Synergistic Processor Unit Model

Wilmer Suarez **|** 109592501  
ESE 545 **|** SPRING 2019

**Table of Contents**

[**I.** **Introduction** 2](#_Toc7794108)

[**II.** **Design Process of Core Architecture** 2](#_Toc7794109)

[**III.** **SPU Core Block Diagram** 3](#_Toc7794110)

[**IV.** **SPU Top Module – Tests & Results** 4](#_Toc7794111)

[**Assembler Test** 4](#_Toc7794112)

[**Cache Miss Test** 4](#_Toc7794113)

[**All Instruction Test** 4](#_Toc7794114)

[**Local Store Fill Test** 4](#_Toc7794115)

[**Dual Instruction – Fetch\_Decode\_Issue\_Execute (No Hazard) Test** 4](#_Toc7794116)

[**Structural Hazard Resolution Test** 4](#_Toc7794117)

[**Data Hazard Resolution by Forwarding (No Stalling) Test** 4](#_Toc7794118)

[**Data Hazard Resolution by Stalling and Forwarding Test** 4](#_Toc7794119)

[**Control Hazard Resolution for Branches Test** 4](#_Toc7794120)

[**Appendix A – ISA** 5](#_Toc7794121)

[**Simple-Fixed 1 Instructions** 5](#_Toc7794122)

[**Simple-Fixed 2 Instructions** 6](#_Toc7794123)

[**Floating Point Instructions** 6](#_Toc7794124)

[**Floating Point Integer Instructions** 7](#_Toc7794125)

[**Byte Instructions** 7](#_Toc7794126)

[**Local Store Instructions** 8](#_Toc7794127)

[**Permute Instructions** 8](#_Toc7794128)

[**Branch Instructions** 9](#_Toc7794129)

[**Control Instructions** 9](#_Toc7794130)

[**Appendix B – Assembler Code** 10](#_Toc7794131)

[**Appendix C – Module Code** 10](#_Toc7794132)

[**Appendix D – Testbench Code** 10](#_Toc7794133)

[**References** 11](#_Toc7794134)

Synergistic Processor Unit Model

Wilmer Suarez **|** 109592501  
ESE 545 **|** SPRING 2019

***Abstract* — This report will describe the design process of a Cell SPU core written in VHDL. The Processor exploits Instruction and Data Level Parallelism. It is a dual-issue, SIMD, RISC, Multimedia Processor that consists of a total of 12 pipeline stages. The sub-set of instructions (ISA) used are presented in** [**Appendix A**](#_Appendix_A_–) **(68 instructions).**

# **Introduction**

The fully synchronous processing core consists of:

A Local SRAM Storage, an Instruction Line Buffer used to hold Instructions read from the Local Store. A 128-entry x128-bit Register File for both, floating-point and integer operations. Two pipes; an “Odd Pipe” with Permute, Local Store, and Branch execution units. An “Even Pipe” with Floating Point, Byte, and two Fixed-Point execution units. Lastly, Forwarding Circuits that forward the instruction results when RAW dependencies arise (Data Hazard).

The SPUs SIMD capability can perform operations on 16 x 8-bit integers, 8 x 16-bit integers, 4 x 32-bit integers, or 4 floating point numbers every cycle. The SPU can complete at most two instructions/cycle; one on each of the execution pipes [[1][2]](#_References). Having a shared Register File “allows the highest level of performance for various workloads with the smallest number of registers.” [[2]](#_References)

Multimedia-based information, and vector processing, is the target of the SPU; performing heavy computations, in parallel, on 128-bit wide data. This data (the instructions) are issued in pairs each cycle. Some taking 2 cycles to complete (simple-fixed instructions) while others up to 7 cycles (integer multiplication) [[2]](#_References).

* [SPU Core Block Diagram](#_SPU_Core_Block)

# **Design Process of Core Architecture**

***Register File***

The Register File is 128 registers x 128 bits. It supports 6 output ports (reads) and 2 inputs (writes). The output sends data directly to the corresponding execution unit latches. Results are only written back to the Register File during the Write Back stage.

The Register Files assures correct readings are made by checking if either of the write back addresses are the same as any of the six read addresses and bypassing the arriving write back data directly to the corresponding output.

***Even Pipe***

* *Simple Fixed 1 Unit*
  + Arithmetic, Logicals, Compares
* *Simple Fixed 2 Unit*
  + Word Shifts & Rotates
* *Single Precision Unit*
  + Single Precision Floating Point & Integer Multiplies
* *Byte Unit*
  + Byte Ops

***Odd Pipe***

* *Permute Unit*
  + Quadword Shifts & Rotates
* *Local Store Unit*
  + Load and Store
* *Branch Unit*
  + Branches

Each pipe produces results for one instruction every cycle. This gives the processor a CPI of ½. 2 Instructions executed each cycle.

All instructions pass through 7 stages prior to writing their result back to the Register File. If an instruction depends on any of the instructions currently in the pipe, the corresponding instruction can be forwarded, using the Forwarding Circuits, after its unit’s latency. The execution unit types are in [Appendix A](#_Appendix_C_–), along with the instructions executed in each unit.

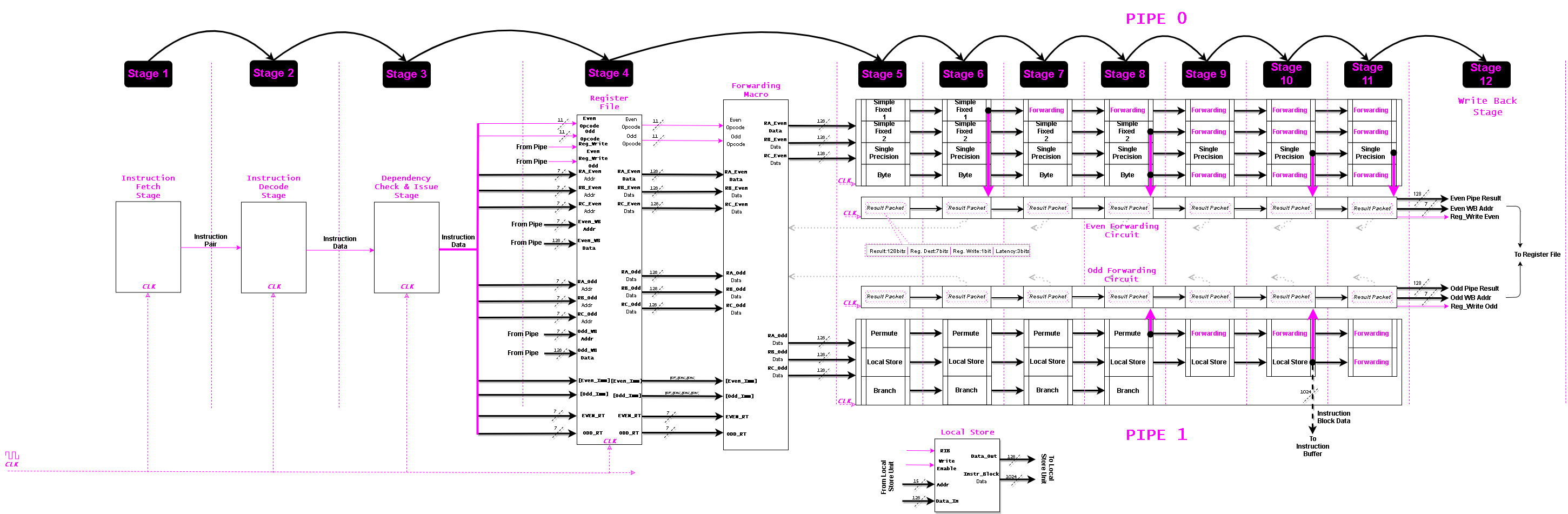
***Forwarding Circuits/Macro:***

If the instruction in the Register File is dependent on an instruction in either pipe, the Forwarding Circuits cycle through the pipes to retrieve the result and replace the dependent operand.

***Local Store***

The Local Store is a 32kB, single port SRAM. The first half is structured to store the current program code (starting at address 0). The second half – starting at address 16,384 – stores the program data. 16-Bytes of data can be read or written by Load & Store instructions, respectively. The Local Store also outputs 128-Byte instruction blocks (32 instructions) to fill the Instruction Line Buffer (ILB). The ILB signals the system to fetch the next instruction block after a miss when trying to fetch the next instruction pair.

# **SPU Core Block Diagram**



# **SPU Top Module – Tests & Results**

UPDATE THIS VVV

## **Assembler Test**

## **Cache Miss Test**

## **All Instruction Test**

## **Local Store Fill Test**

## **Dual Instruction – Fetch\_Decode\_Issue\_Execute (No Hazard) Test**

## **Structural Hazard Resolution Test**

## **Data Hazard Resolution by Forwarding (No Stalling) Test**

The forwarding was evaluated next using the following test cases:

1. When one operand has a dependency
2. When multiple operands have the same dependency
3. When multiple operands have different dependencies
4. Case 1-3 with both Even and Odd Instructions

## **Data Hazard Resolution by Stalling and Forwarding Test**

## **Control Hazard Resolution for Branches Test**

# **Appendix A – ISA**

## **Simple-Fixed 1 Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Add Word** | *a rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Add Word Immediate** | *ai rt, ra, value* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Subtract from Word** | *sf rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Subtract from Word Immediate** | *sfi rt, ra, value* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Count Leading Zeros** | *clz rt, ra* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **AND** | *and rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **AND with Complement** | *andc rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **AND Word Immediate** | *andi rt, ra, value* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **OR** | *or rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **OR with Complement** | *orc rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **OR Word Immediate** | *ori rt, ra, value* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **OR Across** | *orx rt, ra* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Exclusive OR** | *xor rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **NAND** | *nand rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **NOR** | *nor rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Compare Equal Word** | ceq rt, ra, rb | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Compare Equal Word Immediate** | *ceqi rt, ra, value* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Compare Greater Than Word** | *cgt rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Compare Greater Than Word Immediate** | *cgti rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Compare Logical Greater Than Word** | *clgt rt, ra, rb* | **Even** | **Simple Fixed 1** | **2** | **2** |
| **Compare Logical Greater Than Word Immediate** | *clgti rt, ra, value* | **Even** | **Simple Fixed 1** | **2** | **2** |

## **Simple-Fixed 2 Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Shift Left Word Immediate** | *shli rt, ra, value* | **Even** | **Simple Fixed 2** | **3** | **4** |
| **Rotate Halfword Immediate** | *rothi rt, ra, value* | **Even** | **Simple Fixed 2** | **3** | **4** |
| **Rotate Word Immediate** | *roti rt, ra, value* | **Even** | **Simple Fixed 2** | **3** | **4** |

## **Floating Point Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Floating Add** | *fa rt, ra, rb* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Subtract** | *fs rt, ra, rb* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Multiply** | *fm rt, ra, rb* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Multiply and Add** | *fma rt, ra, rb, rc* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Negative Multiply and Subtract** | *fnms rt, ra, rb, rc* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Multiply and Subtract** | *fms rt, ra, rb ,rc* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Compare Equal** | *fceq rt, ra, rb* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Compare Magnitude Equal** | *fcmeq rt, ra, rb* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Compare Greater Than** | *fcgt rt, ra, rb* | **Even** | **Single Precision FP** | **6** | **6** |
| **Floating Compare Magnitude Greater Than** | *fcmgt rt, ra, rb* | **Even** | **Single Precision FP** | **6** | **6** |

## **Floating Point Integer Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Multiply** | *mpy rt, ra, rb* | **Even** | **FP Integer** | **7** | **7** |
| **Multiply Unsigned** | *mpyu rt, ra, rb* | **Even** | **FP Integer** | **7** | **7** |
| **Multiply Immediate** | *mpyi rt, ra, value* | **Even** | **FP Integer** | **7** | **7** |
| **Multiply Unsigned Immediate** | *mpyui rt, ra, value* | **Even** | **FP Integer** | **7** | **7** |
| **Multiply and Add** | *mpya rt, ra, rb, rc* | **Even** | **FP Integer** | **7** | **7** |
| **Multiply High** | *mpyh rt, ra, rb* | **Even** | **FP Integer** | **7** | **7** |
| **Multiply High High Unsigned** | *mpyhhu rt, ra, rb* | **Even** | **FP Integer** | **7** | **7** |

## **Byte Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Count Ones in Bytes** | *cntb rt, ra* | **Even** | **Byte** | **3** | **4** |
| **Average Bytes** | *avgb rt, ra, rb* | **Even** | **Byte** | **3** | **4** |
| **Absolute Differences of Bytes** | *absdb rt, ra, rb* | **Even** | **Byte** | **3** | **4** |
| **Sum Bytes into Halfwords** | *sumb rt, ra, rb* | **Even** | **Byte** | **3** | **4** |

## **Local Store Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Load Quadword**  **(a–form)** | *lqa rt, symbol* | **Odd** | **LS** | **6** | **6** |
| **Load Quadword**  **(d-form)** | *lqd rt, symbol(ra)* | **Odd** | **LS** | **6** | **6** |
| **Store Quadword**  **(d-form)** | *stqd rt, symbol(ra)* | **Odd** | **LS** | **6** | **6** |
| **Store Quadword**  **(a-form)** | *stqa rt, symbol* | **Odd** | **LS** | **6** | **6** |
| **Read Instruction Block** | *rib* | **Odd** | **LS** | **6** | **6** |
| **Immediate Load Half Word Upper** | *ilhu rt, symbol* | **Odd** | **LS** | **6** | **6** |
| **Immediate Load Word** | *il rt, symbol* | **Odd** | **LS** | **6** | **6** |
| **Immediate load Address** | *ila rt, symbol* | **Odd** | **LS** | **6** | **6** |
| **Immediate OR Halfword Lower** | *iohl rt, symbol* | **Odd** | **LS** | **6** | **6** |

## **Permute Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Shift Left Quadword by Bits Immediate** | *shlqbii rt, ra, value* | **Odd** | **Permute** | **3** | **4** |
| **Shift Left Quadword by Bytes Immediate** | *shlqbyi rt, ra, value* | **Odd** | **Permute** | **3** | **4** |
| **Rotate Quadword by Bytes Immediate** | *rotqbyi rt, ra, value* | **Odd** | **Permute** | **3** | **4** |
| **Rotate Quadword by Bits Immediate** | *rotqbii rt, ra, value* | **Odd** | **Permute** | **3** | **4** |

## **Branch Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Branch Relative** | *br symbol* | **Odd** | **Branch** | **3** | **4** |
| **Branch Absolute** | *bra symbol* | **Odd** | **Branch** | **3** | **4** |
| **Branch Indirect** | *bi ra* | **Odd** | **Branch** | **3** | **4** |
| **Branch If Not Zero Word** | *brnz rt, symbol* | **Odd** | **Branch** | **3** | **4** |
| **Branch If Zero Word** | *brz rt, symbol* | **Odd** | **Branch** | **3** | **4** |
| **Branch Indirect If Zero** | *biz rt, ra* | **Odd** | **Branch** | **3** | **4** |
| **Branch Indirect If Not Zero** | *binz rt, ra* | **Odd** | **Branch** | **3** | **4** |

## **Control Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction Name** | **Assembly Code** | **Lane**  **(Execution Pipe)** | **Unit** | **Number of Stages (Pipeline Depth)** | **Instruction Latency** |
| **Stop and Signal** | *stop* | **-** | **-** | **-** | **-** |
| **Nop (Load)** | *lnop* | **Odd** | **-** | **-** | **-** |
| **Nop (Execute)** | *nop* | **Even** | **-** | **-** | **-** |

**Total Instructions: 68**

# **Appendix B – Assembler Code**

FORMAT THIS ^^^

# **Appendix C – Module Code**

UPDATE THIS VVV

***SPU Core Top Module***

***Register File***

***Local Store***

***Forwarding Macro/Circuits***

***Even & Odd Pipeline***

***Component Library***

***ISA Library***

# **Appendix D – Testbench Code**

UPDATE THIS VVV

***SPU Core Top Module***

# **References**

1. *Thomas Chen, Ram Raghavan, Jason Dale, Eiji Iwata. “Cell Broadband Engine Architecture and its First Implementation,” IBM. November 29, 2005. [Online]* [*https://www.ibm.com/developerworks/library/pacellperf/index.html*](https://www.ibm.com/developerworks/library/pacellperf/index.html)
2. *Flachs et al. (2006) “The Microarchitecture of the Synergistic Processor for a Cell Processor, “IEEE Journal of Solid-State Circuits (Volume: 41, Issue: 1, January 2006) [Online]*

<https://ieeexplore.ieee.org/document/1564346/metrics#metrics>