

Proper Debugging of ATSAMD21 Processors

Created by lady ada

<pre>pinMode(13, OUTPUT); } // the loop function runs { init loop() { function runs {</pre>	over	and c	over	again forever
voru 100b() {		+	the	LED on (HTGH is the weltage level)
digitalWrite(13, HIGH);	11	curm	une	rep ou (urou is cue voirage ievei)
<pre>digitalWrite(13, HIGH); delay(1000);</pre>	11	wait	for	a second
<pre>digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW);</pre>	11	wait turn	for the	a second LED off by making the voltage LOW
<pre>digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000);</pre>	""	wait turn wait	for the for	a second LED off by making the voltage LOW a second

Last updated on 2018-03-28 04:35:51 PM UTC

Guide Contents

Guide Contents	2
Overview	3
Install Software	5
Arduino IDE	5
J-Link Software	5
Atmel Studio 7	5
Let's go!	7
Load an Arduino Sketch in Atmel Studio 7	7
Set Up and Check Interface	9
Arduino Zero Debug port	10
J-Link to SWD	10
Identify Interface	12
Build & Start Debugging	13
Paths and Optimizations	21
Correcting Paths to Necessary Files	23
Fixing Some Core Files	24
Restoring Bootloader	26
Arduino Zero	26
Feather M0 or Others	26

Overview

```
// the setup function runs once when you press reset or power the board

=void setup() {

    // initialize digital pin 13 as an output.

    pinMode(13, OUTPUT);

    }

    // the loop function runs over and over again forever

=void loop() {

    digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)

    delay(1000); // wait for a second

    digitalWrite(13, LOW); // turn the LED off by making the voltage LOW

    delay(1000); // wait for a second

    }

}
```

Chances are if you're programming firmware on a microcontroller you've had to do some 'fun' debugging. Button presses, interrupts, small memory spaces...it can make debugging quite a challenge! A lot of beginners lean on triedand-true (if a little frustrating) printf statments or toggling GPIO's with LEDs on them. And don't get me wrong, those techniques work pretty well. But if you come from a software background you're probably used to really nice debugging setups, often built into the IDE

Of course its a ton easier to debug software when the computer is running both software and development platform. It's a lot tougher when the processor is physically separated, with its own memory, clocks, peripherals, and its not even of the same *processor family*!



No worries though, there's a full industry set up to create programming/debug dongles and adapters!One of our favorites is SEGGER's J-Link family. They're not *cheap* but they do support a vast number of chips.

Atmel also has it's own debugger chip, the EDBG (apparently its a AT32UC3A4256 programmed with Atmels proprietary firmware)

This chip comes on every Arduino Zero and is used to both program and debug firmware



You may be wondering "OK so how do I actually do said debugging?" Well you've come to the right place because we're gonna show you how. In this guide we'll show how to debug the ATSAMD21 family (specifically the ATSAMD21G18) which is in the Arduino Zero and Feather MO family, by using the EDBG or J-Link.

Install Software

Before you begin you will need some software. Here's what we're using:



Arduino IDE

As of this writing, 1.6.7 is the latest so we're using that. We also installed the Arduino SAMD support and/or Adafruit SAMD support (for Feather M0)

Make sure you also have drivers set up for the board you're using, and get a sketch working and uploaded to the board. That means you have the IDE and package set up, which is something you want done **before** you continue



J-Link Software

If you're using a J-Link, install all software and drivers for it and run the J-Link commander to make sure you update the firmware, new firmware is constantly being released so best to update your 'Link!



Atmel Studio 7

Here's where the Mac and Linux people will be sad. This is the IDE software that can do step&memory debugging and its only for Windows. Also you have to make an account on Atmel's site, download it from here

Software	Description
<u>@</u>	Atmel Studio 7.0 (build 790) web installer (recommended) (2.38MB, updated February 2016)
	View minimum system requirements
	This installer contains Atmel Studio 7.0 with Atmel Software Framework 3.30.1 and Atmel Toolchains. It is recommended to use this installer if you have internet access while installing, since it enables incremental updates in the future.
<u>(</u>	Atmel Studio 7.0 (build 790) offline installer (823MB, updated February 2016)
	View minimum system requirements
	This installer contains Atmel Studio 7.0 with Atmel Software Framework 3.30.1 and Atmel Toolchains. Use this installer if you do not have internet access while installing. It is highly recommended to use the smaller web installer if you can since it provides the ability to get incremental updates in the future.
	SHA. 8th0h52e1f34321191a3h803058ehf840af0e9e2

Make sure you have the latest version, we used build 790

Let's go!

Note that by uploading a debug sketch you will blow away the bootloader on your Arduino Zero or Feather M0, see the next section for re-loading it!

OK now that we have all that software, the rest isn't too tough!

Load an Arduino Sketch in Atmel Studio 7

Start by launching Atmel Studio 7

Start Page - AtmelStudio Elle Edit View VAssisty ASF Proje	ct Debug Jools Window Help		Quick Launch (Ctrl+Q)	P -	• ×
0-0 8-0 0-0 0 0	「日の「フ・C・画像」を用い、Debug Browser・	- 🏓 Mismalijder	- ភ / © E	a 🗆 - 🖓	000
Start Page • ×		ASF Explorer			• # ×
Start	Discover Atmel Studio	î.			
New Example Project	Getting started with Atmel Studio				
Open Project	Getting started with AVR development				
Recent	Open Atmel Start Configurator				
WINC1500_CHIP_INFO_EXAMPLE1	Download Atmel Studio Extensions				
WINC1500_CHIP_INFO_EXAMPLE3	Download documentation				
Show feeds Keep page open after project load	Announcements				
Show page on startup	This could be to the fact of the same the datasets	with a set of the set	orer		
Find Results 1					* # X
NITE TITLE					
Output Find Results 1				_	•
Ready					

Create a new Project

ŏ	Start Page - AtmelS	itudio							
File	Edit View VA	AssistX ASF	Project De	bug	Tools	Window	Help		
	New		,	<u>ئە</u>	Project	t		Ctrl+Shift+N	ebu
	Open		,	· *ɔ	File			Ctrl+N	83
	Close				Atmel	Start Config	urator		
×	Close Solution			æ	Examp	le Project		Ctrl+Shift+E	
	Import		,						-
Р	Save Selected Items		Ctrl+S						
	Save Selected Items	As							
	Save All		Ctrl+Shift+S		55				
	Export Template								
						A +	-1 CL	1.111	

And select Create project from Arduino sketch

Recent	Sort by:	Default - 📰 📃		Search Installed Templates (Ctrl+E)
Installed C/C++ Assembler AtmelStudio Solo	ution	GCC C ASF Board Project GCC C Executable Project GCC C Static Library Project GCC C++ Executable Project GCC C++ Static Library Project	C/C++ C/C++ C/C++ C/C++ C/C++	Type: C/C++ Creates an Atmel Studio project from Arduino sketch file
Name: .ocation:	GccCppProject1 C:\Users\ladyada\Dropbox	k (Personal)∖Atmel∖AtmelStudio\7.0		Browse

Navigate to your arduino sketchfolder and select the sketch. I recommend starting with the easy-to-understand Blink

Organize 🔻 Ne	ew folder	r			8= • 🗖	(
	^	Name	Date modified	Туре	Size	
Libraries		🛃 Blink.ino	2/25/2016 3:01 PM	INO File	1 KB	:
Pictures						

Also select the Arduino IDE location if necessary. For **Board** go with **Arduino/Genuino Zero (Programming Port)** and under Device, **ATSAMD21G18A**

Create C++ project	from Arduino sketch
Sketch File	C:\Users\ladyada\Dropbox\Arduino\arduino-1.6.7\examples\01.Basics\Blink\Blink
Arduino IDE Path	C:\Users\ladyada\Dropbox\Arduino\arduino-1.6.7
Board	Arduino/Genuino Zero (Programming Pc 💌
Device	ATSAMD21G18A
	<u>Cancel</u> <u>O</u> k

You'll see the following, where the sketch is in a window, you can edit the code here if you like. For now just leave it as is.



Set Up and Check Interface

OK next up we'll attach the chip & debugger. You have two options:

Arduino Zero Debug port

This is super easy, just connect a USB micro B cable to your Arduino Zero



Make sure you're plugged into the DEBUG port not the 'native' port

J-Link to SWD

If you have a board without an EDBG chip on it, you can still debug, but you'll need a helper such as a J-Link.We like using this handy adapter board



To get the large J-Link cable do the 'classic' 2x5 SWD cable connector



If you are debugging a board that doesn't even have an SWD connector on it, you may need to solder to the SWD pads



You need to connect the following to the J-Link:

- Vref / Vtarget Logic voltage of the chip, in this case 3.3V
- **GND** to common ground
- SWDIO to SWDIO
- SWCLK to SWCLK

I haven't found I need to connect the chip's RESET line

Identify Interface

OK now you have your debugger plugged in, its good to check that it works, select Device Programming



you can control. On the Uno and l pin 13. If you're unsure what to on your Arduino model. check

Under Tool make sure you can select EDBG or J-Link

Device Programming Tool Device EDBG T ATSAMD21	Interface G18A 🔻 SWD	Device sig	gnature Read	Target Voltage
		The selected of	combination is not	applied.
				2
·Link (269200338) - Device Pro Tool Device	gramming Interface	Device signature	Target Voltage	U U
J-Link ATSAMD21G18A	▼ SWD ▼ Apply	0x10010305 Rea	ad 3.3 V Read	1 🙀 🔜
Interface settings	SWD Clock			
Tool information		-0		4 M
Device information	The clock frequency show	uld not exceed target CPU sp	eed * 10.	
Memories				Set
Fuses				
Security				

Select ATSAMD21G18A as the device, SWD as the interface and hit Apply

You can then Read the Device Signature. Make sure this all works before you continue!

If you are asked to update the J-Link or EDBG firmware, its OK to do so now.

Firmware Upgrade		×
EDBG firmware is up to dat	e	
	On Tool	On Disk
Firmware Version		3.0c
EDBG firmware successful	y upgraded	
	Upgra	de Close

Build & Start Debugging

OK close out the modal programming window, we dont need it for now. Build the program



Add a Break by clicking on the first DigitalWrite function call, you'll see a red dot



Now run Start Debugging and Break



Note that by uploading a debug sketch you will blow away the bootloader on your Arduino Zero or Feather M0, see the next section for re-loading it!

You'll get prompted to select a debugging tool

ArduinoSketch1 File Edit View O ▼ O Ni Ni Ni	AtmelStudio VAssistX ASF Project Build I 1 1 - 1 → 1 ■ ■ ■ X D A 2 N ↔ 1 3 1 → 1 Hex 3	Debug Tools Window Help - ♥ - ጫ ▷ M Debug - Debug Browser - ♥ - ↓ 때 때 때 때 및 ↓ ↓
Blink* ⊅ × Sketch.c Build Build Events Toolchain Device	pp ASF Wizard Configuration: N/A Selected debugger/programm	
Components Advanced	Debug settings	Please select a connected tool and interface and try again.

Go thru what you did before, selecting the programmer and processor

Build Build Events	Configuration: N/A Platform: N/A
Toolchain Device Tool	Selected debugger/programmer EDBG • 00000000AZE000001222 Interface: SWD
Components Advanced	
	SWD Clock 2 MHz The clock frequency should not exceed target CPU speed * 10. Programming settings Erase entire chip
	Debug settings Override Vector Table Offset Register exception_table Cache all flack memory except

Once done go back and re-run Start Debugging

You'll end up in a strange code, labeled **int main(void)** { this is the main entry point to the sketch. Normally this part is never seen, it's what sets up the Arduino before you get to the **setup** section of the sketch!



Select Continue to skip ahead to your stopping point



Now you'll end up at that DigitalWrite with the red dot. Note that you stop right before this gets run.

Now select **Step Over** to execute that line. Since you're in step-debugging mode you'll have to **Step** each function call you want to run. If you just want to continue running the code without any delays or steps, click on **Continue** like you did before



You can also dig deeper into a function with Step Into



This will let you go *into* the function call, to see what goes on inside. You can then continue to step over, step in or step out (complete the function)

ArduinoSketch1 (Debugging File Edit Ø • Ø Ø • Ø Ø • Ø Ø • Ø Ø • Ø Ø • Ø Ø Ø Ø • Ø	g)-AtmetStudio ASF Project Build Debug ■ ■ ■ 【 A D O ♡ - ♡ - † ⑦ 1 1 1 T Hex 136 S → 1	[cols Window Help Q > M Debug - Debug Browser - - Q @ @ @ @ Q ↓ & & (A 15)	Advanced Mode	ଏ ମାନଙ୍କୁ - ସମନଙ୍କୁ 1222) -
Blink Sketch.cpp			wiring_digital.c 🕷 🗙	▼ Solution Explorer ▼ ♥ ×
→ digitalWrite	🗘 🤿 void digitalWrite(uint32_t ulPi	n, uint32_t ulVal)	- (*G	000 0.000
}				Search Solution Explorer 1 P -
<pre>void digital&rite(ui { // Handle the case if (g_APinOescript { return ; } // Enable pull-up r PORT->Group[g_APinD switch (ulval) { case LON: PORT->Group[g_A break; default: pORT->Group[g_A break; } return ; } @ int digitalRead(uint</pre>	<pre>ntl2_t ulPin, uintl2_t ulVal ; the pin isn't usable as PIO ion[ulPin].ulPinType == PIO_M existor escription[ulPin].ulPort].PIM PinDescription[ulPin].ulPort]. PinDescription[ulPin].ulPort]. 32_t ulPin)</pre>) JT_A_PIN) IFG[g_APinDescription[ulPin].ulPin].reg=(uin OUTCLR.reg = (lul << g_APinDescription[ulPi OUTSET.reg = (lul << g_APinDescription[ulPi	t8_t)(PORT_PINCFG_PULLEN) ; n].ulPin) ; n].ulPin) ;	D pulse, son A Reset.cpp Reset.cpp Starbuck Starb
200 % + 4			Þ	Solution Explorer Properties
Autos		- Q ;	Memory 4	- # ×
Name	Value	Туре	Memory: base FLASH	-
PIO_NOT_A_PIN	PIO_NOT_A_PIN	enum_EPioType(constant.)	0x00000000 00 80 00 20 59 01 00 00	41 01 00 .€. YA ▲
♥ uPin	13	uint32_t/multi-location: Range 0x10c0-0x10c8: -	0:000000000 00 41 01 00	00 00 00 .A 00 00 00 00 00 00 A 00 00 00 A 00 01 01 00 41 01 00 00 41 A A 41 01 00 A •
Autos Locals Watch 1 Watch 2			Call Stack Breakpoints Command Wi Imr	nediate Wi Output Memory 4
Stopped				

You can also see variable names below, and the entirety of memory. Since this is just a basic tutorial we wont go into

the vast depths of debugging, stack traces, and memory twiddling!

There's a ton more details on the Atmel Studio documentation page

Paths and Optimizations

C and C++ compilers make your code better when they compile it! This is great, but when we are trying to debug our code we don't want anything to change it.

If you try to use the debugger and you see that it doesn't move from one line to the next as you would expect, this is because you have compiler optimizations turned on.

to turn them off, right click on the ArduinoCore project in the Solution Explorer pane, and click properties.



Then under Toolchain, go to the ARM/GNU C Compiler heading and click Optimization. Set Optimization Level

to None (-OO).



Then do the same thing under the ARM/GNU C++ Compiler heading.

Core + ×	rariant.h Adafruit_AMG88xch S	Retch.cpp
Events	Configuration: Active (Debug)	✓ Platform: Active (ARM) ✓
hain	Configuration Manager	
ce iponents inced	ABM/GNU Common General Conputifie Conneral Constance Constanc	ASM/GNU C+- Compiler

Then save your project.

Now, repeat the above steps to turn off compiler optimizations for the other project (whatever you have named your sketch) in the solution explorer.

There should be two projects in the Solution Explorer pane. ArduinoCore, and whatever you have named your sketch. Make sure you have done the above steps to turn off compiler optimizations on both projects in the solution explorer pane.

Correcting Paths to Necessary Files

Current versions of Arduino have changed the location of the CMSIS core files that are necessary to compile projects.

We can fix these paths by going back to the **Properties** pane (by right clicking on the project in **Solution Explorer** and selecting **Properties** as we did before) and under **ARM/GNU C Compiler** select **Directories** and add the new path to the CMSIS core files to the Include Paths section.

This can be done by clicking the green plus button, and then finding the folder by clicking the ... button in the window that pops up.

Leave the **Relative Path** box checked.

The current location of the CMSIS core as of the writing of this guide is:

 $\label{eq:linear} C: Users YourNameHere \ AppData \ Local \ Arduino 15 \ ackages \ arduino \ CMSIS-Atmel \ 1.1.0 \ CMSIS \ Device \ ATMEL$

uild	Configuration: Active (Debug)	✓ Platform: Active (ARM) ✓
uild Events solchain	Configuration Manager	
evice		
loc	ARM/GNU Common	ARM/GNU C Compiler => Directories
omponents	CutputFiles	☑ Include Device Support Header Path (-I)
dvanced	ARM/GNU C Compiler	Include Paths (-I)
	Preprocessor	\$(PackRepoDir)\arm\CMSIS\4.2.0\cMSIS\Include\
	Directories	./././././AppData/Local/Arduino15/packages/arduino/tools/CMSIS-Atmel/1.1.0/CMSIS/Device/ATMEL
	C Cpomization	_\include\libraries\wire
	2 Debugging	_\include\libraries\wire\utility
	Miscellaneous	_\include\\ibranes\adafruit_amg88ix
	ARM/GNU C++ Compiler	
	Preprocessor	< >
	Symbols	
	2 Optimization	Add Include Paths (-I) X
	2 Debugging	Include Paths (-I)
	Warnings Miscellaneous	
	 ARM/GNU Linker 	
	General	Kelative Path
	Cloranes	OK Cancel
	Memory Settings	
	Miscellaneous ARM/GNU Assembler	
	🖃 General	
	Debugging	
	General	
	Symbols	
	ARM/GNU Archiver	
	General	
	< >	

Then select the path you just added in the list and click the yellow up arrow icon to move it to the top of the list.



Now repeat those same steps in the Directories pane under the ARM/GNU C++ Compiler section.

inoCore + ×	variant.h Adafruit_AMG88xx.h	Sketch.cpp	HardwareSerial.h 🐚
uild uild Events	Configuration: Active (Debug)	V Platform: Active (ARM) V	
olchain	Configuration Manager		
ol	ARM/GNU Common	ARM/GNU C++ Compiler => Directories	
mponents	CutputFiles	☑ Include Default Include Path (-I)	
lvanced	ARM/GNU C Compiler	Include Paths (-I)	🔊 🔊 🐑 🖓
	Preprocessor Symbols		ISIS/4.5.0/CMSIS/Include ISIS-Atmel/1.1.0/CMSIS/Device/ATMEL
Unectores Unectores Optimization Debugging Warnings Miscellaneous Miscellaneous General Preprocessor Debugging Warnings Miscellaneous Miscellaneous Miscellaneous Miscellaneous Miscellaneous		× *	
	General Libraries Optimization Miscellaneous General		

Do these steps for both projects in the **Solution Explorer** pane.

Make sure you have done the steps under the "Correcting Paths To Necessary Files" heading for both projects in the Solution Explorer pane.

Fixing Some Core Files

If you try to debug your sketch now, it may warn of an "undefined referenced to `vtable for HardwareSerial'"

To fix this, open the includes/core/HardwareSerial.h file under the ArduinoCore project.

Scroll down to the class definition around line 67 and replace the class declaration with the following code:

```
class HardwareSerial : public Stream
{
  public:
HardwareSerial() {};
 virtual ~HardwareSerial() {};
    virtual void begin(unsigned long) {};
    virtual void begin(unsigned long baudrate, uint16_t config) {};
    virtual void end() {};
    virtual int available(void) = 0;
    virtual int peek(void) = 0;
    virtual int read(void) = 0;
    virtual void flush(void) = 0;
    virtual size_t write(uint8_t) = 0;
    using Print::write; // pull in write(str) and write(buf, size) from Print
    virtual operator bool() = 0;
};
```

Your file should look like this:



Once this is done, you should be able to compile and debug your sketch!

Restoring Bootloader

When you program in for debugging you are writing direct to the chip, this deletes the bootloader! You'll want to restore it if you ever want to go back to using the Arduino IDE.

Arduino Zero

This is pretty easy. Launch the IDE, select Arduino Zero (programming port) from the Tools->Board menu, and Atmel EDBG as the Tools->Programmer

Then select Burn Bootloader

💿 sketch_feb25a Arc	Juino 1.6.7	ing direct to	
File Edit Sketch Too	ls Help		
	Auto Format	Ctrl+T	
sketch_feb25a	Archive Sketch Fix Encoding & Reload		
<pre>void setup()</pre>	Serial Monitor	Ctrl+Shift+M	L
// put your	Serial Plotter	Ctrl+Shift+L	
}	Board: "Arduino/Genuino Zero (Programming Port)"	•	
<pre>void loop() {</pre>	Port	•	
// put your	Programmer: "Atmel EDBG"	×	
}	Burn Bootloader		

It only takes a few seconds to burn in the bootloader:



Feather M0 or Others

For this, you'll need to use the Atmel Studio setup, since you're using a J-Link.

Download the bootloader hex file



Wire it up correctly and select Device Programming



Select J-Link and the ATSMD21G18A with SWD. Verify you can read the Device Signature

Tool Device J-Link ATSAMD21	G18A • SWD • Apply	Device signature 0x10010305 Read	Target Voltage 3.3 V Read	
Interface settings Tool information Device information	SWD Clock	d not exceed target CPU spee	d * 10.	4 MHz
Memories Fuses				Set

Unlock the Bootloader protection by going to Fuses and changing BOOTPROT to 0x07 then programming

Fuse	Name	Value	^
TEMP_LOG_WORD_1	.ROOM_ADC_VAL	0x0B73	
TEMP_LOG_WORD_1	.HOT_ADC_VAL	0x0D6A	
USER_WORD_0.NVM	ICTRL_BOOTPROT	0x07	_
USER_WORD_0.NVN	ICTRL_EEPROM_SIZE	0x07	=
USER_WORD_0.BOD	33USERLEVEL	0x07	
USER_WORD_0.BOD	33_EN		
USER WORD 0.BOD	33 ACTION	0.01	*
Fuse Register	Value		
OTP4_WORD_0	0x40004007		
OTP4_WORD_1	0x79F4AE1C		
OTP4_WORD_2	0xFFFFFE00		
TEMP_LOG_WORD_0	0xF905501E		
TEMP_LOG_WORD_1	0xD6AB73F4		
USER_WORD_0	0xD8E0C7FF		
USER_WORD_1	0xFFFFC5D		
		Copy to clipboard	ר
Auto read			
📝 Verify after program	ming	Program Verify Read	

Next click on Memories in the left hand side

ool Device	Interface	Device signature 0x10010305 Rea	Target Voltage d 3.3 V Read	2
Interface settings Tool information	Device Erase Chip Erase now	v		
Memories	C:\Users\ladyada\Desktop\	featherm0bootloader_1511	.01.hex	-
Fuses Security	Erase Flash before progra Advanced	amming	Program Verify	Read
	User Page (256 bytes)			•
	Erase User Page before p Advanced	programming	Program Verify	Read
ading device IDOK				
OK				

Next to the Flash (256 KB) section, click the triple-dots and select the bootloader file.

Then click **Program** to program it in

Tool Device	Interface	Device signature	Target Voltage	
J-Link	G18A - SWD - Apply	0x10010305 Read	d 3.3 V Read	🗱 🔝
Interface settings Tool information	Device Erase Chip Erase no	w		
Device information	Flash (256 KB)			
Memories	C:\Users\ladyada\Desktop	\featherm0bootloader_1511	01.hex	•
Fuses Security	Erase Flash before prog Advanced	ramming	Program	Verify Read
	User Page (256 bytes)			•
	Erase User Page before Advanced	programming	Program	Verify Read
rasing device OK rogramming FlashOK erifying FlashOK				
Verifying FlashO	К			
				Close