How to Use the ADC On the MCU ATtiny1626 with Code

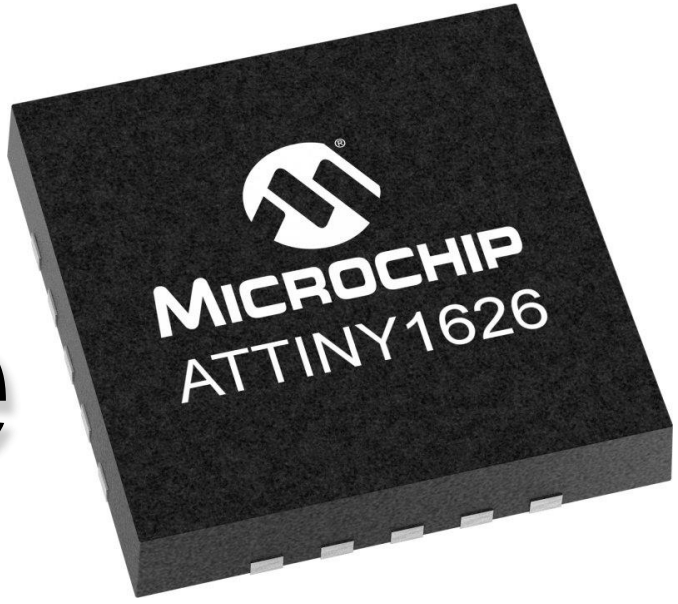


Table of Content

Table of Content (continue)

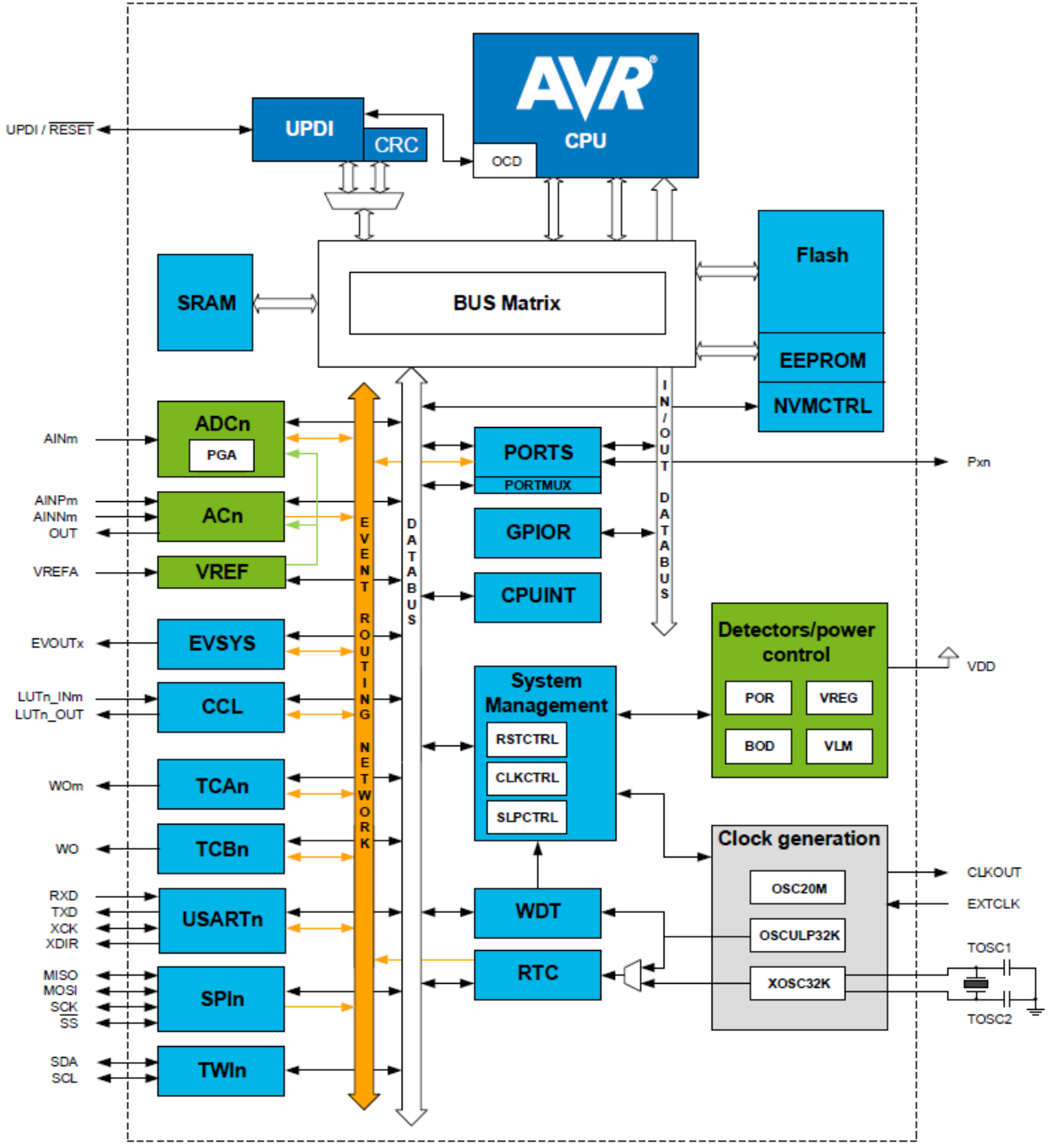
Overview

This tutorial will guide you through the process of create a project using Atmel Studio 7 IDP.

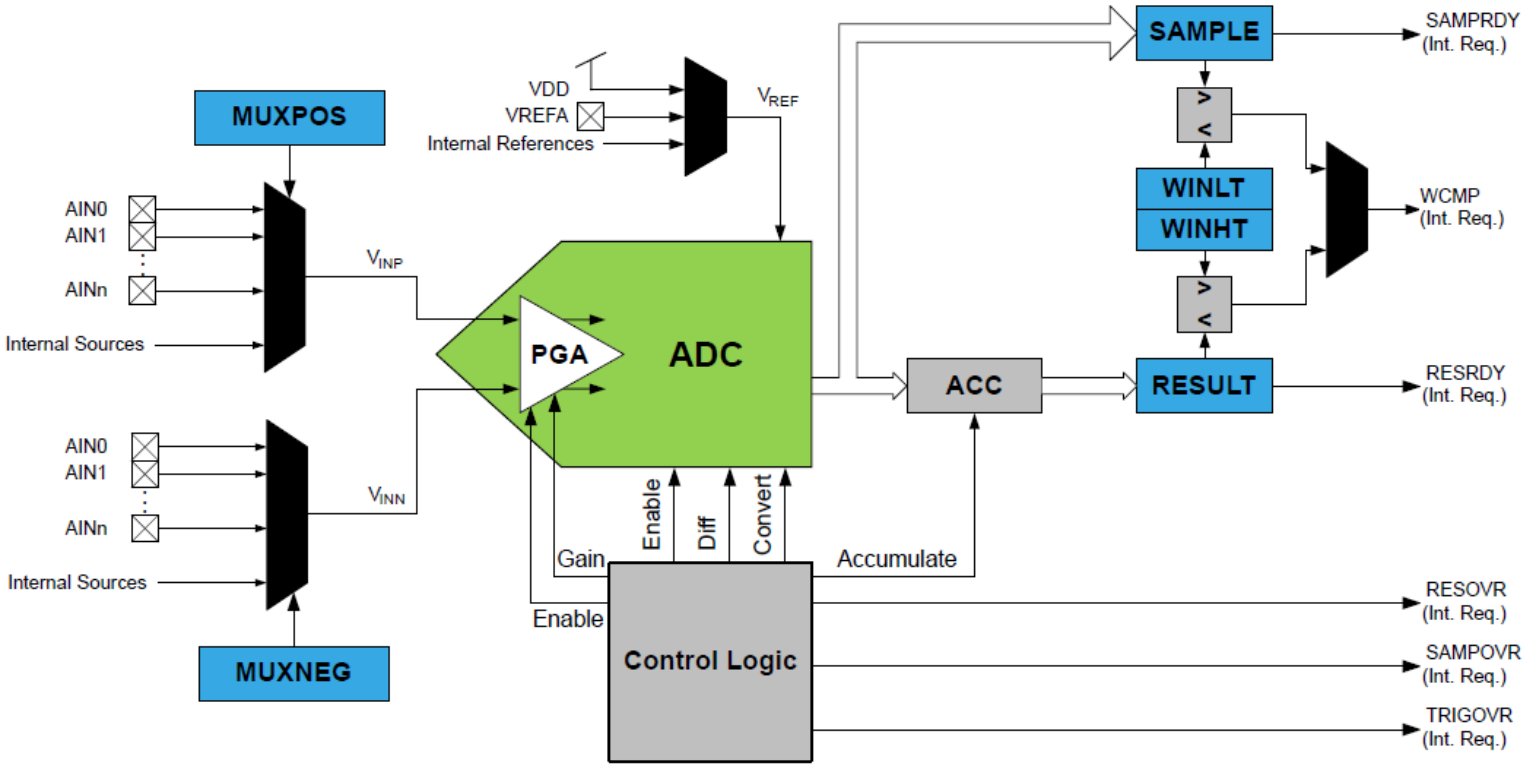
Atmel® Studio 7 is the integrated development platform (IDP) for developing and debugging SMART ARM®-based and AVR® microcontroller (MCU) applications. Studio 7 supports all AVR and SMART MCUs. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. It also connects seamlessly to Atmel debuggers and development kits.

The project is a simple, but intuitive example designed to shows how to configure and use the ADC of the ATtiny1626 MCU to measure the voltage from three different channels.

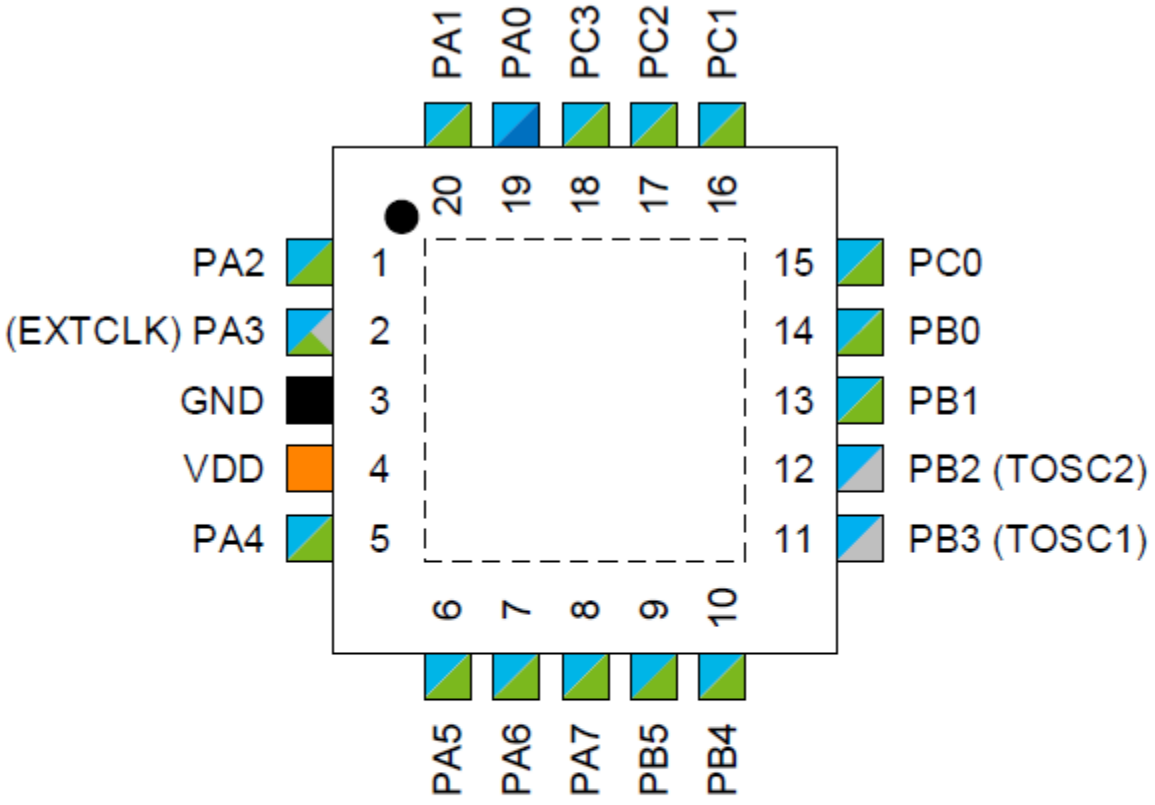
MCU ATtiny1626 Block Diagram



ADC Block Diagram



MCU ATtiny1626 (20-Pin VQFN) Pinout



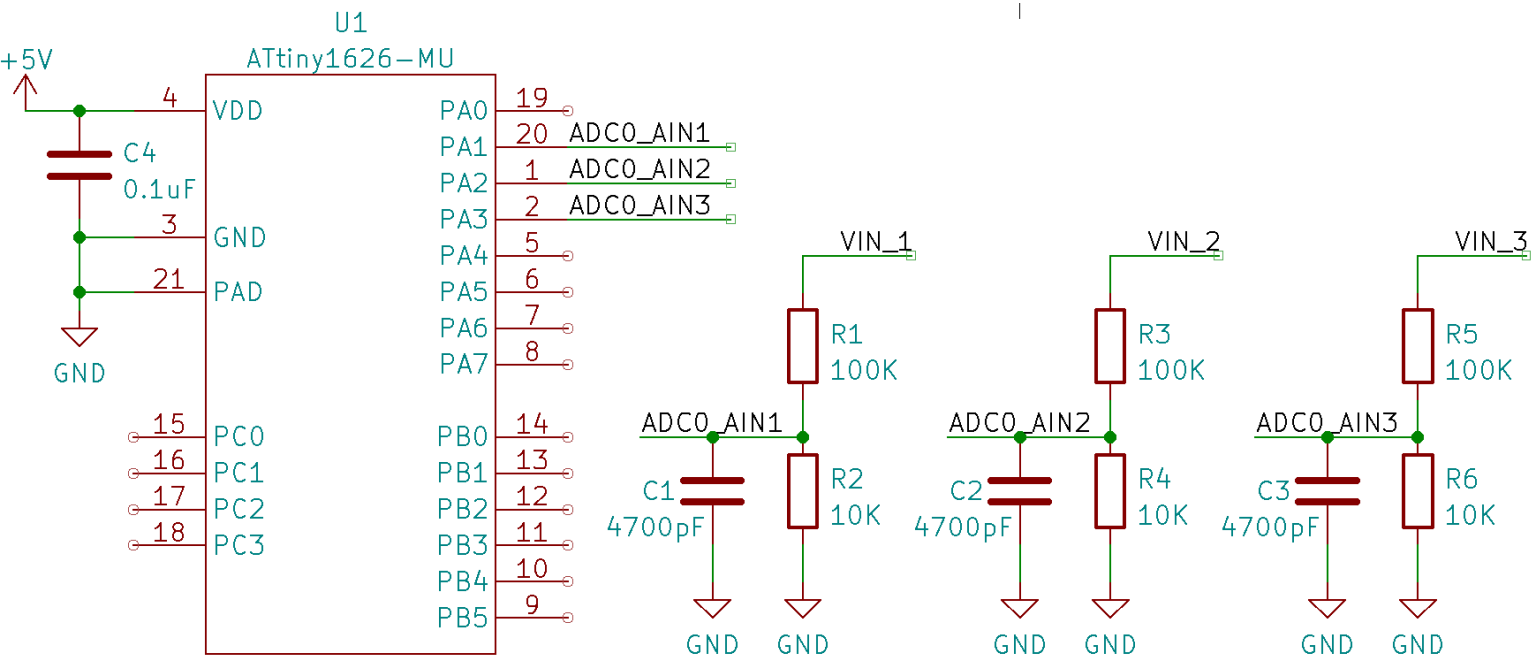
Power

- Power Supply
- Ground
- Pin on VDD Power Domain

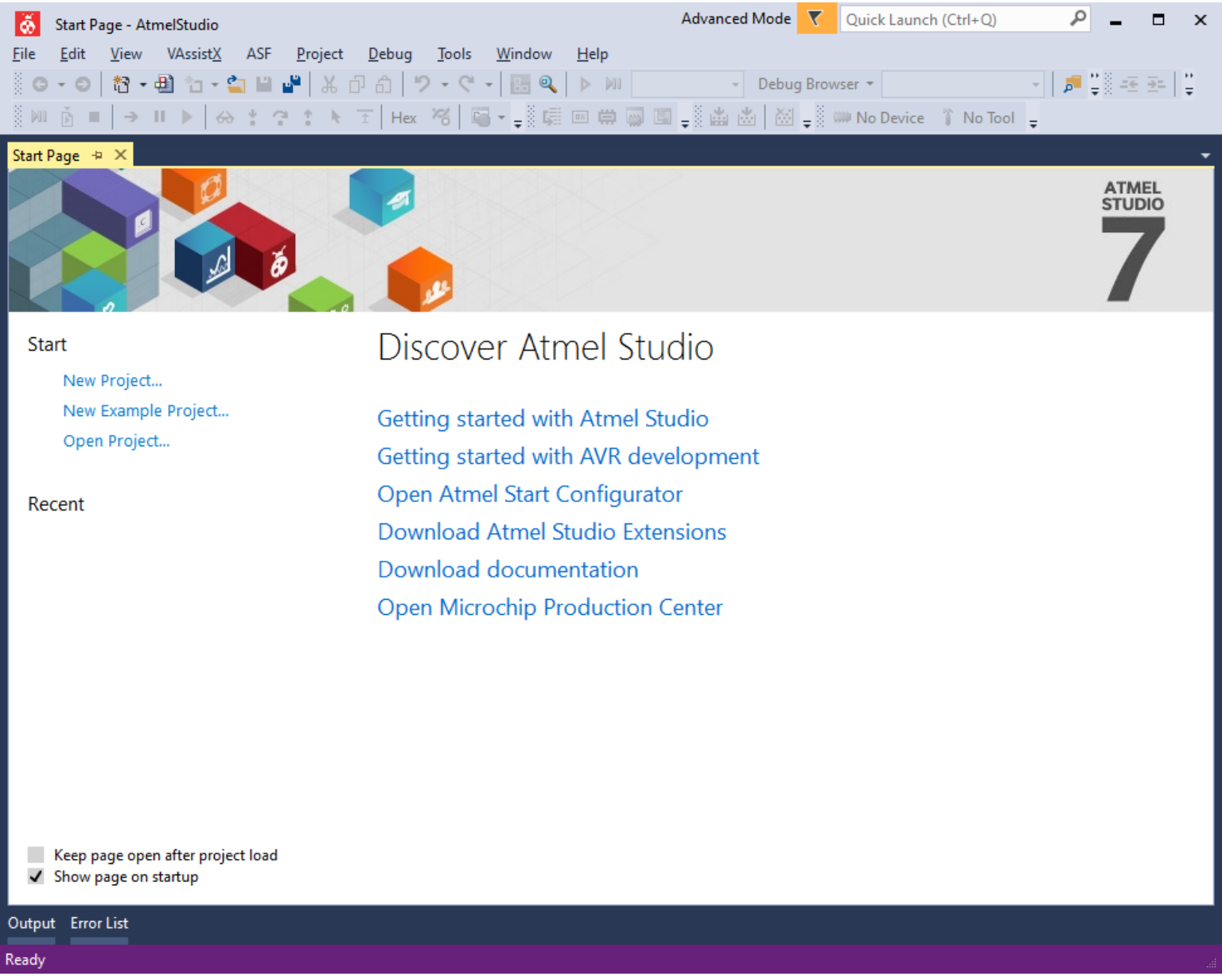
Functionality

- Programming/Debug
- Clock/Crystal
- Analog Function

Project Schematic

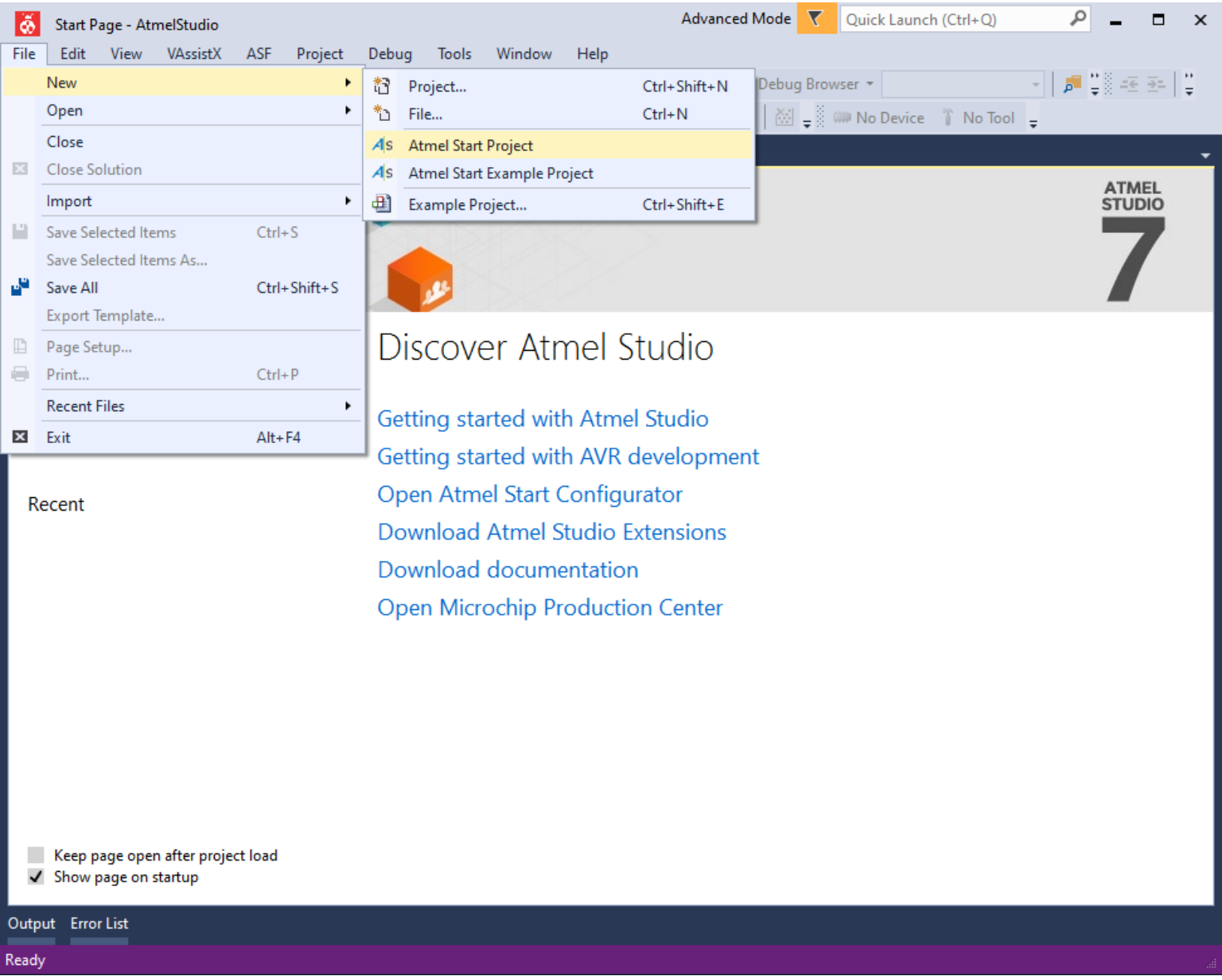


Atmel Studio 7 IDP

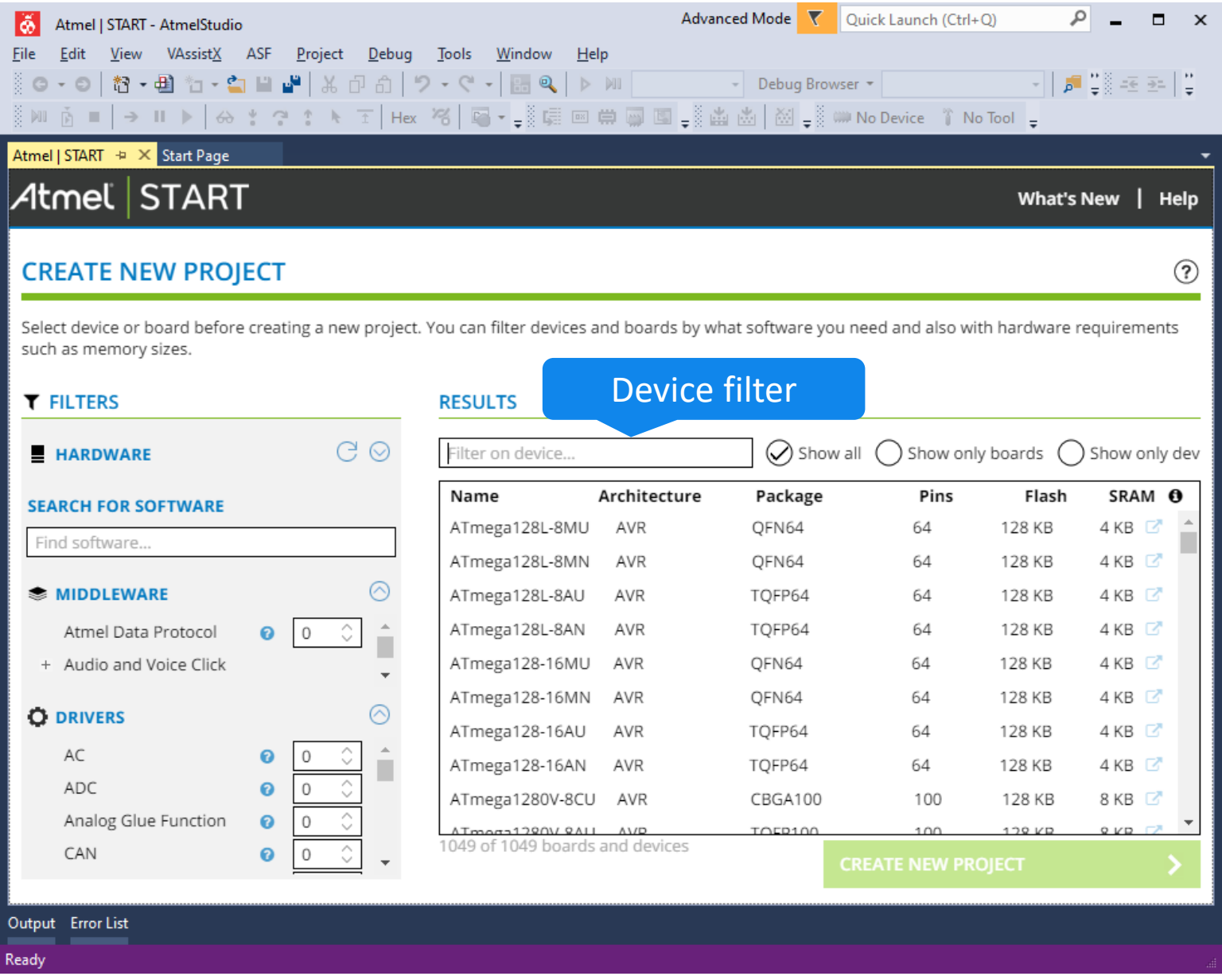


Atmel® Studio 7 is the integrated development platform (IDP) for developing and debugging microcontroller (MCU) applications.

Create a New Project using Atmel Start Project

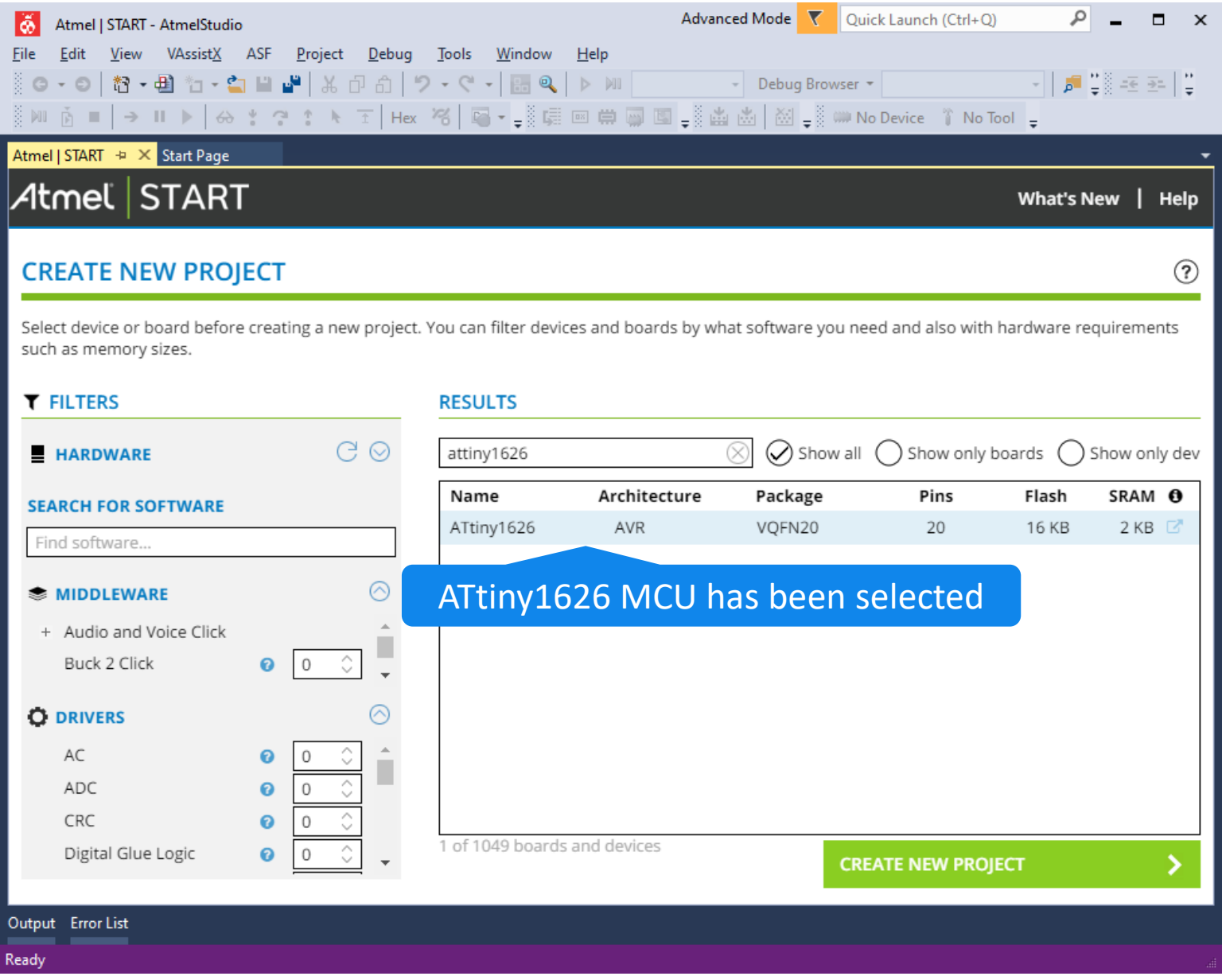


Use Device Filter to Search for MCU

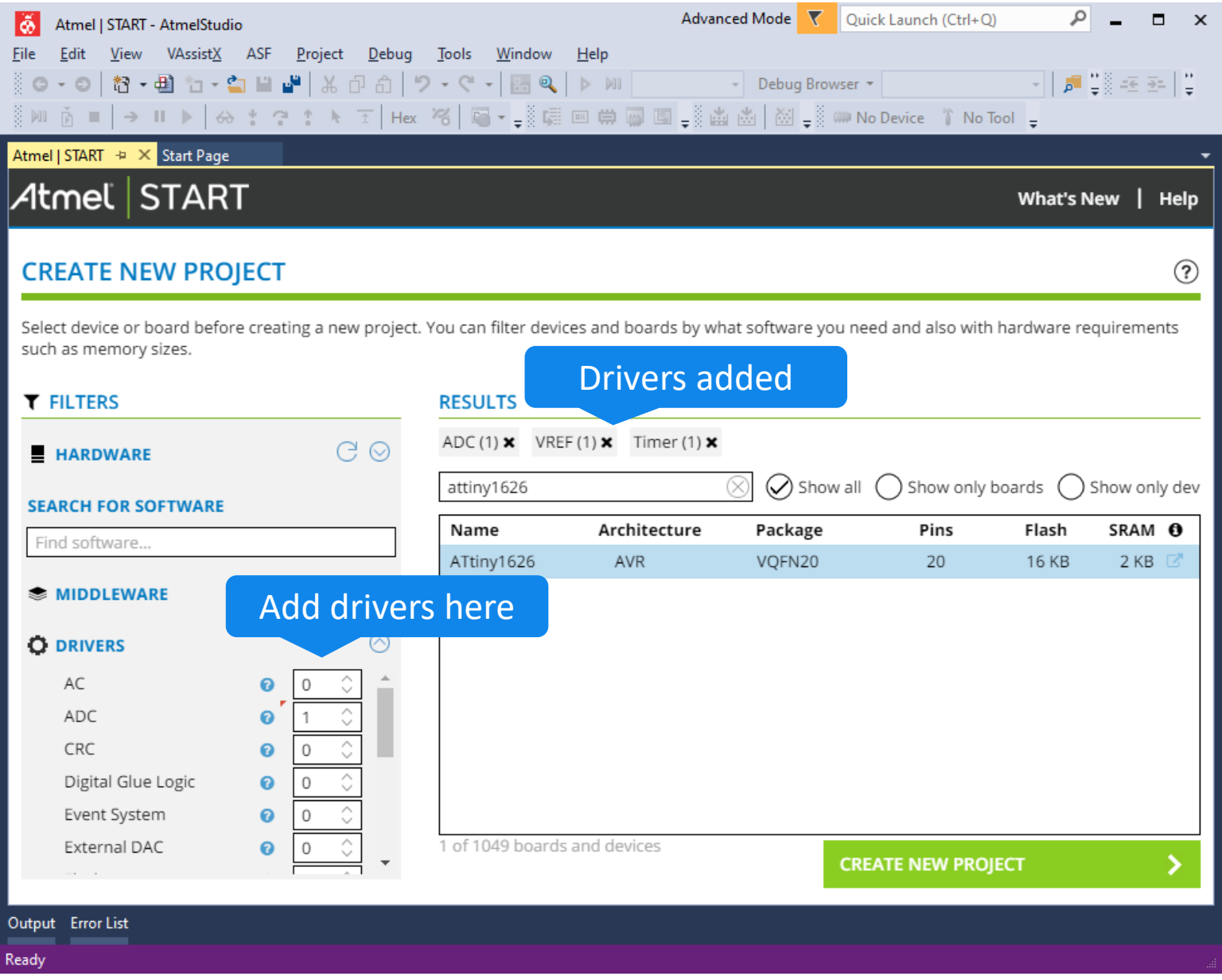


Use the device filter to search for the desired MCU.

Select the Desired MCU

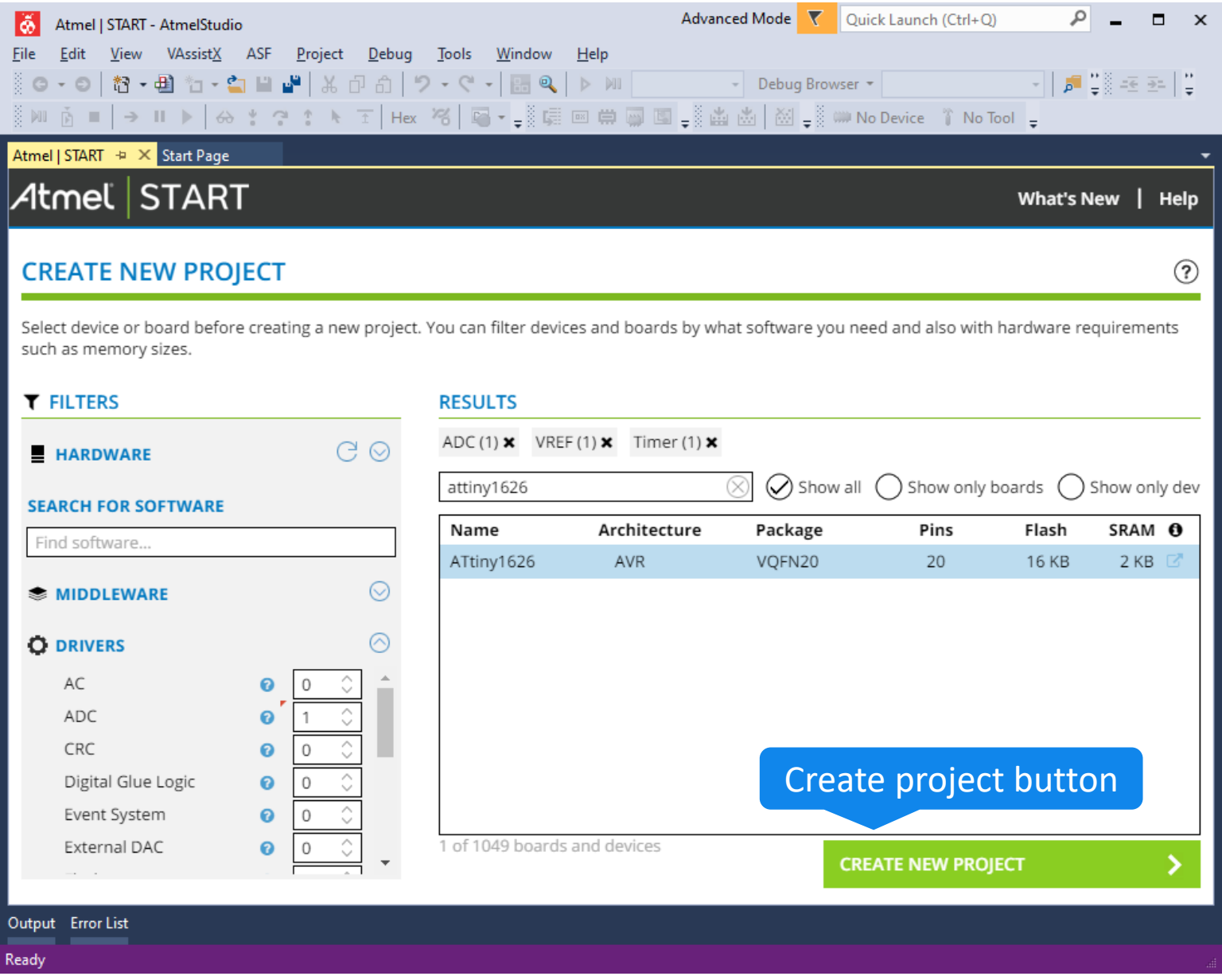


Add the Drivers (ADC, etc)



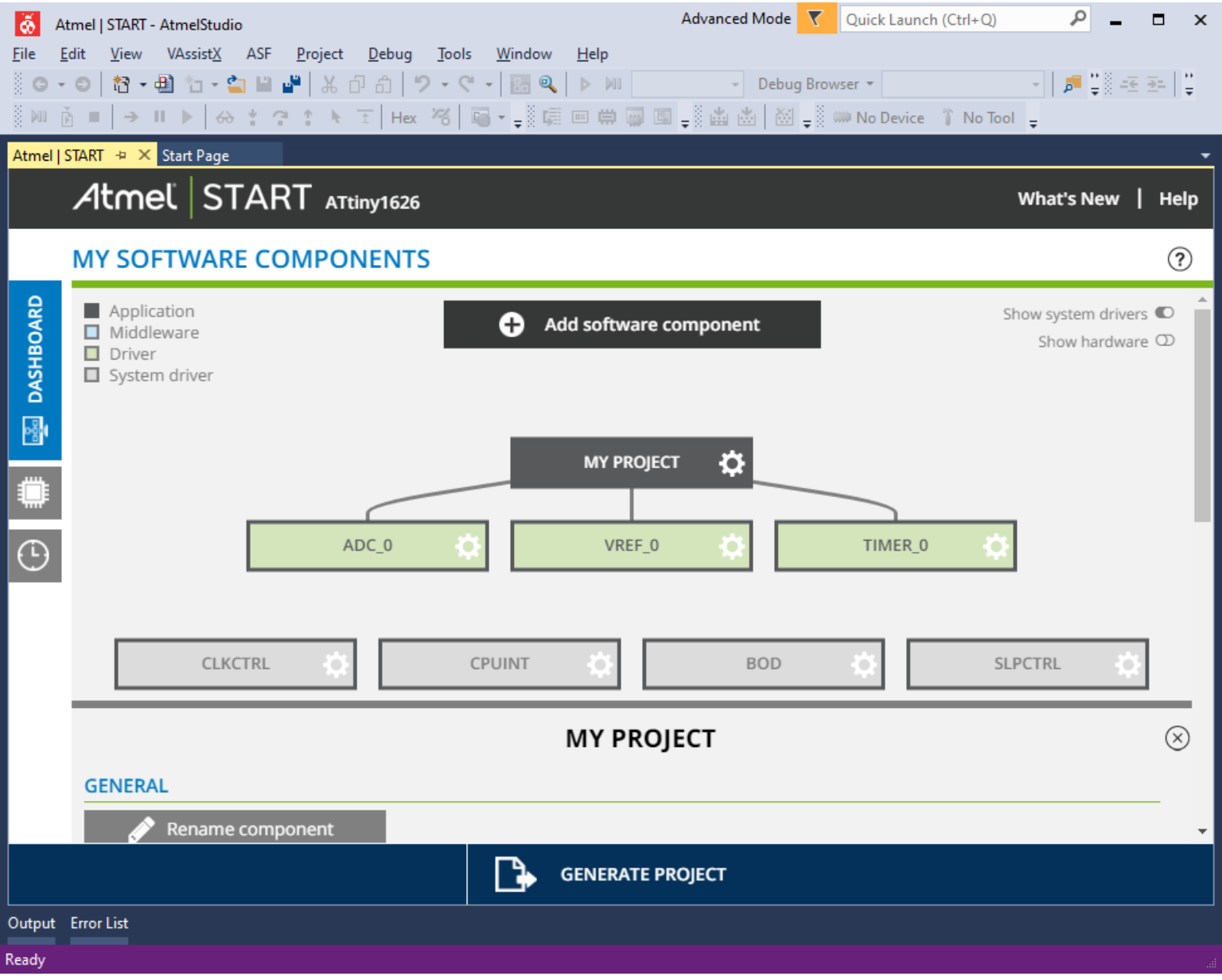
Once drivers have been added, click on **CREATE NEW PROJECT** to create the project.

Create New Project

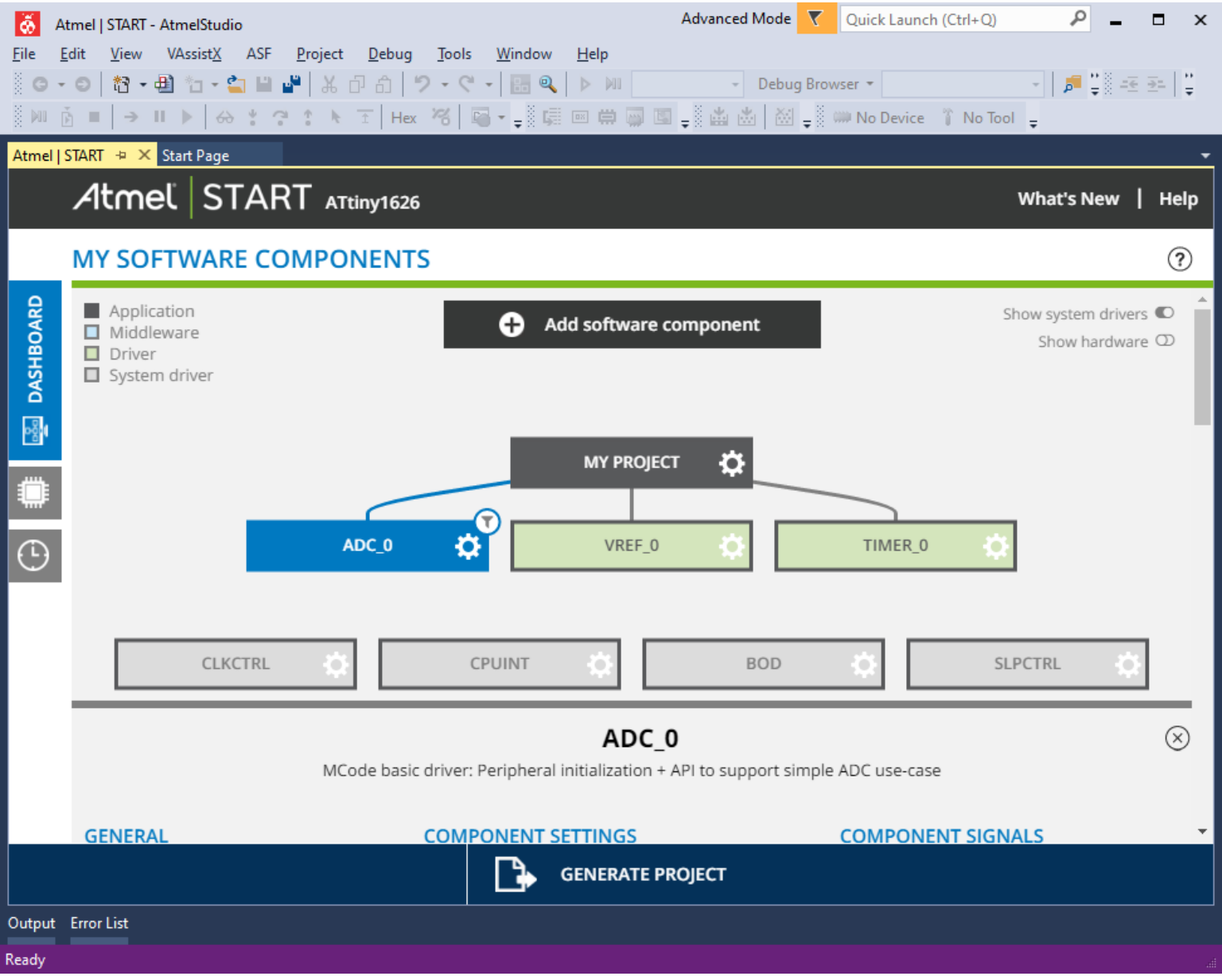


Once drivers have been added for desired MCU, click on button **CREATE NEW PROJECT** to create the project.

Check Project Software Components

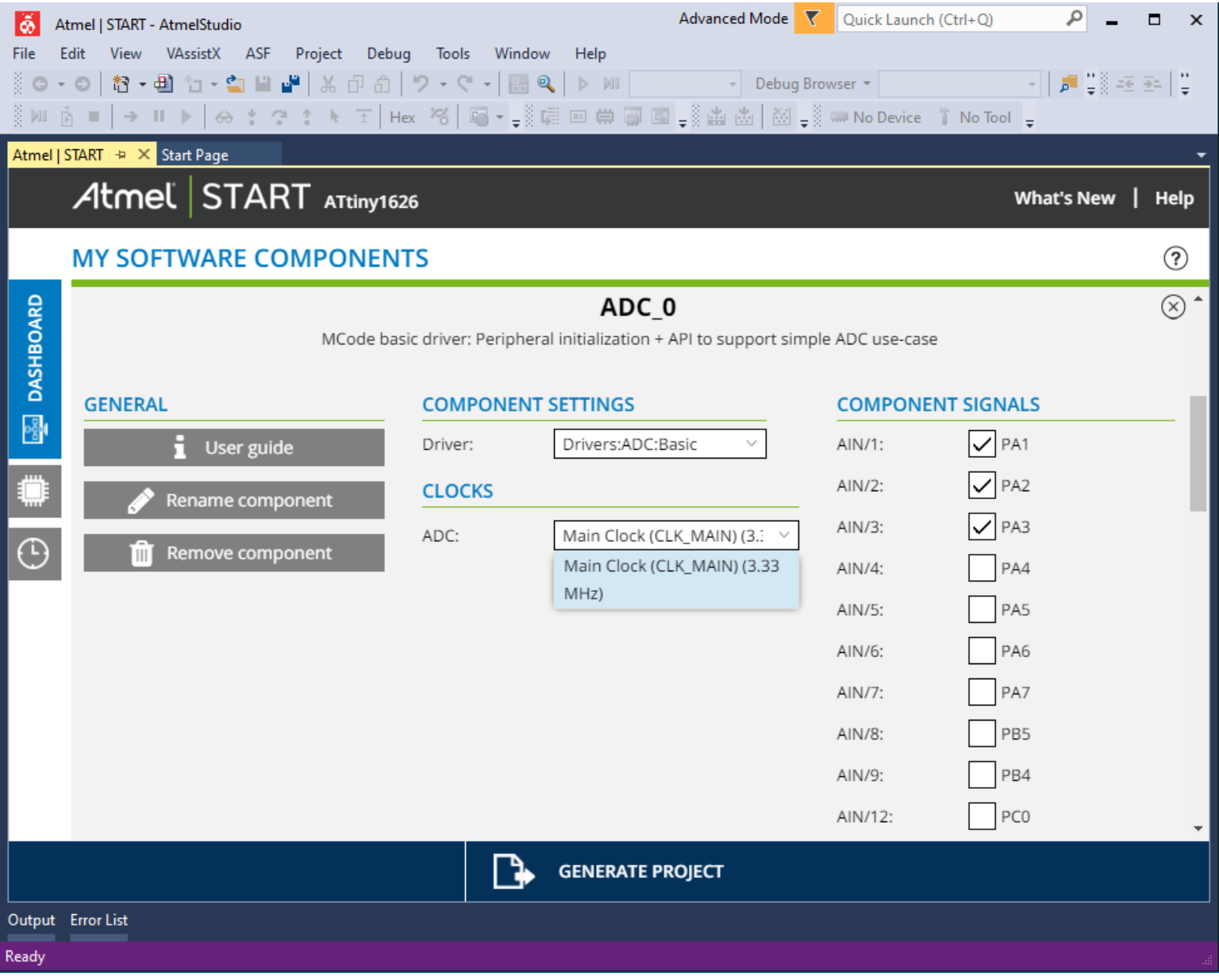


Setup the ADC Peripheral



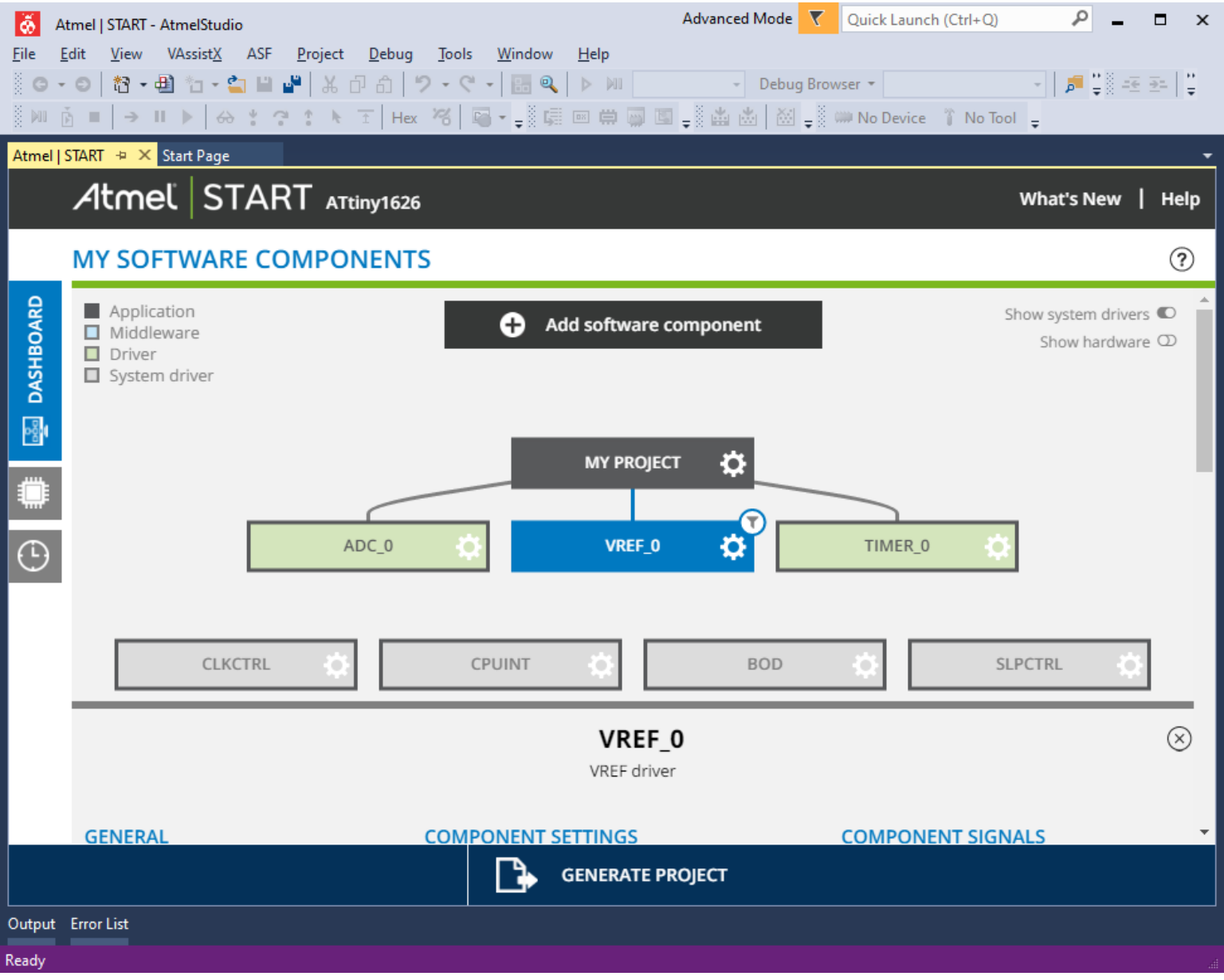
Click on **ADC_0** and scroll down to show the ADC configuration options.

Select ADC Signals



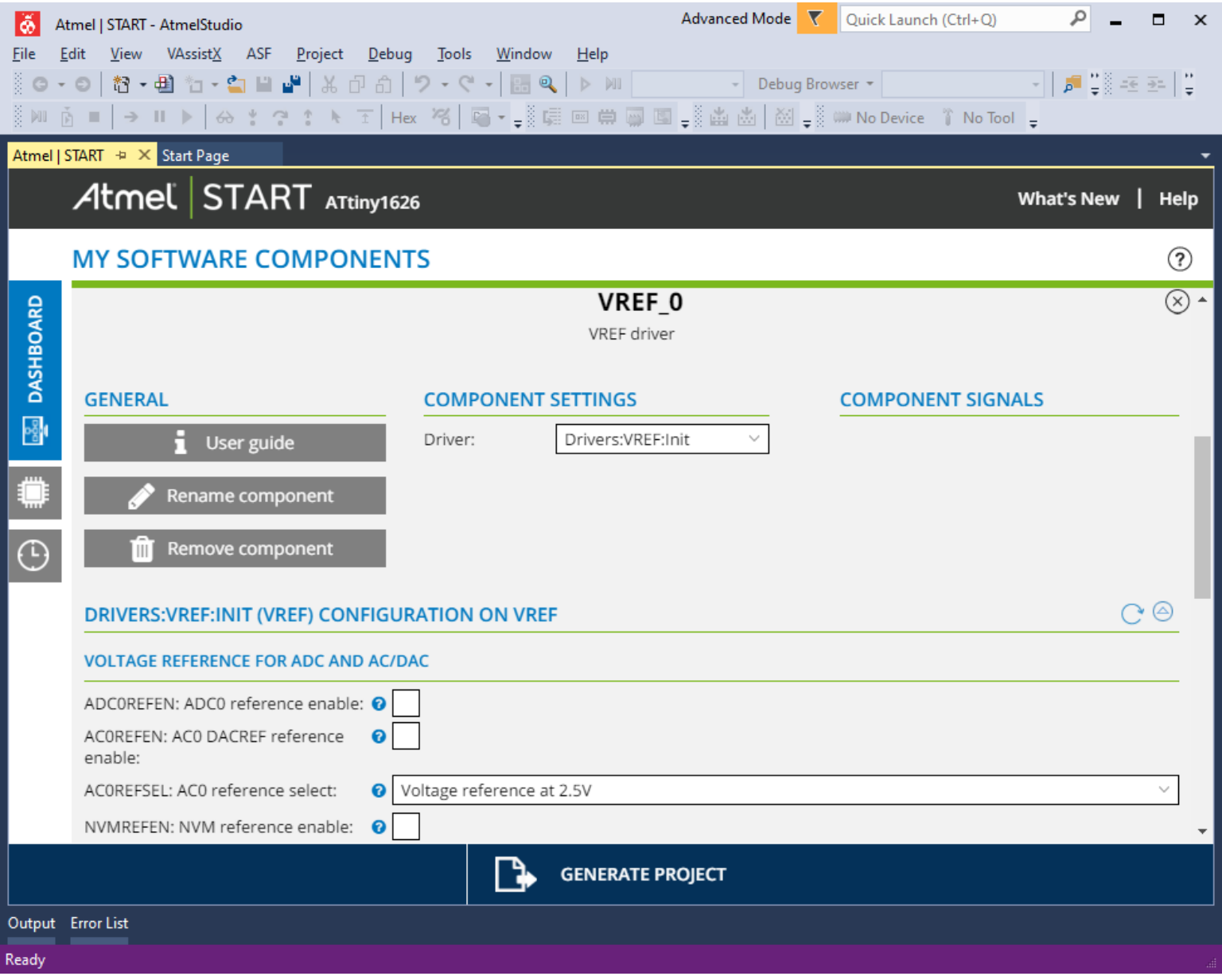
Select the ADC signals as per the schematic. Also select the driver and the clock.

Setup the VREF Peripheral

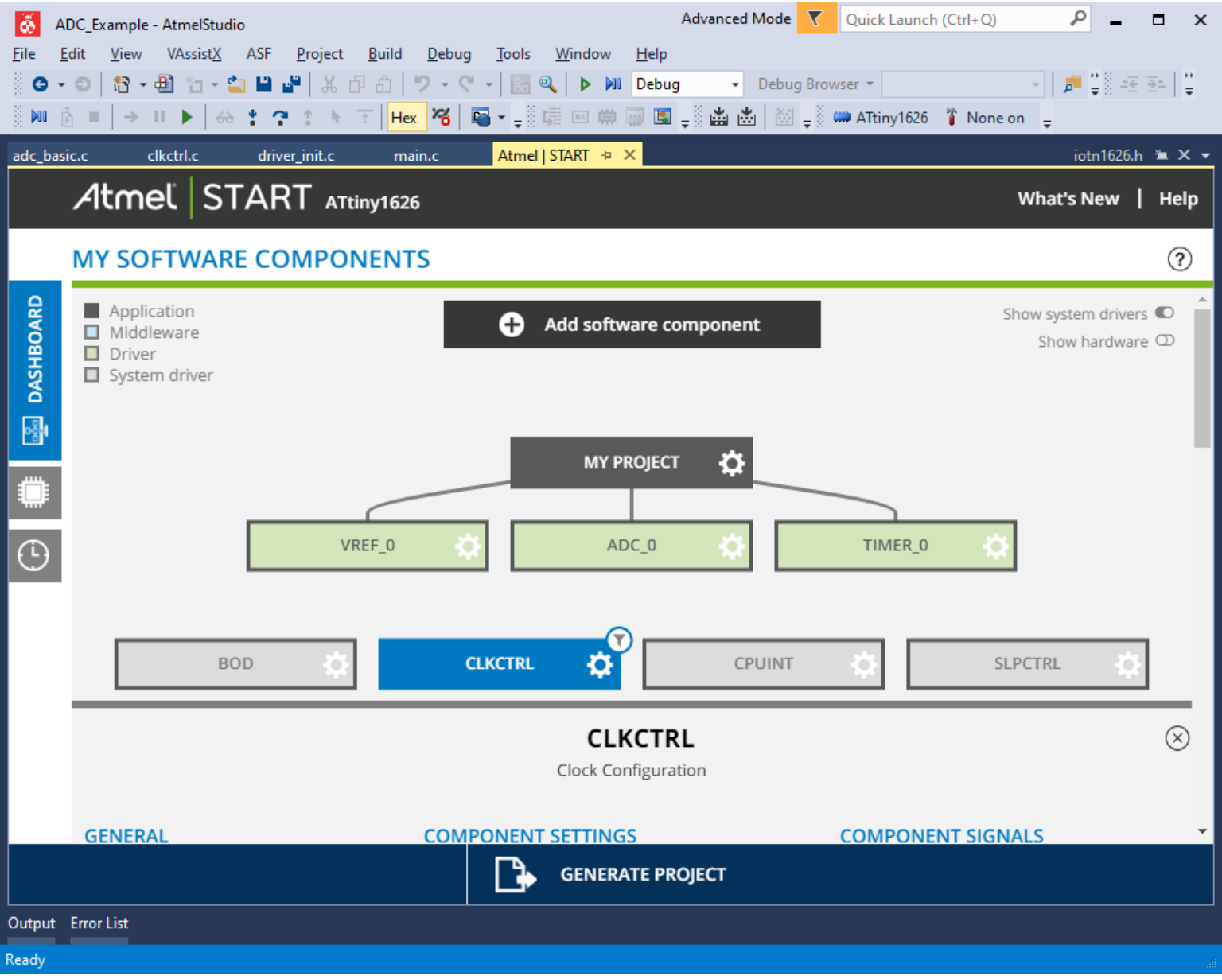


Click on VREF_0 and scroll down to show the VREF driver configuration options.

Setup the VREF Peripheral (continue)

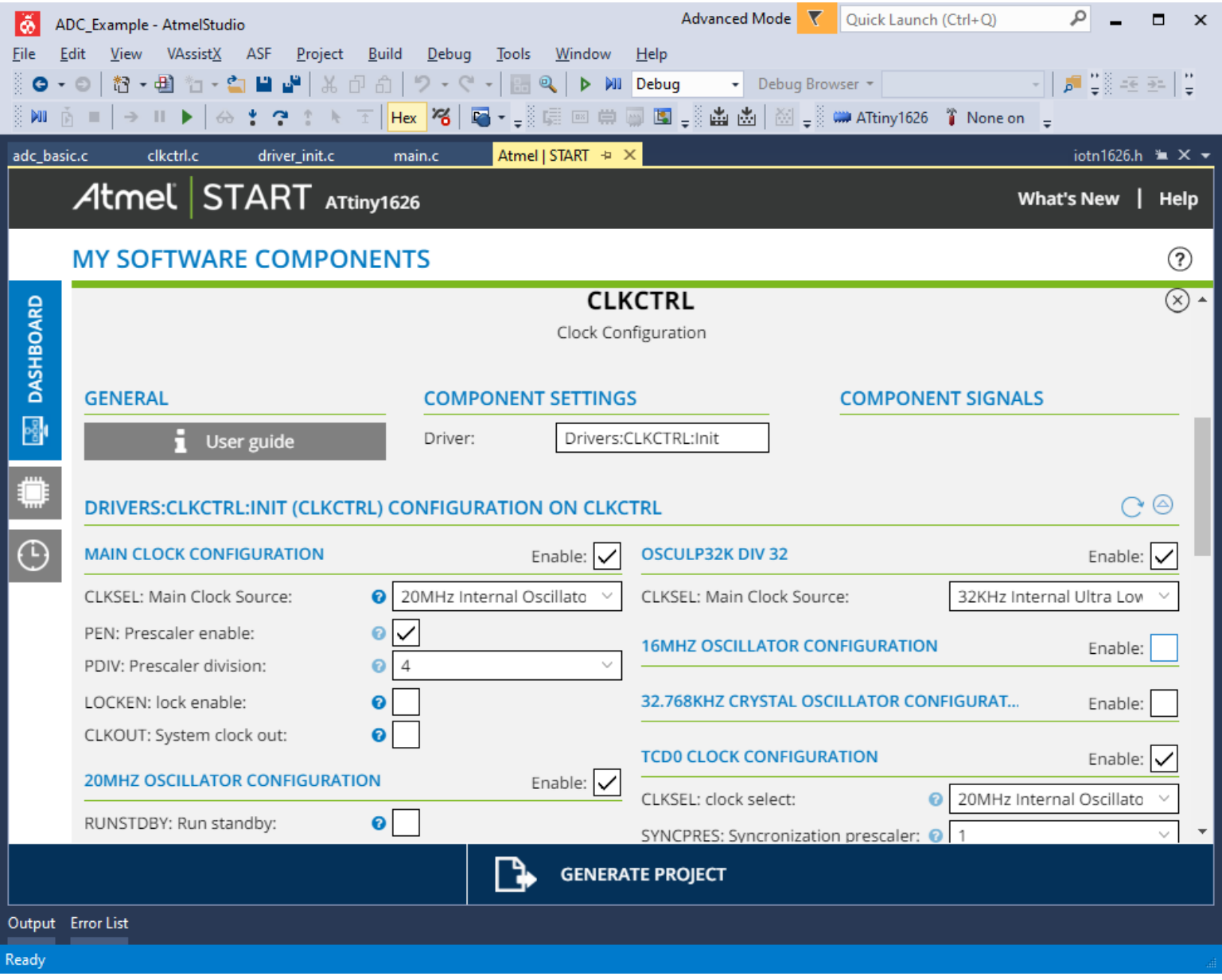


Setup the Clock Settings



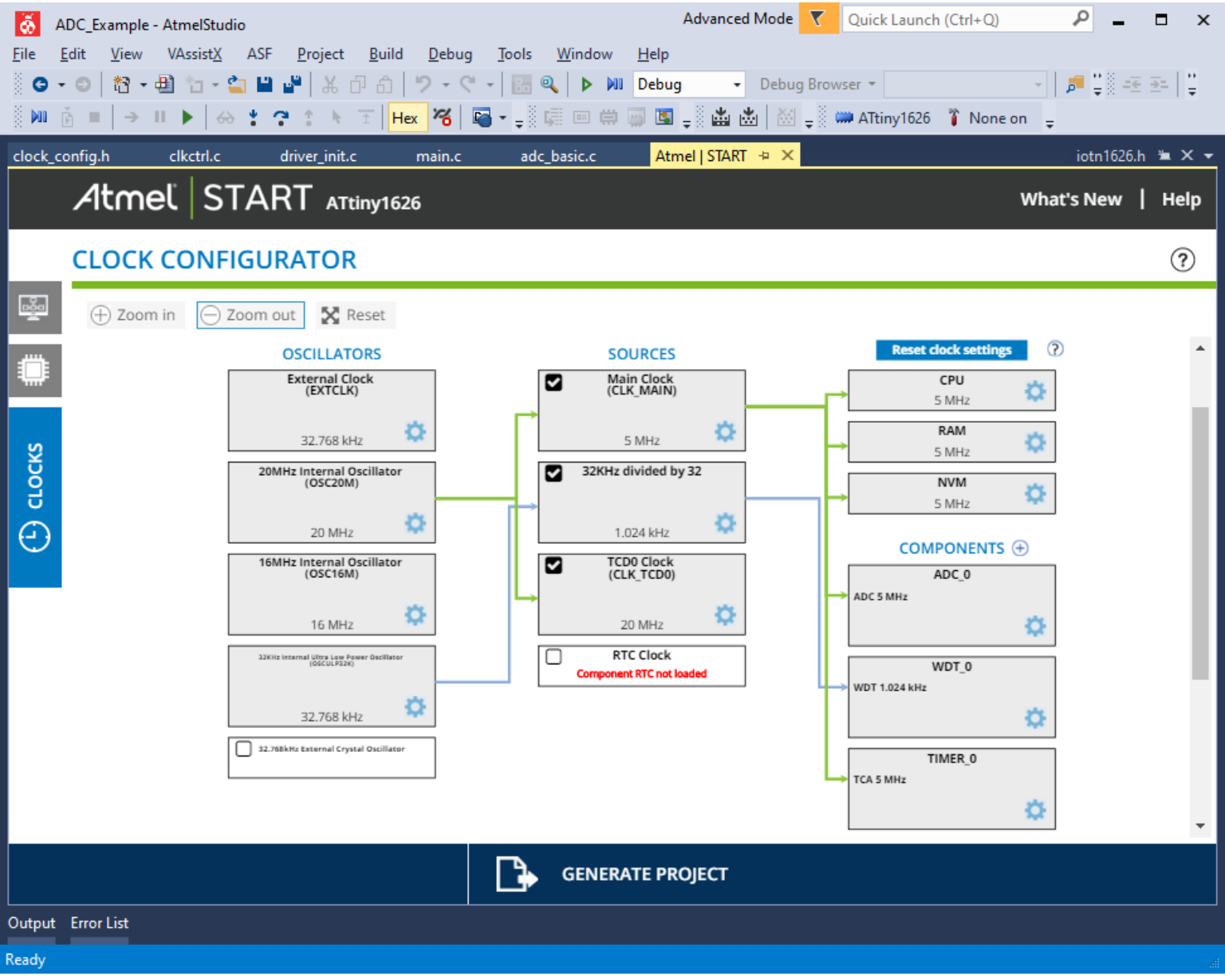
Click on CLKCTRL and scroll down to show clock configuration options.

Setup the Clock Settings (continue)



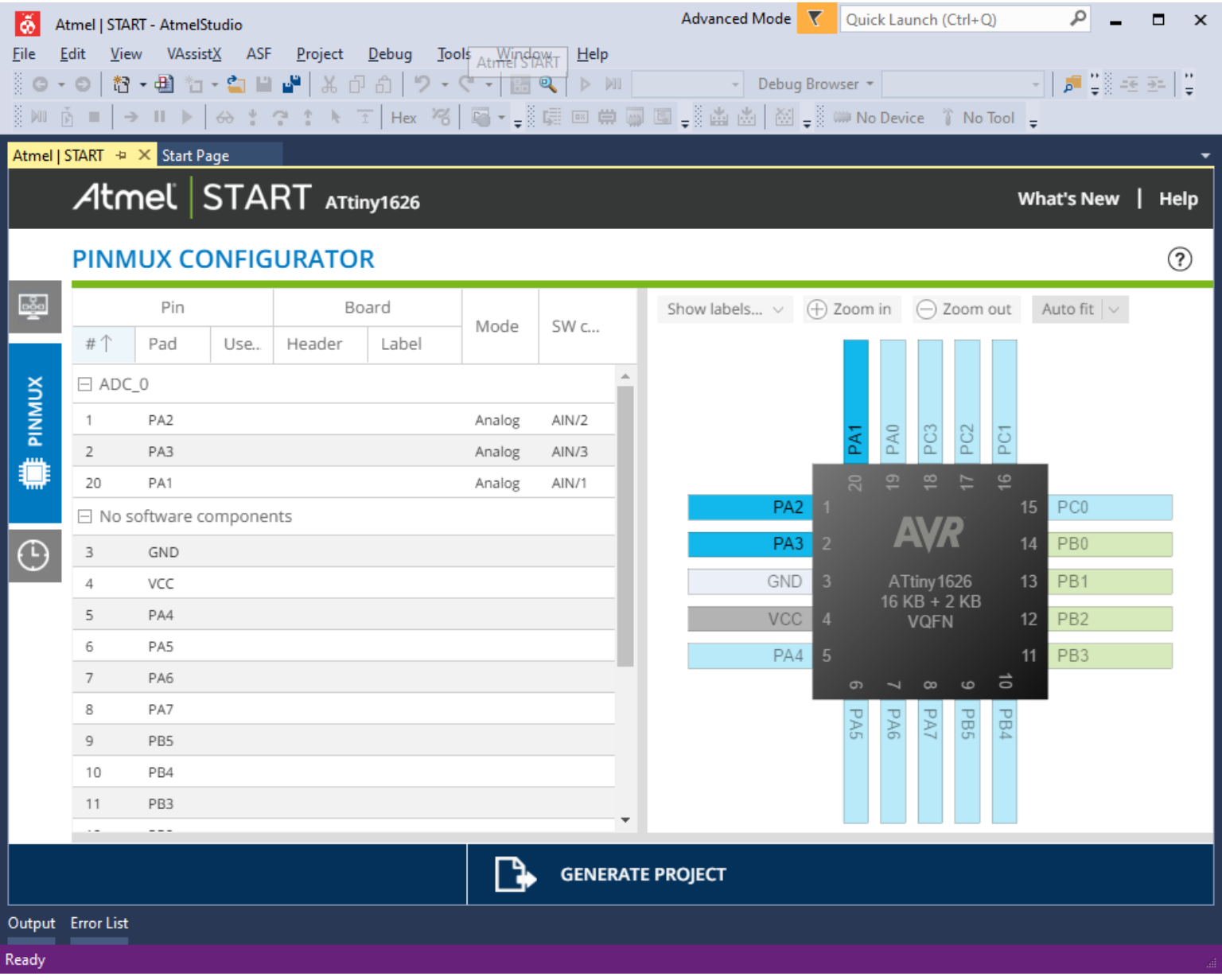
Setup the clock as required.

Setup the Clock Settings (continue)



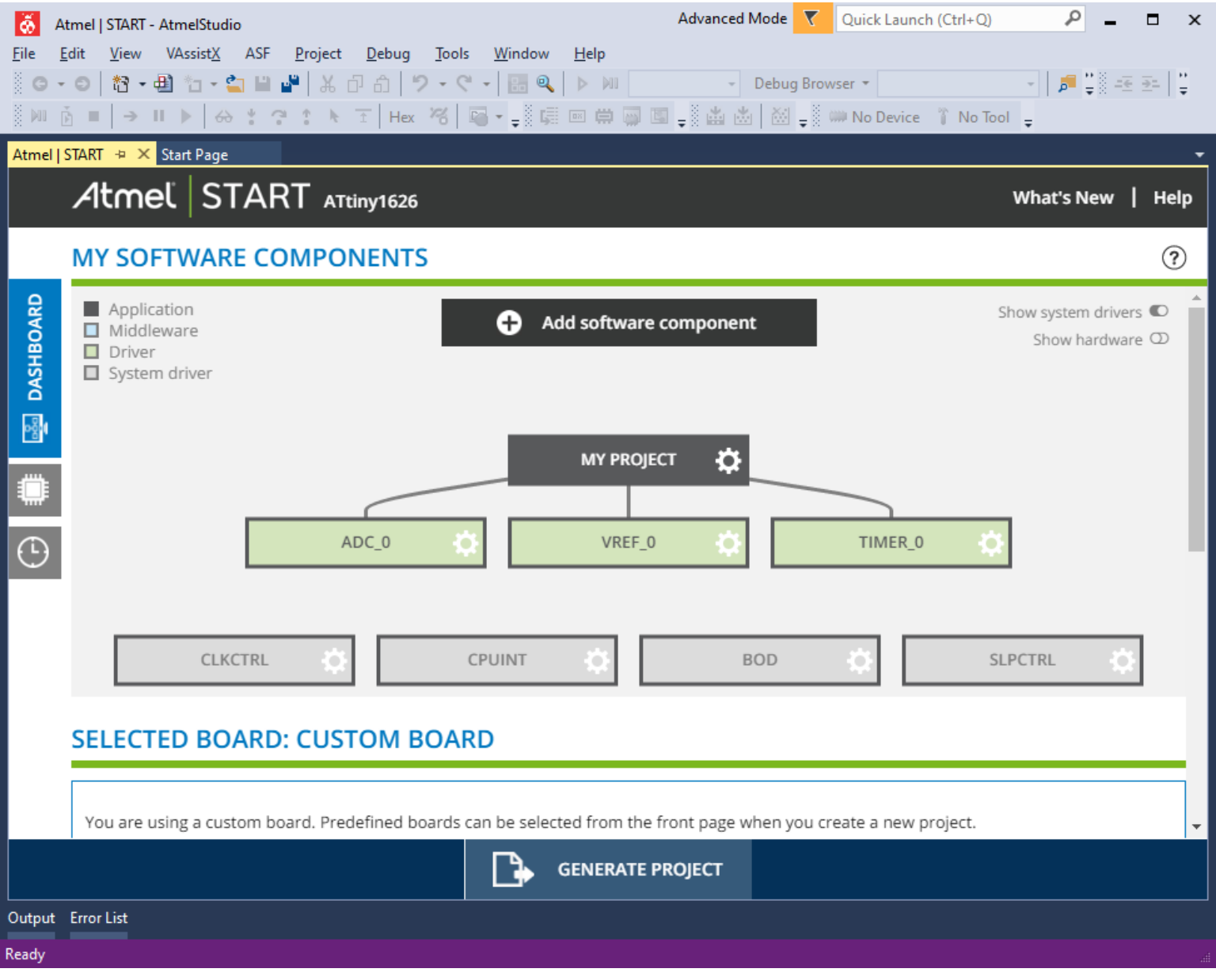
Use the Clock Configurator to easily setup the clock.

Use the Pinmux Configurator to Mux Pins



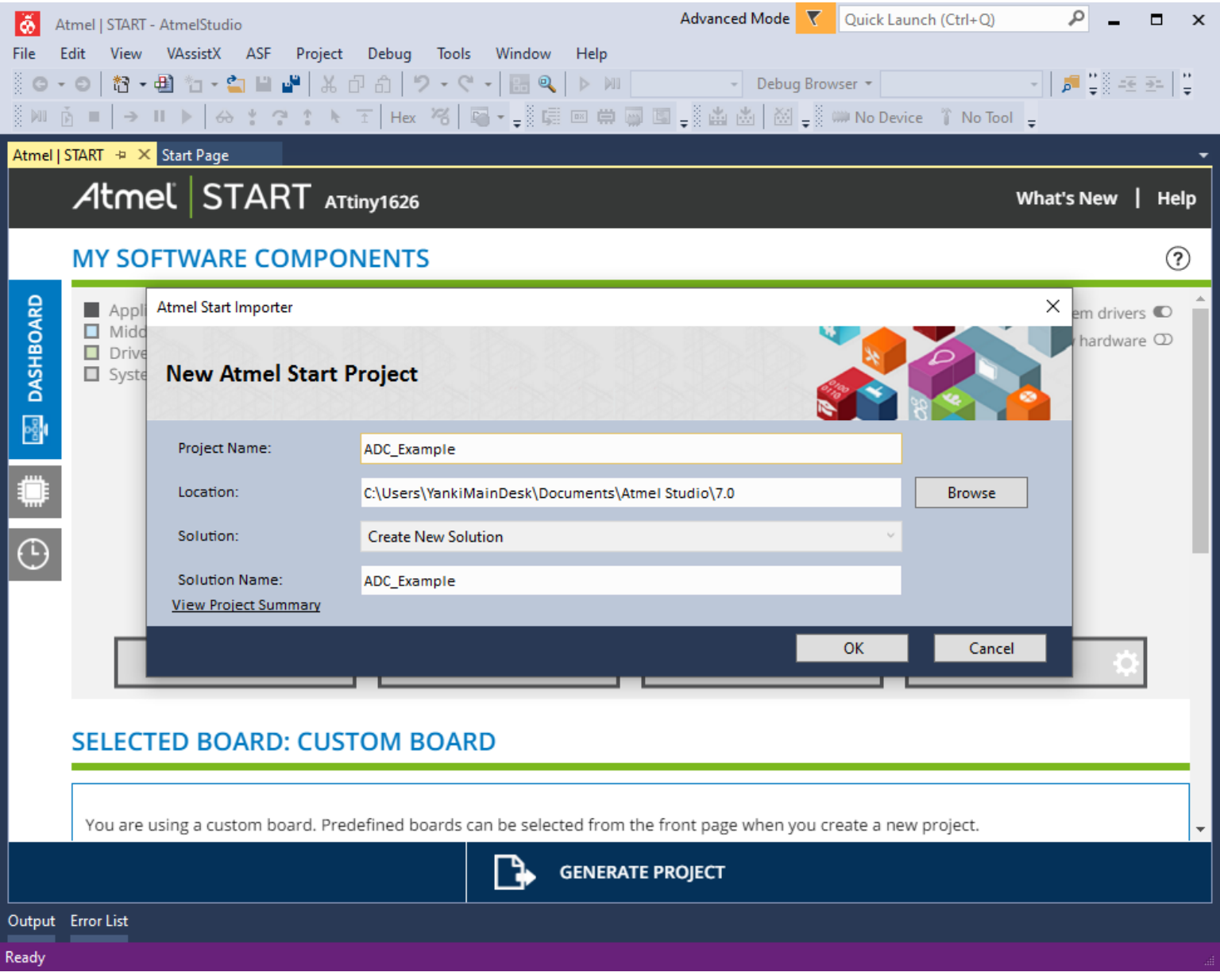
Use the Pinmux Configurator to check the pins mux and to label them.

Generate the Project Files



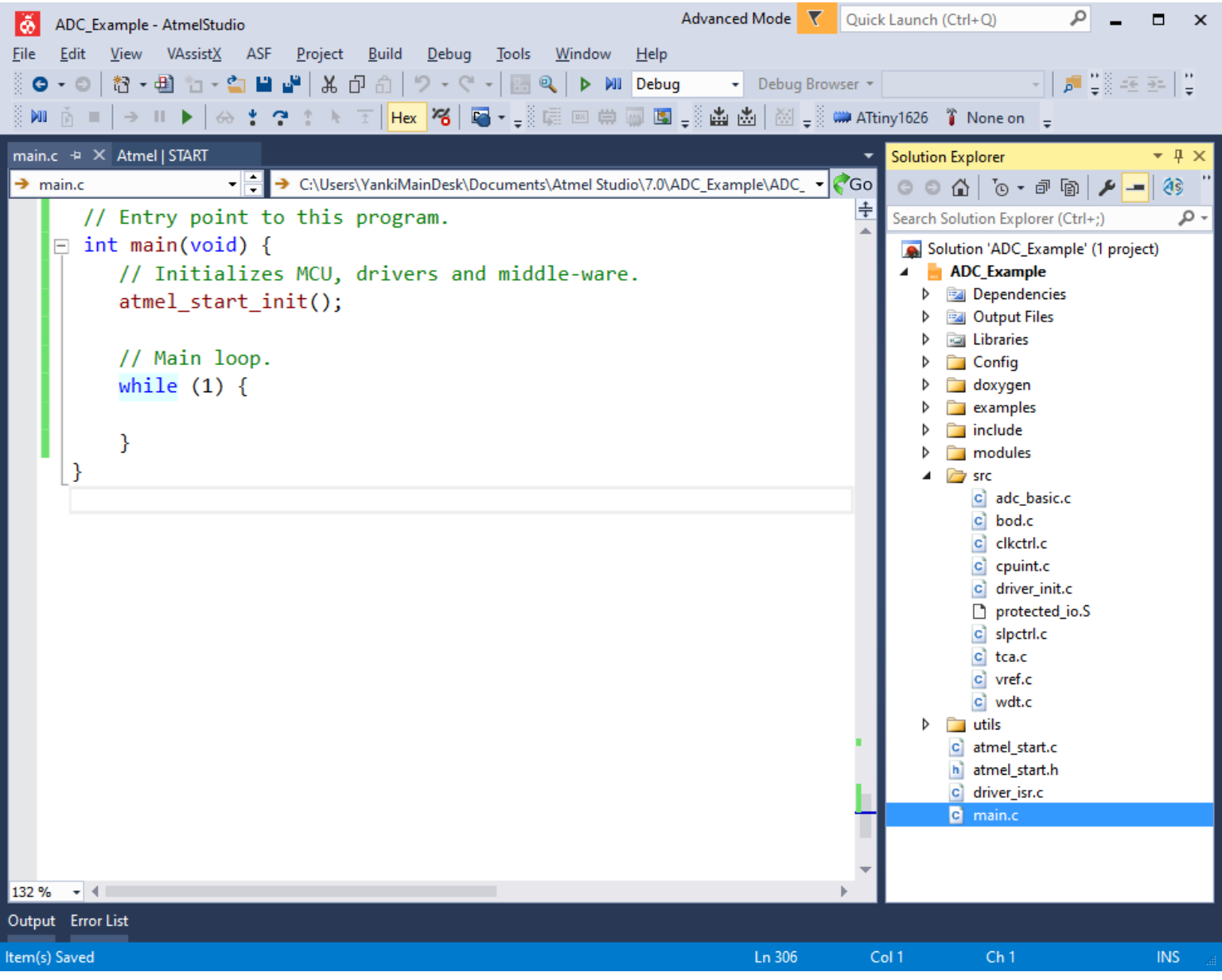
Click on GENERATE PROJECT to generate the project files.

Name the Project and Save it



Enter the name of the project and click OK to generate the project.

Use the Solution Explorer to Browse the Project



Use the Solution Explorer to browse the project files. The IDE setup the code skeleton automatically.

Entry Point to the Program Code

```c
#include <atmel_start.h>

... Missing code here ...

// Entry point to this program.
int main(void) {
    // Initializes MCU, drivers and middle-ware.
    atmel_start_init();
    // Initialize GPIOs not used by peripherals.
    GPIOs_Init();
    // Initializes the TCA0 to overflow every 1 milliseconds.
    // This timer is used as a house keeper timer.
    TCA0_Init(1);
    // Clear the Overflow Interrupt for TCA0.
    TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
    // Enable global interrupts.
    sei();

    // Initializes the Watch Dog to 1K cycles (1.0s).
    WDT_Init(WDT_PERIOD_1KCLK_gc);
    // Main loop.
    while (1) {
        // Measure voltage.
        ADC0_Measure_Voltage();
        // Reset Watch Dog.
        WDT_Reset();
    }
}
```

ADC Pins Initialization Code

```
/* Configure pins and initialize registers */
void ADC_0_initialization(void) {

    // Disable digital input buffer
    PA1_set_isc(PORT_ISC_INPUT_DISABLE_gc);
    // Disable pull-up resistor
    PA1_set_pull_mode(PORT_PULL_OFF);

    // Disable digital input buffer
    PA2_set_isc(PORT_ISC_INPUT_DISABLE_gc);
    // Disable pull-up resistor
    PA2_set_pull_mode(PORT_PULL_OFF);

    // Disable digital input buffer
    PA3_set_isc(PORT_ISC_INPUT_DISABLE_gc);
    // Disable pull-up resistor
    PA3_set_pull_mode(PORT_PULL_OFF);

    ADC_0_init();
}
```

ADC Peripheral Initialization Code

```
/* brief Initialize ADC interface */
int8_t ADC_0_init() {

    ADC0.CTRLB = ADC_PRESC_DIV2_gc; /* System clock divided by 2 */

    ADC0.CTRLF = ADC_SAMPNUM_NONE_gc /* No accumulation */
               | 0 << ADC_FREERUN_bp; /* ADC Freerun mode: disabled */

    ADC0.CTRLC = ADC_REFSEL_2500MV_gc      /* 2.5V */
               | ADC_TIMEBASE_VALUE; /* timebase value */

    ADC0.CTRLD = ADC_WINCM_NONE_gc /* No Window Comparison */
               | ADC_WINSRC_RESULT_gc; /* Result register */
    ADC0.CTRLE = 0x1b; /* Sample Duration: 0x1b */
    ADC0.DBGCTRL = 0 << ADC_DBGRUN_bp; /* Debug run: disabled */
    ADC0.COMMAND = 0 << ADC_DIFF_bp /* Diff. ADC Conv: disabled */
               | ADC_MODE_SINGLE_8BIT_gc /* Single Conversion 8-bit */
               | ADC_START_STOP_gc; /* Stop an ongoing conversion */
    ADC0.INTCTRL = 0 << ADC_RESRDY_bp /* Result Rdy Int disabled */
               | 0 << ADC_WCMP_bp; /* Window Comp Int Enable: disabled */
    ADC0.MUXPOS = ADC_VIA_ADC_gc /* Via ADC */
               | ADC_MUXPOS_AIN1_gc; /* ADC input pin 1 */
    ADC0.MUXNEG = ADC_VIA_ADC_gc /* Via ADC */
               | ADC_MUXNEG_AIN1_gc; /* ADC input pin 1 */
    ADC0.WINHT = 0x0; /* Window Comparator High Threshold: 0x0 */
    ADC0.WINLT = 0x0; /* Window Comparator Low Threshold: 0x0 */
    ADC0.CTRLA = 1 << ADC_ENABLE_bp      /* ADC Enable: enabled */
               | 0 << ADC_RUNSTDBY_bp; /* Run standby mode: disabled */
    return 0;
}
```

Clock Initialization Code

```
/* brief Initialize clkctrl interface */
int8_t CLKCTRL_init() {

    ccp_write_io((void*)&(CLKCTRL.OSC32KCTRLA),0 <<
    CLKCTRL_RUNSTDBY_bp /* Run standby: disabled */);

    ccp_write_io((void*)&(CLKCTRL.XOSC32KCTRLA),
            CLKCTRL_CSUT_1K_gc  /* 1k cycles */
     | 0 << CLKCTRL_ENABLE_bp   /* Enable: disabled */
     | 0 << CLKCTRL_RUNSTDBY_bp /* Run standby: disabled */
     | 0 << CLKCTRL_SEL_bp      /* Select: disabled */);

    ccp_write_io((void*)&(CLKCTRL.OSC20MCTRLA),0 <<
    CLKCTRL_RUNSTDBY_bp /* Run standby: disabled */);

    ccp_write_io((void *)&(CLKCTRL.MCLKCTRLB),
                CLKCTRL_PDIV_4X_gc /* 4 */
             | 1 << CLKCTRL_PEN_bp /* Prescaler enabled */);

    ccp_write_io((void*)&(CLKCTRL.MCLKCTRLA),CLKCTRL_CLKSEL_OSC20M_
    gc /* 20MHz Internal Oscillator (OSC20M) */
     | 0 << CLKCTRL_CLKOUT_bp /* System clock out: disabled */);

    ccp_write_io((void*)&(CLKCTRL.MCLKLOCK),0 << CLKCTRL_LOCKEN_bp
    /* lock enable: disabled */);

    return 0;
}
```

Function to Measure Voltage with ADC Code

```
/* function is designed to measure voltage using the ADC
   peripheral. */
void ADC0_Measure_Voltage(void) {
    // Variable declaration and initialization in this scope (local
    variables).
    adc_result_t adc_result = 0;

    // Switch to the appropriate state.
    switch(adc0_state) {
        // State to select the ADC channel to measure.
        case 0:
            if(adc0_channel == 0) {
                ADC0.MUXPOS = ADC_MEAS_CHANNEL_1;
            } else if(adc0_channel == 1) {
                ADC0.MUXPOS = ADC_MEAS_CHANNEL_2;
            } else if(adc0_channel == 2) {
                ADC0.MUXPOS = ADC_MEAS_CHANNEL_3;
            } else {
                // nothing to do here.
                // The code execution should never enter here.
                asm("nop");
            }
            adc0_state = 1;
            adc0_ms_counter = 0;
            break;
```

Function to Measure Voltage with ADC Code (cont.)

```
// Wait some time for the ADC Muxer to change channel
// before to start a new conversion.
case 1:
     if(adc0_ms_counter >= 1) {
          // Start a new conversion.
          ADC0.COMMAND |= ADC_START_IMMEDIATE_gc;
          // Change to next state.
          adc0_state = 2;
     }
     break;

// Check if conversion is done.
case 2:
     if(ADC_0_is_conversion_done()) {
          // Change state.
          adc0_state = 3;
     }
     break;
```

Function to Measure Voltage with ADC Code (cont.)

```
// State to get the conversion result.
case 3:
    // Get conversion from specified ADC channel.
    adc_result = ADC_0_get_conversion_result();

    // Formula to calculate the voltage at the ADC pin is:
    // Vadc_pin = (ADC Result * Vref) / ADC_Resolution
    // Vref = 2.5V, ADC_Resolution = 12 bits (4095)
    // Vadc_pin = (ADC Result * 2.5) / 4095

    // The voltage applied to the ADC pins is reduced by
    // a voltage divider. The voltage is reduced by a
    // factor of 11.
    // R1 + R2 / R2 = (100K + 10K) / 10K = 11
    // R3 + R4 / R4 = (100K + 10K) / 10K = 11
    // R5 + R6 / R6 = (100K + 10K) / 10K = 11
```

Function to Measure Voltage with ADC Code (cont.)

```
if(adc0_channel == 0) {
    // Calculate voltage Vin_1 (Refer to schematic).
    adc0_voltage_1 = ((adc_result * 2.5) / 4095) * 11;
} else if(adc0_channel == 1) {
    // Calculate voltage Vin_2 (Refer to schematic).
    adc0_voltage_2 = ((adc_result * 2.5) / 4095) * 11;
} else if(adc0_channel == 2) {
    // Calculate voltage Vin_2 (Refer to schematic).
    adc0_voltage_2 = ((adc_result * 2.5) / 4095) * 11;
} else {
    // nothing to do here.
    // The code execution should never enter here.
    asm("nop");
}
// Increment to use next ADC channel next time.
if(adc0_channel < 2) {
    adc0_channel++;
} else {
    adc0_channel = 0;
}
// Change state.
adc0_state = 4;
// Clear counter.
adc0_ms_counter = 0;
// Clear the Result Ready Interrupt Flag.
ADC0.INTFLAGS |= ADC_RESRDY_bm;
break;
```

Function to Measure Voltage with ADC Code (cont.)

```
        // Wait a predefined time to start next measurement.
        case 4:
            if(adc0_ms_counter > 1) {// 1 milli seconds
                // Change state.
                adc0_state = 0;
            }
            break;

        // In case of an error.
        default:
            // Change state.
            adc0_state = 0;
            // Clear the Result Ready Interrupt Flag.
            ADC0.INTFLAGS |= ADC_RESRDY_bm;
            break;
    }
}
```

Conclusion

The Atmel Studio 7 IDP is a great tool to quickly develop projects for AVR MCUs.

The ATtiny1626 is an excellent MCU to use in projects. It uses the latest technologies from Microchip with a flexible and low-power architecture, including an Event System, advanced digital peripherals, and accurate analog features such as a 12-bit differential ADC with Programmable Gain Amplifier (PGA).

References

1. Complete Code of Project
https://github.com/god233012yamil/attiny1626_adc_example

2. Atmel® Studio 7
https://microchipdeveloper.com/atstudio:studio7intro

3. ATTINY1626 with 12-bit diff ADC with PGA
https://www.microchip.com/en-us/product/ATtiny1626

4. ATTINY1627 CURIOSITY NANO EVALUATION KIT
https://www.microchip.com/en-us/development-tool/DM080104

5. Atmel-ICE is a powerful development tool for debugging and programming
https://www.microchip.com/en-us/development-tool/ATATMEL-ICE