HDMI-CEC Application Note & Sample Code

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	This application note describes how to implement HDMI (high-definition multimedia interface) CEC (consumer electronics control) functions as a TV set or a projector that support the HDMI-CEC protocol based on the Timer Capture function of NuMicro® family Cortex®-M0 M051 and M0518 series microcontrollers (MCUs).
Apply to	NuMicro® M051 and M0518 series which support the Timer Capture function.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

List of Figures

List of Tables

1 Overview

This application note describes how to implement HDMI (high-definition multimedia interface) CEC (consumer electronics control) functions as a TV set or a projector that support the HDMI-CEC protocol based on the Timer Capture function of NuMicro® family Cortex®-M0 M051 and M0518 series microcontroller (MCU). At first, the abstract of HDMI-CEC structure and format from the HDMI-CEC specification will be introduced. Then, the Timer Capture function of M051 series and M0518 series will be briefly explained. Finally, the example code and the results using a tiny EVB board of M0516LDN or M0518SD2AE chip will be present to demonstrate the HDMI-CEC behavior, and emulated as a TV set or a projector connected with SONY PS3, Philips DVD player and Google Chromecast.

1.1 The Purpose of HDMI-CEC

Figure 1-1 shows the home entertainment structure in universal home. The purpose of CEC is to reduce the remote controllers to control all the HDMI devices connected together through the HDMI cables serially.

Figure 1-1 Home Entertainment Structure

2 HDMI-CEC Specification Abstract

2.1 Introduction to CEC

CEC is the appendix supplement 1 to the HDMI standard. It is a protocol that provides high-level control functions between all of the various audiovisual products in a user's environment. For more details, refer to the "High-Definition Multimedia Interface" Specification that is available from www.hdmi.org.

2.2 CEC Communication

2.2.1 Address

2.2.1.1 Physical Address

To allow CEC to be able to address specific physical devices and control switches, all devices shall have a physical address. This connectivity has to be worked out whenever a new device is added to the cluster. The physical address discovery process uses only the DDC/EDID (Display Data Channel/Extended Display Identification Data) mechanism and can be applied to all HDMI Sinks and Repeaters, not only to CEC-capable devices. The CEC and DDC connections are shown in Figure 2-1.

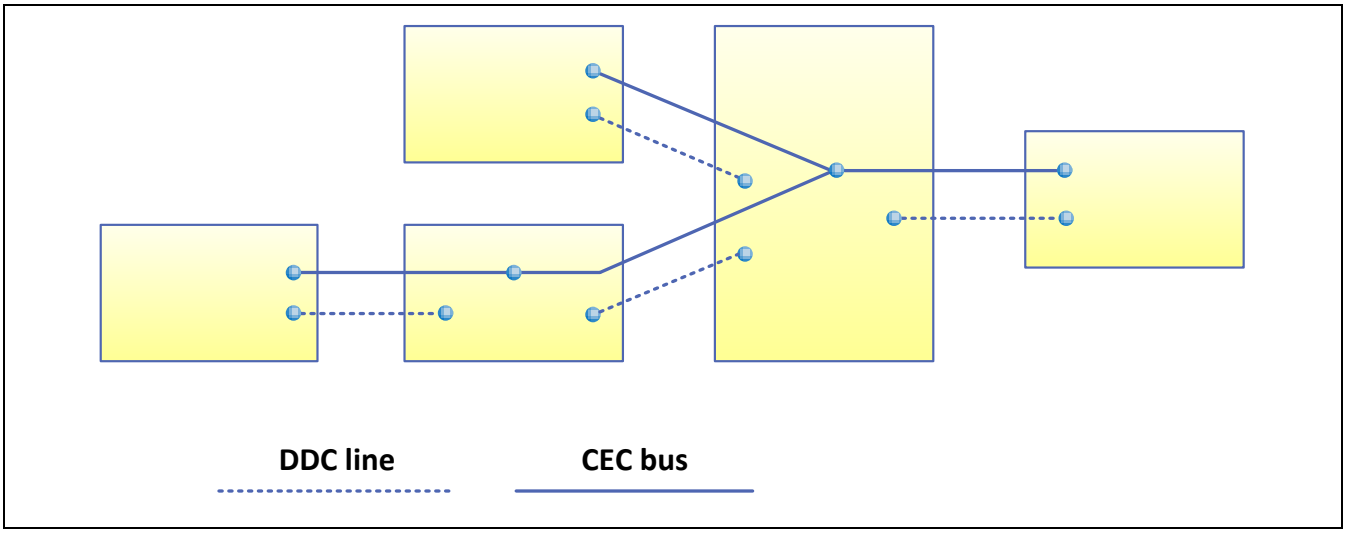

Figure 2-1 CEC and DDC Line Connections

The CEC line is directly connected to all nodes on the network. After discovering their own physical address, the CEC devices transmit their physical and logical addresses to all other devices, thus allowing any device to create a map of the network.

Physical Address Discovery

The physical address of each node is determined through the physical address discovery process. This process is dynamic in that it automatically adjusts physical addresses as required as devices are physically or electrically added or removed from the device tree.

All Sinks and Repeaters shall perform the steps of physical address discovery and propagation even if those devices are not CEC-capable. Sources are not required to determine their own physical address unless they are CEC-capable.

All addresses are 4 digits long allowing for a 5–device-deep hierarchy. All are identified in the form of n.n.n.n in the following description. An example of this is given in Figure 2-1.

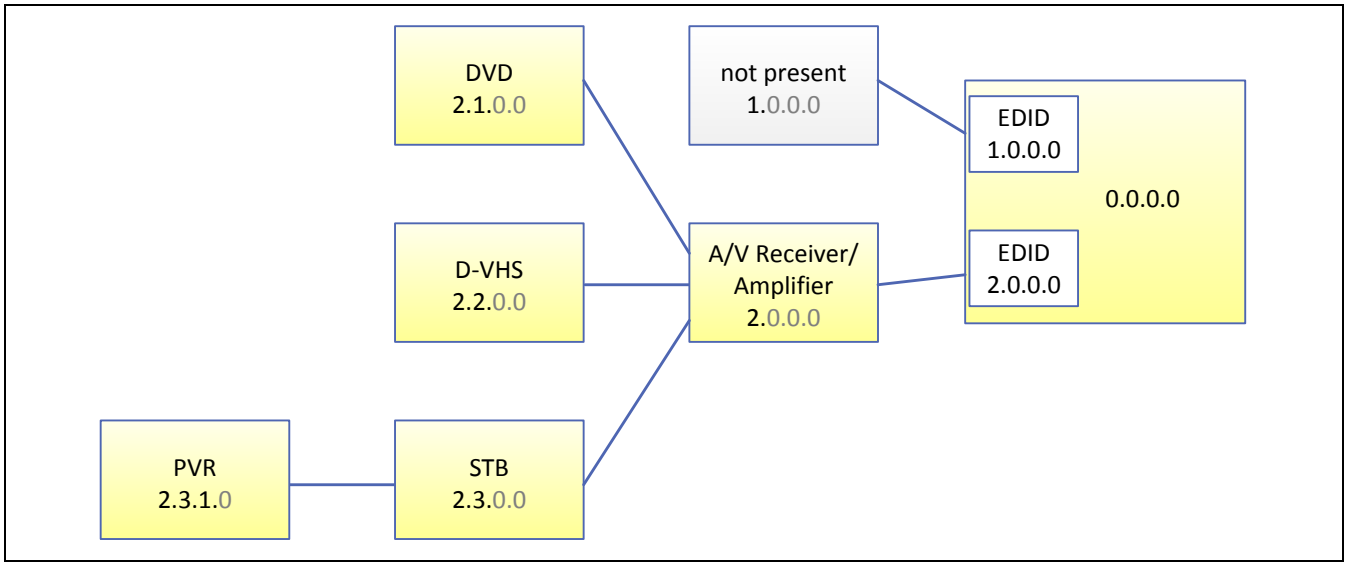

Figure 2-2 Physical Addresses within an HDMI Cluster

2.2.1.2 Logical Address

Each device appearing on the control signal line has a logical address which is allocated to only one device in the system. This address defines a device type as well as being a unique identifier. These are specified in Table 2-1.

Logical Address	Device
0	TV
1	Recording Device 1
2	Recording Device 2

3	Tuner 1
4	Playback Device 1
5	Audio System
6	Tuner 2
7	Tuner 3
8	Playback Device 2
9	Recording Device 3
10	Tuner 4
11	Playback Device 3
12	Reserved
13	Reserved
14	Free Use
15	Unregistered (as initiator address) Broadcast (as destination address)

Table 2-1 Logical Address

All CEC devices have both a physical and logical address, whereas non-CEC devices only have a physical address. The following diagram shows the physical and logical addressing within a HDMI cluster.

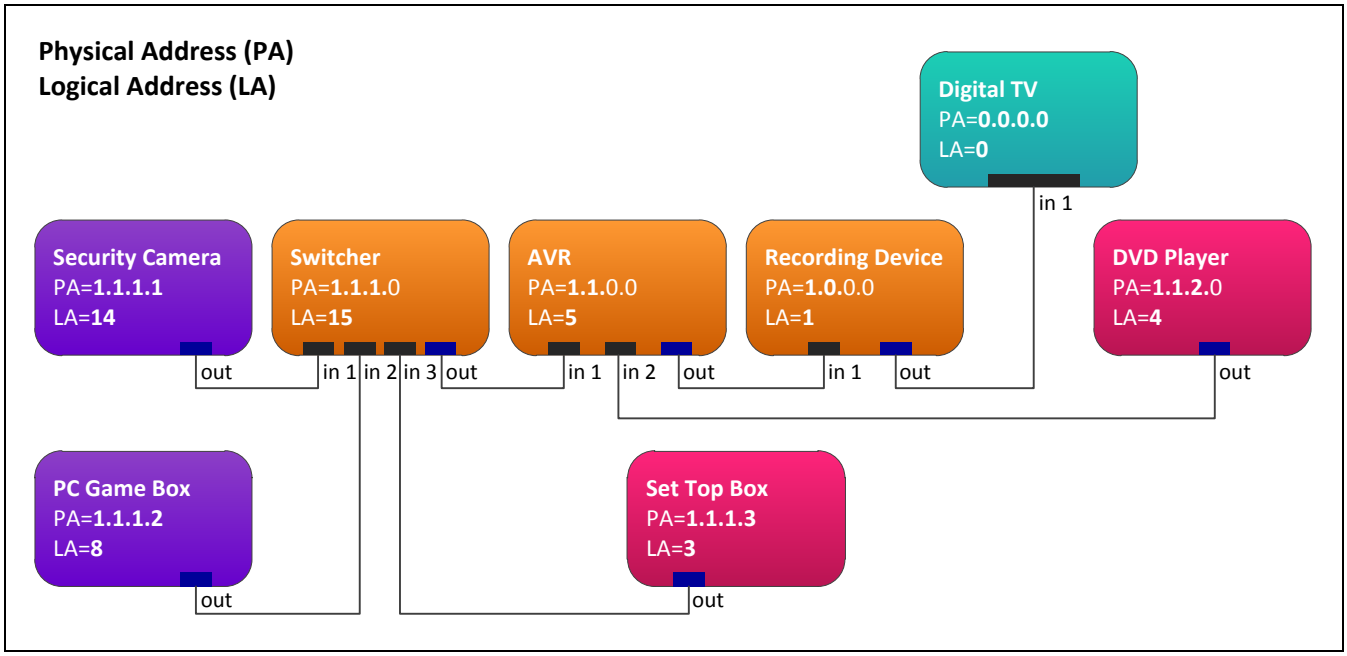

Figure 2-3 Physical and Logical Addressing within a HDMI Cluster

2.2.2 CEC Format

2.2.2.1 CEC Message

Each CEC feature contains one or more messages to perform a special CEC feature, for example, a “One Touch Play” feature described in the section 2.2.3.1 that it needs two messages to achieve it. The following table shows the complete CEC messages in a CEC feature that communication on a CEC bus. Each message is a single frame; the details of each block of the frame are given in the subsequent sections. The maximum message size (header block plus opcode block plus operand blocks) is 16 * 10 bits and message structure is shown below.

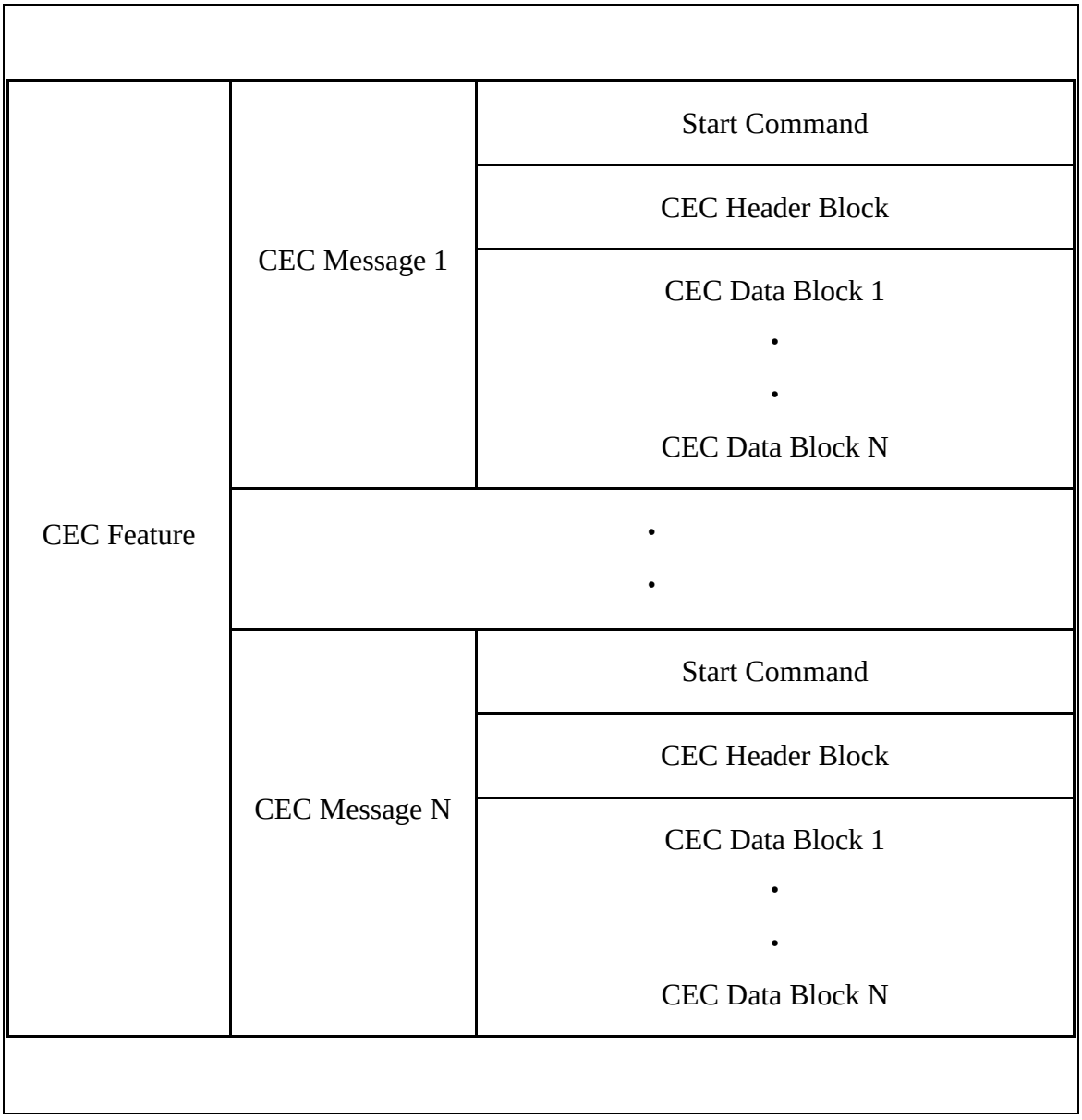

Figure 2-4 CEC Messages in a CEC Feature

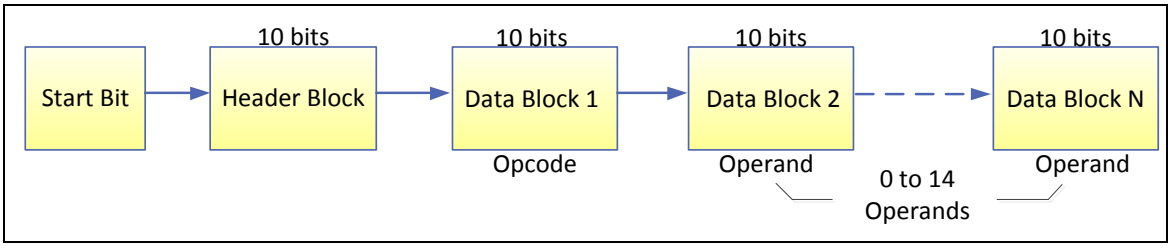

Figure 2-5 Message Structure

2.2.2.2 CEC Frame

The CEC bus is a single wire protocol that can connect up to 10 audiovisual devices through standard HDMI cable. The CEC transaction is made up of a start bit, a 10-bit header and a sequence of n 10-bit data blocks. The Header block and the Data blocks each contain an end-of-message (EOM) bit and an acknowledge (ACK) bit.

Data Block

All Data Blocks and Header Blocks are ten bits long and have the same basic structure, as shown in Figure 2-6 . The information bits are data, opcodes or addresses, depending on context. The control bits, EOM and ACK, are always present and always have the same usage.

Data Block									
7	6	5	4	3	2	1	0	--	--
Informational Bits								EOM	ACK

Figure 2-6 Data Block Format

EOM (End of Message)

= 0, one or more data blocks follow.

= 1, the message is complete.

ACK (Acknowledge)

= 0, Ack.

Header Block

The Header Block consists of the source logical address field, the destination logical address field, the end of message bit (EOM) and the acknowledge bit (ACK) as shown in Figure 2-7. The addresses for the devices are specified in Table 2-1.

Header Block									
3	2	1	0	3	2	1	0	--	--
Initiator Address				Destination Address				EOM	ACK

Figure 2-7 Header Block Format

2.2.2.3 CEC Timing

Start Bit Timing

The pulse format of the start bit is shown in Figure 2-8. It is unique and identifies the start of a frame. The start bit has to be validated by its low duration (a) and its total duration (b).

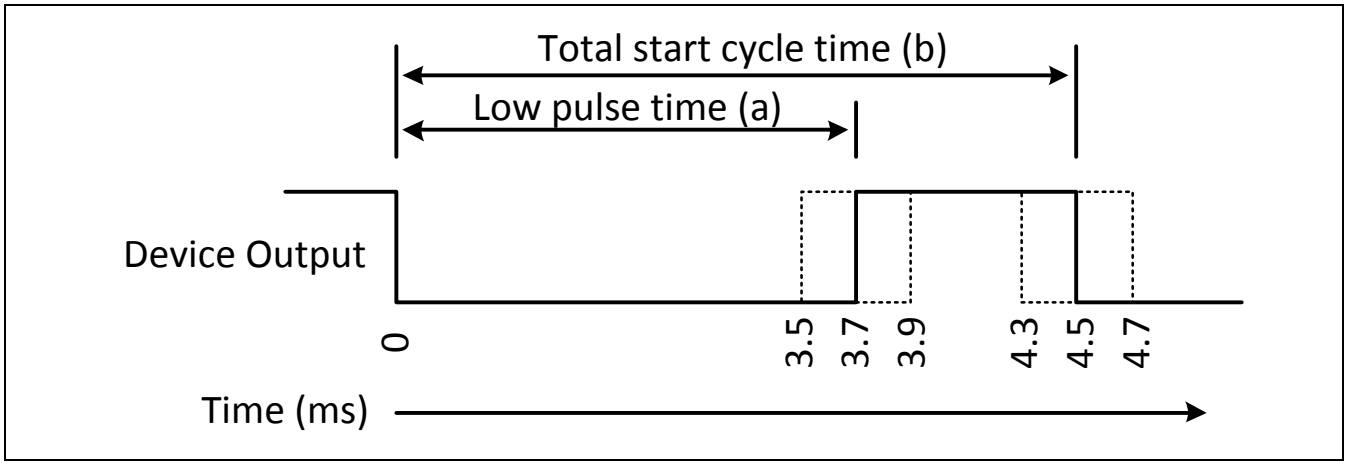

Figure 2-8 Start Bit Timing

Data Bit Timing

All remaining data bits in the frame, after the start bit, have consistent timing. There are, however, two types of bits; an initiator asserted bit and a follower asserted bit. All bits apart from the acknowledge bit are asserted by the initiator. Figure 2-9 shows both logical 1 and logical 0 timing diagrams for an initiator asserted bit.

The high to low transition at the end of a data bit is the start of the next data bit and only occurs if there is a following data bit. After transmission of the final bit the CEC line remains high.

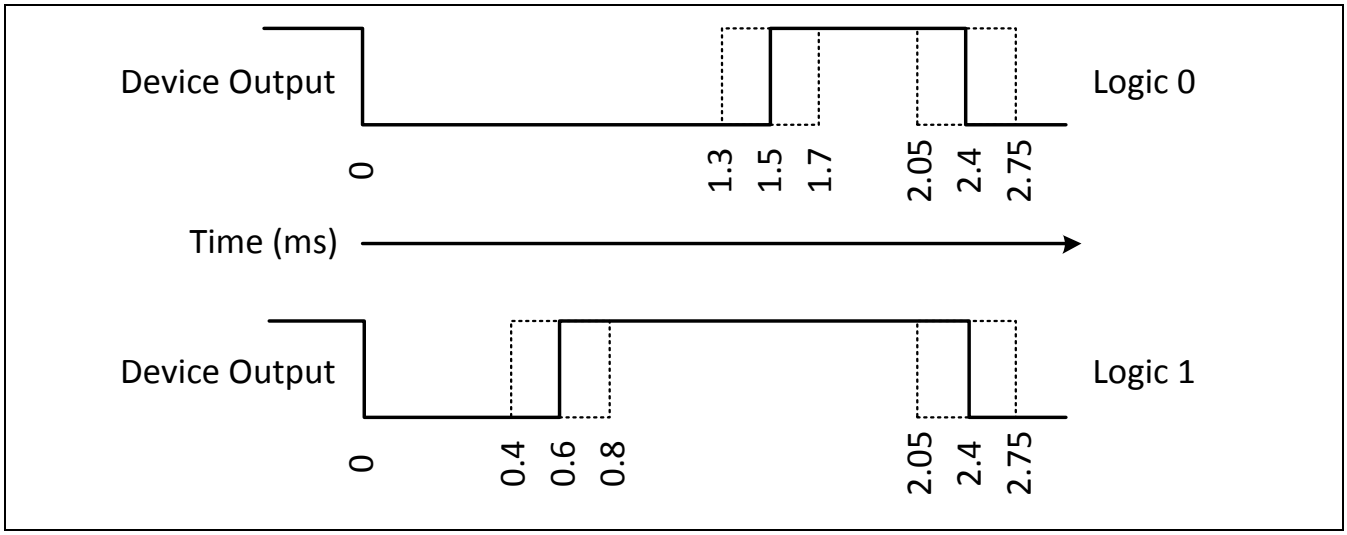

Figure 2-9 Data Bit Timing

Asserted Bit Timing

Figure 2-10 shows an example bit with both the initiator and follower where the follower may assert the bit to a logical 0 to acknowledge a data block. The initiator outputs a logical 1, thus allowing the follower to change the CEC state by pulling the control line low for the duration of the safe sample period.

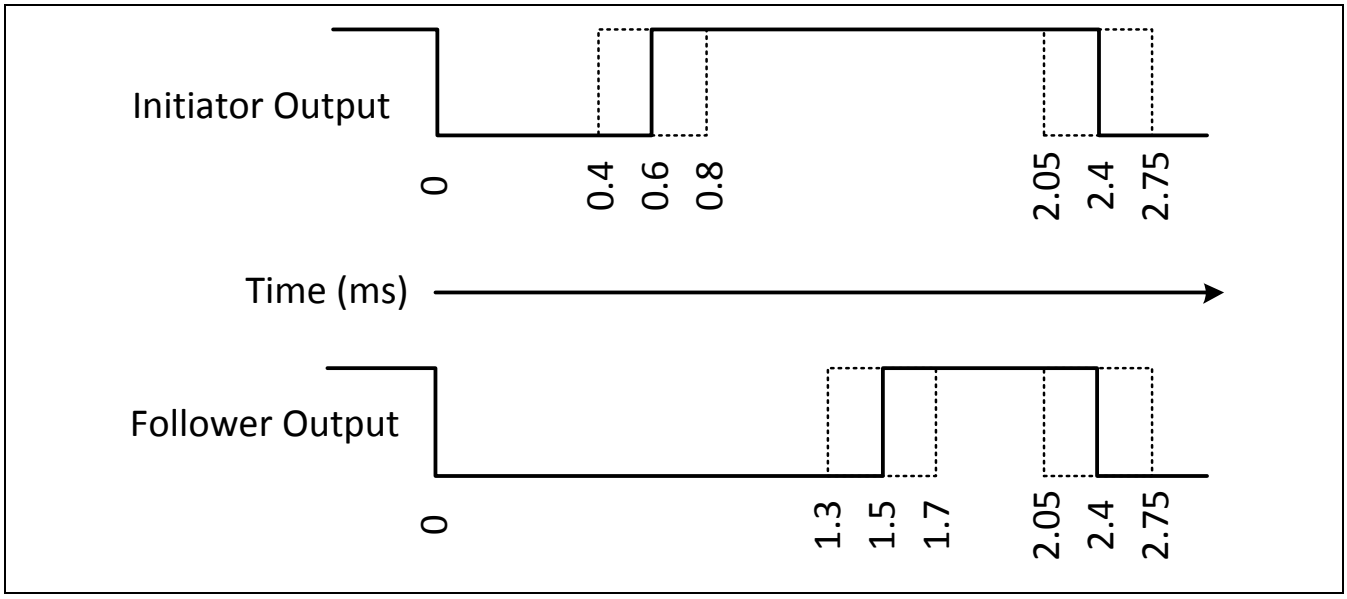

Figure 2-10 Asserted Bit Timing

2.2.3 CEC Features

This section describes the message transfer and additional details for a number of common features enabled by CEC. Note that where a feature is supported, all messages within that feature should be implemented. This document only describes two utility CEC features and the other features that user can refer to the related HDMI-CEC specification.

2.2.3.1 One Touch Play

The One Touch Play feature allows a device to be played and become the active source with a single button press.

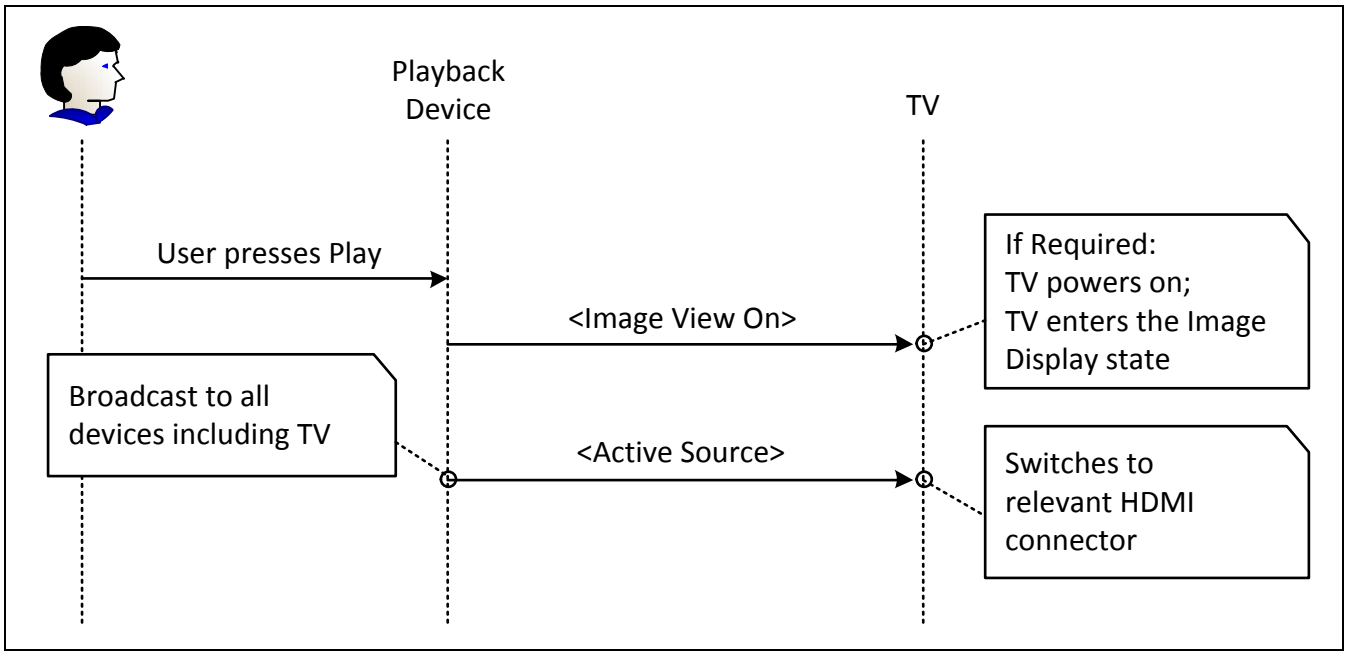

Figure 2-11 Typical Scenario for One Touch Play Feature

2.2.3.2 System Standby

There are two kinds of standby functions in CEC system standby feature, one is the broadcast standby and the other is device standby. The following message is used for the System Standby feature:

Broadcast Standby

The broadcast message <Standby> can be used to switch all CEC devices to standby. A typical scenario where the user sets the whole system to standby is shown below.

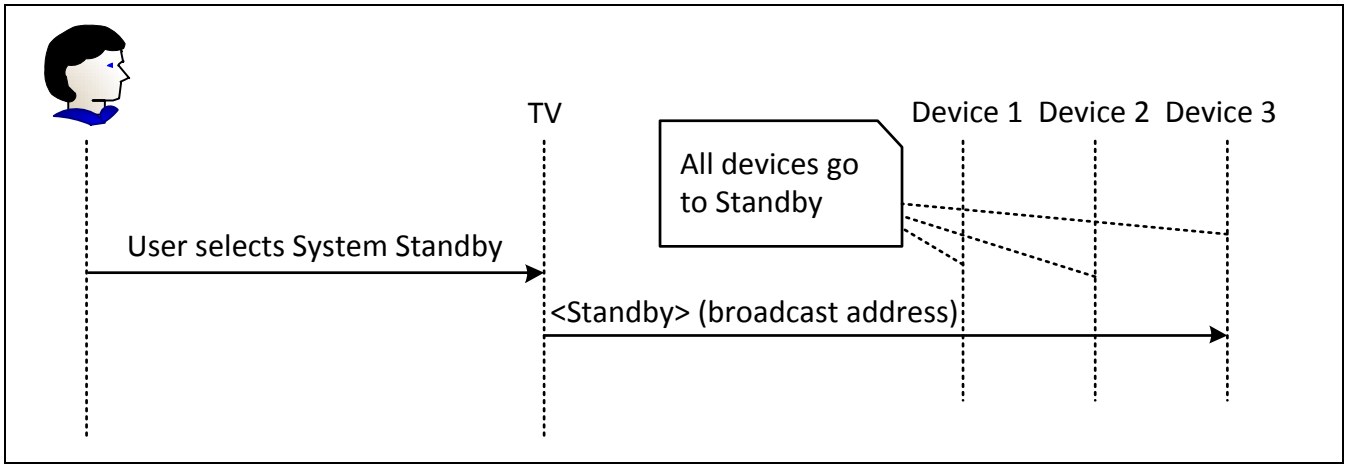

Figure 2-12 Typical Scenario for Broadcast (System) Standby Feature

Device Standby

A device can switch another single device into standby by sending the message <Standby> as a directly addressed message to it.

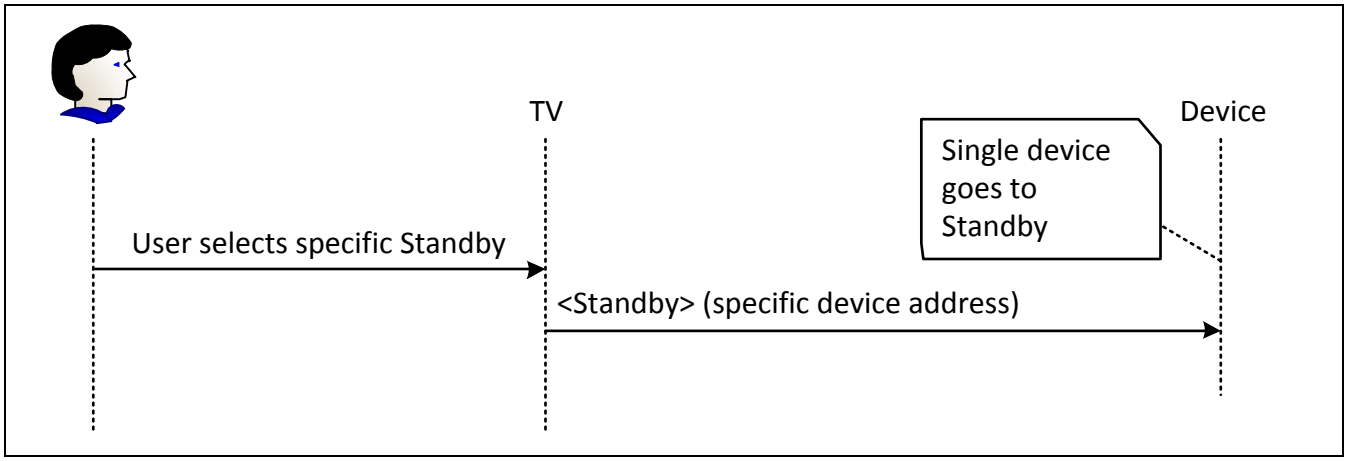

Figure 2-13 Typical Scenario for Standby Feature to a Specific Device

3 Timer Capture Function

3.1 Timer Controller

In the NuMicro® family M051 and M0518 series, the Timer Controller includes four 32-bit timers, TIMER0 ~ TIMER3, allowing user to easily implement a timer control for applications. The timer can perform functions, such as frequency measurement, delay timing, clock generation, event-counting by external input pins, and interval measurement by external capture pins.

3.1.1 Timer Controller of M051 Series

The clock sources and block diagram of Timer Controller in the M051 series are shown as follows.

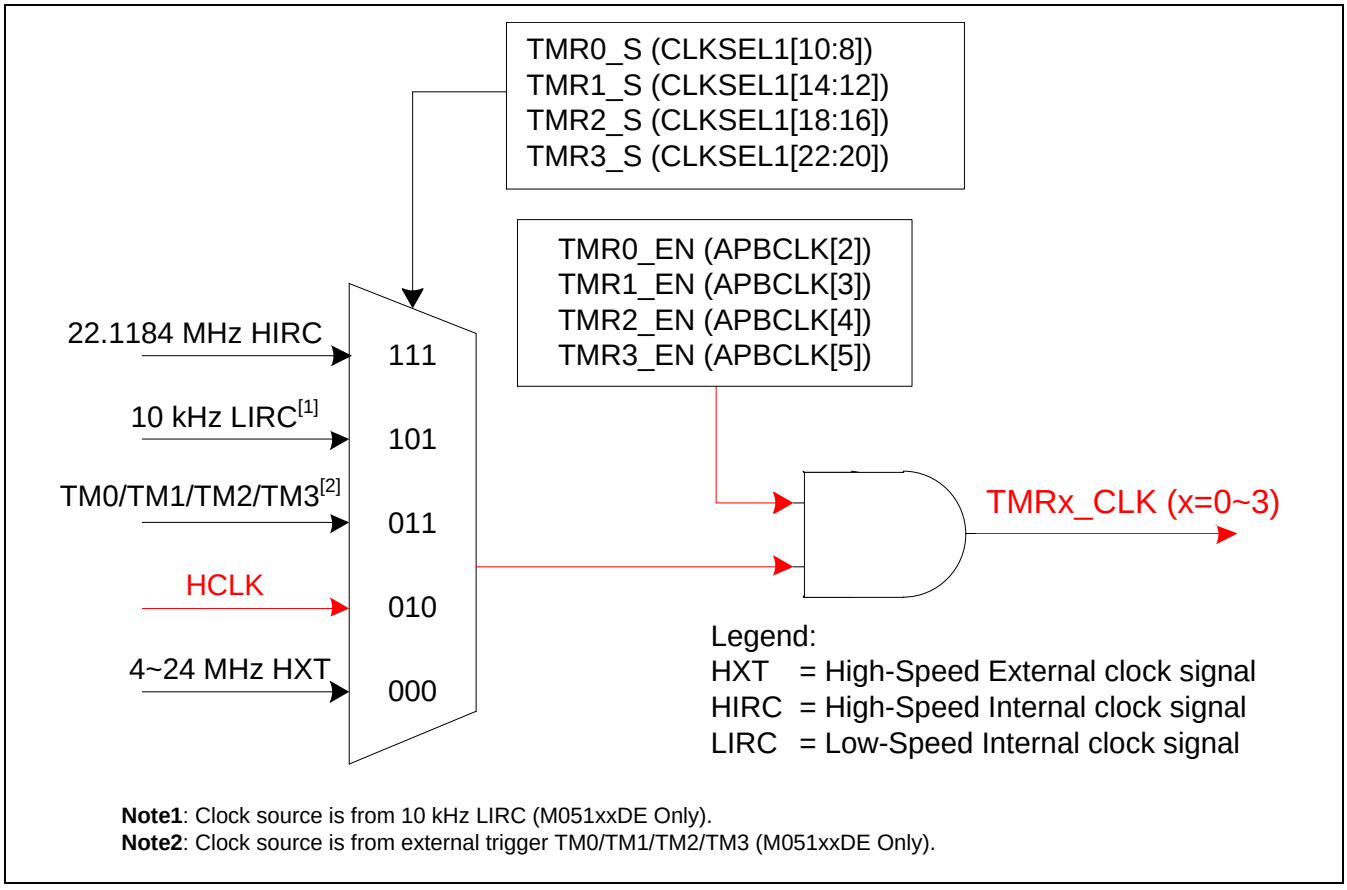

Figure 3-1 Clock Source of Timer Controller in M051 Series

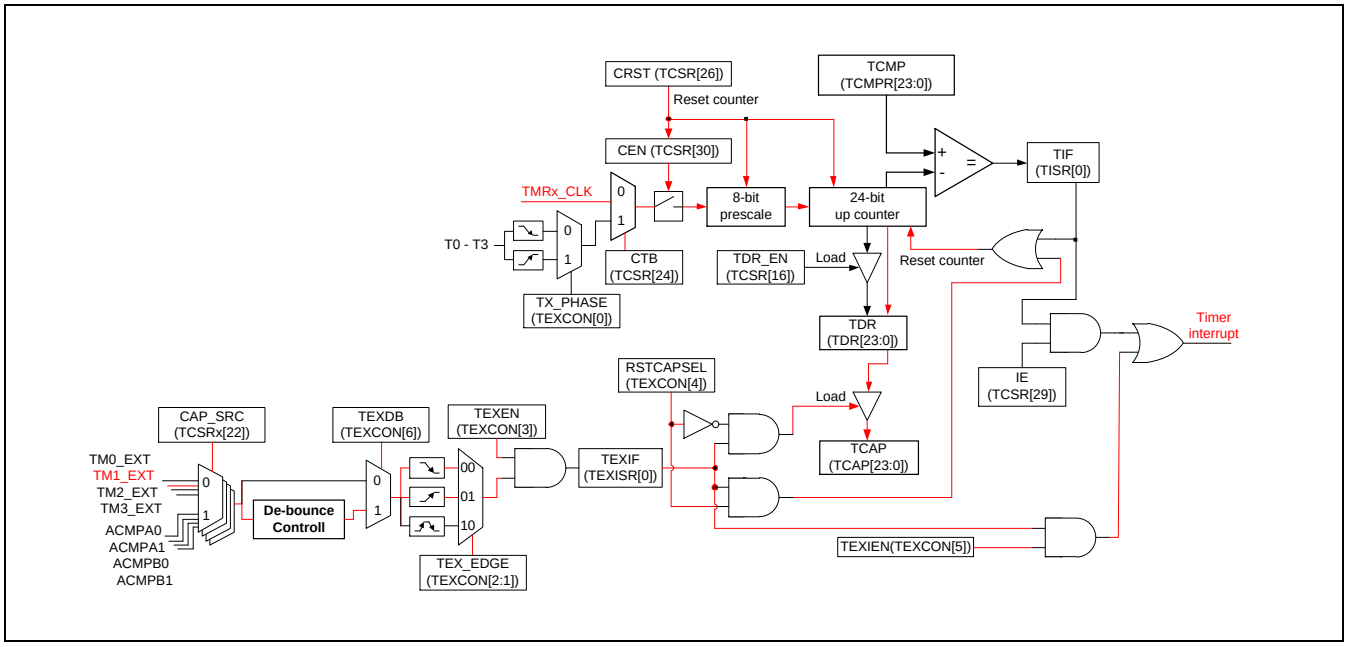

Figure 3-2 Block Diagram of Timer Controller in M051 Series

3.1.2 Timer Controller of M0518 Series

The clock sources and block diagram of Timer Controller in the M0518 series are shown as follows.

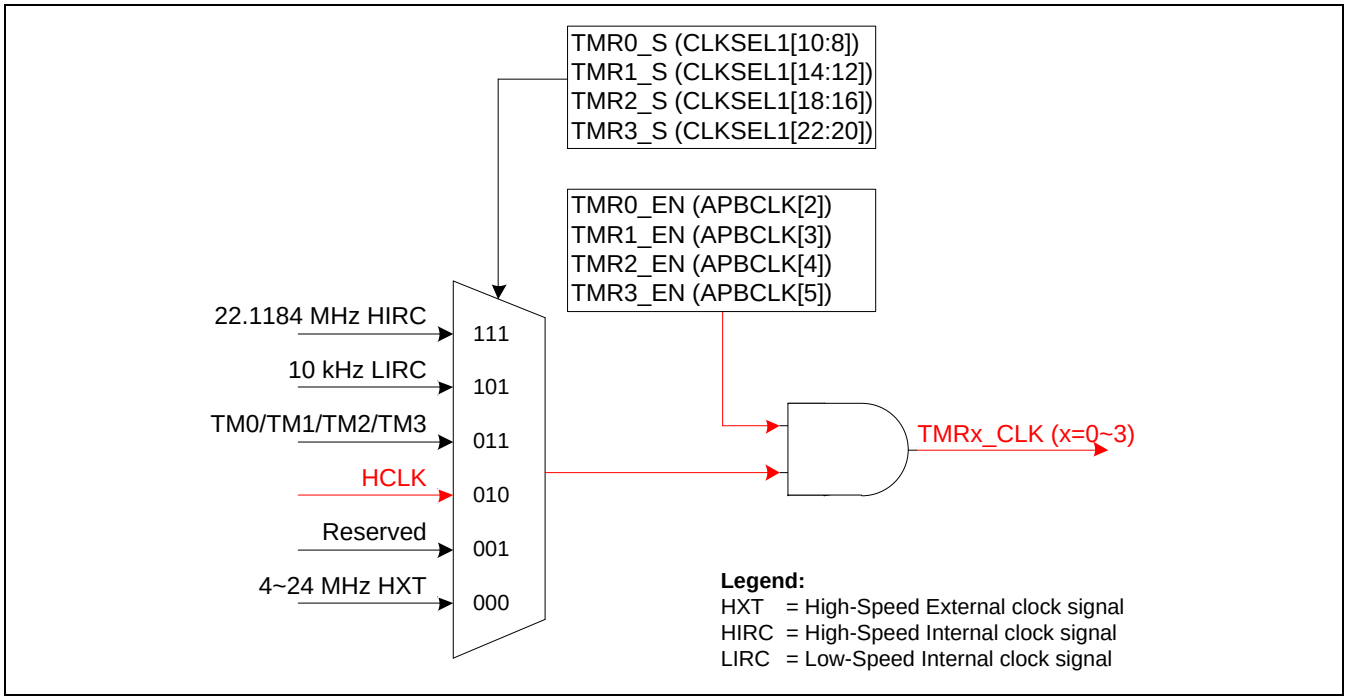

Figure 3-3 Clock Source of Timer Controller in M0518 Series

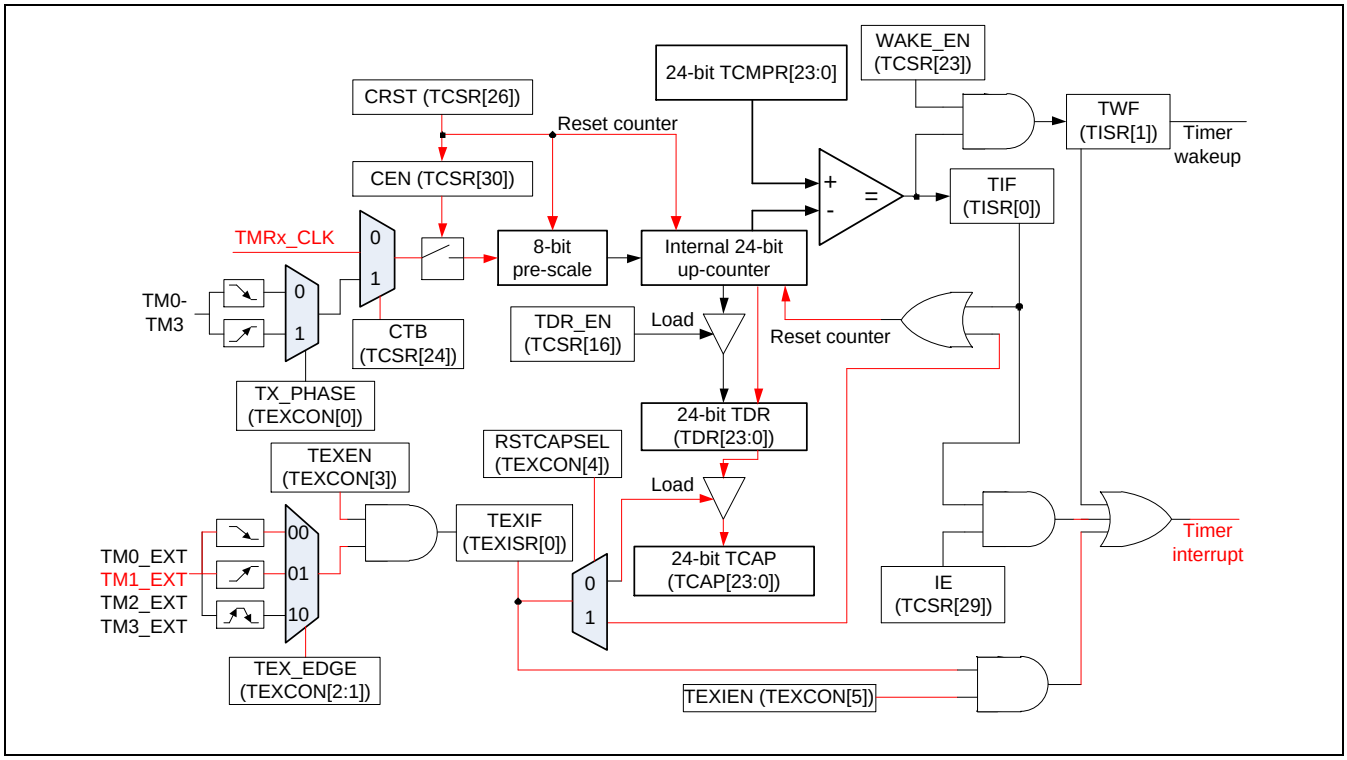

Figure 3-4 Block Diagram of Timer Controller in M0518 Series

3.2 Timer Event Capture Function

The Timer Event Capture function is just one of Timer functions and used to capture the Timer Data Register (TDR) value to Timer Capture Data Register (TCAP) while the edge transition is detected on TMx_EXT pin (x= 0~3). In this mode, RSTCAPSEL (TEXCON[4]) bit should be set as 0 to select the transition edge at TMx_EXT pin when using this event capture function, and the timer peripheral clock source should be set as HCLK (TMRx_S[2:0] = 2 in clock control register CLKSEL1).

The TMx_EXT pin de-bounce circuit can be enabled or disabled by TEXDB (TEXCON[6]) bit through software. The transition frequency of TMx_EXT pin should be less than 1/3 HCLK if the TMx_EXT pin de-bounce is disabled or less than 1/8 HCLK if TMx_EXT pin de-bounce is enabled to assure the capture function can be work normally. User can also select edge transition detection at TMx_EXT pin by TEX_EDGE (TEXCON[2:1]) bits.

In event capture mode, user does not need to know which timer counting operation mode is selected in software. The capture event occurred only if edge transition on TMx_EXT pin is detected.

For more details about Timer Controller and the related control registers of M051 series or M0518 series, refer to the TRM (Technical Reference Manual) specification documents that are available from Nuvoton website: www.nuvoton.com/hq/products/microcontrollers/arm-cortex-m0-mcus/Technical-Reference-Manual?__locale=en.

3.3 Use the Timer Capture to Measure the CEC Signal

The example code uses the Timer 1 event capture hardware logical circuit to measure the low pulse duration of CEC signal from the TM1_EXT input pin, and judges this low pulse duration whether it matches the timing of start bit, logic 0 or logic 1 for each data bit of header block and data block. The low pulse duration is captured into TCAP register by Timer 1 controller hardware shown as the red lines from Figure 3-1 to Figure 3-4, which the clock source of Timer 1 controller is HCLK and when there are any edge-transition detected at the TM1_EXT pin and this event also generates the Timer 1 interrupt under the TEXIEN bit is enabled.

Figure 3-5 shows the timing waveform for Timer 1 to reset or capture the internal counter at the falling-edge or rising-edge of input CEC signal.

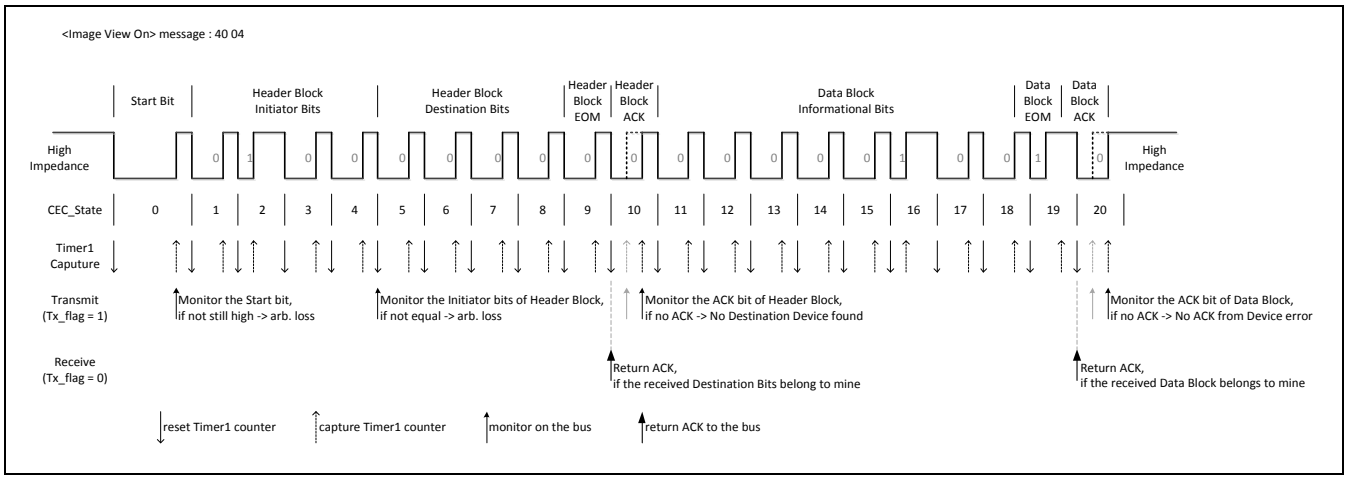

Figure 3-5 Timing Waveform of Timer 1 Capturing

4 Example Code

4.1 Hardware System Structure

To demonstrate the CEC functions on the M051 series and M0518 series with SONY PS3, Philips DVD or Google Chromecast, one of the two tiny EVB boards, NuTiny-SDK-M051 (M0516LDN) or NuTiny-SDK-M0518 (M0518SD2AE), and a HDMI Splitter will be used to develop the hardware system. The system structure for the demonstration is shown as follows.

At first, the CEC (pin 13) line of HDMI connector is connected to the two GPIO1 and TM1_EXT pins of tiny EVB board. Also, the Ground (pin 17) line of HDMI connector is connected to the GND pin of tiny EVB board. An extra 2.7 kΩ resistor is necessary to pull-up the CEC line to 3.3V. The GPIO2 controls the LED on/off state to indicate whether this chip had received the <Image View On> or <Text View On> message of “One Touch Play” feature from the PS3 or DVD Player or Chromecast. The key button is an external interrupt input to trigger this chip to send out the <Device Standby> message of “System Standby” feature to lead the relative device to enter the standby state.

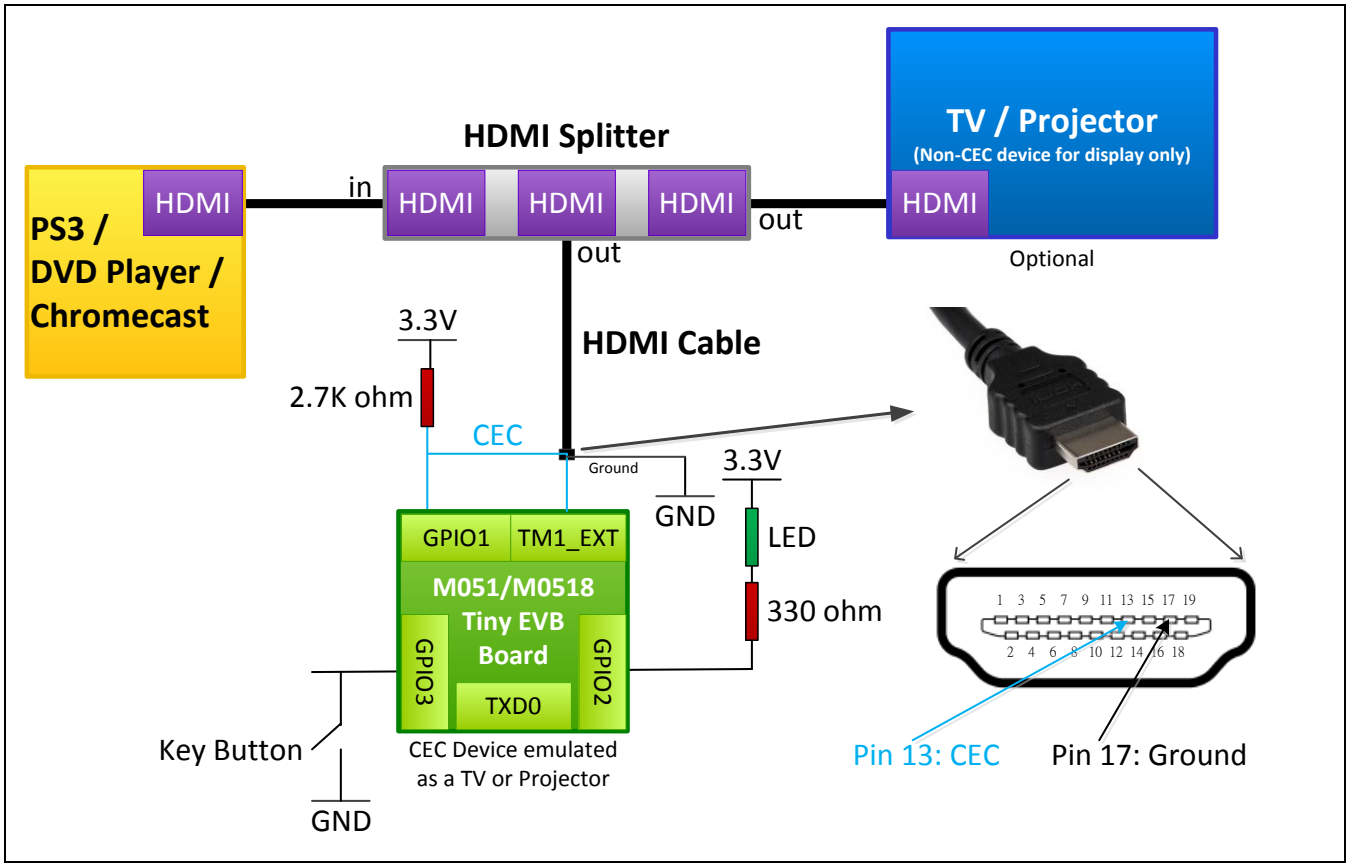

Figure 4-1 Hardware System Structure

Table 4-1 shows the pin function, GPIO mode and pin number respectively on M051-LQFP48 or M0518-LQFP64 chip for each pin used in the demonstration of CEC.

Pin Name	Function Description	GPIO Mode	M051 LQFP48 Pin No.	M0518 LQFP64 Pin No.
TM1_EXT	Timer 1 capture input to measure the low pulse width of CEC signal on CEC bus	Input	P3.3	PE.5
GPIO1	To output CEC protocol to CEC bus	Open-Drain	P3.2	PB.11
GPIO2	To turn on/off LED to pretend TV or projector on or off state	Output	P3.6	PB.13
GPIO3	Key button interrupt input to trigger chip to send out the “Device Standby” message	Quasi-bidirectional	P4.3	PB.10
TXD0	To show the message on PC through the COM port	Output	P3.1	PB.1

Table 4-1 Pin Function and GPIO Mode

For more details about the tiny EVB boards of M051 or M0518 series, refer to the User Manual documents and PCB files that are available from Nuvoton website: www.nuvoton.com/hq/support/tool-and-software/development-tool-hardware/development-kit/?__locale=en

4.2 Firmware Description

In this firmware description, the M0518 series will be used as an example for interpretation. The M0518 CEC firmware source code, like a TV set or a projector that support the HDMI-CEC protocol, is created based on the M0518 series BSP V3.00.002. User can download the last version of M0518 series BSP from Nuvoton website: www.nuvoton.com/hq/support/tool-and-software/software/?__locale=en

4.2.1 Main Function and System Initialization

4.2.1.1 Main Function

The source code is used to define the parameters of CEC timing and the general CEC commands, declare the variables and do the initialization for system clock, UART 0, GPIO and Timers. Finally, the code is waiting for a key button interrupt to send out the “Device Standby” command on the CEC bus to lead the device to enter the standby state. Otherwise, it shows the CEC messages if they are received from or monitored on the CEC bus that the messages are sent by the other device. Figure 4-2 shows the main function flow chart.

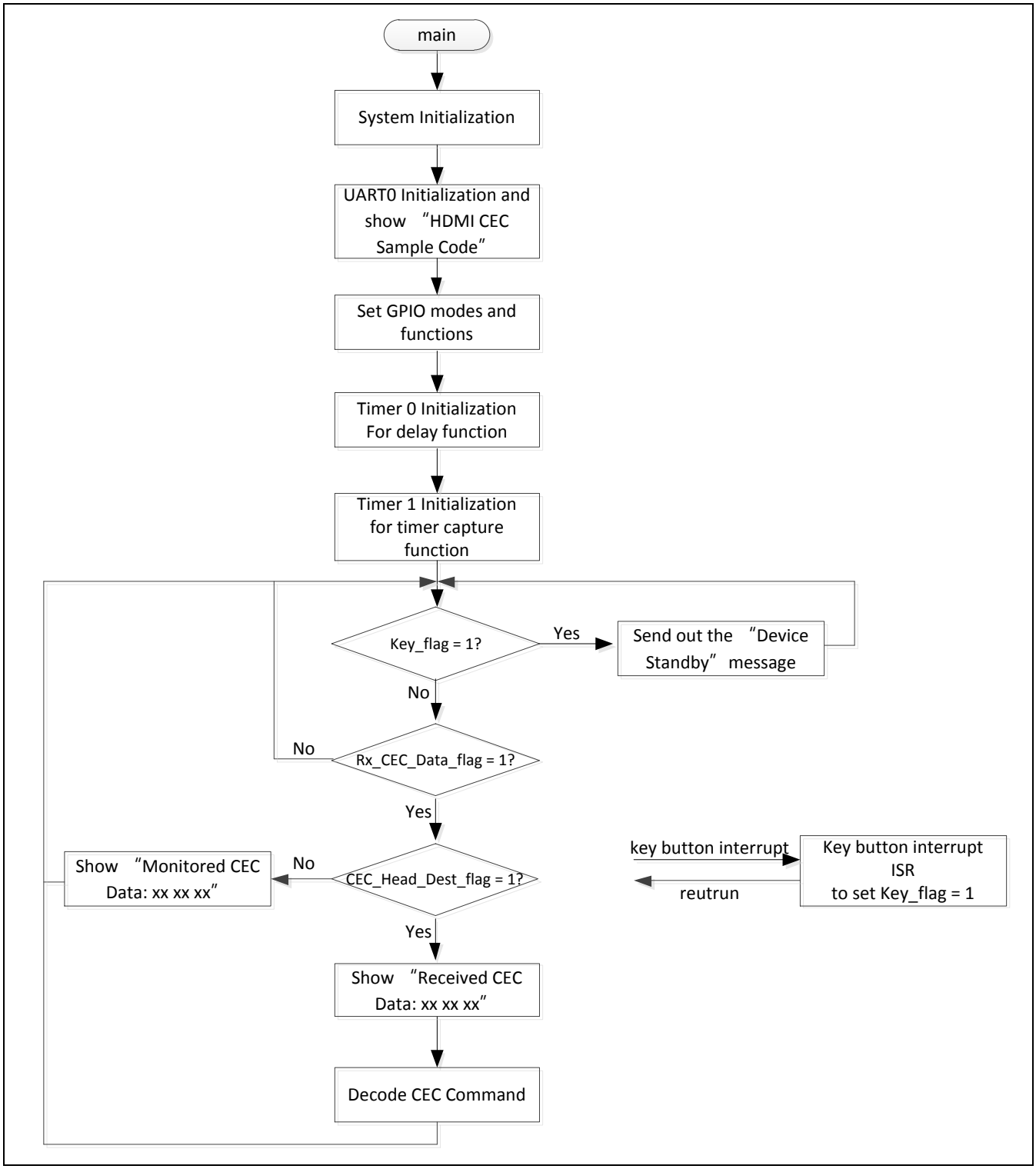

Figure 4-2 Main Function Flow Chart

```
#include <stdio.h>
#include "M0518.h"
#include "cec_ctrl.h"

#define     PLL_CLOCK  48000000

/* Definition for CEC */
// Bit Timing
#define          CEC_START_BIT_L       37     // 37 x 100us = 3.7ms
#define          CEC_START_BIT_H        8     //  8 x 100us = 0.8ms
#define          CEC_BIT_1_L                6     //  6 x 100us = 0.6ms
#define          CEC_BIT_1_H               18     // 18 x 100us = 1.8ms
#define          CEC_BIT_0_L               15     // 15 x 100us = 1.5ms
#define          CEC_BIT_0_H                9     //  9 x 100us = 0.9ms
#define          CEC_MAX_RETRY              5     // Retry
#define          CEC_FREE_TIME_PI      7*24   // Present Initator wants to send
another frame immediately    after its previous frame
#define          CEC_FREE_TIME_NI      5*24   // New Initator wants to send a frame
#define          CEC_FREE_TIME_RS      3*24   // Previous attempt to send frame
unsuccessful
// Destination of Header Block
#define          CEC_HEAD_DEST              0x00   // Header Block Destination
// Tolerance
#define     Percent                         0.05   // 5 %
#define     Percent0                  0.13   // 13 %
#define     Percent1                  0.33   // 33 %
// Start Bit range
#define     STB_MAX                         (CEC_START_BIT_L * 4800 * (1 + Percent))

#define     STB_MIN                         (CEC_START_BIT_L * 4800 * (1 - Percent))

// Bit = 0 range
#define     BIT_0_MAX                 (CEC_BIT_0_L * 4800 * (1 + Percent0))
#define     BIT_0_MIN                 (CEC_BIT_0_L * 4800 * (1 - Percent0))
// Bit = 1 range
#define     BIT_1_MAX                 (CEC_BIT_1_L * 4800 * (1 + Percent1))
#define     BIT_1_MIN                 (CEC_BIT_1_L * 4800 * (1 - Percent1))

/* Definition for CEC Commands */
#define     Give_Physical_Address       0x83
#define     Report_Physical_Address     0x84
#define     Give_Device_Power_Status    0x8f
```

```
#define     Report_Power_Status         0x90
#define     Active_Source                     0x82
#define     Inactive_Source             0x9d
#define     Image_View_On                     0x04
#define     Text_View_On                      0x0d
#define     Get_CEC_Version             0x9f
#define     CEC_Version                       0x9e
#define     Give_Device_Vendor_ID       0x8c
#define     Device_Vendor_ID            0x87
#define     Manu_Status                       0x8e
#define     Feature_Abort                     0x00
#define     Standby                           0x36

/* Declare variables */
volatile uint32_t      Timer0_int_cnt = 0;         // Timer0 interrupt counter

volatile    uint8_t          CEC_State = 0;              // CEC State
volatile    uint8_t          CEC_STB_Ready = 0;          // CEC STart Bit is Ready

volatile    uint8_t          CEC_HEAD = 0;                    // Received Header
volatile    uint8_t          CEC_DATA = 0;                    // Received Data
volatile     int8_t          CEC_EOM_H = -1;             // End of Message for Header
Block
volatile     int8_t          CEC_EOM_D = -1;             // End of Message for Data
Block
volatile     int8_t          CEC_ACK_H = -1;             // ACK for Header     Block
volatile     int8_t          CEC_ACK_D = -1;             // ACK for Data  Block

volatile    uint8_t          Initiator_Device = 0;       // Initiator Device

volatile    uint8_t          Tx_Retry_Cnt = 0;                // Tx Retry Count
volatile    uint8_t          CEC_Head_Dest_flag = 0;     // CEC Header Dest. is correct
volatile    uint8_t          CEC_Broadcast_flag = 0;     // 1=CEC Broadcast Command,
0=None
volatile    uint8_t          Tx_flag = 0;                // 1=Tx, 0=Rx
volatile    uint8_t          Tx_NACK_H = 0;              // Tx receive NACK for Header
Block
volatile    uint8_t          Tx_NACK_D = 0;              // Tx receive NACK for Data
Block
volatile    uint8_t          Rx_CEC_Data_flag = 0;       // 1=received CEC data, 0=None
volatile    uint8_t          Rx_CEC_Data_Len = 0;        // Received data length
volatile    uint8_t          RX_DATA[17] = {0};          // Received Data
```

```
volatile    uint8_t         Key_flag = 0;               // 1=key interrupt flag,
0=None
volatile    uint8_t         TMR1_flag = 0;              // 1=TMR1 ISR, 0=No TMR1_ISR


uint8_t     REPORT_PHYSICAL_ADDRESS[5] = {0x0f,Report_Physical_Address,0x00,0x00,
0x00};
uint8_t     DEVICE_VENDOR_ID[5] = {0x0f,Device_Vendor_ID,0x00,0x00,0x00};
uint8_t     CEC_VERSION[3] = {0x00,CEC_Version,0x04}; // 0x04:1.3a
uint8_t     REPORT_POWER_STATUS[3] = {0x00,Report_Power_Status,0x00};// 0x00:On,
0x01:OFF
uint8_t     STANDBY[2] = {0x00,Standby}; // 0x04: DVD Player/Chromecast, 0x08:SONY PS3



/*---------------------------------------------------------------------------*/
/*  MAIN function                                                            */
/*---------------------------------------------------------------------------*/
int main (void)
{
 uint8_t i;

 /* Init System, IP clock and multi-function I/O */
 SYS_Init();

 /* Init UART0 for printf */
 UART0_Init();

 printf("+-------------------------------------------+\n");
 printf("|              HDMI CEC Sample Code          |\n");
 printf("+-------------------------------------------+\n");
 printf("\n");

 /* Use PB.13 to turn on/off LED */
 GPIO_SetMode(PB, BIT13, GPIO_PMD_OUTPUT);
 PB13 = 1;


 /* Configure PB.10 as Quasi-bidirectional mode and enable interrupt by falling edge
trigger */
      GPIO_SetMode(PB, BIT10, GPIO_PMD_QUASI);
 GPIO_EnableInt(PB, 10, GPIO_INT_FALLING);
      NVIC_EnableIRQ(GPAB_IRQn);


 /* Enable interrupt de-bounce function and select de-bounce sampling cycle time is
1024 clocks of LIRC clock */
```

```
      GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_1024);
 GPIO_ENABLE_DEBOUNCE(PB, BIT10);

 /* Initial PB.11 as Open-drain mode and output high */
 GPIO_SetMode(PB, BIT11, GPIO_PMD_OPEN_DRAIN);
 PB11 = 1;

 /* set PE.5 as Input mode */
 GPIO_SetMode(PE, BIT5, GPIO_PMD_INPUT);
 PE5 = 1;

 /* Initial Timer0 for Delay function */
 Init_Timer0();

    /* Initial Timer1 for Timer Capture function */
 Init_Timer1();

 Tx_flag = 0;
 Key_flag = 0;
 Rx_CEC_Data_flag = 0;

 while(1){
      if(Key_flag == 1){
           GPIO_DisableInt(PB, 10);
           Key_flag = 0;
            STANDBY[0] = Initiator_Device;  // Device Standby, 0x0f: Broadcast
Standby
           Send_CEC_Command(STANDBY, 2);
           GPIO_EnableInt(PB, 10, GPIO_INT_FALLING);
      }else if(Rx_CEC_Data_flag == 1){
           Rx_CEC_Data_flag = 0;

           if(CEC_Head_Dest_flag == 1){
                printf("\nReceived CEC Data: ");
                for(i=0;i<Rx_CEC_Data_Len;i++)
                     printf("%02x ", RX_DATA[i]);
                Decode_CEC_Command();
           }else{
                printf("\nMonitored CEC Data: ");
                for(i=0;i<Rx_CEC_Data_Len;i++)
                     printf("%02x ", RX_DATA[i]);
```

```
            }
        }
    }
}
```

4.2.1.2 System Initialization

The source code is used to enable the external crystal 12 MHz, PLL and peripheral clocks, and determine the clock sources and multi-functions for the related peripherals. The system clock HCLK that comes from PLL output is 48 MHz.

```
void SYS_Init(void)
{
/*---------------------------------------------------------------------------------------------------------*/
/* Init System Clock                                                                                       */
/*---------------------------------------------------------------------------------------------------------*/
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Enable external 12MHz XTAL */
    CLK->PWRCON |= CLK_PWRCON_XTL12M_EN_Msk;

    /* Enable internal 10kHz RC */
    CLK->PWRCON |= CLK_PWRCON_OSC10K_EN_Msk;

    /* PLL_OUT = 48 MHz */
    CLK->PLLCON = 0xCA6E;

    /* Waiting for clock ready */
    CLK_WaitClockReady(CLK_CLKSTATUS_PLL_STB_Msk | CLK_CLKSTATUS_OSC10K_STB_Msk |
CLK_CLKSTATUS_XTL12M_STB_Msk);

    /* Switch HCLK clock source to PLL */
    CLK->CLKSEL0 = CLK_CLKSEL0_HCLK_S_PLL;

    /* Enable IP clock */
    CLK->APBCLK = CLK_APBCLK_UART0_EN_Msk | CLK_APBCLK_TMR0_EN_Msk |
CLK_APBCLK_TMR1_EN_Msk;

    /* IP clock source */
    CLK->CLKSEL1 = CLK_CLKSEL1_UART_S_PLL | CLK_CLKSEL1_TMR0_S_HXT |
CLK_CLKSEL1_TMR1_S_HCLK;
```

```
 /* Update System Core Clock */
 /* User can use SystemCoreClockUpdate() to calculate PllClock, SystemCoreClock and
CycylesPerUs automatically. */
 PllClock        = PLL_CLOCK;            // PLL
 SystemCoreClock = PLL_CLOCK / 1;        // HCLK
 CyclesPerUs     = PLL_CLOCK / 1000000;  // For SYS_SysTickDelay()

/*---------------------------------------------------------------------------------------*/
/* Init I/O Multi-function                                                               */
/*---------------------------------------------------------------------------------------*/
 /* Set PB multi-function pins for UART0 RXD, TXD */
      SYS->GPB_MFP = SYS_GPB_MFP_PB0_UART0_RXD | SYS_GPB_MFP_PB1_UART0_TXD;

 /* Set PE multi-function pins for TM1_EXT on PE.5 */
      SYS->GPE_MFP |= SYS_GPE_MFP_PE5_TM1_EXT;
      SYS->ALT_MFP |= SYS_ALT_MFP_PE5_TM1_EXT;
 SYS->ALT_MFP2 |= SYS_ALT_MFP2_PE5_TM1_EXT;

 /* Lock protected registers */
 SYS_LockReg();
}
```

4.2.1.3 UART0 Initialization

The source code is used to set UART port 0 to show debugging message, the baud rate which is 115200, 8-bit data, none parity and 1 stop bit.

```
void UART0_Init(void)
{
/*---------------------------------------------------------------------------------------*/
/* Init UART                                                                             */
/*---------------------------------------------------------------------------------------*/
       /* Reset IP */
       SYS_ResetModule(UART0_RST);

       /* Configure UART0 and set UART0 Baudrate */
       UART_Open(UART0, 115200);
}
```

4.2.1.4 Timer0 Initialization and Delay Sub-function

The source code is used to set the related control registers of Timer 0. The clock source is external 12 MHz crystal and it operates in periodic mode and generates the interrupt per 100 us. The Timer0_Delay() sub-function is just used to create the regular delay time in this firmware code, which is based on 100 us.

```
//*****************************
// Initial Timer0 interrupt (100us)
//
// Use 12 MHz crystal as Timer0 clock source
// Use mode 1 (periodic mode)
// Timer0 is 24-bit resolution
//*****************************
void Init_Timer0(void)
{
       /* Reset Timer0 counter, prescale and disable Timer0 (CEN = 0) */
       TIMER0->TCSR |= TIMER_TCSR_CRST_Msk;

       /* Select Operation mode */
       TIMER0->TCSR &= ~TIMER_TCSR_MODE_Msk;
       TIMER0->TCSR |= TIMER_PERIODIC_MODE;        // Select periodic mode for
operation mode and enable TDR function

  /* Select Time out period = (Period of timer clock input) * (8-bit Prescale + 1) *
(24-bit TCMP) = 0.0000833ms * 1 * 1200 = 100us */
       TIMER0->TCSR &= ~TIMER_TCSR_PRESCALE_Msk;   // Set Prescale [0~255], must be set
as 0
       TIMER0->TCMPR = 1200;                       // Set TCMPR [0~16777215]

       /* Disable Timer0 */
       TIMER0->TCSR &= ~TIMER_TCSR_CEN_Msk;

       /* Enable Timer0 interrupt and NVIC IRQ */
       TIMER0->TCSR |= TIMER_TCSR_IE_Msk;          // Enable Timer0 interrupt
       NVIC_EnableIRQ(TMR0_IRQn);                  // Enable Timer0 NVIC IRQ
       NVIC_SetPriority(TMR0_IRQn, 0);       // Set Timer0 to 1st interrupt priority

       /* Clear Timer0  interrupt counter */
       Timer0_int_cnt = 0;
}
```

```
//*******************************
// Timer0 Delay (N x 100us)
//
//*******************************
void Timer0_Delay(uint32_t N)
{
     /* Reset Timer0 counter, prescale and disable Timer0 */
      TIMER0->TCSR |= TIMER_TCSR_CRST_Msk;

      /* Reset Timer0 interrupt counter */
      Timer0_int_cnt = 0;

      /* Enable Timer0 */
      TIMER0->TCSR |= TIMER_TCSR_CEN_Msk;

     /* Wait msec */
      while(1){
             if(Timer0_int_cnt >= N)
                   break;
      }

      /* Disable Timer0 */
    TIMER0->TCSR &= ~TIMER_TCSR_CEN_Msk;
}
```

4.2.1.5 Timer1 Initialization

The source code is used to set the related control registers of Timer 1. The clock source is from HCLK, 48 MHz in this example code, and it operates just for the Timer Capture function to measure the duration about CEC bit timing.

```
//*******************************
// Initial Timer1 for CEC Decoder
//
// Use HCLK as Timer1 clock source
// Use Timer Capture Function
// Timer1 is 24-bit resolution
//*******************************
void Init_Timer1(void)
{
```

```
        /* Reset Timer1 counter, prescale and disable Timer1 (CEN = 0) */
        TIMER1->TCSR |= TIMER_TCSR_CRST_Msk;

        /* Select Operation mode */
        TIMER1->TCSR &= ~TIMER_TCSR_MODE_Msk;
        TIMER1->TCSR |= TIMER_ONESHOT_MODE;    // Select one-shot mode for operation
mode and enable TDR function

        /* Select Time out period = (Period of timer clock input) * (8-bit Prescale +
1) * (24-bit TCMP) */
        TIMER1->TCSR &= ~TIMER_TCSR_PRESCALE_Msk;   // Set Prescale [0~255], must be set
as 0
        TIMER1->TCMPR = 0xFFFFFF;

        /* Set TM1_EXT at falling-edge interrupt and to reset counter */
        TIMER1->TEXCON &= ~TIMER_TEXCON_TEX_EDGE_Msk;
        TIMER1->TEXCON |= TIMER_CAPTURE_COUNTER_RESET_MODE;

        /* Enable TM1_EXT debounce function */
        TIMER1->TEXCON |= TIMER_TEXCON_TEXDB_Msk;

        /* Enable TM1_EXT interrupt */
        TIMER1->TEXCON |= TIMER_TEXCON_TEXIEN_Msk;

        /* Enable TM1_EXT pin */
        TIMER1->TEXCON |= TIMER_TEXCON_TEXEN_Msk;

        /* Disable Timer1 */
        TIMER1->TCSR &= ~TIMER_TCSR_CEN_Msk;

    /* Enable Timer1 interrupt and NVIC IRQ */
        TIMER1->TCSR |= TIMER_TCSR_IE_Msk;          // Enable Timer1 interrupt
     NVIC_EnableIRQ(TMR1_IRQn);                     // Enable Timer1 NVIC IRQ
        NVIC_SetPriority(TMR1_IRQn, 3);        // Set Timer1 to lowest interrupt
priority
        /* Reset CEC state */
        CEC_State = 0;
}
```

4.2.2 Interrupt Handler

4.2.2.1 GPIO Port A and B Interrupt Handler

The source code is used to o make sure the interrupt generated by PB.10 and set the key flag (Key_flag).

```
void GPAB_IRQHandler(void)
{
	/* To check if PB.10 interrupt occurred */
	if(GPIO_GET_INT_FLAG(PB, BIT10)){
		Key_flag = 1;
		GPIO_CLR_INT_FLAG(PB, BIT10);
	}else{
		/* Un-expected interrupt. Just clear all PA, PB interrupts */
		PA->ISRC = PA->ISRC;
		PB->ISRC = PB->ISRC;
		printf("\nUn-expected interrupts.");
	}
}
```

4.2.2.2 Timer 0 Interrupt Handler

The source code is used to increase the Timer 0 interrupt count (Timer0_int_cnt) by 1 when each time the Timer 0 generated the interrupt per 100 us.

```
//*******************************
// Timer0 call back function (Timer0 interrupt subroutine)
//
//*******************************
void TMR0_IRQHandler(void)
{
	/* Clear TIMER0 Timeout Interrupt Flag */
	TIMER0->TISR = TIMER_TISR_TIF_Msk;

	/* Timer0 interrupt counter + 1 */
	Timer0_int_cnt = Timer0_int_cnt + 1;
}
```

4.2.2.3 Timer 1 Interrupt Handler

In this Timer 1 interrupt handler, the Timer Capture hardware measures the duration of CEC bit timing and judge the duration is start bit, logic 0 or logic 1. Also, it stores the received bit data from CEC bus for Header Block and Data Block and then responds the ACK bit if needed. The CEC state machine for the receiving and transmitting CEC signal is shown as follows.

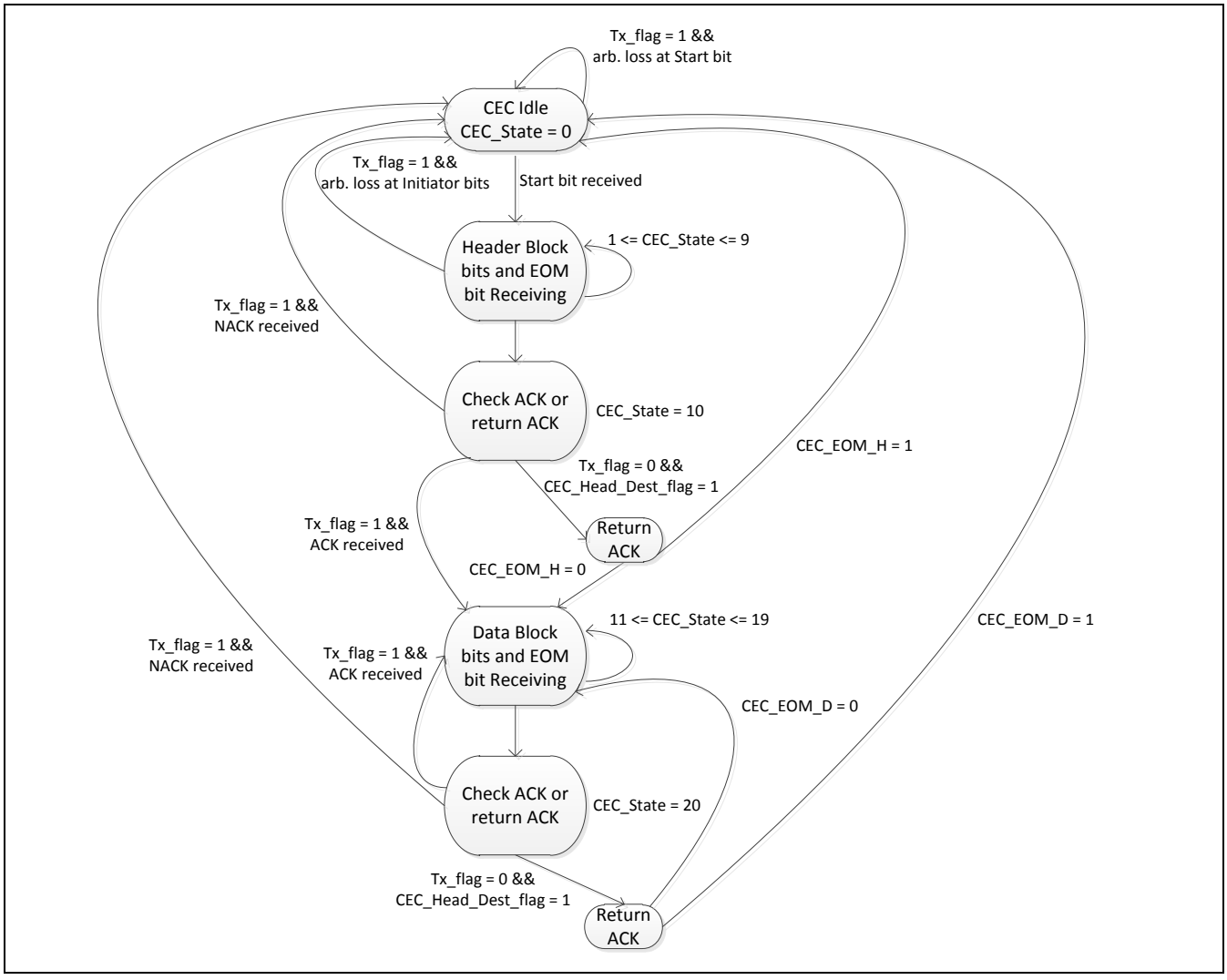

Figure 4-3 CEC State Machine

Figure 4-4 shows the flow chart for Timer 1 capture to receive the CEC signal from the bus.

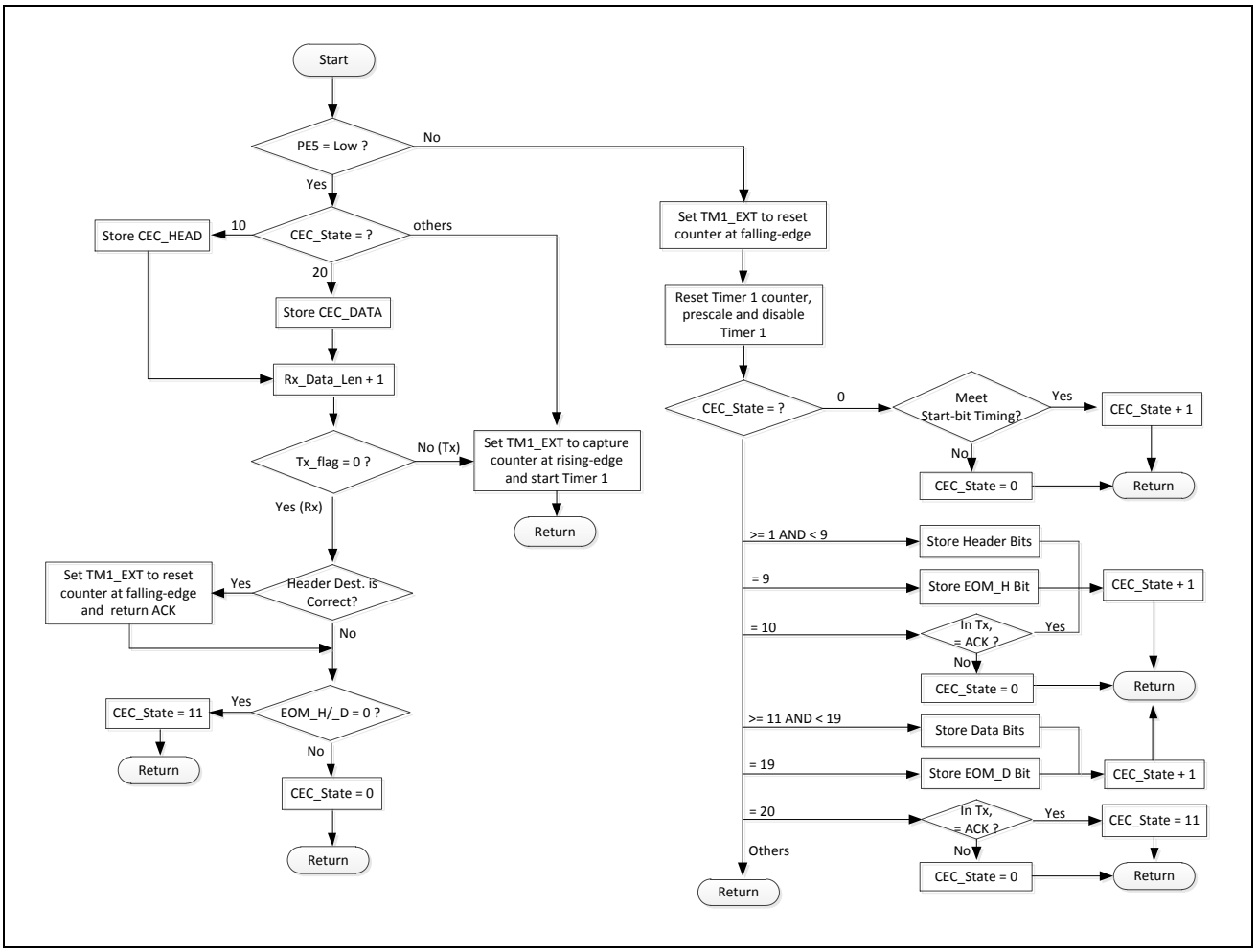

Figure 4-4 Receiving CEC Flow Chart

```
//*******************************
// Timer1 call back function (Timer1 interrupt subroutine)
//
//*******************************
void TMR1_IRQHandler(void)
{
        uint32_t TCAP1_tmp = 0;

        /* Clear TIMER1 Capture Interrupt Flag */
        TIMER1->TEXISR = TIMER_TEXISR_TEXIF_Msk;

        TMR1_flag = 1;
```

```
        // CEC Bus is Low
        if(PE5 == 0){
              // ACK for Header     Block and Data Block
              if(((CEC_State == 10) || (CEC_State == 20)) && (CEC_STB_Ready == 1)){
                    if(CEC_State == 10)
                          RX_DATA[Rx_CEC_Data_Len] = CEC_HEAD;
                    else
                          RX_DATA[Rx_CEC_Data_Len] = CEC_DATA;

                    Rx_CEC_Data_Len++;

                    // Rx
                    if(Tx_flag == 0){
                          if(CEC_Head_Dest_flag == 1){
                                /* Set T1EX at falling-edge interrupt and to reset
counter */
                                TIMER1->TEXCON &= ~TIMER_TEXCON_TEX_EDGE_Msk;
                                TIMER1->TEXCON |= TIMER_CAPTURE_COUNTER_RESET_MODE;

                                /* ACK: PB.11 output low 1.5ms */
                                PB11 = 0;
                                Timer0_Delay(CEC_BIT_0_L);
                                PB11 = 1;
                          }

                          // Next Data
                          if(((CEC_State == 10) && (CEC_EOM_H == 0)) || \
                        ((CEC_State == 20) && (CEC_EOM_D == 0))){
                                CEC_State = 11;
                          // The End
                          }else{
                                PB11 = 1;
                                CEC_State = 0;
                                CEC_STB_Ready = 0;
                                Rx_CEC_Data_flag = 1;
                          }
                    }
                    // Tx
                    else{
                          /* Set T1EX at rising-edge interrupt and to capture counter */
```

```
                    TIMER1->TEXCON |= TIMER_CAPTURE_RISING_EDGE;
                    TIMER1->TEXCON &= ~TIMER_TEXCON_RSTCAPSEL_Msk;

                    /* Enable Timer1 */
                    TIMER1->TCSR |= TIMER_TCSR_CEN_Msk;
               }
          }
          // The Other non-ACK Fields
          else{
               /* Set T1EX at rising-edge interrupt and to capture counter */
               TIMER1->TEXCON |= TIMER_CAPTURE_RISING_EDGE;
               TIMER1->TEXCON &= ~TIMER_TEXCON_RSTCAPSEL_Msk;

               /* Enable Timer1 */
               TIMER1->TCSR |= TIMER_TCSR_CEN_Msk;
          }
     }
     // CEC Bus is High
     else{
          TCAP1_tmp = TIMER1->TCAP;
          /* Set T1EX at falling-edge interrupt and to reset counter */
          TIMER1->TEXCON &= ~TIMER_TEXCON_TEX_EDGE_Msk;
          TIMER1->TEXCON |= TIMER_TEXCON_RSTCAPSEL_Msk;

          /* Disable Timer1 (CEN = 0) */
          TIMER1->TCSR &= ~TIMER_TCSR_CEN_Msk;

          // Start Bit
          if(CEC_State == 0){
               CEC_State++;

               if((TCAP1_tmp >= STB_MIN) && (TCAP1_tmp <= STB_MAX)){  // Start bit
                    CEC_STB_Ready = 1;    // Set STart Bit Ready
                    CEC_HEAD = 0;
                    CEC_EOM_H = -1;
                    CEC_ACK_H = -1;
                    CEC_EOM_D = -1;
                    CEC_ACK_D = -1;
                    CEC_Head_Dest_flag = 0;
                    Rx_CEC_Data_flag = 0;
                    Rx_CEC_Data_Len = 0;
```

```
                }else{
                        CEC_State = 0;
                        CEC_STB_Ready = 0;
                }
        }
        // Header Block
        else if((CEC_State >= 1 && CEC_State < 9) && (CEC_STB_Ready == 1)){
                CEC_State++;

                CEC_HEAD = CEC_HEAD << 1;
                if((TCAP1_tmp >= BIT_1_MIN) && (TCAP1_tmp <= BIT_1_MAX))     // 1
                        CEC_HEAD |= 0x01;
                else if((TCAP1_tmp >= BIT_0_MIN) && (TCAP1_tmp <= BIT_0_MAX))      //
0
                        CEC_HEAD &= 0xFE;
                else{
                        CEC_State = 0;
                        CEC_STB_Ready = 0;
                }
        }
        // End of Message for Header Block
        else if((CEC_State == 9) && (CEC_STB_Ready == 1)){
                CEC_State++;

                // Rx
                if(Tx_flag == 0){
                        /* Check the Destination whether is belong to this device? */
                        if((CEC_HEAD & 0x0F) == CEC_HEAD_DEST)
                                CEC_Head_Dest_flag = 1;
                }

                if((TCAP1_tmp >= BIT_1_MIN) && (TCAP1_tmp <= BIT_1_MAX))     // 1
                        CEC_EOM_H = 1;
                else if((TCAP1_tmp >= BIT_0_MIN) && (TCAP1_tmp <= BIT_0_MAX)) // 0
                        CEC_EOM_H = 0;
                else{
                        CEC_State = 0;
                        CEC_STB_Ready = 0;
                }
        }
        // ACK for Header Block
```

```
            else if((CEC_State == 10) && (CEC_STB_Ready == 1)){
                  // Tx
                  if(Tx_flag == 1){
                        if((TCAP1_tmp >= BIT_1_MIN) && (TCAP1_tmp <= BIT_1_MAX))
// 1                                    CEC_ACK_H = 1;
                        else if((TCAP1_tmp >= BIT_0_MIN) && (TCAP1_tmp <= BIT_0_MAX))
   // 0                                 CEC_ACK_H = 0;

                        /* Receive Not ACK for Header Block */
                        if(CEC_ACK_H != 0){
                              PB11 = 1;
                              CEC_State = 0;
                              CEC_STB_Ready = 0;
                              Tx_NACK_H = 1;
                        }else
                              CEC_State = 11;
                  }
            }
            // Data Blcok
            else if((CEC_State >= 11 && CEC_State < 19) && (CEC_STB_Ready == 1)){
                  CEC_State++;

                  CEC_DATA = CEC_DATA << 1;
                  if((TCAP1_tmp >= BIT_1_MIN) && (TCAP1_tmp <= BIT_1_MAX))     // 1
                        CEC_DATA |= 0x01;
                  else if((TCAP1_tmp >= BIT_0_MIN) && (TCAP1_tmp <= BIT_0_MAX)) // 0
                        CEC_DATA &= 0xFE;
                  else{
                        CEC_State = 0;
                        CEC_STB_Ready = 0;
                  }
            }
            // End of Message for Data Block
            else if((CEC_State == 19) && (CEC_STB_Ready == 1)){
                  CEC_State++;

                  if((TCAP1_tmp >= BIT_1_MIN) && (TCAP1_tmp <= BIT_1_MAX))     // 1
                        CEC_EOM_D = 1;
                  else if((TCAP1_tmp >= BIT_0_MIN) && (TCAP1_tmp <= BIT_0_MAX)) // 0
                        CEC_EOM_D = 0;
                  else{
```

```
                        CEC_State = 0;
                        CEC_STB_Ready = 0;
                    }
                }
                // ACK for Data Block
                else if((CEC_State == 20) && (CEC_STB_Ready == 1)){
                    // Tx
                    if(Tx_flag == 1){
                        if((TCAP1_tmp >= BIT_1_MIN) && (TCAP1_tmp <= BIT_1_MAX))
   // 1
                            CEC_ACK_D = 1;
                        else if((TCAP1_tmp >= BIT_0_MIN) && (TCAP1_tmp <= BIT_0_MAX))
// 0
                            CEC_ACK_D = 0;

                        /* Receive Not ACK for Data Block */
                        if(CEC_ACK_D != 0){
                            PB11 = 1;
                            CEC_State = 0;
                            CEC_STB_Ready = 0;
                            Tx_NACK_D = 1;
                        }else
                            CEC_State = 11;
                    }
                }
        }
}
```

4.2.3 CEC Sub-functions

This paragraph describes three CEC related sub-functions.

4.2.3.1 Send CEC Command

This sub-function tries to send the CEC command to the CEC bus. If it occurs transmitting fail due the arbitration loss or the other reasons, it will retry to send out the command again till this command be sent out successfully or the retrying count reaches to the maximum retried limitation.

```
//*******************************
// Send CEC Command
//
//
//*******************************
void Send_CEC_Command(uint8_t data[], uint8_t byte_len)
{
	uint8_t	i;

	for(Tx_Retry_Cnt = 0; Tx_Retry_Cnt < CEC_MAX_RETRY; Tx_Retry_Cnt++){
		if(SendCEC(data, byte_len) == 0){
			printf("\nTransmitted CEC Data: ");
			for(i=0;i<byte_len;i++)
				printf("%02x ", data[i]);
			break;
		}else{
			printf("\nRe-transmit!!! <%d>", Tx_Retry_Cnt);
		}
	}
}
```

4.2.3.2 Send CEC Format

This sub-function checks the CEC bus status firstly and waiting for the bus is free, then it generates the start bit, data bit of Header Block and Data Block, EOM bit and ACK bit that follow the timing definition in HDMI-CEC specification. The flow chart of transmitting the CEC signal to the bus is shown in Figure 4-5.

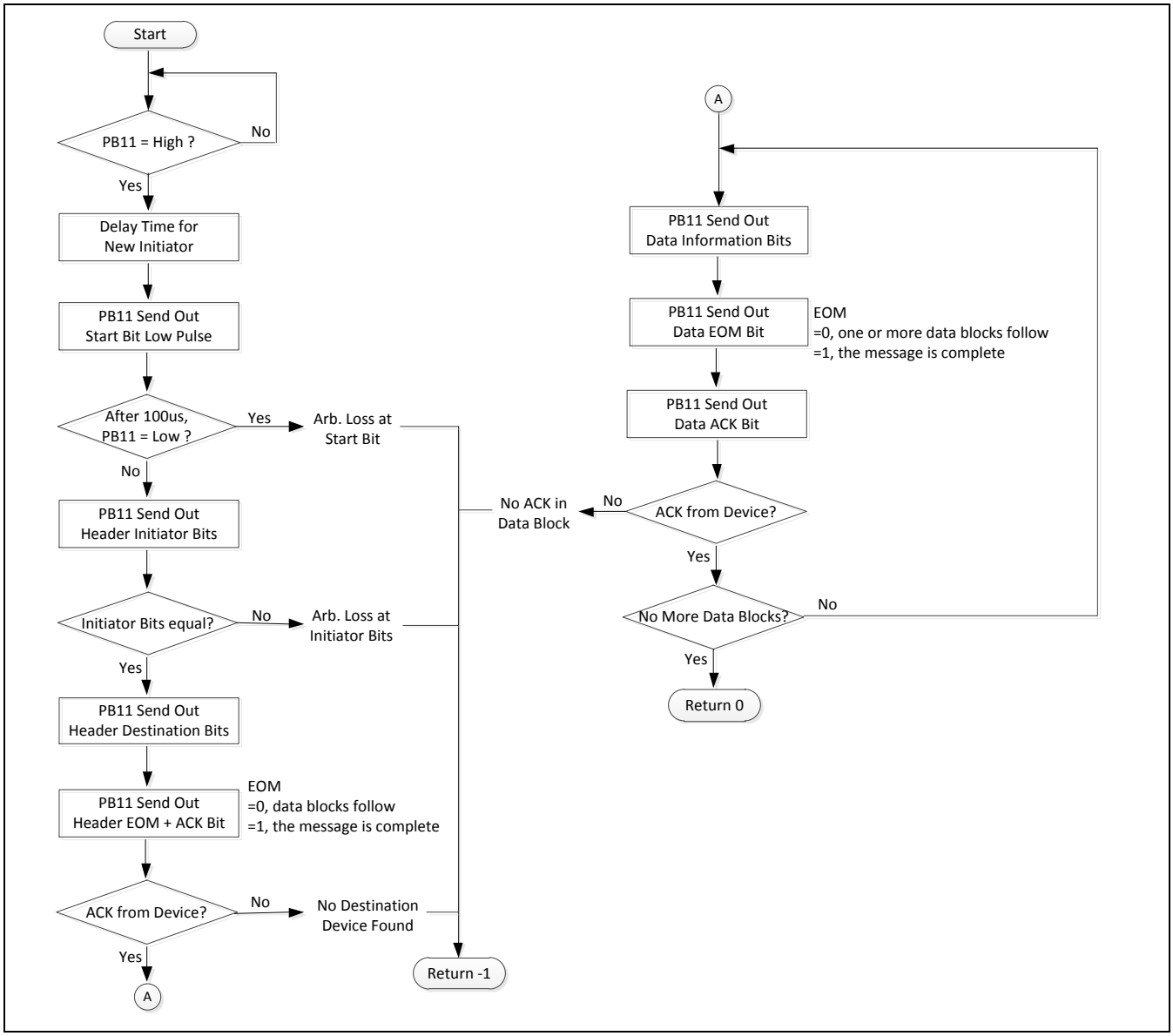

Figure 4-5 Transmitting CEC Flow Chart

```
//*******************************
// Send CEC Format
//
// PortB.11
//*******************************
```

```
int8_t SendCEC(uint8_t data[], uint8_t byte_len)
{
	uint8_t	nbyte;
	int8_t	nbit;

	Tx_flag = 1;		// 1=Transmit
	Tx_NACK_H = 0;	// clear Tx_NACK_H
	Tx_NACK_D = 0;	// clear Tx_NACK_D

	CEC_Broadcast_flag = ((data[0] & 0x0f) == 0x0f);

	/* Waitting for PB.11 (CEC BUS) to release High */
	while(PB11 == 0);

	/* Check CEC Bus = High that over free time >= 5 * 2.4ms */
	TMR1_flag = 0;
	// Delay free time >= 5 * 2.4ms
	Timer0_Delay(CEC_FREE_TIME_NI);
	if(TMR1_flag != 0){
		printf("\nDetect CEC Bus = Low!!!");
		// Turn back as a follower
		Tx_flag = 0;
		// Delay free time >= 3 * 2.4ms
		Timer0_Delay(CEC_FREE_TIME_RS);
		return -1;
	}

  /* Send out Start Bit */
	PB11 = 0;
	Timer0_Delay(CEC_START_BIT_L);
	PB11 = 1;
	if(PB11 == 0){
		printf("\nArbitration Loss at Start Bit!!!");
		// Turn back as a follower
		Tx_flag = 0;
		// Delay free time >= 3 * 2.4ms
		Timer0_Delay(CEC_FREE_TIME_RS);
		return -1;
	}
	Timer0_Delay(CEC_START_BIT_H);
```

```
        /* Send out Header Code and Data Code */
        for(nbyte=0; nbyte < byte_len; nbyte++){
              /* Send out Bits of Header Code and Data Code */
              for(nbit=7; nbit >= 0; nbit--){
                    if(data[nbyte] & (1 << nbit)){        // MSB first
                    PB11 = 0;
                          Timer0_Delay(CEC_BIT_1_L);
                          PB11 = 1;
                          Timer0_Delay(CEC_BIT_1_H);
              }else{
                          PB11 = 0;
                          Timer0_Delay(CEC_BIT_0_L);
                          PB11 = 1;
                          Timer0_Delay(CEC_BIT_0_H);
                    }
                    if((nbyte == 0) && (nbit >= 4)){
                          if(((data[0] >> nbit) & 0x01) != (CEC_HEAD & 0x01) ){
                                printf("\nArbitration Loss at Initiator Bits!!!");
                                // Turn back as a follower
                                Tx_flag = 0;
                                // Delay free time >= 3 * 2.4ms
                                Timer0_Delay(CEC_FREE_TIME_RS);
                                return -1;
                          }
                    }
              }

              /* Send out EOM Bit */
              if(nbyte == byte_len - 1){
                    PB11 = 0;
                    Timer0_Delay(CEC_BIT_1_L);
                    PB11 = 1;
                    Timer0_Delay(CEC_BIT_1_H);
              }else{
                    PB11 = 0;
                    Timer0_Delay(CEC_BIT_0_L);
                    PB11 = 1;
                    Timer0_Delay(CEC_BIT_0_H);
              }

              /* Send out ACK Bit = 1 */
```

```
            PB11 = 0;
            Timer0_Delay(CEC_BIT_1_L);
            PB11 = 1;
            Timer0_Delay(CEC_BIT_1_H);

            /* Check ACK for Header Block */
            if(nbyte == 0 && Tx_NACK_H == 1){
                  if(CEC_Broadcast_flag == 0){
                        printf("\nNo Found Destination Device!!!");
                        // Turn back as a follower
                        Tx_flag = 0;
                        // Delay free time >= 3 * 2.4ms
                        Timer0_Delay(CEC_FREE_TIME_RS);
                        return -1;
                  }
            }else if(nbyte != 0 && Tx_NACK_D == 1){
                  if(CEC_Broadcast_flag == 0){
                        printf("\nNo ACK for Data Block!!!");
                        // Turn back as a follower
                        Tx_flag = 0;
                        // Delay free time >= 3 * 2.4ms
                        Timer0_Delay(CEC_FREE_TIME_RS);
                        return -1;
                  }
            }
      }

      // Turn back as a follower
      Tx_flag = 0;

      // Delay free time >= 7 * 2.4ms for next time to transmit
      Timer0_Delay(CEC_FREE_TIME_PI);

      return 0;
}
```

4.2.3.3 Decode CEC Command

This sub-function decoded the received CEC command from the other device.

```
//*******************************
// Decode CEC Command
//
//
//*******************************
void Decode_CEC_Command(void)
{
       Initiator_Device = (RX_DATA[0] & 0xf0) >> 4;

       switch(RX_DATA[1]){
            case Give_Physical_Address:
                 Send_CEC_Command(REPORT_PHYSICAL_ADDRESS, 5);
                 break;
            case Give_Device_Vendor_ID:
                 Send_CEC_Command(DEVICE_VENDOR_ID, 5);
                 break;
            case Give_Device_Power_Status:
                 REPORT_POWER_STATUS[0] = Initiator_Device;
                 Send_CEC_Command(REPORT_POWER_STATUS, 3);
                 break;
            case Get_CEC_Version:
                 CEC_VERSION[0] = Initiator_Device;
                 Send_CEC_Command(CEC_VERSION, 3);
                 break;
            case Image_View_On:
                 PB13 = 0;  // Turn on LED (as TV/Projector on)
                 break;
         case Text_View_On:
                 PB13 = 0;  // Turn on LED (as TV/Projector on)
                 break;
            case Inactive_Source:
                 PB13 = 1;  // Turn off LED (as TV/Projector off)
                 break;
            default:
                 break;
       }
 }
```

4.2.4 Test Results

Finally, three CEC-supporting devices, SONY PS3, Philips DVD player and Google Chromecast connected with the M051 or M0518 tiny board are used to verify the CEC functions. The results are shown as follows.

4.2.4.1 Connection with SONY PS3

In Figure 4-6, the first line shows “Monitored CEC Data: 88”, which means that the MCU device monitored the CEC messages from CEC bus. The first and only one byte “88” is a header block, and the initiator address, the higher nibble 0x8, indicates that this message is transmitted out by SONY PS3 device, but the destination address, the lower nibble 0x8, does not belong to this device that MCU emulated as a TV, the logical address is 0x0. This is a <Polling> message due to only one byte without the other opcode or parameters. The related CEC message description are described in section 4.2.4.4.

The 7th line shows “Received CEC Data: 80 0d”, which means that the MCU device received the CEC messages from CEC bus. The first byte “80” is a header block, and the destination address, the lower nibble 0x0, belongs to this MCU device. Thus, the MCU receives these messages and responds the relative messages to CEC bus or does relative operations. The second byte “0d” is an opcode of <Text View On> message to turn on the TV or projector (as emulated LED on the tiny board) if it is off by default.

The 12th line shows “Transmitted CEC Data: 08 36”, it means this MCU transmitted the CEC messages to CEC bus. The first byte “08” is a header block, and the initiator address, the higher nibble 0x8, indicates that this message is transmitted out from MCU device, and this destination address, the lower nibble 0x8, is for SONY PS3 device. The second byte “36” is an opcode of <Standby> message for the SONY PS3 to enter the standby state.

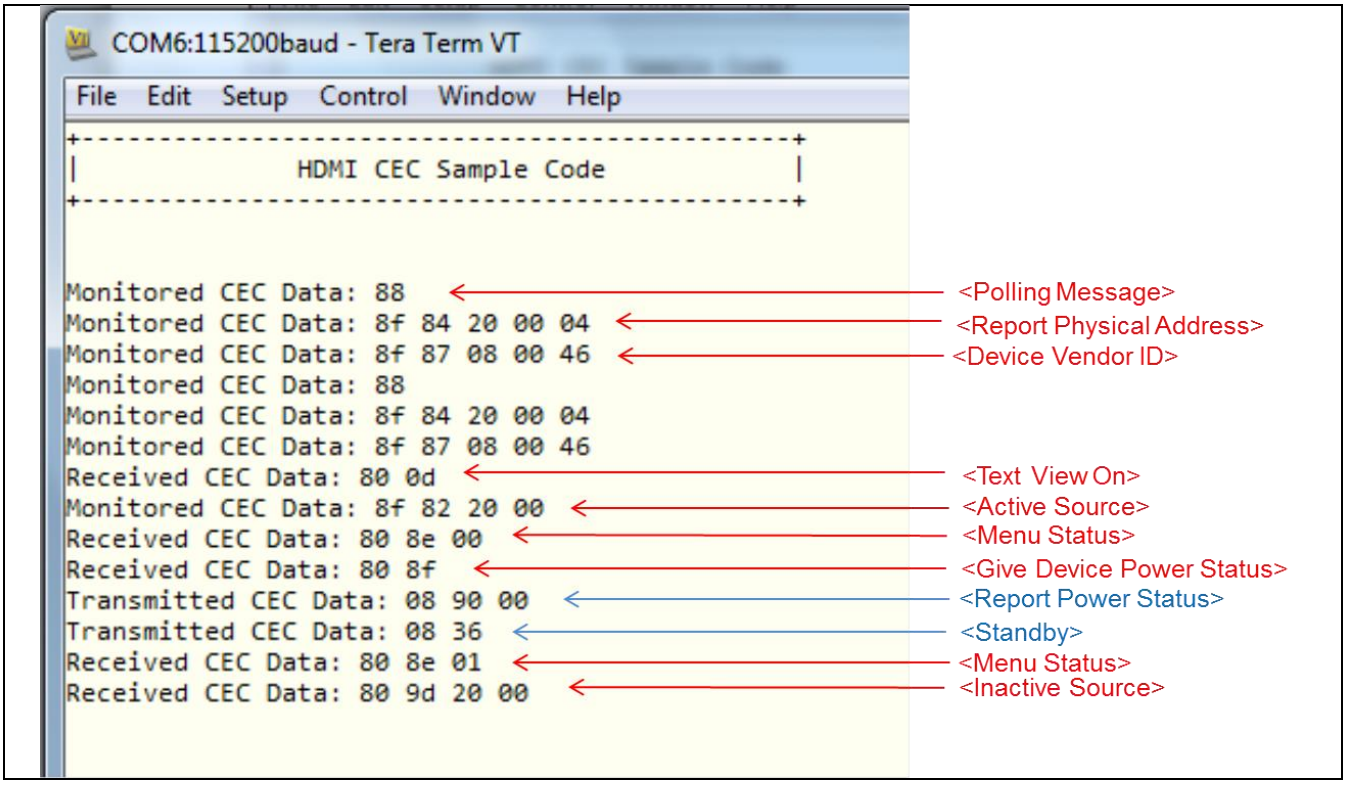

Figure 4-6 Test Result with SONY PS3

4.2.4.2 Connection with Philips DVD Player

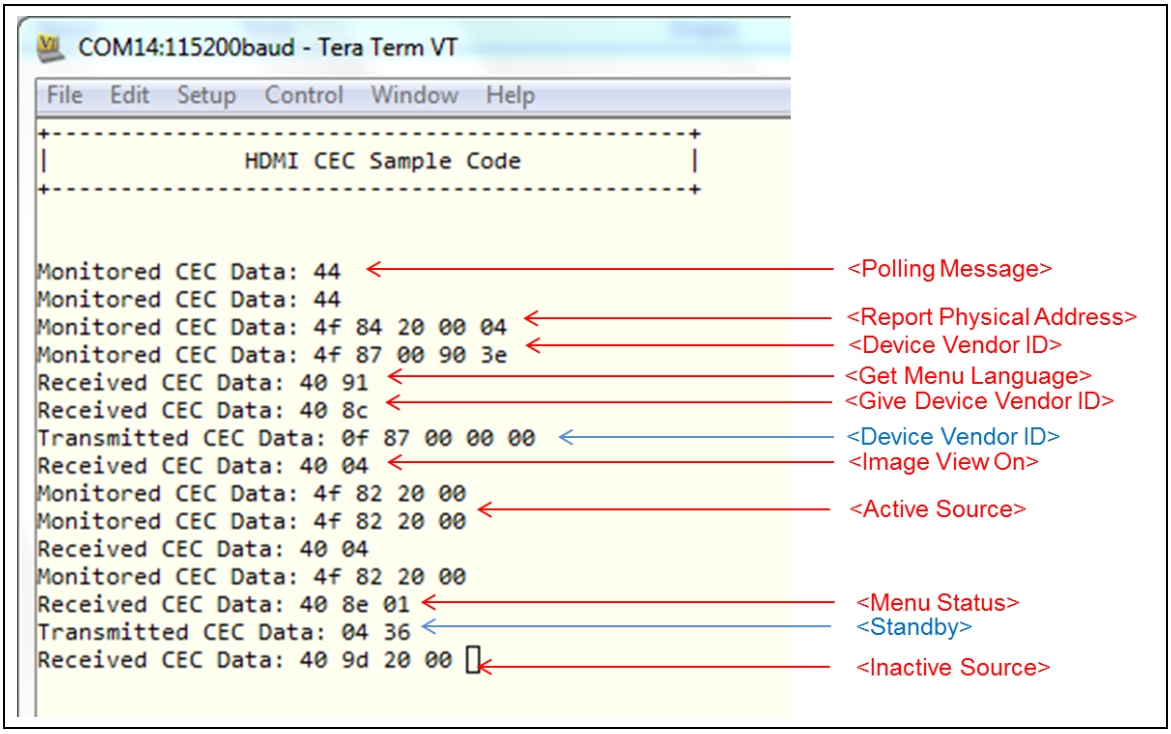

Figure 4-7 Test Result with Philips DVD Player

4.2.4.3 Connection with Google Chromecast

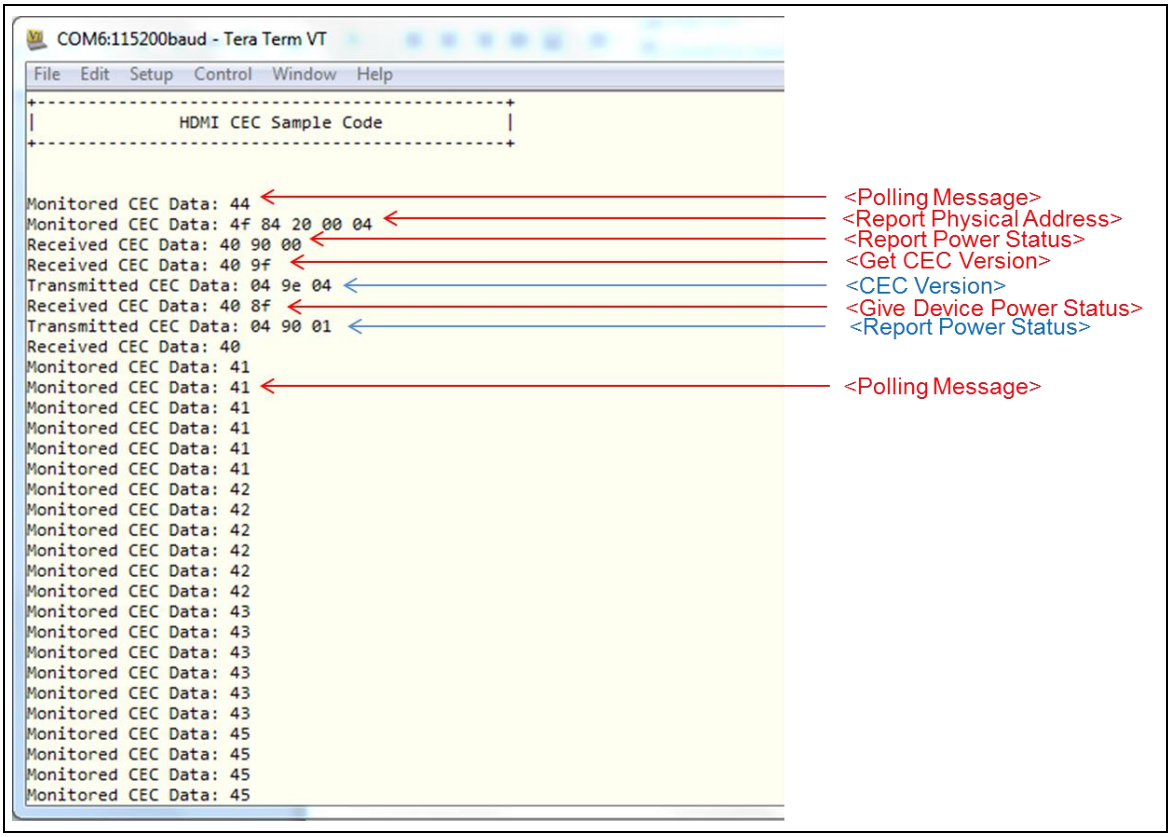

Figure 4-8 Test Result with Google Chromecast

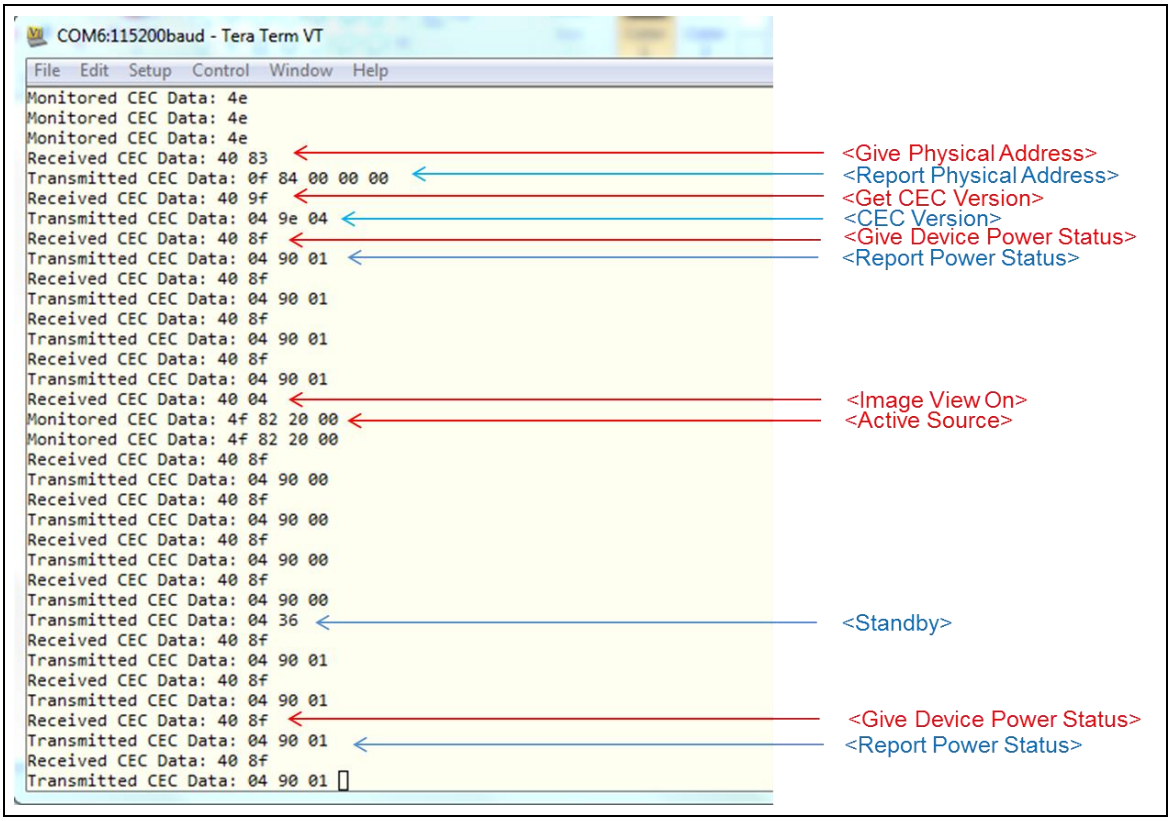

Figure 4-9 Test Result with Google Chromecast (Continued)

4.2.4.4 CEC Message Description

Table 4-2 shows the related description about these messages at the test results above. For more opcodes, parameters and description on these or the other messages that user can refer to the HDMI-CEC specification in detail.

Opcode	Value	Description	Parameters	Response
<Polling Message>	-	Used by any device for device discovery – similar to ping in other protocols.	None	Shall set a low level ACK.
<Give Physical Address>	0x83	A request to a device to return its physical address.	None	<Report Physical Address>
<Report Physical Address>	0x84	Used to inform all other devices of the mapping between physical and logical address of the initiator.	[Physical Address] [Device Type]	
<Give Device Vendor ID>	0x8C	Requests the Vendor ID from a device.	None	<Device Vendor ID>
<Device Vendor ID>	0x87	Reports the vendor ID of this device.	[Vendor ID]	Any other interested device may store the vendor ID of the device.
<Active Source>	0x82	Used by a new source to indicate that it has started to transmit a stream OR used in response to a <Request Active Source>	[Physical Address]	A current active source should take appropriate action. TV should switch to the appropriate input. Any CEC switches between source and root shall switch to the appropriate input and come out of standby if necessary.
<Inactive Source>	0x9D	Used by the currently active source to inform the TV that it has no video to be presented to the user, or is going into standby as the result of a local user command on the device.	[Physical Address]	The TV may display its own internal tuner and shall send an <Active Source> with the address of the TV; or The TV may send <Set Stream Path> to another device for display.
<Image View On>	0x04	Sent by a source device to the TV whenever it enters the active state (alternatively it may send <Text View	None	Turn on (if not on). If in 'Text Display' state then the TV enters 'Image Display' state. Note: Should not change TV

		On>).		menu or PIP status.
<Text View On>	0x0D	As <Image View On>, but should also remove any text, menus and PIP windows from the TV's display.	None	As <Image View On>, but should remove PIPs and menus from the screen. The TV enters 'Image Display' state regardless of its previous state.
<Standby>	0x36	Switches one or all devices into standby mode. Can be used as a broadcast message or be addressed to a specific device.	None	Switch the device into standby. Ignore the message if already in standby.
<CEC Version>	0x9E	<Get CEC Version>	[CEC Version]	
<Get CEC Version>	0x9F	Used by a device to enquire which version of CEC the target supports	None	The source responds with a <CEC Version> message indicating the CEC version
<Menu Status>	0x8E	Used to indicate to the TV that the device is showing/has removed a menu and requests the remote control keys to be passed though.	[Menu State]	If Menu State indicates activated, TV enters 'Device Menu Active' state and forwards those Remote control commands to the initiator. If deactivated, TV enters 'Device Menu Inactive' state and stops forwarding remote control commands.
<Get Menu Language>	0x91	Sent by a device capable of character generation (for OSD and Menus) to a TV in order to discover the currently selected Menu language. Also used by a TV during installation to discover the currently set menu language of other devices.	None	
<Give Device Power Status>	0x8F	Used to determine the current power status of a target device	None	<Report Power Status>
<Report Power Status>	0x90	Used to inform are questing device of the current power status	[Power Status]	

<Feature Abort>	0x00	Used as a response to indicate that the device does not support the requested message type, or that it cannot execute it at the present time.	[Feature Opcode] [Abort Reason]	Assume that request is not supported or has not been actioned.
<Abort> Message	0xFF	This message is reserved for testing purposes.	None	

Table 4-2 CEC Messages

Revision History

Date	Revision	Description
2015.11.16	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.