LAUTERBACH
DEVELOPMENT TOOLS

Arm ETM Programming Dialog

MANUAL

Arm ETM Programming Dialog

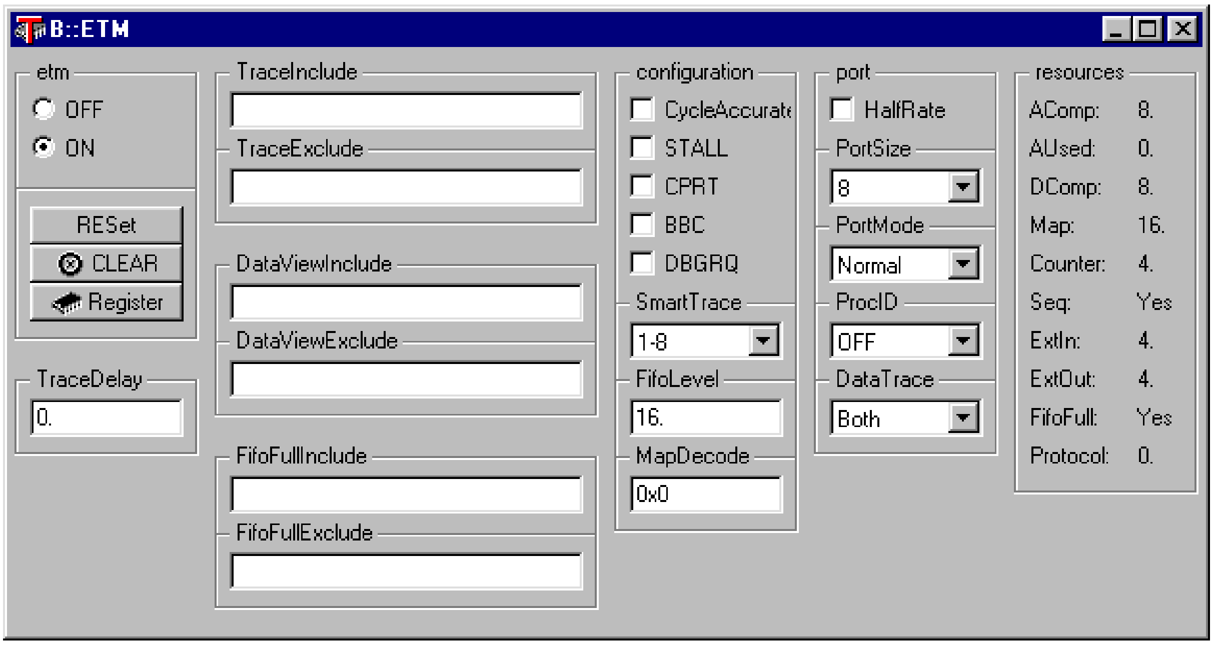
Arm ETM Programming Dialog

Version 06-Jun-2024

Initialization

Initialization of the ETM

Initialize the ETM by using **ETM Settings** in the **Trace** menu or by entering the command **ETM.state.**



etm: OFF [✓ ON]

TraceDelay: 0.

TraceInclude:

TraceExclude:

DataViewInclude:

DataViewExclude:

FifoFullInclude:

FifoFullExclude:

configuration: ☐ CycleAccurate ☐ STALL ☐ CPRT ☐ BBC ☐ DBGRQ

SmartTrace: 1-8

FifoLevel: 16.

MapDecode: 0x0

port: ☐ HalfRate

PortSize: 8

PortMode: Normal

ProcID: OFF

DataTrace: Both

resources:

AComp:	8.
AUsed:	0.
DComp:	8.
Map:	16.
Counter:	4.
Seq:	Yes
ExtIn:	4.
ExtOut:	4.
FifoFull:	Yes
Protocol:	0.

The broadcasting of the **program flow** can be controlled by **TraceInclude** (restrict the broadcasting of the program flow to the defined range) or **TraceExclude** (do not broadcast the program flow within the defined range). Either **TraceInclude** or **TraceExclude** can be specified.

The broadcasting of the **data flow** can be controlled by **DataViewInclude** (restrict the broadcasting of the data accesses to the defined range) or **DataViewExclude** (do not broadcast the data flow within the defined range).

For further details of this window please look for the manual **RiscTrace for ARM-ETM**.

Programming

How to Start

The **ETM Programming** Dialog can be found on any TRACE32 CD dated September 2000 and later. To open the **ETM Programming** dialog use ***Trigger Dialog*** … in the ***Trace*** menu.

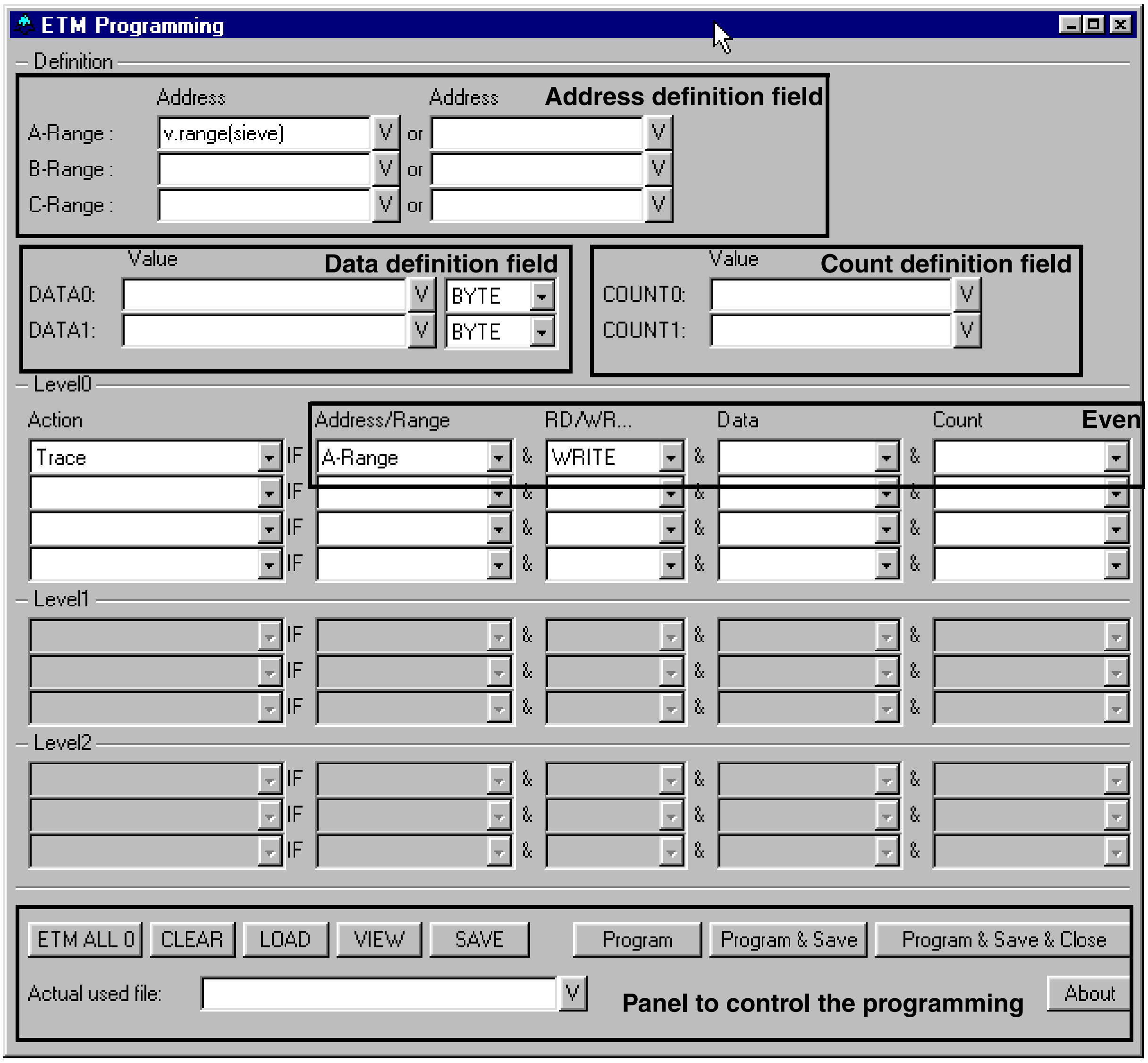
An empty dialog box will appear. Please go on reading this manual to get an overview on the functionality of the dialog.

Elements in the Dialog Box

Remark

- The intention of the **ETM Programming** dialog is to provide an intuitive and easy to use interface to program the **Embedded Trace Macrocell (ETM)** of ARM7 / ARM9.
- The ETM Programming Dialog Window does not represent the full functionality of the ETM.

Overview



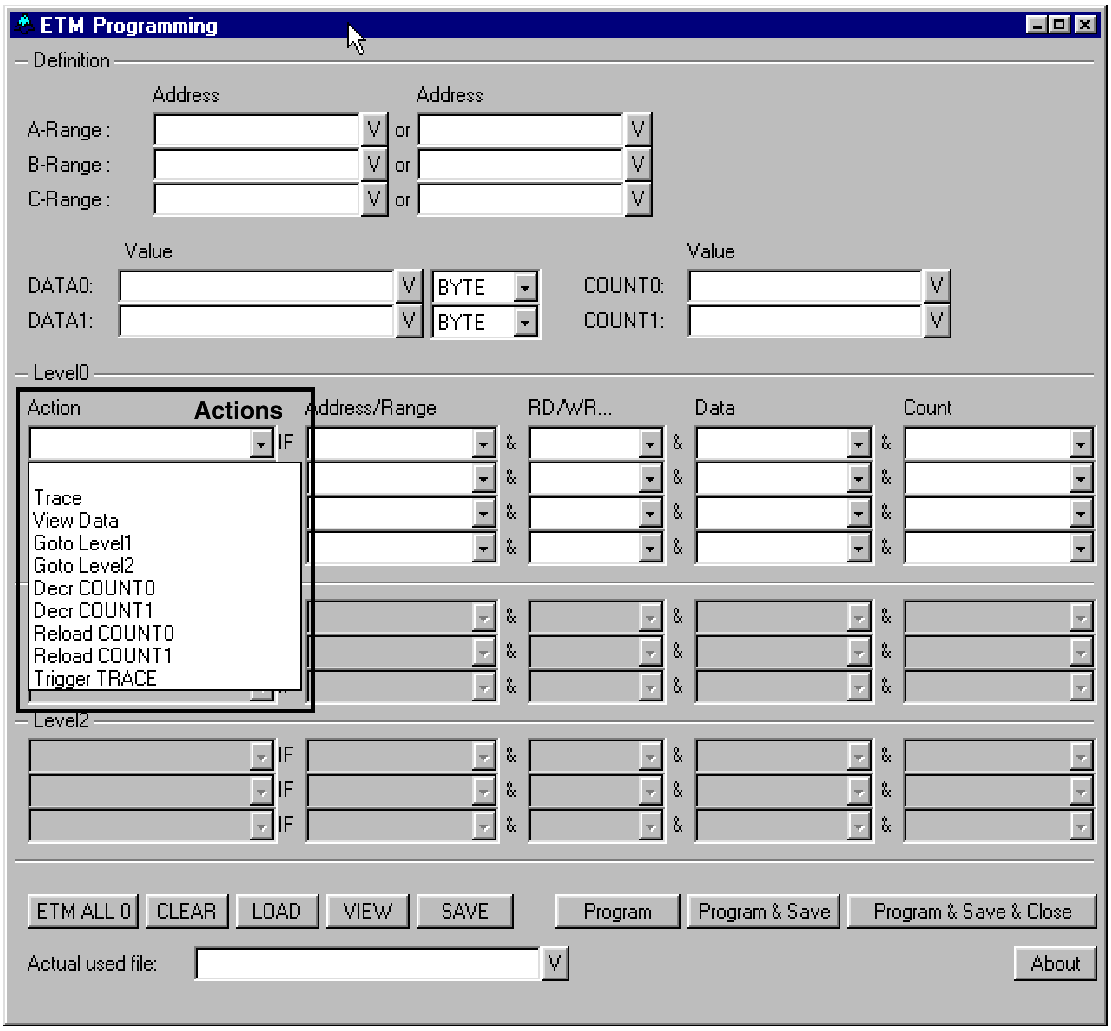
	Address		Address
A-Range :	v.range(sieve)	or	
B-Range :		or	
C-Range :		or	

	Value	
DATA0:		BYTE
DATA1:		BYTE

	Value
COUNT0:	
COUNT1:	

Action		Address/Range		RD/WR...		Data		Count
Trace	IF	A-Range	&	WRITE	&		&	

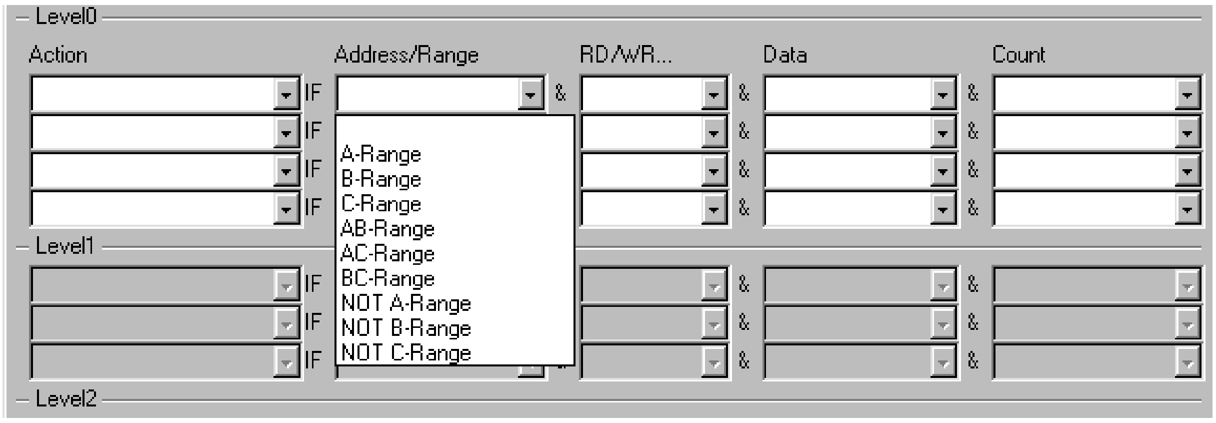
Actions



An ETM Trigger Program performs ***ACTIONS*** depending on the state of the user program/target hardware.

Trace	Trace instruction execution cycle
View Data	Trace data access cycles
Trigger TRACE	Stop tracing
Decr Count0 **Decr Count1**	Decrement the counter identified by the label **COUNT0** or **COUNT1**.
Reload Count0 **Reload Count1**	Set the counter identified by the label **COUNT0** or **COUNT1** to value specified in definition.
Goto Level0 **Goto Level1** **Goto Level2**	Go on tracing according to the actions defined under **Level0**, **Level1** or **Level2**.

Events



The state of the user program/target system is described by different characteristics. Different characteristics combined through **LOGICAL AND** form an event. An action will only be performed if the user program/target system is in the state defined by the event. The characteristics are:

Address/Range	Address or address range To be defined in the address definition fields **A-Range**, **B-Range** or **C-Range**
RD/WR	CPU cycle type **FETCH, EXECUTE, READ**, **WRITE** or **ACCESS** (READ or WRITE)
Data	CPU data To be defined in the data definition fields **DATA0** or **DATA1**
Count	Counter or Time Counter To be defined in the count definition fields **COUNT0** or **COUNT1**

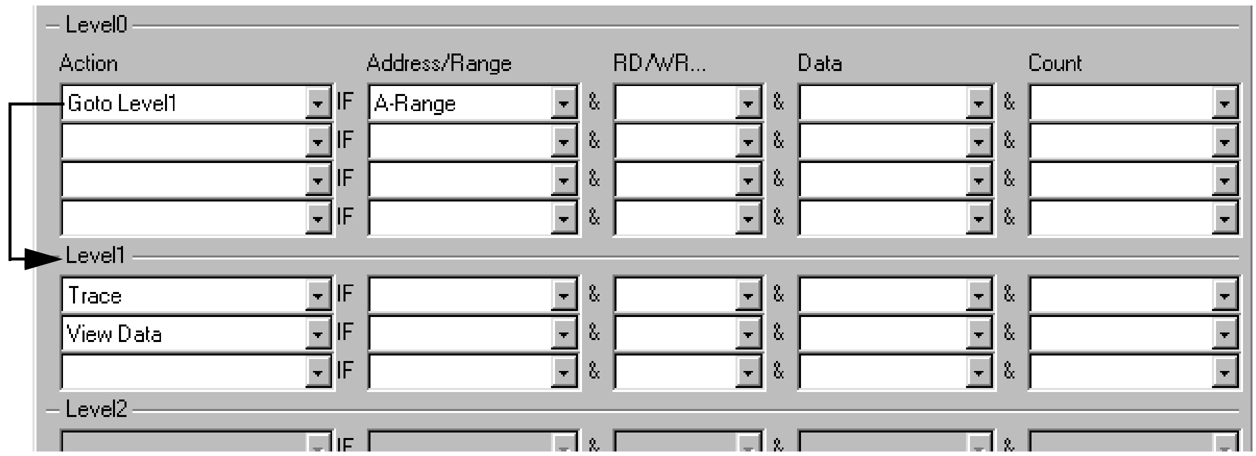
If no characteristic is chosen the action is carried out at **every** CPU cycle.

Levels

Three different levels can be used to define a set of actions. The levels can be referenced by the logical names **Level0**, **Level1** and **Level2**. Within the ETM the levels are implemented as states. Use the action **Goto Level x** to let the system switch to Level x and to carry out only the actions under Level x.

After programming the ETM with **Goto Level *x*** actions the state sequencer will be set to **Level 0**.

Be aware Goto Level x used in different Levels does not mean the same. It always implies a transaction from the current Level to Level x. e.g using Goto Level 2 in the area at Level 0 means the transaction from Level 0 to Level 2



Level0

Action		Address/Range		RD/WR...		Data		Count
Goto Level1	IF	A-Range	&		&		&	
	IF		&		&		&	
	IF		&		&		&	
	IF		&		&		&	

Level1

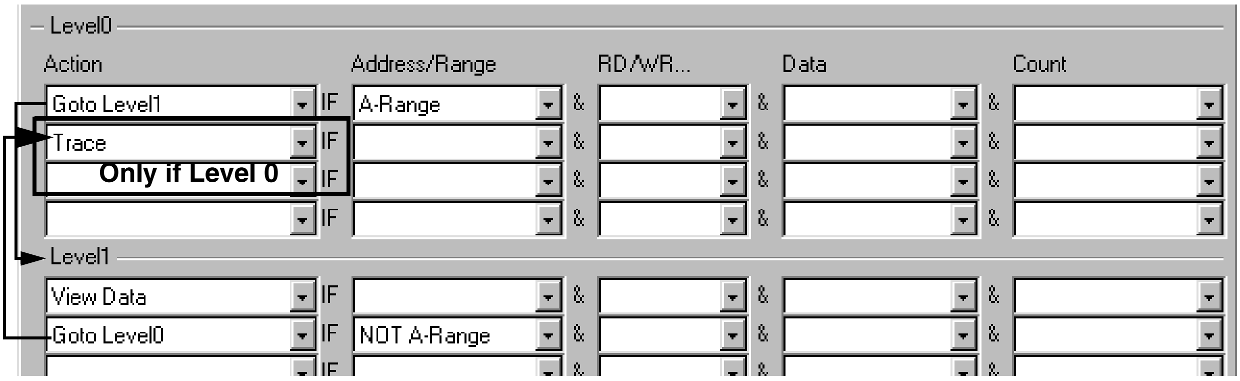
Action		Address/Range		RD/WR...		Data		Count
Trace	IF		&		&		&	
View Data	IF		&		&		&	
	IF		&		&		&	

Level2

Action		Address/Range		RD/WR...		Data		Count
	IF		&		&		&	

For this example trace and data access trace is activated if A-Range is reached. The state sequencer changes form Level 0 to Level1 if an address within A-Range is reached. What you don’t see is that the actions Trace and View Data only take place if the system is in Level 1. That means Trace and View Data have a implied condition “if Level1”.

Special Feature in Level 0:
Actions in Level 0 that are used after a **Goto Level 1** or **Goto Level 2** command get the implied condition “if Level 0”. If they are used before the Goto Level x command Actions are carried out in **every** Level.

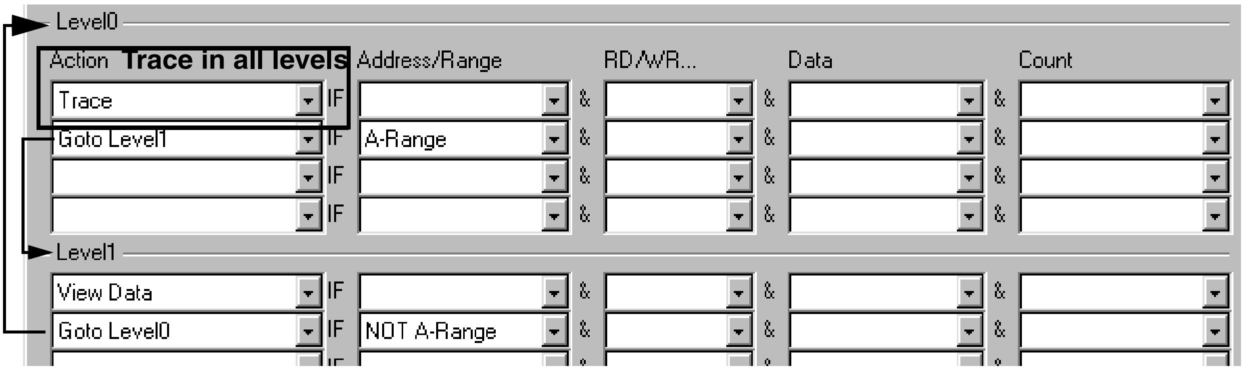


Level0

Action		Address/Range		RD/WR...		Data		Count
Goto Level1	IF	A-Range	&		&		&	
Trace	IF		&		&		&	
	IF		&		&		&	
	IF		&		&		&	

Level1

Action		Address/Range		RD/WR...		Data		Count
View Data	IF		&		&		&	
Goto Level0	IF	NOT A-Range	&		&		&	



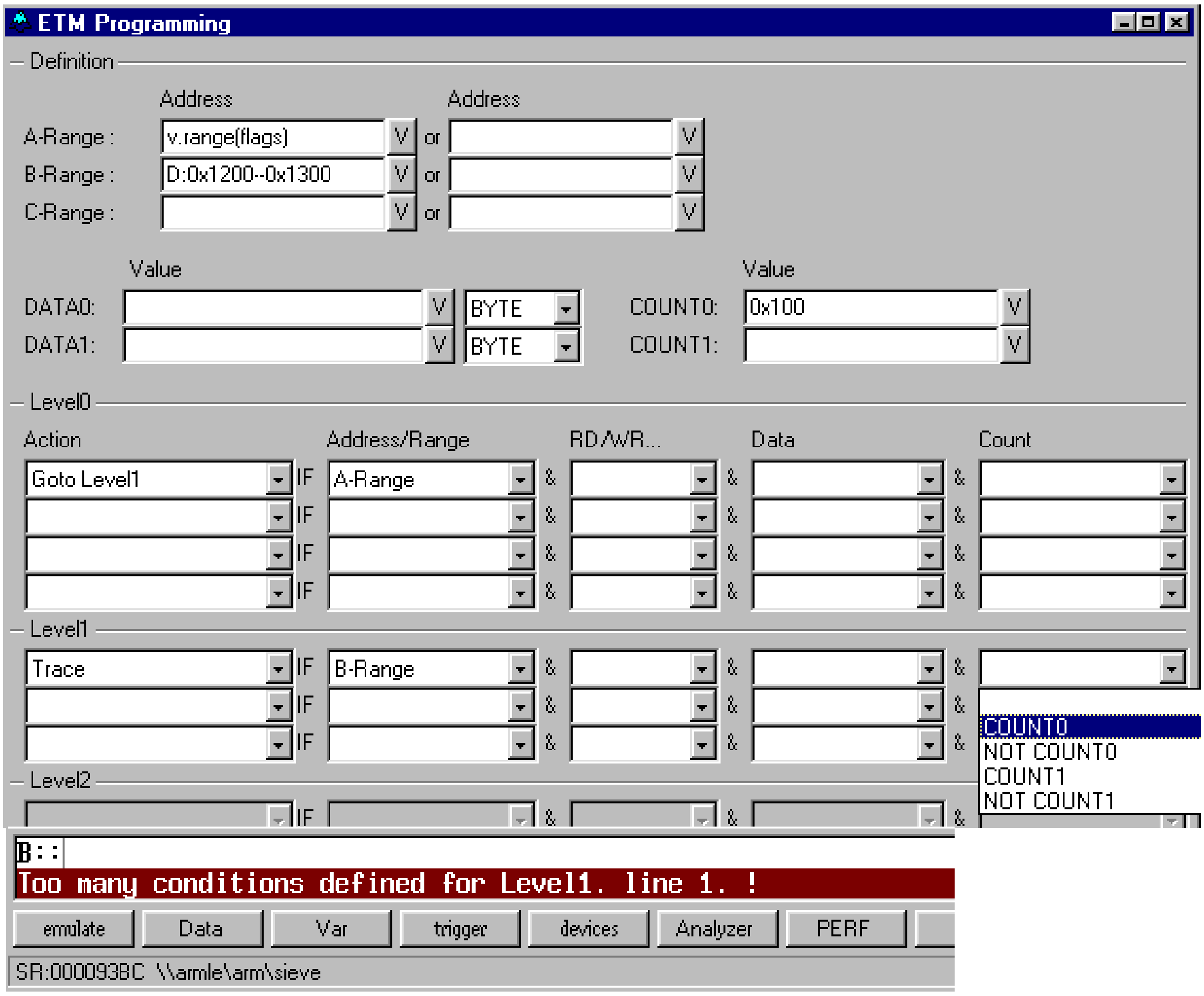
Level0

Action		Address/Range		RD/WR...		Data		Count
Trace	IF		&		&		&	
Goto Level1	IF	A-Range	&		&		&	
	IF		&		&		&	
	IF		&		&		&	

Level1

Action		Address/Range		RD/WR...		Data		Count
View Data	IF		&		&		&	
Goto Level0	IF	NOT A-Range	&		&		&	

⚠	Restrictions due to implied Level x condition: Due to the fact that only one logical combination AND and OR are possible within an event (see also Address Definition) in Level 1 and Level 2 and in the area after a Goto Level x of Level 0 only one further condition is allowed. This means either address range or counter could be used. The first part of the event is used by the Level x condition. If more than one combination has to be used to build an event it will result in an error message.


ETM Programming

Definition

	Address			Address	
A-Range :	v.range(flags)	V	or		V
B-Range :	D:0x1200--0x1300	V	or		V
C-Range :		V	or		V

	Value				Value	
DATA0:		V	BYTE	COUNT0:	0x100	V
DATA1:		V	BYTE	COUNT1:		V

Level0

Action		Address/Range		RD/WR...		Data		Count
Goto Level1	IF	A-Range	&		&		&	
	IF		&		&		&	
	IF		&		&		&	
	IF		&		&		&	

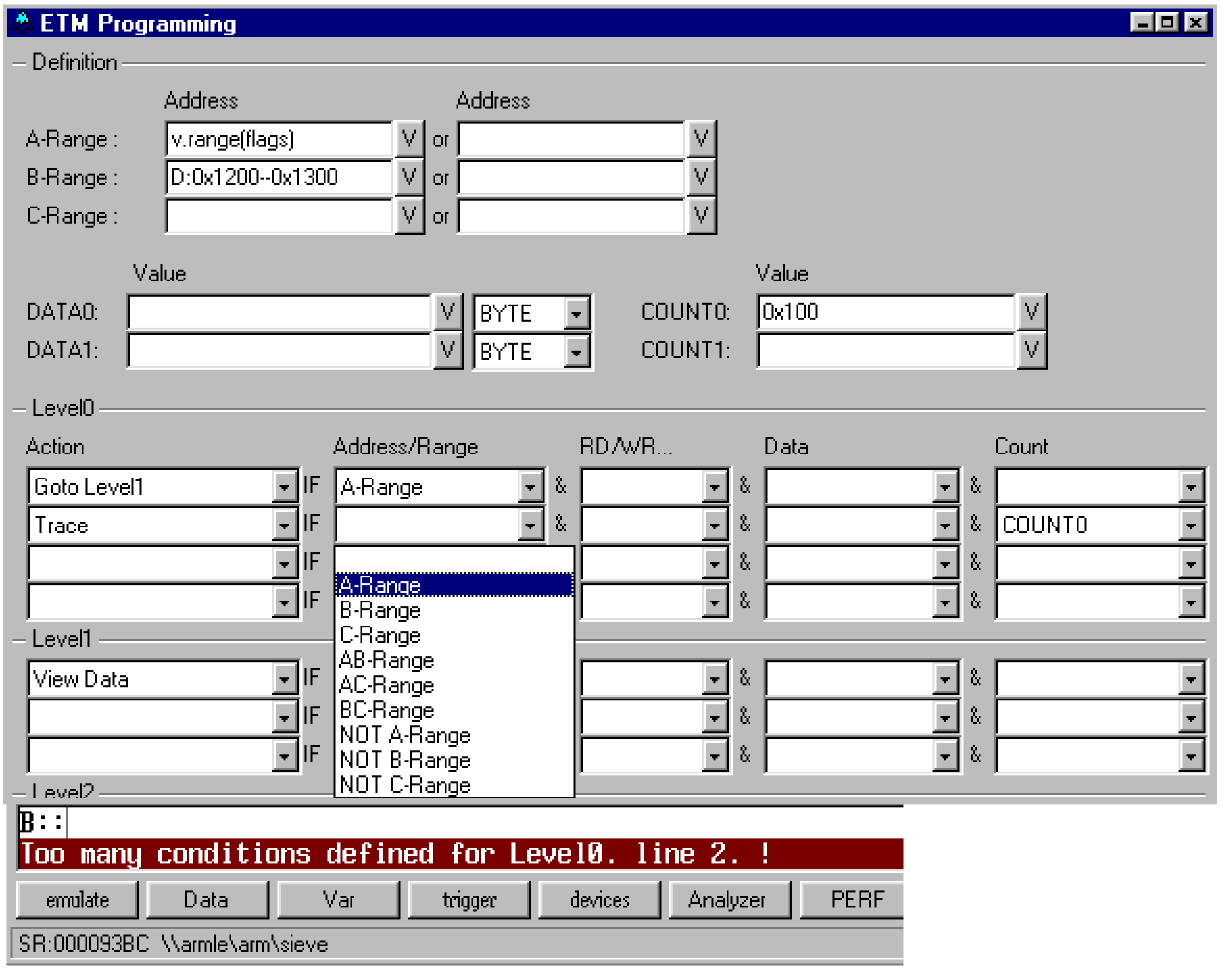
Level1

Action		Address/Range		RD/WR...		Data		Count
Trace	IF	B-Range	&		&		&	
	IF		&		&		&	
	IF		&		&		&	

Level2

```
B::
Too many conditions defined for Level1. line 1. !
```

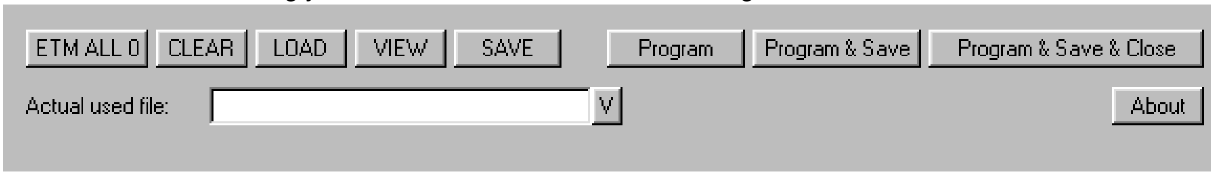
SR:000093BC \\armle\arm\sieve



```
B::
Too many conditions defined for Level0. line 2. !
```

SR:000093BC \\armle\arm\sieve

At the bottom of the dialog you find the buttons to control the dialog.



If you save the contents of the dialog a **PRACTICE** file is generated. To program the ETM either use the buttons Program or Program&Save or Program & Save & Close. You can also run the generated PRACTICE file from the command line of TRACE32 with the command **DO** *<file>* to program the ETM.

Use the button **ETM ALL 0** to set all ETM registers to 0. Compared to **ETM.RESet** this command does not reset the address and data comparators, the counter actions and the sequencer actions in the ETM. This could cause strange result in the output.

Use the button **CLEAR** to reset the dialog window if you want to begin from the scratch.

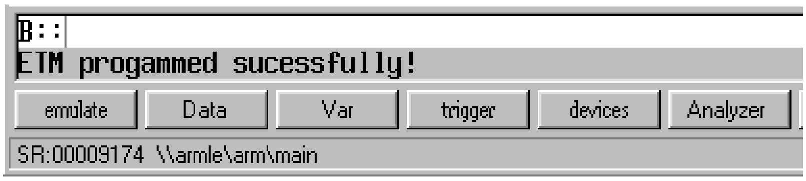
With the button **LOAD** an earlier designed ETM program file can be read into the dialog window. The ETM is not programmed.

!	Only ETM programs written with the ETM Dialog can be reloaded into the dialog.

Press the button **VIEW** and you will see the contents of the dialog written down in statements of the ETM programming language. Use this button now and then to learn the basics of the ETM programming language.

If you do not want the content of the dialog to be save to a file use the button **Program** to program the ETM.

Program & Save will write the contents of the dialog to the file which is specified under **Actual used file**. After that the ETM will be programmed. If no file name is given you will be asked to define one. The ETM is successfully programmed if you get the message in the state line of TRACE32.

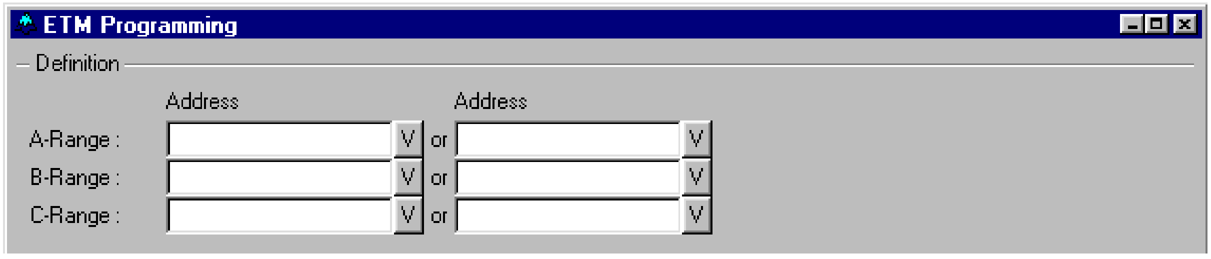


```
B::
ETM progammed sucessfully!
```

Otherwise the error is shown in the state line of TRACE32 or a message box appears.

Program & Save & Close works the same as **Program & Save** except that the dialog will be closed after the ETM is programmed successfully.

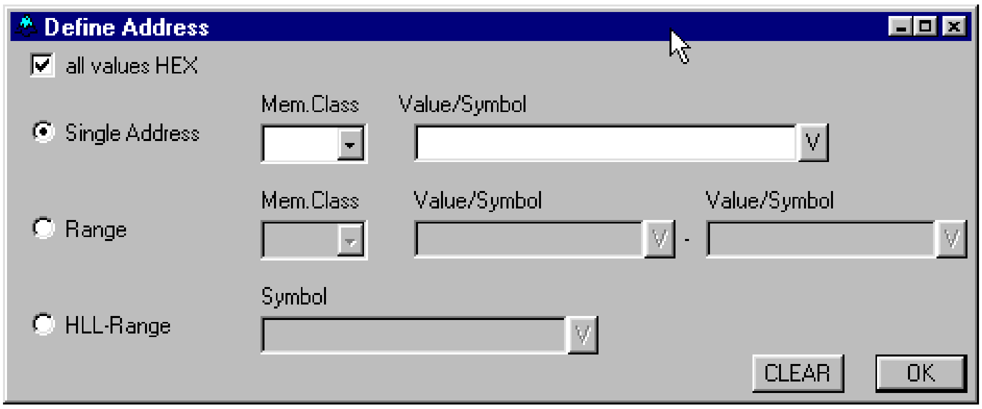
Address/Range Definition



Up to three different address areas can be defined in this part of the dialog. This areas are later referenced by the logical names **A-Range**, **B-Range** or **C-Range**. The logical name **AB-Range** defines an area declared through the area **A-Range** combined with the area **B-Range** by **LOGICAL AND**.

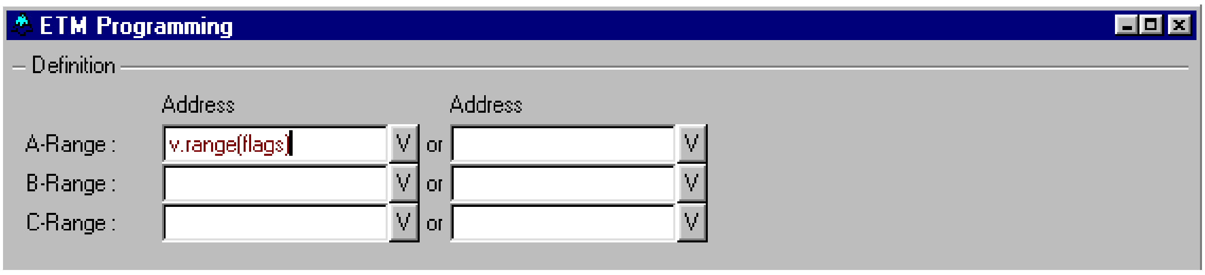
Each area e.g. **A-Range** allows to define the address space through two different singular address parts. One of this part can hold a single address or an address range. The singular address parts are combined through **LOGICAL OR** to form the characteristic named **A-Range**.

If you don't know how to specify an address or an address range click the **V** button.



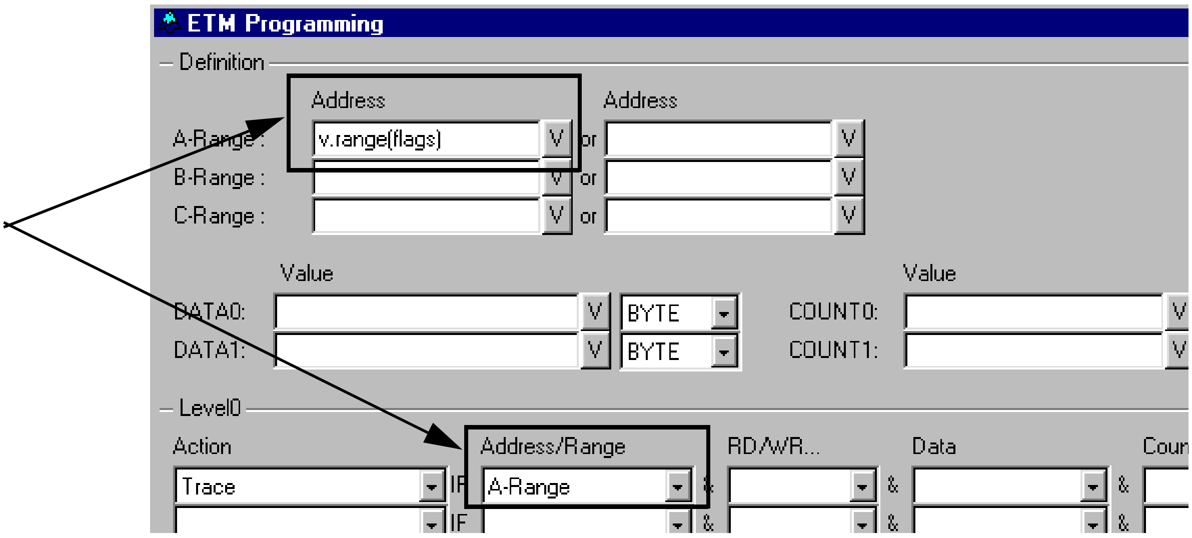
With the appearing sub dialog a single address or any kind of range can be defined easily.

As you are familiar with the syntax of defining an address in TRACE32 just fill out the dialog.

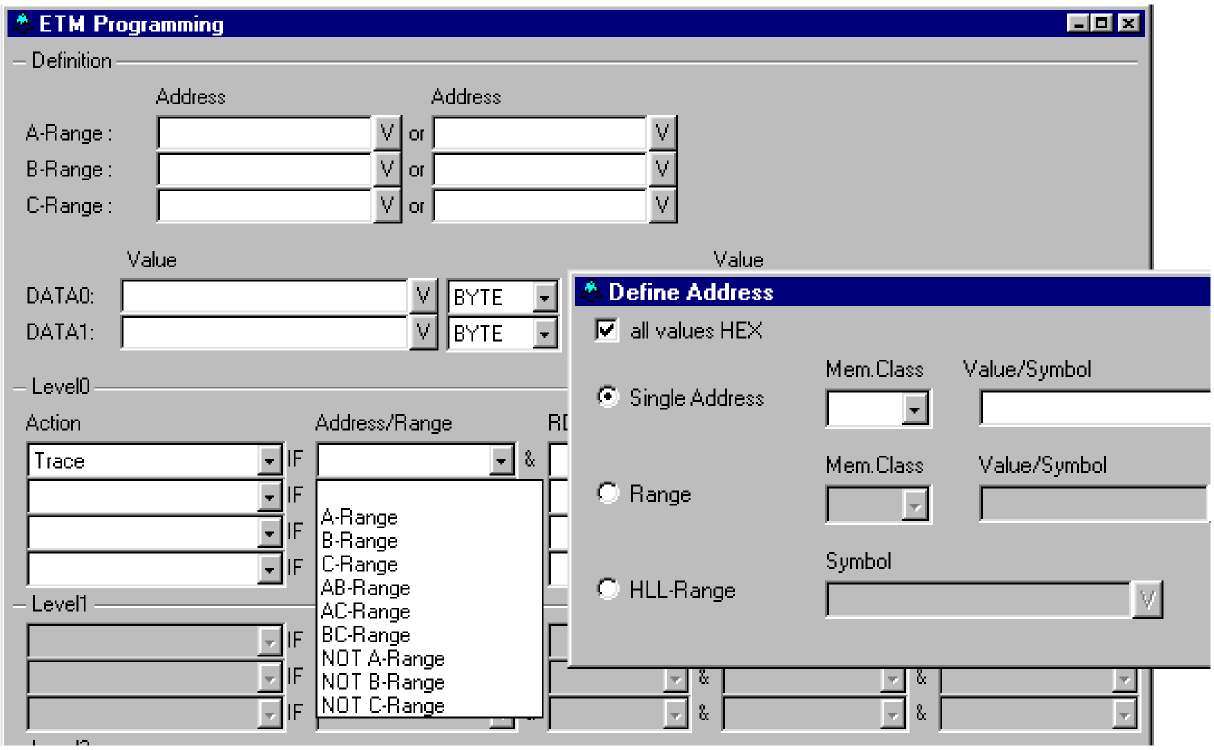


A-Range : v.range(flags)

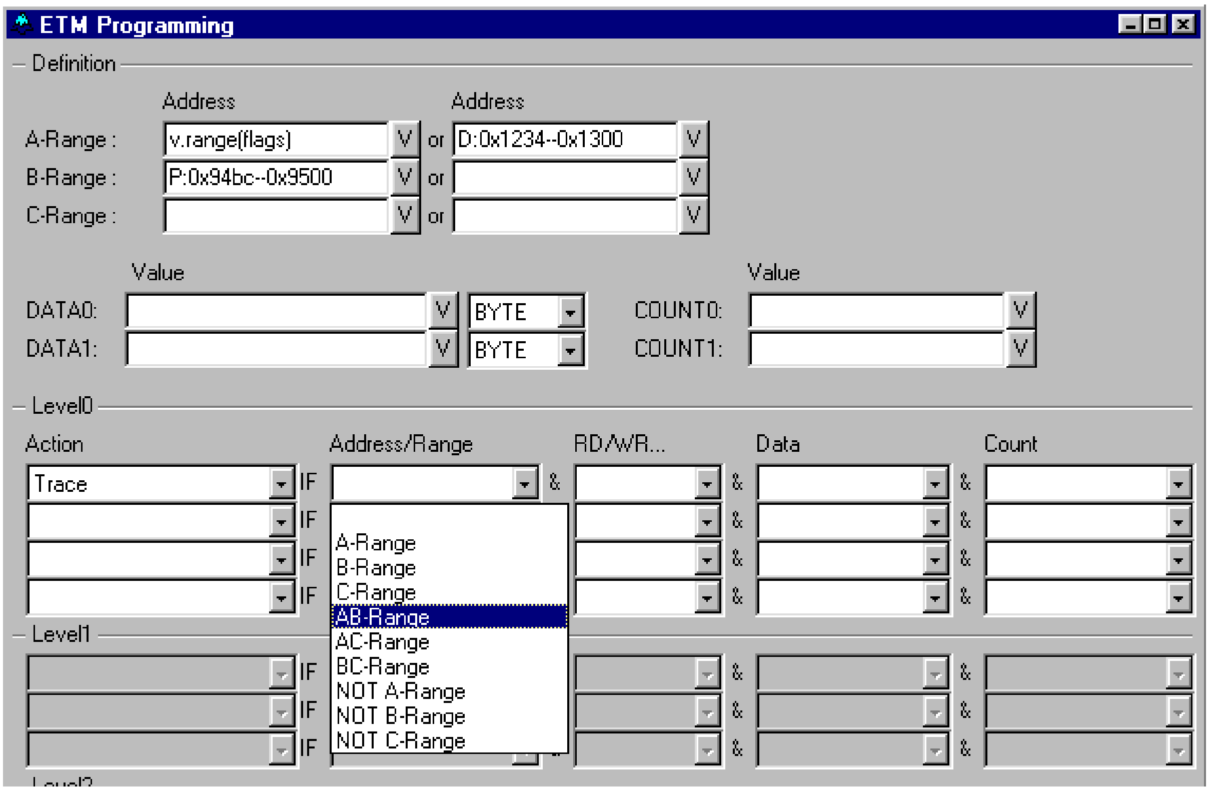
The defined address area will be used to define the condition for carrying out an action in the main dialog.

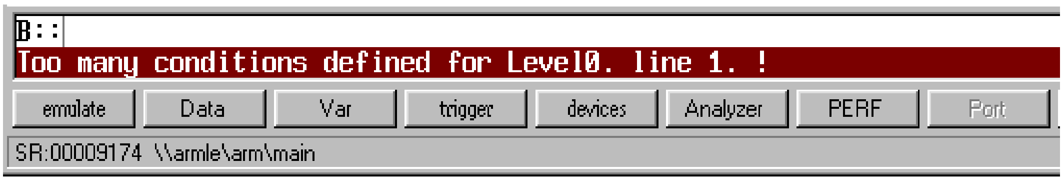


If not address area is defined at the time you want to choose an address condition you will be asked to do so.

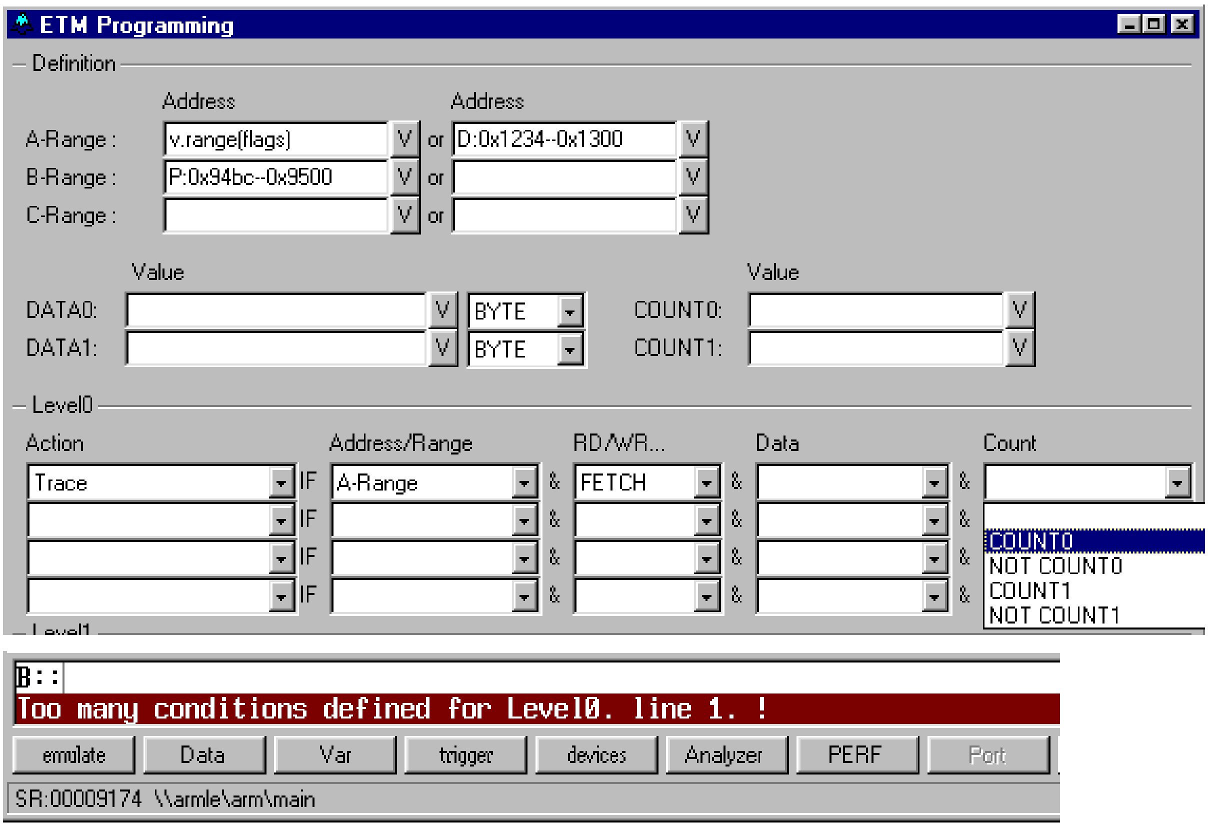


!	There is only one logic combination of AND or OR available at the ETM. Therefore you should be aware if you use the two address definition areas of a x-Range combined by “or” you could not use either a combination range like AB-/AC-/BC-Range nor a counter within the same event.





```
B::
Too many conditions defined for Level0. line 1. !
```



Definition

	Address		Address
A-Range :	v.range(flags)	or	D:0x1234--0x1300
B-Range :	P:0x94bc--0x9500	or	
C-Range :		or	

	Value			Value
DATA0:		BYTE	COUNT0:	
DATA1:		BYTE	COUNT1:	

Level0

Action		Address/Range		RD/WR...		Data		Count
Trace	IF	A-Range	&	FETCH	&		&	
	IF		&		&		&	COUNT0
	IF		&		&		&	NOT COUNT0
	IF		&		&		&	COUNT1
								NOT COUNT1

Level1

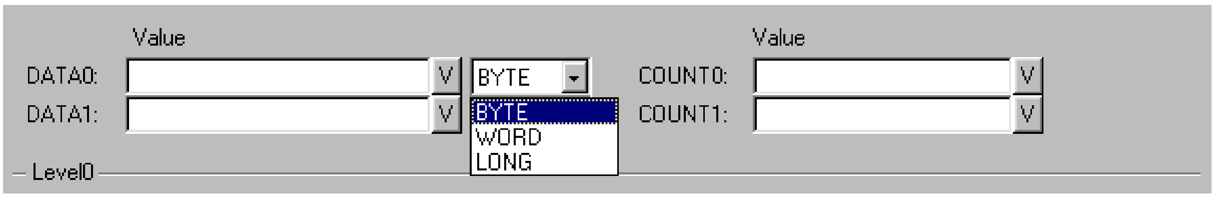
```
B::
Too many conditions defined for Level0. line 1. !
```

SR:00009174 \\armle\arm\main

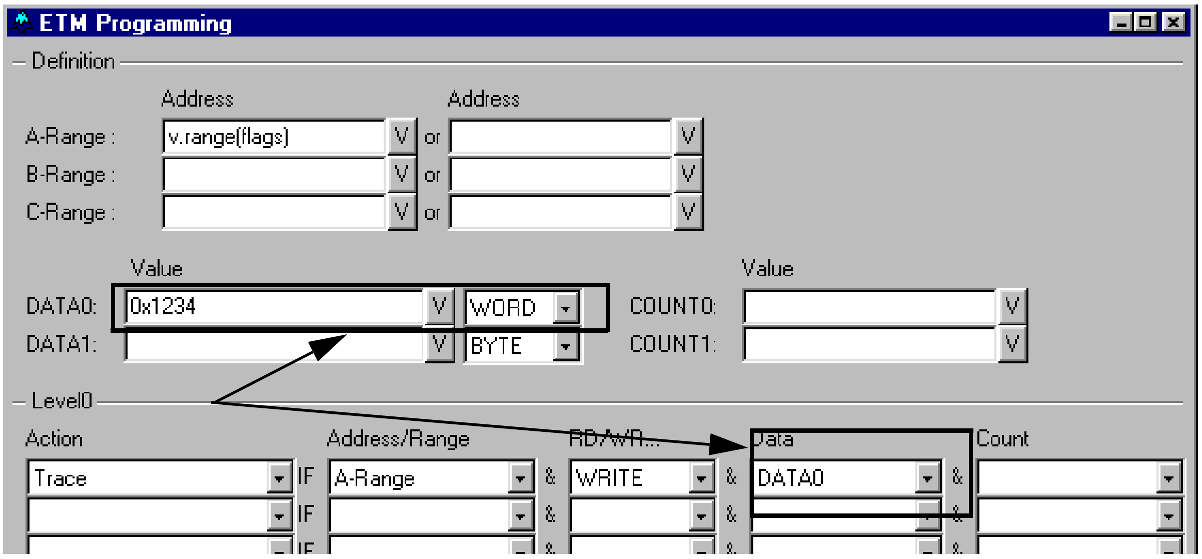
RD/WR.. and **Data** conditions are internally connected to Address/Range therefore you can use them without restrictions for the logical combination.

Be aware if you use RD/WR.. e.g FETCH with a special range e.g. A-Range you are not able to use A-Range with a different RD/WR.. condition e.g. ACCESS. The same applies if you use A-Range with condition DATA0/1.

Data Definition

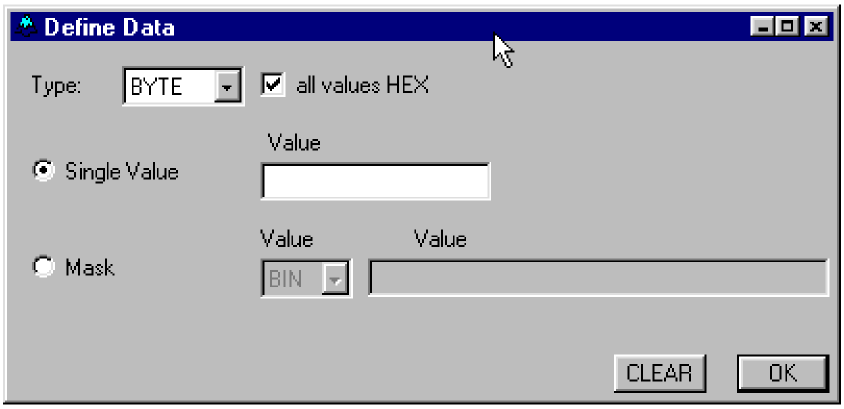


Two different data values can be used to observe the data on the data bus. The data values are later referenced by the labels **DATA0** and **DATA1**.



A-Range : v.range(flags)
B-Range :
C-Range :
DATA0: 0x1234 WORD
DATA1: BYTE
Trace IF A-Range & WRITE & DATA0

The data is defined through the value and the type. If you don’t know how to specify a data value just click the **V** button.

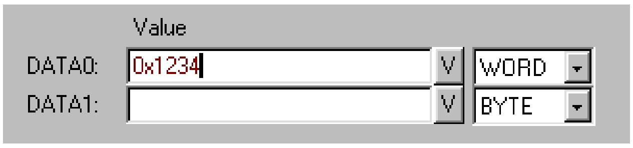


Type: BYTE
[✓] all values HEX
[● Single Value]
○ Mask BIN

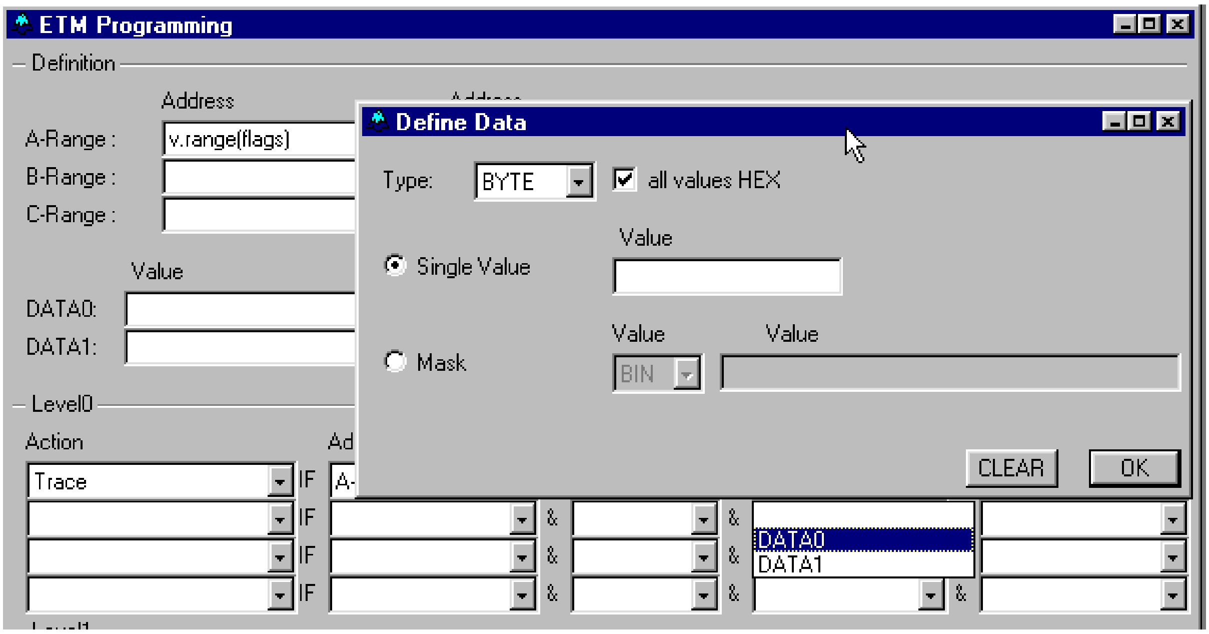
With the appearing subdialog a single data, a data range or a mask can be defined easily.

With the type you specify the width of the value. **BYTE**, **WORD** and **LONG** are available.

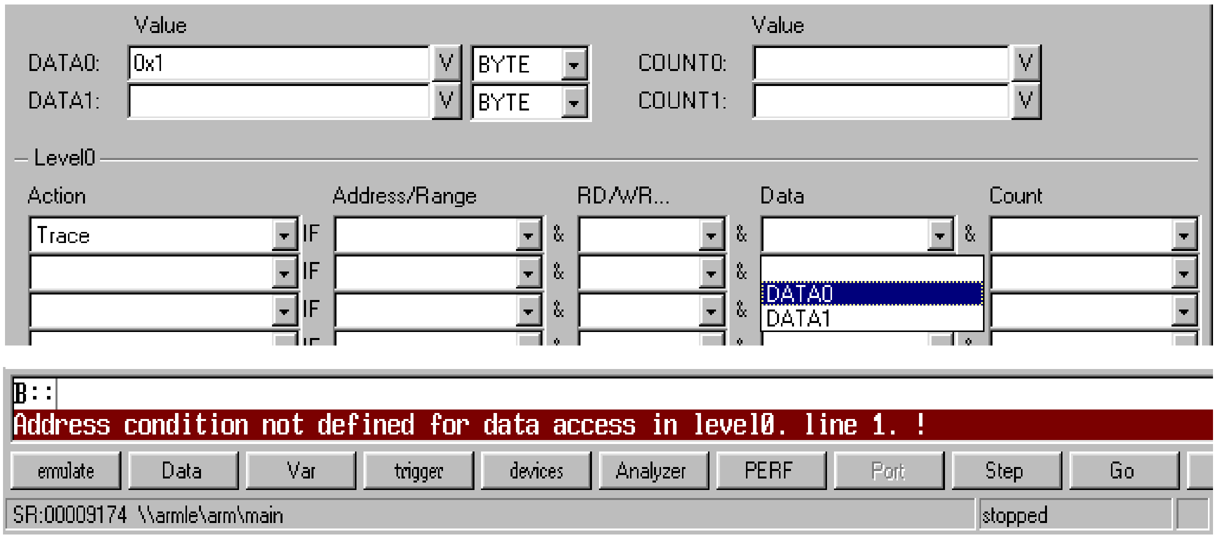
As you are familiar with the syntax of defining data in TRACE32 just fill out the dialog.



If there is no data defined at the time you want to use a data condition you will be asked to do so.



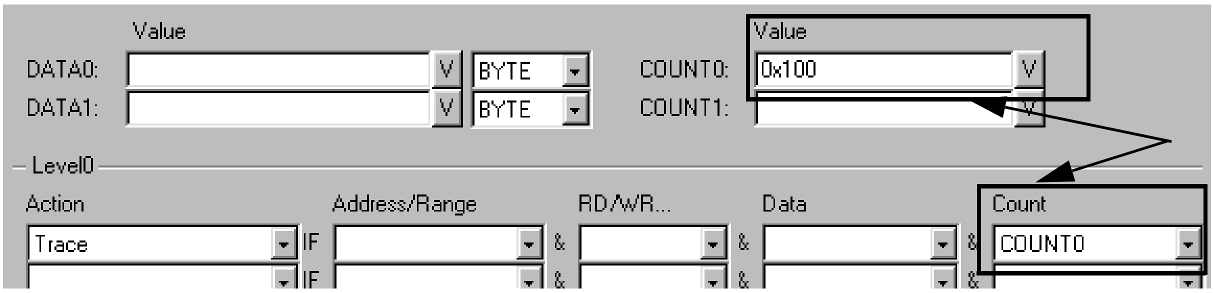
	As the Data comparators of the ETM are strongly connected to the address range you cannot use a Data comparator without an address range. If you do so you get an error message.



```
Address condition not defined for data access in level0. line 1. !
```

Counter Definition

For controlling the flow two different counters can be defined. For each counter you can define the maximum value. The counters are referenced in the condition of an action by the labels **COUNT0** and **COUNT1**.



DATA0: BYTE
DATA1: BYTE
COUNT0: 0x100
COUNT1:

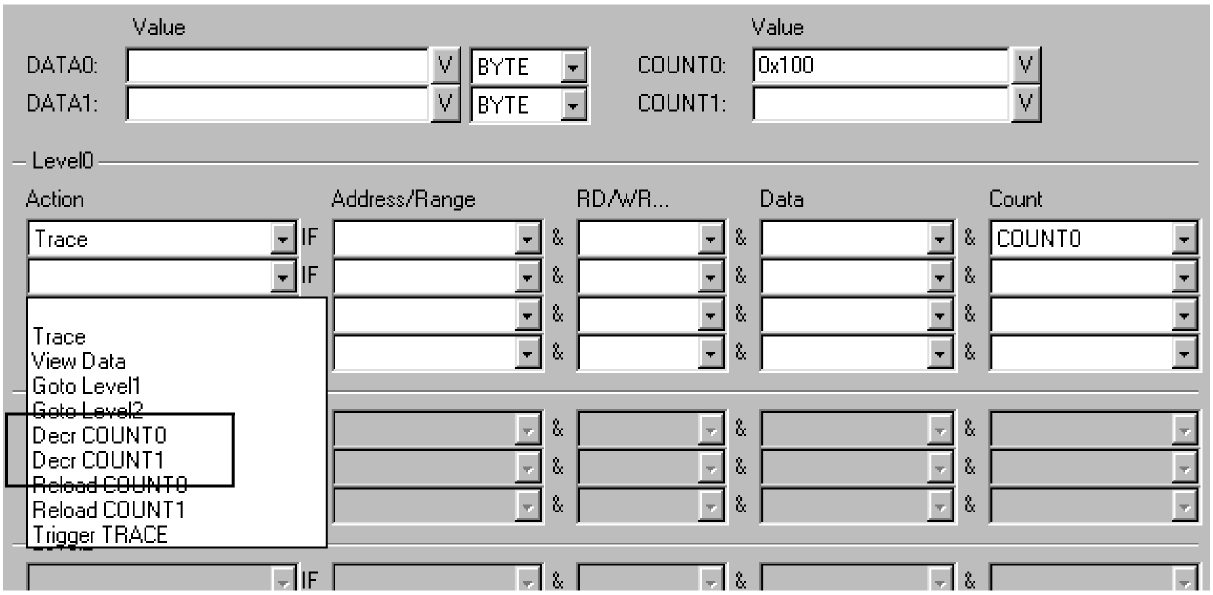
Level0

Action: Trace IF Address/Range & RD/WR... & Data & Count: COUNT0

After programming the ETM the counter is set to 100H. If no action is used to decrement the counter COUNT0 (**Decr Count 0**) defined the counter is continuously decremented at **full system clock speed**.

!	If you use Trace with **condition COUNT0/1** sampling begins when the counter **COUNT0/1 is zero**. If you use Trace with condition **NOT COUNT0/1** sampling only takes place if the counter **COUNT0/1 is bigger than zero**.

To control the value of the counters the actions **Decr Count0/1** and **Restart Count0/1** are available.



DATA0: BYTE
DATA1: BYTE
COUNT0: 0x100
COUNT1:

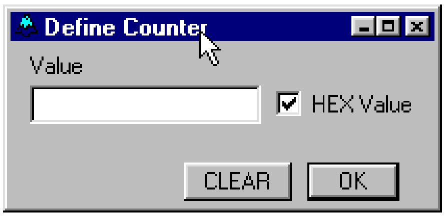
Level0

Action: Trace IF Address/Range & RD/WR... & Data & Count: COUNT0

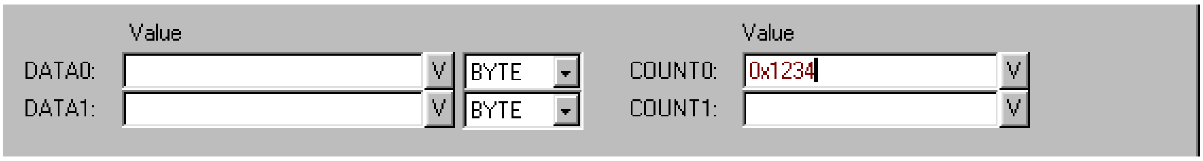
Decr COUNT0/1 substracts 1 from the value of the counter COUNT0/1.
Reload COUNT0/1 sets the value of the counter COUNT0/1 to the value specified in the definition.

If you don't use **Decr Count x** to set the counter or you don't use an event for this action the counter decrements at **full system clock speed**. For this case it is recommended to use the **Reload COUNT** x action to reload the counter at a definite point otherwise the counter will be run to zero till you start the program run.

If you don't know how to define the value of a counter just click the **V** button.

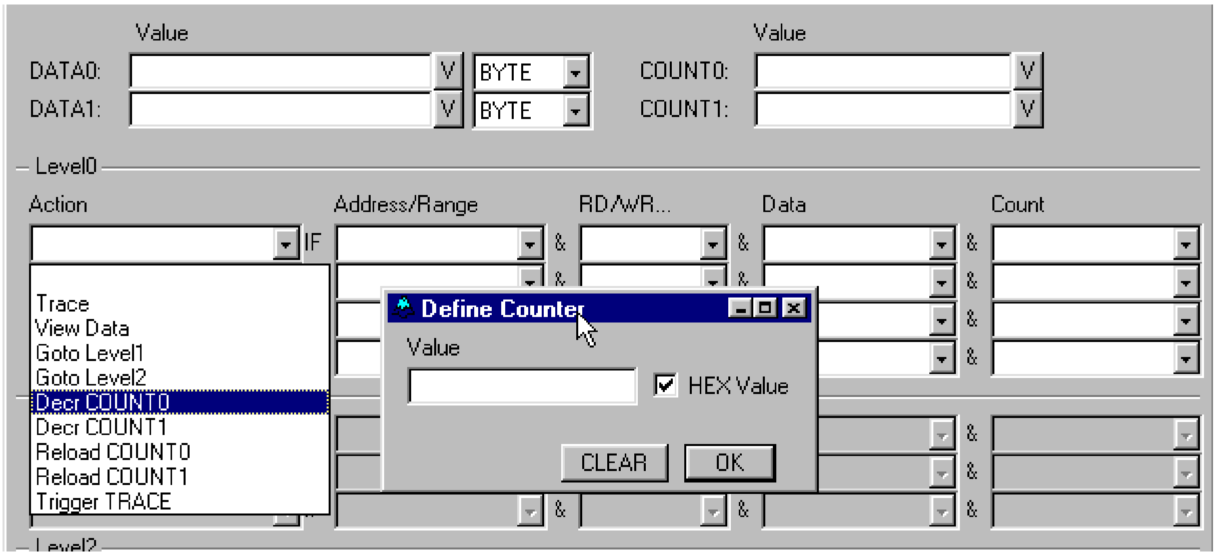


As you are familiar with the syntax of defining counters in TRACE32 just fill out the dialog.

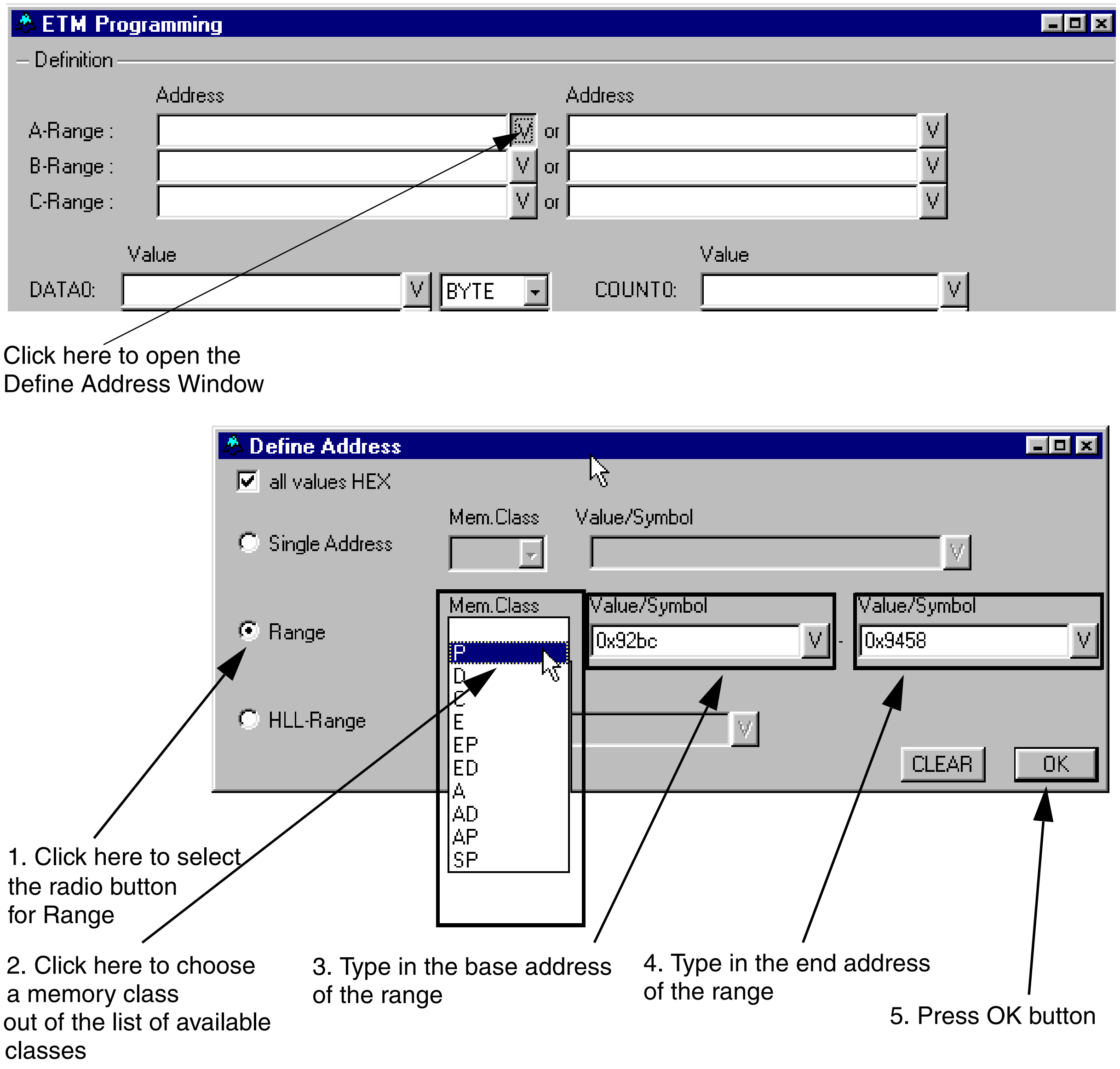


COUNT0: 0x1234

If there is no counter defined at the time you want to use a counter name in a condition or an action you will be asked to define it.



Example 1: Selective Trace on an Address Range



all values HEX: ✓

Range: ●

Value/Symbol: 0x92bc - 0x9458

Click here to open the Define Address Window

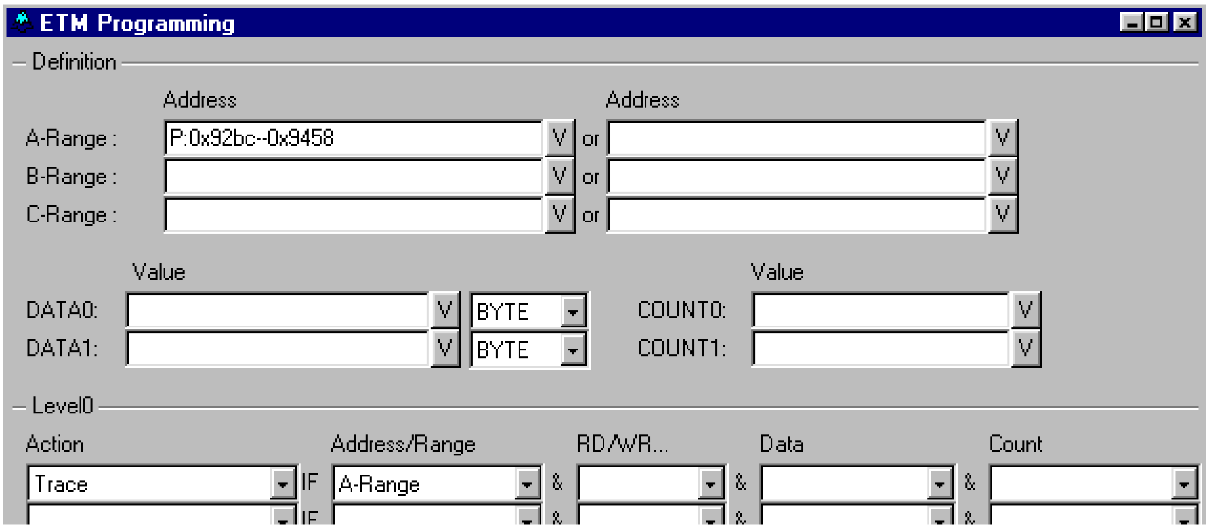
1. Click here to select the radio button for Range

2. Click here to choose a memory class out of the list of available classes

3. Type in the base address of the range

4. Type in the end address of the range

5. Press OK button


ETM Programming

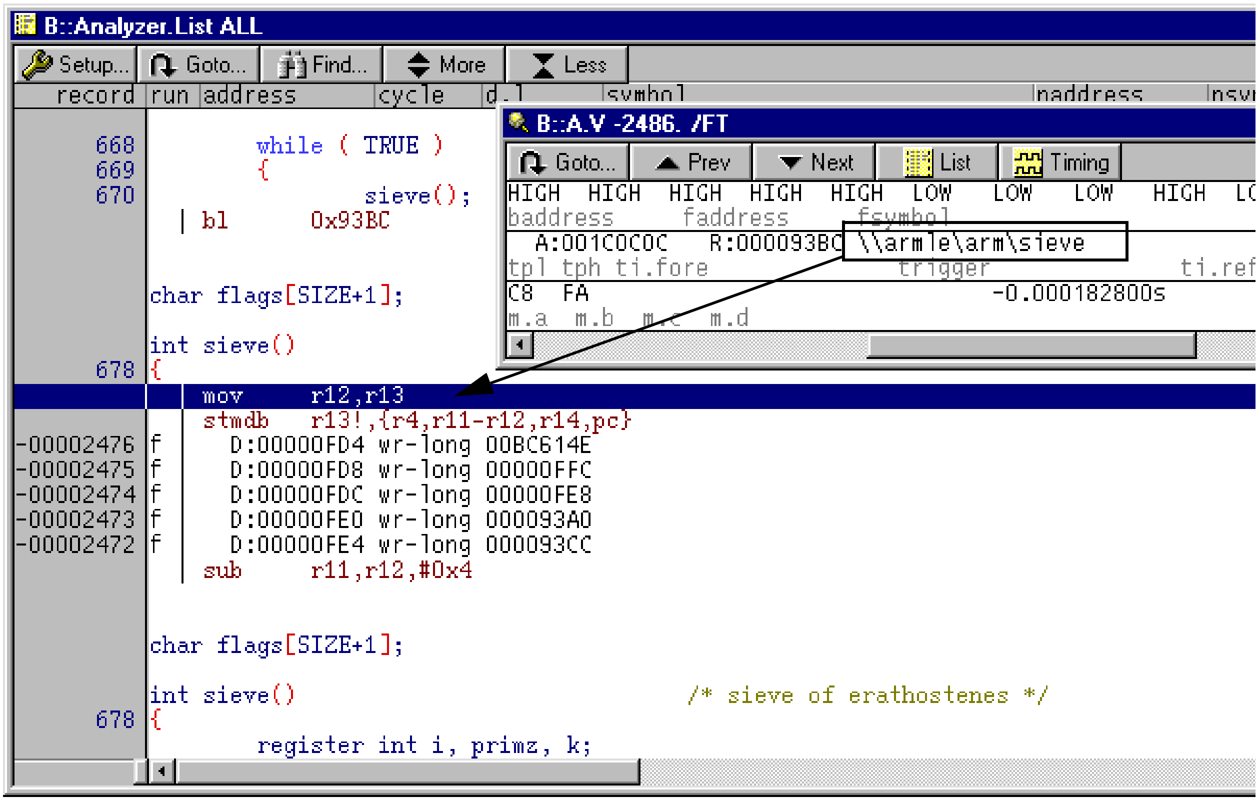
Definition

	Address		Address
A-Range :	P:0x92bc--0x9458	or	
B-Range :		or	
C-Range :		or	

	Value			Value
DATA0:		BYTE	COUNT0:	
DATA1:		BYTE	COUNT1:	

Level0

Action		Address/Range		RD/WR...		Data		Count
Trace	IF	A-Range	&		&		&	

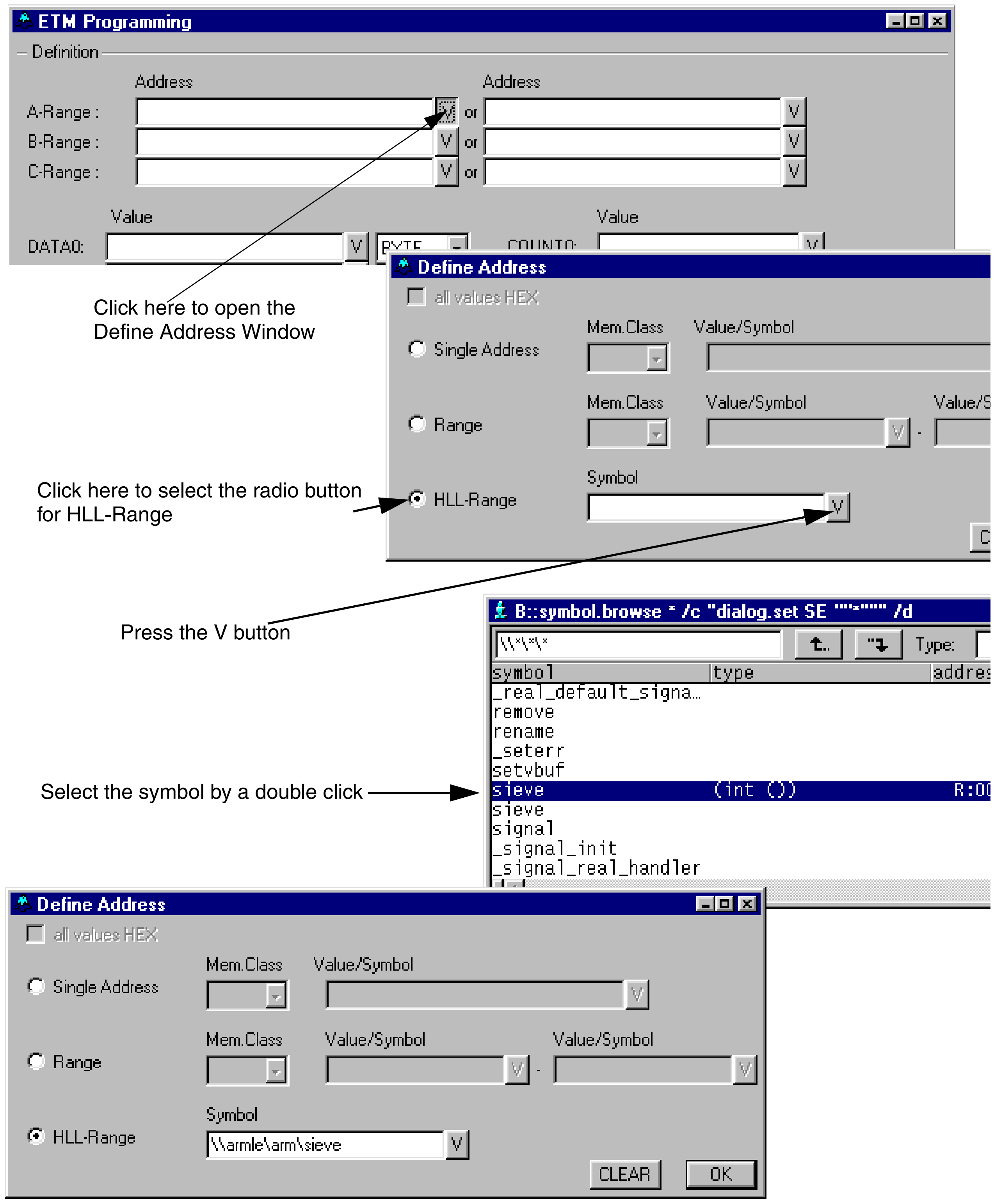

B::Analyzer.List ALL

```
      record run address     cycle   d.l      symbol                naddress  nsym
         668         while ( TRUE )
         669         {
         670                 sieve();
                | bl        0x93BC

            char flags[SIZE+1];

            int sieve()
         678 {
                | mov       r12,r13
                | stmdb     r13!,{r4,r11-r12,r14,pc}
-00002476 f     |   D:00000FD4 wr-long 00BC614E
-00002475 f     |   D:00000FD8 wr-long 00000FFC
-00002474 f     |   D:00000FDC wr-long 00000FE8
-00002473 f     |   D:00000FE0 wr-long 000093A0
-00002472 f     |   D:00000FE4 wr-long 000093CC
                | sub       r11,r12,#0x4

            char flags[SIZE+1];

            int sieve()                            /* sieve of erathostenes */
         678 {
                    register int i, primz, k;
```

B::A.V -2486. /FT

```
HIGH  HIGH  HIGH  HIGH  HIGH  LOW   LOW   LOW   HIGH  LO
baddress      faddress      fsymbol
  A:001C0C0C    R:000093BC  \\armle\arm\sieve
tpl tph ti.fore              trigger                    ti.ref
C8  FA                                    -0.000182800s
m.a  m.b  m.c  m.d
```

Example 2: Selective Trace on an Address Range defined through a Symbol



ETM Programming

Definition

	Address		Address
A-Range :		or	
B-Range :		or	
C-Range :		or	

	Value			Value
DATA0:		BYTE	COUNT0:	

Click here to open the Define Address Window

Define Address

all values HEX

Single Address — Mem.Class — Value/Symbol

Range — Mem.Class — Value/Symbol — Value/S

[✓ HLL-Range] — Symbol

Click here to select the radio button for HLL-Range

Press the V button

B::symbol.browse * /c "dialog.set SE ""*""" /d

```
\\*\*\*
symbol                  type                    addres
_real_default_signa…
remove
rename
_seterr
setvbuf
sieve                   (int ())                R:00
sieve
signal
_signal_init
_signal_real_handler
```

Type:

Select the symbol by a double click

Define Address

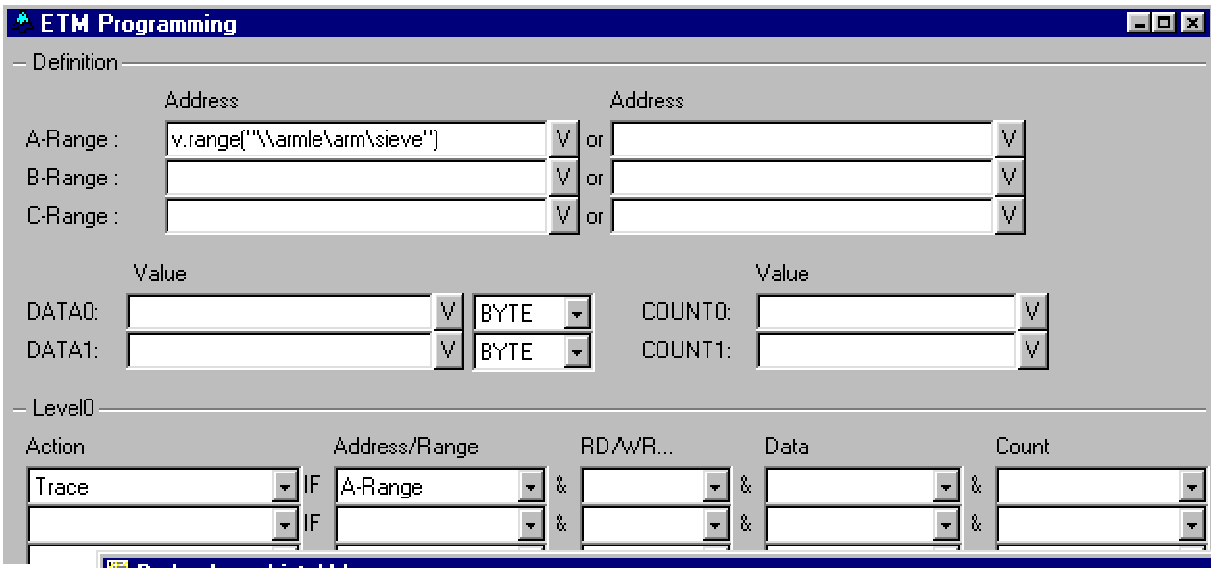
all values HEX

Single Address — Mem.Class — Value/Symbol

Range — Mem.Class — Value/Symbol — Value/Symbol

[✓ HLL-Range] — Symbol: \\armle\arm\sieve

CLEAR OK


ETM Programming

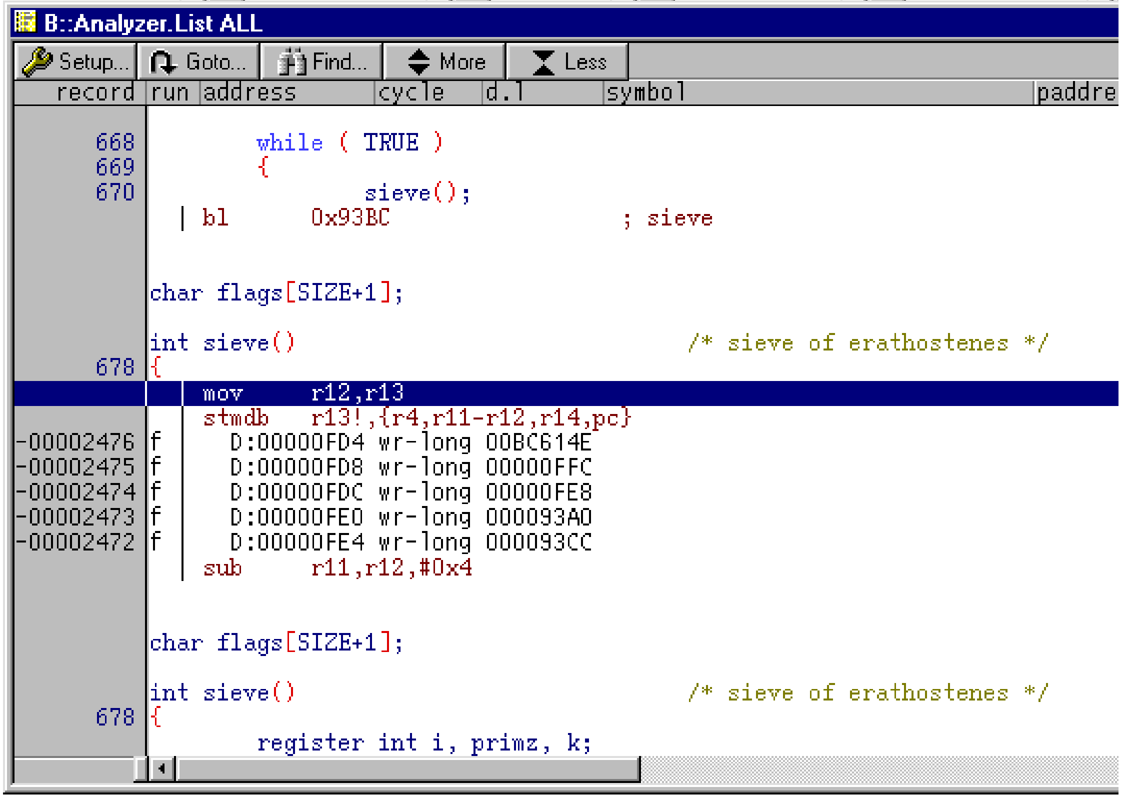
Definition

	Address		Address
A-Range :	v.range("\\armle\arm\sieve")	or	
B-Range :		or	
C-Range :		or	

	Value			Value
DATA0:		BYTE	COUNT0:	
DATA1:		BYTE	COUNT1:	

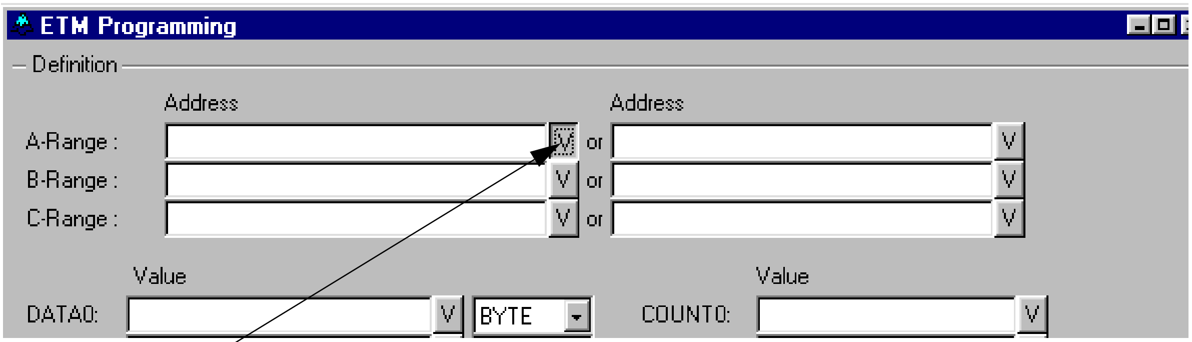
Level0

Action		Address/Range		RD/WR...		Data		Count
Trace	IF	A-Range	&		&		&	
	IF		&		&		&	

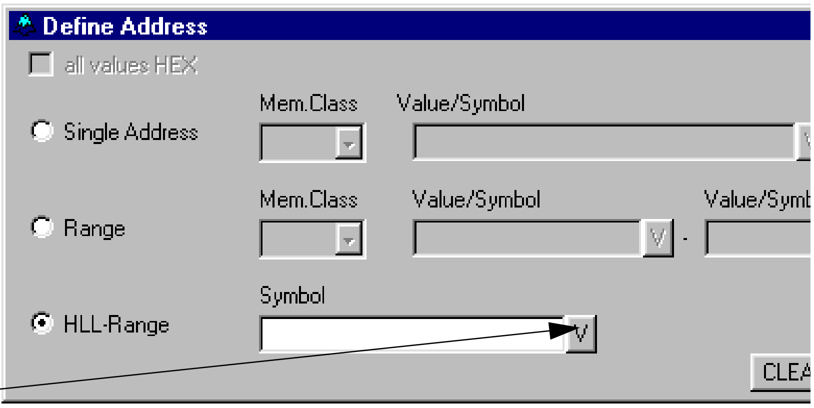

B::Analyzer.List ALL

```
    record run address        cycle  d.l        symbol                          paddre

       668           while ( TRUE )
       669           {
       670                   sieve();
               | bl      0x93BC                   ; sieve

          char flags[SIZE+1];

          int sieve()                                 /* sieve of erathostenes */
       678 {
               | mov     r12,r13
               | stmdb   r13!,{r4,r11-r12,r14,pc}
-00002476 f    |    D:00000FD4 wr-long 00BC614E
-00002475 f    |    D:00000FD8 wr-long 00000FFC
-00002474 f    |    D:00000FDC wr-long 00000FE8
-00002473 f    |    D:00000FE0 wr-long 000093A0
-00002472 f    |    D:00000FE4 wr-long 000093CC
               | sub     r11,r12,#0x4

          char flags[SIZE+1];

          int sieve()                                 /* sieve of erathostenes */
       678 {
                   register int i, primz, k;
```

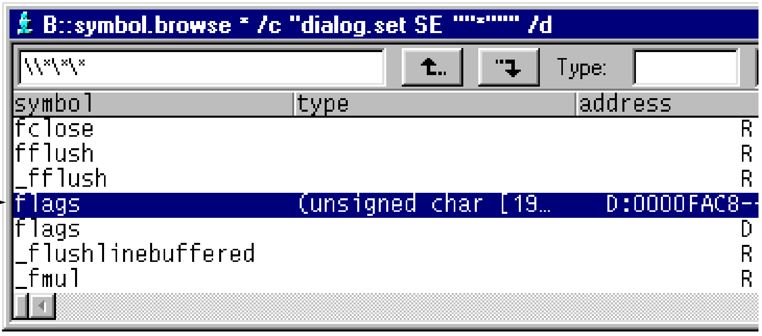
Example 3: Selective Trace on Access to a Symbol



Click here to open the Define Address Window

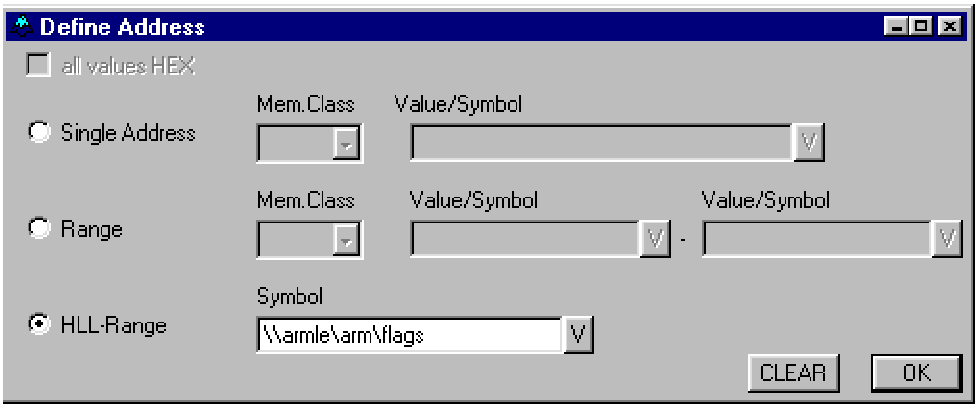


Click here to browse through the symbol data base

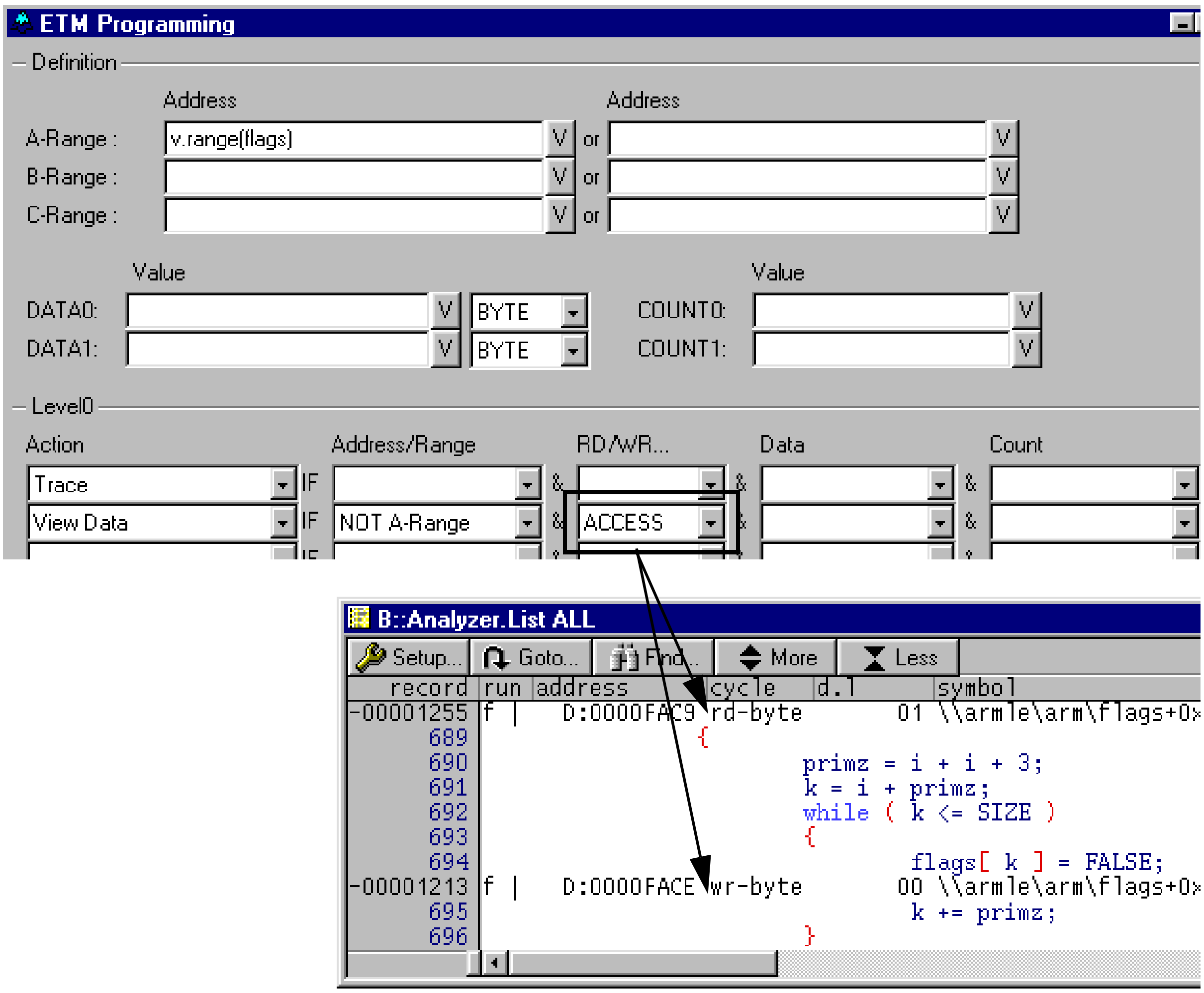


```
\\*\*\*
symbol              type                     address
fclose                                              R
fflush                                              R
_fflush                                             R
flags               (unsigned char [19…    D:0000FAC8-
flags                                               D
_flushlinebuffered                                  R
_fmul                                               R
```

Select the symbol by a double click



Symbol: \\armle\arm\flags



ETM Programming

Definition

	Address		Address
A-Range :	v.range(flags)	or	
B-Range :		or	
C-Range :		or	

	Value			Value
DATA0:		BYTE	COUNT0:	
DATA1:		BYTE	COUNT1:	

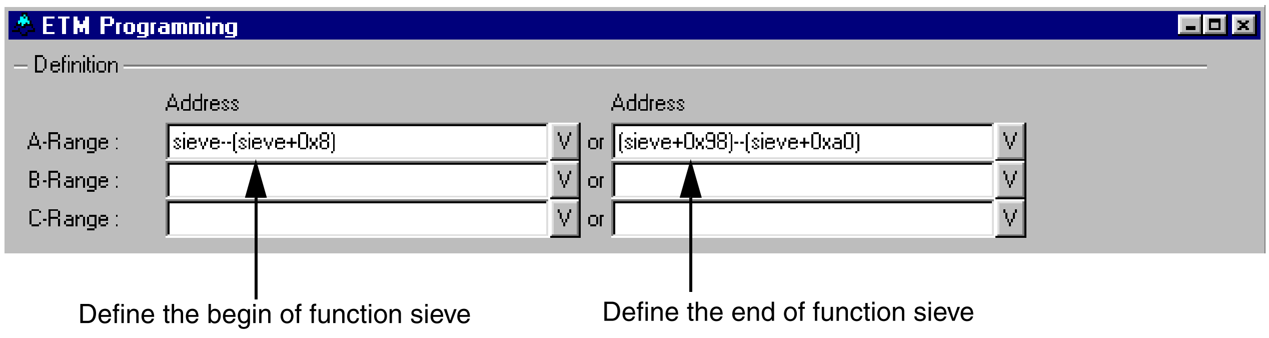
Level0

Action		Address/Range		RD/WR...		Data		Count
Trace	IF		&		&		&	
View Data	IF	NOT A-Range	&	ACCESS	&		&	

B::Analyzer.List ALL

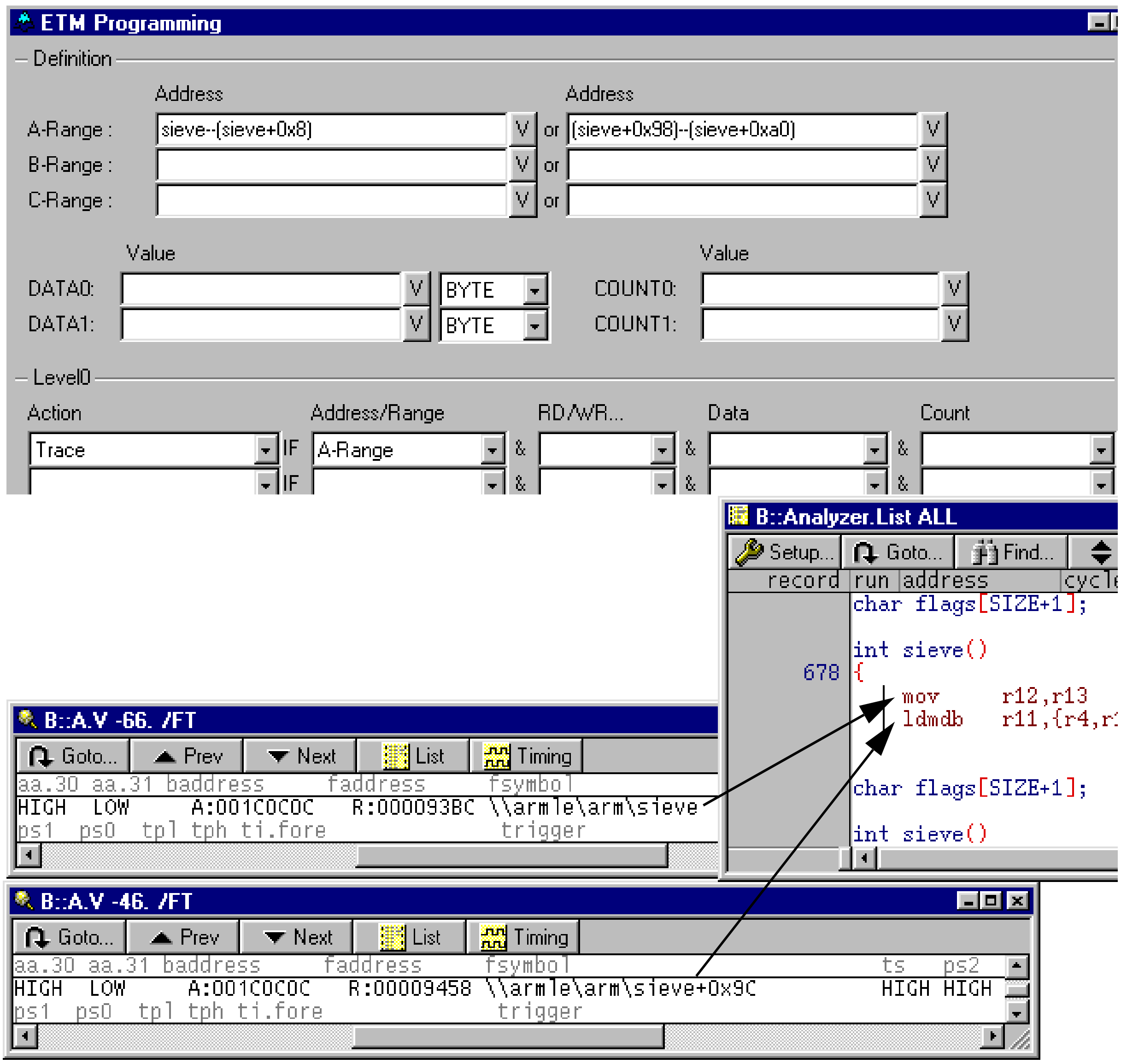
```
     record run address         cycle     d.l        symbol
-00001255 f |    D:0000FAC9 rd-byte        01 \\armle\arm\flags+0x
       689                  {
       690                                primz = i + i + 3;
       691                                k = i + primz;
       692                                while ( k <= SIZE )
       693                                {
       694                                        flags[ k ] = FALSE;
-00001213 f |    D:0000FACE wr-byte        00 \\armle\arm\flags+0x
       695                                        k += primz;
       696                                }
```

Example 4: Trace the Entrance and Exit of Function Sieve


ETM Programming
Define the begin of function sieve
Define the end of function sieve

Definition

	Address			Address	
A-Range :	sieve--(sieve+0x8)	V	or	(sieve+0x98)--(sieve+0xa0)	V
B-Range :		V	or		V
C-Range :		V	or		V



Definition

	Address			Address	
A-Range :	sieve--(sieve+0x8)	V	or	(sieve+0x98)--(sieve+0xa0)	V
B-Range :		V	or		V
C-Range :		V	or		V

	Value				Value	
DATA0:		V	BYTE	COUNT0:		V
DATA1:		V	BYTE	COUNT1:		V

Level0

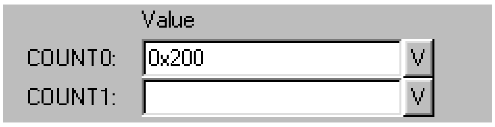
Action		Address/Range		RD/WR...		Data		Count
Trace	IF	A-Range	&		&		&	
	IF		&		&		&	

```
record run address      cycle
            char flags[SIZE+1];

            int sieve()
        678 {
              mov       r12,r13
              ldmdb     r11,{r4,r
            char flags[SIZE+1];

            int sieve()
```

```
aa.30  aa.31  baddress     faddress     fsymbol
HIGH   LOW    A:001C0C0C   R:000093BC  \\arm1e\arm\sieve
ps1   ps0   tpl  tph  ti.fore          trigger
```

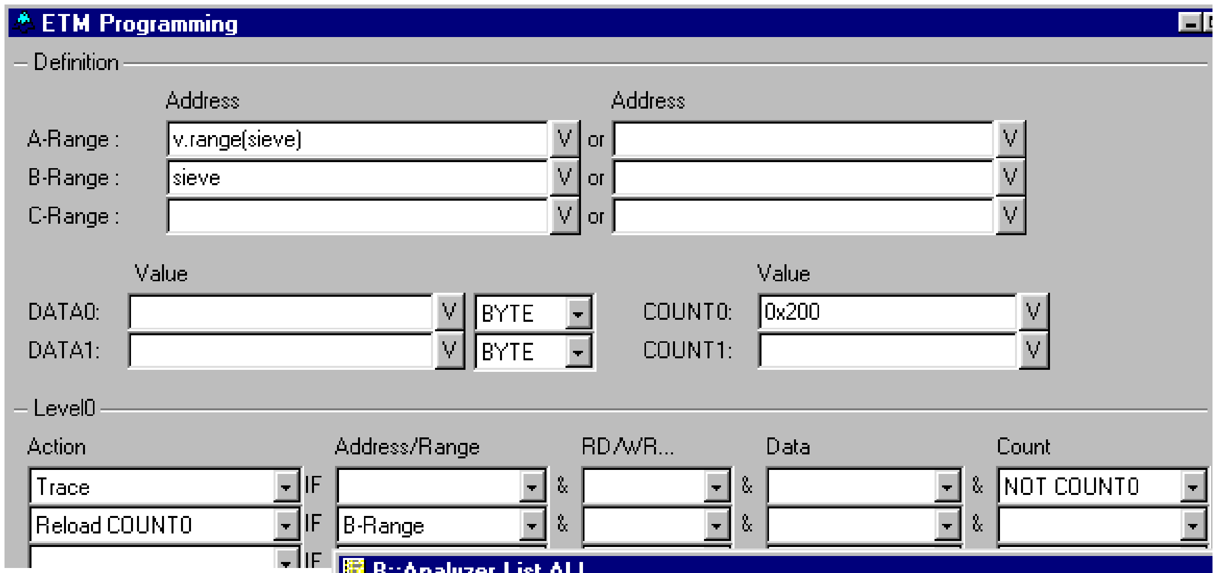
```
aa.30  aa.31  baddress     faddress     fsymbol                      ts    ps2
HIGH   LOW    A:001C0C0C   R:00009458  \\arm1e\arm\sieve+0x9C       HIGH  HIGH
ps1   ps0   tpl  tph  ti.fore          trigger
```

Example 5: Trace the first 200H Cycles in Function Sieve



	Value	
COUNT0:	0x200	V
COUNT1:		V

Define Counter Count0



ETM Programming

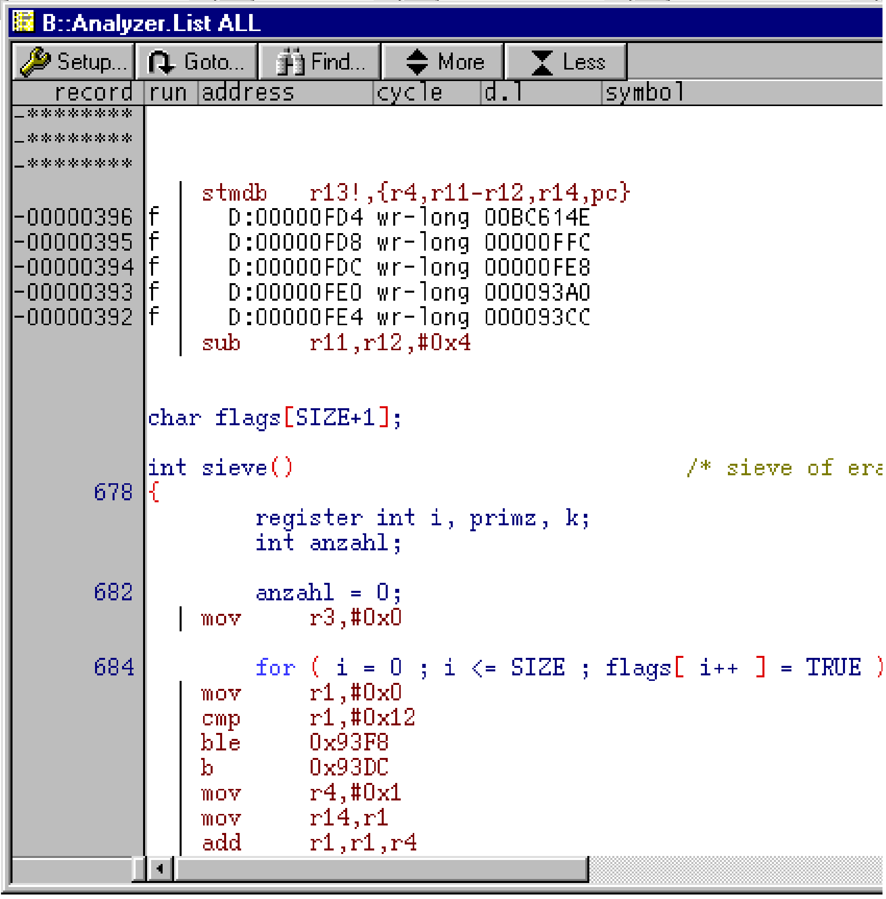
Definition

	Address			Address	
A-Range :	v.range(sieve)	V	or		V
B-Range :	sieve	V	or		V
C-Range :		V	or		V

	Value				Value	
DATA0:		V	BYTE	COUNT0:	0x200	V
DATA1:		V	BYTE	COUNT1:		V

Level0

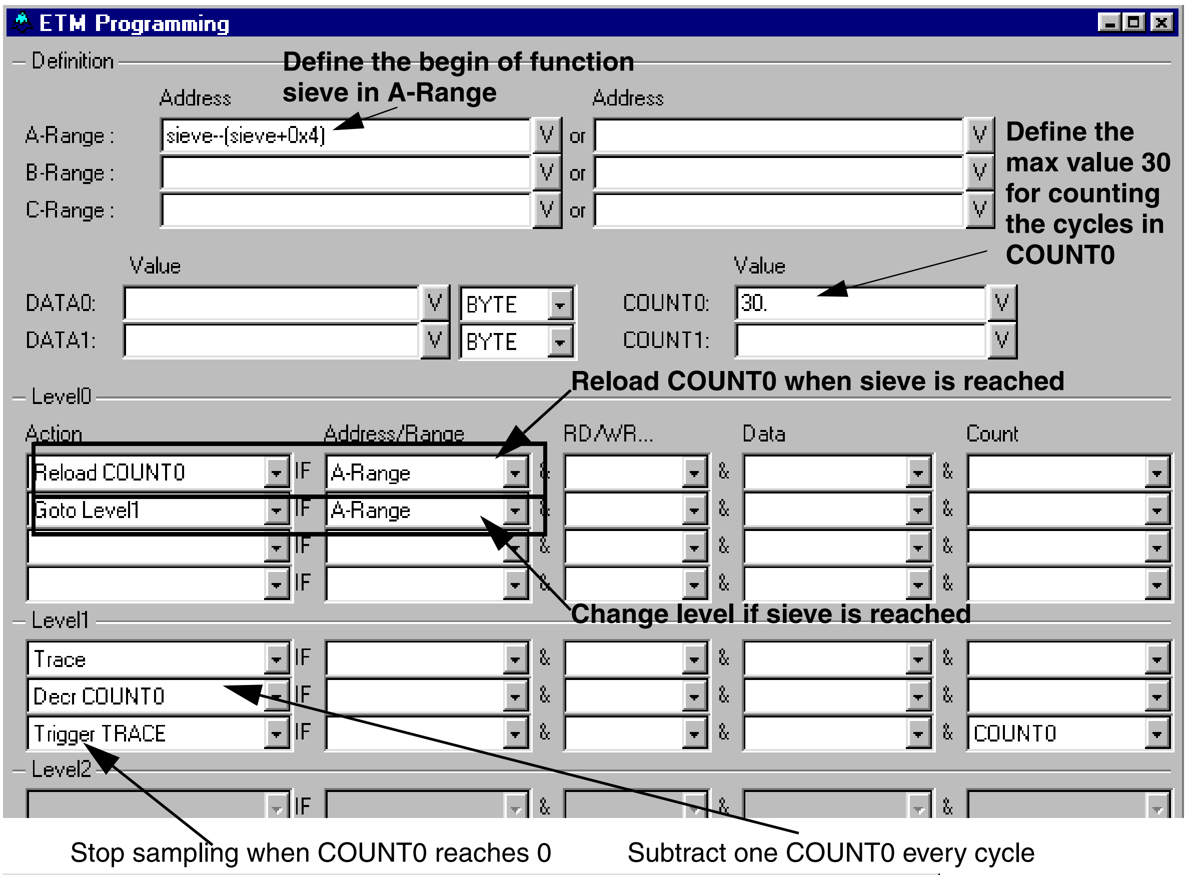
Action		Address/Range		RD/WR...		Data		Count
Trace	IF		&		&		&	NOT COUNT0
Reload COUNT0	IF	B-Range	&		&		&	
	IF							



B::Analyzer.List ALL

```
    record  run address              cycle    d.l         symbol
_*********
_*********
_*********
                |  stmdb   r13!,{r4,r11-r12,r14,pc}
-00000396  f    |    D:00000FD4 wr-long 00BC614E
-00000395  f    |    D:00000FD8 wr-long 00000FFC
-00000394  f    |    D:00000FDC wr-long 00000FE8
-00000393  f    |    D:00000FE0 wr-long 000093A0
-00000392  f    |    D:00000FE4 wr-long 000093CC
                |  sub     r11,r12,#0x4

           char flags[SIZE+1];

           int sieve()                                  /* sieve of era
      678  {
                   register int i, primz, k;
                   int anzahl;

      682          anzahl = 0;
              |  mov     r3,#0x0

      684          for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE )
              |  mov     r1,#0x0
              |  cmp     r1,#0x12
              |  ble     0x93F8
              |  b       0x93DC
              |  mov     r4,#0x1
              |  mov     r14,r1
              |  add     r1,r1,r4
```

Example 6: Trace all, when Function Sieve is reached goto Level 1 and stop Sampling and Debugging after 5 Cycles


ETM Programming
Define the begin of function sieve in A-Range
Define the max value 30 for counting the cycles in COUNT0
Reload COUNT0 when sieve is reached
Change level if sieve is reached
Stop sampling when COUNT0 reaches 0
Subtract one COUNT0 every cycle

Definition

	Address		Address
A-Range :	sieve--(sieve+0x4)	or	
B-Range :		or	
C-Range :		or	

	Value			Value
DATA0:		BYTE	COUNT0:	30.
DATA1:		BYTE	COUNT1:	

Level0

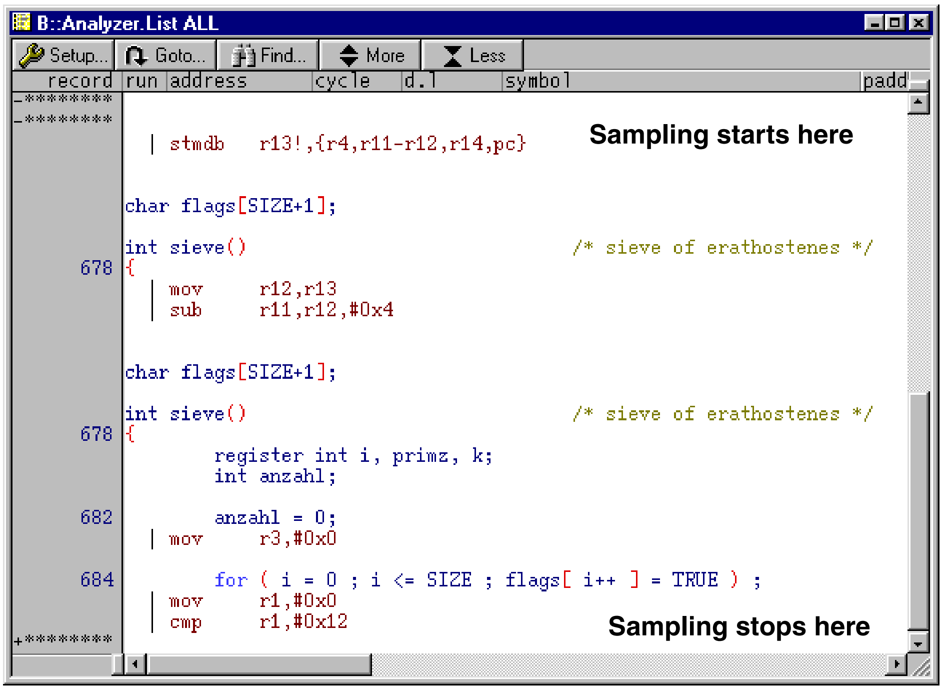
Action		Address/Range		RD/WR...		Data		Count
Reload COUNT0	IF	A-Range	&		&		&	
Goto Level1	IF	A-Range	&		&		&	
	IF		&		&		&	
	IF		&		&		&	

Level1

Action		Address/Range		RD/WR...		Data		Count
Trace	IF		&		&		&	
Decr COUNT0	IF		&		&		&	
Trigger TRACE	IF		&		&		&	COUNT0

Level2

Action		Address/Range		RD/WR...		Data		Count
	IF		&		&		&	



```
 record |run |address          |cycle |d.l         |symbol                  |padd
_*********
_*********

         | stmdb   r13!,{r4,r11-r12,r14,pc}


        char flags[SIZE+1];

        int sieve()                             /* sieve of erathostenes */
     678 {
         | mov     r12,r13
         | sub     r11,r12,#0x4


        char flags[SIZE+1];

        int sieve()                             /* sieve of erathostenes */
     678 {
                register int i, primz, k;
                int anzahl;

     682        anzahl = 0;
         | mov     r3,#0x0

     684        for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
         | mov     r1,#0x0
         | cmp     r1,#0x12
+*********
```