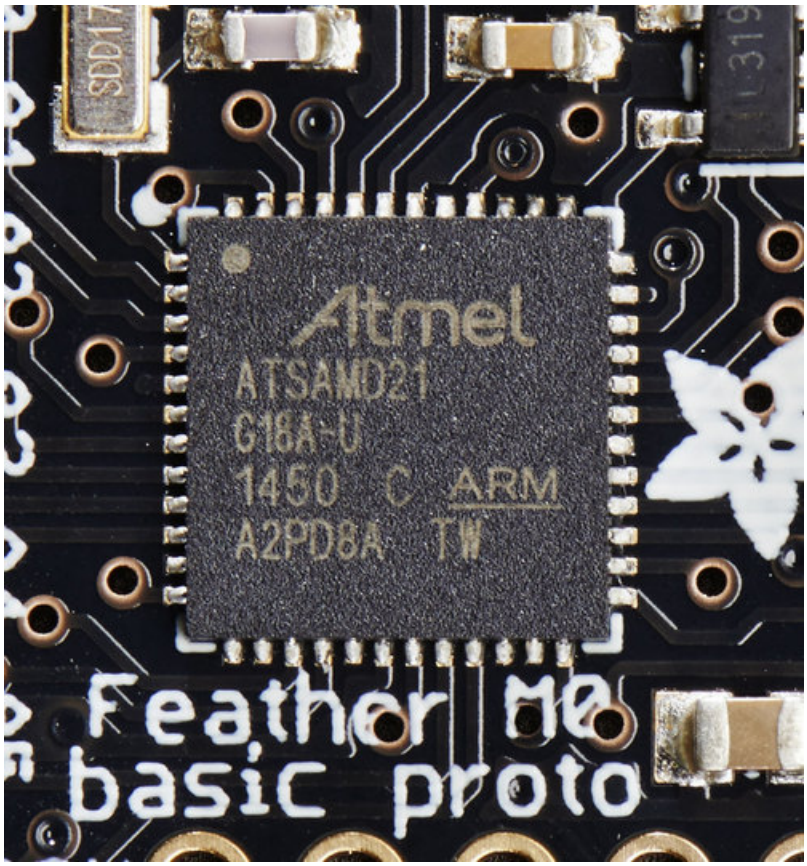
adafruit learning system

Using ATSAMD21 SERCOM for more SPI, I2C and Serial ports

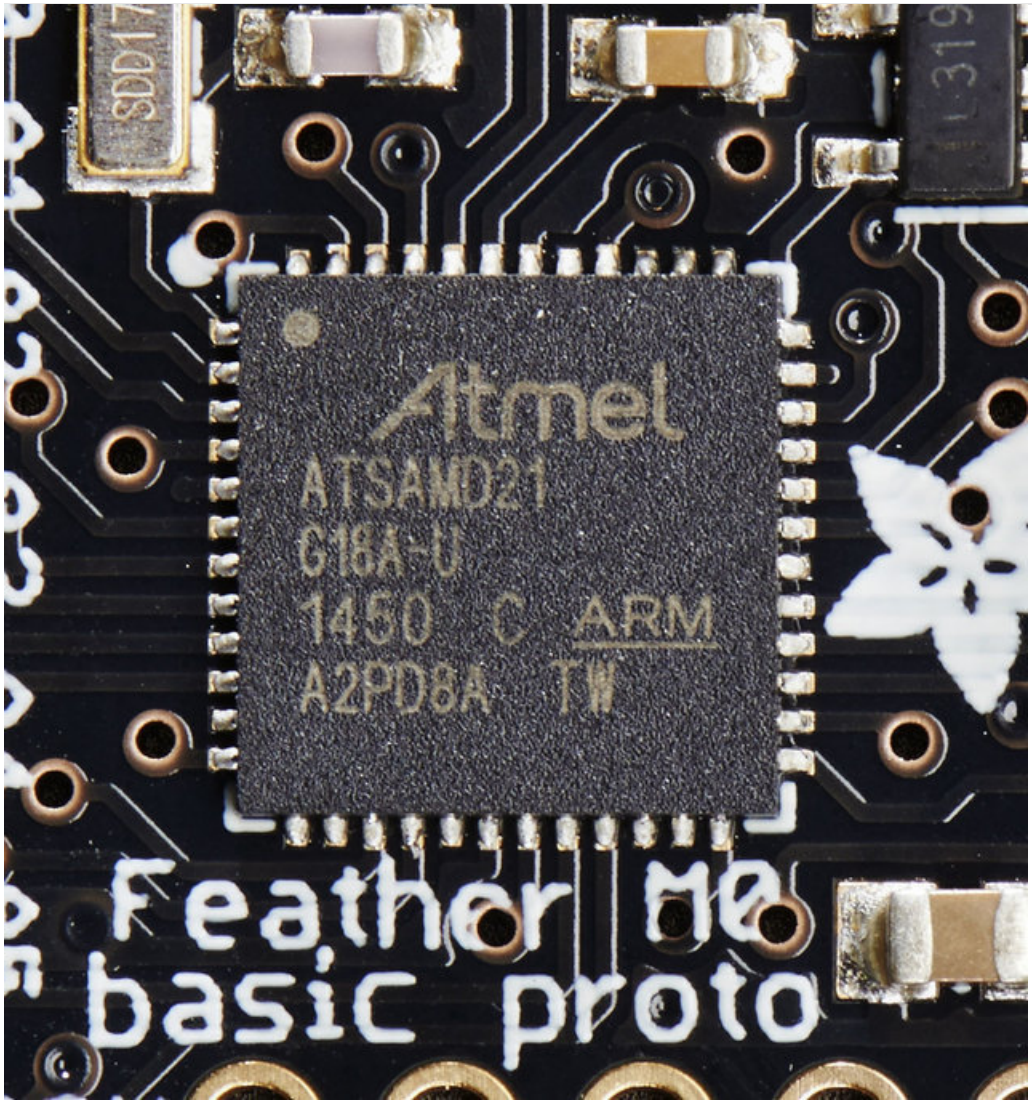
Created by lady ada



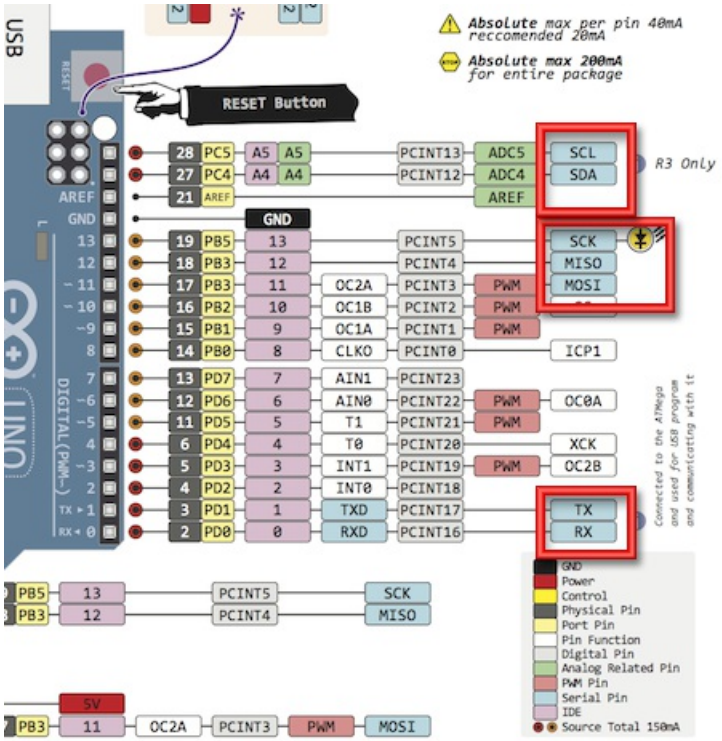
Last updated on 2018-08-22 03:52:08 PM UTC

Guide Contents

Overview



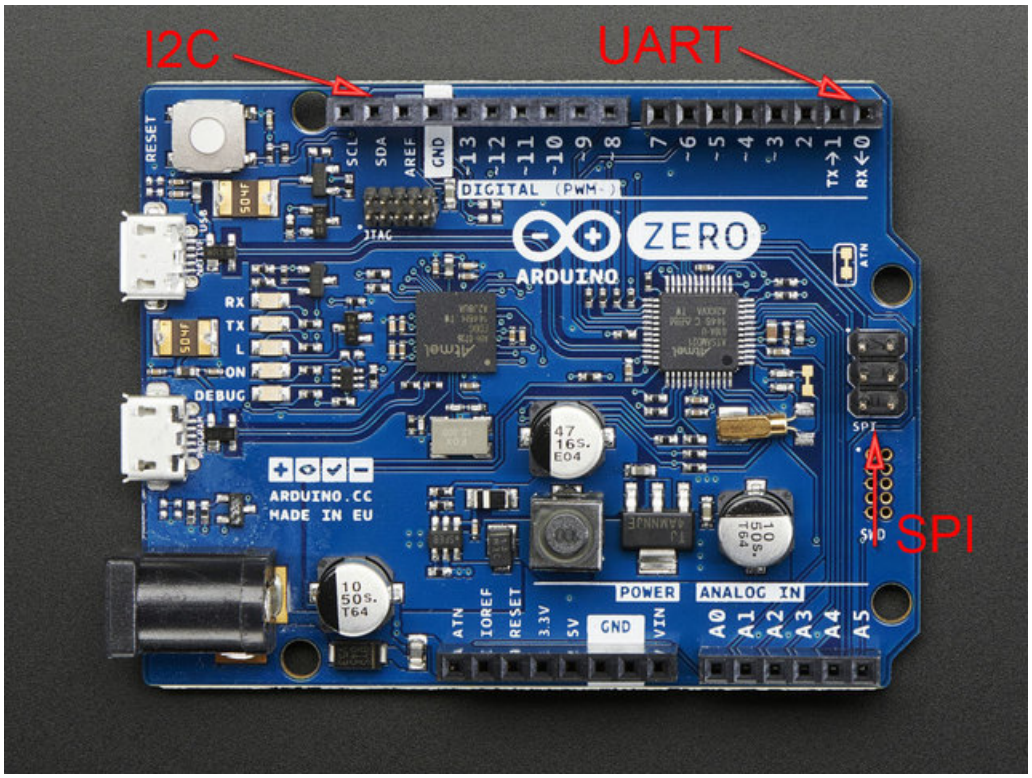
If you've been using AVR's for a bit, particularly the Arduino UNO's ATmega328, you have likely bumped into the limits of having a single SPI, Wire (i2c) and UART (Serial) port. You can update to the Arduino MEGA, that'll give you some more UARTs, but you're still stuck with a single SPI and I2C port.



When we first checked out the ATSAMD21 chip (the processor used in the Arduino Zero and Adafruit Feather M0), we were very happy to see that the chip has 6 "SERCOM"s, a.k.a SERial COMmunication modules. Each one of these modules can be used for I2C, SPI or Serial. That means you can have 3 UART's & 3 SPI's, or maybe 2 SPI's and 4 I2C's. Basically, you have a ton of options for adding more hardware support for the most common 3 chip-to-chip protocols.

- Up to six Serial Communication Interfaces (SERCOM), each configurable to operate as either:
  - USART with full-duplex and single-wire half-duplex configuration
  - I^2C up to 3.4MHz
  - SPI
  - LIN slave

Now, the default Arduino IDE setup for the Zero/ATSAMD21 still only comes with one UART, one SPI and one I2C device, in the 'classic' locations



...but that still leaves you with 3 full SERCOM's to do whatever you like!

Muxing it up

So, wow, 6 SERCOMs, totally configurable - best thing ever! So what's the catch?

It's just a little one really - when setting up the SERCOM you have to configure what pins it will use (you have many options) and it takes a little time to look up the exact multiplexing setup and make sure you have it right. Not a huge deal, but has to be done right.

Multiplex Table

If you look in the ATSAMD21 datasheet (https://adafru.it/l3e), there's a big table in section 6 called **Multiplexing and Considerations**, and it says...

> Each pin is by default controlled by the PORT as a general purpose I/O and alternatively it can be assigned to one of the peripheral functions A, B, C, D, E, F, G or H. To enable a peripheral function on a pin, the Peripheral Multiplexer Enable bit in the Pin Configuration register corresponding to that pin (PINCFGn.PMUXEN, n = 0-31) in the PORT must be written to one.
> The selection of peripheral function A to H is done by writing to the Peripheral Multiplexing Odd and Even bits in the Peripheral Multiplexing register (PMUXn.PMUXE/O) in the PORT

Which basically means - each pin has different capabilities, such as digital I/O, analog input, timer/pwm output, and sercom connectivity. On most chips, there's only *one* pin that can act as the I2C clock pin, but on this chip, there's quite a few pins that can be selected. So the trick is figuring out which pin you want to have that capability and 'MUXing it'

Here's how to read the table, here we've got the first page:

Pin[1]						A	B[2][3]					C	D	E	F	G
SAMD21E	SAMD21G	SAMD21J	I/O Pin	Supply	Type	EIC	REF	ADC	AC	PTC	DAC	SERCOM[3][4]	SERCOM-ALT	TC[4]/TCC	TCC	COM
1	1	1	PA00	VDDANA		EXTINT[0]							SERCOM1/PAD[0]	TCC2/WO[0]		
2	2	2	PA01	VDDANA		EXTINT[1]							SERCOM1/PAD[1]	TCC2/WO[1]		
3	3	3	PA02	VDDANA		EXTINT[2]		AIN[0]		Y[0]	VOUT					
4	4	4	PA03	VDDANA		EXTINT[3]	ADC/VREFA DAC/VREFA	AIN[1]		Y[1]						
		5	PB04	VDDANA		EXTINT[4]		AIN[12]		Y[10]						
		6	PB05	VDDANA		EXTINT[5]		AIN[13]		Y[11]						
		9	PB06	VDDANA		EXTINT[6]		AIN[14]		Y[12]						
		10	PB07	VDDANA		EXTINT[7]		AIN[15]		Y[13]						
	7	11	PB08	VDDANA		EXTINT[8]		AIN[2]		Y[14]			SERCOM4/PAD[0]	TC4/WO[0]		
	8	12	PB09	VDDANA		EXTINT[9]		AIN[3]		Y[15]			SERCOM4/PAD[1]	TC4/WO[1]		
5	9	13	PA04	VDDANA		EXTINT[4]	ADC/VREFB	AIN[4]	AIN[0]	Y[2]			SERCOM0/PAD[0]	TCC0/WO[0]		
6	10	14	PA05	VDDANA		EXTINT[5]		AIN[5]	AIN[1]	Y[3]			SERCOM0/PAD[1]	TCC0/WO[1]		
7	11	15	PA06	VDDANA		EXTINT[6]		AIN[6]	AIN[2]	Y[4]			SERCOM0/PAD[2]	TCC1/WO[0]		
8	12	16	PA07	VDDANA		EXTINT[7]		AIN[7]	AIN[3]	Y[5]			SERCOM0/PAD[3]	TCC1/WO[1]		I2S/SD[
11	13	17	PA08	VDDIO	I²C	NMI		AIN[16]		X[0]		SERCOM0/PAD[0]	SERCOM2/PAD[0]	TCC0/WO[0]	TCC1/WO[2]	I2S/SD[
12	14	18	PA09	VDDIO	I²C	EXTINT[9]		AIN[17]		X[1]		SERCOM0/PAD[1]	SERCOM2/PAD[1]	TCC0/WO[1]	TCC1/WO[3]	I2S/MCK[0
13	15	19	PA10	VDDIO		EXTINT[10]		AIN[18]		X[2]		SERCOM0/PAD[2]	SERCOM2/PAD[2]	TCC1/WO[0]	TCC0/WO[2]	I2S/SCK[0
14	16	20	PA11	VDDIO		EXTINT[11]		AIN[19]		X[3]		SERCOM0/PAD[3]	SERCOM2/PAD[3]	TCC1/WO[1]	TCC0/WO[3]	I2S/FS[

The chip we're using is the **ATSAMD21G** note that the **J** series does have more pins so for example, pins 5-10 on the ATSAMD21J (PB04 thru PB07) don't apply to our chips.

In columns C and D, you can see the SERCOM pads available. I've also circled in blue the matching I/O Pins.

Each SERCOM has 4 possible pads: **SERCOM#/PAD[x]** where # can be 0 thru 5 (6 total SERCOMs), and x can be 0 thru

3 (4 total pads).

Some pins can only act as a single SERCOM's pads - for example **PA00** is the first pin in the table and can only act as **SERCOM1.PAD[0]**

On the other hand, some pins can do dual duty: **PA08** lower on the list can act as *either* SERCOM #0's PAD 0 or it can act as SERCOM #2's PAD 0.

Arduino's MUX Table

To map the raw pin name to the Arduino Zero 'board pin' you can read the table in **variants.cpp (https://adafru.it/l3f)** (in the board package)

```
/*
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * + Pin number +  ZERO Board pin  |  PIN   | Label/Name      | Comments (* is for default peripheral in
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * |            | Digital Low      |        |                 |
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * | 0          | 0 -> RX          |  PA11  |                 | EIC/EXTINT[11] ADC/AIN[19]           PTC/
 * | 1          | 1 <- TX          |  PA10  |                 | EIC/EXTINT[10] ADC/AIN[18]           PTC/
 * | 2          | 2                |  PA14  |                 | EIC/EXTINT[14]
 * | 3          | ~3               |  PA09  |                 | EIC/EXTINT[9]  ADC/AIN[17]           PTC/
 * | 4          | ~4               |  PA08  |                 | EIC/NMI        ADC/AIN[16]           PTC/
 * | 5          | ~5               |  PA15  |                 | EIC/EXTINT[15]
 * | 6          | ~6               |  PA20  |                 | EIC/EXTINT[4]                        PTC/
 * | 7          | 7                |  PA21  |                 | EIC/EXTINT[5]                        PTC/
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * |            | Digital High     |        |                 |
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * | 8          | ~8               |  PA06  |                 | EIC/EXTINT[6]  ADC/AIN[6]  AC/AIN[2] PTC/
 * | 9          | ~9               |  PA07  |                 | EIC/EXTINT[7]  ADC/AIN[7]  AC/AIN[3] PTC/
 * | 10         | ~10              |  PA18  |                 | EIC/EXTINT[2]                        PTC/
 * | 11         | ~11              |  PA16  |                 | EIC/EXTINT[0]                        PTC/
 * | 12         | ~12              |  PA19  |                 | EIC/EXTINT[3]                        PTC/
 * | 13         | ~13              |  PA17  | LED             | EIC/EXTINT[1]                        PTC/
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * |            | Analog Connector |        |                 |
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * | 14         | A0               |  PA02  | A0              | EIC/EXTINT[2] *ADC/AIN[0]  DAC/VOUT  PTC/
 * | 15         | A1               |  PB08  | A1              | EIC/EXTINT[8] *ADC/AIN[2]            PTC/
 * | 16         | A2               |  PB09  | A2              | EIC/EXTINT[9] *ADC/AIN[3]            PTC/
 * | 17         | A3               |  PA04  | A3              | EIC/EXTINT[4] *ADC/AIN[4]  AC/AIN[0] PTC/
 * | 18         | A4               |  PA05  | A4              | EIC/EXTINT[5] *ADC/AIN[5]  AC/AIN[1] PTC/
 * | 19         | A5               |  PB02  | A5              | EIC/EXTINT[2] *ADC/AIN[10]           PTC/
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * |            | Wire             |        |                 |
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * | 20         | SDA              |  PA22  | SDA             | EIC/EXTINT[6]                        PTC/
 * | 21         | SCL              |  PA23  | SCL             | EIC/EXTINT[7]                        PTC/
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * |            |SPI (Legacy ICSP) |        |                 |
 * +------------+------------------+--------+-----------------+-----------------------------------------
 * | 22         | 1                |  PA12  | MISO            | EIC/EXTINT[12] SERCOM2/PAD[0] *SERCOM4/PA
 * |            | 2                |        | 5V0             |
 * | 23         | 4                |  PB10  | MOSI            | EIC/EXTINT[10]                *SERCOM4/PA
 * | 24         | 3                |  PB11  | SCK             | EIC/EXTINT[11]                *SERCOM4/PA
 * |            | 5                |        | RESET           |
```

```
* |            | 6                |        | GND             |
* +------------+------------------+--------+-----------------+-----------------------------------------
* |            | EDBG             |        |                 |
* +------------+------------------+--------+-----------------+-----------------------------------------
* | 30         |                  |  PB22  | EDBG_UART TX    | *SERCOM5/PAD[2]
* | 31         |                  |  PB23  | EDBG_UART RX    | *SERCOM5/PAD[3]
* +------------+------------------+--------+-----------------+-----------------------------------------
* | 32         |                  |  PA22  | EDBG_SDA        | Pin 20 (SDA)
* | 33         |                  |  PA23  | EDBG_SCL        | Pin 21 (SCL)
* +------------+------------------+--------+-----------------+-----------------------------------------
* | 34         |                  |  PA19  | EDBG_MISO       | EIC/EXTINT[3] *SERCOM1/PAD[3] SERCOM3/PAD
* | 35         |                  |  PA16  | EDBG_MOSI       | EIC/EXTINT[0] *SERCOM1/PAD[0] SERCOM3/PAD
* | 36         |                  |  PA18  | EDBG_SS         | EIC/EXTINT[2] *SERCOM1/PAD[2] SERCOM3/PAD
* | 37         |                  |  PA17  | EDBG_SCK        | EIC/EXTINT[1] *SERCOM1/PAD[1] SERCOM3/PAD
* +------------+------------------+--------+-----------------+-----------------------------------------
* | 38         | ATN              |  PA13  | EDBG_GPIO0      | EIC/EXTINT[13] SERCOM2/PAD[1] SERCOM4/PAD
* | 39         |                  |  PA21  | EDBG_GPIO1      | Pin 7
* | 40         |                  |  PA06  | EDBG_GPIO2      | Pin 8
* | 41         |                  |  PA07  | EDBG_GPIO3      | Pin 9
* +------------+------------------+--------+-----------------+-----------------------------------------
* |            |32.768KHz Crystal |        |                 |
* +------------+------------------+--------+-----------------+-----------------------------------------
* |            |                  |  PA00  | XIN32           | EIC/EXTINT[0] SERCOM1/PAD[0] TCC2/WO[0]
* |            |                  |  PA01  | XOUT32          | EIC/EXTINT[1] SERCOM1/PAD[1] TCC2/WO[1]
* +------------+------------------+--------+-----------------+-----------------------------------------
*/
```

To save me from having to constantly look it up, here's a table of all the ATSAMD21G's SERCOM MUXs:

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA00 Used by xtal   SERCOM1.0
PA01 Used by xtal   SERCOM1.1
PB08 A1    SERCOM4.0
PB09 A2    SERCOM4.1
PA04 A3    SERCOM0.0
PA05 A4    SERCOM0.1
PA06 D8    SERCOM0.2
PA07 D9    SERCOM0.3
PA08 D4  SERCOM0.0 SERCOM2.0
PA09 D3  SERCOM0.1 SERCOM2.1
PA10 D1  SERCOM0.2 SERCOM2.2
PA11 D0  SERCOM0.3 SERCOM2.3
PB10 D23 / MOSI   SERCOM4.2
PB11 D24 / SCK   SERCOM4.3
PA12 D22 / MISO SERCOM2.0 SERCOM4.0
PA13 Used by EDBC SERCOM2.1 SERCOM4.1
PA14 D2  SERCOM2.2 SERCOM4.2
PA15 D5  SERCOM2.3 SERCOM4.3
PA16 D11  SERCOM1.0 SERCOM3.0
PA17 D13  SERCOM1.1 SERCOM3.1
PA18 D10  SERCOM1.2 SERCOM3.2
PA19 D12  SERCOM1.3 SERCOM3.3
PA20 D6  SERCOM5.2 SERCOM3.2
PA21 D7  SERCOM5.3 SERCOM3.3
PA22 D20 / SDA SERCOM3.0 SERCOM5.0
PA23 D21 / SCL SERCOM3.1 SERCOM5.1
PA24 Used by USB SERCOM3.2 SERCOM5.2
PA25 Used by USB SERCOM3.3 SERCOM5.3
PB22 D30 / EDBG TX   SERCOM5.2
PB23 D31 / EDBG RX   SERCOM5.3
PA30 Used by SWCLK   SERCOM1.2
PA31 Used by SWDIO   SERCOM1.3
PB02 A5    SERCOM5.0
PB03 D25 / RX LED   SERCOM5.1
```

Note that a bunch of these pins are simply *not available* if you've got a Zero or Feather M0, because they are used for programming or LEDs, or not brought out to headers in some other way, so lets remove them & sort by pin

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA11 D0  SERCOM0.3 SERCOM2.3
PA10 D1  SERCOM0.2 SERCOM2.2
PA14 D2  SERCOM2.2 SERCOM4.2
PA09 D3  SERCOM0.1 SERCOM2.1
PA08 D4  SERCOM0.0 SERCOM2.0
PA15 D5  SERCOM2.3 SERCOM4.3
PA20 D6  SERCOM5.2 SERCOM3.2
PA21 D7  SERCOM5.3 SERCOM3.3
PA06 D8    SERCOM0.2
PA07 D9    SERCOM0.3
PA18 D10  SERCOM1.2 SERCOM3.2
PA16 D11  SERCOM1.0 SERCOM3.0
PA19 D12  SERCOM1.3 SERCOM3.3
PA17 D13  SERCOM1.1 SERCOM3.1
PB08 A1    SERCOM4.0
PB09 A2    SERCOM4.1
PA04 A3    SERCOM0.0
PA05 A4    SERCOM0.1
PB02 A5    SERCOM5.0
PA22 D20 / SDA SERCOM3.0 SERCOM5.0
PA23 D21 / SCL SERCOM3.1 SERCOM5.1
PA12 D22 / MISO SERCOM2.0 SERCOM4.0
PB10 D23 / MOSI   SERCOM4.2
PB11 D24 / SCK   SERCOM4.3
```

Predefined SERCOMs

OK so first things we want to sort out is what SERCOMs are already taken, this tutorial will assume you won't 'bash' existing SERCOMs (altho you are free to, if you want to give up the hardware serial UART for I2C, say).

The debug hardware serial port

which is used when connecting to the **Programming/Debug Port** on an Arduino Zero is on SERCOM 5, pads 2 & 3. It's not used on an Adafruit Feather because there's no debug port

```
PB22 D30 / EDBG TX   SERCOM5.2
PB23 D31 / EDBG RX   SERCOM5.3
```

I2C (SDA & SCL)

use SERCOM 3 (pads 0 & 1)

```
PA22 D20 / SDA SERCOM3.0 SERCOM5.0
PA23 D21 / SCL SERCOM3.1 SERCOM5.1
```

The SPI port

is on SERCOM 4, and uses pads 0, 2, and 3

```
PA12 D22 / MISO SERCOM2.0 SERCOM4.0
PB10 D23 / MOSI   SERCOM4.2
PB11 D24 / SCK   SERCOM4.3
```

And finally, the

hardware Serial interface (e.g. Serial1)

is on SERCOM 0

```
PA11 D0  SERCOM0.3 SERCOM2.3
PA10 D1  SERCOM0.2 SERCOM2.2
```

That leaves you with:

SERCOMs **1** & **2** on an Arduino Zero

and

SERCOMs **1**, **2**, and **5** on a Feather M0

Available SERCOM & Pins

OK we're nearly done paring down that table to the SERCOMs that we can use *and* pins that are available.

SERCOM 1

Available on both Zero's and Feathers, you can use pins 10, 11, 12 and 13 for SERCOM 1

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
----------------------------------------
PA18 D10  SERCOM1.2 SERCOM3.2
PA16 D11  SERCOM1.0 SERCOM3.0
PA19 D12  SERCOM1.3 SERCOM3.3
PA17 D13  SERCOM1.1 SERCOM3.1
```

SERCOM 2

This one has more options

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
----------------------------------------
PA11 D0  SERCOM0.3 SERCOM2.3
PA10 D1  SERCOM0.2 SERCOM2.2
PA14 D2  SERCOM2.2 SERCOM4.2
PA09 D3  SERCOM0.1 SERCOM2.1
PA08 D4  SERCOM0.0 SERCOM2.0
PA15 D5  SERCOM2.3 SERCOM4.3
PA12 D22 / MISO SERCOM2.0 SERCOM4.0
```

but some are already used, so while you could theoretically use D0 & D1 those are already used for the Hardware

Serial UART, and D22/MISO is used for SPI. So that leaves D2 thru D5

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA14 D2  SERCOM2.2 SERCOM4.2
PA09 D3  SERCOM0.1 SERCOM2.1
PA08 D4  SERCOM0.0 SERCOM2.0
PA15 D5  SERCOM2.3 SERCOM4.3
```

SERCOM 5

This one is used for programming on the Zero with EDBG interface but you can use it on the Feather M0 since only native USB is used.

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA20 D6  SERCOM5.2 SERCOM3.2
PA21 D7  SERCOM5.3 SERCOM3.3
PB02 A5    SERCOM5.0
PA22 D20 / SDA SERCOM3.0 SERCOM5.0
PA23 D21 / SCL SERCOM3.1 SERCOM5.1
```

Two of those are used by SDA/SCL which, unless you want to reuse for a different SERCOM, you'll only have:

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA20 D6  SERCOM5.2 SERCOM3.2
PA21 D7  SERCOM5.3 SERCOM3.3
PB02 A5    SERCOM5.0
```

Note that you don't get access to SERCOM5.1!

Freeing up SERCOM5

If you'd like to free up SERCOM5 on a Zero, and you won't be using the programming port (because that's how Serial data is passed back and forth) you can comment out

```
Uart Serial( &sercom5, PIN_SERIAL_RX, PIN_SERIAL_TX, PAD_SERIAL_RX, PAD_SERIAL_TX ) ;
```

and

```
void SERCOM5_Handler()
{
  Serial.IrqHandler();
}
```

At the end of the variants file (https://adafru.it/l3f) (you'll need to dig for where this is downloaded after you've added SAMD support, and you'll also have to re-remove after each board manager update of that package.

Note that you'll definitely not be able to use Serial anymore, you can only use USBSerial. Like I said, its a bit annoying,

you may want to grab the Adafruit SAMD package, and 'pretend' like your Zero is a Feather M0, it'll work just fine, and then Serial will point to USBSerial. whee!

Creating a new SPI

SPI is a high speed, 3-wire protocol that can be clocked at up to 12MHz on the ATSAMD21 and can also take advantage of DMA (that's for another tutorial)

Each SPI SERCOM has 3 pins - Master Clock line (SCK), Master Out/Slave In (MOSI) and Master In/Slave Out (MISO). In these examples we'll assume the ATSAMD21 is acting as the Master, since that's the by-far-most-common example. In this way the chip can control wifi modules, SD cards, DotStar LEDs, various sensors and actuators, etc.

So you might be wondering "Why does she keep bringing up that SPI uses only 3 pins?" It's cuz SERCOM's have 4 pads, but we only need three. There is one unused one that can be used as a normal GPIO (or whatever) pin

How SPI is Created Now

Luckily, Atmel & Arduino did a really great job with structuring SERCOMs so you can create and assign new ones. You can find the actual code that is used to create the "SPI" SERCOM in SPI.cpp (https://adafru.it/l3B)

```
SPIClass SPI (&PERIPH_SPI,  PIN_SPI_MISO,  PIN_SPI_SCK,  PIN_SPI_MOSI,  PAD_SPI_TX,  PAD_SPI_RX);
```

The macros are see in variants.h (https://adafru.it/l3C)

```
#define PIN_SPI_MISO         (22u)
#define PIN_SPI_MOSI         (23u)
#define PIN_SPI_SCK          (24u)
#define PERIPH_SPI           sercom4
#define PAD_SPI_TX           SPI_PAD_2_SCK_3
#define PAD_SPI_RX           SERCOM_RX_PAD_0
```

You can see here that **SPI** is of type **SPIClass** and if you expand the macros:

```
SPIClass SPI (&sercom4, 22, 24, 23, SPI_PAD_2_SCK_3, SERCOM_RX_PAD_0);
```

We can match this up with the pins we know are used for SPI:

```
PA12 D22 / MISO SERCOM2.0 SERCOM4.0
PB10 D23 / MOSI   SERCOM4.2
PB11 D24 / SCK   SERCOM4.3
```

You can see that the first argument (&sercom4) does in fact match our observation that SPI is on SERCOM4. The second, third and fourth arguments match with the MISO/SCK/MOSI pins.

The last two args are what actually define the pin muxing. We can see these definitions in SERCOM.h (https://adafru.it/l3D)

```
typedef enum
{
 SPI_PAD_0_SCK_1 = 0,
 SPI_PAD_2_SCK_3,
 SPI_PAD_3_SCK_1,
 SPI_PAD_0_SCK_3
} SercomSpiTXPad;
```

and

```
typedef enum
{
 SERCOM_RX_PAD_0 = 0,
 SERCOM_RX_PAD_1,
 SERCOM_RX_PAD_2,
 SERCOM_RX_PAD_3
} SercomRXPad;
```

The two together let us define the muxing. For example:

- **SPI_PAD_0_SCK_1** means MOSI is on SERCOMn.0 and SCK is on SERCOMn.1
- **SPI_PAD_2_SCK_3** means MOSI is on SERCOMn.2 and SCK is on SERCOMn.3
- **SPI_PAD_3_SCK_1** means MOSI is on SERCOMn.3 and SCK is on SERCOMn.1
- **SPI_PAD_0_SCK_3** means MOSI is on SERCOMn.0 and SCK is on SERCOMn.3

Then we can define which pin is for MISO

- **SERCOM_RX_PAD_0** means MISO on SERCOMn.0
- **SERCOM_RX_PAD_1** means MISO on SERCOMn.1
- **SERCOM_RX_PAD_2** means MISO on SERCOMn.2
- **SERCOM_RX_PAD_3** means MISO on SERCOMn.3

As you can tell, you can't quite have *every* possibility, for example you cannot have SCK on pad 0 or pad 2. And you cant have MOSI on pad 1. But you do have a lot of options!

OK so let's make a new SPI SERCOM already

ok ok so you are getting impatient! Lets try creating a new SERCOM and testing it out

Let's make a SPI device that 'mimics' the ATmega328 SPI pin numbering on Digital 11, 12, and 13:

- SCK on Digital 13
- MISO on Digital 12
- MOSI on Digital 11

Looking at our SERCOMs from the Mux table, we should go for **SERCOM1**

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA18 D10  SERCOM1.2 SERCOM3.2
PA16 D11  SERCOM1.0 SERCOM3.0
PA19 D12  SERCOM1.3 SERCOM3.3
PA17 D13  SERCOM1.1 SERCOM3.1
```

To put SCK signal on D13 (SERCOM1.1) and MOSI on D11 (SERCOM1.0) we want to use **SPI_PAD_0_SCK_1** for the TX mux. That leaves on MISO to do, on D12 (SERCOM1.3), which is **SERCOM_RX_PAD_3**

So something like this:

```
SPIClass SPI1 (&sercom1, 12, 13, 11, SPI_PAD_0_SCK_1, SERCOM_RX_PAD_3);
```

Lets test it out!

```
#include <SPI.h>

SPIClass mySPI (&sercom1, 12, 13, 11, SPI_PAD_0_SCK_1, SERCOM_RX_PAD_3);

void setup() {
  Serial.begin(115200);
  mySPI.begin();
}

uint8_t i=0;
void loop() {
  Serial.println(i);
  mySPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE0));
  mySPI.transfer(i++);
  mySPI.endTransaction();
}
```

OK ... that wont work. Why? Good question! Check out those pin definitions in the variants.cpp file:

```
const PinDescription g_APinDescription[]=
...
  { PORTA, 16, PIO_TIMER, (PIN_ATTR_DIGITAL|PIN_ATTR_PWM|PIN_ATTR_TIMER), No_ADC_Channel, PWM2_CH0, TCC2_
  { PORTA, 19, PIO_TIMER_ALT, (PIN_ATTR_DIGITAL|PIN_ATTR_PWM|PIN_ATTR_TIMER_ALT), No_ADC_Channel, PWM0_CH

  // 13 (LED)
  { PORTA, 17, PIO_PWM, (PIN_ATTR_DIGITAL|PIN_ATTR_PWM), No_ADC_Channel, PWM2_CH1, NOT_ON_TIMER, EXTERNAL
```

The first argument is the port (e.g. PORTA or PORTB), the second is the port's pin number, e.g. PORTA, 16 -> PA16. The third argument 'sets' the type of pinmux we're going to use.

In this case, these are set to be used as PIO_TIMER, PIO_TIMER_ALT and PIO_PWM

We have to tell the Arduino core to change the MUX type before we reassign the SERCOM to this pin

You can do that by calling **pinPeripheral(***pinnumber, function***)** which is an internal function but hey we're being hardk0re

```
#include <SPI.h>
#include "wiring_private.h" // pinPeripheral() function

SPIClass mySPI (&sercom1, 12, 13, 11, SPI_PAD_0_SCK_1, SERCOM_RX_PAD_3);

void setup() {
  Serial.begin(115200);

  // do this first, for Reasons
  mySPI.begin();

  // Assign pins 11, 12, 13 to SERCOM functionality
  pinPeripheral(11, PIO_SERCOM);
  pinPeripheral(12, PIO_SERCOM);
  pinPeripheral(13, PIO_SERCOM);
}

uint8_t i=0;
void loop() {
  Serial.println(i);
  mySPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE0));
  mySPI.transfer(i++);
  mySPI.endTransaction();
}
```

You can use your oscilloscope to see the data traces

> Say you only want to transmit SPI data, you can leave out the pinPeripheral(12, PIO_SERCOM) line, and you can keep using #12 however you like

We can also try SERCOM2:

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA14 D2  SERCOM2.2 SERCOM4.2
PA09 D3  SERCOM0.1 SERCOM2.1
PA08 D4  SERCOM0.0 SERCOM2.0
PA15 D5  SERCOM2.3 SERCOM4.3
```

Lets put **SCK** on D5 (SERCOM2.3), **MISO** on D3 (SERCOM2.1) and **MOSI** on D4 (SERCOM2.0)

```
#include <SPI.h>
#include "wiring_private.h" // pinPeripheral() function

SPIClass mySPI (&sercom2, 3, 5, 4, SPI_PAD_0_SCK_3, SERCOM_RX_PAD_1);

void setup() {
  Serial.begin(115200);

  // do this first, for Reasons
  mySPI.begin();

  // Assign pins 3, 4, 5 to SERCOM & SERCOM_ALT
  pinPeripheral(3, PIO_SERCOM_ALT);
  pinPeripheral(4, PIO_SERCOM_ALT);
  pinPeripheral(5, PIO_SERCOM);
}

uint8_t i=0;
void loop() {
  Serial.println(i);
  mySPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE0));
  mySPI.transfer(i++);
  mySPI.endTransaction();
}
```

Note that since pin 3 and 4 use a SERCOM alt mux, we have to pass in PIO_SERCOM_ALT rather than PIO_SERCOM when we call pinPeripheral()

Creating a new Serial

Compared to SPI, UART is even simpler - it's only 2 pins, RX & TX. UARTs are a real pain to emulate or bitbang due to their very precise timing and their asynchronous RX lines are very difficult to do if you dont have hardware interrupt pins available.

For that reason, being able to create new Serial's is *awesome*

How Serial is Created Now

Luckily, Atmel & Arduino did a really great job with structuring SERCOMs so you can create and assign new ones. You can find the actual code that is used to create the "Serial" SERCOM in variants.cpp (https://adafru.it/l3f)

```
      Uart Serial1( &sercom0, PIN_SERIAL1_RX, PIN_SERIAL1_TX, PAD_SERIAL1_RX, PAD_SERIAL1_TX ) ;
Uart Serial( &sercom5, PIN_SERIAL_RX, PIN_SERIAL_TX, PAD_SERIAL_RX, PAD_SERIAL_TX ) ;
void SERCOM0_Handler()
{
  Serial1.IrqHandler();
}

void SERCOM5_Handler()
{
  Serial.IrqHandler();
}
```

If you have a Feather, you wont see the SERCOM5_Hander or Serial(&sercom5...) lines

The first hardware Serial object (on D0 and D1) is created with:

```
Uart Serial1( &sercom0, PIN_SERIAL1_RX, PIN_SERIAL1_TX, PAD_SERIAL1_RX, PAD_SERIAL1_TX ) ;
```

The actual definitions of those macros is available in variants.h (https://adafru.it/l3C)

```
// Serial1
#define PIN_SERIAL1_RX       (0ul)
#define PIN_SERIAL1_TX       (1ul)
#define PAD_SERIAL1_TX       (UART_TX_PAD_2)
#define PAD_SERIAL1_RX       (SERCOM_RX_PAD_3)
```

You can see here that **Serial1** is of type **Uart** and if you expand the macros:

```
Uart Serial1( &sercom0, 0, 1, PAD_SERIAL1_RX, PAD_SERIAL1_TX ) ;
```

We can match this up with the pins we know are used for Serial1

```
PA11 D0  SERCOM0.3 SERCOM2.3
PA10 D1  SERCOM0.2 SERCOM2.2
```

You can see that the first argument (&sercom0) does in fact match our observation that Serial1 is on SERCOM0. The

second and third arguments match with the RX & TX pins.

The last two args are what actually define the pin muxing. We can see these definitions in SERCOM.h (https://adafru.it/l3D)

```
typedef enum
{
 SERCOM_RX_PAD_0 = 0,
 SERCOM_RX_PAD_1,
 SERCOM_RX_PAD_2,
 SERCOM_RX_PAD_3
} SercomRXPad;

typedef enum
{
 UART_TX_PAD_0 = 0x0ul, // Only for UART
 UART_TX_PAD_2 = 0x1ul,  // Only for UART
 UART_TX_RTS_CTS_PAD_0_2_3 = 0x2ul,  // Only for UART with TX on PAD0, RTS on PAD2 and CTS on PAD3
} SercomUartTXPad;
```

The two together let us define the muxing. For example we can define which pin is for RX

- **SERCOM_RX_PAD_0** means RX on SERCOMn.0
- **SERCOM_RX_PAD_1** means RX on SERCOMn.1
- **SERCOM_RX_PAD_2** means RX on SERCOMn.2
- **SERCOM_RX_PAD_3** means RX on SERCOMn.3

and then for TX:

- **UART_TX_PAD_0** means TX on SERCOMn.0
- **UART_TX_PAD_2** means TX on SERCOMn.2

As you can tell, you can't quite have *every* possibility, for example you cannot have TX on pad 1 or pad 3. But you do have a lot of options!

OK so let's make a new Serial SERCOM already

I understand, you want to make Serials! Lets try creating a new SERCOM and testing it out

Let's make a Serial device on SERCOM #1, lets look at what pin muxing options we've got:

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA18 D10  SERCOM1.2 SERCOM3.2
PA16 D11  SERCOM1.0 SERCOM3.0
PA19 D12  SERCOM1.3 SERCOM3.3
PA17 D13  SERCOM1.1 SERCOM3.1
```

We can put TX on D11 (SERCOM1.0) or D10 (SERCOM1.2), RX can be on any pin.

How about we have **D10** be **TX** and **D11** be **RX**?

That would mean something like:

```
Uart Serial2 (&sercom1, 11, 10, SERCOM_RX_PAD_0, UART_TX_PAD_2);
```

Lets test it out!

```
Uart Serial2 (&sercom1, 11, 10, SERCOM_RX_PAD_0, UART_TX_PAD_2);

void SERCOM1_Handler()
{
  Serial2.IrqHandler();
}

void setup() {
  Serial.begin(115200);

  Serial2.begin(115200);
}

uint8_t i=0;
void loop() {
  Serial.println(i);
  Serial2.write(i++);
  delay(10);
}
```

aaaand...nothing! Why? Good question! Check out those pin definitions in the variants.cpp file:

```
const PinDescription g_APinDescription[]=
...
 { PORTA, 18, PIO_TIMER, (PIN_ATTR_DIGITAL|PIN_ATTR_PWM|PIN_ATTR_TIMER), No_ADC_Channel, PWM3_CH0, TC3_CH
 { PORTA, 16, PIO_TIMER, (PIN_ATTR_DIGITAL|PIN_ATTR_PWM|PIN_ATTR_TIMER), No_ADC_Channel, PWM2_CH0, TCC2_C
```

The first argument is the port (e.g. PORTA or PORTB), the second is the port's pin number, e.g. PORTA, 18 -> PA18. The third argument 'sets' the type of pinmux we're going to use.

In this case, these are set to be used as PIO_TIMER!

We have to tell the Arduino core to change the MUX type before we reassign the SERCOM to this pin

You can do that by calling **pinPeripheral(***pinnumber, function***)** which is an internal function but hey we're being hardk0re

```
#include <Arduino.h>   // required before wiring_private.h
#include "wiring_private.h" // pinPeripheral() function

Uart Serial2 (&sercom1, 11, 10, SERCOM_RX_PAD_0, UART_TX_PAD_2);
void SERCOM1_Handler()
{
  Serial2.IrqHandler();
}

void setup() {
  Serial.begin(115200);

  Serial2.begin(115200);

  // Assign pins 10 & 11 SERCOM functionality
  pinPeripheral(10, PIO_SERCOM);
  pinPeripheral(11, PIO_SERCOM);
}

uint8_t i=0;
void loop() {
  Serial.print(i);
  Serial2.write(i++);
  if (Serial2.available()) {
    Serial.print(" -> 0x"); Serial.print(Serial2.read(), HEX);
  }
  Serial.println();

  delay(10);
}
```

You can use your oscilloscope to see the data traces

> Say you only want to transmit Serial data, you can leave out the pinPeripheral(10, PIO_SERCOM) line, and you can keep using #10 however you like

We can also try SERCOM2:

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA14 D2  SERCOM2.2 SERCOM4.2
PA09 D3  SERCOM0.1 SERCOM2.1
PA08 D4  SERCOM0.0 SERCOM2.0
PA15 D5  SERCOM2.3 SERCOM4.3
```

Lets put **TX** on D4 (SERCOM2.0) and **RX** on D3 (SERCOM2.1)

```
#include <Arduino.h>   // required before wiring_private.h
#include "wiring_private.h" // pinPeripheral() function

Uart Serial2 (&sercom2, 3, 4, SERCOM_RX_PAD_1, UART_TX_PAD_0);
void SERCOM2_Handler()
{
  Serial2.IrqHandler();
}

void setup() {
  Serial.begin(115200);

  Serial2.begin(115200);

  // Assign pins 3 & 4 SERCOM functionality
  pinPeripheral(3, PIO_SERCOM_ALT);
  pinPeripheral(4, PIO_SERCOM_ALT);
}

uint8_t i=0;
void loop() {
  Serial.print(i);
  Serial2.write(i++);
  if (Serial2.available()) {
    Serial.print(" -> 0x"); Serial.print(Serial2.read(), HEX);
  }
  Serial.println();

  delay(10);

}
```

Note that since pin 3 and 4 use a SERCOM alt mux, we have to pass in PIO_SERCOM_ALT rather than PIO_SERCOM when we call pinPeripheral()

Creating a new Wire

Compared to SPI, I2C (a.k.a Two-Wire or sometimes just refered to it's Arduino name, Wire) is even simpler - it's only 2 pins, SCL and SDA. I2C ports are a *huge* pain to emulate or bitbang due to the somewhat convoluted interface with multiple exceptions (repeated start! clock stretching!) and the bidirectional SDA line.

For that reason, being able to create new Wire's is *awesome*

How Wire is Created Now

Luckily, Atmel & Arduino did a really great job with structuring SERCOMs so you can create and assign new ones. You can find the actual code that is used to create the "Wire" SERCOM in Wire.cpp (https://adafru.it/l4B)

```
TwoWire Wire(&PERIPH_WIRE, PIN_WIRE_SDA, PIN_WIRE_SCL);

void WIRE_IT_HANDLER(void) {
  Wire.onService();
}
```

The macros are see in variants.h (https://adafru.it/l3C)

```
#define PIN_WIRE_SDA         (20u)
#define PIN_WIRE_SCL         (21u)
#define PERIPH_WIRE          sercom3
#define WIRE_IT_HANDLER      SERCOM3_Handler
```

You can see here that **Wire** is of type **TwoWire** and if you expand the macros:

```
      TwoWire Wire(&sercom3, 20, 21);
```

We can match this up with the pins we know are used for I2C:

```
PA22 D20 / SDA SERCOM3.0 SERCOM5.0
PA23 D21 / SCL SERCOM3.1 SERCOM5.1
```

You can see that the first argument (&sercom3) does in fact match our observation that I2C is on SERCOM3. The second and third arguments match with the SDA & SCL pins.

Unlike Serial or SPI, there are no passed in arguments for the MUXing. I2C, rather, has 'fixed' pads:

27.4 Signal Description

Signal Name	Type	Description
PAD[0]	Digital I/O	SDA
PAD[1]	Digital I/O	SCL
PAD[2]	Digital I/O	SDA_OUT (4-wire)
PAD[3]	Digital I/O	SDC_OUT (4-wire)

Refer to "I/O Multiplexing and Considerations" on page 21 for details on the pin mapping for this peripheral. One signal can be mapped on several pins. Note that not all the pins are I^2C pins. Refer to Table 6-1 for details on the pin type for each pin.

Pad **0** will *always* be SDA and pad **1** will *always* be SCL

OK so let's make a new I2C SERCOM already

Now we're ready, lets try creating a new SERCOM and testing it out

Let's make a Wire device on SERCOM 1

Since we have to use pads 0 & 1, lets check the mux table:

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA18 D10  SERCOM1.2 SERCOM3.2
PA16 D11  SERCOM1.0 SERCOM3.0
PA19 D12  SERCOM1.3 SERCOM3.3
PA17 D13  SERCOM1.1 SERCOM3.1
```

SDA will be on SERCOM1.0 **D11** and SCL will be on SERCOM1.1 **D13**

The definition for the new Wire will look like this:

```
TwoWire Wire(&sercom1, 11, 13);
```

Lets test it out!

```
#include <Wire.h>

TwoWire myWire(&sercom1, 11, 13);

void setup() {
  Serial.begin(115200);
  myWire.begin();
}

uint8_t i=0;
void loop() {
  Serial.println(i);
  myWire.beginTransmission(0x55); // start transmission to device
  myWire.write(i); // send a byte
  myWire.endTransmission(); // end transmission, actually sending
}
```

OK ... that wont work. Why? Good question! Check out those pin definitions in the variants.cpp file:

```
const PinDescription g_APinDescription[]=
...
  // 11
  { PORTA, 16, PIO_TIMER, (PIN_ATTR_DIGITAL|PIN_ATTR_PWM|PIN_ATTR_TIMER), No_ADC_Channel, PWM2_CH0, TCC2_
  // 13 (LED)
  { PORTA, 17, PIO_PWM, (PIN_ATTR_DIGITAL|PIN_ATTR_PWM), No_ADC_Channel, PWM2_CH1, NOT_ON_TIMER, EXTERNAL
```

The first argument is the port (e.g. PORTA or PORTB), the second is the port's pin number, e.g. PORTA, 16 -> PA16. The third argument 'sets' the type of pinmux we're going to use.

In this case, these are set to be used as PIO_TIMER & PIO_PWM

We have to tell the Arduino core to change the MUX type before we reassign the SERCOM to this pin

You can do that by calling **pinPeripheral(***pinnumber, function***)** which is an internal function but hey we're being hardk0re

Note that I've also put in code to control an MCP4725 by hand, since its a very simple I2C device (write 2 bytes to 0x40 to set an analog voltage out)

```
#include <Wire.h>
#include "wiring_private.h" // pinPeripheral() function

TwoWire myWire(&sercom1, 11, 13);

#define MCP4725_CMD_WRITEDAC            (0x40)
#define MCP4725_ADDR                    (0x62)
void setup() {
  Serial.begin(115200);
  myWire.begin();

  // Assign pins 13 & 11 to SERCOM functionality
  pinPeripheral(11, PIO_SERCOM);
  pinPeripheral(13, PIO_SERCOM);
}

uint8_t i=0;
void loop() {
  Serial.println(i);
  myWire.beginTransmission(MCP4725_ADDR); // start transmission to device
  myWire.write(MCP4725_CMD_WRITEDAC);
  myWire.write(i++);
  myWire.write((uint8_t)0x0);  // bottom four bits are 0x0
  myWire.endTransmission(); // end transmission, actually sending
}
```

2-10K pullup resistors are required on SDA and SCL, both go to 3.3V! You can use your oscilloscope to see the data traces

We can also try SERCOM2:

```
Pin Arduino 'Pin' SERCOM  SERCOM alt
---------------------------------------
PA14 D2  SERCOM2.2 SERCOM4.2
PA09 D3  SERCOM0.1 SERCOM2.1
PA08 D4  SERCOM0.0 SERCOM2.0
PA15 D5  SERCOM2.3 SERCOM4.3
```

We will have to have **SDA** on D4 (SERCOM2.0), **SCL** on D3 (SERCOM2.1)

```
#include <Wire.h>
#include "wiring_private.h" // pinPeripheral() function

TwoWire myWire(&sercom2, 4, 3);

#define MCP4725_CMD_WRITEDAC            (0x40)
#define MCP4725_ADDR                    (0x62)
void setup() {
  Serial.begin(115200);
  myWire.begin();

  // Assign pins 4 & 3 to SERCOM functionality
  pinPeripheral(4, PIO_SERCOM_ALT);
  pinPeripheral(3, PIO_SERCOM_ALT);
}

uint8_t i=0;
void loop() {
  Serial.println(i);
  myWire.beginTransmission(MCP4725_ADDR); // start transmission to device
  myWire.write(MCP4725_CMD_WRITEDAC);
  myWire.write(i++);
  myWire.write((uint8_t)0x0);  // bottom four bits are 0x0
  myWire.endTransmission(); // end transmission, actually sending
}
```

Note that since pin 3 and 4 use a SERCOM alt mux, we have to pass in PIO_SERCOM_ALT rather than PIO_SERCOM when we call pinPeripheral()