Happ Controls

Manufacturer of Electronic Controls

Specification and Features
Happ UGCI™

For the latest information on this and our other products visit
WWW.HAPPCONTROLS.COM

Disclaimer: Although every effort is made to be accurate, Happ Controls cannot assume any responsibility whatsoever for errors or omissions in this or any other document or software relating to the UGCI or the UGCI SDK.

© 1999 Happ Controls Incorporated
© 1999 R0R3 Software Incorporated

INS-0037

1.0 The UGCI

Thank you for the purchase of your Happ UGCI. The Happ UGCI is the most comprehensive industrial–strength game interface on the market today. The Happ UGCI provides all of the features of a full arcade and amusement interface with the compliment of robust and fast 15 mega-bit USB bus. Leading the technology in USB devices, the UGCI packs every coin-op feature, with the added benefit of security and watch-dog options. The Economical UGCI provides a Plug-And-Play WINDOWS 98/ME/2000 device in a ready to use game interface. The UGCI eliminates the need for the archaic and complicated/expensive "JAMMA" wiring standard by packing the electrical interface in a straightforward USB bus cabling scheme. Thus the UGCI can be mounted directly adjacent to the control panel and the JAMMA discarded entirely. For kit applications this solves the direct problem of rewiring the entire cabinet when a game is changed into old wood.

UGCI Feature List

Happ USB Game Interface Board (UGCI)	Joystick(s)	Trackball(s)	Coin Counter(s)	Buttons Per Joystick	Outputs (Med and Hi current outputs require optional Driver Board)
Driving	Steering Wheel, Throttle, Brake and Clutch	One two axis trackball/mouse interface with 3-Buttons	Two Coin/Bill Counter plus two Start Buttons	Six, and 4-speed shifter plus reverse	14 Med. and two High Current Lamp or Solenoid Drivers with status LED and WatchDog CPU reset
Flying	Two 2-Axis Analog with Rudder Pedals and Throttle plus eight position POV Hat switch	One two axis trackball/mouse interface with 3-Buttons	Two Coin/Bill Counter plus two Start Buttons	Eleven not including POV hat.	14 Med. and two High Current Lamp or Solenoid Drivers with status LED and WatchDog CPU reset
Fighting	Two 8/4-position digital joysticks	One two axis trackball/mouse interface with 3-Buttons	Two Coin/Bill Counters Plus two Start Buttons	Six Buttons Joystick 1 Seven Buttons Joystick 2	14 Med. and two High Current Lamp or Solenoid Drivers with status LED and WatchDog CPU reset

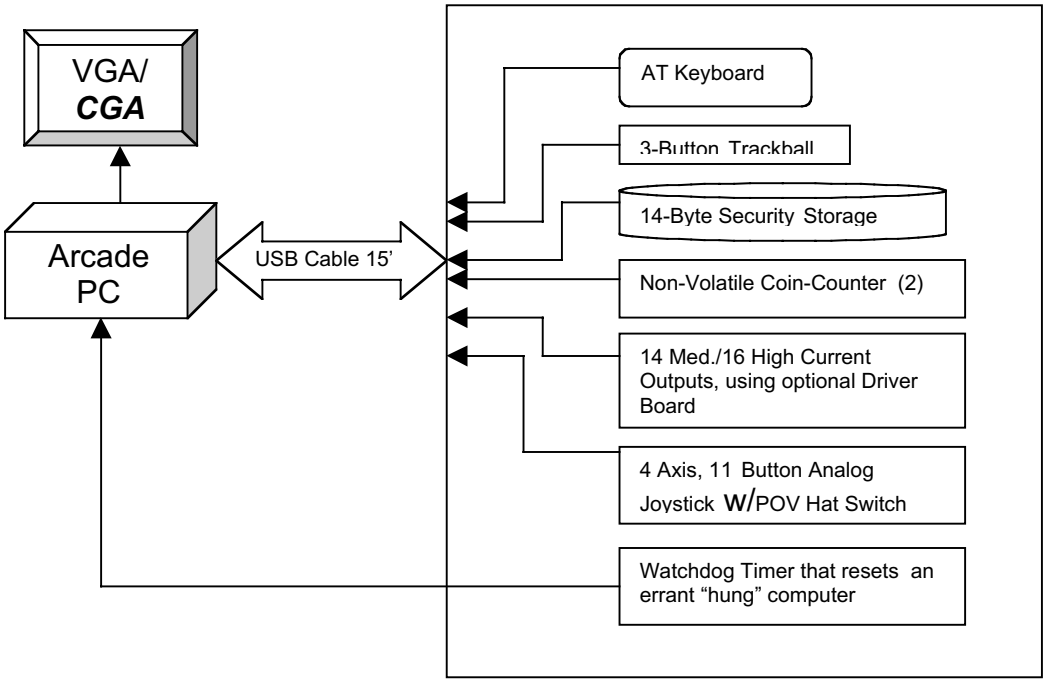
The UGCI consists of a base PC board platform and three interchangeable firmware chips. Each chip has a different personality programmed into it which suits a particular game market segment: The purchase of a UGCI contains a single chip which can be ordered with the following part number:

UGCI	UGCI Chip Number	USB VID-PID Number
Driving	96-0744-10	078B-0010
Flying	96-0744-20	078B-0020
Fighting(stick)	96-0744-30	078B-0030

Each of the three chips contains a product unique USB VID-PID to identify a particular feature(s) in the system. The UGCI is completely USB HID compliant and therefore eliminates the need for a secondary keyboard and trackball to boot the PC based location arcade game. Among the features of the UGCI are:

- A single USB port interface for all devices thus eliminating the JAMMA interface.
- Control panel wiring Harness (included).
- Two coin drop inputs per board with a non-volatile game accessible coin counter - standard. Each input also accepts standard Coin/Bill Counters
- Two Player/Start inputs per board standard
- One high performance optical trackball 2-axis, three button interface per UGCI.
- Read/Writeable 14 byte Security and non-volatile system storage to protect the game program from intellectual theft.
- Up to 504 bytes of user programmable EEPROM (Note: boards older than rev C will have only 120 bytes. Must have firmware version 2.00 and higher.)
- Programmable button interface that programs a button to a keyboard SCAN code key.
- Eliminates the need for a keyboard and mouse attached to the PC for boot purposes thus eliminating the additional expense.
- 2 high current and 14 medium current outputs with the addition of a daughter board.
- Fast, high–performance joystick interface.
- 120 Hz Packet rate at 1.5 Mbits/Second
- High Performance Universal Serial Bus (USB) IO Channel.
- Low cost
- Robust fault tolerant design that is USB HID compliant.
- ***DirectX*** © Ready.
- Built in WINDOWS 98™ and WINDOWS 2000™ Operating System support
- Eight high speed 8-bit 1 Khz analog channels.
- 32 Digital inputs at 10 Khz Each.
- CPU resource efficient 10 times faster than RS-232 base arcade models
- 120 Hz polling frequency for each data packet input.
- Plug-And-Play and ACPI Compliant
- Host CPU Reset watch-dog timer

Base System UGCI and ARCADE PC (Flying UGCI)



1.1 HID Compliance

The UGCI is a full-blown HID compliant system. HID compliance does not require a device driver to be supplied with every UGCI and is supported by the operating system. Each component in the UGCI complies with HID version 1.0.

2.1 Features of Driving UGCI

VID-PID: 078B-010
Functions: Driving, Coin-Drop, Watchdog, Trackball, Serial Number, Daughter Board

For the driving UGCI either an analog pot or switch can be applied to the analog input for brake, throttle and clutch. The analog inputs are pulled high with a 470K resistor so the maximum input DC impedance is 5K ohms. The value is reported as an analog input in the HID report descriptor.

2.2 Features of Flying UGCI

VID-PID: 078B-020
Functions: Flying, Coin-Drop(s), Watchdog, Trackball, Serial Number, Daughter Board

For the Flying UGCI either an analog pot or switch can be applied to the analog input for brake, throttle and clutch. The analog inputs are pulled high with a 470K resistor so the maximum DC input impedance is 5K ohms. For games that utilize a throttle and rudder where no physical device is present, the rudder/throttle inputs should be tied to ground to prevent spurious noise.

2.3 Features of Fighting UGCI, a Two Joystick system

VID-PID: 078B-030
Functions: JS2, JS1, Coin-Drop(s), Trackball, Watchdog, Serial Number, Daughter Board

For the Fighting UGCI a switch can be applied to the analog input for joystick buttons. The analog inputs are pulled high with a 470K resistor so the maximum DC input impedance is 5K ohms. The JS1 and JS2 are digital 8 position joysticks with the input to the UGCI tied to with the NO contact. A complete wiring harness is supplied with the UGCI.

***Note: Some versions of the UGCI have a DIP switch. On these boards all the switches should be set to the OFF position.**

3.0 IO List Pin Orientation

IC	Flying	Fighting	Driving	Connects
U1-P0-0	TB Button 1 (L)	TB Button 1 (L)	TB Button 1 (L)	J2-4
U1-P0-1	Coin 1	Coin 1	Coin 1	J3-1
U1-P0-2	Player 1	Player 1	Player 1	J3-2
U1-P0-3	X	X	X	J1-2
U1-P0-4	X'	X'	X'	J1-3
U1-P0-5	Y	Y	Y	J1-7
U1-P0-6	Y'	Y'	Y'	J1-8
U1-P0-7	TB Button 3 (R)	TB Button 3 (R)	TB Button 3 (R)	J2-2
U1-P1-0	TB Button 2 (M)	TB Button 2 (M)	TB Button 2 (M)	J2-3
U1-P1-1	POVN	JS1N	Button-8	J4-1
U1-P1-2	POVE	JS1W	Button-9	J4-2
U1-P1-3	POVS	JS1S	Button-10	J4-3
U1-P1-4	POVW	JS1E	Button-11	J4-4
U1-P1-5	JSB-1	JS1B-1	Button-1	J4-6
U1-P1-6	JSB-2	JS1B-2	Button-2	J4-7
U1-P1-7	JSB-3	JS1B-3	Button-3	J4-8
U1-P2-0	CS-U3	CS-U3	CS-U3	CS-U3
U1-P2-1	JSB-5	JS2B-1	Button-5	J5-1
U1-P2-2	JSB-6	JS2B-2	Button-6	J5-2
U1-P2-3	JSB-7	JS2B-3	Button-7	J5-3
U1-P2-4	JSB-8	JS2N	Button12	J5-4
U1-P2-5	JSB-9	JS2W	Button-13	J5-5
U1-P2-6	JSB-10	JS2S	Button-14	J5-7
U1-P2-7	JSB-11	JS2E	Button-15	J5-9
U1-P3-0	CS-1 Driver PCB/LED	CS-1 Driver PCB/LED	CS-1 Driver PCB/LED	J7-4/LED D1
U1-P3-1	CS-2 Daughter	CS-2 Daughter	CS-2 Daughter	J7-5
U1-P3-2	CS-3 A/D	CS-3 A/D	CS-3 A/D	U2-16
U1-P3-3	Data In	Data In	Data In	
U1-P3-4	Data Out	Data Out	Data Out	J7-7
U1-P3-5	Clock	Clock	Clock	J7-2
U1-P3-6	WD	WD	WD	J8-1
U1-P3-7	Coin 2	Coin 2	Coin 2	J3-5
U2-A/D-0	X-Axis	JS2B-7	X-Axis (Wheel)	J6-3
U2-A/D-1	Y-Axis	JS2B-6	Y-Axis (Throttle)	J6-4
U2-A/D-2	Rudder	JS2B-5	Throttle	J6-5
U2-A/D-3	Throttle	JS2B-4	Rudder	J6-6
U2-A/D-4	Player 2	Player 2	Player 2	J3-6
U2-A/D-5	N/C	JS1B-6	Brake	J6-9
U2-A/D-6	N/C	JS1B-5	N/C	J6-10
U2-A/D-7	JSB-4	JS1B-4	Button 4	J4-9

NOTE: In the Driving UGCI, the analog controls in your game may be defined differently than above. The above usages are per the HID spec.

Connector Tables

Connector J1 Trackball	*Connects*
PIN1	+5V
PIN2	X
PIN3	X'
PIN4	GND
PIN5	GND
PIN6	GND
PIN7	Y
PIN8	Y'
PIN9	NC
PIN10	+5

Connector J2 Trackball Buttons	Connects
PIN1	GND
PIN2	TB Button 3 (R)
PIN3	TB Button 2 (M)
PIN4	TB Button 1 (L)
PIN5	NC
PIN6	GND

Connector J3 Start	Connects
PIN1	Coin 1
PIN2	Player 1
PIN3	NC
PIN4	GND
PIN5	Coin 2
PIN6	Player 2
PIN7	GND
PIN8	GND

Connector J4	Flying	Fighting	Driving
PIN1	POVN	JS1N	1
PIN2	POVE	JS1W	2
PIN3	POVS	JS1S	3
PIN4	POVW	JS1E	4
PIN5	GND	GND	GND
PIN6	JSB-1	JS1B-1	B1
PIN7	JSB-2	JS1B-2	B2
PIN8	JSB-3	JS1B-3	B3
PIN9	JSB-4	JS1B-4	B4
PIN10	NC	NC	NC
PIN11	GND	GND	GND
PIN12	GND	GND	GND

Con. J5	Flying	Fighting	Driving
PIN1	JSB-5	JS2B-1	B-5
PIN2	JSB-6	JS2B-2	B-6
PIN3	JSB-7	JS2B-3	B-7
PIN4	JSB-8	JS2N	Reverse
PIN5	JSB-9	JS2W	Switch-Type Clutch
PIN6	GND	GND	GND
PIN7	JSB-10	JS2S	5th
PIN8	NC	NC	NC
PIN9	JSB-11	JS2E	
PIN10	GND	GND	GND

Con. J6	Flying	Fighting	Driving
PIN1	+5	+5	+5
PIN2	NC	NC	NC
PIN3	X	JS2-B7	X-Axis (Wheel)
PIN4	Y	JS2-B6	Y-Axis (Throttle)
PIN5	Rudder	JS2-B5	Throttle
PIN6	Throttle	JS2-B4	Rudder
PIN7	GND	GND	GND
PIN8	GND	GND	GND
PIN9		JS1-B6	Brake
PIN10		JS1-B5	
PIN11		Player 2	
PIN12	+5	+5	+5

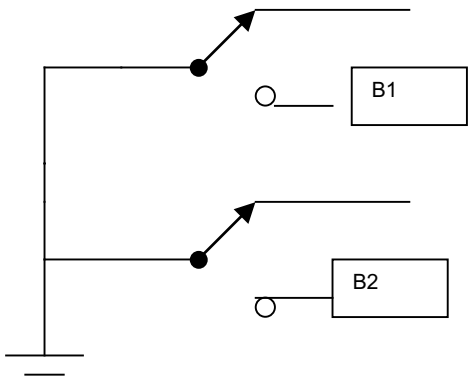
NOTE: In the Driving UGCI, the analog controls in your game may be defined differently than above. The above usages are per the HID spec.

Connector J7	Connects
PIN1	+5
PIN2	Clock
PIN3	GND
PIN4	CS-1
PIN5	CS-2
PIN6	NC
PIN7	Data Out
PIN8	GND

Connector J8 Watchdog Reset	Connects
PIN1	WD
PIN2	GND

3.1 Key/Button Electrical Connections

The button is read utilizing the NO and NC features of the standard Happ input buttons thus eliminating the need for IO scanning e.g.:



3.1 Features of the Happ USB Game Control Interface

Three distinct models:

- ***Driving UGCI*** 3-Button Trackball. Steering wheel, brake, rudder, throttle. Two Coin/Bill-drop and two start buttons. Up to fifteen general purpose buttons. General purpose EEPROM. Mapping of buttons to keyboard scan codes. Host watch-dog timer. 2 high current and 14 medium current Lamp Drivers with daughter board.
- ***Flying UGCI*** 3-Button Trackball. Analog B-8 Joystick, with POV hat. Rudder pedal, Throttle and two coin-drops. Two Coin/Bill-drop and two start buttons. Up to fifteen general purpose buttons. General purpose EEPROM. Mapping of buttons to keyboard scan codes.
- ***Fighting UGCI***: 3-Button Trackball. Two eight position digital joysticks. Up to six buttons for each joystick. Two Coin/Bill-drop and two start buttons. Up to fifteen general purpose buttons. General purpose EEPROM. Mapping of buttons to keyboard scan codes.

4.0 Packet Format and Protocol for the UGCI

The packets are delivered to the host system when change is detected in the composite device interface. A composite device is identified by a leading Report ID byte followed by one or more trailing bytes. Some of the devices on the UGCI do not have a leading report ID byte. Use the following table(s) to determine the report format. The trailing bytes have the format as outlined in the following tables.

4.1 Report Format and Operation, Watchdog Composite (All models)

The watchdog timer resets an out of control game application or system. The watchdog reset will be asserted under the following conditions:

- Watchdog timeout, the game/host fails to send the watchdog refresh packet or the internal timer elapses during boot.

4.1.1 Watchdog Behavior (Runtime)

The watchdog timeout value is stored in non-volatile memory. The boot behavior differs slightly from the runtime operation. Runtime operation is indicated by a reception of the first watchdog refresh packet. The packet contains a 16-bit timer value (0-65535 seconds) which must be refreshed by the host prior to elapse of the timer. The timer value begins decrementing immediately after reception of a watchdog packet. The Watchdog packet has the following format:

Watchdog Packet Format (all versions) Write on Endpoint 0

Byte 0	Byte 1	Byte 2	Byte 3
Report ID = 6	Low Order Byte	High Byte	Action

The watchdog packet is sent on the control point by doing a bulk write to endpoint 0. See the UGCI SDK for information on how to do this. The byte format is little-endian. Writing a value of zero to the timeout turns the watchdog feature off. Any other value refreshes the watchdog timer and begins decrementing an internal one-second timer. When the timer value reaches zero, PIN 1 of J8 is asserted for 1000 ms thus re-booting the host. The watchdog reset output pin is an open–drain output. The watchdog reset cable must be a twisted pair with the shield connected at the host ground to prevent noise. The watchdog cable should be connected to the Reset pin on the PC motherboard.

The action byte (3) determines which behavior to process, either boot =2 or runtime =1. If runtime is selected, the device will expect a refresh packet prior to the timeout period, and the watchdog is initially off until the very first watchdog packet in the session is received with a non-zero watchdog packet value. If boot (2) is selected, the watchdog value is written to non-volatile memory. This value is used during boot, prior to instantiating the WINDOWS or the host operating system. The watchdog will start running after the device is reset. The watchdog will then begin decrementing the boot timeout and will reset the host if the timeout occurs prior to the host sending the first watchdog packet. This is to prevent errant behavior during boot. If boot (2) is selected, the minimum time for the watchdog will be 20 seconds. If a shorter time is sent to the UGCI, then that time will be used for the first timeout, however, after the device resets the host the timeout will be set to 20 seconds.

The UGCI is shipped from Happ with the watchdog off.

4.2 Report Format and Operation, Coin-Op(s)

Each UGCI model contains two internal software based non-volatile 16-bit coin counters and two player start buttons. The internal coin-counters are incremented once each time an active low pulse of duration 50ms is asserted at the inputs. The inputs to the coin counters are TTL/CMOS, Open Drain CMOS or a NO switch compatible. The coin counters (J3-1 and J3-5) are compatible with and have been tested with MARS bill acceptors, various coin-op mechanisms and card readers. The coin-counters are completely de-bounced to prevent spurious noise from inadvertently tripping the counters. The coin counters eliminate the need for a mechanical counter in the cabinet because each coin

event stores the current coin-count in the UGCI non-volatile memory, which is then available to the game application

NOTE: Voltage levels higher than 5-volts on J3-1 and J3-5 will damage the UGCI.

4.2.1 Reading the Coin Count

Coin-op packets are sent to the host every time a coin or bill is inserted in the host or, when the player start button is pressed or released. All three models of the UGCI share the same coin-op packet format:

Coin-OP Packet Format, USB Endpoint 1, (all models)

Byte 0	*Byte 1*	*Byte 2*	*Byte 3*
RID = 3 J3-1 – Coin 1 J3-2 – Start 1	Coin Count LSB	Coin Count MSB	Player Buttons 1=Pressed, 0=Released
RID = 4 J3-5 – Coin 2 J3-6 – Start 2	Coin Count LSB	Coin Count MSB	Player Buttons 1=Pressed, 0=Released

If the bill-acceptor or card reader is configured to assert three pulses into the J3-1 for instance, the coin-count LSB and MSB will be incremented three times as a 16-bit integer. The coin count value is then written into non-volatile memory. The coin count values cannot be reset and will rollover at 64K. The coin count value is sent to the host once after the host Operating System enumerates. See the SDK for information on how to read the coin count. The coin count value will be maintained in non-volatile memory after boot. The non-volatile memory has a life expectancy of over 100 years and millions of cycles.

The coin inputs are written during host boot if they are activated during that time. This will facilitate the loss of play information should the player insert coins during that time.

4.3 Security and ID (all models)

A 14-byte non-volatile security buffer is provided on all models of the UGCI for storage of private data. The security buffer can be used to store game information such as security and protection should the illegal theft of the game occur. The game could then use this information to validate the platform. The security information is transmitted as two separate 8-byte packets, and read as the same. The security information is written as two separate packets on endpoint 0 as a bulk write, and read as two separate values on a host read (endpoint 1):

Security Buffer (write on Endpoint 0)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=9	Char 0	Char 1	Char 2	Char 3	Char 4	Char 5	Char 6
RID=10	Char 7	Char 8	Char 9	Char 10	Char 11	Char 12	Char 13

4.4 Security Buffer Read

The security buffer is read on endpoint 0.

Security Buffer (read on endpoint 1)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=11	Char 0	Char 1	Char 2	Char 3	Char 4	Char 5	Char 6
RID=12	Char 7	Char 8	Char 9	Char 10	Char 11	Char 12	Char 13

4.5 Trackball Interface (all models)

Each model contains a complete high-performance 3-button game ready trackball interface. The axis information is transmitted as two separate signed 8-bit values. The trackball interface has it's own endpoint (endpoint 2) which it does not share with any other composite device. To facilitate high performance, the trackball transmits at 120 Hz.

Trackball Packet

Byte 0	Byte 1	Byte 2
X-Axis, Range = -127 to 127	Y-Axis, Range = -127 to 127	Bit 0 = Button1 (Left) Bit 1 = Button 2(Right) Bit 2 = Button 3 (Middle) Bit 4 – 7 (not used)

4.5.1 Boot Behavior

To save cost to the game developer, the UGCI trackball contains a boot descriptor that emulates a serial or PS-2 mouse, if the PC BIOS supports this feature. Most PC BIOS's support the USB boot protocol for trackballs and mice. The trackball should function under DOS using the BIOS mouse interrupt interface (see your DOS information).

4.5.2 Trackball Runtime Behavior

For a PC arcade platform, no additional mouse is required in the cabinet to boot WINDOWS. The UGCI begins sending out trackball reports as the ball is moved and the buttons are pressed. A trackball report is sent on every trackball event, this includes buttons pressed and released. The electrical interface will support all Happ trackballs.

4.6 Keyboard Emulator (All Models) and Key Mapping

To eliminate the need for an additional keyboard in the cabinet, the UGCI provides keyboard emulation. The emulator consists of a USB keyboard boot descriptor that fools the BIOS into thinking that a physical keyboard is attached to the computer. The keyboard emulator is shared on endpoint 1. Because of this a large programmable delay must be inserted before the first report is sent to prevent report transmissions with the report id attached at boot-up. This prevents some BIOSs from locking up the system thinking that a key has been pressed. The boot delay is configurable using the following report:

Keyboard Emulator Boot Mode/Delay

Byte 0	Byte 1	Byte 2
RID=16	Mode: 0 = No Boot Keyboard (turns off boot descriptor) 1 = HID Compliant 2 = Use Boot descriptor with delay	Boot Delay: 0-255 in seconds

There are 3 modes for the keyboard device that can be configured by sending a report to the UGCI:

1) Non Bootable
In this mode there is no boot descriptor for the keyboard.

2) HID compliant bootable keyboard
There are two protocols for reporting keyboard data to the host: boot protocol and report protocol. Since the keyboard shares the endpoint with other devices a report ID is needed for each device. In report mode the keyboard will send a report ID along with the data. The UGCI defaults to report mode on power up (this is as per the HID spec). When the PC boots up the BIOS is supposed to send a message to the UGCI to put the device into boot protocol. In boot protocol the format for the keyboard report is as per the HID spec (which does not include a report ID). After the OS loads, the host is then supposed to put the device into report protocol.

3) Boot Protocol on Power Up or Bus Reset with a Configurable Delay.

In this mode the UGCI will default to boot protocol on power up or bus reset, and stay in boot protocol for a configurable time period. At the end of this time period the device will switch to report protocol. This does not conform to the HID spec, however it does allow PCs that don't have full BIOS support for a bootable USB Keyboard to be used with the UGCI. The user should be aware that if the device sees a bus reset, no joystick data will be available to the application until after the programmable time period.

4.6.1 Key Mapping

The system keyboard emulation allows the mapping of keyboard keystrokes to the joystick and trackball buttons. The buttons are coded in HID usage keycodes and the usages are available in the HID 1.0 Usage specification available on the WWW.USB.ORG\DEVELOPERS web page. The SDK outlines the usages for key scan codes and demonstrates the use of keycodes. The report descriptor has the following format:

Keyboard Emulation, Key Press/Release Report Format

Byte 0	Byte 1	Byte 2	Byte 3
RID= 14	Modifier Byte This byte contain CTRL, SHIFT, ALT modifiers (in usage code)	(not used)	Key (usage code) that was pressed: "a"= 04h "b"= 05h etc..

Each time a key is pressed a report is sent with the appropriate modifiers set and key codes set. Individual Joystick and Trackball Buttons can also be mapped to key scan-codes. These codes are programmed into non-volatile memory and the keyboard report will be sent when the corresponding button is pressed. Starting with firmware version 2.00 and greater, the keyboard repeat rate and repeat delay time can also be set. The values for repeat rate and repeat delay are set in 4 msec increments with a range of 0 –255 (0 – 1.020 seconds). Key to Button mapping is handled in the following reports:

Report Format for Button Mapping (Prior to firmware ver 2.00)

Keyboard Mapping 1-7 (Output, 8 Bytes) Flying UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=17	Button 1 Mapping Scan Code	Button 2 Mapping Scan Code	Button 3 Mapping Scan Code	Button 4 Mapping Scan Code	Button 5 Mapping Scan Code	Button 6 Mapping Scan Code	Button 7 Mapping Scan Code

Keyboard Mapping 2-8 (Output, 8 Bytes) Flying UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=18	Button 8 Mapping Scan Code	Button 9 Mapping Scan Code	Button 10 Mapping Scan Code	Button 11 Mapping Scan Code	(not used)	(not used)	(not used)

Keyboard Mapping 1-7 (Output, 8 Bytes) Fighting UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=17	JS1 Button 1 Mapping Scan Code	JS1 Button 2 Mapping Scan Code	JS1 Button 3 Mapping Scan Code	JS1 Button 4 Mapping Scan Code	JS1 Button 5 Mapping Scan Code	JS1 Button 6 Mapping Scan Code	(not used)

Keyboard Mapping 2-8 (Output, 8 Bytes) Fighting UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=18	JS2 Button 1 Mapping Scan Code	JS2 Button 2 Mapping Scan Code	JS2 Button 3 Mapping Scan Code	JS2 Button 4 Mapping Scan Code	JS2 Button 5 Mapping Scan Code	JS2 Button 6 Mapping Scan Code	JS2 Button 7 Mapping Scan Code

Keyboard Mapping 1-7 (Output, 8 Bytes) Driving UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=17	Button 1 Mapping Scan Code	Button 2 Mapping Scan Code	Button 3 Mapping Scan Code	Button 4 Mapping Scan Code	Button 5 Mapping Scan Code	Button 6 Mapping Scan Code	Button 7 Mapping Scan Code

Keyboard Mapping 2-8 (Output, 8 Bytes) Driving UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=18	Button 8 Mapping Scan Code	1st Mapping Scan Code	2nd Button 3 Mapping Scan Code	3rd Button 4 Mapping Scan Code	4th Button 5 Mapping Scan Code	5th Mapping Scan Code	Reverse Mapping Scan Code

Keyboard Mapping 9-15 (Output, 8 Bytes) Driving UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=19	Clutch	(not used)	TB Butt 0 (left)	TB Butt 1 (middle)	TB Butt 2 (right)	(not used)	(not used)

Keyboard Mapping 9-15 (Output, 8 Bytes) Flying, Fighting UGCI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RID=19	(not used)	(not used)	TB Butt 0 (left)	TB Butt 1 (middle)	TB Butt 2 (right)	(not used)	(not used)

Report Format for Button Mapping (firmware ver 2.00 and greater)

Starting with firmware ver 2.00 the Key to Button Mapping occupies the first 51 bytes of the general purpose EEPROM space. If key mapping is enabled, then the first 51 bytes of EEPROM are unavailable to the general purpose EEPROM write function. Byte 0 of the general purpose EEPROM contains a flag byte that allows the key mapping to be enabled and disabled. The report format for the key mapping takes a start address in EEPROM to begin writing, a number of bytes to write, and a variable number of data bytes. The format is as follows:

Keyboard Mapping (All models)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	…	Byte N
RID=4Bh ('K')	Start address in EEPROM	Number of bytes to write (51 Max)	Data Byte 1	Data Byte 2	…	Data Byte N

The EEPROM addresses for key mapping are defined as follows:

Byte 0: Flag Byte
Bit 0: 0 = Key Mapping Disabled
1 = Key Mapping Enabled
Bit 1-7:reserved (set to 0)

Byte 1-48: These bytes will be change depending on the UGCI.

For the Flying UGCI they are as follows:

Byte	Function
1	Mouse Left Button key usage
2	Mouse Right Button key usage
3	Mouse Middle Button key usage
4	Joystick Button 1 key usage
5	Joystick Button 2 key usage
6	Joystick Button 3 key usage
7	Joystick Button 4 key usage
8	Joystick Button 5 key usage
9	Joystick Button 6 key usage

Byte	Function
10	Joystick Button 7 key usage
11	Joystick Button 8 key usage
12	Joystick Button 9 key usage
13	Joystick Button 10 key usage
14	Joystick Button 11 key usage
15	POV Hat N key usage
16	POV Hat E key usage
17	POV Hat S key usage
18	POV Hat W key usage
19	not used
20	not used
21	not used
22	not used
23	not used
24	not used
25	Mouse Left Button key modifier
26	Mouse Right Button key modifier
27	Mouse Middle Button key modifier
28	Joystick Button 1 key modifier
29	Joystick Button 2 key modifier
30	Joystick Button 3 key modifier
31	Joystick Button 4 key modifier
32	Joystick Button 5 key modifier
33	Joystick Button 6 key modifier
34	Joystick Button 7 key modifier
35	Joystick Button 8 key modifier
36	Joystick Button 9 key modifier
37	Joystick Button 10 key modifier
38	Joystick Button 11 key modifier
39	POV Hat N key modifier
40	POV Hat E key modifier
41	POV Hat S key modifier
42	POV Hat W key modifier
43	not used
44	not used
45	not used
46	not used
47	not used
48	not used

For the Driving UGCI they are as follows:

Byte	Function
1	Mouse Left Button key usage
2	Mouse Right Button key usage
3	Mouse Middle Button key usage
4	Joystick Button 1 key usage
5	Joystick Button 2 key usage
7	Joystick Button 4 key usage

Byte	Function
8	Joystick Button 5 key usage
9	Joystick Button 6 key usage
10	Joystick Button 7 key usage
11	Joystick Button 8 key usage
12	Joystick Button 9 key usage
13	Joystick Button 10 key usage
14	Joystick Button 11 key usage
15	Joystick Button 12 key usage
16	Joystick Button 13 key usage
17	Joystick Button 14 key usage
18	Joystick Button 15 key usage
19	not used
20	not used
21	not used
22	not used
23	not used
24	not used
25	Mouse Left Button key modifier
26	Mouse Right Button key modifier
27	Mouse Middle Button key modifier
28	Joystick Button 1 key modifier
29	Joystick Button 2 key modifier
30	Joystick Button 3 key modifier
31	Joystick Button 4 key modifier
32	Joystick Button 5 key modifier
33	Joystick Button 6 key modifier
34	Joystick Button 7 key modifier
35	Joystick Button 8 key modifier
36	Joystick Button 9 key modifier
37	Joystick Button 10 key modifier
38	Joystick Button 11 key modifier
39	Joystick Button 12 key modifier
40	Joystick Button 13 key modifier
41	Joystick Button 14 key modifier
42	Joystick Button 15 key modifier
43	not used
44	not used
45	not used
46	not used
47	not used
48	not used

For the Fighting UGCI they are as follows:

Byte	Function
1	Mouse Left Button key usage
2	Mouse Right Button key usage
3	Mouse Middle Button key usage
4	Joystick 1 N key usage

Byte	Function
5	Joystick 1 E key usage
6	Joystick 1 S key usage
7	Joystick 1 W key usage
8	Joystick 2 N key usage
9	Joystick 2 E key usage
10	Joystick 2 S key usage
11	Joystick 2 W key usage
12	Joystick 1 Button 1 key usage
13	Joystick 1 Button 2 key usage
14	Joystick 1 Button 3 key usage
15	Joystick 1 Button 4 key usage
16	Joystick 1 Button 5 key usage
17	Joystick 1 Button 6 key usage
18	Joystick 2 Button 1 key usage
19	Joystick 2 Button 2 key usage
20	Joystick 2 Button 3 key usage
21	Joystick 2 Button 4 key usage
22	Joystick 2 Button 5 key usage
23	Joystick 2 Button 6 key usage
24	Joystick 2 Button 7 key usage
25	Mouse Left Button key modifier
26	Mouse Right Button key modifier
27	Mouse Middle Button key modifier
28	Joystick 1 N key modifier
29	Joystick 1 E key modifier
30	Joystick 1 S key modifier
31	Joystick 1 W key modifier
32	Joystick 2 N key modifier
33	Joystick 2 E key modifier
34	Joystick 2 S key modifier
35	Joystick 2 W key modifier
36	Joystick 1 Button 1 key modifier
37	Joystick 1 Button 2 key modifier
38	Joystick 1 Button 3 key modifier
39	Joystick 1 Button 4 key modifier
40	Joystick 1 Button 5 key modifier
41	Joystick 1 Button 6 key modifier
42	Joystick 2 Button 1 key modifier
43	Joystick 2 Button 2 key modifier
44	Joystick 2 Button 3 key modifier
45	Joystick 2 Button 4 key modifier
46	Joystick 2 Button 5 key modifier
47	Joystick 2 Button 6 key modifier
48	Joystick 2 Button 7 key modifier

The key modifier bytes have the following format:

Bit 0:	Left Control
Bit 1:	Left Shift
Bit 2:	Left Alt
Bit 3:	Left GUI
Bit 4:	Right Control
Bit 5:	Right Shift
Bit 6:	Right Alt
Bit 7:	Right GUI
Byte 49:	Keyboard repeat rate (in 4 ms intervals)
Byte 50:	Keyboard repeat delay (in 4 ms intervals)

UGCI SDK

The SDK outlines the WINDOWS HID methods for mapping buttons. Consult the SDK for more information. The SDK provides a way for the user to access the special functions of the UGCI. The SDK is available to developers on the Happ web site WWW.HAPPCONTROLS.COM on the UGCI page in the Game Developers section.

This posting of the SDK contains no source code.

A version of the SDK containing source code can be provided to Game Developers or other UGCI users demonstrating a legitimate need for the source code. The source code provided can help show how the UGCI communicates with the PC and is intended for those writing WIN98/2000 applications using the UGCI. You will need to be a member of MSDN for the code to be of any use to you. Email technical@happcontrols.com if you wish to obtain the source code. Visit our web site for the latest documentation for the UGCI.

See section 8 for more detail.

4.7 General Purpose EEPROM (All Models, firmware ver 2.00 and greater)

Starting with firmware version 2.00, a general purpose EEPROM read/write function has been added to the UGCI. This allows the user to store and retrieve up to 504 bytes of system or security information in the UGCI. UGCI boards starting with rev C will contain 504 bytes of general purpose EEPROM. Older boards will contain only 120 bytes. The UGCI will automatically detect the size of the EEPROM. This can be read in the flag byte (byte 0 of the EEPROM). Byte 0 of the general purpose EEPROM contains some special flags. This byte is read only and cannot be written by this method. The general purpose EEPROM shares space with the key mapping and the serial number. If you have key mapping enabled, you cannot write to byte 1 – 50 of the EEPROM. If you disbale key mapping, these bytes are available to you. Bytes 51 – 503 are always available, however the last 14 bytes are shared by the serial number. If you write to these bytes, the serial number will be written over. The EEPROM read and write reports are similar to the key mapping report. The reports are the same regardless of the size of the EEPROM, and the formats are as follows:

EEPROM Write (All models)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	…	Byte N
RID=58h ('W')	Start address in EEPROM (LSB)	Start address in EEPROM (MSB)	Number of bytes to write (LSB)	Number of bytes to write (MSB)	Data Byte1	…	Data Byte N

EEPROM Read (All models)

Byte 0
RID=52h ('R')

The EEPROM read report is obtained by using the "GetFeature" API. Consult the SDK for more information on this. This report always returns 504 bytes. Bytes that don't exist (if you have only 120 bytes of EEPROM) will be returned as 0.

The EEPROM addresses are as follows:
Byte 0: Flag Byte (read only)

Bit 0: 0 = Key Mapping disabled
1 = Key Mapping Enabled
Bit 1: 0 = 128 byte EEPROM
1 = 512 byte EEPROM
Bit 2: 0 = Thru hole board
1 = Surface mount board (rev C)
Set this value using the "Set Hardware Rev Level" report
Bit 3-7: Undefined

Byte 1-50: Reserved if key mapping is enabled, Available if not
Byte 51-105: General purpose EEPROM for boards previous to rev C
Byte 51-489: General purpose EEPROM for rev C boards
Byte 106-119:Serial Number for boards previous to rev C
Byte 490-503:Serial Number for rev C boards

4.9 Set Hardware Rev Level (All models firmware rev 2.00 and greater)

The following report allows the user to tell the UGCI the hardware rev level. WARNING: this feature is intended to be set at the factory and then never modified. If this feature is set incorrectly, your UGCI may not function properly. The report is as follows:

Set Hardware Rev Level (All models)

Byte 0	Byte 1	Byte 2
RID=20h	0 = thru hole board 1 = Surface mount board (rev C)	Reserved (set tpo 0)

5.0 Analog Joystick Report (Happ Flying UGCI)

A high performance analog joystick is available with the Happ Flying UGCI. The Flying UGCI supports an eleven-button four axis joystick with a Point of View (POV) hat switch. The switches are completely debounced and the analog axes are all eight-bit filtered which use a hardware A/D converter for the conversions. The joystick is on endpoint 1 and has a maximum packet rate of 120 Hz. The packet rate is determined by the sharing of endpoint 1, but will in most cases deliver the 120 Hz rate. The packet format is as follows:

Analog Joystick Report (Flying UGCI)

Byte 0	*Byte 1*	*Byte 2*	*Byte 3*	*Byte 4*	*Byte 5*	*Byte 6*	*Byte 7*
RID = 1	X-Axis 8-bits Signed 127 to –127	Y-Axis 8-bits signed 127 to -127	Rudder 8-bits unsigned. 0 to 255 (or second joystick)	Throttle 8-bits unsigned 0 to 255 (or second joystick)	POV E=0 NE=45 N=90 NW=135 W= 180 SW=225 S=270 SE=315 S(Center =-1) Degrees.	B9 = Bit0 B10= Bit1 B11=Bit2 1= Down 0=Up	B1= Bit0 B2 = Bit1 B3 = Bit2 B4 = Bit3 B5 = Bit4 B6 = Bit5 B7 = Bit6 B8 = Bit7

5.0.1 Flying UGCI Support for Two Analog Joysticks

The Flying UGCI can support up to two analog joystick. This is accomplished by sharing the analog joystick report and utilizing the rudder and throttle inputs of the same report as the X and Y of the second joystick. The second joystick would then lack the POV hat switch. The eleven buttons could then be split so that 6 buttons are placed on the first joystick and 5 on the second joystick. This in conjunction with the Flying UGCI's two coin-op inputs and 2 start buttons, provides a complete solution for location based multi-player arcade and redemption games.

5.0.2 Flying UGI Rudder and Throttle

Under certain circumstances, the rudder and throttle must be disabled in order to prevent spurious reports from being sent to the game. In this case, the rudder and throttle must be tied to VCC on the connector.

5.0.3 Electrical Limitations of Analog Inputs

The analog inputs provide a robust and filtered input interface to the game. The input impedance is limited to less than 10K.

5.1 Output Report for Driving UGCI

A high IO or "Driving" UGCI supports up to two players in driving configurations. The Driving Game Frame natively supports the Happ Steering wheel, brake and throttle pedals as well as a forward reverse analog shifter. The Driving UGCI supports more button inputs than the flying for high IO applications. In addition, a fifth logical axis provides support for an analog clutch for multi-featured games.

The driving UGCI supports five independent analog axis and 15-digital inputs to support High IO applications. The analog portions are all 8-bits running at a maximum rate of 120 Hz for the packet. In addition, a switch contact closure can be used in lieu of a analog input and read by the game to extend the possible digital portion of up to 21 bits for the packet. With the two coin drops, start buttons, multi-player high IO applications are now feasible at a reasonable cost.

Driving UGCI Joystick Report (Driving UGCI)

Byte 0	*Byte 1*	*Byte 2*	*Byte 3*	*Byte 4*	*Byte 5*	*Byte 6*	*Byte 7*
RID = 1	(X-Axis Usage) Wheel 8-bits Signed +127 to –127	(Y-Axis Usage) 8-bits signed 127 to -127	(Throttle) 8-bits unsigned 0 to 255 (or second Wheel)	(Rudder) 8-bits unsigned 0 to 255 (or second Throttle/ Brake)	(Brake Usage) 8-bits unsigned 0 to 255 (Forward Backward)	B9= Bit0 B10 = Bit1 B11 = Bit2 B12 = Bit3 B13 = Bit4 B14 = Bit5 B15 = Bit6	B1= Bit0 B2 = Bit1 B3 = Bit2 B4 = Bit3 B5 = Bit4 B6 = Bit5 B7 = Bit6 B8 = Bit7

*The brake axis will not show up in the WINDOWS control panel. The Driving UGCI shows up as a 4-axis, 16-button Joystick.

5.1.1 Driving UGCI Rudder and Throttle

Under certain circumstances, the rudder, throttle and brake must be disabled in order to prevent spurious reports from being sent to the game. In this case, the rudder and throttle must be tied to VCC on the connector.

5.1.2 Electrical Limitations of Analog Inputs

The analog inputs provide a robust and filtered input interface to the game. The input impedance is limited to less than 10K.

6.0 The Output Driver Board Part #96-0746-00

The Happ UGCI has an optional Driver board available, part #96-0746-00 used with wire harness 96-0741-00. This board can be used to drive up to 14 open-collector 500MA (lamp) outputs and two open-collector 5 AMP (with external heatsinking) outputs with a maximum voltage of 50 VDC (60 VDC for high-current outputs). Outputs are sink drivers.

The HID output drive utilizes a special report packet. This packet turns addressable outputs on or of:

Addressable Output Reports

Byte 0	Byte 1	Byte 2
RID = 8	Bit0 = OUT0 Bit1 = OUT1 Bit2 = OUT2 Bit3 = OUT3 Bit4 = OUT4 Bit5 = OUT5 Bit6 = OUT6 *Bit7 = OUT7	Bit0 = OUT8 Bit1 = OUT9 Bit2 = OUT10 Bit3 = OUT11 Bit4 = OUT12 Bit5 = OUT13 Bit6 = OUT14 *Bit7 = OUT15

Writing a one drives the output driver in the active state (Pulled Down).
Writing a zero turns the output off (Open Circuit).
This packet is written under endpoint 0.

*High current < 5 Amps

+5 volt power is supplied to the Driver PCB from the UGCI. You must supply external power to your external loads. You cannot power your loads with UGCI power.

7.0 Installation of the Happ UGCI

This is the method for the UGCI in a WINDOWS 98//ME/2000 environment.

7.1 Installation in WINDOWS 98/ME/2000 Environment.

The following assumes that the person installing the UGCI is familiar with installing devices in a computer and is familiar with computers in general. If you are not, find someone who is.

There are no drivers or other Happ supplied system components to install in order to achieve basic functionality with the Happ UGCI. You will need the WINDOWS CD-ROM that came with your computer.

Step 1) Plug the trackball and joystick(s) into your UGCI. Do ***NOT*** plug the controls into the UGCI live or when power is applied or you can damage the board. Make sure that any metal mounting plates, ground wires, etc, of all of the input devices are grounded to an earth ground with their own ground wire. This can be your PC case. Do ***NOT*** ground through the USB power ground.

Step2) Re-boot your system and stop all running games.

Step 3) Place your WINDOWS CD in the CD-ROM drive and plug the UGCI in to the USB cable. You will get an hourglass, and a dialog box that says ”New Hardware Detected.” If you do not, check you BIOS to make sure that you computer has the USB hardware enabled – or call your computer vendor.

Step 4) Carefully follow the instructions in the dialog box. The system will query you to load the system files and drivers necessary to enable the UGCI functionality. Load every system component that the dialog recommends. This will assure that your UGCI will be fully functional. Be patient, the system takes a long time to load the files. If prompted to Restart, you can click “NO” if you wish.

Step 5) The trackball will function at this point. Move the trackball around and verify that it works. The Joysticks will now show up in the Windows Control Panel/Gaming Options. Click Properties and Calibrate to calibrate. Consult your WINDOWS documentation on how to use the control panel. You must calibrate the Joystick prior to using them in any game.

Step 6) Re-boot you computer and verify that step 5 is still valid.

You have now installed your UGCI in the WINDOWS system and are ready to use the device.

8.0 HAPP UGCI SDK Release 1.31

The SDK provides a way for the user to access the special functions of the UGCI. The SDK is available for download at WWW.Happcontrols.com on the UGCI page in our Amusement section.

This release of the SDK contains no source code. A version of the SDK containing source code can be provided to Game Developers or other UGCI users demonstrating a legitimate need for the source code. The source code provided can help show how the UGCI communicates with the PC and is intended for those writing WIN98/2000 applications using the UGCI. You will need to be a member of MSDN for the code to be of any use to you. Email Technical@Happcontrols.com if you wish to obtain the source code. Visit our web site for the latest documentation for the UGCI as well as all our products at WWW.Happcontrols.com. Visit our development partner at WWW.R0R3.com.

KNOWN ISSUES with SDK release 1.3 and 1.31:

1. The cursor (arrow) keys can not be mapped.
2. On some versions of Direct X, when using the Fighting UGCI, Joystick 1 and 2 can be reversed. It is recommended that you make sure that Joystick 1 reads as Joystick 1 in your game or application before completing your design. It may read as Joystick 2.

The UGCI SDK is intended for WIN98 only and will not work with WIN2000 or a MAC OS.

Note: Using the Coin/Bill counters, Watchdog and Player Start buttons require application software to be written for the UGCI. These are not mappable to a keyboard key. You need to be a member of MSDN (Microsoft Developers Network) and use the MSDN DDK to do this.

UGCI SDK: (Cont.)

Basics:
1. Click on HAPP TEST Icon to start.
2. FIND DEVICE button shows what HAPP devices are connected to the PC. You need to click this first, before the other features will work. Select type and click OK.
3. HW REV: Use this to select if your UGCI is a surface Mount Device board or a Thru-Hole board. You need to select or Coin 2 and WD won't work properly.
4. KEY MAP allows you to map any keyboard key A thru Z, numeric, function and control keys (ALT (L&R), TAB, CTL (L&R), ESC, ENT, SHIFT, SPACE, BKSP, joystick direction (when using switch-type joysticks and the Fighting UGCI only) or mouse buttons to a game controller button. Macros (like CTL-ESC or ALT-TAB or CTL-F) are possible.

Advanced Users and Game Developers: NOTE: Many of these features require an application specifically written to use the UGCI.
5. WATCHDOG allows the watchdog timer to be set. NOTE: This requires the use of an application specifically written to take advantage of the UGCI watchdog feature. Set the time to a minimum time necessary to allow the PC to completely boot, otherwise the watchdog will reset the PC before it can completely come to life. The watchdog connector on the UGCI is to be connected to the reset connector in the PC when using the watchdog feature with an application written for this feature. Do not connect it otherwise, or you can get stuck in an eternal boot-reset-boot loop. Set to 0 and do not check BOOT ACTION to disable.
6. SERIAL NUMBER allows the UGCI serial number to be read. It can be written by an application written to use the UGCI.
7. EXPANSION allows outputs to be manually turned on using the optional Happ UGCI Driver Board 96-0746-00. Using this in your game requires a UGCI application.
8. COIN DROP monitors the coin drop inputs. NOTE: The SDK will hang up if it does not see a coin drop after selecting the COIN DROP button. If you do not have a switch connected to a coin drop input, do not select it. Counting coins and implementing credit functions require a UGCI application.
9. TRACKBALL gives a real-time data read of the trackball position and switch status.
10. JOYSTICK gives a real-time data read of the joystick position and switch status.

UGCI SDK: (Cont.)

11. BOOT DELAY: Keyboard Boot Enable should be checked if you do not want to use a keyboard and your PC will not boot without one connected after UGCI installation. Boot Delay Enable should be checked if checking the Keyboard Boot Enable does not work. Set the time to be long enough that the PC will be completely booted. The joysticks or other controls will not operate until this time has run out. Only a very few PCs will need to use this feature. This works by having the UGCI send out a code that will trick the bios into thinking that a keyboard is connected. If everything works OK, do not check these boxes.
12. CLEAR clears the data window. Clear TRACKBALL and JOYSTICK often to see new data.
13. CLEAN REG clears the registry of all keys and data pertaining to the UGCI. This should be used when upgrading to new firmware. Reinstallation will then be necessary.

9.0 Analog Joysticks that connect directly to the UGCI:

B-8 style	95-1345-00
Right Hand Style	95-1347-00
Left Hand Style	95-1432-00
Heavy Duty 2-Button	95-1346-00
Trigger Style	95-1431-00

Appendix A
Electrical Harness Drawings

These harnesses are supplied with the various UGCI kits.
The drawings are provided as a reference to help you connect your controls to the UGCI.

Disclaimer: Although every effort is made to be accurate, Happ Controls cannot assume any responsibility whatsoever for errors or omissions in this or any other document or software relating to the UGCI or the UGCI SDK.