

Extending Compiler Support for the BrainScaleS Plasticity Processor

Bachelor's Thesis Presentation

Arthur Heimbrecht

2017-03-16

Contents

PPU Architecture

Compiler Structure

Extending GCC for the PPU

Results

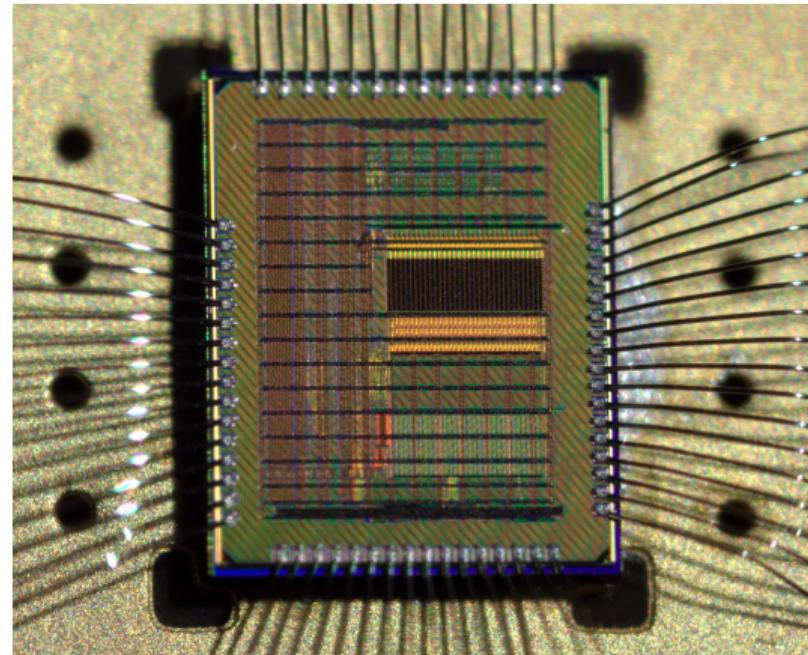


Figure: Photograph of HICANN-DLS v1 chip, Friedmann et al.

HICANN-DLS

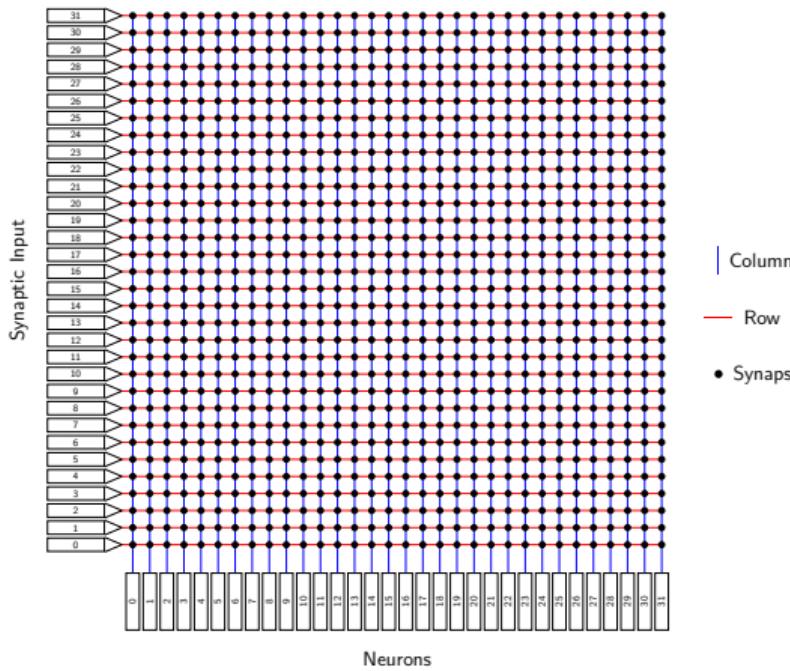


Figure: Schematic representation of the synapse array on HICANN-DLS v2 and newer

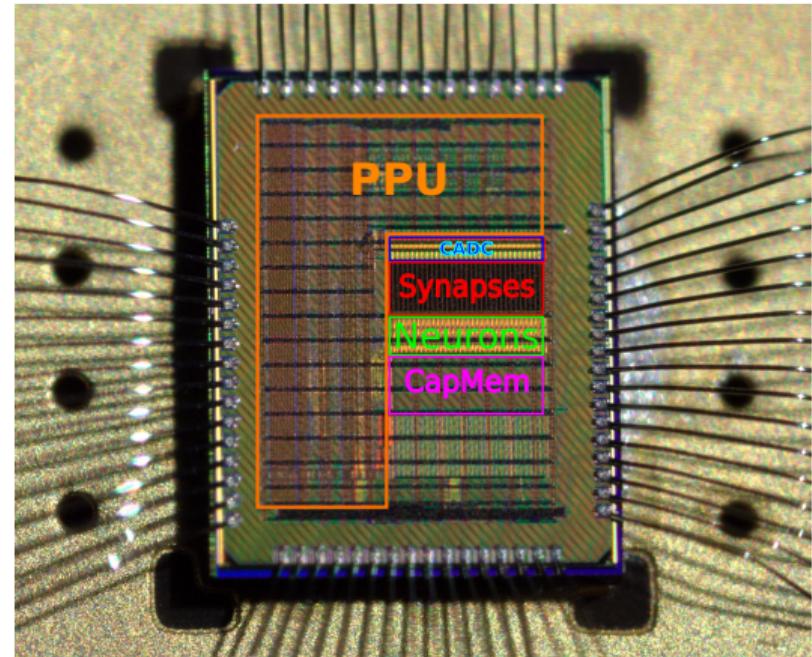


Figure: Photograph of the HICANN-DLS v1 chip, modified from Friedmann et al.

Processors

- ▶ “common” von-Neumann architecture
- ▶ machine instructions in memory
- ▶ data in memory

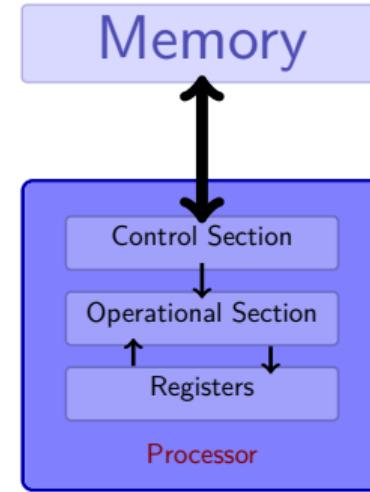


Figure: Schematic of general purpose processor with von-Neumann architecture

Processors

- ▶ “common” von-Neumann architecture
- ▶ machine instructions in memory
- ▶ data in memory
- ▶ latency(register) \approx 1 cycle
- ▶ latency(memory) \approx 100-1000 cycles

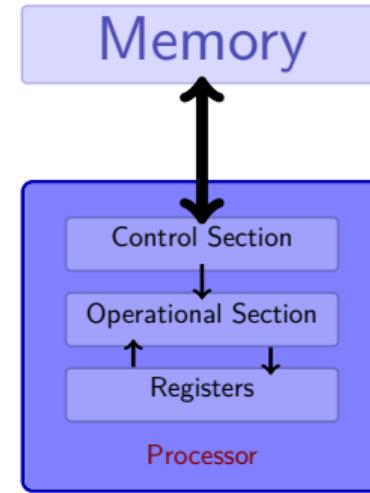


Figure: Schematic of general purpose processor with von-Neumann architecture

Plasticity Processing Unit

- ▶ based on PowerISA
- ▶ extension with vector registers
 - parallelization of operations
 - better performance
- ▶ weight updates
- ▶ access to synaptic array

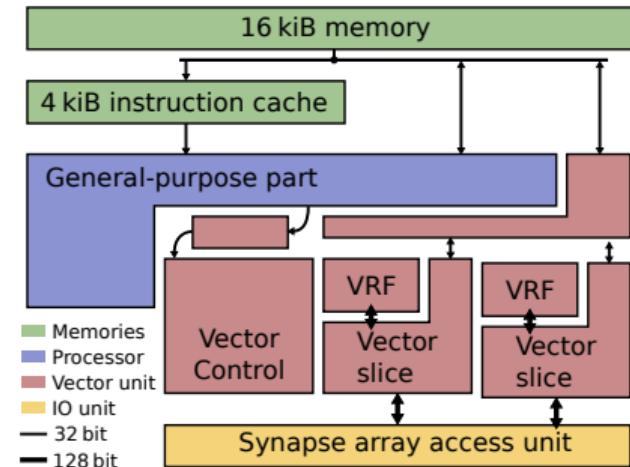


Figure: Structure of the PPU's architecture

Compiling Stages

- ▶ code $\xrightarrow{\text{compile}}$ executable
- ▶ different compiling stages (e.g. Preprocessor)
- ▶ front-end \leftrightarrow programming language
- ▶ middle-end \rightarrow optimization
- ▶ back-end \leftrightarrow processor architecture

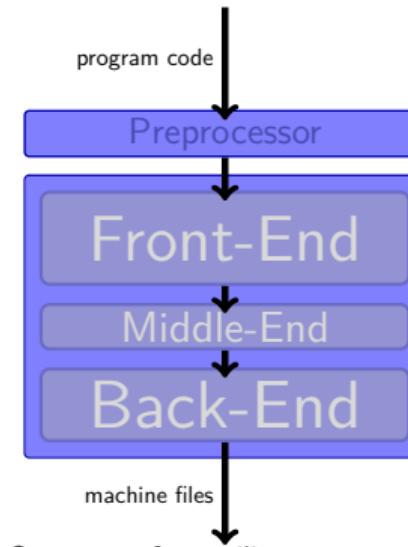


Figure: Structure of compiling process and compiler

GCC and the PPU

- ▶ no “official” support for PPU
- ▶ currently using macros

Example for macro usage

```
1 #define VR_TMP      0
2 #define VR_A       1
3 ...
4 fxv_addbm(VR_A, VR_A, VR_TMP);
```

GCC and the PPU

- ▶ no “official” support for PPU
- ▶ currently using macros

Example for macro usage

```
1 #define VR_TMP      0
2 #define VR_A        1
3 ...
4 fxv_adbm(VR_A, VR_A, VR_TMP);
```

Preprocessed example

```
1 ...
2 fxv_adbm(1, 1, 0);
```

GCC and the PPU

- ▶ no “official” support for PPU
- ▶ currently using macros
- ▶ existing AltiVec extension

Example for macro usage

```
1 #define VR_TMP      0
2 #define VR_A       1
3 ...
4 fxv_addbm(VR_A, VR_A, VR_TMP);
```

Preprocessed example

```
1 ...
2 fxv_addbm(1, 1, 0);
```

Example for AltiVec code

```
1 vector char a, tmp;
2 ...
3 a = vec_add(a, tmp);
```

GCC and the PPU

- ▶ no “official” support for PPU
- ▶ currently using macros
- ▶ existing AltiVec extension
- ▶ intrinsics → **use this**

Example for macro usage

```
1 #define VR_TMP      0
2 #define VR_A        1
3 ...
4 fxv_addbm(VR_A, VR_A, VR_TMP);
```

Preprocessed example

```
1 ...
2 fxv_addbm(1, 1, 0);
```

Example for AltiVec code

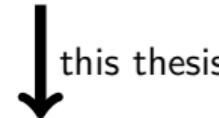
```
1 vector char a, tmp;
2 ...
3 a = vec_add(a, tmp);
```

Main Work of this Thesis

- ▶ add GCC support for PPU

Previous PPU code

```
1 #define VR_TMP      0
2 #define VR_A        1
3 ...
4 fxv_adbbm(VR_A, VR_A, VR_TMP);
```



this thesis

New PPU code

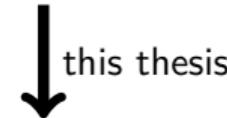
```
1 vector char a, tmp;
2 ...
3 a = fxv_add(a, tmp);
```

Main Work of this Thesis

- ▶ add GCC support for PPU
- ▶ GCC back-ends not documented

Previous PPU code

```
1 #define VR_TMP      0
2 #define VR_A        1
3 ...
4 fxv_adbbm(VR_A, VR_A, VR_TMP);
```



New PPU code

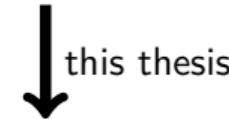
```
1 vector char a, tmp;
2 ...
3 a = fxv_add(a, tmp);
```

Main Work of this Thesis

- ▶ add GCC support for PPU
- ▶ GCC back-ends not documented
- ▶ AltiVec inspiration for
 - internal functions
 - internal macros

Previous PPU code

```
1 #define VR_TMP      0
2 #define VR_A        1
3 ...
4 fxv_adbbm(VR_A, VR_A, VR_TMP);
```



this thesis

New PPU code

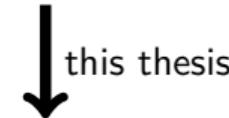
```
1 vector char a, tmp;
2 ...
3 a = fxv_add(a, tmp);
```

Main Work of this Thesis

- ▶ add GCC support for PPU
- ▶ GCC back-ends not documented
- ▶ AltiVec inspiration for
 - internal functions
 - internal macros
- ▶ other components:
 - vector registers
 - vector variables
 - new intrinsics
 - ...

Previous PPU code

```
1 #define VR_TMP      0
2 #define VR_A        1
3 ...
4 fxv_adbbm(VR_A, VR_A, VR_TMP);
```



this thesis

New PPU code

```
1 vector char a, tmp;
2 ...
3 a = fxv_add(a, tmp);
```

New Features

- ▶ -mcpu=nux target flag & s2pp.h header

Compile command

```
1 powerpc-linux-eabi-gcc -mcpu=nux
```

Example file

```
1 #include<s2pp.h>
2
3 start(){
4     return;
5 }
```

New Features

- ▶ -mcpu=nux target flag & s2pp.h header
- ▶ `vector` attribute for variables

Compile command

```
1 powerpc-linux-eabi-gcc -mcpu=nux
```

Example file

```
1 #include<s2pp.h>
2
3 start(){
4     vector char vec;
5     return;
6 }
```

New Features

- ▶ `-mcpu=nux` target flag & `s2pp.h` header
- ▶ `vector` attribute for variables
- ▶ simple vector initialization

Compile command

```
1 powerpc-linux-eabi-gcc -mcpu=nux
```

Example file

```
1 #include<s2pp.h>
2
3 start(){
4     vector<char> vec =
5         {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
6     return;
7 }
```

New Features

- ▶ -mcpu=nux target flag & s2pp.h header
- ▶ `vector` attribute for variables
- ▶ simple vector initialization
- ▶ PPU vector intrinsics

Compile command

```
1 powerpc-linux-eabi-gcc -mcpu=nux
```

Example file

```
1 #include<s2pp.h>
2
3 start(){
4     vector char vec =
5         {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
6     vec = fxx_add(vec, vec);
7     return;
8 }
```

New Features

- ▶ `-mcpu=nux` target flag & `s2pp.h` header
- ▶ `vector` attribute for variables
- ▶ simple vector initialization
- ▶ PPU vector intrinsics
- ▶ inline assembly support

Compile command

```
1 powerpc-linux-eabi-gcc -mcpu=nux
```

Example file

```
1 #include<s2pp.h>
2
3 start(){
4     vector char vec =
5         {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
6     vec = fvx_add(vec, vec);
7     asm ("fxvaddbm %0, %0, %0"
8          :"+kv" (vec)::);
9     return;
10 }
```

New Features

- ▶ `-mcpu=nux` target flag & `s2pp.h` header
- ▶ `vector` attribute for variables
- ▶ simple vector initialization
- ▶ PPU vector intrinsics
- ▶ inline assembly support
- ▶ optimization

Compile command

```
1 powerpc-linux-eabi-gcc -mcpu=nux -O1
```

Example file

```
1 #include<s2pp.h>
2
3 start(){
4     vector char vec =
5         {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
6     vec = fvx_add(vec, vec);
7     asm ("fxvaddbm %0, %0, %0"
8          :"+kv" (vec)::);
9     return;
10 }
```

Advantages

Comparison of unoptimized code (left) and **-O1** optimized code (right)

```

1   start:
2   stwu %r1,-64(%r1)
3   stw %r31,60(%r1)
4   mr %r31,%r1
5   li %r9,1
6   fxv splatb %f12,%r9      1   start:
7   li %r9,16                  2   stwu %r1,-48(%r1)
8   fxv stax %f12,%r31,%r9    3   li %r9,1
9   li %r9,16                  4   fxv splatb %f12,%r9
10  fxv lax %f12,%r31,%r9    5   li %r9,16
11  li %r9,16                  6   fxv stax %f12,%r1,%r9
12  fxv lax %f11,%r31,%r9    7   fxv lax %f12,%r1,%r9
13  fxv addbfs %f12,%f12,%f11 8   fxv lax %f11,%r1,%r9
14  li %r9,16                  9   fxv addbfs %f12,%f12,%f11
15  fxv stax %f12,%r31,%r9    10  fxv stax %f12,%r1,%r9
16  #APP                      11  #APP
17  # 277 "nux/main.c" 1       12  # 277 "nux/main.c" 1
18  fxv addbm %f12, %f12, %f12 13  fxv addbm %f12, %f12, %f12
19  # 0 "" 2                   14  # 0 "" 2
20  #NO_APP                   15  #NO_APP
21  li %r9,16                  16  fxv stax %f12,%r1,%r9
22  fxv stax %f12,%r31,%r9    17  addi %r1,%r1,48
23  nop                       18  blr
24  addi %r11,%r31,64
25  lwz %r31,-4(%r11)
26  mr %r1,%r11
27  blr

```

Advantages

- ▶ easier PPU programs for users
- ▶ efficient programs
- ▶ workarounds in compiler

Comparison of unoptimized code (left) and **-O1** optimized code (right)

```
1  start:  
2  stwu %r1,-64(%r1)  
3  stw %r31,60(%r1)  
4  mr %r31,%r1  
5  li %r9,1  
6  fxv splatb %f12,%r9      1  start:  
7  li %r9,16                 2  stwu %r1,-48(%r1)  
8  fxv stax %f12,%r31,%r9    3  li %r9,1  
9  li %r9,16                 4  fxv splatb %f12,%r9  
10 fxv lax %f12,%r31,%r9     5  li %r9,16  
11 li %r9,16                 6  fxv stax %f12,%r1,%r9  
12 fxv lax %f11,%r31,%r9     7  fxv lax %f12,%r1,%r9  
13 fxv addbfs %f12,%f12,%f11 8  fxv lax %f11,%r1,%r9  
14 li %r9,16                 9  fxv addbfs %f12,%f12,%f11  
15 fxv stax %f12,%r31,%r9    10 fxv stax %f12,%r1,%r9  
16 #APP                      11 #APP  
17 # 277 "nux/main.c" 1       12 # 277 "nux/main.c" 1  
18 fxv addbbm %f12, %f12, %f12 13 fxv addbbm %f12, %f12, %f12  
19 # 0 "" 2                   14 # 0 "" 2  
20 #NO_APP                    15 #NO_APP  
21 li %r9,16                  16 fxv stax %f12,%r1,%r9  
22 fxv stax %f12,%r31,%r9    17 addi %r1,%r1,48  
23 nop                         18 blr
```

Outlook

- ▶ further testing for compiler bugs
- ▶ already existing applications
- ▶ extending PPU test coverage
- ▶ C-library support for PPU
→ soft-float
- ▶ debugging support?

Compiling Directive

```
1 powerpc-linux-eabi-gcc -mcpu=nux -O1
```

Example File

```
1 #include<s2pp.h>
2
3 start(){
4     vector<char> vec =
5         {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
6     vec = fxv_add(vec, vec);
7     asm ("fxvaddbm %0, %0, %0"
8          : "+kv" (vec)::);
9     return;
10 }
```

Outlook

- ▶ further testing for compiler bugs
- ▶ already existing applications
- ▶ extending PPU test coverage
- ▶ C-library support for PPU
→ soft-float
- ▶ debugging support?
- ▶ **new tools for PPU experiments!**

Compiling Directive

```
1 powerpc-linux-eabi-gcc -mcpu=nux -O1
```

Example File

```
1 #include<s2pp.h>
2
3 start(){
4     vector<char> vec =
5         {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
6     vec = fxv_add(vec, vec);
7     asm ("fxvaddbm %0, %0, %0"
8          :"+kv" (vec)::);
9     return;
10 }
```

Outlook

- ▶ further testing for compiler bugs
- ▶ already existing applications
- ▶ extending PPU test coverage
- ▶ C-library support for PPU
→ soft-float
- ▶ debugging support?
- ▶ **new tools for PPU experiments!**
- ▶ **more experiments!**

Compiling Directive

```
1 powerpc-linux-eabi-gcc -mcpu=nux -O1
```

Example File

```
1 #include<s2pp.h>
2
3 start(){
4     vector<char> vec =
5         {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
6     vec = fxv_add(vec, vec);
7     asm ("fxvaddbm %0, %0, %0"
8          :"+kv" (vec)::);
9     return;
10 }
```

References

- S. Friedmann, J. Schemmel, A. Grübl, A. Hartel, M. Hock, and K. Meier, "Demonstrating hybrid learning in a flexible neuromorphic hardware system", (2016).
- T. Flik, *Mikroprozessortechnik und rechnerstrukturen*, ger, 7., neu bearb. Aufl. (Springer, Berlin ; Heidelberg [u.a.], 2005), XIV, 649 S.
- K. D. Cooper and L. Torczon, *Engineering a compiler*, eng, 2. ed., Previous ed.: 2004 (Elsevier, Amsterdam ; Heidelberg [u.a.], 2012), XXIII, 800 S.
- A. V. Aho, *Compiler, Prinzipien, techniken und werkzeuge*, ger, 2., aktualisierte Aufl., it Informatik, Index S. 1227-1253 (Pearson Studium, München [u.a.], 2008), XXXVI, 1253 S.
- S. Friedmann, *Nux manual*, (2016).
- *Gnu compiler collection internals manual*,
<https://gcc.gnu.org/onlinedocs/gccint/index.html>, Free Software Foundation Inc. (2017).

Appendix

Additional Figures

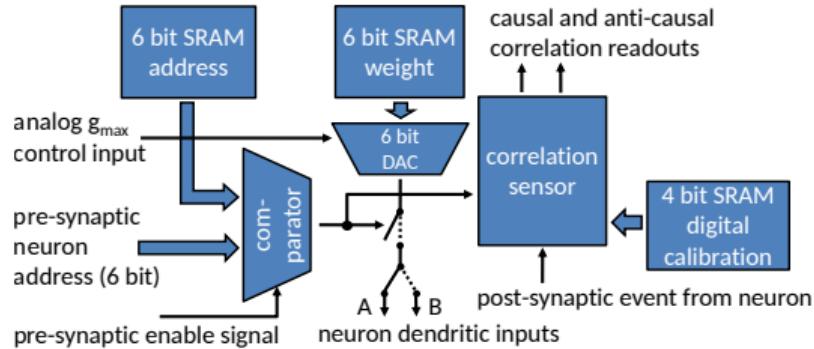


Figure: Block Diagram of the Synapse Circuit (modified from Friedmann et al.).

31	23	15	7	0
mnemonic	operand	operand	operand	
addi	r1 register address	r2 register address	5 immediate operand	

Figure: Representation of Assembly Instruction addi as a Machine Instruction in Memory. The immediate value 5 is added to register r2 and the result written in r1.

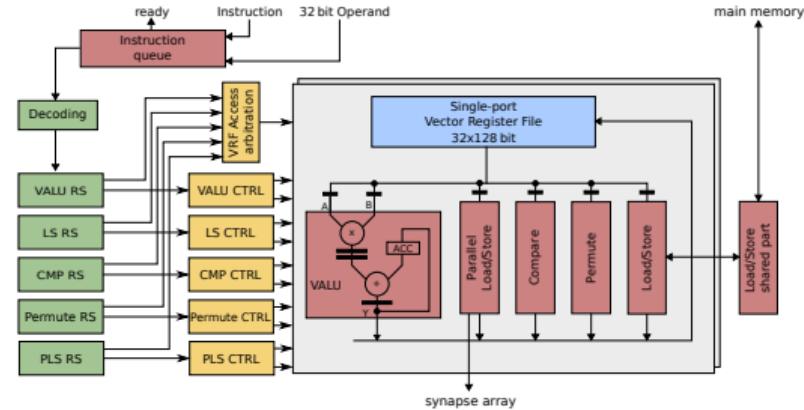


Figure: Detailed Structure of the s2pp Vector Extension (taken from Friedmann et al.)

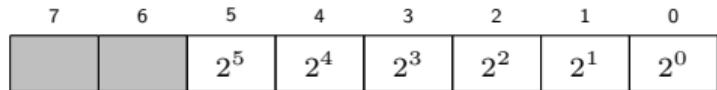


Figure: Comparison of the Representation of Weights in Synapses and the Fractional Representation of Vector Components for Fixed-Point Saturational Arithmetic

Additional Figures

		0	7	15									31									63									127
QI		Quarter Integer																													
HI		Half Integer																													
SI														Single Integer																	
SF														Single Float																	

Figure: Vector structures are 128 bits wide and split into common word sizes.

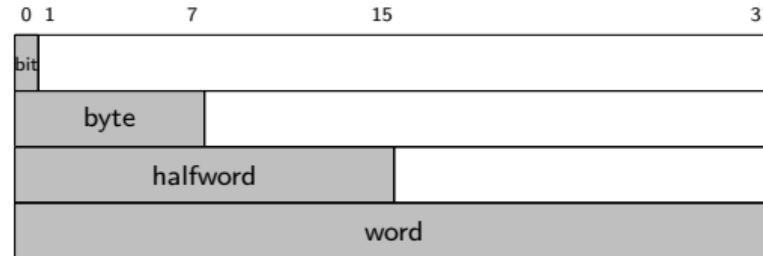


Figure: Illustration of Word Sizes for 32-bit Words