AN12583

Enabling Camera on LPC5460x with State Configurable Timer State Machine

Rev. 0 — 16 September 2019 Application Note

1 Introduction

LPC5460x does not support camera interface by hardware originally. However, to enable the camera device with LPC5460x MCU, this function is implemented with State Configurable Timer (SCT) and Direct Memory Access (DMA) with suitable sync signals under a suitable configuration. SCT is used to capture the sync signals and DMA is used to move data without the interaction of the CPU. This document describes a design of software camera interface solution with SCT and DMA on LPC5460x.

Contents

2 SCT hardware

SCT is a hardware module running based on hardware counter/timer. It has various conditions to create events. The event depends on various conditions including the counter value, input, signals, and other conditions. Then, the event can react with several operations, like controlling the counter, setting the output pin voltage level, changing the state machine, and so on.

The SCT module on LPC5460x MCU supports:

- 8 inputs
- 10 outputs
- 10 match/capture registers
- 10 events
- 10 states

2.1 SCT counter and pins

SCT is configured as two 16-bit counters or one 32-bit counter depending on the SCT_CONFIG[UNIFY] bit. In most case, the 32-bit counter is used by default.

SCT_CONFIG register is used to configure the basic counter.

SCT_CTRL register is used to control the basic counter directly, no need to wait for any events.

SCT_OUTPUTDIRCTRL register specifies (for each output) the counting direction of the basic counter.

SCT_COUNT register holds the current counter value moving per each SCT clock.

INPUT and OUTPUT registers are like the control registers to GPIO, they read and write the voltage level on the SCT input and output signals. Each bit in INPUT register is for each SCT input signal. Each bit in OUTPUT register is for each SCT output signal.

NOTE

The SCT input and output signal are not directly connected to the GPIO pins. INPUT MUX module makes the mapping of the connection between SCT input/output signal and external GPIO pins.

2.2 Generation of events

The following conditions define an event:

- a counter match condition.
- an input (or output) condition such as a rising or falling edge or level.
- a combination of match and/or input/output condition.
- in bidirectional mode, events can be enabled based on the count direction.

The value in SCT_MATCH registers are used to create events by comparing the counter's value. Match event occurs in the SCT clock in which the counter is (or would be) incremented to the next value. After the match event, the SCT_MATCH would load the new value from its responding SCT_MATCHRELn register.

The pin condition, the combination mode, and the bidirectional mode are set up in the SCT_EVn_CTRL register for each event.

2.3 Operation of events

Once the event occurs, it can:

- limit, halt, start, or stop a counter or change its direction.
- change state.
- output indicated voltage level on output pins.
- capture current counter value.
- trigger interrupt.
- trigger DMA.

SCT_LIMIT, SCTHALT, SCT_STOP, and SCT_START registers are to set up the operation to counter when event occurs. Each bit in the register is for an event. For example, set the bit 2 in LIMIT means that the counter would turn to the opposite direction or back to zero directly (according to the settings in SCT_OUTPUTDIRCTRL register) when the event 2 happens.

SCT_OUTn_SET and SCT_OUTn_CLR registers are like the output control registers to GPIO, but only driven by event, not output to pin directly by software. Then write the logic voltage level on the SCT's output pins. Each SCT_OUTn_SET/OUTn_CLR register is for a specific SCT output pin, while each bit in this register defines which event can operate this pin.

If enabling the capture function for several events in SCT_CAPCTRL register, once any enabled event occurs, the current counter value would be captured in SCT_CAPn register. In this case, each index of SCT_CAPCTRL registers and SCT_CAPn registers is for a capture monitor, while each bit in SCT_CAPCTRL register is for an event which can cause the capture operation.

SCT_DMAREQ0 and SCT_DMAREQ1 are used to set up operation to DMA controller when event happens. Each bit in this register is for the index number for each event. SCT has two DMA trigger sources.

SCT_EVEN is used to set up operation to interrupt controller when event happens. Each bit in this register is for the index number for each event.

2.4 State machine in SCT

The state machine's status is kept in SCT_STATE register. its value can be updated per the events. in SCT_EVn_CTRL register for each event, there are several fields to configure the operation to the state machine.

"STATELD" and "STATEV" would tell the SCT how to update the state value when the event occurs:

- When STATELD = 0, STATEV value is added into SCT_STATE.
- When STATELD = 1, STATEV value is loaded into SCT_STATE.

An important thing is, only the enabled events in current state can capture the event condition. The SCT_EVn_STATE register for each event is used to keep the event available in any state. Each bit in registers is for an available state. For example, set the bit 3 in SCT_EVn_STATE means the event 2 can be available in state 3. Then, the SCT_EVn_STATE registers are used to create the routines between states.

3 Camera signals and connections with MCU

OV7620 camera module with onboard crystal clock source is used in the demo application. It is connected to MCU with a group of signals:

- [input] I2C/SCCB bus to set up the camera sensor's internal registers.
- [output] 16 b/8 b parallel data of pixels.
- [output] three sync signal of pixel: PCLK for a new pixel, HREF for a new line and VSYNC for a new frame.

Table 1. Camera pins and connections

Camera Pins	MCU Pins	MCU Peripheral
SCCB_SDA	PIO0_26	I2C2_SDA
SCCB_SCL	PIO0_27	I2C2_SCL
PCLK	PIO0_17	SCT_GPI7 > IN2
HREF	PIO0_14	SCT_GPI1 > IN1
VSYNC	PIO0_13	SCT_GPI0 > IN0
D0	PIO1_24	GPIO1
D1	PIO1_25	GPIO1
D2	PIO1_26	GPIO1
D3	PIO1_27	GPIO1
D4	PIO1_28	GPIO1
D5	PIO1_29	GPIO1
D6	PIO1_30	GPIO1
D7	PIO1_31	GPIO1

To read an available sequence of pixel data, the MCU should wait the VSYNC START (a falling edge on VSYNC pin) for a frame, then wait the HREF START(a rising edge on HREF pin) for a new line, and finally to capture the pixel data from bus alone each the PCLK (a rising edge on PCLK pin). At the end of a line, an HREF END (a falling edge on HREF pin) is used to tell the MCU that the following pixels are not available until the next HREF START. At the end of a frame, the VSYNC END (a rising edge on VSYNC pin) is used to tell that the following lines are not available until the next VSYNC START. Only the available pixels in available lines are the wanted sensing image data.

Actually, in the LPC54605 project, the INPUT MUX module is used to map the external SCT*GPIx pins to SCT*INx signals.

```
#define APP_SCT_INPUT_LINE_VSYNC  0U
#define APP_SCT_INPUT_LINE_HREF   1U
#define APP_SCT_INPUT_LINE_PCLK   2U
     /*
     * Camera   IO         PinMux      SCT line
     * VSYNC -> PIO0_13 -> SCT0_GPI0 -> IN0
     * HREF  -> PIO0_14 -> SCT0_GPI1 -> IN1
     * PCLK  -> PIO0_17 -> SCT0_GPI7 -> IN2
     */
     INPUTMUX_AttachSignal(INPUTMUX, 0U, kINPUTMUX_SctGpi0ToSct0); /* IN0. */
     INPUTMUX_AttachSignal(INPUTMUX, 1U, kINPUTMUX_SctGpi1ToSct0); /* IN1. */
     INPUTMUX_AttachSignal(INPUTMUX, 2U, kINPUTMUX_SctGpi7ToSct0); /* IN2. */
```

4 Design a state machine to capture the camera sync signals

As the camera would send mass data continually, we would like to handle the capture of sync signals and the data movement totally by hardware. If using pin interrupt to capture these sync signals, heavy working load would be cost by MCU. Even then, the MCU with software might miss a few sync tokens during the MCU is processing the data. So, in the desired the state machine, SCT automatically handles all the transform between states and the trigger commands to DMA.

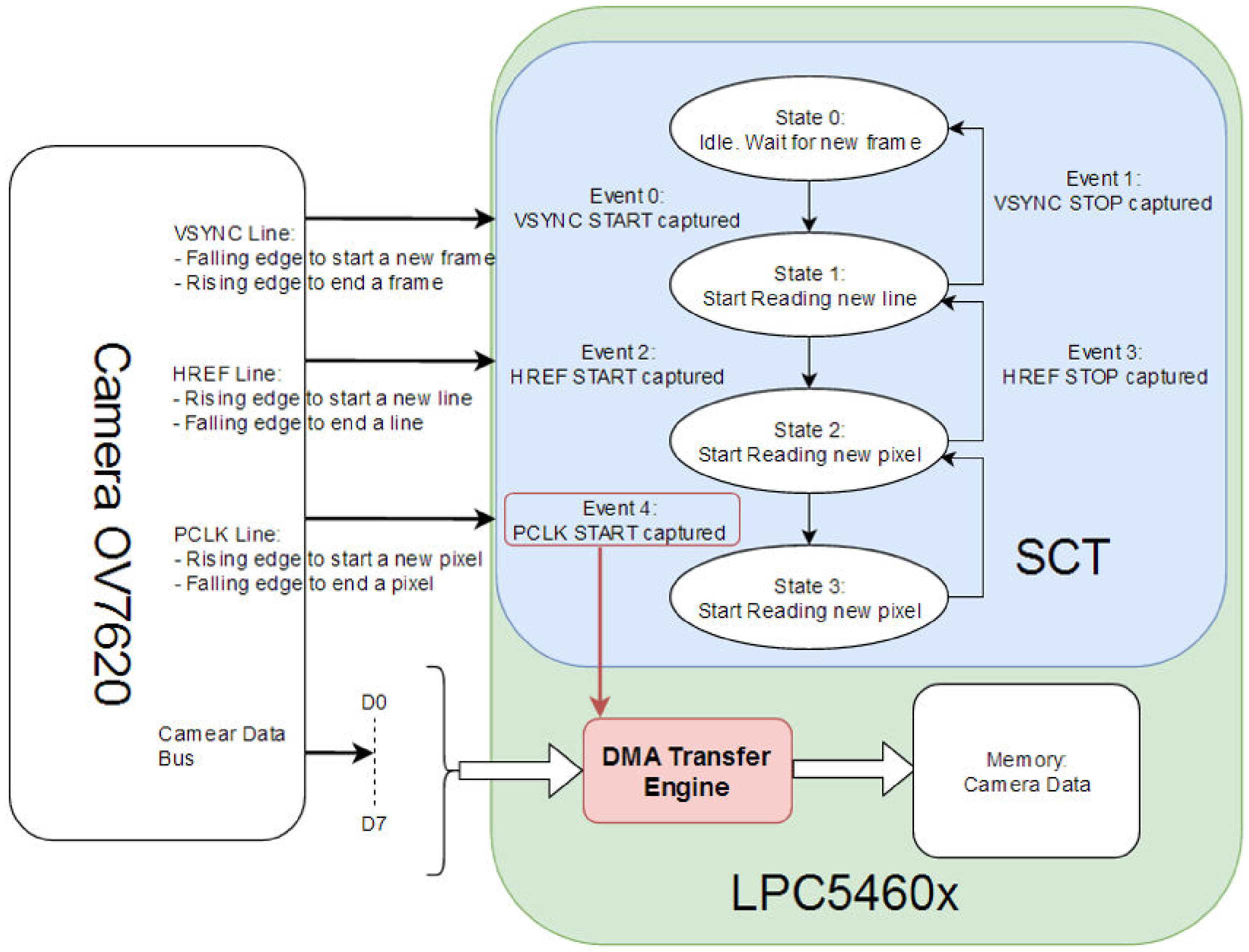
Camera OV7620
VSYNC Line:
- Falling edge to start a new frame
- Rising edge to end a frame
HREF Line:
- Rising edge to start a new line
- Falling edge to end a line
PCLK Line:
- Rising edge to start a new pixel
- Falling edge to end a pixel
Camear Data Bus
D0
D7
State 0: Idle. Wait for new frame
Event 0: VSYNC START captured
Event 1: VSYNC STOP captured
State 1: Start Reading new line
Event 2: HREF START captured
Event 3: HREF STOP captured
State 2: Start Reading new pixel
Event 4: PCLK START captured
State 3: Start Reading new pixel
SCT
DMA Transfer Engine
Memory: Camera Data
LPC5460x

Figure 1. Hardware state machine for camera

In Figure 1, a state machine based on SCT is designed to capture the camera's sync signals and trigger the DMA to move available pixel data to user memory. The hardware implements all the functions aand runs automatically without any software interaction.

5 events are pre-defined as the conditions of movement from one state to another.

- Event 0: VSYNC START (a falling edge on VSYNC line) to an SCT input pin.
- Event 1: VSYNC END (a rising edge on VSYNC line) to an SCT input pin.
- Event 2: HREF START (a rising edge on HREF line) to an SCT input pin.
- Event 3: HREF END (a falling edge on HREF line) to an SCT input pin.

- Event 4: PCLK START (a rising edge on PCLK line) to an SCT input pin.

In the source code, they are defined with relevant codes.

```
#define APP_SCT_EVENT_VSYNC_START 0U /* IN0 falling edge. */
#define APP_SCT_EVENT_VSYNC_END   1U /* IN0 rising  edge. */
#define APP_SCT_EVENT_HREF_START  2U /* IN1 rising  edge. */
#define APP_SCT_EVENT_HREF_END    3U /* IN1 falling edge. */
#define APP_SCT_EVENT_PCLK_START  4U /* IN2 rising  edge. */
```

Then 4 states are defined to represent the temporary states:

- State 0 is the initial state, waiting for a new frame. In the state 0, only the event 0 is available, while other events are masked. Once the event 0 occurs (VSYNC START comes), it moves to the state 1 to wait for a new line.
- State 1 is waiting for a new line. The state 1 has two ways out:
  - — for event 1 (VSYNC STOP comes), it would return to state 0 for a new frame.
  - — for event 2 (HREF START comes), it would go to state 2 and wait for the first pixel data in the current line.
- State 2 is waiting for a new pixel. The state 3 has two ways out:
  - — for event 1 (VSYNC STOP comes), it would return to state 0 for a new frame.
  - — for event 3 (HREF STOP comes), it would return to state 1 for a new line.
  - — for event 4 (PCLK START comes), it would go to its shadow state 3.
  - — every time switch to the state 2, a DMA trigger is generated from SCT to tell DMA moving pixel data from data pins.
- State 3 is the shadow state of State 2. It is waiting for a new pixel as well. It accepts the same move condition to state 0 and state 1. However, here we would like to sample the alternative pixel in a line to reduce the count of data as not all the pixels are necessary (camera is already configured with alternative line sample mode). So the state 3 can be considered to jump(or consume) the additional (unnecessary) triggers to DMA.

In the source code, they are defined with relevant codes.

```
#define APP_SCT_STATE_WAIT_NEW_FRAME  0U
#define APP_SCT_STATE_WAIT_NEW_LINE   1U
#define APP_SCT_STATE_WAIT_NEW_PCLK   2U
#define APP_SCT_STATE_WAIT_NEXT_FRAME 3U
```

If the state machine starts in the middle of frame's transfer, it would wait without any movement, since all the events are disabled except the event 0 (waiting for VSTART START) in the initial state, Only when the next VSYNC START comes. the state machine runs with the transform between states.

Here the DMA is configured as in burst mode. Once a DMA trigger comes, the DMA controller move one pixel data from PIO1[24:31] to user memory. After it moves a line of pixels (320 pixels in a line in current demo application), a DMA transfer done interrupt would be executed. In the DMA ISR function, the transfer task must be reconfigured, since the LPC DMA cannot support longer transfer than 1024 items. There is enough time to run this piece of software during the line sync is inactive. A software counter is used to count the lines. When enough lines are collected for an image (240 lines in current demo application), it can tell the higher-level application software that a full image is ready in MCU's RAM.

The events of line start, line stop, frame start, frame end can be also monitored in SCT module. For example, the frame end in event 1 can be configured to generate the SCT interrupt and tell the higher application software that the image is ready. This way is more more suitable and recommended, as the DMA and the SCT can handle the different task separately: the DMA ISR can focus on the data transfer and SCT ISR can focus on the event detection.

The most important work in MCU source project is to program each event. Finally, when coding for the SCT configuration, we must convert our state machine's view from state-orient to event-orient. It means, even the state machine is describing states and the transform between them under the event condition, we have to say that they are events in their surviving states. Then, the code would be:

Setup the event's condition and the target state:

```
/* setup the event operations. */
    /* VSYNC START event :
     * - APP_SCT_INPUT_LINE_VSYNC occurs on VSYNC input falling edge.
     * - switch to APP_SCT_STATE_WAIT_NEW_LINE, wait for a new line.
     */
    SCT0->EVENT[APP_SCT_EVENT_VSYNC_START].CTRL = SCT_EVENT_CTRL_MATCHSEL(0) /* no use. */
                                                | SCT_EVENT_CTRL_HEVENT(0)   /* no use. */
                                                | SCT_EVENT_CTRL_OUTSEL(0)   /* input pin trigger. */
                                                | SCT_EVENT_CTRL_IOSEL(APP_SCT_INPUT_LINE_VSYNC) /*
VSYNC pin. */
                                                | SCT_EVENT_CTRL_IOCOND(2) /* pin falling edge
trigger. */
                                                | SCT_EVENT_CTRL_COMBMODE(2) /* use io without
counter. */
                                                | SCT_EVENT_CTRL_STATELD(1) /* load state value. */
                                                |
SCT_EVENT_CTRL_STATEV(APP_SCT_STATE_WAIT_NEW_LINE) /* new state. */
                                                | SCT_EVENT_CTRL_MATCHMEM(0) /* no use. */
                                                | SCT_EVENT_CTRL_DIRECTION(0) /* no use. */
                                                ;
    /* HREF START event :
     * - APP_SCT_INPUT_LINE_HREF occurs on HREF input rising edge.
     * - switch to APP_SCT_STATE_WAIT_NEW_PCLK, wait for a new pixel.
     */
    SCT0->EVENT[APP_SCT_EVENT_HREF_START ].CTRL = SCT_EVENT_CTRL_MATCHSEL(0) /* no use. */
                                                | SCT_EVENT_CTRL_HEVENT(0)   /* no use. */
                                                | SCT_EVENT_CTRL_OUTSEL(0)   /* input pin trigger. */
                                                | SCT_EVENT_CTRL_IOSEL(APP_SCT_INPUT_LINE_HREF) /*
HREF  pin. */
                                                | SCT_EVENT_CTRL_IOCOND(1) /* pin rising trigger. */
                                                | SCT_EVENT_CTRL_COMBMODE(2) /* use io without
counter. */
                                                | SCT_EVENT_CTRL_STATELD(1) /* load state value. */
                                                |
SCT_EVENT_CTRL_STATEV(APP_SCT_STATE_WAIT_NEW_PCLK) /* new state. */
                                                | SCT_EVENT_CTRL_MATCHMEM(0) /* no use. */
                                                | SCT_EVENT_CTRL_DIRECTION(0) /* no use. */
                                                ;
    /* PCLK START event :
     * - APP_SCT_INPUT_LINE_PCLK occurs on PCLK input rising edge.
     * - switch to APP_SCT_STATE_WAIT_NEW_PCLK itself, wait for a new pixel.
     */
    SCT0->EVENT[APP_SCT_EVENT_PCLK_START ].CTRL = SCT_EVENT_CTRL_MATCHSEL(0) /* no use. */
                                                | SCT_EVENT_CTRL_HEVENT(0)   /* no use. */
                                                | SCT_EVENT_CTRL_OUTSEL(0)   /* input pin trigger. */
                                                | SCT_EVENT_CTRL_IOSEL(APP_SCT_INPUT_LINE_PCLK) /*
PCLK pin. */
                                                | SCT_EVENT_CTRL_IOCOND(1) /* pin rising trigger. */
                                                | SCT_EVENT_CTRL_COMBMODE(2) /* use io without
counter. */
                                                | SCT_EVENT_CTRL_STATELD(1) /* load state value. */
                                                |
SCT_EVENT_CTRL_STATEV(APP_SCT_STATE_WAIT_NEW_PCLK) /* new state. */
                                                | SCT_EVENT_CTRL_MATCHMEM(0) /* no use. */
                                                | SCT_EVENT_CTRL_DIRECTION(0) /* no use. */
                                                ;
    /* HREF END event :
     * - APP_SCT_INPUT_LINE_HREF occurs on HREF input falling edge.
     * - switch to APP_SCT_STATE_WAIT_NEW_LINE, wait for a new line.
     */
    SCT0->EVENT[APP_SCT_EVENT_HREF_END   ].CTRL = SCT_EVENT_CTRL_MATCHSEL(0) /* no use. */
                                                | SCT_EVENT_CTRL_HEVENT(0)    /* no use. */
                                                | SCT_EVENT_CTRL_OUTSEL(0)    /* input pin trigger. */
                                                | SCT_EVENT_CTRL_IOSEL(APP_SCT_INPUT_LINE_HREF) /*
HREF pin.*/
                                                | SCT_EVENT_CTRL_IOCOND(2) /* pin falling trigger. */
                                                | SCT_EVENT_CTRL_COMBMODE(2) /* use io without
counter. */
                                                | SCT_EVENT_CTRL_STATELD(1) /* load state value. */
                                                |
SCT_EVENT_CTRL_STATEV(APP_SCT_STATE_WAIT_NEW_LINE) /* new state. */
                                                | SCT_EVENT_CTRL_MATCHMEM(0) /* no use. */
                                                | SCT_EVENT_CTRL_DIRECTION(0) /* no use. */
                                                ;
    /* VSYNC END event :
     * - APP_SCT_INPUT_LINE_VSYNC occurs on VSYNC input rising edge.
     * - switch to APP_SCT_STATE_WAIT_NEW_FRAME, wait for a new frame.
     */
    SCT0->EVENT[APP_SCT_EVENT_VSYNC_END  ].CTRL = SCT_EVENT_CTRL_MATCHSEL(0) /* no use. */
                                                | SCT_EVENT_CTRL_HEVENT(0)    /* no use. */
                                                | SCT_EVENT_CTRL_OUTSEL(0)    /* input pin trigger. */
                                                | SCT_EVENT_CTRL_IOSEL(APP_SCT_INPUT_LINE_VSYNC) /*
VSYNC pin.*/
                                                | SCT_EVENT_CTRL_IOCOND(1) /* pin rising trigger. */
                                                | SCT_EVENT_CTRL_COMBMODE(2) /* use io without
counter. */
                                                | SCT_EVENT_CTRL_STATELD(1) /* load state value. */
                                                |
SCT_EVENT_CTRL_STATEV(APP_SCT_STATE_WAIT_NEW_FRAME) /* new state */
                                                | SCT_EVENT_CTRL_MATCHMEM(0) /* no use. */
                                                | SCT_EVENT_CTRL_DIRECTION(0) /* no use. */
                                                ;
```

Setup the events about in which states they can survive:

```
  /* setup the enabled event in each state. */
    /* APP_SCT_EVENT_VSYNC_START event is used in these states :
     * - APP_SCT_STATE_WAIT_NEW_FRAME
     * - APP_SCT_STATE_WAIT_NEXT_FRAME (optional)
     */
    SCT0->EVENT[APP_SCT_EVENT_VSYNC_START].STATE = (1U << APP_SCT_STATE_WAIT_NEW_FRAME )
                                                 | (1U << APP_SCT_STATE_WAIT_NEXT_FRAME)
                                                 ;
    /* APP_SCT_EVENT_HREF_START is used in these states :
     * - APP_SCT_STATE_WAIT_NEW_LINE
     */
    SCT0->EVENT[APP_SCT_EVENT_HREF_START].STATE  = (1U << APP_SCT_STATE_WAIT_NEW_LINE  )
                                                 ;
    SCT0->EVENT[APP_SCT_EVENT_PCLK_START].STATE  = (1U << APP_SCT_STATE_WAIT_NEW_PCLK  )
                                                 ;
    /* APP_SCT_EVENT_VSYNC_END event can be available in all state.
     * It would switch to APP_SCT_STATE_WAIT_NEW_FRAME.
     */
    SCT0->EVENT[APP_SCT_EVENT_VSYNC_END  ].STATE = (1U << APP_SCT_STATE_WAIT_NEW_FRAME )
                                                 | (1U << APP_SCT_STATE_WAIT_NEW_LINE  )
                                                 | (1U << APP_SCT_STATE_WAIT_NEW_PCLK  )
                                                 | (1U << APP_SCT_STATE_WAIT_NEXT_FRAME)
                                                 ;
    /* APP_SCT_EVENT_VSYNC_END only occus in APP_SCT_STATE_WAIT_NEW_FRAME state. */
    SCT0->EVENT[APP_SCT_EVENT_HREF_END  ].STATE  = //(1U << APP_SCT_STATE_WAIT_NEW_FRAME )
                                                   //(1U << APP_SCT_STATE_WAIT_NEW_LINE  )
                                                   (1U << APP_SCT_STATE_WAIT_NEW_PCLK  )
                                                 //| (1U << APP_SCT_STATE_WAIT_NEXT_FRAME)
```

5 Demo show

In the actual demo, an OLED screen is used to display the camera sensing image in runtime. When running the demo, an arrow is drawn as the visual target. In Figure 2, it can be seen that an image with the target arrow is captured from camera and displayed in the OLED screen.

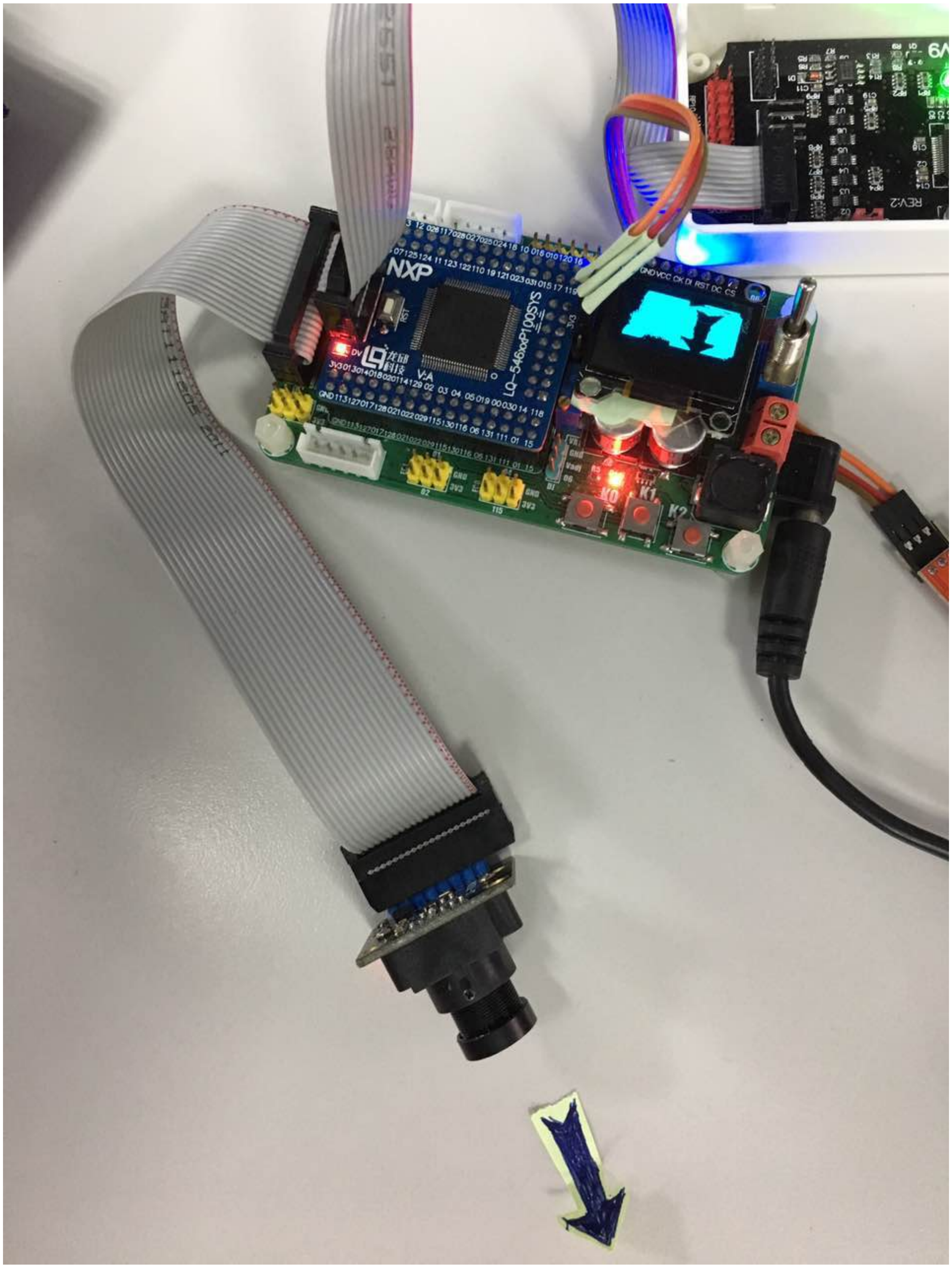

Figure 2. LPC5460x camera demo

6 Revision history

Table 2 summarizes the changes done to this document since the initial release.

Table 2. Revision history

Revision number	Date	Substantive changes
0	09/2019	Initial release

Date of release: 16 September 2019
Document identifier: AN12583