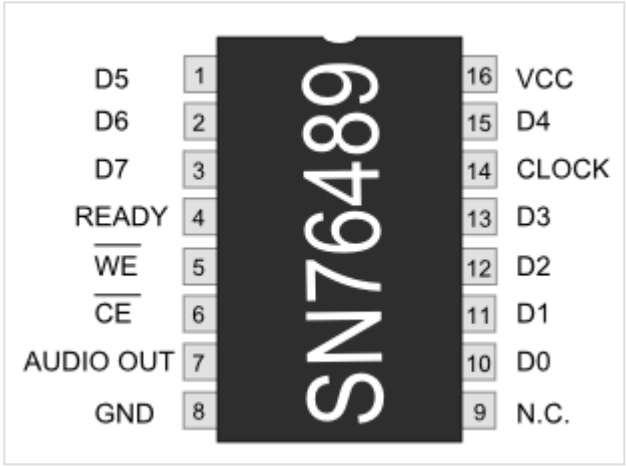
The Oddbloke Geek Blog

I love the smell of solder in the morning

Experimenting with an Arduino and a SN76489

Posted on **2014/01/25**

The SN76489 is a sound generator IC. It has quite a distinctive sound, and is almost iconic in its association with some 8-bit computers. It's what gives the BBC Micro its distinctive boop-bip (which is how I know of it) and is also used in the Sega Master System, Sega Game Gear, and a fair few old arcade machines (such as Mr Do and Wonder Boy).



It produces four-channels of mono sound – "four-channel" meaning that it can play four different notes independently. Actually, it's three channels of notes, and one of "noise" – which is typically used for percussion, gunfire, explosions, and so on.

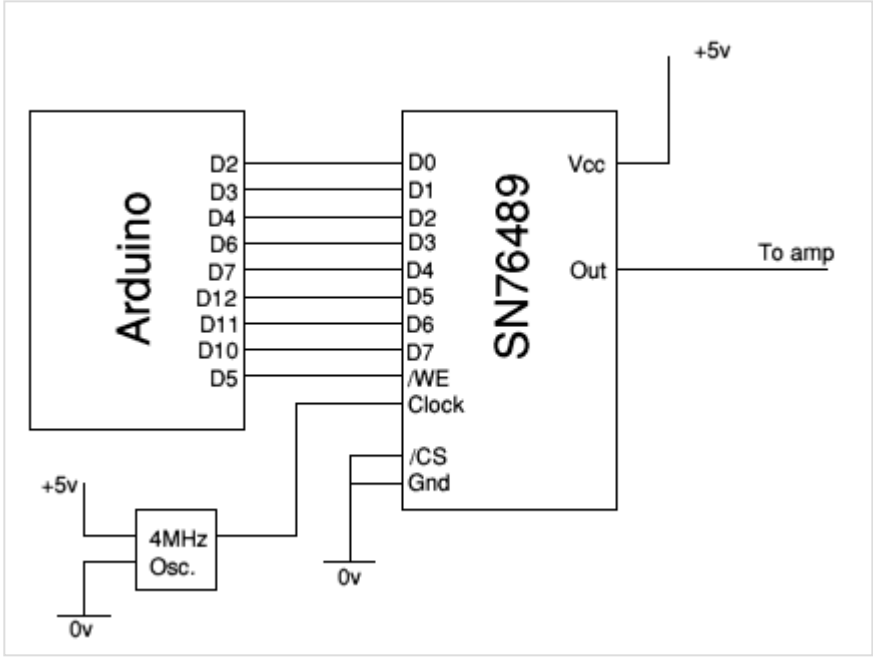
They're still fairly easy to get hold of – eBay, for example. Or you could pull one out of a dead console. I just beg you not to pull one out of a working BBC Micro, though: that would just be **wrong**.

Controlling the SN76489

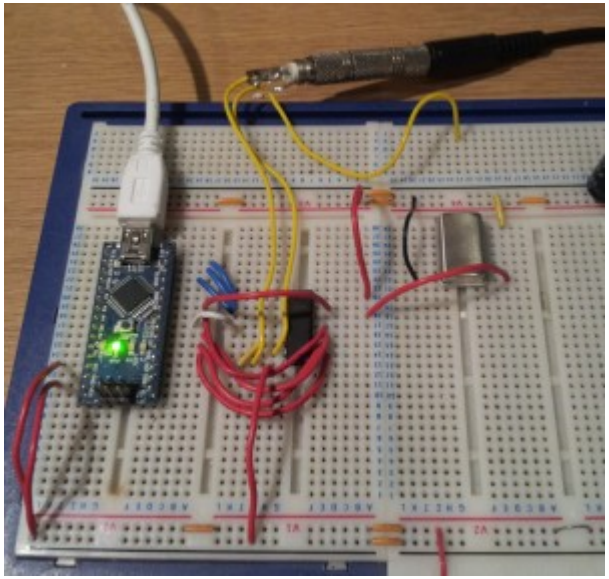
To control it, you send it commands. Commands are simple eight-bit bytes. Sending those commands is as simple as putting the byte on the eight data lines, then strobing /WE low to store. It has a chip-select pin (also active low), so you can run it on a shared-bus

with other devices if needed. For the purposes of this demonstration it'll be active all the time, so just tie /CS to 0v. Depending on which datasheet you look at, the /CS (chip select) pin may also be known as /OE (output enable) or /CE (chip enable).

Driving the IC from an Arduino is fairly straightforward – you'll need eight digital outputs for the byte, and a ninth for /WE. If you're not an Arduino person then just substitute your processor of choice.



To be honest, the wiring is so simple I nearly didn't bother drawing it. Note that the pin assignments on the Arduino-end are completely arbitrary – they're just what was convenient when I was building the circuit on breadboard. They match with the code example (see later). Feel free to change them if it makes sense for your circuit.



— My circuit, on a breadboard. I'm using an Arduino Nano, which is just like the Uno but cuter and cheaper. The black cable you see at the top is where I'm feeding the sound to line-in on a mini hifi.

There is nothing much to read from the IC – you can send it commands, but you can't communicate with it to find out any current values (for example). There's no two-way conversation needed, and this makes it fairly simple to work with. There is a "ready" pin (which goes LOW whilst the IC is processing your last command) but it's such a quick affair it probably isn't worth it. Even the BBC Micro doesn't bother.

Here's a snippet of Arduino code that demonstrates how you might send it a byte:

```
void SendByte(byte b)
{
  digitalWrite(PIN_D0, (b&1)?HIGH:LOW);
  digitalWrite(PIN_D1, (b&2)?HIGH:LOW);
  digitalWrite(PIN_D2, (b&4)?HIGH:LOW);
  digitalWrite(PIN_D3, (b&8)?HIGH:LOW);
  digitalWrite(PIN_D4, (b&16)?HIGH:LOW);
```

```
  digitalWrite(PIN_D5, (b&32)?HIGH:LOW);
  digitalWrite(PIN_D6, (b&64)?HIGH:LOW);
  digitalWrite(PIN_D7, (b&128)?HIGH:LOW);
  digitalWrite(PIN_NotWE, HIGH);
  digitalWrite(PIN_NotWE, LOW);
  digitalWrite(PIN_NotWE, HIGH);
}
```

Yes, I know it's long-winded compared to just bashing the ports directly. I intended this posting to be simple and educational – feel free to optimise away when you have your circuit running.

Obviously you must have defined the PIN_ constants yourself beforehand, and set them up as outputs.

Sound output

Sound comes out of pin 7. The easiest way to hear it might be to connect it to the line-in input on a mini-hifi, or computer speakers, or a headphone amp. If you're planning on building a doorbell then you'll need a mini speaker circuit – an easy way would be to look for a circuit based on the LM386 power-amp, which is a little 8-pin IC that only requires a 5-volt supply to function. If you're feeling especially lazy, you can buy little circuit board amps from eBay that even include screw-terminals to wire your speaker, for only a few quid.

Frequency input

The chip works by modulating a square-wave signal you provide – the frequency isn't critical, but be aware that if the frequency you generate isn't exactly the same as whatever hardware you're used to, then the tones will be off. The BBC Micro uses a frequency of exactly 4 MHz, and I found it no trouble to get hold of 4 MHz oscillator components (eBay again).

Powering on

When the IC is first powered-on, one of the channels will begin to produce a low-tone straight away. (If you have ever used a BBC Micro, you might be interested to know that this is the "boop" of the "boop-bip".) So when you first power it up and you hear a potentially alarming sound then **don't panic** – that sound means it is working! It is up to the host processor to send the sound-off commands.

What to play?

There are already some excellent websites around (including SMS Power's comprehensive page, the SN76489 datasheet and this page) that describe the 8-bit commands, so I shall not repeat them here. But as simple examples:

```
SendByte(0x83); SendByte(0x12); // sets channel 0 tone to 0x123
SendByte(0x90); // sets channel 0 to loudest possible
SendByte(0x9F); // sets channel 0 to lowest volume (silencing it)
```

Note that I can't tell you what note would be played by 0x123 in the above example, because this depends on the frequency of clock you use as input.

Playing something more impressive

If manually entering SendByte() commands and pauses to play "When the saints go marching in" is your thing then great – you have enough to be getting on with and we'll see you next year. But for those of us with short attention spans and no musical knowledge – where do we get our data?

Manipulating MIDI on the Arduino is fairly well documented, and there are some interesting projects that do this – great for the musician who wants their thousand-quid electric piano to sound like a thirty-year old 8-bit computer! Have a look at this page or this page, for example.

Or you could buy a rubbish children's toy keyboard from a carboot sale, and get your Arduino to scan the keyboard and drive the SN76489 that way.

However, I've gone a different route. My initial reason for controlling a SN76489 was to build a sort of retro-doorbell for our new house – I had this idea that the doorbell could be the death jingle to Chuckie Egg or the intro music to Repton 3.

So: I added a small amount of extra code to BeebEm (a very mature BBC Micro emulator) running on my Mac, so that as bytes were sent to the emulated SN76489 they would also be logged to a textfile (along with the time they were sent). I then wrote a quick command-line utility to process these numbers and spit them out as C-sourcecode (along with some more meaningful comments), wrote a quick Arduino sketch that sends these bytes with the same timings to a *real* SN76489 and … hey presto … an authentic retro doorbell which will impress your friends and infuriate your wife!

Here is my Arduino sketch that plays the Chuckie Egg death music repeatedly, by parsing the bytes to be sent as an array of values included in the code:

SN76489 example sketch for Arduino

It would be straightforward to nominate a digital input for the doorbell pushbutton, then house the Arduino and SN76489 with a small amplifier circuit driving a speaker.

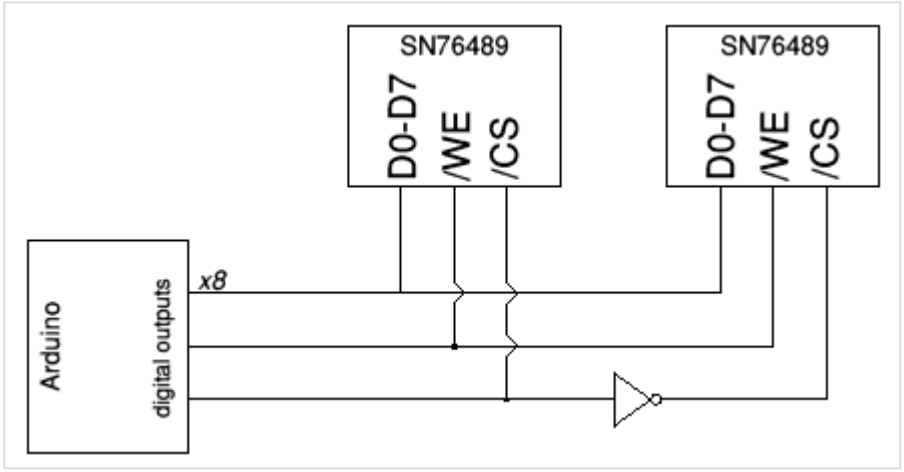
Playing in stereo?

Just as an afterthought: the SN76489 IC produces mono sound – what if you want stereo?

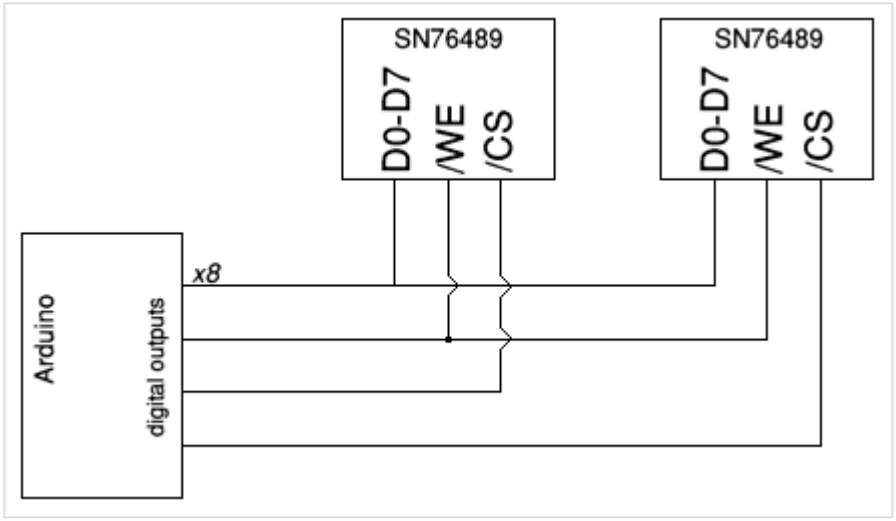
To play a note that appears to come from over to one side rather than bang in the middle of your ears, the note must be played at different amplitudes in each ear. So: play it loud in your left ear and quietly in your right will make your brain think the sound is coming from over to the left.

To do this you would add a second IC – then designate one for the left ear, and one for the right. Connect the eight datalines and the /WE line from the Arduino to *both* ICs, then utilise the /CS lines so that when you send data you have first selected which IC should receive it.

You can do this with one extra digital-out from the Arduino, and a NOT gate so that one chip is selected when the line is high, and the other when the line is low.



Alternatively, if you want to keep your component-count down and you can afford to use a second digital-out from your Arduino, then just control the /CS lines separately.



But of course, to utilise this circuit, you'd need to find some stereo music data.

This entry was posted in **Arduino**, **Digital**, **SN76489**, **Sound** by **admin**. Bookmark the **permalink [http://danceswithferrets.org/geekblog/?p=93]** .

14 THOUGHTS ON "EXPERIMENTING WITH AN ARDUINO AND A SN76489"

jonah
on **2014/02/06 at 21:52** said:

haha, i was looking into a BBC micro + BBC Basic to mess with the sound chip when i read that raspberry pi runs BBC basic. so i searched for "sn76489 raspberry pi" and you're the top result, coincidently talking about bbc basic! 🙂

i guess for a doorbell arduino makes more sense. 😉 have you experimented with using the raspberry pi to control the sn76489 though?

i was thinking about using the sn76489 with raspberry pi and programing it with BBC basic but i wasn't sure if it supported working with 1 or even better multiple chips.

if BBC basic runs i wonder if any of the other dedicated music programs run too….

admin
on **2014/02/09 at 18:59** said:

"BBC Basic running on a raspberry Pi" just means that someone has written an interpreter that will accept and parse a BASIC program and show the same result as a BBC. But the interpreter itself won't be able to drive a SN76489. If the BASIC interpreter happened to support controlling external hardware then you could presumably build your own circuit and control it with BASIC, though.

So you'd need to investigate how to control nine digital outputs from the Pi (I imagine a couple of serial-to-parallel registers would do it) and then write a PROCsendbyte procedure.

So in my diagrams, mentally remove the Arduino and replace it with a Pi and two shift registers (something like the PCF8574 or the 74xx595) and keep the rest of the circuit the same.

Alternatively … the Pi is a powerful enough machine to run a BBC Micro emulator. So you could find a version of BeebEm compiled for the Pi, and use that.

Jonny
on **2014/04/03 at 11:31** said:

Heya! Great little bit of info. I've been researching Electron/BBC micro sounds a bit myself recently for something. I didn't consider just "grabbing the chip" though haha. I'm still very green around electronics so tend to be not thinking at the chip level yet. Anyhow just wanted to say thanks and to point out the chuckie egg code is missing so if you could put it back that would be ace. Thanks again Jonny

admin
on **2014/04/03 at 14:00** said:

Interesting … wonder where that's gone? No worries, I'll sort it ASAP.

Samuel
on **2015/02/06 at 08:38** said:

Hey admin,

I successfully got this working using a serial-in, parallel out shift register (74HC164) connected to my SN76489 with a slight modification to your example code, drastically reducing the number of connections I have to make between my breadboard and my Arduino from 9, down to 3, not counting the two power pins.

Anyhow, are you familiar with the .vgm format? I am looking to convert .vgm files that use the SN76496 (I believe the only difference between that and the SN76489 is the noise generator pattern) into the code your example uses. The best I have gotten so far is to convert said .vgm files into human-readable text and type it all by hand (I am a very patient person), but the timings are different (being terrible with math doesn't help), and the data looks different too (for example, tone data only shows the second byte, although the comment states which channel that second byte is going to. Do you have any advice that might help me in my endeavour to play my favourite .vgm files on a real SN76489?

Thanks,

– Samuel

admin
on **2015/02/06 at 08:56** said:

Hi Sam,

I'm delighted the article was useful to you! I'm afraid I'm not familiar with the VGM format – but now I know of its existence, I think I portable-VGM-player project would be rather cool! If you could read VGM files off an SD card and use a small Hitachi screen to display the track name, I think that would be really cute!

That format's ability to represent data for a wide range of chip types might make things problematic. I also note that it's possible to store the clock frequency – and as my circuit is hardwired to 4MHz, that means there may potentially be a lot of pitch issues.

If you've written code which "explains" the file data in a human-readable format, then that means you have a good understanding of the file format. But I don't think I'd try to convert that data into a new format. I think I'd probably use something like the bin2c utility to turn that data into something that can be copied-and-pasted into the Sketch and therefore compiled-in to your project. Then try and work with the file in its native format. If you could get that to work, then the next step (reading it from an SD card instead) would become more straightforward.

Right, I need to see if anyone has made VGM files from BBC Micro games …

Jacob
on **2015/12/29 at 22:14** said:

I'm having a consistent problem where the SN76489AN seems to be out of tune. Some notes play in tune, and some are several notes off. I read the datasheet and I am doing the equation to figure out the correct 10 bit binary number to correspond to the frequencies I want. When I try to play a simple chord, some notes are completely off, some are in the next octave down, and occasionally one is right where it should be. Does anyone know what might cause this? I can post my code and setup if needs be.

admin
on **2016/01/05 at 10:17** said:

This will sound dumb but … are you SURE your wiring is right for the eight data lines? If you had some swapped around, then it might explain why some notes are correct and others not.

Another possibility might be to rig up a second Arduino to sit in place of the SN chip, and “receive” the commands and print them out to the serial port. That way you can check that the numbers you think you’re sending are actually what the SN chip receives.

If you haven’t already, try to check the quality of your oscillator circuit too. Just monitoring it with a good frequency counter (or a DSO with frequency-counting ability) would be enough.

Code and circuit would be interesting to see!

Nigel
on **2016/07/22 at 15:40** said:

Hi

I bought some of the SN76489A variant to give me sound output from a retro computer that I’m building.

I made the test setup connected to an Arduino as above but got no sound.

So, I got another chip out and took it down to the basics, with just a 4MHz clock and /CS low.

You said that the chip should make an initial tone (as in the first of the BBC Micro power on bleeps) until told to be quiet.

However, the scope shows a sine wave at around 430 kHz on the output. At that frequency, I’m not expecting to hear anything.

When I was running the Arduino code, the frequency was about the same.

I disconnected the oscillator and fed in a square wave, going all the way down to 100kHz. However, the output frequency remained pretty much the same.

Any ideas please ?

Thanks

admin
on **2016/07/24 at 21:17** said:

No idea, I'm afraid! I know there's clock requirement differences between the variants – one divides by 32 and one does not, for example. Try getting hold of an AN version and see if the behaviour improves?

Alternatively … if the input clock does not affect the output, then perhaps it's something as primitive as a noisy power supply. Try a 100nF cap for decoupling, and maybe an electrolytic for smoothing.

What are you building your retro computer around, then? I'm doing something similar, with a 6502.

Nigel
on **2016/07/31 at 07:45** said:

Hi

I'm using a 6502 with a Parallax Propeller for the I/O.

It runs a VGA monitor with a PS/2 keyboard and a micro SD card.

I've been emulating aspects of the BBC Micro OS and although things still need doing, it runs BBC BASIC from the same machine.

Logan
on **2016/09/23 at 01:41** said:

Hi! I was wondering if I was doing something wrong with the code. When I try to compile, it complains about not having prog_uint8_t in its data. Am I missing something?

admin
on **2016/09/23 at 08:38** said:

According to Google, that datatype has been deprecated in later versions of the Arduino IDE. If you put this line above the offending one, it should fix it:

typedef uint8_t PROGMEM prog_uint8_t;

See here: https://github.com/sparkfun/SevSeg/issues/10

Pingback: How Streets of Rage 2 used the Mega Drive Hardware to Create an Iconic Soundtrack – Sarah Crafton's Portfolio