# Single Slope 10-Bit ADC Datasheet ADC10 V 1.30

Copyright © 2005-2013 Cypress Semiconductor Corporation. All Rights Reserved.

| Resources | PSoC® Blocks | | | API Memory (Bytes) | | Pins (per External I/O) |
|---|---|---|---|---|---|---|
| | Digital | Analog CT | Analog SC | Flash | RAM | |
| CY8C21x23, CY8C21x34, CY8CLED02, CY8CTST110, CY8CTMG110, CY8C21x45, CY8C22x45 | 1 | 1 | 1 | 336 | 4 | 1 |
| CY7C603xx | 1 | 1 | 1 | 336 | 4 | 1 |
| CYWUSB6953 | 1 | 1 | 1 | 336 | 4 | 1 |
| Die Temperature measurement | - | - | - | 380 | 3 | - |

See AN2239, ADC Selection Guide for other converters.

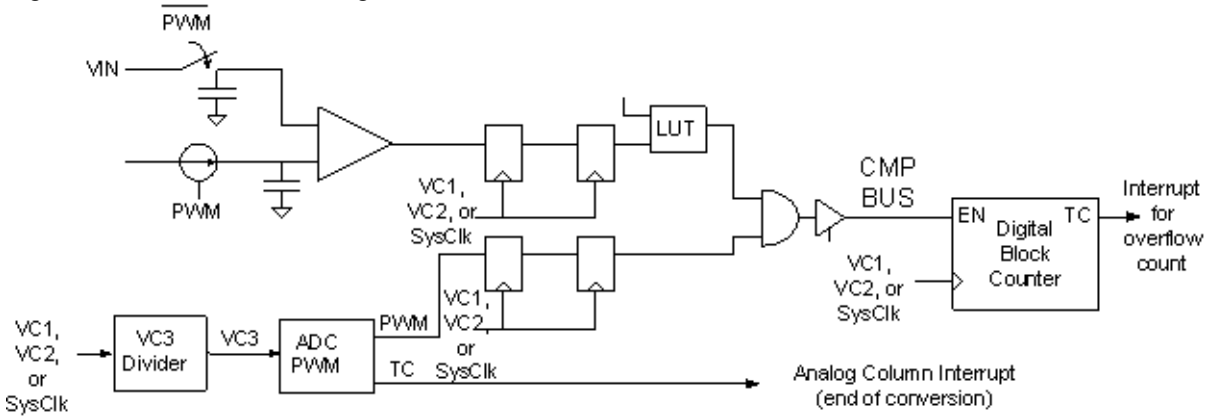
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

## Features and Overview

- Nominal 10-bit resolution
- Selectable resolution from 2-bit to 12-bit
- Input range 0 to Vdd-1
- Allows a coarse temperature measurement: range of -40°C to +125°; accuracy of ± 40°C with resolution ± 2°C

The ADC10 User Module implements a Single Slope A/D Converter that generates up to a 12-bit, full scale output (0 to 4095 count range). Although capable of generating a 12-bit output, it has only 10 effective bits of resolution. Further resolution is achieved by averaging multiple samples.

Figure 1. ADC10 Block Diagram



## Functional Description

The ADC10 User Module implements a Single Slope A/D Converter that generates up to a 12-bit, full scale output (0 to 4095 count range). The sampling rate is determined by your configuration of the clock sources and the selected parameters.

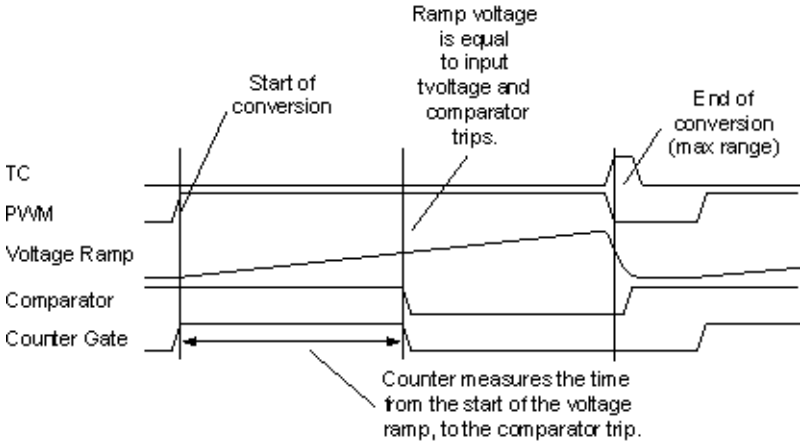
The ADC10 is constructed with these PSoC resources:

- An Analog CT Block, configured as a comparator
- An Analog SC Block, configured as a ramp generator
- A Digital Block, configured as an eight-bit counter
- A dedicated PWM used for ADC timing

A block diagram of the single slope ADC implementation is shown in Figure 1. The core of the conversion algorithm involves a current source, an integrating capacitor, and a comparator. When the current source is activated, a linear voltage ramp is generated on the capacitor. The capacitor voltage is one input to an analog comparator circuit; the other is the analog input voltage to be converted. The comparator is high until the ramp voltage exceeds the input voltage, then it transitions low. The counter keeps track of the time between the start of the ramp and the time when the comparator is tripped.

The basic conversion waveform is shown in Figure 2. In the case of 10 bits, the high time of the PWM is set so that the counter reaches 1024. The low time of the PWM is designed to allow the capacitor to discharge and process the answer. The Terminal Count of the PWM is used as an interrupt to read the results of the conversion. Bit 7 of ADC_CR is set high if the ramp voltage never exceeds the input value. This PWM generator is clocked from VC3 only and has a limited selection of high and low times.

Figure 2. ADC10 Conversion Timing Diagram




The ADC input is a sample hold of the comparator input. The sample and hold circuitry is controlled by the PWM. This guarantees that the signal remains constant while the conversion takes place.

### Calibration

Calibrate the ADC before use. Each column has an ADC capacitor trim register for this purpose. This register controls the capacitor value that determines the slope of the ADC voltage ramp. The CAPVAL [7:0] bits of the ADC_TR register are used for calibration. The goal of this calibration process is to tune the ramp time (slope) so that a full scale ADC input value results in a full scale ADC code. This is accomplished by matching the ramp time to the desired full scale conversion period.

The user module API includes a routine called ADC10_iCal, which executes a calibration algorithm to trim the ADC_TR register based upon a calibration voltage and calibration count value. The ADC10_iCal routine accepts two arguments: the expected calibration count value and the source of the reference voltage.

The return value of the ADC10_iCal routine is the result of running the ADC on the calibration signal. Ideally, it must be the expected calibration count value. However, if you do not get the expected value, then use software methods to guarantee accuracy.

### Die Temperature Measurement

The ADC provides the measurement of the on-chip die temperature circuit output signal. The signal is available as PMux analog input of the continuous time block ACE01. As a result, the feature is available only when the user module occupies the ACE01 analog block. This signal is roughly 1.0V nominally and ranges approximately between 0.7V and 1.3V. The output voltage is a function of the die temperature.

## DC and AC Electrical Characteristics

The following values indicate performance and are based upon initial characterization data. Unless otherwise specified in the following table, $T_A$ = 25°C, $V_{DD}$ = 5.0V, Full Range = 4V, calibrated to VBG (1.3V), 10 bit, sample rate of 2.7 ksps, and a data clock of 4.8 MHz.

Table 1. 5.0V ADC10 DC and AC Electrical Characteristics

| Parameter | Typical | Limit | Units | Conditions and Notes |
|---|---|---|---|---|
| Input Voltage Range | --- | 0 to $V_{DD}$-1 | V | |
| Resolution | 10 | 12 | Bits | |
| Monotonicity | Guaranteed | | | |
| Integral Error | | -- | LSB | |
| Differential Error | | -- | LSB | |
| Offset Error | | -- | mV | |
| Data Clock | -- | 0.24 to SysClk | MHz | Input to digital blocks and analog column clock |
| SNR | 47.9 | -- | dB | |
| DC Accuracy | | | | |

| Parameter | Typical | Limit | Units | Conditions and Notes |
|---|---|---|---|---|
| DNL | 0.2 | -- | LSB | |
| INL | 1.2 | -- | LSB | |
| Offset Error | -16.9 | -- | mV | |
| Gain Error | | | | |
| Including Reference Gain Error | -3.4 | -- | %FSR | |

The following values indicate performance and are based upon initial characterization data. Unless otherwise specified in the following table, $T_A$ = 25°C, $V_{DD}$ = 3.3V, Full range = 2.3V, calibrated to VBG (1.3V), 10 bit, sample rate of 3.3 ksps, and a data clock of 6 MHz.

Table 2. 3.3V ADC10 DC and AC Electrical Characteristics

| Parameter | Typical | Limit | Units | Conditions and Notes |
|---|---|---|---|---|
| Input Voltage Range | --- | 0 to $V_{DD}$-1 | V | |
| Resolution | 10 | 12 | Bits | |
| Monotonicity | Guaranteed | | | |
| Integral Error | | -- | LSB | |
| Differential Error | | -- | LSB | |
| Offset Error | | -- | mV | |
| Data Clock | -- | 0.24 to SysClk | MHz | Input to digital blocks and analog column clock |
| SNR | 49.3 | -- | dB | |
| DC Accuracy | | | | |
| DNL | 0.16 | -- | LSB | |
| INL | 2.7 | -- | LSB | |
| Offset Error | 24.4 | -- | mV | |
| Gain Error | | | | |
| Including Reference Gain Error | -13.5 | -- | %FSR | |

The following values indicate performance and are based upon initial characterization data. Unless otherwise specified in the following table, $T_A$ = 25°C, $V_{DD}$ = 2.7V, Full range = 2V, calibrated to VBG (1.3V), 10 bit, sample rate of 3.3 ksps, and a data clock of 6 MHz.

Table 3. 2.7V ADC10 DC and AC Electrical Characteristics

| Parameter | Typical | Limit | Units | Conditions and Notes |
|---|---|---|---|---|
| Input Voltage Range | --- | Vss to $V_{DD}$-0.6V | | |
| Resolution | 10 | 12 | Bits | |
| Monotonicity | Guaranteed | | | |
| Integral Error | | -- | LSB | |
| Differential Error | | -- | LSB | |
| Offset Error | | -- | mV | |
| Data Clock | -- | 0.24 to SysClk | MHz | Input to digital blocks and analog column clock |
| SNR | 47.8 | -- | dB | |
| DC Accuracy | | | | |
| DNL | 0.68 | -- | LSB | |
| INL | 15.9 | -- | LSB | |
| Offset Error | 19.9 | -- | mV | |
| Gain Error | | | | |
| Including Reference Gain Error | -27.2 | -- | %FSR | |

## Placement

Place the Analog blocks in the same column. There are two columns available on the CY8C21xxxfamily of devices. Place the counter block in any of the available digital blocks.

**Note** The single slope ADCs, ADC8 and ADC10, use the hardware resource ADCPWM for timing. There is only one ADCPWM resource on the PSoC devices that support this user module, so if you place more than one of these ADCs in a design, one or both of the ADCs may exhibit timing problems.

## Parameters and Resources

**Input**

Determines the signal source for the A/D conversion. After you place the analog blocks, you can configure this parameter to one of the allowed values. Connect a signal on a pin through the use of the Analog Column Input Mux or the Analog Mux Bus (CY8C21x34 family only). You can also connect to the neighboring column or to the Analog Reference VBG.

**Note** When multiplexing between inputs of the ADC, the first sample on a new input may not be valid. To ensure that the first reading on a new channel is valid, the ADC can either be stopped and restarted between samples or the first sample on the channel may be discarded.

**Data Clock**

Selects the clock to feed into the Data block of the user module. This in turn determines the clock that is used to feed into VC3 and the Analog Column Clock. Each ADC sample consists of a measurement period (the high time of the PWM) and a calculation period (the low time of the PWM).

The possible values of the Data Clock parameter are limited to VC1, VC2, or SysClk. Set the divider value for those clocks so that they lie within the limits set by the maximum and minimum slope generated by the SC block. The clock source provided by VC1, VC2, or SysClk is between SysClk/256 and SysClk. The selected clock must never be slower than 0.24 MHz.

The Analog Column Clock for the ADC is the same as the Data Clock. This is necessary for the user module to function properly. If SysClk is selected, API software overrides the column clock selection to set it correctly.

**Current Setting**

Selects either Normal Current (150 nA) or Low Current (37.5 nA).

**PWM High**

Selects the number of VC3 clocks during which the ADC's dedicated PWM stays high. This number multiplied by the VC3 divider is the resolution of the ADC.

**PWM Low**

Selects the number of VC3 clocks during which the ADC's dedicated PWM stays low. This period represents the calculation period of the conversion. In deciding this value, remember that the calculation period requires 160 CPU cycles to complete. If you add code to the ISR it becomes important to accommodate those extra cycles.

**Die Temp Measurement**

Enables or disables the existing ADC capability to measure the VTEMP voltage and convert the voltage into a corresponding temperature value. The parameter can be set to Enable only when the user module occupies the ACE01 analog block.

### Interrupt Generation Control

There are two additional parameters that become available when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Device Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays:

- Interrupt API
- IntDispatchMode

**Interrupt API**

The parameter is always enabled and cannot be disabled.

**IntDispatchMode**

Specifies how an interrupt request is handled for interrupts shared by multiple user modules that exist in the same block but in different overlays. Selecting ActiveStatus causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting OffsetPreCalc causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

## Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow you to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include” files.

**Note**

In this, as in all user module APIs, the values of the A and X register are altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This “registers are volatile” policy was selected for efficiency reasons. The C compiler automatically takes care of this requirement. Assembly language programmers must also make certain that their code observes the policy. Although some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

### ADC10_Start

**Description:**

Performs all required initialization for this user module and turns on power for the ADC block.

**C Prototype:**

```
void  ADC10_Start(BYTE bRange)
```

**Assembly:**

```
mov   A,ADC10_FULLRANGE
lcall  ACD10_Start
```

**Parameters:**

bRange: Specifies the measurable input range for the ADC. Symbolic names provided in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name | Value | Description |
|---|---|---|
| ADC10_LOWRANGE | 1 | The input signal can be measured from Vss to Vdd - 1.3V. |
| ADC10_FULLRANGE | 3 | The input signal can be measured within specified Input Voltage Range. |

**Return Value:**

None

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

## ADC10_Stop

**Description:**

Turns off power to the ADC block.

**C Prototype:**

```
void  ADC10_Stop (void)
```

**Assembly:**

```
lcall  ACD8_Stop
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

The A and X registers can be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions.

## ADC10_iCal

**Description:**

Calibrates the ADC by trimming the Capacitor in the SC block to vary the slope of the reference ramp used in the conversion. The ADC input must be set to a user selected fixed reference voltage to which you can calibrate. This function is called after calling ADC10_Start and before calling ADC10_StartADC. Refer to the Functional Description for more information about calibration and about this function.

**C Prototype:**

```
int   ADC10_iCal(int iVal, BYTE bCalIn)
```

**Assembly:**

```
mov   A, ADC10_CAL_P0_5
push  A
```

```
mov   A, [iVal+1]
push  A
mov   A, [iVal]
push  A
lcall  ADC10_iCal
```

**Parameters:**

iVal: This number is an expected value given the reference voltage to which the input is set. The ADC10_iCal routine trims the capacitor in the SC block until the result is equal to or as close to as the iVal as possible.

bCalIn: This value specifies the source of the calibration input. This can be an internal reference, such as AGND, which provides a 1.3V signal or an external reference available on one of the pins. Symbolic names provided in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name | Description |
|---|---|
| ADC10_CAL_VBG | The signal used for calibration comes from VBG the analog reference |
| ADC10_CAL_AMUXBUS | The signal used for calibration comes from a pin connected to AMUXBUS |
| ADC10_CAL_P0_0 | The signal used for calibration comes from Port_0_0 (Not available for Col 0) |
| ADC10_CAL_P0_1 | The signal used for calibration comes from Port_0_1 |
| ADC10_CAL_P0_2 | The signal used for calibration comes from Port_0_2 (Not available for Col 0) |
| ADC10_CAL_P0_3 | The signal used for calibration comes from Port_0_3 |
| ADC10_CAL_P0_4 | The signal used for calibration comes from Port_0_4 (Not available for Col 0) |
| ADC10_CAL_P0_5 | The signal used for calibration comes from Port_0_5 |
| ADC10_CAL_P0_6 | The signal used for calibration comes from Port_0_6 (Not available for Col 0) |
| ADC10_CAL_P0_7 | The signal used for calibration comes from Port_0_7 |

**Return Value:**

Returns an integer indicating the nearest result to the iVal that was possible.

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_StartADC

**Description:**

Turns on the ADC and runs it continuously. Global interrupts must be enabled for the ADC to function.

**C Prototype:**

```
void  ADC10_StartADC(void)
```

**Assembly:**

```
lcall  ACD10_StartADC
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_StopADC

**Description:**

Immediately stops the ADC.

**C Prototype:**

```
void  ADC10_StopADC (void)
```

**Assembly:**

```
lcall  ADC10_StopADC
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_fIsDataAvailable

**Description:**

Checks the status of the ADC.

**C Prototype:**

```
BYTE  ADC10_fIsDataAvailable(void)
```

**Assembly:**

```
lcall  ADC10_fIsDataAvailable
```

**Parameters:**

None

**Return Value:**

Returns a nonzero value if data is converted and ready to read.

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_ClearFlag

**Description:**

Resets the data-available flag.

**C Prototype:**

```
void  ADC10_ClearFlag(void)
```

**Assembly:**

```
lcall  ADC10_ClearFlag
```

**Parameters:**

None

**Return Value:**

None

**Side Effect:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_iGetData

**Description:**

Returns converted data. ADC10_fIsDataAvailable() must be called to verify that the data sample is ready.

**C Prototype:**

```
int   ADC10_iGetData(void)
```

**Assembly:**

```
lcall  ADC10_iGetData
```

**Parameters:**

None

**Return Value:**

Returns the converted data sample. A value of 0xFFFF indicates the input is out of range.

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_iGetDataClearFlag

**Description:**

Returns converted data and resets the data available flag. ADC10_fIsDataAvailable() must be called to verify that the data sample is ready.

**C Prototype:**

```
int  ADC10_iGetDataClearFlag(void)
```

**Assembly:**

```
lcall  ADC10_iGetDataClearFlag
```

**Parameters:**

None

**Return Value:**

Returns the converted data sample. A value of 0xFFFF indicates the input is out of range.

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_StartTempMeasurement

**Note** The API is available only when the "Die Temp Measurement" parameter is set to "Enabled")

**Description:**

Stores the input of the ACE01 Block in the internal variable and connects the VTemp sensor output to the PMux input. Then starts the ADC measurement. For an example of the API usage, see the "Die Temperature Measurement" section.

**C Prototype:**

```
void ADC10_StartTempMeasurement (void)
```

**Assembly:**

```
lcall  ADC10_StartTempMeasurement
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

## ADC10_GetTemperature

**Note** This API is available only when the "Die Temp Measurement" parameter is set to "Enabled"

**Description:**

Stops the ADC measurement, calculates Die Temperature value, restores the ADC input, and resets the data-available flag. The ADC10_fIsDataAvailable API must be called before the API to verify that the data sample is ready. For an example of the API usage, see the "Die Temperature Measurement" section.

The Die Temperature is calculated by using the following equation:

DieTemp = M * wADCCount * wCalibrationVoltage / wCalibrationCount - B

In this equation:

wADCCount - result of the ADC VTemp voltage measurement;

wCalibrationVoltage - the API parameter;

wCalibrationCount - internally stored value of the iVal parameter of the ADC10_ iCal(int iVal, BYTE bCalIn) API function;

M, B - coefficients that are stored in the flash.

**C Prototype:**

```
CHAR ADC10_GetTemperature(WORD wCalibrationVoltage)
```

**Assembly:**

```
mov A, [wCalibrationVoltage+1] ; LSB of API parameter
mov X, [wCalibrationVoltage] ; MSB of API parameter
lcall  ADC10_GetTemperature
```

**Parameters:**

WORD wCalibrationVoltage: The calibration voltage (in mV) that was used for the ADC calibration. This voltage corresponds to the iVal parameter (counts) of the ADC10_iCal API function.

**Return Value:**

'A' contains the Die Temperature in Celsius degrees.

**Side Effects:**

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, the calling function is responsible to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

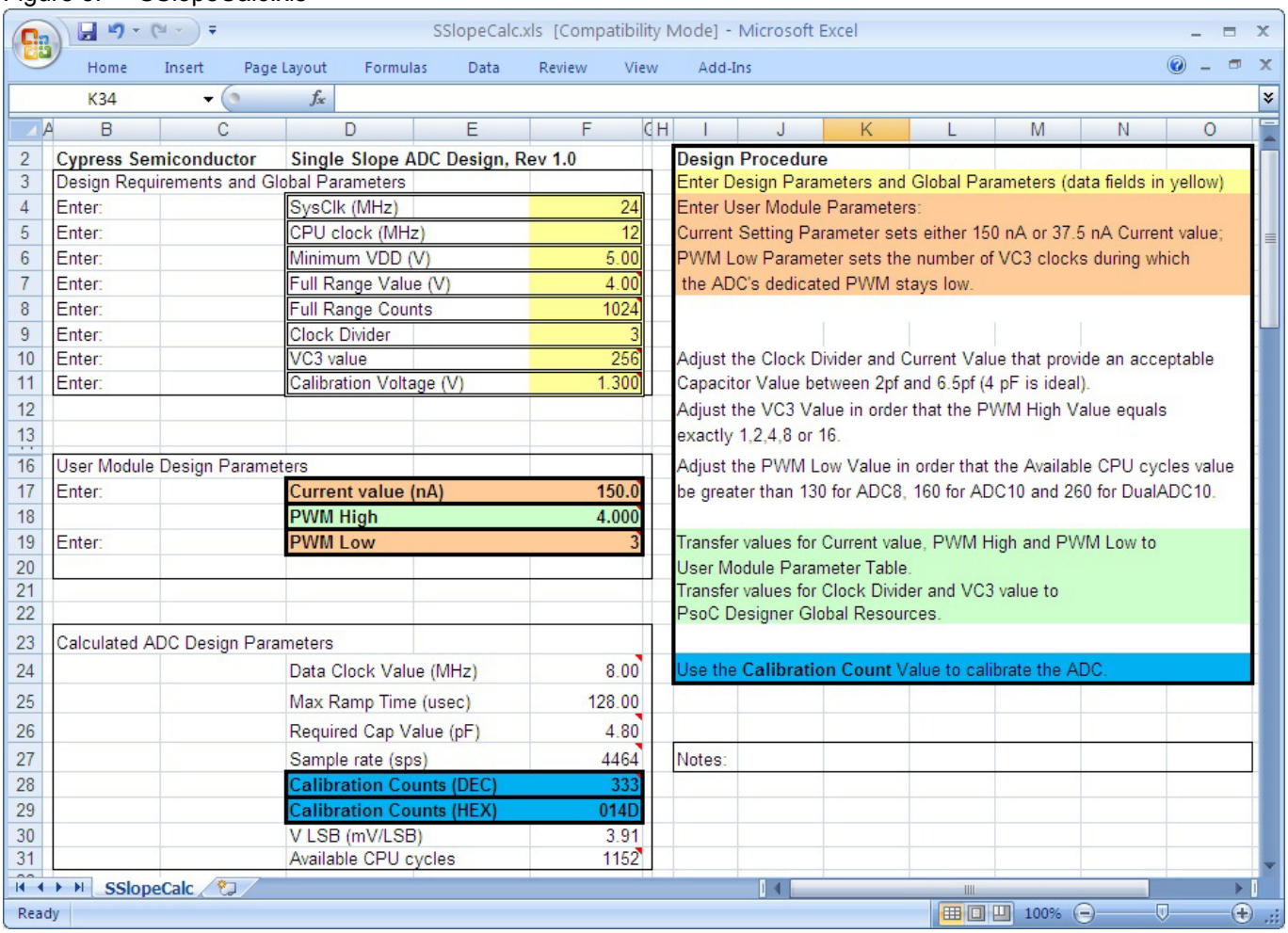
## Calibration Example

There are five parameters that are used to correctly configure the ADC. They are:

- VC3 Source (and Data Clock)
- VC3 Divider
- Current Setting
- PWM_High
- PWM_Low

These five parameters can be set for a desired input range and sample rate. These parameters are not independent and care must be taken to define an operable solution. To assist with this, a spreadsheet is included to help with this selection process. You can access it in PSoC Designer by going to **Help** => **Documentation** => **SingleSlopeADCs** => **SSlopeCalc.xls.**

Figure 3. SSlopeCalc.xls




For this example, configure a 10 bit ADC with an input range of 4V.

As shown in Figure 1, a ramp is generated by pushing a current into a capacitor. The resulting equation is Equation 1:

**Equation 1**

$$i = C \cdot \frac{\Delta V}{\Delta t}$$

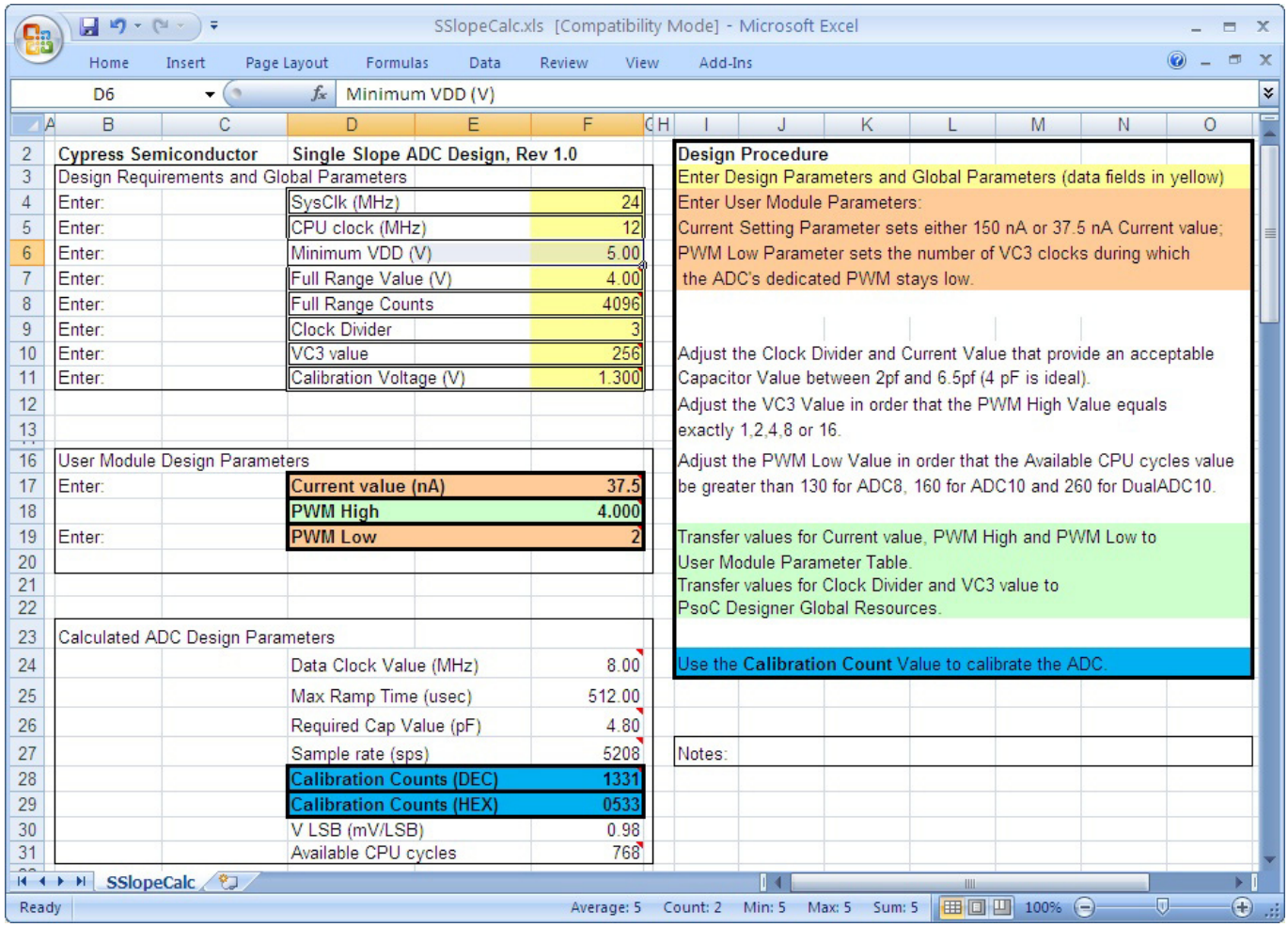
The delta voltage is the input range (4V) and the delta time is determined by the data clock and ADC resolution (1024). Equation 2 shows the capacitor value as a function of these three parameters:

**Equation 2**

$$C = i \times \frac{(\mathrm{FullRange}_{Counts} / \mathrm{DataClock})}{(\mathrm{FullRange}_{Volts})}$$

The capacitor in the ramp generator can be adjusted from 2 pF to 6.5 pF. Ideally, it must be close to 4 pF. The three parameters must be adjusted to meet this qualification.

1. Open the spreadsheet and enter a **SysClk** value of 24 MHz, a **Minimum VDD** of 5V, **Full Range Value** of 4V, and a **Full Range Count** value of 1024.
2. Select values for **Clock Divider** and **Current Value** that provide an acceptable capacitor value. A **Clock Divider** of 3 results in a **Data Clock Value** of 8 MHz and a **Current Value** of 37.5 nA sets the required capacitor value to 4.8 pF as shown in the following figure. There are many possible solutions, but this value is used for the example.



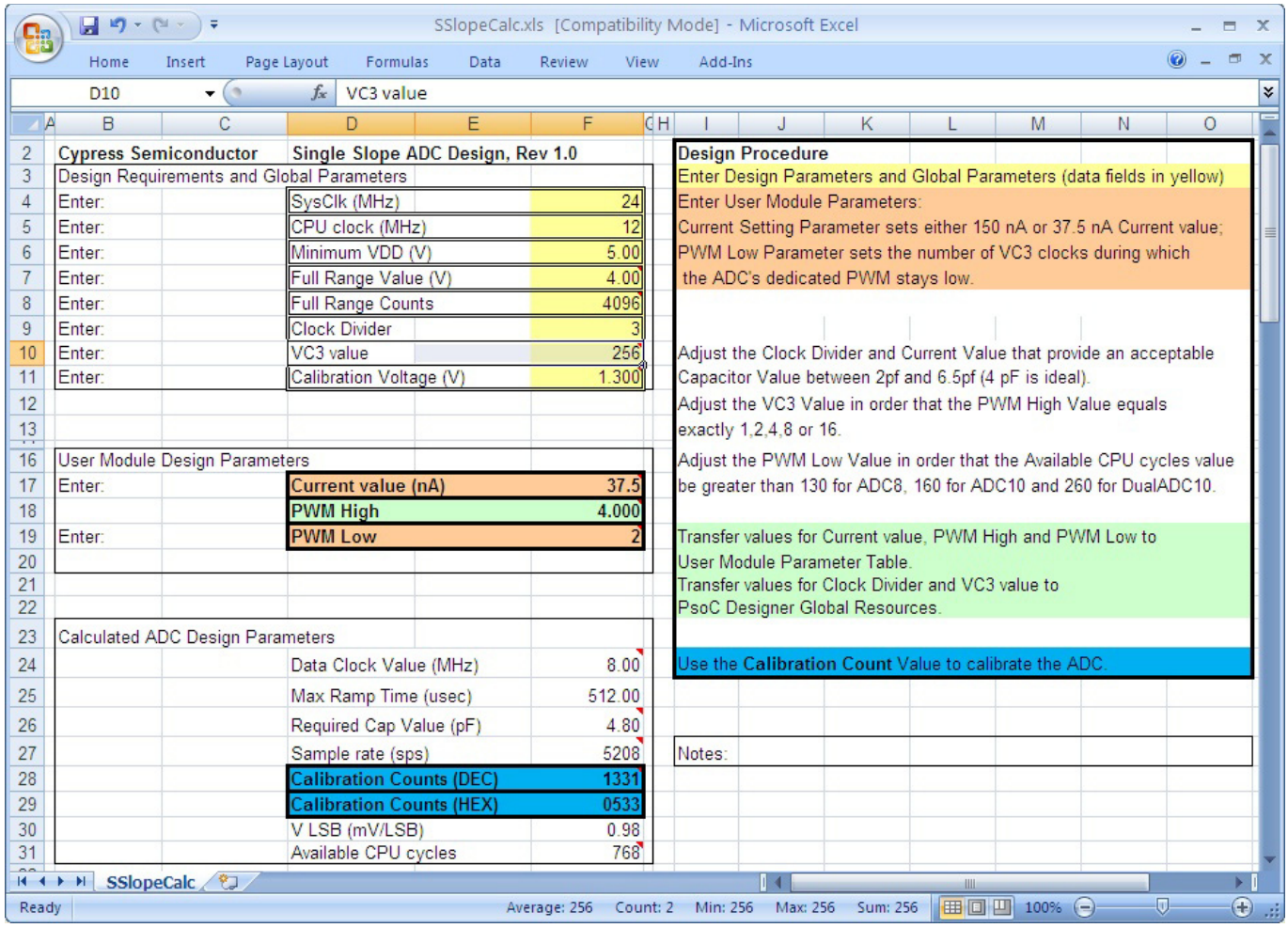

In PSoC Designer, set **VC1=SysClk/N** to `3` to generate a 8 MHz clock. Set the **VC3 Source** and **Data Clock** value to `VC1`.

The Full Range Counts is defined by Equation 3:

**Equation 3**

$$FullRange_{Counts} = VC3_{divider} \cdot PWM_{high}$$

3. The **VC3 Divider Value** and **PWM High Value** must be selected so that their product equals the full range count (1024). The constraints are that the VC3 Divider must be <= 256 and the PWM High Value can be 1, 2, 4, 8, or 16. As shown later in this section, one solution is to set the VC3 Divider Value to 256. Doing so causes the spreadsheet to set PWM High Value to 4 (one of the acceptable values).

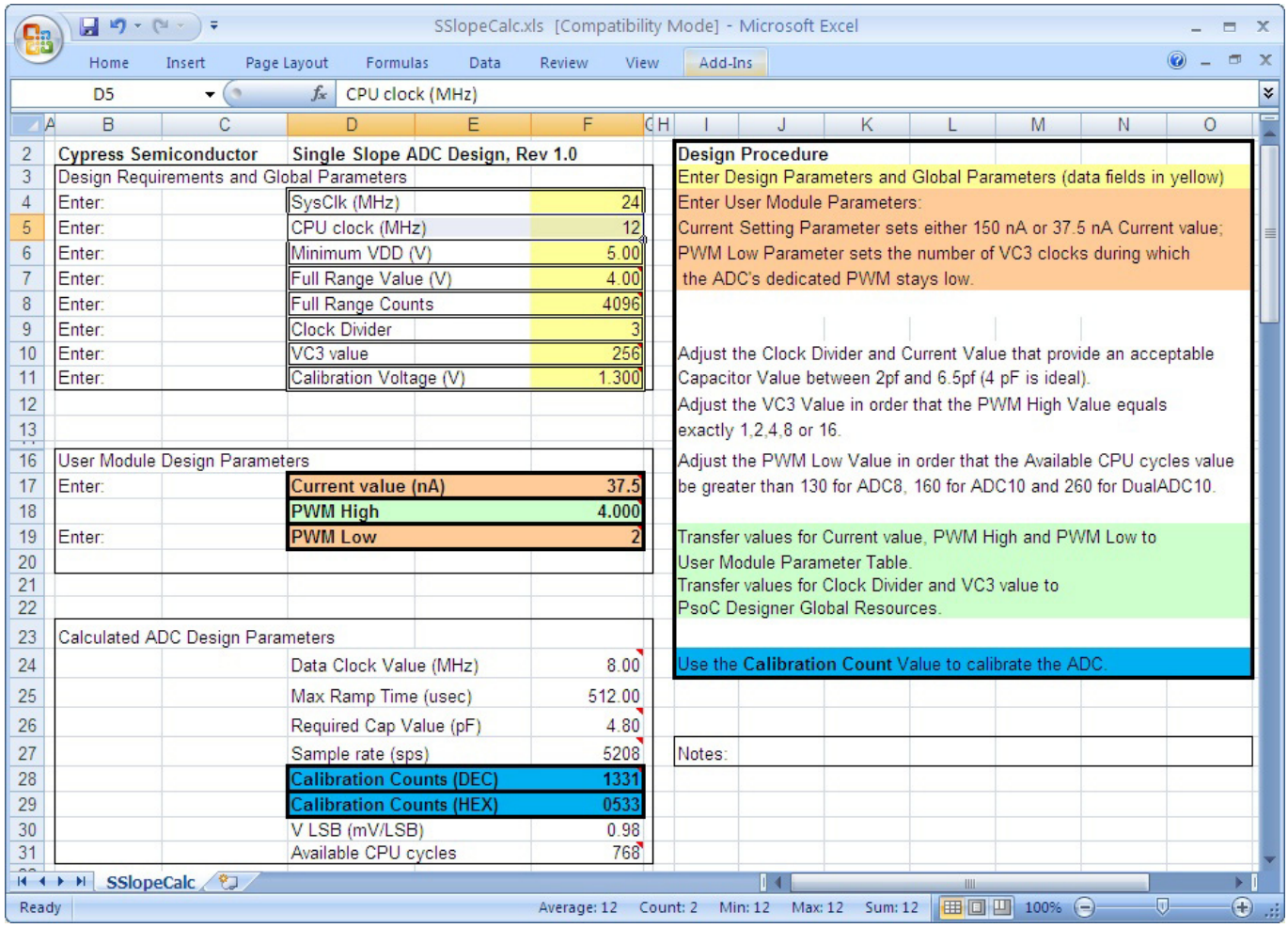



The PWM Low parameter sets the time available to process the answer. The software requires 160 CPU clocks. Equation 4 defines the number of available CPU clocks given the Data Clock, CPU clock, and PWM Low values.

**Equation 4**

$$CPUCycles = \frac{CPUclock}{Dataclock} \cdot VC3_{divider} \cdot PWM_{low}$$

4. Select a **CPU Clock** value and a **PWM Low** value that results in more than 160 available CPU cycles. As shown in the following figure, a CPU clock of 12 MHz and a PWM Low Value of 2 results in 768 available CPU cycles. This is far larger than the minimum 160 cycles. The greater the available CPU cycles, the more time available for processing the answer in the presence of other interrupts.




The sample rate is calculated using Equation 5:

**Equation 5**

$$SampleRate = \frac{Dataclock}{VC3} \cdot \frac{1}{PWM_{high} + PWM_{low}}$$

For this example it works out to 5.208 ksps.

This ADC has to be calibrated. This is done by calling ADC10_iCal. This routine requires two parameters:

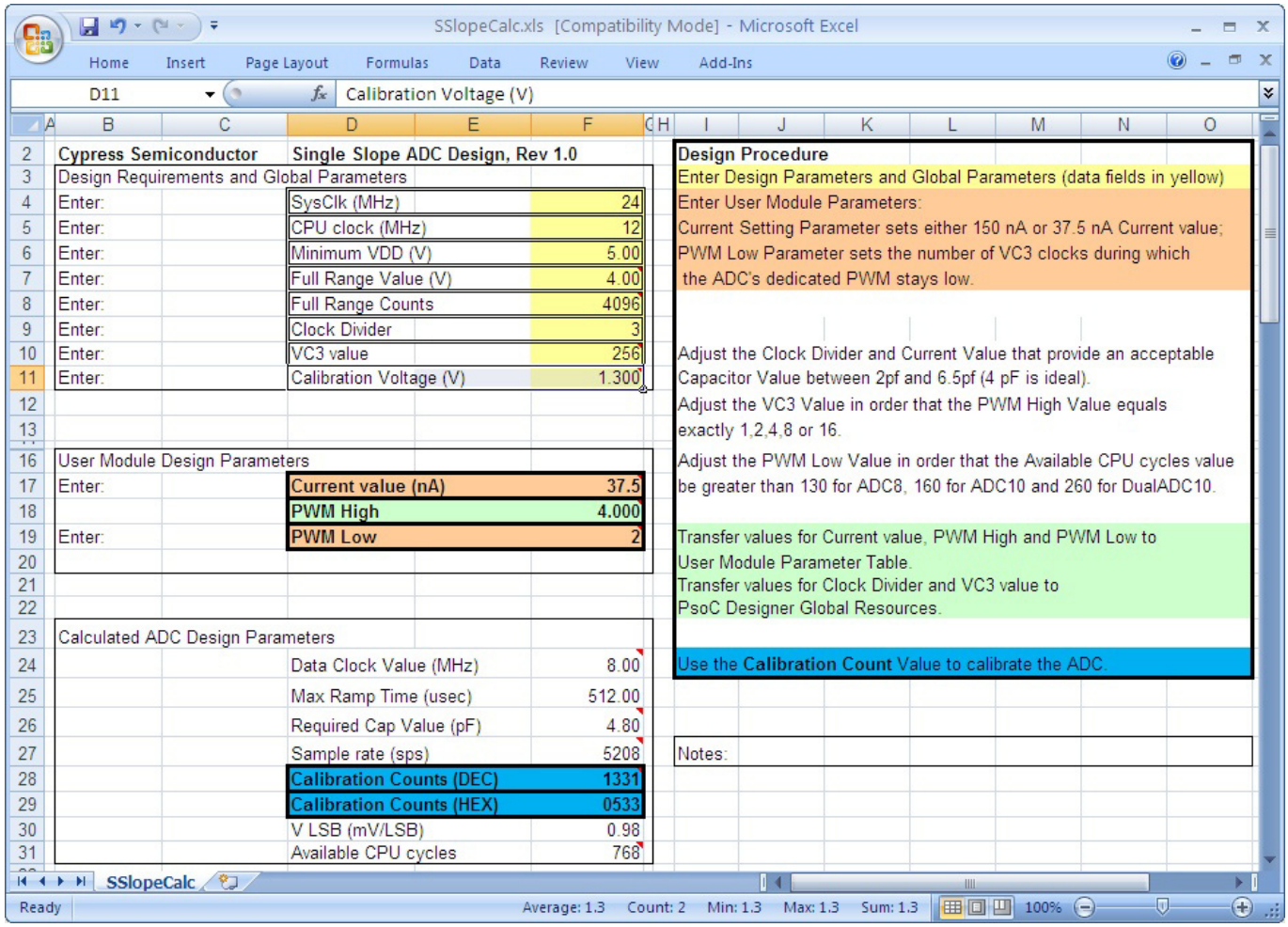
- The calibration voltage value
- The calibration count value

A 1.3V bandgap reference is provided to perform this calibration. Although an external reference can be brought in, 1.3V value may be used. Equation 6 shows that for the generated slope, the voltage to time ratio remains constant.

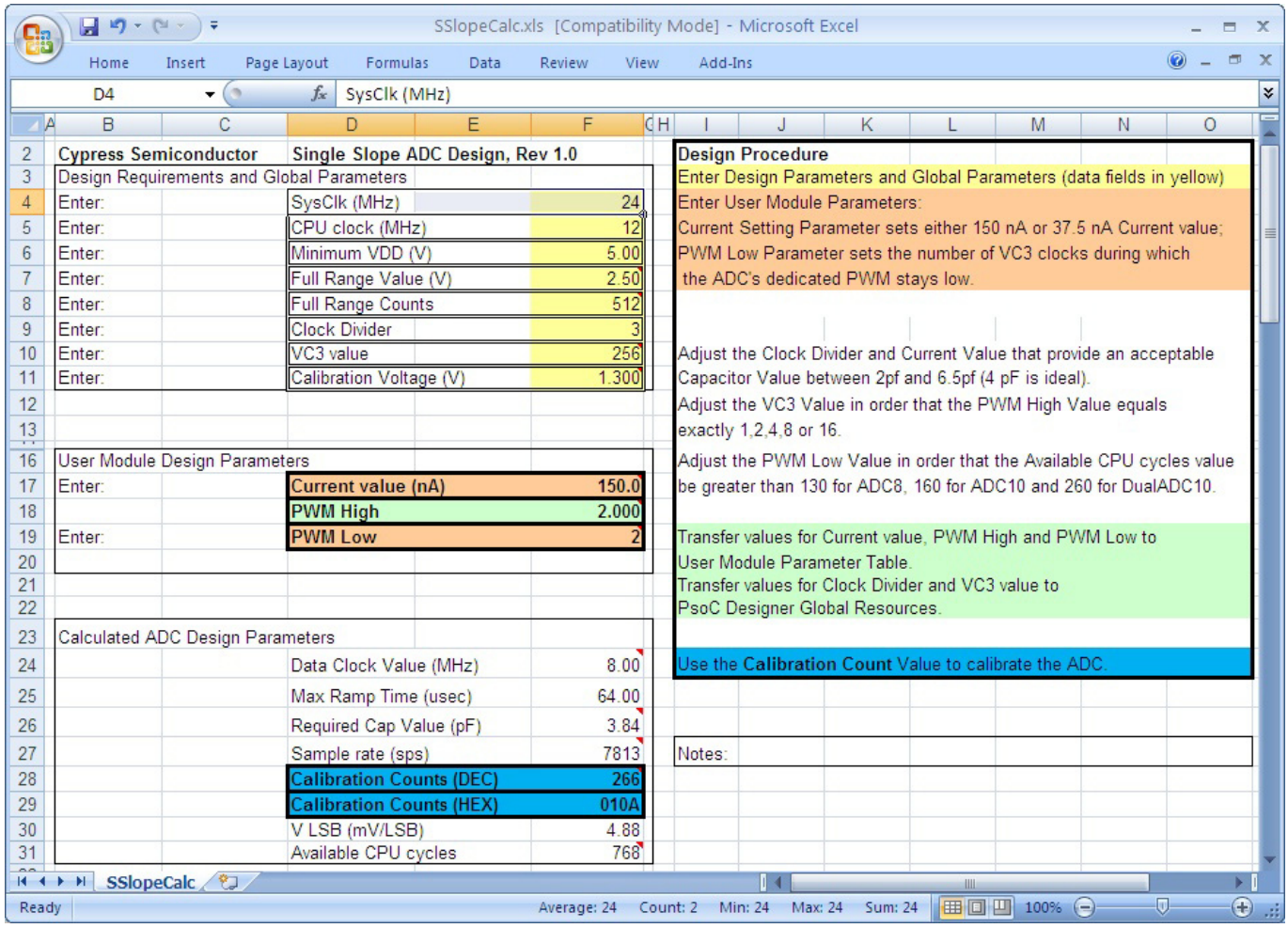
**Equation 6**

$$\frac{FullRange_{Volts}}{FullRange_{Counts}} = \frac{CalValue_{Volts}}{CalValue_{Counts}}$$

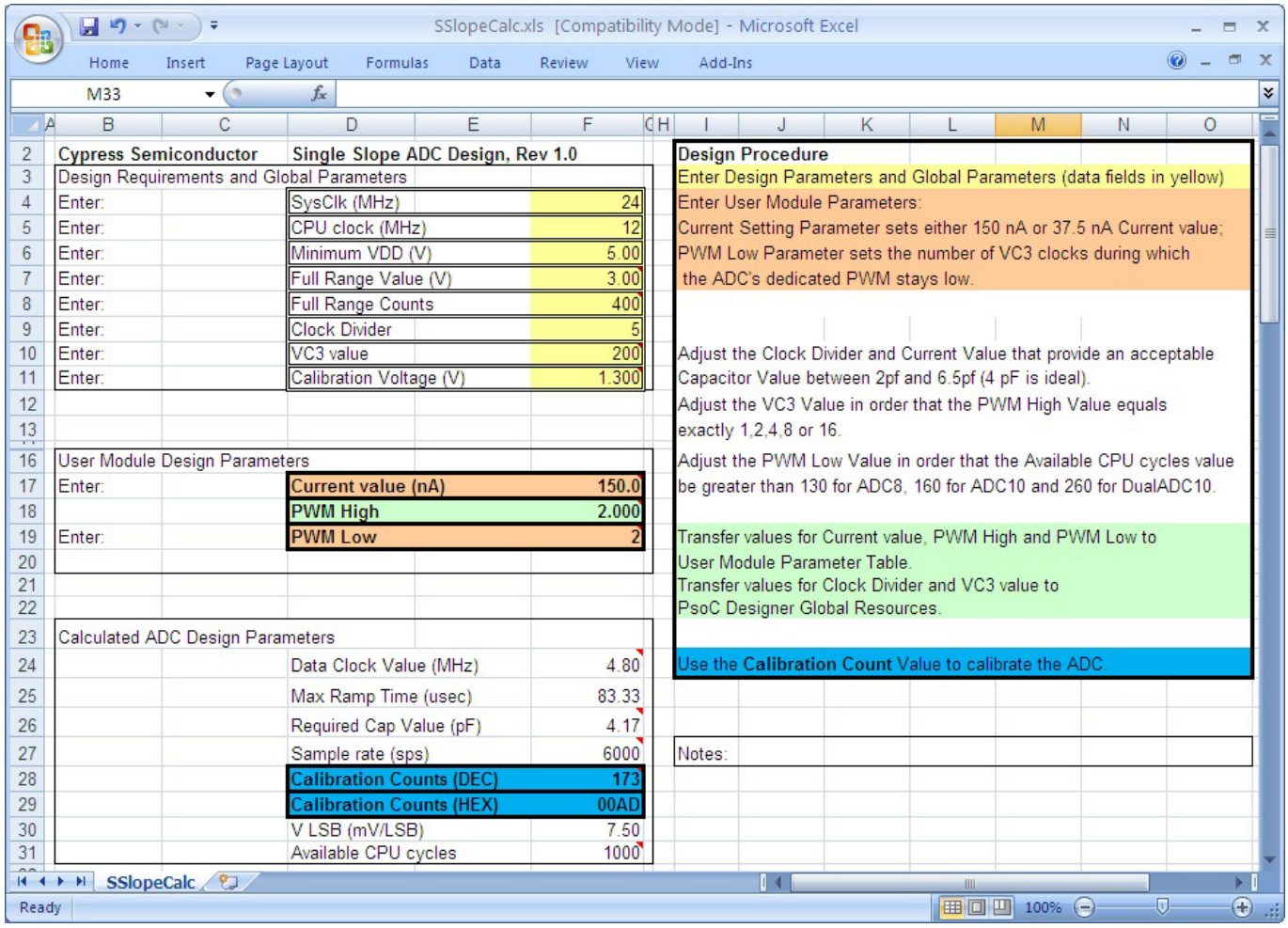
5. For this example the ADC is calibrated with the bandgap voltage (1.3V). In the spreadsheet, set the **Calibration Voltage** value to 1.3V. As shown in the following figure, the spreadsheet calculates the **Calibration Counts Value**. It is this value that is used the calibrate the ADC. For this example, the Calibration Count Value is 1331 counts.



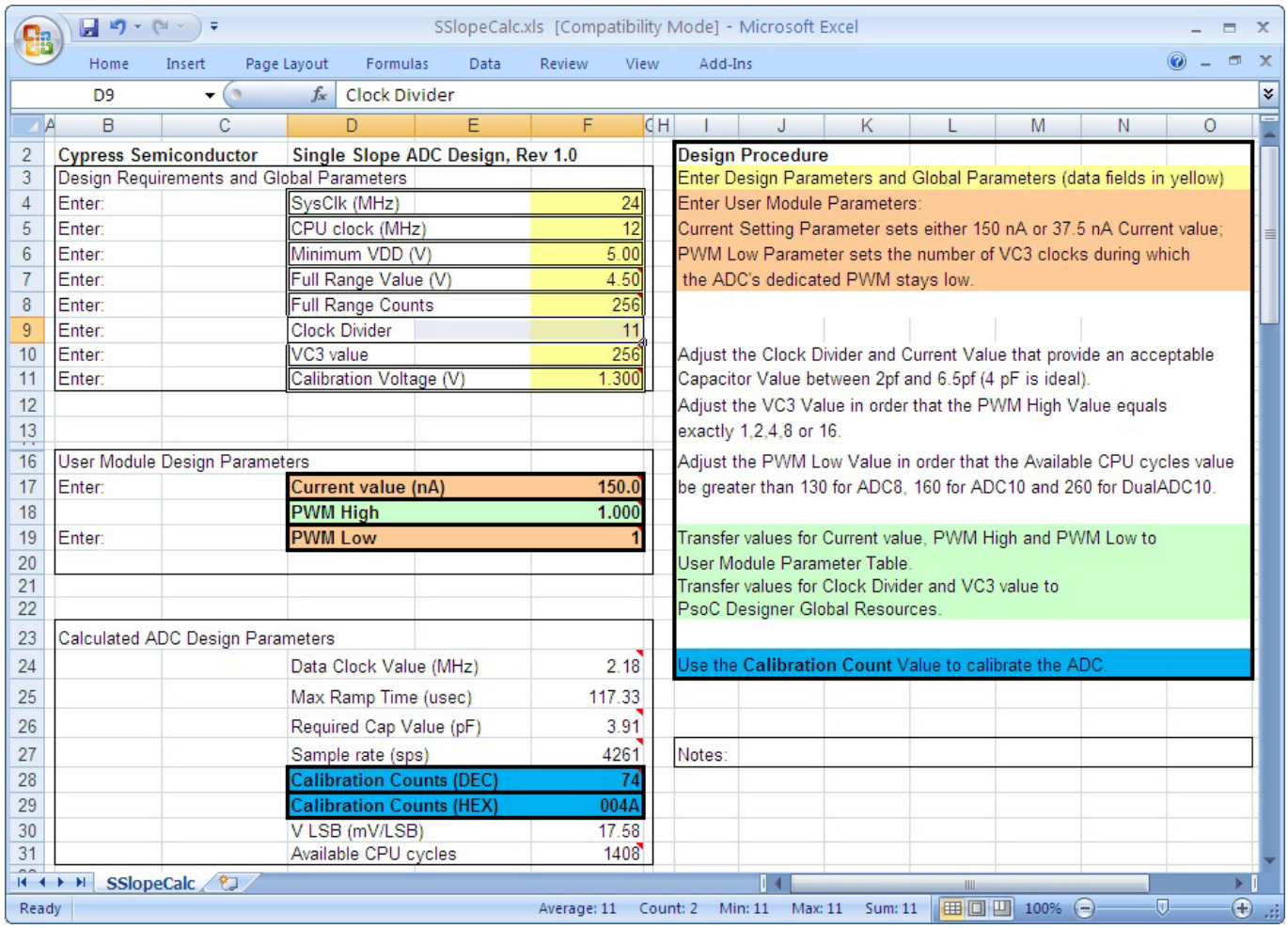
Here is a spreadsheet for a 9 bit 2.5V ADC solution:



Here is the spreadsheet for a 400 count/3V ADC solution. It is not necessary to have a binary range.



Here is the spreadsheet for a 8-bit/4.5V ADC solution. This ADC can be used for 8-bit applications instead of the ADC8.




## Comments on Calibration

Calibration requires a calibration voltage and desired calibration count. With these two variables, the routine calibrates the ADC. It returns the ADC value of the reference voltage. For calibration with the bandgap, the calibration routine takes this form:

iCalYouGot = ADC10_iCal(iCalYouWanted, ADC10_CAL_VBG);

If the value obtained (what you got) is close to the expected value (what you wanted), the gain error may be acceptable. If the ADC calibrates for 1.3V to 334 counts instead of 333, then the gain error is only 0.3% (334/333) off. Determine how close the obtained value must be to the expected value for each application. If an adjustment is needed, the following correction must be run on all ADC data:

iAdjustedADCValue = (iCalYouWanted/iCalYouGot)*iADCValue;

For example, if the ADC was calibrated to the bandgap, the following line of code converts the ADC value to millivolts.

```
iADCmV =(int) ((long)1300 * (long)iADCValue * (long)iCalYouWanted / (long)iCalYouGot);
```

The input multiplexer determines which signal is digitized.

## Die Temperature Measurement

The Die Temperature can be measured only when the user module occupies the ACE01 analog block and the "Die Temp Measurement" parameter is set to "Enabled".

Example code:

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
    CHAR cTemp;
    WORD wVoltage;

    M8C_EnableGInt;
    ADC10_Start(ADC10_FULLRANGE); /* Start the User Module*/

/* Calibrates the user module using the internal BandGap reference source */
    ADC10_iCal(0x14d, ADC10_CAL_VBG); // 1.3V = 333 counts

while (1)
    {
        ADC10_StartTempMeasurement(); /* Begin sampling */
        while(!ADC10_fIsDataAvailable()); /* Wait till the end of ADC sampling */
        /* Get the Die Temperature and reset the data-available flag*/
        cTemp = ADC10_GetTemperature(1300);

        ADC10_StartADC();                    /* Begin sampling */
        while(!ADC10_fIsDataAvailable());  /* Wait till the end of ADC sampling */
        /* Get the ADC result and reset the data-available flag*/
        wVoltage = ADC10_iGetDataClearFlag();
  ADC10_StopADC();

  /* Add user code here to display the results of
        voltage and temperature measurement */
    }
}
```

### Note

1. The ADC must be calibrated before calling the ADC10_StartTempMeasurement API.
2. The ADC10_StartTempMeasurement API calls the ADC10_StartADC API; as a result it is not necessary to call the API before starting the temperature measurement.
3. The ADC10_GetTemperature API stops the ADC converting. In case you need to measure a voltage after Die Temperature measurement, the ADC10_StartADC API must be called.

## Sample Firmware Source Code

This sample code polls the Flag register, reads the ADC value, and with some user defined code, displays the data. If the parameters for the project are set as shown earlier, the output is 10bit/4V range ADC values.

```
;;; Sample Code for the ADC10
;;; Continuously Sample input voltage
;;;
include "m8c.inc"
include "PSoCAPI.inc"
export _main
_main:
M8C_EnableGInt                  ; Enable global interrupts
mov  A,ADC10_FULLRANGE
call ADC10_Start
mov  A, ADC10_CAL_VBG
push A
mov  A, 01h
push A
mov  A, 4dh
push A
call ADC10_iCal                 ; Calibrate the ADC to 1.3V = 333
   call ADC10_StartADC             ; Begin Sampling
wait:
call ADC10_fIsDataAvailable     ; Wait for vaild data
   jz wait
call ADC10_iGetDataClearFlag
;;  Add user code to display the result
   jmp wait
```

The same project written in C is:

```
//-------------------------------------------------------------------
// Sample C Code for the ADC10
// Continuously Sample input voltage
//-------------------------------------------------------------------
#include <m8c.h>
#include "PSoCAPI.h"
int iData;
void main(void){
   M8C_EnableGInt;
ADC10_Start(ADC10_FULLRANGE);      // Start the User Module
ADC10_iCal(0x14d, ADC10_CAL_VBG);  // 1.3V = 333 counts
ADC10_StartADC();                  // Begin sampling
while(1)    {
while(0 == ADC10_fIsDataAvailable());
iData = ADC10_iGetDataClearFlag();
// Add user code here to display the result
}
}
```

## Configuration Registers

The input multiplexer determines which signal is digitized.

Table 4. Block ADC: Register ACE_CR1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 1 | 0 | 1 | Input | | |

The Input bits determine the signal source for the A/D conversion. This value is set by the Input parameter and can be changed to provide alternative reference voltage for calibration.

Table 5. Block ADC: Register ACE_CR2

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Enable |

The Enable bit is used to start the user module.

Table 6. Block RAMP: Register ASE_CR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 7. Block RAMP Register ADC_CR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | CMPST | 1 | 1 | 0 | 0 | 1 | 0 | Enable |

The Enable bit is used to start the user module. The CMPST bit is a read-only bit that indicates whether the comparator tripped during the conversion period or not. This is used to determine whether or not an over range condition occurred.

Table 8. Block RAMP: Register ADC_TR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | Capacitor Trim Value | | | | | | | |

The Capacitor Trim Value is set during the ADC10_iCal function. This 8-bit value sets the value of the adjustable capacitor and thus the slope of the reference ramp used by the ADC.

The CNT is a digital PSoC block configured as a counter.

Table 9. Block CNT: Register Function

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Table 10. Block CNT: Register Input

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | Data | | | | Clock | | | |

Data selects the column comparator where the ADC block has been placed. Clock selects the input clock from one of 16 sources and is set in the Device Editor.

Table 11. Block CNT: Register Output

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 12. Block CNT: Register DR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | Count Value | | | | | | | |

Table 13. Block CNT: Register DR1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 14. Block CNT: Register DR2

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | Data Out | | | | | | | |

Data Out is used by the API to get the counter value.

Table 15. Block CNT: Register CR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Enable |

Enable enables CNT when set. It is modified and controlled by the ADC10 API.

Table 16. Register CMP_CR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | | | | | | | AINT1 | AINT0 |

The AINT0/1 bits are used to specify the source of the Analog Column Interrupts. The placement of the user module decides which of the two bits are set. AINT0 is set if the user module is placed in Column 0, and AINT1 is set if the user module is placed in column 1.

Table 17. Register INT_MSK

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | | | | | | | AColumn1 | AColumn2 |

The mask bits corresponding to the Analog Column are set here to enable their respective interrupts. The actual mask values are determined by the placement position of each block.

# Version History

| Version | Originator | Description |
|---|---|---|
| 1.1 | DHA | Added DRC to check if:<br><br>1. The source clock is different between digital and analog resources.<br><br>2. The ADC Clock is higher than CPU Clock. |
| 1.1.b | DHA | Added characterization data for CY8C21x23 devices. |
| 1.20 | DHA | Updated the ADC10 User Module to implement ADC10_StartTempMeasurement and ADC10_GetTemperature functions. |
| 1.30 | HPHA | Corrected method of clearing posted interrupts.Added design rules check for the situation when the ADC clock is faster than 8 MHz. |

**Note** PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.



Copyright © 2005-2013 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.