Chiplet-Package Co-Design For 2.5D Systems Using Standard ASIC CAD Tools

MD Arafat Kabir, Yarui Peng
Computer Science and Computer Engineering Department
University of Arkansas, Fayetteville, AR, US

***Abstract*—Chiplet integration using 2.5D packaging is gaining popularity nowadays which enables several interesting features like heterogeneous integration and drop-in design method. In the traditional die-by-die approach of designing a 2.5D system, each chiplet is designed independently without any knowledge of the package RDLs. In this paper, we propose a Chip-Package Co-Design flow for implementing 2.5D systems using existing commercial chip design tools. Our flow encompasses 2.5D-aware partitioning suitable for SoC design, Chip-Package Floorplanning, and post-design analysis and verification of the entire 2.5D system. We also designed our own package planners to route RDL layers on top of chiplet layers. We use an ARM Cortex-M0 SoC system to illustrate our flow and compare analysis results with a monolithic 2D implementation of the same system. We also compare two different 2.5D implementations of the same SoC system following the drop-in approach. Alongside the traditional die-by-die approach, our holistic flow enables design efficiency and flexibility with accurate cross-boundary parasitic extraction and design verification.**

***Keywords*—2.5D Design, Chip-Package Co-Design, Redistribution Layer Planning, Package Design, Track Assignment.**

I. Introduction

In the current industry approach of designing 2.5D systems, all functional circuit blocks are designed independently in their own design environments and then mounted on the package RDLs as a complete system [1]. The design and planning of the package are also performed independently with very little knowledge of the circuit blocks it houses [1, 2]. The analysis and optimization of both chiplets and the package are conducted separately [1] in the current design flow. For system reliability and signal integrity, accurate timing and power analyses of the entire system are essential. Though it is possible to perform the whole system analysis using the profiles of individual chiplet and package, it cannot accurately consider the interactions among tightly-connected components.

The die-by-die design flow is currently the most practical approach that leverages 2.5D integration technology. Using this flow, a designer can achieve the shortest possible design time using off-the-shelf chiplets to implement a 2.5D system. Fig. 1(a) shows the traditional die-by-die design flow for 2.5D systems. In this flow, chiplets and the package never actually interact with each other until after they are fabricated and assembled. All steps of design and optimization are performed independently in their own environments. However, it is not the best flow to design a complete system that can achieve maximum performance with highest reliability and analysis accuracy. While designing a complete system for 2.5D integration, to obtain the best partition, the partitioning steps need to be aware of the redistribution layers (RDL) of the 2.5D package. To optimize the floorplan and routing for RDLs, the planner may need to make small modifications to the chiplet pin arrangements. The optimization steps of an individual chiplet also need to consider package routing as well as the other chiplets. In short, to extract the best performance

This material is based upon work supported by the National Science Foundation under Grant No. 1755981. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

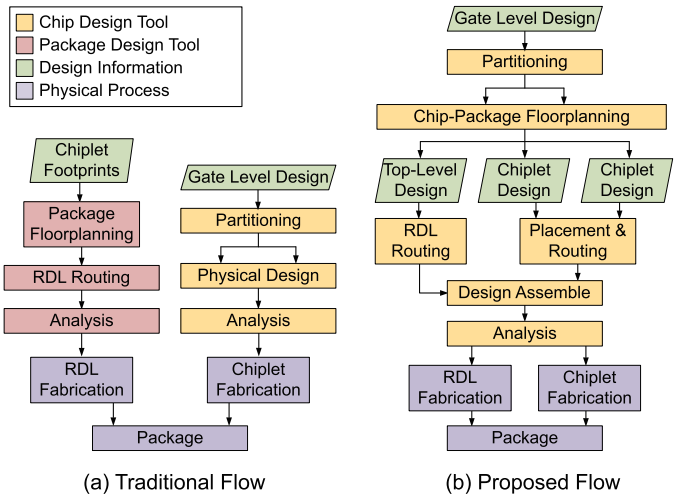

Fig. 1. The traditional Die-by-Die design flow of a 2.5D system versus our proposed holistic approach with standard ASIC tools

out of the system, optimizations in each design step need to be performed in a holistic [3] way rather than optimizing individual parts independently without taking into account rest of the system.

In this paper, we propose a design flow that incorporates the features required to achieve the Chip-Package Co-Design goals in 2.5D integration technology leveraging the industry-standard chip design tools. Fig. 1(b) shows the overall steps of our proposed flow. In this flow, we design the 2.5D package together with the chiplets in the same design environment of the existing commercial chip design tools. This enables exchanging design information between the chiplets and the package during the optimization steps, which is essential to achieve the co-design goals. To illustrate our design flow, we use an ARM Cortex-M0 based simple SoC system and a modified Nangate45nm PDK that can handle the chiplets and package routing layers together in a chip design environment. First, we present some of existing partitioning schemes compatible with 2.5D package and the results applying these schemes in our example system. Next, we discuss challenges in the floorplanning stage of a 2.5D system and demonstrate our algorithms and in-house tools to perform RDL planning and routing within the chip design environment. Further, we present the rest of the physical design steps of the individual chiplets and the package. Our flow supports the drop-in design approach enabled by 2.5D integration, which we illustrate by designing two versions of the same SoC system with different memory sizes. In the results section, we present comparisons between the 2.5D system with a monolithic 2D implementation of the same system and between the two versions of the 2.5D system designed using the drop-in approach.

We claim the following contributions: (1) A new flow to design chiplets and the package of 2.5D systems together using existing commercial chip design tools taking into account the impacts of package layers on system performance; (2) A new strategy to perform the floorplanning and routing of the package and pin placement of

978-1-7281-4123-7/20/$31.00 © 2020 IEEE

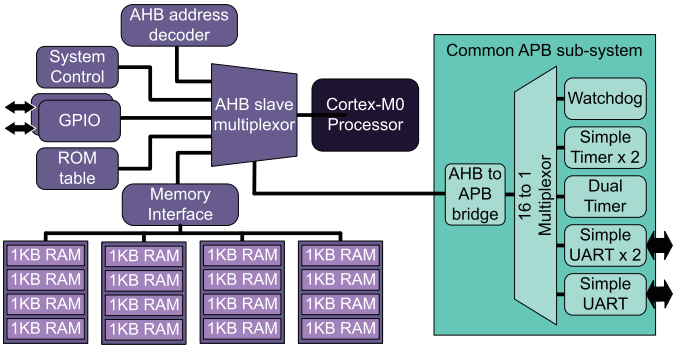

Fig. 2. System architecture of the ARM Cortex-M0-based example design.

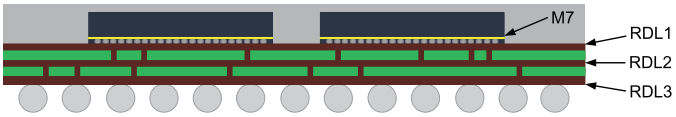

Fig. 3. Package redistribution layer stack

TABLE I
TECHNOLOGY PARAMETERS OF ROUTING LAYERS

	M6	via6	M7	via7	RDL1	viar1	RDL2	viar2	RDL3
Height	2.28	3.08	3.9	7.5	12.5	17.5	22.5	27.5	32.5
Thickness	0.8	0.82	3.6	5	5	5	5	5	5
Width	0.4	0.4	2	5	10	10	10	10	10
Spacing	0.4	0.44	2	10	10	20	10	20	10

the chiplets that reduces package routing issues without affecting chiplet floorplans significantly; (3) A new parastic extraction and STA analysis flow with the entire 2.5D system, chiplets and package together; (4) A comparison between two 2.5D systems designed validating the our drop-in approach taking into account the key factors needed for optimization of the entire system. To our best knowledge, there exists no tool flow that places and routes chiplets and package of 2.5D systems together using commercial chip design tools.

II. Design and Technology Settings

A. System Architecture

As a proof of concept, we design a simple microcontroller system based on ARM Cortex-M0 core using our design flow. Fig. 2 shows the system architecture of the example design. This SoC consists of 16KB RAM and other common peripheral devices. The processor core is connected to the rest of the system through an AHB bus. The AHB bus connects the processor to a system controller, two low-latency GPIO modules, a ROM table, an address decoder, the memory interface, and a subsystem of APB peripheral and APB infrastructure. The APB subsystem consists of a watchdog timer, two simple timers, one dualtimer, and three UART modules. The 16KB memory system is divided into four banks of 4KB each. Each memory bank is again subdivided into four memory blocks of 1KB each. Such breakdown of the memory system is intentional to add flexibility during the partition stage. We use OpenRAM [4] memory compiler to compile the 1KB memory modules with one-byte word size.

B. Technology Settings

We use Nangate45nm as our Process Design Kit in the physical design of the chiplets and the monolithic 2D chip. We are using up to metal7 for chiplet routing. For package design, we modify the top three layers of Nangate45nm PDK to simulate the attributes of 2.5D package redistribution layers. Table I shows our settings for the top routing layers. The layer via7 corresponds to the micro bumps connecting the chiplets to the package top layer named RDL1. The layer RDL3 corresponds to the bottom redistribution layer of the package, where the C4 bumps will be attached. The package redistribution layers are illustrated in Fig. 3.

C. Reference 2D Design

Before moving on to 2.5D system design flow, we design a monolithic 2D version of our example system as a reference design. Traditional chip design flow is very mature and existing commercial IC design tools make it very straight forward. When the system architecture is ready, Gate Level Netlists are generated by the synthesis tool. We use Synopsys Design Compiler to perform the synthesis. Then this gate-level netlist is imported into the chip design environment for performing physical design. We use Cadence Innovus to perform the Physical design of the chip. In the floorplanning stage, at first we allocate an area for the large modules like memory macros. Then we perform the floorplanning of the standard logic modules. The next step is to design the power delivery network of the system. In this step, power and ground wires are routed around and across the chip core, macro blocks and other modules as needed. In the standard cell placement stage, the logic cells are assigned their physical location by the placement tool based on the floorplan and the connectivity among the cells. The next step is the time design and optimization of the system which includes pre-CTS timing optimizations, Clock Tree Synthesis (CTS) and post-CTS timing optimizations. Then the design is ready to be routed. We use the routing tool of Cadence Innovus to perform the chip routing. After routing, post-routing timing optimizations are performed. Finally, filler cells and metal fills are inserted to cover the empty space of the chip. When the design is complete, Design Rule Checks (DRC) are performed to make sure that the chip is designed following the foundry specified rules. This concludes the physical design steps. The next step is to perform analysis of the system to ensure the signal integrity and reliability of the system. We use Synopsys StarRC to perform the parasitics extraction and Synopsys PrimeTime to perform the timing and power analysis of the chip. Fig. 7(d) shows the finished 2D design.

D. Proposed 2.5D Design Flow

Our proposed design flow is demonstrated in Fig. 1(b). Synthesis step is the same as the traditional 2D flow which generates the gate-level netlists. Then the partitioner takes the gate level netlists and the partitioning scheme settings to generate the chiplet netlists which fulfill the scheme requirements. The next step is to prepare the floorplans for the package and the chiplets. We perform the floorplanning of the package and chiplets together in the same design environment within a chip design tool. The design environment takes in the chiplet netlists and the modified PDK that includes the package layers. We perform the floorplanning and pin assignments of the chiplets and the package in a way that reduces the impacts of package routing on system performance. At the end of this step we get the package floorplan, chiplet floorplans, timing budgets and the connectivity among the chiplets. Then the package and the chiplets are routed and optimized individually. The physical design flow of the chiplets is exactly the same as that of the traditional 2D flow. After standard cell placement and routing if a chiplet passes the Design Rule Checks, it is ready to be integrated with the rest of the system. The Design Assemble step takes all the chiplets and the package design to assemble the whole system. With the entire system in place, we extract the parasitics, considering the interactions among the routing layers across the chiplets and the package. Finally, the

analysis step takes in the netlist and parasitics information of the assembled system to perform the full system analysis.

III. Partition

After performing synthesis of the system, it needs to be partitioned into chiplets [1]. In 2.5D packaging, chiplets are interconnected through package wires. This introduces some new factors in the partitioning problem not present in the partitioning stage of the monolithic 2D technology. While performing partition, these factors need to be considered carefully as the overall system performance is highly dependent on them.

In general, the main objective of the partition problem is to minimize the cut size among the partitions keeping a specified area balance. However, the min-cut solution may not be the best solution for 2.5D system performance. A larger cutsize may be preferred to a slower system. While performing the partition, we need to avoid cutting the critical paths as much as possible without exceeding the number of interconnects each chiplet can accommodate. Better results might be obtained if the architecture is designed keeping package layers in mind. However, the partitioner needs to account for package RDL wires while exploring solutions that might be obtained through making minor changes (*e.g.* resizing buffers) in the partition netlists. To understand the impact of partition on 2.5D systems, we study some of the existing partition schemes compatible with this integration technology. As our example design is a small SoC system, we partition the system into two chiplets. The result of our study is shown in Table II.

Balanced Partition: The first partition scheme that we study aims at achieving minimum cut size keeping a good area balance. To implement this scheme, we use hMetis [5] and FLARE [6] as partitioning algorithms. While hMetis is purely a k-way hypergraph partitioning algorithm, being a performance-driven partitioner, FLARE takes into account the local and global interconnect delays during performing the partition. This difference is reflected in Table II. For both the partitions, hMetis has a lower pin count compared to FLARE. However, FLARE requires less number of buffer cells and also consumes less power and takes less area to achieve the same speed of operation compared to hMetis.

Memory vs. Logic Partition: The next scheme is based on the natural boundary between memory and logic. We keep all the standard logic cells in one partition and all the memory macros in the other partition. From Table II we can see that the memory partition consists of almost 90% of the total area. This scheme is a good example of 2.5D integration because it takes advantage of the heterogeneity between the memory and logic implementation technologies. Among all the partition schemes, we are able to achieve the highest speed of operation in this scheme for our example design. However, the number of pins for the logic partition cannot be accommodated due to the small area. Moreover, it requires the second-highest number of buffer cells among all the schemes to achieve that speed.

Architecture-Aware Partition: In the last partition scheme, we try to utilize the knowledge of the system architecture to come up with a reasonable partition. In this scheme, one of the partitions contains half the memory macros with all logic cells while the other partition contains the rest half of memory macros. The first partition can be considered as the core system while the other partition can be considered as a memory extension of the core system. This scheme gives a good area balance and a reasonable pin count for both partitions. However, it requires the highest number of buffer cells and the total area to achieve the same speed of operation as the balanced partition schemes.

TABLE II
Comparison of Partition Schemes

Paritition Scheme	hMetis	FLARE	Mem/Logic	Arch-Aware
Max Frequency	300 MHz	300 MHz	333 MHz	300 MHz
Power	6.19 mW	6.17 mW	6.73 mW	6.13 mW
No. of Buffers	2,152	1,907	2,383	2,521
Cell Area (μm^2)	274,450	273,944	275,726	275,902
Area Balance	49.4/50.6	49.8/50.2	89.7/10.3	55.2/44.8
Pin Count	257/313	374/370	191/228	141/110

We select the Architecture Aware Partition scheme for implementing using our design flow because the sizes of both partitions are large enough to accommodate their pins.

IV. Chip-Package Floorplanning

Traditionally, floorplans of package and chiplets are prepared independently. However, the best floorplans for individual chiplets, without any knowledge and consideration of the RDLs, might have pin configurations that create package routing congestions and long package wires due to detours and unequal bus wire delays between the chiplets. All of these deteriorate system performance. In this step of our flow, we try to design the package floorplan, chiplet pin configurations, and their connectivity together in a way that minimizes all of the aforementioned package-routing-related issues without significantly affecting the floorplans of individual chiplets. Note that the chiplet pin configurations generated applying our strategy do not disregard the flexibility and parallelism of independent chiplet design. The pin configurations just describe the connectivity between the chiplet pins locations. The actual signal assignment can be performed during the placement and routing stage of individual chiplets if necessary.

In this step, first we need to determine the chiplet dimensions and pin arrangement. Pin arrangement information includes the number of rows and columns in the pin array, pin pitch, pin dimensions, and physical pin locations in the chiplet. When these parameters are determined and fixed for all the chiplets, RDL planning can start. We code an RDL planner program that implements our strategy of Chip-Package floorplanning. The RDL planner takes in the chiplet netlists and technology information to generate package floorplan and interconnect routing, chiplet footprints, and connectivity information among chiplet pins. When floorplanning and signal assignment of a chiplet is finished, the planner can map the signals of the pins of the planned chiplet to pins of other chiplets using this connectivity information. At present, it can perform signal mapping only if one of the two chiplets has already finished signal assignment to all of its pins and the other chiplet has not been assigned signal to any of its pins. Our interconnect routing and chiplet connection strategy is described in the latter part of this section.

A. RDL Planning

In our RDL planning stage, we consider two chiplets at a time. For the sake of illustration, we assume a cut-line between the two chiplets, perpendicular and across l RDLs as shown in Fig. 4. If there are n connections between these two chiplets, it requires total n tracks across l RDLs crossing the cut-line to route all the connected pins between these chiplets. These n tracks occupy the least routing area if we arrange them uniformly in l layers using the minimum track width and spacing. In our strategy, before assigning signals to chiplet pins, we first route all the pins of a chiplet to their nearest available tracks crossing the cut-line using the shortest possible wire. Then we determine a relative position between the chiplets that produces sufficient overlap of these tracks for connecting n pins. Finally, we

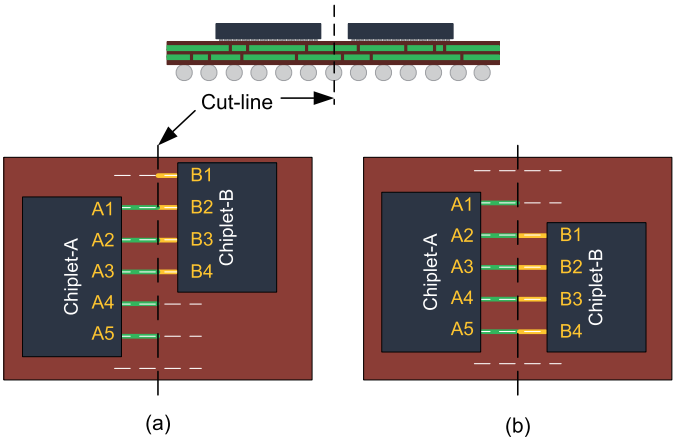

Fig. 4. Illustration of the RDL planner strategy. (a) Rejected floorplan while finding the relative location (b) Selected solution that satisfies the pin connectivity requirement

connect the pins of the two chiplets which are routed to the same track crossing the cut-line. The unconnected pins of the chiplets are assigned the package I/O signal and are routed during the package routing stage. Algorithm 1 describes our RLD planning strategy.

Fig. 4 illustrates this strategy for connecting four pins ($n = 4$) between the chiplets using only one RDL ($l = 1$). The dashed white lines show the available tracks crossing the cut-line and the thick lines connected to the chiplets show the assigned tracks to the chiplet pins. At first, Chiplet-A and Chiplet-B pins are routed to the nearest tracks crossing the cut-line. Next, while finding the relative position between the chiplets, the floorplans similar to Fig. 4(a) are rejected as it does not have sufficient track overlap for four connections. Finally, among two viable solutions, we arbitrarily pick the floorplan in Fig. 4(b) which supports the required number of connections between the chiplets. And then we define connectivity between pin A2 of Chiplet-A and pin B1 of Chiplet-B because they are routed to the same track. Similarly, pins A3, A4, and A5 of Chiplet-A will be connected to pins B2, B3, and B4 of Chiplet-B, respectively. The pin A1 of Chiplet-A, which is not connected to any other chiplets, can be used for package I/Os.

So far, our RDL planner can handle only one-to-one pin connections between two chiplets. More sophisticated planning like clock tree synthesis of the package, power distribution network design of the entire system, etc. should be performed in this step. However, our planner cannot handle those tasks at the moment of writing this paper.

B. Track Assignment

Assignment of the cut-line crossing tracks to the chiplet pins is performed in two steps: (1) Pin routing to the chiplet boundary and (2) Track assignment to these boundary locations. Lines 8-14 of Algorithm 1 describes our track assignment strategy. Before a pin can be routed to a pin of another chiplet it needs to cross its chiplet boundary. In the first step, we bring as many internal pins as we can to the chiplet boundary using all the RDL routing tracks crossing the boundary. We name the boundary locations where the pins are routed to as "Boundary Points." Next, we assign tracks to these Boundary Points in the Track assignment step. For each chiplet, track assignment is performed following a greedy strategy where the Boundary Points closest to the cut-line is assigned its nearest track first. As a result, in the order of track assignment, the Boundary Points on the side facing the cut-line comes first, then the Boundary Points on the sides perpendicular to the cut-line come in the increasing order

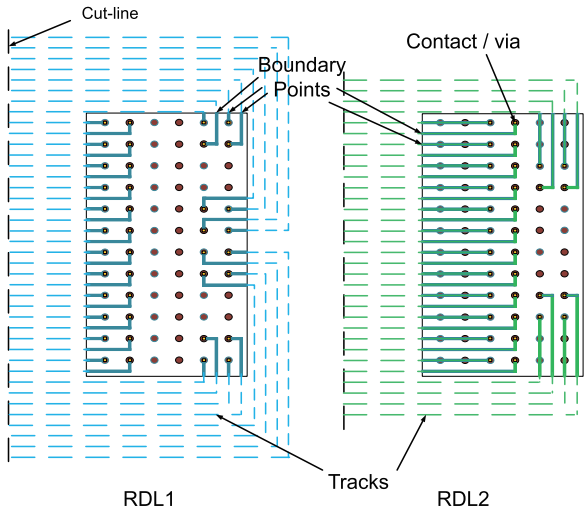

Fig. 5. Track assignment result for a chiplet with 12×6 pin array, two package routing layers, and pin pitch of two tracks

of their distances from the cut-line and the Boundary Points on the side of the chiplet opposite to the cut-line comes the last.

Fig. 5 shows the Track Assignment result for a chiplet with 72 pins. The pins are arranged in a grid of 12 rows and 6 columns with a pin pitch of two tracks. There are two RDLs available for routing the chiplet pins. The solid lines connected to the pins show the route to the Boundary Points and the horizontal dashed lines show the RDL tracks crossing the cut-line. At first, the pins on the side nearest to the cut-line are routed to the Boundary Points of that side using both the routing layers. Then, the pins on the sides perpendicular to the cut-line are routed to the Boundary Points on those sides. The rest of the pins are routed to the Boundary Points on the side opposite to the cut-line. Boundary Points of the opposite side is least preferred because of the detours needed to reach the cut-line. After routing all the pins to the Boundary Points, tracks are assigned to them in the increasing order of their distance from the cut-line. The vertical short dashed lines show the track assignment of the Boundary Points that cannot be directly connected to the package routing tracks.

Algorithm 1: RDL Planning Algorithm

```
1  Input: Chiplet netlists and PDK
2  Output: RDL Floorplan and Routing Script
3  Calculate area required for the chiplets
4  Generate pin array according to pin pitch and chiplet area
5  Draw a cut-line between the chiplets
6  sideOrder = [ near cut-line, top, bottom, opposite side ]
7  layerOrder = [ RDL layers from bottom to top ]
8  foreach Chiplet do
9      foreach s in sideOrder do
10         foreach l in layerOrder do
11             Route pins to the Boundary Points of s on l
12     bpOrder = sort(Boundary Points, key=dist. from cut-line)
13     foreach bp in bpOrder do
14         Assign the nearest available track to bp
15 while Floorplan not valid do
16     Floorplan = New relative position of the chiplets
17     Check if Floorplan is valid for connecting the chiplets
18 Create TCL script for pin assignment and RDL routing
19 Return Floorplan and Routing Script
```

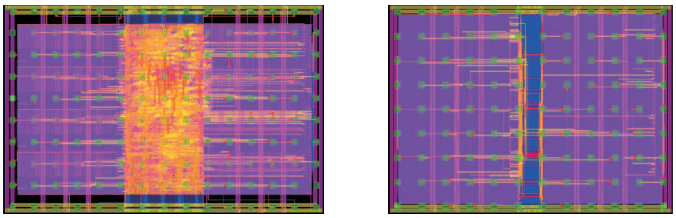

Fig. 6. Chiplets layouts for 2.5D integration

V. Placement and Routing

After the Package floorplan, Chiplet pin configurations and connectivity information are ready, the placement and routing of individual chiplet and package can be performed using any commercial chip design environment that supports hierarchical design flow. We use Cadence Innovus to perform the placement and routing. In our flow, At first, we load the whole system, including the package, into the design environment. During the partition stage, in the netlist, each chiplet is defined as an individual module. So, these modules appear as regular 2D modules in the design environment. Then we define the chiplet modules as partitions using the same steps of defining partitions in 2D design flow. When the package floorplan and chiplet pin configurations are prepared according to the plan generated by the RDL planner, each chiplet can be designed independently. When the chiplet designs are completed, their interface timing models can be extracted. The package design can be performed either using the extracted models or the entire chiplet designs into consideration. However, because of the differences in the chip and package routing techniques, currently we are not using the chip routing tools to route the package. Rather, we are using the routing generated by our RDL planner for connecting the chiplets and manual routing for package I/O pins.

A. Hierarchical Splitting

At first, we set up the chip design environment with the PDK containing both chip and package routing layers. Then we load the partitioned gate-level netlists in this design environment. We prepare the package floorplan as generated by the RDL planner. Then we perform trial route to determine the signal assignment to the chiplet pin locations. We define a routing blockage in the package region outside the chiplets, so that the router uses only the package routing layers (RDL1, RDL2, and RDL3 in our design) outside the chiplets while performing the trial route. In 2D chip design tools, pins are assigned on the boundary of the partitions based on trial route. From this pin assignment result and the connectivity information generated by the RDL planner, we determine the signal assignment for the chiplet pins. Because of this co-planning, we can get a pin configuration for the chiplets, which minimizes package routing issues without affecting the chiplet floorplans significantly. Then we derive the time budget for the chiplets which takes into account the routing through the package layers. Finally, we commit the partitions (chiplets) and push them down for individual physical design. After this step, all the chiplets can be designed independently in their own design environments.

B. Chiplet Placement and Routing

In this step, each chiplet is designed and optimized as a single chip using 2D design techniques. The floorplan can be adjusted as required as long as the pin configuration remains the same as determined in the Floorplanning stage. After fixing the final floorplan, we perform power distribution network design of the chiplets. When PDN design

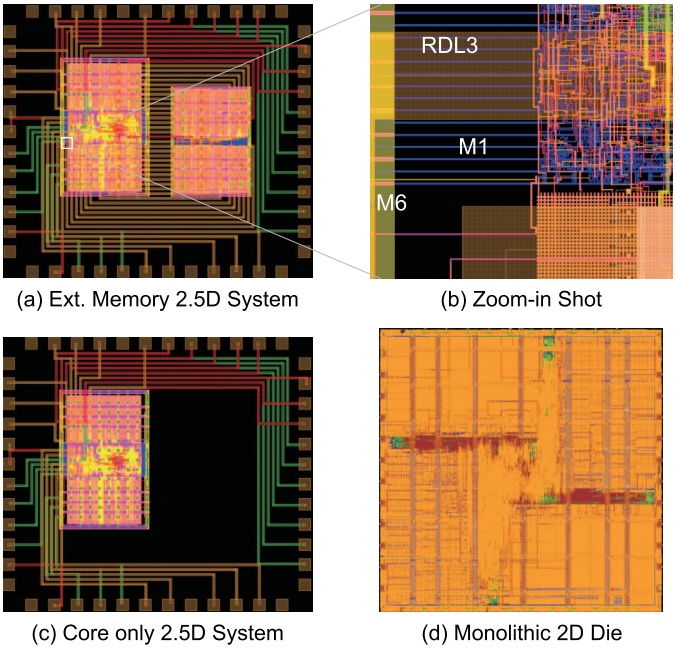

Fig. 7. Design layouts of the 2.5D systems and the monolithic 2D design. (a) Assembled 2.5D system with extended memory, (b) zoom-in shot of the assembled 2.5D system, (c) assembled core-only system, and (d) monolithic 2D reference design.

is done, we use the placement tool to perform the cell placement. Then we perform the time design steps which include clock tree synthesis and timing optimization of the chiplet. We run the routing tool to finish the signal routing and then perform the post-route timing optimizations. Lastly, we add filler cells and metal fills to finish the design. Fig. 6 shows the routing of the two chiplets in our design. As mentioned in Section III, we are using the Architecture Aware partition scheme. The chiplet in Fig. 6(a) is the Core System partition with the logic blocks and 8KB of memory and the chiplet in Fig. 6(b) is the Extended Memory partition with extra 8KB memory. We design one system with the Core chiplet only and another system with both the chiplets using our design flow.

C. Package Routing

When the chiplet designs are complete, we can extract their interface timing models which can be used while performing the routing and optimizations of the package. Because of the differences in the package and chip routing techniques, IC routing tools cannot produce good routing for the package routing layers. Based on the strategy explained in Section IV, Our RDL planner can generate routing scripts for commercial tools like Cadence Innovus. We are using this generated script to complete the connections between the chiplets pins. Currently, our RDL planner cannot handle the techniques like clock tree synthesis, 45-degree routing, etc. which can be applied while performing the package routing. After inter-chiplet routing is done, we are manually routing the rest of the I/O pins of the chiplets to the package I/O pads. Fig. 7(a),(c) shows the package routing of the two 2.5D systems we designed. For power and ground connection, power and ground planes can be created which are not used in our design.

D. Design Assemble

After chiplets are designed, DRC is performed on each chiplet separately in the same way as in the traditional 2D flow. DRC of the package is also performed before integrating it with chiplets.

TABLE III
COUPLING CAPACITANCES (IN FF) BETWEEN ROUTING LAYERS

	M1-M5	M6	M7	RDL1	RDL2	RDL3
M1-M5	3625	479.1	22.52	58.16	9.889	7.547
M6	479.1	533.7	84.89	101.1	11.62	10.85
M7	22.52	84.89	26.68	14.84	1.739	1.663
RDL1	58.16	101.1	14.84	297.1	1009	41.49
RDL2	9.889	11.62	1.739	1009	297.4	1076
RDL3	7.547	10.85	1.663	41.49	1076	513.1

Using clean-DRC chiplets, the entire system is assembled in the chip design environment with the modified PDK in the same way partitions are assembled in the 2D design flow. At this stage, using the drop-in approach we design two 2.5D systems with the chiplets. The same package is designed to host both chiplets. In one system we just include the Core chiplet that has only 8KB of memory and can perform all the tasks within this memory capacity. The other system with both the chiplets has 16KB memory which is suitable for memory-intensive applications. Our design flow supports this drop-in design approach enabled by 2.5D integration technology. Fig. 7(c) shows the Core Chiplet Only system, and Fig. 7(a) shows the assembled memory extended system. Fig. 7(b) shows a zoomed-in view of the assembled system, which clearly shows the wires from the chiplet and package together. The horizontal wires marked M1 are Core Chiplet wires that are connecting the power and ground rails of the cell rows to the metal6 wire marked as M6. The M6 wire is a part of the power ring around the chiplet core. The RDL3 wire is a package wire connecting one of the package I/O pads with a Core Chiplet pin. After this design-assemble step, we can use the chip extraction tools to capture parasitics of the whole system and perform analysis to determine the system performance and reliability.

VI. HOLISTIC ANALYSES RESULTS

Industry-standard flow uses FEM tools to perform package extraction with S-parameters to determine package signal and power integrity. However, in our flow, both package and chiplets are in the same design environment after the design assemble stage. Therefore, we can extract the distributed parasitic netlist of the entire chip-package system, which is the key to more accurate and reliable signal integrity analysis. We extract the parasitics with coupling capacitances from the assembled system using StarRC from Synopsys. Table III shows the extraction result. For readability, we lumped the coupling capacitances among layers M1-M5 in one column and one row. The columns for RDL1, RDL2, and RDL3 show the coupling capacitance between package layers and chiplet layers. The coupling of package layers with M7 is low because of a less number of wires on M7. However, there exists significant coupling with the wires on M6, which is captured in the parasitics extraction process.

We use the extracted parasitics information to perform timing and power analysis with Synopsys PrimeTime. Then we compare the results with the monolithic 2D implementation of the system shown in Fig. 7(d). This comparison is shown in Table IV. Though this small SoC is chosen to illustrate the flow, we can still get some important information from the comparison. The timing analysis result considers the impact of RDLs and so for the 2.5D system the highest system frequency we achieve is 245 MHz which is much worse compared to the maximum frequency (333 MHz) for the 2D implementation. To achieve this performance target, a large number of buffers are needed. As a result, the cell count of the Core Chiplet Only is almost double the cell count of the entire 2D die. The overall silicon area and chip wire length in both implementations are comparable. However, the power number for 2D die is greater than that of the chiplets of 2.5D system because of the higher system frequency.

TABLE IV
COMPARISON OF DIE/CHIPLET ANALYSIS RESULTS

Chip Design	2D Die	Core Chiplet	Ext. Mem Chiplet
Standard Cells#	35,904	51,733	11,531
Total Wirelength	412.99 mm	350.89 mm	40.143 mm
Die Size (μm×μm)	550×550	390×590	350×470
System Frequency	333 MHz	245 MHz	
Chip Power	10.6 mW	7.751 mW	0.194 mW

TABLE V
COMPARISON BETWEEN THREE IMPLEMENTATIONS OF THE EXAMPLE SYSTEM IN 2D/2.5D TECHNOLOGY

System Design	Core-Only System	Core with Extra Memory
Chip-Package Cap	120.786 fF	217.409 fF
Max Frequency	300 MHz	245 MHz
System Power	9.578 mW	8.26 mW
Pakacage wirelength	35.41 mm	94.03 mm
Package Size	1.3 mm × 1.15 mm	

Moreover, this table illustrates that the design and optimization steps performed using the existing chip design tools have taken into account the impact of the RDLs. And that is the main goal we want to achieve with this flow. Table V shows the comparison between these two 2.5D systems. The Chip-Package coupling capacitance is larger for the extended system because of more package wires. The critical timing path for the extended system is between the core and memory chiplets. In the absence of the extra memory chiplet, we could achieve a higher system frequency for the Core-Only system.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our flow to design chip and package together using existing commercial chip design tools for 2.5D integration. As a proof of concept, we designed a very simple and small system using our design flow. Our analysis results show a number of design issues that our flow can address while the traditional flow cannot. One of these is to co-optimize chiplets and package floorplan and routing. We present an RDL planning and routing algorithm to solve this problem. Results show that our design flow is compatible with all standard ASIC CAD tools, but able to consider chiplet and package altogether during physical design, parasitic extraction, timing and power analysis. Further, our flow also supports the drop-in approach, enabling design flexibility with efficient chiplet integration. In future work, more sophisticated algorithms and techniques will be studied to support clock tree synthesis, non-manhattan routing, and one-to-many routing on package layers.

REFERENCES

[1] J. Kim, G. Murali, H. Park *et al.*, "Architecture, Chip, and Package Co-design Flow for 2.5D IC Design Enabling Heterogeneous IP Reuse," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: ACM, 2019, pp. 178:1–178:6.

[2] W. Liu, Min-Sheng Chang, and T. Wang, "Floorplanning and signal assignment for silicon interposer-based 3D ICs," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[3] Y. Peng, T. Song, D. Petranovic, and S. K. Lim, "Parasitic Extraction for Heterogeneous Face-to-Face Bonded 3-D ICs," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 7, no. 6, pp. 912–924, June 2017.

[4] M. R. Guthaus, J. E. Stine, S. Ataei *et al.*, "OpenRAM: An Open-source Memory Compiler," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: ACM, 2016, pp. 93:1–93:6.

[5] G. Karypis and V. Kumar, "Multilevel k-way Hypergraph Partitioning," *VLSI Design*, vol. 11, no. 3, pp. 285–300, 2000.

[6] Jason, J. Cong, S. K. Lim, and C. Wu, "Performance Driven Multi-level and Multiway Partitioning with Retiming," in *Proceedings of the 37th Annual Design Automation Conference*, ser. DAC '00. New York, NY, USA: ACM, 2000, pp. 274–279.