SAMD21/SAMR21 GPIO (I/O)

Overview

This tutorial will cover GPIO access for the SAMD21 & SAMR21 Microcontrollers. For the tutorial you can use a SAMD21 or a SAMR21 xplained pro development boards. The tutorial is broken into 2 sections; Using ASF to access the GPIO and accessing the GPIO registers directly without ASF.

This tutorial covers the following:

- How to add the Delay modules (ASF Libraries) to your AS7 Project
- GPIO access for the SAMD21 & SAMR21 Microcontrollers
- Flashing (programming) the SAMD21 & SAMR21 Microcontrollers

Tutorial Series Information

Tutorial Name	Series	Board	Category	Tutorial in Series	Difficulty
SAMD21/SAMR21 GPIO (I/O) Tutorial	SAMD21	SAMD21 xplained Pro	GPIO	2	Easy

Requirements

- Atmel Studio 7 (AS7)
- SAMD21 xplained Pro or SAMR21 xplained pro.
- Micro USB 2.0 Cable, Micro-B Male,
- 1 LED, 2 220ohm resister and a push button switch (optional, you can use the onboard button and switch)

Note – This tutorial will also work with a SAMR21 xplained Pro board or SAMD21 xplained pro. Substitute SAMD21 for SAMR21 as required.

Contact

You can reach me at acmbug@gmail.com if you have any questions of comments.

GPIO Overview

By default, an example project generated in Atmel Studio 7 with ASF provides all the functions you need access the GPIO pins. Because the SAMD21 and SAMDR21 xplained pro boards are prototyping boards, you may need to refer to both the datasheet and the xplained pro user guide for pin mappings.

I have taken the liberty of making a convenient table to show which pins are available. Table 1 shows a list of available pins, shared pins and pins that cannot be used.

SAMR21 xplained pro

Ext 1	Ext 2	Ext 3
PA04	**NA**	PA08
PA05	**NA**	PA14
PA06	**NA**	PA15
PA07	**NA**	PA16
PA13	**NA**	PA17
PA16	**NA**	PB02
PA17	**NA**	PB22
PA18	**NA**	PB23
PA19	**NA**	
PA22	**NA**	
PA23	**NA**	
PB02	**NA**	
PB03	**NA**	
PB22	**NA**	
PB23	**NA**	

SAMD21 xplained pro

Ext 1	Ext 2	Ext 3
PB00	PA10	PA02
PB01	PA11	PA03
PB06	PA20	PB30
PB07	PA21	PA15
PB02	PB12	PA12
PB03	PB13	PA13
PB04	PB14	PA28
PB05	PB15	PA27
PA08	PA08	PA08
PA09	PA09	PA09
PB09	PB11	PB11
PB08	PB10	PB10
PA05	PA17	PB17
PA06	PA18	PB22
PA04	PA16	PB16
PA07	PA19	PB23

Ext = Extension Header	
(yellow)	Shared Pin
(red)	Un useable * **

* PA13 on the SAMD21 is by default connected to the Serial Flash SS line and is disconnected from the EXT3 PIN8. This can be changed by moving the 0Ω resistor R314 to R313.
** PA17 on the SAMR21 is shared with EDBG
Ext = Extension

Table 1 - SAMD21/SAMR21 xplained pro Pins

In order to configure the GPIO pins, the following rules apply:

1. You must configure the pin or pins as INPUT or OUTPUT
2. If the Pin or Pins are set to OUTPUT, you must set its value to 1 in the corresponding control register or set the pin direction directly in the OUT register

PORT - I/O Pin Controller

The SAMD21 & SAMDR21 have 2 groups of 32 bit port controllers for configuring GPIO direction and output values. In addition, you can directly set the direction pins in the DIR register and output levels via the OUT register. Per the SAMD21 Datasheet states:

> The IO Pin Controller (PORT) controls the I/O pins of the device. The I/O pins are organized in a series of groups, collectively referred to as a port group. Each group can have up to 32 pins that can be configured and controlled individually or as a group. Each pin may either be used for general-purpose I/O under direct application control or be assigned to an embedded device peripheral. When used for general purpose I/O, each pin can be configured as input or output, with highly configurable driver and pull settings.
>
> ***All I/O pins have true read-modify-write functionality when used for general-purpose I/O; the direction or the output value of one or more pins may be changed (set, reset or toggled) explicitly without unintentionally changing the state of any other pins in the same port group by a single, atomic 8-, 16- or 32-bit write.***

The statement also applies to the SAMR21, but with a lot less pins (ref Table 1). For both devices, there are 2 IO PORT Controllers, A & B also referenced as PORTA & PORTB. I've highlighted the second paragraph to emphasize the importance of the statement. The statement means you can configure a pin or pins, change the state of the pin or pins without affecting other pins in the register with a single write. This is very convenient when you need to change many pins at once.

For this tutorial, you may need to reference the SAMD21/SAMR21 xplained pro user guide for pin mapping and functionality. You may also reference Table 1 of this tutorial.

Atmel Software Framework (ASF) and Structures

Before we start writing some code, I think it is important to discuss ASF and Structures. In my opinion, it is one of the key components to how ASF works. ASF leverages Structures AKA "struct" for configuring many of the devices functions including pin I/O. If you are coming from the Arduino world and never really worked with c, or an Object Oriented Programming language, then using ASF might be difficult for you to grasp. But, don't worry; I hope to take the sting out of ASF for you with this tutorial.

Structures provide a way of storing many different values in variables of potentially different types under the same name. Covering Structures is out of scope for this tutorial. However you can read more about structures here.

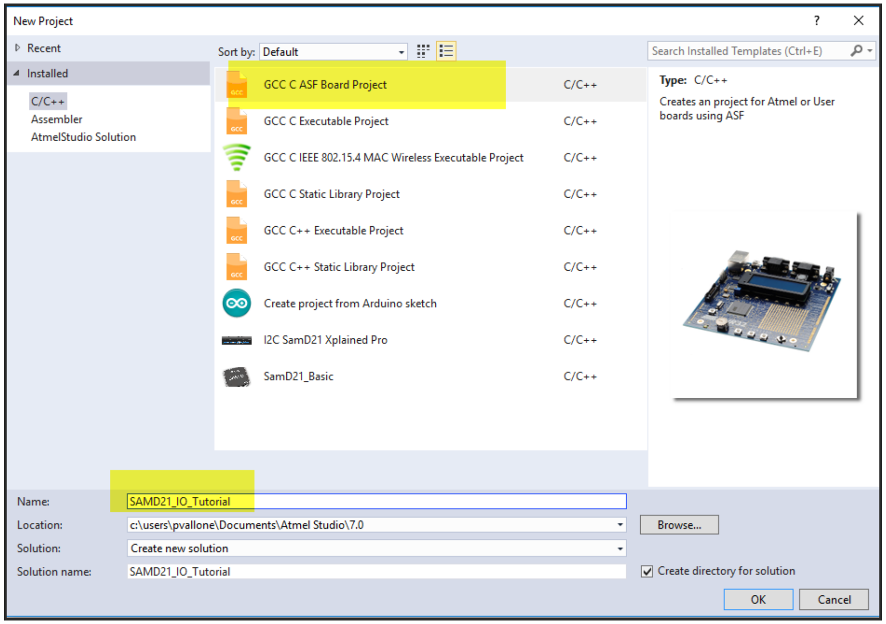
Create the Project

Lets start by creating a project in Atmel Studio 7 (AS7)

Open Atmel Studio 7

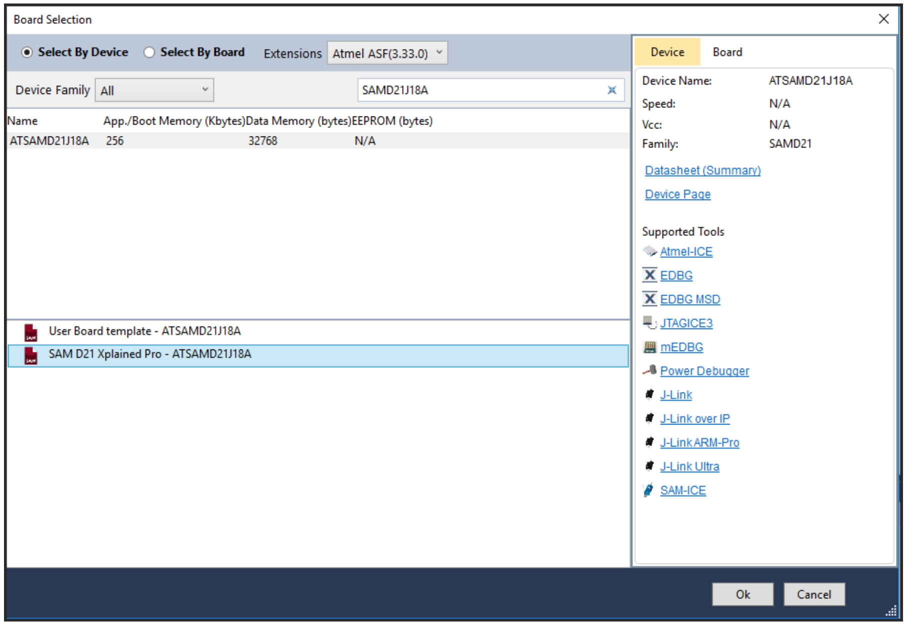
Click File->New->Project

Select GCC C ASF Board Project & enter a name for your project:



In the search box, enter SAMD21J18A if you are using a SAMD21 xplained pro. Enter ATSAMR21G18A if you are using a SAMR21 xplained pro.

Select SAMD21 xplained Pro board - SAMD21J18A or SAMR21 xplained pro - ATSAMR21G18A as applicable from the bottom window.



Click Ok

By creating a project this way, AS7 will add a lot of underlining code which will allow us easy access to the SAMD hardware.

By default when you start a **GCC C ASF Board Project** for an xplained pro board, ASF already sets the onboard LED and SWITCH directions for you. See Table 2 of the pin mapping for the built in LED and Button. The onboard LED and SWITCH can be seen in Figure 1.

Device	Pin	Function
SAMD21 xplained Pro	PB30	LED
SAMD21 xplained Pro	PA15	Button
SAMR1 xplained Pro	PA19	LED
SAMR1 xplained Pro	PA28	Button

Table 2 – Led and Button Mappings

In addition to setting the pin directions, AS7 also creates some code to allow you to toggle the LED when pressing the onboard button:

```
#include <asf.h>

int main (void)
{
	system_init();

	while (1) {
		/* Is button pressed? */
		if (port_pin_get_input_level(BUTTON_0_PIN) == BUTTON_0_ACTIVE) {
			/* Yes, so turn LED on. */
			port_pin_set_output_level(LED_0_PIN, LED_0_ACTIVE);
		} else {
			/* No, so turn LED off. */
			port_pin_set_output_level(LED_0_PIN, !LED_0_ACTIVE);
		}
	}
}
```

Before we upload our code to the xplained pro, we need to connect it to the PC.

Connecting the SAMD21/SAMR21 xplained pro

The SAMD21/SAMR21 xplained pro have 2 micro-b USB connectors. The DEBUG USB is for programming and debugging and the TARGET USB is for attaching a USB drive, which can be used for mass storage, like a thumb drive (see Figure 1).

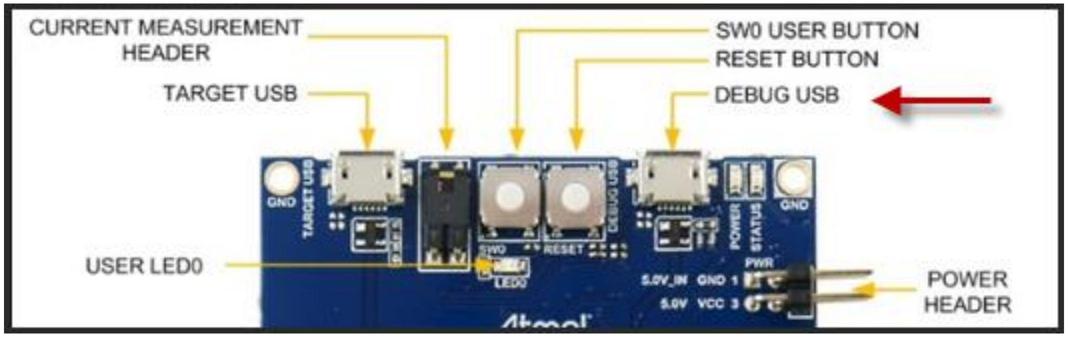


Figure 1 – SAMD21 Target and Debug connections

We need to connect to the DEBUG USB port. Connect the USB micro-b to the DEBUG USB port and the other end to your PC (see Figure 2).

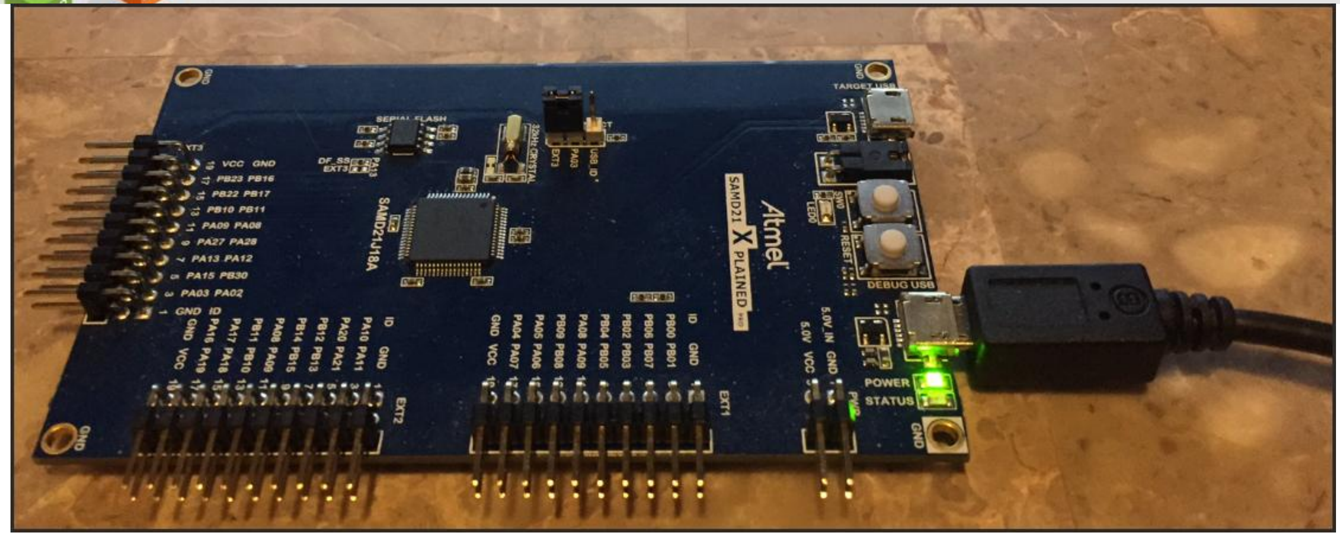


Figure 2 – SAMD21 xplained Pro connect to the PC

When you connect an xplained pro board (see Figure 2), AS7 will detect the board and launch a window which provides a bunch of information about your board (see Figure 3).

Also, note that each board has a unique identifier, which allows you to have multiple boards connected to your PC at one time. This means you will have to select which board to program. We will cover this later in the tutorial.

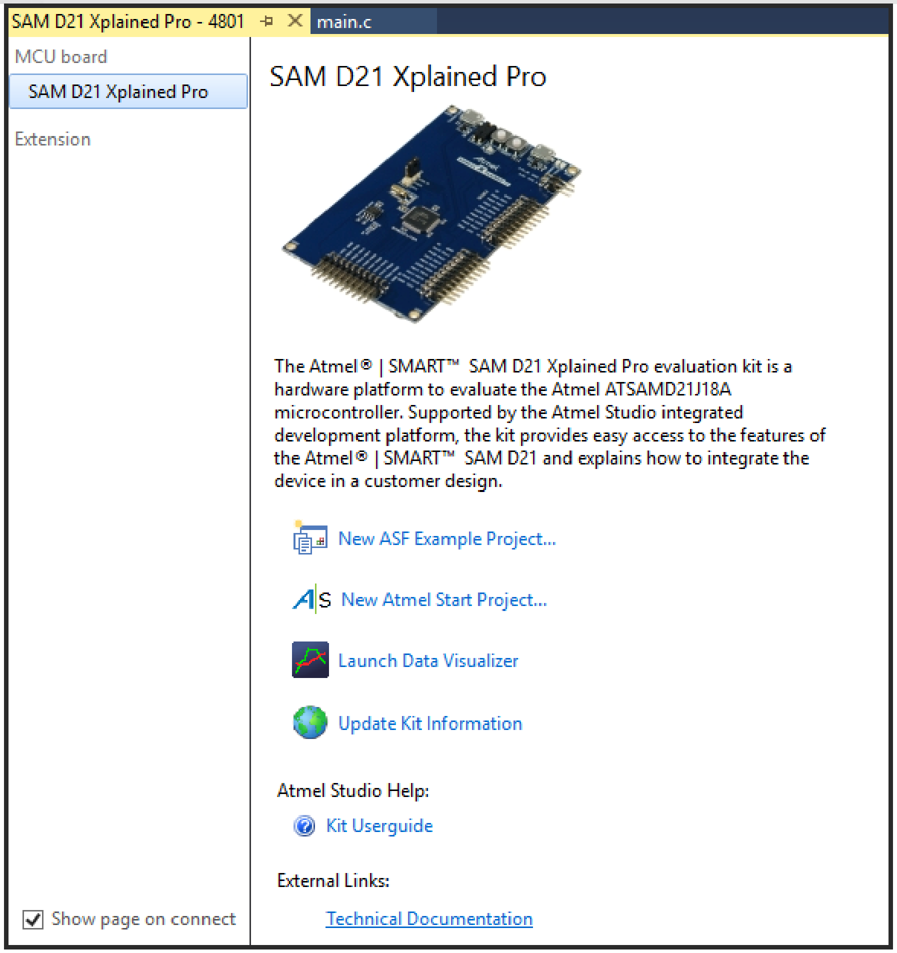


Figure 3 - SAMD21 xplained Pro details page

The project is ready to be compiled and programmed to our xplained board.

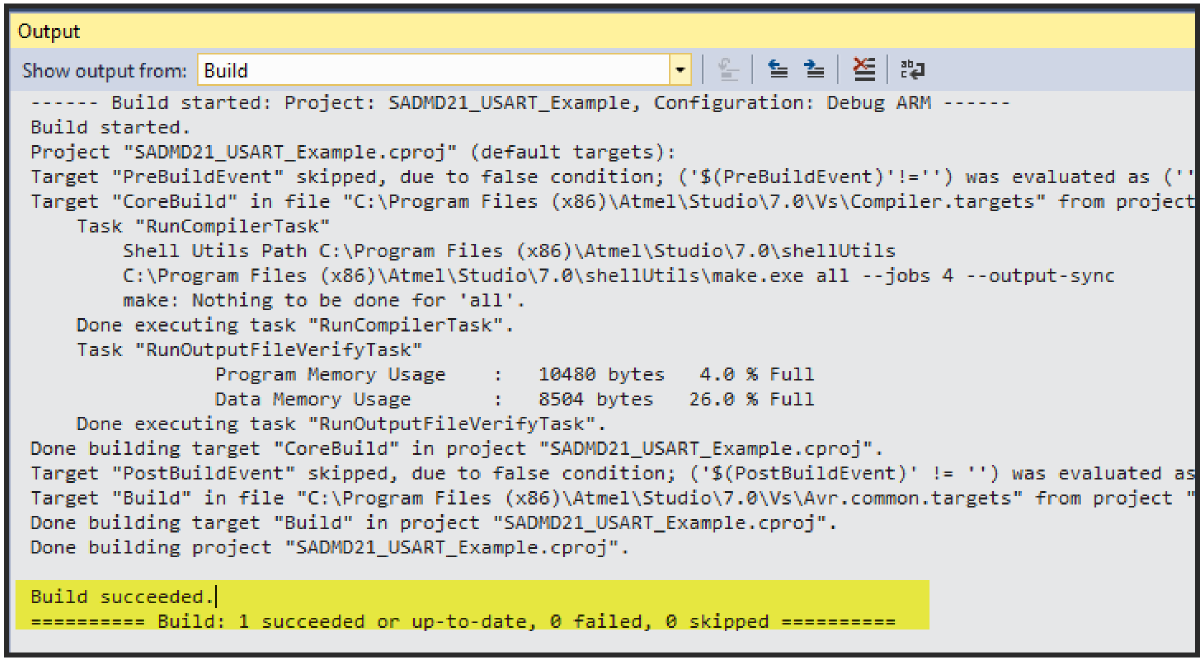
Build and Flash the SAMD21/SAMR21

Now we are ready to build the solution and program (flash) the SAMD21/SAMR21.

Build Solution

Click F7 or Build > Build Solution

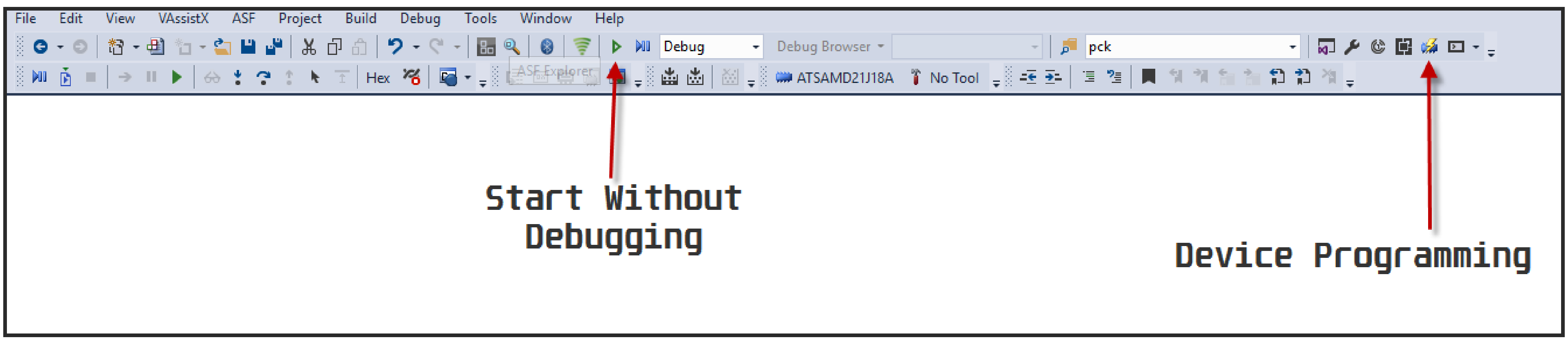
On a successful build, you will see something like this in the output window:



```
Output
Show output from: Build
------ Build started: Project: SADMD21_USART_Example, Configuration: Debug ARM ------
Build started.
Project "SADMD21_USART_Example.cproj" (default targets):
Target "PreBuildEvent" skipped, due to false condition; ('$(PreBuildEvent)'!='') was evaluated as (''
Target "CoreBuild" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Compiler.targets" from project
    Task "RunCompilerTask"
        Shell Utils Path C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils
        C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils\make.exe all --jobs 4 --output-sync
        make: Nothing to be done for 'all'.
    Done executing task "RunCompilerTask".
    Task "RunOutputFileVerifyTask"
                Program Memory Usage    :   10480 bytes   4.0 % Full
                Data Memory Usage       :   8504 bytes   26.0 % Full
    Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "SADMD21_USART_Example.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "
Done building target "Build" in project "SADMD21_USART_Example.cproj".
Done building project "SADMD21_USART_Example.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Flash your program on the device

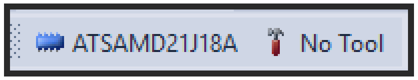
There are 2 ways to flash the device. The first is through **Device Programming button** and the second is through the **Start Without Debugging button**:



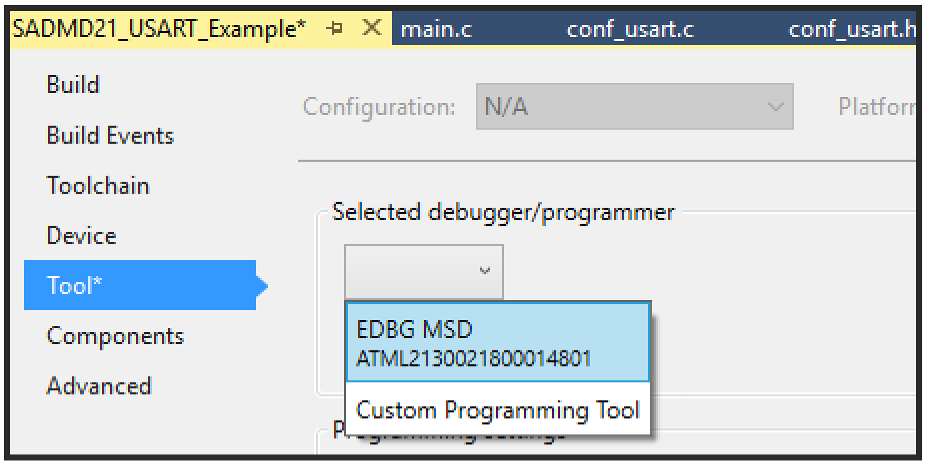
Start without Debugging

In order to use this option, you first need to select the Device you want to program.

To select a device, click the No Tool button:

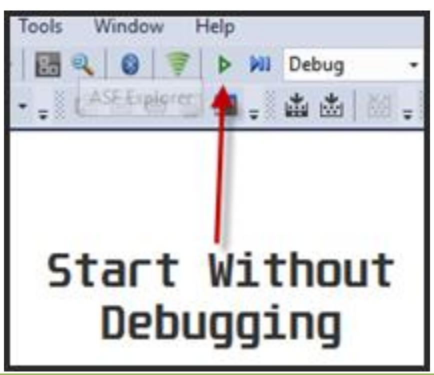


Under **Selected debugger/programmer**, choose your device:



Your device is now ready to be programmed with the **Start Without Debugging** button.

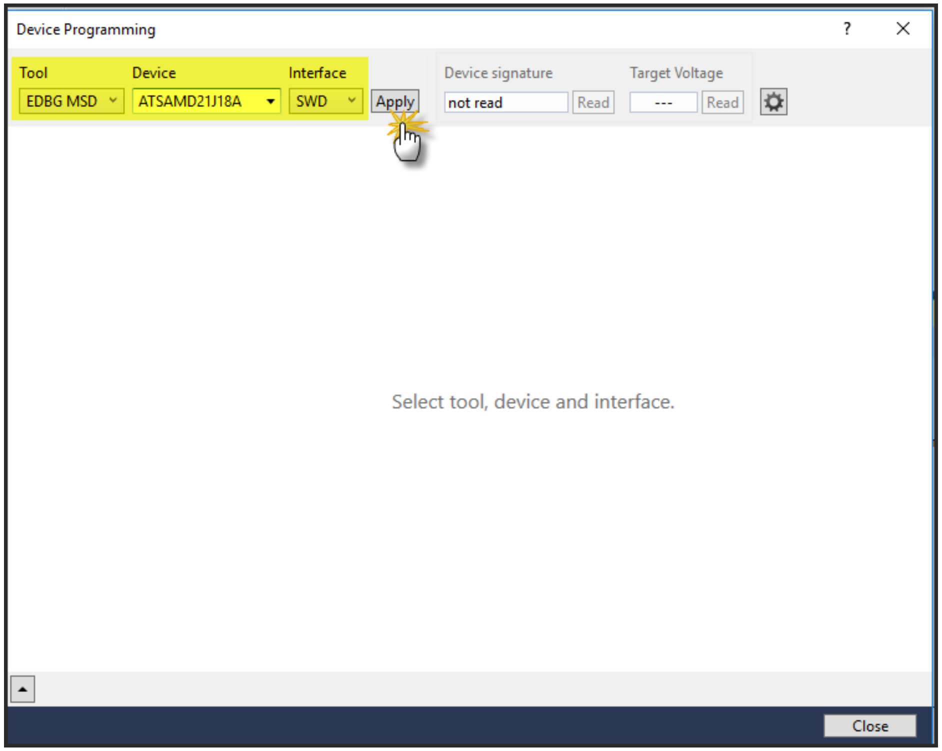
Click the **Start Without Debugging** to program your device.



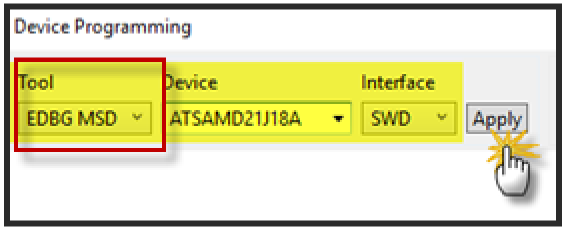
Device Programming

Another way to program a device in AS& is to use the **Device Programming button**. Click the **Device Programming button**:

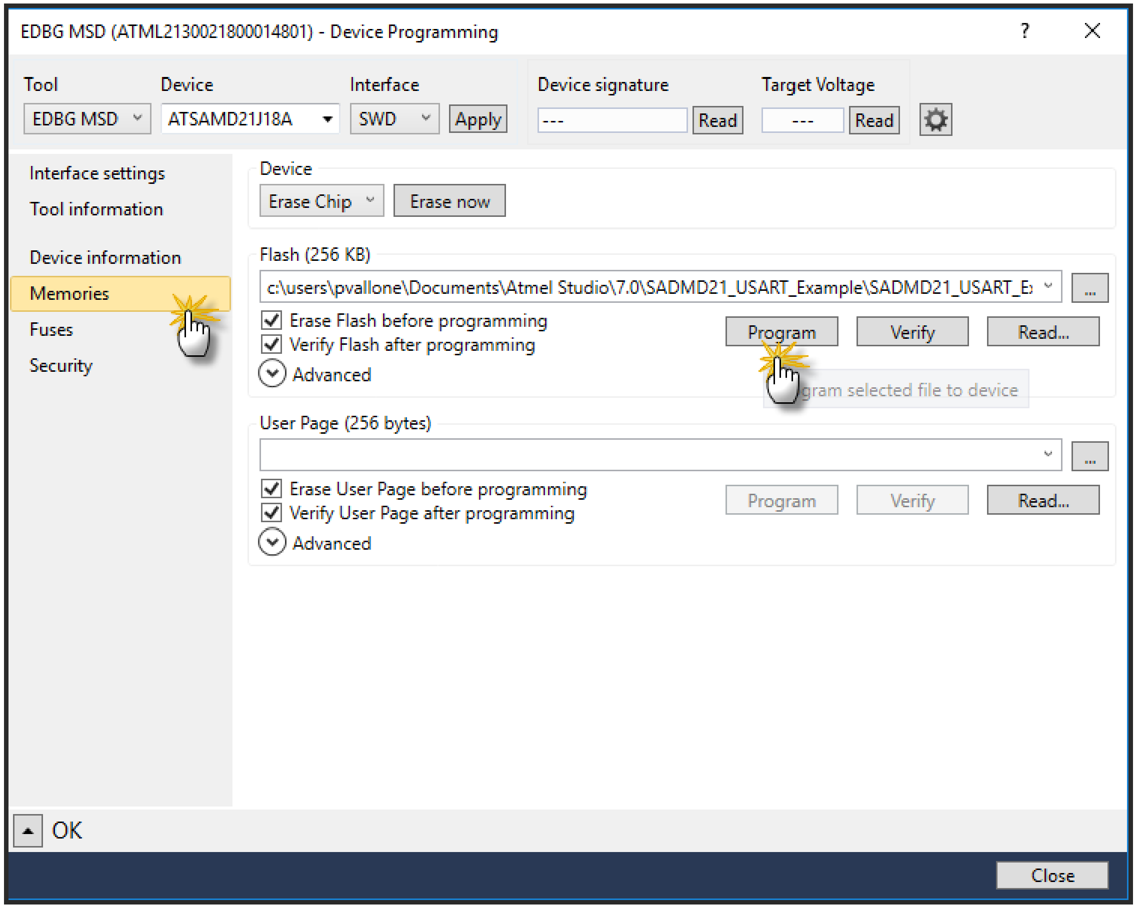
Select your Tool, Device and Interface as follows:



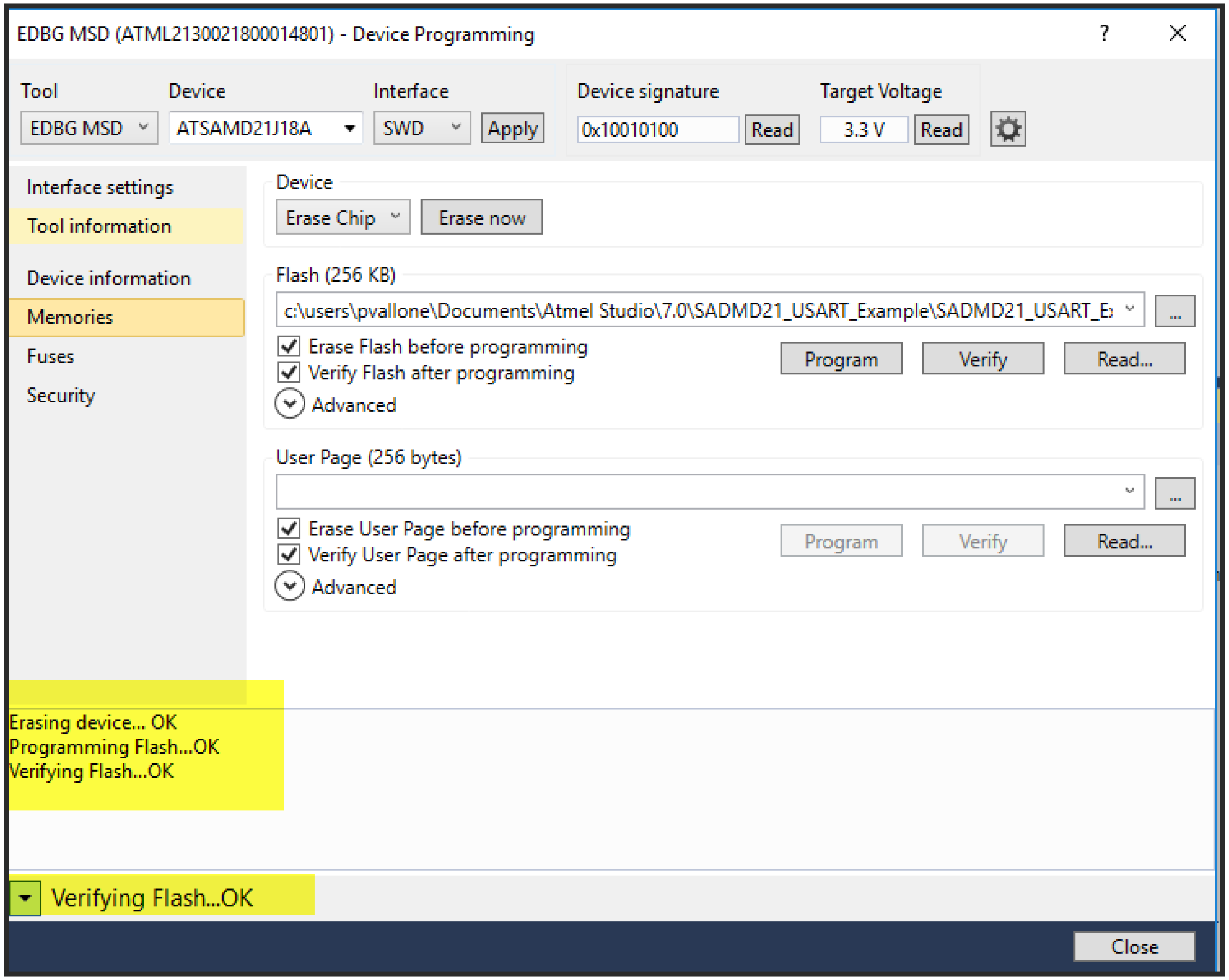
Note, if you have multiple xplained boards connected, you can choose with one to program from the Tool drop-down:



Click Memories > Program:



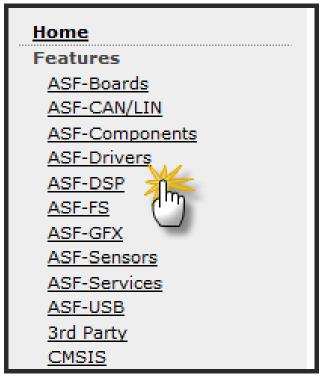
Verify the programming was Successful:



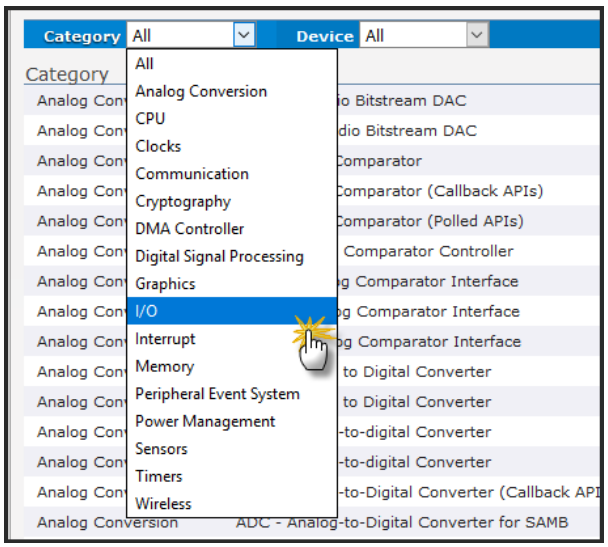
With the board programmed, you can press the button and the LED will light up. To see what is actually going on, you need to dive into the code or read the ASF Documentation. This is how all the Atmel examples work. You will have to dig into the code and see how it all works, read the docs or do both. This can be overwhelming at first. But don't panic; all you need to do it take it one step at a time.

How to Use the ASF Documentation

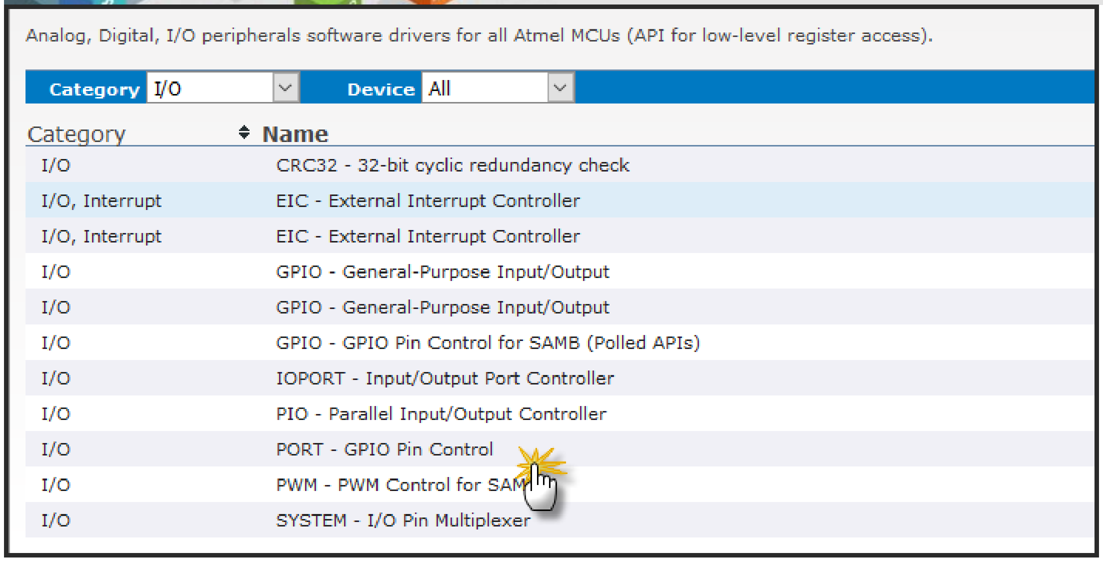
We are now going to see how to use the ASF documentation to some code. As a reference, the ASF documentation is located [here](). We will need to access the GPIO drivers. To access the GPIO drivers, click on ASF-Drivers under Features:



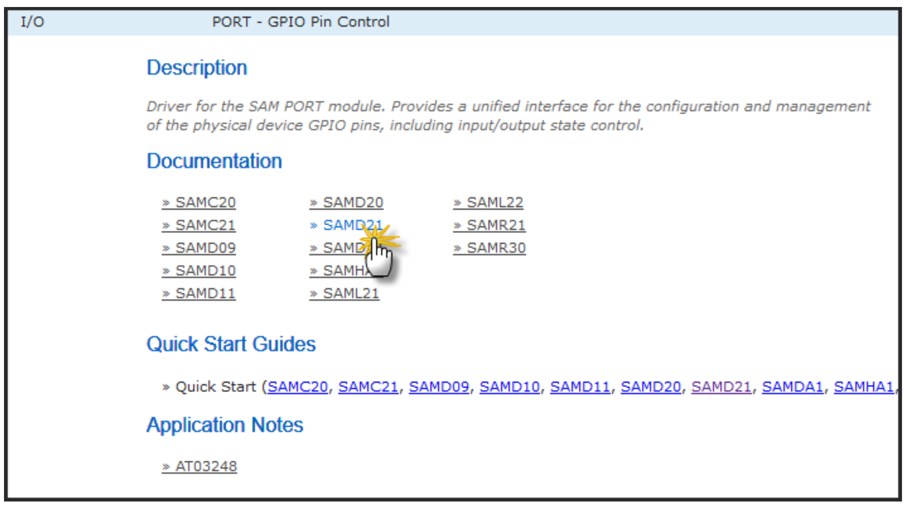
Now select IO from the Category:



Select PORT – GPIO Pin Control:

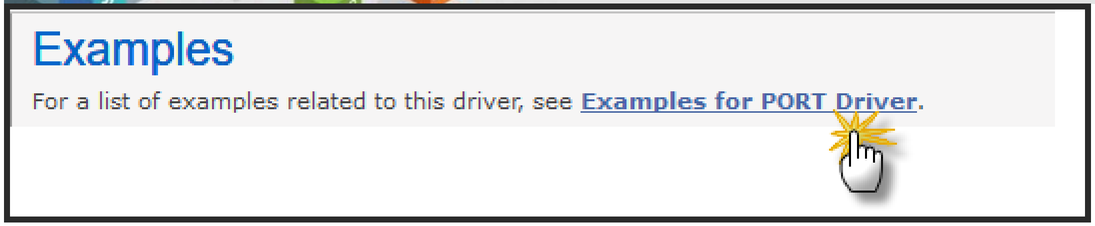


Select your device:



The documentation provides all the information about the GPIO driver as well as examples. For the purpose of this tutorial, click Examples:

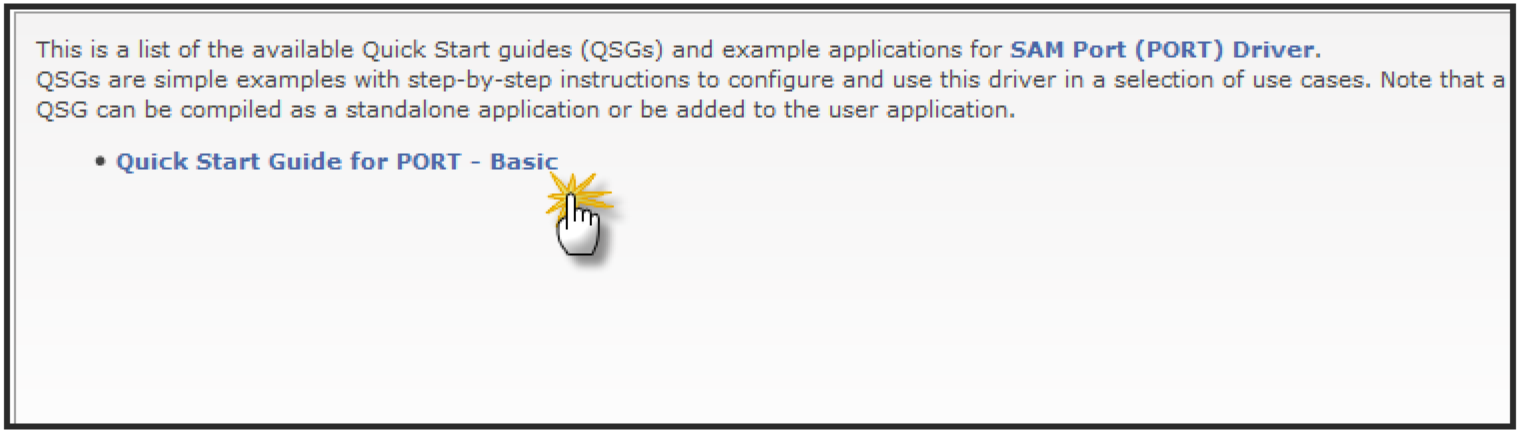
Then click Examples for PORT Driver.



Examples

For a list of examples related to this driver, see Examples for PORT Driver.

Then Click on the **Quick Start Guide for PORT – Basic**



This is a list of the available Quick Start guides (QSGs) and example applications for SAM Port (PORT) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for PORT - Basic

The ASF documentation provides nice examples for setting the PIN IO directions.

In this use case, the PORT module is configured for:

- One pin in input mode, with pull-up enabled
- One pin in output mode

This use case sets up the PORT to read the current state of a GPIO pin set as an input, and mirrors the opposite logical state on a pin configured as an output.

Setup

Prerequisites

There are no special setup requirements for this use-case.

Code

Copy-paste the following setup code to your user application:

```
void configure_port_pins(void)
{
    struct port_config config_port_pin;
    port_get_config_defaults(&config_port_pin);

    config_port_pin.direction  = PORT_PIN_DIR_INPUT;
    config_port_pin.input_pull = PORT_PIN_PULL_UP;
    port_pin_set_config(BUTTON_0_PIN, &config_port_pin);

    config_port_pin.direction = PORT_PIN_DIR_OUTPUT;
    port_pin_set_config(LED_0_PIN, &config_port_pin);
}
```

Add to user application initialization (typically the start of `main()`):

```
    configure_port_pins();
```

Please note, the documentation is not always straight forward. I found that the code can be incomplete. For example, ASF uses #defines to set up functions and variables. But the documentation doesn't always tell you what to define in order access a variable or function. This where you need to inspect the examples to see what is needed. This can be frustrating at times.

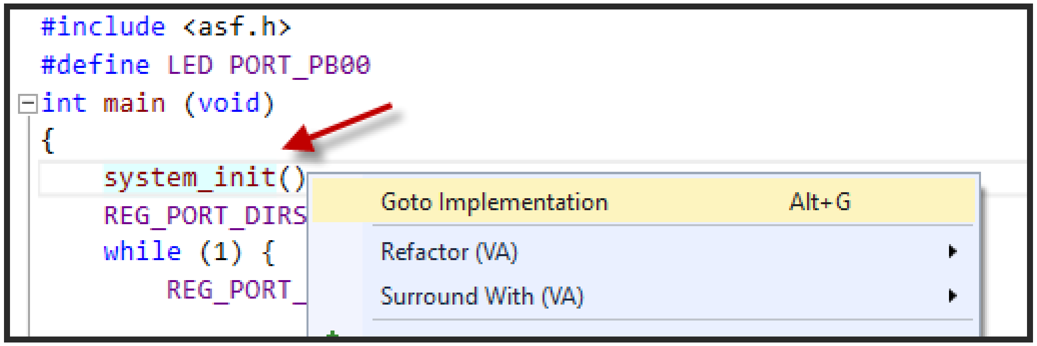
Inspecting the Example Project

The good news here is that AS7 which is built on Visual Studio allows you to easily inspect code. If you right click on a variable or function, you can navigate to the declaration or implementation. Pretty cool.

```
int main (void)
{
    system_init();
        ASF\sam0\drivers\system\system.c:95   void system_init(void){...}   1
        ASF\sam0\drivers\system\system.h:558  void system_init(void)        2
    while (1) {
        /* Is but
        if (port_pin_get_input_level(BUTTON_0_PIN) == BUTTON_0_ACTIVE) {
            /* Yes, so turn LED on. */
            port_pin_set_output_level(LED_0_PIN, LED_0_ACTIVE);
        } else {
```

For example, if we want to see what system_init() does, we can go to the implementation.

Right click on system_init()



You can then drill down further. By looking at the methods being called, we can see one called system_board_init(). This is where ASF will sets the I/O for the project's board (SAMD21 xplained pro or SAMR21 xplained pro).

```
void system_init(void)
{
    /* Configure GCLK and clock sources according to conf_clocks.h */
    system_clock_init();

    /* Initialize board hardware */
    system_board_init();

    /* Initialize EVSYS hardware */
    _system_events_init();

    /* Initialize External hardware */
    _system_extint_init();

    /* Initialize DIVAS hardware */
    _system_divas_init();
}
```

Right Click on system_board_init() and select **Goto Implementation**.

```
void system_board_init(void)
{
    struct port_config pin_conf;
    port_get_config_defaults(&pin_conf);

    /* Configure LEDs as outputs, turn them off */
    pin_conf.direction  = PORT_PIN_DIR_OUTPUT;
    port_pin_set_config(LED_0_PIN, &pin_conf);
    port_pin_set_output_level(LED_0_PIN, LED_0_INACTIVE);

    /* Set buttons as inputs */
    pin_conf.direction  = PORT_PIN_DIR_INPUT;
    pin_conf.input_pull = PORT_PIN_PULL_UP;
    port_pin_set_config(BUTTON_0_PIN, &pin_conf);

#ifdef CONF_BOARD_AT86RFX
    port_get_config_defaults(&pin_conf);
    pin_conf.direction  = PORT_PIN_DIR_OUTPUT;
    port_pin_set_config(AT86RFX_SPI_SCK, &pin_conf);
    port_pin_set_config(AT86RFX_SPI_MOSI, &pin_conf);
    port_pin_set_config(AT86RFX_SPI_CS, &pin_conf);
    port_pin_set_config(AT86RFX_RST_PIN, &pin_conf);
    port_pin_set_config(AT86RFX_SLP_PIN, &pin_conf);
    port_pin_set_output_level(AT86RFX_SPI_SCK, true);
    port_pin_set_output_level(AT86RFX_SPI_MOSI, true);
    port_pin_set_output_level(AT86RFX_SPI_CS, true);
    port_pin_set_output_level(AT86RFX_RST_PIN, true);
    port_pin_set_output_level(AT86RFX_SLP_PIN, true);
    pin_conf.direction  = PORT_PIN_DIR_INPUT;
    port_pin_set_config(AT86RFX_SPI_MISO, &pin_conf);
#endif
}
```

You can clearly see the where the code is setting the LED and Button directions. You can also see if `CONF_BOARD_AT86RFX` was defined, the block of code would be compiled. This is where ASF gets tricking. ASF uses “defines” a lot in the examples. It makes sense because there code is used on many devices. The trick is knowing how and where they all are and what to define.

Just so be clear, using ASF, here is how you set a pin as output & input:

```
struct port_config pin_conf; // create a strut to hold defaults
port_get_config_defaults(&pin_conf);

/* Configure LEDs as outputs, turn them off */
pin_conf.direction  = PORT_PIN_DIR_OUTPUT; // set pin to output
port_pin_set_config(LED_0_PIN, &pin_conf); // configure pin direction
port_pin_set_output_level(LED_0_PIN, LED_0_INACTIVE); // set pin output level low

/* Set buttons as inputs */
pin_conf.direction  = PORT_PIN_DIR_INPUT; // set pin direction as input
pin_conf.input_pull = PORT_PIN_PULL_UP; // set has a pull up
port_pin_set_config(BUTTON_0_PIN, &pin_conf); // configure pin
```

If you want to see what LED_0_PIN is defined as, right click on it and select **Goto Implementation**.

```
void system_init(void)
{
    /* Configure GCLK and clock sources according to conf_clocks.h */
    system_clock_init();

    /* Initialize board hardware */
    system_board_init();

    /* Init
    _system

    /* Init
```

boards\samd21_xplained_pro\board_init.c:60	void **system_board_init**(void){...}	1
boards\samd21_xplained_pro\samd21_xplained_pro.h:64	void **system_board_init**(void)	2
system.c:63	void **system_board_init**(void) WEAK __attribute__((alias("_system_dummy_init")))	3
system.c:69	void **system_board_init**(void)	4

```
 *  @{ */
#define LED_0_NAME                  "LED0 (yellow)"
#define LED_0_PIN                   LED0_PIN
#define LED_0_ACTIVE                LED0_ACTIVE
#define LED_0_INACTIVE              LED0_INACTIVE
#define LED0_GPIO                   LED0_PIN
```

Now right click on LED0_PIN and select **Goto Implementation**

```
/** \name LED0 definitions
 *  @{ */
#define LED0_PIN                  PIN_PB30
#define LED0_ACTIVE               false
#define LED0_INACTIVE             !LED0_ACTIVE
/** @} */
```

We can see the LED_0_PIN is mapped to PIN_PB30. Right click on PIN_PB30 and select **Goto Implementation** and select your board.

```
/** \name LED0 definitions
 *  @{ */
#define LED0_PIN                  PIN_PB30
#define LED0_ACTIVE               fa
#define LED0_INACTIVE             !L
/** @} */

/** \name SW0 definitions
 *  @{ */
#define SW0_PIN                   PI

utils\cmsis\samd21\include\pio\samd21j15a.h:150   #define PIN_PB30 62   1
utils\cmsis\samd21\include\pio\samd21j15b.h:147   #define PIN_PB30 62   2
utils\cmsis\samd21\include\pio\samd21j16a.h:150   #define PIN_PB30 62   3
utils\cmsis\samd21\include\pio\samd21j16b.h:147   #define PIN_PB30 62   4
utils\cmsis\samd21\include\pio\samd21j17a.h:150   #define PIN_PB30 62   5
utils\cmsis\samd21\include\pio\samd21j18a.h:150   #define PIN_PB30 62   6
```

LED_0_PIN's value is 62. This is the index ASF uses to select the port in the underlying code to configure the pin.

If you want to see exactly how ASF configures the pins, right click port_pin_set_config() select **Goto Implementation** and select your implementation. Keep drilling down and you can see how ASF does this set by step.

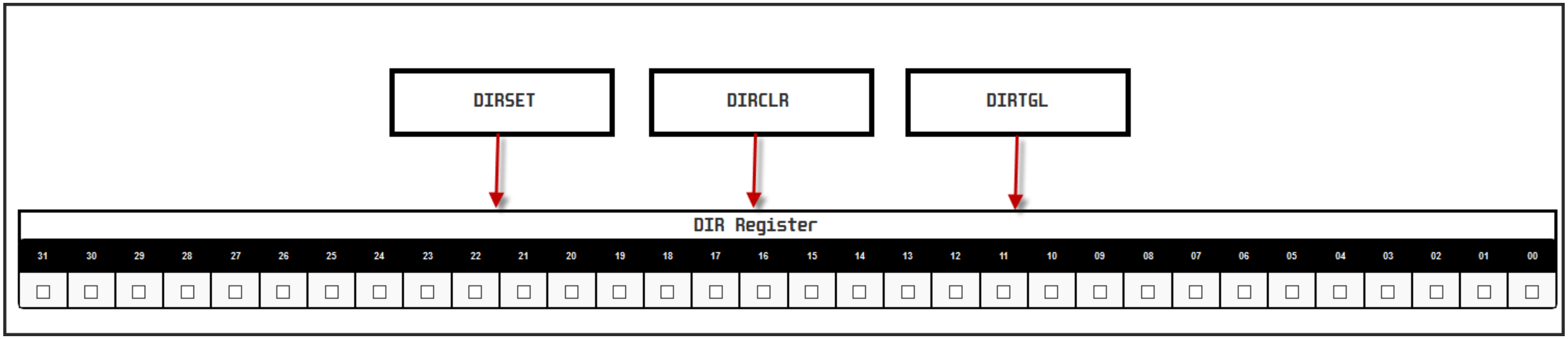
```
void system_board_init(void)
{
    struct port_config pin_conf;
    port_get_config_defaults(&pin_conf);

    /* Configure LEDs as outputs, turn them off */
    pin_conf.direction  = PORT_PIN_DIR_OUTPUT;
    port_pin_set_config(LED_0_PIN, &pin_conf);
    port_pin_set_outp
        drivers\port\port.c:63   void port_pin_set_config( const uint8_t gpio_pin, const struct port_config *const config){...}   1
        drivers\port\port.h:446  void port_pin_set_config( const uint8_t gpio_pin, const struct port_config *const config)   2
    /* Set buttons as
    pin_conf.direction  = PORT_PIN_DIR_INPUT;
    pin_conf.input_pull = PORT_PIN_PULL_UP;
    port_pin_set_config(BUTTON_0_PIN, &pin_conf);
```

Accessing GPIO without ASF

One of the first tutorials I found about the SAMD21 was Getting Started with the SAM D21 Xplained Pro without ASF. This is a great tutorial and I highly recommend you read it. We will be covering everything found in the tutorial and also going a little bit further.

Setting the PIN direction

Before we can use a pin for input or output, we need to set the pin's direction. Pin direction is controlled by the Data Control Registers and/or the DIR register. The control registers are broken into 2 groups PORTA & PORTB. You can configure pin directions by using the Data Direction Toggle (DIRTGL), Data Direction Clear (DIRCLR) and Data Direction Set (DIRSET) registers, or directly through the DIR register.



Data Direction Register (DIR)

This register allows the user to configure one or more I/O pins as an input or output. This register can be manipulated without doing a read-modify-write operation by using the Data Direction Toggle (DIRTGL), Data Direction Clear (DIRCLR) and Data Direction Set (DIRSET) registers.

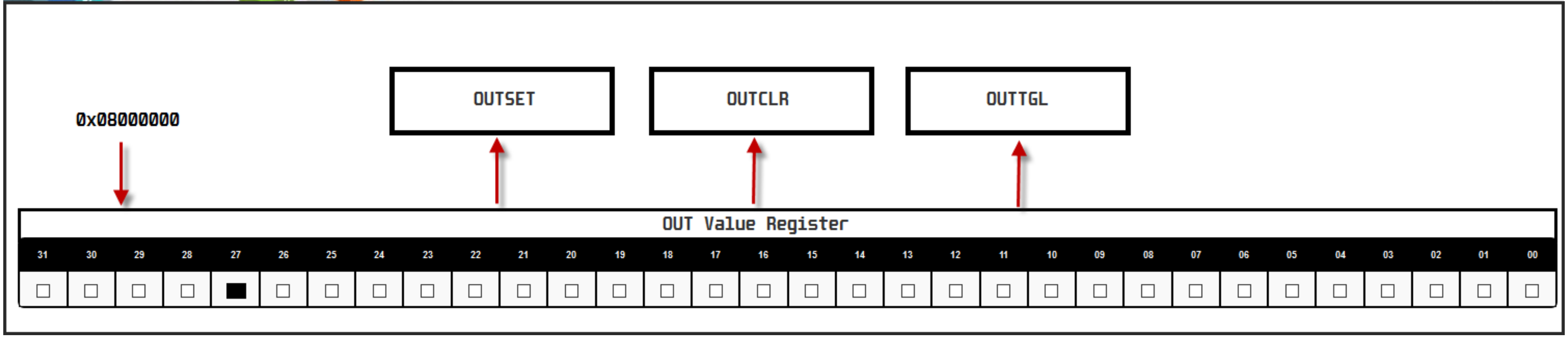
This means you do not have to access the DIR register directly. To configure one or more pins, you can set the values in the Data Direction Toggle (DIRTGL), Data Direction Clear (DIRCLR) and Data Direction Set (DIRSET) registers, which will configure the pin(s) in the DIR register.

The Data Control registers only requires you to set the pin to 1 to set the value. Setting the pin to 0 has no affect.

The DIR register will require you to use a bitwise inclusive OR and assignment operator |= when setting a pin or pins.

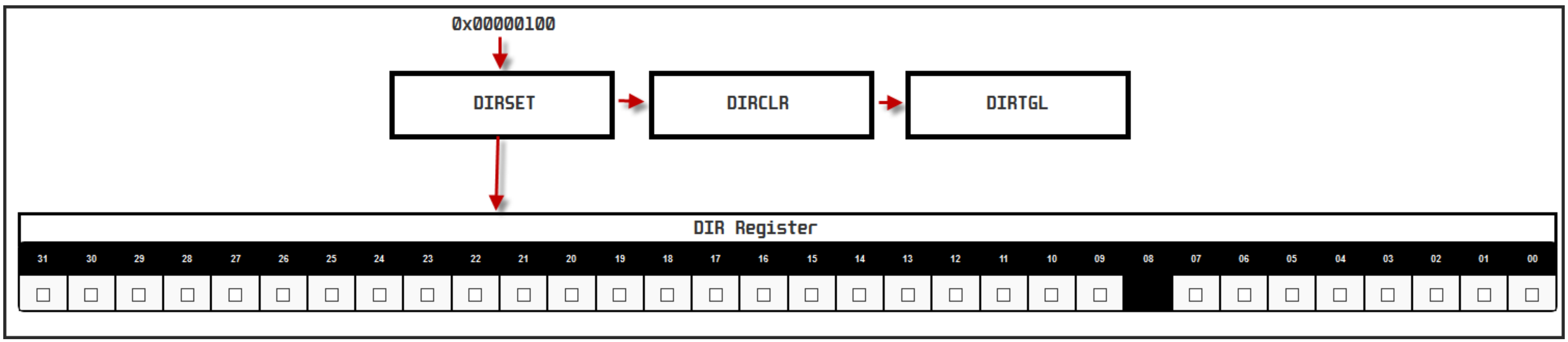
You have the choice to use either option.

Example: Setting the values in the DIR register directly will also set the values in the OUTSET, OUTCLR and OUTTGL registers:



Data Direction Set (DIRSET)

This register allows the user to set one or more I/O pins as an output, without doing a read-modify-write operation. Changes in this register will also be reflected in the Data Direction (DIR), Data Direction Toggle (DIRTGL) and Data Direction Clear (DIRCLR) registers.



Use the DIRSET register to set the direction of the pin in the DIR register.

For example, to set the direction of a pin in PORTA use:

```
REG_PORT_DIRSET0
```

To set the direction of a pin in PORTB use:

```
REG_PORT_DIRSET1
```

The 0 or 1 at the end of the variable name indicates the PORT group; 0 for PORTA and 1 for PORTB.

Setting a single pin

Here is an example for setting a single pin (PB08) to output. We will set the value using the hexadecimal representation of the binary 0b000100000000.

PORTB																															
PB31	**PB30**	**PB29**	**PB28**	**PB27**	**PB26**	**PB25**	**PB24**	**PB23**	**PB22**	**PB21**	**PB20**	**PB19**	**PB18**	**PB17**	**PB16**	**PB15**	**PB14**	**PB13**	**PB12**	**PB11**	**PB10**	**PB09**	**PB08**	**PB07**	**PB06**	**PB05**	**PB04**	**PB03**	**PB02**	**PB01**	**PB00**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
REG_PORT_DIRSET1 = PORT_PB08; // PORT_PB08 as binary = 0b00000000000000000000000100000000
```

If you convert 00000000000000000000000100000000 to Hexadecimal, the value is 0x00000100.

The SAMD21 PORT GPIO Pin Control driver provides a define for each port pin:

```
#define PIN_PB00                  32 /**< \brief Pin Number for PB00 */
#define PORT_PB00         (1ul <<  0) /**< \brief PORT Mask  for PB00 */
#define PIN_PB01                  33 /**< \brief Pin Number for PB01 */
#define PORT_PB01         (1ul <<  1) /**< \brief PORT Mask  for PB01 */
#define PIN_PB02                  34 /**< \brief Pin Number for PB02 */
#define PORT_PB02         (1ul <<  2) /**< \brief PORT Mask  for PB02 */
#define PIN_PB03                  35 /**< \brief Pin Number for PB03 */
#define PORT_PB03         (1ul <<  3) /**< \brief PORT Mask  for PB03 */
#define PIN_PB04                  36 /**< \brief Pin Number for PB04 */
#define PORT_PB04         (1ul <<  4) /**< \brief PORT Mask  for PB04 */
#define PIN_PB05                  37 /**< \brief Pin Number for PB05 */
#define PORT_PB05         (1ul <<  5) /**< \brief PORT Mask  for PB05 */
#define PIN_PB06                  38 /**< \brief Pin Number for PB06 */
#define PORT_PB06         (1ul <<  6) /**< \brief PORT Mask  for PB06 */
#define PIN_PB07                  39 /**< \brief Pin Number for PB07 */
#define PORT_PB07         (1ul <<  7) /**< \brief PORT Mask  for PB07 */
#define PIN_PB08                  40 /**< \brief Pin Number for PB08 */
#define PORT_PB08         (1ul <<  8) /**< \brief PORT Mask  for PB08 */
```

The define uses some bit shifting here. Starting from the LSB, they shift 8 bits << to the left, giving you 0b000100000000, or the hexadecimal value of 0x00000100. You have the option of using the defined values from ASF or you can set the values yourself.

Example:

```
#include <asf.h>

int main (void)
{
	system_init();

	REG_PORT_DIRSET1 = PORT_PB08; // Direction set to OUTPUT for PB08

	while (1) {

	}
}
```

Notice we don't need a bitwise inclusive OR and assignment operator |= when setting values in the DIRSET, DIRCLR or DIRTGL registers. Remember the port control registers only requires you to set the pin to 1 to set the value. Setting the pin to 0 has no affect.

Setting multiple pins

You can also set multiple bits at once. Suppose I want to set pins PB00 thru PB15 to output:

PORTB																															
PB31	PB30	PB29	PB28	PB27	PB26	PB25	PB24	PB23	PB22	PB21	PB20	PB19	PB18	PB17	PB16	PB15	PB14	PB13	PB12	PB11	PB10	PB09	PB08	PB07	PB06	PB05	PB04	PB03	PB02	PB01	PB00
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

If you convert 00000000000000001111111111111111 to hexadecimal, the value is 0x0000FFFF.

```
REG_PORT_DIRSET1 = 0x00FFFF; // PB00-PB15
```

Example:

```
#include <asf.h>

int main (void)
{
	system_init();

	REG_PORT_DIRSET1 = 0x0000FFFF; // Direction set to OUTPUT PB00-PB15
	REG_PORT_OUTSET1 = 0x0000FFFF; // set state of pin(s) to HIGH PB00-PB15

	while (1) {

	}
}
```

Or just set a few pins:

PORTB																															
PB31	PB30	PB29	PB28	PB27	PB26	PB25	PB24	PB23	PB22	PB21	PB20	PB19	PB18	PB17	PB16	PB15	PB14	PB13	PB12	PB11	PB10	PB09	PB08	PB07	PB06	PB05	PB04	PB03	PB02	PB01	PB00
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

```
REG_PORT_DIRSET1 = 0x00810401;
```

Setting a pin or pins output HIGH or LOW

Like setting the pins direction, to configure a pin or pins output level, you have several options.

Option 1 – directly setting bits in the Data Output Value (OUT) register. This is a more traditional method where you can set and flip bits.

Option 2 – Use the Data Output Value Clear (OUTCLR), Data Output Value Set (OUTSET), and Data Output Value Toggle (OUTTGL) registers which allows you to simply set a bits value to 1 to change the state of a pin in the Data Output Value (OUT) register.

Option 2 is very convenient, as I will demonstrate both options. You can choose your approach depending on your application.

Data Output Value (OUT) Register

This register sets the data output drive value for the individual I/O pins in the PORT. This register can be manipulated without doing a read-modify-write operation by using the Data Output Value Clear (OUTCLR), Data Output Value Set (OUTSET), and Data Output Value Toggle (OUTTGL) registers.

As stated you can directly access the OUT register to configure your pin(s) output levels directly. Here is an example to set PORT_PA27 to HIGH (Option 1):

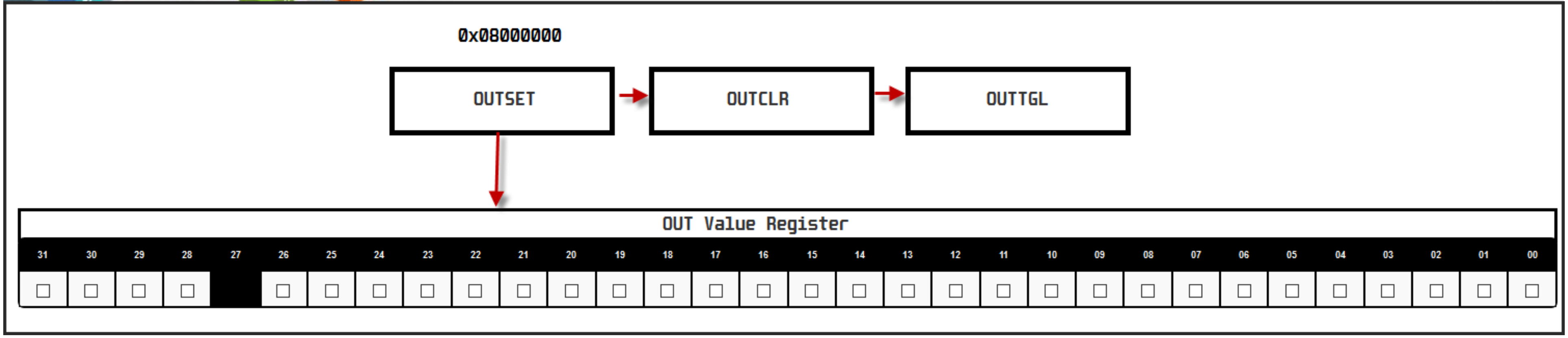
```
#include <asf.h>
#define LED PORT_PA27 // 0x08000000

int main (void)
{
	system_init();

	REG_PORT_DIR0 |= LED; // Direction set to OUTPUT directly
	REG_PORT_OUT0 |= LED; // set pin to HIGH directly

	while (1) {

	}
}
```

Later in the tutorial, we'll discuss adding a delay to toggle bits in the OUT registers.

As mentioned, we have another option - Option 2, use the Data Output Value Clear (OUTCLR), Data Output Value Set (OUTSET), and Data Output Value Toggle (OUTTGL) registers to allow you to simply set a bits value to 1 to change the state of a pin in the Data Output Value (OUT) register.

Data Output Value Set (OUTSET) Register

This register allows the user to set one or more output I/O pin drive levels high, without doing a read-modify-write operation. Changes in this register will also be reflected in the Data Output Value (OUT), Data Output Value Toggle (OUTTGL) and Data Output Value Clear (OUTCLR) registers.



To set pins or a single pin to HIGH or LOW, we need to configure the OUT register. The OUT register is controlled by the **OUTSET**, **OUTTGL** & **OUTCLR** registers.

You can access the **REG_PORT_OUTSET** registers in PORTA and PORT B with the variable `REG_PORT_OUTSET0` or variable `REG_PORT_OUTSET1`.

`REG_PORT_OUTSET0` for **PORTA**

`REG_PORT_OUTSET1` for **PORTB**

Remember, all you need to do is set the value of the pin(s) to 1. Setting 0 has no affect.

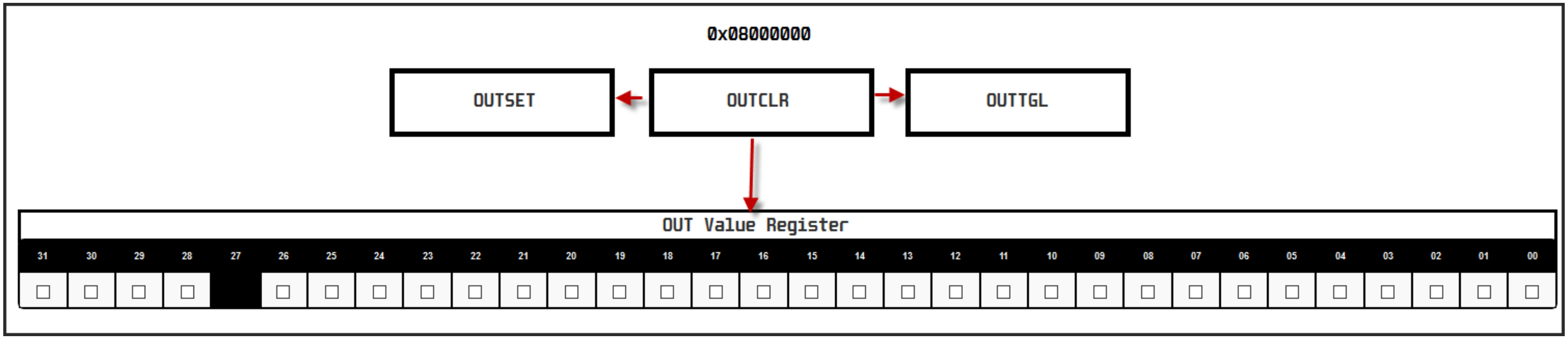
Here is an example for setting PIN PB27 to HIGH:

Example - Configure a pin for output and setting it HIGH:

```
#include <asf.h>

int main (void)
{
	system_init();

	REG_PORT_DIRSET1 = PORT_PB27; // set PB27 to output
	REG_PORT_OUTSET1 = PORT_PB27; // set PB27 HIGH

	while (1) {

	}
}
```

Data Output Value Clear (OUTCLR) Register

This register allows the user to set one or more output I/O pin drive levels low, without doing a readmodify-write operation. Changes in this register will also be reflected in the Data Output Value (OUT), Data Output Value Toggle (OUTTGL) and Data Output Value Set (OUTSET) registers.



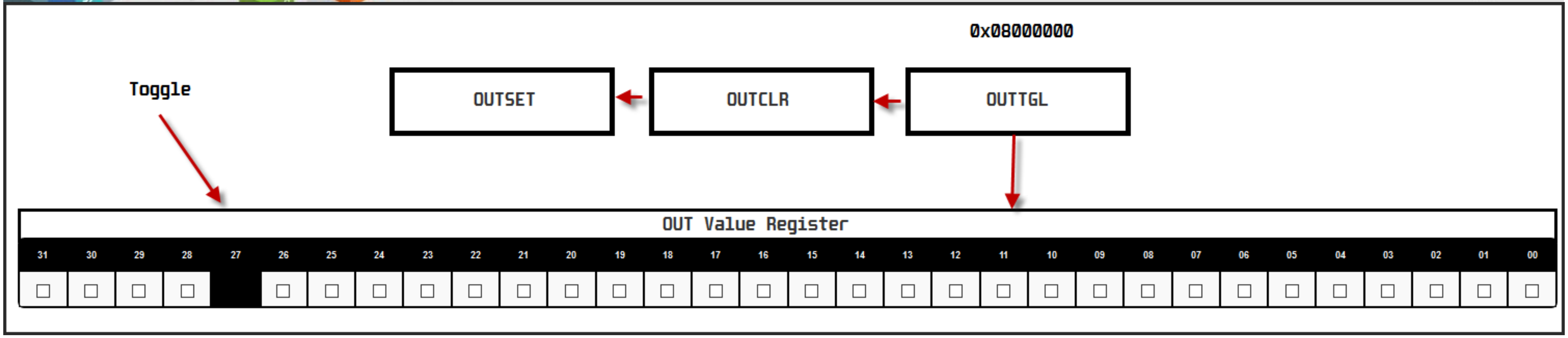
REG_PORT_OUTCLR clears the value in the **OUT** register of the by setting the values of the register to 1. Setting 0 has no affect.

Example - Configure a pin for output and setting it LOW:

```
#include <asf.h>

int main (void)
{
	system_init();

	REG_PORT_DIRSET1 = PORT_PB27; // set PB00 to output
	REG_PORT_OUTCLR1 = PORT_PB27; // set PB00 LOW

	while (1) {

	}
}
```

Data Output Toggle Clear (OUTTGL) Register

This register allows the user to toggle the drive level of one or more output I/O pins, without doing a readmodify-write operation. Changes in this register will also be reflected in the Data Output Value (OUT), Data Output Value Set (OUTSET) and Data Output Value Clear (OUTCLR) registers.



The variable **REG_PORT_OUTTGL** toggles the value in the **OUT** register by setting the values to 1. Setting 0 has no affect. Later in the tutorial we'll implement a blink program using the OUTTGL I/O controller.

Example - Configure a pin for output and toggle the pin:

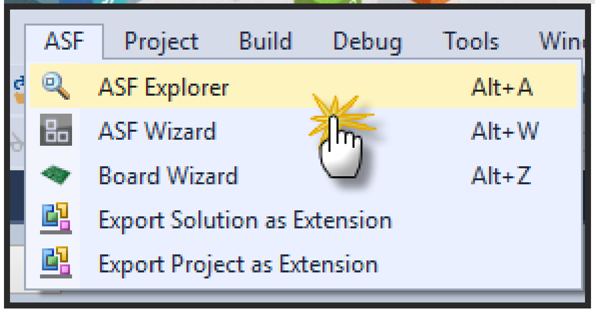
```
#include <asf.h>

int main (void)
{
	system_init();

	REG_PORT_DIRSET1 = PORT_PB27; // set PB27 to output
	REG_PORT_OUTTGL1 = PORT_PB27; // toggle PB27

	while (1) {

	}
}
```

I think you are getting the idea. Next we are going to introduce a delay to blink the LED. To add the delay modules, we need to tell AS7 to import the delay module to our project.

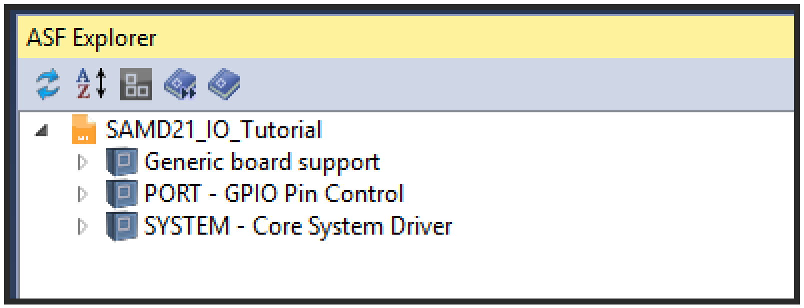
Adding a Delay

ASF has broken down its libraries into modules. In order to access these modules/libraries, you need to use the ASF Wizard to add the modules to your project.

By default ASF already adds modules to your project when you select your board when creating a project. To view the currently install modules, click ASF>ASF Explorer from the top menu:



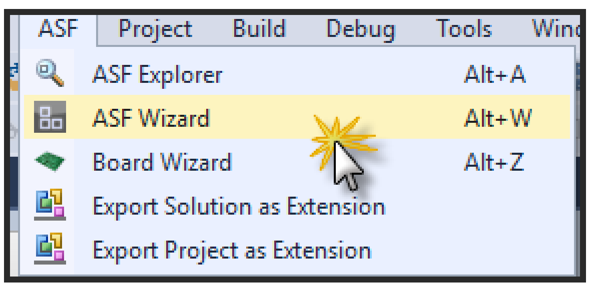
This will open the ASF Explorer panel. Click the triangle to see which modules are loaded for your project. By default 3 modules have been added:



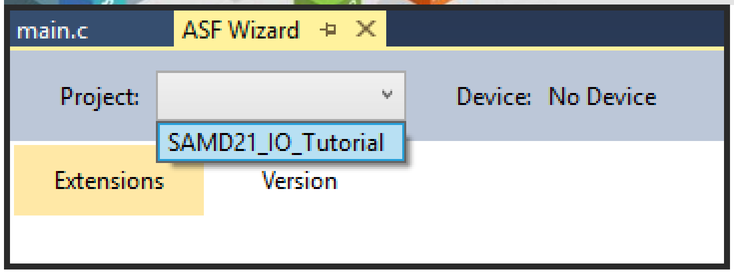
As you can see, we have 3 modules already. For this tutorial, you are going to need 1 more:

- Delay routines (service)

To add new modules, click ASF->ASF Wizard from the top menu



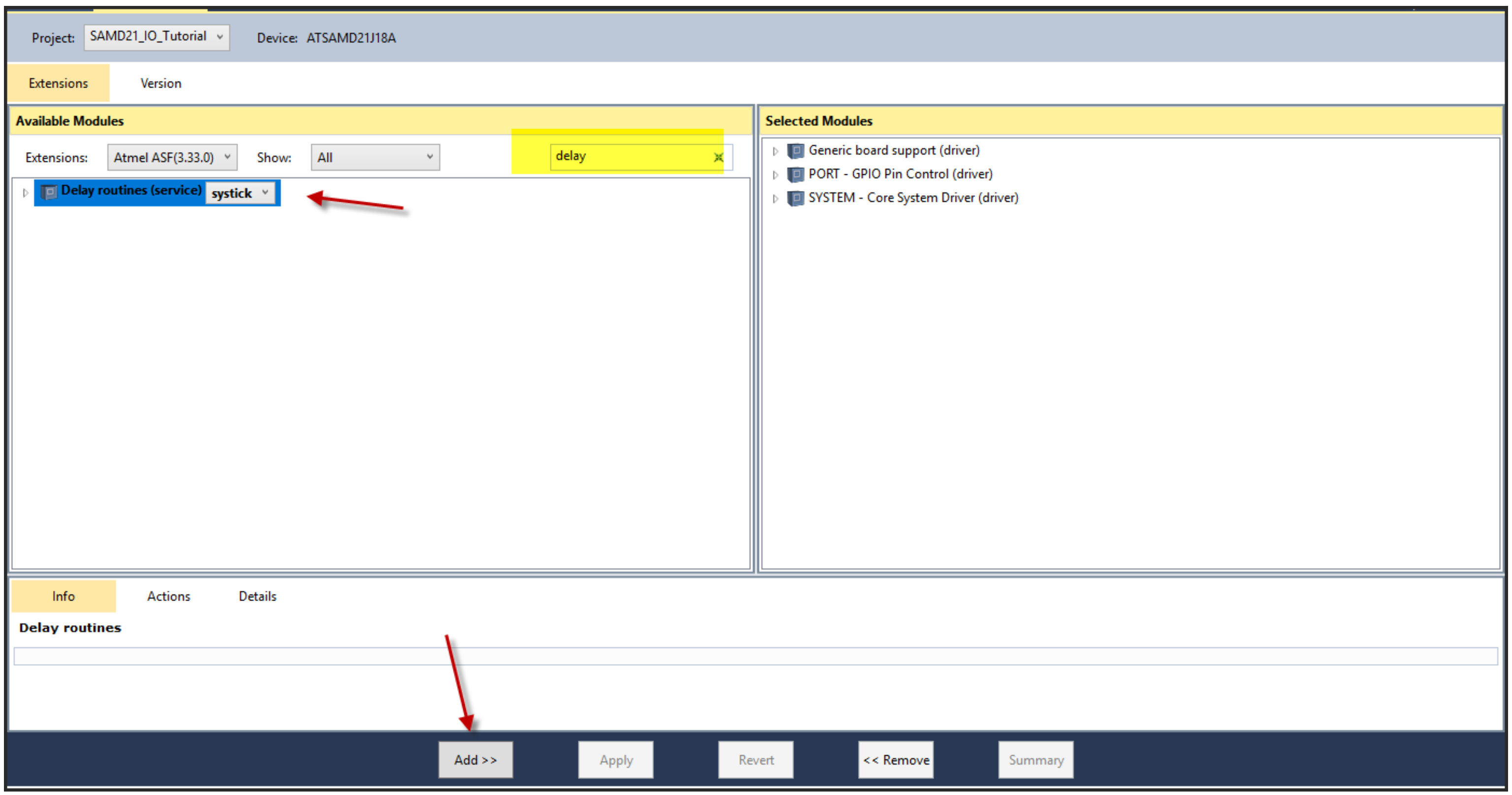
In the Project Dropdown, select your project:



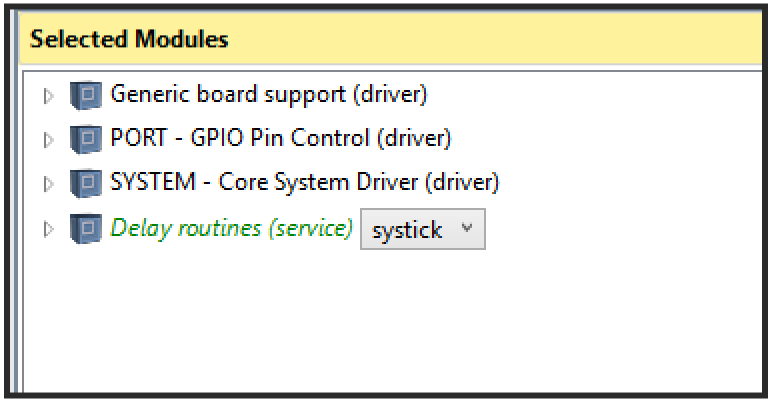
This will open the ASF Wizard. Here you can see all the available modules in the left pain and all modules currently installed in your project on the right.

In the search box, type “delay” to filter the results.

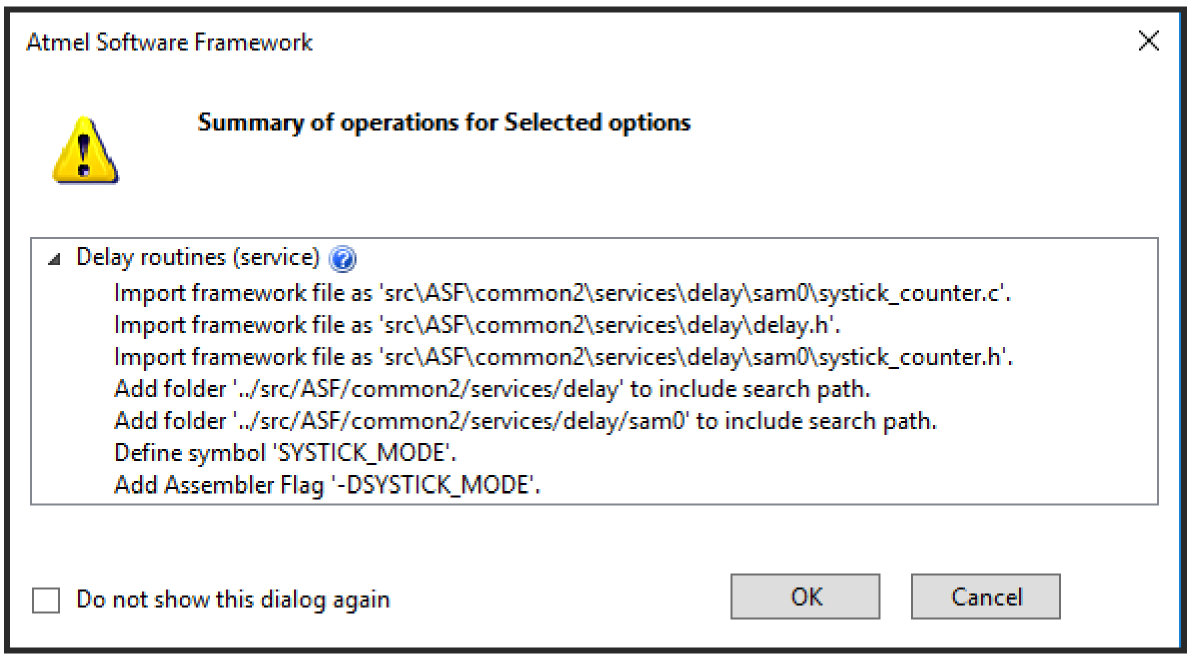
Select “Delay routines (service) systick”, click add



The module we be added to the Selected Modules panel:



Click Apply. A window will pop up asking if you want to add the modules files to your project. Click OK:



Implementing a Blink program

With the delay library installed, we will need to init the delay library. We are going to blink a LED connected to the `PORT_PA27` pin. We will use Option 1, accessing the OUT register directly and Option 2, use the port control Data Output Value (OUT), Data Output Value Toggle (OUTTGL) and Data Output Value Clear (OUTCLR) registers.

Change your main.c to look like this:

```
// example using option 2 Port Control Registers
#include <asf.h>
#define LED PORT_PA27 // define LED as PORT_PA27 (0x08000000)

int main (void)
{
	system_init();
	delay_init(); // init delay
	REG_PORT_DIRSET0 = LED; // Direction set to OUTPUT

	while (1) {

		REG_PORT_OUTTGL0 = LED; // toggle PORT_PA26
		delay_s(1); // delay for 1 second

	}
}
```

Each line is commented describing its function/purpose.

Alternately, you can use the REG_PORT_OUTCLR0 register to set PORT _PA26

```
// example using option 2 Port Control Registers
#include <asf.h>
#define LED PORT_PA27 // define LED as PORT_PA27 (0x08000000)

int main (void)
{
	system_init();
	delay_init(); // init delay
	REG_PORT_DIRSET0 = LED; // Direction set to OUTPUT

	while (1) {

		REG_PORT_OUTSET0 = LED; // Set PORT_PA27
		delay_s(1); // delay for 1 second
		REG_PORT_OUTCLR0 = LED; // clear the bit which sets PORT_PA27 to LOW
		delay_s(1); // delay for 1 second
	}
}
```

If you want to use option 1, directly accessing the OUT register, you can use the Bitwise AND Assignment Operator (&=) with a bitwise complement to flip the bit in the OUT register.

```
// example using option 1 Accessing the OUT Register directly
#include <asf.h>
#define LED PORT_PA27 // define LED as PORT_PA26 (0x08000000)

int main (void)
{
	system_init();
	delay_init(); // init delay
	REG_PORT_DIR0 |= LED; // Direction set to OUTPUT

	while (1) {

		REG_PORT_OUT0 |= LED; // Set PORT_PA27 to HIGH directly
		delay_s(1); // delay for 1 second
		REG_PORT_OUT0 &= ~(LED); // flip the bit which sets PORT_PA27 to LOW
		delay_s(1); // delay for 1 second
	}
}
```

Suppose you are working with a TFT and you are using ports PB00 thru PB15 to create a 16 bit bus. You can do something like this:

```
#define MYMASK 0x0000FFFF

void send_bus(uint16_t data){

   REG_PORT_OUTCLR0 = MYMASK; // clear the register
   REG_PORT_OUTSET0 = data; // set state of pin(s) to TRUE

}
```

GPIO Input

Configuring a pin as INPUT is a little bit more complicated then setting a pin output. You have the options to set a pin or pins as input but also configure the pin to response to pull–up or pull-down event. Please reference the SAMD21 or SAMR21 datasheet for advance configurations.

We will discuss how to configure a pin as a simple input. The pin will respond to a pull-up event (button pressed) and we will read the state of that pin via a polling method.

To configure the pin, we need to access the register via a structure pointer. Suppose we want to configure PORT_PA02 as input:

```
PORT->Group[0].PINCFG[2].reg = PORT_PINCFG_INEN | PORT_PINCFG_PULLEN;
```

Group[0] is PORTA and PINCFG[2] is pin PA02. The ‘reg’ is the register.

Suppose we wanted to configure PORT_PB00:

```
PORT->Group[1].PINCFG[0].reg = PORT_PINCFG_INEN | PORT_PINCFG_PULLEN;
```

Here is an example:

```
#include <asf.h>

int main (void)
{
	system_init();

	PORT->Group[0].PINCFG[2].reg = PORT_PINCFG_INEN | PORT_PINCFG_PULLEN; /* Set pin(s) to INPUT, PULL-UP */

	while (1) {
		/* Example to check pin state using PORT_PA02 as an example */
		if((PORT->Group[0].IN.reg & PORT_PA02) != 0){
			// do something
		}
	}
}
```

Alternately you can configure the PIN like so:

```
#include <asf.h>

int main (void)
{
	system_init();

	PORT->Group[0].PINCFG[2].bit.INEN = 1;
	PORT->Group[0].PINCFG[2].bit.PULLEN = 1;

	while (1) {

		if((PORT->Group[0].IN.reg & PORT_PA02) != 0){
			// do something
		}
	}
}
```

Putting it all together, here is an example on how to blink a LED with the press of a button:

```
#include <asf.h>
#define LED PORT_PA27

int main (void)
{
	system_init();
	delay_init();
	PORT->Group[0].PINCFG[2].bit.INEN = 1;
	PORT->Group[0].PINCFG[2].bit.PULLEN = 1;

	REG_PORT_DIRSET0 = LED; // Direction set to OUTPUT
	while (1) {

		if((PORT->Group[0].IN.reg & PORT_PA02) != 0){

			REG_PORT_OUTTGL0 = LED; // toggle led PORT_PA27
			delay_ms(750); // crude debounce
		}
	}
}
```

You can also read the state of a pin using the REG_PORT_IN register. For example:

```
#include <asf.h>
#define LED PORT_PA27
int main (void)
{
	system_init();
	delay_init();
	PORT->Group[0].PINCFG[2].bit.INEN = 1;
	PORT->Group[0].PINCFG[2].bit.PULLEN = 1;

	REG_PORT_DIRSET0 = LED; // Direction set to OUTPUT
	while (1) {
		if((REG_PORT_IN0 & PORT_PA02) != 0){ // using REG_PORT_IN register
			REG_PORT_OUTTGL0 = LED;
			delay_ms(750); // crude debounce
		}
	}
}
```

You can read more about the SAMD21’s GPIO registers. Please refer to section 23. PORT - I/O Pin Controller of the datasheet.