Automate Stack 2.0

Reference Design

FPGA-RD-02255-1.0

June 2022

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Contents

Acronyms in This Document 11
1. Introduction 12
1.1. Components 13
2. Design Overview 14
2.1. Theory of Operation 14
2.2. FPGA Design 15
2.2.1. Main System 15
2.2.2. Node System....18
2.3. EtherControl IP 20
2.3.1. Features....21
2.3.2. EtherControl Master....22
2.3.3. Register Description 23
2.3.4. EtherControl Slave....42
2.4. RISC-V to PCIe Bridge 45
2.5. FIFO DMA 48
2.6. PCIe DMA IP Design Details....51
2.6.1. Descriptor Field Format....52
2.6.2. Status Field Format....53
2.6.3. Triggering the DMA Operation 53
2.6.4. PCIe DMA Register Space 54
2.7. SPI Flash Controller (QSPI Streamer)....56
2.8. CNN Co-Processor Unit (CCU) 56
2.9. Motor Control and PDM Data Collector....59
2.10. SPI Master IP Design Details....68
2.10.1. Overview....68
2.10.2. SPI Master Register Map 69
2.10.3. Programming Flow 69
2.11. I²C Master IP Design Details 70
2.11.1. Overview....71
2.11.2. I²C Master Register Map....71
2.11.3. Programming Flow 72
2.12. UART IP Design Details 73
2.12.1. Overview....74
2.12.2. Programming Flow 75
3. Resource Utilization....77
4. Software APIs....78
4.1. Main System APIs....78
4.1.1. Tasks of the Main System....78
4.1.2. Key Functions 79
4.2. Node System APIs....84
4.2.1. Tasks of the Node System 84
4.2.2. Key Functions 84
4.3. PCie Driver....86
4.3.1. Linux Device Driver Design 86
4.3.2. User-Space to Kernel-Space Access....86
4.3.3. File Operation and API Description 87
4.3.4. PCIeProbe 88
4.3.5. PCIeRemove....88
4.3.6. Bus Master DMA Overview and Implementation....89
4.4. Programming the DMA Write/Read....90
4.4.1. Supported Operating System 90
4.4.2. Package Requirements 91

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.4.3. Installing the Package 92
4.4.4. Manual Installation and Setup....93
4.4.5. Automatic Installation and Setup 93
5. Communications 94
5.1. Communication between Host and Main System 94
5.1.1. Messages from Host to Main System 94
5.1.2. Messages from Main System to Host 94
5.2. Communication between Main System and Node System(s) 94
5.2.1. Messages from Main System to Node System 94
5.2.2. Messages from Node System to Main System 94
6. Demo Package Directory Structure....95
6.1. Automate Stack Demonstration 95
6.1.1. Documentation....95
7. Summary....96
Appendix A. Predictive Maintenance with TensorFlow Lite 97
A.1. Setting Up the Linux Environment for Neural Network Training 97
A.1.1. Installing the NVIDIA CUDA and cuDNN Library for Machine Learning Training on GPU 97
A.1.2. Setting Up the Environment for Training and Model Freezing Scripts 99
A.1.3. Installing the TensorFlow version 1.15 100
A.1.4. Installing the Python Package....101
A.2. Creating the TensorFlow Lite Conversion Environment....102
A.3. Preparing the Dataset 102
A.3.1. Dataset Information 103
A.4. Preparing the Training Code....103
A.4.1. Training Code Structure....103
A.4.2. Generating tfrecords from Augmented Dataset 104
A.4.3 Neural Network Architecture 104
A.4.4. Training Code Overview 106
A.4.5. Training from Scratch and/or Transfer Learning 114
A.5. Creating Frozen File....116
A.5.1. Generating .pbtxt File for Inference 116
A.5.2. Generating the Frozen (.pb) File....116
A.6. TensorFlow Lite Conversion and Evaluation 117
A.6.1. Converting Frozen Model to TensorFlow Lite 117
A.6.2. Evaluating TensorFlow Lite model....118
A.6.3. Converting TensorFlow Lite To C-Array....118
Appendix B. Setting up the Auto-Bootable MQTT-Based Client....119
B.1. Unzipping the Folder....119
B.2. OpenSSL Error 119
B.3. Making the New Server Executable....119
B.4. Installing the Mosquitto Broker....119
B.5. Automating the Application....120
B.6. Setting Up the IPV4 Address and Router on Raspberry Pi 120
Technical Support Assistance 122
Revision History 123

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Figures

Figure 1.1. Top Level Block Diagram of Automate Stack 2.0 12
Figure 2.1. Automate Stack 2.0 Architecture 14
Figure 2.2. Main System Architecture 16
Figure 2.3. Node System Architecture 19
Figure 2.4. EtherControl Block Diagram 20
Figure 2.5. EtherControl Master Block Diagram 22
Figure 2.6. EtherControl Slave 42
Figure 2.7. Top Level Architecture of PCIe DMA IP Design 51
Figure 2.8. FPGA Device Memory Segregation 54
Figure 2.9. Motor Controller Interface with Motor 59
Figure 2.10. SPI Master IP Core Block Diagram 68
Figure 2.11. I^2C Master IP Core Functional Diagram 71
Figure 2.12. UART IP Core Functional Block Diagram 74
Figure 2.13. UART Data Format 76
Figure 4.1. Main Function 80
Figure 4.2. User-Space and Kernel-Space Access Diagram 86
Figure 4.3. Top-level Block Diagram 89
Figure 4.4. BMD with Descriptor and Fixed Physical Memory in RAM 90
Figure 4.5. Make FIle 91
Figure 4.6. GCC Command 91
Figure 4.7. G++ Command 92
Figure 4.8. Kernel Version Command 92
Figure A.1. Download CUDA Repo 97
Figure A.2. Install CUDA Repo 97
Figure A.3. Fetch Keys 97
Figure A.4. Update Ubuntu Packages Repositories 98
Figure A.5. CUDA Installation 98
Figure A.6. cuDNN Library Installation 98
Figure A.7. Anaconda Installation 99
Figure A.8. Accept License Terms 99
Figure A.9. Confirm/Edit Installation Location 99
Figure A.10. Launch/Initialize Anaconda Environment on Installation Completion 100
Figure A.11. Anaconda Environment Activation 100
Figure A.12. TensorFlow Installation 100
Figure A.13. TensorFlow Installation Confirmation 100
Figure A.14. TensorFlow Installation Completion 101
Figure A.15. Easydict Installation 101
Figure A.16. Joblib Installation 101
Figure A.17. Keras Installation 101
Figure A.18. OpenCV Installation 102
Figure A.19. Pillow Installation 102
Figure A.20. Predictive Maintenance Dataset 103
Figure A.21. Training Code Directory Structure 103
Figure A.22. Training Code Flow Diagram 106
Figure A.23. Code Snippet: Hyper Parameters 107
Figure A.24. Code Snippet: Build Input 107
Figure A.25. Code Snippet: Parse tfrecords 108
Figure A.26. Code Snippet: Convert Image to Gray Scale 108
Figure A.27. Code Snippet: Convert Image to BGR and Scale the Image 108
Figure A.28. Code Snippet: Create Queue 108
Figure A.29. Code Snippet: Add Queue Runners 109
Figure A.30. Code Snippet: Create Model 109

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Figure A.31. Code Snippet: Fire Layer 109
Figure A.32. Code Snippet: Convolution Block 110
Figure A.33. Code Snippet: Feature Depth Array for Fire Layers 110
Figure A.34. Code Snippet: Forward Graph Fire Layers 111
Figure A.35. Code Snippet: Loss Function 111
Figure A.36. Code Snippet: Optimizers 111
Figure A.37. Code Snippet: Restore Checkpoints 112
Figure A.38. Code Snippet: Save .pbtxt 112
Figure A.39. Code Snippet: Training Loop 112
Figure A.40. Code Snippet: _ LearningRateSetterHook 113
Figure A.41. Code Snippet: Save Summary for Tensorboard 113
Figure A.42. Code Snippet: logging hook 113
Figure A.43. Predictive Maintenance – Run Script 114
Figure A.44. Predictive Maintenance – Trigger Training 114
Figure A.45. Predictive Maintenance – Trigger Training with Transfer Learning 114
Figure A.46. Predictive Maintenance – Training Logs 115
Figure A.47. Predictive Maintenance – Confusion Matrix 115
Figure A.48. TensorBoard – Launch 115
Figure A.49. TensorBoard – Link Default Output in Browser 115
Figure A.50. Checkpoint Storage Directory Structure 116
Figure A.51. Generated '.pbtxt' for Inference 116
Figure A.52. Run genpb.py To Generate Inference .pb 117
Figure A.53. Frozen Inference .pb Output 117
Figure B.1. IPV4 Address Setting 120
Figure B.2. Network Preferences Settings 120
Figure B.3. IP Configuration 121

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Tables

Table 2.1. Main System Memory Map (RISC-V) 17
Table 2.2. Main System Memory Map (PCIe) 17
Table 2.3. Node System Memory Map 19
Table 2.4. EtherControl Interfaces 20
Table 2.5. EtherControl Master Global Register Map (RISC-V) 23
Table 2.6. EtherControl Master Local Chain 1 Register Map (RISC-V) 23
Table 2.7. EtherControl Master Local Chain 2 Register Map (RISC-V) 23
Table 2.8. DMA FIFO Enable/AHBL Disable Register 24
Table 2.9. PHY Link Status Register 24
Table 2.10. Active Nodes Register 25
Table 2.11. FIFO Status Register for PDM Data 25
Table 2.12. Clear Interrupt Received Register 25
Table 2.13. Interrupt Polling Register 25
Table 2.14. Start Transaction in All Chains 26
Table 2.15. IP Busy Register 26
Table 2.16. AHBL_TOUT_R 26
Table 2.17. Chain 1 Start Transaction Register 26
Table 2.18. Chain 1 Packet Head Register 26
Table 2.19. Chain 1 Frame Number Register 27
Table 2.20. Chain 1 Number of Node Register 27
Table 2.21. Chain 1 Node Data Length Register 27
Table 2.22. Chain 1 Node Request Data Burst Register 27
Table 2.23. Chain 1 Node Request Type Register 27
Table 2.24. Chain 1 Node Address Register 28
Table 2.25. Chain 1 CRC Count Register 28
Table 2.26. Chain 1 Interrupt Info Register 28
Table 2.27. Chain 1 FIFO Status Register Request Data 28
Table 2.28. Chain 1 Node Motor Status Register 29
Table 2.29. Chain 1 Node Delay Register 29
Table 2.30. Chain 2 Start Transaction Register 29
Table 2.31. Chain 2 Packet Head Register 29
Table 2.32. Chain 2 Frame Number Register 29
Table 2.33. Chain 2 Number of Node Register 30
Table 2.34. Chain 2 Node Data Length Register 30
Table 2.35. Chain 2 Node Request Data Burst Register 30
Table 2.36. Chain 2 Node Request Type Register 30
Table 2.37. Chain 2 Node Address Register 30
Table 2.38. Chain 2 CRC Count Register 31
Table 2.39. Interrupt Info Register 31
Table 2.40. Chain 2 FIFO Status Register Request Data 31
Table 2.41. Chain 2 Node Motor Status Register 32
Table 2.42. Chain 2 Node Delay Register 32
Table 2.43. EtherControl Master Global Register Map (PCIe) 32
Table 2.44. EtherControl Master Local Chain 1 Register Map (PCIe) 32
Table 2.45. EtherControl Master Local Chain 2 Register Map (PCIe) 33
Table 2.46. DMA FIFO Enable/AHBL Disable Register 33
Table 2.47. PHY Link Status Register 33
Table 2.48. Active Nodes Register 34
Table 2.49. FIFO Status Register for PDM Data 34
Table 2.50. Interrupt Polling Register 34
Table 2.51. Clear Interrupt Received Register 34
Table 2.52. Start Transaction in All Chains 35

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.53. IP Busy Register 35
Table 2.54. AHBL Bus Timeout Count Register 35
Table 2.55. Node Response PDM Data Register 35
Table 2.56. Chain 1 Start Transaction Register 36
Table 2.57. Chain 1 Packet Head Register 36
Table 2.58. Chain 1 Frame Number Register 36
Table 2.59. Chain 1 Number of Node Register 36
Table 2.60. Chain 1 Node Data Length Register 36
Table 2.61. Chain 1 FIFO Status Register Request Data 37
Table 2.62. Chain 1 Node Request Type Register 37
Table 2.63. Chain 1 Node Address Register 37
Table 2.64. Chain 1 CRC Count Register....37
Table 2.65. Chain 1 Interrupt Info Register....37
Table 2.66. Chain 1 FIFO Status Register Request Data 38
Table 2.67. Chain 1 Node Request Burst Register 38
Table 2.68. Chain 1 Node Motor Status Register....38
Table 2.69. Chain 1 Node Delay Register 39
Table 2.70. Chain 2 Start Transaction Register 39
Table 2.71. Chain 2 Packet Head Register 39
Table 2.72. Chain 2 Frame Number Register 39
Table 2.73. Chain 2 Number of Node Register 39
Table 2.74. Chain 2 Node Data Length Register 40
Table 2.75. Chain 2 FIFO Status Register Request Data 40
Table 2.76. Chain 2 Node Request Type Register 40
Table 2.77. Chain 2 Node Address Register 40
Table 2.78. Chain 2 CRC Count Register....41
Table 2.79. Interrupt Info Register....41
Table 2.80. Chain 2 Node Request Burst Register 41
Table 2.81. Chain 2 Node Motor Status Register....41
Table 2.82. Chain 2 Node Delay Register 42
Table 2.83. EtherControl Slave Register Map 42
Table 2.84. DMA Control Register 43
Table 2.85. FIFO Data Register....43
Table 2.86. Motor Status Register 43
Table 2.87. DMA Done Indication Register....43
Table 2.88. Interrupt Status Register....43
Table 2.89. Motor Config/Status Address Register (or) PDM Data Transfer Size Register 44
Table 2.90. Motor Configuration Data Register....44
Table 2.91. FIFO Error Register 44
Table 2.92. Clear Interrupt Received Register 44
Table 2.93. RISC-V to PCIe Register Map 45
Table 2.94. RISC-V Config Register 1....45
Table 2.95. RISC-V Config Register 2....45
Table 2.96. FIFO Error Register 46
Table 2.97. RISC-V Bulk Data Register....46
Table 2.98. PCIe Config Register 1 @ RISC-V Clock....46
Table 2.99. PCIe Config Register 2 @ RISC-V Clock....46
Table 2.100. PCIe Bulk Data Register @ RISC-V Clock....47
Table 2.101. PCIe Config Register 1 47
Table 2.102. PCIe Config Register 2 47
Table 2.103. PCIe FIFO Status Register 47
Table 2.104. PCIe Bulk Data Register....48
Table 2.105. RISC-V Config Register 1 @ PCIe Clock....48
Table 2.106. RISC-V Config Register 2 @ PCIe Clock....48

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.107. RISC-V Bulk Data Register @ PCIe Clock....48
Table 2.108. FIFO DMA Register Map....49
Table 2.109. FIFO DMA Control Registers....49
Table 2.110. DEST_BASE_ADDR Register....49
Table 2.111. DEST_END_ADDR Register....49
Table 2.112. PING Ready Address Register....49
Table 2.113. PONG Ready Address Register....50
Table 2.114. PING PONG Index Register....50
Table 2.115. Write Status Register....50
Table 2.116. Read Status Register....50
Table 2.117. Descriptor Format....52
Table 2.118. Status Format....53
Table 2.119. Register Address (0x0)....54
Table 2.120. Register Address (0x4)....55
Table 2.121. Register Address (0x8)....55
Table 2.122. Register Address (0xC)....55
Table 2.123. Register Address (0x10)....55
Table 2.124. Register Address (0x14)....55
Table 2.125. Register Address (0x18)....55
Table 2.126. Register Address (0x1C)....56
Table 2.127. Register Address (0x20)....56
Table 2.128. Register Address (0x24)....56
Table 2.129. Register Address (0x28)....56
Table 2.130. CNN Co-Processor Unit Registers....57
Table 2.131. CNN Co-Processor unit control register....57
Table 2.132. CNN Co-Processor Unit Register....57
Table 2.133. Sign Select Configuration Register....57
Table 2.134. Input Offset Configuration Register....58
Table 2.135. Filter Offset Configuration Register....58
Table 2.136. Filter Offset Configuration Register....58
Table 2.137. Input Depth Configuration Register....58
Table 2.138. Input Data Address Configuration Register....58
Table 2.139. Filter Data Address Configuration Register....59
Table 2.140. CNN Co-Processor Unit Output Register....59
Table 2.141. Predictive Maintenance and Motor Control Registers....60
Table 2.142. Motor Control 0 – Minimum RPM....60
Table 2.143. Motor Control 1 – Maximum RPM....60
Table 2.144. Motor Control 2 – RPM PI Control Loop Integrator Gain (kI)....61
Table 2.145. Motor Control 3 – RPM PI Control Loop Proportional Gain (kP)....61
Table 2.146. Motor Control 4 – Torque PI Control Loop Integrator Gain (kI)....61
Table 2.147. Motor Control 5 – Torque PI Control Loop Proportional Gain (kP)....61
Table 2.148. Motor Control 6 – Synchronization Delay and Control....62
Table 2.149. Motor Control Register 7 – Target RPM....63
Table 2.150. Motor Control Register 8 – Target Location....63
Table 2.151. Motor Control Register 9 – Current Location....63
Table 2.152. Motor Status Register 0 – RPM....63
Table 2.153. Motor Status Register 1....63
Table 2.154. Predictive Maintenance Control Register 0....64
Table 2.155. Predictive Maintenance Control Register 1....65
Table 2.156. Predictive Maintenance Status Register....65
Table 2.157. Predictive Maintenance Current/Voltage Data Register....66
Table 2.158. Predictive Maintenance Current/Voltage Data Register....66
Table 2.159. Versa Board Switch Status Register....66
Table 2.160. Versa Board LED & PMOD Control Register....67

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.161. SPI Master Register Map 69
Table 2.162. I^{2}C Master IP Core Registers Summary 71
Table 2.163. UART Register Map 75
Table 3.1. Main System Resource Utilization 77
Table 3.2. Node System Resource Utilization 77
Table 4.1. Types of UART Commands 78
Table 4.2. Types of GPIO Commands 79
Table A.1. Predictive Maintenance Training Network Topology 104

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AHBL	Advanced High-performance Bus-Lite
AI	Artificial Intelligence
API	Application Programming Interface
BLDC	Brushless DC
CCU	CNN Co-Processor Unit
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DMA	Direct Memory Access
FIFO	First-In-First-Out
ISR	Interrupt Service Routines
ML	Machine Learning
QSPI	Quad Serial Peripheral Interface
RISC-V	Reduced Instruction Set Computer-V
RTL	Register-Transfer Level
UART	Universal Asynchronous Receiver-Transmitter

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

1. Introduction

Lattice Automate stack provides a solution for industrial automation that includes predictive maintenance using ML/AI, communication over Ethernet cable and a BLDC motor control IP implemented in RTL. The solution enables user to control multiple motors connected to node systems that are chained using Ethernet cable. The main system that synchronizes operations of node system also runs neural network trained using RISC-V and CNN Co-Processor for predictive maintenance. The entire solution can work with or without external host. We provide reference design with a user interface that runs on host and controls motor operations. The user interface also displays the status of motor and alerts user when motor requires maintenance. User can use all APIs provided with this reference design and can implement entire system without host system. In this case C/C++ code running on RISC-V sends required commands to control motors. The entire system with all sub-components are shown in further sections.

Lattice Automate Stack 1.0 supports web-based user interface which is running on host (system PC) and single chain of nodes for controlling the motors.

Lattice Automate Stack 1.1 supports two chains of nodes which can be connected to 1 main system board. All the nodes are synchronized physically. Main system supports dynamic pulse based system synchronization scheme, in which it checks nodes disconnection during runtime and compensate clock ppm to calculate synchronization delay. It supports OPC UA server/client-based user interface which is running on host PC and client are running on Raspberry Pi board.

Lattice Automate Stack 2.0 supports all features of Lattice Automate Stack 1.1. It supports MQTT broker/client-based host application, Python Interface as host control, and also supports PCIe® interface as host for high speed applications. In the node side, it has motor IP for motor-based features and also has standard SPI Master, UART Interface (Modbus) and I²C Master interfaces to connect various peripherals (sensors) into system.

Figure 1.1 shows the Automate Stack solution and its sub components.

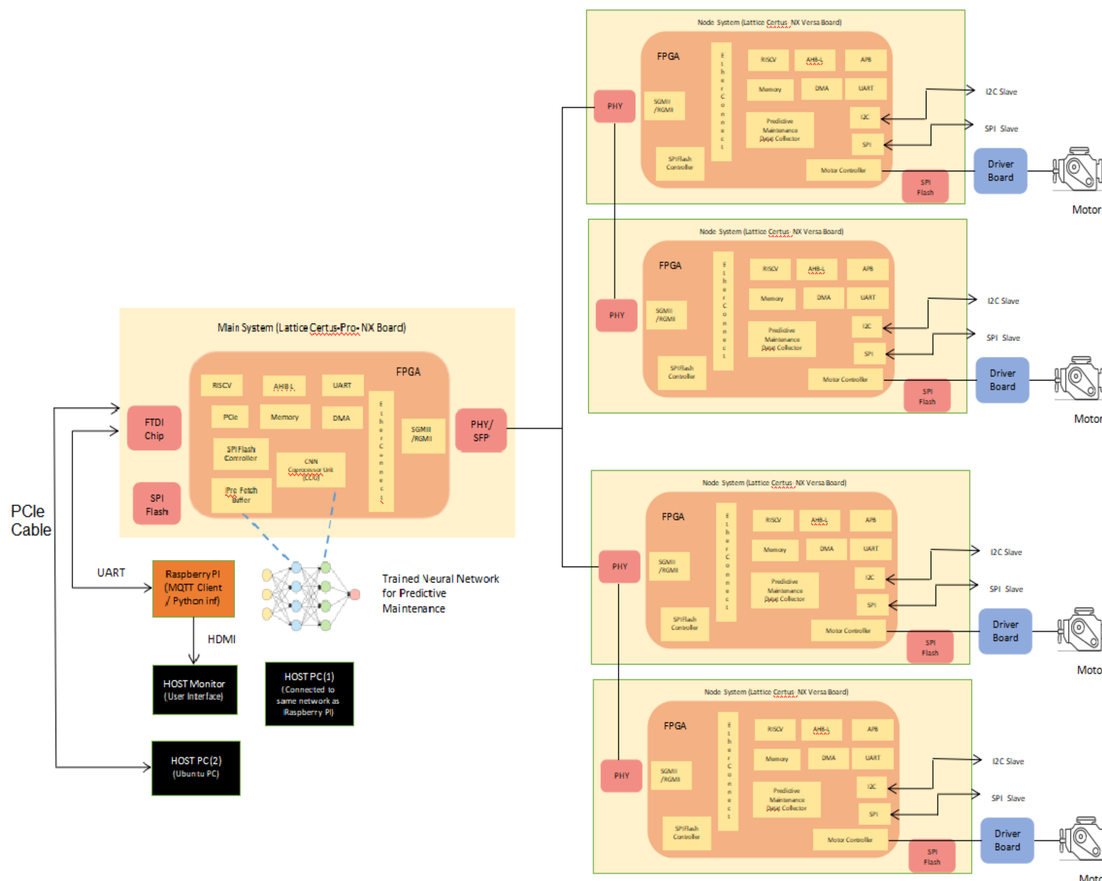


Figure 1.1. Top Level Block Diagram of Automate Stack 2.0

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

1.1. Components

The Automate Stack 2.0 release includes the following components:

- System on Chip (SOC)
  - Main System IPs
    EtherControl IP (With SGMII/RGMII (phy or sfp)), FIFO DMA, CNN Co-Processor Unit (CCU) and SPI Flash Controller, PCIe DMA, PCIe-to-RISC-V Bridge and Reset Synchronizer.
  - Node System IPs
    EtherControl IP (With SGMII/RGMII (phy or sfp)), FIFO DMA, BLDC motor control IP, Data collector for predictive maintenance, UART for Modbus, I^2C Master and SPI Master.
- Software
  - Firmware (APIs)
    APIs to send instructions to motor control IP, collect status of motors and collect data for predictive maintenance
    Compiled TensorFlow-Lite C++ library for RISC-V (Required for neural network inference).
  - Machine Learning
    Trained Neural Network for predictive maintenance, script to train network with user collected data.
  - User Interface
    Controls motor, collects status and data for predictive maintenance, displays warning when maintenance required.

Note: The generic RISC-V subsystem components are excluded from the list of components.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2. Design Overview

2.1. Theory of Operation

The overall architecture is shown in Figure 2.1. The automate stack consists of one Main System (MS) with multiple EtherControl master and multiple Node Systems (NS). A host R-pi board is connected to the MS through Uart port and another host is connected to MS through PCIe cable .An application software with user interface that can send commands to the MS and receive motor maintenance data from the system for AI training. The MS can propagate the commands to NS for motor control and gather maintenance data from NS.

The Certus™-NX versa board and CertusPro™-NX versa board are used for basic demo of complete system.

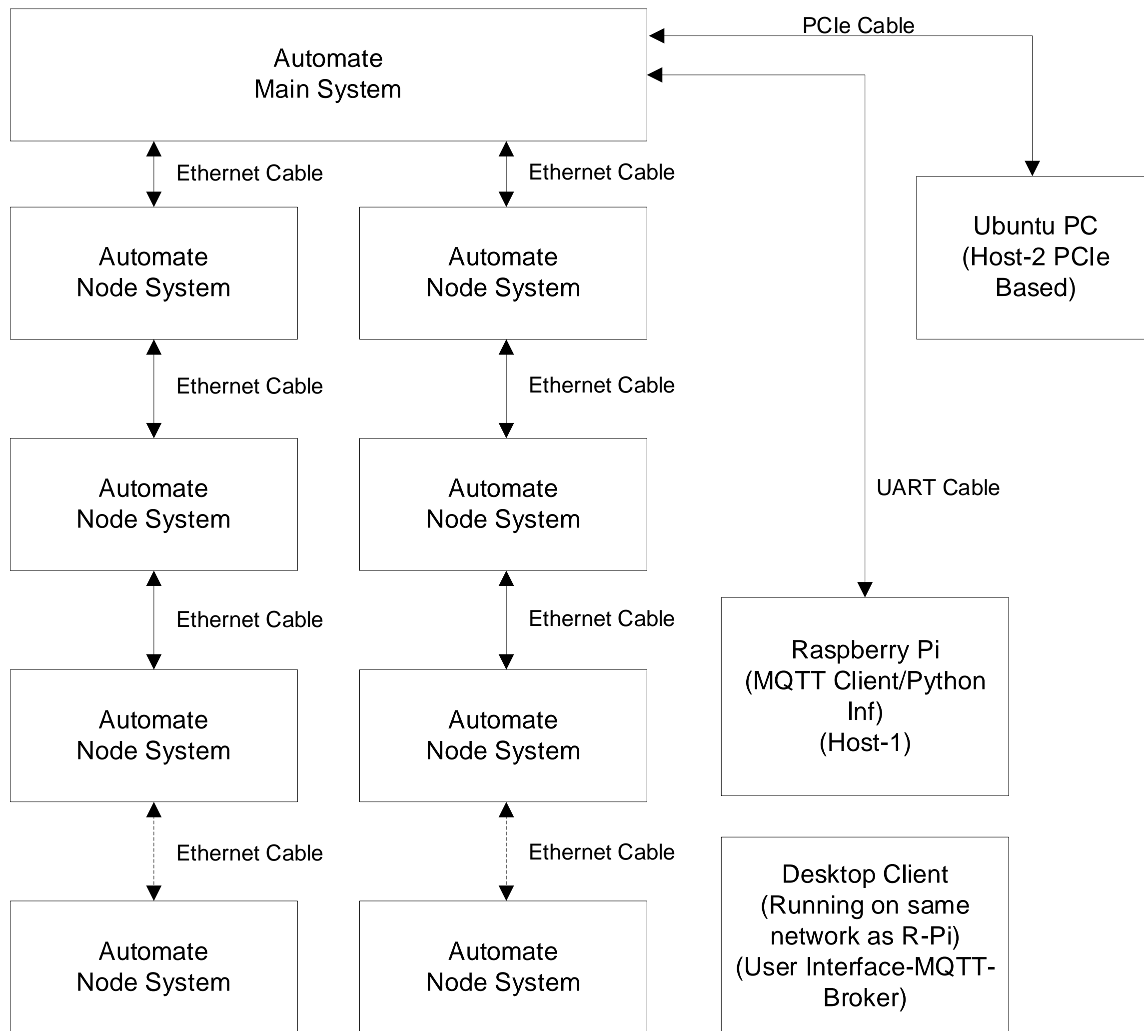


Figure 2.1. Automate Stack 2.0 Architecture

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.2. FPGA Design

2.2.1. Main System

The Main System architecture is shown in Figure 2.2. The AHBL Interconnect has four masters and nineslaves:

The following are the AHBL Interconnect masters:

- RISC-V CPU Instruction Cache
- RISC-V CPU Data Cache
- CNN Co-Processor Unit
- FIFO DMA

The following are the AHBL Interconnect slaves:

- ISR RAM
- Data RAM (port S0 and S1)
- CNN Co-Processor Unit
- FIFO DMA
- EtherControl
- AHBL2APB Bridge
- PCIe-to-RISC-V Bridge
- QSPI Memory Controller with prefetch buffer (SPI Flash Controller)

The RISC-V CPU, CNN Co-Processor Unit, and FIFO DMA access data to the shared memory Data RAM, EtherControl, UART, and QSPI through the AHBL2APB bridge. The UART, SPI Flash Controller, and EtherControl generate interrupts to the RISC-V CPU.

For performance and nearly deterministic latency (DL), it uses port S0 of the Data RAM exclusively for RISC-V CPU access. The other two masters, CNN Co-Processor Unit and FIFO DMA, access port S1 of the Data RAM. This way, the contention is avoided.

EtherControl Master has two AHBL bus interfaces. One interface is used to connect EtherControl master to RISC-V based host path and another interface is used to connect EtherControl master to PCIe DMA based high speed host path.

PCIe DMA AHBL interconnect interfaces EtherControl Master, PCIe-to-RISC-V Bridge and PCIe DMA IPs. PCIe DMA IP supports PCIe-based host interface which can be connected to Linux-based PC.

Both the host path are independent of each other and both can perform same operation on nodes. But through PCIe-to-RISC-V Bridge information can be exchanged between RISC-V and PCIe DMA IP.

Note: Physically there is only one piece of shared memory but with two independent ports. In the memory map, S0 is assigned with a lower base address and S1 is assigned with a higher base address. In real terms, these refer to the same physical address. The two different address spaces for S0 and S1 allow the AHBL Interconnect to route the transaction to the right port.

For better performance and nearly deterministic latency, EtherControl port supports two physical interface (two master port) and it allows system to maintain two different chains of node and each chain can support up to 8 nodes.

The main firmware is stored in the external SPI flash. The ISR RAM contains the initial boot code for RISC-V as well as the interrupt service routines (ISRs) and other performance-critical functions. There are two implementation options:

- During boot, the bootloader copies the ISR code from the external flash to internal ISR RAM. It then sets up the ISR function pointer to this internal memory address.
- The ISR code is integrated in the bitstream and firm the ISR code in the ISR RAM as ROM code.

The first option can be used during initial development for debugging. The second option can be used in the final production release since it does not increase any system boot time.

The system is working at CPU frequency of 75 MHz, the protocol is working at 125 MHz and PCIe based host @ 62.5 MHz.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

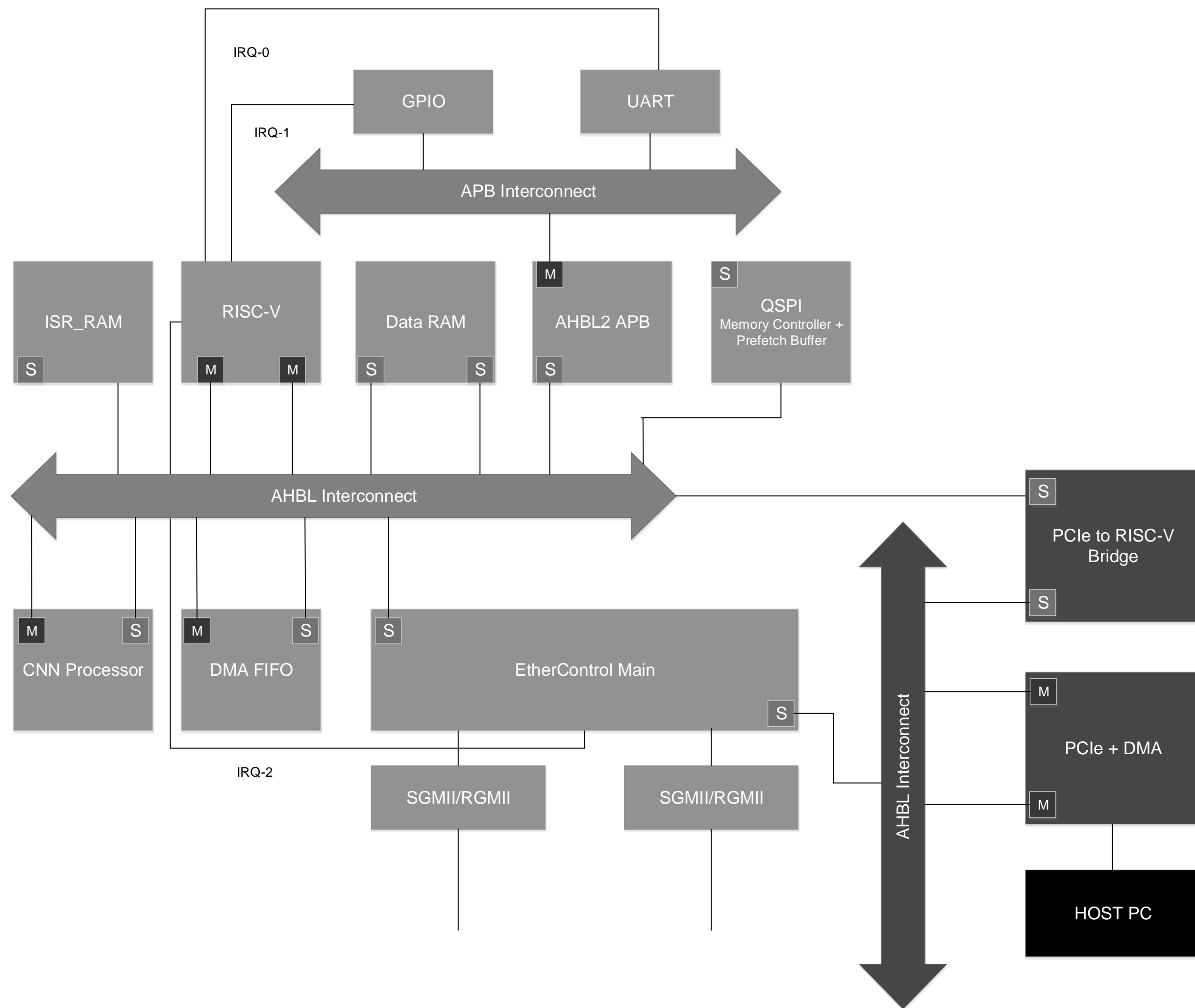


Figure 2.2. Main System Architecture

RISC-V-Based Path

The firmware binary is stored in the external QSPI flash. When RISC-V at the Main System boots up, it sets the registers at QSPI Memory Controller (SPI Flash Controller). Then RISC_V jumps to the loaded firmware and executes the binary.

The application software at the host PC sends an interrupt through the UART port. Then RISC-V CPU fetches the commands and data from the host through UART by accessing APB port through the AHBL Interconnect, AHBL2APB bridge, and APB Interconnect.

RISC-V CPU sets the registers inside the CNN Co-Processor Unit and starts PDA operation. RISC-V CPU polls another register in the CNN Co-Processor Unit to check its operation status. RISC-V CPU requests for the new data for predictive maintenance from the slave PDM data collector by sending instruction through EtherControl IP. The data received from the slave through EtherConnect is transferred to data memory with DMA operation or sent to the host PC through UART.

For the motor control commands from the host PC, RISC-V CPU fetches them from UART and then sends them to EtherControl, which performs packetization and sends them to the downstream Node System.

The information is written/read to/from peripherals connected to Nodes through SPI Master/I^{2}C Master/ Modbus same as information is written (config)/read to (status)/from motor control IP.

RISC-V CPU gathers predictive maintenance data from downstream Node Systems through EtherControl and sends this data to the host PC through UART. RISC-V CPU reads data from EtherControl through its AHBL Slave port, performs data processing, stores the data in the Data RAM, and then sends it to UART through APB. Alternatively, EtherControl can send downstream data to the FIFO DMA through its FIFO port, and FIFO DMA can write the data to the Data RAM or UART directly.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

User can press buttons on the board that generate an interrupt by the GPIO block to RISC-V CPU. The interrupt service route firmware queries the interrupt status of the GPIO block and performs corresponding actions such as sending commands to start, stop, accelerate or decelerate all motors downstream.

At the end of every predictive maintenance cycle in software running on RISC-V, an update is sent to the host PC through UART.

PCIe DMA-Based Path

The application software running Linux PC connected to main system through the PCIe interface can check if EtherControl Path is engaged by other host(R-Pi). Linux PC can send/receive data to/from PCIe DMA module. PCIe DMA module can send data to EtherControl module to set certain register of EtherControl to send commands to nodes for motor controlling and different peripherals controlling connected to SPI/I^{2}C/UART interfaces. PCIe DMA module can also receive data from node through the EtherControl module for training purpose. The PCIe DMA based host path is faster than RISC-V based host path.

Figure 2.2 shows that EtherControl block have two Ethernet ports (port 0 and port 1) in downstream direction. It means EtherControl master supports two master to support two split chains of nodes to improve performance.

Both chains can be synchronized and both the ethernet ports have options to select RGMII/SGMII physical interfaces. Both ports have options to select PHY or SFP IC for MII-to-RJ45 conversion.

2.2.1.1. Memory Map

The Main System memory map is defined in Table 2.1 and Table 2.1.

Table 2.1. Main System Memory Map (RISC-V)

Base Address	End Address	Range (Bytes)	Range (Bytes in hex)	Size (kB)	Block
00190000	0197FFFF	32768	8000	32	CPU instruction RAM
00080000	000807FF	2048	800	2	CPU PIC TIMER
00080800	00080BFF	1024	400	1	GPIO
00080C00	0009FFFF	128000	1F400	125	RESERVED
000A0000	000A03FF	1024	400	1	CNN Co-Processor Unit (CCU)
000C0000	000FFFFF	262144	40000	256	CPU Data Ram Port S0: base address 0x000C0000 Port S1: base address: 0x000E0000
00100000	00107FFF	32768	8000	32	FIFO DMA
00108000	0010FFFF	32768	8000	32	EtherControl
00110000	0017FFFF	458752	70000	448	RESERVED
00000000	0007FFFF	512000	7D000	512	SPI FLASH CONTROLLER
00182000	001823FF	1024	400	1	UART
00184000	00FFFFFF	1.5E+07	E7C000	14832	RESERVED
01000000	01FFFFFF	1.7E+07	1000000	16384	External SPI flash
00110000	00117FFF	32768	8000	32	PCIe to RISC-V Bridge

Table 2.2. Main System Memory Map (PCIe)

Base Address	End Address	Range (Bytes)	Range (Bytes in hex)	Size (kB)	Block
00108000	0010FFFF	32768	8000	32	EtherControl
00110000	00117FFF	32768	8000	32	PCIe to RISC-V Bridge

Note: The above address are for PCIe DMA IP (AHBL Master) to Slaves. But from host to PCIe DMA, the IP addressing is defined in further sections. Host use 0x0000 base address for PCIe register space, 0x1000 base address for descriptor memory and 0x3000 base address for application IPs (EtherControl and PCIe-to-RISC-V Bridge).

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.2.2. Node System

The Node System architecture is shown in Figure 2.3. It consists of one AHBL Interconnect with three masters and eight slaves.

The following are the Node System masters:

- RISC-V CPU Instruction Cache
- RISC-V CPU Data Cache
- FIFO DMA

The following are the Node System slaves:

- ISR RAM
- Data RAM (port S0 and S1)
- Motor Control and PDM Data Collector (port S0 and S1)
- FIFO DMA
- EtherControl
- QSPI Memory Controller with prefect buffer(SPI Flash Controller)
- AHBL2APB bridge.
- SPI Master
- I^2C Master

AHBL2APB bridge is connected to APB Interconnect which is having 3 APB interface based slaves SPI Master, I^2C Master and UART to interface different peripherals in the system (for example, sensors).

For Data RAM with two AHBL slave ports, see the description in the previous section. For Motor Control and PDM Data Collector, it has two AHBL slave ports (S0 and S1). Port S0 is used to access the Motor Control and PDM registers while port S1 is used to access the data collected by PDM Data Collector.

The main firmware is stored in the external SPI flash. The ISR RAM contains the initial boot code for RISC-V as well as the interrupt service routines (ISRs) and other performance-critical functions. There are two implementation options:

- During boot, the bootloader copies the ISR code from the external flash to internal ISR RAM. It then sets up the ISR function pointer to this internal memory address.
- The ISR code is integrated in the bitstream and firm the ISR code in the ISR RAM as ROM code.

  The first option can be used during initial development for debugging. The second option can be used in the final production release since it does not increase any system boot time.

The system is working at frequency of 75 MHz while the protocol is working at 125 MHz.

The CPU can access data from the Data RAM, access the register file inside EtherControl, and control the registers at FIFO DMA and QSPI Memory Controller. Either RISC-V CPU or FIFO DMA can move the data stored at the register file inside EtherControl to Motor Control block. They can also move the data collected by PDM Data Collector back to EtherControl and send out through the Ethernet upstream port.

There is one feature added in EtherControl IP in protocol layer. It supports additional frame/packet type *10* which enables the system to enhance performance while fetching bulk data. More details is given in the EtherControl user guide.

There is no major changes in EtherControl Slave module, but it has three more slave addition for different applications. RISC-V CPU can also send/receive data to/from different peripherals connected to system through SPI Master/I^2C Master/UART (modbus).

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

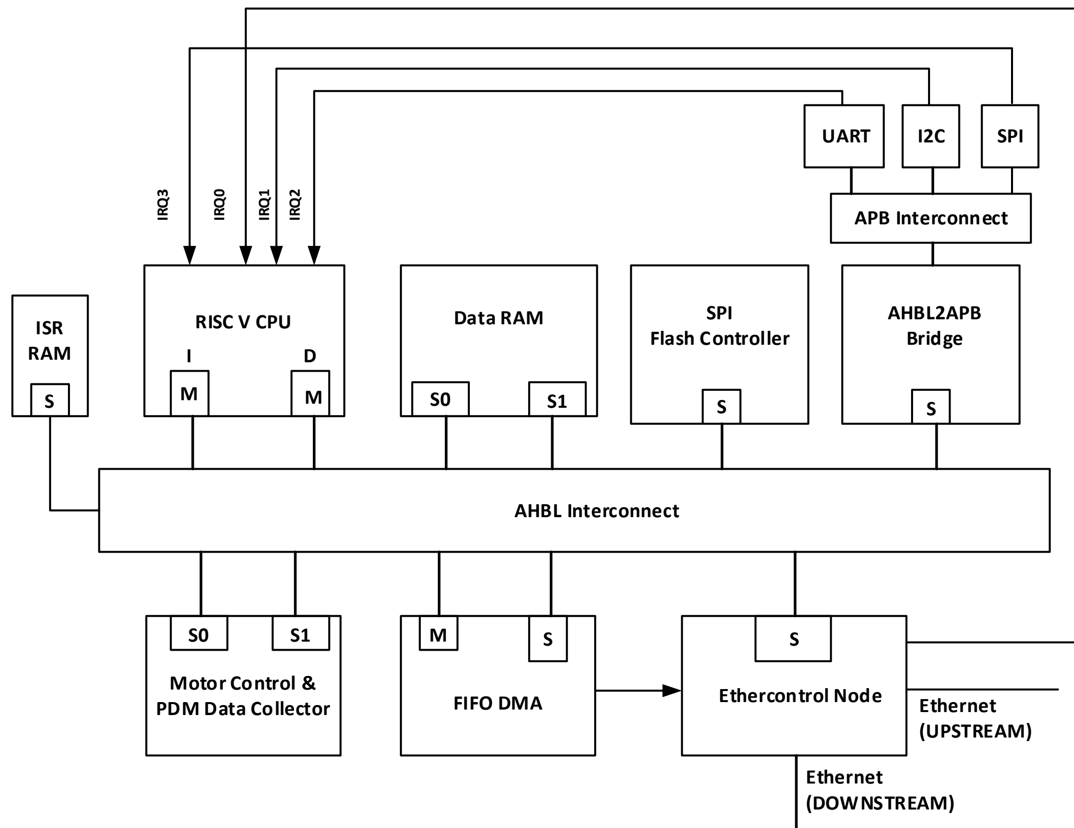


Figure 2.3. Node System Architecture

2.2.2.1. Node System Memory Map of Node System

The Node System memory map is defined in Table 2.3.

Table 2.3. Node System Memory Map

Base Address	End Address	Range (Bytes)	Range (Bytes in hex)	Size (Kbytes)	Block
00190000	00197FFF	32768	8000	32	CPU instruction RAM
00080000	000807FF	2048	800	2	CPU PIC TIMER
00080800	000BFFFF	260096	3F800	254	RESERVED
000C0000	000FFFFF	262144	40000	256	CPU Data Ram Port S0 base address: 0x000C0000 Port S1 base address: 0x000E0000
00100000	00107FFF	32768	8000	32	FIFO DMA
00108000	0010FFFF	32768	8000	32	EtherControl
00110000	0017FFFF	458752	70000	448	RESERVED
00000000	0007FFFF	512000	7D000	512	SPI Flash Controller
001864000	001867FF	1024	400	2	UART
00184000	00185FFF	8192	2000	8	Motor Control & PDM Data Collector Port S0 base address: 0x00184000 Port S1 base address: 0x00185000
00186000	00FFFFFF	15179776	E7A000	14824	RESERVED
01400000	01FFFFFF	16777216	1000000	16384	External SPI flash
001868000	00186BFF	1024	400	1	SPI Master
00186000	001863FF	1024	400	1	I^{2}C Master

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.3. EtherControl IP

The EtherControl block is needed in both Main System and Node System. There is a Verilog parameter SYSTEM_TYPE, which sets this block as either Main System or Node System. For the Main System, there is no Ethernet Upstream; only two Ethernet downstream ports. For Node System, it is has both Ethernet Upstream(1) and Ethernet Downstream(1) ports. For the last Node System, the Ethernet Downstream port can be disabled. Input/Output FIFO interface is selected using SYSTEM_TYPE parameter.

In the main system, the EtherControl IP has an output FIFO interface to send bulk data to DMA FIFO block while in the node system, the EtherControl block has an input FIFO interface to receive bulk data from DMA FIFO module, which is coming from the Data Collector IP. The AHBL interface 0 (AHBL_S_0) is used to support one host along with FIFO interface for bulk data. The AHBL interface 1 (AHBL_S_1) is used to support another host without FIFO interface to control same operations of IP as AHBL_S_0. The AHBL_S_1 interface is used to read/write bulk data from/to IP as well.

Both AHBL interfaces are used independently from user end, but user needs to check each time if IP is engaged by another AHBL interface.

Two AHBL interfaces are available for EtherControl Master only, depending on the System Type parameter.

The Sync Pulse generator block is available in the EtherControl master only. It is used to generate pulse for dynamically synchronization of nodes.

The EtherControl consists of an existing IP block, EtherConnect, register file, and glue logic as shown in Figure 2.4.

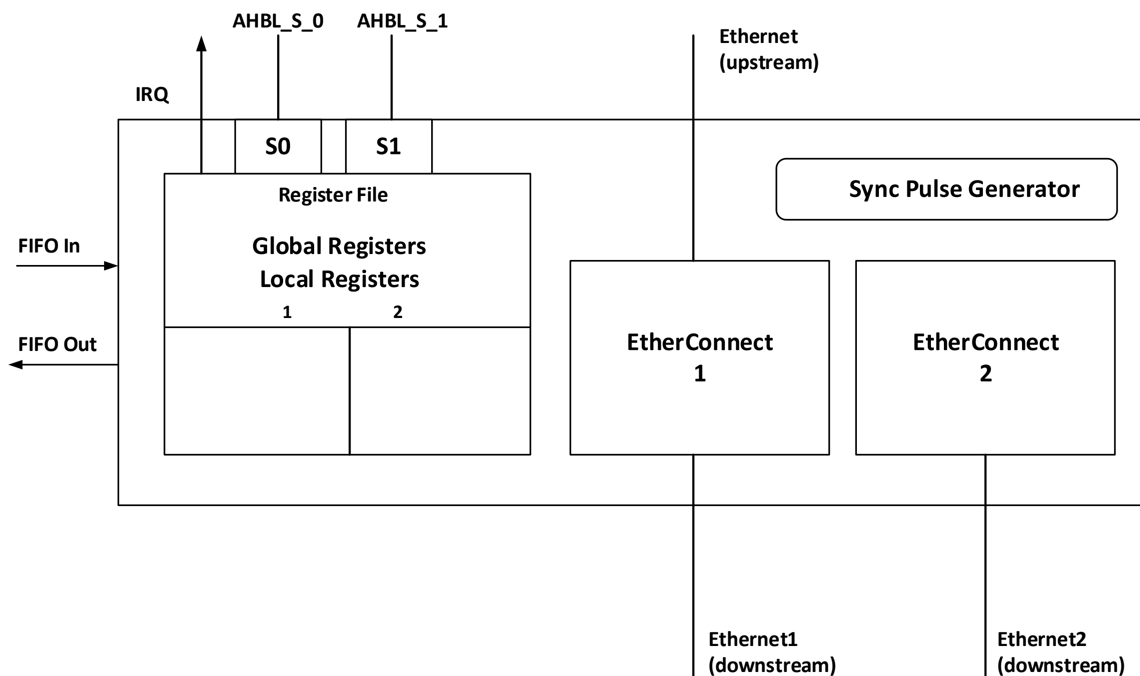


Figure 2.4. EtherControl Block Diagram

Table 2.4. EtherControl Interfaces

Interface	Direction	Description
IRQ	Output	Interrupt to RISC-V CPU
FIFO	Output	FIFO output to FIFO DMA
FIFO	Input	FIFO input to EtherControl
AHBL Slave 0	Input and Output	AHBL slave port for host 1 along with FIFO output interface to control IP.
AHBL Slave 1	Input and Output	AHBL slave port for host 2 independent of FIFO and AHBL 0 interface to control IP.
Ethernet Upstream	Input and Output	Send Ethernet packets to Main System or Upper Node System This interface is disabled for Main System EtherControl.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Interface	Direction	Description
Ethernet 1 Downstream	Input and Output	Send ethernet packets to lower Node Systems This interface is disabled for the last Node System EtherControl.
Ethernet 2 Downstream	Input and Output	Send ethernet packets to lower Node Systems This interface is disabled for the last Node System EtherControl.

2.3.1. Features

The key features of the EtherControl IP include:

- Real time Ethernet network support
- Two chain support
- Full Duplex data communication support
- RGMII interface support
- SGMII interface support
- AHBL Node interfaces for controlling IP from AHBL based master block
- FIFO Interface for bulk data transfer (both normal and extended mode)(only for AHBL Bus 0)
- Runtime Cable Break Detection Support
- Propagation Delay adjustment(Synchronization) Support
- Parameter based Main and Node Selection
- Maximum of 32 nodes support
- Two AHBL Bus support for EtherControl Master
- Max 256 bytes data length support
- Random Node access support – EtherControl Master
- RGMII/SGMII Selection
- 1G(125 MHz) physical interface support- RGMII/SGMII PHY, SFP support
- Dynamic/Runtime node scanning
- 4 kB Rx and Tx data buffers support
- Configuration Write(Motor Configuration) , Status Read(Motor Status), Bulk data read(PDM data) – Normal and Extended
- Interrupt support (only for AHBL Bus 0)

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.3.2. EtherControl Master

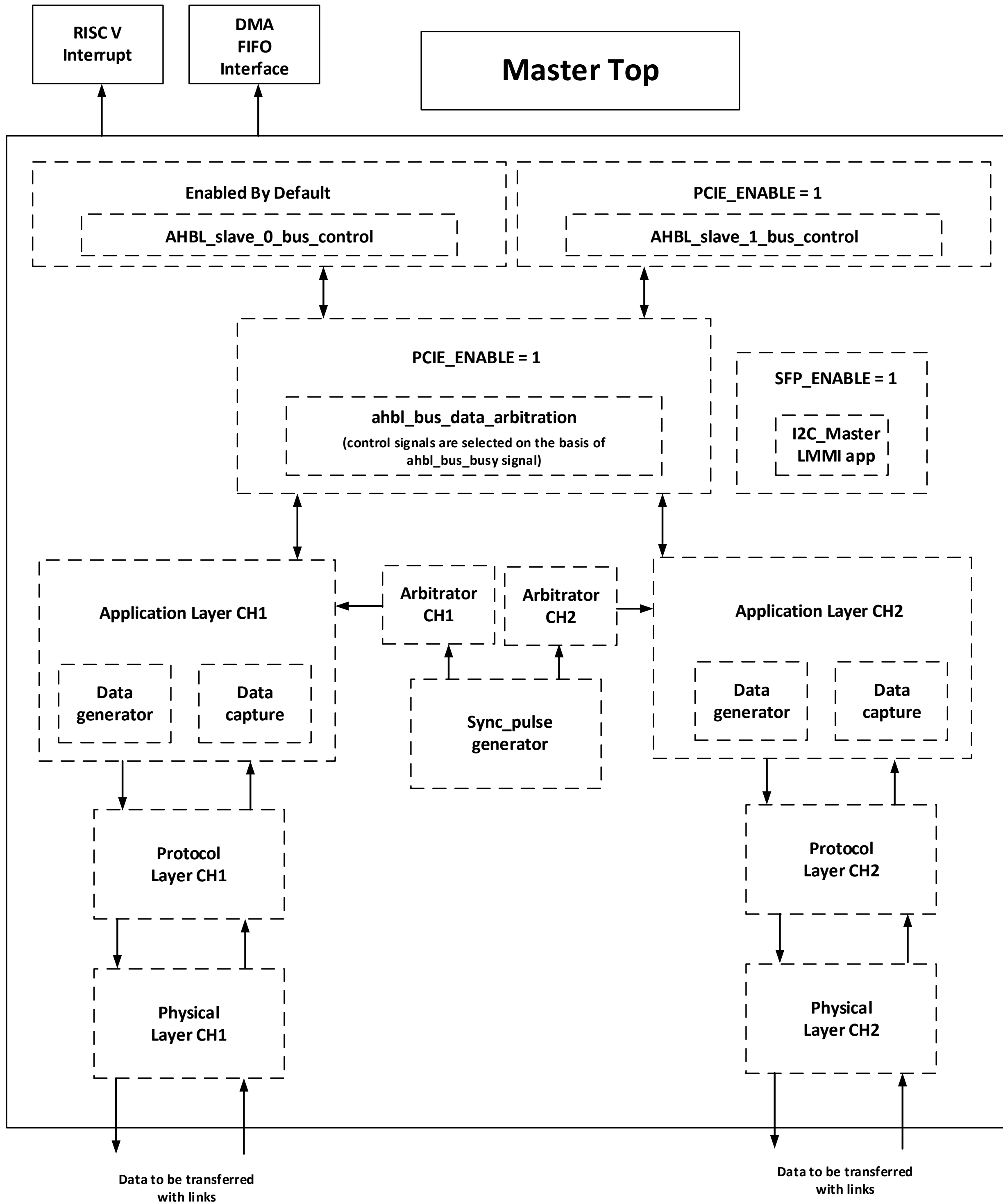


Figure 2.5. EtherControl Master Block Diagram

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.3.3. Register Description

2.3.3.1. EtherControl Master (RISC-V)

The register address map for AHBL Bus 0 (RISC-V) shown in Table 2.5 specifies the available IP Core registers for the main system configuration. The offset of each register increments by four to allow easy interfacing with the processor and System Buses. In this case, each register is 32-bit wide wherein the used and unused bits are mentioned. The unused bits are treated as reserved – read access returns 0. The registers are divided into Global and Local ones. The global registers are common for all the chains, while the local ones are local to the respective chains.

Table 2.5. EtherControl Master Global Register Map (RISC-V)

EtherControl Register Name	Register Function	Base Address (0x00108000)	Access
DMACTR_R	DMA FIFO Enable/AHBL Disable Register	Base + 0x00	Read/Write
PHLNK_R	PHY Link Status Register	Base + 0x04	Read
NDACT_R	Active Nodes Register	Base + 0x08	Read
FSRPDM_R	FIFO Status Register for PDM Data CDC	Base + 0x0C	Read
ETHINTR_R	Interrupt Poll Register	Base + 0x10	Read
CLRCVD_R	Clear Interrupt Received Register	Base + 0x14	Read/Write
TX_ALL_STRT_R	Transaction start for all chains	Base + 0x18	Read/Write
DTOUT_R	Node Response PDM Data Register	Base + 0x1C	Read
IP_STATUS_R	IP Busy Status	Base + 0x20	Read/Write
AHBL_TOUT_R	AHBL Bus Timeout Count Register	Base + 0x28	Write

Table 2.6. EtherControl Master Local Chain 1 Register Map (RISC-V)

EtherControl Register Name	Register Function	Base Address (0x0010800)	Access
TXSTR_R_1	Start Transaction Register	Base + 0x00	Read/Write
PKTHD_R_1	Packet Head Register	Base + 0x04	Read/Write
FRNUM_R_1	Frame Number Register	Base + 0x08	Read/Write
NDCNT_R_1	Number of Node Register	Base + 0x0C	Read/Write
NDLN_R_1	Node Data Length Register	Base + 0x10	Read/Write
MTDT_R_1	Node Request Data Burst Register	Base + 0x14	Read/Write
RQDT_R_1	Node Request Type Register	Base + 0x18	Read/Write
RQAD_R_1	Node Address Register	Base + 0x1C	Read/Write
CRCNT_R_1	CRC Count Register	Base + 0x20	Read
INTR_R_1	Interrupt Info Register	Base + 0x24	Read
FSRREQD_R_1	FIFO Status Register Request Data	Base + 0x28	Read
MTRST_R_1	Node Motor Status Register	Base + 0x100 to 0x1FC	Read
DLY_R_1	Node Delay Register	Base + 0x200 to 0x2FC	Read

Table 2.7. EtherControl Master Local Chain 2 Register Map (RISC-V)

EtherControl Register Name	Register Function	Base Address (0x00108400)	Access
TXSTR_R_2	Start Transaction Register	Base + 0x00	Read/Write
PKTHD_R_2	Packet Head Register	Base + 0x04	Read/Write
FRNUM_R_2	Frame Number Register	Base + 0x08	Read/Write
NDCNT_R_2	Number of Node Register	Base + 0x0C	Read/Write
NDLN_R_2	Node Data Length Register	Base + 0x10	Read/Write
MTDT_R_2	Node Request Data Burst Register	Base + 0x14	Read/Write
RQDT_R_2	Node Request Type Register	Base + 0x18	Read/Write
RQAD_R_2	Node Address Register	Base + 0x1C	Read/Write
CRCNT_R_2	CRC Count Register	Base + 0x20	Read

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

EtherControl Register Name	Register Function	Base Address (0x00108400)	Access
INTR_R_2	Interrupt Info Register	Base + 0x24	Read
FSRREQD_R_2	FIFO Status Register Request Data	Base + 0x28	Read
MTRST_R_2	Node Motor Status Register	Base + 0x100 to 0x1FC	Read
DLY_R_2	Node Delay Register	Base + 0x200 to 0x2FC	Read

The Global register description is given below:

Table 2.8. DMA FIFO Enable/AHBL Disable Register

DMACTR_R				Base + 0x00
Byte	3	2	1	0
Name	DMACTR_R			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

DMACTR_R[0]: 0: DMA FIFO enabled, AHBL disabled | 1: DMA FIFO disabled, AHBL enabled

Table 2.9. PHY Link Status Register

PHLNK_R				Base + 0x04
Byte	3	2	1	0
Name	Physical Link Chain 2		Physical Chain 1	
Default	0	0	0	0
Access	R			

PHLNK_R[0]: 1: Main System PHY link established for chain 1 and 0: Main System PHY link not established for chain 1

PHLNK_R[15:1]: Each bit from bit 1 to bit 15 shows the link status of the respective nodes in chain 1

PHLNK_R[16]: 1: Main System PHY link established for chain 2 and 0: Main System PHY link not established for chain 2

PHLNK_R[31:17]: Each bit from bit 17 to bit 31 shows the link status of the respective nodes in chain 2

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.10. Active Nodes Register

NDACT_R				Base + 0x08
Byte	3	2	1	0
Name	Active Node Chain 1+2	Active Node Chain 1+2	Active Node Chain 2	Active Node Chain 1
Default	0	0	0	0
Access	R			

NDACT_R[7:0]: Gives number of nodes actually connected physically to the system

NDACT_R[15:8]: Gives number of nodes actually connected physically to the system in chain 2

NDACT_R[31:16]: Gives total number of physically connected nodes in both chains

Table 2.11. FIFO Status Register for PDM Data

FSRPDM_R				Base + 0x0C
Byte	3	2	1	0
Name	FSRPDM_R			
Default	Reserved	Reserved	Reserved	0
Access	R			

FSRPDM_R[0]: Empty signal of RX FIFO

FSRPDM_R[1]: Full signal of RX FIFO

FSRPDM_R[2]: Overflow error of RX FIFO

FSRPDM_R[3]: Underflow error of RX FIFO

FSRPDM_R[4]: Reserved

FSRPDM_R[5]: Reserved

FSRPDM_R[6]: Reserved

FSRPDM_R[7]: Reserved

Table 2.12. Clear Interrupt Received Register

CLRCVD_R				Base + 0x10
Byte	3	2	1	0
Name	CLRCVD_R			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

CLRCVD_R[0] : Received clr bit from CPU

CLRCVD_R[7:1] : Reserved

CLRCVD_R[31:8] : Reserved

Table 2.13. Interrupt Polling Register

ETHINTR_R				Base + 0x14
Byte	3	2	1	0
Name	Ethernet Interrupt from Chain 2		Ethernet Interrupt from Chain 1	
Default	Reserved	0	Reserved	0
Access	R/W			

ETHINTR_R[0]: Interrupt bit from Chain 1

ETHINTR_R[7:1]: Reserved

ETHINTR_R[15:8]: Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

ETHINTR_R[16]: Interrupt bit from Chain 2

ETHINTR_R[31:17]: Reserved

Table 2.14. Start Transaction in All Chains

TX_ALL_STRT_R				Base + 0x18
Byte	3	2	1	0
Name	TX_ALL_STRT_R			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

TX_ALL_START_R[0]: Received clr bit from CPU

TX_ALL_START_R[7:1]: Reserved

Table 2.15. IP Busy Register

IP_BUSY_R				Base + 0x20
Byte	3	2	1	0
Name	AHBL_Busy_R			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

AHBL_BUSY_R[0]: 1 : AHBL Bus 0 Busy | 0: AHBL 0 bus Free (Only for reading)

AHBL_BUSY_R[1]: 1 : AHBL Bus 1 Busy | 0: AHBL 1 bus Free

AHBL_BUSY_R[7:2]: Reserved

Table 2.16. AHBL_TOUT_R

IP_BUSY_R				Base + 0x28
Byte	3	2	1	0
Name	AHBL_TOUT_R			
Default	0	0	0	0
Access	W			

AHBL_TOUT_R[31:0]: Sets the value of AHBL timeout count to free the bus

The local register 1 description is given below:

Table 2.17. Chain 1 Start Transaction Register

TXSTR_R_1				Base + 0x00
Byte	3	2	1	0
Name	TXSTR_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

TXSTR_R_1[0]: 1: Start the transaction | 0: No transaction

Table 2.18. Chain 1 Packet Head Register

PKTHD_R_1				Base + 0x04
Byte	3	2	1	0
Name	PKTHD_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

PKTHD_R_1[0]: 1: User values are updated | 0: No update

Table 2.19. Chain 1 Frame Number Register

FRNUM_R_1				Base + 0x08
Byte	3	2	1	0
Name	FRNUM_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

FRNUM_R_1[7:0]: Frame number for the current frame

Table 2.20. Chain 1 Number of Node Register

NDCNT_R_1				Base + 0x0C
Byte	3	2	1	0
Name	NDCNT_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDCNT_R_1[7:0]: Number of nodes configured by the user

Table 2.21. Chain 1 Node Data Length Register

NDLN_R_1				Base + 0x10
Byte	3	2	1	0
Name	NDLN_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDLN_R_1[7:0]: Data length of nodes to be configured by the user

Table 2.22. Chain 1 Node Request Data Burst Register

MTDT_R_1				Base + 0x14
Byte	3	2	1	0
Name	MTDT_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

MTDT_R_1[7:0]: Data to be send from the Main System to Node Systems by the user

Table 2.23. Chain 1 Node Request Type Register

RQDT_R_1				Base + 0x18
Byte	3	2	1	0
Name	RQDT_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQDT_R_1[7:0]: Type of data requested by the user

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.24. Chain 1 Node Address Register

RQAD_R_1				Base + 0x1C
Byte	3	2	1	0
Name	RQAD_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQAD_R_1[7:0]: Address requested by the user

Table 2.25. Chain 1 CRC Count Register

CRCNT_R_1				Base + 0x20
Byte	3	2	1	0
Name	CRCNT_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R			

CRCNT_R_1[7:0]: Gives the count of error generated by doing CRC on the data

Table 2.26. Chain 1 Interrupt Info Register

INTR_R_1				Base + 0x24
Byte	3	2	1	0
Name	INTR_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R			

INTR_R_1[31:0]: Gives the type of interrupt generated according to type of available data

0x01 : Motor Configuration

0x02 : Motor Status

0x03 : PDM Data

0x04 : Training Pkt

0x05 : Pkt Head

0x06 : Extended PDM Data

Table 2.27. Chain 1 FIFO Status Register Request Data

FSRREQD_R_1				Base + 0x28
Byte	3	2	1	0
Name	FSRREQD_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R			

FSRREQD_R_1[0]: Overflow error of TX 1 FIFO

FSRREQD_R_1[1]: Underflow error of TX 1 FIFO

FSRREQD_R_1[2]: Empty signal of TX 1 FIFO

FSRREQD_R_1[3]: Full signal of TX 1 FIFO

FSRREQD_R_1[4]: Reserved

FSRREQD_R_1[5]: Reserved

FSRREQD_R_1[6]: Reserved

FSRREQD_R_1[7]: Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.28. Chain 1 Node Motor Status Register

MTRST_R_1				Base + 0x100 - 0x1FC
Byte	3	2	1	0
Name	MTRST_R_1			
Default	0	0	0	0
Access	R			

Base + 0x100 : Node 1 status

Base + 0x104 : Node 2 status ……..(will progress like this for other nodes)

Table 2.29. Chain 1 Node Delay Register

DLY_1				Base +0x200 - 0x2FC
Byte	3	2	1	0
Name	DLY_1			
Default	0	0	0	0
Access	R			

Base + 0x200 : Node 1 Delay

Base + 0x204 : Node 2 Delay ……..(will progress like this for other nodes)

The local register 2 description is given below:

Table 2.30. Chain 2 Start Transaction Register

TXSTR_R_2				Base +0x00
Byte	3	2	1	0
Name	TXSTR_R_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

TXSTR_R_2[0]: 1: Start the transaction | 0: No transaction

Table 2.31. Chain 2 Packet Head Register

PKTHD_R_2				Base +0x04
Byte	3	2	1	0
Name	PKTHD_R_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

PKTHD_R_2[0]: 1: User values are updated | 0: No update

Table 2.32. Chain 2 Frame Number Register

FRNUM_R_2				Base +0x08
Byte	3	2	1	0
Name	FRNUM_R_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

FRNUM_R_2[7:0]: Frame number for the current frame

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.33. Chain 2 Number of Node Register

NDCNT_R_2				Base +0x0C
Byte	3	2	1	0
Name	NDCNT_R_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDCNT_R_2[7:0]: Number of nodes configured by the user

Table 2.34. Chain 2 Node Data Length Register

NDLN_R_2				Base +0x10
Byte	3	2	1	0
Name	NDLN_R_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDLN_R_2[7:0]: Data length of nodes to be configured by the user

Table 2.35. Chain 2 Node Request Data Burst Register

MTDT_R_2				Base +0x14
Byte	3	2	1	0
Name	MTDT_R_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

MTDT_R_2[7:0]: Data to be send from the Main System to Node Systems by the user

Table 2.36. Chain 2 Node Request Type Register

RQDT_R_2				Base +0x18
Byte	3	2	1	0
Name	RQDT_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQDT_R_2[7:0]: Type of data requested by the user

Table 2.37. Chain 2 Node Address Register

RQAD_R_2				Base +0x1C
Byte	3	2	1	0
Name	RQAD_R_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQAD_R_2[7:0]: Address requested by the user

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.38. Chain 2 CRC Count Register

CRCNT_R_2				Base +0x20
Byte	3	2	1	0
Name	CRCNT_R_2			
Default	0	0	0	0
Access	R			

CRCNT_R_2[7:0]: Gives the count of error generated by doing CRC on the data

Table 2.39. Interrupt Info Register

INTR_R_2				Base +0x24
Byte	3	2	1	0
Name	INTR_R_2			
Default	0	0	0	0
Access	R			

INTR_R_2[31:0]: Gives the type of interrupt generated according to type of available data

0x01: Motor Configuration

0x02: Motor Status

0x03: PDM Data

0x04: Training Pkt

0x05: Pkt Head

0x06: Extended PDM Data

Table 2.40. Chain 2 FIFO Status Register Request Data

FSRREQD_R_2				Base +0x28
Byte	3	2	1	0
Name	FSRREQD_R_2			
Default	Reserved	Reserved	Reserved	0
Access	R			

FSRREQD_R_2[0]: Overflow error of TX 1 FIFO

FSRREQD_R_2[1]: Underflow error of TX 1 FIFO

FSRREQD_R_2[2]: Empty signal of TX 1 FIFO

FSRREQD_R_2[3]: Full signal of TX 1 FIFO

FSRREQD_R_2[4]: Reserved

FSRREQD_R_2[5]: Reserved

FSRREQD_R_2[6]: Reserved

FSRREQD_R_2[7]: Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.41. Chain 2 Node Motor Status Register

MTRST_R_2				Base +0x100 – 0x1FC
Byte	3	2	1	0
Name	MTRST_R_2			
Default	0	0	0	0
Access	R			

Base + 0x100 : Node 1 status

Base + 0x104 : Node 2 status ……..(will progress like this for other nodes)

Table 2.42. Chain 2 Node Delay Register

DLY_R_2				Base +0x200 – 0x2FC
Byte	3	2	1	0
Name	DLY_R_2			
Default	0	0	0	0
Access	R			

Base + 0x200 : Node 1 Delay

Base + 0x204 : Node 2 Delay ……..(will progress like this for other nodes)

2.3.3.2. EtherControl Master (PCIe)

The register address map for AHBL Bus 1 (PCIe) shown in Table 2.43 specifies the available IP Core registers for main system configuration. The offset of each register increments by eight to allow easy interfacing with the Processor and System Buses. In this case, each register is 64-bit wide wherein the used and unused bits are mentioned. The unused bits are treated as reserved – read access returns 0. The registers are divided into Global and Local ones. The global registers are common for all the chains while the local ones are local to the respective chains.

Table 2.43. EtherControl Master Global Register Map (PCIe)

PCIe Register Name	Register Function	Base Address (0x00108000)	Access	Used Bits
DMACTR_P	DMA FIFO Enable/AHBL Disable Register	Base + 0x00	Read/Write	[31:0]
PHLNK_P	Phy Link Status Register	Base + 0x00	Read	[63:32]
NDACT_P	Active Nodes Register	Base + 0x08	Read	[31:0]
FSRPDM_P	FIFO Status Register for PDM Data CDC	Base + 0x08	Read	[63:32]
ETHINTR_P	Interrupt Poll Register	Base + 0x10	Read	[31:0]
CLRCVD_P	Clear Interrupt Received Register	Base + 0x10	Read/Write	[63:32]
TX_ALL_STRT_P	Transaction start for all chains	Base + 0x18	Read/Write	[31:0]
IP_STATUS_P	IP Busy Status	Base + 0x20	Read/Write	[31:0]
AHBL_TOUT_P	AHBL Bus Timeout Count Register	Base + 0x28	Write	[31:0]
DTOUT_P	Node Response PDM Data Register	0x00108400 + 0x40	Read	[63:0]

Note: For PCIe based path, the PDM data goes through the AHBL interface to the DMACTR register that needs to be controlled.

Table 2.44. EtherControl Master Local Chain 1 Register Map (PCIe)

PCIe Register Name	Register Function	Base Address (0x00108100)	Access	Used Bits
TXSTR_P_1	Start Transaction Register	Base + 0x00	Read/Write	[31:0]
PKTHD_P_1	Packet Head Register	Base + 0x00	Read/Write	[63:32]
FRNUM_P_1	Frame Number Register	Base + 0x08	Read/Write	[31:0]
NDCNT_P_1	Number of Node Register	Base + 0x08	Read/Write	[63:32]
NDLN_P_1	Node Data Length Register	Base + 0x10	Read/Write	[31:0]
FSRREQD_P_1	FIFO Status Register Request Data	Base + 0x10	Read	[63:32]

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

PCIe Register Name	Register Function	Base Address (0x00108100)	Access	Used Bits
RQDT_P_1	Node Request Type Register	Base + 0x18	Read/Write	[31:0]
RQAD_P_1	Node Address Register	Base + 0x18	Read/Write	[63:32]
CRCNT_P_1	CRC Count Register	Base + 0x20	Read	[31:0]
INTR_P_1	Interrupt Info Register	Base + 0x20	Read	[63:32]
MTDT_P_1	Node Request Data Burst Register	Base + 0x28	Read/Write	[63:0]
MTRST_P_1	Node Motor Status Register	Base + 0x100 to 0x1FC	Read	[63:0]
DLY_P_1	Node Delay Register	Base + 0x200 to 0x2FC	Read	[63:0]

Table 2.45. EtherControl Master Local Chain 2 Register Map (PCIe)

PCIe Register Name	Register Function	Base Address (0x00108400)	Access	Used Bits
TXSTR_P_2	Start Transaction Register	Base + 0x00	Read/Write	[31:0]
PKTHD_P_2	Packet Head Register	Base + 0x00	Read/Write	[63:32]
FRNUM_P_2	Frame Number Register	Base + 0x08	Read/Write	[31:0]
NDCNT_P_2	Number of Node Register	Base + 0x08	Read/Write	[63:32]
NDLN_P_2	Node Data Length Register	Base + 0x10	Read/Write	[31:0]
FSRREQD_P_2	FIFO Status Register Request Data	Base + 0x10	Read/Write	[63:32]
RQDT_P_2	Node Request Type Register	Base + 0x18	Read/Write	[31:0]
RQAD_P_2	Node Address Register	Base + 0x18	Read/Write	[63:32]
CRCNT_P_2	CRC Count Register	Base + 0x20	Read	]31:0]
INTR_P_2	Interrupt Info Register	Base + 0x20	Read	[63:32]
MTDT_P_2	Node Request Data Burst Register	Base + 0x28	Read	[63:0]
MTRST_P_2	Node Motor Status Register	Base + 0x100 to 0x1FC	Read	[63:0]
DLY_P_2	Node Delay Register	Base + 0x200 to 0x2FC	Read	[63:0]

The Global register description is given below:

Table 2.46. DMA FIFO Enable/AHBL Disable Register

DMACTR_P				Base + 0x00
Byte	3	2	1	0
Name	DMACTR_P			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

DMACTR_P[0]: 0: DMA FIFO enabled, AHBL disabled | 1: DMA FIFO disabled, AHBL enabled

Table 2.47. PHY Link Status Register

PHLNK_P				Base + 0x00
Byte	7	6	5	4
Name	Physical Link Chain 2		Physical Chain 1	
Default	0	0	0	0
Access	R			

PHLNK_P[32]: 1: Main System PHY link established for chain 1 and 0: Main System PHY link not established for chain 1

PHLNK_P[47:33]: Each bit from bit 33 to bit 47 shows the link status of the respective nodes in chain 1

PHLNK_P[48]: 1: Main System PHY link established for chain 2 and 0: Main System PHY link not established for chain 2

PHLNK_P[63:49]: Each bit from bit 49to bit 63 shows the link status of the respective nodes in chain 2

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.48. Active Nodes Register

NDACT_P				Base + 0x08
Byte	3	2	1	0
Name	Active Node Chain 1+2	Active Node Chain 1+2	Active Node Chain 2	Active Node Chain 1
Default	0	0	0	0
Access	R			

NDACT_P[7:0]: Gives number of nodes actually connected physically to the system

Table 2.49. FIFO Status Register for PDM Data

FSRPDM_P				Base + 0x08
Byte	7	6	5	4
Name	FSRPDM_P			
Default	Reserved	Reserved	Reserved	0
Access	R			

FSRPDM_P[32]: Empty signal of RX FIFO

FSRPDM_P[33]: Full signal of RX FIFO

FSRPDM_P[34]: Overflow error of RX FIFO

FSRPDM_P[35]: Underflow error of RX FIFO

FSRPDM_P[36]: Reserved

FSRPDM_P[37]: Reserved

FSRPDM_P[38]: Reserved

FSRPDM_P[39]: Reserved

Table 2.50. Interrupt Polling Register

ETHINTR_P				Base + 0x10
Byte	3	2	1	0
Name	Ethernet Interrupt from Chain 2		Ethernet Interrupt from Chain 1	
Default	Reserved	0	Reserved	0
Access	R/W			

ETHINTR_P[0] : Interrupt bit from Chain 1

ETHINTR[7:1] : Reserved

ETHINTR[15:8] : Reserved

ETHINTR[16] : Interrupt bit from Chain 2

ETHINTR[31:17] : Reserved

Table 2.51. Clear Interrupt Received Register

CLRCVD_P				Base + 0x10
Byte	7	6	5	4
Name	CLRCVD			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

CLRCVD[39]: Received CLR bit from CPU

CLRCVD[39:33]: Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.52. Start Transaction in All Chains

TX_ALL_STRT_P				Base + 0x18
Byte	3	2	1	0
Name	TX_ALL_STRT_P			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

TX_ALL_STRT_P [0]: 1: Start the transaction | 0: No transaction

Table 2.53. IP Busy Register

IP_Busy_P				Base + 0x20
Byte	3	2	1	0
Name	AHBL_Bus_P			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

AHBL_BUSY_P[0]: 1 : AHBL Bus 0 Busy | 0: AHBL 0 bus Free (only for reading)

AHBL_BUSY_P[1]: 1 : AHBL Bus 1 Busy | 0: AHBL 1 bus Free

AHBL_BUSY_P[7:2]: Reserved

Table 2.54. AHBL Bus Timeout Count Register

AHBL_TOUT_P				Base + 0x28
Byte	3	2	1	0
Name	AHBL_TOUT_P			
Default	Reserved	Reserved	Reserved	0
Access	W			

AHBL_TOUT_P[31:0]: Sets the value of AHBL timeout count to free the bus

Table 2.55. Node Response PDM Data Register

DTOUT_P								0x00108400 + 0x40
Byte	7	6	5	4	3	2	1	0
Name	DTOUT_P							
Default	0	0	0	0	0	0	0	0
	R							

DTOUT_P[63:0]: PDM Data out from the nodes which is requested by the user

The local register 1 description is given below:

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.56. Chain 1 Start Transaction Register

TXSTR_P_1				Base + 0x00
Byte	3	2	1	0
Name	TXSTR_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

TXSTR_P_1[0]: 1: Start the transaction | 0: No transaction

Table 2.57. Chain 1 Packet Head Register

PKTHD_P_1				Base + 0x00
Byte	7	6	5	4
Name	PKTHD_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

PKTHD_P_1[32]: 1: User values are updated | 0: No update

Table 2.58. Chain 1 Frame Number Register

FRNUM_P_1				Base + 0x08
Byte	3	2	1	0
Name	FRNUM_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

FRNUM_P_1[7:0]: Frame number for the current frame

Table 2.59. Chain 1 Number of Node Register

NDCNT_P_1				Base + 0x08
Byte	7	6	5	4
Name	NDCNT_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDCNT_P_1[39:32]: Number of nodes configured by the user

Table 2.60. Chain 1 Node Data Length Register

NDLN_P_1				Base + 0x10
Byte	3	2	1	0
Name	NDLN_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDLN_1[7:0]: Data length of nodes to be configured by user

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.61. Chain 1 FIFO Status Register Request Data

FSRREQD_P_1				Base + 0x10
Byte	7	6	5	4
Name	FSRREQD_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R			

FSRREQD_P_1[32]: Overflow error of TX 1 FIFO

FSRREQD_P_1[33]: Underflow error of TX 1 FIFO

FSRREQD_P_1[34]: Empty signal of TX 1 FIFO

FSRREQD_P_1[35]: Full signal of TX 1 FIFO

FSRREQD_P_1[36]: Reserved

FSRREQD_P_1[37]: Reserved

FSRREQD_P_1[38]: Reserved

FSRREQD_P_1[39]: Reserved

Table 2.62. Chain 1 Node Request Type Register

RQDT_P_1				Base + 0x18
Byte	3	2	1	0
Name	RQDT_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQDT_1[7:0]: Type of data requested by the user

Table 2.63. Chain 1 Node Address Register

RQAD_P_1				Base + 0x18
Byte	3	2	1	0
Name	RQAD_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQAD_P_1[39:32]: Address requested by the user

Table 2.64. Chain 1 CRC Count Register

CRCNT_P_1				Base + 0x20
Byte	3	2	1	0
Name	CRCNT_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R			

CRCNT_P_1[7:0]: Gives the count of error generated by doing CRC on the data

Table 2.65. Chain 1 Interrupt Info Register

INTR_P_1				Base + 0x24
Byte	7	6	5	4
Name	INTR_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R			

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

INTR_P_1[63:32]: Gives the type of interrupt generated according to type of available data

0x01: Motor Configuration

0x02: Motor Status

0x03: PDM Data

0x04: Training Pkt

0x05: Pkt Head

0x06: Extended PDM Data

Table 2.66. Chain 1 FIFO Status Register Request Data

FSRREQD_1				Base + 0x28
Byte	3	2	1	0
Name	FSRREQD_1			
Default	Reserved	Reserved	Reserved	0
Access	R			

FSRREQD_1[0]: Overflow error of TX 1 FIFO

FSRREQD_1[1]: Underflow error of TX 1 FIFO

FSRREQD_1[2]: Empty signal of TX 1 FIFO

FSRREQD_1[3]: Full signal of TX 1 FIFO

FSRREQD_1[4]: Reserved

FSRREQD_1[5]: Reserved

FSRREQD_1[6]: Reserved

FSRREQD_1[7]: Reserved

Table 2.67. Chain 1 Node Request Burst Register

MTDT_P_1								Base + 0x28
Byte	7	6	5	4	3	2	1	0
Name	MTDT_P_1							
Default	0	0	0	0	0	0	0	0
	R/W							

MTDT_P_1[63:0]: Data to be send from the master to nodes by the user

Table 2.68. Chain 1 Node Motor Status Register

MTRST_P_1								Base + 0x100 – 0x1FC
Byte	7	6	5	4	3	2	1	0
Name	MTRST_P_1							
Default	0	0	0	0	0	0	0	0
	R							

Base + 0x100: Node 1 status

Base + 0x104: Node 2 status …….. (will progress like this for other nodes)

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.69. Chain 1 Node Delay Register

DLY_P_1								Base + 0x200 – 0x1FC
Byte	7	6	5	4	3	2	1	0
Name	DLY_P_1							
Default	0	0	0	0	0	0	0	0
	R							

Base + 0x200: Node 1 Delay

Base + 0x204: Node 2 Delay …….. (will progress like this for other nodes)

The local register 2 description is given below:

Table 2.70. Chain 2 Start Transaction Register

TXSTR_P_2				Base +0x00
Byte	3	2	1	0
Name	TXSTR_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

TXSTR_P_2[0]: 1: Start the transaction | 0: No transaction

Table 2.71. Chain 2 Packet Head Register

PKTHD_P_2				Base +0x00
Byte	7	6	5	4
Name	PKTHD_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

PKTHD_P_2[32]: 1: User values are updated | 0: No update

Table 2.72. Chain 2 Frame Number Register

FRNUM_P_2				Base +0x08
Byte	3	2	1	0
Name	FRNUM_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

FRNUM_P_2[7:0]: Frame number for the current frame

Table 2.73. Chain 2 Number of Node Register

NDCNT_P_2				Base +0x08
Byte	7	6	5	4
Name	NDCNT_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDCNT_P_2[39:32]: Number of nodes configured by the user

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.74. Chain 2 Node Data Length Register

NDLN_P_2				Base +0x10
Byte	3	2	1	0
Name	NDLN_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

NDLN_P_2[7:0]: Data length of nodes to be configured by the user

Table 2.75. Chain 2 FIFO Status Register Request Data

FSRREQD_P_2				Base +0x10
Byte	7	6	5	4
Name	FSRREQD_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R			

FSRREQD_P_2[32]: Overflow error of TX 1 FIFO

FSRREQD_P_2[33]: Underflow error of TX 1 FIFO

FSRREQD_P_2[34]: Empty signal of TX 1 FIFO

FSRREQD_P_2[35]: Full signal of TX 1 FIFO

FSRREQD_P_2[36]: Reserved

FSRREQD_P_2[37]: Reserved

FSRREQD_P_2[38]: Reserved

FSRREQD_P_2[39]: Reserved

Table 2.76. Chain 2 Node Request Type Register

RQDT_P_2				Base + 0x14
Byte	3	2	1	0
Name	RQDT_P_1			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQDT_P_2[7:0]: Type of data requested by the user

Table 2.77. Chain 2 Node Address Register

RQAD_P_2				Base + 0x14
Byte	7	6	5	4
Name	RQAD_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RQAD_P_2[39:32]: Address requested by the user

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.78. Chain 2 CRC Count Register

CRCNT_P_2				Base + 0x20
Byte	3	2	1	0
Name	CRCNT_P_2			
Default	Reserved	Reserved	Reserved	0
Access	R			

CRCNT_P_2[7:0]: Gives the count of error generated by doing CRC on the data

Table 2.79. Interrupt Info Register

INTR_P_2				Base + 0x20
Byte	7	6	5	4
Name	INTR_P_2			
Default	0	0	0	0
Access	R			

INTR_P_2[63:32]: Gives the type of interrupt generated according to type of available data

0x01: Motor Configuration

0x02: Motor Status

0x03: PDM Data

0x04: Training Pkt

0x05: Pkt Head

0x06: Extended PDM Data

Table 2.80. Chain 2 Node Request Burst Register

MTDT_P_2								Base + 0x28
Byte	7	6	5	4	3	2	1	0
Name	MTDT_P_1							
Default	0	0	0	0	0	0	0	0
Access	R/W							

MTDT_P_2[63:0]: Data to be send from the master to nodes by the user

Table 2.81. Chain 2 Node Motor Status Register

MTRST_P_2								Base + 0x100 – 0x1FC
Byte	7	6	5	4	3	2	1	0
Name	MTRST_P_2							
Default	0	0	0	0	0	0	0	0
Access	R							

Base + 0x100: Node 1 status

Base + 0x104: Node 2 status ……..(will progress like this for other nodes)

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.82. Chain 2 Node Delay Register

DLY_P_2								Base + 0x200 – 0x2FC
Byte	7	6	5	4	3	2	1	0
Name	DLY_P_2							
Default	0	0	0	0	0	0	0	0
Access	R							

Base + 0x200: Node 1 status

Base + 0x204: Node 2 status ……..(will progress like this for other nodes)

2.3.4. EtherControl Slave

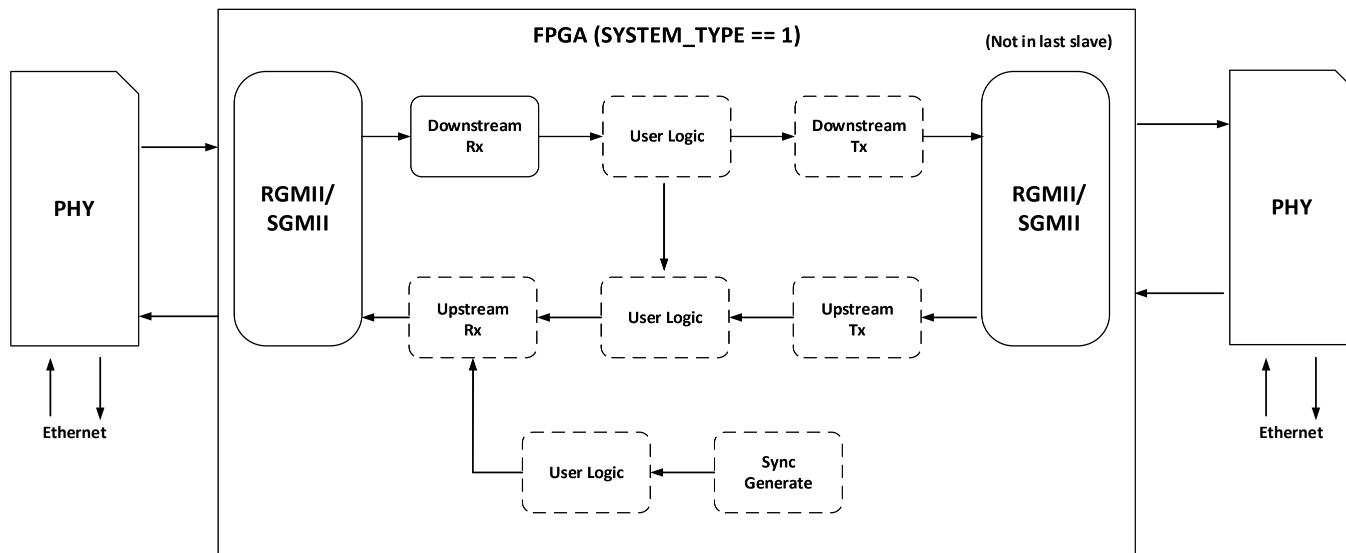


Figure 2.6. EtherControl Slave

Table 2.83. EtherControl Slave Register Map

EtherControl Register Name	Register Function	Address	Access
DMACTR	DMA control Register	Base + 0x00	Read/Write
FFDT	FIFO data Register	Base + 0x04	Write
RCMTR	Motor Status Register	Base + 0x08	Write
DMST	DMA Done Indication Register	Base + 0x0C	Write
INTST	Interrupt Status Register	Base + 0x10	Read
MTAD	Motor Config/Status Address Register or PDM Data Transfer Size Register	Base + 0x14	Read
MTDT	Motor Config Data Register	Base + 0x18	Read
FFER	FIFO error Register	Base + 0x1C	Read
CLRCVD	Clear Interrupt Received Register	Base + 0x3C	Read/Write

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.84. DMA Control Register

DMACTR				Base +0x00
Byte	3	2	1	0
Name	DMACTR			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

DMACTR[0]: 0: DMA FIFO enabled, AHBL disabled | 1: DMA FIFO disabled, AHBL enabled

Table 2.85. FIFO Data Register

FFDT				Base +0x04
Byte	3	2	1	0
Name	FFDT			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

FFDT[7:0]: Data incoming to the FIFO present in EtherControl Node System

Table 2.86. Motor Status Register

RCMTR				Base +0x08
Byte	3	2	1	0
Name	RCMTR			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

RCMTR[7:0]: Motor status

Table 2.87. DMA Done Indication Register

DMST				Base +0x0C
Byte	3	2	1	0
Name	DMST			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

DMST[7:0]: DMA done status

Table 2.88. Interrupt Status Register

INTST				Base +0x10
Byte	3	2	1	0
Name	INTST			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

INTST[7:0]: Gives the type of interrupt thrown by the node system

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.89. Motor Config/Status Address Register (or) PDM Data Transfer Size Register

MTAD				Base +0x14
Byte	3	2	1	0
Name	MTAD			
Default	0	0	0	0
Access	R			

MTAD[31:0]: Gives the config/status address or PDM data transfer size

Table 2.90. Motor Configuration Data Register

MCDR				Base +0x18
Byte	3	2	1	0
Name	MCDR			
Default	0	0	0	0
Access	R			

MCDR[31:0]: Gives the data available for motor configuration

Table 2.91. FIFO Error Register

FFER				Base +0x1C
Byte	3	2	1	0
Name	FFER			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

FFER[0]: Overflow error of FIFO

FFER[1]: Underflow error of FIFO

FFER[2]: Downstream sync signal for particular node

FFER[3]: Empty status of FIFO

FFER[4]: Full status of FIFO

FFER[5]: Reserved

FFER[6]: Reserved

FFER[7]: Reserved

Table 2.92. Clear Interrupt Received Register

CLRCVD				Base +0x3C
Byte	3	2	1	0
Name	CLRCVD			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

CLRCVD[0] : Received clr bit from CPU

CLRCVD[7:1] : Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.4. RISC-V to PCIe Bridge

This IP acts as a bridge between RISC-V (AHBL Bus 0) and PCIe (AHBL Bus 1). Information/status of AHBL bus 0 can be read by AHBL bus 1 and vice versa. The registers of AHBL bus 0 are of 32 bits whereas registers of AHBL Bus 1 are of 64 bits. The register map is shown in Table 2.93.

Table 2.93. RISC-V to PCIe Register Map

Bridge Register Name	Register Function	Base Address (0x00110000)	Access
RISC-V Registers			
REG_R_1	RISC-V Config Register 1	Base + 0x00	Read / Write
REG_R_2	RISC-V Config Register 2	Base + 0x04	Read / Write
FIFO_R	RISC-V FIFO Status Register	Base + 0x08	Read
BULK_R	RISC-V Bulk Data Register	Base + 0x20	Read /Write
PCIe_SYNC_1_R	PCIe Config Register 1 synced @ RISC-V clock	Base + 0x100	Read
PCIe_SYNC_2_R	PCIe Config Register 2 synced @ RISC-V clock	Base + 0x104	Read
PCIe_SYNC_BULK_R	PCIe bulk Data Register synced @ RISC-V clock	Base + 0x200	Read
PCIe Registers			
REG_P_1	PCIe Config Register 1	Base + 0x00	Read / Write
REG_P_2	PCIe Config Register 2	Base + 0x00	Read / Write
FIFO_P	PCIe FIFO Status Register	Base + 0x08	Read
BULK_P	PCIe Bulk Data Register	Base + 0x20	Read /Write
RISC_SYNC_1_P	RISC-V Config Register 1 synced @ PCIe clock	Base + 0x100	Read
RISC_SYNC_2_P	RISC-V Config Register 2 synced @ PCIe clock	Base + 0x100	Read
RISC_SYNC_BULK_P	RISC-V bulk Data Register synced @ PCIe clock	Base + 0x200	Read

Table 2.94. RISC-V Config Register 1

REG_R_1				**Base +0x00**
Byte	3	2	1	0
Name	REG_R_1			
Default	0	0	0	0
Access	R/W			

REG_R_1[31:0]: RISC-V Config Register 1

Table 2.95. RISC-V Config Register 2

REG_R_2				**Base +0x04**
Byte	3	2	1	0
Name	REG_R_2			
Default	0	0	0	0
Access	R/W			

REG_R_2[31:0]: RISC-V Config Register 2

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.96. FIFO Error Register

FIFO_R				Base +0x08
Byte	3	2	1	0
Name	FIFO_R			
Default	Reserved	Reserved	Reserved	0
Access	R			

FIFO_R[0]: RISC-V FIFO Full

FIFO_R[1]: PCIe FIFO Empty

FIFO_R[2]: Reserved

FIFO_R[3]: Reserved

FIFO_R[4]: Reserved

FIFO_R[5]: Reserved

FIFO_R[6]: Reserved

FIFO_R[7]: Reserved

Table 2.97. RISC-V Bulk Data Register

BULK_R				Base +0 x20
Byte	3	2	1	0
Name	BULK_R			
Default	0	0	0	0
Access	R/W			

BULK_R[31:0]: RISC-V Bulk Data Register

Table 2.98. PCIe Config Register 1 @ RISC-V Clock

PCIe_SYNC_1_R				Base + 0x100
Byte	3	2	1	0
Name	PCIe_SYNC_1_R			
Default	0	0	0	0
Access	R			

PCIe_SYNC_1_R[31:0]: PCIe Config 1 synced @ RISC-V clock

Table 2.99. PCIe Config Register 2 @ RISC-V Clock

PCIe_SYNC_2_R				Base + 0x104
Byte	3	2	1	0
Name	PCIe_SYNC_2_R			
Default	0	0	0	0
Access	R			

PCIe_SYNC_2_R[31:0]: PCIe Data 2 synced at RISC-V clock

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.100. PCIe Bulk Data Register @ RISC-V Clock

PCIe_SYNC_BULK_R				Base + 0x200
Byte	3	2	1	0
Name	PCIe_SYNC_BULK_R			
Default	0	0	0	0
Access	R			

PCIe_SYNC_BULK_R[31:0]: PCIe bulk Data synced at RISC-V clock

Table 2.101. PCIe Config Register 1

REG_P_1				Base + 0x00
Byte	3	2	1	0
Name	REG_P_1			
Default	0	0	0	0
Access	R/W			

REG_P_1[31:0]: PCIe Config Register 1

Table 2.102. PCIe Config Register 2

REG_P_2				Base + 0x00
Byte	7	6	5	4
Name	REG_P_2			
Default	0	0	0	0
Access	R/W			

REG_P_1[31:0]: PCIe Config Register 2

Table 2.103. PCIe FIFO Status Register

FIFO_P				Base + 0x08
Byte	3	2	1	0
Name	FIFO_P			
Default	Reserved	Reserved	Reserved	0
Access	R			

FIFO_P[0]: PCIe FIFO Full

FIFO_P[1]: RISC-V FIFO empty

FIFO_P[2]: Reserved

FIFO_P[3]: Reserved

FIFO_P[4]: Reserved

FIFO_P[5]: Reserved

FIFO_P[6]: Reserved

FIFO_P[7]: Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.104. PCIe Bulk Data Register

BULK_P								Base + 0x20
Byte	7	6	5	4	3	2	1	0
Name	BULK_P							
Default	0	0	0	0	0	0	0	0
Access	R/W							

BULK_P[63:0]: PCIe Bulk Data Register

Table 2.105. RISC-V Config Register 1 @ PCIe Clock

RISC_SYNC_1_P				Base + 0x100
Byte	3	2	1	0
Name	RISC_SYNC_1_P			
Default	0	0	0	0
Access	R			

RISC_SYNC_1_P[31:0]: RISC-V Config 1 synced @ PCIe clock

Table 2.106. RISC-V Config Register 2 @ PCIe Clock

RISC_SYNC_2_P				Base + 0x100
Byte	7	6	5	4
Name	RISC_SYNC_2_P			
Default	0	0	0	0
Access	R			

RISC_SYNC_2_P[63:32]: RISC-V Config 2 synced @ PCIe clock

Table 2.107. RISC-V Bulk Data Register @ PCIe Clock

RISC_SYNC_BULK_P								Base + 0x200
Byte	7	6	5	4	3	2	1	0
Name	RISC_SYNC_BULK_P							
Default	0	0	0	0	0	0	0	0
Access	R							

RISC_SYNC_BULK_P[63:0]: RISC-V bulk Data synced @ PCIe clock

2.5. FIFO DMA

This block has two FIFO interfaces, one is active when it is used in the main system to collect the PDM data received by the EtherControl master Bus 0. The other interface is active for node and has the pdm data from the motor control data collector block.

This block also has an AHBL slave and master interface. The register space for this block is as shown in Table 2.108.

The AHBL Slave interface is used to control DMA operations by external master (which is CPU) and AHBL master interface is used to perform for DMA operations.

More information about this IP is given FIFO DMA user guide.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.108. FIFO DMA Register Map

Register Name	Register Function	Address	Access
CNTR	FIFO DMA Control Register	Base + 0x00	Read/ Write
DEST_BASE_ADDR	Destination Base Address Register	Base + 0x04	Read/ Write
DEST_END_ADDR	Destination End Address Register	Base + 0x08	Read/ Write
PING_READY_ADDR	Ping Ready Address Register	Base + 0x10	Read/ Write
PONG_READY_ADDR	Pong Ready Address Register	Base + 0x14	Read/ Write
SET_PING_PONG_INDEX	Ping Pong Index Register	Base + 0x18	Read/ Write
STATUS	Write Status Register	Base + 0x0C	Read
STATUS_RD	Read Status Register	Base + 0x1C	Read

Table 2.109. FIFO DMA Control Registers

CNTR				Base +0x00
Byte	3	2	1	0
Name	CNTR			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

CNTR[0]: Used to control read operation.

CNTR[1]: Used to reset the destination register to destination base address.

CNTR[2-7]: Reserved

Table 2.110. DEST_BASE_ADDR Register

DEST_BASE_ADDR				Base +0x04
Byte	3	2	1	0
Name	DEST_BASE_ADDR			
Default	0	0	0	0
Access	R/W			

DEST_BASE_ADDR[31:0]: Base Address Location

Table 2.111. DEST_END_ADDR Register

DEST_END_ADDR				Base +0x08
Byte	3	2	1	0
Name	DEST_END_ADDR			
Default	0	0	0	0
Access	R/W			

DEST_END_ADDR[31:0]: END Address Location

Table 2.112. PING Ready Address Register

PING_RDY_ADDR				Base +0x10
Byte	3	2	1	0
Name	PING_RDY_ADDR			
Default	0	0	0	0
Access	R/W			

PING_RDY_ADDR[31:0]: PING Ready Address Location

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.113. PONG Ready Address Register

PONG_RDY_ADDR				Base +0x14
Byte	3	2	1	0
Name	PONG_RDY_ADDR			
Default	0	0	0	0
Access	R/W			

PONG_RDY_ADDR[31:0]: PONG Ready address Location

Table 2.114. PING PONG Index Register

SET_PING_PONG_INDEX				Base +0x18
Byte	3	2	1	0
Name	SET_PING_PONG_INDEX			
Default	Reserved	Reserved	Reserved	0
Access	R/W			

SET_PING_PONG_INDEX[0]: Setting index for PING PONG memory access.

1: Use PONG memory

0: Use PING memory

SET_PING_PONG_INDEX[1:7]: Reserved

Table 2.115. Write Status Register

STATUS				Base +0x0C
Byte	3	2	1	0
Name	STATUS			
Default	Reserved	Reserved	Reserved	0
Access	R			

STATUS[2:0] : Write Status

STATUS[3:31] : Reserved

Table 2.116. Read Status Register

STATUS_RD				Base +0x1C
Byte	3	2	1	0
Name	STATUS_RD			
Default	Reserved	Reserved	Reserved	0
Access	R			

STATUS_RD[2:0] : Read Status

STATUS_RD[3:31] : Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.6. PCIe DMA IP Design Details

The PCIe DMA IP is used to integrate in main system as defined in previous sections of this document. This IP is used to connect EtherControl master IP or main system to PCIe interface based high speed host. In this application, Linux-based PC is used as host, which has the software application to control main system's EtherControl IP operations and to control nodes through the main system.

This section only provides minimum details on the PCIe DMA IP required for integration and controlling. For more details, refer to the PCIe DMA IP user guide.

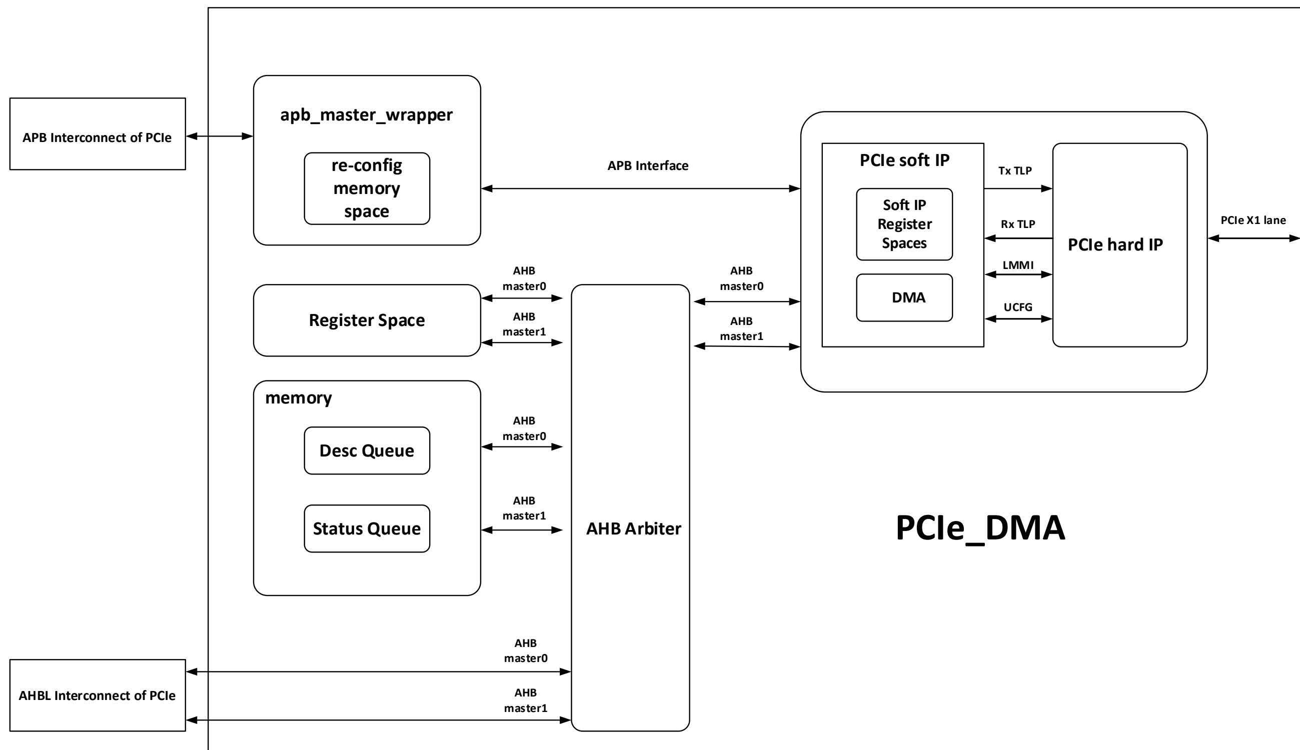


Figure 2.7. Top Level Architecture of PCIe DMA IP Design

Figure 2.7 shows the top-level architecture of FPGA design.

DMA support is and option provided by Lattice soft IP to enable more efficient data transfer when endpoint is acting as initiator or master. This feature is only available when AHB lite data interface is selected.

To transfer a data through DMA the core requires source address, destination address and transfer control i.e. length and direction of transfer, this information is collectively called descriptor.

To store the descriptor, two queues are implemented in a local memory: descriptor queue and status queue. When data transfer is completed or aborted, the status which contains the done flag, error flag, length of transfer, and data address offset is written into the status queue.

The description of each block of PCIe DMA design architecture is given below.

- AHB Arbiter
  This block selects the three blocks, which is APB master, system memory, and FIFO wrapper block, depending on the address received in the TLP (see memory segregation for address range of different block).
  The user can select the address range by modifying the parameter in *AHB_arbiter.v* file. AHB master0 port is being used for receiving side (Rx TLP) and AHB master1 port is being used for transmitting side (Tx TLP).
- APB master
  APB slave port is available to access the registers of soft IP or hard IP. To access the registers through software/driver, the user needs the APB master. This block is used to make the APB master interface. Initial reconfiguration of Soft IP and Hard IP is done through this APB master. A config space is implemented in the design which stores all the configuration values required for the PCIe IP.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- Desc Queue
  As this DMA implementation is based on the descriptors. So to store the descriptors a queue is implemented. The pointer of the descriptor queue is to be updated by the driver/software after/before writing the descriptors in the descriptors queue. Descriptors are fetched by the DMA soft IP in order serve the descriptors and corresponding read pointer is updated by the DMA core.
- Status Queue
  After one descriptor is served by the DMA Engine its status is to be stored somewhere, so to report the status of a transfer a status queue is implemented. The status of each descriptor or transfer is stored in this queue.
- Register Space
  A register space is implemented in the design to configure the DMA or to get the status of transfer and throughput achieved.

2.6.1. Descriptor Field Format

Table 2.117 lists the descriptor format.

Table 2.117. Descriptor Format

DW	DW name	Field name	Bit offset	Size	Description
0	desc_ctrl	length	0	13	Size of data transfer in bytes. (4096 bytes maximum)
		direction	13	1	Direction of transfer. 0 – AHB-Lite to PCIe 1 – PCIe to AHB-Lite
			14	10	Reserved
		desc_id	24	8	Optional descriptor ID. If the parameter EN_DESC_ID == “Enable” the Core adds this information in the Status entry.
1	desc_src	addr_offset	0	32	Source address/ offset
2	desc_dst	addr_offset	0	32	Destination address/offset
3	desc_hdr	requester_id	0	16	Requester ID to be used in TLP Header requester_id [7:0] – bus number[7:0] requester_id [10:8] – function number[2:0] requester_id [15:11] – device number[4:0]
		traffic_class	16	3	Traffic Class to be used in TLP Header
		use_requestor_id	19	1	When set, indicates that the requester_id field is valid and should be used in TLP header. Otherwise, the Core uses the captured configuration ID of function 0 as the default requester ID.
			20	12	Reserved

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.6.2. Status Field Format

Table 2.118 lists the status field format.

Table 2.118. Status Format

<table>
<tr><th>DW</th><th>DW name</th><th>Field name</th><th>Bit offset</th><th>Size</th><th>Description</th></tr>
<tr><td rowspan="8">1</td><td rowspan="8">stat_flag</td><td>done</td><td>0</td><td>1</td><td>If this bit is asserted, it indicates that the transfer has been completed</td></tr>
<tr><td>with_error</td><td>1</td><td>1</td><td>If this bit is asserted, it Indicates an error occurred during transfer.</td></tr>
<tr><td>aborted</td><td>2</td><td>1</td><td>If this bit is asserted, it indicates the transfer was terminated before it completes the specified length.</td></tr>
<tr><td rowspan="2">direction</td><td>3</td><td>1</td><td>Direction of transfer. 0 – AHB-Lite to PCIe 1 – PCIe to AHB-Lite</td></tr>
<tr><td>4</td><td>4</td><td>Reserved</td></tr>
<tr><td>desc_id</td><td>8</td><td>8</td><td>Optional descriptor ID. Available if the parameter EN_DESC_ID == “Enable”</td></tr>
<tr><td rowspan="2">length</td><td>16</td><td>13</td><td>Size of data transfer in bytes. (maximum of 4096 bytes)</td></tr>
<tr><td>29</td><td>3</td><td>Reserved</td></tr>
<tr><td>2</td><td>stat_buff</td><td>addr</td><td>0</td><td>32</td><td>Data Address. This is the local memory address where the data is stored (direction==1) or fetched (direction==0).</td></tr>
</table>

2.6.3. Triggering the DMA Operation

Before proceeding with the procedure below, ensure that the user is aware of the descriptor and status queue.

To trigger/start the DMA operation:

1. Write the Descriptors starting from address 0x1000 (note that one descriptor needs four DW (32-bit) space, see Table 2.117, if first descriptor is written at 0x1000 then next descriptor should be written at 0x1010 address).
2. Write the number of descriptors at 0x8.
3. Write 0x1 at address 0x1 to start the DMA operation

   Note: Soft reset is being asserted and de-asserted automatically when the DMA Done is asserted.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.6.4. PCIe DMA Register Space

2.6.4.1. FPGA Device Memory Segregation

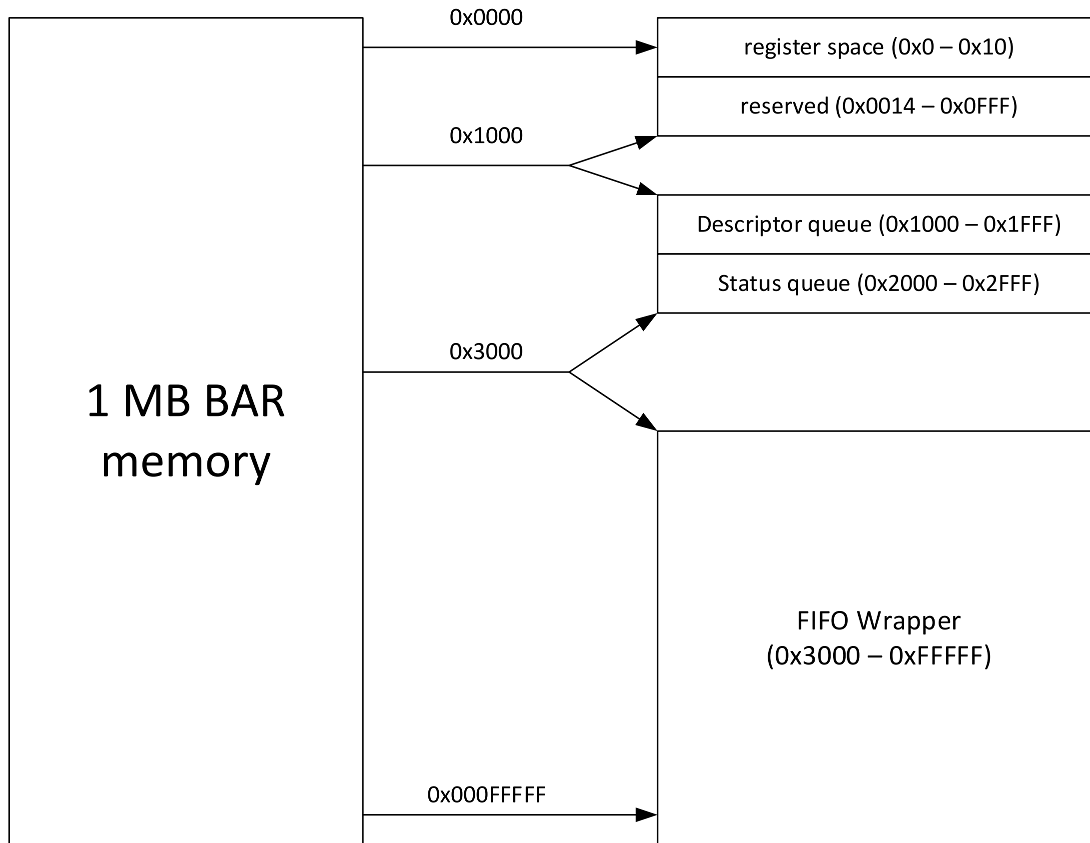


Figure 2.8. FPGA Device Memory Segregation

2.6.4.2. Register Address (0x0)

Table 2.119. Register Address (0x0)

31:6	7	6	5	4	3	2	1	0
Reserved	DMA Read operation done	Reserved	Reserved	Reserved	DMA aborted in one iteration	Error in one iteration	DMA write Operation done	Reserved
Read Only								
—	DMA read operation is completed	Reserved	Reserved	—	DMA iteration is aborted	DMA iteration is completed with error	DMA write iteration is completed	—

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.6.4.3. Register Address (0x4)

Table 2.120. Register Address (0x4)

31:0
Throughput Counter Value
Read Only
Multiply this counter value by 8 to get the total time (in ns) of one iteration

2.6.4.4. Register Address (0x8)

Table 2.121. Register Address (0x8)

31:8 Reserved	7:0
Reserved	Number of descriptors written in one iteration; valid values are between 1-255
Read Only	Write Only

2.6.4.5. Register Address (0xC)

Table 2.122. Register Address (0xC)

31:2	1	0
Reserved	Start DMA read operation	Start DMA write operation
	Write Only	
—	Write 1 to start the DMA read operation	Write 1 to start the DMA write operation

2.6.4.6. Register Address (0x10)

Table 2.123. Register Address (0x10)

31:0
Not Implemented
Read Only

2.6.4.7. Register Address (0x14)

Table 2.124. Register Address (0x14)

31:0
FPGA Version register; upper 8-bit [31:24] is indicating the date in decimal, next 8-bit [23:16] is indicating month in decimal, next 8-bit [15:8] is indicating the hour in 24 hour format, next 8-bit [7:0] is indicating the minute;
Read Only

2.6.4.8. Register Address (0x18)

Table 2.125. Register Address (0x18)

31:0
Not Implemented
Read Only

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.6.4.9. Register Address (0x1C)

Table 2.126. Register Address (0x1C)

31:0
Software Read /Write register, it can be used to read / write any information from software.
Read/Write

2.6.4.10. Register Address (0x20)

Table 2.127. Register Address (0x20)

31:0
DMA write size in DW, this indicates how much DWs users have to write in DMA write operation
Read/Write

2.6.4.11. Register Address (0x24)

Table 2.128. Register Address (0x24)

31:0
DMA write size in DW, this indicates how much DWs users have to write in DMA read operation
Read/Write

2.6.4.12. Register Address (0x28)

Table 2.129. Register Address (0x28)

31:1	0
Reserved	Data type
Read ONLY	0 means fixed pattern, 1 means incremental pattern.

2.7. SPI Flash Controller (QSPI Streamer)

This module is designed to stream data from external flash to FPGA using quad SPI data lines. It support max 100 MHz for LFD2NX and LFCL devices. It has prefetch buffer to enable cache feature for internal block of FPGA. Register space for this block is defined in register map section. This block does not have any configuration register to control it. There are basic settings (static configuration) which needs to be selected during build generation. As per register map complete SPI flash memory is directly accessible through AHBL interface. AHBL interface supports both code and data reading feature. This block does not support flash data write operation. This module is used in both main system and node system SOCs. This module only supports Micron, Macronix, and Winbond flash only.

In main system it is used to steam instructions and static data into RISC-V from external SPI flash and other parts of data section is stored in data ram.

At node end, it is used to stream instruction only into RISC-V from external SPI flash.

2.8. CNN Co-Processor Unit (CCU)

This block has an AHBL Master interface so that it can retrieve data directly from Data RAM or EtherControl block. This block can also fetch data from UART. For example, after the host PC has processed the training data and come up with a new set of weights, the CCU can get the new weights through UART.

This block also has an AHBL slave interface so that RISC-V CPU can control CNN Co-Processor Unit (CCU) through its registers.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.130. CNN Co-Processor Unit Registers

CCU Register Name	Register Function	Address	Access
PDMACR	CCU Control Register	Base + 0x00	Read/Write
PDMASR	CCU Status Register	Base + 0x04	Read
SIGSELR	Sign Select Configuration Register	Base + 0x08	Read/Write
INOFFSETCR	Input Offset Configuration Register	Base + 0x0C	Read/Write
FILOFFSETCR	Filter Offset Configuration Register	Base + 0x10	Read/Write
INDEPTHCR	Input Depth Configuration Register	Base + 0x14	Read/Write
INADDRCR	Input Data Address Configuration Register	Base + 0x18	Read/Write
FILADDRCR	Filter Data Address Configuration Register	Base + 0x1C	Read/Write
ACCOUTR	CCU Output Register	Base + 0x20	Read

Table 2.131. CNN Co-Processor unit control register

PDMACR		Base + 0x00
Bits	Others	0
Name	Unused	START
Default	Unused	0
Access	Unused	R/W

START: Setting 1’b1 to this register triggers the start of CCU process

Table 2.132. CNN Co-Processor Unit Register

PDMASR		Base + 0x04
Bits	Others	0
Name	Unused	DONE
Default	Unused	0
Access	Unused	R

DONE :

1’b0: CCU process is NOT completed

1’b1: CCU process is completed

Table 2.133. Sign Select Configuration Register

SIGSELR		Base + 0x08
Bits	Others	0
Name	Unused	SIGN_SEL
Default	Unused	0
Access	Unused	R/W

SIGN_SEL: Sign selector of input and filter values

1’b0: Unsigned (TinyML HPD)

1’b1: Signed (ours)

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.134. Input Offset Configuration Register

INOFFSETCR		Base + 0x0C
Bits	Others	8 : 0
Name	Unused	INPUT_OFFSET
Default	Unused	0
Access	Unused	R/W

INPUT_OFFSET: Input offset (2s complement - signed number [–256 ~ 255])

Table 2.135. Filter Offset Configuration Register

FILOFFSETCR		Base + 0x10
Bits	Others	8 : 0
Name	Unused	FILTER_OFFSET
Default	Unused	0
Access	Unused	R/W

FILTER_OFFSET: Filter offset (2s complement - signed number [–256 ~ 255])

Table 2.136. Filter Offset Configuration Register

FILOFFSETCR		Base + 0x10
Bits	Others	8 : 0
Name	Unused	FILTER_OFFSET
Default	Unused	0
Access	Unused	R/W

Table 2.137. Input Depth Configuration Register

INDEPTHCR		Base + 0x14
Bits	Others	9 : 0
Name	Unused	INPUT_DEPTH_BY_2_M1
Default	Unused	0
Access	Unused	R/W

INPUT_DEPTH_BY_2_M1: Input depth × 2 – 1 (0 ~ 1023); cover 512 depth

Table 2.138. Input Data Address Configuration Register

INADDRCR		Base + 0x18
Bits	Others	16 : 0
Name	Unused	INPUT_DATA_ADDR
Default	Unused	0
Access	Unused	R/W

INPUT_DATA_ADDR: Address to INPUT_DATA – start point of blob

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.139. Filter Data Address Configuration Register

FILADDRCR		Base + 0x1C
Bits	Others	16 : 0
Name	Unused	FILTER_DATA_ADDR
Default	Unused	0
Access	Unused	R/W

FILTER_DATA_ADDR: Address to FILTER_DATA – start point of filter

Table 2.140. CNN Co-Processor Unit Output Register

ACCOUTR		Base + 0x20
Bits	Others	31 : 0
Name	Unused	ACC_OUT
Default	Unused	0
Access	Unused	R

ACC_OUT: Accelerator output data

2.9. Motor Control and PDM Data Collector

This block has two AHBL slave interfaces that reside in the Node System. It provides direct control to motors through its logic and interface to power electronics. It also collects predictive maintenance data from the motors.

This block is used only in the Node Systems. The top level of the Node System has an AHBL wrapper which has two AHBL slave ports. Mainly it consists of Motor Control and Predictive Maintenance (MC/PDM) Registers, Motor Control logic, and PDM Data Collector as shown in Figure 2.9.

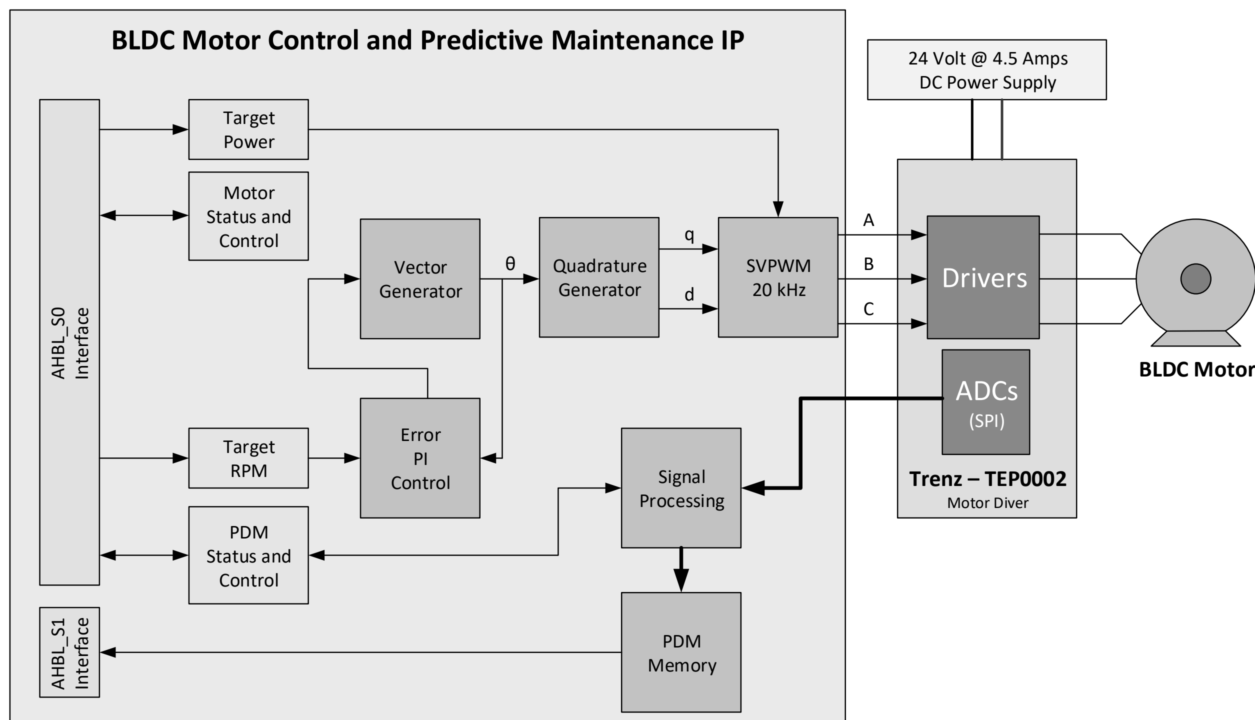


Figure 2.9. Motor Controller Interface with Motor

The Motor Control and PDM Registers interface with the AHB-L bus to configure, control, and monitor the Motor Control IP.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.141. Predictive Maintenance and Motor Control Registers

PDM/Motor Register Name	Register Function	Address	Access
MTRCR0	Motor Control Register 0 – Min RPM	Base + 0x00	Read/Write
MTRCR1	Motor Control Register 1 – Max RPM	Base + 0x04	Read/Write
MTRCR2	Motor Control Register 2 – RPM PI kI	Base + 0x08	Read/Write
MTRCR3	Motor Control Register 3 – RPM PI kP	Base + 0x0C	Read/Write
MTRCR4	Motor Control Register 4 – Torque PI kI	Base + 0x10	Read/Write
MTRCR5	Motor Control Register 5 – Torque PI kP	Base + 0x14	Read/Write
MTRCR6	Motor Control Register 6 – Sync Delay & Control	Base + 0x18	Read/Write
MTRCR7	Motor Control Register 7 – Target RPM	Base + 0x1C	Read/Write
MTRCR8	Motor Control Register 8 – Target Location	Base + 0x20	Read/Write
MTRCR9	Motor Control Register 9 – Location	Base + 0x24	Read/Write
MTRSR0	Motor Status Register 0 - RPM	Base + 0x28	Read
MTRSR1	Motor Status Register 1 – Limit SW & System Status	Base + 0x2C	Read
PDMCR0	Predictive Maintenance Control Register 0	Base + 0x30	Read/Write
PDMCR1	Predictive Maintenance Control Register 1	Base + 0x34	Read/Write
PDMSR	Predictive Maintenance Status Register	Base + 0x38	Read
PDMDDR	Predictive Maintenance ADC Data Register	Base + 0x3C	Read
PDMQDR	Predictive Maintenance ADC Data Register	Base + 0x40	Read
BRDSW	DIP and Push Button Switches	Base + 0x50	Read
BRDLEDS	LEDs and 7-Segment	Base + 0x54	Read/Write

Table 2.142. Motor Control 0 – Minimum RPM

MTRCR0				Base + 0x00
Byte	3	2	1	0
Name	PI_DELAY	MTRPOLES	MINRPM	
Default	0	0	0	0
Access	R/W			

MTRCR0[15:0]: MINRPM – Minimum RPM is the initial open loop motor starting RPM. Valid values are 10 to (2^{16} -1).

MTRCR0[23:16]: MTRPOLES : Number of motor stator poles. Valid values are 1 to 255.

MTRCR0[31:24]: PI_DELAY : Is the RPM PI update rate. Valid values are 1 to 255.

Table 2.143. Motor Control 1 – Maximum RPM

MTRCR1				Base + 0x04
Byte	3	2	1	0
Name	tbd		MAXRPM	
Default	0	0	0	0
Access	R/W			

MTRCR1[15:0]: MAXRPM – Maximum RPM is the upper limit RPM. Valid values are MINRPM to (2^{16} -1).

MTRCR1[31:16]: TBD

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.144. Motor Control 2 – RPM PI Control Loop Integrator Gain (kI)

MTRCR2				Base + 0x08
Byte	3	2	1	0
Name	RPMINT_MIN		RPMINTK	
Default	0	0	0	0
Access	R/W			

MTRCR2[15:0]: RPMINTK – Is the gain of the Integrator part of the RPM PI control loop. Valid values are 1 to (2^{16} -1).

MTRCR2[31:16]: RPMINT_MIN – Is the Integrator Anti-Windup Threshold. Valid values are 1 to (2^{16} -1).

Table 2.145. Motor Control 3 – RPM PI Control Loop Proportional Gain (kP)

MTRCR3				Base + 0x0C
Byte	3	2	1	0
Name	RPMINT_LIM		RPMPRPK	
Default	0	0	0	0
Access	R/W			

MTRCR3[15:0]: RPMPRPK – Is the gain of the Proportional part of the RPM PI control loop. Valid values are 1 to (2^{16} -1).

MTRCR3[31:16]: RPMINT_LIM – Is the Integrator Anti-Windup Clamp. Valid values are 1 to (2^{16} -1).

Table 2.146. Motor Control 4 – Torque PI Control Loop Integrator Gain (kI)

MTRCR4				Base + 0x10
Byte	3	2	1	0
Name	TRQINT_MIN		TRQINTK	
Default	0	0	0	0
Access	R/W			

MTRCR4[15:0]: TRQINTK – Is the gain of the Integrator part of the Torque PI control loop. Valid values are 1 to (2^{16} -1).

MTRCR4[31:16]: TRQINT_MIN – Is the Integrator Anti-Windup Threshold. Valid values are 1 to (2^{16} -1).

Table 2.147. Motor Control 5 – Torque PI Control Loop Proportional Gain (kP)

MTRCR5				Base + 0x14
Byte	3	2	1	0
Name	TRQINT_LIM		TRQPRPK	
Default	0	0	0	0
Access	R/W			

MTRCR5[15:0]: TRQPRPK – Motor Power or Torque PI Proportional Gain, depends on value of MTRCR6[2].

MTRCR6[2] = 0 : Motor Power - valid values are 0 to 1023.

MTRCR6[2] = 1 : Torque PI Proportional Gain - valid values are 1 to (216-1).[1]

MTRCR5[31:16]: TRQINT_LIM – Is the Integrator Anti-Windup Clamp. Valid values are 1 to (2^{16} -1).

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.148. Motor Control 6 – Synchronization Delay and Control

MTRCR6				Base + 0x18
Byte	3	2	1	0
Name	MTRCTRL	SYNCDLY		
Default	0	0	0	0
Access	R/W			

MTRCR6[21:0]: SYNCDLY[1] – Is the Motor control delay to compensate for Ethernet daisy-chain and processing delay. Used to synchronize starting and stopping of multiple motors simultaneously. Valid values are 0 to (2^{22} -1).

MTRCR6[23:22]: MTRCTRL_SYNDLYSF[1] – Sync Delay Scale Factor

00 = Disable Sync Delay (single motor control or sync not used).

01 = Sync Delay Units is nano-seconds (10-9)

10 = Reserved

11 = Reserved

MTRCR6[24]: RESET_PI – Reset the RPM PI Control

0 = Normal Operation

1 = Force the output to match the input (zero input values force the output to default of 120 rpm)

MTRCR6[25]: STOP – Hold the Motor in Position

0 = Normal Operation

1 = Stop the motor rotation

MTRCR6[26]: TRQPI_MODE – Torque Control Mode controls how MTRCR5[15:0] : TRQPRPK is used:

0 = Open Loop Mode – TRQPRPK value specifies Motor Power.

1 = Closed Loop Mode – TRQPRPK value specifies the gain of the Proportional part of the Torque PI control loop.[1]

MTRCR6[27]: ESTOP – Emergency Stop

0 = Normal Operation.

1 = Engage E-Brakes without sync delay or MTR_ENGAGE.[1]

MTRCR6[28]: ENABLE – Enable Motor Drivers

0 = Disable Motor Drivers

1 = Enable Motor Drivers

MTRCR6[29]: MTR_MODE

0 = RPM Control – Slew to target RPM and continue to run until stop or change in RPM target

1 = Location Control – Rotate specified number of degrees or turns then stop. Ramp up from zero to Max RPM, run as needed, then ramp back down to zero.[1]

MTRCR6[30]: DIRECTION

0 = Clockwise Rotation

1 = Counter-Clockwise Rotation

MTRCR6[31]: ENGAGE – Sync Signal to latch all Control Registers from AHBL clock domain (50–100 MHz) to Motor clock domain (24–25 MHz). Write to all other control registers first (including this one with this bit off). Write to this register (read-modify-write) to set this bit. It can also be used to synchronize multiple nodes.

0 = No Updates to Motor or PDM Control registers.

1 = Transfer all control register from AHBL holding registers to Motor PDM active registers.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.149. Motor Control Register 7 – Target RPM

MTRCR7				Base + 0x1C
Byte	3	2	1	0
Name	tbd		TRGRPM	
Default	0	0	0	0
Access	R/W			

MTRCR7[15:0]: TRGRPM – Target RPM. Valid values are 0 to (2^{16} -1).

MTRCR7 [31:16]: tbd

Table 2.150. Motor Control Register 8 – Target Location

MTRCR8				Base + 0x20
Byte	3	2	1	0
Name	TRGLOC			
Default	0	0	0	0
Access	R/W			

MTRCR8[31:0]: TRGLOC – Target Location. Valid values are -2,147,483,648 (-2^{32}) to 2,147,483,647 (2^{32} -1).[1]

Approximately 24.8 hours @ 4,000 RPM counting each degree.

Table 2.151. Motor Control Register 9 – Current Location

MTRCR9				Base + 0x24
Byte	3	2	1	0
Name	MTRLOC			
Default	0	0	0	0
Access	R			

MTRCR9[31:0]: MTRLOC – Motor Location. Valid values are -2,147,483,648 (-2^{32}) to 2,147,483,647 (2^{32} -1).[1]

Table 2.152. Motor Status Register 0 – RPM

MTRSR0				Base + 0x28
Byte	3	2	1	0
Name	tbd		MTRSTRPM	
Default	0	0	0	0
Access	R			

MTRSR0[15:0]: MTRSTRPM – Current Motor RPM. Valid values are 0 to (2^{16} -1).[1]

MTRSR0[31:16]: tbd.

Table 2.153. Motor Status Register 1

MTRSR1				Base + 0x2C
Byte	3	2	1	0
Name	MTRSR1			
Default	0	0	0	0
Access	R			

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

MTRSR1[0] : MTRSTR_MOV – Motor Moving

0 = Motor Stopped or coasting

1 = Motor Moving under control

MTRSR1[1]: ACCEL – Motor Accelerating

0 = Motor Not Accelerating

1 = Motor Accelerating

MTRSR1[2]: DECL - Motor Deaccelerating

0 = Motor Not Deaccelerating

1 = Motor Deaccelerating

MTRSR1[3]: RPM_LOCK - Motor At Target RPM

0 = Motor Not @ Target RPM

1 = Motor @ Target RPM

MTRSR1[4]: MTRSTR_STOP

0 = Motor not stopped

1 = Motor at zero RPM

MTRSR1[5]: MTRSTR_VLD_RPM

0 = RPM to Theta period calculation is still in process or invalid RPM request

1 = RPM to Theta period calculation is complete

MTRSR1[31:6]: tbd

Table 2.154. Predictive Maintenance Control Register 0

PDMCR0				Base + 0x30
Byte	3	2	1	0
Name	PDMCR0			
Default	0	0	0	0
Access	R/W			

PDMCR0[0]: START – Start PDM data collection.

0 = Collection not started

1 = Collection started

PDMCR0[1]: PKDTEN – PDM Normalization Peak Detect Enable

0 = PDM Peak Detect is Disabled

1 = PDM Peak Detect is Enabled

PDMCR0[2]: FOLDEN – Enable Single Folding of PDM data

0 = Single Fold disabled

1 = Single Fold enabled

PDMCR0[3]: 2FOLDEN – Enable Double Folding of PDM data. All PDM training data was captured using Double Folding.

0 = Double Folding disabled

1 = Double Folding enabled

PDMCR0[4]: CONTINUOUS – Collect data as long as START = 1.

0 = Fixed – Collect PDM data for set number of rotations

1 = Continuous – Collect PDM data continuously (counting rotations in status reg)

PDMCR0[5]: TBD

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

PDMCR0[6]: CALIB – ADC offset calibration

0 = Normal operation

1 = Calibrate ADC offsets (motor not running)

PDMCR0[7]: ADCH – ADC Channel Select for PDMDDR and PDMQDR registers

0 = ADC Channel = Amps

1 = ADC Channel = Volts

PDMCR0[15:8]: PREREVS – Pre Data Collection Revolutions

Number of Theta (Field Vector) revolutions to ignore before Data Collection. All PDM training data was captured using a value of 15.

PDMCR0[31:16]: DCREVS – Data Collection Revolutions

Theta (Field Vector) revolutions to capture PDM data (armature revs scale based on number of motor stator poles. The motor used for training has 4-poles – 16 Theta rotations equate to four motor shaft rotations). Valid values 1 to 65,536. All PDM training data was captured using 200 rotations.

Table 2.155. Predictive Maintenance Control Register 1

PDMCR1				Base + 0x34
Byte	3	2	1	0
Name	PDMCR1			
Default	0	0	0	0
Access	R/W			

PDMCR1: TBD

Table 2.156. Predictive Maintenance Status Register

PDMSR				Base + 0x38
Byte	3	2	1	0
Name	PDMSR			
Default	0	0	0	0
Access	R			

PDMSR[0]: DONE – PDM activity status

0 = PDM is not done with collecting data

1 = PDM is done with collecting data

PDMSR[1]: BUSY – PDM activity status

0 = PDM is not active

1 = PDM is busy collecting data

PDMSR[2]: CAL_DONE – ADC Offset Calibration status

0 = Offset calibration is not done

1 = Offset calibration is done

PDMSR[3]: READY – PDM Data Collector status

0 = Not ready to collect data

1 = Ready to collect data

PDMSR[15:4]: TBD

PDMSR[31:16]: PDMSR_ROT – Current count of Theta rotations PDM data has been collected for.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.157. Predictive Maintenance Current/Voltage Data Register

PDMDDR				Base + 0x3C
Byte	3	2	1	0
Name	ADC1		ADC0	
Default	0	0	0	0
Access	R			

PDMDDR[15:0]: ADC0 Voltage or Current reading Phase A[1]

PDMDDR[31:16]: ADC1 Voltage or Current reading Phase B[1]

Table 2.158. Predictive Maintenance Current/Voltage Data Register

PDMQDR				Base + 0x40
Byte	3	2	1	0
Name	ADC3		ADC2	
Default	0	0	0	0
Access	R			

PDMQDR[15:0]: ADC2 Voltage or Current reading Phase C[1]

PDMQDR[31:16]: ADC3 Voltage or Current reading of DC supply[1]

Table 2.159. Versa Board Switch Status Register

BRDSW				Base + 0x50
Byte	3	2	1	0
Name	TBD	PMOD2	DIPSW	PBSW
Default	0	0	0	0
Access	R			

PBSW[0]: SW5 – Pushbutton 2

0 = Switch active (pressed)

1 = Switch inactive

PBSW[1]: SW3 – Pushbutton 1

0 = Switch active (pressed)

1 = Switch inactive

PBSW[2]: SW2 – Pushbutton 3

0 = Switch active (pressed)

1 = Switch inactive

PBSW[7:3]: n/c - undefined

DIPSW[3:0]: SW10 – DIP Switch

0 = Switch closed

1 = Switch open

DIPSW[7:4]: n/c – undefined

PMOD2[0]: J8 Pin 1 I/O

PMOD2[1]: J8 Pin 2 I/O

PMOD2[2]: J8 Pin 3 I/O

PMOD2[3]: J8 Pin 4 I/O

PMOD2[4]: J8 Pin 7 I/O

PMOD2[5]: J8 Pin 8 I/O

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

PMOD2[6]: J8 Pin 9 I/O

PMOD2[7]: J8 Pin 10 I/O

Table 2.160. Versa Board LED & PMOD Control Register

BRDLEDS				**Base + 0x54**
Byte	3	2	1	0
Name	PMOD2DIR	PMOD2	7SEG	LED
Default	0xF	0xF	0xF	0xF
Access	R/W			

LED[0]: LED D18 – 0 = On, 1 = Off

LED[1]: LED D19 – 0 = On, 1 = Off

LED[2]: LED D20 – 0 = On, 1 = Off

LED[3]: LED D21 – 0 = On, 1 = Off

LED[4]: LED D22 – 0 = On, 1 = Off

LED[5]: LED D23 – 0 = On, 1 = Off

LED[6]: LED D24 – 0 = On, 1 = Off

LED[7]: LED D25 – 0 = On, 1 = Off

7SEG[0]: D36 Segment a – 0 = On, 1 = Off

7SEG[1]: D36 Segment b – 0 = On, 1 = Off

7SEG[2]: D36 Segment c – 0 = On, 1 = Off

7SEG[3]: D36 Segment d – 0 = On, 1 = Off

7SEG[4]: D36 Segment e – 0 = On, 1 = Off

7SEG[5]: D36 Segment f – 0 = On, 1 = Off

7SEG[6]: D36 Segment g – 0 = On, 1 = Off

7SEG[7]: D36 Segment dp – 0 = On, 1 = Off

PMOD2[0]: J8 Pin 1 I/O

PMOD2[1]: J8 Pin 2 I/O

PMOD2[2]: J8 Pin 3 I/O

PMOD2[3]: J8 Pin 4 I/O

PMOD2[4]: J8 Pin 7 I/O

PMOD2[5]: J8 Pin 8 I/O

PMOD2[6]: J8 Pin 9 I/O

PMOD2[7]: J8 Pin 10 I/O

PMOD2DIR[0]: J8 Pin 1 Direction – 0 = Input, 1 = Output

PMOD2DIR[1]: J8 Pin 2 Direction – 0 = Input, 1 = Output

PMOD2DIR[2]: J8 Pin 3 Direction – 0 = Input, 1 = Output

PMOD2DIR[3]: J8 Pin 4 Direction – 0 = Input, 1 = Output

PMOD2DIR[4]: J8 Pin 7 Direction – 0 = Input, 1 = Output

PMOD2DIR[5]: J8 Pin 8 Direction – 0 = Input, 1 = Output

PMOD2DIR[6]: J8 Pin 9 Direction – 0 = Input, 1 = Output

PMOD2DIR[7]: J8 Pin 10 Direction – 0 = Input, 1 = Output

Note:

1. Register function is not supported in the initial release.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.10. SPI Master IP Design Details

The Serial Peripheral Interface (SPI) is a high-speed synchronous, serial, full-duplex interface that allows a serial bitstream of configured length (8, 16, 24, and 32 bits) to be shifted into and out of the device at a programmed bit transfer rate. The Lattice SPI Master IP Core is normally used to communicate with external SPI slave devices such as display drivers, SPI EPROMS, and analog-to-digital converters.

The SPI Master IP is used to be integrated in node system SOC design as defined in node system top level architectural diagram. This IP can be controlled by C/C++ APIs of node system CPU to read/write data from/to certain SPI based peripheral/sensor. These C/C++ based APIs can be controlled by main system as well.

This section only provides minimum details on the SPI Master IP required for integration and controlling. For more details, refer SPI Master IP user guide.

2.10.1. Overview

The SPI Master IP Core allows the CPU inside the FPGA to communicate with multiple external SPI Slave devices. The data size of the SPI transaction can be configured to be 8, 16, 24, or 32 bits. This IP is designed to use an internal FIFO of configurable depth to minimize the host intervention during data transfer. SPI Master IP Core supports all SPI clocking modes – combinations of Clock Polarity (CPOL) and Clock Phase (CPHA) to match the settings of external devices.

The SPI Master IP provides a bridge between LMMI/AHB-Lite/APB and standard external SPI bus interfaces (functional diagram is shown in Figure 2.10). On the external, off-chip side the SPI Master Controller IP has a standard SPI bus interface. On the internal, on-chip side, the SPI Master Controller IP has LMMI/AHB-Lite/APB slave interface depending on the Interface attribute settings.

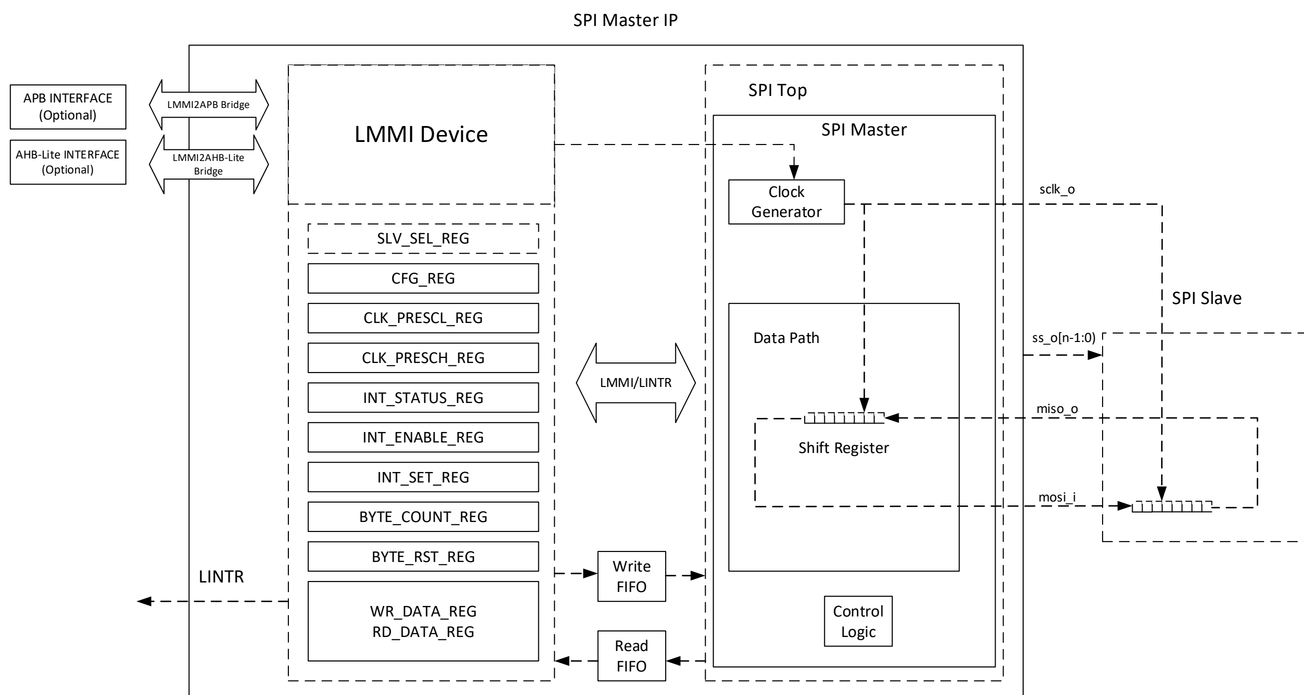


Figure 2.10. SPI Master IP Core Block Diagram

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.10.2. SPI Master Register Map

Table 2.161. SPI Master Register Map

Offset LMMI	Offset APB/AHBL	Register Name	Access Type	Description
0x0	0x00	WR_DATA_REG	WO	Write Data Register
0x0	0x00	RD_DATA_REG	RO	Read Data Register
0x1	0x04	SLV_SEL_REG	RW	Slave Select Register
0x2	0x08	CFG_REG	RW	Configuration Register
0x3	0x0C	CLK_PRESCL_REG	RW	Clock Pre-Scaler Low Register
0x4	0x10	CLK_PRESCH_REG	RW	Clock Pre-Scaler High Register
0x5	0x14	INT_STATUS_REG	RW1C	Interrupt Status Register
0x6	0x18	INT_ENABLE_REG	RW	Interrupt Enable Register
0x7	0x1C	INT_SET_REG	WO	Interrupt Set Register
0x8	0x20	WORD_CNT_REG	RO	Word Count Register
0x9	0x24	WORD_CNT_RST_REG	WO	Word Count Reset Register
0xA	0x28	TGT_WORD_CNT_REG	RW	Target Word Count Register
0xB	0x2C	FIFO_RST_REG	WO	FIFO Reset Register
0xC	0x30	SLV_SEL_POL_REG	RW	Slave Select Polarity Register
0xD	0x34	FIFO_STATUS_REG	RO	FIFO Status Register
0xE 0xF	0x38-0x3C	Reserved	RSVD	Reserved. Write access is ignored and 0 is returned on read access.

Table 2.161 lists the address map and specifies the registers available to the user. The offset of each register is dependent on the Interface attribute setting as follows:

- Interface selected to be LMMI: the offset increments by one
- Interface selected to be either AHBL or APB: the offset increments by four to allow easy interfacing with the Processor and System Buses. In this mode, each register is 32-bit wide wherein the upper unused bits are reserved and the lower bits are described in each register description.

Note:

1. For more details on the registers above, refer to the SPI Master IP Core – Lattice Radiant Software User Guide (FPGA-IPUG-02069).
2. The RD_DATA_REG and WR_DATA_REG share the same offset. Write access to this offset goes to WR_DATA_REG while read access goes to RD_DATA_REG.

2.10.3. Programming Flow

2.10.3.1. Initialization

The following SPI Master registers should be set properly before performing SPI transaction:

- SLV_SEL_REG – Set 1’b1 to the bit for the target slave. Set 1’b0 to other bits.
- SLV_SEL_POL_REG – may be configured once after reset since this setting is usually fixed.
- CLK_PRESCL_REG – Set based on target sclk_o frequency.
- CLK_PRESCH_REG – Set based on target sclk_o frequency.

The CPU needs to update the above registers only when SPI Master is switching to different slave device. This means there is no need to perform initialization again if the next transaction is for the currently selected slave device.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.10.3.2. Transmit/Receive Operation

The following are the recommended steps on performing the SPI transaction. This assumes that the module is not currently performing any operation.

1. Set the following CFG_REG fields according to the target Slave settings: cpha, cpol, ssnp and lsb_first. Set the only_write field based on the current transaction. If CFG_REG.only_write is 1'b0, SPI Master performs both transmit and receive operations (full-duplex). On the other hand, if CFG_REG.only_write is 1'b1, SPI Master IP Core performs Transmit operation only.
2. Set TGT_WORD_CNT_REG according to the number of words to transfer.
3. Reset WORD_CNT_REG by writing 8'hFF to Word Count Reset Register
4. Write data words to WR_DATA_REG, amounting to ≤ FIFO Depth.

   Optional: If interrupt mode is desired, enable target interrupts in INT_ENABLE_REG If number of words to transfer is ≤ FIFO Depth, set tr_cmp_en = 1'b1.If number of words to transfer is > FIFO Depth, set the following: tx_fifo_aempty_en = 1'b1 and tr_cmp_en = 1'b1. Other interrupts not specified above are disabled.
5. If total number of words to transfer > FIFO Depth, wait for Transmit FIFO Almost Empty Interrupt.
   a. If polling mode is desired, read INT_STATUS_REG until tx_fifo_aempty_int asserts.
   b. If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS_REG and check that tx_fifo_aempt_int is asserted.
6. Clear Transmit FIFO Almost Empty Interrupt by writing 1'b1 to INT_STATUS_REG.tx_fifo_aempty_int. Clearing all interrupts by writing 8'hFF to INT_STATUS_REG is also okay since the user is not interested in other interrupts for this recommended sequence.
7. Write data words to WR_DATA_REG, amounting to less than or equal to (FIFO Depth – TX FIFO Almost Empty Flag).
8. If CFG_REG.only_write = 1'b0, read all the data in RD_DATA_REG. It is expected that Receive FIFO has (FIFO Depth – TX FIFO Almost Empty Flag - 1) amount of data words. Read INT_STATUS_REG.rx_fifo_ready_int to check if RD_DATA_REG is already empty.
9. If there is remaining data to transfer, go back to Step 6. Note that you can read Word Count Register to determine the number of words already transferred in SPI interface.
10. Wait for Transfer Complete Interrupt.
    a. If polling mode is desired, read INT_STATUS_REG until tr_cmp_int asserts.
    b. If interrupt mode is desired, set INT_ENABLE_REG = 8'h80 then wait for interrupt signal to assert. Then read INT_STATUS_REG and check that tr_cmp_int is asserted.
11. Clear all interrupts by writing 8'hFF to INT_STATUS_REG.
12. If CFG_REG.ONLY_WRITE = 1'b0, read all the data in RD_DATA_REG. Read INT_STATUS_REG.rx_fifo_ready_int to check if RD_DATA_REG is already empty

2.11. I^2C Master IP Design Details

The I^2C (Inter-Integrated Circuit) bus is a simple, low-bandwidth, short-distance protocol. It is often seen in systems with peripheral devices that are accessed intermittently. It is commonly used in short-distance systems, where the number of traces on the board should be minimized. The device that initiates the transmission on the I^2C bus is commonly known as the Master, while the device being addressed is called the Slave.

The I^2C Master IP is used to be integrated in node system SOC design as defined in node system top level architectural diagram. This IP can be controlled by C/C++ APIs of node system CPU to read/write data from/to certain I^2C based peripheral/sensor. These C/C++ based APIs can be controlled by main system as well.

This section only provides minimum details of the I^2C Master IP required for the integration and controlling. Refer to the I^2C Master IP user guide for more details.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.11.1. Overview

The I²C Master IP Core accepts commands from LMMI/APB interface through the register programming. These commands are decoded into I²C read/write transactions to the external I²C slave device. The I²C bus transactions can be configured to be 1 to 256 bytes in length.

The I²C Master Controller can operate in interrupt or polling mode. This means that the CPU can choose to poll the I²C Master for a change in status at periodic intervals (Polling Mode) or wait to be interrupted by the I²C Master Controller when data needs to be read or written (Interrupt Mode).

Figure 2.11 shows the functional diagram of the I²C Master Controller.

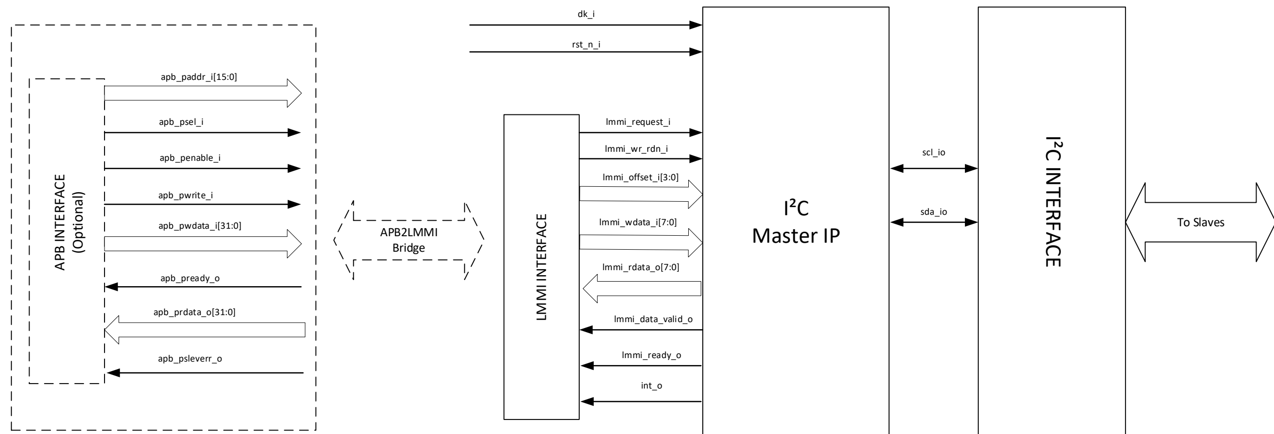


Figure 2.11. I²C Master IP Core Functional Diagram

2.11.2. I²C Master Register Map

The CPU can control the I²C Master IP Core by writing to and reading from the configuration registers. The I²C Master IP Core configuration registers can be performed at the run-time.

Table 2.162 lists the address map and specifies the registers available to you. The offset of each register is dependent on attribute APB Mode Enable setting as follows:

- APB Mode Enable is Unchecked – the offset increments by 1
- APB Mode Enable is Checked – the offset increments by 4 to allow easy interfacing with the Processor and System Buses. In this mode, each register is 32-bit wide wherein the upper bits [31:8] are reserved and the lower 8 bits [7:0] are described in the Programming Flow section.

The RD_DATA_REG and WR_DATA_REG share the same offset. Write access to this offset goes to WR_DATA_REG while read access goes to RD_DATA_REG.

Table 2.162. I²C Master IP Core Registers Summary

Offset LMMI	Offset APB/AHBL	Register Name	Access Type	Description
0x0	0x00	WR_DATA_REG	WO	Write Data Register
0x0	0x00	RD_DATA_REG	RO	Read Data Register
0x1	0x04	SLAVE_ADDRL_REG	RW	Slave Address Lower Register
0x2	0x08	SLAVE_ADDRH_REG	RW	Slave Address Higher Register
0x3	0x0C	CONTROL_REG	WO	Control Register
0x4	0x10	TGT_BYTE_CNT_REG	RW	Byte Count Register
0x5	0x14	MODE_REG	RW	Mode Register
0x6	0x18	CLK_PRESCL_REG	RW	Clock Prescaler Low Register
0x7	0x1C	INT_STATUS1_REG	RW1C	First Interrupt Status Register
0x8	0x20	INT_ENABLE1_REG	RO	First Interrupt Enable Register
0x9	0x24	INT_SET1_REG	WO	First Interrupt Set Register

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Offset LMMI	Offset APB/AHBL	Register Name	Access Type	Description
0xA	0x28	INT_STATUS2_REG	RW1C	Second Interrupt Status Register
0xB	0x2C	INT_ENABLE2_REG	RO	Second Interrupt Enable Register
0xC	0x30	INT_SET2_REG	WO	Second Interrupt Set Register
0xD	0x34	FIFO_STATUS_REG	RO	FIFO Status Register
0xE	0x38	SCL_TIMEOUT_REG	RW	SCL Timeout Register
0xF	0x3C	Reserved	RSVD	Reserved. Write access is ignored and 0 is returned on read access.

Note: RW1C (Writing 1’b1 on register bit clears the bit to 1’b0. Writing 1’b0 on register bit is ignored). For more details on the registers above, refer to I^{2}C Master IP Core – Lattice Radiant Software User Guide (FPGA-IPUG-02071).

2.11.3. Programming Flow

2.11.3.1. Initialization

The following I^{2}C Master registers can be set outside of the actual transaction sequence. These should be set properly before starting an I^{2}C transaction:

- SLAVE_ADDRL_REG, SLAVE_ADDRH_REG – Set the address of the target Slave Device
- CLK_PRESCL_REG – Set based on target scl_io frequency. The upper bits, MODE_REG. clk_presc_high are set during transaction because they are grouped with mode register.
- SCL_TIMEOUT_REG – Set to 8’h00 if the user does not want to check the SCL timeout or set to desired timeout value.
- INT_ENABLE2_REG – it is recommended to enable all interrupts in this register to check for error/unexpected event.

When accessing multiple devices, the SLAVE_ADDRL_REG or SLAVE_ADDRH_REG registers should be set prior to transaction.

2.11.3.2. Writing to the Slave Device

The following are the recommended steps for performing I^{2}C write transaction, this assumes that the module is not currently performing any operation and initialization is completed.

To perform I^{2}C write transaction:

1. Set the following MODE_REG fields according to the desired transfer mode: bus_speed_mode, addr_mode, ack_mode, clk_presc_high. Set the trx_mode field to 1’b0 for write transaction.
2. Set TGT_BYTE_CNT_REG according to the number of bytes to transfer.
3. Write data to WR_DATA_REG, amounting to ≤ FIFO Depth.
4. Set CONTROL_REG.start to 1’b1 to start the I^{2}C transaction.

   Optional: If interrupt mode is desired, Enable target interrupts in INT_ENABLE1_REG. If number of words to transfer is ≤ FIFO Depth, set tr_cmp_en = 1’b1 If number of words to transfer is > FIFO Depth, set the following: tx_fifo_aempty_en = 1’b1 and tr_cmp_en = 1’b1. Other interrupts in this register are disabled.
5. If total number of bytes to transfer > FIFO Depth, wait for Transmit FIFO Almost Empty Interrupt. If polling mode is desired, read INT_STATUS1_REG until tx_fifo_aempty_int asserts. If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS1_REG and check that tx_fifo_aempt_int is asserted. In both cases, read also INT_STATUS2_REG to ensure that the transfer is good. I^{2}C Master IP Core
6. Clear Transmit Buffer Almost Empty Interrupt by writing 1’b1 to INT_STATUS1_REG.tx_fifo_aempty_int. Clearing all interrupts in this register by writing 8’hFF to INT_STATUS1_REG is also okay since the user is not interested in other interrupts for this recommended sequence.
7. Write data to WR_DATA_REG, amounting to less than or equal to (FIFO Depth – TX FIFO Almost Empty Flag).
8. If there is remaining data to transfer, go back to Step 6.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

9. Wait for Transfer Complete Interrupt.
    a. If polling mode is desired, read INT_STATUS1_REG until tr_cmp_int asserts. If interrupt mode is desired, set INT_ENABLE1_REG = 8'h80 then wait for interrupt signal to assert. Read INT_STATUS1_REG and if tr_cmp_int is asserted.
10. Clear all interrupts by writing 8'hFF to INT_STATUS1_REG.

2.11.3.3. Reading from the Slave Device

The following are the recommended steps for performing I^{2}C read transaction, assuming that the module is currently not performing any operation and if initialization is completed.

To perform I^{2}C read transaction:

1. Set the following MODE_REG fields according to the desired transfer mode: bus_speed_mode, addr_mode, ack_mode, clk_presc_high. Set the trx_mode field to 1'b1 for read transaction.
2. Set TGT_BYTE_CNT_REG according to the number of bytes to transfer.
3. Set CONTROL_REG.start to 1'b1 to start the I^{2}C transaction.

   Optional: If interrupt mode is desired, Enable target interrupts in INT_ENABLE1_REG If number of words to transfer is ≤ FIFO Depth, set tr_cmp_en = 1'b1.
4. If number of words to transfer is > FIFO Depth, set the following: rx_fifo_afull_en = 1'b1 and tr_cmp_en = 1'b1. Other interrupts in this register are disabled.
5. If total number of bytes to receive > FIFO Depth, wait for Receive FIFO Almost Full Interrupt. If polling mode is desired, read INT_STATUS1_REG until rx_fifo_afull_int asserts. If interrupt mode is desired, wait for the interrupt signal to assert, and then read INT_STATUS1_REG and check if rx_fifo_afull_int is asserted. In both cases, read also INT_STATUS2_REG to ensure that the transfer is good.
6. Clear Receive FIFO Almost Full Interrupt by writing 1'b1 to INT_STATUS1_REG.rx_fifo_afull_int. Clearing all interrupts in this register by writing 8'hFF to INT_STATUS1_REG is also okay since the user is not interested in other interrupts for this recommended sequence.
7. Read all data from RD_DATA_REG. It is expected the amount of received data is less than or equal to (FIFO Depth – TX FIFO Almost Empty Flag). Read FIFO_STATUS_REG to confirm if Receive FIFO is emptied.
8. If there is remaining data to receive, go back to Step 5.
9. Wait for Transfer Complete Interrupt. If polling mode is desired, read INT_STATUS1_REG until tr_cmp_int asserts If interrupt mode is desired, set INT_ENABLE1_REG = 8'h80 and wait for the interrupt signal to assert. Read INT_STATUS1_REG and check that tr_cmp_int is asserted.
10. Clear all interrupts by writing 8'hFF to INT_STATUS1_REG.
11. Read all the remaining data from RD_DATA_REG.

2.12. UART IP Design Details

The Lattice Semiconductor UART (Universal Asynchronous Receiver/Transmitter) IP Core is designed for use in serial communication, supporting the RS-232.

The UART IP is used to be integrated in the node system SOC design as defined in node system top level architectural diagram. This IP can be controlled by C/C++ APIs of node system CPU to read/write data from/to certain UART/modbus based peripheral/sensor. These C/C++ based APIs can be controlled by main system as well.

This sections only provides minimum details of the UART IP required for the integration and controlling. Refer to the UART IP user guide for more details.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.12.1. Overview

The UART IP Core performs two main functions:

- Serial-to-parallel conversion on data characters received from an external UART device; and
- Parallel-to-serial conversion on data characters received from the Host located in the FPGA

The CPU can read the complete status of the UART at any time during the functional operation. Status information reported includes the type and condition of the transfer operations being performed by the UART IP Core, as well as any error conditions (parity, overrun, framing, or break interrupt).

The UART IP has implemented a processor-interrupt system similar to UART 16450. Interrupts can be programmed to your requirements, minimizing the computing required to handle the communications link. The UART IP currently does not implement the MODEM-control feature of UART 16450.

The registers of UART IP Core are accessed by the CPU (FPGA internal components) through an AMBA APB interface. The functional block diagram of UART IP Core is shown in Figure 2.12. The dashed lines in the figure are optional components/signals, which means they may not be available in the IP when disabled in the attribute.

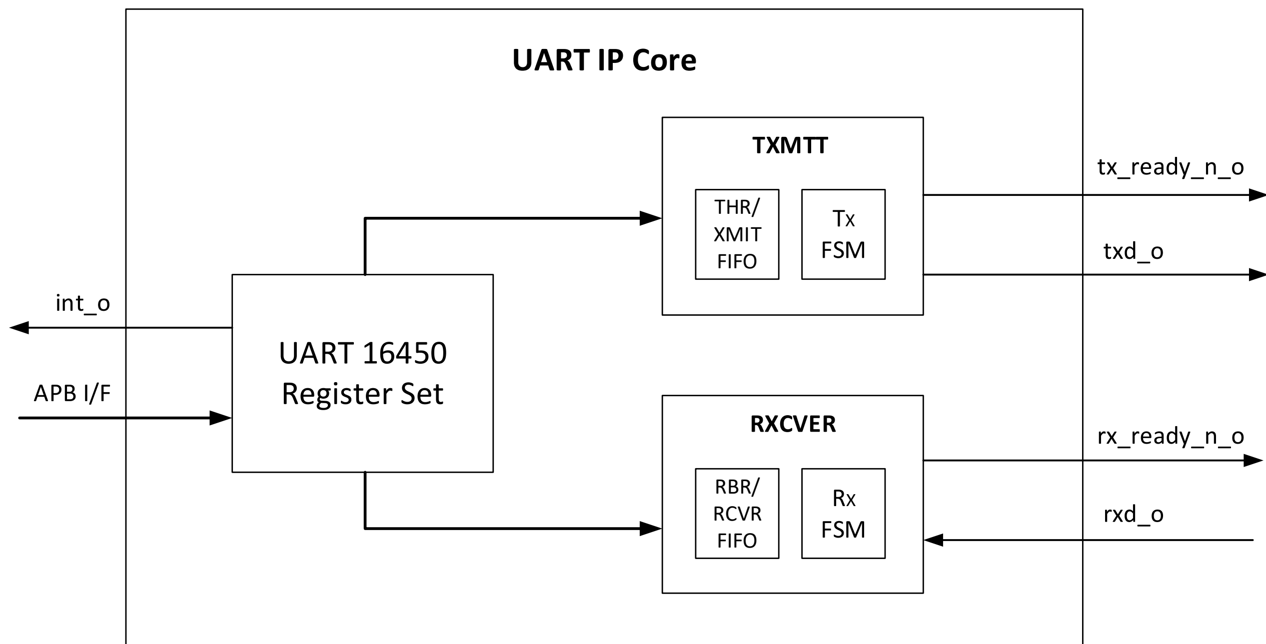


Figure 2.12. UART IP Core Functional Block Diagram

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.12.1.1. UART Register Description

The register address map, shown in Table 2.163, specifies the available IP Core registers. This is based on register set of UART 16450 but the offset address is changed to simplify the access to each registers. The offset of each register increments by four to allow easy interfacing with the Processor and System Buses. In this case, each register is 32-bit wide wherein the lower 8 bits are used and the upper 24 bits are unused. The unused bits are treated as reserved – write access is ignored and read access returns 0.

Table 2.163. UART Register Map

Offset	Register Name	Access Type	Description
0x00	RBR	RO	Receive Buffer Register
0x00	THR	WO	Transmitter Holding Register
0x04	IER	RW	Interrupt Enable Register
0x08	IIR	RO	Interrupt Identification Register
0x0C	LCR	RW	Line Control Register
0x10	Reserved	RSVD	Reserved
0x14	LSR	RO	Line Status Register
0x18-0x1C	Reserved	RSVD	Reserved
0x20	DLR_LSB	WO	Divisor Latch Register LSB
0x24	DLR_MSB	WO	Divisor Latch Register MSB
0x28-0x3C	Reserved	RSVD	Reserved

Note: Details of Registers is given in UART IP Core – Lattice Propel Builder User Guide (FPGA-IPUG-02105).

2.12.2. Programming Flow

2.12.2.1. Initialization

The following UART register fields should be set properly before performing UART transaction:

- Line Control Register – even_parity_sel, parity_en, stop_bit_ctrl, char_len_sel
- Divisor Latch Registers – divisor_msb, divisor_lsb

These should match the corresponding setting in the communicating UART for the serial transaction to be successful.

Note that reset values of these register fields are configurable during IP generation. Thus in some applications, initialization step is not necessary when attributes are properly set.

2.12.2.2. Transmit Operation

The following are the steps for transmitting character data through the UART IP Core. This is assuming that the IP is not performing transmit operation or at least the XMIT FIFO is empty.

Transmit Operation – Interrupt Mode

To perform transmit operation in interrupt mode:

1. Write the data to THR. In FIFO mode, user can write up to 16-character data.
2. Set IER.thre_int_en=1'b1 to enable Transmit Holding Register Empty interrupt.
3. Wait for Transmit Holding Register Empty interrupt to assert.
4. Wait for interrupt assertion and check that IIR[3:0]= 4'b0010.
5. If the user needs to send more characters, repeat Steps 1-3 until all characters are sent.
   When using interrupt, set IER.thre_int_en=1'b0 to disable the interrupt.

Transmit Operation – Polling Mode

To perform transmit operation in polling mode:

1. Write a data to THR. It is recommended not to enable FIFO for polling mode to save resource.
2. Read LSR until the thr_empty bit asserts.
3. If the user needs to send more characters, repeat Steps 1 and 2 until all characters are sent.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.12.2.3. Receive Operation

The following are the steps for the receiving character data through the UART IP Core. This is assuming that the IP core is not performing receive operation.

Receive Operation – Interrupt Mode

To perform receive operation in interrupt mode:

1. Enable the following interrupts:
    a. Received Data Available Interrupt (IER.rda_int_en=1’b1) – to notify the host that a data is received.
    b. Receiver Line Status interrupt (IER.rls_int_en=1’b1) – to notify the host of receive status such as error and break condition.
2. Wait for interrupt assertion and check that IIR[2:0]= 3’b100 (Receive Data Available). If Receiver Line Status Interrupt asserts (IIR[2:0]=3’b110), read the LSR to determine the cause.
3. If Receiver Line Status Interrupt does not occur, read the character data from RBR:
    a. If Receive Data Available Interrupt occurs, read a data from RBR.
    b. If Character Timeout Interrupt occurs, read LSR. If LSR.data_rdy=1’b1, read RBR.
4. Repeat Steps 2-3 until all expected data are received.

Receive Operation – Polling Mode

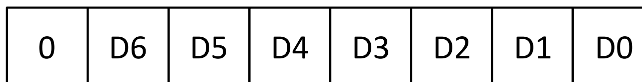
To perform receive operation in polling mode:

1. Read LSR until the thr_empty bit asserts. Also, check that no error status bits are asserted.
2. Read RBR if there is no error.
3. If the user needs to receive more characters, repeat Steps 1 and 2 until all characters are received.

2.12.2.4. Data Format

The character data written to THR and read from RBR is in little endian format as shown in Figure 2.13.

7-Bit Data



8-Bit Data

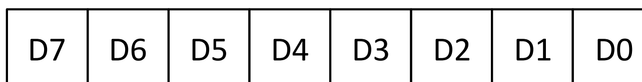


Figure 2.13. UART Data Format

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Resource Utilization

The resource utilization for the Main System is shown in Table 3.2.

Table 3.1. Main System Resource Utilization

Blocks	LUTs	EBRs	LRAMs	DSPs	Comments
RISC-V CPU	2462	2	—	—	—
ISR RAM	61	16	—	—	—
Data RAM (System Memory)	151	—	2	—	—
AHBL Interconnect 0	2041	—	—	—	—
AHBL Interconnect 1	218	—	—	—	—
FIFO DMA	927	16	—	—	—
EtherControl	34630	52	—	—	—
UART	257	—	—	—	—
SPI Flash Controller	232	1	—	—	—
AHBL2APB	141	0	—	—	—
APB Interconnect	13	0	—	—	—
CNN Coprocessor Unit (CCU)	1040	—	—	4	—
PCIe DMA	18973	40	—	—	—
PCIe RISCV Bridge	1117	16	—	—	—
Reset Sync	78	—	—	—	—
Top-level	1	—	—	—	—
Total	**62462**	**143**	**2**	**4**	**—**

The resource utilization for the Node System is shown in Table 3.2.

Table 3.2. Node System Resource Utilization

Blocks	LUTs	EBRs	LRAMs	DSP MULT	Comments
RISC-V CPU	2537	2	—	—	—
ISR RAM	51	16	—	—	—
Data RAM (System Memory)	155	0	2	—	—
AHBL Interconnect	1721	—	—	—	—
APB Interconnect	14	—	—	—	—
FIFO DMA	754	16	—	—	—
EtherControl	4209	11	—	—	—
SPI Flash Controller	229	1	—	—	—
AHBL2APB	148	—	—	—	—
Motor Control Data Collector	4152	17	—	15.5	—
UART	261	—	—	—	—
I^2C Master	585	—	—	—	—
SPI Master	398	—	—	—	—
Top-level	2	—	—	—	—
Total	**15216**	**63**	**2**	**15.5**	**—**

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4. Software APIs

4.1. Main System APIs

4.1.1. Tasks of the Main System

The Main System acts as an interface between the user interface and the node-system, which controls the motor IP. It communicates with the user interface and receives commands based on your requirements through UART. The commands are then sent to the nodes for configuration through EtherConnect. The Main System also enables the user interface to monitor various parameters of the motors. The system also receives commands from the GPIO switches attached on the board and sends these commands to the nodes for configuration through EtherConnect as well.

The tasks to be carried out by the Main System can be categorized as follows:

- System Initialization
  This API is used to configure the EtherControl and establish communication between the Main system and nodes. This takes place as soon as there is a power cycle or reset is pressed.
- Handle all the interrupts (GPIO, UART, EtherConnect) and respond to the interrupts by taking appropriate actions. Communication with the host system, Node System, and mechanical switches occur through interrupts and the Main System takes appropriate actions based on the interrupts caused. The priority order of all the interrupts is GPIO > UART > EtherConnect.
- Switch Configuration over GPIO
  User can Start, Stop, Accelerate, and Decelerate the motors with the help of switches provided. The Main System configures the node motor IP as per the switch configuration.
- Communicate with host system user interface over UART
  The host system user interface sends configuration data and status check commands to the Main System, and the Main System responds based on the command.
- Communicate with Node System and motor IP over EtherConnect
  As per the commands received by the Main System, it creates particular burst packets to send to the Node System, that the Node System then receives and implements them. This communication between the main and Node System happens over EtherConnect and at a given time, a maximum of 256 bytes can only be transmitted from either direction.

4.1.1.1. UART Commands

Table 4.1. Types of UART Commands

Command	Description	Remarks
Motor Config (Data Write)	The command specifies the register and value of the Motor IP to be updated.	High Priority Command. Can only be interrupted by GPIO configuration update.
Motor Status (Data Read)	The command specifies the status register of the Motor IP to be read from the Node System. The status data is stored at the Main System level, which the interface can read later.	High Priority Command. Can only be interrupted by GPIO configuration update.
Data Memory Reg Update (Data Write)	This command is used to update some registers that are not present at the Motor IP level but at the Main System level. For example, Node Select Reg, Node Addr Reg, Data Size Reg and others.	High Priority Command. Can only be interrupted by GPIO configuration update.
Data Memory Status Read (Data Read)	The command is called when the Main System needs to read the status data of any particular node, which it had asked the Main System to fetch earlier.	High Priority Command. Can only be interrupted by GPIO configuration update.
PDM Data Fetch (Data Read)	The PDM data fetch command asks the Main System to fetch the bulk maintenance data from the Node System and send it forward to the user interface system.	Can run in the background and has the least priority. It can be interrupted any time for any of the above four commands and GPIO switch configurations.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.1.1.2. GPIO Commands

Table 4.2. Types of GPIO Commands

Command	Description
Dip SW1 (0 to 1)	Start the motors
Dip SW2 (0 to 1)	Stop the motors
Dip SW3 (0 to 1)	Accelerate the motors
Dip SW4 (0 to 1)	Decelerate the motors

4.1.2. Key Functions

- Main () function
  int main (void)
  Upon a power on or a reset of the board, it is the job of the main function to initialize and configure the interrupts (GPIO, UART, EtherConnect). It also is tasked with the responsibility of calling the system_initialization API so that the communication between the Main System and the Node System can be established. Once the initialization is done, the Main System calls a power_on sequence function which configures the motor IP with default values.
  The ISR sets the flag, captures the data from UART and GPIO and stores it into global variables. After all the initialization tasks, the main function then monitors the flags and based on the data captured, calls appropriate functions.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

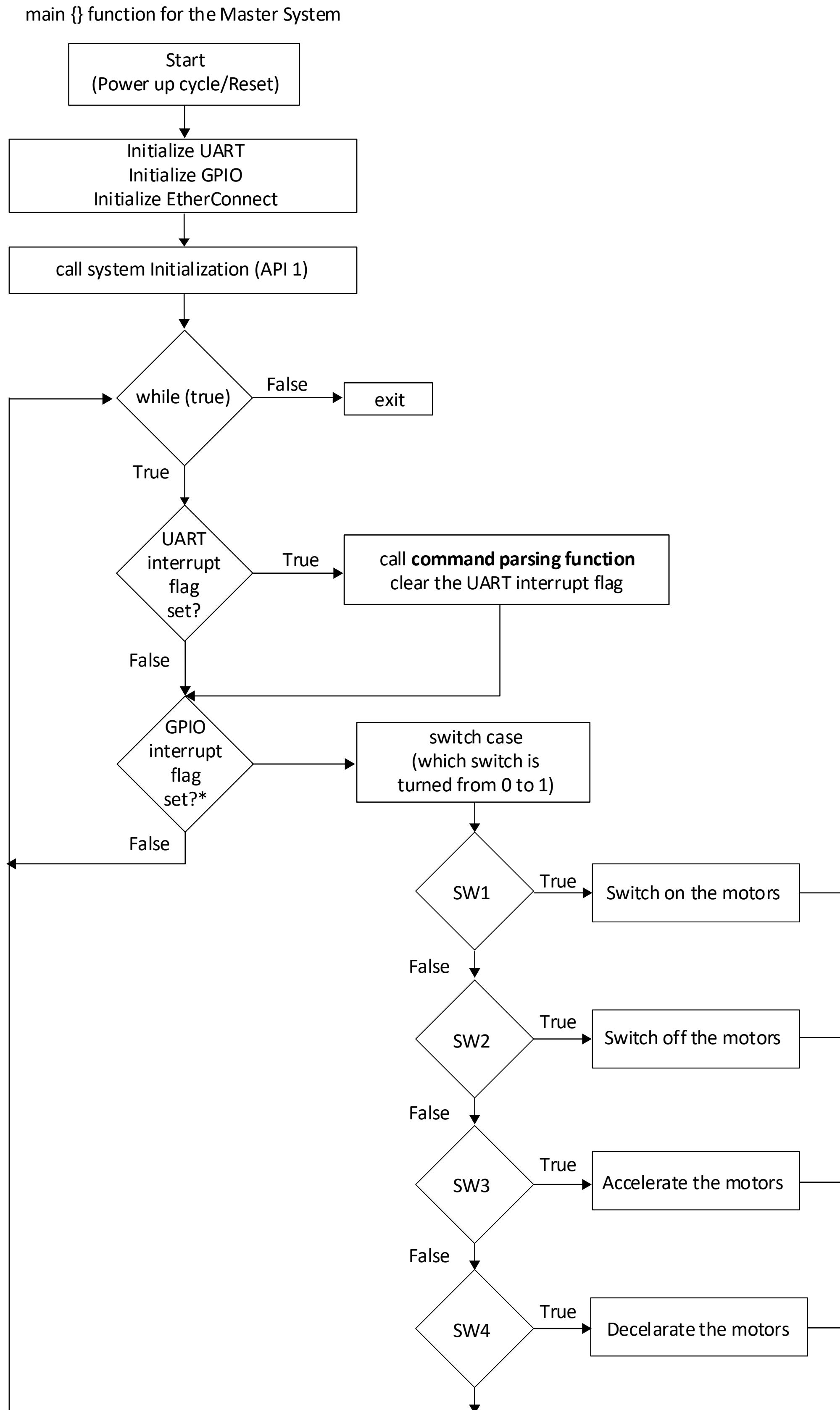


Figure 4.1. Main Function

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- ISR1_GPIO[1]
  *static void gpio_isr (void *ctx)*

  The interrupt service routine for GPIO is called every time there is a transition from 0 to 1 in the four switches that are connected to the Main System. After the 0 to 1 interrupt (rising edge), the gpio_interrupt_flag is set. In a given global variable, it stores a value which indicates the gpio switch that is triggered. The ISR then can acknowledges the interrupt and returns an integer value. The IRQ value for GPIO is IRQ1. This ISR is present in the gpio.c file.
- ISR2_UART
  *static void uart_isr (void *ctx)*

  The interrupt service routine for UART is called every time it receives a byte in its buffer module. The maximum size of the buffer is 16 bytes. The ISR then validates the command by checking the total size of data received and the data itself. It sets the uart_interrupt_flag once, only after confirmation. The data captured, if valid, is stored into a global variable so that it can be accessed later by the command parsing function. The IRQ value for UART is IRQ2. The ISR then acknowledges the interrupt and returns an integer value. This ISR is present in the uart.c file.
- ISR3_EtherConnect
  *static void etherConnect_isr (void *ctx)*

  The primary function of the EtherConnect ISR function is to set the interrupt flag, acknowledge the interrupt, and return a value. The EtherConnect interrupt is used as an acknowledgement of the completion of a single transaction of a command sent by the Main System to the Node System. The IRQ value for EtherConnect is IRQ3.
- System Initialisation API
  int system_initialisation (void)

  This API is present in the main.c file. It does not take any parameter and returns an integer value. It returns 0 if everything is successfully completed or a – 1 if there is an error.

  This API is used to establish communication between the Main System and the Node System. It enables the DMA FIFO module and sends 10 broadcast packets to detect the number of nodes available and active in the whole setup. By reading the PHY Link Status register, it affirms whether the communication is established or not, and accordingly, turns ON the Main System LEDs. This API then sends three training packets and one normal packet to the Node System through the EtherConnect in order to affirm the connection establishment with the Node System.
- Motor Configuration API
  int motor_config_api(uint32_t address, uint32_t data, uint32_t multi)

  This API is present in the main.c file. It needs three parameters namely:
  - address: signifies a register in the Motor Control IP
  - data: what needs to be written in that register
  - multi: data to be transmitted on multiple chains or selected chain only

  It returns the following integer values:
  - 0: if everything is correct
  - –1: if there was any error

  The API is called when there is a requirement to configure a register in the Motor Control IP of the Node System. This occurs in two cases:
  - when there is an ON switch on any GPIO
  - when the command parsing function decodes a UART command wishing to configure the motors

  The API creates burst packets which are sent to the Node System over EtherConnect. The header in the burst packet indicates that a particular packet is for Motor Configuration and for which nodes this packet is intended. Once the burst packet is written in a FIFO module, it is sent to the Node System by a trigger of 1 to 0 signal in a Start Transaction Register. After the Node System completes the task successfully, the Main System receives an interrupt and validates the value of the interrupt info register. Upon the confirmation of the value of the interrupt info register, this API returns a 0 value or a –1 if there is an error.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- Motor Status API
  int motor_status_api(uint32_t address, uint32_t multi)
  This API is present in the main.c file. It needs one parameter:
  - address: signifies a register in the Motor Control IP
  - multi: etherconnet packet to be transmitted on multiple chains or selected chain only

  It returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  The API is called when there is a requirement to read a register in the Motor Control IP of the Node System. This happens when the command parsing function decodes a UART command wishing to read the status of the motors. The API creates burst packets which are sent to the Node System over EtherConnect. The header in the burst packet indicates that a particular packet is for Motor Status Read and for which nodes this packet intended. Once the burst packet is written in a FIFO module, it is sent to the Node System by a trigger of 1 to 0 signal in a *Start Transaction Register*. After the Node System has taken appropriate actions successfully, the Main System receives an interrupt and it validates the value of the interrupt info register. Upon the confirmation of the value of the interrupt info register, this API returns a 0 value or a –1 if there is an error.
- PDM Data Fetch API
  int pdm_data_fetch_api(uint32_t total_size, uint32_t node_addr)
  The API is present in the main.c file. It needs one parameter:
  - total_size: the size of the PDM data required from user interface
  - node_addr: node select value sent in packet

  It returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  The API is called when there is a requirement to read a bulk maintenance data from the Motor Control IP of the Node System. This happens when the command parsing function decodes a UART command wishing to read the bulk PDM data.
  The maximum data that can be transferred in a single transaction from node to Main System is 256 bytes. Therefore, if the total_size is larger than 256 bytes, chunks of 256 bytes are requested one by one until the total_size requirement is met.
  This API first configures the DMA register by writing the destination base and destination end address in specific registers. The API creates burst packets which are sent to the Node System over EtherConnect. The header in the burst packet indicates that a particular packet is for PDM Data Fetch and for which particular node this packet intended. Once the burst packet is written in a FIFO module, it is sent to the Node System by a trigger of 1 to 0 signal in a *Start Transaction Register*. After the Node System completes the task successfully, the Main System receives an EtherConnect interrupt and it validates the value of the interrupt info register. The value of the DMA status register is to be validated as confirmation of the same. A successful validation signifies that a single chunk of data is successfully written into the Main System memory. This process is repeated until all the chunks are received by the Main System. A final EtherConnect interrupt is then received from the Node System signifying the completion of the PDM data fetch command for the total_size. Upon confirmation of the value of the interrupt info register, this API returns with 0 value.

- PDM bulk Data Fetch API
  int pdm_bulk_data_fetch_api (uint32_t total_size, uint32_t node_addr)
  The API is present in the main.c file. It needs two parameter:
  - total_size: the size of the PDM data required from user interface
  - node_addr : node select value sent in packet

  It returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  The API is called when there is a requirement to read a bulk maintenance data from the Motor Control IP of the Node System. This happens when the command parsing function decodes a UART command wishing to read the bulk PDM data.
  This API is extended version of PDM Data Fetch API, as total size of data fetch depends on number of active nodes present in that chain.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- UART Command Parsing Function
  *int uart_cmd_parse_func (char *cmd_packet)*
  The API is present in the main.c file. It needs 1 parameter namely:
  - *cmd_packet:	buffer pointer to the command received from UART

  It returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  This function is used to decode the incoming UART command packet and take actions accordingly. This function checks the validity of the command, and based on the UART commands in Table 4.1, it categorizes the command into one of the five different types. According to the decoded command, it calls/performs the corresponding APIs/actions. The APIs called or the actions performed are expected to return 0 int value in case of successful completion. If a –1 is returned, the operation is considered erroneous. This function then calls the UART response function passing over the error/no-error value so that the user interface can be informed of the same. After getting a successful response from the UART response function, this function returns to the main function from where it is called, completing the cycle.
- UART Response Function
  int uart_response_func (int function_type, uint32_t error_value, uint32_t data)
  The API is present in the main.c file. It needs two parameters namely:
  - function_type: signifies which type of response needs to be sent
  - error_value: indicates if there is an error or everything is successfully completed

  A third parameter, which is *data*, is only used in the case of Data Memory Status Read (Data Read) command. It returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  This function is used to send a response packet to the user interface over UART. The UART Command Parsing calls the UART Response after the required actions are executed. While the response to Motor Config Write, Motor Status, and Data Memory Reg Update commands only returns an acknowledgement of either a successful or unsuccessful execution, the Data Memory Status Read and PDM Data Fetch commands have data in the response packet from the 12th byte. The Data Memory Status Read writes the data variable sent as a parameter to the function while the PDM Data Response reads memory registers from the base address. It writes the response packet to the UART buffers and returns to the command parsing function.
- Power UP Sequence Function
  int power_up_sequence (void)
  The API is present in the main.c file. It does not require any parameters. It returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  After a power cycle or reset, the main function calls the Power UP Sequence Function to initialize the Motor control IP registers with default values. The Motor Config API is called to implement this requirement. The Power UP Sequence Function also calls the Motor Status API to read a particular register, and afterwards, calls the PDM Data Fetch API with a default size of 64 bytes.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.2. Node System APIs

4.2.1. Tasks of the Node System

The Node System acts as a way to control the Motor IP and get its status as commanded by the Main System. It communicates with the Main System by receiving commands through EtherConnect. It performs the actions and responds to the Main System with interrupts as acknowledgement for the tasks executed.

The tasks to be carried out by a master system can be categorized as follows:

- Communicate with the master system over EtherConnect
  As per the commands sent by the Main System, the Node System is supposed to either configure the motor, share the motor status or share the bulk PDM data
- Perform key functions

4.2.2. Key Functions

- Main () function
  int main (void)
  Upon a power on or a reset of the board, it is the job of the main function to initialize and configure the interrupts (EtherConnect, UART).
  The main function then waits for the *ether_interrupt_flag* to get high. The EtherConnect ISR sets the flag, *ether_interrupt_flag* when a command is received from the Main System. When the main function finds that the flag is set, it reads the INTERRUPT STATUS register to decode which command is received. Based on the value of this register, the main function calls the appropriate functions.
- Node Perpherials init
  u08 general_init (void)
  Upon a power on or a reset of the board, it is the job of the main function to initialize and configure the interrupts for UART, EtherConnect. It also initializes Modbus, SPI, and I^{2}C protocols.
- ISR1_EtherConnect
  *static void etherConnect_isr (void *ctx)*
  The primary function of the EtherConnect ISR function is to set the interrupt flag, acknowledge/clear the interrupt and return an integer value. The EtherConnect interrupts are used as indicators of the receipt of a command sent by the Main System to the Node System. The IRQ value for EtherConnect is 0.
- Node Configuration API
  int node_config_api(void)
  The API is present in the main.c file. It does not require any parameter.
  It returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  The API is called when the main function receives a Node Config command in its Interrupt Status Register. This API reads the NODE ADDRESS register. This register contains an *address* of the peripheral (I^{2}C, Modbus, SPI, and Motor IP) which is supposed to be configured. Next, the NODE CONFIG DATA register is read. This register has the configuration *data*. This *data* is then written into the *address*. If there is a read or write error, the API returns a –1 value. Once completed, the API returns a 0 value.
- Node Status API
  int node_status_api(void)
  The API is present in the main.c file. It does not require any parameter. This returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

The API is called whenever the main function receives a Node Status command in its Interrupt Status Register. This API reads the NODE ADDRESS register. This register contains an address of the Node peripherial (Modbus, SPI, I^{2}C, Motor IP) whose configuration value is supposed to be read. This address is then read and stored in a local variable data. This data is then written into the NODE STATUS register. If there is any read or write error, the API sends –1 value back. If everything goes okay, the API returns 0 value.

- PDM Data Fetch API
  int pdm_data_fetch_api(void)
  The API is present in the main.c file. It does not require any parameter. This returns the following integer values:
  - 0: if all tasks are successfully completed
  - –1: if there is an error

  The API first reads the *size* of PDM data required from the PDM ADDRESS register. It then writes the *base address* value and the *end address (base address + size)* value at the designated registers in the FIFO DMA Module. It then enables the FIFO DMA module by sending writing 0x00000003 first and then 0x00000000 to the FIFO DMA CONTROL register. Once done, it polls the DMA STATUS register for the indication of completion of the PDM data fetch. Once it receives the done value, it sets the DMA DONE INDICATE register. If there is any read or write error, the API sends –1 value back. If everything goes okay, the API returns 0 value.
- Node Peripherial APIs
  - I^{2}C Master
    The following are the I^{2}C BSP functions used in the main.c file for writing and reading the I^{2}C slave data:
    - uint8_t i2c_master_write(struct i2cm_instance × this_i2cm, uint16_t address,uint8_t data_size, uint8_t × data_buffer)
    - *uint8_t i2c_master_read(struct i2cm_instance × this_i2cm, uint16_t address,uint8_t read_length, uint8_t × data_buffer)*
  - SPI Master
    The following are the SPI BSP functions used in the main.c file for writing and reading SPI slave data:
    - *uint8_t spi_master_write(struct spim_instance × this_spim,uint8_t data_size, uint8_t × data_buffer)*
    - *uint8_t spi_master_read(struct spim_instance × this_spim,uint8_t read_length, uint8_t × data_buffer)*
  - Modbus RTU Master
    The following are the Modbus module functions used in the main.c file for writing and reading Modbus RTU slave data:
    - *eMBErrorCode eMBMasterInit(eMBMode eMode, void *dHUART, ULONG ulBaudRate, void *dHTIM)*

  This functions initializes the ASCII or RTU module and calls the init functions of the porting layer to prepare the hardware. Note that the receiver is still disabled and no Modbus frames are processed until eMBMasterEnable() is called.
- eMBErrorCode eMBMasterPoll(void)
  This function must be called periodically. The timer interval required is given by the application dependent Modbus slave timeout. Internally thefunction calls xMBMasterPortEventGet() and waits for an event from the receiver or transmitter state machines.
  - *unsigned int modbus_req (unsigned int mod_addr, unsigned int mod_data)*

  This function parse the data received from Main system and fetch slave id command type and data from it. This calls the functions below based on the command type.
  - *eMBMasterReqWriteHoldingRegister (slaveid, regnum, regdata, timeout)*
  - *eMBMasterReqWriteCoil (slaveid, regnum, regdata, timeout)*

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.3. PCie Driver

4.3.1. Linux Device Driver Design

When developing Linux kernel features, it is a good practice to expose the necessary details to user-space to enable extensibility. This allows the development of new features and sophisticated configurations from user-space. Commonly, software developers have to face the task of looking for a good way to communicate between kernel and user-space in Linux. This documents introduces you to char driver read/write operation, a flexible and extensible messaging so system that provides communication between kernel and user-space. In order to continue to see in details first see layer of which shown in below image to understand these driver API working.

Kernel Interfaces are key parts of operating systems. The more flexible the interface to communicate kernel and user-space is, the more likely tasks can be efficiently implemented in user-space. As a result, this can reduce the common bloat of adding every new feature into kernel-space.

4.3.2. User-Space to Kernel-Space Access

Figure 4.2 shows the communication details between kernel-space and user-space. The diagram also shows how the application and library interact with the kernel space.

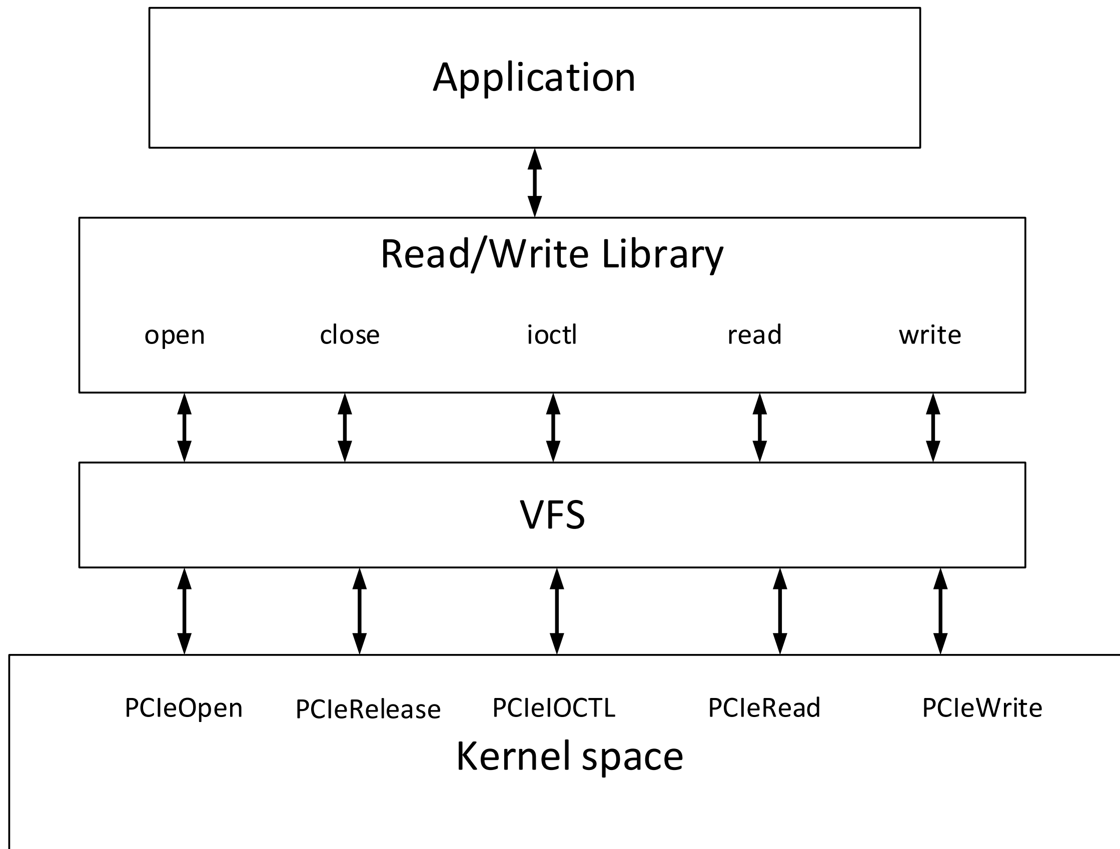


Figure 4.2. User-Space and Kernel-Space Access Diagram

When user calls *open* system call, *PCIeOpen* api of the driver code is invoked, for *close* system call *PCIeRelease* api is invoked and for *ioctl PCIeIOCTL* hits, for *read PCIeRead* hits and write system call invokes the *PCIeWrite* function of the driver code.

In order to access these functions, the user must register the character driver in the kernel. To register character driver, kernel provides some APIs these are mentioned below. For more information, go to: https://www.kernel.org/doc/html/latest/driver-api/index.html.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- alloc_chrdev_region(&brd.ldev_node, 0, 1,gDrvrName);
- cdev_alloc().
- cdev_init(brd.lattice_cdev, &PCIeFileOpt);
- class_create(THIS_MODULE,gDrvrName).
- class_create(THIS_MODULE,gDrvrName).

4.3.3. File Operation and API Description

Currently in this driver, only below mentioned file operation is implemented.

```
struct file_operations PCIeFileOpt = {
        .read     =   PCIeRead,
        .write    =   PCIeWrite,
        .unlocked_ioctl =   PCIeIOCTL,
        .open =       PCIeOpen,
        .release =    PCIeRelease,
};
```

4.3.3.1. PCIeRead

API is used to read data from device, on the call of the *read* system call this API hits. In order to take data from the device to the PC, this function does DMA configuration, waits for the DMA operation to finish, and then reads the system memory once DMA completes. After receiving data from the device, it returns data to user-space on success and it returns error on error.

4.3.3.2. PCIeWrite

API is used to write data from the user space to the device. This function does DMA configuration to write data on the device from the PC, on the success it returns written data size and on erro,r it returns an error.

4.3.3.3. PCIeIOCTL

API is used to do read and write register space of device.

4.3.3.4. PCIeOpen

API is used to open device file to access the read/write and ioctl API, it does common and one-time configuration to use the device. It hits when user calls *open* system call on the driver file.

4.3.3.5. PCIeRelease

API is used to release resource it hits when the user calls the *close* API.

4.3.3.6. Driver API Description

Driver registration process in the kernel is shown below. When the user runs the *insmod* command, the driver entry function module_init(PCIeInit) is called. This is the first function which gets called when the user inserts the driver in the kernel. Inside the pci_register_driver (&lattice_driver) function, is called to register the driver. This API takes the pci_driver structure as input, and this structure should contain the correct information about the device. Each field of the structure is described below.

```
static struct pci_driver lattice_driver = {
        .name = "lthru_demo",
        .id_table = lattice_pci_id_tbl,
        .probe = PCIeProbe,
        .remove = PCIeRemove,
};
```

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

The following is the description of the structure elements:

- name
  This is the driver's name.
- id_table
  This is the pointer to the device ID table the driver is interested. Below is the table structure.

```
struct pci_device_id {
  __u32 vendor, device;
  __u32 subvendor, subdevice;
  __u32 class, class_mask;
  kernel_ulong_t driver_data;
};
```

- probe

This probing function gets called (during execution of pci_register_driver() for already existing devices or later if a new device gets inserted) for all PCI devices which match the ID table and are not *owned* by the other drivers yet. This function gets passed a *struct pci_dev** for each device whose entry in the ID table matches the device. The probe function returns zero when the driver chooses to take *ownership* of the device or an error code (negative number) otherwise.

- remove

The remove() function gets called whenever a device being handled by this driver is removed (either during deregistration of the driver).

4.3.4. PCIeProbe

This function is called during the device registration or on the insertion of device. This function also does all the initializations which are required to get accessed from the user space. On successful registration, it gets the pci_dev structure. This structure contains all the required information which is used in the probe function to get the details of the device.

The Probe function does the below steps. For details on all bus driver APIs, go to:
https://www.kernel.org/doc/html/latest/PCI/pci.html.

- Get bus start address for the given region using the *pci_resource_start(brd.pPciDev,barno);* API.
- Get bus end address for the given region for the device using the *pci_resource_end(brd.pPciDev, barno);* API.
- Get the length in byte of pci region using the *pci_resource_len(brd.pPciDev,barno)* API;
- Get virtual memory for read/write operation on device using the *ioremap(start_addr,len)* API.
- Initialize device before it's used by a driver. It asks the low-level code to enable I/O and memory. Wake up the device if it was suspended by using the *pci_enable_device(brd.pPciDev)* API**.**
- *request_mem_region(pci_start,size,"driver_name");* informs the kernel that your driver is going to use this range of I/O addresses, which prevents other drivers to make any overlapping call to the same region through.
- *pci_alloc_consistent(struct pci_dev *pdev, size_t size, dma_addr_t *dma_handle);* function allocates a DMA buffer, generates its bus address, and returns the associated kernel virtual address. The first two arguments respectively hold the PCI device structure (which is discussed later) and the size of the requested DMA buffer.
- Enable pcie master using *pci_set_master()*.
- Register interrupt handler.
- Create character device node to provide interface to access from user-space. At this stage, it register the user interface API like open, close, read, write, and ioctl.

4.3.5. PCIeRemove

The PCIeRemove function is called whenever the user tries to unload the driver. It deallocates all allocated resource and destroys character unregistered the character driver.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.3.6. Bus Master DMA Overview and Implementation

The term Bus Master, used in the context of PCI express, indicates the ability of a PCIe port to initiate PCIe transactions, typically Memory Read and Write transactions. The most common application for Bus Mastering Endpoints is for DMA. DMA is a technique used for efficient transfer of data to and from host CPU system memory. DMA implementations have many advantages over standard programmed input/output (PIO) data transfers. PIO data transfers are executed directly by the CPU and are typically limited to one (or in some cases two) DWORDs at a time. For large data transfers, DMA implementations result in higher data throughput because the DMA hardware engine is not limited to one or two DWORD transfers. In addition, the DMA engine offloads the CPU from directly transferring the data, resulting in better overall system performance through lower CPU utilization. There are two basic types of DMA hardware implementations found in systems using PCI express: System DMA implementation and Bus Master DMA (BMD) implementation. System DMA implementations typically consist of a shared DMA engine that resides in a central location on the bus and can be used by any device that resides on the bus. System DMA implementations are not commonly found anymore and very few root complexes and operating systems support their use. A BMD implementation is by far the most common type of DMA found in systems based on PCI express. BMD implementations reside within the Endpoint device and are called Bus Masters because they initiate the movement of data to (Memory Writes) and from (Memory Reads) system memory. Figure 4.3 shows a typical system architecture that includes a root complex, PCI express switch device, and an integrated Endpoint block for PCI express. A DMA transfer either transfers data from an integrated Endpoint block for PCI express buffer into system memory or from system memory into the integrated Endpoint block for PCI express buffer. Instead of the CPU having to initiate the transactions needed to move the data, the BMD relieves the processor and allows other processing activities to occur while the data is moved. The DMA request is always initiated by the integrated Endpoint block for PCI express after receiving instructions and buffer location information from the application driver.

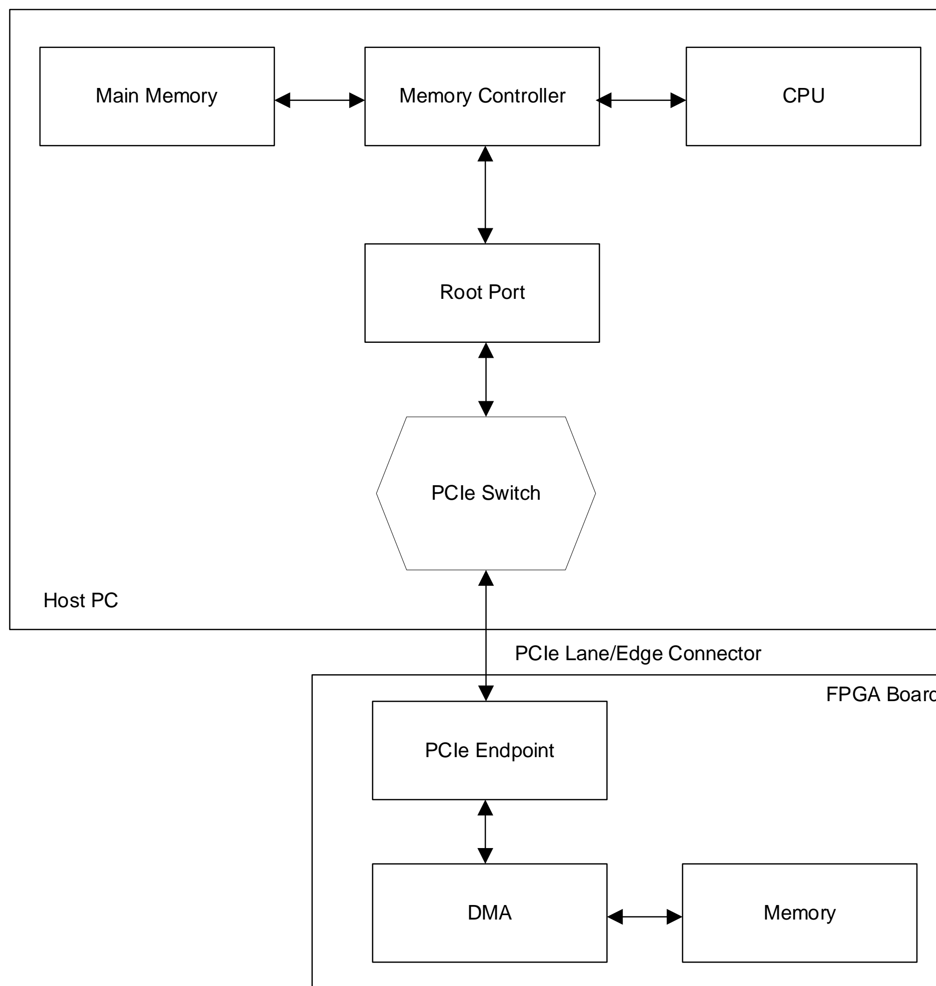


Figure 4.3. Top-level Block Diagram

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

In addition to the data-throughput advantages of DMA versus PIO transactions for large data transfers, many other variables can affect data throughput in PCI express systems. For example, link width and speed, receive buffer sizing, return credit latency, end-to-end latency, and congestion within switches and root complexes. For these reasons, the use of PCI express for high data-throughput applications requires a BMD engine.

Users use BMD with descriptor and Fix Physical memory design as shown in Figure 4.4.

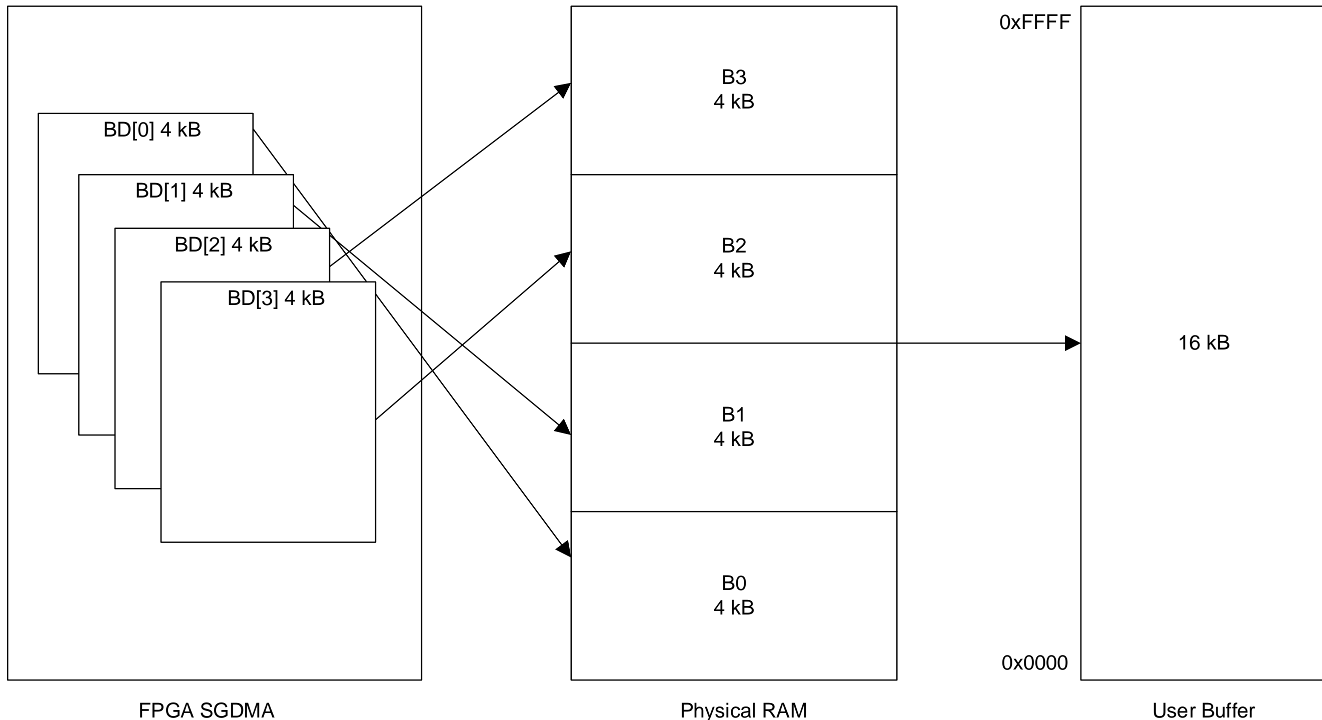


Figure 4.4. BMD with Descriptor and Fixed Physical Memory in RAM

4.4. Programming the DMA Write/Read

To program the DMA Write/Read:

1. Program descriptor in specific format as discussed in the Main System APIs section. Below are some examples for pseudo code to program first descriptor is mentioned,
   - Descriptor queue base address 0x1000.
   - (baseaddr+0) for ((srcAddr << 32) | configuration).
   - (baseaddr+8) for destination address.

   The next descriptor location is *baseaddr+16Byte* because one descriptor size is 16byte.
2. Program all required descriptor. Maximum 256 descriptor supported in current design.
3. Program descriptor pointer register with number of descriptors.
4. Trigger the DMA start.

4.4.1. Supported Operating System

- Distributor ID: Ubuntu
- Description: Ubuntu 18.04.3 LTS
- Release: 18.04.
- OS Type: 64 bit
- Codename: bionic

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.4.2. Package Requirements

To check whether packages are installed or not, run the following commands as shown in the images below.

- make

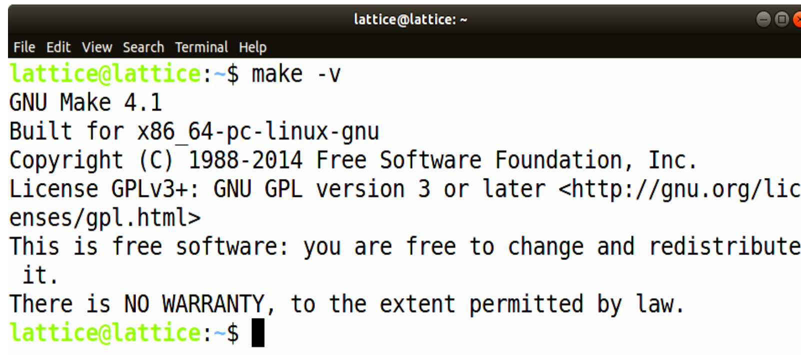


Figure 4.5. Make FIle

- GCC

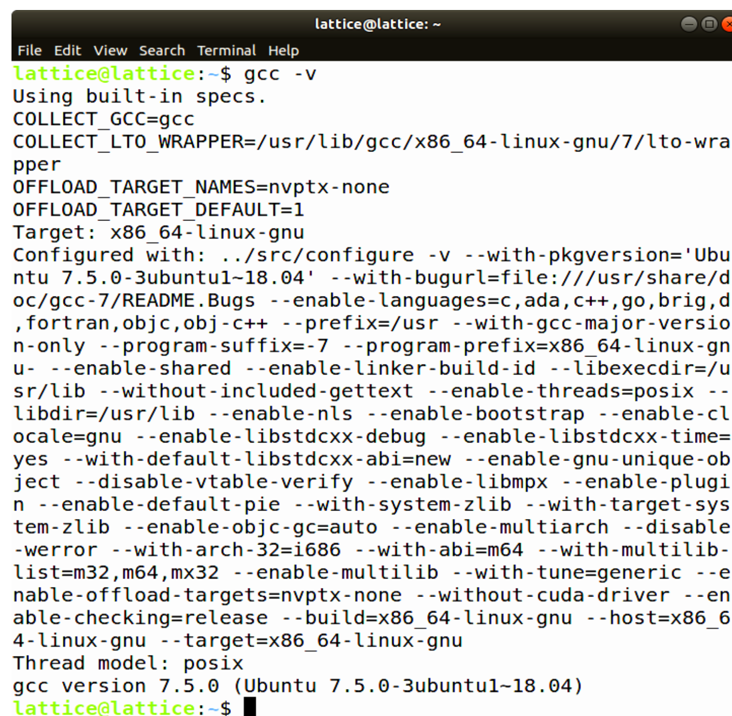


Figure 4.6. GCC Command

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- G++

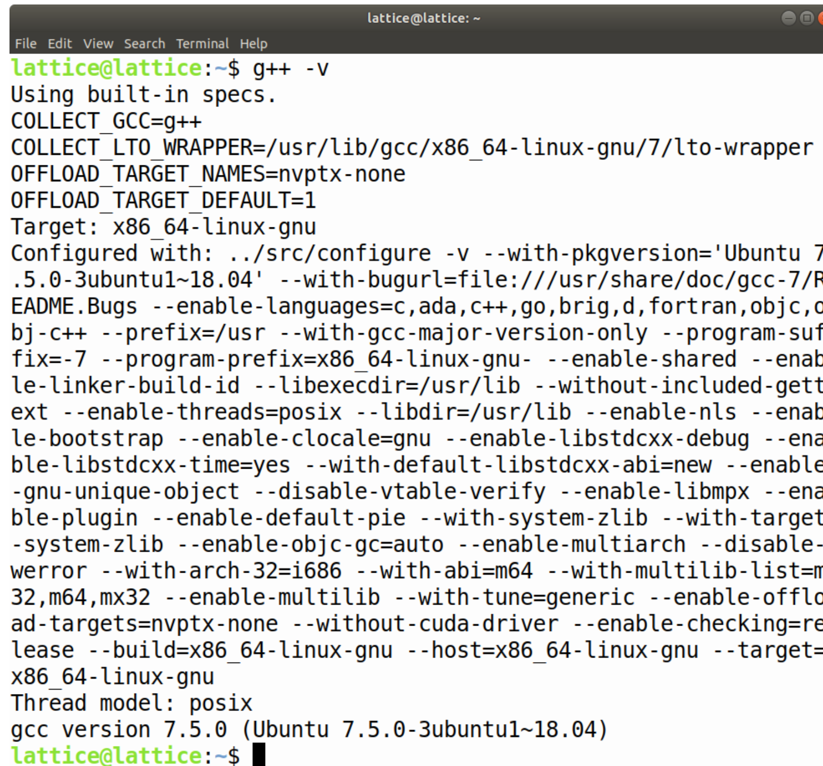


Figure 4.7. G++ Command

- kernel version

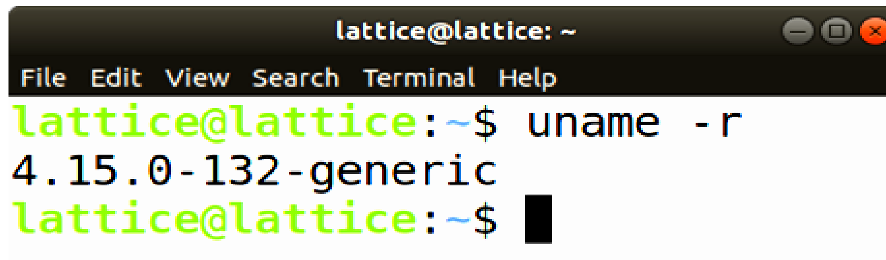


Figure 4.8. Kernel Version Command

4.4.3. Installing the Package

To install the package, run the command below:

```
sudo apt update.
sudo apt install build-essential.
```

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.4.4. Manual Installation and Setup

Ensure to build the driver first before installing it. To build the driver, perform the following steps:

1. Go to the **Demonstration/Linux** directory.
2. Run the *sudo chmod 777 -R Source_Code* command.
3. Go to the **Source_Code** directory.
4. Run the *sudo make clean* and *sudo make* commands to build the driver, API library, and console test application.
5. In the **Source_Code/wrapper/build_wrapper.py** file on **line 7,** replace the python version if the user has a different python version than 3.7. For instance, if the installed python version is 3.6, then update the version from 3.6 to 3.7 as shown below:
   ```
   "-o {1}`python3.6-config --extension-suffix` "
   ```
   Python version can be found by running this command:
   ```
   python3 -V
   ```
6. Run the command below before starting the demo:
   ```
   python3 Source_Code/wrapper/build_wrapper.py
   ```
   This command creates the python binding over the *C* shared library *libmem_rw.so* and is a one-time step.
7. Install the driver using the *insmod* command. Make sure the driver is not installed before doing this step.
8. Run the command below to remove the driver.
   ```
   sudo rmmod lthruput_main.ko
   ```
9. To install the driver, go to ithe */drv_src/lthruput_drv/* directory and run the command below.
   ```
   sudo insmod lthruput_main.ko”
   ```
10. Run the python script.
    ```
    sudo python3 script/script.py
    ```

4.4.5. Automatic Installation and Setup

To setup the demo in automatic mode, perform the following steps:

1. Go to the **Demonstration/Linux** directory.
2. Change the permission of *script.sh* file by running the command below:
   ```
   sudo chmod 777 script.sh
   ```
3. Run the ***sudo ./script.sh*** to build the driver’s API library, console application, and install the PCIe driver.
4. To uninstall the driver, run the ***sudo ./uninstall.sh*** command.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

5. Communications

This section describes the communications between the host to the Main System and the communication between the Main System and the Node Systems. Detailed breakdown of message vocabulary and packet structure may be covered in a separate document.

5.1. Communication between Host and Main System

Initially, this connection is implemented using a USB-2 cable and a UART interface. Most of the messages should be ASCII to facilitate debugging using a terminal program on the Host.

5.1.1. Messages from Host to Main System

- Motor Configuration and Control
- PDM Configuration and Control
- Request Motor Status
- Request PDM Status
- Request PDM Data - Normal
- Request PDM Data - Extended

5.1.2. Messages from Main System to Host

- System Information (Link Status, Connected Nodes, Local Delay of Nodes, and others)
- Motor Status
- PDM Status
- PDM Data - Normal
- PDM Data - Extended

5.2. Communication between Main System and Node System(s)

The physical connection between the Main System and Node System is implemented using Ethernet Cat-5 cables. The physical connection between the first Node System and subsequent Node System(s) also uses Ethernet Cat-5 cables, in a daisy-chain fashion for both chains.

5.2.1. Messages from Main System to Node System

- Motor Configuration and Control
- PDM Configuration and Control
- Request Motor Status
- Request PDM Status
- Request PDM Data - Normal
- Request PDM Data - Extended

5.2.2. Messages from Node System to Main System

- Node Information(Link Status, Connected Nodes, Local Delay etc)
- Motor Status
- PDM Status
- PDM Data - Normal
- PDM Data - Extended

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

6. Demo Package Directory Structure

The directory structure of the Automate Stack Demo Package is listed below.

6.1. Automate Stack Demonstration

- Documentation
- Executables
  - Main System
    a. PDM_DataSection.mem
    b. PDM_ISRCodeSection.mem
    c. riscv-pdm.bin
    d. soc_main_system_impl_1.bit
  - Node System
    a. NodeSystem_AS2_001.bin
    b. NodeSystem_AS2_001.bit
- Host PC
  - User Interface
  - PciScript
- Raspberrypi
  - Mqtt_Lattice_Automate_2.0.zip
  - Readme.txt
- Script
  - Lattice_Automate_Stack_2_0_Docklight.ptp

6.1.1. Documentation

Below is the brief description of the main directories.

- The Automate Stack Demonstration folder is the parent folder for all the files. It has three sub folders:
  - Images
  - Project
  - Documentation.
- The Images sub-folder has the FPGA Images (bit files) and Binary Images (Firmware) for both Main System and node.
- The Project sub-folder contains the whole project package and files for both Main System and node. The FPGA project can be accessed in the *soc_main_system/soc_node* section and firmware project can be accessed in the *c_main_system/c_node* section.
- The documentation subfolder contains the user guide for the project.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

7. Summary

The Lattice Automate Stack 2.0 reference design demonstrates the use of Lattice FPGA devices for industrial motor control and predictive maintenance using ML/AI. It includes foundational IPs from Lattice and firmware for fast development and validation.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Appendix A. Predictive Maintenance with TensorFlow Lite

A.1. Setting Up the Linux Environment for Neural Network Training

This section describes the steps for setting up NVIDIA GPU drivers and/or libraries for 64-bit Ubuntu 16.04 OS. The NVIDIA library and TensorFlow version is dependent on the PC and Ubuntu/Windows version.

A.1.1. Installing the NVIDIA CUDA and cuDNN Library for Machine Learning Training on GPU

A.1.1.1. Installing the CUDA Toolkit

To install the CUDA toolkit, run the following commands in the order specified below:

```
$ curl -O
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-
repo-ubuntu1604_10.1.105-1_amd64.deb
```

```
$ curl -O https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2832  100  2832    0     0   2204      0  0:00:01  0:00:01 --:--:--  2205
```

Figure A.1. Download CUDA Repo

```
$ sudo dpkg –I ./cuda-repo-ubuntu1604_10.1.105-1_amd64.debA
```

```
$ sudo dpkg -i ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
Selecting previously unselected package cuda-repo-ubuntu1604.
(Reading database ... 5287 files and directories currently installed.)
Preparing to unpack .../cuda-repo-ubuntu1604_10.1.105-1_amd64.deb ...
Unpacking cuda-repo-ubuntu1604 (10.1.105-1) ...
Setting up cuda-repo-ubuntu1604 (10.1.105-1) ...

The public CUDA GPG key does not appear to be installed.
To install the key, run this command:
sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
```

Figure A.2. Install CUDA Repo

```
$ sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.p
ub
```

```
$ sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
Executing: /tmp/tmp.a2QZZnTMUX/gpg.1.sh --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
gpg: key 7FA2AF80: public key "cudatools <cudatools@nvidia.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Figure A.3. Fetch Keys

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
$sudo apt-get update
```

```
$ sudo apt-get update
Ign:1 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  InRelease
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Release [697 B]
Get:4 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Release.gpg [836 B]
Hit:5 http://archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:6 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:7 http://archive.ubuntu.com/ubuntu xenial-backports InRelease
Ign:8 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Packages
Get:8 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Packages [428 kB]
Fetched 429 kB in 1s (386 kB/s)
Reading package lists... Done
```

Figure A.4. Update Ubuntu Packages Repositories

```
$ sudo apt-get install cuda-9-0
```

```
$ sudo apt-get install cuda-9-0
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure A.5. CUDA Installation

A.1.1.2. Installing the cuDNN

To install the cuDNN:

1. Create NVIDIA developer account: https://developer.nvidia.com.
2. Download cuDNN lib: https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.1.4/prod/9.0_20180516/cudnn-9.0-linux-x64-v7.1
3. Execute the commands below to install cuDNN

```
$ tar xvfcudnn-9.0-linux-x64-v7.1.tgz
$ sudo cp cuda/include/cudnn.h/usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudochmoda+r/usr/local/cuda/include/cudnn.h/usr/local/cuda/lib64/libcudnn*
```

```
$ tar xcf cudnn-9.0-linux-x64-v7.1.tgz
cuda/include/cudnn.h
cuda/NVIDIA_SLA_cuDNN_Support.txt
cuda/lib64/libcudnn.so
cuda/lib64/libcudnn.so.7
cuda/lib64/libcudnn.so.7.1.4
cuda/lib64/libcudnn_static.a

$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

Figure A.6. cuDNN Library Installation

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

A.1.2. Setting Up the Environment for Training and Model Freezing Scripts

This section describes the environment setup information for training and model freezing scripts for 64-bit Ubuntu 16.04. Anaconda provides one of the easiest ways to perform machine learning development and training on Linux.

A.1.2.1. Installing the Anaconda Python

To install the Anaconda and Python 3:

1. Go to the https://www.anaconda.com/products/individual#download web page.
2. Download Python3 version of Anaconda for Linux.
3. Run the command below to install the Anaconda environment:

```
$  sh  Anaconda3-2019.03-Linux-x86_64.sh
```

Note: Anaconda3-**<version>**-Linux-x86_64.sh, version may vary based on the release.

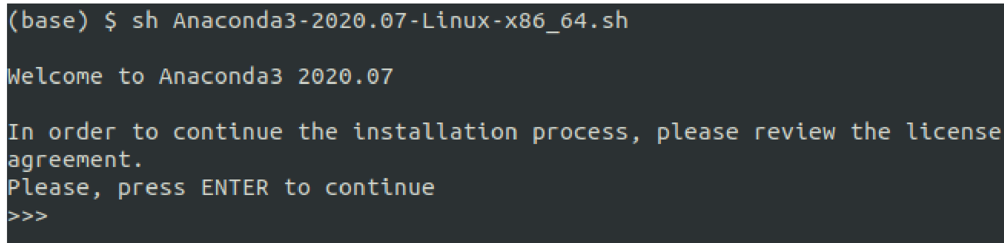


Figure A.7. Anaconda Installation

4. Accept the license.

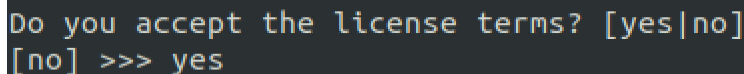


Figure A.8. Accept License Terms

5. Confirm the installation path. Follow the instruction onscreen if you want to change the default path.

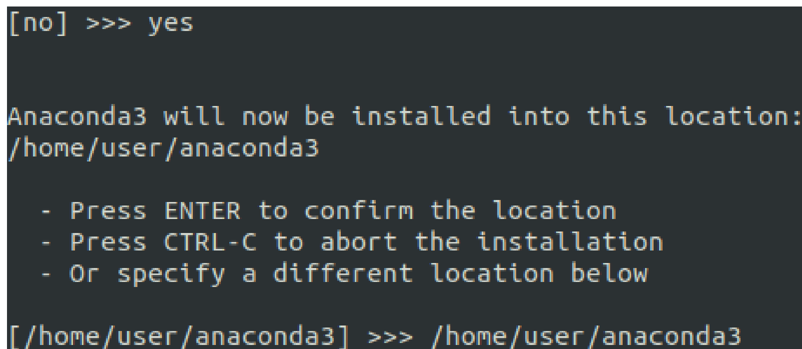


Figure A.9. Confirm/Edit Installation Location

6. After installation, enter *no* as shown in Figure A.10.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>>  no
```

Figure A.10. Launch/Initialize Anaconda Environment on Installation Completion

A.1.3. Installing the TensorFlow version 1.15

To install the TensorFlow version 1.15:

1. Activate the Anaconda environment by running the command below:

```
$ source <conda directory>/bin/activate
```

```
$ source anaconda3/bin/activate
(base) ~$
```

Figure A.11. Anaconda Environment Activation

2. Install the TensorFlow by running the command below:

```
$ conda install tensorflow-gpu==1.15.0
```

```
$ conda install tensorflow-gpu==1.15.0
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3/

  added / updated specs:
    - tensorflow-gpu==1.15.0
```

Figure A.12. TensorFlow Installation

3. After installation, enter **Y** as shown in Figure A.13.

```
  tensorboard         pkgs/main/noarch::tensorboard-1.15.0-pyhb230dea_0
  tensorflow          pkgs/main/linux-64::tensorflow-1.15.0-mkl_py36h4920b83_0
  tensorflow-base     pkgs/main/linux-64::tensorflow-base-1.15.0-mkl_py36he1670d9_0
  tensorflow-estima~ pkgs/main/noarch::tensorflow-estimator-1.15.1-pyh2649769_0
  termcolor           pkgs/main/linux-64::termcolor-1.1.0-py36h06a4308_1
  webencodings        pkgs/main/linux-64::webencodings-0.5.1-py36_1
  werkzeug            pkgs/main/noarch::werkzeug-0.16.1-py_0
  wrapt               pkgs/main/linux-64::wrapt-1.12.1-py36h7b6447c_1
  zipp                pkgs/main/noarch::zipp-3.4.0-pyhd3eb1b0_0


Proceed ([y]/n)? y
```

Figure A.13. TensorFlow Installation Confirmation

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Figure A.14 shows TensorFlow installation is complete.

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Figure A.14. TensorFlow Installation Completion

A.1.4. Installing the Python Package

To install the Python package:

1. Install Easydict by running the command below:
```
$ conda install –c conda-forge easydict
```

```
(base) $ conda install -c conda-forge easydict
Solving environment: done
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - easydict
```

Figure A.15. Easydict Installation

2. Install Joblib by running the command below:
```
$ conda install joblib
```

```
(base) $ conda install joblib
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - joblib
```

Figure A.16. Joblib Installation

3. Install Keras by running the command below:
```
$ conda install keras
```

```
(base) $ conda install keras
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - keras
```

Figure A.17. Keras Installation

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4. Install OpenCV by running the command below:
```
$ conda install opencv
```

```
(base) $ conda install opencv
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - opencv
```

Figure A.18. OpenCV Installation

5. Install Pillow by running the command below:
```
$ conda install pillow
```

```
(base) $ conda install pillow
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - pillow
```

Figure A.19. Pillow Installation

A.2. Creating the TensorFlow Lite Conversion Environment

To create a new Anaconda environment and install tensorflow=2.2.0:

1. Create a new Anaconda environment.
```
$ conda create -n <New Environment Name> python=3.6
```
2. Activate new created environment.
```
$ conda activate <New Environment Name>
```
3. Install Tensorflow 2.2.0.

   Note: We have noticed output difference in Tensorflow(2.2.0) and Tensorflow-gpu(2.2.0) in terms of tflite size.

   It is recommended to use TensorFlow (2.2.0).
```
$ conda install tensorflow=2.2.0
```
4. Install opencv.
```
$conda install opencv
```

A.3. Preparing the Dataset

This section describes the steps and guidelines used to prepare the dataset for training the predictive maintenance.

Note: In the following sections, Lattice provides guidelines and/or examples that can be used as references for preparing the dataset for the given use cases. Lattice is not recommending and/or endorsing any dataset(s). It is recommended that customers gather and prepare their own datasets for their specific end applications.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

A.3.1. Dataset Information

In the predictive maintenance demonstration, there are three classes: bad, Normal, and unknown. The dataset should be organized as shown in below Fig. 35 *0* contains bad motor data and *1* contains normal motor data.

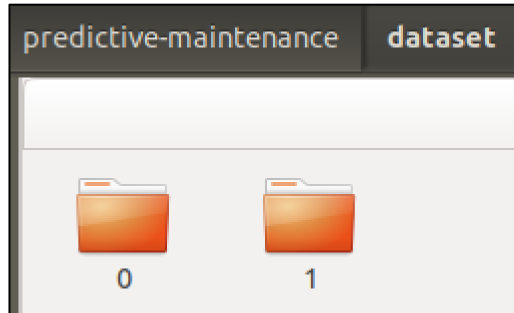


Figure A.20. Predictive Maintenance Dataset

A.4. Preparing the Training Code

Notes:

- Training and freezing code uses Tensorflow 1.15.0 since some of the APIs used in training code are not available in Tensorflow 2.x.
- For the TensorFlow Lite conversion in the TensorFlow Lite Conversion and Evaluation section**,** TensorFlow 2.2.0 is used.

A.4.1. Training Code Structure

Download the Lattice predictive maintenance demo training code. Its directory structure is shown in Fig. 36

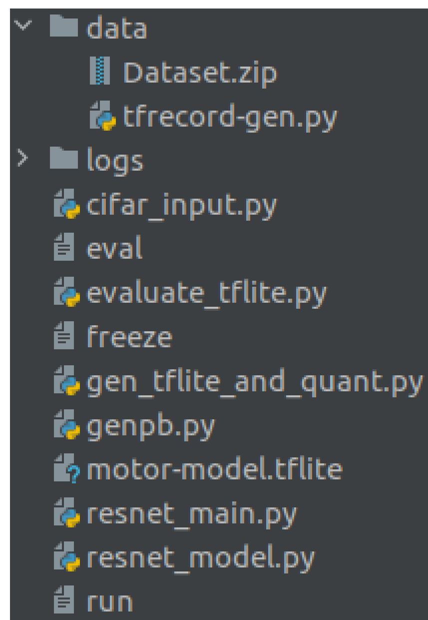


Figure A.21. Training Code Directory Structure

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

A.4.2. Generating tfrecords from Augmented Dataset

This demo only takes tfrecords of a specific format for input. As susch, generate the tfrecords first. Run the command below to generate tfrecords from input dataset.

```
$ python tfrecord-gen.py -i <Input_augmented_dataset_root> -o <Output_tfrecord_path>
```

The input directory should follow the structure shown in Figure A.21.

A.4.3 Neural Network Architecture

This section provides information on the Convolution Neural Network Configuration of the Predictive Maintenance design.

Table A.1. Predictive Maintenance Training Network Topology

<table>
<tr><th colspan="3">Input Gray Scale Image (64×64×1)</th></tr>
<tr><td rowspan="4">Fire1</td><td>Conv3x3 – 8</td><td rowspan="28">Conv3×3 - # where:
• Conv3×3 – 3 × 3 Convolution filter Kernel size
• # - The number of filters
For example, Conv3×3 - 8 = 8 3 × 3 convolution filter

Batchnorm: Batch Normalization
FC - # where:
• FC – Fully connected layer
• # - The number of outputs</td></tr>
<tr><td>Batchnorm</td></tr>
<tr><td>ReLU</td></tr>
<tr><td>Maxpool</td></tr>
<tr><td rowspan="3">Fire2</td><td>Conv3×3 – 8</td></tr>
<tr><td>Batchnorm</td></tr>
<tr><td>ReLU</td></tr>
<tr><td rowspan="4">Fire3</td><td>Conv3×3 – 16</td></tr>
<tr><td>Batchnorm</td></tr>
<tr><td>ReLU</td></tr>
<tr><td>Maxpool</td></tr>
<tr><td rowspan="3">Fire4</td><td>Conv3×3 – 16</td></tr>
<tr><td>Batchnorm</td></tr>
<tr><td>ReLU</td></tr>
<tr><td rowspan="4">Fire5</td><td>Conv3×3 – 16</td></tr>
<tr><td>Batchnorm</td></tr>
<tr><td>ReLU</td></tr>
<tr><td>Maxpool</td></tr>
<tr><td rowspan="3">Fire6</td><td>Conv3×3 – 22</td></tr>
<tr><td>Batchnorm</td></tr>
<tr><td>ReLU</td></tr>
<tr><td rowspan="4">Fire7</td><td>Conv3×3 – 24</td></tr>
<tr><td>Batchnorm</td></tr>
<tr><td>ReLU</td></tr>
<tr><td>Maxpool</td></tr>
<tr><td>Dropout</td><td>Dropout - 0.80</td></tr>
<tr><td>logit</td><td>FC – (3)</td></tr>
</table>

In Table A.1, Layer contains Convolution (conv), batch normalization (BN), ReLU, pooling, and dropout layers. Output of layer logit is (Broken [0], Normal [1], Unknown [2]) 3 values.

- Layer information
  - Convolutional Layer
    In general, the first layer in a CNN is always a convolutional layer. Each layer consists of number of filters (sometimes referred as kernels) which convolves with input layer/image and generates activation map (such as feature map). This filter is an array of numbers (the numbers are called weights or parameters). Each of these filters can be thought of as feature identifiers, like straight edges, simple colors, and curves and other high-level features. For example, the filters on the first layer convolve around the input image and "activate" (or compute high values) when the specific feature (say curve) it is looking for is in the input volume.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- ReLU (Activation Layer)
  After each conv layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. The ReLU layer applies the function f(x) = max (0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.
- Pooling Layer
  After some ReLU layers, programmers may choose to apply a pooling layer. It is also referred to as a down sampling layer. In this category, there are also several layer options, with Maxpooling being the most popular. This basically takes a filter (normally of size 2×2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every sub region that the filter convolves around.
  The intuitive reasoning behind this layer is that once the user knows that a specific feature is in the original input volume (a high activation value results), its exact location is not as important as its relative location to the other features. As you can imagine, this layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume. This serves two main purposes. The first is that the number of parameters or weights is reduced by 75%, thus lessening the computation cost. The second is that it controls over fitting. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. A symptom of over fitting is having a model that gets 100% or 99% on the training set, but only 50% on the test data.
- Batchnorm Layer
  Batch normalization layer reduces the internal covariance shift. In order to train a neural network, perform pre-processing to the input data. For example, the user can normalize all data so that it resembles a normal distribution (that means, zero mean and a unitary variance). Reason being preventing the early saturation of non-linear activation functions like the sigmoid function, assuring that all input data is in the same range of values, etc.
  But the problem appears in the intermediate layers because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt themselves to a new distribution in every training step. This problem is known as internal covariate shift.
  Batch normalization layer forces the input of every layer to have approximately the same distribution in every training step by following below process during training time:
  - Calculate the mean and variance of the layers input.
  - Normalize the layer inputs using the previously calculated batch statistics.
  - Scales and shifts in order to obtain the output of the layer.

  This makes the learning of layers in the network more independent of each other and allows you to be care free about weight initialization, works as regularization in place of dropout and other regularization techniques.
- Drop-out Layer
  Dropout layers have a very specific function in neural networks. After training, the weights of the network are so tuned to the training examples they are given that the network doesn’t perform well when given new examples. The idea of dropout is simplistic in nature. This layer *drops out* a random set of activations in that layer by setting them to zero. It forces the network to be redundant. That means the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out. It makes sure that the network is not getting too “fitted” to the training data and thus helps alleviate the over fitting problem. An important note is that this layer is only used during training, and not during test time.
- Fully connected Layer
  This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program must choose from.
- Quantization
  Quantization is a method to bring the neural network to a reasonable size, while also achieving high performance accuracy. This is especially important for on-device applications, where the memory size and number of computations are necessarily limited. Quantization for deep learning is the process of approximating a neural network that uses floating-point numbers by a neural network of low bit width numbers. This dramatically reduces both the memory requirement and computational cost of using neural networks.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

The above architecture provides nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting.

A.4.4. Training Code Overview

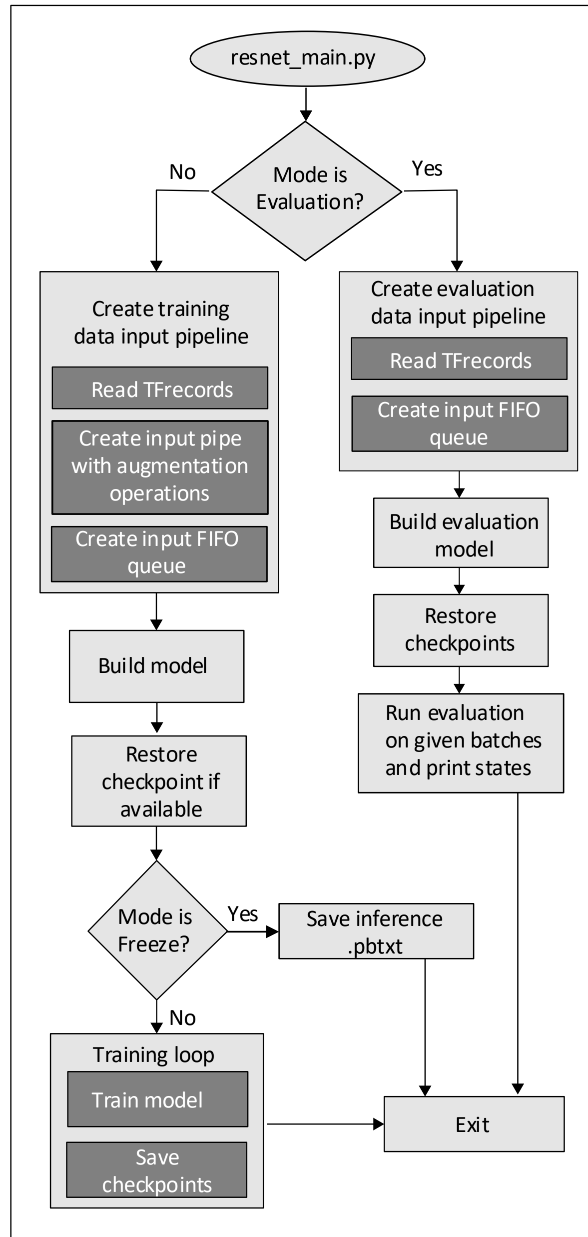


Figure A.22. Training Code Flow Diagram

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

A.4.4.1. Configuring Hyper-Parameters

```
if FLAGS.mode == 'train':
    batch_size = 128
elif FLAGS.mode == 'eval':
    batch_size = 100
if FLAGS.mode == 'freeze':
    batch_size = 1

if FLAGS.dataset == 'signlang':
    num_classes = 2 + 1  # broken, normal, unknown

hps = resnet_model.HParams(batch_size=batch_size,
                           num_classes=num_classes,
                           min_lrn_rate=0.0001,
                           lrn_rate=0.1,
                           num_residual_units=5,  # 2*3*this
                           use_bottleneck=False,
                           weight_decay_rate=0.0002,
                           relu_leakiness=0.1,
                           optimizer='mom'  # sgd, mom, adam,
                           )  # resNet enable
```

Figure A.23. Code Snippet: Hyper Parameters

- Set number of class in *num_classes* (default = 3).
- Change batch size for specific mode if required.
- hps: it contains list of hyper parameters for custom resnet backbone and optimizer.

A.4.4.2. Creating Training Data Input Pipeline

```
images, labels = cifar_input.build_input(
    FLAGS.dataset, FLAGS.train_data_path, hps.batch_size, FLAGS.mode, FLAGS.gray, hps[1])
```

Figure A.24. Code Snippet: Build Input

- *build_input* () from *cifer_input.py* reads Tfrecords and creates some augmentation operations before pushing the input data to FIFO queue.
    - *FLAGS.dataset :* dataset type (signlang)
    - FLAGS.train_data_path: input path to tfrecords
    - FLAGS.batch_size: training batch size
    - FLAGS.mode: train or eval
    - *FLAGS.gray:* True if model is of 1 channel otherwise False
    - *hps[1]:* num_classes configured in model hyper parameters

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Read tfrecords

```
if dataset == 'signlang':  # TFRecord format
    reader = tf.TFRecordReader()
    _, serialized_example = reader.read(file_queue)
    features = tf.parse_single_example(
        serialized_example,
        features={
            'image/height': tf.FixedLenFeature([], tf.int64),
            'image/width': tf.FixedLenFeature([], tf.int64),
            'image/class/label': tf.FixedLenFeature([], tf.int64),
            'image/encoded': tf.FixedLenFeature([], tf.string)
        }
    )
```

Figure A.25. Code Snippet: Parse tfrecords

- Above snippet reads tfrecord files and parse its features that are height, width, label and image.

Converting Image to Grayscale and Scaling the Image

```
if gray:  # Gray color
    image = tf.image.rgb_to_grayscale(image)
    depth = 1
```

Figure A.26. Code Snippet: Convert Image to Gray Scale

- Convert RGB image to gray scale if gray flag is true.

```
channels = tf.unstack(image, axis=-1)

if gray:
    image = tf.stack([channels[0]], axis=-1)
else:
    # RGB to BGR Conversion
    image = tf.stack([channels[2], channels[1], channels[0]], axis=-1)

# image /= 128.0  # [0, 2)
image = image - 128.0
```

Figure A.27. Code Snippet: Convert Image to BGR and Scale the Image

- Unstack channel layers and convert to BGR format if the image mode is not gray. The RGB is converted to BGR because the iCE40 works on BGR image.
- Divide every element on image with 128 so that the values can be scaled to 0-2 range.

Creating Input Queue

```
example_queue = tf.RandomShuffleQueue(
    capacity=16 * batch_size,
    min_after_dequeue=8 * batch_size,
    dtypes=[tf.float32, tf.int32],
    shapes=[[image_size, image_size, depth], [1]])
num_threads = 16
```

Figure A.28. Code Snippet: Create Queue

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- tf.RandomShuffleQueue is queue implementation that dequeues elements in random order.

```
example_enqueue_op = example_queue.enqueue([image, label])
tf.train.add_queue_runner(tf.train.queue_runner.QueueRunner(
    example_queue, [example_enqueue_op] * num_threads))

# Read 'batch' labels + images from the example queue.
images, labels = example_queue.dequeue_many(batch_size)
labels = tf.reshape(labels, [batch_size, 1])
indices = tf.reshape(tf.range(0, batch_size, 1), [batch_size, 1])
labels = tf.sparse_to_dense(
    tf.concat(values=[indices, labels], axis=1),
    [batch_size, num_classes], 1.0, 0.0)
```

Figure A.29. Code Snippet: Add Queue Runners

- Above snippet enqueues images and labels to the *RandomShuffleQueue* and add queue runners. This directly feeds data to network.

A.4.4.3. Model Building

CNN Architecture

```
model = resnet_model.ResNet(hps, images, labels, FLAGS.mode)
model.build_graph()
```

Figure A.30. Code Snippet: Create Model

- *Build_graph ()* method creates training graph or training model using given configuration.
- *Build_graph* creates model with seven fire layers followed by dropout layer and fully connected layers. Where each fire layer contains convolution, relu as activation, batch normalization, and max pooling (in Fire 1, 3, 5 & 7 only). Fully connected layer provides the final output.

```
fire1 = self._vgg_layer('fire1', self._images, oc=depth[0], freeze=False, pool_en=True,
                        bias_on=bias_on, phase_train=phase_train)
```

Figure A.31. Code Snippet: Fire Layer

Arguments of *_vgg_layer*:

- First argument is name of the block.
- Second argument is input node to new fire block.
- *oc*: output channels is the number of filters of the convolution.
- *freeze*: setting weighs are trainable or not.
- *w_bin*: Quantization parameter for convolution
- *a_bin*: quantization parameter for activation binarization(relu).
- *pool_en:* flag to include Maxpool in firelayer.
- *min_rng, max_rng*: Setting maximum and minimum values of quantized activation. Default values for *min_rng = 0.0* and *max_rng = 2.0*.
- *bias_on*: Sets bias add operation in graph if true.
- *phase_train*: Argument to generate graph for inference and training.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
def _vgg_layer(self, layer_name, inputs, oc, stddev=0.01, freeze=False, w_bin=16, a_bin=16, pool_en=True,
               min_rng=-0.5, max_rng=0.5, bias_on=True, phase_train=True):
    with tf.variable_scope(layer_name):
        net = self._conv_layer('conv3x3', inputs, filters=oc, size=3, stride=1, xavier=False,
                               padding='SAME', stddev=stddev, freeze=freeze, relu=False, w_bin=w_bin,
                               bias_on=bias_on)
        tf.summary.histogram('before_bn', net)
        net = self._batch_norm_tensor2('bn', net, phase_train=phase_train)  # BatchNorm
        tf.summary.histogram('before_relu', net)
        net = self.binary_wrapper(net, a_bin=a_bin, min_rng=min_rng, max_rng=max_rng)  # ReLU
        tf.summary.histogram('after_relu', net)
        if pool_en:
            pool = self._pooling_layer('pool', net, size=2, stride=2, padding='SAME')
        else:
            pool = net
        tf.summary.histogram('pool', pool)

        return pool
```

Figure A.32. Code Snippet: Convolution Block

- In the *resnet_model.py* file, the basic network construction blocks are implemented in specific functions as below:
  - Convolution – _conv_layer
  - Batch normalization – _batch_norm_tensor2
  - ReLU – binary_wrapper
  - Maxpool – _pooling_layer
- _conv_layer
  - Contains code to create convolution block. Which contains kernel variable, variable initializer, quantization code, convolution operation and ReLU if argument *relu* is True.
- _batch_norm_tensor2
  - Contains code to create batch-normalization operation for both training and inference phase.
- Binary_wrapper
  - Used for quantized activation with ReLU.
- _pooling_layer
  - Adds Max pooling with given kernel-size and stride size to training and inference graph.

Feature Depth of Fire Layer

```
depth = [8, 8, 16, 16, 16, 22, 24]
```

Figure A.33. Code Snippet: Feature Depth Array for Fire Layers

- List *depth* contains feature depth for seven fire layers in network.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
fire1 = self._vgg_layer('fire1', self._images, oc=depth[0], freeze=False, pool_en=True,
                        bias_on=bias_on, phase_train=phase_train)
fire2 = self._vgg_layer('fire2', fire1, oc=depth[1], freeze=False, pool_en=False,
                        bias_on=bias_on, phase_train=phase_train)
fire3 = self._vgg_layer('fire3', fire2, oc=depth[2], freeze=False, pool_en=True,
                        bias_on=bias_on, phase_train=phase_train)
fire4 = self._vgg_layer('fire4', fire3, oc=depth[3], freeze=False, pool_en=False,
                        bias_on=bias_on, phase_train=phase_train)
fire5 = self._vgg_layer('fire5', fire4, oc=depth[4], freeze=False, pool_en=True,
                        bias_on=bias_on, phase_train=phase_train)
fire6 = self._vgg_layer('fire6', fire5, oc=depth[5], freeze=False, pool_en=False,
                        bias_on=bias_on, phase_train=phase_train)
fire7 = self._vgg_layer('fire7', fire6, oc=depth[6], freeze=False, pool_en=True,
                        bias_on=bias_on, phase_train=phase_train)
if phase_train:
    fire_o = tf.nn.dropout(fire7, 0.8)
else:
    fire_o = tf.nn.dropout(fire7, 1)
logits = self._fc_layer('logit', fire_o, self.hps.num_classes, flatten=True, relu=False, xavier=True)
```

Figure A.34. Code Snippet: Forward Graph Fire Layers

Loss Function and Optimizers

```
with tf.variable_scope('costs'):
    xent = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=self.labels)
    self.cost = tf.reduce_mean(xent, name='xent')
    self.cost += self._decay()

    tf.summary.scalar('cost', self.cost)
```

Figure A.35. Code Snippet: Loss Function

- Model uses *softmax_cross_entropy_with_logitds* because the labels are in form of class index.

```
if self.hps.optimizer == 'sgd':
    optimizer = tf.train.GradientDescentOptimizer(self.lrn_rate)
elif self.hps.optimizer == 'mom':
    optimizer = tf.train.MomentumOptimizer(self.lrn_rate, 0.9)
elif self.hps.optimizer == 'adam':
    optimizer = tf.train.AdamOptimizer(self.lrn_rate)
elif self.hps.optimizer == 'rmsprop':
    optimizer = tf.train.RMSPropOptimizer(self.lrn_rate, decay=0.9, momentum=0.9, epsilon=1.0)
```

Figure A.36. Code Snippet: Optimizers

- Here, there are four options for selecting optimizers. In this model, use the *mom* optimizer as default.

A.4.4.4. Restore Checkpoints

Checkpoints are restored from log directory and then starts training from that checkpoint if checkpoints exist in log directory.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
try:
    ckpt_state = tf.train.get_checkpoint_state(FLAGS.ref_log_root)
    if not (ckpt_state and ckpt_state.model_checkpoint_path):
        tf.logging.info('No model to eval yet at %s', FLAGS.ref_log_root)
    else:
        tf.logging.info('Loading checkpoint %s', ckpt_state.model_checkpoint_path)
        saver.restore(sess, ckpt_state.model_checkpoint_path)
except Exception as e:
    tf.logging.error('Cannot restore checkpoint: %s', e)
```

Figure A.37. Code Snippet: Restore Checkpoints

A.4.4.5. Saving .pbtxt

If mode is *freeze* it saves the inference graph (model) as *.pbtxt* file. The *.pbtxt* file is used later for freezing.

```
if FLAGS.mode == "freeze":
    tf.train.write_graph(sess.graph_def, FLAGS.log_root, "model.pbtxt")
    print("Saved model.pbtxt at", FLAGS.log_root)
    sys.exit()
tf.train.start_queue_runners(sess)
```

Figure A.38. Code Snippet: Save .pbtxt

A.4.4.6. Training Loop

```
with tf.train.MonitoredTrainingSession(
        checkpoint_dir=FLAGS.log_root,
        hooks=[logging_hook, _LearningRateSetterHook()],
        chief_only_hooks=[summary_hook],
        save_summaries_steps=0,
        save_checkpoint_steps=FLAGS.ckptinterval,
        config=tf.ConfigProto(allow_soft_placement=True)) as mon_sess:
    confusion_matrix = np.zeros((hps[1], hps[1]))
    while not mon_sess.should_stop() and mon_sess.run(model.global_step) < FLAGS.maxsteps:
        _, confusion = mon_sess.run([model.train_op, model.confusion_matrix])
        confusion_matrix = np.add(confusion_matrix, np.array(confusion))
        if mon_sess.run(model.global_step) % FLAGS.ckptinterval == 0 and mon_sess.run(model.global_step) != 0:
            print("Confusion_Matrix :\n {}".format(confusion_matrix.astype(np.int)))
            confusion_matrix = np.zeros((hps[1], hps[1]))
```

Figure A.39. Code Snippet: Training Loop

- *MonitoredTrainingSession* utility sets proper session initializer/restorer. It also creates hooks related to checkpoint and summary saving. For workers, this utility sets proper session creator which waits for the chief to initialize/restore. Refer to tf.compat.v1.train.MonitoredSession for more information.
- _LearningRateSetterHook:

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
def after_run(self, run_context, run_values):
    train_step = run_values.results
    if train_step < 20000:
        self._lrn_rate = 0.1
    elif train_step < 35000:
        self._lrn_rate = 0.01
    elif train_step < 50000:
        self._lrn_rate = 0.001
    elif train_step < 60000:
        self._lrn_rate = 0.0001
    else:
        self._lrn_rate = 0.00001
```

Figure A.40. Code Snippet: _ LearningRateSetterHook

- This hook sets learning rate based on training steps performed.
- Summary_hook

```
summary_hook = tf.train.SummarySaverHook(
    save_steps=100,
    output_dir=FLAGS.train_dir,
    summary_op=tf.summary.merge([model.summaries,
                             tf.summary.scalar('Precision', precision)]))
```

Figure A.41. Code Snippet: Save Summary for Tensorboard

- Saves tensorboard summary for every 100 steps.
- Logging_hook

```
logging_hook = tf.train.LoggingTensorHook(
    tensors={'step': model.global_step,
             'loss': model.cost,
             'precision': precision},
    every_n_iter=100)
```

Figure A.42. Code Snippet: logging hook

- Prints logs after every 100 iterations.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

A.4.5. Training from Scratch and/or Transfer Learning

A.4.5.1. Training

Open the *run* script and edit parameters as required.

```
python resnet_main.py \
    --train_data_path=/home/dataset/training/data/tfrecords/ \
    --log_root=./logs/train \
    --train_dir=./logs/train \
    --dataset='signlang' \
    --image_size=64 \
    --num_gpus=1 \
    --mode=train
```

Figure A.43. Predictive Maintenance – Run Script

To start training run the run script as mentioned below.

```
$ ./run
```

```
INFO:tensorflow:Graph was finalized.
I0421 12:06:05.044778 140199668410176 monitored_session.py:240] Graph was finalized.
INFO:tensorflow:Running local_init_op.
I0421 12:06:05.397494 140199668410176 session_manager.py:500] Running local_init_op.
INFO:tensorflow:Done running local_init_op.
I0421 12:06:05.415019 140199668410176 session_manager.py:502] Done running local_init_op.
I0421 12:06:06.224065 140199668410176 basic_session_run_hooks.py:606] Saving checkpoints for 0 into ./logs/train/model.ckpt.
INFO:tensorflow:loss = 2.7537637, precision = 0.03125, step = 0
I0421 12:06:07.673088 140199668410176 basic_session_run_hooks.py:262] loss = 2.7537637, precision = 0.03125, step = 0
INFO:tensorflow:loss = 8.1982155, precision = 0.984375, step = 34 (12.621 sec)
I0421 12:06:20.293941 140199668410176 basic_session_run_hooks.py:260] loss = 8.1982155, precision = 0.984375, step = 34 (12.621 sec)
INFO:tensorflow:loss = 8.56728, precision = 1.0, step = 67 (10.728 sec)
I0421 12:06:31.022384 140199668410176 basic_session_run_hooks.py:260] loss = 8.56728, precision = 1.0, step = 67 (10.728 sec)
INFO:tensorflow:global_step/sec: 2.78888
I0421 12:06:43.529809 140199668410176 basic_session_run_hooks.py:692] global_step/sec: 2.78888
INFO:tensorflow:loss = 8.464728, precision = 1.0, step = 100 (12.649 sec)
I0421 12:06:43.671326 140199668410176 basic_session_run_hooks.py:260] loss = 8.464728, precision = 1.0, step = 100 (12.649 sec)
INFO:tensorflow:loss = 8.351438, precision = 1.0, step = 134 (9.906 sec)
I0421 12:06:53.577040 140199668410176 basic_session_run_hooks.py:260] loss = 8.351438, precision = 1.0, step = 134 (9.906 sec)
```

Figure A.44. Predictive Maintenance – Trigger Training

A.4.5.2. Transfer Learning

```
INFO:tensorflow:Create CheckpointSaverHook.
I0421 12:27:24.113183 139632644269888 basic_session_run_hooks.py:541] Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from ./logs/train/model.ckpt-4000
I0421 12:27:24.581049 139632644269888 saver.py:1280] Restoring parameters from ./logs/train/model.ckpt-4000
INFO:tensorflow:Saving checkpoints for 4000 into ./logs/train/model.ckpt.
I0421 12:27:25.606637 139632644269888 basic_session_run_hooks.py:606] Saving checkpoints for 4000 into ./logs/train/model.ckpt.
INFO:tensorflow:loss = 1.7730277, precision = 1.0, step = 4000
I0421 12:27:26.982490 139632644269888 basic_session_run_hooks.py:262] loss = 1.7730277, precision = 1.0, step = 4000
INFO:tensorflow:loss = 1.7497265, precision = 1.0, step = 4034 (11.972 sec)
I0421 12:27:38.954078 139632644269888 basic_session_run_hooks.py:260] loss = 1.7497265, precision = 1.0, step = 4034 (11.972 sec)
INFO:tensorflow:loss = 1.7260615, precision = 1.0, step = 4067 (9.745 sec)
I0421 12:27:48.698793 139632644269888 basic_session_run_hooks.py:260] loss = 1.7260615, precision = 1.0, step = 4067 (9.745 sec)
INFO:tensorflow:global_step/sec: 3.12938
I0421 12:27:58.937695 139632644269888 basic_session_run_hooks.py:692] global_step/sec: 3.12938
INFO:tensorflow:loss = 1.7033625, precision = 1.0, step = 4100 (10.303 sec)
I0421 12:27:59.001640 139632644269888 basic_session_run_hooks.py:260] loss = 1.7033625, precision = 1.0, step = 4100 (10.303 sec)
INFO:tensorflow:loss = 1.6810057, precision = 1.0, step = 4134 (9.778 sec)
I0421 12:28:08.779979 139632644269888 basic_session_run_hooks.py:260] loss = 1.6810057, precision = 1.0, step = 4134 (9.778 sec)
INFO:tensorflow:loss = 1.6583471, precision = 1.0, step = 4167 (10.514 sec)
I0421 12:28:19.294208 139632644269888 basic_session_run_hooks.py:260] loss = 1.6583471, precision = 1.0, step = 4167 (10.514 sec)
```

Figure A.45. Predictive Maintenance – Trigger Training with Transfer Learning

- To restore checkpoints, no additional action is required. Run the same command again with the same log directory. if the checkpoints are present in log path where it is be restored and continue training from that step.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

A.4.5.3. Training Status

- Training status can be checked in logs by observing different terminologies like loss, precision and confusion matrix.

```
I0707 12:36:19.063314 139684916700992 basic_session_run_hooks.py:260] loss = 0.18542665, precision = 0.984375, step = 8500 (5.558 sec)
INFO:tensorflow:loss = 0.20943533, precision = 0.9609375, step = 8550 (5.665 sec)
I0707 12:36:24.728753 139684916700992 basic_session_run_hooks.py:260] loss = 0.20943533, precision = 0.9609375, step = 8550 (5.665 sec)
INFO:tensorflow:global_step/sec: 8.87325
I0707 12:36:30.285727 139684916700992 basic_session_run_hooks.py:692] global_step/sec: 8.87325
INFO:tensorflow:loss = 0.22918972, precision = 0.96875, step = 8600 (5.601 sec)
I0707 12:36:30.329452 139684916700992 basic_session_run_hooks.py:260] loss = 0.22918972, precision = 0.96875, step = 8600 (5.601 sec)
INFO:tensorflow:loss = 0.2660838, precision = 0.96875, step = 8650 (5.712 sec)
I0707 12:36:36.041846 139684916700992 basic_session_run_hooks.py:260] loss = 0.2660838, precision = 0.96875, step = 8650 (5.712 sec)
INFO:tensorflow:global_step/sec: 8.83564
I0707 12:36:41.603530 139684916700992 basic_session_run_hooks.py:692] global_step/sec: 8.83564
INFO:tensorflow:loss = 0.2278277, precision = 0.9609375, step = 8700 (5.610 sec)
I0707 12:36:41.652254 139684916700992 basic_session_run_hooks.py:260] loss = 0.2278277, precision = 0.9609375, step = 8700 (5.610 sec)
```

Figure A.46. Predictive Maintenance – Training Logs

```
Confusion_Matrix :
 [[ 14726      0      0]
 [      0 113274      0]
 [      0      0      0]]
```

Figure A.47. Predictive Maintenance – Confusion Matrix

- You can use TensorBoard utility for checking training status.
  - Start TensorBoard by below command:

```
$ tensorboard –logdir=<log directory of training>
```

```
$ tensorboard --logdir logs/train/
TensorBoard 1.15.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Figure A.48. TensorBoard – Launch

  - This command provides the link, which needs to be copied and open in any browser such as Chrome, Firefox, and others or right click on the link and click on **Open Link**.

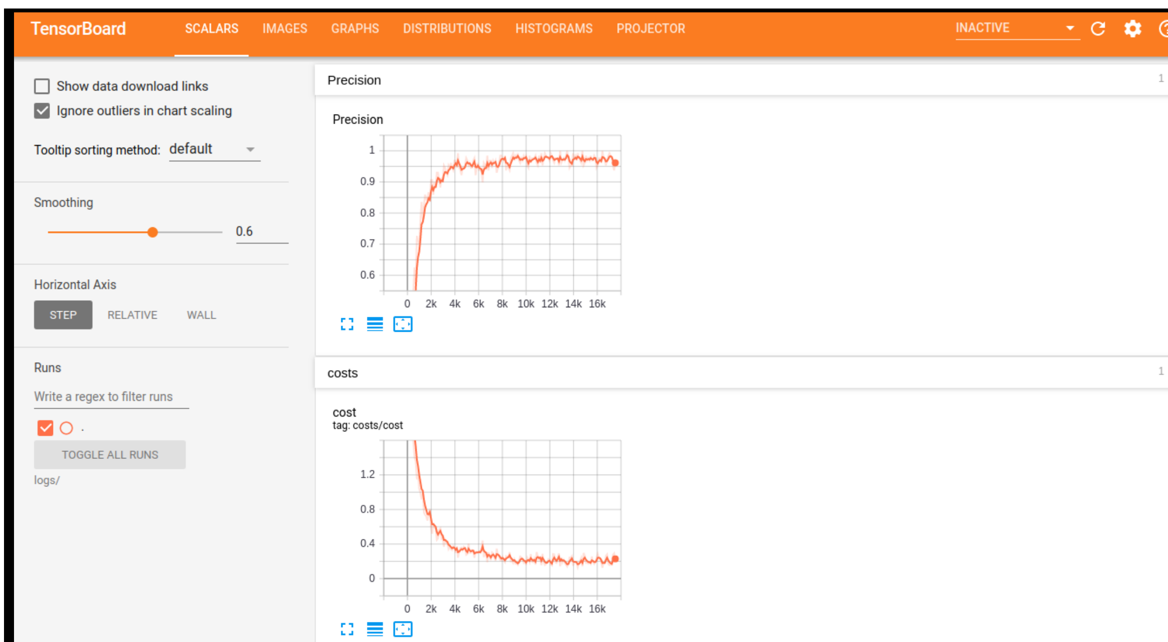


Figure A.49. TensorBoard – Link Default Output in Browser

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- Similarly, other graphs can be investigated from the available list.
- Check if the *checkpoint*, *data*, *meta* and *index* files are created at the log directory. These files are used for creating the frozen file (*.pb).

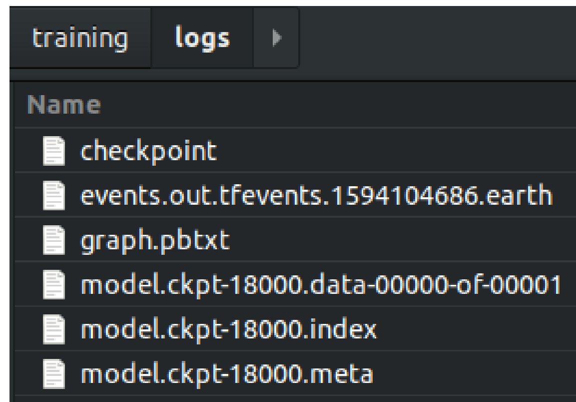


Figure A.50. Checkpoint Storage Directory Structure

A.5. Creating Frozen File

This section describes the procedure for freezing the model, which is aligned with the Lattice SensAI tool. Perform the steps below to generate the frozen protobuf file:

A.5.1. Generating .pbtxt File for Inference

Once the training is completed run below command to generate inference .pbtxt file.

Note: Do not modify config.sh after training.

```
$ python resnet_main.py --train_data_path=<TFRecord_root_path> --
log_root=<Logging_Checkpoint_Path> --train_dir=<tensorboard_summary_path> --
dataset='signlang' --image_size=64 --num_gpus=<num_GPUs> --mode=freeze
```

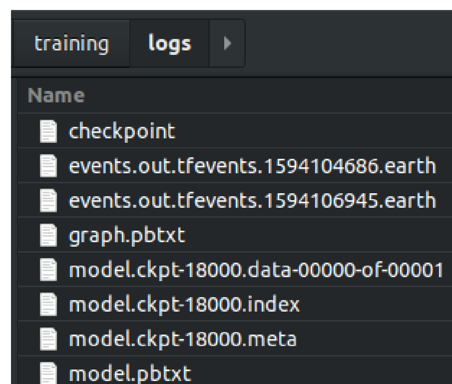


Figure A.51. Generated '.pbtxt' for Inference

- It generates the .pbtxt file for inference under the train log directory.

A.5.2. Generating the Frozen (.pb) File

```
$ python genpb.py --ckpt_dir <COMPLETE_PATH_TO_LOG_DIRECTORY>
```

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
inputShape shape [None, None, None, None]
inputShape shapes [None, None, None, None]
output_shapes of input Node [None, None, None, None]
 **TensorFlow**: can not locate input shape information at: random_shuffle_queue_DequeueMany
node to modify name: "random_shuffle_queue_DequeueMany"
op: "Placeholder"
attr {
  key: "dtype"
  value {
    type: DT_FLOAT
  }
}

--Name of the node - random_shuffle_queue_DequeueMany shape set to  random_shuffle_queue_DequeueMany [1, 64, 64, 1]
node after modify name: "random_shuffle_queue_DequeueMany"
op: "Placeholder"
attr {
  key: "dtype"
  value {
    type: DT_FLOAT
  }
}
```

Figure A.52. Run genpb.py To Generate Inference .pb

- *genpb.py* uses *.pbtxt* generated by procedure in the Generating .pbtxt File for Inference section and latest checkpoint in train directory to generate frozen .pb file.
- Once the genpb.py is executed successfully, the <ckpt-prefix>_frozenforinference.pb becomes available in the log directory as shown in below figure

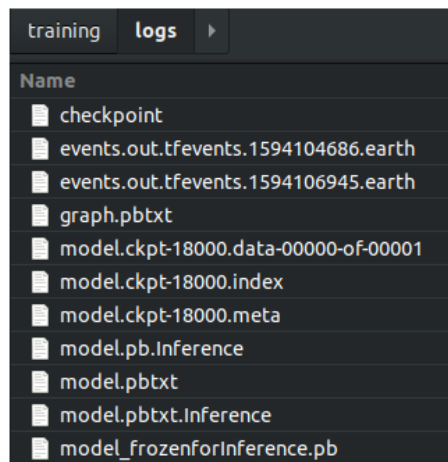


Figure A.53. Frozen Inference .pb Output

A.6. TensorFlow Lite Conversion and Evaluation

This section contains information for converting frozen pb to TensorFlow Lite model, quantize the model and evaluate on test dataset.

Note: It is recommended to use *Tensorflow 2.2.0 (CPU Only)* instead *Tensorflow 1.15.0* In TensorFlow Lite conversion flow.

Use Environment created from the Creating the TensorFlow Lite Conversion Environment section**.**

A.6.1. Converting Frozen Model to TensorFlow Lite

User can find "gen_tflite_and_quant.py" under training code which converts frozen model to TensorFlow Lite and also quantize it with INT8 quantization.

```
$ python gen_tflite_and_quant.py --input_path <sample images path> --tflite_path
<output tflite path> --pb <frozen pb file>
```

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Arguments information:

- --input_path: sample images that are used for quantization.
- --tflite_path: (default motor-model.tflite) output tflite path
- --pb: Frozen pb path

The command saves TensorFlow Lite at given path.

A.6.2. Evaluating TensorFlow Lite model

```
$ python evaluate_tflite.py --dataset_path <dataset_path> --tflite_path <tflite
path>
```

Argument information:

- --dataset_path: Test set path. Note that the labels should be (0, 1) for predictive maintenance.
- --tflite_path: tflite model path

The command shows accuracy on both classes.

A.6.3. Converting TensorFlow Lite To C-Array

```
$ xxd -i your-tflite-model-path.tflite > out_c_array.cc
```

The command generates c array at path given by user.

For detailed instructions on setting the Raspberry Pi, compiling the code, installing the client-end application, automating stack 2.0 bit file and generating binary, programming the Automate Stack on SPI Flash memory, troubleshooting the main system board, debugging using Docklight, OPCUA Modeler, and CSV file, refer to Automate Stack 2.0 Demo User Guide (FPGA-UG-02164).

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Appendix B. Setting up the Auto-Bootable MQTT-Based Client

To set up the auto-bootable MQTT-based client, perform the steps in the sections below.

B.1. Unzipping the Folder

To unzip the folder:

1. Open the *terminal.S*.
2. Unzip the bundle.
3. Run the command: **unzip Mqtt_Lattice_AutomateStack_2_0.zip.**

After unzipping the folder, perform the steps below:

1. Run the command: cd Mqtt_Lattice_AutomateStack_2_0/ lib/paho.mqtt.c/.
2. Run the command to clean the old package: **sudo make clean**.
3. Run the command for compilation: **sudo make && sudo make install**.

B.2. OpenSSL Error

Run the commands below if you get the Open SSL error:

```
sudo apt-get install libssl-dev
sudo make && sudo make install
```

B.3. Making the New Server Executable

To make the new server executable, run the following commands:

```
cd Mqtt_Lattice_AutomateStack_2_0/Scripts/
sudo make clean
sudo make
```

B.4. Installing the Mosquitto Broker

To run the mosquito broker, perform the steps below:

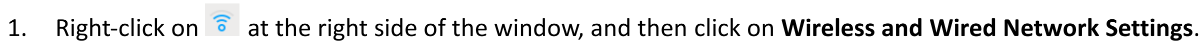
1. Run the commands below:
   ```
   sudo apt update && sudo apt upgrade
   sudo apt install -y mosquitto mosquitto-clients
   sudo systemctl enable mosquitto.service
   sudo nano /etc/mosquitto/mosquitto.conf
   ```
2. The *mosquitto.conf* file opens. Copy the two lines below at the end of this file.
   ```
   listener 1883
   allow_anonymous true
   ```
3. To save and close the file, press **ctrl+x**, type **y** and press **Enter**.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

B.5. Automating the Application

To automate the application:

1. Run the command: **sudo crontab -e**
2. Type 1 and press **Enter** to select the nano editor. The crontab opens.
3. Copy the line below and paste it in the crontab at the end of the file.



```
@reboot /home/pi/Mqtt_Lattice_AutomateStack_2_0/Scripts/autoapp.sh
```

4. Press **ctrl+x**, type **y** and press **Enter**.

B.6. Setting Up the IPV4 Address and Router on Raspberry Pi

To set up the IPV4 address and router:

1. Right-click on at the right side of the window, and then click on **Wireless and Wired Network Settings**.

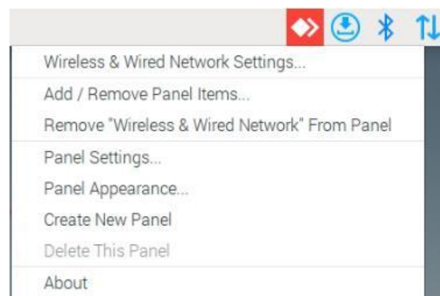


Figure B.1. IPV4 Address Setting

2. The Network Preferences screen is displayed. Select the **eth0** from the drop-down menu.

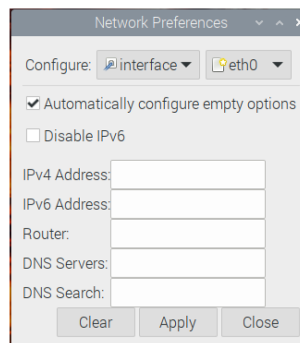


Figure B.2. Network Preferences Settings

3. Untick the *Automatically configure empty options* box.
4. Enter the IP address and router as shown in Figure B.3 and click **Apply**.
   - **IPV4 Address – 10.0.1.112**
   - **Router – 192.168.1.1**

   Note: Router value same as the value of Default gateway in your laptop.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

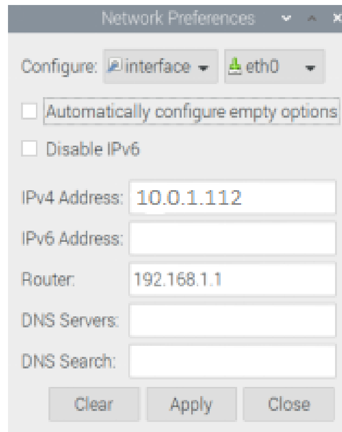


Figure B.3. IP Configuration

5. Run this command to reboot the Raspberry Pi: **sudo reboot**.

For details on connecting the laptop with the Raspberry Pi, refer to Automate Stack 2.0 Demo User Guide (FPGA-UG-02164).

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Revision History

Revision 1.0, June 2022

Section	Change Summary
All	Initial release.

© 2022 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

www.latticesemi.com