Table of Contents

Overview 3
• Prerequisite Guides
• Parts
• Tools & Supplies

Circuit Diagram 5
• GEMMA + NeoPixels
• Position and Placement
• Power

3D Printing 6
• PLA Filament
• Slicing Software
• PLA or ABS Material

Assembly 7
• Measure Wire Length
• Strip Wires
• Tin Wires
• Extend wire strip
• Twist Two Wires
• Flatten Twist
• Solder NeoPixels
• Solder Gemma
• Solder NeoPixel 24x Ring
• Check Circuit
• Attach Plate Ring Clips
• Test Position Circuit
• Mount Gemma
• Attach NeoPixels
• Assemble Gear Ring
• NeoPixel 16x Ring
• Mount NeoPixel 16x Ring
• Attach Top and Bottom Circuit
• Tighten Parts
• Final Unibeam

Code 16
• Arduino IDE
• Uploading Code
Overview

Build a chest-mounted cosplay project with a 3D printed UniBeam! In this project, we'll show you how to use 3D printing and electronics to build the infamous repulsion weapon from ironman. This project was built in collaboration with James Bruton from XRobots.

This 3d printed build was made to go inside of James' Hulkbuster project. The Hulkbuster is a massive Iron Man-inspired suit James is building with 3D printed parts and various other materials. Watch James' video and checkout his YouTube channel for more details on his Hulkbuster project.

To make the Unibeam circuit we're using two NeoPixel rings, a couple of single NeoPixel LEDs, and GEMMA, which is an easy to use, byte-sized microcontroller.

Prerequisite Guides

We recommend walking through the follow tutorials below before starting this project. These guides will help you get familiar with the components and get setup and configured with the Arduino IDE and libraries.

- Introducing GEMMA
- Introducing Trinket
- Introducing Pro Trinket
- NeoPixel Uberguide
Parts

We have all the lovely components and tools to build this project. Be sure to check out the featured products on the right sidebar.

- NeoPixel 16x Ring ()
- NeoPixel 24x Ring ()
- NeoPixel Sheet ()
- Gemma ()
- JST Extension ()
- AAA Battery Holder ()
- Toggle Switch ()

Tools & Supplies

You'll need a couple of hand tools and accessories to assist you in the build.

- Solder Iron (http://adafru.it/1204) + Solder (http://adafru.it/734)
- Silicone Wire (http://adafru.it/1877)
- PLA Filament ()
- 3D Printer ()
Circuit Diagram

GEMMA + NeoPixels

Follow the circuit diagram above to get a visual reference of the wiring. The NeoPixels are connected together in a chain, sharing power, ground and data. Two tactile momentary buttons alternate between color modes and activates a firing animation sequence.

Position and Placement

The placement of the components in the diagram are not exact. This is an outline of how the components are wired together. In the final circuit, the 12x NeoPixel ring will be positioned inside of the 24x NeoPixel ring, over the GEMMA.

Power

To power this circuit, use a battery with a JST male connector. The GEMMA has a female JST connection on board. In this project, we're using the AAA battery holder with on/off switch.
3D Printing

PLA Filament

These parts are optimized for printing in PLA material due to the large build area. PLA is more dimensionally stable than ABS when it comes to big prints. The parts print best with no support or raft material.

Edit Unibeam files

Download STL files

Slicing Software

The recommend settings above should work with most slicing software. However, you are encouraged to use your own settings since 3D printers and slicing software will vary from printer to printer.

uniGearClip2.stl
uniRing1.stl
uniRing2.stl
uniPlate.stl

@235 PLA
10% Infill
0.2 Layer Height
2 Shells
90/120 Speeds

about 4 hours to print all parts.
PLA or ABS Material

We recommend using PLA material for an easier print with high quality. The tolerance has been tested with PLA filament but should also work with ABS. The parts do not require any support material or a raft.

Assembly

Measure Wire Length

Make sure to leave extra slack for the connections that are on the outer side of the circle.
Strip Wires

Prepare the wire for tinning by stripping a small amount at first. We'll tin the ends so the strands don't come apart.

Tin Wires

Use solder to tin the tips of the stripped wire.

Extend wire strip

We'll need to fit two wires together into the LED through holes to chain them together. With the tips tinned, strip a bit more of the wire off, this way we'll have enough bare wire to twist together.
Twist Two Wires

Align up two wires and use one to twist along the other.

Flatten Twist

Use a pair of flat pliers to smush the wires together. Resolder the twisted wires to connect the two threads together.
Solder NeoPixels

With the wires twisted, solder the six NeoPixels into a chain.

Solder Gemma

Use a third hand tool to hold Gemma and the NeoPixel 24x ring. Position both so that they have a short distance to each soldering point.
Solder NeoPixel 24x Ring

Tin the NeoPixel 24x ring and then solder connects to Gemma.

Check Circuit

Test power your circuit to make sure no cold solder points are present.
Attach Plate Ring Clips

Align up uniRing1Clip.stl to the rectangular holes on uniPlate.stl. The clips should fit tightly in place.

Test Position Circuit

Test wiring layout with a small amount of mounting putty or tack to position the circuit on to the uniPlate.stl part.
Mount Gemma

Snap Gemma into the grooves. You can use tack putty to secure the circuit into place. Rotate the NeoPixel 24x ring and mount to uniPlate.stl part.

Attach NeoPixels

Mounting putty / tack works great here as well. Apply a small amount to secure each Pixel around uniPlate.stl.
Assemble Gear Ring

Use uniGearClip2.stl to attach uniGear.stl to uniRing.stl

NeoPixel 16x Ring

Slip the NeoPixel 16x ring through uniGear.stl part.
Mount NeoPixel 16x Ring

Snap the NeoPixel 16x Ring to the mount. Make sure to position the wires close to the rest of the circuit.

Attach Top and Bottom Circuit

Snap together uniRing.stl to uniRingClip.stl part. Alternatively, we can use three pieces of 30 gauge wire to secure both halves together.
Tighten Parts

Use a pair of pliers to assist you while you fasten the 30awg wire to the printed parts.

Final Unibeam

Now you have an Iron Man-inspired repulsion weapon! This build can be mounted to your chest, on a wall or even propped up on a stand for your desk.

Code

Arduino IDE

Please be sure you have configured your Arduino IDE to work with the GEMMA microcontroller and that the NeoPixel library are installed and working ala strandtest. Once those are in order, proceede to upload some code!
Uploading Code

Copy the code below. In the Arduino IDE, create a new sketch and paste the code inside the new sketch window. In the top menu, select GEMMA 8Mhz from the Tools > Board menu. Select USBtinyISP from the Tools > Programmer menu. Plug in the GEMMA with a USB cable to your computer, wait for the red LED to blink and hit the upload code button. You should be prompted that code was successfully uploaded.

This sketch was developed by Phillip Burgess ()

```c
/* Hulkbuster Unibeam sketch for Adafruit Gemma &amp; NeoPixels. Requires one 16-pixel NeoPixel ring, one 24-pixel ring, and six single NeoPixels arranged in a large circle. Gemma pin D1 connects to the 24-pixel ring first, then the 6-pixel, then 16...but the physical 'stacking order' is 6 (bottom), then 24, then 16 (top). Add momentary buttons between D1 &amp; GND (mode) and D2 &amp; GND (fire). Unfortunately this ended up resembling an obfuscated C contest entry, and beginners reading it probably won't gain much besides a headache, but I'll comment as best I can. The code is trying to generate some very fluid animation (I don't care for 8-bit-looking twitchy blinky stuff) on a very limited microcontroller (about 5K flash space and no multiply instruction) and thus relies heavily on fixed-point math (using the most-significant digits of large integers to approximate the handling of fractions). Ran out of space toward the end, so parts (like button handling) are a bit slipshod...working but not great. The core idea is that each NeoPixel ring depicts a triangle wave, the phase (rotation) and other parameters can be adjusted on a subpixel basis so it's all buttery smooth. There may be some number of repetitions of the wave (always integer) around the circumference of the ring...for example, as written, it's 3, 4 and 2 reps for the center (16), middle (24) and outer (6 pixel) rings, respectively. If the circles were 'unrolled,' the waves might resemble:
   /\\ /\\ /\\ Each ring then has different colors corresponding to the 'peaks' and 'troughs' of these waves. Those colors change over time...but never just an abrupt reassignment, there's always a smooth transition from one set of colors to the next. Other parameters, like rotation phase and speed, are also interpolated smoothly through those transitions. Oh, also, there's timer interrupts, another really complicated subject. Sorry about that. */

#include &lt;Adafruit_NeoPixel.h&gt;
#include &lt;avr/power.h&gt;

#ifdef _AVR_ATtiny85_ // Trinket/Gemma:
#define LEDPIN 1 // NeoPixels connect here
#define MODEPIN 0 // Switch modes w/button between this pin &amp;amp; GND
#define FIREPIN 2 // 'Fire' w/button between this pin &amp;amp; GND
#else
#define LEDPIN 6 // NeoPixels connect here
#define MODEPIN 4 // Mode button between this &amp;amp; GND
#define FIREPIN 5 // Fire button between this &amp;amp; GND
#endif
```
// Partial color assignments for various ring states; HSV colorspace,
// not RGB. Hue is always determined by the current mode, so these
// just include saturation and value (brightness).
#define IDLE_PEAK_SAT       255 // "Dim" rings
#define IDLE_PEAK_VALUE     130
#define IDLE_TROUGH_SAT     255
#define IDLE_TROUGH_VALUE    45
#define BRIGHT_PEAK_SAT      80 // Current "bright" ring
#define BRIGHT_PEAK_VALUE   230
#define BRIGHT_TROUGH_SAT   255
#define BRIGHT_TROUGH_VALUE 135
#define AIM_SAT               0 // Ramping up to fire...
#define AIM_PEAK_VALUE      140
#define AIM_TROUGH_VALUE    120
#define FIRE_SAT              0 // Firing
#define FIRE_VALUE          255
#define DEBOUNCE             10 // Counter limit for mode, fire buttons

// Okay, this is just terrible, don't try this at home. I'd been curious
// if one could use the preprocessor, not runtime code, to sum values in
// a struct array. Here, counting the number of pixels in a list of
// NeoPixel rings in order to declare the single Adafruit NeoPixel strip
// length. List can then change without editing the declaration every
// time; it follows automatically. Short answer: yes, it can be done.
// Long answer: this is a hack and not well-behaved C syntax for anything,
// please don't use as a model for your own code, was just an experiment.

// List rings here -- index of first LED, number of LEDs, number of wave
// repetitions. This is the order in which they're referenced in the
// ring[] array later, and doesn't need to match the order in which they're
// physically connected. They should not overlap, nor should there be any
// gaps. No code is generated here...all just preprocessor shenanigans.
#define RING_TABLE
  ring(30, 16, 3)  
  ring( 0, 24, 4)  
  ring(24,  6, 2)  
  // Don't remove this comment
  // Preprocessor trick counts the total number of LEDs in the list.
  // Adapted from supercat's code at stackoverflow.com/questions/3539549
#define ring(f,n,r) EN_##n}; enum {EN_SZ_##n=n,EN_TMP_##n=EN_##n+(n-1),
  enum {EN_IDX=-1, RING_TABLE N_LEDS};
#undef  ring // After enumeration, same macro now inits array defaults:
#define ring(first,n,r) { first, n, r, {128,128,128}, 0, {0,0,0}, {{{0,0,0}, {0,0,0}}, {{0,0,0}, {0,0,0}}},
  // All so we can do this:
Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_LEDS, LEDPIN);
  // instead of ring[0].length + ring[1].length + ... + ring[2].length

// Each ring animates based on one or more full cycles of a triangle wave,
// which may be skewed toward a sawtooth wave. RGB colors are interpolated
// between the peaks and troughs of the wave.
struct Ring {
  const uint8_t     firstLED; // Index of first pixel along full strand
  const uint8_t     nLEDs;    // Number of LEDs in ring
  uint8_t           reps;     // Number of wave cycles around ring
  uint8_t           peak[3];  // [to/from/current] sawtooth peak offset (0-255)
  volatile uint16_t phase;    // Offset angle (65536 = 360 degrees)
  volatile int16_t  spd[3];   // [to/from/current] phase increment per frame
  volatile uint8_t  color[3][2][3]; // [to/from/current][peak/trough][R,G,B]
} ring[] = {
  // RING_TABLE // Macro above expands this into data for all rings
};
#define N_RINGS (sizeof(ring) / sizeof(ring[0]))

// Each of three animation modes has a distinct color and a rotation
// speed/direction for each of the rings.

---
Clockwise direction is different on 16 pixel ring vs 24 (and the arrangement of 6 discrete NeoPixels). On 16 pixel ring: + is clockwise, - is counterclockwise. On 24 pixel ring (and our 6 pixel ring), - is clockwise, + is counterclockwise.

```c
struct Mode {
    const int16_t hue;
    const int16_t speed[3];
};
modeData[] = {
    1100, {  2500, -2000,  1500}, // Blue mode (CW, CW, CCW)
    510, {  2500,  2000, -1500}, // Green mode (CW, CCW, CW)
    15, { -2500, -2000, -1500}, // Red mode (CCW, CW, CW)
};
#define N_MODES (sizeof(modeData) / sizeof(modeData[0]))

uint8_t mode = 0,
      brightRing = N_RINGS - 1; // Counts through rings for pulsing effect
volatile uint8_t interpFrames = 0, // Duration of color/speed transition
                 interpolating = 0; // Current transition counter, 0=done

void setup() {
    #if defined(__AVR_ATtiny85__) && (F_CPU == 16000000L) // 16 MHz Trinket?
        clock_prescale_set(clock_div_1);
    #endif
    strip.begin();
    strip.show(); // Initialize all pixels to 'off' ASAP
    pinMode(MODEPIN, INPUT_PULLUP);
    pinMode(FIREPIN, INPUT_PULLUP);

    // Configure "bootup" transition
    setRingsToIdleValues();
    startInterp(105); // ~3.5 second startup

    // Configure Timer/Counter 1 for 30-ish Hz interrupt
    #ifdef __AVR_ATtiny85__ // Trinket/Gemma:
        #if (F_CPU == 16000000) // 16 MHz:
            TCCR1 = _BV(CS13) | _BV(CS12); // 1:2048 prescale
        #else // 8 MHz:
            TCCR1 = _BV(CS13) | _BV(CS11) | _BV(CS10); // 1:1024 prescale
        #endif
        GTCCR = 0; // No PWM out
        TIMSK = _BV(TOIE1); // Overflow interrupt
    #else // Everything else:
        TCCR1A = _BV(WGM11) | _BV(WGM10); // Mode 15, no PWM out
        TCCR1B = _BV(WGM13) | _BV(WGM12) | _BV(CS11) | _BV(CS10); // 1:64 prescale
        OCR1A = F_CPU / 64 / 30; // ~30 Hz cycle
        TIMSK1 = _BV(TOIE1); // Overflow interrupt
    #endif

    // Once the interrupt is enabled, animation (and transitions between animation states) all occur automatically. We just monitor the 'interpolating' variable to indicate when the current transition is done before issuing another.

    while(interpolating); // Wait for bootup transition to complete
}
```

// Don't return until both buttons are released and debounced
void waitForButtonRelease() {
    for(uint8_t t = 0; t < N_MODES; t++) {
        if((digitalRead(MODEPIN) == LOW) || (digitalRead(FIREPIN) == LOW)) i=0;
```
else if(++i &gt;= DEBOUNCE) return;
}

// Set all rings to 'dim' colors for current mode
void setRingsToIdleValues() {
    for(uint8_t r=0; r&lt;N_RINGS; r++) {
        setTargetColor(r, modeData[mode].hue, IDLE_PEAK_SAT, IDLE_PEAK_VALUE, IDLE_TROUGH_SAT, IDLE_TROUGH_VALUE);
        setTargetSpeed(r, modeData[mode].speed[r]);
    }
}

void nextMode() {
    setRingsToIdleValues(); // Bright ring will fade off
    for(startInterp(10); interpolating; ); // ~1/3 sec transition
    if(++mode &gt;= N_MODES) mode = 0; // Advance/wrap mode counter
    setRingsToIdleValues(); // Set new colors/speeds
    brightRing = N_RINGS - 1; // Start w/center after transition
    for(startInterp(45); interpolating; ); // ~1.5 sec. transition
    waitForButtonRelease();
}

void fire() {
    uint8_t r;
    setRingsToIdleValues(); // Bright ring will fade off
    for(startInterp(10); interpolating; ); // ~1/3 sec transition
    for(r=0; r&lt;N_RINGS; r++) { // All rings fade to 'aim' state...
        setTargetColor(r, modeData[mode].hue, AIM_SAT, AIM_PEAK_VALUE, AIM_SAT, AIM_TROUGH_VALUE);
        setTargetSpeed(r, modeData[mode].speed[r] * 10);
    }
    for(startInterp(105); interpolating; ); // 3.5 sec warmup transition
    for(startInterp(40); interpolating; ); // 1.3 sec hold
    for(r=0; r&lt;N_RINGS; r++) { // All rings fade to 'fire' state...
        setTargetColor(r, modeData[mode].hue, FIRE_SAT, FIRE_VALUE, FIRE_SAT, FIRE_VALUE);
    }
    for(startInterp(7); interpolating; ); // ~1/4 sec ramp up..."boom"
    setRingsToIdleValues(); // Fade back to prior colors
    for(startInterp(120); interpolating; ); // ~4 sec cool-down
    brightRing = N_RINGS - 1; // Resume @ center after firing
    waitForButtonRelease();
}

void loop() {
    uint8_t pinState, priorPinState, debounceCounter, nextAction = 0;

    // For next transition...first, set bright ring back to idle color...
    setTargetColor(brightRing, modeData[mode].hue, IDLE_PEAK_SAT, IDLE_PEAK_VALUE, IDLE_TROUGH_SAT, IDLE_TROUGH_VALUE);
    if(++brightRing &gt;= N_RINGS) brightRing = 0; // Next ring
    // ...then set new ring to bright state...
    setTargetColor(brightRing, modeData[mode].hue, BRIGHT_PEAK_SAT, BRIGHT_PEAK_VALUE, BRIGHT_TROUGH_SAT, BRIGHT_TROUGH_VALUE);
    startInterp(18); // Start pulse transition, about 2/3 sec
    // While the transition takes place, we can do other things, like
    // poll the buttons for mode change or fire events...
    priorPinState = digitalRead(MODEPIN) + (digitalRead(FIREPIN) &lt;&lt; 1);
    debounceCounter = 0;
    while(interpolating) { // Still transitioning...
        if(nextAction) continue; // Once action is decided, stop polling buttons
        pinState = digitalRead(MODEPIN) + (digitalRead(FIREPIN) &lt;&lt; 1);
        if(pinState == priorPinState) {
            if(++debounceCounter &gt;= DEBOUNCE) nextAction = pinState;
        } else {
            debounceCounter = 0;
            priorPinState = pinState;
        }
    }
}
// Downside is that mode/fire actions don't take place until the prior
// transition ends (worst case, about 2/3 sec.), so this sometimes
// requires a bit of button mashing. I'd been working on making the
// pulse transitions interruptible but ran out of code space. Ah well,
// it's for a fun cosplay thing, not a cure for rocket surgery.

if(nextAction == 1)      nextMode();
else if(nextAction == 2) fire();

// Set ring colors for the next transition
void setTargetColor(uint8_t r, int16_t h, uint8_t sPeak, uint8_t vPeak, uint8_t sTrough, uint8_t vTrough) {
    uint32_t cPeak = hsv2rgb(h, sPeak, vPeak); // Color @ peak
    uint32_t cTrough = hsv2rgb(h, sTrough, vTrough); // Color @ trough
    ring[r].color[1][0][0] = cPeak >> 16;
    ring[r].color[1][0][1] = cPeak >> 8;
    ring[r].color[1][0][2] = cPeak;
    ring[r].color[1][1][0] = cTrough >> 16;
    ring[r].color[1][1][1] = cTrough >> 8;
    ring[r].color[1][1][2] = cTrough;
}

// Set ring speed for the next transition
void setTargetSpeed(uint8_t r, int16_t s) {
    ring[r].spd[1] = s;
    ring[r].peak[1] = (s > 0) ? 48 : 208;
}

// Begin transition
void startInterp(uint8_t n) {
    interpFrames = n;
    interpolating = n;
}

ISR(TIMER1_OVF_vect) { // Timer/Counter 1 overflow, configured for ~30 Hz
    uint8_t i, r, y, l, x8;
    uint16_t ts, x, xinc, n;
    // Refresh the strip with results calculated during the -prior- frame.
// Ensures a uniform frame rate; calc time for each frame may vary.
strip.show();

for(r=0; r<N_RINGS; r++) { // For each ring...
    l = ring[r].firstLED; // l = current LED index
    x = ring[r].phase; // x = ring rotation (65536 = 360 degrees)
    xinc = (uint16_t)(65536L * ring[r].reps / ring[r].nLEDs);
    for(i=0; i&lt;ring[r].nLEDs; i++, x += xinc) { // Each LED on current ring...
        x8 = x &gt;&gt; 8;
        y = (x8 &amp;&lt; ring[r].peak[2] ) ? // Which side of triangle wave?
            ( (uint16_t)x8 * 255) / ring[r].peak[2] : // Rising edge
            ( (256 - (uint16_t)x8) * 255) / (256 - ring[r].peak[2]); // Falling edge

        // y is blending factor (0-255) between peak &amp; trough colors
        ps = y + 1; // Peak color scale factor (1-255)
        ts = 257 - ps; // Trough color scale factor (1-255, inverse of peak)

        // Blend peak &amp; trough RGB, process through gamma correction table
        strip.setPixelColor(l++, // and store resulting color
            pgm_read_byte(&amp;gamma8[(ring[r].color[2][0][0] * ps +
            ring[r].color[2][1][0] * ts) &gt;&gt; 8]),
            pgm_read_byte(&amp;gamma8[(ring[r].color[2][0][1] * ps +
            ring[r].color[2][1][1] * ts) &gt;&gt; 8]),
            pgm_read_byte(&amp;gamma8[(ring[r].color[2][0][2] * ps +
            ring[r].color[2][1][2] * ts) &gt;&gt; 8]));
    }
    ring[r].phase += ring[r].spd[2]; // Rotate ring for next frame
}

if(interpolating) { // Frame-to-frame speed &amp; color interpolation
    if(!--interpolating) { // Interpolation target reached?
        // Set current &amp; prior color &amp; speed to target values
        for(r=0; r&lt;N_RINGS; r++) { // For each ring...
            for(x=0; x&lt;2; x++) { // Wave peak, trough
                for(i=0; i&lt;3; i++) { // R,G,B
                    ring[r].color[2][x][i] = ring[r].color[0][x][i] +
                        (((ring[r].color[1][x][i] - ring[r].color[0][x][i]) * n) &gt;&gt; 8);
                }
            }
            ring[r].spd[2] = ring[r].spd[0] = ring[r].spd[1];
            ring[r].peak[2] = ring[r].peak[0] = ring[r].peak[1];
        }
    } else { // Interpolation target not reached yet
        n = 257 - (((uint16_t)interpolating &amp;&lt; 8) / interpFrames);
        for(r=0; r&lt;N_RINGS; r++) { // For each ring...
            ring[r].spd[2] = ring[r].spd[0] +
                (((uint32_t)(ring[r].spd[1] - ring[r].spd[0]) * n) &gt;&gt; 8);
            ring[r].peak[2] = ring[r].peak[0] +
                (((uint32_t)(ring[r].peak[1] - ring[r].peak[0]) * n) &gt;&gt; 8);
            for(x=0; x&lt;2; x++) { // Wave peak, trough
                for(i=0; i&lt;3; i++) { // R,G,B
                    ring[r].color[2][x][i] = ring[r].color[0][x][i] +
                        (((ring[r].color[1][x][i] - ring[r].color[0][x][i]) * n) &gt;&gt; 8);
                }
            }
        }
    }
}

const uint8_t PROGMEM
gamma8[] = { // x^2.8 improves appearance of midrange colors
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5,
    6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 10,
    10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 14, 15, 15, 16, 16,
    17, 17, 18, 18, 19, 19, 20, 20, 20, 21, 21, 22, 22, 22, 23, 23, 24, 24, 25,
}