Optimization of Behavioral IPs in Multi-Processor System-on-Chips

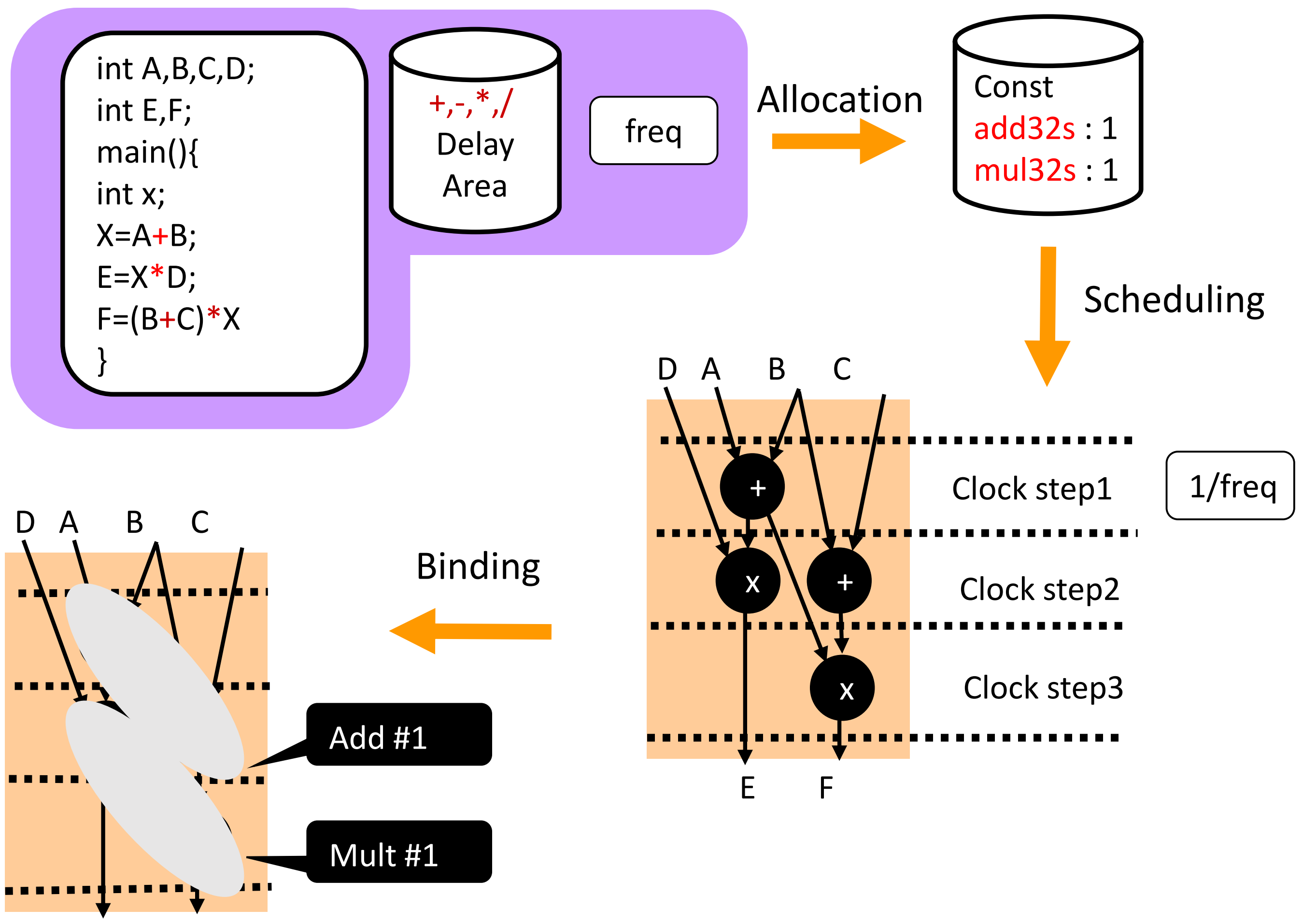
Yidi Liu and Benjamin Carrion Schafer[#]
Department of Electronic and Information Engineering
b.carrionschafer@polyu.edu.hk [#]

Design Automation and Reconfigurable Computing LAB dARC

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

Outline

High Level Synthesis 101



```
int A,B,C,D;
int E,F;
main(){
int x;
X=A+B;
E=X*D;
F=(B+C)*X
}
```

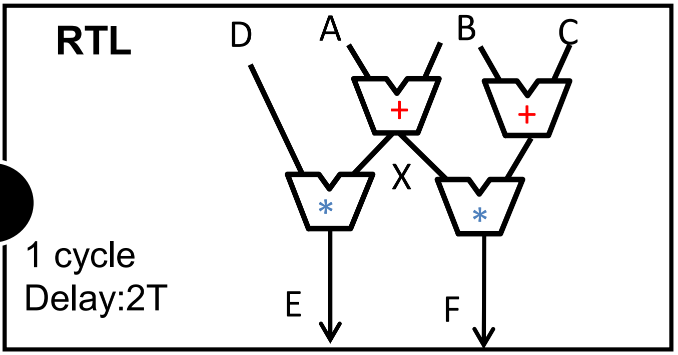
Benefits of HLS: Automatic Alternative Architecture Generation

Behavioral Description in C

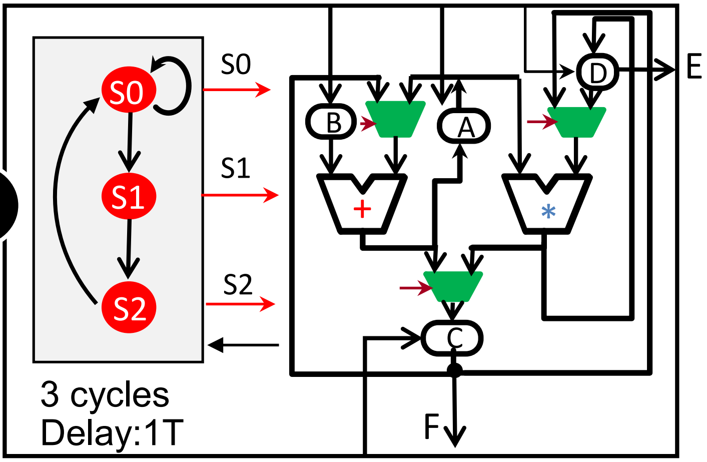
```
char A,B,C,D;
char E,F;
main(){
char X;
X = A + B;
E = X * D;
F = (B + C) * X;
}
```

FU constraints

1. + : 2, * : 2

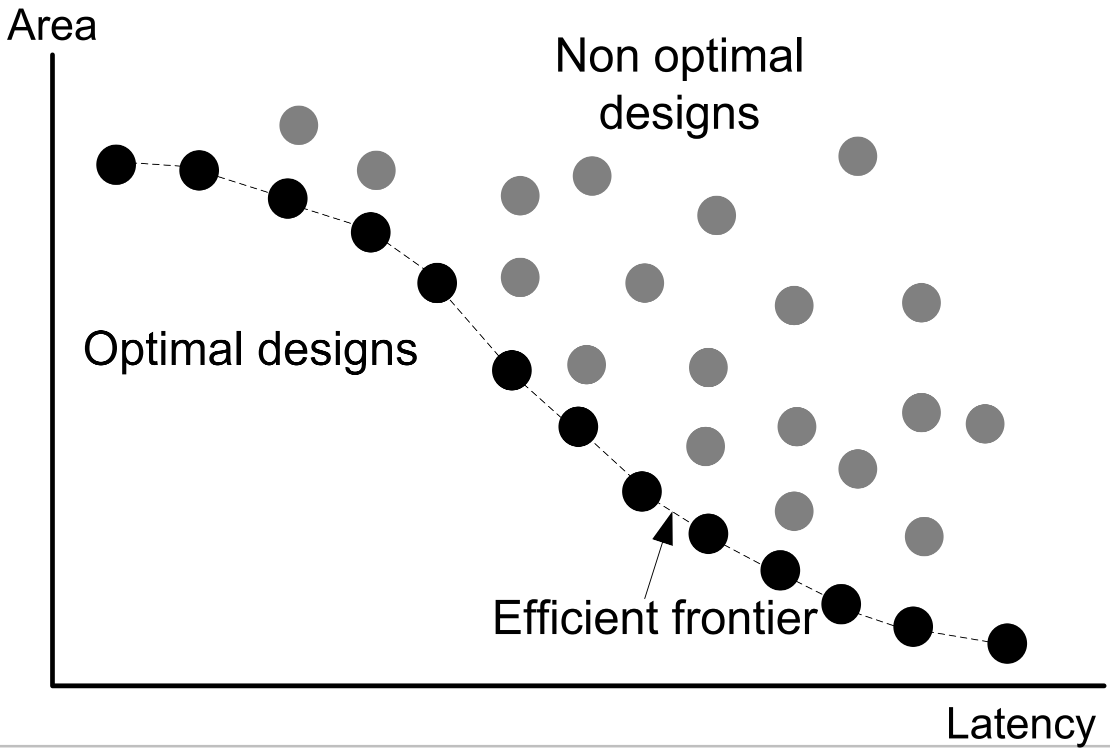


2. + : 1, * : 1



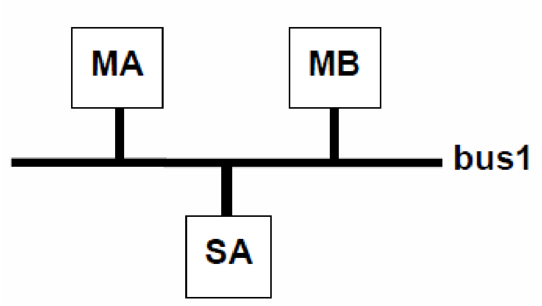
Micro-Architectural Design Space Exploration

- Pareto-optimal (dominating) designs
- 3 main exploration knobs:
  - Synthesis attributes (pragmas inserted in source code)
  - Global synthesis options
  - FU number

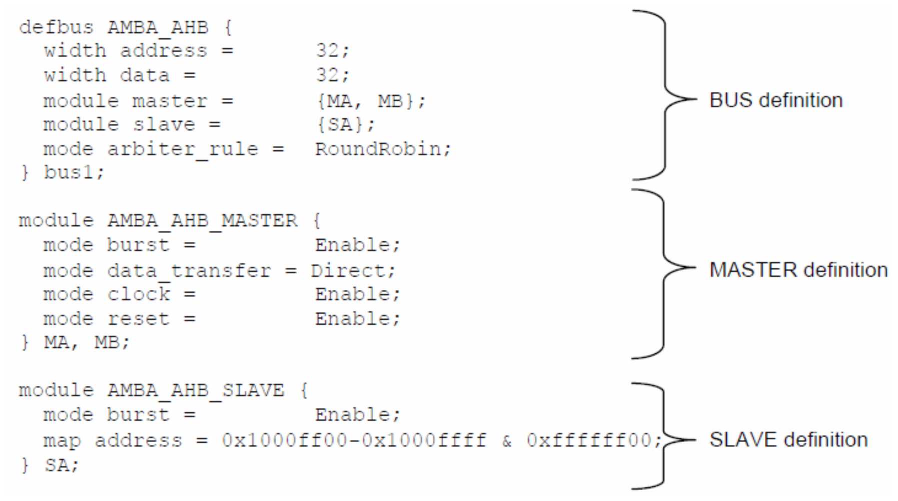


C-Based SoC

- Commercial Tools provide bus generators (AHB/AXI)
  - Inputs:
    1. Masters
    2. Slaves
    3. Arbiter type (e.g. fixed, round robin)
    4. Memory map
  - Outputs
    1. Synthesizable C code for bus and bus interface



- After HLS of the entire system → cycle-accurate model is generated



```
defbus AMBA_AHB {
  width address =       32;
  width data =          32;
  module master =       {MA, MB};
  module slave =        {SA};
  mode arbiter_rule =   RoundRobin;
} bus1;

module AMBA_AHB_MASTER {
  mode burst =          Enable;
  mode data_transfer = Direct;
  mode clock =          Enable;
  mode reset =          Enable;
} MA, MB;

module AMBA_AHB_SLAVE {
  mode burst =          Enable;
  map address = 0x1000ff00-0x1000ffff & 0xffffff00;
} SA;
```

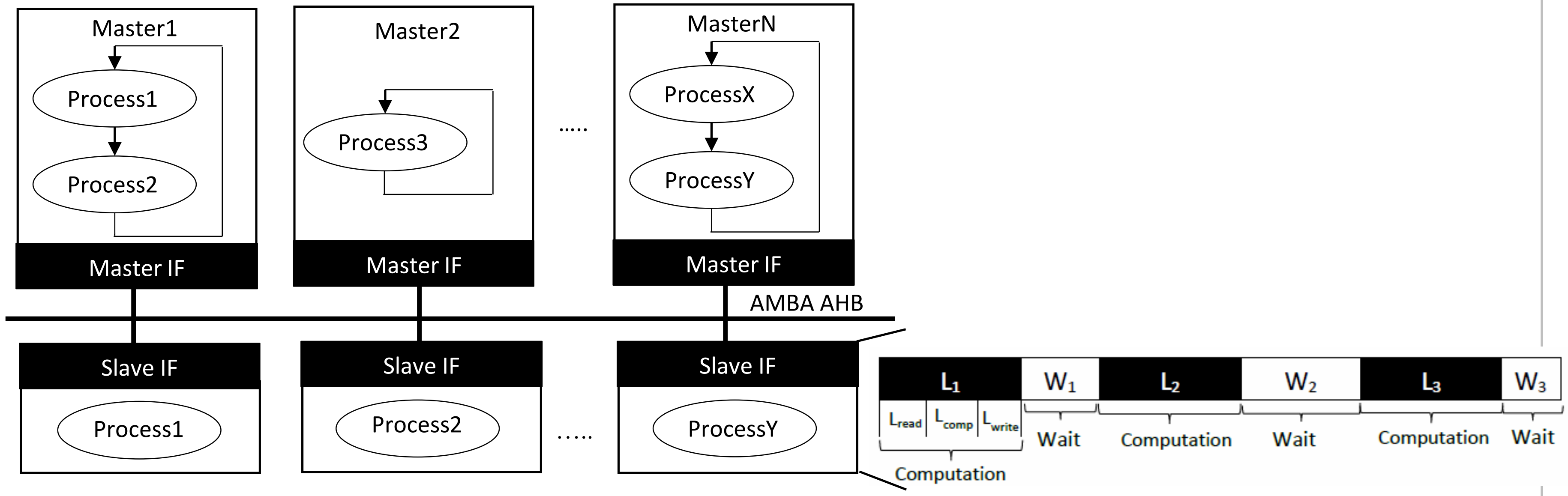
BUS definition
MASTER definition
SLAVE definition

Synthesizable C code

```
int i;
for (i = 0; i < DSIZE; i++) {
abc[i] = i;}
CBM_burst_write(0x1000ff00, abc, DSIZE);
```

Target Architecture

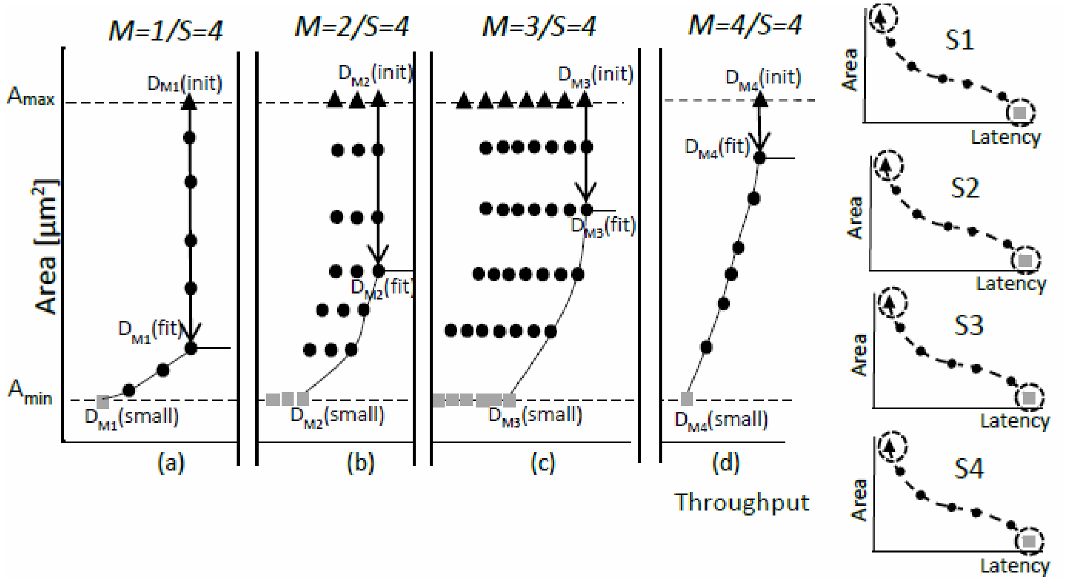
- Heterogeneous MPSoC
- Memory mapped shared bus
- BIPs instantiated loosely coupled HWAccs slaves
  - Each BIP Optimized for performance separately BUT:
    - Wait for master to start communication
    - Need to wait for arbiter to pass control of bus to return data



Motivational Example

- Observation 1 : Different Task mappings for the same system lead have the same area, but different performance.
- Observation 2: There is a design D_M(fit), with same performance, but smaller area than using fastest micro-architecture designs

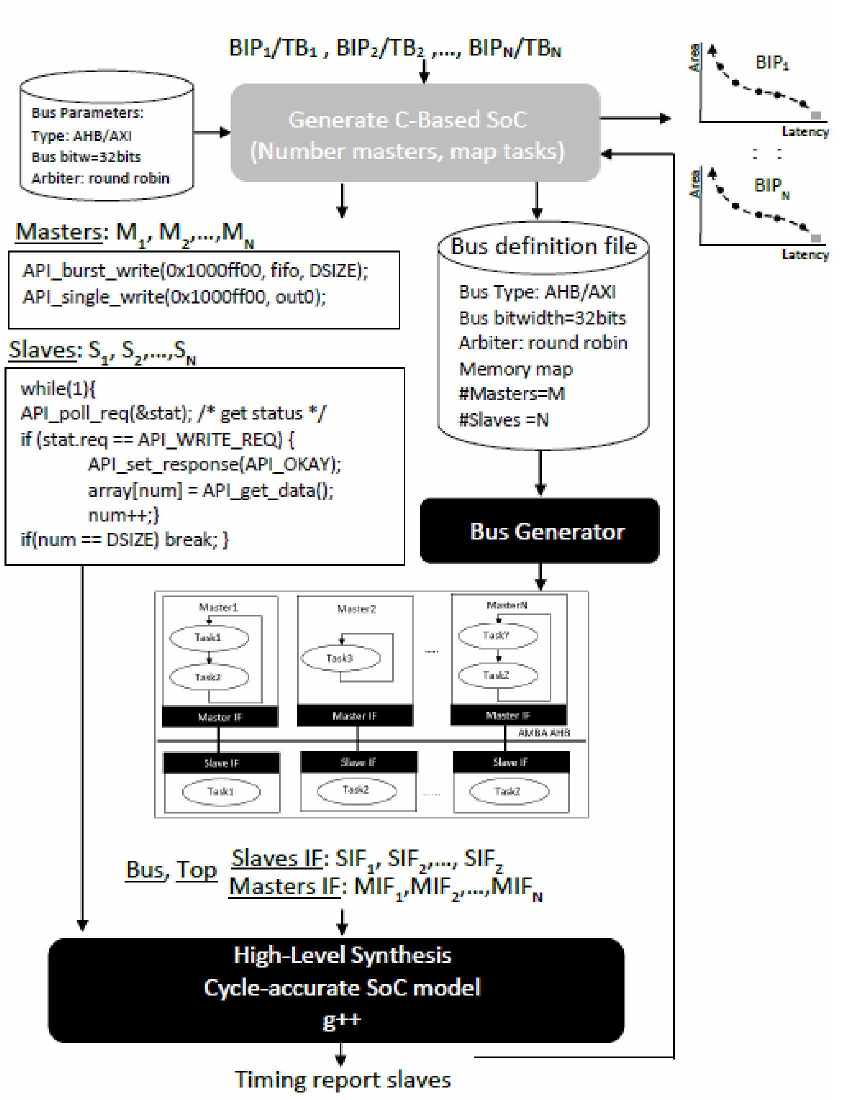
Masters (M) = 1-4 / Slaves (S) = 4



Objective: Find the smallest micro-architecture of each BIP mapped as a HWacc slave for the fastest SoC configuration (D_M(fit))

Proposed Optimization Flow

- **Pre-Step**: HLS DSE - for each BIP in the system.
- **Step 1**: SoC Generation. Generate systems with 1-N masters and different tasks' mappings using <u>fastest</u> BIP micro-architecture.
- **Step 2**: System Generation. Reads bus definition file and creates synthesizable SystemC files of entire system.
- **Step 3**: Cycle-accurate Simulation. HLS on each process, generates cycle-accurate model, compile (g++) and execute.
- **Step 4**: BIP Optimization. Read cycle-accurate timing report of each slaves' idle time and select smallest micro-architecture based on slack.



Bus Parameters:
Type: AHB/AXI
Bus bitw=32bits
Arbiter: round robin

```
API_burst_write(0x1000ff00, fifo, DSIZE);
API_single_write(0x1000ff00, out0);
```

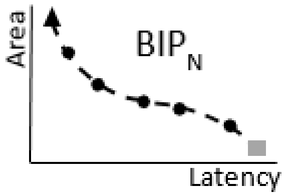
```
while(1){
API_poll_req(&stat); /* get status */
if (stat.req == API_WRITE_REQ) {
          API_set_response(API_OKAY);
          array[num] = API_get_data();
          num++;}
if(num == DSIZE) break; }
```

Bus Type: AHB/AXI
Bus bitwidth=32bits
Arbiter: round robin
Memory map
#Masters=M
#Slaves =N

Pre-Step : HLS DSE

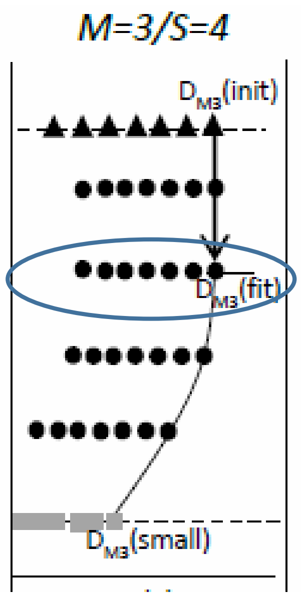
- 3 main *knobs*
  - Synthesis attributes (pragmas inserted in source code)
  - Global synthesis options
  - FU number $\rightarrow$ Used in this work

1. Synthesize each BIP without FU constraint file (FCNT) :
   - HLS tool allocates as many FUs to fully parallelize description
   - Generates FCNT indicating the type and number of FUs
2. Reduce FCNT file by X % until a single FU of teach time is reached

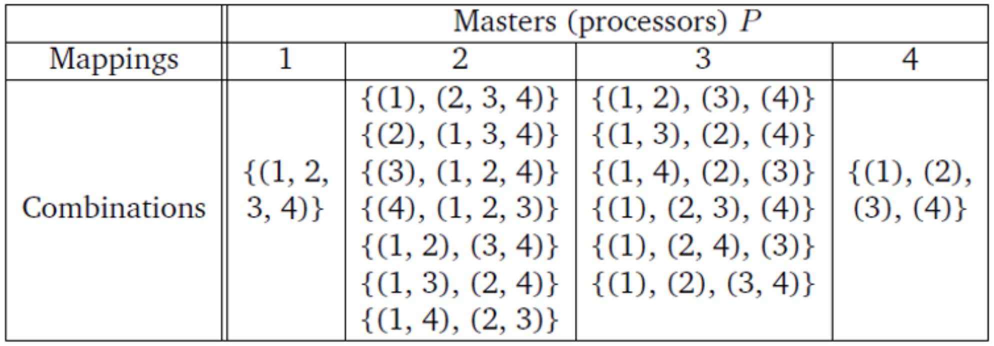


Step 1 : SoC Generation

- Given *S Slaves* → Generate SoCs for 1 to *S* masters
- For each configuration *m* all possible task mappings
  - Tasks periodically repeating
  - Execution order is not considered → Number of mappings follow Stirling number of second kind *S*(*s*,*m*), with s=Slaves and m=[1,M]masters
- E.g. M=3 (masters), S=4(slaves)



$$S(s,m)=\frac{1}{m!}\sum_{i=0}^{m}(-1)^{m-i}\binom{s}{k}i^{s}$$

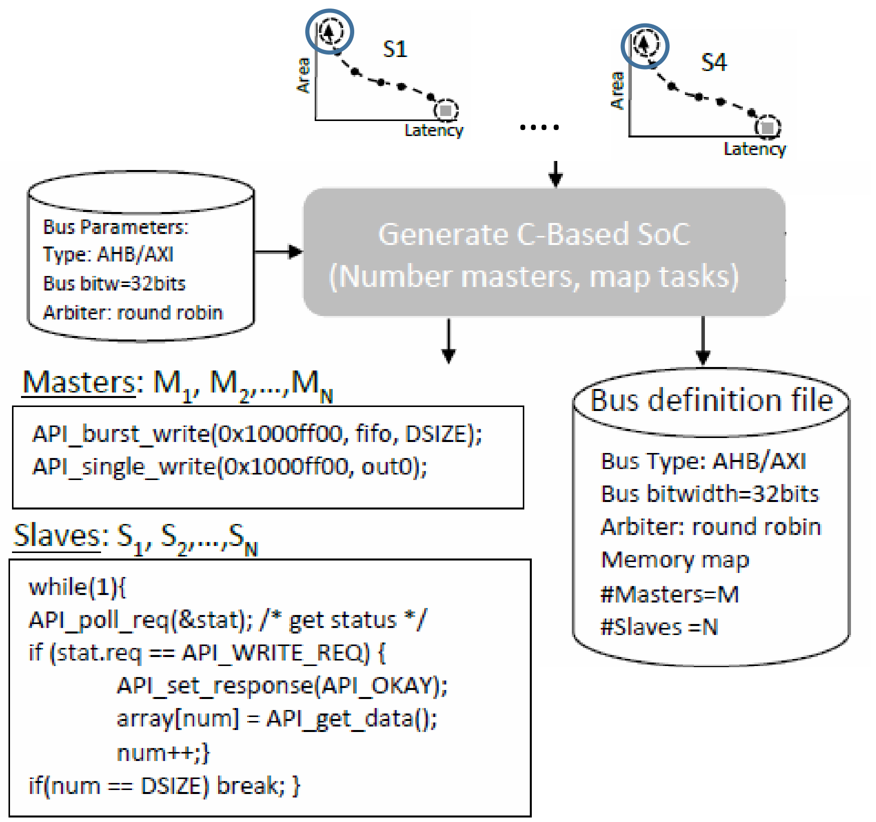


	Masters (processors) P			
Mappings	1	2	3	4

Combinations	{(1, 2, 3, 4)}	{(1), (2, 3, 4)}
		{(2), (1, 3, 4)}
		{(3), (1, 2, 4)}
		{(4), (1, 2, 3)}
		{(1, 2), (3, 4)}
		{(1, 3), (2, 4)}
		{(1, 4), (2, 3)}
			{(1, 2), (3), (4)}
			{(1, 3), (2), (4)}
			{(1, 4), (2), (3)}
			{(1), (2, 3), (4)}
			{(1), (2, 4), (3)}
			{(1), (2), (3, 4)}
				{(1), (2), (3), (4)}

Step 1 : SoC Generation con’t

- Inputs:
  - BIPs trade-off curves (fastest design used)
  - Bus parameters (AHB/AXI, bus bitwidth, arbiter)
- Outputs
  - Bus definition file for bus generator
  - Synthesizable C code for masters and slaves using synthesizable bus read/write APIs
  - Tasks mappings following Sterling number of second kind for each system with unique masters.



Masters: $M_1, M_2,\ldots,M_N$

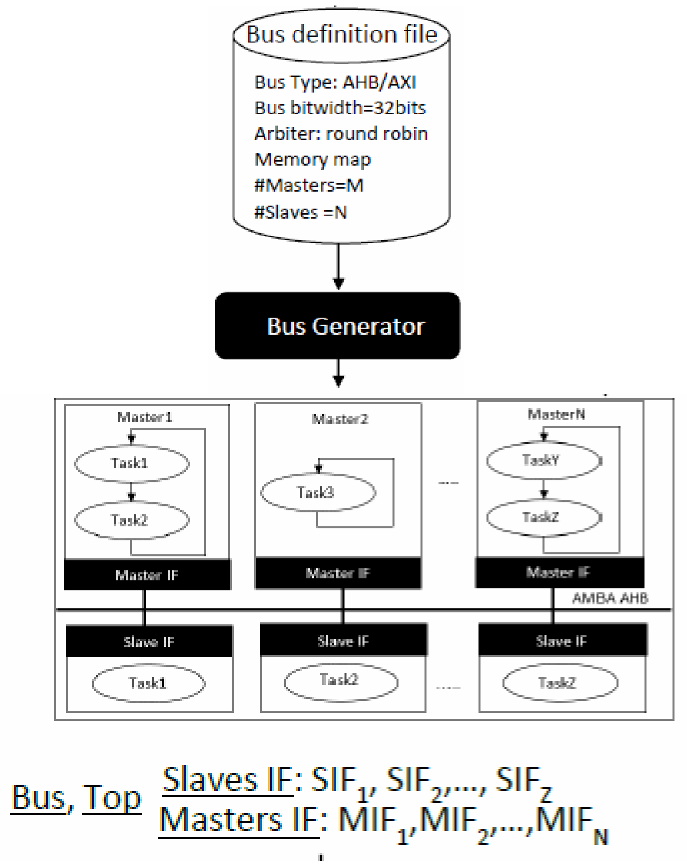
```
API_burst_write(0x1000ff00, fifo, DSIZE);
API_single_write(0x1000ff00, out0);
```

Slaves: $S_1, S_2,\ldots,S_N$

```
while(1){
API_poll_req(&stat); /* get status */
if (stat.req == API_WRITE_REQ) {
        API_set_response(API_OKAY);
        array[num] = API_get_data();
        num++;}
if(num == DSIZE) break; }
```

Step 2 : System Generation

- Commercial HLS tool bus generator called with :
  - bus definition file generated in step
  - Masters and slaves
- Generates synthesizable files for:
  - Top level module
  - Bus
  - Bus interfaces (masters and slaves)

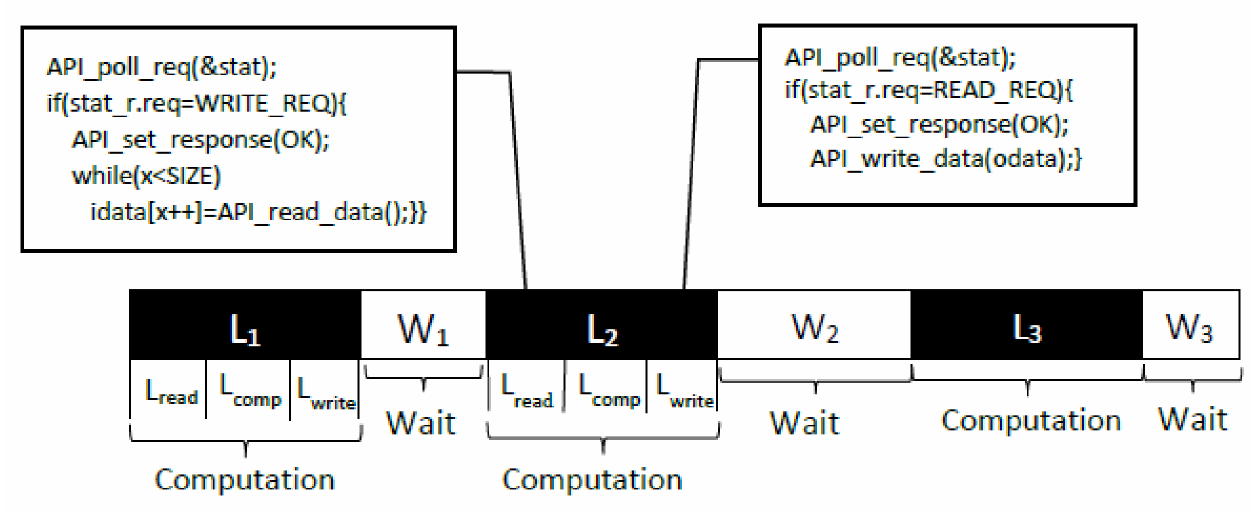


Step 3 :HLS and Cycle-accurate Simulation

1. HLS is a single process synthesis method → synthesize each synthesizable process
2. Call cycle-accurate model generator
   - Input: the scheduling result of each process
   - Output: cycle-accurate SystemC model of the entire system
3. Update slaves' cycle-accurate model to report time when reading, computing or writing data
4. Compile (g++) and execute SoC model
5. Read timing report of each BIP
   - Read, Write, computation and idle time of each BIP

Step 4 : Slave (BIP) Optimizations

- Computation latency $L_i=L_{read}+L_{comp}+L_{write}$
- Extract for each BIP smallest idle (waiting) time W_{min}
- New adjusted Latency $L_{adj}=$ floor($L_{comp}+W_{min}$)

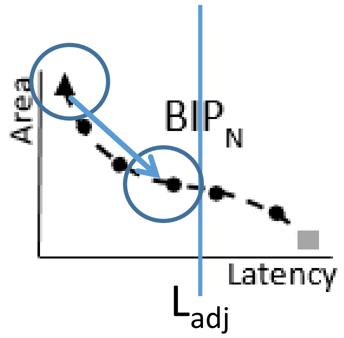


```
API_poll_req(&stat);
if(stat_r.req=WRITE_REQ){
  API_set_response(OK);
  while(x<SIZE)
    idata[x++]=API_read_data();}}
```

```
API_poll_req(&stat);
if(stat_r.req=READ_REQ){
  API_set_response(OK);
  API_write_data(odata);}
```

Step 4 : Slave (BIP) Optimizations con’t

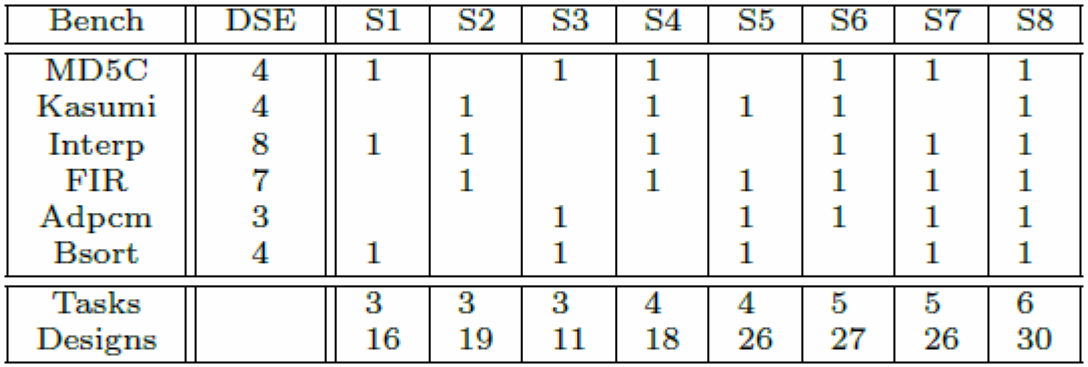
- Choose micro-architecture with closes smallest latency to new latency



- Re-synthesize and re-simulate the new system with each new micro-architecture

Experimental Setup

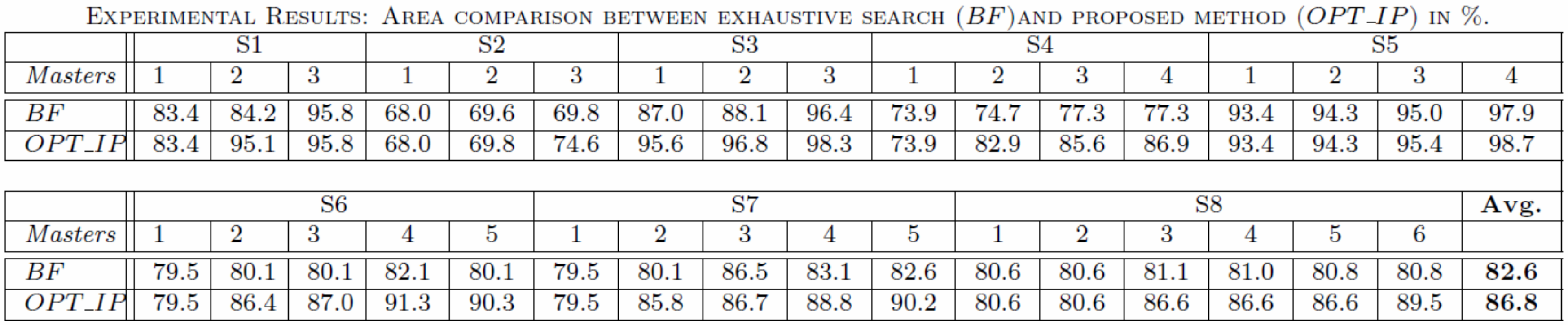
- Complex systems based on computationally intensive tasks were formed by grouping individual benchmarks as HWacc
- S2CBench benchmark suite (www.s2cbench.org)
- Experiments run on Intel dual 2.4 GHz Xeon with 16GBytes of RAM running Linux Fedora release 19
- HLS tool NEC's CyberWorkBench v. 5.5
- Target technology Nangate's 45nm's Opencell
- Target synthesis frequency 100MHz



Bench	DSE	S1	S2	S3	S4	S5	S6	S7	S8
MD5C	4	1		1	1		1	1	1
Kasumi	4		1		1	1	1		1
Interp	8	1	1		1		1	1	1
FIR	7		1		1	1	1	1	1
Adpcm	3			1		1	1	1	1
Bsort	4	1		1		1		1	1
Tasks		3	3	3	4	4	5	5	6
Designs		16	19	11	18	26	27	26	30

Experimental Results : Area

- OPT_IP vs. exhaustive search (BF)
- BF tries all possible micro-architectures of HLS DSE result
- Find the micro-architecture of each BIP for the fastest system
- In all cases same throughput within 1% can be achieved
- On average the area is reduced:
  - BF =17.43%
  - OPT_IP = 13.21% (~5% larger than BF)



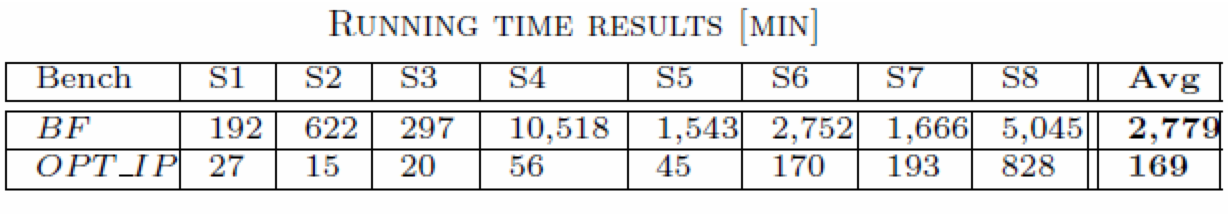
Experimental Results: Area comparison between exhaustive search (*BF*)and proposed method (*OPT_IP*) in %.

	S1			S2			S3			S4				S5			
Masters	1	2	3	1	2	3	1	2	3	1	2	3	4	1	2	3	4
BF	83.4	84.2	95.8	68.0	69.6	69.8	87.0	88.1	96.4	73.9	74.7	77.3	77.3	93.4	94.3	95.0	97.9
OPT_IP	83.4	95.1	95.8	68.0	69.8	74.6	95.6	96.8	98.3	73.9	82.9	85.6	86.9	93.4	94.3	95.4	98.7

	S6					S7					S8						**Avg.**
Masters	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	6	
BF	79.5	80.1	80.1	82.1	80.1	79.5	80.1	86.5	83.1	82.6	80.6	80.6	81.1	81.0	80.8	80.8	**82.6**
OPT_IP	79.5	86.4	87.0	91.3	90.3	79.5	85.8	86.7	88.8	90.2	80.6	80.6	86.6	86.6	86.6	89.5	**86.8**

Experimental Results : Running Time

- OPT_IP is on average ~16x faster than BF



RUNNING TIME RESULTS [MIN]

Bench	S1	S2	S3	S4	S5	S6	S7	S8	**Avg**
BF	192	622	297	10,518	1,543	2,752	1,666	5,045	**2,779**
OPT_IP	27	15	20	56	45	170	193	828	**169**

Summary and Conclusions

- Presented a method to optimize the micro-architecture of BIPs mapped onto heterogeneous MPSoCs as loosely coupled HWAcc.
- Two main advantages of C-Based VLSI design leveraged in this work:
    1. HLS DSE to achieve micro-architectures of different characteristics.
    2. State of the art HLS tools allow the generate and simulation (cycle-accurate) of entire SoCs.
- Results show that our proposed method leads to good results while being much faster than an exhaustive search.

www.eie.polyu.edu.hk/~schaferb/darclab

Design Automation and Reconfigurable Computing LAB

dARC