# Table of Contents

**Overview**
- iCade Standard  
- Parts

**Code the Teensy**
- Keyboard Mapping  
- Test Code  
- Controller Code  
- Full Code

**Duo Pinball Teardown**
- Open it Up  
- Identify the Parts  
- Remove Old Board  
- Prep the Wires  
- Wire Common Ground  
- USB Wiring Hole

**Teensy Transplant**
- Be a Pinball Wizard
Overview

Playing virtual pinball on an iPad is a whole lot of fun! Pinball Arcade is my personal favorite -- it runs a huge selection of classic tables reproduced down to the finest detail, and the physics are incredibly accurate and satisfying.

Not so satisfying, however is the lack of tactile feedback when tapping the screen to trigger the flippers or sliding your finger down the screen to pull and release the plunger. So how about using real arcade buttons and a spring loaded plunger instead?

Originally intending to build my own from scratch, I ran across an old accessory called the Duo Pinball controller, which is available online for less than $9 new! (it retailed for $60 when launched in 2012). It was designed for a different pinball game, Pinball HD Collection, and isn't compatible with the one I prefer, but it didn't take too much effort to transplant a new brain in it and make it work!

This guide will show you how to mod your own controller, or use these techniques to build your own from scratch.
iCade Standard

The Duo Pinball controller was locked into a single game, using a Bluetooth board to pair with the iPad and send proprietary commands. In order to free it from these shackles, we'll need to swap out the electronics for something that can use a different iOS game controller standard.

One such standard is the iCade protocol, made by ION Audio for their line of iPad desktop arcade cabinets and controllers. It too requires the games you play to support the standard, but in the case of iCade, there are many, many such games, including my beloved Pinball Arcade!
The iCade standard interfaces with iOS as an HID keyboard, and is typically paired over Bluetooth. However, plugging in a wired HID keyboard works just as well, which makes it very straightforward to build your own controller!

**Parts**

You'll need just a few things to mod or build an iCade-compatible controller for iPad or iPhone:
a microcontroller that can act as a USB HID keyboard, and draws less than 200 mA of current, such as the Teensy 3.2 an adapter for the iOS device to connect USB to the Lightning port, such as the Apple USB to Lightning Camera Adapter. (If you have an older iPad or iPhone you will need the 30-pin dock connector version of the camera kit) micro B to A USB cable arcade buttons and an enclosure (such as a shoebox), or the Duo Pinball controller to hack
Next, let's get the Teensy coded to act like an iCade controller!

## Code the Teensy

Let's look a bit closer at the iCade standard. If you head to the [iCade page on ION Audio's site](https://www.ionaudio.com/support/arcade) you'll see a link to the [spec](https://www.ionaudio.com/support/arcade/spec.pdf). In this .pdf it states:

> The ION Arcade family of products communicates through Bluetooth wireless technology, using a specific key map and protocol to talk with the applications. For each button pressed, the ION Arcade hardware device sends a Key Down character (when button is pressed or joystick is moved directionally) and a Key Up character (when the button is released, or the joystick is moved from its location). The application being made to work with the ION Arcade hardware device will need to be able to interpret these commands, as Key Up and Key Down characters are necessary for successful Application Submissions.

---

Note, they say that it is a Bluetooth connected protocol, but in reality, a hard wired keyboard device works just as well. So, instead of dealing with the complexities, pairing, and power consumption of a Bluetooth device, we can instead program a Teensy to press the keys for us over a USB cable!

The Teensy is going to act as a USB HID keyboard so that it can "type" the keystrokes that the iCade standard uses.

---
Here's what the button mapping looks like for the standard iCade:

Note, this project should work with most Teensy boards, but has only been tested on the 3.2.

Keyboard Mapping

The first test I performed was to figure out the key mappings needed. I did so by pairing a Bluetooth keyboard to the iPad, launching Pinball Arcade, and setting the controller type to iCade in the game's settings. Then, I typed the key pairs seen in the diagram above until I knew which key combos did what.

The mapping for pinball looks like this:
The next test I did was to have the Teensy press the flipper buttons by telling it to "type" the "LVHR" combo over and over. Here's how you can make the Teensy type the keystrokes. First, make sure you've followed these basic guide to setting up the Arduino IDE, and installing and using the Teensy board. Once you've successfully uploaded the basic Blink sketch to your Teensy, move on to the next step.

There are also additional key mappings for a few table tilt directions, so you could probably implement those with physical tilt or vibration sensors!
Load the Simple.ino sketch found in the Arduino IDE under **File > Examples > Teensy > USB_Keyboard > Simple**.

Make sure your Arduino IDE board settings are set up for the Teensy as seen here, including the USB type as Keyboard, plug in the Teensy over USB, launch the Teensy loader application, and then upload the sketch.
Once the program is uploaded, place your cursor in a text field and you'll see that the Teensy will type the words "Hello World" followed by an incrementing number.

Let's adjust this program so that it types the keystrokes for the pinball flippers. Remember, in the iCade standard it is a keystroke to press a button and a second, different keystroke to release the button. "hr" controls the left flipper, and "lr" controls the right flipper.

Despite the chart from the iCade spec showing upper case letters, it actually recognizes lower case instead.

```c
void setup() {
  Serial.begin(9600);
  delay(1000);
}
```
void loop() {
  // Your computer will receive these characters from a USB keyboard.
  Keyboard.print("h");
  delay(100);
  Keyboard.print("l");
  delay(100);
  Keyboard.print("r");
  delay(100);
  Keyboard.print("v");
  
  
}  

OK, time to test it on the iPad or iPhone! Open your notepad app on the iOS device, and then plug the Teensy into it with a USB micro B/A cable and the USB to Lighting "camera adapter". It will start typing those letters furiously!
Unplug the Teensy before you proceed.

Next, let's see it flip those flippers. Install The Pinball Arcade () from the App Store, and then launch it.

In the settings (gear icon on the main screen) tap the iCade button so that it reads iCade: iPad

Now, launch a table, and press START. Before you pull the plunger, plug in the Teensy. After a moment you'll see the flippers start to go! Launch the ball and watch as the random flipper spamming manages to play a fairly decent game of pinball!

Controller Code

Now, let's program the Teensy to use the real button presses to send the keystrokes.

```c
//Pinball Arcade Controller Test
//Uses Teensy 3.2 to act as iCade interface to The Pinball Arcade on iPad
//John Park for Adafruit
/* USB Keyboard button state example
   Teensy becomes a USB keyboard and types characters
   You must select Keyboard from the "Tools &gt; USB Type" menu
*/
#define LED_PIN 13
#define RIGHTFLIPPER_PIN 12

int buttonState = 0; //variable to store current state of button
int lastState = 1; //variable to store state of last button press

void setup() {
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH); //turn on LED so we know it's on
  pinMode(RIGHTFLIPPER_PIN, INPUT_PULLUP);
}

void loop() {
  buttonState = digitalRead(RIGHTFLIPPER_PIN);
  if(buttonState==LOW && buttonState != lastState){ //it's been pressed
    Serial.println("Flipped ");
    Keyboard.print("l");
    delay(100);
    digitalWrite(LED_PIN, LOW); //blink off the LED
    lastState=0;
  }
  else if(buttonState==HIGH && buttonState != lastState){ //it's been released
    Serial.println(" Not flipped");
    Keyboard.print("v");
    delay(100);
    digitalWrite(LED_PIN, HIGH); //turn back on the LED
    lastState=1;
  }
}
```

©Adafruit Industries
Copy and upload this to your Teensy, then use a jumper wire to short pin 12 to ground and release it. Each time you do, the LED will blink and it will send an "l" and a "v". Plug this in to your iPad and try it out as the right flipper button.

Full Code

In researching the iCade standard I came across an excellent project by Allen C. Huffman () that uses a Teensy as an iCade gamepad controller. You can use that code and the pin mappings for this project, too!
I made some small adjustments to the code, and included keyboard mapping comments in line. Copy the code seen here, and upload it to your Teensy. Here are the mappings:

- Plunger on pin 1
- Left flipper on pin 8
- Right flipper on pin 12
- LED on pin 13

```c
/*-------------------------------
Teensy iCade Input
by Allen C. Huffman (alsplace@pobox.com)
   http://subethasoft.com/2013/01/04/teensy-2-0-icade-source-code/

Monitor digital inputs, then emit a USB keyboard character mapped to an iCade button depending on the pin status. The character will be the "hold" character for pin connected (N.O. button push) and "release" character for pin disconnected (N.O. button released).

Pin 13 is used for lighting the onboard LED as a "we are alive" indicator.

This software was written to allow a Teensy 2.0/3.2 to interface between arcade buttons and an iPad via USB and Camera Connector Kit.

2012-12-04 0.0 allenh - Initial version, based on my ArduinoAIDI code.
minor adjustments by john park, July 2017 for Pinball Arcade
*/
#define VERSION "0.0"
#define LED_OFF

#include <eeprom.h>
#include <avr/wdt.h>

/*
iCade keyboard mappings.

Atari joystick port, looking at the male DB9 on the Atari.
See: http://old.pinouts.ru/Inputs/JoystickAtari2600_pinout.shtml

1 2 3 4 5/ Up Dn Lt Rt PA
6 7 8 9/ Bt +5 Gd PB
*/

/*
The following I/O pins will be used as digital inputs for each specific iCade function.
*/
#define UP_PIN 0
#define DOWN_PIN 1 //iCade plunger
#define LEFT_PIN 2
#define RIGHT_PIN 3
#define BTN1_PIN 4
#define BTN2_PIN 5
```

©Adafruit Industries
```c
#define BTN3_PIN  6
#define BTN4_PIN  7
#define BTN5_PIN  8 // iCade left flipper
#define BTN6_PIN  9
#define BTN7_PIN  10
#define BTN8_PIN  12 // iCade right flipper

/*
The following keys are the iCade sequence (hold, release) for each function. Send "W" to indicate UP, and "E" when UP is released.
*/
#define UP_KEYS    "we"
#define DOWN_KEYS  "xz" //iCade plunger original key order "xz"
//#define DOWN_KEYS  "zx" //iCade plunger flipped order for NC switch wiring
#define LEFT_KEYS  "aq"
#define RIGHT_KEYS "dc"
#define BTN1_KEYS  "yt"
#define BTN2_KEYS  "uf"
#define BTN3_KEYS  "im"
#define BTN4_KEYS  "og"
#define BTN5_KEYS  "hr" // iCade left flipper
#define BTN6_KEYS  "jn"
#define BTN7_KEYS  "kp"
#define BTN8_KEYS  "lv" // iCade right flipper

#define DI_PIN_COUNT  12 // 12 pins used.
#define DI_PIN_START  1 // First I/O pin.
#define DI_PIN_END    20 // Last I/O pin.

byte myPins[DI_PIN_COUNT] =
{UP_PIN, DOWN_PIN, LEFT_PIN, RIGHT_PIN,
 BTN1_PIN, BTN2_PIN, BTN3_PIN, BTN4_PIN,
 BTN5_PIN, BTN6_PIN, BTN7_PIN, BTN8_PIN};

char iCadeKeymap[][DI_PIN_COUNT] =
{UP_KEYS, DOWN_KEYS, LEFT_KEYS, RIGHT_KEYS,
 BTN1_KEYS, BTN2_KEYS, BTN3_KEYS, BTN4_KEYS,
 BTN5_KEYS, BTN6_KEYS, BTN7_KEYS, BTN8_KEYS};

char iCadeDesc[][DI_PIN_COUNT] =
{"Up", "Down", "Left", "Right",
 "Btn1", "Btn2", "Btn3", "Btn4",
 "Btn5", "Btn6", "Btn7", "Btn8"};

/* We want a very short debounce delay for an arcade controller. */
#define DI_DEBOUNCE_MS 10 // 100ms (1/10th second)

#define LED_PIN 11
#define POWER_LED 13 //to show power on
#define LEDBLINK_MS 1000

/**************************************************************************/

#define DI_PIN_COUNT 12 // 12 pins used.
#define DI_PIN_START 1 // First I/O pin.
#define DI_PIN_END 20 // Last I/O pin.
```

---

© Adafruit Industries
void setup()
{
    // Just in case it was left on...
    // wdt_disable();

    // Initialize the serial port.
    Serial.begin(9600);

    // Docs say this isn't necessary for Uno.
    // while(!Serial) {} 

    showHeader();

    // Initialize watchdog timer for 2 seconds.
    // wdt_enable(WDTO_4S);

    // LOW POWER MODE!
    // Pins default to INPUT mode. To save power, turn them all to OUTPUT
    // initially, so only those being used will be turn on. See:
    // http://www.pjrc.com/teensy/low_power.html
    for (int thisPin=0; thisPin < DI_PIN_COUNT; thisPin++)
    {
        pinMode(thisPin, OUTPUT);
    }

    // Disable Unused Peripherals
    // ADCSRA = 0;

    // Initialize the pins and digitalPin array.
    for (int thisPin=0; thisPin < DI_PIN_COUNT; thisPin++)
    {
        // Set pin to be digital input using pullup resistor.
        pinMode(myPins[thisPin], INPUT_PULLUP);
        // Set the current initial pin status.
        digitalStatus[thisPin] = HIGH; //digitalRead(thisPin+DI_PIN_START);
        // Clear debounce time.
        digitalDebounceTime[thisPin] = 0;
        //digitalCounter[thisPin] = 0;
    }

    // Set LED pin to output, since it has an LED we can use.
    pinMode(LED_PIN, OUTPUT);
    pinMode(POWER_LED, OUTPUT);

    digitalWrite(POWER_LED, HIGH);
    Serial.println("Ready.");
}

/*--------------------------------------------------------------------------*/

void loop()
{
    // Tell the watchdog timer we are still alive.
    // wdt_reset();

#ifdef LED_OFF
    // LED blinking heartbeat. Yes, we are alive.
    if ( (long)(millis()-ledBlinkTime) &gt;= 0 )
    {
        // Toggle LED.
        if (ledStatus==LOW) // If LED is LOW...
        {
            ledStatus = HIGH; // ...make it HIGH.
        } else {
            ledStatus = LOW; // ...else, make it LOW.
        }
    // Set LED pin status.
#else
    // No LED - not a heartbeat
#endif
}
if (pinsOn==0) digitalWrite(LED_PIN, ledStatus);
// Reset "next time to toggle" time.
ledBlinkTime = millis()+ledBlinkRate;
#endif

// Check for serial data.
if (Serial.available() &gt; 0) {
    // If data ready, read a byte.
    int incomingByte = Serial.read();
    // Parse the byte we read.
    switch(incomingByte)
    {
        case '?':
            showStatus();
            break;
        default:
            break;
    }
}
/*-------------------------------------------------------------------------*/

// Loop through each Digital Input pin.
for (int thisPin=0; thisPin &lt; DI_PIN_COUNT; thisPin++ )
{
    // Read the pin's current status.
    unsigned int status = digitalRead(myPins[thisPin]);

    // In pin status has changed from our last toggle...
    if (status != digitalStatus[thisPin])
    {
        // Remember when it changed, starting debounce mode.
        // If not currently in debounce mode,
        if (digitalDebounceTime[thisPin]==0)
        {
            // Set when we can accept this as valid (debounce is considered
            // done if the time gets to this point with the status still the same).
            digitalDebounceTime[thisPin] = millis()+digitalDebounceRate;
        }
        // Check to see if we are in debounce detect mode.
        if (digitalDebounceTime[thisPin]>0)
        {
            // Yes we are. Have we delayed long enough yet?
            if ( (long)(millis()-digitalDebounceTime[thisPin]) &gt;= 0 )
            {
                // Yes, so consider it switched.
                // If pin is Active LOW,
                if (status==LOW)
                {
                    // Emit BUTTON PRESSED string.
                    Serial.print(iCadeDesc[thisPin]);
                    Serial.print(" pressed (sending ");
                    Serial.print(iCadeKeymap[thisPin][0]);
                    Serial.println(" to iCade.");
                    Keyboard.print(iCadeKeymap[thisPin][0]);
                    //digitalCounter[thisPin]++;
                    pinsOn++;
                    #ifndef LED_OFF
                    digitalWrite(LED_PIN, HIGH);
                    #endif
                } else {
                    // Emit BUTTON RELEASED string.
                    Serial.print(iCadeDesc[thisPin]);
                    Serial.print(" released (sending ");
                    Serial.print(iCadeKeymap[thisPin][1]);
                    Serial.println(" to iCade.");
                    Keyboard.print(iCadeKeymap[thisPin][1]);
                }
            }
        }
    } else {
        // Emit BUTTON RELEASED string.
        Serial.print(iCadeDesc[thisPin]);
        Serial.print(" released (sending ");
        Serial.print(iCadeKeymap[thisPin][1]);
        Serial.println(" to iCade.");
        Keyboard.print(iCadeKeymap[thisPin][1]);
    }
}
if (pinsOn>0) pinsOn--;
if (pinsOn==0) digitalWrite(LED_PIN, LOW);
}
// Remember current (last set) status for this pin.
digitalStatus[thisPin] = status;
// Reset debounce time (disable, not looking any more).
digitalDebounceTime[thisPin] = 0;
} // End of if ( (long)(millis())-digitalDebounceTime[thisPin] ) &gt;= 0 
} // End of if (digitalDebounceTime[thisPin]&gt;0)
else // No change? Flag no change.
{
    // If we were debouncing, we are no longer debouncing.
digitalDebounceTime[thisPin] = 0;
} // End of (int thisPin=0; thisPin &lt; DI_PIN_COUNT; thisPin++)
*/

void showHeader()
{
    int i;
    // Emit some startup stuff to the serial port.
    Serial.print("iCadeTeensy ");
    Serial.print(VERSION);
    Serial.println(" by Allen C. Huffman (alsplace@pobox.com)");
    Serial.print(DI_PIN_COUNT);
    Serial.print(" DI Pins ");
    for (i=0; i&lt;DI_PIN_COUNT; i++)
    {
        Serial.print(myPins[i]);
        Serial.print("=");
        Serial.print(iCadeDesc[i]);
        Serial.print(" ");
    }
    Serial.print(" ");
    Serial.print(digitalDebounceRate);
    Serial.println("ms Debounce.");
}

void showStatus()
{
    showDigitalInputStatus();
}

void showDigitalInputStatus()
{
    Serial.print("DI: ");
    for (int thisPin=0; thisPin &lt; DI_PIN_COUNT; thisPin++)
    {
        // Read the pin's current status.
        Serial.print(iCadeDesc[thisPin]);
        Serial.print("=");
        Serial.print(digitalRead(myPins[thisPin]));
        Serial.print(" ");
        //Serial.print(" ( ");
        //Serial.print(digitalCounter[thisPin]);
        //Serial.print(" ) ");
    }
    Serial.println(" ");
}
If you don't have the Duo Pinball controller you can skip to the last page of the guide and wire up three arcade buttons instead, put them in a cardboard enclosure, and you'll be ready to play!

To modify the Duo Pinball controller and transplant the Teensy into it, move on to the next page.
Duo Pinball Teardown

Let's take apart the Duo Pinball controller and have a look inside.
Open it Up

Using a #1 Philips screwdriver, remove the seven screws that hold together the case. Note, two of the screws are hiding under the rubber feet, so go ahead and peel them off and save them for later.
Now, pry open the case using a spudger or slotted screwdriver. Be careful not to pull to hard on the wire that connects the LED to the board -- you can remove the piece of tape that holds it in place so you can lay both halves side by side.
Now, pry open the case using a spudger or slotted screwdriver. Be careful not to pull too hard on the wire that connects the LED to the board -- you can remove the piece of tape that holds it in place so you can lay both halves side by side.
Identify the Parts
Inside you'll see a few key elements:

- Controller board
- Left flipper button momentary switch
- Right flipper button momentary switch
- Plunger momentary switch and IR sensor
- Plunger/spring assembly
- Battery box
- Indicator LED
Remove Old Board

You can unplug each of the cables from the main board now, and then unscrew it. We won't be using it, so you can set it aside in your old electronic junk pile.

One curiosity is the reset button on the back. Mysterious!
Prep the Wires

To prep the wires, first remove the battery box wires, we won't use them.
Next, snip the connectors off of the remaining wires running from both buttons, the plunger sensor, and the LED indicator.
The plunger sensor board is pretty darned nifty! It's got a momentary snap action switch on it to detect the plunger "closed/opened" state, as well as an IR distance sensor to determine the position of the plunger! We'll only use the switch, since the iCade standard doesn't have a way to express the analog value of plunger position to the game.
You can test the button wiring with the continuity testing mode on your multimeter -- the plunger switch actually has two modes you can use depending on which wires you connect. The white wire, labeled SIN on the board, is a constant, so connect your first lead to it. If you connect the other lead to the red VCC wire the switch is normally closed (NC) as seen here.
Instead, if you connect the second lead to the black GND wire, it is normally open (NO).
You could pick either one and then make an adaptation in software (I chose red originally when I built one on the video live stream and then reversed the key commands in the Arduino sketch to make it work) but let's look at the way the plunger works to decide the best method.

The plunger resting position keeps the switch pushed down. So, when the software checks it, we don't want it to react to the resting state. By choosing the black wire, we can have the software see that switch as open (what we'd colloquially call "unpressed" even though it is physically pushed in). Then, when the plunger is pulled back it will register the switch as closed (what we'd called "pressed" again, even though the button is now no longer pushed in), and send the keystroke to the game.

So, the black wire is the one we'll go with when it's time so solder!
Wire Common Ground

To simplify the wiring runs back to the board later, we can solder all of the black ground wires to each other, and then make a single connection to the Teensy board. This is good if you want to keep things small, since the Teensy 3.2 only has a single GND pin exposed, and this allows us to avoid adding perf board with an extra ground rail.

Cut and strip a short length of black wire to add to the bundle. Twist and solder them all together, or use a wire splice tap to connect them, leaving the free end of the added wire to solder to the Teensy later.

You can add a bit of heat shrink tubing to insulate the connection.
USB Wiring Hole

One other bit of prep to do here is to create a hole in the case for the USB cable. I made mine in the back of the battery box using a 5/8" Forstner bit in a power drill, but you can use a Dremel, saw, hand drill, or any other method you like so long as you can fit the small end of the cable through.
Now it's time to transplant the Teensy!
Your controller is now ready for its new brain! You'll wire and solder these connections:

- Switch common ground black wires to Teensy GND
- Plunger white wire to Teensy pin 1
- Left flipper red wire to Teensy pin 8
- Right flipper red wire to Teensy pin 12
- LED anode red wire (or blue if you prefer the blue glow) to Teensy pin 13
- LED cathode black wire to a 4.7K resistor and then to common GND

The resistor isn't strictly required -- the LED is just very bright without it!
When you solder the resistor in line with the LED, you can add heat shrink tubing over the resistor to insulate it.
After you've soldered the wires, test it out to make sure everything works as planned.

Be a Pinball Wizard

You're just about ready to play! Carefully close up the controller case, being careful not to pinch any of the wires. You can use some hot glue or tape to keep the USB wire in place.

Then, screw in the case screws and reapply the rubber feet. If the original adhesive isn't enough to keep them in place, you can use some double stick tape.
Now, turn on your iPad or iPhone, launch the game, and plug in the controller USB cable to the Lightning connector and into the iPad. The LED will light up, and you're all set. Have fun playing virtual pinball with physical controls!