Linux Device Driver Kit User's Manual

v2.1

Background

eSrijan Innovations Private Limited specializes in creation of innovative Automation Products & Solutions. eSrijan innovations are a combination of one or more of various Engineering streams, &/or Medical Systems, with applied Science & Technology. eSrijan Products are primarily targetted for the Indian majority. And, eSrijan Solutions are customized OEM products for specific customers.

eSrijan Innovations, an Indian Institute of Science Alumni venture, was incorporated in February 2010. It is supported by a strong group of mentors including entrepreneurs and field experts. It is powered by a team, with average industry experience of more than 25 years.

In product space, eSrijan is creating energy & education related products. In the solution space, it has its say in both rural and urban automation, ranging from daily chores to automotive. eSrijan is driving its solution development from its head-quarters in Bangalore, India.

Contact Details

Email: products@eSrijan.com
Website: http://lddk.eSrijan.com/

Errata & Correspondence: For any errata, correction, feedback or any other correspondence regarding this document, feel free to contact us at the above mentioned contact details.

Revision History

Revision	Published Date	Description
1.0	13th March 2011	First version of the Linux Device Driver Kit's User Manual as per fw v1.0
1.1	5th April 2011	Updated as per fw v1.1. Control endpoint request CUSTOM_RQ_GET_MEM_SIZE added. USB driver samples renamed with name prefix 'uctl' changed to 'ddk'. Reference to serial_comm (minicom like application) added. LDDK block driver sample information added.
1.2	13th June 2011	Mentioned that 8-LED Array would work only with a direct serial port connection. Updated the various sample driver and application file names, as per the enhancements in the LDD templates.
1.3	1st August 2011	Updated as per fw v1.2. Default state of PGM LED has been changed to “on”, indicating a healthy firmware.
1.4	22nd May 2013	Corrected the website link. Added information on the Linux Device Driver Kit's USB vendor id & device id. Also added the information about the request type of the control requests.
2.0	29th May 2013	Updated as per DDK v2.1 release. Maps to fw v2.0.
2.1	21st June 2013	Indented features of the LDDK v2.1. Updated the missed out information from the previous version: switching on the LDDK using the power switch, jumper pair configuration required for USB2Serial transfer. Corrected the MEM_EP_IN number in chapter 5. Updated chapter 5 and added chapter 6 as per feature additions (flash & I/O access) in fw v2.1.

Table of Contents

Chapter 1: Introduction

The Linux Device Driver Kit or LDDK (in short) is the first board of its kind based on popular Atmel's microcontroller ATMega16. It is a USB Device clocked at 16 Mhz with the free vendor ID 0x16C0, reserved by Objective Development (http://www.obdev.at), for shared usage for open hardwares. The product ID of LDDK has been chosen as 0x05DC, the one specified for vendor class devices.

The LDDK is based on the design of DDK v2.1 and has been designed for both non-experienced users who make their first steps in the world of microcontrollers and professional programmers who search for a universal platform to write a Linux Device Driver.

Features:

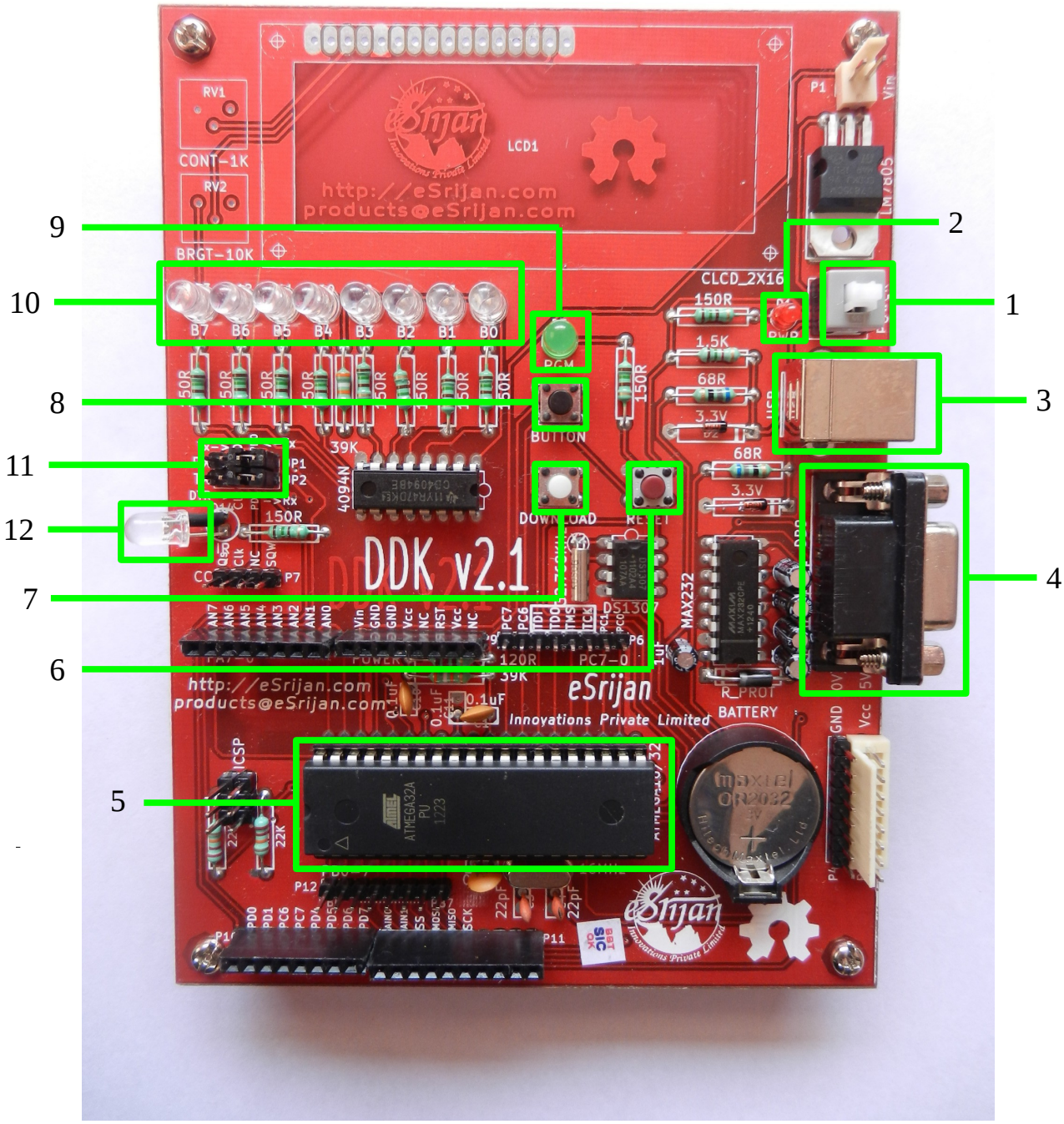
1) AVR uC based USB Device (clocked @ 16MHz)
2) With 32KiB Flash and 1024B EEPROM
3) USB Powered - No additional power adaptor required
4) 8-LED Array for Character Device Driver programming through DB9 i/f
5) Memory (EEPROM) accessibility as Character Device
6) Contains USB2Serial circuitry for USB Device Driver programming
7) Enabled with Block Device Driver programming over Memory
   - ❍ EEPROM
   - ❍ Flash (fw ver >= v2.1)
8) Various on-board I/O accesses through control endpoint zero (fw ver >= 2.1)
9) 2 line x 16 character driver controllable display interface
10) Real Time Clock accessibility provision
11) Self-upgradable for future new functionalities
12) And for an Electronics Hobbyist, it can be additionally used for:

- Minimum of 16 independent Digital I/O Controls
- Upto 8 independent ADC Inputs
- Upto 4 independent PWM/Timer Outputs
- I^2C programming of Real Time Clock
- Menu controls using two Push Buttons
- Infra-red transmission using the on-board IR LED
- Arduino shields using the on-board extension headers
- Various automation using Sensors

By custom programming as a uC-based device through standard HID i/f, Or

By direct programming using the ICSP programming interface.

Chapter 2: Sections of LDDK board



1. Power Switch – “Press” to power on LDDK from USB
2. Red Power LED – Glows to show the LDDK is powered up
3. USB Cable connector – USB Device Port and Power Supply input
4. Female Serial (DB9) connector – For LED-Array & RS232 communication
5. Atmega32 Micro-controller – Brain of the LDDK
6. Red Reset Switch – “Press” to reset the LDDK
7. White Download Switch – Used to download new code or as a menu button
8. Black Button Switch – Used to test LDDK or as a menu button
9. Green Program LED – Toggle LED (controllable over USB). Continuously blinking indicates Bootloader mode. Continuously on indicates USB Device mode (i.e. running the LDDK firmware)
10. Blue B7:B0 LEDs – 8-LED Array indicating bits B7 to B0 (driven either by DB9 connector or through the Micro-controller)
11. RX & TX Jumpers – Jumper pair to select the various modes of operation between Serial (RS232), Blue LEDs, Micro-controller. Jumper pair position indications are as follows:
    a) Left: LED Array control through RS232 communication
    b) Centre: LED Array control by the Micro-controller (programmable)
    c) Right: RS232 communication with the Micro-controller
12. IR LED – Infra-red Transmission LED

Chapter 3: LED-Array Control through DB9 connector

LED-Array is a sequence of 8 LEDs, which are driven by serial data input from B7 to B0, one bit per clock. So, 8 -ve edge (high → low) clock transitions are needed to drive the complete 8-bit data onto the 8 LEDs.

Configuration: RX & TX jumper pair needs to be on the left side.

Connection: Connect the female connector of the serial cable to your PC's serial port and male connector of the serial cable to the LDDK's DB9 connector. Please note that the PC's serial port should be a direct one, not an indirect one like USB2SERIAL or so. Power on the LDDK.

Relevant DB9 Pin Connections:

Male DB9 Pin No.	USART Pin Meaning	LED-Array Significance
3	Tx (Transmitter)	Data (0: LED On; 1: LED Off)
4	DTR (Data Terminal Ready)	Clock (High to Low: Shifts Data)

Examples:

1) To glow B0 – Set USART “Tx” line low, and then send a “High” → “Low” signal over USART “DTR” line. Note that pulling “Tx” line low is not same as sending “0” on it, rather it setting the break.
2) To switch off B0 – Set USART “Tx” line high, and then send a “High” → “Low” signal over USART “DTR” line. Note that setting “Tx” line high is not same as sending “1” on it, rather it releasing the break (the default state of “Tx” line)
3) To glow B1 & switch off B0 – Set USART “Tx” line low, and then send a “High” → “Low” signal over USART “DTR” line. Then, set USART “Tx” line high, and then send a “High” → “Low” signal over USART “DTR” line.

Driver Code Sample: led_board.c

Explanation: This sample interfaces the LDDK as a character device interface

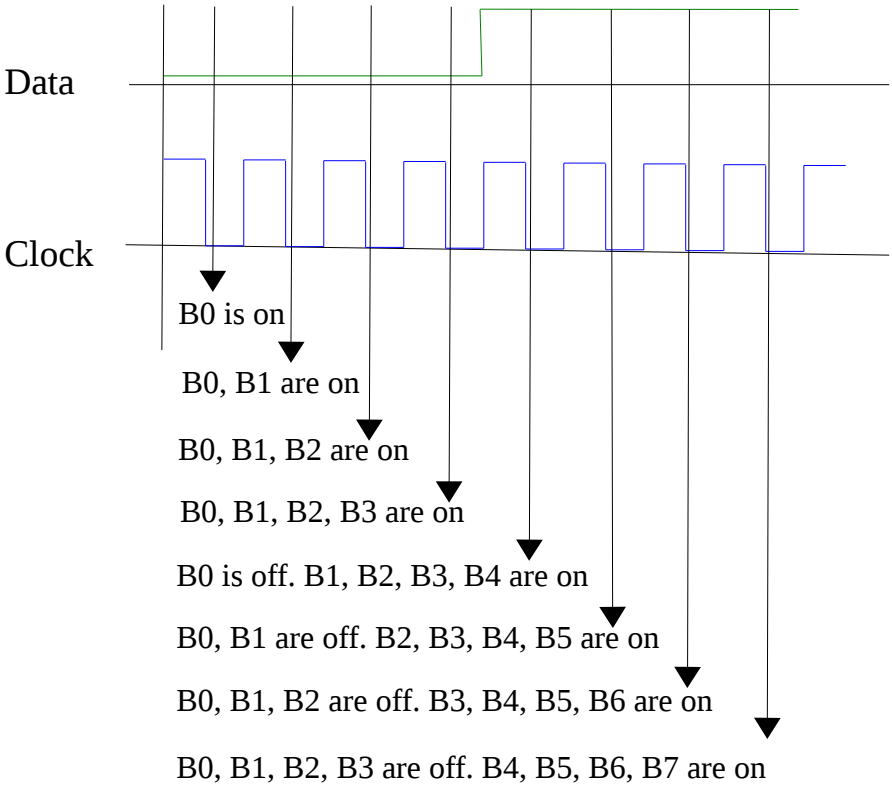
/dev/led0 to display the data sent as the ASCII value on the LED-Array.

Application Code Sample: led_ops.c

Other possible Applications: Pattern Display, Blinking Lights, Buzzer Display for 8 Teams.

Waveforms: (Data & Clock with the LED bit states on every clock)

For switching on B7, B6, B5, B4, and switching off B3, B2, B1, B0



Chapter 4: LDD Kit as a USB device

LDDK's firmware provides majorly 3 USB device functionalities. The most basic form of USB communication is through something called an *endpoint*. A USB endpoint can carry data in only one direction, either from the host computer to the device (called an OUT endpoint) or from the device to the host computer (called an IN endpoint). Endpoints can be thought of as unidirectional pipes. LDDK's firmware provides 5 such endpoints for the various functionalities. The universal control endpoint 0,which is both IN & OUT. And 2 interrupt IN and 2 interrupt OUT endpoints.

Configuration: RX & TX jumper pair needs to be on the right side. (Required only for the USB to Serial communication.)

Connection: Connect the type A connector of the USB cable to your PC's USB connector and the type B connector of the USB cable to the LDDK's USB Device Port. Additionally for USB to Serial communication, connect the female connector of the serial cable to your PC's serial port and male connector of the serial cable to the LDDK's DB9 connector. Power on the LDDK.

Endpoint Details:

1. **Control endpoint 0** – Commonly used for configuring the device, retrieving information about the device, sending commands to the device, or retrieving status reports about the device. Usually used for small control transactions. Every valid USB device has to have this control endpoint 0 that is used by the USB host controller to understand the device. For the LDDK, it is additionally used to:
   a) On/off the PGM LED (CUSTOM_RQ_SET_LED_STATUS: 1)
   b) Know the current status PGM LED (CUSTOM_RQ_GET_LED_STATUS: 2)
   c) Set the memory read offset (CUSTOM_RQ_SET_MEM_RD_OFFSET: 3)
   d) Get the memory read offset (CUSTOM_RQ_GET_MEM_RD_OFFSET: 4)
   e) Set the memory write offset (CUSTOM_RQ_SET_MEM_WR_OFFSET: 5)
   f) Get the memory write offset (CUSTOM_RQ_GET_MEM_WR_OFFSET: 6)
   g) Get the memory size (CUSTOM_RQ_GET_MEM_SIZE: 7)
   The defines & numbers in () are the corresponding request numbers for the

vendor specific request type for the recipient device, typically denoted by the flags USB_TYPE_VENDOR | USB_RECIP_DEVICE. Additionally, the request numbers 1, 3, 5 are for OUT (USB_DIR_OUT), and 2, 4, 6 are for IN direction (USB_DIR_IN), respectively. For the OUT requests, the parameter is passed through the value field of the control request. For the IN requests, the data is received through the data field of the control request. The index field of control request is ignored for all the above requests.

For request CUSTOM_RQ_SET_LED_STATUS, a value of 0 switches off the PGM LED and a value of 1 switches it on.
For request CUSTOM_RQ_GET_LED_STATUS, the current PGM LED status is available in the bit 0 of byte 0 of the data received – 0 for off and 1 for on.
For requests CUSTOM_RQ_SET_MEM_RD_OFFSET and CUSTOM_RQ_SET_MEM_WR_OFFSET, value is the 16-bit memory offset to be set.
For requests CUSTOM_RQ_GET_MEM_RD_OFFSET and CUSTOM_RQ_GET_MEM_WR_OFFSET, the 16-bit memory offset to be obtained through the byte 0 & byte 1 of the data received – byte 0 being the LSB and byte 1 the MSB.

Driver Code Samples:

- For LED specific control commands: ddk_led.c, ddk_led.h, ddk.h
- For memory specific control commands: ddk_mem.c, ddk_mem.h, ddk.h

Explanation: These samples provide the LDDK as auto-detected character device interfaces /dev/usb/ddk_led0 and /dev/usb/ddk_mem0, respectively. These interfaces provide various ioctl system calls to provide the above control endpoint request functionalities to the user space.

Application Code Samples:

- For LED specific control commands: usb_ops.c, ddk_led.h
- For memory specific control commands: usb_mem.c, ddk_mem.h

Addendum: A single collated example driver made of ddk_usb.c, ddk_usb.h, ddk.h is also available, providing character device interface /dev/usb/ddk0. The corresponding application for this is usb_test.c.

Interrupt IN endpoints transfer small amounts of data at a fixed rate every time the USB host asks the device for data. **Interrupt OUT endpoints** transfer small amounts of data at a fixed rate every time the USB host sends the data to the device. Both these transfers are guaranteed by the USB protocol to always have enough reserved bandwidth to make it through.

2. I**nterrupt IN endpoint (MEM_EP_IN: 0x81):** This is the endpoint to receive/read data from the memory (EEPROM) from the current memory read offset, which is incremented automatically by how many bytes have been read. A single read request has to be exactly of 8 bytes. The accessible memory size is 1024 bytes (0x400).

   Driver Code Sample: ddk_mem.c, ddk_mem.h, ddk.h

   Explanation: This sample provides the LDDK as a auto-detected character device interface /dev/usb/ddk_mem0. This interface provides the read system call to provide the above interrupt endpoint request functionality to the user space.

   Application Code Sample: usb_mem.c, ddk_mem.h

3. I**nterrupt OUT endpoint (MEM_EP_OUT: 0x01):** This is the endpoint to send/write data to the memory (EEPROM) starting at the current memory write offset, which is incremented automatically by how many bytes have been written. A single write request can be maximum of 8 bytes. The accessible memory size is 1024 bytes (0x400).

   Driver Code Sample: ddk_mem.c, ddk_mem.h, ddk.h

   Explanation: This sample provides the LDDK as a auto-detected character device interface /dev/usb/ddk_mem0. This interface provides the write system call to provide the above interrupt endpoint request functionality to the user space.

   Application Code Sample: usb_mem.c, ddk_mem.h

4. **Interrupt IN endpoint (SER_EP_IN: 0x82):** This is the endpoint to receive/read data from the serial port (DB9 connector of LDDK) through USB. A single read request has to be exactly of 8 bytes, though less than 8 bytes may be returned, which need to appropriately handled by the driver developer.

   Driver Code Sample: ddk_u2s.c, ddk_u2s.h, ddk.h

   Explanation: This sample provides the LDDK as a auto-detected character device interface /dev/usb/ddk_u2s0. This interface provides the read system call to provide the above interrupt endpoint request functionality to the user space.

   Application Code Sample: usb_ops.c, serial_comm.c

5. **Interrupt OUT endpoint (SER_EP_OUT: 0x02):** This is the endpoint to send/write data to the serial port (DB9 connector of LDDK) through USB. A single write request can be maximum of 8 bytes.

   Driver Code Sample: ddk_u2s.c, ddk_u2s.h, ddk.h

   Explanation: This sample provides the LDDK as a auto-detected character device interface /dev/usb/ddk_u2s0. This interface provides the write system call to provide the above interrupt endpoint request functionality to the user space.

   Application Code Sample: usb_ops.c, serial_comm.c

   Help: For this functionality to work properly, the RX & TX jumper pair should be on the right side. In case, if it still gives problems, try resetting the micro-controller using the red “reset” switch. Make sure that all the serial communication applications like serial_comm, minicom, gtkterm, hyper-terminal, ... are quit before trying this.

NB All these USB related code samples must be tried mutually exclusively – meaning unload all others to try the next one. (Important !!!)

Chapter 5: LDD Kit as a Block device

LDDK's firmware provides 1 USB device functionality for supporting the Block device interface over the LDDK's memory, through the endpoints MEM_EP_IN (0x81) and MEM_EP_OUT (0x01). These endpoints have been already explained in the previous chapter in sections 2 & 3. The same endpoints may be used to access the LDDK as a block device.

Connection: Connect the type A connector of the USB cable to your PC's USB connector and the type B connector of the USB cable to the LDDK's USB Device Port. Power on the LDDK.

Multiple Memory Access (fw ver >= 2.1): Till LDDK firmware version 2.0, the only accessible memory, through the MEM_EP_IN and MEM_EP_OUT endpoints was the EEPROM, as explained in the previous chapter. However, since LDDK firmware version 2.1, the following requests on the control endpoint 0 has been added to enable selection of the memory type to be accessed through the MEM_EP_IN and MEM_EP_OUT endpoints:

a) Set the current memory type (CUSTOM_RQ_SET_MEM_TYPE: 8)
b) Get the current memory type (CUSTOM_RQ_GET_MEM_TYPE: 9)

The defines & numbers in () are the corresponding request numbers for the vendor specific request type for the recipient device, typically denoted by the flags USB_TYPE_VENDOR | USB_RECIP_DEVICE. Additionally, the request number 8 is for OUT (USB_DIR_OUT), and 9 is for IN direction (USB_DIR_IN), respectively. For the OUT request, the parameter is passed through the value field of the control request. For the IN request, the data is received through the data field of the control request. The index field of control request is ignored for both of the above requests. The supported parameters for the above requests are: 0 for eeprom and 1 for flash.

On every power up / reset of LDDK, the memory type selection defaults to EEPROM. So, for it to be flash, an explicit request needs to be made, on every power on of LDDK.

In case the selected memory type is flash (for LDDK firmware version >= 2.1), the following changes apply:

1. **Memory Size:** The accessible flash memory size is typically 24KiB (0x6000), unless the LDDK firmware is being compiled with CLCD support, in which case the size would be 20KiB (0x5000). So, to obtain the correct size use the CUSTOM_RQ_GET_MEM_SIZE request on the control endpoint 0. Also, all the four requests on control endpoint 0 related to setting and getting the read and write offsets of memory, work for the flash memory.

2. **Interrupt IN endpoint (MEM_EP_IN: 0x81):** This is the endpoint to receive/read data from the flash memory from the current memory read offset, which is incremented automatically by how many bytes have been read. A single read request has to be exactly of 8 bytes.

3. **Interrupt OUT endpoint (MEM_EP_OUT: 0x01):** This is the endpoint to send/write data to the flash memory starting at the current memory write offset, which is incremented automatically by how many bytes have been written. A single write request can be maximum of 8 bytes. However, the actual write happens only when such a write completes the 128-byte page boundary. So, for a proper write behaviour, a write shall always be made in a multiple of 16 continuous 8-byte requests, starting from an address aligned to a 128-byte page boundary.

Driver Code Sample: ddk_storage.c, ddk_storage.h, ddk_block.c, ddk_block.h

Explanation: This sample provides the LDDK as a auto-detected block device interface /dev/ddk. This interface provides the block reads & writes to provide the above interrupt endpoints' requests functionality to the user space.

Driver Code Reference: ram_block.c, ram_device.c, ram_device.h, partition.c, partition.h

Explanation: This reference code interfaces the RAM as block device interface /dev/rb*, with partitioning. It may be used as a reference to provide the LDDK as a similar block device interface with partitioning.

Recommended Applications: dd, cat, echo, od -t x1

Chapter 6: LDD Kit I/O Access

With LDDK firmware version >= 2.1, the various I/Os on the LDDK are available through requests on the control endpoint 0.

Connection: Connect the type A connector of the USB cable to your PC's USB connector and the type B connector of the USB cable to the LDDK's USB Device Port. Power on the LDDK.

Access Requests: The following requests on the control endpoint 0 has been added to enable the access of various I/Os on the LDDK:

a) Write into a register on the LDDK (CUSTOM_RQ_SET_REGISTER: 10)
b) Read a register from the LDDK (CUSTOM_RQ_GET_REGISTER: 11)

The defines & numbers in () are the corresponding request numbers for the vendor specific request type for the recipient device, typically denoted by the flags USB_TYPE_VENDOR | USB_RECIP_DEVICE. Additionally, the request number 10 is for OUT (USB_DIR_OUT), and 11 is for IN direction (USB_DIR_IN), respectively. For the OUT request, the parameters are passed through the value and index fields of the control request. For the IN request, the parameter is passed through the index field and the data is received through the data field of the control request. The value field is ignored for the IN control request. In both the requests, the index indicates the LDDK register to be operated on. For the OUT request, value field contains the value to be written into that register. And, for the IN request data field receives the value read from the register. The supported index values for the above requests are:

a) 0 – Reserved. Writes ignored. Reads 0x00.
b) 1 – DIRA for read/write of the direction of Port A. (Dir: 0 – R; 1– W)
c) 2 – DIRB for read/write of the direction of Port B. (Dir: 0 – R; 1– W)
d) 3 – DIRC for read/write of the direction of Port C. (Dir: 0 – R; 1– W)
e) 4 – DIRD for read/write of the direction of Port D. (Dir: 0 – R; 1– W)
f) 5 – PORTA for read/write to Port A.
g) 6 – PORTB for read/write to Port B.
h) 7 – PORTC for read/write to Port C.
i) 8 – PORTD for read/write to Port D.

NB On every power up / reset of LDDK, the direction of all accessible the port bits

default to 0 (Read). So, for them to be writable, the corresponding DIRx register bits need to be set to 1, before the writes.

Apart from the normal I/O operations of the four ports A, B, C, D of the LDDK, the following port bits can be used for the following special purposes:

i. PB2 – Input of the pulled-up BUTTON switch. Direction fixed to input.
ii. PB3 – Output to the IR LED.
iii. PB4 – Input of the pulled-up DOWNLOAD switch. Direction fixed to input.
iv. PD0 – Output DATA for 8-LED Array, with RX jumper on the centre pins. Data is active low, i.e. inverted (0 for on, 1 for off).
v. PD1 – Output CLK for 8-LED Array, with TX jumper on the centre pins. Clock is +ve edge, i.e. rising (low to high).

NB PD0 and PD1 are shared with the serial communication, as well. So, setting them as output for 8-LED Array control disables the serial communication. And setting both of them as input enables the serial communication, for which then the RX-TX jumper pair need to be placed in the appropriate position.

Driver Code Sample: ddk_io.c, ddk_io.h

Explanation: This sample provides the LDDK as auto-detected character device interfaces /dev/usb/ddk_io0. This interface provides various ioctl system calls to provide the above control endpoint request functionalities to the user space.

Application Code Sample: usb_io.c, ddk_io.h