The ATMega1284P Guitar and Music Effects Pedal

I have ported the Arduino Uno ATMega328 Pedalshield (as developed by Electrosmash and in part based on work at the Open Music Lab) to the ATMega1284P which has eight times more RAM than the Uno (16kB versus 2kB). An additional unexpected benefit is that the Mega1284 build has much lower noise component – to the extent that when I compare the Uno and the Mega1284 using the same support circuitry it is not unreasonable to describe the Uno as “noisy” and the Mega1284 as “quiet”. The larger RAM means that a much longer delay effect can be obtained – and that is demonstrated by the Arduino sketch example that I have included. The background breathing noise when using the Tremelo effect is also (almost) absent with the ATMega1284.

A comparison of three of the Atmel AVR microprocessors namely the 328P that is the Uno, the 2560P that is the Mega2560, and the Mega1284 shows the latter to have the most RAM of the three:

Aspect 328P 1284P 2560P

RAM 2k 16k 8k

Flash 32k 128k 256k

EEPROM 1k 4k 4k

UART 1 2 4

IO Pins 23 32 86

Interrupts 2 3 8

Analog Inputs 6 8 16

I started by bread-boarding the Uno-based pedalSHIELD as in the Electrosmash specification but I did not have the same RRO OpAmp as specified. As a result I ended up with a circuit which I considered to give acceptable results. The details of this Uno version are given in appendix 2.

This same circuit was then ported to the ATMega1284 – surprisingly apart from the non-essential changes such as assigning the switches and LED to a different port, and allocating 12,000 kB instead of 2,000 kB of RAM for the delay buffer, only one essential change had to be made in the source code, namely changing the Timer1/PWM OC1A and OC1B outputs from Port B on the Uno to Port D (PD5 and PD4) on the ATMega1284.

I later discovered the excellent modifications to the electrosmash circuit by Paul Gallagher and after testing, this is the circuit I will present here - but then also with the substitution of the Uno with the Mega1284 and using a Texas Instruments TLC2272 as the OpAmp, and because of the excellent noise performance of the Mega1284 I could also raise the low-pass filter frequency level.

It is important to note that although development boards for the ATMega1284 are available (Github: MCUdude MightyCore), it is an easy exercise to buy the bare (bootloader-free) chip (buy the PDIP version which is bread-board and strip-board friendly), then load the Mark Pendrith fork of the Maniacbug Mighty-1284p Core Optiboot bootloader or the MCUdude Mightycore, by using an Uno as the ISP programmer, and then loading sketches again via the Uno to the AtMega1284. Details and links for this process are given in appendix 1.

I would like to acknowledge the three most important sources from which further information can be obtained and will give links to their websites and the end of this article: Electrosmash, Openlabs Music, and Tardate/Paul Gallagher

Construction and Parts list:  
Parts list:

ATMega1284P (PDIP 40 pin package version)

Arduino Uno R3 (used as an ISP to transfer the bootloader and sketches to the ATMega1284)

OpAmp TLC2272 (or similar RRIO (Rail to Rail Input and Output) OpAmp such as MCP6002, LMC6482, TL972)

Red LED

16 MHz crystal

2 x 27 pF capacitors

5 x 6n8 capacitors

270 pF capacitor

4 x 100n capacitors

2 x 10uF 16v electrolytic capacitors

6 x 4k7 resistors

100k resistor

2 x 1M resistors

470 ohm resistor

1M2 resistor

100k Potentiometer

3 x pushbutton switches (one of them should be substituted with a 3-pole 2-way footswitch if the effects box is going to be used for live work)

Schematic 1 gives the circuit used and Breadboard 1 is its physical representation (Fritzing 1) with Photo 1 the actual bread-boarded circuit in operation. It may be advantageous to have a potentiometer as a mixer for the dry (equal to the input) and the wet (after processing by the MCU) signal, and Schematic 2, Breadboard 2 and Photo 2 gives the circuit details of a previously constructed circuit which incorporates such an input to output mixer. Also look at the Open Music Labs StompBox for another mixer implementation using four OpAmps.

OpAmp Input and Output Stages: It is important that a RRO or preferably a RRIO OpAmp is used because of the large voltage swing required at the OpAmp output to the ADC of the ATMega1284. The parts list contains a number of alternative OpAmp types. The 100k potentiometer is used to adjust the input gain to a level just below any distortion, and it can also be used to adjust the input sensitivity for an input source other than a guitar such as a music player. The OpAmp output stage has a higher order RC filter to remove the digitally generated MCU noise from the audio stream.

ADC Stage: The ADC is configured to be reading via an interrupt all the whole time. Note that a 100nF capacitor should be connected between the AREF pin of the ATMega1284 and ground to reduce noise as an internal Vcc source is used as a reference voltage – do NOT connect the AREF pin to +5 volt directly!

DAC PWM Stage: As the ATMega1284 does not have its own DAC, the output audio waveforms are generated using a pulse width modulation of an RC filter. The two PWM outputs on PD4 and PD5 are set as the high and low bytes of the audio output and mixed with the two resistors (4k7 and 1M2) in a 1:256 ratio (low byte and high byte) – which generates the audio output. It may be worthwhile experimenting with other resistor pairs such as the 3k9 1M ohm pair used by Open Music Labs in their StompBox.

Software:

The software is based on the electrosmash sketches, and the example included (pedalshield1284delay.ino), has been adapted from their Uno delay sketch. Some of the switches and LED had been moved to other ports away from those used by the ISP programmer (SCLK, MISO, MOSI and Reset), the delay buffer has been increased from 2000 bytes to 12000 bytes, and PortD has been set as the output for the two PWM signals. Even with the increase in the delay buffer the sketch still use only about 70% of the available 1284 RAM.

Other examples such as the octaver or tremolo from the electrosmash website for the pedalSHIELD Uno can be adapted for use by the Mega1284 by changing three sections in the code:

(1) Change

DDRB |= ((PWM\_QTY << 1) | 0x02);

to

DDRD |= 0x30;

// This the ONLY essential code change when moving from the AtMega328 to the ATMega1284   
  
(2) Change

#define LED 13

#define FOOTSWITCH 12

#define TOGGLE 2

#define PUSHBUTTON\_1 A5

#define PUSHBUTTON\_2 A4  
  
to  
  
#define LED PB0

#define FOOTSWITCH PB1

#define PUSHBUTTON\_1 A5

#define PUSHBUTTON\_2 A4

(3) Change  
 pinMode(FOOTSWITCH, INPUT\_PULLUP);

pinMode(TOGGLE, INPUT\_PULLUP);

pinMode(PUSHBUTTON\_1, INPUT\_PULLUP);

pinMode(PUSHBUTTON\_2, INPUT\_PULLUP);

pinMode(LED, OUTPUT)

to

pinMode(FOOTSWITCH, INPUT\_PULLUP);

pinMode(PUSHBUTTON\_1, INPUT\_PULLUP);

pinMode(PUSHBUTTON\_2, INPUT\_PULLUP);

pinMode(LED, OUTPUT);

The pushbuttons 1 and 2 are used in some of the sketches to increase or decrease an effect. In the delay example it increases or decreases the delay time. When the sketch is first loaded it starts with the maximum delay. Press the down button (it takes about 15 seconds if held down to reach the delay-off position), and then press the up button once a few times and listen how the effect changes from a chorus, phaser, flanging to delay as the up button is pressed. The footswitch should be a three pole two way switch and must be connected as described on the electrosmash website.

Links:  
(1) Electrosmash: <https://www.electrosmash.com/pedalshield-uno>  
 License: <https://creativecommons.org/licenses/by-nc/3.0/>

(2) Openlabs Music: <http://wiki.openmusiclabs.com/wiki/StompShield>  
 License: <http://www.openmusiclabs.com/>

(3) Paul Gallagher: <https://github.com/tardate/LittleArduinoProjects/tree/master/playground/AudioDSP>  
 License: <https://github.com/tardate/LittleArduinoProjects/blob/master/LICENSE> (MIT)  
(4) 1284 Bootloader: <https://github.com/JChristensen/mighty-1284p>

<https://github.com/maniacbug/mighty-1284p>  
(5) ATmega1284 8bit AVR Microcontroller: <https://www.microchip.com/wwwproducts/en/ATmega1284>

Appendix 1 Programming the ATMega1284P

There are a few websites that gives a good explanation on how to program the bare ATMega1284 chip for use with the Arduino IDE. The process is essentially as follows:

(1) Install the Mark Pendrith fork of the Maniacbug Mighty-1284p Core Optiboot bootloader into the Arduino IDE.

(2) Wire up the ATMega1284 on a breadboard with its minimum configuration that is a 16 MHz crystal, 2 x 22 pF capacitors that grounds the two end of the crystal, Connect the two ground pins together (pins 11 and 31), and then to the Arduino Uno ground, connect the Vcc and AVcc together (pins 10 and 30), and then to the Uno +5v, then connect the reset pin 9 to the Uno D10 pin, the MISO pin 7 to the UNO D12, The MOSI pin 8 to the Uno D11, and the SCLK pin 7 to the Uno D13 pin.

(3) Connect the Uno to the Arduino IDE and load the sketch example Arduino as ISP onto the Uno.

(4) Now select the 1284 “maniac” mighty optiboot board, and choose the Burn bootloader option.

(5) Then select the 1284 delay sketch given here as an example and upload it by using use the Uno as programmer option in the sketches menu.

Links that explain the process in more detail are:

<http://www.technoblogy.com/show?19OV>  
<https://github.com/MCUdude/MightyCore>

<https://dave.cheney.net/2014/12/31/building-an-atmega1284p-prototype>

<https://forum.arduino.cc/index.php?topic=137559.0>

Appendix 2 Arduino Uno pedalSHIELD variation

Schematic3, Breadboard3, and Photo3 gives details of the Uno-based circuit that preceded the AtMega1284 build.