FTF FREESCALE TECHNOLOGY FORUM 2015

How To **Demystify Simultaneous** Multi-Threading

FTF-ACC-F1259

John West | Systems Architect

J U N E . 2 0 1 5

freescale™ Automotive

External Use

Intro

- What will you learn?
- Why are we here?

Objective: At the end of this course, you will be able to :

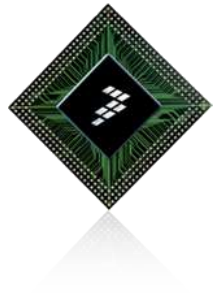
1) Understand terminology used in multithreading
2) Understand different types of multithreading and their pro's and con's
3) Know why FSL has moved forward with simultaneous multithreading (SMT)

Agenda

- Terminology
- Differences between Multiple Issue and Multiple Threaded
- Blocked multithreading
- Simultaneous multithreading (SMT)
- Review of current architecture and automotive code
- How SMT helps improve performance

Terminology

Definition of Terms



- Single Issue Core
  - A single issue core can issue a maximum of one assembly instruction on every clock.
- Dual Issue Core (Multiple Issue)
  - A dual issue core can issue a maximum of two assembly instructions per clock.
  - There are restrictions on what can be dual issued, however typically most instructions can be.
  - FSL automotive cores in 90nm, 55nm, and 28nm are dual issue core.
    - Future cores maybe more than dual issue (many issue).

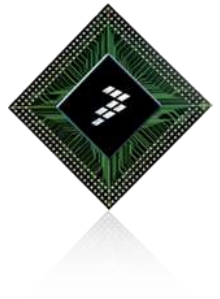
What Can Dual Issue?

- The following pairs can dual issue on e200zX-SMT core

	Branch	Ld/st	Scalar Integer	Scalar Mul	Scalar Float	Vector Integer	Vector Mul	Vector Float
Branch	✓1	✓	✓	✓	✓	✓	✓	✓
Ld/st	✓		✓	✓	✓	✓	✓	✓
Scalar Integer	✓	✓	✓	✓	✓	✓	✓	✓
Scalar Mul	✓	✓	✓	✓	✓	✓		✓
Scalar Float	✓	✓	✓	✓	✓	✓	✓	
Vector Integer	✓	✓	✓	✓	✓	✓	✓	✓
Vector Mul	✓	✓	✓		✓	✓		✓
Vector Float	✓	✓	✓	✓		✓	✓	

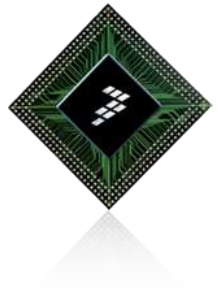
[1]Interthread branches only

Definition of Terms (cont)



- What is a thread
  - A thread is a sequence of programmed instructions that can be controlled by a scheduler, usually an Operating System (OS).
    - Could be an entire application (i.e. complete engine control application)
    - Could be a time task (i.e. 10ms, 20ms, etc)
    - An interrupt service routine
    - Could be as simple as one software routine (i.e. table lookup)
  - All automotive applications already have thread(s)
- Multithreading
  - System that contains multiple threads and a means of controlling and executing the threads

Types of multithreading cores



- Blocked multithreading
  - Simplest type of multithreading. One thread runs until it is blocked by an event (often a long latency memory access). The processor will switch to the next thread and run that until it is blocked or completed.
  - Context and register set is shared amongst all threads and a context switch must be performed to run the next thread.
  - This is not a good solution for automotive application due to context switch time and stalling important tasks can cause missing real-time events.

This is NOT the type of multithreading FSL is implementing

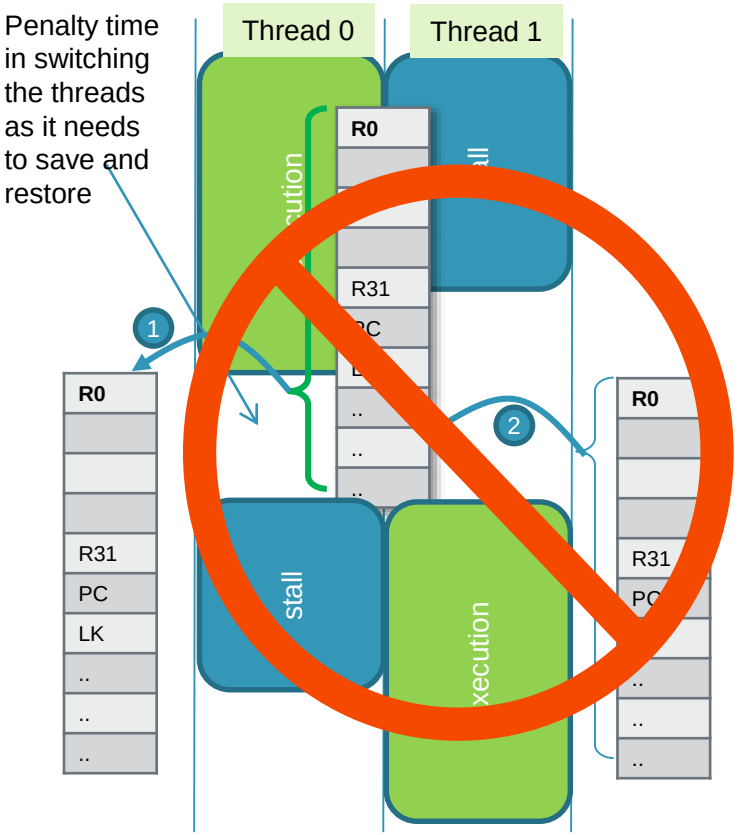
Types of multithreading cores (cont)

- Simultaneous multithreading
  - An advanced type of multithreading that's allows multiple threads to run at the same time. One thread does not cause other threads to stall by taking ownership of shared resources.
  - Each thread has it's own register set and context so there is no costly context switch.
  - This allows for better utilization of core resources when running cache unfriendly branchy automotive code.

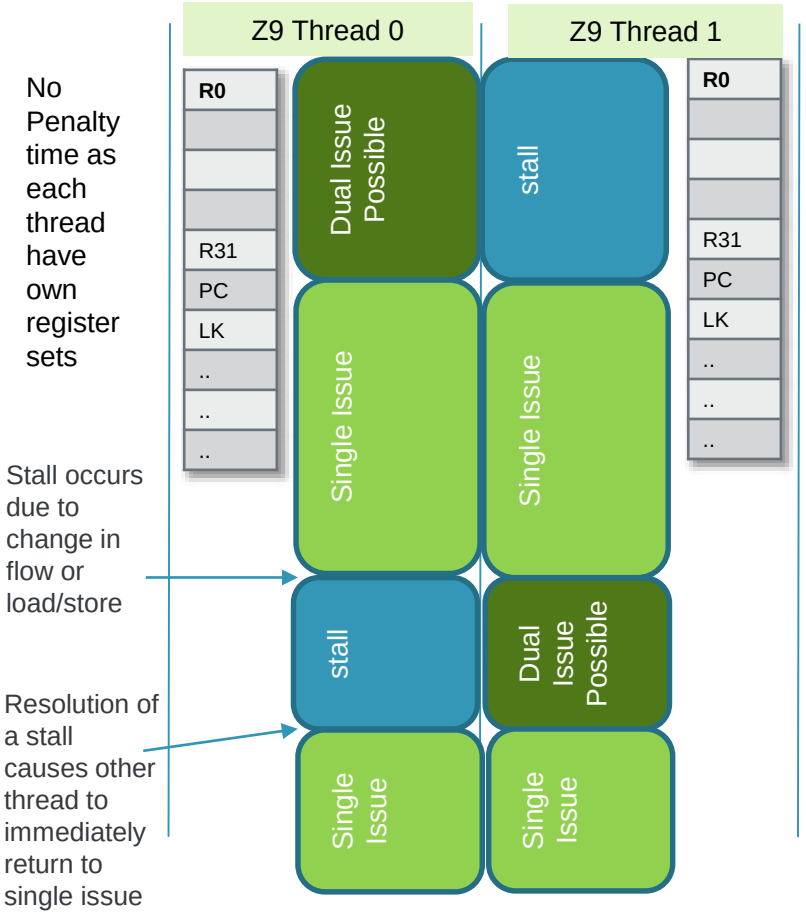
This is the type of multithreading FSL is implementing.

Implementation of blocking multithreading

Freescale Implementation of simultaneous multi threads



Thread execution is mutually exclusive



Thread execution is not mutually exclusive

Why SMT?

Why multithreading?

- One multithreaded core has look and feel of two single threaded cores from user model view
- Works very well for cache unfriendly code (automotive code very cache unfriendly)
- Optimize for performance/mW
- Same or better single-threaded performance as a single threaded core
- Reduce the number of cores required at the SoC level, reducing area and power consumption
- Expandable to >2 threads in the future
- Allows more efficient use the CPU resources
- Simplified core platform/xbar implementation

Core Capabilities

- A dual issue core is capable of executing up to two instructions per clock.
- Program flow, data dependencies, interrupts, branches, etc determine how many instructions can be executed each clock.
- Automotive code is typically cache unfriendly, heavily branched (approximately every 4-6 instructions), interrupts often, and can contain many data dependencies.
  - This results in a high number of zero and single issue cycles.
- Performance is a direct result of combination of zero, single, dual issue cycles.
- Freescale goal with z200zX-SMT core family is to maximize the number of dual issue cycles, increase number of single issue cycles, and minimize number of zero issue cycles.

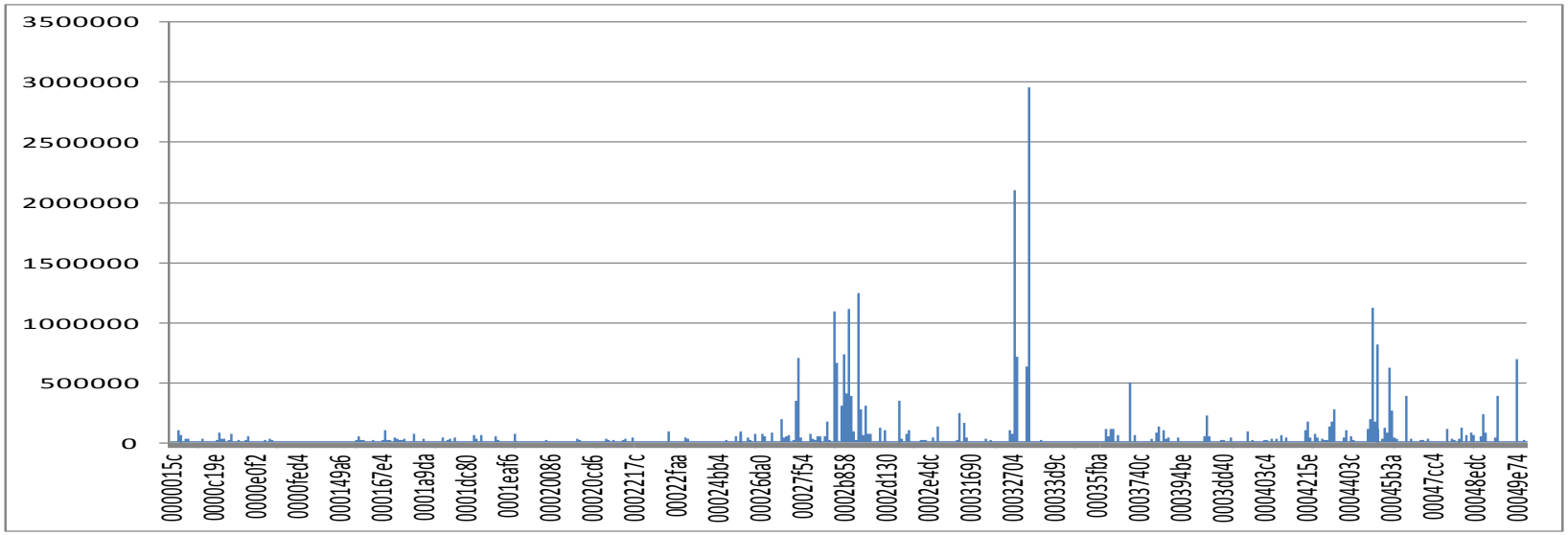
Amroth – Complete Automotive Application

- High percentage of single cycle integer instructions
- Code profile does not contain floating point code
- No SIMD instructions used
- Very low usage of mul and div
- Very flat code profile (lots of branches)
- Verified as representative of actual application by customers.

Instruction	Count	Percent
Integer		
Fast (1 cycle)	17746725	47.6
mul	669030	1.8
div	138406	0.4
Memory		
Load	7895236	21.2
Store	4125896	11.1
Branches		
Conditional	5206232	14.0
Unconditional	1520876	4.1
Total	**37302401**	**100**

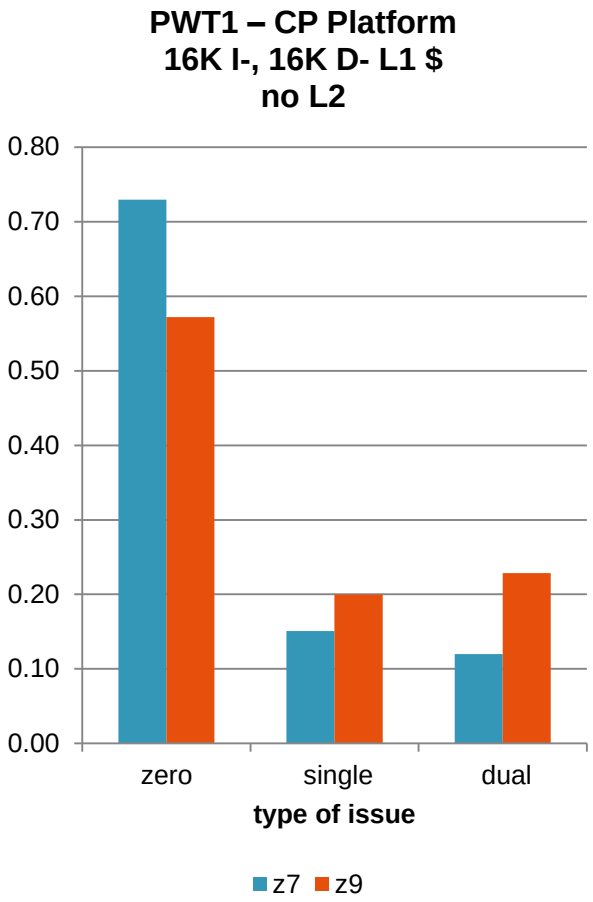
Automotive Code Characteristics

- Flat nature of the code makes limited use of small caches
  - Cache lookup performed on every clock even when waiting for fetches from flash. Results in high current consumption for L1 cache
- Core speed is faster than flash access speed resulting in high % of memory stall cycles (zero instructions being issued)
- Limited dual issue effectiveness due to change-of-flow frequency

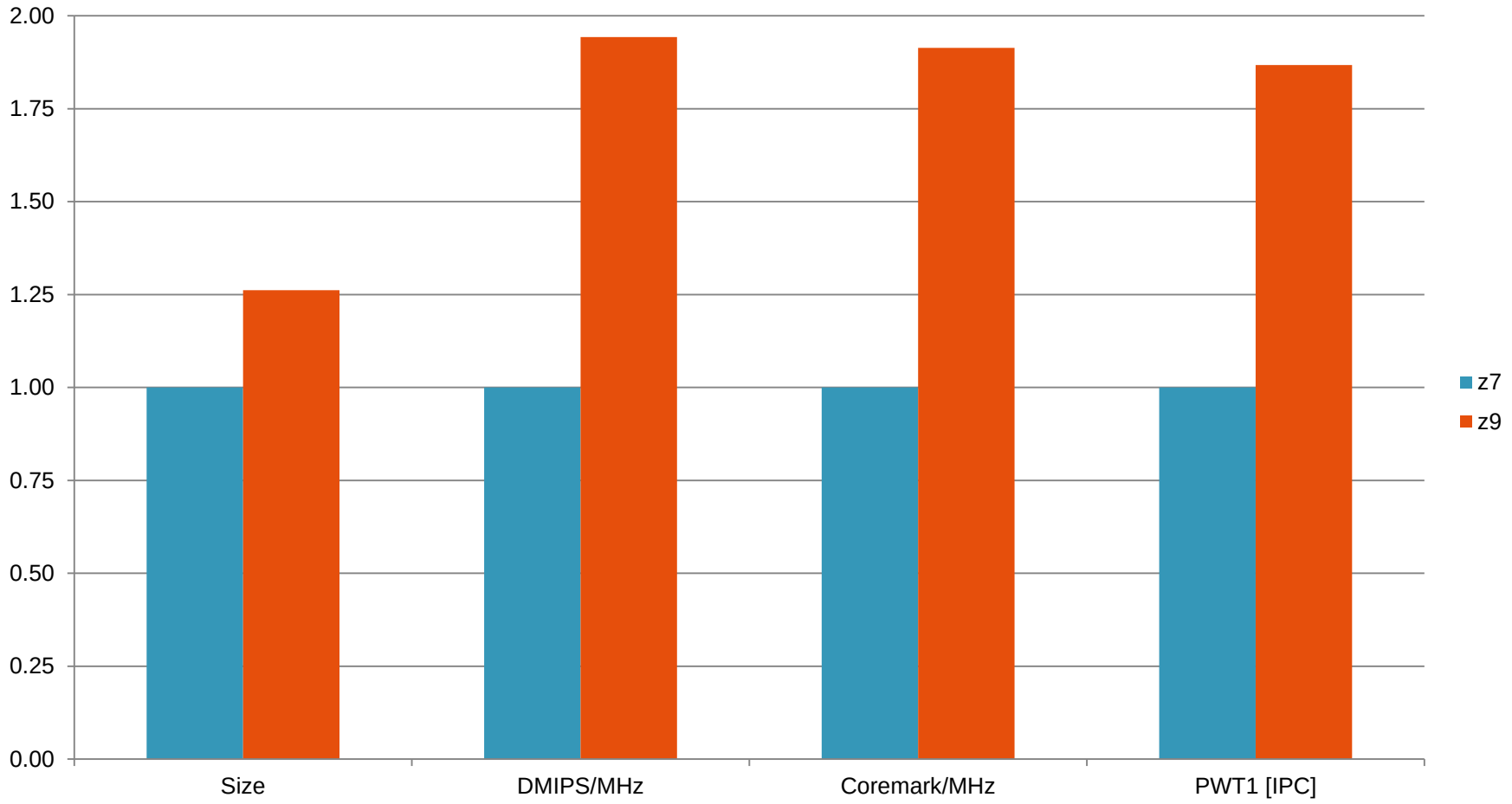


Measured Core Platform Performance Data

- Zero instructions issued indicates core is stalled waiting for memory access and/or data dependency to resolve
- A single thread has less chance to dual issue due to branch frequency
- Two threads greatly reduces the cycles with 0-inst issued
- Two threads allows both threads to single issue simultaneously, resulting in more dual issues per cycle and the resulting increased performance



e200zX-SMT cores offer large performance/gate increase



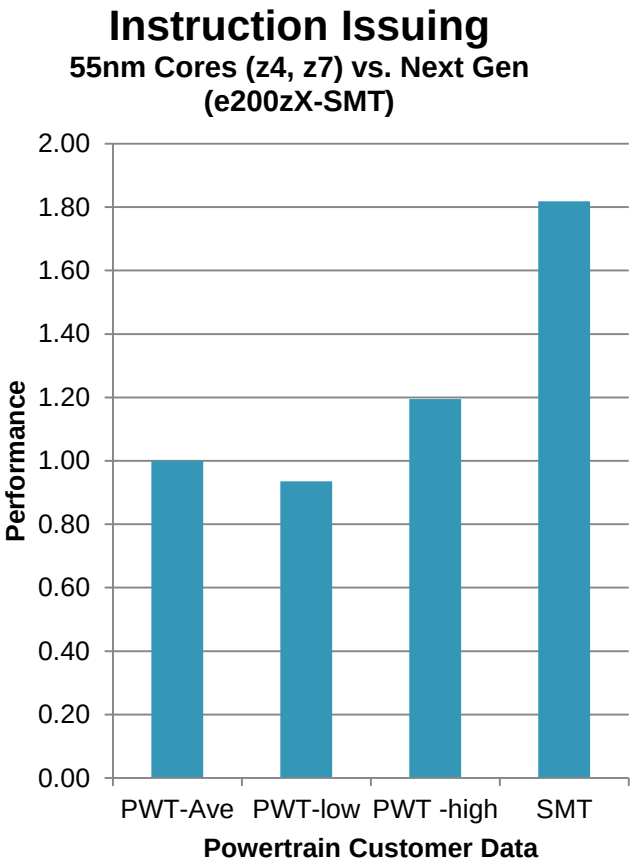
- A single e200zX-SMT core approaches 2x the performance of a z7 core at the incremental size cost of 1.25x at same clock frequency.

Simultaneous multithreading (SMT) optimized for cache unfriendly auto code

PowerTrain Customer Instruction Issue Profiles

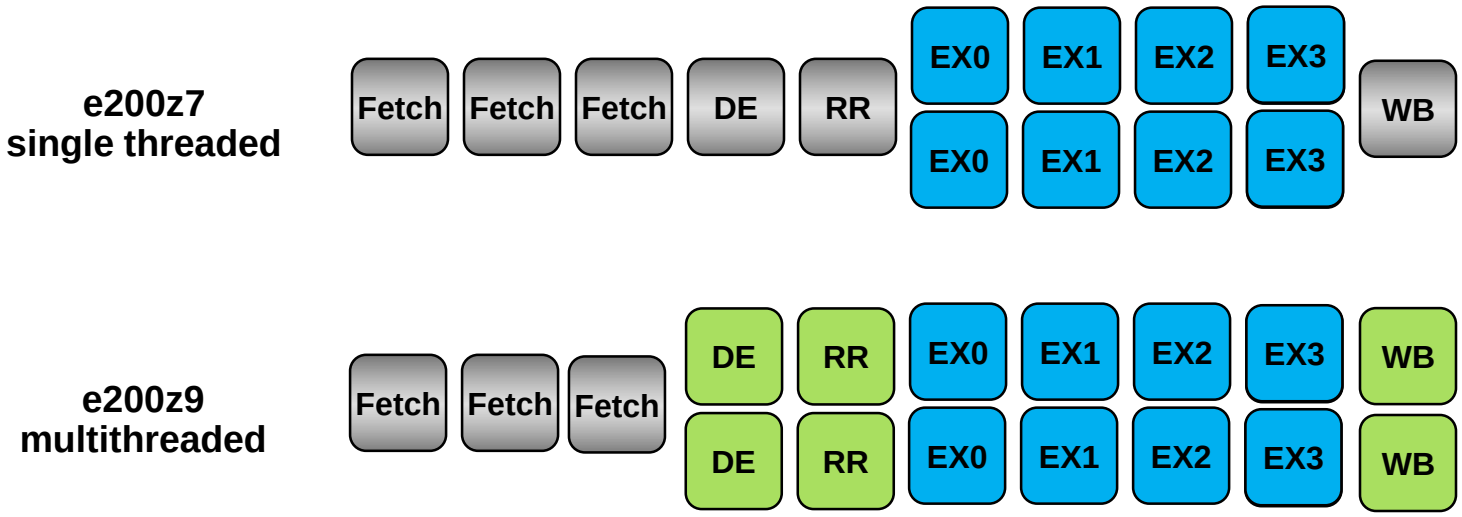
	PWT-Ave	PWT-Low	PWT-high	SMT
zero issue	73	75	68	55
single issue	15	14	18	20
dual issue	12	11	14	25

- Zero instructions issued indicates core is stalled waiting for memory access and/or data dependency to resolve
- A single thread has less chance to dual issue due to branch frequency
- Two threads greatly reduces the cycles with 0-inst issued
- Two threads allows both threads to single issue simultaneously, resulting in more dual issues per cycle and the resulting increased performance



Power Architecture e200z*X* Pipeline Comparisons

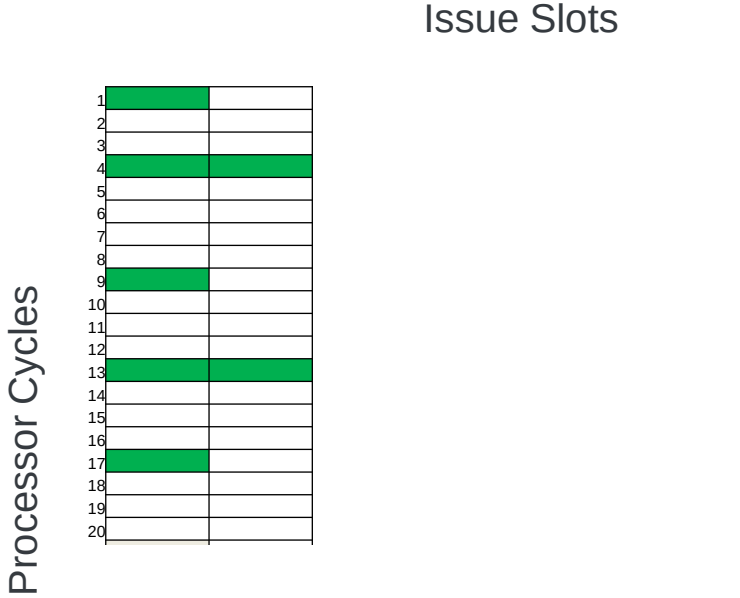
Pipe Stages:
Fx: Fetch from memory stages
Dx: Instruction Decode
RR: Register Read
Ex: Instruction execute stages
WB: Write Back to registers



Power of SMT

Power of multithreading with automotive code

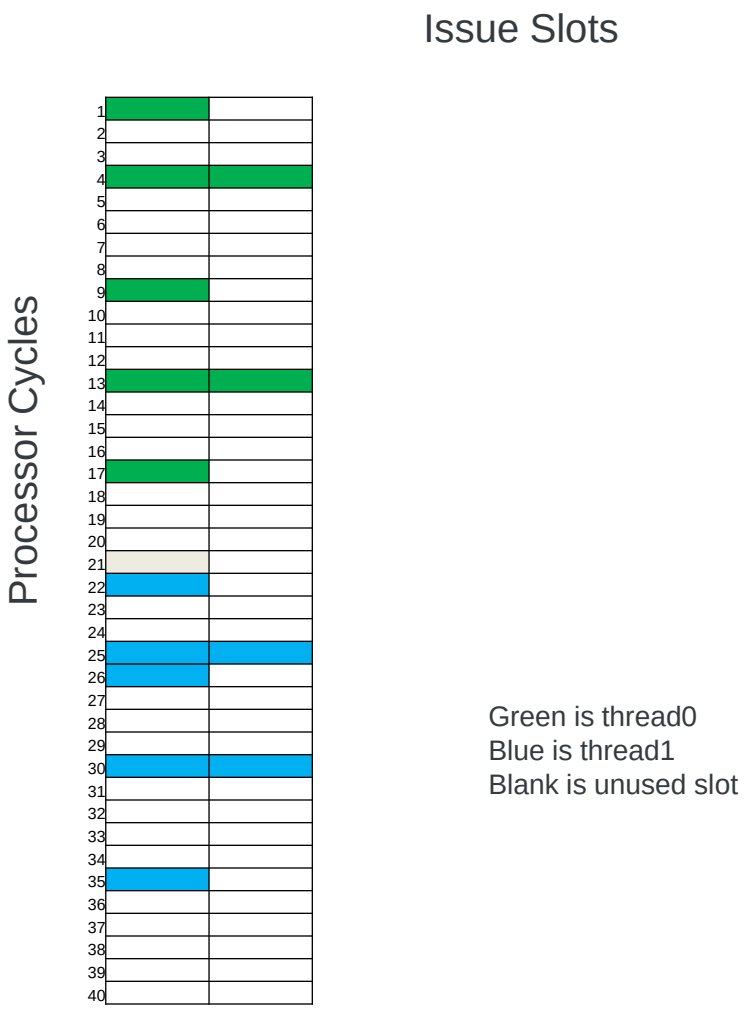
- Single threaded code profile
  - mostly zero issue stall cycles
  - Very few dual issue
  - Very few single issue



Green is thread0
Blue is thread1
Blank is unused slot

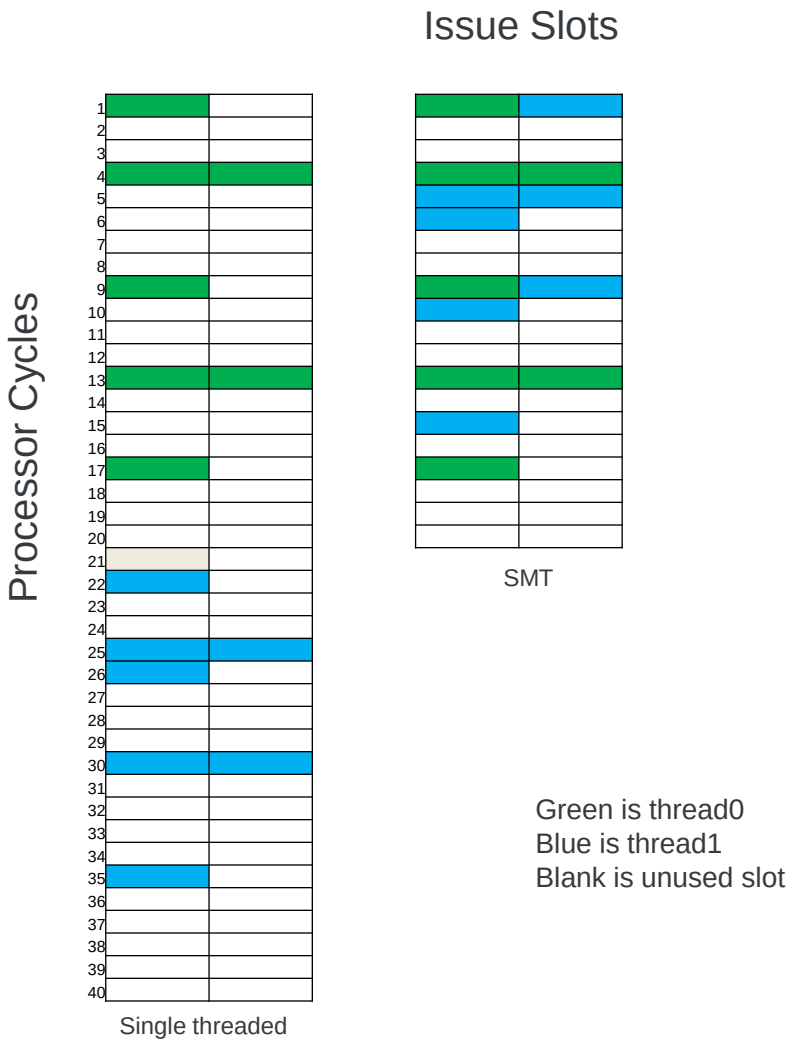
Power of multithreading with automotive code

- Single threaded code profile
  - mostly zero issue stall cycles
  - Very few dual issue
  - Very few single issue
- Each thread runs sequentially under utilizing CPU resources.



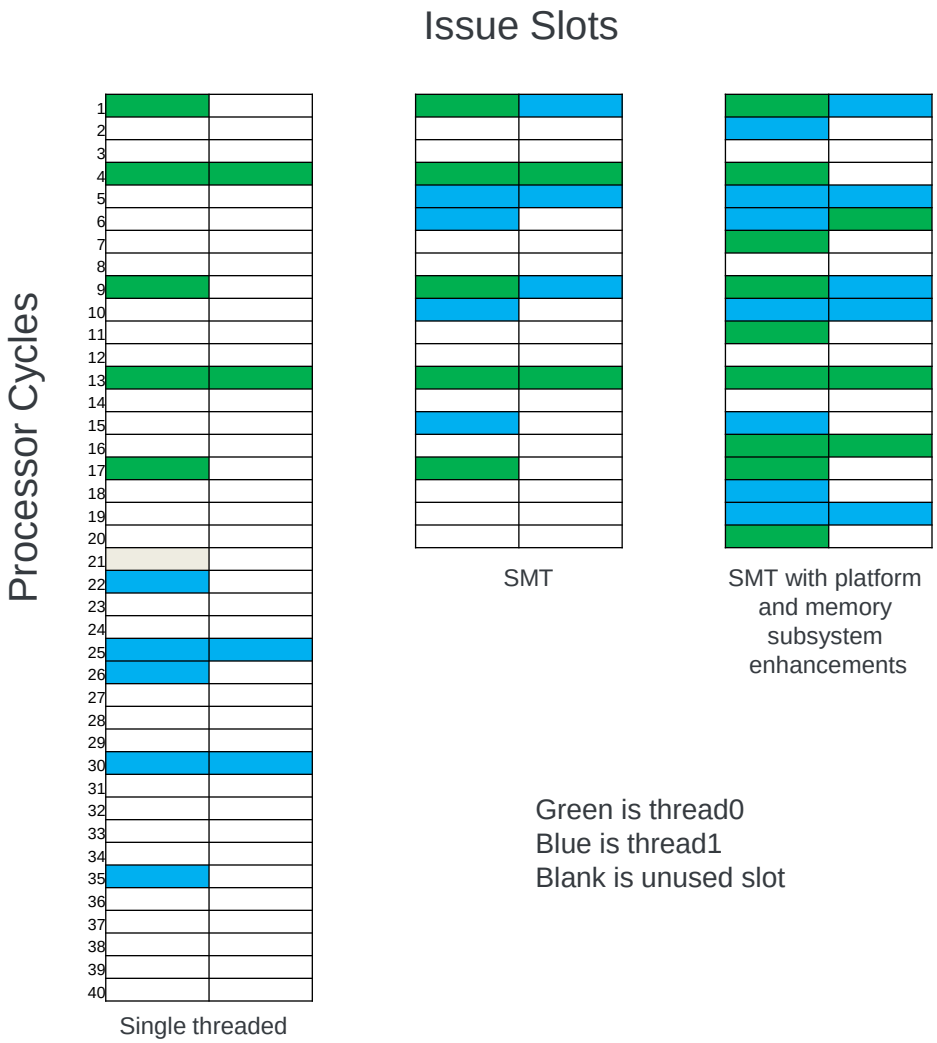
Power of multithreading with automotive code

- SMT code profile
  - allows code to run simultaneously drastically reducing over cycle count
  - Utilizes more of CPU resources
  - With no other changes to SoC there is a significant increase in overall performance



Power of multithreading with automotive code

- With enhancements in memory and platform subsystem can maximize the SMT benefits.
- Potential for enormous performance increases with minimal size and power increase.



Simultaneous Multithreading Summary

- **Automotive code is already multithreaded**
  Interrupts, multiple cores
- **SMT core maximizes performance/gate**
  Better utilization of CPU resources, non-blocking parallelism, optimized for automotive code profiles
- **Huge potential for further enhancements**
  Enhanced switch fabric, reduced memory latencies, advanced bus protocols

Q&A

www.Freescale.com