List of commands (public functions) of the INA226_WE library

Function	Parameters	what it does
void Init( )	none	initiates the INA226 with some default register values
void reset_INA226( )	none	reset of the device
void getI2cErrorCode( )	none	returns the current error code from endTransmission(); 0 = success.
void setCorrectionFactor(*factor*)	factor (float)	if INA226 current values differ from currents measured with calibrated equipment, you can apply a factor
void setAverage(*mode*)	AVERAGE_*X* *X* = 1, 4, 16, 64, 128, 256, 512, 1024	sets the number of samples that are averaged for one measurement
void setConversionTime(*time*)	CONV_TIME_*X* *X* = 140, 204, 332, 588, 1100, 2116, 4156, 8244	sets time for conversion for shunt and bus voltage in microseconds
void setMeasureMode(*mode*)	CONTINUOUS, TRIGGERED, POWER_DOWN	sets the mode; for POWER_DOWN please use the powerDown function since it remembers the mode before power-down
void setCurrentRange(*range*)	MA_800, MA_400	sets the current range in mA
void setResistorRange(*resistorValue, range*)	resistorValue in Ohm (float), range in Ampere (float)	Sets resistor value in case you don't use the INA226 module with 0.1 Ohm. Don't use setCurrentRange if you are using this function.
float getShuntVoltage_mV( )	none	delivers shunt voltage in mV
float getBusVoltage( )	none	delivers bus voltage in V
float getCurrent_mA( )	none	delivers current in mA
float getBusPower_mW( )	none	delivers the power in mW
void startSingleMeasurement( )	none	starts single shot measurement and waits until data is available
void powerDown( )	none	switches the module off and saves the configuration before
void powerUp( )	none	switches the module on after Power Down and writes back the configuration (modes, gains, etc)
void waitUntilConversionCompleted( )	none	waits until the current conversions and calculations are completed.
void setAlertPinActiveHigh( )	none	by default the the alert pin is active-low; this function changes this
void enableAlertLatch( )	none	the alert flag is set and the alert pin is active, when the limit in the alert register is exceeded; by default it will be deleted with the next measurement in limit; with enableAlertLatch the flag will have to be cleared manually, which gives better control
void setAlertType(*type, limit*)	types: SHUNT_UNDER, SHUNT_OVER, BUS_UNDER, BUS_OVER, CURRENT_UNDER, CURRENT_OVER, POWER_OVER limit: float	sets the alert type and the limit: SHUNT_OVER/_UNDER: limit in mV BUS_OVER / _UNDER: limit in V CURRENT_OVER / _UNDER: limit in mA POWER_OVER: limit in mW
void readAndClearFlags( )	none	reads the Mask/Enable register; this clears the overflow, conversion ready and limit alert flags; the status of the flags are saved in the following bool variables: - overflow - convAlert - limitAlert