Знакомство с новыми микроконтроллерами Smart ARM компании Atmel: работаем с библиотекой готовых решений ASF

Николай АРТЕМОВ
anv@efo.ru
Максим МАСЛОВСКИЙ
masma@efo.ru

Запуск нового проекта всегда связан с созданием решений по реализации необходимых требований. На это тратится значительная часть времени, отведенного на проект. Поэтому использование уже готовых решений может заметно сократить сроки разработки. Данная статья посвящена применению программной библиотеки ASF на примере нового сверхмалопотребляющего микроконтроллера Atmel ATSAM L21. В ней мы рассмотрим возможности ASF для работы с графическим дисплеем и встроенным аналого-цифровым преобразователем микроконтроллера.

В помощь разработчикам компания Atmel предоставляет бесплатную библиотеку программ ASF (Atmel Software Framework), которая содержит готовые программные решения с драйверами, стеками протоколов для обмена данными и сервисы общего назначения. ASF служит дополнением к традиционным компиляторам, таким как AVR-GCC и IAR, или может быть использована в составе интегрированной среды разработки Atmel Studio (рис. 1).

Библиотека ASF является результатом многолетнего труда специалистов, собравших и систематизировавших свои лучшие программные компоненты, созданные для микроконтроллеров Atmel. Тщательно формализовав программные модули, компания Atmel предоставила в свободный доступ огромный ресурс компонентов, ставший в действительности высококачественной, документированной, структурированной библиотекой. Работа с ASF ведется постоянно, она направлена на добавление «свежих» компонентов как для новых, так и для давно выпускаемых семейств микроконтроллеров, корректировку существующих, удаление или замену устаревших.

В ASF выделяют шесть типов компонентов, которые формируют многослойную иерархию современного адаптивного программного продукта:

- Boards — нижний слой иерархии, компоненты которого обеспечивают конфигурирование аппаратной части. Компоненты Boards позволяют выполнить первичную конфигурацию микроконтроллера, абстрагироваться на более высоких уровнях от наименований выводов за счет логических имен.
- Drivers — второй слой, чьи компоненты организуют удобный доступ к периферийным модулям микроконтроллера. Слой Drivers позволяет «отвязаться» от аппаратной части, что обеспечивает возможность на следующих уровнях использовать стороннее программное обеспечение, разработанное для других семейств микроконтроллеров.
- Components — один из верхних слоев, компоненты которого эффективно обеспечивают работу с внешними аппаратными модулями различных производителей, без глубокого изучения нюансов работы устройств. Компоненты данного уровня без труда позволят подключить и полноценно использовать LCD- и TFT-дисплеи, аудиокодеки, различные сенсоры.

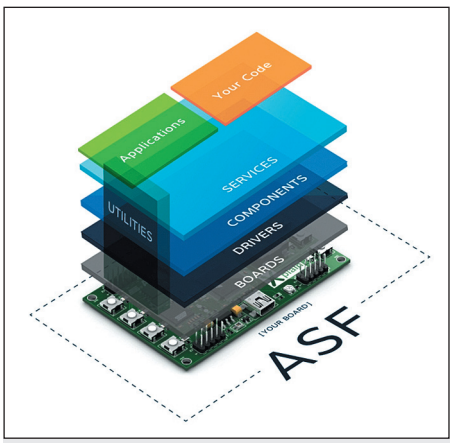

Рис. 1. Структура Atmel Software Framework (ASF)

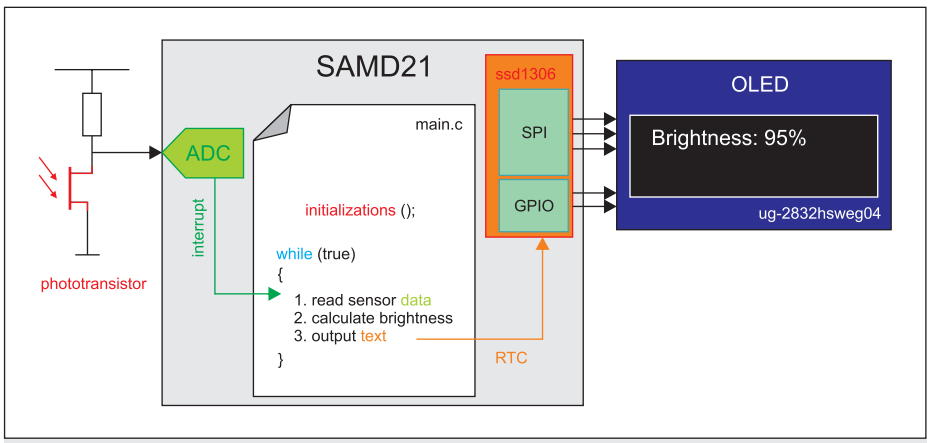

Рис. 2. Функциональная схема реализуемой программы

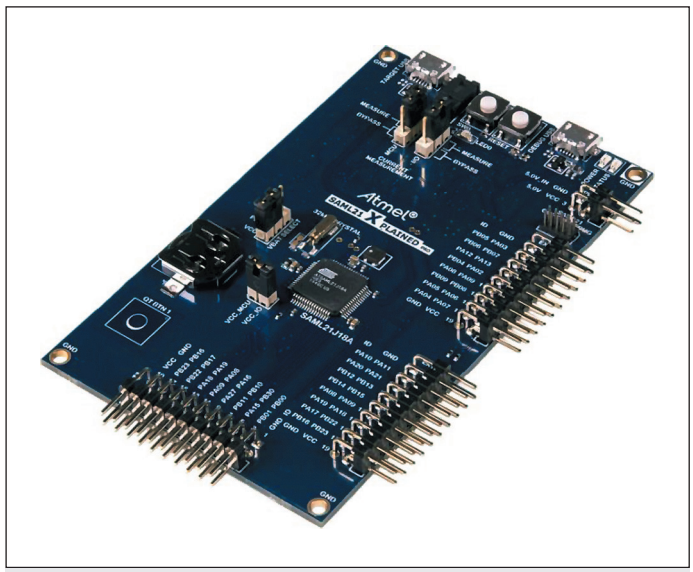

Рис. 3. Отладочная плата SAM L21 XPRO (ATSAML21J18A)

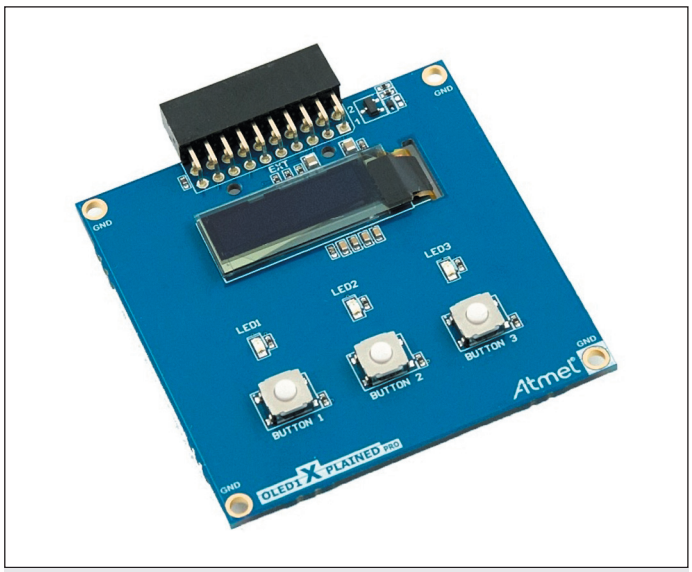

Рис. 4. Модуль расширения ATOLED1-XPRO

- Services — верхний слой, компоненты данного слоя обеспечивают доступ к высокоуровневым приложениям и службам. Благодаря этим компонентам можно с минимальными затратами ресурсов реализовать различные приложения типа Ethernet, ZigBee, USB, GUI и другие.
- Application — верхний слой, чьи компоненты представляют собой набор примеров, демонстрирующих оптимальные варианты решения распространенных задач для встраиваемых приложений.
- Utilities — служебный вертикальный слой, компоненты которого обеспечивают возможность использования ASF со сторонними средствами разработки, отладки и компиляции программных продуктов (IAR, GCC).

Для практического знакомства с готовыми решениями из библиотеки ASF создадим программу, в которой микроконтроллер будет считывать данные в аналоговой форме об освещенности с фотодатчика, оцифровывать их, вычислять освещенность в процентах и выводить информацию на графический дисплей (рис. 2).

Для этой программы нам потребуется задействовать:

- модуль аналого-цифрового преобразователя (ADC) для оцифровки данных от фотодатчика;
- последовательный интерфейс обмена данными в режиме SPI (SERCOM SPI) для передачи данных на графический дисплей;
- систему реального времени (RTC) для настройки периодов обновления данных на дисплее.

Для удобства обратимся к оценочной плате SAM L21 XPRO с микроконтроллером ATSAML21 и дочерними модулями ATOLED1-XPRO и ATIO1-XPRO (рис. 3–5).

Подключение дочерних плат к SAM L21 XPRO показано на рис. 6.

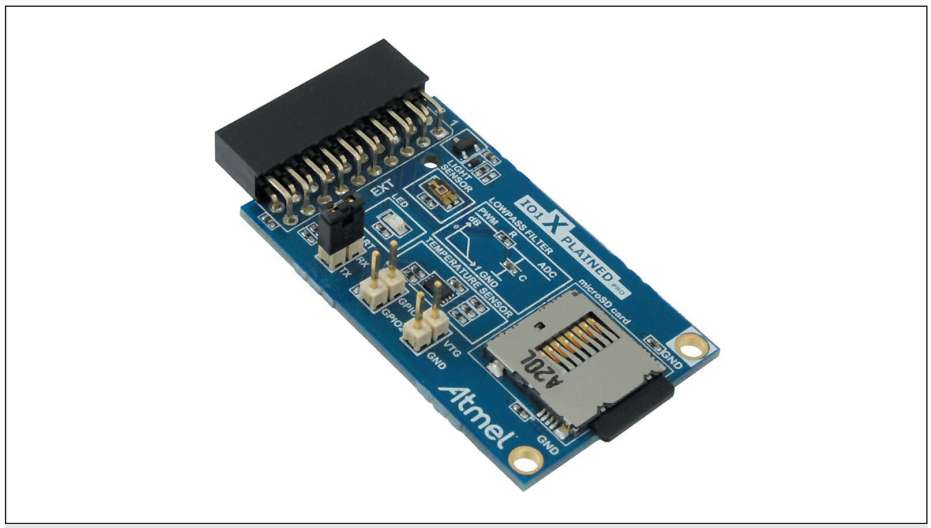

Рис. 5. Модуль расширения ATIO1-XPRO

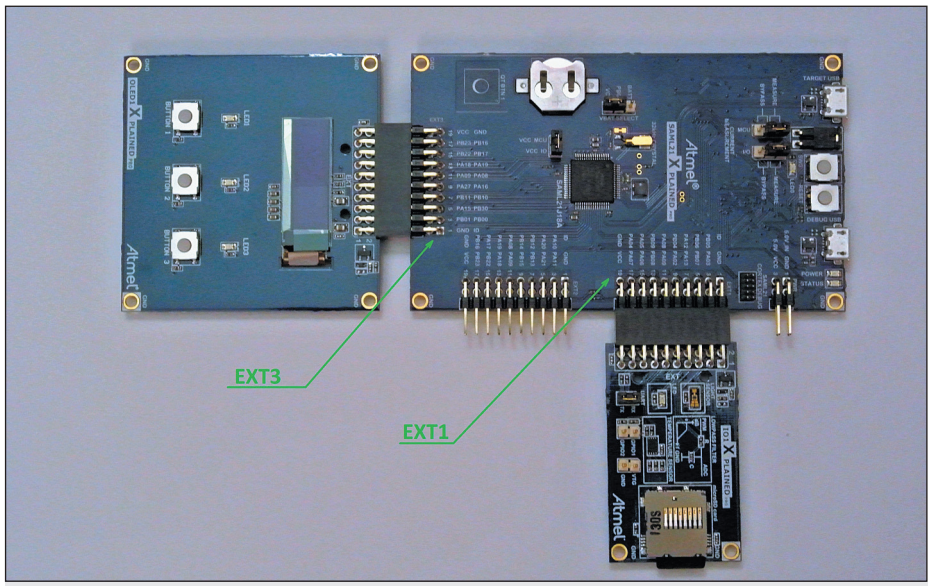

Рис. 6. Подключение дочерних модулей к отладочной плате SAM L21 XPRO (ATSAML21J18A)

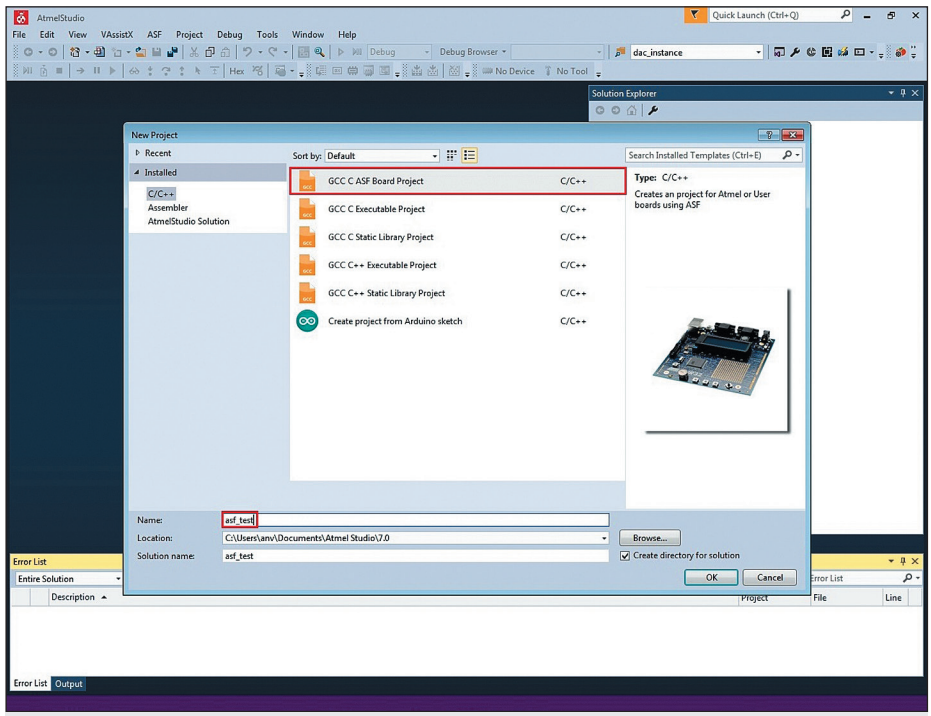

Рис. 7. Окно создания нового проекта в Studio 7

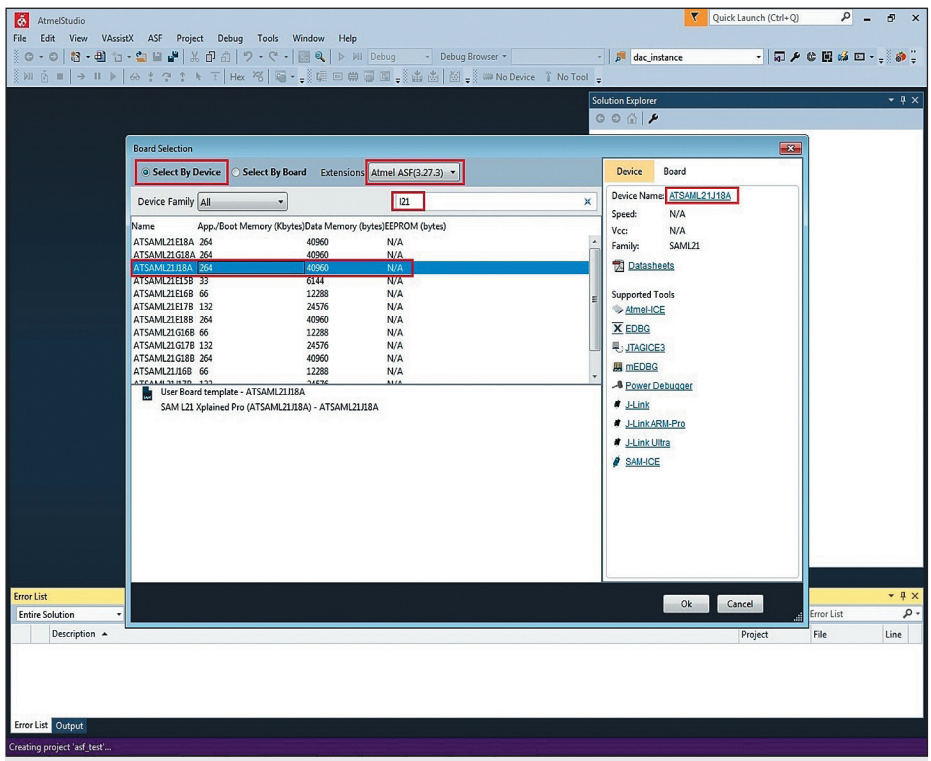

Рис. 8. Окно выбора целевого микроконтроллера проекта

Для написания и отладки программы воспользуемся интегрированной средой разработки IDE Atmel Studio 7 с уже встроенной библиотекой ASF 3.27. Запускаем Atmel Studio 7. Подключаем оценочную плату SAM L21 XPRO с коммутированными дочерними модулями. Для написания и отладки программы кабель micro-USB необходимо подсоединить к разъему DEBUG USB. После того как платы определятся в Atmel Studio, создаем новый проект GCC C ASF Board Project (**File/New/Projec**t) (рис. 7), указав имя нашего проекта в поле Name.

При создании нового проекта на протяжении всего процесса на экране будет присутствовать окно **Atmel Studio Creating project:**, на него не стоит обращать внимания — оно лишь сигнализирует о выполнении процесса.

Выбираем наш микроконтроллер ATSAML21J18A из списка (рис. 8), установив флаг **Select by Device**. Для быстрого поиска можно воспользоваться фильтром, указав ключевой фрагмент наименования **L21** или выбрав семейство SAML21. Выбранный контроллер и информация о нем будут отображаться справа от списка микроконтроллеров. Под списком также отображаются доступные для нашего кристалла отладочные средства. Проверяем, что в выпадающем списке Extensions выбрана наиболее «свежая» версия ASF — 3.27 или старше.

Выбираем необходимую целевую плату SAM L21 Xplained Pro (ATSAML21J18A) — ATSAML21J18A, предварительно установив флаг **Select by Board**. Если этот шаг не выполнить, Atmel Studio не подгрузит конфигурационный файл для нашей отладочной платы! Для более быстрого поиска можно также воспользоваться фильтром по ключевому фрагменту в названии L21 или выбрать в выпадающем списке Board Types значение Atmel. Под списком отладочных плат отображается установленный на данном устройстве микроконтроллер (рис. 9). Выбрав микроконтроллер и отладочную плату и нажав клавишу **OK**, мы заканчиваем создание проекта.

После создания проекта отобразится окно **ASF Wizard**, где мы можем подключить необходимые нам компоненты из окружения ASF. Для этого необходимо выбрать наш проект в выпадающем списке Project:, после чего программа подгрузит все компоненты, доступные для нашего микроконтроллера и отладочной платы (рис. 10).

В правой части окна отображаются уже подключенные к проекту компоненты, слева — список доступных компонентов. Подключим компонент для работы с RTC. Для этого находим **RTC-Real Timer Counter Driver (driver)**, нажимаем клавишу **Add**. Настраиваем компонент так, чтобы он вызывал прерывание. Для этого в выпадающем списке справа от выбранного компонента выбираем **count_callback** и нажимаем клавишу **Apply** для загрузки компонента в проект. Таким же образом подключим компонент GFX Monochrome-System Font (service) к нашему проекту.

Настройка контроллера для работы с установленным на ATOLED1-XPRO графическим дисплеем UG-2832HSWEG04 с контроллером SSD1306 потребует загрузить в проект компонент GFX Monochrome — Monochrome Graphic library (service). Находим в списке интересующий нас элемент, нажимаем **Add**. Теперь компонент нужно настроить под нашу аппаратную часть. Раскрываем GFX Monochrome — Monochrome Graphic library (service), нажав на треугольник слева от компонента, выбираем тип драйвера **ug_2832hsweg04** справа от GFX Monochrome — Display driver (service). Затем указываем режим работы периферийного модуля SERCOM по запросу, для это-

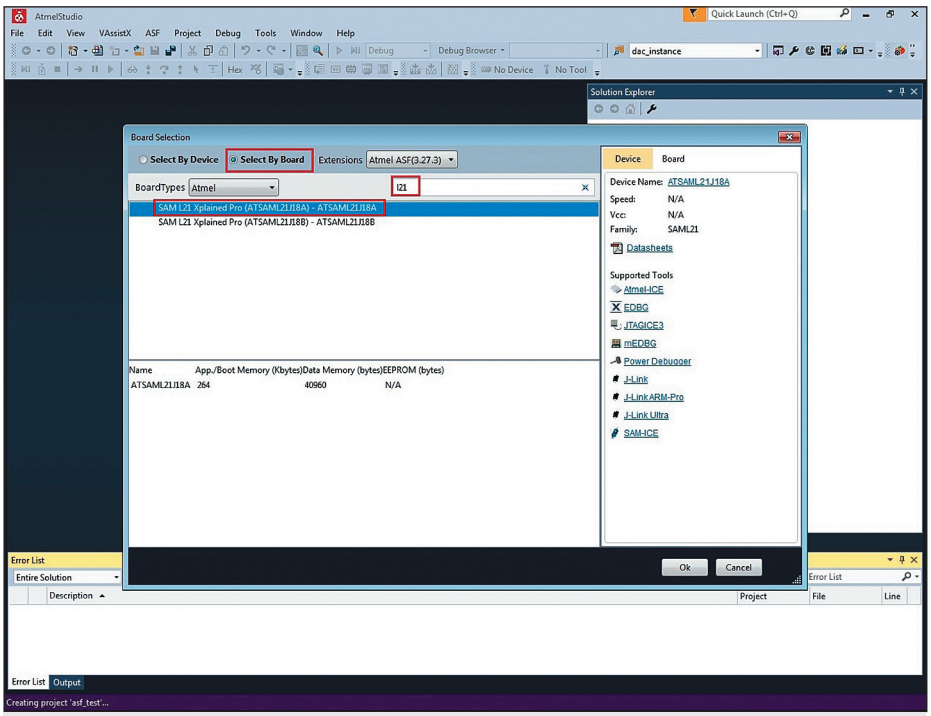

Рис. 9. Окно выбора целевой отладочной платы

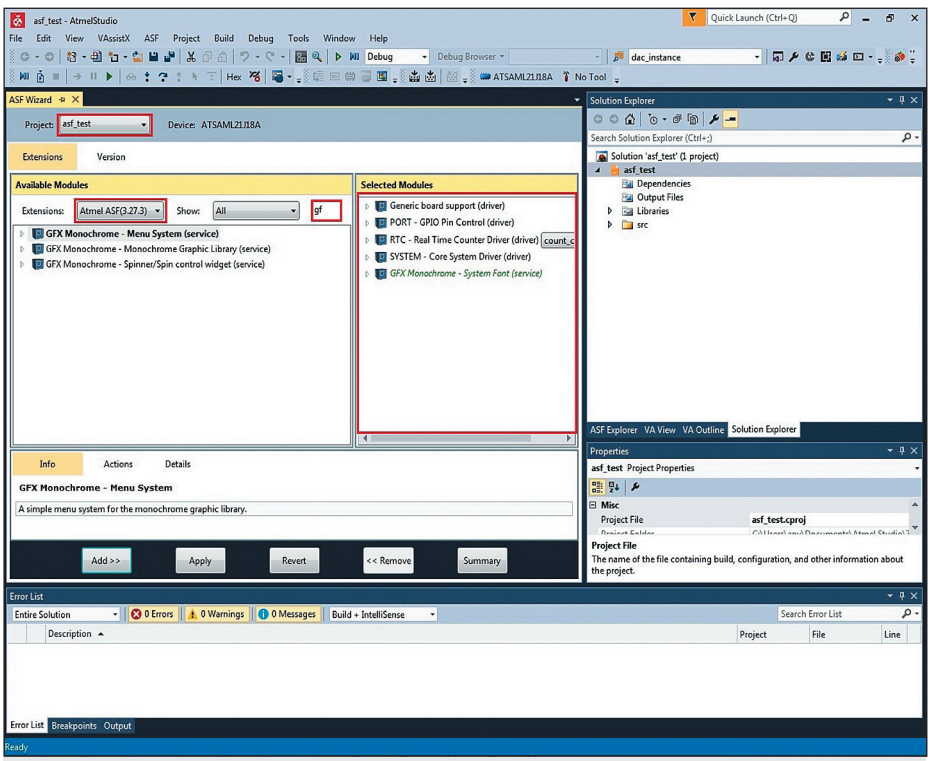

Рис. 10. Окно ASF Wizard

го находим драйвер SERCOM SPI — Serial Peripheral Interface (driver), который имеется в структуре компонента GFX Monochrome — Display driver (service), и в выпадающем списке справа от него выбираем **polled**. Обновляем весь проект при помощи клавиши **Apply** (рис. 11).

После того как все необходимые компоненты ASF добавлены в проект, открываем файл создаваемой программы *main.c*. Данный файл можно найти в правой части экрана, в окне **Solution Explorer**, в папке src.

Перед началом написания самой программы разберемся, как работать с периферийными модулями, используя компоненты ASF. Работа с периферийными модулями посредством ASF состоит из пяти последовательных шагов:

- создание структуры для хранения параметров периферийного модуля;
- загрузка настроек по умолчанию периферийного модуля в созданную структуру;
- настройка параметров периферийного модуля под конкретный проект (корректируем данные в структуре);
- загрузка измененных настроек в регистры микроконтроллера;
- активация периферийного модуля.

Данный подход касается большинства периферийных модулей, за исключением портов ввода/вывода и модуля тактирования при работе с отладочной или оценочной платой Atmel, когда основная конфигурация уже создана разработчиками компании Atmel и вы загружаете ее в создаваемый проект вместе с остальными компонентами ASF, при этом остается только инициализировать систему. Документация на компоненты ASF доступна по адресу: www.asf.atmel.com.

Приступаем к написанию самой программы. Создаем функцию ***port_init()*** инициализации порта для индикации работы программы:

```
void port_init(void)
{
struct port_config pin_conf;
pin_conf.direction = PORT_PIN_DIR_OUTPUT;
port_pin_set_config(LED1, &pin_conf);
}
```

В теле функции объявляем структуру pin_conf для хранения настроек порта:

```
struct port_config pin_conf;
```

Настраиваем порт как выход:

```
pin_conf.direction = PORT_PIN_DIR_OUTPUT;
```

Записываем настройки в регистры контроллера:

```
port_pin_set_config(LED1, &pin_conf);
```

Задаем в самом начале нашей программы, в начале файла *main.c*, идентификатор:

```
#define LED1 EXT3_PIN_7
```

Для того чтобы программа периодически обновляла данные на графическом дисплее, необходимо сконфигурировать систему реального времени (RTC), настроить систему прерываний и написать функцию обработки данного прерывания. Объявляем глобальную структуру для работы с RTC:

```
struct rtc_module rtc_instance;
```

Создаем функцию ***configure_rtc_count()***, конфигурирующую RTC:

```
void configure_rtc_count(void)
{
struct rtc_count_config config_rtc_count;
rtc_count_get_config_defaults(&config_rtc_count);
config_rtc_count.prescaler = RTC_COUNT_PRESCALER_DIV_1;
config_rtc_count.mode = RTC_COUNT_MODE_16BIT;
config_rtc_count.compare_values[0] = 1000;
rtc_count_init(&rtc_instance, RTC, &config_rtc_count);
rtc_count_enable(&rtc_instance);
}
```

В теле данной функции объявляем структуру config_rtc_count типа rtc_count_config для хранения настроек RTC и считываем в нее настройки по умолчанию:

```
struct rtc_count_config config_rtc_count;
rtc_count_get_config_defaults(&config_rtc_count);
```

Меняем настройки RTC: настраиваем делитель частоты, устанавливаем 16-битный режим работы, загружаем в регистр compare значение 1000:

```
config_rtc_count.prescaler = RTC_COUNT_PRESCALER_DIV_1;
config_rtc_count.mode = RTC_COUNT_MODE_16BIT;
config_rtc_count.compare_values[0] = 1000;
```

Записываем настройки из структуры в регистры микроконтроллера и активируем работу RTC:

```
rtc_count_init(&rtc_instance, RTC, &config_rtc_count);
rtc_count_enable(&rtc_instance);
```

Настраиваем работу прерывания по событию «Переполнение RTC», для этого создаем конфигурирующую функцию ***config_rtc_callbaks()***:

```
void config_rtc_callbaks(void)
{
rtc_count_set_period(&rtc_instance,20000);
rtc_count_set_count(&rtc_instance, 0);
rtc_count_register_callback(&rtc_instance,rtc_overflow_callback,
RTC_COUNT_CALLBACK_OVERFLOW);
rtc_count_enable_callback(&rtc_instance, RTC_COUNT_
CALLBACK_OVERFLOW);
}
```

В теле функции устанавливаем период RTC и начальное значение для счета:

```
rtc_count_set_period(&rtc_instance,20000);
rtc_count_set_count(&rtc_instance, 0);
```

Настраиваем событие, по которому должно происходить прерывание (в нашем случае это переполнение), и назначаем функцию ***rtc_overflow_callback()*** для обработки данного прерывания:

```
rtc_count_register_callback(&rtc_instance,rtc_overflow_
callback,RTC_COUNT_CALLBACK_OVERFLOW);
```

Активируем прерывание по переполнению RTC:

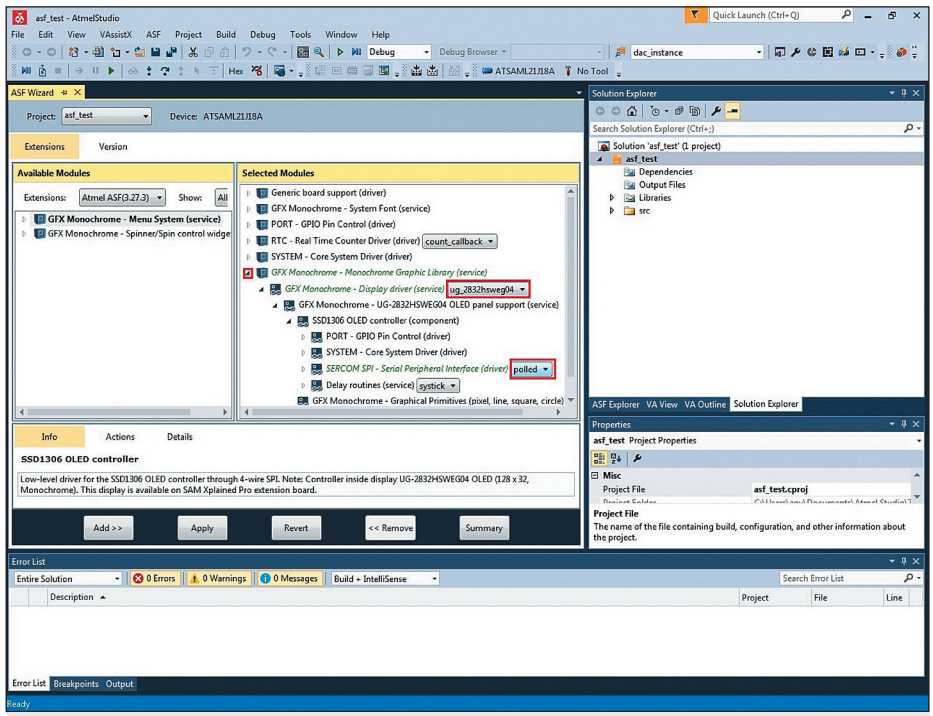

Рис. 11. Загрузка и настройка компонентов ASF

```
rtc_count_enable_callback(&rtc_instance, RTC_COUNT_
CALLBACK_OVERFLOW);
```

Создаем саму функцию ***rtc_overflow_callback()*** обработки прерывания:

```
void rtc_overflow_callback(void)
{
port_pin_toggle_output_level(LED_0_PIN);
oled_refresh_ready=true;
}
```

Данная функция меняет состояние светодиода, что будет сигнализировать о работе прерывания и программы в целом, и устанавливает значение глобальной переменной oled_refresh_ready. Эта переменная является для нас флагом, по которому мы будем обновлять информацию на дисплее. Глобальную переменную oled_refresh_ready типа bool нужно объявить в самом начале нашей программы.

Следующим шагом реализуем часть программы, отвечающую за аналого-цифровые преобразования данных, поступающих от датчика освещенности. Для этого создаем глобальный массив adc_result_buffer [ADC_SAMPLES] типа uint16_t для хранения результатов измерений. В нашем случае массив вырожденный и имеет размер #define ADC_SAMPLES 1.

Объявляем глобальную структуру adc_instance типа adc_module для работы с ADC:

```
struct adc_module adc_instance;
```

Создаем функцию ***light_sensor_init()***, конфигурирующую ADC:

```
void light_sensor_init(void)
{
struct adc_config config_adc;
adc_get_config_defaults(&config_adc);
config_adc.clock_prescaler = ADC_CLOCK_PRESCALER_DIV64;
config_adc.reference = ADC_REFERENCE_INTVCC1;
config_adc.positive_input = ADC_POSITIVE_INPUT_PIN13;
config_adc.resolution = ADC_RESOLUTION_12BIT;
adc_init(&adc_instance, ADC, &config_adc);
adc_enable(&adc_instance);
adc_register_callback(&adc_instance, adc_complete_callback, ADC_
CALLBACK_READ_BUFFER);
adc_enable_callback(&adc_instance, ADC_CALLBACK_READ_
BUFFER);
}
```

В теле данной функции объявляем структуру config_adc для хранения настроек ADC и записываем в нее настройки по умолчанию:

```
struct adc_config config_adc;
adc_get_config_defaults(&config_adc);
```

Меняем настройки делителя частоты, настраиваем источник опорного напряжения:

```
config_adc.clock_prescaler = ADC_CLOCK_PRESCALER_DIV64;
config_adc.reference = ADC_REFERENCE_INTVCC1;
```

Выбираем канал ADC и настраиваем разрешение 12 бит:

```
config_adc.positive_input = ADC_POSITIVE_INPUT_PIN13;
config_adc.resolution = ADC_RESOLUTION_12BIT;
```

Записываем новые настройки в регистры микроконтроллера и активируем работу ADC:

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

Теперь настраиваем прерывание ADC. Программа будет забирать данные из буфера ADC после окончания преобразования данных. Настраиваем прерывание и назначаем функцию ***adc_complete_callback()*** для обработки прерывания:

```
adc_register_callback(&adc_instance, adc_complete_callback, ADC_CALLBACK_READ_BUFFER);
```

Активируем прерывание:

```
adc_enable_callback(&adc_instance, ADC_CALLBACK_READ_BUFFER);
```

Создаем функцию ***adc_complete_callback()*** обработки прерывания, в теле которой считываем данные из буфера ADC, и устанавливаем переменную adc_read_done, указывающую, что данные актуальны:

```
void adc_complete_callback (void)
{
adc_read_buffer_job(&adc_instance, adc_result_buffer, ADC_SAMPLES);
adc_read_done = true;
}
```

Далее создаем функцию для вычисления и вывода на дисплей данных об освещенности. Предварительно объявляем глобальную переменную brightness типа uint16_t для хранения значений освещенности. Объявляем идентификаторы LOWER_BOUND и UPPER_BOUND, задающие границы значений параметра brightness:

```
#define UPPER_BOUND 4000
#define LOWER_BOUND 200
```

Объявляем переменную disp_string для хранения данных об освещенности в строковом виде:

```
char disp_string[10];
```

Создаем функцию ***display_result()***:

```
void display_result (void)
{
if(adc_read_done==true)
{
brightness=4096-adc_result_buffer[0];
if(brightness<LOWER_BOUND)
{
gfx_mono_draw_string(“Min”,76,5,&sysfont);
    }
    if(brightness>UPPER_BOUND)
    {
    gfx_mono_draw_string(“Max”,76,5,&sysfont);
    }
    if ((brightness>=LOWER_BOUND)&&(brightness<=UPPER_BOUND))
    {
    snprintf(disp_string, APP_STR_LENGHT, “%4d%%”,(brightness*100-LOWER_BOUND*100)/(UPPER_BOUND-LOWER_BOUND));
    gfx_mono_draw_string(disp_string, 64, 5, &sysfont);
    }
    adc_read_done=false;
}
}
```

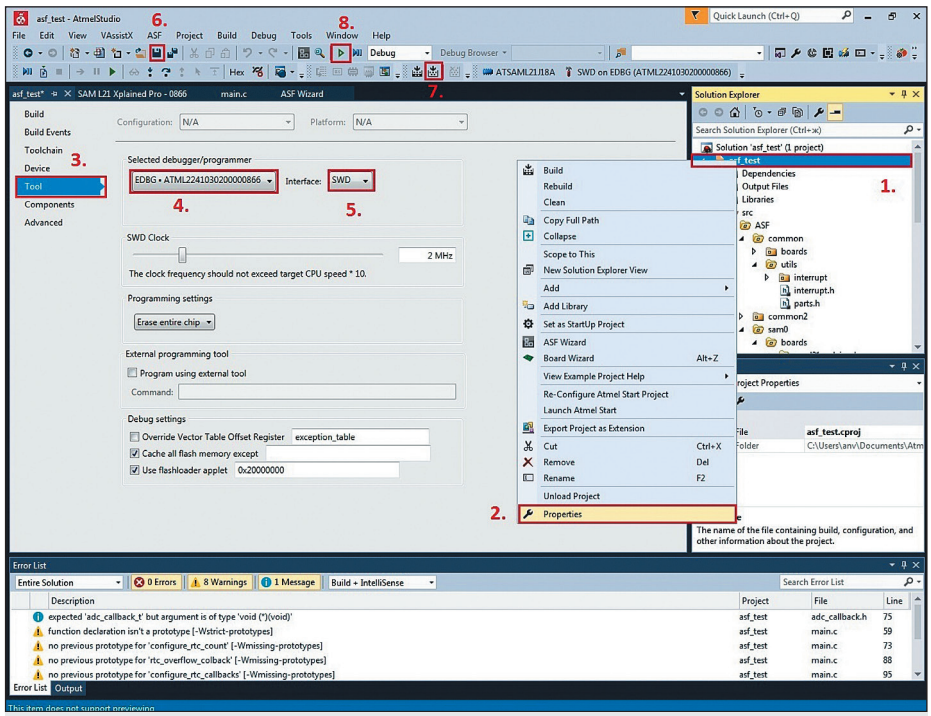

Рис. 12. Загрузка программы в микроконтроллер

В теле функции проверяем актуальность данных (adc_read_done==true). Если данные актуальные, тогда вычисляем освещенность brightness:

```
brightness=4096-adc_result_buffer[0];
```

Проверяем, выходит ли вычисленное значение brightness за нижнюю границу. Если да, то выводим на дисплей сообщение “Min”:

```
if(brightness<LOWER_BOUND)
{
gfx_mono_draw_string(“Min”,76,5,&sysfont);
}
```

Проверяем, выходит ли значение параметра brightness за верхний предел. Если да, выводим на дисплей сообщение “Max”:

```
if(brightness>UPPER_BOUND)
{
gfx_mono_draw_string(“Max”,76,5,&sysfont);
}
```

Если значение лежит в диапазоне от LOWER_BOUND до UPPER_BOUND, тогда преобразуем данные, хранящиеся в переменной brightness, в строковый тип и выводим их на дисплей:

```
snprintf(disp_string, APP_STR_LENGHT, “%4d%%”,(brightness*100 LOWER_BOUND*100)/(UPPER_BOUND-LOWER_BOUND));
gfx_mono_draw_string(disp_string, 64, 5, &sysfont);
```

В конце функции снимаем флаг актуальности данных:

```
adc_read_done=false;
```

Теперь все необходимые нам сервисные функции реализованы, приступаем к написанию основной программы. Инициализируем систему, порты, RTC:

```
system_init();
port_init();
config_rtc_callbaks();
```

Вызываем нашу функцию конфигурирования ADC:

```
light_sensor_init();
```

Делаем самостоятельно первое чтение из буфера ADC:

```
adc_read_buffer_job(&adc_instance, adc_result_buffer, ADC_SAMPLES);
```

Настраиваем графический интерфейс и выводим статические данные: рамку и надпись “Brightness:”:

```
gfx_mono_init();
gfx_mono_draw_rect(0,0,128,32,GFX_PIXEL_SET);
gfx_mono_draw_string(“Brightness:”,5, 5, &sysfont);
```

Разрешаем глобальное прерывание:

```
system_interrupt_enable_global();
```

Далее в основном цикле while (true) постоянно проверяется состояние нашего флага oled_refresh_ready. Если он установлен, обновляем данные на дисплее при по-

мощи функции ***display_result()*** и сбрасываем флаг:

```
while (true)
{
if (oled_refresh_ready == true)
{
display_result();
oled_refresh_ready = false;
}
}
```

Программа готова, можем загрузить ее в микроконтроллер и проверить работоспособность приложения.

Для того чтобы загрузить программу в микроконтроллер в новом проекте, необходимо сделать восемь последовательных шагов, показанных на рис. 12:

- открыть окно со свойствами проекта, щелкнув в окне **Solution Explorer** правой кнопкой мыши по нашему проекту, и выбрать в контекстном меню пункт Properties (шаги 1 и 2);
- в свойствах проекта зайти в раздел Tools (шаг 3);
- выбрать в окне **Selected Debugger/Programmer** тип программатора EDBG и тип интерфейса SWD в выпадающем списке Interface (шаги 4 и 5);
- сохранить настройки, нажав сочетание клавиш **Ctrl+S** или кликнув на иконку на панели инструментов (шаг 6);
- скомпилировать проект, нажав клавишу **F7** или кликнув на иконку на панели инструментов (шаг 7);
- загрузить программу в микроконтроллер, нажав сочетание клавиш **Ctrl+Alt+F5** или кликнув на иконку на панели инструментов (шаг 8).

Полученный результат представлен на рис. 13.

Рабочее окружение разработчика ASF является мощным инструментом для создания адаптивных, трудоемких, сложных приложений. С выходом онлайн-утилиты Atmel Start, являющейся дальнейшим развитием ASF, процесс создания стал еще проще и понятнее. Следует отметить, что сегодня компания Atmel в корне меняет интерфейс взаимодействия разработчика со средой ASF. В классическом варианте ASF будет существовать только в версиях 3.XX. Более новые версии, начиная с 4.0, интегрированы в программное обеспечение Atmel Start. Проект, выполненный в Atmel Start и содержащий необходимые для разработчика программные компоненты и конфигурацию аппаратной части, может быть импортирован в бесплатную среду Atmel Studio или в среду разработки от стороннего производителя (IAR Embedded Workbench, Keil µVision). Благодаря высокой степени интегрированности, предоставляемой ASF и Atmel Start, освоение новых микроконтроллеров, переход от семейства к семейству значительно облегчается. В следующих статьях мы продолжим знакомить читателей с особенностями работы ASF на примере различных микроконтроллеров Atmel.

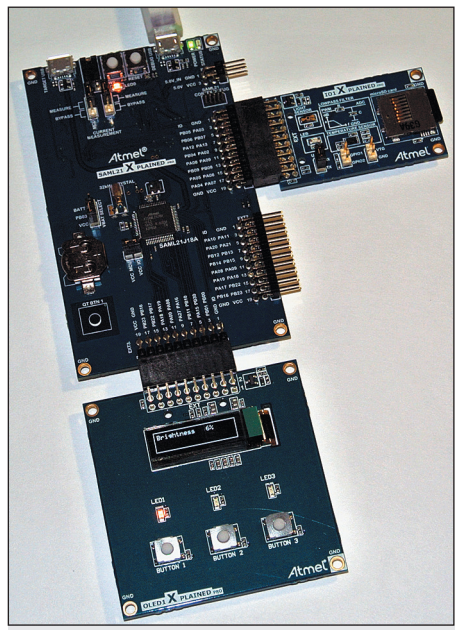

Рис. 13. Собранный стенд с загруженной программой