received (reception complete)
R_RXdata2T:     DS      1       ; data2 temp
R_RXdata1T:     DS      1       ; data1 temp (during reception)
R_RXlastTE:     DS      1       ; record of the last half-bit or transmit element
R_RXdata_Last   DS      1
R_RXbitCount:   DS      1       ; count of bits received since last good command
(like error count)
R_Drive_PWM     DS      1       ; programmed PWM - 0=stop
R_Drive_Dir     DS      1       ; programmed direction

R_Steer_PWM     DS      1
R_Steer_Dir     DS      1

R_PWM_Counter   DS      1       ; rolling counter used for PWM

R_Rx_Error_Flag DS      1

R_Drive_Cmd     DS      1
R_Steer_Cmd     DS      1
R_Wait_Time_Array_H     DS      5
R_Wait_Time_Array_L     DS      5
R_Wait_Time_H   DS      1
R_Wait_Time_L   DS      1
```

Shft96Rx.ASM

```
R_Tuning          DS      1
R_Last_Mid_Pos    DS      1

R_Relay_Off_Counter_Hi  DS      1
R_Relay_Off_Counter_Lo  DS      1
R_PWM_On_Delay_Counter  DS      1
R_Drive_PWM_On          DS      1

R_Been_To_Center        DS      1
R_Startup_Steer         DS      1

R_Startup_Motor_Counter_Small   DS      1
R_Startup_Motor_Counter_Large   DS      1

R_Motor_Toggle                  DS      1
R_Current_Steer_Pos             DS      1

        .PAGE0

        .CODE
        .ORG    000H
        DB      FFH
        .ORG    600H

; ********************* Begin Main Code *****************


; V_ is by convention a vector.  The reset vector is where the code goes when the
micro is reset
V_Reset:

        ;(From Demo Code)
        LDX     #FFH            ; load ff into the x reg
        TXS                     ; transfer x reg contents to stack

                ; ***** Initialize variables

        lda     #0
        ldx     #0

        sta     R_IntFlags
        sta     R_IntTemps
        sta     R_TempA
        sta     R_TempX
        sta     R_Temp1
        sta     R_temp
        STA     R_RXdata
        STA     R_RXdata2T
        STA     R_RXdata1T
        STA     R_RXlastTE
        sta     R_RXdata_Last
        STA     R_RXbitCount
        sta     R_Drive_PWM
        sta     R_Drive_Dir
        sta     R_PWM_Counter
        sta     R_RX_Error_Flag
        sta     R_Drive_Cmd
        sta     R_Steer_Cmd
        sta     R_Wait_Time_H
        sta     R_Wait_Time_L
```

Shft96Rx.ASM

```
        sta     R_Tuning
        sta     R_Last_Mid_Pos
        sta     R_Relay_Off_Counter_Hi
        sta     R_Relay_Off_Counter_Lo
        sta     R_PWM_On_Delay_Counter
        sta     R_Drive_PWM_On
        sta     R_Been_To_Center
        sta     R_Startup_Steer
;       sta     R_Far_R_Motor_Counter_Small
;       sta     R_Far_R_Motor_Counter_Large
;       sta     R_Far_L_Motor_Counter_Small
;       sta     R_Far_L_Motor_Counter_Large
        sta     R_Startup_Motor_Counter_Small
        sta     R_Startup_Motor_Counter_Large
        sta     R_Current_Steer_Pos

        sta     R_Motor_Toggle

        ldx     #D_Wait_DPLL
        lda     #D_TmBH_DPLLmin
        sta     R_Wait_Time_Array_H,X

        lda     #D_TmBL_DPLLmin
        sta     R_Wait_Time_Array_L,X

        ldx     #D_Wait_BR1
        lda     #D_TmBH_Bit_Read1
        sta     R_Wait_Time_Array_H,X

        lda     #D_TmBL_Bit_Read1
        sta     R_Wait_Time_Array_L,X

        ldx     #D_Wait_BR2
        lda     #D_TmBH_Bit_Read2
        sta     R_Wait_Time_Array_H,X

        lda     #D_TmBL_Bit_Read2
        sta     R_Wait_Time_Array_L,X

        ldx     #D_Wait_PWM3
        lda     #D_TmBH_PWM3
        sta     R_Wait_Time_Array_H,X

        lda     #D_TmBL_PWM3
        sta     R_Wait_Time_Array_L,X

        ldx     #D_Wait_Tune
        lda     #D_TmBH_Tune
        sta     R_Wait_Time_Array_H,X

        lda     #D_TmBL_Tune
        sta     R_Wait_Time_Array_L,X

; ***** Port configuration

        lda     #0
        sta     P_PortD
        sta     P_PortC

        LDA     #10111111b              ; D-C-B-A, high-low, 1=output D: out C: b
output for debug
        STA     P_PortIO_Ctrl
```

Shft96Rx.ASM

```
        LDA     #00000000b               ; outputs buffer, except pull b1 low; Inputs
Pure
        STA     P_Port_Attrib

        LDA     #00000010b               ;
        STA     P_MultiPhase             ; turn off multi-phase on A2, but set 1/3
duty
                                         ; in case it does turn on for diagnosis

        ; ***** Configure interrupts
        lda     #%11000000               ; disable watchdog
                                         ; disable nmi
                                         ; enable TimerA interrupt
                                         ; enable TimerB interrupt
                                         ; disable 4 khz interrupt
                                         ; enable 500 Hz and
                                         ; enable 62.5 Hz interrupts
                                         ; disable external interupt
        STA     P_Ints                   ; store interrupt settings
        STA     R_IntFlags               ; store interrupt settings here, too

        SEI                              ; disable interrupts
; ***** Preload Timers

        LDA     #00h                     ;
        STA     P_TmAL                   ; preload: D58h = 315 us
        LDA     #00h                     ;
        STA     P_TmAH                   ; above and mode bits

        LDA     #00h                     ;
        STA     P_TmBL                   ;
        LDA     #00h                     ;
        STA     P_TmBH                   ;

        ; ***** Set port values

L_Main:

        ; check to see if in tuning mode
        lda     P_PortD
        and     #D_Pin_Tune_Switch
        beq     L_Tuning_Mode

L_Main_Loop:

        lda     #0
        sta     R_Tuning

;       ldx     #D_Wait_DPLL
;       JSR     F_Wait          ;
        JSR     F_DPLL

        JSR     F_PWM                    ;

        ldx     #D_Wait_BR1
        jsr     F_Wait
        JSR     F_BitRead1               ;

        JSR     F_PWM                    ;

        jsr     F_Service_Motors
        JSR     F_CheckBitCount          ; inc RXbitCount and see if it's been too
long
```

Shft96Rx.ASM

```
;       ldx     #D_Wait_PWM3
;       jsr     F_Wait
        JSR     F_PWM                 ;

        ldx     #D_Wait_BR2
        JSR     F_Wait                          ;
        JSR     F_BitRead2            ;

;       lda     P_PortB
;       ora     #00001000b
;       sta     P_PortB
        JSR     F_PWM
;       lda     P_PortB
;       and     #11110111b
;       sta     P_PortB;

        JMP     L_Main_Loop                     ;

; *** Tuning Mode
; Used to tune the oscillator frequency during product manufacture
; With the right oscillator, a pin on port D will blip every 630 Us

L_Tuning_Mode:

        LDA     #00h
        STA     P_TmBL
        LDA     #0
        STA     P_TmBH

        lda     #1
        sta     R_Tuning              ; for debug

        lda     P_PortC
        ora     #D_Pin_Tune_Out
        sta     P_PortC

        lda     P_PortC
        and     #.NOT.D_Pin_Tune_Out
        sta     P_PortC

        ldx     #D_Wait_Tune
        jsr     F_Wait
        jmp     L_Tuning_Mode

;*********************Functions***********************************


; *************************************************
; F_Wait functions
;
; This is a generic while (TmB < Limit);" function
; where the Limit is different for each one.

; Note that you shouldn't do the second loop, the lower
; byte, by itself. If you do, it can get stuck if it's
; waiting for P_TmBL to exceed FDh, for example.


; -------------------------------------------------
; F_Wait
```

Shft96Rx.ASM

```
F_Wait:

        lda     R_Wait_Time_Array_H,X
        sta     R_Wait_Time_H

        lda     R_Wait_Time_Array_L,X
        sta     R_Wait_Time_L

L_WT_Loop:
        LDA     P_TmBH                  ;
        AND     #0Fh                    ; strip away top nibble
        CMP     R_Wait_Time_H           ; and see if we're still within time limit
        BCC     L_WT_Loop               ; loop if still within time limit
        BNE     L_WT_Done               ; but if above limit, get out

        LDA     P_TmBL                  ; and if we're at the right TmBH, check TmBL
        CMP     R_Wait_Time_L           ;
        BCC     L_WT_Loop               ; loop if still within time limit

L_WT_Done:
        RTS                             ; and now the time has elapsed


; ----------------------------------------------

; ************************************************
; F_DPLL
;
; Syncs up to between-bit edges. If a transition
; isn't seen in time, it resets the timer anyway
; and lets things go on. This allows it to not cry
; "error" if it doesn't see a transition, in the event
; of good data with a missing transition, but it also
; can gradually get synced up with a new data stream.
; If there was a transition before this was called,
; it resets the timer right away, to also try to get
; on sync with the data stream.

F_DPLL:

        ; DIAGNOSTIC
        LDA     P_PortB
        ORA     #00000100b              ; B2 during DPLL
        STA     P_PortB

        LDA     R_RXlastTE              ;
        BEQ     L_DPLL_WaitForHigh      ; if it was low, watch for high, and vice
versa

L_DPLL_WaitForLow:


;       lda     P_PortB
;       ora     #00100000b              ; on b5
;       sta     P_PortB


        LDA     P_PortD                 ;
        AND     #D_Pin_RX               ;
        BEQ     L_DPLL_FoundEdge        ; BEQ b/c looking for "low"

        LDA     P_TmBH                  ;
        AND     #0Fh                    ; strip away top nibble
```

Shft96Rx.ASM

```
        CMP     #D_TmBH_DPLLmax         ; and see if we're still within time limit
        BCC     L_DPLL_WaitForLow       ; loop if still within time limit
        BNE     L_DPLL_FoundEdge        ; but if above limit, get out

        LDA     P_TmBL                  ; and if we're up to the TmBH, check TmBL
        CMP     #D_TmBL_DPLLmax         ;
        BCC     L_DPLL_WaitForLow       ; and loop if still within time

                                ; so the time did expire...

        JMP     L_DPLL_FoundEdge        ;

L_DPLL_WaitForHigh:

;       lda     P_PortB
;       ora     #01000000b
;       and     #11011111b
;       sta     P_PortB

        LDA     P_PortD                 ;
        AND     #D_Pin_RX               ;
        BNE     L_DPLL_FoundEdge        ; BNE b/c looking for "high"

        LDA     P_TmBH                  ;
        AND     #0Fh                    ; strip away top nibble
        CMP     #D_TmBH_DPLLmax         ; and see if we're still within time limit
        BCC     L_DPLL_WaitForHigh      ; loop if still within time limit
        BNE     L_DPLL_FoundEdge        ; but if above limit, get out

        LDA     P_TmBL                  ; and if we're up to the TmBH, check TmBL
        CMP     #D_TmBL_DPLLmax         ;
        BCC     L_DPLL_WaitForHigh      ; and loop if still within time

                                ; so the time did expire...

;       JMP     L_DPLL_FoundEdge                ; go ahead and reset timer

L_DPLL_FoundEdge:

        LDA     #00h
        STA     P_TmBL
        LDA     #0
        STA     P_TmBH


        ; DIAGNOSTIC
        LDA     P_PortB                 ;
        AND     #.NOT.00000100b         ; B2 during entire DPLL window
        STA     P_PortB

        RTS                             ; and you're done


; ***************************************************
; F_BitRead1
;
; Takes one look at Pin_RX to see if it has changed
; across the bit boundary, as it should, and records
; the new state. If not, sets R_RXerror = #D_Error_NoBitBoundary

F_BitRead1:
```

Shft96Rx.ASM

```
        ; DIAGNOSTIC
        LDA     P_PortB
        ORA     #00000100b          ; blip B2 for bit read
        STA     P_PortB

        LDA     P_PortB             ;
        AND     #.NOT.00000100b     ; B2 during entire DPLL window
        STA     P_PortB
        ;

        LDA     R_RXlastTE          ;
        BEQ     L_BR1_LastWasLow    ; main branch based on whether last TE was
0/1

L_BR1_LastWasHigh:

        LDA     P_PortD             ;
        AND     #D_Pin_RX           ; check PinRX -- it should be low now
        BNE     L_BR1_Error         ; and if not, it's an error

        LDA     #0                  ;
        STA     R_RXlastTE          ; and now RXlastTE is reassigned with
present state

        RTS

L_BR1_LastWasLow:

        LDA     P_PortD             ;
        AND     #D_Pin_RX           ; check PinRX -- it should be high now
        BEQ     L_BR1_Error         ; and if not, it's an error

        LDA     #1                  ;
        STA     R_RXlastTE          ; and now RXlastTE is reassigned with
present state

        RTS

L_BR1_Error:


;        DIAGNOSTIC
;       LDA     P_PortD
;       AND     #.NOT.10000000b     ; D7 off to show failure
;       STA     P_PortD

        ; DIAGNOSTIC
        ;LDA    P_PortB
        ;ORA    #00010000b          ; B4 to show any error
        ;STA    P_PortB
        ;AND    #.NOT.00010000b     ;
        ;STA    P_PortB

        LDA     #0                  ;
        STA     R_RXdata1T          ;
        STA     R_RXdata2T          ; clear the data buffer

        ;LDA    #D_Error_NoBitBoundary
        ;STA    R_RXerror
        lda     #1
        sta     R_RX_Error_Flag
```

Shft96Rx.ASM

```
        RTS

; ***************************************************
; F_BitRead2
;
; Takes one look at Pin_RX to see if it has changed
; from the first TE of the bit. If it has, the bit
; is a 1. If not, it's a 0. As such, there's no
; error detection here, though it could be added by
; doing redundant Pin_RX reads.
;
; The end of the routine does packet-level checks,
; looking for the flag and then computing the checksum
; and comparing it.

F_BitRead2:

        ; DIAGNOSTIC
        NOP
        LDA     P_PortB
        ORA     #00000100b              ; blip B2 for bit read
        STA     P_PortB
        AND     #11111011b              ; blip B2 for bit read
        STA     P_PortB


        LDA     R_RXlastTE              ;
        BEQ     L_BR2_LastWasLow        ; main branch based on whether last TE was
0/1

L_BR2_LastWasHigh:

        LDA     P_PortD                 ;
        AND     #D_Pin_RX               ; check Pin_RX
        BNE     L_BR2_BitIs0            ; if still high, it's a 0
                                        ; and else it's a 1
        LDA     #0                      ;
        STA     R_RXlastTE              ; record the change in Pin_RX

        SEC                             ;
        JMP     L_BR2_ShiftBitIn        ;

L_BR2_LastWasLow:

        LDA     P_PortD                 ;
        AND     #D_Pin_RX               ; check PinRX
        BEQ     L_BR2_BitIs0            ; if still low, it's a 0
                                        ; and else it's a 1
        LDA     #1                      ;
        STA     R_RXlastTE              ; record the change in Pin_RX

        SEC                             ;
        JMP     L_BR2_ShiftBitIn        ;

L_BR2_BitIs0:

        CLC                             ; clear Carry, which will be shifted in

L_BR2_ShiftBitIn:

        LDA     R_Rx_Error_Flag         ;
```

```
        BNE     L_BR2_EarlierError_Dummy        ; if an error was detected by now,
don't count the bit

        ROL     R_RXdata1T              ;
        ROL     R_RXdata2T              ;

        LDA     R_RXdata2T              ;
        AND     #11111100b              ; strip off bottom two bits
        CMP     #D_RX_Flag              ; and check for Flag
        BNE     L_BR2_Done              ; and if not found, just keep receiving

        ; Flag found, now calculate and compare checksum

        ; Checksum: counts the number of 1 bits in data

        LDA     #0                      ;
        STA     R_temp                  ; R_temp will be our checksum count
        LDX     #8                      ; X will be the loop counter
        LDA     R_RXdata1T              ; A will be the rotated byte

L_BR2_Checksum:

        ROL     A                       ; shift out MSB
        BCC     L_BR2_CS_LoopEnd        ; and don't add one if that bit is zero
        INC     R_temp                  ; but if it's a 1, increment the checksum

L_BR2_CS_LoopEnd:

        DEX                             ; end of loop
        BNE     L_BR2_Checksum          ;

                                ; and now compare the checksums

        LDA     R_temp                  ;
        AND     #00000011b              ; clear flag bits
        ORA     #D_RX_Flag              ; paste in the Flag so that it _should_
equal data2T
        CMP     R_RXdata2T              ;
        BNE     L_BR2_BadChecksum       ;

                                ; So it's good!

        ; DIAGNOSTIC
;       LDA     P_PortB
;       ORA     #00000010b              ; B1 on to show success
;       STA     P_PortB
        ; DIAGNOSTIC
;       LDA     P_PortD
;       ORA     #10000000b              ; D7 on to show success
;       STA     P_PortD

        LDA     #0                      ;
        STA     R_RXbitCount            ; reset the bit count b/c success

        LDA     R_RXdata1T              ;
        cmp     R_RXdata_Last
        beq     L_BR2_Good_Data

; data bit is different than last time
L_BR2_Store_Rx_Reading:
        lda     R_RXdata1T
        sta     R_RXdata_Last           ; store the received data for next time.
        jmp     L_BR2_Done
```

```
L_BR2_EarlierError_Dummy:
        jmp     L_BR2_EarlierError

L_BR2_Good_Data:

        STA     R_RXdata                ; data is same 2 times in a row, so count
it.
        sta     R_RXdata_Last

        LDA     R_RXdata                ; store for external consumption (steer and
drive)
        AND     #00000011b              ; look at steering bits
        STA     R_Steer_Cmd             ; and store them as SteerCmd

        LDA     R_RXdata                ;
        AND     #00111100b              ; strip away non-drive bits
        STA     R_Drive_Cmd             ; and store for drive function

L_BR2_Done:

        RTS

L_BR2_BadChecksum:

        ; DIAGNOSTIC
;       LDA     P_PortD
;       AND     #.NOT.10000000b         ; D7 off to show failure
;       STA     P_PortD

        ; DIAGNOSTIC
;       LDA     P_PortB
;       ORA     #00010000b              ; B4 to show any error
;       STA     P_PortB
;       AND     #.NOT.00010000b         ;
;       STA     P_PortB

L_BR2_EarlierError:

        LDA     #0                      ;
        STA     R_Rx_Error_Flag         ; clear the error flag for next bit

        LDA     #0                      ;
        STA     R_RXdata1T              ;
        STA     R_RXdata2T              ; clear the data buffer
        sta     R_RXdata


        ; Checksum: counts the number of 1 bits in data

                                ; So it's good!

; ************************************************
; F_CheckBitCount
;
; If the RXbitCount since the last good packet exceeds
; a limit, the last good packet is forgotten and the
; motors are given default "off" commands.
; This func gets called every bit.  The counter gets reset
; when a good packet is received.
```

Shft96Rx.ASM

```
F_CheckBitCount:

        INC     R_RXbitCount            ; increment every time through

        LDA     R_RXbitCount
        CMP     #D_RXbitCount_Limit     ;
        BCC     L_CBC_Done              ; if count < limit, exit

                                ; over limit
        ; DIAGNOSTIC
;       LDA     P_PortB
;       AND     #.NOT.00000010b         ; turn off B1 as command is erased
;       STA     P_PortB

        LDA     #00b            ;
        STA     R_RXdata                ; drive=3, steer=3, twist=0
        sta     R_RXdata_Last
        sta     R_Steer_Cmd
        sta     R_Drive_Cmd

        STA     R_RXbitCount            ; and reset the bit count

L_CBC_Done:

        RTS


F_Service_Motors:
        ; alternate btwn service of drive and steering motors

        lda     R_Motor_Toggle
        beq     L_SM_Drive
        jsr     F_Service_Steering_Motor
        lda     #0
        sta     R_Motor_Toggle
        jmp     L_SM_Done

L_SM_Drive:

        jsr     F_Service_Drive_Motor
        lda     #1                      ; set for steering motor for next time
        sta     R_Motor_Toggle

L_SM_Done:
        rts

; ***************************************************
; F_ServiceSteeringMotor
;
; Manages the steering motor servo-style.
; The commands are:
;       00000001b       Steer Right
;       00000010b       Steer Left
;       00000000b       Steer Straight
;       00000011b       Error-Invalid command
; The measured positions are:
;       00000010b       Near Right
;       00000001b       Near Left
;       00000000b       Center
;       00000011b       Either Far Right or Far Left

; since there is not a direct mapping btwn the commands and the positions,
```

Shft96Rx.ASM

```
; the code is a little more lengthy and a little less slick.

F_Service_Steering_Motor:

        lda     R_Been_To_Center
        bne     L_SSM_Normal

        lda     P_PortD
        and     #D_Steer_Pos_Bits
        cmp     #D_Steer_Far_Pos
        beq     L_SSM_Init_Motor_Move

        lda     #1
        sta     R_Been_To_Center ;(or near r or L)
        jmp     L_SSM_Normal


; in the case that the vehicle is turned on and doesn't know if it is far r or far L
L_SSM_Init_Motor_Move:

        ; move motor fast for 0.5 s Right, if it doesn't get to ctr, move for 0.5 S
Left
        ; if it's still not at center turn motors off

        lda     R_Startup_Steer
        beq     L_SSM_Motor_Right_Init

        cmp     #1
        beq     L_SSM_Motor_Left_Init

        jmp     L_SSM_Motor_Off

L_SSM_Motor_Right_Init:
        inc     R_Startup_Motor_Counter_Small
        lda     R_Startup_Motor_Counter_Small
        cmp     #ffh
        bne     L_Dummy_SSM_Motor_Right_Fast_A_Spring   ; still not time
        lda     #0
        sta     R_Startup_Motor_Counter_Small
        inc     R_Startup_Motor_Counter_Large

        lda     R_Startup_Motor_Counter_Large
        cmp     #D_Startup_Motor_Counts
        bne     L_Dummy_SSM_Motor_Right_Fast_A_Spring   ; still not time

        ; try moving left now
        lda     #1
        sta     R_Startup_Steer
        lda     #0
        sta     R_Startup_Motor_Counter_Small
        sta     R_Startup_Motor_Counter_Large

L_SSM_Motor_Left_Init:

        inc     R_Startup_Motor_Counter_Small
        lda     R_Startup_Motor_Counter_Small
        cmp     #ffh
        bne     L_Dummy_SSM_Motor_Left_Fast_A_Spring    ; still not time
        lda     #0
        sta     R_Startup_Motor_Counter_Small
        inc     R_Startup_Motor_Counter_Large
```

Shft96Rx.ASM

```
        lda     R_Startup_Motor_Counter_Large
        cmp     #D_Startup_Motor_Counts
        bne     L_Dummy_SSM_Motor_Left_Fast_A_Spring    ; still not time

        lda     #2
        sta     R_Startup_Steer

        jmp     L_SSM_Motor_Off         ; never got out of far l or r something is
wrong

L_Dummy_SSM_Motor_Left_Fast_A_Spring:
        jmp     L_SSM_Motor_Left_Fast_A_Spring

L_Dummy_SSM_Motor_Right_Fast_A_Spring
        jmp     L_SSM_Motor_Right_Fast_A_Spring

L_SSM_Normal:
; check for command error

        lda     R_Steer_Cmd
        and     #00000011b
        cmp     #00000011b
        bne     L_SSM_Get_Current

        jmp     L_SSM_Error

; get the current position
L_SSM_Get_Current:

        lda     P_PortD
        and     #D_Steer_Pos_Bits

        cmp     #D_Steer_Ctr_Pos
        beq     L_SSM_Cur_Center

        cmp     #D_Steer_Near_Left_Pos
        beq     L_SSM_Cur_Near_L

        cmp     #D_Steer_Near_Right_Pos
        beq     L_SSM_Cur_Near_R

        ; current position bits indicate it's either far left or far right
        ; check where it was last time to see where it must be now
        lda     R_Last_Mid_Pos
        cmp     #D_Steer_Near_Left_Pos
        beq     L_SSM_Cur_Far_L

        cmp     #D_Steer_Near_Right_Pos
        beq     L_SSM_Cur_Far_R

        jmp     L_SSM_Error             ; if sensor is broken or unplugged it will
go here
                                        ; since last mid pos was always center

; compare to command and decide which way to move and at what pwm

L_SSM_Cur_Center:

        lda     #0
        sta     R_Current_Steer_Pos

        sta     R_Last_Mid_Pos
```

Shft96Rx.ASM

```
        lda     R_Steer_Cmd
        beq     L_SSM_Motor_Off                 ; commanded straight

        cmp     #D_Steer_Left_Cmd               ; commanded left
        beq     L_SSM_Motor_Left_Fast_A_Spring
                                                ; commanded right
        jmp     L_SSM_Motor_Right_Fast_A_Spring

L_SSM_Cur_Near_L:

        lda     #2
        sta     R_Current_Steer_Pos

        sta     R_Last_Mid_Pos

        lda     R_Steer_Cmd
        ;cmp    #0                              ; commanded straight
        beq     L_SSM_Motor_Right_Slow_W_Spring

        cmp     #D_Steer_Left_Cmd               ; commanded left
        beq     L_SSM_Motor_Left_Slow_A_Spring
                                                ; commanded right
        jmp     L_SSM_Motor_Right_Fast_W_Spring


L_SSM_Cur_Near_R:

        lda     #1
        sta     R_Current_Steer_Pos

        sta     R_Last_Mid_Pos

        lda     R_Steer_Cmd
        ; cmp #0                                ; commanded straight
        beq     L_SSM_Motor_Left_Slow_W_Spring

        cmp     #D_Steer_Left_Cmd               ; commanded left
        beq     L_SSM_Motor_Left_Fast_W_Spring
                                                ; commanded right
        jmp     L_SSM_Motor_Right_Slow_A_Spring


L_SSM_Cur_Far_R:

        lda     #3
        sta     R_Current_Steer_Pos

        lda     R_Steer_Cmd
        cmp     #D_Steer_Right_Cmd              ; commanded right
        beq     L_SSM_Motor_Off
        jmp     L_SSM_Motor_Left_Fast_W_Spring  ; steer command is straight or left


L_SSM_Cur_Far_L:

        lda     #4
        sta     R_Current_Steer_Pos
        lda     R_Steer_Cmd
        cmp     #D_Steer_Left_Cmd               ; commanded left
        beq     L_SSM_Motor_Off

        jmp     L_SSM_Motor_Right_Fast_W_Spring ; command is right or straight
```

Shft96Rx.ASM

```
; set the directions and pwm rates

L_SSM_Motor_Off:

        lda     #0
        sta     R_Steer_Dir

        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Left_Slow_W_Spring:

        lda     #D_Pin_Left
        sta     R_Steer_Dir

        lda     #D_Steer_PWM_Lo_W_Spring
        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Left_Fast_W_Spring:

        lda     #D_Pin_Left
        sta     R_Steer_Dir

        lda     #D_Steer_PWM_Hi_W_Spring
        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Right_Slow_W_Spring:

        lda     #D_Pin_Right
        sta     R_Steer_Dir

        lda     #D_Steer_PWM_Lo_W_Spring;
        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Right_Fast_W_Spring:

        lda     #D_Pin_Right
        sta     R_Steer_Dir
        lda     #D_Steer_PWM_Hi_W_Spring
        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Left_Slow_A_Spring:

        lda     #D_Pin_Left
        sta     R_Steer_Dir

        lda     #D_Steer_PWM_Lo_A_Spring
        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Left_Fast_A_Spring:

        lda     #D_Pin_Left
        sta     R_Steer_Dir

        lda     #D_Steer_PWM_Hi_A_Spring
        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Right_Slow_A_Spring:

        lda     #D_Pin_Right
        sta     R_Steer_Dir
```

```
                                    Shft96Rx.ASM
        lda     #D_Steer_PWM_Lo_A_Spring;
        jmp     L_SSM_Set_PWM_Done

L_SSM_Motor_Right_Fast_A_Spring:

        lda     #D_Pin_Right
        sta     R_Steer_Dir
        lda     #D_Steer_PWM_Hi_A_Spring
        jmp     L_SSM_Set_PWM_Done

L_SSM_Set_PWM_Done:

        sta     R_Steer_PWM
        jmp     L_SSM_Done

L_SSM_Error:

        lda     P_PortD                 ; steering motor off
        and     #11111100b
        sta     P_PortD

L_SSM_Done:
        rts

; *************************************************
; F_ServiceDriveMotor
;
;       The motor driver circuit is a little different than the usual H-Bridge
configuration.
;       Because of the high drive current, the cicuit uses relays which are enabled
with
;       a FET.  The FET is PWMed, while the relays are just turned on whenever the
motor is on
;       all the way or just at some PWM rate.  Also, a Current sense enable pin is
brought high
;       whenever the vehicle is driven.
;
;

F_Service_Drive_Motor:

        lda     R_Drive_Cmd

        CMP     #00001100b                      ;
        BEQ     L_SDM_Reverse_High      ;

        CMP     #00001000b                      ;
        BEQ     L_SDM_Reverse_Medium    ;

        CMP     #00000100b                      ;
        BEQ     L_SDM_Reverse_Low       ;

        CMP     #00110000b                      ;
        BEQ     L_SDM_Forward_High      ;

        CMP     #00100000b                      ;
        BEQ     L_SDM_Forward_Medium    ;

        CMP     #00010000b                      ;
        BEQ     L_SDM_Forward_Low       ;

        ; stop by default
```

Shft96Rx.ASM

```
        LDA     #0                      ; turn off pwm
        STA     R_Drive_PWM             ;

        ; A delay is required before switching the relays.  This keeps from damaging
the relays

        ; check to see if the relays have already been shut off

        lda     P_PortC
        and     #D_Pins_Drive
        bne     L_SDM_Inc_Relay_Off_Counter

        jmp     L_SDM_Done      ; relays have already been shut off.

        ; inc counter for turning off drive pins--should wait x seconds to turn
relays after
        ; turning off pwm.

L_SDM_Inc_Relay_Off_Counter:

        inc     R_Relay_Off_Counter_Lo
        lda     R_Relay_Off_Counter_Lo
        cmp     #ffh
        bne     L_SDM_Check_Limit

        lda     #0                              ; rollover
        sta     R_Relay_Off_Counter_Lo

        inc     R_Relay_Off_Counter_Hi          ; and inc the hi counter

L_SDM_Check_Limit:

        lda     R_Relay_Off_Counter_Hi
        cmp     #D_Relay_Off_Delay_Hi
        beq     L_SDM_Check_Low_Delay

        jmp     L_SDM_Done                      ; not there yet

L_SDM_Check_Low_Delay:

        lda     R_Relay_Off_Counter_Lo
        cmp     #D_Relay_Off_Delay_Lo
        beq     L_SDM_Shut_Off_Relays

        jmp     L_SDM_Done

        ; time to shut off relays which will activate dynamic braking
L_SDM_Shut_Off_Relays:

        lda     #0                      ; reset counters for next time
        sta     R_Relay_Off_Counter_Lo
        sta     R_Relay_Off_Counter_Hi

        lda     P_PortC
        and     #.NOT.D_Pins_Drive      ; brakes
        and     #.NOT.D_Pin_Overcurrent ;
        sta     P_PortC

        JMP     L_SDM_Done              ;


L_SDM_Reverse_High:
```

Shft96Rx.ASM

```
        LDA     #D_Drive_PWM_High       ;
        STA     R_Drive_PWM             ;

        LDA     #D_Pin_Reverse          ; is this necessary?
        STA     R_Drive_Dir             ;
        JMP     L_Set_Direction         ;

L_SDM_Reverse_Medium:

        LDA     #D_Drive_PWM_Medium     ;
        STA     R_Drive_PWM             ;

        LDA     #D_Pin_Reverse          ;
        STA     R_Drive_Dir             ;
        JMP     L_Set_Direction         ;


L_SDM_Reverse_Low:

        LDA     #D_Drive_PWM_Low        ;
        STA     R_Drive_PWM             ;

        LDA     #D_Pin_Reverse          ;
        STA     R_Drive_Dir             ;
        JMP     L_Set_Direction         ;

L_SDM_Forward_Low:

        LDA     #D_Drive_PWM_Low                ;
        STA     R_Drive_PWM             ;;

        LDA     #D_Pin_Forward          ;
        STA     R_Drive_Dir             ;
        JMP     L_Set_Direction         ;

L_SDM_Forward_Medium:

        LDA     #D_Drive_PWM_Medium     ;
        STA     R_Drive_PWM             ;;

        LDA     #D_Pin_Forward          ;
        STA     R_Drive_Dir             ;
        JMP     L_Set_Direction         ;

L_SDM_Forward_High:

        LDA     #D_Drive_PWM_High       ;
        STA     R_Drive_PWM             ;;

        LDA     #D_Pin_Forward          ;
        STA     R_Drive_Dir             ;
        JMP     L_Set_Direction         ;

L_Set_Direction:

        lda     P_PortC
        and     #.NOT.D_Pins_Drive
        ora     R_Drive_Dir             ; switch relay
        ora     #D_Pin_Overcurrent      ; and overcurrent enble pin
        sta     P_PortC

        lda     #0                      ; keep clear
```

```
                          Shft96Rx.ASM
        sta     R_Relay_Off_Counter_Hi
        sta     R_Relay_Off_Counter_Lo

                                        ; turning pwm on
        jmp     L_SDM_Done


L_SDM_Done:

        RTS


;* PWM MOTORS *
; pwm drive and steering motors
; Note that the drive motors use relays (in place of where the power transistors in
an
; H-bridge usually are) along with a drive enable pin.  The relays are connected
first, then
; motor is PWM with some non-zero frequency some finite (~.1 second?)  This is done
to protect
; the relay.

F_PWM:
        ; increment counter; will be used for both drive and steering pwm
determination

        INC     R_PWM_Counter           ;
        LDA     R_PWM_Counter           ;
        CMP     #D_PWM_Max              ;
        BNE     L_PWM_Drive_Service     ; don't reset counter until it matches "Max"

        LDA     #0                      ; rollover reset
        STA     R_PWM_Counter           ;

L_PWM_Drive_Service:


        lda     R_Drive_PWM
        bne     L_PWM_Check_Delay

; motors are commanded off (pwm=0)
        lda     #0                      ; reset counter
        sta     R_PWM_On_Delay_Counter

        jmp     L_PWM_Drive_Off

; pwm is non-zero

L_PWM_Check_Delay:

        lda     R_Drive_PWM_On
        bne     L_PWM_Drive_Decide      ; pwm turned already turned on (on delay has
passed)

        ; still in delay btwn when relays are turned on and when it is time to turn
on FET
        inc     R_PWM_On_Delay_Counter  ; increment and check the counter to see if
we can turn
        lda     R_PWM_On_Delay_Counter
        cmp     #D_PWM_On_Delay_Time
        bne     L_PWM_Drive_Done        ; not yet

        ; it's time
```

Shft96Rx.ASM

```
        lda     #1
        sta     R_Drive_PWM_On

L_PWM_Drive_Decide:

        ; now that it's okay to turn FET on, this routine will do the actual PWMing
of the FET

;       LDA     R_Drive_PWM             ; if set to zero, stop the motor
;       BEQ     L_PWM_Drive_Off ;

        LDA     R_PWM_Counter           ;
        CMP     R_Drive_PWM             ;
        BCC     L_PWM_Drive_On          ; if Counter less than setting, turn on
                                        ; else turn off
L_PWM_Drive_Off:
                                        ; Set pin hi for PWM off
        LDA     P_PortC                 ;
        and     #.NOT.D_Pin_Drive_Enable        ; and OR in the motor pin to turn
transistor on
        STA     P_PortC                 ;       ; and motor off
        JMP     L_PWM_Drive_Done

L_PWM_Drive_On:
        LDA     P_PortC                 ;
        ora     #D_Pin_Drive_Enable
        STA     P_PortC                 ;


L_PWM_Drive_Done:                       ;



 ;PWM for steering motor
        lda     R_Steer_PWM
        beq     L_PWM_Steer_Off

        lda     R_PWM_Counter
        cmp     R_Steer_PWM
        bcc     L_PWM_Steer_On

L_PWM_Steer_Off:

        lda     P_PortD
        and     #.NOT.D_Pins_Steer
        sta     P_PortD

        jmp     L_PWM_Steer_Done

L_PWM_Steer_On:

        lda     P_PortD
        and     #.NOT.D_Pins_Steer
        ora     R_Steer_Dir
        sta     P_PortD

L_PWM_Steer_Done:

        RTS


; ********************** Interrupt Service Routine
```

Shft96Rx.ASM

```
******************************************

V_Irq:

        STA     R_TempA                 ; save accumulator value
        STX     R_TempX                 ; save x value
        lda     #C0H                    ; clear the interrupt flags
        STA     P_Ints

V_Nmi:                          ; non maskable interrupt--Sunplus does not support
this code
                                ; very well, and we have been warned not to use any
or mess
                                ; with it
        LDA     R_TempA
        LDX     R_TempX
        RTI

        ;.Include        Channel.asm

        .DB             'PEND',0                ; no idea what this is

        ; Vectors settings - do not change  (from Sunplus Demo Code)

        .ORG    7FFAH
        DW      V_Nmi
        DW      V_Reset
        DW      V_Irq

        .ORG    FFFAH
        DW      V_Nmi
        DW      V_Reset
        DW      V_Irq
        END
```

What is claimed:

1. A toy vehicle remote control transmitter unit comprising:

a housing;

a plurality of manual input elements mounted on the housing for manual movement;

a microprocessor in the housing operably coupled with each manual input element on the housing;

a signal transmitter operably coupled with the microprocessor to transmit wireless control signals generated by the microprocessor; and

wherein the microprocessor is configured for at least two different modes of operation, the microprocessor being configured in one of the at least two different modes of operation to emulate manual transmission operation of the toy vehicle by being in any of a plurality of different gear states and to transmit through the transmitter forward propulsion control signals representing different toy vehicle speed ratios for each of the plurality of different gear states, the microprocessor further being configured to be at least advanced through the plurality of different consecutive gear states by successive manual operations of at least one of the manual input devices.

2. The remote control transmitter unit of claim 1 wherein the microprocessor is configured to further generate the forward propulsion control signals for the toy vehicle in response to manual operations of the one manual input device.

3. The remote control transmitter unit of claim 2 wherein the microprocessor is further configured to respond to two successive changes of state of the one manual input element within a predetermined period of time to change a current gear state of the microprocessor to a next consecutive gear state.

4. The remote control transmitter unit of claim 1 further comprising a sound generation circuit with a speaker controlled by the microprocessor and wherein the microprocessor is programmed to generate sound effects controlled at least in part by the current gear state of the microprocessor.

5. The remote control transmitter unit of claim 1 wherein the microprocessor is configured to respond to a propulsion input element of the plurality of manual input elements to generate the forward propulsion control signals for the toy vehicle and wherein the microprocessor is configured for at least a second mode of operation wherein the microprocessor responds to the propulsion input element to generate only a single forward propulsion control signal with a maximum forward speed ratio of the toy vehicle under any mode of operation of the remote control transmitter unit.

6. The remote control transmitter unit of claim **14** wherein the forward propulsion control signals generated by the microprocessor include at least a variable duty cycle component, each transmitted duty cycle component corresponding to one of a plurality of predetermined speed ratios of the toy vehicle.

7. The remote control transmitter unit of claim 6 in combination with the toy vehicle, the toy vehicle including a receiver circuit, a toy vehicle microprocessor coupled with the receiver circuit, a variable speed steering motor and a variable speed propulsion motor, each motor being operably coupled with the vehicle microprocessor, and the vehicle microprocessor being configured to operate the variable speed propulsion motor at a duty cycle corresponding to the variable duty cycle component of the propulsion control signals.

8. The combination of claim 7 wherein the remote control unit microprocessor is configured to generate and transmit steering control signals to the toy vehicle and wherein the toy vehicle microprocessor is configured to control the steering motor in response to the steering command signals and to a current steering position of the toy vehicle.

9. The combination of claim 8 wherein the microprocessor is further configured to control the steering motor at a first speed where a new steering position in a steering control signal is adjacent to a current steering position of the toy vehicle and at second speed greater than the first speed where the new steering position is other than adjacent to the current steering position.

* * * * *