SHAKTI

Introduction to SHAKTI SDK

SHAKTI Group | CSE Dept | PS-CDISHA - RISE Lab | IIT Madras

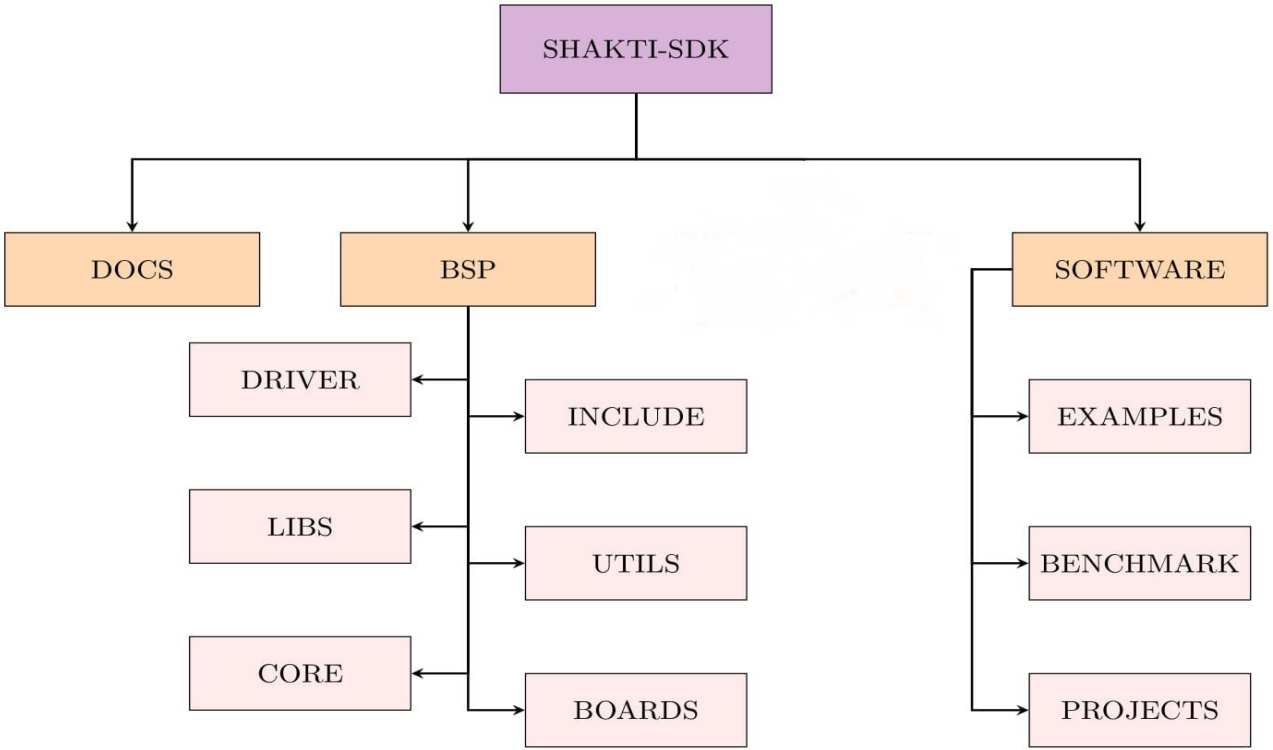
Shakti SDK Features

- Open-source software development platform for SHAKTI.
- Clean separation between boot, drivers, core and application layers.
- Driver support for PWM, QSPI, SPI, PLIC, CLINT, UART, I2C, GPIO, RTC, Watchdog, GPTimer and XADC.
- Drivers written for multiple sensors and tested.
- Standalone and Debug mode supported.

Shakti SDK Features

- Multilevel logging & Flash programming supported.
- Single place for bare-metal application development, projects and benchmarks.
- SDK extendable for any FPGA board.
- Visual Studio Code IDE
  - Using Platform IO Extension
- Arduino IDE support.
- ESP8266 & ESP 32, FTDI, External Flashes and many sensors integrated.

Shakti SDK Architecture



https://gitlab.com/shaktiproject/software/shakti-sdk

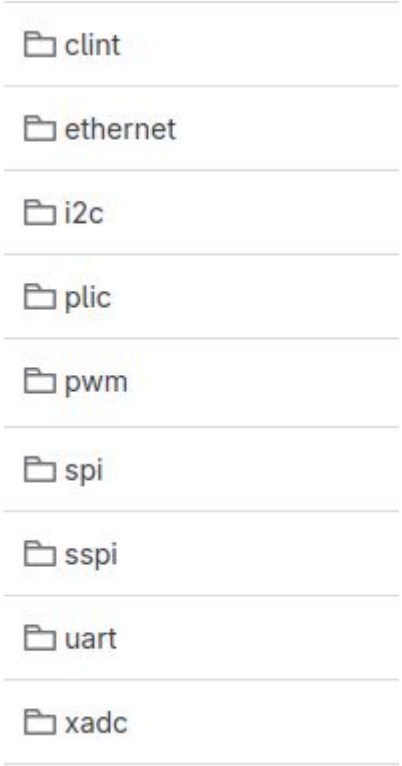
Understanding structure of SHAKTI SDK

SHAKTI

Main files to know (Core)

init.c	Trap Handling initialisations & Peri. Initialisation & calling the main function.
start.S	Register initialisation, Section initialisation like (stack, text, global, data, etc.)
trap.S	Pushing the register values into Stack, Call ISR, Pop the register values.
traps.c	Find the cause for the trap and jump to the Handler

Main files to know (drivers)



- Drivers for all the peripherals.
- Generic But Board specific.

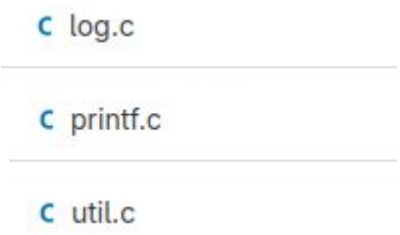
Main files to know (include)

h clint_driver.h	h pinmux.h
h defines.h	h plic_driver.h
h eth_driver.h	h pwm_driver.h
h gpio.h	h qspi.h
h gpio_i2c.h	h spi.h
h gpio_spi.h	h traps.h
h i2c.h	h uart.h
h log.h	h utils.h
h memory.h	h xadc_driver.h

- Header files for all the peripheral drivers.
- Generic But Board specific.

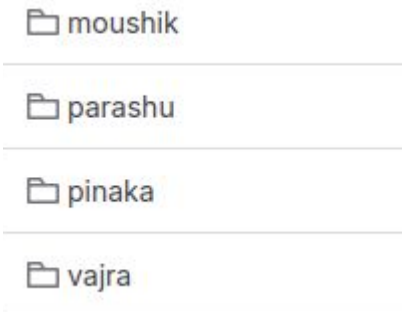
SHAKTI

Main files to know (lib)



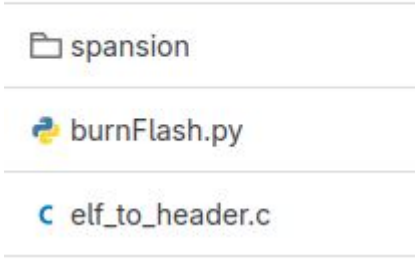
- Print library
- Multilevel log control
- Few more function implementations for delay & string operations.

Main files to know (third party)



- Board specific address mapping (platform.h).
- Configuration files to be used for debugging (xxxx.cfg).
- Linker for various sections of the code.

Main files to know (utils/uploader)

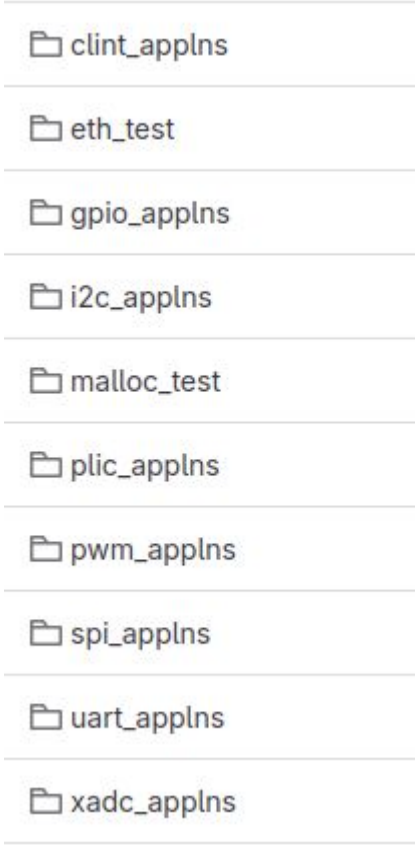


- Board SPI or External SPI can be used for booting of the application.
- Has the required python scripts required to flash the application code.
- Elf is converted to hex file.
- Hex file is added in the uploader c coder.
- The uploader code reads the header file and writes into flash memory.

Main files to know (software)

examples	Example driver codes for peripherals.
projects	Application Project with many drivers.

Main files to know (examples)



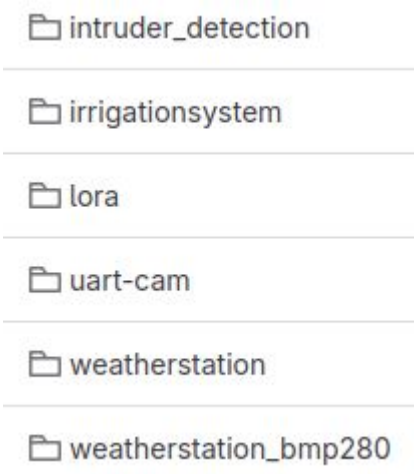
- Peripheral specific example codes.
- Around 45 - 50 example codes.
- Beginning point for all program developers.

Main files to know (examples/i2c)

- I2C specific example codes.
- Different sensors being written helps easy understanding of the driver codes and new sensor additions.

SHAKTI

Main files to know (projects)



- Integrated application codes with more than one peripheral.

Add new peripheral to Shakti SDK

Adding a new peripheral Details (platform.h)

```
/*!Pulse Width Modulation Start Offsets */
#define PWM_BASE_ADDRESS 0x00030000 /*PWM Base address*/
#define PWM_MODULE_OFFSET 0x00000100 /*Offset value to be incremen

/*!Serial Peripheral Interface Offsets */
#define SPI0_START 0x00020000 /* Serial Peripheral Interface 0 */
#define SPI1_START 0x00020100 /* Serial Peripheral Interface 1 */

/*!Universal Synchronous Receiver Transmitter Interface Offsets */
#define UART0_START 0x00011300 /*! UART 0 */
#define UART_OFFSET 0x100
#define MAX_UART_COUNT 3

/*! Inter Integrated Circuit (I2C) Interface */
#define I2C0_BASE 0x00040000 /*! I2C Start Address */
#define I2C_OFFSET 0x1400
#define MAX_I2C_COUNT 2
```

- Add new peripherals base address.
- Peripheral offset.
- Peripheral count.

Adding a new peripheral Register details (drivers/xxxx.h)

```
/* Struct to access UART registers as 32 bit registers */
typedef struct
{
        unsigned short baud;     /*! Baud rate configuration Re
        unsigned short reserv0;  /*! reserved */
        unsigned int  tx_reg;    /*! Transmit register -- the v
        unsigned int  rcv_reg;   /*! Receive register -- the va
        unsigned char  status;   /*! Status register -- Reads v
        unsigned char  reserv1;  /*! reserved */
        unsigned short  reserv2; /*! reserved */
        unsigned short delay;    /*! Delays the transmit with s
        unsigned short reserv3;  /*! reserved */
        unsigned short control;   /*! Control Register -- Confi
        unsigned short reserv5;  /*! reserved */
        unsigned char ien;           /*! Enables the required i
        unsigned char reserv6;   /*! reserved */
        unsigned short reserv7;  /*! reserved */
        unsigned char  iqcycles; /*! 8-bit register that indica
        unsigned char reserv8;   /*! reserved */
        unsigned short reserv9;  /*! reserved */
#ifdef USE_RX_THRESHOLD /*! This is to be used only when suppor
        unsigned char rx_threshold;     /*! RX FIFO size config
        unsigned char reserv10;    /*! reserved */
        unsigned short reserv11;    /*! reserved */
#endif
} uart_struct;
```

- Add new peripherals registers as structures.
- Be careful in adding variable length registers.
- All registers are uniformly placed with 32 bit offset.

SHAKTI

Adding a peripheral address init(drivers/xxxx/xxxx.c)

```
/**
 * @fn void uart_init()
 * @brief Initialise UART Array
 * @details Initialises given number of UART Arrays which has
 *          complete set of UART registers.
 */
void uart_init()
{
        for(int i=0; i< MAX_UART_COUNT; i++)
        {
                uart_instance[i] = (uart_struct*) (UART0_START+i*UART_OFFSET);
        }
}
```

- Initialises the base address for a particular set of peripheral.
- Peripheral instance with array index can be used to point to the registers of the peripheral.

Adding a peripheral functions (drivers/xxxx/xxxx.c)

```
uint32_t write_uart_character(uart_struct * instance, uint8_t prn_character)
{
        while(instance->status & STS_TX_FULL);

        instance->tx_reg = prn_character;

        return 0;
}
```

- Takes the peripheral instance as argument.
- Function is peripheral register access specific.

Include header files (examples/xxxx_appIns/xxxx.c)

```c
#include <string.h>
#include "uart.h"
#include "pinmux.h"
#include "i2c.h"
#include "log.h"

#define LORA_UART uart_instance[1]
#define BAUDRATE 9600
#define LENGTH 100
```

- Include the required header files and macros.

Call the function in example application code

```
void write_to_lora(char *data)
{
        while (*data != '\0')
        {
                write_uart_character(LORA_UART, *data);
                data++;
        }
        write_enter_to_lora();
}
```

- Pass the required details and call the function.

Adding new appli. code (software/examples/Makefile)

```
ifeq ($(PROGRAM),poll_eg)
filepath := eth_test/poll_eg
else
ifeq ($(PROGRAM),ping_req)
filepath := eth_test/ping_req
else
ifeq ($(PROGRAM),ping_res)
filepath := eth_test/ping_res
else
ifeq ($(PROGRAM),lora_receive)
filepath := uart_applns/lora_receive
else
ifeq ($(PROGRAM),lora_transmit)
filepath := uart_applns/lora_transmit
```

- Folder name and application file name should be same.
- Mention the file path for the code to be compiled.

Compile the code

```
make software PROGRAM=hello TARGET=vajra
```

- Make sure the terminal is configured with the required RISC-V tool chain path.
- Else add the path to the environment using export command or alias if exist in .bashrc.
- Run the make command with required target core & program name.

SHAKTI

Compile the code

```
alias rv64imafdc='export PATH=$PATH:~/softwares/build__rv64imafdc/bin:/home/kottee/softwares/build__rv64imafdc
alias rv32imac='export PATH=$PATH:~/softwares/build__rv32imac/bin:/home/kottee/softwares/build__rv32imac'
alias rv32imafc='export PATH=$PATH:~/softwares/build__rv32imafc/bin:/home/kottee/softwares/build__rv32imafc'
```

Successful compilation

```
make[2]: Leaving directory '/home/shakti/Documents/shakti-sdk/software/
examples'
cd uart_applns/hello && make hello.riscv TARGET=vajra DEBUG=
make[2]: Entering directory '/home/shakti/Documents/shakti-sdk/software/
examples/uart_applns/hello'
/toolchain/riscv/bin/../lib/gcc/riscv64-unknown-elf/12.2.0/../../../../
riscv64-unknown-elf/bin/ld: warning: ./output/hello.shakti has a LOAD
segment with RWX permissions
make[2]: Leaving directory '/home/shakti/Documents/shakti-sdk/software/
examples/uart_applns/hello'
All done !
make[1]: Leaving directory '/home/shakti/Documents/shakti-sdk/software/
examples'
```

SHAKTI

Run the code

Follow the steps given in SHAKTI SDK user manual.

- Run **Openocd (sudo openocd -f xxx.cfg)** in first terminal.
- Run **miniterm (pyserial-miniterm /dev/ttyUSBx 19200)** second terminal.
- Run **risc-v gdb** window **(riscv64-unknown-elf-gdb)** in third terminal.
- Run the following in the gdb window.
  - Connect gdb with openocd ($ source gdb.script).
  - Select the file ($ <file path>)
  - Load the file ($ load)
  - Run the file ($ c)
- See the output prints in miniterm window.

References

Shakti Documentation: https://shakti.org.in/documentation.html

Shakti Blogs: https://blogshakti.org.in/

Shakti FPGA files: https://gitlab.com/shaktiproject/sp2020

Shakti SDK: https://gitlab.com/shaktiproject/software/shakti-sdk

Shakti on Arduino: https://blogshakti.org.in/how-to-print-hello-world-on-shakti-using-arduino-ide/

Linux on Shakti: https://gitlab.com/shaktiproject/software/linux-on-shakti

Platform IO: https://registry.platformio.org/platforms/platformio/shakti

Shakti cores: https://gitlab.com/shaktiproject/cores

Shakti peripherals: https://gitlab.com/shaktiproject/uncore/devices

Thank you