Method to automatic calibrate ULS24 integration time

Overview

In many applications, the light emission level of the target analyte may vary widely from sample to sample. This may be due to different chemistry can be used. As a result, the base signal intensity and full dynamic range of the signal output throughout the reaction protocol maybe widely different.

In this case, there is a need to adjust the sensitivity control parameters of ULS24 device, including the integration time so that the signal strength and the sensor dynamic range would match each other.

In this application note, we explain the methods to perform automatic adjustment of sensor integration time in a few practical application scenarios. Specifically, the following 2 use cases are discussed:

1. Static calibration: a calibration sample with stable signal strength is introduced, and a program to automatically choose an integration time so that the sensor read out is at a desired range.
2. Dynamic calibration: a sample is undergoing some reaction while being imaged continuously, and an algorithm that automatically adjust integration time to avoid signal saturation.

The ULS24 pixels have wide dynamic range and excellent linearity. Such properties make it possible to apply the simple methods described in this application note to ensure proper capturing of signals of widely varying strength from the sample.

Static calibration

In this case, a calibration sample with stable output strength is used to calibrate the sensor. For example, in qPCR, the samples will emit some base level of fluorescence before the temperature cycling starts (see Figure 1, the "background" signal level). We may need to adjust the integration time so that this base level of signal can be detected, but is at the low end of the sensor's entire dynamic range.

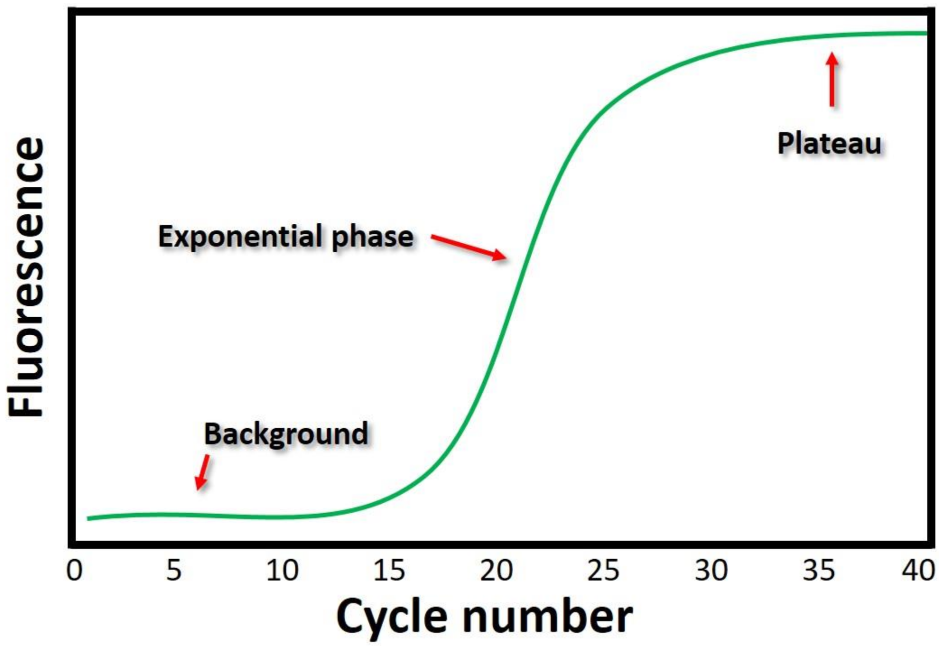

Figure 1 qPCR fluorescence curve. The fluorescence emission level from the sample is at some baseline level prior to amplification. It will then monotonically rise and saturate at some higher level.

Below are steps to perform static calibration:

1. Place calibration sample into the sample holder
2. Choose the lowest integration time. For example, that could be 1ms in some systems.
3. Take an image shot. (it may be a good idea to take several shots and average the output).
4. Extract the image data to get a reading of the light intensity level of the sample.
5. Compare the reading with a desired level of reading and calculate the new integration time (more details of the algorithm provided below).
6. Repeat steps 3 to 5 until reading is sufficiently close to the desired level, or for a fix number of steps.

Below is the sample code to perform the task:

```
float opt_int_time = 1.0f;         // optimal integration time, initialized to 1.0ms
int autoRepeatCounter = 0;

// This is called once to start the automatic calibration
void OnAutocalibInt()
{
  if (!g_DeviceDetected) {
    MessageBox("Device Not Attached");
    return;
  }

  int result = MessageBox("Please insert tubes for calibration.");
  if (result != IDOK) return;

  SetIntegrationTime(opt_int_time);

  StartTimer(TIMER_ID, 1000);      //Start timer to perform auto calibration
}

// Timer portion of the code

..............

       case TIMER_ID:
       {
              int done = AutocalibInt();
              if (done) KillTimer(TIMER_ID);
       }

...............


#define SAMPLE_READING_TARGET 600  // Target sample reading
#define REPEAT_TIMES 10

// This is called by the timer interrupt
int CGraDlg::AutocalibInt()
{
  int done = 0;

  CaptureFrame();                  // capture a frame of image

  int sample_reading;
```

```
  float inc, inc_factor;

  sample_reading = GetSampleReading();          // Sample reading

  if (autoRepeatCounter == 0)
  {
    if (sample_reading < 10) sample_reading = 10;      // avoid divide by zero
    inc_factor = 0.6f / (float)sample_reading;         // 0.6f is an empirical number

  }

  inc = (float)(SAMPLE_READING_TARGET - sample_reading) * inc_factor;
  // slowly approach the opt int time to avoid saturation, smaller step length.

  opt_int_time += inc;

  if (opt_int_time < 1)     // we set an lower limit for integration time.
    opt_int_time = 1;
  else if (opt_int_time > 800)     // we set an upper limit for integration time too.
    opt_int_time = 800;

  SetIntegrationTime(opt_int_time);
  autoRepeatCounter++;

  if (autoRepeatCounter >= REPEAT_TIMES
     || abs(SAMPLE_READING_TARGET - sample_reading) < 30) {
       done = 1;

       autoRepeatCounter = 0;
       MessageBox("Int Time Calibration Done");
  }

       return done;
}
```

Note after each step, we calculate the amount of increase to be applied to the integration time, this is done by:

```
inc = (float)(SAMPLE_READING_TARGET - sample_reading) * inc_factor;
```

So, the increase is proportional to the difference between sample reading and desired sample reading. However, there is a factor “inc_factor” here, which is a small number.

The factor is inversely proportional to the initial reading:

```
if (autoRepeatCounter == 0)
{
	if (sample_reading < 10) sample_reading = 10;	// avoid divide by zero
	inc_factor = 0.6f / (float)sample_reading;	// 0.6f is an empirical number

}
```

This means the smaller the initial reading, the weaker the signal, the bigger step we will take to increase integration time in proportion to sample reading difference to target level.

Dynamic integration time adjustment

In many applications, the light emission level of the analyte is continuously monitored while it is undergoing some sort of chemical reaction. The light emission level of the analyte will change as a result of the reaction. We want to make sure that the light intensity level will not exceed the saturation point of the sensor, or else the information about the reaction will be lost.

In this example, we discuss a method to dynamically adjust the integration time of the ULS24 sensor so that the signal output is always within the detection dynamic range of the sensor.

It is worth pointing out that we base this method on the assumption that the light intensity level of the analyte does not just change wildly and erratically. This assumption is true in most cases concerning chemical reactions of an analyte.

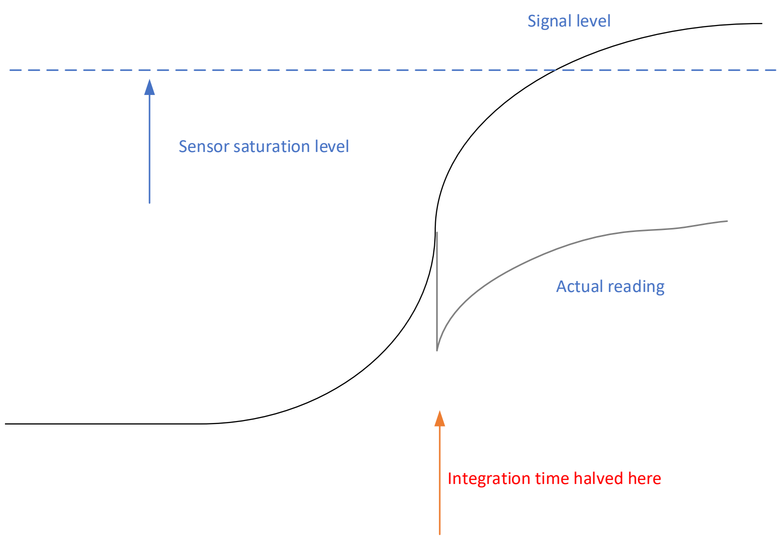

Figure 2 Dynamically adjust integration time to avoid signal saturation.

Again, we can use the qPCR reaction shown in Figure 1 as the example. If the test is positive, the fluorescence emission level may rapidly increase before it saturates. In the case of qPCR, the general trend of the fluorescence level is that it will only increase. This makes things a little simpler. But the method can be adapted to other cases.

The general approach is to use the current and last reading of the sample to predict if the next reading and judge whether a saturation will likely occur. If saturation is predicted, we will then half the integration time.

For each sample reading, we will not only store the light intensity reading for the sample, but also the actual integration time used. When we finally calculate the sample readings, we will normalize it against the integration time. Since ULS24 response to light level and integration time is very linear, this method works well.

```
// Dynamic method, after a sample is read, the following test is performed.

#define SATURATION_LEVEL 4096
#define MARGIN       500

int max, last_max;
bool dynIntTime = false;

void FrameCapture
{
  ...............
  // Decide whether to adjust int time for the next capture

  max = GetMaxSampleReading();
  //Since many samples are read, this returns the maximum sample reading

  if (max + (max - last_max) > SATURATION_LEVEL - MARGIN) {
  // we just perform a linear extraction. "last_max" is the remembered max value from
last sample reading.
    dynIntTime = true;
    // indicate need to adjust integration time

    MessageBox("Overflow detected max %d, last max %d", max, last_max);
  }

  last_max = max;

}

// Before next sample reading, update integration time

float factorIntTime = 1.0f;

void DynamicUpdateIntTime()
{
    if (dynIntTime) {
       factorIntTime *= 0.5;
       SetIntTimeFactor(factorIntTime);
```

```
        dynIntTime = false;  // clear the flag
        MessageBox ("IntTime factor adjusted to %3.2f", factorIntTime);
    }
}
```

Contact

Anitoa Systems, LLC
149 Commonwealth Drive, Suite 1001
Menlo Park, CA 94025
www.anitoa.com
info@anitoa.com