SHAKTI Development Board

User Manual

developed by: SHAKTI Development Team @ iitm '19

shakti.org.in

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

0.1 Proprietary Notice

`The project was funded by MeITY, Government of India`

0.2 Release Information

Version	Date	Changes
0.1	February 27, 2020	Initial Release
0.2	June 22, 2020	Updated Section 5.2

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 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]. While the primary focus of the team is architecture research, these SoCs will be competitive with commercial offerings in the market with respect to area, power and performance. The Current SoC (as of 2019) developments are Controller class (C-Class) [5] and Embedded Class (E- Class) [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 32/64 bit micro-controller 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 [19].

E-arty35T[14] is an SoC built around E-class [1]. The *E-arty35T* SoC is a single-chip 32-bit E class micro controller with 128kB RAM, has 32 General Purpose Input Output (GPIO) pins (out of which upper 16 GPIO pins are dedicated to onboard LEDs and switches), a Platform Level Interrupt Controller (PLIC), a Counter, 2 Serial Peripheral (SPI), 2 Universal Asynchronous Receiver Transmitter (UART), 1 Inter Integrated Circuit (I2C), 6 Pulse Width Modulator (PWM) and an in-built Xilinx Analog Digital Converter (X-ADC).

I	Base Integer Instruction Set
M	Standard Extension for Integer Multiplication and Division
A	Standard Extension for Atomic Instructions
F	Standard Extension for Single-Precision Floating-Point
D	Standard Extension for Double-Precision Floating-Point
C	Standard Extension for Compressed Instructions

Table 1: RISC-V ISA extensions in SHAKTI

1.1.2 C-class

The C-class [5] is an in-order 6 stage 64-bit micro controller supporting the entire stable RISC-V ISA. It targets the mid-range compute systems running over 200-800MHz. It can also be customized upto 2 Ghz. The C-class is highly customizable and there are variants for low-power and high-performance. It is positioned against ARM's Cortex A35/A55. Linux, Sel4 and Free RTOS are some of the Operating systems ported and verified with C-class [3].

C-arty100T [15] is a SoC build around C-class. The *C-arty100T* SoC is a single-chip 64-bit C class micro controller with 128MB DDR3 RAM, 16 General Purpose Input Output (GPIO) pins, a Platform Level Interrupt Controller (PLIC), a Counter, 1 Universal Asynchronous Receiver Transmitter (UART) and 1 Inter Integrated Circuit (I2C). It is aimed at mid-range application workloads and has a very low power profile, as well as support for optional memory protection [1] [3].

1.2 Tapeouts

Two tapeouts of the C-class processors have been performed. They have been code named as *RIMO* and *Rise-creek*.

1.2.1 RIMO

RIMO is the code name of the SHAKTI C-class based SoC that has been taped-out at Semi-Conductor Laboratory (SCL) at Chandigarh using 180 nm process technology. The 144 sq.mm. chip has been tested to operate at a frequency of up to 70 MHz. The chip has been packaged on a 208-pin Ceramic Quad Flat Pack (CQFP).

Figure 1: RIMO

The features of the SoC are :

- In-order 5 stage 64-bit micro controller supporting the entire stable RISC-V ISA (RV64IMAFD).
- Compatible with privilege spec (v1.10) of RISC-V ISA and supports the sv39 virtualisation scheme.
- Includes a branch predictor with a Return-Address-Stack.
- Pipelined IEEE-754 compliant single and double precision floating point units and Multi-channel Direct Memory Access (DMA) support.
- Peripherals like 2 x I2C, 2 x UART, 2 x QSPI, a Debugger, a 256KB tightly coupled memory, 32-bit GPIOs and an expansion bus that can be connected to an FPGA.

1.2.2 Risecreek

CREEK is the code name of the SHAKTI C-class based SoC that has been taped-out at INTEL,Oregon, USA using 22 nm process technology. The 16 sq.mm. chip has been tested to operate at a frequency of up to 350 MHz. The chip has been packaged on a 208-pin Ball Grid Array (BGA).

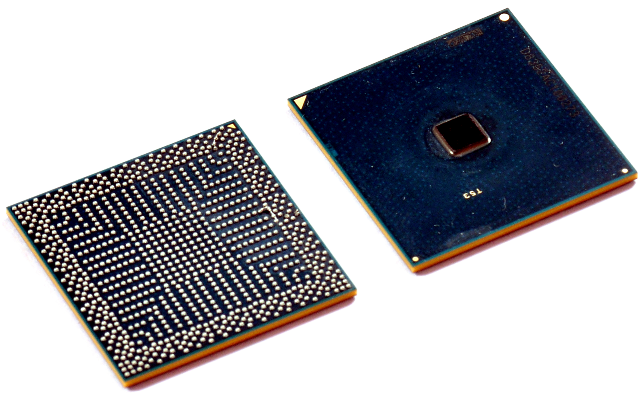


Figure 2: RISECREEK

The features of the SoC are :

- In-order 5 stage 64-bit micro controller supporting the entire stable RISC-V ISA (RV64IMAFD).
- Compatible with privilege spec (v1.10) of RISC-V ISA and supports the sv39 virtualisation scheme.
- Includes a branch predictor with a Return-Address-Stack.
- Pipelined IEEE-754 compliant single and double precision floating point units Multi-channel Direct Memory Access (DMA) support.
- Peripherals like 2 x I2C, 1 x UART, 2 x QSPI, a Debugger, a 128KB tightly coupled memory, 32-bit GPIOs and an expansion bus that can be connected to an FPGA.

For more Technical specifications visit: `https://shakti.org.in/tapeouts`

1.2.3 Aardonyx

Coming soon....

1.3 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.

1.3.1 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. We provide the firmware support for end users to develop application. The SHAKTI-SDK is simple and easily customizable. Some of the essential features like DEBUG codes and board support libraries are provided.

1.3.2 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://github.com/platformio/platform-shakti

1.3.3 Arduino IDE

Arduino IDE is an open-source software used for electronic prototyping, helping user create interactive projects. We have added SHAKTI board support. The aim of this IDE is to make application development over SHAKTI as close as possible to Arduino.

1.3.4 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 standard operating systems proven to work on SHAKTI. Linux porting is available at:
https://gitlab.com/shaktiproject/software/shakti-linux

SECTION 2

Board Details

We aim to provide SHAKTI support on different types of development boards. As part of this effort, initially we are supporting two varieties of FPGA boards. They are Xilinx's Arty7 35T and Arty7 100T respectively. Below sections, list the detail information on the boards and purchase of boards.

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. E-arty35T[14]
   - *E-arty*35*T* is a SoC based on SHAKTI E class [6].
   - *E-arty*35*T* is supported on Artix 7 35T board.
   - It has an abridged version of 32 bit E class. It includes I, M, A and C.
2. C-arty100T[15]
   - *C-arty*100*T* is a SoC based on SHAKTI C class [5].
   - *C-arty*100*T* is supported on Artix 7 100T board.
   - It has an abridged version of 64 bit C class. It includes I, M, A, F, D and C.

2.1.1 Board Availability

The boards can be bought from,

1. Digilent
   `https://store.digilentinc.com/arty-a7-artix-7-fpga-development-board-for-maker`
2. Amazon
   `https://www.amazon.in/Digilent-Artix-7-Development-Makers-Hobbyists/dp/B017BOBNEO?tag=googinhydr18418-21`.

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`

SECTION 3

Board setup

The earlier section listed out the development boards supported by SHAKTI. This section, discusses the procedure to set up the board for application development. Topics include connecting a debugger, installing Vivado, building the SHAKTI C-class/E-class bit stream, programming the on-board configuration memory and running example programs. Before running example programs, we need to make sure the board is up and running. Broadly, the following steps needs to be followed, 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 and the other end to the MicroUSB connector (J10) in the board. This will power ON the board. The connector J10 is a JTAG & 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 will be discussed later. The debugger for the board is the Xilinx FTDI chip on the Arty boards. The details 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 UART via FTDI interface which provides debugger support.

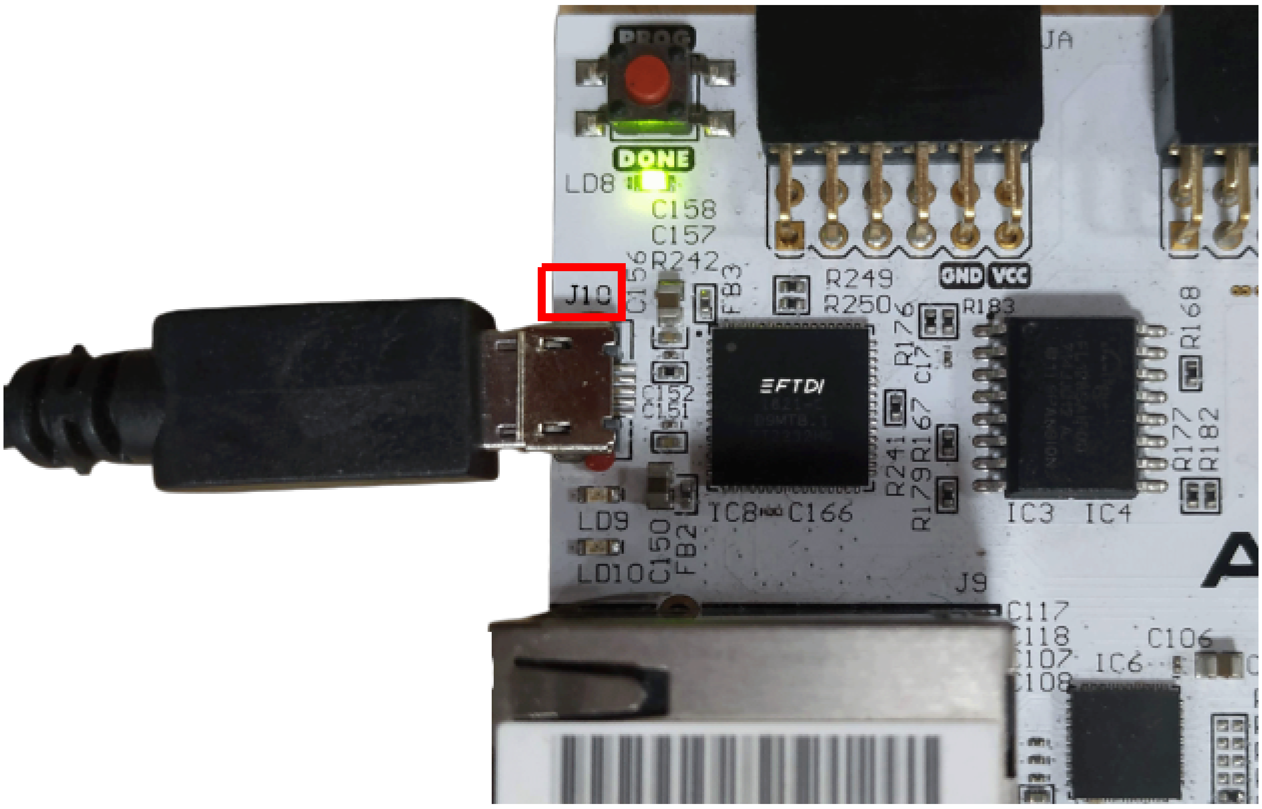


Figure 3: FTDI connection

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 realize SHAKTI on Xilinx development boards, the relevant Register Transfer Level (RTL) design has to be programmed on to the FPGA. And over that, the applications can be run. The procedure to do the same is listed below.

3.3.1 Prerequisites

Ensure that Ubuntu 16.04 or above is used. In this machine, the following list of software packages has to be installed.

A. Bluespec Compiler.

B. Device Tree Compiler.

C. Vivado 2017 or above.

D. Miniterm.

E. OpenOCD.

Before starting the Board has to be connected to the PC. The following links host the RTL design. The next few sections takes you to generating a RTL bisttream and programming it to FPGA.

- E class on Artix7 35T.
  `https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-35t/e-class`

- C class on Artix7 100T.
  `https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-100t/c-class`

3.3.2 Tool Installation

A. Installation of Bluespec Compiler [1]

An open source version of the Bluespec Compiler is available online. By installing the open-source Bluespec compiler, one will be able to generate the synthesizable verilog compatible for FPGA targets.

- Open a new terminal and move to the Home folder. Copy paste the below commands in the terminal and press enter.

```
sudo apt install ghc libghc-regex-compat-dev libghc-syb-dev iverilog
```

[1] The latest compiler has been tested and known to work for Ubuntu 18.04. Also a binary built on 16.04 will not work on 18.04 due to libgc version mismatch. It is suggested you do a fresh install for 16.04.

```
sudo apt install libghc-old-time-dev libfontconfig1-dev libx11-dev
```

```
sudo apt install libghc-split-dev libxft-dev flex bison libxft-dev
```

```
sudo apt install tcl-dev tk-dev libfontconfig1-dev libx11-dev gperf
```

```
sudo apt install itcl3-dev itk3-dev autoconf git
```

- Download the repository

  ```
  git clone --recursive https://github.com/B-Lang-org/bsc
  cd bsc
  make PREFIX=/path/to/installation/folder
  ```

- After you have done the above steps, add the path you have installed the bsc compiler to your $PATH in the ḃashrc or ċshrc

  ```
  export PATH=$PATH:/path/to/installation/folder/bin
  ```

- Typing bsc in your terminal should display the help options

  ```
  bsc -help
  ```

B. Install DTC (device tree compiler)

We use the DTC 1.4.7 to generate the device tree map in the $bootfiles$. Please be in the Home folder and run the below commands:

```
sudo wget https://git.kernel.org/pub/scm/utils/dtc/dtc.git/snapshot/dtc-1.4.7.tar.gz
sudo tar -xvzf dtc-1.4.7.tar.gz
cd dtc-1.4.7/
sudo make NO_PYTHON=1 PREFIX=/usr/
sudo make install NO_PYTHON=1 PREFIX=/usr/
```

C. Installing Vivado HLx 2018.3

1. If you dont have a Xilinx account, create a free account, using below url
   `https://www.xilinx.com/registration/create-account.html`

2. Download the Vivado HLx 2018.3 Linux Self Extracting Web Installer, by clicking on the below link
   `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 fine. 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.
```

15. Download the board files and Copy it to 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
```

D. Installation of Miniterm

[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

```
sudo apt-get install python-serial
```

E. Installation of RISC-V OpenOCD

- Clone the RISC-V OpenOCD repository

```
git clone https://github.com/riscv/riscv-openocd.git
cd riscv-openocd
./bootstrap
```

- Building the OpenOCD toolchain.

```
export RISCV=/path/to/install/riscv/OpenOCD
./configure --prefix=$RISCV --enable-remote-bitbang --enable-ftdi
--enable-jlink --enable-jtag_vpi --disable-werror
make
sudo make install
export PATH=$PATH:$RISCV/bin
```

3.3.3 Programming E-arty35t RTL bitstream onto the FPGA

- Connect the board to the PC and move to the HOME directory.

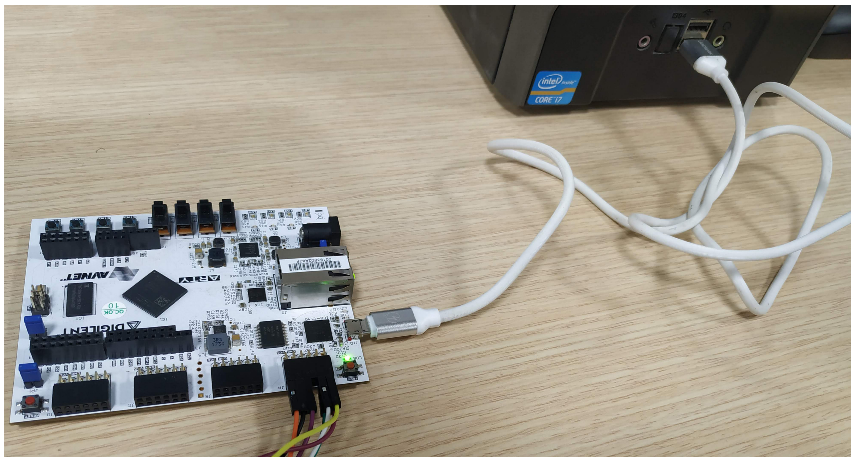


Figure 4: BOARD – PC

- Clone the shakti-soc repository to PC.

```
git clone https://gitlab.com/shaktiproject/cores/shakti-soc.git
cd shakti-soc/fpga/boards/artya7-35t/e-class
```

- Program the FPGA.[4]

```
make quick_build_xilinx
```

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

- Run OpenOCD command.[5]

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

[4]To rerun "make quick_build_xilinx", delete the folder shakti-soc/fpga/boards/artya7-35t/e-class/fpga_project/e-class and try.

[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

3.3.4 Programming C-arty100t RTL bitstream onto the FPGA

- Connect the board to the PC and move to the HOME directory.
- Clone the shakti-soc repository to PC.

```
git clone https://gitlab.com/shaktiproject/cores/shakti-soc.git
cd shakti-soc/fpga/boards/artya7-100t/c-class
sed -i 's/BSCAN2E=.*/BSCAN2E=enable/g'  core_config.inc
```

- Program the FPGA[6] .

```
make quick_build_xilinx
```

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

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

[6]To rerun "make quick_build_xilinx", delete the folder shakti-soc/fpga/boards/artya7-35t/e-class/fpga_project/c-class and try

4

SECTION

SoC Device Information

The SHAKTI based SoC's are built 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. SHAKTI E and C class use a Common AXI extension bus to realize this memory mapped I/O. The next two sections deal with the list of devices and the memory map of the devices for E-arty35T and C-arty100T.

Sl. No	Device name	Abbreviation
1	GPIO	General Purpose Input Output
2	UART	Universal Asynchronous Receiver Transmitter
3	I2C	Inter-Integrated Circuit
4	SPI	Serial Peripheral Interface
5	PWM	Pulse Width Modulation
6	PLIC	Platform Level Interrupt Controller

Sl. No	Device name	Abbreviation
7	CLIC	Core Level Interrupt Controller
8	ADC	Analog Digital Converter
9	SDRAM	Synchronous Dynamic Random Access Memory
10	BRAM	Block Random Access Memory
11	DDR	Double Data Rate

Table 2: 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 E-arty35T memory map

Sl.No	Peripheral	Base Address Start	Base Address End
1.	Memory (TCM)	0x80000000	0x8001FFFF
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.	CLINT	0x02000000	0x020BFFFF
14.	GPIO	0x00040100	0x000401FF
15.	PLIC	0x0C000000	0x0C01001F
16.	I2C	0x00040000	0x000400FF
17.	XADC	0x00041000	0x00041400

Table 3: E-arty35T class memory map

4.1.2 C-arty100T memory map

Sl. No	Peripheral	Base Address Start	Base Address End
1.	Memory (DDR)	0x80000000	0x87FFFFFF
2.	Debug	0x00000000	0x0000000F
3.	UART0	0x00011300	0x00011340
5.	I2C	0x020C0000	0x020C00FF
4.	GPIO	0x020D0000	0x020D00FF
6.	CLINT	0x02000000	0x020BFFFF
7.	PLIC	0x0C000000	0x0C020000

Table 4: C-arty100T memory map

4.1.3 Aardonyx memory map

Sl.No	Peripheral	Base Address Start	Base Address End
1.	SDRAM address	0x80000000	0x8FFFFFFF
2.	SDRAM CFG	0x00000200	0x000002FF
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
8.	PWM 5	0x00030500	0x000305FF
9.	SPI 0	0x00020000	0x000200FF
10.	SPI 1	0x00020100	0x000201FF
11.	SPI 2	0x00020200	0x000202FF

Sl.No	Peripheral	Base Address Start	Base Address End
12.	UART 0	0x00011300	0x00011340
13.	UART 1	0x00011400	0x00011440
14.	UART 2	0x00011500	0x00011540
15.	CLINT	0x02000000	0x020BFFFF
16.	I2C 0	0x00040000	0x000400FF
17.	GPIO	0x00040100	0x000401FF
18.	Bootrom	0x00040200	0x000402FF
19.	pin mux	0x00040300	0x000403FF
20.	I2C 1	0x00040400	0x000404FF
21.	PLIC	0x0C000000	0x0C01001F
22.	QSPI0	0x00000100	0x000001FF
23.	QSPI0 Mem	0x90000000	0x9FFFFFFF

Table 5: Aardonyx memory map

5

SECTION

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.

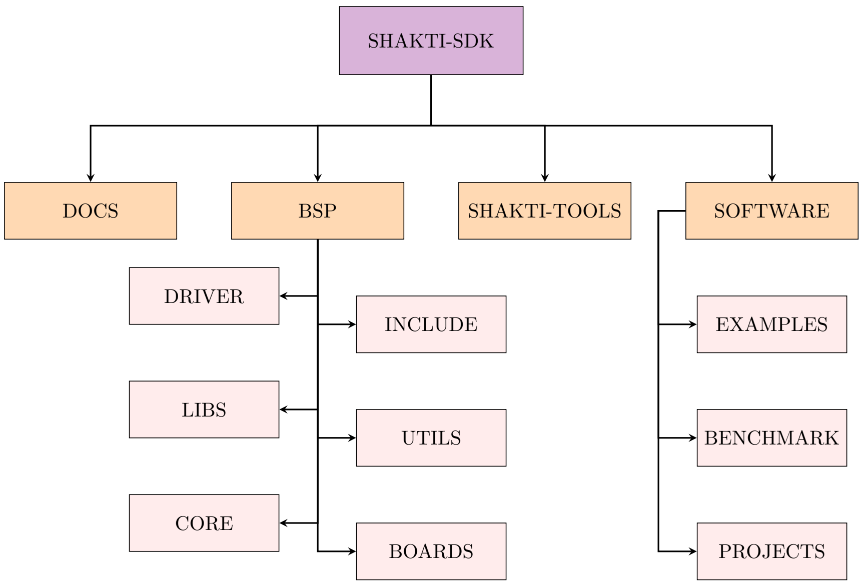


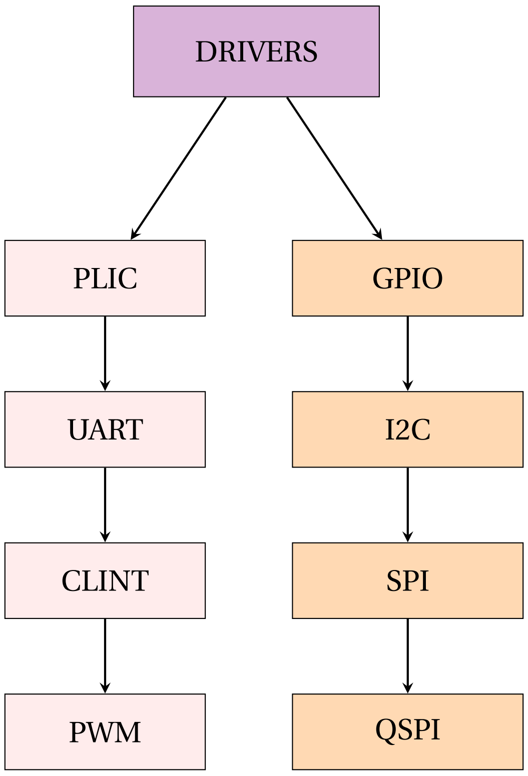
Figure 5: SDK architecture

5.1.1 Board Support Package

The BSP consists of driver files for various devices and system files. 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 folder has header files for core and each driver. The board independent variable/macro definitions and declarations pertaining to each driver is included here.

3. libs

   The library utilities, 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 codes 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 folder provides support for external boards as well as custom boards.Includes the definitions of board specific functions such as console drivers.

5.1.2 shakti-tools

SHAKTI is a RISC-V based architecture. It uses the RISC-V toolchain to develop software. The shakti-tools folder has "ready to use" RISC-V tools. It has a RISC-V GNU tool chain, RISC-V instruction set simulator, OpenOCD (debugger) and RISC-V spike simulator. These tools can be downloaded, along with the SHAKTI-SDK. Please check section 5.2.4 for further details.

5.1.3 Software

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

1. Projects

   This folder 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.

5.1.4 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= artix7_35t or artix7_100t or aardonyx

  Default TARGET is artix7_35t

- make debug PROGRAM=? TARGET=?

  PROGRAM can be found from "make list_applns"

  TARGET= artix7_35t or artix7_100t or aardonyx

  Default TARGET is artix7_35t

  debug command adds the debug support to applns.

- make all

  TARGET= artix7_35t

  All the applications under example folder 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.

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 paste the below command 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 64bit and 32bit 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 below command 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 for SHAKTI-SDK.

- Automatic method (Recommended)

    - The SHAKTI-SDK repository provides the board support packagers and an environment in which the applications can be developed.
    - The Tool-chain executables are hosted in the SHAKTI-TOOLS repository. (shakti-tools) [16].
    - 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,

Assuming, one is in SHAKTI-TOOLS repository. Copy, paste the following commands in the same terminal. This will essentially set the $PATH variable to the exact tool-chain path for that **particular session**.

```
SHAKTITOOLS=/path/to/shakti-tools
export PATH=$PATH:$SHAKTITOOLS/bin
export PATH=$PATH:$SHAKTITOOLS/riscv64/bin
export PATH=$PATH:$SHAKTITOOLS/riscv64/riscv64-unknown-elf/bin
export PATH=$PATH:$SHAKTITOOLS/riscv32/bin
export PATH=$PATH:$SHAKTITOOLS/riscv32/riscv32-unknown-elf/bin
```

Things to know:

1. The availability of the tool chain across the file system is ensured by the **export** command.
2. The above commands will export both **64 bit and 32 bit tool-chains**. If only one of the tool-chains is required, it can exported separately.
3. Please add the above lines in **.bashrc** file in the home folder to set the PATH **permanently** instead of that particular session.
4. The variable $SHAKTITOOLS has the location of SHAKTI-TOOLS in the file system.
5. The command **which riscv64-unknown-elf-gcc** helps you to verify whether tool-chain path exported correctly.

Steps:

Automatic method

```
$ pwd
/home/user
$ git clone https://gitlab.com/shaktiproject/software/shakti-sdk.git
$ git clone --recursive https://gitlab.com/shaktiproject/software/
shakti-tools.git
$ cd shakti-tools
$ pwd
/home/user/shakti-tools
$ SHAKTITOOLS=/path/to/shakti-tools
$ export PATH=$PATH:$SHAKTITOOLS/bin:$SHAKTITOOLS/riscv64/bin
$ export PATH=$PATH::$SHAKTITOOLS/riscv32/bin
$ export PATH=$PATH:$SHAKTITOOLS/riscv64/riscv64-unknown-elf/bin
$ export PATH=$PATH:$SHAKTITOOLS/riscv32/riscv32-unknown-elf/bin
$ which riscv64-unknown-elf-gcc
/home/user/shakti-tools/riscv64/bin/riscv64-unknown-elf-gcc
```

Add export commands to .bashrc, save and close the file.

5.2.5 Update the SDK or TOOLS

To update your SDK or TOOLS repository to the latest version, move to the respective repository **(cd shakti-tools or cd shakti-sdk)** and please 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. We are listing the steps to develop a small application using SHAKTI-SDK. 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.
- Download and install SHAKTI-SDK.
- Install the RISC-V tool chain.
- set PATH variable for 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* folder.
- 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 folder for the new application and name it accordingly.
- Inside the folder, 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 folder 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 folder 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' folder. Use existing programs in example folder for reference.
- Make a new entry for the program in the 'existing Makefile' under examples folder.

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 TARGET=artix7_35t
```

Interpreting above commands:

- PROGRAM is the new one created. It is listed by typing "make list_applns".
- TARGET= artix7_35t or artix7_100t or aardonyx.
- Default TARGET is artix7_35t.

5.3.4 Run

Once the application is built, the executable is generated in the output folder. The executable is in ELF file format and they have the extension *.shakti*. There are two modes to run an application on the arty 35t board. 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 debug applications easily.

The Arty35T 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 35T 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 python-serial
   ```
2. Open three terminals, one for each of the following
   a. One terminal for UART terminal display.
   b. Another for GDB server.
   c. And the last one for OpenOCD.

Follow the steps below to set up and run programs[7]

1. In the first terminal, open a serial port to display output from UART.
   ```
   $ sudo miniterm.py /dev/ttyUSB0 19200
   ```

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

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

   For example,

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

   Press reset in the board and run the following commands.

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

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

   ```
   $ riscv32-unknown-elf-gdb
   (gdb) set remotetimeout unlimited
   (gdb) target remote localhost:3333
   (gdb) file path/to/executable
   (gdb) load
   (gdb) c
   ```

Note:

1. "/dev/ttyUSB0" - ttyUSB means "USB serial port adapter" and the "0" ("0" or "1" or whatever) is the USB device number.

2. For C class (64 bit) applications, please use riscv64-unknown-elf-gdb instead of riscv32-unknown-elf-gdb.

3. The **default** baudrate is **19200** but if the target is **AARDONYX** the baud rate is **7668.**

4. While using **Aardonyx** as a Target the corresponding pheriperal PIN MUX Register Value has to be included in the application program.
   For Example ,if one want to run applications of the PWM,in **Aardonyx** and want to make use of all GPIO pins as PWM pins initialize Pin Mux Register with 0x0002AA80

5. The values of the Pin MUX register for different peripherals is mentioned under Aardonyx in Device mapping section of the user manual.

5.4.2 Application flow

5.5 Running application in Standalone mode

Until now, we have been running Arty-35T board in a Debug mode. We need a Host PC to build and run the application every time. In stand alone mode, the Arty-35t board when

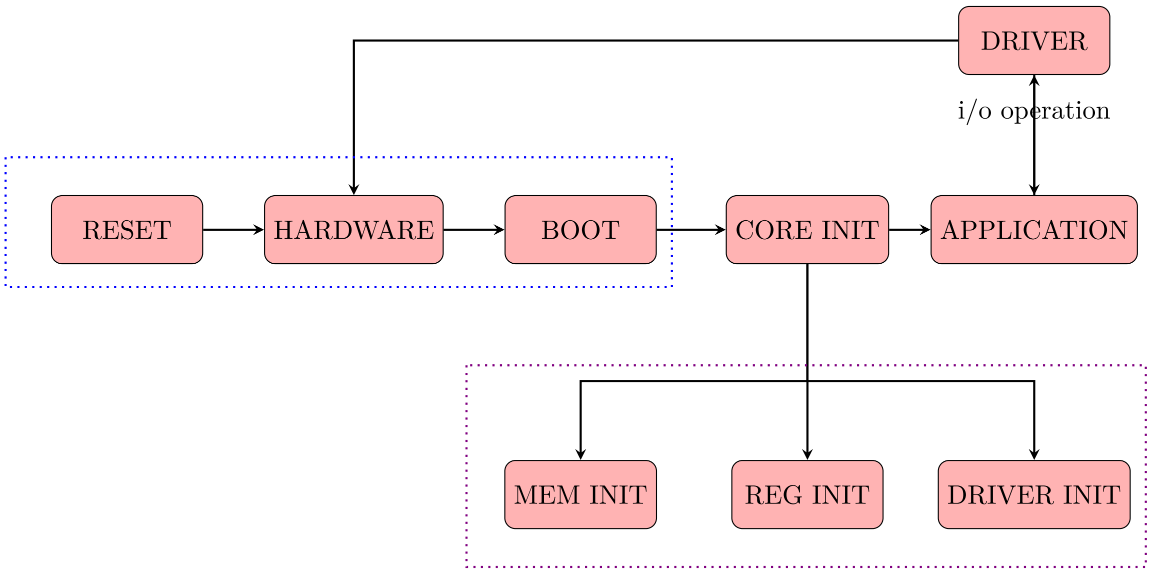


Figure 6: Execution flow, after every reset

booted starts executing the code autonomously. 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 for 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=artix7_35t
```

Interpreting above commands:

- PROGRAM is the new bare metal user application that is created. It is listed by typing "`make list_applns`".
- TARGET= artix7_35t, refers to the board.[8]

[8] Currently SPI boot is available only in E-arty35T.

SECTION A

Device pin mapping

A.1 C-arty100T

Sl. No	Pin Description	Pin mapping	Peripheral
1.1	GPIO0	CKIO0 (J4[1],IO - Lower)	GPIO
1.2	GPIO1	CKIO1 (J4[3],IO - Lower)	
1.3	GPIO2	CIIO2 (J4[5],IO - Lower)	
1.4	GPIO3	CKIO3 (J4[7],IO - Lower)	
1.5	GPIO4	CKIO4 (J4[9],IO - Lower)	
1.6	GPIO5	CKIO5 (J4[11],IO - Lower)	

Sl. No	Pin Description	Pin mapping	Peripheral
1.7	GPIO6	CKIO6 (J4[13],IO - Lower)	
1.8	GPIO7	CKIO7 (J4[15],IO - Lower)	
1.9	GPIO8	CKIO8 (J2[1],IO - Higher)	
1.10	GPIO9	CKIO9 (J2[3],IO - Higher)	
1.11	GPIO10	CKIO10 (J2[5],IO - Higher)	
1.12	GPIO11	CKIO11 (J2[7],IO - Higher)	
1.13	GPIO12	CKIO12 (J2[9],IO - Higher)	
1.14	GPIO13	CKIO13 (J2[11],IO - Higher)	
1.15	GPIO14	CKIO26 (J4[2],IO - Lower)	
1.16	GPIO15	CKIO27 (J4[4],IO - Lower)	
2.1	SDA	CK_SDA (J3[1])	I2C
2.2	SCL	CK_SCL (J3[2])	
3.1	UART0 TX	J10	UART
3.2	UART0 RX	J10	

Sl. No	Pin Description	Pin mapping	Peripheral
4.1	INTERRUPT 0	CKIO28 (J4[6],IO - Lower)	PLIC
4.2	INTERRUPT 1	CKIO29 (J4[8],IO - Lower)	
4.3	INTERRUPT 2	CKIO30 (J4[10],IO - Lower)	
4.4	INTERRUPT 3	CKIO31 (J4[12],IO - Lower)	
4.5	INTERRUPT 4	CKIO32 (J4[14],IO - Lower)	
4.6	INTERRUPT 5	CKIO33 (J4[16],IO - Lower)	
4.7	INTERRUPT 6	CKIO34 (J2[2],IO - Lower)	
4.8	INTERRUPT 7	CKIO35 (J2[4],IO - Lower)	

Table 6: C-arty100T Device pin map

A.2 E-arty35T

Sl. No	Pin Description	Pin mapping	Peripheral
1.1	GPIO0	CKIO0 (J4[1],IO - Lower	GPIO
1.2	GPIO1	CKIO1 (J4[3],IO - Lower)	
1.3	GPIO2	CIIO2 (J4[5],IO - Lower)	
1.4	GPIO3	CKIO3 (J4[7],IO - Lower)	
1.5	GPIO4	CKIO4 (J4[9],IO - Lower)	
1.6	GPIO5	CKIO5 (J4[11],IO - Lower)	
1.7	GPIO6	CKIO6 (J4[13],IO - Lower)	
1.8	GPIO7	CKIO7 (J4[15],IO - Lower)	
1.9	GPIO8	CKIO8 (J2[1],IO - Higher)	
1.10	GPIO9	CKIO9 (J2[3],IO - Higher)	
1.11	GPIO10	CKIO10 (J2[5],IO - Higher)	
1.12	GPIO11	CKIO11 (J2[7],IO - Higher)	
1.13	GPIO12	CKIO12 (J2[9],IO - Higher)	

Sl. No	Pin Description	Pin mapping	Peripheral
1.14	GPIO13	CKIO13 (J2[11],IO - Higher)	
1.15	GPIO14	CKIO26 (J4[2],IO - Lower)	
1.16	GPIO15	CKIO27 (J4[4],IO - Lower)	
2.1	SDA	CK_SDA(J3[1])	I2C
2.2	SCL	CK_SCL (J3[2])	
3.1	UART0 TX	J10	UART
3.2	UART0 RX	J10	
4.1	UART1 TX	JC[7] - 3P	
4.2	UART1 RX	JC[8] - 3N	
5.1	PWM 0	JD[1]	PWM PINS
5.2	PWM 1	JD[2]	
5.3	PWM 2	JD[3]	
5.4	PWM 3	JD[4]	
5.5	PWM 4	JD[7]	
5.6	PWM 5	JD[8]	

Sl. No	Pin Description	Pin mapping	Peripheral
6.1	SPI0 CS	JB[1] - 1P	SPI0
6.2	SPI0 SCLK	JB[2] - 1N	
6.3	SPI0 MISO	JB[3] - 2P	
6.4	SPI0 MOSI	JB[4] - 2N	
7.1	SPI1 CS	JB[7] - 3P	SPI1
7.2	SPI1 SCLK	JB[8] - 3N	
7.3	SPI1 MISO	JB[9] - 4P	
7.4	SPI1 MOSI	JB[10] - 4N	
8.1	ADC 4	CKA0	Single ended ADC
8.2	ADC 5	CK A1	
8.3	ADC 6	CK A2	
8.4	ADC 7	CK A3	
8.5	ADC 15	CK A4	

Sl. No	Pin Description	Pin mapping	Peripheral
8.6	ADC 0	CK A5	
9.1	ADC 12P	CK A6	Double ended ADC
9.2	ADC 12N	CK A7	
10.1	ADC 13P	CK A8	Double ended ADC
10.2	ADC 13N	CK A9	
11.1	ADC 14P	CK A10	Double ended ADC
11.2	ADC 14N	CK A11	

Table 7: E-arty35T Device pin map

A.3 Aardonyx

Sl. No	Pin Description	Pin mapping	Peripheral
1.1	TEST_MODE	SW[0]	SWITCHES
1.2	BOOT_MODE[0]	SW[1]	
1.3	BOOT_MODE[1]	SW[2]	
2.1	PIN_TMS	JA[1]	JTAG
2.2	PIN_TDI	JA[2]	
2.3	PIN_TRST	JA[4]	
2.4	PIN_TDO	JA[7]	
2.5	PIN_TCK	JA[8]	
3.0	I2C0_SDA	JB[1]	I2C0
3.1	I2C0_SCL	JB[2]	
4.1	GPIO_14	JB[3]	GPIO
4.2	GPIO_15	JB[4]	
5.1	SPI1_NSS	JC[1]	SPI1

Sl. No	Pin Description	Pin mapping	Peripheral
5.2	SPI1_MOSI	JC[2]	
5.3	SPI1_MISO	JC[3]	
5.4	SPI1_SCLK	JC[7]	
6.1	QSPI_NCS	JD[1]	QSPI
6.2	QSPI_IO[0]	JD[2]	
6.3	QSPI_IO[1]	JD[3]	
6.4	QSPI_CLK	JD[4]	
6.5	QSPI_IO[2]	JD[9]	
6.6	QSPI_IO[3]	JD[10]	
7.1	UART0_SOUT	UART_RXD_OUT	USB
7.2	UART0_SIN	UART_TXD_IN	
8.1	SPI0_SCLK	A[0]	SPI0
8.2	SPI0_NSS	A[1]	

[9]

Sl. No	Pin Description	Pin mapping	Peripheral
8.3	SPI0_MOSI	A[2]	
8.4	SPI0_MISO	A[3]	
9.1	I2C1_SCL	SCL	I2C1
9.2	I2C1_SDA	SDA	
10.1	IO[0]*	GPIO_0 / UART1_RX	IO MUX
10.2	IO[1]*	GPIO_1 / UART1_TX	
10.3	IO[2]*	GPIO_2 / UART2_RX	IO MUX
10.4	IO[3]*	GPIO_3/ UART2_TX / PWM_0	
10.5	IO[4]	GPIO_4	
10.6	IO[5]*	GPIO_5/ PWM_1	
10.7	IO[6]*	GPIO_6/PWM_2	
10.8	IO[7]	GPIO_7	
10.9	IO[8]	GPIO_8	

[9]* denotes Pinmux pins

[10]

Sl. No	Pin Description	Pin mapping	Peripheral
10.10	IO[9]*	GPIO_9 /PWM_3	
10.11	IO[10]*	GPIO_10/ SPI2_NCS / PWM_4	
10.12	GPIO_11	SPI2_MOSI	
10.13	GPIO_12	SPI2_MISO	
10.14	GPIO_13	SPI2_CLK	

Table 8: Aardonyx Device pin map

[10]* denotes Pinmux pins

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	-		-		-		-		-		S2CK		S2SO		S2SI	
10	-		-		-		-		-		-		-		PWM5	

Table 9: Aardonyx 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	S2_CS		-		-		-[11]		U2TX		U2RX		U1TX		U1RX	
10	PWM4		PWM3		PWM2		PWM1		PWM0		-		-		-	

Table 10: Aardonyx pinmux memory map (lower bytes)

Note:

1. The above tables are used for configuring GPIO pins as I2C, PWM, UART & SPI.
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 as described in section A.3.

SECTION C

Platform IO for SHAKTI

PlatformIO IDE is the next-generation integrated development environment for IoT [17].

C.1 Prerequisites

- Operating System :
  - linux : Ubuntu 16.04 or later
  - Windows : Windows 10
- Python Interpreter: Python 2.7 or Python 3.5+.
- Access to Serial Ports (USB/UART):
  - Windows Users: Please check that you have correctly installed USB driver from board manufacturer
  - Linux Users : Please install 99-platformio-udev.rules
- Software : Visual Studio Code 2018-2019
- FPGA Board : Pio currently supports the below boards. [12]
  - Artix 7 100t
  - Artix 7 35t

[12] Refer section 2 for board details

C.2 Installation

- Getting Platform Io for Visual Studio
  - Download The Visual Studio from the following link : https://code.visualstudio.com/?wt.mc_id=DX_841432
  - Once you have downloaded the visual studio , go to the extensions tab , in the search bar search for Platform IO IDE and install it .
  - Restart the visual studio once you have installed the Platform IO extension.
  - Open the platforms tab and install the platform IO framework using the following two methods
    - Use the Advanced Installation by Clicking the Advanced Installation Button -> Paste the link https://github.com/platformio/platform-shakti to install the corresponding platform.
    - Use the Command Line for installing the respective platform by typing the following in the terminal pio platform install https://github.com/platformio/platform shakti
  - Once the the above steps are complete restart the viisual studio to refresh the IDE

C.3 Using SHAKTI SDK in PlatformIO IDE

The shakti software development kit is a ready to use kit where one can develop software's and projects on the respective boards available. The SHAKTI-SDK is embedded into the PlatformIO IDE, along with risc-v tools. There by the application development process is integrated into the PlatformIO IDE [18].

C.3.1 Getting Started with your first application

To have a clear understanding of how the software is built and deployed one can go through a quick example by selecting Project Examples and follow the following steps

- Select the hello application from the drop down and press import.
- Once the project is imported , you can start building the project by going to project tasks (Alien icon on left side)->Build
- To connect the board with IDE follow the instructions
- Once you have connected the board you can proceed by Clicking on Upload and Monitor on the Project task column, which is followed by a pop up terminal displaying the monitor rate also it uploads the program automatically.
- To execute the corresponding program to the board go to Debug ->Start Debugging or press F5 . This should insert a break point automatically and execute.

- The above two steps should upload the program and open the terminal displaying the output "Hello World"

C.3.2 Creating a project from scratch

1. Start creating a project by doing the following
   - New Project -> Enter Details in the boxes as below
     - Project Name : Give any name of your choice
     - Board : Choose appropriate board of your choice from drop down as below
       * Arty7 100t
       * Arty7 35t
     - Framework : By default it will choose SHAKTI-SDK , if not select SHAKTI-SDK from dropdown
   - Click Finish
2. Creating your first application When creating your first application one should know where to put the files , hence please refer the below structure
   - Once you have created appropriate files according to the above steps , Build the project and acquire a elf output file so that you can upload to the board and test.
   - Use the "Upload and Monitor " Command copy the elf to the board and execute.

5

SECTION

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], organization="Linux Foundation"
https://www.zephyrproject.org/what-is-zephyr/

[11] Program SHAKTI on Arty A7-35T
https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-35t/e-class/pre-built-mcs

[12] Program SHAKTI on Arty A7-100T
https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-100t/c-class/pre-built-mcs

[13] 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

[14] SHAKTI E class SoC on Artix 35T
https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-35t/e-class

[15] SHAKTI C class SoC in Artix 100T
https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-100t/c-class

[16] Generated RISC V tool chain
https://gitlab.com/shaktiproject/software/shakti-tools

[17] PlatformIO extensions for VSCODE
https://platformio.org/

[18] SHAKTI support on PlatformIO
https://github.com/platformio/platform-shakti

[19] SHAKTI support on FreeRTOS
https://gitlab.com/shaktiproject/software/FreeRTOS