Research Problem Statement

Edge devices require efficient on-device intelligence to process visual data with minimal latency and energy consumption. Traditional approaches rely on cloud-based or centralized processing, leading to bandwidth constraints, security concerns, and increased power demands. To address this, near-sensor processing techniques, such as image data filtering and event-based data generation, offer promising solutions. However, existing architectures face challenges in optimizing these techniques at the circuit level while ensuring seamless communication between system components. This research investigates circuit-level implementations of near-sensor processing using FPGA-based prototyping and explores efficient interconnect solutions through automation protocol translation to enhance on-device intelligence.

Research Questions

1. How can near-sensor image data filtering and event-based data generation be optimized at the circuit level to improve on-device visual processing?
2. What FPGA-based digital circuit architecture can best prototype near-sensor processing for edge intelligence?
3. How can automation protocol translation enhance communication between components in an on-device intelligent chip, ensuring efficient and scalable integration?

Research Objectives

1. **Design and implement** near-sensor processing techniques for visual data, focusing on image data filtering and event-based data generation.
2. **Prototype and evaluate** digital circuit implementations on FPGAs to validate the effectiveness of near-sensor processing for edge devices.
3. **Develop and analyze** interconnect architecture incorporating automation protocol translation to optimize component communication in on-device intelligent chips.

Formalism of Proposed Framework

To formalize your chip architecture mathematically, we can define a framework that models the **on-device system** as a graph-based interconnect with well-defined communication parameters. Here's an approach:

1. System Representation as a Graph

We represent the chip architecture as a **directed graph** G=(V,E)G = (V, E)G=(V,E), where:

- VVV is the set of components (chiplets): V={Cf,Cp,Cm,Cai}V = \{ C_f, C_p, C_m, C_{ai} \}V={Cf,Cp,Cm,Cai} where
    - CfC_fCf = Near-sensor filter
    - CpC_pCp = Low-power control processor

- CmC_mCm = Memory unit
- CaiC_{ai}Cai = Specialized AI module

- EEE is the set of edges representing communication channels (interconnects) between these components.

2. Interconnect as a Weighted Graph

Each edge eij∈Ee_{ij} \in Eeij∈E represents a data exchange link between components CiC_iCi and CjC_jCj, with attributes:

- **Bandwidth**: BijB_{ij}Bij (bits per second)
- **Latency**: LijL_{ij}Lij (nanoseconds)
- **Energy Consumption**: EijE_{ij}Eij (joules per bit)

Thus, we define:

E={eij=(Ci,Cj,Bij,Lij,Eij)}E = \{ e_{ij} = (C_i, C_j, B_{ij}, L_{ij}, E_{ij}) \}E={eij=(Ci,Cj,Bij,Lij,Eij)}

3. Communication Protocol Translation

To ensure efficient communication, we introduce an **interconnect translation function** T\mathcal{T}T, which maps data formats and protocols between chiplets:

T:Pi×Di→Pj×Dj\mathcal{T}: P_i \times D_i \to P_j \times D_jT:Pi×Di→Pj×Dj

where:

- Pi,PjP_i, P_jPi,Pj are the protocol specifications of component CiC_iCi and CjC_jCj, respectively.
- Di,DjD_i, D_jDi,Dj are data representations (bit-width, encoding schemes, etc.).
- T\mathcal{T}T ensures compatibility across the interconnect.

4. Optimization Objectives

To optimize the interconnect, we minimize the **latency-energy product**:

min⁡∑(Ci,Cj)∈ELij⋅Eij\min \sum_{(C_i, C_j) \in E} L_{ij} \cdot E_{ij}min(Ci,Cj)∈E∑Lij⋅Eij

subject to:

- **Bandwidth constraints**: Bij≥BminB_{ij} \geq B_{\text{min}}Bij≥Bmin
- **Protocol compatibility**: T(Pi,Di)=(Pj,Dj)\mathcal{T}(P_i, D_i) = (P_j, D_j)T(Pi,Di)=(Pj,Dj)
- **Real-time processing limits**: ∑Lij≤Lmax\sum L_{ij} \leq L_{\text{max}}∑Lij≤Lmax

Scaling the protocol

Hardware Design System Flow for Protocol Translation Module

System Overview

The protocol translation module is integrated into an **adaptive interconnect** that automatically translates between I2C (Inter-Integrated Circuit) and AXI (Advanced eXtensible Interface). The system can scale from one-to-one (1:1) to many-to-many (N: N) communication scenarios, enabling seamless interoperability between heterogeneous chiplets.

a) Hardware Design System Flow

1. **Input Interface (I2C):**
    - Receives data from I2C-compatible chiplets.
    - Includes an I2C controller for start/stop conditions, clock synchronization, and data packet formatting.
2. **Protocol Translation Unit (PTU):**
    - Converts I2C data into an intermediate representation (IR).
    - Maps I2C fields (address, data, control) to corresponding AXI fields using a **mapping table** stored in memory (e.g., LUTs or SRAM).
    - Handles asynchronous data transfer and clock domain crossing.
3. **Adaptive Interconnect:**
    - Routes the translated data based on the destination address.
    - Employs **dynamic routing algorithms** for many-to-many communication.
    - Manages priority-based scheduling for high-priority chiplets.
4. **Output Interface (AXI):**
    - Converts the IR into AXI protocol.
    - Implements AXI features like burst transactions, pipelining, and QoS signaling.
5. **Control Unit:**
    - Monitors data flow, handles errors, and adapts routing paths dynamically.
    - Supports reconfiguration of protocol mappings during runtime.

Addressing Chiplet Interoperability Challenges

This approach solves the **interoperability problem** in the following ways:

1. **Protocol Mismatch Resolution:**
    - By employing a protocol translation unit (PTU), the system bridges the communication gap between heterogeneous protocols like I2C and AXI.
2. **Scalability:**
    - The many-to-many routing capability of the adaptive interconnect ensures that the architecture can scale as more chiplets are added.
3. **Low Latency and High Throughput:**
    - The direct hardware implementation of protocol translation minimizes latency compared to software-based translation.
    - Pipelining and parallel processing in the interconnect enhance throughput.
4. **Dynamic Adaptability:**
    - The reconfigurable nature of the PTL and adaptive routing mechanisms enables the system to respond to changing workloads and chiplet configurations.
5. **Energy Efficiency:**
    - Hardware-efficient components like LUTs and DSPs in the FPGA reduce power consumption compared to traditional CPU-based solutions.

Avenues for future study:

- Handling faulty chiplets

Approach to Scale Protocol Translation for Multiple Protocols

To scale the protocol translation system to adapt multiple protocols seamlessly, the architecture must incorporate a **modular, hierarchical, and adaptive design**. Here's the proposed technique:

1. Modular Multi-Protocol Translation Layer (MMPTL)

Design Approach

1. **Protocol Abstraction Layer (PAL):**
    - Introduce an intermediate **protocol-independent representation (PIR)** to abstract the specific characteristics of each protocol.
    - Each protocol (e.g., I2C, SPI, AXI, UART) is mapped to the PIR, which acts as the universal "language" for communication.
2. **Dynamic Protocol Mapper (DPM):**
    - Maintain a **mapping database** for translating each protocol's fields into the PIR.
    - Employ **lookup tables (LUTs)** or programmable logic for efficient real-time mapping.
    - Enable reconfigurability to support new protocols by dynamically updating the mapping database.
3. **Adaptive Interconnect:**
    - Route translated data to the destination protocol layer.
    - Implement **priority-based scheduling** and **dynamic routing** to handle traffic efficiently in many-to-many scenarios.
4. **Protocol-Specific Translators (PSTs):**
    - Provide specialized hardware modules for translating the PIR back into the destination protocol format (e.g., AXI, I2C).
    - Use modular design to scale the system by adding or removing PSTs as needed.

---

2. Scaled System Features

- **Protocol Support:** Supports n protocols without requiring pairwise mappings ($O(n^2)$ complexity). Instead, use a centralized PIR approach (O(n) complexity).
- **Reconfigurability:** Hardware modules can be updated or swapped to support emerging protocols.
- **Scalability:** Handles multiple simultaneous translations (many-to-many communication).

Math Model:

Scaling in Action

Steps to Scale

1. Add new protocols by extending MMM with new mapping functions ϕ(pn+1)\phi(p_{n+1})ϕ(pn+1).
2. Use reconfigurable hardware (e.g., FPGAs) to implement new PSTs without disrupting existing components.
3. Dynamically adapt GGG by adding new nodes (VnewV_{new}Vnew) and edges (EnewE_{new}Enew) while recalculating routing paths.

Example

For a system with I2C, AXI, SPI, and UART protocols:

1. Map I2C fields to PIR (UUU).
2. Translate PIR to AXI for the destination chiplet.
3. Introduce SPI and UART by updating MMM with new mappings ϕ(p3),ϕ(p4)\phi(p_3), \phi(p_4)ϕ(p3),ϕ(p4).
4. Adjust the graph GGG by adding nodes and edges for the new chiplets.

Key Benefits of the Model

1. **Efficiency:** Linear scaling of mapping database compared to quadratic complexity in pairwise translations.
2. **Flexibility:** Dynamically adapts to new protocols and topologies.
3. **Performance:** Optimized interconnect routing ensures minimal latency and energy overhead.

APTLink v1.0

I have implemented a mathematical model of automatic protocol translator in an FPGA. The translator is called APTLink v1.0 and it is embedded into an interconnect fabric for chiplets. The main aim of the interconnect fabric is to seamlessly interconnect chiplets with different communication protocols in a SiP (System-in-package). The APTlink has a FIFO-based receiver node that converts the transmitting protocol into an intermediate representation (IR) this representation extracts and presents the protocol's information into the header and payload. The header consists of fields like the chiplet ID, the destination chiplet ID, and the transaction type (Write or Read, control) while the payload contains the actual data in a lone field. The receiver node arranges each chiplet protocol info in fifo-based reservation structures from which the IR processing unit can read the header field of a FIFO at a given time. Within the IR processing unit, the arbiter uses the transaction type and chiplet destination ID to determine the type of transaction and the destination chiplet. The arbiter unit in the IR processor uses this information to connect the transmitting chiplet to the receiving chiplet. The receiver IR node receives the data and enqueues in its header and payload fifos which are easily accessible and suitable to the receiving chiplet. The way I have implemented a validation hardware system prototype of this design such that I used IP1 for chiplet 1 which is an I2C data generation module, the APTLink translation wrapped in an interconnect module, and the IP2 is an AXI-based storage IP. The top module has a control command processing logic that starts the system with a reset signal connected to the external reset switch. At the start, IP1 generates the data and streams to the interconnect, the APTlink translator within the interconnect translates and targets IP2. The command processor reads transmitted data and received data and then compares the two. If the data matches, the data pass signal is asserted as connected to an LED on the FPGA board, similarly, the data fail signal is asserted otherwise. 1000 sample data cases were tested for this setup.

Onboard PMBus/I2C-Based Power Monitoring

- **PMBus Sensors**: Many FPGA boards include PMBus-compliant power sensors (e.g., TI INA230, LTC2977) for real-time voltage/current measurement.
    - **Implementation**:
        1. Instantiate an **I2C controller IP core** in the FPGA to communicate with PMBus sensors.
        2. Use PMBus commands (e.g., READ_VIN, READ_IOUT) to fetch power data.
        3. Log power metrics (average, peak) during APTLink operation.
    - **Tools**: Xilinx AXI IIC IP or custom I2C state machines.

To address **chiplet interoperability** in a System-in-Package (SiP), a mathematical and systems engineering approach is essential. Chiplets are heterogeneous dies (e.g., CPUs, GPUs, memory, accelerators) integrated into a single package, and their interoperability challenges span **communication protocols, power/thermal management, signal integrity, and standardization**. Below is a structured framework to model and solve these problems:

---

1. Define the Interoperability Problem

- **Key Challenges**:
    - **Communication Protocols**: Mismatched signaling standards (e.g., PCIe vs. UCIe).
    - **Power Delivery**: Voltage/current mismatches or noise coupling.
    - **Thermal Coupling**: Heat dissipation conflicts between chiplets.
    - **Physical Interfaces**: Die-to-die (D2D) interconnect density, latency, and bandwidth.
    - **Testing/Validation**: Lack of standardized pre-silicon verification.

---

2. Mathematical Modeling Approaches

A. Communication Protocol Standardization

- **Graph Theory**: Model inter-chiplet communication as a network graph.
    - Nodes = Chiplets, Edges = Interconnects (e.g., TSVs, microbumps).
    - Optimize topology for latency/bandwidth using shortest-path algorithms (e.g., Dijkstra's).
- **Queuing Theory**: Analyze traffic congestion and buffer requirements.
    - Model data flow as M/M/1$M/M/1$ queues with arrival rate $\lambda\lambda$ and service rate $\mu\mu$.

B. Signal Integrity and Power Delivery

- **Transmission Line Models**:
    - Use telegrapher's equations to characterize D2D interconnects:

$\partial V\partial x = -L\partial I\partial t, \partial I\partial x = -C\partial V\partial t \partial x\partial V = -L\partial t\partial I, \partial x\partial I = -C\partial t\partial V$

    - Simulate reflections and crosstalk using S-parameters.
- **Power Distribution Network (PDN) Optimization**:
    - Model PDN as a resistive-capacitive network, minimizing IR drop $\Delta V = I \cdot R \Delta V = I \cdot R$.
    - Use linear programming to allocate power budgets across chiplets.

C. Thermal Management

- **Heat Equation**:

∂T∂t=α∇2T+qk∂*t*∂*T*=*α*∇2*T*+*kq*

where T*T* = temperature, α*α* = thermal diffusivity, q*q* = heat flux.

- **Finite Element Analysis (FEA)**: Simulate thermal gradients and hotspots in 3D SiP layouts.

D. Standardization and Interface Compatibility

- **Compatibility Matrices**:
    - Represent protocols (e.g., UCIe, BoW) as binary matrices where Cij=1*Cij*=1 if chiplet i*i* is compatible with j*j*.
    - Solve for maximal compatibility using linear algebra or constraint programming.

---

3. Practical Implementation Strategies

A. Adopt Universal Standards

- Use **Universal Chiplet Interconnect Express (UCIe)** or **Bunch of Wires (BoW)** for D2D communication.
- Define common power/ground (P/G) and clock distribution architectures.

B. Co-Design Chiplets and Interconnects

- Optimize interconnect pitch and materials (e.g., organic vs. silicon interposers).
- **Multi-Physics Co-Simulation**:
    - Combine SPICE (electrical), FEA (thermal), and mechanical stress models.

C. Adaptive Power Management

- Use **reinforcement learning (RL)** to dynamically adjust voltage/frequency:

Policy: π(s)=arg⁡min⁡V,f(Pleakage+Pdynamic)Policy: *π*(*s*)=arg*V*,*f*min(*P*leakage+*P*dynamic)

- Implement distributed voltage regulators (LDOs) near each chiplet.

D. Pre-Silicon Verification

- **Digital Twins**: Create virtual SiP prototypes for interoperability testing.
- **Formal Methods**: Verify protocol compliance using temporal logic (e.g., Linear Temporal Logic, LTL).

---

4. Case Study: AMD's EPYC Processors

- **Problem**: Integrating Zen cores, I/O dies, and memory chiplets.
- **Solution**:
    - **Infinity Fabric**: Standardized D2D protocol for low-latency communication.
    - **3D V-Cache**: Optimized TSV density and thermal modeling for stacked cache.
    - **Power Management**: Adaptive voltage scaling via embedded sensors.

---

5. Key Tools and Standards

- **Tools**: ANSYS HFSS (signal integrity), Cadence Voltus (power), Synopsys 3DIC Compiler.
- **Standards**: UCIe, HBM (memory), IEEE 1801 (power formats).

---

6. Future Directions

- **Machine Learning**: Predict interoperability failures using graph neural networks (GNNs).
- **Quantum Computing**: Optimize SiP layouts with quantum annealing (e.g., D-Wave).
- **Open-Source Ecosystems**: Collaborative frameworks like CHIPS Alliance.

Summary

By combining **mathematical models** (graph theory, PDEs, optimization) with **industry standards** (UCIe) and **multi-physics simulations**, chiplet interoperability can be systematically addressed. The goal is to balance performance, power, and thermal constraints while enabling a plug-and-play ecosystem for heterogeneous integration.

Approach 2

To mathematically model **protocol synthesis** for chiplet interoperability in a System-in-Package (SiP), you need to formalize the communication requirements, constraints, and interactions between heterogeneous chiplets. Protocol synthesis involves designing rules and signaling mechanisms that ensure seamless data exchange, timing synchronization, and resource sharing. Below is a structured approach to model this problem mathematically:

---

1. Define the Problem Space

- **Key Challenges**:
    - **Mismatched protocols**: Different chiplets may use distinct communication standards (e.g., UCIe vs. PCIe).
    - **Timing constraints**: Clock domain synchronization and latency requirements.
    - **Resource contention**: Shared buses, buffers, or power rails.
    - **Formal verification**: Ensuring synthesized protocols are deadlock-free and meet specifications.

---

2. Mathematical Frameworks for Protocol Synthesis

A. Finite State Machines (FSMs)

- **Model states and transitions**:
    - Define states for each chiplet (e.g., idle, transmitting, receiving).
    - Use transition rules to enforce protocol handshakes (e.g., request-acknowledge).
    - **Example**: A 2-chiplet handshake protocol:

Stransmitter:{Idle,Sending,Wait_ACK}Sreceiver:{Idle,Receiving,Send_ACK}*S*transmitter*S*receiver :{Idle,Sending,Wait_ACK}:{Idle,Receiving,Send_ACK}

Transitions governed by Boolean guards (e.g., data_valid ∧ ¬busy).

B. Temporal Logic (LTL/CTL)

- **Specify timing requirements**:
    - Use **Linear Temporal Logic (LTL)** or **Computation Tree Logic (CTL)** to formalize safety/liveness properties.
    - **Example**:

AG(req→AF ack)(Every request must eventually be acknowledged).AG(*req*→AF *ack*)(Every request m ust eventually be acknowledged).

C. Petri Nets

- **Model concurrency and resource allocation**:
    - Represent protocol steps as places and transitions.
    - Use token flow to enforce sequencing and resolve conflicts (e.g., bus arbitration).
    - **Example**: A Petri net for shared bus access:

P={Bus_Free,Chiplet1_Req,Chiplet2_Req},T={Grant1,Grant2}.*P*={Bus_Free,Chiplet1_Req,Chiplet2_Req},*T*={Grant1,Grant2}.

D. Automata Theory for Protocol Synthesis

- **Automated synthesis**:
    - Use **ω-automata** (e.g., Büchi automata) to generate protocols from temporal logic specifications.
    - Tools like **SPIN** or **TuLiP** synthesize protocols that satisfy LTL constraints.

E. Optimization Models

- **Minimize latency/power**:
    - Formulate protocol synthesis as an optimization problem with constraints:

Minimize∑i,jLatency(Ci↔Cj)Subject toBandwidth≥Bmin,Power≤Pmax.MinimizeSubject to*i*,*j*∑Latency(*Ci*↔*Cj*)Bandwidth≥*B*min,Power≤*P*max.

    - Use **integer linear programming (ILP)** or **genetic algorithms** for Pareto-optimal solutions.

---

3. Step-by-Step Protocol Synthesis

Step 1: Formalize Requirements

- Capture specifications in temporal logic (e.g., "No two chiplets transmit simultaneously").

G(¬(TX1∧TX2))G(¬(TX1∧TX2))

Step 2: Model Chiplets as FSMs

- Represent each chiplet's behavior as an FSM with inputs (e.g., data_in), outputs (e.g., data_out), and synchronization signals (e.g., clk).

Step 3: Compose Parallel Automata

- Use **synchronous product** to combine individual FSMs into a global system model.

Aglobal=A1×A2×⋯×AnAglobal=A1×A2×⋯×A*n*

- Verify compatibility using **model checking** (e.g., NuSMV).

Step 4: Synthesize Protocol Controllers

- **Supervisory Control Theory (SCT)**:
    - Automatically generate a supervisor (protocol) that restricts chiplet behaviors to meet specifications.
    - Solve for the **maximally permissive** controller using algorithms like **Ramadge-Wonham**.

Step 5: Optimize for Physical Constraints

- Map logical protocols to physical interconnects:
    - Use **graph embedding** to assign protocol states to die-to-die (D2D) links.
    - Optimize wirelength and parasitics via **mixed-integer programming (MIP)**.

---

4. Example: Synthesizing a Memory Access Protocol

- **Problem**: Two chiplets (CPU and GPU) contend for shared HBM memory.
- **Specifications**:
    - Mutual exclusion: G(¬(CPU_Access∧GPU_Access))G(¬(CPU_Access∧GPU_Access))
    - Fairness: GF(CPU_Access)∧GF(GPU_Access)GF(CPU_Access)∧GF(GPU_Access)
- **Synthesis**:
    - Model as a **Büchi automaton** and synthesize a round-robin arbiter using **GR(1)** synthesis tools.
    - Optimize arbitration latency with ILP.

---

5. Verification and Validation

- **Model Checking**:
    - Verify synthesized protocols against LTL/CTL properties (e.g., deadlock freedom).
- **Co-Simulation**:
    - Integrate protocol models with SPICE (electrical) and FEA (thermal) simulations.
- **Formal equivalence checking**:
    - Ensure synthesized RTL matches the abstract FSM.

---

6. Tools and Standards

- **Synthesis Tools**: TuLiP (Temporal Logic Planning), SPIN, Cadence JasperGold.
- **Verification**: NuSMV, UPPAAL.
- **Standards**: UCIe (Universal Chiplet Interconnect Express), AXI (Advanced eXtensible Interface).

---

7. Case Study: Intel's Advanced Interface Bus (AIB)

- **Protocol Synthesis**:
  - AIB uses a **latency-insensitive**, packetized protocol with credit-based flow control.
  - Synthesized via finite-state machines and validated using LTL properties.
- **Key Features**:
  - Modular die-to-die PHY layer.
  - Formal guarantees on bandwidth and deadlock avoidance.

---

8. Future Directions

- **Machine Learning for Synthesis**:
  - Use reinforcement learning (RL) to explore protocol design spaces.
  - Example: Train an RL agent to minimize arbitration latency.
- **Quantum Protocol Synthesis**:
  - Leverage quantum annealing to solve NP-hard optimization in protocol mapping.

Summary

To mathematically model protocol synthesis for chiplet interoperability:

1. Formalize requirements using **temporal logic**.
2. Model chiplets as **FSMs/Petri nets**.
3. Synthesize protocols via **automata theory** or **SCT**.
4. Optimize for physical constraints (latency, power).
5. Verify correctness with **model checking**.

This approach bridges abstract specifications (e.g., "no deadlocks") to concrete RTL implementations, enabling robust, standardized communication in heterogeneous SiPs.

Comparison: AMD, Intel, NVIDIA Chiplet Interconnects

Feature	AMD Infinity Fabric	Intel EMIB + UCIe	NVIDIA NVLink-C2C
Type	Proprietary interconnect for AMD chiplets	Open-standard interconnect for multi-vendor chiplets	High-speed GPU & accelerator interconnect
Heterogeneous Chiplets?	❌ No (Only AMD-designed chiplets)	✅ Yes (Supports 3rd-party chiplets with UCIe)	✅ Limited (Mainly GPU/AI-focused)
Multi-Vendor Support?	❌ No	✅ Yes (With UCIe)	❌ No (Mostly NVIDIA hardware)
Target Applications	CPUs, GPUs, APUs, AI chips	CPUs, FPGAs, AI, general-purpose chiplets	AI/ML accelerators, GPU scaling
Protocol Support	AMD-specific protocol	PCIe, CXL, AXI, Custom	NVLink, PCIe
Future-Proofing	❓ May adopt UCIe in future	✅ UCIe ensures long-term interoperability	❓ Unclear if UCIe will be supported

Application-Specific Chiplet Interconnect Methodology

This methodology explains the approach to synthesize a structured interconnect for an Application Specific System-in-package (AS-SiP). Core to an optimized structured interconnect is the chiplet characterization stage which provides initial information about the specific chiplets on the SiP and the holistic functional nature of the SiP. Thus, for every given application, the synthesis of chiplet intercommunication will be tailored to its specific needs at design time. After important chiplet specifications such chiplets operational frequency, required transmission bandwidth, or power constraint have been obtained in the characterization phase, interoperability constraints will be defined which will guide the entity synthesis and the selection of the physical link interconnect fabric. The result of this phase will be a structured interconnect with a tailored protocol translation for seamless interoperability satisfying the critical needs of the AS-SiP.

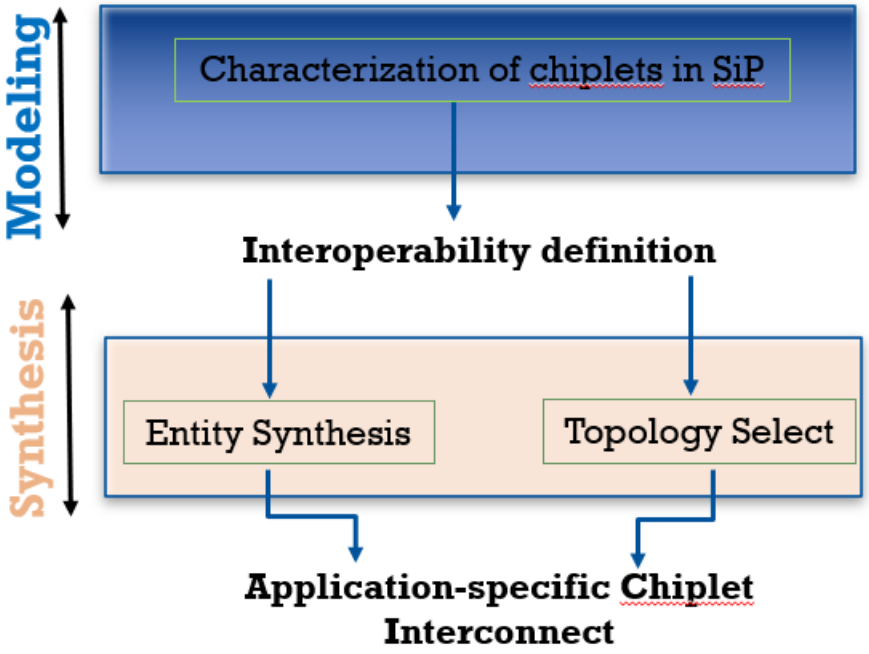

Figure 1. Synthesis Flow of an AS-SiP Chiplet Interconnect

Characterization of Chiplets in an AS-SiP

As part of the modeling phase, this stage provides the unique functionality signature of the SiP by deep analysis of its constituent chiplets. The main goal of this process is to model an objective performance parameter for which the created interconnect will be constrained to, this will ensure that the developed structured interconnect is the optimized one for seamless interoperability between the chiplets to perform the task of the AS-SiP efficiently. For a given SiP, the aspects of analysis during the modelling include *chiplet functionality, data flow, power constraints, and interface analysis*. This section of this chapter detailly explains how this parameter can be used to model the SiP performance constraint parameters for optimization.

Functional and Dataflow Analyses:

The objective of functional analysis is to determine the primary function of each chiplet In the SiP. At the end of the functional analysis, the behavioral representations of each chiplet are extracted and these representations guide the dataflow analysis. The dataflow analysis uses the representation of the different chiplets to establish data movement requirements on the Sip. This analysis generates a communication graph for the chiplets referred to as *chiplet communication graph, CCG*. The CCG is a vital component in the characterization phase because it is a customized representation of data movement for a particular application specific SiP, this reduces the generic bidirectional directed graph (BDG) for chiplets communication which indicates a many-to-many communication pattern. From the unique CCG for a particular AS-SiP, the structure of the interconnect topology can be deduced (i.e. is it a partially connected crossbar, ring, mesh, memory bus, NoC etc.).

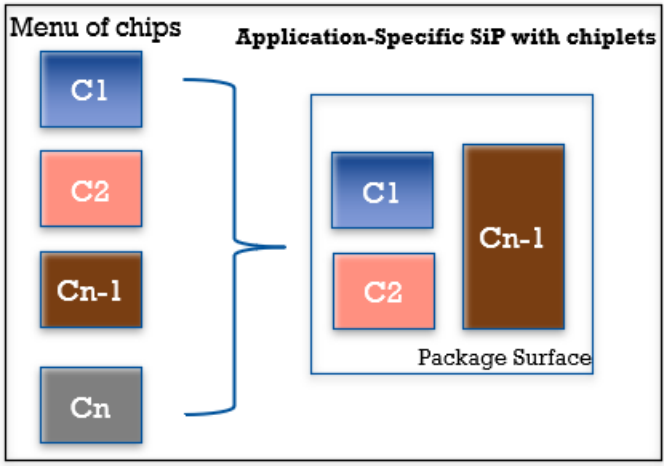

Figure 2. Heterogenous Chiplets Makeup an AS-SiP

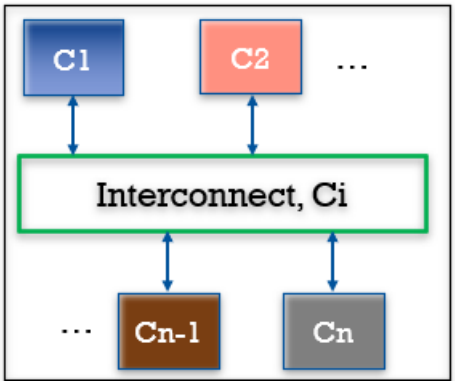

Figure 3a. Generic Chiplete Interoperability of an AS-SiP

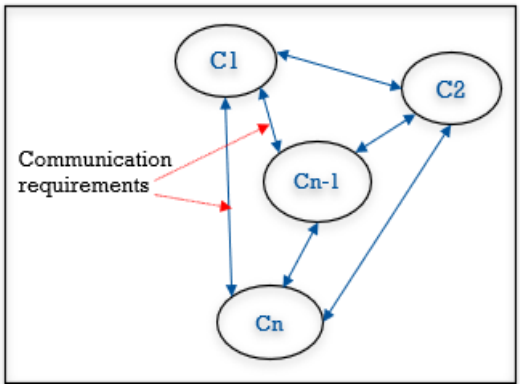

Figure 4b. Chiplet Communication Graph (CCG)

From figure 2, to build an AS-SiP, off-the-shelf chiplets are picked from a menu of chiplets. To carefully craft efficient interoperability support for this SiP, the characterization phase is important. After the functional and dataflow analysis, the interoperability definition is established with the bidirectional directed graph (CCG) as a conveying representation (figure 3b). The viability of this

modeling will be carefully demonstrated in the case study example at the in the second part of this chapter. For the functional and dataflow analysis, we have the following definitions.

Definition 1: Components of an AS-SiP, C

For a given application-specific system-in-package, the components on the SIP are a set of selected chiplets which may be heterogenous in terms of primary function and communication protocols/interfaces.

$$\text{Components, } C \mid\text{-> } C \in \{C1, C2, Cn\text{-}1, Cn\}$$

Where:

- C1, C2, Cn-1, Cn are heterogenous chiplets
- N = natural numbers greater than 2.

Definition 2: Chiplet Interoperability, I

For the SiP to function effectively, the chiplets must interact with each other according to the specification requirements of the system. After characterization of the system, an Interoperability definition tailored to that AS-SiP, such

$$I = \{C, T\}$$

Where:

- C defines the set of components,
- T, the transmission communication requirements according to the dataflow analysis. Characterized principally by latency and bandwidth requirements of each chiplet-to-chiplet link.

At the end of this stage of the characterization phase, and objective interoperability definition, Iobj will exist such that a given the design of the chiplet interconnect will be constrained to. The goal at the end is to ensure that I_i is close to I_obj as possible.

$$I_i <= Iobj$$

Where:

I_i is the interoperability function for a realized chiplet interconnect for that AS-SiP, and

Iobj is the objective interoperability function.

Power Analysis

The objective of the power analysis is to define the power constrain of the SiP's interconnect. To achieve this, each chipets rated power is referenced to deduce the sum of chiplet power. For a given AS-SiP, the power constraint must be defined as a general specification requirement

depending on the application. This target system power, and the total chiplet power can be used to constrain the interconnect synthesis.

Definition 3: Power of Interconnect System, PIS

The general specification power requirement of a given AS-SiP.

Target System Power, TSP = Power of Interconnect System (PIS) + Total Component Power (TCP)

Where:

- TCP = Sum of power (C1 + C2 + Cn-1 + Cn)

Thus,

$$PIS = TSP - TCP$$

At the end of the power analysis stage, the objective PIS will be defined, such that for that AS-SiP, which will constraint the interconnect system. This will ensure that the synthesis of an optimal Chiplet interconnect structure for the SiP.

$$PIS_i <= PiS_obj$$

Where:

PIS_i is the actual power consumption of the structured interconnect.

Interface Analysis:

The interface characterization specifies what interfaces and streaming protocols are unique to each chiplet on the SiP. From low compute to high performance and memory units, all chiplets are be characterized to establish whether serial or parallel, packet-based, or streaming mechanisms are employed. The result of the interface characterization guides the *Entity Synthesis* step which is one part of the Synthesis phase as shown in figure 1. The entity synthesis stage builds specific protocol receiving nodes for each chiplet based on the interface analysis. Similarly, the topology synthesis stage syntheses appropriate topology using results from the functional and dataflow analysis.

At the end of the characterization, the interoperability objectives are defined. These definitions are critical for the synthesis of the unique entities for each component as will be seen section () protocol translation logic. Also, the structured topology-select is guided by the modeling (characterization) phase.

Synthesis

The main goal of the synthesis phase is to synthesize the entities receiving nodes for each component (chiplet) and to select the appropriate structured topology. This phase is implemented with two-concurrent stages *entity synthesis* and *topology-select*.

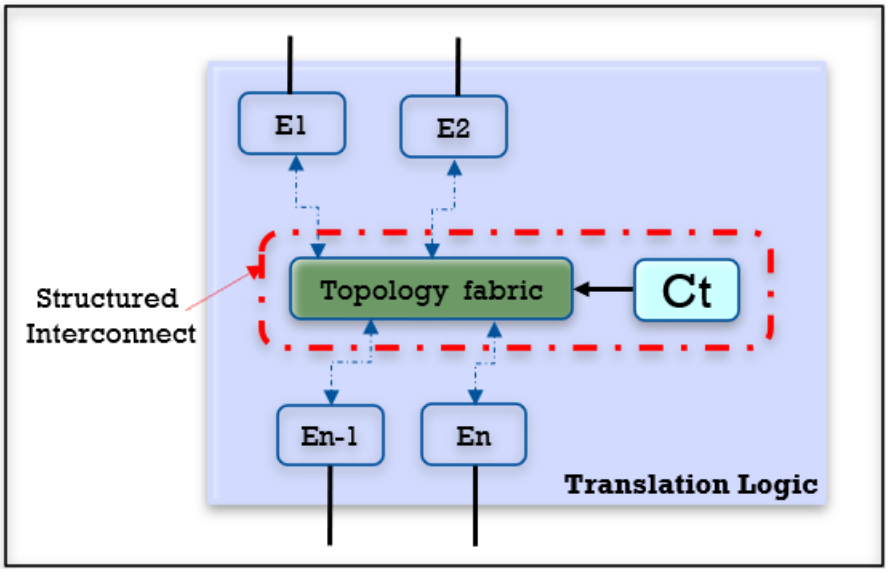

Figure 5. Synthesis of Chiplet Interconnect

Entity Synthesis

The unique entities are synthesized based on the characterization of the chiplets. Each entity has a primary function to convert protocol-specific data to an intermediate or universal representation (IR). At the level of topology-fabric, every chiplet's during outstanding communication is in the IR format, thus every chiplet involved in that communication link can properly transmit and receive data irrespective of its protocol. The entity is a bidirectional agnostic protocol module.

Definition 4: Intermediate Representation, IR

For AS-SiP with a set of chiplets, C E {C1, C2, ... Cn-1, Cn} with corresponding protocol set P E {p1, p2, ... pn-1, pn}, there exist a set of entities E E {E1, E2, ... En-1, En} such that a translation function can defined

$$IR = E(P)$$

In the case study example illustration (section), the details of the hardware features used to realize this translation will be explained.

Mechanics of the Entity Synthesis process:

The unique entity synthesis is defined as the entity module is a receiver node for a given chiplet's protocol. When this node receives data, two FIFO-based buffer structures split the data to separate the address information (destination chiplet) and data information in case of a write. For a read transaction, the entity receives data in the IR format and arranges the data to suit the connected chiplet's protocol. Similarly, two FIFO-based structures are utilized to provide both address and data information.

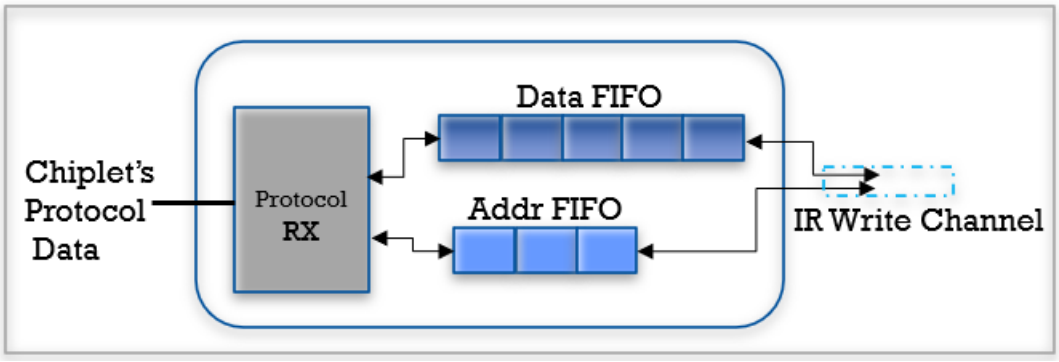

Figure 6. Entity Mechanism

The Protocol receiving module, protocol RX, receives the

Topology Select:

The Topology fabric is defined as,