PICO EXPRESS TUTORIAL

DIGITAL SYSTEM DESIGN METHODOLOGY LAB CLASS

Codesign and Parallel Processing Lab Sungjin Yoon

Contents

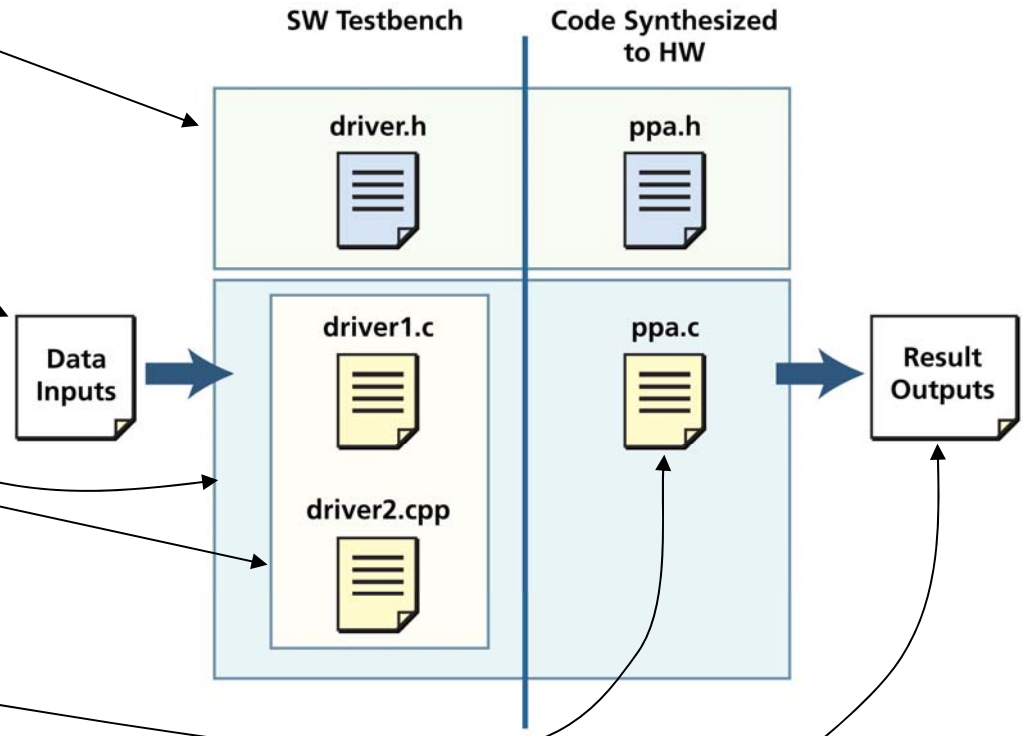
Contents

Introduction

- C based design
  - Time-to-market pressure
    - Development (x5)
      – Initial RTL design is very hard and time consuming job.
    - Design Space Exploration (x20)
      - Modifying the system is very much easier in C than in RTL.
  - System Level Design
    - Abstract level is becoming higher and higher
    - transistor level → gate level → register transfer level →?

High Level Code Structure

- Header files
- Data files
- Source files
  - Driver files
    - arbitrary C/C++
  - PPA file
    - synthesizable C
- Result files



Real Code Example

Example - /home/sjyoon/example1/

Driver Code

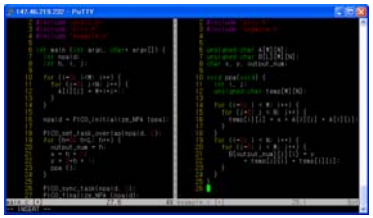
```
147.46.219.232 - PuTTY
 2 #include <stdio.h>
 3 #include "pico.h"
 4 #include "example.h"
 5
 6 int main (int argc, char* argv[]) {
 7   int npaid;
 8   int h, i, j;
 9
10   for (i=0; i<M; i++) {
11     for (j=0; j<N; j++) {
12       A[i][j] = M*i+j+1;
13     }
14   }
15
16   npaid = PICO_initialize_NPA (ppa);
17
18   PICO_set_task_overlap(npaid, 2);
19   for (h=0; h<L; h++) {
20     output_num = h;
21     x = h + 1;
22     y = 2*h + 1;
23     ppa ();
24   }
25
26   PICO_sync_task(npaid, 0);
27   PICO_finalize_NPA (npaid);
main.c [+]                    27,6          4%
-- INSERT --
```

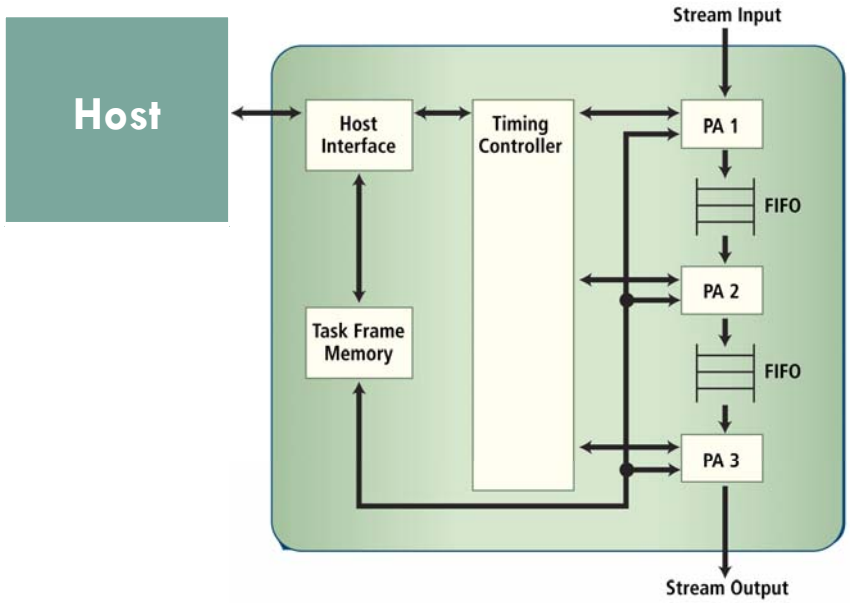
PPA Code

```
 2 #include "pico.h"
 3 #include "example.h"
 4
 5
 6 unsigned char A[M][N];
 7 unsigned char B[L][M][N];
 8 char x, y, output_num;
 9
10 void ppa(void) {
11   int i, j;
12   unsigned char temp[M][N];
13
14   for (i=0; i < M; i++) {
15     for (j=0; j < N; j++) {
16       temp[i][j] = x * A[i][j] + A[i][j];
17     }
18   }
19   for (i=0; i < N; i++) {
20     for (j=0; j < M; j++) {
21       B[output_num][j][i] = y
22           * temp[j][i] * temp[i][j];
23     }
24   }
25 }
26
~
example.c [+]                 26,1          Bot
```

Real Code Example

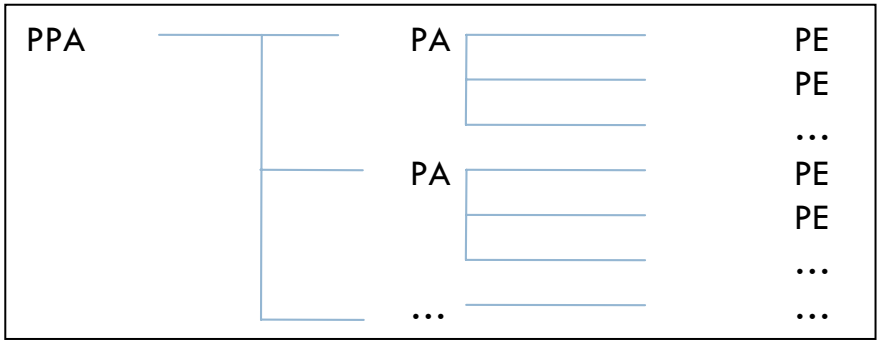
Driver Code| PPA Code

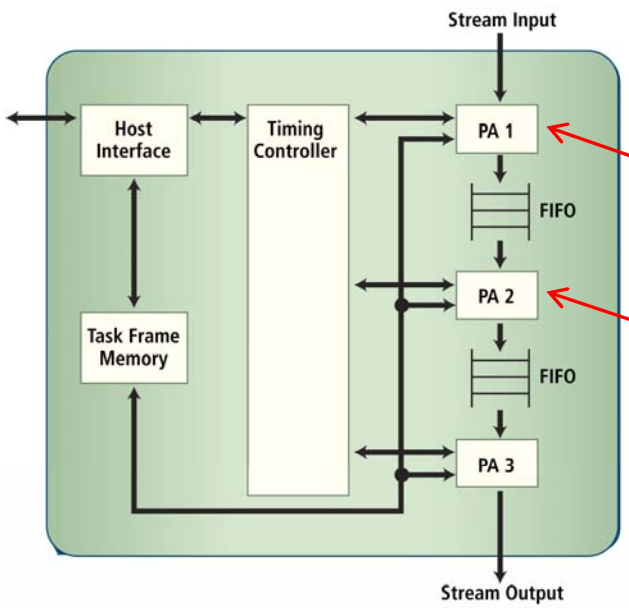




Pipeline of Processing Arrays

Real Code Example



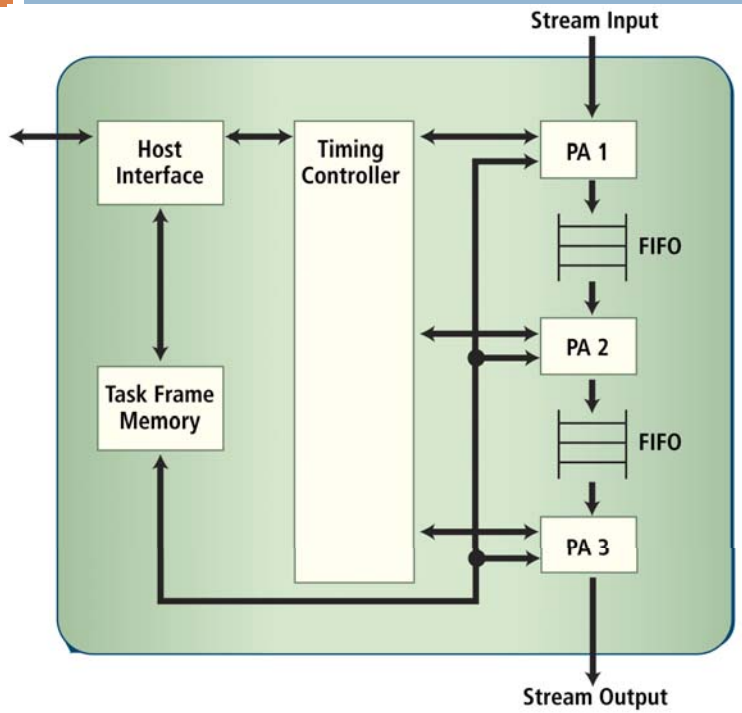


Pipeline of Processing Arrays

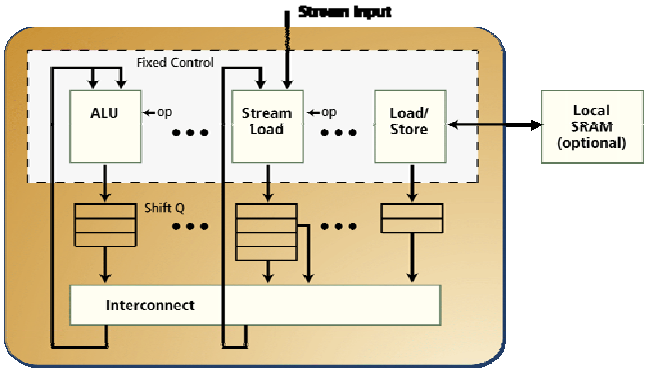
PPA Code

```
 2 #include "pico.h"
 3 #include "example.h"
 4
 5
 6 unsigned char A[M][N];
 7 unsigned char B[L][M][N];
 8 char x, y, output_num;
 9
10 void ppa(void) {
11   int i, j;
12   unsigned char temp[M][N];
13
14   for (i=0; i < M; i++) {
15     for (j=0; j < N; j++) {
16       temp[i][j] = x * A[i][j] * A[i][j];
17     }
18   }
19   for (i=0; i < N; i++) {
20     for (j=0; j < M; j++) {
21       B[output_num][j][i] = y
22           * temp[j][i] * temp[i][j];
23     }
24   }
25 }
26
ple.c [+]                    26,1          Bot
```

Architecture Template for PICO Express Hardware



Pipeline of Processing Arrays



A Processing Element

- Architecture exploits parallelism at all levels at minimal cost
- Each PA is highly optimized – can be as small as 200 gates
- PAs communicate using streams, shared RAMs and shared scalars
- Each PA is in own stall domain
  - Allows for FIFO flow control
  - Allows for highly parallel design

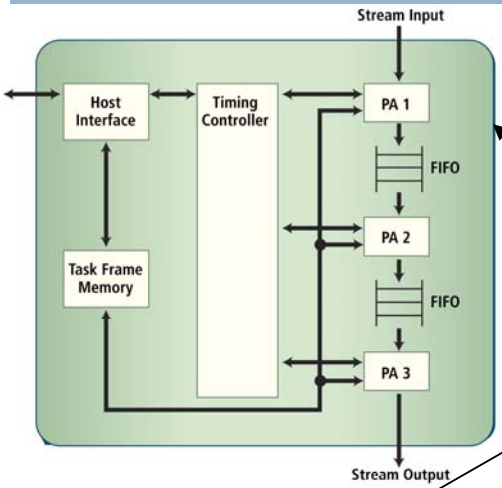
PICO Express Designs Data Path as well as Control Logic

- Host interface and memory mapping
- Task frame memory to hold multiple windows of configuration parameters for overlapped task execution
- Controlling the execution order of Pas
- Multi-buffered memories from single arrays in source
- Memory arbitration
- Flow control using streams
- You can gain maximum benefits by understanding PICO's ability to design control logic

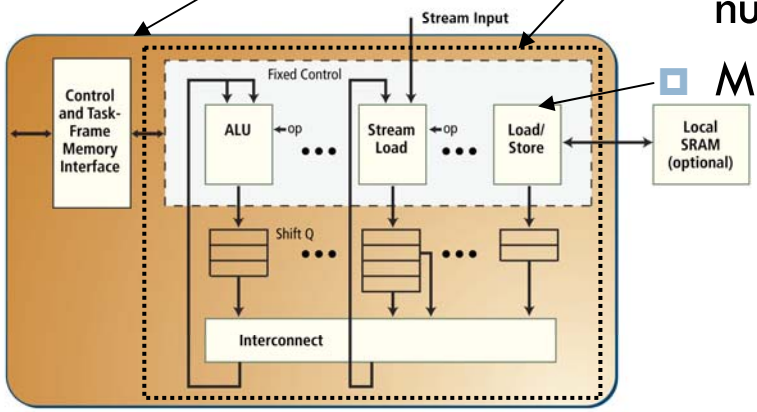
RTL and Its Location

- RTL consists of
  - Compiler generated structural RTL for the design
  - Manually created behavioral RTL for “macrocells”
    - Macrocells are building blocks like adder, register, mux, etc.
  - Like the architectural template, RTL is hierarchically organized
- RTL is in the rtl_package directory
  - Implementation Directory window: <implementation-name>/rtl_package
- rtl_package directory contains
  - Synthesizable RTL
    - rtl/ - contains the compiler generated RTL for the design
    - macrocells/ - contains RTL for macrocells
    - synth/ - contains scripts for RTL synthesis tools
  - Simulation infrastructure and testbenches
    - scripts/ - contains scripts for RTL simulation and analysis tools
    - simu_stubs/ - contains testbenches and external stubs

PICO Express RTL Overview



Pipeline of Processing Arrays (PPA)



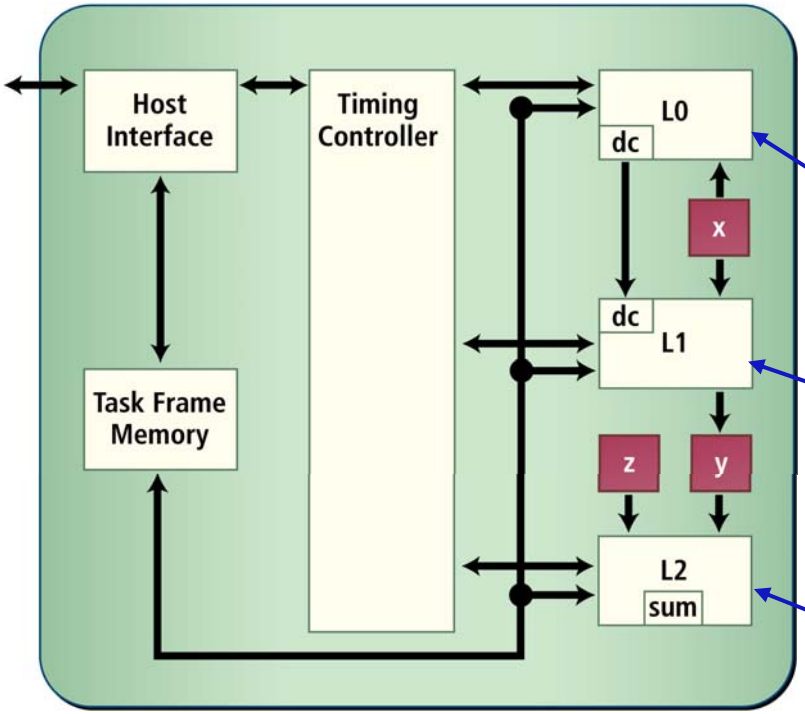
A Processing Array (PA)

- Like the architectural template, RTL is hierarchically organized
  - <file_name>_ppa.v: Top level RTL for a PPA
  - <file_name>_paw_<n>.v: Wrapper for PA number n
  - <file_name>_pa_<n>.v, and
  - <file_name>_pe_<n>.v : RTL for PA number n
  - Macrocells (addw.v, etc.)

Mapping of Code to a PPA: the C Code

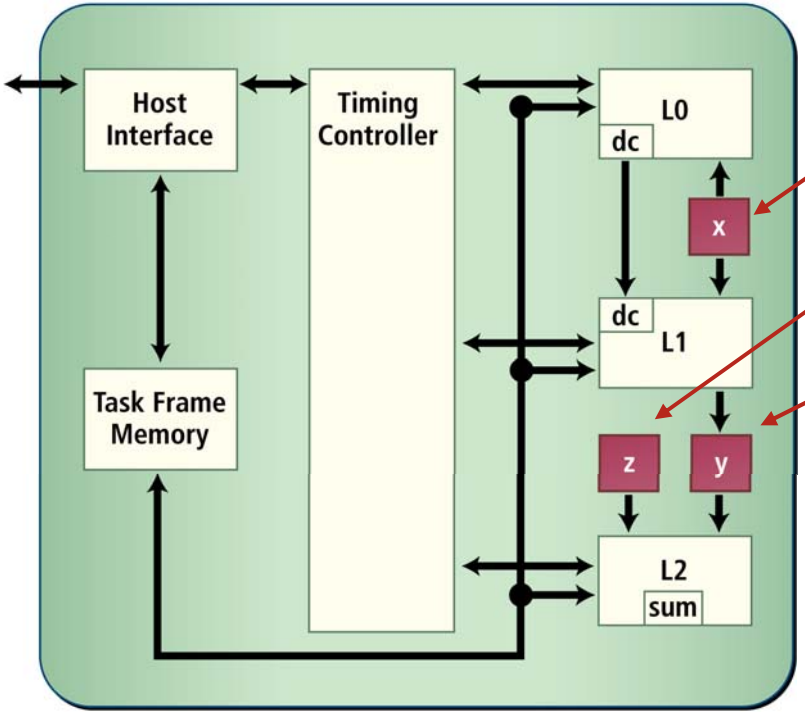
```
char x[64], z[64];
int sum;

void ppa(void) {
  int i,dc=0;
  char y[64];
  for (i=0; i<64; i++) dc +=
x[i];
  dc = dc >> 6;

  for(i=0; i<64; i++) y[i] =
x[i]-dc;

  sum = 0;
  for (i=0; i<64; i++) sum +=
y[i]*z[i];
}
```

High Level Mapping of Code to a PPA




```
char x[64], z[64];
int sum;

void ppa(void) {
  int i,dc=0;
  char y[64];
  for (i=0; i<64; i++) dc +=
x[i];
  dc = dc >> 6;

  for(i=0; i<64; i++) y[i] =
x[i]-dc;

  sum = 0;
  for (i=0; i<64; i++) sum +=
y[i]*z[i];
}
```

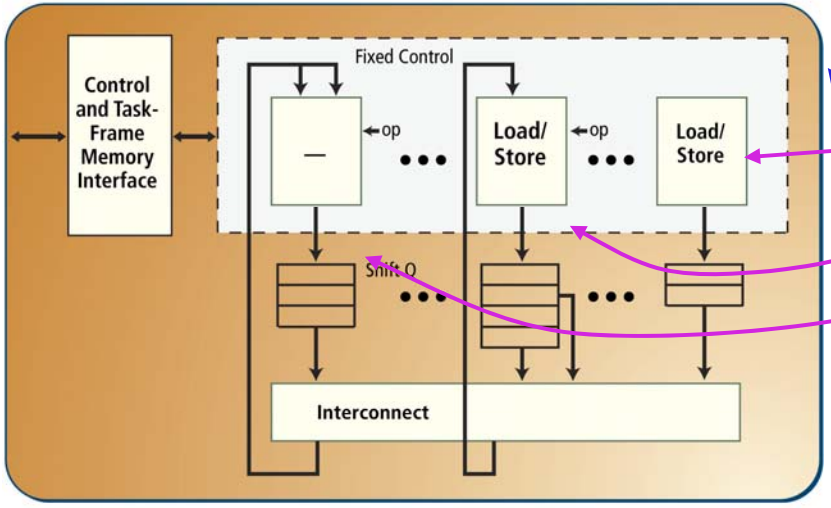
High Level Mapping of Data




```
char x[64], z[64];
int sum;

void ppa(void) {
  int i,dc=0;
  char y[64];
  for (i=0; i<64; i++) dc +=
x[i];
  dc = dc >> 6;

  for(i=0; i<64; i++) y[i] =
x[i]-dc;

  sum = 0;
  for (i=0; i<64; i++) sum +=
y[i]*z[i];
}
```

Mapping of C Operations To a PE

```
char x[64], z[64];
int sum;

void ppa(void) {
  int i,dc=0;
  char y[64];
  for (i=0; i<64; i++) dc +=
x[i];
  dc = dc >> 6;

  for(i=0; i<64; i++) y[i] =
x[i]-dc;

  sum = 0;
  for (i=0; i<64; i++) sum +=
y[i]*z[i];
}
```

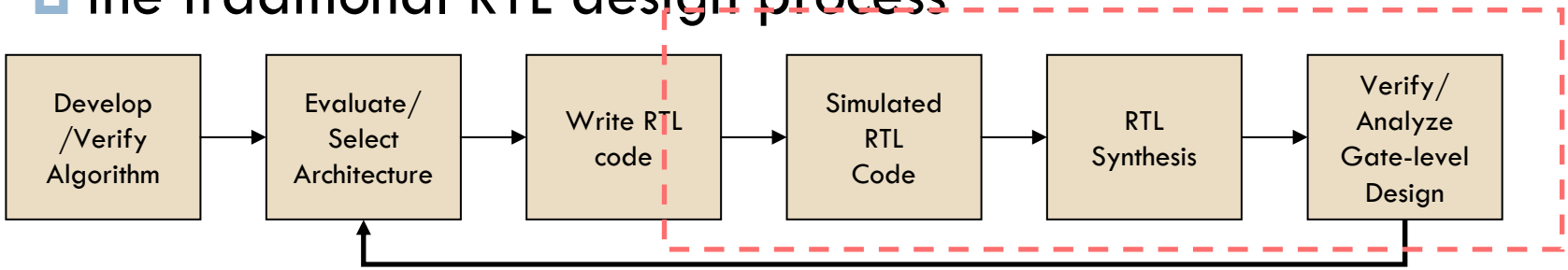


A Processing Array (PA)

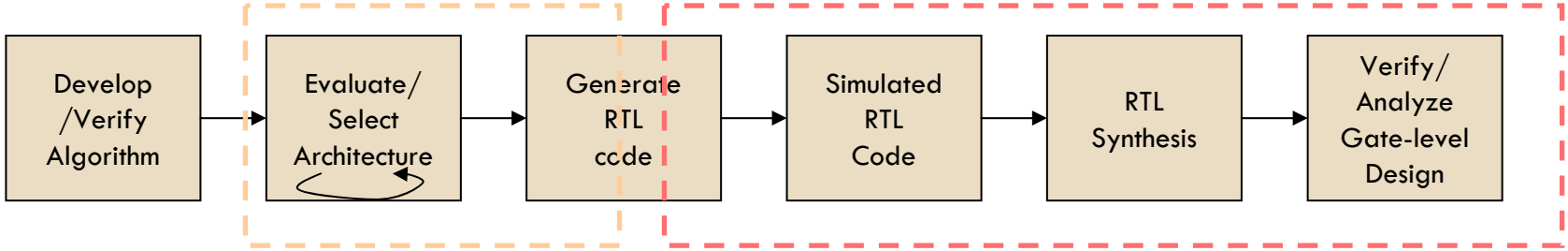
Contents

Design and Verification Flow

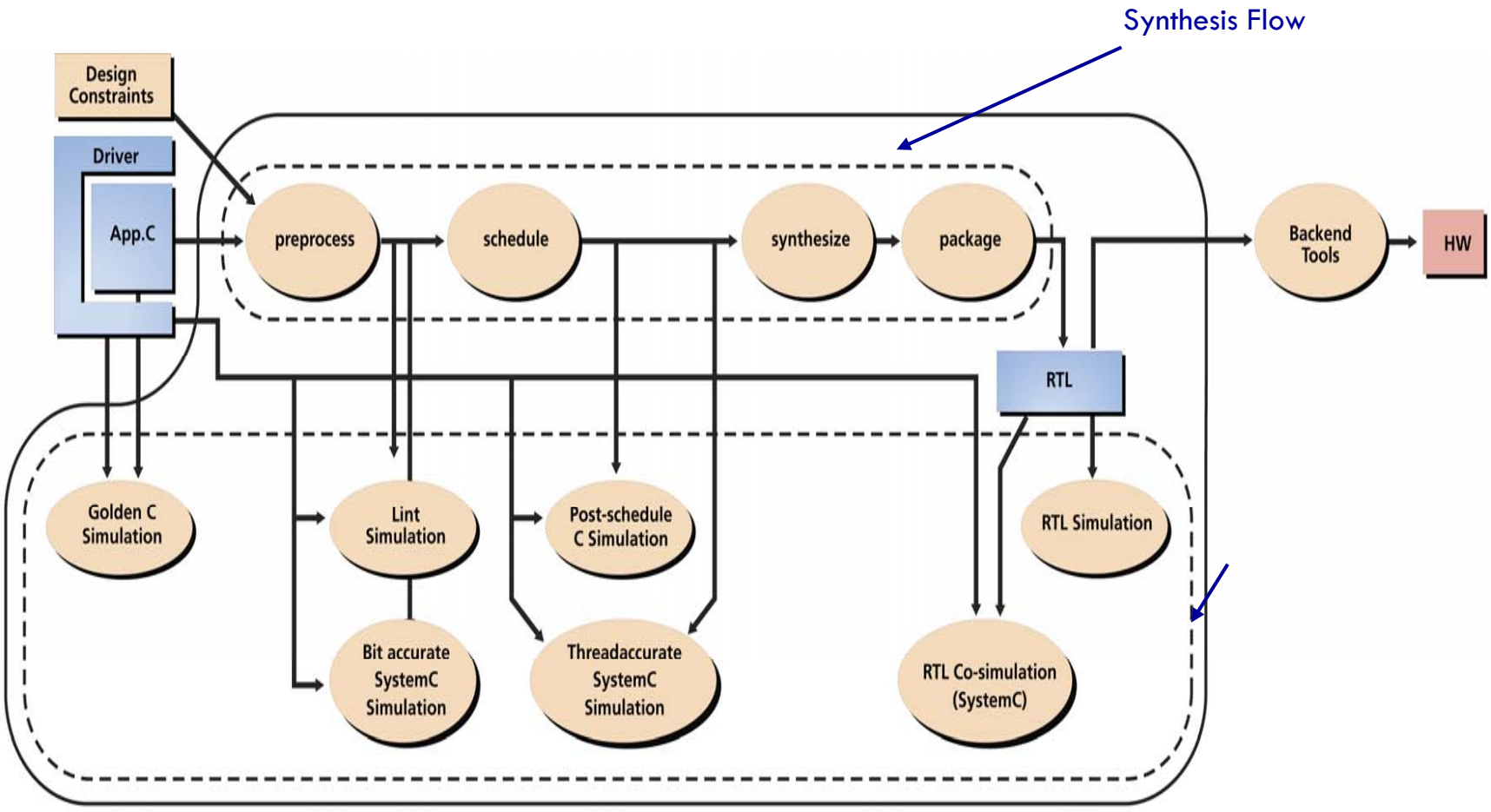
- High Level Synthesis
  - the traditional RTL design process



  - the design process utilizing behavioral synthesis



Design and Verification Flow



Synthesis Flow Components

- Preprocess
  - High level program transformations like function inlining
  - Performs static checks for accepted C Syntax
  - Code generation for Lint simulation and Bit-accurate SystemC simulation
- Schedule
  - High level program and loop optimization and loop scheduling
  - Code generation for Post-schedule C simulation and Thread-accurate SystemC simulation
- Synthesize
  - Instruction level optimization and scheduling, resource allocation and RTL creation
  - Generation of RTL simulation and RTL co-simulation testbenches
- Package
  - Collect relevant information in an easily accessible place to work with other tools like Synopsys DC

Verification and Performance Modeling Flow (1)

- Golden C simulation
  - Runs original c code using gcc
  - Compares the results against reference output
  - Or produces golden output for comparison during later phases
- Lint simulation
  - Performs dynamic checks to catch errors early on
    - Un-initialized variable
    - Bitsize overflow and underflow
    - Out-of-bound array accesses
- Post-schedule C simulation
  - Produces test vectors for RTL simulation
- RTL Simulation
  - Verifies that the RTL is correct using test vectors produced by the above phase
  - Allows user-controlled or random perturbation of operating conditions

Verification and Performance Modeling Flow (2)

- Bit-accurate SystemC simulation
  - Transaction-level simulation for functional verification
- Thread-accurate SystemC simulation
  - Transaction-level simulation for performance estimation
    - Models parallel behavior of hardware
  - Runs significantly faster than RTL
- RTL Co-simulation
  - Transaction-level simulation at the hardware level
  - Verifies RTL and interaction with the host processor

Representative Key Flows

- Lint to detect errors early on
  - Golden Simulation → Preprocess → Lint Simulation
- Performance Estimation
  - Golden Simulation → Preprocess → Lint Simulation → Schedule → Thread accurate SystemC simulation
- RTL simulation
  - Golden Simulation → Preprocess → Lint Simulation → Schedule → Post-schedule C simulation → Synthesize → Package → RTL simulation
- RTL co-simulation
  - Golden Simulation → Preprocess → Lint Simulation → Schedule → Synthesize → Package → RTL co-simulation

GUI Overview

PICO Express (/home/sjyoon/training2007/module1/quadratic)

File View Project Implementation Group Run Analyze Window Help

Active Implementation: a0

Configure New Implementation

Run

Default

- Golden C Simulation
- PreProcess
- Lint Simulation
- Bit Accurate SystemC Simulation
- Schedule
- Post-Schedule C Simulation
- Thread Accurate SystemC Simulation
- Synthesize
- Export RTL Package
- RTL Simulation
- Co-Simulation

Project View

- quadratic
  - Groups
  - Implementations
    - a0
  - sources
  - headers
  - data
  - results

Implementation D

- a0
  - data
  - info
  - Logs
  - Reports
  - simulation
  - src

Synfora

PICO Express™

http://www.synfora.com

© 2004-2007 Synfora Inc. All Rights Reserved.

```
pico:[quadratic:-]>
set_project_params -projdir /home/sjyoon/training2007/module1/quadratic
Project directory now set to: /home/sjyoon/training2007/module1/quadratic
pico:[quadratic:-]> select_experiment a0
<no exp> select_experiment a0
pico:[quadratic:a0]>
```

	Input MITI (Cycles)	Input Clock Frequency (MHz)	Estimated Cost (Gates)	Delivered MITI (Cycles)	Delivered Task Latency (Cycles)	Delivered Throughput (Tasks per second)
a0	0	100	N/A	N/A	N/A	N/A

Set Active Implementation completed successfully

Exercise 1-1

- Start PICO Express in GUI mode (if not already running)
  - Shell> peg
  - (alias of pico_express –g –GCC /opt/gcc-3.2.3/bin)
- Open quadratic project
  - File OpenProject... ~/exercise1
- Open the file app.c to understand the source code
- Select implementation a0
- Run Default flow on implementation a0. The Default flow doesn't run any RTL simulation
- Look at the results in the Watch Window. If the Watch Window is not visible, select it from the View menu
- Questions
  - What is the clock frequency for this design?
  - What is the PICO estimated area for this design?

Implementation Directory View in GUI

- It allows access to relevant information about an implementation
  - data: Data input files for the application
  - info: Intermediate files giving more information about program transformations
  - Logs: Logfile for each command
  - Reports: Reports providing feedback about the design
  - rtl_package: Collects the whole design including RTL, testbenches and scripts in one directory
  - src: Source and header C files for this implementation
  - simulation: Data used or produced during various simulation step
  - Work: Temporary workspace

Analysis Tools in GUI

- PPA Graph
  - Shows the computation blocks (loop nests) and the data structures used for communication
  - PPA graph is the anchor point for looking at design characteristics

- Resource Browser
  - Shows the hardware resources used in the RTL
    - Function units, registers, memories, streams
  - Shows C source mapping and aggregate cost reports

Exercise 1-2

- Navigate through the implementation view
  - Show all accessible files
- Show the PPA graph
- Show resource browser and aggregate cost report
- Show summary results report

Contents

- PICO Express
  - Introduction
  - Design Flow
  - **Memories and Arrays**
  - Performance Specification
  - Coding Restrictions
  - Coding Issues
  - Exercise

Memories and Arrays

- Arrays may map to:
  - **External memories** – (compiled by a memory compiler outside of the PICO-generated RTL)
    - We call these **user_supplied**
  - **Synthesized memories** – (flip-flops for RAM, logic for ROM)
    - We call these **internal_fast**
  - **Registers/Wires** – In certain cases, the compiler can eliminate arrays entirely and map to appropriate registers or wires
    - This transformation is called **Scalarization**
- To control external (user_supplied) vs. synthesized (internal_fast) memory:
  - Use `#pragma user_supplied <array>`
  - Use `#pragma internal_fast <array>`
  - Defaults are based on array size
    - For RAM, if an array has <= 16 entries OR <= 256 bits it will default to `internal_fast`

Contents

Performance Specification -- Basics

- Performance constraints drive the parallelism and the number of hardware resources in an implementation

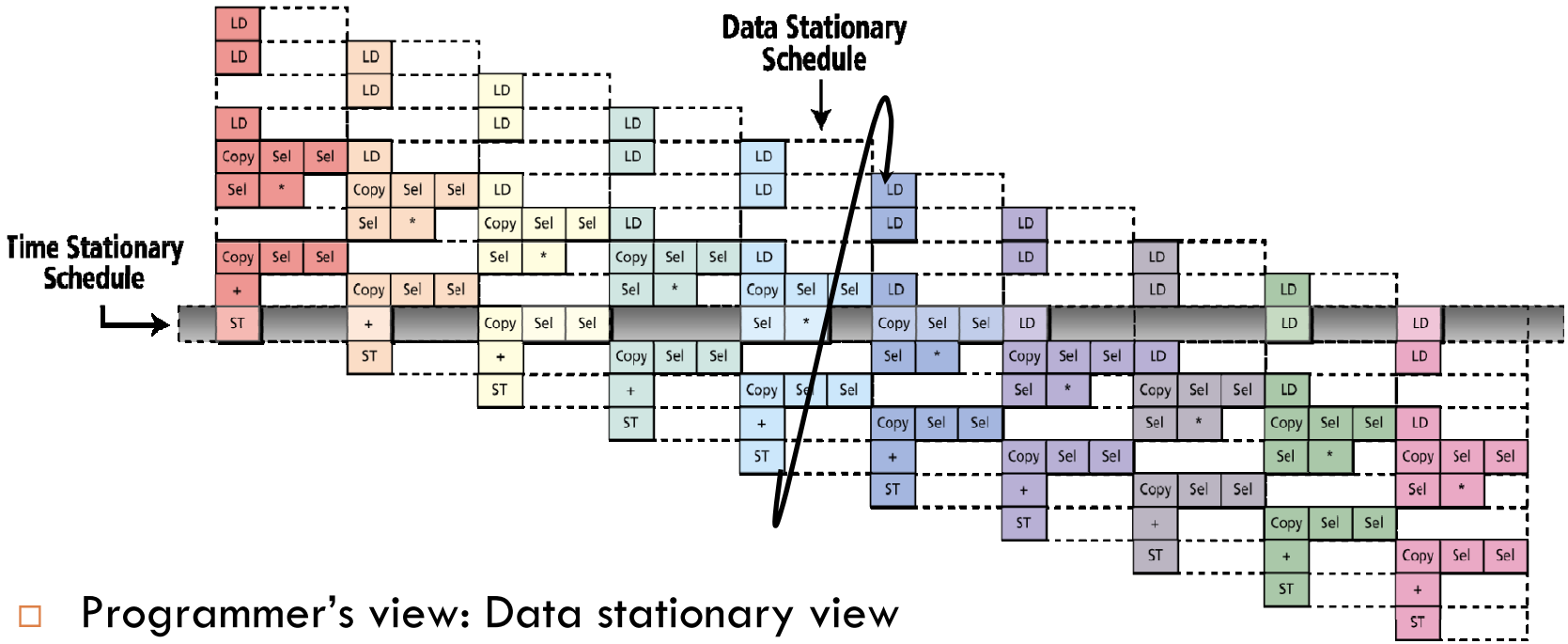
- To explain performance constraints, we need to explain the following two concepts
  - What is a “task”
  - The compiler-directed parallel/overlapped execution of C programs used in PICO Express

What is a Task?

```
void ppa();//PPA procedure

int main (int argc, char
**argv) {
  int ppaid,i;

  ppaid =
PICO_initialize_NPA(ppa);
  for(i=0;i<N;i++)
    ppa();
  }

PICO_finalize_NPA(ppaid);
}
```

- **Task** corresponds to a single execution of the top level PPA procedure that gets converted to hardware
- One task is one runtime call of t he procedure ppa
- In the code to the left, there will be N tasks, one for each call to procedure ppa

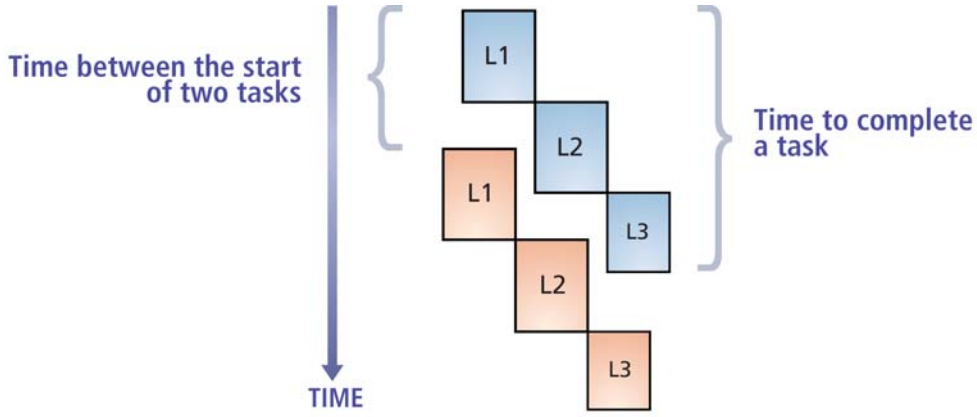
C Sequential Model vs. RTL Parallel Model



- Programmer's view: Data stationary view
  - What happens on a loop iteration
- Hardware designer's view: Time stationary view
  - What happens on a clock cycle

How to Measure Performance at the Task Level?

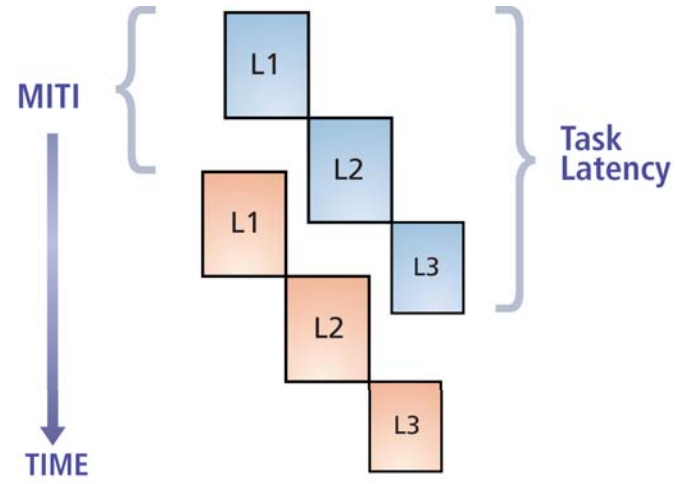
- Like any pipelined execution, there are two measures associated with the overlapped task execution
  - How fast tasks can be started? That is, what is the time between the start of a task and the start of the next task?
    - This determines the throughput of the system
  - How long does it take to complete a task? That is what is the time between the start and end of a task?
    - This determines the latency of a task



Blue and Red represent two different tasks

PICO Terminology: MITI and Task Latency

- Minimum Inter-task Interval (MITI)
  - Minimum time between the start of two tasks
  - That is, how fast tasks can be started
- Task Latency
  - Time to complete a task
- For execution with task overlap
  - MITI < Task Latency
- For execution with no task overlap
  - MITI ≥ Task Latency



Blue and Red represent two different tasks

Performance Specification in PICO Express

- Performance is specified using two parameters
  - Clock Frequency in MHz
  - MITI in cycles

- They are specified by opening the panel:
  - Implementation→ Configure [New… | Current...] → General

Specified Performance vs. Delivered Performance

- PICO Express uses MITI at compile time to design hardware meeting the performance
- Delivered performance may be different
- PICO Express provides simulation based performance charts to analyze delivered performance

Simulation Based Performance Chart

Difference is MITI

Difference is Task Latency

Contents

Coding Restrictions

- Use arrays instead of pointers
- Structured code – no goto
- No floating point
- No structs, unions, switch/case
- No static, volatile declarations
- No outer-loop around sequence of loops

Contents

Coding Issues

- Loop Transformation
- Array Partitioning
- Affine Array Indexing
- Logical vs Bitwise Operations
- Reducing Multiplier
- Reducing Porting

Loop Transformation

- PICO's coding restriction
  - No outer-loop around sequence of loops

Possible

```
f1(){
  for(int i = 0 ; i < 100; ++i){
    for(int j = 0 ; j < 100; ++j){
      for(int k = 0 ; k < 100; ++k){
        …code…
      }
    }
  }
  for(int ii = 0 ; ii < 100; ++i){
    for(int jj = 0 ; jj < 100; ++j){
        …code…
    }
  }
}
```

Impossible

```
f2(){
  for(int i = 0 ; i < 100; ++i){
    for(int j = 0 ; j < 100; ++j){
      for(int k = 0 ; k < 100; ++k){
        …code…
      }
    }
    for(int ii = 0 ; ii < 100; ++i){
      for(int jj = 0 ; jj < 100; ++j){
          …code…
      }
    }
  }
}
```

Loop Transformation

- We need transformation!
- Outer Loop Case 1 : Sequential Loop Jamming

```
f1(){
   for(int i = 0 ; i < M; ++i){
      for(int j = 0 ; j < N0; ++j){
         L0;
      }
      for(int j = 0 ; j < N1; ++j){
         L1;
      }
   }
}
```

→

```
f1(){
   for(int i = 0 ; i < M; ++i){
      for(int j j= 0 ; jj < N0+N1; ++jj){
         if(jj < N0){
            j = jj;
            L0;
         }
      }else{
            j = jj;
            L0;
         }
      }
   }
}
```

Loop Transformation

- We need transformation!
- Outer Loop Case 2 : Unrolling

```
f1(){
  for(int i = 0 ; i < M; ++i){
    for(int j = 0 ; j < N0; ++j){
      L0;
    }
    for(int j = 0 ; j < N1; ++j){
      L1;
    }
  }
}
```

→

```
f1(){
  for(int i = 0 ; i < M; ++i){
    for(int j = 0 ; j < N0; ++j){
      L0;
    }
    #pragma unroll j
    for(int j = 0 ; j < N1; ++j){
      L1;
    }
  }
}
```

Loop Transformation

- We need transformation!
- Outer Loop Case 3 : Task Overlap

```
f1(){
   for(int i = 0 ; i < M; ++i){
      for(int j = 0 ; j < N0; ++j){
         L0;
      }
      for(int j = 0 ; j < N1; ++j){
         L1;
      }
   }
}
```

→

```
//PPA code
f1(){
   for(int j = 0 ; j < N0; ++j){
      L0;
   }
   for(int j = 0 ; j < N1; ++j){
      L1;
   }
}
```

```
//Driver code
Int main(){
   for(i = 0 ; i < M; i++)f1();
}
```

Loop Transformation

- We need transformation!
- Outer Loop Case 4 : Fully Parallel
  - Not possible when there's a feedback from L1 to L0

```
f1(){
  for(int i = 0 ; i < M; ++i){
    for(int j = 0 ; j < N0; ++j){
      L0;
    }
    for(int j = 0 ; j < N1; ++j){
      L1;
    }
  }
}
```

→

```
f1(){
  for(int i = 0 ; i < M; ++i){
    for(int j = 0 ; j < N0; ++j){
      L0;
    }
  }
  for(int i = 0 ; i < M; ++i){
    for(int j = 0 ; j < N1; ++j){
      L1;
    }
  }
}
```

Loop Transformation

- We need transformation!
- Outer Loop Case 5 : Parallel Loop Merging

```
f1(){
  for(int i = 0 ; i < M; ++i){
    for(int j = 0 ; j < N0; ++j){
      L0;
    }
    for(int j = 0 ; j < N1; ++j){
      L1;
    }
  }
}
```

→

```
f1(){
  for(int i = 0 ; i < M; ++i){
    for(int j = 0 ; j < N0; ++j){
      L0;
      L1;
    }
  }
}
```

Array Partitioning

- Assume calculation on line 5 should take only 1 cycle by constraint
  - Need 4-port memory for original design
  - Need single port memory after partitioning

```
1 char in[4][1024], out[1024];
2 void app(){
3    int i;
4    for(i = 0 ; i < 1024; ++i){
5       out[i] =
     (in[0][i]+in[1][i]+in[2][i]+in[3][i])/4;
6    }
7 }
```

```
1 char in0[1024], in1[1024], in2[1024],
in3[1024], out[1024];
2 void app(){
3    int i;
4    for(i = 0 ; i < 1024; ++i){
5       out[i] =
     (in0[i]+in1[i]+in2[i]+in3[i])/4;
6    }
7 }
```

Logical vs Bitwise Operation

- Logical operators often lead to more hardware comparators than the bitwise operators.
- Replace “&&” and “||” with “&” and “|” with caution

Reducing Multiplier

- Make PICO recognize common sub-expressions and reduce number of multiplier
- Assoc. and dist. transformation are not always bit equivalent in 2's complement arithmetic due to overflow and underflow

//original	//transformation 1	//transformation 2
x = -a*b+c*(d+e);	x = -(a*b)+c*(d+e);	tmp1 = a*b;
y = a*b+c*d+c*e;	y = (a*b)+c*(d+e);	tmp2 = c*(d+e);
		x = -m1+ m2;
		y = m1 + m2;

Reducing Porting

- Reduce memory porting by restructuring the code

```
if(mode == 0)
   m = X[a][b];
else if (mode == 1)
   m = X[c][d+e];
else
   m = X[d][e];
```

```
if(mode == 0){
   tmp1 = a;
   tmp2 = b;
}
else if(mode == 1){
   tmp1 = c;
   tmp2 = d+e;
}
else if(mode == 2){
   tmp1 = d;
   tmp2 = e;
}
m = X[tmp1][tmp2];
```

Contents

- PICO Express
  - Introduction
  - Design Flow
  - Memories and Arrays
  - Performance Specification
  - Coding Restrictions
  - Coding Issues
  - **Exercise**

Exercise 2

- Lint error debugging
  - Debug the program “exercise2”
  - It is a modified version of “three_filters” for testing inlining
  - It has no compile error but has a lint error
  - Run “Lint” procedure to detect error and with this information, compare “excercise2” and “three_filters”
  - You should edit the source file in implementation directory to make it take effect

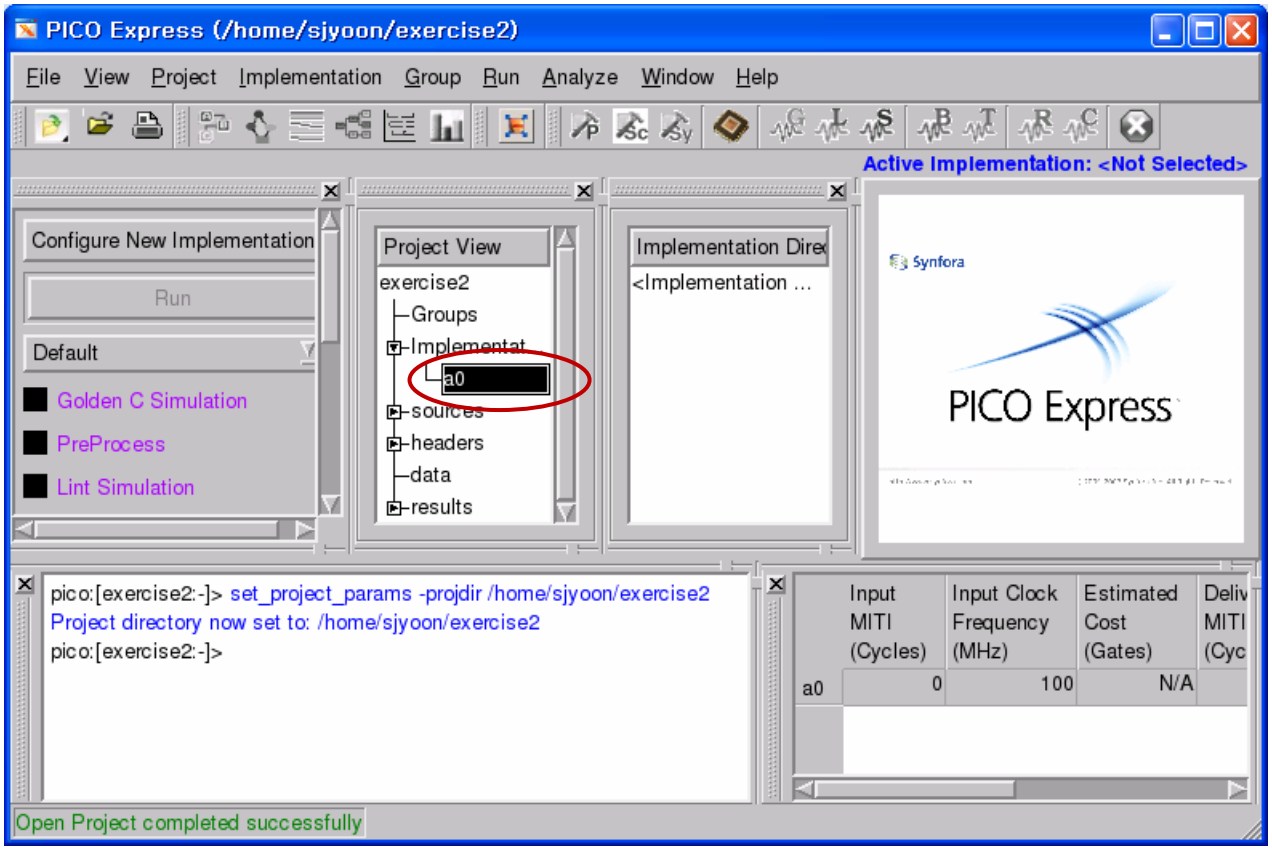
Exercise 2

- Execute the PICO Express

```
viper:~> cd exercise2
viper:~/exercise2> peg&
[1] 13726
viper:~/exercise2> PICO EXPRESS 07.02-2
Copyright (c) 2004-2007 Synfora, Inc.  All rights reserved.
Build Time 07/16/07 06:56:56
```

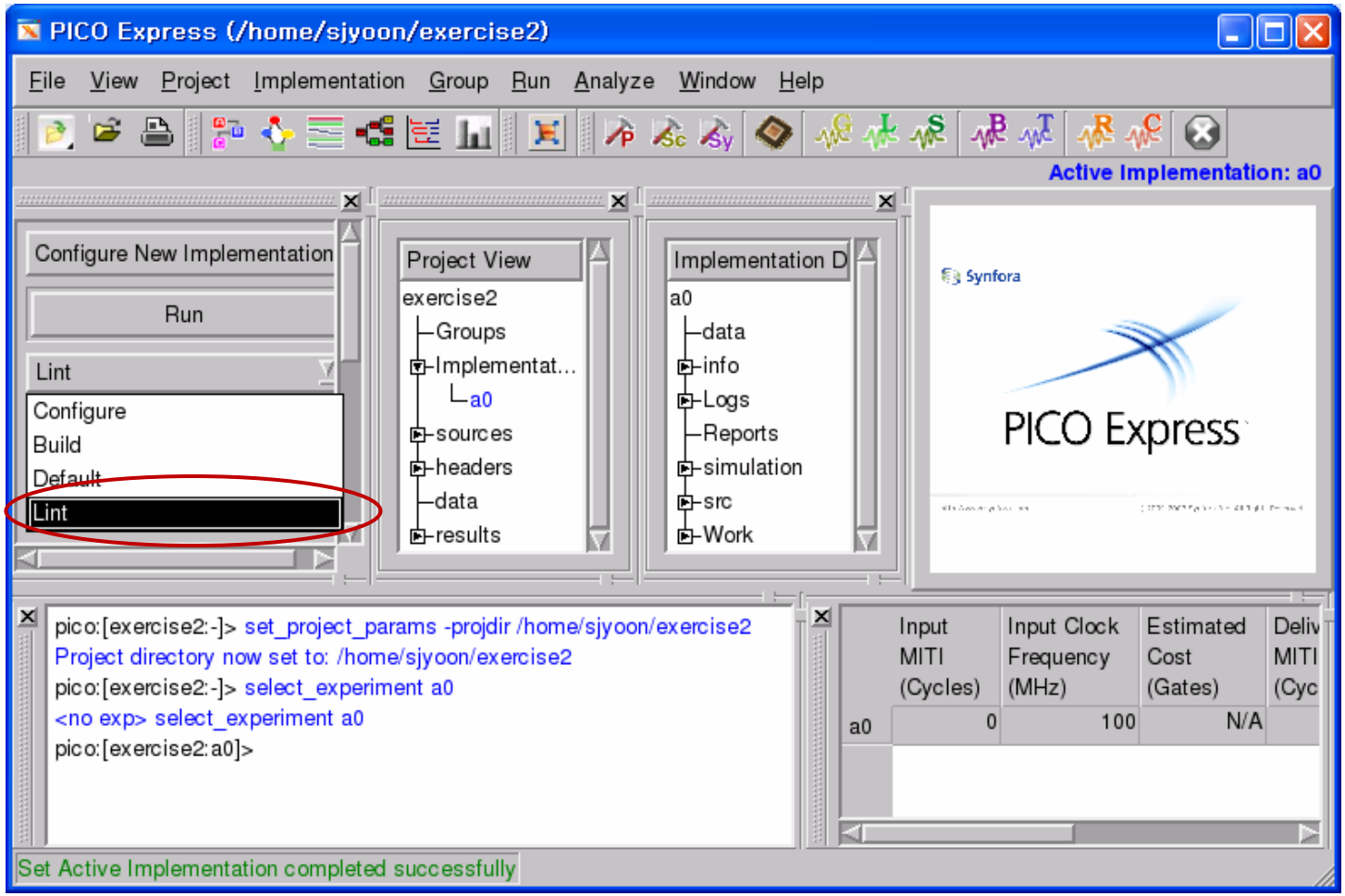
Exercise 2

- Select the architecture a0 on Project View by double clicking



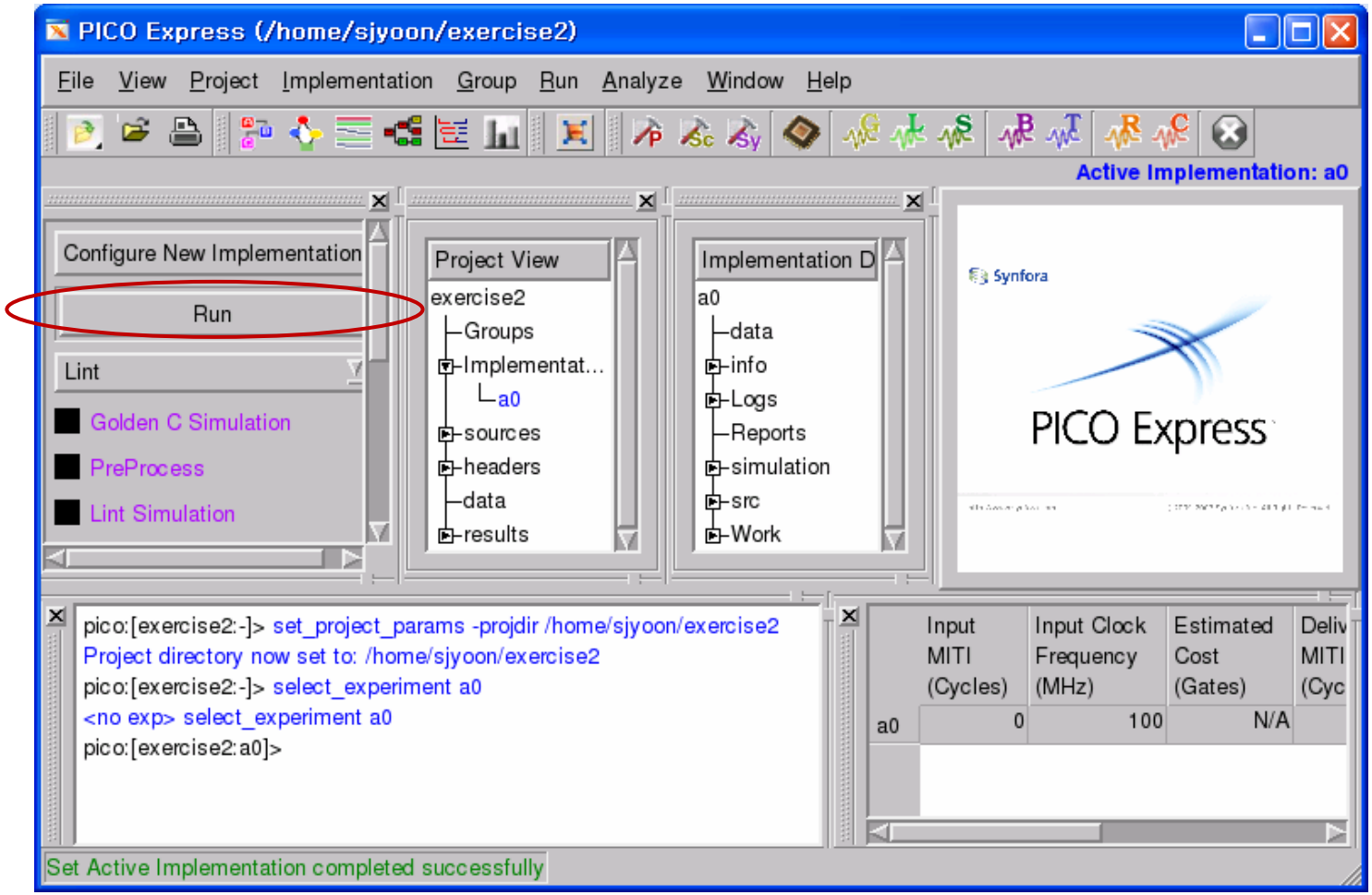
Exercise 2

- Select Lint Process



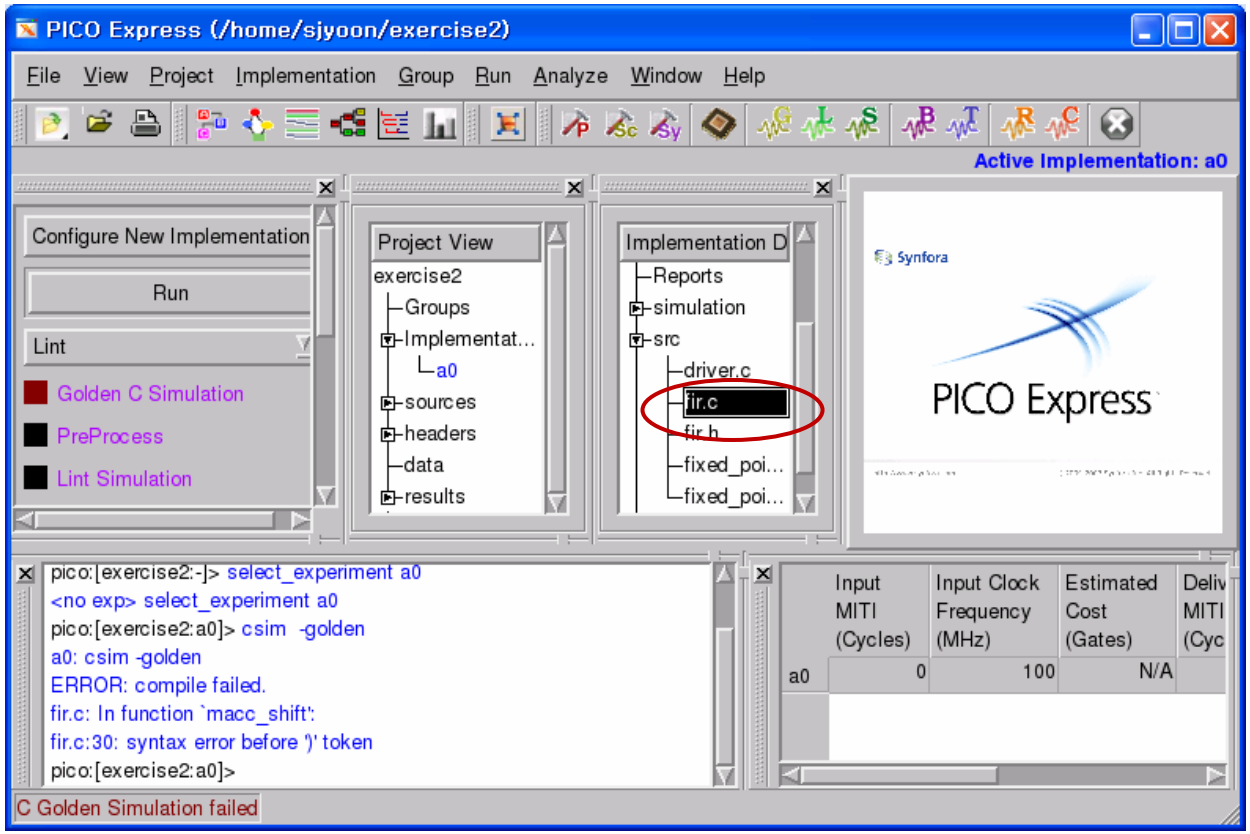
Exercise 2

- Run



Exercise 2

- Compile failed because of syntax error
- Edit fir.c file on Implementation View



Exercise 2

- Edit

fir.c (~/exercise2/a0/src) - GVIM

File Edit Tools Syntax Buffers Window Help

```
25 short y3[L][M2-N];
26
27 char task_num;
28
29 short macc_shift(short a, short b, short partial_sum, int shift_am
30   short result = partial_sum + ((a*b) >> shift_amount);
31   return result;
32 }
33
34 void fir() {
35   int i,j;
36
37   // Create a fixed, 31-point rectangular window with cutoff frequ
38   const short w3[N] = {0x0032, 0x000f, 0xffe3, 0xffb6, 0xff92, 0xf
39                        0xfff2, 0x0051, 0x00c2, 0x013a, 0x01ab, 0x0
40                        0x0243, 0x0207, 0x01ab, 0x013a, 0x00c2, 0x0
41                        0xff87, 0xff7f, 0xff92, 0xffb6, 0xffe3, 0x0
42
                                                    25,1          45%
```

Exercise 2

- Lint Error

```
WARNING:2039:lsim:pico_lint.c:338
In function 'macc_shift' called at fir.c:50
WARNING:fir.c:30: Shift value 18446744073709551587 out of legal range (0,31)
END WARNING
WARNING:2000:lsim:pico_lint.c:507
WARNING:fir.c:52: Width overflow for "y[0]" type i13, value -31998 (0xffff8302).
END WARNING
WARNING:2000:lsim:pico_lint.c:507
WARNING:fir.c:61: Width overflow for "y2[0][0]" type i13, value -4643
(0xffffeddd).
END WARNING
WARNING:2000:lsim:pico_lint.c:507
WARNING:fir.c:69: Width overflow for "y3[0][40]" type i13, value 4158 (0x103e).
END WARNING
```

Exercise 2

- Negative shift amount?

fir.c (~/exercise2/a0/src) - GVIM

File Edit Tools Syntax Buffers Window Help

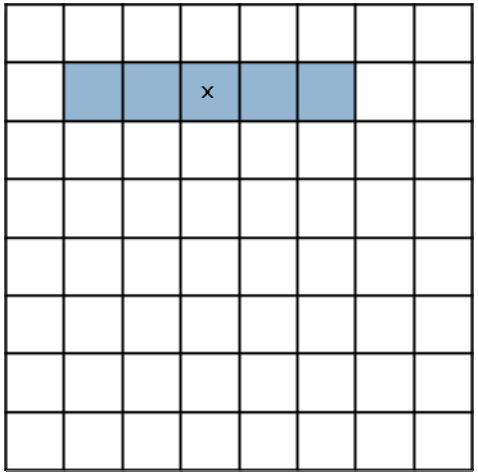
```
50          sum = macc_shift(w[j], x[task_num][i+j], sum, (EXPY+EXPX+EXPW));
51        }
52        y[i] = sum;
53      }
54
55
56      for (i=0; i < M2-N; i++) {
57        short sum = y2[task_num][i];
58        for (j=0; j < N; j++) {
59          sum = macc_shift(w2[j], y[i+j], sum, (EXPY-EXPX-EXPW));
60        }
61        y2[task_num][i] = sum;
62      }
63
64      for (i=0; i < M2-N; i++) {
65        short sum = y3[task_num][i];
66        for (j=0; j < N; j++) {
67          sum = macc_shift(w3[j], y[i+j], sum, (EXPY-EXPX-EXPW));
                                                          50,70          92%
```

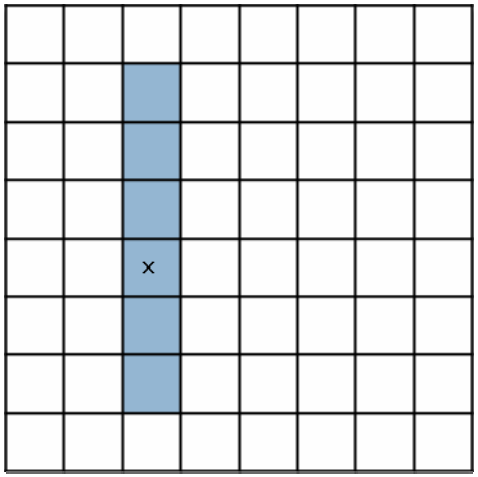
Exercise 3

- Fibonacci
  - Basic approach
    - Implement simple program by your hand
      - Write driver code and ppa code
      - Driver code is for initializing the input, setting up the PPA and printing the output.
      - PPA code is for synthesizing the hardware. Should be synthesizable C code which meets PICO's coding constraints
    - If you are not familiar with PICO, it is okay to use existing programs for framework
  - Sliding window
    - See "Writing C Application" page 32 for reference (http://iris.snu.ac.kr/synfora/Writing_C_apps.pdf)

Exercise 4

- Implement 5-tap FIR FILTER without streaming
  - 5-tap filter needs 5 pixel at once to filter
  - Create intermediate pixel “x” using a 5-tap FIR filter. Filter all pixels horizontally in this manner(top left to bottom right)
  - And then, create final pixel “x” using a 5-tap FIR filter. Filter all pixels vertically in this manner. (top left to bottom right)





Exercise 4

- Horizontal filtering

```
for(j = 0 + MARGIN; j < HEIGHT+ MARGIN; j++){
    for(i = 0 + MARGIN ; i < WIDTH + MARGIN; i++){
        short output = (c[4] * yin[j][i-2] +
                c[3] * yin[j][i-1] +
                c[2] * yin[j][i  ] +
                c[1] * yin[j][i+1] +
                c[0] * yin[j][i+2] )>>3;
        yinter[j][i] = output;
    }
}
```

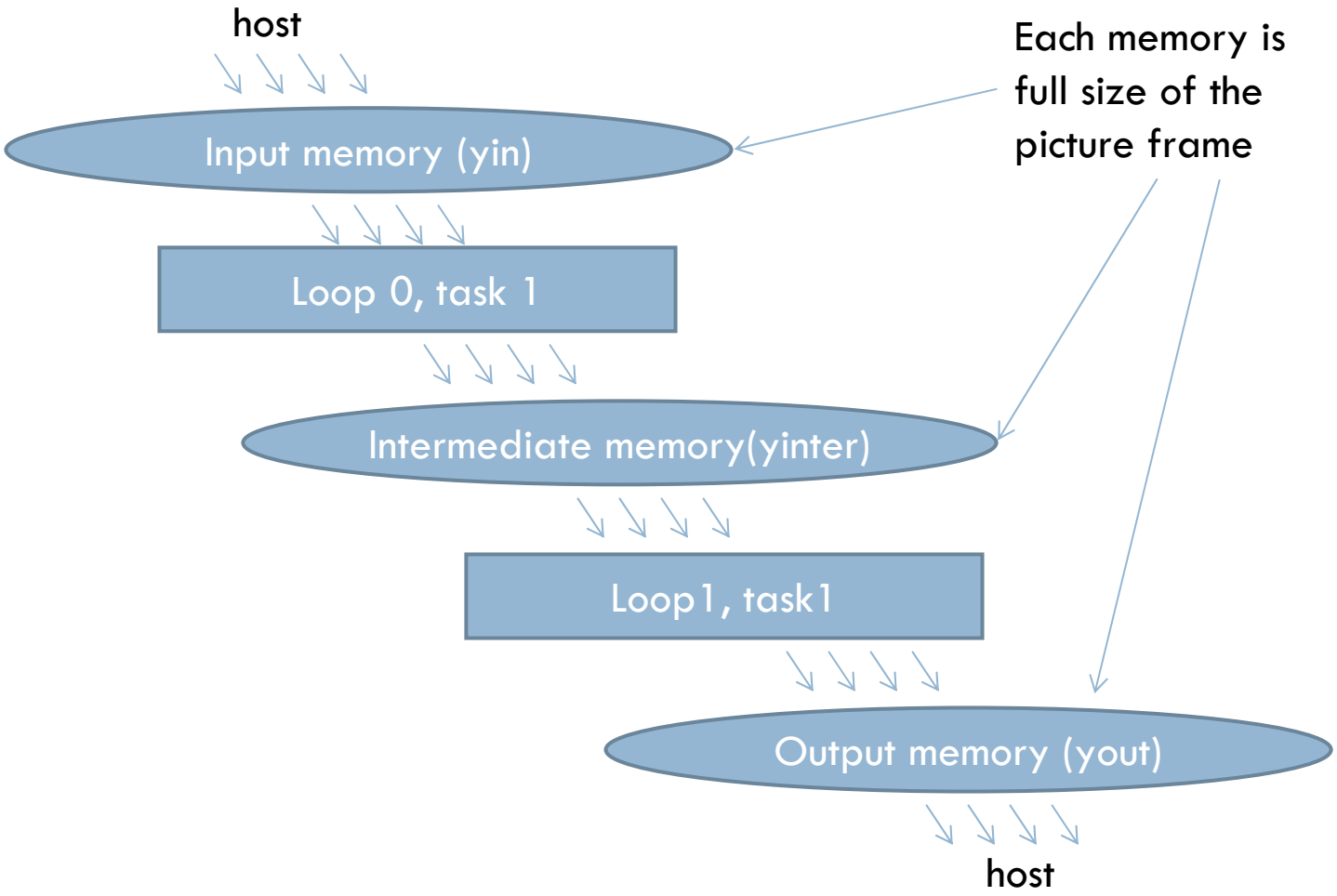
Exercise 4

- Vertical filtering

```
for(jj = 0 + MARGIN; jj < HEIGHT + MARGIN; jj++){
    for(ii = 0 + MARGIN ; ii < WIDTH + MARGIN; ii++){
        short output = (c[4] * yinter[jj-2][ii] +
                c[3] * yinter[jj-1][ii] +
                c[2] * yinter[jj][ii  ] +
                c[1] * yinter[jj+1][ii] +
                c[0] * yinter[jj+2][ii] )>>3;
        yout[jj][ii] = output;
    }
}
```

Exercise 4

- Current design operates…

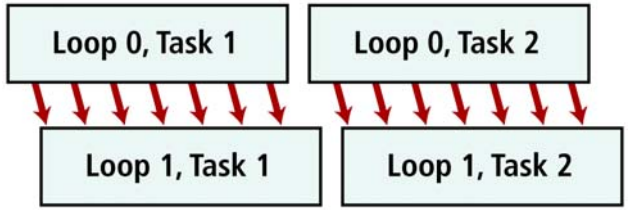


Homework

- Implement 5-tap FIR FILTER with internal streaming and sliding window to improve performance
- Problems of previous implementation
  - Memory for full size of picture frame is needed – large memory
  - Large number of memory access occurs
  - It prevents parallel execution of Loop0 and Loop1 because of data dependency between L0 and L1
  - These problems can be solved by **streaming** and **sliding window**

Streaming Data (1)

- Streaming data enables parallel execution of communicating loop nests
  - Communicate via 2-way handshake
  - Time-independent synchronization
  - "Block-on-read" – no peeking



- Specification for external streams:
  - Use `<type> pico_stream_input_<name>(void)` (input)
  - Use `void pico_stream_output_<name>(<type>)` (output)
  - The user should write these procedures to define the communication between driver and PPA via streams

time

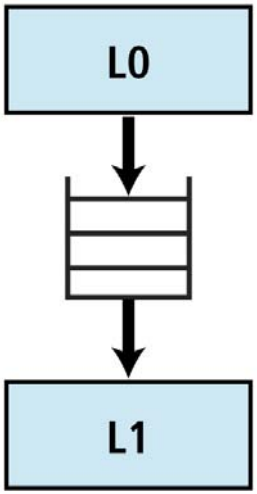
- Specification for internal streams:
  - Use `FIFO(<name>,<type>)` to declare inter-loop FIFO
  - Use `pico_stream_input_<name>` and `pico_stream_output<name>`
  - The stream procedures are automatically generated by the FIFO macro

Streaming Data (2)

- What happens when we call pico_stream_*?
  - In software – procedure call
    - The code for the pico_stream_* procedure is executed
    - For internal streams this procedure comes from the FIFO macro and is predefined by PICO Express
    - For external streams this procedure is user-defined and can do anything the user wishes
  - In hardware – data handshake
    - Each data stream has 3 signals - data, ready, request
    - When pico_stream_* call is encountered, the request is asserted (either for read or write)
    - If the corresponding ready is true, the transaction takes place and the PA proceeds
    - If the ready is false, the PA stalls and continues asserting request until ready goes true

Streaming Data (3)

- Without streams, the PPA would always execute in a pre-determined time
  - Streams introduce variability due to dynamic synchronization and flow control
    - Waiting for input data to be available
    - Waiting for output buffer to be free
    - Producer and consumer block independently
    - The computation is still deterministic



  - Streams may cause deadlocks due to bounded buffers

```
#pragma fifo_length x 4
```

  - Deadlocks can always be removed by increasing FIFO sizes. However, this may indicate unintended sequentialization in the code

Streaming Data(4)

- Example Code
  - Function pico_stream_output_y() is for external streaming. It is manually implemented in the driver code by user and does something, for example, writing value of y[j] to output file.
  - Function pico_stream_output_z() and pico_stream_output_y() are for internal streaming. It's code is automatically generated by PICO. Function pico_stream_output_z() writes "x[j] + offset" to FIFO and pico_stream_input_z() reads value from FIFO.

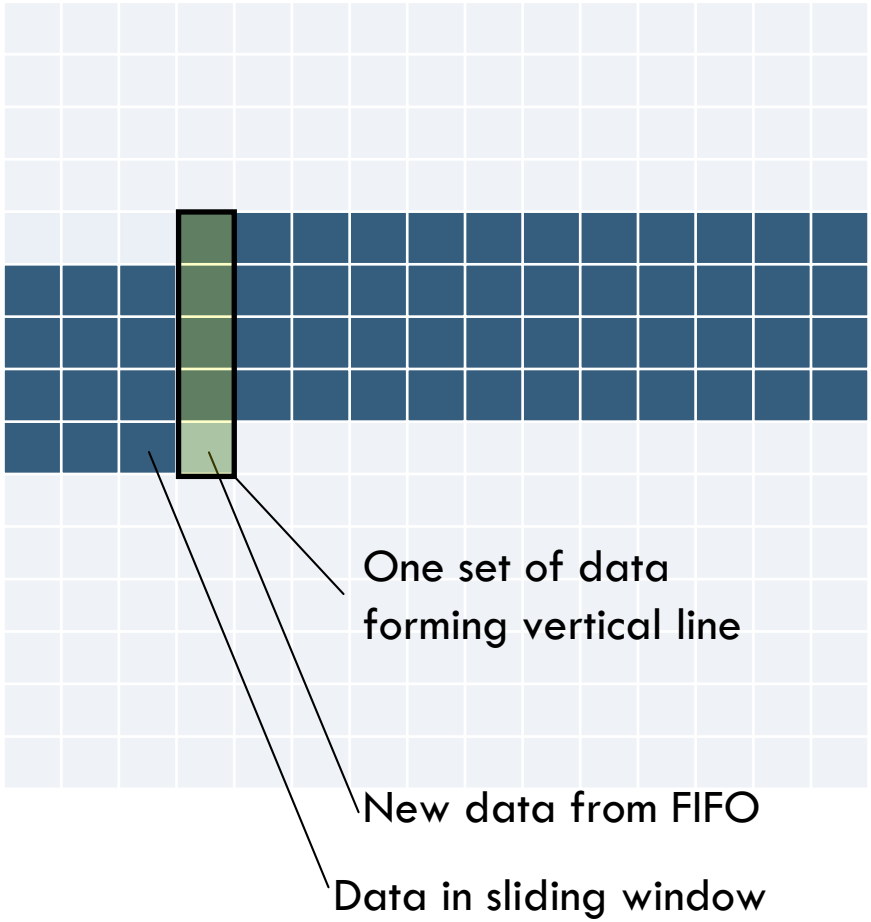
```
int x[100],y[100],z[100];
FIFO(z,int)
extern int pico_stream_output_y(int);
int offset;

void ppa(void) {
  int j;

  for(j=0;j<100;j++) {
    y[j] = x[j]*x[j];
    pico_stream_output_z(x[j]+offset);
  }
  for(j=0;j<100;j++){
    pico_stream_output_y(y[j]);
  }
  for(j=0;j<100;j++){
    z[j] = pico_stream_input_z();
  }
}
```

Sliding Window

- A sliding window is variable with the following properties:
  - It is declared as a one-dimensional array in procedure scope.
  - All references have compile-time constant indices.
  - It is an argument to a single pico_shift procedure call.
- Within a for-loop, at most shift locations can be written, where shift is the shift argument to the pico_shift call. In addition, the locations written must have consecutive indices.
- The pico_shift() call takes two arguments:
  - The array to be shifted
    - Must be a single dimensional array
    - Must be declared in procedure scope
  - The shift amount
    - Must be a compile-time constant
    - Must be between one and the array size minus one

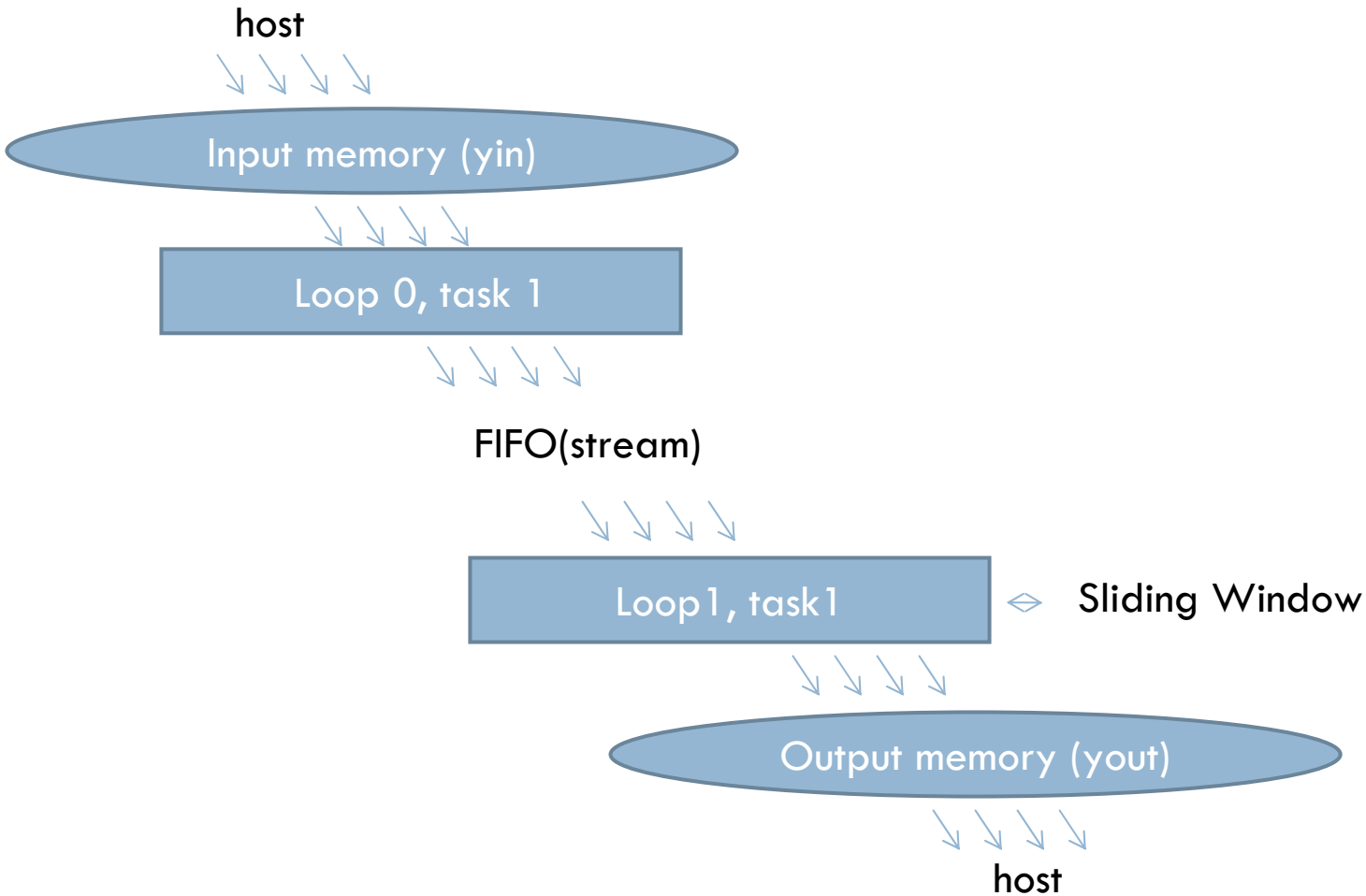
Homework

- FIFO is used to keep the data from L0 which filters horizontally
- What L1 needs is the data in vertical order while FIFO fires the data in horizontal order.
  - In the L1, Someone should keep the data from FIFO and provides data access in vertical order
  - Sliding window can be used here
- Sliding windows should keep more than 4 lines to provide vertical data access to L1



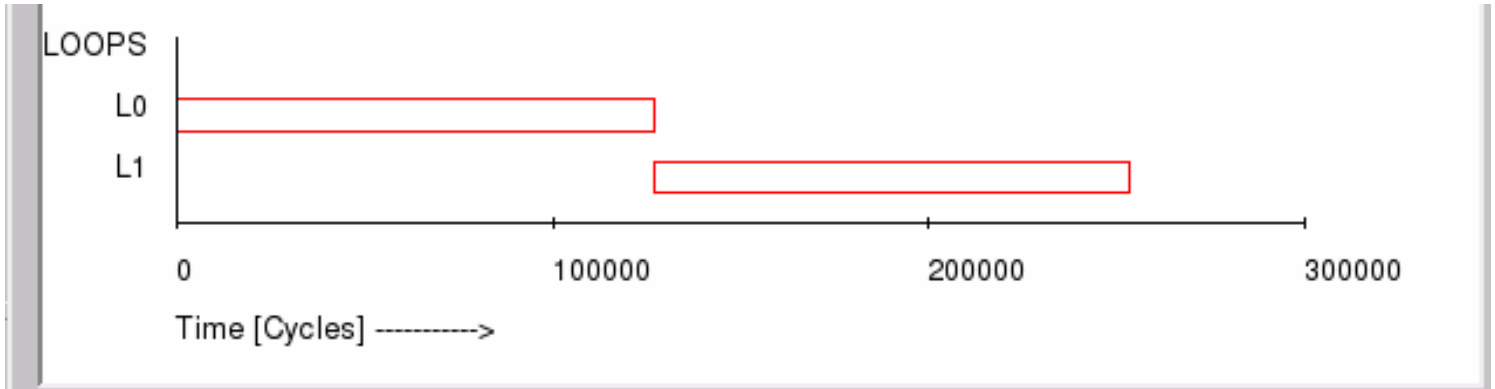
Homework

- Revised structure with stream and sliding window



Homework

- Sample Result of fir filter with shared memory

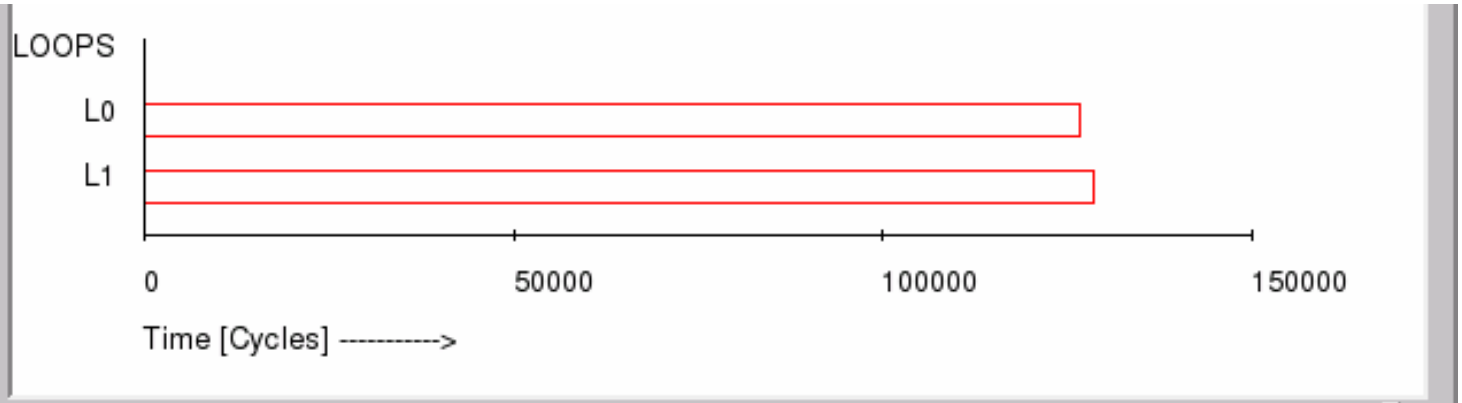


Items	Start	Stop	Stall
Task0	8	253448	0
L0	8	126728	0
L1	126728	253448	0

	Input MITI (Cycles)	Input Clock Frequency (MHz)	Estimated Cost (Gates)	Delivered MITI (Cycles)	Delivered Task Latency (Cycles)	Delivered Throughput (Tasks per second)
imp000	0	100	14,533	N/A	253,441	394

Homework

- Sample result of fir filter with stream



Dynamic Performance [imp000]

	Input MITI (Cycles)	Input Clock Frequency (MHz)	Estimated Cost (Gates)	Delivered MITI (Cycles)	Delivered Task Latency (Cycles)	Delivered Throughput (Tasks per second)
imp000	0	100	76,375	N/A	128,481	778