NeuroMatrix(r) SDK

Debugging target library `nm6403emu.dll`. Emulator Target.

User's Guide.

RC Module [http://www.module.ru]

NeuroMatrix(r) SDK: Debugging target library `nm6403emu.dll`. Emulator Target.

RC Module [http://www.module.ru]

Table of Contents

Preface

This reference describes the library of debug targets `nm6403emu.dll` version 1.2, English variant, for Multitarget JIT Debugger 1.2 included in NeuroMatrix® SDK 2.00.

Chapter 1. Purpose and contents

`nm6403.dll` library contains software in-process emulator of the NeuroMatrix® NM6403 processor to be used as a debugging target for Debugger. This library can be seen from the Debugger dialogs as:

“`NM6403 NeuroMatrix® Software Emulators v1.00`”

The library contents:

1. “`Instruction level NM6403 emulator`” debug target.

The strings listed in quotes are used in the Debugger dialogs as names of targets.

Chapter 2. Debug target: Instruction level NM6403 emulator

This debug target realizes the universal processor interface over the instruction level emulator.

In the Debugger target specific window this target displays the Vector Unit state: registers and matrixes.

1. Instruction level software emulator

The Instruction level software emulator simulates execution of the NM6403 instruction set, though it does not model time characteristics of the execution (so called “fast” emulator). The emulator runs instructions strictly sequentially (one by one), so for example an instruction followed by a vector instruction would start only after all operations of the previous instruction is completed (which makes a difference to a real processor behaviour in case of parallel mode execution).

Nevertheless the results of execution of a correct program on “fast” simulator are the same as those acquired by execution on a hardware processor.

1.1. Memory map

The emulator has four equal sized memory banks. Memory banks have the following start addresses:

- `00000000` – 1st local memory bank,
- `40000000` – 2nd local memory bank,
- `80000000` – 1st global memory bank,
- `c0000000` – 2nd global memory bank.

Size of memory banks is specified by startup parameter.

1.2. Performance

Performance (rate of execution) of the emulator obviously depends on the host system performance and the current load of the operating system.

The emulator runs up to 3000000 scalar instructions per second on Pentium II 300MHz CPU. The rate of execution of vector instructions is about 10-100 instructions per second.

2. Startup parameters

The Startup parameters dialog for a target can be opened from the Choose debug target dialog box (*Debugger menu command*: Target/Connect to...) by selecting that target from the list of available targets and clicking on Settings button.

Startup parameters are used to initialize a debug target during the connection phase (where the simulator connection actually means instance creation).

This debug target has the following startup parameters:

2.1. Size of memory banks

The size is specified in kilobytes. The initial value – **is 128 KB**. This parameter sets the size of a simulated memory bank (equal for all four memory banks).

3. Connecting to the Debugger

A new emulator instance is created every time when the debugger is connected to this debug target. The Debugger allocates memory blocks for simulated global and local memory banks.

The target initialization time depends indirectly on the allocated memory size, it significantly increases if the system needs work with a swap file.

4. Disconnecting from the Debugger

When the debugger disconnects from this target, the emulator instance is destroyed. All allocated memory blocks will be freed.

No matter how you answered the question about the program on the target. the program will be destroyed with the emulator instance.

5. Debugging actions features

Target reset means that the debugger destroys the current emulator instance and creates a new one.

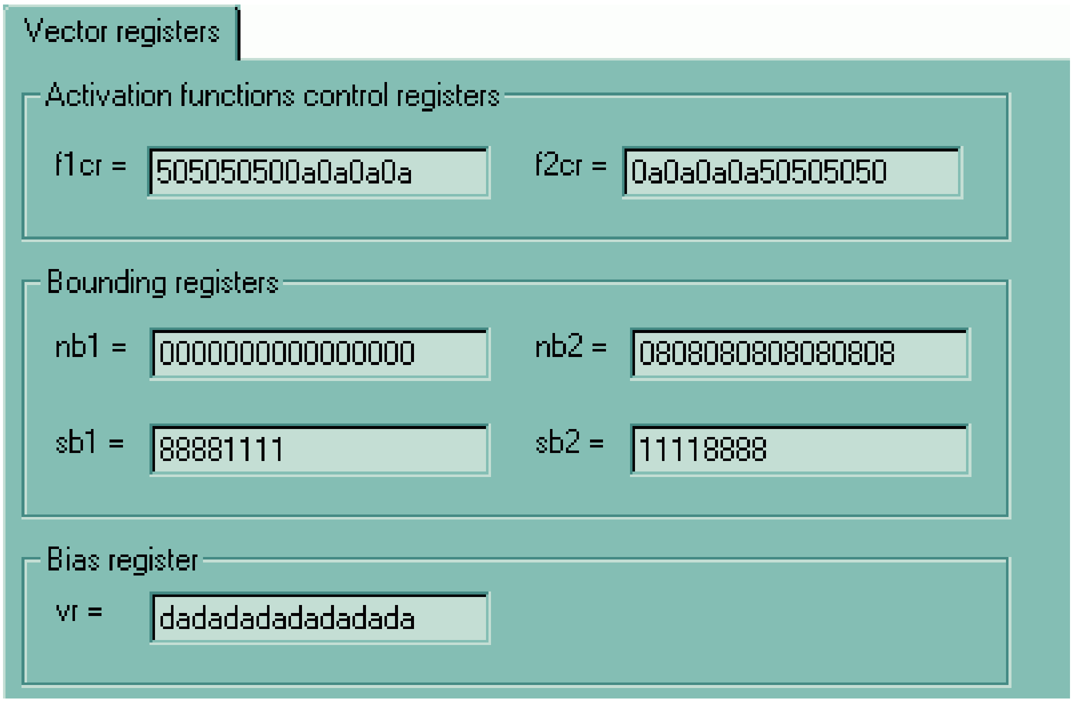
6. Target specifics window

Use *the Debugger menu* View/Target specifics to open target specifics window.

Using target specifics window of this debugging target you can control the state of the Vector Unit of a simulated processor. You can trace the contents of vector registers, matrixes and internal memory blocks.

You can find more information about the Vector Unit of the processor in the NeuroMatrix® NM6403 Assembly Language Overview in the NeuroMatrix® SDK (`doc/AsmOver(Eng).pdf`).

6.1. Vector registers property page



f1cr = 505050500a0a0a0a

f2cr = 0a0a0a0a50505050

nb1 = 0000000000000000

nb2 = 0808080808080808

sb1 = 88881111

sb2 = 11118888

vr = dadadadadadadada

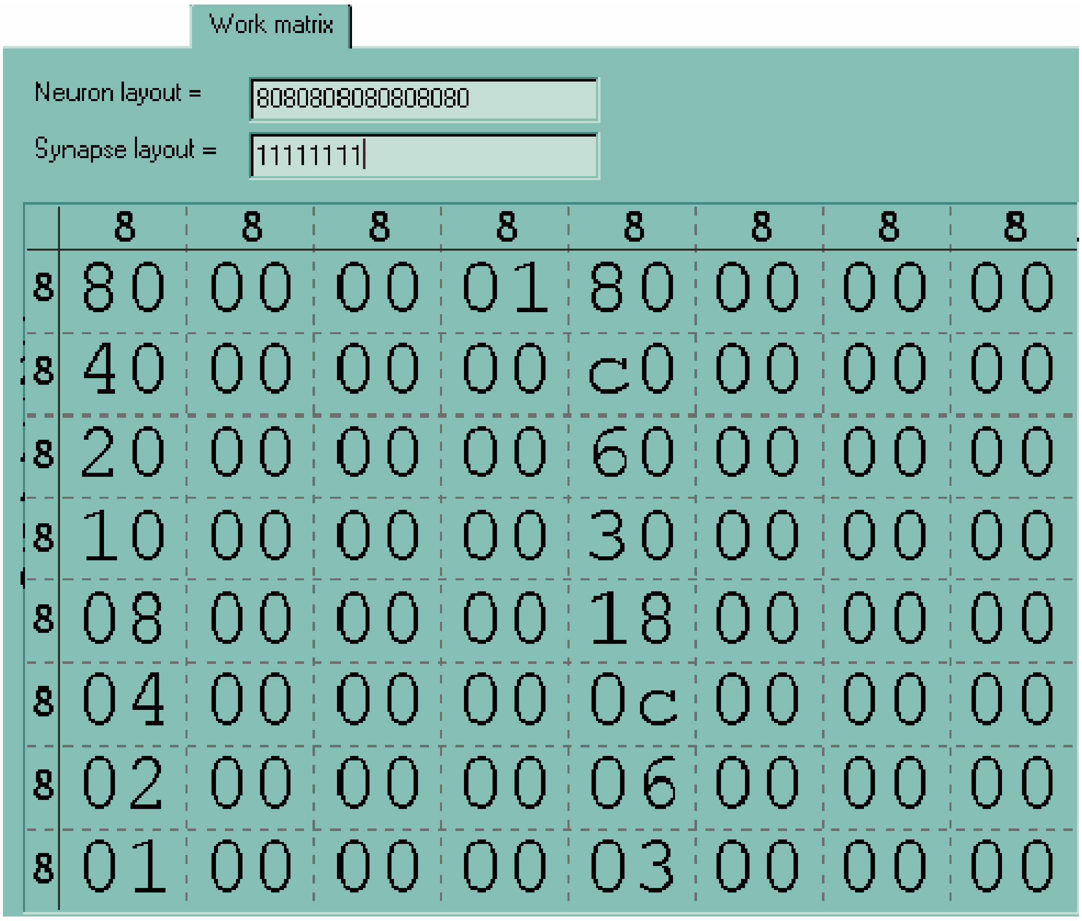
Use this property page to control the state of Vector Unit control registers. These registers are not directly readable on a hardware processor.

These registers include:

- `f1cr` – register is used to control configuration of `X` operand when applying the activation function,
- `f2cr` – register is used to control configuration of `Y` operand when applying the activation function,
- `nb1` – register controls Shadow Matrix split into columns,
- `nb2` – register controls Active Matrix split into columns. It also specifies split of packed data words when the data are processed on the Vector ALU. This register is not directly accessible on a hardware processor. `nb1` content is copied to this register during `wtw` operation,
- `sb1` – register controls Shadow Matrix split into rows. This register consists of odd bits of `sb`,
- `sb2` – register controls Active Matrix split into rows. This register consists of even bits of `sb`. This register is not directly accessible on a hardware processor. `sb1` content is copied to this register during `wtw` operation,
- `vr` – bias register.

You can modify registers contents.

6.2. Matrixes pages



Neuron layout = 8080808080808080

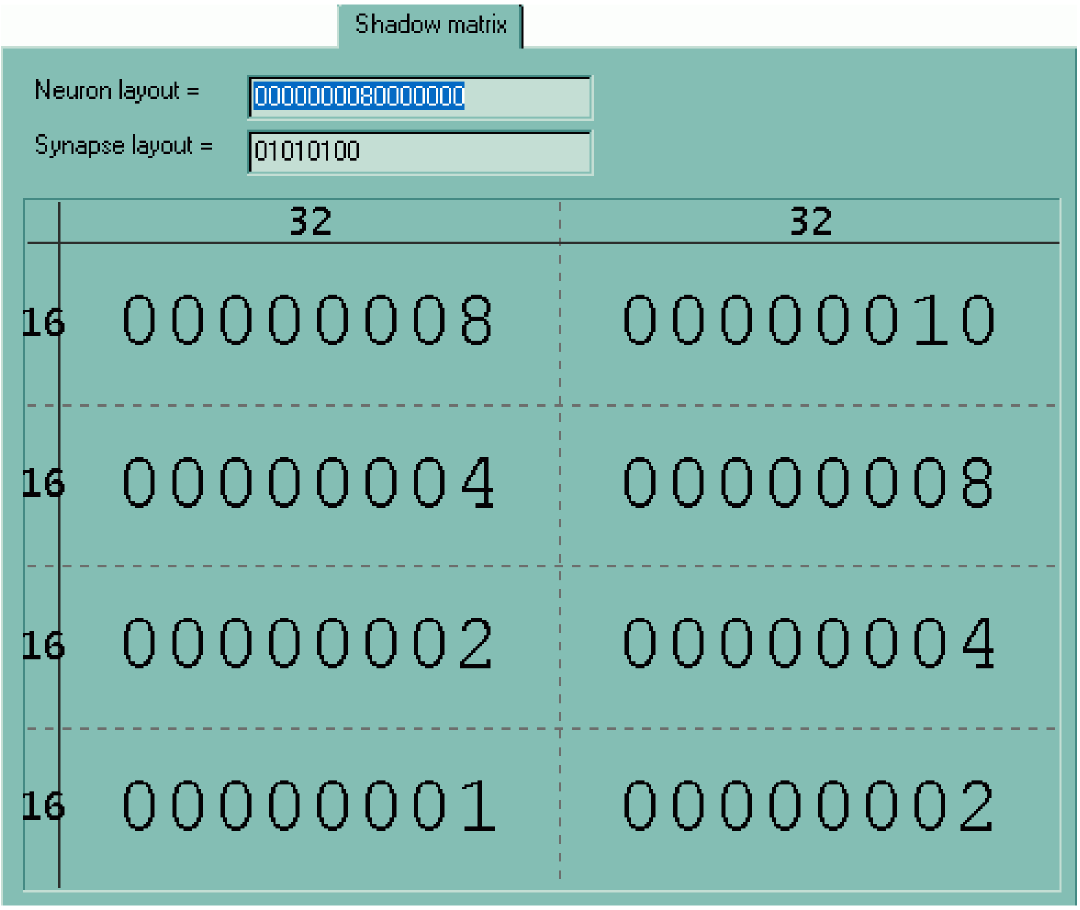
Synapse layout = 11111111

	8	8	8	8	8	8	8	8
8	80	00	00	01	80	00	00	00
8	40	00	00	00	c0	00	00	00
8	20	00	00	00	60	00	00	00
8	10	00	00	00	30	00	00	00
8	08	00	00	00	18	00	00	00
8	04	00	00	00	0c	00	00	00
8	02	00	00	00	06	00	00	00
8	01	00	00	00	03	00	00	00

These property pages display the contents of the Vector Unit matrixes (Shadow and Active) and current splits of the matrixes. The upper row and the left column (separated by full lines) display numbers of bits in the column and in the row.

The values are represented in hexadecimal or decimal formats. Use context menu to specify the displayed format of values.

You can change split of the matrixes using configuration values:



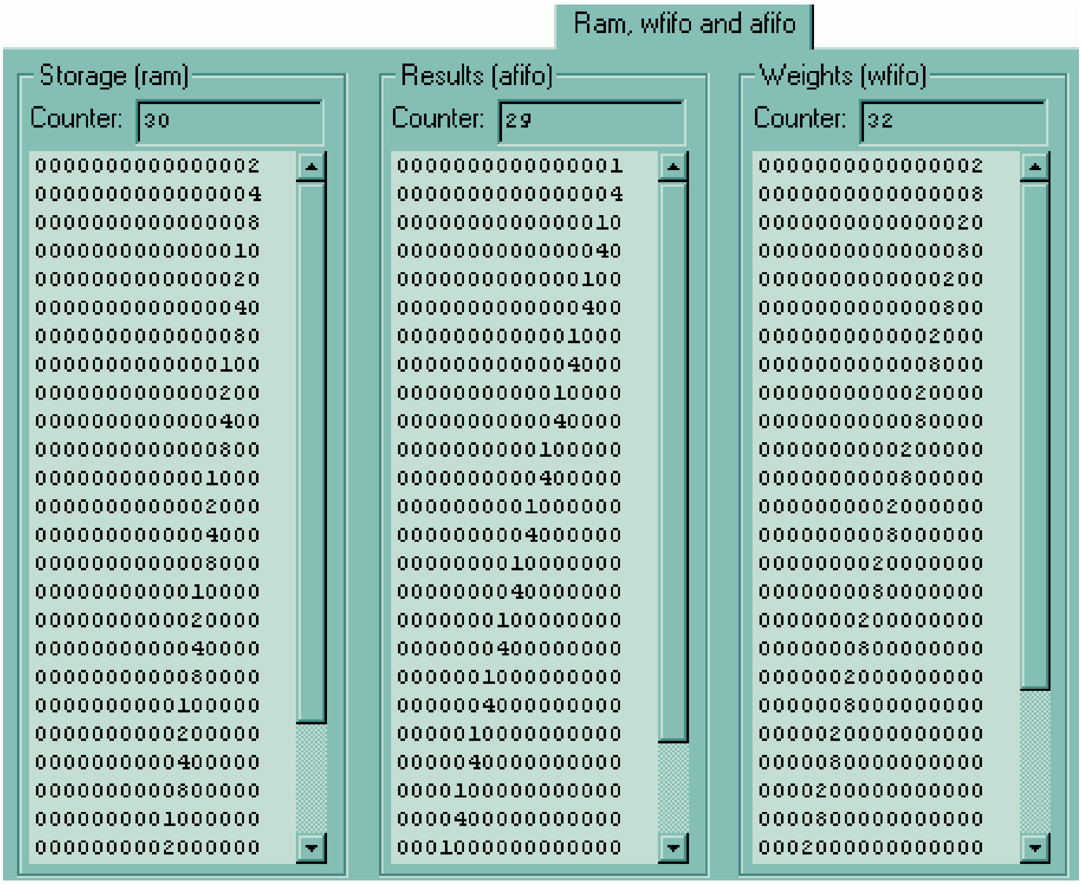
Neuron layout = 0000000080000000

Synapse layout = 01010100

	32	32
16	00000008	00000010
16	00000004	00000008
16	00000002	00000004
16	00000001	00000002

You cannot modify the contents of the matrixes .

6.3. Ram, wfifo and afifo page



Storage (ram)

Counter: 30

```
0000000000000002
0000000000000004
0000000000000008
0000000000000010
0000000000000020
0000000000000040
0000000000000080
0000000000000100
0000000000000200
0000000000000400
0000000000000800
0000000000001000
0000000000002000
0000000000004000
0000000000008000
0000000000010000
0000000000020000
0000000000040000
0000000000080000
0000000000100000
0000000000200000
0000000000400000
0000000000800000
0000000001000000
0000000002000000
```

Results (afifo)

Counter: 29

```
0000000000000001
0000000000000004
0000000000000010
0000000000000040
0000000000000100
0000000000000400
0000000000001000
0000000000004000
0000000000010000
0000000000040000
0000000000100000
0000000000400000
0000000001000000
0000000004000000
0000000010000000
0000000040000000
0000000100000000
0000000400000000
0000001000000000
0000004000000000
0000010000000000
0000040000000000
0000100000000000
0000400000000000
0001000000000000
```

Weights (wfifo)

Counter: 32

```
0000000000000002
0000000000000008
0000000000000020
0000000000000080
0000000000000200
0000000000000800
0000000000002000
0000000000008000
0000000000020000
0000000000080000
0000000000200000
0000000000800000
0000000002000000
0000000008000000
0000000020000000
0000000080000000
0000000200000000
0000000800000000
0000002000000000
0000008000000000
0000020000000000
0000080000000000
0000200000000000
0000800000000000
0002000000000000
```

This page displays the Vector Unit internal memory blocks based on the FIFO principle (so-called "registers-containers") and numbers of words in the registers-containers.

Registers-containers include:

- `ram` – register-container that is used to store and reuse the same data block in calculations,
- `wfifo` – register-container that is used to store weights which are then loaded to the Shadow Matrix,
- `afifo` – register-container that is used to store the result of every vector insruction.

You cannot modify the contents of internal memory blocks.