SAMD21 ARM Cortex-M0+ Microcontroller & MCC Harmony

SAM2001ADV v2.02

A Leading Provider of Smart, Connected and Secure Embedded Control Solutions

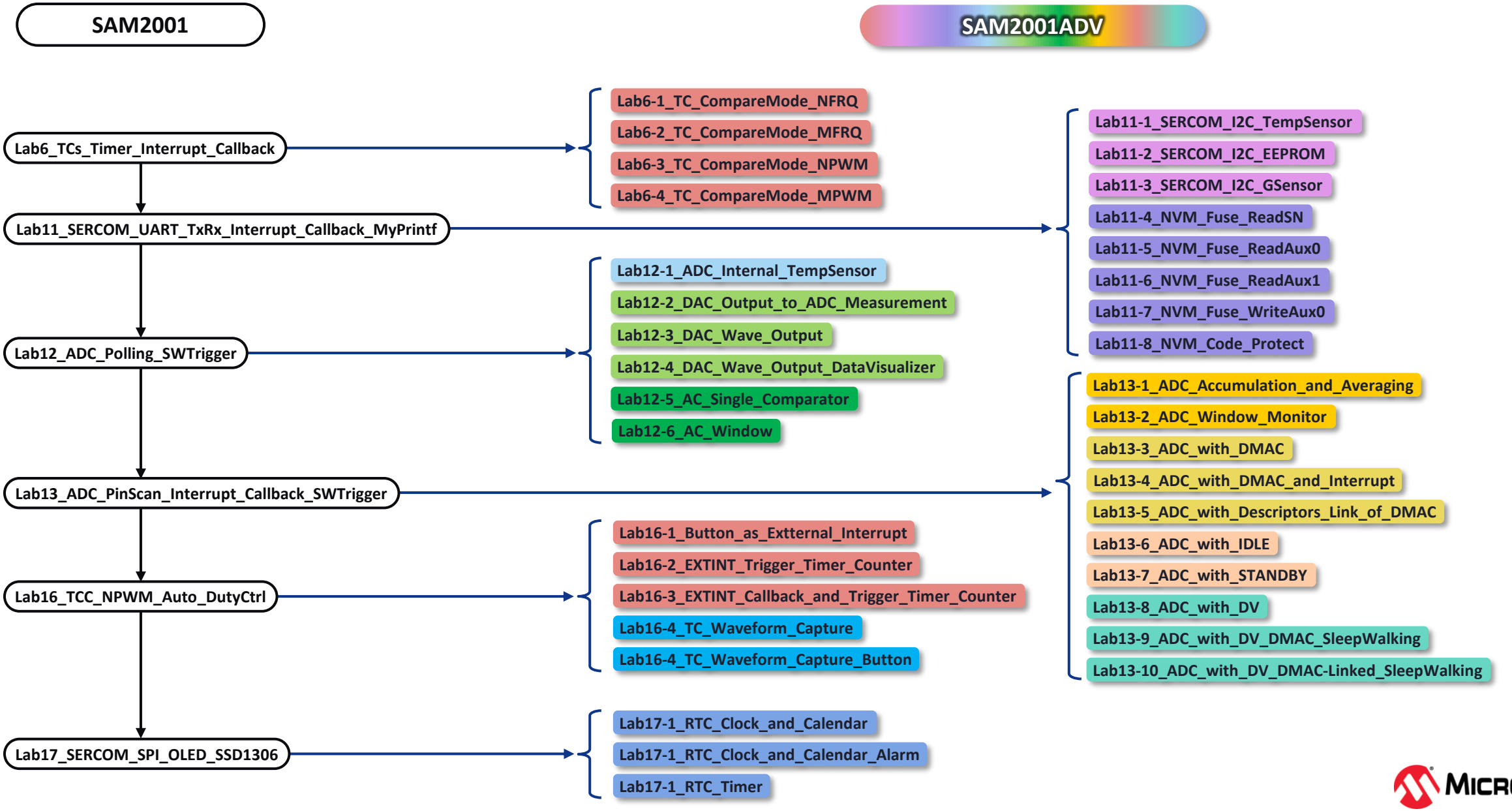
CAE Taiwan Team
Microchip Technology Inc.

2024 March

Class Objectives

- **When you finish this course, you will be able to:**
  - Be familiar with MPLAB X IDE and MCC Harmony
  - Understand how to configure and use the SAMD21 series peripherals.
- **There have two level course included.**
  - Major course : General study of basic features.
  - Appendix : Advance features and appendix.

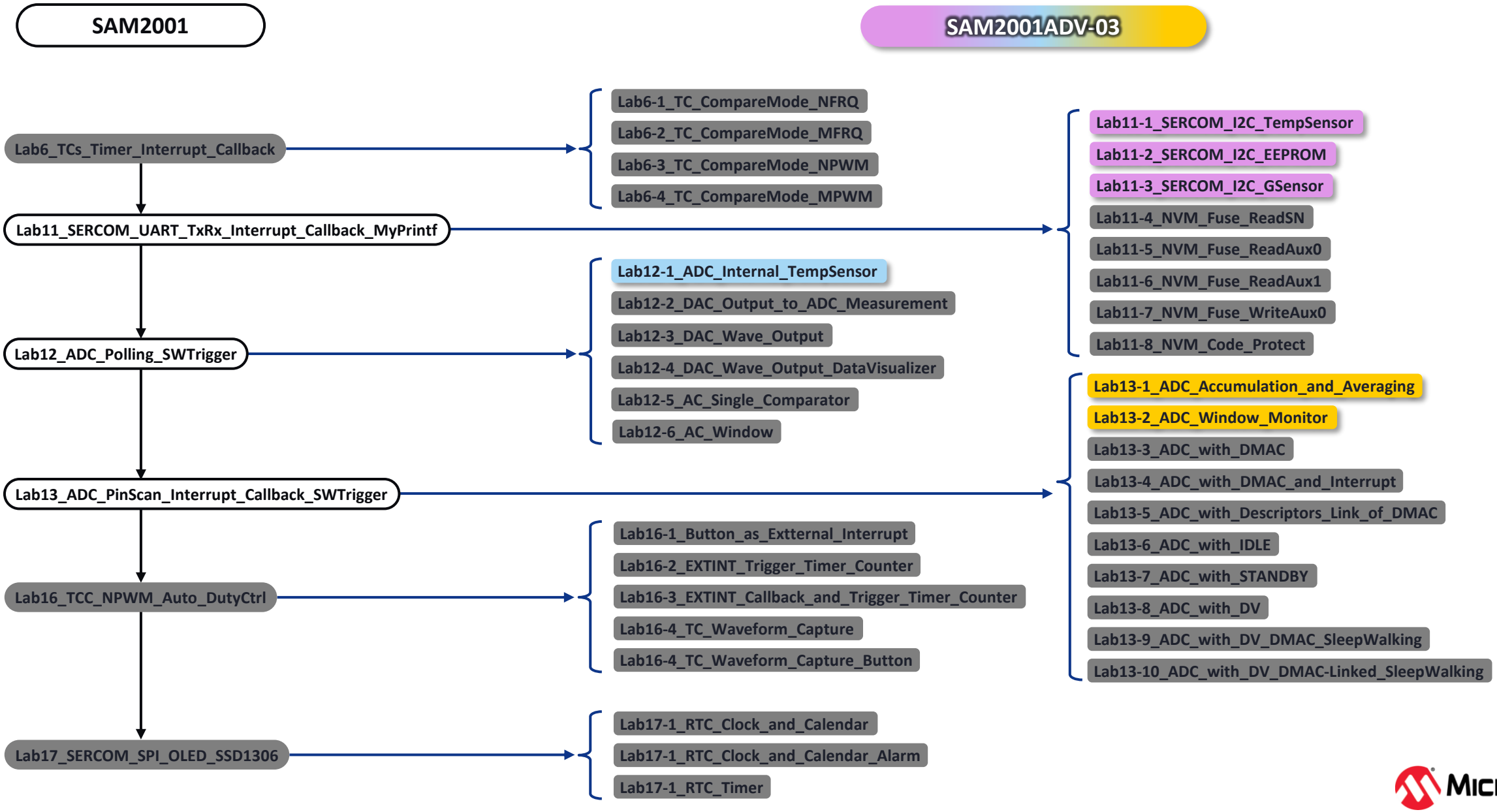
Lab Tree Branch from SAM2001 to SAM2001ADV



Separate Index for SAM2001ADV-03

- SAM2001ADV-03
  - SERCOM - I2C Architecture (Temp Sensor/EEPROM/G-Sensor)
  - ADC Accumulation, Averaging and Window Monitor
  - Internal Temperature Sensor (ADC)

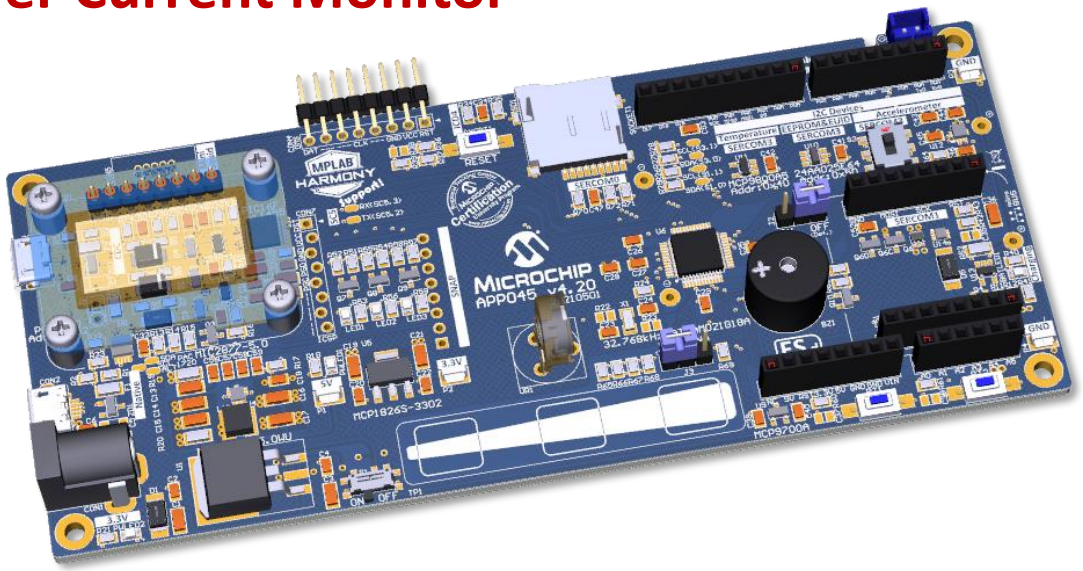
Separate Labs for SAM2001ADV-03



SAMD21 EVB APP045 v4
Introduction

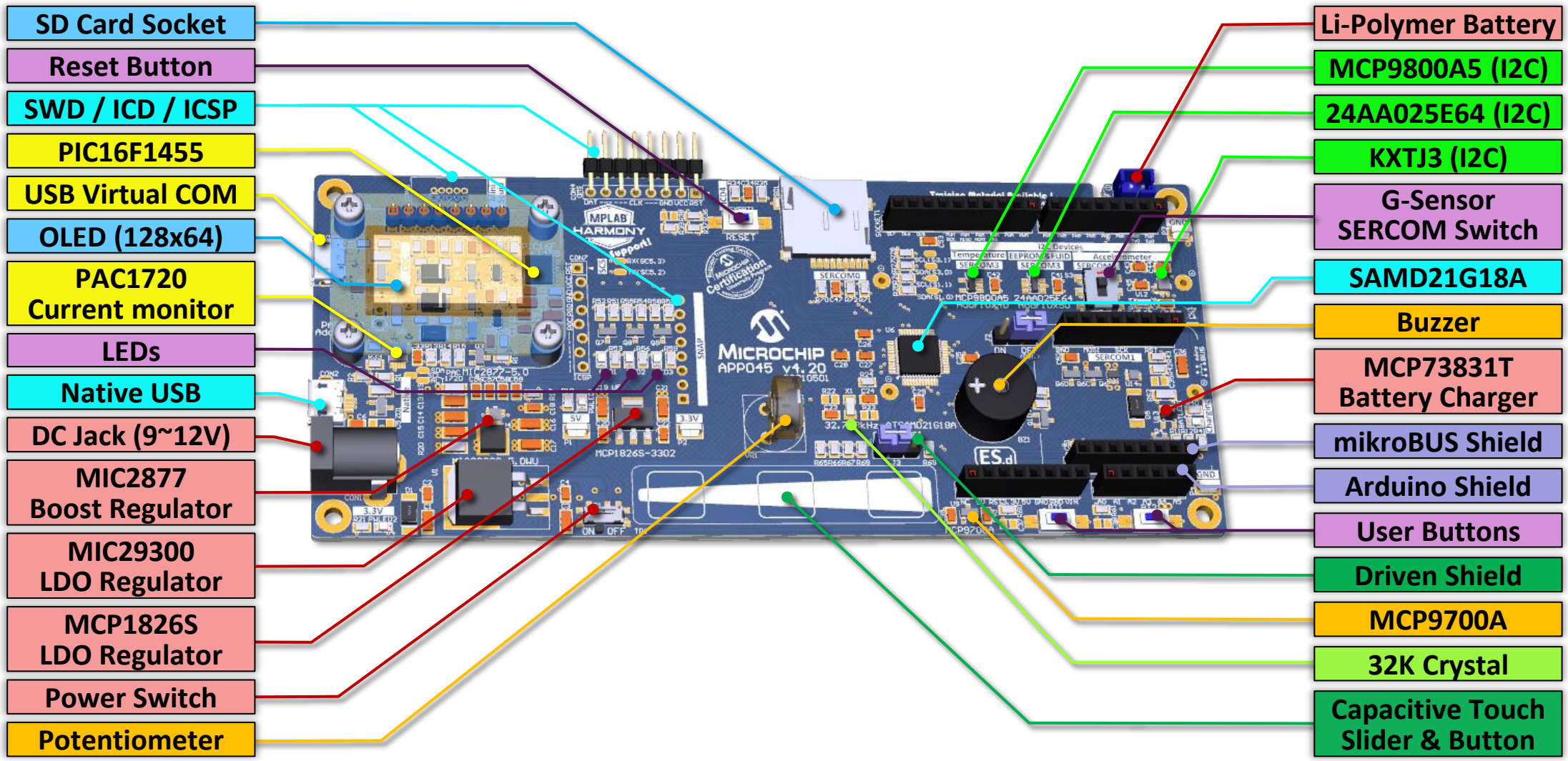
APP045 v4 Features

- On board SAMD21G18A 32Bit ARM Cortex M0+ Microcontroller
- Support plugged MPLAB SNAP external programmer/debugger
- Onboard USB CDC Virtual COM Port and Power Current Monitor
- Multiple Power Supply Source
- Support Li-on Battery, boost and charger
- 2 x Buttons, 3 x LEDs
- 1 x I2C Temperature Sensor
- 1 x I2C EEPROM with Unique ID
- 1 x I2C Accelerometer
- 1 x MicroSD Socket (SPI interface)
- 1 x SPI OLED (128 x 64, SSD1306 Compliant)
- 1 x Potentiometer, Analog Temperature Sensor and Buzzer
- 1 x Capacitive QTouch® Slider & Buttons with Driven Shield
- 1 x mikroBUS™ Click Board Socket
- 1 x Arduino® Shield Socket (Arduino M0 Pro board Compliant)



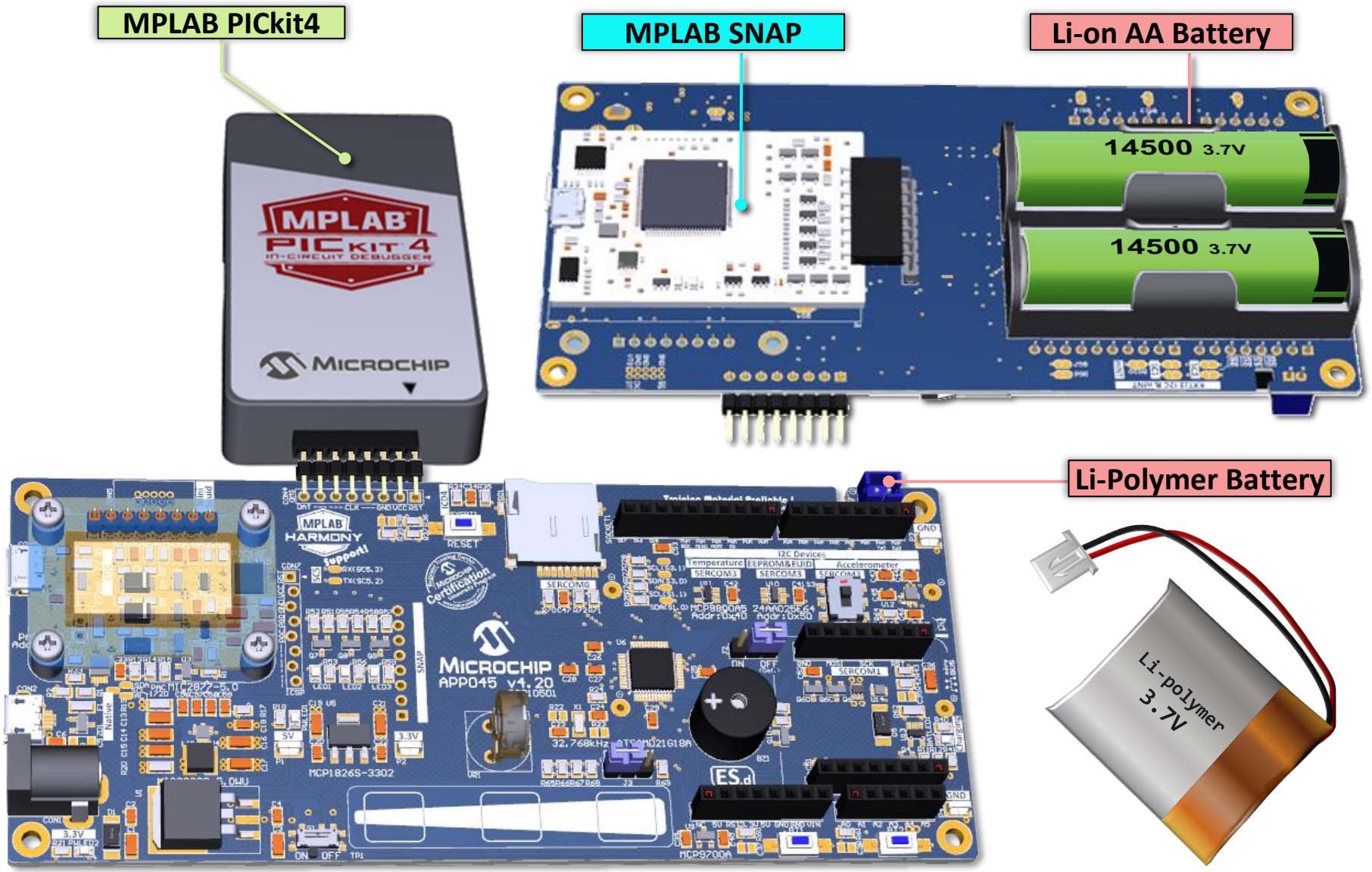
MICROCHIP

APP045 v4 Overview

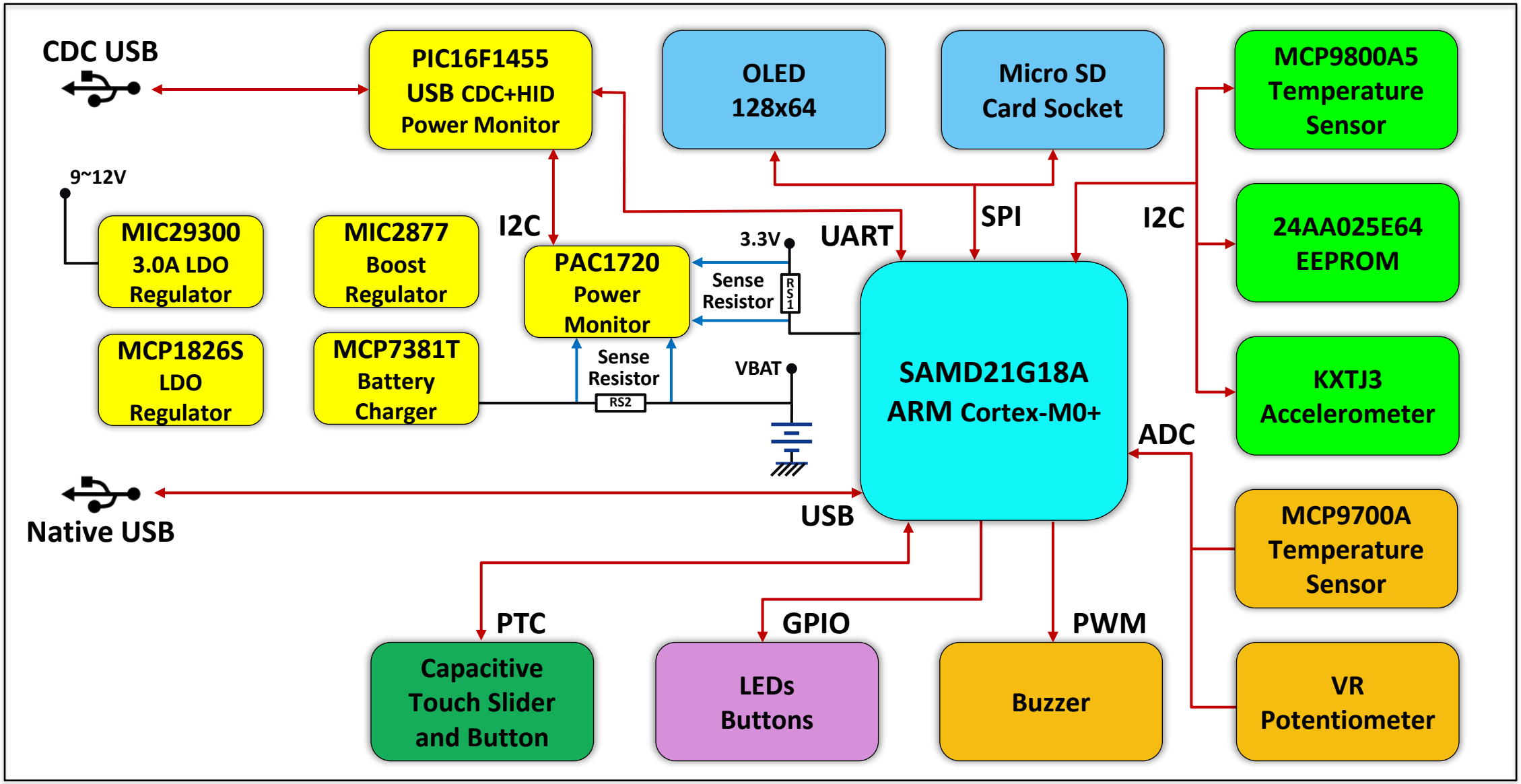


MICROCHIP

APP045 v4 Programmer and Debugger



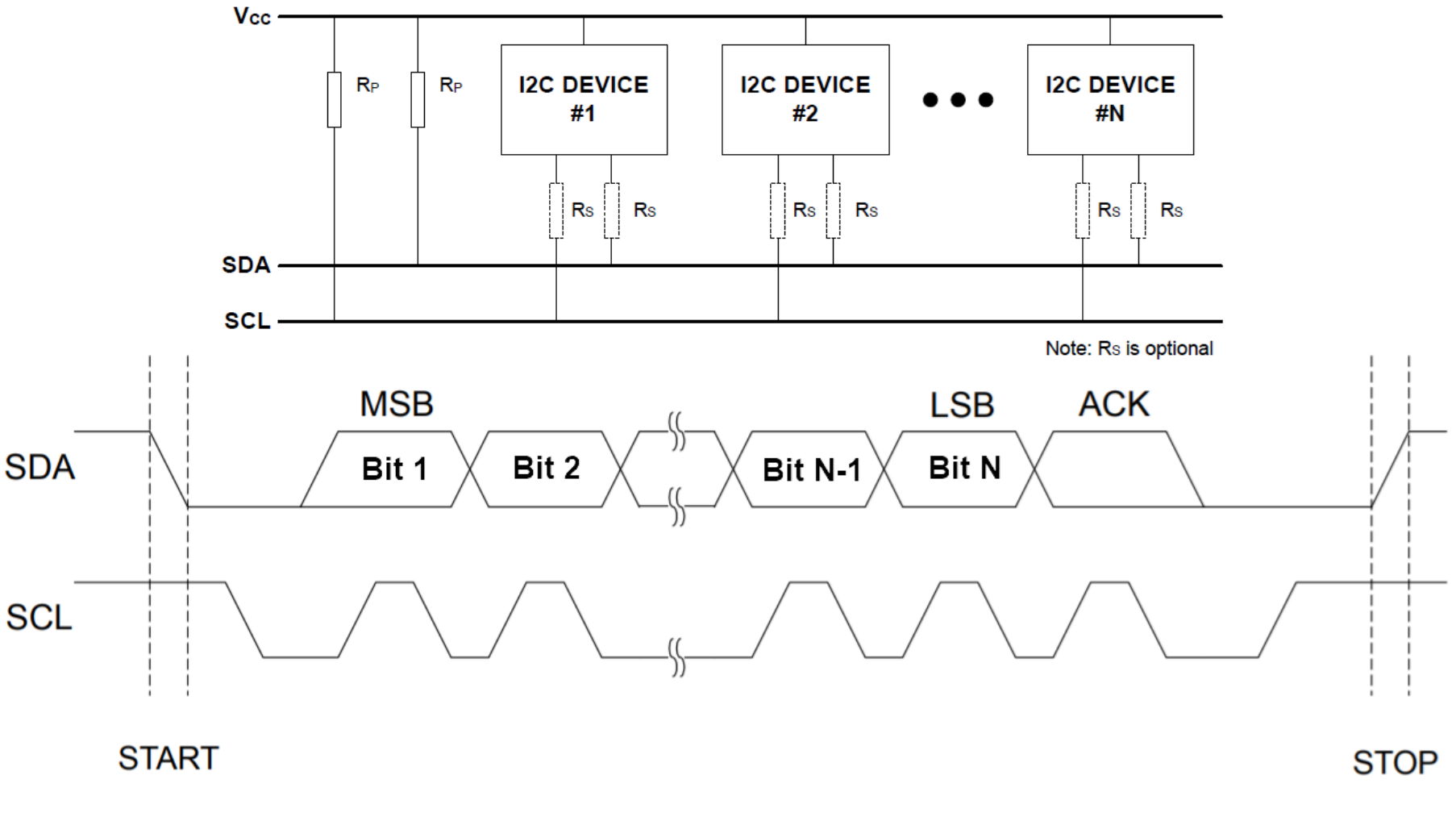
APP045 v4 Block Diagram



SERCOM - I2C Architecture

What's I2C ?

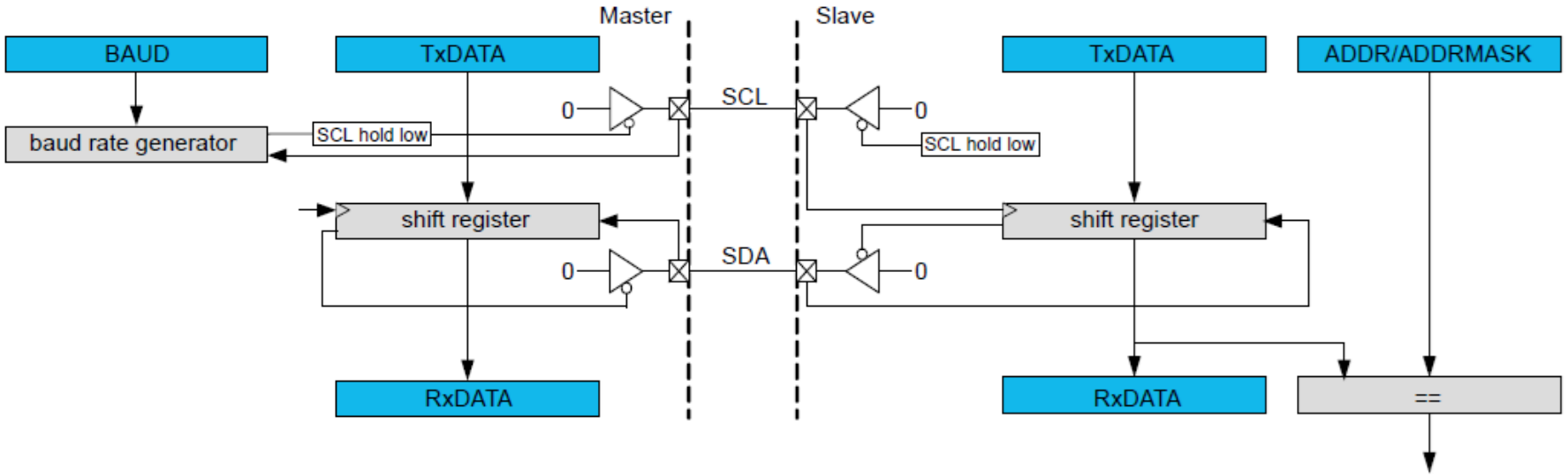
- **The I2C (Inter-Integrated Circuit) is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented in 1982 by Philips Semiconductor.**



SERCOM - I2C

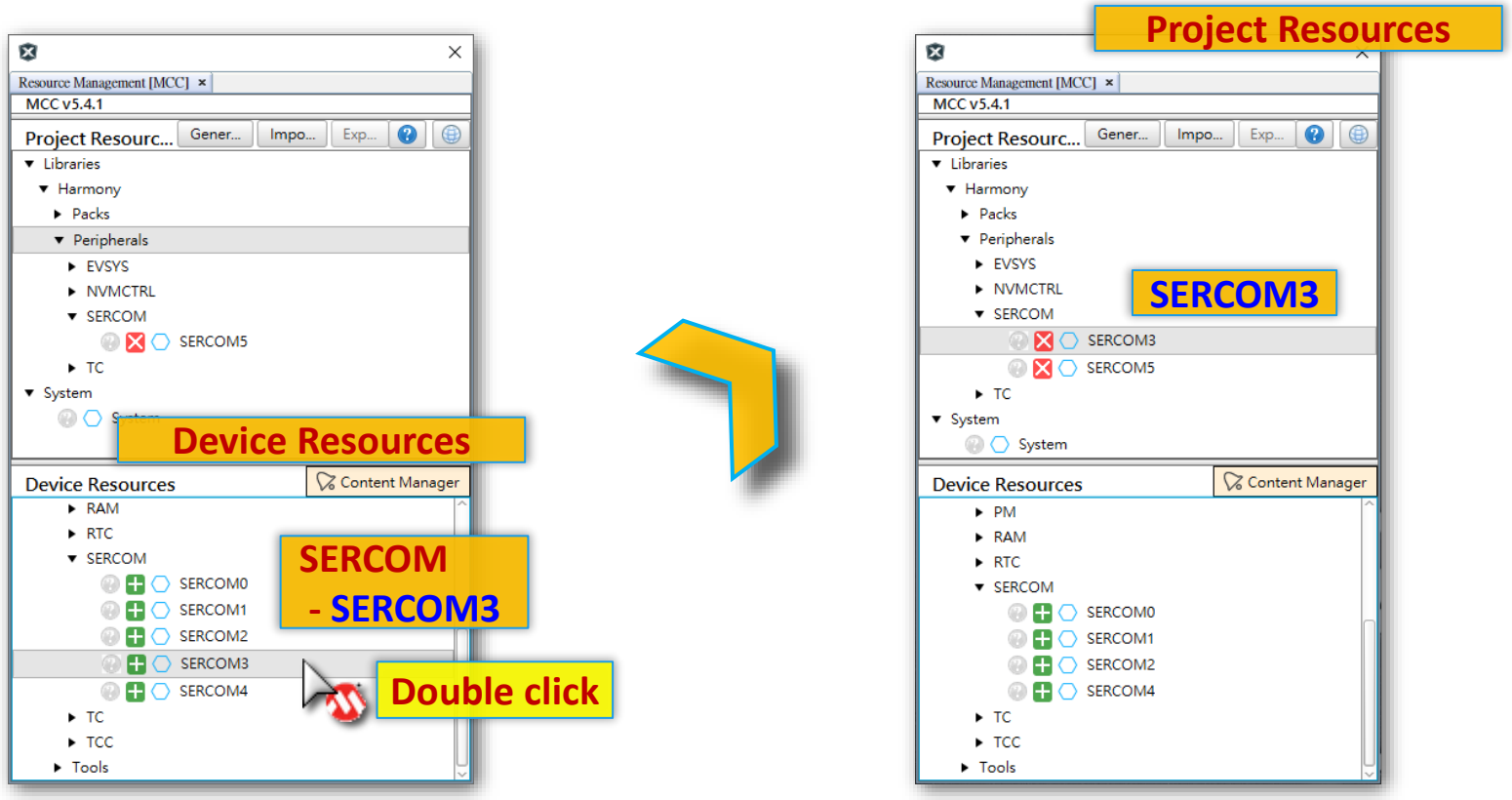
The I2C interface is one of the available modes in the serial communication interface (SERCOM).

- **Master or slave operation can be used with DMA.**
- **Philips I2C / SMBus™/ PMBus compatible.**
- **Support of 100kHz and 400kHz, 1MHz and 3.4MHz mode.**
- **Up to 16-bytes internal FIFO.**
- **Operation in all sleep modes.**
- **Wake-up on address match (Slave mode)**



Add SERCOM Function using MHC

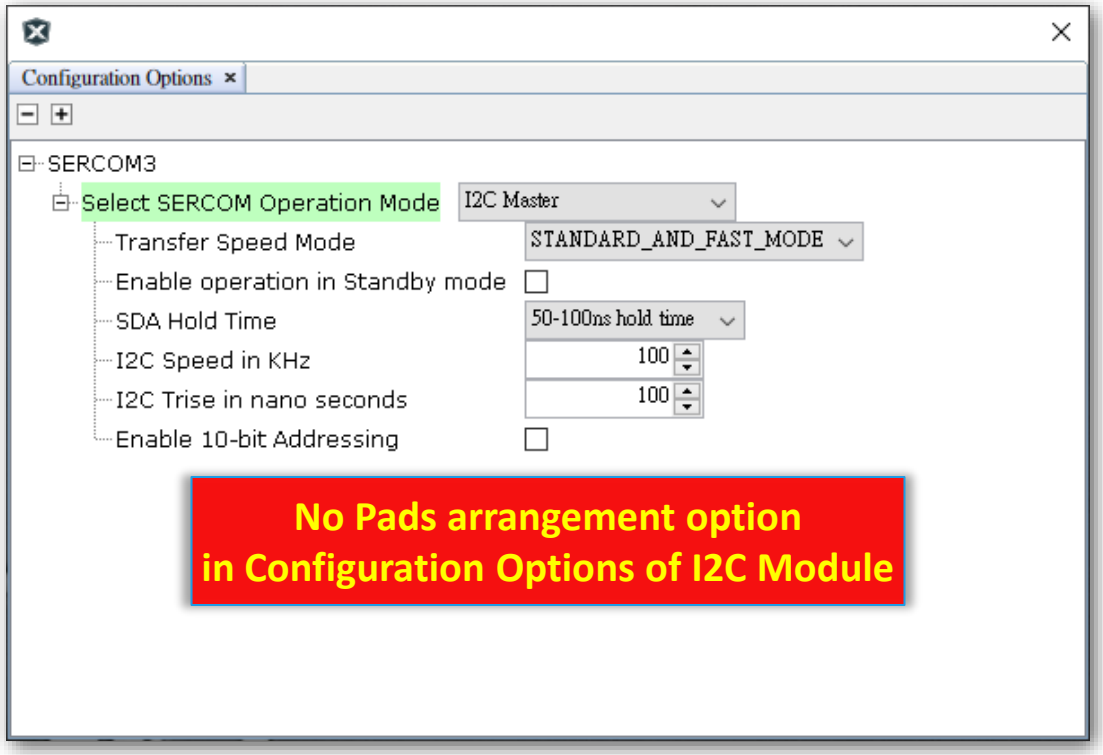
- Find SERCOM component in Device Resources window.
- Double click SERCOM3 to add to Project Resources window.



SERCOM-I2C

PINMUX Configuration Example

- **No need to arrange SERCOM pads in SERCOM-I2C module.**
- **SERCOM-I2C module use dedicate pads, PAD0 for SDA and PAD1 for SCL.**



SERCOM-I2C

PLIB SERCOM-I2C pad configurations

	PAD[0]	PAD[1]	PAD[2]	PAD[3]	PMUX
SERCOM0	PA04	PA05	PA06	PA07	D
	PA08	PA09	PA10	PA11	C
SERCOM1	PA00	PA01	PA30	PA31	D
	PA16	PA17	PA18	PA19	C
SERCOM2	PA08	PA09	PA10	PA11	D
	PA12	PA13	PA14	PA15	C
SERCOM3	PA16	PA17	PA18	PA19	D
			PA20	PA21	D
	PA22	PA23	PA24	PA25	C
SERCOM4	PA12	PA13	PA14	PA15	D
	PB08	PB09	PB10	PB11	D
SERCOM5			PA20	PA21	C
	PA22	PA23	PA24	PA25	D
	PB02	PB03	PB22	PB23	D

Signal Name	Type	Description
PAD[0]	Digital I/O	SDA
PAD[1]	Digital I/O	SCL
PAD[2]	Digital I/O	SDA_OUT (4-wire operation)
PAD[3]	Digital I/O	SCL_OUT (4-wire operation)

SERCOM-I2C

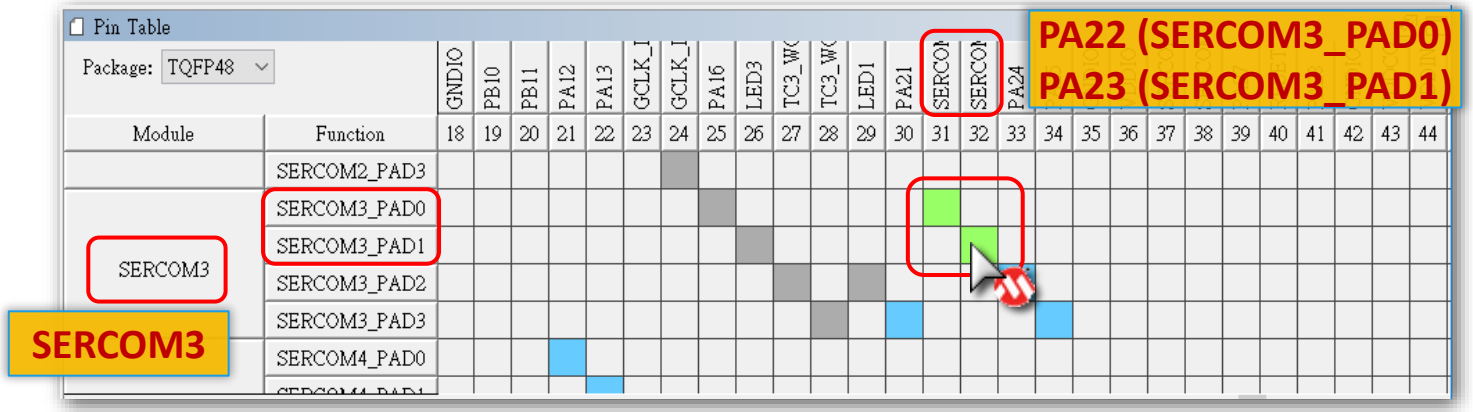
PINMUX Configuration Example

- SERCOM allow limited alternate pad index assignment.
- For example : PA22 as SERCOM3_PAD0 (SDA)
  PA23 as SERCOM3_PAD1 (SCL)

Pin(1)			I/O Pin	Supply	A	B(2)(3)					C	D	E	F	G	H
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC	SERCOM(2)(3)	SERCOM-ALT	TC(4) /TCC	TCC	COM	AC/ GCLK
21	31		PA22	VDDIO	EXTINT[6]				X[10]		SERCOM3/ PAD[0]					GCLK_IO[6]
22	32		PA23	VDDIO	EXTINT[7]				X[11]		SERCOM3/ PAD[1]	PAD[1]		WO[5]	USB/SOF 1kHz	GCLK_IO[7]

PA22
PA23

SERCOM3/PAD[0]
SERCOM3/PAD[1]



SERCOM-I2C

Interface Routines

```
void SERCOM3_I2C_Initialize( void );
bool SERCOM3_I2C_Read(uint16_t address, uint8_t *pdata, uint32_t length);
bool SERCOM3_I2C_Write(uint16_t address, uint8_t *pdata, uint32_t length);
bool SERCOM3_I2C_WriteRead(uint16_t address, uint8_t *wdata,
                                                    uint32_t wlength, uint8_t *rdata, uint32_t rlength);
bool SERCOM3_I2C_IsBusy(void);
SERCOM_I2C_ERROR SERCOM3_I2C_ErrorGet(void);
void SERCOM3_I2C_CallbackRegister(SERCOM_I2C_CALLBACK callback, uintptr_t context);
```

SERCOM-I2C Code Example

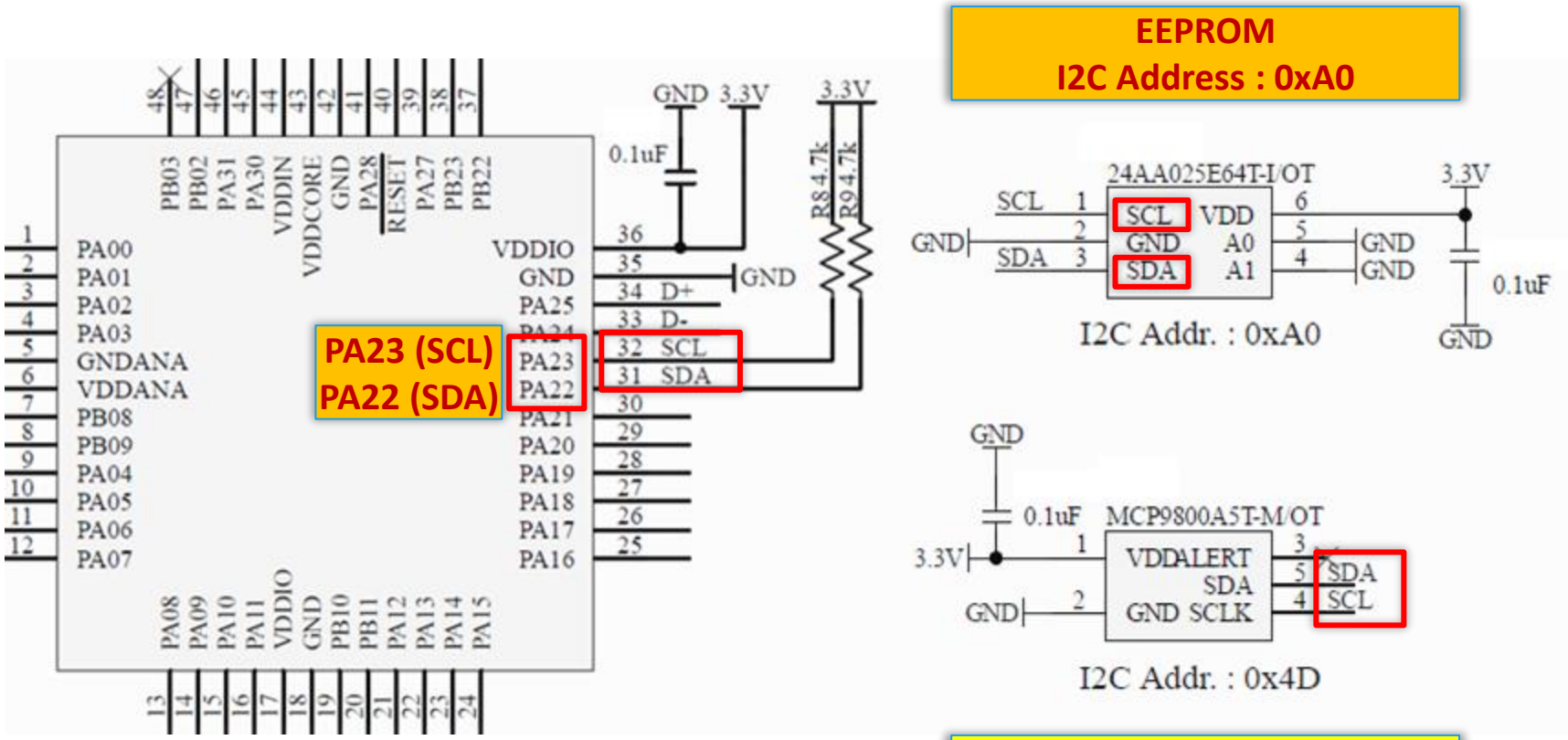
```
 …

int main (void)
{
   SYS_Initialize ( NULL );
   …
  SERCOM3_I2C_WriteRead( Slave_Address, WriteBytes, 1, ReadBytes, 2 );
   while( true )
   {
      …
      SERCOM3_I2C_Read( Slave_Address, ReadBytes, 2 );
   }
}
```

🛠 Lab11-1 SERCOM - I2C Access Temperature Sensor

- **Please continue from SAM2001 Lab11 (SERCOM - UART TxRx Interrupt Callback with myprintf)**
- **Add SERCOM3-I2C master function to project to communication the on-board Digital Temperature Sensor MCP9800A5T.**
- **Enable SERCOM3_PAD0(PA22) and SERCOM_3_PAD1(PA23) for I2C SDA and SCL.**
- **Send Ambient Temperature Register (TA) command 0x00 to acquire temperature data from Digital Temperature Sensor MCP9800A and then continue to read data.**
- **Output temperature data to UART.**
- **MCP9800 configuration:**
  - **SDA** connect to PA22 (SERCOM3 PAD0)
  - **SCL** connect to PA23 (SERCOM3 PAD1)
  - Slave address is 0x4D.
  - **TA** register address is 0x00.

- **Let's go !**

APP045 SERCOM-I2C Schematic



MCP9800A access command and Temperature calculation

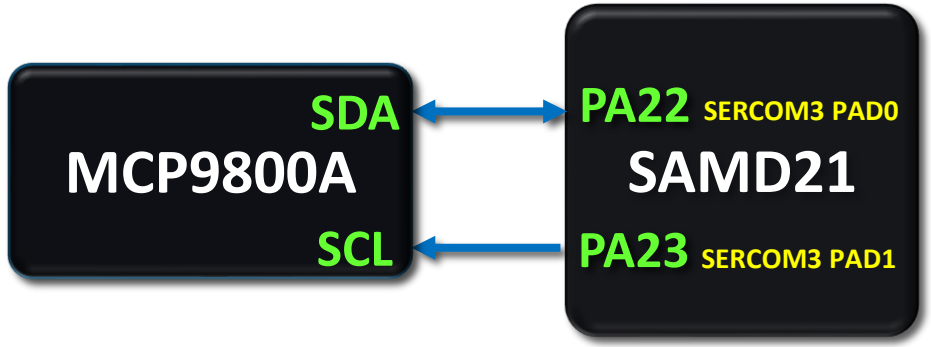
$$T_A = \text{Code} \times 2^{-4}$$

T_A = Ambient Temperature (°C)

Code = MCP9800 output in decimal

```
Result = (TA[0]<<4|TA[1]>>4)/16
```

Register Pointer P1 P0	MSB/ LSB	Bit Assignment								
		7	6	5	4	3	2	1	0	
Ambient Temperature Register (T_A)										
0 0	MSB	Sign	2^6°C	2^5°C	2^4°C	2^3°C	2^2°C	2^1°C	2^0°C	TA[0]
	LSB	2^{-1}°C	2^{-2}°C	2^{-3}°C	2^{-4}°C	0	0	0	0	TA[1]

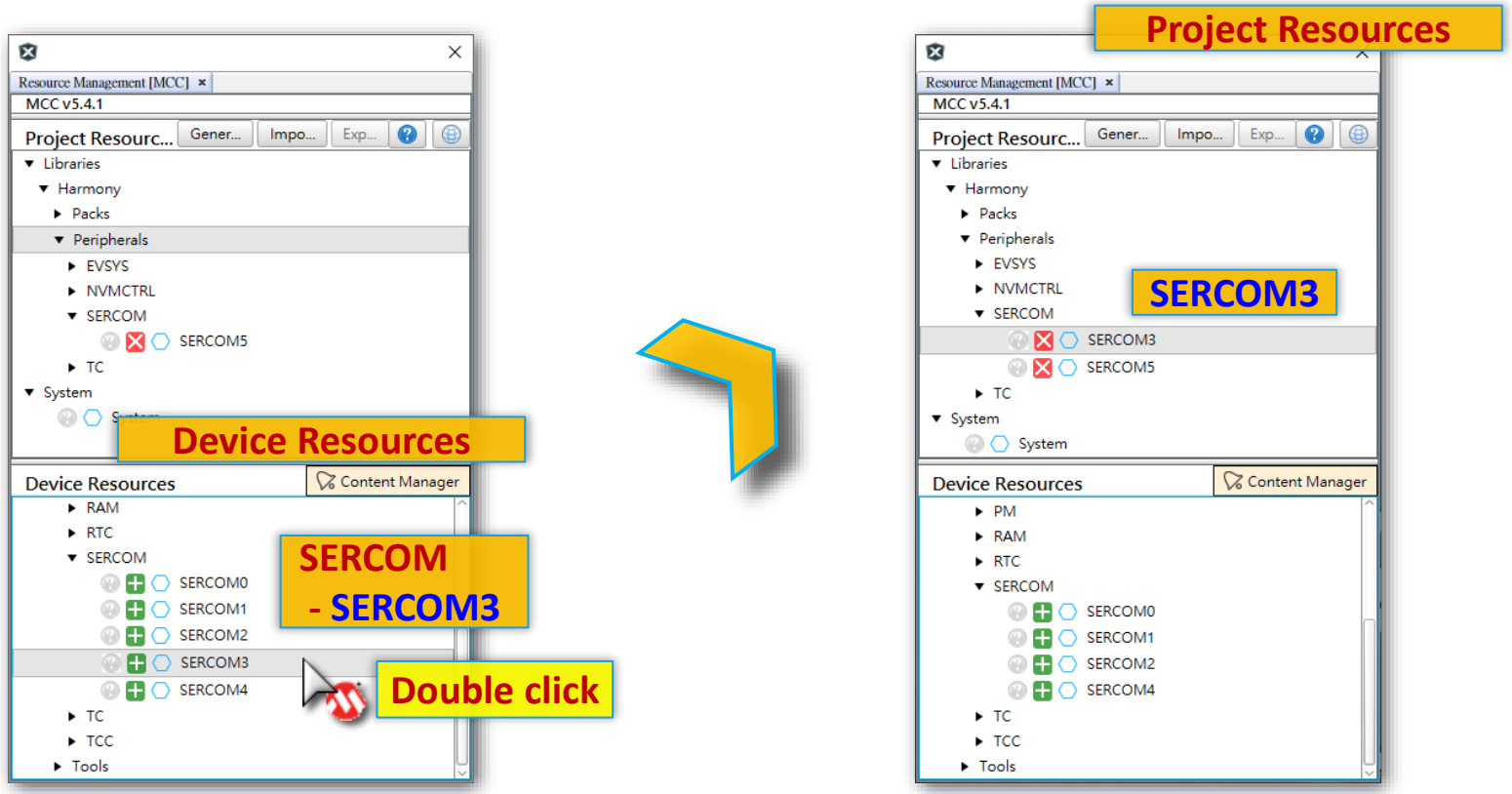


MICROCHIP

🛠 Lab11-1 SERCOM – I2C Access Temperature Sensor

Step 1

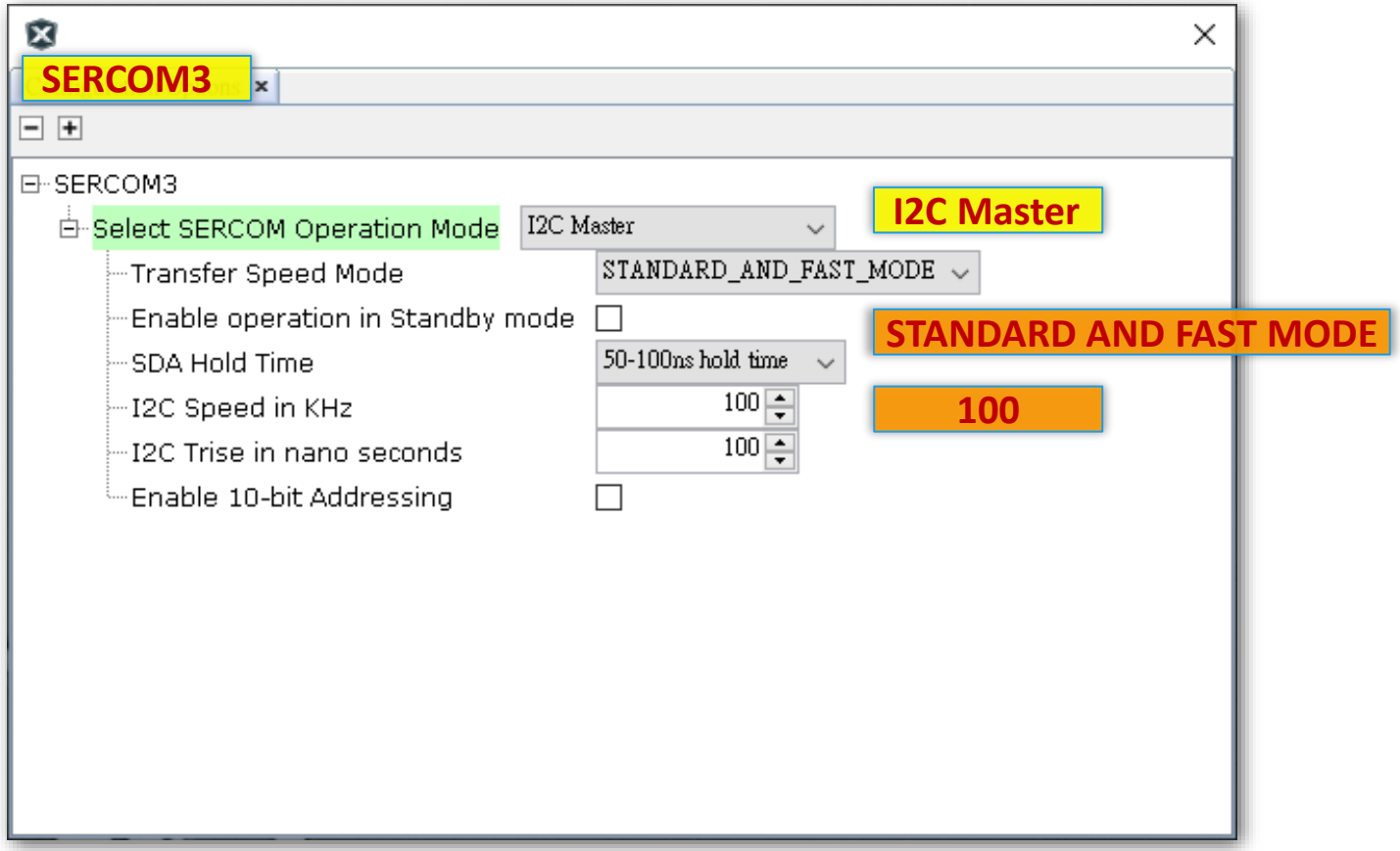
- Find **SERCOM** component in Device Resources window.
- Double click **SERCOM3** to add to Project Resources window.



🛠 Lab11-1 SERCOM – I2C Access Temperature Sensor

Step 2

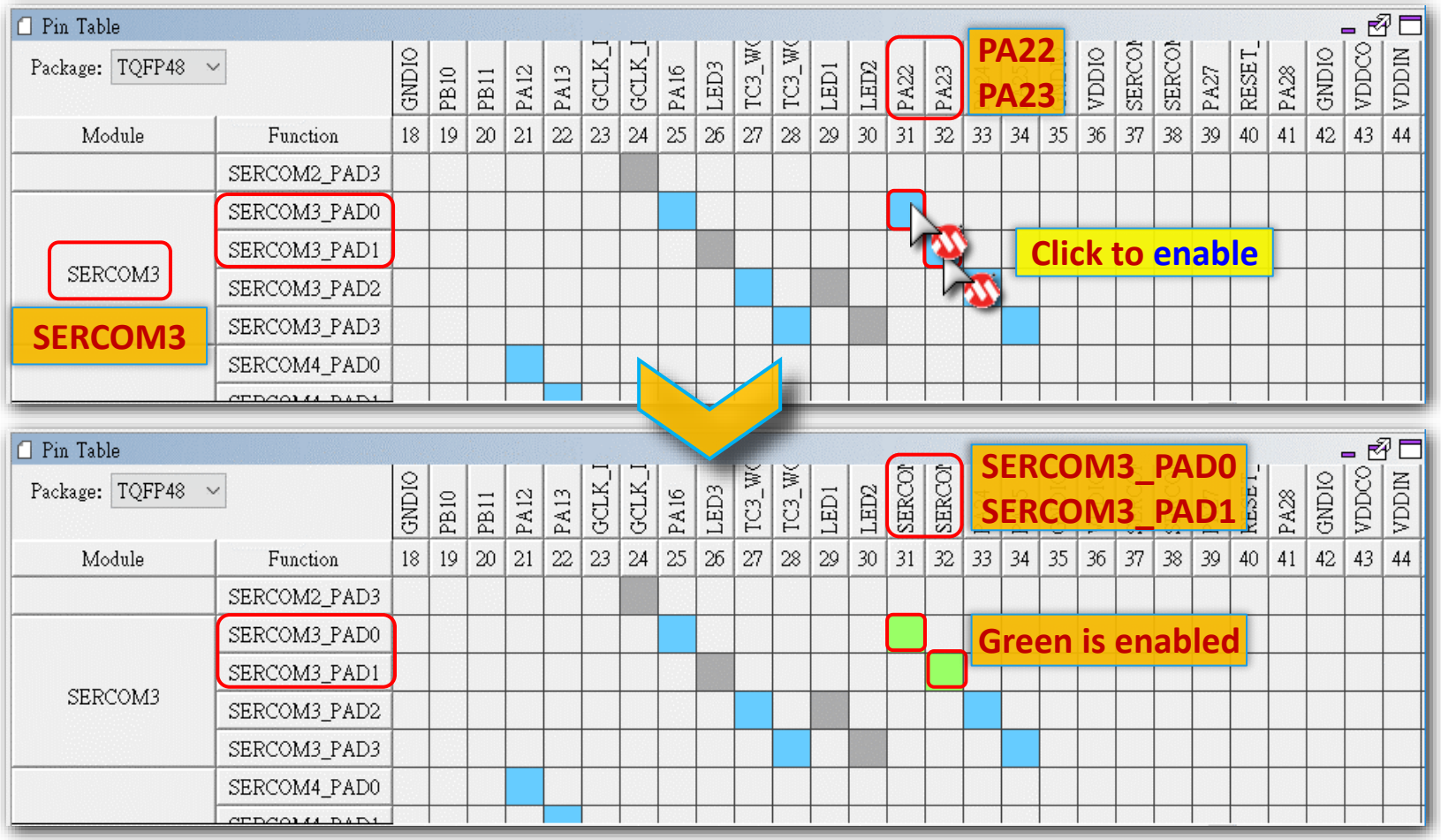
- **Configure SERCOM3 to I2C Master mode.**



🛠 Lab11-1 SERCOM – I2C Access Temperature Sensor

Step 3

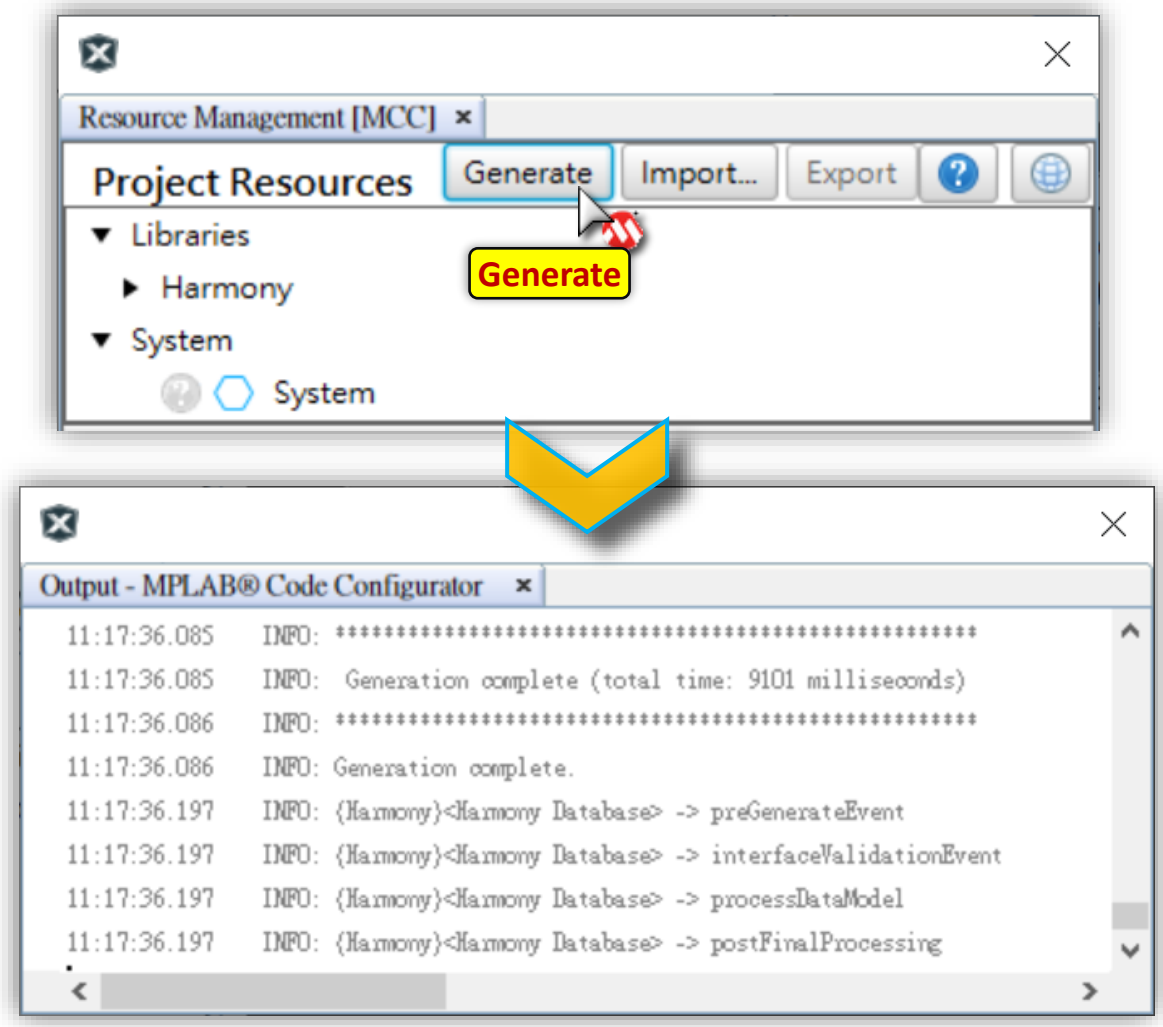
- Reassign PA22 to SERCOM3_PAD0 and PA23 to SERCOM3_PAD1.



🛠 Lab11-1 SERCOM – I2C Access Temperature Sensor

Step 4

- Click "Generate" Button to Generate your Code.



MICROCHIP

🛠 Lab11-1 SERCOM – I2C Access Temperature Sensor

Step 5

```
// TODO 11-1.01
#define MCP9800_ADDRESS        0x4D
#define MCP9800_REG_TA         0x0
uint8_t MCP9800_Data[2];

void USART5_Transmit( uintptr_t context ) { ... }
...
int main (void)
{  …
    // TODO 11-1.02
    MCP9800_Data[0] = MCP9800_REG_TA;
    SERCOM3_I2C_WriteRead( MCP9800_ADDRESS, MCP9800_Data, 1, MCP9800_Data, 2 );

    while ( true )
    {   …
        if (TC3_HasExpired)
        {   …
            // TODO 11-1.03
            SERCOM3_I2C_Read( MCP9800_ADDRESS, MCP9800_Data, 2 );
            myprintf( "Temperature : %2.1f'C\r\n",
                            (float)((MCP9800_Data[0]<<4)|(MCP9800_Data[1]>>4))/16.0f);
        }
    }
```

$$T_A = \text{Code} \times 2^{-4}$$

T_A = Ambient Temperature (°C)

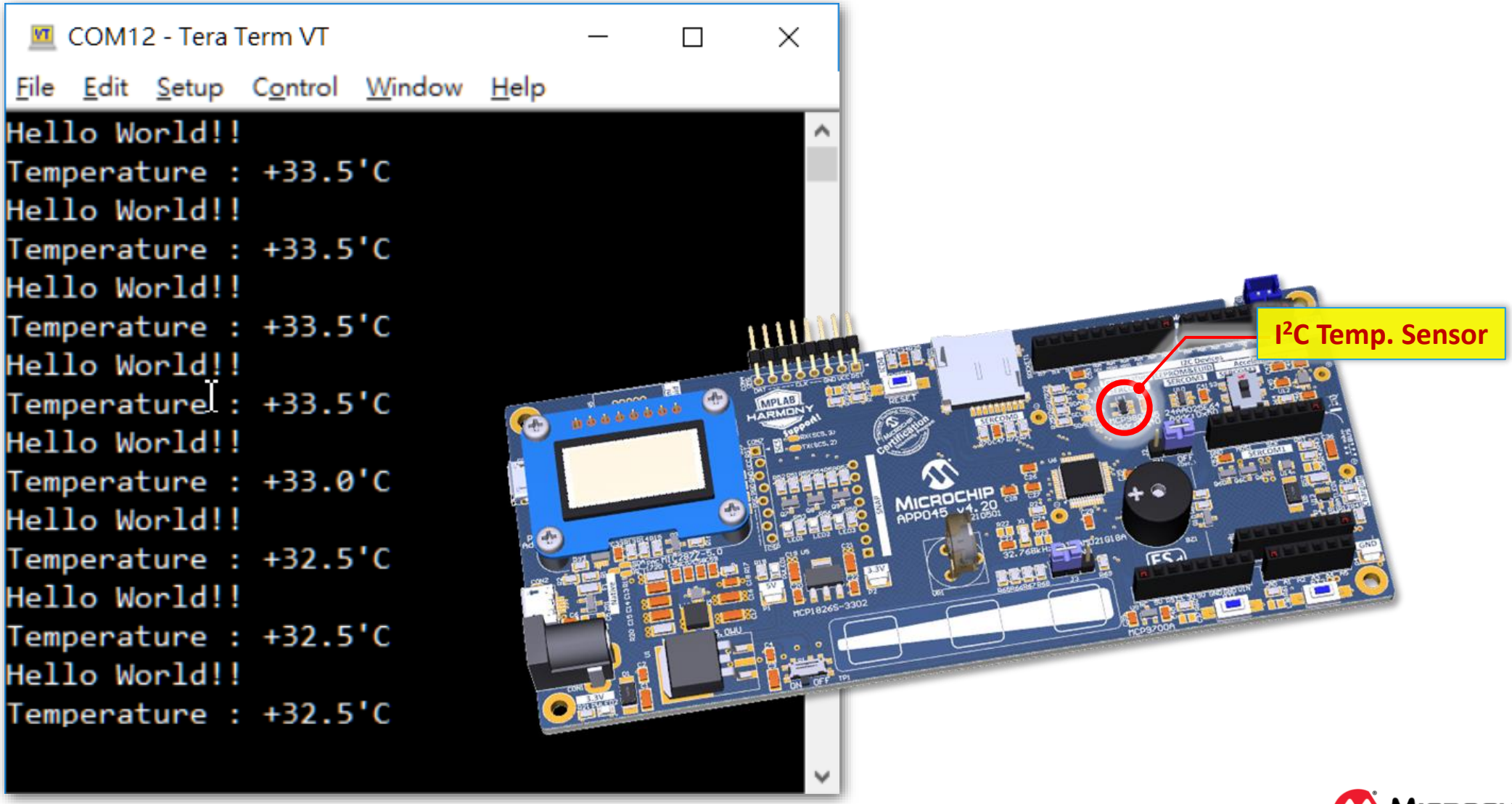
Code = MCP9800 output in decimal

```
Result = (TA[0]<<4|TA[1]>>4)/16
```

MICROCHIP

Lab11-1 SERCOM – I2C Access Temperature Sensor

Result

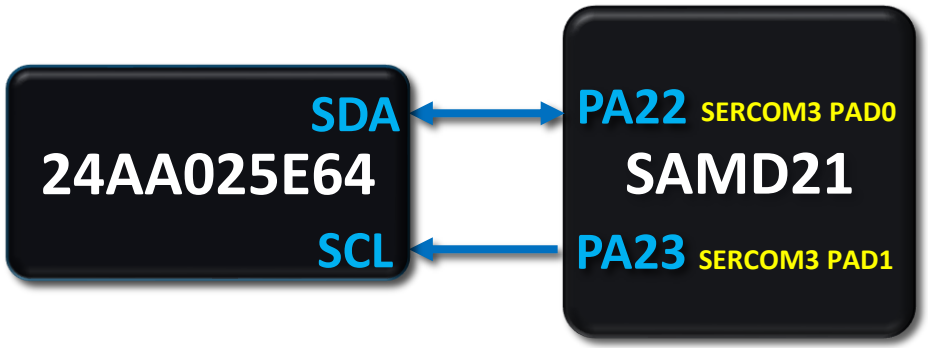


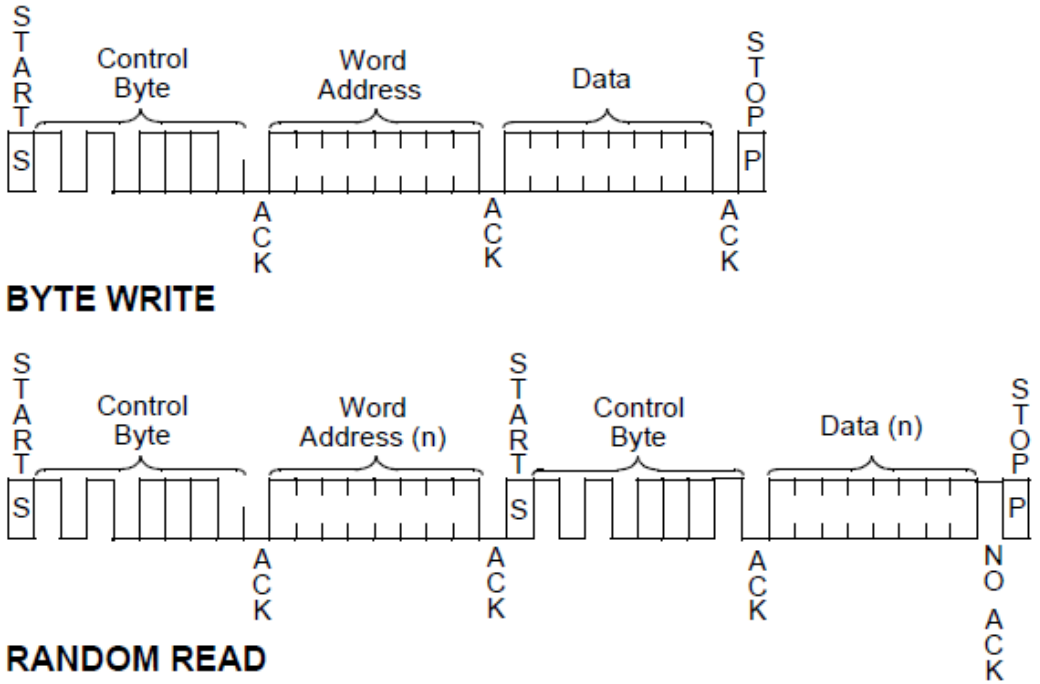
🛠 Lab11-2 SERCOM - I2C Access Serial EEPROM

- **Use SERCOM3-I2C master function to project to communication the on-board Serial EEPROM 24AA025E64.**
- **Write one byte "0x24" to target word address 0x00 after system initialize.**
- **Random Read it back and output to UART console.**
- **24AA025E64 configuration:**
  - **SDA** connect to PA22 (SERCOM3 PAD0)
  - **SCL** connect to PA23 (SERCOM3 PAD1)
  - Slave address is 0x50.

- **Let's go !**

24AA025E64 access command and connection





🛠 Lab11-2 SERCOM - I2C Access Serial EEPROM

Step 1

```
// TODO 11-2.01
#define MCP24AA_ADDRESS 0x50
uint8_t MCP24AA_WriteData[2];
uint8_t MCP24AA_ReadData[1];
...
int main (void)
{  …
    // TODO 11-2.04
    while (SERCOM3_I2C_IsBusy());
    // TODO 11-2.02
    MCP24AA_WriteData[0] = 0x00;
    MCP24AA_WriteData[1] = 0x24;
    SERCOM3_I2C_Write( MCP24AA_ADDRESS, MCP24AA_WriteData, 2 );
    myprintf( "Wrote 0x24 to EEPROM 24AA\r\n");

    while ( true )
    {   …
        if (TC3_HasExpired)
        {   …
            // TODO 11-2.03
            MCP24AA_WriteData[0] = 0x00;
            SERCOM3_I2C_WriteRead( MCP24AA_ADDRESS, MCP24AA_WriteData, 1,
                                                    MCP24AA_ReadData, 1 );
            while( SERCOM3_I2C_IsBusy());
            myprintf( "EEPROM Read : 0x%02X\r\n", MCP24AA_ReadData[0] );
        }
    }
```

MICROCHIP

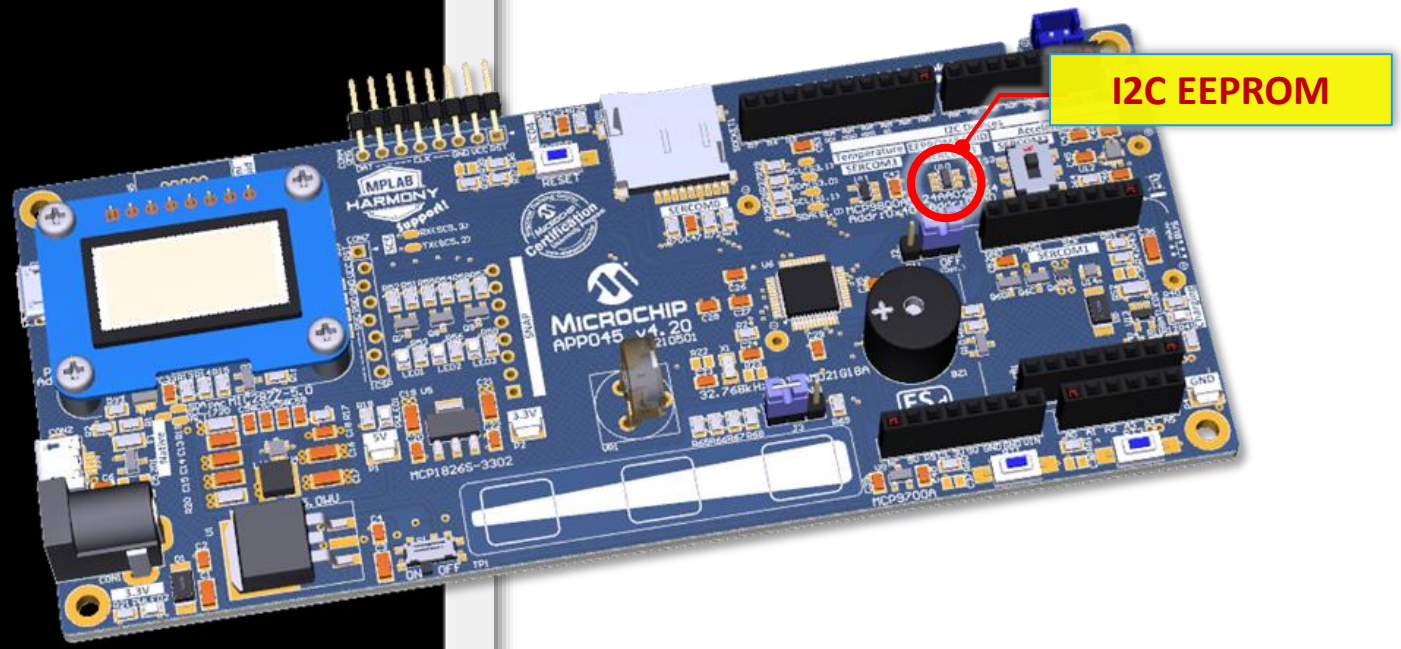
Lab11-2 SERCOM - I2C Access Serial EEPROM

Result

COM12 - Tera Term VT

File Edit Setup Control Window Help

```
Wrote 0x24 to EEPROM 24AA
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
```

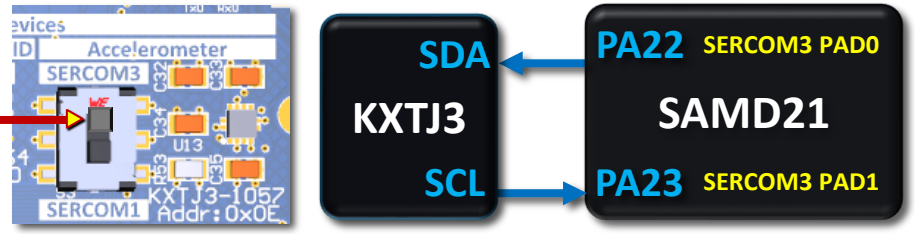


🛠 Lab11-3 SERCOM - I2C Access G-Sensor

- **Use SERCOM3-I2C master function to project to communication the on-board Accelerometer KXTJ3.**
- **KXTJ3 initialize:**
  - Write Software Reset "0x80" to Control Register 2 "0x1D" and wait 10ms for stable.
  - Write follow Read the WHO AM I Register "0x0F" to check the ID is equal to "0x35" or not.
  - Write Initial Command "0xF0" to Control Register 1 "0x1B" for op-mode and resolution.
- **KXTJ3 Interrupt Source Register read by Callback:**
  - Write follow Read the Interrupt Source Register "0x16".
- **KXTJ3 Data Ready Interrupt check in Callback function:**
  - Check "Bit 4 DRDY INT" of Interrupt Source Register to make sure data prepared for read.
- **KXTJ3 Axis Data Read:**
  - Write follow Read the Axis Output Register in below sequence to get X, Y, Z data.

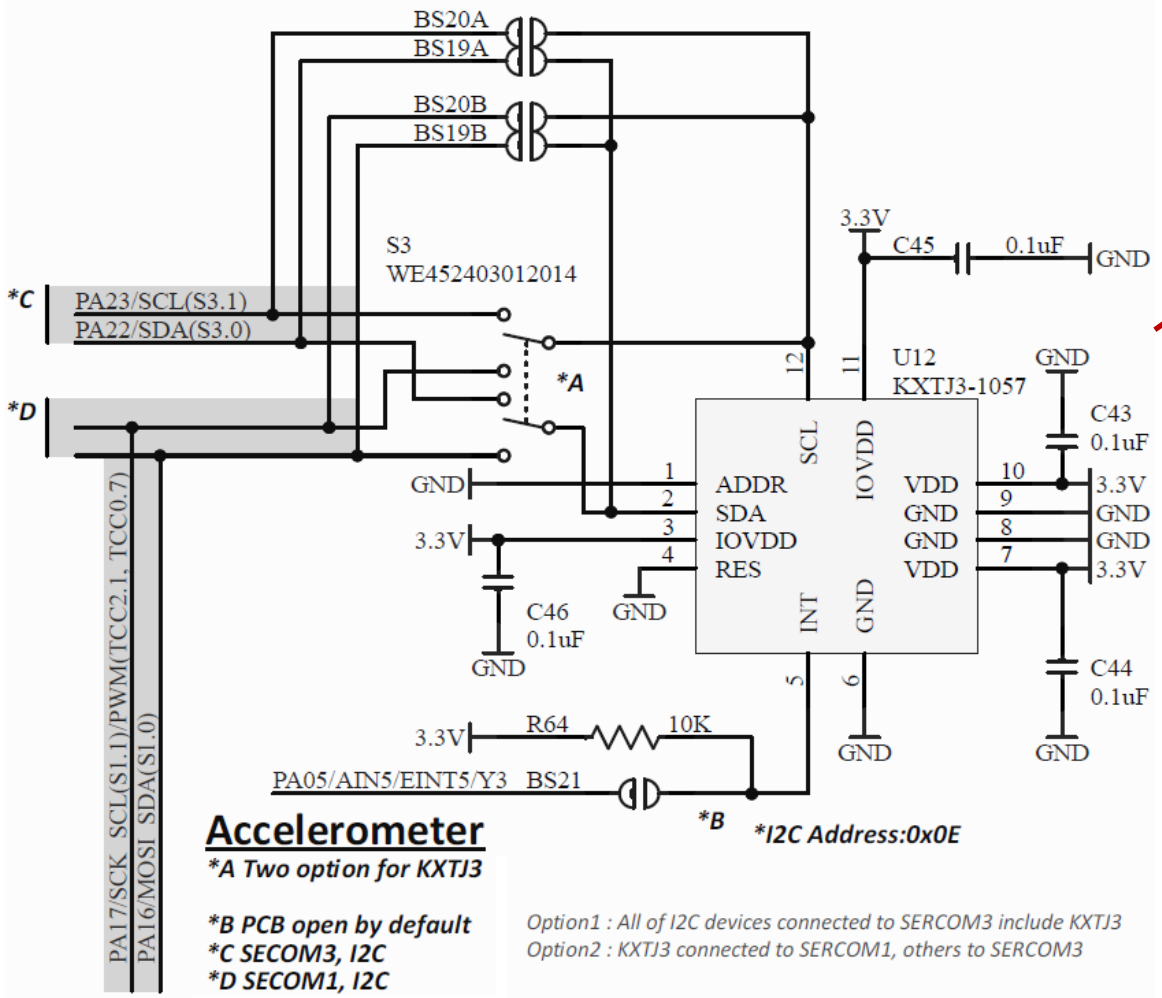
XOUT_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

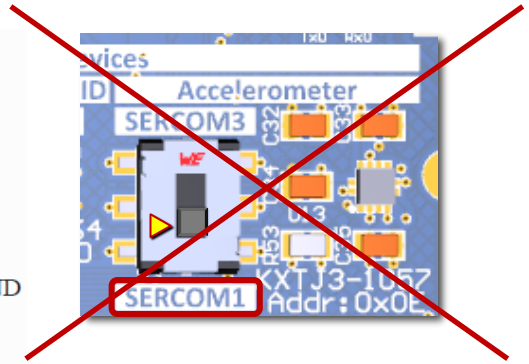
- **KXTJ3 Axis Data Output:**
  - Output X, Y, Z Axis data as format "`G-Sensor = (+XXXX, +YYYY, +ZZZZ)`"
- **KXTJ3 configuration:**
  - **Switch the S3 to SERCOM3 on EVB**
  - **SDA** connect to **PA22** (SERCOM3 PAD0)
  - **SCL** connect to **PA23** (SERCOM3 PAD1)
  - Slave address is **0x0E**. (ADDR pin tie to GND)

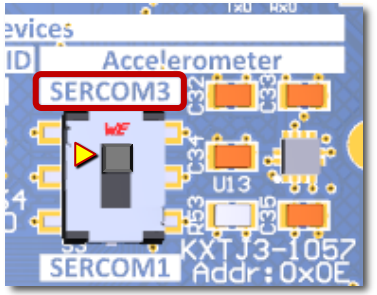


MICROCHIP

KXTJ3 could connect to SERCOM3 or SERCOM1







Switch the S3 to SERCOM3 on EVB

MICROCHIP

🛠 Lab11-3 SERCOM – I2C Access G-Sensor

Step 1

XOUT_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

```
// TODO 11-3.01
#define KXTJ3_ADDRESS   0x0E
uint8_t KXTJ3_WriteData[2];
uint8_t KXTJ3_ReadData[1];
volatile uint8_t I2C3_IsDataReady = 0;
uint8_t AxisOutReg[6] = { 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B };
uint8_t AxisOutByte[6];

void USART5_Transmit( uintptr_t context ) { ... }
...
// TODO 11-3.02
void I2C3_TransferCallback( uintptr_t contextHandle )
{
    if( KXTJ3_ReadData[0] & (1<<4) )
    {
        I2C3_IsDataReady = 1;
    }
}

void myprintf(const char *format, ...) { ... }
...
int main (void)
{  …
    while ( true )
    {   …
```

MICROCHIP

Lab11-3 SERCOM – I2C Access G-Sensor

Step 2

```
int main (void)
{  …
    // TODO 11-3.03
    KXTJ3_WriteData[0] = 0x1D;
    KXTJ3_WriteData[1] = 0x80;
    SERCOM3_I2C_Write( KXTJ3_ADDRESS, KXTJ3_WriteData, 2 );
    myprintf( "G-Sensor Software Reset Completed\r\n" );
    KXTJ3_WriteData[0] = 0x0F;
    SERCOM3_I2C_WriteRead( KXTJ3_ADDRESS, KXTJ3_WriteData, 1, KXTJ3_ReadData, 1 );
    while( SERCOM3_I2C_IsBusy() );
    myprintf( "G-Sensor WHO AM I : 0x%X\r\n", KXTJ3_ReadData[0]);
    KXTJ3_WriteData[0] = 0x1B;
    KXTJ3_WriteData[1] = 0xF0;
    SERCOM3_I2C_Write( KXTJ3_ADDRESS, KXTJ3_WriteData, 2 );
    myprintf( "G-Sensor Initial Completed\r\n" );
    SERCOM3_I2C_CallbackRegister( I2C3_TransferCallback, 0 );

    while ( true )
    {   …
        if (TC3_HasExpired)
        {   …
            // TODO 11-3.04
            KXTJ3_WriteData[0] = 0x16;
            SERCOM3_I2C_WriteRead( KXTJ3_ADDRESS, KXTJ3_WriteData, 1, KXTJ3_ReadData, 1 );
        }
```

MICROCHIP

Lab11-3 SERCOM – I2C Access G-Sensor

Step 3

```
...
int main (void)
{  …
    while ( true )
    {   …
        if (TC3_HasExpired) { … }

        if (TC4_HasExpired) { … }

        // TODO 11-3.05
        if( I2C3_IsDataReady )
        {
            for( i=0 ; i<6 ; i++ )
            {
                SERCOM3_I2C_WriteRead( KXTJ3_ADDRESS, &AxisOutReg[i], 1,
                                                      &AxisOutByte[i], 1 );
                while( SERCOM3_I2C_IsBusy() );
            }
            myprintf("G-Sensor = (%5d, %5d, %5d)\r\n",
                              (short)(AxisOutByte[1]<<8|AxisOutByte[0]),
                              (short)(AxisOutByte[3]<<8|AxisOutByte[2]),
                              (short)(AxisOutByte[5]<<8|AxisOutByte[4]));
            I2C3_IsDataReady = 0;
        }
        …
```

XOUT_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

MICROCHIP

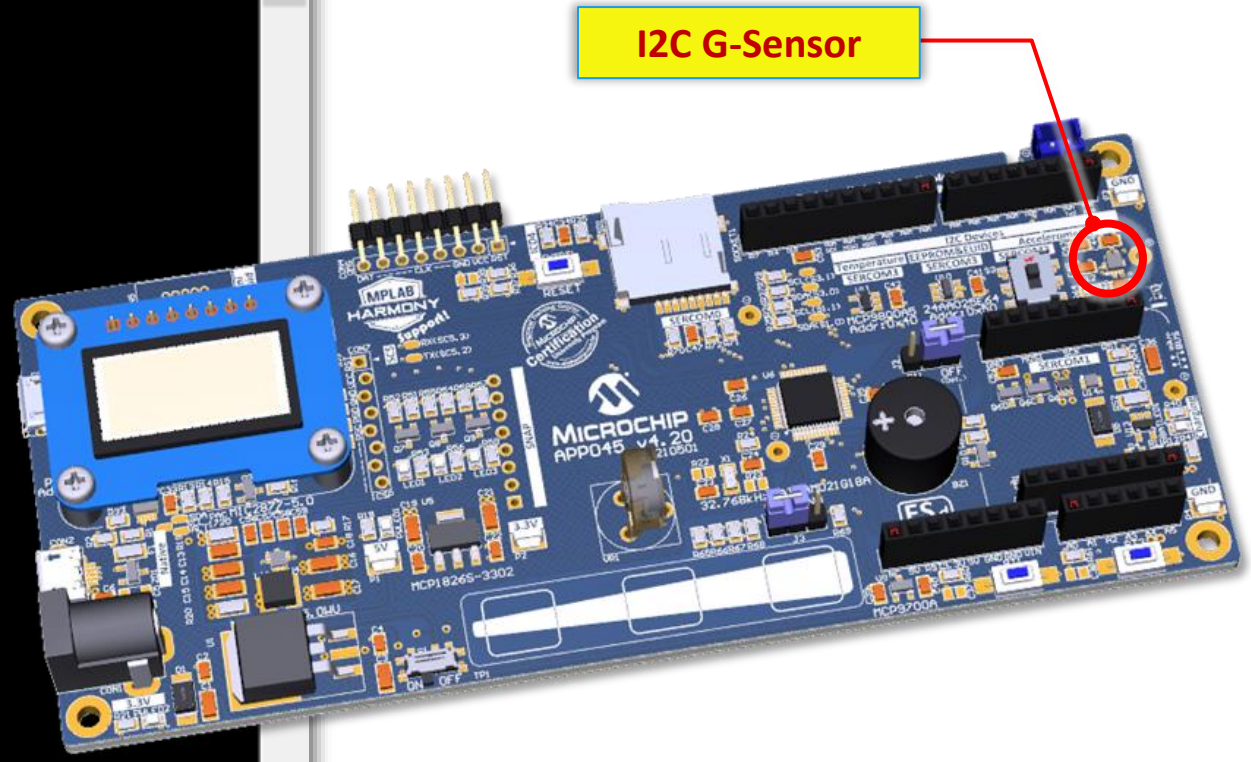
Lab11-3 SERCOM - I2C Access G-Sensor

Result

COM12 - Tera Term VT

File Edit Setup Control Window Help

```
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
G-Sensor = (    0,    -32,   4224)
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
G-Sensor = (    0,    -32,   4224)
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
G-Sensor = (    0,    -48,   4224)
Hello World!
Temperature : 27.5'C
EEPROM Read : 0x24
G-Sensor = (    0,    -32,   4224)
```



ADC Accumulation and Averaging

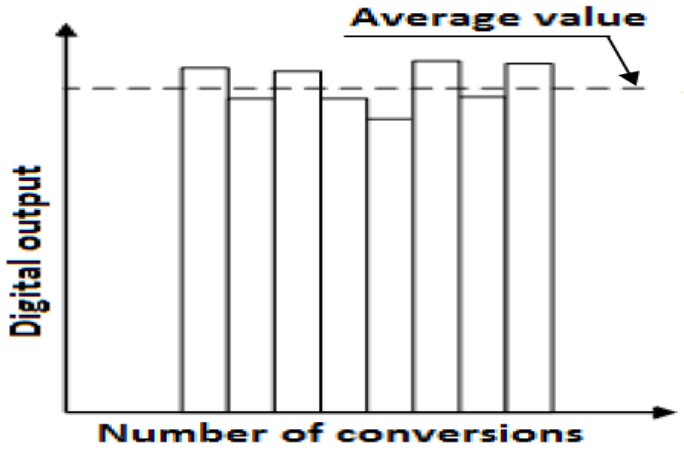
ADC Accumulation and Averaging

- The result from multiple consecutive of ADC conversions can be accumulated. When accumulating more than 16 samples, the result will be too large to match the 16-bit RESULT register size. To avoid overflow, the result is right shifted automatically to fit within the available register size.
- Averaging is a feature that increases stability at the cost of a reduced sampling rate. This feature is suitable when operating in noisy conditions. Averaging is done by accumulating m samples, and dividing the result by m.

$$\frac{\text{Sample 1} + \text{Sample 2} + \ldots + \text{Sample m}}{m}$$

Accumulation

Averaging



ADC Accumulation and Averaging

Example :

Number of Accumulated Samples : ∑ 128 (2^7 samples)
Intermediate Result precision : 19 bits (12bits + 7 = 19 bits overflow)
Number of Automatic Right Shift : 3 bits (19bits >> 3 = 16bits)
Adjust Division Factor : /16 (2^4)
Finial Result Precision : 12 bits (16bit >> 4 = 12 bits)

Number of Accumulated Samples	AVGCTRL. SAMPLENUM	Intermediate Result Precision	Number of Automatic Right Shifts	Automatic Division Factor	AVGCTRL. ADJRES	Adjust Division Factor	Total Number of Right Shifts	Final Result Precision
1	0x0	12 bits	0	0	0x0	1		12 bits
2	0x1	13 bits	0	0	0x1	2	1	12 bits
4	0x2	14 bits	0	0	0x2	4	2	12 bits
8	0x3	15 bits	0	0	0x3	8	3	12 bits
16	0x4	16 bits	(19>>3)	0	0x4	16	(3+4)	12 bits
32		17 bits		2		16		12 bits
64	2^7	18 bits	>>3	4	>>4	16	7	12 bits
128	0x7	19 bits	3	8	0x4	16	7	12 bits
∑ 128	0x8	19 bits	4	8	0x4	16	8	12 bits
	0x9		5		0x4		9	
(2^7)	0xA	(12+7)	6	(2^3)	0x4	(2^4)	10	(16>>4)
Reserved	0xB –0xF							

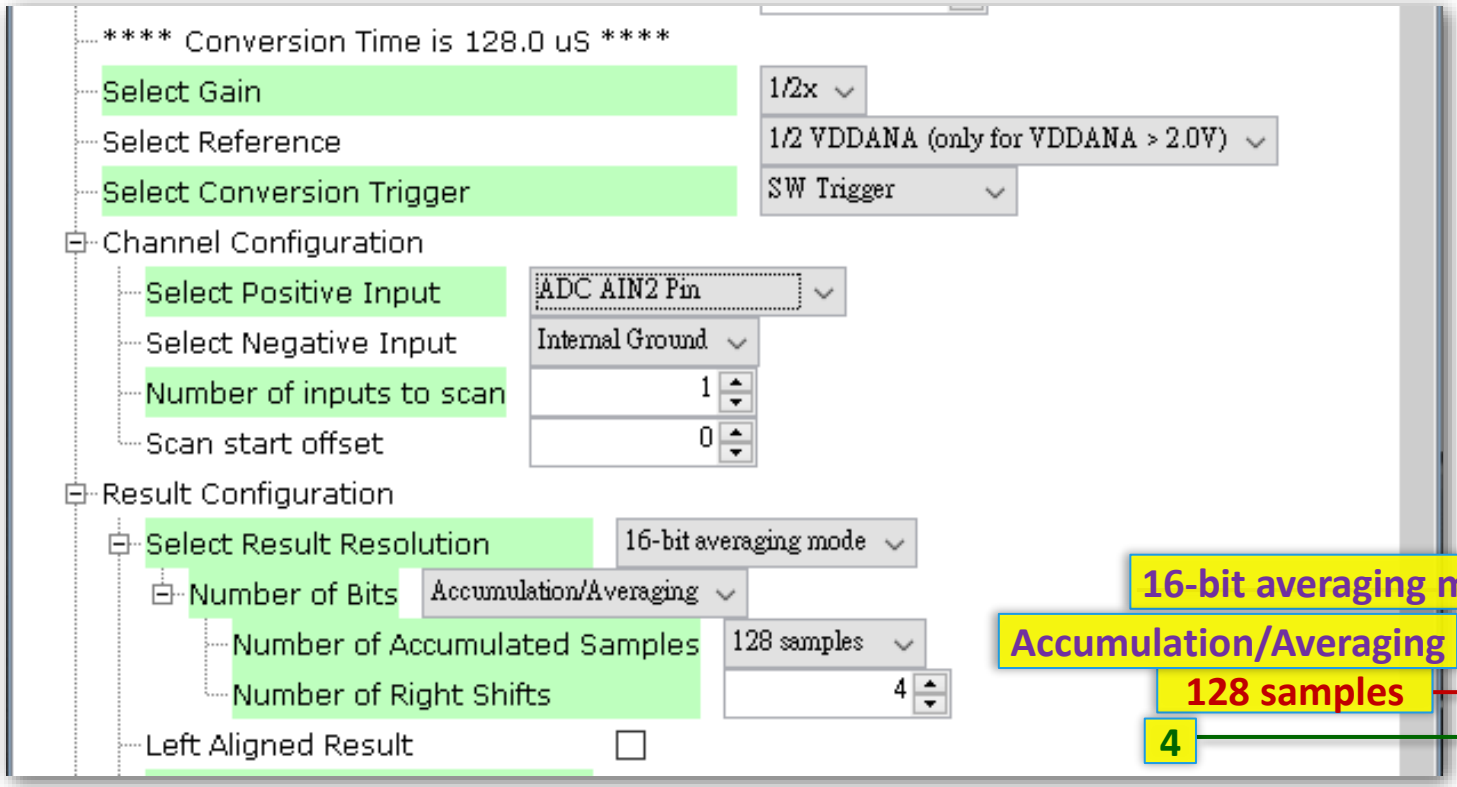
RESSEL[1:0]	Name	Description
0x0	12BIT	12-bit result
0x1	16BIT	For averaging mode output
0x2	10BIT	10-bit result
0x3	8BIT	8-bit result

MICROCHIP

ADC Accumulation and Averaging

Example :

Number of Accumulated Samples : ∑ 128 (2^7 samples)
Intermediate Result precision : 19 bits (12bits + 7 = 19 bits overflow)
Number of Automatic Right Shift : 3 bits (19bits >> 3 = 16bits)
Adjust Division Factor : /16 (2^4)
Finial Result Precision : 12 bits (16bit >> 4 = 12 bits)



RESSEL[1:0]	Name	Description
0x0	12BIT	12-bit result
0x1	16BIT	For averaging mode output
0x2	10BIT	10-bit result
0x3	8BIT	8-bit result

16-bit averaging mode
Accumulation/Averaging
128 samples
4

MICROCHIP

🛠 Lab13-1 ADC Accumulation and Averaging

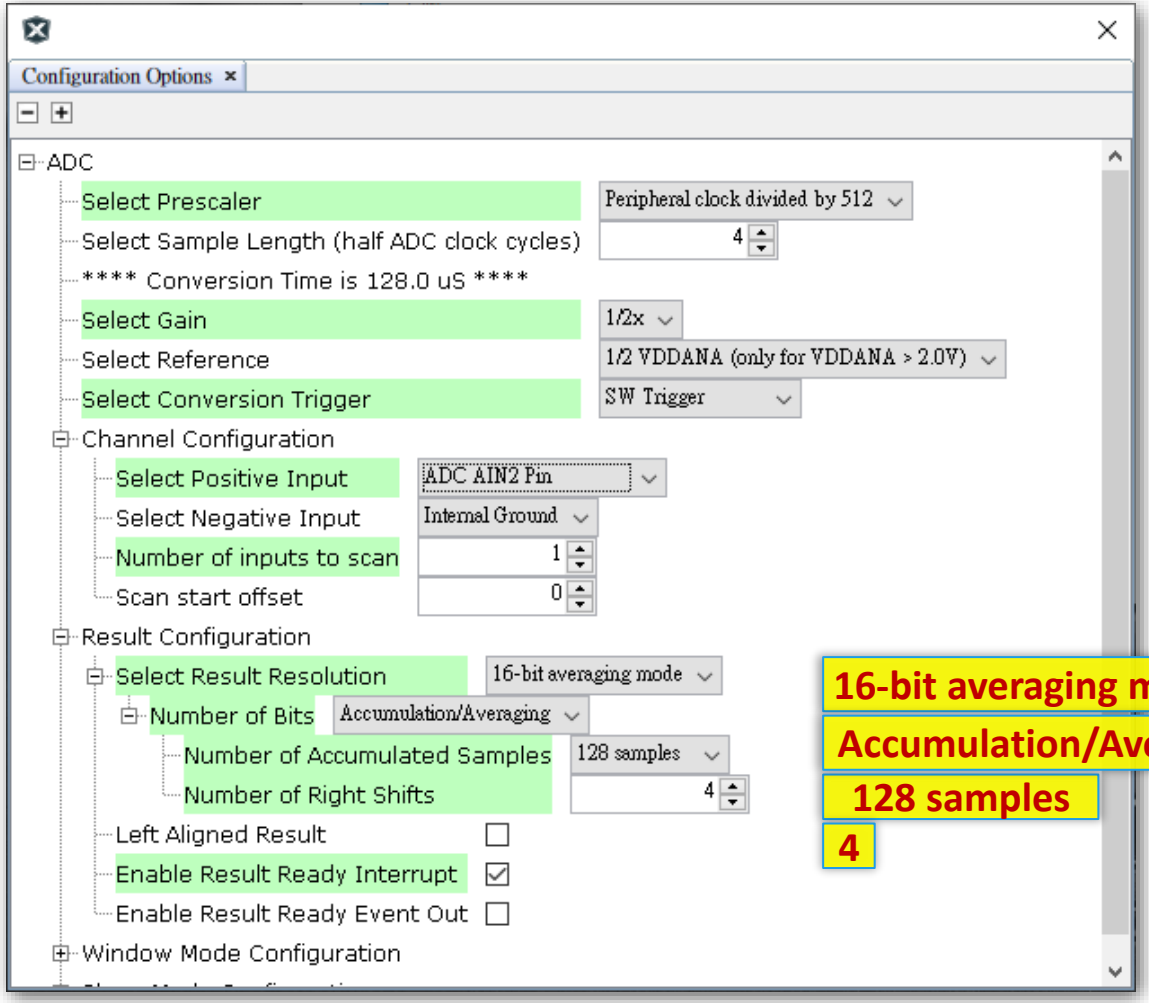
- Please continue from SAM2001 Lab13 (ADC Pin Scan)
- Accumulated 128 Samples and output 12 bits resolution.
- Modify ADC conversion interval to 10ms.
- Modify output message to keep update at the same line.

- **Let's go!**

🛠 Lab13-1 ADC Accumulation and Averaging

Step 1

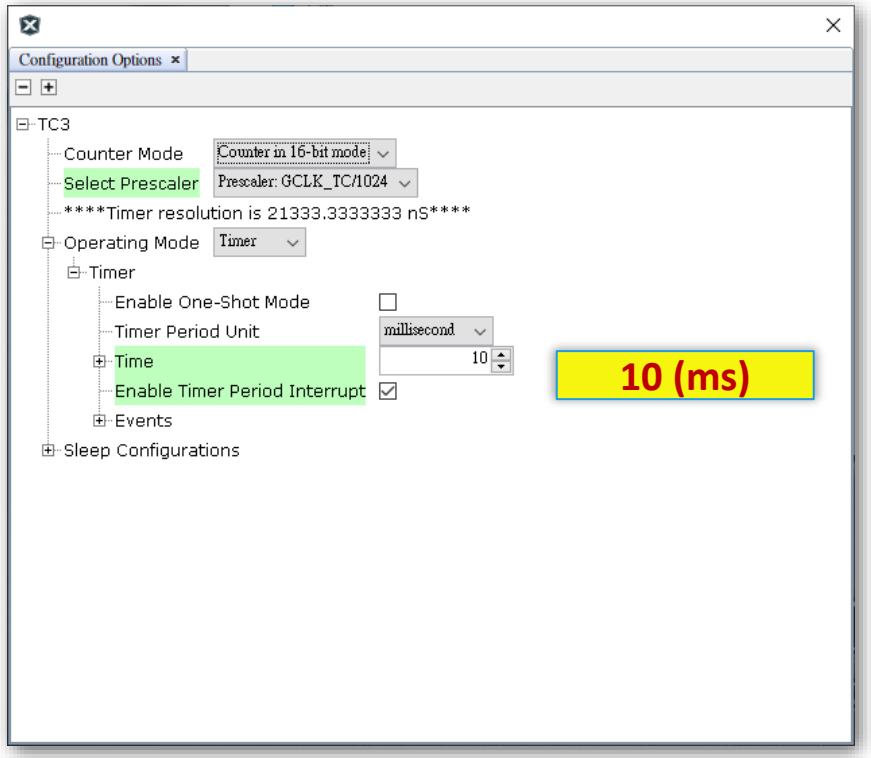
- **Configure ADC configuration options as below.**



🛠 Lab13-1 ADC Accumulation and Averaging

Step 2

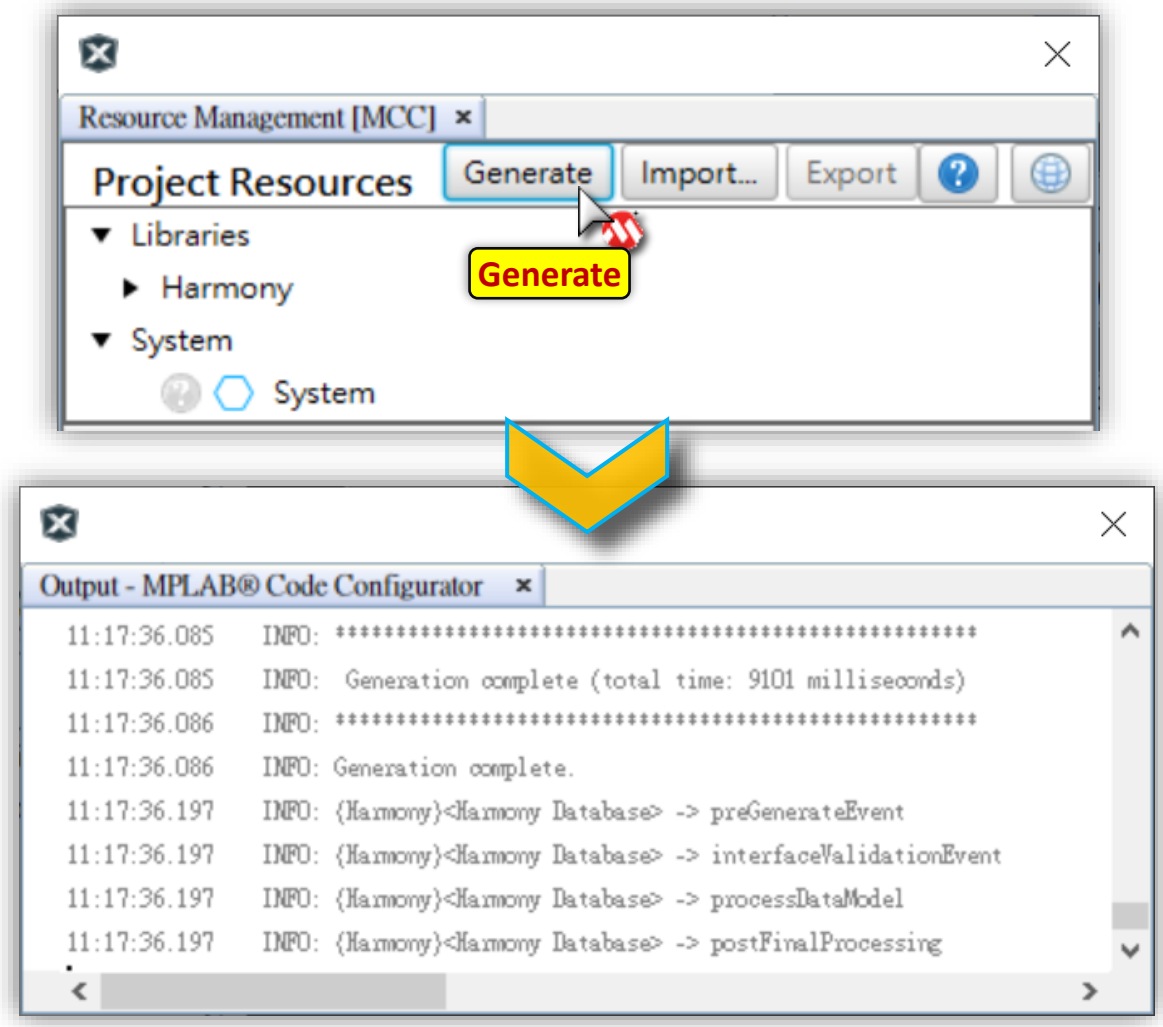
- **Modify TC3 to set 10ms period interval for speed up ADC conversion rate.**



🛠 Lab13-1 ADC Accumulation and Averaging

Step 3

- Click "Generate" Button to Generate your Code.



🛠 Lab13-1 ADC Accumulation and Averaging

Step 4

```
uint16_t ADC_Result[2];
volatile uint8_t ADC_IsCompleted = 0;
volatile int8_t ADC_ChannelIdx = 0;
...
void ADC_Complete( ADC_STATUS status, uintptr_t context ) { ... }
...
int main (void)
{  …
    while ( true )
    {   …
        if (TC3_HasExpired)
        {   …
            ADC_Conversi
        }
        …
        if( ADC_IsCompleted )
        {
            ADC_IsCompleted = 0;

            myprintf("\033[2;1H");
            myprintf( "VR1 Value : %4d", ADC_Result[0] );
            myprintf("\033[3;1H");
            myprintf( "Temperature Value : %4d", ADC_Result[1] );
        }
    }
```

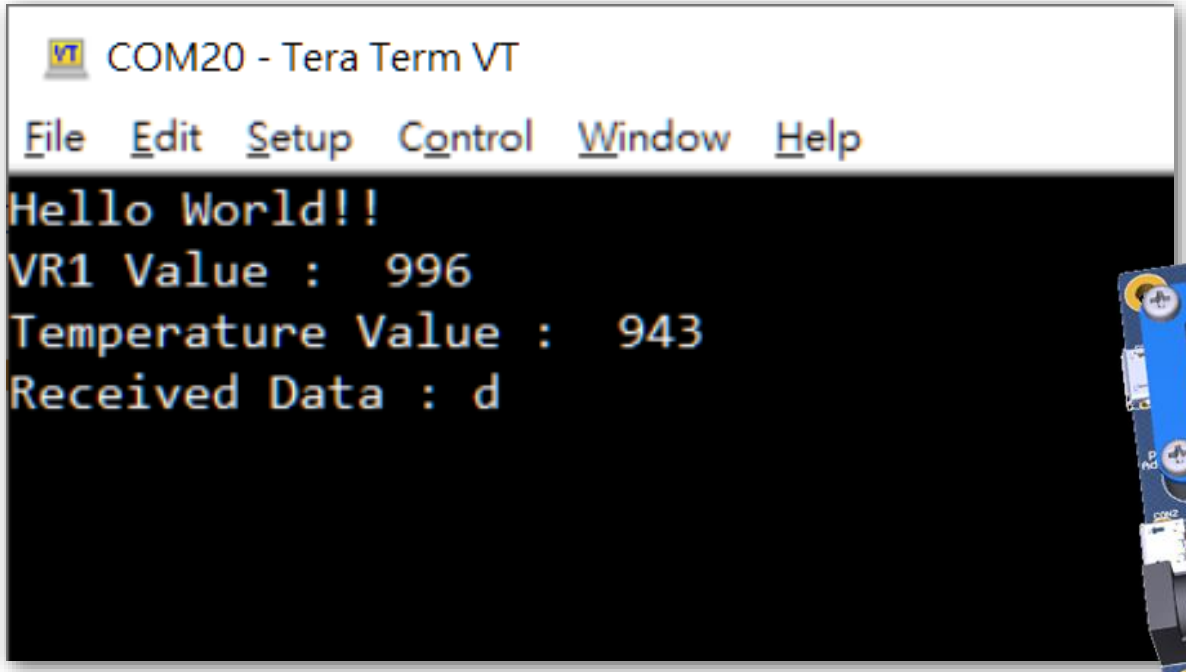
Nothing to modify in main.c

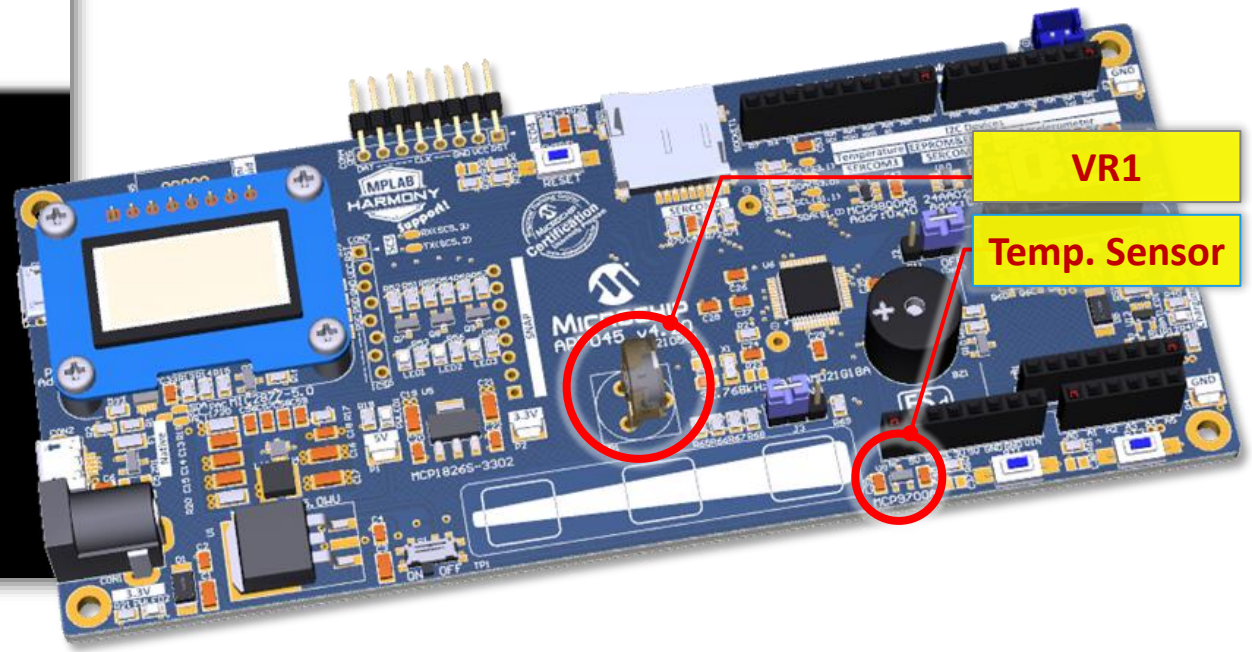
MICROCHIP

🛠 Lab13-1 ADC Accumulation and Averaging

Result

- **VR1 / Temperature Sensor ADC value will be averaged and more stable and smoothly.**

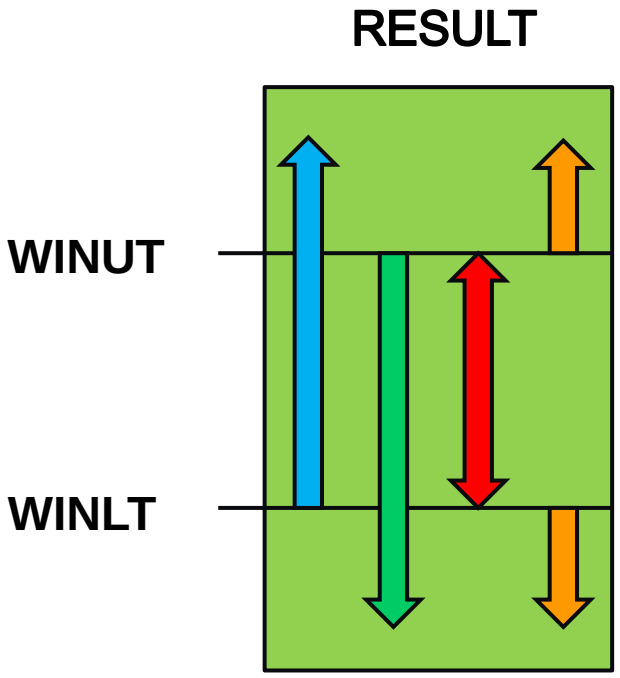




ADC Window Monitor

ADC Window Monitor

- The window monitor feature allows the conversion result in the RESULT register to be compared to predefined threshold values.
- There have four window mode could be use for monitor.



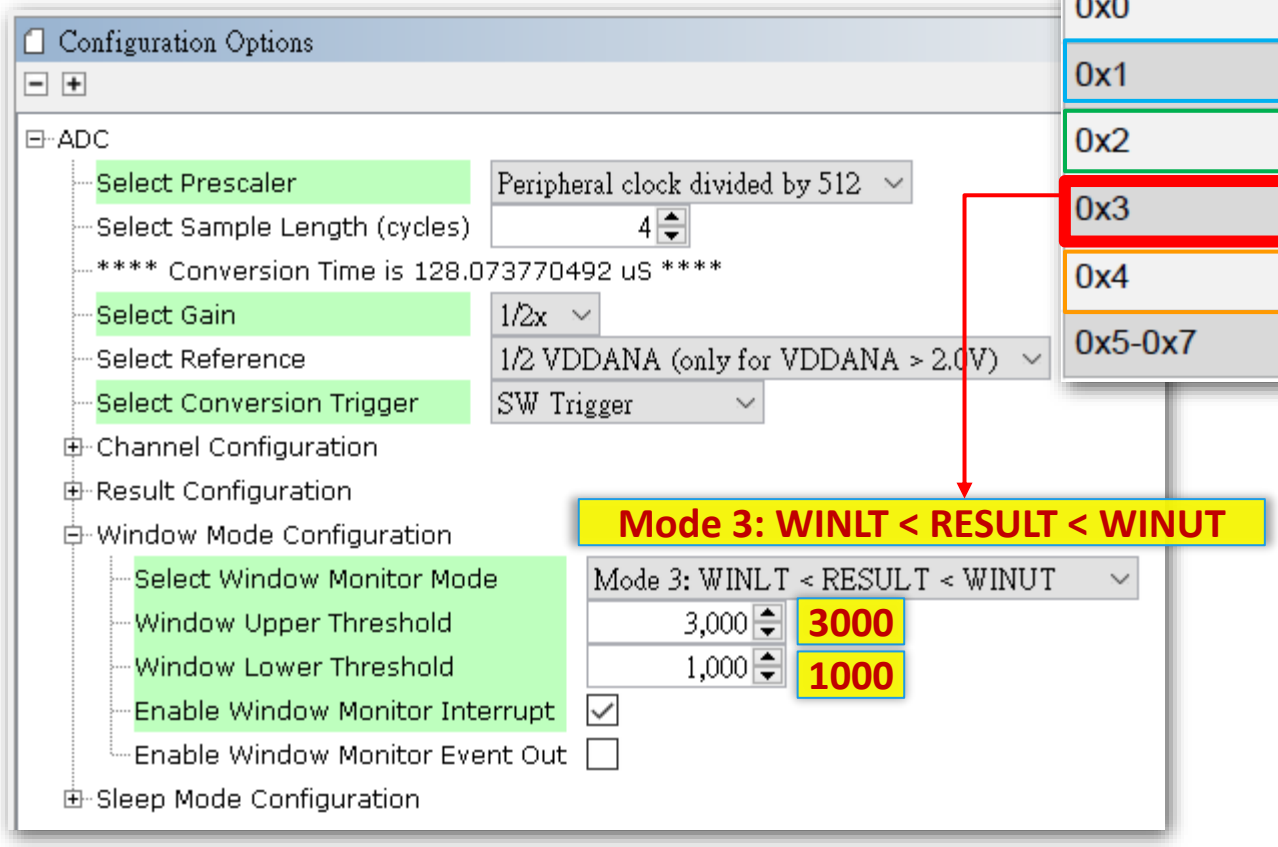
WINMODE[2:0]	Name	Description
0x0	DISABLE	No window mode (default)
0x1	MODE1	Mode 1: RESULT > WINLT
0x2	MODE2	Mode 2: RESULT < WINUT
0x3	MODE3	Mode 3: WINLT < RESULT < WINUT
0x4	MODE4	Mode 4: !(WINLT < RESULT < WINUT)
0x5-0x7		Reserved

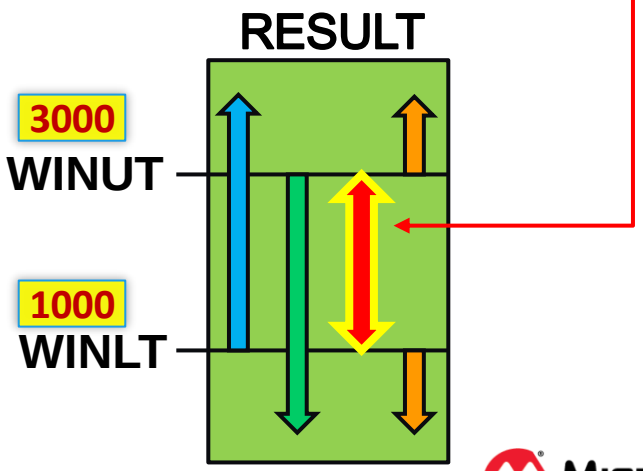
ADC Window Monitor

Example (Use Configuration Options)

- **Window Mode Mode3 with upper threshold 3000 and lower threshold 1000.**

WINMODE[2:0]	Name	Description
0x0	DISABLE	No window mode (default)
0x1	MODE1	Mode 1: RESULT > WINLT
0x2	MODE2	Mode 2: RESULT < WINUT
0x3	MODE3	Mode 3: WINLT < RESULT < WINUT
0x4	MODE4	Mode 4: !(WINLT < RESULT < WINUT)
0x5-0x7		Reserved



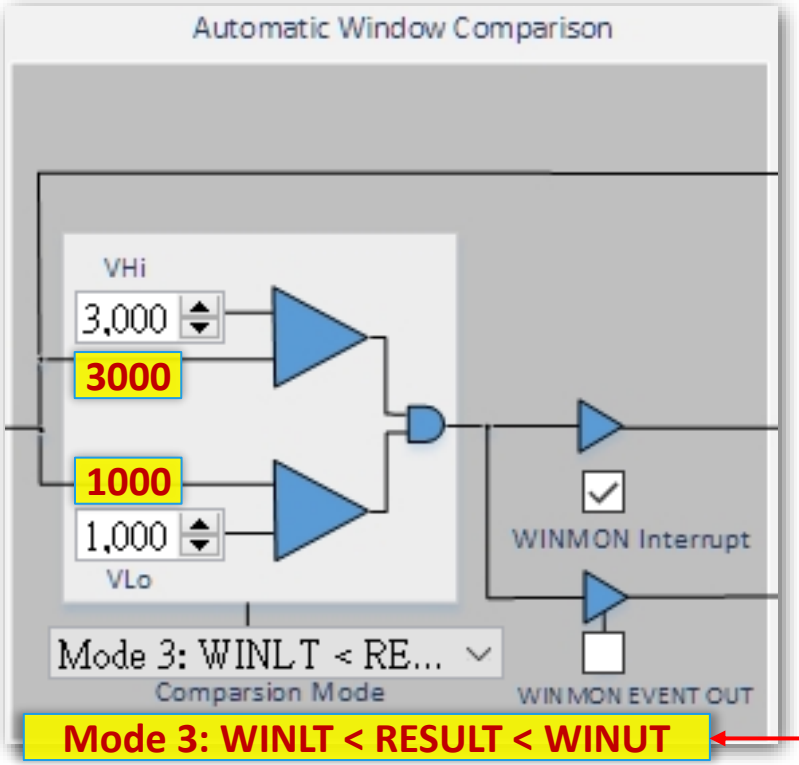


MICROCHIP

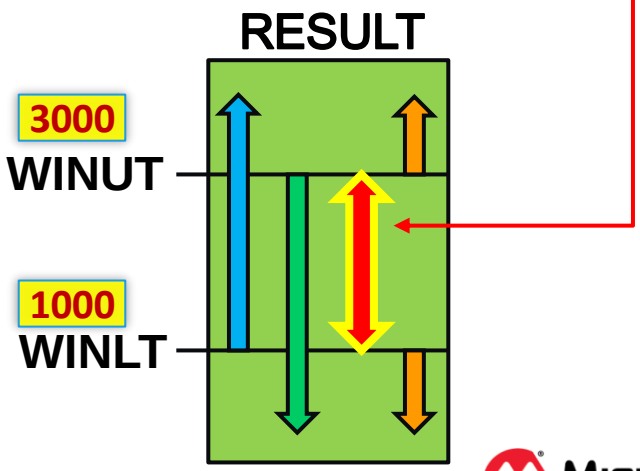
ADC Window Monitor

Example (Use Easy View)

- **Window Mode Mode3 with upper threshold 3000 and lower threshold 1000.**



WINMODE[2:0]	Name	Description
0x0	DISABLE	No window mode (default)
0x1	MODE1	Mode 1: RESULT > WINLT
0x2	MODE2	Mode 2: RESULT < WINUT
0x3	MODE3	Mode 3: WINLT < RESULT < WINUT
0x4	MODE4	Mode 4: !(WINLT < RESULT < WINUT)
0x5-0x7		Reserved



MICROCHIP

ADC Window Monitor

Interface Routines

```
void ADC_Initialize( void );
void ADC_Enable( void );
void ADC_Disable( void );
void ADC_ChannelSelect( ADC_POSINPUT positiveInput, ADC_NEGINPUT negativeInput );
void ADC_ConversionStart( void );
uint16_t ADC_ConversionResultGet( void );
void ADC_ComparisonWindowSet(uint16_t low_threshold, uint16_t high_threshold);
void ADC_WindowModeSet(ADC_WINMODE mode);
void ADC_CallbackRegister( ADC_CALLBACK callback, uintptr_t context );
```

🛠 Lab13-2 ADC Window Monitor

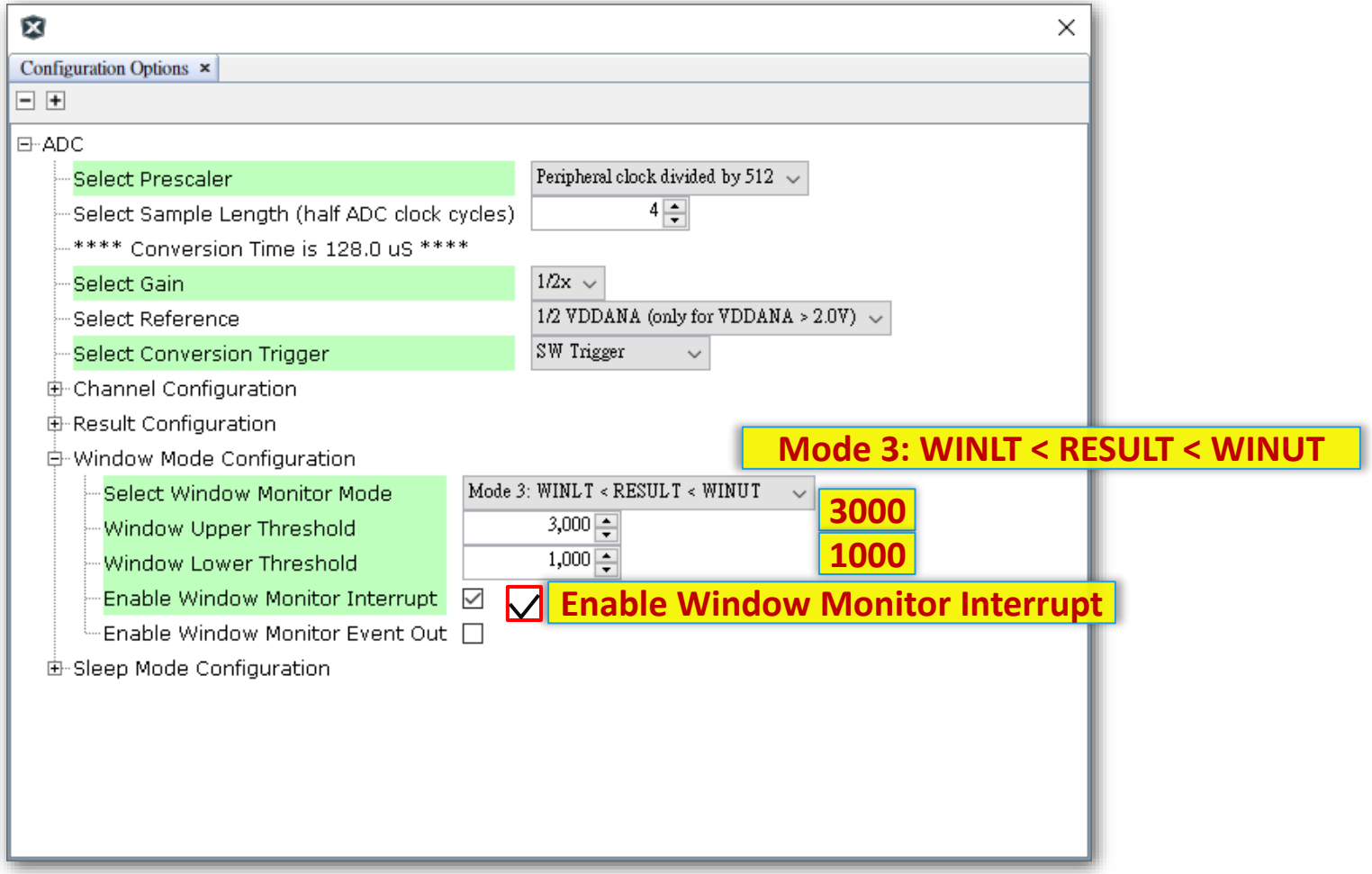
- Enable Window Monitor mode.
- Set upper threshold to 3000 and lower threshold to 1000.
- Output ADC window status while ADC value within upper and lower threshold. (Mode 3).

- **Let’s go!**

Lab13-2 ADC Window Monitor

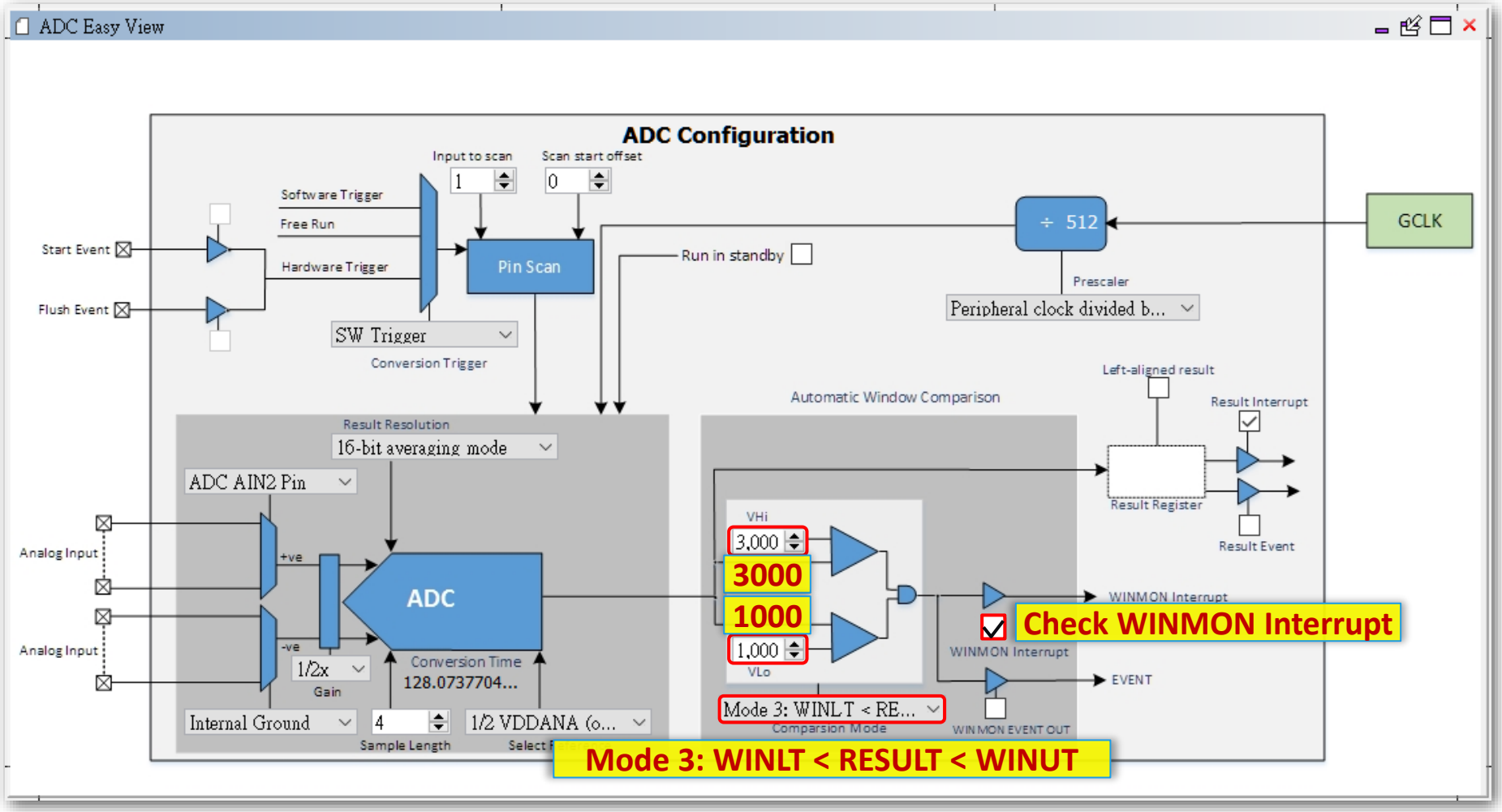
Step 1 (Use Configuration Options)

- **Configure ADC configuration options as below.**



Lab13-2 ADC Window Monitor

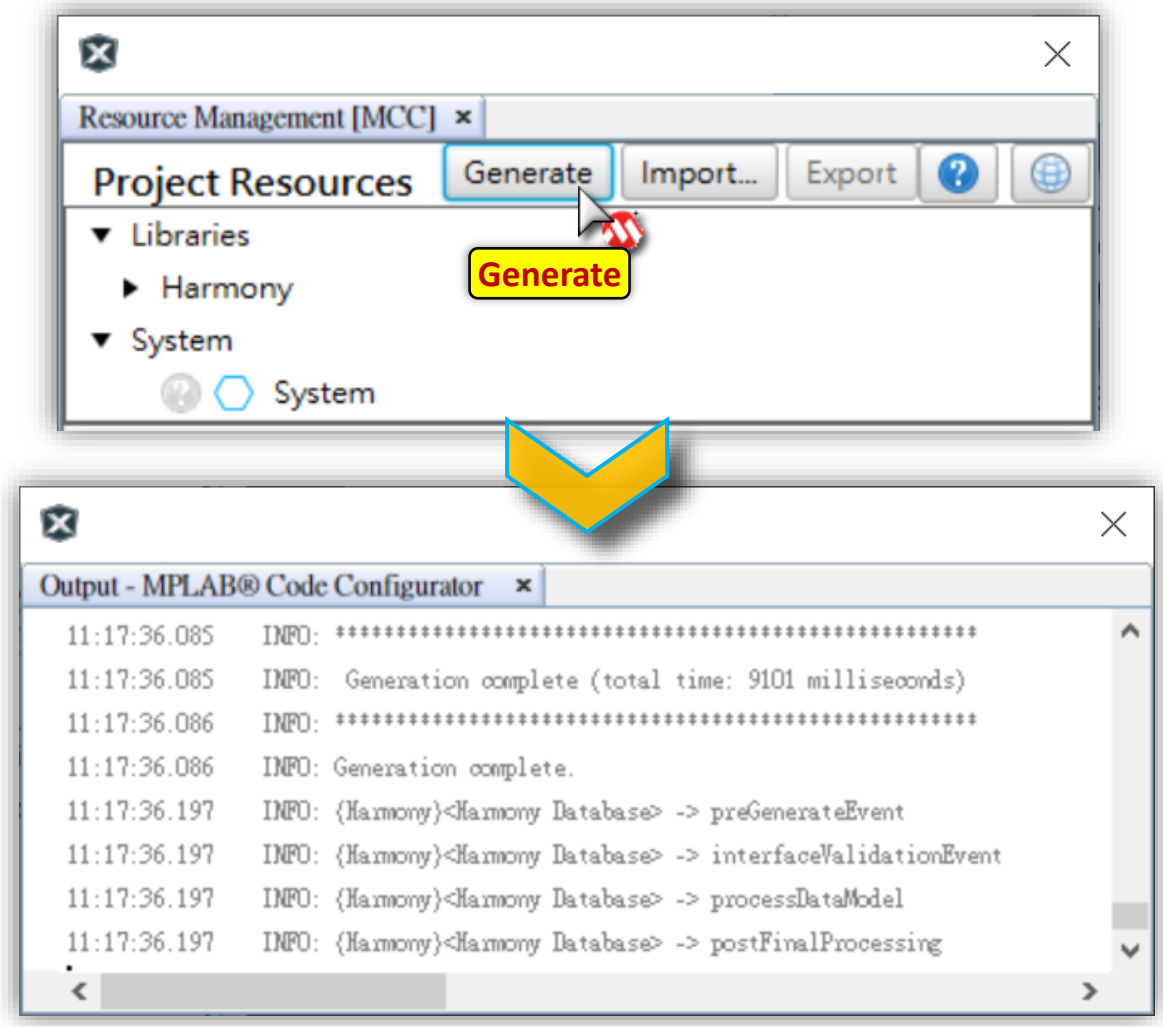
Step 1 (Use Easy View)



🛠 Lab13-2 ADC Window Monitor

Step 2

- Click "Generate" Button to Generate your Code.



MICROCHIP

🛠 Lab13-2 ADC Window Monitor

Step 3

```
// TODO 13-2.01
volatile uint16_t ADC_InWindow[2] = { 0, 0 };

void ADC_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_INTFLAG_RESRDY_Msk )
    {
        ADC_Result[ADC_ChannelIdx] = ADC_ConversionResultGet();

        // TODO 13-2.02
        ADC_InWindow[ADC_ChannelIdx] = 0;
        if( status & ADC_INTFLAG_WINMON_Msk )
        {
            ADC_InWindow[ADC_ChannelIdx] = 1;
        }

        ADC_ChannelIdx = ( ADC_ChannelIdx + 1 ) % 2;
        if( ADC_ChannelIdx == 0 )
        {
            ADC_IsCompleted = 1;
        }
    }
}

void USART5_Transmit( uintptr_t context ) { ... }
```

🛠 Lab13-2 ADC Window Monitor

Step 4

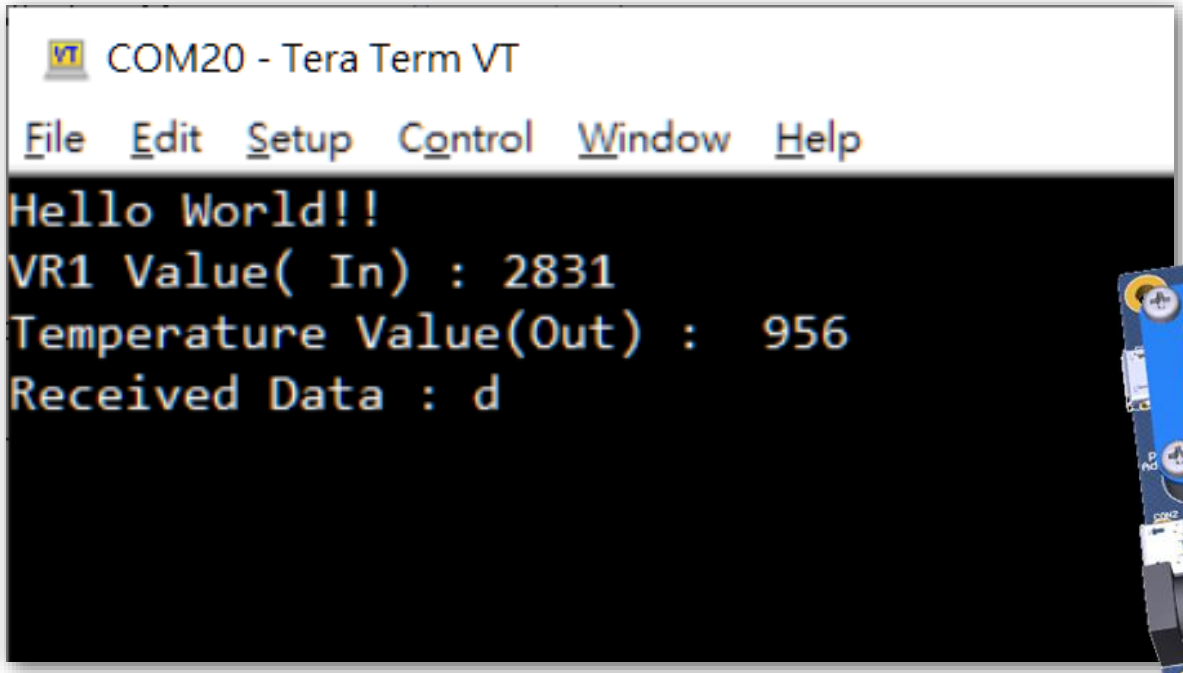
```
...
void ADC_Complete( ADC_STATUS status, uintptr_t context ) { ... }
...
int main (void)
{  …
    while ( true )
    {   …
        if( ADC_IsCompleted )
        {
            ADC_IsCompleted = 0;

            myprintf("\033[2;1H");
            // TODO 13-2.03
            myprintf( "VR1 Value(%s) : %4d",
                                    (ADC_InWindow[0] ? " In" : "Out"), ADC_Result[0]);

            myprintf("\033[3;1H");
            // TODO 13-2.04
            myprintf( "Temperature Value(%s) : %4d",
                                    (ADC_InWindow[1] ? " In" : "Out"), ADC_Result[1] );
        }
    }

    return ( EXIT_FAILURE );
}
```

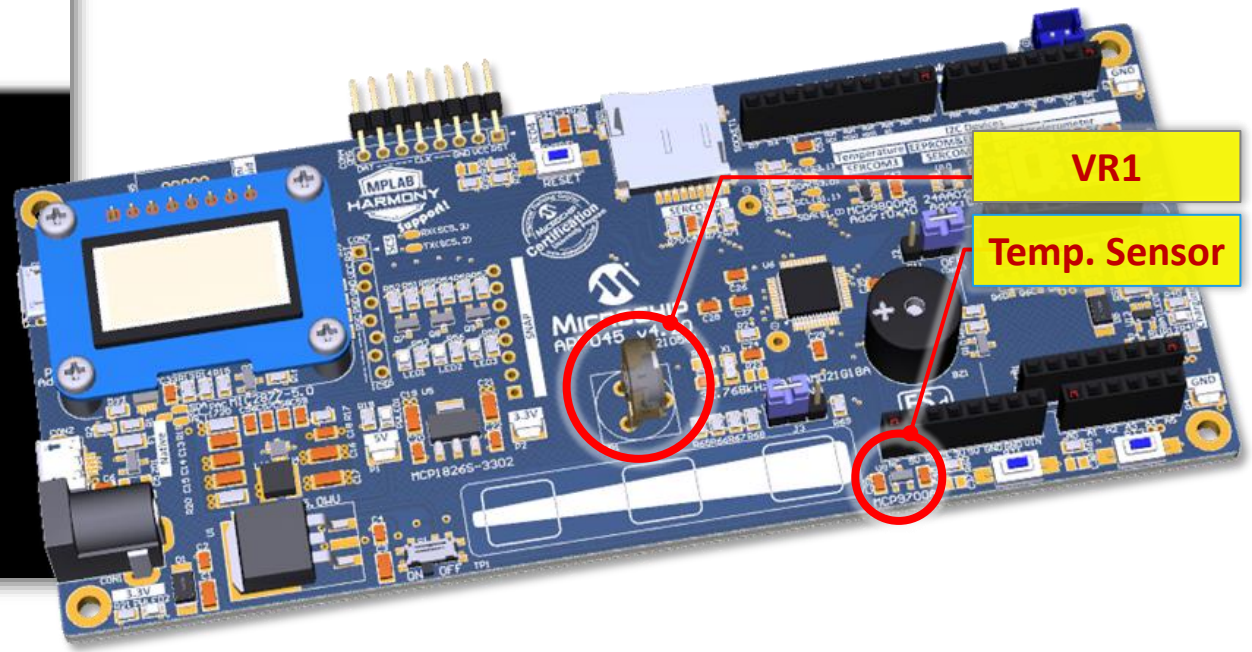
MICROCHIP

🛠 Lab13-2 ADC Window Monitor

Result

- If VR1 / Temperature Sensor ADC value in window will shows (In) else shows (Out).





🛠 Lab13-2 ADC Window Monitor

Discuss

- Use **INPUTOFFSET** of **INPUTCTRL** register to identify **next channel** of ADC PIN Scan.

```
void ADC_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_INTFLAG_RESRDY_Msk )
    {
        ADC_Result[ADC_ChannelIdx] = ADC_ConversionResultGet();

        ADC_InWindow[ADC_ChannelIdx] = 0;
        if( status & ADC_INTFLAG_WINMON_Msk )
        {
            ADC_InWindow[ADC_ChannelIdx] = 1;
        }

        // ADC_ChannelIdx = ( ADC_ChannelIdx + 1 ) % 2;
        ADC_ChannelIdx = (ADC_REGS->ADC_INPUTCTRL & ADC_INPUTCTRL_INPUTOFFSET_Msk)
                                                    >> ADC_INPUTCTRL_INPUTOFFSET_Pos;
        if( ADC_ChannelIdx == 0 )
        {
            ADC_IsCompleted = 1;
        }
    }
}
```

MICROCHIP

SAM2001 recall - ADC Pin Scan

- The ADC support analog channel switch call Pin Scan function.
- Pin scan switch analog channel sequential after a conversion automatically.
- Start Pin : Positive Input + Scan Start Offset
- End Pin : Start Pin + Number of Inputs to Scan

For Example : To do ADC Pin Scan from AIN[2] to AIN[3].

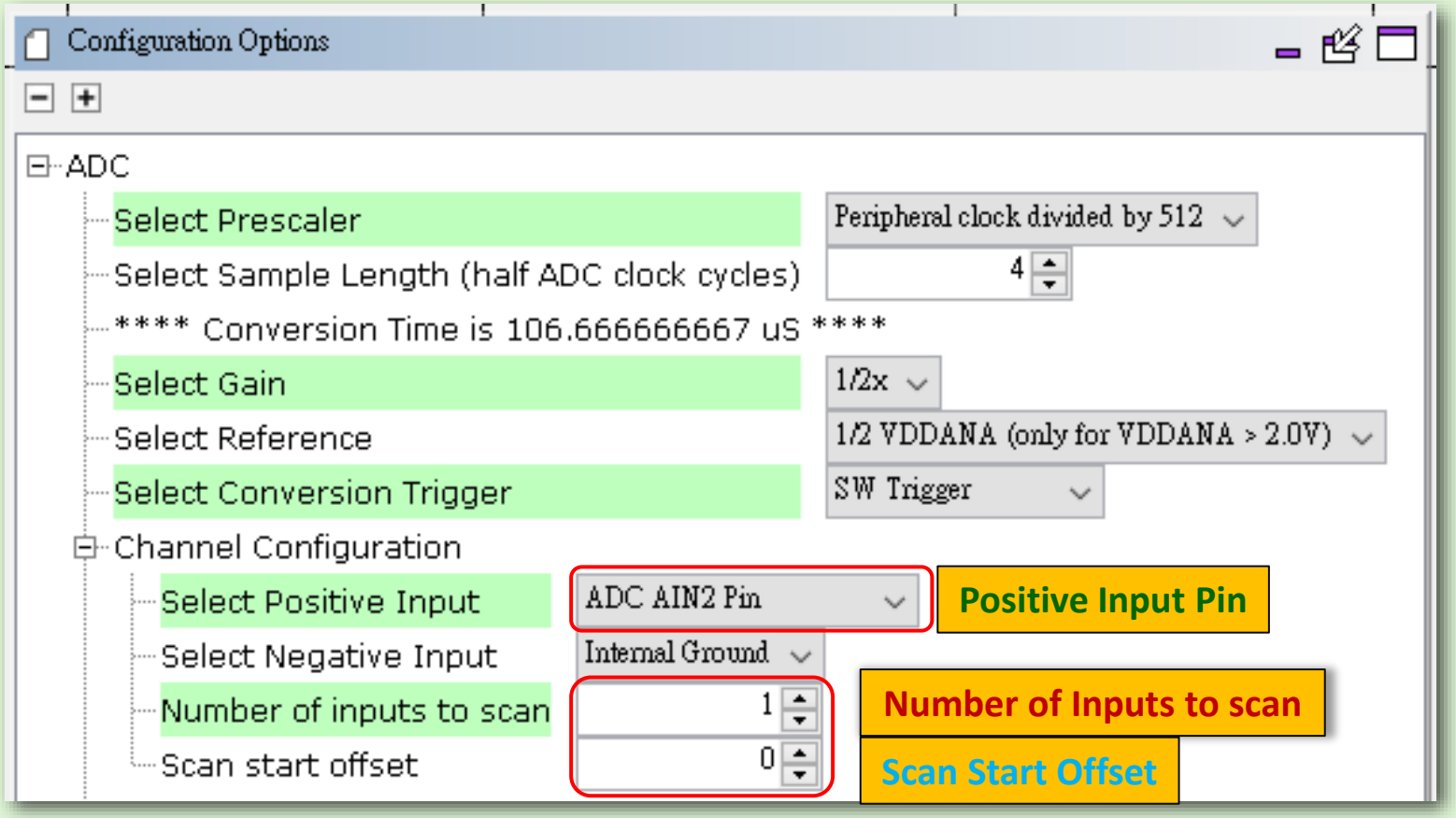
Positive Input Pin = AIN[0] Scan Start Offset = 2 Start Pin = AIN[0+2] = AIN[2]	OR	Positive Input Pin = AIN[2] Scan Start Offset = 0 Start Pin = AIN[2+0] = AIN[2]

Number of Inputs to scan = 1

End Pin = AIN[2+1] = AIN[3]

SAM2001 recall - ADC Pin Scan

Configuration Options Example



ADC Oversampling

Discuss

Example :

Result Resolution : 15 bits
Increase bits of resolution : 3 bits (15bits – 12bits = 3 bits)
Number of Sample to Average : ∑ 64 (4^3 samples = 2^6 samples)
Intermediate Result precision : 18 bits (12bits + 6 = 18 bits overflow)
Number of Automatic Right Shift : 2 bits (18bits >> 2 = 16bits)
Adjust Division Factor : /2 (2^1)
Finial Result Precision : 15 bits (16bit >> 1 = 15 bits)

4^n samples must be accumulated

RESSEL[1:0]	Name	Description
0x0	12BIT	12-bit result
0x1	16BIT	For averaging mode output
0x2	10BIT	10-bit result
0x3	8BIT	8-bit result

Table 33-4. Configuration Required for Oversampling and Decimation

Result Resolution	Number of Samples to Average	AVGCTRL.SAMPLENUM[3:0]	Number of Automatic Right Shifts	AVGCTRL.ADJRES[2:0]
13 bits	$4^1 = 4$	0x2	0	0x1
14 bits	$4^2 = 16$	0x4	0	0x2
15 bits	$4^3 = 64$	0x6	2	0x1
16 bits	∑ 64 ($4^3 = 2^6$)	0x8	4	0x0

4^3 2^6 (18>>2) 2^1
18 bits (12+6)
16 bits (18-2)
15 bits (16>>1)

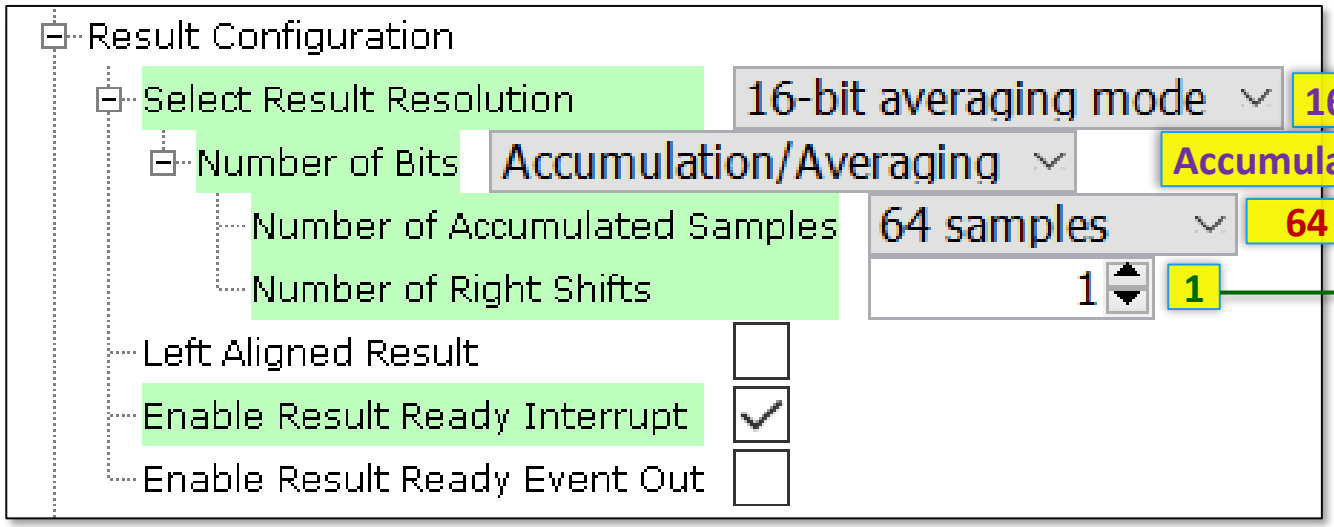
MICROCHIP

ADC Oversampling

Discuss

Example :

Result Resolution : 15 bits
Increase bits of resolution : 3 bits (15bits – 12bits = 3 bits)
Number of Sample to Average : ∑ 64 (4^3 samples = 2^6 samples)
Intermediate Result precision : 18 bits (12bits + 6 = 18 bits overflow)
Number of Automatic Right Shift : 2 bits (18bits >> 2 = 16bits)
Adjust Division Factor : /2 (2^1)
Finial Result Precision : 15 bits (16bit >> 1 = 15 bits)



16-bit averaging mode

Accumulation/Averaging

64 samples

1

RESSEL[1:0]	Name	Description
0x0	12BIT	12-bit result
0x1	16BIT	For averaging mode output
0x2	10BIT	10-bit result
0x3	8BIT	8-bit result

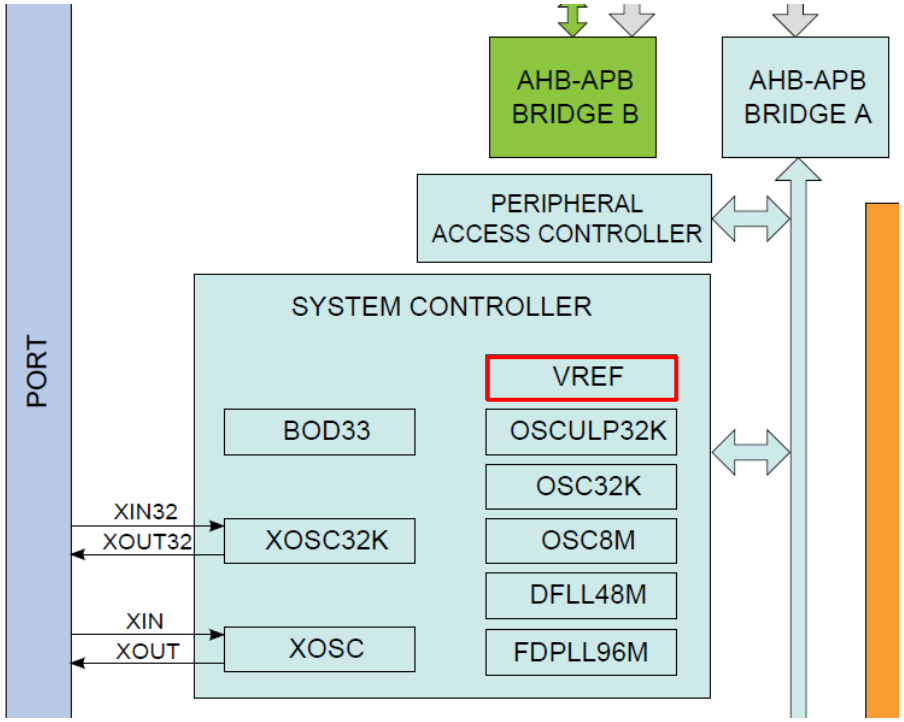
MICROCHIP

Internal Temperature Sensor

Internal Temperature Sensor

Voltage Reference System

- The **Voltage Reference System (VREF)** consists of a **Bandgap Reference Voltage** Generator and a **Temperature Sensor**.
- Write one to the **Temperature Sensor Enable** bit (**VREF.TSEN**) in **SYSCTRL.VREF** register could enable the integrated temperature sensor.



SYSCTRL – System Controller

Offset	Name	Bit Pos.								
0x40	VREF	7:0						BGOUTEN	TSEN	
		15:8								
		23:16	CALIB[7:0]							
		31:24					CALIB[10:8]			

Note: The TSEN bit is not available on the SAMDA1 device

https://onlinedocs.microchip.com/pr/GUID-2BBF0A73-A043-4A40-A6CA-0FF154E4BFFE-en-US-6/index.html?GUID-81A97F6E-0A69-40BC-8663-385337B2B80E

Internal Temperature Sensor

Characteristics

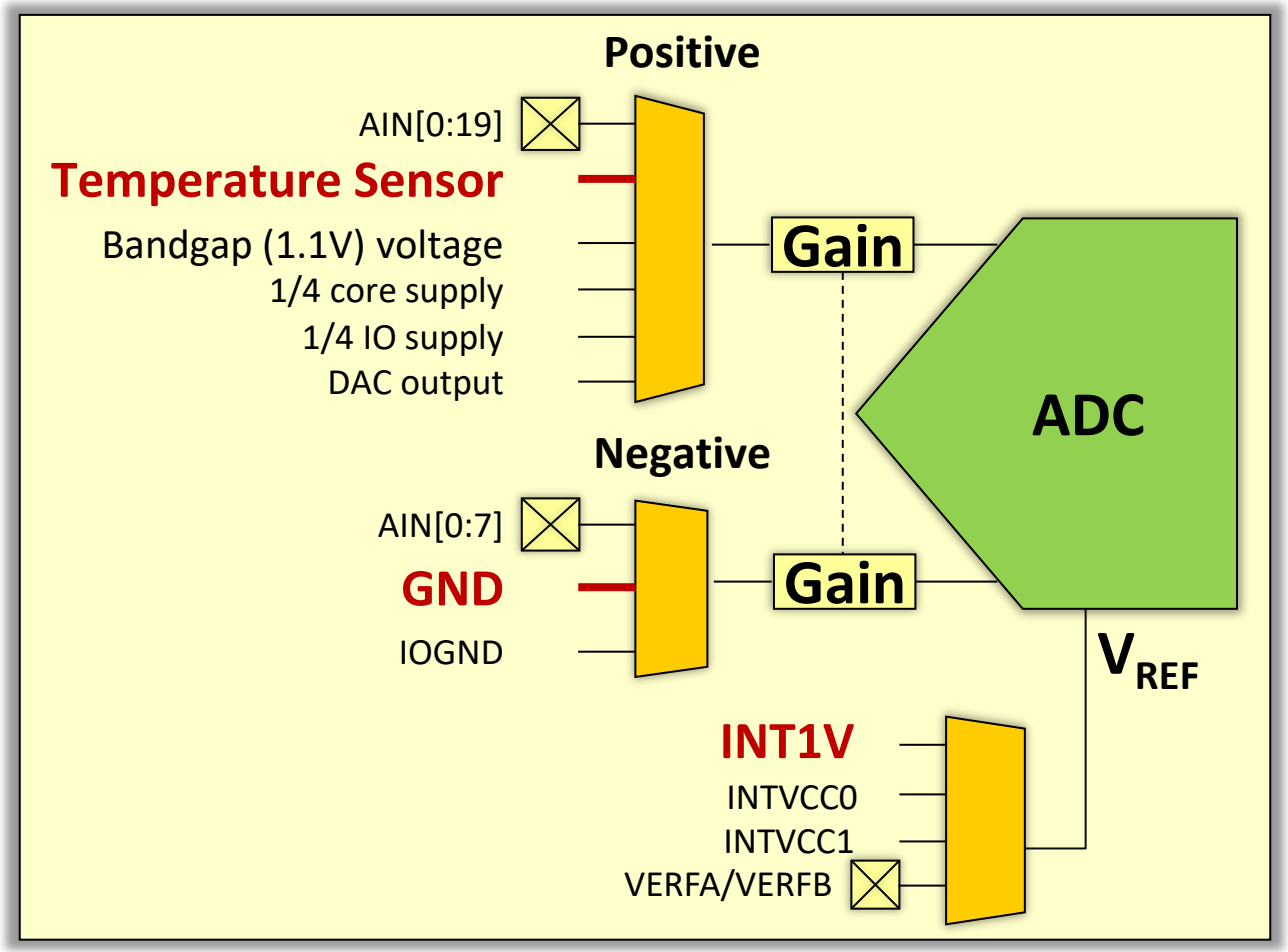
Table 37-40. Temperature Sensor Characteristics(1) (Device Variant B,C, D and L)

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Units
	Temperature sensor output voltage	T= 25°C, V_{DDANA} = 3.3V	-	0.688	-	V
	Temperature sensor slope		2.06	2.16	2.26	mV/°C
	Variation over V_{DDANA} voltage	V_{DDANA}=1.62V to 3.6V	-0.4	1.4	3	mV/V
	Temperature Sensor accuracy	Using the method described in the 37.11.8.2 Software-based Refinement of the Actual Temperature	-10	-	10	°C

Internal Temperature Sensor

Measurement by ADC with INT1V reference voltage

- The **Integrated Temperature Sensor** could be accessible by using **ADC** with **INT1V** reference voltage for measure the **Core Temperature** in linear voltage proportional.



Internal Temperature Sensor

Factory Calibration

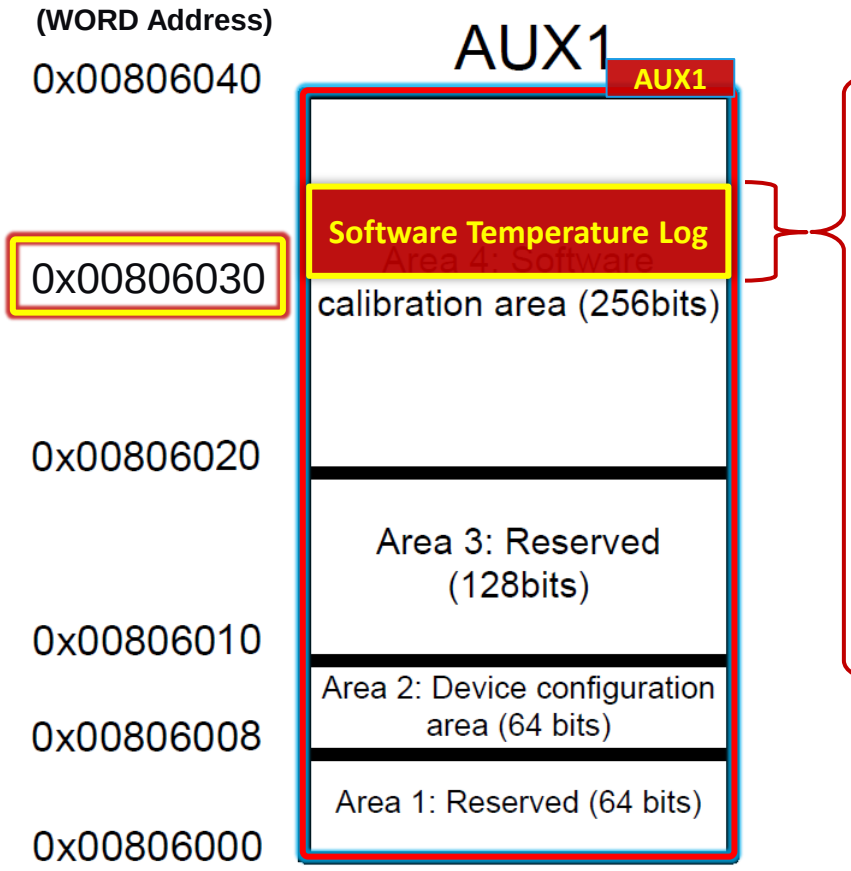
- **Software-based Refinement of the Actual Temperature**
  The temperature sensor behavior is linear but depends on several parameters such as the **internal voltage reference**, which itself depends on the temperature.

- To take this into account, each device contains a **Temperature Log row** with data measured and **written during the production tests**. These calibration values should be read by software to infer the most accurate temperature readings possible.

- This Software **Temperature Log row** can be read at address **0x00806030**, This section specifies the Temperature Log row content and explains how to refine the temperature sensor output using the values in the Temperature Log row.

Temperature Log Row of NVM

AUX1 (Area 4 : Software Calibration area) – Read Only



Temperature Log Row Content

Bit position	Name	Description
7:0	ROOM_TEMP_VAL_INT	Integer part of room temperature in °C
11:8	ROOM_TEMP_VAL_DEC	Decimal part of room temperature
19:12	HOT_TEMP_VAL_INT	Integer part of hot temperature in °C
23:20	HOT_TEMP_VAL_DEC	Decimal part of hot temperature
31:24	ROOM_INT1V_VAL	2's complement of the internal 1V reference drift at room temperature (versus a 1.0 centered value)
39:32	HOT_INT1V_VAL	2's complement of the internal 1V reference drift at hot temperature (versus a 1.0 centered value)
51:40	ROOM_ADC_VAL	12-bit ADC conversion at room temperature
63:52	HOT_ADC_VAL	12-bit ADC conversion at hot temperature

Calibration condition :

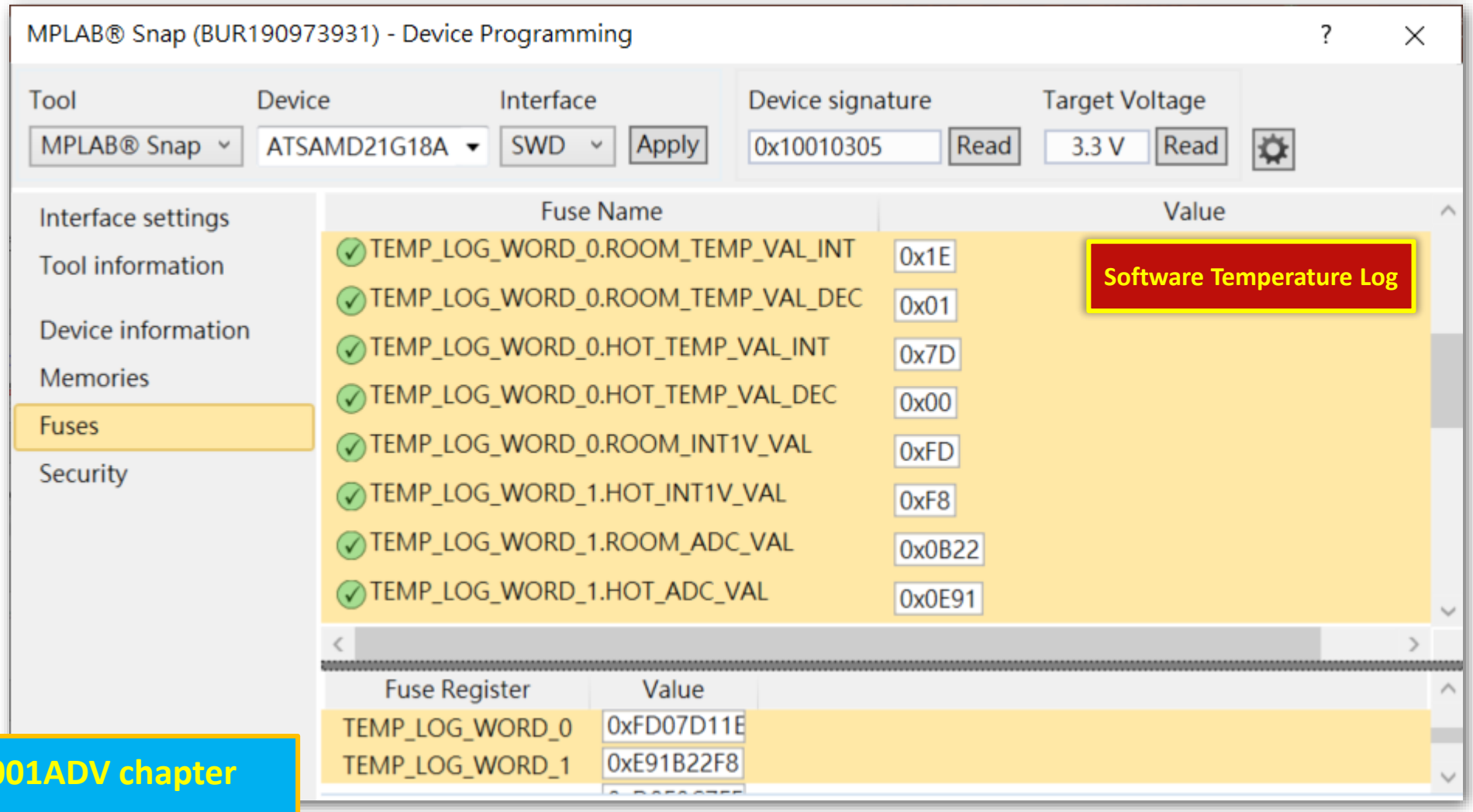
- $V_{DDIN} = V_{DDIO} = V_{DDANA} = 3.3V$
- ADC Clock speed = 1MHz (Free running mode)
- ADC averaging mode with 4 averaged samples
- ADC voltage reference = 1.0V (Internal reference INT1V)
- ADC input = Temperature sensor
- Room Temperature 25.2°C, • Hot temperature 83.3°C

Learn this detail in SAM2001ADV chapter Auxiliary Space of NVM and Fuse Maintains

MICROCHIP

Temperature Log Row of NVM

Temperature Log could be read and check in Microchip Studio

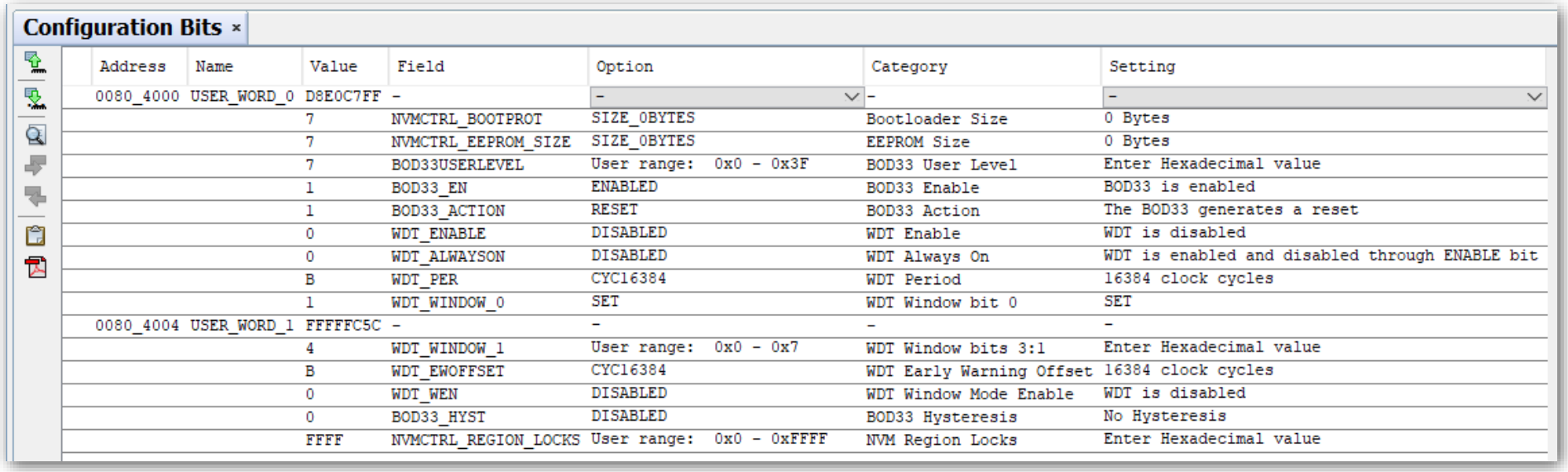


Learn this detail in SAM2001ADV chapter Auxiliary Space of NVM and Fuse Maintains

Temperature Log Row of NVM

Temperature Log Row cannot read for check in MPLAB X IDE

Temperature Log not in MPLAB XIDE Configuration Bits window



Address	Name	Value	Field	Option	Category	Setting
0080_4000	USER_WORD_0	D8E0C7FF	-	-	-	-
		7	NVMCTRL_BOOTPROT	SIZE_0BYTES	Bootloader Size	0 Bytes
		7	NVMCTRL_EEPROM_SIZE	SIZE_0BYTES	EEPROM Size	0 Bytes
		7	BOD33USERLEVEL	User range: 0x0 - 0x3F	BOD33 User Level	Enter Hexadecimal value
		1	BOD33_EN	ENABLED	BOD33 Enable	BOD33 is enabled
		1	BOD33_ACTION	RESET	BOD33 Action	The BOD33 generates a reset
		0	WDT_ENABLE	DISABLED	WDT Enable	WDT is disabled
		0	WDT_ALWAYSON	DISABLED	WDT Always On	WDT is enabled and disabled through ENABLE bit
		B	WDT_PER	CYC16384	WDT Period	16384 clock cycles
		1	WDT_WINDOW_0	SET	WDT Window bit 0	SET
0080_4004	USER_WORD_1	FFFFFC5C	-	-	-	-
		4	WDT_WINDOW_1	User range: 0x0 - 0x7	WDT Window bits 3:1	Enter Hexadecimal value
		B	WDT_EWOFFSET	CYC16384	WDT Early Warning Offset	16384 clock cycles
		0	WDT_WEN	DISABLED	WDT Window Mode Enable	WDT is disabled
		0	BOD33_HYST	DISABLED	BOD33 Hysteresis	No Hysteresis
		FFFF	NVMCTRL_REGION_LOCKS	User range: 0x0 - 0xFFFF	NVM Region Locks	Enter Hexadecimal value

Use Temperature Log Row to Calculation Temperature

- (ROOM_TEMP_VAL_INT, ROOM_TEMP_VAL_DEC) to denote as **$Temp_R$**
- (HOT_TEMP_VAL_INT, HOT_TEMP_VAL_DEC) to denote as **$Temp_H$**
- ROOM_INT1V_VAL calculate by **1-(ROOM_INT1V_VAL/1000)** to denote as **$INT1V_R$**
- HOT_INT1V_VAL calculate by **1-(HOT_INT1V_VAL/1000)** to denote as **$INT1V_H$**
- ROOM_ADC_VAL to denote as **ADC_R** with V_{REF} **$INT1V_R$** , then the measured voltage is denoted **V_{ADCR}**
- HOT_ADC_VAL to denote as **ADC_H** with V_{REF} **$INT1V_H$** , then the measured voltage is denoted **V_{ADCH}**

	Source Field of Log Row	Symbol	Room	Hot
Temperature	XXXX_TEMP_VAL_INT XXXX_TEMP_VAL_DEC	Temp	30.1 ℃	125 ℃
V_{REF} voltage	XXXX_INT1V_VAL	INT1V	1.00 V	1.01 V
ADC value	XXXX_ADC_VAL	ADC	2850	3729
ADV voltage	XXXX_ADC_VAL	V_{ADC}	0.7 V	0.92 V

Note : Temperature Log Row are not equal from chip to chip, here is an example !!

MICROCHIP

Use Temperature Log Row to Calculation Temperature

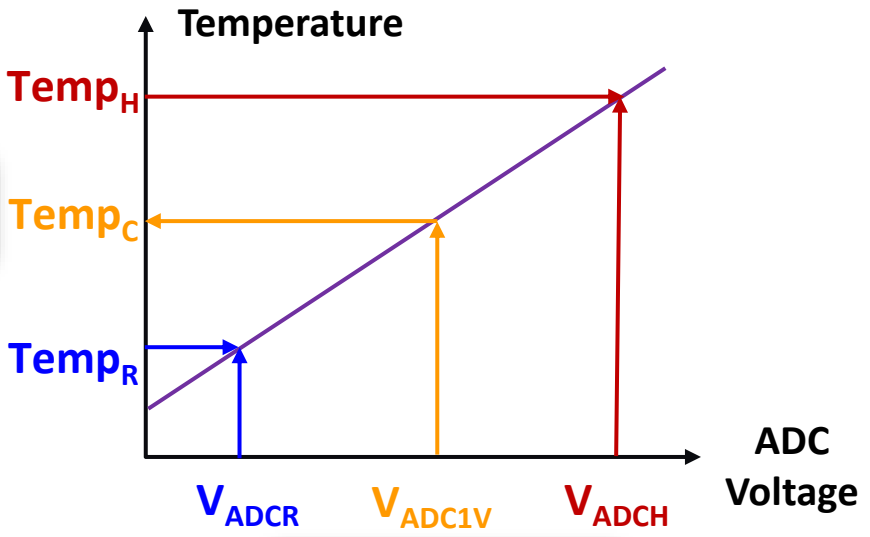
- (ROOM_TEMP_VAL_INT, ROOM_TEMP_VAL_DEC) to denote as $Temp_R$
- (HOT_TEMP_VAL_INT, HOT_TEMP_VAL_DEC) to denote as $Temp_H$
- ROOM_INT1V_VAL calculate by **1-(ROOM_INT1V_VAL/1000)** to denote as $INT1V_R$
- HOT_INT1V_VAL calculate by **1-(HOT_INT1V_VAL/1000)** to denote as $INT1V_H$
- ROOM_ADC_VAL to denote as ADC_R with V_{REF} $INT1V_R$, then the measured voltage is denoted V_{ADCR}
- HOT_ADC_VAL to denote as ADC_H with V_{REF} $INT1V_H$, then the measured voltage is denoted V_{ADCH}
- Temperature to measurement correspond to **Coarse** and **Finer** are denoted as $Temp_C$ and $Temp_F$
- V_{REF} INT1V correspond to **Ideal 1.0V** and **reality Measurement** are denoted as INT1V and $INT1V_M$
- ADC result denoted ADC_M, the ADC voltage to use the **Ideal 1.0V** V_{RFE} INT1V is denoted V_{ADC1V}
- ADC result denoted ADC_M, the ADC voltage to use the **reality Measurement** V_{RFE} $INT1V_M$ is denoted V_{ADCM}

Using the linear proportional, we got following equation:

$$\frac{(V_{ADC1V} - V_{ADCR})}{(Temp_C - Temp_R)} = \frac{(V_{ADCH} - V_{ADCR})}{(Temp_H - Temp_R)}$$

$$Temp_C = Temp_R + \frac{(V_{ADC1V} - V_{ADCR}) \times (Temp_H - Temp_R)}{(V_{ADCH} - V_{ADCR})}$$

Coarse Temperature



MICROCHIP

Use Temperature Log Row to Calculation Temperature

- **Now we convert the Voltage of previous expression to be expressed in ADC value to calculate the Coarse Temperature Value $Temp_C$**

$$Temp_C = Temp_R + \frac{(V_{ADC1V} - V_{ADCR}) \times (Temp_H - Temp_R)}{(V_{ADCH} - V_{ADCR})}$$

Ideal $1.0V_{REF}$

$$Temp_C = Temp_R + \frac{\left(\left(ADC_M \times \frac{INT1V}{4095}\right) - \left(ADC_R \times \frac{INT1V_R}{4095}\right)\right) \times (Temp_H - Temp_R)}{\left(\left(ADC_H \times \frac{INT1V_H}{4095}\right) - \left(ADC_R \times \frac{INT1V_R}{4095}\right)\right)}$$

$Temp_C$ we say the "Coarse Temperature value" because
we assume the reference voltage INT1V == 1.0V ideal voltage

The physical ADC resolution is 12 bits, then 2^{12}-1 = 4095

$$\text{Measured Voltage} = \text{ADC value} \times \frac{\text{Reference Voltage}}{4095}$$

MICROCHIP

Use Temperature Log Row to Calculation Temperature

- Using the linear proportional and by previous calculated result $Temp_C$ to calculate the reality V_{REF} $INT1V_M$ then derive the V_{ADCM}

Coarse Temperature

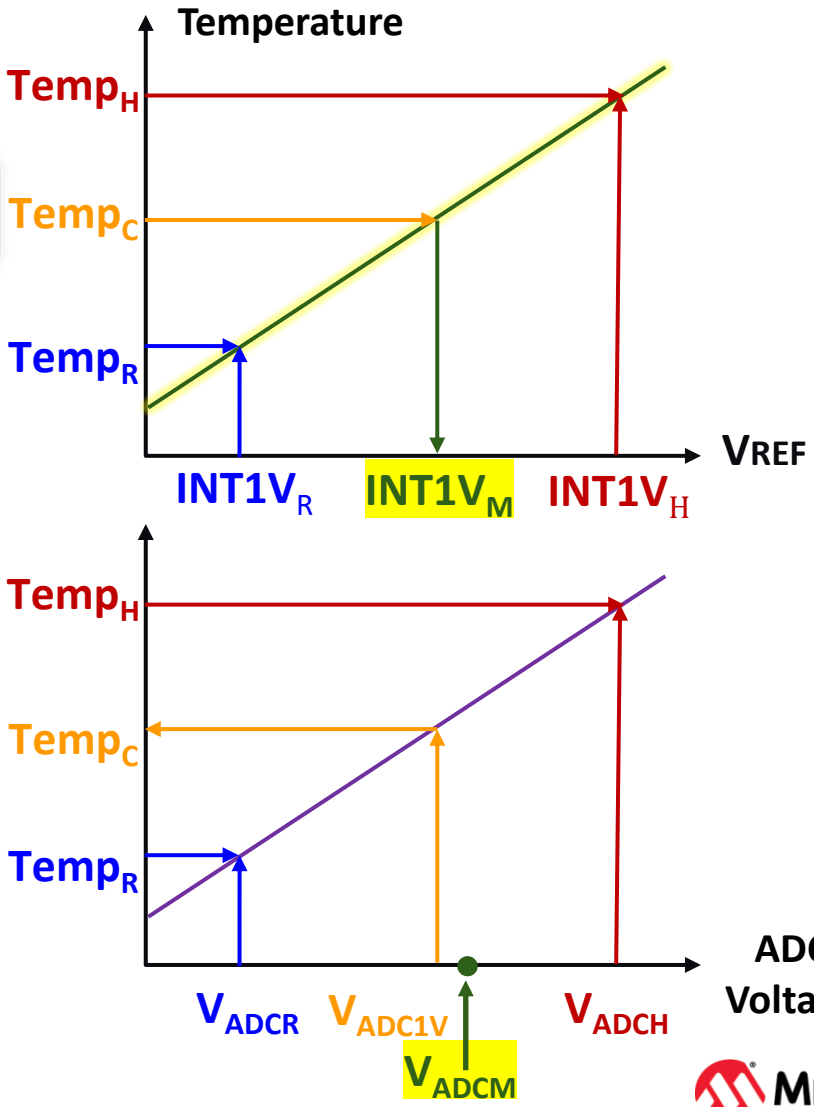
$$\frac{(INT1V_M - INT1V_R)}{(Temp_C - Temp_R)} = \frac{(INT1V_H - INT1V_R)}{(Temp_H - Temp_R)}$$

$$INT1V_M = INT1V_R + \frac{(INT1V_H - INT1V_R) \times (Temp_C - Temp_R)}{(Temp_H - Temp_R)}$$

Ideal 1.0V_{REF} Reality 1.0V_{REF}

$$V_{ADC1V} = ADC_M \times \frac{INT1V}{4095}, \quad V_{ADCM} = ADC_M \times \frac{INT1V_M}{4095}$$

Replace INT1V(1.0V ideal) by $INT1V_M$ to calculate the V_{ADCM}

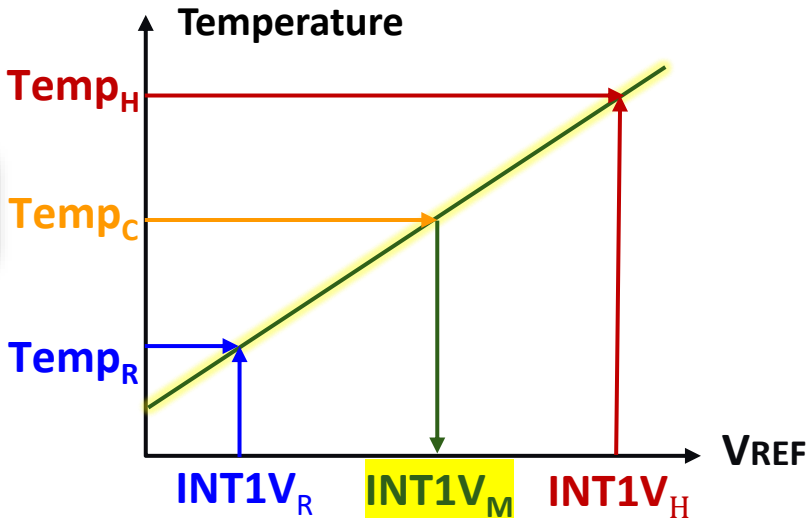


Use Temperature Log Row to Calculation Temperature

- Use V_{ADCM} to instead of the V_{ADC1V} to calculate the finer temperature value $Temp_F$

$$\frac{(V_{ADC1V} - V_{ADCR})}{(Temp_C - Temp_R)} = \frac{(V_{ADCH} - V_{ADCR})}{(Temp_H - Temp_R)}$$

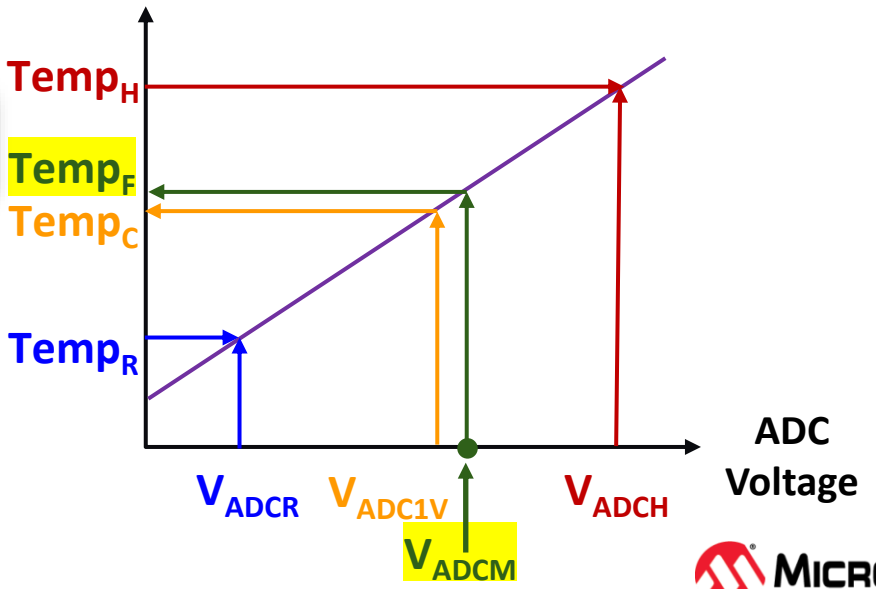
Coarse Temperature



Ideal 1.0V

$$Temp_C = Temp_R + \frac{(V_{ADC1V} - V_{ADCR}) \times (Temp_H - Temp_R)}{(V_{ADCH} - V_{ADCR})}$$

Finer Temperature



Reality 1.0V

$$Temp_F = Temp_R + \frac{(V_{ADCM} - V_{ADCR}) \times (Temp_H - Temp_R)}{(V_{ADCH} - V_{ADCR})}$$

Replace V_{ADC1V} by V_{ADCM} to calculate the finer result $Temp_F$

MICROCHIP

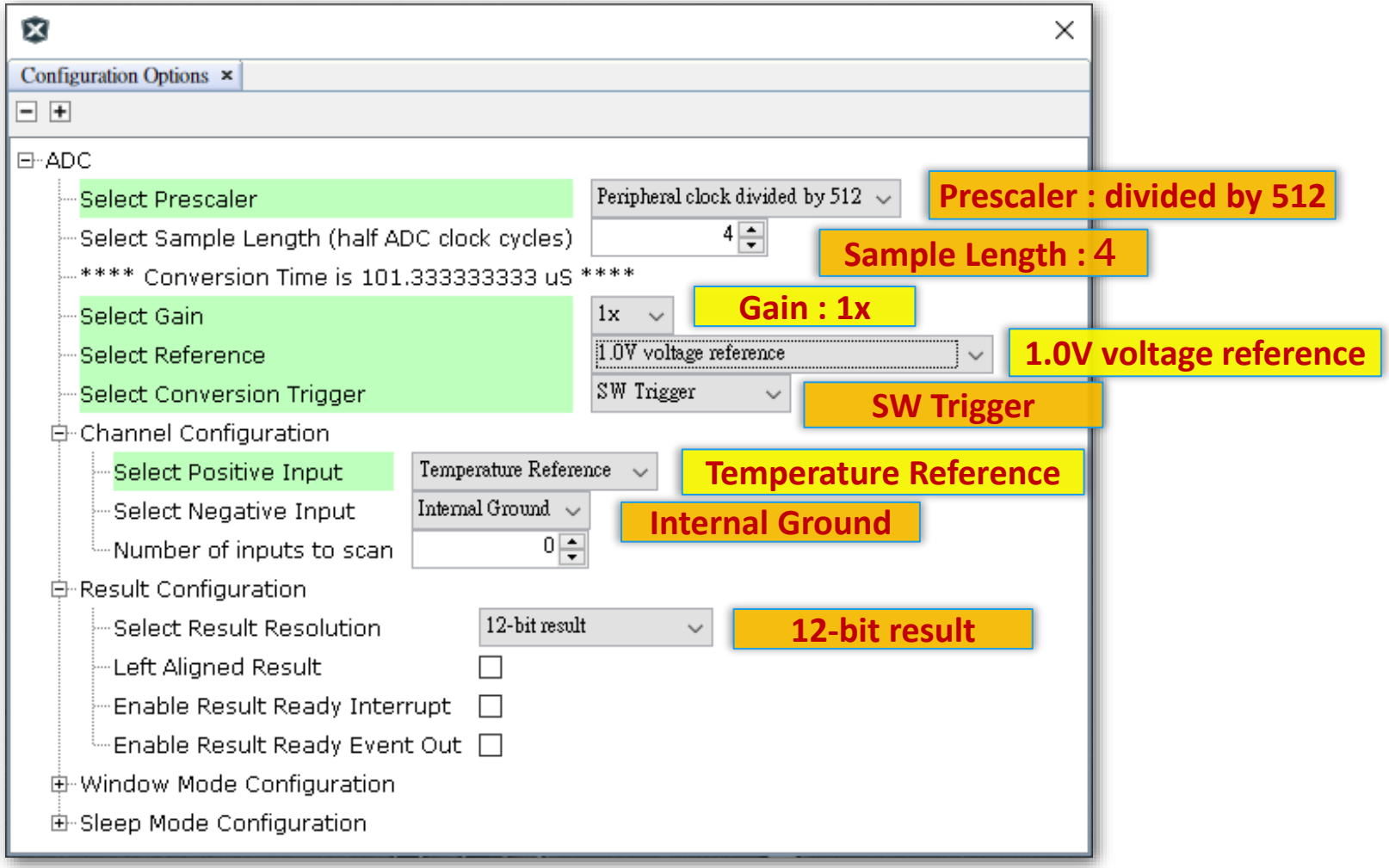
Lab12-1 Internal Temperature Sensor through ADC

- Please continue from SAM2001 Lab12 (ADC Polling Software Trigger)
- Try enable Internal Temperature Sensor and load NVM Temperature Log Row to calculate reality temperature through ADC conversion.
- Internal Temperature Sensor should configure to choose internal 1.0V bandgap voltage as reference voltage of ADC, no gain control (Gain 1x), Temperature Reference as positive input and 12-bits resolution.
- Try observe both ADC and Temperature Value use UART output.

- **Let's go!**

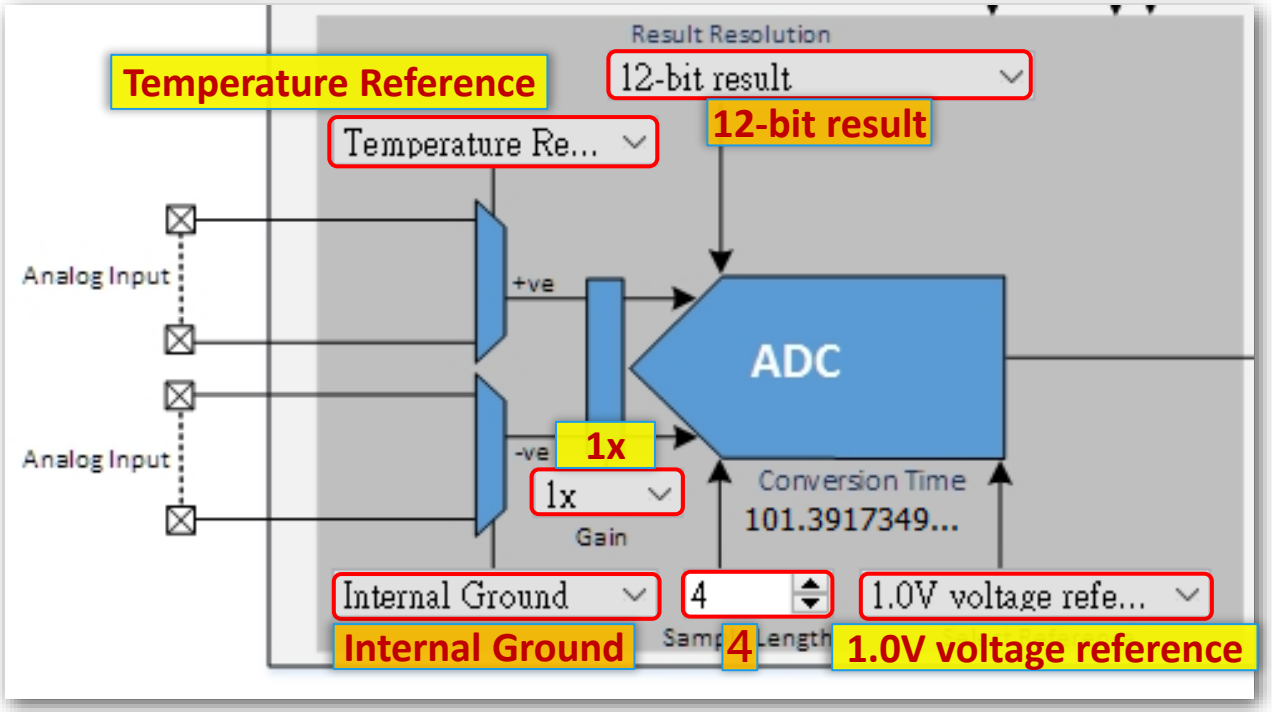
Lab12-1 Internal Temperature Sensor through ADC

Step 1 (Use Configuration Options)



Lab12-1 Internal Temperature Sensor through ADC

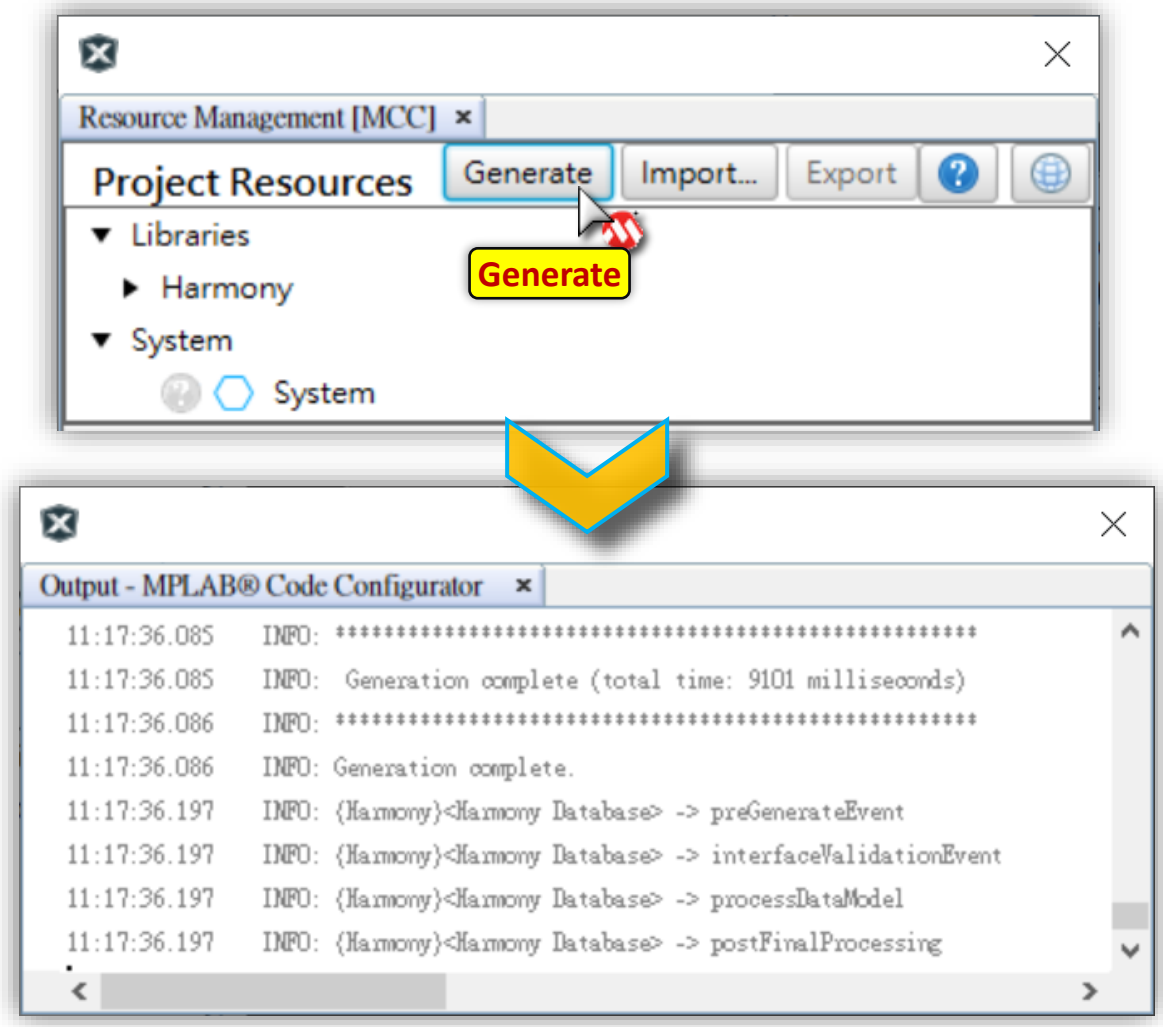
Step 1 (Use Easy View)



Lab12-1 Internal Temperature Sensor through ADC

Step 2

- Click "Generate" Button to Generate your Code.



Lab12-1 Internal Temperature Sensor through ADC

Step 3

```
...
#include <string.h>
#include <stdarg.h>
// TODO 12-1.01
#include <math.h>

uint32_t i = 0;
...
uint16_t ADC_Result;

// TODO 12-1.02
float TempR;
float TempH;
float INT1VR;
float INT1VH;
float VADCR;
float VADCH;
float TempM;

float Dec2Frac( uint8_t DecVal )
{
    if      (DecVal < 10) return ((float)DecVal/10.0);
    else if (DecVal <100) return ((float)DecVal/100.0);
    else                  return ((float)DecVal/1000.0);
}
```

Lab12-1 Internal Temperature Sensor through ADC

Step 4

```
// TODO 12-1.02
...
float TempH;
float INT1VR;
float INT1VH;
float VADCR;
float VADCH;
float TempM;

float Dec2Frac( uint8_t DecVal ) {...}

void Load_NVM_TempLogRow( void )
{
    uint8_t room_temp_val_int;
    uint8_t room_temp_val_dec;
    uint8_t hot_temp_val_int;
    uint8_t hot_temp_val_dec;
    int8_t room_int1v_val;
    int8_t hot_int1v_val;
    uint16_t ADCR;
    uint16_t ADCH;

    // Enable System Voltage Reference of Internal Temperature Sensor
    SYSCTRL_REGS->SYSCTRL_VREF|=SYSCTRL_VREF_TSEN(1);
```

SYSCTRL – System Controller

Offset	Name	Bit Pos.								
0x40	VREF	7:0						BGOUTEN	TSEN	
		15:8								
		23:16	CALIB[7:0]							
		31:24					CALIB[10:8]			

MICROCHIP

Lab12-1 Internal Temperature Sensor through ADC

Step 5

```
// TODO 12-1.02
...

void Load_NVM_TempLogRow( void )
{   ...
    SYSCTRL_REGS->SYSCTRL_VREF|=SYSCTRL_VREF_TSEN(1);

    // Load Temperature Sensor Calibration Data From NVM Fuse
    room_temp_val_int = (uint8_t)(((*(uint32_t*)TEMP_LOG_ADDR)>>0) & 0xFF);
    room_temp_val_dec = (uint8_t)(((*(uint32_t*)TEMP_LOG_ADDR)>>8) & 0xF);
    hot_temp_val_int  = (uint8_t)(((*(uint32_t*)TEMP_LOG_ADDR)>>12) & 0xFF);
    hot_temp_val_dec  = (uint8_t)(((*(uint32_t*)TEMP_LOG_ADDR)>>20) & 0xF);
    room_int1v_val    = (int8_t)(((*(uint32_t*)TEMP_LOG_ADDR)>>24) & 0xFF);
    hot_int1v_val     = (int8_t)(((*(uint32_t*)(TEMP_LOG_ADDR+4))>>0) & 0xFF);
    ADCR              = (uint16_t)(((*(uint32_t*)(TEMP_LOG_ADDR+4))>>8) & 0xFFF);
    ADCH              = (uint16_t)(((*(uint32_t*)(TEMP_LOG_ADDR+4))>>20) & 0xFFF);

    // Combination Integer and Decimal to a float type temperature value
    TempR = room_temp_val_int + Dec2Frac(room_temp_val_dec);
    TempH = hot_temp_val_int  + Dec2Frac(hot_temp_val_dec);
    INT1VR = 1 - ((float)room_int1v_val / 1000.0);
    INT1VH = 1 - ((float)hot_int1v_val / 1000.0);
    VADCR = ((float)ADCR * INT1VR) / 4095.0;
    VADCH = ((float)ADCH * INT1VH) / 4095.0;
}
```

Lab12-1 Internal Temperature Sensor through ADC

Step 6

```
// TODO 12-1.02
...
void Float2IntDec( float FloatVal, int *IntVal, int *DecVal, int DecNum ) {...}
void Load_NVM_TempLogRow( void ) { ... }

float CalculateTemperature( uint16_t ADCM )
{
    float TempC;
    float TempF;
    float VADC1V;
    float VADCM;
    float INT1V;
    float INT1VM;

    INT1V = 1.0;
    VADC1V = (( float )ADCM * INT1V) / 4095.0;
    TempC = TempR + ((VADC1V - VADCR) * (TempH - TempR) / (VADCH - VADCR));
    INT1VM = INT1VR + ((INT1VH - INT1VR) * (TempC - TempR) / (TempH - TempR));
    VADCM = (( float )ADCM * INT1VM) / 4095.0;
    TempF = TempR + ((VADCM - VADCR) * (TempH - TempR) / (VADCH - VADCR));

    return TempF;
}

void USART5_Transmit( uintptr_t context ) { ... }
```

MICROCHIP

Lab12-1 Internal Temperature Sensor through ADC

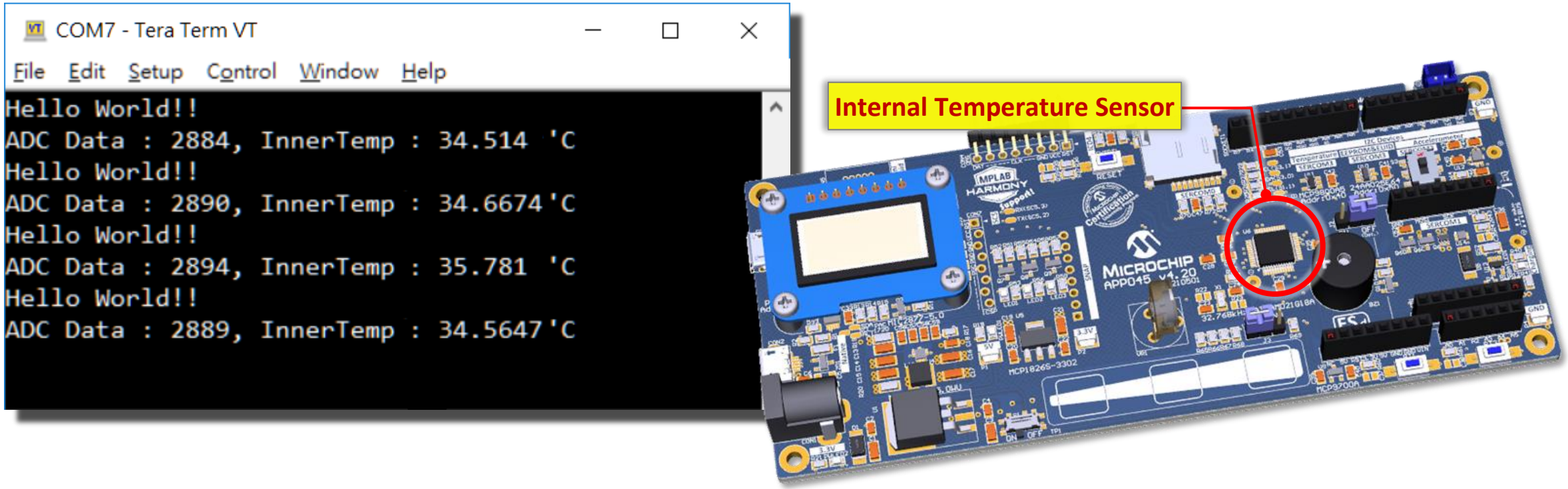
Step 7

```
...
int main ( void )
{
    ...
    SERCOM5_USART_Read( USART5_ReceiveData, 1 );

    // TODO 12-1.03
    Load_NVM_TempLogRow();

    ADC_Enable();

    while ( true )
    {
        ...
        if (TC3_HasExpired)
        {   ...
            ADC_Result = ADC_ConversionResultGet();

            // TODO 12-1.04
            TempM = CalculateTemperature( ADC_Result );
            myprintf( “ADC Data : %4d, InnerTemp : %3.4f'C\r\n", ADC_Result, TempM );
        }
        ...
    }
}
```

Lab12-1 Internal Temperature Sensor through ADC

Result

- **MCU Internal Temperature Sensor result output.**



Lab12-1 Internal Temperature Sensor through ADC

Discuss

- **Try output Temperature Log Row, Coarse and Finer result of data.**

```
#include <math.h>
void myprintf(const char *format, ...);
...
void Load_NVM_TempLogRow( void )
{   ...
    VADCH = ((float)ADCH * INT1VH) / 4095.0;

    myprintf("\033[2;1H\t\tSymbol\tRoom\tHot");
    myprintf("\033[3;1HTemperature\tTemp\t%3.2f'C\t%3.2f'C", TempR, TempH);
    myprintf("\033[4;1HVREF voltage\tINT1V\t%3.2fV\t%3.2fV", INT1VR, INT1VH);
    myprintf("\033[5;1HADC value\tADC\t%4d\t%4d", ADCR, ADCH);
    myprintf("\033[6;1HADC voltage\tVADC\t%3.2fV\t%3.2fV", VADCR, VADCH);
}

float CalculateTemperature( uint16_t ADCM )
{   ...
    TempF = TempR + ((VADCM - VADCR) * (TempH - TempR) / (VADCH - VADCR));

    myprintf("\033[8;1HINT1V =%1.4fV, VADC1V = %1.4fV, TempC = %3.2f'C", INT1V,  VADC1V, TempC);
    myprintf("\033[9;1HINT1VM=%1.4fV, VADCM  = %1.4fV, TempF = %3.2f'C", INT1VM, VADCM,  TempF);

    return TempF;
}
```

Lab12-1 Internal Temperature Sensor through ADC

Discuss

- **Try output Temperature Log Row, Coarse and Finer result of data.**

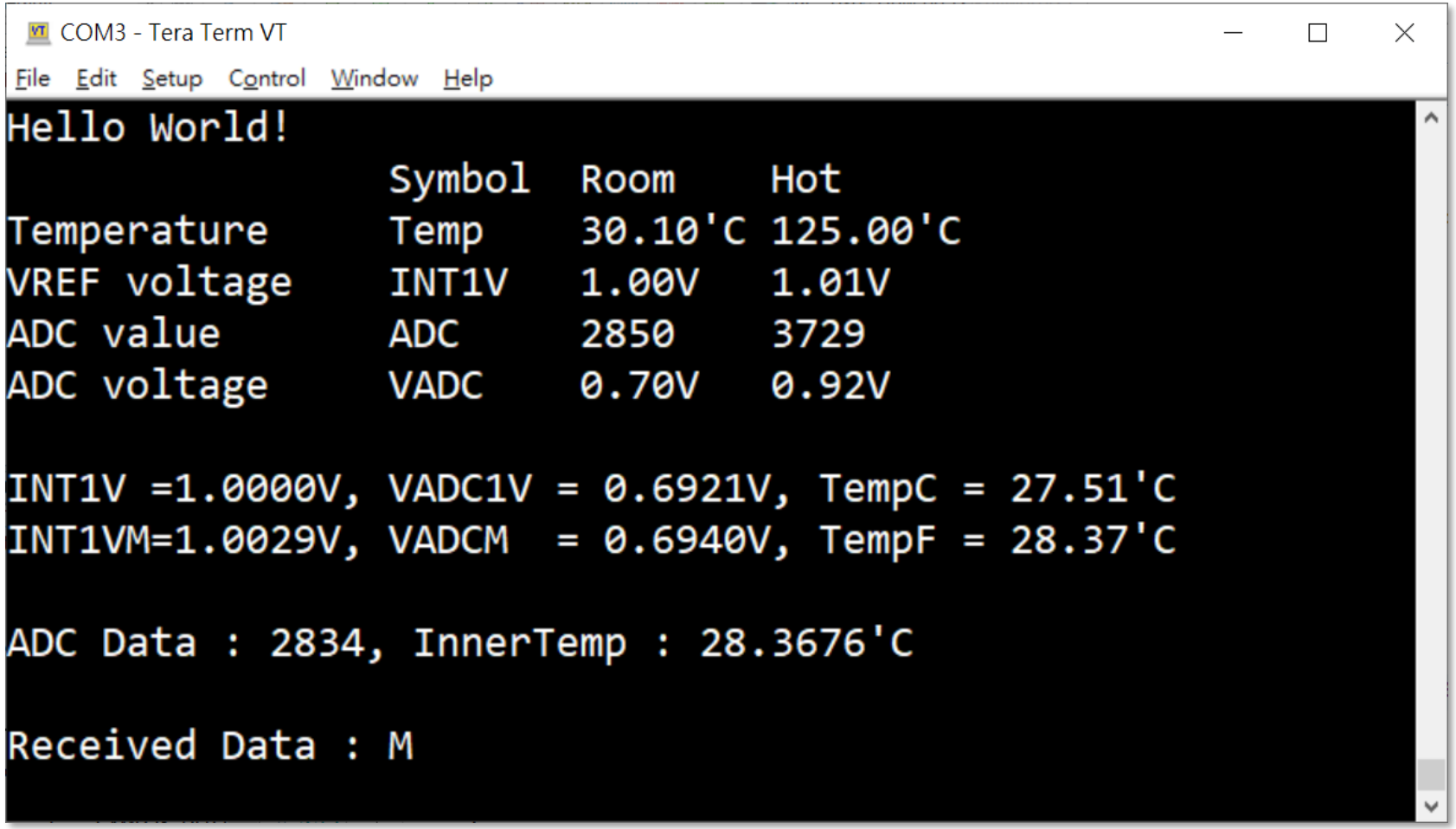
```
...
int main ( void )
{
    ...
    SERCOM5_USART_Read( USART5_ReceiveData, 1 );

    myprintf("\033[2J");    // Clear screen
    myprintf("\033[?25l");  // Hide cursor
    ...
    while ( true )
    {
        ...
        if (TC3_HasExpired)
        {
            ...
            myprintf( "\033[1;1HHello World!" );

            ...
            myprintf( "\033[11;1HADC Data : %4d, InnerTemp : %3.4f'C", ADC_Result, TempM );
        }
        ...
    }
}
```

MICROCHIP

Lab12-1 Internal Temperature Sensor through ADC

Discuss

- **Output result (Please use the console has supported VT100, for example the TeraTerm)**



Microchip Trademarks

Overview

Microchip Technology Incorporated's products are marketed and sold under one or more of the following trademarks belonging to Microchip or one of Microchip's subsidiaries. Questions regarding use of these trademarks should be addressed to the Microchip's Legal Department via email: legal.department@microchip.com.

The following are registered trademarks of Microchip in the U.S.A. and other countries:

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA

The following are registered trademarks of Microchip in the U.S.A:

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL

The following are registered trademarks of Microchip in other countries: BeaconThings, GestIC, Silicon Storage Technology

The following are trademarks of Microchip in the U.S.A. and other countries:

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA

The following is a service mark of Microchip in the U.S.A.: SQTP

The following are logos of Microchip in the U.S.A. and other countries: The Adaptec logo, Frequency on Demand, Silicon Storage Technology, Symmcom, and Trusted Time

The following is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries: GestIC

MICROCHIP