The AY Teletalk

The AY Teletalk box is basically a classic game synthesizer chip, the AY-3-8912, controlled by an Arduino Uno microprocessor.

AY Synth History

The AY-3-8910 is first of a series of 3-voice programmable sound generator chips designed by General Instrument at the end of the 1970's. The series was very successful, becoming the most-used audio chip of the arcade industry through the early 1980s. They were used on, among others, the Intellivision and Vectrix video game consoles, Amstrad CPC, Oric 1, Colour Genie, Elektor TV Games Computer and Sinclair ZX Spectrum computers as well as the Mockingboard and Cricket sound cards for the Apple II family. By 1987 it was modified, under license, by Yamaha as the YM2149F; the Atari ST used this version. Texas Instruments followed with its own version, the SN76489. The chips are no longer made, but a declining stock for servicing vintage machines is still available as well as on eBay.

AY Synth Basics

The AY chip features 3 square wave tone generators output to three separate chip pins. Each tone generator may be mixed with a single pseudo random pulse width modulated noise generator, and the amplitude of each mix can be controlled. A single envelope generator can be applied to any combination of the tone generators.

AY Synth Programmable Registers

The synthesizer parameters are set by the contents of 14 registers described below. The job of the Arduino microcomputer is to load these registers with values and at times designated by the programmer.

Registers 0-5: Each tone generator has an 8-bit Fine Tune register and a 4-bit Coarse Tune register to set the pitch of its square wave.

Fine Tune A (0-255)
Course Tune A (0-15)
Fine Tune B (0-255)
Course Tune B (0-15)
Fine Tune C (0-255)
Course Tune C (0-15)

Register 6: The single noise generator has a 5-bit Period Control for changing the general frequency of the colored noise.

Noise Frequency (0-31)

Register 7: This 6-bit register controls the mix; each of the 3 tone generators has one bit to turn on or off its square wave tone and another bit to turn on or off the noise generator in the mix. Set the bit to zero to enable that voice. For example, to enable only tones in C, nothing in B, and both tones and noise in A, load the register with a value equal to (32+16+0 + 0+2+0) or 'B110010' in binary.

| NoiseC (32) | NoiseB (16) | NoiseA (8) || ToneC (4) | ToneB (2) | ToneA (1) |

Registers 8-10: Another 3 registers control the amplitude of the 3 voices. Four bits of each register controls the logarithmic D/A converter which can set the amplitude of each voice mix. An extra single bit determines whether the voice amplitude will be controlled by this register's 4-bit amplitude control or by the shared Envelope Generator.

A Amplitude (0-15, 16 - Use Envelope)
B Amplitude (0-15, 16 - Use Envelope)
C Amplitude (0-15, 16 - Use Envelope)

Register 11: This 8-bit register is the Fine Tune adjustment for the Envelope Generator's Period.

Envelope Period - Fine (0-255)

Register 12: This 8-bit register is the Course Tune adjustment for the Envelope Generator's Period.

Envelope Period - Course (0-155)

Register 13: This is a 4-bit selection of the Envelope Shape.

Envelope Shape (0-15)

___ 0-3 Ramp Down, Stay Off
/|____ 4-7 Ramp Up, Immediate Off, Stay Off
|\|\ 8 Repeated Backward Sawtooth (Immediate Up, Ramp Down)
__ 9 Same as 0-3
\/\/ 10 Repeated Triangle (Ramp Down, Ramp Up)
\| 11 Ramp Down, Immediate Up, Stay On
/|/| 12 Repeated Sawtooth (Ramp Up, Immediate Down)
/ 13 Ramp Up, Stay On
/\/\ 14 Repeated Triangle (Ramp Up, Ramp Down)
/|__ 15 Same as 4-7

The AY Teletalk Box

The AY Teletalk box outputs the three AY synth chip voices, labeled A, B, and C. There is also a fourth voice, D, which comes from the digital output pin 6 on the Arduino board (named TONE_D). This pin outputs a square wave that can be conveniently programmed with the Arduino "tone" command which comes in two forms: tone(pin, frequency) or tone(pin, frequency, duration).

Volume controls for each of the four voices are on the back of the box. A voice select switch with 6 positions is on the front of the box. The switch settings are as follows:

Voice A Only
Voice B Only
Voice C Only
Voice D Only
All 4 Voices In Amplitude Modulation
A+B+C Mixed and Modulated with D

The top of the box has 3 switches (2 pushbuttons and one toggle) and 3 slide pots. These are connected to Arduino pins A0 – A5. The programmer can use these to affect the 4 synthesizer voices in any way they want. On the left side of the box is a USB jack used for programming the Arduino from a computer. It can also be used as a serial pipe for feeding data to the synth from a program like Cycling 74's Max/MSP.

Power to AY Teletalk comes from a power jack in the back. Please use a 9v DC power module with positive center. A regulator on the Arduino board drops the 9volts to 5volts for those circuit components needing a 5 volt supply. You may also be able to power the box, without its internal speaker, from the USB cable if it is connected to a powered hub.

The box includes an internal speaker for that tinny, old-time synth sound. For better fidelity use the front phone jack to connect to a good sound system.

The AY Teletalk box expands on the classic AY chip sounds with a special semi-digital Amplitude Modulation and an added Arduino voice. To hear the original AY chip sound use the 6^{th} switch setting with the D voice turned down. This setting will simply add the three chip voices together. Use the volume controls on the back to control the mix of the 3 voices. To add some spice to the mix turn up the volume on the D voice, which is wired to amplitude modulate the 3 voice chip mix. For a wild crazy sound, turn the switch to the 5^{th} position where all 4 voices are modulating each other. Play with the volume controls to change which voice or voices are the dominant modulators.

The Arduino Uno Controller

The AY-3-8912 synthesizer chip is wired to an Arduino Uno microprocessor. With the Arduino, a programmer can control when and with what data the 14 AY registers are loaded. An Arduino template program has been written to make this as easy as possible. By starting your program

from this template you need not worry about all the details of addressing and writing to the registers. Just use the functions provided in your main loop:

ldSynth(register number, register data)

ldFineTuneA(8-bit data)
ldFineTuneB(8-bit data)
ldFineTuneC(8-bit data)

ldCourseTuneA(4-bit data)
ldCourseTuneB(4-bit data)
ldCourseTuneC(4-bit data)

ldNoisePeriod(5-bit data)
ldEnable(ABC 6-bit noise/tone enable)

ldAmpA(5-bit data)
ldAmpB(5-bit data)
ldAmpC(5-bit data)

ldEnvCourseTune(8-bit data)
ldEnvFineTune(8-bit data)
ldEnvShape(4-bit data)

In addition to the 3 chip voices, an Arduino voice has been wired in. Digital Pin #6 has been assigned to Voice D. In the program template it is given the constant name TONE_D. For most applications, use the Arduino command "tone" to set the pitch and/or duration, as in tone(TONE_D, frequency) or tone(TONE_D, frequency, duration).

Three switches and 3 slide pots have been mounted to the top of the AY teletalk box. These are wired to the Arduino input pins A0 to A5. Pins A0, A1, and A2 have been defined in the template program as digital inputs and connected to the 3 switches. Pins A3, A4, and A5 have been define as analog inputs and connected to the 3 slide pots. Six variable have been defined for them and it is suggested that you use them as follows:

```
Button0 = digitalRead(A0);
Button1 = digitalRead(A1);
Toggle = digitalRead(A2);
Slider0 = analogRead(A3);
Slider1 = analogRead(A4);
Slider2 = analogRead(A5);
```

To reset the AY chip to silence load all the registers with zero. The template uses the following code to do this in "setup":

```
for (int x = 0; x < 14; x++){  ldSynth(x, 0); }
```

Programming the AY Teletalk

With just a little programming, you can turn the AY Teletalk into a unique performance synthesizer. Most of the detailed work of programming the AY Chip has been done for you in an AY_Template.ino program for the Arduino Uno. All that you need to add is the Arduino "loop" program. To get you started here are a couple example loop programs.

This first example illustrates some simple ways to use the sliders and buttons to affect the 4 voices of the AY Teletalk.

```
void loop() {


//---------------------------Button 0 Sets Arduino Voice D--------------------

              Button0 = digitalRead(A0);

              if (Button0) {
                tone(TONE_D,600, 10);
              }
              else{
                tone(TONE_D, 300);
              }

//-------------------------- Button 1 changes AY 3-voice pitches --------------

              Button1 = digitalRead(A1);
              if (Button1){
                ldFineTuneA(40);
              ldFineTuneB(45);
              ldFineTuneC(34);
              }
              else{
              ldFineTuneA(255);
              ldFineTuneB(55);
              ldFineTuneC(100);
              }


//---------------------------  Slider 1 adjusts the Noise Frequency -------------

          Slider1 = analogRead(A4) >> 5;  // shift the 10-bit read result to 5-bits
          ldNoisePeriod(Slider1); //Noise Period to Max

//---------------------------  Toggle Enables Noise or Tones on the AY voices -----------
```

```
        Toggle = digitalRead(A2);
        if (Toggle){
        ldEnable(B000111);  //Enable only noise (low enable)
        }
        else {
        ldEnable(B111000);  //Enable only tones (low enable)
        }

//-------------------------  Slider 2 controls the Envelope on the 3 AY voices ---------

        ldAmpA(B10000);  // Amplitude controlled by Envelope
        ldAmpB(B10000);
        ldAmpC(B10000);

        Slider2 = analogRead(A5) >> 6;
        ldEnvCourseTune(Slider2); //env period


        ldEnvShape(12);  //Decay with one cycle only

//-------------------------  Slider 0 controls the tempo ---------

        Slider0 = analogRead(A3) ;
        delay(50 + Slider0);

} //End of loop

//*************************************************************************
*****
//*************************************************************************
*****
//*************************************************************************
*****
```

This second example uses the Arduino random number generator to create random pitches with the center and spread of the random pitches controlled by sliders. Each AY voice uses exactly the same routine but the random generator creates different results.

Instead of using the AY chip's Envelope generator as in the previous example, amplitude ramps are created in the program and loaded directly into the AY Amplitude control registers.

Having just three slider pots may be rather limiting in some cases. Here, one of the buttons is used to give a slider two different functions depending on whether the button is depressed or released.

```
void loop() {


// -----------Voice D from Arduino D6 ------------------

Slider0 = analogRead(A3); //frequency of Arduino Tone D6
    if (digitalRead(A0)) {  // sharing Slider0 to set parameters by using Switch0
       dur = Slider0 >> 3; //set envelope durations
    }
    else{
      tone(TONE_D, Slider0);  // set frequencey of Voice D

    }

// -----------Toggle Switch turns on noise in one voice ------------------

            Toggle = digitalRead(A2);
            if (Toggle){
            ldEnable(B110000);  //Enable noise and tones in one voice (low enable)
            }
            else {
            ldEnable(B111000);  //Enable only tones (low enable)
            }


// ------------Voice A from AY Chip----------------------

if (envA != 0){  //ramping down voice A envelope, 15 to 0

        if (durA_count != 0){  //wait for a count of durA
            durA_count -= 1;
        }
          else {  // when the count reaches zero decrement voice A envelope, reset count
            durA_count = durA;
            envA -= 1;
            ldAmpA(envA);
          }
  }
  else{  // when envelope reaches zero, reset voice A with new frequency and envelope

        freq = getFreq();  //get new random pitch for voice A
```

```
        ldFineTuneA(freq & B11111111);  //use lower 8 bits for fine tune
        ldCourseTuneA(freq >>8);  //use higher bits for course tune

        durA = random(1, dur) ;  // get random 8 bit duration for envelope A

        durA_count = durA;
        envA = 15;
        ldAmpA(envA); //set voice A full on
  }



 // ------------Voice B from AY Chip----------------------

if (envB != 0){  //ramping down voice B envelope, 15 to 0

        if (durB_count != 0){  //wait for a count of durB
            durB_count -= 1;
        }
          else {  // when the count reaches zero decrement voice B envelope, reset count
            durB_count = durB;
            envB -= 1;
            ldAmpB(envB);
          }
 }
 else{  // when envelope reaches zero, reset voice B with new frequency and envelope

        freq = getFreq();  //get new random pitch for voice B
        ldFineTuneB(freq & B11111111);
        ldCourseTuneB(freq >>8);

        durB = random(1, dur) ;  // get random 8 bit duration for envelope B

        durB_count = durB;
        envB = 15;
        ldAmpB(envB); //set voice B full on
  }


// ------------Voice C from AY Chip----------------------

if (envC != 0){  //ramping down voice C envelope, 15 to 0

        if (durC_count != 0){  //wait for a count of durC
            durC_count -= 1;
        }
```

```
          else {  // when the count reaches zero decrement voice C envelope, reset count
             durC_count = durC;
             envC -= 1;
             ldAmpC(envC);
          }
  }
  else{  // when envelope reaches zero, reset voice C with new frequency and envelope

          freq = getFreq();  //get new random pitch for voice C
          ldFineTuneC(freq & B11111111);
          ldCourseTuneC(freq >>8);

          durC = random(1, dur) ;  // get random 8 bit duration for envelope C

          durC_count = durC;
          envC = 15;
          ldAmpC(envC); //set voice C full on
    }

 // ------------Switch 1 Slows everything to almost a standstill----------------------
     Button1 = digitalRead(A1);
    if (Button1 == 0){
     delay(250);
    }
} // End of Loop

int getFreq() {   // getting random frequency for Voices
    Slider2 = analogRead(A5) >> 2;  //base frequencey
    Slider1 = analogRead(A4)  ;  // range of frequencies around the base
    int basefreq = Slider2 + 10;
    int result = basefreq + random(Slider1);
    return result;

}

//*************************************************************************
*****
//*************************************************************************
*****
//*************************************************************************
*****
```

AY Data Streaming

The programming examples above are just two out of an unlimited number of possibilities for turning the AY Teletalk into a uniquely sounding performance device. However, this is not how the chip was traditionally used in the game machines of the 70s and 80s. In these machines, all 14 AY chip registers are continually updated every 20 milliseconds, whether the values change or not, from a large playback "song" file.

Because of this constant updating, voice envelopes are created by the data stream using the 4-bit Amplitude Control registers. Rarely is the chip's single Envelope Generator used. Because of this, the 14th register is most often given the value "FF" or 255, denoting that the Envelope Generator is not being used and this Envelope Shape register need not be loaded.

The .YM File Format

These "music" or "song" files are simply dumps of an AY chip frame every 20 milliseconds (50Hz). Each frame consists of 14 (sometimes 16) 8-bit "bytes" of data - a full update of all the data parameters of the AY synth. If you had a 5-minute song in which all 14 register are updated every 20ms, you would need a file of 210,000 bytes. These AY frame dump files were standardized into the YM file format created for ST-Sound by Arnaud Carre. The file format is freeware, so everybody can use, read, or produce YM files that will work for the AY chip synthesizer and all its relatives (http://leonard.oxg.free.fr http://ym2149.org/resources/YM-File-format.html). The YM file format adds useful information such as the length, song title, comments, and composer to the file. It also interleaves the data so that data for each register is gathered together in sequence. Since much of a register's data doesn't change from one frame to the next, this format allows for easy file compacting to a much smaller filesize.

A large body of YM files used in the old Atari, Amstrad, and Spectrum computers are still available on the Web:

http://www.cpcmuseum.com/
http://www.genesis8bit.fr/frontend/music.php
http://chipmusic.org/
ftp://ftp.modland.com/pub/modules/YM/

Even if you don't have a vintage computer with a sound chip, you can still listen to these YM files using a sound chip simulator program such as AudioOverload2 for the Mac by Richard Bannister (http://www.bannister.org/software/ao.htm).

Streaming YM files to the AY Teletalk

These same YM files can be played back on the AY sound chip in the Teletalk box. First of all, the YM file must be de-compressed using an LHA/LZH extractor. The Mac Unix system has one. Find the location of the lha command and, in the Terminal app, run lha -x myfile.ym

Next, the register data must be de-interlaced into consecutive frames of data bytes for the 14 AY chip registers. A C-program to do that has been created by Daniel Tufvesson (http://www.waveguide.se/?article=32&file=ymextract.c). He has also kindly provided 7 ready made de-interlaced YM.reg register files from the Modland archive – Cybernoid, Cybernoid II, Stormlord, Delta, Warhawk, Sidewinder, and Outrun (http://www.waveguide.se/?article=ym-playback-on-the-ymz284).

The Arduino Uno has limited memory space that is not large enough to hold an entire YM file. One solution would be to incorporate an Arduino shield with an SD card interface to hold the file. However, here I've chosen to create a Cycling '74 Max patch to feed the Arduino Uno the register frames from a desktop computer file through a USB serial line.

The Arduino part of this setup, YMLoadFromMax.ino, is very simple. It reads the data coming from Max over the serial line, while keeping track of the register number, and loads it into the chip. A reset button is created in case the data stream gets out of sequence. Also, the Arduino voice_D frequency set is put on a slider; though a more interesting possibility may be to derive it from one of the chip voice's CourseTune values.

```
y = 0;

void loop() {

 // ~~~~~~~~Add tone D for Modulation~~~~~~~~~~~~~~~~~~~~

     tone(TONE_D, (analogRead(A3) + 100));

 // ~~~~~~~~~~~~~~If A0 button is pushed, RESET the synth ~~~~~~~~

    if ( ! digitalRead(A0)) {
          while(  (Serial.available())  ) { val = Serial.read(); } //empty Serial buffer
          for (int x = 0; x < 14; x++){  ldSynth(x, 0);}  //reset the AY chip
          y=0;
      }   //  end Reset IF

 //~~~~~~~~~~~If serial data is available, Load it into an AY register ~~~~~~

          if ( Serial.available() ) {

               if (y==13) { // last YM synth register, load if not FF, reset register count to 0
                     val = Serial.read();
                     if (val != 0xFF) { ldSynth(y, val); }
                     y=0;

               } else {
                     ldSynth(y, Serial.read() );  //fill AY register y with serial data
                     y += 1;
```

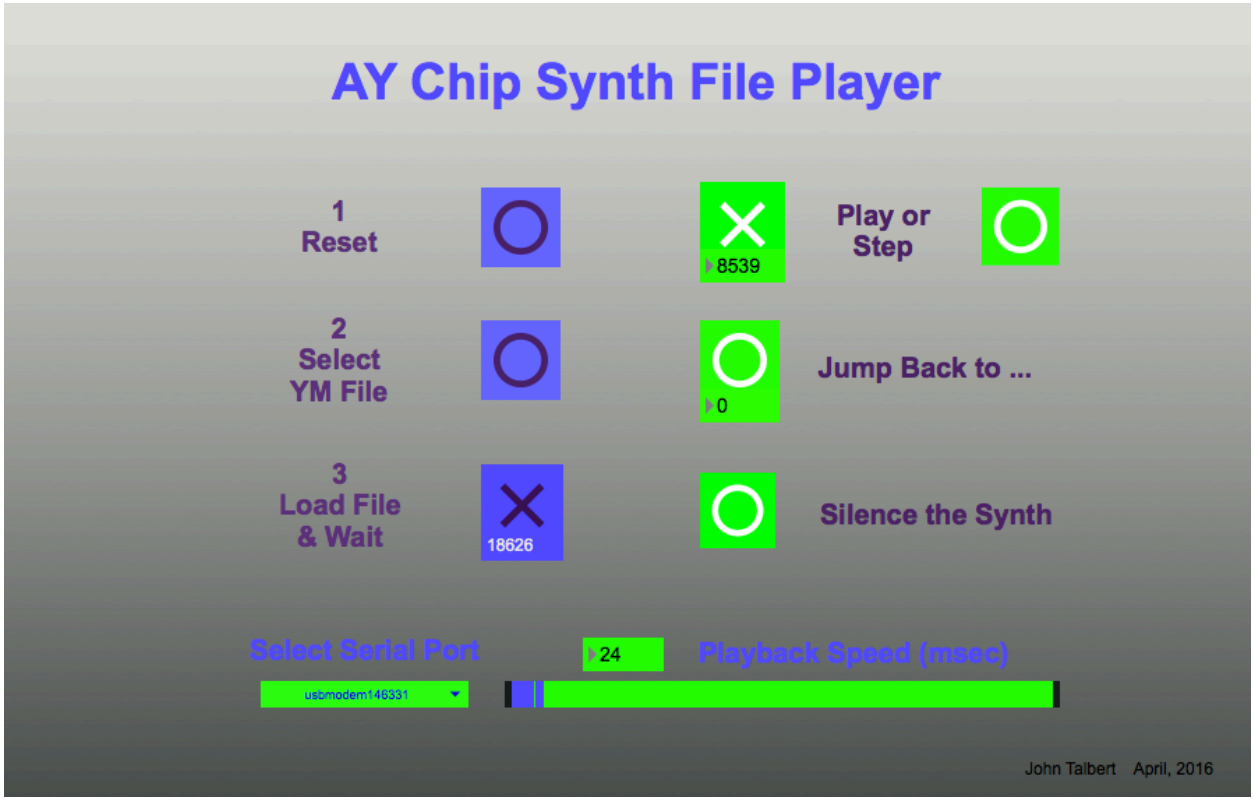
```
            } //End of else

        } //End of if Serial.available

}  // End of MAIN LOOP

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

The Max Patch YMfeed.maxpat is shown below. The first step in using the patch is to load a de-interlaced YM file. The actual patch uses a "filein" object to fill a "coll" object.



The "Reset" button erases any previous YM file in the patch. The "Select YM File" button brings up a dialog box for you to choose a YM File. Finally, the "Load File &Wait" button starts the upload. While uploading, a counter below this button will increment and eventually stop at the total number of frames uploaded in the file.

To play the File on the AY Teletalk box, first select the serial port to which the Arduino Uno is connected. It will be one of the non-Bluetooth ports. Set the Playback speed at the usual speed of 20ms, though you have the option of playback up to twice the speed (10ms) or down to super slow. Speed values can be entered either from the slider or by entering a millisecond value in the number box. Finally, hit the "Play" button for playback at the speed you chose; the button includes a frame counter. There is also a button for stepping through the frames one at a time

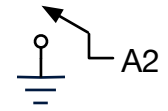
after the “Play” button has been stopped. The Synth will hold the last note after a stop, which you can turn off by hitting the “Silence” button. The “Jump Back” button has a number box for entering any frame you might want to jump to on hitting this button.

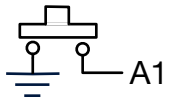
AY Teletalk 1/2

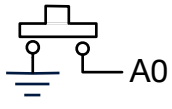
John Talbert April 2016

Arduino Inputs A0-A5

Button0 = digitalRead(A0);

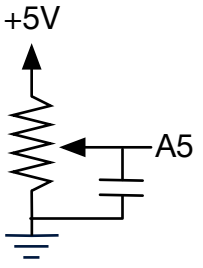


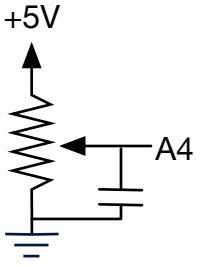


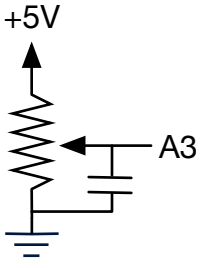


Switches use internal pullups

pinMode(A0, INPUT);
digitalWrite(A0, HIGH);

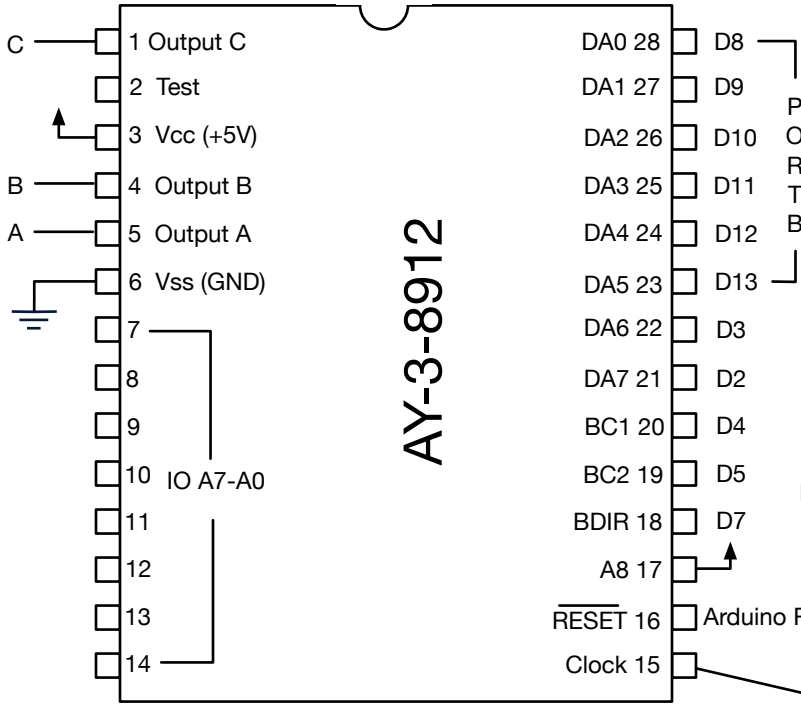






Slider0 = analogRead(A3);

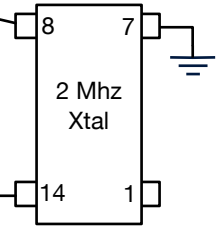
Arduino Digital Pins D0-D13



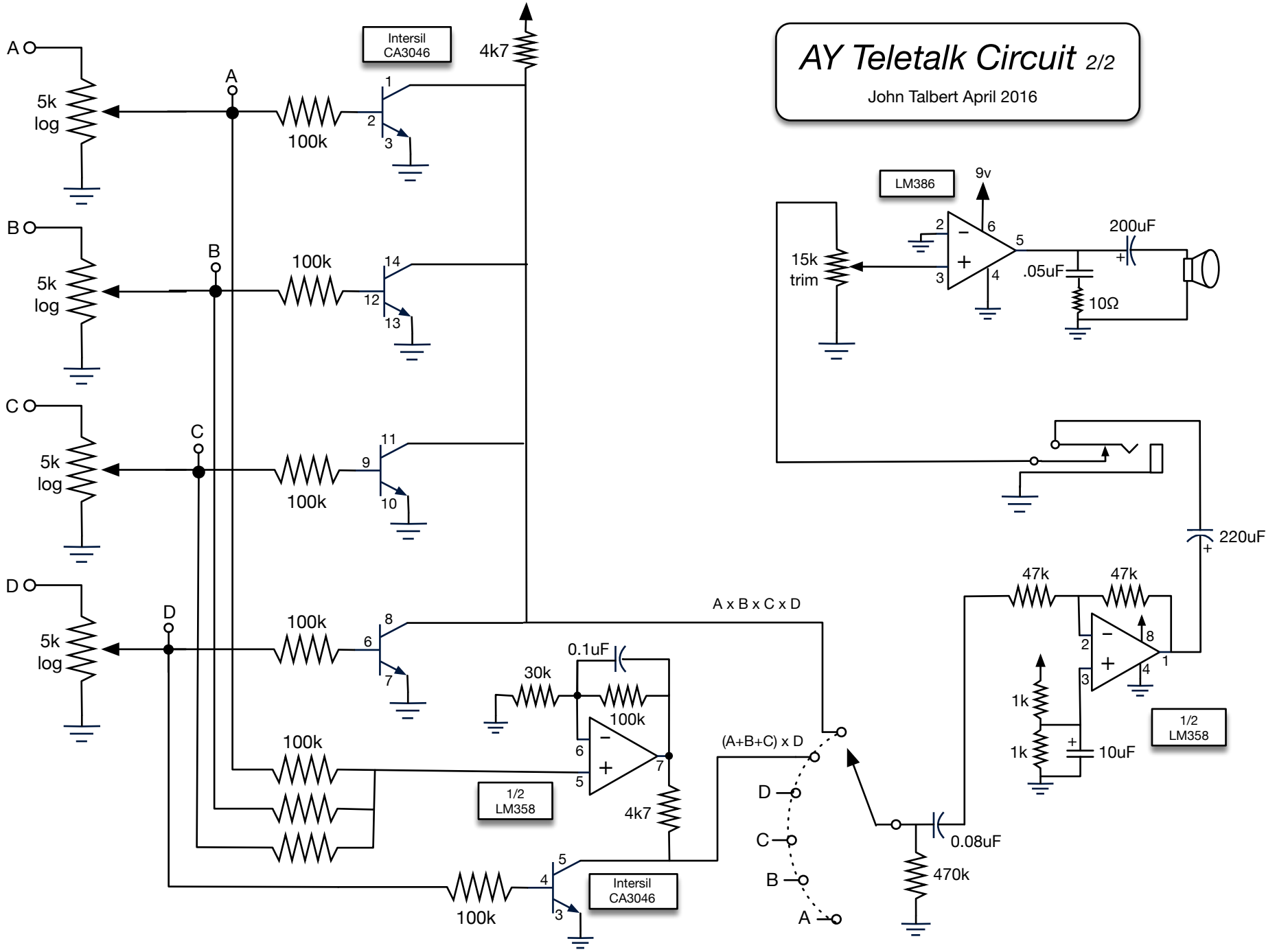
DDRB = B0011 1111;
PORTB =

D6 - Tone D

D0,D1 - Serial I/O



BDIR	BC2	BC1	
0	0	0	Inactive
1	0	0	Latch Address
0	1	0	Inactive
1	1	0	Write Data


AY Teletalk Circuit 2/2
John Talbert April 2016
Intersil CA3046
4k7
A
5k log
100k
B
C
D
LM386
9v
15k trim
200uF
.05uF
10Ω
220uF
47k
1k
10uF
1/2 LM358
0.1uF
30k
A x B x C x D
(A+B+C) x D
0.08uF
470k

OBSOLETE PRODUCT
RECOMMENDED REPLACEMENT PART
HFA3046

December 15, 2011 File Number 341.6

General Purpose NPN Transistor Array

The CA3046 consists of five general purpose silicon NPN transistors on a common monolithic substrate. Two of the transistors are internally connected to form a differentially connected pair.

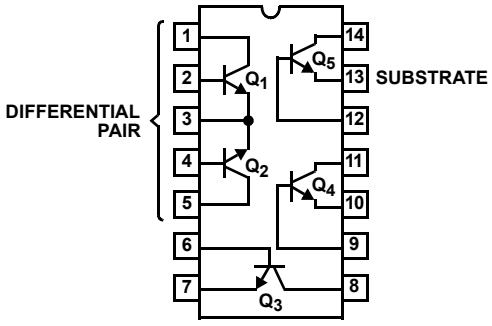
The transistors of the CA3046 are well suited to a wide variety of applications in low power systems in the DC through VHF range. They may be used as discrete transistors in conventional circuits. However, in addition, they provide the very significant inherent integrated circuit advantages of close electrical and thermal matching.

Ordering Information

PART NUMBER (BRAND)	TEMP. RANGE (^oC)	PACKAGE	PKG. NO.
CA3046	-55 to 125	14 Ld PDIP	E14.3
CA3046M (3046)	-55 to 125	14 Ld SOIC	M14.15
CA3046M96 (3046)	-55 to 125	14 Ld SOIC Tape and Reel	M14.15

Pinout

CA3046 (PDIP, SOIC)
TOP VIEW



Features

- Two Matched Transistors
  - V_{BE} Match . . . . . ±5mV
  - I_{IO} Match. . . . . .2μA (Max)
- Low Noise Figure . . . . . 3.2dB (Typ) at 1kHz
- 5 General Purpose Monolithic Transistors
- Operation From DC to 120MHz
- Wide Operating Current Range
- Full Military Temperature Range

Applications

- Three Isolated Transistors and One Differentially Connected Transistor Pair for Low Power Applications at Frequencies from DC Through the VHF Range
- Custom Designed Differential Amplifiers
- Temperature Compensated Amplifiers
- See Application Note, AN5296 “Application of the CA3018 Integrated-Circuit Transistor Array” for Suggested Applications

Absolute Maximum Ratings

Collector-to-Emitter Voltage (V_{CEO}) 15V
Collector-to-Base Voltage (V_{CBO}) 20V
Collector-to-Substrate Voltage (V_{CIO}, Note 1) 20V
Emitter-to-Base Voltage (V_{EBO}) 5V
Collector Current (I_C) 50mA

Operating Conditions

Temperature Range . . . . . -55^oC to 125^oC

Thermal Information

Thermal Resistance (Typical, Note 2)	θ_{JA} ($^oC/W$)	θ_{JC} ($^oC/W$)
PDIP Package	180	N/A
SOIC Package	220	N/A

Maximum Power Dissipation (Any One Transistor) 300mW
Maximum Junction Temperature (Plastic Package) . . . . . 150^oC
Maximum Storage Temperature Range . . . . . -65^oC to 150^oC
Maximum Lead Temperature (Soldering 10s) . . . . . 300^oC
(SOIC - Lead Tips Only)

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTES:

1. The collector of each transistor of the CA3046 is isolated from the substrate by an integral diode. The substrate (Terminal 13) must be connected to the most negative point in the external circuit to maintain isolation between transistors and to provide for normal transistor action.
2. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications

$T_A = 25^oC$, characteristics apply for each transistor in CA3046 as specified

PARAMETER	SYMBOL	TEST CONDITIONS		MIN	TYP	MAX	UNITS
DC CHARACTERISTICS							
Collector-to-Base Breakdown Voltage	$V_{(BR)CBO}$	$I_C = 10\mu A$, $I_E = 0$		20	60	-	V
Collector-to-Emitter Breakdown Voltage	$V_{(BR)CEO}$	$I_C = 1mA$, $I_B = 0$		15	24	-	V
Collector-to-Substrate Breakdown Voltage	$V_{(BR)CIO}$	$I_C = 10\mu A$, $I_{CI} = 0$		20	60	-	V
Emitter-to-Base Breakdown Voltage	$V_{(BR)EBO}$	$I_E = 10\mu A$, $I_C = 0$		5	7	-	V
Collector Cutoff Current (Figure 1)	I_{CBO}	$V_{CB} = 10V$, $I_E = 0$		-	0.002	40	nA
Collector Cutoff Current (Figure 2)	I_{CEO}	$V_{CE} = 10V$, $I_B = 0$		-	See Fig. 2	0.5	μA
Forward Current Transfer Ratio (Static Beta) (Note 3) (Figure 3)	h_{FE}	$V_{CE} = 3V$	$I_C = 10mA$	-	100	-	-
			$I_C = 1mA$	40	100	-	-
			$I_C = 10\mu A$	-	54	-	-
Input Offset Current for Matched Pair Q_1 and Q_2. $\|I_{IO1} - I_{IO2}\|$ (Note 3) (Figure 4)		$V_{CE} = 3V$, $I_C = 1mA$		-	0.3	2	μA
Base-to-Emitter Voltage (Note 3) (Figure 5)	V_{BE}	$V_{CE} = 3V$	$I_E = 1mA$	-	0.715	-	V
			$I_E = 10mA$	-	0.800	-	V
Magnitude of Input Offet Voltage for Differential Pair $\|V_{BE1} - V_{BE2}\|$ (Note 3) (Figures 5, 7)		$V_{CE} = 3V$, $I_C = 1mA$		-	0.45	5	mV
Magnitude of Input Offset Voltage for Isolated Transistors $\|V_{BE3} - V_{BE4}\|$, $\|V_{BE4} - V_{BE5}\|$, $\|V_{BE5} - V_{BE3}\|$ (Note 3) (Figures 5, 7)		$V_{CE} = 3V$, $I_C = 1mA$		-	0.45	5	mV
Temperature Coefficient of Base-to-Emitter Voltage (Figure 6)	$\frac{\Delta V_{BE}}{\Delta T}$	$V_{CE} = 3V$, $I_C = 1mA$		-	-1.9	-	$mV/^oC$
Collector-to-Emitter Saturation Voltage	V_{CES}	$I_B = 1mA$, $I_C = 10mA$		-	0.23	-	V
Temperature Coefficient: Magnitude of Input Offset Voltage (Figure 7)	$\frac{\lvert\Delta V_{IO}\rvert}{\Delta T}$	$V_{CE} = 3V$, $I_C = 1mA$		-	1.1	-	$\mu V/^oC$
DYNAMIC CHARACTERISTICS							
Low Frequency Noise Figure (Figure 9)	NF	f = 1kHz, $V_{CE} = 3V$, $I_C = 100\mu A$, Source Resistance = $1k\Omega$		-	3.25	-	dB
Low Frequency, Small Signal Equivalent Circuit Characteristics							
Forward Current Transfer Ratio (Figure 11)	h_{FE}	f = 1kHz, $V_{CE} = 3V$, $I_C = 1mA$		-	110	-	-
Short Circuit Input Impedance (Figure 11)	h_{IE}	f = 1kHz, $V_{CE} = 3V$, $I_C = 1mA$		-	3.5	-	$k\Omega$
Open Circuit Output Impedance (Figure 11)	h_{OE}	f = 1kHz, $V_{CE} = 3V$, $I_C = 1mA$		-	15.6	-	μS

LM2904,LM358/LM358A,LM258/ LM258A

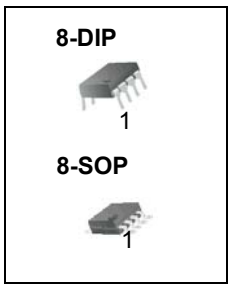
Dual Operational Amplifier

Features

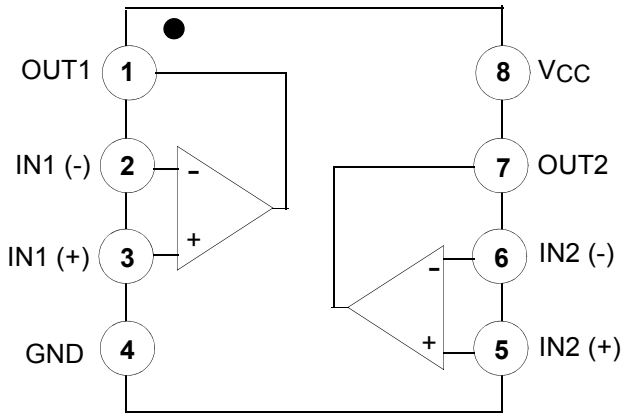
- Internally Frequency Compensated for Unity Gain
- Large DC Voltage Gain: 100dB
- Wide Power Supply Range:
  LM258/LM258A, LM358/LM358A: 3V~32V (or ±1.5V ~ 16V)
  LM2904 : 3V~26V (or ±1.5V ~ 13V)
- Input Common Mode Voltage Range Includes Ground
- Large Output Voltage Swing: 0V DC to Vcc -1.5V DC
- Power Drain Suitable for Battery Operation.

Description

The LM2904,LM358/LM358A, LM258/LM258A consist of two independent, high gain, internally frequency compensated operational amplifiers which were designed specifically to operate from a single power supply over a wide range of voltage. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage. Application areas include transducer amplifier, DC gain blocks and all the conventional OP-AMP circuits which now can be easily implemented in single power supply systems.



Internal Block Diagram



Electrical Characteristics (Continued)

(V_{CC}= 5.0V, V_{EE} = GND, unless otherwise specified)
The following specification apply over the range of -25°C ≤ T_A ≤ +85°C for the LM258; and the 0°C ≤ T_A ≤ +70°C for the LM358; and the -40°C ≤ T_A ≤ +85°C for the LM2904

Parameter	Symbol	Conditions	LM258			LM358			LM2904			Unit
			Min.	Typ.	Max.	Min.	Typ.	Max.	Min.	Typ.	Max.	
Input Offset Voltage	V_{IO}	V_{CM} = 0V to V_{CC} -1.5V $V_{O(P)}$ = 1.4V, R_S = 0Ω	-	-	7.0	-	-	9.0	-	-	10.0	mV
Input Offset Voltage Drift	$\Delta V_{IO}/\Delta T$	R_S = 0Ω	-	7.0	-	-	7.0	-	-	7.0	-	μV/°C
Input Offset Current	I_{IO}	-	-	-	100	-	-	150	-	45	200	nA
Input Offset Current Drift	$\Delta I_{IO}/\Delta T$	-	-	10	-	-	10	-	-	10	-	pA/°C
Input Bias Current	I_{BIAS}	-	-	40	300	-	40	500	-	40	500	nA
Input Voltage Range	$V_{I(R)}$	V_{CC} = 30V (LM2904 , V_{CC} = 26V)	0	-	V_{CC} -2.0	0	-	V_{CC} -2.0	0	-	V_{CC} -2.0	V
Large Signal Voltage Gain	G_V	V_{CC} = 15V, R_L =2.0kΩ $V_{O(P)}$ = 1V to 11V	25	-	-	15	-	-	15	-	-	V/mV
Output Voltage Swing	$V_{O(H)}$	V_{CC}=30V (V_{CC} = 26V for LM2904) R_L = 2kΩ	26	-	-	26	-	-	22	-	-	V
		R_L=10kΩ	27	28	-	27	28	-	23	24	-	V
	$V_{O(L)}$	V_{CC} = 5V, R_L=10kΩ	-	5	20	-	5	20	-	5	20	mV
Output Current	I_{SOURCE}	$V_{I(+)}$ = 1V, $V_{I(-)}$ = 0V, V_{CC} = 15V, $V_{O(P)}$ = 2V	10	30	-	10	30	-	10	30	-	mA
	I_{SINK}	$V_{I(+)}$ = 0V, $V_{I(-)}$ = 1V, V_{CC} = 15V, $V_{O(P)}$ = 2V	5	8	-	5	9	-	5	9	-	mA
Differential Input Voltage	$V_{I(DIFF)}$	-	-	-	V_{CC}	-	-	V_{CC}	-	-	V_{CC}	V