TB3263

Getting Started with ADCC for PIC18

Introduction

Author: Ioan Pop, Microchip Technology Inc.

The Analog-to-Digital Converter with Computation (ADCC) peripheral converts an analog voltage value into a 10-bit numerical value and offers different computational actions that can be performed on the result such as averaging or low-pass filtering. The code examples in this technical brief also apply to other PIC18 families of devices that feature a 12-bit ADCC.

For each use case, there are two different implementations which have the same functionalities: one code generated with MPLAB® Code Configurator (MCC) and one bare metal code. The MCC-generated code offers hardware abstraction layers that ease the use of the code across different devices from the same family. The bare metal code is easier to follow, allowing a fast ramp-up on the use case associated code.

This technical brief describes the application area, the modes of operation and the hardware and software requirements of the Analog-to-Digital Converter with Computation. It covers the following use cases:

- **ADCC Single Conversion:**
  This example shows how to configure the ADCC to perform a single conversion.
- **ADCC Temperature Measurement:**
  This example shows how to configure the ADCC to read the temperature sensor and provide a value in Celsius and Fahrenheit.
- **ADCC Low-pass Filtering of a Conversion:**
  This example shows how to configure the ADCC to perform a conversion in the low-pass filter mode.
- **ADCC Spike Detection:**
  This example shows how to configure the ADCC to trigger an interrupt when a voltage spike is detected.

Note: The examples in this technical brief have been developed using the PIC18F47Q10 Curiosity Nano development board. The PIC18F47Q10 pin package present on the board is QFN.

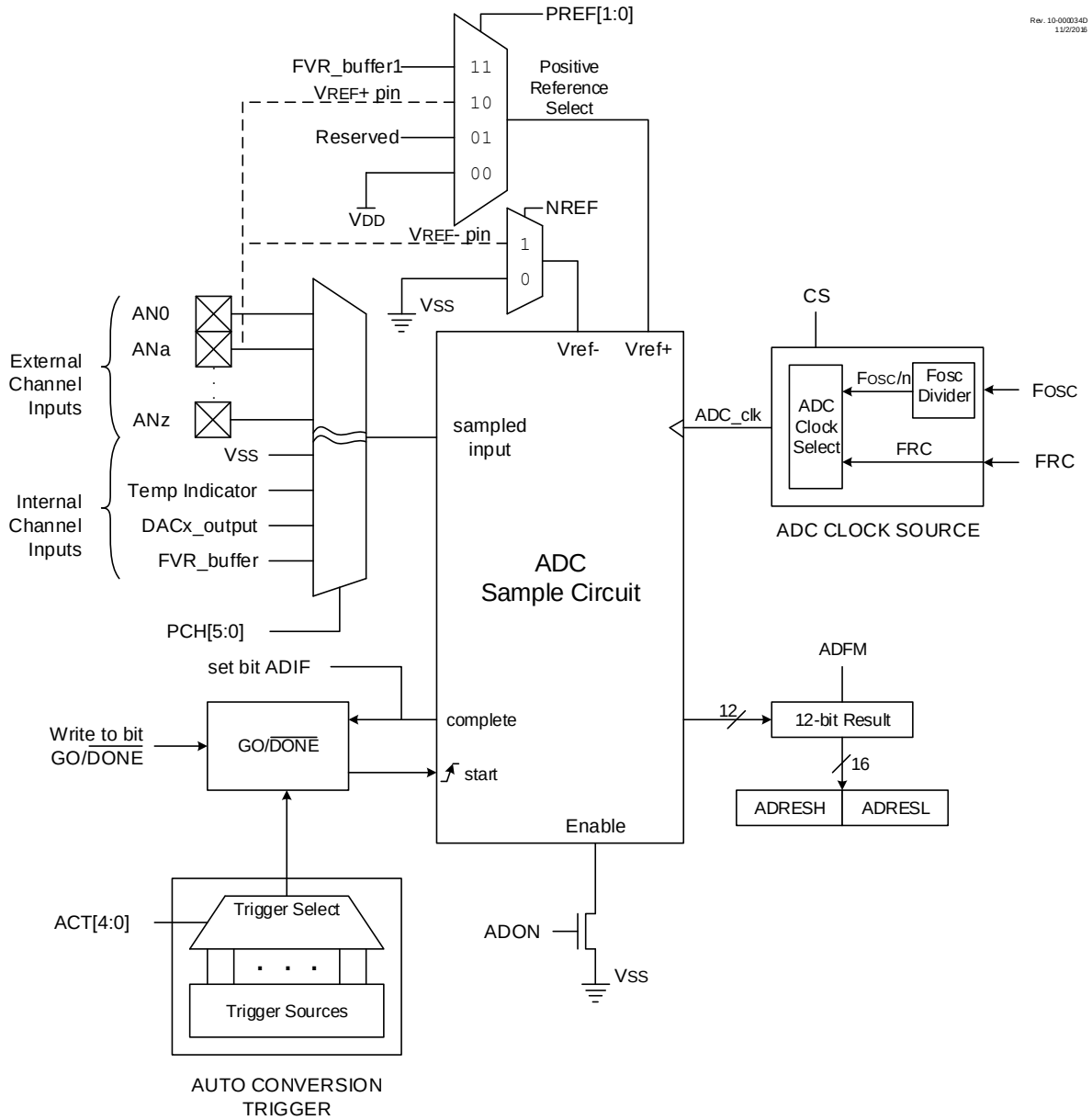
Table of Contents

1. Peripheral Overview

The Analog-to-Digital Converter with Computation (ADCC) is a peripheral that reads an analog voltage, transforms it into a digital value and performs various computations on the results. The ADCC input can be selected from several internal channels (Fixed Voltage Reference (FVR), temperature sensor or ground reference) or from an external pin. The ADC Positive Channel Selection (ADPCH) register selects the channel.

The ADCC can have different positive and negative references. The positive reference can be chosen by writing in the ADC Positive Voltage Reference Selection (ADPREF) bits of the ADC Reference Selection (ADREF) register. The available sources are the FVR output, the external V_{REF+} pin or the supply voltage (V_{DD}). The negative reference can be selected from the ground reference (V_{SS}) or external V_{REF-} pin by writing in the ADC Negative Voltage Reference Selection (ADNREF) bit in the same register as before.

Figure 1-1. ADCC Block Diagram



The ADCC has two clock sources. They are selected by the value of the Clock Selection (ADCS) bit in the Control 0 (ADCON0) register. The value '0' selects the main microcontroller clock prescaled by a value determined by the ADCLK register. The value '1' of ADCS selects the dedicated fixed-frequency ADCC clock called FRC.

The ADCC can provide right- or left-justified results depending on the setting of the results Format/Alignment Selection (ADFM) bit in the Control 0 (ADCON0) register in the following way: '1' for right-justified results and '0' for left-justified. The result is found in the 16-bit ADC Result (ADRES) register.

The ADCC can be configured to start conversions on a trigger signal. The trigger can come from an external pin or from other peripherals. The timing requirements for a conversion still need to be followed (e.g. if a conversion takes 100 us, then a trigger that comes faster than 100 us will not produce a new conversion).

The ADCC is enabled by setting the ADC Enable (ADON) bit in the Control 0 (ADCON0) register. There are then three ways to start a conversion. The first is by setting the ADC Conversion Status (ADGO) bit in the ADCON0 register. The second is by receiving an auto-conversion trigger that is configured for this purpose. The third is automatic, if the ADC Continuous Operation Enable (ADCONT) bit in the ADCON0 register is set. A conversion will start immediately after the end of the previous one.

The ADCC module is equipped with post-conversion computation features that can be configured through the ADC Control 2 (ADCON2) register. There are five available computation modes:

- Basic: The normal conversion mode. A single or double conversion is done and then the result is stored in the ADRES register.
- Accumulate: With each trigger, the result is right-shifted by a number of bits equal to the value in the ADC Accumulated Calculation Right Shift Select bits in the ADCON2 register and then added to the accumulator, a threshold test is performed and ADC Repeat Counter (ADCNT) register is incremented.
- Average: With each trigger, the result is right-shifted by a number of bits equal to the value in the ADCRS bits in the ADCON2 register and then added to the accumulator. When a number of conversions equal to the ADC Repeat Setting (ADRPT) is completed, the value in the accumulator is divided by the number of samples and a threshold test is performed and the accumulator is cleared on the next trigger.
- Burst average: It is similar to the Average mode, but it re-triggers the conversion until ADCNT becomes equal to ADRPT even if continuous mode is not enabled.
- Low-pass filter: It works the same as the Average mode except it performs a low-pass filtering operation on all the samples and reduces the effect of high-frequency noise on the average.

At the end of each computation, the results are latched and held stable. The error is then calculated by a formula selected by the ADC Error Calculation Mode Select (ADCALC) bits in the ADC Control 3 (ADCON3) register. The error is stored in the ADC Setpoint Error (ADERR) register.

The error is then compared to the upper and lower thresholds depending on the setting of the Threshold Interrupt Mode Select (ADTMD) bits in the ADC Control 3 (ADCON3) register and if it is between the required bounds an interrupt will be triggered.

2. ADCC Single Conversion

This example shows how to configure the ADCC module to give one conversion of an analog value read from a pin. The ADCC module is configured to use its dedicated clock (FRC) with results right-justified. First, the sample capacitor will be discharged by connecting it to the ground potential of the microcontroller and then a conversion will be started on the analog channel.

The value can then be read from the debugger's variable watch menu.

To achieve the functionality described by the use case, the following actions will be performed:

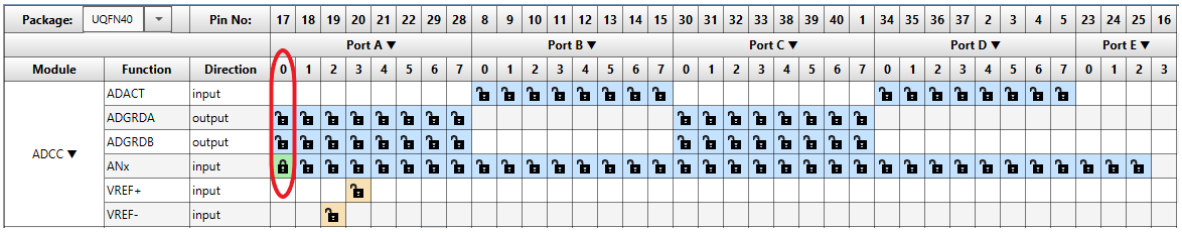
- System clock initialization
- Port initialization
- ADCC initialization
- Discharge sample capacitor
- Starting the conversion

2.1 MCC Generated Code

To generate this project using MPLAB Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar (more information about how to install the MCC plug-in can be found here.
3. Go to *Project Resources → System → System Module* and do the following configuration:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 1 MHz
   - Clock Divider: 1
   - In the Watchdog Timer Enable field in **WWDT** tab, make sure "**WDT Disabled**" is selected
   - In the **Programming** tab, make sure **Low-Voltage Programming Enable** is checked.
4. From the Device Resources window, add ADCC, then do the following configuration:
   - Enable ADC: Check
   - Operating: Basic Mode
   - Clock Source: FRC
   - Result Alignment: Right
   - Positive Reference: V_{DD}
   - Negative Reference: V_{SS}
5. Go to *Pin Manager → Grid View* and select the RA0 pin as ANx input for the ADCC.

Figure 2-1. Pin Mapping



6. Go to the Pin Module in the **Project Resources** tab and set the RA0 pin as analog and as input (check to see if output is not checked).
7. Press the **Generate** button. The generated code can now be found in the project folder.
8. Add the following code to the main function.
   Add this before the main function. This is the variable that stores the value and can be read with the debugger.

```
uint16_t volatile adcVal;
```

Add this code inside the body of the `main()` function. This discharges the sample capacitor and adds a function that starts the conversion on the analog channel and returns the value.

```
ADCC_DischargeSampleCapacitor();
adcVal = ADCC_GetSingleConversion(channel_ANA0);

while (1)
{
    ;
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

2.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section. It has five functions: three of them configure the ADCC, the main oscillator and the pin, while the fourth one discharges the sample capacitor and the last one initiates the conversion and returns the result.

The first step will be to configure the microcontroller to disable the Watchdog Timer (WDT) and to enable low-voltage programming.

```
/*disable Watchdog*/
#pragma config WDTE = OFF
/* Low voltage programming enabled, RE3 pin is MCLR */
#pragma config LVP = ON
```

The `CLK_Init` function selects HFINTOSC as the main oscillator and sets the frequency to 1 MHz.

```
static void CLK_Init(void)
{
    /* set HFINTOSC Oscillator */
    OSCCON1bits.NOSC = 6;
    /* set HFFRQ to 1 MHz */
    OSCFRQbits.HFFRQ = 0;
}
```

The `PORT_Init` function sets the RA0 pin as analog input.

```
static void PORT_Init(void)
{
    /*set pin RA0 as analog*/
    ANSELAbits.ANSELA0 = 1;
    /*set pin RA0 as input*/
    TRISAbits.TRISA0 = 1;
}
```

The `ADCC_Init` function enables the ADCC, configures it to give right-justified results and use its dedicated clock.

```
static void ADCC_Init(void)
{
    /* Enable the ADCC module */
    ADCON0bits.ADON = 1;
    /* Select FRC clock */
    ADCON0bits.ADCS = 1;
    /* result right justified */
    ADCON0bits.ADFM = 1;
}
```

The `ADCC_DischargeSampleCap` function connects the ADCC to V_{SS} and thus discharges the capacitor to provide an accurate reading.

```
static void ADCC_DischargeSampleCap(void)
{
    /*channel number that connects to VSS*/
    ADPCH = 0x3C;
}
```

The `ADCC_ReadValue` function selects the channel, starts the conversion, waits for it to end and then returns the result. This result is saved into a variable that can be watched with the debugger. The channel that connects to pin RA0 is `0x00`.

```
static uint16_t ADCC_ReadValue(uint8_t channel)
{
    ADPCH = channel;
    /*start conversion*/
    ADCON0bits.ADGO = 1;
    while (ADCON0bits.ADGO)
    {
        ;
    }

    return ((uint16_t)((ADRESH << 8) + ADRESL));
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

3. ADCC Temperature Measurement

This example shows how to read the internal temperature sensor of the microcontroller. To achieve that, the ADCC should be enabled and configured to use its dedicated clock (FRC) with results right-justified.

The microcontroller is equipped with a temperature circuit designed to measure the operating temperature of the silicon die (internal temperature). It is integrated into the Fixed Voltage Reference (FVR) module and provides a voltage proportional with the die temperature.

The temperature sense circuit is enabled by setting the Temperature Indicator Enable (TSEN) bit in the Fixed Voltage Reference Control (FVRCON) register. It provides two selectable output ranges: the high range provides more resolution over temperature range and a higher operating range (>3.6V), while the lower range is provided for low-voltage operation. The range is selected by the Temperature Indicator Range Selection (TSRNG) bit in the FVRCON register

First, the sample capacitor will be discharged and then a conversion will be started on the temperature channel.

Two macros are provided for transforming the received ADCC value into a Celsius or Fahrenheit value.

To achieve the functionality described by the use case, the following actions will be performed:

- System clock initialization
- Port initialization
- ADCC initialization
- Discharge sample capacitor
- Starting the conversion

3.1 MCC Generated Code

To generate this project using MPLAB Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar (more information about how to install the MCC plug-in can be found here.
3. Go to *Project Resources → System → System Module* and do the following configuration:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 1 MHz
   - Clock Divider: 1
   - In the Watchdog Timer Enable field in **WWDT** tab, make sure “**WDT Disabled**” is selected
   - In the **Programming** tab, make sure **Low-Voltage Programming Enable** is checked.
4. The peripherals can be added to the project from the Device Resources window. Add ADCC and FVR. The peripherals are now available for configuration in the Project Resources window under Peripherals.
5. ADCC configuration:
   - Enable ADC: Check
   - Operating: Basic Mode
   - Clock Source: FRC
   - Result Alignment: Right
   - Positive Reference: V_{DD}
   - Negative Reference: V_{SS}
6. FVR configuration:
   - Enable FVR: Check
   - Enable Temperature Sensor: Check
   - Voltage Range Selection: Lo_Range
7. Press the **Generate** button. The generated code can now be found in the project folder.

8. Add the following code to the main.c file. Add the following code before the `main()` function. It provides two macros that will convert the value read from the ADCC to a value in Celsius or Fahrenheit.

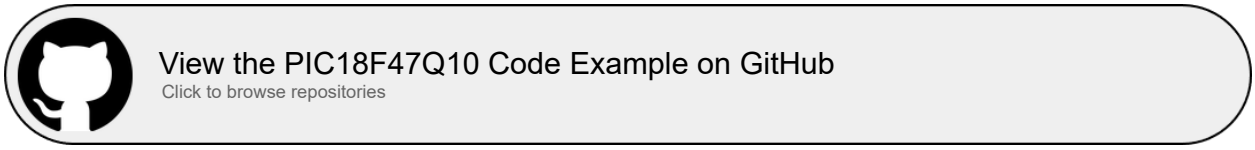
```
#define VDD                                 3.3
#define ADC_TO_CELSIUS(adcVal)              (int16_t) \
                ((1241.4967 - VDD * (1024 - (adcVal))) / 2.70336)
#define ADC_TO_FAHRENHEIT(adcVal)           (int16_t) \
                ((((1241.4967 - VDD * (1024 - (adcVal))) / 2.70336) * 1.8) + 32)

uint16_t volatile adcVal;
int16_t volatile celsiusValue;
int16_t volatile fahrenheitValue;
```

Add the following code to the `main()` function. It will discharge the sampling capacitor and start a conversion on the temperature channel and return the value in a variable.

```
ADCC_DischargeSampleCapacitor();
adcVal = ADCC_GetSingleConversion(channel_Temp);

celsiusValue = ADC_TO_CELSIUS(adcVal);
fahrenheitValue = ADC_TO_FAHRENHEIT(adcVal);
```



3.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section. It has five functions: three of them configure the ADCC, FVR and the main oscillator and one discharges the sample capacitor. The last one initiates the conversion on the temperature channel.

The first step will be to configure the microcontroller to disable the Watchdog Timer (WDT) and to enable low-voltage programming.

```
/*disable Watchdog*/
#pragma config WDTE = OFF
/* Low voltage programming enabled, RE3 pin is MCLR */
#pragma config LVP = ON
```

The `CLK_Init` function selects HFINTOSC as the main osciilator and sets the frequency to 1 MHz.

```
static void CLK_Init(void)
{
    /* set HFINTOSC Oscillator */
    OSCCON1bits.NOSC = 6;
    /* set HFFRQ to 1 MHz */
    OSCFRQbits.HFFRQ = 0;
}
```

The `FVR_Init` function enables the Fixed Voltage Reference (FVR) and the temperature sensor. The sensor is configured to work on the low range as the high range requires an operating voltage above 3.6V.

```
static void FVR_Init(void)
{
    /*Enable temperature sensor*/
    FVRCONbits.TSEN = 1;
    /*Enable FVR*/
    FVRCONbits.FVREN = 1;
}
```

The `ADCC_Init` function configures the peripheral to use its FRC clock and give the results right justified.

```
static void ADCC_Init(void)
{
    /* Enable the ADCC module */
```

```
    ADCON0bits.ADON = 1;
    /* Select FRC clock */
    ADCON0bits.ADCS = 1;
    /* result right justified */
    ADCON0bits.ADFM = 1;
}
```

The `ADCC_DischargeSampleCap` function connects the ADCC channel to V_{SS} to discharge the sampling capacitor and provide accurate results.

```
static void ADCC_DischargeSampleCap(void)
{
    /*channel number that connects to VSS*/
    ADPCH = 0x3C;
}
```

The `ADCC_ReadValue` function initiates the conversion on the temperature channel, waits for it to end and returns the value. The channel number to connect to the temperature sensor is `0x3C`.

```
static uint16_t ADCC_ReadValue(uint8_t channel)
{
    ADPCH = channel;
    /*start conversion*/
    ADCON0bits.ADGO = 1;
    while (ADCON0bits.ADGO)
    {
        ;
    }

    return ((uint16_t)((ADRESH << 8) + ADRESL));
}
```

The value from the ADC is then converted to Celsius and Fahrenheit values that give the temperature of the device. The variables can be checked with the debugger.

```
#define VDD                                 3.3
#define ADC_TO_CELSIUS(adcVal)              (int16_t) \
                ((1241.4967 - VDD * (1024 - (adcVal))) / 2.70336)
#define ADC_TO_FAHRENHEIT(adcVal)           (int16_t) \
                ((((1241.4967 - VDD * (1024 - (adcVal))) / 2.70336) * 1.8) + 32)
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

4. ADCC Low-Pass Filtering of a Conversion

This example shows how to set the ADCC in the Low-pass Filter mode, read several values and give an average result with the high-frequency noise removed by the filter.

The mode of the ADCC is set through the ADCON2 register. This example sets the mode to low-pass filter (the ADMD bit field is equal to `0x4`) and sets the time constant of the filter equal to 16 (from the ADCRS bit field; the time constant is equal to 2 to the power of this bit field).

The ADCC will do a number of conversions equal to the value set in ADRPT, 100 in this example.

The ADCC is configured with right-justified results, running on its dedicated FRC clock.

RA0 is configured as analog input and the main oscillator is HFINTOSC with 1 MHz frequency.

The code will discharge the sample capacitor, initiate the conversion on the analog channel and return the result in a variable that can be checked with the debugger.

To achieve the functionality described by the use case, the following actions will have to be performed:

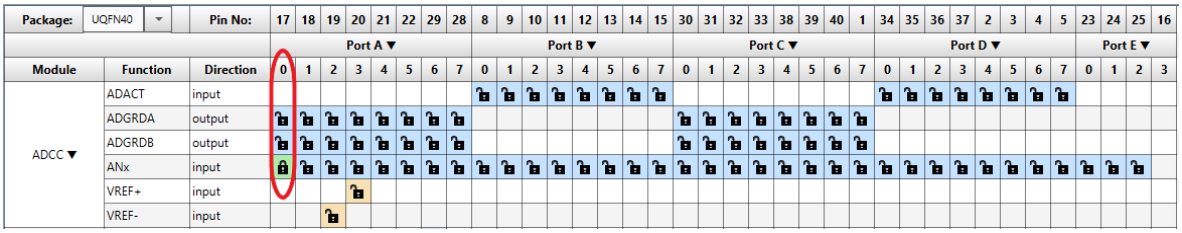
- System clock initialization
- Port initialization
- ADCC initialization
- Discharge sample capacitor
- Starting the conversion

4.1 MCC Generated Code

To generate this project using MPLAB Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar (more information about how to install the MCC plug-in can be found here.
3. Go to *Project Resources → System → System Module* and do the following configuration:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 1 MHz
   - Clock Divider: 1
   - In the Watchdog Timer Enable field in **WWDT** tab, make sure “**WDT Disabled**” is selected
   - In the **Programming** tab, make sure **Low-Voltage Programming Enable** is checked.
4. From the Device Resources window, add ADCC. Then do the following configuration:
   - Enable ADC: Check
   - Operating: Low_pass_filter_mode
   - Clock Source: FRC
   - Result Alignment: Right
   - Positive Reference: V_{DD}
   - Negative Reference: V_{SS}
   - In **Computation Features** tab:
     - Repeat: 100
     - Arc Right Shift: 4
5. Go to *Pin Manager → Grid View* and select the RA0 pin as ANx input for the ADCC:

Figure 4-1. Pin Mapping



6. Go to the Pin Module in the **Project Resources** tab and set the RA0 pin as analog and as input (check to see if Output is not checked).
7. Press the **Generate** button. The generated code can now be found in the project folder.
8. Add to the main.c file before the `main()` function the following line of code.

```
uint16_t volatile adcVal;
```

In the `main()` function add the code that discharges the sample capacitor, starts the conversion on the analog channel and returns the value in the variable that can be read with the debugger.

```
ADCC_DischargeSampleCapacitor();
adcVal = ADCC_GetSingleConversion(channel_ANA0);

while (1)
{
    ;
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

4.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section.

The first step will be to configure the microcontroller to disable the Watchdog Timer (WDT) and to enable low-voltage programming.

```
/*disable Watchdog*/
#pragma config WDTE = OFF
/* Low voltage programming enabled, RE3 pin is MCLR */
#pragma config LVP = ON
```

The `CLK_Init` function selects HFINTOSC as the main oscillator and sets its frequency to 1 MHz.

```
static void CLK_Init(void)
{
    /* set HFINTOSC Oscillator */
    OSCCON1bits.NOSC = 6;
    /* set HFFRQ to 1 MHz */
    OSCFRQbits.HFFRQ = 0;
}
```

The `PORT_Init` function sets the RA0 pin as analog input.

```
static void PORT_Init(void)
{
    /*set pin RA0 as analog*/
    ANSELAbits.ANSELA0 = 1;
    /*set pin RA0 as input*/
    TRISAbits.TRISA0 = 1;
}
```

The `ADCC_Init` function configures the mode and time constant in ADCON2 and the number of repetitions in ADRPT as well as selecting the FRC dedicated clock and setting the results right-justified.

```
static void ADCC_Init(void)
{
    /* Enable the ADCC module */
    ADCON0bits.ADON = 1;
    /* Select FRC clock */
    ADCON0bits.ADCS = 1;
    /* result right justified */
    ADCON0bits.ADFM = 1;
     /* Low pass filter mode */
    ADCON2bits.ADMD = 4;
    /* lpf time constant = 16 */
    ADCON2bits.ADCRS = 4;

    ADRPT = 100;
}
```

The `ADCC_DischargeSampleCap` function connects the ADCC channel to V_{SS} in order to discharge the sampling capacitor.

```
static void ADCC_DischargeSampleCap(void)
{
    /*channel number that connects to VSS*/
    ADPCH = 0x3C;
}
```

The `ADCC_ReadValue` function initiates a conversion on the analog channel and returns the result. The channel number that connects to pin RA0 is `0x00`. The variable can then be checked with the debugger.

```
static uint16_t ADCC_ReadValue(uint8_t channel)
{
    ADPCH = channel;
    /*start conversion*/
    ADCON0bits.ADGO = 1;
    while (ADCON0bits.ADGO)
    {
        ;
    }

    return ((uint16_t)((ADRESH << 8) + ADRESL));
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

5. ADCC Spike Detection

This example presents a solution that will generate an interrupt when a spike is detected in the value that is being read. A sharp increase or decrease is necessary to generate the interrupt, a slow increase or decrease will not trigger it. The code was tested with a potentiometer that can be turned quickly between its maximum and minimum values.

The main clock is configured to run from HFINTOSC with 1 MHz with a prescaler of 4 for a clock frequency of 250 kHz. The RA0 pin is configured as analog input.

The ADCC is enabled, Continuous mode is enabled, and the result is right-justified. The clock is given by the main clock of the microcontroller, divided by two times the value found in the ADC Clock Selection (ADCLK) register plus one, 128 in this case. This gives a sampling frequency of 169 Hz for the ADCC which is small enough that the interrupt can be triggered by turning the potentiometer quickly.

The ADCC is configured to run in the Average mode with 16 samples per conversion and a right shift of 16, selected by setting the ADC Accumulated Calculation Right Shift Select (ADCRS) bits in the ADCON2 register to 4. In order to ensure a correct average, it is recommended to have the same number of samples as the bits shifted to the right. The A/D Accumulator Clear Command (ADACLR) bit in the ADCON2 register is set. This Command bit clears the Overflow Status bit, the accumulator and the count register after each conversion and is necessary for the correct triggering of the threshold interrupt. When the accumulator overflows and the Overflow Status bit is set, a threshold interrupt is generated. This condition can be checked in the interrupt handler or the ADACLR bit can be set to prevent triggers from overflows.

The error calculation is configured as the first derivative of a single measurement (e.g. the error is the difference between this conversion and the last conversion) and the threshold interrupt is configured to trigger if the error is higher than the upper threshold of 35 or smaller than the lower threshold of -35.

Finally, the ADCC threshold interrupt, peripheral and global interrupts are enabled.

As in the previous examples, the sample capacitor is discharged by connecting it to Ground and then the conversion is started.

The interrupt handler returns the value of the error which is found in the ADERRH and ADERRL registers and this can be checked with the debugger.

To achieve the functionality described by the use case, the following actions will have to be performed:

- System clock initialization
- Port initialization
- ADCC initialization
- Interrupt initialization
- Discharge sample capacitor
- Starting the conversion
- ADCC threshold interrupt handling

5.1 MCC Generated Code

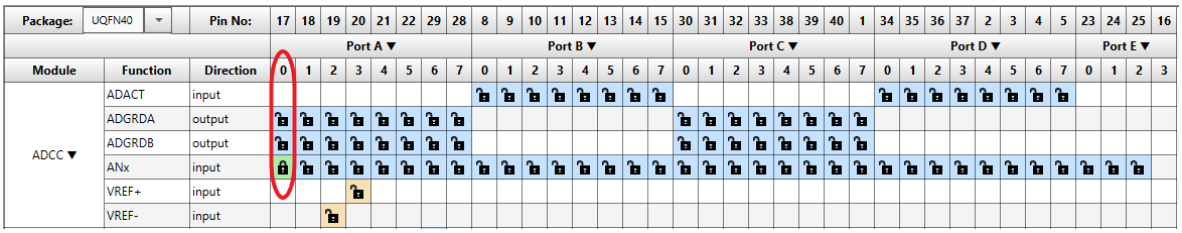
To generate this project using MPLAB Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar (more information about how to install the MCC plug-in can be found here.
3. Go to Project *Resources → System → System Module* and do the following configuration:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 1 MHz
   - Clock Divider: 4
   - In the Watchdog Timer Enable field in **WWDT** tab, make sure “**WDT Disabled**” is selected
   - In the **Programming** tab, make sure **Low-Voltage Programming Enable** is checked.
4. From the Device Resources window, add ADCC, then do the following configuration:
   - Enable ADC: Check

- Operating: Average_mode
- Clock Source: FOSC/ADCLK
- Clock: FOSC/128
- Result Alignment: Right
- Positive Reference: V_{DD}
- Negative Reference: V_{SS}
- Enable Continuous Operation: Check
- In **Computation Features** tab:
  - Error Calculation: First Derivative of Single measurement
  - Threshold Interrupt: ADERR < ADLTH or ADERR > ADUTH
  - Lower Threshold: -35
  - Upper Threshold: 35
  - Repeat: 16
  - Arc Right Shift: 4
- In CVD Features:
  - Enable ADC Threshold Interrupt: Check

5. Go to *Pin Manager → Grid View* and select the RA0 pin as ANx input for the ADCC:

Figure 5-1. Pin Mapping



6. Go to the Pin Module in the **Project Resources** tab and set the RA0 pin as analog and as input (check to see if Output is not checked).
7. Press the **Generate** button. The generated code can now be found in the project folder.
8. Add the following code to the main.c file. The code sets the Interrupt Service Routine (ISR) for the ADCC threshold interrupt to the `ThresholdISR()` function that returns the value of the error when the interrupt is triggered and clears the Interrupt flag. The rest of the code enables the global and peripheral interrupts as well as discharging the sampling capacitor and starting the conversion on the analog channel.

```
uint16_t volatile errVal;

void main(void)
{

    SYSTEM_Initialize();

    INTERRUPT_GlobalInterruptEnable();

    INTERRUPT_PeripheralInterruptEnable();
    ADCC_SetADTIInterruptHandler(ThresholdISR);
    ADCC_DischargeSampleCapacitor();
    ADCC_StartConversion(channel_ANA0);


    while (1)
    {
        ;
    }
}

void ThresholdISR(void)
{
    errVal = ADCC_GetErrorCalculation();
    PIR1bits.ADTIF = 0;

}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

5.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section. The code contains seven functions and the main function that implements the previously described functionality.

The first step will be to configure the microcontroller to disable the Watchdog Timer (WDT) and to enable low-voltage programming.

```
/*disable Watchdog*/
#pragma config WDTE = OFF
/* Low voltage programming enabled, RE3 pin is MCLR */
#pragma config LVP = ON
```

The `CLK_Init` function selects HFINTOSC as the main oscillator, sets its frequency to 1 MHz and sets the prescaler to 4 for a operating frequency of 250 kHz.

```
static void CLK_Init(void)
{
    /* set HFINTOSC Oscillator */
    OSCCON1bits.NOSC = 6;
    /* clk divided by 4 */
    OSCCON1bits.NDIV = 4;
    /* set HFFRQ to 1 MHz */
    OSCFRQbits.HFFRQ = 0;
}
```

The `PORT_Init` function configures the RA0 pin as analog input.

```
static void PORT_Init(void)
{
    /*set pin RA0 as analog*/
    ANSELAbits.ANSELA0 = 1;
    /*set pin RA0 as input*/
    TRISAbits.TRISA0 = 1;
}
```

The function `ADCC_Init` configures the ADCC: it enables it, activates Continuous Conversion mode, sets the result right-justified, selects the main clock divided by 128 as the ADCC clock.

```
static void ADCC_Init(void)
{
    /* Enable the ADCC module */
    ADCON0bits.ADON = 1;
    /* Enable continuous operation*/
    ADCON0bits.ADCONT = 1;
    /* result right justified */
    ADCON0bits.ADFM = 1;
    /*FOSC divided by 128*/
    ADCLKbits.ADCS = 63;
```

It sets the number of repetitions to 16, sets the ADACLR bit, selects the Average mode and sets the number of bits that will be right-shifted at the end of a conversion to 16.

```
    ADRPT = 16;

    /*clear status bits on overflow enabled (this setting prevents overflow
    interrupts that  trigger the same interrupt as the threshold interrupt)*/
    ADCON2bits.ADACLR = 1;
    /* Average mode */
    ADCON2bits.ADMD = 2;
```

```
    /*result is right shifted by 16*/
    ADCON2bits.ADCRS = 4;
```

It then selects the Error Calculation mode as the first derivative of a single measurement, the Threshold Comparison mode as the error being higher than the upper threshold or smaller than the lower threshold and sets the upper threshold to 35 and the lower threshold to -35.

```
    /* mode: error bigger than upper threshold
    or lower than lower threshold*/
    ADCON3bits.ADTMD = 4;
    /*error calculation method:
    difference between the last result and the current result*/
    ADCON3bits.ADCALC = 0;
    /*upper threshold*/
    ADUTH = 35;
    /*lower threshold*/
    ADLTH = -35;
```

Finally, it clears the Threshold Interrupt bit and enables the Threshold interrupt.

```
    /* Clear the ADC Threshold interrupt flag */
    PIR1bits.ADTIF = 0;
    /* Enable ADCC threshold interrupt*/
    PIE1bits.ADTIE = 1;
}
```

The `ADCC_DischargeSampleCap` function connects the ADCC channel to V_{SS} in order to discharge the sampling capacitor.

```
static void ADCC_DischargeSampleCap(void)
{
    /*channel number that connects to VSS*/
    ADPCH = 0x3C;
}
```

The `INTERRUPT_Init` function enables the global and peripheral interrupts.

```
static void INTERRUPT_Init(void)
{
    /* Enable global interrupts */
    INTCONbits.GIE = 1;
    /* Enable peripheral interrupts */
    INTCONbits.PEIE = 1;
}
```

The `ADCC_StartConversion` function selects the RA0 analog channel for the ADCC and starts the conversion by setting the ADGO bit in the ADCON0 register. The channel number to connect to pin RA0 is `0x00`.

```
static void ADCC_StartConversion(uint8_t channel)
{
    ADPCH = channel;
    /* Start the conversion */
    ADCON0bits.ADGO = 1;
}
```

The `INTERRUPT_InterruptManager` function is called at every interrupt. It checks to see if the peripheral interrupts are enabled and then checks to see if the ADCC threshold interrupt is enabled and triggered. It then calls `ADCC_ThresholdISR` to store the value of the error into the `errVal` variable.

```
void __interrupt() INTERRUPT_InterruptManager(void)
{
    if (INTCONbits.PEIE)
    {
        if ((PIE1bits.ADTIE) && (PIR1bits.ADTIF))

        {
            ADCC_ThresholdISR();
```

```
        }
    }
}
static void ADCC_ThresholdISR(void)
{
    /*read the error*/
    errVal =  ((ADERRH << 8) + ADERRL);
    /*clear interrupt flag*/
    PIR1bits.ADTIF = 0;
}
```

The `main` function calls the previously described functions in order and then waits for the interrupt. The variable can be checked with a debugger after an interrupt has been triggered,

```
void main(void)
{
    CLK_Init();
    PORT_Init();
    ADCC_Init();
    ADCC_DischargeSampleCap();
    INTERRUPT_Init();

    /*channel number that connects to RA0*/
    ADCC_StartConversion(0x00);
    while (1)
    {
        ;
    }
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

6. References

1. MPLAB Code Configurator User's Guide
2. PIC1000: Getting Started with Writing C-Code for PIC16 and PIC18 Technical Brief
3. TB3194: PIC16/PIC18 ADC² Technical Brief

7. Revision History

Document Revision	Date	Comments
B	10/2020	Updated PIC1000 link from References section.
A	05/2020	Initial document release.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-6881-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820