Aerospace & Defence | Lighting | Power Management | Internet of Things | Transportation

Components – EMEA

Engineering DACH

Lab 1

Quang Hai Nguyen

Revision 0.99, 07.01.2020

SAMD21 workshop

Five Years Out

Revision History

Revision, Date	Editor	Subject (major changes)
Revision 0.99, 07.01.2020	Quang Hai Nguyen	Preliminary

Table of Contents

List of Figures

No table of figures entries found.

List of Icon Identifiers

Table 1: Icon Identifiers List

	Extra information about a topic
	Task needs to be done
	Important information or a warning
	Result expected to see

Prerequisite

Hardware

- Laptop or PC with Windows 7 or Windows 10
- SAMD21-Xplained-Pro evaluation board (provided during the workshop)
- Extension boards, LEDs, Potential meter, buttons and jumper wires (provided during the workshop)
- Micro USB cable (provided during the workshop)
- Breadboard (provided during the workshop)
- **Internet access without proxy or firewall**

Software

- Terminal program (e.g. TeraTerm)
- Atmel Studio 7 (link)

Labs description

Description

Assignment

1. GPIO Lab
   - Blinky LED
   - Button with polling
   - Button with interrupt
2. UART Lab
   - Setup UART and print text on the terminal program
3. Deep dive into generated code

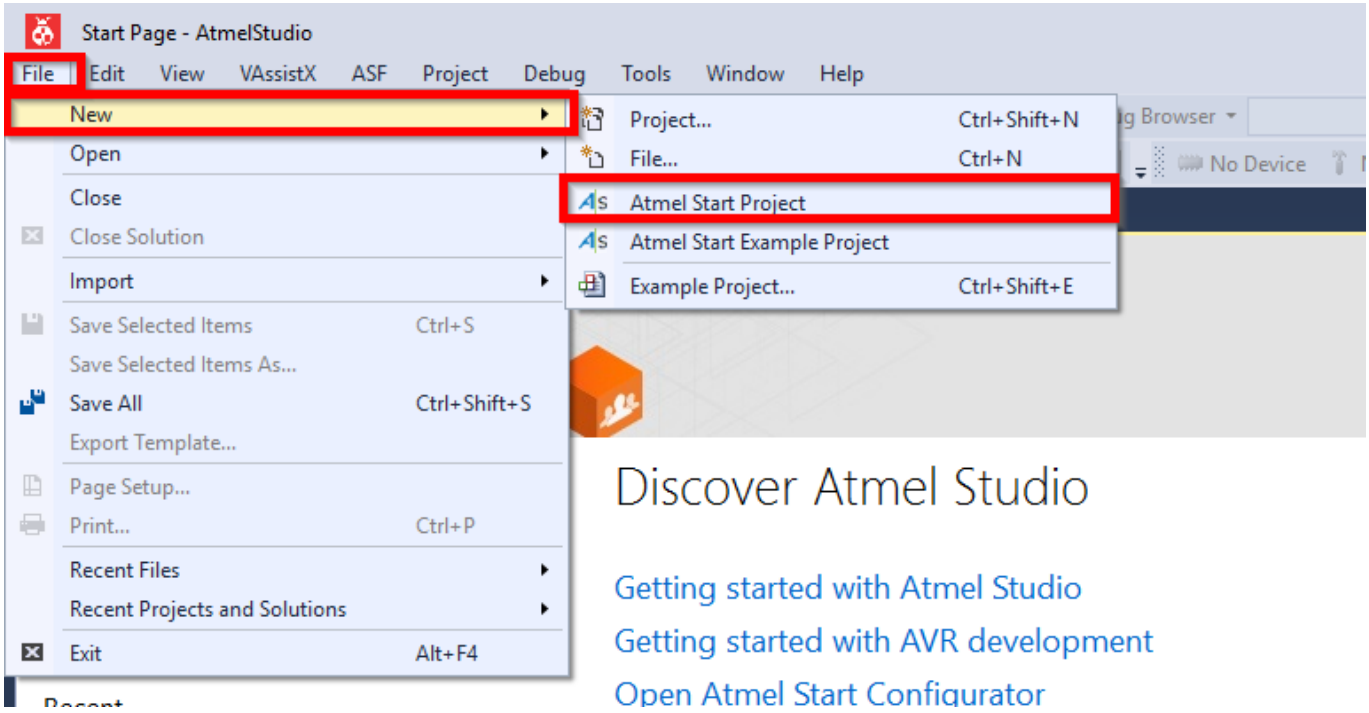
GPIO Lab

Blinky LED

In this Lab, we will use Atmel Start to configure the Pin as GPIO and to generate the project files.

Using Atmel Start to generate the code
Start Atmel Studio.

In Atmel Studio, click File → New → Atmel Start Project

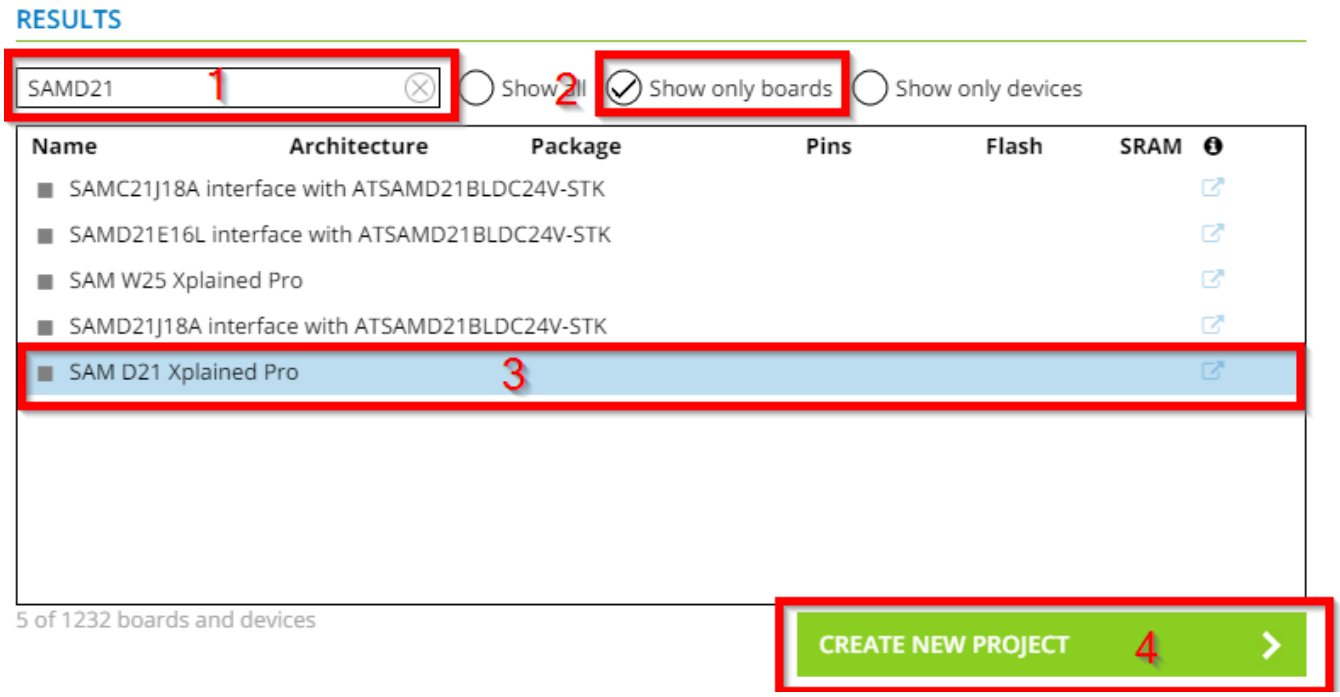


Atmel Start will be prompted shortly afterward

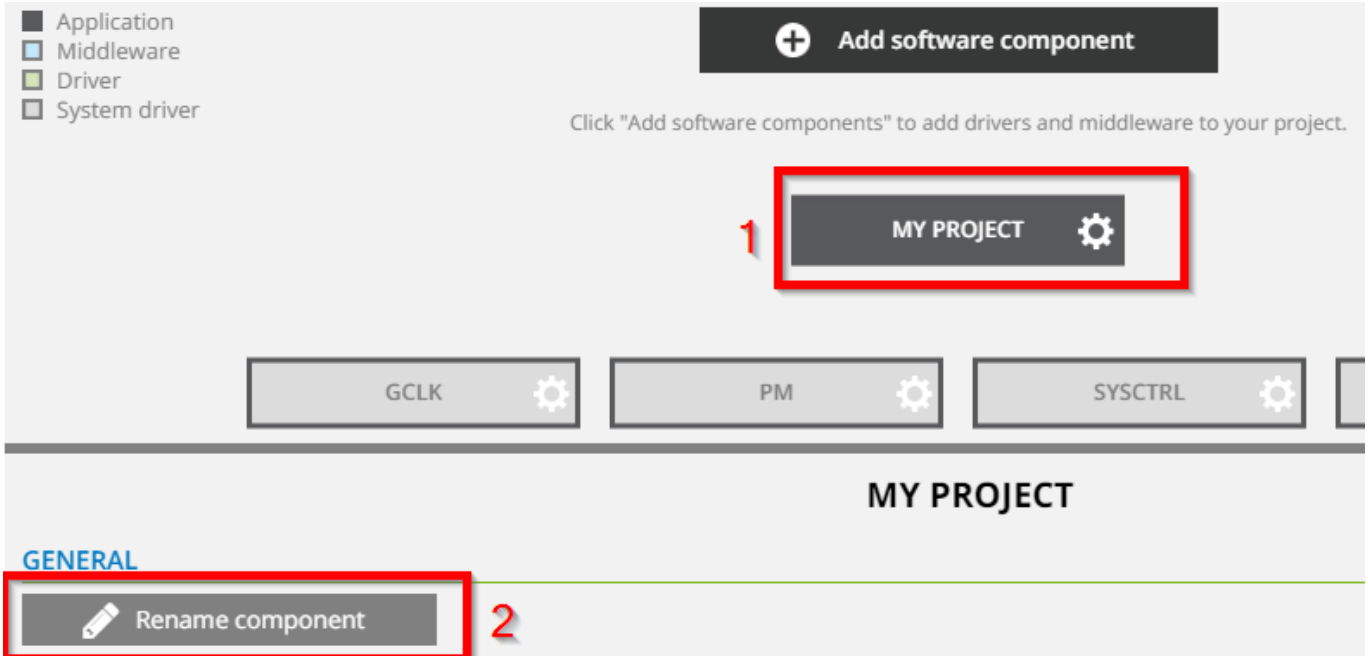
Please note that Atmel Start is an online tool. Therefore, we need internet access to use it.

In Atmel Start, we tick on the option “Show only boards”, enter the text “SAMD21” in the filter text box and choose “SAM D21 Xplained Pro”.

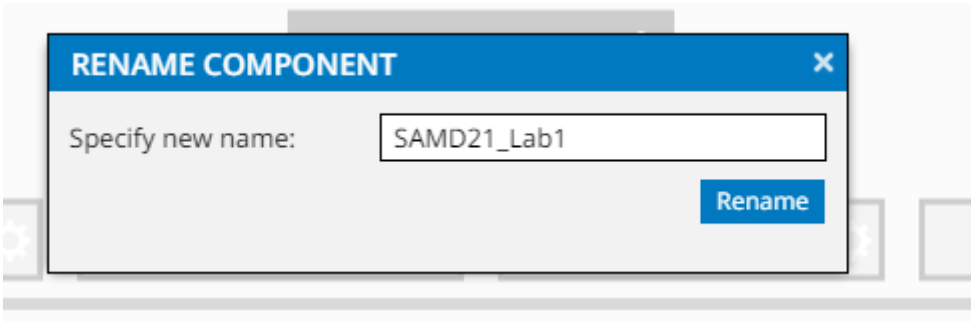
After that, we click on “CREATE NEW PROJECT” to start configuring our peripheral.



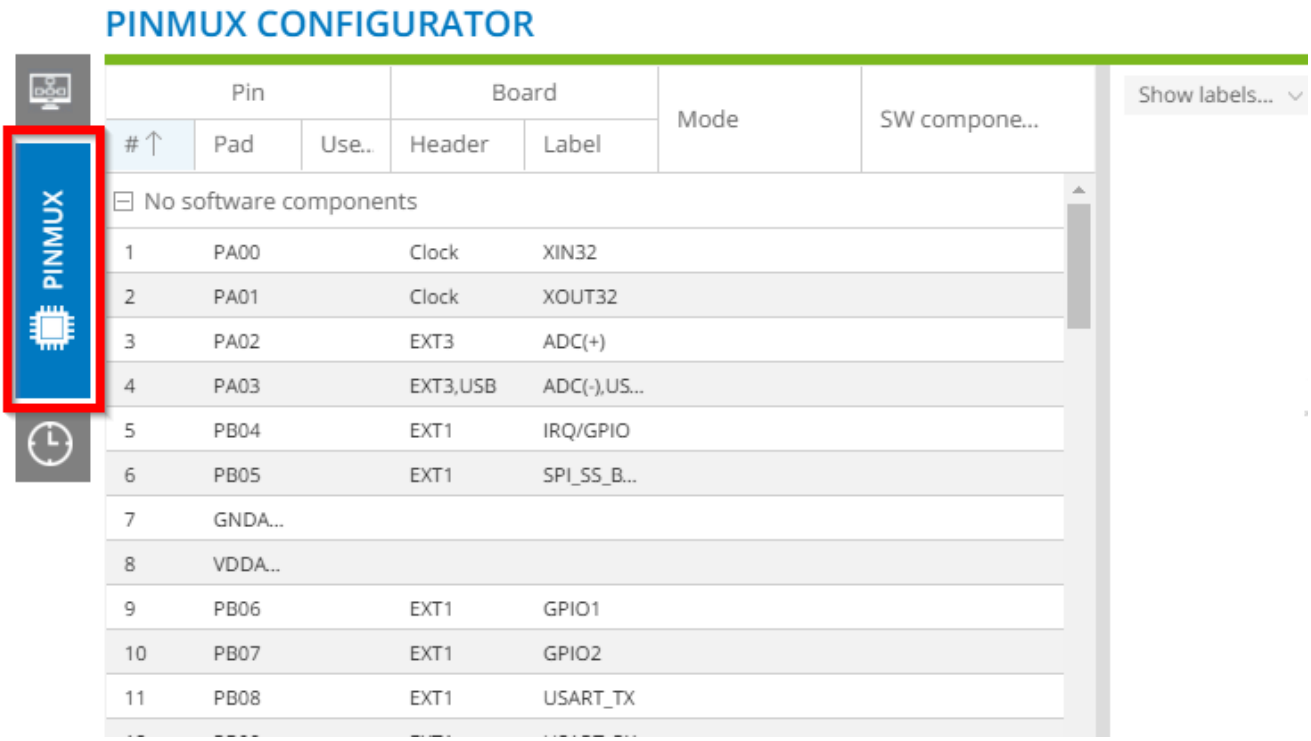
The first thing must be done is giving your project a meaningful name. To do so, please click “MY PROJECT” button and click “Rename component”



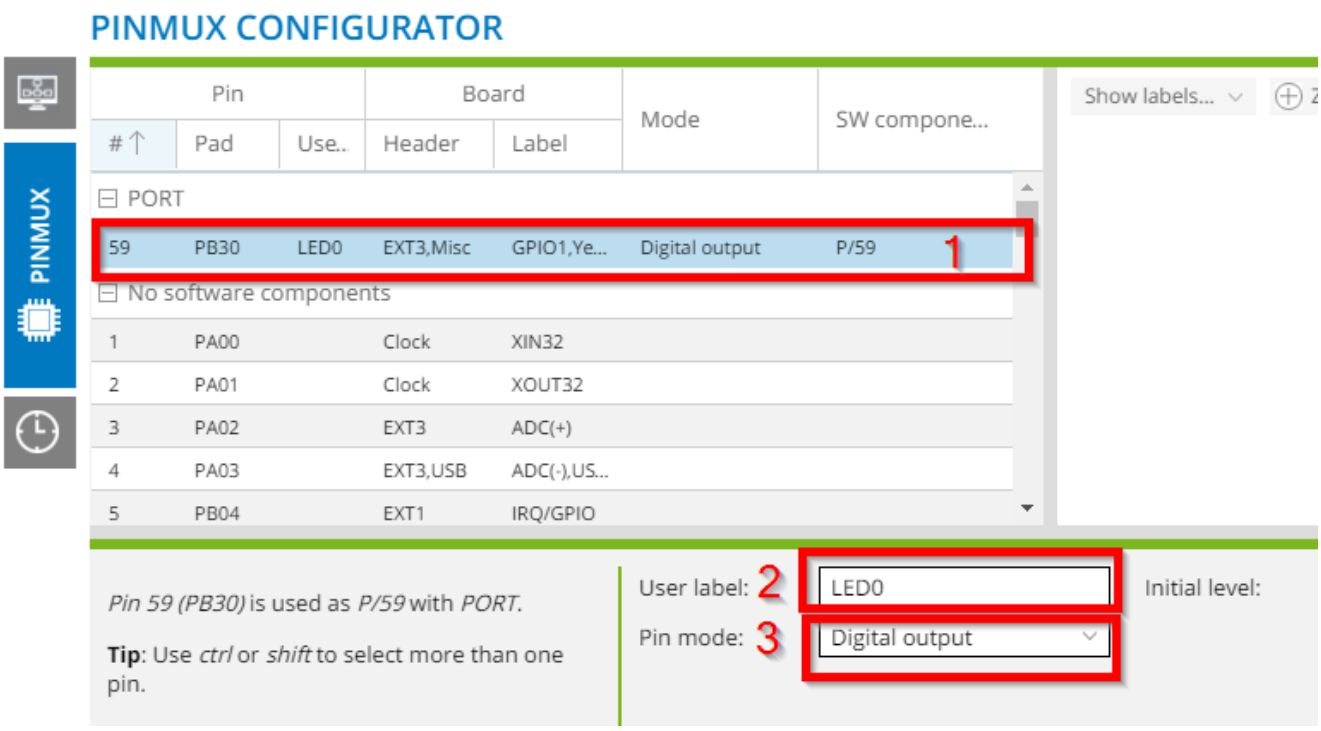
Give your project a name. For my case, I will go with “SAMD21_Lab1”



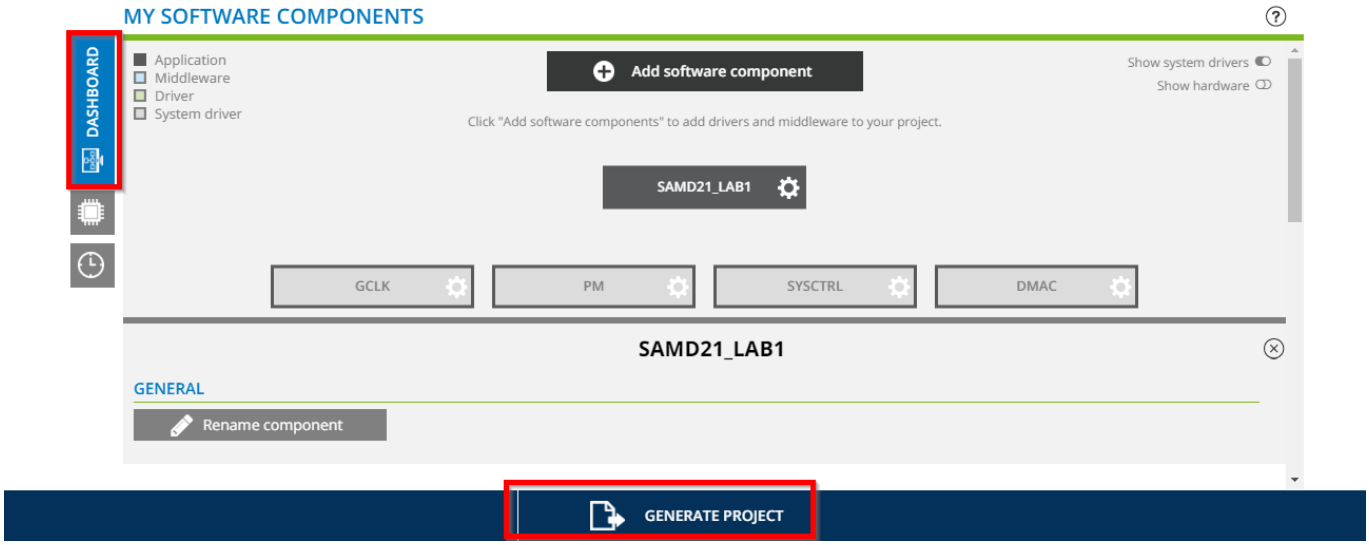
We move to “PINMUX” tab to set up the pin



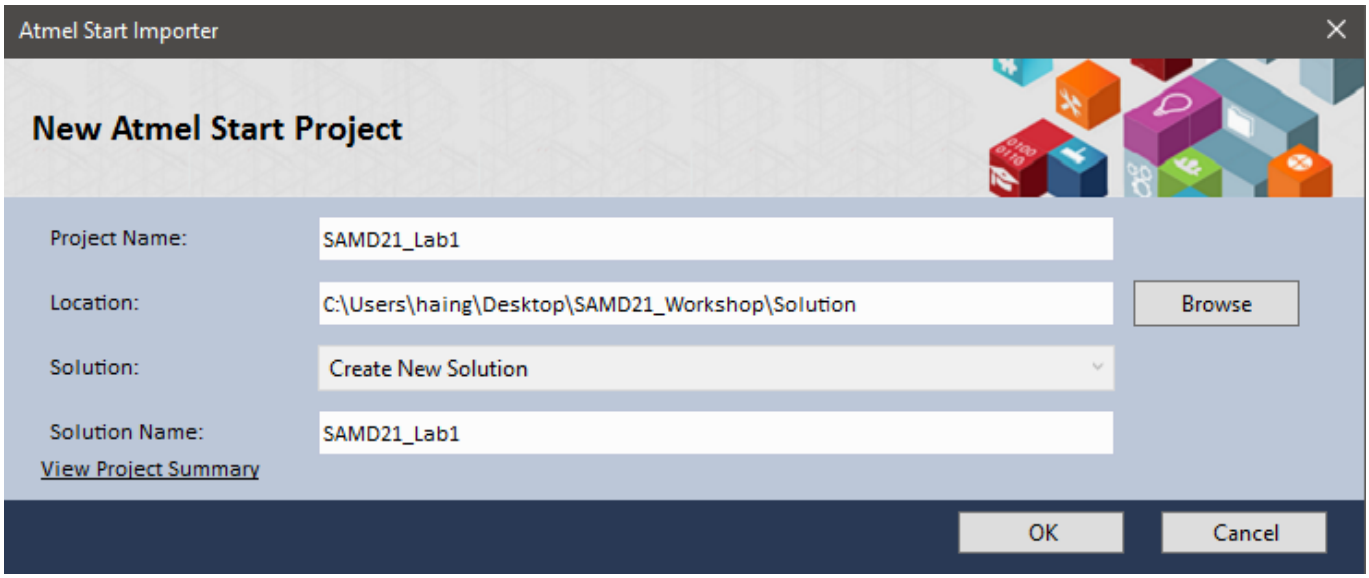
Accord to the evaluation board user manual, the LED is located on Pin PB30. Therefore, we look for pin PB30 in the list, give it an intuitive name, and set it as a digital output.



That's it pretty much everything. We go back to "DASHBOARD" tab and click "GENERATE PROJECT".



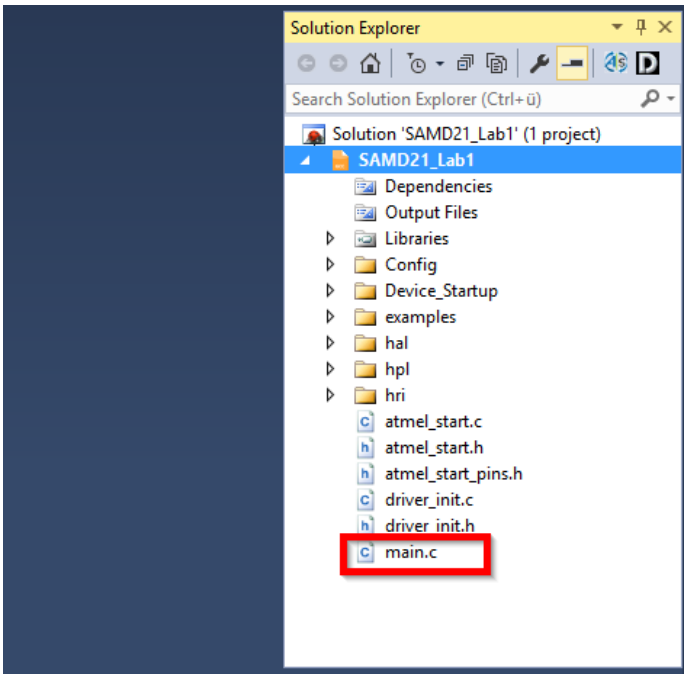
After clicking "GENERATE PROJECT", Atmel Start prompts a new message to ask you where you want to save your project. Please choose a place to store the project and then click "OK"



Please make sure that your project path is not too long and contains no special character

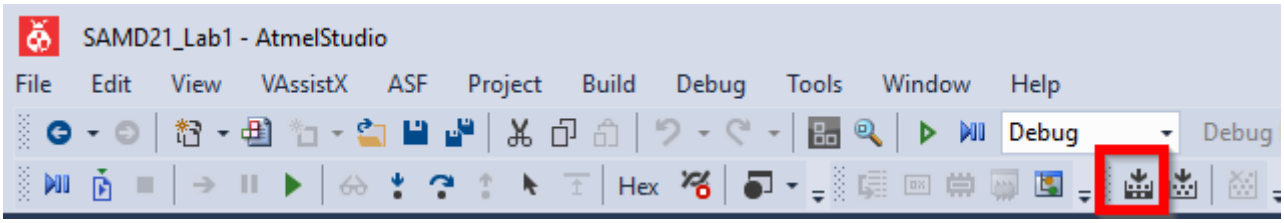
Programming blinky LED application

After the project is generated, navigate to main.c file



File main.c is the entry of your application, it is where you will write your code. Before adding anything to the project, we firstly build it to make sure there is no error in the code generation process.

Click on the “Build” button to start to build the project



Alternatively, we can click Build → Build Solution or F7 to build the project

At this point, you should not get any error from the project. If the error shows up, please use Atmel Start to re-generate the project.

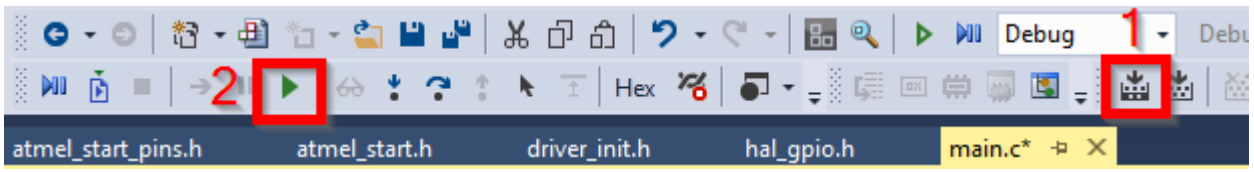
The next task is looking for the GPIO toggle function to put it in the main loop. Since we are working with HAL (Hardware Abstraction Layer) and GPIO library, so we will look for the function name in the file “hal/include/hal_gpio.h”.

In hal_gpio.h file, we will find the function gpio_toggle_pin_level(const uint8_t pin), which is a perfect fit for our purpose. The only missing argument is the pin. If you still remember, we have labeled our pin as LED0. The “LED0” is our parameter to pass into the gpio_toggle_pin_level function. Those pin definitions can be found in atmel_start_pins.h file.

The code block for blinking LED is following

```
while (1) {
        gpio_toggle_pin_level(LED0);
        delay_ms(1000)
}
```

Build the project and program it into the board.



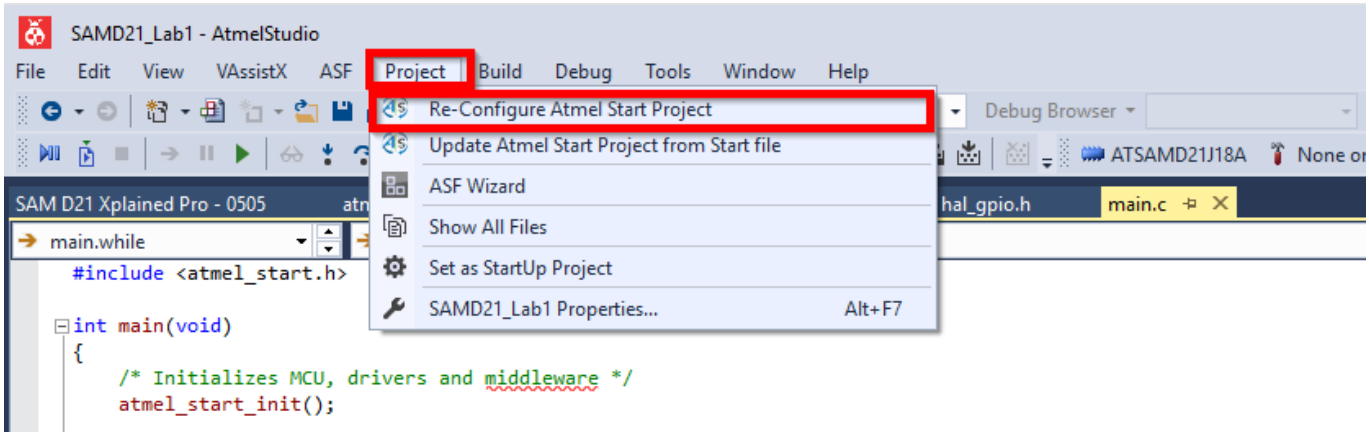
You should see LED0 blinking

Button with polling

Using Atmel Start to add more peripheral to the project

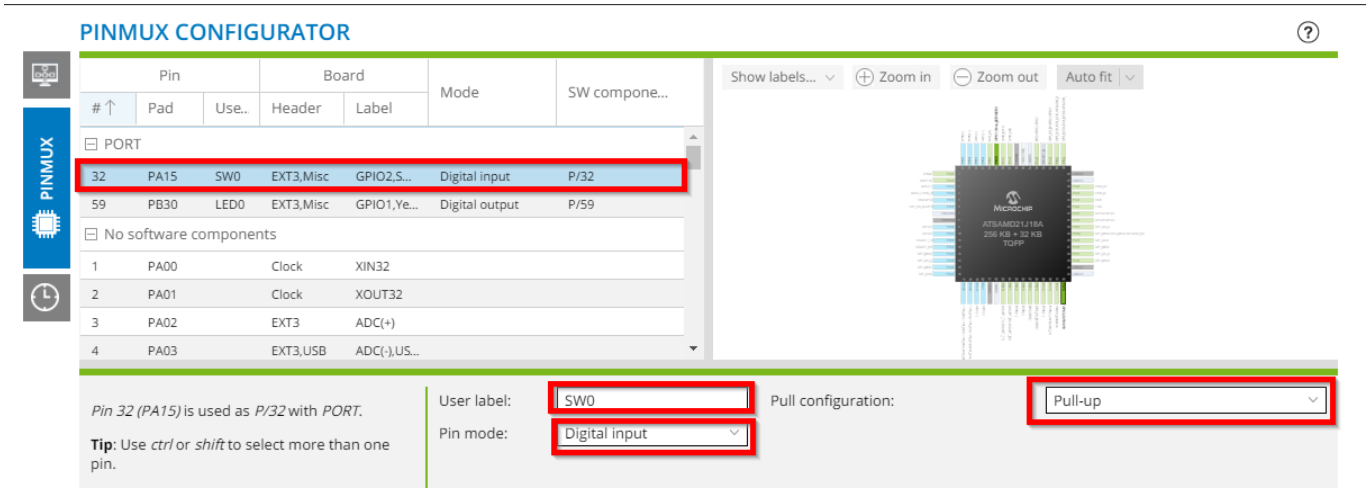
In this Lab, we will use Atmel Start to add a button to our project

To re.start Atmel Start again, click Project → Re-Configure Atmel Start Project

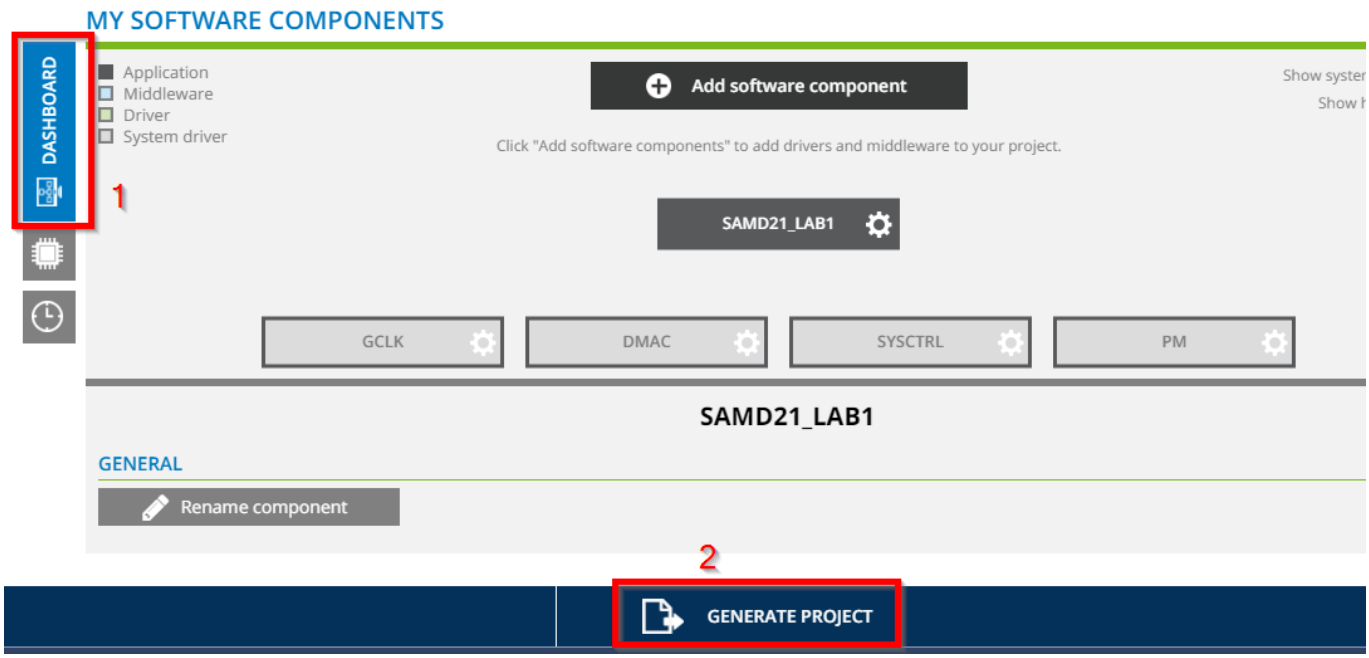


To add a button (pin as digital input) to our project, the procedure is more or less the same as previous steps.

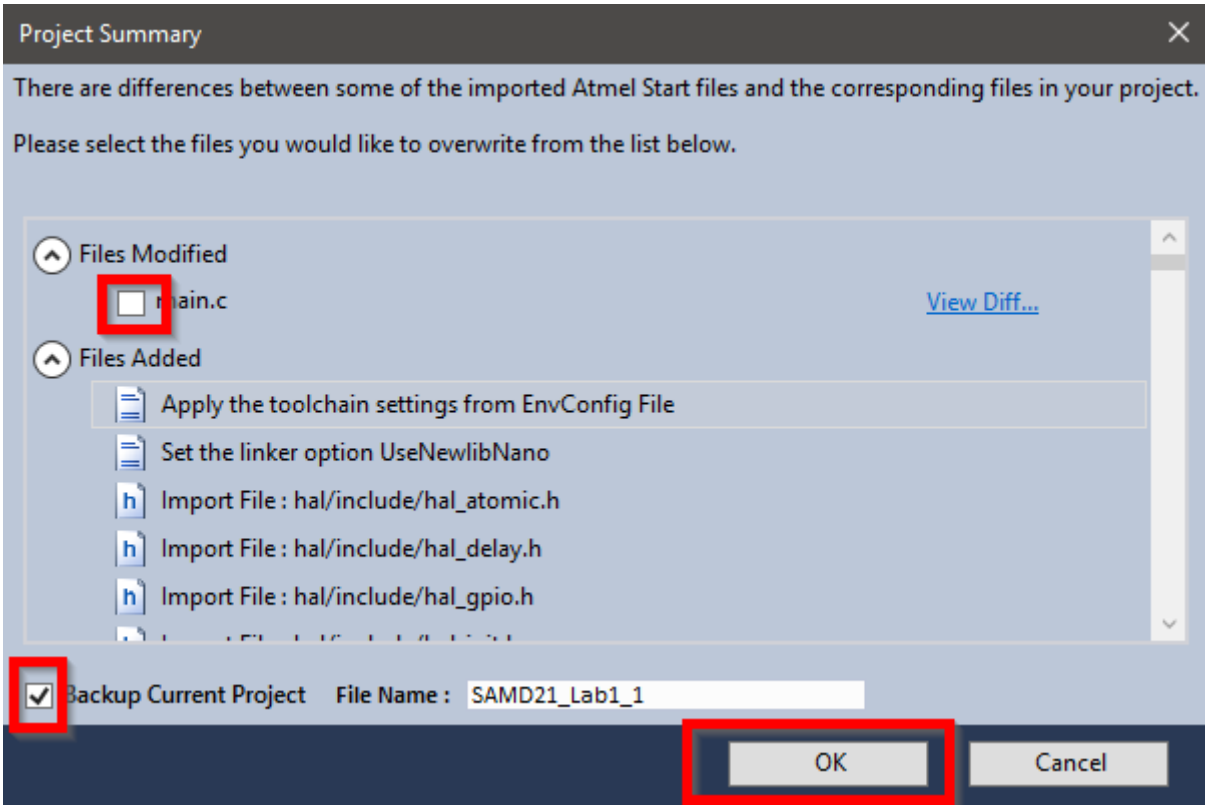
Go to PINMUX tab, locate pin PA15 (according to the datasheet, the button is connected to this pin), label the pin with an intuitive name and set it as Digital Input



Go back to “DASHBOARD” tab and click “GENERATE PROJECT”



After clicking “GENERATE PROJECT”, Atmel Start shows us a list of modified and added files. If you want to back up your previous project setup, click “Backup Current Project”. Otherwise, click “OK” to process to the next step.



Programming button application

Since we didn’t choose the option to modify the main.c file, this file remains the same.

Build the code to verify there are no errors during the generation process.

You should not get any error at this point. If the errors happen indeed, please re-start Atmel Start and re-generate the project again.

Comment out the blinky LED code since we do not need it

```
while (1) {
#if 0 //Blinky LED Lab
        gpio_toggle_pin_level(LED0);
        delay_ms(1000);
#endif
}
```

For this application, we simply want to switch on the LED when the button is pressed and vice versa. Therefore we check again the file “hal_gpio.h” for the corresponding functions.

The two functions that required are

- gpio_get_pin_level(const uint8_t pin)
- gpio_set_pin_level(const uint8_t pin, const bool level)

Therefore, the code block is following:

```
while (1) {
#if 0 //Blinky LED Lab
        gpio_toggle_pin_level(LED0);
        delay_ms(1000);
#endif
        if(gpio_get_pin_level(SW0) == false){
                // Button pressed, turn LED on
                gpio_set_pin_level(LED0, false);
        }
        else{
                // vice versa
                gpio_set_pin_level(LED0, true);
        }
}
```

Build the project and program the binary into the board.

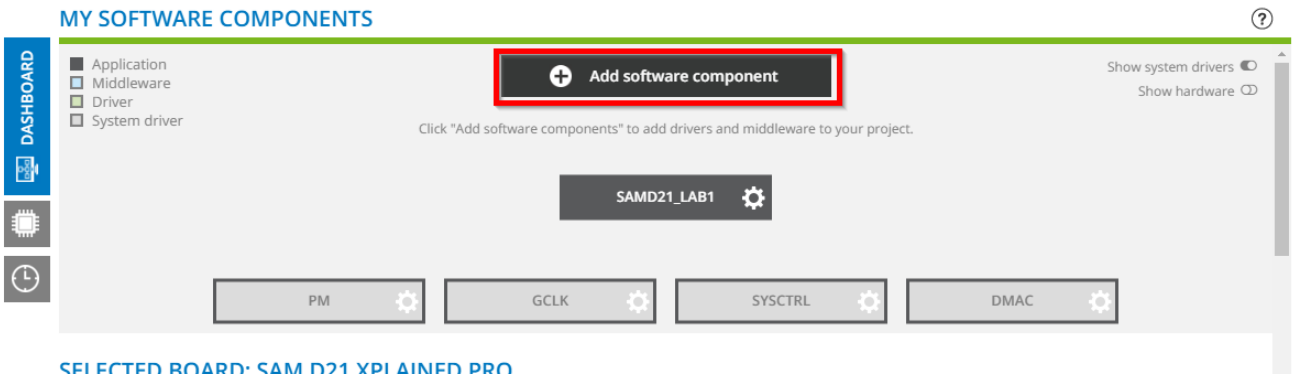
You should be able to turn on-off the LED via the button

Button with interrupt

Using Atmel Start to add more peripheral to the project

The next thing we want to do is using the button with interrupt instead of polling.

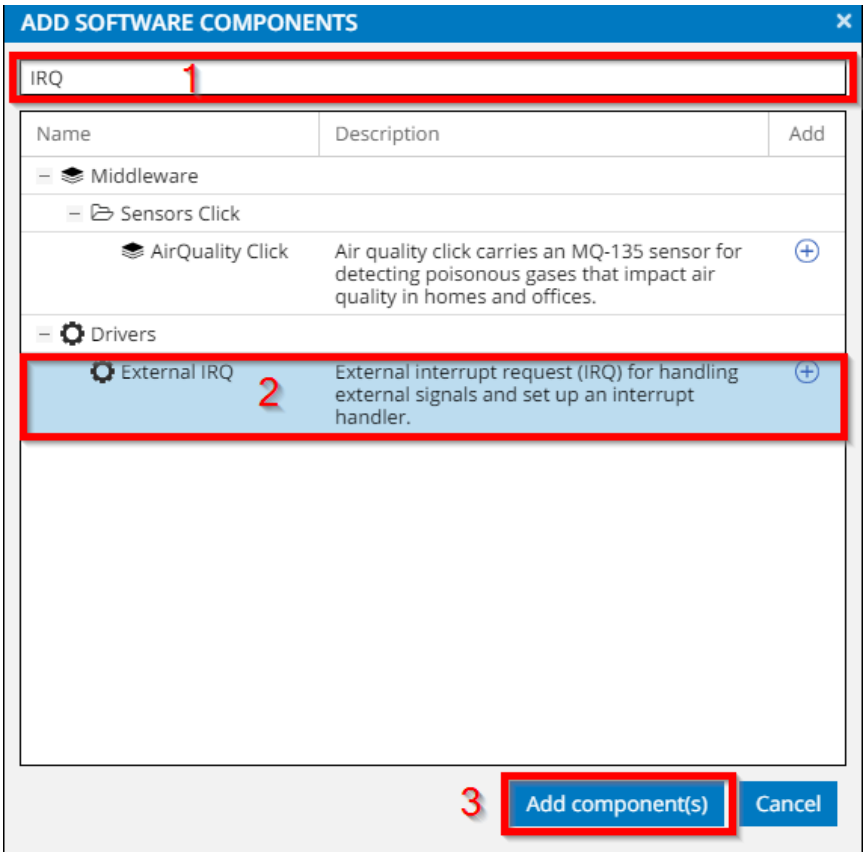
Re-start Atmel Start. In Atmel Start click “Add software component”



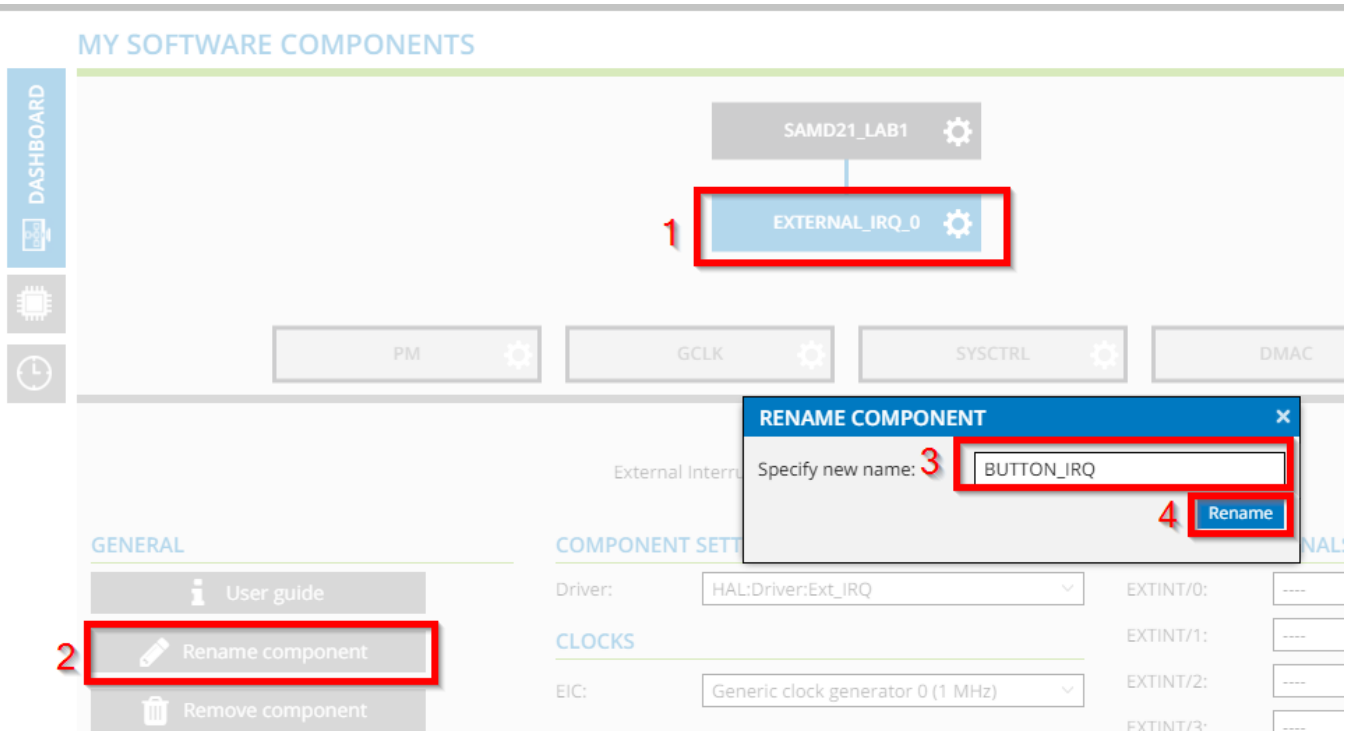
This option allows us to add software components, e.g. peripheral driver, middlewares, third-party libraries into our project.

Since we want to work with Interrupt, type “IRQ” into the Filter text box, choose “External IRQ” and click “Add Component(s)”.

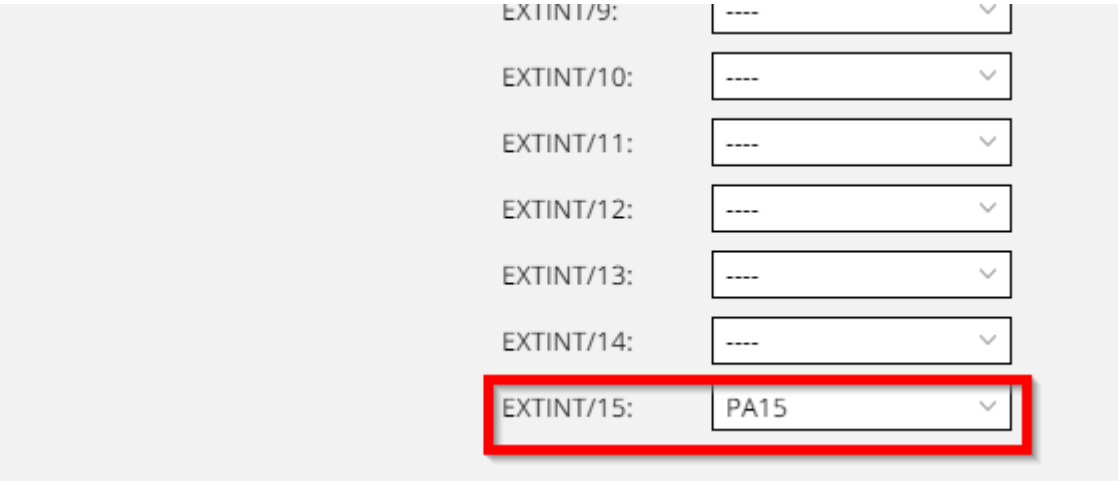
External IRQ enables us to handle the external signal and setup interrupt handlers.



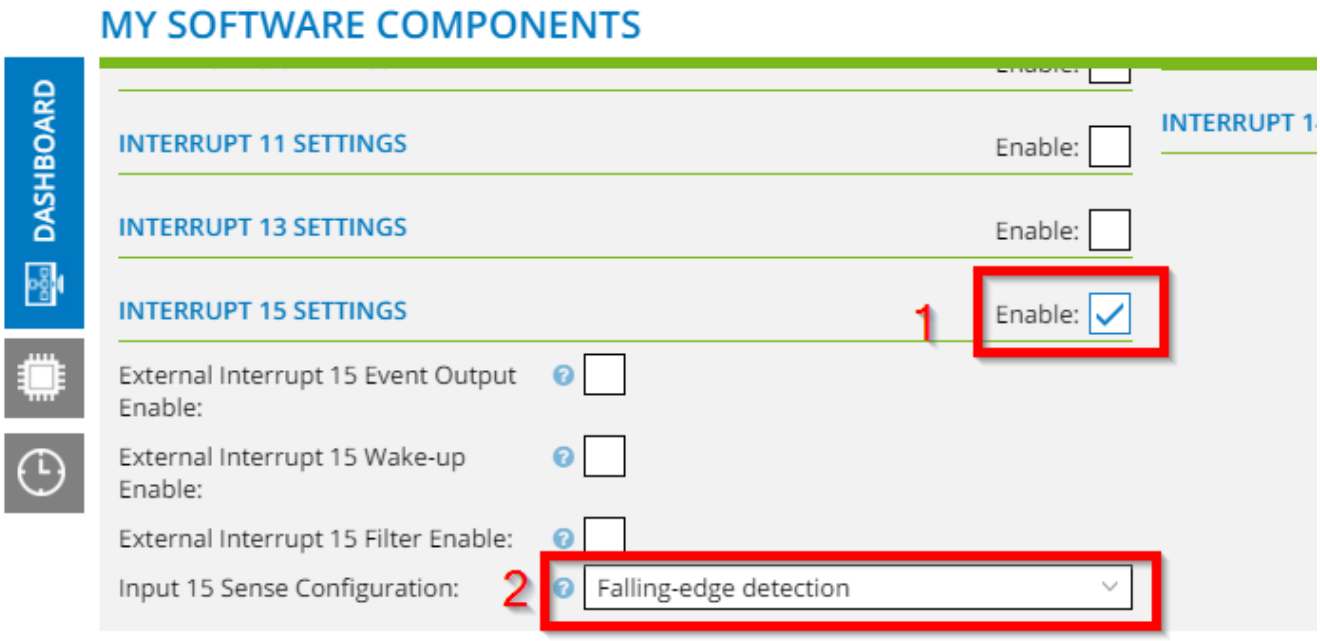
Next, we will give out component a name. Please click “EXTERNAL_IRQ_0” and choose “Rename component”. Give your component a name, for my case, it is “BUTTON_IRQ”



Next, we will configure the external IRQ. Click the external IRQ again and scroll down to the setting. According to the datasheet, PA15 (button pin) is connected to External line 15. Therefore, we configure EXTINT/15 as following:



Then we scroll down to interrupt setting, enable “INTERRUPT 15 SETTINGS”, and set “Input 15 Sense Configuration” to “Falling-edge detection”



Generate the code and re-build the project for error verification.

Developing button application with interrupt

Comment out the code from the previous lab since we are now using an interrupt to handle the signal from the button.

To work with interrupts, we have to create an interrupt callback handler, then we register it to the corresponding interrupt source (pin PA15). To do so, initialize the following function on top of the main()

```
static void button_pressed_interrupt_handler(void)
{
	gpio_toggle_pin_level(LED0);
}
```

The above function simply toggles the LED0. Then we have to register it.In the main(), below atmel_start_init(), at the following line:

```
ext_irq_register(PA15, button_pressed_interrupt_handler);
```

There are examples of how to use generated peripherals or libraries which are located in “examples/driver_examples.c” and “examples/driver_examples.h”

In the end, this is our main.c file

```
#include <atmel_start.h>

static void button_pressed_interrupt_handler(void)
{
	gpio_toggle_pin_level(LED0);
}

int main(void)
{

	/* Initializes MCU, drivers and middleware */
	atmel_start_init();
	ext_irq_register(PA15, button_pressed_interrupt_handler);

	/* Replace with your application code */
	while (1) {

#if 0 //Blinky LED Lab
		gpio_toggle_pin_level(LED0);
		delay_ms(1000);
#endif

#if 0 // Polling Button Lab
		if(gpio_get_pin_level(SW0) == false){
			// Button pressed, turn LED on
			gpio_set_pin_level(LED0, false);
		}
		else{
			// vice versa
			gpio_set_pin_level(LED0, true);
		}
#endif

	}
}
```

Compile the project and program the binary into the evaluation board

You should be able to turn on-off the LED via the button

Did you notice that the performance of the button is…not so great? In the next step, we will investigate what wrong with our interrupt code.

We would like to investigate the code generated by Atmel Start in order to see how the interrupt has been configured. To do so, navigate to file “driver_init.c”. This file contains the code generated by Atmel Start based on the HAL library.

In the function BUTTON_IRQ_init(void), pin PA15 is configured as GPIO_PULL_OFF by Atmel Start, which is not correct in our case:

```
void BUTTON_IRQ_init(void)
{
	_gclk_enable_channel(EIC_GCLK_ID, CONF_GCLK_EIC_SRC);

	// Set pin direction to input
	gpio_set_pin_direction(PA15, GPIO_DIRECTION_IN);

	gpio_set_pin_pull_mode(PA15,
			// <y> Pull configuration
			// <id> pad_pull_config
			// <GPIO_PULL_OFF"> Off
			// <GPIO_PULL_UP"> Pull-up
			// <GPIO_PULL_DOWN"> Pull-down
			GPIO_PULL_OFF);

	gpio_set_pin_function(PA15, PINMUX_PA15A_EIC_EXTINT15);

	ext_irq_init();
}
```

Therefore, we have to modify the code manually by changing from GPIO_PULL_OFF to GPIO_PULL_UP. In conclusion, the pin PA15 is modified as following:

```
gpio_set_pin_pull_mode(PA15,
		// <y> Pull configuration
		// <id> pad_pull_config
		// <GPIO_PULL_OFF"> Off
		// <GPIO_PULL_UP"> Pull-up
		// <GPIO_PULL_DOWN"> Pull-down
		GPIO_PULL_UP);
```

Compile and run the application again. The button’s behavior is now working properly.

UART Lab

In this Lab, along toggling the LED when the button is pressed, we also want to print the message “button pressed” on the terminal program

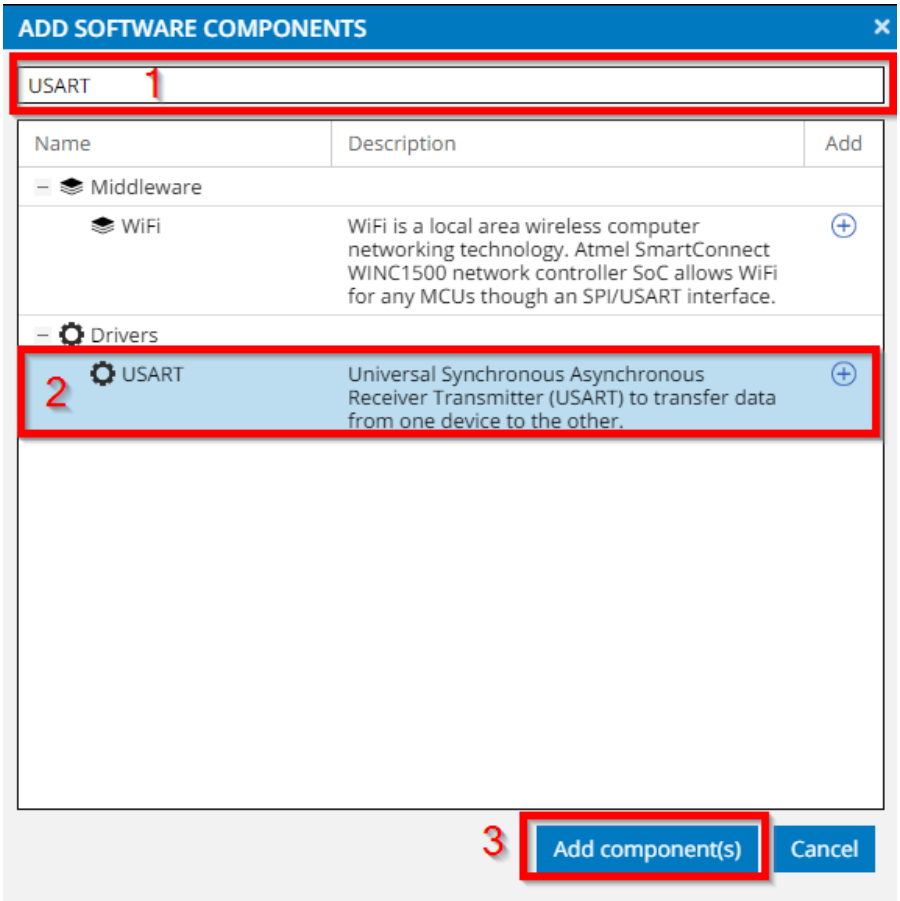
To print the message on the terminal program, we must know how to initialize USART peripheral and which pins are routed to the virtual comport pin (according to the user manual, they are pin PA22 and PA23).

Configure USART with Atmel Start

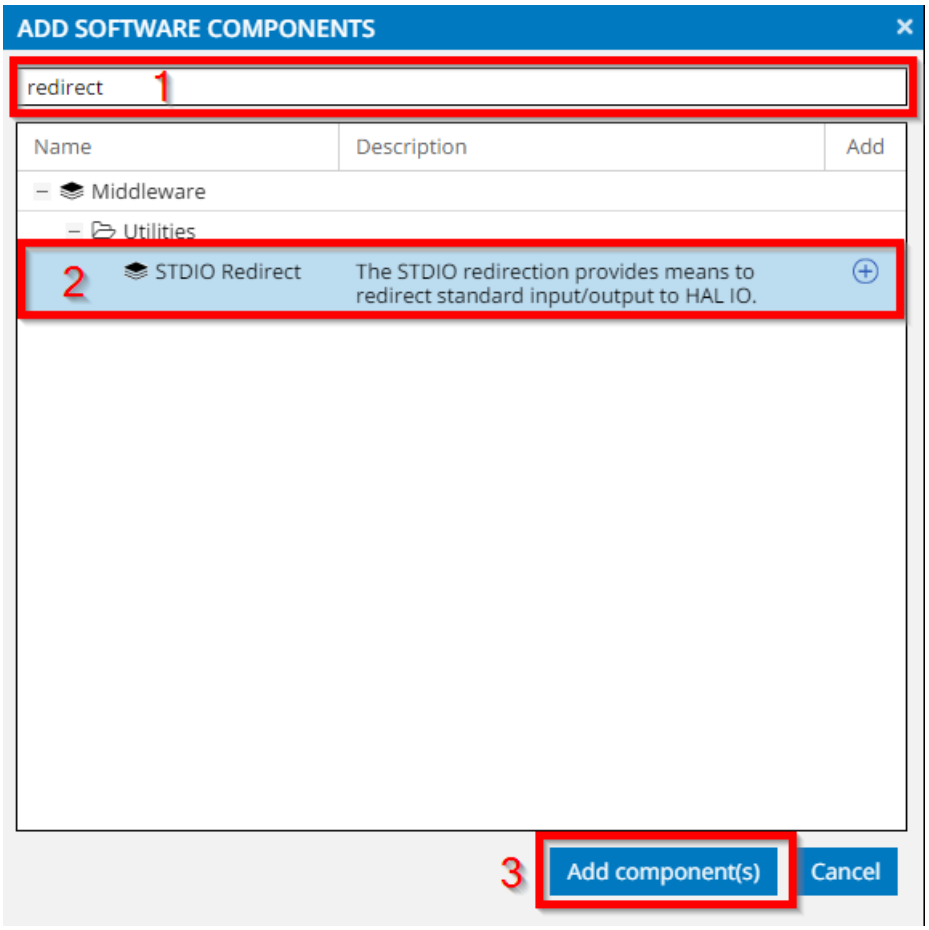
Re-start Atmel Start again.

In Atmel Start, click “Add software component”

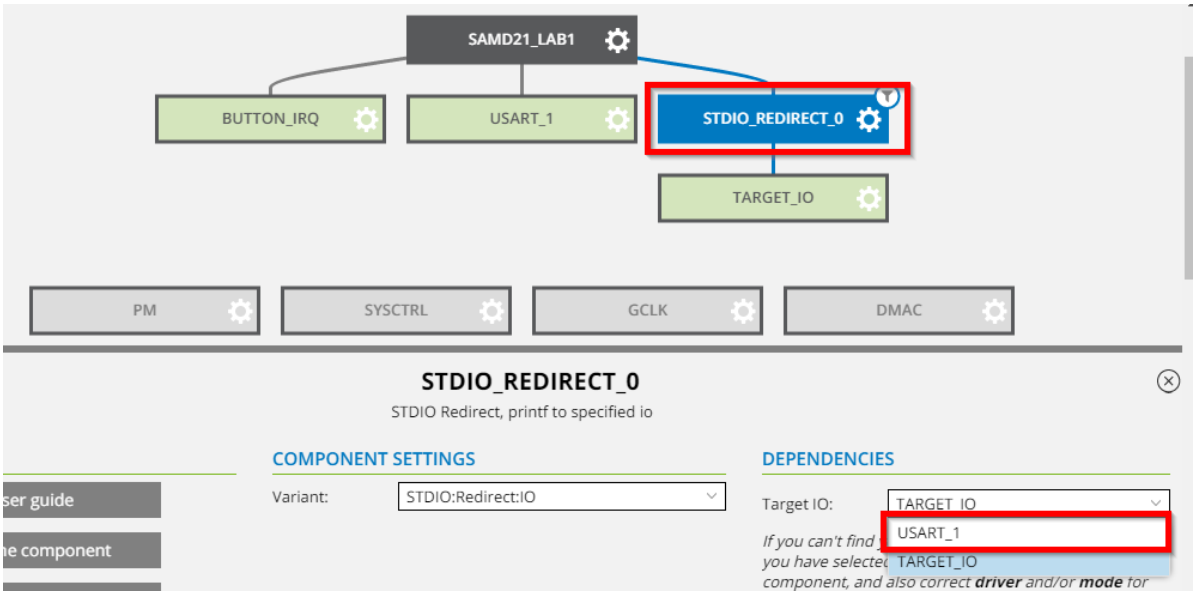
Type „USART“ in the filter text box, choose USART and click “Add component(s)”



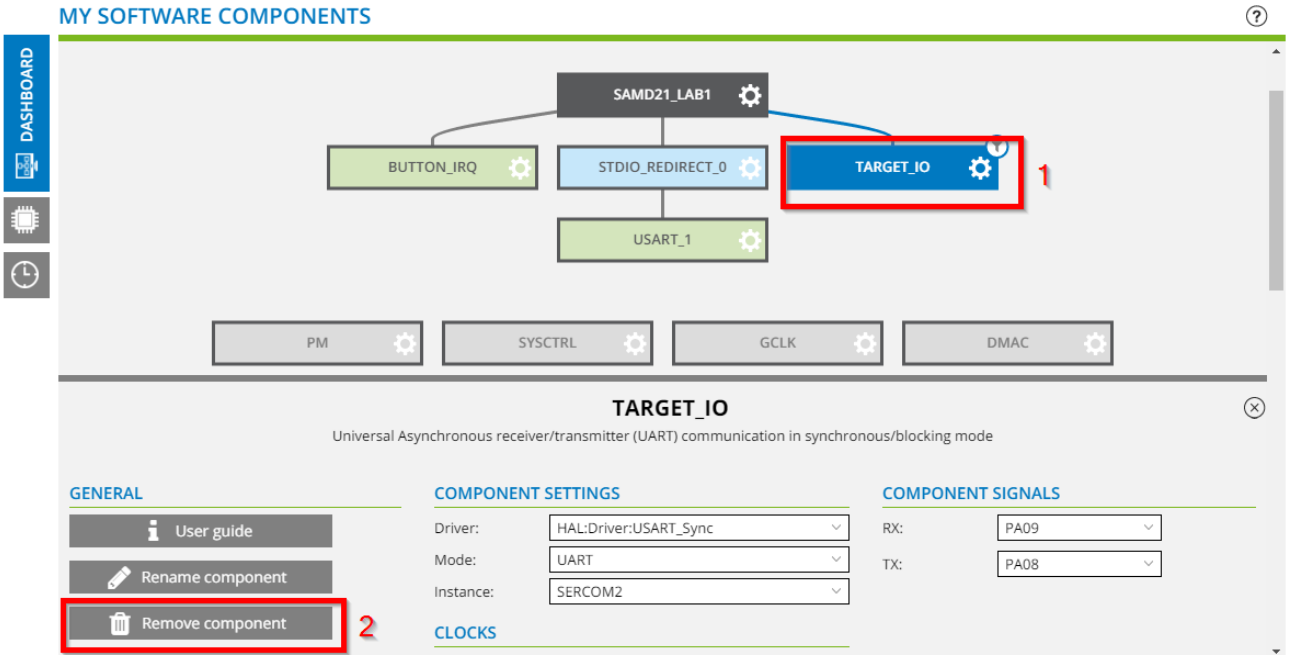
We also want to redirect printf() to USART, so we need another software component. Click “Add software component” again. In this case we type “redirect” into the filter text box and choose “STDIO Redirect”.



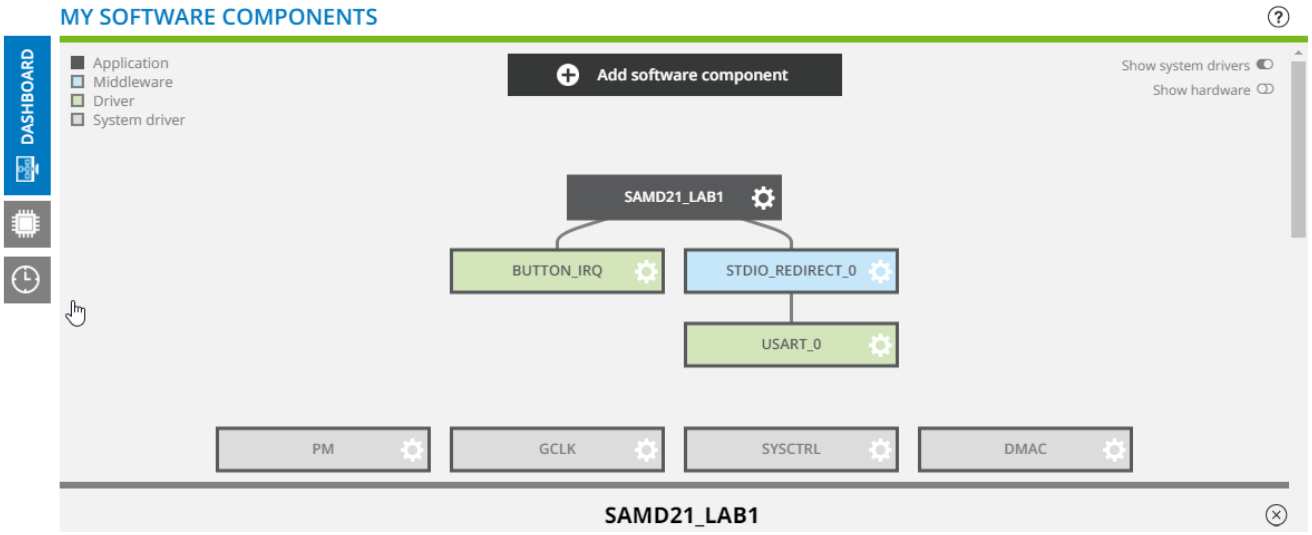
Next step, we will route the STDIO redirect to USART by doing:



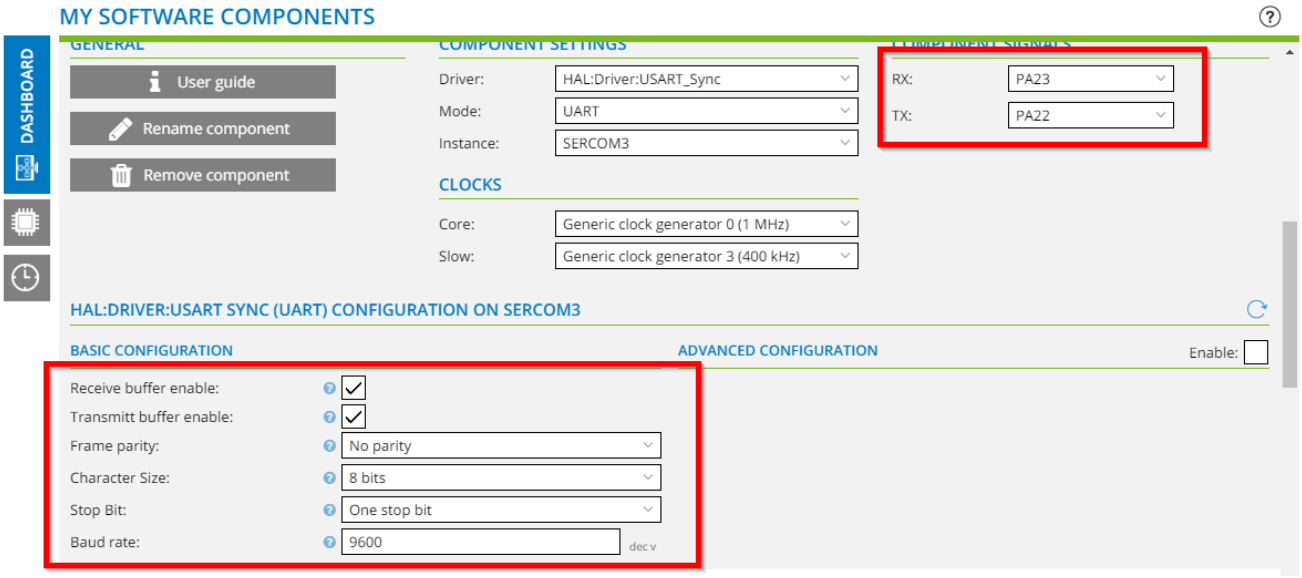
We don’t need the component “TARGET IO” so we will remove it:



This is what we have expected in Atmel Start:



The last task must be done here is assigning the correct pins for USART. From the user manual, we know that pin PA22 and PA23 are connected to TX and RX pin of the virtual comport. Therefore the pin configuration is following:



We also use the default value for baud rate, stop bit, parity, character size.

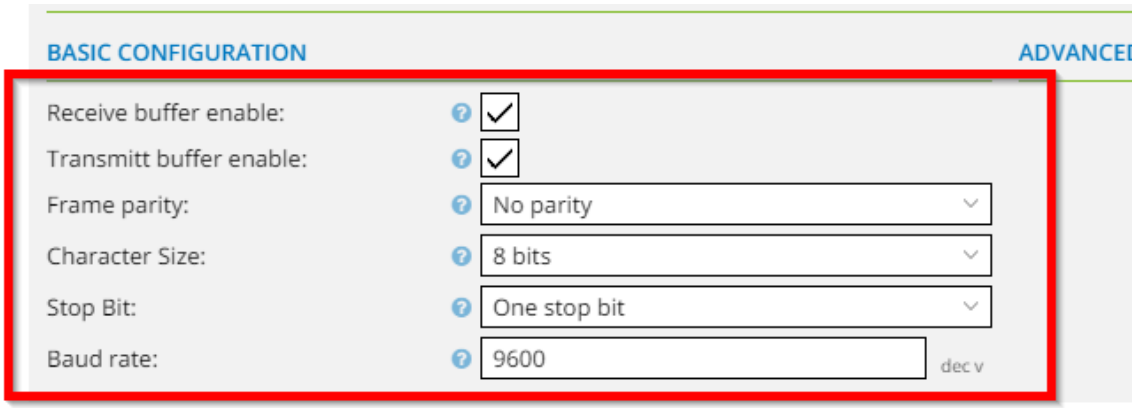
Generate the code again.

Adding USART code to the application

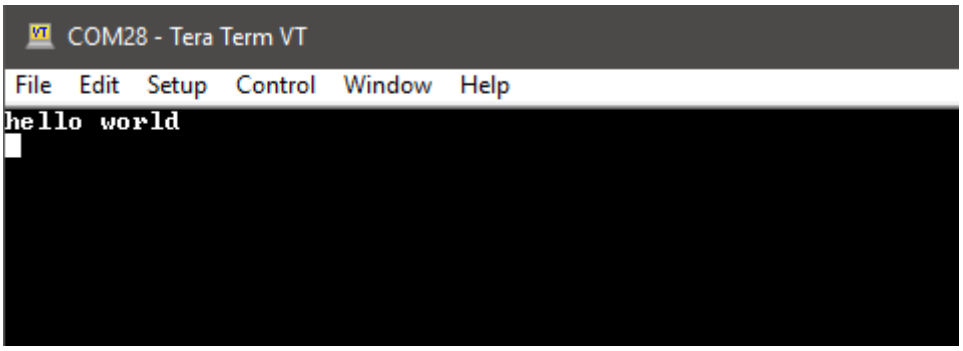
We can test right away if the printf() is working by adding the following line to the application, below atmel_start_init() function

```
printf("hello world\n")
```

Setup the terminal program with the following setup:

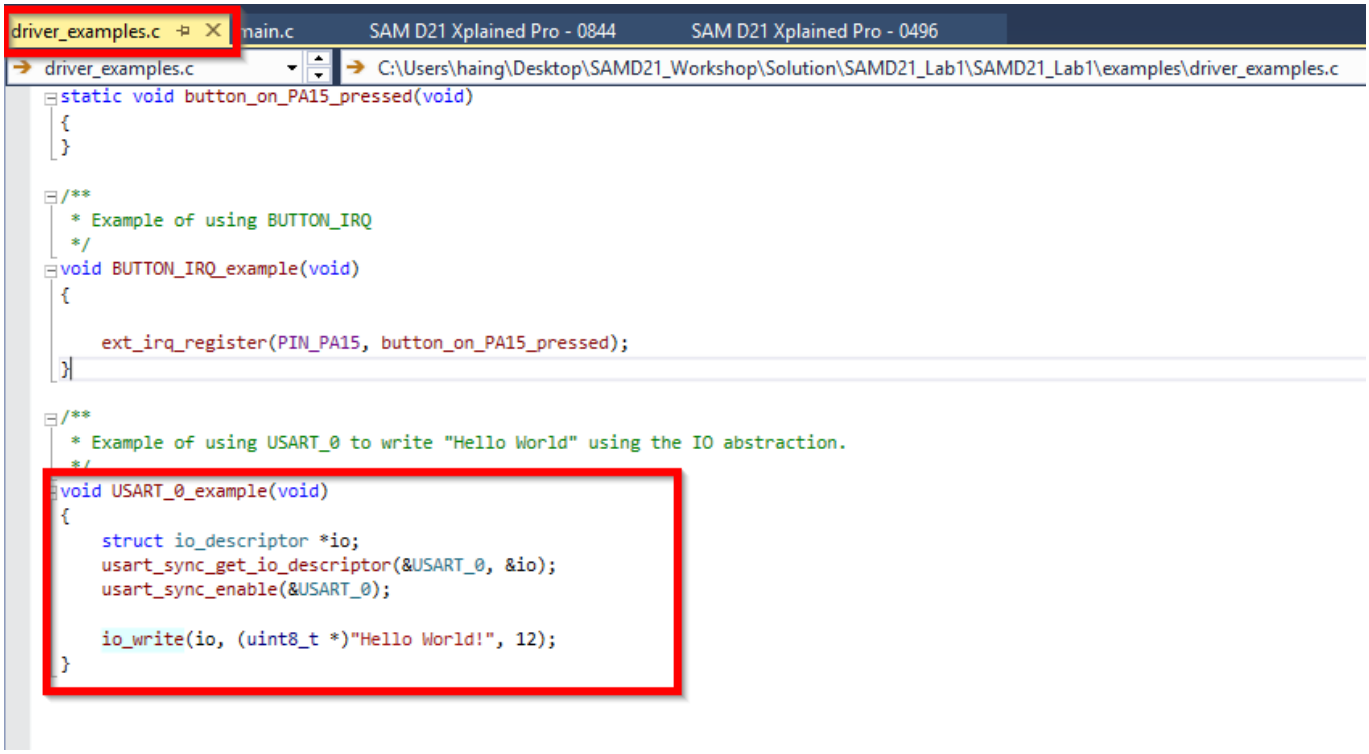


Build and Run the application. This is what you should get on the terminal:



We also want to print the message whenever the button is pressed. Since printf() is considered as a resource-expensive function so it is not a good idea to use it inside the interrupt function. Therefore we have to apply the native usart write function.

To look for this kind of implementation, the quickest way is to have a look at the file ”examples/driver_examples.c”



The code snippet above shows us how to use the USART function to transmit data on the USART bus. Let’s jump back to our main.c file and start to implement it.

In main.c file, right after #include <atmel_start.h>, add this line:

```
struct io_descriptor *io
```

In the main() function, we add these lines of code to get the USART descriptor and enable it.

```
usart_sync_get_io_descriptor(&USART_0, &io);
usart_sync_enable(&USART_0);
```

Finally, we add the write function to our interrupt handler:

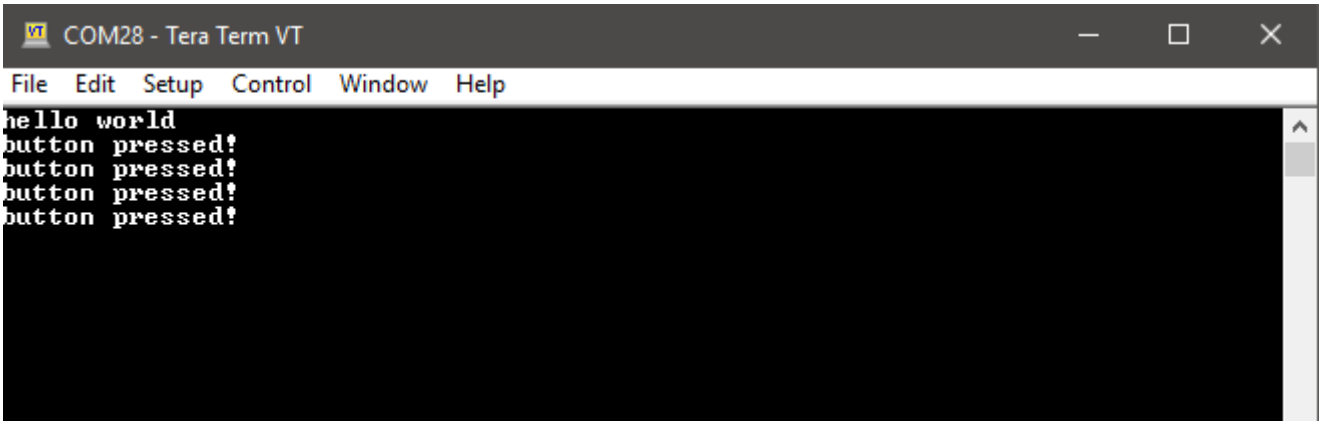
```
static void button_pressed_interrupt_handler(void)
{
	gpio_toggle_pin_level(LED0);
	io_write(io, (uint8_t *)"button pressed!\n", 16);
}
```

This is our complete main.c file

```
#include <atmel_start.h>

struct io_descriptor *io;

static void button_pressed_interrupt_handler(void)
{
	gpio_toggle_pin_level(LED0);
	io_write(io, (uint8_t *)"button pressed!\n", 16);
}

int main(void)
{

	/* Initializes MCU, drivers and middleware */
	atmel_start_init();
	ext_irq_register(PA15, button_pressed_interrupt_handler);

	printf("hello world\n");

	usart_sync_get_io_descriptor(&USART_0, &io);
	usart_sync_enable(&USART_0);



	/* Replace with your application code */
	while (1) {

#if 0 //Blinky LED Lab
		gpio_toggle_pin_level(LED0);
		delay_ms(1000);
#endif

#if 0 // Polling Button Lab
		if(gpio_get_pin_level(SW0) == false){
			// Button pressed, turn LED on
			gpio_set_pin_level(LED0, false);
		}
		else{
			// vice versa
			gpio_set_pin_level(LED0, true);
		}
#endif
	}
}
```

Compile and run the code again. This is the result whenever the button is pressed

You have to navigate to the file driver_init.c and modify the PA15 to GPIO_PULL_UP again because this file was overwritten by Atmel Start during the generation process.



Deep dive into generated code

Please see companion slides

Contact information

Quang Hai Nguyen, B. Eng.
Field Application Engineer

P +49 6102 50308228
M +49 1511 6242003
quanghai.nguyen@arrow.com

Arrow Central Europe GmbH
Frankfurter Straße, 211
63263 Neu-Isenburg

THE END