Freescale Semiconductor

Line scan camera with the FRDM-KL25Z and KSDK [ADC + PIT + GPIO]

By: Technical Information Center

Introduction

This document explains how to enable the FRDM-TFC shield for the FRDM-KL25Z with KSDK which involves two line scan cameras, four LEDs, four dip switches and two push buttons. This document is focused on demonstrate the ease of use of the KSDK peripheral drivers applied to control the Freescale Cup smart car.

Freescale Semiconductor

Contents

1. Create a new KSDK project

This document is focused in KSDK 1.2.0 with KDS 3.0.0, the document "Create a new KSDK 1.2.0 project in KDS 3.0.0" explains with detail the way to create a new KSDK project.

2. The line scan camera

This document is focused on implement the control of a line scan camera without a detailed functional description. The functional description is better explained in the sensor datasheet, in the application note AN4244 and in the document "Line scan camera use".

Figure 1. The line scan camera, (a) front view, (b) back view.

2.1 How the line scan camera works

From the sensor datasheet can be seen the time diagram to request and read the 128 pixels from the camera through the ADC (Analog-to-Digital Converter). The image below shows that the MCU must generate a SI (serial input) signal that defines the start of the data-out sequence, a CLK (clock) signal that controls the pixel output, internally the camera latches the output in every CLK rising edge, after this, the MCU must read the image data through the AO (analog output) signal.

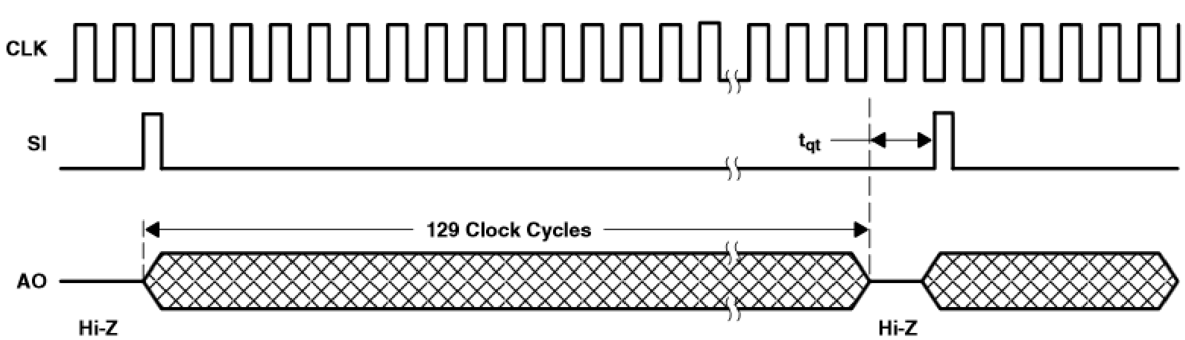

Figure 2. Line scan timing waveforms.

2.2 A FSM to control the line scan camera

To control the two line scan cameras that the FRMD-TFC shield lets to connect, a FSM (Finite State Machine) has been designed to read all the analog inputs. From the figure 2 can be seen that the SI pulse and one CLK cycle have to be generated before the first pixel read. This can be achieved during the states before the first pixel read. The FSM is shown below.

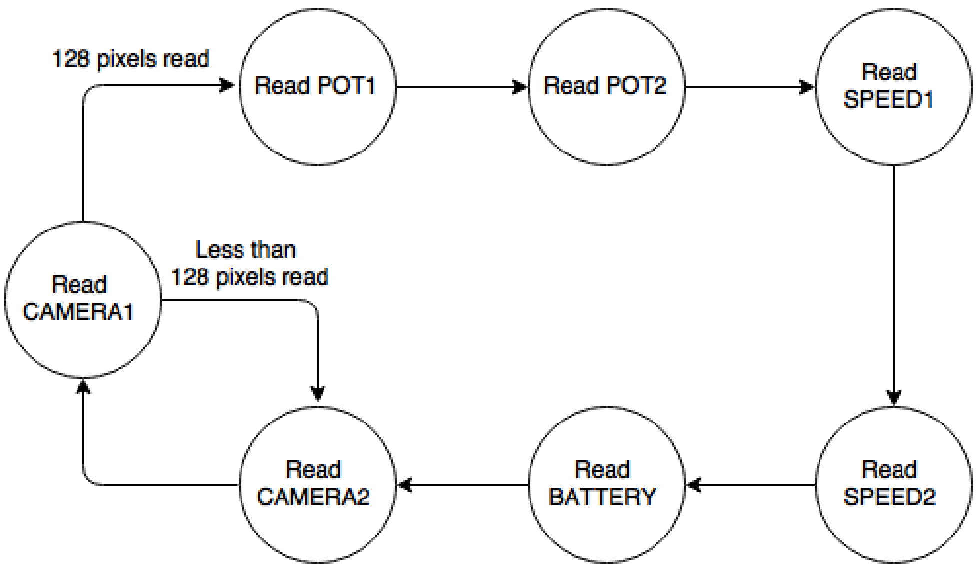

Figure 3. Analog reads FSM.

Before to implement the FSM, the peripherals must be initialized. For this project there will be used the ADC, PIT (Periodic Interrupt Timer) and the GPIO (General-Purpose Input/Output) peripherals.

3. Modify the board.c file

The board.c file contains structures with the configurations about the system such the clock. The configurations about the ADC channels must be added in this file. The FRDM-TFC shield schematics show where the peripherals are routed. The schematics below show the analog channels that will be used to enable the FRDM-TFC shield.

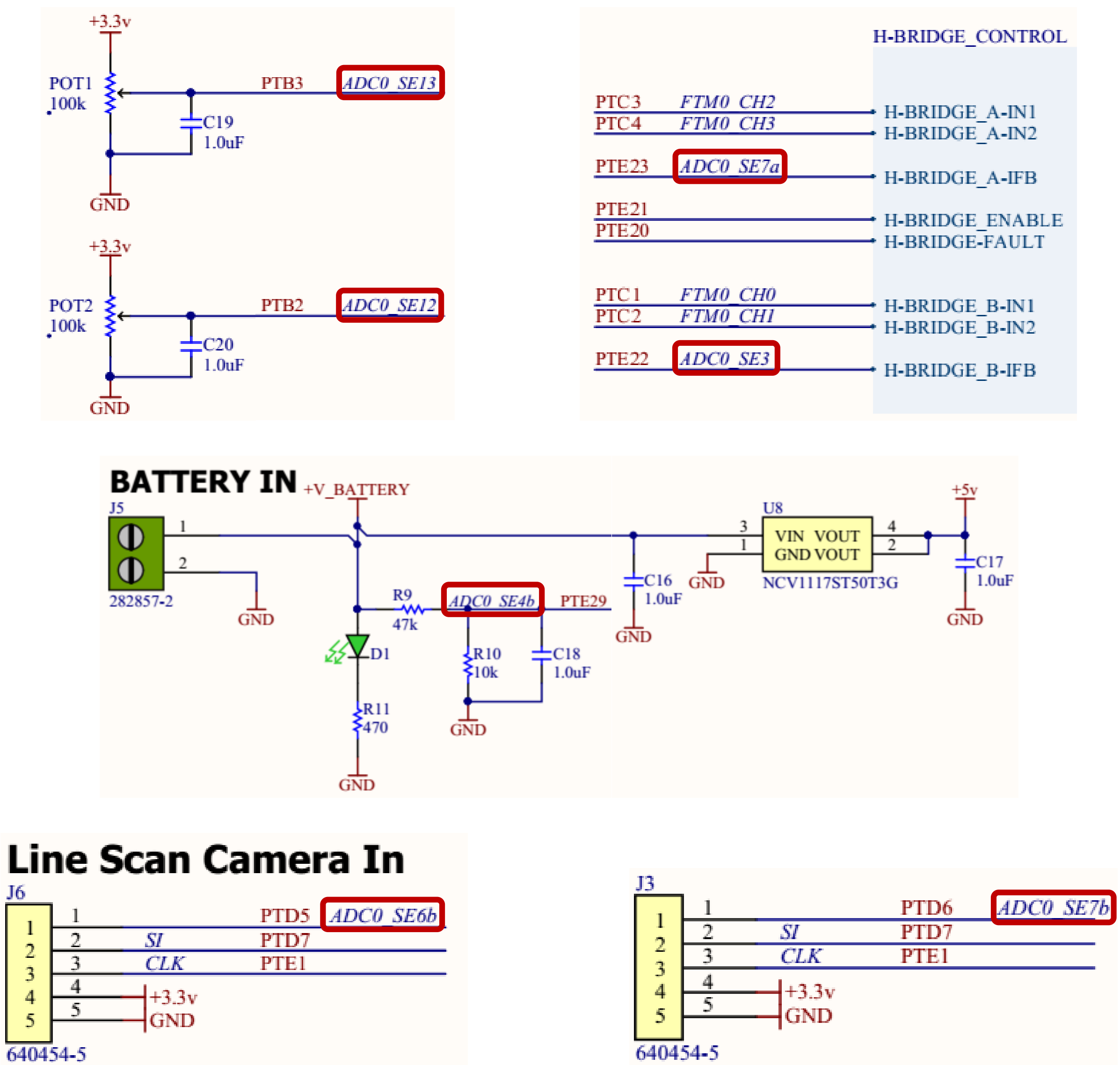

Figure 4. FRDM-TFC analog inputs schematics.

A channel is configured through the structure `adc16_chn_config_t`, the required configuration fields are:

- chnIdx (channel index). For this case it depends on the schematics because the peripherals are already routed.
- convCompletedIntEnable (enable of the interrupt at the end of the conversion). For this case this is true for all the channels because we want to execute the FSM after every conversion.
- diffConvEnable (choose between the differential and the single ended modes). For this case all the channels are single ended.

The channel configurations are shown below.

```
/* Configuration of ADC channels for FRDM-TFC. */
const adc16_chn_config_t TFC_Shield_ADC_Channels[] =
{
		/* kTFCChnPot1 */
		{
				.chnIdx = kAdc16Chn13,
				.convCompletedIntEnable = true,
#if FSL_FEATURE_ADC16_HAS_DIFF_MODE
				.diffConvEnable = false,
#endif
		},
		/* kTFCChnPot2 */
		{
				.chnIdx = kAdc16Chn12,
				.convCompletedIntEnable = true,
#if FSL_FEATURE_ADC16_HAS_DIFF_MODE
				.diffConvEnable = false,
#endif
		},
		/* kTFCChnSpeedA */
		{
				.chnIdx = kAdc16Chn7a,
				.convCompletedIntEnable = true,
#if FSL_FEATURE_ADC16_HAS_DIFF_MODE
				.diffConvEnable = false,
#endif
		},
		/* kTFCChnSpeedB */
		{
				.chnIdx = kAdc16Chn3,
				.convCompletedIntEnable = true,
#if FSL_FEATURE_ADC16_HAS_DIFF_MODE
				.diffConvEnable = false,
#endif
		},
		/* kTFCChnBattery */
		{
```

```
                          .chnIdx = kAdc16Chn4b,
                          .convCompletedIntEnable = true,
#if FSL_FEATURE_ADC16_HAS_DIFF_MODE
                          .diffConvEnable = false,
#endif
             },
             /* kTFCChnCamera1 */
             {
                          .chnIdx = kAdc16Chn6b,
                          .convCompletedIntEnable = true,
#if FSL_FEATURE_ADC16_HAS_DIFF_MODE
                          .diffConvEnable = false,
#endif
             },
             /* kTFCChnCamera2 */
             {
                          .chnIdx = kAdc16Chn7b,
                          .convCompletedIntEnable = true,
#if FSL_FEATURE_ADC16_HAS_DIFF_MODE
                          .diffConvEnable = false,
#endif
             }
};
```

4. Modify the board.h file

The board.h file contains macros used for the GPIOs (General-Purpose Input/Output) which let the application read or write the pins though a simple one-line call. The macros are shown below and there are defined all the GPIOs present in the FRDM-TFC shield and the FRDM-KL25Z board. These macros use functions present in the GPIO peripheral driver to write or read.

```
#define LED_GREEN_EN (GPIO_DRV_OutputPinInit(&ledPins[0]))
      /*!< Enable target LED green */
#define LED_RED_EN (GPIO_DRV_OutputPinInit(&ledPins[1]))
      /*!< Enable target LED red */
#define LED_BLUE_EN (GPIO_DRV_OutputPinInit(&ledPins[2]))
      /*!< Enable target LED blue */
#define TFC_SHIELD_LED1_EN (GPIO_DRV_OutputPinInit(&ledPins[3]))
      /*!< Enable target TFC_LED1 */
#define TFC_SHIELD_LED2_EN (GPIO_DRV_OutputPinInit(&ledPins[4]))
      /*!< Enable target TFC_LED2 */
#define TFC_SHIELD_LED3_EN (GPIO_DRV_OutputPinInit(&ledPins[5]))
      /*!< Enable target TFC_LED3 */
#define TFC_SHIELD_LED4_EN (GPIO_DRV_OutputPinInit(&ledPins[6]))
      /*!< Enable target TFC_LED4 */
#define TFC_LINEAR_CAMERA_SI_EN (GPIO_DRV_OutputPinInit(&ledPins[7]))
      /*!< Enable target TFC_CAMERA_SI */
#define TFC_LINEAR_CAMERA_CLK_EN (GPIO_DRV_OutputPinInit(&ledPins[8]))
      /*!< Enable target TFC_CAMERA_CLK */

#define LED_GREEN_DIS (PORT_HAL_SetMuxMode(PORTB, 19, kPortMuxAsGpio))
      /*!< Disable target LED green */
#define LED_RED_DIS (PORT_HAL_SetMuxMode(PORTB, 18, kPortMuxAsGpio))
      /*!< Disable target LED red */
#define LED_BLUE_DIS (PORT_HAL_SetMuxMode(PORTD,  1, kPortMuxAsGpio))
      /*!< Disable target LED blue */
#define TFC_SHIELD_LED1_DIS (PORT_HAL_SetMuxMode(PORTB,  11, kPortMuxAsGpio))
      /*!< Disable target TFC_LED1 */
#define TFC_SHIELD_LED2_DIS (PORT_HAL_SetMuxMode(PORTB,  10, kPortMuxAsGpio))
      /*!< Disable target TFC_LED2 */
#define TFC_SHIELD_LED3_DIS (PORT_HAL_SetMuxMode(PORTB,  9, kPortMuxAsGpio))
      /*!< Disable target TFC_LED3 */
#define TFC_SHIELD_LED4_DIS (PORT_HAL_SetMuxMode(PORTB,  8, kPortMuxAsGpio))
      /*!< Disable target TFC_LED4 */
#define TFC_LINEAR_CAMERA_SI_DIS (PORT_HAL_SetMuxMode(PORTD,  7, kPortMuxAsGpio))
      /*!< Disable target TFC_CAMERA_SI */
#define TFC_LINEAR_CAMERA_CLK_DIS (PORT_HAL_SetMuxMode(PORTE,  1, kPortMuxAsGpio))
      /*!< Disable target TFC_CAMERA_CLK */

#define LED_GREEN_OFF (GPIO_DRV_WritePinOutput(ledPins[0].pinName, 1))
      /*!< Turn off target LED green */
#define LED_RED_OFF (GPIO_DRV_WritePinOutput(ledPins[1].pinName, 1))
      /*!< Turn off target LED red */
```

```
#define LED_BLUE_OFF (GPIO_DRV_WritePinOutput(ledPins[2].pinName, 1))
       /*!< Turn off target LED blue */
#define TFC_SHIELD_LED1_OFF (GPIO_DRV_WritePinOutput(ledPins[3].pinName, 1))
/*!< Turn off target TFC_LED1 */
#define TFC_SHIELD_LED2_OFF (GPIO_DRV_WritePinOutput(ledPins[4].pinName, 1))
/*!< Turn off target TFC_LED2 */
#define TFC_SHIELD_LED3_OFF (GPIO_DRV_WritePinOutput(ledPins[5].pinName, 1))
/*!< Turn off target TFC_LED3 */
#define TFC_SHIELD_LED4_OFF (GPIO_DRV_WritePinOutput(ledPins[6].pinName, 1))
/*!< Turn off target TFC_LED4 */
#define TFC_LINEAR_CAMERA_SI_OFF (GPIO_DRV_WritePinOutput(ledPins[7].pinName, 0))
/*!< Turn off target TFC_CAMERA_SI */
#define TFC_LINEAR_CAMERA_CLK_OFF (GPIO_DRV_WritePinOutput(ledPins[8].pinName, 0))
/*!< Turn off target TFC_CAMERA_CLK */

#define LED_GREEN_ON (GPIO_DRV_WritePinOutput(ledPins[0].pinName, 0))
       /*!< Turn on target LED green */
#define LED_RED_ON (GPIO_DRV_WritePinOutput(ledPins[1].pinName, 0))
       /*!< Turn on target LED red */
#define LED_BLUE_ON (GPIO_DRV_WritePinOutput(ledPins[2].pinName, 0))
       /*!< Turn on target LED blue */
#define TFC_SHIELD_LED1_ON (GPIO_DRV_WritePinOutput(ledPins[3].pinName, 0))
/*!< Turn on target TFC_LED1 */
#define TFC_SHIELD_LED2_ON (GPIO_DRV_WritePinOutput(ledPins[4].pinName, 0))
/*!< Turn on target TFC_LED2 */
#define TFC_SHIELD_LED3_ON (GPIO_DRV_WritePinOutput(ledPins[5].pinName, 0))
/*!< Turn on target TFC_LED3 */
#define TFC_SHIELD_LED4_ON (GPIO_DRV_WritePinOutput(ledPins[6].pinName, 0))
/*!< Turn on target TFC_LED4 */
#define TFC_LINEAR_CAMERA_SI_ON (GPIO_DRV_WritePinOutput(ledPins[7].pinName, 1))
/*!< Turn on target TFC_CAMERA_SI */
#define TFC_LINEAR_CAMERA_CLK_ON (GPIO_DRV_WritePinOutput(ledPins[8].pinName, 1))
/*!< Turn on target TFC_CAMERA_CLK */

#define LED_GREEN_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[0].pinName))
       /*!< Toggle on target LED blue */
#define LED_RED_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[1].pinName))
       /*!< Toggle on target LED red */
#define LED_BLUE_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[2].pinName))
       /*!< Toggle on target LED blue */
#define TFC_SHIELD_LED1_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[3].pinName))
/*!< Toggle on target TFC_LED1 */
#define TFC_SHIELD_LED2_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[4].pinName))
/*!< Toggle on target TFC_LED2 */
#define TFC_SHIELD_LED3_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[5].pinName))
/*!< Toggle on target TFC_LED3 */
#define TFC_SHIELD_LED4_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[6].pinName))
/*!< Toggle on target TFC_LED4 */
#define TFC_LINEAR_CAMERA_SI_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[7].pinName))
/*!< Toggle on target TFC_CAMERA_SI */
#define TFC_LINEAR_CAMERA_CLK_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[8].pinName))
/*!< Toggle on target TFC_CAMERA_CLK */
```

```
#define TFC_SHIELD_SW1_READ      (GPIO_DRV_ReadPinInput(switchPins[0].pinName))
      /*!< Read target TFC_SW1 */
#define TFC_SHIELD_SW2_READ      (GPIO_DRV_ReadPinInput(switchPins[1].pinName))
      /*!< Read target TFC_SW2 */
#define TFC_SHIELD_DIP1_READ     (GPIO_DRV_ReadPinInput(switchPins[2].pinName))
      /*!< Read target TFC_DIP1 */
#define TFC_SHIELD_DIP2_READ     (GPIO_DRV_ReadPinInput(switchPins[3].pinName))
      /*!< Read target TFC_DIP2 */
#define TFC_SHIELD_DIP3_READ     (GPIO_DRV_ReadPinInput(switchPins[4].pinName))
      /*!< Read target TFC_DIP3 */
#define TFC_SHIELD_DIP4_READ     (GPIO_DRV_ReadPinInput(switchPins[5].pinName))
      /*!< Read target TFC_DIP4 */
```

5. Modify the gpio_pins.c file

The gpio_pins.c file contains two structures, one for the GPIO input pins and one for the output pins. The input structure `gpio_input_pin_user_config_t` have the following configuration fields:

- pinName (pin name). This value is later explained and generated in the file gpio_pins.h.
- config (the GPIO input pin configuration `gpio_input_pin_t`). This structure defines the pin specific hardware configurations and is later explained.

The `gpio_input_pin_t` structure have the following fields:

- isPullEnable (pull up/pull down enable). This field is to enable or disable the pull resistor.
- pullSelect (choose between pull up or pull down if enabled). This field is to select between to use a pull up or use a pull down resistor.
- isPassiveFilterEnabled (enable the passive filter). Field that enables or disables the internal passive filter.
- interrupt (interrupt enable). Enable the interrupt or the DMA request function for the pin.

The output structure `gpio_output_pin_user_config_t` have the following configuration fields:

- pinName (pin name). This value is later explained and generated in the file gpio_pins.h.
- config (the GPIO output pin configuration `gpio_output_pin_t`). This structure defines the pin specific hardware configurations and is later explained.

The `gpio_output_pin_t` structure have the following fields:

- outputLogic (default output value). This is the default value after the GPIO initialization.
- slewRate (slew rate select). Selects between the slow and the fast slew rate.
- driveStrength (drive strength select). Selects between the low and high drive strength.

These configuration structures are shown below.

```
gpio_input_pin_user_config_t switchPins[] = {
            {
                        .pinName = kGpioTFC_SW1,
                        .config.isPullEnable = true,
                        .config.isPassiveFilterEnabled = false,
                        .config.interrupt = kPortIntDisabled,
            },
            {
                        .pinName = kGpioTFC_SW2,
                        .config.isPullEnable = true,
                        .config.isPassiveFilterEnabled = false,
                        .config.interrupt = kPortIntDisabled,
            },
            {
                        .pinName = kGpioTFC_DIP1,
                        .config.isPullEnable = true,
                        .config.isPassiveFilterEnabled = false,
                        .config.interrupt = kPortIntDisabled,
            },
            {
                        .pinName = kGpioTFC_DIP2,
                        .config.isPullEnable = true,
                        .config.isPassiveFilterEnabled = false,
                        .config.interrupt = kPortIntDisabled,
            },
            {
                        .pinName = kGpioTFC_DIP3,
                        .config.isPullEnable = true,
                        .config.isPassiveFilterEnabled = false,
                        .config.interrupt = kPortIntDisabled,
            },
            {
                        .pinName = kGpioTFC_DIP4,
                        .config.isPullEnable = true,
                        .config.isPassiveFilterEnabled = false,
                        .config.interrupt = kPortIntDisabled,
            },
            {
                        .pinName = kGpioSW1,
                        .config.isPullEnable = true,
                        .config.isPassiveFilterEnabled = false,
                        .config.interrupt = kPortIntDisabled,
            },
            {
                        .pinName = GPIO_PINS_OUT_OF_RANGE,
            }
};
```

```
/* Declare Output GPIO pins */
gpio_output_pin_user_config_t ledPins[] = {
        {
                .pinName = kGpioLED_Green,
                .config.outputLogic = 1,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
        {
                .pinName = kGpioLED_Red,
                .config.outputLogic = 1,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
        {
                .pinName = kGpioLED_Blue,
                .config.outputLogic = 1,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
        {
                .pinName = kGpioTFC_LED1,
                .config.outputLogic = 1,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
        {
                .pinName = kGpioTFC_LED2,
                .config.outputLogic = 1,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
        {
                .pinName = kGpioTFC_LED3,
                .config.outputLogic = 1,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
        {
                .pinName = kGpioTFC_LED4,
                .config.outputLogic = 1,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
        /* Linear camera signals. */
        {
                .pinName = kGpioTFC_CameraSI,
                .config.outputLogic = 0,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },
```

```
        {
                .pinName = kGpioTFC_CameraClk,
                .config.outputLogic = 0,
                .config.slewRate = kPortSlowSlewRate,
                .config.driveStrength = kPortLowDriveStrength,
        },

        /* End of linear camera signals.*/
        {
                .pinName = GPIO_PINS_OUT_OF_RANGE,
        }
};
```

6. Modify the gpio_pins.h file

The pinName field in the GPIO configurations structures in the file gpio_pins.c are generated in the file gpio_pins.h. This is an enumeration which contains the information about the port and the pin number and it is generated through the macro GPIO_MAKE_PIN.

```
enum _gpio_pins
{
    kGpioLED_Green      = GPIO_MAKE_PIN(GPIOB_IDX, 19),   /* FRDM-KL25Z4 Green LED */
    kGpioLED_Red        = GPIO_MAKE_PIN(GPIOB_IDX, 18),   /* FRDM-KL25Z4 Red LED */
    kGpioLED_Blue       = GPIO_MAKE_PIN(GPIOD_IDX,  1),   /* FRDM-KL25Z4 Blue LED */

    kGpioTFC_LED1       = GPIO_MAKE_PIN(GPIOB_IDX,  11),  /* FRDM-TFC battery LED 1 */
    kGpioTFC_LED2       = GPIO_MAKE_PIN(GPIOB_IDX,  10),  /* FRDM-TFC battery LED 2 */
    kGpioTFC_LED3       = GPIO_MAKE_PIN(GPIOB_IDX,  9),   /* FRDM-TFC battery LED 3 */
    kGpioTFC_LED4       = GPIO_MAKE_PIN(GPIOB_IDX,  8),   /* FRDM-TFC battery LED 4 */

    kGpioTFC_SW1        = GPIO_MAKE_PIN(GPIOC_IDX,  13),  /* FRDM-TFC push button 1 */
    kGpioTFC_SW2        = GPIO_MAKE_PIN(GPIOC_IDX,  17),  /* FRDM-TFC push button 2 */

    kGpioTFC_DIP1       = GPIO_MAKE_PIN(GPIOE_IDX,  2),   /* FRDM-TFC dip switch 1 */
    kGpioTFC_DIP2       = GPIO_MAKE_PIN(GPIOE_IDX,  3),   /* FRDM-TFC dip switch 2 */
    kGpioTFC_DIP3       = GPIO_MAKE_PIN(GPIOE_IDX,  4),   /* FRDM-TFC dip switch 3 */
    kGpioTFC_DIP4       = GPIO_MAKE_PIN(GPIOE_IDX,  5),   /* FRDM-TFC dip switch 4 */

    kGpioTFC_CameraSI   = GPIO_MAKE_PIN(GPIOD_IDX,  7),/* FRDM-TFC linear camera SI */
    kGpioTFC_CameraClk = GPIO_MAKE_PIN(GPIOE_IDX,  1),/* FRDM-TFC linear camera Clk*/

    kGpioSW1            = GPIO_MAKE_PIN(GPIOD_IDX,  6),/* FRDM-KL25Z4 power manager */
};
```

7. Modify the pin_mux.c file

The pin_mux.c file contains functions to configure the signal multiplexing for the used pins. For this project have been added the new GPIOs that will be used and the ADC pins for the analog peripherals present in the FRDM-TFC shield. This functions must be called in the GPIO or ADC initializations, depending on the case. The new code in this file is shown below.

```
void configure_gpio_pins(uint32_t instance)
{
	switch(instance) {
	case 0:                              /* PTA */
		/* PORTA_PCR14  MMA8451 - INT1 */
		PORT_HAL_SetMuxMode(PORTA,14u,kPortMuxAsGpio);
		/* PORTA_PCR15  MMA8451 - INT2 */
		PORT_HAL_SetMuxMode(PORTA,15u,kPortMuxAsGpio);
		break;
	case 1:                              /* PTB */
		/* PORTB_PCR19 LED1 - Green */
		PORT_HAL_SetMuxMode(PORTB,19u,kPortMuxAsGpio);
		/* PORTB_PCR18 LED2 - Red */
		PORT_HAL_SetMuxMode(PORTB,18u,kPortMuxAsGpio);
		/* PORTB_PCR8 FRDM-TFC LED4 */
		PORT_HAL_SetMuxMode(PORTB,8u,kPortMuxAsGpio);
		/* PORTB_PCR9 FRDM-TFC LED3 */
		PORT_HAL_SetMuxMode(PORTB,9u,kPortMuxAsGpio);
		/* PORTB_PCR10 FRDM-TFC LED2 */
		PORT_HAL_SetMuxMode(PORTB,10u,kPortMuxAsGpio);
		/* PORTB_PCR11 FRDM-TFC LED1 */
		PORT_HAL_SetMuxMode(PORTB,11u,kPortMuxAsGpio);
		break;
	case 2:                              /* PTC */
		/* PORTC_PCR13 FRDM-TFC SW1 */
		PORT_HAL_SetMuxMode(PORTC,13u,kPortMuxAsGpio);
		/* PORTC_PCR17 FRDM-TFC SW2 */
		PORT_HAL_SetMuxMode(PORTC,17u,kPortMuxAsGpio);
		break;
	case 3:                              /* PTD */
		/* PORTD_PCR1 LED3 - Blue */
		PORT_HAL_SetMuxMode(PORTD,1u,kPortMuxAsGpio);
		/* PORTD_PCR6    LLWU_P15 SW1 - Power Manager demo */
		PORT_HAL_SetMuxMode(PORTD,6u,kPortMuxAsGpio);
		/* PORTD_PCR7 FRDM-TFC Camera SI */
		PORT_HAL_SetMuxMode(PORTD,7u,kPortMuxAsGpio);
		break;
	case 4:                              /* PTE */
		/* PORTE_PCR1 FRDM-TFC Camera Clk */
		PORT_HAL_SetMuxMode(PORTE,1u,kPortMuxAsGpio);
		/* PORTE_PCR2 FRDM-TFC DIP1 */
		PORT_HAL_SetMuxMode(PORTE,2u,kPortMuxAsGpio);
```

```
            /* PORTE_PCR3 FRDM-TFC DIP2 */
            PORT_HAL_SetMuxMode(PORTE,3u,kPortMuxAsGpio);
            /* PORTE_PCR4 FRDM-TFC DIP3 */
            PORT_HAL_SetMuxMode(PORTE,4u,kPortMuxAsGpio);
            /* PORTE_PCR5 FRDM-TFC DIP4 */
            PORT_HAL_SetMuxMode(PORTE,5u,kPortMuxAsGpio);
            break;
      default:
            break;
      }
}

void configure_adc_pins(uint32_t instance)
{
      switch (instance)
      {
      case 0U:
            /* PORTE_PCR29 FRDM-TFC Battery */
            PORT_HAL_SetMuxMode(PORTE,29u,kPortPinDisabled);
            /* PORTD_PCR5 FRDM-TFC Camera AO0 */
            PORT_HAL_SetMuxMode(PORTD,5u,kPortPinDisabled);
            /* PORTD_PCR6 FRDM-TFC Camera AO1 */
            PORT_HAL_SetMuxMode(PORTD,6u,kPortPinDisabled);
            /* PORTB_PCR3 FRDM-TFC POT1 */
            PORT_HAL_SetMuxMode(PORTB,3u,kPortPinDisabled);
            /* PORTB_PCR2 FRDM-TFC POT3 */
            PORT_HAL_SetMuxMode(PORTB,2u,kPortPinDisabled);
            /* PORTE_PCR23 FRDM-TFC Speed A */
            PORT_HAL_SetMuxMode(PORTE,23u,kPortPinDisabled);
            /* PORTE_PCR22 FRDM-TFC Speed B */
            PORT_HAL_SetMuxMode(PORTE,22u,kPortPinDisabled);
            break;
      default:
            break;
      }
}
```

8. Modify the pin_mux.h file

The pin_mux.h file contains the prototypes of the functions defined in the file pin_mux.c. For this project has been added just one more prototype which configures the ADC pins.

```
void configure_adc_pins(uint32_t instance);
```

9. Reading the line scan camera data

Once the peripherals are initialized it is time to implement the FSM designed in the section 2. This FSM must generate the SI signal and the first CLK cycle before the camera state. Due to this FSM is executed after an ADC conversion, the start of the analog channels will start when the POT1 conversion finishes. Also, the CLK signal is generated in the states *kTFCChnCamera1* and *kTFCChnCamera2*. The implementation of this FSM is shown below.

```
void TFC_LinearCamera_ADCRead_FSM()
{
	static tfc_shield_adc_chn_t ADC_Read_FSM = kTFCChnPot1;
	static tfc_shield_linear_camera_buffers_t TFC_LinearCamera_BufferCapture = 
kTFCBuffer0;
	static uint8_t TFC_LinearCamera_PixelCapture = 0;
	/* FSM to read all the ADC peripherals in the FRDM-TFC. */
	switch(ADC_Read_FSM)
	{
	case kTFCChnPot1:
		/* Read the POT1 converted value. */
		TFC_Shield_ADC_ReadValues[kTFCChnPot1] = 
ADC16_DRV_GetConvValueRAW(TFC_LINEAR_CAMERA_ADC_INSTANCE, 
TFC_LINEAR_CAMERA_ADC_GROUP);

		/* Start to read the POT2. */
		ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE, 
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnPot2]);
		ADC_Read_FSM = kTFCChnPot2;
		break;
	case kTFCChnPot2:
		TFC_LINEAR_CAMERA_CLK_OFF;

		/* Read the POT2 converted value. */
		TFC_Shield_ADC_ReadValues[kTFCChnPot2] = 
ADC16_DRV_GetConvValueRAW(TFC_LINEAR_CAMERA_ADC_INSTANCE, 
TFC_LINEAR_CAMERA_ADC_GROUP);

		/* Select the side A of the ADC mux. */
		ADC16_DRV_SetChnMux(TFC_LINEAR_CAMERA_ADC_INSTANCE, kAdc16ChnMuxOfA);

		/* Start to read the SPEEDA. */
		ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE, 
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnSpeedA]);

		ADC_Read_FSM = kTFCChnSpeedA;
		break;
	case kTFCChnSpeedA:
		TFC_LINEAR_CAMERA_SI_ON;
```

```
                /* Read the SPEEDA converted value. */
                TFC_Shield_ADC_ReadValues[kTFCChnSpeedA] =
ADC16_DRV_GetConvValueRAW(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP);

                /* Start to read the SPEEDB. */
                ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnSpeedB]);

                ADC_Read_FSM = kTFCChnSpeedB;
                break;
        case kTFCChnSpeedB:
                TFC_LINEAR_CAMERA_CLK_ON;

                /* Read the SPEEDB converted value. */
                TFC_Shield_ADC_ReadValues[kTFCChnSpeedA] =
ADC16_DRV_GetConvValueRAW(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP);

                /* Select the side B of the ADC mux. */
                ADC16_DRV_SetChnMux(TFC_LINEAR_CAMERA_ADC_INSTANCE, kAdc16ChnMuxOfB);

                /* Start to read the BATTERY. */
                ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnBattery]);

                ADC_Read_FSM = kTFCChnBattery;
                break;
        case kTFCChnBattery:
                TFC_LINEAR_CAMERA_SI_OFF;

                /* Read the BATTERY converted value. */
                TFC_Shield_ADC_ReadValues[kTFCChnBattery] =
ADC16_DRV_GetConvValueRAW(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP);

                /* Start to read the CAMERA1. */
                ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnCamera1]);
                ADC_Read_FSM = kTFCChnCamera1;
                break;
        case kTFCChnCamera1:
                TFC_LINEAR_CAMERA_CLK_OFF;

                /* Read the CAMERA1 converted value. */
        TFC_LinearCamera[kTFCCamera1][TFC_LinearCamera_BufferCapture][TFC_LinearCamera
_PixelCapture] = ADC16_DRV_GetConvValueRAW(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP);

                /* Start to read the CAMERA2. */
                ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnCamera2]);
```

```
            ADC_Read_FSM = kTFCChnCamera2;
            break;
      case kTFCChnCamera2:
            TFC_LINEAR_CAMERA_CLK_ON;

            /* Read the CAMERA2 converted value. */

      TFC_LinearCamera[kTFCCamera2][TFC_LinearCamera_BufferCapture][TFC_LinearCamera
_PixelCapture] = ADC16_DRV_GetConvValueRAW(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP);
            TFC_LinearCamera_PixelCapture++;

            /* If the FSM finished to read all the pixels. */
            if(TFC_LINEAR_CAMERA_PIXELS == TFC_LinearCamera_PixelCapture)
            {
                  /* If you want to be always reading the ADC channels un-comment
this line to restart the FSM (the ADC conversions) and disable the PIT timer. */
                  /* Start to read the POT1. */
                  //ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnPot1]);

                  /* Reset the pixels counter. */
                  TFC_LinearCamera_PixelCapture = 0;

                  /* Swap capture buffers. */
                  TFC_LinearCamera_BufferCapture = (kTFCBuffer0 ==
TFC_LinearCamera_BufferCapture) ? kTFCBuffer1 : kTFCBuffer0;

                  /* A new camera frame has been captured. */
                  TFC_LinearCamera_ReadFinished = true;

                  /* Restart the FSM. */
                  ADC_Read_FSM = kTFCChnPot1;
            }
            else
            {
                  /* Start to read the CAMERA1. */
                  ADC16_DRV_ConfigConvChn(TFC_LINEAR_CAMERA_ADC_INSTANCE,
TFC_LINEAR_CAMERA_ADC_GROUP, &TFC_Shield_ADC_Channels[kTFCChnCamera1]);

                  ADC_Read_FSM = kTFCChnCamera1;
            }

            break;
      default:
            /* If there is an unknown state go to the first one. */
            ADC_Read_FSM = kTFCChnPot1;
            break;
      }
}
```

9.1 Base time to generate a frame rate

The camera data is not useful if it is not processed. That processing needs time so it is desired to avoid an always-capturing scheme. To avoid to be always capturing data from the camera a base time must be configured. For this project a PIT (Periodic Interrupt Timer) has been configured with an interrupt every 100 ms, that interrupt is used to start a new frame read. The PIT to generate this base time can be later replaced by another peripheral.

```
void TFC_LinearCamera_PeriodicCaptureInit()
{
	/* Enable a PIT with a period of 20 ms to avoid to be always reading the
camera
	 * because it is not needed because the information has to be processed
between frames. */
	pit_user_config_t pitConfig = {
			.isInterruptEnabled = true,	/* Interrupt enabled. */
			.periodUs = 100000U		/* Interrupt period set to 100 ms.
*/
	};

	printf("Linear Camera PIT init... ");

	/* Initialize PIT instance. Timers will stop running in debug mode. */
	if(kStatus_PIT_Success == PIT_DRV_Init(TFC_LINEAR_CAMERA_PIT_INSTANCE, false))
	{
		printf("OK!\n\r");
	}
	else
	{
		printf("Error!\n\r");
	}

	/* Initialize PIT instance 0, timer 0. */
	PIT_DRV_InitChannel(TFC_LINEAR_CAMERA_PIT_INSTANCE,
TFC_LINEAR_CAMERA_PIT_CHANNEL, &pitConfig);

	/* Start PIT instance 0, timer 0. */
	TFC_LINEAR_CAMERA_START_CAPTURE;
}
```

10. Results

The images below show the AO, CLK and SI signals captured with a logic analyzer and the plot of the read ADC data.

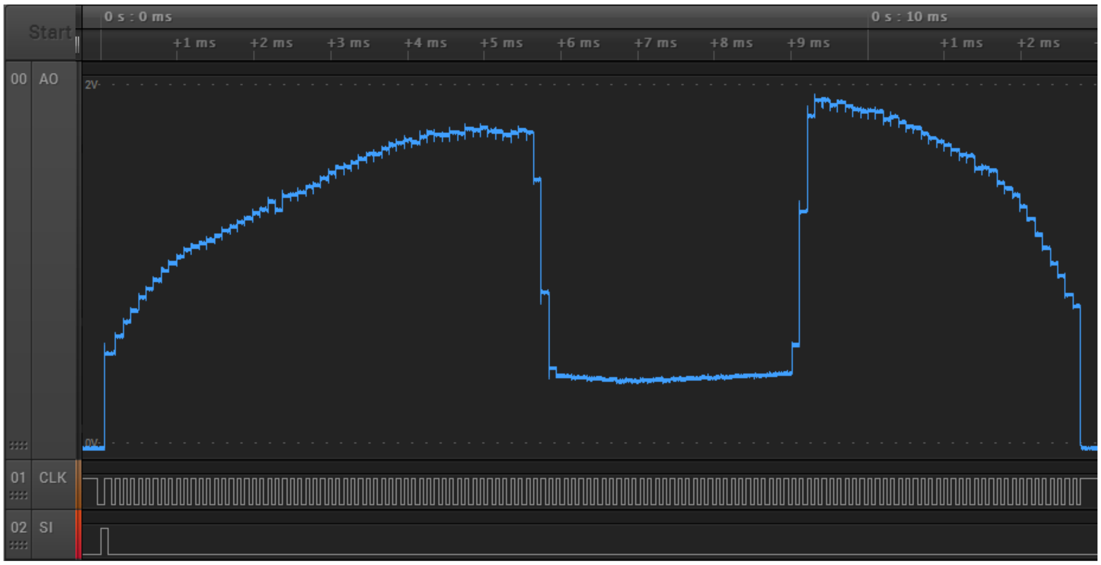

Figure 5. The AO, CLK and SI signals of a dark belt over a clear background. Signals captured with a logic analyzer.

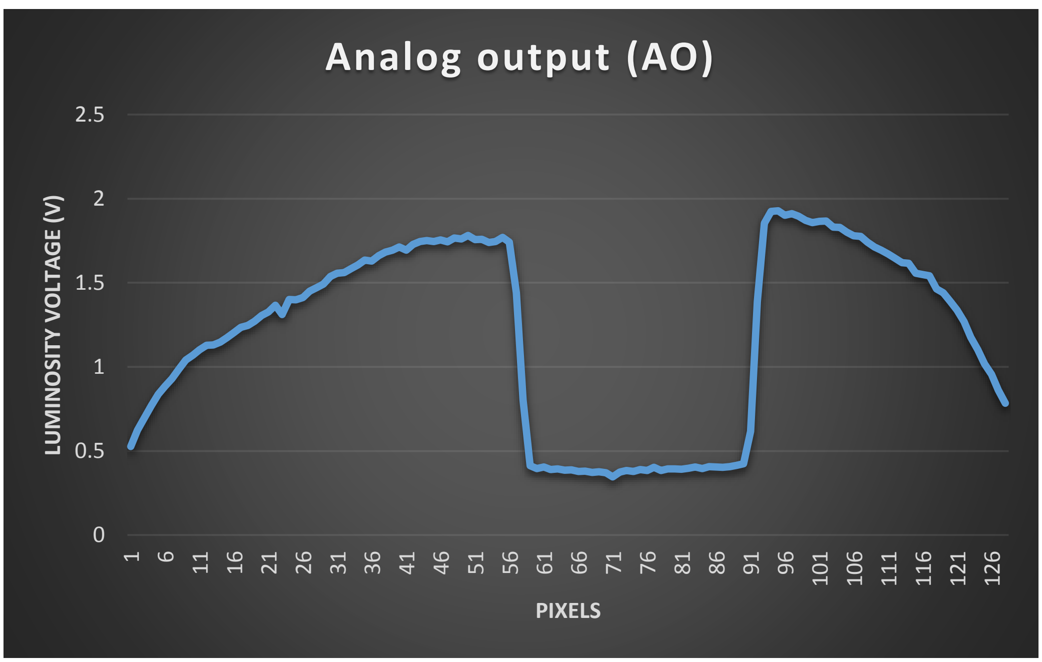

Figure 6. The AO signal of a dark belt over a clear background. Plot of the ADC read data.

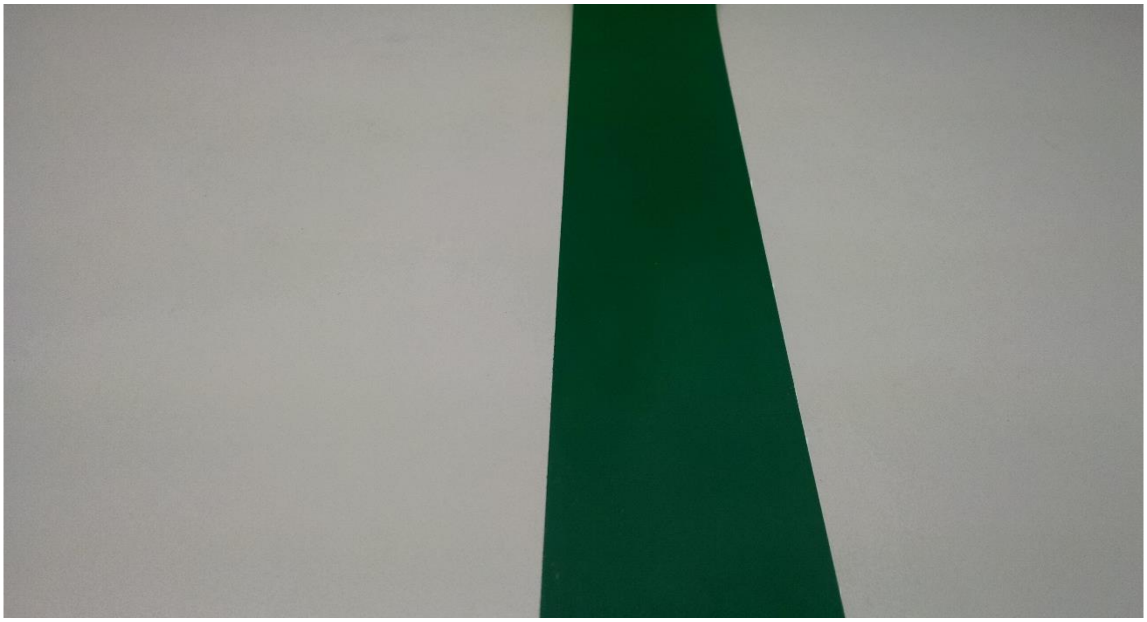

Figure 7. A dark belt over a clear background.

Can be seen that the data stored in the arrays corresponds to the analog signal. This data can be processed to detect where the dark belt is located in the frame.

11. Conclusions

This document has demonstrated the ease of use of the KSDK peripherals, with a few lines of code have been enough to get data from the line scan camera provided in The Freescale Cup Kit. Also, this document is a reference to learn the considerations and the steps to configure a KSDK peripheral such ADC, PIT and GPIO.