AN2474

将Atmel软件框架（ASF）项目从ASFv3.3移植到ASFv4

简介

本应用笔记介绍从Atmel Start ASFv3项目到ASFv4项目的移植。

由于ASFv4针对代码密度进行了优化并增加了灵活性，因此这两种框架是互斥的。这意味着ASF v4.0无法构建ASF v3.3项目。因此，项目移植需要“从头开始”。

ASFv3

ASF 3架构开发的主要目标是实现代码长度、性能和低功耗的三重优化。外设驱动程序和中间件开发的目标是缩短使用ASF项目时的设计周期。

ASF旨在提供一组丰富的驱动程序和代码模块（由Atmel专家开发且经过验证）以缩短客户的设计时间。它简化了单片机的使用，实现了硬件和高价值中间件的抽象化。

ASF包含源代码模块和演示其使用方法的应用程序。

- **驱动程序**由`driver.c`和`driver.h`文件组成，提供了访问外设或器件特定功能的低级寄存器接口函数。服务和组件将与驱动程序接口。
- **服务**是一种模块类型，用于提供更多面向应用的软件（例如USB类、FAT文件系统、架构优化的DSP库和图形库等）。
- **组件**是一种模块类型，用于提供软件驱动程序以访问存储器（例如Atmel DataFlash®、SDRAM、SRAM和NAND闪存）、显示器、传感器和无线设备等外部硬件组件。
- **开发板**包含所有数字和模拟外设到Atmel开发工具包的各I/O引脚的映射。

ASFv4

在Atmel Start中，驱动程序和软件协议栈作为新一代Atmel软件框架（ASFv4）的一部分提供。该版本从头开始构建，全面地重新设计并实现了整个框架来解决旧版ASF用户和反馈者报告的问题，并且更好地集成了Atmel Start Web用户界面。与此同时，确保ASF老用户对ASFv4仍然保有一种熟悉感，新用户又能够轻松上手。为满足此版本要求，ASFv4中进行了一些必要更改，*Atmel Start用户指南*的ASFv4与ASFv3基准测试部分列出了最为重要的更改。

ASFv4紧密集成到Atmel Start中，这意味着ASFv4代码较以往能够更加具有针对性的面向用户规范进行定制。例如，无需再使用C预处理器条件表达式来使能/禁止代码块，禁止的代码块可以从项目源中完全删除，从而生成更加清晰易读的代码。集成至Atmel Start中意味着能够在更加用户友好的环境完成软件配置，加载到器件上的惟一配置信息是原始外设寄存器内容，这样可使固件映像更加紧凑。

我们解决的一个重要问题是基于ASF的代码的内存占用和性能。许多用户认为运行基于ASFv3的代码对闪存要求太高。现在已经通过使用代码生成和更改外设的初始化方式得以解决。反馈的性能问题通常是中断延时长/代码执行慢，此问题已通过使中断处理程序更加简短和简单得到了解决。

移植

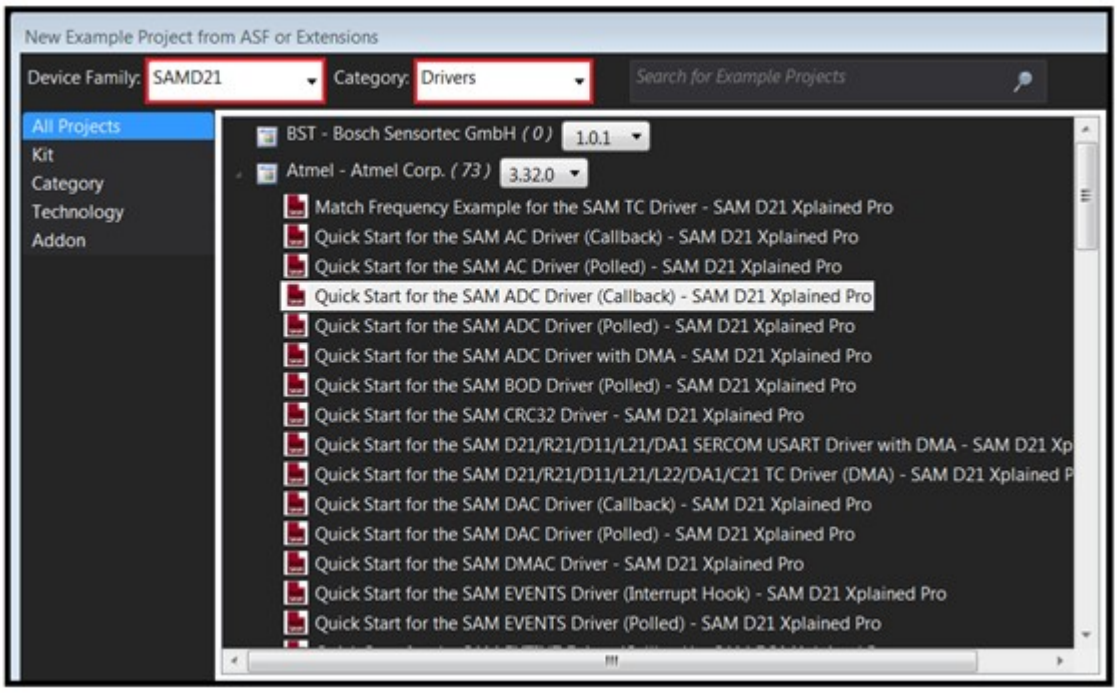
遗憾的是，ASFv3项目无法直接移植为ASFv4项目。对于全新的架构和API，惟一的移植方法是使用在线Atmel Start工具（start.atmel.com）从头开始重新编译ASFv3项目。虽然现阶段可能面临着挑战，但未来在使用ASFv4项目的器件之间进行移植会比使用ASFv3项目的器件之间的移植更加方便。

启动示例项目

为了逐步介绍移植过程，将使用ASF v3.3示例项目列表中的示例项目。所选示例项目将与SAMD21 Xplained Pro评估工具包板搭配使用。

1. 在 Atmel Studio 中， 选 择 ***File > New > Example Project***（文件 > 新建 > 示例项目），加载ASFv3项目。
2. 在Device Family（器件系列）菜单中，选择SAMD21（如图1所示）。
3. 单击All Projects（全部项目）以显示所有Atmel项目，然后选择Quick Start for the SAM ADC Driver (Callback) – SAM D21 Xplained Pro（SAM ADC驱动程序快速入门（回调）——SAM D21 Xplained Pro）。
4. 单击**OK**（确定），在默认Studio工作区中创建项目。
5. 编译并运行示例项目。

图1： 导入ASFv3示例项目



6. 应用程序说明

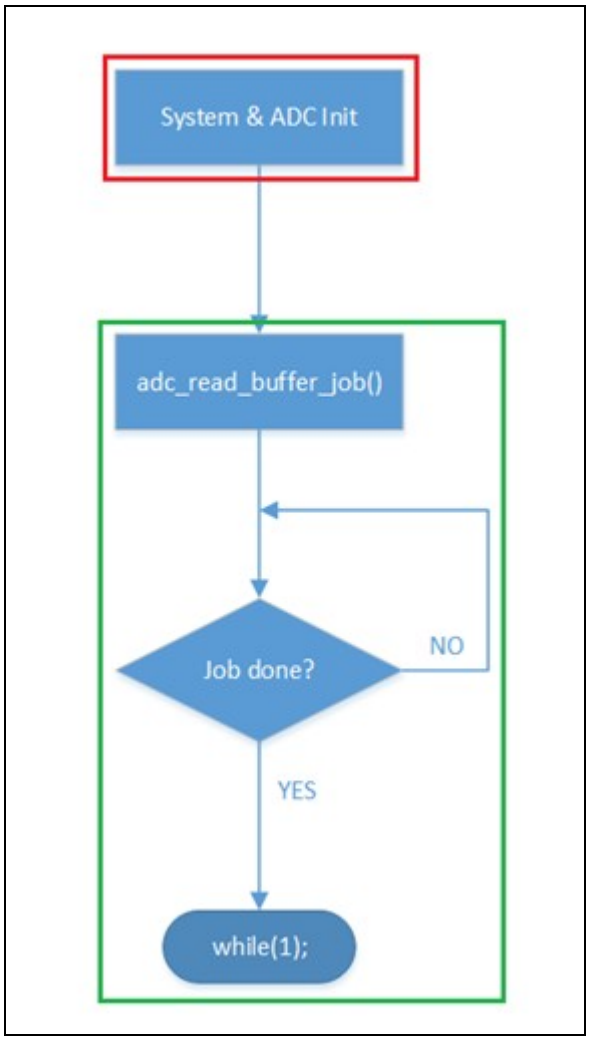
示例ADC程序流程图如图2所示。示例应用程序将启动ADC来收集一组样本，并在数据收集完成后一直等待。

应用程序非常简单，但需要分两步进行移植。步骤1（如红色框所示），从ASFv3的系统和ADC初始化中提取所需信息，以便稍后在Atmel Start中使用。步骤2（如绿色框所示），对应用程序所需API进行功能验证，该步骤在Atmel Start项目导入Atmel Studio中后完成。

7. 应用程序组件

为了确保应用程序正常工作，需要了解系统（时钟等）以及ADC外设。图2中的红色框包含任何项目所需的系统驱动程序。为了确保系统可以为外设提供必要的时钟和配置，需要进行一些相应的配置。ADC也包含在此框内，需要经过评估才能为应用程序传输所需的配置。

图2： 示例ADC程序流程图



提取系统和ADC的ASFv3配置

1. 捕捉时钟配置

ASFv3的系统时钟设置位于示例项目的conf_clocks.h文件中。从这里可以看到，主时钟（GCLK_0）源自内部的8 MHz振荡器。

如图3所示，GCLK_0未进行预分频，并以8 MHz的频率运行SAMD21内核。这些设置将在Atmel Start中用于配置ADC项目的时钟树。对于此项目中的ADC外设，GCLK_0将用作时钟源。运行时，可以在I/O窗口的通用时钟发生器（GCLK）和系统控制（SYSCTRL）下查看这些值。

2. 捕捉外设配置。

ASFv3中的外设模块使用配置结构来初始化应用程序的外设。ASFv3包含写入外设的初始化默认值。需要注意的是，外设配置结构会在运行时填充。ASFv4采用的方法有所不同，其通过预处理器宏来处理所有应用程序配置。

在本示例应用程序中，adc_config结构用于在configure_adc()函数中初始化ADC。在项目执行期间，可使用Watch（观察）窗口捕捉初始化参数，如图4所示。在ASFv4开发过程中，尽可能地保持了配置值的命名约定。这表示，图4中间列中的值将用于在Atmel Start中设置ADC。

ADC寄存器图5显示了ASFv3初始化后的寄存器值，这些值将用于通过Atmel Start验证ADC初始化。

图3： 系统时钟配置

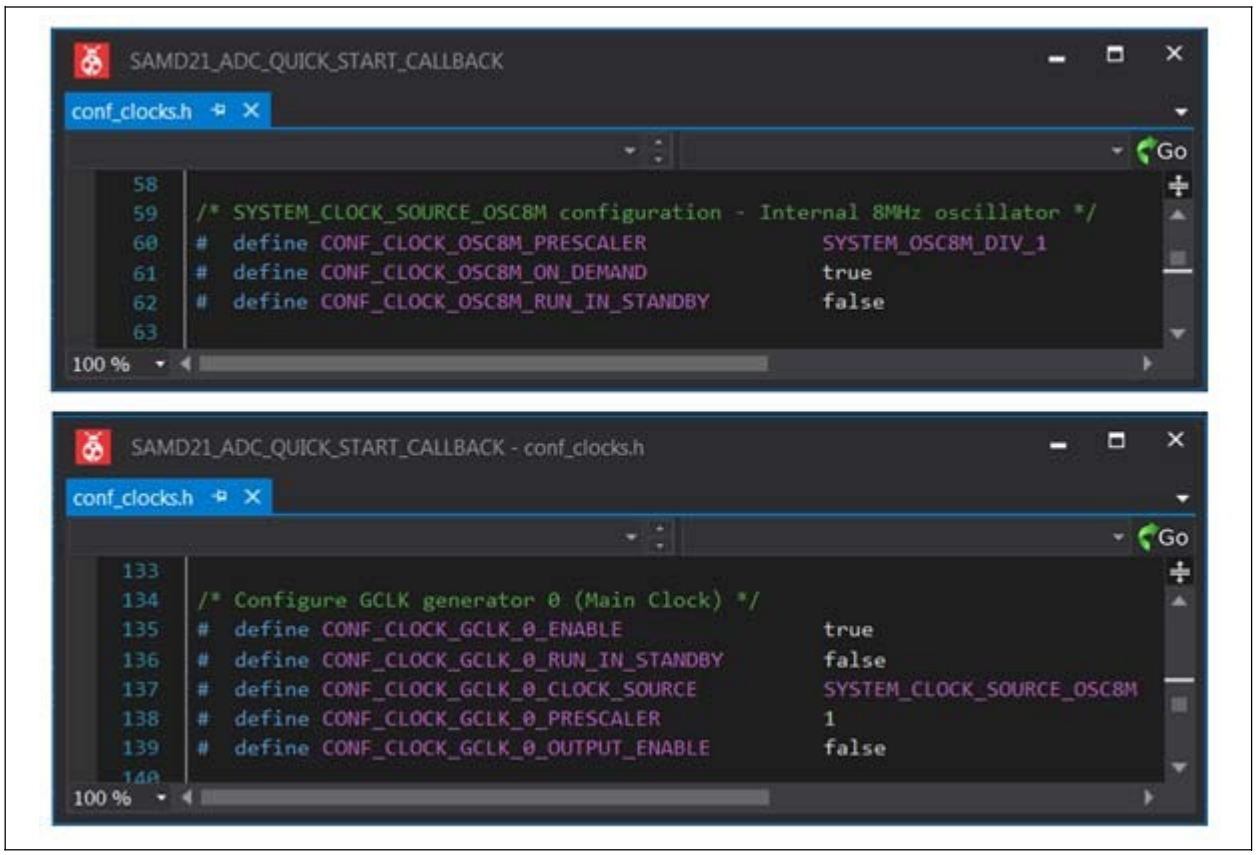


图4： ASFv3 ADC配置参数

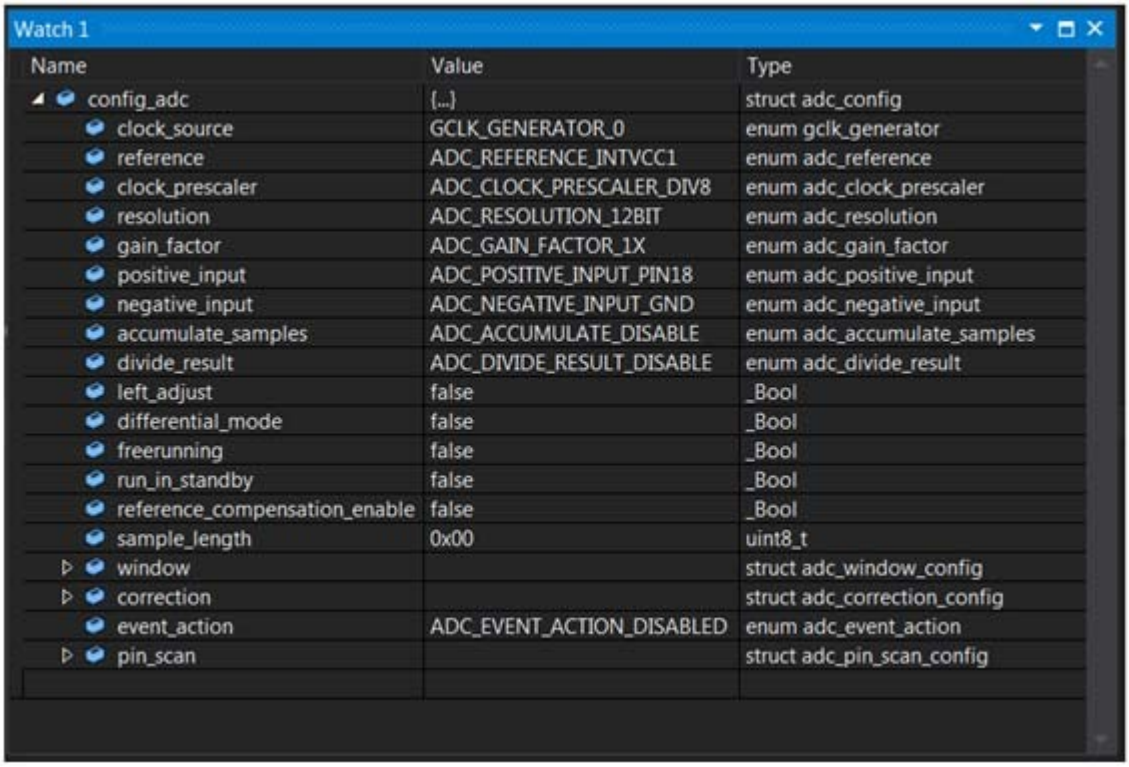
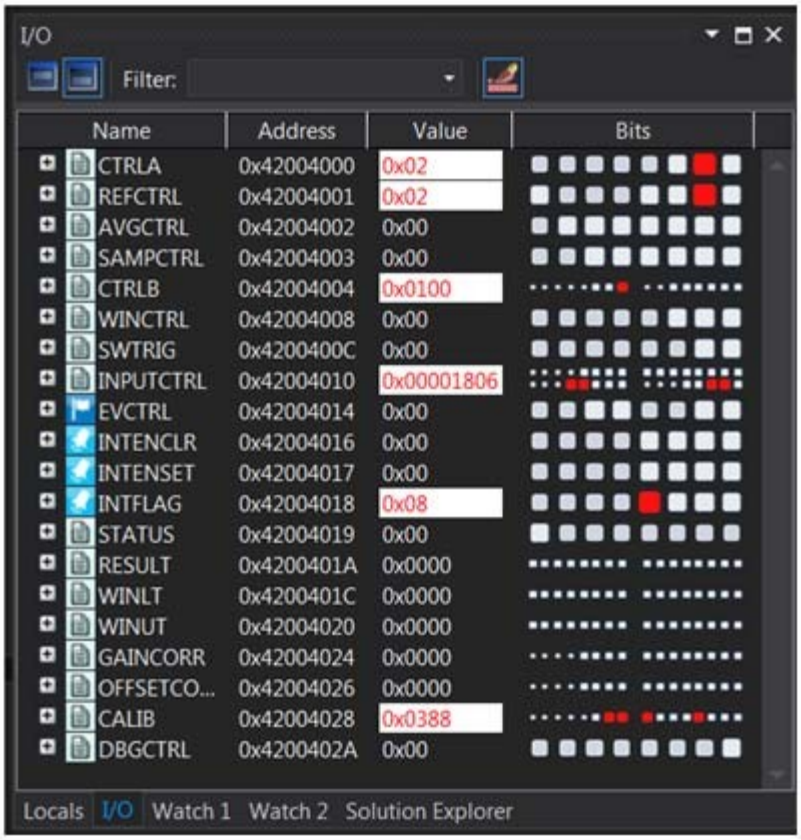


图5： ASFv3 ADC寄存器值



3. 捕捉引脚配置。

所选示例项目仅为应用程序使用一个ADC通道。在这种情况下，很容易找到引脚分配，它们位于之前引用的`config_adc`结构或`configure_adc()`函数中。对于使用Atmel硬件（如Xplained Pro）的ASFv3示例项目，该信息可能位于`system_board_init()`函数中初始化硬件的位置。

ATMEL START——系统初始化

1. 浏览到start.atmel.com，然后单击**Create New Project**（创建新项目）选项卡来创建新的Atmel Start项目，如图6所示。
2. 在Create New Project窗口的Results（结果）部分下，搜索SAM D21 Xplained Pro并选择SAM D21 Xplained Pro，然后单击**Create New Project**（见图7）。
3. 随即将显示新项目仪表板，如图8所示。

图6： ATMEL START主页

图7： ATMEL START创建新项目

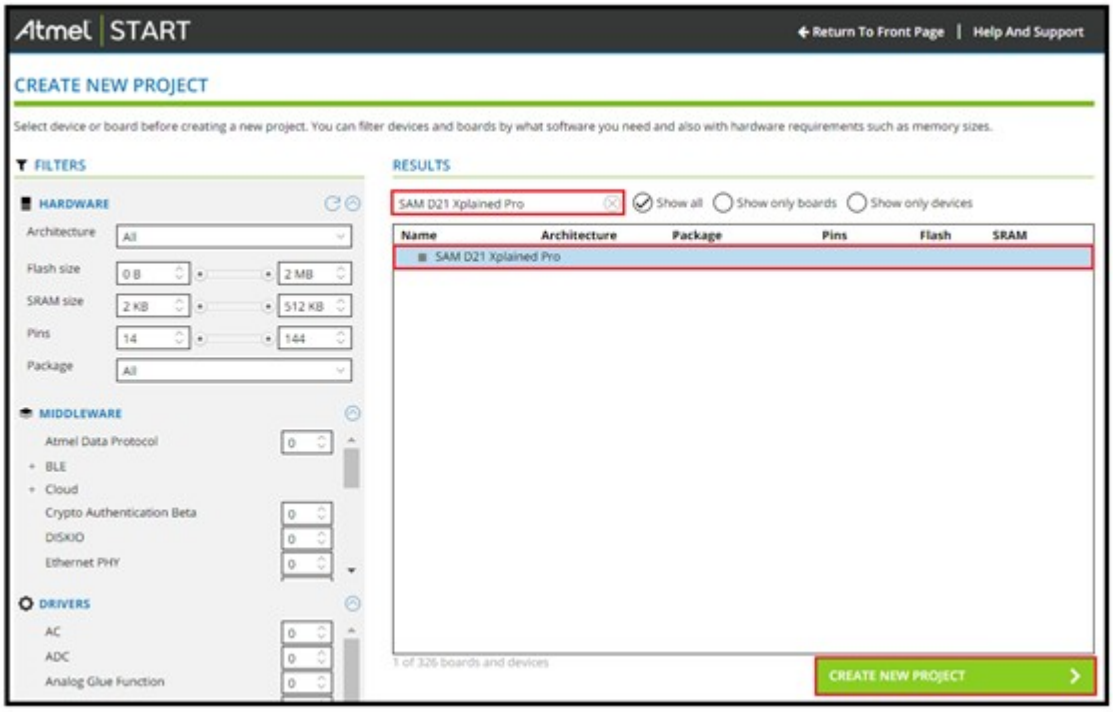
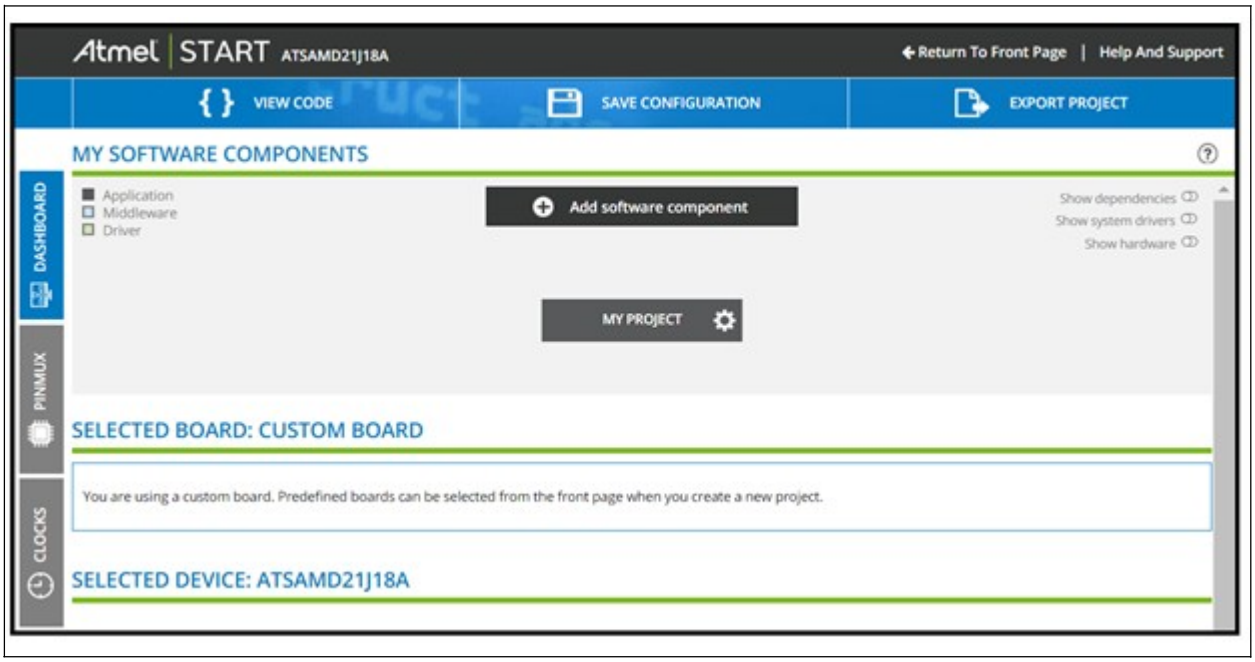
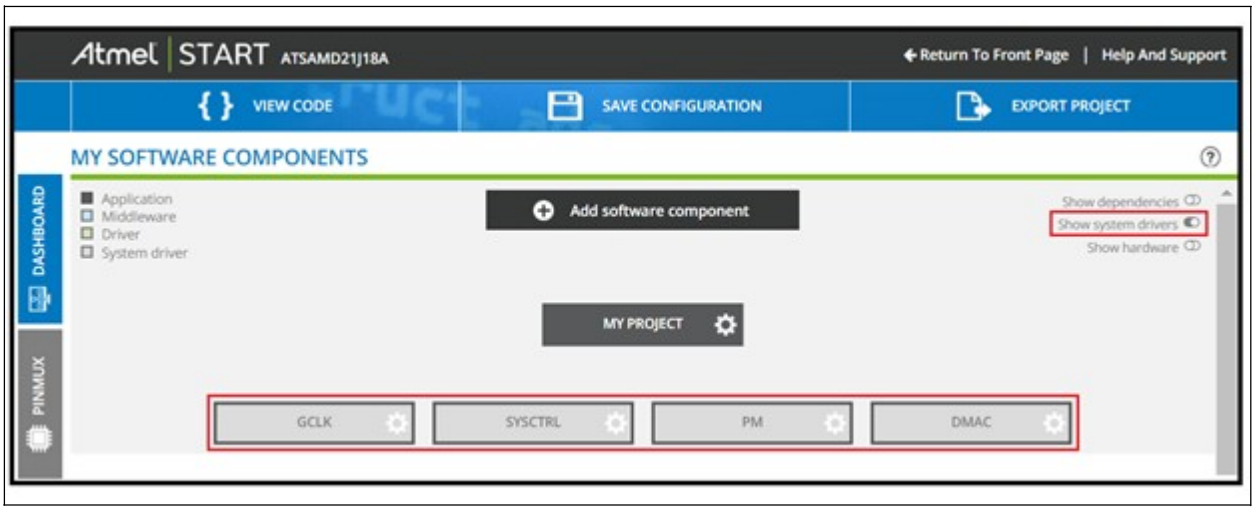


图8：　新项目仪表板



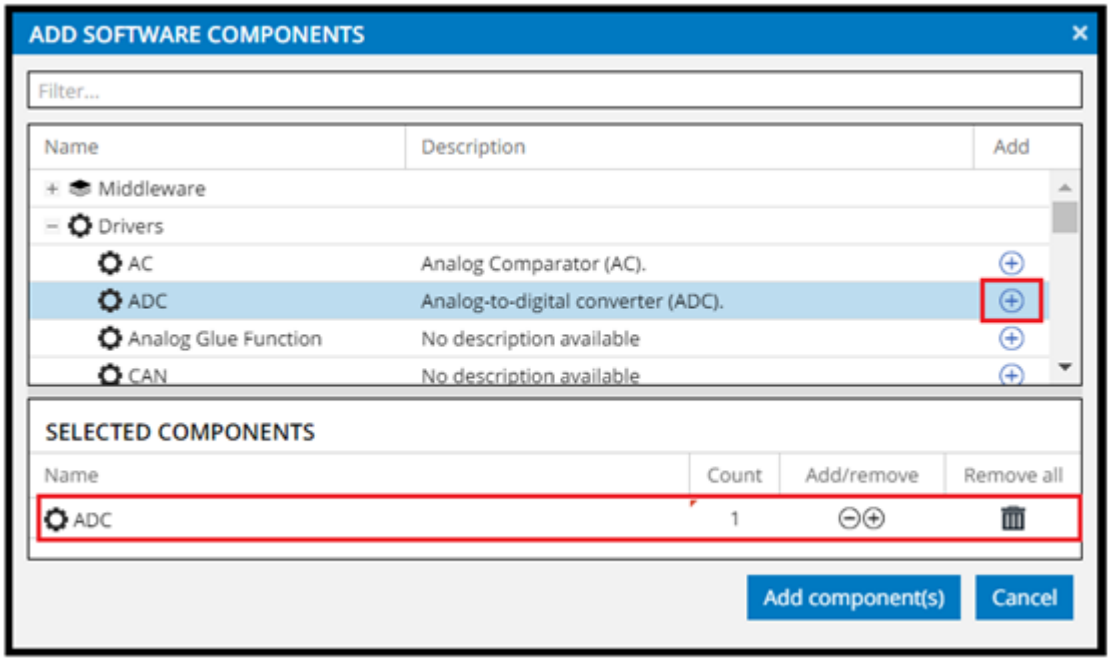
请注意，每个ASFv4项目中都包含多个系统驱动程序，这一点与ASFv3相同。这些驱动程序最初隐藏在Start设置中。要查看自动包含的系统驱动程序，单击Show system drivers（显示系统驱动程序）指示器。四个组件（SYSCTRL、DMAC、PM和GCLK）允许在Dashboard（仪表板）屏幕中配置时钟、总线、NVM和DMA设置，如图9所示。

图9：　包含的系统驱动程序



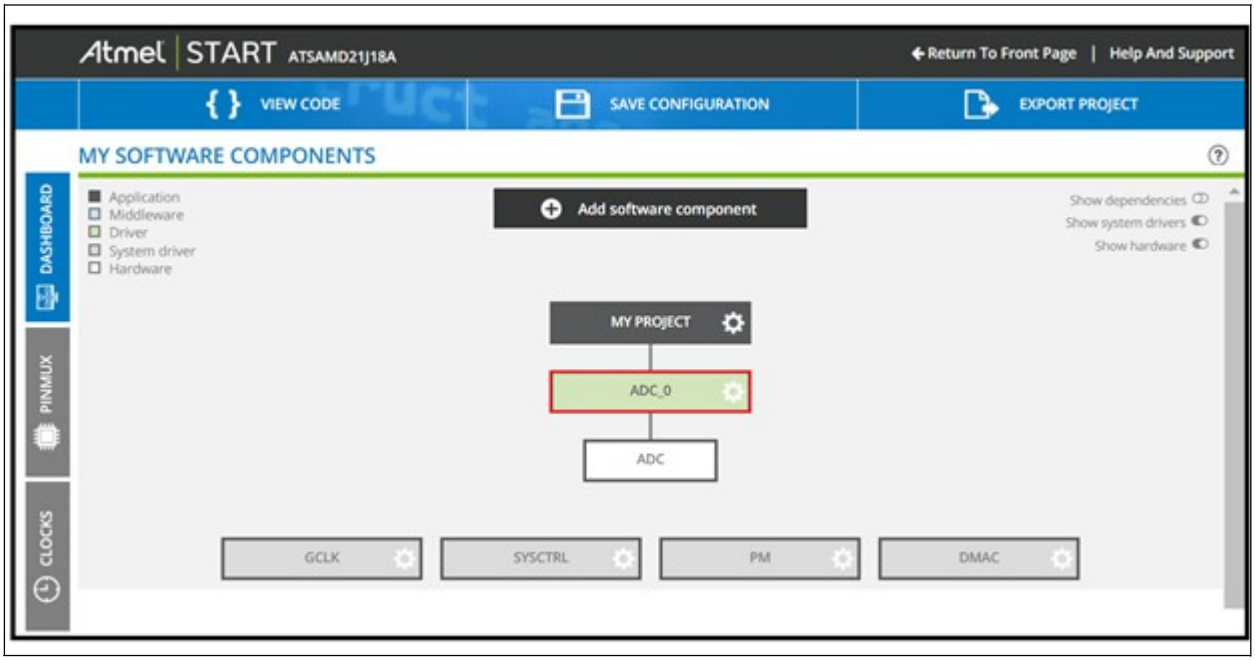
4. 在 **Dashboard** 选项卡中，单击 **Add software component**（添加软件组件），将外设模块添加到项目中。Add Software Components对话框将显示SAMD21的可用驱动程序，如图10所示。
5. 单击“+”显示所有驱动程序，然后单击“+”符号选择ADC。ADC模块将显示在Selected Components（已选组件）部分下。
6. 单击**Add Component(s)**（添加组件），将所选组件添加到项目中。

图10： ATMEL START添加软件组件



7. 配置外设模块。添加ADC模块后，它将立即显示在**My Project**（我的项目）选项卡下，如图11所示。

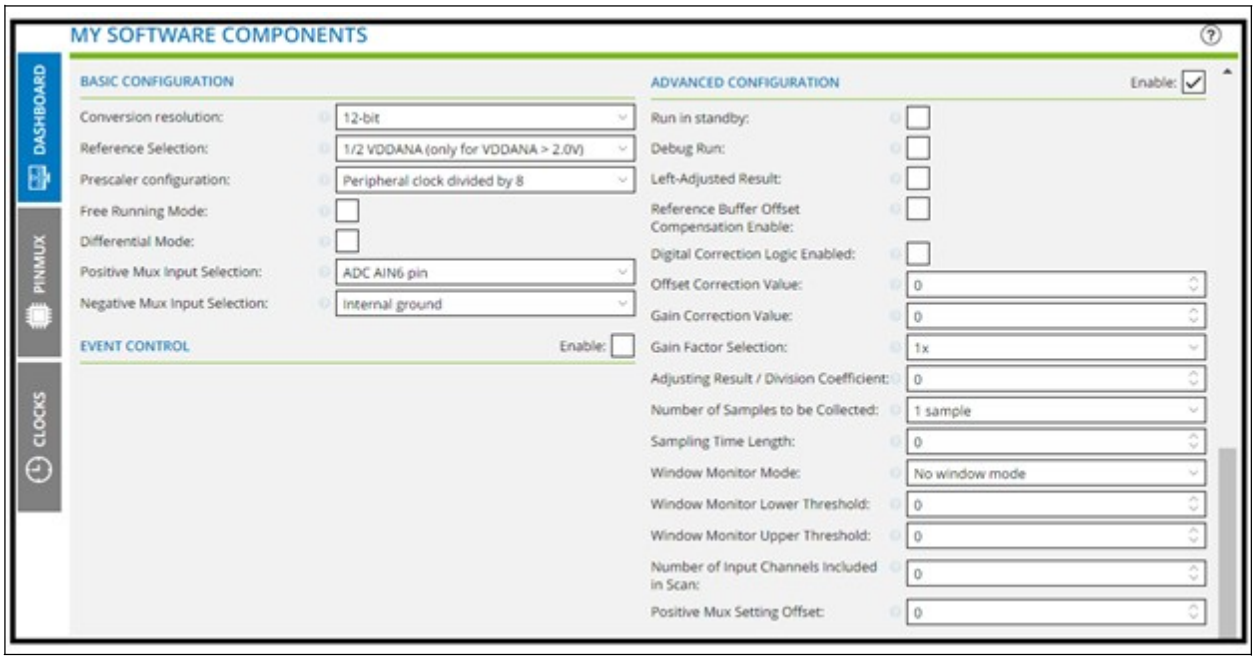
图11： 配置外设模块



8. 单击**ADC_0**选项卡，使用从ASFv3示例项目的`adc_config`结构中收集的信息来填充配置选项，然后为ADC输入使能PA06（AIN/6），如图12所示。

图12： 配置ADC外设（1/2）

图13： 配置ADC外设（2/2）



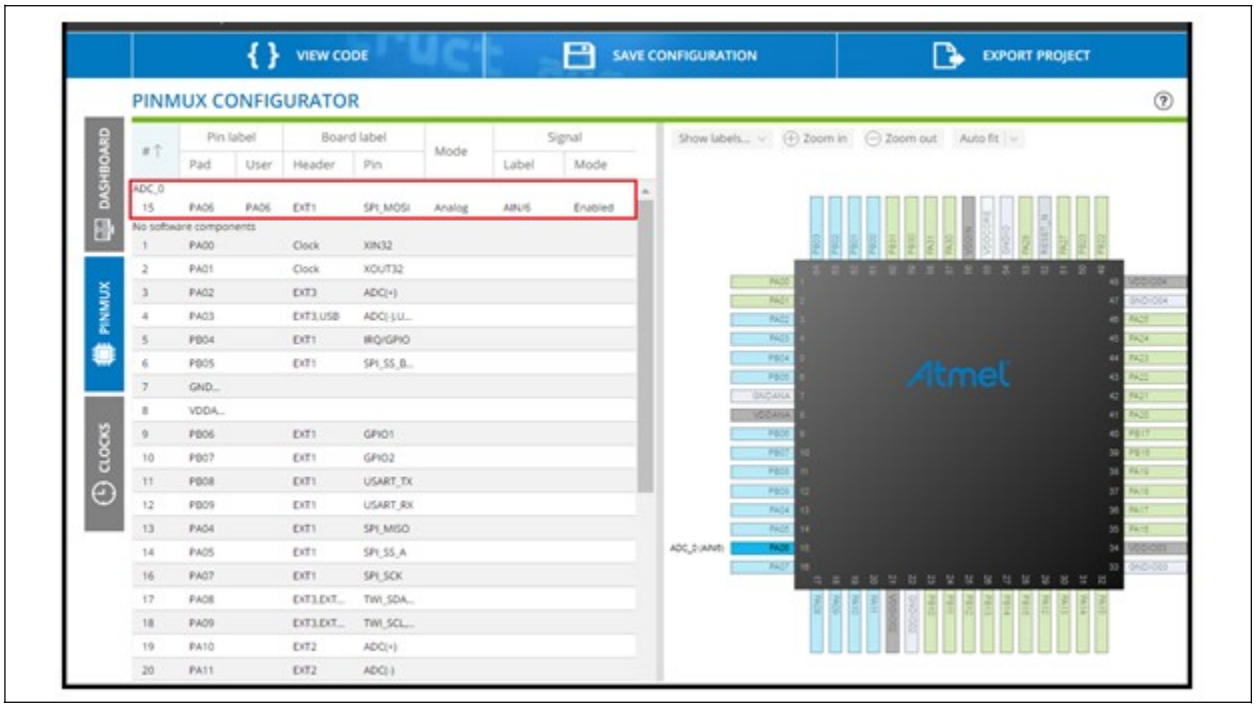
9. 配置引脚排列。

   从ASFv3项目中收集信息时，ADC在此项目中仅使用一个引脚，并在仪表板的ADC配置中对其进行配置。但是，GPIO或引脚命名在PINMUX CONFIGURATOR（引脚复用配置器）中完成，如图14所示。在PINMUX CONFIGURATOR中选择要修改的引脚后，用户窗口将随即显示在屏幕下方以设置参数。

10. 配置时钟树。

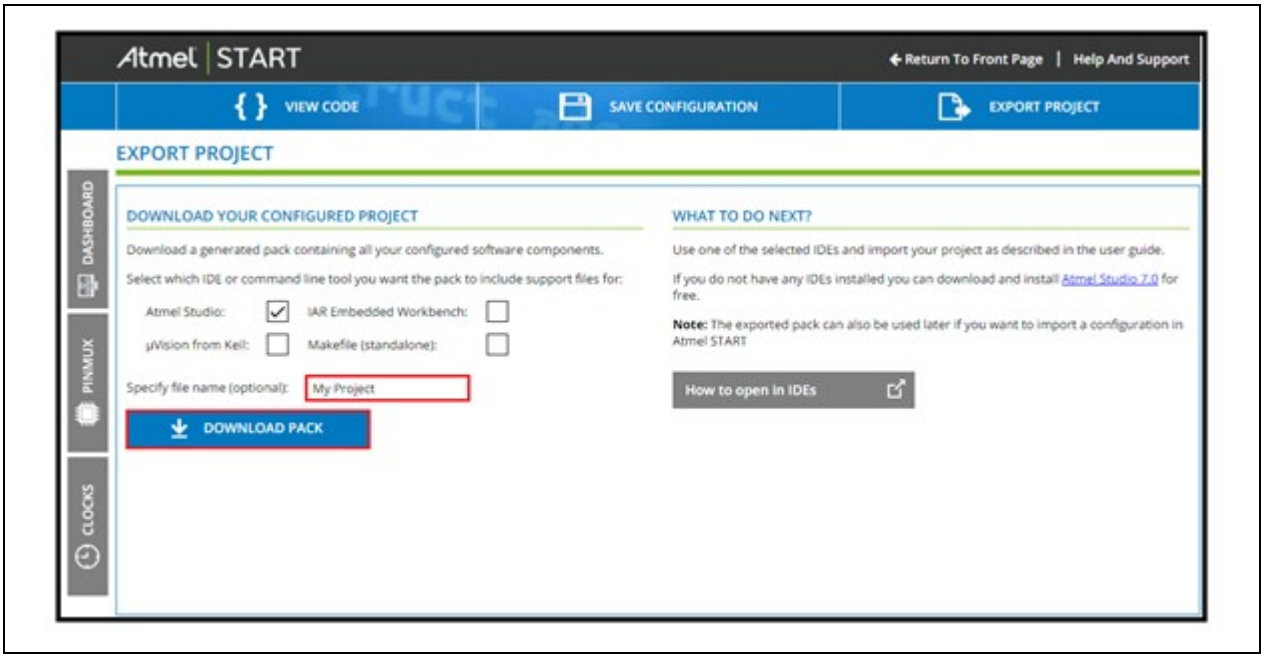
    在Start工具中，可以在两个位置对时钟树进行配置。仪表板（通过系统驱动程序模块）和时钟配置器。时钟配置器以图形方式表示时钟树，提供与仪表板相同的配置功能。本示例中将使用时钟配置器。

图14： ATMEL START——引脚复用



11. 导出项目。单击Atmel Start屏幕右上方的Export Project（导出项目）选项卡以导出项目，如图15所示。应仅选择Atmel Studio，因为它是惟一要使用此导出项目的IDE。Atmel Start的输出为atzip文件。

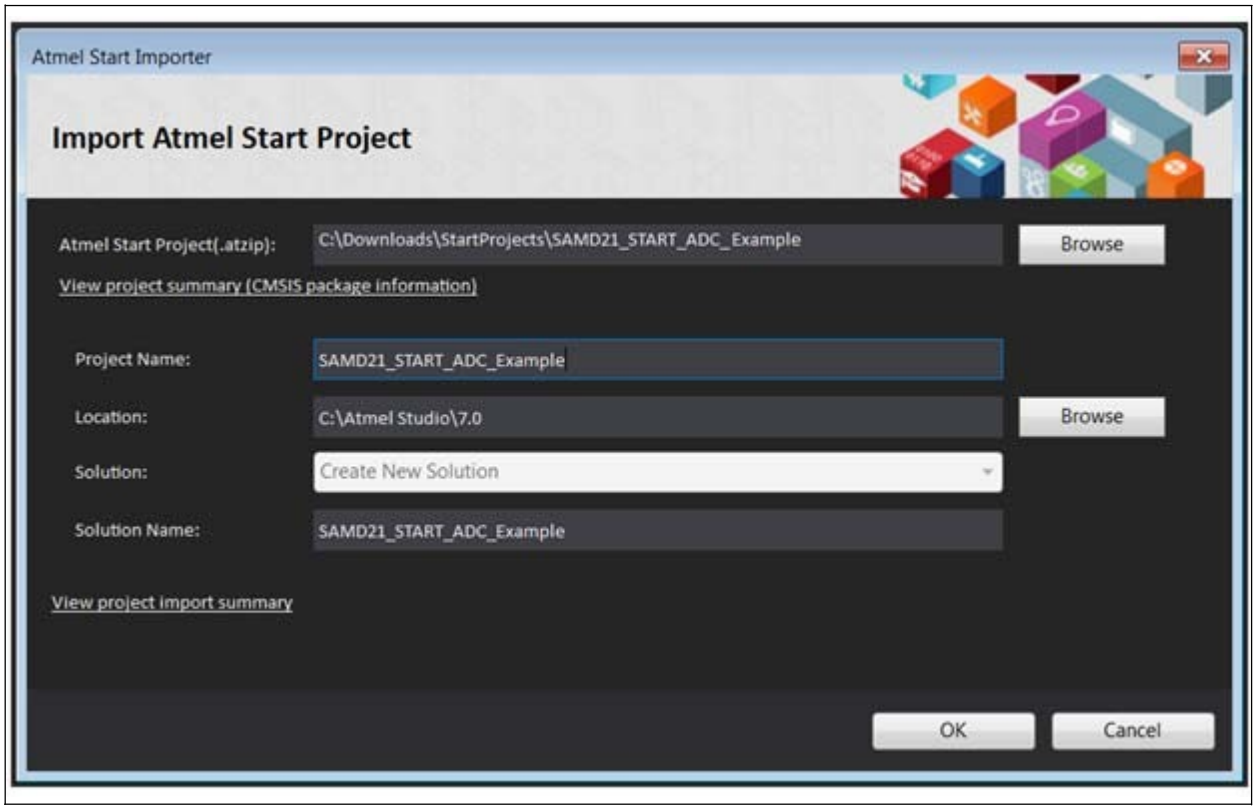
图15： ATMEL START——导出项目



验证系统配置

1. 在 Atmel Studio 中， 选 择 *File > Import > Atmel Start Project*（文件 > 导入 > Atmel Start项目），导入从Atmel Start中导出的atzip文件。

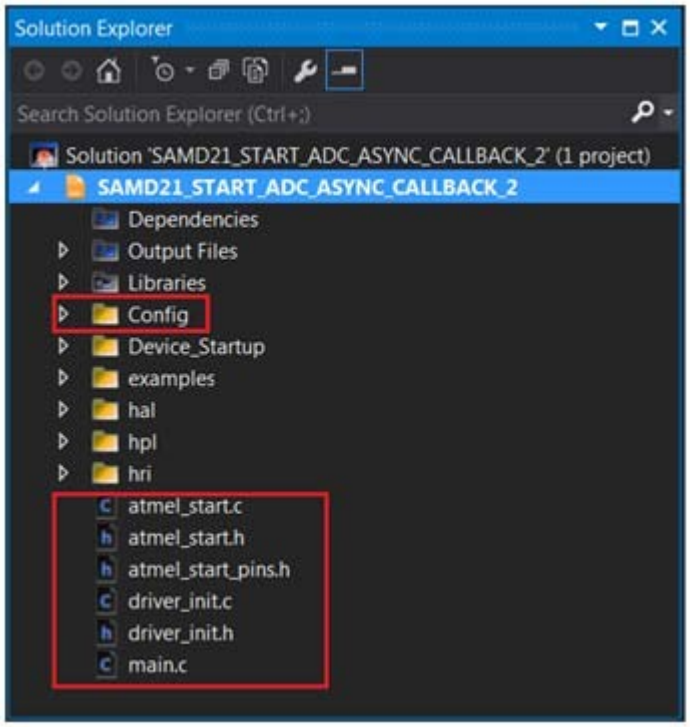
图16： ATMEL START——导入项目



可在Config文件夹（用于系统初始化）、atmel_start_pins.h（用于引脚分配）和driver_init.c/driver_init.h（用于外设初始化）中轻松找到Start中设置的参数（见图17）。

- Config——包含用于预处理器器件配置的寄存器设置（来自Start）
- atmel_start.c——初始化MCU、驱动程序和中间件
- atmel_start_pins.h——保存来自Start配置的MCU引脚分配和命名
- driver_init.c——包含外设的初始化函数
- main.c——调用atmel_start_init()以初始化系统

图17： ASFV4文件结构



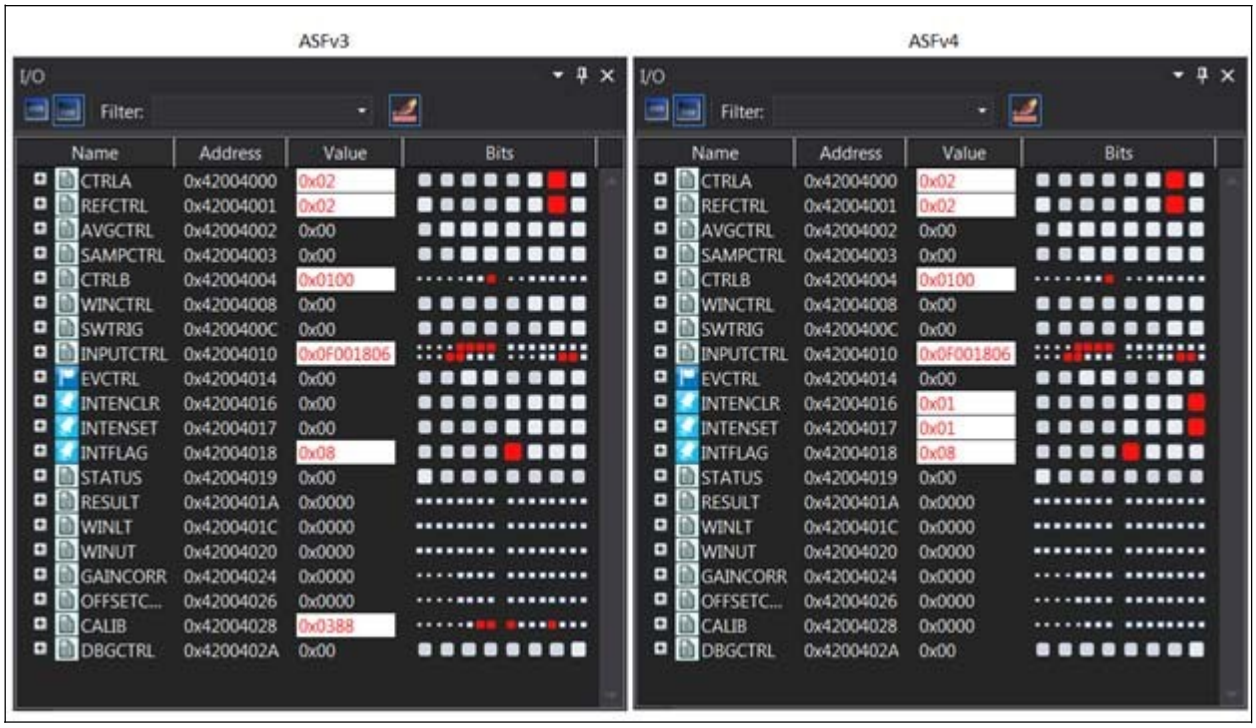
2. 编译并运行项目。

   由于新项目中不包含任何应用程序代码，因此Start不会为`system_init()`中定义的外设输出初始化代码。运行项目并在初始化序列后停止，在I/O窗口中查看寄存器设置。

   单击I/O图标，可以在Atmel Studio中打开I/O窗口。

3. 比较ASFv3和ASFv4之间的寄存器设置，首先打开I/O窗口并选择模数转换器，然后将结果与从ASFv3中收集的结果进行比较（见图18）。

图18： ASFv3与ASFv4 ADC寄存器比较



通过比较，二者之间的差异一目了然。例如，允许ADC中断的位置和ADC的校准设置。

ASFv3在ADC作业函数中允许中断，而ASFv4在分配回调函数时允许中断。

此外，ASFv4 ADC驱动程序不导入ADC的出厂校准设置。此操作必须由用户完成。校准提供偏置和线性设置以获取更精确的ADC读数。更多信息，请参见《SAM D21系列数据手册》（DS40001882）的**第9.3.2节“NVM软件校准区域映射”**。要将校准导入ASFv4项目，请参见**附录A：“应用程序代码示例”**。

应用程序转换

1. 应用程序说明。

   ADC将在初始化后启动并收集128个12位样本，然后将数据存储在缓冲区中。样本收集完成后，将进入ADC回调并将标志置1以指示采样完成。

2. 比较API。

   除Start设置提供的配置外，使用ADC模块无需任何其他配置。ASFv3与之非常相似，其提供了一组默认值，但需要开发人员创建初始化函数并直接修改默认值。

   示例程序使用adc_read_buffer_job以定义数量的样本来填充缓冲区。该功能在ASFv4中没有直接匹配项。ASFv4中最接近adc_read_buffer_job的是adc_async_start_conversion。但是，这两个函数产生的结果会非常不同。在ASFv3中，ADC _adc_interrupt_handler配置为完成作业并以请求的样本数量填充分配的缓冲区。ASFv4 ADC处理程序设计为采用一种更通用的方式，要求开发人员在回调中处理附加采样。为匹配ASFv3的功能，ASFv4中的ADC回调会将ADC结果放置到数组中并启动下一个ADC事务。

   有关完整的设置代码，请参见**附录A：“应用程序代码示例”**。

3. 回调设置。

   对于一般应用程序，主项目树中的example文件夹包含的代码将用于设置为项目使能的外设。在这种情况下，示例文件夹包含使能应用程序回调所需的一切内容。

   在ASFv3和ASFv4中，设置ADC回调的方式相似。区别在于发出中断信号时ADC中断处理程序起作用的方式。搜索_adc_inter- rupt_handler查看ASFv3和ASFv4之间的差别。ASFv4的开销更小，允许开发人员在回调中处理特殊功能。

   有关完整的设置代码，请参见**附录A：“应用程序代码示例”**。

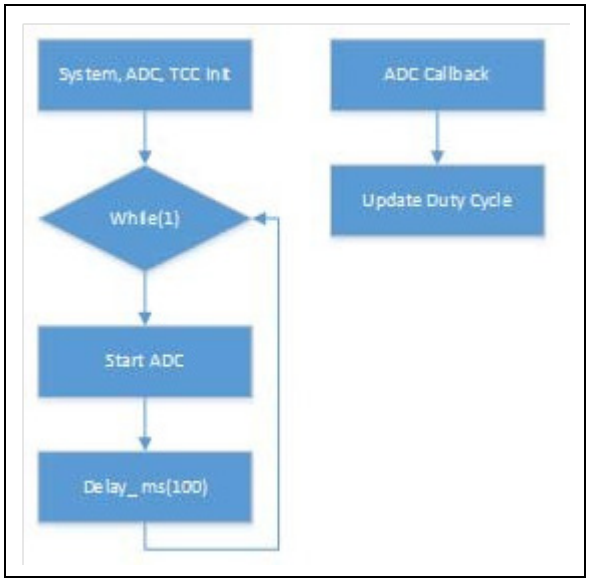
项目扩展

按以下方式将定时器控制模块添加到ASFv3项目中。

1. 应用程序说明。

   更新后的应用程序将使用ADC回调来更新与LED相连的PWM通道的占空比。占空比将使用从ADC中读取的值进行计算。此外，ADC不是通过单个作业收集一连串ADC值，而是连续循环启动ADC以便每100 ms更新一次PWM通道的占空比。

图19： 扩展项目流程图



2. 应用程序组件。

   为实现项目的新功能，必须从ASF Wizard（ASF向导）中添加延时服务和TCC0驱动程序模块。

3. 打开ASFv3项目并添加定时器控制模块。

   在Atmel Studio的Project（项目）下，选择ASF Wizard。窗口打开后，确保项目下拉菜单中已填充要修改的示例项目。

   在Available Modules（可用模块）菜单中，选择已勾选回调选项的TC——定时器计数器（驱动程序）和延时服务模块。单击窗口底部的**Apply**（应用），将添加的模块应用到项目中。TC驱动程序和延时服务将显示在Select Modules（选择模块）菜单中。

通过ASF Wizard将TCC添加到项目中后，就可以在项目中找到tcc驱动程序文件夹（如图20所示），相应的头文件添加到asf.h以供在应用程序中使用。

图20： 添加的TCC0驱动程序

设置ASFv3定时器配置

1. 设置定时器配置。

   由于项目是在Xplained Pro上编译的，因此已为板上LED分配一些定义，将其用作PWM输出。这些定义包含在项目的samd21_xplained_pro.h文件中（见图21）。

   使用samd21_xplained_pro.h.中定义的宏创建configure_tcc函数，如图22所示。创建一个tcc_module结构的实例，称为tcc_instance。

   请注意，configure_tcc函数中的前几行中的一行为函数调用，用于获取默认TCC模块的一组配置值。ASFv3包含每个外设的一组默认设置。这些默认设置将用作此项目的基础，仅在需要时进行修改。

2. 捕捉TCC0寄存器设置。

   配置完TCC模块后，立即在Atmel Studio中编译并运行代码。捕捉寄存器设置的方法与之前使用I/O窗口为ADC捕捉设置的方法相同。这些值将用于与来自Atmel Start的输出进行比较。

图21： XPLAINED PRO定义

```
116  #define LED0_GPIO                 LED0_PIN
117  #define LED0                      LED0_PIN
118
119  #define LED_0_PWM4CTRL_MODULE     TCC0
120  #define LED_0_PWM4CTRL_CHANNEL    0
121  #define LED_0_PWM4CTRL_OUTPUT     0
122  #define LED_0_PWM4CTRL_PIN        PIN_PB30E_TCC0_WO0
123  #define LED_0_PWM4CTRL_MUX        MUX_PB30E_TCC0_WO0
124  #define LED_0_PWM4CTRL_PINMUX     PINMUX_PB30E_TCC0_WO0
```

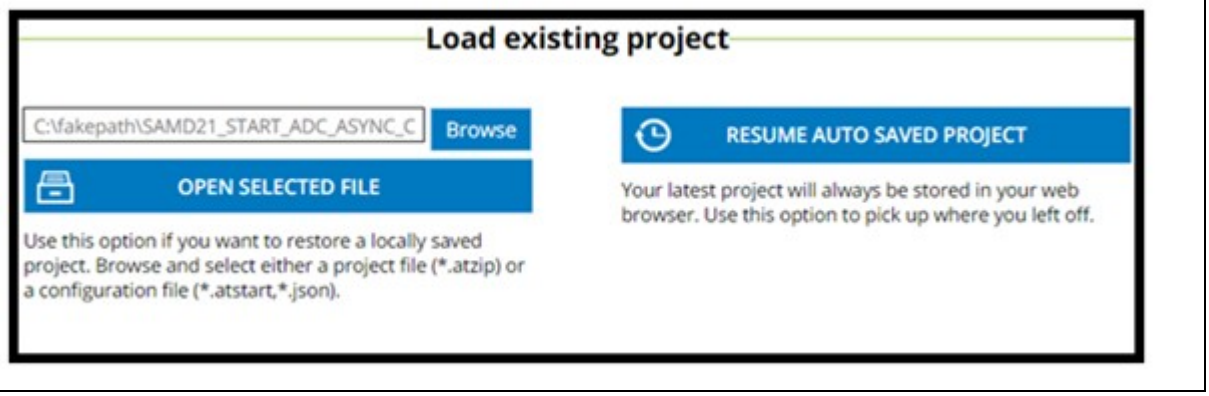
图22： XPLAINED PRO定义

```
struct tcc_module tcc_instance;

static void configure_tcc(void)
{
    struct tcc_config config_tcc;

    tcc_get_config_defaults(&config_tcc, LED_0_PWM4CTRL_MODULE);

    config_tcc.counter.clock_prescaler = TCC_CLOCK_PRESCALER_DIV8;
    config_tcc.counter.period = 0x7C;
    config_tcc.compare.wave_generation = TCC_WAVE_GENERATION_SINGLE_SLOPE_PWM;
    config_tcc.compare.match[LED_0_PWM4CTRL_CHANNEL] = 0x3E;

    config_tcc.pins.enable_wave_out_pin[LED_0_PWM4CTRL_OUTPUT] = true;
    config_tcc.pins.wave_out_pin[LED_0_PWM4CTRL_OUTPUT]        = LED_0_PWM4CTRL_PIN;
    config_tcc.pins.wave_out_pin_mux[LED_0_PWM4CTRL_OUTPUT]    = LED_0_PWM4CTRL_MUX;
    //config_tcc.double_buffering_enabled  = false;

    tcc_init(&tcc_instance, LED_0_PWM4CTRL_MODULE, &config_tcc);

    tcc_enable(&tcc_instance);
}
```

ATMEL START——添加定时器

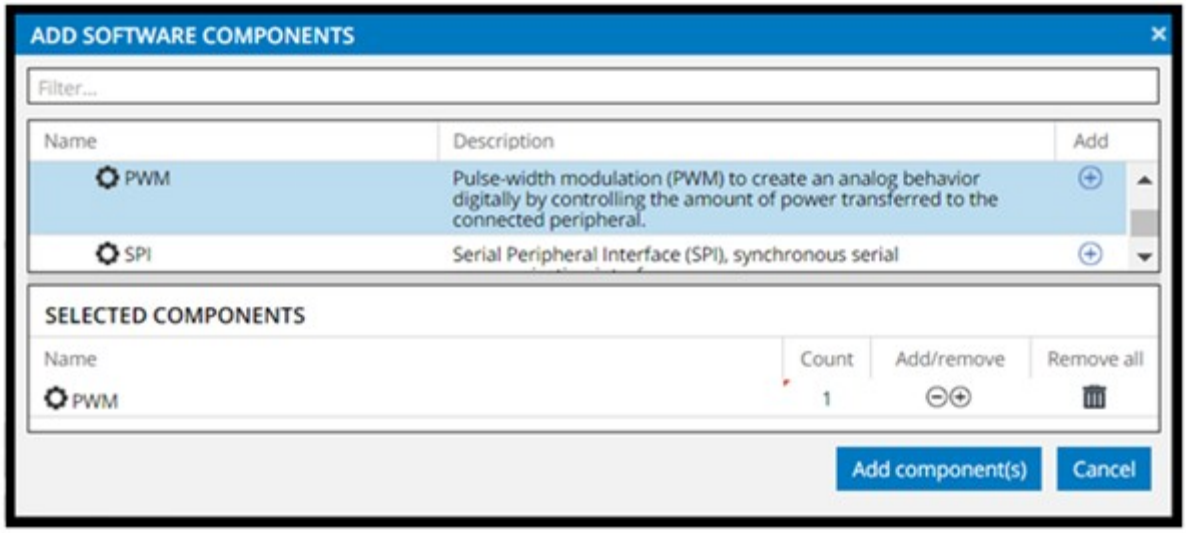
1. 通过浏览到Atmel Start主页，然后选择用于加载现有项目的选项导入Atmel Start项目。然后，浏览到下载的atzip文件并单击**Open Selected File**（打开所选文件）。

图23： ATMEL START——加载项目



2. 通过单击**Add Software Components**，然后在窗口中选择PWM组件，将外设模块添加到项目中。请注意，PWM组件和TCC组件虽然都使用TCC模块，但二者有所不同。

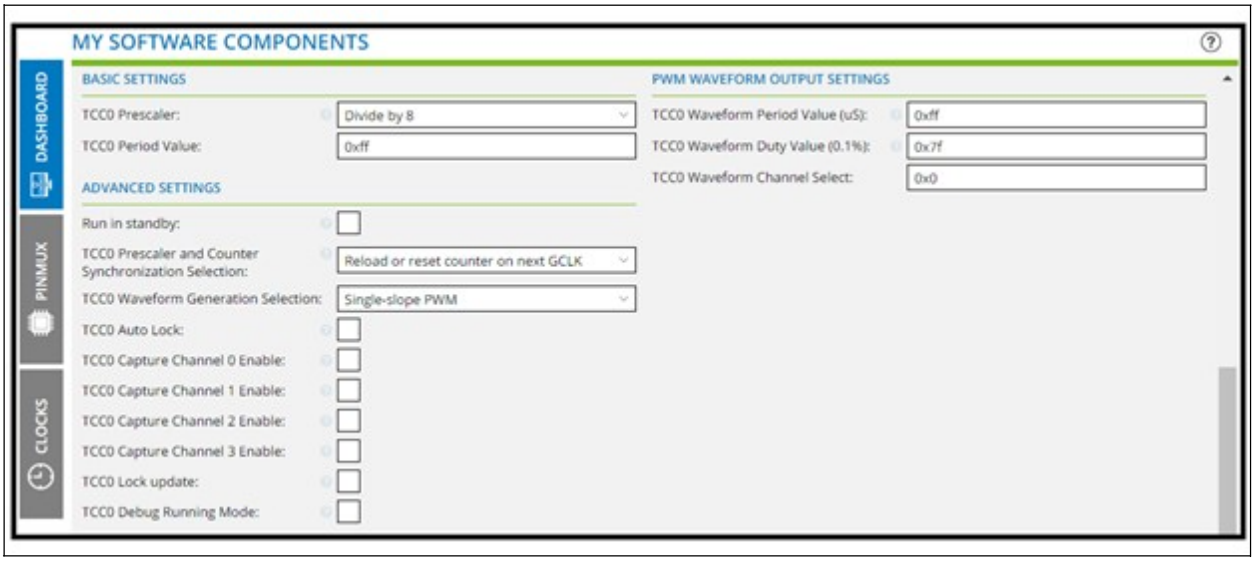
图24： ATMEL START——添加软件组件



3. 通过选择新添加的PWM_0模块并使用从ASFv3项目中收集到的配置值来配置外设模块，如图25和图26所示。

图25： ATMEL START——PWM配置（1/2）

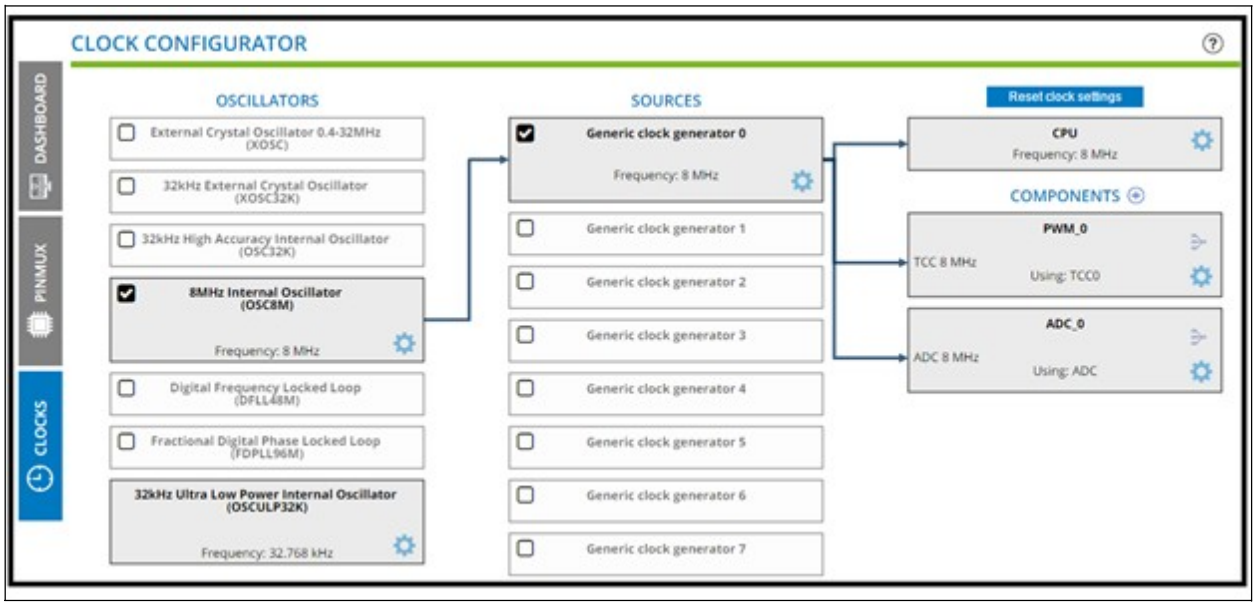
图26： ATMEL START——PWM配置（2/2）



4. 配置时钟树。

   在该项目的初始版本中，通用时钟发生器0（GCLK_0）的频率为8 MHz。务必确保ADC_0和PWM_0的时钟信号均由GCLK_0提供。如果PWM_0组件的时钟信号并非由GCLK_0提供，可单击组件块上的设置图标并将TCC时钟源设置为GCLK_0。

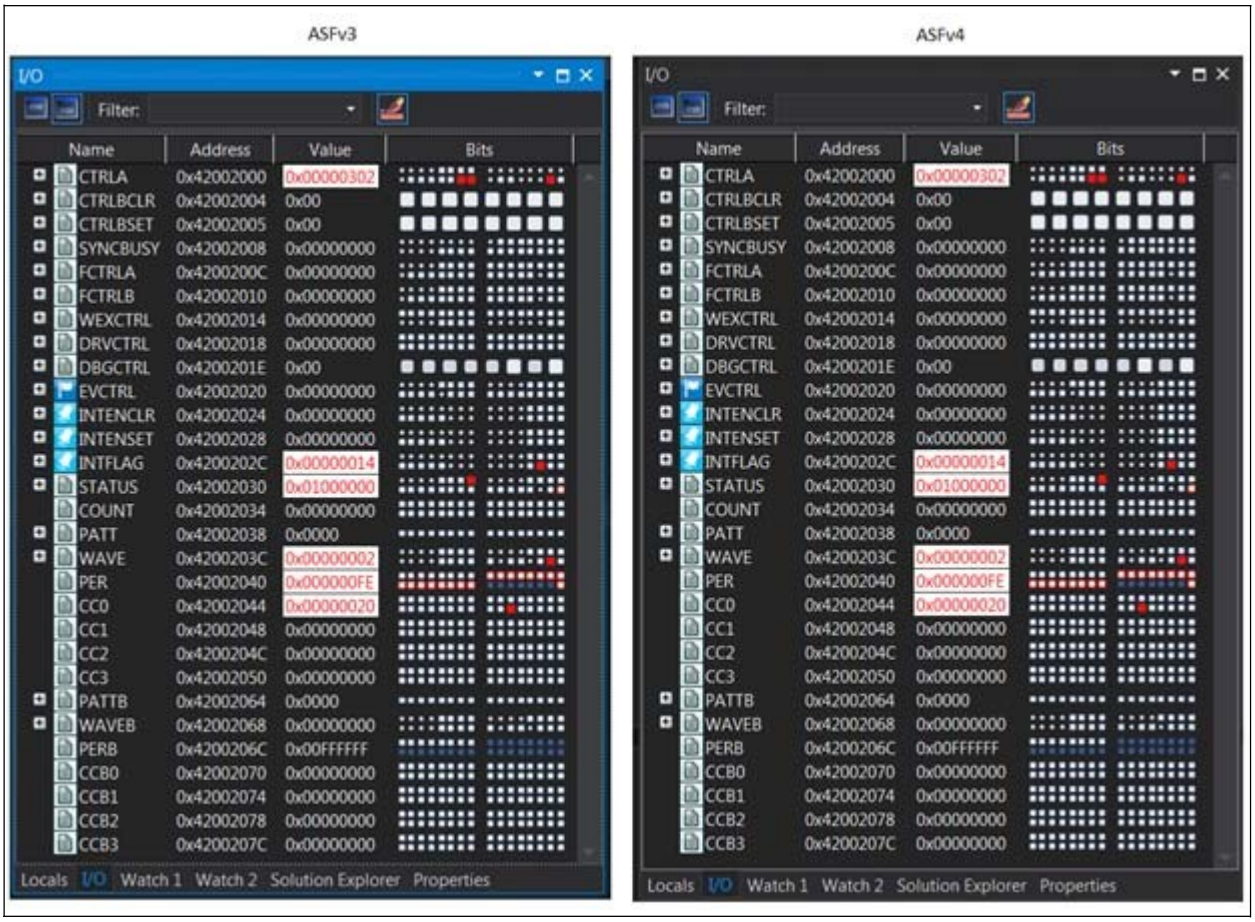
图27： ATMEL START——TCC0时钟配置器



验证系统配置

1. 按照上述说明导入并编译项目，比较ASFv3和ASFv4之间的寄存器设置。
2. 运行项目，直到TCC0外设完成初始化并使能。图28显示了ASFv3和ASFv4之间的初始化差别。根据图28中的比较结果可知，在ASFv4中配置TCC0模块初始化。

图28： ATMEL START——TCC0寄存器比较



应用程序更新

1. 应用程序说明。

   为了针对PWM占空比计算提供输入，只需实现ADC回调中的算法。ADC设置为收集单个12位样本，然后触发回调。在ADC回调中，将直接读取ADC结果寄存器并移位以提供8位值。该8位值将写入PWM的占空比。PWM配置为具有8位周期，因此占空比将相应调整。

2. 比较API。

   除Start设置提供的配置外，使用TCC0模块无需任何其他配置。ASFv3与之非常相似，其提供了一组默认值，但需要开发人员创建初始化函数并直接修改默认值。

   ASFv4在整个框架中采用更注重用例驱动的模型。通过在ASFv3中添加TCC组件以及在Start中添加PWM组件，即体现了这一模型的使用。ASFv4中PWM组件的API（`pwm_set_parameters`）也采用了该模型。而ASFv3使用`tcc_set_compare_value`来实现相同功能。传入这两个函数的参数也没有明显差异，但在ASFv4中实现了用例驱动方法并大幅简化了代码。

3. 回调设置。

   此项目中的TCC0模块不使用回调，但需在ADC回调函数中更改占空比。

结论

ASFv4是一个全新的框架，采用应用程序驱动的用例方法改善了代码长度和驱动程序效率。虽然ASFv4的架构与ASFv3有所不同，但与器件相关的命名约定和基本要素均有保留。

从ASFv3到ASFv4的转换没有一揽子解决方案。移植必须从头开始。虽然本文档仅涵盖了若干个功能差异，但提供了实现功能移植的基本原则。

附录 A： 应用代码示例

示例1： ASFv3项目扩展——ADC和PWM

```
#define ADC_SAMPLES 1
uint16_t adc_result_buffer[ADC_SAMPLES];

struct adc_module adc_instance;
struct tcc_module tcc_instance;

volatile bool adc_read_done = false;

void adc_complete_callback(struct adc_module *const module)
{
    uint32_t duty;

    duty = (adc_result_buffer[0] >> 4) & 0xFFFF;

    tcc_set_compare_value(&tcc_instance, (enum tcc_match_capture_channel)(CONF_PWM_CHANNEL),
duty);
}

void configure_adc(void)
{
    struct adc_config config_adc;

    adc_get_config_defaults(&config_adc);

    config_adc.clock_prescaler = ADC_CLOCK_PRESCALER_DIV8;
    config_adc.reference       = ADC_REFERENCE_INTVCC1;
    config_adc.positive_input  = ADC_POSITIVE_INPUT_PIN18;
    config_adc.resolution      = ADC_RESOLUTION_12BIT;

    adc_init(&adc_instance, ADC, &config_adc);
    adc_enable(&adc_instance);
}

static void configure_tcc(void)
{
    struct tcc_config config_tcc;

    tcc_get_config_defaults(&config_tcc, LED_0_PWM4CTRL_MODULE);

    config_tcc.counter.clock_prescaler = TCC_CLOCK_PRESCALER_DIV8;
    config_tcc.counter.period = 0xFE;
    config_tcc.compare.wave_generation = TCC_WAVE_GENERATION_SINGLE_SLOPE_PWM;
    config_tcc.compare.match[LED_0_PWM4CTRL_CHANNEL] = 0x7F;

    config_tcc.pins.enable_wave_out_pin[LED_0_PWM4CTRL_OUTPUT] = true;
    config_tcc.pins.wave_out_pin[LED_0_PWM4CTRL_OUTPUT]        = LED_0_PWM4CTRL_PIN;
    config_tcc.pins.wave_out_pin_mux[LED_0_PWM4CTRL_OUTPUT]    = LED_0_PWM4CTRL_MUX;

    tcc_init(&tcc_instance, LED_0_PWM4CTRL_MODULE, &config_tcc);

    tcc_enable(&tcc_instance);
}

void configure_adc_callbacks(void)
{
    adc_register_callback(&adc_instance, adc_complete_callback, ADC_CALLBACK_READ_BUFFER);
    adc_enable_callback(&adc_instance, ADC_CALLBACK_READ_BUFFER);
}

int main(void)
{
    system_init();
    delay_init();

    configure_adc();
    configure_adc_callbacks();
    configure_tcc();

    system_interrupt_enable_global();

    while (1) {
        adc_read_buffer_job(&adc_instance, adc_result_buffer, ADC_SAMPLES);
        delay_cycles_ms(100);
    }
}
```

示例2：　　ASFv3移植——仅限ADC

```
void adc_complete_callback(const struct adc_async_descriptor *const descr, const uint8_t
channel)
{
    static uint8_t i = 0;

    if (i < ADC_SAMPLES)
    {
        adc_result_buffer[i++] = ADC->RESULT.reg;
        adc_async_start_conversion(&ADC_0);
    }
    else
    {
        adc_read_done = true;
    }
}

int main(void)
{
    system_init();

    adc_async_register_callback(&ADC_0, 0, ADC_ASYNC_CONVERT_CB, adc_complete_callback);
    adc_async_enable_channel(&ADC_0, 0);

    adc_async_start_conversion(&ADC_0);

    while (adc_read_done == false)
    {
        /* Wait for asynchronous ADC reads to complete */
    }

    while(1)
    {
        asm("nop");
    }
}
```

示例3：　　项目扩展——ADC和PWM

```
#define PWM_PERIOD 254
static uint32_t pwm_duty;
static uint16_t adc_value;

static void adc_cb(const struct adc_async_descriptor *const descr, const uint8_t channel)
{
   adc_value = ADC->RESULT.reg;

   pwm_duty = (adc_value >> 4) & 0xFF;

   pwm_set_parameters(&PWM_0, PWM_PERIOD, pwm_duty);
}

int main(void)
{
   atmel_start_init();

   adc_async_register_callback(&ADC_0, 0, ADC_ASYNC_CONVERT_CB, adc_cb);

   ADC->CALIB.reg = ADC_CALIB_BIAS_CAL((*(uint32_t *)ADC_FUSES_BIASCAL_ADDR >>
ADC_FUSES_BIASCAL_Pos)) |
    ADC_CALIB_LINEARITY_CAL((*(uint64_t *)ADC_FUSES_LINEARITY_0_ADDR >>
ADC_FUSES_LINEARITY_0_Pos));

   adc_async_enable_channel(&ADC_0, 0);
   pwm_enable(&PWM_0);

   while(1)
   {
    adc_async_start_conversion(&ADC_0);
    delay_ms(100);
   }
}
```

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了 《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。Microchip 的 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品严格遵守公司的质量体系流程。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949═

ISBN：978-1-5224-3142-8

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://www.microchip.com/support
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

奥斯汀 Austin, TX
Tel: 1-512-257-3370

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Novi, MI
Tel: 1-248-848-4000

休斯敦 Houston, TX
Tel: 1-281-894-5983

印第安纳波利斯 Indianapolis
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453
Tel: 1-317-536-2380

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608
Tel: 1-951-273-7800

罗利 Raleigh, NC
Tel: 1-919-844-7510

纽约 New York, NY
Tel: 1-631-435-6000

圣何塞 San Jose, CA
Tel: 1-408-735-9110
Tel: 1-408-436-4270

加拿大多伦多 Toronto
Tel: 1-905-695-1980
Fax: 1-905-695-2078

亚太地区

中国 - 北京
Tel: 86-10-8569-7000

中国 - 成都
Tel: 86-28-8665-5511

中国 - 重庆
Tel: 86-23-8980-9588

中国 - 东莞
Tel: 86-769-8702-9880

中国 - 广州
Tel: 86-20-8755-8029

中国 - 杭州
Tel: 86-571-8792-8115

中国 - 南京
Tel: 86-25-8473-2460

中国 - 青岛
Tel: 86-532-8502-7355

中国 - 上海
Tel: 86-21-3326-8000

中国 - 沈阳
Tel: 86-24-2334-2829

中国 - 深圳
Tel: 86-755-8864-2200

中国 - 苏州
Tel: 86-186-6233-1526

中国 - 武汉
Tel: 86-27-5980-5300

中国 - 西安
Tel: 86-29-8833-7252

中国 - 厦门
Tel: 86-592-238-8138

中国 - 香港特别行政区
Tel: 852-2943-5100

中国 - 珠海
Tel: 86-756-321-0040

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600

台湾地区 - 新竹
Tel: 886-3-577-8366

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733

印度 India - Bangalore
Tel: 91-80-3090-4444

印度 India - New Delhi
Tel: 91-11-4160-8631

印度 India - Pune
Tel: 91-20-4121-0141

日本 Japan - Osaka
Tel: 81-6-6152-7160

日本 Japan - Tokyo
Tel: 81-3-6880-3770

韩国 Korea - Daegu
Tel: 82-53-744-4301

韩国 Korea - Seoul
Tel: 82-2-554-7200

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870

菲律宾 Philippines - Manila
Tel: 63-2-634-9065

新加坡 Singapore
Tel: 65-6334-8870

泰国 Thailand - Bangkok
Tel: 66-2-694-1351

越南 Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

芬兰 Finland - Espoo
Tel: 358-9-4520-820

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Garching
Tel: 49-8931-9700

德国 Germany - Haan
Tel: 49-2129-3766400

德国 Germany - Heilbronn
Tel: 49-7131-67-3636

德国 Germany - Karlsruhe
Tel: 49-721-625370

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

德国 Germany - Rosenheim
Tel: 49-8031-354-560

以色列 Israel - Ra'anana
Tel: 972-9-744-7705

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

意大利 Italy - Padova
Tel: 39-049-7625286

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

挪威 Norway - Trondheim
Tel: 47-7289-7561

波兰 Poland - Warsaw
Tel: 48-22-3325737

罗马尼亚 Romania - Bucharest
Tel: 40-21-407-87-50

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

瑞典 Sweden - Gothenberg
Tel: 46-31-704-60-40

瑞典 Sweden - Stockholm
Tel: 46-8-5090-4654

英国 UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820