SWADESHI MICROPROCESSOR CHALLENGE-2020

Shakti User Manual v1.3

developed by: SHAKTI Development Team @ iitm

shakti.org.in

contact @ shakti [dot] iitm [@] gmail [dot] com

0.1 Proprietary Notice

`The project was funded by Ministry of Electronics and Information Technology (MeITY), Government of India`

0.2 Release Information

Version	Date	Changes
0.1	February 27, 2020	Initial Release
0.2	June 22, 2020	Updated Appendix A
0.3	July 21, 2020	Updated Sections 1.1, 3.3
1.0	August 22, 2020	Updated Sections 4.1, Appendix
1.1	November 24, 2020	Review comments & GPIO changes
1.2	January 24, 2021	Correct 7.1, 7.2 pin mapping
1.3	March 04, 2021	Add Ethernet memory map address

Table of Contents

1

SECTION

Brief Introduction to SHAKTI

SHAKTI is an open-source initiative by the Reconfigurable Intelligent Systems Engineering (RISE) group at IIT-Madras [1]. The aim of the SHAKTI initiative includes building open source production grade processors, complete System on Chips (SoCs), development boards and SHAKTI-based software platform. The SHAKTI project is building a family of 6 processors, based on the RISC-V ISA [2]. There is a road-map to develop reference System on Chips (SoC) for each class of processors, which will serve as an exemplar for that family [3]. The team has channelized years of research on processor architecture to build these SoCs which has competitive commercial offerings in the market with respect to occupied power, area and performance. The current SoC (as of December 2019) developments are for the Controller (C- Class) [5] and Embedded (E- Class) classes [6].

1.1 Processors

SHAKTI is a RISC-V [2] based processor developed at RISE lab, IIT Madras [1, 7]. SHAKTI has envisioned a family of processors as part of its road-map, catering to different segments of the market. They have been broadly categorized into "Base Processors", "Multi-Core Processors" and "Experimental Processors" [3]. The E and C-classes are the first set of indigenous processors aimed at Internet of Things (IoT), Embedded and Desktop markets. The processor design is free of any royalty and is open-sourced under BSD-3 license. A brief overview of the E and C-classes of processors is described below.

1.1.1 E-class

The E-Class [6] is a 32 bit micro processor capable of supporting all extensions of RISC-V ISA as listed in Table 1. The E-class is an In-order 3-stage pipeline having an operational frequency of less than 200MHz on silicon. It is positioned against ARM's M-class (CorTex-M series) cores [3]. The major anticipated use of the E-class of processors is in low-power compute environments, automotive and IoT applications such as smart-cards, motor-controls and home automation. The E-class is also capable of running Real Time Operating Systems (RTOS) like Zephyr OS [10] and FreeRTOS [18].

I	Base Integer Instruction Set
M	Standard Extension for Integer Multiplication and Division
A	Standard Extension for Atomic Instructions
C	Standard Extension for Compressed Instructions

Table 1: RISC-V ISA extensions in SHAKTI SP 2020 SoC's

PINAKA (E32-A35) [12] is a SoC built around E-class [6]. Pinaka is a 32-bit E-class micro controller with 4KB ROM and 128KB BRAM, has 32 General Purpose Input Output (GPIO) pins (out of which upper 8 GPIO pins are dedicated to onboard LEDs and switches), a Platform Level Interrupt Controller (PLIC), a Timer (CLINT), 2 Serial Peripheral (SPI), 3 Universal Asynchronous Receiver Transmitter (UART), 2 Inter Integrated Circuit (I2C), 6 Pulse Width Modulator (PWM), an in-built Xilinx Analog Digital Converter (X-ADC), Soft Float library support, Physical Memory Protection (PMP) enabled, onboard FTDI based debugger and Pin Mux support (Arduino compatible pin assignments). Table 2 describes in detail.
PARASHU (E32-A100) [13] is a SoC built around E-class [6]. Parashu is a 32-bit E-class micro controller with 4 KB of ROM and 256 MB of DDR. The rest of the configuration in this SoC, is the same as PINAKA and is given in Table 2.

1.1.2 C-class

The C-class [5] is an in-order 6-stage 64-bit micro controller supporting the entire RISC-V ISA. It targets the mid-range compute systems supporting 200-800MHz. C-class targets compute applications in the 0.5-1.5 Ghz range. The C-class is customizable for low-power and high-performance variants. It is positioned against ARM's Cortex A35/A55. Linux, SEL4 and FreeRTOS are some of the Operating Systems ported and verified with C-class [5].

VAJRA(C64-A100) [14] is an SoC built around C-class. This SoC is a single-chip 64-bit C-class micro controller with 4KB of ROM and 256MB DDR3 RAM. The rest of the SoC configuration is as given in Table 2. VAJRA is aimed at mid-range application workloads like Industrial controllers and Desktop market. [1, 3].

GPIO Pins	32
Upper 8 pins	Onboard LEDs and switches
PLIC	1
Counter	1
SPI	2
UART	3
I2C	2
PWM	6
ADC	Xilinx
CLINT	1
Float	Soft library
PMP	Enabled
Debugger	Onboard FTDI based
Pin Mux	Yes
Pin assignment	Arduino compatible
Ethernet lite	PARASHU, VAJRA

Table 2: PINAKA, PARASHU and VAJRA SoC details

1.2 Software

SHAKTI class of processors have a wide range of system softwares and tool chain support. There are Software Development Kits (SDK) and Integrated Development Environment (IDE) dedicated for SHAKTI SoCs.

1.2.1 SHAKTI-SDK

Software Development Kits (SDKs) are integral part of any product development. The main objective behind using a SDK is to reduce the development time. The SHAKTI-SDK is a platform that enables developing applications over SHAKTI class of processors. Firmware support is provided for the end users to develop applications. The SHAKTI-SDK is simple and easily customizable. Some of the essential features like DEBUG codes and board support libraries are provided in the SDK.

1.2.2 PlatformIO IDE

PlatformIO is an All-In-One IDE extension in Visual Studio that now supports SHAKTI and its applications across all desktops (Linux, Mac, Windows). This IDE enables developers to code, build, upload, test and debug their applications in a single place without the need to switch to multiple terminals and run complex commands. PlatformIO has an extension that supports SHAKTI development boards. For more details visit
`https://shakti.org.in/sp2020-shakti.html`

1.2.3 Supported Operating systems

Several operating systems have been ported to SHAKTI class of processors. There is also a simple software framework to port different softwares to SHAKTI. Linux, SEL4, Free RTOS, and Zephyr are some of the well known operating systems that work on SHAKTI.
`https://gitlab.com/shaktiproject/software/zephyr-rtos`

2

SECTION

Board Details

SHAKTI support on different types of development boards is crucial as this expands the hardware choice of FPGAs. As part of this effort, initially two varieties of FPGA boards are being supported. They are Xilinx's Arty7 35T and Arty7 100T. This section lists the details on the supported boards and purchase information.

2.1 Development boards

There are development boards for both E and C-class of processors. The details on the board support for different classes of processors are given below.

1. PINAKA [12]
   - *PINAKA* is a SoC based on SHAKTI E-class [6].
   - *PINAKA* is supported on **Artix 7 35T** board.
   - It has an abridged version of 32 bit E-class. It includes I, M, A and C [1].
2. PARASHU [13]
   - *PARASHU* is a SoC based on SHAKTI E-class [6].
   - *PARASHU* is supported on **Artix 7 100T** board.
   - It has an abridged version of 32 bit E-class. It includes I, M, A and C[1].

[1]Refer Table. 1

3. VAJRA[14]

   - *VAJRA* is a SoC based on SHAKTI C-class [5].
   - *VAJRA* is supported on **Artix 7 100T** board.
   - It has an abridged version of 64 bit C-class. It includes I, M, A and C[1].

2.1.1 Board Availability

The boards for the SP2020 competition will be provided by Ministry of Electronics and Information Technology (MeITY). For further details, please refer to
`https://innovate.mygov.in/swadeshi-microprocessor-challenge/`

2.1.2 Documentation

1. Xilinx - Vivado Design Suite
   `https://www.xilinx.com/products/design-tools/vivado.html`

2. Arty A7 - User manual
   `https://reference.digilentinc.com/reference/programmable-logic/arty-a7/reference-manual`

3. Xilinx ADC
   `https://www.xilinx.com/products/intellectual-property/xadc-wizard.html#overview`
   `https://www.xilinx.com/support/documentation/user_guides/ug580-ultrascale-sysmo`
   `pdf`

4. Ethernet Lite
   `https://www.xilinx.com/products/intellectual-property/temac.html#documentation`

SECTION 3

Board setup

The board has to be programmed with any one of the SoCs listed earlier. This section, presents the procedure to set up the board for application development. Topics include connecting a debugger, installing Vivado, building the SHAKTI SoC bit stream, programming the on-board configuration memory and running example programs. Broadly, the following steps are needed to setup the board:

1. Connect the board to the PC.
2. Program the SHAKTI BitStream to the board.
3. Run OpenOCD to test above step.
4. Setup necessary wiring for devices or sensors.

3.1 Powering the board

Plug one end of the micro USB cable into the PC's slot and the other end to the MicroUSB connector (J10) in the board. This will power ON the board. please see Figure 1. The connector J10 is a JTAG and UART port combination. If a sensor requires more power, an external 12V power supply can be connected via Power Jack (J12) . Refer Arty reference manual for detailed power on instructions.

3.2 Setting up the Debugger

This section explains setting up the board for debug mode. The setup for standalone mode is discussed in the Section 5.5 of this manual. The debugger for the board is the

Xilinx FTDI chip on the Arty boards. The details on how to connect the debugger to the board is given below.

3.2.1 Debug interface over Xilinx FTDI (recommended)

The FPGA board is powered on by connecting the micro USB to pin J10. This also connects internally to the UART0 via FTDI interface which provides debugger support.

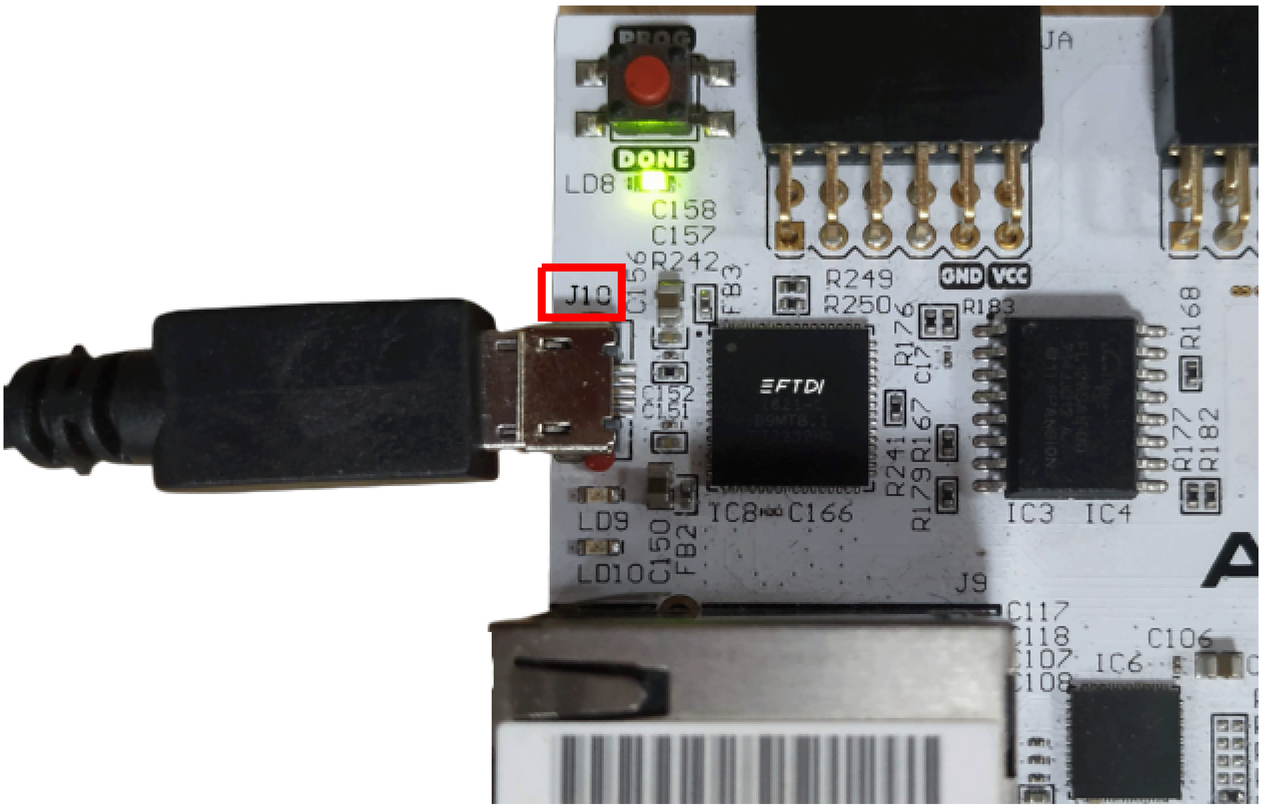


Figure 1: FTDI connection

3.3 Programming SHAKTI

This section walks through implementing SHAKTI C and E-class SoC's on Xilinx's Arty7 100T and 35T. In order to run SHAKTI on Xilinx development boards, the relevant Register Transfer Level (RTL) design has to be programmed on to the FPGA. The procedure to do the same is listed below.

3.3.1 Prerequisites

Ensure that 64-bit version of Ubuntu 18.04 is used. In this machine, the following list of software packages has to be installed.

A. Vivado 2018 and above.

B. Miniterm.

C. OpenOCD.

D. RTL for the SoC

Before starting, the board has to be connected to the PC. The following links host the RTL design.

- Pinaka
  `https://gitlab.com/shaktiproject/sp2020/-/tree/master/e32-a35`
- Parashu
  `https://gitlab.com/shaktiproject/sp2020/-/tree/master/e32-a100`
- Vajra
  `https://gitlab.com/shaktiproject/sp2020/-/tree/master/c64-a100`

The next few sections explains about generating a RTL bisttream, and programming it to FPGA.

3.3.2 Tool Installation

`https://www.youtube.com/playlist?list=PL3o7X5EfdcL4_w0aGs0sQCY33VrRH8d3_`
Note: The above url is the video guide for configuring the FPGA.

A. Installing Vivado

1. If you dont have a Xilinx account, create a free account, using url below:
   `https://www.xilinx.com/registration/create-account.html`
2. Download the Vivado HLx 2018.3 Linux Self Extracting Web Installer, by clicking on the link below:
   `https://www.xilinx.com/member/forms/download/xef-vivado.html?filename=Xilinx_Vivado_SDK_Web_2018.3_1207_2324_Lin64.bin`

3. Make the Vivado installer executable and run it using:
```
chmod +x Xilinx_*.bin
sudo ./Xilinx_*.bin
```
4. Once the installer loads[2], click "Next".
5. Now enter your Xilinx username and password. Then Click "Next".
6. Agree to all three statements and Click "Next". Incase, you disagree you can't proceed further.
7. Select "Vivado HL WebPACK" and click "Next".
8. Click "Reset to Defaults" and then press "Next". [3]
9. By default, the "installation directory" is "/tools/Xilinx". This is the default installation directory. Click "Next".
10. Click "Install" and wait for the installer to finish.
11. Install the Xilinx cable drivers:
```
cd /tools/Xilinx/Vivado/2018.3/data/xicom/cable_drivers/lin64/install_script/install_drivers
sudo ./install_drivers
```
12. Do some permissions cleanup:
```
cd
cd .Xilinx/Vivado
sudo chown -R $USER *
sudo chmod -R 777 *
sudo chgrp -R $USER *
```
13. Add Vivado path to the environmental variable PATH in .bashrc :
```
export PATH=$PATH:/tools/Xilinx/Vivado/2018.3/bin
export PATH=$PATH:/tools/Xilinx/SDK/2018.3/bin
```
14. Test Vivado
```
vivado -version
 Vivado v2018.3 (64-bit)
SW Build 2405991 on Thu Dec 6 23:36:41 MST 2018
IP Build 2404404 on Fri Dec 7 01:43:56 MST 2018
Copyright 1986-2018 Xilinx, Inc.  All Rights Reserved.
```

[2] If installer says, a newer version is available. Please press continue and stay in the current version

[3] Incase, size is a constraint in your system. Just select Artix-7 under "Devices->Production Devices->7 Series. Let, the other two top menus remain untouched

15. Download the board files and copy it to the Vivado repository

```
git clone https://github.com/Digilent/vivado-boards.git
cd vivado-boards/new/board_files
sudo cp -r ./* /tools/Xilinx/Vivado/2018.3/data/boards/board_files
```

B. Installation of Miniterm

```
sudo apt-get install python3-serial
```

C. Installation of Shakti-tools OpenOCD

- Clone the Shakti-tools repository

```
git clone --recursive https://gitlab.com/shaktiproject/software/
shakti-tools.git
```

3.3.3 Programming PINAKA (e32-a35) mcs File onto the FPGA

Note: You must make use of the 32-bit tool chain for programming.

- Connect the board with the USB cable to the PC and move to the HOME directory.

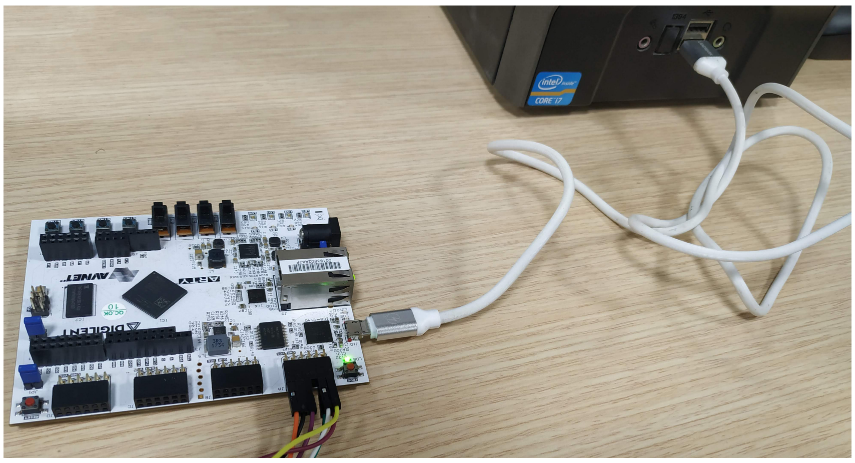


Figure 2: BOARD – PC

- Clone the sp2020 repository to PC.

```
git clone --recursive  https://gitlab.com/shaktiproject/sp2020.git
cd e32-a35/
```

- Program the FPGA.[4]

```
make generate_verilog generate_boot_files ip_build arty_build
generate_mcs program_mcs JOBS=<jobs>
```

- Disconnect the USB Cable from the board and reconnect again.

[4]To re-program the FPGA, please run "make generate_verilog generate_boot_files ip_build arty_build generate_mcs program_mcs JOBS=<jobs>", delete the directory sp2020/e32-a35 and try again.

- Run OpenOCD command.[5]

```
sudo $(which openocd) -f ./shakti-arty.cfg
```

3.3.4 Programming PARASHU(e32-a100) mcs File onto FPGA

Note: You must make use of the 32-bit tool chain for programming.

- Connect the board with the USB cable to the PC and move to the HOME directory.
- Clone the sp2020 repository to PC.

```
git clone --recursive https://gitlab.com/shaktiproject/sp2020.git
cd e32-a100/
```

- Program the FPGA[6] .

```
make generate_verilog generate_boot_files ip_build arty_build
generate_mcs program_mcs JOBS=<jobs>
```

- Disconnect the USB Cable to the board and reconnect again.
- Run OpenOCD command[5].

```
sudo $(which openocd) -f ./shakti-arty.cfg
```

3.3.5 Programming Vajra (c64-a100) mcs File onto FPGA

Note: You must make use of the 64-bit tool chain for programming.

- Connect the board with the USB cable to the PC and move to the HOME directory.
- Clone the sp2020 repository to PC.

```
git clone --recursive https://gitlab.com/shaktiproject/sp2020.git
cd c64-a100/
pip3 install -r requirements.txt
python3 -m configure.main
```

- Program the FPGA[7] .

```
make -j<jobs> generate_verilog
make generate_boot_files ip_build arty_build generate_mcs program_mcs
JOBS=<jobs>
```

- Disconnect the USB Cable to the board and reconnect again.
- Run OpenOCD command[5].

```
sudo $(which openocd) -f ./shakti-arty.cfg
```

[5]If OpenOCD runs and listens on port 3333, then your board is programmed with SHAKTI. You are ready to run applications, benchmarks, etc... on it

[6]To rerun "make generate_boot_files i_build arty_build generate_mcs program_mcs JOBS=<jobs>", delete the directory sp2020/e32-a100 and try

[7]To rerun "make generate_boot_files i_build arty_build generate_mcs program_mcs JOBS=<jobs>", delete the directory sp2020/c64-a100 and try

3.3.6 Programming SHAKTI onto the Arty7 boards with readily available '.mcs' file

The .mcs files are generated and hosted in our repository.

- `https://gitlab.com/shaktiproject/sp2020/-/tree/master/mcs`

Video Tutorial: `https://youtu.be/4EYoEWHGpHI`

Steps for Arty7 35t/100t:

- Connect the board with the USB cable to the PC and move to the HOME directory.
- Please download the .mcs files from the below link.

```
git clone --recursive https://gitlab.com/shaktiproject/sp2020/-/tree/
master/mcs
```

- Start Vivado 2018.3 and Open Hardware Manager.

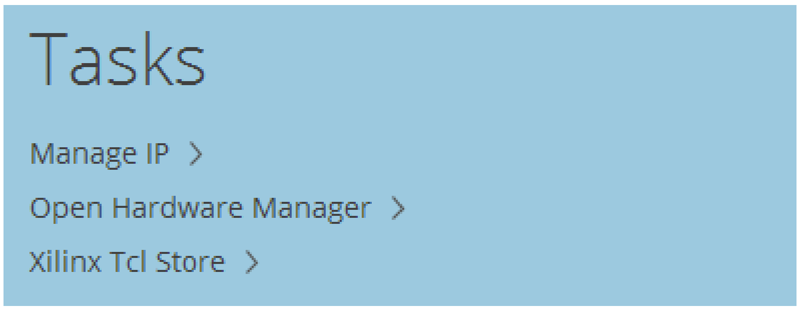
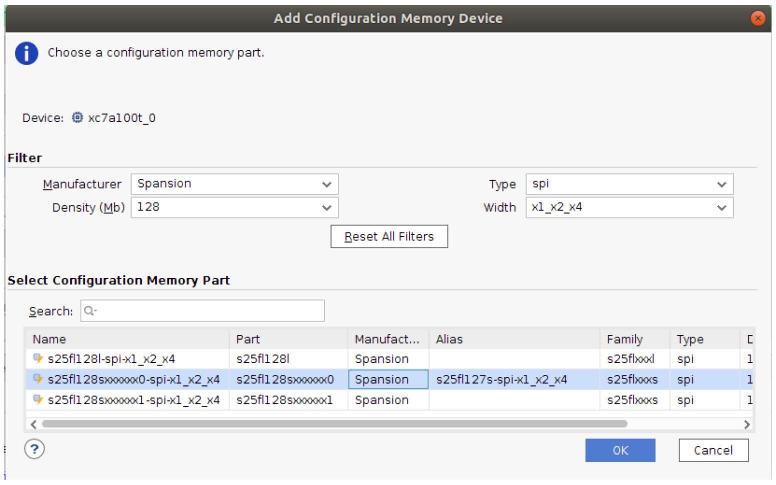


Figure 3: Vivado 2018.3

- Click on open target, Proceed to click on Auto connect. Now the FPGA board is connected to Vivado.
- Right-click on the board name "xc7a100t" or "xc7a35t". Select "Add Configuration memory device".



Choose a configuration memory part.

Device: xc7a100t_0

Filter

Manufacturer: Spansion
Type: spi
Density (Mb): 128
Width: x1_x2_x4

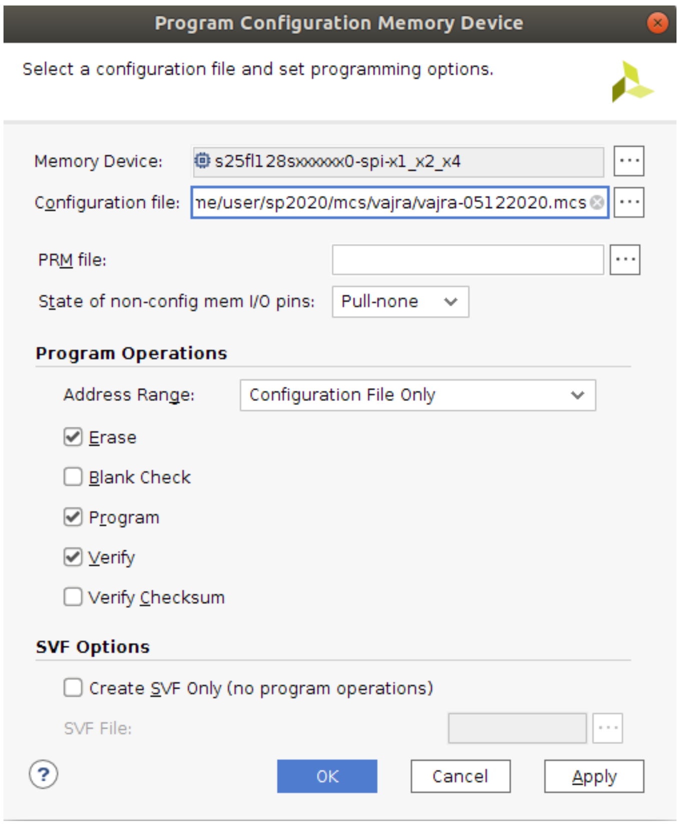
Select Configuration Memory Part

Search:

Name	Part	Manufact...	Alias	Family	Type	D
s25fl128l-spi-x1_x2_x4	s25fl128l	Spansion		s25flxxxl	spi	1
s25fl128sxxxxxx0-spi-x1_x2_x4	s25fl128sxxxxxx0	Spansion	s25fl127s-spi-x1_x2_x4	s25flxxxs	spi	1
s25fl128sxxxxxx1-spi-x1_x2_x4	s25fl128sxxxxxx1	Spansion		s25flxxxs	spi	1

Figure 4: Add Configuration memory device

- Select:
  - Manufacturer as Spansion or Micron (based on the Micron or Spansion flash on the board)
  - Density -128
  - Type - spi
  - Width - $x1\_x2\_x4$
- Select the Configuration memory part and click on Ok.



Program Configuration Memory Device

Select a configuration file and set programming options.

Memory Device: s25fl128sxxxxxx0-spi-x1_x2_x4

Configuration file: ne/user/sp2020/mcs/vajra/vajra-05122020.mcs

PRM file:

State of non-config mem I/O pins: Pull-none

Program Operations

Address Range: Configuration File Only

[✓] Erase

[ ] Blank Check

[✓] Program

[✓] Verify

[ ] Verify Checksum

SVF Options

[ ] Create SVF Only (no program operations)

SVF File:

Figure 5: Program Configuration Memory Device

- In the new window, select the downloaded .mcs file and click ok to program SHAKTI onto the board.
- Disconnect the USB Cable to the board and reconnect again.
- Run OpenOCD command[5].

```
sudo $(which openocd) -f ./shakti-arty.cfg
```

4

SECTION

SoC Device Information

The SHAKTI based SoC's consist of processor, memory and various Input-Output devices (I/O). The devices in the SoC are memory mapped. Memory mapped I/O is a method to communicate between the core and the peripheral devices. In this method the device address and the internal registers of the devices are mapped to memory locations. The processor and these devices communicate with the help of the AXI system bus. The next two sections deal with the list of devices and the memory map of the devices on different shakti based SoC's.

Sl. No	Device name	Abbreviation
1	BRAM	Block Random Access Memory
2	CLINT	Core Local INterrupt Controller
3	DDR	Double Data Rate RAM
4	GPIO	General Purpose Input Output
5	I2C	Inter-Integrated Circuit
6	PLIC	Platform Level Interrupt Controller

Sl. No	Device name	Abbreviation
7	PWM	Pulse Width Modulation
8	SDRAM	Synchronous Dynamic Random Access
9	SPI	Serial Peripheral
10	UART	Universal Asynchronous Receiver Transmitter
11	XADC	Xlinix Analog Digital Converter
12	Ethernet lite	AXI Ethernet Lite MAC

Table 3: Device description table

4.1 Device memory map

The overall layout of the memory map of a device based around the SHAKTI class of processor is listed below. This allows easy porting of software.

4.1.1 PINAKA memory map

Sl.No	Peripheral	Base Address Start	Base Address End
1.	Memory (TCM)	0x80000000	0x87FFFFFF
2.	Debug	0x00000010	0x0000001F
3.	PWM 0	0x00030000	0x000300FF
4.	PWM 1	0x00030100	0x000301FF
5.	PWM 2	0x00030200	0x000302FF
6.	PWM 3	0x00030300	0x000303FF
7.	PWM 4	0x00030400	0x000304FF

Sl.No	Peripheral	Base Address Start	Base Address End
8.	PWM 5	0x00030500	0x000305FF
9.	SPI 0	0x00020000	0x000200FF
10.	SPI 1	0x00020100	0x000201FF
11.	UART0	0x00011300	0x00011340
12.	UART1	0x00011400	0x00011440
13.	UART2	0x00011500	0x00011540
14.	CLINT	0x02000000	0x020BFFFF
15.	GPIO	0x00040100	0x000401FF
16.	PLIC	0x0C000000	0x0C01001F
17.	I2C0	0x00040000	0x000400FF
18.	XADC	0x00041000	0x000413FF
19.	Boot Rom	0x00001000	0x00002FFF
20.	I2C1	0x00041400	0x000414FF
21.	PinMux	0x00041500	0x00041510

Table 4: PINAKA class memory map

4.1.2 PARASHU memory map

Sl. No	Peripheral	Base Address Start	Base Address End
1.	Memory (DDR)	0x80000000	0x8FFFFFFF
2.	Debug	0x00000010	0x0000001F
3.	UART0	0x00011300	0x00011340
4.	UART1	0x00011400	0x00011440
5.	UART2	0x00011500	0x00011540
6.	I2C0	0x00040000	0x000400FF
7.	GPIO	0x00040100	0x000401FF
8.	CLINT	0x02000000	0x020BFFFF
9.	PLIC	0x0C000000	0x0C01001F
10.	PWM0	0x00030000	0x000300FF
11.	PWM1	0x00030100	0x000301FF
12.	PWM2	0x00030200	0x000302FF
13.	PWM3	0x00030300	0x000303FF

Sl. No	Peripheral	Base Address Start	Base Address End
14.	PWM4	0x00030400	0x000304FF
15.	PWM5	0x00030500	0x000305FF
16.	SPI0	0x00020000	0x000200FF
17.	SPI1	0x00020100	0x000201FF
18.	I2C1	0x00041400	0x000414FF
19.	XADC	0x00041000	0x000413FF
20.	PinMux	0x00041500	0x000415FF
21.	Boot Rom	0x00001000	0x000415FF

Table 5: PARASHU memory map

4.1.3 VAJRA memory map

Sl. No	Peripheral	Base Address Start	Base Address End
1.	Memory (DDR)	0x80000000	0x87FFFFFF
2.	Debug	0x00000010	0x0000001F
3.	UART0	0x00011300	0x00011340
4.	UART1	0x00011400	0x00011440
5.	UART2	0x00011500	0x00011540
6.	I2C0	0x00040000	0x000400FF
7.	GPIO	0x00040100	0x000401FF
8.	CLINT	0x02000000	0x020BFFFF
9.	PLIC	0x0C000000	0x0C01001F
10.	PWM0	0x00030000	0x000300FF
11.	PWM1	0x00030100	0x000301FF
12.	PWM2	0x00030200	0x000302FF
13.	PWM3	0x00030300	0x000303FF

Sl. No	Peripheral	Base Address Start	Base Address End
14.	PWM4	0x00030400	0x000304FF
15.	PWM5	0x00030500	0x000305FF
16.	SPI0	0x00020000	0x000200FF
17.	SPI1	0x00020100	0x000201FF
18.	I2C1	0x00041400	0x000414FF
19.	XADC	0x00041000	0x000413FF
20.	PinMux	0x00041500	0x000415FF
21.	Boot Rom	0x00001000	0x00041400
22.	Xil Ethernet lite	0x00044000	0x00047FFF

Table 6: VAJRA memory map

Note: The Xadc and Ethernet are Xilinx IP's. Please refer the Document section (2.1.2).

SECTION 5

Software Development Flow

This section presents the software framework for design and implementation of embedded/IoT applications. We discuss in detail, on how to develop applications using the SHAKTI Software Development Kit (SHAKTI-SDK).

5.1 SHAKTI-SDK Architecture

The SHAKTI-SDK is a C/C++ platform to develop applications over SHAKTI. The SDK has the necessary firmware code and framework to develop newer applications on the hardware. The framework is light weight and customizable.

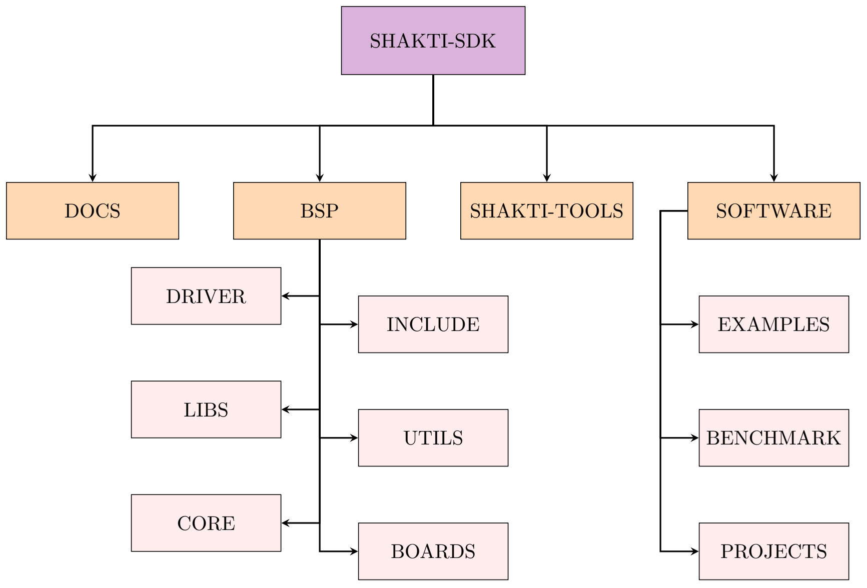


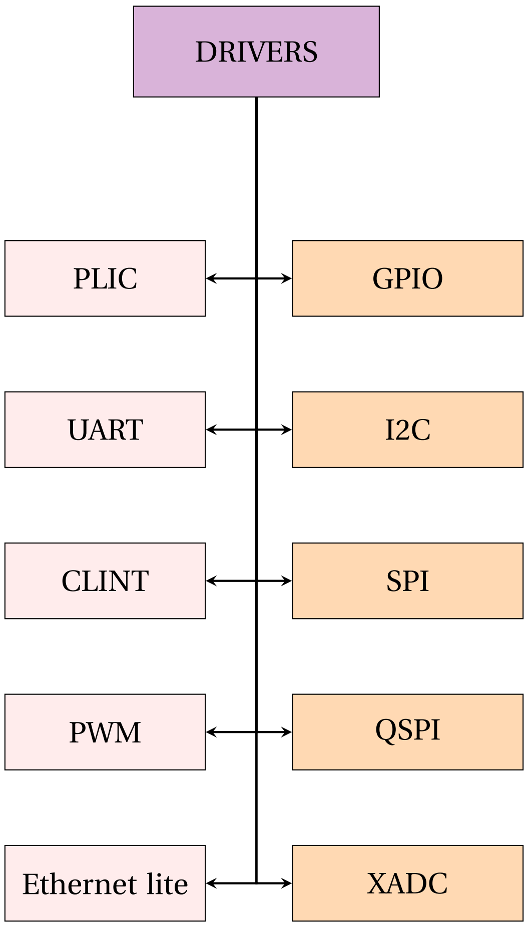
Figure 6: SDK architecture

5.1.1 Board Support Package

The BSP consists of system files and driver files for various devices. It contains certain platform dependent definitions for each board. Essentially, the BSP is the layer above the hardware. It includes the following sub directories:

1. Drivers

   The drivers are a set of software constructs that help software applications to access the devices in the SoC. They are generally low level API's, that execute a particular task in the hardware.

2. Include

   This directory has header files for core and for each driver. The board independent variable/macro definitions and declarations pertaining to each driver is included here.

3. Libs

   The library utilities and the boot code are hosted here. Library is a common place for reusable code. The libraries can be compiled as a separate "lib" file and used.

4. Core

   The core usually has functions related to the startup codes, trap handlers and interrupt vectors.The code related to memory initialisation are also available here.

5. Utils

   This contains the code related to standalone mode feature of the shakti processor.

6. Third_party

   This directory provides support for external boards as well as custom boards. This includes the definitions of board specific functions such as console drivers.

5.1.2 Software

The software directory provides a platform for developing various applications, independent of the underlying BSP. All the applications/projects developed in SHAKTI-SDK reside in this directory. In general, an application will involve writing high level C/C++ code that uses BSP API's. The software directory is broadly classified in to three sub-directories,

1. Projects

   This directory consists of applications developed using different sensors. These are usually a combination of standalone applications.

2. Benchmarking

   Applications or bare metal codes that are developed for bench-marking a core reside here. These programs usually describe the capability of the SHAKTI class of processors.

3. Examples

   This is the place where any new standalone application is developed. Few example programs involving sensors are already developed for different peripherals and kept here. These programs demonstrate the integration of BSP and the core support libraries with the user programs.

5.1.3 Makefile

To compile programs more efficiently the GNU's MAKE utility is used. The make utility uses the *Makefile* to compile program from source code. The output generated by the MAKE utility is in ELF format. The Makefile has support for different target boards and applications. The Makefile's are mostly non-recursive and devoid of complex expressions. The supported make commands are listed below.

- make help

  Lists the possible commands supported in Makefile.

- make list_targets

  List the boards that are supported.

- make list_applns

Lists the samples that are available in SHAKTI-SDK

- make software PROGRAM=? TARGET=?

  PROGRAM can be found from "make list_applns"

  TARGET= pinaka or parashu or vajra

  Default TARGET is pinaka

- make debug[8] PROGRAM=? TARGET=?

  PROGRAM can be found from "make list_applns"

  TARGET= pinaka or parashu or vajra

  Default TARGET is pinaka

  debug command adds the debug support to applns.

- make all TARGET=?

  TARGET= pinaka or parashu or vajra

  Default TARGET is pinaka

  All the applications under example directory are compiled for above target.

- make clean

  clean all the executable.

  The design overrides the executable generated by the last target with current target.

- make clean CLEAR=?

  CLEAR ?= any application under list_applns

  clean the executable for a application.

[8]This command is necessary, if the program is going to be debugged using gdb

5.2 Setting up the SHAKTI-SDK

5.2.1 Pre-requisites

Ensure that the following packages are installed in the host system. To solve the software dependencies, copy and paste each of the commands below in terminal and press enter.

```
sudo apt-get install autoconf automake autotools-dev curl make-guile
```

```
sudo apt install libmpc-dev libmpfr-dev libgmp-dev libusb-1.0-0-dev bc
```

```
sudo apt install gawk build-essential bison flex texinfo gperf libtool
```

```
sudo apt install make patchutils zlib1g-dev pkg-config libexpat-dev
```

```
sudo apt install libusb-0.1 libftdi1 libftdi1-2
```

```
sudo apt install libpython3.6-dev
```

5.2.2 Download the SHAKTI-SDK repository

SHAKTI-SDK repository contains scripts, board support packages to build your application. It can be cloned by running the following command:

```
git clone https://gitlab.com/shaktiproject/software/shakti-sdk.git
```

5.2.3 Download the SHAKTI-TOOLS repository

The SHAKTI-TOOLS repository contains both 64-bit and 32-bit toolchain, for building your application. It can be cloned by running the following command:

```
git clone --recursive https://gitlab.com/shaktiproject/software/
shakti-tools.git
```

If you had omitted --recursive option earlier, then run the command below to clone the submodules repository:

```
git submodule update --init --recursive
```

5.2.4 Setting up SHAKTI Tool-chain

SHAKTI uses RISC-V tools. The tool-chain can be installed in two ways,

- Manual method
  - · Build and install toolchain from riscv-tools [8].

- The riscv-tools repository has the readme to install RISC-V toolchain.

- Automatic method (Recommended)
  - The tool-chain executable is hosted in the SHAKTI-TOOLS repository [9].
  - The tool-chain was prebuilt in Ubuntu 18.04 system and hosted here.
  - The absolute path of the tool-chain has to be added to "PATH" variable and exported, to use it across the file system.
  - The steps to export the tool chain to the PATH variable is provided below,

Steps for automatic method

```
$ pwd
/home/user
$ git clone --recursive https://gitlab.com/shaktiproject/software/
shakti-tools.git
$ cd shakti-tools
$ pwd
/home/user/shakti-tools
For 32-bit toolchain
$ SHAKTITOOLS=/home/user/shakti-tools
$ export PATH=$PATH:$SHAKTITOOLS/bin
$ export PATH=$PATH:$SHAKTITOOLS/riscv32/bin
$ export PATH=$PATH:$SHAKTITOOLS/riscv32/riscv32-unknown-elf/bin
For 64-bit toolchain
$ SHAKTITOOLS=/home/user/shakti-tools
$ export PATH=$PATH:$SHAKTITOOLS/bin
$ export PATH=$PATH:$SHAKTITOOLS/riscv64/bin
$ export PATH=$PATH:$SHAKTITOOLS/riscv64/riscv64-unknown-elf/bin
$ which riscv64-unknown-elf-gcc
/home/user/shakti-tools/riscv64/bin/riscv64-unknown-elf-gcc
```

Things to know:

1. Please add the particular toolchain needed (32 or 64 bit toolchain) in **.bashrc** file in the home directory to set the PATH **permanently** instead of that particular session.

2. The variable $SHAKTITOOLS is the location of SHAKTI-TOOLS in the file system.

3. The command **which riscv64-unknown-elf-gcc** helps you to verify whether tool-chain path is exported correctly.

5.2.5 Update the SDK or TOOLS

To update the SDK or TOOLS repository to the latest version, move to the respective repository **(cd shakti-sdk) or cd shakti-tools** and use the command below.

```
$ git pull origin master
$ git submodule update --init --recursive
```

This updates the required repository to its latest version

5.3 Application Development

As discussed earlier, SHAKTI-SDK helps in developing applications for SHAKTI class of processors. The steps to develop a small application using SHAKTI-SDK is discussed. SHAKTI-SDK comes with a separate repository for Applications and Projects. A project usually has its own design environment, except that it imports the BSP. An application is a simple program that demonstrates the working of sensors using SHAKTI class of processors. Any program with a smaller memory footprint is put under Applications. Before developing an application, make sure that the pre-requisites mentioned below are ready.

- Board is up and running with SHAKTI.
- SHAKTI-SDK has been downloaded and installed.
- SHAKTI Tool chain has been installed.
- PATH variable has been set on the tool chain.

5.3.1 Steps to add a new application to SHAKTI-SDK

An application program has to use the the BSP API's for any device access. The steps followed to develop a simple program is listed below

- The new application is created under one of the *example/XYZ_applns* directory.
- XYZ should be a device type in the SoC. For example it can be UART, I2C, etc...
- Lets assume, we are under the XYZ_applns directory.
- Create a directory for the new application and name it accordingly.
- Inside the directory, create source and header files for the application.
- Create and edit a new Makefile for the application (refer existing examples).
- The name of the application directory corresponds to the name of the application.
- Make an entry in the existing Makefile under *./shakti-sdk/software/examples* for the new application.

- Now typing *make list-applns* will list the new application as one under SHAKTI-SDK.

5.3.2 My first program !

Follow the steps given below to compile and run a program to print "Hello World!"

- The device required is UART. Include UART device headers for the program.
- Write your program under *software/examples/uart_applns/*.
- Create a directory called 'first'.
- Create a first.c file and write a program to print "Hello World !".
- Create and edit a Makefile for the program and save in the 'first' directory (refer software/examples directory).
- Make a new entry for the program in the 'existing Makefile' under examples directory.

5.3.3 Build

The make commands in SHAKTI-SDK gives various options to build and run a program. The list of *make* commands can be found by typing **make help** in the terminal. Once the program is built using MAKE command, the ELF file is generated. The ELF file is the final executable that can be loaded into the memory and run.

```
$ cd shakti-sdk
$ make software PROGRAM=hello[9] TARGET=PINAKA
```

Interpreting above commands:

- PROGRAM is the new one created. It is listed by typing "make list_applns".
- TARGET= PINAKA or PARASHU or VAJRA.
- Default TARGET is PINAKA.

5.3.4 Run

Once the application is built, the executable is generated in the output directory. The executable is in ELF file format and they have the extension *.shakti*. There are two modes to run an application on the arty boards. The two modes are discussed in the following sections.

5.4 Running application in Debug mode

After the ELF for the target application is generated, the program can be run in Debug or Standalone mode. Debug mode helps in incremental development. It also helps to understand the program flow and helps debug applications easily.

The Arty35/100T board should be connected to the OpenOCD debugger, in order to debug your program using the RISC-V GNU Debugger (GDB) software. The standard GDB commands supported by RISC-V GDB can be used to debug. Since we have built the application already, we can start loading it to the Arty board and test. The following steps list out the actions to be taken:

5.4.1 Steps to run

Prerequisites

1. Install miniterm

   ```
   $ sudo apt-get install python3-serial
   ```

2. Open three terminals, one for each of the following

   a. One terminal for UART terminal display.

   b. Another for OpenOCD

   c. And the last one for GDB server.

Follow the steps below to set up and run programs[10]

1. In the first terminal, open a serial port to display output from UART.

   ```
   $ sudo miniterm.py /dev/ttyUSB0 19200
   ```

2. In the second terminal, launch OpenOCD with super user (sudo) permission. Please ensure that, you are in the SHAKTI-SDK directory.

   For example,

   ```
   $ pwd
   /home/user/shakti-sdk ------> you are in the right directory
   ```

   Press reset in the board and run the following commands.

   ```
   $ cd ./bsp/third_party/artix7_35t
   $ sudo $(which openocd) -f ftdi.cfg
   ```

[10]Open the terminals in the above mentioned order.

3. In the third terminal launch RISC-V GDB. Applications will be loaded and run here.

```
$ riscv32-unknown-elf-gdb (For RISC-V 32bit use)
(gdb) set remotetimeout unlimited
(gdb) target remote localhost:3333 (Connect to remote target)
(gdb) file path/to/executable (Specify the program to be debugged)
(gdb) load (Load the file to memory)
(gdb) c (Execute the program)
```

Note:

1. "/dev/ttyUSB0" - ttyUSB means "USB serial port adapter" and the "0" is the USB device number. The device number can change based on USB port availability.
2. For C-class (64 bit) applications, please use riscv64-unknown-elf-gdb instead of riscv32-unknown-elf-gdb.

5.4.2 Application flow

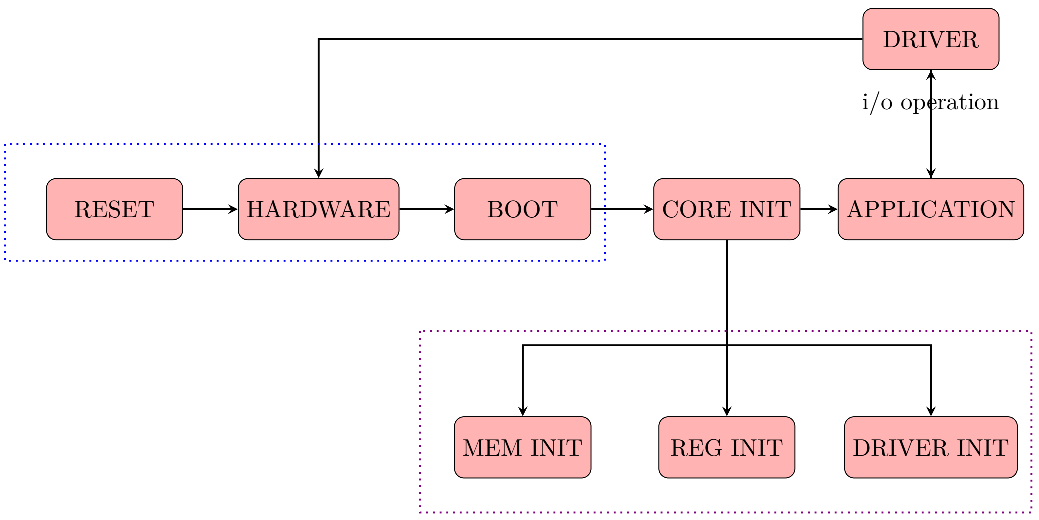


Figure 7: Execution flow, after every reset

5.5 Running application in Standalone mode

Until now, we have been running applications in Debug mode. We need a Host PC to build and run the application every time. In standalone mode, on reset, application starts running. The application is no longer downloaded from the PC through a debugger and executed. Instead, it is stored in the flash memory. When the system starts, the boot loader loads the application from the flash memory to the physical memory (RAM). Then the control transfers to the application residing in RAM. This mode of running the application is usually used in standalone systems.

5.5.1 Steps to generate standalone user application

The `make upload` command is used to build and upload the application to the flash automatically. The SHAKTI-SDK has a *uploader* tool that is used to load a content (such as ELF) to flash, after building the image.

```
$ cd shakti-sdk
$ make upload PROGRAM= <bare metal appln> TARGET=pinaka
```

Interpreting above commands:

- PROGRAM is the new bare metal user application that is created. It is listed by typing "`make list_applns`".
- TARGET= pinaka, refers to the target SoC.

Tip: It is always better to run the program in debug mode and once you are confident that there are no bugs, then use the standalone mode.

SECTION A

Device pin mapping

The Device Pin Mapping corresponding to all the three different SoC's is listed below. The pin mapping is same for all the three SoC's.

A.1 PINAKA, PARASHU and VAJRA

Sl. No	Pin Description	Pin mapping	Peripheral
1.1	GPIO0	CK_IO[0] (J4[1],IO - Lower)	GPIO
1.2	GPIO1	CK_IO[1] (J4[3],IO - Lower)	
1.3	GPIO2	CK_IO[2] (J4[5],IO - Lower)	
1.4	GPIO3	CK_IO[3] (J4[7],IO - Lower)	
1.5	GPIO4	CK_IO[4] (J4[9],IO - Lower)	
1.6	GPIO5	CK_IO[5] (J4[11],IO - Lower)	

Sl. No	Pin Description	Pin mapping	Peripheral
1.7	GPIO6	CK_IO[6] (J4[13],IO - Lower)	
1.8	GPIO7	CK_IO[7] (J4[15],IO - Lower)	
1.9	GPIO8	CK_IO[8] (J2[1],IO - Higher)	
1.10	GPIO9	CK_IO[9] (J2[3],IO - Higher)	
1.11	GPIO10	CK_IO[10] (J2[5],IO - Higher)	
1.12	GPIO11	CK_IO[11] (J2[7],IO - Higher)	
1.13	GPIO12	CK_IO[12] (J2[9],IO - Higher)	
1.14	GPIO13	CK_IO[13] (J2[11],IO - Higher)	
1.15	GPIO14	CK_IO[26] (J4[2],IO - Lower)	
1.16	GPIO15	CK_IO[27] (J4[4],IO - Lower)	
1.17	GPIO16	LD4	
1.18	GPIO17	LD5	
1.19	GPIO18	LD6	
1.20	GPIO19	LD7	
1.21	GPIO20	BTN0	

Sl. No	Pin Description	Pin mapping	Peripheral
1.22	GPIO21	BTN1	
1.23	GPIO22	BTN2	
1.24	GPIO23	BTN3	
1.25	GPIO24	JD[1] -1P	
1.26	GPIO25	JD[2] -1N	
1.27	GPIO26	JD[3] -2P	
1.28	GPIO27	JD[4] -2N	
1.29	GPIO28	JD[7] -3P	
1.30	GPIO29	JD[8] -3N	
1.31	GPIO30	JD[9] -4P	
1.32	GPIO31	JD[10] -4N	
2.1	ADC P4,N4	CKA0	Single ended ADC
2.2	ADC P5,N5	CK A1	
2.3	ADC P6,N6	CK A2	

Sl. No	Pin Description	Pin mapping	Peripheral
2.4	ADC P7,N7	CK A3	
2.5	ADC P15,N15	CK A4	
2.6	ADC P0,N0	CK A5	
3.1	ADC 12P	CK A6	Double ended ADC
3.2	ADC 12N	CK A7	
4.1	ADC 13P	CK A8	Double ended ADC
4.2	ADC 13N	CK A9	
5.1	ADC 14P	CK A10	Double ended ADC
5.2	ADC 14N	CK A11	
6.1	I2C0_SDA	JB_P[1]	I2C0
6.2	I2C0_SCL	JB_N[1]	
7.1	I2C1_SDA	CK_SDA	I2C1
7.2	I2C1_SCL	CK_SCL	
8.1	UART0 TX	J10	UART
8.2	UART0 RX	J10	

Sl. No	Pin Description	Pin mapping	Peripheral
9.1	UART1 TX	CK_IO[1] (J4[3],IO - Lower)	
9.2	UART1 RX	CK_IO[0] (J4[1],IO - Lower)	
10.1	UART2 TX	CK_IO[3] (J4[7],IO - Lower)	
10.2	UART2 RX	CK_IO[2] (J4[5],IO - Lower)	
11.1	PWM 0	CK_IO[3] (J4[7],IO - Lower)	PWM PINS
11.2	PWM 1	CK_IO[5] (J4[11],IO - Lower)	
11.3	PWM 2	CK_IO[6] (J4[13],IO - Lower)	
11.4	PWM 3	CK_IO[9] (J2[3],IO - Higher)	
11.5	PWM 4	CK_IO[10] (J2[5],IO - Higher)	
11.6	PWM 5	CK_IO[11] (J2[7],IO - Higher)	
12.1	SPI0 CS	QSPI_CS	SPI0
12.2	SPI0 SCLK	QSPI_DQ[2]	
12.3	SPI0 MISO	QSPI_DQ[1]	
12.4	SPI0 MOSI	QSPI_DQ[0]	
13.1	SPI1 CS	CK_IO[10] (J2[11],IO - Higher)	SPI1

Sl. No	Pin Description	Pin mapping	Peripheral
13.2	SPI1 SCLK	CK_IO[13] (J2[11],IO - Higher)	
13.3	SPI1 MISO	CK_IO[12] (J2[11],IO - Higher)	
13.4	SPI1 MOSI	CK_IO[11] (J2[11],IO - Higher)	

Table 7: Device pin map

B

SECTION

Understanding PinMux design

Example B.1

How to Configure a PinMux Register?

- A pair of bit, maps to a device pin. A device can have one or more pins.
- The bit pair can take values 00, 01, 10. Undefined behavior, for value 11.
- If all the pair of bit's are zero. Then all the IO pins are configured as GPIO.
- If bits |7 |6| are set to 10. Then, PWM0 is enabled.
- If bits |7 | 6| are set to 01. Then, U2TX is enabled.

PinMux	**Bit Positions**															
Config Value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
00	-		-		-		-		-		GP13		GP12		GP11	
01	-		-		-		-		-		S1_CK		S1_SO		S1_SI	
10	-		-		-		-		-		-		-		PWM5	

Table 8: pinmux memory map (upper bytes)

PinMux	**Bit Positions**							
Config Value	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
00	GP10	GP9	GP6	GP5	GP3	GP2	GP1	GP0
01	S1_CS	-	-	-	U2TX	U2RX	U1TX	U1RX
10	PWM4	PWM3	PWM2	PWM1	PWM0	-	-	-

Table 9: pinmux memory map (lower bytes)

Note:

1. The above tables are used for configuring GPIO pins as I2C, PWM, SPI & UART.
2. For example UART needs two pins. Therefore, UART Tx and Rx pins are configured to use the GPIO pins as UART. Similarly, SPI needs 4 pins.
3. Peripherals not mentioned in this table have their own dedicated pins.
4. **U1 & U2** corresponds to **UART1** and **UART2**.
5. **S1** in the table corresponds to **SPI1**.
5. **-** denotes these bit position are unused.

SECTION

C Reach us at!

Any issues or clarification in SDK or documentation can be raised under issues in the following url `https://gitlab.com/shaktiproject/software/shakti-sdk`. Before raising an issue, please check if there are any similar issues.

C.1 Steps to create an issue

- Go to Issues.
- After clicking on New Issue you will get an option to select a template.
- Click on choose template, list of available templates will be displayed, Select template "Bug".
- Once the template named 'Bug' is selected, the description text box is populated by the template.
- Please fill all the fields in the description text box.

SECTION 5

Bibliography

[1] N. Gala, A. Menon, R. Bodduna, G. S. Madhusudan and V. Kamakoti, "SHAKTI Processors: An Open-Source Hardware Initiative," 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), Kolkata, 2016, pp. 7-8.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7434907&isnumber=7434885

[2] Design of the RISC-V Instruction Set Architecture
https://riscv.org/specifications/

[3] SHAKTI Processor Program Open-source Processor Development Ecosystem
https://shakti.org.in

[4] Shakti Software Development Kit
https://gitlab.com/shaktiproject/software/shakti-sdk

[5] SHAKTI C-class Micro Architecture Design
https://gitlab.com/shaktiproject/cores/c-class

[6] SHAKTI E-class Micro Architecture Design
https://gitlab.com/shaktiproject/cores/e-class

[7] RISC-V Cores
https://riscv.org/risc-v-cores/

[8] RISC V Tool Chain
https://gitlab.com/shaktiproject/software/riscv-tools

[9] Generated RISC V Tool Chain
https://gitlab.com/shaktiproject/software/shakti-tools

[10] Zephyr Project and Zephyr OS Kernel, [online],
https://www.zephyrproject.org

[11] Arty A7-100T and 35T with RISC-V
https://www.digikey.in/en/product-highlight/x/xilinx/arty-a7-100t-and-35t-with-risc-v

[12] Pinaka
https://gitlab.com/shaktiproject/sp2020/-/tree/master/e32-a35

[13] Parashu
https://gitlab.com/shaktiproject/sp2020/-/tree/master/e32-a100

[14] Vajra
https://gitlab.com/shaktiproject/sp2020/-/tree/master/c64-a100

[15] Xilinx Ethernet Lite
https://www.xilinx.com/products/intellectual-property/temac.html#documentation

[16] PlatformIO Extensions for VSCODE
https://platformio.org/

[17] SHAKTI Support on PlatformIO
https://github.com/platformio/platform-shakti

[18] SHAKTI Support on FreeRTOS
https://gitlab.com/shaktiproject/software/FreeRTOS