DIRECT PROGRAMMING SAMD21 REGISTERS FOR OUTPUT PWM FREQUENCY / RESOLUTION OF ARDUINO ZERO

TABLE OF CONTENTS

1. GENERAL

This paper forms the basis for the definition and the choice of the operating parameters of the Arduino Zero board to control either the PWM output frequency or the PWM output resolution of up to 8 pins. It must be noted that, with the contemporary IDE, the maximum available device output resolution is 8 Bit, because analogWriteResolution is not effective with Arduino Zero. The proposed software obtains resolutions up to and exceeding 16 Bit, trading off with output PWM frequency.

The proposed system works faultlessly with Windows XP SP3 and with Windows W7. To date no tests with Window 8 or Windows 10 have been carried out.

The system produces a high quality Dual Slope (or Phase Correct) PWM signal by using up to 3 UP/DOWN Registers or Timers.

The information herewith contained is only valid for Arduino/Genuino Zero boards equipped with IDE 1.6.0 and later, supplied by **Arduino c.c.** Initial tests with Arduino M0 and Arduino M0 PRO boards equipped with IDE 1.7.8 supplied by **Arduino Org.** were not successful. This information describes just a few of the infinite configurations made possible by the remarkable complexity and flexibility of the SAMD21 Microprocessor Architecture. The present work consists of the following documents:

a) This description in WORD or ACROBAT format.
b) Functional Blocks Diagram.
c) PRECISION PWM Excel tool.
d) Example sketches explaining the setup and use of the information provided.

The functional Blocks Diagram explains the frequency generation / PWM generation system of this particular set-up within the ATMEL SAMD21 Microprocessor that is the engine of the Arduino Zero board. This is a functional diagram, which follows the system logic and does not necessarily respect the physical interconnection of the various parts. Study of this diagram will immediately clarify the meaning of the expressions used in the sketches and give a comprehensive system picture.

2. BLOCKS DIAGRAM DESCRIPTION

NOTE: all information from the SAMD 21 Technical Spreadsheet and from ASF Source Code Documentation, available on the ATMEL site.

With reference to the block diagram, the SAMD 21 chip is equipped with the following oscillators, of which one can be selected according to the job to be implemented, as shown further on.

- XOSC32K – frequency 32.768 KHz.
- OSCULP32K – frequency 32 KHz, low power.
- OSC 8M – frequency 8 MHz.
- DFLL48M – frequency 48 MHz. Phase Locked Loop Oscillator locked to the 8 Mhz carrier.

The oscillators serve a group of 8 Generic Clock Controllers, or GCLKs. Only GCLKs 4 to 7 are available for independent use. The output of the chosen GCLK is connected to frequency scaler GENDIV, which can be set to divide the Clock frequency by any divisor **D** in the range:1 – 250 with the command:

GCLK_GENDIV_DIV(**D**)

The processed Clock signal is then further divided by Prescaler CTRLA, which has a default setting of 16. CTRLA possible Prescaler settings are: **N** = 1, 2, 4, 8, 16, 64, 256 and 1024 and can be set with the command:

TCC_CTRLA_PRESCALER_DIV(**N**)

Prescaler's output drives the three U/D Registers TCC0 (24Bit), TCC1 (24Bit) & TCC2 (16Bit). They are programmed with the PER parameter and count UP to the PER value, then down to zero and so on: in this mode of operation the PER parameter is used TWICE, hence we have an additional /2 factor. The value of the PER parameter is crucial for setting either the PWM output resolution, or the PWM output frequency or finding a good compromise between them. By Divisor and Prescaler manipulation a very large number of solutions is possible, as will be seen further on. The PER parameter setting command is:

REG_TCCX_PER = **K** (with X = 0, 1, 2)

We now need to connect the outputs of the Timers to the output pins by means of a Multiplexer. We do this in two steps:

a) first we enable the Multiplexer with the PORT MULTIPLEXER ENABLE expression shown in the blocks diagram and
b) second we connect the Timers to the outputs with the CONNECT THE TCCX TIMERS TO THE OUTPUTS expression, shown in the blocks diagram.

Output pins are paired in EVEN and ODD numbers, as shown both in the above expression and in the outputs with different colours.

The COMPARE CHANNEL BUFFERS CCBX set the PWM percentage or duration **either** by reading an integer from zero (PWM 0% D.C.) to PER (PWM 100% D.C.) set with the command below, **or** by reading voltage values on the INPUT PINS (A0 to A5) **or** by reading data received through Serial or other COMMS. The command is:

REG_TCCX_CCB**W**, where W = 0, 1, 2, & 3.

The PWM capable output pins are:

ARDUINO	**REGISTERS**	**SAMD 21**
D2	TCC0 CCB0	Chip Pin PA8 (even)
D3	TCC1 CCB1	Chip Pin PA9 (odd)
D4	TCC1 CCB0	Chip Pin PA14 (even)
D5	TCC0 CCB1	Chip Pin PA15 (odd)
D6	TCC0 CCB2	Chip Pin PA20 (even)
D7	TCC0 CCB3	Chip Pin PA21 (odd)
D11	TCC2 CCB0	Chip Pin PA16 (even)
D13	TCC2 CCB1	Chip Pin PA17 (odd)

The Arduino Zero board is therefore capable of up to 8 Multi-Frequency / Multi-Resolution PWM outputs. The following pins are free for other uses: for example pins D8 & D9 can be used as inputs when D11 and D13 are used as outputs.

D0 & D1: Serial Comms.
D8, D9,D10, D12

3. SYSTEM MATHEMATICS & THE PPWM TOOL

The Excel file PPWM TOOL allow easy and quick interactive selection of the system operating parameters based on PWM requirements needed. Math. expressions are as follows.

Resolution (Bits) = R
Resolution (Ratio) = PER = $2^{\wedge R} - 1$
Example 1: R = 10, V = $2^{\wedge 10} - 1 = 1024 - 1 = 1023$

Clock Frequency GCLK
Clock Divisor GENDIV, D
Prescaler Factor CTRLA, N
Timer frequency, Ft = GCLK/(D*N)

Example 2: Clock = 48 Mhz = 48.000.000 Hz; D = 3 ; N = 16;
Timer frequency Ft = 48.000.000/(3*16) = 1.000.000 Hz = 1 MHz.

PWM frequency, Fpwm = Ft/(2*PER)
Example 3: Fpwm = 1.000.000/(2*1023) = 488.76 Hz.

Fractional resolution (Bits) = R = (LOG_{10} (PER + 1) / LOG_{10} (2)
Example 4: PER = 1728; R = (LOG (1728 + 1) / 0,301030 = 3,237795/0,301030 = 10, 756 Bits

The PPWM tool allows interactive selection and calculation of parameters to achieve the required result. The top section uses the **Resolution** as the **Main Variable**, while the bottom section uses the **Frequency** as the **Main Variable**. **Data must be written in the DATA column <u>only</u> and results are given in the CALC column.**

Example 5
We need to generate PWM with exact 12 Bit resolution (main requirement) and PWM must have a frequency in the order of low audio, below 500 Hz, because it must drive low frequency Optocouplers. In the top section of the PPWM Tool we enter in the **DATA COLUMN:**
Resolution: 12Bit Tool calculates required PER = 4095
GENDIV D : 1 and
Prescaler N : 8 and
Clock Frequency: 48.000.000 Hz Tool calculates: Timer frequency Ft: 6.000.000Hz (6 MHz) and
PWM frequency Fpwm: 732,60 Hz.
This value of Fpwm is too high, so we change N from 8 to 16 and obtain Fpwm: 366,30 which is in the ballpark.

In general the RESOLUTION section is used when we can compromise on PWM frequency, but we must have a **non-fractional** resolution value, because the PWM must be converted into a precision D.C. voltage by lowpass filtering.

When we do not have the above requirement, but we need a specific PWM frequency, then the same mathematical expressions are used in the PWM Frequency (lower) section of the PPWM Tool. Of course, even in this case we shall manipulate parameters in order to maximise resolution as much as possible.

EXAMPLE 6

We need to have a PWM output of exactly 17 KHz. We enter in the DATA column:
PWM frequency: 17 KHz = 17.000 Hz
GENDIV, D: 1
Prescaler N: 4 with this data we end up with a PER = 347, a fractional resolution of 8.47 and a rounded resolution of 8 Bit.
To maximize the resolution we have to further reduce prescaling, so we enter N = 1. Now PER climbs to 1411, fractional resolution to 10,46 and rounded resolution to 10.

EXAMPLE 7

After testing, we realize that frequency must be changed to precisely 17.307 HZ. Entering this frequency into The PPWM Tool, the new PER value is: 1387 and the effect on resolution is negligible.

4. SKETCHES GENERAL

The sketches follow the structure outlined by the functional Blocks Diagram and work as soon as uploaded to Arduino Zero after all parameters (obtained by the PPWM Tool) are entered. Up to now the REFERENCE VOLTAGE has never been mentioned, but this parameter is essential to define the relationship between Arduino Zero inputs and PWM outputs. The standard Reference Voltage commands do not work and must be replaced as follows.

In place of AR_EXTERNAL:

```
ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val;      // Gain Factor Selection
ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_AREFA_Val;
```

In place of AR_INTERNAL1V0:

```
ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val;      // Gain Factor Selection
ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_INT1V_Val;   // 1.0V voltage reference
```

In place of AR_INTERNAL1V65:

```
ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val;      // Gain Factor Selection
ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_INTVCC1_Val; // 1/2 VDDANA = 0.5* 3V3 = 1.65V
```

In place of AR_INTERNAL2V23:

```
ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val;      // Gain Factor Selection
ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_INTVCC0_Val;
// 1/1.48 VDDANA = 1/1.48* 3V3 = 2.2297.  If driving PWM the corr. Factor  3300/12229.7 =1.48 must be used.
```

Please note that the system defaults to 3.3V Reference if one of the above expressions is not used. Also the default Reference of 1.65V is half the Arduino Zero operating voltage of 3.3 V.
Remember maximum voltage on any pin cannot exceed 3V and this rule also holds for the AREF pin!

PARAMETERS TO BE ENTERED
The following list only concerns the parameters required to PWM System set – up. The numbers shown are for clarification only

- int PERX = **1023**; //Resolution Parameter PER calculated by the PPWM tool.
- AR_INTERNAL**2V23**; //Sets Voltage Reference at 2230 mV.
- REG_GCLK_GENDIV = GCLK_GENDIV_DIV**(D)** | // Generic clock Divisor: calculated by PPWM tool.
- GCLK_GENDIV_ID**(4)**; // Select Generic Clock (GCLK) 4.
- GCLK_GENCTRL_SRC_**DFLL48M** | // clock source selected by PPWM.
- GCLK_GENCTRL_ID**(4)**; // Select GCLK4.
- const uint8_t CHANNELS = **6**;
- const uint8_t pwmPins[] = { **2, 3, 4, 5, 6, 7** };
- GCLK_CLKCTRL_GEN_GCLK**4** | // Select GCLK4
- REG_TCC0_CTRLA |= TCC_CTRLA_PRESCALER_DIV**16** | // Prescaler Factor from the PPWM tool.
- REG_TCC1_CTRLA |= TCC_CTRLA_PRESCALER_DIV**16** | // Prescaler Factor from the PPWM tool.

5. PRECISION VOLTAGE TRANSLATOR VREF. = 2229.7 mV

SYSTEM REQUIREMENTS

Most data acquisition equipment accepts analogue data from analogue output sensors in the range 0-1 V, 0-2V or 0-2.5V. DA Logger resolution is normally 8 to 12 Bit. The Arduino Zero Voltage Translator/DA Buffer accepts either analogue or digital sensor outputs and translates them to a voltage range suitable for the DA Logger input while providing galvanic isolation between inputs and outputs. Galvanic isolation is obtained by converting the generic input Data to a low frequency (500Hz approx.) PWM signal. The PWM drives an optocoupler (thereby providing input/output isolation) and the optocoupler output is processed by low pass filtering to recover the original data for feeding the DA Logger's inputs.

DATA CONVERSION & MANIPULATION

The sketch must keep track of input and output resolution. With:

```
int PERX = 1023;                //Resolution Parameter PER calculated by the PPWM tool.
int BitX = log(PERX+1)/log(2);  // Bits of PER parameter.
```

We compute the bit related to the calculated & required resolution PERX. With:

```
analogReadResolution(BitX);     //Sets the INPUT resolution equal to the OUTPUT resolution.
```

This can be done within limits, as maximum AnalogReadResolution is 12 Bits. With:

```
float Vz_0 =(Vy_0 * Vref./PERX);
```

we recover the analogue voltage . With Serial.print we show both the bits value and the analogue channel value. With the expression:

```
REG_TCC0_CCB0 = corr. factor *Vy_0;         //Output pin D2  e.g.  corr. factor = 3300/2229.7 = 1.48
 while(TCC0->SYNCBUSY.bit.CCB0);
```

we modulate the PWM output with the value read on pin A0. In other words the input of the PWM compare register is driven by our original analogue voltage input. It is important to remember that, by changing the value of PER, the Arduino Zero will follow with the new resolution, but, at the same time, the PWM frequency will change accordingly, while the duty cycle will stay the same.

Here below is copy of the PPWM Tool results.

DESCRIPTION	DATA	CALC
RESOLUTION (Analogue output function)	**10,00**	
Dual Slope U/D count factor PER		1.023,00
Clock Divisor, GENDIV	3,00	
Prescaler factor CTRLA, N	16,00	
GLK Clock frequency, Hz	48.000.000,00	
Timer frequency - Ft, Hz		1.000.000,00
PWM frequency - Fpwm, Hz		**488,76**
CCBx load for 50% duty cycle		511,50

```
/*  G. LOVISOLO - "glovisol" - Work based on original research by MartinL whose expertise, help and
   guidance is gratefully acknowledged. This sketch in the Public Domain.
    VER. 8.0 - 02.04.2016--- GENUINO ZERO (c.c.) LOW FREQUENCY PRECISION VOLTAGE TRANSLATOR SKETCH.
   Provides high resolution variable duty cycle square-wave proportional to input voltage. PWM freq. = 488Hz at 10Bit Res.
   ANALOG INPUTS: A0, A1, A2, A3, A4, A5; ANALOG OUTPUTS: D2, D3, D4, D5, D6, D7. Resolution adjusted
   by REG_TCCX_PER (X can be 0 or 1)and Duty Cycle adjusted by  REG_TCCX_CCBY (Y can be 1,2 or 3).
   INPUT VOLTAGE RANGE: 0 - 2290 mV. DUTY CYCLE ALL OUTPUT PINS: 0 - 70%. OUTPUT VOLTAGE RANGE
   AFTER FILTERING: 0-2290 mV. Operating parameters & resolution calculated by Excel "PRECISION PWM" - "PPWM" tool.
   Please see Description & Functional blocks diagram.  Changing Ref. Voltage, other input/output voltage ranges possible  */

//DECLARE CONSTANTS
#define Pin13LED   13
//DECLARE VARIABLES
int (timex) = 1000;              //Iteration time.
int sensorValue_0 = A0;          //Analogue input.
int Vy_0 = 0;                    //Bits reading of analogue input.
int Vz_0 = 0;                    //Analogue voltage output.
int sensorValue_1 = A1;
int Vy_1 = 0;
int Vz_1 = 0;
int sensorValue_2 = A2;
int Vy_2 = 0;
int Vz_2 = 0;
int sensorValue_3 = A3;
int Vy_3 = 0;
int Vz_3 = 0;
int sensorValue_4 = A4;
int Vy_4 = 0;
int Vz_4 = 0;
int sensorValue_5 = A5;
int Vy_5 = 0;
int Vz_5 = 0;
int CalX = 2229.7;                      //Analogue calibration factor.
int PERX = 1023;                        //Resolution Parameter PER calculated by the PPWM tool.
int BitX = log(PERX+1)/log(2);    // Bits of PER parameter.

//SETTING UP
void setup()
{
//Sets internal reference voltage at 2229.7 mV.
ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val;      // Gain Factor Selection
   ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_INTVCC0_Val;
// 1/1.48 VDDANA = 1/1.48* 3V3 = 2.2297.  The 1.48 =3300/2229.7 Correction Factor must be used in the PWM below.

 REG_GCLK_GENDIV = GCLK_GENDIV_DIV(3) |      // Generic clock Divisor: calculated by PPWM tool.
              GCLK_GENDIV_ID(4);                     // Select Generic Clock (GCLK) 4.
 while (GCLK->STATUS.bit.SYNCBUSY);               // Wait for synchronization.

 REG_GCLK_GENCTRL = GCLK_GENCTRL_IDC |       // Set the duty cycle to 50/50 HIGH/LOW.
              GCLK_GENCTRL_GENEN |                 // Enable GCLK4.
              GCLK_GENCTRL_SRC_DFLL48M |         // 48MHz clock source selected by PPWM.
              GCLK_GENCTRL_ID(4);                     // Select GCLK4.
 while (GCLK->STATUS.bit.SYNCBUSY);                // Wait for synchronization.
 // Enable the port multiplexer for the 6 PWM channels: timer TCC0 & TCC1 outputs.
 const uint8_t CHANNELS = 6;
 const uint8_t pwmPins[] = { 2, 3, 4, 5, 6, 7 };
 for (uint8_t i = 0; i < CHANNELS; i++)
 {
  PORT->Group[g_APinDescription[pwmPins[i]].ulPort].PINCFG[g_APinDescription[pwmPins[i]].ulPin].bit.PMUXEN = 1;
 }
 // Connect the TCC0 timer to the port outputs - port pins are paired odd PMUXO and even PMUXE
 // F & E specify the timers: TCC0, TCC1 and TCC2
 PORT->Group[g_APinDescription[2].ulPort].PMUX[g_APinDescription[2].ulPin >> 1].reg = PORT_PMUX_PMUXO_F |
PORT_PMUX_PMUXE_F;
 PORT->Group[g_APinDescription[4].ulPort].PMUX[g_APinDescription[4].ulPin >> 1].reg = PORT_PMUX_PMUXO_F |
PORT_PMUX_PMUXE_F;
 PORT->Group[g_APinDescription[6].ulPort].PMUX[g_APinDescription[6].ulPin >> 1].reg = PORT_PMUX_PMUXO_F |
PORT_PMUX_PMUXE_F;


 // Feed GCLK4 to TCC0 and TCC1
 REG_GCLK_CLKCTRL = GCLK_CLKCTRL_CLKEN |         // Enable GCLK4 to TCC0 and TCC1
                    GCLK_CLKCTRL_GEN_GCLK4 |     // Select GCLK4
                    GCLK_CLKCTRL_ID_TCC0_TCC1;   // Feed GCLK4 to TCC0 and TCC1
 while (GCLK->STATUS.bit.SYNCBUSY);              // Wait for synchronization

 // Dual slope PWM operation: timers continuously count up to PER register value then down 0
 REG_TCC0_WAVE |= TCC_WAVE_POL(0xF) |            // Reverse the output polarity on all TCC0 outputs
                  TCC_WAVE_WAVEGEN_DSBOTTOM;     // Setup dual slope PWM on TCC0
 while (TCC0->SYNCBUSY.bit.WAVE);                // Wait for synchronization

 REG_TCC1_WAVE |= TCC_WAVE_POL(0xF) |            // Reverse the output polarity on all TCC1 outputs
                  TCC_WAVE_WAVEGEN_DSBOTTOM;     // Setup dual slope PWM on TCC1
 while (TCC1->SYNCBUSY.bit.WAVE);                // Wait for synchronization

 // Each timer counts up to TOP value set by the PER register, then down to zero.
 // PER parameter value selected by PPWM tool for non-decimal, precision Resolution figure.
 REG_TCC0_PER = PERX;                            //Resolution for TCC0 Register.
 while(TCC0->SYNCBUSY.bit.PER);
 REG_TCC1_PER = PERX;                            //Resolution for TCC1 Register.
 while(TCC1->SYNCBUSY.bit.PER);

 // The CCBx register value corresponds to the pulse width in microseconds (us)
 REG_TCC0_CCB0 = (PERX/2);      // TCC0 CCB0 - 50% PWM on D2
 while(TCC0->SYNCBUSY.bit.CCB0);
 REG_TCC1_CCB1 = (PERX/2);      // TCC1 CCB1 - 50% PWM on D3
 while(TCC1->SYNCBUSY.bit.CCB1);
 REG_TCC1_CCB0 = (PERX/2);      // TCC1 CCB0 - 50% PWM on D4
 while(TCC1->SYNCBUSY.bit.CCB0);
 REG_TCC0_CCB1 = (PERX/2);      // TCC0 CCB1 - 50% PWM on D5
 while(TCC0->SYNCBUSY.bit.CCB1);
 REG_TCC0_CCB2 = (PERX/2);      // TCC0 CCB2 - 50% PWM on D6
 while(TCC0->SYNCBUSY.bit.CCB2);
 REG_TCC0_CCB3 = (PERX/2);      // TCC0 CCB3 - 50% PWM on D7
 while(TCC0->SYNCBUSY.bit.CCB3);
 // Divide the Divisor processed Clock signal by the Prescaler factor and enable the outputs.
 REG_TCC0_CTRLA |= TCC_CTRLA_PRESCALER_DIV16 |    // Prescaler Factor from the PPWM tool.
                 TCC_CTRLA_ENABLE;                 // Enable the TCC0 output.
 while (TCC0->SYNCBUSY.bit.ENABLE);               // Wait for synchronization.

 REG_TCC1_CTRLA |= TCC_CTRLA_PRESCALER_DIV16 |  // Prescaler Factor from the PPWM tool.
                 TCC_CTRLA_ENABLE;               // Enable the TCC0 output.
 while (TCC1->SYNCBUSY.bit.ENABLE);             // Wait for synchronization.

 pinMode(Pin13LED, OUTPUT);                      // Sets signaling pin 13 as output.
 delay(50);
}

//OPERATION
void loop()
{
digitalWrite(Pin13LED, HIGH);
Serial.begin(9600);
analogReadResolution(BitX);         //Sets the INPUT resolution equal to the OUTPUT resolution.

Vy_0 = analogRead(sensorValue_0);
float Vz_0 =(Vy_0 * 3300/PERX);
Serial.print("Channel/0, Bits = ");
Serial.print(Vy_0);
Serial.print(" and mV = ");
Serial.println(Vz_0);
REG_TCC0_CCB0 = (Vy_0/1.48);              //Output pin D2. Note corr. Factor 1.48 for Internal ref. 2229.7.
while(TCC0->SYNCBUSY.bit.CCB0);

Vy_1 = analogRead(sensorValue_1);
float Vz_1 =(Vy_1 * 3300/PERX);
Serial.print("Channel/1, Bits = ");
Serial.print(Vy_1);
Serial.print(" and mV = ");
Serial.println(Vz_1);
REG_TCC1_CCB1 = (Vy_1/1.48);              //Output pin D3
while(TCC1->SYNCBUSY.bit.CCB1);

Vy_2 = analogRead(sensorValue_2);
float Vz_2 =(Vy_2 * 3300/PERX);
Serial.print("Channel/2, Bits = ");
Serial.print(Vy_2);
Serial.print(" and mV = ");
Serial.println(Vz_2);
REG_TCC1_CCB0 = (Vy_2/1.48);              //Output Pin D4
while(TCC1->SYNCBUSY.bit.CCB0);
```

```
 Vy_3 = analogRead(sensorValue_3);
 float Vz_3 =(Vy_3 * 3300/PERX);
 Serial.print("Channel/3, Bits = ");
 Serial.print(Vy_3);
 Serial.print(" and mV = ");
 Serial.println(Vz_3);
 REG_TCC0_CCB1 = (Vy_3/1.48);              //OUTPUT pin D5
 while(TCC0->SYNCBUSY.bit.CCB1);

 Vy_4 = analogRead(sensorValue_4);
 float Vz_4 =(Vy_4 * 3300/PERX);
 Serial.print("Channel/4, Bits = ");
 Serial.print(Vy_4);
 Serial.print(" and mV = ");
 Serial.println(Vz_4);
 REG_TCC0_CCB2 = (Vy_4/1.48);              //OUTPUT pin D6
 while(TCC0->SYNCBUSY.bit.CCB2);

 Vy_5 = analogRead(sensorValue_5);
 float Vz_5 =(Vy_5 * 3300/PERX);
 Serial.print("Channel/5, Bits = ");
 Serial.print(Vy_5);
 Serial.print(" and mV = ");
 Serial.println(Vz_5);
 Serial.println("...................");
 REG_TCC0_CCB3 = (Vy_5/1.48);              //OUTPUT pin D7
 while(TCC0->SYNCBUSY.bit.CCB3);

 digitalWrite(Pin13LED, LOW);
 delay(timex);


}//End of  Iteration – Iteration re-starts
```

6. PRECISION VOLTAGE TRANSLATOR VREF. = 2500 mV

```
/*  G. LOVISOLO - "glovisol" - Work based on original research by MartinL whose expertise, help and
   guidance is gratefully acknowledged. This sketch in the Public Domain.

   VER. 9.0 - 05.04.2016--- GENUINO ZERO (c.c.) LOW FREQUENCY PRECISION VOLTAGE TRANSLATOR SKETCH.
   Provides high resolution variable duty cycle square-wave proportional to input voltage. Resolution
   can be: 8, 10, 12 Bit. Changing Resolution affects output frequency which can be adjusted with
   Divisor D and Prescaler Factor, as calculated by Ppwm tool. For Res. = 12 Bit , PWM Freq. = 122 Hz.
   ANALOG INPUTS: A0, A1, A2, A3, A4, A5; ANALOG OUTPUTS: D2, D3, D4, D5, D6, D7. Resolution adjusted
   by REG_TCCX_PER (X can be 0 or 1)and Duty Cycle adjusted by  REG_TCCX_CCBY (Y can be 1,2 or 3).
   INPUT VOLTAGE RANGE: 0 - 2500 mV. DUTY CYCLE ALL OUTPUT PINS: 0 - 75%. OUTPUT VOLTAGE RANGE
   AFTER FILTERING: 0-2500 mV. Operating parameters & resolution calculated by Excel
   "PRECISION PWM" - "PPWM" tool. Please see Description & Functional blocks diagram.
   Changing Ref. Voltage, AR_EXTERNAL other input/output voltage ranges possible  */

//DECLARE CONSTANTS
#define Pin13LED   13

//DECLARE VARIABLES
int (timex) = 2000;            //Iteration time.
int sensorValue_0 = A0;      //Analogue input.
int Vy_0 = 0;                  //Bits reading of analogue input.
int Vz_0 = 0;                  //Analogue voltage output.
int sensorValue_1 = A1;
int Vy_1 = 0;
int Vz_1 = 0;
int sensorValue_2 = A2;
int Vy_2 = 0;
int Vz_2 = 0;
int sensorValue_3 = A3;
int Vy_3 = 0;
int Vz_3 = 0;
int sensorValue_4 = A4;
int Vy_4 = 0;
int Vz_4 = 0;
int sensorValue_5 = A5;
int Vy_5 = 0;
int Vz_5 = 0;
int CalX = 2500;                    //Reference voltage & Analogue calibration factor.
int PERX = 4096;                   //Resolution Parameter PER calculated by the PPWM tool.
int BitX = log(PERX+1)/log(2);   // Bits of PER parameter.

//SETTING UP
void setup()
{
 //Setting the INTERNAL Reference Voltage @ 2500 mV.
 ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val;            // Gain Factor Selection
    ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_AREFA_Val;       // EXTERNAL VREF.
 REG_GCLK_GENDIV = GCLK_GENDIV_DIV(3) |          // Generic clock Divisor: calculated by PPWM tool.
                   GCLK_GENDIV_ID(4);            // Select Generic Clock (GCLK) 4.
 while (GCLK->STATUS.bit.SYNCBUSY);              // Wait for synchronization.

 REG_GCLK_GENCTRL = GCLK_GENCTRL_IDC |           // Set the duty cycle to 50/50 HIGH/LOW.
          GCLK_GENCTRL_GENEN |                   // Enable GCLK4.
          GCLK_GENCTRL_SRC_DFLL48M |             // 48MHz clock source selected by PPWM.
          GCLK_GENCTRL_ID(4);                    // Select GCLK4.
 while (GCLK->STATUS.bit.SYNCBUSY);              // Wait for synchronization.

 // Enable the port multiplexer for the 6 PWM channels: timer TCC0 & TCC1 outputs.
 const uint8_t CHANNELS = 6;
 const uint8_t pwmPins[] = { 2, 3, 4, 5, 6, 7 };
 for (uint8_t i = 0; i < CHANNELS; i++)
 {
  PORT->Group[g_APinDescription[pwmPins[i]].ulPort].PINCFG[g_APinDescription[pwmPins[i]].ulPin].bit.PMUXEN = 1;
 }
 // Connect the TCC0 timer to the port outputs - port pins are paired odd PMUXO and even PMUXE
 // F & E specify the timers: TCC0, TCC1 and TCC2
 PORT->Group[g_APinDescription[2].ulPort].PMUX[g_APinDescription[2].ulPin >> 1].reg = PORT_PMUX_PMUXO_F |
PORT_PMUX_PMUXE_F;
 PORT->Group[g_APinDescription[4].ulPort].PMUX[g_APinDescription[4].ulPin >> 1].reg = PORT_PMUX_PMUXO_F |
PORT_PMUX_PMUXE_F;
 PORT->Group[g_APinDescription[6].ulPort].PMUX[g_APinDescription[6].ulPin >> 1].reg = PORT_PMUX_PMUXO_F |
PORT_PMUX_PMUXE_F;

 // Feed GCLK4 to TCC0 and TCC1
 REG_GCLK_CLKCTRL = GCLK_CLKCTRL_CLKEN |                // Enable GCLK4 to TCC0 and TCC1
                    GCLK_CLKCTRL_GEN_GCLK4 |            // Select GCLK4
                    GCLK_CLKCTRL_ID_TCC0_TCC1;          // Feed GCLK4 to TCC0 and TCC1
 while (GCLK->STATUS.bit.SYNCBUSY);                     // Wait for synchronization

 // Dual slope PWM operation: timers continuously count up to PER register value then down 0.
 REG_TCC0_WAVE |= TCC_WAVE_POL(0xF) |                          // Reverse the output polarity on all TCC0 outputs
                  TCC_WAVE_WAVEGEN_DSBOTTOM;                   // Setup dual slope PWM on TCC0
 while (TCC0->SYNCBUSY.bit.WAVE);                              // Wait for synchronization

 REG_TCC1_WAVE |= TCC_WAVE_POL(0xF) |                          // Reverse the output polarity on all TCC1 outputs
                  TCC_WAVE_WAVEGEN_DSBOTTOM;                   // Setup dual slope PWM on TCC1
 while (TCC1->SYNCBUSY.bit.WAVE);                              // Wait for synchronization

 // Each timer counts up to TOP value set by the PER register, then down to zero.
 // PER parameter value selected by PPWM tool for non-decimal, Precision Resolution figure.
 REG_TCC0_PER = PERX;              //Resolution for TCC0 Register.
 while(TCC0->SYNCBUSY.bit.PER);
 REG_TCC1_PER = PERX;              //Resolution for TCC1 Register.
 while(TCC1->SYNCBUSY.bit.PER);

 // Divide the Divisor processed Clock signal by the Prescaler factor and enable the outputs.
 REG_TCC0_CTRLA |= TCC_CTRLA_PRESCALER_DIV16 |             // Prescaler Factor from the PPWM tool.
                   TCC_CTRLA_ENABLE;                       // Enable the TCC0 output.
 while (TCC0->SYNCBUSY.bit.ENABLE);                        // Wait for synchronization.
 REG_TCC1_CTRLA |= TCC_CTRLA_PRESCALER_DIV16 |                 // Prescaler Factor from the PPWM tool.
                            TCC_CTRLA_ENABLE;                          // Enable the TCC0 output.
 while (TCC1->SYNCBUSY.bit.ENABLE);                            // Wait for synchronization.

 pinMode(Pin13LED, OUTPUT); // Sets signaling pin 13 as output.
 delay(50);
}

//OPERATION
void loop()
{
 digitalWrite(Pin13LED, HIGH);
 Serial.begin(9600);
 analogReadResolution(BitX);                          //Sets the INPUT resolution equal to the OUTPUT resolution.

 Vy_0 = analogRead(sensorValue_0);
 float Vz_0 =(Vy_0 * CalX/PERX);
 Serial.print("Channel/0, Bits = ");
 Serial.print(Vy_0);
 Serial.print(" and mV = ");
 Serial.println(Vz_0);
 REG_TCC0_CCB0 = (Vy_0*CalX/3300);                //Output pin D2
 while(TCC0->SYNCBUSY.bit.CCB0);

 Vy_1 = analogRead(sensorValue_1);
 float Vz_1 =(Vy_1 * CalX/PERX);
 Serial.print("Channel/1, Bits = ");
 Serial.print(Vy_1);
 Serial.print(" and mV = ");
 Serial.println(Vz_1);
 REG_TCC1_CCB1 = (Vy_1*CalX/3300);                //Output pin D3
 while(TCC1->SYNCBUSY.bit.CCB1);

 Vy_2 = analogRead(sensorValue_2);
 float Vz_2 =(Vy_2 * CalX/PERX);
 Serial.print("Channel/2, Bits = ");
 Serial.print(Vy_2);
 Serial.print(" and mV = ");
 Serial.println(Vz_2);
 REG_TCC1_CCB0 = (Vy_2*CalX/3300);                //Output Pin D4
 while(TCC1->SYNCBUSY.bit.CCB0);

 Vy_3 = analogRead(sensorValue_3);
 float Vz_3 =(Vy_3 * CalX/PERX);
 Serial.print("Channel/3, Bits = ");
 Serial.print(Vy_3);
 Serial.print(" and mV = ");
 Serial.println(Vz_3);
 REG_TCC0_CCB1 = (Vy_3*CalX/3300);                 //OUTPUT pin D5
 while(TCC0->SYNCBUSY.bit.CCB1);
 Vy_4 = analogRead(sensorValue_4);
 float Vz_4 =(Vy_4 * CalX/PERX);
 Serial.print("Channel/4, Bits = ");
 Serial.print(Vy_4);
 Serial.print(" and mV = ");
 Serial.println(Vz_4);
 REG_TCC0_CCB2 = (Vy_4*CalX/3300);                    //OUTPUT pin D6
 while(TCC0->SYNCBUSY.bit.CCB2);

 Vy_5 = analogRead(sensorValue_5);
 float Vz_5 =(Vy_5 * CalX/PERX);
 Serial.print("Channel/5, Bits = ");
 Serial.print(Vy_5);
 Serial.print(" and mV = ");
 Serial.println(Vz_5*CalX/3300);
 Serial.println("...................");
 REG_TCC0_CCB3 = (Vy_5*CalX/3300);                    //OUTPUT pin D7
 while(TCC0->SYNCBUSY.bit.CCB3);

 digitalWrite(Pin13LED, LOW);
 delay(timex);

}//End of  Iteration – Iteration re-starts
```