Real Time Image Processing: Celoxica DK Design Suite & PixelStreams

Guillaume Lemaître
Heriot-Watt University, Universitat de Girona, Université de Bourgogne
g.lemaitre58@gmail.com

I. Introduction

In this paper, we will present a brief introduction regarding FPGA programming using Celoxica DK Design Suite. We will present for different exercises. The basic idea is to use a High-level language to program an FPGA. In this paper, we will use PixelStreams which is a software with pre-implemented graphical block function. In order to create a program, we will have only to place some specific blocks. Then, the diagram have to be translate in a Low-level language. The Low-level language used by celoxica is the Handel-C. We can also modify some part directly in this code. Before to implement the program inside the FPGA, we have to compile the Handel-C files which are translate in machine language.

Hence, in this paper, for each different exercise, we will present the different diagram block implemented and the Handel-C file generated. We will present equally the result after to have implemented the code inside the FPGA.

II. Exercice 1

The first exercise is to create a simple example to check if the board is working. The diagram block created in PixelStreams is shown in figure 1

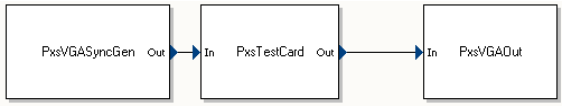

Figure 1. Diagram block to test the board

We generate a synthetic signal with the block *PxsTestCard* which is a function which should create a test pattern. In order to see the result, this signal is sent to the VGA output using *PxsVGAOut*. The *PxsVGASyncGen* block allows to generate a VGA signal sync impulses with synchronous coordinates.

After the compilation with PixelStreams, we obtain the Handel-C file as shown in appendix A. The result observed is about the same than shown on the figure 2.

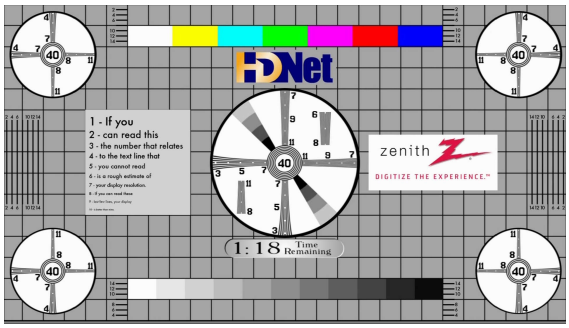

Figure 2. Example of test pattern

III. Exercice 2

A. CMOS camera to VGA output

The idea of the second exercise is to send an input signal from the CMOS camera to the VGA output of the board. The block diagram created with PixelStream is shown on the figure 3

After the compilation with PixelStreams, we obtain the Handel-C file as shown in appendix B-A.

The scale power filters allow to modify the resolution of the image. The block connected to the VGA clock generate the coordinates. If we divide this clock by two, we will copy two pixels at the same coordinates. If we divide this clock by four, we will copy four pixels at the same coordinates. The block connected to the VGA input allows to take one pixel over 2^{power}. If the value of this scale is too important, the camera does not have enough data to deliver. Hence, we can see on the image a corrupted part corresponding on random values.

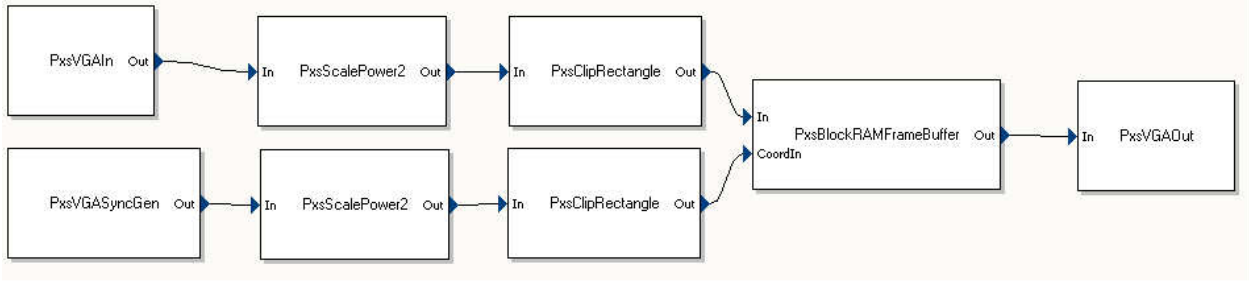

Figure 3. Block diagram for CMOS camera to VGA output application

B. CMOS camera to VGA output and gray-scale conversion

To convert the image in gray-scale image, we have to add to block *PxsConvert*. However, we need to change the type of signal at the input and output of the first and second convert block. The first block convert RGB to gray scale. However, we need a RGB signal in the input of the VGA output. Hence, we need a second converter to convert the gray scale signal to color signal. The block diagram created with PixelStream is shown on the figure 4

After the compilation with PixelStreams, we obtain the Handel-C file as shown in appendix B-B.

IV. Exercice 3

A. Addition of salt and pepper noise

In this section, we will present how to add salt and pepper noise. To add salt and pepper noise we have add the *PxsSaltAndPepper* block. The block diagram created with PixelStream is shown on the figure 5

After the compilation with PixelStreams, we obtain the Handel-C file as shown in appendix C-A.

B. Frequency changement

To change the frequency using the joystick, we have to add the following code:

```
/*
 * Set the starting frequency level
 */
 //Declaration of the gray level
static unsigned 9 Level = 25;

/*
 * Threshold level can be adjusted with
     joystick
 */
while (1)
{
                //Interruption
    PxsAwaitVSync (&syncSignal);
    if (RC10ButtonUpRead () && Level !=
        250)
    {
        Level++;
    }
    else if (RC10ButtonDownRead () &&
        Level != 0)
    {
        Level--;
    }
    else
    {
        delay;
    }
}
```

After the compilation with PixelStreams, we obtain the Handel-C file as shown in appendix C-B.

V. Exercice 4

In the last section we have to remove the noise. Salt and pepper is a noise with 0 or 1 as added value. The best filter for this kind of noise is a median filter. To apply the median filter, we have to split the red, green and blue channels using *PxsExtractRGB* block and apply a median filter using *PxsMedianFilter* on each different channel. Then, we have to recombine all channels with *PxsCombineRGB* block. The block diagram created with PixelStream is shown on the figure 6

After the compilation with PixelStreams, we obtain the Handel-C file as shown in appendix D.

Regarding the latency time, we can compute the number of cycles (values found in the documentation):

$Latency = 1+4+(640+160)+12+1+1+2 = 821$ cycles per pixels.

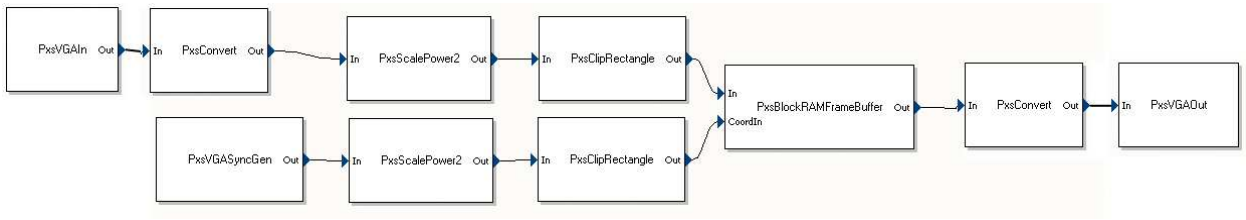

Figure 4. Block diagram for CMOS camera to VGA output application and gray-scale conversion

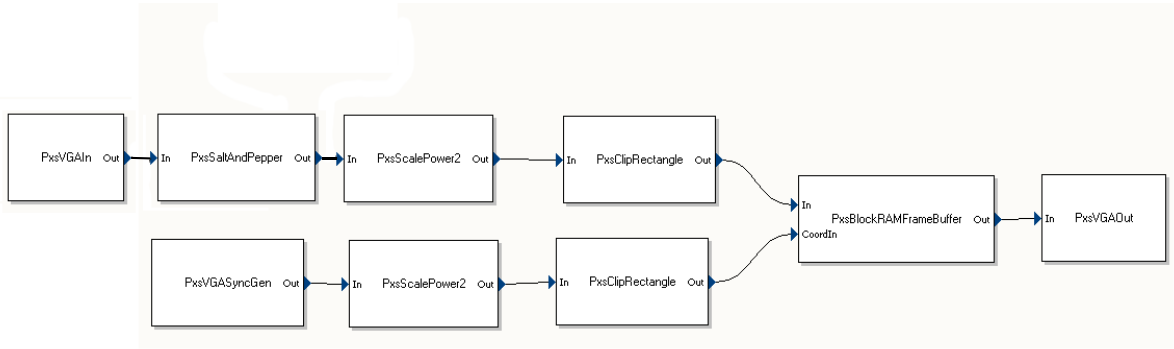

Figure 5. Addition of salt and pepper noise

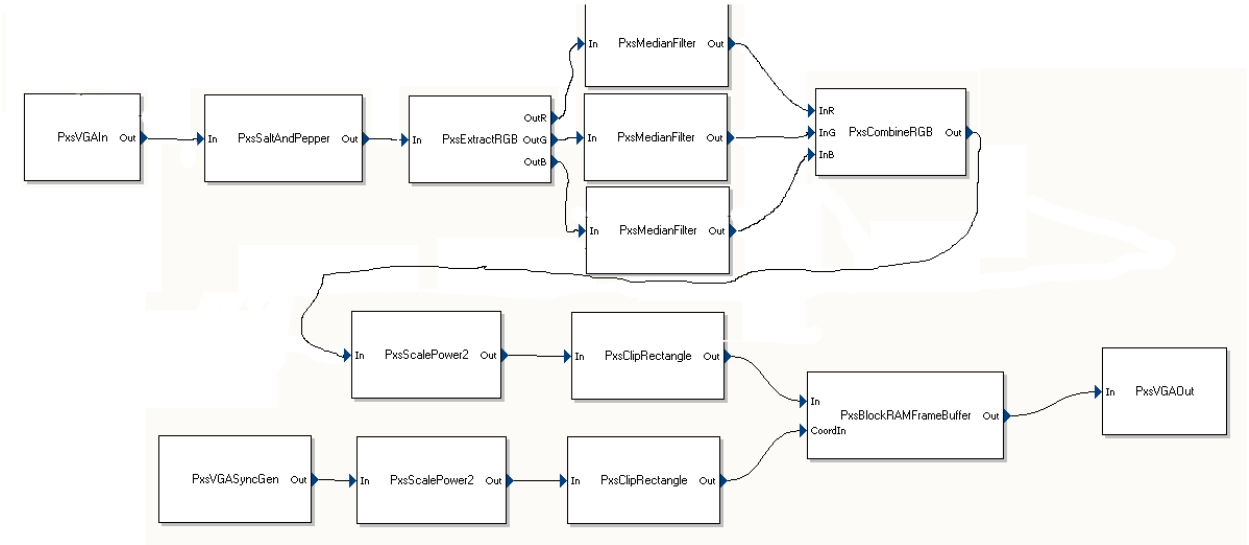

Figure 6. Addition of salt and pepper noise

This number corresponds at: $809 + 800 \times 524 - 1 = 420008$ cycles per pixels.

Hence, with a clock of 100 MHz, we obtain 4.2 frames per milliseconds.

VI. Conclusion

In this paper, we presented a brief introduction of FPGA programming using Celoxica DK Design Suite.

APPENDIX A
EXERCICE 1

```
/*
 * This is an auto-generated source file,
 * created by the PixelStreams GUI application.
 * Copyright (c) Celoxica Ltd. 1991 - 2004
 */

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"


void main (void)
{
    macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
    macro expr Mode      = SyncGen2GetOptimalModeCT   (ClockRate);
    macro expr Width     = SyncGen2GetHActivePixelsCT (Mode);
    macro expr Height    = SyncGen2GetVActiveLinesCT  (Mode);


    /*
     * Streams
     */
    PXS_PV_S (SyncSignal, PXS_EMPTY);
    PXS_PV_S (OutSignal, PXS_RGB_U8);

    /*
     * Filters
     */
    par
    {
        PxsVGASyncGen (&SyncSignal, Mode);
        PxsVGAOut (&OutSignal, 0, 0, ClockRate);
        PxsTestCard (&SyncSignal, &OutSignal, Width, Height);

    }
}
```

APPENDIX B
EXERCICE 2

A. CMOS camera to VGA output

```
/*
 * This is an auto-generated source file,
 * created by the PixelStreams GUI application.
 * Copyright (c) Celoxica Ltd. 1991 - 2004
 */

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"


void main (void)
{
    macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
```

```
    macro expr Mode      = SyncGen2GetOptimalModeCT   (ClockRate);
    macro expr Width     = SyncGen2GetHActivePixelsCT (Mode);
    macro expr Height    = SyncGen2GetVActiveLinesCT  (Mode);


    /*
     * Streams
     */
    PXS_P_S (inSignal, PXS_RGB_U8);
    PXS_PV_S (syncSignal, PXS_EMPTY);
    PXS_P_A (inScSignal, PXS_RGB_U8);
    PXS_PV_A (syncScSignal, PXS_EMPTY);
    PXS_P_A (inClipSignal, PXS_RGB_U8);
    PXS_PV_A (syncClipSignal, PXS_EMPTY);
    PXS_PV_A (outSignal, PXS_RGB_U8);

    /*
     * Filters
     */
    par
    {
        PxsVGAIn (&inSignal, 0, 0, ClockRate);
        PxsVGASyncGen (&syncSignal, Mode);
        PxsScalePower2 (&inSignal, &inScSignal, 1);
        PxsScalePower2 (&syncSignal, &syncScSignal, 1);
        PxsClipRectangle (&syncScSignal, &syncClipSignal, 0, 0, 100, 100);
        PxsClipRectangle (&inScSignal, &inClipSignal, 0, 0, 100, 100);
        PxsBlockRAMFrameBuffer (&inClipSignal, &syncClipSignal, &outSignal, 160, 120,
            PXS_WEAVE, ClockRate);
        PxsVGAOut (&outSignal, 0, 0, ClockRate);

    }
}
```

B. CMOS camera to VGA output and gray-scale conversion

```
/*
 * This is an auto-generated source file,
 * created by the PixelStreams GUI application.
 * Copyright (c) Celoxica Ltd. 1991 - 2004
 */

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"


void main (void)
{
    macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
    macro expr Mode      = SyncGen2GetOptimalModeCT   (ClockRate);
    macro expr Width     = SyncGen2GetHActivePixelsCT (Mode);
    macro expr Height    = SyncGen2GetVActiveLinesCT  (Mode);


    /*
     * Streams
     */
    PXS_PV_A (outSignal, PXS_RGB_U8);
    PXS_P_A (inClipSignal, PXS_MONO_U8);
    PXS_PV_A (syncClipSignal, PXS_EMPTY);
```

```
    PXS_P_A (inClipSignal, PXS_MONO_U8);
    PXS_PV_S (syncSignal, PXS_EMPTY);
    PXS_PV_A (syncScSignal, PXS_EMPTY);
    PXS_P_S (inGraySignal, PXS_MONO_U8);
    PXS_P_S (inSignal, PXS_RGB_U8);
    PXS_P_S (outSignal, PXS_RGB_U8);

     /*
      * Filters
      */
    par
    {
        PxsVGAIn (&inSignal, 0, 0, ClockRate);
        PxsVGASyncGen (&syncSignal, Mode);
        PxsScalePower2 (&inGraySignal, &inScSignal, 1);
        PxsScalePower2 (&syncSignal, &syncScSignal, 1);
        PxsClipRectangle (&syncScSignal, &syncClipSignal, 0, 0, 100, 100);
        PxsClipRectangle (&inScSignal, &inClipSignal, 0, 0, 100, 100);
        PxsBlockRAMFrameBuffer (&inClipSignal, &syncClipSignal, &outGraySignal, 160, 120,
            PXS_WEAVE, ClockRate);
        PxsVGAOut (&outSignal, 0, 0, ClockRate);
                                  PxsConvert (&inSignal, &inGraySignal);
        PxsConvert (&outGraySignal, &outSignal);
    }
}
```

APPENDIX C
EXERCICE 3

A. Addition of salt and pepper noise

```
/*
 * This is an auto-generated source file,
 * created by the PixelStreams GUI application.
 * Copyright (c) Celoxica Ltd. 1991 - 2004
 */

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"


void main (void)
{
    macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
    macro expr Mode      = SyncGen2GetOptimalModeCT   (ClockRate);
    macro expr Width     = SyncGen2GetHActivePixelsCT (Mode);
    macro expr Height    = SyncGen2GetVActiveLinesCT  (Mode);


    /*
     * Streams
     */
    PXS_P_S (inSignal, PXS_RGB_U8);
    PXS_PV_S (syncSignal, PXS_EMPTY);
    PXS_P_A (inScSignal, PXS_RGB_U8);
    PXS_PV_A (syncScSignal, PXS_EMPTY);
    PXS_P_A (inClipSignal, PXS_RGB_U8);
    PXS_PV_A (syncClipSignal, PXS_EMPTY);
    PXS_PV_A (outSignal, PXS_RGB_U8);
```

```
    PXS_P_S (noiseSignal, PXS_RGB_U8);

    /*
     * Filters
     */
    par
    {
        PxsVGAIn (&inSignal, 0, 0, ClockRate);
        PxsVGASyncGen (&syncSignal, Mode);
        PxsScalePower2 (&noiseSignal, &inScSignal, 1);
        PxsScalePower2 (&syncSignal, &syncScSignal, 1);
        PxsClipRectangle (&syncScSignal, &syncClipSignal, 0, 0, 100, 100);
        PxsClipRectangle (&inScSignal, &inClipSignal, 0, 0, 100, 100);
        PxsBlockRAMFrameBuffer (&inClipSignal, &syncClipSignal, &outSignal, 160, 120,
            PXS_WEAVE, ClockRate);
        PxsVGAOut (&outSignal, 0, 0, ClockRate);
        PxsSaltAndPepper (&inSignal, &noiseSignal, 1, 0, 16);

    }
}
```

B. Frequency changements

```
/*
 * This is an auto-generated source file,
 * created by the PixelStreams GUI application.
 * Copyright (c) Celoxica Ltd. 1991 - 2004
 */

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"


void main (void)
{
    macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
    macro expr Mode      = SyncGen2GetOptimalModeCT   (ClockRate);
    macro expr Width     = SyncGen2GetHActivePixelsCT (Mode);
    macro expr Height    = SyncGen2GetVActiveLinesCT  (Mode);


    /*
     * Streams
     */
    PXS_P_S (inSignal, PXS_RGB_U8);
    PXS_PV_S (syncSignal, PXS_EMPTY);
    PXS_P_A (inScSignal, PXS_RGB_U8);
    PXS_PV_A (syncScSignal, PXS_EMPTY);
    PXS_P_A (inClipSignal, PXS_RGB_U8);
    PXS_PV_A (syncClipSignal, PXS_EMPTY);
    PXS_PV_A (outSignal, PXS_RGB_U8);
    PXS_P_S (noiseSignal, PXS_RGB_U8);

    /*
     * Set the starting frequency level
     */
     //Declaration of the gray level
    static unsigned 9 Level = 25;

    /*
```

```
     * Filters
     */
    par
    {
        PxsVGAIn (&inSignal, 0, 0, ClockRate);
        PxsVGASyncGen (&syncSignal, Mode);
        PxsScalePower2 (&noiseSignal, &inScSignal, 1);
        PxsScalePower2 (&syncSignal, &syncScSignal, 1);
        PxsClipRectangle (&syncScSignal, &syncClipSignal, 0, 0, 100, 100);
        PxsClipRectangle (&inScSignal, &inClipSignal, 0, 0, 100, 100);
        PxsBlockRAMFrameBuffer (&inClipSignal, &syncClipSignal, &outSignal, 160, 120,
            PXS_WEAVE, ClockRate);
        PxsVGAOut (&outSignal, 0, 0, ClockRate);
        PxsSaltAndPepper (&inSignal, &noiseSignal, 1, 0, 16);

        /*
         * Threshold level can be adjusted with joystick
         */
        while (1)
        {
                        //Interruption
            PxsAwaitVSync (&syncSignal);
            if (RC10ButtonUpRead () && Level != 250)
            {
                Level++;
            }
            else if (RC10ButtonDownRead () && Level != 0)
            {
                Level--;
            }
            else
            {
                delay;
            }
        }

    }
}
```

Appendix D
Exercice 4

```
/*
 * This is an auto-generated source file,
 * created by the PixelStreams GUI application.
 * Copyright (c) Celoxica Ltd. 1991 - 2004
 */

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"


void main (void)
{
    macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
    macro expr Mode      = SyncGen2GetOptimalModeCT   (ClockRate);
    macro expr Width     = SyncGen2GetHActivePixelsCT (Mode);
    macro expr Height    = SyncGen2GetVActiveLinesCT  (Mode);
```

```
/*
 * Streams
 */
PXS_P_S (inSignal, PXS_RGB_U8);
PXS_PV_S (syncSignal, PXS_EMPTY);
PXS_P_A (inScSignal, PXS_RGB_U8);
PXS_PV_A (syncScSignal, PXS_EMPTY);
PXS_P_A (inClipSignal, PXS_RGB_U8);
PXS_PV_A (syncClipSignal, PXS_EMPTY);
PXS_PV_A (outSignal, PXS_RGB_U8);
PXS_P_S (noiseSignal, PXS_RGB_U8);
PXS_P_S (blueNoiSignal, PXS_MONO_U8);
PXS_P_S (greenNoiSignal, PXS_MONO_U8);
PXS_P_S (redNoisSignal, PXS_MONO_U8);
PXS_P_S (blueSignal, PXS_MONO_U8);
PXS_P_S (greenSignal, PXS_MONO_U8);
PXS_P_S (redSignal, PXS_MONO_U8);
PXS_P_S (rgbSignal, PXS_RGB_U8);


/*
 * Set the starting frequency level
 */
 //Declaration of the gray level
static unsigned 9 Level = 25;

/*
 * Filters
 */
par
{
    PxsVGAIn (&inSignal, 0, 0, ClockRate);
    PxsVGASyncGen (&syncSignal, Mode);
    PxsScalePower2 (&rgbSignal, &inScSignal, 1);
    PxsScalePower2 (&syncSignal, &syncScSignal, 1);
    PxsClipRectangle (&syncScSignal, &syncClipSignal, 0, 0, 100, 100);
    PxsClipRectangle (&inScSignal, &inClipSignal, 0, 0, 100, 100);
    PxsBlockRAMFrameBuffer (&inClipSignal, &syncClipSignal, &outSignal, 160, 120,
        PXS_WEAVE, ClockRate);
    PxsVGAOut (&outSignal, 0, 0, ClockRate);
    PxsSaltAndPepper (&inSignal, &noiseSignal, 1, 0, 16);
    PxsExtractRGB (&noiseSignal, &redNoiSignal, &greenNoiSignal, &blueNoiSignal);
    PxsMedianFilter (&redNoiSignal, &redSignal, Width);
    PxsMedianFilter (&greenNoiSignal, &greenSignal, Width);
    PxsMedianFilter (&blueNoiSignal, &blueSignal, Width);
    PxsCombineRGB (&redSignal, &greenSignal, &blueSignal, &rgbSignal);

    /*
     * Threshold level can be adjusted with joystick
     */
    while (1)
    {
                    //Interruption
        PxsAwaitVSync (&syncSignal);
        if (RC10ButtonUpRead () && Level != 250)
        {
            Level++;
        }
        else if (RC10ButtonDownRead () && Level != 0)
        {
```

```
                Level--;
            }
            else
            {
                delay;
            }
        }

    }
}
```